# COMPUTER GRAPHICS

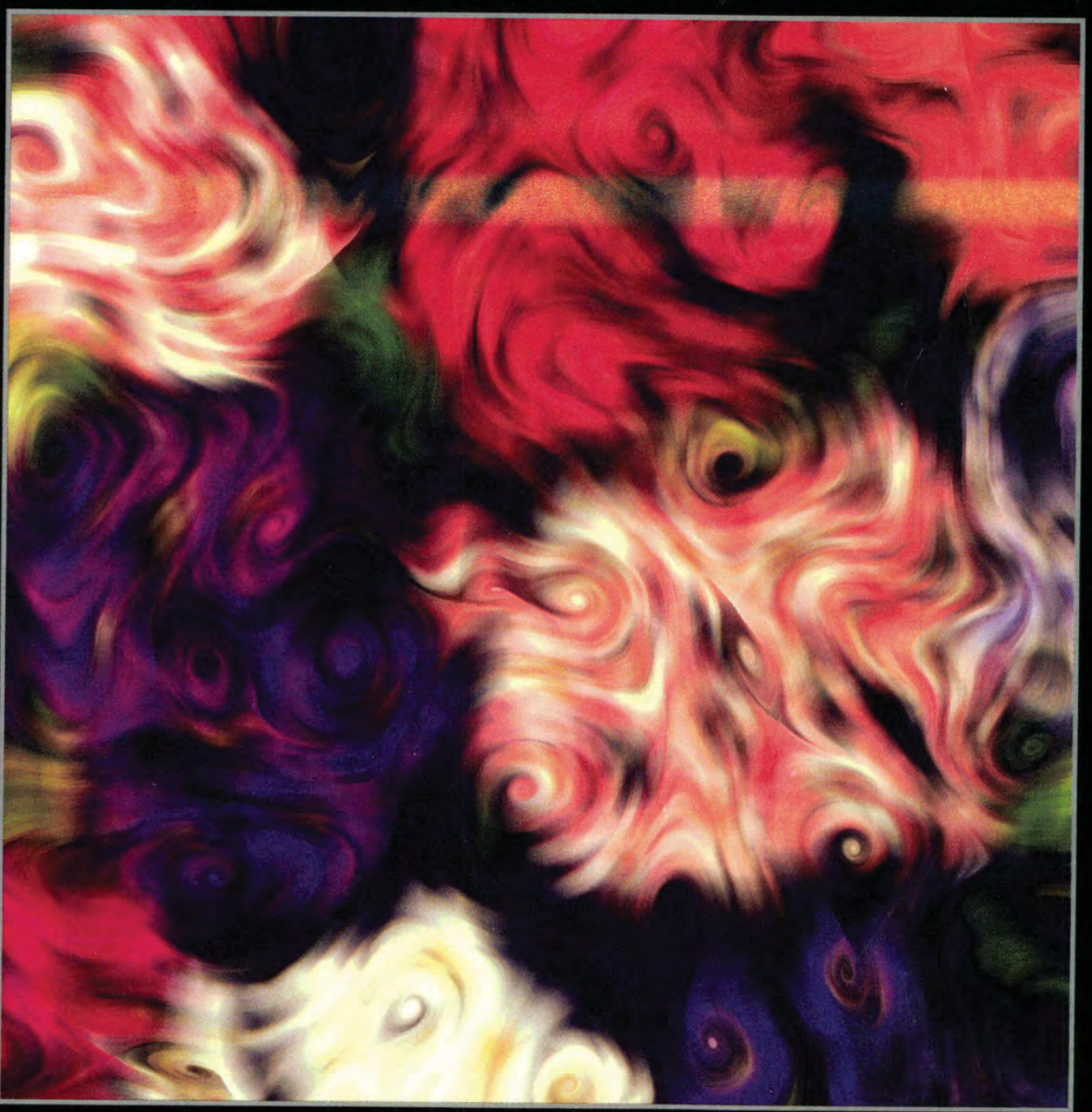
PROCEEDINGS

Annual Conference Series 1993

SIGGRAPH 93
Conference Proceedings
1–6 August 1993
Papers Chair James T. Kajiya
Panels Chair Donna Cox

A publication of ACM SIGGRAPH

*Sponsored by the Association for Computing Machinery's Special Interest Group on Computer Graphics*



---

Association for Computing Machinery
1515 Broadway, 17th Floor
New York, NY 10036
(212) 869-7440

ISSN No. 1069-529X

# COMPUTER GRAPHICS

PROCEEDINGS

Annual Conference Series 1993

SIGGRAPH 93
Conference Proceedings
August 1–6, 1993
Papers Chair James T. Kajiya
Panels Chair Donna Cox

A publication of ACM SIGGRAPH
Production Editor Steve Cunningham

*Sponsored by the Association for Computing Machinery's Special Interest Group on Computer Graphics*

**The Association for Computing Machinery, Inc.**
1515 Broadway, 17th Floor
New York, NY 10036



Sample Citation Information:
...Proceedings of SIGGRAPH 93 (Anaheim, California, August 1-6, 1993). In *Computer Graphics* Proceedings, Annual Conference Series, 1993, ACM SIGGRAPH, New York, 1993, pp. xx–yy.

## ORDERING INFORMATION

**Orders from nonmembers of ACM placed within the United States should be directed to:**

Addison-Wesley Publishing Company
Order Department
Jacob Way
Reading, MA 01867
Tel: 1-800-447-2226

Addison-Wesley will pay postage and handling on orders accompanied by check. Credit card orders may be placed by mail or by calling the Addison-Wesley Order Department at the number above. Follow-up inquiries should be directed to the Customer Service Department at the same number. Please include the Addison-Wesley ISBN number with your order:
A-W Softcover ISBN 0-201-58889-7
A-W CD-ROM ISBN 0-201-56997-3

**Orders from nonmembers of ACM placed from outside the United States should be addressed as noted below.**

**Europe/Middle East:**
Addison-Wesley Publishing Group
Concertgebouwplein 25
1071 LM Amsterdam
The Netherlands
Tel: +31 20 6717296
Fax: _31 20 6645334

**Germany/Austria/Switzerland:**
Addison-Wesley Verlag Deutschland GmbH
Wachsbleiche 7-12
W-5300 Bonn 1
Germany
Tel: +49 228 98 515 0
Fax: +49 228 98 515 99

**United Kingdom/Africa:**
Addison-Wesley Publishers Ltd.
Finchampstead Road
Wokingham, Berkshire RG11 2NZ
United Kingdom
Tel: +44 734 794000
Fax: +44 734 794035

**Asia:**
Addison-Wesley Singapore Pte. Ltd.
15 Beach Road
#05-02/09/10 Beach Centre
Singapore 0718
Tel: +65 339 7503
Fax: +65 339 9709

**Japan:**
Addison-Wesley Publishers Japan Ltd.
Nichibo Building
1-2-2 Sarugakucho
Chiyoda-ku, Tokyo 101
Japan
Tel: +81 33 2914581
Fax: +81 33 2914592

**Australia/New Zealand:**
Addison-Wesley Publishers Pty. Ltd.
6 Byfield Street
North Ryde, N.S.W. 2113
Australia
Tel: +61 2 878 5411
Fax: +61 2 878 5830

**Latin America:**
Addison Wesley Iberoamericana S.A.
Boulevard de las Cataratas #3
Colonia Jardines del Pedregal
Delegacion Alvaro Obregon
01900 Mexico D.F.
Tel: +52 5 660 2695
Fax: +52 5 660 4930

**Canada:**
Addison-Wesley Publishing (Canada) Ltd.
26 Prince Andrew Place
Don Mills, Ontario M3C 2T8 Canada
Tel: 416-447-5101
Fax: 416-443-0948

**Orders from ACM Members:**

A limited number of copies are available at the ACM member discount. Send order with payment in U.S. dollars to:

ACM Order Department
P.O. Box 64145
Baltimore, MD 21264

OR, for informatio on accepted European currencies and exchange rates, contact:

ACM European Service Center
Avenue Marcel Thiry 204
1200 Brussells
Belgium
Tel: +32 2 774 9602
Fax: +32 2 774 9690
Email: acm_europe@acm.org

ACM will pay postage and handling on orders accompanied by check.

Credit card orders only: 1-800-342-6626
Credit card orders may also be placed by mail.

Customer service, or credit card orders from Alaska, Maryland, and outside the U.S.:
+1 410 528 4261

Single-copy orders placed by fax:
+1 410 528 8596

Electronic mail inquiries may be directed to acmpubs@acm.org.

**Please include your ACM member number and the ACM order number with your order.**

ACM Order Number: 428930

ACM ISBN: 0-89791-601-8

ISSN: 1069-529X

# Contents

## Papers Sessions, Wednesday, 4 August 1993

**8:30–10:00** **Interaction**
*Chair: Jock Mackinlay*



**1:30–3:15** **Rendering Architectures**
*Chair: Forest Baskett*



**3:30–5:00** **Virtual Reality**
*Chair: Andries van Dam*

## Papers Sessions, Thursday, 5 August 1993

**8:30–10:00** **Global Illumination**
Chair: Francois Sillion



**10:15–12:00** **Light and Color**
Chair: Ken Torrance



**1:30–3:15** **Numerical Methods for Radiosity**
Chair: Paul Heckbert



**3:30–5:00** **Visibility**
*Chair: Frank Crow*

## Paper Sessions, Friday, 6 August 1993

**8:30–10:00** **Visualization**
*Chair: Mike Keeler*



**10:15–12:00** **Processing Synthetic Images**
*Chair: Don Mitchell*



**1:30–3:15** **Techniques for Animation**
*Chair: Andrew Glassner*



**3:30–5:00** **Natural Phenomena**
*Chair: Darwyn Peachey*

## Panel Sessions, Tuesday, 3 August 1993



## Panel Sessions, Wednesday, 4 August 1993

## Panel Sessions, Thursday, 5 August 1993



## Panel Sessions, Friday, 6 August 1993

# Preface

You hold in your hands a distillation of the work of hundreds of people representing over a hundred thousand hours of collective brain work: the technical program of the 20th annual SIGGRAPH conference held in Anaheim, California.

Each year the technical program is modified in many small and hopefully better ways. This year you will notice that there are more papers than SIGGRAPH has accepted in many years, that we have expanded the number of sessions, and that the number of days during which papers are presented has grown.

But some things we've modified do not show up in the papers themselves. This year the composition of the selection committee is considerably different than in previous years. The SIGGRAPH conference planning committee mandated "term limits" for members of the selection committee. This year, no one was a senior reviewer if they served on the committee for the previous two years.

Also new this year is the establishment of reviewer ethics guidelines which sought to achieve a uniform level of protection for the information contained within SIGGRAPH submissions. The prospective author's kit also contained a look into how papers were processed, judged, and accepted or rejected. This information was intended to give people an insight into the paper review and selection mechanism. Since so much of this process deals with specific papers and people's opinions of the significance of someone's ideas, the record and discussion that occurs during this process is of necessity secret. However, everyone should know what happens in general.

We received 225 submissions this year, a new record, and accepted 46 papers, the most since 1978. Andrew Glassner and I read and discussed every submission and—within the constraints of load balancing—attempted to assign each submission with the best senior reviewer for that submission. The review process and the selection meeting were very much as in previous years. Everyone on the committee strove to include quality papers over as wide a range of topics as was feasible. The individual merits of papers were extensively discussed and judged by those members of the committee allowed to attend. As in previous years, those who had a connection with the institutions or authors represented in a particular paper were asked to leave the discussion. We tried to be as fair and objective as could be possible.

I wish to congratulate the committee on their display of wisdom and insight during the selection meeting. The discussion that occurred in March impressed me with its high professional level and sensitive consideration given to every possible conflict of interest.

Of course, as many well know, the SIGGRAPH review process is far from perfect: I may have sent a submission to the wrong person, reviewers may misunderstand the ideas in a paper, or some critical piece of information may not have reached the author. If you had a paper rejected unfairly by SIGGRAPH 93, I apologize for our mistakes. If you have ideas on how we may improve future cycles of reviewing, SIGGRAPH is eager to hear them. I urge you please to contact me or Andrew Glassner, the program chair for SIGGRAPH 94.

Even though we accepted more papers than ever before, the publication budget for this proceedings was fixed by the severe financial constraints that SIGGRAPH has been forced to adopt. We have thus had to be very careful on issues that impact the ultimate cost of this proceedings. Most of the authors of the papers in this document have struggled valiantly to accomplish the difficult task of meeting the hard page limits given to them. The committee considered the content of each paper and carefully set length and color restrictions. Steve Cunningham and I were given the unhappy task of enforcing these restrictions and denying many authors' desperate pleas for more space.

Those who know me personally know that I am, to put it delicately, organizationally challenged. Without the crucial support and help of of a number of people, SIGGRAPH 93 would probably not have had a technical program this year. These people have my deep thanks and gratitude: Debbie Buuck, Steve Cunningham, Mary Kate Haley, Kevin Luster, and Pey Jen Wu. I also wish to thank the SIGGRAPH 93 cochairs, Bob Judd and Mark Resch, for establishing an exciting and creative atmosphere that allowed us to take part in shaping the conference, its content, and its future.

James T. Kajiya
SIGGRAPH 93 Papers Chair

## 1993 ACM SIGGRAPH Awards

# Steven A. Coons Award

### for Outstanding Creative Contributions to Computer Graphics

# Ed Catmull

This year ACM SIGGRAPH has selected Dr. Edwin E. Catmull to receive the Steven A. Coons Award for Outstanding Creative Contributions to Computer Graphics. Over the past twenty years, Ed Catmull has made many and noteworthy advances in computer graphics as an individual researcher, as an inspiring leader in the field, as a director of organizations, and as a mentor for many.

Ed has made important direct contributions to the field of computer graphics. With his doctoral dissertation at the University of Utah, he introduced the notion of subdivision to pixel level as a display method, added a fast adaptive subdivision method for bi-cubic surface patches, and provided the first published description of the ubiquitous z-buffer visibility algorithm. He also developed the Catmull-Rom interpolating spline and an early system for generating animated articulated figures. At the New York Institute of Technology, he wrote the first spline inbetweening animation program. At Lucasfilm, with Alvy Ray Smith, he invented a two-pass image warping algorithm.

In addition to his own research contributions, Ed has founded and led three important and influential centers of computer graphics research and development: the Computer Graphics Laboratory at New York Institute of Technology (NYIT), the Lucasfilm Computer Division, and Pixar. In each of these organizations, he attracted and developed some of the best talent in the computer graphics business. These organizations rose quickly to become leading centers of research in our field. The common ingredient in these three organizations is Catmull and the talented people he attracts and develops; wherever Catmull goes, exciting things seem to happen.

Engineers at NYIT developed the first RGB painting program, were pioneers in the use of computer-controlled video equipment, invented mip-maps, and wrote the Tween and Bop animation programs. People working for Ed at Lucasfilm/Pixar made many contributions to image rendering, including particle systems, the first shading language, distributed ray tracing, stochastic sampling, and the Reyes/RenderMan software. They also developed volume rendering software, digital compositing, the Computer Animation Production System (CAPS) developed with Walt Disney Pictures, the Pixar Image Computer, laser input/output scanning, and video and audio editing systems. The group produced a number of special effects such as the "Genesis" effect in "Star Trek II: The Wrath of Khan" and the stained glass man in "The Young Sherlock Holmes," short animated films as exemplified by "Andre and Wally B.," "Red's Dream," "Luxo Jr.," and "Tin Toy," and numerous commercials.

Four of SIGGRAPH's first five Achievement Award winners (and six of eleven overall) have worked for Ed at one time or another. "Luxo Jr." was one of the earliest computer animated films to be nominated for an Academy Award and "Tin Toy" was the first to win one. The Academy of Motion Picture Arts and Sciences last year awarded a Scientific and Technical Academy Award for the development of CAPS to Disney employees and Pixar employees who reported to Catmull. This year the Academy gave a Scientific and Technical Academy Award for the RenderMan software to Catmull and his collaborators[1].

Ed Catmull earned the BS in Physics and the BS in Computer Science (1969) and then the Ph.D. in Computer Science (1974), all from the University of Utah. We note that his doctoral dissertation committee included Steve Coons and Ivan Sutherland, the first recipient of the Coons Award. As noted above, his career spans three positions as Director of the Computer Graphics Laboratory at the New York Institute of Technology (1974-79), Vice President and Managing Director of the Computer Division of Lucasfilm, Ltd. (1979-1986), and now as President of Pixar.

It is impossible to know how many of us have aimed higher and worked harder because Ed encouraged us by collaboration or by being an important figure in the field. It is impossible to know how many of us have taken our research a little further out on the fringe because we thought it was something that Ed might do. His influence at the person-to-person level is magical, and though difficult to describe in words, it continues to affect the practice of computer graphics in subtle and important ways.

**References**

Catmull, Edwin E., "A System for Computer Generated Movies," Proceedings of ACM Annual Conference, August 1972.

Catmull, Edwin E., "Computer Display of Curved Surfaces," Proceedings of the IEEE Conference on Computer Graphics, Pattern Recognition and Data Structures, May 1975.

Catmull, Edwin E. and Raphael Rom, "A Class of Local Interpolating Splines," *Computer Aided Design*, Academic Press, 1974.

Catmull, Edwin E., "The Use of the Computer in Animation Production," *Digital Video*, Vol.. 2, 1979.

Catmull, Edwin E., "Computer Aided Animation: A system in Full Production," *American Cinematographer*, October 1979.

Catmull, Edwin E., "The Problems of Computer Assisted Animation," SIGGRAPH Conference Proceedings, 1978.

Catmull, Edwin E., "A Hidden-Surface Algorithm with Anti-Aliasing," SIGGRAPH Conference Proceedings, 1978.

Catmull, Edwin E. and James Clark, "Recursively Generated B-spline Surfaces on Arbitrary Topological Meshes," *Computer Aided Design*, November 1978.

Catmull, Edwin E., "A Tutorial on Compensating Tables," SIGGRAPH Conference Proceedings, 1979.

Catmull, Edwin E. and Alvy Ray Smith, "3-D Transformations of Images in Scanline Order," SIGGRAPH Conference Proceedings, 1980.

Catmull, Edwin E., "An Analytic Visible Surface Algorithm for Independent Pixel Processing," SIGGRAPH Conference Proceedings, 1984.

[1]The colleagues from Pixar sharing the Scientific and Technical Academy Award are Patrick Hanrahan, Rob Cook, Loren Carpenter, Tony Apodaca, Darwyn Peachey, and Tom Porter. (This list includes three SIGGRAPH Achievement Award recipients!)

**Previous award winners**

| | |
|---|---|
| 1991 | Andries van Dam |
| 1989 | David C. Evans |
| 1987 | Donald P. Greenberg |
| 1985 | Pierre Bézier |
| 1983 | Ivan E. Sutherland |

## 1993 ACM SIGGRAPH Awards

# Computer Graphics Achievement Award

# Pat Hanrahan

The SIGGRAPH Computer Graphics Achievement Award is presented to Dr. Patrick M. Hanrahan for his contributions to rendering systems and algorithms. We recognize his research and publications on volume rendering, ray tracing, and radiosity algorithms as well as his role as the architect of the RenderMan™ interface.

Hanrahan started his career in physics and biology at the University of Wisconsin. He received a BS. Degree in Nuclear Engineering, graduating first in class of 1977. While at Wisconsin he worked in the Department of Zoology developing computer models of the motornervous system of the nematode *Ascaris*. During this period he became interested in models of shape and the potential of the computer for visualizing the results of simulations. He quickly recognized the importance of modeling and did some work on creating models from edge-vertex graphs (SIGGRAPH '82).

Realizing that he needed to know more, he inquired by letter about a summer position at the New York Institute of Technology (NYIT) Computer Graphics Laboratory. His letter was persuasive enough to land him a spot for the summer that quickly turned into a full-time staff position. There he was initially responsible for modeling and animation software and eventually was the Director of the 3D Animation Systems Group. While at NYIT, Hanrahan published papers on ray tracing algebraic surfaces ('83) and 'beam tracing' polygonal surfaces ('84).

Hanrahan returned to the University of Wisconsin to finish his dissertation, which ended up having much more to do with graphics than biology, and received the Ph.D. in Biophysics in 1985. After a short stint at Digital's Systems Research Lab in Palo Alto, he accepted a position at Pixar in 1986 shortly after Pixar separated from Lucasfilm. He collaborated with Bob Drebin and Loren Carpenter in developing the first volume rendering algorithms for the Pixar image computer ('88). These algorithms were quite different from earlier approaches in that they created images directly from three-dimensional arrays without the intermediate steps of converting to standard surface representations such as polygons. Volume rendering is now a major component of scientific and medical visualization systems.

He later joined the REYES machine group and was responsible for the rendering software and the graphics architecture. The rendering interface of the system evolved into the RenderMan standard that now is widely used in the movie industry[1]. In particular Hanrahan was the principal architect of the RenderMan Interface (Pixar '88). His paper with Jim Lawson describes one of the more interesting aspects of the system, the shading language, which allows users to extend the capabilities of the re ndering system by defining new procedurally defined appearances.

Since 1989 Hanrahan has been on the Faculty of the Computer Science Department at Princeton University, where he became tenured as Associate Professor in 1991. His goal since returning to academia is to put computer graphics on a sound mathematical and scientific foundation. Placed in an environment where publication is more than encouraged, Hanrahan's publications have blossomed. He has been arguably the most prolific single contributor to SIGGRAPH in the last few years. His name appears on no fewer than five papers in this year's proceedings. He is also extremely interested in computer graphics education and has won three university teaching awards since joining Princeton.

Recently Hanrahan made important contributions to accelerating radiosity computations through hierarchical methods (91a, 93c). He has continued his work in volume rendering (91b, 93b). He has contributed pioneering work for rendering caustics (92), for investigating wavelets for radiosity (93d), and determining global visibility (93e). He has discovered a fundamental closed form result for the radiosity form factor between two polygons (93a). In addition to these efforts he has also found time to contribute to texturing through direct manipulation (90a), to develop a shading language (90b), and to keep up with rendering architectures for parallel machines.

This extraordinarily high level of productivity is due in part to Hanrahan's ability to find and to cooperate with a wide variety of collaborators as well as his own creativity. In addition to the students at Princeton, he has managed to work with colleagues around the world in both academia and industry. Furthermore, the work just described has an element of scholarship to it that has often been elusive in computer graphics. The fast-moving nature of the field often makes work more than a few years old seem out of date. Hanrahan has been one of those who goes back to the basics both inside computer graphics and in the many fields that can contribute.

Hanrahan's work has a significant ongoing effect on computer graphics in a wide variety of rendering applications, in the high quality of his scholarship, and especially in the force of his ideas. His influence on computer graphics is still accelerating, leaving us eagerly anticipating his future achievements as well as honoring those of the past.

**References**

All references except one are to SIGGRAPH conference proceedings in the year indicated.

[82] "Creating Volume Models from Edge-Vertex Graphs," pp. 77-84.

[83] "Ray Tracing Algebraic Surfaces," PP 83-90.

[84] with P. Heckbert, "Beam Tracing Polygonal Objects," pp. 119-127.

[Pixar 88] The RenderMan Interface.

[88] with R. Drebin and L. Carpenter, "Volume Rendering," pp. 65-74.

[90a] with P. Haeberli, "Direct WYSIWYG Painting and Texturing on 3D Shapes," pp. 215-224.

[90b] with J. Lawson, "A Language for Shading and Lighting," pp. 289-298.

[91a] with D. Salzman and L. Aupperle, "A Rapid Hierarchical Radiosity Algorithm," pp. 197-206.

[91b] with D. Laur, "Hierarchical Splatting: A Progressive Refinement Algorithm for Volume Rendering," pp. 285-288.

[92] with D. Mitchell, "Illumination from Curved Reflectors," pp. 283-292.

[93a] with P. Schroeder, "On the Form Factor between Two Polygons."

[93b] with W. Krueger, "Reflection from Layered Surfaces due to Subsurface Scattering."

[93c] with L. Aupperle, "A Hierarchical Illumination Algorithm for Surfaces with Glossy Reflection."

[93d] with S. Gortler, P. Schroeder, and M. Cohen, "Wavelet Radiosity."

[93e] with S. Teller, "Global Visibility Algorithms for Illumination Computations."

[1]Earlier this year Hanrahan and his colleagues from Pixar, Ed Catmull, Rob Cook, Loren Carpenter, Tony Apodaca, Darwyn Peachey, and Tom Porter, were awarded a Scientific and Technical Academy Award by the Academy of Motion Picture Arts and Sciences for the development of software that produces images used in motion pictures from 3D computer descriptions of shape and appearance.

**Previous award winners**

| | |
|---|---|
| 1992 | Henry Fuchs |
| 1991 | James T. Kajiya |
| 1990 | Richard Shoup and Alvy Ray Smith |
| 1989 | John Warnock |
| 1988 | Alan H. Barr |
| 1987 | Robert Cook |
| 1986 | Turner Whitted |
| 1985 | Loren Carpenter |
| 1984 | James H. Clark |
| 1983 | James F. Blinn |

# 2–D Shape Blending: An Intrinsic Solution to the Vertex Path Problem

Thomas W. Sederberg[1], Peisheng Gao[1], Guojin Wang[2], and Hong Mu[1]

## Abstract

This paper presents an algorithm for determining the paths along which corresponding vertices travel in a 2–D shape blending. Rather than considering the vertex paths explicitly, the algorithm defines the intermediate *shapes* by interpolating the intrinsic definitions of the initial and final shapes. The algorithm produces shape blends which generally are more satisfactory than those produced using linear or cubic curve paths. Particularly, the algorithm can avoid the shrinkage that normally occurs when rotating rigid bodies are linearly blended, and avoids kinks in the blend when there were none in the key polygons.

Categories and Subject Descriptors: I.3.3 [Computer Graphics]: Picture/Image Generation; I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling.

General Terms: Algorithms

Additional Key Words and Phrases: Shape blending, character animation, numerical algorithms.

## 1 Introduction

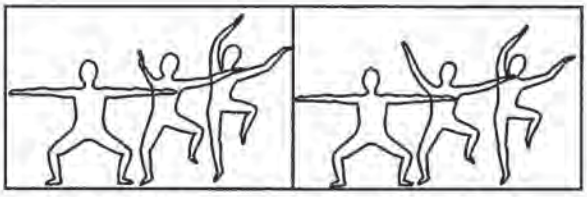
This paper deals with shape blending of 2–D polygons. As illustrated in Figures 1 and 2, a shape blend algorithm determines the in-between polygons which provide a smooth transformation between two given 2–D polygons, referred to as the *key polygons*.

Shape blending requires the solution of two main subproblems: the vertex correspondence problem (that is, determining which vertex on one key polygon will travel to which vertex on the other key polygon), and the vertex path problem (that is, determining along what path each vertex will travel).

For 2–D polygonal shapes, a solution to the vertex correspondence problem is presented in [12]. Shape blending of 2–D Bézier curve shapes is addressed in [11]. Various solutions to the shape interpolation of 3–D polyhedra are presented in [3, 5, 7, 8].

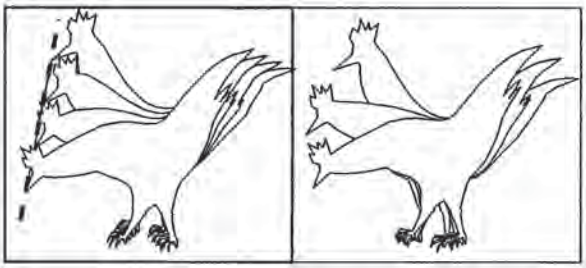
This paper addresses the vertex path problem and is motivated by two figures from [12]. Figure 1.a provides an example of a shape blend in which the middle shapes is derived from its neighboring key polygons. This is basically a good shape blend, except that the dancer's arm in the middle frame is only half as long as it is in the key frames.

The shape blend in Figure 2.a looks fine except that the chicken's neck gets shorter. These shortenings occur because of the linear path followed by vertices during the shape blend, as shown by the path travelled by the chicken's beak.



a. Withered Arm b. Normal Arm

Figure 1: Dancer



a. Compressed Neck b. More Natural Neck

Figure 2: Chicken

These problems seriously weaken the practical use of 2–D shape blending using linear vertex motion, for applications such as character animation. The contribution of this paper is an improved method for computing in-between frames, once the correspondence between key polygons has been determined as in [12]. Sample results of the new vertex path algorithm are shown in Figures 1.b and 2.b.

A referee of paper [12] remarked: "I am unhappy with the phrase, 'physically based,' in this context. The 'physics' here has nothing to do with the physics of chickens, . . . , or any of the other nominal subjects of interpolation." That observation formulates precisely the problem we confront in trying to infer the correct motion between two changing shapes. While [12] demonstrates that an algorithm which knows nothing about the "physics of a chicken" is able to correlate the prominent features of two chicken outlines, the accurate computation of *motion* as a chicken lowers his head really calls for a model of the chicken's skeleton, musculature, etc.. What we seek is a tool that might assist a traditional animator to create convincing computer-assisted in-betweens, when the only information available is contained in the two key frames. The solution presented here is a heuristic whose justification lies



[1]Engineering Computer Graphics Laboratory
368 Clyde Building
Brigham Young University
Provo, UT 84602
tom@byu.edu
[2]Zhejiang University, China

 ACM-0-89791-601-8/93/008/0015 $01.50

in the fact that it generally seems to work rather well.

### 1.1 Proposed solution

A polygon definition which lists the Cartesian coordinates of its vertices might be called an *explicit* description. An alternate means of defining a polygon is in terms of the lengths of its edges and the angles at its vertices. Such a polygon description forms the basis of an approach to geometry, popular in elementary education, known as *Turtle Graphics* [2] wherein a polygon is defined by instructions such as: walk 10 paces to the east, turn 45° to the left and proceed 6 paces, turn 30° to the right and go 5 more paces, ....

This paper postulates that the heuristic of blending intrinsic definitions (edge lengths and vertex angles) of two key polygons will generally produce a more satisfactory in-between motion than will linear vertex paths. Evidence that this is so is provided in the figures.

### 1.2 Related work

One alternative to linear vertex paths is to define vertex paths of higher degree. For 3–D polyhedral shape transformations, [8] proposes using an Hermite cubic path with end tangents set equal to the vertex normals. While this idea evidently is effective for the transformations between highly dissimilar shapes addressed in [8], it would not generally work too well for character animation since motion does not uniformly occur normal to a curve outline.

[14] develops an approach to character animation using quadratic Bézier vertex paths. By default, vertices travel along a parabolic arc such that the distance from each vertex to the center of mass of all vertices changes monotonically. Also, it allows the user to signify a pivot point for appendages. The current algorithm works with less user interaction.

In other approaches to shape blending, such as Minkowski sums [7], the vertex path and vertex correspondence problems are coupled and solved simultaneously. Minkowski sums, however, blur even gross details such as arms and legs when blending non-convex objects, and hence are not suitable for character animation. Shape blends that operate on an *implicit* definition of the curve or surface, $f(x,y) = 0$ or $f(x,y,z) = 0$, [6] likewise don't currently support the detail required for character animation.

Of course, the substantial literature on physically based modelling and synthetic actors is also highly relevant, though such methods rely on more information than is available to us.

Ideas for modeling with intrinsically defined curves are proposed in [1], and [13] looks at curve and surface kinematics based on differential equations.

## 2 Intrinsic shape interpolation

Denote the vertices of the two key polygons by $\mathbf{P}_{A_i}, \mathbf{P}_{B_i}, (i = 0, 1, \ldots, n-1)$. We assume that both key polygons have the same number of edges, as will be the case after vertex correspondence is established [12]. In this discussion, we use the convention that counter-clockwise angles are positive. For convenience, we adopt the notation $m = n - 1$ where $n$ is the number of polygon edges.

Our goal is to compute the vertices $\mathbf{P}_i (i = 1, 2, \ldots, m)$ for the polygon which is "$t$" of the way between $\mathbf{P}_A$ and $\mathbf{P}_B$, $0 \le t \le 1$. $\mathbf{P}_0$ will be taken as the *anchor point*, and its position determines the rigid body translation of the shape. This, along with the directed angles $\alpha_{A_0}$ and $\alpha_{B_0}$ formed by the $x$-axis and the vectors $\mathbf{P}_{A_0}\mathbf{P}_{A_1}$ and $\mathbf{P}_{B_0}\mathbf{P}_{B_1}$, is discussed further in section 3.

Begin by obtaining the intrinsic definitions of $\mathbf{P}_A$ and $\mathbf{P}_B$ by computing the polygon angles and edge lengths shown in Figure 3:

$$\theta_{A_i}, \theta_{B_i}, (i = 1, 2, \ldots, m). \quad (1)$$

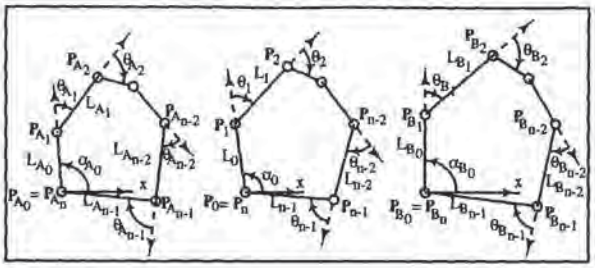


Figure 3: Intrinsic variables

$$L_{A_i} = |\mathbf{P}_{A_{i+1}} - \mathbf{P}_{A_i}| \text{ and } L_{B_i} = |\mathbf{P}_{B_{i+1}} - \mathbf{P}_{B_i}|. \quad (2)$$

The intermediate polygons in the shape blend are then computed by interpolating the respective vertex angles and edge lengths:

$$\alpha_0 = (1-t)\alpha_{A_0} + t\alpha_{B_0}, \quad (3)$$

$$\theta_i = (1-t)\theta_{A_i} + t\theta_{B_i}, \quad (i = 1, 2, \ldots, m). \quad (4)$$

$$L_i = (1-t)L_{A_i} + tL_{B_i}, \quad (i = 0, 1, 2, \ldots, m). \quad (5)$$

Unfortunately, the problem is not completely solved at this point, since the resulting polygon will not generally close. Figure 4 shows what the chicken and dancer polygons look like at this stage of the algorithm. It is somewhat surprising that

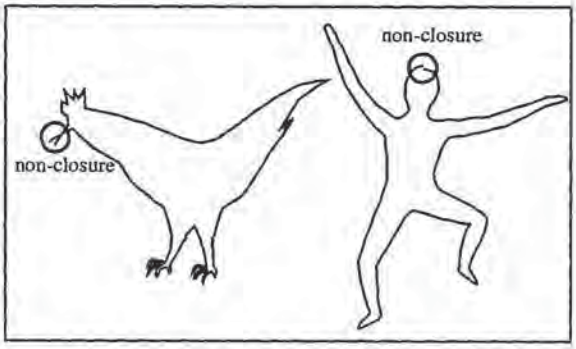



Figure 4: Unclosed polygons

these polygons, each with over 200 vertices, come so close to ending where they started (and this happens typically, in our experience). But the problem remains, how do we best adjust the lengths and angles so that the polygon *does* close.

There are two solutions to this problem. The first is to leave the angles unchanged and tweak the lengths (section 2.1). This turns out to have a straightforward, closed-form solution. The other approach is to treat the open polygon as a piece of wire for which we define the physical rules for stretching and vertex bending. Adjustments to angles and/or edges can then be computed iteratively by determining the equilibrium shape when the two open polygon vertices are forced to coincide.

### 2.1 Edge Tweaking

To close the polygons by adjusting the edge lengths only, rewrite equation 5 as

$$L_i = (1-t)L_{A_i} + tL_{B_i} + S_i, \quad (i = 0, 1, 2, \ldots, m). \quad (6)$$

It seems smart that the magnitudes of $S_i$ should roughly be proportional to $|L_{A_i} - L_{B_i}|$, since if an edge has the same length on both key polygons, it ought to have about that same

length throughout the shape blend. One can dream up simple examples for which this is not desirable, but for most reasonable cases, experience has verified this to be wise. Therefore, define

$$L_{AB_i} = \max\{|L_{A_i} - L_{B_i}|, L_{tol}\}, \quad (i = 0, 1, 2, \ldots, m). \quad (7)$$

where $L_{tol} = 0.0001 \times (\max_{i\in[0,m]} |L_{A_i} - L_{B_i}|)$ is needed to avoid division by zero.

Our goal is to find $S_0, S_1, \ldots, S_m$, so that the objective function

$$f(S_0, S_1, \ldots, S_m) = \sum_{i=0}^{m} \frac{S_i^2}{L_{AB_i}^2}$$

is minimized subject to the two equality constraints (which force closure of the polygon):

$$\varphi_1(S_0, S_1, \ldots, S_m) = \sum_{i=0}^{m} [(1-t)L_{A_i} + tL_{B_i} + S_i]\cos\alpha_i = 0,$$

$$\varphi_2(S_0, S_1, \ldots, S_m) = \sum_{i=0}^{m} [(1-t)L_{A_i} + tL_{B_i} + S_i]\sin\alpha_i = 0,$$

where $\alpha_i$ are the directed angles from the $x$-axis to the vectors $\mathbf{P}_i\mathbf{P}_{i+1}$,

$$\alpha_i = \alpha_{i-1} + \theta_i, \quad (i = 1, 2, \ldots, m). \quad (8)$$

The method of Lagrange multipliers [9] can now solve for the desired tweak values $S_i$ as follows. Set

$$\Phi(\lambda_1, \lambda_2, S_0, S_1, \ldots, S_m) = f + \lambda_1\varphi_1 + \lambda_2\varphi_2,$$

where $\lambda_1$ and $\lambda_2$ are the multipliers.

From

$$\begin{cases} \frac{\partial\Phi}{\partial S_i} = \frac{2S_i}{L_{AB_i}^2} + \lambda_1\cos\alpha_i + \lambda_2\sin\alpha_i = 0 (i = 0, 1, \ldots, m) \\ \sum_{i=0}^{m}[(1-t)L_{A_i} + tL_{B_i} + S_i]\cos\alpha_i = 0 \\ \sum_{i=0}^{m}[(1-t)L_{A_i} + tL_{B_i} + S_i]\sin\alpha_i = 0, \end{cases}$$

we obtain

$$\begin{cases} E\lambda_1 + F\lambda_2 = U \\ F\lambda_1 + G\lambda_2 = V, \end{cases} \quad (9)$$

where

$$E = \sum_{i=0}^{m} L_{AB_i}^2 \cos^2\alpha_i, \quad (10)$$

$$F = \sum_{i=0}^{m} L_{AB_i}^2 \sin\alpha_i cos\alpha_i, \quad (11)$$

$$G = \sum_{i=0}^{m} L_{AB_i}^2 \sin^2\alpha_i, \quad (12)$$

$$U = 2\left\{\sum_{i=0}^{m}[(1-t)L_{A_i} + tL_{B_i}]\cos\alpha_i\right\}, \quad (13)$$

$$V = 2\left\{\sum_{i=0}^{m}[(1-t)L_{A_i} + tL_{B_i}]\sin\alpha_i\right\}. \quad (14)$$

Thus under the condition $EG - F^2 \neq 0$ we can get

$$\lambda_1 = \begin{vmatrix} U & F \\ V & G \end{vmatrix} \Big/ \begin{vmatrix} E & F \\ F & G \end{vmatrix}, \quad (15)$$

$$\lambda_2 = \begin{vmatrix} E & U \\ F & V \end{vmatrix} \Big/ \begin{vmatrix} E & F \\ F & G \end{vmatrix}, \quad (16)$$

and

$$S_i = -\frac{1}{2}L_{AB_i}^2(\lambda_1\cos\alpha_i + \lambda_2\sin\alpha_i), \quad (i = 0, 1, \ldots, m). \quad (17)$$

Using equations 4, 6, and 8, we can now calculate the coordinates $(x_i, y_i)$ of the vertices $\mathbf{P}_i (i = 1, 2, \ldots, m)$:

$$x_i = x_{i-1} + L_{i-1}\cos\alpha_{i-1}, \quad y_i = y_{i-1} + L_{i-1}\sin\alpha_{i-1}. \quad (18)$$

## 2.2 Tweaking Lengths and/or Angles

The edge-tweaking-only method generally gives good results and is relatively fast. Also, as suggested from Figure 4, often very little edge length adjustment is needed.

However, some simple examples can be found where the edge lengths may change more than is desirable using the edge-tweaking-only method. This can be detected by checking the values of $S_i$. In such a case, the required edge length adjustments can be diminished by also allowing the angles to change.

A good solution to this problem is to treat the unclosed polygon as a piece of wire which can possibly stretch, but which can only bend at polygon vertices. The stretching stiffness for each polygon edge is inversely proportional to the change in length experienced by that edge between the two key frames. Likewise, the bending stiffness of each angle is inversely proportional to the change between key frames of the respective angle. These stiffness values tend to enforce rigidity for identical portions of the two key polygons.

The shape of the closed intermediate polygon is then computed by forcing the two unclosed joints to coincide, and determining the unique equilibrium shape of the wire. Further details can be found in [4].

# 3 Anchor points and angle lines

Since an intrinsic definition of a polygon is invariant to rigid body motion, a shape blend must specify translations and rotations for the intermediate shapes. Translation is specified using an anchor point path, and rotation is constrained by designating the rotation function of an angle line. The anchor point can be a polygon vertex, or any other point that is well defined for each step in the shape blend. For example, center of area is a good anchor point for objects in free fall. For bodies in free fall, such as a diver, a parabolic anchor path simulates the effects of gravity.

The angle line can be any line whose association with each shape in the blend can be determined, such as a non-degenerate polygon edge, the line between any two points on the polygon, or a principle axis of a shape (if the major and minor axes are well defined, ie., the product of inertia is non-zero). In Figure 3, the anchor point is $\mathbf{P}_0$ and the angle line is $L_0$.

# 4 Discussion

Figure 1 shows that the main advantage of using turtle graphics in shape blending is that it helps solve the withering arm problem. Another benefit is that it can provide more nearly monotonic angle changes than does linear-vertex-path shape blending. Figure 5 shows a shape blend, taken from [12], in which a shape which should undergo a simple rigid body motion experiences shrinking *and* kinking. The kinking occurs in this case because of a poor choice of vertices, a common occurrence in linear-vertex-path shape blending. Clearly, turtle graphics shape blending would have no problem in this case.

A more impelling example is the dancer's arm which withers under linear vertex path motion. The magnification in Figure 6 illustrates that the intrinsic method produces inherently smoother blends than the linear vertex paths. [12] goes

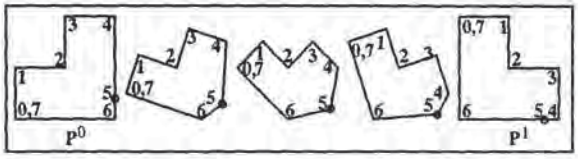



Figure 5: Shrinking plus kinking

to great lengths investigating how to minimize this angle non-monotonicity which can occur with linear vertex paths. As an added benefit of the intrinsic algorithm, this detailed search for non-monotonic angle changes is rendered unnecessary.

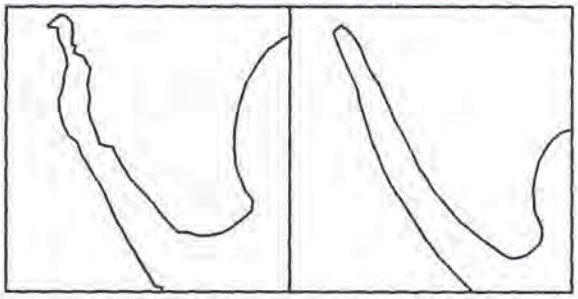



Figure 6: Closeup of dancer's arm

Although slower than the linear patch method, the intrinsic algorithm can compute a shape blend for the chicken in Figure 2.b (which has 230 vertices) in 0.02 seconds using the method in Section 2.1 and in 0.05 seconds using the method in Section 2.2, on an HP 730 workstation.

It is easy to contrive examples for which this algorithm performs poorly, although most of the realistic cases we have tried produced good results. In cases where some adjustment is called for, additional constraints can be imposed, such as specifying that the distance between specified pairs of non-adjacent polygon vertices should change monotonically from one key frame to the next. See [4] for more details.

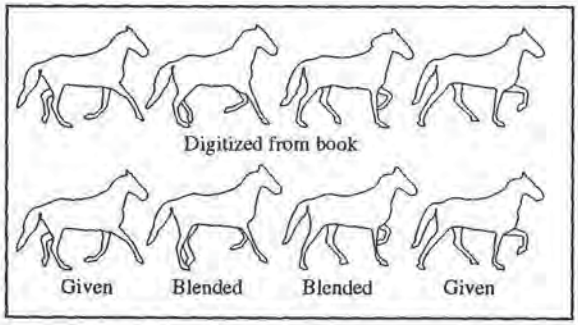



Figure 7: Cantering Horse

Experience suggests that this algorithm may work well enough for many applications to character animation. The sequence of a cantering horse in Figure 7 was taken from the classic photographic study, *Animals in Motion* [10], first published in 1887. The top four figures are digitizations of actual photographs from the book. In the bottom row, the middle two figures are shape blends interpolating the first and last figures. The vertex correspondence was determined using the algorithm in [12], and the vertex paths were computed using the algorithm in this paper. The horse's two left legs were treated as independent shape blends.

**Acknowledgements**
Geoffrey Slinker, Hank Christiansen, Alan Zundel, and Kris Klimaszewski provided much helpful discussion. This work was supported under NSF grant DMC-8657057.

## References

[1] J. Alan Adams. The intrinsic method for curve definition. *Computer-Aided Design*, 7(4):243–249, 1975.

[2] Harold J. Bailey, Kathleen M. Brautigam, and Trudy H. Doran. *Apple Logo.* Brady Communications Company, Inc., Bowie, MD, 1984.

[3] Shenchang Eric Chen and Richard Parent. Shape averaging and its applications to industrial design. *IEEE CG&A*, 9(1):47–54, 1989.

[4] Peisheng Gao. 2–d shape blending:
an intrinsic solution to the vertex path problem. Master's thesis, Brigham Young University, Department of Civil Engineering, 1993.

[5] Andrew Glassner. Metamorphosis. *preprint*, 1991.

[6] John F. Hughes. Scheduled fourier volume morphing. *Computer Graphics (Proc. SIGGRAPH)*, 26(2):43–46, 1992.

[7] Anil Kaul and Jarek Rossignac. Solid-interpolating deformations: Construction and animation of PIPs. In F.H. Post and W. Barth, editors, *Proc. Eurographics '91*, pages 493—505. Elsevier Science Publishers B.V, 1991.

[8] James R. Kent, Wayne E. Carlson, and Richard E. Parent. Shape transformation for polyhedral objects. *Computer Graphics (Proc. SIGGRAPH)*, 26(2):47–54, 1992.

[9] S. C. Malik. *Mathematical Analysis.* John Wiley & Sons, Inc., New York, 1984.

[10] Eadweard Muybridge. *Animals in Motion.* Dover Publications, Inc., New York, 1957.

[11] Thomas W. Sederberg and Eugene Greenwood. Shape blending of 2–d piecewise curves. *Submitted.*

[12] Thomas W. Sederberg and Eugene Greenwood. A physically based approach to 2–d shape blending. *Computer Graphics (Proc. SIGGRAPH)*, 26(2):25–34, 1992.

[13] Yoshihisa Shinagawa and Tosiyasu L. Kunii. The differential model: A model for animating transformation of objects using differntial information. In Tosiyasu L. Kunii, editor, *Modeling in Computer Graphics*, pages 5–15, Tokyo, 1991. Springer-Verlag.

[14] Geoffrey Slinker. Inbetweening using a physically based model and nonlinear path interpolation. Master's thesis, Brigham Young University, Department of Computer Science, 1992.



# Mesh Optimization

Hugues Hoppe* Tony DeRose* Tom Duchamp†
John McDonald‡ Werner Stuetzle‡

University of Washington
Seattle, WA 98195

## Abstract

We present a method for solving the following problem: Given a set of data points scattered in three dimensions and an initial triangular mesh $M_0$, produce a mesh $M$, of the same topological type as $M_0$, that fits the data well and has a small number of vertices. Our approach is to minimize an energy function that explicitly models the competing desires of conciseness of representation and fidelity to the data. We show that mesh optimization can be effectively used in at least two applications: surface reconstruction from unorganized points, and mesh simplification (the reduction of the number of vertices in an initially dense mesh of triangles).

**CR Categories and Subject Descriptors:** I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling.

**Additional Keywords:** Geometric Modeling, Surface Fitting, Three-Dimensional Shape Recovery, Range Data Analysis, Model Simplification.

## 1 Introduction

The *mesh optimization* problem considered in this paper can be roughly stated as follows: Given a collection of data points $X$ in $\mathbb{R}^3$ and an initial triangular mesh $M_0$ near the data, find a mesh $M$ of the same topological type as $M_0$ that fits the data well and has a small number of vertices.

As an example, Figure 7b shows a set of 4102 data points sampled from the object shown in Figure 7a. The input to the mesh optimization algorithm consists of the points together with the initial mesh shown in Figure 7c. The optimized mesh is shown in Figure 7h. Notice that the sharp edges and corners indicated by the data have been faithfully recovered and that the number of vertices has been significantly reduced (from 1572 to 163).

*Department of Computer Science and Engineering, FR-35
†Department of Mathematics, GN-50
‡Department of Statistics, GN-22

This work was supported in part by Bellcore, the Xerox Corporation, IBM, Hewlett-Packard, AT&T Bell Labs, the Digital Equipment Corporation, the Department of Energy under grant DE-FG06-85-ER25006, the National Library of Medicine under grant NIH LM-04174, and the National Science Foundation under grants CCR-8957323 and DMS-9103002.



To solve the mesh optimization problem we minimize an *energy function* that captures the competing desires of tight geometric fit and compact representation. The tradeoff between geometric fit and compact representation is controlled via a user-selectable parameter $c_{rep}$. A large value of $c_{rep}$ indicates that a sparse representation is to be strongly preferred over a dense one, usually at the expense of degrading the fit.

We use the input mesh $M_0$ as a starting point for a non-linear optimization process. During the optimization we vary the number of vertices, their positions, and their connectivity. Although we can give no guarantee of finding a global minimum, we have run the method on a wide variety of data sets; the method has produced good results in all cases (see Figure 1).

We see at least two applications of mesh optimization: surface reconstruction and mesh simplification.

The problem of surface reconstruction from sampled data occurs in many scientific and engineering applications. In [2], we outlined a two phase procedure for reconstructing a surface from a set of unorganized data points. The goal of phase one is to determine the topological type of the unknown surface and to obtain a crude estimate of its geometry. An algorithm for phase one was described in [5]. The goal of phase two is to improve the fit and reduce the number of faces. Mesh optimization can be used for this purpose.

Although we were originally led to consider the mesh optimization problem by our research on surface reconstruction, the algorithm we have developed can also be applied to the problem of mesh simplification. Mesh simplification, as considered by Turk [15] and Schroeder et al. [10], refers to the problem of reducing the number of faces in a dense mesh while minimally perturbing the shape. Mesh optimization can be used to solve this problem as follows: sample data points $X$ from the initial mesh and use the initial mesh as the starting point $M_0$ of the optimization procedure. For instance, Figure 7q shows a triangular approximation of a minimal surface with 2032 vertices. Application of our mesh optimization algorithm to a sample of 6752 points (Figure 7r) from this mesh produces the meshes shown in Figures 7s (487 vertices) and 7t (239 vertices). The mesh of Figure 7s corresponds to a relatively small value of $c_{rep}$, and therefore has more vertices than the mesh of Figure 7t which corresponds to a somewhat larger value of $c_{rep}$.

The principal contributions of this paper are:

- It presents an algorithm for fitting a mesh of arbitrary topological type to a set of data points (as opposed to volume data, etc.). During the fitting process, the number and connectivity of the vertices, as well as their positions, are allowed to vary.
- It casts mesh simplification as an optimization problem with an energy function that directly measures deviation of the final mesh from the original. As a consequence, the final mesh

 ACM-0-89791-601-8/93/008/0019 $01.50

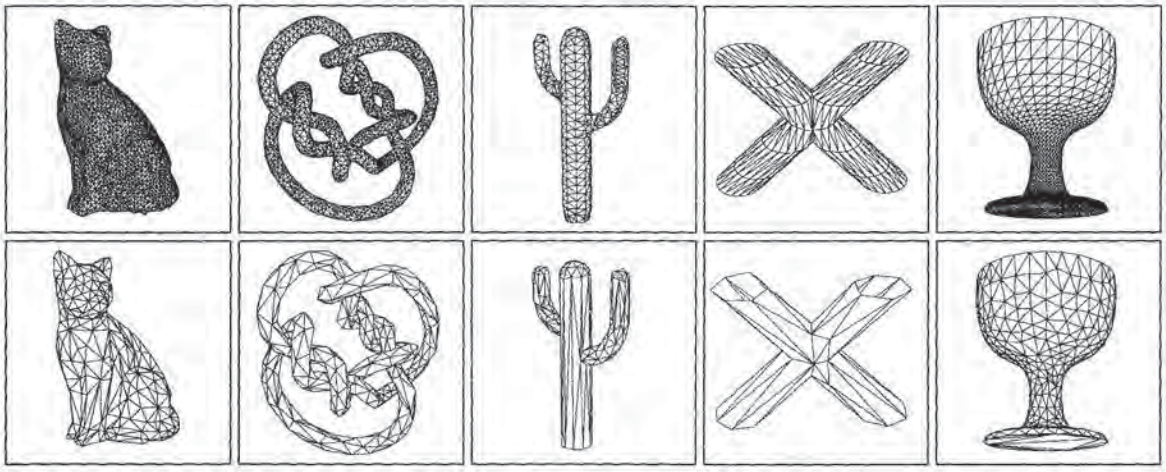


Figure 1: Examples of mesh optimization. The meshes in the top row are the initial meshes $M_0$; the meshes in the bottom row are the corresponding optimized meshes. The first 3 columns are reconstructions; the last 2 columns are simplifications.

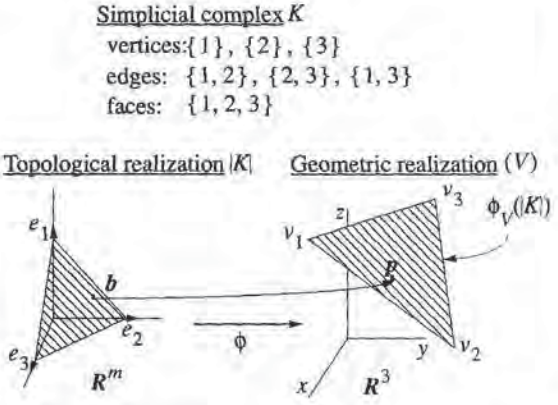



Figure 2: Example of mesh representation: a mesh consisting of a single face.

naturally adapts to curvature variations in the original mesh.

- It demonstrates how the algorithm's ability to recover sharp edges and corners can be exploited to automatically segment the final mesh into smooth connected components (see Figure 7i).

## 2 Mesh Representation

Intuitively, a *mesh* is a piecewise linear surface, consisting of triangular faces pasted together along their edges. For our purposes it is important to maintain the distinction between the connectivity of the vertices and their geometric positions. Formally, a mesh $M$ is a pair $(K, V)$, where: $K$ is a *simplicial complex* representing the connectivity of the vertices, edges, and faces, thus determining the topological type of the mesh; $V = \{\mathbf{v}_1, \ldots, \mathbf{v}_m\}$, $\mathbf{v}_i \in \mathbf{R}^3$ is a set of vertex positions defining the shape of the mesh in $\mathbf{R}^3$ (its geometric realization).

A simplicial complex $K$ consists of a set of vertices $\{1, \ldots, m\}$, together with a set of non-empty subsets of the vertices, called the simplices of $K$, such that any set consisting of exactly one vertex is a simplex in $K$, and every non-empty subset of a simplex in $K$ is again a simplex in $K$ (cf. Spanier [14]). The 0-simplices $\{i\} \in K$ are called vertices, the 1-simplices $\{i, j\} \in K$ are called edges, and the 2-simplices $\{i, j, k\} \in K$ are called faces.

A geometric realization of a mesh as a surface in $\mathbf{R}^3$ can be obtained as follows. For a given simplicial complex $K$, form its *topological realization* $|K|$ in $\mathbf{R}^m$ by identifying the vertices $\{1, \ldots, m\}$ with the standard basis vectors $\{\mathbf{e}_1, \ldots, \mathbf{e}_m\}$ of $\mathbf{R}^m$. For each simplex $s \in K$ let $|s|$ denote the convex hull of its vertices in $\mathbf{R}^m$, and let $|K| = \cup_{s \in K} |s|$. Let $\phi : \mathbf{R}^m \to \mathbf{R}^3$ be the linear map that sends the $i$-th standard basis vector $\mathbf{e}_i \in \mathbf{R}^m$ to $\mathbf{v}_i \in \mathbf{R}^3$ (see Figure 2).

The *geometric realization* of $M$ is the image $\phi_V(|K|)$, where we write the map as $\phi_V$ to emphasize that it is fully specified by the set of vertex positions $V = \{\mathbf{v}_1, \ldots, \mathbf{v}_m\}$. The map $\phi_V$ is called an *embedding* if it is 1-1, that is if $\phi_V(|K|)$ is not self-intersecting. Only a restricted set of vertex positions $V$ result in $\phi_V$ being an embedding.

If $\phi_V$ is an embedding, any point $\mathbf{p} \in \phi_V(|K|)$ can be parameterized by finding its unique pre-image on $|K|$. The vector $\mathbf{b} \in |K|$ with $\mathbf{p} = \phi_V(\mathbf{b})$ is called the *barycentric coordinate vector* of $\mathbf{p}$ (with respect to the simplicial complex $K$). Note that barycentric coordinate vectors are convex combinations of standard basis vectors $\mathbf{e}_i \in \mathbf{R}^m$ corresponding to the vertices of a face of $K$. Any barycentric coordinate vector has at most three non-zero entries; it has only two non-zero entries if it lies on an edge of $|K|$, and only one if it is a vertex.

## 3 Definition of the Energy Function

Recall that the goal of mesh optimization is to obtain a mesh that provides a good fit to the point set $X$ and has a small number of vertices. We find a simplicial complex $K$ and a set of vertex positions $V$ defining a mesh $M = (K, V)$ that minimizes the energy function

$$E(K, V) = E_{dist}(K, V) + E_{rep}(K) + E_{spring}(K, V).$$

The first two terms correspond to the two stated goals; the third term is motivated below.

The distance energy $E_{dist}$ is equal to the sum of squared distances from the points $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ to the mesh,

$$E_{dist}(K, V) = \sum_{i=1}^{n} d^2(\mathbf{x}_i, \phi_V(|K|)).$$

The representation energy $E_{rep}$ penalizes meshes with a large number of vertices. It is set to be proportional to the number of vertices $m$ of $K$:

$$E_{rep}(K) = c_{rep} m.$$

The optimization allows vertices to be both added to and removed from the mesh. When a vertex is added, the distance energy $E_{dist}$ is likely to be reduced; the term $E_{rep}$ makes this operation incur a penalty so that vertices are not added indefinitely. Similarly, one wants to remove vertices from a dense mesh even if $E_{dist}$ increases slightly; in this case $E_{rep}$ acts to encourage the vertex removal. The user-specified parameter $c_{rep}$ provides a controllable trade-off between fidelity of geometric fit and parsimony of representation.

We discovered, as others have before us [8], that minimizing $E_{dist} + E_{rep}$ does not produce the desired results. As an illustration of what can go wrong, Figure 7d shows the result of minimizing $E_{dist}$ alone. The estimated surface has several spikes in regions where there is no data. These spikes are a manifestation of the fundamental problem that a minimum of $E_{dist} + E_{rep}$ may not exist.

To guarantee the existence of a minimum [6], we add the third term, the spring energy $E_{spring}$. It places on each edge of the mesh a spring of rest length zero and spring constant $\kappa$:

$$E_{spring}(K, V) = \sum_{\{j,k\} \in K} \kappa \|\mathbf{v}_j - \mathbf{v}_k\|^2$$

It is worthwhile emphasizing that the spring energy is not a smoothness penalty. Our intent is not to penalize sharp dihedral angles in the mesh, since such features may be present in the underlying surface and should be recovered. We view $E_{spring}$ as a regularizing term that helps guide the optimization to a desirable local minimum. As the optimization converges to the solution, the magnitude of $E_{spring}$ can be gradually reduced. We return to this issue in Section 4.4.

For some applications we want the procedure to be scale-invariant, which is equivalent to defining a unitless energy function $E$. To achieve invariance under Euclidean motion and uniform scaling, the points $X$ and the initial mesh $M_0$ are pre-scaled uniformly to fit in a unit cube. After optimization, a post-processing step can undo this initial transformation.

# 4 Minimization of the Energy Function

Our goal is to minimize the energy function

$$E(K, V) = E_{dist}(K, V) + E_{rep}(K) + E_{spring}(K, V)$$

over the set $\mathcal{K}$ of simplicial complexes $K$ homeomorphic to the initial simplicial complex $K_0$, and the vertex positions V defining the embedding. We now present an outline of our optimization algorithm, a pseudo-code version of which appears in Figure 3. The details are deferred to the next two subsections.

To minimize $E(K, V)$ over both $K$ and $V$, we partition the problem into two nested subproblems: an inner minimization over $V$ for fixed simplicial complex $K$, and a outer minimization over $K$.

In Section 4.1 we describe an algorithm that solves the inner minimization problem. It finds $E(K) = \min_V E(K, V)$, the energy of the best possible embedding of the fixed simplicial complex $K$, and the corresponding vertex positions $V$, given an initial guess for $V$. This corresponds to the procedure OptimizeVertexPositions in Figure 3.

```
OptimizeMesh(K0,V0) {
    K := K0
    V := OptimizeVertexPositions(K0,V0)
    – Solve the outer minimization problem.
    repeat {
        (K',V') := GenerateLegalMove(K,V)
        V' = OptimizeVertexPositions(K',V')
        if E(K',V') < E(K,V) then
            (K,V) := (K',V')
        endif
    } until convergence
    return (K,V)
}

– Solve the inner optimization problem
–   E(K) = min_V E(K,V)
– for fixed simplicial complex K.
OptimizeVertexPositions(K,V) {
    repeat {
        – Compute barycentric coordinates by projection.
        B := ProjectPoints(K,V)
        – Minimize E(K,V,B) over V using conjugate gradients.
        V := ImproveVertexPositions(K,B)
    } until convergence
    return V
}

GenerateLegalMove(K,V) {
    Select a legal move K ⇒ K'.
    Locally modify V to obtain V' appropriate for K'.
    return (K',V')
}
```

Figure 3: An idealized pseudo-code version of the minimization algorithm.

Whereas the inner minimization is a continuous optimization problem, the outer minimization of $E(K)$ over the simplicial complexes $K \in \mathcal{K}$ (procedure OptimizeMesh) is a discrete optimization problem. An algorithm for its solution is presented in Section 4.2.

The energy function $E(K, V)$ depends on two parameters $c_{rep}$ and $\kappa$. The parameter $c_{rep}$ controls the tradeoff between conciseness and fidelity to the data and should be set by the user. The parameter $\kappa$, on the other hand, is a regularizing parameter that, ideally, would be chosen automatically. Our method of setting $\kappa$ is described in Section 4.4.

## 4.1 Optimization for Fixed Simplicial Complex
(Procedure OptimizeVertexPositions)

In this section, we consider the problem of finding a set of vertex positions $V$ that minimizes the energy function $E(K, V)$ for a given simplicial complex $K$. As $E_{rep}(K)$ does not depend on $V$, this amounts to minimizing $E_{dist}(K, V) + E_{spring}(K, V)$.

To evaluate the distance energy $E_{dist}(K, V)$, it is necessary to compute the distance of each data point $\mathbf{x}_i$ to $M = \phi_V(|K|)$. Each of these distances is itself the solution to the minimization problem

$$d^2(\mathbf{x}_i, \phi_V(|K|)) = \min_{\mathbf{b}_i \in |K|} \|\mathbf{x}_i - \phi_V(\mathbf{b}_i)\|^2,$$

in which the unknown is the barycentric coordinate vector $\mathbf{b}_i \in$

$|K| \subset \mathbf{R}^m$ of the projection of $\mathbf{x}_i$ onto $M$. Thus, minimizing $E(K,V)$ for fixed $K$ is equivalent to minimizing the new objective function

$$\begin{aligned} E(K,V,B) &= \sum_{i=1}^{n} \|\mathbf{x}_i - \phi_V(\mathbf{b}_i)\|^2 + E_{spring}(K,V) \\ &= \sum_{i=1}^{n} \|\mathbf{x}_i - \phi_V(\mathbf{b}_i)\|^2 + \sum_{\{j,k\}\in K} \kappa \|\mathbf{v}_j - \mathbf{v}_k\|^2 \end{aligned}$$

over the vertex positions $V = \{\mathbf{v}_1, \ldots, \mathbf{v}_m\}$, $\mathbf{v}_i \in \mathbf{R}^3$ and the barycentric coordinates $B = \{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$, $\mathbf{b}_i \in |K| \subset \mathbf{R}^m$.

To solve this optimization problem (procedure OptimizeVertexPositions), our method alternates between two subproblems:

1. For fixed vertex positions $V$, find optimal barycentric coordinate vectors $B$ by *projection* (procedure ProjectPoints).
2. For fixed barycentric coordinate vectors $B$, find optimal vertex positions $V$ by solving a *linear* least squares problem (procedure ImproveVertexPositions).

Because we find optimal solutions to both of these subproblems, $E(K,V,B)$ can never increase, and since it is bounded from below, it must converge. In principle, one could iterate until some formal convergence criterion is met. Instead, as is common, we perform a fixed number of iterations. As an example, Figure 7e shows the result of optimizing the mesh of Figure 7c over the vertex positions while holding the simplicial complex fixed.

It is conceivable that procedure OptimizeVertexPositions returns a set $V$ of vertices for which the mesh is self-intersecting, i.e. $\phi_V$ is not an embedding. While it is possible to check *a posteriori* whether $\phi_V$ is an embedding, constraining the optimization to always produce an embedding appears to be difficult. This has not presented a problem in the examples we have run.

#### 4.1.1 Projection Subproblem
(Procedure ProjectPoints)

The problem of optimizing $E(K,V,B)$ over the barycentric coordinate vectors $B = \{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$, while holding the vertex positions $V = \{\mathbf{v}_1, \ldots, \mathbf{v}_m\}$ and the simplicial complex $K$ constant, decomposes into $n$ separate optimization problems:

$$\mathbf{b}_i = \operatorname*{argmin}_{\mathbf{b}\in|K|} \|\mathbf{x}_i - \phi_V(\mathbf{b})\|$$

In other words, $\mathbf{b}_i$ is the barycentric coordinate vector corresponding to the point $\mathbf{p} \in \phi_V(|K|)$ closest to $\mathbf{x}_i$.

A naive approach to computing $\mathbf{b}_i$ is to project $\mathbf{x}_i$ onto all of the faces of $M$, and then find the projection with minimal distance. To speed up the projection, we first enter the faces of the mesh into a spatial partitioning data structure (similar to the one used in [16]). Then for each point $\mathbf{x}_i$ only a nearby subset of the faces needs to be considered, and the projection step takes expected time $O(n)$. For additional speedup we exploit coherence between iterations. Instead of projecting each point globally onto the mesh, we assume that a point's projection lies in a neighborhood of its projection in the previous iteration. Specifically, we project the point onto all faces that share a vertex with the previous face. Although this is a heuristic that can fail, it has performed well in practice.

#### 4.1.2 Linear Least Squares Subproblem
(Procedure ImproveVertexPositions)

Minimizing $E(K,V,B)$ over the vertex positions $V$ while holding $B$ and $K$ fixed is a linear least squares problem. It decomposes into three independent subproblems, one for each of the three coordinates of the vertex positions. We will write down the problem for the first coordinate.

Let $e$ be the number of edges (1-simplices) in $K$; note that $e$ is $O(m)$. Let $\mathbf{v}^1$ be the $m$-vector whose $i$-th element is the first coordinate of $\mathbf{v}_i$. Let $\mathbf{d}^1$ be the $(n+e)$-vector whose first $n$ elements are the first coordinates of the data points $\mathbf{x}_i$, and whose last $e$ elements are zero. With these definitions we can express the least squares problem for the first coordinate as minimizing $\|A\mathbf{v}^1 - \mathbf{d}^1\|^2$ over $\mathbf{v}^1$. The design matrix $A$ is an $(n+e) \times m$ matrix of scalars. The first $n$ rows of $A$ are the barycentric coordinate vectors $\mathbf{b}_i$. Each of the trailing $e$ rows contains 2 non-zero entries with values $\sqrt{\kappa}$ and $-\sqrt{\kappa}$ in the columns corresponding to the indices of the edge's endpoints. The first $n$ rows of the least squares problem correspond to $E_{dist}(K,V)$, while the last $e$ rows correspond to $E_{spring}(K,V)$. An important feature of the matrix $A$ is that it contains at most 3 non-zero entries in each row, for a total of $O(n+m)$ non-zero entries.

To solve the least squares problem, we use the conjugate gradient method (cf. [3]). This is an iterative method guaranteed to find the exact solution in as many iterations as there are distinct singular values of $A$, i.e. in at most $m$ iterations. Usually far fewer iterations are required to get a result with acceptable precision. For example, we find that for $m$ as large as $10^4$, as few as 200 iterations are sufficient.

The two time-consuming operations in each iteration of the conjugate gradient algorithm are the multiplication of $A$ by an $(n+e)$-vector and the multiplication of $A^T$ by an $m$-vector. Because $A$ is sparse, these two operations can be executed in $O(n+m)$ time. We store $A$ in a sparse form that requires only $O(n+m)$ space. Thus, an acceptable solution to the least squares problem is obtained in $O(n+m)$ time. In contrast, a typical noniterative method for solving dense least squares problems, such as QR decomposition, would require $O((n+m)m^2)$ time to find an exact solution.

### 4.2 Optimization over Simplicial Complexes
(Procedure OptimizeMesh)

To solve the outer minimization problem, minimizing $E(K)$ over $K$, we define a set of three elementary transformations, *edge collapse*, *edge split*, and *edge swap*, taking a simplicial complex $K$ to another simplicial complex $K'$ (see Figure 4).

We define a *legal move* to be the application of one of these elementary transformations to an edge of $K$ that leaves the topological type of $K$ unchanged. The set of elementary transformations is complete in the sense that *any* simplicial complex in $\mathcal{K}$ can be obtained from $K_0$ through a sequence of legal moves[1].

Our goal then is to find such a sequence taking us from $K_0$ to a minimum of $E(K)$. We do this using a variant of random descent: we randomly select a legal move, $K \Rightarrow K'$. If $E(K') < E(K)$, we accept the move, otherwise we try again. If a large number of trials fails to produce an acceptable move, we terminate the search.

More elaborate selection strategies, such as steepest descent or simulated annealing, are possible. As we have obtained good results with the simple strategy of random descent, we have not yet implemented the other strategies.

**Identifying Legal Moves** An edge split transformation is always a legal move, as it can never change the topological type of $K$. The other two transformations, on the other hand, can cause a change of

[1] In fact, we prove in [6] that edge collapse and edge split are sufficient; we include edge swap to allow the optimization procedure to "tunnel" through small hills in the energy function.

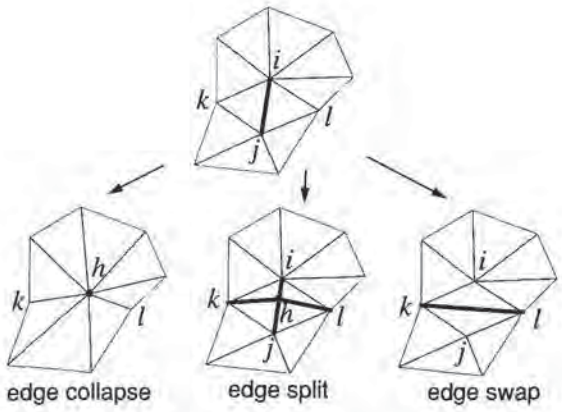



Figure 4: Local simplicial complex transformations

topological type, so tests must be performed to determine if they are legal moves.

We define an edge $\{i, j\} \in K$ to be a *boundary edge* if it is a subset of only one face $\{i, j, k\} \in K$, and a vertex $\{i\}$ to be a *boundary vertex* if there exists a boundary edge $\{i, j\} \in K$.

An edge collapse transformation $K \Rightarrow K'$ that collapses the edge $\{i, j\} \in K$ is a legal move if and only if the following conditions are satisfied (proof in [6]):

- For all vertices $\{k\}$ adjacent to both $\{i\}$ and $\{j\}$ ($\{i, k\} \in K$ and $\{j, k\} \in K$), $\{i, j, k\}$ is a face of $K$.
- If $\{i\}$ and $\{j\}$ are both boundary vertices, $\{i, j\}$ is a boundary edge.
- $K$ has more than 4 vertices if neither $\{i\}$ nor $\{j\}$ are boundary vertices, or $K$ has more than 3 vertices if either $\{i\}$ or $\{j\}$ are boundary vertices.

An edge swap transformation $K \Rightarrow K'$ that replaces the edge $\{i, j\} \in K$ with $\{k, l\} \in K'$ is a legal move if and only if $\{k, l\} \notin K$.

## 4.3 Exploiting Locality

The idealized algorithm described so far is too inefficient to be of practical use. In this section, we describe some heuristics which dramatically reduce the running time. These heuristics capitalize on the fact that a local change in the structure of the mesh leaves the optimal positions of distant vertices essentially unchanged.

### 4.3.1 Heuristics for Evaluating the Effect of Legal Moves

Our strategy for selecting legal moves requires evaluation of $E(K') = \min_V E(K', V)$ for a simplicial complex $K'$ obtained from $K$ through a legal move. Ideally, we would use procedure OptimizeVertexPositions of Section 4.1 for this purpose, as indicated in Figure 3. In practice, however, this is too slow. Instead, we use fast local heuristics to estimate the effect of a legal move on the energy function.

Each of the heuristics is based on extracting a submesh in the neighborhood of the transformation, along with the subset of the data points projecting onto the submesh. The change in overall energy is estimated by only considering the contribution of the submesh and the corresponding point set. This estimate is always pessimistic, as full optimization would only further reduce the energy.

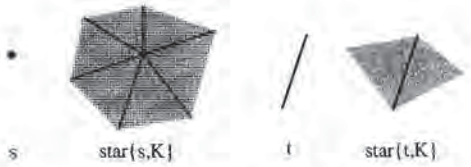



Figure 5: Neighborhood subsets of $K$.

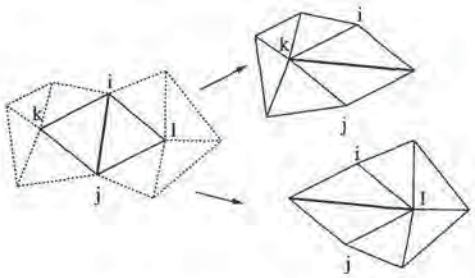



Figure 6: Two local optimizations to evaluate edge swap

Therefore, the heuristics never suggest changes that will increase the true energy of the mesh.

**Definition of neighborhoods in a simplicial complex** To refer to neighborhoods in a simplicial complex, we need to introduce some further notation. We write $s' \leq s$ to denote that simplex $s'$ is a non-empty subset of simplex $s$. For simplex $s \in K$, $\text{star}(s; K) = \{s' \in K : s \leq s'\}$ (Figure 5).

**Evaluation of Edge Collapse** To evaluate a transformation $K \Rightarrow K'$ collapsing an edge $\{i, j\}$ into a single vertex $\{h\}$ (Figure 4), we take the submesh to be $\text{star}(\{i\}; K) \cup \text{star}(\{j\}; K)$, and optimize over the single vertex position $\mathbf{v}_h$ while holding all other vertex positions constant.

Because we perform only a small number of iterations (for reasons of efficiency), the initial choice of $\mathbf{v}_h$ greatly influences the accuracy of the result. Therefore, we attempt three optimizations, with $\mathbf{v}_h$ starting at $\mathbf{v}_i$, $\mathbf{v}_j$, and $\frac{1}{2}(\mathbf{v}_i + \mathbf{v}_j)$, and accept the best one.

The edge collapse should be allowed only if the new mesh does not intersect itself. Checking for this would be costly; instead we settle for a less expensive heuristic check. If, after the local optimization, the maximum dihedral angle of the edges in $\text{star}(\{h\}; K')$ is greater than some threshold, the edge collapse is rejected.

**Evaluation of Edge Split** The procedure is the same as for edge collapse, except that the submesh is defined to be $\text{star}(\{i, j\}; K)$, and the initial value of the new vertex $\mathbf{v}_h$ is chosen to be $\frac{1}{2}(\mathbf{v}_i + \mathbf{v}_j)$.

**Evaluation of Edge Swap** To evaluate an edge swap transformation $K \Rightarrow K'$ that replaces an edge $\{i, j\} \in K$ with $\{k, l\} \in K'$, we consider two local optimizations, one with submesh $\text{star}(\{k\}; K')$, varying vertex $\mathbf{v}_k$, and one with submesh $\text{star}(\{l\}; K')$, varying vertex $\mathbf{v}_l$ (Figure 6). The change in energy is taken to best of these. As is the case in evaluating an edge collapse, we reject the transformation if the maximum dihedral angle after the local optimization exceeds a threshold.

#### 4.3.2 Legal Move Selection Strategy (Procedure GenerateLegalMove)

The simple strategy for selecting legal moves described in Section 4.2 can be improved by exploiting locality. Instead of selecting edges completely at random, edges are selected from a candidate set. This candidate set consists of all edges that may lead to beneficial moves, and initially contains all edges.

To generate a legal move, we randomly remove an edge from the candidate set. We first consider collapsing the edge, accepting the move if it is legal and reduces the total energy. If the edge collapse is not accepted, we then consider edge swap and edge split in that order. If one of the transformations is accepted, we update the candidate set by adding all neighboring edges. The candidate set becomes very useful toward the end of optimization, when the fraction of beneficial moves diminishes.

### 4.4 Setting of the Spring Constant

We view the spring energy $E_{spring}$ as a regularizing term that helps guide the optimization process to a good minimum. The spring constant $\kappa$ determines the contribution of this term to the total energy. We have obtained good results by making successive calls to procedure OptimizeMesh, each with a different value of $\kappa$, according to a schedule that gradually decreases $\kappa$.

As an example, to obtain the final mesh in Figure 7h starting from the mesh in Figure 7c, we successively set $\kappa$ to $10^{-2}, 10^{-3}, 10^{-4}$, and $10^{-8}$ (see Figures 7f–7h). This same schedule was used in all the examples.

## 5 Results

### 5.1 Surface Reconstruction

From the set of points shown in Figure 7b, phase one of our reconstruction algorithm [5] produces the mesh shown in Figure 7c; this mesh has the correct topological type, but it is rather dense, is far away from the data, and lacks the sharp features of the original model (Figure 7a). Using this mesh as a starting point, mesh optimization produces the mesh in Figure 7h.

Figures 7i–7k,7m–7o show two examples of surface reconstruction from actual laser range data (courtesy of Technical Arts, Redmond, WA). Figures 7j and 7n show sets of points obtained by sampling two physical objects (a distributor cap and a golf club head) with a laser range finder. The outputs of phase one are shown in Figures 7k and 7o. The holes present in the surface of Figure 7k are artifacts of the data, as self-shadowing prevented some regions of the surface from being scanned. Adaptive selection of scanning paths preventing such shadowing is an interesting area of future research. In this case, we manually filled the holes, leaving a single boundary at the bottom. Figures 7l and 7p show the optimized meshes obtained with our algorithm.

### 5.2 Mesh Simplification

For mesh simplification, we first sample a set of points randomly from the original mesh using uniform random sampling over area. Next, we add the vertices of the mesh to this point set. Finally, to more faithfully preserve the boundaries of the mesh, we sample additional points from boundary edges.

As an example of mesh simplification, we start with the mesh containing 2032 vertices shown in Figure 7q. From it, we obtain a sample of 6752 points shown in Figure 7r (4000 random points, 2032 vertex points, and 720 boundary points). Mesh optimization, with $c_{rep} = 10^{-5}$, reduces the mesh down to 487 vertices (Figure 7s). By setting $c_{rep} = 10^{-4}$, we obtain a coarser mesh of 239 vertices (Figure 7t).

| Fig. | #vert. $m$ | #faces | #data $n$ | Parameters $c_{rep}$ | Parameters $\kappa$ | Resulting energies $E_{dist}$ | Resulting energies $E$ | time (min.) |
|---|---|---|---|---|---|---|---|---|
| 7c | 1572 | 3152 | 4102 | - | - | $8.57\times10^{-2}$ | - | - |
| 7e | 1572 | 3152 | 4102 | $10^{-5}$ | $10^{-2}$ | $8.04\times10^{-4}$ | $4.84\times10^{-2}$ | 1.5 |
| 7f | 508 | 1024 | 4102 | $10^{-5}$ | $10^{-2}$ | $6.84\times10^{-4}$ | $3.62\times10^{-2}$ | (+3.0) |
| 7g | 270 | 548 | 4102 | $10^{-5}$ | $10^{-3}$ | $6.08\times10^{-4}$ | $6.94\times10^{-3}$ | (+2.2) |
| 7h | 163 | 334 | 4102 | $10^{-5}$ | *varied* | $4.86\times10^{-4}$ | $2.12\times10^{-3}$ | 17.0 |
| 7k | 9220 | 18272 | 12745 | - | - | $6.41\times10^{-2}$ | - | - |
| 7l | 690 | 1348 | 12745 | $10^{-5}$ | *varied* | $4.23\times10^{-3}$ | $1.18\times10^{-2}$ | 47.0 |
| 7o | 4059 | 8073 | 16864 | - | - | $2.20\times10^{-2}$ | - | - |
| 7p | 262 | 515 | 16864 | $10^{-5}$ | *varied* | $2.19\times10^{-3}$ | $4.95\times10^{-3}$ | 44.5 |
| 7q | 2032 | 3832 | - | - | - | - | - | - |
| 7s | 487 | 916 | 6752 | $10^{-5}$ | *varied* | $1.86\times10^{-3}$ | $8.05\times10^{-3}$ | 9.9 |
| 7t | 239 | 432 | 6752 | $10^{-4}$ | *varied* | $9.19\times10^{-3}$ | $4.39\times10^{-2}$ | 10.2 |

Table 1: Performance statistics for meshes shown in Figure 7.

As these examples illustrate, basing mesh simplification on a measure of distance between the simplified mesh and the original has a number of benefits:

- Vertices are dense in regions of high Gaussian curvature, whereas a few large faces span the flat regions.
- Long edges are aligned in directions of low curvature, and the aspect ratios of the triangles adjust to local curvature.
- Edges and vertices of the simplified mesh are placed near sharp features of the original mesh.

### 5.3 Segmentation

Mesh optimization enables us to detect sharp features in the underlying surface. Using a simple thresholding method, the optimized mesh can be segmented into smooth components. To this end, we build a graph in which the nodes are the faces of mesh. Two nodes of this graph are connected if the two corresponding faces are adjacent and their dihedral angle is smaller than a given threshold. The connected components of this graph identify the desired smooth segments. As an example, Figure 7i shows the segmentation of the optimized mesh into 11 components. After segmentation, vertex normals can be estimated from neighboring faces within each component, and a smoothly shaded surface can be created (Figure 7m).

### 5.4 Parameter Settings and Performance Statistics

Table 1 lists the specific parameter values of $c_{rep}$ and $\kappa$ used to generate the meshes in the examples, along with other performance statistics. In all these examples, the table entry *"varied"* refers to a spring constant schedule of $\{10^{-2}, 10^{-3}, 10^{-4}, 10^{-8}\}$. In fact, all meshes in Figure 1 are also created using the same parameters (except that $c_{rep}$ was changed in two cases). Execution times were obtained on a DEC uniprocessor Alpha workstation.

## 6 Related Work

**Surface Fitting** There is a large body of literature on fitting embeddings of a rectangular domain; see Bolle and Vemuri [1] for a review. Schudy and Ballard [11, 12] fit embeddings of a sphere to point data. Goshtasby [4] works with embeddings of cylinders and tori. Sclaroff and Pentland [13] consider embeddings of a deformed superquadric. Miller et al. [9] approximate an isosurface of volume data by fitting a mesh homeomorphic to a sphere. While it appears that their method could be extended to finding isosurfaces of arbitrary topological type, it it less obvious how it could be modified to

handle point instead of volume data. Mallet [7] discusses interpolation of functions over simplicial complexes of arbitrary topological type.

Our method allows fitting of a parametric surface of arbitrary topological type to a set of three-dimensional points. In [2], we sketched an algorithm for fitting a mesh of *fixed* vertex connectivity to the data. The algorithm presented here is an extension of this idea in which we also allow the number of vertices and their connectivity to vary. To the best of our knowledge, this has not been done before.

**Mesh Simplification** Two notable papers discussing the mesh simplification problem are Schroeder et al. [10] and Turk [15].

The motivation of Schroeder et al. is to simplify meshes generated by "marching cubes" that may consist of more than a million triangles. In their iterative approach, the basic operation is removal of a vertex and re-triangulation of the hole thus created. The criterion for vertex removal in the simplest case (interior vertex not on edge or corner) is the distance from the vertex to the plane approximating its surrounding vertices. It is worthwhile noting that this criterion only considers deviation of the new mesh from the mesh created in the previous iteration; deviation from the original mesh does not figure in the strategy.

Turk's goal is to reduce the amount of detail in a mesh while remaining faithful to the original topology and geometry. His basic idea is to distribute points on the existing mesh that are to become the new vertices. He then creates a triangulation containing both old and new vertices, and finally removes the old vertices. The density of the new vertices is chosen to be higher in areas of high curvature.

The principal advantage of our mesh simplification method compared to the techniques mentioned above is that we cast mesh simplification as an optimization problem: we find a new mesh of lower complexity that is as close as possible to the original mesh. This is recognized as a desirable property by Turk (Section 8, p. 63): "Another topic is finding measures of how closely matched a given re-tiling is to the original model. Can such a quality measure be used to guide the re-tiling process?". Optimization automatically retains more vertices in areas of high curvature, and leads to faces that are elongated along directions of low curvature, another property recognized as desirable by Turk.

# 7 Summary and Future Work

We have described an energy minimization approach to solving the mesh optimization problem. The energy function we use consists of three terms: a distance energy that measures the closeness of fit, a representation energy that penalizes meshes with a large number of vertices, and a regularizing term that conceptually places springs of rest length zero on the edges of the mesh. Our minimization algorithm partitions the problem into two nested subproblems: an inner continuous minimization and an outer discrete minimization. The search space consists of all meshes homeomorphic to the starting mesh.

Mesh optimization has proven effective as the second phase of our method for surface reconstruction from unorganized points, as discussed in [5]. (Phase two is responsible for improving the geometric fit and reducing the number of vertices of the mesh produced in phase one.)

Our method has also performed well for mesh simplification, that is, the reduction of the number of vertices in a dense triangular mesh. It produces meshes whose edges align themselves along directions of low curvature, and whose vertices concentrate in areas of high Gaussian curvature. Because the energy does not penalize surfaces with sharp dihedral angles, the method can recover sharp edges and corners.

A number of areas of future research still remain, including:

- Investigate the use of more sophisticated optimization methods, such as simulated annealing for discrete optimization and quadratic methods for non-linear least squares optimization, in order to avoid undesirable local minima in the energy and to accelerate convergence.
- Gain more insight into the use of the spring energy as a regularizing term, especially in the presence of appreciable noise.
- Improve the speed of the algorithm and investigate implementations on parallel architectures.
- Develop methods for fitting higher order splines to more accurately and concisely model curved surfaces.
- Experiment with sparse, non-uniform, and noisy data.
- Extend the current algorithm to other distance measures such as maximum error ($L^\infty$ norm) or average error ($L^1$ norm), instead of the current $L^2$ norm.

# References


[1] Ruud M. Bolle and Baba C. Vemuri. On three-dimensional surface reconstruction methods. *IEEE PAMI*, 13(1):1–13, January 1991.

[2] T. DeRose, H. Hoppe, T. Duchamp, J. McDonald, and W. Stuetzle. Fitting of surfaces to scattered data. *SPIE*, 1830:212–220, 1992.

[3] Gene Golub and Charles Van Loan. *Matrix Computations*. John Hopkins University Press, 2nd edition, 1989.

[4] Ardeshir Goshtasby. Surface reconstruction from scattered measurements. *SPIE*, 1830:247–256, 1992.

[5] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Surface reconstruction from unorganized points. *Computer Graphics (SIGGRAPH '92 Proceedings)*, 26(2):71–78, July 1992.

[6] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh optimization. TR 93-01-01, Dept. of Computer Science and Engineering, University of Washington, January 1993.

[7] J.L. Mallet. Discrete smooth interpolation in geometric modeling. *CAD*, 24(4):178–191, April 1992.

[8] Samuel Marin and Philip Smith. Parametric approximation of data using ODR splines. GMR 7057, General Motors Research Laboratories, May 1990.

[9] J.V. Miller, D.E. Breen, W.E. Lorensen, R.M. O'Bara, and M.J. Wozny. Geometrically deformed models: A method for extracting closed geometric models from volume data. *Computer Graphics (SIGGRAPH '91 Proceedings)*, 25(4):217–226, July 1991.

[10] William Schroeder, Jonathan Zarge, and William Lorensen. Decimation of triangle meshes. *Computer Graphics (SIGGRAPH '92 Proceedings)*, 26(2):65–70, July 1992.

[11] R. B. Schudy and D. H. Ballard. Model detection of cardiac chambers in ultrasound images. Technical Report 12, Computer Science Department, University of Rochester, 1978.

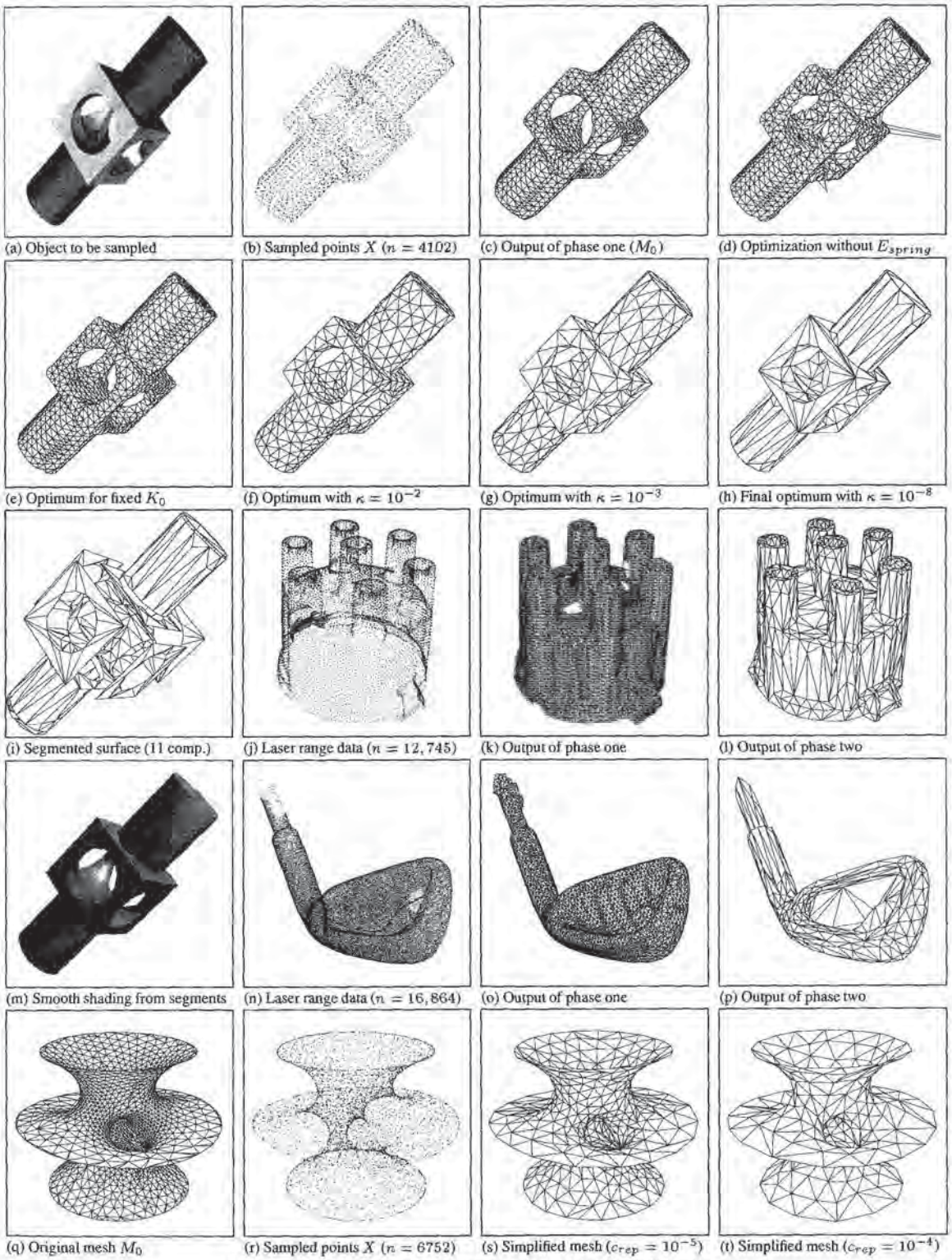
[12] R. B. Schudy and D. H. Ballard. Towards an anatomical model of heart motion as seen in 4-d cardiac ultrasound data. In *Proceedings of the 6th Conference on Computer Applications in Radiology and Computer-Aided Analysis of Radiological Images*, 1979.

[13] Stan Sclaroff and Alex Pentland. Generalized implicit functions for computer graphics. *Computer Graphics (SIGGRAPH '91 Proceedings)*, 25(4):247–250, July 1991.

[14] E. H. Spanier. *Algebraic Topology*. McGraw-Hill, New York, 1966.

[15] Greg Turk. Re-tiling polygonal surfaces. *Computer Graphics (SIGGRAPH '92 Proceedings)*, 26(2):55–64, July 1992.

[16] G. Wyvill, C. McPheeters, and B. Wyvill. Data structures for soft objects. *The Visual Computer*, 2(4):227–234, August 1986.



(a) Object to be sampled
(b) Sampled points $X$ ($n = 4102$)
(c) Output of phase one ($M_0$)
(d) Optimization without $E_{spring}$
(e) Optimum for fixed $K_0$
(f) Optimum with $\kappa = 10^{-2}$
(g) Optimum with $\kappa = 10^{-3}$
(h) Final optimum with $\kappa = 10^{-8}$
(i) Segmented surface (11 comp.)
(j) Laser range data ($n = 12,745$)
(k) Output of phase one
(l) Output of phase two
(m) Smooth shading from segments
(n) Laser range data ($n = 16,864$)
(o) Output of phase one
(p) Output of phase two
(q) Original mesh $M_0$
(r) Sampled points $X$ ($n = 6752$)
(s) Simplified mesh ($c_{rep} = 10^{-5}$)
(t) Simplified mesh ($c_{rep} = 10^{-4}$)

Figure 7: Examples of surface reconstruction and mesh simplification.



# Interactive Texture Mapping

Jérôme Maillot* , Hussein Yahia‡, Anne Verroust‡
* Thomson Digital Image
‡ INRIA-Rocquencourt

## Abstract

This paper describes a new approach to texture mapping. A global method to lower the distortion of the mapped image is presented; by considering a general optimization function we view the mapping as an energy-minimization process. We have constructed an interactive texture tool, which is fast and easy to use, to manipulate atlases in texture space. We present the tool's large set of interactive operations on mapping functions. We also introduce an algorithm which automatically generates an atlas for any type of object. These techniques allow the mapping of different textures onto the same object and handle non-continuous mapping functions, needed for complicated mapped objects.

**CR Categories and subject descriptors:** I.3.3 [Computer Graphics] Picture/Image Generation. I.3.7 [Computer Graphics] Graphics and Realism - Color, Shading and Texture.

**Additional Keywords:** Texture Mapping, Texture Map Distortion, Realistic Rendering, Interaction.

## 1 Introduction.

Texture mapping is a method in Computer Graphics to enhance the richness of computer-generated images [3, 13]. A texture is a 2D image to be mapped onto a synthetic 3D object. The 2D space of the texture image is often called texture space. Each point on the object has to be associated with an element in texture space. One of the first algorithms used the parametric representation of patches to find texture addresses [3]. With this method, some problems may occur at the junction of two patches [4]. Another method is to project the texture onto the object using an intermediate 3D shape like a box or a cylinder [2]. Also, some applications have been developed to provide direct drawing onto the object, in which the user interactively modifies the texture via the mapping function [12].

* Thomson Digital Image, 20-22, rue Hégésippe Moreau, 75018 Paris, France.

‡ INRIA-Rocquencourt B.P. 105, 78153 Le Chesnay, France.



In addition, the rendering of an object mapped with a pre-existing image, such as a digitized one, has been improved using dedicated techniques [3, 8, 2, 1]. We are interested in the case of 2D textures. In general, there is no natural mapping from the texture to the object: the image is necessarily distorted (see [13]). Also, there is a need for interactive tools to help the user define how to map a pre-defined image onto a surface, or to improve a mapping function. We present new methods for solving these problems.

We first propose, in section two, a mathematical formulation for the distortion of the mapped image. After deriving a general formula for the deformation energy, we describe simplified formulae, fast enough to be used inside an interactive loop to improve the mapping function.

Section three is devoted to the notion of an *atlas*, derived from a mathematical notion and adapted to the special case of textures. In the first subsection curvature is used to build an ultrametric for automatically creating atlases on any object. In the second subsection a set of interactive functions is presented for manipulating atlases.

We address only the case of polygonized surfaces. This simplifies the definition of the mapping function to one that depends only on the position in the texture plane of the points associated to all vertices. Our texture mapping tool is designed for a naive user who is not necessarily a specialist in image synthesis. The interface is very intuitive for the basic functions, and also provides some control structures which free the user from repetitive tasks. This program could be used, for example, by fashion designers to map woven or leather textures onto polygonal surfaces describing shoes, clothes or seats.

## 2 Deformation measures.

The first problem one has to solve when mapping textures is to define the quality of the final rendered object. We propose to measure the distortion introduced by the mapping as the deformation energy $E$. In the first subsection, we derive a general formula for $E$. We then propose, in the following subsections, a simplified formula for $E$, whose minimization can be done in real-time, and which gives very good visual results.

 ACM-0-89791-601-8/93/008/0027 $01.50

## 2.1 General formula.

Suppose that the surface on which we want to map textures is defined by a parametric function:

$$\phi : U \longrightarrow \mathcal{E}$$

$U$ being an open set of $\mathbb{R}^2$ and $\mathcal{E}$ Euclidean ordinary 3-space: to each point $(u, v)$ of $U$, $\phi$ associates a point $\phi(u, v)$ on the surface; $\phi$ thus defines a trivial mapping function onto the surface (for example, in [5], the surface is a bicubic patch, and the mapping function is exactly given by the parameterization). It should be noticed that any regular surface can be defined in this way, at least locally.

If the texture is to be mapped onto an elastic surface, one can measure the deformation of the texture through $\phi$ by computing the elastic deformation of the planar section when one applies $\phi$ to it. We use the first fundamental form $I_\phi$ [9, chap 2.5, 4.2] to measure at each point $(u, v) \in U$ the differences of lengths and angles between the initial plane and the tangent plane of the surface. Denoting by $\nabla\phi$ the Jacobian Matrix of $\phi$, we let

$$I_\phi(u, v) = \nabla\phi \cdot {}^t\nabla\phi \qquad (1)$$

In particular, $I_\phi$ is the identity matrix of $\mathbb{R}^2$ if and only if $\phi$ is an (infinitesimal) isometry, which means that the mapping function does not distort the image. Let Id be the identity matrix of $\mathbb{R}^2$, and $||\ ||$ be any norm defined on the set of $2 \times 2$ matrices. We can take as a measure of deformation energy at a point $(u, v)$ of parameter space the quantity $||I_\phi(u, v) - \mathrm{Id}||^2$, known by mechanical engineers as the Green-Lagrange deformation tensor. The deformation energy $E$ can then be defined over the whole underlying set $U$ since

$$E(U) = \int\int_U ||I_\phi - \mathrm{Id}||^2 \, du dv = \int\int_U ||e||^2 \, du dv \qquad (2)$$

This equation can be written as (see Appendix A):

$$E(U) = \int\int_U (\frac{\partial\phi}{\partial u}^2 - 1)^2 + 2(\frac{\partial\phi}{\partial u} \cdot \frac{\partial\phi}{\partial v})^2 + (\frac{\partial\phi}{\partial v}^2 - 1)^2 du dv \qquad (3)$$

If the surface is defined locally by non-overlapping regions, then the total energy is obtained by summing all the energies of each region $U_i$: $E = \sum_i E(U_i)$.

## 2.2 Interpretation in the Linear Theory of Elasticity.

We may imagine that our surface is made of rubber, and that we want to deform it in such a way that it can be equated with the texture image. If the material is isotropic, the elastic energy depends only on two parameters $\lambda$ and $\mu$ (see [17, 6]). Writing $\mathrm{tr}(e)$ for the trace of matrix $e$ (the sum of its diagonal terms), then:

$$E = \int\int_U \frac{\lambda}{2} (\mathrm{tr}(e))^2 + \mu \, \mathrm{tr}(e^2) \qquad (4)$$

assuming that $\lambda + \mu > 0$ and $\mu > 0$, which express that $E$ is positive definite. Since only the ratio $\frac{\lambda}{\mu}$ is significant for comparing mapping distortions, we can take $\mu = 1$. The coefficient $\lambda$ characterizes how the material is deformed orthogonally to the direction of a tensile stress. All existing materials have a positive $\lambda$, referring the fact that any object shrinks in direction $v$ when one stretches it along $u$. With $\lambda = 0$ the deformation in $u$ and $v$ are independent. Figure 1 shows the influence of the sign of $\lambda$ on the shape of a deformed object. Equation 3 correspond to the case $\lambda = 0$, $\mu = 1$. For texture mapping, except in very special cases, the best results are obtained with by setting $\lambda = 0$. It is important to note that two symmetrical surfaces have the same deformation energy. This implies that it is not possible to determine whether the mapping function inverts the image or not. We will show in section 2.4 that this may lead to problems.

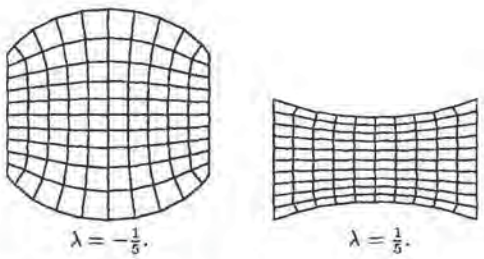



Fig. 1: Two squares of elastic material with different values of $\lambda$ were stretched in $u$ direction.

## 2.3 Triangulated surfaces.

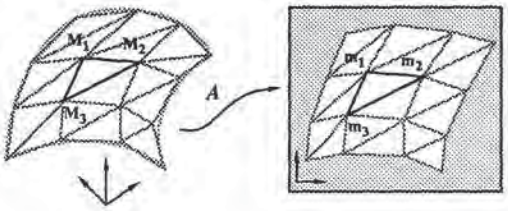



Fig. 2: Locally, the mapping function is an affine application $A$ which associates triangle $M_1M_2M_3$ with triangle $m_1m_2m_3$. It is the inverse function of $\phi$.

The deformation of the mapped image can be computed by evaluating the deformation of each triangle. As a result, the parametric function $\phi$ is affine (see figure 2), and its gradient is the associated linear map. Write $M_i$ and $m_i$, $i \in \{1, 2, 3\}$, for the vertices in $\mathbb{R}^3$ and their associated positions in texture plane. Then $\phi$ is defined by nine numbers. If the previous formula is expanded, the total energy can be expressed as the sum of rational fractions of $m_i$, with numerator of degree 8 and denominator of degree 6. To find the best mapping function, one has to find the coordinates of all $m_i$'s that minimize this energy. The solution can only be computed numerically, using an optimization method. The efficient algorithms need to know the gradient of the energy. Even in the simplest case of triangulated surfaces, the expression is complex and long to process. One can remark that most of the existing finite

element programs are built to treat linear elasticity*. They are highly optimized, but cannot be used to solve texture mapping problems. Thus, to be fast enough for interactive applications, the optimization process requires a simplified form of the energy equation that we present below.

In [1] a flattening algorithm is proposed for parametric patches. It is based on a relaxation procedure and runs incrementally. We want to find a global minimum for any type of polyhedral surface, using energy-minimization techniques. The method to be presented here will tackle the problem from a different point of view.

## 2.4 A simple, distance based energy.

If the surface is triangulated, its first fundamental form is completely characterized by the length of its edges. Furthermore, the lengths measured in $\mathbb{R}^3$ and in the texture plane are all the same if and only if the mapping function is an isometry. This is a consequence of the fact that two triangles are isometric as soon as their three edges have same lengths. Let's introduce the length energy $E_l$. The simplest form of energy that preserves length is the following:

$$E_l = \sum_{(i,j)\in \text{Edges}} \frac{\left(||m_i - m_j||^2 - ||M_i - M_j||^2\right)^2}{||M_i - M_j||^2} \qquad (5)$$

Using the squared norm gives us a simple form for the gradient. $E_l$ represents the energy of a spring net initially lying on the surface, and for which each spring induces a force proportional to the square of the distance (instead of the distance, as for classical springs). This give a higher energy for the most elongated springs than is given by the classical spring response, and thus increases the mean elongation, but lowers the maximum elongation. The final state is not very different from what would be obtained with standard springs, but using this formula we obtain a faster optimization algorithm. Normalization (that is, dividing by the term $||M_i - M_j||^2$) is chosen so that the energy does not change when the surface is subdivided by splitting each triangle into four similar parts. Without such a normalization, an object with very different face sizes would not be processed correctly. This would be the case, for example, if the surface is constructed from hierarchical splines.

Taking the symmetry of formula 5 into account, the part of the energy depending on point $i$ is:

$$E_l = 2 \sum_{m_k \text{ adjacent to } m_i} \frac{\left(||m_i - m_k||^2 - ||M_i - M_k||^2\right)^2}{||M_i - M_k||^2} \qquad (6)$$

The energy gradient is a degree three polynomial:

$$\frac{\partial E_l}{\partial x_i} = 8 \sum_k \frac{\left(||m_i - m_k||^2 - ||M_i - M_k||^2\right)}{||M_i - M_k||^2} (x_i - x_k)$$

$$\frac{\partial E_l}{\partial y_i} = 8 \sum_k \frac{\left(||m_i - m_k||^2 - ||M_i - M_k||^2\right)}{||M_i - M_k||^2} (y_i - y_k)$$

This form of energy is easy to compute and gives good results as long as the surface is simple. But when the surface is difficult to map (when the total curvature is too large), some triangles reverse their orientations. One can notice this effect in figure 3. In figure 3a we show two objects: object 1 on the left and object 2 on the right. Object 2 is just object 1 with a little band added near the equator. In figure 3b we see the results of minimization, orthogonally mapped into texture space: the contraction becomes too high at the center and the best solution to preserve the lengths is to "fold" the map. This is a direct consequence of the fact that the energies of two symmetrical triangles are the same. The same problem occurs with linear springs. If one compresses a spring along its axis, it bends and its projection onto its axis may overlap. This phenomenon is named *buckling* (see [7]) in elasticity. The final result is chaotic, in the sense that compressing two almost identical springs may produce two very different results.

Fig. 3.a: Object 1 and object 2

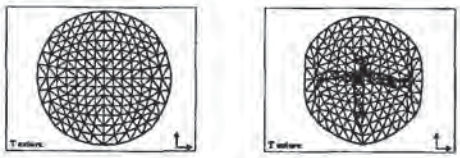


Fig. 3.b: The two maps.

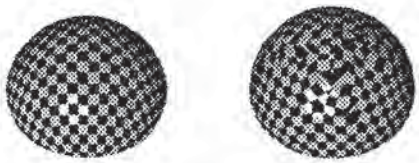


Fig. 3.c: Rendered objects.

Fig. 3: Problems with length based energy.

The two main problems with energy measure $E_l$ are that, firstly, the final state of the map becomes unstable when the object is complex. We can see for example in figure 3 that the symmetry along the $X$ and $Y$ axes is broken in the map. The result may depend strongly on small numerical errors. Secondly, the rendering is poor when some triangles have reversed their orientations. The patterns are multiplied, and grouped by triples, with opposite orientations. Such a final state is not acceptable.

## 2.5 A surface and length based energy.

To solve the problem of overlapping regions in the texture map, a second term can be added in the energy formula. It is chosen so that wrongly-oriented triangles will have

*Although in linear elasticity theory, the displacement of any point is supposed to remain small compared to the object, this is not the case in a rotation, for example. Such an approximation gives very bad results when applied to texture mapping.

a high energy. Energy $E_s$ is defined using the difference of signed areas for each triangle. It can be computed with cross products in $\mathbb{R}^3$ and determinants in the texture plane. The final energy is a linear combination of $E_l$ and $E_s$. By default we take the arithmetic mean.

$$E_s = \sum_{M_iM_jM_k \text{ triangle}} \frac{\left[\det(\overrightarrow{m_im_j}, \overrightarrow{m_im_k}) - \left\|\overrightarrow{M_iM_j} \wedge \overrightarrow{M_iM_k}\right\|\right]^2}{\left\|\overrightarrow{M_iM_j} \wedge \overrightarrow{M_iM_k}\right\|}$$

For this definition, the surface is implicitly supposed to be orientable, and with all triangles $M_iM_jM_k$ described in a direct sense, according to the normal. Again we

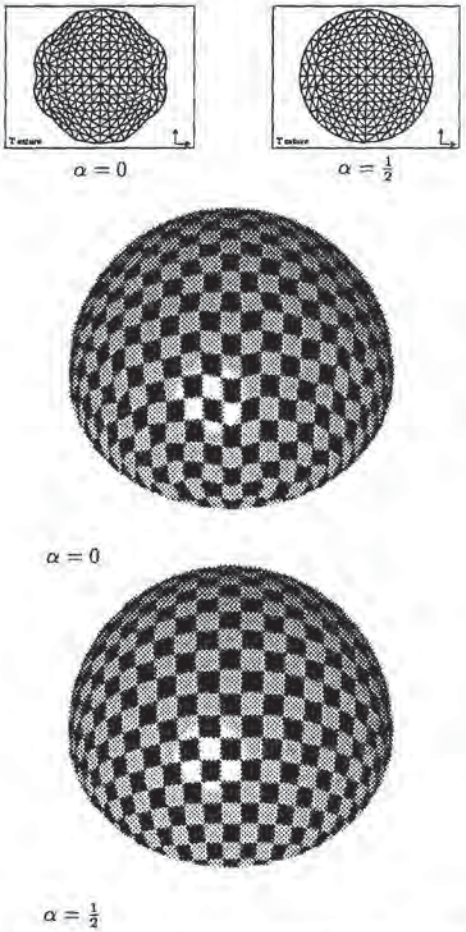



Fig. 4: Object 2 from figure 3 optimized with $E = \alpha E_l + (1-\alpha)E_s$.

normalize $E_s$. The dimensions of $E_l$ and $E_s$ are identical, which justifies the final form:

$$E = \alpha E_l + (1-\alpha)E_s$$

where $\alpha$ is a real coefficient to be taken between 0 and 1.

Gradient $\nabla E$ is a degree three polynomial in $x_i$ and $y_i$ which can be quickly computed. Thus, a conjugate gradient method can be used to find the best mapping function [18].

Case $\alpha = 1$ correspond to $E_l$ and is not satisfactory. Figure 4 shows that $\alpha = 0$ is not good either. The problem is that there are an infinite number of triangles with the same surface. In particular, when a vertex is translated in a parallel direction with the opposite edge, the surface does not change. This explains why triangles appear so stretched close to the border of the object. However, we have measured the surface differences surface, and our experience is that the differences between planar and 3-D triangles was less than 0.1%. A good result is obtained in figure 4 by combining both terms of the energy. Depending on the geometry, the optimal visual effect is obtained by tuning $\alpha$ between 0 and 1. In many cases, $\alpha = \frac{1}{2}$ is satisfactory.

There are still some objects for which optimization with any value of $\alpha$ would give bad results. In these cases, the problem results from the fact that the object is too complicated to be mapped with a single image. The solution is then to use atlases, as we will show in the next section.

## 3 Use of atlases.

As we have noticed in the previous section, a global continuous mapping function may excessively distort the image of a complex or a highly curved object. The natural way to solve this problem is to split the object into several independent regions. The practice has been to do so implicitly using the construction of the object. For example in [16], a textured teapot is split into three parts whose shape comes from the patch description. To be as general as possible, we disconnect the texturing regions from the 3-D representation of the surface: the user may want to represent a surface mapped only partially (for example, an object with a logo stuck on), with different textures, such as a patchwork, or with local discontinuities (as on some clothes). Thus we introduce a data structure called an "atlas" which is derived from the notion of atlas used in differential geometry [14, 21]. In our case, an *atlas* is composed of a set of *charts* $\{\phi_1, ..., \phi_n\}$, where each $\phi_i$ is an application from a subset $U_i$ of the surface to the Euclidean plane, such that:

- $\{U_1, ..., U_n\}$ is a cover of the surface,
- each $\phi_i$ is continuous inside the faces, and discontinuities are allowed along edges
- for $i \neq j$, $\phi_i$ and $\phi_j$ do not overlap except on the edges.

Each chart is associated to its own image. The words *atlas* and *chart* have here slightly different meanings than the ones mathematicians give them, due to differences in the regularity and boundary conditions, but the main idea of atlases is kept: the covering of a surface.

Good atlases are closely linked to the geometry of the object and can be difficult to build. Hence in the first subsection we present an algorithm to automatically build an atlas from scratch. Then we describe in the second subsection an interactive tool which makes easy the manipulation of atlases and of texture mappings for a given polyhedral surface.

### 3.1 A creation tool.

To automatically define an atlas, it seems natural to use the curvature information: a surface is developable (isometric

to a plane) if and only if the curvature matrix has a zero determinant [9, pp. 194-197]. In fact, a first rough subdivision of the surface in buckets is made using only the normal vectors of the surface and then the curvature information is used to control the merging of adjacent buckets and obtain the atlas. Then the regions are flattened on the texture plane (a more detailed description of the whole process can be found in [15]). The curvature information is essential in our computation. Let us describe first how we proceed to get this information.

### 3.1.1 Computing the curvature.

There exist precise but costly algorithms to compute the curvature of a polyhedral surface [19]. Since interactive visual feedback is one of our main goals, we designed a very fast algorithm, whose results are accurate enough for our needs. Let $u$ be a tangent vector to the surface and $\underline{N}$ the Gauss map [21] which associates to each point of the surface the unit normal vector at that point. The unit normal vector is represented as a point on the unit sphere $\mathcal{S}^2$. Curvature in direction $u$ is defined by the formula [9, pp. 135-151]:

$$C(u) = u \cdot \mathrm{d}\underline{N}(u) \tag{7}$$

We use finite differences to approximate derivatives. Normals must be evaluated at three close, non-aligned locations to evaluate the three coefficients of the curvature matrix*. We use smoothed normals coming from the rendering procedure. We put normal $N$ at the center of gravity of the face, $G$. We get the point $G$, normal $N$, and a set of vertices $S_i$ associated to normals $N_i$. With this data we seek to evaluate $\mathrm{d}\underline{N}$: we must find a symmetrical linear map $\mathcal{L}$ in the tangent plane $\mathcal{P} = (G, \{N\}^{\perp})$ such that, for all $i$, $\mathcal{L}(\overrightarrow{GS_i})$ draws nearer to $N_i - N$. Formula 7 shows that the matrix of the linear map $\mathcal{L}$ is also a curvature matrix. To eliminate the case of non-planar facets, all differences $N_i - N$ are projected in plane $\mathcal{P}$. Coefficients of curvature are then computed using a least-squares method. The reader is referred to Appendix B for the details.

### 3.1.2 Subdividing the surface.

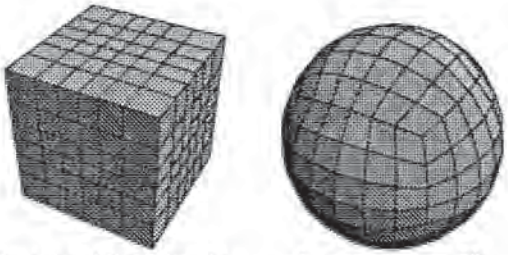
The Gauss map defines for each surface a partition of $\mathcal{S}^2$ into areas of various densities. To efficiently define an atlas of connected regions related to curvature information, we introduce the buckets induced by a homogeneous cover $\mathcal{C}$ of $\mathcal{S}^2$ as shown in figure 5: the *buckets* are the maximal connected regions of the surface, composed of faces which normal vectors belong to the same element of $\mathcal{C}$.

A connectivity graph $\mathcal{G}_B$ is built from the sets of buckets, adding an edge between two buckets when they have a common boundary (see figure 6).

### 3.1.3 Merging of buckets.

To keep control of the distortion when merging two adjacent regions of the subdivision, we define a notion of similarity on the set of buckets. To each bucket $\beta$ we compute

*Three coefficients because the curvature matrix is symmetric.



Squaring of the unit cube. Its projection on $\mathcal{S}^2$.

Fig. 5: Uniform cover of the sphere $\mathcal{S}^2$.

Fig. 6: The battered surface is represented by the buckets whose connectivity graph is on the right of the figure. One can see how this surface is simplified: only 6 average directions are kept from the initial surface.

the following information which will be used in defining the similarity:

- $N_\beta$: average of normal to faces in bucket $\beta$.
- $\Delta_\beta$: average of directions of maximal curvature.

Besides these geometric attributes, we use the connectivity graph $\mathcal{G}_\beta$ defined in the last subsection. Between two neighboring buckets $\beta_1$ and $\beta_2$ we introduce the similarity $\mathrm{d}(\beta_1, \beta_2)$:

$$\delta N = \frac{N_{\beta_1} - N_{\beta_2}}{||N_{\beta_1} - N_{\beta_2}||}$$

$$\mathrm{d}(\beta_1, \beta_2) = (1 - |\delta N \cdot \Delta_{\beta_1}|)^2 + (1 - |\delta N \cdot \Delta_{\beta_2}|)^2$$

An ultrametric (see [10]) on the set of regions is then defined by:

$$\mathrm{d}(\mathcal{R}_1, \mathcal{R}_2) = \min_{\beta_1 \in \mathcal{R}_1,\, \beta_2 \in \mathcal{R}_2} \mathrm{d}(\beta_1, \beta_2)$$

Given a threshold value $d$, we build a segmentation of the object by merging all region whose distances are below $d$. The edges of the connectivity graph are sorted with respect to their corresponding similarity values in a preprocessing phase in which we sort the edges of graph $\mathcal{G}_\beta$ according to d. The selection of the $n$ first edges on the connectivity graph will trigger $n$ successive unions of the adjacent regions associated with the edges. The user can then choose the number of charts in the atlas by selecting a number corresponding to the number of edges in $\mathcal{G}_\beta$.

### 3.1.4 Flattening the regions.

Once the segmentation is computed, the last step is to flatten the regions in order to define the charts. We use a region growing algorithm in which new faces are added in

such a way that the surface's geometry is not distorted too much and so that the final energy of flattened pieces not too high. We first pick up one face of each region, orthogonally flatten it with respect to its normal, and then depth-first traverse the faces graph from the initial facet. For each new vertex $M_i$ adjacent to two already fixed points, the location $m_i$ of $M_i$ in texture space is computed taking the average of the $A_{jk}(M_i)$ over all the triangles $M_iM_jM_k$ such that

1. $M_j$ and $M_k$ have the corresponding points $m_j$ and $m_k$ in texture plane and

2. $A_{jk}(M_i)$ is such that the following two triangles are similar:

- the triangle defined by the projection of $M_i$ $M_j$ and $M_k$ orthogonally w.r.t. the face $M_i$ $M_j$ $M_k$ and
- the triangle $(A_{jk}(M_i), m_j, m_k)$.

This atlas creation algorithm is useful if the surface is complex, but it may not be entirely satisfying. For this reason we provide the set of interactive functions to manipulate atlases described below.

## 3.2 An interactive tool.

As emphasized in [12], interactivity is important when texturing 3D shapes.

### 3.2.1 Drawing an atlas.

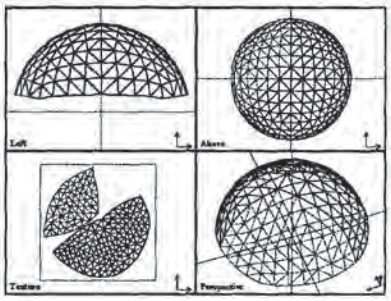


Fig. 7: Sphere portion associated with a two-charts atlas.

To visualize an atlas on the screen, we use two displays, one showing a wire frame projection of the polyhedral surface and the other showing the chart represented by the same network of polygons after the mapping transformation (see Figure 7). The chart is put in a special *Texture* view. The current chart and the selected points are highlighted in all the views.

### 3.2.2 Data structures.

To compute the texture mapping inside a face, the positions in the texture plane of all of its vertices are needed. Then, the rendering is computed using an algorithm similar to color calculation for Gouraud shading [11, 20]. Since local discontinuities are allowed along edges, a vertex may have as many 2-D positions as there are faces adjacent to it. Thus, an atlas depends on the location in the texture plane of the *angles* $(v_i, f_j)$ where $v_i$ is a vertex belonging to face $f_j$ (see Figure 8.a). To avoid redundancies, angles are regrouped in *sectors*: sets of connected angles for which the mapping function is continuous (see Figure 8.b). The mapping function is then defined by a position in the texture plane for each sector. All sectors belonging to the same vertex are stored in a linked list. Pointers to the list heads are stored in an array associated with the 3-D vertices of the surface. Access to a sector and data structure modifications can be computed in almost constant time.

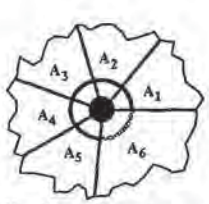



Fig. 8.a: A vertex on the 3-D polyhedron.

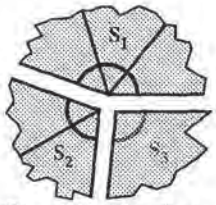



Fig. 8.b: The charts, involving three *sectors*.

Fig. 8: Case of a vertex corresponding to three *sectors* and six *angles*.

### 3.2.3 Interactive functions.

To modify an existing atlas, the user is provided with several types of interactive functions which operate in the texture plane.

**Positioning functions** let the user adjust the scale, stretch, angle and position of the whole atlas or of each chart using linear transformations. Finer-grained operations are obtained by selecting only a group of sectors. A function to align a set of sectors in the $x$ or $y$ direction is also provided. This simple function is very useful when one wants the edges of the charts to exactly match the border of a texture.

**The constraint function** marks sectors for the optimization procedure. Sectors can be fixed in $x$, $y$, or along both axes. With this function, one can adjust sectors in the mapped image so that a specific pattern lies precisely on a given place of the surface, and let all the rest of the texture be optimized.

**The cut function** defines discontinuities in the charts. Cuts can be seen as the snips of a tailor's scissors inside the piece of material that may stretch or shrink during the optimization procedure. The user gives a path along the edges which must not self-intersect. The chart is then possibly separated into several charts, or may be only internally cut.

**The merge function** reconnects charts along a given path. Faces connected do not necessarily belong to different charts. The user can either select the connection path, or pick two charts and let the program find the common edges and the best displacement to attach the first chart to the second.

**The optimization function** improves the charts, taking

into account the user specified constraints. Parameter $\alpha$ of the energy can be adjusted for special cases.

Note the difference between the constraint and the merge function: using the constraint function, one may fix sectors together corresponding to the adjacent path of two charts, producing the same visual result in a texture space as a merge between these two charts. But a call to the optimization function will lead to different results:

- when the charts have been merged, the optimization is performed on the resulting charts and globally reduces the distortion. The sectors belonging to the common path may have moved during this process.
- on the other hand, when the sectors of the common path have been fixed together, two optimization processes are performed independently, one for each chart, keeping the fixed sector in the same position in the texture space.

## 4 Applications in the field of animation.

The tool we have described has been built principally for static objects. Nevertheless, it appears that special effects in animation could be obtained very easily.

One example consists in moving and distorting some charts of the object. The result is a sliding texture onto a fixed shape. For example, one can draw a scrolling text onto any shape by simply translating down the chart of the object.

Another example involves interpolated objects. In this case, it is generally difficult to obtain a deformation that mimics an elastic deformation. Optimizing the mapping function with the criterion described previously can replace the 3-d elasticity system by a 2-d optimization which is simpler and faster. To obtain a realistic deformation of a textured object, one has to optimize the chart of the undeformed object, then apply any geometric transformation to it, and optimize again constraining the edge points not to move. This constraint corresponds to the fact that the piece of texture used for the object does not change with time. We made a short test animation in which the deformation was obtained by interpolating between key-objects, and it appeared that the mapping function could also be interpolated, thus requiring only a few optimizations, one for each key.

## 5 Remarks on Figures.

Let us add some comments on the figures appearing at the end of the paper:

- Figure 9 shows the different effects obtained using cylindrical and spherical projections of a checkerboard as a texture mapping onto the Utah teapot. We also display the same teapot textured with our tool, thus demonstrating automatic creation of an atlas and the use of interactive functions. Here the atlas is composed of four charts: the spout, the handle, the cap and the body of the teapot. The common boundaries of the cap and the body of the teapot are fixed in the texture plane to ensure visual continuity between the two charts. We see in this figure that the distortion is very low.
- The last two pictures illustrate the capabilities of atlases. Each shows geometrical objects whose atlases have been split into several charts (4 charts per object).

## 6 Conclusion.

We have presented a method for measuring the deformation energy of the mapping of an image onto a surface. The measure proposed here is an approximation of the integral of the Green-Lagrange deformation tensor. It can be minimized in real time and gives accurate results.

We have also addressed the problem of segmenting a 3D object in regions on which the mapping is not too distorted. We solved the problem by introducing the concept of an atlas together with interactive functions to edit and manipulate atlases and data structures which are efficient for these operations. We described a method which for any object automatically generates atlases, and we showed how to efficiently merge charts on an existing atlas. Efficient merging uses segmentation techniques based on curvature and ultrametrics. Specific data structures are proposed to handle atlases efficiently.

**Acknowledgements:** We wish to thank Alain Chesnais for revising an early version of this paper, Lars W. Ericson for carefully proof-reading the manuscript, Francis Lazarus, Arghyro Paouri, Marie-Luce Viaud and Jean-Luc De Antoni for their help during modeling, and the audiovisual department of INRIA for the photos and videos.

## References

[1] **C. Bennis, J.M. Vezien, G. Iglesias,** *Piecewise flattening for non-distorted texture mapping.* SIGGRAPH 91, Proc. of Computer Graphics, 25(4):237-246. July 1991.

[2] **E.Bier, K.Sloan,** *Two-part texture mapping.* IEEE Computer Graphics and applications, 40-53. September 1986.

[3] **J. F. Blinn and M. E. Newell,** *Texture and Reflection in Computer Generated Images.* Communications of the ACM, 19(10):542-547. October 1976.

[4] **J. Bloomenthal,** *Modeling the mighty maple.* SIGGRAPH 85, Proc. of Computer Graphics, 19(3):305-311. July 1985.

[5] **E. Catmull,** *A subdivision algorithm for computer display of curved surfaces.* Phd dissertation, University of Utah. December 1974.

[6] **P. Ciarlet,** *Mathematical Elasticity, Vol. I, 3-Dimensionnal Elasticity.* North Holland, 1988

[7] **R. M. Christensen,** *Mechanics of Composite Materials.* McGraw-Hill, 1967

[8] **F. Crow,** *Summed-area tables for texture mapping.* SIGGRAPH 84, Proc. of Computer Graphics, 18(3):207-212. July 1984.

[9] **M.P. Do Carmo,** *Differential Geometry of curves and surfaces.* Prentice-Hall, Inc. 1976.

[10] **E. Diday, J.C. Simon,** *Clustering Analysis.* Communication and Cybernetics, 10, Digital pattern recognition, 47-94. 1976.

[11] **J.D. Foley, A. van Dam, S.K. Feiner and J.F. Hughes,** *Computer Graphics, Principles and Practice, 2nd edition.* Addison-Wesley, 1990

[12] **P. Hanrahan, P. Haeberly,** *Direct WYSIWYG painting and texturing on 3D shapes.* SIGGRAPH 90, Proc. of Computer Graphics, 24(4):215-223. August 1990.

[13] **P. S. Heckbert,** *Survey of Texture Mapping.* IEEE Computer Graphics and Applications 6(11):56-67. November 1986.

[14] **M. Hirsch,** *Differential topology.* Graduate texts in mathematics 33, Springer-Verlag. 1976.

[15] **J. Maillot,** *Trois approches du plaquage de texture sur un objet tridimensionnel.* Thèse de doctorat en sciences, Université de Paris-Sud, Centre d'Orsay, may 1992.

[16] **S.D. Ma, H. Lin,** *Optimal texture mapping.* EUROGRAPHICS 88, 421-428. September 1988.

[17] **J. E. Marsden, T. J. R. Hughes,** *Mathematical Fundations of Elasticity.* Prentice Hall, 1983

[18] **W. H. Press, B. P. Flannery, S. A. Teukolsky and W. T. Vetterling,** *Numerical Recipes.* Cambridge University Press, Cambridge, 1986

[19] **P. Sander, S. Zucker,** *Inferring surface trace and differential structure from 3-D images.* Rapport de recherche No 1117, INRIA. 1989.

[20] *Graphic library, programming guide.* Silicon Graphics manual, IRIS 4D VGX series.

[21] **M. Spivak,** *A comprehensive introduction to differential geometry, Vol I.* Publish or perish, Inc., Berkeley. 1979.

## A Derivation of equation 5.

All norms on the vector space of $2 \times 2$ matrices being equivalent, we take the Euclidean norm. It is basis independent because an easy calculation shows that $||M|| = \mathrm{tr}({}^tM M)^{\frac{1}{2}}$, tr being the trace i.e. the sum of diagonal coefficients. Now

$$\begin{aligned} E &= \int\int_U \left\| \begin{pmatrix} \frac{\partial\phi}{\partial u}^2 - 1 & \frac{\partial\phi}{\partial u}\cdot\frac{\partial\phi}{\partial v} \\ \frac{\partial\phi}{\partial u}\cdot\frac{\partial\phi}{\partial v} & \frac{\partial\phi}{\partial v}^2 - 1 \end{pmatrix} \right\|^2 \mathrm{d}u\mathrm{d}v \\ &= \int\int_U \mathrm{tr}((I_\phi - \mathrm{Id})\,{}^t(I_\phi - \mathrm{Id}))\mathrm{d}u\mathrm{d}v \\ &= \int\int_U (\frac{\partial\phi}{\partial u}^2 - 1)^2 + 2(\frac{\partial\phi}{\partial u}\cdot\frac{\partial\phi}{\partial v})^2 + (\frac{\partial\phi}{\partial v}^2 - 1)^2 \mathrm{d}u\mathrm{d}v \end{aligned}$$

## B Computation of the curvature matrix.

Let $n_i$ and $s_i$ be the projected normals and vertices on $\mathcal{P}$, written in the same basis, $G$ being the origin. By the definition of $\mathcal{P}$, the projection of $N$ is the zero-vector. In a basis of the tangent plane, the curvature matrix is written as $C = \begin{pmatrix} a & b \\ b & c \end{pmatrix}$. We have to minimize:

$$A = \sum_i ||Cs_i - n_i||^2$$

We set $\sigma_x = \sum {s_i^x}^2$, $\sigma_y = \sum {s_i^y}^2$, $\sigma_{xy} = \sum s_i^x s_i^y$, and we get:

$$d = (\sigma_x + \sigma_y)\left(-{\sigma_{xy}}^2 + \sigma_x\sigma_y\right)$$

$$\begin{pmatrix} a \\ b \\ c \end{pmatrix} = \frac{1}{d} M_1 \cdot M_2$$

with

$$M_1 = \begin{pmatrix} -{\sigma_{xy}}^2 + \sigma_x\sigma_y + {\sigma_y}^2 & -\sigma_{xy}\sigma_y & {\sigma_{xy}}^2 \\ -\sigma_{xy}\sigma_y & \sigma_x\sigma_y & -\sigma_x\sigma_{xy} \\ {\sigma_{xy}}^2 & -\sigma_x\sigma_{xy} & {\sigma_x}^2 - {\sigma_{xy}}^2 + \sigma_x\sigma_y \end{pmatrix}$$

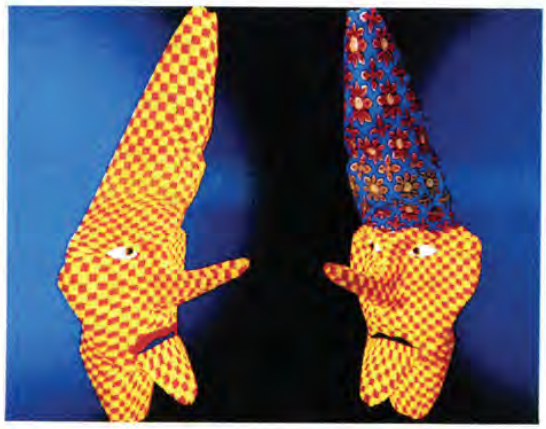
and $M_2 = \begin{pmatrix} \sum s_i^x n_i^x \\ \sum s_i^x n_i^y + s_i^y n_i^x \\ \sum s_i^y n_i^y \end{pmatrix}$

Eigenvectors $v$ and eigenvalues $\alpha$ are given by:

$$v_\pm = \begin{pmatrix} 2b \\ c - a + \Delta \end{pmatrix}, \quad \alpha_\pm = \frac{1}{2}(\mathrm{tr}(C) + \Delta)$$

$$\text{with: } \Delta = \pm \sqrt{\mathrm{tr}(C)^2 - 4\det(C)}$$

Fig. 9: Classical projections and optimized atlas.





# Efficient, Fair Interpolation using Catmull-Clark Surfaces

Mark Halstead* Michael Kass Tony DeRose†
Apple Computer, Inc.

## Abstract

We describe an efficient method for constructing a smooth surface that interpolates the vertices of a mesh of arbitrary topological type. Normal vectors can also be interpolated at an arbitrary subset of the vertices. The method improves on existing interpolation techniques in that it is fast, robust and general.

Our approach is to compute a control mesh whose Catmull-Clark subdivision surface interpolates the given data and minimizes a smoothness or "fairness" measure of the surface. Following Celniker and Gossard, the norm we use is based on a linear combination of thin-plate and membrane energies. Even though Catmull-Clark surfaces do not possess closed-form parametrizations, we show that the relevant properties of the surfaces can be computed efficiently and without approximation. In particular, we show that (1) simple, exact interpolation conditions can be derived, and (2) the fairness norm and its derivatives can be computed exactly, without resort to numerical integration.

**CR Categories and Subject Descriptors:** I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling - curve, surface, solid, and object representations; J.6 [Computer-Aided Engineering]: Computer-Aided Design (CAD); G.1.2 [Approximation]: Spline Approximation.

**Additional Key Words and Phrases:** Computer-aided geometric design, B-spline surfaces, subdivision surfaces, thin-plate splines.

## 1 Introduction

The construction of smooth interpolating surfaces is becoming increasingly important in a number of applications including statistical data modeling, interactive design, and scientific visualization. Typical input to an interpolating method is a collection of points to be interpolated, and a "mesh" that describes the connectivity of the points. Normal vectors are sometimes also specified at some or all of the data points.

If the shape to be modeled is a deformed plane, techniques from function approximation, such as Clough-Tocher interpolation [5], can be used. An advantage of the Clough-Tocher interpolant is that the construction is local, meaning that modification of a data point affects only a local portion of the surface. However, a drawback of Clough-Tocher interpolation is that there are typically remaining degrees of freedom not directly constrained by the data. These extra degrees of freedom are often set using local heuristics and typically result in surfaces that are not "fair", that is, surfaces having extraneous bumps and wiggles. Another serious drawback to Clough-Tocher interpolation, and indeed to any method that requires continuity of parametric derivatives (so-called parametric continuity), is the inability to model surfaces of arbitrary topological type (cf. Herron [8]). It is not possible, for instance, to model a sphere or a deformed sphere using a Clough-Tocher interpolant.

Celniker and Gossard [3] recently presented an interpolation method that extends Clough-Tocher interpolation by setting the remaining degrees of freedom so as to minimize a fairness norm. The fairness norm they use is quadratic, so it can be minimized by solving a (sparse) linear system. As a result, their method is fast enough for interactive design. However, being based on Clough-Tocher interpolants, their technique is not capable of describing surfaces of arbitrary genus.

A number of interpolation methods appropriate for surfaces of arbitrary genus have been developed in recent years. A survey of these can be found in Lounsbery et al. [10]. The method developed by Shirman and Séquin [14] is a generalization of Clough-Tocher interpolation to surfaces of arbitrary topology. The generalization is achieved by replacing parametric continuity with first order geometric continuity (continuity of tangent planes). Like Clough-Tocher interpolation, Shirman-Séquin interpolants have degrees of freedom not directly constrained by the data, and local heuristics for setting these degrees of freedom have fallen well short of producing fair surfaces (see Figure 4).

Last year Moreton and Séquin [11] presented a method capable of producing fair interpolating surfaces of arbitrary genus. They achieved this in much the same way as Celniker and Gossard by solving a minimization problem using finite elements. However, rather than using Clough-Tocher elements and a quadratic fairness norm, Moreton and Séquin used biquintic Bézier patches and a fairness norm based on

*Work done while a summer intern from the University of California, Berkeley.

†Work done while on sabbatical leave from the University of Washington.



© 1993 ACM-0-89791-601-8/93/008/0035 $01.50

intrinsic measures of curvature variation. The surfaces produced are the most impressive to date, but improved shape and arbitrary genus are obtained at the expense of dramatically increased running time. It appears that Moreton and Séquin's method is far too expensive for use in an interactive environment today (computation time is on the order of hours). Another shortcoming of their method is that it constructs surfaces that are only approximately tangent plane smooth since inter-patch continuity is modeled using a penalty function added to the fairness norm. Finally, their surfaces are only curvature continuous within each biquintic patch.

Here we present a scheme that combines the speed of Celniker and Gossard's method with the ability to model tangent plane continuous surfaces of arbitrary genus. We do this by using a quadratic fairness norm similar to the one used by Celniker and Gossard together with Catmull-Clark subdivision surfaces. We show that Catmull-Clark surfaces offer a number of advantages over previous methods based on piecewise polynomial elements; these include:

- They are curvature continuous everywhere except at a finite number of isolated "extraordinary" points.
- The high order of continuity is obtained with very few control points, meaning that the dimension of the space over which the optimizer must search is far lower for Catmull-Clark surfaces than for the method described by Moreton and Séquin.
- They reduce to traditional bicubic B-splines when the points to be interpolated form a regular rectangular grid. It should therefore be possible to more smoothly incorporate them into existing geometric modeling systems.

The use of Catmull-Clark surfaces presents some challenges, however. First, Catmull-Clark surfaces do not generally interpolate their control points, so to achieve interpolation, a system of interpolation constraints must be solved. The constraints relate the data points and normals to be interpolated with points and normals on the final surface. Formulating the interpolation constraints at first appears problematic for a Catmull-Clark surface because the surface is defined as the limit of an infinite number of subdivisions. We show that it is possible to derive *closed form* expressions for these constraints. A second challenge posed by Catmull-Clark surfaces is that efficient surface optimization depends on fast and reliable evaluation of the fairness norm and its derivatives. We show that it is possible to evaluate the fairness integral and its derivatives *exactly*, without resort to numerical integration, even though Catmull-Clark surfaces do not possess a closed form polynomial representation.

Figure 5 illustrates the basic idea of our approach. The original mesh is shown in the upper left. Subdividing it using Catmull-Clark subdivision results in the surface shown in the lower left. The surface approximates, but does not interpolate the vertices of the original mesh. By solving the system of interpolation constraints, we obtain a new mesh which is shown in the upper center. Subdividing the new mesh results in the surface in the lower center which does interpolate the vertices of the original mesh. Unfortunately, the direct application of the interpolation conditions to the mesh causes undesirable undulations in the surface. To combat this difficulty, we subdivide the mesh to add new degrees of freedom, and we set these new degrees of freedom to minimize a fairness measure subject to the interpolation constraints. The resulting mesh is shown in the upper right of Figure 5 and the corresponding subdivision surface is shown in the lower right. Note that minimizing the fairness measure removes the spurious undulations introduced by the direct application of the interpolation constraints.

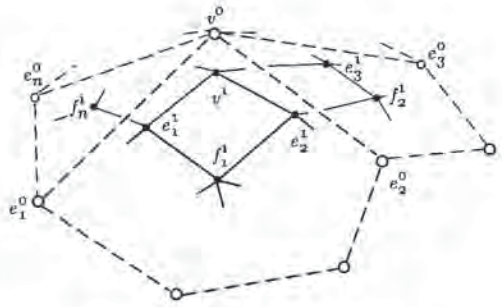



Figure 1: The situation around a vertex $v^0$ of order $n$.

The remainder of the paper is structured as follows. In Section 2 we provide some necessary background on subdivision surfaces in general, paying particular attention to Catmull-Clark surfaces. In Section 3, we derive the linear constraints on a Catmull-Clark mesh which guarantee that the surface interpolates given points and normals. We also show that applying these constraints directly to a mesh results in a surface which solves the interpolation conditions, but is unsatisfactory because of spurious wiggles. Then, in Section 4, we show how to reduce these artifacts by adding additional degrees of freedom through subdivision, and then setting them by optimizing a fairness norm based on the membrane/plate energy. Several implementation details along with performance statistics are provided in Section 5. In Section 6 we present a number of examples, and provide some comparisons to previous methods. Finally, in Section 7 we summarize our findings and describe several avenues of future research.

## 2 Subdivision Surfaces

In 1974 Chaikin [4] introduced the idea of generating a curve from a polygon by successively refining the polygon with the addition of new vertices and edges. In 1978, Catmull and Clark [2] and Doo and Sabin [6] generalized the idea to surfaces. In these schemes, an initial control mesh is refined by adding new vertices, faces and edges at each subdivision step. In the limit as the number of subdivision steps goes to infinity, the control mesh converges to a surface. With careful choice of the rules by which new vertices, edges and faces are introduced, it is possible to show that the limiting surface exists, is continuous, and possesses a continuous tangent plane. The Doo-Sabin subdivision rules generalize the subdivision rules for biquadratic B-splines, and the Catmull-Clark subdivision generalizes bicubic B-splines. An example of a Catmull-Clark surface of genus 3 is shown in Figure 3. A more recent method developed by Loop [9] generalizes quartic triangular B-splines. We focus on the Catmull-Clark scheme primarily because of the popularity of bicubic patches, however, much of the analysis we present is applicable to a wide class of subdivision schemes including those of Doo-Sabin and Loop.

When dealing with spline surfaces it is often helpful to maintain the distinction between global and local control meshes. By a local control mesh, we mean a subset of the

global mesh that influences a local region of the surface. Toward this end we use carets to denote global quantities.

Let $\widehat{M}^0$ denote the initial mesh, and let $\widehat{M}^i$ denote the mesh produced after $i$ applications of the Catmull-Clark subdivision step. To describe the $i+1$-st subdivision step, consider the neighborhood of a vertex $v^i$ of $\widehat{M}^i$ surrounded by $n$ edge points $e_1^i, ..., e_n^i$ and $n$ faces, as shown in Figure 1 for $i = 0$. Such a vertex is said to be of order $n$. As indicated in Figure 1, a new face point $f_1^{i+1}, ..., f_n^{i+1}$ is placed at the centroid of each face of $\widehat{M}^i$. Each new edge point $e_1^{i+1}, ..., e_n^{i+1}$ is then computed by taking an average of surrounding points. Specifically,

$$e_j^{i+1} = \frac{v^i + e_j^i + f_{j-1}^{i+1} + f_j^{i+1}}{4},$$

where subscripts are to be taken modulo $n$. Finally, a new vertex point $v^{i+1}$ is computed as

$$v^{i+1} = \frac{n-2}{n} v^i + \frac{1}{n^2} \sum_j e_j^i + \frac{1}{n^2} \sum_j f_j^{i+1}.$$

The Catmull-Clark subdivision process is such that:

- The surfaces can be of arbitrary genus since the subdivision rules can be carried out on a mesh of arbitrary topological type.
- After the first subdivision step all faces are quadrilaterals.
- Except at extraordinary vertices (vertices of order $n \neq 4$) the limiting surface can be shown to converge to a bicubic B-spline. The surface is therefore curvature continuous except at extraordinary vertices.
- The number of extraordinary vertices is fixed, and is equal to the number of extraordinary vertices in $\widehat{M}^1$, the mesh produced after the first subdivision step.
- Near an extraordinary vertex the surface does not possess a closed form parametrization; it consists of an infinite number of bicubic patches that converge to a limit point. The surface can be shown to have a well defined tangent plane at the limit point, but the curvature there is generally not well defined [1].

# 3 Interpolation using Subdivision Surfaces

Given a mesh $\widehat{I}$ of arbitrary topological type, the idea is to generate a control mesh $\widehat{M}^0$ such that the subdivision surface it defines interpolates some or all of the vertices of $\widehat{I}$. It is also possible to constrain the surface to have a specified normal at each interpolation point.

Nasri [12] generates interpolating surfaces using the biquadratic formulation of Doo and Sabin [6]. Like biquadratic B-splines, Doo-Sabin surfaces interpolate the centroid of each face in the control mesh. Thus a linear constraint on the control vertices can be generated for each interpolation point and the resultant system solved for the desired control mesh[1]. It appears that Nasri had no simple formulation for the surface normal at the centroid, and so was unable to specify normals at these points.

[1]Although Nasri does not mention it, it is possible for the coefficient matrix in the linear system to be singular.

To generate interpolating surfaces for other subdivision schemes we need a method of determining the position and normal at a set of points on the limit surface. Because the surface is the result of repeated application of a subdivision step, we can analyze the behavior of a small neighborhood of points as they converge to the limit surface in order to determine the surface properties at the point of convergence.

## 3.1 Interpolation Conditions

After one subdivision step there arises an arrangement of vertices that persists (i.e. the same topology will be observable) for any number of subsequent subdivisions. To analyze the limiting behavior of the surface near a vertex it is therefore convenient to introduce a matrix that describes the subdivision process locally, that is, in the neighborhood of the vertex [6]. It is not necessary to compute local subdivision matrices in practice; they are simply tools used to derive formulas describing the limiting behavior of the surface.

Let $v^i$ be a vertex of order $n$ of the mesh $\widehat{M}^i$, let $V_n^i = (v^i, e_1^i, ..., e_n^i, f_1^i, ..., f_n^i)^T$ be the column vector of vertices in the neighborhood of $v^i$, and let $V_n^{i+1}$ be the corresponding column vector of points in the neighborhood after subdivision. Since the points in $V_n^{i+1}$ are computed by linear combinations of the points in $V_n^i$, we can use a square matrix $\mathbf{S}_n$ to express the subdivision:

$$V_n^{i+1} = \mathbf{S}_n V_n^i.$$

For instance, for Catmull-Clark surfaces the matrix $\mathbf{S}_4$ is

$$\mathbf{S}_4 = \frac{1}{16} * \begin{pmatrix} 9 & \frac{3}{2} & \frac{3}{2} & \frac{3}{2} & \frac{3}{2} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ 6 & 6 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 6 & 1 & 6 & 1 & 0 & 1 & 1 & 0 & 0 \\ 6 & 0 & 1 & 6 & 1 & 0 & 1 & 1 & 0 \\ 6 & 1 & 0 & 1 & 6 & 0 & 0 & 1 & 1 \\ 4 & 4 & 4 & 0 & 0 & 4 & 0 & 0 & 0 \\ 4 & 0 & 4 & 4 & 0 & 0 & 4 & 0 & 0 \\ 4 & 0 & 0 & 4 & 4 & 0 & 0 & 4 & 0 \\ 4 & 4 & 0 & 0 & 4 & 0 & 0 & 0 & 4 \end{pmatrix}.$$

Repeated subdivision is expressed by repeated multiplication and hence powers of $\mathbf{S}_n$, so

$$V_n^{i+1} = \mathbf{S}_n^i V_n^1.$$

The properties of the limit surface will be governed by the properties of $V_n^{i+1}$ as $i$ approaches infinity. Since $V_n^{i+1}$ is the image of $V_n^1$ under $\mathbf{S}_n^i$, the eigenstructure of $\mathbf{S}_n$ naturally plays a key role.

In Appendix A we analyze the behavior of the limit surface in terms of the matrix $\mathbf{S}_n$ by building on the analytical techniques of Doo and Sabin [6] and Ball and Storry [1]. Like Loop [9], we find that the positions and normals of the limit surface can be expressed explicitly in terms of the vertices of the control mesh. However, whereas Loop's analysis was peculiar to his subdivision surfaces, our analysis applies to any subdivision scheme whose local matrix $\mathbf{S}_n$ satisfies the conditions listed in Appendix A. In particular, our analysis exposes the following simple dependence between the left eigenvectors of $\mathbf{S}_n$ and limit points and normals.

Let $\lambda_1 \geq \lambda_2 \geq \lambda_3$ be the three largest eigenvalues of $\mathbf{S}_n$ and let $l_1, l_2, l_3$ be the corresponding left eigenvectors. In Appendix A we show that a point $v^1$ having a neighborhood $V_n^1$ converges to the point

$$v^\infty = l_1 \cdot V_n^1 \qquad (1)$$

and the normal vector to the surface at $v^\infty$ is given by

$$N^\infty = c_2 \times c_3 \tag{2}$$

where $c_2 = l_2 \cdot V_n^1$ and $c_3 = l_3 \cdot V_n^1$, and where "$\times$" denotes vector cross product. Explicit formulas for $l_1, l_2$ and $l_3$ for Catmull-Clark surfaces can be found in Appendix A.

Equation 1 provides an interpolation condition that is linear in the control points of $V_n^1$, but Equation 2 at first appears to impose a quadratic constraint on $V_n^1$'s control points. Fortunately, we can require a surface to have a given normal vector $N$, using the following two linear constraints:

$$N \cdot c_2 = 0 \qquad \text{and} \qquad N \cdot c_3 = 0 \tag{3}$$

In addition to providing interpolation constraints, the limit point and normal vector formulas can also be used to compute exact points and normal vectors on the surface for use during rendering [9]. The color images (Figures 3 through 7) have all been computed this way.

### 3.2 Solving the Interpolation Problem

Ignoring the interpolation of normals for the time being, we can use the interpolation condition in Equation 1 to compute a control mesh $\widehat{M}^0$ with the property that the subdivision surface it defines interpolates the vertices of a given mesh $\widehat{I}$. It is natural to do this by selecting $\widehat{M}^0$ to have the same mesh topology as $\widehat{I}$, that is, the same number and connectivity of vertices, faces, and edges. This approach leads to a square linear system of the form

$$\mathbf{A}x = b \tag{4}$$

where $x$ is the column vector of the unknown vertex coordinates in $\widehat{M}^0$, and $b$ is the corresponding column vector of vertex coordinates of $\widehat{I}$. The rows of the square matrix $\mathbf{A}$ are determined by the interpolation conditions and mesh topology. In some cases, the matrix $\mathbf{A}$ is singular, so we use a least-squares solution to Equation 4. An example is shown in Figure 5. The original mesh is shown in the upper left. Subdividing it according to the usual Catmull-Clark rules results in the lower-left surface which approximates, but does not interpolate the vertices of the original mesh. By solving Equation 4, we obtain a new mesh which is shown in the upper center. Subdividing the new mesh according to the usual Catmull-Clark rules gives the surface in the lower center which does interpolate the vertices of the original mesh.

## 4 Fairing

The surface in the lower center of Figure 5 is curvature continuous almost everywhere and interpolates the vertices of the original mesh. Nonetheless, for many purposes it is an unsatisfactory interpolating surface because of its excessive undulations. These undulations appear to be artifacts of the interpolation process since they are not indicated by the shape of the original mesh. For example, the surface has a number of concavities where the original mesh is convex. Note that some of the undulations are present in the ordinary approximating Catmull-Clark surface, but they have become more severe and objectionable in the interpolating surface. This difference is typical of interpolating and approximating surfaces.

Nothing in our formulation of the interpolation conditions in Section 3 prohibits or discourages undulations in the surface, so this type of behaviour should not be surprising. In order to improve the quality of the interpolant, we introduce additional degrees of freedom into the surface by subdivision, and then set the degrees of freedom by optimizing a fairness norm on the surface subject to a set of linear constraints given by the interpolation conditions.

### 4.1 Evaluating the Fairness Norm

Celniker and Gossard [3] were able to improve the quality of interpolating surfaces using a fairness norm based on a linear combination of the energy of a membrane and a thin plate. Without any fundamental changes, the norm can be given directional preferences and nonuniform weighting over the surface, but for clarity of presentation, we consider the isotropic uniform case:

$$E(W) = \alpha E_m(W) + \beta E_p(W) \tag{5}$$

where $E_m(W)$ and $E_p(W)$ denote the membrane and thin-plate energies respectively:

$$\begin{aligned} E_m(W) &= \int\int \|W_u\|^2 + \|W_v\|^2 \, du\, dv \\ E_p(W) &= \int\int \|W_{uu}\|^2 + 2\|W_{uv}\|^2 + \|W_{vv}\|^2 \, du\, dv, \end{aligned}$$

and where $W(u,v) = (x(u,v), y(u,v), z(u,v))$ is a parametric representation of the surface, where subscripts on $W$ represent parametric derivatives, and where $\alpha$ and $\beta$ are freely selectable weights.

Since the membrane/plate norm is defined in terms of a parametric representation of the surface, it cannot be directly applied to Catmull-Clark surfaces since in general they have no "natural" parametrization near extraordinary points. The remainder of this section describes how we extend the definition of the norm in a way that can be used with Catmull-Clark surfaces. As we show below, the extended norm will be constructed to be quadratic in the control points of the mesh. The optimization can consequently be performed quickly without iteration by solving a linear system. Moreover, there is a unique minimum since the Hessian of the norm is symmetric and positive definite.

The membrane/plate norm can be evaluated without modification on a bicubic patch $W$ as follows. First, we note that the norm can be written as $E = E_x + E_y + E_z$, where $E_x$ depends only on the $x$ component of $W$, $E_y$ only on the $y$ component and $E_z$ only on the $z$ component of $W$. Let $P_x$ be a 16-element column vector of positions of the $x$ coordinates of the control points $W$. Figure 2(a) schematically depicts a 16 element control net and the bicubic patch it defines. The $x$ component of the fairness norm for the patch can be expressed as

$$E_x = P_x^T \cdot \mathbf{K} \cdot P_x \tag{6}$$

where the entries of the $16 \times 16$ matrix $\mathbf{K}$ can be computed exactly from the integrals in Equation 5 for bicubic B-spline basis functions. Similar formulas hold for the $y$ and $z$ components.

Figure 2(b) depicts a mesh that includes an extraordinary point. The region of the limit surface corresponding to the central face in the mesh is shown at the center bottom, but the limit surface is not in general a parametric polynomial, so we cannot directly apply the membrane/plate norm used above for a bicubic mesh. However, we can subdivide the

mesh in Figure 2(b) to obtain the mesh in (c). After subdivision, the limit surface is divided into four subpatches. Three of these subpatches (shown shaded in (c)) are bicubic B-splines, so on these patches we can in principle evaluate the fairness norm exactly. By repeating this procedure we can write an infinite series for the fairness norm of the original extraordinary patch of Figure 2(b). In order to fully define the series, we must choose a parametrization for each of the B-spline subpatches during subdivision. Unfortunately, the most straightforward way to assign the parametrizations causes the infinite series for the thin plate energy to diverge (see Appendix B).

There are several methods that could be applied to overcome the problem of the divergent series. For instance, we might try to find an alternate method of parametrizing the subpatches that leads to convergent sequences. We are currently investigating this possibility, but we have found that the following method gives good results. Intuitively, we intend to modify the thin plate energy so that it integrates to zero for surface patches defined by planar and "regular" control meshes. For a bicubic mesh it is relatively clear that a regular mesh is one that is an affine image of Figure 2(a) since such a mesh has vanishing second derivatives. As shown in Appendix B, it is possible to generalize the notion of regularity for meshes containing an extraordinary vertex. It is also possible to measure the deviation of an arbitrary mesh of control points $P$ from it's regular component $P'$. We therefore define the modified thin plate energy of $P$ to be the thin plate energy of $P - P'$. In symbols, the norm we use can be written as

$$E(P) = \alpha E_m(P) + \beta E_p(P - P'). \tag{7}$$

We have written this norm as a function of the control mesh $P$ rather than the limit surface that $P$ defines. This is to emphasize that the norm is not, strictly speaking, a property of the limit surface. It is more appropriate to think of Equation 7 as a norm on meshes, because it is not generally the case that $E(P^i) = E(P^{i+1})$ where $P^i$ and $P^{i+1}$ denote the mesh after $i$ and $i+1$ subdivisions. Although this might be considered a theoretical deficiency, it has posed no difficulties in practice.

Using the modified norm, the infinite series is a convergent geometric series, so we can express its limiting value analytically. Appendix B contains the relevant details, but the result is that we can exactly compute the entries of a new quadratic form $\mathbf{K}_n$ that can be applied around an extraordinary vertex of order $n$.

Now that we have defined the local fairness norm for patches surrounding extraordinary patches, we define the global fairness norm as the sum of the fairness norms over each of the patches using the standard membrane/plate norm for bicubic patches and the modified norm of Equation 7 for extraordinary patches. We can write the global fairness norm as $\widehat{P}^T\widehat{\mathbf{K}}\widehat{P}$ where $\widehat{\mathbf{K}}$ is a sparse matrix obtained from the various $\mathbf{K}_n$ by iterating over the individual vertices and collecting the entries into a global system, and where $\widehat{P}$ is a column vector containing the $x, y$ and $z$ coordinates of the control vertices in the global mesh $\widehat{M}^0$.

## 4.2 Minimizing the Fairness Norm

Since we have a global expression for the fairness norm, we are now in a position to express and solve the minimization problem. Given a mesh $\widehat{I}$ with $t$ vertices, $r$ of which are constrained to have a specified limit point and $s$ of which are constrained to have specified normals, we seek the vector of $3t$ vertex coordinates $\widehat{P}$ such that the limit surface satisfies the $3r + 2s$ linear interpolation conditions and the fairness norm $\widehat{P}^T\widehat{\mathbf{K}}\widehat{P}$ is minimized over all possible $\widehat{P}$.[2] Because the constraints are linear and the norm is quadratic in the unknowns, this problem can be solved directly without iteration.

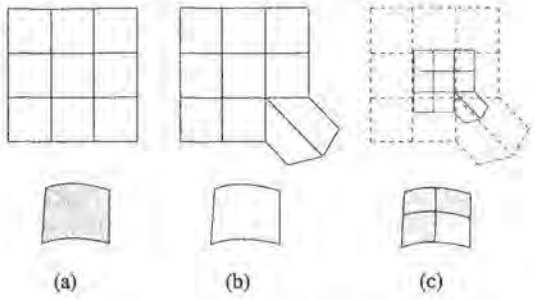



Figure 2: (a) A regular control mesh (above) which generates a bicubic B-spline patch on the limit surface (below). (b) A control mesh with an extraordinary point (above), and the extraordinary surface patch it defines (below). (c) The control mesh after one subdivision (above), and the four subpatches after subdivision (below). The three bicubic subpatches are shaded gray, and the remaining extraordinary subpatch is shaded white.

If we have only positional constraints, the $x, y$ and $z$ components of the mesh are independent, so the whole problem decouples into three completely independent optimizations, one for each component of the mesh. If normal vectors are to be interpolated, the $x, y$ and $z$ components of the mesh are no longer independent, so the problem must be solved as a single optimization. Even so, the $x, y$ and $z$ components of the mesh remain nearly decoupled (in the sense that the linear system is block diagonal except for a few off-diagonal terms) and sparse matrix methods exist that can exploit this fact[7].

The $r$ position constraints and $s$ normal constraints on the $t$ mesh points can be represented by the equation $\mathbf{B}\widehat{P} = D$ where $\mathbf{B}$ is a $(3r+2s) \times 3t$ matrix and $D$ is a vector of length $3r+2s$. Let $\mathbf{C}$ be the $3t$ by $l$ matrix whose columns span the null space of $\mathbf{B}$ and let $\widehat{P}_0$ be any vector satisfying $\mathbf{B}\widehat{P}_0 = D$. Then all $\widehat{P}$ which satisfy the interpolation constraints can be written in the form $\widehat{P}_0 + \mathbf{C}R$ for some $l$-vector $R$. Therefore we wish to find the vector $R$ that minimizes:

$$(\widehat{P}_0 + \mathbf{C}R)^T\,\widehat{\mathbf{K}}\,(\widehat{P}_0 + \mathbf{C}R) = R^T\mathbf{C}^T\widehat{\mathbf{K}}\mathbf{C}R + 2R^T\mathbf{C}^T\widehat{\mathbf{K}}\widehat{P}_0 + \widehat{P}_0^T\widehat{\mathbf{K}}\widehat{P}_0.$$

$\widehat{\mathbf{K}}$ is symmetric and positive definite, so $R$ is found by setting the gradient of this function to zero:

$$\mathbf{C}^T\widehat{\mathbf{K}}\mathbf{C}R + \mathbf{C}^T\widehat{\mathbf{K}}\widehat{P}_0 = 0. \tag{8}$$

[2] Each of the $t$ vertices has three coordinates, so the total number of unknowns is $3t$. Each position interpolation constraint imposes three conditions, one per coordinate, and each normal vector constraint imposes the two conditions in Equation 3.

# 5 Implementation

For simplicity and speed, our current implementation of the fairing process uses only positional constraints and exploits the fact that the linear systems for $x$, $y$ and $z$ decouple in this case. As a result, the implementation is able to compute the minimum energy mesh by solving three linear systems, each involving one third as many variables as Equation 8. To further speed the computation, each of these systems is solved using sparse-matrix methods.

Given a mesh $\widehat{I}$ whose vertices are to be interpolated, we must first choose the structure of the mesh $\widehat{M}^0$ whose vertices we compute. Our current implementation chooses $\widehat{M}^0$ to have the structure that would result from subdividing $\widehat{I}$ twice. This choice has two benefits. First, it adds enough extra degrees of freedom for the fairing to be effective. Second, it places enough new vertices between the interpolation points to ensure that the interpolation conditions for all vertices of $\widehat{I}$ are independent, making the construction of a sparse representation of the required null space easy.

Since we are considering a single component $x$, $y$ or $z$ at a time and not allowing normal constraints, we can still write the interpolation conditions as $\mathbf{B}\widehat{P} = D$ but now $\mathbf{B}$ is an $r \times t$ matrix and $D$ is a vector of length $r$. We compute a sparse set of null-space vectors for $\mathbf{B}$ as follows. Suppose the $i$th row of $\mathbf{B}$ has $k$ non-zero entries in columns $(a_1, a_2, \ldots, a_k)$. Because of the way the positional constraints decouple after two subdivisions, all other entries of $\mathbf{B}$ in those $k$ columns are zero. As a result, it is an easy matter to find $k-1$ independent null-space column vectors which are zero except in rows $(a_1, a_2, \ldots, a_k)$. Collecting these for each row of $\mathbf{B}$ yields a collection of sparse vectors that completely span the null space of $\mathbf{B}$ unless $\mathbf{B}$ contains zero columns. If $(b_1, b_2, \ldots, b_m)$ are the zero columns of $\mathbf{B}$, we complete the null space by adding the $m$ vectors $Q_s$, $1 \le s \le m$ where $Q_s$ is one in the $b_s$th entry and zero elsewhere.

In addition to the null space, we need a feasible mesh $\widehat{P}_0$ which satisfies the constraints. We construct this mesh as follows. For each row $i$ in $\mathbf{B}$, with non-zero entries in columns $(a_1, a_2, \ldots, a_k)$, set the entries of $\widehat{P}_0$ at indices $(a_1, a_2, \ldots, a_k)$ to $D_i$ and set any remaining entries of $\widehat{P}_0$ to zero. Then since all the rows of $\mathbf{B}$ sum to one, the resulting $\widehat{P}_0$ will solve the equation $\mathbf{B}\widehat{P}_0 = D$.

Finally, given the null space basis $\mathbf{C}$ and the feasible mesh $\widehat{P}_0$, we compute the minimum energy mesh by solving Equation 8 three times using sparse LU decomposition, once for each component of the mesh. If the mesh is a regular square grid, the bandwidth of the linear system will be $O(\sqrt{n})$, and the linear system will take $O(n^2)$ time to solve. The running time is more difficult to analyze for general meshes, but the times we have observed to date are consistent with $O(n^2)$ performance.

# 6 Results

Figure 5 shows the complete process of interpolation and fairing. The original mesh is shown in the top left. The interpolating mesh is shown at top center. The faired, interpolating mesh is shown at top right. Below each mesh is the corresponding Catmull-Clark limit surface. Note that the spurious undulations in the interpolating limit surface are greatly reduced in the faired interpolating surface. The additional subdivisions in the faired interpolating mesh provide the degrees of freedom necessary to do this. For the examples presented in this paper, we set $\alpha = 0$ and $\beta = 1$.

Often it is desirable to fair only a local region of the surface, either to have more control over the fairing or because the number of vertices in the control mesh is large. In this case we select a subset of control vertices that are free to move and compute the solution to the constrained minimization over the surface patches affected by this set. Figure 6 illustrates this process. The user has selected a subset of 52 vertices that are allowed to vary during the minimization process. These vertices are highlighted in red. Other nearby vertices which influence the minimization, but are not allowed to change, are shown in magenta. After fairing, the undulations in the faired region have been reduced, but they persist in the unfaired regions. In this case, the fairing took .18 seconds on an SGI Crimson workstation.

Lounsbery et al. [10] have done a survey of the previously published interpolation methods and found that existing local interpolation schemes do an unsatisfactory job of constructing fair surfaces, even for the simple cases such a data sampled from a torus. To facilitate comparison with these methods, we have run our algorithm and a representative local interpolant, that of Shirman and Séquin [14], on the same coarsely sampled toroidal data set. The results are shown in Figure 4. The upper left shows the original mesh used as input for the interpolants. The upper right shows the surface produced by the Shirman-Séquin algorithm. The odd looking specular highlights in the Shirman-Séquin interpolant point out some interpolation artifacts which are typical of local methods. Global methods tend to have a different appearance. The surface in the lower left of Figure 4 is a Catmull-Clark surface that interpolates the original mesh using the methods of Section 3. This surface has different (lower frequency) artifacts than the Shirman-Séquin interpolant, but they are nonetheless objectionable. The surface in the lower right is an interpolating faired surface computed using our method. The surface has no visible artifacts, an observation confirmed by examining the surface from other viewpoints. The implementation took 36.5 seconds to fair the entire 600 point mesh at once on an SGI Crimson workstation.

The result of applying the interpolation algorithm to a more complicated model is shown in Figure 7. The original mesh is shown at the far left. The left center shows the ordinary approximating Catmull-Clark surface. Note the artifacts throughout the stem and where the stem meets the base. These artifacts are accentuated in the interpolating Catmull-Clark surface shown in the right center. In addition, the interpolating surface shows severe overshoot at the bottom of the stem. This type of overshoot is typical of interpolation without fairing. The far right shows the faired interpolating Catmull-Clark surface computed using our method. The artifacts along the stem and where the stem joins the base have been removed. Fairing the 1273 point mesh took 127.8 seconds on an SGI Crimson workstation.

# 7 Conclusions

We have described an efficient method for constructing fair surfaces that interpolate the vertices of a mesh of arbitrary topological type; normal vectors can also be interpolated at an arbitrary subset of the vertices. Our approach is to compute a control mesh describing a Catmull-Clark surface that interpolates the given data and minimizes a quadratic

norm that combines thin plate and membrane energies.

Our method improves on previous techniques by combining many of the strengths of the methods described by Celniker and Gossard and by Moreton and Séquin. Like Celniker and Gossard, we use a quadratic norm to achieve practical fairing at interactive rates. Like Moreton and Séquin, we use a representation capable of modeling arbitrary topological surfaces. In addition, the Catmull-Clark representation we use provides improved surface continuity with remarkably few degrees of freedom. More specifically, Celniker-Gossard surfaces meet with only tangent plane continuity along patch boundaries, and those of Moreton-Séquin meet with only approximate tangent plane continuity. Our surfaces, in contrast, are curvature continuous everywhere except at a finite number of isolated points.

Our work also provides two new analytical tools for analyzing and manipulating subdivision surfaces: limit point and normal vector analysis based on left eigenvectors of the local subdivision matrix, and a method for developing exact formulas for evaluating quadratic membrane/plate functionals and their derivatives.

As a topic for future research, we plan to investigate using the surfaces produced by our method as a starting point for minimizing the intrinsic "MVS" norm developed by Moreton and Séquin. We are also interested in developing subdivision schemes that are curvature continous everywhere.

## References


[1] A. A. Ball and J. T. Storry. Conditions for tangent plane continuity over recursively defined B-spline surfaces. *ACM Transactions on Graphics*, 7(2):83–102, April 1988.

[2] E. Catmull and J. Clark. Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer Aided Design*, 10(6):350–355, 1978.

[3] George Celniker and Dave Gossard. Deformable curve and surface finite elements for free-form shape design. In *Proceedings of SIGGRAPH '91*, pages 257–265, July 1991.

[4] G. Chaikin. An algorithm for high speed curve generation. *Computer Graphics and Image Processing*, 3:346–349, 1974.

[5] R. Clough and J. Tocher. Finite element stiffness matrices for analysis of plate bending. In *Matrix Methods in Structural Mechanics (Proceedings of the conference held at Wright-Patterson Air Force Base, Ohio, 26-28 October 1965)*, pages 515–545, 1966.

[6] D. Doo and M. Sabin. Behaviour of recursive division surfaces near extraordinary points. *Computer Aided Design*, 10(6):356–360, 1978.

[7] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, 2nd edition, 1989.

[8] G. Herron. Techniques for visual continuity. In G. Farin, editor, *Geometric Modeling*, pages 163–174. SIAM, 1987.

[9] Charles T. Loop. Smooth subdivision surfaces based on triangles. M.S. Thesis, Department of Mathematics, University of Utah, August 1987.

[10] Michael Lounsbery, Stephen Mann, and Tony DeRose. Parametric surface interpolation. *IEEE Computer Graphics and Applications*, 12(5):45–52, September 1992.

[11] Henry P. Moreton and Carlo Séquin. Functional optimization for fair surface design. In *Proceedings of SIGGRAPH '92*, pages 167–176, July 1992.

[12] Ahmad H. Nasri. Polyhedral subdivision methods for free-form surfaces. *ACM Transactions on Graphics*, 6(1):29–73, January 1987.

[13] Malcolm Sabin. Recursive division singular points. Unpublished manuscript, June 1992.

[14] L. Shirman and C. Séquin. Local surface interpolation with Bézier patches. *Computer Aided Geometric Design*, 4(4):279–296, 1988.


# Appendix

## A Properties of the Limit Surface

To develop formulas for limit points and normals on subdivision surfaces, we examine the eigenstructure of the local subdivision matrix $\mathbf{S}_n$ associated with the subdivision scheme. (Some of the following analysis appears to have been developed independently by Sabin [13].)

Let $m = 2n + 1$ denote the size of $\mathbf{S}_n$, and let $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_m$ denote the eigenvalues of $\mathbf{S}_n$ with corresponding right eigenvectors $r_1, \ldots, r_m$ and left eigenvectors $l_1, \ldots, l_m$. If $\mathbf{S}_n$ is not defective, the right eigenvectors form a basis, and the left eigenvectors can be chosen so that (cf. Golub and Van Loan [7])

$$l_k \cdot r_j = \delta_{kj}. \tag{9}$$

Thus, assuming that $\mathbf{S}_n$ is not defective, the neighborhood $V_n^1$ can be expanded uniquely as

$$V_n^1 = c_1 r_1 + \cdots + c_m r_m \tag{10}$$

where the $c$'s are geometric position vectors and where the $r$'s are column vectors of scalars. The $c_k$, $k = 1, \ldots, m$ can be determined by dotting both sides of Equation 10 with $l_k$ and using Equation 9:

$$l_k \cdot V_n^1 = c_1 l_k \cdot r_1 + \cdots + c_k l_k \cdot r_k + \cdots + c_m l_k \cdot r_m = c_k. \tag{11}$$

Using this expansion of $V_n^1$,

$$V_n^i = \mathbf{S}_n^i V_n^1 = \lambda_1^i c_1 r_1 + \cdots + \lambda_m^i c_m r_m.$$

For a non-trivial limit to exist as $i \to \infty$, it is necessary for the magnitude of the largest eigenvalue $\lambda_1$ to be 1. In this case,

$$V_n^\infty := \lim_{i \to \infty} V_n^i = c_1 r_1 = \left(l_1 \cdot V_n^1\right) r_1$$

For a subdvision scheme to be affine invariant (that is, independent of the coordinate system in which the calculation is performed), the points of $\widehat{M}^{i+1}$ must be affine combinations of the points in $\widehat{M}^i$, meaning that each of the rows of $\mathbf{S}_n$ must sum to one. In matrix form:

$$\mathbf{S}_n (1, \ldots, 1)^T = (1, \ldots, 1)^T.$$

In other words, the column vector of 1's is the eigenvector $r_1$ associated with eigenvalue 1. Since $r_1$ is a column vector of 1's, every point in the neighborhood converges to the point

$$c_1 = l_1 \cdot V_n^1 \tag{12}$$

on the limit surface. Stated more formally, we have proven that:

**Proposition 1:** A point $v^1$ of $\widehat{M}^1$ with neighborhood $V_n^1$ and local subdivision matrix $\mathbf{S}_n$, converges to the point

$$v^\infty = l_1 \cdot V_n^1$$

on the limit surface where $l_1$ is the left eigenvector of $S_n$ associated with eigenvalue 1, assuming that $S_n$ satisfies the following conditions:

i) $\mathbf{S}_n$ is not defective.
ii) $\mathbf{S}_n$ describes an affine invariant process.
iii) The magnitude of the largest eigenvalue is 1 and it has multiplicity 1.

Using a discrete Fourier analysis similar to the one described by Ball and Storry [1], one can show that for Catmull-Clark surfaces the above conditions on $\mathbf{S}_n$ hold and that

$$l_1 = \frac{1}{n(n+5)}(n^2, 4, ..., 4, 1, ..., 1),$$

meaning that

$$v^\infty = \frac{n^2 v^1 + 4\sum_j e_j^1 + \sum_j f_j^1}{n(n+5)}. \qquad (13)$$

Equation 13 can be used as an interpolation condition on the points of $\widehat{M}^1$ by setting $v^\infty$ to a point to be interpolated. Note that the interpolation conditions are on the vertices of $\widehat{M}^1$, not on the vertices of the initial control mesh $\widehat{M}^0$, since the analysis above requires that each face has exactly four edges. This apparent restriction poses no problem in practice since fairing requires the extra degrees of freedom present in $\widehat{M}^1$.

To develop an interpolation condition on normal vectors, we must determine the normal vector (if it exists) to the limit surface at $v^\infty$. This normal vector can be simply computed from the eigenstructure of $\mathbf{S}_n$, as indicated by the following proposition.

**Proposition 2:** The normal vector to a subdivision surface at a limit point $v^\infty$ corresponding to a vertex $v^1$ whose neighborhood is $\widehat{M}_n^1$ is the vector

$$N^\infty = c_2 \times c_3$$

where $c_2 = l_2 \cdot \widehat{M}_n^1$ and $c_3 = l_3 \cdot \widehat{M}_n^1$, assuming that the local subdivision matrix $\mathbf{S}_n$ satisfies the conditions of Proposition 1 in addition to:

iv) The eigenvalues $\lambda_1 = 1 \geq \lambda_2 ...$ are such that $\lambda_2 = \lambda_3 > \lambda_4$.

*Proof sketch:* The general idea behind the proof is to show that there is a common plane to which all points in the neighborhood are converging. The vector $N^\infty$ will then be chosen to be perpendicular to this plane. Let $u_j^i$ denote the vector from $v^\infty$ to the $j$-th point $p_j^i$ of the neighborhood $\widehat{M}^i$. Roughly speaking, if a common plane exists, then it should be possible to find an expression for a vector $N^\infty$ that is perpendicular to each of the $u_j^i$'s in the limit $i \to \infty$. Stated as an equation, we might seek a vector $N^\infty$ such that

$$N^\infty \cdot u_j^i \to 0$$

for $j = 2, ..., m$ as $i \to \infty$. This does not quite work, however, because each $u_j^i$ is approaching the zero vector, implying that the above condition would trivially hold for any vector $N^\infty$. This problem is overcome by considering the unit vectors $\hat{u}_j^i$. Thus, we seek a vector $N^\infty$ such that

$$N^\infty \cdot \hat{u}_j^i \to 0$$

for $j = 2, ..., m$ as $i \to \infty$.

If $r_{jk}$ denotes the entry in the $j$-th row of $r_k$, then

$$\begin{aligned}
\hat{u}_j^i &= \frac{p_j^i - v^\infty}{\|p_j^i - v^\infty\|} \\
&= \frac{\lambda^i(c_2 r_{j2} + c_3 r_{j3}) + \lambda_4^i c_4 r_{j4} + \cdots}{\|\lambda^i(c_2 r_{j2} + c_3 r_{j3}) + \lambda_4^i c_4 r_{j4} + \cdots\|} \\
&= \frac{(c_2 r_{j2} + c_3 r_{j3}) + \frac{\lambda_4^i}{\lambda^i} c_4 r_{j4} + \cdots}{\|(c_2 r_{j2} + c_3 r_{j3}) + \frac{\lambda_4^i}{\lambda^i} c_4 r_{j4} + \cdots\|}
\end{aligned}$$

In the limit as $i \to \infty$,

$$\hat{u}_j^\infty = \lim_{i\to\infty} \hat{u}_j^i = \frac{c_2 r_{j2} + c_3 r_{j3}}{\|c_2 r_{j2} + c_3 r_{j3}\|}. \qquad (14)$$

Equation 14 implies that each of the limiting unit vectors $\hat{u}_j^\infty$, $j = 2, ..., m$ is a linear combination of the vectors $c_2$ and $c_3$. All the vectors $\hat{u}_j^\infty$ must therefore lie in the plane spanned by $c_2$ and $c_3$. The normal vector $N^\infty$ we seek is therefore $c_2 \times c_3$. □

Again using a discrete Fourier transform technique, one can show that for Catmull-Clark surfaces,

$$\lambda := \lambda_2 = \lambda_3 = \frac{4 + A_n}{16}$$

$$c_2 = \sum_j A_n \cos(\frac{2\pi j}{n}) e_j^1 + (\cos(\frac{2\pi j}{n}) + \cos(\frac{2\pi(j+1)}{n})) f_j^1$$

where

$$A_n = 1 + \cos(\frac{2\pi}{n}) + \cos(\frac{\pi}{n})\sqrt{2(9 + \cos(\frac{2\pi}{n}))}.$$

The vector $c_3$ is obtained from $c_2$ by replacing $e_j^1$ with $e_{j+1}^1$ and $f_j^1$ with $f_{j+1}^1$.

# B Integrating the fairness functional

In this appendix, we consider the problem of evaluating the fairness norm of Equation 7 for a patch whose local control mesh $P$ contains an extraordinary point, such as the one shown in Figure 2(b). As motivated in Section 4, we will ultimately evaluate only the non-divergent part of the thin plate energy corresponding to the deviation of $P$ from its regular component $P'$. As we show below, it is not necessary to compute $P'$ explicity, so we will for the time being evaluate the energy of $P$.

The quadratic form $\mathbf{K}$ referred to in Equation 6 can be written as a weighted sum of two quadratic forms $\mathbf{K}_m$ and $\mathbf{K}_p$, representing the membrane and plate energies, respectively for a bicubic patch:

$$\mathbf{K} = \alpha\mathbf{K}_m + \beta\mathbf{K}_p.$$

Let $E(n, P, j)$ denote the fairness norm of Equation 7 integrated over a patch containing at most one extraordinary point of order $n$ whose local mesh is described by the column vector of control points $P$, and whose level of subdivision is $j$. As outlined in Section 4, when $n \neq 4$, we evaluate $E(n, P, j)$ by splitting the patch into four subpatches, three of which are ordinary (shown in gray in Figure 2), and one of the same form as the original. the This leads to the following recurrence relation for $E(n, P, j)$:

$$\begin{aligned} E(4, P, j) &= P^T \left(\alpha \mathbf{K}_m + 4^j \beta \mathbf{K}_p\right) P \\ E(n, P, j) &= \sum_{k=1}^{3} E(4, \Omega_k P, j+1) + E(n, \Omega_4 P, j+1) \end{aligned}$$

where $\Omega_1, \Omega_2, \Omega_3$ are matrices that carry $P$ into the local meshes for the ordinary (shaded) subpatches, and where $\Omega_4$ is the matrix that carries $P$ into the local mesh for the remaining (unshaded) extraordinary subpatch.

The factor of $4^j$ in front of $\mathbf{K}_p$ reflects the change of integration variables when a patch is subdivided $j$ times. The choice of powers of 4 is somewhat arbitrary. It corresponds to the parametrization assigned to the bicubic subpatches created when the extraordinary patch is subdivided. We have chosen powers of 4 since it is the correct factor for bicubic patches. We are, however, currently experimenting with methods to select this factor based on $n$.

The above recurrence can be unrolled to produce an infinite series for $E(n, P, 0)$:

$$E(n, P, 0) = \sum_{j=1}^{\infty} \sum_{k=1}^{3} E(4, \Omega_k \Omega_4^{j-1} P, j)$$

which can be written as

$$E(n, P, 0) = P^T \mathbf{K}_n P$$

where

$$\mathbf{K}_n := \sum_{j=1}^{\infty} (\Omega_4^{j-1})^T (\bar{\mathbf{K}}_m + 4^j \bar{\mathbf{K}}_p) \Omega_4^{j-1},$$

and where

$$\bar{\mathbf{K}}_m := \sum_{k=1}^{3} \alpha \Omega_k^T \mathbf{K}_m \Omega_k,$$

$$\bar{\mathbf{K}}_p := \sum_{k=1}^{3} \beta \Omega_k^T \mathbf{K}_p \Omega_k.$$

The limiting value of the series can be found by expanding $\Omega_4$ in its basis of eigenvectors:

$$\Omega_4 = \mathbf{X} \Lambda \mathbf{X}^{-1}$$

where $\Lambda$ is a diagonal matrix containing the eigenvalues of $\Omega_4$, and where the columns of $\mathbf{X}$ are the corresponding right eigenvectors. Without loss of generality we can assume that the eigenvalues appear in decreasing order down the diagonal. $\mathbf{K}_n$ can now be written as

$$\begin{aligned} \mathbf{K}_n = \mathbf{X}^{-T} \underbrace{\left\{ \sum_{j=1}^{\infty} \Lambda^{j-1} \mathbf{X}^T \bar{\mathbf{K}}_m \mathbf{X} \Lambda^{j-1} \right\}}_{\bar{\mathbf{K}}_m} \mathbf{X}^{-1} + \\ \mathbf{X}^{-T} \underbrace{\left\{ \sum_{j=1}^{\infty} 4^j \Lambda^{j-1} \mathbf{X}^T \bar{\mathbf{K}}_p \mathbf{X} \Lambda^{j-1} \right\}}_{\bar{\mathbf{K}}_p} \mathbf{X}^{-1}. \end{aligned}$$

Since $\Lambda$ is diagonal, the $ab$-th entry of $\bar{\mathbf{K}}_m$ is

$$(\bar{\mathbf{K}}_m)_{ab} = (\mathbf{X}^T \bar{\mathbf{K}}_m \mathbf{X})_{ab} \sum_{j=1}^{\infty} (\Lambda_{aa})^{j-1} (\Lambda_{bb})^{j-1}.$$

The above series is geometric, so if $\Lambda_{aa}\Lambda_{bb} < 1$, it converges to

$$(\bar{\mathbf{K}}_m)_{ab} = \frac{(\mathbf{X}^T \bar{\mathbf{K}}_m \mathbf{X})_{ab}}{1 - \Lambda_{aa}\Lambda_{bb}}.$$

Using arguments as in appendix A, it can be shown that the largest eigenvalue of $\Omega_4$ is one, meaning that the product $\Lambda_{aa}\Lambda_{bb}$ is at most one, and this occurs only when $a = b = 1$. The membrane energy is invariant under translation, which is reflected in the fact that $(\mathbf{X}^T \bar{\mathbf{K}}_m \mathbf{X})_{11}$ is zero; hence $(\bar{\mathbf{K}}_m)_{11} = 0$.

A similar analysis for $\bar{\mathbf{K}}_p$ shows that

$$(\bar{\mathbf{K}}_p)_{ab} = 4(\mathbf{X}^T \bar{\mathbf{K}}_p \mathbf{X})_{ab} \sum_{j=1}^{\infty} 4^{j-1} (\Lambda_{aa})^{j-1} (\Lambda_{bb})^{j-1}.$$

Thus, $(\bar{\mathbf{K}}_p)_{ab}$ is finite whenever $4\Lambda_{aa}\Lambda_{bb} < 1$. The factor $4\Lambda_{aa}\Lambda_{bb}$ can be shown to be one or larger when $1 \leq a, b \leq 3$. Just as for the membrane energy, the 11 entry poses no difficulty since $(\mathbf{X}^T \bar{\mathbf{K}}_p \mathbf{X})_{11} = 0$, indicating that the thin plate energy is invariant under translation.

The remaining 8 entries of $\bar{\mathbf{K}}_p$ are unbounded for $n > 4$. When $n = 4$ (i.e., the ordinary case), $4\Lambda_{aa}\Lambda_{bb} = 1$, yet we know that the entries of $\bar{\mathbf{K}}_p$ are finite since bicubic patches have finite thin plate energy. We therefore conclude that for $n = 4$, $(\bar{\mathbf{K}}_p)_{ab} = 0$ for $1 \leq a, b \leq 3$. This reflects the fact that regular control meshes have zero thin plate energy. To generalize this idea to arbitrary $n$, we simply set the remaining 8 divergent terms to zero, which is equivalent to evaluating the norm on $P - P'$.

To summarize, the quadratic form related to the thin plate energy is taken to be

$$(\bar{\mathbf{K}}_p)_{ab} = \begin{cases} \dfrac{4(\mathbf{X}^T \bar{\mathbf{K}}_p \mathbf{X})_{ab}}{1 - 4\Lambda_{aa}\Lambda_{bb}} & \text{if } 4\Lambda_{aa}\Lambda_{bb} < 1 \\ 0 & \text{otherwise} \end{cases}$$

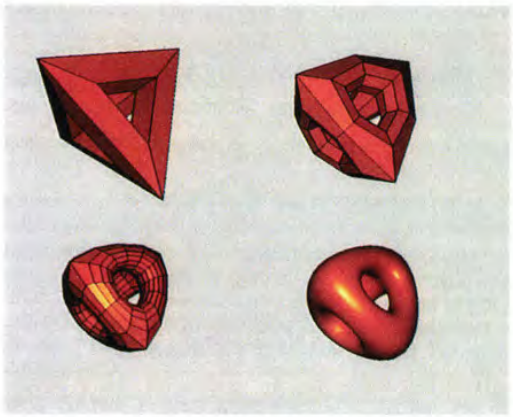


Figure 3: Upper Left: Tetrahedral mesh with holes. Upper Right: The mesh after one Catmull-Clark subdivision. Lower Left: The mesh after two subdivisions. Lower Right: The limit surface.

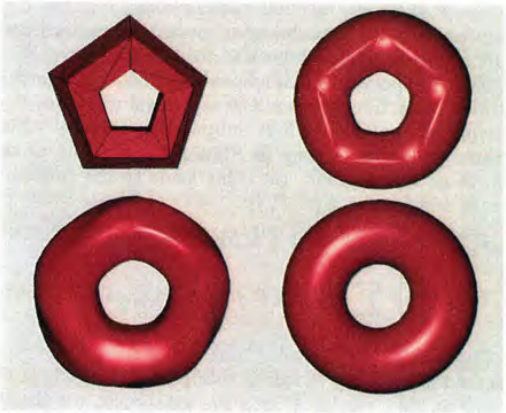


Figure 4: Interpolating a coarsely polygonized torus. Upper left: original mesh. Upper right: Shirman-Séquin interpolation[14]. Lower left: Interpolating Catmull-Clark surface. Lower right: Faired interpolating Catmull-Clark surface.

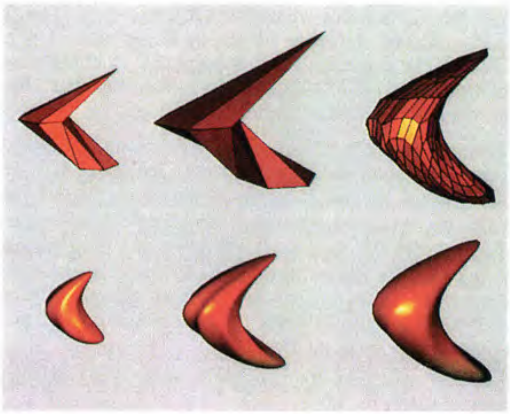


Figure 5: Top row: Original mesh, Interpolating mesh, Faired interpolating mesh. Bottom row: Corresponding Catmull-Clark surfaces. Interpolation introduces wiggles which are removed by fairing.

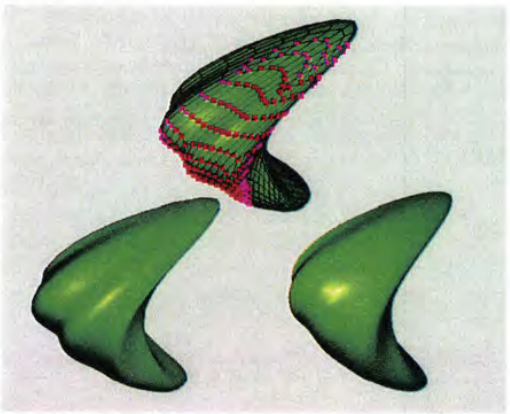


Figure 6: Lower left: unfaired interpolating surface. Upper center: Interactive fairing. Red vertices are allowed to move. Magenta vertices influence the minimization, but remain fixed. Lower right: Result after fairing.

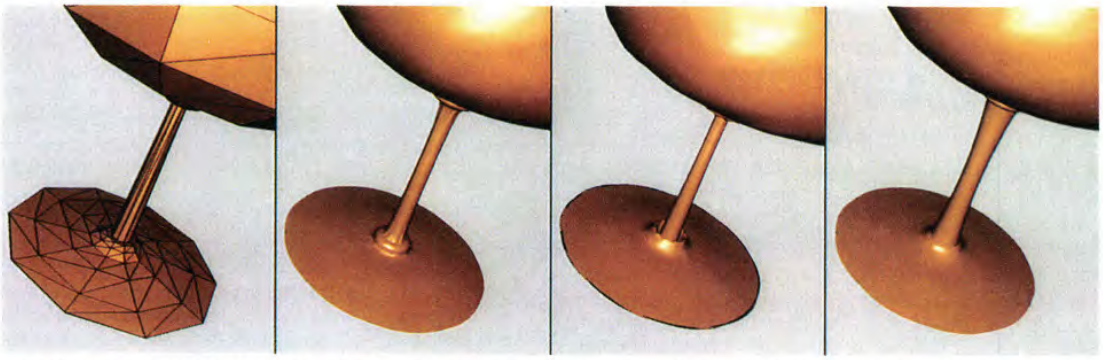


Figure 7: From left to right: Original goblet mesh containing 190 vertices. Ordinary Catmull-Clark surface (approximating). Interpolating Catmull-Clark surface. Faired interpolating Catmull-Clark surface. The far right surface interpolates the original mesh without the artifacts present in the middle two surfaces.



# Implementing Rotation Matrix Constraints in Analog VLSI

David B. Kirk Alan H. Barr

California Institute of Technology
Computer Graphics 350-74
Pasadena, CA 91125
email: dk@egg.gg.caltech.edu

**Abstract**

We describe an algorithm for continuously producing a 3x3 rotation matrix from 9 changing input values that form an approximate rotation matrix, and we describe the implementation of that constraint in analog VLSI circuits. This constraint is useful when some source (e.g., sensors, a modeling system, other analog VLSI circuits), produces a potentially "imperfect" matrix, to be used as a rotation. The 9 values are continuously adjusted over time to find the "nearest" true rotation matrix, based on a least-squares metric. The constraint solution is implemented in analog VLSI circuitry; with appropriate design methodology [Kirk 93], adaptive analog VLSI is a fast, accurate, and low-power computational medium. The implementation is potentially interesting to the graphics community because there is an opportunity to apply adaptive analog VLSI to many other graphics problems.

**CR Categories and Subject Descriptors:** C.1.2—**[Processor Architectures]:** Multiprocessors - parallel processors; C.1.3—**[Processor Architectures]:** Other Architecture Styles; I.3.1—**[Computer Graphics]:** Hardware Architecture - raster display devices; I.3.3—**[Computer Graphics]:** Picture/Image Generation; I.3.5—**[Computer Graphics]:** Computational Geometry and Object Modeling; I.3.7—**[Computer Graphics]:** Three-Dimensional Graphics and Realism
**General Terms:** Algorithms, Graphics, Hardware
**Additional Key Words and Phrases:** Animation, rotation, robotics, simulation, constraint solution, interaction, adaptive, analog, CMOS, VLSI.

## 1 Introduction

This paper has two main purposes. First, we demonstrate the implementation of a nontrivial constraint technique in analog VLSI. Second, since some of the computer graphics community may not be familiar with recent developments in analog VLSI technology, we describe some of the potential benefits. We believe that analog VLSI has great potential as a computation medium for implementing rendering, modeling, and interactive operations.



### 1.1 Computation in Computer Graphics

There is a history of digital VLSI acceleration in computer graphics: geometry engines [Clark 82], hardware frame buffer assists [Rhoden 89], vector generators [Barkans 90], systems [Voorhies 88] [Fuchs 89], etc. Most high-performance graphics workstations have a substantial amount of special purpose digital chips to provide the kind of interactive performance that we have come to expect. Most of this silicon is dedicated to rendering tasks, although it can be argued that the geometric transformations performed in hardware constitute modeling hardware.

Any computational medium used for graphics needs to be able to perform mathematical operations accurately and precisely. The bulk of simulation and modeling calculations for computer graphics are performed in software. For instance, for *physically-based modeling*, the shapes and motions of graphical objects are computed according to the physics underlying the simulation. This process requires the solution of differential equations (for converting the relation $F = ma$ into position and velocity). As we consider collisions between objects, we may also be required to solve for roots of nonlinear equations. For modeling, we require the ability to accurately and precisely solve a variety of mathematical equations.

Applying real physical constraints to computer graphics models requires great computational resources. Even relatively simple simulations, involving only a few primitives, may consume many seconds of CPU time on a fast computer. The traditional arguments are that CPU price-performance doubles every year, and that massive parallelism will save us. We claim that current digital computation approaches are approximately a factor of 10,000 times too slow for real-time simulation of complex scenes.

In order to be effective in addressing this problem, a computational medium must be fast and accurate. If we can produce a technology which can accurately and precisely compute the solutions of equations, we can then use the technology to construct computer graphics hardware. We hope that adaptive analog VLSI can be used to realize the goal of performing graphics calculations thousands of times faster than is possible today.

### 1.2 Adaptive Analog VLSI

There has been increasing interest recently in using analog VLSI [Mead 89] for a variety of computational tasks. Mead and others have pursued the paradigm of using ana-

 ACM-0-89791-601-8/93/008/0045 $01.50

log transistors to model components of neural systems. Related research has focused on increasing the accuracy and precision of computation with analog VLSI [Kirk 93], and on developing a design methodology for creating analog VLSI circuits which can be adjusted to perform to the desired accuracy [Kirk 91]. These techniques make analog VLSI more tractable for quantitative computation.

This is not the first appearance of analog computation in computer graphics. Certainly, there is some amount of analog hardware in every graphics system, at least in the form of a D/A (digital-to-analog) converter in the path to the video monitor. There have also been more extensive uses of analog, however. For instance, Vector General implemented matrix multiplication for the purpose of performing coordinate transformations in analog circuitry, although not in analog VLSI.

It is important to note that in these discussions, we have chosen a *particular* constraint to demonstrate the general technique of implementing a constraint in analog VLSI. There are many other examples of useful constraint computation that could be formulated in a similar fashion [Platt 89] [Barzel 92], and also could be implemented in analog hardware. The particular constraint that we have chosen to implement is meant to be representative of a large set of possibilities. Our example raises the exciting prospect of implementation of extensive hardware *modeling* assists in analog VLSI. There are also many rendering tasks which are appropriate for analog VLSI hardware implementation, but we won't discuss them in this paper. We have chosen to describe a constraint technique that is appropriate for interactive input devices, and has application to modeling as well.

### 1.3 The Rotation Matrix Constraint

The constraint technique that we have chosen is the orthonormalization of a rotation matrix. We chose the 3x3 matrix formulation because it is easier to perform coordinate transformations with the same underlying computational modules that are used to implement the constraints. In Sec. 4, we describe several computational blocks that we can also use to construct coordinate transformation hardware. The matrix formulation is also complex enough to be interesting as an example problem for hardware.

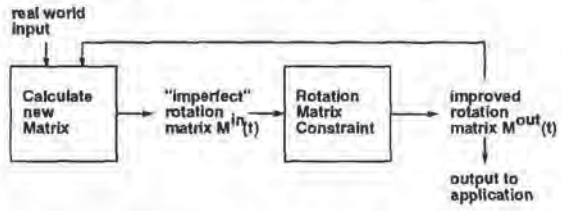



Figure 1: A system-level view of the rotation matrix constraint enforcement, and how the result of applying the constraint might be used.

The rotation matrix constraint is particularly useful as part of an interactive system, as shown in Fig. 1. For virtual reality applications, a sensor may be used to produce a 3D orientation, in the form of a 3x3 rotation matrix. Sensors are often flawed, noisy, or otherwise inaccurate and do not provide sufficient and reliable information for producing an accurate rotation matrix. In such cases, we then wish to continuously produce a "best estimate" rotation matrix, based on the sensor measurements. One example of such a system involves producing rotation matrices from approximate inputs from sensors or interactive devices. The system produces approximate rotation matrices over time from angular velocity $\omega(t)$, according to the following relation:

$$\underline{\underline{M}}'(t) = \underline{\omega} \times \underline{\underline{M}}(t) \tag{1}$$

Such a system would produce an approximate rotation matrix at each time step, and may accumulate errors over time. The errors could be corrected by the constraint technique described in this paper.

A similar task exists in robotics applications. We might have a sensor which can detect the position of an end effector of a robot arm, and also a measure of the control inputs. In practice, a robot arm is often controlled by providing joint angle control inputs. However, the control may be inaccurate, and there may be "slop" in the joints. We may want to then compute an estimate of the actual joint angles, which, if the arm segments are rigid, must be pure rotations.

There are also many applications to physically-based modeling. When solving constraint equations for motion of rigid bodies, we may produce values that are inaccurate due to accumulating arithmetic roundoff errors, integration step size, or approximations in our model. When combined to form a rotation matrix to describe the orientation of a body, the errors may cause the introduction of scaling or skewing into the matrix. The constraint technique described in this paper will allow us to automatically adjust for these errors.

In Sec. 2, we describe the constraint algorithm that we use to produce the rotation matrix. In Sec. 3, we introduce in more detail the technology used for the implementation (analog VLSI), and explain why we believe that it has great potential to be useful for computer graphics. In Sec. 4, we present a block diagram description of the constraint chip.

## 2 The Constraint Algorithm

Our goal is to produce a 3x3 rotation matrix containing no scale or skew components, given 9 numbers which are already nearly a rotation matrix.

For a mathematically perfect rotation matrix $M$,

$$MM^T = I \tag{2}$$

where $I$ is the identity matrix.

We define the function $f()$:

$$f(M) = (MM^T - I) : (MM^T - I) \tag{3}$$

where the double-dot operator (:) denotes the sum of products of terms of the two matrices, producing a scalar result, analogous to the dot product of two vectors. When $M$ is a rotation matrix (or reflection), $f(M)$ in Eqn. 3 is equal to zero, and when $M$ is not purely a rotation matrix, $f(M) \neq 0$. Since $f(M)$ is always greater than or equal to zero, $M$ is a rotation matrix when $f(M)$ is minimized.

We perform continuous gradient descent to minimize the function $f()$, as follows:

$$M'(t) = -\epsilon \, \nabla f(M(t)) \tag{4}$$

where epsilon is a parameter which determines the speed of the descent. Appendix 1 describes the derivation of our gradient calculation method in detail.

The analog VLSI implementation does not suffer from many of the problems of digital implementations, since analog circuits can operate in continuous time. For instance, in a digital implementation, Euler's method might be used to solve Eqn. 4. With large step sizes, Euler's method frequently becomes unstable. With small step sizes, Euler's method may converge slowly or not at all. Other techniques, such as the conjugate gradient method, may improve the performance in digital implementations. The continuous nature of an analog implementation, however, avoids this type of problem entirely.

As the computation proceeds, two kinds of changes are occurring. First, the imperfect input matrix may be changing over time. Second, based on our optimization process, the output matrix will be changing to fulfill our rotation matrix constraint. Since the analog VLSI circuit operates very quickly, and in continuous time, the optimization can occur at a much finer time scale than the changing of the input matrix.

## 3 Adaptive Analog VLSI

There has been increasing interest recently in using analog VLSI [Mead 89] for a variety of computational tasks. One of Mead's insights is that rather than developing an entirely new manufacturing technology for producing analog VLSI chips, we can produce analog CMOS VLSI chips using standard digital CMOS VLSI processes. The key element in this strategy is to produce designs that are tolerant to the device variations that are present in a digital production process. Another component of this design philosophy is the exploration of architectures and circuits that are tolerant of device variations.

Other research has focused on increasing the accuracy and precision of computation with analog VLSI [Kirk 93], and on developing a design methodology for creating analog VLSI circuits which can be adjusted to perform to the desired accuracy [Kirk 91]. This work can be characterized as using adaptation and optimization to harness analog VLSI for more "conventional computing" applications. This approach is attractive because analog transistors provide a rich computational gamut. Fig. 2 (upper) shows the current flowing through an analog transistor as its gate voltage is varied. Fig. 2 (lower) shows the current as the source-to-drain voltage is varied, while holding the gate voltage constant. These figures are meant as a qualitative demonstration of the variety of current-voltage responses available from a single transistor. Note the regions of roughly linear, exponential, and quadratic I-V relation.

It is possible make analog circuits more quantitatively useful, by designing *compensatable* circuit building blocks that can be adjusted to perform more closely to some performance metric. For example, let us assume that our goal is to build a "perfect" analog multiplier. In analog VLSI, we can easily build a circuit which computes an "imperfect" multiply-like operation, but the "perfect" multiply is more elusive. We can design a multiplier that is monotonic within some input range, and operates in four quadrants (the sign of the output is correct for all combinations of inputs' sign). Without extreme care in

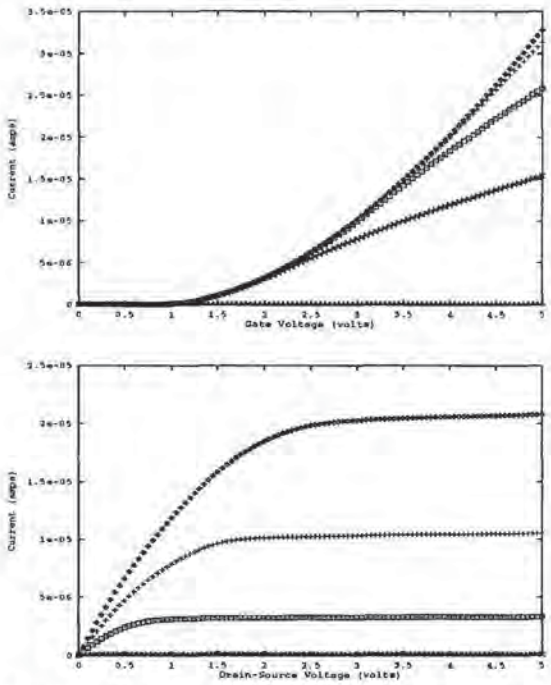



Figure 2: The upper graph shows the drain current of a single transistor, as the gate voltage is varied from 0 to 5 volts. The family of curves represents varying the difference between the source and drain voltage. The lower graph shows the drain current as the source-to-drain voltage difference is varied from 0 to 5 volts. The family of curves represents varying the gate voltage. The analog VLSI multipliers discussed in Sec. 3 operate in the nearly linear region to the right of the upper graph and to the left of the lower graph.

the design, however, the "multiplier" would have a number of drawbacks. The circuit's response might deviate significantly from the desired linear function of its inputs

$$f(x, y) = x * y \tag{5}$$

The "multiplier" would also, very likely, have nonzero input offsets[1].

A compensated multiplier has adjustable parameters which allow for the improvement of the linear range of behavior, as well as the cancellation of input offsets. A description of how to design, build, and optimize compensatable components is presented in detail in [Kirk 93]. Sec. 5 presents some measurements from chips implemented and compensated using these techniques.

## 4 Applying Analog VLSI to the Constraint Problem

Now that we have described the desired constraints (in Sec. 2) and the substrate technology of adaptive analog VLSI (Sec. 3), we will explain, at a block diagram level,

[1] Input offsets are present for an analog multiplier $f(x, y) \approx x * y$ when $f(x, 0) \neq 0$ or $f(0, y) \neq 0$.

how we use analog VLSI to solve the constraint problem. These block diagrams represent a hierarchical decomposition of the chip that we built.

As one might guess from the form of the equations of the derivation in Sec. 2, the circuit architecture is a nested, structured hierarchy of dot products, with some additional computation.

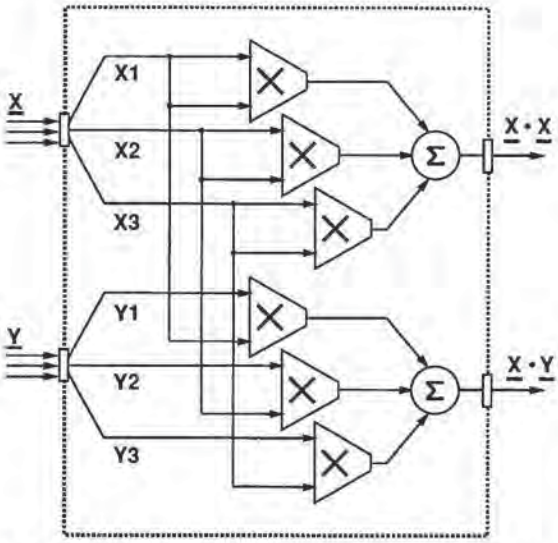



Figure 3: A functional block containing two dot products. The inputs are the matrix column vectors $\underline{X}$ and $\underline{Y}$, and the outputs are the scalars $X \cdot X$ and $X \cdot Y$.

We can think of the nine input values, the "imperfect" rotation matrix, as the three 3D basis vectors, $\underline{X}$, $\underline{Y}$, and $\underline{Z}$, (the three columns of the matrix). We can see by examining Eqn. 13 that the computation of the various components of the gradient, $\eta_{pq}$, requires dot products of the matrix basis vectors. Appendix 1 describes the calculation of $\eta_{pq}$. Fig. 3 shows a functional block which computes two of the six basis vector dot products that are required.

Fig. 4 shows a set of three functional blocks (from Fig. 3) which together compute the six 3D basis vector dot products that are required to form the gradient, $\eta_{pq}$, as shown in Eqn. 13. The details of the circuit, the device layout, and the compensation procedure for the multiplier and dot product blocks are presented in [Kirk 93].

Fig. 5 shows the use of the basis vector inputs and three of the dot product results to produce the gradient components for one of the basis vectors, in this case, $\underline{X}$.

Fig. 6 shows a set of three constraint blocks, from Fig. 5, which together compute all of the components of the gradient for the correction of the imperfect matrix. The combination of these three constraint blocks and the three dot product blocks from Fig. 4 forms the gradient calculation hardware. $X_1'$, $X_2'$, $X_3'$, $Y_1'$, $Y_2'$, $Y_3'$, $Z_1'$, $Z_2'$, and $Z_3'$ are the nine derivative components. Together, they form the gradient, which we will use to optimize the components of the matrix $\underline{\underline{M}}$. Descending along the direction of the gradient will produce a matrix which fulfills our constraints.

We use the derivative terms from Fig. 6 to add or subtract from the original input values of the matrix, $\underline{\underline{M}}$. Since the circuits are analog and operate in continuous

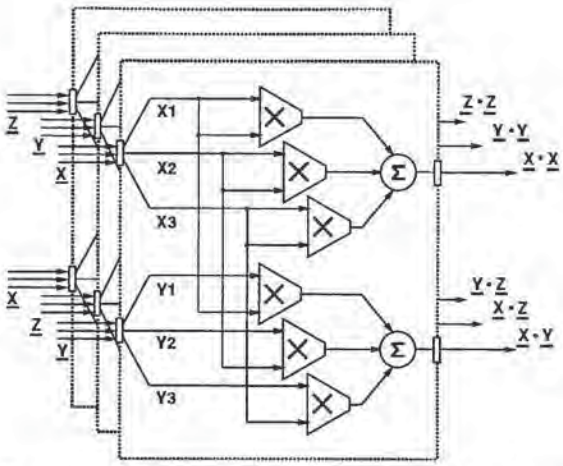



Figure 4: A collection of three dot product blocks, from Fig. 3. With the 3D basis vector inputs $\underline{X}$, $\underline{Y}$, and $\underline{Z}$, they compute the six dot products required to enforce the constraints.

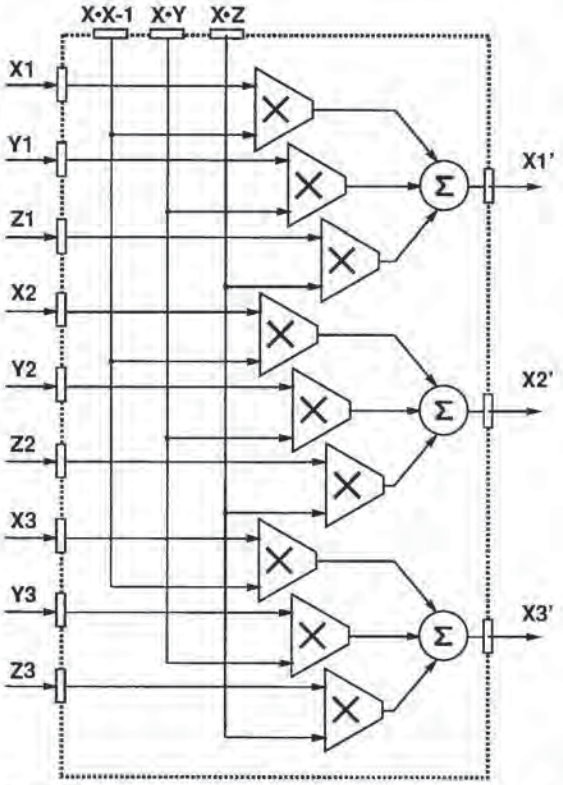



Figure 5: A basis vector constraint block, using the outputs from Fig. 4. This computational element implements the rotation matrix constraint for one of the three matrix column vectors.

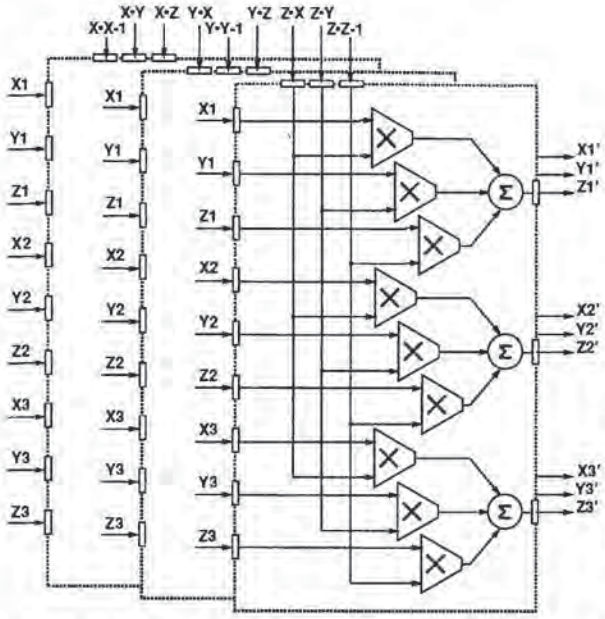



Figure 6: A collection of three constraint blocks, from Fig. 5. The combination of these three constraint blocks and the three dot product blocks from Fig. 3 forms the gradient calculation hardware.

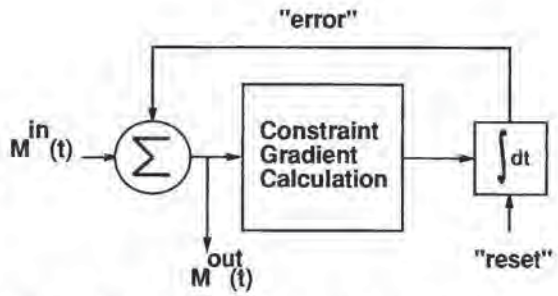



Figure 7: An example of a gradient descent process, as employed to enforce the rotation matrix constraint. The feedback from the gradient calculation modifies the effective inputs to the constraint gradient calculation box. As the constraint is satisfied, the output, $M^{out}(t)$ settles to a rotation matrix, if $M'^{in}(t)$ is not changing, or is changing at a slower time scale. Note the "reset" input to the integrator box. If the input matrix $M'^{in}(t)$ changes discontinuously, we want to restart the constraint optimization from the new matrix, and we can accomplish this using the integrator reset.

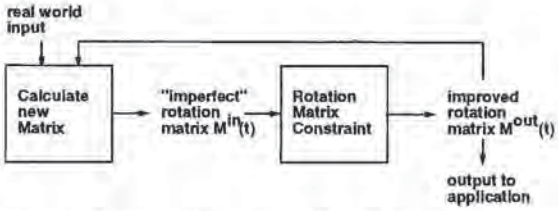



Figure 8: A system-level view of the rotation matrix constraint enforcement, and how the result of applying the constraint might be used.

time, we can integrate these corrections on capacitors, and use the gradient components to set the level of current to add/subtract. Thus, this circuit structure can be use to continuously track and correct a (potentially flawed) matrix that changes over time. Fig. 7 shows the connections required to provide the feedback from the calculated gradient components to modify the input matrix components. The gradient calculation occurs in continuous time, using the analog VLSI hardware. The input can change continuously, or discretely (using the "reset" input in Fig. 7), and the constraint solution will track the input.

Fig. 8 shows a schematic view of the the rotation matrix constraint solution box connected as part of a system. Given a source of approximate rotation matrices $M'^{in}(t)$, the constraint enforcement produces rotation matrices $M^{out}(t)$, which can be used for modeling, rendering, or control applications.

## 5 Results

We have designed, implemented, fabricated, and tested chips which contain compensated multipliers, dot products, and constraint blocks, as described in Fig. 3 through Fig. 6. The design is modular (similar to the structure of the figures), so that we are confident that the system will work, given the partial test results. We have tested all of the components, and present the data in this paper. This section contains measured chip data for the compensated multiplier, (along with some raw data for the uncompensated version), and the dot product with hierarchical compensation. We also present a software simulation of the constraint process in action, using the constraint technique described herein.

Fig. 9 shows the results of the raw multiplier before compensation. Note the nonzero offsets, as evidenced by the nonzero slope line formed by the square symbols. That line represents the results of multiplying zero by a set of other quantities, so should be horizontal, at zero.

Fig. 10 shows the output of a compensated multiplier circuit. Note that the "zero" line (again delineated by the square symbols), is much closer to horizontal at zero, due to the effects of the compensation. It is appropriate to discuss accuracy and precision at this time. As a multiplier, the circuit is highly accurate: it computes a function that is very close to the desired $f(x, y) = kxy$. The precision is more difficult to quantify than the accuracy, however. The relative error quantity (0.1%) seems to indicate 10 bits of precision, although noise may reduce the repeatable precision to somewhat less than that. Although in this case we have only compensated for first-order effects of device variations, it is possible to design circuits which compensate for higher order nonlinearities as well. In order to use compensated components to produce an accurate and precise computational *system*, care must be taken to consider the quality and magnitude of errors that can be tolerated at each stage of the computation.

Fig. 11 shows the compensated voltage-in, voltage-out

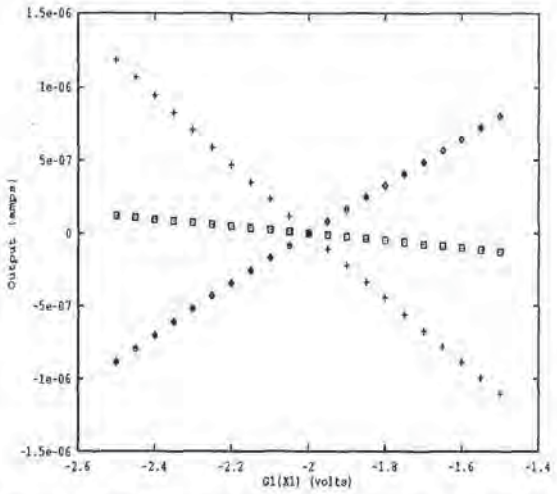



Figure 9: The output from an *uncompensated* multiplier circuit (actual measured chip data). The analog multiplier circuit has not been adapted to compensate for input offsets and other device variations. Note the nonzero offsets, as evidenced by the nonzero slope line formed by the square symbols. That line represents the results of multiplying zero by a set of other quantities, so should be horizontal, at zero.

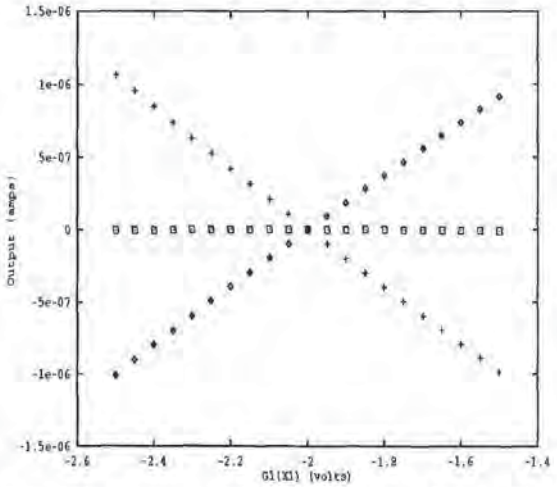



Figure 10: The output from a compensated multiplier circuit (actual measured chip data). The relative error (ourput error / input range) is less than 0.1% over most of the operating range. At extreme (large) inputs, the relative error may be as large as 2%. For this application, the precision is most important for small values.

multiplier performance. The signal presented in this figure is an intermediate value in the hierarchical constraint computation. Its nonlinearity and nonzero offset characteristics reflect the fact that this output contains biases to compensate for variations in the next stage of computation. These curves represent the sum of the multiplier output and the compensation input for a subsequent computational element. [Kirk 93] contains more detailed descriptions of hierarchical compensation techniques.

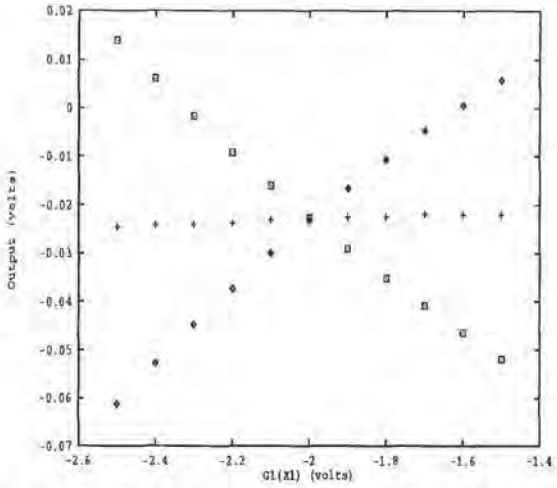



Figure 11: The output of a multiplier, after nearly linear current-to-voltage conversion (actual measured chip data).

Fig. 12 shows the three multiply components of a compensated dot product. Note that the offset correction is very accurate, but that the linearity is somewhat less accurate.

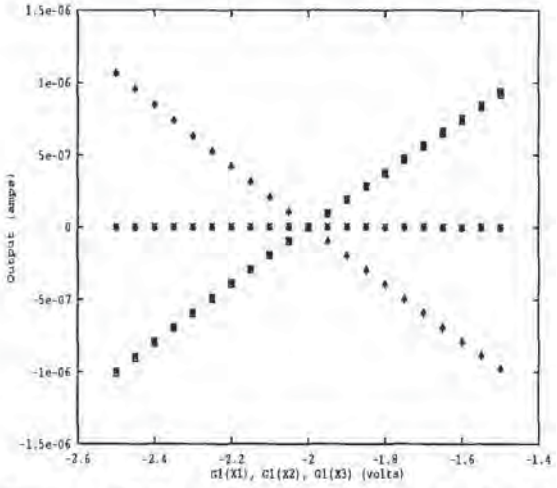



Figure 12: The 3 components of a dot product (actual measured chip data). The characteristics of the three multiply operations are similar, with respect to the input offset magnitudes and shape of nonlinearities.

Fig. 13 shows the results of a simulation of our constraint technique in action.

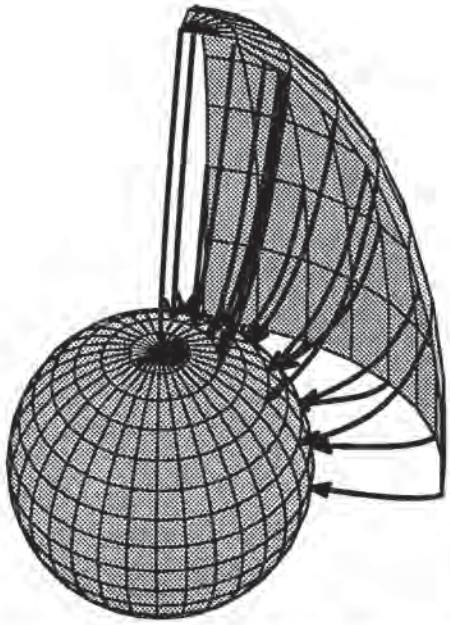


Figure 13: The results of a software simulation of our constraint technique in action. The outer curved octant represents the manifold of a set of points transformed using the input imperfect matrix. The inner, more spherical shape represents the same points (and more) transformed through the constrained rotation matrix. The lines drawn between the two shapes represent the constraint optimization path taken by our algorithm.

### 5.1 Expected Performance

We compare the expected performance of the continuous-time analog VLSI rotation constraint chip to a software implementation on a fast digital computer. Using the constraint algorithm described in this paper, and Euler's method to perform the optimization, we expect that the orthonormalized matrix can be produced in about 75 microseconds on a roughly 100 Mflop workstation.

The multiplier core used in the analog VLSI rotation constraint chip can easily be run at product rates in excess of 2 Mhz [Denyer 81]. Since the multipliers and constraint circuitry operate in continuous time, we expect convergence at a much greater rate than in the discrete digital case. The analog VLSI rotation constraint chip should produce an orthonormalized matrix in roughly 2-3 microseconds. So, the current implementation should outperform a general-purpose digital solution by about a factor of 25, and we believe that this is a conservative estimate. Furthermore, the analog VLSI solution is extremely low cost, and low power, and leaves the workstation processor free to pursue other tasks. The analog VLSI chips were fabricated in 2.0 micron CMOS using the MOSIS fabrication service, and dissipate power on the order of microwatts. The entire constraint solution circuit consumes roughly 2 square millimeters of chip area. Finally, faster multiplier circuits can be used to further increase the analog VLSI performance.

## 6 Conclusions

We describe a constraint technique for producing orthogonal, unit scale rotation matrices from "imperfect" inputs. The technique is potentially useful in a system which produces a sequence of approximate rotation matrices over time. Additional potential applications are covered briefly in Sec. 1.

We also describe the emerging and evolving technology of adaptive analog VLSI and speculate on its possible value to the field of computer graphics. In the example of the rotation system above, an analog VLSI rotation matrix constraint solver could enforce the rotation constraint *continuously* as the matrix is updated.

Interpreting this result with a broader view, we have demonstrated the implementation of a nontrivial constraint in analog VLSI. This is significant because it implies a future of implementing "hardware for modeling" in the form of hardware constraint solution. Current digital implementations of constraint systems cannot compute real time constraint solutions for models containing more than a few bodies.

Many of the tasks in computer graphics simulation and modeling involve the solution of various types of mathematical equations. The development of analog VLSI technology for accurate and precise computation [Kirk 93], makes it possible to build analog hardware to solve these equations. The use of CMOS VLSI fabrication makes analog implementations scalable and mass producible. Therefore, adaptive analog VLSI presents an exciting opportunity to consider building hardware to accelerate modeling to a level of performance commensurate with that of digital rendering hardware. We believe that Analog VLSI has the potential to be a significant tool for computer graphics.

## 7 Acknowledgements

This work was supported in part by an AT&T Bell Laboratories Ph.D. Fellowship, and by grants from Apple, DEC, Hewlett Packard, and IBM. Additional support was provided by NSF (ASC-89-20219), as part of the NSF/DARPA STC for Computer Graphics and Scientific Visualization. All opinions, findings, conclusions, or recommendations expressed in this document are those of the authors and do not necessarily reflect the views of the sponsoring agencies. Thanks also to the anonymous reviewers for their many helpful comments.

## References

[Barkans 90] Barkans, Anthony C., "High Speed High Quality Antialiased Vector Generation," Computer Graphics, Vol. 24, No. 4, August, 1990, pp. 319-326.

[Barzel 92] Barzel, Ronen, "Structured Modeling for Computer Graphics," Academic Press, Cambridge, MA, 1992.

[Clark 82] Clark, James, "The Geometry Engine: A VLSI Geometry System for Graphics," Computer Graphics, Vol. 16, No. 3, July, 1982, pp. 127-133.

[Denyer 81] Denyer, Peter B., John Mavor, "MOST Transconductance Multipliers for Array Applications,"

IEEE Proceedings, Volume 128, Pt. I, Number 3, pp. 81-86, June 1981.

[Fuchs 89] Fuchs, Henry, J. Poulton, J. Eyles, T. Greer, J. Goldfeather, D. Ellsworth, S. Molnar, G. Turk, B. Tebbs, and L. Israel, "Pixel-Planes 5: A Heterogeneous Multiprocessor Graphics System Using Processor-Enhanced Memories," Computer Graphics, Vol. 23, No. 3, July, 1989, pp. 79-88.

[Kirk 93] Kirk, David B., "Accurate and Precise Computation using Analog VLSI, with Applications to Computer Graphics and Neural Networks," Ph.D. Thesis, California Institute of Technology, Caltech-CS-TR-93-08, June, 1993.

[Kirk 91] Kirk, David, Kurt Fleischer, and Alan Barr, "Constrained Optimization Applied to the Parameter Setting Problem for Analog Circuits," IEEE Neural Information Processing Systems 1991 (NIPS 91), Morgan Kaufman, San Diego, 1991.

[Mead 89] Mead, Carver, "Analog VLSI and Neural Systems," Addison-Wesley, 1989.

[Platt 89] Platt, John, "Constrained Optimization for Neural Networks and Computer Graphics," Ph.D. Thesis, California Institute of Technology, Caltech-CS-TR-89-07, June, 1989.

[Rhoden 89] Rhoden, Desi, and Chris Wilcox, "Hardware Acceleration for Window Systems," Computer Graphics, Vol. 23, No. 3, July, 1989, pp. 61-67.

[Voorhies 88] Voorhies, Douglas, D. Kirk, and O. Lathrop, "Virtual Graphics," Computer Graphics, Vol. 22, No. 4, August, 1988, pp. 247-253.

### Appendix 1: Derivation of Constraint Equations

The expression $A : A$ can be written:

$$A : A = \sum_{jk} A_{jk} A_{jk} \tag{6}$$

So, we can rewrite Eqn. 3 as:

$$\mathrm{f}(\underline{\underline{M}}) = \sum_{jk} ((\sum_{i} \mathrm{M_{ij}} \mathrm{M_{ik}}) - \delta_{jk})((\sum_{\ell} \mathrm{M_{\ell j}} \mathrm{M_{\ell k}}) - \delta_{jk}) \tag{7}$$

where $\delta_{ij}$ indicates the identity matrix ($\delta_{ij} = 1$ when $i = j$ and 0 otherwise).

In order to use Eqn. 7 to enforce a constraint, we would like to pose it in a form which allows us to do some sort of optimization. More specifically, in order to perform a gradient descent operation, we require a gradient. So, we compute the gradient, using Einstein Summation Notation (ESN):

$$\begin{aligned} \nabla f &= \frac{\partial f}{\partial M_{pq}} && (8) \\ &= 2(M_{ij} M_{ik} - \delta_{jk})(\delta_{\ell p}\delta_{jq} M_{\ell k} + M_{\ell j}\delta_{\ell p}\delta_{qk}) && (9) \\ &= 4(M_{iq} M_{ik} - \delta_{qk}) M_{pk} && (10) \end{aligned}$$

We wish to use Eqn. 8 to perform gradient descent to minimize the function $f()$, as follows:

$$M'(t) = -\epsilon \, \nabla f(M(t)) \tag{11}$$

where epsilon is a parameter which determines the speed of the descent.

We define $\eta$ as the gradient of $f()$:

$$\eta_{pq} = 4(M_{iq} M_{ik} - \delta_{qk}) M_{pk} \tag{12}$$

We can also simplify $M_{iq} M_{ik}$ by introducing $\underline{B}_1$, $\underline{B}_2$, and $\underline{B}_3$ as basis vectors of the matrix $\underline{\underline{M}}$, and $D_{ij}$ as the dot product of $\underline{B}_i$ and $\underline{B}_j$:

$$\begin{aligned} \eta_{pq} &= 4(\underline{B}_q \cdot \underline{B}_k - \delta_{qk}) M_{pk} && (13) \\ &= 4(D_{qk} - \delta_{qk}) M_{pk} && (14) \end{aligned}$$

Since the dot products are symmetric, there are only 6 unique $D_{qk}$ terms: the 3 diagonal terms, $D_{11}$, $D_{22}$, and $D_{33}$, and the three unique cross terms, $D_{12}$ (or $D_{21}$), $D_{23}$ (or $D_{32}$), and $D_{13}$ (or $D_{31}$).

So, the following set of equations describe a form of the gradient descent process:

$$M_{pq}^{\mathrm{new}} = M_{pq}^{\mathrm{old}} - \epsilon \, \eta_{pq} \tag{15}$$

and we can absorb the 4 from Eqn. 13 into $\epsilon$, since $\epsilon$ is an arbitrary constant.

We have the following set of 9 equations for the components of the gradient:

$$\begin{aligned} \eta_{11} &= (D_{11} - 1)M_{11} + D_{12}M_{21} + D_{13}M_{31} && (16) \\ \eta_{12} &= D_{21}M_{11} + (D_{22} - 1)M_{21} + D_{23}M_{31} && (17) \\ \eta_{13} &= D_{31}M_{11} + D_{32}M_{21} + (D_{33} - 1)M_{31} && (18) \\ \eta_{21} &= (D_{11} - 1)M_{12} + D_{12}M_{22} + D_{13}M_{32} && (19) \\ \eta_{22} &= D_{21}M_{12} + (D_{22} - 1)M_{22} + D_{23}M_{32} && (20) \\ \eta_{23} &= D_{31}M_{12} + D_{32}M_{22} + (D_{33} - 1)M_{32} && (21) \\ \eta_{31} &= (D_{11} - 1)M_{13} + D_{12}M_{23} + D_{13}M_{33} && (22) \\ \eta_{32} &= D_{21}M_{13} + (D_{22} - 1)M_{23} + D_{23}M_{33} && (23) \\ \eta_{33} &= D_{31}M_{13} + D_{32}M_{23} + (D_{33} - 1)M_{33} && (24) \end{aligned}$$

We can define $B_1 = \underline{X}$, $B_2 = \underline{Y}$, and $B_3 = \underline{Z}$, so we can now write the discrete time step gradient descent optimization as:

$$\begin{aligned} \underline{X}^{\mathrm{new}} &= \underline{X}^{\mathrm{old}} - \epsilon \, \eta_{p1} && (25) \\ \underline{Y}^{\mathrm{new}} &= \underline{Y}^{\mathrm{old}} - \epsilon \, \eta_{p2} && (26) \\ \underline{Z}^{\mathrm{new}} &= \underline{Z}^{\mathrm{old}} - \epsilon \, \eta_{p3} && (27) \end{aligned}$$

and, we can now write $\eta$ in terms of $\underline{X}$, $\underline{Y}$, and $\underline{Z}$:

$$\begin{aligned} \eta_{11} &= (D_{11} - 1)X_1 - D_{12}Y_1 - D_{13}Z_1 && (28) \\ \eta_{12} &= D_{21}X_1 - (D_{22} - 1)Y_1 - D_{23}Z_1 && (29) \\ \eta_{13} &= D_{31}X_1 - D_{32}Y_1 - (D_{33} - 1)Z_1 && (30) \\ \eta_{21} &= (D_{11} - 1)X_2 - D_{12}Y_2 - D_{13}Z_2 && (31) \\ \eta_{22} &= D_{21}X_2 - (D_{22} - 1)Y_2 - D_{23}Z_2 && (32) \\ \eta_{23} &= D_{31}X_2 - D_{32}Y_2 - (D_{33} - 1)Z_2 && (33) \\ \eta_{31} &= (D_{11} - 1)X_3 - D_{12}Y_3 - D_{13}Z_3 && (34) \\ \eta_{32} &= D_{21}X_3 - (D_{22} - 1)Y_3 - D_{23}Z_3 && (35) \\ \eta_{33} &= D_{31}X_3 - D_{32}Y_3 - (D_{33} - 1)Z_3 && (36) \end{aligned}$$

# Correcting for Short-Range Spatial Non-Linearities of CRT-based Output Devices

R. Victor Klassen
**Xerox Webster Research Center**

Krishna Bharat
**Georgia Institute of Technology**

## ABSTRACT

Most graphical output devices exhibit what has been termed spatial non-linearity: the effect of setting two adjacent pixels to a given value is not the same as the sum of the effects of setting those two pixels to the same value in isolation: checkerboards of different frequencies do not have the same apparent luminance. We present a method applicable to bit-mapped devices for compensating for short-range spatial non-linearity in error-diffused images. The modification to error diffusion is such that it can be used with any error diffusion technique. In essence, it consists of finding the influence of the neighbouring (output) pixels when making the decision of whether to turn on a given pixel, and passing errors computed accordingly.

CR Descriptors: B.4.2 **[Input/Output and Data Communications]**: Input/Output Devices — *Image display;* I.3.1 **[Computer Graphics]**: Hardware Architecture — *Raster display devices* I.3.3 **[Computer Graphics]**: Picture/image generation — *Display algorithms;* I.3.6 **[Computer Graphics]**: Methodology and Techniques; I.4.3 **[Image Processing]**: enhancement.

## 1 Introduction

While the full-colour display is becoming more and more common, bit-mapped CRTs remain commonplace as well. These have advantages in terms of speed, resolution, and cost that cannot be matched by colour displays. Occasionally it is necessary to display an image on such a device. Moreover, certain colour-table animation techniques rely on the use of single bit-planes of a full-colour display. Here the full colour display is being used to simulate a bit-mapped display with a very fast frame update rate. A common method of converting from full-colour continuous tone to black and white binary is to error diffuse the luminance component. Various forms of error diffusion have been suggested[7, 14, 6, 8, 3, 15, 13]; the particular choice of error diffusion technique has relatively little effect on the appearance of an image when it is displayed on a sufficiently high-resolution monitor.

†Xerox Corporation, Webster Research Center, Building 128–27E, 800 Phillips Road, Webster, NY 14580.

‡College of Computing, Georgia Institute of Technology, 801 Atlantic Drive, Atlanta, GA 30332–0280

 ACM-0-89791-601-8/93/008/0053 $01.50

The value of gamma-correction of colour displays (or better still instrumented compensation)[2, 4], is well known. On a bit-mapped display the concept of gamma-correction is meaningless. As Naiman has noted, CRTs exhibit spatial non-linearities ([10], pp39–48), as can be easily seen by displaying a checkerboard of period two pixels adjacent to a checkerboard of twice that period. When viewed from a sufficient distance to cause the coarser checkerboard to appear smooth, these two images should in principle appear the same intensity. On most output devices they do not. (An LCD display may be an exception).

Much has been said about correcting for neighbourhood effects in prints. Commonly, it is based on a simple model of circular pixels with greater than unit area [12, 1, 11, 5]. For the SIGGRAPH audience, two more important display devices are the CRT and the film recorder. We begin with the simplest example: the bit-mapped CRT. Bit-mapped CRTs are so common that most readers of this paper are likely to have one. The improvement can be quite striking, as shown by figures 1 and 2. A linearity assumption (ie. that the phosphors are not saturated) allows the extension to greyscale and colour monitors, and to film recorders.

## 2 SIMPLE CRT CORRECTION

The general idea behind neighbourhood-based compensation is that the intensity generated at a pixel depends not only on the setting of that pixel but also on the intensity of the neighbouring pixels. The CRT is a special case. Here the non-linearities are primarily in the amplifiers driving the electron gun(s), so it is sufficient to consider only the left and right neighbours (whichever have been visited). An isolated pixel does not contribute as much intensity as it would with its neighbour on. (The amplifiers aren't fast enough to turn the electron beam on and off in one pixel). Neighbours in adjacent scanlines have no effect under this assumption.

To test the assumption of independent scanlines, display four images: with a) alternate scanlines b) alternate columns c) alternate pairs of scanlines, and d) alternate pairs of columns intensified. If scanlines are independent, a) and c) should have the same intensity. In the unlikely event that b) and d) appear the same, the monitor has excellent high frequency response, and no correction is necessary. If flicker causes a problem with interlaced displays when displaying single scanlines it can be alleviated by using a checkerboard and changing only the vertical frequency.

The second assumption is one of single neighbours contributing. This can be tested using a pattern of decreasing frequency vertical lines. For the (SONY) monitors we tested, the difference between single and double pixel lines was much greater than that between double and triple pixel width lines, so the assumption appears safe.

**Figure 1** A radiosity-like scene, error diffused without correction. Note the dark band in the shadow.

To correct for the presence or absence of a neighbouring pixel, the algorithm in the CRT case is as follows:

```
for each pixel
    if no neighbouring output pixel is on (white)
        if value (including errors passed in) > threshold − δ
            set the pixel
            quantization error = value − ( 1 − δ )
        else value ≤ threshold − δ
            quantization error = value
    else a neighbouring output pixel is on
        if value (including errors passed in) > threshold
            set the pixel
            quantization error = value − 1
        else value ≤ threshold
            quantization error = value
    Diffuse quantization error in the normal way
```

If there is no neighbouring pixel on, the effect of turning the current pixel on is reduced. This is reflected both in the turn-on decision, and in the calculation of the quantization error.

The specification deliberately leaves open the choice of error diffusion algorithm, including the order in which pixels are visited. Left and right neighbours are treated equally, although in reality pixels are only affected by the state of their left neighbours. The result of processing some pixels in right to left order, rather than left to right, results in the same average intensity overall, with a slight phase shift.

The value of $\delta$ must be determined experimentally; to do so, display a checkerboard containing $2\times2$ squares adjacent to a region of mid-grey that has been error diffused using the modified error diffusion algorithm. Vary $\delta$ across the error diffused region (Figure 3), and find the point where the two regions have the same luminance. We have found values in the 5–30% range apply to the monitors we tried. Figure 4 is a photograph of a screen with the pattern of Figure 3 displayed on the screen. The crossover point on the screen photographed is about midway across the figure (the process of photographing and printing the image may have changed the crossover point in the picture).

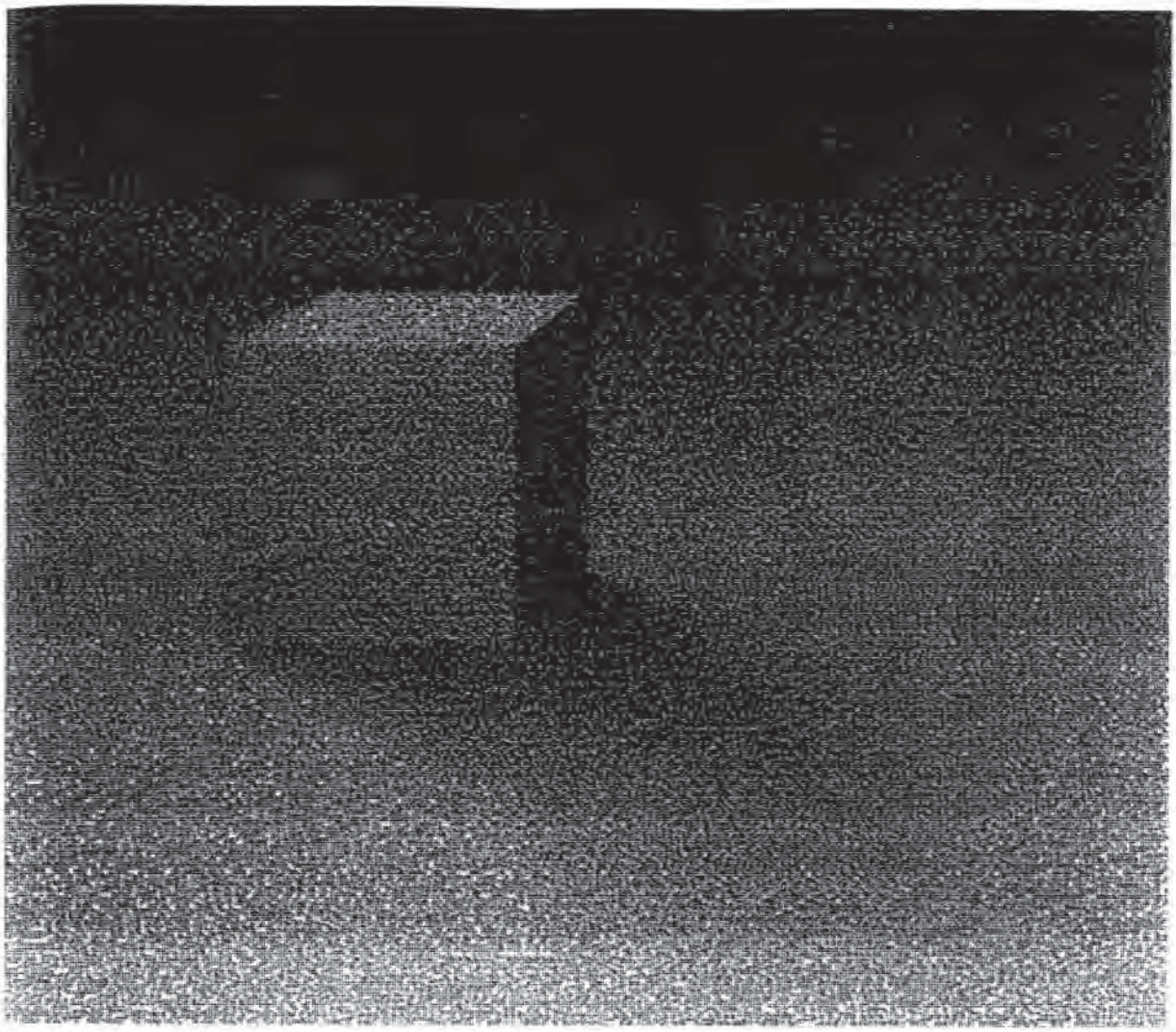


**Figure 2** After correction the shadow fades smoothly through its penumbral region.

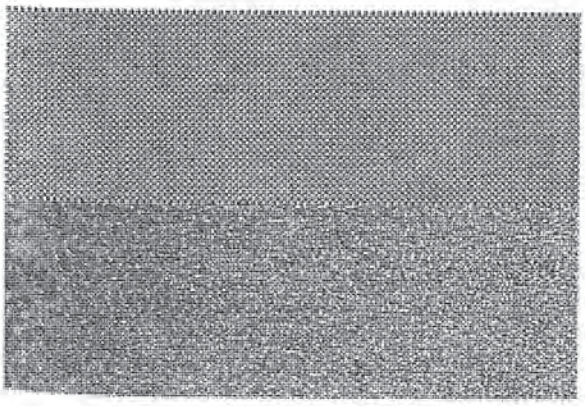


**Figure 3** Finding the value of $\delta$. The checkerboard has period two pixels. Below is an error diffused version of a 50% grey with $\delta$ varying from 0 (at the left) to 20%. Figure 4 shows the result of displaying this pattern on a CRT.

## 3 GREY SCALE MONITOR OR FILM RECORDER

The difference between a bit-mapped monitor and a greyscale one is the frame buffer behind it. Both employ an electron beam directed at phosphors; the spatial non-linearity effects are identical. As long as images displayed on greyscale monitors do not have high frequency information in them, their spatial non-linearities will be hidden. Where high contrast edges appear, the non-linearities can affect image quality. Fortunately, spatial non-linearities due to gun amplifier non-linearity are close enough to intensity invariant that the methods above can be safely generalized.

Before proceeding to correct for spatial non-linearities, it should be ascertained that the monitor is corrected for gun non-linearities. Given an otherwise corrected monitor, the value of $\delta$ can be determined as above, using patterns of full-on, full-off.

It is not normal to error diffuse images unless the display is operating from a low depth frame buffer (eg. 8 bits for all three components). If it is, the error diffusion algorithm can be adjusted in the same way as described above. In the typical

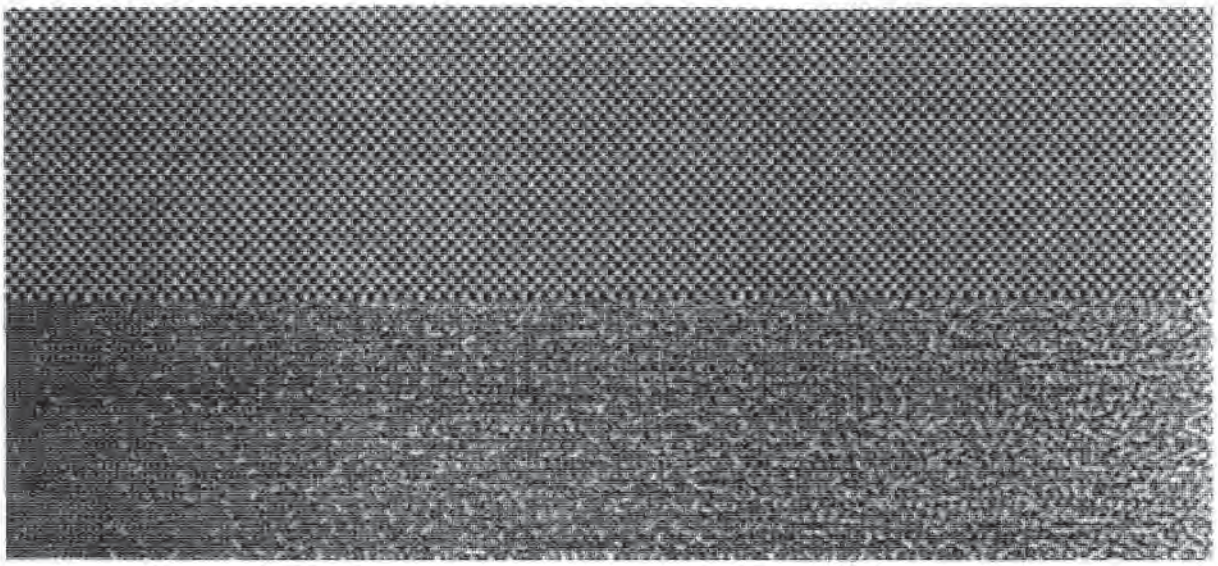


**Figure 4** The result of displaying a pattern similar to that shown in Figure 3.

case of a 24 (or higher) bit frame buffer, error diffusion can still be applied, without the quantization step. Normally there would be no error generated, but the alteration to the input values can still be applied, possibly generating out-of-gamut values. For example, a white pixel immediately followed by a black pixel would lead to a request for a negative pixel value for the second one. A remapping of the input (reducing the contrast) can prevent such negative pixel values entirely. A partial contrast reduction can make such negative pixel values infrequent. This is similar to eliminating phosphor trails in temporally varying displays, as described in [9]

## 4 SUMMARY & CAVEAT

We have described a simple technique for improving the tonal reproduction accuracy of CRTs. For bit-mapped displays, it serves the usual function of gamma correction. For regular CRTs it performs in image regions of high spatial frequency what gamma correction or instrumented compensation does in image regions of low spatial frequency. The method involves very little extra computation over that required for conventional error diffusion, and is simple to implement and calibrate. It should be noted that the generalization to print is complicated by the larger neighbourhoods affecting pixels, two (spatial) dimensional interactions, and non-linear colour mixing in the case of coloured printing.

## REFERENCES


[1] Allebach, J. Binary display of images when spot size exceeds step size. *Applied Optics 19*, 15 (August 1980), 2513–2519.

[2] Catmull, E. A tutorial on compensation tables. *Computer Graphics 13*, 2 (1979), 1–7.

[3] Cole, A. Naive halftoning. In *CG International '90* (1991), Springer-Verlag, pp. 203–222.

[4] Cowan, W. An inexpensive scheme for calibration of a colour monitor in terms of CIE standard coordinates. *Computer Graphics 17*, 3 (1983), 315–321.

[5] Dong, C.-K. Perceptial printing of gray scale images. Master's thesis, MIT, 1992.

[6] Fawcett, G., and Schrack, G. Halftoning techniques using error correction. In *Proceedings of the SID* (1986), vol. 27, no. 4, pp. 305–308.

[7] Floyd, R., and Steinberg, L. An adaptive algorithm for spatial gray scale. In *Society for Information Display 1975 Digest of Technical Papers* (1975), pp. 36–37.

[8] Griffiths, J., and Yang, C. Algorithms for generating improved images of curved surfaces by distributing errors along Hilbert's curve. *Computer-Aided Design 19*, 6 (July 1987), 299–304.

[9] Klassen, R. *Device Dependent Image Construction for Computer Graphics*. PhD thesis, University of Waterloo, 1989. Available as technical report #CS-91-19.

[10] Naiman, A. *The Use of Grayscale for Improved Character Presentation*. PhD thesis, University of Toronto, 1991.

[11] Pappas, T., and Neuhoff, D. Model-based halftoning. In *Proc. SPIE/IS&T Symposium on Electronic Imaging Science and Technology, Human Vision, Visual Processing, and Digital Display II* (1991).

[12] Roetling, P., and Holladay, T. Tone reproduction and screen design for pictorial electrographic printing. *Journal of Applied Phot. Eng. 15*, 4 (1979), 179–182.

[13] Velho, L., and Gomes, J. Digital halftoning with space filling curves. In *Proceedings SIGGRAPH '91* (1991), pp. 81–90.

[14] Whitten, I., and Neal, R. Using Peano curves for bilicvel display of continuous tone images. *IEEE Computer Graphics and Applications 202* (May 1982), 47–52.

[15] Wyvill, G., and McNaughton, C. Three plus five makes eight: a simplified approach to halftoning. In *CG International '91* (1991), Springer-Verlag, pp. 397–392.



# Pad
## An Alternative Approach to the Computer Interface

Ken Perlin
David Fox

Courant Institute of Mathematical Sciences
New York University
719 Broadway 12th Floor
New York, NY 10003

## Abstract

We believe that navigation in information spaces is best supported by tapping into our natural spatial and geographic ways of thinking. To this end, we are developing a new computer interface model called Pad.

The ongoing Pad project uses a spatial metaphor for computer interface design. It provides an intuitive base for the support of such applications as electronic marketplaces, information services, and on-line collaboration. Pad is an infinite two dimensional information plane that is shared among users, much as a network file system is shared. Objects are organized geographically; every object occupies a well defined region on the Pad surface.

For navigation, Pad uses "portals" - magnifying glasses that can peer into and roam over different parts of this single infinite shared desktop; links to specific items are established and broken continually as the portal's view changes. Portals can recursively look onto other portals. This paradigm enables the sort of peripheral activity generally found in real physical working environments. The apparent size of an object to any user determines the amount of detail it presents. Different users can share and view multiple applications while assigning each a desired degree of interaction. Documents can be visually nested and zoomed as they move back and forth between primary and secondary working attention. Things can be *peripherally* accessible.

In this paper we describe the Pad interface. We discuss how to efficiently implement its graphical aspects, and we illustrate some of our initial applications.

## 1 Introduction

Imagine that the computer screen is a section of wall about the size of a typical bulletin board or whiteboard. Any area of this surface can then be accessed comfortably without leaving one's chair. Imagine further that by applying extraordinarily good eyesight and eye-hand coordination, a user can both read and write as comfortably on any micron wide section of this surface as on any larger section. This would allow the full use of a surface which is several million pixels long and high, on which one can comfortably create, move, read and compare information at many different scales.



ACM-0-89791-601-8/93/008/0057 $01.50

The above scenario would, if feasible, put vast quantities of information directly at the user's fingertips. For example, several million pages of text could be fit on the surface by reducing it sufficiently in scale, making any number of on-line information services, encyclopedias, etc., directly available. In practice one would arrange such a work surface hierarchically, to make things easier to find. In a collaborative environment, one could then see the layout (in miniature) of many other collaborators' surfaces at a glance.

The above scenario is impossible because we can't read or write at microscopic scale. Yet the concept is very natural since it mimics the way we continually manage to find things by giving everything a physical *place*. A good approximation to the ideal depicted would be to provide ourselves with some sort of system of 'magic magnifying glasses' through which we can read, write, or create cross-references on an indefinitely enlargeable ('zoomable') surface. This paper describes the Pad interface, which is designed using these principles.

### 1.1 Overview of the Paper

We begin section one with a brief summary of the basic ideas and components of the Pad Model. We then finish section one with a comparison of Pad to the window/icon paradigm and a summary of prior work. Section two is a description of a typical Pad application, and section three covers the principles of the Pad system. Section four covers several issues in our implementation of Pad, and section five lists some ongoing and future projects. Finally, section six presents our conclusions and acknowledgments.

### 1.2 Basic Pad Model

The **Pad Surface** is an infinite two dimensional information plane that is shared among users, much as a network file system is shared. It is populated by **Pad Objects**, where we define a Pad Object to be any entity that the user can interact with (examples are: a text file that can be viewed or edited, a clock program, a personal calendar). Pad Objects are organized geographically; every object occupies a well defined region on the Pad surface.

To make themselves visible, Pad Objects can create two types of "ink", **graphics** and **portals**, and place them on the Pad Surface. A graphic is simply any sort of mark such as a bitmap or a vector. Portals are used for navigation, they are like magnifying glasses that can peer into and roam over different parts of the Pad Surface. A portal may have a highly magnified view or a very broad, panoramic view, and this view can be easily changed. The screen itself is just a special "root" portal.

A portal is *not* like a window, which represents a dedicated link between a section of screen and a specific thing (e.g.: a Unix shell in X-Windows or a directory in the Macintosh Finder). A

portal is, rather, a view into the single infinite shared desktop; links to specific items are established and broken continually as the portal's view changes. Also, unlike windows, portals can recursively look onto (and into) other portals.

Figure 1 shows a very large financial document on the Pad surface. The small portal at the top of the figure shows an overview of the entire report. The two other portals show successive closeups of portions of the report.

### 1.3 Object/Portal Interaction

A Pad object may look quite different when seen through different portals. There are two techniques that allow objects vary their appearance: *semantic zooming* and *portal filters*.

Every object visible on the screen has a magnification that depends upon the sequence of portals it is being seen through. As the magnification of an object changes, the user generally finds it useful to see different types of information about that object. For example, when a text document is small on the screen the user may only want to see its title. As the object is magnified, this may be augmented by a short summary or outline. At some point the entire text is revealed. We call this **semantic zooming**.

Semantic zooming works using the *expose event*, which says that a particular portion of the Pad Surface will be rendered at a particular magnification. When an object receives this event it generates the display items needed to give an appropriate appearance at that magnification.

Objects can also manage **portal filters** - portals that show non-literal views of cooperating objects. For example, a portal may show all objects that contain tabular data as a bar chart, but display other objects as would any other portal. This would enable an application to embed a bar chart within a document by placing in it a portal filter that looks onto an object that contains tabular data. Another application can then allow text or spreadsheet style editing of the tabular data itself by some user. These edits will be seen as changes in the bar chart by any user who is looking at the document.

The effect is that the bar chart filter portal will "see" any tabular data as a bar chart, but will see other objects in the usual way. Portal filters work by intercepting the expose event for objects which it knows how to render. It then asks the object or objects for any information it needs to create the display items to render them.

Another interesting portal filter would be a control modifier. Imagine for example that a paint program has several types of brush. Normally one would click on an image of a particular brush to select it. When seen through a control modifier portal filter, each brush image would appear as a panel of parameter controls with which the user can change that brush's internal state (width, spattering law, etc). The *same* portal filter could be used to modify the controls of any application on Pad that recognizes its message conventions.

### 1.4 Pad vs. the Window/Icon Paradigm

An important distinction between the Pad universe and the universe of other window systems is that in Pad every interaction object possesses a definite physical location. In this sense Pad is a two dimensional virtual reality. Yet a user's changing view can allow objects to *appear* larger or smaller.

This paradigm allows for the sort of peripheral activity found in real physical working environments. Each object on a user's screen commands a degree of attention commensurate with how big the object appears to that user. This allows each object to vary the amount of detail it presents to each user. Different users can share and view multiple applications while assigning to each one a desired degree of interaction. Documents can be visually nested and zoomed as they move back and forth between primary and secondary working attention. Things can be *peripherally* accessible.

For example, on the Macintosh desktop a user double clicks on a folder icon to see the contents of a directory in a window. But to see the contents of any folder within that folder, the user must double click to create a separate window.

In comparison, a user of Pad generally views a directory through a portal. The contents of any subdirectories are visible, in miniature, through sub-portals. This allows the user a peripheral awareness of a subdirectory's contents, without the user having to perform any explicit action. In this sense, Pad is better suited to non-command user interfaces[16].

### 1.5 Prior and related work

A number of researchers developed ways to visually structure interactive information that offer an alternative to windows/icons. One of the first such systems was the Spatial Data Management System [4] at MIT, which presented an information landscape on two screens: one screen for a panoramic overview and another (application) screen providing a closer view. The user could either pan locally around on the application screen or else could go directly to an area by pointing on the panoramic view.

On the other hand, Hypertext systems[15][10] allow the user to jump from one place to another in a conceptual information space. A notable problem with the current state of hypertext systems is the difficulty of knowing one's location in this space; unless the application is designed very carefully the user can easily get lost.

In other related work, many desktop publishing systems provide tiny "thumbnail sketches" of images that are stored on disk. To open an image file the user simply points to these miniature images instead of specifying a file name.

A unique approach to providing peripheral information has been developed by George Furnas at Bellcore Applied Research. His **Fisheye** user interface[8] shows information of current interest in great detail, while showing a progressively less detailed view of surrounding information.

Also, some of the components of fast image zooming have existed for a while. Williams[25] has used a pyramid of images for texture filtering, and Burt[2] for image processing, both based on the prior work of Tanimoto[22]. The **Bad Windows** interface[19] allows drawings to be accessed at multiple levels of detail.

Three dimensional interactive virtual offices that allow a user to change viewpoint are being developed by Mackinlay et. al. as well as Feiner [12][6]. Changes of scale have long been used in computer graphics for both entertainment and for scientific visualization.[3] One notable early example was the molecular simulation work of Nelson Max[13].

At Xerox PARC there has been a large body of interesting work on enabling groups to remotely share a common drawing surface for collaborative work.[11][14][21] This is part of their larger ongoing research effort in shared "Media Spaces"[1]. Similarly, the **Rendezvous** system at Bellcore is a general metasystem for building shared "conversational" interfaces for teleconferencing situations[9], as is the work of Smith et. al.[20]

## 2 An Example Application

The multiscale daily/monthly calendar is a study of "semantic zooming". Figures 2 through 4 show what the calendar looks

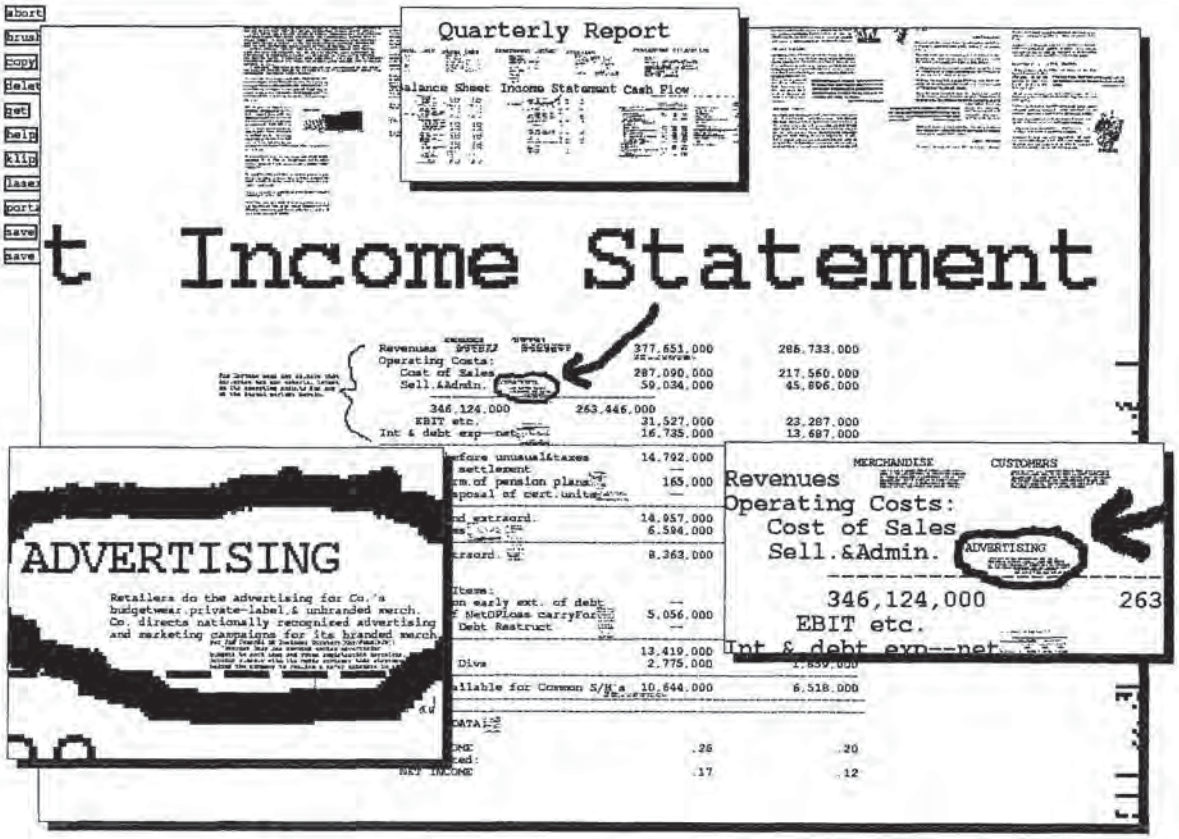



Figure 1: Quarterly report. Portals are views onto other parts of the Pad surface.

COMPUTER GRAPHICS Proceedings, Annual Conference Series, 1993

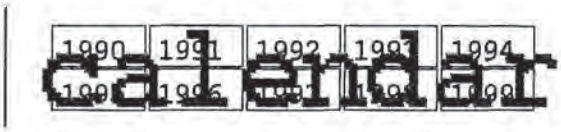



Figure 2: As you approach the calendar object the large scale display items fade out and disappear.

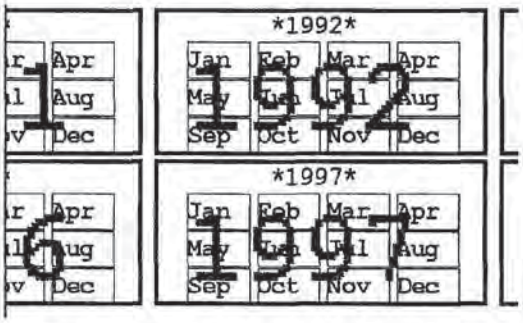



Figure 3: The calendar object generates smaller scale display items only for the area visible on the user's screen. Display items that are off the screen may be garbage collected and destroyed.

like at various successive magnifications. At any level, the user can type or draw on the calendar. As the user zooms away from the scale at which the annotations were drawn they become first translucent, then invisible. In this way, a user can overlay many levels of annotation on a calendar without confusion.

The major problem with an application of this type is that it can involve a large number of display items, since the spatial density of display items on the Pad grows geometrically as the user zooms into the calendar. Yet at any one time only a fairly small number of display items is visible, since as the user zooms in the screen occupies an ever smaller absolute area on the Pad.

We address this problem by designing the calendar object as an expandable semantic tree, and identifying display items with different nodes of this tree. Each time the calendar is displayed this semantic tree is traversed. As each node is reached, display items are generated as needed. Individual display items are ephemeral – if an item is off the screen for a while it is quietly removed by the calendar object. In this way the total number of display items always remains manageably small.

This general notion of a geographic database that will expand and self-prune as the user roams around the Pad has now been encapsulated in a Scheme library called an "ephemeral database manager". We plan to apply this library to other Pad applications that have an inherently tree structured semantics.

# 3 System Structure

In this section we introduce the abstract data types needed to implement Pad. First we will describe the concepts necessary for display, then those needed to support interaction.

## 3.1 Addresses and Regions

A Pad **address** $A = (x, y, z)$ has both a location and a scale, and defines the linear transformation $T_A : (u, v) \rightarrow (x+u2^z, y+v2^z)$. Here $z$ represents the $log_2$ of scale.

A Pad **region** $R = [A, w, h]$ is a rectangle defined by an address together with a raster width and height $(w, h)$. A region


Figure 4: The user's annotations are created in ink that also fades out at greater magnifications.

covers the portion of the Pad surface from $T_A(0,0)$ to $T_A(w,h)$, or from $(x, y)$ to $(x + w2^z, y + h2^z)$.

## 3.2 Display Items

The lowest level entities in the Pad universe are the *display items*, which come in two basic types: graphic and *portal*. Display items are the only entities actually visible on the user's screen. A *graphic* consists of a raster image $I$ and an address $A$. Every display item is said to have a region $[A, I_w, I_h]$, which is the portion of the pad surface which it occupies.

A *portal* is a graphic that has an additional address, called its *look-on* $L$. Using its raster image $I$ as a mask, a portal have as its "look-on" the region $[L, I_w, I_h]$ on the Pad surface. The portion of the Pad surface which the look-on covers and which is not masked by the portal's graphic is visible at the location of the portal's region. This raster masking enables a portal to give a shaped view onto the Pad surface. Thus, a portal can be square, round, or even shaped like some well known corporate logo.

We refer to a display item's $A_z$ as its "scale". In general, a display item becomes visible on the screen only after being viewed through a succession of portals, each of which may transform it. We refer to a display item's apparent $z$, as it is seen on the screen, as its "magnification".

The image on the user's screen is created from a set of display items. There is one portal associated with the user's screen called the "root portal"; the display process consists of rendering the root portal. This means rendering the region of the Pad surface which the root portal looks onto. Those display items that overlap the root portal's look-on are rendered. This procedure is then applied recursively to render any display item which is itself a portal.

As the display process recurses through each portal, the transformation $T(A)T^{-1}(L)$ is applied, where $A$ is that portal's address and $L$ is that portal's look-on. This recursion can be expanded to compute the location of any display item on the screen. Suppose item $i$ is viewed through successively nested portals $p_1 \ldots p_n$. Then to determine where (and at what magnification) to display $i$ on the screen, we apply the transforma-

tion:

$$T^{-1}(L_{root})T(A_{p_1})T^{-1}(L_{p_1})\ldots T(A_{p_n})T^{-1}(L_{p_n})T(A_i)$$

Incrementing the $z$ component of a display item's address will increase its magnification. Incrementing the $z$ component of a portal's look-on will double the size of its looked on region – and will therefore decrease the magnification of every item seen through it. (Think of it as increasing the viewer's altitude.)

There are several other properties of primitive display items which are important to note:

**Visibility Range:** Each graphic object can have a range of magnification outside of which it is invisible. This is important since most display items are only useful within a certain range of magnification.

**Transparency Range:** Similarly, each graphic can have a range of magnification outside of which the graphic is transparent. This allows objects to fade away gracefully as they are magnified up or down. Transparency is achieved by masking with a patterned pixel mask at screen resolution.

**Private Display Items:** Display items may be attached to a portal, in which case they are only visible when viewed through that portal and their addresses are relative to that of the portal. This creates a hierarchy of display items and is used to implement the filters described below.

### 3.3 Pad Objects

Graphics and Portals suffice to make an interesting multi-scale drawing program. However to use Pad as a system for building general user interfaces requires a higher level structure called a *Pad Object* to interpret events and control these display items so they behave as a single application. In Pad an object consists of a region together with a package of code and data which respond to event messages. An object's behavior is specified by the application developer. In order to make itself seen, each object manages a collection of display items, creating, modifying, and deleting them.

Pad Objects receive events from the user's mouse and keyboard, plus timer events, channel events (events representing other types of input, e.g. the output of a process), and expose events which inform the object that some portion of itself will become visible on someone's screen. Events which would normally have an x-y location have instead an *address*, and this address is transformed if the event passes through a portal before being received by an object which is interested in it. Similarly, an expose event covers a *region* rather than just a rectangle, and this region is also transformed by portals so that each object can be informed which portion of its region will be rendered and at what magnification.

Objects are maintained in an order, just as display items have a drawing order, so that if two or more objects are at the mouse address the mouse events are sent to the one in front. The object may use this event for its own purposes, or it may pass the event on to the objects behind it, or it may transform the event's address and pass it on to some other part of the Pad. Events thus passed may go unused by the objects below, in which case the original object may then use the event for its own purposes.

### 3.4 Display

Display is complicated by the fact that objects may be continually creating and destroying display items. Before we can create the display we first need to give each object an opportunity to know at what magnification it will be called upon to appear, since this will probably influence what display items it chooses to show.

Therefore display is a two phase process. In the first phase, each object gathers all the necessary information about what portions of it will appear on the screen and at what magnifications. During this first phase display items may be spawned. In the second phase the screen image is actually drawn.

During phase one each portal is displayed by having the Pad object that controls it communicate with all objects that intersect the portal's look-on region. This process begins with a special root object, which controls the user's root portal. For a portal controlled by an object $O_1$ the procedure is as follows:

- $O_1$ sends an expose event for the portal's look-on region. This event will be received by all objects whose regions intersect the portal's look-on region.
- for each object $O_2$ that responds:
  - $O_1$ tells $O_2$ to produce display items for itself with the proper magnification and clip. If $O_2$ controls any portals, the procedure is invoked for them recursively.
  - any display items that $O_1$ receives back, it attaches to the portal.

This process continues recursively until all items large enough to see on the screen are accounted for.

In the second phase, each portal is painted from its accumulated list of display items. This process starts with the root portal, and continues on through all portals seen by the root portal, and then recursively through those portals. Note that if two portals on the screen have overlapping look-on regions, their lists may have display items in common.

### 3.5 Interacting Objects and Portals

Semantic zooming is implemented by having the object's display method depend upon its magnification. The object is always told its magnification during display phase one.

Portal filters are implemented as follows. Consider the case of the bar chart filter portal described earlier. Suppose this portal filter is managed by object $O_1$. During phase one of the portal display procedure, $O_1$ sends an expose event for this portal, and receives a number of acknowledgments. Suppose $O_1$ has just received such an acknowledgment from object $O_2$. $O_1$ queries $O_2$ to find out whether $O_2$ is a tabular object. If yes, then $O_1$ gets the tabular data from $O_2$, builds its *own* display items for the bar chart, and attaches these to the portal. If no, then $O_1$ asks $O_2$ to produce a list of display items as usual. The effect is that the filter portal will "see" any tabular data as a bar chart, but will see other objects in the usual way.

## 4 Implementation Details

The Pad system is written in three layers, a real-time display layer written in C++, a Scheme interpreter providing an interface to the C++ layer, and a collection of Scheme code implementing the Pad application interface. It currently runs under X Windows and MS-DOS. The X Windows version has been compiled and run on SunOS, AIX and Linux. The source code of the most recent released version is available via anonymous FTP from cs.nyu.edu in the directory pub/local/perlin.

### 4.1 Rendering Display Items

It is absolutely essential to our system that arbitrarily scaled bitmaps can be displayed in real time. Without an algorithm to achieve this, our desktop model would either require special purpose hardware, or else would lose real-time response. Either scenario would limit the model's general usefulness on typical currently available graphical workstations. The method we use to render the raster image of a graphic item depends upon the item's magnification. The following decisions are based on our trial and error experiences; they reflect our best results in "tuning" this process.

We use four different techniques for drawing the raster image of a graphic, depending on the range of magnification $m$.

- $m > 16$. At the largest magnifications it is quickest to simply draw individual filled squares for each pixel.
- $1 > m \geq 16$. At moderate magnifications we use look up tables indexed by the byte pattern, amount of magnification, and bits of shift to properly position the result within the destination word. Different tables are used depending on the depth of the image.
- $m = 1$. With no magnification we only need to worry about the amount of shift necessary to position the result.
- $\frac{1}{1024} \leq m < 1$. To demagnify images we index into a precomputed pyramid of images.[25] This precomputation is done at the time a graphic is created; it creates about a 3/2 speed penalty to that process. Since graphic items are generally reused over many screen refreshes, this penalty is not usually a problem in practice.
- $m < \frac{1}{1024}$. Beyond some amount of demagnification the bitmap is not visible and need not be drawn at all.

These techniques yield a display time for each object approximately proportional to the size of the entire screen image. In practice this tends to keep refresh time dependent only upon screen resolution, not upon image complexity.

### 4.2 Address Space Limits

Addresses are implemented using floating point arithmetic, so we cannot claim an "infinite" address space for our current system. A true unbounded address space could be achieved by using extended integer arithmetic. Even in its current form, the space provided is astronomical. Suppose our numbers have a 48 bit mantissa and we have a $2^{12}$ by $2^{12}$ screen. To position an object on the screen uses 12 of those 48 bits, leaving a minimum of 36 bits of precision to position our look-on anywhere within the square $-1 \leq x, y \leq 1$. This means, for example, that you could lay out $2^{36}$ by $2^{36}$ pages of text in that area.

## 5 Ongoing and Future Work

### 5.1 Shared Object Space

Perhaps our most important goal is to create a truely distributed Pad system, where Pad objects can exist on remote machines and can *migrate from machine to machine*. When Pad objects are distributed over many computers the problem of updating the display of a region on one's screen becomes a combined distributed database and computational geometry problem. This is the subject of ongoing research,[7] and is beyond the scope of this paper. For in-depth discussions of the implementation problems we refer the readers to Preparata & Shamos[18] for an overview of computational geometry and to Edelsbrunner[5] for an optimal data structure for rendering.

### 5.2 Continuous Zoom

Early prototypes have used discreet zoom levels to achieve high performance. We have also implemented a continuous zoom algorithm (based on Bresenham's midpoint line drawing algorithm) that allows continuous scaling of raster images at approximately half the speed of discreet zooming on unenhanced bitmapped workstations. The algorithm uses table lookups to greatly speed up the calculation.

### 5.3 Hierarchical Text Editor

A number of generalizations of familiar applications to the hierarchical domain suggest themselves. A multiscale text editor is a generalization of a traditional text editor, with the added capabilities that text can appear at many different sizes, with recursively inserted text. Therefore the screen structure is no longer a two dimensional array - it is more like a set of nested boxes. This allows a more direct look-and-feel for hypertext - footnotes and references can be embedded in their entirety at the point of reference. Successive zooming by the user gradually expands the contents seen of the work referenced. Text is structured as hypertext – a text string may contain embedded links to other text strings. The structure of the document can be an arbitrary directed graph. Visually, text that is linked to appears to be at the location of the link, only smaller. Contents of a hyperlink can be accessed without a disruptive sudden change in the view of the text that references it.

Text can also be made semantically zoomable: When text is visibly small it appears only as a title. As the user zooms in, this expands to include an abstract. Further zooming reveals first an outline with short text descriptions, then finally the full text.

There are several options for where exactly to visually place linked-to text. The text can appear in miniature either beneath the lines of parent text or, alternatively, superimposed on the parent text. The latter option requires zoom-dependent translucency. As the user zooms in, text seen through hyperlinks "fades up" and the visually larger text that references it simultaneously "fades out".

Text can be visible simultaneously in any number of portals. Each view must maintain a certain amount of state information. For example, there needs to be a cursor for each view. This means that if the mouse is over a particular portal, and the user types, the insertion point is at the cursor of that view. Since portals can contain ownership attributes, they can be used to restrict access to parts of a document. Text visibility through any particular portal depends upon the text's ownership - public (shared by many users) or private (seen by only one user). Public text can contain links to private text. In general, the visibility attributes of text can vary, depending upon whether the text is being viewed by its owner or by someone else.

### 5.4 An Infinitely Scalable Painting Program

We have, together with Luis Velho, begun applying multiscale principles to an infinitely detailable painting program[17]. Organizing an infinite multiscale canvas is straightforward, requiring only a Quad-tree. Unfortunately, simulating the application of a paint brush requires a compositing operation - an *alpha* blending of the underlying image with the brush image.

Since this operation is non-commutative, it is easy to run into problems. For example, Let's say the user zooms way in to paint


Figure 5: Overview of branching tree story. The story begins with a single sentence. The branches of the tree represent story paths - as the reader zooms into different branches, different stories unfold.

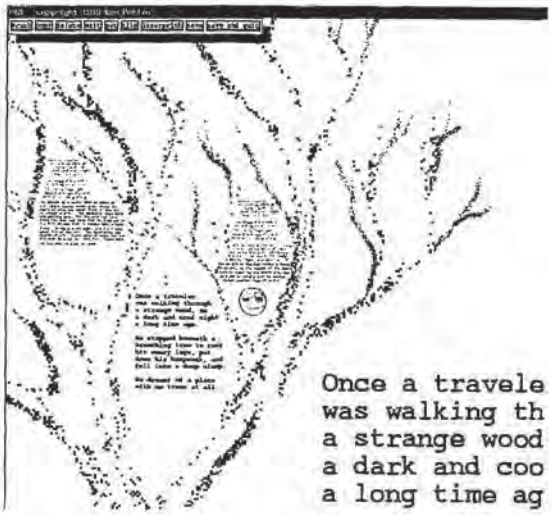



Figure 6: One level of zoom into branching tree story. At this scale the narrative contains one or two paragraphs of detail.

a scene at a fine scale, then pulls out to paint an atmospheric wash at a coarse scale, and finally zooms back in to touch up fine scale details. How should the system implement this? A straightforward approach, used by Williams[26], is to immediately apply the coarse scale operations to the finer level pixels. But this is computationally prohibitive for highly scaled scenes, since the number of fine scale pixels affected grows exponentially with the difference between coarse and fine scale.

Clearly a pyramid of some kind is called for. But because of non-commutativity, successive operations at different levels cannot be separated into a traditional Laplacian or similar multilevel pyramid (as they could be in, say, a strictly additive system). Our solution is to use B-spline wavelets. We break the brush image into its component wavelet basis, and apply independently at each level of a wavelet basis pyramid. Then the B-spline wavelet reconstruction will produce the correct result. We have implemented this to a one-dimensional canvas, and are now working on a two or more dimensional version.

### 5.5 Multiple Narrative Paths

Pad is a good way to store documents with hierarchy and multiple narrative pathways. Side discussions in a textbook can be embedded *in situ*. This allows for some interesting possibilities. For example, a novel may be written with bifurcations, allowing its reader to explore many interleaving stories - a sort of visual *Alexandria Quartet*. For example, we have been creating a user browseable novel literally shaped into a tree, as seen in figures 5, 6, 7.

### 5.6 Cooperative Pad Applications

With the onset of high bandwidth consumer information services, Pad provides a viable look-and-feel for information browsing. As the customer zooms in to an information service, the semantic zoom level (and hence the information content) increases. Zoomed-down browsing can be made freely available, and the customer can be billed at successively higher rates for more specific data.

For example, the title and a brief synopsis of a video may be accessible at low zoom levels. Higher zoom levels actually play the movie. At the browsing level, the customer might see geographically arranged clusters of films that may be of related interest (e.g. films by a particular director).

Similarly, our **Pad Map** project will provide a substantial user community with access to a shared map of Manhattan, annotated with information about cultural events. The users will be able to add their own annotations, such as restaurant or movie reviews, or just graffiti. As part of the Pad system, annotations could be at any scale, and contain links to other annotations: though it is desirable to keep all the reviews of a given film together, portals could make them visible at each theatre which is showing that film. The project will explore the mechanisms necessary manage user contributions without any one user monopolizing or degrading the system for others.

Our **Shared Spreadsheet** project re-casts the spreadsheet application in a more hierarchical and sharable form. For example, hierarchy can be imposed by placing spreadsheet $A$ in a cell of spreadsheet $B$, and designating a particular cell of $A$ to be the value that appears in $B$'s cell when the magnification of $A$ is low. The value of sharing such a spreadsheet among users comes from immediate access to the latest data, and the elimination of the need to merge copies of the spreadsheet which have been updated independently, etc.

Eventually, as display and communication technology improves, pieces of display surface scattered around a work environment will become more common – on walls, desks, electronic PostIt$^{tm}$ notes[24]. Pad is well suited to such a distributed environment, since it places the user at a floating location in an information geography. The Windows/Icon/Menu/Pointer model is less well suited to this, since it is motivated by the desire to create a "desktop" metaphor on a single display screen.

## 6 Conclusions

We have described a new kind of graphical space that has a number of advantages over traditional window systems. Its key advantage is that it allows a user or a group of users to share and view multiple applications in a manner that assigns them vari-

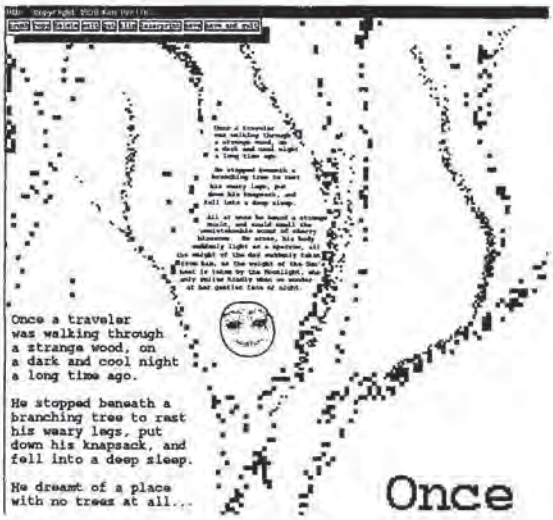



Figure 7: Two levels of zoom into branching tree story. Here we can see the story beginning to take a definite shape - in one possible narrative path.

ous levels of importance, with easy visual nesting and zooming of documents as they move from peripheral to primary working attention.

As compared to standard current window models, this system makes it easier for the user to exploit visual memory of places to organize informationally large workspaces.

We believe that this approach enriches the workstation/window paradigm in a fundamental way.

### 6.1 Acknowledgments

This research was funded by a grant from the NYNEX Corporation and by NSF grant number IRI-9015445. We would like to thank Nathan Felde at NYNEX for the initial discussions leading to this work, and Jack Schwartz, Lorie Loeb, Raj Raichoudhury, Allison Druin, and Gene Miller, all of whom contributed valuable ideas and time, as well as the Apple corporation for their generous equipment donation. Particular credit goes to Matthew Fuchs, who is developing the Distributed Pad/Scheme system DREME.

## References

[1] Sara Bly et. al., *Media Spaces: Bringing People Together in a Video, Audio, and Computing Environment.*, CACM, Vol. 36, 1993, No. 1., pp. 28-47.

[2] Peter Burt, *A multiresolution spline with applications to image mosaics*, ACM Transactions on Graphics, Vol. 2, No. 4, Oct. 1983, pp. 217-236.

[3] James H. Clark. *Hierarichical geometric models for visible surface algorithms.* ACM Communications, Vol. 19, No. 10, Oct. 1976, pages 547-554.

[4] William C. Donelson, *Spatial Management of Information*, ACM SIGGRAPH 1978 Conference Proceedings.

[5] H. Edelsbrunner, *A new approach to rectangle intersections, Part II*, Int'l Journal of Computational Mathematics, No. 13, pp. 221-229, 1983.

[6] S. Feiner and C. Beshers, *Worlds within worlds: Metaphors for exploring n-dimensional virtual worlds.* Proc. UIST '90 (ACM Symp. on User Interface Software and Technology), Snowbird, UT, Oct. 3-5, 1990, pp. 76-83.

[7] Matthew Fuchs, unpublished Ph.D. dissertation in progress.

[8] George Furnas, *Generalized Fisheye Views, Human Factors & Computer Systems*, CHI 89 Conference proceedings, pp. 16-23.

[9] Ralph Hill, et. al., *The Rendezvous Language and Architecture*, CACM Vol. 36, 1993, No. 1., pp. 62-67.

[10] *Hypertext on Hypertext*, Macintosh Version: Disk #1 and #2. ACM Press, New York, 1988.

[11] I. Lu et. al., *Idea management in a shared drawing tool.* Proceedings of the Second European Conference on Computer-Supported Cooperative Work-ECSCW '91, Amsterdam, Holland, 1991.

[12] J. Mackinlay et. al., *Rapid Controlled Movement Through a Virtual 3D Workspace.* ACM SIGGRAPH 1990 Conference Proceedings.

[13] Nelson Max, ACM SIGGRAPH 1975 Film show.

[14] Minneman, S.. and Bly, S.a. *Managing a trois: A study of a multi-user drawing tool in distributed design work* Proceedings of the CHI'91 Conference on Human Factors in Computer Systems., New Orleans, La., 1991.

[15] Ted Nelson, *Literary Machines.* Swarthmore, PA, 1981.

[16] Jakob Nielsen, *Non-command User Interfaces*, CACM, Vol. 36 No. 4, (April 1993), pp. 83-99.

[17] Ken Perlin and Luis Velho, *A Wavelet Representation for Unbounded Resolution Painting*, NYU Technical Report.

[18] Franco P. Preparata, Michael Ian Shamos, *Computational Geometry: An Introduction*, Springer Verlag, New York, 1989.

[19] David Small, Masters Thesis, MIT Media Laboratory, 1989.

[20] Randall B. Smith, Tim O'Shea, Claire O'Malley, Eileen Scanlon, and Josie Taylor. *Preliminary Experiments with a distributed, multi-media, problem solving environment.* In Proceedings of the First European Conference on Computer Supported Cooperative Work (Gatwick, UK) 1989, pages 19-34.

[21] J.C. Tang and S.L. Minneman, *Videodraw: A video interface for collaborative drawing.* Proceedings of the CHI '90 Conference on Human Factors in Computing Systems, Seattle, Wash., 1990.

[22] S. L. Tanimoto, and T. Pavlidis, *A hierarchical data structure for picture processing.* Computer Graphics and Image Processing, Vol. 4, 1975, pp. 104-119.

[23] Edward Tufte, *The Visual Display of Quantitative Information*, Graphics Press, 1983.

[24] M. Weiser *The Computer for the 21st Century*, Sci. Am. 265,3 (September 1991), pp. 94-104.

[25] Lance Williams, *Pyramidal Parametrics.* ACM SIGGRAPH 1982 Conference Proceedings.

[26] Lance Williams, personal communication.



# Autocalibration for Virtual Environments Tracking Hardware *

Stefan Gottschalk
Computer Science Department
University of North Carolina
Chapel Hill, NC
gottscha@cs.unc.edu

John F. Hughes
Computer Science Department
Brown University
Providence, RI
jfh@cs.brown.edu

## Abstract

We describe two instances in which precise mechanical calibration of virtual environments equipment has been replaced by automated algorithmic calibration through software that encapsulates the hardware design and uses a goal-based approach to adjust calibration parameters. We describe a back-projection system for adjusting the assumed locations of beacons in a head-mounted display tracking system; the calculated errors in the navigation system are used to compute adjustments to the beacon positions to reduce such errors. In a second application, a piggyback head-tracking/hand-tracking system is calibrated by a similar reduction of computed errors.

**CR Categories:** I.3.m [Computer Graphics]: Miscellaneous; I.3.7 [Computer Graphics]: 3-dimensional Graphics and Realism — Virtual Reality; I.4.8 [Image Processing] Scene Analysis — Photometry

**Additional Keywords:** Virtual environments, tracking, autocalibration.

## 1 Introduction

A number of calibration issues for virtual environments (VE) hardware are approached with standard engineering techniques in which the accuracy of the calibration is directly dependent on the accuracy of the assemblies in the VE machinery. This approach is successful to a degree but has several drawbacks. First, it makes the machinery very sensitive to rough handling. Second, frequent realignment may be required, which may be time-consuming and may be necessary so frequently that extended use of the equipment becomes impossible. Third, modifications of the machinery become very difficult.

We therefore take a *goal-based* approach to these problems, applying methods learned in computer graphics to solve engineering problems. Instead of requiring precise calibration of parts, we ask the systems to autocalibrate, a notion that was inspired in part by the auto-assembling systems of Barzel and Barr [BB88] but which first appeared in Wang's dissertation [Wan90]. This allows us to write a program encoding the *design* of the system that uses the system's observations to adjust itself. Since realignment can sometimes actually be done while the machinery is in use, rather than in a separate calibration phase, the first and second problems above are reduced. And because the software that implements the autocalibration encodes the intent of the design, the mechanical design can be modified in parallel with software modification, helping to reduce the third problem. In this paper, we discuss two sample applications: calibration of a head-tracking system and of a piggyback hand tracker attached to the head-tracking unit.

We stress that the techniques here serve the general goal of head-tracking. The current interest in Virtual Reality, evidenced by the attention it has attracted in both the technical literature and the media, may well have led to unjustified expectations. There is a belief that "any day now" the technology will become available. But there are three substantial obstacles: (1) for comfort, the units need small, high-resolution displays; (2) graphics hardware must be capable of real-time, low-latency image generation; (3) a low-latency, high-accuracy system for head tracking in unprepared, possibly-noisy environments is necessary. We are addressing the third of these issues. There is as yet no tracking system that is lightweight and works in unprepared environments and in large spaces. As far as we know, no one has demonstrated a working head-tracking system for a room-sized environment (about 15' x 15'). The ceiling tracker described here is a start: the environment is large and expandable and the equipment, although heavy, is bearable. We envision an eventual system in which methods similar to those described here are used to calibrate the system's view of its environment. The algorithms may differ, but the principle — having the system model its sources of error and calibrate itself against them — will remain.

We wish to make one more point: the two examples presented in this paper give details of a general principle, and this general principle is applicable to cases other than the ones we describe. In short, as one designs a tracker (or other electro-mechanical assembly), one has the opportunity to leave some physical parameters fixed but unknown, and to then determine their exact values after construction. Doing this kind of post-construction calibration does, however, require that some aspects of the system be overdetermined. In the head-tracker example below, we could not have performed autocalibration if the tracker computed its position from just three LED beacons, since there would be no "error measure" as we computed the position–the equations would be exactly determined rather than overdetermined. Similarly, without multiple samples in the hand-tracking application, we could not determine the orientation matrix. So

*This work was supported in part by grants from NSF, DARPA, IBM, NCR, Sun Microsystems, DEC, and HP.



ACM-0-89791-601-8/93/008/0065 $01.50

the principle is this: if one wishes to use autocalibration, the system must have a surplus of information and a way to measure whether this information is internally consistent. The cost of obtaining this surplus of information is a design tradeoff, and should be considered during the design rather than after.

## 2 Operation of the Ceiling Tracker

Most current head trackers achieve a large working volume at the expense of accuracy and precision. However, some virtual environments applications require a large working volume *and* some minimum tracking precision.

A team at UNC–CH has developed an optoelectronic tracking system capable of tracking head motion with precision of approximately 0.2 degrees orientation and 1 mm translation. (A description of this system and its design can be found in the references [WAB+92] [WAB+90].). System accuracy has not been measured precisely, but has been found to be very adequate for the purposes of head-mounted display (HMD) applications. At present, the working volume is a 10' by 12' area, but the tracking area can in principle be expanded arbitrarily by adding LED-studded ceiling panels.

The method used by the optoelectronic tracker is conceptually similar to celestial navigation. A mariner observes the angles between some number of stars and the horizon, and then, knowing the stars' locations in the heavens, determines the vessel's position. Similarly, we observe a number of ceiling-mounted infrared LEDs, and knowing their positions, we compute the location (and orientation) of the head-tracking unit.

To be more precise, we have a helmet with cameras mounted on it. Some of the ceiling LEDs are rapidly flashed in a known sequence, and each one is possibly sighted by a camera. The choice of subset and sequence is not preset, but is determined "on the fly" as it is learned which LEDs are visible to which cameras. The cameras are lateral-effect photodiodes with lenses, and each can report the centroid of a spot of light that strikes its surface. The centroid's location is reported in image plane coordinates, $x$ and $y$. We call these *photocoordinates* (see Figure 1).

The placements of the cameras on the helmet are known, as are the locations of the principal points of the lens systems and the placements of the photodiodes' image planes within the camera casings. Thus, when the camera reports the photocoordinates of LED image on its image plane, we can compute the line, in head space, along which the LED must lie. We call this line a *back-projection*, because it is the result of projecting the ray from the photodiode back through the lens system and outward.

Now, given several back-projections in head space, and given the true locations of the LEDs in world space, where must the head be in world space so as to cause the back-projections to pass through their respective LEDs? With three (sufficiently general) back-projections, an unique solution can be found. With more than three, we have an overdetermined system and we compute a best fit according to a least-squares criterion, using a method called "space-resection by collinearity" (abbreviated "CA" for "collinearity algorithm"). We briefly describe CA in Section 3.1; full details can be found elsewhere [AW91]. Several questions about this tracker design that are often raised are discussed in an Appendix.

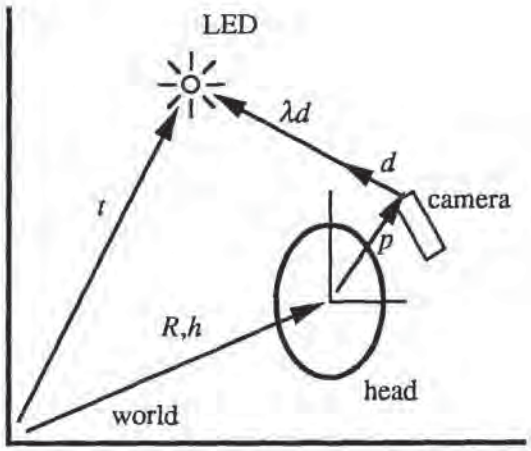



Figure 1: The geometry of the head-tracking system.

## 3 Explaining the Problem

The current design uses an adjustable superstructure to support the ceiling panels. The adjustments are needed because any conceivable support structure would bend under the loading of the panels, giving an undesirable curvature to the ceiling's surface.

With the current design of 10' by 12' (30 2' by 2' panels), the leveling process requires about 90 minutes of operator time, with specialized equipment.

We plan to build another, larger ceiling without this superstructure. The panels will be of the same size, but will drop directly into the standard ceiling grid, replacing the acoustic tiles found in many buildings. This ceiling will be 18' by 30'; the expense of a comparable-size superstructure is prohibitive, and leveling time would be several hours.

Standard ceiling grids are by no means flat, and we have therefore developed the autocalibration technique described here to determine the location of the LEDs after the panels are installed. Before describing that technique, however, we give more details of the collinearity algorithm.

### 3.1 The collinearity algorithm

The collinearity algorithm (CA) works by observing many (typically 10 to 20) LEDs and then computing a best estimate of headmount position and orientation. When an LED shines onto a photodiode, the photodiode reports the centroid of the LED's image on its face. Since the algorithm knows the headmount geometry, it is able to compute, in head space, where the back-projection emerges and in what direction it is pointed. Somewhere along this back-projection lies the LED (see Figure 2).

Thus

$$R(p + \lambda d) + h = t, \qquad \lambda > 0 \tag{1}$$

where $t$ the is location of the LED in world space, $R$ is the matrix that takes vectors in head coordinates to world coordinates (i.e., $R$ defines the *orientation* of the head-mount), $h$ is the world-space coordinates of the origin of the head-mount coordinate system; and $p$ and $d$ are the basepoint and direction (unit vector) of the back-projection ray in head-coordinates; $\lambda$ is the distance from the camera to the LED.

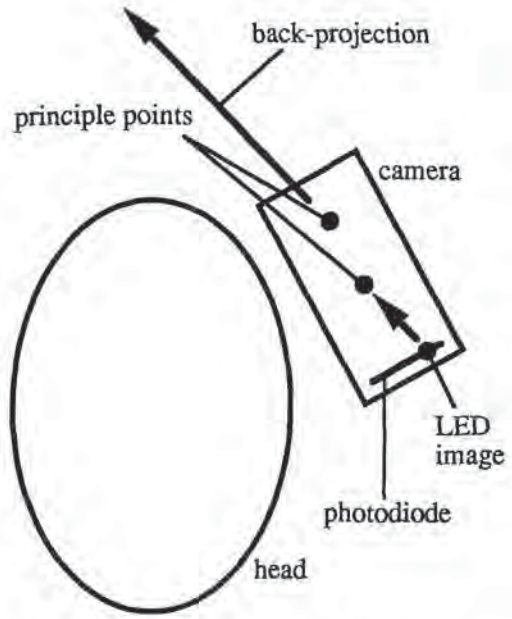



Figure 2: The back-projection ray from the camera towards the ceiling.

Equation 1 actually consists of three scalar equations, one for each of the $x$-, $y$-, and $z$-components. We can solve the $z$ component for $\lambda$ and substitute this into the $x$ and $y$ components. This eliminates $\lambda$ and leaves us with two scalar equations in the unknowns $R$ and $h$.

Many LEDs are seen at the same moment. Each of these generates two scalar equations. So each observation, which sights 12 to 20 LEDs, constructs a system of 24 to 40 equations in the unknowns $R$ and $h$. CA seeks those values of $R$ and $h$ that minimize the residuals of these equations in the least-squares sense. These values are found by applying a multidimensional Newton's method.

The method is most successful when given an initial guess very close to the optimal solution. In practice, this is easy to supply. The optical tracker typically provides updates every 12 to 20 milliseconds, and a person does not move far in that interval. Thus, for the initial guess, the algorithm merely uses the value of the previous update, which is guaranteed to be close.

## 4 Autocalibration: Rationale and Description

We have pointed out that it is very desirable to be able to construct the ceiling with loose tolerances, and be able to determine the locations of the LEDs afterward. CA does not depend upon any particular configuration of LED beacons – all places are alike to it. It does, however, require an exact knowledge of the locations of the LEDs, wherever they may be.

An "engineering" approach to achieving agreement between the physical geometry of the beacons and their software representation is prohibitively expensive. Therefore, we sought a way to determine the locations of the LEDs using existing hardware and some numerical processing.

The collinearity algorithm was derived from photgrammetric methods. Our LED calibration method, which makes use of CA as one step, was based primarily on influences from mathematics and computer graphics rather than the photgrammetry literature. We have since learned, however, that our approach has parallels in that literature, although we have found no exact analog. Nonethless, we strongly recommend that others working on optical tracking systems consult the photogrammetry literature [Sla80] for many ideas which, with slight modifications, may prove valuable in tracking.

We begin with an estimate of the beacon locations. We then take several thousand headmount observations (collecting 25,000 observations takes about 45 minutes) from a variety of positions, and use CA to "fit" the position of each observation to its beacon data. Of course, we know only approximately where the LEDs are, but fitting the headmount position to the beacon data allows some of the error in the beacon location estimates to cancel. The CA solution for the location of the headmount at each of these thousands of observations is likely to be rather bad: the sum of squares value will be large. To return to the marine analogy, it is as though the several circles of positions on the earth, each determined by a single star, failed to intersect at a single point, and instead intersected pairwise at several different points that surrounded a large region. The mariner estimates the vessel's position as somewhere at the center of the region, and begins to doubt the accuracy of the almanac's star locations.

After this initial set of observations, we derive the back-projections from each of these computed headmount locations, to yield "sightings" of the LEDs from roughly known positions. An LED sighted from several positions should be located at the intersection of the back-projections extending from those positions, but in general, the back-projections do not come together at a point, but tend instead to cluster in a particular region. We therefore adjust our estimate of each LED to be closer to this back-projection cluster. (The mariner, after several sets of inconsistent observations, decides to correct the almanac). This is the second step of our autocalibration.

After we adjust all the LEDs, the old observation positions are no longer optimal solutions in CA. So, we apply CA again to the observation positions, using the same data as before, but with the new beacon location estimates. (The mariner re-computes the vessel's position on each of the previous days, and now has circles of position that come closer to intersecting at single points). Thus we repeat the first step. We now continue, alternating between the two steps in this fashion, adjusting first one set of parameters and then the other, until we have settled to some configuration.

It seems surprising at first that this process converges at all; it is even more surprising to see how fast and how accurately it converges. We tested this by perturbing three of the ceiling panels as shown in Figure 4, and then running the algorithm. The average error-vector magnitudes for the first five full iterations were 13.1 mm, 4.7 mm, 3.2 mm, 2.5 mm, 2.2 mm, and 1.9 mm. After 20 iterations, which takes about two hours for 25,000 observations, the average error vector is down to 1.1 mm. Figure 5 is a computer-generated picture of the tracker ceiling. The beacons on the tilted panels are clearly visible.

The adjustment made to an LED's location depends on its relationship to the back-projections associated with it. A back-projection, in general, passes nearby the LED's es-

timated location. The vector drawn from the LED's estimated position to the back-projection's closest approach to that position is the *error vector* for that back-projection. A given LED has many back-projections, for each of which there is an associated error vector. We average these error vectors, and use this average as the adjustment to the LED's estimated position.

In a sense, each observation of an LED "votes" in the adjustment. An observation typically sees many LEDs, and cannot find a position from which to spear all its LEDs with its back-projections. The smallest adjustment possible for each LED that would completely satisfy an observation's collinearity conditions, would be an adjustment along the error vector. However, such an adjustment might conflict with the adjustment required by another observation.

The averaging is thus done as a compromise among the needs of the various observations that sight a given LED. It *is* possible to determine a new position for the LED that actually minimizes the sum of the squared lengths of the error vectors, but it is computationally expensive, and the averaging method works well and fast in practice.

### 4.1 Concerns About Noise: the Method in Practice

The autocalibration method was originally tried with simulated data so that it could be evaluated in the absence of noise and other complicating factors. It was found to be quite effective, providing rapid convergence. Performance on real data was not nearly as good – for reasons we now discuss.

First, the photodiode readings are noisy. The photocoordinates have as much as 12 microns of uncertainty. If the LED is a meter away from the camera, which has a 50-mm lens, the back-projection will miss by more than .25 mm even if headmount's position and orientation are exactly correct.

Second, the system of equations produced by each observation assumes that the LEDs are sighted simultaneously, and this is not true in practice. The LEDs are sampled in sequence, and each sample may take as much as a millisecond. If the user's head is turning at the (reasonable) rate of 180 degrees per second, the LED is 1 m. away from the axis of rotation, and 20 LEDS are sampled for the observation, then in the 20 milliseconds of sampling, the back-projection to the first LED may have traveled 6 cm. This causes the system of equations given by the observation data to be inconsistent, so that it cannot be satisfied by any position and orientation. The fact that the equations cannot be satisfied implies that the back-projections are simply wrong, and hence will "pull on" the LEDs wherever the observation settles.

Third, acquiring the right spatial distribution of observations is surprisingly difficult. The LEDs in the corner of the ceiling are typically seen in many fewer observations than the ones in the center. And when the LEDs in the corner are seen, it tends to be from one direction. Naturally, an LED in the corner can be seen only from one octant: below ceiling height and beneath the ceiling. But diversity in the angles from which the LED is seen is helpful. If an LED is seen from within a narrow cone of positions, then the location of the back-projection cluster is more sensitive to the errors mentioned earlier: a slight distortion in the back-projections' placement tends to disperse the cluster, denying the LED a strong centering influence.

Three observations can be made about the first source of error. First, in addition to using superior photodiodes and electronics, the error can be reduced by using lenses of longer focal length. With longer focal lengths, the 12-micron error in the LED image location would translate to an even narrower error cone for the corresponding back-projection. The primary disadvantage of the resulting small fields of view is that they can slip between the LEDs and fail to see any at all. Second, one can allow the headmount to sit still, accumulating photocoordinates, and average them over time to distill a more accurate reading. Unfortunately, with thousands of observations required, data acquisition for calibration would be very time-consuming. Third and most important, however, sensor noise error is insignificant in comparison to the other two sources of error.

The second source of error comes from the motion of the headmount. Again, for calibration purposes, we could take data points only when the headmount is still. But this again would make data acquisition intolerably slow. In practice, we have found that moving the headmount slowly helps substantially in reducing this error. A better solution is to change the system of equations to take into account the headmount velocity, both linear and rotational. This would require a minimum of six LEDs per observation to obtain a fully-determined system, but typical counts are already 12 to 20 LEDs per observation. This is future work.

The third problem is being addressed by an graphics application that assists in data acquisition. A top view (map) of the ceiling is displayed on a nearby workstation, on which LEDs presently observed are marked. (This is needed because the LEDs emit infrared light, invisible to the naked eye.) The least-sampled LEDs are marked in a different color, allowing the operator to direct his efforts to sighting those LEDs. During the calibration process, in addition, certain LEDs are identified as having unusually large error vectors, meaning that their associated back-projections do not cluster tightly enough. A second run of data collection can be made, and special attention paid to these trouble spots.

In addition to the precautions and program assistance mentioned above, the calibration algorithm tests for high error vectors and culls out observations for which CA cannot find a satisfactory solution. (This is similar to computing robust statistics by eliminating outliers.) In this way, the algorithm is made somewhat more tolerant of operator mistakes or wild readings from the sensors (which are very rare).

Two features of the automated calibration method have not yet mentioned. First, the ceiling tracker is in frequent use. We can simply collect the observations *during use* and use these in an off-line calibration computation, so that we can keep the tracking system aligned without downtime. At present the system does not need frequent recalibration, and we do separate calibration runs, allowing us to collect only "good" data (i.e., data taken with slow head motion). Second, the entire algorithm is subject to a kind of systematic error: if we apply a rigid motion to our estimates of the beacon locations, CA converges exactly as well as before. This means that if one wishes to calibrate the system in absolute coordinates (relative to some frame of reference for the room in which the ceiling tracker sits), one may have to apply a rigid motion to the computed beacon positions so taht the estimated lcoations of a few key beacons are their actual positions as determined, for example, by measurements from the walls of the room.

## 5 Using a Headmounted Magnetic Tracker for Handtracking

Although the optical tracker gives satisfactory accuracy over a large working volume, its design does not lend itself to

hand tracking for several reasons: the bulkiness of the cameras, the geometry of the situation (the user's body may obscure the hand's "view" of the ceiling, and the hand may not be held upright), and the dynamic range requirements on photodiode sensitivity (because of changing distances from the ceiling). We have found, however, that magnetic trackers [RBSJ79] usually provide satisfactory performance within a small tracking volume, although in our environment they report significantly distorted position and orientation outside of a range of about five feet. Since one's hands never get farther than a few feet from one's head, we decided to place a magnetic source on the headmount and track hand motion from there.

Ultimately, however, we want to know the hand's location in the ceiling coordinate system. The optical tracker reports the head location in ceiling space, the magnetic source lies at some fixed location in head space, and the Polhemus tracking system reports the hand's location in source space. We compose the change-of-coordinate transformations among these three systems to get the hand's location in ceiling space.

Of course, the fixed location of the magnetic source within head space must be known before we can compose the transforms. As before, we have two choices: engineering, i.e., careful placement of the source on a precise rigid mount attached to the headframe, and autocalibration, in which we place the source approximately and then infer its position precisely using autocalibration. We chose the latter approach.

### 5.1 The calibration problem and solution

We attach the magnetic source to the headmount with a rigid Plexiglas framework whose position is known within a few inches, and whose orientation is easy to measure within about 10 degrees. These are clearly not adequate measurements: if the hand is held 3' from the source, a 1 degree error in the measurement of the source's orientation would cause a 15 mm error in the computation of the hand's placement.

Our calibration approach is simple. We take simultaneous optical tracker and magnetic tracker readings, and use them to recover the placement of the source within head space. The algorithm starts with a very approximate estimate of the source's placement, such as might be obtained by inspection.

We start by fixing the Polhemus sensor at some location in ceiling space. The exact location is not important – it need only stay still.

Now consider what *should* happen (if the system were calibrated properly) as the headmount moves about in the proximity of the sensor. We receive readings from the optical and magnetic trackers. The optical tracker produces the *Ceiling-from-Head* transform, and the magnetic tracker provides the *Source-from-Sensor*transform. If the *Head-from-Source* is correct, then the composition of these transforms, *Ceiling-from-Head* × *Head-from-Source* × *Source-from-Sensor*, should remain constant and should be the *Ceiling-from-Sensor* transform, which is constant because the sensor is not moving) (see Figure 3.

If, for observation $i$, $R_i$ is the reported *Ceiling-from-Head* transform, $T_i$ is the reported *Source-from-Sensor* transform, $S$ is the unknown but fixed *Head-from-Source*transform, and $M$ is the unknown *Ceiling-from-Sensor* transform, then for any pair of reports from the trackers,

$$R_i S T_i = M,$$

provided the trackers are accurate. But if $S$ is wrong, then as we walk around the room, the sensor's position and orientation (i.e., $M$), as computed by the transform composition,

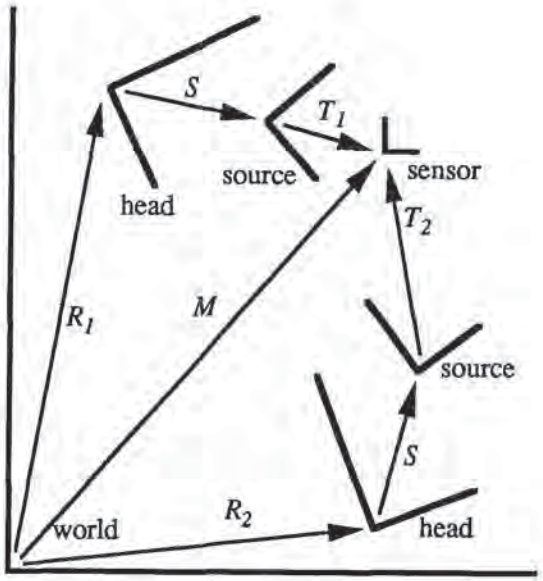



Figure 3: The geometry of the composite tracker during two different observations. The source is held fixed in the head-mounted-display coordinate system, and the sensor is fixed in the world coordinate system, but the relationship of the head to the world and of the sensor to the source change with each observation.

will drift, appearing to be in different places, depending on where we are standing.

After $n$ readings from $n$ different places, we have a system of $n$ equations,

$$R_i S T_i = M_i, \qquad i = 0 \ldots n-1,$$

where $S$ is our (incorrect) estimate of *Head-from-Source*, and each $M_i$ is computed as $R_i S T_i$. We seek the value of $S$ that will make the $M_i$s equal (i.e., the value of $S$ that keeps our reports of the sensor positions and orientation constant).

Our estimate of $S$ and the readings $R_i$ and $T_i$ give rise to many estimates of the sensor location $M_i$. We might get closer to the true value of $M$ by taking some compromise among the $M_i$s, say, by estimating that it is the average of the $M_i$s. We actually bias this average slightly by averaging the matrix entries, and then performing the Gram-Schmidt process on the rotational part of the matrix. This averaging and orthonormalization step is likely to prompt objections, which we address below. For now, we continue with our description of the algorithm.

Let's call this resulting average transform $Q$. If we imagine that this is the correct value for the sensor location, then we can write the system

$$R_i S T_i = Q, \qquad i = 0 \ldots n-1,$$

If $Q$ really were the correct location, then we could take any one of the equations $R_i S T_i = Q$ and solve for $S$ to recover that value, since the remaining transforms would be known. However, when we actually do this we find that we get different values for $S$. Why? Because $Q$ is not correct – but it might be close. Solving for $S$ gets us

$$S_i = R_i^{-1} Q T_i^{-1}, \qquad i = 0 \ldots n-1,$$

each of which suggests a different value for $S$. We average these in exactly the way we did the $M_i$s to arrive at a new estimate for $S$.

This completes one iteration of the algorithm. With our new estimate of $S$ we go back and acquire new $M_i$s, which we average to get $Q$, which we substitute back into the system so we can solve for the $S_i$'s, which we average to get our new $S$. We iterate until the value of $S$ stabilizes.

## 5.2 Justification for averaging matrices

Averaging makes sense for points in a linear space like a plane, but we are trying to use it in a nonlinear space (the set of $3 \times 3$ rotation matrices). But in general, the average of a set of points on a non-linear space like a sphere is almost always a point that is not on the sphere. Even so, if all the points are very close together on the sphere, this averaging yields a point that is near to a point on the sphere that one might call the "average.". The reason is that the local geometry of the sphere is well approximated by any of the tangent planes within the local region, and so the sphere-based averaging is a close approximation to the tangent-plane averaging. Since the average of the sphere points does not lie on the sphere, however,to get a meaningful average we must project back onto the sphere. The critical properties of the projection map here are (1) it is continuous in a neighborhood of the sphere, and (2) for points already *on* the sphere, the projection is the identity. We now explain why the process we used in averaging matrices is analogous.

The set $Q$ of $4 \times 4$ translation-and-rotation matrices is a subset of $R^{16}$; it is curved in much the same way that the sphere is a curved subset of $R^3$. We can average a collection of points on the object $Q$ (i.e., several matrices), in much the same way as we averaged points on the sphere. Before this can make sense, though, we must honor the restriction that the points being averaged should be close to one another. And the same caveat applies: the $R^{16}$-average of a set of points in $Q$ is not likely to lie in $Q$, and will need to be projected back to $Q$, which is what the Gram-Schmidt process does. Note, though, that the Gram-Schmidt process has the same properties as radial projection: it is a continuous function of the entries of the matrix (at least for matrices that are close to rotation matrices), and for a rotation matrix, the Gram-Schmidt process does nothing.

Still, there remains the question, "How small a region must the points be gathered in for averaging to make sense?" On the sphere, it certainly makes sense when all the points are contained in some hemisphere. For matrices, the averaging of the translational part is simply an average in a linear space, and needs no justification; for the rotational part, we *believe* (but have not proved formally) that the averaging process makes sense for any collection of (rotation) matrices $\{A_i\}$ for which all the inner products $z_{ij} = trace(A_i A_j^t)$ are greater than $1/2$. In practice, however, our matrices are all quite close to one another, and these inner products are large. Furthermore, the algorithm in practice is far more robust than we had expected. In a 2D simulation of the problem, for example, it takes some effort to give an initial estimate of the matrix $S$ that makes the algorithm diverge.

## 5.3 Noise in the data, and the algorithm in practice

The accuracy of this method depends on the accuracy of the trackers providing the data. The optical and magnetic trackers, providing the $R_i$'s and $T_i$'s are noisy. In general, no choice of $S$ and $M$ satisfies all the equations simultaneously. It is impossible to determine what the correct values are, and we can only hope to get an approximation to the correct $S$. Nonetheless, the error in the estimates of $S$ and $M$, since they are based on multiple samples, should average out the random noise from the trackers. The *systematic* noise (e.g., one tracker always reports a slightly scaled $x$-coordinate) is not averaged out, but is also inherent in the system; if such systematic noise were too large, the system would be unusable in practice. Our experience is that the values of $S$ and $M$ converge quite rapidly to values that provide quite good hand-tracking.

There is one important observation about this instance of autocalibration: the sensor readings from which the calibration is done must be in fairly general position. In some cases, for example if the orientation of the headmount remains constant throughout the sampling process and the headmount is translated only along a single axis, then a little linear algebra shows that the estimates of $S$ and $M$ can all be identical but nonetheless be incorrect. But if the headmount is tilted and translated about all three axes during data gathering, and if multiple tilts and translations about each axis are included, then the equations will be sufficiently general to guarantee convergence (given a good enough initial estimate of $S$).

## 5.4 Remarks on the Method

One nice aspect of this method is that no exact measurements are needed. The location of the sensor somewhere in lab space may remain unknown. The position of the source in head space need only be estimated – and that is the only measurement necessary: the rest of the information is taken directly from the tracker sensors themselves.

The calibration procedure takes about 20 minutes in all: 5 minutes to put the sensor in place and gather data, and about 15 minutes (including graphical display of progress at each step) to settle on a value for $S$.

The number of equations and the "tightness" of the cluster of estimates can give a feel for the accuracy of the estimate. In averaging the $S_i$s, we can compute a residual for each, that is the magnitude of the deviation from the average $S_i$ (deviation, here, being the difference in the translation components of the transforms). The angular deviation could be treated in precisely the same manner: the angle of rotation required to get from one transform's orientation to the other's. The root mean square of these residuals can be used as a reasonable metric for the "tightness" of the estimates of $S$. In a typical calibration run of 25 measurements, the RMS value of the deviations from the mean $S_i$ was about 4.6 millimeters.

These residuals are not the same as the error in the result, although they are related. The more equations we use, the more likely the resulting transform is to be close to the actual one. This is somewhat like averaging a random variable – the variance can be very high, but the longer we average, the closer we are likely to get to the expected value.

# 6 Conclusion

We have described two applications of a goal-based approach to alignment of mechanical systems in VE tracking. In both cases, the automated calibration simplifies the construction of the systems, and makes it easier to modify the systems without extensive redesign of hardware or software. Note that the autocalibration system is designed to calibrate against a particular source of error, LED position error in the first case and Polhemus source location error in the second. Other sources of error in the system will confound the autocalibration process, so that if they are persistent enough, the autocalibration model should be revised to incorporate

them as well. As the number of variables to be calibrated is increased, the number of observations must increase as well, of course, but in the head-tracking system, we have calibrated about 3000 variables successfully.

## 7 Acknowledgments

We would like to thank Al Barr for his initial involvement in the discussion of autocalibration, which helped to lead us away from the engineering approach and into the mathematical one. We also thank J.-F. Wang for having the idea of autocalibration for headtracking systems in the first place. Henry Fuchs' persistent demands for greater accuracy and bigger tracking spaces have provided a constant impetus. And we both owe a debt to our colleagues who have supported us in this project, particularly Ron Azuma and Russell Taylor.

## References

[AW91] Ronald Azuma and Mark Ward. Space Resection by Collinearity: Mathematics behind the Optical Ceiling Head-Tracker. Technical Report TR91-048, UNC-Chapel Hill Department of Computer Science, November 1991.

[BB88] Ronen Barzel and Alan H. Barr. A Modeling System Based on Dynamic Constraints. *Computer Graphics*, 22(4):179–188, August 1988.

[RBSJ79] F. H. Raab, E. B. Blood, T. O. Steiner, and H. R. Jones. Magnetic Position and Orientation Tracking System. *IEEE Transactions on Aerospace and Electronic Systems*, AES-15(5):709–718, September 1979.

[Sla80] C.C. Slama, editor. *Manual of Photogrammetry*. American Society of Photogrammetry, Falls Church, Va, fourth edition, 1980.

[WAB+90] J. F. Wang, R. Azuma, G. Bishop, V. Chi, J. Eyles, and H. Fuchs. Tracking a Head-Mounted Display in a Room-sized Environment with Head-Mounted Cameras. *Proc SPIE 1990 Technical Symposium on Optical Engineering and Photonics in Aerospace Sensing*, 1290, 1990.

[WAB+92] M. Ward, R. Azuma, R. Bennet, S. Gottschalk, and H. Fuchs. A Demonstrated Optical Tracker with Scalable Work Area for Head-Mounted Display Systems. In *Proceedings of 1992 Symposium on Interactive 3D Graphics, Cambridge, Mass.*, pages 43–52, March 1992.

[Wan90] Jih-Fang Wang. A Real-time Optical 6D Tracker for Head-mounted Display Systems. Technical Report TR90-011, UNC-Chapel Hill Department of Computer Science, March 1990.

## 8 Appendix: Head Tracker Design

Several issues concerning our current tracker design are often raised by those unfamiliar with it. First, why use exotic, expensive lateral-effect photodiodes instead of the highly developed, inexpensive CCD technologies?

The reason is timing. We want updates from the tracking system every 12 to 20 milliseconds. With CCDs, both the bandwidth required for data transfer and the image processing necessary per frame were prohibitive. We found it more feasible to digitize the voltages coming from the lateral-effect photodiodes (the only thing of interest after all in the image that a CCD camera would have seen) and transmit this comparatively low bandwidth signal.

Second, why use multiple cameras with narrow fields of view? Why not use a single camera with a wide-angle lens?

The problem here is the limited precision of the photocoordinates. We have observed that, in practice, the photocoordinates reported by the camera may be off by as much as 12 microns. A narrow field of view helps reduce this problem, but since CA requires disparate angles to operate effectively (otherwise the matrices involved tend to become ill-conditioned), this field-of-view requirement compels us to use multiple cameras.

Lastly, why put cameras on the head and LEDs on the ceiling, rather than vice versa, since the headmount would be much lighter with LEDs rather than cameras?

To explain our strategy, we call the cameras on the walls the "outside-looking-in" approach, and the cameras on the headmount the "inside-looking-out" approach. "Inside-looking-out" has three advantages over its counterpart: sensitivity to orientation, economical scalability, and energetics considerations.

*Sensitivity to orientation* is the ability to detect a head rotation. In the current system, a .5 degree turn of the head, for instance, causes a very significant change in the LEDs' coordinates on the photodiodes, regardless of their distances from the camera. By contrast, in the outside-looking-in approach this change in orientation would be almost imperceptible.

An *economically scalable system* is one in which the cost of increasing the working volume is low in terms of cost per unit tracking space. Because of the narrow field-of-view requirement on the cameras, the outside-looking-in approach (on a 30 $ft^2$ area) would need many cameras mounted on the walls. Covering the ceiling with LEDs is less expensive.

*Energetics* refers to how light energy is received from an LED. Quadrupling the distance between LED and camera, for instance, decreases the light energy received by a factor of 16. Furthermore, LEDs do not emit light uniformly in every direction: most of their power is emitted in the direction they face, and drops off with the angle away from their axis (depending on the packaging). If the cameras are wall-mounted, and the LEDs are head-mounted, then, as the user walks about, many LEDs may be oblique to the cameras, and the distances between user and cameras may vary a great deal. These two effects combine to make the range of signal strengths received by the cameras too wide.

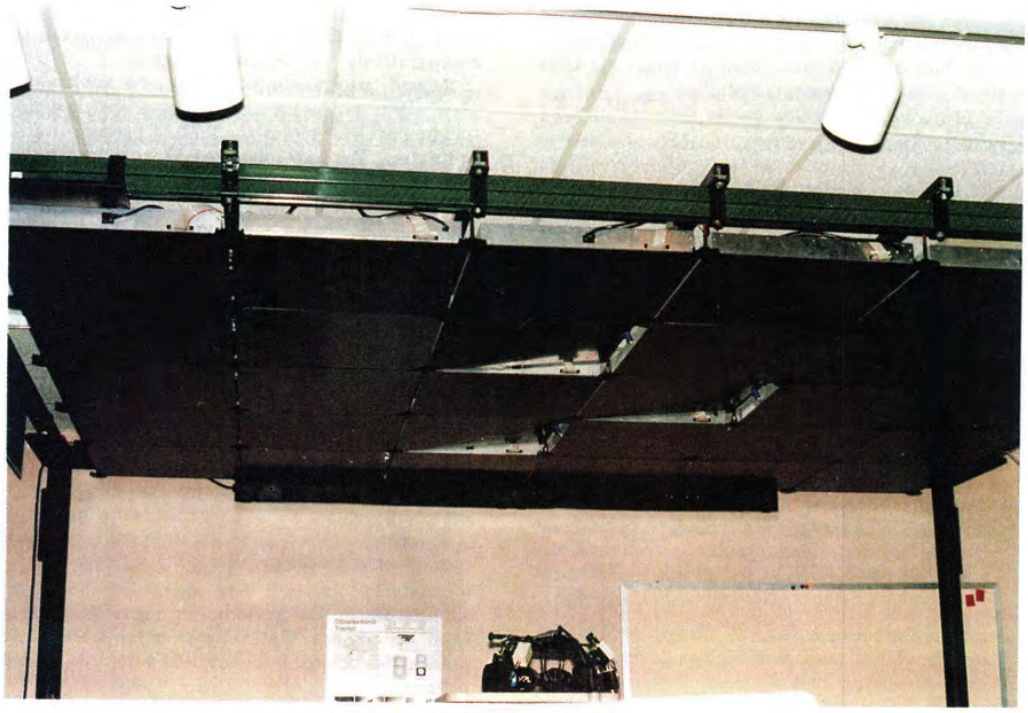


Figure 4: Tilting three panels in the ceiling to test autocalibration.

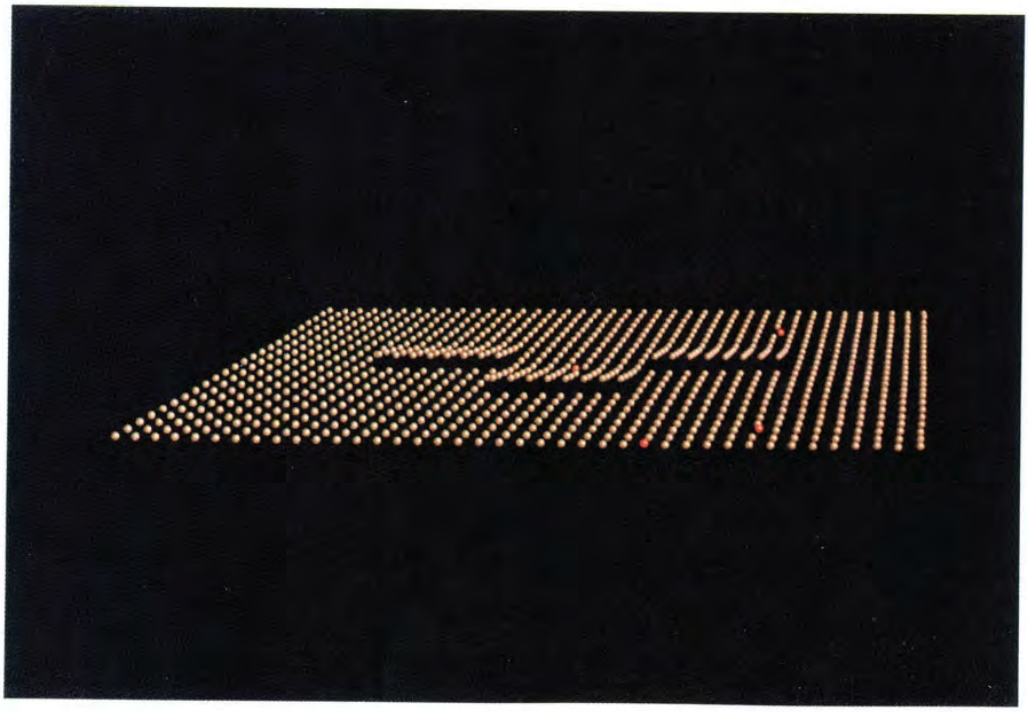


Figure 5: Computer display of calibrated beacon locations. The beacons shown in red were insufficiently sampled and could not be calibrated by the algorithm (see Section 4.1).

# Toolglass and Magic Lenses: The See-Through Interface

Eric A. Bier, Maureen C. Stone, Ken Pier, William Buxton†, Tony D. DeRose‡
Xerox PARC, 3333 Coyote Hill Road, Palo Alto, CA 94304
†University of Toronto, ‡University of Washington

## Abstract

*Toolglass™ widgets* are new user interface tools that can appear, as though on a transparent sheet of glass, between an application and a traditional cursor. They can be positioned with one hand while the other positions the cursor. The widgets provide a rich and concise vocabulary for operating on application objects. These widgets may incorporate visual filters, called *Magic Lens™ filters*, that modify the presentation of application objects to reveal hidden information, to enhance data of interest, or to suppress distracting information. Together, these tools form a *see-through interface* that offers many advantages over traditional controls. They provide a new style of interaction that better exploits the user's everyday skills. They can reduce steps, cursor motion, and errors. Many widgets can be provided in a user interface, by designers and by users, without requiring dedicated screen space. In addition, lenses provide rich context-dependent feedback and the ability to view details and context simultaneously. Our widgets and lenses can be combined to form operation and viewing macros, and can be used over multiple applications.

**CR Categories and Subject Descriptors:** I.3.6 [**Computer Graphics**]: Methodology and Techniques–interaction techniques; H.5.2 [**Information Interfaces and Presentation**]: User Interfaces–interaction styles; I.3.3 [**Computer Graphics**]: Picture/Image Generation–viewing algorithms; I.3.4 [**Computer Graphics**]: Graphics Utilities–graphics editors

**Key Words:** multi-hand, button, lens, viewing filter, control panel, menu, transparent, macro

## 1. Introduction

We introduce a new style of graphical user interface, called the *see-through interface*. The see-through interface includes semi-transparent interactive tools, called *Toolglass™ widgets*, that are used in an application work area. They appear on a virtual sheet of transparent glass, called a *Toolglass sheet*, between the application and a traditional cursor. These widgets may provide a customized view of the application underneath them, using viewing filters called *Magic Lens™* filters. Each lens is a screen region together with an operator, such as "magnification" or "render in wireframe," performed on objects viewed in the region. The user positions a Toolglass sheet over desired objects and then points through the widgets and lenses. These tools create *spatial modes* that can replace temporal modes in user interface systems.

Two hands can be used to operate the see-through interface. The user can position the sheet with the non-dominant hand, using a device such as a trackball or touchpad, at the same time as the dominant hand positions a cursor (e.g., with a mouse or stylus). Thus, the user can line up a widget, a cursor, and an application object in a single two-handed gesture.



A set of simple widgets called *click-through buttons* is shown in figure 1. These buttons can be used to change the color of objects below them. The user positions the widget in the vicinity and indicates precisely which object to color by clicking through the button with the cursor over that object, as shown in figure 1(b). The buttons in figure 1(c) change the outline colors of objects. In addition, these buttons include a filter that shows only outlines, suppressing filled areas. This filter both reminds the user that these buttons do not affect filled areas and allows the user to change the color of outlines that were obscured.

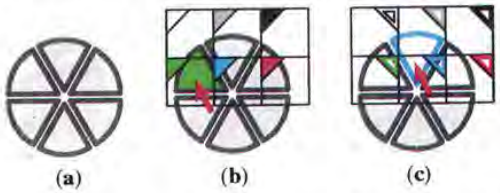



**Figure 1. Click-through buttons. (a) Six wedge objects. (b) Clicking through a green fill-color button. (c) Clicking through a cyan outline-color button.**

Many widgets can be placed on a single sheet, as shown in figure 2. The user can switch from one command or viewing mode to another simply by repositioning the sheet.

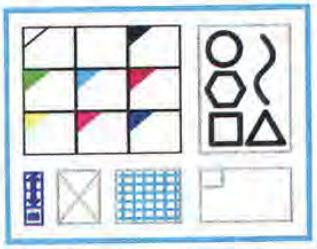


**Figure 2. A sheet of widgets. Clockwise from upper left: color palette, shape palette, clipboard, grid, delete button, and buttons that navigate to additional widgets.**

Widgets and lenses can be composed by overlapping them, allowing a large number of specialized tools to be created from a small basic set. Figure 3 shows an outline color palette over a magnifying lens, which makes it easy to point to individual edges.

**Figure 3. An outline color palette over a magnifying lens.**

The see-through interface has been implemented in the Multi-Device Multi-User Multi-Editor (MMM) framework[5] in the Cedar

 ACM-0-89791-601-8/93/008/0073 $01.50

programming language and environment,[24] running on the SunOS UNIX™-compatible operating system on Sun Microsystems SPARCstations and other computers. The Gargoyle graphics editor,[20] as integrated into MMM, serves as a complex application on which to test our interface. We use a standard mouse for the dominant hand and a MicroSpeed FastTRAP™ trackball for the non-dominant hand. The trackball includes three buttons and a thumbwheel, which can be used to supply additional parameters to the interface.

The remainder of this paper is organized as follows. The next section describes related work. Section 3 describes some examples of the tools we have developed. Section 4 discusses general techniques for using the see-through interface. Section 5 discusses some advantages of this approach. Section 6 describes our implementation. Sections 7 and 8 present our conclusions and plans for future work.

Except for figures 12 and 16, all of the figures in this paper reflect current capabilities of our software.

## 2. Related Work

The components of the see-through interface combine work in four areas: simultaneous use of two hands, movable tools, transparent tools, and viewing filters. In this section, we describe related work in these four areas.

### Multi-Handed Interfaces

Several authors have studied interfaces that interpret continuous gestures of both hands. In Krueger's VIDEOPLACEs system,[15] the position and motion of both of a participant's hands, as seen by a video camera, determine the behavior of a variety of on-screen objects, including animated creatures and B-spline curves. Buxton and Myers discovered that users naturally overlap the use of both hands, when this is possible, and that, even when the two hands are used sequentially, there is still a performance advantage over single-hand use.[7,8]

Other work characterizes the situations under which people successfully perform two-handed tasks. Guiard presents evidence that people are well-adapted to tasks where the non-dominant hand coarsely positions a context and the dominant hand performs detailed work in that context.[4] Similarly, Kabbash presents evidence that a user's non-dominant hand performs as well or better than the dominant hand on coarse positioning tasks.[13]

Our system takes full advantage of a user's two-handed skills; the non-dominant hand sets up a context by coarsely positioning the sheet, and the dominant hand acts in that context, pointing precisely at objects through the sheet.

### Movable Tools

Menus that pop up at the cursor position are movable tools in the work area. However, such a menu's position is determined by the cursor position before it appears, making it difficult to position it relative to application objects.

Several existing systems provide menus that can be positioned in the same work area as application objects. For example, MacDraw "tear-off menus" allow a pull-down menu to be positioned in the work area and repositioned by clicking and dragging its header.[17] Unfortunately, moving these menus takes the cursor hand away from its task, and they must be moved whenever the user needs to see or manipulate objects under them.

Toolglass sheets can be positioned relative to application objects and moved without tying up the cursor.

### Transparent Tools

Some existing systems that allow menus to be positioned over the work area make these menus transparent. For example, the Alto Markup system[18] displays a menu of modes when a mouse button goes down. Each menu item is drawn as an icon, with the space between icons transparent. Bartlett's transparent controls for interactive graphics use stipple patterns to get the effect of transparency in X Windows.[2]

While these systems allow the user to continue to see the underlying application while a menu is in place, they don't allow the user to interact with the application through the menu and they don't use filters to modify the view of the application, as does our interface.

### Viewing Filters

Many existing window systems provide a pixel magnifier. Our Magic Lens filters generalize the lens metaphor to many representations other than pixels and to many operations other than magnification. Because they can access application-specific data structures, our lenses are able to perform qualitatively different viewing operations, including showing hidden information and showing information in a completely different format. Even when the operation is magnification, our lenses can produce results of superior quality, since they are not limited to processing data at screen resolution.

The concept of using a filter to change the way information is visualized in a complex system has been introduced before.[25,10,14] Recent image processing systems support compostition of overlapping filters.[23] However, none of these systems combine the filtered views with the metaphor of a movable viewing lens.

Other systems provide special-purpose lenses that provide more detailed views of state in complex diagrams. For example, a fisheye lens can enhance the presentation of complicated graphs.[21] The bifocal display[22] provides similar functionallity for viewing a large space of documents. The MasPar Profiler[3] uses a tool based on the magnifying lens metaphor to generate more detail (including numerical data) from a graphical display of a program.

Magic Lens filters combine viewing filters with interaction and composition in a much broader way than do previous systems. They are useful both as a component of the see-through interface and as a general-purpose visualization paradigm, in which the lenses become an integral part of the model being viewed.

## 3. Examples

This section shows several tools that demonstrate features of the see-through interface. Because we have implemented primarily in the graphical editing domain, most of these tools are tailored to that application. However, the see-through interface can be used in a wide variety of other application domains.

### Shape and Property Palettes

Palettes are collections of objects or properties that can be added to a scene. Figure 1 showed two widgets that apply color to shapes. Similar tools can be designed to apply other graphical properties, such as type and line styles to an illustration, shading parameters to a 3D model, or initial values to a simulation. Figure 4 illustrates a widget containing graphical shapes that can be "pushed through" from the tool into the illustration below. In figure 4(a), the user has positioned a shape palette widget (shown in cyan) over an illustration (shown in magenta). When the user clicks on a shape on the tool, a copy of that shape is added to the illustration. The widget attaches the copied shape to the cursor for interactive dragging until the final shape position is achieved (figure 4(b)).


**Figure 4. Shape palette. (a) Choosing a shape. (b) Placing the shape.**

Figure 5 shows a design for a property palette for setting the face of text in a document. Each face (regular, bold, etc.) has an active region on the right side of the tool. Selecting the text displayed in this region changes its face.

| | |
|---|---|
| regular | temporal modes and modes created by holding down a keyboard key with |
| *italic* | *spatial modes*. Because these spatial |
| **bold** | modes can be changed **directly** in the |
| ***bold italic*** | application work area, the cursor and the user's attention can remain on the |

**Figure 5. Font face palette. The word "directly" is being selected and changed to bold face.**

## Clipboards

*Clipboard* widgets pick up shapes and properties from underlying objects, acting as visible instantiations of the copy and paste keys common in many applications. Clipboards can pick up entire objects or specific properties such as color, dash pattern or font. They can hold single or multiple copies of an object. The objects or properties captured on the clipboard can be copied from the clipboard by clicking on them, as in the palette tools.

Figure 6 shows a symmetry clipboard that picks up the shape that the user clicks on (figure 6(a)) and produces all of the rotations of that shape by multiples of 90 degrees (figure 6(b)). Moving the clipboard and clicking on it again, the user drops a translated copy of the resulting symmetrical shape (figure 6(c)). Clicking the small square in the upper left corner of the widget clears the widget so that new shapes can be clipped.

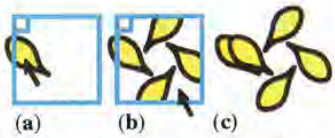



**Figure 6. Symmetry clipboard. (a) Picking up an object. (b) Rotated copies appear. (c) The copies are moved and pasted.**

Figure 7 shows an example of a type of clipboard that we call a *rubbing*. It picks up the fill color of an object when the user clicks on that object through the widget (figure 7(a)). The widget also picks up the shape of the object as a reminder of where the color came from (figure 7(b)). Many fill-color rubbings can be placed on a single sheet, allowing the user to store several colors and remember where they came from. The stored color is applied to new shapes when the user clicks on the applicator nib of the rubbing (figure 7(c)).

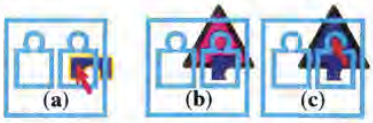



**Figure 7. Fill-color rubbings. (a) Lifting a color. (b) Moving the clipboard. (c) Applying the color.**

Besides implementing graphical cut and paste, clipboards provide a general mechanism for building customized libraries of shapes and properties.

## Previewing Lenses

In graphical editing, a lens can be used to modify the visual properties of any graphical object, to provide a preview of what changing the property would look like. Properties include color, line thickness, dash patterns, typeface, arrowheads and drop shadows. A previewing lens can also be used to see what an illustration would look like under different circumstances; for example, showing a color illustration as it would be rendered on a black/white display or on a particular printer. Figure 8 shows a Celtic knotwork viewed through two lenses, one that adds drop shadows and one that shows the picture in black and white. The achromatic lens reveals that the drop shadows may be difficult to distinguish from the figure on a black/white display.

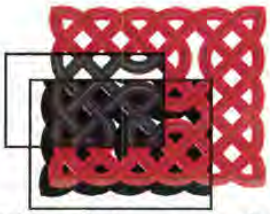


**Figure 8. An achromatic lens over a drop shadow lens over a knotwork. (Knotwork by Andrew Glassner)**

Previewing lenses can be parameterized. For example, the drop shadow lens has parameters to control the color and displacement of the shadow. These parameters can be included as graphical controls on the sheet near the lens, attached to input devices such as the thumbwheel, or set using other widgets.

## Selection Tools

Selection is difficult in graphical editing when objects overlap or share a common edge. Our selection widgets address this problem by modifying the view and the interpretation of input actions. For example, figure 9 shows a widget that makes it easy to select a shape vertex even when it is obscured by other shapes. This tool contains a wire-frame lens that reveals all vertices by making shape interiors transparent. Mouse events are modified to snap to the nearest vertex.

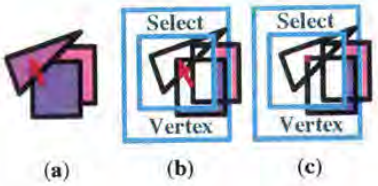



**Figure 9. Vertex selection widget. (a) Shapes. (b) The widget is placed. (c) A selected vertex.**

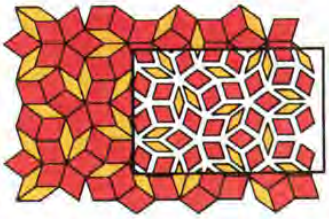


**Figure 10. The local scaling lens. (Tiling by Doug Wyatt)**

Figure 10 shows a lens that shrinks each object around its own centroid. This lens makes it easy to select an edge that is coincident with one or more other edges.

### Grids

Figure 11 shows three widgets, each of which displays a different kind of grid. The leftmost two grids are rectangular with different spacings. The rightmost grid is hexagonal. Although each grid only appears when the lens is in place, the coordinates of the grid are bound to the scene, so that grid points do not move when the sheet moves. By clicking on the grid points and moving the widget, the user can draw precise shapes larger than the widget. If the sheet is moved by the non-dominant hand, the user can quickly switch between the grids during an editing motion.

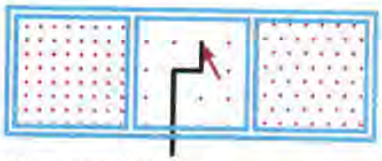


**Figure 11. Three grid tools.**

### Visualization

Figure 12 illustrates the use of tools and lenses to measure Gaussian curvature in the context of a shaded rendering of a 3D model. The pseudo-color view indicates the sign and relative magnitude of the curvature,[9] and the evaluation tool displays the value at the point indicated.

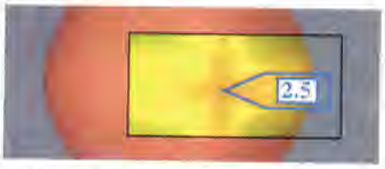



**Figure 12. Gaussian curvature pseudo-color lens with overlaid tool to read the numeric value of the curvature. (Original images courtesy of Steve Mann)**

## 4. Using the See-Through Interface

Widgets and lenses are most effective when supported by appropriate conventions specifying how to position, size, organize, and customize them. This section discusses a few of these issues.

### Moving and Sizing the Sheet or the Application

A Toolglass sheet can be moved by clicking and dragging on its border with a mouse or by rolling the trackball. The sheet and all its widgets can stretch and shrink as a unit when the user works a a second controller such as a thumbwheel. With these moving and sizing controls, the user can center a widget on any application object and size the widget to cover any screen region. Large widgets can be used to minimize sheet motion when applying a widget to several objects. A widget that has been stretched to cover the entire work area effectively creates a command mode over the entire application.

By clicking a button on the trackball, the user can disconnect the trackball from the sheet and enable its use for scrolling and zooming a selected application area. If a sheet is over this application, the user can now move an application object to a widget instead of moving a widget to an object. This is a convenient way to use the see-through interface on illustrations that are too large to fit on the screen.

### Managing Sheets

A typical application will have a large number of widgets in its interface. To avoid clutter, we need a way to organize these widgets and sheets. One approach is to put all of the widgets on a single sheet that can be navigated by scrolling and zooming. Perlin and Fox's paper in these proceedings[19] describes techniques for creating and navigating unlimited structures on a single sheet. A second approach is to have a master sheet that generates other sheets. Each of these sheets could generate more sheets, like hierarchical menus. A third technique, used in our prototype, is to allow a single sheet to show different sets of widgets at different times. The set to display can be selected in several ways: the user can click a special widget in the set, like the arrows in HyperCard,™[11] that jumps to another set. In addition, a master view provides a table of contents of the available sets allowing the user to jump to any one. To use different sets simultaneously, the user creates additional sheets.

### Customizing Sheets

Because sheets can contain an unlimited number of widgets, they provide a valuable new substrate on which users can create their own customized widgets and widget sets. In effect, the sheets can provide a user interface *editor*, allowing users to move and copy existing widgets, compose macros by overlapping widgets, and snap widgets together in new configurations. Indeed, with the techniques described in this paper, one Toolglass sheet could even be used to edit another.

## 5. Advantages of See-Through Tools

In this section, we describe some advantages we see for using the see-through interface. Most of these advantages result from placing tools on overlapping layers and from the graphical nature of the interface.

In most applications, a control panel competes for screen space with the work area of the application. Toolglass sheets exist on a layer above the work area. With proper management of the sheets, they can provide an unlimited space for tools. The widgets in use can take up the entire work area. Then, they can be scrolled entirely off the screen to provide an unobstructed view of the application or space for a different set of widgets.

The see-through user interface can be used on tiny displays, such as notebook computers or personal digital assistants, that have little screen real estate for fixed-position control panels. It can also be used on wall-sized displays, where a fixed control panel might be physically out of reach from some screen positions. These tools can move with the user to stay close at hand.

A user interface layer over the desktop provides a natural place to locate application-independent tools, such as a clipboard that can copy material from one window to another.

These widgets can combine multiple task steps into a single step. For example, the vertex selection widget of figure 9 allows the user to turn on a viewing mode (wire-frame), turn on a command mode (selection), and point to an object in a single two-handed gesture.

Most user interfaces have temporal modes that can cause the same action to have different effects at different times. With our interface, modes are defined spatially by placing a widget and the cursor over the object to be operated on. Thus, the user can easily see what the current mode is (e.g., by the label on the widget) and how to get out of it (e.g., move the cursor out of the widget). In addition, each widget can provide customized feedback for its operation. For example, a widget that edits text in an illustration can include a lens that filters out all the objects except text. When several widgets are visible at once, the feedback in each one

serves a dual role. It helps the user make proper use of the widget and it helps the user choose the correct widget.

The visual nature of the see-through interface also allows users to construct personalized collections of widgets as described above.

## 6. Implementation

This section provides an overview of our implementation of the see-through interface.

### Toolglass Sheets

We describe three Toolglass subsystems: one that handles simultaneous input from two pointing devices and updates the screen after multiple simultaneous changes, one that modifies pointing events as they pass through widgets, and one that modifies graphical output as it passes up through each widget.

#### *Multi-Device Input and Screen Refresh*

Our Toolglass software uses the MMM framework.[5] The see-through interface relies on the following features of MMM.

MMM takes events from multiple input devices, such as the mouse and trackball, keeps track of which device produced which event, and places all events on a single queue. It dequeues each event in order and determines to which application that event should be delivered. MMM applications are arranged in a hierarchy that indicates how they are nested on the screen. Each event is passed to the root application, which may pass the event on to one of its child applications, which may in turn pass the event on down the tree. Mouse events are generally delivered to the most deeply nested application whose screen region contains the mouse coordinates. However, when the user is dragging or rubberbanding an object in a particular application, all mouse coordinates go to that application until the dragging or rubberbanding is completed. Keyboard events go to the currently selected application.

To support Toolglass sheets, MMM's rules for handling trackball input were modified. When a sheet is movable, trackball and thumbwheel events go to the top-level application, which interprets them as commands to move or resize the sheet, respectively. When the sheet is not movable, the trackball and thumbwheel events are delivered to the selected application, which interprets them as commands to scroll or zoom that application.

#### *Filtering Input Through Lenses and Widgets*

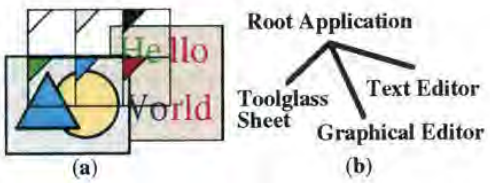



**Figure 13. A simple hierarchy of applications**

Ordinarily, MMM input events move strictly from the root application towards the leaf applications. However, to support the see-through interface, input events must be passed back up this tree. For example, figure 13(b) shows an application hierarchy. The left-to-right order at the lower level of this tree indicates the top-to-bottom order of applications on the screen. Input events are first delivered to the Toolglass sheet to determine if the user is interacting with a widget or lens. If so, the event is modified by the sheet. In any case, the event is returned to the root application, which either accepts the event itself or passes it on to the child applications that appear farther to the right in the tree.

The data structure that represents an MMM event is modified in three ways to support Toolglass sheets. First, an event is annotated with a representation of the parts of the application tree it has already visited. In figure 13, this prevents the root application from delivering the event to the sheet more than once. Second, an event is tagged with a command string to be interpreted when it reaches its final application. For example, a color palette click-through button annotates each mouse-click event with the command name "FillColor" followed by a color. Finally, if the widget contains a lens, the mouse coordinates of an event may be modified so the event will be correctly directed to the object that appears under the cursor through that lens.


**Figure 14. Composing color-changing widgets.**

Widgets can be composed by overlapping them. When a stack of overlapped widgets receives input (e.g., a mouse click), the input event is passed top-to-bottom through the widgets. Each widget in turn modifies the command string that has been assembled so far. For example, a widget might concatenate an additional command onto the current command string. In figure 14, a widget that changes fill colors (figure 14(a)) is composed with a widget that changes line colors (figure 14(b)) to form a widget that changes both fill and line colors (figure 14(c)). If the line color widget is on top, then the command string would be "LineColor blue" after passing through this widget, and "LineColor blue; FillColor cyan" after both widgets.

#### *Filtering Output Through Lenses and Widgets*

Ordinarily, MMM output is composed from the leaf applications up. To support lenses, the normal screen refresh composition has been extended to allow information to flow down and across the tree as well as up. For example, if the widgets in figure 13 contain one or more lenses, and if any of those lenses is situated over the graphical editor, each lens must examine the contents of the graphical editor (which is the lens's sibling in the hierarchy) in order to draw itself.

In addition, to improve performance, MMM applications compute the rectangular bounding box of the regions that have recently changed, and propagate this box to the root application, which determines which screen pixels will need to be updated. Generally, this bounding box is passed up the tree, transformed along the way by the coordinate transformation between each application and the next one up the tree. However, lenses can modify the set of pixels that an operation affects. A magnifying lens, for example, generally increases the number of pixels affected. As a result, the bounding box must be passed to all lenses that affect it to determine the final bounding box.

### Magic Lens Filters

A Magic Lens filter modifies the image displayed on a region of the screen, called the *viewing region*, by applying a *viewing filter* to objects in a model. The *input region* for the lens is defined by the viewing region and the viewing filter. It may be the same size as the viewing region, or different, as in the magnification lens. For a 3D model, the input region is a cone-shaped volume defined by the eye point and the viewing region. Input regions can be used to cull away all model objects except those needed to produce the lens image. Our current implementations do not perform this culling; as described below, there are advantages to lenses that operate on the entire model.

When several lenses are composed, the effect is as though the

model were passed sequentially through the stack of lenses from bottom to top, with each lens operating on the model in turn. In addition, when one lens has other lenses below it, it may modify how the boundaries of these other lenses are mapped onto the screen within its own boundary. The input region of a group of lenses taken as a whole can be computed by applying the inverses of the viewing filters to the lens boundaries themselves.

Our lenses depend on the implementation of Toolglass sheets to manage the size, shape and motion of their viewing regions. This section describes two strategies we have tried for implementing viewing filters: a procedural method that we call *recursive ambush*, and a declarative method that we call *model-in model-out*. We also describe a third method that promises to be convenient when applicable, called *reparameterize-and-clip*. Finally, we discuss issues that arise in the presence of multiple model types.

### Recursive Ambush

In the recursive ambush method, the original model is described procedurally as a set of calls in a graphics language such as Interpress™[12] or PostScript.®[1] The lens is a new interpreter for the graphics language, with a different implementation for each graphics primitive. In most cases, the implementation of a given graphics primitive first performs some actions that carry out the modifying effect of the lens and then calls the previous implementation of the primitive. For example, a lens that modifies a picture such that all of its lines are drawn in red would modify the "DrawLine" primitive to set the color to red and then call the original "DrawLine" primitive.

When lenses are composed, the previous implementation may not be the original graphics language primitive, but another lens primitive that performs yet another modification, making composition recursive.

Recursive ambush lenses appear to have important advantages. Because they work at the graphics language level, they work across many applications. Because they work procedurally, they need not allocate storage. However, the other methods can also work at the graphics language level. In addition, recursive ambush lenses have three major disadvantages. First, making a new lens usually requires modifying many graphics language primitives. Second, debugging several composed lenses is difficult because the effects of several cooperating interpreters are hard to understand. Finally, performance deteriorates rapidly as lenses are composed because the result of each lens is computed many times; the number of computations doubles with the addition of each lens that overlaps all of the others.

### Model-In Model-Out

In the model-in model-out (MIMO) method, we make a copy of the original model as the first step. This model might be the data structure of an editor, a representation of graphics language calls, an array of pixels or some other picture representation. The implementation walks through this data structure and modifies it in accordance with the desired behavior of the lens. When composed with other lenses, a MIMO lens takes each model that is produced by each lens under it, produces a modified version of that model, and associates it with the clipping region formed by intersecting its clipping region with that of the lens underneath. The resulting models are passed on to lenses above.

Although MIMO lenses must allocate storage, this investment pays off in several ways. First, during the rendering of a single image, each lens computes its output models only once, and then saves them for use by any lenses that are over it. In addition, if the computed model is based on the entire original model, then redrawing the picture after a lens moves is just a matter of changing clipping regions; no new model filtering is needed. In this case, each lens maintains a table of the models it has produced. The table is indexed by the models it has received as input and when they were last modified. The action of such a lens often consists of a single table lookup.

MIMO lenses have many other advantages. Given routines to copy and visit parts of the model, the incremental effort to write a MIMO lens is small. Many of our lenses for graphical editor data structures were written in under 20 minutes and consist of under 20 lines of code. Debugging composed lenses is easy because the intermediate steps can easily be viewed. Finally, MIMO lenses can perform a large class of filtering functions because they can access the input model in any order. In particular, they can compute their output using graphical search and replace,[16] as shown in figure 15 where each line segment is replaced by multiple line segments to create a "snowflake" pattern.

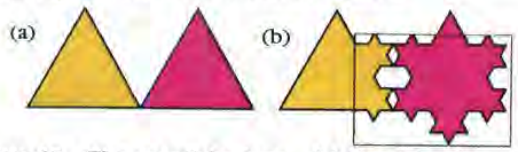



**Figure 15. The snowflake lens. (a) Two triangles. (b) Snowflake lens over part of the scene.**

An important variation of MIMO is to allow the output model to differ in type from the input model. For example, a lens might take a graphics language as input and produce pixels as output. In this case, the lens walks the original model, rather than copying it, and allocates data structures of the new model type.

### Reparameterize and Clip

If the original image is being produced on the screen by a renderer with variable parameters, it is easy to implement lenses that show the effects of varying those parameters. To function, the lens modifies a renderer parameter and asks the renderer to redraw the model clipped to the boundary shape of the lens. For example, a lens showing the wireframe version of a 3D shaded model can be implemented this way.

Several reparameterize-and-clip lenses can be composed if the parameter changes made by these lenses are compatible. In the region of overlap, the renderer re-renders the original model after each of the overlapping lenses has made its changes to the renderer parameters. The flow of control and performance of a stack of these lenses is like that of MIMO lenses; a new output is computed for each input region received from lenses underneath. These lenses differ from MIMO in that each output is computed from the original model, and each output is always a rendering.

### Multiple Model Types

In our discussion above, lenses are used to view a single type of model, such as a graphical editor data structure or a graphical language. In practice, multiple model types are often present, for two reasons. First, a lens can overlap multiple applications at the same time, where the applications have different model types, as shown above in figure 13. Second, a lens may overlap both an application and a lens, where the lens output and application model are of different types. For example, in figure 16, the wireframe lens converts from a 3D model to a 2D line drawing. The magnifier lens, which operates on 2D drawings, overlaps both the original image and the output of the wireframe lens. Rich illustrations can be produced by permitting lenses to overlap multiple model types in this way.

Supporting multiple model types requires *type conversion* and

*type tolerance*. When a lens that expects one type of model as input is moved over a model of a different type, the system may automatically convert the model to be of the type required; this is type conversion. For example, all of our applications produce Interpress graphics language calls as part of drawing themselves on the screen. When a lens that takes Interpress as input is positioned over one of these applications, that application converts its model to Interpress on demand for that lens.

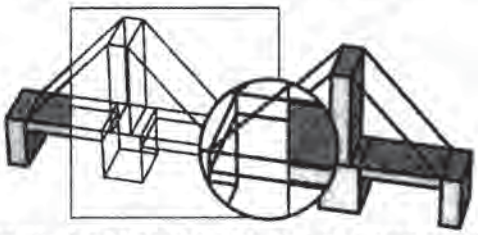


**Figure 16. A bridge made of shaded, 3D blocks showing a 3D wireframe lens and a 2D magnifier.**

Alternatively, when presented with a model it does not understand, a lens can simply pass that model through unchanged; this is type tolerance. For example, a lens that operates only on a graphics editor's data structures will only modify the image in the part of that lens's boundary that overlaps the graphics editor; other regions are unchanged.

### Composing Widgets and Lenses

When a widget and a lens are composed, their functions combine. For example, consider a click-through button on top of a magnifying lens. Mouse events pass through the button, are annotated with a command, and then pass through the lens, which applies the inverse of its transformation to the mouse coordinates. During screen refresh, the widget adds its appearance to the output of the lens. If the lens is on top of the widget, input events are first transformed by the lens and then tested to see if they fall within the button or not; during refresh, the widget adds its appearance to the model, which is then acted on by the lens. A widget and lens can be very tightly coupled. For example, an editing tool could include a lens that displayed control points or editing handles implemented as widgets.

### Performance

Our sheets and lenses are already fast enough to be useful on current hardware, but need to be faster for smooth motion. For example, using our prototype on a SPARCstation 10, we measured the time it takes to redraw the screen after moving a wireframe lens of size 70 by 70 pixels over the Penrose tiling of figure 10, containing 117 filled and outlined shapes. For the MIMO implementation of the lens, once it has cached its output scene, it takes an average of 300 milliseconds to repaint the scene, of which 120 milliseconds are spent drawing the lens interior. The same lens implemented using recursive ambush takes %15 longer to redraw the lens interior, which we attribute to the procedure call overhead of the recursive approach. Computing the filtered scene for the MIMO lens takes an average of 480 milliseconds for this example. This computation is performed whenever the illustration under the lens is changed or lens parameters are modified.


**Figure 17. The Magic Lenses logo.**

## 7. Conclusions

We have described a new style of user interface, the see-through interface, based on Toolglass widgets and Magic Lens filters. The see-through interface offers a new design space for user interfaces based on spatial rather than temporal modes and provides a natural medium for two-handed interaction. Because the interface is movable and overlays the application area, it takes no permanent screen space and can be conveniently adapted to a wide range of display sizes. Because the overlaid tools are selected and brought to the work area simply by moving the Toolglass sheet, the user's attention can remain focused on the work area. Because the operations and views are spatially defined, the user can work without changing the global context.

The see-through interface provides a new paradigm to support open software architecture. Because Toolglass sheets can be moved from one application to another, rather than being tied to a single application window, they provide an interface to the common functionality of several applications and may encourage more applications to provide common functionality. Similarly, Magic Lens filters that take standard graphics languages as input work over many applications.

In addition to their role in user interfaces, Magic Lens filters provide a new medium for computer graphics artists and a new tool for scientific visualization. When integrated into drawing tools, these filters will enable a new set of effects and will speed the production of traditional effects. Figure 17 shows a magnifying lens and a wireframe lens used to produce our Magic Lenses logo.

Integrated into scientific visualization tools, these filters can enhance understanding by providing filtered views of local regions of the data while leaving the rest of the view unchanged to provide context, as was shown in the visualization example in figure 12.

We hope the see-through interface will prove to be valuable in a wide variety of applications. While the examples in this paper stress applications in graphical editing, these tools can potentially be used in any screen-based application, including spreadsheets, text editors, multi-media editors, paint programs, solid modelers, circuit editors, scientific visualizers, or meeting support tools. Consider that most applications have some hidden state, such as the equations in a spreadsheet, the grouping of objects in a graphical editor, or the position of water pipes in an architectural model. A collection of widgets and lenses can be provided to view and edit this hidden state in a way that takes up no permanent screen space and requires no memorization of commands.

We believe that the see-through interface will increase productivity by reducing task steps and learning time, providing good graphical feedback, and allowing users to construct their own control panels and spatial modes.

## 8. Plans for Future Work

The see-through interface is a framework that can be used to create many new tools in many application domains. Exploring the current space of possibilities will take many people many years. Furthermore, this design space will be enlarged by future software and hardware. We will carry out some of this exploration ourselves, creating new widgets in different application domains, working out taxonomies for the tools we discover, designing new conventions for composing, editing, navigating, organizing and triggering these tools, combining them with existing user interface techniques, and testing them on users performing real work.

We are building two Toolglass widget toolkits. The first is a

traditional toolkit in which widgets are created through object-oriented programming. The second toolkit is based on our EmbeddedButtons project;[6] here, users draw new widgets and collections of widgets using a graphical editor and then apply behavior to these graphical forms, where the behavior is expressed in a user customization language.

We are designing new algorithms to increase the speed of these tools. It is clear that Magic Lens filters and, to a lesser extent, Toolglass widgets provide a new way to consume the graphics power of modern computers.

Finally, we are working to better understand how to model and implement general composition of widgets and lenses, especially those that work with multiple model and applications types.

## Acknowledgments

We thank Blair MacIntyre for implementing our first lenses for 2D graphics and Ken Fishkin for his demonstration of lenses for text editing. We thank many of our colleagues at PARC for fruitful discussions and enthusiasm, including Stu Card, Ken Fishkin, Andrew Glassner, David Goldberg, Christian Jacobi, Jock Mackinlay, David Marimont, George Robertson, Marvin Theimer, Annie Zaenen, and Polle Zellweger, plus our consultants Randy Pausch and John Tukey. Finally, we thank Xerox Corporation for supporting this work.



## References

1. Adobe Systems Incorporated. *PostScript® Language Reference Manual, second edition.* Addison-Wesley, 1990.
2. Bartlett, Joel F. Transparent Controls for Interactive Graphics. WRL Technical Note TN-30, Digital Equipment Corp., Palo Alto, CA. July 1992.
3. Beck, Kent, Becher, Jon, and Zaide, Liu. Integrating Profiling into Debugging. *Proceedings of the 1991 International Conference on Parallel Processing, Vol. II, Software*, August 1991, pp. II-284-II-285.
4. Guiard, Yves. Asymmetric Division of Labor in Human Skilled Bimanual Action: The Kinematic Chain as a Model. *The Journal of Motor Behavior, 19*, 4, (1987), pp. 486-517.
5. Bier, Eric A. and Freeman, Steve. MMM: A User Interface Architecture for Shared Editors on a Single Screen. *Proceedings of the ACM SIGGRAPH Symposium on User Interface Software and Technology* (Hilton Head, SC, November 11-13), ACM, New York, (1991), pp. 79-86.
6. Bier, Eric A., EmbeddedButtons: Supporting Buttons in Documents. *ACM Transactions on Information Systems, 10*, 4, (1992), pp. 381-407.
7. Buxton, William and Myers, Brad A.. A Study in Two-Handed Input. *Proceedings of CHI '86* (Boston, MA, April 13-17), ACM, New York, (1986), pp. 321-326.
8. Buxton, William. There's More to Interaction Than Meets the Eye: Some Issues in Manual Input. *Readings in Human-Computer Interaction: A Multidisciplinary Approach.* (Ronald M. Baecker, William A.S. Buxton, editors). Morgan Kaufmann Publishers, Inc., San Mateo, CA. 1987.
9. Dill, John. An Application of Color Graphics to the Display of Surface Curvature. Proceedings of SIGGRAPH '81 (Dallas, Texas, August 3-7). *Computer Graphics, 15*, 3, (1981), pp. 153-161.
10. Goldberg, Adele and Robson, Dave, A Metaphor for User Interface Design, *Proceedings of the University of Hawaii Twelfth Annual Symposium on System Sciences*, Honolulu, January 4-6, (1979), pp.148-157.
11. Goodman, Danny. *The Complete HyperCard Handbook.* Bantam Books, 1987.
12. Harrington, Steven J. and Buckley, Robert R.. *Interpress, The Source Book.* Simon & Schuster, Inc. New York, NY. 1988.
13. Kabbash, Paul, MacKenzie, I. Scott, and Buxton, William. Human Performance Using Computer Input Devices in the Preferred and Non-preferred Hands. *Proceedings of InterCHI '93*, (Amsterdam, April 24-29), pp. 474-481.
14. Krasner, Glenn and Hope, Stephen, A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80, *Journal of Object-Oriented Programming, 1*, 3, (1988), pp. 26-49.
15. Krueger, Myron W., Gionfriddo, Thomas, and Hinrichsen, Katrin. VIDEOPLACE – An Artificial Reality. *Proceedings of CHI '85* (San Francisco, April 14-18). ACM, New York, (1985), pp. 35-40.
16. Kurlander, David and Bier, Eric A.. Graphical Search and Replace. Proceedings of SIGGRAPH '88 (Atlanta, Georgia, August 1-5) *Computer Graphics, 22*, 4, (1988), pp. 113-120.
17. *MacDraw Manual.* Apple Computer Inc. Cupertino, CA 95014, 1984.
18. Newman, William. *Markup User's Manual.* Alto User's Handbook, Xerox PARC technical report, (1979), pp. 85-96.
19. Perlin, Ken and Fox, David. Pad: An Alternative Approach to the Computer Interface. this proceedings.
20. Pier, Ken, Bier, Eric, and Stone, Maureen. An Introduction to Gargoyle: An Interactive Illustration Tool. *Proceedings of the Intl. Conf. on Electronic Publishing, Document Manipulation and Typography* (Nice, France, April). Cambridge Univ. Press, (1988), pp. 223-238.
21. Sarkar, Manojit and Brown, Marc H.. Graphical Fisheye Views of Graphs. *Proceedings of CHI '92*, (Monterey, CA, May 3-5, 1992) ACM, New York, (1992), pp. 83-91.
22. Spence, Robert and Apperley, Mark. Data Base Navigation: An Office Environment of the Professional. *Behaviour and Invormation Technology, 1*, 1, (1982), 43-54.
23. *ImageVision*, Silicon Graphics Inc., Mountain View, CA.
24. Swinehart, Daniel C., Zellweger, Polle T., Beach, Richard J., Hagmann, Robert B.. A Structural View of the Cedar Programming Environment. *ACM Transactions on Programming Languages and Systems, 8*, 4, (1986), pp. 419-490.
25. Weyer, Stephen A. and Borning, Alan H., A Prototype Electronic Encyclopedia, *ACM Transactions on Office Systems, 3*, 1, (1985), pp. 63-88.



# An Interactive 3D Toolkit for Constructing 3D Widgets

Robert C. Zeleznik, Kenneth P. Herndon, Daniel C. Robbins,
Nate Huang, Tom Meyer, Noah Parker and John F. Hughes

Brown University
Department of Computer Science
Providence, RI 02912
(401) 863-7693; {bcz,kph,dcr,nth,twm,nfp,jfh}@cs.brown.edu

## CR Categories

I.3.6 [Computer Graphics]: Methodology and Techniques; Interaction Techniques D.1.7 [Programming Languages]: Programming Techniques; Visual Programming D.2.2 [Software Engineering]: Tools and Techniques; User Interfaces

## 1 Introduction

Today's user interfaces for most 3D graphics applications still depend heavily on 2D GUIs and keyboard input. There have been several recent attempts both to extend these user interfaces into 3D and to describe intermediary 3D widgets[1] that control application objects [3; 4; 5; 7; 13; 15]. Even though this style of interaction is a straightforward extension of interaction through intermediary 2D widgets such as dials or sliders, we know of no efforts to develop interactive 3D toolkits akin to UIMX or Garnet [11].

The Brown Graphics Group has had considerable experience using its Unified Graphics Architecture (UGA) system [16] to script 3D widgets such as deformation racks [14], interactive shadows [9], parameterized models, and other constrained 3D geometries. Using this experience, we have developed an interactive toolkit to facilitate the visual programming of the geometry and behavior of such interactive models. The toolkit provides both a core set of 3D widget primitives for constructing interactive behaviors based on constrained affine transformations, and an interactive 3D interface for combining these primitives into more complex widgets.

This video paper describes the fundamental concepts of the toolkit and its core set of primitives. In particular, we describe (i) the conceptual structure of the primitives, (ii) the criteria used to select a particular primitive widget set that would be expressive enough to let us construct a wide range of interactive 3D objects, and (iii) the constraint relationships among the primitives.

## 2 Overview of our 3D Toolkit

The traditional approach to designing user interface toolkits is to create a library of software objects and customize them through instantiation and specialization within standard programming languages [12; 15]. Although this approach is extremely powerful, exploring the full potential requires that programmers be able to visualize complex relationships among software objects (*e.g.*, constraint networks, data- and control-flow graphs). A second paradigm, based on graphically manipulating function networks [1; 8; 10], is more accessible to the non-programmer, but still suffers because inherently geometric relationships must be specified by wiring 2D boxes together.

Our toolkit uses direct manipulation of 3D widgets to model the construction of widgets and application objects whose geometric components are affinely constrained. This paradigm is more natural than scripting or dataflow programming because the process of constructing such objects is inherently geometric, and also enables non-programmers and designers to construct these objects visually. The scope of these constructions includes, for example, all of the widgets we have built in the last few years and standard joints such as slider, pin, and ball joints.

We introduce the notion of primitive 3D widgets that can be combined with other primitive 3D widgets, using a process called *linking*, to establish one or more constraint relations between them. In some cases, the resulting composite objects are still considered widgets; in others, they are thought of as the behavioral scaffolding to which the geometry of application objects can be attached. The fact that the interface and application objects exist in the same underlying system, UGA, allows us to blur the distinction between them. We feel that such blurring is natural for 3D applications in general, and especially for virtual reality applications.

Linking is related to snapping [3], but differs in requiring explicit interactive selection of source and destination objects, followed by explicit user confirmation. This protocol reduces clutter by eliminating alignment objects. In the interest of simplifying the user interface, all linking operations are unparameterized, although in future work, parameterized linking for more advanced users and more complicated widgets will be explored.

## 3 Conceptual Structure of Widget Primitives

A primitive widget combines the geometries and behaviors of its ports and other more simple primitives. A *port* is an encapsulation of one or more constraint values and a geometric representation. It can be loosely considered a data type with the additional requirement that its visual appearance suggest the meaning of the data. Ports are related to one another within a single widget via a network of bi-directional constraints. In addition, specific interaction techniques are associated with each port. Each interaction technique tells how to modify a port while maintaining constraints on other ports. For example, if a user manipulates a point that is constrained to be on a line, the constraint could be resolved by moving the line with the point, by restricting the user's interaction so that the point never leaves the line, or by a combination of the two. We must choose one of these as we implement the toolkit. These interaction techniques can be thought of as hints to a constraint solver when the constraint network is underdetermined so it can provide real-time, precise interactions.



[1] That is, encapsulations of geometry and behavior.

 ACM-0-89791-601-8/93/008/0081 $01.50

In addition to having an internal constraint network, a primitive widget can be related to another primitive widget by linking a port of the former to a port of the latter. This establishes a constraint (bi- or uni-directional) between the two ports. Ports are already constrained by the internal constraint network of a primitive, and the new constraints must be consistent with the existing constraints. Therefore, associated with each port is a function that determines how to attach new constraints to that port and how to modify its interaction techniques so as to facilitate constraint maintenance.

## 4 Description of the Toolkit Primitives

Having selected this framework to build our toolkit, we designed a general set of primitives to allow the interactive construction of not only the various 3D widgets previously scripted, but also application objects such as parameterized geometric models. These primitives are intended to be general enough to allow exploration of a wide set of object designs without having to resort to hand-coding.

We chose a "coordinate system" metaphor as a basis for our primitives. Each primitive visually represents a 0D, 1D, 2D, or 3D coordinate system and each can be constrained by affine transformations to the coordinate systems of other primitives. This metaphor can be used to express a wide variety of user interactions, including those of our previous 3D widgets [5; 14; 9]. However, the coordinate-system metaphor is only a framework for conceptualizing the primitives, not a strict definition of them. That is, the primitives were designed with regard to the sometimes antagonistic desires both to represent the coordinate system metaphor faithfully and to provide the semantics most useful for geometric and behavioral constructions.

The toolkit has primitives that correspond to position, orientation, measure (linear and angular), 2D and 3D Cartesian coordinate systems, a general extension mechanism for importing an arbitrary relationship, and the full set of UGA's geometric models.

The two most basic primitives, *Point* and *Ray*, encapsulate position and orientation respectively. Points and Rays represent 0D coordinate system entities; *i.e.*, there is only one element[2] of a Point or a Ray and therefore 0 coordinates are required to specify it. (Contrast this with a line, which has an infinite number of elements, each specified by one coordinate.) The Point primitive, represented by a small sphere, is an abstraction of a single 3D point. The Ray primitive, represented by an arrow, corresponds to a based vector, although we often treat it as just a vector (its position being a display convenience). Both primitives can be freely translated in space, but only the Ray can be rotated.

The notion of distance (linear measure) is represented through a 1D coordinate system primitive, the *Length*, represented by two Points, a port for the 1D coordinate system (represented by a thin cylinder connecting the Points), and a port for the Length's measure (represented by a small marker at the middle of the thin cylinder). While the Length appears as a bounded line segment, it actually encapsulates the notion of an infinite 1D coordinate system whose origin is at the line's start point (indicated by a small disc) and whose unit length is equal to the distance between the two points measured in the world coordinate system. We reuse the Point for the endpoints to help define the user interaction with the Length. Each of the Length's endpoints can be directly translated while the other remains fixed. Translating the cylinder joining the two Points translates both endpoints by the same amount. An alternate formulation of the Length would have both endpoints move whenever either was translated. Choosing either formulation is difficult in the absence of an application, so we chose the technique that seemed most useful.

Angular measure is represented by a two-handed clock-like primitive, the *Angle*. Each hand of the clock represents a vector and the outer ring of the clock represents the angle between the two vectors.

[2] In the sense of sets.

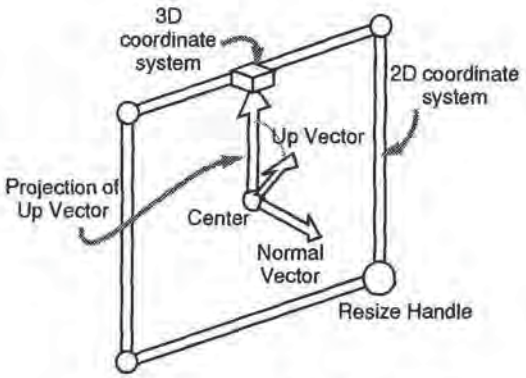



Figure 1: The ports of the Plane primitive.

The most complex primitive, the *Plane*, represents both a 2D Cartesian plane and a 3D Cartesian space. We opted to combine both concepts into a single primitive because users frequently use the two concepts in conjunction with one another and because the sets of ports are nearly identical, with a space being a superset of a plane. Visualizing an oriented plane requires ports for the plane's normal, center, and up-vector (similar to the PHIGS *VUP*), and for the size of a unit vector in each of the plane's axes. In addition, a port is required for the concept of the plane itself (as opposed to parameters that define the plane). A rectangle in the plane represents this port; its size determines the magnitude of each unit vector in the Plane's coordinate system. We also include a useful port for the projection of the plane's up-vector onto the plane, although this is not a required part of a Cartesian plane. To handle a 3D Cartesian space, the only additional port required is something to represent the concept of the space itself. The Plane reuses Points and Rays and introduces new geometry to represent the concept of the plane (a rectangle) and the space (a cube at the top of the up vector).

In order that the toolkit be extensible enough to handle new problem domains, there are also *Black-box* primitives, each representing a relationship with some number of ports that lacks a natural geometric representation. Ports on black-boxes are geometrically represented as labeled buttons. The accompanying video shows two Black-boxes: an interface to Barr's nonlinear deformation functions [2] and a PHIGS camera specification [6].

Finally, all the geometric objects in 3D modeling environments (cubes, spheres, CSGs, etc.) are considered collectively as a single primitive class called *Geometries*. In terms of the data it represents, each of the Geometries is essentially equivalent to a Cartesian space, although it is not annotated with additional geometry (as is the Plane primitive). In our system, each geometric object has an internal boundary representation relative to a local object coordinate system. This local coordinate system is used as a default coordinate system associated with a Geometry primitive to make it functionally equivalent to a Cartesian space. Since Geometries are not annotated with the ports of a Plane primitive, linking operations must infer from the context of the link operation which port of the implicit Cartesian space is intended. Linking operations usually apply to the origin of the Geometry's local coordinate system, though they can apply to the local coordinate system's normal and up-vector. When the default linking operation chooses the wrong port, the user can override the choice by making the object's local coordinate system explicit and choosing ports directly.

## 5 Linking the Toolkit Primitives

We now describe what occurs in the toolkit when a port of one primitive is linked to a port of another. Again, our choices for

the semantics of inter-primitive linking are guided by the desire to stay close to the coordinate-system metaphor and the desire to have reasonable behaviors when there is no obvious answer in the underlying metaphor.

A linking operation generally asserts one of two types of relations: it either establishes a bi-directional equality relationship between two similarly typed ports or projects one port into the coordinate system of the other port, using their common 3D embedding as the medium of projection.

Consider linking a Point to another Point: here, the first Point is set to be positionally equivalent to the second Point. However, linking a Ray to a Ray is slightly different in that the orientation of the first Ray is made equivalent to that of the second Ray, but the positions of the two Rays remain distinct. Rotating either Ray causes the other to change, but translating either Ray has no effect on the other. This choice of how to link two Rays together is ambiguous, because a Ray actually represents two geometric values, a position and an orientation. Thus the action is chosen by considering the context of the linking operation. In linking a Ray to a Ray, the user typically wants them both to have the same orientation, so only the orientation values are linked. If a user wishes to equate the positions of the Rays, then the position port of the Ray must be made explicit by linking each Ray to a common Point.

A different form of linking occurs when a lower-dimensional primitive is linked to a higher-dimensional one. Such a link causes the lower-dimensional primitive to be geometrically projected onto the implied *span*[3] of the higher-dimensional primitive. After this projection, the lower-dimensional primitive is associated with a coordinate in the higher-dimensional primitive based on the location of the lower-dimensional primitive in the span of the higher-dimensional primitive. This association is then enforced during subsequent manipulation. Typically, higher-dimensional primitives are composed of a number of lower-dimensional primitives, each of which can still be linked to higher-level primitives (*e.g.*, the center point of a Plane primitive is a Point primitive and can be linked to other higher-dimensional primitives.)

To illustrate, consider linking a Point to a Plane. This link operation causes the Point's position to be projected onto the Plane. The Point is then constrained to be at the coordinate associated with that projection point, unless it is moved directly. Whenever the Plane is manipulated, the Point will remain at the same position relative to the origin and orientation of the Plane. Yet, if the Point is manipulated, it will move in the span of the Plane, and thereby change its associated coordinate in the Plane's span.

Some link operations do not fall directly into either category. When this occurs, we chose what we considered the most reasonable solution. For example, we defined the linking of a Geometry primitive to a Length's measure port as a scale operation on the Geometry primitive along the axis of the Length. If the Length's orientation is linked to a principal axis of the Geometry primitive (or vice versa), then the Length acts as a standard 1D scale operation along that axis; otherwise it is a shear.

Figure 2 displays the link behavior that applies to the toolkit primitives when neither primitive has been linked to anything else. In cases where one primitive has already been linked, very different behavior may result; space prevents us from defining all these possibilities. Consider a Point linked to a Plane. The Point becomes constrained to move only in the Plane. If the Point is subsequently linked to a second Point, a different table takes into account the pre-existing constraints on the first Point. In this case, the first Point is constrained to lie at the position of the projection of the second Point onto the Plane.

[3]In the linear algebra sense; a Length's span is the line defined by the endpoints, a Plane's span is the plane defined by the Plane's center point and normal vector.

## 6 Implementation details

The toolkit is implemented in UGA's scripting language, with geometry provided by UGA's interactive solids modeler. The linking constraints between primitives are established using UGA's object-dependency network.

User feedback is provided in the course of a linking operation to aid in link specification. When the user picks a primitive to be linked, it is highlighted and the cursor changes to indicate that the system is waiting for the user to pick the object to link to. After the user picks the object to link to, the system indicates its "ready" state through a cursor change that prompts for a mouse click to confirm the link.

Other highlighting methods indicate a primitive's degrees of freedom. For example, a Ray, like other primitive widgets, is green when it is created, indicating that it is unconstrained. If it is linked to another Ray, its orientation is linked but not its translation, and it turns yellow to indicate a partial constraint. When it is linked again to a Point, it turns red, indicating that all of its degrees of freedom are constrained. Another possibility would have been to change the primitive geometries after linking (*e.g.*, a spherical Point primitive could become a thin cylinder when it is linked to a Length, and could become a disc when linked to a Plane, although this strategy can result in a overly large collection of shapes).

## 7 Future Work

The toolkit as described lacks techniques for specifying range limits on a primitive's degrees of freedom. These would be especially useful when modeling the behavior of real-world objects, or when creating interface objects such as bounded sliders, joints, and dials. We intend to add this functionality (and perhaps other inequality constraints too), and also extend the range of our toolkit to deal with other graphics concerns, such as surface and volumetric modeling, scientific data exploration of scalar and vector fields, and behavior modeling including dynamic simulations.

When two primitives are linked together, a single constraint based on Figure 2 is installed. However, it would often be useful to have a set of possible link behaviors that the user can select from. Advanced users would be able directly select the desired behavior with only a single link operation.

Once a complex widget has been constructed from primitives, it is useful to interactively encapsulate it, along with appropriate parameters, for reuse in a tool library. For example, having constructed a shadow widget, the user should be able to easily apply the same process to any other object. This amounts to interactively defining a function and embodying it in a new, higher-level primitive.

Highly complex widgets linked together from dozens of primitives may present efficiency problems, especially for real-time interaction. It may be necessary to optimize the constraint network after the widget has been completed in order to maximize the toolkit's evaluation speed. It would also be useful to display graphically the constraint relations between primitives to provide feedback on the links established on any widget.

## 8 Conclusions

This toolkit provides a methodology for interactively constructing the geometric behavior of a variety of 3D widgets and parameterized 3D application objects, so that non-technical users can rapidly and interactively generate constrained 3D objects. Previously, such widget construction required programming in C or our scripting language. Even for experienced programmers, graphical construction is a more suitable and efficient environment to conceive, prototype, and implement many types of interactive 3D objects.

destination

| source: Linking constrains | Point | Ray | Length Body | Length Measure | Angle Measure | Plane Frame | Plane Space | Geometry |
|---|---|---|---|---|---|---|---|---|
| Point | positions are equated | Point to lie on Ray | Point to lie on Length body | X | X | Point to lie in Plane | Point to be in Plane's 3D coordinate system | position of Point to position of Geometry |
| Ray | Ray to position of Point | orientations are equated | orientation of Ray to orientation of Length | orientation of Ray to be orientation of length measure | X | orientation and position of Ray to lie in Plane | Ray to be in Plane's 3D coordinate system | orientation of Ray to orientation of Geometry |
| Length Body | X | End Points of Length to lie on Ray | X | X | X | End Points of Length to lie in Plane | End Points of Length to be in Plane's 3D coordinate system | X |
| Length Measure | X | End Points of Length to lie on Ray | X | length of first Length to be length of second Length | length of Length to map to Angle's measure | X | X | X |
| Angle Measure | X | X | X | Angle's measure to map to length of Length | first Angle's measure to be second Angle's measure | X | X | X |
| Geometry | position of Geometry to position of second Point | orientation of Geometry to Ray's orientation | Geometry to lie on Length body | scale of Geometry to length of Length | X | Geometry to lie in Plane | Geometry to be in Plane's 3D coordinate system | positions are equated |

Figure 2: Linking behaviors for unconstrained primitives.

## Acknowledgments

This work was supported in part by the NSF/ARPA Science and Technology Center for Computer Graphics and Scientific Visualization and by ONR Contract N00014-91-J-4052, ARPA Order 8225. We also gratefully acknowledge the sponsorship of IBM, NCR, Sun Microsystems, Hewlett Packard, Digital Equipment Corporation, and NASA. We thank Andries van Dam and the members of the Brown University Graphics Group for their help and support. Please contact the authors for a copy of the accompanying videotape.

## References

[1] AVS, Inc. *AVS Developer's Guide, v. 3.0*, 1991.

[2] A. H. Barr. Global and local deformations of solid primitives. *Computer Graphics (SIGGRAPH '84 Proceedings)*, 18(3):21–30, July 1984.

[3] Eric A. Bier. Snap-dragging in three dimensions. *Computer Graphics (1990 Symposium on Interactive 3D Graphics)*, 24(2):193–204, March 1990.

[4] Stuart K. Card, George G. Robertson, and Jock D. Mackinlay. The information visualizer, an information workspace. In *Proceedings of ACM CHI'91 Conference on Human Factors in Computing Systems*, pages 181–188, 1991.

[5] D. Brookshire Conner, Scott S. Snibbe, Kenneth P. Herndon, Daniel C. Robbins, Robert C. Zeleznik, and Andries van Dam. Three-dimensional widgets. *Computer Graphics (1992 Symposium on Interactive 3D Graphics)*, 25(2):183–188, March 1992.

[6] James D. Foley, Andries van Dam, Steven Feiner, and John F. Hughes. *Computer Graphics: Principles and Practice.* Addison-Wesley, 2nd edition, 1990.

[7] Michael Gleicher and Andrew Witkin. Through-the-lens camera control. *Computer Graphics (SIGGRAPH '92 Proceedings)*, 26(2):331–340, July 1992.

[8] Paul E. Haeberli. Conman: A visual programming language for interactive graphics. *Computer Graphics (SIGGRAPH '88 Proceedings)*, 22(4):103–111, August 1988.

[9] Kenneth P. Herndon, Robert C. Zeleznik, Daniel C. Robbins, D. Brookshire Conner, Scott S. Snibbe, and Andries van Dam. Interactive shadows. *1992 UIST Proceedings*, pages 1–6, November 1992.

[10] Michael Kass. CONDOR: Constraint-based dataflow. *Computer Graphics (SIGGRAPH '92 Proceedings)*, 26(2):321–330, July 1992.

[11] Brad A. Myers, Dario A. Guise, Roger B. Dannenberg, Brad Vander Zanden, David S. Kosbie, Edward Pervin, Andrew Mickish, and Philippe Marchal. GARNET comprehensive support for graphical, highly interactive user interfaces. *IEEE COMPUTER magazine*, pages 71–85, November 1990.

[12] Open Software Foundation. *OSF/Motif Reference Guide.*

[13] Steve Sistare. Graphical interaction techniques in constraint-based geometric modeling. In Steve MacKay and Evelyn M. Kidd, editors, *Graphics Interface '91 Proceedings*, pages 161–164. Canadian Man-Computer Communications Society, March 1991.

[14] Scott S. Snibbe, Kenneth P. Herndon, Daniel C. Robbins, D. Brookshire Conner, and Andries van Dam. Using deformations to explore 3d widget design. *Computer Graphics (SIGGRAPH '92 Proceedings)*, 26(2):351–352, July 1992.

[15] Paul S. Strauss and Rikk Carey. An object-oriented 3d graphics toolkit. *Computer Graphics (SIGGRAPH '92 Proceedings)*, 26(2):341–349, July 1992.

[16] Robert C. Zeleznik, D. Brookshire Conner, Matthias M. Wloka, Daniel G. Aliaga, Nathan T. Huang, Philip M. Hubbard, Brian Knep, Henry Kaufman, John F. Hughes, and Andries van Dam. An object-oriented framework for the integration of interactive animation techniques. *Computer Graphics (SIGGRAPH '91 Proceedings)*, 25(4):105–112, July 1991.



# EXACT: Algorithm and Hardware Architecture for an Improved A-Buffer

Andreas Schilling, Wolfgang Straßer
Universität Tübingen
Bundesrepublik Deutschland *

## Abstract

The EXACT (EXact Area Coverage calculaTion) algorithm presented in this paper solves the Hidden Surface Elimination (HSE) problem on the subpixel level.

The use of subpixel masks for anti-aliasing causes some problems with the HSE on the pixel level that are difficult to overcome. The approximations of the well known A-buffer algorithm are replaced by an exact solution that avoids erratic pixels along intersecting or touching surfaces.

With EXACT the HSE problem on the subpixel level is solved with the help of p-masks. P-masks (priority masks) are subpixel masks that indicate for each subpixel which one of two given planes is closer to the viewer. An algorithm to produce the p-masks in an efficient way and its hardware implementation are presented. The p-mask generator is used in a hardware implementation of an A-buffer algorithm in the form of a rendering pipeline. Of course the algorithm can also be used in software to enhance an existing A-buffer implementation.

The paper ends with the description of the list processing architecture for which the EXACT A-buffer has been built[1].

**CR Categories and Subject Descriptors:** I.3.1 **[Computer Graphics]:** Hardware Architecture - *raster display devices*; I.3.3 **[Computer Graphics]:** Picture/Image generation - *display algorithms*

**Additional Key Words and Phrases:** anti-aliasing, A-buffer, priority-masks, exact area coverage calculation.

*Wilhelm–Schickard–Institut für Informatik, Graphisch–Interaktive Systeme, Auf der Morgenstelle 10/C9, 7400 Tübingen, E-mail: andreas@gris.informatik.uni-tuebingen.de, strasser@gris.informatik.uni-tuebingen.de.

[1]The experiences described here were gained in a research project partly supported by the Commission of the European Communities through the ESPRIT II-Project SPIRIT-workstation, Project No. 2484.



## 1 The Problem: Exact anti-aliasing

Rasterizing produces aliasing artifacts. If a box filter is used to perform anti-aliasing the brightness and color of edge pixels are functions of the pixel area covered by the objects as well as of the object colors. The ideal intensity would be described by the formula $I = \frac{1}{A}\sum_i I_i A_i$, where $A_i$ and $I_i$ are the areas and intensities of the visible surfaces within the pixel and $A$ is the total pixel area. Subpixel masks can be used to calculate the fraction of the pixel area covered by an object. However, if the sample point is outside the polygon, its z-value is more or less useless for a correct HSE. A complete hidden surface elimination for the pixel area is required [5].

## 2 Current Status

A traditional algorithm that approximately evaluates the box-filtered intensity is the A-buffer Algorithm described by Carpenter [1]. The contributions of surfaces that cover a pixel partially are arranged in a list that is sorted front-to-back. Two z-values are stored for each fragment, $z_{min}$ and $z_{max}$. When all fragments have been added to the list, the intensity is calculated in a process called packing. Beginning with the frontmost object the contribution is determined using subpixel masks. For each fragment the exact covered pixel area is stored in addition to the subpixel mask. In certain cases the exact area can be used instead of the subpixel count to calculate the contribution. A subpixel already

 ACM-0-89791-601-8/93/008/0085 $01.50

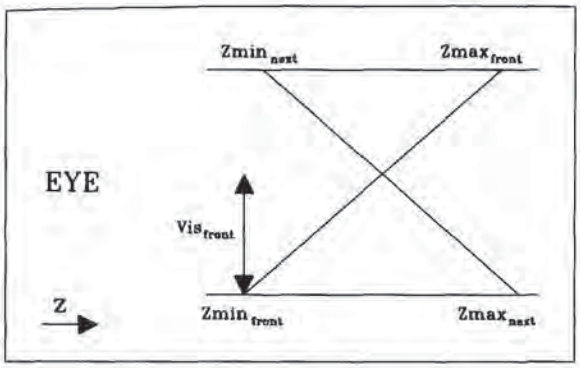



Figure 1: Visible fraction of front fragment ([1]).

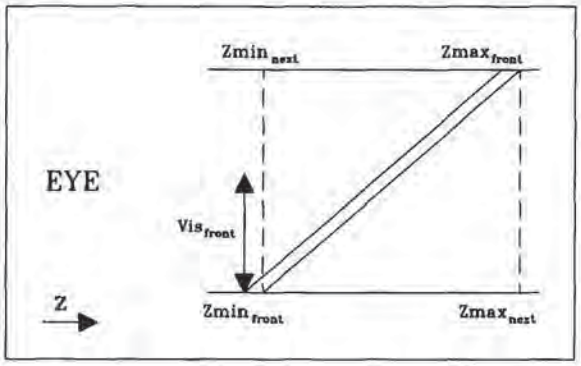



Figure 2: Front fragment should cover the whole pixel.

covered by an opaque object is excluded from further processing which results in a z-buffer-like behavior on the subpixel level. The difference to an actual z-buffer on the subpixel level is that for each fragment only two z values are stored per pixel. Intersecting surfaces are treated with an approximation. Intersection is assumed if the z ranges of two different objects overlap. It is further assumed that the two surfaces are oriented as indicated in Fig. 1.

The visible area of the front fragment is then calculated as:

$$Vis_{front} = \frac{Zmax_{next} - Zmin_{front}}{(Zmax - Zmin)_{front} + (Zmax - Zmin)_{next}}$$

The method will fail very often though, because it depends on assumptions that are hardly ever fulfilled. For example the surfaces in Fig. 2 are rendered exactly like the ones in Fig. 1 although one of the objects is not visible at all.

It should also be mentioned that other even more troublesome[2] problem cases exist that are very difficult

[2]More troublesome: Intersecting surfaces could be forbidden and don't exist in many implementations of rendering systems. But objects touching each other as e.g. in Fig. 3 appear in nearly every picture and cannot be avoided.

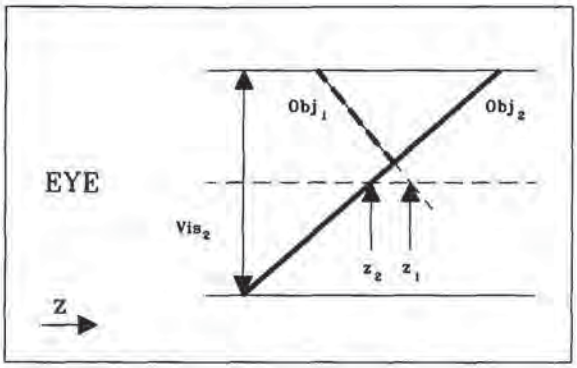



Figure 3: Object 1 disappears (z value sampled at pixel center seems further away).

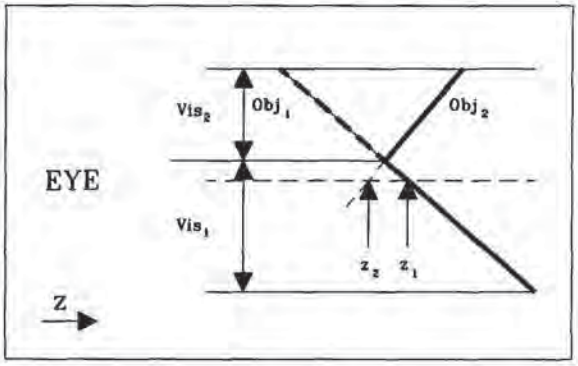



Figure 4: Object 2 shines through (z value sampled at pixel center seems closer).

to handle. If only one $z$-value is available as it is the case in the z-buffer things become especially difficult. If the center of the pixel where the $z$-values are sampled is outside of the object the $z$-values are nearly useless because they don't tell anything about the real location of the object if the slopes in $z$-direction are not known.

Some of the very common problem cases are shown in Fig. 3 - 5. The bold dashed objects are not drawn although they should be visible. These problems are not taken into account with most rendering algorithms. Fig. 13 shows some of the resulting artifacts; the correct image is produced with the EXACT method, described in the following section (Fig. 14).

## 3 Solution

If two objects (or the planes of the two objects resp.) intersect within a pixel a subpixel mask is generated which we call priority mask (p-mask). It indicates in

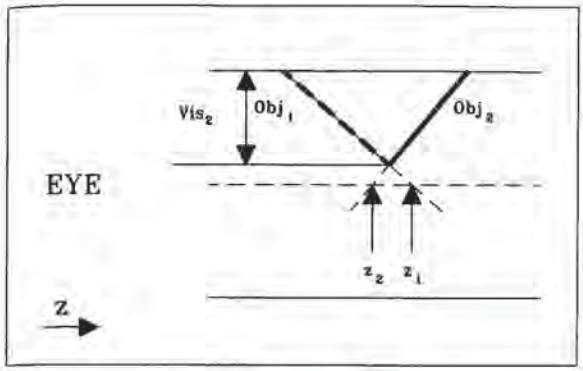



Figure 5: Object 1 disappears, but should be visible, object 2 is visible (situation similar to Fig. 3).

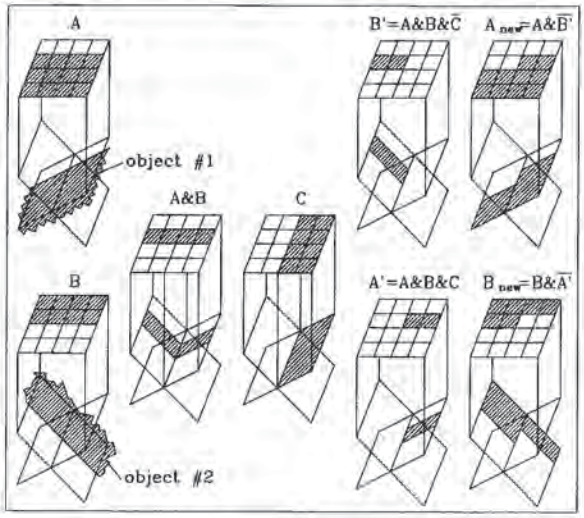



Figure 6: Generation of the modified edge subpixel masks $A_{new}$ and $B_{new}$ from the original edge subpixel masks $A$ and $B$ using the priority mask $C$. Shown is the subdivided pixel area, projected on the planes of two intersecting objects.

which part of the pixel object #1 is the front object and in which part of the pixel object #2 is the front object. This subpixel mask is used to modify the edge subpixel masks of the two objects in the following way (see Fig. 6):

$$A_{new} = A\&\overline{A\&B\&\overline{C}} \quad (1)$$
$$B_{new} = B\&\overline{A\&B\&C} \quad (2)$$

where
A: edge subpixel mask for object # 1
B: edge subpixel mask for object # 2
C: p-mask for objects #1 and #2,
plane #1 in front of plane #2 ⇒ subpixel = 1

Two tasks remain to be solved:

1. The priority mask has to be calculated in an efficient way.
2. The decision has to be made, when two object planes intersect within a pixel's area.

### 3.1 The calculation of the priority mask

Virtually any rasterization system uses a unit that interpolates colors and z-values by repeatedly adding increments to a starting value. The priority mask generator uses the increments for the z value in the x and y directions $dz_x = \frac{\partial z(x,y)}{\partial x}$ and $dz_y = \frac{\partial z(x,y)}{\partial y}$. (The values for the two objects are marked with indices, e.g. $dz_{1,x}$). The z-values at the pixel centers are known ($z_1$ and $z_2$). If we calculate the difference of the corresponding values for the two objects we get:

$$z = z_1 - z_2 \quad (3)$$
$$dz_x = dz_{1,x} - dz_{2,x} \quad (4)$$
$$dz_y = dz_{1,y} - dz_{2,y} \quad (5)$$

These parameters describe a plane that indicates, where plane #1 is in front of plane #2 by the sign of its z-value. The intersection with the plane z=0 denotes the border between the two areas where plane #1 or plane #2 resp. is in front of the other plane.

The representation of this plane with the above mentioned parameters resembles very much the representation of the polygon edges in some rendering systems, e.g. in the PIXEL PLANES system [6]. The mechanisms that exist to generate subpixel masks representing edges can therefore be used to generate the priority mask. A scheme producing subpixel masks that exactly represent the covered fraction of the pixel is described in [8].

The generation of the priority mask can be done by software of hardware. Our contribution aims for a hardware solution. If a software solution is considered, several criteria can be used to reduce significantly the number of cases where the priority mask has to be calculated:

$$A\&B \neq 0 \quad (6)$$

$$z_{1min} < z_{2max} \quad \text{and} \quad z_{2min} < z_{1max} \quad (7)$$

or a much better criterion instead of (7):

$$z_2 - z_1 < (|dz_{2,x} - dz_{1,x}| + |dz_{2,y} - dz_{1,y}|)/2 \quad (8)$$

The first criterion (6) is obvious: if the subpixel masks of the two objects don't overlap, none of the objects can hide the other one.

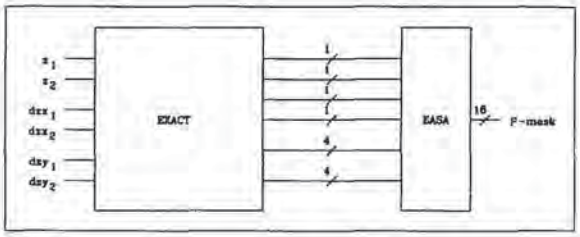



Figure 7: Block diagram of the p-mask generation on the EXACT-Chip

The second criterion is expressed by relations (7). It eliminates the trivial cases where the $z$-ranges of the two objects don't overlap. The priority mask thus consists of only 1s or only 0s, resp. This criterion is not very strong however, because objects with overlapping $z$-ranges do not necessarily have to intersect each other (see e.g. Fig. 2). Also the values of $z_{min}$ and $z_{max}$ might not be known, though they could easily be calculated. This leads us to the stronger criterion expressed in equation (8). Only if this relation is true, will an intersection of the two objects occur within the pixel area. Using this criterion, the case of Fig. 2 is a trivial case with only 0s or 1s in the priority mask.

# 4 Hardware Implementation of the P-Mask Generation

The block diagram of the p-mask generation in Fig. 7 shows, how the mask is calculated. The block labelled EXACT takes two $z$-values and the corresponding increments as input and calculates from these values the parameters of the intersection line. These parameters are used to lookup the final p-mask. The contents of the corresponding lookup table can simply represent the order of the planes at the subpixel locations. It should however, be consistent with the method used for the generation of the coverage masks. The EXACT chip, like the render chip in the SPIRIT workstation [4] uses the EASA concept[3], described in detail in [8].

The design of the EXACT block (Fig.8) is intended to exploit parallelism as much as possible. Three parallel subtractors calculate the $z$-difference and the differences of the $z$-increments. The absolute values of the results are calculated in the next stage.

The resulting three values ($z$, $dz_x$ and $dz_y$ ) are the parameters of the equation (9) for a straight line, the

[3]The EASA (Exact Area Subpixel Algorithm) is used to determine the subpixel mask. In contrast to the conventional approach, we do not sample at the subpixel centers. Instead, the covered portion of the pixel area is calculated exactly and converted into the corresponding subpixel count. The location of the subpixels is chosen in a way, that preserves the geometry best. For details see [8]

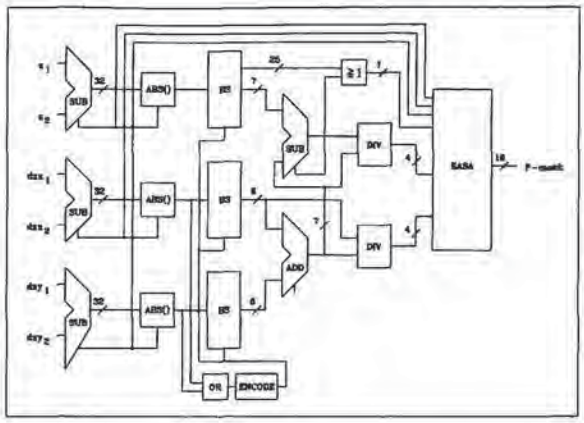



Figure 8: The p-mask generation on the EXACT-Chip ($\approx$ 12000 Gates)

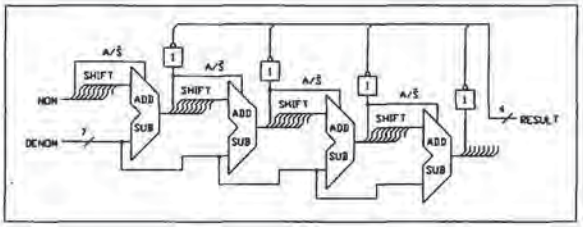



Figure 9: The Dividers on the EXACT-Chip

line of intersection between the two planes (origin of the coordinate system is the pixel center).

$$F(x,y) = z + x * dz_x + y * dz_y = 0 \qquad (9)$$

This equation has to be normalized so that the parameters can be used to look up the resulting p-mask. The normalization could be performed by dividing the equation by $\sqrt{dz_x^2 + dz_y^2}$. However the square root can be avoided if we divide by the $L_1$-norm instead of the $L_2$-norm[4]. This means that we divide by the sum of the absolute values of $dz_x$ and $dz_y$.

The precision that is required so that the error introduced by the parameter calculation is smaller than one subpixel can be found if we apply the law of error propagation. For a $4 \times 4$ subpixel mask, only four bits are needed for each normalized parameter.

To keep the dividers simple (Fig. 9), barrel shifters are used to properly scale the input parameters.

[4]The $L_1$ norm is also known as Manhattan distance, because, rather than the shortest distance, it describes the distance between two points, one would have to walk in a city with a rectangular grid of streets.

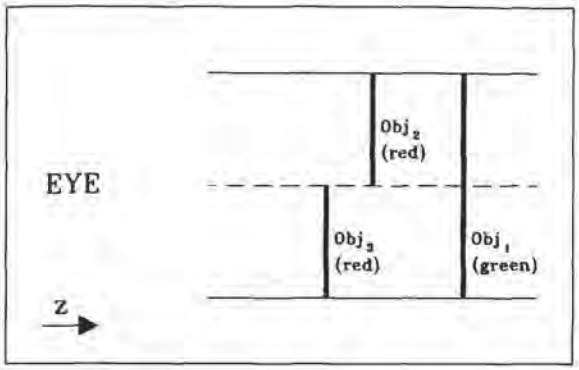



Figure 10: If the colors of object 1 and object 2 are blended (each of them contributing 50% to the final color), green will be part of the pixel color (25%). If the color of object 3 is then blended to the pixel color, green will erroneously still be part of the final pixel color.

# 5 System aspects

The EXACT-hardware is part of a new graphics system. The main concepts of its architecture are described in the following section.

## 5.1 Processing of lists — the concept of the A-buffer

A big difference between the A-buffer and a traditional z-buffer lies in the fact that in the A-buffer lists of contributions to each pixel are stored whereas in the z-buffer only one item per pixel has to be stored — the one currently closest to the viewer. Most rendering hardware today supports the z-buffer for obvious reasons: the list handling required by the A-buffer is much more difficult to implement in hardware.

The question that could be asked at this point is:

**Why should we store more than one object per pixel?**

There are several answers to this question. The first one: Anti-aliasing. The second one: Transparency.

Anti-aliasing of edges implies the blending of the colors of different objects. There are cases in which the colors can be blended using a normal z-buffer. For example, if one object appears in front of an other big object the colors can be blended with the weight factors $A$ and $(1 - A)$, $A$ being the pixel area covered by the second object. But what if three or more objects contribute to a pixel? A blending in the described way will lead to errors (see Fig. 10).

The second reason, transparency handling, is obvious. There may be several transparent objects covering a pixel. They have to be depth-sorted before their colors can be blended using the appropriate transparency factors and sorting requires that more than one object is stored.

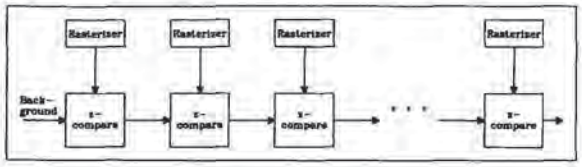



Figure 11: Pipeline of comparators performing n z-buffer operations simultaneously without problem of buffer access bottleneck. While e.g. the third comparator works on pixel #1, the second comparator works on pixel #2 and the first comparator is already working on Pixel #3.

## 5.2 The List Processing Pipeline

Which hardware architecture is capable of supporting an A-buffer like rendering scheme? It is an architecture that has been known quite a while but normally was only used as a functional replacement for the z-buffer: the pixel processing pipeline. Cohen and Demetrescu presented such a processor pipeline already in 1980 [2]. Systems like the Triangle Processor and Normal Vector Shader System [3] or PixelFlow [7] form such a pipeline and use it for what Molnar calls image composition. As multiple z-buffer operations take place at the same time (see Fig. 11), the traditional frame buffer access bottleneck problem is solved in an elegant way. This might be a reason for this type of system to be more widely used in the future. Simply by adding more stages to the pipeline the rendering speed of the system can be increased indefinitely. The only penalty is a slightly increased latency time that up to several hundred pipeline stages doesn't exceed the frame time.

But now this pipeline architecture can not only be used as a z-buffer replacement; it is an outstanding architecture to perform the list processing required by the A-buffer algorithm. Schneider proposed in 1988 the PROOF system [9] that uses a pipeline and transfers not only one object per pixel through the pipeline but a list of contributing objects for each pixel, similar to a proposal by Weinberg [10]. The hidden surface elimination was performed in a special post-processing stage. The architecture proposed in this paper performs the whole list processing in list processors that contain the EXACT hardware for the hidden surface elimination on the subpixel level[5].

[5]Other features of the list processing pipeline, like image processing capabilities (filtering with arbitrary kernel) are not subject of this paper but also are arguments for using such an architecture.

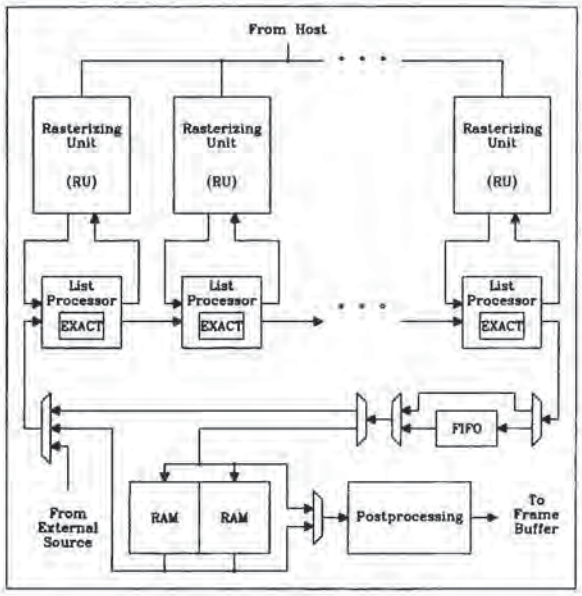



Figure 12: List Processor Pipeline Architecture

Fig. 12 shows the block diagram of a list processor pipeline. The polygon descriptions are distributed in a round robin fashion among the rasterizer units (RU), which ensures a good load distribution with minimum effort. The rasterizers interpolate the z- and color-values (or resp. normals or texture coordinates) and send the sorted pixel contributions down to the list processors. Each rasterization unit is capable of rendering several thousand objects per second (about 20 MPixel/sec.) and contains a standard RISC Processor and RAM as well as an ASIC for the pixel generation.

The list processors, realized as ASICs, contain the described hardware for the EXACT algorithm and perform the modification of the subpixel masks coming from the RUs as well as the depth-sorting of the pixel contributions. Visible fragments are inserted into the lists at their appropriate positions which is important for transparent objects. Mutually intersecting transparent objects can be handled by splitting the subpixel mask of one of the objects in two parts: one in front of, the other behind the second object. The output of the pipeline consists of a depth sorted list of object contributions for each pixel, with nonoverlapping subpixel masks for opaque objects and transparent objects appearing in the correct sequence.

As each list processor can only handle one additional object per pixel, list processors that receive several objects concerning one pixel flag all but the last of these objects as *not processed* and send them in front of the already processed list to the next stage. If this stage didn't receive an object from its RU for this pixel the last of the *not processed* objects is treated by this stage. If any objects remain unprocessed at the end of the pipeline the concerned pixels are cycled through the pipeline again to handle the unresolved objects. In order to keep the sequence of the pixels intact a FIFO is used to store the output of the pipeline during the recycling of the incompletely processed pixels. By adding several list processors without connected RUs to the end of the pipeline the probability for such cases can be significantly reduced.

The output of the pipeline can be directed to one of two RAM buffers. This allows the rendering of scenes with changing parts. The static parts are rendered once into the RAM buffer. Then the RAM serves as input for the pipeline where only the changing parts have to be added for each frame. The RAM buffer is also used in other applications like image processing or form factor calculations for a radiosity algorithm. In the post-processing stage the transparency calculations are performed and the subpixel contributions are summed up.

# 6 Conclusion

A principle of rasterization is, that it produces aliasing artifacts. The quest for increased realism by developing sophisticated illumination models can not be successful without properly dealing with anti-aliasing. This problem can be partially solved by increasing the screen resolution of color monitors, but this is very costly and limited by physical constraints. On the other hand anti-aliasing by means of the EXACT A-buffer solves the problem adequately and offers a better cost/performance ratio for future display systems.

# References

[1] CARPENTER, L. The a-buffer, an antialiased hidden surface method. *Computer Graphics 18*, 3 (July 1984), 103–108.

[2] COHEN, D. A vlsi approach to the cig problem. Presentation at SIGGRAPH 1980, 1980.

[3] DEERING, M., WINNER, S., SCHEDIWY, B., DUFFY, C., AND HUNT, N. The triangle processor and normal vector shader: A vlsi system for high performance graphics. *Computer Graphics 22*, 4 (Aug. 1988), 21–30.

[4] DUNNETT, G. J., WHITE, M., LISTER, P. F., GRIMSDALE, R. L., AND GLEMOT, F. The image chip for high performance 3d rendering. *IEEE Computer Graphics & Applications 12*, 6 (Nov. 1992), 41–52.

[5] FIUME, E., FOURNIER, A., AND RUDOLPH, L. A parallel scan conversion algorithm with anti-

Figure 13: Visible artifacts at edges are the result of using subpixel masks with the standard z-buffer.

Figure 14: Same scene with the EXACT algorithm.

aliasing for a general-purpose ultracomputer. *Computer Graphics 17*, 3 (July 1983), 141–150.

[6] FUCHS, H., POULTON, J., EYLES, J., GREER, T., GOLDFEATHER, J., ELLSWORTH, D., MOLNAR, S., TURK, G., TEBBS, B., AND ISRAEL, L. Pixel-planes 5: A heterogeneous multiprocessor graphics system using processor-enhanced memories. *Computer Graphics 23*, 3 (July 1989), 79–88.

[7] MOLNAR, S. Pixelflow: High-speed rendering unsing image composition. *Computer Graphics 26*, 2 (July 1992), 231–240.

[8] SCHILLING, A. G. A new simple and efficient antialiasing with subpixel masks. *Computer Graphics 25*, 4 (July 1991), 133–141.

[9] SCHNEIDER, B.-O. A processor for an object-oriented rendering system. *Computer Graphics Forum 7* (1988), 301–310.

[10] WEINBERG, R. Parallel processing image synthesis and anti-aliasing. *Computer Graphics 15*, 3 (Aug. 1981), 55–62.



# Graphics Rendering Architecture for a High Performance Desktop Workstation

*Chandlee B. Harrell*
*Farhad Fouladi*

Silicon Graphics Computer Systems
2011 North Shoreline Blvd.
Mountain View, CA 94039-7311

## Abstract

Hundreds of commercial applications used in mainstream design activities have demonstrated proven demand for 3D graphics rendering products. The demand is for faster and more powerful renderers, thus creating the system design problem of how to achieve maximum rendering performance from the technology available to implement the system. This paper describes a graphics rendering architecture that takes advantage of several novel architectural features: a custom floating point processing core with tailored data stores and bussing structures, the arrangement of these cores into a SIMD processor for low overhead multiprocessing, and the hyperpipelining of the fixed point scan conversion units for low overhead, high bandwidth pixel generation into an interleaved frame buffer. These features combine to form a solution to the system design problem which distinguishes itself by its overall performance and its ability to maximize performance while minimizing system size. The resulting architecture is capable of over a half million gouraud shaded Z-buffered triangles per second, with a sustained fill rate for gouraud shaded and Z-buffered pixels of 80M pixels per second. The architecture fits in a desktop workstation.

## Introduction

A graphics rendering architecture for a high performance desktop workstation is described.

3D graphics workstations are used by a broad range of applications [IRIS92]. Many of the applications fall into the categories traditionally called computer-aided design (CAD), where the designer makes progressive refinements on the shape and dimensioning of a product based on feedback from visual modeling, and computer-aided engineering (CAE), where the designer also wishes to analyze properties of the design such as thermal and stress gradients or structural strength, in addition to shape and appearance. 3D graphics workstations are used in the following applications, among others: car and airplane design, tool design, packaging design, industrial and product design, furniture design, clothing and shoe design, architectural and civil engineering, production floor and plant design, geothermal and atmospheric analysis, molecular modeling, pharmaceutical design, chemical analysis, and film animation and special effects.

 ACM-0-89791-601-8/93/008/0093 $01.50

Application packages today running on 3D workstations enable design efforts that are compute intensive, limited only by today's renderers. The complexity of models that renderers can effectively handle is far less than the model complexity with which users are attempting to work. This creates tremendous demand for faster and more powerful graphics rendering systems. How to achieve the highest performance rendering system from the technology available is the system design problem that this demand presents to the system designer.

Further clarification of the graphics rendering system design problem is necessary. Most graphics renderers today perform rapid, accelerated rendering of 3-sided polygons and straight line segments. The renderer receives these basic graphics primitives, each primitive with vertex descriptions defined by the application, and performs the calculations to render the primitive as pixel values into the frame buffer [FOLEY90,SEGAL92,VAND87]. The basic graphics primitives allow close approximation to any arbitrary curve or surface by sub-dividing the curve into line segments or the surface into polygons to the point where the rendered image is visually acceptable to the user. For the system designer, the primitives provide a simple and limited set of processing algorithms that must be accelerated, enabling the focus to achieve high performance systems.

A top level flow diagram is presented in Figure 1 illustrating the process for rendering the basic graphics primitives. The graphics renderer receives polygons or lines from the application process and performs the steps shown in the flow diagram to render each polygon or line as color and Z pixel values into the frame buffer. Details of each processing step are carefully discussed in [FOLEY90] and [NEWM79].

Implementation bottlenecks in a graphics rendering system typically appear: 1) in the floating point compute power available for the world coordinate to screen coordinate transformations and for vertex color computations; 2) in the floating or fixed point compute power available for triangle slope and line slope calculations; 3) in the rate of generation of pixel values from the fixed point iterators; 4) and in the achieved pixel bandwidth into the frame buffer.

Commercial architectures have approached these bottlenecks in a variety of ways. [KIRK90] presents an architecture where the per vertex and slope calculations are performed on the host CPU and multiple iteration engines drive an interleaved frame buffer. [APGAR88] also executes the per vertex calculations on the host, but off-loads most of the slope calculations to a fixed point engine, and uses a unique combination of multiple iteration units to drive pixel results into an interleaved system memory. [AKEL88,89] describe an approach utilizing a serial pipeline of floating point processors for the per vertex calculations, fixed point engines for the slope cal-

culations, and multiple iteration units to drive an interleaved frame buffer. The architecture introduced in [TORB87] also uses multiple floating point processors but arranges them into a MIMD parallel processor, uses a fixed point slope engine, and multiple iterators to drive an interleaved frame buffer. [PERS88] uses a single floating point processor to perform both per vertex and slope calculations, and a single iterator to drive an interleaved frame buffer. Note that the interleaved frame buffer is the only feature common to all the approaches, and that most approaches use multiple iteration units.

The goal of the architecture described here is to provide a powerful graphics rendering system, maximizing performance while minimizing size. The architecture utilizes several novel approaches to overcoming rendering bottlenecks. Floating point performance is accomplished through the custom design of a highly efficient floating point processing core, and by employing multiple cores controlled in a low overhead SIMD parallel processor. The floating point core is tailored to accommodate both the per vertex calculations and the triangle and line slope calculations. Fixed point iteration performance is achieved through hyper-pipelining two identical iteration units, allowing each unit to sustain the pixel generation requirements of multiple pixel memory busses. Each iteration unit is pipelined until technology limits of integration are encountered. The multiple memory busses provide the necessary bandwidth into the frame buffer memory.

These features result in a graphics rendering system solution distinguished by overall performance, and by compactness of size. The architecture is implemented in a desktop workstation [INDIG93]. It is capable of over 1.3 million depth-cued lines per second, over half a million gouraud shaded Z-buffered polygons per second, with a sustained fill rate of 80M gouraud shaded Z-buffered pixels per second.

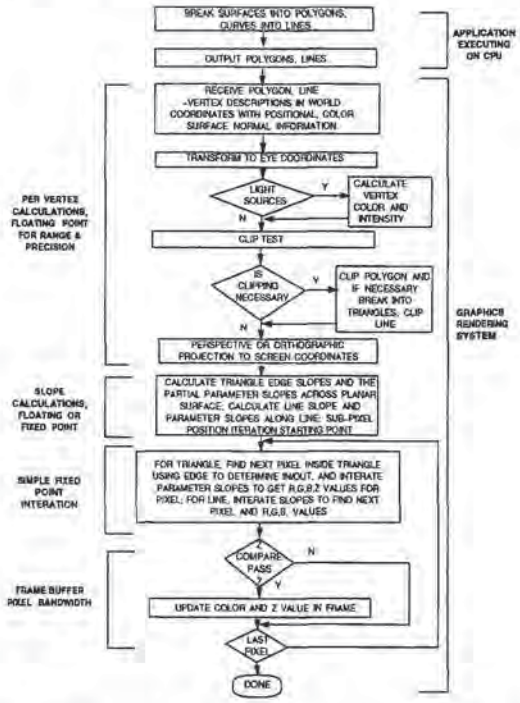



Figure 1. Process for rendering basic graphics primitives

## TOP LEVEL SYSTEM VIEW

This section presents a block diagram of the architecture in Figure 2. The key components are briefly introduced, followed by a description of the overall control structure and the data flow through the system. The subsequent sections discuss each of these key components in detail, describing the critical decisions made to determine their structure, then detailing the internal operation of each component. The final section discusses the technology targeted for the architectural implementation and the implementation results.

The block diagram is shown in Figure 2. The key components are the FIFO interface to the system bus, the Command Processor (CP), the SIMD parallel processor, the dual Raster Engines (RE), and the frame buffer. The SIMD processor is made up of a sequencer, a microcode store, and multiple Geometry Engines (GE). Each GE is a custom floating point processing core. Each Raster Engine is a hyper-pipelined iteration unit.

The SIMD parallel processor executes all the per vertex calculations and the slope calculations shown in Figure 1, the REs perform the fixed point iteration, and the frame buffer pixel bandwidth is determined by the multiple busses into the frame buffer.

Operation is initiated by the CPU sending polygon and line rendering commands into the FIFO across the system bus. The FIFO allows the CPU to generate commands at a rate independent of how fast the rendering occurs. If the FIFO fills up, an interrupt is generated to the CPU for exception handling.

The SIMD parallel processor is fed data from the FIFO by the Command Parser. The CP moves data from the FIFO into the ping-pong input buffers of the Geometry Engines. The GEs read data from the ping-pong buffers, perform necessary floating point computations, and write results to their respective output FIFOs. GE execution is controlled by the common sequencer and control store.

A bus controller resident in the even Raster Engine reads data from the GE output FIFOs and transfers the data into the RE input ping-pong buffers. The REs perform necessary iterations to generate color and Z values and perform the correct pixel updates into the frame buffer. The odd RE generates pixels for the odd numbered scan lines of the frame buffer, and the even RE generates pixels for the even numbered scan lines.

The sections below first discuss the GE custom floating point core solution, followed by a discussion of the control structures required to arrange the GEs into the SIMD parallel processor. This is followed by a description of the hyper-pipelined RE iteration solution.

## GEOMETRY ENGINE

The goal for the Geometry Engine design is to achieve the maximum *realized* floating point performance for graphics algorithms, in a single chip solution. The algorithms used for evaluating performance are the per vertex and slope calculations of Figure 1. The decision is made to combine the per vertex and slope calculations into a single floating point solution. Slope calculations are comprised of relatively complex algorithms, difficult to implement in a hard-wired fashion, and therefore most effectively implemented in a microcoded processor. Also, the compute cycles required for per vertex calculations is almost evenly balanced with the cycles required for slope calculations. Combining the per vertex and slope calculations into the GE relieves the need to design a second microcoded fixed point processor of similar complexity; and the replication of GEs in the SIMD parallel processor increases both the per vertex and slope processing power together.

The GE design goal is met with a custom floating point processing core. Analysis shows that a custom unit with tailored data stores,

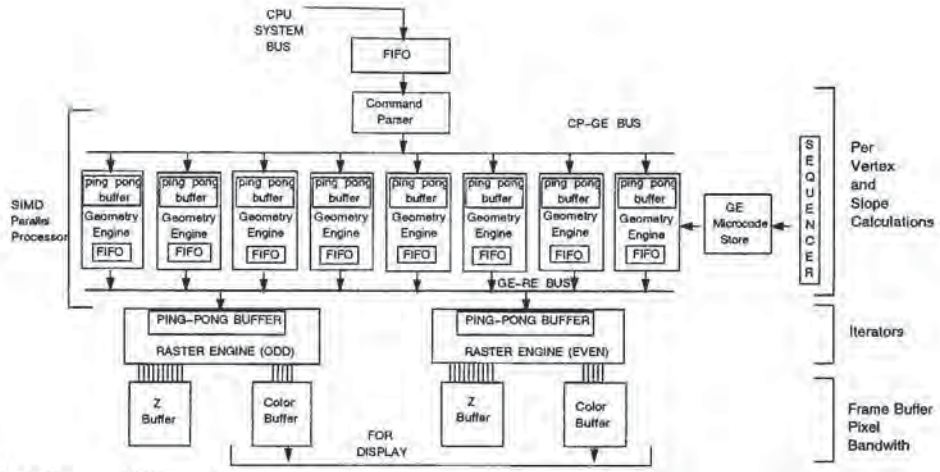



Figure 2. Block Diagram of the architecture

bussing structures, and sequencing control achieves higher realized performance and a more compact solution than available commercial alternatives. Therefore a custom approach is chosen.

Analysis of the per vertex calculations and the slope calculations shows an even balance between multiplies and adds, therefore one multiplier and one adder are chosen for the GE core. The GE design approach follows the fundamental principle of maximizing the utilization of the most expensive resource: the floating point multiplier (FMPY) and the floating point adder (FALU). The following observations for maximizing utilization are taken into account in the GE design: high data bandwidth to the correct operands is needed into the FMPY and FALU; multiple threads of the same algorithm must be active simultaneously. Enough bandwidth to appropriate data storage and data sources is needed to avoid lost cycles waiting on an operand that is slow to retrieve. A single thread of execution may have several additions followed by several multiply operations, thus wasting the FMPY or the FALU until a result is available from the other unit. Multiple threads of execution is the solution.

The Geometry Engine block diagram is shown in Figure 3. Six different busses and four ports from the register file drive the four inputs to the FMPY and the FALU. Two of the busses provide immediate wrap-around of FMPY and FALU results back to their inputs. One bus gives access to the ping-pong buffer loaded by the Command Parser, while two more busses give access to a pair of special data stores. The sixth bus accesses off-chip memory that is used for expansion, and typically holds the global variables for the GE.

A multi-port register file is included for scratch storage of intermediate results. The register file is critical to allowing multiple simultaneous threads of calculation. Feedback paths from FMPY and FALU result outputs are provided for single-threaded operation, but when two threads conflict by needing the same unit for their next computation, then one thread must be stalled by storing the intermediate result in the register file until the appropriate unit becomes free.

On the other hand, a multi-port register file is an expensive commodity and its size is limited. Reviewing the per vertex calculations concludes that the ping-pong buffer and the register file are sufficient to perform the per vertex calculations with maximum FMPY and FALU utilization. On reviewing the slope calculations, however, it is noted that frequently data from each vertex of a triangle, or both vertices of a line, are needed simultaneously during multi-threaded computation. The register file cannot be made big enough to hold the data structures for each vertex. The GE is designed to have three separate data stores, one for each vertex of a triangle or for the two vertices of a line, used during the slope calculation process. The ping-pong buffer is used to hold the data structure for one vertex, while the two special data stores hold the data structures for up to two more vertices.

This extensive memory and bussing structure is wasted without flexible independent addressing and flexible control of data movement. This is accomplished through a very wide instruction word which allows control of the breadth of resources.

The result of the described structure is that simultaneous access can be made to the ping-pong buffer, the two special data stores, the global variables memory, the result outputs, and the register file by any of the four FMPY and FALU inputs. Multiple threads of execution supported by this accessible bandwidth into the FMPY and FALU inputs maximizes FMPY and FALU utilization.

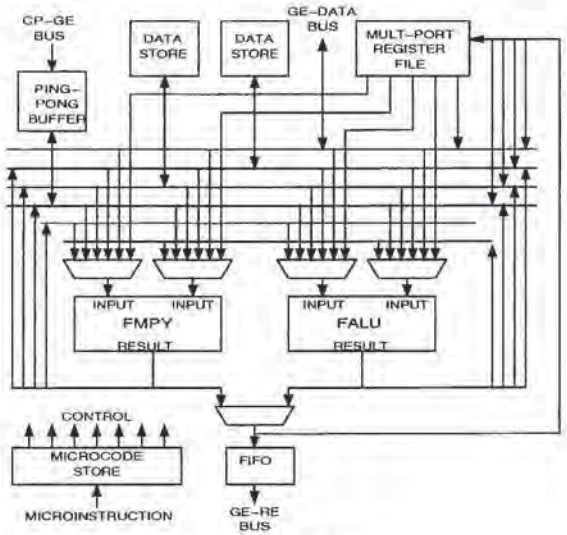



Figure 3. Geometry Engine block diagram

GE operation occurs as follows. The Command Parser loads data into the ping-pong buffer. The ping-pong buffer allows CP loading of data into one side of the buffer while the GE is executing and accessing the other side of the buffer. The CP initiates GE execution by informing the GE sequencer that data is fully loaded. The sequencer looks up instructions in the GE microcode store, and these instructions control the execution functions of the GE. For lines and triangles, the GE performs per vertex calculations, accessing data from the ping-pong buffer, then constructs vertex data structures based on screen space coordinates and puts one vertex data structure back in the ping-pong buffer and up to two more vertex data structures into each of the special data stores. Slope calculations are then performed, drawing operands from the ping-pong buffer and the two special data stores. Calculated iteration coefficients and initial values are passed to the Raster Engines by storing them to the output FIFO.

## SIMD PARALLEL PROCESSOR

A single floating point processor cannot achieve the desired performance. Therefore multiple floating point processors are used in the design. The following goals for multiprocessing led to the SIMD parallel processor solution: 1) a linear performance increase must be achieved with the addition of Geometry Engines; 2) the multiprocessing solution must have the lowest possible impact over and above a uniprocessor solution.

Three approaches are considered for the multiprocessing solution. The first is a pipeline of floating point processors [AKEL88, 89]. Each pipeline stage performs a subset of the per vertex and slope computations, passing intermediate results to the next processor in the pipeline. Each pipeline processor is executing a different set of code to implement its separate subset of the algorithm. This approach has several disadvantages. The throughput of a pipeline is the speed of the slowest processing step. Overall performance is determined by the processor with the biggest subset of the algorithm to process. Since the algorithm cannot be divided into perfectly equal subsets, a less-than-linear performance gain is achieved. Also note, that to add processors, a new subdivision of the algorithm must take place and new code must be written and tuned. The final disadvantage of this approach is in the burden of overhead the approach requires. Although having the advantage of not requiring the distribution mechanism at the head of the pipe needed by the next two approaches considered, each processor does require its own sequencer, microcode store, globals data store, in addition to control logic to interface each of the pipeline stages.

The second approach considered is a parallel MIMD (Multiple Instruction Multiple Data) array of processors [TORB87]. Each processor performs independent execution of the per vertex and slope calculations for its own polygon or line primitive. Linear performance gains are attained when the same kind of primitive is distributed to each processor, thus satisfying the first multiprocessing goal. Processors may be added without requiring changes to processor code. The disadvantage of the MIMD parallel processor lies in the overhead required to implement such an approach. A parallel processor requires a distribution function that takes primitives in the FIFO (received from the CPU) and disburses a primitive to each of the processors present. A MIMD parallel processor also requires that each processor has its own sequencer, microcode store, and globals data store.

The third approach considered is a parallel SIMD (Single Instruction Multiple Data) array of processors. Each processor executes the same instruction in lockstep, but is computing results for its own polygon or line primitive. Like the MIMD processor already examined, the SIMD parallel processor achieves linear performance gains with the addition of processors when the same kind of primitive is distributed to each processor. The advantage of the SIMD approach is in the low overhead required to implement a multiprocessor. All processors share the same sequencer, the same microcode store, and the same globals data memory. The only implementation overhead required over a uniprocessing solution is the addition of the distribution function. It is worth noting that this is a simple function and therefore a small overhead to tolerate. The SIMD parallel processor is chosen as it optimally achieves the multiprocessing goals.

Note that a key assumption to accomplishing linear performance gain from a parallel processor (SIMD or MIMD) is that the same kind of primitive is distributed to each of the processors (all lines or all polygons). This requires that the primitives coming through the FIFO from the CPU arrive in significant groupings of lines together and polygons together, rather than a fully random distribution of lines and polygons. For a MIMD processor, if the FIFO holds alternating lines and polygons, the throughput slows down to the rate of the slower primitive - the polygon. For a SIMD processor, alternating lines and polygons is a worst case scenario. Performance will reduce to that of a uniprocessor. Extensive analysis of model data sets used on 3D workstations shows polygons typically clump in large bunches and lines do the same. This is particularly true of CAD/CAE applications. The result is linear performance gain for parallel processor arrangements.

The unique system features required for SIMD parallel processing will now be discussed. Please refer to Figure 2. The features included for SIMD processing are the distribution function performed by the CP, sequencing functions to allow SIMD branching, common bus for the microinstruction, common bus for the globals data store, and indirect addressing requirements into GE memories. The GE input ping-pong buffer and output FIFO are also crucial to performance.

### COMMAND PARSER

To describe the operation of the Command Parser, we must first explain the needs of the distribution function. The purpose of the CP is to analyze the command and data stream coming through the FIFO, distribute data accordingly to the GEs, and subsequently initiate GE execution. To perform this function, the CP must detect boundaries between primitives, detect whether subsequent primitives are of the same or different kind, and maintain the correct order of primitive disbursement to the GEs.

Please refer to Figure 4 for a diagram of the Command Parser. The CP is microcoded for flexibility. This allows different routines for primitives comprised of vertices with different kinds of attributes, and the exception handling of polygons with greater than three sides.

CP operation begins with the arrival of a command token in the FIFO. The command token causes the CP sequencer to branch to a routine appropriate for the kind of primitive arriving in the FIFO. This branch mechanism inherently defines primitive boundaries. The command token is read from the FIFO and stored in the Current Command register. A compare function allows branching based on whether the current command token just arrived is identical or different from the last command token received. If the token is identical, then the arriving primitive can be distributed to the next GE in the parallel processor. If the token is different, then the GEs that have already been loaded with data must swap their input ping-pong buffer and begin executing before the arriving primitive can be distributed to the next GE. The token compare mechanism allows the CP to branch to different routines to handle these two cases.

The CP must determine to which GE the arriving primitive should be written. A round robin scheme of distribution is chosen,

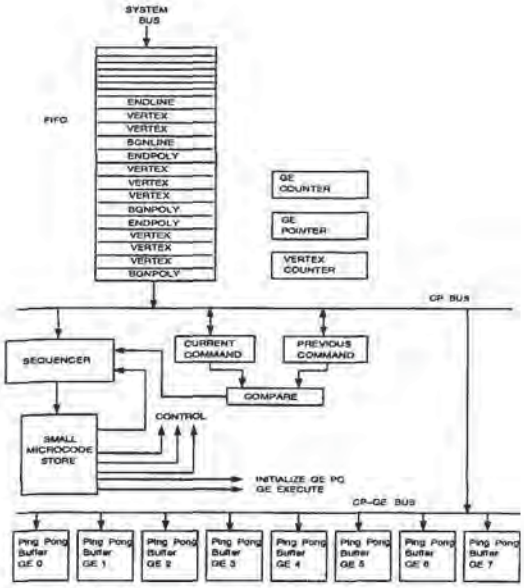


Figure 4. Command Parser block diagram

primitives being loaded in a continuous sequence from GE #0 through to GE #7, and back around. Referring back to Figure 2, primitive coefficients calculated by the GEs are pulled from the GE output FIFOs in the same round robin order. A pointer to the GE that is currently being loaded, and a counter which maintains the number of GEs that have been loaded since the last execute command provide the tools to determine for which GE the arriving primitive is destined. The incrementing and clearing of these counters is under microcode control. After choosing the appropriate GE, the CP pulls vertex data from the FIFO and writes it across the CP-GE Bus and into the GE's ping-pong buffer.

Once all 8 GEs have been loaded, or when the current primitive is different from the previous primitive, the CP must initiate GE execution. The CP first tells the GE sequencer which GEs are loaded, passes the GE sequencer the appropriate address to begin execution, and then issues the GE sequencer an execute command. An interlock mechanism will stall the CP if the GE is currently executing at the time of the CP execute command, and will initiate GE execution only when the previous execution is complete. Once the interlock mechanism clears, it is an indication that the GE ping-pong buffers have been swapped, and the CP resumes distribution of primitives from the FIFO.

### GE SEQUENCER

The GE sequencer is shown in Figure 2. The sequencer is based on a standard uniprocessor design. Flexible branch functions are supported for jumps and subroutine calls. Branching is controlled within separate fields of the GE's wide instruction word. This allows concurrent branching with the GE datapath control, thus not affecting datapath performance thru branches.

To this uniprocessor design base are added functions which allow control of multiple SIMD processors. The GE sequencer has control to stall each of the GEs independently. This control is used in two different ways. The first is on receipt of an execute command from the CP once the GEs are idle. The GE sequencer will decode which GEs the CP has loaded from information passed by the CP. Those GEs not loaded will be stalled by the GE sequencer for the duration of the primitive execution. The second fashion the stall control is used is for implementing conditional subroutine calls across SIMD processors. If a subset of the processors does not pass the condition, that subset is stalled by the GE sequencer for the duration of the subroutine call, while the remaining processors execute the subroutine. As an example, conditional subroutine calls are used for implementing the lighting and clipping branches shown in Figure 1.

### MICROCODE STORE AND GLOBALS MEMORY STORE

The GE sequencer accesses the next microinstruction from the GE microcode store (Figure 2). The microinstruction word controls all the GE internal functions, as well as the GE sequencer. The piece of the microinstruction word controlling the GEs is bussed to all the GEs for simultaneous execution.

Additional memory (not depicted) can be added external to the GEs as an expansion memory to store global variables required in execution. The GE Data Bus (Figure 3) of each GE is bussed together and connected to a globals memory store.

### INDIRECT ADDRESSING

As explained in the section above on the Geometry Engine (Figure 3), data is read from the ping-pong buffer and the two special data stores to perform the slope calculations for a line or triangle. Depending upon orientation of the primitive on the screen, these data stores may need to be accessed differently by different processors. In order to do this effectively in a SIMD processing environment, indirect addressing is provided into these data stores. This minimizes cycles spent out of SIMD lockstep execution and is crucial to SIMD performance.

### INPUT PING-PONG BUFFER AND OUTPUT FIFO

The GE input ping-pong buffer and the GE output FIFO are also crucial to SIMD performance. Without a ping-pong buffer at the input to the GE, the CP would have to load 8 GEs *after* GE execution of the previous primitive completes, eliminating significant parallelism. The FIFO at the GE output allows all GEs to write their results in lockstep execution. Without the FIFO, a SIMD implementation would not be feasible.

## RASTER ENGINE

The goal for the Raster Engine is to obtain the fastest gouraud shaded Z-buffered fill rate in a single chip. It is also desired to be able to use multiple copies of the same chip to obtain further increases in rendering performance.

There are two major bottlenecks in rasterization: pixel generation, and memory bandwidth. Pixel generation, the first bottleneck, can be increased in two different ways. Contemporary architectures have traditionally increased the rate of pixel generation by replicating in parallel the number of fixed point iterators, utilizing enough iterators to achieve the desired pixel rate. Hyper-pipelining a single iteration unit is the approach taken in this architecture. Hyper-pipelining adds pipeline stages to a single iterator until the desired rate of pixel generation is achieved. The pipeline stages added to the iterator require significantly fewer gates than would be required to replicate iterators. Therefore, hyper-pipelining is chosen as the minimum solution for performance. Memory bandwidth, the second bottleneck, is increased by using an interleaved frame buffer across multiple memory banks.

Determining the total number of pipeline stages and the number of memory busses for the RE is a recursive process, and depends on the integration limits of technology. To achieve the maximum fill rates, the iteration pipeline must support a pixel generation rate of

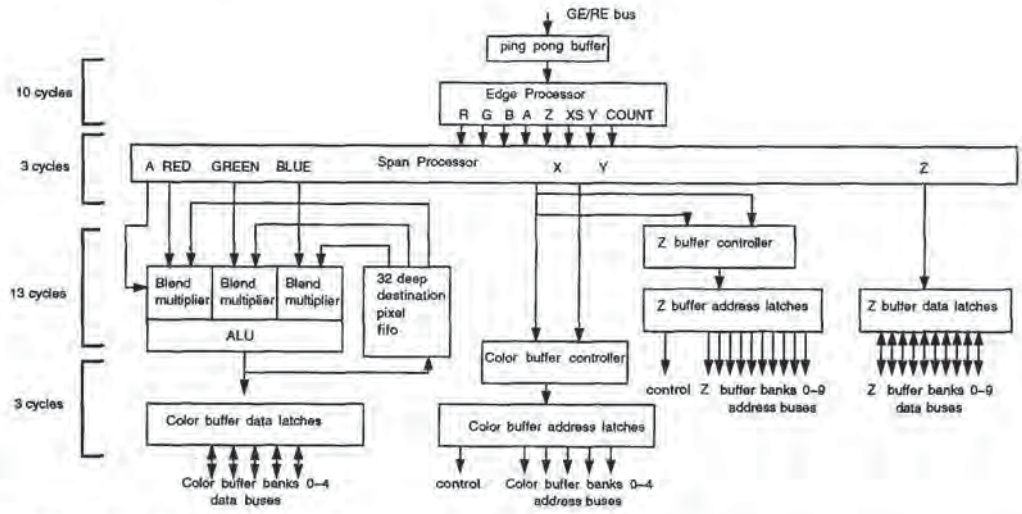



Figure 5. Raster Engine block diagram

N times the page mode bandwidth of a frame buffer DRAM, where N is the number of memory busses used. A sample pipeline depth is analyzed and the die size computed. The conclusion of this recursive process led to the resultant architecture with a single hyper-pipelined Raster Engine driving a five-way interleaved color buffer.

Given a five-way interleave on the color buffer, the pipeline clock rate is set at five times the DRAM page mode bandwidth, under the assumption a pixel is generated every clock. The slowest element of the RE pipeline is the key to ensuring the clock rate can be met, and is what was checked during the recursive analysis. This element is the DDA unit of the iterators. A DDA unit consists of a two input adder with a 2:1 multiplexer on one of its inputs. The output of the adder is fed into a register which is then fed back to the second input of the adder. The resultant clock rate for a five-way interleave color buffer drives the number of pipeline stages in the Raster Engine. The hyper-pipelined Raster Engine has 26 pipeline stages from the input ping-pong registers which hold the line and triangle iteration parameters to the point where pixels are written into the color buffer.

For the system architecture implemented, it is decided to incorporate two raster engines to obtain the desired performance on the desktop.

The RE implementation is now discussed in detail. A diagram of the Raster Engine is shown in Figure 5. The RE is capable of drawing rectangle, triangle and line primitives. Each primitive requires a set of iteration coefficients which are downloaded from the GE FIFOs into the RE ping-pong buffers. Once the ping-pong buffers are loaded, the RE initiates rendering of the primitive.

The execution units of the Raster Engine consist of four major sections:

- > edge processor;
- > span processor;
- > per-pixel operators;
- > memory controllers.

The edge processor combines with the span processor to perform the task of converting a primitive into pixels. The edge processor decomposes triangles into horizontal spans, and decomposes lines into pixels. It has two iterators for computing the beginning and end X location of the span, and six iterators to computer R,G,B,A,Z,Y for the first pixel on the span. Next some terms must be defined. The major edge of a triangle connects the vertex with maximum Y coordinate value to the vertex with minimum Y. The edge connecting the vertex with maximum Y to the vertex with the middle Y value is called the first minor edge. The edge that runs between the vertex with the middle Y and the vertex with the minimum Y value is termed the second minor edge. The edge processor begins by iterating down the major edge and the first minor edge. When the processor detects the middle Y has been crossed, it swaps the first minor edge with the second minor edge and continues down the triangle until the minimum Y coordinate is reached. For each span, the edge processor computes the initial R,G,B,A,X,Y,Z values for the first pixel on the span as well as the number of pixels that have to be rendered for that span. This information is passed to the span processor. When drawing lines, only one of the two edge iterators is used to generate the X coordinate. The edge processor has 10 pipe stages and can generate a new span every other clock.

The span processor has 6 iterators. These iterators walk through the pixels on a span and generate the R,G,B,A,X,Z parameters for each pixel on the span. The processor can generate one or four pixels per clock. When gouraud shading and/or Z-buffering, the span processor will generate one pixel per clock in the X direction. When a span is flat shaded and not Z-buffered, the span processor generates 4 pixels per clock. The block write feature of the VRAMs used in the color buffer is utilized to write all 4 pixels generated in one memory cycle, thus quadrupling the fill performance for screen clears and for rendering flat shaded 2D surfaces. For lines, parameters from the edge processor get passed through. The span processor has a pipeline latency of 3 clocks.

The Raster Engine supports a rich set of pixel operators required by commonly used graphics libraries [SEGAL92, VAND87]. Pixels operators fall into two categories. The first category of operators modify the color of the pixel, such as logicop and blend. Blend and logicop are operations performed between the generated source color and the destination color that is already stored in the color buffer. They require readback from the color buffer which is described below. There are three sets of multipliers to perform the blend function for the R,G,B components. These multipliers are followed by an ALU which performs the logic operations. These two sections together contain 10 pipeline stages.

The second category of pixel operators perform tests on pixel pa-

rameters to allow conditional updating of color pixel values. Examples in this category are the Z-compare test and stencil test. The Z-compare test is used to determine pixel visibility in the third dimension. The stencil test is used to provide more general conditional test operations. The Z-comparison is done in parallel with blend and logicop in the same number of pipeline stages.

There are memory controllers for two separate memory ports on the Raster Engine: the color buffer port and the Z-buffer port. The color buffer is a five-way interleaved memory port, and the Z-buffer is a 10-way interleaved memory port. The Z-buffer operation consists of reading back the old Z value stored in the Z-buffer, comparing that Z value with the newly generated Z value and, if the comparison passes indicating the new pixel is visible, the new Z value and color value are written into the Z-buffer and color buffer respectively. Since the Z-buffer requires two accesses (a read and a write) for every write access to the color buffer, the Z-buffer port is designed with twice the interleaving of the color buffer to accommodate Z-buffered fill at the color gouraud shaded update rates.

As we noted above, a write access to the color buffer takes 5 clocks. Similarly, the pipelined read-modify-write access to the Z-buffer takes 10 clocks. Adjacent pixels along a span are allocated to adjacent banks of the Z-buffer interleave. Since it takes 10 clocks to perform a read-modify-write, and we have a 10-way interleave, bank contention does not occur along a span and a one pixel per clock comparison rate is achieved.

The 10 banks of the Z-buffer interleave share the same page address to reduce memory controller complexity. There is a single block of logic for page fault detection. Each bank can access a different column address within the page. A score boarding technique is used to keep track of the state of each bank. When a pixel is dispatched to a bank, a bit in the score board is set to specify that the bank is busy. Thus, any pixel accesses to the same bank will be blocked and a bank contention stall generated to stop pixel flow until the bank is again idle.

The color buffer has a five-way interleave. As explained above, the pipeline depth is chosen such that five pixels are generated in a single VRAM page mode cycle time, allowing contentionless color fills along a span. Read-modify-write operations to the color frame buffer (for blend and logicop) are supported at half the fill performance of straight color write operations. Values in the color buffer are first read into a FIFO in the RE to await the "modify" step of the operation. When the FIFO fills, the contents of the FIFO are then merged with the newly generated incoming pixel stream and the result is written back into the color buffer. This two-pass operation is continued until rendering is complete. The color buffer memory controller has a 3 clock latency.

The operation of two REs together will be briefly discussed. The two Raster Engines work on the same primitive together. The rendering task is split based on span number. All even spans of a primitive (when the Y coordinate is even) are rendered by the "even" Raster Engine; all odd spans are rendered by the "odd" Raster Engine. This results in a doubling of fill performance. The edge processor in each RE iterates through all spans, but each RE rejects the spans that do not belong to it, and the edge processor continues iteration to the next span.

## TECHNOLOGY

This section briefly discusses the technology used in the implementation. The technology targeted for the custom logic design is a 1.0 micron double metal CMOS gate array and standard cell process. The process can achieve the equivalent of 100K gates on a single die. The 1M-bit DRAM family is the targeted memory technology. The design consists primarily of custom parts and memory components. The design contains over a million gates of custom logic, and is implemented across three 5" x 13" PC boards.

## CONCLUSION

A graphics rendering architecture has been described which is distinguished by its overall performance, and by its ability to maximize performance while minimizing system size. The architecture is shipping as a product in the IRIS Indigo Extreme. A scaled version of the architecture was introduced in IRIS Indigo$^2$ Elan. The architecture provides state-of-the-art rendering performance in a desktop 3D workstation.

## ACKNOWLEDGEMENTS

Sincere thanks to Marc Hannah and Dave Galbi for their major efforts on the architecture. Thanks to Vimal Parikh for his advice on the floating point solution. Finally, overwhelming appreciation must go to the whole design team, every one of whom made significant contributions, and who made it possible.

## REFERENCES

[AKEL88] K. Akeley, T. Jermoluk, "High-Performance Polygon Rendering", Computer Graphics (Proc. SIGGRAPH), Vol. 22, No. 4, August 1988, pp. 239-246.

[[AKEL89] K. Akeley, "The Silicon Graphics 4D/240GTX Superworkstation", IEEE Computer Graphics and Applications, Vol. 9, No. 4, July 1989, pp. 71-83.

[APGAR88] B. Apgar, B. Bersack, A. Mammen, "A display system for the Stellar Graphics Supercomputer Model GS1000", Computer Graphics (Proc. SIGGRAPH), Vol. 22, No. 4, August 1988, pp. 255-262.

[FOLEY90] J. Foley, A. van Dam, S. Feiner, J. Hughes, "Computer Graphics, Principles and Practice", 2nd edition,, Addison-Wesley Publishing, 1990.

[INDIG93] "The Indigo2 Technical Report", Silicon Graphics Computer Systems, 1993.

[IRIS92] "Iris Partner Catalogue", Silicon Graphics Computer Systems, 1992.

[KIRK90] D. Kirk, D. Voorhies, "The Rendering Architecture of the DN10000VS", Computer Graphics (Proc. SIGGRAPH), Vol. 24, No. 4, August 1990, pp. 299-308.

[NEWM79] W. Newman, R. Sproull, "Principles of Interactive Computer Graphics", McGraw-Hill Book Company, Second Edition, 1979.

[PERS88] "The Personnal Iris™: A Technical Report", Silicon Graphics Computer Systems, 1988.

[SEGAL92] M. Segal, K. Akeley, "The OpenGL™ Graphics System: A Specification (version 1.0)", Silicon Graphics Computer Systems, 30 June 1992.

[TORB87] J. Torborg, "A Parrallel Processor Architecture for Graphics Arithmetic Operations", Computer Graphics (Proc. SIGGRAPH), Vol. 21, No. 4, July 1987, pp 197-204.

[VAND87] A. van Dam, et. al., "PHIGS+ Functional Description Rev. 2", Jointly developed PHIGS+ specification, 1987.

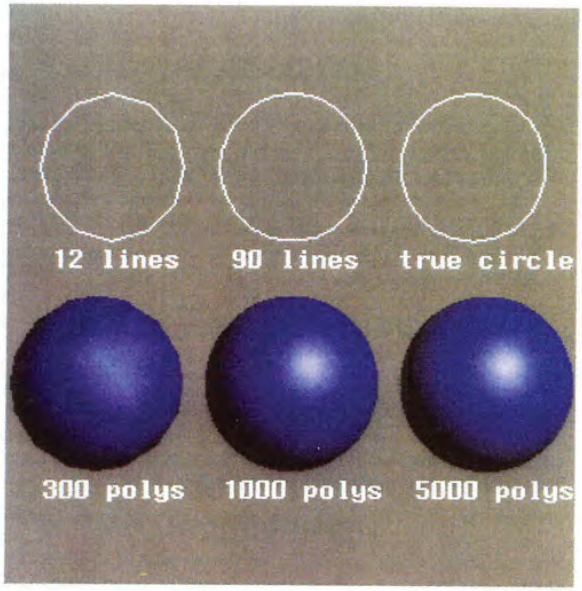



Figure 6. Demonstration of curve and surface approximation using graphics primitives. Note effect of increasing tessellation depth on image quality, and on the number of primitives to render.

Figure 7. Shaded-lighted image (2 directional lights) (Data Courtesy of Cisigraph Corporation) has 31774 triangles, 827961 pixels and was rendered in 0.13 seconds.

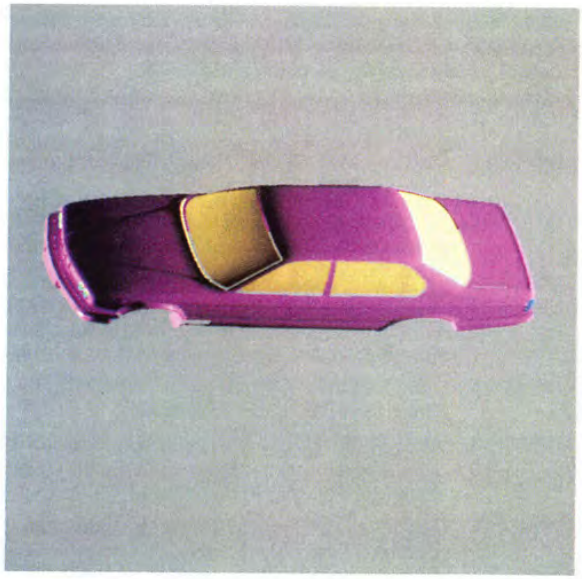


Figure 8. Shaded-lighted image (2 directional lights) (data courtesy of Cisigraph Corporation) has 77420 triangles, 526235 pixels, and was rendered in 0.29 seconds.

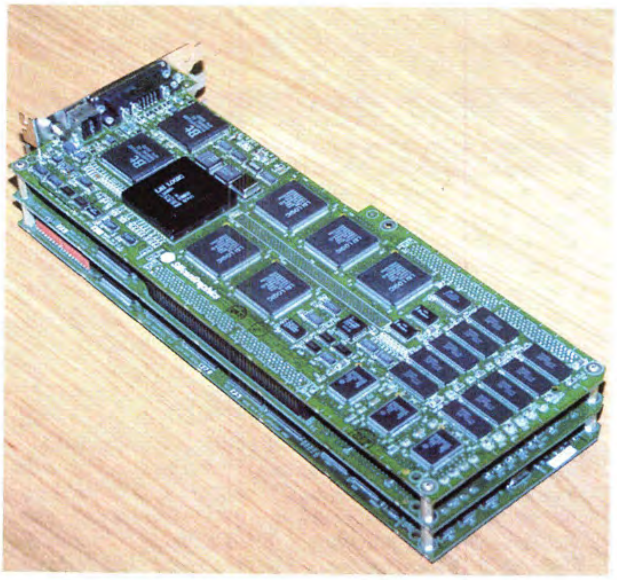


Figure 9. Indigo$^2$ Extreme graphics render board set..

# Leo: A System for Cost Effective 3D Shaded Graphics

*Michael F Deering, Scott R Nelson*
*Sun Microsystems Computer Corporation*[†]

## ABSTRACT

A physically compact, low cost, high performance 3D graphics accelerator is presented. It supports shaded rendering of triangles and antialiased lines into a double-buffered 24-bit true color frame buffer with a 24-bit Z-buffer. Nearly the only chips used besides standard memory parts are 11 ASICs (of four types). Special geometry data reformatting hardware on one ASIC greatly speeds and simplifies the data input pipeline. Floating-point performance is enhanced by another ASIC: a custom graphics microprocessor, with specialized graphics instructions and features. Screen primitive rasterization is carried out in parallel by five drawing ASICs, employing a new partitioning of the back-end rendering task. For typical rendering cases, the only system performance bottleneck is that intrinsically imposed by VRAM.

**CR Categories and Subject Descriptors:** C.1.2 [Processor Architectures]: Multiprocessors; I.3.1 [Computer Graphics]: Hardware Architecture; I.3.3 [Computer Graphics]: Picture/Image Generation *Display algorithms*; I.3.7 [Computer Graphics]: Three Dimensional Graphics and Realism.

**Additional Keywords and Phrases:** 3D graphics hardware, rendering, parallel graphics algorithms, gouraud shading, antialiased lines, floating-point microprocessors.

## 1 INTRODUCTION

To expand the role of 3D graphics in the mainstream computer industry, cost effective, physically small, usable performance 3D shaded graphics architectures must be developed. For such systems, new features and sheer performance at any price can no longer be the driving force behind the architecture; instead, the focus must be on affordable desktop systems.

The historical approach to achieving low cost in 3D graphics systems has been to compromise both performance and image quality. But now, falling memory component prices are bringing nearly ideal frame buffers into the price range of the volume market: double buffered 24-bit color with a 24-bit Z-buffer. The challenge is to drive these memory chips at their maximum rate with a minimum of supporting rendering chips, keeping the total system cost and physical size to an absolute minimum. To achieve this, graphics architectures must be repartitioned to reduce chip count and internal bus sizes, while still supporting existing 2D and 3D functionality.

This paper describes a new 3D graphics system, Leo, designed to these philosophies. For typical cases, Leo's only performance limit is that intrinsically imposed by VRAM. This was achieved by a combination of new architectural techniques and advances in VLSI technology. The result is a system without performance or image quality compromises, at an affordable cost and small physical size. The Leo board set is about the size of one and a half paperback novels; the complete workstation is slightly larger than two copies of Foley and Van Dam [7]. Leo supports both the traditional requirements of the 2D X window system and the needs of 3D rendering: shaded triangles, antialiased vectors, etc.

## 2 ARCHITECTURAL ALTERNATIVES

A generic pipeline for 3D shaded graphics is shown in Figure 1. ([7] Chapter 18 is a good overview of 3D graphics hardware pipeline issues.) This pipeline is truly generic, as at the top level nearly every commercial 3D graphics accelerator fits this abstraction. Where individual systems differ is in the partitioning of this rendering pipeline, especially in how they employ parallelism. Two major areas have been subject to separate optimization: the floating-point intensive initial stages of processing up to, and many times including, primitive set-up; and the drawing-intensive operation of generating pixels within a primitive and Z-buffering them into the frame buffer.

For low end accelerators, only portions of the pixel drawing stages of the pipeline are in hardware; the floating-point intensive parts of the pipe are processed by the host in software. As general purpose processors increase in floating-point power, such systems are starting to support interesting rendering rates, while minimizing cost [8]. But, beyond some limit, support of higher performance requires dedicated hardware for the entire pipeline.

There are several choices available for partitioning the floating-point intensive stages. Historically, older systems performed these tasks in a serial fashion [2]. In time though, breaking the pipe into more pieces for more parallelism (and thus performance) meant that each section was devoting more and more of its time to I/O overhead rather than to real work. Also, computational variance meant that many portions of the pipe would commonly be idle while others were overloaded. This led to the data parallel designs of most recent 3D graphics architectures [12].

[†]2550 Garcia Avenue, MTV18-212
Mountain View, CA 94043-1100
michael.deering@Eng.Sun.COM (415)336-3017
scott.nelson@Eng.Sun.COM (415)336-3106



 ACM-0-89791-601-8/93/008/0101 $01.50

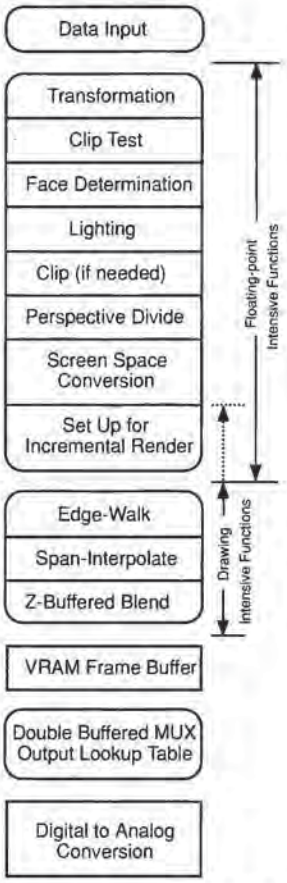



Figure 1: Generic 3D Graphics Pipeline

Here the concept is that multiple parallel computation units can each process the entire floating-point intensive task, working in parallel on different parts of the scene to be rendered. This allows each pipe to be given a large task to chew on, minimizing handshake overhead. But now there is a different load balancing problem. If one pipe has an extra large task, the other parallel pipes may go idle waiting for their slowest peer, if the common requirement of in-order execution of tasks is to be maintained. Minor load imbalances can be averaged out by adding FIFO buffers to the inputs and outputs of the parallel pipes. Limiting the maximum size of task given to any one pipe also limits the maximum imbalance, at the expense of further fragmenting the tasks and inducing additional overhead.

But the most severe performance bottleneck lies in the pixel drawing back-end. The most fundamental constraint on 3D computer graphics architecture over the last ten years has been the memory chips that comprise the frame buffer. Several research systems have attempted to avoid this bottleneck by various techniques [10][4][8], but all commercial workstation systems use conventional Z-buffer rendering algorithms into standard VRAMs or DRAMs. How this RAM is organized is an important defining feature of any high performance rendering system.

## 3 Leo Overview

Figure 2 is a diagram of the Leo system. This figure is *not* just a block diagram; it is also a *chip level* diagram, as every chip in the system is shown in this diagram. All input data and window system interactions enter through the LeoCommand chip. Geometry data is reformatted in this chip before being distributed to the array of LeoFloat chips below. The LeoFloat chips are microcoded specialized DSP-like processors that tackle the floating-point intensive stages of the rendering pipeline. The LeoDraw chips handle all screen space pixel rendering and are directly connected to the frame buffer RAM chips. LeoCross handles the back-end color look-up tables, double buffering, and video timing, passing the final digital pixel values to the RAMDAC.

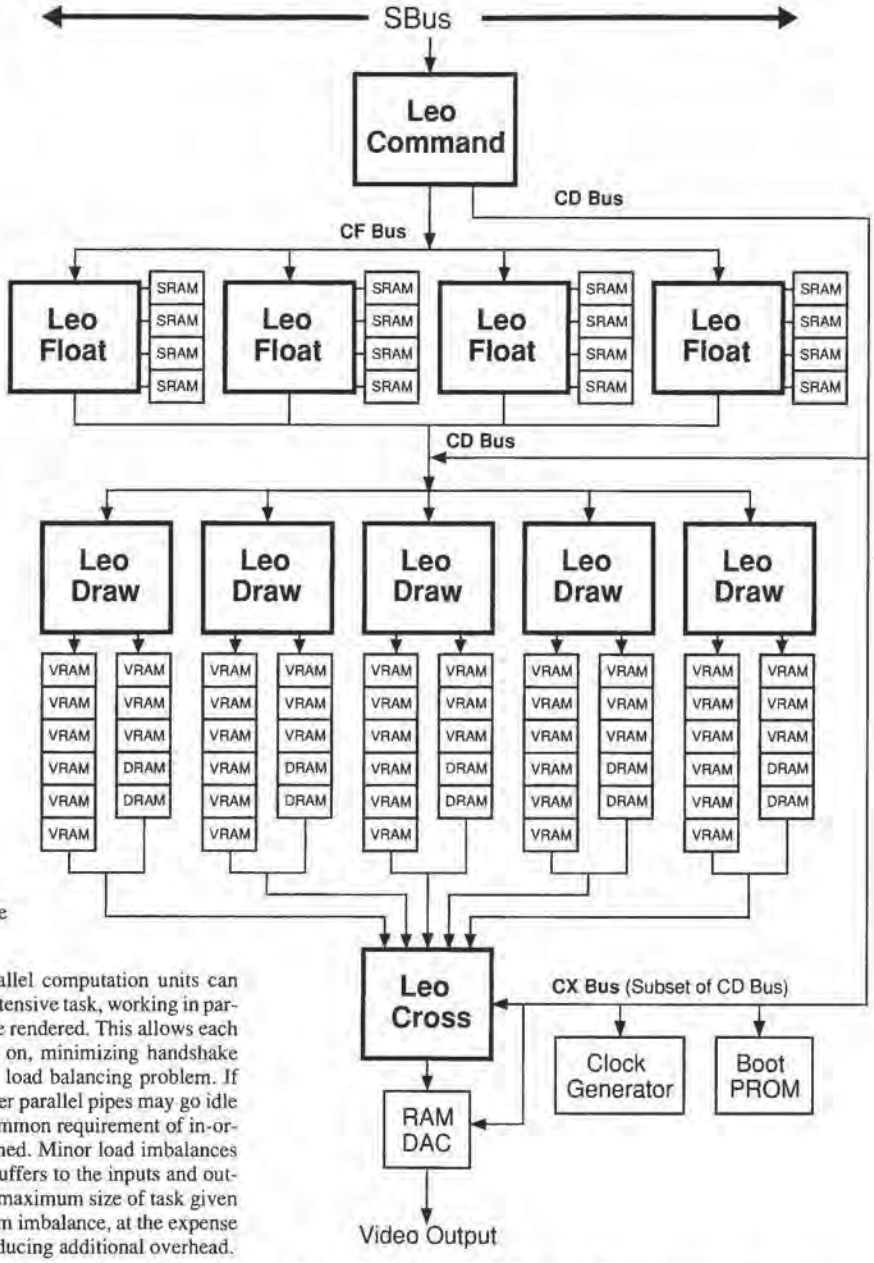



Figure 2: The Leo Block Diagram. Every chip in the system is represented in this diagram.

The development of the Leo architecture started with the constraints imposed by contemporary VRAM technology. As will be derived in the LeoDraw section below, these constraints led to the partitioning of the VRAM controlling LeoDraw chips, and set a maximum back-end rendering rate. This rate in turn set the performance goal for LeoFloat, as well as the data input bandwidth and processing rate for LeoCommand. After the initial partitioning of the rendering pipeline into these chips, each chip was subjected to additional optimization. Throughput bottlenecks in input geometry format conversion, floating-point processing, and pixel rendering were identified and overcome by adding reinforcing hardware to the appropriate chips.

Leo's floating-point intensive section uses data parallel partitioning. LeoCommand helps minimize load balancing problems by breaking down rendering tasks to the smallest isolated primitives: individual triangles, vectors, dots, portions of pixel rasters, rendering attributes, etc., at the cost of precluding optimizations for shared data in triangle strips and polylines. This was considered acceptable due to the very low average strip length empirically observed in real applications. The overhead of splitting geometric data into isolated primitives is minimized by the use of dedicated hardware for this task. Another benefit of converting all rendering operations to isolated primitives is that down-stream processing of primitives is considerably simplified by only needing to focus on the isolated case.

## 4 Input Processing: LeoCommand

### Feeding the pipe

Leo supports input of geometry data both as programmed I/O and through DMA. The host CPU can directly store up to 32 data words in an internal LeoCommand buffer without expensive read back testing of input status every few words. This is useful on hosts that do not support DMA, or when the host must perform format conversions beyond those supported in hardware. In DMA mode, LeoCommand employs efficient block transfer protocols on the system bus to transfer data from system memory to its input buffer, allowing much higher bandwidth than simple programmed I/O. Virtual memory pointers to application's geometry arrays are passed directly to LeoCommand, which converts them to physical memory addresses without operating system intervention (except when a page is marked as currently non-resident). This frees the host CPU to perform other computations during the data transfer. Thus the DMA can be efficient even for pure immediate-mode applications, where the geometry is being created on the fly.

### Problem: Tower of Babel of input formats

One of the problems modern display systems face is the explosion of different input formats for similar drawing functions that need to be supported. Providing optimized microcode for each format rapidly becomes unwieldy. The host CPU could be used to pretranslate the primitive formats, but at high speeds this conversion operation can itself become a system bottleneck. Because DMA completely bypasses the host CPU, LeoCommand includes a programmable format conversion unit in the geometry data pipeline. This reformatter is considerably less complex than a general purpose CPU, but can handle the most commonly used input formats, and at very high speeds.

The geometry reformatting subsystem allows several orthogonal operations to be applied to input data. This geometric input data is abstracted as a stream of vertex packets. Each vertex packet may contain any combination of vertex position, vertex normal, vertex color, facet normal, facet color, texture map coordinates, pick IDs, headers, and other information. One conversion supports arbitrary re-ordering of data within a vertex, allowing a standardized element order after reformatting. Another operation supports the conversion of multiple numeric formats to 32-bit IEEE floating-point. The source data can be 8-bit or 16-bit fixed-point, or 32-bit or 64-bit IEEE floating-point. Additional miscellaneous reformatting allows the stripping of headers and other fields, the addition of an internally generated sequential pick ID, and insertion of constants. The final reformatting stage re-packages vertex packets into complete isolated geometry primitives (points, lines, triangles). Chaining bits in vertex headers delineate which vertices form primitives.

Like some other systems, Leo supports a generalized form of triangle strip (see Figure 3), where vertex header bits within a strip specify how the incoming vertex should be combined with previous vertices to form the next triangle. A stack of the last three vertices used to form a triangle is kept. The three vertices are labeled oldest, middle, and newest. An incoming vertex of type *replace_oldest* causes the oldest vertex to be replaced by the middle, the middle to be replaced by the newest, and the incoming vertex becomes the newest. This corresponds to a PHIGS PLUS triangle strip (sometimes called a "zig-zag" strip). The replacement type *replace_middle* leaves the oldest vertex unchanged, replaces the middle vertex by the newest, and the incoming vertex becomes the newest. This corresponds to a triangle star. The replacement type *restart* marks the oldest and middle vertices as invalid, and the incoming vertex becomes the newest. Generalized triangle strips must always start with this code. A triangle will be output only when a replacement operation results in three valid vertices. *Restart* corresponds to a "move" operation in polylines, and allows multiple unconnected variable-length triangle strips to be described by a single data structure passed in by the user.

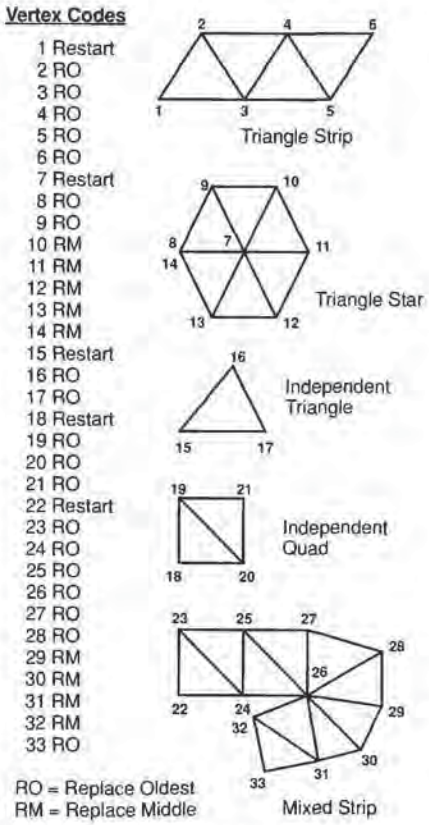



Figure 3: A Generalized Triangle Strip

reducing the overhead. The generalized triangle strip's ability to effectively change from "strip" to "star" mode in the middle of a strip allows more complex geometry to be represented compactly, and requires less input data bandwidth. The restart capability allows several pieces of disconnected geometry to be passed in one DMA operation. Figure 3 shows a *single* generalized triangle strip, and the associated replacement codes. LeoCommand also supports headerless strips of triangle vertices either as pure strips, pure stars, or pure independent triangles.

LeoCommand hardware automatically converts generalized triangle strips into isolated triangles. Triangles are normalized such that the front face is always defined by a clockwise vertex order after transformation. To support this, a header bit in each *restart* defines the initial face order of each sub-strip, and the vertex order is reversed after every *replace_oldest*. LeoCommand passes each completed triangle to the next available LeoFloat chip, as indicated by the input FIFO status that each LeoFloat sends back to LeoCommand. The order in which triangles have been sent to each LeoFloat is scoreboarded by LeoCommand, so that processed triangles are let out of the LeoFloat array in the same order as they entered. Non-sequential rendering order is also supported, but the automatic rendering task distribution hardware works so well that the performance difference is less than 3%. A similar, but less complex vertex repackaging is supported for polylines and multi-polylines via a move/draw bit in the vertex packet header.

To save IC pins and PC board complexity, the internal Leo data busses connecting LeoCommand, LeoFloat, and LeoDraw are 16 bits in size. When colors, normals, and texture map coefficients are being transmitted on the CF-bus between LeoCommand and the LeoFloats, these components are (optionally) compressed from 32-bit IEEE floating-point into 16-bit fixed point fractions by LeoCommand, and then automatically reconverted back to 32-bit IEEE floating-point values by LeoFloat. This quantization does not effect quality. Color components will eventually end up as 8-bit values in the frame buffer. For normals, 16-bit (signed) accuracy represents a resolution of approximately plus or minus an inch at one mile. This optimization reduces the required data transfer bandwidth by 25%.

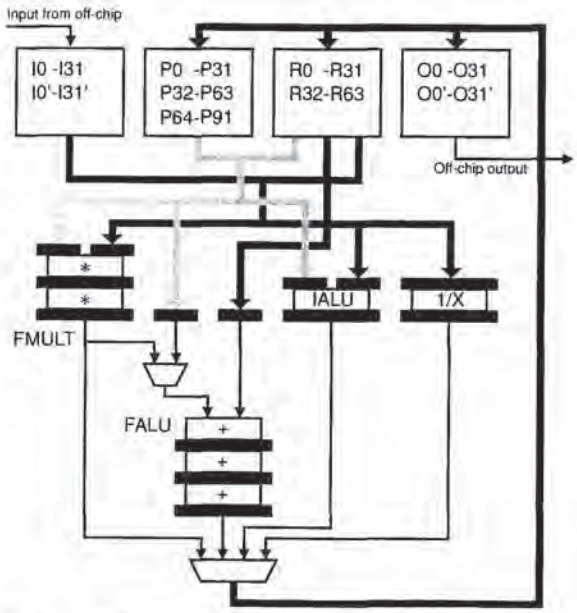



Figure 4: LeoFloat arithmetic function units, registers and data paths.

## 5 Floating-point processing: LeoFloat

After canonical format conversion, the next stages of processing triangles in a display pipeline are: transformation, clip test, face determination, lighting, clipping (if required), screen space conversion, and set-up. These operations are complex enough to require the use of a general purpose processor.

Use of commercially available DSP (Digital Signal Processing) chips for this work has two major drawbacks. First, most such processors require a considerable number of surrounding glue chips, especially when they are deployed as multi-processors. These glue chips can easily quadruple the board area dedicated to the DSP chip, as well as adversely affecting power, heat, cost, and reliability. Second, few of these chips have been optimized for 3D graphics.

A better solution might be to augment the DSP with a special ASIC that would replace all of these glue chips. Given the expense of developing an ASIC, we decided to merge that ASIC with a custom DSP core optimized for graphics.

The resulting chip was LeoFloat. LeoFloat combines a 32-bit microcodable floating-point core with concurrent input and output packet communication subsystems (see Figure 4.), similar to the approach of [3]. The only support chips required are four SRAM chips for external microcode store. A number of specialized graphics instructions and features make LeoFloat different from existing DSP processors. Each individual feature only makes a modest incremental contribution to performance, and indeed many have appeared in other designs. What is novel about LeoFloat is the combination of features, whose cumulative effect leads to impressive overall system performance. The following sections describe some of the more important special graphics instructions and features.

**Double buffered asynchronous I/O register files.** All input and output commands are packaged up by separate I/O packet hardware. Variable length packets of up to 32 32-bit words are automatically written into (or out of) on-chip double-buffered register files (the I and O registers). These are mapped directly into microcode register space. Special instructions allow complete packets to be requested, relinquished, or queued for transmission in one instruction cycle.

**Enough internal registers.** Most commercial DSP chips support a very small number of internal fast registers, certainly much smaller than the data needed by the inner loops of most 3D pipeline algorithms. They attempt to make up for this with on-chip SRAM or data caches, but typically SRAMs are not multi-ported and the caches not user-schedulable. We cheated with LeoFloat. We first wrote the code for the largest important inner loop (triangles), counted how many registers were needed (288), and built that many into the chip.

**Parallel internal function units.** The floating-point core functions (32-bit IEEE format) include multiply, ALU, reciprocal, and integer operations, all of which can often be executed in parallel. It is particularly important that the floating-point reciprocal operation not tie up the multiply and add units, so that perspective or slope calculations can proceed in parallel with the rest of geometric processing. Less frequently used reciprocal square root hardware is shared with the integer function unit.

**Put all non-critical algorithms on the host.** We avoided the necessity of building a high level language compiler (and support instructions) for LeoFloat by moving any code not worth hand coding in microcode to the host processor. The result is a small, clean kernel of graphics routines in microcode. (A fairly powerful macro-assembler with a 'C'-like syntax was built to support the hand coding.)

**Software pipeline scheduling.** One of the most complex parts of modern CPUs to design and debug is their scoreboard section, which schedules the execution of instructions across multiple steps in time and function units, presenting the programmer with the

illusion that individual instructions are executed in one shot. LeoFloat avoided all this hardware by using more direct control fields, like horizontal microprogrammable machines, and leaving it to the assembler (and occasionally the programmer) to skew one logical instruction across several physical instructions.

**Special clip condition codes & clip branch.** For clip testing we employ a modified Sutherland-Hodgman algorithm, which first computes a vector of clip condition bits. LeoFloat has a clip test instruction that computes these bits two at a time, shifting them into a special clip-bits register. After the bits have been computed, special branch instructions decode these bits into the appropriate case: clip rejected, clip accepted, single edge clip (six cases), or needs general clipping. There are separate branch instructions for triangles and vectors. (A similar approach was taken in [9].) The branch instructions allow multiple other conditions to be checked at the same time, including backfacing and model clipping.

**Register Y sort instruction.** The first step of the algorithm we used for setting up triangles for scan conversion sorts the three triangle vertices in ascending Y order. On a conventional processor this requires either moving a lot of data, always referring to vertex data through indirect pointers, or replicating the set-up code for all six possible permutations of triangle vertex order. LeoFloat has a special instruction that takes the results of the last three comparisons and reorders part of the R register file to place vertices in sorted order.

**Miscellaneous.** LeoFloat contains many performance features traditionally found on DSP chips, including an internal subroutine stack, block load/store SRAM, and integer functions. Also there is a "kitchen sink" instruction that initiates multiple housekeeping functions in one instruction, such as "transmit current output packet (if not clip pending), request new input packet, extract op-code and dispatch to next task."

**Code results: equivalent to 150 megaflop DSP.** Each 25 MHz LeoFloat processes the benchmark isolated triangle (including clip-test and set-up) in 379 clocks. (With a few exceptions, microcode instructions issue at a rate of one per clock tick.) The same graphics algorithm was tightly coded on several RISC processors and DSP chips (SPARC, i860, C30, etc.), and typically took on the order of 1100 clocks. Thus the 379 LeoFloat instruction at 25 MHz do the equivalent work of a traditional DSP chip running at 75 MHz (even though there are only 54 megaflops of hardware). Of course these numbers only hold for triangles and vectors, but that's most of what LeoFloat does. Four LeoFloats assure that floating-point processing is not the bottleneck for 100-pixel isolated, lighted triangles.

## 6 Screen Space Rendering: LeoDraw

### VRAM limits

Commercial VRAM chips represent a fundamental constraint on the possible pixel rendering performance of Leo's class of graphics accelerator. The goal of the Leo architecture was to ensure to the greatest extent possible that this was the *only* performance limit for typical rendering operations.

The fundamental memory transaction for Z-buffered rendering algorithms is a conditional read-modify-write cycle. Given an XY address and a computed RGBZ value, the old Z value at the XY address is first read, and then if the computed Z is in front of the old Z, the computed RGBZ value is written into the memory. Such transactions can be mapped to allowable VRAM control signals in many different ways: reads and writes may be batched, Z may be read out through the video port, etc.

VRAM chips constrain system rendering performance in two ways. First, they impose a minimum cycle time per RAM bank for the Z-buffered read-modify-write cycle. Figure 5 is a plot of this cycle

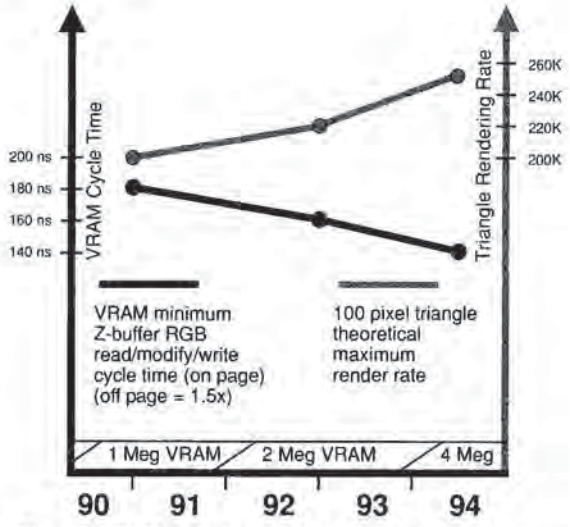



Figure 5: VRAM cycle time and theoretical maximum triangle rendering rate (for five-way interleaved frame buffers).

time (when in "page" mode) and its changes over a half-decade period. VRAMs also constrain the ways in which a frame buffer can be partitioned into independently addressable banks. Throughout the five year period in Figure 5, three generations of VRAM technology have been organized as 256K by 4, 8, and 16-bit memories. For contemporary display resolutions of 1280 × 1024, the chips comprising a minimum frame buffer can be organized into no more than five separately-addressed interleave banks. Combining this information, a theoretical maximum rendering speed for a primitive can be computed. The second line in Figure 5 is the corresponding performance for rendering 100-pixel Z-buffered triangles, including the overhead for entering page mode, content refresh, and video shift register transfers (video refresh). Higher rendering rates are only possible if additional redundant memory chips are added, allowing for higher interleaving factors, at the price of increased system cost.

Even supporting five parallel interleaves has a cost: at least 305 memory interface pins (five banks of (24 RGB + 24 Z + 13 address/control)) are required, more pins than it is currently possible to dedicate to a memory interface on one chip. Some systems have used external buffer chips, but on a minimum cost and board area system, this costs almost as much as additional custom chips. Thus, on the Leo system we opted for five separate VRAM control chips (LeoDraws).

### Triangle scan conversion

Traditional shaded triangle scan conversion has typically been via a linear pipeline of edge-walking followed by scan interpolation [12]. There have been several approaches to achieving higher throughput in rasterization. [2] employed a single edge-walker, but parallel scan interpolation. [4][10] employed massively parallel rasterizers. [6] and other recent machines use moderately parallel rasterizers, with additional logic to merge the pixel rasterization streams back together.

In the Leo design we chose to broadcast the identical triangle specification to five parallel rendering chips, each tasked with rendering only those pixels visible in the local interleave. Each chip performs its own complete edge-walk and span interpolation of the triangle, biased by the chip's local interleave. By paying careful attention to proper mathematical sampling theory for rasterized pixels, the five

chips can act in concert to produce the correct combined rasterized image. Mathematically, each chip thinks it is rasterizing the triangle into an image memory with valid pixel centers only every five original pixels horizontally, with each chip starting off biased one more pixel to the right.

To obtain the speed benefits of parallel chips, most high performance graphics systems have split the edge-walk and span-interpolate functions into separate chips. But an examination of the relative amounts of data flow between rendering pipeline stages shows that the overall peak data transfer bandwidth demand occurs between the edge-walk and span-interpolate sections, induced by long thin triangles, which commonly occur in tessellated geometry. To minimize pin counts and PC board bus complexity, Leo decided to replicate the edge-walking function into each of the five span-interpolation chips.

One potential drawback of this approach is that the edge-walking section of each LeoDraw chip will have to advance to the next scan line up to five times more often than a single rasterization chip would. Thus LeoDraw's edge-walking circuit was designed to operate in one single pixel cycle time (160 ns. read-modify-write VRAM cycle), so it would never hold back scan conversion. Other usual pipelining techniques were used, such as loading in and buffering the next triangle to be drawn in parallel with rasterizing the current triangle. Window clipping, blending, and other pixel post processing are handled in later pipelined stages.

### Line scan conversion

As with triangles, the mathematics of the line rasterization algorithms were set up to allow distributed rendering of aliased and antialiased lines and dots, with each LeoDraw chip handling the 1/5 of the frame buffer pixels that it owns. While the Leo system uses the X11 semantics of Bresenham lines for window system operations, these produce unacceptable motion artifacts in 3D wireframe rendering. Therefore, when rendering 3D lines, Leo employs a high-accuracy DDA algorithm, using 32 bits internally for sufficient subpixel precision.

At present there is no agreement in the industry on the definition of a high quality antialiased line. We choose to use the image quality of vector strokers of years ago as our quality standard, and we tested different algorithms with end users, many of whom were still using calligraphic displays. We found users desired algorithms that displayed no roping, angle sensitivities, short vector artifacts, or end-point artifacts. We submitted the resulting antialiased line quality test patterns as a GPC [11] test image. In achieving the desired image quality level, we determined several properties that a successful line antialiasing algorithm must have. First, the lines must have at least three pixels of width across the minor axis. Two-pixel wide antialiased lines exhibit serious roping artifacts. Four-pixel wide lines offer no visible improvement except for lines near 45 degrees. Second, proper endpoint ramps spread over at least two pixels are necessary both for seamless line segment joins as well as for isolated line-ends. Third, proper care must be taken when sampling lines of subpixel length to maintain proper final intensity. Fourth, intensity or filter adjustments based on the slope are necessary to avoid artifacts when rotating wireframe images. To implement all this, we found that we needed at least four bits of subpixel positional accuracy *after* cumulative interpolation error is factored in. That is why we used 32 bits for XY coordinate accuracy: 12 for pixel location, 4 for subpixel location, and 16 for DDA interpolation error. (The actual error limit is imposed by the original, user-supplied 32-bit IEEE floating-point data.)

Because of the horizontal interleaving and preferred scan direction, the X-major and Y-major aliased and antialiased line rasterization algorithms are not symmetric, so separate optimized algorithms were employed for each.

### Antialiased dots

Empirical testing showed that only three bits of subpixel precision are necessary for accurate rendering of antialiased dots. For ASIC implementation, this was most easily accomplished using a brute-force table lookup of one of 64 precomputed $3 \times 3$ pixel dot images. These images are stored in on-chip ROM, and were generated using a circular symmetric Gaussian filter.

### Triangle, line, and dot hardware

Implementation of the triangle and antialiased vector rasterization algorithms require substantial hardware resources. Triangles need single pixel cycle edge-walking hardware in parallel with RGBZ span interpolation hardware. To obtain the desired quality of antialiased vectors, our algorithms require hardware to apply multiple waveform shaping functions to every generated pixel. As a result, the total VLSI area needed for antialiased vectors is nearly as large as for triangles. To keep the chip die size reasonable, we reformulated both the triangle and antialiased vector algorithms to combine and reuse the same function units. The only difference is how the separate sequencers set up the rasterization pipeline.

### Per-pixel depth cue

Depth cueing has long been a heavily-used staple of wireframe applications, but in most modern rendering systems it is an extra time expense feature, performed on endpoints back in the floating-point section. We felt that we were architecting Leo not for benchmarks, but for users, and many wireframe users want to have depth cueing on all the time. Therefore, we built a parallel hardware depth cue function unit into each LeoDraw. Each triangle, vector, or dot rendered by Leo can be optionally depth cued at absolutely no cost in performance. Another benefit of per-pixel depth cueing is full compliance with the PHIGS PLUS depth cueing specification. For Leo, per-pixel depth cueing hardware also simplifies the LeoFloat microcode, by freeing the LeoFloats from ever having to deal with it.

### Picking support

Interactive graphics requires not only the rapid display of geometric data, but also interaction with that data: the ability to pick a particular part or primitive within a part. Any pixels drawn within the bounds of a 3D pick aperture result in a pick hit, causing the current pick IDs to be automatically DMAed back to host memory.

### Window system support

Many otherwise sophisticated 3D display systems become somewhat befuddled when having to deal simultaneously with 3D rendering applications and a 2D window system. Modern window systems on interactive workstations require frequent context switching of the rendering pipeline state. Some 3D architectures have tried to minimize the overhead associated with context switching by supporting multiple 3D contexts in hardware. Leo goes one step further, maintaining two completely separate pipelines in hardware: one for traditional 2D window operations; the other for full 3D rendering. Because the majority of context switch requests are for 2D window system operations, the need for more complex 3D pipeline context switching is significantly reduced. The 2D context is much lighter weight and correspondingly easier to context switch. The two separate graphics pipelines operate completely in parallel, allowing simultaneous access by two independent CPUs on a multiprocessor host.

2D functionality abstracts the frame buffer as a 1-bit, 8-bit, or 24-bit pixel array. Operations include random pixel access, optimized character cell writes, block clear, block copy, and the usual menagerie of

boolean operations, write masks, etc. Vertical block moves are special cased, as they are typically used in vertical scrolling of text windows, and can be processed faster than the general block move because the pixel data does not have to move across LeoDraw chip interleaves. Rendering into non-rectangular shaped windows is supported by special clip hardware, resulting in no loss in performance. A special block clear function allows designated windows (and their Z-buffers) to be initialized to any given constant in under 200 microseconds. Without this last feature, 30 Hz or faster animation of non-trivial objects would have been impossible.

## 7 VIDEO OUTPUT: LEOCROSS

Leo's standard video output format is $1280 \times 1024$ at 76 Hz refresh rate, but it also supports other resolutions, including $1152 \times 900$, interlaced $640 \times 480$ RS-170 (NTSC), interlaced $768 \times 576$ PAL timing, and $960 \times 680$ 113 Hz field sequential stereo. LeoCross contains several color look-up tables, supporting multiple pseudo color maps without color map flashing. The look-up table also supports two different true color abstractions: 24-bit linear color (needed by rendering applications), and REC-709 non-linear color (required by many imaging applications).

### Virtual reality support

Stereo output is becoming increasingly important for use in Virtual Reality applications. Leo's design goals included support for the Virtual Holographic Workstation system configuration described in [5]. Leo's stereo resolution was chosen to support square pixels, so that lines and antialiased lines are displayed properly in stereo, and standard window system applications can co-exist with stereo. Stereo can be enabled on a per-window basis (when in stereo mode windows are effectively quad-buffered). Hooks were included in LeoCross to support display technologies other than CRT's, that may be needed for head-mounted virtual reality displays.

## 8 NURBS AND TEXTURE MAP SUPPORT

One of the advantages to using programmable elements within a graphics accelerator is that additional complex functionality, such as NURBS and texture mapping, can be accelerated. Texture mapping is supported through special LeoFloat microcode and features of LeoCommand. LeoFloat microcode also includes algorithms to accelerate dynamic tessellation of trimmed NURBS surfaces. The dynamic tessellation technique involves reducing trimmed NURBS surfaces into properly sized triangles according to a display/pixel space approximation criteria [1]; i.e. the fineness of tessellation is view dependent. In the past, dynamic tessellation tended to be mainly useful as a compression technique, to avoid storing all the flattened triangles from a NURBS surface in memory. Dynamic tessellation was not viewed as a performance enhancer, for while it might generate only a third as many triangles as a static tessellation, the triangles were generated at least an order of magnitude or more slower than brute force triangle rendering. In addition it had other problems, such as not handling general trimming. For many cases, Leo's dynamic tesselator can generate and render triangles only a small integer multiple slower than prestored triangle rendering, which for some views, can result in *faster* overall object rendering.

## 9 RESULTS

Leo is physically a-two board sandwich, measuring $5.7 \times 6.7 \times 0.6$ inches, that fits in a standard 2S SBus slot. Figure 6 is a photo of the two boards, separated, showing all the custom ASICs. Figure 7 is a photo of the complete Leo workstation, next to two of our units of scale and the board set.

Leo can render 210K 100-pixel isolated, lighted, Gouraud shaded, Z-buffered, depth cued triangles per second, with one infinite diffuse and one ambient light source enabled. At 100 pixels, Leo is still VRAM rendering speed limited; smaller triangles render faster. Isolated 10-pixel antialiased, constant color, Z-buffered, depth cued lines (which are actually 12 pixels long due to endpoint ramps, and three pixels wide) render at a 422K per second rate. Corresponding aliased lines render at 730K. Aliased and antialiased constant color, Z-buffered, depth cued dots are clocked at 1100K. 24-bit image rasters can be loaded onto the screen at a 10M pixel per second rate. Screen scrolls, block moves, and raster character draws all also have competitive performance. Figure 8 is a sample of shaded triangle rendering.

## 10 SIMULATION

A system as complex as Leo cannot be debugged after the fact. All the new rendering mathematics were extensively simulated before being committed to hardware design. As each chip was defined, high, medium, and low level simulators of its function were written and continuously used to verify functionality and performance. Complete images of simulated rendering were generated throughout the course of the project, from within weeks of its start. As a result, the window system and complex 3D rendering were up and running on a complete board set within a week of receiving the first set of chips.

## 11 CONCLUSIONS

By paying careful attention to the forces that drive both performance and cost, a physically compact complete 3D shaded graphics accelerator was created. The focus was not on new rendering features, but on cost reduction and performance enhancement of the most useful core of 3D graphics primitives. New parallel algorithms were developed to allow accurate screen space rendering of primitives. Judicious use of hardware to perform some key traditional software functions (such as format conversion and primitive vertex reassembly) greatly simplified the microcode task. A specialized floating-point core optimized for the primary task of processing lines and triangles also supports more general graphics processing, such as rasters and NURBS. The final system performance is limited by the *only* chips not custom designed for Leo: the standard RAM chips.

## ACKNOWLEDGEMENTS

The authors would like to thank the entire Leo team for their efforts in producing the system, and Mike Lavelle for help with the paper.

## REFERENCES

1. **Abi-Ezzi, Salim, and L. Shirman.** Tessellation of Curved Surfaces under Highly Varying Transformations. Proc. Eurographics '91 (Vienna, Austria, September 1991), 385-397.
2. **Akeley, Kurt and T. Jermoluk.** High-Performance Polygon Rendering, Proceedings of SIGGRAPH '88 (Atlanta, GA, Aug 1-5, 1988). In *Computer Graphics* 22, 4 (July 1988), 239-246.
3. **Anido, M., D. Allerton and E. Zaluska.** MIGS - A Multiprocessor Image Generation System using RISC-like Microprocessors. Proceedings of CGI '89 (Leeds, UK, June 1989), Springer Verlag 1990.
4. **Deering, Michael, S. Winner, B. Schediwy, C. Duffy and N. Hunt.** The Triangle Processor and Normal Vector Shader: A VLSI system for High Performance Graphics. Proceedings of SIGGRAPH '88 (Atlanta, GA, Aug 1-5, 1988). In *Computer Graphics* 22, 4 (July 1988), 21-30.

5. **Deering, Michael.** High Resolution Virtual Reality. Proceedings of SIGGRAPH '92 (Chicago, IL, July 26-31, 1992). In *Computer Graphics* 26, 2 (July 1992), 195-202.
6. **Dunnett, Graham, M. White, P. Lister and R. Grimsdale.** The Image Chip for High Performance 3D Rendering. *IEEE Computer Graphics and Applications* 12, 6 (November 1992), 41-52.
7. **Foley, James, A. van Dam, S. Feiner and J Hughes.** Computer Graphics: Principles and Practice, 2nd ed., Addison-Wesley, 1990.
8. **Kelley, Michael, S. Winner, K. Gould.** A Scalable Hardware Render Accelerator using a Modified Scanline Algorithm. Proceedings of SIGGRAPH '92 (Chicago, IL, July 26-31, 1992). In *Computer Graphics* 26, 2 (July 1992), 241-248.
9. **Kirk, David, and D. Voorhies.** The Rendering Architecture of the DN10000VS. Proceedings of SIGGRAPH '90 (Dallas, TX, August 6-10, 1990). In *Computer Graphics* 24, 4 (August 1990), 299-307.
10. **Molnar, Steven, J. Eyles, J. Poulton.** PixelFlow: High-Speed Rendering Using Image Composition. Proceedings of SIGGRAPH '92 (Chicago, IL, July 26-31, 1992). In *Computer Graphics* 26, 2 (July 1992), 231-240.
11. **Nelson, Scott.** GPC Line Quality Benchmark Test. GPC Test Suite, NCGA GPC committee 1991.
12. **Torborg, John.** A Parallel Processor Architecture for Graphics Arithmetic Operations. Proceedings of SIGGRAPH '87 (Anaheim, CA, July 27-31, 1987). In *Computer Graphics* 21, 4 (July 1987), 197-204.

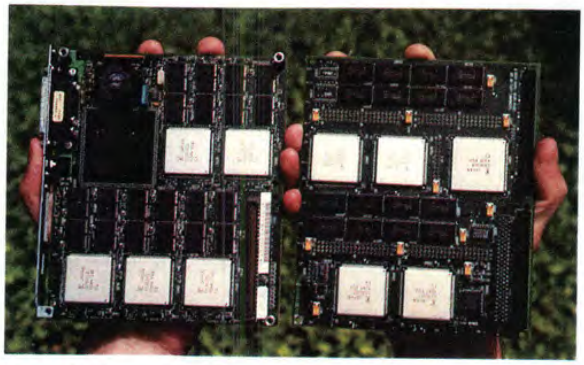



Figure 6: The two boards, unfolded.


Figure 7: The complete SPARCstation ZX workstation, next to two of our units of scale and the Leo board set.

Figure 8: **Traffic Jam to Point Reyes**. A scene containing 2,322,000 triangles, rendered by Leo Hardware. Stochastically super-sampled 8 times. Models courtesy of Viewpoint Animation Engineering.



# RealityEngine Graphics

Kurt Akeley
Silicon Graphics Computer Systems*

## Abstract

The RealityEngine™ graphics system is the first of a new generation of systems designed primarily to render texture mapped, antialiased polygons. This paper describes the architecture of the RealityEngine graphics system, then justifies some of the decisions made during its design. The implementation is near-massively parallel, employing 353 independent processors in its fullest configuration, resulting in a measured fill rate of over 240 million antialiased, texture mapped pixels per second. Rendering performance exceeds 1 million antialiased, texture mapped triangles per second. In addition to supporting the functions required of a general purpose, high-end graphics workstation, the system enables realtime, "out-the-window" image generation and interactive image processing.

**CR Categories and Subject Descriptors:** I.3.1 **[Computer Graphics]:** Hardware Architecture; I.3.7 **[Computer Graphics]:** Three-Dimensional Graphics and Realism - *color, shading, shadowing, and texture*

## 1 Introduction

This paper describes and to a large extent justifies the architecture chosen for the RealityEngine graphics system. The designers think of this system as our first implementation of a third-generation graphics system. To us a generation is characterized not by the scope of capabilities of an architecture, but rather by the capabilities for which the architecture was primarily designed – the target capabilities with maximized performance. Because we designed our first machine in the early eighties, our notion of first generation corresponds to this period. Floating point hardware was just becoming available at reasonable prices, framebuffer memory was still quite expensive, and application-specific integrated circuits (ASICs) were not readily available. The resulting machines had workable transformation capabilities, but very limited framebuffer processing capabilities. In particular, smooth shading and depth buffering, which require substantial framebuffer hardware and memory, were not available. Thus the target capabilities of first-generation machines were the transformation and rendering of flat-shaded points, lines, and polygons. These primitives were not lighted, and hidden surface elimination, if required, was accomplished by algorithms implemented by the application. Examples of such systems are the Silicon Graphics Iris 3000 (1985) and the Apollo DN570 (1985). Toward the end of the first-generation period advances in technology allowed lighting, smooth shading, and depth buffering to be implemented, but only with an order of magnitude less performance than was available to render flat-shaded lines and polygons. Thus the target capability of these machines remained first-generation. The Silicon Graphics 4DG (1986) is an example of such an architecture.

Because first-generation machines could not efficiently eliminate hidden surfaces, and could not efficiently shade surfaces even if the application was able to eliminate them, they were more effective at rendering wireframe images than at rendering solids. Beginning in 1988 a second-generation of graphics systems, primarily workstations rather than terminals, became available. These machines took advantage of reduced memory costs and the increased availability of ASICs to implement deep framebuffers with multiple rendering processors. These framebuffers had the numeric ability to interpolate colors and depths with little or no performance loss, and the memory capacity and bandwidth to support depth buffering with minimal performance loss. They were therefore able to render solids and full-frame scenes efficiently, as well as wireframe images. The Silicon Graphics GT (1988)[11] and the Apollo DN590 (1988) are early examples of second-generation machines. Later second-generation machines, such as the Silicon Graphics VGX[12] the Hewlett Packard VRX, and the Apollo DN10000[4] include texture mapping and antialiasing of points and lines, but not of polygons. Their performances are substantially reduced, however, when texture mapping is enabled, and the texture size (of the VGX) and filtering capabilities (of the VRX and the DN10000) are limited.

The RealityEngine system is our first third-generation design. Its target capability is the rendering of lighted, smooth shaded, depth buffered, texture mapped, antialiased triangles. The initial target performance was 1/2 million such triangles per second, assuming the triangles are in short strips, and 10 percent intersect the viewing frustum boundaries. Textures were to be well filtered (8-sample linear interpolation within and between two mipmap[13] levels) and large enough (1024 × 1024) to be usable as true images, rather than simply as repeated *textures*. Antialiasing was to result in high-quality images of solids, and was to work in conjunction with depth buffering, meaning that no application sorting was to be required. Pixels were to be filled at a rate sufficient to support 30Hz rendering of full-screen images. Finally, the performance on second-generation primitives (lighted, smooth shaded, depth buffered) was to be no lower than that of the VGX, which renders roughly 800,000 such mesh triangles per second. All of these goals were achieved.

The remainder of this paper is in four parts: a description of the architecture, some specifics of features supported by the architecture, alternatives considered during the design of the architecture, and finally some appendixes that describe performance and implementation details.

*2011 N. Shoreline Blvd., Mountain View, CA 94043 USA, kurt@sgi.com



ACM-0-89791-601-8/93/008/0109 $01.50

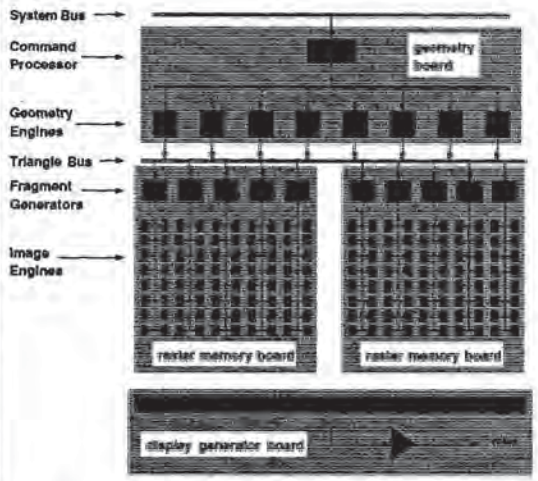



Figure 1. Board-level block diagram of an intermediate configuration with 8 Geometry Engines on the geometry board, 2 raster memory boards, and a display generator board.

## 2 Architecture

The RealityEngine system is a 3, 4, or 6 board graphics accelerator that is installed in a MIPS RISC workstation. The graphics system and one or more MIPS processors are connected by a single system bus. Figure 1 is a board-level block diagram of the RealityEngine graphics accelerator. The geometry board comprises an input FIFO, the Command Processor, and 6, 8, or 12 Geometry Engines. Each raster memory board comprises 5 Fragment Generators (each with its own complete copy of the texture memory), 80 Image Engines, and enough framebuffer memory to allocate 256 bits per pixel to a $1280 \times 1024$ framebuffer. The display generator board supports all video functions, including video timing, genlock, color mapping, and digital-to-analog conversion. Systems can be configured with 1, 2, or 4 raster memory boards, resulting in 5, 10, or 20 Fragment Generators and 80, 160, or 320 Image Engines.

To get an initial notion of how the system works, let's follow a single triangle as it is rendered. The position, color, normal, and texture coordinate commands that describe the vertexes of the triangle in object coordinates are queued by the input FIFO, then interpreted by the Command Processor. The Command Processor directs all of this data to one of the Geometry Engines, where the coordinates and normals are transformed to eye coordinates, lighted, transformed to clip coordinates, clipped, and projected to window coordinates. The associated texture coordinates are transformed by a third matrix and associated with the window coordinates and colors. Then window coordinate slope information regarding the red, green, blue, alpha, depth, and texture coordinates is computed.

The projected triangle, ready for rasterization, is then output from the Geometry Engine and broadcast on the Triangle Bus to the 5, 10, or 20 Fragment Generators. (We distinguish between pixels generated by rasterization and pixels in the framebuffer, referring to the former as fragments.) Each Fragment Generator is responsible for the rasterization of 1/5, 1/10, or 1/20 of the pixels in the framebuffer, with the pixel assignments finely interleaved to insure that even small triangles are partially rasterized by each of the Fragment Generators. Each Fragment Generator computes the intersection of the set of pixels that are fully or partially covered by the triangle and the set of pixels in the framebuffer that it is responsible for, generating a fragment for each of these pixels. Color, depth, and texture coordinates are assigned to each fragment based on the initial and slope values computed by the Geometry Engine. A subsample mask is assigned to the fragment based on the portion of each pixel that is covered by the triangle. The local copy of the texture memory is indexed by the texture coordinates, and the 8 resulting samples are reduced by linear interpolation to a single color value, which then modulates the fragment's color.

The resulting fragments, each comprising a pixel coordinate, a color, a depth, and a coverage mask, are then distributed to the Image Engines. Like the Fragment Generators, the Image Engines are each assigned a fixed subset of the pixels in the framebuffer. These subsets are themselves subsets of the Fragment Generator allocations, so that each Fragment Generator communicates only with the 16 Image Engines assigned to it. Each Image Engine manages its own dynamic RAM that implements its subset of the framebuffer. When a fragment is received by an Image Engine, its depth and color sample data are merged with the data already stored at that pixel, and a new aggregate pixel color is immediately computed. Thus the image is complete as soon as the last primitive has been rendered; there is no need for a final framebuffer operation to resolve the multiple color samples at each pixel location to a single displayable color.

Before describing each of the rendering operations in more detail, we make the following observations. First, after it is separated by the Command Processor, the stream of rendering commands merges only at the Triangle Bus. Second, triangles of sufficient size (a function of the number of raster memory boards) are processed by almost all the processors in the system, avoiding only 5, 7, or 11 Geometry Engines. Finally, small to moderate FIFO memories are included at the input and output of each Geometry Engine, at the input of each Fragment Generator, and at the input of each Image Engine. These memories smooth the flow of rendering commands, helping to insure that the processors are utilized efficiently.

### 2.1 Command Processor

That the Command Processor is required at all is primarily a function of the OpenGL™ [8][7] graphics language. OpenGL is modal, meaning that much of the state that controls rendering is included in the command stream only when it changes, rather than with each graphics primitive. The Command Processor distinguishes between two classes of this modal state. OpenGL commands that are expected infrequently, such as matrix manipulations and lighting model changes, are broadcast to all the Geometry Engines. OpenGL commands that are expected frequently, such as vertex colors, normals, and texture coordinates, are shadowed by the Command Processor, and the current values are bundled with each rendering command that is passed to an individual Geometry Engine. The Command Processor also breaks long connected sequences of line segments or triangles into smaller groups, each group passing to a single Geometry Engine. The size of these groups is a tradeoff between the increased vertex processing efficiency of larger groups (due to shared vertexes within a group) and the improved load balancing that results from smaller groups. Finally, because the Command Processor must interpret each graphics command, it is also able to detect invalid command sequences and protect the

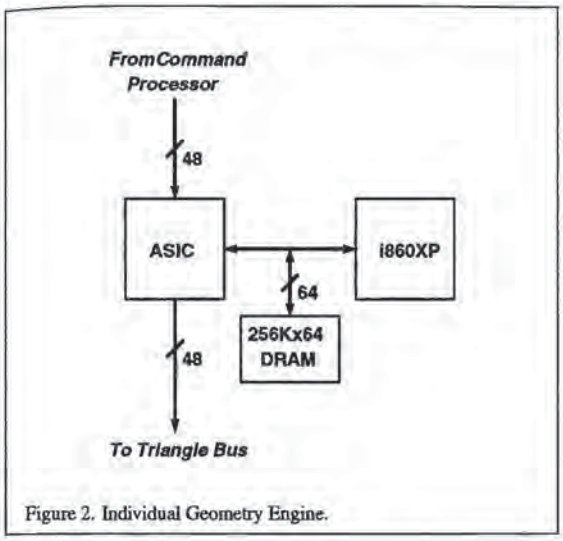



Figure 2. Individual Geometry Engine.

subsequent processors from their effects.

Non-broadcast rendering commands are distributed to the Geometry Engines in pure round-robin sequence, taking no account of Geometry Engine loading. This approach was chosen for its simplicity, and is efficient because the processing requirements of primitives are usually very similar, and because the input and output FIFOs of each Geometry Engine smooth the imbalances due to data-dependent processing such as clipping.

## 2.2 Geometry Engines

The core of each Geometry Engine is an Intel i860XP processor. Operating at 50MHz, the combined floating point multiplier and ALU can achieve a peak performance of 100 MFLOPS. Each Intel processor is provided 2 Mbytes of combined code/data dynamic memory, and is supported by a single ASIC that implements the input and output FIFOs, a small register space from which the i860XP accesses incoming commands, and specialized data conversion facilities that pack computed slope data into a format accepted by the Fragment Generators. (Figure 2.)

All Geometry Engine code is first developed in C, which is cross compiled for the i860XP on MIPS RISC development systems. Code that is executed frequently is then re-coded in i860XP assembly code, showing the greatest improvement in performance where scheduling of the vector floating point unit is hand optimized. The assembly code is written to conform to the compiler's link conventions, so that hand-coded and compiled modules are interchangeable for development and documentation purposes.

Most floating point arithmetic is done in single precision, but much of the texture arithmetic, and all depth arithmetic after projection transformation, must be done in double precision to maintain the required accuracy. After transformation, lighting, and clipping, the rasterization setup code treats each parameter as a plane equation, computing its signed slope in the positive X and Y screen directions. Because the parameters of polygons with more than 3 vertexes may be non-planar, the Geometry Engine decomposes all polygons to triangles.

## 2.3 Triangle Bus

The Triangle Bus acts as a crossbar, connecting the output of each Geometry Engine to the inputs of all the Fragment Generators. Because all Geometry Engine output converges at this bus, it is a potential bottleneck. To avoid performance loss, the Triangle Bus was designed with bandwidth to handle over one million shaded, depth buffered, texture mapped, antialiased triangles per second, more than twice the number of primitives per second that were anticipated from an 8 Geometry Engine system. This performance cushion allows the later-conceived 12 Geometry Engine system to render at full performance, in spite of the greater than expected performance of the individual engines.

In addition to broadcasting the rasterization data for triangles to the Fragment Generators, the Triangle Bus broadcasts point and line segment descriptions, texture images, and rasterization mode changes such as blending functions.

## 2.4 Fragment Generators

Although each Fragment Generator may be thought of as a single processor, the data path of each unit is actually a deep pipeline. This pipeline sequentially performs the initial generation of fragments, generation of the coverage mask, texture address generation, texture lookup, texture sample filtering, texture modulation of the fragment color, and fog computation and blending. These tasks are distributed among the four ASICs and eight dynamic RAMs that comprise each Fragment Generator. (Figure 3.)

Fragments are generated using Pineda arithmetic[9], with the algorithm modified to traverse only pixels that are in the domain of the Fragment Generator. A coverage mask is generated for 4, 8, or 16 sample locations, chosen on a regular $8 \times 8$ subsample grid within the square boundaries of the pixel. The hardware imposes no constraints on which subset of the 64 subsample locations is chosen, except that the same subset is chosen for each pixel. The subset may be changed by the application between frames.

Depth and texture coordinate sample values are always computed at the center-most sample location, regardless of the fragment coverage mask. The single depth sample is later used by the Image Engines to derive accurate depth samples at each subpixel location, using the X and Y depth slopes. Taking the texture sample at a consistent location insures that discontinuities are avoided at pixels that span multiple triangles. Color sample values are computed at the center-most sample location only if it is within the perimeter of the triangle. Otherwise the color sample is taken at a sample location within the triangle perimeter that is near the centroid of the covered region. Thus color samples are always taken within the triangle perimeter, and therefore never wrap to inappropriate values.

Based on a level-of-detail (LOD) calculation and the texture coordinate values at the fragment center, the addresses of the eight texels nearest the sample location in the mipmap of texture images are produced. Eight separate banks of texture memory are then accessed in parallel at these locations. The 8 16-bit values that result are merged with a trilinear blend, based on the subtexel coordinates and the LOD fraction, resulting in a single texture color that varies smoothly from frame to frame in an animation. The entire bandwidth of the 8-bank texture memory is consumed by a single Fragment Engine, so each Fragment Engine includes its own complete copy of all texture images in its texture memory, allowing all Fragment Generators to operate in parallel. Separate FIFO memories on the address and data ports of each texture memory bank

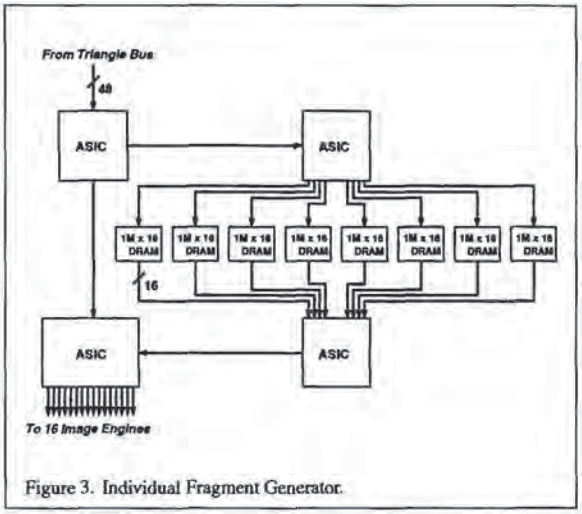



Figure 3. Individual Fragment Generator.

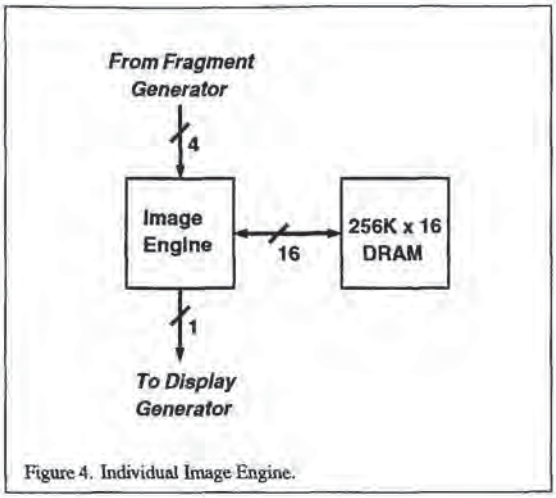



Figure 4. Individual Image Engine.

insure that random page boundary crossings do not significantly degrade the bandwidth available from the dynamic RAMs.

The last ASIC in the Fragment Generator applies the texture color to the fragment's smooth shaded color, typically by modulation. It then indexes its internal fog table with the fragment's depth value and uses the resulting fog blend factor (computed by linear interpolation between the two nearest table entries) to blend the fragment color with the application-defined fog color.

## 2.5 Image Engines

Fragments output by a single Fragment Generator are distributed equally among the 16 Image Engines connected to that generator. When the triangle was first accepted by the Fragment Generator for processing, its depth slopes in the X and Y screen directions were broadcast to each Image Engine, which stored them for later use. When an Image Engine accepts a fragment, it first uses these two slope values and the fragment's depth sample value to reconstruct the depth values at each subpixel sample location. The arithmetic required for this operation is simplified because the subpixel sample locations are fixed to a regular $8 \times 8$ grid. The calculations are linear because depth values have been projected to window coordinates just like the X and Y pixel coordinates. At each sample location corresponding to a '1' in the fragment's coverage mask, the computed depth value is compared to the depth value stored in the framebuffer. If the comparison succeeds, the framebuffer color at that subsample location is replaced by the fragment color, and the framebuffer depth is replaced by the derived fragment depth. If any change is made to the pixel's contents, the aggregate pixel color is recomputed by averaging the subpixel sample colors, and is immediately written to the displayable color buffer that will contain the final image.

Each Image Engine controls a single $256K \times 16$ dynamic RAM that comprises its portion of the framebuffer. (Figure 4.) When the framebuffer is initialized, this memory is partitioned equally among 4K, 8K, or 16K pixels, resulting in pixels with 1024, 512, or 256 bits. All subsample depth and color samples, as well as the one, two, or four displayable color buffers and other auxiliary buffers, are stored in this memory. By default, colors are stored with 12 bits per red, green, blue, and alpha component in both the displayable buffers and the subpixel samples. Depth values are 32 bits each, and are normally required only for each subpixel sample, not for the displayable color buffer or buffers. Color and depth sample resolutions can be reduced to 8,8,8 and 24 bits to allow more samples to be stored per pixel. The 4K partition stores 8 high-resolution samples per pixel, or 16 low-resolution samples per pixel, in addition to two displayable color buffers of the same resolution. The 8K partition stores 4 high-resolution samples per pixel, or 8 low-resolution samples per pixel, again with two displayable color buffers of the same resolution. The 16K partition cannot be used to support multisample antialiasing.

Because the number of raster memory boards (1, 2, or 4) and the number of pixels per Image Engine (4K, 8K, or 16K) are independent, the RealityEngine system supports a wide variety of framebuffer dimensions, color and depth resolutions, and subpixel samples. For example, a single raster board system supports 16-sample antialiasing at $640 \times 512$ resolution or aliased rendering at $1280 \times 1024$ resolution, and a 4-board system supports 8-sample antialiasing at true HDTV ($1920 \times 1035$) resolution or 16-sample antialiasing at $1280 \times 1024$ resolution.

## 2.6 Display Hardware

Each of the 80 Image Engines on the raster memory board drives a single-bit, 50 MHz path to the display board, delivering video data at 500 MBytes per second. All 160 single-bit paths of a two raster memory board configuration are active, doubling the peak video data rate. The paths are time multiplexed by pairs of raster memory boards in the four board configuration. Ten crossbar ASICs on the display board assemble the 80 or 160 single-bit streams into individual color components or color indexes. Color components are then dithered from 12 bits to 10 bits and gamma corrected using $1024 \times 8$ lookup tables. The resulting 8-bit color components drive digital-to-analog converters and are output to the monitor. Color indexes are dereferenced in a 32K-location lookup table, supporting separate color lookup tables for each of up to 40 windows on the screen. Per-pixel display modes, such as the color index offset, are supported by a combination of Image Engine and display board hardware, driven by window ID bits stored in the framebuffer [1].

# 3 Features

This section provides additional information regarding the architecture's antialiasing, texture mapping, stereo, and clipping capabilities.

## 3.1 Antialiasing

The architecture supports two fundamentally different antialiasing techniques: alpha and multisample. Alpha antialiasing of points and lines is common to second generation architectures. Alpha antialiasing is implemented using subpixel and line-slope indexed tables to generate appropriate coverage values for points and lines, compensating for the subpixel position of line endpoints. Polygon coverage values are computed by counting the '1's in the full precision 8 × 8 coverage mask. The fragment alpha value is scaled by the fractional coverage value, which varies from 0.0, indicating no coverage, to 1.0, indicating complete coverage. If pixel blending is enabled, fragments are blended directly into the color buffer – no subpixel sample locations are accessed or required. Alpha antialiasing results in higher quality points and lines than does multisample antialiasing, because the resolution of the filter tables is greater than the 4 bit equivalent of the 16-sample mask. While alpha antialiased primitives should be rendered back-to-front or front-to-back (depending on the blend function being used) to generate a correct image, it is often possible to get an acceptable point or line image without such sorting. Alpha antialiased polygons, however, must be sorted near to far to get an acceptable image. Thus this technique is efficiently applied to polygons only in 2D scenes, such as instrument panels, where primitive ordering is fixed and a slight increase in quality is desired.

Multisample antialiasing has already been described. Its principal advantage over alpha antialiasing is its order invariance - points, lines, and polygons can be drawn into a multisample buffer in any order to produce the same final image. Two different mask generation techniques are supported in multisample mode, each with its own advantages and disadvantages. The default mask generation mode is called point sampled; the alternate mode is area sampled. A point sampled mask is geometrically accurate, meaning that each mask bit is set if and only if its subpixel location is within the perimeter of the point, line, or polygon outline. (Samples on the primitive's edge are included in exactly one of the two adjacent primitives.) Such masks insure the correctness of the final image, at the expense of its filtered quality. The final image is correct because all the samples that comprise it are geometrically valid - none having been taken outside their corresponding primitives. It is poorly sampled because the number of bits set in the mask may not closely correspond to the actual area of the pixel that is covered by the primitive, and the final filtering quality depends on this correspondence. Area sampling attempts to insure that the number of '1's in the sample mask is correct plus or minus 1/2 a sample, based on the actual coverage of pixel area by the primitive. (Figure 5.) In order to accomplish this, area sampled masks necessarily include samples that are outside the primitive outline, resulting in image artifacts such as polygon protrusions at silhouettes and T-junctions. Area sampled masks are implemented with a technique that is related to the one described by Andreas Schilling[10]. Point and area sampling can be selected by the application program on a per-primitive basis.

The desirable multisample property of order invariance is lost if alpha transparency and pixel blending are used. Alpha does sometimes carry significant information, usually as a result of the alpha channel in the texture application. For example, trees are often drawn as single polygons, using an alpha matte to express their shape. In order to handle alpha transparency without requiring pixel blending, the Image Engines have the ability to convert fragment alpha values to pseudo-random masks, which are then logically ANDed with the fragment's coverage mask. This method, while not geometrically accurate, provides usable antialiasing of texture mattes, and is order invariant.

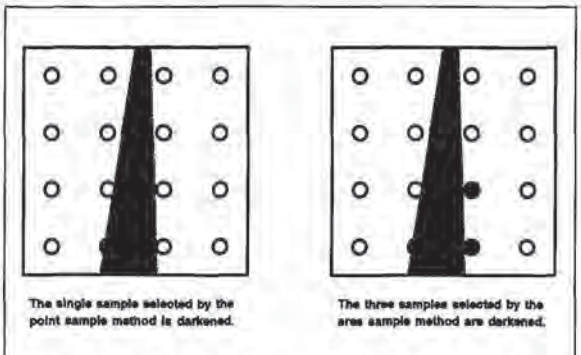



Figure 5. A narrow triangle intersected with a single, 16-sample pixel. The three samples selected by the area sample method accurately represent the fact that almost 20 percent of the pixel is covered by the triangle.

## 3.2 Texture Mapping

In addition to the 2-dimension texture maps described in the architecture section, 1- and 3-dimension maps are also supported. The eight million texel memory associated with each Fragment Generator stores 2D mipmapped images up to 1024 × 1024, and 3D non-mipmapped images up to 256 × 256 × 64. Thus 3D textures can be used to render volumetric images of substantial resolution, at rates up to 30 frames per second. The S, T, and R texture coordinates of each fragment are computed by interpolating S/W, T/W, R/W, and 1/W, then doing the correct divisions at each pixel, resulting in perspective-corrected mapping. Level-of-detail is also computed for each pixel, based on the worst-case of the four pixel-to-texel X and Y ratios.

Linear filtering of the nearest texels and mipmap levels is supported for 1D, 2D, and 3D textures, blending a total of 16 texel colors in the 3D mode. In the 2D case such linear filtering is commonly known as trilinear. Bicubic interpolation is supported for 2D, nonmipmapped textures, again blending 16 texels. There is no support for cubic filtering of 1D or 3D textures, or of any mipmapped textures. The default 16-bit texel size supports RGBA texels at 4-bits per component, RGB texels at 5-bits per component (6 bits for green), intensity-alpha texels at 8-bits per component, and intensity texels at 12-bits per component. 32-bit and 48-bit texels can be specified by the application with proportional loss of performance. The maximum RBGA texel resolution is 12-bits per component, equal to the maximum framebuffer color resolution.

Texture magnification can be done by extrapolation of mipmap levels, resulting in a sharpening of the highest resolution mipmap image, or the highest resolution image can be blended with a replicated 256 × 256 detail image, greatly increasing the apparent resolution of the texture without requiring excessive texture storage. Filter functions for RGB and for alpha can be specified separately

to improve the quality of texture mattes. Finally, texture memory can be loaded from the application processor's memory at the rate of 80 million 16-bit texels per second, allowing the application to treat texture memory as a managed cache of images.

### 3.3 Stereo in a Window

Image Engine memory can be configured with separate left and right color buffers for both the visible and nonvisible displayable color buffers, resulting in a total of four 48-bit color buffers per pixel. The display hardware alternately displays the left and right buffer contents of the visible buffers of all windows so configured, and drives a sync signal that can be used to control screen or head-mounted shutters. This stereo-in-a-window capability is both formally and practically compatible with the X protocol: formally because neither framebuffer dimensions nor pixel aspect ratio are changed when it is enabled or disabled, and practically because it allows monoscopic windows such as menus to be rendered and displayed correctly. To reduce eye fatigue, it is advisable to select a reduced-dimension framebuffer when the window system is initialized, allowing the frame display rate to be increased to 90+ Hz within the 140 MHz pixel limit of the display board.

### 3.4 Fast Clipping

RealityEngine polygon clipping is faster than that of our earlier designs for two fundamental reasons: it is implemented more efficiently, and it is required less often. Higher efficiency results from the MIMD Geometry Engine architecture. Because each of the engines executes an independent code sequence, and because each has significant input and output FIFOs, random clipping delays affect only a single engine and are averaged statistically across all the engines. Also, because each Geometry Engine comprises only a single processor, all of that engine's processing power can be devoted to the clipping process. SIMD architectures are less efficient because all processors are slowed when a single processor must clip a polygon. Pipelines of processors, and even MIMD arrangements of short pipelines, are less efficient because only a fraction of available processing power is available to the clipping process.

The requirement for clipping is reduced through a technique we call scissoring. Near and far plane clipping are done as usual, but the left, right, bottom, and top frustum edges are moved well away from the specified frustum, and all triangles that fall within the expanded frustum are projected to extended window coordinates. If culling is done by the application, almost no triangles will actually intersect the sides of the expanded frustum. Projected triangles that are not fully within the viewport are then scissored to match the edges of the viewport, eliminating the portions that are not within the viewport. The Pineda rasterization algorithm that is employed easily and efficiently handles the additional rectilinear edges that result, and no fragment generation performance is lost on scissored regions.

## 4 Design Alternatives

We think that the most interesting part of design is the alternatives considered, and the reasons for choices, rather than the details of the result. This section highlights some of these alternatives, in roughly decreasing order of significance.

### 4.1 Single-pass Antialiasing

Multi-pass accumulation buffer antialiasing using an accumulation buffer [3] is order invariant, and produces high-quality images in 10 to 20 passes. Further, a system that was fast enough to render 10 to 20 full scene images per frame would be a fantastic generator of aliased images. So why design a complex, multisample framebuffer to accomplish the same thing in one pass? The answer is that significantly more hardware would be required to implement a multi-pass machine with equivalent performance. This is true not only because the multi-pass machine must traverse and transform the object coordinates each pass, but in particular because texture mapping would also be performed for each pass. The component costs for traversal, transformation, parameter interpolation, and texture mapping constitute well over half of the multisample machine cost, and they are not replicated in the multisample architecture. A competing multi-pass architecture would have to replicate this hardware in some manner to achieve the required performance. Even the PixelFlow architecture[6], which avoids repeated traversal and transformation by buffering intermediate results, must still rasterize and texture map repeatedly.

### 4.2 Multisample Antialiasing

Multisample antialiasing is a rather brute-force technique for achieving order invariant single-pass antialiasing. We investigated alternative sorting buffer techniques derived from the A-buffer algorithm[2], hoping for higher filter quality and correct, single-pass transparency. These techniques were rejected for several reasons. First, sort buffers are inherently more complex than the multisample buffer and, with finite storage allocations per pixel, they may fail in undesirable ways. Second, any solution that is less exact than multisampling with point sampled mask generation will admit rendering errors such as polygon protrusions at silhouettes and T-junctions. Finally, the multisample algorithm matches the single-sample algorithm closely, allowing OpenGL pixel techniques such as stencil, alpha test, and depth test to work identically in single or multisample mode.

### 4.3 Immediate Resolution of Multisample Color

Our initial expectation was that rendering would update only the multisample color and depth values, requiring a subsequent resolution pass to reduce these values to the single color values for display. The computational expense of visiting all the pixels in the framebuffer is high, however, and the resolution pass damaged the software model, because OpenGL has no explicit scene demarcations. Immediate resolution became much more desirable when we realized that the single most common resolution case, where the fragment completely replaces the pixel's contents (i.e. the fragment mask is all ones and all depth comparisons pass) could be implemented by simply writing the fragment color to the color buffer, making no change to the 4, 8, or 16 subsample colors, and specially tagging the pixel. Only if the pixel is subsequently partially covered by a fragment is the color in the color buffer copied to the appropriate subsample color locations. This technique increases the performance in the typical rendering case and eliminates the need for a resolution pass.

### 4.4 Triangle Bus

All graphics architectures that implement parallel primitive processing and parallel fragment/pixel processing must also implement a crossbar somewhere between the geometry processors and the framebuffer[5]. While many of the issues concerning the placement of this crossbar are beyond the scope of this paper, we will mention some of the considerations that resulted in our Triangle Bus architecture. The RealityEngine Triangle Bus is a crossbar between the Geometry Engines and the Fragment Generators. Described in RealityEngine terms, architectures such as the Evans & Sutherland Freedom Series™ implement Geometry Engines and Fragment Generators in pairs, then switch the resulting fragments to the appropriate Image Engines using a fragment crossbar network. Such architectures have an advantage in fragment generation efficiency, due both to the improved locality of the fragments and to only one Fragment Generator being initialized per primitive. They suffer in comparison, however, for several reasons. First, transformation and fragment generation rates are linked, eliminating the possibility of tuning a machine for unbalanced rendering requirements by adding transformation or rasterization processors. Second, ultimate fill rate is limited by the fragment bandwidth, rather than the primitive bandwidth. For all but the smallest triangles the quantity of data generated by rasterization is much greater than that required for geometric specification, so this is a significant bottleneck. (See Appendix 2.) Finally, if primitives must be rendered in the order that they are specified, load balancing is almost impossible, because the number of fragments generated by a primitive varies by many orders of magnitude, and cannot be predicted prior to processor assignment. Both OpenGL and the core X renderer require such ordered rendering.

The PixelFlow[6] architecture also pairs Geometry Engines and Fragment Generators, but the equivalent of Image Engines and memory for a $128 \times 128$ pixel tile are also bundled with each Geometry/Fragment pair. The crossbar in this architecture is the compositing tree that funnels the contents of rasterized tiles to a final display buffer. Because the framebuffer associated with each processor is smaller than the final display buffer, the final image is assembled as a sequence of $128 \times 128$ logical tiles. Efficient operation is achieved only when each logical tile is rasterized once in its entirety, rather than being revisited when additional primitives are transformed. To insure that all primitives that correspond to a logical tile are known, all primitives must be transformed and sorted before rasterization can begin. This substantially increases the system's latency, and requires that the rendering software support the notion of frame demarcation. Neither the core X renderer nor OpenGL support this notion.

### 4.5 12-bit Color

Color component resolution was increased from the usual 8 bits to 12 bits for two reasons. First, the RealityEngine framebuffer stores color components in linear, rather than gamma-corrected, format. When 8-bit linear intensities are gamma corrected, single bit changes at low intensities are discernible, resulting in visible banding. The combination of 12-to-10 bit dithering and 10-bit gamma lookup tables used at display time eliminates visible banding. Second, it is intended that images be computed, rather than just stored, in the RealityEngine framebuffer. Volume rendering using 3D textures, for example, requires back-to-front composition of multiple slices through the data set. If the framebuffer resolution is just sufficient to display an acceptable image, repeated compositions will degrade the resolution visibly. The 12-bit components allow substantial framebuffer composition to take place before artifacts become visible.

Figure 6. A scene from a driving simulation running full-screen at 30 Hz.

Figure 7. A 12x magnified subregion of the scene in figure 6. The sky texture is properly sampled and the silhouettes of the ground and buildings against the sky are antialiased.

## Conclusion

The RealityEngine system was designed as a high-end workstation graphics accelerator with special abilities in image generation and image processing. This paper has described its architecture and capabilities in the realm of image generation: 20 to 60 Hz animations of full-screen, fully-textured, antialiased scenes. (Figures 6 and 7.) The image processing capabilities of the architecture have not been described at all; they include convolution, color space conversion, table lookup, histogramming, and a variety of warping and mapping operations using the texture mapping hardware. Future developments will investigate additional advanced rendering features, while continually reducing the cost of high-performance, high-quality graphics.

## Acknowledgments

It was a privilege to be a part of the team that created RealityEngine. While many team members made important contributions to the design, I especially acknowledge Mark Leather for developing the multisample antialiasing technique that was eventually adopted, and for designing a remarkable integrated circuit (the Image Engine) that implemented his design. Also, special thanks to Doug Voorhies, who read and carefully marked up several drafts of this paper, Finally, thanks to John Montrym, Dan Baum, Rolf van Widenfelt, and the anonymous reviewers for their clarifications and insights.

## Appendix 1: Measured Performance

The two most significant performance categories are transform rate: the number of primitives per second that can be processed by the Geometry Engines, and fill rate: the number of fragments per second that can be generated and merged into the framebuffer. Running in third-generation mode (lighting, smooth shading, depth buffering, texturing and multisample antialiasing) a 12 Geometry Engine system can process 1.5 million points, 0.7 million connected lines, and 1.0 million connected triangles per second. In second-generation mode (lighting, smooth shading, and depth buffering) the same system can process 2.0 million points, 1.3 million connected lines, and 1.2 million connected triangles per second. Measured third-generation fill rates for 2 and 4 raster board systems are 120 and 240 million fragments per second. Measured second-generation fill rates for 1, 2, and 4 raster board systems are 85, 180, and 360 million fragments per second. The third-generation fill rate numbers are somewhat dependent on rendering order, and are therefore chosen as averages over a range of actual performances.

## Appendix 2: Bandwidth and other Statistics

Triangle Bus, fragment transfer path, and Image Engine to framebuffer memory bandwidths are in roughly the ratios of 1:10:20. Specific numbers for the typical two raster board configuration are 240 Mbyte/sec on the Triangle Bus, 3,200 Mbyte/sec aggregate on the 160 Fragment Generator to Image Engine busses, and 6,400 Mbyte/sec aggregate on the 160 Image Engine to framebuffer connections.

Because the 6,400 Mbyte/sec framebuffer bandwidth is so much larger than the bandwidth required to refresh a monitor (roughly 800 Mbyte/sec at $1280 \times 1024 \times 76$Hz) we implement the framebuffer memory with dynamic RAM rather than video RAM, accepting the 12 percent fill rate degradation in favor of the lower cost of commodity memory. Geometry Engine memory and texture memory are also implemented with commodity, 16-bit data path dynamic RAM. Total dynamic memory in the maximally configured system is just over 1/2 Gigabyte.

## References

[1] AKELEY, KURT AND TOM JERMOLUK. High-Performance Polygon Rendering. In *Proceedings of SIGGRAPH '88* (August 1988), pp. 239–246.

[2] CARPENTER, LOREN. The A-buffer, An Antialiased Hidden Surface Method. In *Proceedings of SIGGRAPH '84* (July 1984), pp. 103–108.

[3] HAEBERLI, PAUL AND KURT AKELEY. The Accumulation Buffer: Hardware Support for High-Quality Rendering. In *Proceedings of SIGGRAPH '90* (August 1990), pp. 309–318.

[4] KIRK, DAVID AND DOUGLAS VOORHIES. The Rendering Architecture of the DN10000VS. In *Proceedings of SIGGRAPH '90* (August 1990), pp. 299–308.

[5] MOLNAR, STEVEN. *Image-Composition Architectures for Real-Time Image Generation*. University of North Carolina at Chapel Hill, Chapel Hill, NC, 1991.

[6] MOLNAR, STEVEN, JOHN EYLES AND JOHN POULTON. PixelFlow: High-Speed Rendering Using Image Composition. In *Proceedings of SIGGRAPH '92* (July 1992), pp. 231–240.

[7] NEIDER, JACQUELINE, MASON WOO AND TOM DAVIS. *OpenGL Programming Guide*. Addison Wesley, 1993.

[8] OPENGL ARCHITECTURE REVIEW BOARD. *OpenGL Reference Manual*. Addison Wesley, 1992.

[9] PINEDA, JUAN. A Parallel Algorithm for Polygon Rasterization. In *Proceedings of SIGGRAPH '88* (August 1988), pp. 17–20.

[10] SCHILLING, ANDREAS. A New Simple and Efficient Antialiasing with Subpixel Masks. In *Proceedings of SIGGRAPH '91* (July 1991), pp. 133–141.

[11] SILICON GRAPHICS, INC. *Iris 4DGT Technical Report*. Silicon Graphics, Inc., Mountain View, CA, 1988.

[12] SILICON GRAPHICS, INC. *Technical Report - Power Series*. Silicon Graphics, Inc., Mountain View, CA, 1990.

[13] WILLIAMS, LANCE. Pyramidal Parametrics. In *Proceedings of SIGGRAPH '83* (July 1983), pp. 1–11.

# VIEW – An Exploratory Molecular Visualization System with User-Definable Interaction Sequences

**Lawrence D. Bergman*, Jane S. Richardson†, David C. Richardson†, and Frederick P. Brooks, Jr.***

* GRIP Molecular Graphics Research Resource
Department of Computer Science
University of North Carolina at Chapel Hill

† Department of Biochemistry
Duke University

## ABSTRACT

VIEW is an exploratory visualization system for studying the structures of molecules. The system supports a high degree of complex user interaction with the image. Visualizations are constructed by selecting drawing tools from a library. Each tool uses parameters obtained from interactive selection of on-screen geometry by the user, and from a molecular database.

The system is based on a tight coupling of on-screen geometry with the underlying database. Using these links, tools can create true-scale drawing elements that are constrained to database values.

VIEW is highly extensible by the user or a paraprogrammer associated with the user. Drawing tools are written in a C-like programming language with constructs for managing databases, constructs for creating and altering geometry, as well as standard statements such as If-Else and For loops.

An event-definition mechanism allows the user to describe actions to be performed when keys are depressed or dials turned. In addition, the user is able to specify conditional events – actions that are to be taken whenever a user-defined condition becomes true. These conditions are automatically evaluated by the system as part of event processing. Such conditional events allow simple simulations to be readily programmed. Applications of conditional events have included animations of protein binding activity, and an interactive "flashlight" which highlights structures as a cursor is steered through a molecule.

The system includes a development environment complete with a WYSIWYG editor, an interactive debugger, and a set of innovative graphical debugging features.

* CB 3175, UNC, Chapel Hill NC 27599-3175.
(919) 962-1932 bergman@cs.unc.edu
(919) 962-1931 brooks@cs.unc.edu

† Department of Biochemistry, Duke University 27710
(919) 684-6010 jsr@suna.biochem.duke.edu
dcr@suna.biochem.duke.edu



VIEW has been installed for over a year in a protein crystallography laboratory at Duke University. Graduate students and faculty have used the system both for exploring molecular structures and for producing presentation graphics. These users have developed their own set of tools and made extensive use of the tool library. In January 1993, a beta-version of the software was released to a small set of laboratories in the US and Europe. It is now generally available.

**CR Categories and Subject Descriptors:** D.2.2 [**Software Engineering**]: Tools and Techniques – *Programmer workbench, Software libraries, User interfaces*; D.2.6 [**Software Engineering**]: Programming Environments – *Interactive*; D.3.2 [**Software Engineering**]: Language Classifications – *Design languages, Extensible languages, Specialized application languages*; I.3.6 [**Computer Graphics**]: Methodology and Techniques – *Interaction techniques*; I.3.8 [**Computer Graphics**]: Applications; J.3 [**Computer Applications**]: Life and Medical Sciences – *Biology*.

**Additional keywords:** scientific visualization, graphical debugging, molecular graphics, data-constrained sketching.

## MOTIVATION

**Visualization is powerful.** Over the past few years, scientific visualization has received a great deal of attention and is widely acknowledged as an important tool for the exploration of scientific data. Through visualization, a scientist assimilates large quantities of data, and may acquire new insights [1].

**The visualization design space is large.** Different representations of a dataset highlight and reveal different properties (see [2] for an excellent example). The number of possible exploratory visualizations of any dataset is limitless. Some of them will reveal or emphasize certain properties of the data; others will reveal or emphasize other properties; most will be uninformative. For this reason, a great deal of guidance by the scientist is usually required in constructing useful visualizations.

**Creating new geometric representations is important but difficult.** Graphical representations consist of a display of geometry with associated surface attributes such as color and texture. The shapes used to represent data entities and their relative position, size, and orientation tell us a great deal – they often contain most of the information in an image.

Designing new geometry is difficult. The user must specify the algorithm used to convert database information into geometric parameters. Using existing visualization systems, the user writes new code, usually in C, for each new geometric representation. Little

support is provided for this code development. The user usually works outside the boundary of the visualization system, employing a cumbersome code-compile-link-test-recode cycle.

**The problem.** The problem addressed in this work is: "How can we facilitate the design of new visual representations of scientific data, particularly new forms of geometry?"

**The design process.** A system for visualization should be based on the process by which a scientist or programmer designs a new visualization:

- Design usually starts with some sketching, done outside the confines of any visualization system – what we call the "paper napkin stage."
- Design is an iterative process – the user repeatedly tries new approaches and gradually refines her notion of what is needed to understand or emphasize the data. Most of the tries are unsuccessful.
- When a satisfactory sketch is achieved, a scale drawing reflecting the actual data is made.
- Visual feedback is crucial in guiding the design process. The user usually determines what she wants based on what she sees; she rarely knows exactly what she wants when she begins.
- New designs usually start with an existing design. Simple and aggregate elements from one design are often reused in a series of designs.

Examination of the design process tells us that the user needs a sketching facility, and an easy way to get from sketch to scale drawing. The ability to interact with a partially complete image, to try and discard a number of alternatives, is critical. Users must also be able to customize the tools or craft their own.

**Lack of design-process-based systems.** Existing visualization systems fall into two categories, neither of which have heretofore tried to support the design process as described. Application-specific software systems (such as commercial molecular modeling packages, geographic information systems, or flow visualization systems) often provide for interactive design, but the toolkit is fixed and often small. General-purpose visualization systems (such as AVS or Explorer) provide for user-specification of a visualization in a highly interactive fashion, but do not allow the user to directly interact with the visualization itself.

## THE VIEW SOLUTION

**A design-process-based exploratory system.** VIEW is a molecular visualization system designed to provide an exploratory environment. The goals of VIEW are to bring the "napkin" on-line, to support an iterative design process, to provide immediate visual feedback, to allow user-extension of the design tools, and to promote reuse of the design components.

**The data-drawing model.** Sketching for visualization is distinguished from that for most other forms of design in one important respect – the visualization represents an underlying database. We want geometric parameters of the visualization, primarily positions, to be specified using information from the database. Unlike a free-hand sketch, which can only portray the topology of a form, a database-driven sketch may be constrained to data values, providing a true-scale representation of the geometry.

The VIEW system supplies a method for interactive visualization design that we call *data-drawing*. With data-drawing, the user specifies database parameters by selecting geometric objects on-screen. These geometric objects serve as stand-ins for records in the database.

Design tools that incorporate the data-drawing method operate as follows:

1) The user picks a graphical element.
2) The tool retrieves data associated with that element and additional related data.
3) The tool creates new geometry based on the database information and associates that information with the geometry. This new geometry is in turn available as a visual template for future data-drawing.

With the language that specifies drawing tools, the user associates database records with on-screen geometry and retrieves this information from picks. This uniform philosophy of tool design and use incorporates two desirable features:

- As the user constructs geometries, the new forms are available for data-drawing.
- The tool-user may select any form of geometry that represents the desired database element. She is not restricted to a small set of special representations.

**Use of VIEW.** A user of the VIEW system starts a session with a simple representation of a molecule. An initial tool creates geometry from molecular data such as Brookhaven Protein Databank atomic coordinates. Often vectors are chosen to represent atomic bonds. Using a variety of drawing tools from a library provided, the user sketches in additional geometry. For example, a user may sketch: individual amino acids with the bonds represented as small cylinders, larger cylinders that represent the axes of helices within a protein, a spline-like representation of a portion of the backbone, or any of a number of other representations.

**Interaction with the image is crucial.** We recognize that each user has preferred interaction styles. For this reason, the user may customize interaction sequences. The drawing tool language provides a facility known as *interactive events-monitors* for defining actions that are to be performed based on mouse movement, dial movement, and key presses.

Interactive sequences that have been coded to date include: moving a small molecule with the mouse, changing sphere and cylinder radii using a dial, moving atoms using dials while maintaining bond connections, and triggering actions on key depressions.

**Geometry and data closely linked.** Central to the data-drawing model is a tight coupling of on-screen geometry with the underlying data. This allows on-screen picks to return database records directly to the drawing routines.

**Design of new drawing tools.** The VIEW user can code new drawing tools which become members of the library. This ability to design new tools is an important feature in an impromptu visualization system.

**Tool development environment.** VIEW supports the user in extending the toolkit by providing a development environment that includes a Macintosh-like text editor and a visual debugger that interacts with the on-screen image.

## AN EXAMPLE

Most of the visualizations produced using VIEW have displayed protein molecules. Proteins consist of a linear chain of *amino acid residues* which fold into a few well-defined 3-D structures such as *beta sheets* and *alpha helices*. These in turn form larger motifs such as *beta barrels*. A protein has a *mainchain* consisting of carbon, oxygen, and nitrogen atoms. Extending from the mainchain are *sidechains*. Each amino acid type, of which there are approximately twenty, has a distinctive sidechain. Hydrogen bond connections between non-sequential amino acids define the *topology* of the protein.

Jane Richardson produced the visualization of the protein Concanavalin A shown in Figure 1 using VIEW drawing tools. A variation of this image appeared in *Biophysical Journal* [3]. The image shows the orientation of a phenylalanine amino acid and two possible but less favorable orientations.

The steps in constructing the visualization were:

1) Richardson selected a tool that creates initial geometry starting with atomic coordinates. The tool produces vectors connecting just the alpha carbons of adjacent amino acids (Figure 2a). This representation gives a clear global view of the structure.

2) Using a mainchain drawing tool, she sketched in atomic-level detail for three strands of the chain. She specified starting and ending points by picking atom positions in the original representation. On each pick, the tool drew a small red sphere to mark the selection. After both ends were selected, the tool drew the connecting main chain at the atomic level automatically, using atom coordinates fetched from the molecular data. After drawing the main chain, the tool removed the marker spheres. Figure 2b shows the drawing after specification of the second strand, just before the system removed the red markers. Only one tool selection and four datapoint selections were required to produce this detailed scale drawing.

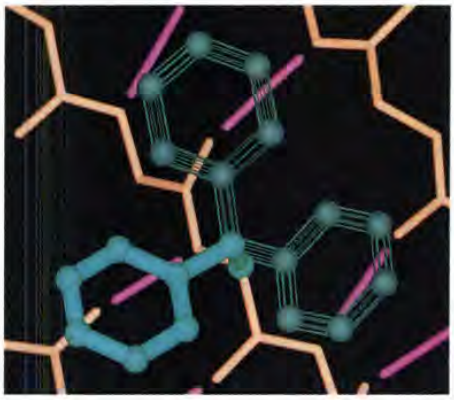


Figure 1: A VIEW visualization

3) Using a line drawing tool, she sketched in hydrogen bonds that couple the strands together. The particular tool employed knows nothing about hydrogen bonds; Richardson selected the termini of each bond. We could have written a new tool to automatically draw in all hydrogen bonds for the molecule. Since we wanted only a small, selected set, manual specification of each bond seemed reasonable.

   The line drawing tool bases each drawing operation on two atom selections. This time, the selections were made using the geometry produced in step 2; display of the original representation was toggled off. VIEW users often turn off individual groups of geometry by pressing virtual buttons in the interface. Users frequently switch between sparse global views and detailed local views.

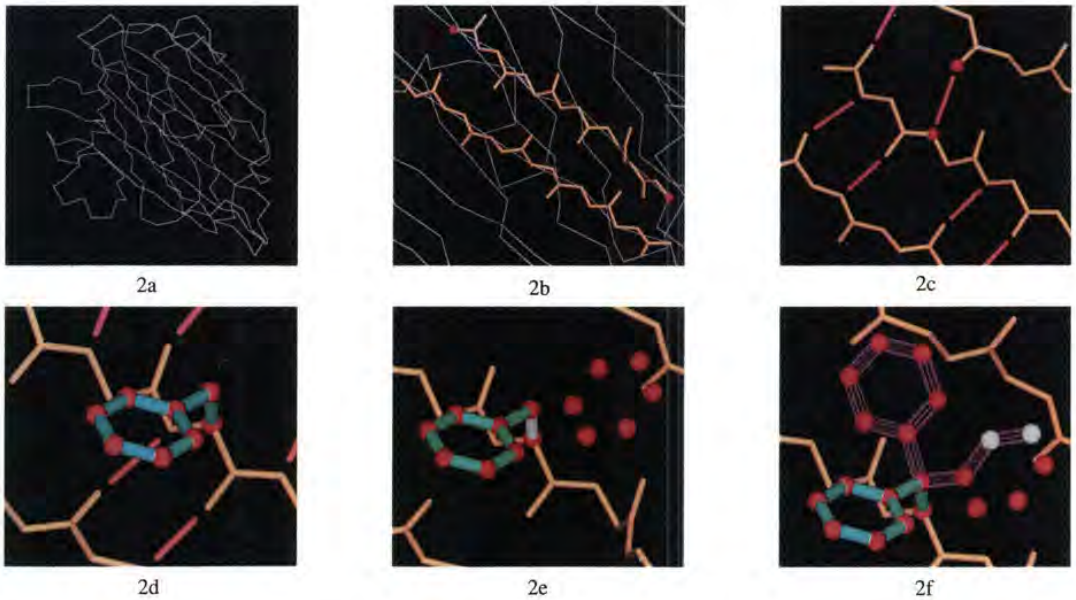



Figure 2. Construction of a visualization using VIEW drawing tools

With the line drawing tool, the user may lengthen or shorten lines that connect the centers of selected atoms. By pressing the "l" key, Richardson triggered an interactive event for defining the length scaling. The event popped up a query window requesting a scaling factor. She specified .65 before proceeding with the drawing shown in Figure 2c.

4) Next, she sketched a single sidechain using a tool which draws an entire sidechain based on one atom selection and information from the molecular database. Once the bonds of the sidechain were drawn, Richardson used a marker-sphere tool to mark the sidechain atoms. Figure 2d shows the result of a single selection with the sidechain tool and eight selections with the marker-sphere tool.

5) In order to produce the two rotated positions of the sidechain, Richardson created a duplicate of the marker spheres using a group duplication tool. VIEW places the geometry created by each drawing tool in a separate *geometry group* labeled with the name of the tool that produced it. These groups may be individually manipulated by other tools. The duplication tool requested that she select any element of geometry from the group to be duplicated, and then queried her for a name for the new group. This specification of group by identifying a member is a common theme in VIEW drawing tools.

6) Once the group had been duplicated, she used a rotation tool to rotate the duplicate into position. The rotation tool requests selection of a rotation axis, followed by requests for selection of one or more geometry groups to be rotated. Rotation may then be performed with a dial or by key presses. In this case, Richardson wanted to rotate the group by a precise amount. An event triggered by pressing the "d" key allowed her to type in the desired angle (120 degrees). She applied the rotation by pressing the "r" key. Figure 2e shows the rotated markers. The rotation axis is highlighted in white.

7) Steps 4 and 5 were repeated to produce a second duplicate rotated to 240 degrees.

8) In order to generate the open framework of lines connecting markers at the rotated positions, Bergman coded a new drawing tool that connects a sequence of selected positions with wireframe cylinders. The tool was created by merging and modifying two prior tools. The first of these tools connected a sequence of positions with solid cylinders using the system cylinder primitive. The other produced a tessellated cylinder not using the system cylinder primitive. The new tool was developed in under a half-hour, including testing. Figure 2f shows the open cylinders being drawn; the last two selected positions are highlighted white.

9) Richardson used a group recoloring tool to finalize colors in the image, shown in Figure 1. The sequence of operations described (excluding the tool development in step 8) can be carried out in about fifteen minutes. In actuality, Richardson spent a couple of hours producing the visualization. A large amount of trial-and-error is required to design a useful image. She tried a score of possibilities before settling on the above result, including: changing colors, radii, lengths, and number of facets for the wireframe cylinders.

## SYSTEM DESCRIPTION

The VIEW system (Figure 3) is written in C++ and runs on SGI 4D, Indigo, and Crimson workstations. Three-dimensional manipulation of geometry is performed using a mouse-based virtual trackball or a dialbox.

**Drawing tools.** The toolkit supplied with the VIEW system contains

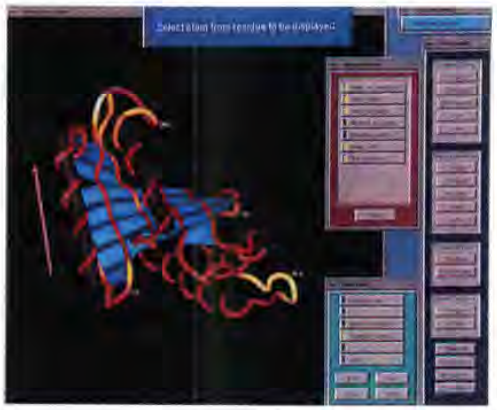


Figure 3: The VIEW system

about fifty interactive drawing tools. Several tools are provided to generate initial representations of a database; the majority of the tools are used for click-and-draw geometry generation or modification. Drawing operations are reversible using a multi-level undo feature.

**Tool language.** Drawing tools are specified in a C-like language with most of the standard C datatypes and control structures. Additionally, the language includes geometric and database datatypes, constructs for database access and modification, and constructs for selecting and manipulating individual geometric elements and groups of geometry. A key feature is the ability to associate database records with individual geometric primitives. Picking a geometric primitive will retrieve its associated data properties.

The drawing tool language was designed for ease of use. Ability to prototype quickly was given higher priority than runtime speed. For this reason the language is dynamically typed with no type declaration statements. Objects are sized dynamically – no size declarations are given for arrays, sets, geometry groups or databases. Scope rules are very simple. All variables are global within a routine, and they are also global to all event-monitors defined within the routine. Variables are only available outside a routine if passed as subroutine parameters.

The language is interpreted, allowing changes to a tool to be quickly retried. The tools in the library are all coded in the tool language, not in C++. The prompts, highlighting, and final geometry created are specified by tool language statements. The user can readily change the interaction sequence and any of the intermediate or final representations produced by these tools.

Several considerations led to development of a new, interpreted language:

1) We wanted a procedural language similar to C or FORTRAN. Scientists are understandably reluctant to invest in learning new programming styles. This consideration ruled out popular interpreted languages such as LISP and Smalltalk.

2) We wanted to supply special syntax to simplify expression of certain commonly used constructs, particularly database and geometry access. This could have been accomplished by supplying a subroutine library for a language such as C. However, code developed with such a library will be less concise and less readable than if special constructs are available.

3) To simplify coding, we wanted to avoid type statements and size declarations. We also wanted to avoid certain constructs such as pointers that provide flexibility at the expense of code comprehensibility.

4) We wanted to support interactive event definition.

5) We wanted interpretation of the language to be closely integrated with debugging functions, particularly graphical debugging facilities.

**Close connection between geometry and databases.** One of the characteristics that most distinguishes VIEW from existing general-purpose visualization systems, is the intimate connection between on-screen geometry and an underlying database that the geometry represents. This connection is fundamental to on-screen sketching in a representation of the database – the basic notion of the data-drawing model.

VIEW drawing tools typically associate atom or bond records with each element of geometry created. This association allows other tools to use these elements as visual stand-ins for database entries; the user may select atoms or bonds by clicking on geometry that represents them. Although this is a tried-and-true technique, VIEW is unique in allowing the user to specify the connections; she is not tied down to a system-defined schema. The following code fragment presents an example of establishing links between a geometric object and database records:

```
cyl = CYLINDER (pnt1, pnt2, radius);          (1)
cyl.DB_PTR = atom_rec;                        (2)
cyl.DB_PTR = bond_rec;                        (3)
```

Statement (1) creates a cylinder with the variable name *cyl*. Statement (2) establishes a connection between the cylinder and the atom record stored in *atom_rec*. Statement (3) assigns an additional database pointer to the cylinder, this time to a bond record.

Another tool can access the database information associated with this on-screen geometry as follows:

```
SELECT (item, "Select an object");            (1)
selected_atom_rec = item.atom;                (2)
```

Statement (1) is a pick. The user is told to select a geometric object on-screen using the mouse. The selected object will be returned in the variable *item*. Statement (2) specifies that the atom record pointed to by that object is to be assigned to the variable *selected_atom_rec*. If the geometric object selected happens to be the cylinder created in the previous example, the value of *selected_atom_rec* will be the record contained in *atom_rec*.

**Databases.** VIEW databases are stored in a non-application specific format. In fact, the only portion of the VIEW system that is specific to molecular visualization is certain drawing tools in the library. We convert molecular data from Brookhaven Protein Databank format to the more generic VIEW format using a filter run outside of VIEW.

Databases are stored in ASCII files, each consisting of one or more named *subsets*. Our molecular databases have two subsets – an atom subset, and a bond subset. A subset consists of a header which describes the record format, followed by a sequence of records. All records in a subset have the same number and ordering of fields. Record fields may be integers, floating point numbers, or strings. Each record contains a single integer- or string-valued key field used for key-access.

Record access from a database requires naming the subset to be accessed and the retrieval key. For example:

```
rec = dbase.atom(num);
```

will retrieve the record that has atom number *num* from the *atom* subset of the database *dbase*. Fields can be retrieved from a record by naming the field. For example:

```
type = rec.atom_type;
```

will retrieve the *atom_type* field from the record *rec*. These forms may be combined. For example:

```
type = dbase.atom(num).atom_type;
```

A special iterator, FOREACH, allows iteration through a subset's records in the order in which they are stored in the file (FOREACH is also used for iterating on arrays, sets, and geometry groups). The NEXT_RECORD statement retrieves the record following a given record, allowing manual control of record access. Similarly a PREV_RECORD statement allows backward movement through a subset.

The VIEW drawing tool language allows the user a great deal of flexibility in modifying databases including: 1) changing fields, 2) adding or deleting fields or records, and 3) writing and reading databases to and from files. Additionally, the user may define her own database formats.

**Geometry groups.** The drawing tool language provides constructors for geometric objects including spheres, triangles, lines, cylinders, and text. Geometric objects, created by drawing tools, or read from files are stored in *geometry groups*. Groups allow named access to related sets of geometric objects. The system provides a mechanism for controlling which groups are displayed on-screen. Other management functions are available including: 1) removing groups from the system, 2) writing groups to file, and 3) renaming groups.

Each geometric object is contained in one and only one geometry group. There is no nesting of groups. These properties were dictated by two simple design rules.

1) We wanted all geometric objects to be contained in a named geometry group. This ensures that display of any object may be turned on and off using a "group display" function in the interface.

2) We wanted to be able to access a group through a geometric object. This allows the user to specify a geometry group by selecting a member of that group. The "geometry group by example" model focuses the user's attention on on-screen geometry, not on interface buttons or menus. For this reason, we wanted a tool language construct that would query an object: "what group are you in?" To make this construct simple, both syntactically and semantically, we restricted objects to membership in a single group.

The contents of each geometry group is thus distinct. This property is highly desirable. The semantics of group display and group removal are thereby simple and intuitive. If groups are permitted to overlap, these semantics become more involved and may be counterintuitive.

By default, each drawing tool adds geometry to a group that has the name of the tool. If no such group exists, the system automatically creates it the first time the tool generates display geometry. In addition, the tool writer may define groups with other names in which geometry is to be placed. Tools for duplicating geometry groups and merging the contents of geometry groups are provided in the tool library.

**Interface operations.** Another design criterion for the language was that tools be able to specify any operation that can be performed from the user interface. Statements are available to provide tool control over user interface functions such as toggling the display of geometry groups, removing groups, reading and writing databases from and to files, etc.

**Event-monitor definition.** The tool language provides a mechanism which allows the tool creator to specify blocks of code that are to be executed when specified keyboard keys are pressed or when dials are rotated. These definitions, known as *interactive event-monitors*, consist of two portions – a *monitor*, which watches the specified device; and an *event body*, which is the code to be executed when the monitor is triggered. A tool may define a suite of event-monitors on different devices which communicate through a common symbol table; the tool builder can readily design a sophisticated interactive interface to her tools.

The following example specifies a dynamic radius-changing tool.

```
SELECT (obj, "Select a geometric object to be
                changed");                          (1)

EVENT ("change_radius"; ON DIAL 7)                  (2)
{
    obj.RADIUS = obj.RADIUS *
                 (1 + DIALRATE/50 );                (3)
    IF (obj.RADIUS < 0.01)  obj.RADIUS = 0.01;      (4)
    REDRAW();                                       (5)
}
```

Line (1) is a pick specification. Line (2) defines an event named "change_radius" which will be executed whenever dial 7 is rotated. Line (3) modifies the selected object's radius, using a system-defined variable, DIALRATE, which contains the angular change in dial position, positive for clockwise movement, negative for counter-clockwise. Line (4) ensures that the radius remains positive, and line (5) redraws the screen. With this tool, the user selects an object and then rotates dial 7 to increase or decrease its radius. The radius will alter smoothly as the dial is rotated, because the system continually reexecutes the event body as long as the dial state is changing.

Although only one drawing tool may execute at a time, the event-monitors that it defines persist. Thus, a whole set of event-monitors defined by different tools can be active simultaneously, each monitoring a different device. For example, the rotation tool described above might be used in conjunction with a translation tool that translates geometry along a selected axis using a different dial. The two dials may be used to rotate and translate "concurrently." A panel in the interface gives a summary of all currently active event-monitors.

**Conditional event-monitors.** In addition to event-monitors that are bound to dials and keys, the system supports *conditional event-monitors*. The monitor is a conditional expression; the event is triggered when the monitor expression is True. The system event manager stores a parse tree for each of these expressions, and the interpreter evaluates them on each iteration of the inner event loop. When any of the conditionals is True, the event body is executed. Thus, the tool builder defines actions to be taken based on certain conditions without coding a polling loop. The conditional evaluation slows the system, but it still responds at interactive rates even when several event-monitors are defined. A simple conditional event-monitor is shown in the following example. This code segment will turn a predefined object red when a probe is within a specified distance.

```
EVENT ("highlight_dist"; DIST(probe.CENTER,
                obj.CENTER) < 5.0)                  (1)
{
    obj.COLOR = COLOR(255,0,0);                     (2)
    REDRAW();
    STOP_EVENT ("highlight_dist");                  (3)
}
```

Statement (1) is a conditional event-monitor definition. This statement creates an event, "highlight_dist", which will be triggered whenever the distance between the center of the object stored in *probe* and the object stored in *obj* is less than 5 units (*probe* and *obj* would be defined elsewhere in the tool, and additional event-monitors provided for moving *probe* through the scene). Line (2) sets the color of *obj* to red, and line (3) deactivates the monitor so that the color change is only applied once.

**Spatial search.** Frequently we wish a tool to simultaneously monitor a group of 3-D points. When any point is near a specified location, an event is to be triggered. This function occurs so often that it calls for its own underlying mechanism. Simultaneous monitoring is implemented by means of a spatial search function. The function takes as input an array of points to be checked, a probe location and a radius. Any points from the check list that fall within the search radius are placed in an output array. If the output array contains any points, the function returns True. This mechanism allows the tool to trigger events at any of a large number of positions.

**Programmable undo.** The language allows the developer to define the scope of the system **undo** function. The keyword UNDOABLE may be supplied as an argument to either the pick function or in an event-monitor header. This keyword indicates that a checkpoint is to be created prior to execution of the statement. Whenever **undo** is clicked, the system restores the state of the latest checkpoint. The undo stack stores up to twenty-five checkpoints, allowing the user to backup through a number of drawing operations.

**Tool development environment.** The VIEW development environment is modeled on that of Smalltalk-80. Code may be modified and executed from editors or pop-up debuggers, allowing a rapid code-test-recode cycle. The debugger supports many features of traditional interactive debuggers including setting breakpoints, step, next, and print.

**Graphical debugging.** Several *graphical debugging* features are provided that go beyond those provided in Smalltalk or interactive debuggers such as dbx. Using a *construction* facility, the developer automatically views graphical representations of intermediate construction points and lines as they are created by the code. This graphical auto-print makes it easy to follow the progress of algorithms that construct geometry. The *display* function within the debugger highlights the representation corresponding to any selected geometric variable.

*Graphical breakpoints* are also available. These are similar to the conditional breakpoints provided in interactive debuggers such as dbx. Rather than providing an expression that must be True for execution to pause, however, the user selects a graphical entity, whose display is the condition on which execution is to pause. With this facility, the user selects an object at which the algorithm is to stop. The system will pause on reexecution, allowing her to display variable values (graphically or textually) or manually control the execution.

## SAMPLE APPLICATIONS

**Interactive superposition.** We have developed several interactive exploratory applications using VIEW. Kim Gernert, a biochemist at Duke University, has been studying the geometry of close atomic contacts within protein molecules. She wished to superimpose similar structures from a number of proteins, and then measure geometric parameters from each.

Bergman and Gernert prototyped the superpositioning procedure using a sequence of tools. We began by sketching an axis in each of two structures to be superimposed. We then selected a tool that computes a transformation to superimpose two selected axes. The

tool applies the transformation to the geometry of the structure to be superimposed. Once the two structures were oriented on a common axis, the rotation tool described previously was used to rotate one of the structures around that axis. A dial controlled the rotation. Using a translation tool, we moved the structure along the common axis under control of a different dial. The rotation and translation tools allowed us to manually superimpose the structures. Figure 4 shows the completed superposition.

Figure 4: Superposition of alpha helix turns using interactive tools

**Interactive structure highlighting.** Using the conditional event mechanism with spatial search, we have constructed a "flashlight" tool for exploring proteins. Using the mouse, we steered a probe sphere through a skeletal representation of a protein. As the probe comes near portions of the molecule, more detailed representations are generated. The flashlight has several "lenses" selected by key toggles. With one lens, amino acid sidechains near the probe are dynamically drawn in. With another lens, the mainchain is highlighted. Yet another lens displays pinwheel icons representing close contacts between neighboring atoms. Figure 5 shows a protein with sections of the molecule traced by the flashlight.

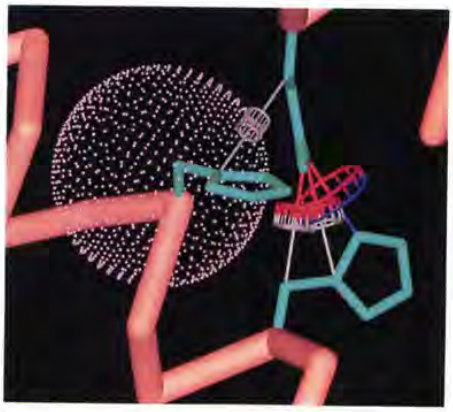


Figure 5: Flashlight tool for highlighting protein structure

**Algorithm construction and visualization.** VIEW has proven to be quite useful for visualizing geometric algorithms. We used a set of 3-D "ruler-and-compass" construction tools to develop a parameterization of the helix contact geometry discussed above. Using four tools: "project a point onto a line", "construct a plane normal to a line through a given point", "project a point onto a plane", and "connect two points", we were able to construct the geometry in Figure 6. The figure displays in a plane the angles that we decided to use for the study.

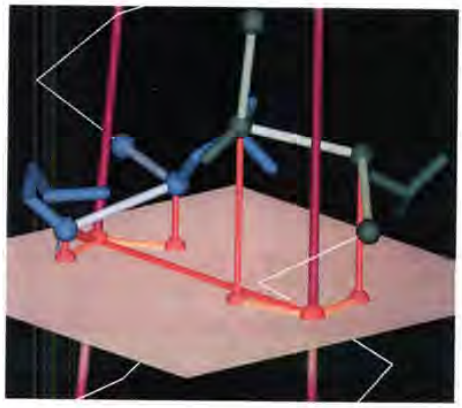


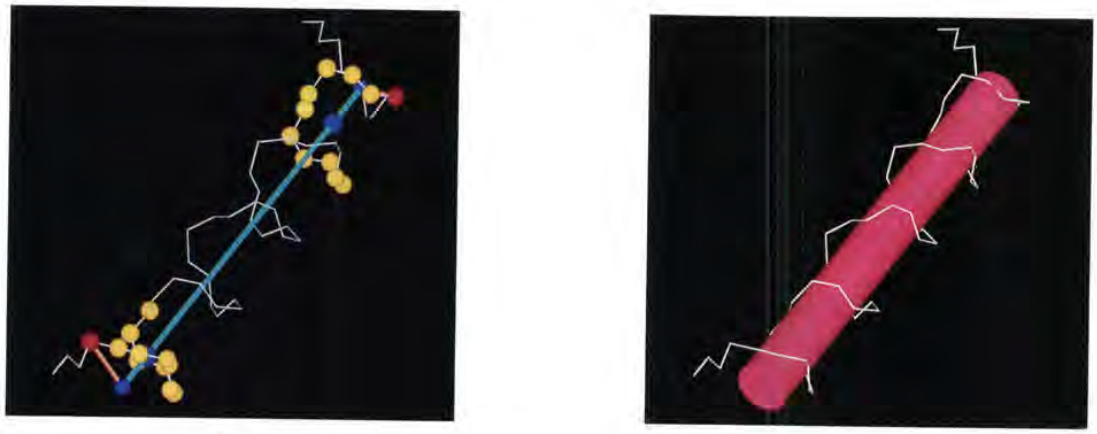
Figure 6: Ruler-and-compass construction of an algorithm

VIEW has also allowed us to visualize the workings of existing algorithms. Figure 7a shows the technique used for constructing axes of alpha helices in proteins (shown in Figure 7b). The red spheres mark user-selected atoms. The yellow spheres mark positions obtained from the database. The blue spheres mark positions computed by the tool. Display statements to generate the construction spheres and cylinders that illustrate the algorithm were added to the already working tool in about 15 minutes.

**Interactive topology tracing.** Several educational applications have been constructed using VIEW. Figure 8 shows a tool that is used to interactively outline the topology of protein backbone. With a sequence of events on keyboard keys, the user guides a cursor along the backbone, indicating where segments of interest begin and end (Figure 8a). As each segment is identified, a simplified representation replaces the backbone. Once all segments are identified, an event is available to specify ordering and orientation of the segments (Figure 8b). Finally, the tool flattens the connected segments into a map of the chain topology (Figure 8c).

**Interactive simulation of binding activity.** Another educational application is simulation of the binding activity of enzymes. Several interactive applications have been constructed that allow a student to steer a small molecule into a protein's active site, triggering an animated conformational change.

Binding of a dipeptide in the active site of the protein carboxypepsidase (Figure 9) requires that the dipeptide be close to the ideal position, and oriented properly. The simulation is implemented using a conditional event-monitor that checks the distance between the dipeptide and the binding site and also evaluates orientation by checking two dot products. When the distance and both dot products are within specified limits, the event is triggered. The dipeptide is rotated and translated into the exact binding alignment, while the



7a 7b

Figure 7: Display of helix axis tool algorithm

conformational change is animated by displaying a series of precomputed frames. The dipeptide may be driven in and out of the binding sites with alternate conformational changes.

## RELATED WORK

The vision driving development of the VIEW system was first described by Brooks [4].

General purpose visualization systems (such as AVS [5], ApE [6], Explorer [7], Data explorer [8]) allow the user to configure their own applications using a data-flow programming model. These systems tend towards a batch visualization pipeline – the user's ability to interact with the image (beyond viewing manipulations) is limited. VIEW extends the capability of these systems by adding a high degree of interaction with the image; allowing the user to direct the visualization process on-the-fly. VIEW also goes beyond these systems by supporting new module development within the confines of the system (although IRIS Explorer has recently introduced several embedded languages [7]). The use of an interpreted language, with built-in graphical debugging, greatly facilitates tool development.

Several visualization programming languages and systems have been developed in recent years. Palmer's pdbq language [9] provides support for visualization of molecular structures. Hultquist's LISP-based system for flow-visualization [10] allows rapid prototyping of new algorithms. The VIEW language extends these systems by adding interaction with the image. With Hibbert's system for developing algorithms to process meteorological data [11], the user steps through an algorithm, choosing a variety of display representations for intermediate values. The selection of locations to be examined is performed on-screen. VIEW provides a similar facility, with the addition of user-specified geometries and interaction sequences. Gramps [12] is a general purpose graphics language which has been extended for molecular modeling [13]. VIEW goes beyond Gramps by providing a general-purpose programming language and scriptable interaction sequences.

The MAGE system, developed by D.C. and J.S. Richardson, pioneers a new concept in scientific visualization [14]. Authors in the journal *Protein Science* publish not only their visual images, but also the associated 3-D display lists on a diskette. The diskette also contains the MAGE software for the Macintosh and the PC. Animated visualizations are pre-scripted using a scripting language available to any reader. In addition to viewing the animation, the reader may use a fixed set of database query and visualization tools to explore the images.

## WHAT'S NEW

In summary, the VIEW system goes beyond previous work by providing:

- data-drawing

8a 8b 8c

Figure 8. Steps in tracing chain topology. The final figure shows a classic Greek-key barrel motif.

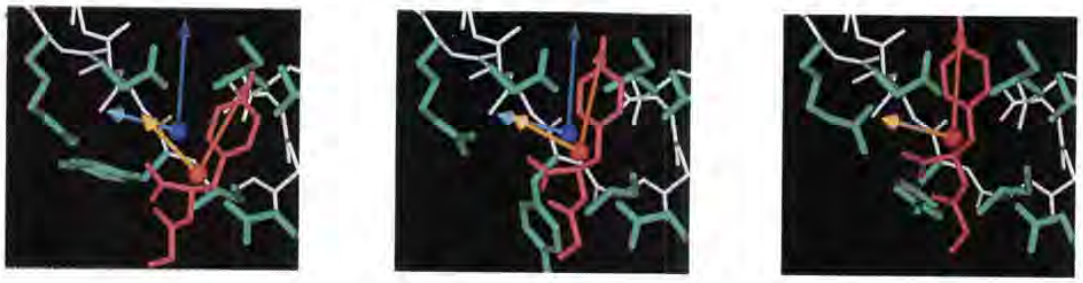


Figure 9. Interactive simulation of dipeptide binding

- image-based interaction in which the user changes the visualization by touching parts of it.
- user-customizable interaction sequences.
- conditional event-monitors that allow parts of the visualization to respond automatically to user actions on other parts.
- multiple event-monitors based on spatial search that allow simultaneous evaluation of potential actions at a large number of 3-D locations.
- graphical debugging in which the image is treated as a trace of a routine's execution state.

## APPLICATION AND SYSTEM EXTENSIONS

The VIEW system as is could be readily applied to other datasets that are naturally represented by discrete geometric structures. Airframes or finite-element models are good examples. Users commonly superimpose analytical artifacts such as grids, region boundaries, and isovalue surfaces into scientific databases. These artifacts provide structured geometry to which the VIEW approach could be applied.

Volume data presents difficulties for current visualization systems; the sheer volume of information precludes real-time specification of parameters. An interactive exploratory system might provide a solution. We can imagine starting with a skeletal representation of a volume dataset, perhaps a sparse cloud of points. A flashlight tool could be used to produce a higher quality rendering for user-selected portions of the data set.

Selection of regions of interest is a common task in studying a dataset. A scripting capability would allow users to build tools to display anomalies in the data, search for and highlight extrema, and construct a variety of other interactive filters to limit the amount of visual information displayed. A language tailored to volume data, containing commonly used datatypes and operations, should provide powerful exploratory capabilities.

The ability to script interaction demonstrated in VIEW might prove extremely useful in dealing with fluid-flow data and other time-varying datasets. Interactive placement of trace particles and other types of probes is already common in systems for studying flow fields. The conformational animations produced using VIEW show simple cases of scripting interaction with time-varying datasets. Additional language support for handling time-varying data would greatly enhance this capability. Scientists would be able to readily tailor the visualization process to accommodate a variety of datasets and interaction styles.

A useful extension of VIEW would be an ability to sketch geometry into an animation. After a series of animation frames have been created, the user might wish to sketch new geometry into a single frame using any of the VIEW drawing tools. The tools would automatically add the same geometric forms to all frames, with geometric parameters properly updated to account for between-frame positional changes.

The conditional event-monitor mechanism has proved to be a powerful tool for specifying interaction sequences. The shortcoming of the technique, however, is that evaluation of the conditionals slows the overall response of the system. A natural solution would be to implement a multi-processor architecture for the system. Each processor would be assigned evaluation of one or more monitors, with access to the required symbol tables and parse trees through shared memory. Any processor that detects a monitor trigger would set a global flag, with an associated record indicating the event to be executed. The main control process would simply check the flag as part of its polling loop and when appropriate, would initiate event processing.

Conditional event-monitors are also limited by the restriction of the header to a single relational expression. We would like to specify an arbitrary code block to be evaluated continually, with some portion of that block serving as the conditional expression. We have not taken that approach in the current implementation because read-only relational expressions are easier to deal with than general blocks of code. In the latter case, we would need to implement a critical section mechanism, ensuring that only a single event-monitor or tool attempts to update the symbol table at a time. Doing so would make the conditional event-monitor mechanism much more powerful.

## SYSTEM AVAILABILITY

VIEW is available for public use via anonymous ftp. The ftp site is ftp.cs.unc.edu (152.2.128.159). Executables, data files, and documentation are located in the pub/VIEW directory. More extensive documentation is available from the UNC Department of Computer Science.

## ACKNOWLEDGMENTS

This work is supported by the Biotechnology Research Program, National Center for Research Resources, NIH, grant number RR02170. Our thanks to the many graduate research assistants who worked on previous versions of the VIEW system, and to Daniel Aliaga for assistance in implementing the current version. Tom Palmer and Dave Bock offered many useful suggestions on the applicability of VIEW concepts to other datatypes. Thanks to Amitabh Varshney, and Mike Bajura for assistance in preparing the manuscript, and to Laura Bollinger for careful editing. We especially thank collaborating biochemist Kim Gernert of Duke University, for numerous contributions to this research.

## REFERENCES


[1] B.H. McCormick, T.A. DeFanti, and M.D. Brown, eds., "Visualization in Scientific Computing," *Computer Graphics*, Vol. 21, No. 6, Nov. 1987.

[2] M. Pique, J.S. Richardson, and F.P. Brooks, Jr., "What Does a Protein Look Like?" Invited videotape presented at 1982 SIGGRAPH Conference, July 1982.

[3] J.S. Richardson *et al*, "Looking at Proteins: Representations, Folding, Packing, and Design," *Biophys. J.*, Vol. 63, Nov. 1992, pp. 1186-1209.

[4] Bergman, *et al*, "VIEW – Visualization Impromptu Evaluation Workbench," abstract in *J. Mol. Graphics*, Vol. 6, Dec. 1988, pp. 223.

[5] C. Upson *et al*, "The Application Visualization System: A Computational Environment for Scientific Visualization," *IEEE Computer Graphics & Applications*, Vol. 9, No. 4, July 1989, pp. 30-42.

[6] D.S. Dyer, "A Dataflow Toolkit for Visualization," *IEEE Computer Graphics & Applications*, Vol. 10, No. 4, July 1990, pp. 60-69.

[7] *IRIS Explorer (TM) User's Guide*, Document Number 007-1371-010, Silicon Graphics, Inc., Mountain View, CA, Jan. 1992.

[8] B. Lucas *et al*, "An Architecture for a Scientific Visualization System," *Proc. Visualization '92* (Oct. 1992), pp. 107-114.

[9] T.C. Palmer, "A Language for Molecular Visualization," *IEEE Computer Graphics & Applications*, Vol. 12, No. 3, May 1992, pp. 23-32.

[10] J.P. Hultquist and E.L. Raible, "SuperGlue: A Programming Environment for Scientific Visualization," *Proc. Visualization '92* (Oct. 1992), pp. 243-251.

[11] W. Hibbert, C.R. Dyer, and B. Paul, "Display of Scientific Data Structures for Algorithm Visualization," *Proc. Visualization '92* (Oct. 1992), pp. 139-146.

[12] T.J. O'Donnell and A.J. Olson, "Gramps – A Graphics Language Interpreter for Real-Time Interactive Three-Dimensional Picture Editing and Animation," *Computer Graphics* (Proceedings of SIGGRAPH 1981), Vol. 15, No. 3, Aug. 1981, pp. 133-142.

[13] M.C. Connolly and A.J. Olson, "Granny, a Companion to Gramps for the Real-Time Manipulation of Macromolecular Models." *Computers and Chemistry*, Vol. 9, No. 1, 1985, pp. 1-6.

[14] D.C. Richardson and J.S. Richardson, "The Kinemage: A Tool for Scientific Communication," *Protein Science*, Vol. 1, 1992, pp. 3-9.

# The Nanomanipulator:
## A Virtual-Reality Interface for a Scanning Tunneling Microscope

Russell M. Taylor II[1]
Warren Robinett[1]
Vernon L. Chi[1]
Frederick P. Brooks, Jr.[1]
William V. Wright[1]

Department of Computer Science
University of North Carolina, Chapel Hill

R. Stanley Williams[2]
Erik J. Snyder[3]

Department of Chemistry
University of California, Los Angeles

**Abstract**

We present an atomic-scale teleoperation system that uses a head-mounted display and force-feedback manipulator arm for a user interface and a Scanning Tunneling Microscope (STM) as a sensor and effector. The system approximates presence at the atomic scale, placing the scientist *on* the surface, *in* control, *while* the experiment is happening. A scientist using the Nanomanipulator can view incoming STM data, feel the surface, and modify the surface (using voltage pulses) in real time. The Nanomanipulator has been used to study the effects of bias pulse duration on the creation of gold mounds. We intend to use the system to make controlled modifications to silicon surfaces.

**CR Categories:** C.3 (Special-purpose and application-based systems), I.3.7 (Virtual reality), J.2 (Computer Applications Physical Sciences)

**Keywords:** haptic, force, scientific visualization, interactive graphics, virtual worlds, scanning tunneling microscopy, telepresence, teleoperation.

### 1. Introduction

We are just beginning to have fast enough graphics engines and acceptable trackers to allow us to provide scientists with a real-time immersive virtual-world interface to their instruments. We have brought this power to bear on the visualization of data from and control of a Scanning Tunneling Microscope with the UNC/UCLA Nanomanipulator system. The virtual-world interface demonstrably contributes to the power of the instrument.

The Scanning Tunneling Microscope (STM) was conceived in 1978 by G. Binnig and H. Rohrer at the IBM Zurich Research Laboratory and first demonstrated in 1981. It was originally designed to aid in understanding the growth, structures, and electrical properties of very thin oxide layers. [4] [5]

An STM consists of a piezoelectric positioning element, a conducting (usually metal) tip and a conducting sample (the surface under study). In our instrument, built by E.A. Eklund at UCLA, the piezoelectric crystal elements are arranged as three orthogonal bars, each of which controls one axis (see figure 1). As voltages are applied across the crystals, they change their lengths. Since the tip is rigidly attached to the crystals, they can be used to position the tip relative to the sample. Our STM can scan areas up to 200 nanometers (nm) on a side.

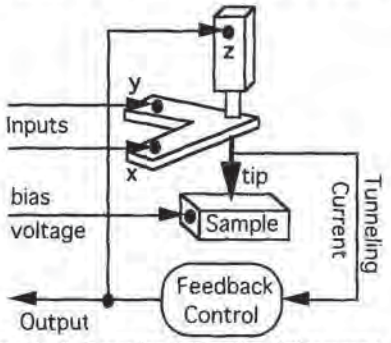



**Figure 1:** A Scanning Tunneling Microscope. The feedback control maintains the tip at a constant distance above the surface.

A bias voltage is applied to the sample with respect to the tip. At very close range (on the order of a few tenths of a nm), a tunneling current flows between the tip and the surface. This current decreases exponentially with increasing distance between the tip and the sample. In our configuration, the X and Y piezoelectric crystals are used to raster the tip back and forth across the surface in a boustrophedonic* pattern and the Z crystal is controlled by a feedback circuit that attempts to

[1] CB #3175, UNC, Chapel Hill NC 27599-3175.
(919) 962-1701 taylorr@cs.unc.edu
(919) 967-6375 robinett@cs.unc.edu
(919) 962-1742 chi@cs.unc.edu
(919) 962-1931 brooks@cs.unc.edu
(919) 962-1838 wright@cs.unc.edu

[2] Chemistry Dept., UCLA, Los Angeles, CA 90024-1569. williams@uclach.bitnet

[3] Aono Atomcraft Project, 5-9-9 Tohkohdai, Tsukuba-shi, Ibaraki. 300-26 JAPAN. ejs@odysseus.chem.ucla.edu



ACM-0-89791-601-8/93/008/0127 $01.50

*Literally, "as the ox plows."

maintain a fixed tunneling current and thus a constant distance between the tip and sample. A scan of the surface proceeds by repeatedly moving the tip in X and Y and then reading the voltage on the Z axis to determine surface height.

The STM is capable of resolving individual atoms in a sample. The radii of atoms range from 0.03 to 0.27 nm, or approximately 1 billionth the size of common objects, such as a golf ball or a basketball. Typical chemical bonds range from 0.15 to 0.25 nm. For comparison, a typical feature on a current integrated circuit might be 1 micron (1,000 nm) across. Optical microscopes are limited in resolving power to approximately the wavelength of the radiation used in imaging, which is 400-700 nm.

The STM uses a very sharp physical probe to gather information about the sample surface, rather than analyzing reflected photons or electrons. The key to the resolution of the STM is that the length of the piezoelectric positioners can be accurately controlled to 0.01 nm, and that the tunneling current is extremely sensitive to tip-to-sample separation (moving the tip 0.1 nm closer to the surface increases the tunneling current by a factor of 10).

Unlike other microscopes, the STM provides its information as an elevation map rather than a projected image. The scientist wants to understand the geometry of the three-dimensional surface, so these values must be interpreted. The most natural method of interpretation is to reconstruct the surface from the sampled height information, a common process in computer graphics.

In addition to its ability to map the surface, the tip of the STM can be used as a local probe to modify the surface. [2] This makes the STM useful for nanofabrication. There are at least two ways this can be accomplished. The first is to physically contact the surface with the tip, which causes large and unpredictable modifications to both the tip and the surface. The second, more controlled method is to apply a voltage pulse between the tip and the surface. Since the distance between the two is so small, even moderate voltages produce a strong electric field. Both Lyo and Avouris and Kobayashi et al. have shown that it is possible to alter the surface of a silicon crystal with such fields. The former authors have successfully removed clusters and even individual Si atoms from a surface by applying voltage pulses of +3V to the sample under study (with the tip grounded). The amount of material transferred in each pulse depended on the distance from the bottom of the tip to the sample surface (the smaller the distance, the larger the field and thus the more material transferred). They were also able to transfer atoms from the tip back onto the surface by applying voltage pulses of -3V to the sample. Kobayashi et al. have been able to form trenches only a few nanometers wide. They scanned the tip over the surface at a speed of 50 nm/s while holding the sample at a constant voltage of either polarity in the range from 4-10 volts. They used tip-sample separations significantly larger than those used by Lyo and Avouris. These studies have demonstrated the feasibility of altering the structure of a surface literally one atom at a time. What they lacked was the ability to interactively view the surface while it was being modified. [14] [11]

We have built a virtual-worlds interface that converts the STM from a remote batch data collector to a real-time user-guided data collector, and from a remote batch surface modifier to a real-time user-guided surface modifier. The material surface under the STM is sampled and then graphically reconstructed, lighted, and presented to the user at human scale, magnified approximately a billion times. In January 1992, an STM built at UCLA was brought to North Carolina and interfaced to the existing hardware and software of UNC's Head-Mounted Display project and the GROPE force display project. [18] We named the system the "Nanomanipulator" because it allows the user to see, feel, and manipulate matter at the nanometer scale.

The STM functions as both the imager and effector in this atomic-scale teleoperator system. The system operates in three modes. In *raster-scan mode*, the STM tip moves back and forth, continually streaming in new surface height data on a user-specified grid. This data updates the reconstructed surface model in real time. Independently and asynchronously, the viewer may fly about the surface, or hold it at arm's length and tilt it so that the directional illumination reveals and highlights surface detail. The surface model serves as the buffer converting between the back and forth slow scanning of the STM and the TV-like fast scanning of the display system.

In *feel mode*, the scientist uses the manipulator arm to move the STM tip directly (as the crow flies) over the surface, feeling the contours, and perceiving particular point heights, as the STM visual cursor traverses the surface image.

In *pulse mode*, the user also moves the tip directly over the surface, and, with a hand trigger, may select locations to fire bias pulses, modifying the surface.

The user interface through which the human user perceives the microscopic world consists of a stereoscopic head-mounted display and a force-feedback handgrip. The viewpoint changes of the user's head and control gestures of the user's hand are scaled down by the Nanomanipulator to control the viewpoint from which the microscopic world is seen by the user and to control surface modifications enacted by the STM.

Figure 2 shows the user interface for the Nanomanipulator.

**Figure 2:** User interface for the Nanomanipulator system. The user can control the action of the STM tip with the force-feedback ARM, feel surface contours, and specify bias pulses by pressing the finger trigger. The user sees the sample surface through the head-mounted display.

The graphics system that generates the stereoscopic images for the HMD provides highly detailed shaded 3D color images in real time. The HMD and head-tracker allow the graphics to be generated in coordination with the user's voluntary head

motions, so that users perceive themselves to be surrounded by the microscopic environment.

## 2. Previous STM Visualization

The standard method of STM data visualization during data collection is to construct gray scale images where dot brightness corresponds to surface height at each point in the image. For later offline viewing or publication, most visualization is done using various graphing routines on personal computers. These packages draw a connected line for each scanline the tip has made, possibly doing hidden line removal on areas that would be occluded. Color is used effectively to show either surface height or other surface properties. For presentation, shaded surface images are sometimes computed. [7] [15] [20] [21]

Dr. Joe Lyding at the Beckman Institute of the University of Illinois has produced an Application Visualization System (AVS) interface module that gives data from the STM scans to other AVS modules for high-quality interactive rendering of surfaces using Gouraud shading. This results in shaded images from a given point of view at rates on the order of one image per minute. Dr. Lyding has found that his interactive viewing system guides experiments by showing interesting areas of study while the data is still being collected. [personal communication] We have found the same effect in our system.

Dr. Besenbacher et al. at the University of Aarhus, Denmark, have taken another approach. They still draw the images on a personal computer, but they make images from successive scans and put them onto videotape for later viewing. This allows them to view surface dynamics that happened during the experiment. The dynamics cannot be viewed while the experiment is in progress, but nonetheless they have found the motion display to be very useful. Referring to his videotaped images of the scanned surface, Dr. Besenbacher writes "...one can record *STM movies* and thereby visualize in real time and space dynamical processes on metal and semiconductor surfaces. Such information, which cannot be obtained by any other means, is very decisive for a full understanding of both the growth mode of reconstructed phases and the resulting static structure." [3] We have also found this property of our system to be valuable.

## 3. The Nanomanipulator

The goal of this system is to approach an ideal interface for the scientist — presence on the surface itself, with the ability to interact with the surface in real time. The Nanomanipulator system mediates between the human-scale actions of the user and the atomic-scale actions of the STM. A Head-Mounted Display (HMD) and Force-Feedback Argonne-III Remote Manipulator (ARM) provide an immersive virtual environment in which the user is given the ability to act at the atomic scale. The purpose of this system is to scale the STM environment (nanometer scale) up to human size (meter scale) and to provide a means for making changes in this environment - it seeks to create teleoperation at the atomic scale, which requires a scaling factor of $10^9$.

### 3.1 System Structure

The Nanomanipulator system comprises several parts, including the STM itself, the Pixel-Planes 5 graphics engine, a real-time control computer (currently an IRIS 240) to control the STM, and a user interface subsystem (running on a Sun 4). See figure 3 for a system diagram.

Due to the varying requirements of its different parts, the system has been segmented into a distributed set of heterogeneous processors. These subsystems act as communicating sequential processes. The ARM, STM, and user interface sections communicate over an ethernet. The user interface and Pixel-Planes 5 communicate over a VME bus. The various graphics processors within Pixel-Planes 5 communicate over a 640 MByte/s ring network.

Our ethernet LAN provides about a 1 megabit/second sustained path between hosts on the network. This bandwidth is adequate to handle the data coming from the STM, which is limited in scan rate by the electronics and piezoelectric crystal resonances to less than 50 kilobits/second.

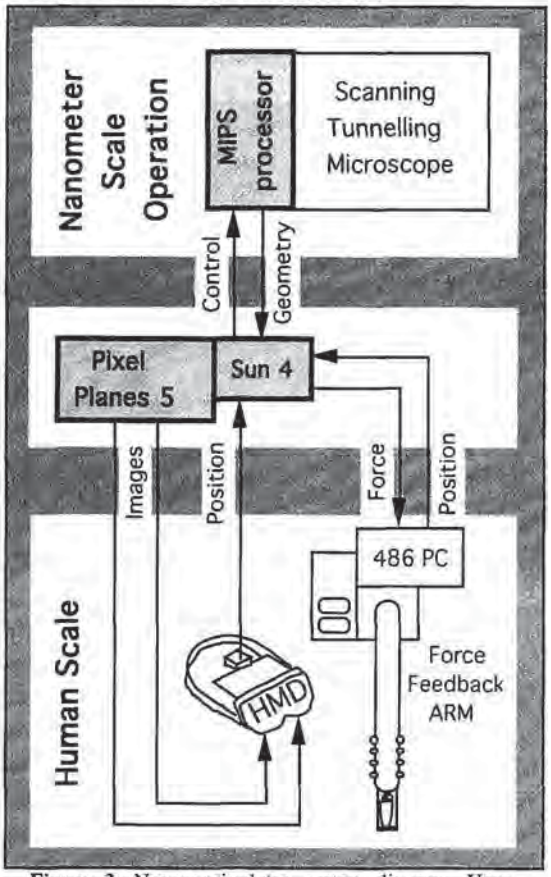



**Figure 3:** Nanomanipulator system diagram. User Interface code running on the Sun 4 mediates between the atomic-scale operations in the STM and the human-scale operations of the user. Pixel-Planes 5 provides real time shaded stereo images.

**STM Controller Process** The STM is controlled by one of the processors on an IRIS 240. The X and Y position are controlled by D/A cards and the Z value is read back in through an A/D card. The bias pulses are provided by an HP8131A pulse generator that is interfaced to the computer through an IEEE488.2 (HPIB) bus. The pulse generator is capable of producing controlled pulses as narrow as 500 ps. We have carefully controlled the impedance out to the end of the sample; however the tip side impedance is unknown. As a result, we

expect the pulses to be repeatable but of unknown shape as they cross the sample-tip boundary.

The control process accepts commands from the user interface and produces geometry information. When the process is scanning, it streams data to the user interface continually, sending up to 30 packets per second with whatever results have accumulated from scanning during that time. The process is also capable of moving the tip to a given location and returning the height there immediately or applying a pulse there.

**User Interface Process** The user interface process routes information among the various parts of the system and translates user commands into system commands. It is built on top of several existing software systems that were developed at UNC. The tracking, display, virtual-world editing, menus, and force feedback ARM control were all provided by existing libraries. [10] [17]

The HMD that we currently use is made by Virtual Research. This is a stereo color display helmet that is tracked by a Polhemus tracker. The HMD group has written a software library to handle the interface between the Polhemus and user code. Several projects at UNC have designed virtual worlds for building walkthrough, radiation treatment planning, molecular visualization, and particle systems. [10] The Ultrasound project has interfaced incoming data to Pixel-Planes 5 and overlaid the data on the real world. [1]

The ARM is an Argonne III Remote Manipulator that is interfaced to an IBM PC through A/D and D/A cards. A control library has been written by the GROPE project to access the ARM. Several systems have been designed that use the ARM, leading to the force feedback molecular docking program. [6]

**Pixel-Planes 5** The images that are generated for the HMD are produced by Pixel-Planes 5 (Pxpl5). This is a massively parallel graphics engine that was developed at UNC-CH under the direction of Henry Fuchs and John Poulton [9]. Along with the hardware itself, the Pixel-Planes team has created the PPHIGS library as a programmer's interface to the machine. This is a graphics library modeled after the PHIGS standard. Pxpl5 has dozens of I860 processors used as graphics processors (GPs). These processors run the display code and can also be programmed via C code callbacks within the display list.

Pixel-Planes 5 is optimized for drawing static display lists of triangles in real time; performing many updates to the display list ordinarily slows the update rate down unacceptably. New height data arrives from the STM at a rate of up to 500 samples per second, which is 25 data points per frame. Each data point affects the normals at the four surrounding points, and each normal affects up to six surrounding triangles. The system must therefore be able to perform about 600 triangle updates per frame while still maintaining the 20 Hz update rate.

Fortunately, it is possible to program the GPs on Pxpl5 directly, and the display list is split among the GPs. Each GP makes a list for each point of where it is stored in the display list (usually in 6 different triangles) and this list is used to propagate changes into the display list. We send the new point information to the GPs and each one modifies its portion of the display list in parallel. This provides sufficient speed to handle the changes.

### 3.2 Towards the ideal user interface

The data from the STM is presented to the user as a three-dimensional surface drawn in the head-mounted display. The surface appears to be made of shiny plastic and is colored according to height, with lower areas bluer and higher areas redder. The user sees a line sweeping across the surface as new position updates are received from the STM. The surface is sampled on a regular grid whose spacing is specified at run time. We typically use 80 samples per line and 80 lines on the surface, but the user can interactively trade speed for resolution by re-running the program.

The user's hand is tracked with the ARM. An icon is drawn at the hand location to graphically indicate the current mode of operation, much as the cursor on a Macintosh computer changes shape to indicate current mode. The user selects between modes using a pull-down menu system that is brought up by a thumb trigger on the ARM handgrip. The menu also allows saving the STM data to disk for later analysis.

The finger trigger on the handgrip has different behaviors depending on the mode the user is in. The modes dealing with viewing the virtual world are *fly*, *grab*, *scale up*, and *scale down*. [19] The modes dealing with controlling the STM are *feel*, *pulse*, *select part*, and *select all*.

**Fly** In *fly* mode, the user holds down the finger trigger to translate through the microscopic landscape in the direction pointed by the handgrip.

**Grab** In *grab* mode, the user can change the orientation of the virtual world by, in effect, grabbing the "fabric of space" and rotating it. The user moves the handgrip to the desired center of rotation and then holds the trigger down while rotating the handgrip.

**Scale** One wants to dynamically change the scale factor between the user's scale and the virtual world that represents the microscopic landscape. Such a change is perceived by the user as the virtual world expanding or shrinking. At a magnification of $10^9$, a 10 cm gesture by the user would move the STM tip a distance of 0.1 nm on the sample surface, whereas at a magnification of $10^7$ the same gesture would move the tip 10 nm. In *scale up* mode, when the user holds down the trigger, the virtual world expands at a fixed rate, using the handgrip location as the center of expansion. *Scale down* mode is similar.

**Feel** In *feel* mode, the user moves the handgrip around on the surface and feels a force that pulls it up or down to the surface. As the handgrip moves about in X and Y, the STM tip follows the motion. As the user moves, the height of the surface at the cursor location is sampled and a linear restoring force is applied to the handgrip in the Z direction towards the surface. The X and Y position do not have to correspond to grid locations in this mode, so it can be used to supersample the surface. At no time does the user actually control the Z motion of the tip; that is controlled by an electronic feedback circuit. This was a design choice that prevents the user from crashing the tip into the surface. The forces felt are simulated spring forces based on the handgrip height versus the surface height at the given (X,Y) location.

**Pulse** In *pulse* mode, the user can cause the STM to produce bias pulses. The user moves the handgrip cursor over the place on the surface where the pulse is to go and presses the finger trigger. The STM moves the tip to the indicated spot,

pulses the bias, and then resumes scanning the image. This is the mode that allows surface modification; for example, placing blobs of gold from the tip onto the surface.

**Select** The *select* commands are used to examine part of the surface rapidly. The STM takes about 33 seconds to scan the whole surface, since the tip velocity is constrained. Faster updates for smaller surface areas are useful when making changes to the surface. The user can select a section of the grid to be scanned by pressing the finger trigger, dragging, and releasing to indicate a rectangle of interest. The faster updates allow the user to react to mistakes in the modifications as they occur and immediately see when modifications have been completed and with what result.

Since all of the above interaction is possible while the surface is still under the microscope and being scanned, the user can direct the study of the surface based on data just obtained. If part of the surface looks more interesting and merits further study, the user can select that region of interest and get faster scanning, since only part of the surface is scanned. If there is some question whether an area contains a certain feature or if it contains only noise, it can either be rapidly scanned multiple times or felt (and thereby resampled) by moving the tip over it to see if the apparent feature persists.

### 3.3 System Performance

Performance tests were run for the system on a Pixel-Planes machine with 41 GPs and 20 renderers. The region examined was a 200x200 nm area of gold scanned with a cut gold tip. We took a 100x100 grid of samples and tessellated each grid square with two triangles. For stereoscopic vision, this is 40,000 triangles each frame. The display rate was 20 frames/second when viewing the entire data set (800K triangles per second). For an 80x80 grid, the rate was 24 hz. These rates are sufficient to make the user feel present on the remote (in scale) surface.

The tip was acquiring new data at a rate of three grid lines per second. Since there were 100 lines in the grid, this means that the entire grid was scanned every 33 seconds. The world is updated as the samples come in, so the user sees a line of updates sweeping back and forth across the scene. Our scan rate is presently limited by the piezo resonance frequencies and the frequency response of our feedback circuit.

We were initially unable to maintain the desired update rate in the *feel surface* mode. In this mode, the position of the hand must be sensed, the tip moved to the correct location, and the force applied to the user. Doing all of this each frame halved the update rate until we pipelined the operation with the display operation. The user therefore feels the force for the previous video frame. When the user moves slowly over a surface, the effect of this lag is negligible.

## 4. Results

A prototype viewing system, called the Microscape, was set up at UNC before the actual STM was installed. This system allowed the user to fly around and feel a surface that was generated from STM data stored on disk. One of the data sets was an ion-bombarded graphite sample that contained several sheets of graphite that had been pushed upward out of the sample, much like the earth is pushed upwards along an earthquake fault. Previously, these features had been thought to be noise, but the real time shading of the Microscape system showed clearly that they were not aligned with the scan direction and that they were regularly spaced. This discovery convinced us of the power of the visualization and encouraged us to put the current system in place.

Once the UCLA STM had been set up at UNC and the system could modify surfaces, we duplicated the experiments discussed in [16], depositing gold from the tip onto a gold surface. We used a cut gold tip, rather than an etched tip. We used a bias voltage of 235 mV and a tunneling current of 1 nA. For 5V pulse heights, we found that 20 ns pulses repeatably moved material but that 10 and 15 ns pulses rarely did. We also found that the gold mounds we made would anneal as they were repeatedly scanned, often disappearing entirely within a few scans, even though they persisted for long periods if not scanned. More robust structures could be formed by repeatedly pulsing the same location after each scan. One such feature remained in place for the duration of several experiments - tens of scans. Figure 4 shows a gold surface before and after voltage pulses were applied.

**Figure 4a:** Gold surface before bias pulses were applied. Surface is colored according to height, with higher areas being redder. Scan area is 100 by 100 nanometers. There are 80 samples each in x and y.

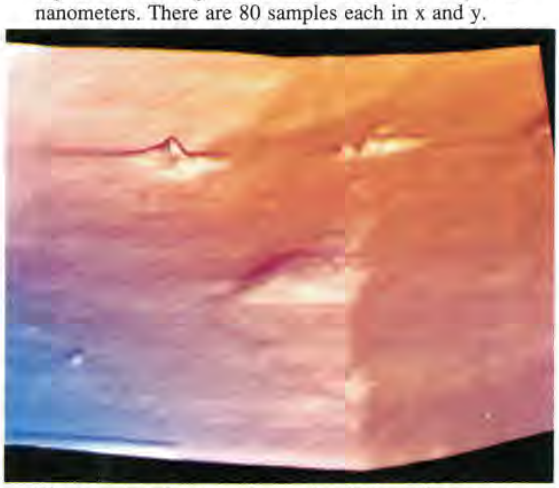


**Figure 4b:** Gold surface after bias pulses were applied. Several gold mounds have been deposited on the surface. The large mound in the center is about 20 nanometers wide. The smaller bumps were unintentional.

## 5. Significance

The significance of the virtual-reality interface to the STM is that it gives the scientist simulated *presence* on the sample surface. The benefits of this are: improved perception of 3D structures, more effective exploration of the sample, the ability to observe dynamic processes in near real time, and the ability to interactively modify the surface. To put it in plain language, when you are present somewhere, you can look around, you can look at things from different angles, you can feel interesting things at arm's length, you can watch the behavior of things that move or change, you can pick up things and rearrange them, and you can tweak things to see how they respond. People use all of these behaviors when they investigate places and things in the macroscopic world. Scientists, through the mediation of the Nanomanipulator, can engage in all these exploratory behaviors at the atomic scale, with their actions scaled down from meters to nanometers.

The scientist's ability to recognize specific molecular structures within the noisy, sampled data is improved by using stereoscopic, shaded 3D color graphics with specular highlights. This improved perception of 3D structures, in comparison with 2D gray-scale images with brightness coding height, was evident from the first month of the collaboration; The Williams team at UCLA recognized the up-tilted graphite planes on the first viewing of their STM data rendered as a fly-through with shaded 3D color graphics. They had puzzled over the data for months previously.

Providing stereoscopic, rather than monoscopic, viewing is useful to the scientist because the stereo provides a direct perception of depth for nearby virtual objects. Allowing accurate perception of the 3D spatial structure of STM data makes it possible for scientists to use their own specialized knowledge to recognize structures and features of interest in the data.

Displaying the STM data through a head-mounted display as an intuitively accessed surrounding virtual world allows the 3D graphical world seen by the scientist to be spatially superimposed with the force field felt through the force-feedback handgrip. This allows the user to see and feel a virtual object at a single location in space, just as occurs with real objects. This is harder to do without an HMD, since a large monitor screen tends to get in the way of the feeling gestures of the hand. Also, we believe that displaying the 3D data as a surrounding virtual world helps to better orient the scientist within the data.

The Nanomanipulator allows the scientist to interactively explore the sample in the STM in new ways. First, since the data produced by the STM is a grid of elevations, it can be graphically rendered from any viewpoint. This means that scientists, by means of gestural commands, can translate themselves over the sample surface, scale the surface up and down, and rotate the surface to any orientation. This allows the scientist to fly down into canyons on the surface, and even to fly beneath the surface and see it from below. This ability to see the sample from an arbitrary viewpoint is not possible with imaging-type microscopes, such as optical and scanning-electron microscopes.

The Nanomanipulator allows a second new type of interactive exploration of the sample: the user can interactively modify the scanning parameters of the STM. Current practice at most STM sites is to collect data with the STM first, and then to view and analyze it later, off-line. In such an arrangement, if a feature of interest lies halfway off the sample grid, or if the grid is too coarse to get a good look at the feature, there is not much to be done. But with the ability to scan different areas and at different scales as the exploration progresses, the scientist is empowered to explore more effectively. For example, if a feature of possible interest is seen in a wide-area coarse scan, the user can then interactively focus the scan on the feature of interest to get a high-resolution view of the feature. Having an expert human observer in the control loop makes this sort of interactive exploration very powerful.

**Figure 5a:** Gray-scale image of ion-bombarded graphite sample. This is the standard representation given in real time by STM user interfaces.

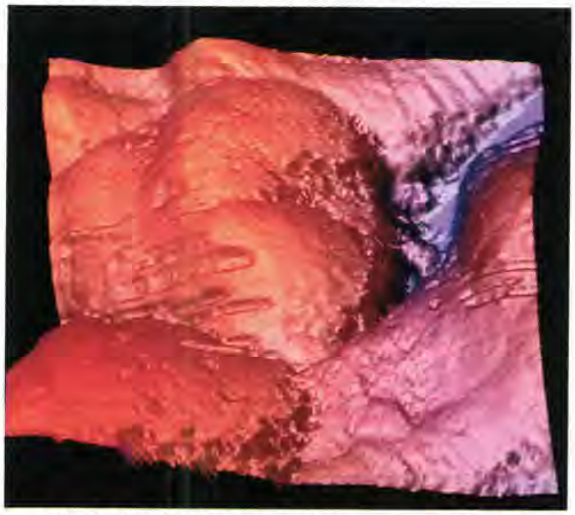


**Figure 5b:** Shaded image of the same sample. This image shows clearly the tip scratches on the lower left part of the sample and the ripples in the upper right corner caused by sheets of graphite pushing upward out of the surface.

Since the data from the STM comes in as the sampling occurs in real-time, the scientist can observe dynamic processes with time scales of seconds as they evolve on the surface.

The Nanomanipulator gives the scientist the ability to make controlled modifications to the sample surface. This capability takes the Nanomanipulator beyond being a mere passive observing instrument, and makes it rather a tool for conducting experiments on the atomic scale, for fabricating nanometer-scale structures, and perhaps ultimately for building molecular structures atom by atom. We foresee that this will allow the construction of new structures and materials, such as nanometer-scale electronic circuits, which are not now possible to fabricate in any way whatsoever. The advantage of using the Nanomanipulator in this process is that the operator can detect any mistakes or aberrations in the structure being built and correct them in real time. A skilled user can respond to surprise much more creatively than a computer algorithm, and when sculpting at atomic scale, there are bound to be many surprises.

## 6. Future Directions

The first applications of the Nanomanipulator will be for *nanomachining* of structures on surfaces. This will involve processes similar to those in present electronic device technology (which has about 1/2 micron feature size), where a thin film, in this case only a few monolayers thick, is deposited onto a substrate. The Nanomanipulator will then be used as a mill to directly remove material and pattern a structure on the deposited film, without going through the stages of resist deposition, exposure, and removal. Initially, these structures will be test devices, such as single-electron transistors or platforms for recognizing and immobilizing particular molecular species. The Nanomanipulator will also be used as a probe to study the properties of these devices.

In order for nanomachined devices to become useful tools rather than laboratory curiosities or atomic scale artworks, they will have to be manufactured in massively parallel processes. This will only be possible after we have learned how to manipulate atoms as reproducibly as quantum uncertainty and the second law of thermodynamics will allow. The experience gained in working with the Nanomanipulator may provide the basis for that understanding.

## Acknowledgments

We keenly appreciate support for this research: from the National Institutes of Health, The Office of Naval Research, the Defense Advanced Research Projects Agency, and the National Science Foundation. Argonne National Laboratories furnished the force-feedback ARM.

The local projects at UCLA were supported in part by the Office of Naval Research and by NSF grant DMR-89-22027.

Thanks to Roberto Melo for help characterizing the system and for designing and building a new feedback circuit.

We appreciate the groundwork laid by other projects at UNC –the Pixel-Planes 5 group, the Head-Mounted Display group, the GROPE project, and the Tracker project – without which we would not have been able to get up and running.

Thanks to David Banks for his help and suggestions that made the paper twice as good as it was.

Thanks to UNC computer Facilities Group and to the Microelectronic Systems Lab for providing an environment where we can work on our project rather than just on our computers.

## References

[1] Bajura, Michael, Henry Fuchs, and Ryutarou Ohbuchi. Merging Virtual Objects with the Real World: Seeing Ultrasound Imagery within the Patient. Proceedings of SIGGRAPH '92 (Chicago, Illinois, July 26-31, 1992). In *Computer Graphics* 26, 2 (July 1992), 203-210.

[2] Becker, R.S., J.A. Golovchenko and B.S. Swartzentruber. Atomic-Scale Surface Modifications Using a Tunnelling Microscope. *Nature* 325(1987), 419.

[3] Besenbacher, F., F. Jensen, E. Lægsgaard, K. Mortensen, and I. Stensgaard. Visualization of the Dynamics in Surface Reconstructions. *Journal of Vacuum Science Technology*, B 9 (2), Mar/Apr 1991, 874-877.

[4] Binnig, G. and H. Rohrer. Scanning Tunneling Microscopy. *Helvetica Physica Acta*. 55 (1982), 726-735.

[5] Binnig, G. and H. Rohrer. Scanning Tunneling Microscopy - From Birth to Adolescence. *Reviews of Modern Physics*, 59(3) July 1987, 615-625.

[6] Brooks, F. P., Jr., M. Ouh-Young, J. J. Batter, and P. J. Kilpatrick. Project GROPE - Haptic Displays for Scientific Visualization. Proceedings of SIGGRAPH '90. In *Computer Graphics* 24, 4 (August 1990), 177-185.

[7] Edstrom, Ronald D. and Maria A. Miller. Scanning Tunneling Microscopy and Atomic Force Microscopy Visualization of the Components of the Skeletal Muscle Glycogenolytic Complex. *Journal of Vacuum Science Technology, B* 9 (2) (Mar/Apr 1991), 1248-1252.

[8] Eklund, E.A.. Correlation from Randomness: Scanning Tunneling Microscopy Applied to the Quantitative Analysis of Sputtered Graphite Surfaces. *Ph. D. Thesis*, UCLA. 1991.

[9] Fuchs, Henry, John Poulton, John Eyles, Trey Greer, Jack Goldfeather, David Ellsworth, Steve Molnar, Greg Turk, Brice Tebbs, and Laura Israel. Pixel-Planes 5: A Heterogeneous Multiprocessor Graphics System Using Processor-Enhanced Memories. Proceedings of SIGGRAPH '89. In *Computer Graphics*, 19, 3 (1989). 79-88.

[10] Holloway, Richard, Henry Fuchs, and Warren Robinett. Virtual-Worlds Research at the University of North Carolina at Chapel Hill. *Proc. Computer Graphics '91*, London.

[11] Kobayashi, A., F. Grey, R. S. Williams, and M. Aono. Nanometer-Scale Silicon Groove Formation by STM. *Science*, in press, 1993.

[12] Komuro, M., S. Okayama, O. Kitamura, W. Mizutani, H. Tokumoto, and K. Kajimura. Nanometer Structure Fabricated by FIB and its Observation by STM. *Microelectronic Engineering* 6 (1987), 343-348.

[13] Lyding, J. W. , S. Skala, J. S. Hubacek, R. Brockenbrough, and G. Gammie. Variable-temperature scanning tunneling microscope. *Rev. Sci. Instrum.* 59 (9), (September 1988), 1897-1902.

[14] Lyo, I.W. and Ph. Avouris. Field-Induced Nanometer to Atomic Scale Manipulation of Si Surfaces with the Scanning Tunneling Microscope. *Science* 253 (1991), 173.

[15] Magonov, S. N., G. Bar, E. Keller, E. B. Yagubskii, and H. J. Cantow. Atomic Scale Surface Studies of Conductive Organic Compounds. *Synthetic Metals*, 40 (1991), 247-256.

[16] Mamin, H. J., S. Chiang, H. Birj, P. H. Guethner, and D. Rugar. Gold deposition from a scanning tunneling microscope tip. J. *Vac. Sci. Technol. B* 9 (2) (Mar/Apr 1991), 1398-1402.

[17] Ouh-young, Ming. Force Display In Molecular Docking. *Ph. D. Thesis*, University of North Carolina at Chapel Hill, 1990.

[18] Robinett, W., R. Taylor, V. Chi, W. V. Wright, F. P. Brooks Jr., R. S. Williams, and E. J. Snyder. The Nanomanipulator: An Atomic-Scale Teleoperator. *SIGGRAPH '92 course notes* for "Implementation of Immersive Virtual Environments."

[19] Robinett, W., and R. Holloway. Implementation of Flying, Scaling, and Grabbing in Virtual Worlds. *ACM Symposium on Interactive 3D Graphics*, Cambridge MA (1992).

[20] Snyder, Eric J., Mark S. Anderson, William M. Tong, R. Stanley Williams, Samir J. Anz, Marcos M. Alvarez, Yves Rubin, François N. Diederich, and Robert L. Whetten. Atomic Force Microscope Studies of Fullerene Films: Highly Stable C60 fcc (311) Free Surfaces. *Science*, 253, 12 (July 1991), 171-173.

[21] Stoll, E. P.. Picture processing and three-dimensional visualization of data from scanning tunneling and atomic force microscopy. *IBM Journal of Research and Development*, 35, . 1/2 (January/March 1991), 67-77.



# Surround-Screen Projection-Based Virtual Reality: The Design and Implementation of the CAVE

Carolina Cruz-Neira†
Daniel J. Sandin
Thomas A. DeFanti
Electronic Visualization Laboratory (EVL)
The University of Illinois at Chicago

## Abstract

This paper describes the CAVE (CAVE Automatic Virtual Environment) virtual reality/scientific visualization system in detail and demonstrates that projection technology applied to virtual-reality goals achieves a system that matches the quality of workstation screens in terms of resolution, color, and flicker-free stereo. In addition, this format helps reduce the effect of common tracking and system latency errors. The off-axis perspective projection techniques we use are shown to be simple and straightforward. Our techniques for doing multi-screen stereo vision are enumerated, and design barriers, past and current, are described. Advantages and disadvantages of the projection paradigm are discussed, with an analysis of the effect of tracking noise and delay on the user. Successive refinement, a necessary tool for scientific visualization, is developed in the virtual reality context. The use of the CAVE as a one-to-many presentation device at SIGGRAPH '92 and Supercomputing '92 for computational science data is also mentioned.

**Keywords:** Virtual Reality, Stereoscopic Display, Head-Tracking, Projection Paradigms, Real-Time Manipulation

**CR Categories and Subject Descriptors:** I.3.7 [Three-Dimensional Graphics and Realism]: Virtual Reality; I.3.1 [Hardware Architecture]: Three-Dimensional Displays.

## 1. Introduction

### 1.1. Virtual Reality Overview

Howard Rheingold [11] defines virtual reality (VR) as an experience in which a person is "surrounded by a three-dimensional computer-generated representation, and is able to move around in the virtual world and see it from different angles, to reach into it, grab it, and reshape it." The authors of this paper prefer a definition more confined to the visual domain: a VR system is one which provides real-time viewer-centered head-tracking perspective with a large angle of view, interactive control, and binocular display. A competing term, *virtual environments (VE)*, chosen for "truth in advertising" [1], has a somewhat grander definition which also correctly encompasses touch, smell, and sound. Although VE is part of the CAVE acronym, we will use the initials VR herein to conform to mainstream usage.

† 851 S. Morgan, Room 1120 SEO (M/C 154). Chicago, IL 60607-7053. E-mail: cruz@bert.eecs.uic.edu



Several common systems satisfy some but not all of the VR definition above. Flight simulators provide vehicle tracking, not head tracking, and do not generally operate in binocular stereo. Omnimax theaters give a large angle of view [8], occasionally in stereo, but are not interactive. Head-tracked monitors [4][6] provide all but a large angle of view. Head-mounted displays (HMD) [7][13] and BOOMs [9] use motion of the actual display screens to achieve VR by our definition. Correct projection of the imagery on large screens can also create a VR experience, this being the subject of this paper.

Previous work in the VR area dates back to Sutherland [12], who in 1965 wrote about the "Ultimate Display." Later in the decade at the University of Utah, Jim Clark developed a system that allowed wireframe graphics VR to be seen through a head-mounted, BOOM-type display for his dissertation. The common VR devices today are the HMD and the BOOM. Lipscomb [4] showed a monitor-based system in the IBM booth at SIGGRAPH '91 and Deering [6] demonstrated the Virtual Portal, a closet-sized three-wall projection-based system, in the Sun Microsystems' booth at SIGGRAPH '92. The CAVE, our projection-based VR display [3], also premiered at SIGGRAPH '92. The Virtual Portal and CAVE have similar intent, but different implementation schemes.

To distinguish VR from previous developments in computer graphics, we list the depth cues one gets in the real world.

1 Occlusion (hidden surface)
2 Perspective projection
3 Binocular disparity (stereo glasses)
4 Motion Parallax (head motion)
5 Convergence (amount eyes rotate toward center of interest, basically your optical range finder)
6 Accommodation (eye focus, like a single-lens reflex as range finder)
7 Atmospheric (fog)
8 Lighting and Shadows

Conventional workstation graphics gives us 1, 2, 7, and 8. VR adds 3, 4, and 5. No graphics system implements accommodation clues; this is a source of confusion until a user learns to ignore the fact that everything is in focus, even things very close to the eyelash cutoff plane that should be blurry.

The name of our virtual reality theater, "CAVE," is both a recursive acronym (CAVE Automatic Virtual Environment) and a reference to "The Simile of the Cave" found in Plato's *Republic* [10], in which the philosopher discusses inferring reality (ideal forms) from projections (shadows) on the cave wall. The current CAVE was designed in early 1991, and it was implemented and demonstrated to visitors in late 1991. This paper discusses details of the CAVE design and implementation.

 ACM-0-89791-601-8/93/008/0135 $01.50

## 1.2. CAVE Motivation

Rather than having evolved from video games or flight simulation, the CAVE has its motivation rooted in scientific visualization and the SIGGRAPH '92 Showcase effort. The CAVE was designed to be a useful tool for scientific visualization. Showcase was an experiment; the Showcase chair, James E. George, and the Showcase committee advocated an environment for computational scientists to interactively present their research at a major professional conference in a one-to-many format on high-end workstations attached to large projection screens. The CAVE was developed as a "Virtual Reality Theater" with scientific content and projection that met the criteria of Showcase. The Showcase jury selected participants based on the scientific content of their research and the suitability of the content to projected presentations.

Attracting leading-edge computational scientists to use VR was not simple. The VR had to help them achieve scientific discoveries faster, without compromising the color, resolution, and flicker-free qualities they have come to expect using workstations. Scientists have been doing single-screen stereo graphics for more than 25 years; any VR system had to successfully compete. Most important, the VR display had to couple to remote data sources, supercomputers, and scientific instruments in a functional way. In total, the VR system had to offer a significant advantage to offset its packaging. The CAVE, which basically met all these criteria, therefore had success attracting serious collaborators in the high-performance computing and communications (HPCC) community.

To retain computational scientists as users, we have tried to match the VR display to the researchers' needs. Minimizing attachments and encumbrances have been goals, as has diminishing the effect of errors in the tracking and updating of data. Our overall motivation is to create a VR display that is good enough to get scientists to get up from their chairs, out of their offices, over to another building, perhaps even to travel to another institution.

## 1.3. CAVE Design

The CAVE we exhibit at conferences is a theater 10'x10'x10' made up of three rear-projection screens for walls and a down-projection screen for the floor, as shown in Figure 1. (Our development system at EVL is actually 7'x7'x7' due to ceiling height limitations.) Projectors throw full-color workstation fields (1280x512 stereo) at 120Hz onto the screens, giving between 2,000 and 4,000 linear pixel resolution to the surrounding composite image. Computer-controlled audio provides a sonification capability to multiple speakers. A user's head and hand are tracked with Polhemus or Ascension tethered electromagnetic sensors. Stereographics' LCD stereo shutter glasses are used to separate the alternate fields going to the eyes. Four Silicon Graphics high-end workstations create the imagery (one for each screen); they are tied to a fifth for serial communications to input devices and synchronization via fiber-optic reflective memory by Systran Corporation. The CAVE's theater area sits in a 30'x20'x13' room, provided that the projectors' optics are folded by mirrors. Conference use thus far has necessitated the building of a light-tight structure of this size on site to house the screens and projectors.

Goals that inspired the CAVE engineering effort include:

1 The desire for higher-resolution color images and good surround vision without geometric distortion.
2 Less sensitivity to head-rotation induced errors
3 The ability to mix VR imagery with real devices (like one's hand, for instance)
4 The need to guide and teach others in a reasonable way in artificial worlds
5 The desire to couple to networked supercomputers and data sources for successive refinement

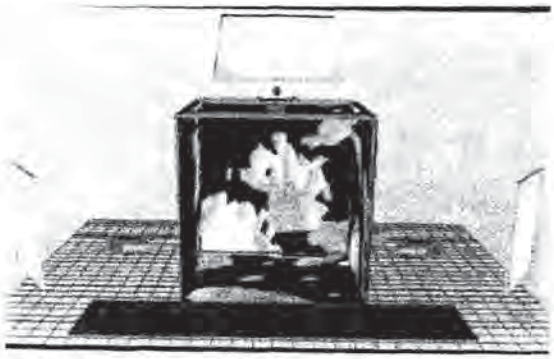


Figure 1: *CAVE diagram.* Graphics by Milana Huang, University of Illinois at Chicago

Significant barriers, now hurdled, include eliminating the lag inherent in common green video projector tubes, corner detailing, and frame accurate synchronization of the workstations; our solutions to these problems are described in detail in section 3. The electromagnetic trackers required building the CAVE screen support structure out of non-magnetic stainless steel (which is also relatively non-conductive), but non-linearities are still a problem, partially because conductive metal exists on the mirrors and in the floor under the concrete. Wheelchairs, especially electric ones, increase tracker noise and non-linearities as well.

Unsolved problems to date include removing the tracking tether so the user is less encumbered, moving the shutters from the eyes to the projectors so cheap cardboard polarizing glasses can be used, incorporating accurate directional sound with speakers, and bringing down the cost. These, and other problems we've encountered, are described in section 6.

The implementation details fall mainly into two categories: projection and stereo. These will be presented next.

# 2. Projection Details

## 2.1. Cube Sides As Projection Planes

One rarely noted fact in computer graphics is that the projection plane can be anywhere; it does not have to be perpendicular to the viewer (as typical on workstations, the HMD, and the BOOM). An example of an unusual projection plane is the hemisphere (like in Omnimax theaters or some flight simulators). However, projection on a sphere is outside the real-time capability of the ordinary high-end workstation. And, real-time capability is a necessity in VR.

The CAVE uses a cube as an approximation of a sphere. This simplification greatly aids people trying to stand in the space, and fits the capabilities of off-the-shelf graphics and high-resolution projection equipment, both of which are made to create and project imagery focused on flat rectangles. The defects one encounters in attempting to build a perfect cube are fortunately within the range of adjustment by standard video projectors; in particular, keystoning and pincushion corrections

can be utilized. Thus, the ability to match projected images at the seams and corners is effectively perfect, with tuning effort.

## 2.2. Window Projection Paradigm

The most common computer graphics projection paradigm is the camera view. This type of projection simulates the way an image is captured on film, and includes the direction the camera is pointed and the focal length, position, and twist angle of the lens. In the camera paradigm, stereo is typically achieved by using two cameras; this is the technique used by the HMD and BOOM. The CAVE instead uses a window projection paradigm in which the projection plane and projection point relative to the plane are specified, thus creating an off-axis perspective projection.

Fortunately, the Silicon Graphics' Graphics Library (GL) [14] provides a window projection function. Since this function can also be performed by two shears and a standard perspective projection, or, alternatively, by a translation, a standard perspective projection and a translation back, the window projection function can easily be constructed from more primitive functions, if not available in another graphics library.

In the CAVE, the projection plane locations correspond to the locations of the actual walls. Therefore, as the viewer moves around in the environment, the off-axis stereo projection is calculated according to his/her position with respect to the walls (see Figure 2).

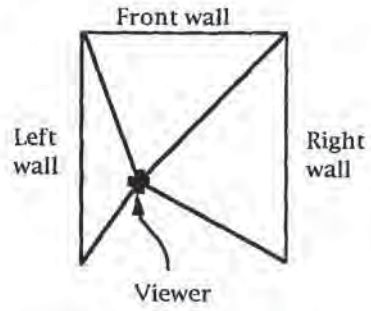



**Figure 2:** Off-axis projection

For the simplicity of the calculations, we assume that all the walls share the same reference coordinate system as shown in Figure 3. The origin of the coordinate system is placed in the center of the CAVE and it is a right-handed system with respect to the front wall. All the measurements from the trackers (position and orientation) are transformed to match this convention.

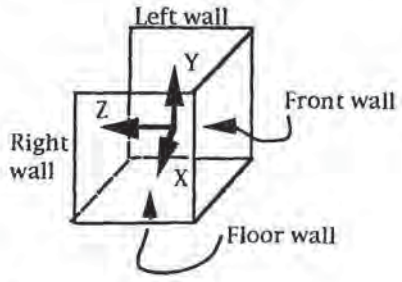



**Figure 3:** CAVE reference system.

Figure 4 shows a top diagram of the CAVE. The point Q' is the projection of the point Q. PP is the distance from the center of the CAVE to the front wall (5' for the 10'x10'x10' CAVE).

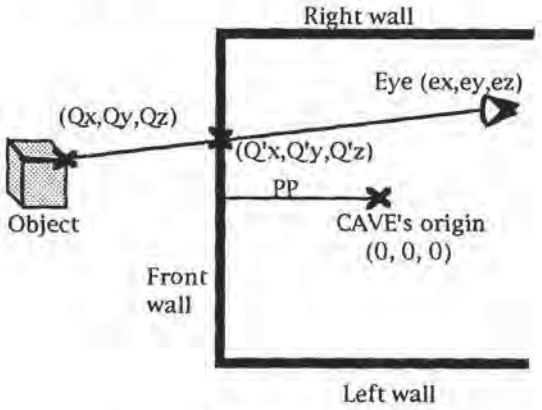



**Figure 4:** CAVE projection diagram

Using straightforward algebra and following the conventions in Figure 4, the projection Q' of a point $Q(Q_x, Q_y, Q_z)$ on the front wall is given by:

$$Q'_x = Q_x + \frac{(PP - Q_z)(e_x - Q_x)}{e_z - Q_z}$$

$$Q'_y = Q_y + \frac{(PP - Q_z)(e_y - Q_y)}{e_z - Q_z}$$

Thus, the general projection matrix is:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -\frac{e_x}{e_z - PP} & -\frac{e_y}{e_z - PP} & 1 & -\frac{1}{e_z - PP} \\ \frac{e_x PP}{e_z - PP} & \frac{e_y PP}{e_z - PP} & 0 & \frac{e_z}{e_z - PP} \end{pmatrix}$$

One important issue to mention is that, in the CAVE, the eyes are not assumed to be horizontal and in a plane that is perpendicular to the projection plane. A clear example of this is a situation in which the viewer is looking at one of the corners of the CAVE with his/her head tilted. Our tracker is mounted on top of the stereo glasses; it is raised 5.5" from the glasses to minimize interference and centered between the eyes. From the values obtined from the tracker, and assuming an interpupilar distance of 2.75", we can determine the position of each eye and its orientation with respect to each one of the walls before applying the projection matrix.

The reader can easily derive the matrices for the other walls of the CAVE. Notice that, since the walls of the CAVE are at exactly 90° from each other, the viewer's position with respect to the other walls are:

Left wall: $(e_z, e_y, e_x)$ Right wall: $(-e_z, e_y, e_x)$
Floor wall: $(e_x, e_z, -e_y)$

## 3. Stereo Vision Details

### 3.1. Convergence

To achieve stereo vision in the CAVE, we, in principle, do two off-axis stereo projections per screen, one for each eye. We need to obtain information from the tracker to accurately place each eye. We assume that the center of rotation of the eye is close enough to the nodal point (projection point) of the eye to not introduce significant error. Thus, as with other VR systems, where the eyes are looking does not enter into the calculations.

### 3.2. Frame Sequential Stereo

To get a different image to each eye, we use frame sequential stereo with synchronized shutter glasses. Infrared transmitters cause the lens for each eye to stay transparent for the proper 512 lines of the 1280x1024 image per screen, switching during vertical retrace time. We produce 120 fields per second, thus updating the whole image at 60Hz, producing a flicker-free image.

Note, however, that the green phosphor used in commercially available projection tubes has a persistence that is too long, so a user always sees both images anyway, destroying the stereo effect. Until Stereographics provided us with P43 coated green tubes by special order, we did our experiments (in 1991) in blue and red and shades of magenta. With luck, tube manufacturers will be motivated to add such tubes to their catalogs soon.

### 3.3. Distortion Correction

The HMD, BOOM, and monitor VR systems have significant geometric distortion inherent in their optics. Modern data projectors have extensive electronic adjustments to accurately correct geometric distortions.

### 3.4. Minimizing User Shadows

The three wall screens are rear projected so that the participants in the CAVE do not cast shadows. The floor is down projected so shadows are cast. We off-axis project the image from the front top instead of directly overhead, so the shadow of the user falls mainly behind him/her.

### 3.5. Frame Accurate Synchronization

Another problem we had to solve was the perfect synchronization of the screen updates. If the images are even one frame out of sync, the images in the corners crease and start to look sucked in like sofa cushions. We were unable to get adequate response from the UNIX system to synchronize within the 8ms needed, so (at the suggestion of Silicon Graphics staff) we went to reflective memory, a sort of shared cache arrangement among all the workstations. Reflective memory allows C-pointers to directly access chunks of memory, neatly bypassing the operating system. We intend to use the reflective memory for more sophisticated data sharing, including broadcasting of meshes, textures, and polygon lists. For now, however, reflective memory solves a nasty problem.

### 3.6. Edge Matching

Particular attention is paid to the edges and corners of the screen to avoid occlusion of stereo objects inside the room. We minimize the seams by stretching a 10'x30' plastic screen over 1/8" stainless steel cable under tension. This gives a seam of about a pixel or so in width, which can be seen but can also be easily ignored. Hence, the illusion of stereo in the CAVE is extremely powerful to the viewer. The floor butts up against the screen fairly perfectly (1/16") and presents no problem.

In the case of 3D movies and workstation screens, stereo objects in front of the screen (often the most interesting ones) have to stay pretty much centered. When a stereo object in front of a screen hits the edge (called "frame violation" in the jargon), it collapses the depth illusion since occlusion is a stronger depth cue than binocular disparity. The CAVE's screen edges are basically out of view (one can see the tops of the screens, but they are high up) so the stereo objects can be anywhere.

We were amazed at how much the floor adds to the experience; a user can walk around convincing objects that are being projected into the room. Since the tracker provides six degrees of information, the user's head can tilt as well, a natural way to look at objects. The HMD provides this capability, but BOOM hardware does not.

### 3.7 Minimizing Occlusion by Participants

A user's hand can cause stereo violation if an object is between the eyes and the hand, a rare enough situation. People are very eager to resolve stereo violation whenever it's easy so, in these instances, the user simply moves his/her hand out of the way.

A much more serious situation occurs with multiple people in the CAVE. If someone gets in the way of another viewer and an object is supposed to be projected between the two of them, the stereo collapses. We avoid this by having a "teacher" or "guide" control the navigation, but let the "student" or "tourist" be tracked and stand in front, thereby getting the best stereo experience without first having to learn to be an expert navigator of the data space, whatever it is. At conferences, we often jam a dozen people at a time in the CAVE and try to keep the images in front of the crowd. Since people more or less have to stay still or move together, the VR experience for all, however limited, is nevertheless pleasing.

### 3.8. Motion Sickness

Seeing one's own body or those of other people may in fact be a good idea. Of 9,000 or so people who have been in the CAVE, two have experienced enough nausea to complain about it, a very low ratio (apparently) for VR [1]. We don't yet know why the CAVE doesn't make people nauseous; perhaps it is content related. Our images primarily have to do with scientific data that changes over time, not roller coaster type motions with fast tilting horizons typical of many VR applications. Another explanation may be our better coping with fast head rotation (see next section).

## 4. Quantitative Analysis of the Effect of Tracking Noise and Latency

### 4.1. Introduction

Different VR modes have different responses to errors in tracking viewer position. One reason for the differences depends on whether the projection plane moves with the viewer (as with BOOMs and HMDs) or not (in the case of the monitor and CAVE). A second reason is the difference in the distance of the projection plane to the eye, which distinguishes the monitor implementation from the CAVE's.

## 4.2. Rotation errors

Tracking errors can be resolved into displacement errors and rotation errors. Actual problems are often a combination of the two. In the monitor and CAVE paradigms, since the projection plane does not move with the viewer's position and angle, a rotation about the projection point in the eye creates zero error. In the HMD/BOOM paradigm, a given rotational tracking error produces the same magnitude of rotational error in the image, but of opposite sign. This is a serious problem if the user's head rotates quickly because the whole visual scene first rotates with the head and then steps back into the proper place.

## 4.3. Analysis of displacement errors in the CAVE and monitor paradigms

The effect of displacement error for both the CAVE and the monitor paradigms is illustrated in Figure 8. The displacement error in eye tracking is ΔP (in a plane parallel to the projection plane), the distance from the eye to the projection plane is PD, and the distance to the object is Z. DISP is the distance error on the projection plane. α is the angular error.

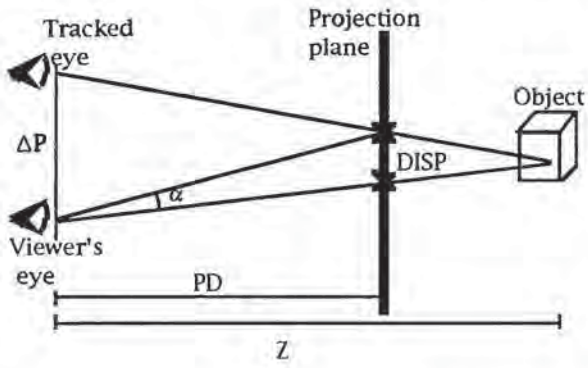



Figure 8: Effect of displacement error for both the CAVE and the monitor paradigms

$$DISP = \Delta P\left(\frac{Z-PD}{Z}\right)$$

$$\alpha = \arctan\left(\frac{DISP}{PD}\right)$$

$$\alpha \cong \frac{DISP}{PD} \text{ for small angles}$$

therefore,

$$(1)\ \alpha \cong \frac{\Delta P\frac{(Z-PD)}{Z}}{PD}$$

For large Z, $\frac{(Z-PD)}{Z} \cong 1$

therefore,

$$(2)\ \alpha \cong \frac{\Delta P}{PD}$$

For small Z, $\frac{(Z-PD)}{Z} \cong -\frac{PD}{Z}$

therefore,

$$(3)\ \alpha \cong -\frac{\Delta P}{Z}$$

For Z = PD (when the object is on the projection plane),

$$\frac{(Z-PD)}{Z} = 0$$

therefore,

$$(4)\ \alpha = 0$$

Equation (1) represents the approximate angular error α for a displacement tracking error ΔP in the monitor and CAVE paradigms.

Equation (2) shows that the larger projection distance PD associated with the CAVE, as compared to the monitor, makes angular error α due to displacement ΔP smaller for large distances Z to the object viewed.

Equation (3) shows that for very small Z values, the monitor and CAVE have similar responses.

Equation (4) shows that when objects are on the projection planes of the monitor or CAVE, the angular error α due to displacement is zero.

## 4.4. Analysis of displacement errors in the BOOM and HMD

A similar analysis for the BOOM and HMD is indicated in Figure 9.

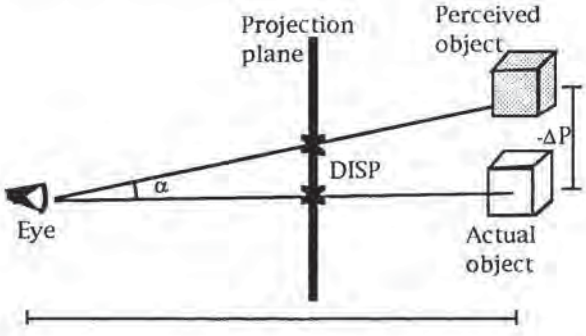



Figure 9: Effect of displacement error for both the HMD and the BOOM paradigms

A displacement error in tracking head position results in identical errors in both the eye position and the projection

plane position. This results in a negative displacement of the object being viewed.

$$\alpha = \arctan\left(\frac{-\Delta P}{Z}\right)$$

For small angles,

$$(5)\quad \alpha \cong \frac{-\Delta P}{Z}$$

Equation (5) shows that the angular error α is independent of the projection distance PD to the projection plane. Comparing equation (5) with (2), we see that the BOOM and HMD have less angular error α for displacement errors ΔP for large object distances Z than the CAVE/monitor models. Comparing equation (5) with (3), we see that the BOOM and HMD have similar angular errors α for small object distance Z.

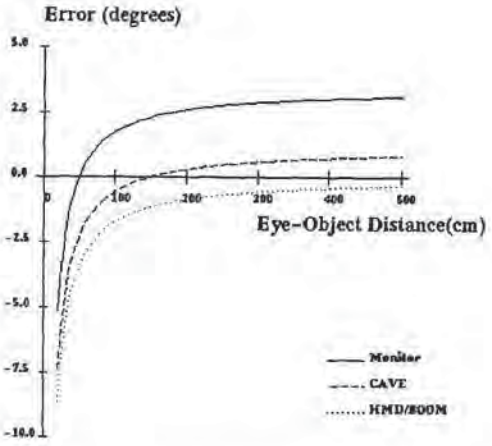



Figure 10: Angular error for a 3cm tracker displacement

Figure 10 graphs the angular error α due to a tracker displacement error ΔP of 3cm for object distances Z. This case represents a tracking error due to latency of a person moving 30cm/second combined with a display rate of 10 frames/second. For large object viewing distances (Z=500cm), the HMD/BOOM have the best performance, the CAVE has 2-1/2 times the error, and the monitor has 9 times the error. For small object viewing distances (Z=20cm), the monitor has the best performance, and the CAVE and HMD/BOOM have only slightly worse error magnitudes.

## 4.5. Examples of combined rotation and displacement tracking errors

Normal head motions like nodding and panning involve both rotation and displacement of the eyes. The combined effect of these errors may be approximated by summing the individual angular errors α. The assumed projection distances PD for the monitor and 10' CAVE are 50cm and 150cm, respectively.

Figure 11 graphs the angular error α as a function of eye/object distance Z due to a head rotation (pan) of 90 degrees/second and a display rate of 10 frames/second. It is assumed that the eyes are 5cm from the center of rotation. For large Z, the CAVE is 43 times better than the HMD/BOOM and 4 times better than the monitor. For small Z, the CAVE and monitor are 6 times better than the HMD/BOOM.

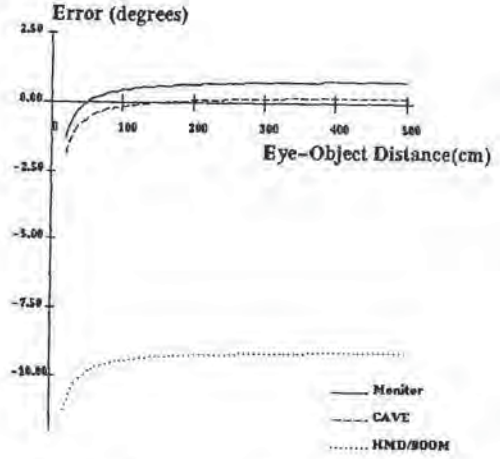



Figure 11: Tracking errors introduced by head panning

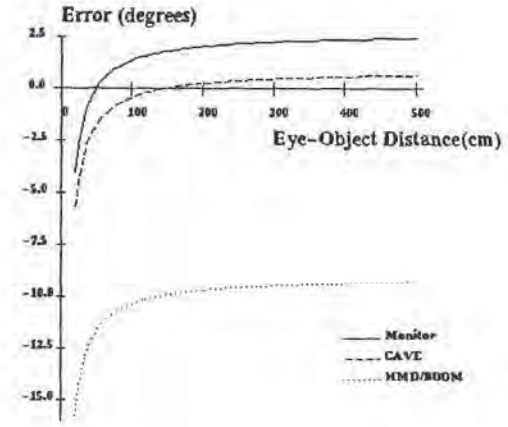



Figure 12: Tracking errors introduced by head nodding

Figure 12 graphs the angular error α as a function of eye/object distance Z due to a head rotation (nod) of 90 degrees/second and a display rate of 10 frames/second. It is assumed that the eyes are 15cm from the center of rotation. For large Z, the CAVE is 15 times better than the HMD/BOOM and 4 times better than the monitor. For small Z, the CAVE and monitor are 3 times better than the HMD/BOOM.

The examples above are all due to tracking errors caused by latency. Tracking errors from other sources, such as electrical interference, tend to be about an order of magnitude smaller, but the ratios are the same and we can draw the same conclusions. For the head-panning example in section 4.5, the problem was caused by normal head motion; if, however, we divide the angular error $\alpha$ by 20, we could interpret the graph as representing the case of a 0.5-degree tracking error combined with a tracking receiver mounted 5cm from the eye.

## 5. Successive Refinement

One benefit of the wrap-around screens in the CAVE is the potential for successive refinement of images. It is fair to say that we will never, in our lifetimes, have enough computing power to create complex models and display them in real time. Successive refinement trades off motion for time, freezing the image and filling it in, a now common computer graphics technique. Yet, one cannot freeze the image in a HMD without major disorientation. In the BOOM, successive refinement is possible but the user cannot look around. In the CAVE, one can navigate to a place in real time and then send off to a supercomputer for a highly detailed set of four images, still in stereo. When the images come back, the user can still pan around, although he/she cannot navigate while in this mode. The best stereo is achieved when looking in the last interactively tracked direction. Optimizing for this mode is the subject of active ongoing research.

Making VR usable in less-than-real-time situations is important. Supercomputers are essentially floating-point machines. One popular vector machine we use cannot create 1280x1024 pixel maps in real time because the floating-to-fixed conversions are done by non-vectorized subroutine calls (at three conversions, one for each pixel color component, it gets time consuming). There are no floating-point frame buffers for sale. In addition, the desire to transmit a 1280x1024 24-bit image to a workstation 60 times a second requires nearly 2 gigabits of network throughput! Multiply that by 4 for the CAVE screens. Since an update rate of only 10 times a second is closer to VR industry standards, divide by 6, which results in a need for 1.25 gigabits/second. Clearly, we try to transmit polygon lists and meshes in floating point and let the workstation's graphics engine do its job whenever possible.

Naturally, it is important to consider more than image complexity; the basic science being computed often is extremely complex and will not respond in real time. Sometimes large stores of precomputed data are meaningful to explore; perhaps disk-based playback will be useful. The CAVE is a research resource now being used by scientists at the University of Illinois at Chicago, the National Center for Supercomputing Applications, Argonne National Laboratory, University of Chicago, California Institute of Technology, and the University of Minnesota. The overall goal is to match the capabilities of supercomputing, high-speed networking, and the CAVE for scientific visualization applications.

## 6. CAVE Shortcomings

### 6.1. Cost

The CAVE is big and expensive, although, given inflation, it is no more expensive than the PDP-11/Evans & Sutherland single-user display system was 20 years ago. Also, considering that up to 12 people can space-share the CAVE, the cost per person comes down in some circumstances. Cheap wall-sized LCD screens with low latency that one could stand on would be great to have, if they only existed. The desire for the rendering afforded by $100,000 state-of-the-art graphics engines will not abate; however, current effects will be achievable at more modest cost as time goes on.

### 6.2. Ability to Project on All Six Sides of the CAVE

Six screens would make a better CAVE. We originally planned to do both floor and ceiling "rear" projections, which would have necessitated raising the CAVE structure 10'. A hole in the floor and a large sheet of strong glass or plastic would be a better solution, but not one easily achieved at conferences or universities.

A rear screen for the fourth wall might be possible, although the details for human entrance and exit would have to be worked out, especially if the cable-stretched screen technique were used. Four screens work very well, yielding large surround views for both panning actions and looking down. Consequently, objects inside the room can be walked around and virtually beg to be touched.

### 6.3. Light Spillage

One problem is the light spillage from the "screen" on the floor (the wall screens are fortunately not very reflective). Our floor screen is simply a painted floor board; the floor paint was quickly chosen by using the color-matching computer at the local paint distributor to duplicate the wall screens' color as a first approximation. The only time there would be a problem having one screen brighter than the others would be when the center of interest is not an object on the brightest screen, an unusual case. Very bright screens all around do tend to reduce image contrast somewhat, but this, too, has not been an issue. Naturally, good graphic design optimizes for the strengths and weaknesses of any medium.

### 6.4. Utilizing the CAVE Medium to Its Full Potential

The CAVE, like Omnimax, represents a different visual paradigm: inside out instead of outside in. From working with students and colleagues, we realize that getting people to design visualizations and think in terms of inside-out is difficult, especially since the CAVE simulator used in the early stages of application development has an outside-in presentation on the workstation screen. Nonetheless, it is a concept into which it is fairly easy to incorporate data.

### 6.5. Fragility

The CAVE is not "museum hardy." The screens, tracker, and glasses are not kid-proof, thereby limiting use in museums, malls, arcades, and so on. More research is needed.

### 6.6. New Control Paradigms

As the computing community went from command-line terminals to 2D raster systems, the pull-down menu and mouse provided an alternative to the command line and keyboard. The CAVE has not produced any significant new control paradigms to date, although "step-on" menus have been proposed. One graduate student (Randy Hudson) has achieved a nice way to control rotation by having the user stroke a barely perceptible tessellated wireframe sphere with his/her hand. We look forward to the challenge of finding the next control models and encourage anyone with ideas to come and discuss collaboration.

## 6.7. Directional Sound

Another issue to address is the effective implementation of directional sound. In theory, with speakers in all corners, one should be able to achieve good directionality with the proper audio synthesis gear. In practice, however, sound localization is compromised by reflections off the screens.

## 6.8. Ability to Document

The CAVE is very hard to photograph. Imaginations soar when readers are presented with excellent suggestive 2D photos of other VR devices in use. We have not been able to compete in this domain. However, the CAVE and monitor are both amenable to video documentation if the tracking device is attached to the camera and the interoccular distance is adjusted to zero.

## 7. Conclusions

The CAVE has proven to be an effective and convincing VR paradigm that widens the applicability and increases the quality of the virtual experience. The CAVE achieves the goals of producing a large angle of view, creating high-resolution (HDTV to twice HDTV) full-color images, allowing a multi-person (teacher/student or salesperson/client) presentation format, and permitting some usage of successive refinement. Furthermore, the flatness of the projection screens and the quality of geometric corrections available in projectors allow presentations of 3D stereo images with very low distortion as compared to monitor-based, HMD, and BOOM VR systems. The user is relatively unencumbered given that the required stereo glasses are lightweight and the wires to the head and hand trackers for the tracked individual are very thin. Since the projection plane does not rotate with the viewer, the CAVE has dramatically minimized error sensitivity due to rotational tracking noise and latency associated with head rotation, as compared to the HMD and BOOM.

At SIGGRAPH '92 and Supercomputing '92, more than a dozen scientists, in fields as diverse as neuroscience, astrophysics, superconductivity, molecular dynamics, computational fluid dynamics, fractals, and medical imaging, showed the potential of the CAVE for teaching and communicating research results. Collaborative projects are currently underway in non-Euclidean geometries, cosmology, meteorology, and parallel processing. The CAVE is proving itself a useful tool for scientific visualization, in keeping with our Laboratory's goal of providing scientists with visualization tools for scientific insight, discovery, and communication.

## 8. Future Work

Further research efforts will tie the CAVE into high-speed networks and supercomputers. We have interest in adding motion-control platforms and other highly tactile devices. Hardening and simplifying the CAVE's design for the nation's science museums, schools, and shopping malls is a goal as well. Design and implementation of quantitative experiments to measure CAVE performance are also planned.

## 9. References


[1] Bishop, G., Fuchs, H., et al. Research Directions in Virtual Environments. *Computer Graphics*, Vol. 26, 3, Aug. 1992, pp. 153–177.

[2] Brooks, F.P. Grasping Reality Through Illusion: Interactive Graphics serving Science. *Proc. SIGCHI '88*, May 1988, pp. 1-11.

[3] Cruz-Neira, C., Sandin, D.J., DeFanti, T.A., Kenyon, R., and Hart, J.C. The CAVE, Audio Visual Experience Automatic Virtual Environment. *Communications of the ACM*, June 1992, pp. 64-72.

[4] Codella, C., Jalili, R., Koved, L., Lewis, B., Ling, D.T., Lipscomb, J.S., Rabenhorst, D., Wang, C.P., Norton, A., Sweeny, P., and Turk, G. Interactive simulation in a multi-person virtual world. *ACM Human Factors in Computing Systems*, CHI '92 Conf., May 1992, pp. 329-334.

[5] Chung, J.C., Harris et al. Exploring Virtual Worlds with Head-Mounted Displays. *Proc. SPIE*, Vol. 1083-05, Feb.1990, pp. 42-52.

[6] Deering, M. High Resolution Virtual Reality. *Computer Graphics*, Vol. 26, 2, July 1992, pp.195-201.

[7] Fisher, S. The AMES Virtual Environment Workstation (VIEW). *SIGGRAPH '89, Course #29 Notes*, Aug. 1989.

[8] Max, N. SIGGRAPH'84 Call for Omnimax Films. *Computer Graphics*, Vol 16, 4, Dec. 1982, pp. 208-214.

[9] McDowall, I.E., Bolas, M., Pieper, S., Fisher, S.S. and Humphries, J. Implementation and Integration of a Counterbalanced CRT-based Stereoscopic Display for Interactive Viewpoint Control in Virtual Environments Applications. *Proc. SPIE*, Vol. 1256-16.

[10] Plato. *The Republic*. The Academy, Athens, c.375 BC.

[11] Rheingold, H. *Virtual Reality*. Summit, New York, 1991.

[12] Sutherland, I.E. The Ultimate Display. *Proc. IFIP* 65, 2, pp. 506-508, 582-583.

[13] Teitel, M.A. The Eyephone: A Head-Mounted Stereo Display. *Proc. SPIE*, Vol.1256-20, Feb. 1990, pp. 168-171.

[14] *Graphics Library Programming Guide*. Silicon Graphics, Inc. 1991.


## Acknowledgments

CAVE research is being conducted by the Electronic Visualization Laboratory of the University of Illinois at Chicago, with extraordinary support from Argonne National Laboratory and the National Center for Supercomputing Applications at the University of Illinois at Urbana-Champaign. Equipment support is provided by Ascension Technology Company, DataDisplay Corporation, Electrohome Projection Systems, Polhemus, Silicon Graphics Computer Systems, Stereographics Corporation, and Systran Corporation. Major funding is provided by the National Science Foundation (NSF) grant ASC-92113813, which includes support from the Defense Advanced Research Projects Agency and the National Institute for Mental Health, NSF grant IRI-9213822, and the Illinois Technology Challenge Grant.



# Painting with Light

*Chris Schoeneman*
*Julie Dorsey*
*Brian Smits*
*James Arvo*
*Donald Greenberg*

Program of Computer Graphics
Cornell University
Ithaca, NY 14853

## ABSTRACT

We present a new approach to lighting design for image synthesis. It is based on the *inverse problem* of determining light settings for an environment from a description of the desired solution. The method is useful for determining light intensities to achieve a desired effect in a computer simulation and can be used in conjunction with any rendering algorithm. Given a set of lights with fixed positions, we determine the light intensities and colors that most closely match the target image painted by the designer using a constrained least squares approach. We describe an interactive system that allows flexible input and display of the solution.

**CR Categories and Subject Descriptors:** I.3.7 **[Computer Graphics]**: Three-Dimensional Graphics and Realism; I.3.3 **[Computer Graphics]**: Picture/Image Generation; I.3.6 **[Computer Graphics]**: Methodology and Techniques - Interaction techniques.

**Additional Key Words:** simulation, global illumination, radiosity, ray tracing, lighting design, inverse problems.

## 1 INTRODUCTION

Although global illumination algorithms can produce strikingly realistic images, these algorithms can be difficult to use for lighting design. Currently the only tools available to designers are based upon *direct* methods—those that determine an image from a complete description of an environment and its lighting parameters. This forces a designer to begin with a geometric model, position the lights, assign their colors and intensity distributions, and finally compute a solution. The process is repeated until the solution matches the desired effect. This method is generally time-consuming, tedious, and often counter-intuitive. Given that we usually begin with a notion of the final appearance, a more natural, albeit more difficult, approach is to solve the *inverse problem*—that is, to allow the user to create a *target image* and have the algorithm work backwards to establish the lighting parameters. Inverse problems infer parameters of a system from observed or desired data [1]—in contrast with direct problems, which simulate the effects given all parameters. Although inverse problems are common in radiative transfer, thus far the field of computer graphics has been almost exclusively concerned with direct problems. Yet, inverse problems match a central goal of lighting design—determining how to achieve a desired effect.

In this paper, we present an approach that allows a designer to "paint" a scene as it is desired to appear. Given static geometry and a set of lights with fixed positions, a *constrained least squares* approach is used to determine the light intensities and colors that most closely match the target image painted by the designer. In the domain of lighting design, geometry often constrains the placement of the lights [2]; the designers frequently know about where to put the lights but not how the lights will combine or how bright to make them. Consequently, the task of selecting appropriate intensities for static lights is a useful subproblem of lighting design, and this is our focus. We do not address the automatic placement of lights, nor the mapping of simulated intensities to physical properties of the lights [3, 9].

## 2 INVERSE PROBLEM

The problem can be phrased more formally as follows: given static scene geometry and a desired appearance, determine the lights that will most closely match the target. There are constraints on possible solutions: only certain objects can emit light and only positive energy can be emitted—keeping us in the realm of physically meaningful solutions. The existence of constraints implies that not every target is realizable. The most general problem of determining how many lights to use, where the lights should be placed, as well as the distribution, color, and intensity of the lights is a non-linear optimization problem. However, if all possible lights have been positioned, and their distributions have been fixed, the determination of which lights to use and what their colors and intensities should be is a linear optimization problem.

### 2.1 Constrained Least Squares

Suppose $\{\Phi^1, \ldots, \Phi^n\}$ is the set of functions resulting from $n$ distinct light sources illuminating an environment independently. These functions can be computed by any illumination algorithm, including those that account for interreflection and shadows. For example, they may be ray traced images [10] of a scene for each light from the same viewpoint, or radiance functions over surfaces in the environment computed via radiosity [4]. Let $\Psi$ be the target function we wish to approximate. To formulate the approximation problem we require some minimal structure on the space of func-

 ACM-0-89791-601-8/93/008/0143 $01.50

tions. In particular, we require vector addition and scaling, which we define pointwise, as well as an inner product defined on pairs of functions (i.e. a symmetric positive definite bilinear form). From the inner product we gain the useful notion of the "size" of a function via the norm

$$||\Phi|| = \sqrt{\langle \Phi , \Phi \rangle}, \tag{1}$$

which provides a measure of error. The approximation problem can then be stated in terms of finding non-negative weights $w_1, \ldots, w_n$ such that the function

$$\widehat{\Psi} = \sum_{i=1}^{n} w_i \Phi^i \tag{2}$$

minimizes the objective function $||\Psi - \widehat{\Psi}||$. Stated in this way, the problem is one of least squares. Its unique solution is easily expressed in terms of the inner products:

$$\underbrace{\begin{bmatrix} \langle \Phi^1 , \Phi^1 \rangle & \cdots & \langle \Phi^1 , \Phi^n \rangle \\ \vdots & & \vdots \\ \langle \Phi^n , \Phi^1 \rangle & \cdots & \langle \Phi^n , \Phi^n \rangle \end{bmatrix}}_{M} \underbrace{\begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix}}_{w} = \underbrace{\begin{bmatrix} \langle \Phi^1 , \Psi \rangle \\ \vdots \\ \langle \Phi^n , \Psi \rangle \end{bmatrix}}_{b}. \tag{3}$$

The $n \times n$ matrix $M$ is the Gram matrix of the inner product, which consists of the coefficients of the normal equations [7]. The Gram matrix is non-singular if and only if the functions $\{\Phi^1, \ldots, \Phi^n\}$ are linearly independent, which will normally be the case if all $n$ lights produce distinct effects on the environment. Naturally, this excludes coincident light sources.

The remaining task is to define an appropriate inner product on the space of functions. Here we make use of the exact nature of the functions. If the functions assign intensities to a set of $p$ discrete points, such as images consisting of $p$ pixels, then the natural inner product is the $p$-dimensional vector dot product.

Alternatively, if the functions define surface radiance, the most natural inner product is the integral of the pointwise product of the functions. We further assume that the functions are piecewise linear, defined by interpolating a finite set of patch vertices. This representation is easily integrated yielding

$$\langle \Phi^i , \Phi^j \rangle = \sum_{k=1}^{v} \Phi_k^i \alpha_k^2 \Phi_k^j \tag{4}$$

where $v$ is the number of patch vertices, $\alpha_k$ is proportional to the sum of all patch areas adjacent to the $k^{th}$ vertex, and $\Phi_k^i$ is the radiosity at vertex $k$ due to light $i$. Under these assumptions, the *normal equations* can be written

$$A^T D A w = A^T D \Psi \tag{5}$$

where $A$ is the $v \times n$ matrix of the $n$ vectors $\Phi^i$, and $D$ is the $v \times v$ diagonal matrix $\text{diag}(\alpha_1^2, \ldots, \alpha_v^2)$ of the weights used for the inner product. With this definition, $||\Phi||$ is proportional to the total power leaving all surfaces. Also, changes to the inner product are easily expressed as changes to $D$.

### 2.2 Solving the Normal Equations

The problem now is to solve the system of equations from Equation 5. This system contains $n$ equations in $n$ unknowns where $n$, the number of lights, is generally much smaller than the number of vertices in the environment or pixels in the image. Let $M = A^T D A$ and $b = A^T D \Psi$ as in Equation 3. We chose to solve the system $Mw = b$ using a modified Gauss-Seidel iteration.

There is no guarantee that the solution to the system has only positive entries. Simply clipping to zero after convergence is not a viable approach because negative values counteract some of the positive energy; ignoring them causes the environment to be too bright. To avoid this difficulty, we modify the Gauss-Seidel algorithm so that negative values are clipped to zero during each iteration. On the $k+1$ iteration of the modified algorithm, the updated value of $w_i$ is

$$w_i^{(k+1)} = \max\left( \frac{b_i - \sum_{j=0}^{i-1} M_{ij} w_j^{(k+1)} - \sum_{j=i+1}^{n} M_{ij} w_j^{(k)}}{M_{ii}} , 0 \right). \tag{6}$$

Since a zero value does not influence other entries of $w$, we are effectively ignoring that light while the iteration is producing a negative value for it. In practice, this approach always converges in the sense that the difference between two iterations goes to zero. An alternative method may be found in [6].

## 3 IMPLEMENTATION

Our implementation is based on surface radiance functions as opposed to images. The system is therefore view-independent, solving for light intensities that are meaningful in a global sense, not simply for a given view. Although the system does no automatic placement of lights, the user may modify light source positions and distributions at any time. However, any such change requires that a new solution $\Phi_i$ be computed. To keep these operations fast, we have currently limited the solutions to direct illumination from each of the lights, accounting for distance and visibility but not secondary reflections. Similarly we restrict surfaces to be ideal diffuse reflectors. Using more complex techniques to find the light source functions makes moving a light more expensive, but does not affect the algorithm. By solving for the intensity of each color channel separately, the colors are determined as well as the intensities.

The user modifies the radiance function of the target by "painting" light onto surfaces. We also adjust the matrix $D$ so that painted surfaces have more weight (or more area) in the solution, causing the system to try harder to match painted surfaces than unpainted ones. This is necessary in complex environments where the large unpainted areas can overwhelm the effect of small painted areas.

To achieve interactive speeds while painting we use the method introduced by Hanrahan and Haeberli [5] to quickly find which patch the brush is currently affecting. Object id's and the patch $uv$ coordinates are rendered into auxiliary buffers. A lookup at the paint brush position in these buffers quickly identifies the patch being painted. Only painted patches are redrawn. Since very few patches change at once, updates are easily made in real time.

The patch's reflectance function modifies the light as it gets painted on a surface. This prevents a surface from being painted with physically unattainable colors. For example, a purely red surface cannot be painted blue. The modified light then gets distributed to the patch's vertices according to their proximity to the paint brush. We restrict the radiosity at a vertex to between zero and one and linearly map this to the full dynamic range of the display.

The system recomputes the closest fitting combination of lights after each brush stroke. All vertices painted between a button press and release comprise a stroke. To maintain interactivity, we perform all the updates incrementally. Instead of completely rebuilding $\Psi$

(the target radiosities) and re-solving, however, we only change the elements corresponding to painted vertices and make incremental changes to the inner products. If $\Delta\Psi$ is a vector of the changes to the radiosities with $p$ non-zero terms, then

$$b_{new} = A^T D(\Psi + \Delta\Psi) = b_{old} + A^T D\Delta\Psi. \quad (7)$$

Since $\Delta\Psi$ is typically very sparse, we can update $b$ with $O(np)$ operations by ignoring all zero entries of $\Delta\Psi$. Since most of the environment hasn't changed, the old intensities provide a good initial guess for the modified Gauss-Seidel iteration and it converges quickly. We can similarly update the weight (i.e. effective area) of vertices. Consider changing the importance of one vertex. Let $\Delta D$ be the diagonal matrix with its sole non-zero entry being the change in weight of the vertex. Then

$$b_{new} = A^T(D + \Delta D)\Psi = b_{old} + A^T\Delta D\Psi. \quad (8)$$

Because $\Delta D$ has only one non-zero entry, $\Delta D\Psi$ has only one non-zero entry and $b_{old}$ can be updated with $O(n)$ operations. Changing the inner product, though, requires that $M$ be updated as well. This can be done incrementally, observing that

$$M_{new} = A^T(D + \Delta D)A = M_{old} + A^T\Delta DA. \quad (9)$$

Since $\Delta DA$ has only one non-zero row, we need to look at only one column of $A^T$ so we can do the multiplication in $O(n^2)$ steps.

In addition to painting, the user can also interactively move and aim light sources. Changing a light requires recomputing the direct illumination due to that light. Since $A$ changes, $M$ must be recomputed as well; however the cost of recomputing a column of $A$ greatly overshadows the matrix multiply used to determine $M$. Because this can take time for large environments, the user can defer these computations until all the lights have been satisfactorily placed.

The user may also move the camera interactively. Because we paint directly onto the geometry, painted surfaces are view-independent. Also, since no directional effects are accounted for, the functions $\Phi^j$ for each light are independent of the position and orientation of the camera. Therefore we need not recompute $A = [\Phi^1 \cdots \Phi^n]$ or re-solve for the light intensities as a result of moving the camera.

## 4 RESULTS

We tested the system on a moderately complex environment consisting of polygonal meshes with about 19,000 polygons, 27,000 vertices, and 12 lights. Figure 1 shows the user's painted environment at the top and the system's solution on the bottom. A user can see both views at once while working to get immediate feedback on how closely the design is being met. Figure 2 shows the same environment with the same light positions but with different painted intensities and colors (left) and a distinct best approximation (middle). The lighting parameters determined by the interactive lighting design were then used to compute a ray traced solution, which is shown in Figure 2 (right). The large scale washes of color and illumination levels are captured well in the rendered image. The user can quickly and easily modify a design to have a very different appearance.

Figure 3 shows the screen during a painting session. The window in which the user paints is on the left and the best fit solution is on the right. Some of the support tools for choosing light to paint and positioning lights are also shown. In this design, 14 lights were placed in another environment of similar complexity.

Figure 1: Design (top) and associated best approximation (bottom).

## 5 CONCLUSIONS AND FUTURE WORK

We have created an interactive system to help with lighting design in image synthesis by solving a restricted inverse lighting problem. The user paints an approximation of the desired result and the system computes light intensities and colors to match it. This approach can be more intuitive and easier to use than the usual direct edit–render cycle.

Given fixed geometry and a desired target, the problem of determining light intensities and colors can be solved in the least squares sense using a modified Gauss-Seidel algorithm. The method can be made more interactive by using incremental updates to the matrices and vectors involved in the solution process. Magnifying the effect of each brush stroke by increasing the weight of the affected vertices allows the user to make changes to the environment with relatively little effort.

Although they have received little attention in computer graphics, inverse lighting algorithms have great potential as design tools. Clearly there is much to do beyond automatic selection of light source intensities. Automatic light source placement would greatly increase the utility of the technique, but will require more elabo-

Figure 2: Design (left); best approximation (middle); ray tracing (right).

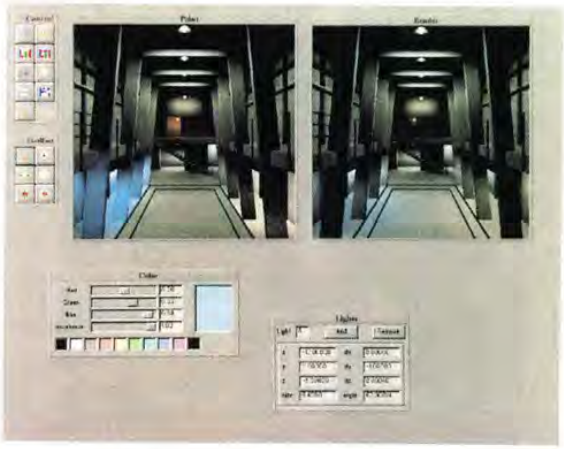


Figure 3: Interactive system.

rate optimization methods, as this requires solving non-linear constrained optimization problems.

Any rendering technique will work for determining the contributions from each of the lights. Our use of direct illumination only was motivated by a desire to allow interactive light placement. A more elaborate implementation might compute more accurate solutions for those lights that were unlikely to change position or distribution.

In order to make the system usable for lighting designers, some way of mapping screen intensities to physical units in the system must be found. Since the system is being driven by the user's perception of what is being painted, the lighting conditions of the user's environment must be accounted for, as well as the non-linearities of the monitor, the reproduction of color on the monitor, and most importantly, the extremely limited dynamic range of the monitor.

## ACKNOWLEDGEMENTS

We would like to thank Jed Lengyel for his helpful comments and Kurk Dorsey and Suzanne Smits for their help assembling the paper. Much thanks to Matthew Bannister who created the model and the lighting designs. This work was supported by the NSF grant "Interactive Computer Graphics Input and Display Techniques" (CCR-8617880), and by the NSF/DARPA Science and Technology Center for Computer Graphics and Scientific Visualization (ASC-8920219). The authors gratefully acknowledge the generous equipment grant from Hewlett Packard Corporation on whose workstations the research was conducted.

## REFERENCES

[1] Baltes, H. P., editor. *Inverse Source Problems in Optics*, Springer-Verlag, New York, 1978.

[2] Dorsey, Julie O'B., François X. Sillion, and Donald P. Greenberg. "Design and Simulation of Opera Lighting and Projection Effects," in *Computer Graphics*, 25(4), August 1991, pages 41–50.

[3] Evans, Ralph M. *Eye, Film, and Camera in Color Photography*, John Wiley & Sons, New York, 1959.

[4] Goral, Cindy M., Kenneth E. Torrance, Donald P. Greenberg, and Bennett Battaile. "Modeling the Interaction of Light Between Diffuse Surfaces," in *Computer Graphics*, 18(3), July 1984, pages 213–222.

[5] Hanrahan, Pat and Paul Haeberli. "Direct WYSIWYG Painting and Texturing on 3D Shapes," in *Computer Graphics*, 24(4), August 1990, pages 215–223.

[6] Lawson, Charles L. and Hanson Richard J. *Solving Least Squares Problems*, Prentice-Hall, Englewood Cliffs, 1974.

[7] Luenberger, David G. *Optimization by Vector Space Methods*, John Wiley & Sons, New York, 1969.

[8] Poulin, Pierre and Alain Fournier. "Lights from Highlights and Shadows," Proceedings of the 1992 Symposium on Interactive 3D Graphics, in *Computer Graphics*, April 1992, pages 31–38.

[9] Tumblin, Jack and Holly Rushmeier. "Tone Reproduction for Realistic Computer Generated Images," in Radiosity Course Notes of SIGGRAPH'91, ACM, August 1991, pages 229–257.

[10] Whitted, Turner. "An Improved Illumination Model for Shaded Display," *CACM*, 32(6), June 1980, pages 343–349.



# Radioptimization — Goal Based Rendering

John K. Kawai
Department of Computer Science
University of Utah

James S. Painter
Department of Computer Science
University of Utah

Michael F. Cohen
Department of Computer Science
Princeton University

## Abstract

This paper presents a method for *designing* the illumination in an environment using optimization techniques applied to a radiosity based image synthesis system. An optimization of lighting parameters is performed based on user specified constraints and objectives for the illumination of the environment. The *Radioptimization* system solves for the "best" possible settings for: light source emissivities, element reflectivities, and spotlight directionality parameters so that the design goals, such as to minimize energy or to give the room an impression of "privacy", are met. The system absorbs much of the burden for searching the design space allowing the user to focus on the goals of the illumination design rather than the intricate details of a complete lighting specification.

The system employs an object space perceptual model based on work by Tumblin and Rushmeier to account for psychophysical effects such as subjective brightness and the visual adaptation level of a viewer. This provides a higher fidelity when comparing the illumination in a computer simulated environment against what would be viewed in the "real" world. Optimization criteria are based on subjective impressions of illumination with qualities such as "pleasantness", and "privateness". The qualities were selected based on Flynn's work in illuminating engineering. These criteria were applied to the radiosity context through an experiment conducted with subjects viewing rendered images, and the respondents evaluated with a Multi-Dimensional Scaling analysis.

## 1 Introduction

Historically, lighting design has been a black art. The lighting designer first received a design specification of the customer's expectations and of the room's function. The designer then made a lighting lay out and from experience would sketch what the room would look like from rough lighting calculations. With the advent of computer aided rendering, this process has been simplified allowing the designer to model lighting specifications with a CAD system and have it simulate the lighting calculations giving the designer a quick design check of what the room would look like. This also provides the customer who has no experience with lighting units a realistic preview of the finished room early in the design cycle [16]. Progress in rendering to date has mainly focused on improving the realism of the physical simulation and the development of algorithms with faster performance. Although great advances have been made in these areas, little work has been done on addressing the design problems in creating better quality lighting, except for a few systems that determine lighting placement by indicating desired areas of highlights and/or shadow [12].

Lighting designers base their art on the belief that spatial lighting patterns are a visual communicative medium, in which some patterns of light suggest or reinforce shared attitudes and impressions to people of the same cultural background [5]. In addition, the designer must be aware of the need to conserve the electrical energy used in implementing their designs. An over-reaction to the wasteful energy consumption of the 1960s and 1970s often led to buildings which were inadequately lit for their designed purposes, hampering the productivity of the residents. A better balance of goals between energy conservation and the quality of the lighting is needed [10].

This paper proposes a goal based illumination design approach, that has been termed *Radioptimization*, to help a lighting designer search the space of possible lighting specifications. Though computers will never replace artists, the system may generate configurations not previous considered or optimize on an already considered configuration. The approach allows the designer to concentrate on high level goals such as "visual clarity" and specify constraints such as minimum lighting levels in specific locations. The system then determines optimal settings for the lighting parameters of the modeled environment by searching for the "best" possible settings for light source emissivities, surface reflectivities, and spotlight directionality. Unconstrained optimization techniques are employed in conjunction with classical radiosity [3, 2, 8] to simulate global illumination.

Creating an appropriate two-way link between the designer and the rendering system requires two important enhancements to basic rendering methods. First, since the designer is asked to iteratively evaluate the visual impression from a rendered image, the images must provide (as much as possible) a subjective match to a "real" environment. The work of Tumblin and Rushmeier [18] on the psycho-physical quantities of subjective brightness has been applied to map luminance values to brightness values to provide higher fidelity for comparing the illumination of a computer generated scene.

Secondly, the optimization objectives presented to the de-



© 1993 ACM-0-89791-601-8/93/008/0147 $01.50

signer are based on John Flynn's work [5] whose experiments allowed one to measure impressions of lighting patterns. To develop the objective functions, experiments were conducted with subjects viewing computer generated images to create a mapping from Flynn's criteria to quantifiable qualities in the radiosity simulations.

There are three bodies of technology and related literature that are central to the work reported here: numerical optimization, radiosity based image synthesis, and knowledge about human perception as it relates to subjective impressions of lighting and to subjective impressions from images presented on a CRT. We will briefly review each of these areas concentrating on the pertinent subtopics in each that relate directly to this work.

## 1.1 Optimization

The basic constrained optimization problem is to minimize the scalar quantity of an objective function of $n$ system parameters while satisfying a set of constraints. Although this is a well researched area, to date there is no computational algorithm for optimization which will always find the global minimum of a general non-linear objective function.

Most methods for dealing with constraints transform the constrained problem to an (approximately) equivalent unconstrained optimization by either removing the constraints by explicitly solving for one optimization variable, or by adding a new function to the objective [14, 15]. In the simplest case a constraint can be transformed into a *penalty* function, which when added to the objective returns a high value on a constraint violation.

Once the constraints are removed or transformed, the problem reduces to finding a minimum of the objective. Most optimization methods are performed iteratively from a starting point, in the multidimensional search space. Local information about the value, gradient, and Hessian (matrix of second order partial derivatives) of the function is gathered and a search direction is selected to move the solution to a new guess. One such technique for selecting a search direction is Newton's Method which solves for a step direction as the inverse of the Hessian times the negative gradient, i.e, $\Delta X = -(\nabla^2 f)^{-1} \cdot \nabla f$.

Although Newton's method can have great success, a number of Quasi-Newton methods have been developed to numerically approximate the Hessian from a series of gradients for applications where it is either inefficient or impossible to derive the Hessian directly. These include the Broyden-Fletcher-Goldfarb-Shanno (BFGS) method [13, 11], which due to non-linearities does a series of one dimensional line searches until it converges on a local minimum.

## 1.2 Radiosity

*Radiosity* methods simulate the illumination of Lambertian diffuse environments by deriving an energy balance equation. Discretizing the environment into a set of elements with an assumed functional form, typically a constant value, for the radiosity across the surface, the balance of energy between elements is defined as through a set of interdependent linear constraints in the form:

$$B_i = E_i + \rho_i \sum_{j=1}^{n} F_{i,j} B_j \qquad (1)$$

where $B_i$ is the radiosity of element $i$, $E_i$ is the emission of element $i$, $\rho_i$ is the reflectivity of element $i$, and $F_{i,j}$ is the form factor from element $i$ to element $j$.

The form factor is the fraction of light leaving one element ($i$) that arrives at another ($j$) and is given by:

$$F_{i,j} = \frac{1}{A_i} \int_{p_i \in A_i} \int_{p_j \in A_j} \delta(p_i, p_j) \frac{cos(\phi_i) cos(\phi_j)}{\pi r_{ij}^2} dA_i dA_j$$

where $A_i$ and $A_j$ are the area of the element surfaces, $p_i$ and $p_j$ are points on elements $i$ and $j$ respectively, $\delta(p_i, p_j)$ returns 1 if $p_i$ and $p_j$ are mutually visible and 0 otherwise, $\phi_i$ is the angle between the normal vector at $p_i$ and the vector from $p_i$ to $p_j$, $\phi_j$ is the angle between the normal vector at $p_j$ and the vector from $p_j$ to $p_i$, and $r_{ij}$ is the distance from $p_i$ to $p_j$. For an environment of $n$ patches, equaton 1 can be expressed as a set of $n$ simultaneous linear equations.

This system of equations can be solved numerically by "gathering" or "shooting" methods [3, 2]. The solution to this system yields the element radiosities, $B_i$, which can be projected from any view point onto the view plane for a final image. At first glance a direct solution to the radiosity equation appears to require at least $O(n^2)$ space and time, given $n$ elements. Hanrahan *et al.* have shown, however, that an equivalent to the form factor matrix can be computed and stored in $O(n)$ space and time by exploiting the cohenerent structure of the matrix [8].

Directional lighting effects such as spotlights can be added to the radiosity equation by replacing the $cos(\phi_i)$ term in the form factor equation with a different distribution function:

$$F_{i,j} = \frac{1}{A_i} \int_{p_i \in A_i} \int_{p_j \in A_j} \delta(p_i, p_j) s(\phi_i) \frac{cos(\phi_j)}{\pi r_{ij}^2} dA_i dA_j$$

where $s(\phi_i)$ is the directionality distribution weight for the light source as a function of the angle between the direction vector of the light (element $i$) and the vector between the points $p_i$ and $p_j$. Here we restrict ourselves to distributions of the form, $s_n(\phi) = w(n) cos^n(\phi)$ for values $n >= 1$. It is useful to be able to change the beam width without affecting the total energy emitted by the light. This requires a normalization factor, $w(n)$, in the emission function $s_n$. The normalization factor $w(n)$ must be chosen so that the total energy emitted over the hemisphere is constant, independent of $n$, as the beam width is adjusted. The value of the constant is chosen so that $w(1) = 1$. That is, $\int_{\text{hemisphere}} s_n d\omega = \pi$, where $d\omega$ is the differential solid angle on the sphere. Carrying out the integration in spherical coordinates yields the normalization weight, $w(n) = (n+1)/2$.

## 1.3 Human Perception

### 1.3.1 Brightness

*Brightness* is a measure of the subjective sensation produced by visible light. Brightness, measured in units of brils, relates linearly to human visual response. For example, if two light sources are compared and one appears to be twice as bright as the other, the brightness of the first, in brils, will be twice that of the second.

The human eye is sensitive to a luminance range of approximately ten orders of magnitude. However, at any one time the eye can only detect a brightness range of 100 to 1 with good accuracy. The iris adjusts, limiting the amount of light entering the eye, in order to seek a state of equilibrium that is appropriate for the general brightness conditions. Tumblin and Rushmeier [18] studied work by Stevens [17] who theorized that the adaptation level of a scene can be estimated by the expected value (mean)

of the $log_{10}$ of the luminances visible on the retina, i.e., $\mathrm{EXP}_{p \in \mathrm{retina}}\{log_{10}(L(p))\}$ where $L(p)$ is the luminance at a point $p$ on the retina. Miller *et al.* also theorized that differing adaptations of the eye result in a family of curves relating luminance and brightness values in the form, $log_{10}(P) = aa * log_{10}(L) + bb$ where $P$ is the brightness value specified in *brils*, $L$ is the luminance value specified in *nits*, $bb$ is $-0.4(log_{10}(L_w))^2 + (-2.58log_{10}(L_w)) + 2.02$, $aa$ is $0.4log_{10}(L_w) + 2.92$, and $L_w$ is the white adapting luminance which can be approximated by the equation $log_{10}(L_w) = \mathrm{EXP}\{log_{10}(L_i)\} + 0.84$.

This perceptual model accepts luminance values in units of *nits* which in photometric units are related to *lux* on a diffuse surface by, 1 lux = 1 nit / 10,000. Thus solving for brils in terms of an element radiosity of $B$ lux yields:

$$P = 10^{aa * log_{10}(B/10{,}000) + bb} \qquad (2)$$

Since the adaptation of the eye is affected only by what is visible to the retina, perceptual processing is usually done as a view dependent process in screen space. This assumes that the viewer adapts to a single view rather than to an entire environment. In practice, we are constantly moving our head and eyes to scan a room and hence adapt to the overall room lighting rather than to a single view. In our work we propose a view independent approach to lighting design, since the designer's goal is to optimize on the overall impression of a room rather than a particular view of the room. Therefore, the conversion from luminance units into perceptual units is performed in object space. Each element is considered to contribute to the adaptation proportional to its physical size. This neglects the view dependent effects of perspective foreshortening and occlusion but has the advantage that it yields view independent results. We have found that the object space, view independent, method gives results that are nearly identical to view dependent screen space methods for typical, single room, architectural models. In addition to the view independence, calculating perception in object space has the added advantage of faster performance if the number of elements is much smaller than the number of screen pixels.

#### 1.3.2 Subjective Impressions of Illumination

In the 1970's, John Flynn published a series of articles [6, 4, 5], introducing a methodology with which to quantify parameters that elicit a shared human behavioral response and subjective impression. In particular, Flynn examined how non-uniform, peripheral, and bright lighting affects impressions of visual clarity, spaciousness, relaxation, and privacy. Flynn created six different light settings for a conference room and subjectively associated each room with a non-uniform, peripheral, and brightness value so that each room corresponded to a point in a 3 dimensional space of the different lighting characteristics. Flynn also associated a set of semantic differential (SD) rating scales such as large-small and spacious-cramped with each category of impression. Test subjects were then asked to make pair wise comparisons of the differences between each room from the set of SD rating scales where 0 meant no difference and 10 meant a large difference.

The data gathered resulted in a 6x6 symmetric dissimilarity matrix comparing the 6 rooms for each subject tested and each SD comparison made, *e.g.* large-small. The multidimensional scaling program INDSCAL [1, 7], was used to determine how each subjected weighted the non-uniformity, peripheral and brightness values in making each SD comparison. A weighting of each dimension for each subject was determined that best fit the data. The results showed a correlation between the room positions hypothesized by Flynn and the positions computed by INDSCAL, supporting Flynn's hypothesis that brightness, non-uniformity, and peripheral lighting reinforce particular impressions. In addition, there also was a correlation for the weights for each parameter among all the subjects, supporting the concept that particular lighting patterns elicit a shared impression. By this process, Flynn was not only able to demonstrate that there is a definite correlation between the measurable quantities (non-uniform, peripheral, and bright lighting) and the subjective impressions (visual clarity, spaciousness, and relaxation), but was able to quantify how much each of the measurable dimensions affects each subjective impression.

As described shortly, we have adapted this work through an additional level of experimentation in which subjects reported impressions from computer generated images.

## 2 Problem Formulation

To pose the illumination design task as a constrained optimization problem we must identify: the variables involved in the optimization process, the constraints that must be satisfied, and the objective function.

### 2.1 Optimization Variables

In a normal radiosity based renderer, the element radiosities $B_i$ are the unknowns to be computed in terms of fixed material and light property parameters. In the optimization setting the material and light properties are no longer fixed and must also be considered as variables. Constraints may be imposed on any of these variables and the objective function may involve any or all of them.

In the illumination design problem the optimization variables are light source specification parameters (emissions, spotlight directions, spotlight focus), element radiosities, $B_i$, and element reflectivities, $\rho_i$. Two types of light sources are considered: diffusely emitting elements described by a single emissivity parameter $E_i$, and directional lights idealized as spotlights described by a position, direction, and a $cos^n$ directional distribution. Light source positions are assumed to be fixed and only the direction and distribution pattern is allowed to change during optimization.

Every light source emission $E_i$, light direction vector $V_i$, cosine distribution exponent $n_i$, element radiosity $B_i$ and reflectivity $\rho_i$, has the potential to be a variable in the optimization problem. If all are treated explicitly as domain variables in the optimization an intractably large system will result. Fortunately, the $B_i$'s can be eliminated by direct substitution of the radiosity equation, and typically only a small number of the elements will have variable emission, reflectivity or directionality parameters. These remaining variables are called the "free" variables of the optimization problem.

### 2.2 Constraints

Constraints fall into three categories.

*Physical constraints* specify the relationships between light emission and element radiosities that are dictated by the physics of light transport. The constraints are captured in the rendering equation [9]. We assume perfect diffuse surfaces and a discretized environment yielding the radiosity approximation given in equation 1.



149

*Design goals* are constraints provided by the user. These may be either equality or inequality constraints and may apply to a single element, or a conglomeration of elements. For example, the requirement that a particular element's radiosity is a given constant, $B_i = K$ for some constant $K$ is an equality constraint on a single element that expresses a fixed radiosity for the element. Inequality constraints such as $K_{low} \leq B_i \leq K_{high}$ can also be specified (in essence two inequality constraints) requiring the radiosity of element $i$ to stay within the bounds $K_{low}$ and $K_{high}$.

*Barrier constraints* are hard bounds on the allowable ranges of the optimization variables that must be satisfied to insure that the model is physically realizable. For example, light emissions must remain positive and element reflectivities must remain in the range $0 <= \rho_i <= 1$. Barrier constraints are conceptually similar to inequality design goals. The main difference is that a barrier constraint *must* be satisfied in order to produce a valid model. Design goals are *desires* that need not be satisfied exactly.

## 2.3 Objective Function

In general, radioptimization problems are underconstrained. There may be an infinite number of possible solutions that satisfy the problem constraints. The *objective* function is used to select between the many possible solutions. The simplest, directly measurable objective is the minimization of energy, $f_{energy} = \sum_i B_i A_i$.

In theory, any user specified function of the optimization variables could be used as an objective function. An alternative is to provide a fixed library of objective functions and allow the user to construct an objective function via linear combinations of the library functions. Each individual objective function in the library has a well defined and intuitive behavior. The user can then control the weights of the individual objectives to determine the final objective function. This allows user control without an undo amount of complexity.

A variation of Flynn's work, described in the previous section, was used to develop a way of quantifying subjective impressions. Flynn's experiment was duplicated except, instead of having the subjects judge actual rooms with different lighting characteristics, they were shown rendered images of an identical room with different light patterns (see figure 4). Once the data set was collected, it was processed by INDSCAL with the brightness, non-uniform, and peripheral values for each room computed by the following functions:

$$f_{brightness}(P,A) = -\frac{\sum_{i\in\chi} P_i A_i}{\sum_{i\in\chi} A_i}$$

$$f_{non-uniform}(P,A) = -\left(\frac{\sum_{i\in\psi}(P_{avg,i}-P_i)^2 A_i}{\sum_{i\in\psi} A_i)}\right)^{\frac{1}{2}}$$

$$f_{peripheral}(P,A) = \frac{\sum_{i\in\mu} P_i A_i}{\sum_{i\in\mu} A_i} - \frac{\sum_{i\in\psi} P_i A_i}{\sum_{i\in\psi} A_i}$$

where $\chi$ is the set of all elements in the environment, $\psi$ is the set of elements that make up the walls, $\mu$ is the set of all horizontally oriented elements, $P_i$ is the brightness of element i, $A_i$ is the area of element i, and $P_{avg,i}$ is the average brightness of the elements around element i. The functions are defined in terms of perceptual values because humans subjectively quantify illumination by brightness not by actual luminance.

The results from INDSCAL showed that there was a correlation among those tested in the relationship between the measurable quantities, brightness, non-uniform, and peripheral lighting, and the subjective impressions of visual clarity, privacy, and pleasantness. A linear transformation was fit to the INDSCAL data resulting in linear relationships between the subjective impressions and the measured values:

$$\begin{aligned} f_{clear} &= 0.90 \cdot f_{brightness} - 0.38 \cdot f_{non-uniform} - \\ &\quad 0.58 \cdot f_{peripheral} \\ f_{pleasant} &= 0.78 \cdot f_{brightness} - 0.53 \cdot f_{non-uniform} + \\ &\quad 0.24 \cdot f_{peripheral} \\ f_{private} &= 0.90 \cdot f_{brightness} + 0.32 \cdot f_{non-uniform} - \\ &\quad 0.09 \cdot f_{peripheral} \end{aligned}$$

## 2.4 Conversion of the Constrained Problem to an Unconstrained Problem

The design goal constraints can be included in the objective function through the *penalty method* [11] by penalizing deviations from constraints through explicit terms in the objective function. The penalty imposed on the objective is defined as the square of the constraint violation. For example, if the $j^{th}$ constraint, $C_j$, is an equality constraint specifying a particular radiosity[1] to be a given constant, $(B_{i_j} = K_j)$, this will result in a penalty term $f_{C_j}$ in the cost function given by $f_{C_j} = A_{i_j}(K_j - B_{i_j})^2$. Inequality constraints can be handled through a penalty function that "turns on" when the constraint is not satisfied. For example, the inequality constraint $C_j$ given by $(B_{i_j} < K_j)$ results in a penalty term $f_{C_j} = A_{i_j}(K_j - B_{i_j})^2$ when $B_{i_j}$ is greater than $K_j$ and is zero otherwise.

Barrier Constraints are handled in a similar fashion to impose hard physical restrictions on certain values, for example, the emission variables must always remain positive. Similarly, reflectivities must remain between 0 and 1. A barrier term is added to the objective function for each barrier constraint to avoid violations of these constraints. The barrier constraint $G_j$ given by $(X_j > K_j)$ for some free variable $X_j$ results in a barrier term $f_{G_j} = (X_j - K_j)^{-4}$ for $X_j > K_j$. In addition, the optimization search explicitly enforces the constraint $(X_j > K_j)$ by clamping the $X_j$ to $K_j + \epsilon$ when $X_j$ drops below $K_j$, where $\epsilon$ is a small positive constant. This will yield a large barrier term in the objective function tending to lead the search away from the barrier in the next iteration.

The remaining constraints are the "physical constraints" specified by the radiosity equation (equation 1). These are dealt with by direct substitution. The radiosity equation implicitly defines each $B_i$ in terms of all the $E, \mathbf{V}, n$ and $\rho$'s. The $B_i$'s are calculated via a radiosity solution algorithm [8]. The values for the $P_i$'s can then be computed directly from the $B_i$'s by equation 2. The $B_i$ and $P_i$ values can be directly substituted into the objective function. This effectively eliminates all the $B_i$'s and $P_i$'s from the set of optimization domain variables.

Thus the modified optimization problem is given by:

[1] $B_{i_j}$ indicates the radiosity of the $i^{th}$ element, where $i$ was selected by the $j^{th}$ constraint, $C_j$.

$$f(\mathbf{X}) = \begin{array}{lll} W_{energy} & f_{energy} & + \\ W_{brightness} & f_{brightness} & + \\ W_{non-uniform} & f_{non-uniform} & + \\ W_{peripheral} & f_{peripheral} & + \\ W_{clear} & f_{clear} & + \\ W_{pleasant} & f_{pleasant} & + \\ W_{private} & f_{private} & + \\ W_{designgoals} & \sum_j f_{C_j} & + \\ & \sum_j f_{G_j} & \end{array} \quad (3)$$

where $\mathbf{X}$ is a point in the multidimensional space spanned by the remaining free variables, $E_i$, $\mathbf{V}_i$, $n_i$, and $\rho_i$.

Through the use of the penalty method, barrier functions, and substitution of physics constraints, the optimization problem can now be stated as a simple unconstrained, multidimensional minimization problem. Let $\mathbf{X}$ be a multidimensional vector in the "design space", the space spanned by the free variables in the design. We must identify a point in the design space, $\mathbf{X}^*$, such that the objective function $f(\mathbf{X}^*)$ is (at least locally) minimized. There are many solution methods for such a minimization problem. We use the well known BFGS method described above [13].

## 3 Implementation

The user provides an initial model that is rendered to provide a baseline rendering. The user can select elements interactively from an image generated from the baseline solution to specify the free variables in the optimization process. The user can also specify the objective function weights $W_{energy}, W_{brightness}, W_{non-uniform}$, etc. to direct the optimization process. After all the design goals and objective weights are specified, the optimization process is run until convergence is achieved.

This process can be described in Pseudo code by:

*Compute baseline rendering.*
*Establish constraints and objectives.*
*REPEAT*
    *Evaluate partial derivatives.*
    *Compute search direction $\Delta\mathbf{X}$ using BFGS.*
    *Perform line search in the direction $\Delta\mathbf{X}$.*
    *Display results, and allow user to modify constraints and objectives*
*UNTIL convergence.*

### 3.1 Baseline rendering

The initial model is rendered and displayed by the hierarchical radiosity solution algorithm of Hanrahan *et al.* [8]. During baseline rendering, the input model is subdivided into a hierarchical structure and links are established between nodes in the hierarchy to establish the block structured form factor matrix as described in [8].

### 3.2 Establishing Constraints and Objectives

Once an image is displayed the user can select elements directly from the screen with the mouse and set constraints via the user interface shown in figure 5. In this example, the desk top has been selected as indicated by the green outline. Current illumination information for the selected element is displayed in the lower right corner of the interface. Through a set of buttons in the interface, the user can elect to impose a constraint on the element radiosity, and/or specify that the element reflectivity or emission should be a free variable in the optimization process. Spot lights are handled with a similar interface that allows the light direction vector and/or distribution parameter $n$ to be marked as free variables in the optimization. The objective function weights can also be adjusted with slider bars in this interface.

### 3.3 Partial Derivative Estimation

Evaluation of partial derivatives of the modified objective with respect to each free variables is required by the optimization process. For example, to compute the partial derivative of the objective function with respect to a light emission, $E_k$, we must evaluate:

$$\partial f/\partial E_k = \begin{array}{lll} W_{energy} & \sum_j \partial E_j/\partial E_k \; A_j & + \\ W_{brightness} & \partial f_{brightness}/\partial E_k & + \\ W_{non-uniform} & \partial f_{non-uniform}/\partial E_k & + \\ W_{peripheral} & \partial f_{peripheral}/\partial E_k & + \\ W_{clear} & \partial f_{clear}/\partial E_k & + \\ W_{pleasant} & \partial f_{pleasant}/\partial E_k & + \\ W_{private} & \partial f_{private}/\partial E_k & + \\ W_{design} & \partial \sum_j f_{C_j}/\partial E_k & + \\ & \partial \sum_j f_{G_j}/\partial E_k & \end{array} \quad (4)$$

The partial derivative of the constraint function $f_{C_j}$ for an equality constraint $C_j : (B_{i_j} = K_j)$ is: $\frac{\partial f_{C_j}}{\partial E_k} = -2A_{i_j} \cdot (K_j - B_{i_j})\frac{\partial B_{i_j}}{\partial E_k}$. For an inequality constraint, the partial $\partial f_{C_j}/\partial E_k$ is zero when the constraint is satisfied and is given by the above equation otherwise. The partial of a barrier function $f_{G_j}$ can also be expressed directly as: $\frac{\partial f_{G_j}}{\partial E_k} = -4(E_j - G_j)^{-5}\frac{\partial E_j}{\partial E_k}$.

The partials of the form $\partial E_j/\partial E_k$ are 1 if $j = k$ and zero otherwise. The partials in the form $\partial B_j/\partial E_k$ represent the "influence" that the free variable $E_k$ has on each element radiosity $B_j$. These *influence factors* are equivalent to entries in the inverse of the form factor matrix. Once the influence factors are known, the scene can be rerendered with new light source emissivities without resolving the radiosity equations. Besides providing the partial derivatives necessary for the optimization process, explicit storage of the influence factors also allows interactive, near real time, user adjustments to the lighting.

Rather than perform an explicit inversion of the block structured system, the partial derivatives can be estimated

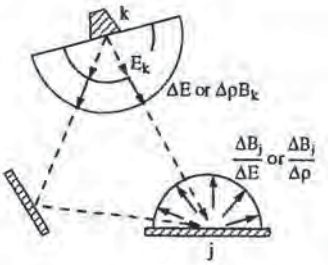



Figure 1: Estimation of $\partial B_j/\partial E_k$ or $\partial B_j/\partial \rho_k$ by shooting a delta emission from source $k$.

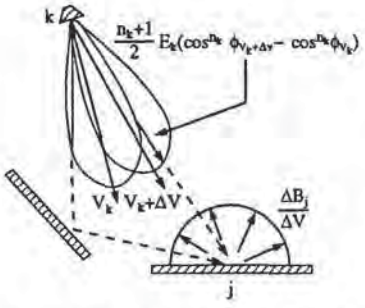



Figure 2: Estimation of $\partial B_j/\partial \mathbf{V}_k$ by shooting a delta emission from source $k$.

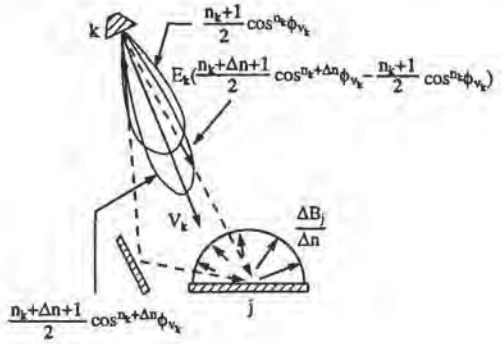



Figure 3: Estimation of $\partial B_j/\partial n_k$ by shooting a delta emission from source $k$.

by finite differences. A small "delta" emission, $\Delta E$, is shot from the variable emission light source as indicated in figure 1 and allowed to interreflect. The iterative shooting operations are very rapid since the links representing the form factors are precomputed during the baseline rendering.

The result of shooting a small amount of energy through the network of links results in an effect on each element radiosity, $\Delta B_j$, thus providing all the derivative estimates $\Delta B_j/\Delta E$. If the only free variables in the optimization are light emissions, these influence factors need only be evaluated once, due to linearity. On the other hand, if any spotlight directionality or element reflectance is allowed to be variable, light emission influence factors must be updated each iteration.

The partial derivative of the objective with respect to a variable element reflectivity is handled in a similar fashion. The element reflectivity $\rho_k$ is adjusted by a small delta $\Delta\rho$. The effect on all other elements can be evaluated by "shooting" the unshot radiosity due to the change in reflectivity: $B_k\Delta\rho$. As with light sources, several shooting iterations may be necessary to account for multiple bounce effects. Once convergence has been achieved, the effect of $\Delta\rho$ on element radiosity $\Delta B_j$ is available and the influence factor estimate $\Delta B_j/\Delta\rho$ can be recorded.

Influence factors for spotlight directionality variables, $\mathbf{V}_k$ and $n_k$, are also approximated through finite differences. For example, a small change, $\Delta\mathbf{V}$, can be made to the direction vector $\mathbf{V}_k$ and the effect on each element radiosity can be determined by a series of shooting steps. The first shooting step, illustrated in figure 2, shoots a delta emission from the modified spotlight to all other elements. The delta emission is determined according to the change in the directionality parameter, in this case, $E_k\frac{(n_k+1)}{2}(cos^{n_k}(\phi_{\mathbf{v}_k+\Delta\mathbf{v}}) - cos^{n_k}(\phi_{\mathbf{v}_k}))$ where $\phi_{\mathbf{v}_k}$ is the angle between the original direction vector of the light and the direction of the element and $\phi_{\mathbf{v}_k+\Delta\mathbf{v}}$ is the angle between the *new* spotlight direction vector and the direction of the element. Subsequent shooting steps proceed in the normal fashion in order to handle multiple bounce effects. The same technique can be used when the distribution pattern parameter $n_k$ is changed as illustrated in figure 3. In this case the radiosity cast is $E_k(\frac{(n_k+\Delta n+1)}{2}cos^{n_k}(\phi_{\mathbf{v}_k}) - \frac{(n_k+1)}{2}cos^{n_k}(\phi_{\mathbf{v}_k}))$.

The cost functions that measure patterns of light or subjective impressions are defined in terms of perception. The partial derivatives of the functions examining lighting patterns with respect to light emission $E_k$ are:

$$\frac{\partial f_{brightness}}{\partial E_k} = -\frac{\sum_i \frac{\partial P_i}{\partial E_i} A_i}{\sum_i A_i}$$

$$\frac{\partial f_{non-uniform}}{\partial E_k} = -\left[\frac{\sum_i (P_{avg,i} - P_i)^2 A_i}{\sum_i A_i}\right]^{-\frac{1}{2}} * \left[\frac{\sum_i (P_{avg,i} - P_i)\left(\frac{\partial P_{avg,i}}{\partial E_k} - \frac{\partial P_i}{\partial E_k}\right)}{\sum_i A_i}\right]$$

$$\frac{\partial f_{peripheral}}{\partial E_k} = \left[\frac{\sum_i \frac{\partial P_i}{\partial E_k} A_i}{\sum_i A_i} - \frac{\sum_j \frac{\partial P_j}{\partial E_k} A_j}{\sum_j A_j}\right]$$

The partials of the subjective impressions are just a linear combination of the partial derivatives of $f_{brightness}$, $f_{non-uniform}$, and $f_{peripheral}$.

The partials $\partial P_j/\partial E_k$ are derived by differentiating equation 2 giving,

$$\frac{\partial P_j}{\partial E_k} = 10^{\gamma}\left[\frac{aa}{B_j}\frac{\partial B_j}{\partial E_k} + \frac{\partial \alpha}{\partial E_k}\zeta\right] \tag{5}$$

where $\alpha$ is the adaption level which can be approximated by $(\sum_i log_{10}(B_i/10,000)A_i)/\sum_i A_i$, $\gamma$ is $aa * log_{10}(B_i/10,000) + bb$, and $\zeta$ is $0.4log_{10}(B_i/10,000) - ln(10)(0.8\alpha + 2.6)$.

If the the adaptation level is assumed constant with respect to a change in emission $E_k$, $\partial\alpha/\partial E_k = 0$, otherwise

$$\frac{\partial \alpha}{\partial E_k} = \frac{A_j}{B_j ln(10)\sum_i(A_i)}\frac{\partial B_j}{\partial E_k}$$

### 3.4 Optimization

The optimization process uses the BFGS algorithm, which evaluates the objective function and gradient at a current step in the design space in order to compute a search direction. Once a search direction is derived, a line search is performed in this direction. Each step in the line search involves a reevaluation of the objective function, hence a reevaluation of the element radiosities which are displayed, allowing the user to watch the progress of the optimization. This process is repeated until the system has converged to a minimum.

## 4 Experiences and Results

The first implementation of the Radioptimization system allowed an objective function based only on photometric measures and did not take into account the psychophysical properties of lighting. The system could successfully optimize lighting but required quite a bit of unintuitive "tweaking" of the objective function weights in order to achieve lighting that had the right subjective appearance. These early experiences led to the investigation of the psychophysical objective functions.

Figure 6 shows the effects that the subjective impressions have on an optimization. The top image constrains the table to have a small amount of illumination while conserving energy and creating an overall impression of visual clarity. To improve efficiency the optimization was run at a low resolution on a simplified model, without the chairs and television set. The optimization process took 1 minute and 21 seconds on an IBM Model 550 RISC System 6000. The bottom image has the same design goals as the top image except that it tries to elicit an impression of privateness. This optimization took 2 minutes and 11 seconds.

It took two or three hours of performing design iterations before developing an intuitive "feel" for the optimization process and the effects of the weights on the objective function. One of the problems with the design cycle is that there may be local minima of the specified objective that are visually unattractive. For example, in addition to the design goals mentioned above for figure 6, we needed to add an additional constraint limiting the illumination of the ceiling because pointing the lights directly at the ceiling was an optimal way of increasing the overall brightness of the room.

One drawback of the system at this point is that it is not fast enough to allow a highly interactive feedback cycle for complex models. However since the system allows a designer to think in terms of their own design goals, it requires fewer design iterations to achieve the desired result.

## 5 Conclusions

This paper has presented a new method of designing illumination in a computer simulated environment, based on goal directed modeling. A library of functions were developed that approximate a room's success in meeting certain lighting design goals such as minimizing energy or evoking an impression of privacy. The objective functions were developed through an experiment in which subjects ordered a set of images according to a particular impression. Processing this data with INDSCAL, showed a correlation between quantitative lighting patterns and subjective measures of visually clarity, pleasantness, and privacy. Once the lighting design goals have been set, the software system searches the space of lighting configurations for the illumination pattern that "best" meets the design specifications. The system absorbs much of the burden for searching the design space allowing the user to focus on the goals of the illumination design rather than the intricate details of a complete illumination specification.

The radioptimization system explores only one possible path in the application of optimization techniques to image synthesis design problems. Constrained optimization techniques may be more suitable than the unconstrained penalty method technique used here when the design goals must be satisfied precisely. Discrete optimization methods may be appropriate in some instances, for example when emissivities are constrained to a finite set, *e.g.* {60 Watts , 100 Watts , $\cdots$}. Geometric properties of the model, such as the position of the lights or the size and position of the windows, could be allowed as free variables. More general image synthesis methods could be applied to account for non-diffuse effects such as glare.

## Acknowledgments

Pat Hanrahan, Larry Aupperle and David Salzman provided the radiosity software used as a basis for this work. Greg Ward offered suggestions on useful objective functions for lighting design. Shinichi Kasahara participated in discussions about the work as it developed.

The first and second author's work was supported by NFS (CCR-9210587). All opinions, findings, conclusions, or recommendations expressed in this document are those of the authors and do not necessarily reflect the views of the sponsoring agencies.

## References

[1] J. J. Chang and J. D. Carroll. How to use INDSCAL: a computer program for canonical decomposition of N-way tables and individual differences in multidimensional scaling. Technical report, Bell Telephone Laboratories, 1972.

[2] M. F. Cohen, S. E. Chen, J. R. Wallace, and D. P. Greenberg. A progressive refinement approach to fast radiosity image generation. *Computer Graphics (SIGGRAPH '88 Proceedings)*, 22(4):75–82, July 1988.

[3] M. F. Cohen and D. P. Greenberg. The hemi-cube: A radiosity for complex environments. *Computer Graphics (SIGGRAPH '85 Proceedings)*, 19(3):31–40, July 1985.

[4] J. E. Flynn. A study of subjective responses to low energy and nonuniform lighting systems. *Lighting Design and Application*, Feb. 1977.

[5] J. E. Flynn, C. Hendrick, T. J. Spencer, and O. Martyniuk. A guide to methodology procedures for measuring subjective impressions in lighting. *Journal of the IES*, Jan. 1979.

[6] J. E. Flynn, T. J. Spencer, O. Martyniuk, and C. Hendrick. Interim study of procedures for investigating the effect of light on impression and behavior. *Journal of the IES*, Oct. 1973.

[7] P. E. Green, F. J. Carmone, Jr., and S. M. Smith. *Multidimensional Scaling Concepts and Applications.* Smith, Allyn, and Bacon, 1989.

[8] P. Hanrahan, D. Salzman, and L. Aupperle. A Rapid Hierarchical Radiosity Algorithm. *Computer Graphics (SIGGRAPH '91 Proceedings)*, 25(4):197–206, July 1991.

[9] J. T. Kajiya. The rendering equation. *Computer Graphics (SIGGRAPH '86 Proceedings)*, 20(4):143–150, Aug. 1986.

[10] H. N. McKay. Energy optimization and quality lighting design. *Lighting Design and Application*, Mar. 1986.

[11] P. Y. Papalambros and D. J. Wilde. *Principles of Optimal Design.* Cambridge University Press, Cambridge, England, 1988.

[12] P. Poulin and A. Fournier. Lights from highlights and shadows. *1992 Symposium on Interactive 3D Graphics*, pages 31–38, Mar. 1992.

[13] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes.* Cambridge University Press, New York, 1986.

[14] J. B. Rosen. The gradient projection method for non-linear programming, part i: Linear constraints. *SIAM*, 8:181–217, 1960.

[15] J. B. Rosen. The gradient projection method for non-linear programming, part ii: Non-linear constraints. *SIAM*, 9:514–532, 1961.

[16] P. C. Sorcar. *Architectural Lighting for Commercial Interiors.* John Wiley and Sons Inc., 1987.

[17] S. S. Stevens and J. C. Stevens. Brightness function: Effects of adaptation. *Journal of the Optical Society of America*, 53(3), Mar. 1963.

[18] J. Tumblin and H. Rushmeier. Tone reproductions for realistic computer generated images. Technical Report GIT-GVU-91-13, Graphics, Visualization, and Usability Center, Georgia Institute of Technology, July 1991.

Figure 5: Sample interface which allows the user to set the weights of the objective and/or specify constraints.

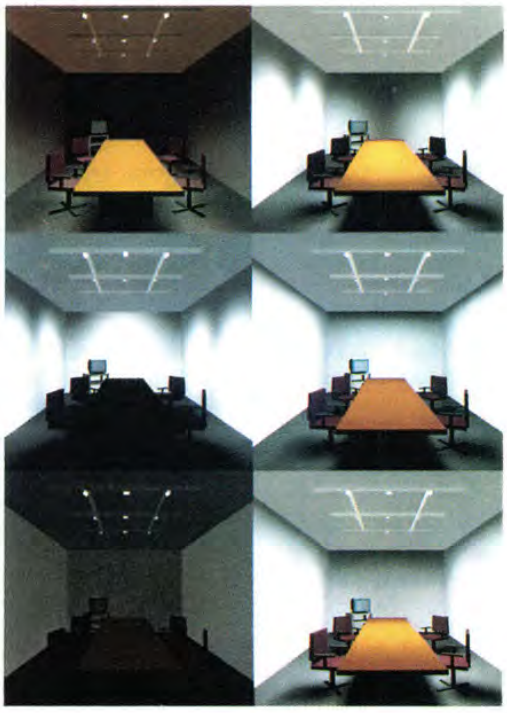


Figure 4: Computer generated rooms used to test subjects on which illumination patterns illicit particular subjective impressions.

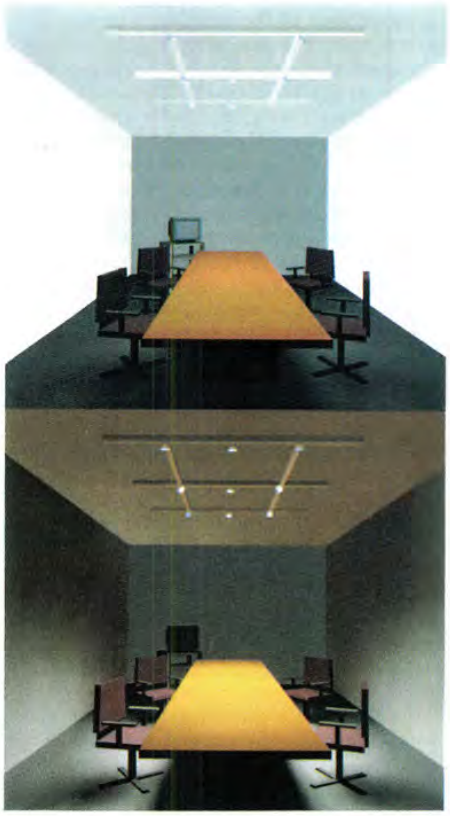


Figure 6: The top image constrains the table to have a small amount of illumination while preserving energy and creating an overall impression of visual clarity. The bottom image also constrains the table to have a small amount of illumination while preserving energy. In addition, it trys to create a feeling of privacy.



# A Hierarchical Illumination Algorithm for Surfaces with Glossy Reflection

Larry Aupperle Pat Hanrahan

Department of Computer Science
Princeton University

## Abstract

We develop a radiance formulation for discrete three point transport, and a new measure and description of reflectance: *area reflectance*. This formulation and associated reflectance allow an estimate of error in the computation of radiance across triples of surface elements, and lead directly to a hierarchical refinement algorithm for global illumination.

We have implemented and analyzed this algorithm over surfaces exhibiting glossy specular and diffuse reflection. Theoretical growth in light transport computation is shown to be $O(n+k^3)$ for sufficient refinement, where $n$ is the number of elements at the finest level of subdivision over an environment consisting of $k$ input polygonal patches — this growth is exhibited in experimental trials. Naive application of three point transport would require computation over $O(n^3)$ element-triple interactions.

**CR Categories and Subject Descriptors:** I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism.

**Key Words**: adaptive meshing, global illumination, radiosity, ray tracing.

## 1 Introduction

A major open problem in image synthesis is the efficient solution of the rendering equation. Radiosity methods have been quite successful over environments containing surfaces that exhibit only diffuse reflection. Unfortunately, very few materials are purely Lambertian reflectors, and efficient solution techniques have not yet been developed for more general specular or glossy reflection functions.

The rendering equation is an integral equation, and the solutions to complicated integral equations are generally obtained using either Monte Carlo or finite element techniques. Monte Carlo algorithms sometimes go under the name of distributed or stochastic ray tracing and are the most commonly employed in computer graphics (e.g. see [4, 5, 9, 12, 16]). Monte Carlo techniques have the advantage that they are easy to implement and can be used for complicated geometries and reflection functions. Unfortunately, their disadvantage is that they are notoriously inefficient. The second approach, the finite element method, has been very successfully applied to the rendering equation under the radiosity assumption, but has only begun to be employed in the general case, and with limited success. For example, Immel et al. [8] discretized radiance into a lattice of cubical environment maps, and solved the resulting system. More recently, Sillion et al. [13] used a mesh of spherical harmonic functions to represent radiance, and solved the resulting system using a shooting algorithm.

 ACM-0-89791-601-8/93/008/0155 $01.50

There are many ways to parameterize the rendering equation, and each leads to a different choice of basis functions. In the transport theory community two techniques are common: directional subdivision (the method of discrete ordinates or $S_N$), and spherical harmonics ($P_N$). These two techniques roughly correspond to the methods of Immel et al. and Sillion et al., although many interesting variations are possible. Our approach is somewhat different, and based on Kajiya's original formulation of the rendering equation [9]. Under this formulation, the rendering equation is expressed in terms of three point transport. That is, the kernel of the integral expresses the transport of light from a point on the source to a point on the receiver, via a point on a reflector. Given this formulation, the three point rendering equation can be discretized over pairs of elements to form a linear system of equations. Solving this system yields the radiance transported between elements. Note that this approach is very similar to the radiosity formulation.

The problem with finite element methods is that the matrix of interactions is very large for interesting environments. For a given environment of $k$ input polygonal patches containing $n$ elements at the finest level of refinement, the three point discretization that we are proposing generates an $n^3$ matrix of interactions. However, in this paper we show that we can accurately approximate the $n^3$ reflectance matrix with $O(n + k^3)$ blocks, in a way very similar to our recent hierarchical radiosity algorithm [7]. In that paper we showed how the $n^2$ form factor matrix could be approximated with $O(n + k^2)$ blocks, resulting in a very efficient algorithm in both space and time. Although the results presented in this paper are preliminary, we believe a hierarchical finite element approach along these lines will ultimately lead to a fast, efficient algorithm.

In the following section we describe our application of the finite element element method to the three point rendering equation, yielding a radiance formulation for discrete transport. In Section 3 we present a simple adaptive refinement algorithm for computation over this formulation, and the iterative solution technique employed for the actual calculation of transport. In Section 4 we discuss our implementation of the algorithm over glossy reflection, and in Section 5 we present some experiments and results. An appendix to this paper contains details of our error analysis for discrete transport under the glossy model.

## 2 Discrete Three Point Transport

The algorithm presented in this paper operates through two functions: refinement of the environment to form a hierarchy of discrete interactions, patches and elements, and the actual computation of illumination over this hierarchy.

In this section we develop the basis for both discretization and transport. We derive a radiance formulation for three point transport, and a new measure and description of reflectance, *area reflectance*. This radiance formulation and associated reflectance provide a natural criterion for discretization under illumination and reflection, and allow both the computation of radiance across triples of individual surface elements, and the expression and computation of all light transport over all surfaces.

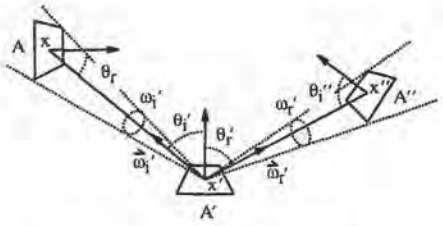



Figure 1: Geometry of Reflection

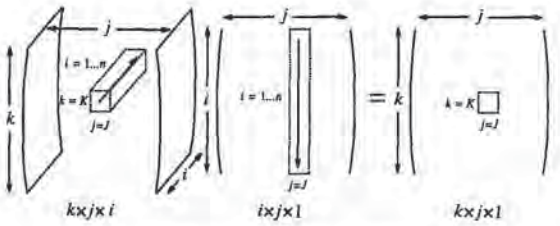



Figure 2: A Reflection Product

## 2.1 A Radiance Formulation for Three Point Transport

When computing and imaging illumination within an environment, we are interested in the transport of light from surface to surface — it is this interaction of surfaces that characterizes illumination, in the absence of participatory media. Reflection within an environment may thus be naturally expressed over triples of surfaces. Consider surfaces $A$, $A'$, and $A''$ (Figure 1) — we will examine the transport of light incident at $A'$ originating at $A$ and reflected toward $A''$.

Let $\omega_i'$ and $\omega_r'$ be the solid angles subtended at point $x'$ by $A$ and $A''$, respectively. Consider differential solid angles at $\vec{\omega}_i'$ and $\vec{\omega}_r'$ — by definition of the bidirectional reflectance-distribution function (BRDF), $f_r$ [11], the radiance $L(\vec{\omega}_r')$ along $\vec{\omega}_r'$ due to illumination through solid angle $\omega_i'$ is:

$$L(\vec{\omega}_r') = \int_{\omega_i'} f_r(\vec{\omega}_i', \vec{\omega}_r') L(\vec{\omega}_i') \cos\theta_i' d\omega_i'$$

Integrating this expression over $\omega_r'$, and introducing $\cos\theta_r'$, we have:

$$\int_{\omega_r'} L(\vec{\omega}_r') \cos\theta_r' d\omega_r' = \int_{\omega_r'} \int_{\omega_i'} f_r(\vec{\omega}_i', \vec{\omega}_r') L(\vec{\omega}_i') \cos\theta_i' \cos\theta_r' d\omega_i' d\omega_r'$$

We may then reparameterize over $A$ and $A''$ to yield:

$$\int_{A''} L(x', x'') G(x', x'') dx'' = \int_A \int_{A''} f_r(x, x', x'') L(x, x') G(x, x') G(x', x'') dx'' dx$$

where

$$G(x, x') = \frac{\cos\theta_r \cos\theta_i'}{|x - x'|^2} v(x, x')$$

where $v(x, x')$ is 1 if points $x$, $x'$, are mutually visible, and 0 otherwise. Note that $G$ is very similar to a differential form factor.

We integrate over $A'$, thus introducing all three areas into the formulation:

$$\int_{A'} \int_{A''} L(x', x'') G(x', x'') dx'' dx' = \int_A \int_{A'} \int_{A''} f_r(x, x', x'') L(x, x') G(x, x') G(x', x'') dx'' dx' dx \quad (1)$$

We may now rewrite the equation in discrete form. Let $A_j$ and $A_k$ be subareas of $A'$ and $A''$ such that $L(x', x'')$ is nearly constant over their surfaces. The left side of equation (1) may then be rewritten, bringing radiance out of the integral as $L_{jk}$:

$$L_{jk} \int_{A_j} \int_{A_k} G(x', x'') dx'' dx' = \pi L_{jk} A_j F_{jk}$$

by definition of the diffuse form factor, $F_{jk}$.

We may similarly discretize $A$ as $A_i$, and rewrite the right side of equation (1) as:

$$L_{ij} \int_{A_i} \int_{A_j} \int_{A_k} f_r(x, x', x'') G(x, x') G(x', x'') dx'' dx' dx = \sum_i \pi L_{ij} A_i F_{ij} R_{ijk}$$

where $R_{ijk}$ is defined such that

$$\pi A_i F_{ij} R_{ijk} = \int_{A_i} \int_{A_j} \int_{A_k} f_r(x, x', x'') G(x, x') G(x', x'') dx'' dx' dx$$

Note that, by the symmetry of $f_r$ and $G$:

$$A_i F_{ij} R_{ijk} = A_k F_{kj} R_{kji}$$

We thus have:

$$\pi L_{jk} A_j F_{jk} = \pi \sum_i L_{ij} A_k F_{kj} R_{kji} = \pi \sum_i L_{ij} A_j F_{jk} R_{kji}$$

by the reciprocity of form factors, and thus:

$$L_{jk} = \sum_i L_{ij} R_{kji}$$

The three dimensional character of $R_{kji}$ over indices $i$, $j$, $k$ leads naturally to a three dimensional matrix formulation for the above system. Consider a product over an $n \times n \times n$ $R_{kji}$ "matrix" and an $n \times n \times 1$ $L_{ij}$ matrix producing an $n \times n \times 1$ matrix of reflected radiances, as shown in Figure 2. Note that the $R_{kji}$ matrix is of size $O(n^3)$ — the hierarchical method discussed in subsequent sections of this paper addresses more tractable representation of this matrix.

Taking into account emission, we have derived a radiance formulation for three point transport:

$$L_{jk} = E_{jk} + \sum_i L_{ij} R_{kji} \quad (2)$$

This formulation states that:

> *The radiance at Area $j$ in the direction of Area $k$ is equal to the radiance emitted by $j$ in the direction of $k$, plus, for every Area $i$, the radiance at $i$ in the direction of $j$ multiplied by the area reflectance $R_{kji}$.*

Note that equation (2) is very similar to the radiosity formulation:

$$B_j = E_j + \rho_j \sum_i B_i F_{ji}$$

### 2.2 Area Reflectance

The quantity $R_{kji}$ has a natural and satisfying physical significance — it is an expression of reflectance over areas $A_i$, $A_j$, and $A_k$.

Consider the fraction of the radiant flux transported from $A_i$ incident to $A_j$ that is reflected in the direction of area $A_k$:

$$\frac{\int_{A_i}\int_{A_j}\int_{A_k} f_r(x,x',x'')L(x,x')G(x,x')G(x',x'')dx''dx'dx}{\int_{A_i}\int_{A_j} L(x,x')G(x,x')dx'dx}$$

If we assume that incident radiance is uniform and isotropic over both $\omega_i'$ (as induced by $A_i$) and $A_j$, we may divide through by $L(x,x')$, yielding:

$$\rho(A_i,A_j,A_k) \equiv \frac{\int_{A_i}\int_{A_j}\int_{A_k} f_r(x,x',x'')G(x,x')G(x',x'')dx''dx'dx}{\int_{A_i}\int_{A_j} G(x,x')dx'dx}$$

We define $\rho(A_i,A_j,A_k)$ to be *area reflectance*. Note that area reflectance is similar to biconical reflectance [11], save that it is also integrated over the reflecting surface.

By definition of $R_{ijk}$:

$$R_{ijk} = \rho(A_i,A_j,A_k)$$

Conservation of energy over reflection, and the reciprocity relation derived for $R_{ijk}$ above, constitute fundamental properties of area reflectance:

1. $\sum_k R_{ijk} \leq 1$, for fixed $i$, $j$.
2. $A_iF_{ij}R_{ijk} = A_kF_{kj}R_{kji}$.

where equality is achieved in property 1 over complete enclosures and perfect reflectivity.

### 2.3 Evaluation of $R_{kji}$

In this section we examine the evaluation of $R_{kji}$ over given patches $A_i$, $A_j$, $A_k$.

Recall:

$$R_{kji} = \frac{\int_{A_k}\int_{A_j}\int_{A_i} f_r(x'',x',x)G(x'',x')G(x',x)dxdx'dx''}{\int_{A_k}\int_{A_j} G(x'',x')dx'dx''}$$

We assume that discrete areas $A_i$, $A_j$, $A_k$ are of small enough scale that $f_r$ and $G$ are relatively constant over their surfaces. Then:

$$\begin{aligned} R_{kji} &= \frac{S_{kji}G_{kj}G_{ji}A_kA_jA_i}{G_{kj}A_kA_j} \\ &= S_{kji}G_{ji}A_i \end{aligned}$$

where $S$ is the discretized value of $f_r$, $S_{ijk} = S_{kji} = S_{x_kx_jx_i}$.

Note that the average value of $G(x',x)$ over $A_i$ and $A_j$ is $\pi F_{ji}/A_i$ — we thus estimate $G_{ji}A_i$ by $\pi F_{ji}$, and compute $R_{kji}$ as:

$$R_{kji} = \pi F_{ji}S_{kji}$$

In practice, it will not be possible to compute the exact values of $F_{ji}$ and $S_{kji}$ over $A_i$, $A_j$, $A_k$. We assume that we are able to estimate these values, along with error bounds for each estimation. Let $\Delta F_{ji}$ and $\Delta S_{kji}$ be error estimates for computed $F_{ji}$ and $S_{kji}$, respectively. We then have an estimate for area reflectance in the form:

$$\begin{aligned} R_{kji} &= \pi(F_{ji}+\Delta F_{ji})(S_{kji}+\Delta S_{kji}) \\ &= \pi(F_{ji}S_{kji} + \Delta F_{ji}S_{kji} + \Delta S_{kji}F_{ji} + \Delta F_{ji}\Delta S_{kji}) \\ &\approx \pi(F_{ji}S_{kji} + \Delta F_{ji}S_{kji} + \Delta S_{kji}F_{ji}) \end{aligned}$$

Assuming $\Delta F_{ji} < F_{ji}$, $\Delta S_{kji} < S_{kji}$, we have neglected the last term and estimate the error in $R_{kji}$ as $\pi(\Delta F_{ji}S_{kji} + \Delta S_{kji}F_{ji})$.

In general, and as is shown for glossy reflection in Section 4, the accuracy of estimators for $F_{ji}$ and $S_{kji}$ is dependent on the size of the patches over which reflectance is computed, relative to their distance apart. As relative size decreases, so does error in computation, leading directly to the adaptive refinement strategy for illumination presented in Section 3 below.

## 3 Algorithms for Three Point Transport

### 3.1 Introduction

Recall equation (2):

$$L_{jk} = E_{jk} + \sum_i L_{ij}R_{kji}$$

This equation suggests both a solution strategy for radiance under three point transport, and a natural representation for illumination within the solution system.

We may interpret equation (2) as a gathering iteration similar to that employed for radiosity under diffuse reflection: the radiance $L_{jk}$ at patch $A_j$ in the direction of patch $A_k$ is found by gathering radiances $L_{ij}$ in the direction of $A_j$ at patches $A_i$. We may solve for transport by gathering radiance for each $L_{jk}$, and successively iterating to capture all significant re-reflection.

We are left with the question of what structure we are gathering over and iterating upon. Note that all illumination is expressed as the radiance at a given patch in the direction of another — it is these patch-patch interactions that form the primary structure within the solution system. All operation is over interactions: both the representation and transport of radiance, and the iteration and solution for illumination.

Consider the following structure:

```
typedef struct _interaction {

    Patch *from;
    Patch *to;

    Color  L;
    Color  Lg;

    List  *gather;

    struct _interaction *nw, *sw, *se, *ne;

} Interaction;
```

A given interaction $ij$ is defined by two patches `ij->from` and `ij->to`, and represents the radiance at `from` in the direction of `to`. This radiance is stored within the interaction as attribute `L`. `Lg` is radiance gathered during the current solution iteration from interactions contained in the list `gather`. Subinteractions `nw`, `sw`, `se`, `ne` are the children of $ij$, induced by subdivision over either `from` or `to`. The structure assumes quadtree refinement, leaving northwest, southwest, southeast, and northeast descendants.

In the following sections we will present an algorithm for the refinement and computation of illumination over a hierarchy of interactions. The algorithm will operate by refining pairs of interactions $ij$, $jk$ (such that `ij->to == jk->from`), to ensure that computed reflectance across the interaction pairs, and associated patch triples, satisfies user specified error bounds. If a given interaction pair $ij$, $jk$ is satisfactory, the interactions are linked to record that radiance may be gathered from $ij$ to $jk$, otherwise one or both interactions are subdivided and refinement applied to their descendants.

After refinement, a gathering iteration may be carried out, each interaction gathering radiance from interactions to which it has been linked. The gathered radiances are then distributed within each receiving interaction hierarchy, and subsequent iterations computed until satisfactory convergence has been achieved.

Note that, within this system, the eye may be regarded as simply another object with which patches may interact. The radiance along interactions to the eye provides the resulting view.

## 3.2 Adaptive Refinement

Consider the following procedure:

```
Refine(Interaction *ij, Interaction *jk,
       float Feps, float Seps, float Aeps)
{
    float feps, seps;

    feps = GeometryErrorEstimate(ij);
    seps = ReflectionErrorEstimate(ij, jk);

    if (feps < Feps && seps < Seps)
        Link(ij, jk);

    else if (seps >= Seps) {

        switch(SubdivS(ij, jk, Aeps)) {

        case PATCH_I:

            Refine(ij->nw, jk, Seps, Feps, Aeps);
            Refine(ij->sw, jk, Seps, Feps, Aeps);
            Refine(ij->se, jk, Seps, Feps, Aeps);
            Refine(ij->ne, jk, Seps, Feps, Aeps);

            break;

        case PATCH_J:
            /* refine over children of ij and jk */
        case PATCH_K:
            /* refine over children of jk        */
        case NONE:
            Link(ij, jk);
        }
    }

    else {    /* feps >= Feps */

        switch(SubdivG(ij, jk, Aeps)) {

        /* refine over children, or link, as    */
        /* directed by PATCH_I, J, K, or NONE.  */

        }
    }
}
```

This procedure computes over pairs of interactions, and associated patch triples, subdividing and recursively refining if estimated error exceeds user specified bounds, linking the interactions for gathering if the bounds are satisfied, or if no further subdivision is possible. `Feps` and `Seps` are the bounds for geometric and reflection error, respectively; `Aeps` specifies the minimum area a patch may possess and still be subdivided. `GeometryErrorEstimate` and `ReflectionErrorEstimate` provide estimations for $\pi \Delta F_{ji} S_{kji}$ and $\pi \Delta S_{kji} F_{ji}$.

`SubdivS` and `SubdivG` control refinement for reflection and geometry error, respectively. Both routines select a patch for refinement, subdividing the patch and associated interaction(s) if required. An identifier for the selected patch is returned — if no patch may be subdivided, then `NONE` is passed back. Note that a given interaction/patch may be refined against many different interactions within the system, and thus may have already been subdivided when selected by a `Subdiv` routine — in this case, the routine simply returns the proper identifier.

The `Subdiv` routines should select for refinement patches that are of large size relative to their distance from their partner(s) in the transport triple. Form factor estimation is a convenient criterion for the determination of such patches — a large differential to area form factor $F_{dpq}$ indicates that patch $q$ is of large relative size. Care must be taken in subdivision, however, to ensure that each interaction is always subdivided in the same way for all refinements involving that interaction.

The `Subdiv` routines thus choose for refinement the patch of size at least `Aeps` that is of greatest form factor within $ij$ and/or $jk$ that will not induce multiple sets of children over either interaction. If patch $p_j$ is of greatest form factor over both $ij$ and $jk$, and of area greater than `Aeps`, then it is chosen for refinement (Figure 3 at middle). Otherwise, if $p_j$ is selected over one interaction, but $p_i$ or $p_k$ is selected over the other, then the "outside" patch is chosen for refinement. Given two selected outside patches, `SubdivS` selects the one of greater form factor relative to $p_j$; `SubdivG` selects $p_i$ over $p_k$, as $p_k$ has no direct effect on geometric accuracy. Note, however, that even under `SubdivG`, if only $p_j$ and $p_k$ are allowed subdivision, $p_k$ will be selected, although with further subdivision the triple will eventually balance sufficiently to allow refinement over $p_j$.

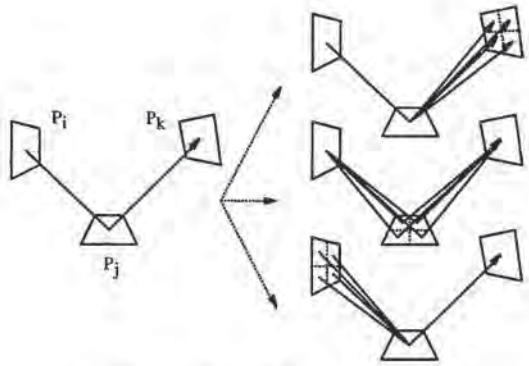



Figure 3: Refinement and Subdivision

## 3.3 Gathering Radiance

Gathering radiance over interactions may be written as a simple procedure:

```
Gather(Interaction *jk)
{
    Interaction *ij;

    if (jk) {

        jk->Lg = 0;

        ForAllElements(ij, jk->gather)
            jk->Lg += ij->L * Reflectance(ij, jk);

        Gather(jk->nw);
        Gather(jk->sw);
        Gather(jk->se);
        Gather(jk->ne);
    }
}
```

We gather radiance into `jk->Lg` rather than directly into `jk->L` to avoid the necessity of a push/pull with every invocation of the procedure (see Section 3.4). The solution method is thus simple Jacobi iteration, as opposed to Gauss-Seidel, as the hierarchical structure imposes simultaneous rather than successive displacement.

## 3.4 Radiance within a Hierarchy

A gathering iteration results in received radiance scattered throughout each interaction hierarchy. This gathered radiance must be distributed and accounted for over all ancestors and descendants of each receiving interaction, in order to maintain the consistency and correctness of the hierarchical representation of radiance between patches.

We employ a distribution algorithm similar to that presented in [7] for radiosity over patch/element hierarchies: gathered radiance is "pushed" to the leaf interactions within each hierarchy to ensure propagation to all descendants, and then "pulled" and distributed back up from the leaves through all higher level interactions to their common ancestor at the root. As is shown in [2], radiance may be pushed unchanged within the interaction hierarchy, and area averaged as it is pulled from child to parent.

# 4 Application over Glossy Reflection

In this section we discuss our implementation of the above algorithms over glossy reflection.

## 4.1 The Reflection Function

We employ a highly simplified Torrance-Sparrow [15] model for our glossy reflection function:

$$f_g(\vec{\omega}_i, \vec{\omega}_r) = \frac{\kappa + 2}{8\pi} \frac{\cos^{\kappa} \theta_m}{\cos\theta_i \cos\theta_r} sh(\theta_i, \theta_r)$$

This function incorporates the facet distribution function $\cos^{\kappa} \theta_m$ developed by Blinn [3], normalized for projected facet area under

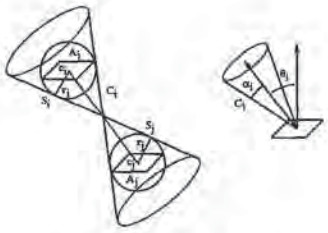


Figure 4: Estimating Cones

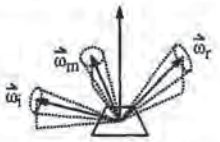


Figure 5: $C_i$, $C_r$, and $C_m$

[10]. Angle $\theta_m$ is that made to the mean surface normal by $\vec{\omega}_m$, the microfacet mirror orientation normal lying halfway between $\vec{\omega}_i$ and $\vec{\omega}_r$.

Function $sh(\theta_i, \theta_r)$ expresses self-shadowing over microfacets — for near specular surfaces, such self-shadowing or masking does not become critical until relatively high $\theta_i$ or $\theta_r$ [6]. The implemented system thus simply clamps $sh$ from 1 to 0 when $\theta_i$ or $\theta_r$ exceeds a preset $\theta_{\text{bound}}$ near the horizon. This scheme serves as a crude approximation to the shadowing function; however, a better strategy would be to employ a much fuller tabulation of the function, incorporated into the error analysis presented below. A more complete discussion of shadowing and conservation of energy over $f_g$ is presented in [2].

### 4.2 Error Estimation

Recall the general expression for error derived in Section 2.3:

$$\pi(\Delta F_{ji} S_{kji} + \Delta S_{kji} F_{ji})$$

In implementation we have estimated the form factor $F_{ji}$ by $F_{dji}$, the form factor from a differential area at $A_j$ to a disk of area $A_i$ centered at $A_i$, as was employed in [7]. As discussed in [7], the relative error in this estimate is proportional to the estimate itself. In our implementation we have thus estimated absolute error $\Delta F_{ji}$ as at most proportional to $F^2_{dji}$.A brief discussion of relative and absolute error over hierarchical methods is presented in [2].

We now consider the error estimate $\Delta S_{kji}$. As discussed in the appendix to this paper, we may compute bounding cones $C_i$, $C_r$, and $C_m$ over all possible incident, reflected, and mirror orientation directions induced at $A_j$ by $A_i$ and $A_k$ (Figures 4 and 5 — these figures are discussed more fully in the appendix). We may then compute maximum and minimum $\cos^\kappa \theta_m$, $\cos\theta_i$, $\cos\theta_r$ over these cones, and estimate error by interval width. The full expression for estimated error over transport is given in the appendix.

### 4.3 Clamping and Visibility

Evaluation of glossy reflectance over three surface areas, as required by the gather iteration, may be difficult, particularly if surface subdivision has been limited by Aeps rather than satisfaction of error bounds, and if $\kappa$, the facet distribution exponent, has high value. In this case we must estimate the integral of a spikey function over a relatively broad area.

Our solution is to band limit the BRDF in a fashion similar to that presented by Amanatides [1]. We employ the cone estimation techniques of the previous section to determine if the BRDF varies significantly over the given patches — if this variance exceeds a set bound, we "roughen" the reflecting surface, lowering $\kappa$ to broaden the resulting reflection over the estimated cones. We then renormalize the resulting blurred function, as described in [1], to

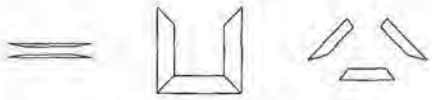


Figure 6: Geometric Configurations

prevent amplification of its low frequency components. We note that the resulting antialiasing is relatively aggressive, significantly dimming or eliminating reflections requiring overmuch blurring.

In implementation, we have computed visibility via jittered ray casting and inheritance similar to that of [7], storing visibility data in interactions as it is computed.

## 5 Results

### 5.1 Growth in Transport

We have measured the growth in transport triples (linked interactions) versus $n$, the maximum number of elements at the finest level of subdivision, over parallel, perpendicular, and "oriented" patches (Figure 7). The corresponding geometries are shown in Figure 6. The graphs show linear or near linear behavior over each range — the graph of triples vs. $n$ for the perpendicular case is slightly concave over the lower data points, but subsides to linear with further refinement.

In previous work [7] on hierarchical refinement for radiosity, it was shown that for error estimate proportional to $F_{dji}$, and sufficient refinement, each subpatch may only interact with other patches in a limited local neighborhood. As discussed in [7], each patch may thus participate in at most $c$ interactions, for some constant $c$ independent of $n$ and $k$. Adaptive refinement thus generates at most $O(n)$ transport interactions. We will show a similar bound for discrete three point transport under glossy reflection.

Recall that the estimate for error in computed transport is proportional to $\Delta F_{ji} S_{kji} + \Delta S_{kji} F_{ji}$. Our argument depends on two assumptions:

1. We may bound both $\Delta S_{kji}$ and $S_{kji}$ by some $S_{\max}$.

   As discussed below, the lower this $S_{\max}$, the smaller the magnitude of the leading coefficient underling the resulting bound.

   Note that our argument thus does not apply to perfect specular reflection, as the corresponding BRDF incorporates the Dirac delta function [11]. Equivalently, the argument does not hold over $f_g$ for $\kappa = \infty$ (inducing mirror reflection), as we can not provide a finite bound for $S$ in this case.

   For finite $\kappa$, however, the desired bound over glossy reflection is achieved by:

$$\frac{\kappa + 2}{8} \max(\cos^\kappa \theta_m) \max(\sec\theta_i) \max(\sec\theta_r)$$

   The maxima over the secant terms are bounded by microfacet self-shadowing.

2. $\Delta F_{ji}$ and $F_{ji}$ within our error estimate are at most proportional to $F_{dji}$.

   Recall that we estimate $F_{ji}$ as $F_{dji}$, and $\Delta F_{ji}$ as $F^2_{dji}$, thus satisfying this assumption.

Given these assumptions, estimated error is at most proportional to $S_{\max} F_{dji}$.

We may now show $O(n)$ growth, for sufficient refinement. Consider refinement over interaction $ij$ under an error estimate at worst proportional to $S_{\max} F_{dji}$. The error estimate is thus proportional to $F_{dji}$, and therefore, for sufficient refinement, there are at most $O(n)$ such interactions, as discussed in [7].

Consider now an error satisfied link from $ij$ to an interaction $jk$. For sufficient refinement under our subdivision scheme, we may assume that form factors $F_{ij}$, $F_{ji}$, $F_{jk}$, $F_{kj}$ over $p_i$, $p_j$, and $p_k$ are roughly equal. Furthermore, these satisfying form factors depend only on the error estimate, reflection function, and error bounds, not on $n$ or $k$.

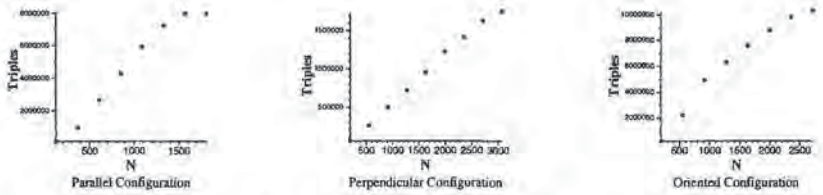



Figure 7: Triples vs. $N$ over Geometry. Error bounds $e = 0.1$. Glossy exponent $\kappa = 25$. For oriented case $e = 0.005$.

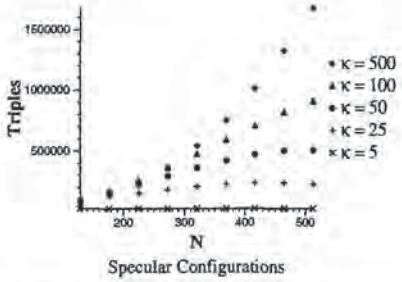



Figure 8: Triples vs. $N$ over $\kappa$. The graph is over parallel polygons for which the error bounds and interpolygon distance have been doubled.

At worst the above form factors are such that $F_{**}S_{\max} < \texttt{Eps}$, where Eps is the most restrictive error bound. Note that, as stated above, $F_{**}$ depends only on the error estimate, reflection function (ie. $S_{\max}$), and error bounds. Only some constant number of such form factors may be fitted over the directional hemisphere above $p_j$, and thus $ij$ may only be linked to some constant number of interactions $jk$. The total number of linked interactions, and corresponding transport triples, is thus $O(n)$.

Note that the above argument, although it establishes the desired bound, may overstate the potential for links at a given interaction. For a given $ij$, much of the directional reflection into the hemisphere over $p_j$ may not achieve $S_{\max}$, and may even be of maximum 0. That is, the analysis ignores the modulation between the paired error and value terms within the error estimate.

As $\kappa$ increases in magnitude, the corresponding bound $S_{\max}$ must increase as well. We may thus expect greater growth in transport computation with higher specular exponent, as shown in Figure 8. Within this graph, growth is superlinear for $\kappa = 500$, though further trials over a higher range of $n = 500 \ldots 2000$ have shown that the rate subsides to linear as $n$ increases, allowing sufficient refinement for the local neighborhood property to obtain.

Finally, we note that under specular reflection each element is reflected across every other element perfectly, and to a first approximation is visible from a constant number of other elements in the environment (at least in the case of a convex enclosed room; the analysis is complicated by occlusion and certain worst case alignments). Thus, the number of interactions is at least $O(n^2)$ — we conjecture that it is no worse than this bound.

## 5.2 Illumination and Refinement

Figure 9 shows illumination and meshing over surfaces of varying glossiness (specular exponent). Within each image, the reflecting surface is perpendicular to the diamond shaped light source, and we see the resulting reflection in the direction of the eye. Note the conformation of meshing to the highlight over each surface. The "stretched" nature of the highlight along the axis to the eye is characteristic of Torrance-Sparrow reflection over fairly oblique angles, and accounts for the increased sensitivity of meshing along this axis. The rightmost three images in the figure show the meshing from above. The illumination shown in these images is somewhat unusual - it shows the reflection to the eye as though it had been painted on the reflecting surface, and then viewed from a different location, directly above. The images in Figure 11 show similar eye/offset views for the reflection of a garish checkerboard.

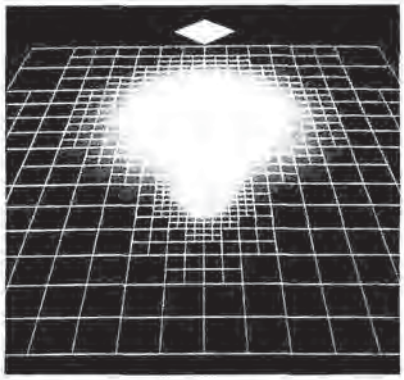


Figure 10: Meshing for glossy and diffuse reflection

| Figure 9 | | | | |
|---|---|---|---|---|
| Max Elements | | 4160 | | |
| Max Triples | | 262144 | | |
| Computed | patches | elements | triples | time |
| $\kappa = 25$ | 790 | 593 | 6706 (2.6%) | 2.2s |
| $\kappa = 100$ | 1290 | 968 | 24214 (9.2%) | 8.0s |
| $\kappa = 500$ | 874 | 656 | 12106 (4.6%) | 4.1s |
| **Figure 10** | | | | |
| Max Elements | | 16448 | | |
| Max Triples | | 1048576 | | |
| Computed | patches | elements | triples | time |
| $\kappa = 500$ | 1578 | 1184 | 7834 (0.75%) | 5.0s |
| **Figure 12** | | | | |
| Max Elements | | 15138 | | |
| Max Triples | | 222385209344 | | |
| Computed | patches | elements | triples | time |
| $\kappa = 500$ | 6479 | 4866 | 70995 (0.00003%) | 3m13s |

Table 1: Image Statistics

The image in Figure 10 shows contrasting illumination and meshing induced by diffuse and glossy reflection. Note the distinct meshing for each highlight. Glossy reflection is at a less oblique angle, and thus both the highlight and meshing exhibit less distortion in the direction of the eye.

Note that these scenes are extremely simple — application to more complex environments is still very expensive, despite the employment of hierarchical methods. Motivated by the work of Smits et al. [14] in hierarchical radiosity, we are currently experimenting with importance and radiance weighting over three point transport — preliminary results of this work are shown in Figure 12. The given environment contains four reflectors: the broad face of each of the three "slabs" and the top of the central cube. In addition to the reflections seen in the slabs, note the play of light originating at the lamp at left, reflected off the cube top, and over the upper part of the green wall at right. Total potential transport triples over this environment at the finest level of subdivision is just over 222 billion — our system, under importance and radiance weighting, employs 70,995, a reduction to 3 hundred-thousandths of 1 percent.

Table 1 provides further statistics for the images. Timings are given for a Silicon Graphics indigo workstation with a single 50 MHz R4000 processor. The image shown in Figure 12 was generated after seven complete iterations (gathers to all interactions), and total time just over three minutes.

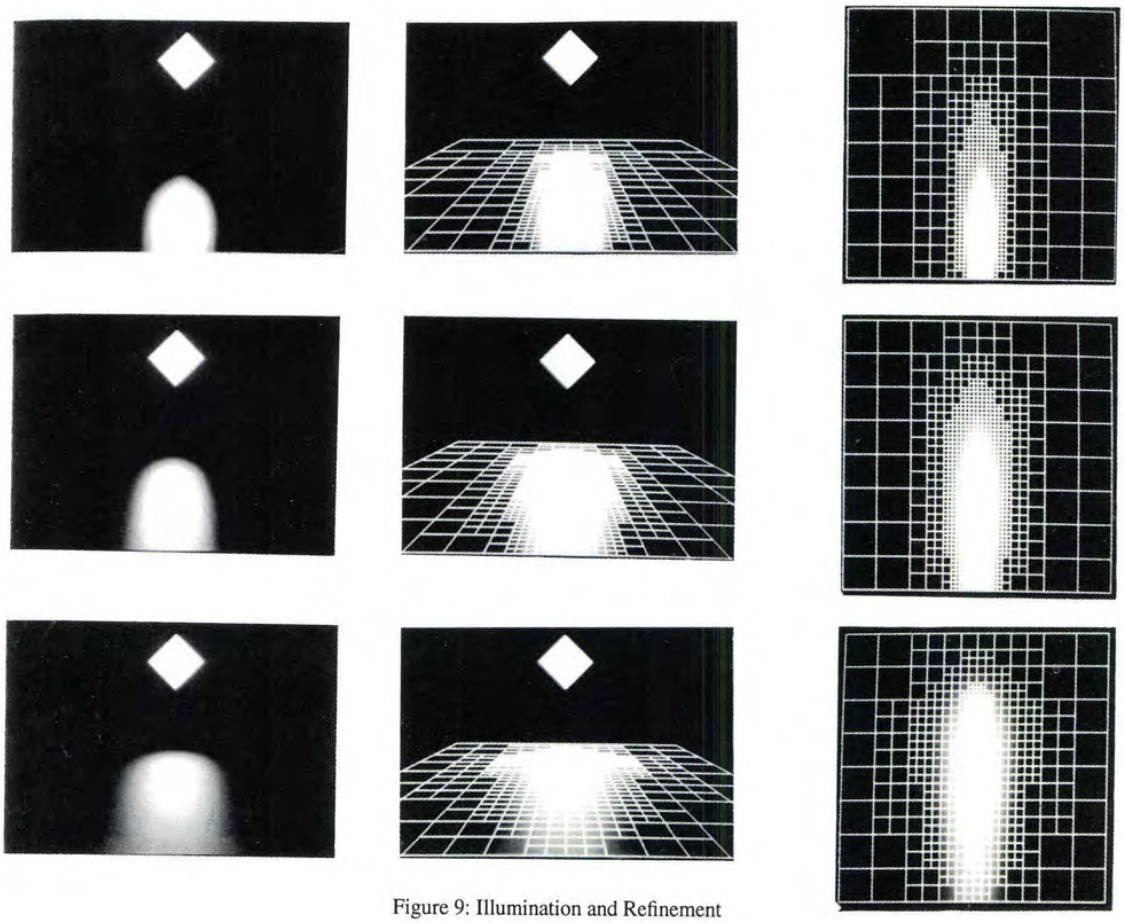


Figure 9: Illumination and Refinement

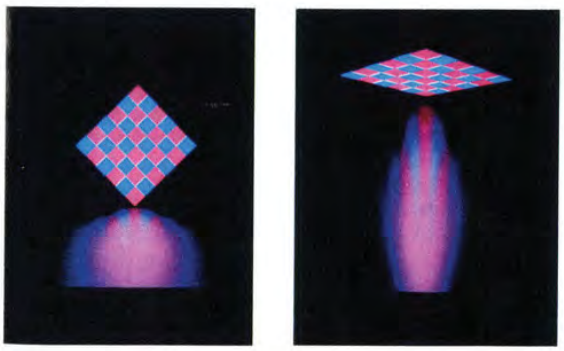


Figure 11: Eye and Offset Views

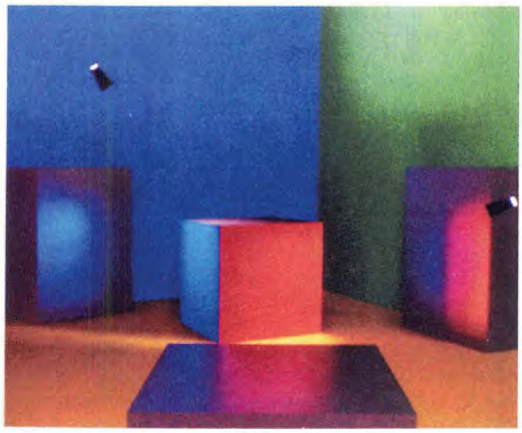


Figure 12: Cube and Slabs

## 6 Discussion

Recall the matrix formulation shown in Figure 2. For any $n$ of reasonable size, the resulting $n^3$ matrix will be unmanageable — we have shown, however, that for sufficient refinement the $n^3$ entries in the matrix may be approximated to within user specified bounds by $O(n)$ subblocks. The gather and push/pull procedures described in preceding sections allow manipulation and solution over this representation. As discussed in [2], the resulting system may be shown to converge.

Growth in transport is more accurately described as $O(n + k^3)$, where $k$ is the number of input polygonal patches within the environment, as opposed to elements. The $k^3$ term is generated by the initial examination of all polygon triples for reflection, and is subsumed by $n$ as the number of elements increases. As the number of polygons in an environment grows, however, the $k^3$ term will become prohibitively large. As discussed in [14] with respect to the related problem under hierarchical radiosity, the capability to cluster as well as refine polygons would reduce the difficulty of unnecessary initial interactions. Clustering is arguably the most important open problem in the computation of global illumination.

The hierarchical approach described in this paper was derived by writing the rendering equation in a three point transport formulation. Another option would be to parameterize radiance by position and direction – we believe that a similar hierarchical approach could be employed with the method of discrete ordinates or spherical harmonics.

Finally, we note that, similarly to other algorithms for hierarchical illumination [7, 14], the algorithm described in this paper bounds estimated error over individual transport computations. As discussed in [14], bounding estimated error over individual transport does not easily or necessarily provide a rigorous bound for overall error in the solution. An analysis and means of computing such a bound over hierarchical illumination remains an interesting open problem.

## 7 Acknowledgements

This research was partially supported by equipment grants from Apple and Silicon Graphics Computer Systems and a research grant from the National Science Foundation (CCR 9207966). The authors would like to thank Dr. P. Prusinkiewicz for access to the graphics research facilities at the University of Calgary during the final stages of this work, and Deborah Fowler for her crucial assistance in shooting test images, paste up and much other support and encouragement. Thanks to Cullen Jennings and David Laur for all of their help recording images. We especially thank the anonymous referees for their many helpful comments and suggestions.

## 8 References

[1] Amanatides, J. (1992) Algorithms for the detection and elimination of specular aliasing. *Proc. Graphics Interface '92*, 86-93.

[2] Aupperle, L. (1993) Hierarchical algorithms for illumination. Doctoral Dissertation, Princeton University.

[3] Blinn, J.F. (1977) Models of light refection for computer synthesized pictures. *Computer Graphics* 11 (2), 192-198.

[4] Chen, S.E., Rushmeier, H.E., Miller, G., Turner, D. (1991) A progressive multi-pass method for global illumination. *Computer Graphics* 25 (4), 165-174.

[5] Cook, R.L. (1986) Stochastic sampling in computer graphics. *ACM Transactions on Graphics* 5 (1), 51-72.

[6] Hall, R. (1989) Illumination and color in computer generated imagery. Springer-Verlag, New York.

[7] Hanrahan, P., Salzman, D., Aupperle, L. (1991) A rapid hierarchical radiosity algorithm. *Computer Graphics* 25 (4), 197-206.

[8] Immel, D.S., Cohen, M.F., Greenberg, D.P. (1986) A radiosity method for non-diffuse environments. *Computer Graphics* 20 (4), 133-142.

[9] Kajiya, J.T. (1986) The rendering equation. *Computer Graphics* 20 (4), 143-150.

[10] Mitchell, D. (1992) Manuscript.

[11] Nicodemus, F.E., Richmond, J.C., Hsia, J.J., Ginsberg, I.W., Limperis, T. (1977) Geometrical considerations and nomenclature for reflectance. National Bureau of Standards monograph, no. 160.

[12] Shirley, P. (1990) A ray tracing method for illumination calculation in diffuse-specular scenes. *Proc. Graphics Interace '90*, 205-212.

[13] Sillion, F.X., Arvo, J.R., Westin, S.H., Greenberg, D.P. (1991) A global illumination solution for general reflectance distributions. *Computer Graphics* 25 (4), 187-196.

[14] Smits, B.E., Arvo, J.R., Salesin, D.H. (1992) An importance-driven radiosity algorithm. *Computer Graphics* 26 (2), 273-282.

[15] Torrance, K.E., Sparrow, E.M. (1967) Theory for off-specular reflection from roughened surfaces. *J. of the Optical Society of America* 57 (9), 1105-1114.

[16] Ward, G.J., Rubinstein, F.M., Clear, R.D. (1988) A ray tracing solution for diffuse environments. *Computer Graphics* 22 (3), 85-92.

## Appendix: Error Analysis

Recall the error expression derived in Section 2.3:

$$\pi(\Delta F_{ji} S_{kji} + \Delta S_{kji} F_{ji})$$

In implementation, we have divided $\Delta S_{kji}$ into separate components for each subfactor of $f_g$. We thus have:

$$\frac{\kappa+2}{8}\Big(\Delta F_{ji}\frac{\cos^\kappa\theta_m}{\cos\theta_i\cos\theta_r} + \Delta\cos^\kappa\theta_m F_{ji}\frac{1}{\cos\theta_i\cos\theta_r} + \Delta\sec\theta_i F_{ji}\frac{\cos^\kappa\theta_m}{\cos\theta_r} + \Delta\sec\theta_r F_{ji}\frac{\cos^\kappa\theta_m}{\cos\theta_i}\Big)$$

In implementation, the refinement procedure of Section 3.2 takes an additional argument, `Ceps`, against which the two estimates of error in reciprocal cosine are tested.

We are left with the computation of $\Delta\sec\theta_i$, $\Delta\sec\theta_r$, and $\Delta\cos^\kappa\theta_m$. The variance (and associated error) in these cosine terms over given patches $A_i$, $A_j$, $A_k$ is determined by the set of possible $\vec{\omega}_i$, $\vec{\omega}_r$ lying between the patches (we dispense with ' notation in this section).

Consider patches $A_i$, and $A_j$ (Figure 4): we enclose these patches in spheres $S_i$, $S_j$ with centers $c_i$, $c_j$, and radii $r_i$, $r_j$, respectively. For the moment we will assume that the interiors of $S_i$ and $S_j$ do not intersect, and thus there exists a tangent cone lying between the spheres.

Note that this cone is a right circular cone centered on the line joining $c_i$ and $c_j$. Consider the nappe containing $S_i$: it may be regarded as a cone of direction vectors centered about the vector $c_i - c_j$. We will call this vector cone $C_i$. If $p_i$ and $p_j$ are any two points on or in $S_i$, $S_j$, then the vector $p_i - p_j$ lies within $C_i$. $C_i$ thus bounds the set of possible $\vec{\omega}_i$. We may characterize $C_i$ by the angle $\alpha_i$ defined by its axis, $c_i - c_j$, and boundary — cone $C_r$ and angle $\alpha_r$ may be similarly defined over $A_j$ and $A_k$. If either pair of spheres intersect, we set the corresponding $\alpha = \pi$. We may easily compute maxima and minima for $\sec\theta_i$ and $\sec\theta_r$ given $C_i$ and $C_r$, and may then compute error in estimation as $(\max - \min)/2$.

The cones $C_i$ and $C_r$ centered about $\vec{\omega}_i$ and $\vec{\omega}_r$ induce a similar cone of variation about $\vec{\omega}_m$ (Figure 5). Application of basic spherical trigonometry yields [2]:

$$\alpha_m \le \arcsin\min\Big(\frac{\sin(\alpha_i/2)+\sin(\alpha_r/2)}{\vec{\omega}_i\cdot\vec{\omega}_m}, 1.0\Big)$$

Given $\alpha_m$, determination of $\max(\cos^\kappa\theta_m)$, $\min(\cos^\kappa\theta_m)$, and thus $\Delta\cos^\kappa\theta_m$ immediately follows.

Having computed these estimates and maxima, and incorporating the estimates for form factor computation, we may bound and estimate error in transport as:

$$\frac{\kappa+2}{8}\Big(F_{dji}^2\max(\cos^\kappa\theta_m)\max(\sec\theta_i)\max(\sec\theta_r) + \Delta\cos^\kappa\theta_m F_{dji}\max(\sec\theta_i)\max(\sec\theta_r) + \Delta\sec\theta_i F_{dji}\max(\cos^\kappa\theta_i)\max(\sec\theta_r) + \Delta\sec\theta_r F_{dji}\max(\cos^\kappa\theta_i)\max(\sec\theta_i)\Big)$$

It is this error measure that we employ in our implementation.

# On the Form Factor between Two Polygons

**Peter Schröder** **Pat Hanrahan**

Department of Computer Science
Princeton University

## Abstract

Form factors are used in radiosity to describe the fraction of diffusely reflected light leaving one surface and arriving at another. They are a fundamental geometric property used for computation. Many special configurations admit closed form solutions. However, the important case of the form factor between two polygons in three space has had no known closed form solution. We give such a solution for the case of general (planar, convex or concave, possibly containing holes) polygons.

**CR Categories and Subject Descriptors:** I.3.7 [Computer Graphics]: *Three-Dimensional Graphics and Realism – Radiosity*; J.2 [Physical Sciences and Engineering]: *Engineering*.

**Additional Key Words and Phrases:** Closed form solution; form factor; polygons.

## 1 Introduction

When using the radiosity technique to create images the form factor plays a central role. It describes the fraction of radiation diffusely emitted from one surface reaching another surface. The accurate computation of form factors is the central theme in many recent papers. Goral et al. [4], who introduced radiosity to the computer graphics community, used numerical contour integration to compute form factors between polygons. Cohen and Greenberg [3] took visibility into account with their hemicube algorithm. More recent hierarchical and adaptive algorithms compute still more accurate form factors [10; 5]. Nishita and Nakamae [8] and Baum et al. [2] have used an exact solution for the form factor between a differential surface element and a polygon. Most radiosity algorithms are restricted to polygonal environments, and so a closed form solution for the form factor between polygons is potentially of great utility.

The history of computing form factors is very long. A closed form expression for the form factor between a differential surface element and a polygon was found by Lambert in 1760 [7]. Lambert proceeded to derive the form factor for a number of special configurations among them the form factor between two perpendicular rectangles sharing a common edge. He writes about the latter derivation:

> Although this task appears very simple its solution is considerably more knotted than one would expect. For it would be very easy to write down the differential expression of fourth order, which one would need to integrate four fold; but the highly laborious computation would fill even the most patient with disgust and drive them away from the task.

Other workers have derived closed form solutions for the form factors between many different geometric configurations and these can be found in standard textbooks. However, we are not aware of a closed form solution for the form factor between two general polygons. Thus, this problem has remained open for over 230 years.

 ACM-0-89791-601-8/93/008/0163 $01.50

In this paper we present a formula for the form factor integral between two general polygons. The derivation of this formula is quite involved, and the interested reader is referred to [9] for a detailed derivation. The purpose of this paper is to bring this result to the attention of the graphics community.

## 2 Closed form solution

The form factor integral can be reduced to a double contour integral by two applications of Stokes' theorem [6]

$$\begin{aligned}\pi A_1 F_{12} &= \int_{A_1}\int_{A_2} \frac{\cos\theta_1 \cos\theta_2}{\|\vec{r}\|^2}\, dA_2\, dA_1 \\ &= \frac{1}{4}\int_{\partial A_1}\int_{\partial A_2} \ln(\vec{r}\cdot\vec{r})\, d\vec{x}_2 \cdot d\vec{x}_1\end{aligned}$$

where $\theta_1$, $\theta_2$ are the angles between the normal vector of the respective surface and a radius vector $\vec{r}$, which connects two points on the surfaces. The above equation holds for all surfaces such that every point on either surface sees the same contour of the other surface.

In the case of polygons $P_1$ and $P_2$ the contour integral reduces to a sum of double line integrals over all pairwise combinations of edges

$$4\pi A_{P_1} F_{P_1 P_2} = \sum_{E_i E_j} \cos\angle E_i E_j \int_{E_i}\int_{E_j} \ln(\vec{r}\cdot\vec{r})\, ds_j\, dt_i$$

Ignoring the factor $\cos\angle E_i E_j$ we are left with the task of giving a solution to integrals of the general form $\int_0^{c_2}\int_0^{c_0} \ln f(s,t)\, ds\, dt$. $c_0$ and $c_2$ are the lengths of the edges over which a given double contour integral is taken and $f(s,t) = s^2 + c_1 st + t^2 + c_3 s + c_4 t + c_5$ is the bi-quadratic form which arises from the expansion of the dot product (see Table 2 for definitions of all variables). If the two line segments lie in a common plane we can factor $f(s,t)$ into two bi-linear forms and a solution is readily obtained with standard integration tables (see [9]). Lines in general position lead to the following result:

$$\begin{aligned}&\int_0^{c_2}\int_0^{c_0} \ln f(s,t)\, ds\, dt \\ &= \left\{\left[(s+\frac{c_3}{2})\mathrm{G}(f(.,t))(s) + \frac{c_1}{2}\mathrm{H}(f(.,t))(s)\right]\Big|_{s=0}^{s=c_0}\right\}\Big|_{t=0}^{t=c_2} \\ &\quad -2c_0c_2 + c_{14}c_{15}\left\{\left[\pi(2k(s)+1)\mathrm{M}(t)\right.\right. \\ &\quad -i\left(\mathrm{L}(-c_{17}(s))(t) + \mathrm{L}(-c_{18}(s))(t)\right. \\ &\quad \left.\left.\left. -\mathrm{L}(c_{17}(s))(t) - \mathrm{L}(c_{18}(s))(t)\right)\right]\Big|_{s=0}^{s=c_0}\right\}\Bigg|_{t=\sqrt{\frac{c_{13}}{c_{13}}}}^{t=\sqrt{\frac{c_{13}*c_2}{c_{13}*c_2}}}\end{aligned}$$

where $k(s) \in \{-1, 0, 1\}$ according to the particular branchcut of the complex logarithm choosen in L. The auxiliary functions G, H, L, and M are given in Table 1.

## 3 An example

We have implemented our closed form solution in Mathematica [11] (this code is available from `ps@princeton.edu`). The

$$
\begin{aligned}
\mathrm{L}(b)(y) &:= \int^y t^2(1-t^2)^{-3}\ln(b+t)\,dt &&= \frac{1}{16}\left[\frac{-b\ln(y-1)}{(b+1)^2} - \frac{b\ln(1+y)}{(b-1)^2} + \left(\frac{2(b+y)(1+by)[(b-y)^2+(by-1)^2]}{(b^2-1)^2(y^2-1)^2} + \ln\frac{(1-y)(1-b)}{(1+y)(1+b)}\right)\ln(b+y)\right.\\
&&&\left. + \frac{2(b-y)}{(b^2-1)(y^2-1)} + \mathrm{Li}_2\left(\frac{1-y}{1+b}\right) - \mathrm{Li}_2\left(\frac{1+y}{1-b}\right)\right]\\
\mathrm{M}(y) &:= \int^y t^2(1-t^2)^{-3}\,dt &&= \frac{1}{16}\left[4y(y^2-1)^{-2} + 2y(y^2-1)^{-1} + \ln\frac{y-1}{y+1}\right]\\
\mathrm{G}(q)(y) &:= \int^y \ln q(t)\,dt &&= \frac{q'(y)}{2a}\ln q(y) - 2y + \frac{d}{a}\tan^{-1}\frac{q'(y)}{d}\\
\mathrm{H}(q)(y) &:= \int^y t\ln q(t)\,dt &&= \left(\frac{y^2}{2} + \frac{c}{2a} - \frac{b^2}{4a^2}\right)\ln q(y) - \frac{y(ay-b)}{2a} - \frac{bd}{2a^2}\tan^{-1}\frac{q'(y)}{d}
\end{aligned}
$$

**Table 1**: Four auxiliary integrals needed in the solution. Notice that $\mathrm{L}(b)(y)$ uses the dilogarithm [1], $\mathrm{Li}_2(z) = \sum_1^\infty \frac{z^k}{k^2}$, $\frac{d}{dz}\mathrm{Li}_2(z) = -\frac{\ln(1-z)}{z}$. In G and H the argument $q$ is an arbitrary quadratic polynomial $q(t) = at^2 + bt + c$ and $d = \sqrt{4ac - b^2}$.

$$
\begin{aligned}
c_0 &= \|E_j\| & c_{13} &= \frac{c_{11} - \sqrt{c_{11}^2 - 4c_{10}c_{12}}}{2c_{10}}\\
c_1 &= -2\vec{d}_i \cdot \vec{d}_j & c_{14} &= \frac{\sqrt{c_{11}^2 - 4c_{10}c_{12}}}{c_{10}}\\
c_2 &= \|E_i\| & c_{15} &= \sqrt{c_{10}}c_{14}\\
c_3 &= -2\vec{d}_j \cdot (\vec{p}_i - \vec{p}_j) & c_{16}(s) &= c_1c_{13} - c_3 - 2s\\
c_4 &= 2\vec{d}_i \cdot (\vec{p}_i - \vec{p}_j) & c_{17}(s) &= \frac{-c_{15} + \sqrt{c_{15}^2 - 4|c_{16}(s)|^2}}{2i\bar{c}_{16}(s)}\\
c_5 &= \|\vec{p}_i - \vec{p}_j\|^2 & c_{18}(s) &= \frac{-c_{15} - \sqrt{c_{15}^2 - 4|c_{16}(s)|^2}}{2i\bar{c}_{16}(s)}\\
c_{10} &= 4 - c_1^2\\
c_{11} &= 4c_4 - 2c_1c_3\\
c_{12} &= 4c_5 - c_3^2
\end{aligned}
$$

**Table 2**: All expressions for two edges $E_{ij}$ with parameterization $\vec{x}_i(t) = \vec{p}_i + t\vec{d}_i$ and $\vec{x}_j(s) = \vec{p}_j + s\vec{d}_j$ ($\|\vec{d}_{i,j}\| = 1$).

implementation requires some care because of the complexities of the functions that are involved.

A simple example, which requires the full power of our formula, concerns the form factor between two equal width rectangles sharing an edge with an enclosing angle $\theta \in [0, \pi]$. The configuration is illustrated in Figure 1 together with the form factor as a function of $\theta$ for different aspect ratios $l = \frac{a}{b}$ (common edge length $b$).

## 4 Conclusion

We have given a closed form solution for the form factor between two general polygons. This solution is non-elementary since it involves the dilogarithm function. The principal value of our solution is in determining exact answers for general polygonal configurations. This can be used in practice for reference solutions to check more efficient approximations. Baum et al. [2] have also shown that the error in the computed solution can be reduced significantly when using a closed form solution near singularities of the integrand.

There has been a long history of computing closed form expressions for form factors starting with Lambert in 1760. The literature lists many special cases for which closed form solutions exist, but hitherto no solution had been given for general polygonal configurations. The present paper closes this gap.

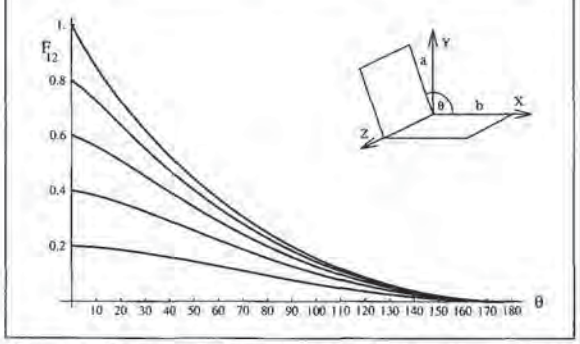



**Figure 1**: Geometry for two rectangles sharing a common edge with an enclosing angle of $\theta$. The graphs show the form factor as a function of $\theta$ for edge ratios $l = \frac{a}{b}$ of .2, .4, .6, .8, and 1.0.

## Acknowledgements

The first author would like to thank the Sci-Vis group at HLRZ for their support. Other support came from Apple, Silicon Graphics and the NSF (contract no. CCR 9207966).

## References

[1] Abramowitz, M., and Stegun, I. A. *Handbook of Mathematical Functions*, 9th ed. Dover Publications, 1970.

[2] Baum, D. R., Rushmeier, H. E., and Winget, J. M. Improving Radiosity Solutions Through the Use of Analytically Determined Form-Factors. *Computer Graphics 23*, 3 (July 1989), 325–334.

[3] Cohen, M. F., and Greenberg, D. P. The Hemi-Cube: A Radiosity Solution for Complex Environments. *Computer Graphics 19*, 3 (July 1985), 31–40.

[4] Goral, C. M., Torrance, K. E., Greenberg, D. P., and Battaile, B. Modelling the Interaction of Light between Diffuse Surfaces. *Computer Graphics 18*, 3 (July 1984), 212–222.

[5] Hanrahan, P., Salzman, D., and Aupperle, L. A Rapid Hierarchical Radiosity Algorithm. *Computer Graphics 25*, 4 (July 1991), 197–206.

[6] Herman, R. A. *A Treatise on Geometrical Optics*. Cambridge University Press, 1900.

[7] Lambert. *Photometria sive de mensura et gradibus luminis, colorum et umbrae*. 1760. German translation by E. Anding in *Ostwald's Klassiker der Exakten Wissenschaften*, Vol. 31-33, Leipzig, 1892.

[8] Nishita, T., and Nakamae, E. Continuous Tone Representation of Three-Dimensional Objects Taking Account of Shadows and Interreflection. *Computer Graphics 19*, 3 (July 1985), 23–30.

[9] Schröder, P., and Hanrahan, P. A Closed Form Expression for the Form Factor between Two Polygons. Tech. Rep. CS-404-93, Department of Computer Science, Princeton University, January 1993.

[10] Wallace, J. R., Elmquist, K. A., and Haines, E. A. A Ray Tracing Algorithm for Progressive Radiosity. *Computer Graphics 23*, 3 (July 1989), 315–324.

[11] Wolfram, S. *Mathematica*. Addison-Wesley, 1988.



# Reflection from Layered Surfaces due to Subsurface Scattering

Pat Hanrahan

Department of Computer Science
Princeton University

Wolfgang Krueger

Department of Scientific Visualization
German National Research Center
for Computer Science

## Abstract

The reflection of light from most materials consists of two major terms: the specular and the diffuse. Specular reflection may be modeled from first principles by considering a rough surface consisting of perfect reflectors, or micro-facets. Diffuse reflection is generally considered to result from multiple scattering either from a rough surface or from within a layer near the surface. Accounting for diffuse reflection by Lambert's Cosine Law, as is universally done in computer graphics, is not a physical theory based on first principles.

This paper presents a model for subsurface scattering in layered surfaces in terms of one-dimensional linear transport theory. We derive explicit formulas for backscattering and transmission that can be directly incorporated in most rendering systems, and a general Monte Carlo method that is easily added to a ray tracer. This model is particularly appropriate for common layered materials appearing in nature, such as biological tissues (e.g. skin, leaves, etc.) or inorganic materials (e.g. snow, sand, paint, varnished or dusty surfaces). As an application of the model, we simulate the appearance of a face and a cluster of leaves from experimental data describing their layer properties.

**CR Categories and Subject Descriptors:** I.3.7 [Computer Graphics]: *Three-Dimensional Graphics and Realism.*

**Additional Key Words and Phrases:** Reflection models, integral equations, Monte Carlo.

## 1 Motivation

An important goal of image synthesis research is to develop a comprehensive shading model suitable for a wide range of materials. Recent research has concentrated on developing a model of specular reflection from rough surfaces from first principles. In particular, the micro-facet model first proposed by Bouguer in 1759 [4], and developed further by Beckmann[1], Torrance & Sparrow[26], and others, has been applied to computer graphics by Blinn [2] and Cook & Torrance[8]. A still more comprehensive version of the model was recently proposed by He et al[12]. These models have also been extended to handle anisotropic microfacets distributions[24, 5] and multiple scattering from complex microscale geometries[28].

Another important component of surface reflection is, however, diffuse reflection. Diffuse reflection in computer graphics has almost universally been modeled by Lambert's Cosine Law. This law states that the exiting radiance is isotropic, and proportional to the surface irradiance, which for a light ray impinging on the surface from a given direction depends on the cosine of the angle of incidence. Diffuse reflection is qualitatively explained as due to subsurface scattering [18]: Light enters the material, is absorbed and scattered, and eventually exits the material. In the process of this subsurface interaction, light at different wavelengths is differentially absorbed and scattered, and hence is filtered accounting for the color of the material. Moreover, in the limit as the light ray is scattered multiple times, it becomes isotropic, and hence the direction in which it leaves the material is essentially random. This qualitative explanation accounts for both the directional and colormetric properties of diffuse materials. This explanation is also motivated by an early proof that there cannot exist a micro-facet distribution that causes equal reflection in all outgoing directions independent of the incoming direction [10].

The above model of diffuse reflection is qualitative and not very satisfying because it does not refer to any physical parameter of the material. Furthermore, there is no freedom to adjust coefficients to account for subtle variations in reflection from different materials. However, it does contain the essential insight: an important component of reflection can arise from subsurface scattering. In this paper, we present a model of reflection of light due to subsurface scattering in layered materials suitable for computer graphics. The only other work in computer graphics to take this approach is due to Blinn, who in a very early paper presented a model for the reflection and transmission of light through thin clouds of particles in order to model the rings of Saturn[2]. Our model differs from Blinn's in that it is based on one-dimensional linear transport theory—a simplification of the general volume rendering equation [19]— and hence is considerably more general and powerful. Of course, Blinn was certainly aware of the transport theory approach, but chose to present his model in a simpler way based on probabilistic arguments.

In our model the relative contributions of surface and subsurface reflection are very sensitive to the Fresnel effect (which Blinn did not consider). This is particularly important in biological tissues which, because cells contain large quantities of water, are translucent. A further prediction of the theory is that the subsurface reflectance term is not necessarily isotropic, but varies in different directions. This arises because the subsurface scattering by particles is predominantly in the forward direction. In fact, it has long been known experimentally that very few materials are ideal diffuse reflectors (for a nice survey of experiments pertaining to this question, see [18]).

We formulate the model in the currently emerging standard terminology for describing illumination in computer graphics [16, 11]. We also discuss efficient methods for implementation within the context of standard rendering techniques. We also describe how to construct materials with multiple thin layers. Finally, we apply the model to two examples: skin and leaves. For these examples, we build on experimental data collected in the last few years, and provide pointers to the relevant literature.

Another goal of this paper is to point out the large amount of recent work in the applied physics community in the application of linear transport theory to modeling appearance.



ACM-0-89791-601-8/93/008/0165 $01.50

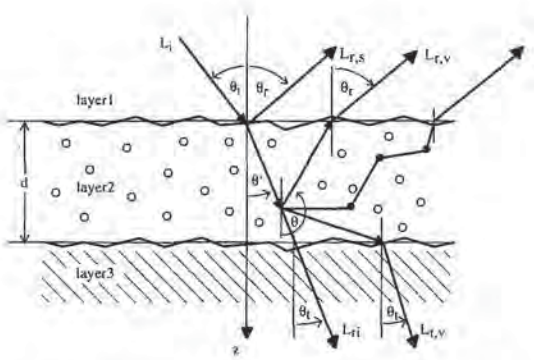



**Figure 1**: The geometry of scattering from a layered surface

| | |
|---|---|
| $(\theta_i, \phi_i)$ | Angles of incidence (incoming) |
| $(\theta_r, \phi_r)$ | Angles of reflection (outgoing) |
| $(\theta_t, \phi_t)$ | Angles of transmission |
| $L(z; \theta, \phi)$ | Radiance [W / (m$^2$ sr)] |
| $L_i$ | Incident (incoming) radiance |
| $L_r$ | Reflected (outgoing) radiance |
| $L_t$ | Transmitted radiance |
| $L_+$ | forward-scattered radiance |
| $L_-$ | backward-scattered radiance |
| $f_r(\theta_i, \phi_i; \theta_r, \phi_r)$ | BRDF |
| $f_t(\theta_i, \phi_i; \theta_t, \phi_t)$ | BTDF |
| $f_{r,s}(\theta_i, \phi_i; \theta_r, \phi_r)$ | Surface or boundary BRDF |
| $f_{t,s}(\theta_i, \phi_i; \theta_t, \phi_t)$ | Surface or boundary BTDF |
| $f_{r,v}(\theta_i, \phi_i; \theta_r, \phi_r)$ | Volume or subsurface BRDF |
| $f_{t,v}(\theta_i, \phi_i; \theta_t, \phi_t)$ | Volume or subsurface BTDF |
| $n$ | Index of refraction |
| $\sigma_s(z; \lambda)$ | Scattering cross section [mm$^{-1}$] |
| $\sigma_a(z; \lambda)$ | Absorption cross section [mm$^{-1}$] |
| $\sigma_t(z; \lambda)$ | Total cross section ($\sigma_t = \sigma_a + \sigma_s$) [mm$^{-1}$] |
| $W$ | Albedo ($W = \frac{\sigma_s}{\sigma_t}$) |
| $d$ | Layer thickness [mm] |
| $p(z; \theta, \phi; \theta', \phi'; \lambda)$ | Scattering phase function ($(\theta', \phi')$ to $(\theta, \phi)$) |

**Table 1**: Nomenclature

## 2 Reflection and Transmission due to Layered Surfaces

As a starting point we will assume that the reflected radiance $L_r$ from a surface has two components. One component arises due to surface reflectance, the other component due to subsurface volume scattering. (The notation used in this paper is collected in Table 1 and shown diagramatically in Figure 1.)

$$L_r(\theta_r, \phi_r) = L_{r,s}(\theta_r, \phi_r) + L_{r,v}(\theta_r, \phi_r)$$

where:

$L_{r,s}$ - reflected radiance due to surface scattering

$L_{r,v}$ - reflected radiance due to volume or subsurface scattering

The models developed in this paper also predict the transmission through a layered surface. This is useful both for materials made of multiple layers, as well as the transmission through thin translucent surfaces when they are back illuminated. The transmitted radiance has two components. The first component is called the *reduced intensity*; this is the amount of incident light transmitted through the layer without scattering inside the layer, but accounting for absorption. The second is due to scattering in the volume.

$$L_t(\theta_t, \phi_t) = L_{ri}(\theta_t, \phi_t) + L_{t,v}(\theta_t, \phi_t)$$

where:

$L_{ri}$ - reduced intensity

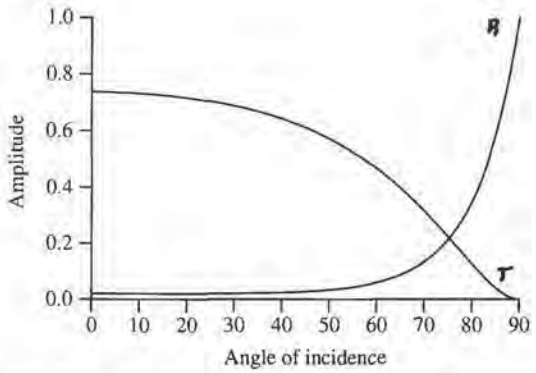



**Figure 2**: Fresnel transmission and reflection coefficients for a ray leaving air ($n = 1.0$) and entering water ($n = 1.33$).

$L_{t,v}$ - transmitted radiance due to volume or subsurface scattering

The bidirectional reflection-distribution function (BRDF) is defined to the differential reflected radiance in the outgoing direction per differential incident irradiance in the incoming direction [23].

$$f_r(\theta_i, \phi_i; \theta_r, \theta_r) \equiv \frac{L_r(\theta_r, \phi_r)}{L_i(\theta_i, \phi_i) \cos\theta_i d\omega_i}$$

The bidirectional transmission-distribution function (BTDF) has a similar definition:

$$f_t(\theta_i, \phi_i; \theta_t, \theta_t) \equiv \frac{L_t(\theta_t, \phi_t)}{L_i(\theta_i, \phi_i) \cos\theta_i d\omega_i}$$

Since we have separated the reflected and transmitted light into two components, the BRDF and BTDF also have two components.

$$\begin{aligned} f_r &= f_{r,s} + f_{r,v} \\ f_t &= f_{rt} + f_{t,v} \end{aligned}$$

If we assume a planar surface, then the radiance reflected from and transmitted across the plane is given by the classic Fresnel coefficients.

$$\begin{aligned} L_r(\theta_r, \phi_r) &= R^{12}(n_i, n_t; \theta_i, \phi_i \rightarrow \theta_r, \phi_r) L_i(\theta_i, \phi_i) \\ L_t(\theta_t, \phi_t) &= T^{12}(n_i, n_t; \theta_i, \phi_i \rightarrow \theta_t, \phi_t) L_i(\theta_i, \phi_i) \end{aligned}$$

where

$$\begin{aligned} R^{12}(n_i, n_t; \theta_i, \phi_i \rightarrow \theta_r, \phi_r) &= R(n_i, n_t, \cos\theta_i, \cos\theta_t) \\ T^{12}(n_i, n_t; \theta_i, \phi_i \rightarrow \theta_t, \phi_t) &= \frac{n_t^2}{n_i^2} T = \frac{n_t^2}{n_i^2}(1 - R) \end{aligned}$$

where $R$ and $T$ are the Fresnel reflection formulae and are described in the standard texts (e.g. Ishimura[14]) and $\theta_t$ is the angle of transmission. Besides returning the amount of reflection and transmission across the boundary, the functions $R^{12}$ and $T^{12}$, as a side effect, compute the reflected and refracted angles from the Reflection Law ($\theta_r = \theta_i$) and Snell's Law ($n_i \sin\theta_i = n_t \sin\theta_t$). Note also the factor of $(n_t/n_i)^2$) in the transmitted coefficient of the above formula; this arises due to the change in differential solid angle under refraction and is discussed in Ishimura[pp. 154-155]. Plots of the Fresnel functions for the boundary between air and water are shown in Figure 2.

In our model of reflection, the relative contributions of the surface and subsurface terms are modulated by the Fresnel coefficients.

$$f_r = R f_{r,s} + T f_{r,v} = R f_{r,s} + (1 - R) f_{r,v}$$

Thus, an immediate prediction of the model is that reflection due to subsurface scattering is high when Fresnel reflection is low, since more light enters the surface layer. Notice in Figure 2 that the percentage of transmission is very high for a quite wide range of angles of incidence. Thus, the reflectance properties of materials impregnated with water or oil (dielectrics with low indices of refraction) are dominated by subsurface reflectance components at near perpendicular angles of incidence, and surface components at glancing angles of incidence.

Actually, light returning from the subsurface layers must refract across the boundary again. Thus, it will be attenuated by yet another Fresnel transmission factor. Recall that if light returns from a media with a higher index of refraction, then total internal reflection may occur. All light with an incident angle greater than the critical angle ($\theta_c = sin^{-1}\, n_i/n_t$) will not be transmitted across the boundary. By assuming an isotropic distribution of returning light, we can compute the percentage that will be transmitted and hence considered reflected. This sets an upper bound on the subsurface reflectance of $1 - (n_i/n_t)^2$ (remember, $n_t > n_i$). For example, for an air-water boundary, the maximum subsurface reflectance is approximately .44.

## 3 Description of Materials

The aim of this work is to simulate the appearance of natural materials such as human skin, plant leaves, snow, sand, paint, etc. The surface of these materials is comprised of one or more layers of material composed of a mixture of randomly distributed particles or inhomogeneities embedded in a translucent media. Particle distributions can also exist, in which case the properties are the material are given by the product of each particle's properties times the number of particles per unit volume.

The layers of such materials can be described by a set of macroscopic parameters as shown in the following table. Measurements of these properties have been made for a large variety of natural materials.

| Symbol | Property |
|---|---|
| $n$ | index of refraction |
| $\sigma_a$ [mm$^{-1}$] | absorption cross section |
| $\sigma_s$ [mm$^{-1}$] | scattering cross section |
| $d$ [mm] | depth or thickness |
| $p(\cos j)$ | scattering phase function |
| $g$ | mean cosine of phase function |

- *Index of Refraction*
  The materials considered are dielectrics where $n$ is on the order of the index of refraction of water (1.33).
- *Absorption and scattering cross section*
  The intensity of the backscattered and transmitted light depends on the absorption and scattering properties of the material. The cross section may be interpreted as the probability per unit length of an interaction of a particular type. The total scattering cross section $\sigma_t = \sigma_a + \sigma_s$. The mean free path is equal to the reciprocal of the total cross section. An important quantity is the albedo, which equals $W = \sigma_s/\sigma_t$. If the albedo is close to 1, the scattering cross section is much greater than the absorption cross section, whereas if the albedo is close to 0, absorption is much more likely than scattering.
- *Scattering phase function*
  The phase function, $p(\vec{x}; \theta, \phi; \theta', \phi')$ represents the directional scattering from $(\theta', \phi')$ to $(\theta, \phi)$ of the light incident onto a particle. This function depends on the nature of the scattering medium. The form of $p$ is affected by the size, form and orientation of the suspended particles, the dielectric properties of the particles, and the wavelength of the incident light. The scattering of light from particles small compared to the wavelength of light is given by the Rayleigh scattering formula, and the scattering due to dielectric spheres of different radii by the Mie formula.

  However, most materials contain distributions of particles of many different sizes, so simple single particle phase functions are not applicable. For this reason, we describe the material phase function with the empirical formula, the Henyey-Greenstein formula[13].

$$p_{HG}(\cos j) = \frac{1}{4\pi} \frac{1 - g^2}{(1 + g^2 - 2g\cos j)^{3/2}}$$

  where $j$ is the angle between the incoming and the outgoing direction (if the phase function depends only on this angle the scattering is symmetric about the incident direction). The Henyey-Greenstein formula depends on a single parameter $g$, the mean cosine of the scattered light. The Henyey-Greenstein phase function for different values of $g$ is shown in Figure 3. Note that if $g = 0$ the scattering is isotropic, whereas positive $g$ indicates predominantly forward scattering and negative $g$ indicates predominantly backward scattering.

**Figure 3**: Henyey-Greenstein phase function for $g = -.3$ and $g = .6$.

In the model employed in this paper, material properties are described macroscopically as averages over the underlying microscopic material property definitions. If the material is made of several components, the resulting properties of the composite materials can be computed by simple summation.

$$\sigma_a = \sum_{i=1}^{n} w_i \sigma_{a,i}$$

$$\sigma_s\, p(\cos j, g) = \sum_{i=1}^{n} w_i \sigma_{s,i}\, p(\cos j, g_i)$$

and so on. Here $w_i$ is the volume fraction of the volume occupied by material $i$.

Another very important property of real materials is that the properties randomly vary or fluctuate. Such fluctuations cause variation in the appearance of natural surfaces. This type of fluctuation is easy to model with a random noise function or a texture map.

Optical propagation in random media has been studied in a variety of applications, including blood oximetry, skin photometry, plant physiology, remote sensing for canopies and snow, the paint and paper industry, and oceanic and atmospheric propagation. For many examples the macroscopic parameters have been measured across many frequency bands. A major attempt of our work is the simulation of the appearance of natural surfaces by using measured parameters to be inserted into the subsurface reflection and transmission formulas. This approach is similar to the attempt of Cook & Torrance [8] to simulate the appearance of metallic surfaces by using appropriate values for the refractive index and the roughness parameters.

## 4 Light Transport Equations

Linear transport theory is a heuristic description of the propagation of light in materials. Transport theory is an approximation to elec-

tromagnetic scattering theory, and hence cannot predict diffraction, interference or quantum effects. In particular, the specular reflection of light from rough surfaces whose height variation is comparable in size to the wavelength of incident light requires the full electromagnetic theory as is done in He et al[12]. A nice discussion of the derivation of transport theory from electromagnetism and the conditions under which it is valid is contained in an recent article by Fante[9]. The applicability of transport theory, however, has been verified by its application to a large class of practical problems involving turbid materials, including inorganic materials such as ponds, atmospheres, snow, sand and organic materials such as human skin and plant tissue[14].

Transport theory models the distribution of light in a volume by a linear integro-differential equation.

$$\frac{\partial L(\vec{x},\theta,\phi)}{\partial s} = -\sigma_t L(\vec{x},\theta,\phi) + \sigma_s \int p(\vec{x};\theta,\phi;\theta',\phi') L(\vec{x},\theta',\phi')\, d\theta'\, d\phi'$$

This equation is easily derived by accounting for energy balance within a differential volume element. It simply states that the change in radiance along a particular infinitesimal direction $ds$ consists of two terms. The first term decreases the radiance due to absorption and scattering. The second term accounts for light scattered in the direction of $ds$ from all other directions. Thus, it equals the integral over all incoming directions.

For layered media, the assumption is made that all quantities only depend on $z$ and not on $x$ and $y$. This assumption is valid if the incoming illumination is reasonably constant over the region of interest. It is also roughly equivalent to saying the reflected light emanates from the same point upon which it hits the surface. With this assumption, the above equation simplifies to

$$\cos\theta \frac{\partial L(\theta,\phi)}{\partial z} = -\sigma_t L(\theta,\phi) + \sigma_s \int p(z;\theta,\phi;\theta',\phi')\, L(\theta',\phi')\, d\theta'\, d\phi'$$

The above equation is an integro-differential equation. It can be converted to an equivalent double integral equation, whose solution is the same as the original integro-differential equation.

$$L(z;\theta,\phi) = \int_0^z e^{-\int_0^{z'} \sigma_t \frac{dz''}{\cos\theta}} \int \sigma_s(z')\, p(z';\theta,\phi;\theta',\phi')\, L(z';\theta',\phi')\, d\omega' \frac{dz'}{\cos\theta}$$

This is the basis of most current approaches to volume rendering.

The 1-dimensional linear transport equation must also satisfy certain boundary conditions. This is most easily seen by considering the forward and the backward radiance separately.

$$L(\theta,\phi) = L_+(\theta,\phi) + L_-(\pi-\theta,\phi)$$

Where $L_+$ is energy propagating in the positive $z$ direction, and $L_-$ in the negative direction. Note that $L_-$ is defined to be a function of of $\pi-\theta$, the angle between the backward direction of propagation and the negative $z$ axis. It is important to remember this convention when using formulas involving backward radiances.

At the top boundary the forward radiance is related to the incident radiance.

$$L_+(z=0;\theta',\phi') = \int f_{t,s}(\theta_i,\phi_i;\theta',\phi')\, L_i(\theta_i,\phi_i)\, d\omega_i$$

This simply states that the forward component of radiance entering the volume at the boundary is due to light transmitted across the surface. If we assume a planar surface and parallel incident rays, then $f_{t,s}$ equals the Fresnel transmission term times a $\delta$-function that picks up the appropriate angle of incidence.

$$L_+(z=0;\theta',\phi') = T^{12}(n_i,n_t;\theta_i,\phi_i \rightarrow \theta',\phi') L_i(\theta_i,\phi_i)$$

In the more general case of a rough surface, $f_{t,s}$ is given by a transmission coefficient times the probability that light will refract in the desired direction.

The boundary conditions at the top let us formally state the contribution to reflection due to subsurface scattering in terms of the solution of the integral equation at the boundary $z=0$.

$$L_{r,v}(\theta_r,\phi_r) = \int f_{t,s}(\theta,\phi;\theta_r,\phi_r)\, L_-(z=0;\theta,\phi)\, d\omega$$

Assuming a planar surface, this integral simplifies to

$$L_{r,v}(\theta_r,\phi_r) = T^{21}(n_i,n_t;\theta,\phi \rightarrow \theta_r,\phi_r) L_-(z=0;\theta,\phi)$$

Similar reasoning allows the transmitted radiance to be determined from the boundary conditions at the bottom boundary.

$$L_{t,v}(\theta_t,\phi_t) = \int f_{t,s}(\theta,\phi;\theta_t,\phi_t)\, L_+(z=d;\theta,\phi)\, d\omega$$

Once again, assuming a smooth surface,

$$L_{t,v}(\theta_t,\phi_t) = T^{23}(n_2,n_3;\theta,\phi \rightarrow \theta_t,\phi_t) L_+(z=d;\theta,\phi)$$

Thus, the determination of the reflection functions has been reduced to the computation of $L_-(z=0)$ and $L_+(z=d)$—the solution of the one-dimensional transport equation.

## 5 Solving the Integral Equation

There are very few cases in which integro-differential equations can be directly solved. The most famous solution is for the case of isotropic scattering and was derived by Chandrasekhar[7, p. 124]. Even for this simple phase function the solution is anisotropic.

The classic way to solve such an equation is to write it in terms of the Neumann series. Physically, this can be interpreted as expanding the solution in terms of the radiance due to an integer number of scattering events. That is,

$$L = \sum_{i=0}^{\infty} L^{(i)}$$

where $L^{(0)}$ is the direct radiance assuming no scattering, $L^{(1)}$ is the radiance due to a single scattering event, and $L^{(i)}$ is the radiance due to $i$ scattering events. Similar equations apply to the forward and backward radiances, $L_+^{(i)}$ and $L_-^{(i)}$.

The radiance due to the $i$ scattering events can be written using the following recurrence.

$$L^{(i+1)}(z;\theta,\phi) = \int_0^z e^{-\int_0^{z'} \sigma_t \frac{dz''}{\cos\theta}} \int \sigma_s(z')\, p(z';\theta,\phi;\theta',\phi')\, L^{(i)}(z';\theta',\phi')\, d\omega' \frac{dz'}{\cos\theta}$$

This is the basis for most iterative approaches for numerically calculating transport quantities.

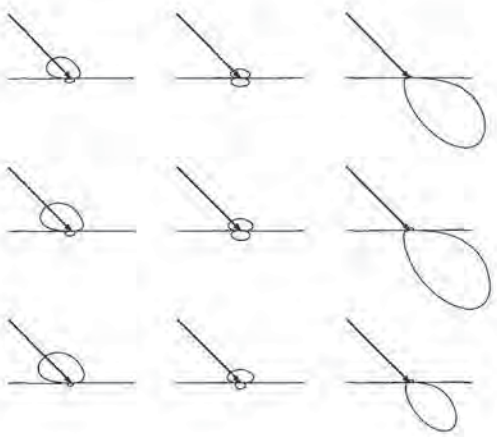


**Figure 4**: Solutions for $f_{r,v}^{(1)}$ and $f_{t,v}^{(1)}$ for different values of $g$ and $\tau_d$. From left to right the phase function shifts from predominately backward scattering ($g = -0.3$) to isotropic scattering ($g = 0.0$) to forward scattering ($g = 0.6$). From top to bottom the optical depth of the layer increases from 0.5 to 1.0 to 2.0.

### 5.1 First-Order Approximation

Another classic result in radiative transport, also derived by Chandrasekhar[7], is the analytic solution to the integral equation assuming only a single scattering event. As mentioned previously, this is equivalent to the method described by Blinn but derived using a completely different technique [2].

The 0th-order solution assumes that light is attenuated by the scattering and absorption, but not scattered. The attenuated incident light is called the *reduced intensity* and equals

$$L_+^{(0)}(z) = L_+(z=0)e^{-\tau/\cos\theta}$$

Here,

$$\tau(z) = \int_0^z \sigma_t \, dz$$

is called the *optical depth*. If $\sigma_t$ is constant, then $\tau_d = \sigma_t d$.

Using the boundary conditions for incident and reflected light, and also rewriting the above equation in terms of the angles of incidence and reflection, we arrive at the following formula for the 0th-order transmitted intensity

$$L_{t,v}^{(0)}(\theta_t, \phi_t) = T^{12}T^{23}e^{-\tau_d} L_i(\theta_i, \phi_i)$$

By substituting the 0th-order solution, or reduced intensity, into the integral equation, the 1st-order solutions for forward and backward scattering can be calculated. The details of this calculation are described in Chandrasekhar and Ishimura and there is no need to repeat them here.

Using the boundary conditions for incident and reflected light, and also rewriting in terms of the angles of incidence and reflection, we arrive at the following formula for the backscattered radiance:

$$L_{r,v}^{(1)}(\theta_r, \phi_r) = W T^{12} T^{21} p(\pi - \theta_v, \phi_r; \theta_i, \phi_i) \frac{\cos\theta_i}{\cos\theta_i + \cos\theta_r} (1 - e^{-\tau_d(1/\cos\theta_i + 1/\cos\theta_r)}) L_i(\theta_i, \phi_i)$$

This general formula shows that the backscattered light intensity depends on the Fresnel transmission coefficients, the albedo, the layer depth, and the backward part of the scattering phase function.

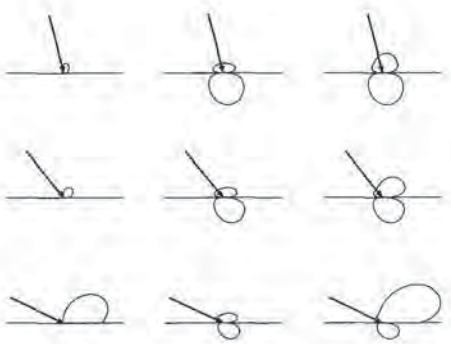


**Figure 5**: Solutions for $f_r$ and $f_t$. In the left column is the surface specular reflection and in the middle is the subsurface reflection and transmission. On the right is the sum of surface and subsurface modulated by the Fresnel coefficients. From top to bottom the angle of incidence increases from 10 to 40 to 65 degrees.

A special case of this equation is Seeliger's Law, the first attempt to model diffuse reflection from first principles[25]. Seeliger's Law can be derived by assuming a semi-infinite layer ($\tau_d = \infty$) and ignoring Fresnel effects.

$$L_{r,v}(\theta_r, \phi_r) = \frac{\cos\theta_i}{\cos\theta_i + \cos\theta_r} L_i(\theta_i, \phi_i)$$

At the boundary $z = d$, the forward scattered radiance is given by

$$L_{t,v}^{(1)}(\theta_t, \phi_t) = W T^{12} T^{23} p(\theta_t, \phi_t; \theta_i, \phi_i) \frac{\cos\theta_i}{\cos\theta_i - \cos\theta_t} (e^{-\tau_d/\cos\theta_i} - e^{-\tau_d/\cos\theta_t}) L_i(\theta_i, \phi_i)$$

For $\cos\theta_t = \cos\theta_i$, the singular factors can be avoided by using L'Hospital's rule, yielding

$$L_{t,v}^{(1)}(\theta_t, \phi_t) = W T^{12} T^{23} p(\theta_t, \phi_t; \theta_t, \phi_t) \frac{\tau_d}{\cos\theta_t} e^{-\tau_d/\cos\theta_t} L_i(\theta_t, \phi_t)$$

Figure 4 shows $f_{r,v}$ and $f_{t,v}$ for various values of $g$ and $d$. Figure 5 shows the surface and subsurface components of the reflection model for various angles of incidence. These reflection and transmission distribution functions have several interesting properties:

1. The reflection steadily increases as the layer becomes thicker; in contrast, the transmission due to scattering increases to a point, then begins to decrease because of further scattering events.
2. Subsurface reflection and transmission can be predominately backward or forward depending on the phase function.
3. As the angle of incidence becomes more glancing, the surface scattering tends to dominate, causing both the reflection and the transmission due to subsurface scattering to decrease.
4. Due to the Fresnel effect, the reflection goes to zero at the horizons. Also, the reflection function appears "flattened" relative to a hemicircle. Thus, reflection for near normal angles of incidence varies less than Lambert's Law predicts.
5. The distributions vary as a function of reflection direction. Lambert's Law predicts a constant reflectance in all directions (which would be drawn as a hemicircle in these diagrams).

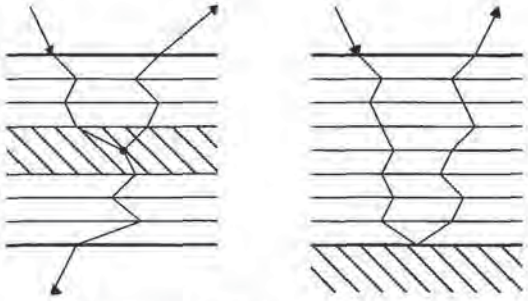


**Figure 6**: Determining first-order solutions for multiple layers. On the left, the contribution to the first order solution for a single layer. One the right, the contribution to the first order solution due to reflectance off a single layer.

The above formulas can be used to generate first-order solutions for multiple layers. (This is shown diagrammatically in Figure 6.) The total first-order scattering will be the sum of the first-order scattering from each layer, weighted by the percentage of light making it to the layer and returning from the layer. The percentage of light making it to the layer is the product of the 0th-order transmission functions (or reduced intensity) for a path through the layers above the reflecting layer. Similarly, the percentage of light leaving the entire layer after reflection is equal to the product of the 0th-order transmission functions for the path taken on the way out. Note that across each boundary the light may refract, and thus change direction and be attenuated by the Fresnel coefficient, but this is easy to handle. The process simplifies, of course, if each layer has the same index of refraction, since no reflection or change of direction occurs between layers. Given the above formulas it is very easy to construct a procedure to perform this calculation and we will make use of it in the results section.

The above formula can also be generalized to include reflection from a boundary between layers. In many situations reflection can only occur from the bottom layer. In this case, we add a single term accounting for the reduced intensity to reach the lower boundary, and also weight the returning light from that boundary. Such a model is commonly employed to model the reflection of light from a pool of water[15], and has been employed by Nishita and Nakamae[22]. Further generalizations of this type are described in Ishimura[14, p. 172].

## 6 Multiple Scattering

The above process of substituting the ith-order solution and then computing the integral to arrive at the (i+1)th-order solution can be repeated, but is very laborious. Note that subsequent integrals now involve angular distributions, because, although the input radiance is non-zero in only a single direction, the scattered radiance essentially comes from the directional properties of the phase function. Thus, this approach to solving the system analytically quickly becomes intractable.

We have implemented a Monte-Carlo algorithm for computing light transport in layered media. This algorithm is described in Figure 7. A thorough discussion of the application of Monte Carlo algorithms for layered media is discussed in the book [21], and the techniques we are using are quite standard.

To investigate the effects of multiple scattering terms, we simulated a semi-infinite turbid media with different albedos. The reflectance was computed and when the particles returning from the media are scored, we keep track of how many scattering events they underwent. Figure 8 shows the results of this experiment. The top curve is the total reflectance, and the lower curves represent scattering up to some order. Note that when the albedo is high, implying that $\sigma_s >> \sigma_a$, the first order term is only a small percentage of the total reflectance. However, as the albedo decreases, corresponding to greater absorption, a few low-order terms accurately approximate the reflectance. This effect can be explained by recalling that each term in the Neumann series representing the reflection is on the order of $W^i$, and since $W$ is always less than one, the magnitude of higher-order terms quickly goes to zero.

We have also computed the BRDF as a function of the angle of reflection using our Monte Carlo algorithm for the same configuration as described in the last experiment. The results are shown in Figure 9. Recall that the 1st-order reflection due to a semi-infinite media is given by Seeliger's Law: $\cos\theta_i/(\cos\theta_i + \cos\theta_r)$. The computed 1st-order BRDF matches the theoretical result quite well. In this figure we also plot the total BRDF due to any number of scattering events, and the difference between the total and the 1st-order BRDF. Note as in the previous experiment when the albedo $W$ is small, the BRDF is closely approximated by the 1st-order term. However, note that the shape of the reflection function is also largely determined by the shape of the 1st-order reflection, which in turn is largely determined by the phase function. Fur-

1 *Initialize*: A particle enters the layer at the origin. Initialize $\vec{p}$ to the origin and the direction $\vec{s}$ to the direction at which the ray enters the layer. Set the weight $w = 1$.

2 *Events*: Repeat the following steps until the ray weight drops below some threshold or the ray exits the layer.

**2A** *Step*: First, estimate the distance to the next interaction:

$$d = -\frac{\log r}{\sigma_t}$$

Where $r$ in this and the following formulas is a uniformly distributed random number between 0 and 1. Then, compute the new position:

$$\vec{p} = \vec{p} + d\,\vec{s}$$

And, finally set the particle weight to

$$w = w\frac{\sigma_s}{\sigma_s + \sigma_a}$$

Note: If $d$ causes the particle to leave the layer, break from the repeat loop and adjust the weight using the distance to the boundary.

**2B** *Scatter*: First, estimate the cosine of the scattering angle for the Henyey-Greenstein phase function using the following formula.

$$\cos j = \frac{1}{|2g|}(1 + g^2 - (\frac{1 - g^2}{1 - g + 2gr})^2)$$

and $\cos\phi$ and $\sin\phi$ with $\phi = 2\pi r$. Then, compute the new direction:

$$\vec{t} = \begin{pmatrix} (\vec{s}.x\cos\phi\cos\theta - \vec{s}.y\sin\phi)/\sin\theta \\ (\vec{s}.y\cos\phi\cos\theta + \vec{s}.x\sin\phi)/\sin\theta \\ \sin\theta \end{pmatrix}$$

$$\vec{s} = \vec{s}\cos j + \vec{t}\sin j$$

Here, $\cos\theta = \vec{s}.z$ and $\sin\theta = \sqrt{1 - \vec{s}.z^2}$. Note: Care must be taken if $\sin\theta = 0$.

3 *Score*: Divide the sphere into regions of equal solid angle and add the weight of the particle to the weight associated with the bin in which it is contained.

**Figure 7**: Basic Monte Carlo algorithm for layered media

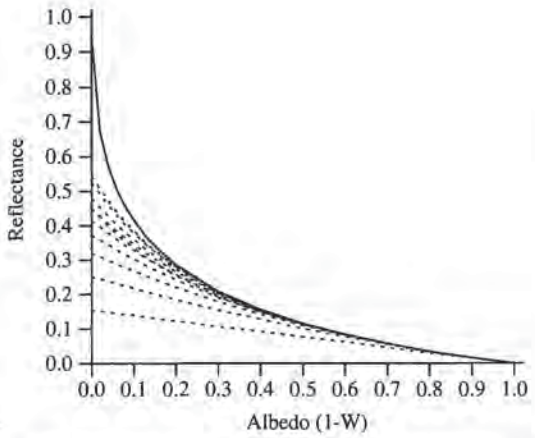



**Figure 8**: A plot of reflectance versus albedo for a semi-infinite media. The top curve is the total reflectance (the total radiant energy per unit area reflected divided by the incident irradiance). The bottom curve is the reflectance assuming only a single scattering event. Moving upward is a sequence of curves consisting of additional terms corresponding to a single additional scattering event. The first 10 terms in the solution are shown; In our simulations, we recorded terms involving thousands of scattering events.

ther, observe that the difference between the 1st-order solution and the full solution is approximately independent of the angle of reflection. Thus, the sum of the higher order terms roughly obeys Lambert's Law. For this reason it is often convenient to divide the subsurface reflection into two terms:

$$L_{r,v}(\theta_r, \phi_r) = L^{(1)}(\theta_r, \phi_r) + L^m$$

where $L^m$ is constant and represents the sum of all the multiple scattering terms.

Finally, we have begun preliminary experiments where we incorporate a Monte Carlo subsurface ray tracer within a standard ray tracer. When the global ray tracer calls the subsurface ray tracer it attempts to estimate the BRDF and BTDF to a particular light source. This is done by *biasing* the Monte Carlo procedure to estimate the energy transported to the light. A simple method to do this is to send a ray to the light at each scattering event, as described in Carter and Cashwell[6]. This ray must be weighted by the phase function and the attenuation caused by the traversal through the media on the way to the light. If the albedo is less than 1, then only a few scattering events are important, and thus the subsurface ray tracer consumes very little time on average (the cost is proportional the the mean number of scattering events). Also, since the subsurface ray tracer does not consider the global environment when tracing its rays, the cost of subsurface Monte Carlo simulation at every shading calculation is relatively low. The advantage of this approach is that the BRDF's do not have to precomputed, and so if material parameters are varying across the surface, the correct answer is still estimated correctly at each point.

## 7 Results

The subsurface scattering models developed in this paper has been tested on two common natural surfaces: human skin and plant leaves. The goal of these experiments are twofold: First, to compare our anisotropic diffuse reflection model with Lambertian shading. Second, to attempt to simulate the optical appearance from measured parameters. Our experiments are meant to be suggestive of the power of this approach; we do not claim to have an experimentally validated model.

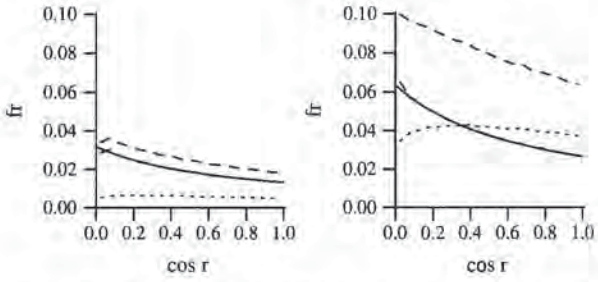



**Figure 9**: Graphs of the BRDF ($f_r$) as a function of the angle of reflection for a semi-infinite slab with different albedos (on the left $W = 0.4$ and on the right $W = 0.8$) and an angle of incidence of 45°. The solid line is the theoretical BRDF as given by Seeliger's Law (the superimposed dashed line is the computed 1st-order BRDF showing a good match). The top dashed curve is the total computed BRDF; The bottom dotted curve is the difference between the total BRDF due to multiple scattering events and the 1st-order BRDF.

| Property | Epidermis | Dermis | Pigment | Blood |
|---|---|---|---|---|
| $n$ | 1.37-1.5 | 1.37-1.5 | 1.37-1.5 | 1.37-1.5 |
| $\sigma_a$ [mm$^{-1}$] | 3.8 | 0.3 | | 32.6 |
| $\sigma_s$ [mm$^{-1}$] | 50.0 | 21.7 | | 0.96 |
| $d$ [mm] | 0.001-0.15 | 1-4 | | |
| $g$ | 0.79 | 0.81 | .79 | .0 |

**Table 2**: Two Layer Skin Model Properties. Pigment coefficients are mixed with epidermal coefficients to compute the properties of the outer layer. Blood coefficients are mixed with dermal coefficients to compute the properties of the inner layer.

### 7.1 Skin

Human skin can be modeled as two layers with almost homogeneous properties. Both layers are assumed to have the same refractive index but a different density of randomly distributed absorbers and scatterers. The outer epidermis essentially consists of randomly sized tissue particles and imbedded pigment particles containing melanin. The pigment particles act as strongly wavelength dependent absorbers causing a brown/black coloration as their density increases. The inner dermis is considered to be a composition of weakly absorbing and strongly scattering tissue material and of blood which scatters light isotropically and has strong absorption for the green and blue parts of the spectrum. Experimental evidence also supports the hypothesis that light scattering in the skin is anisotropic with significant forward scattering. A comprehensive study of optical properties of human skin can be found in van Gemert et al.[27]. The values chosen for our test pictures are given in Table 2. We also add a thin outer layer of oil that reflects light using the Torrance-Sparrow model of rough surfaces.

A head data set was acquired using a medical MRI scanner. Unfortunately, the ears and the chin were clipped in the process, but enough of the head is visible to test our shading models. A volume ray tracer was adapted to output the position and normal vector of the skin layer for each pixel into a file, and this input was used to evaluate the shading models described in this paper.

The influence of the various factors appearing in the subsurface reflection formula are shown on Plate 1. These pictures are

**Plate 1.**

**Plate 2.**

not shaded in the conventional way. In particular, a Lambertian shading model would yield a constant image. The first picture (upper left) shows the influence of the Fresnel factors. Observe that the intensity is almost flat, but strongly attenuated for glancing incident and viewing angles. The second picture (upper middle) shows the action of Seeliger's Law alone. Seeliger's Law leads to very little variation in shading, which makes the surface appear even more chalky or dusty. The third picture (upper right) demonstrates the action of the factor accounting for the finite layer depth giving only weak enhancements for glancing angles. This is a minor effect. The fourth picture (lower left) shows the influence of the Henyey-Greenstein scattering phase function for small backward scattering ($g = -.25$) and the fifth picture (lower middle) shows the effect of large forward scattering ($g = .75$). The result is strong enhancement of glancing reflection for low angles of incidence and viewing, assuming they are properly aligned. The last picture (lower right) shows the superposition of these four factors with $g = .75$ giving a complex behavior. An overall smoothing of the reflection appears; the surface appears to be more "silk-like" (see also Plate 3). Although these effects are all subtle, their combination when controlled properly can create a wide variation in appearance.

The appearance of the face with the new subsurface reflection model is compared to the Lambertian diffuse reflection model for different angles of incidence in Plate 2. The left column shows the results for the Lambert scattering for angles 0 and 45 degrees, and the middle column is rendered for the new model. Again,

**Plate 3:** Dark complexion controlled by setting the concentration of melanin. On the left are images with just subsurface scattering. On the right, an specular surface term is added to simulate an oily coat. In these pictures $g = .65$.

**Plate 4:** Human face with variation in subsurface blood concentration, an oily outer layer and Gaussian variation in parameters to create the "freckles."

notice a much smoother "silk-like" appearance. The right column gives the relative difference of both models, red indicates more reflection from the new model, and blue vice versa.

To illustrate the degrees of freedom of the model, we rendered several faces with their parameters controlled by texture maps. One texture map controls the relative concentration of blood in the dermis; another texture map controls the concentration of melanin in the epidermal layer. These faces are shown in Plates 3 and 4. To create a dark complexion we modulate the percentage of pigment in the otherwise transparent epidermis. This creates a dark brown appearance due to the strong absorption of melanin (in this case we set the absorption to .6). For the lips the epidermis is set to be very thin such that the appearance is dominated by the reflection from the dermis which has for the lips a large blood content (strong absorption for green and blue light component). The epidermis pigment part also has been varied locally with about 20% with a Gaussian process. This allows us to create a wide variety of skin colors, from black to suntanned to Caucasian, and from flushed to burnt to relaxed. The pictures in Plate 3 also show the effect of an additional specular term due to a thin layer of oil on the skin. Finally, Plate 4 shows another picture created by our program.

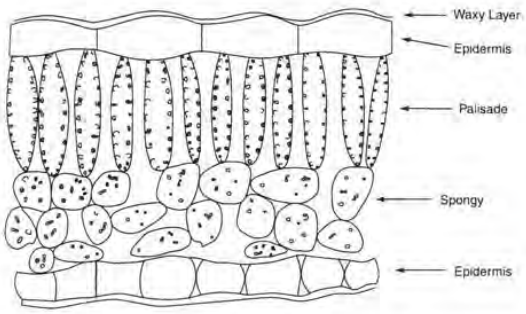



**Figure 10**: Typical leaf cross-section (Redrawn from [20]).

**Plate 5:** Leaf Model. On the left is the albedo image and on the right is a thickness image (white indicates thick)

This picture took approximately 20 seconds to render on a Silicon Graphics Personal Iris.

### 7.2 Leaves

Figure 10 shows an idealized leaf in cross-section. The leaf is composed of several layers of cells. On the top and bottom are epidermal cells with a thin smooth, waxy cuticular outer layer. The waxy cuticular layer is largely responsible for specularly reflected light. Below the upper epidermal cells there are a series of long palisaide cells which are highly absorbing due to the numerous chloroplasts contained within them. Below the palisaide cells are a loosely packed layer of irregularly shaped spongy cells. The spaces between the spongy cells are filled with air, which causes them to scatter light. Both the palisaide and the spongy cells are quite large (approximately 20 $\mu$m) compared to the wavelength, so their scattering phase function is forward directed. Furthermore, the cells are high in water content, so the index of refraction of the leaf is approximately equal to that of water—1.33. A typical leaf is .5 to 1 mm thick, with an optical depth of 5 to 10.

To test our model on a leaf, we constructed a leaf model using the technique described in Bloomenthal[3]. Although spectral transmission and reflectance curves are available for leaves[29], we have set the color of the leaf from an image acquired from a digital scanner. An albedo image is texture mapped onto a series of simply-shaped, bent polygons to create the leaf. Where the texture map is transparent the polygon is considered transparent and the leaf is not visible. We also modulate the thickness of the leaf with a thickness map drawn on top of the original leaf image. The texture maps we used are shown in Plate 5. The waxy cuticle is modeled using a rough specular surface with a specular exponent of 10. The interior of the leaf is modeled as a single homogeneous layer with an optical depth of 5 and a mean scattering cosine of .3[20].

Pictures were generated by modifying a conventional ray tracer

**Plate 6:** A cluster of leaves. A series of leaf images under different simulated lighting conditions. On the left are two backlit images, on the right, front lit.

to account for subsurface reflection and transmission. When a ray encounters a leaf, the BRDF and BTDF are evaluated for direct illumination from light sources. Shadow rays are cast to the light source, and if the ray stabs any other leaves the light intensity is attenuated by the 0th-order transmission function through each leaf. Plate 6 shows a picture of a cluster of leaves with the sun in different positions. Note that the reflection from leaves is largely determined by specular reflection due to the waxy cuticle; there is very little diffuse reflection and hence when the light source is on the same side of the leaf as the viewer, the leaf is quite dark. The transmission term, however, can be quite large, and therefore the leaves may actually be brighter when the are illuminated from behind. Note also that the increased thickness of the veins cause dark shadows to be cast on other leaves. The veins also appear dark when the leaf is back lit because they absorb more light, and bright when the leaf is front lit because their increased thickness causes more light to be reflected.

## 8 Summary and Discussion

We have presented a reflectance model consisting of two terms: the standard surface reflectance and a new subsurface reflectance due to backscattering in a layered turbid media. This model is applicable to biological and inorganic materials with low indices of refraction, because their translucent nature implies that a high percentage of the incident light enters the material, and so the subsurface reflection is quite large. This model incorporates directional scattering within the layer, so the resulting subsurface reflection is not isotropic. This model can be interpreted as a theoretical model of diffuse reflectance. Thus, this model predicts a directionally varying diffuse reflection, in contrast to Lambert's Law. However, if multiple scattering contributes significantly to the reflection, then the higher scattering terms contribute to a reflection function with roughly the same shape.

As in any model, our model makes many assumptions. The two most important are that the physical optics may be approximated with transport theory, and that the material can be abstracted into layered, turbid media with macroscopic scattering and absorption properties. An "exact" model of biological tissues would explicitly model individual cells, organelles and so on, in considerably more detail. The Monte-Carlo algorithm for simulating reflection by Westin et al.[28] is an example of such an approach. Although such an approach may seem more accurate, often the experimental data needed to describe the arrangements of these structures is simply not available, and so in the end the results may be difficult to validate. An advantage of the transport theory approach is that the parameters of the model often may be directly extracted from experimental data.

A legitimate criticism of our work is that we did not directly compare the predictions of our model with experiment. The predictions of our model and the influence of measured material parameters should be checked carefully. However, we believe that this model has many applications in computer graphics even if it does not perfectly predict measured reflection functions. The metaphor of layered surfaces is very easy for users to understand because is a natural way to describe phenomenologically the appearance of many materials. It also fits easily into most rendering systems and can be implemented efficiently.

Finally, transport theory is a heuristic theory based on abstracting microscopic parameters into statistical averages. Transport theory is also the basis of the rendering equation, which is widely viewed as the correct theoretical framework for global illumination calculations. In this paper we propose to model surface reflection from layered surfaces with transport theory. Thus, when our reflectance model for layered surfaces is incorporated into a ray tracer, there is a hierarchy of transport calculations being performed. Within this hierarchy, the lower level transport equation computes the reflectance for the higher level transport equation. When performing this calculation, the lower level transport equation uses as its initial conditions the values from the higher level transport solution. Thus the two levels are coupled in a very simple way. In fact, it is possible to reformulate transport theory entirely in terms of reflection functions, the result is an integral equation for the reflection function itself; in this formulation the radiance does not appear at all. Coupling transport equations at different levels of detail in this manner is a promising approach to tackling the problem of constructing representations with many different levels of detail as proposed by Kajiya[17].

## 9 Acknowledgements

We would like to thank Craig Kolb for his help with RayShade and the leaf pictures. We would also like to thank David Laur for his help with the color plates. This research was partially supported by Apple, Silicon Graphics Computer Systems, David Sarnoff Research Center, and the National Science Foundation (CCR 9207966).

## References

[1] BECKMANN, P., AND SPIZZICHINO, A. *The scattering of electromagnetic waves from rough surfaces.* Pergamon, Oxford, 1963.

[2] BLINN, J. F. Light Reflection Functions for Simulation of Clouds and Dusty Surfaces. *Computer Graphics 16*, 3 (July 1982), 21–29.

[3] BLOOMENTHAL, J. Modeling the Mighty Maple. *Computer Graphics 19*, 3 (July 1985), 305–311.

[4] BOUGUER, P. *The Gradation of Light.* University of Toronto Press, 1960.

[5] CABRAL, B., MAX, N., AND SPRINGMEYER, R. Bidirectional reflection functions from surface bump maps. *Computer Graphics 21*, 4 (July 1990), 273–281.

[6] CARTER, L., AND CASHWELL, E. *Particle Transport Simulation with the Monte Carlo Method.* Energy Research and Development Administration, 1975.

[7] CHANDRASEKHAR, S. *Radiative Transfer.* Dover, New York, 1960.

[8] COOK, R. L., AND TORRANCE, K. E. A Reflection Model for Computer Graphics. *ACM Transactions on Graphics 1*, 1 (1982), 7–24.

[9] FANTE, R. Relationship between Radiative Transport Theory and Maxwell's Equations in Dielectric Media. *J. Opt. Soc. Am. 71*, 4 (April 1981), 460–468.

[10] GRAWBOSKI, L. *Astrophysics J. 39* (1914), 299.

[11] HANRAHAN, P. From Radiometry to the Rendering Equation. *SIGGRAPH Course Notes: An Introduction to Radiosity* (1992).

[12] HE, X. D., TORRANCE, K. E., SILLION, F. X., AND GREENBERG, D. P. A Comprehensive Physical Model for Light Reflection. *Computer Graphics 25*, 4 (July 1991), 175–186.

[13] HENYEY, L. G., AND GREENSTEIN, J. L. Diffuse radiation in the galaxy. *Astrophysics J. 93* (1941), 70–83.

[14] ISHIMURA, A. *Wave Propagation and Scattering in Random Media.* Academic Press, New York, 1978.

[15] JERLOV, N. G. *Optical Oceanography.* Elsevier, Amsterdam, 1968.

[16] KAJIYA, J. Radiometry and Photometry for Computer Graphics. *SIGGRAPH Course Notes: State of the Art in Image Synthesis* (1990).

[17] KAJIYA, J. Anisotropic Reflection Models. *Computer Graphics 19*, 3 (July 1985), 15–22.

[18] KORTUM, G. *Reflectance Spectroscopy.* Springer-Verlag, Berlin, 1969.

[19] KRUEGER, W. The Application of Transport Theory to the Visualization of 3-D Scalar Fields. *Computers in Physics 5* (April 1991), 397–406.

[20] MA, Q., ISHIMURA, A., PHU, P., AND KUGA, Y. Transmission, Reflection and Depolarization of an Optical Wave For a Single Leaf. *IEEE Transactions on Geoscience and Remote Sensing 28*, 5 (September 1990), 865–872.

[21] MARCHUK, G., MIKHAILOV, G., NAZARALIEV, M., DARBINJAN, R., KARGIN, B., AND ELEPOV, B. *The Monte Carlo Methods in Atmospheric Optics.* Springer Verlag, Berlin, 1980.

[22] NAKAMAE, E., KANEDA, K., OKAMOTO, T., AND NISHITA, T. A Lighting Model Aiming at Drive Simulators. *Computer Graphics 24*, 4 (August 1990), 395–404.

[23] NICODEMUS, F. E., RICHMOND, J. C., AND HSIA, J. J. *Geometrical Considerations and Reflectance.* National Bureau of Standards, October 1977.

[24] POULIN, P., AND FOURNIER, A. A Model for Anisotropic Reflection. *Computer Graphics 24*, 4 (August 1990), 273–282.

[25] SEELIGER, R. *Munch. Akad. II. Kl. Sitzungsber 18* (1888), 201.

[26] TORRANCE, K. E., AND SPARROW, E. M. Theory of Off-Specular Reflection From Roughened Surfaces. *Journal of the Optical Society of America 57* (September 1967), 1104–1114.

[27] VAN GEMERT, M. F. C., JACQUES, S. L., STERENBERG, H. J. C. M., AND STAR, W. M. Skin Optics. *IEEE Transactions on Biomedical Engineering 36*, 12 (December, 1989), 1146–1154.

[28] WESTIN, S. H., ARVO, J. R., AND TORRANCE, K. E. Predicting Reflectance Functions from Complex Surfaces. *Computer Graphics 26*, 2 (July 1992), 255–264.

[29] WOOLLEY, J. T. Reflectance and Transmittance of Light by Leaves. *Plant Physiology 47* (1971), 656–662.



# Display of The Earth Taking into Account Atmospheric Scattering

**Tomoyuki Nishita** **Takao Sirai**
Fukuyama University
Higashimura-cho, Fukuyama, 729-02 Japan

**Katsumi Tadamura** **Eihachiro Nakamae**
Hiroshima Prefectural University
Nanatsuka-cho, Shoubara City, 727 Japan

**Abstract**

A method to display the earth as viewed from outer space (or a spaceship) is proposed. The intention of the paper is application to space flight simulators (e.g., reentry to the atmosphere) and the simulation of surveys of the earth (comparisons with observations from weather satellites and weather simulations); it is not for geometric modeling of terrains and/or clouds viewed from the ground, but for displaying the earth including the surface of the sea viewed from outer space taking into account particles (air molecules and aerosols) in the atmosphere and water molecules in the sea.

The major points of the algorithm proposed here are the efficient calculation of optical length and sky light, with lookup tables taking advantage of the facts that the earth is spherical, and that sunlight is parallel.

**CR Categories and Subject Descriptors:**
I.3.3 **[Computer Graphics]**:Picture/Image Generation
I.3.7 **[Computer Graphics]**: Three-Dimensional Graphics and Realism
**Key Words:** Earth, Atmospheric Scattering, Optical Length, Sky light, Color of Water, Photo-realism, Radiative Transfer

## 1 INTRODUCTION

Research on image synthesis of realistic 3-D models is one of the most popular fields these days. Displays of natural scenes such as mountains, trees, sea, clouds have been attractively rendered, and an image synthesis of the earth has also been developed. Images of the earth are widely used in movies or TV commercials, e.g., the CG library of earth images[6] was recently released for use in this field. These images, however, are focused on how to create attractive images without any requirement of physical based accuracy. However, physically-based images are required for the study of the simulation of surveys of the earth, such as observation from weather satellites in comparison to weather simulation, and flight simulators in space. The color of the earth when viewed from space varies according to the relationship between the view direction and the position of the sun. In the famous words of the astronaut, "the earth was blue". When we observe the earth from relatively close to the atmosphere, the atmosphere surrounding the earth appears as blue, and the atmosphere near the boundary of the shadow due to the sun appears red (i.e., sunset). The color of clouds also varies according to the sun's position. These phenomena are optical effects caused by particles in the atmosphere, and cannot be ignored. The color of the surface of the sea is not uniform, such as navy blue; it has various colors which depend on incident light to the sea and absorption/scattering effects due to water molecules.

This paper proposes an algorithm of physically-based image synthesis of the earth viewed from space. The method proposed here has the following advantages:

(1) Calculation of the spectrum of the earth viewed through the atmosphere; the earth is illuminated by direct sunlight and sky light affected by atmospheric scattering.

(2) Calculation of the spectrum of the atmosphere taking account of absorption/scattering due to particles in the atmosphere.

(3) Calculation of the spectrum on the surface of the sea taking into account radiative transfer of water molecules.

The major parts in 1) and 2) are concerned with the calculation of optical length and sky light. For these calculations, numerical integrations taking into account atmospheric scattering are required, but they are effectively solved by using several (various) lookup tables making good use of the facts that the shape of the earth is a sphere and that sunlight is a parallel light. For 3), we show that an analytical solution is available instead of numerical integrations.

In the following sections, the basic idea of the lighting model for rendering the color of the earth taking into account atmospheric scattering, rendering the color of clouds, and spectrum calculation of the sea is described. Finally, several examples are demonstrated in order to show the effectiveness of the method proposed here.

## 2 BASIC IDEAS

In order to render the earth, the following elements should be taken into account: a geometric model of the earth, the atmosphere (air molecules, aerosols), sea, clouds, and the spectrum of the sunlight.

This paper discusses rendering an algorithm of the earth, the atmosphere, sea, and clouds viewed from outer space or various positions within the atmosphere; the following optical characteristics should be considered:

(1) The color of the atmosphere: the atmosphere contains air molecules and aerosols, and scattered sunlight from those particles reaches the viewpoint; the intensity of the light reaching the viewpoint is obtained by integrating scattered light from every particle on the ray, and the light scattered from the atmosphere around the earth also reaches this viewpoint.

(2) The color of the earth's surface: the earth is illuminated by both direct sunlight and sky light. Sunlight is absorbed when light passes through the atmosphere, and sky light consists of light scattered by particles in the air. On the way, passing through the atmosphere the light is attenuated, and its spectrum changes.

 ACM-0-89791-601-8/93/008/0175 $01.50

(3) The color of the sea: sunlight reaching the surface of the sea is divided into reflected light at the surface and light scattered from water molecules. Both of them pass through the atmosphere and reach the viewpoint.

(4) The color of clouds: sunlight is scattered from particles of clouds, the scattered light is attenuated and reaches the viewpoint.

These phenomena should be simulated as precisely as possible in the calculation of the spectrum of the earth and the atmosphere. As we intend to concentrate on close views of the earth, the bumped terrain model of the earth is used instead of a simple sphere; the continents are modeled by 3D fractals, and the sea is expressed by a sphere consisting of some curved surfaces. Geometric models such as a spaceship are also dealt with.

For hidden surface removal, the scanline algorithm for free form surfaces developed by the authors is employed[11]; the surfaces are expressed by Bézier surfaces.

# 3 MODELING OF THE EARTH

Even though we may use a modeling in which the earth is treated as a sphere and the land is modeled by bump mapping, we consider the earth as having two components, land and sea: the sea consist of eight cubic Bézier patches, and the land consists of a set of curved surfaces.

The land data is made by mapping small patches onto the sphere, which are subdivided by using fractals after giving the altitude data for each mesh point overlapped onto a world map: the random midpoint displacement algorithm is employed as a fractal.

A scanned image of the map is used as the texture of the land. Therefore the color is not the real color of the earth.

# 4 SPECTRUM OF THE ATMOSPHERE

Previous work taking account scattering/absorption due to particles include; a) the display of Saturn's rings (reflective ice particles)[1], b) for light scattering from particles in the air, shafts of light caused by spot lights[12], and light beams passing through gaps in the clouds or through trees[8], c) scattered light due to nonuniform density particles such as clouds and smoke[12][4], d) sky color taking account atmospheric scattering[5]. In this paper we focus our discussion on the atmosphere. On this topic, Klassen[5] approximated the atmosphere as multiple layers of plane-parallel atmosphere with uniform density; however, this method results in a large error near the horizon. We discuss here a spherical-shell atmosphere with continuous variation of density in order to improve accuracy. Though his method can only render the color of the sky viewed from a point on the earth, the method discussed here can render the color of the atmosphere viewed from space.

The color of the atmosphere is much influenced by the spectrum of the sunlight, scattering/absorption effects due to particles in the air, reflected light from the earth's surface, and the relationship between the sun's position and the viewpoint (and direction). The sunlight entering the atmosphere is scattered/absorbed by air molecules and aerosol, and ozone layers. The characteristics of scattering depend on the size of particles in the atmosphere. Scattering by small particles such as air molecules is called Rayleigh scattering, and scattering by aerosols such as dust is called Mie scattering. Light is attenuated by both scattering and absorption.

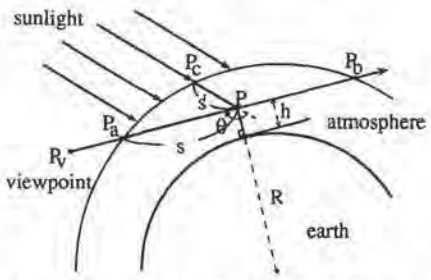



Figure 1: Intensity calculation for the ray intersecting only with the atmosphere.

## 4.1 Assumptions for Spectrum Calculation

For the spectrum calculation, we use the following assumptions:

(1) The multiple scattering of light between air molecules and aerosols in the atmosphere is ignored because of its negligible values and large computational cost, so only single scattering is considered. The interreflection of light between the earth's surface and particles in the air is also neglected because of the same reasons.

(2) For visible wavelengths, absorption in the ozone layer is negligible compared to absorption by air molecules and aerosols.

(3) The density distributions of air molecules and aerosols are taken into account; their densities vary exponentially with altitude[16].

(4) It is assumed that light travels in a straight line even though the actual path is curved due to the variation of index of refraction with altitudes.

## 4.2 Atmospheric Scattering

Let's consider scattering due to air molecules and aerosols.

First, single scattering due to air molecules is described. The light reflected due to Rayleigh scattering, $I$, is generally given by the following equation;

$$I(\lambda,\theta) = I_0(\lambda) K \rho F_r(\theta)/\lambda^4$$
$$K = \frac{2\pi^2(n^2-1)^2}{3N_s} \qquad (1)$$

where $I_o$ is the intensity of incident light, $K$ is a constant for the standard atmosphere (molecular density at sea level), $\theta$ the scattering angle (see Fig. 1), $F_r$ the scattering phase function indicating the directional characteristic of scattering ( given by $3/4(1+cos^2(\theta))$), $\lambda$ the wavelength of incident light, $n$ the index of refraction of the air, $N_s$ the molecular number density of the standard atmosphere, and $\rho$ the density ratio. $\rho$ depends on the altitude $h$ ($\rho = 1$ at sea level) and is given by

$$\rho = exp(\frac{-h}{H_0}), \qquad (2)$$

where $H_0$ is a scale height ($H_o = 7994$m), which corresponds to the thickness of the atmosphere if the density were uniform.

Eq. (1) indicates that the intensity of scattering is inversely proportional to the 4th power of the wavelength. Short wavelength light is very strongly attenuated by traversing the atmosphere, but long wavelength light is scarcely affected. This is why the sky appears blue in the daytime. Conversely, at sunset or sunrise, the distance traversed by the light increases, and the color of sky changes

to red because of increased scattering of short wavelengths. The attenuation coefficient $\beta$ (i.e., the extinction ratio per unit length) is given by

$$\beta = \frac{8\pi^3(n^2-1)^2}{3N_s\lambda^4} = \frac{4\pi K}{\lambda^4} \tag{3}$$

As shown in Fig.1, the light reaching viewpoint $P_v$ can be obtained as the remainder after scattering and absorption due to air molecules along the path between $P_b$ and $P_v$. The light at $P$ has been attenuated due to travel in the atmosphere ($P_cP$), and the light scattering from $P$ is also attenuated before reaching $P_v$.

To calculate the attenuation caused by particles for light of wavelength $\lambda$ traversing distance $s$, we use the optical depth, which is obtained by integrating $\beta$ of Eq. (3) along the path $s$. Let's denote the integration variable $s$ and the distance $S$, then the optical depth is given by

$$t(S,\lambda) = \int_0^S \beta(s)\rho(s)ds = \frac{4\pi K}{\lambda^4}\int_0^S \rho(s)ds \tag{4}$$

Next, single scattering due to aerosols is described. Scattering optics and the density distribution for aerosols differ from air molecules; Eq. (4) is different, too. Because the size range of particles of aerosols is very great, Mie scattering is applied for the phase function in Eq. (1) which exhibits a strong forward directivity. The Henyey-Greenstein function is well known as a phase function. Recently, Cornette[18] improved it, which gives a more reasonable physical expression:

$$F(\theta,g) = \frac{3(1-g^2)}{2(2+g^2)}\frac{(1+cos^2\theta)}{(1+g^2-2gcos\theta)^{3/2}}, \tag{5}$$

where $g$ is an asymmetry factor and given by

$$g = \frac{5}{9}u - (\frac{4}{3} - \frac{25}{81}u^2)x^{-1/3} + x^{1/3},$$
$$x = \frac{5}{9}u + \frac{125}{729}u^3 + (\frac{64}{27} - \frac{325}{243}u^2 + \frac{1250}{2187}u^4)^{1/2},$$

where if $g = 0$ then this function is equivalent to Rayleigh scattering. $u$ is determined by the atmospheric condition (e.g., haze) and wavelength; $u$ varies from 0.7 to 0.85(see [18]).

Like the density distribution of air molecules, the density of aerosols decreases exponentially with altitude, but the rate of decrease is different from that of air molecules. The density can be obtained by setting the scale height, $H_a$, of Eq. (2) to 1.2km[16].

### 4.3 Intensity Calculation due to Atmospheric Scattering

Let's discuss a ray from viewpoint $P_v$ to the earth, the light reaching the viewpoint has the following three passes: a) the ray passing through only the atmosphere, b) the ray intersecting with the earth, c) the ray passing through only space. For c) intensity calculation is not required. The calculation methods for a) and b) are described in the following.

#### 4.3.1 Spectrum calculation for only the atmosphere

Let's discuss light scattering due to air molecules on the ray passing just through the atmosphere. The discussion for aerosols is omitted because the optics is similar except for $1/\lambda^4$ dependence. As shown in Fig.1, the light reaching $P_v$ can be obtained as the remainder after scattering and absorption due to air molecules along the intersection line between the ray and the atmosphere, $P_bP_a$. The intensity of the light scattered at point P (at distance s from $P_v$) in the direction of $P_v$, $I_p$, is obtained by Eq.(1). The light scattered at $P$ is attenuated before arriving at $P_v$. The intensity of the light arriving at $P$, $I_p$, can be obtained by setting the integration interval to $P_cP$ in Eq. (4) of optical depth, that is

$$I_p(\lambda) = I_s(\lambda)KF_r(\theta)\rho\frac{1}{\lambda^4}exp(-t(PP_c,\lambda)), \tag{6}$$

where $I_s$ is the solar radiation at the top of the atmosphere, and $t(PP_c,\lambda)$ the optical depth from the top of the atmosphere to point P ($l$ is the integration variable) and given by

$$t(PP_c,\lambda) = \int_P^{P_c}\beta(l)\rho(l)dl.$$

As the light scattering from $P$ is also attenuated before reaching $P_v$, the intensity of the light reaching $P_v$, $I_{pv}$, can be obtained by multiplying the attenuation by the intensity at $P$, that is

$$I_{pv}(\lambda) = I_p(\lambda)exp(-t(PP_a,\lambda)). \tag{7}$$

As the distance to the sun can be considered almost infinite, the sunlight can be assumed to be a parallel beam. Thus the scattering angle at every point along $P_aP_b$ can be considered constant. That is, $I_v$ reaching $P_v$ can be obtained by integrating scattered light due to air molecules on $P_aP_b$:

$$I_v(\lambda) = \int_{Pa}^{Pb} I_{pv}(\lambda)ds$$
$$= I_s(\lambda)\frac{KF_r(\theta)}{\lambda^4}\int_{P_a}^{P_b}\rho\, exp(-t(PP_c,\lambda) - t(PP_a,\lambda))ds \tag{8}$$

#### 4.3.2 Spectrum calculation of the earth

Let's consider the ray intersecting with the earth as shown in Fig.2. The intensity scattered due to particles on the path, $P_aP_b$, can be obtained in the same manner as the description in 4.3.1. When point $P$ coincides with point $P_b$ (i.e., on the earth surface), the light reaching the viewpoint is obtained by adding reflected light from the earth to the light scattered due to molecules on $P_aP_b$. The intensity of light reaching viewpoint $P_v$, $I'_v$, is expressed by

$$I'_v(\lambda) = I_v(\lambda) + I_e(\lambda)exp(-t(P_aP_b,\lambda)), \tag{9}$$

where $I_v$ is the scattered light of Eq. (8). $I_e$ is reflected light at the earth;the direct component of sunlight and ambient light. The ambient light is mainly sky light. By considering

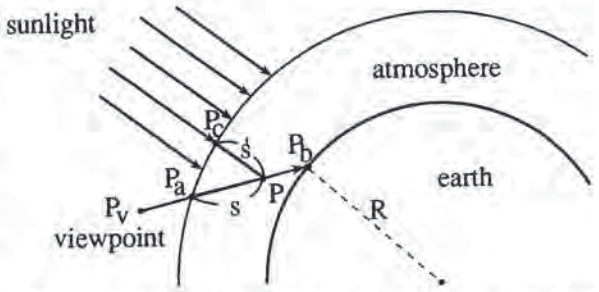



Figure 2: Intensity calculation for the ray intersecting with the earth.

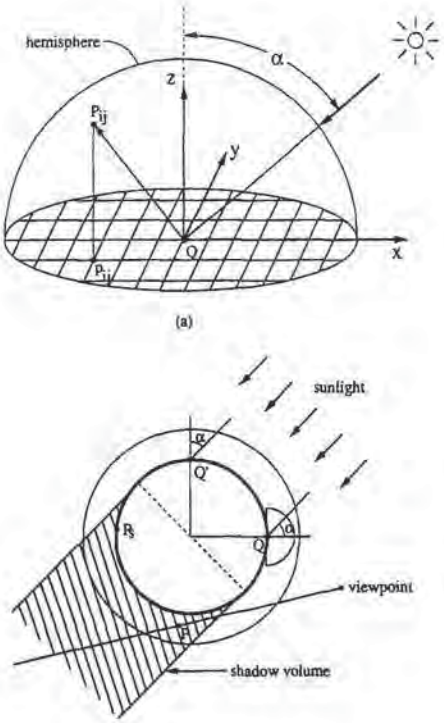



Figure 3: Calculation of sky light and shadow detection due to the earth.

attenuation of sunlight reaching the earth surface, $I_e$ is given by

$$I_e(\lambda) = r(\lambda)(cos\alpha\ I_s(\lambda)\ exp(-t(P_cP_b, \lambda)) + I_{sky}(\lambda, \alpha)), \tag{10}$$

where $r(\lambda)$ is the diffuse reflection of the earth, $\alpha$ the angle between the normal vector of the earth and light vector (sunlight), and $I_{sky}$ sky light. The direct component is small at the region where $\alpha$ is large (i.e., nearby the boundary of shadow) and tends to be reddish because of its long optical length.

Sky light is scattered light due to particles in the atmosphere. The radiance distribution of sky light can be obtained by setting the viewpoint on the earth in Eq.(8). As we are discussing the earth as viewed from space, shadows caused by obstacles on the surface are ignored, even though we take into account shadows due to the earth itself. That is, for shadow calculation, the earth is assumed to be a sphere with a smooth surface. Sky light due to scattered light from clouds is also ignored here. The illuminance at point $Q$ on the earth due to the whole sky is obtained by using the following method: let's consider an element on a hemisphere whose center is $Q$ (see Fig.3), calculate the intensity at each element on the hemisphere, and project each element onto the base of the hemisphere, then the illuminance is obtained by integrating the intensity of each element by weighting its projected area[13].

$I_{sky}$ is calculated as follows: as shown in Fig.3 (a), the base of the hemisphere is divided into a mesh. Let's consider point $P_{ij}$ on the hemisphere, which is mapped onto the hemisphere of the mesh point $p_{ij}$ inversely, and calculate the intensity in the direction of $QP_{ij}$. The illuminance due to the whole sky is obtained by adding intensities at every mesh point within the base circle of the hemisphere. As shown in Fig. 3(a), the $x$-axis is set so that the sun exists on the $x - z$ plane; the region in the half circle (e.g., $y > 0$) is enough to get $I_{sky}$ because of symmetry.

The radiance distribution of the sky is determined by angle $\alpha$ between the normal of the surface of the earth and the direction of the sunlight. Even though the direction of the sunlight is different at each point on the earth, the illuminance due to sky light (integrated values) at any point with the same angle $\alpha$ has the same value (e.g., $Q$ and $Q'$ in Fig.3). This means that the illuminance due to sky light at arbitrary angle $\alpha$ can be obtained by linear interpolation of a precalculated lookup table of $I_{sky}$. Note that $I_{sky}$ is not zero at regions where there is no direct sunlight ($\alpha > 90$ degrees, e.g., $P_s$ in Fig.3), so that $I_{sky}$ for $\alpha = 0$ to $\alpha = 110$ degrees must be prepared in the lookup table.

#### 4.3.3 Detection of shadow caused by the earth

As shown in Fig.3 (b), point $P$ on the ray exists in the shadow region caused by the earth (we refer to it as a *shadow volume*), the scattered light in this region is zero because there is no incident light. Therefore it is sufficient to consider only attenuation in this region.

As the shadow volume is expressed by a cylinder, which is obtained by sweeping the circle (i.e., the contour of the earth viewed from the sun), the shadow segment on the ray can be calculated as the intersection segment between the cylinder and the ray.

#### 4.3.4 Calculation of optical depth

The optical length of air molecules is calculated by numerical integration of Eq. (4) ( in the case of aerosols, the density distribution and the extinction coefficient are different). The optical length is calculated by trapezoidal integration of sampled density. The optical length at sampling point $P_i$ on the ray is obtained by adding the optical length of interval $P_{i-1}P_i$ to the optical length at $P_{i-1}$. Therefore the integration of the optical depth should start from the viewpoint. The optical length between the light source and point $P_i$ on the ray is also required (e.g., $PP_c$ in Fig.1). This calculation is required at every sampling point on the ray; optimization should be considered because of computational expense. We use a lookup table to save on computation time.

The density distribution of particles in the atmosphere varies exponentially with altitude. This means that the errors in the numerical integration become large when it is performed with a constant interval. Intervals which are inversely proportional to the density are desired; that is small intervals for low altitude and long intervals for high altitude. In order to realize this condition, the atmosphere is assumed as multiple spherical-shells. The radius of each sphere is set so that the difference in density between every adjacent sphere is within a given value. As a result, the difference between the radii of the shell is small for low altitude, and is large for high altitude, as shown in Fig.4. As Rayleigh scattering governs the calculation of optical length, the radius of each sphere is determined by the density distribution of air molecules. Let's consider N layers of spheres. The radius is given by(see Fig. 4)

$$r_i = H_0 log(\rho_i) + R, \quad \rho_i = 1. - i/N, \tag{11}$$

where $R$ is the radius of the earth. For $i = N$, $r_N$ is set to the radius of the atmosphere. For aerosols, the scale height is smaller than that for air molecules; aerosols mainly exist at low altitude. Therefore aerosols exist in the dense radii of shells; this fact assures the correctness of the above mentioned algorithm.

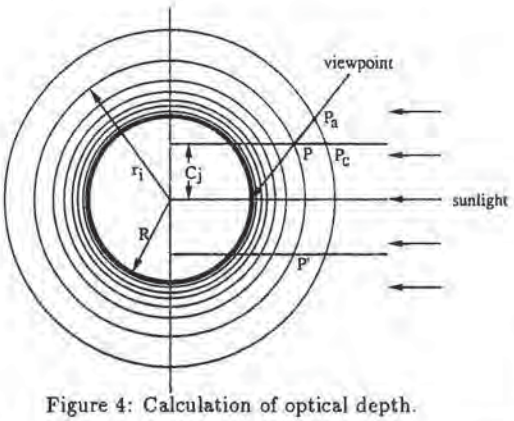



Figure 4: Calculation of optical depth.

The sampling points used in the integration are employed as the intersection points between the ray (view sight or light ray) and the multi-imaginary spheres and these intersection are easily obtained. The density at every sampling point is easily found from the lookup table indexed by the index numbers of the sphere, which is easily get from the altitude of the point.

The optical length between the sun and an arbitrary point on the ray can easily be precalculated because the earth is a sphere and sunlight is parallel light. As shown in Fig.4, let's consider a cylinder defined by sweeping the circle which passes through the center of the earth and is perpendicular to the light direction. Every optical length at the intersection (i.e. circle) between the cylinder and each one of the multi-imaginary spheres is equal (e.g., $P$ and $P'$ in figure). The optical lengths at the intersection points between the cylinders with radius $C_j$ and the spheres with radius $r_i$ is calculated(e.g.,$P_aP$ in fig.) and are stored in the lookup table. The optical depth at arbitrary point $P$ on the ray is easily calculated by linear interpolation, after the radius of the cylinder including $P$ and the radius of the sphere are calculated. The lookup table here is 2D array: $[r_i, C_j]$. After getting indeces $i$ and $j$ from point $P$, the optical depth can be obtained by linear interpolation from $[r_i, C_j]$,$[r_{i+1}, C_j]$, $[r_{i+1}, C_{j+1}]$,$[r_i, C_{j+1}]$.

As described above, the light intensity of one wavelength reaching the viewpoint can be calculated by numerical integration with respect to pass length. Therefore the light intensity in the range of visible wavelengths ( r, g, b in this paper) can be calculated.

## 5 THE COLOR OF CLOUDS

Since the geometric modeling of clouds is not our main subject, we are displaying the earth as viewed from space, clouds are simply modeled by applying 2D fractals. That is, the density distribution of clouds is expressed by mapping the fractal images of the necessary Mandelbrot set (0.39032+ 0.23775i is used in this paper)[15]. To take into account clouds with various altitudes, multiple imaginary spheres are employed to map fractal images on them.

Their color is determined by the following two light paths. One is on the light which passes through the atmosphere of scattered light due to cloud particles, again passing through the atmosphere, and reaches the viewpoint. Another one is on the light which passes through the atmosphere, reflected light at the earth's surface is attenuated by cloud particles, again passing through the atmosphere. Multiple scattering in clouds is ignored here.

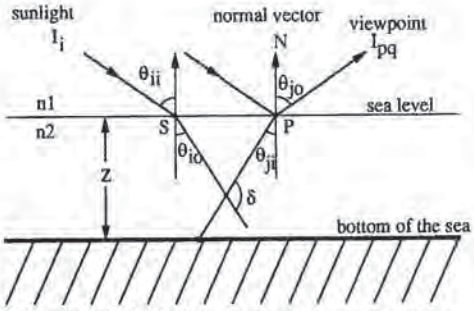



Figure 5: Calculation of color of water surface.

The size of particles in clouds is larger than that of air molecules or of aerosols. Light scattered by such large particles is little influenced by wavelength. ( However, the spectrum of incident sunlight onto clouds depends fairly strongly on the sun position.) The light reflected from clouds depends on the phase function ( the angle between the view vector and light vector); the phase function is expressed by Eq.(5) (see reference[18] on the value $u$). In the case of clouds not being illuminated by the sunlight because of the shadow due to the earth; the shadow detection is executed by using the shadow volume described before. The shadows on the earth due to clouds are ignored in this paper. In the near future, a more precise model for clouds is slated in order to get images of the earth viewed from relatively close to the earth's surface.

## 6 COLOR OF THE SEA

Let's consider the light reaching a viewpoint from the surface of the sea, There are three paths (see Fig. 5): (1) reflected light on the water surface, (2) scattered light due to particles within the water leaving the water surface (3) attenuated light passing through the sea after reaching the bottom of water.

Calculation methods of the color of water have been developed by Max[8], Fournier[2], Ts'o[17], and Mastin[7] . However their methods focused on (1) and shapes of waves, and did not refer to (2)(scattered light due to particles in the water). The method proposed here takes into account (1) and (2). Furthermore the attenuation of the light passing through the atmosphere is taken into account. For (3), the light from the bottom of the sea can be neglected because of the depth of the sea.

When the light is incident to the water surface, the light path is divided into reflection and refraction. The relation between the reflection and refraction on the water surface obeys Fresnel's law of reflection. Incident light is refracted at the water surface; the relation between the incident angle and reflection angle obeys Snell's law. The refracted light is scattered/absorbed by water molecules in the sea, and reaches the viewpoint after refracting at the water surface again. For this phenomena, Gordon and McCluney [3, 9] proposed a quasi-single-scattering (QSS) model based on the radiative transfer equation. However, in the model the sun's position is limited to the zenith. We improved upon this. The light intensity transmitted in water, $I_{PQ}$, is given by

$$I_{PQ}(\theta_{ii}, \theta_{io}, z) = \frac{I_i(\lambda)T_i(\theta_{ii}, \theta_{i0})T_o(\theta_{ji}, \theta_{jo})\beta(\delta, \lambda)}{n^2(\cos\theta_{io} + \cos\theta_{ji})c(\lambda)[1 - \omega_0(\lambda)F(\lambda)]}$$
$$\times(1 - exp(-zc(\lambda)[1 - \omega_0(\lambda)F(\lambda)](\sec\theta_{ji} + \sec\theta_{io})),$$

(12)

where $\lambda$ is wave length, $z$ the depth of the sea, $\theta_{ii}$ the angle between the surface normal at point $P$ and the direction of the viewing direction, $\theta_{io}$ the angle between the direction of the zenith and the direction of incident sunlight, $\theta_{jo}$ the angle between the reverse direction of the zenith and the sunlight after refraction, $I_i(\lambda)$ the irradiance of sunlight just above the water surface, $n$ the refractive index of water, $T_i$ and $T_o$ the transmittance of the incident light at point $S$ and $P$, respectively, $c(\lambda)$ the attenuation coefficient of light which expresses the ratio of lost energy of light when the light travels a unit length, $\beta$ a volume scattering function $\omega_0$ the albedo of water , and $F$ the fraction of the scattering coefficient in a forward direction. Data of $\beta, \omega_0$, and $F$ used in this paper is obtained from [10]. Eq. (12) shows that the color of water depends on the depth, the incident angles and viewing direction. The surface of the sea is not flat, and is a spherical surface (i.e., the normal vector of each point on the surface is different); the color of the sea varies according to the position because the incident and viewing angles to the surface normal at each position are different.

As described above, both the incident light to the sea and the color (intensity) of the sea are attenuated by the atmosphere. By using the same method as described in 4.3.2, this effect can be calculated by taking into account two optical lengths; from the sun to the surface and from the surface to the viewpoint.

# 7 EXAMPLES

Fig. 6 shows an example of the color of the atmosphere. The color of the earth is assumed to be black in order to demonstrate the atmospheric color only. The position of the sun is behind and to the left of the observation point. Even though the earth is assumed to be a black body, it looks blue, and the boundary of the earth is white.

Fig. 7 shows the images of the earth with texture-mapped continents viewed from space; the location of the observation is at altitude 36,000 km, which corresponds to the altitude of the Japanese weather satellite called *Himawari*, at 135°E 0° N and the direction of the sun is 70° E 20° N. In Fig. (a), the color of the sea, direct sunlight, and sky light are taken into account, but the attenuation from the earth to the viewpoint is ignored(i.e., it corresponds to the color when the observer stands on the earth). In Fig. (b), atmospheric scattering/absorption is also taken into account (i.e., the color of the atmosphere is added). In Fig. (c), clouds are added.

Figs. 8,9 show examples of the earth viewed from relatively close-by; the viewpoint is at altitude 500km at 0° E 60° N. The direction of the sun in Fig. 8 is 0° E 20N, and the directions of the sun in Fig. 9 are 200° E 20°N and 240° E 15°N. Fig. 8 corresponds to noon(daytime), and Fig. 9 correspond to evening or dawn sky. In Fig. 9(b), one can observe the shadow (the dark part in the red atmosphere) due to the earth. The color of clouds changes to red due to the change of color of direct sunlight. These examples depict beautiful variations in color of the earth and the atmosphere. The space shuttle in the figure consists of 178 Bézier patches.

Let's show the photographs taken by the first Japanese astronaut aboard space shuttle, Dr. M. Mouri(NASDA), in Fig.10 (altitude 300km, September, 1992). Fig.11 displays the results of our simulation. One may observe differences between the photos and the simulation results. One of the reasons on Fig.11(a) may be due to the poor modeling of clouds and lands. In Fig.(b) some horizontal layers(e.g., orange color) are observed, one of them may be aerosols due to explosion of Volcano in Philippine. These facts suggest the necessity for further researching.

For hidden surface removal, the scanline algorithm for curved surfaces [11] is employed, and for anti-aliasing the multi-scanning algorithm[14] is employed. The calculation was done on an IRIS Indigo Elan. The computation times for Fig.7 (c) and Fig. 9 were 3.8 minutes and 12.0 minutes, respectively(image size=500 x 490).

# 8 CONCLUSION

We have proposed an algorithm for physically-based image synthesis of the earth viewed from space. As shown in the examples, the proposed method gives us photo-realistic images taking into account the color of the earth, clouds, and the sea. The advantages of the proposed method are as follows:

(1) The spectrum of the surface of the earth is calculated by taking into account direct sunlight and sky light as affected by atmospheric scattering.
(2) The spectrum of the atmosphere is calculated by taking into account absorption/scattering due to particles in the atmosphere.
(3) The spectrum on the surface of the sea is calculated by taking into account radiative transfer of water molecules.
(4) The optical depth and illuminance due to sky light are efficiently calculated by using several lookup tables taking advantages of the facts that the earth is spherical and that sunlight is parallel.

**Acknowledgment** : The authors would like to acknowledge A. Wakayama (currently Fujitsu Co.) for his help in coding of the prototype of our program.

## References

[1] J.F. Blinn, "Light Reflection Functions for Simulation of Clouds and Dusty Surfaces," *Computer Graphics*, Vol. 16, No. 3 (1982),pp. 21-29.

[2] A. Fournier, " A Simple Model of Ocean Waves," *Computer Graphics*, Vol. 20, No. 4,(1986),pp. 75-84.

[3] H.R, Gordon, " Simple Calculation of the Diffuse Reflectance of the Ocean," *Applied Optics*, Vol. 12, No. 12,(1973),pp. 2803-2804.

[4] J.T. Kajiya, "Ray tracing Volume Densities," *Computer Graphics*, Vol.18, No.3 (1984) pp.165-174.

[5] RV Klassen, "Modeling the Effect of the Atmosphere on Light," *ACM Transaction on Graphics*, Vol. 6, No. 3,(1987),pp. 215-237.

[6] LINKS Corporation, leaflet of "LINKS CG LIBRARY", (1991)

[7] G. A. Mastin, P. A. Watterberg, and J. F. Mareda, " Fourier Synthesis of Ocean Scenes," *IEEE Computer Graphics & Applications*, Vol. 7, No. 3,(1987),pp. 16-23.

[8] N. Max, "Light Diffusion through Clouds and Haze," *Graphics and Image Processing*, Vol.33, No.3 (1986) pp.280-292.

[9] McCluney, W.R. Ocean Color Spectrum Calculations. *Applied Optics*, Vol. 13, No. 10,(1974),pp. 2422-2429.

[10] N. G. Jerlov, "Optical Oceanography," *Elsevier, Amsterdom* (1968).

[11] T. Nishita, K. Kaneda, E. Nakamae, " A Scanline Algorithm for Displaying Trimmed Surfaces by Using Bézier Clipping," *The Visual Computer*, Vol.7, No.5 (1991) pp.269-279.

[12] T. Nishita, Y. Miyawaki, E. Nakamae, "A Shading Model for Atmospheric Scattering Considering Distribution of Light Sources," *Computer Graphics*, Vol. 21, No. 4,(1987),pp. 303-310.

[13] T. Nishita, E. Nakamae, "Continuous tone Representation of Three-Dimensional Objects Illuminated by Sky Light," *Computer Graphics*, Vol. 20, No. 4,(1986),pp. 125-132.

[14] T. Nishita, E. Nakamae, "Half-Tone Representation of 3D Objects with Smooth Edge by Using a Multi-Scanning Method," *J.Information Processing(in Japanese)*, Vol.25, No.5,(1984), pp.703-711.

[15] K. Sato, "Fractal Graphics," *Rassel Co.(in Japanese)* (1989) p.74

[16] S. Sekine, "Optical characteristics of turbid atmosphere," *J Illum Eng Int Jpn*, Vol.71, No.6, (1987) pp.333

[17] P. Y. Ts'o, B. A Barsky,. " Modeling and Rendering Waves: Wave-Tracing Using Beta-Splines and Reflective and Refractive Texture Mapping," *ACM Transactions on Graphics*, Vol. 6, No. 3,(1987),pp. 191-214.

[18] W.M. Cornette, J.G. Shanks, "Physical reasonable analytic expression for the single-scattering phase function," *Applied Optics*, Vol.31, No.16 (1992), pp.3152-

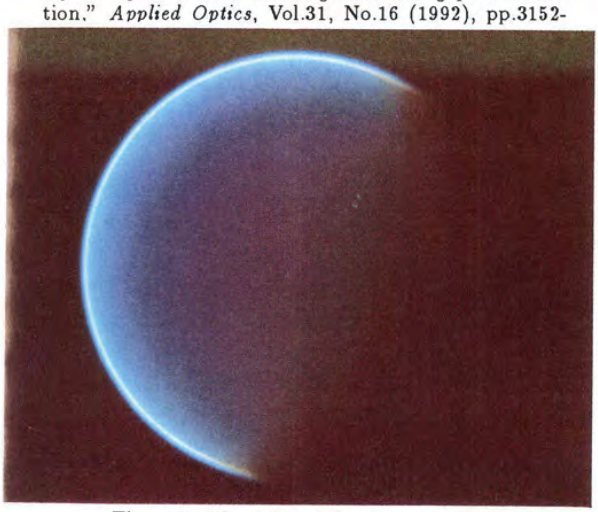
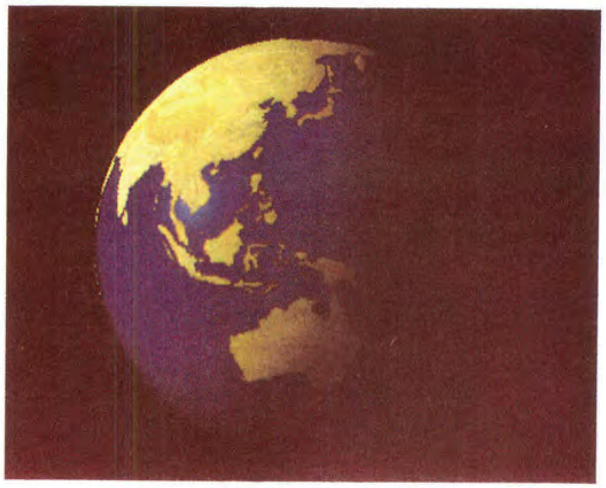


Figure 6: The color of the atmosphere.

Figure 8: The earth viewed from relatively close-by.

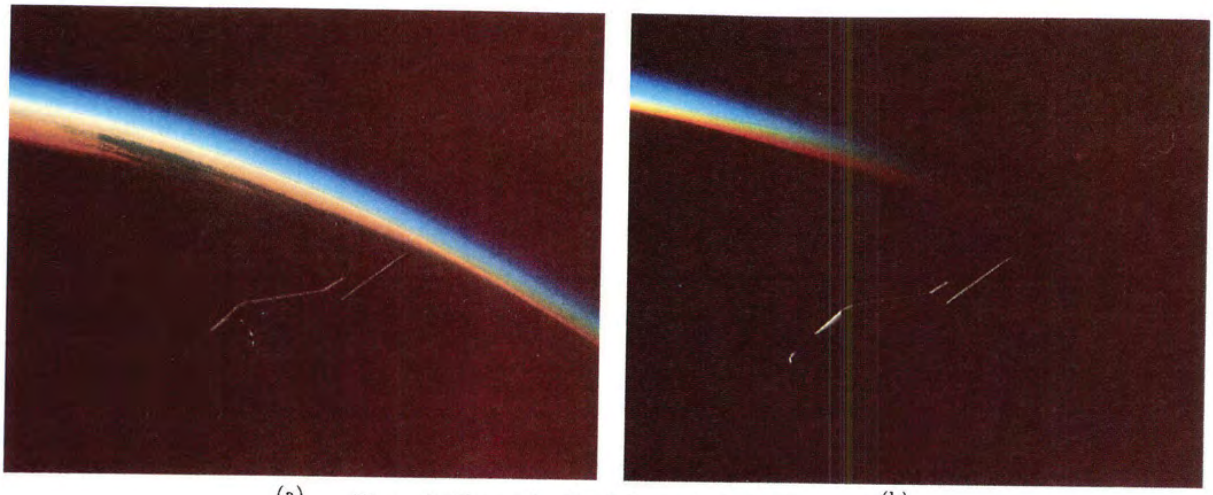


(a)

(b)

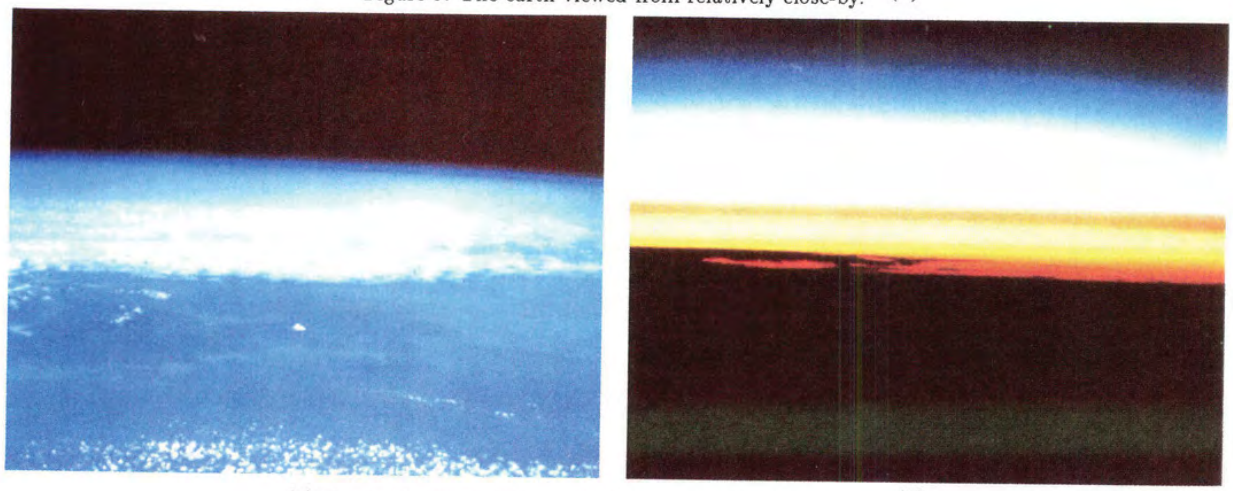
(c)

Figure 7: The earth viewed from space.



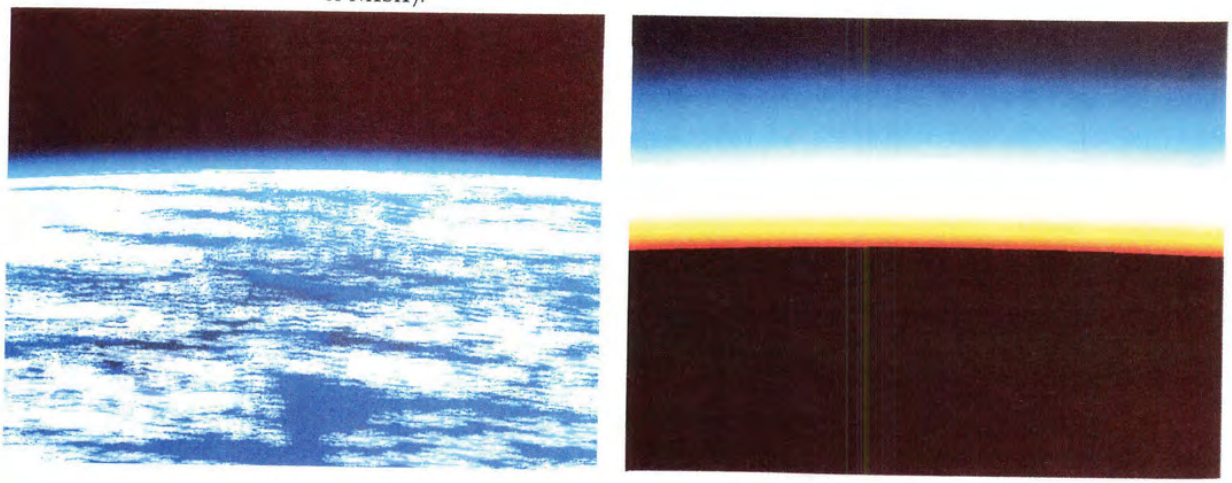
(a) (b)

Figure 9: The earth viewed from relatively close-by.



(a) (b)

Figure 10: Real photographs from space shuttle (courtesy of NASA).



(a) (b)

Figure 11: Comparisons with simulation.



# Smooth Transitions between Bump Rendering Algorithms

Barry G. Becker[1]
Nelson L. Max[2]

University of California, Davis
and
Lawrence Livermore National Laboratory

## ABSTRACT

A method is described for switching smoothly between rendering algorithms as required by the amount of visible surface detail. The result will be more realism with less computation for displaying objects whose surface detail can be described by one or more bump maps. The three rendering algorithms considered are a BRDF, bump-mapping, and displacement-mapping. The bump-mapping has been modified to make it consistent with the other two. For a given viewpoint, one of these algorithms will show a better tradeoff between quality, computation time, and aliasing than the other two. The decision as to which algorithm is appropriate is a function of distance, viewing angle, and the frequency of bumps in the bump map.

**CR Categories:** I.3.3 [Computer Graphics]: Picture/Image Generation; I.3.5 [Computer Graphics]: Three-Dimensional Graphics and Realism.

**Keywords:** animation, BRDF, bump map, displacement map, rendering, surface detail, volume texture.

## 1. INTRODUCTION

Objects in animation are sometimes distant specks; at other times a tiny part of one will fill the whole screen. If these objects have rough surfaces, the same rendering algorithm should not be used in both cases. Almost all real materials have a hierarchy of surface detail. We assume that the macro-structure of all objects is described by parameterized patches or a polygonal mesh. The micro-structure is then described by one or more bump tables for each level of detail below the geometrical, each giving bump height as a function of the 2-D surface parameters. An alternative way to describe the surface detail is through the use of volume textures to specify bump height as a function of 3-D coordinates[10, 12].

LLNL, P.O. Box 808/L-301, Livermore, CA 94550
1. (510)422-3724 becker@mozart.llnl.gov
2. (510)422-4074 max2@llnl.gov

The Bidirectional Reflection Distribution Function or BRDF[13, 14, 6] captures the surface properties which are too small to be visible. Most real surfaces are neither purely specular (mirror-like) nor purely diffuse, but rather somewhere in between. To represent this non-trivial distribution of light reflectance a BRDF is used. It can be represented by a table indexed by a lighting direction and a viewing direction, to give the reflectance as a function of these directions. The BRDF used for this research is constructed from distributions of normals recorded from various views of a single displaced surface patch.

Bump-mapping[2] is an inexpensive way to achieve a good approximation to macroscopic surface roughness. The parameterized surface is treated as smooth for the purpose of visible surface determination, while the surface normals are perturbed to a first order approximation of what the actual bump normals would be.

The third algorithm, displacement-mapping[4, 5], is used when any shortcut in computation will be noticeable to the eye. Displacement-mapping is different in that the surface is actually offset by the appropriate bump height so that the full 3-D geometry can be rendered. For purposes of maintaining consistent shading, the same approximated normal is used to shade the displaced surface as was used in the bump map. However, now it is applied to the displaced surface rather than to the flat parametric one.

Bump-mapping is good for economically rendering bumps which can be described as a height field. Unfortunately it does not account for occlusion. It is necessary to modify flat bump-mapping so that it yields images statistically similar to images produced by the other two methods. This revised procedure will be termed 'redistribution bump-mapping' because it redistributes the normals in a way that is statistically similar to those seen on the displaced surface viewed from a specific direction.

The three methods are blended together so that the parts of the scene which are close to the viewer, or close to the extreme edge (silhouette), would be displacement-mapped, since this is where missing detail would be noticed most. Smooth silhouette edges are an artifact of bump mapping which is easy to detect. Parts farther away, or whose normals are parallel to the viewing direction, will be bump-mapped. When surfaces have microscopic material-specific qualities or are very far from the viewer, they are rendered using a BRDF. More

specifically, for a given scene, those features with a spatial frequency higher than one half cycle per pixel (the Nyquist limit) are considered in the BRDF. At the other end of the spectrum, features that are large enough to cause noticeable occlusion need to be displacement-mapped. The parts in between are rendered with varying degrees of redistributed bump-mapping. Most importantly, there is a smooth transition among the three. The effect is that the whole scene looks as if it were displacement-mapped, when in fact much of it was rendered with cheaper algorithms. Extending this concept we can have high frequency rough surfaces on top of low frequency rough surfaces, each bumpy level of detail having three rendered representations.

In Figure 1 we see a teapot rendered in the four different ways. All renderings are based on the same height function. A major consideration for a smooth transitions among these is the consistency of the shading between methods. The amount of light emitted by a surface rendered with one method does not necessarily equal that amount emitted by the same surface rendered with another. Nor is the distribution of that light necessarily equivalent. A key aspect of this research is the determination of how the varying algorithms need to be modified in order to have their overall area-averaged light intensity contributions consistent.

There are five reasons why the average reflected intensity from a bump-mapped image is inconsistent with the reflected intensity from either the BRDF rendered image or the displacement-mapped image of the same object. Usually the BRDF is constructed under the assumption that the microfeatures of the surface are composed entirely of specular, mirrored facets. Bump- and displacement-mapping contain both specular and diffuse components. The easy solution to this inconsistency is to include a diffuse component for each microfacet when constructing the BRDF for the highest frequency bumps. Usually there is an inconsistency between bump- and displacement-mapping because actual surface displacement creates a geometrically computed facet normal for the shader while the perturbed normals for bump maps are only approximations. As previously mentioned this is overcome by using the approximated bump-mapped normals on the displaced surface. The approximated bump normals also vary more smoothly than the facet normals, especially with our quadratic interpolation, which is smoother than Blinn's approximation[2]. Note that if a procedural displacement function is employed, it is possible to compute the surface normal analytically. Since the BRDF is constructed from a displacement-mapped patch, the same inconsistency may arise for it. Again the solution is remedied by using the bump normal for tabulating the BRDF. The most difficult consistency problem is caused by occlusion. Occlusion, which is the hiding of some bumps by others, can change the distribution of visible surface normals. A solution is presented which redistributes bump normals so they match a distribution of normals similar to one derived from displacement-mapping. Lastly, there is the problem of consistency of shadowing. We have not yet found a general solution for shadowing, so we draw our images and compute our BRDF without it.

The concept of blending between methods is not new. The difficulty in overcoming the intensity distribution inconsistencies is perhaps the main reason why there are few coded examples. Kajiya[8] mentioned a hierarchy of scale which is appropriate for modelling the complexity of nature. He states that each level of detail contains the three subscales discussed above. Westin *et al.*[14] describes these levels as the geometrical, milliscale, and microscale. Perlin[11] proposed a method to shift between the BRDF and perturbed normals. Perlin's method does not include an explicit height table for determining the new normals, making displacement-mapping difficult. Fournier[7] has presented a promising approach for filtering normal maps by recording a discrete number of Phong peaks.

The software for each of the three algorithms described in this paper has been combined according to the previously discussed considerations. The result is an animation which explores a surface from changing distances and directions, showing that there are no significant side effects while transitioning between renderers. For more detail concerning the implementation refer to Becker[1].

## 2. BASIC ALGORITHMS

### 2.1 Bidirectional Reflection Distribution Functions

The BRDF is used to capture the microscopic reflectance properties of a surface. The BRDF itself can be a table of reflectivities or it can be represented by a spherical harmonic series approximation[3, 14]. It is a function of either three or four variables representing the polar and azimuthal angles of the light rays. The polar angle is called $\theta$ and it measures the angle away from the normal. Its domain is $[0, \pi/2]$. The azimuthal angle is denoted by $\phi$ and has domain $[0, 2\pi)$, with 0 and $2\pi$ both in the direction of the viewer. An isotropic surface is one for which the emitted intensity does not vary as the surface is rotated radially about its surface normal. If only isotropic textures are used, then the arguments to the BRDF reduce to the two polar viewing directions and the difference in the azimuthal angle between the viewing and lighting directions. In the most general anisotropic case, the BRDF is a function of viewing direction and lighting directions, requiring all four angles.

There are several different ways to construct a BRDF. Cabral[3] constructed the BRDF directly from a bump map using horizon tables. Westin *et al.*[14] ray traced a generalized 3-D surface sample in order to calculate the intensities for their BRDF. Our method uses normal distributions. They are already required in order to create redistribution functions for the new bump-mapping method. The same normal distributions are used to create the BRDF. Fournier[7] has also discussed normal distributions.

A normal distribution is obtained by tabulating sampled normals from a projected displacement-mapped flat patch. The range of normals is a hemisphere. The hemisphere can be discretized into a finite number of $(\theta_N, \phi_N)$ bins. When the displacement map is projected, each pixel of the projected image represents a sample normal, and the count for the bin containing that normal is incremented. If bump-mapping is used to draw the flat patch, then the approximated normal distribution is independent of $\theta$. However, when looking from some direction with $\theta > 0$, self-occlusion may occur in the displacement-mapped image. This occlusion is accounted for by rendering the displacement-mapped geometry with a hardware z-buffer, coding the normal directions into the pixel colors. For grazing angles many potentially occluding patches may have to be rendered in order to get the occlusion correct on a single patch. The problem is solved by rendering a single patch using parallel projection, and then using a block read from the screen buffer to copy the patch to all the positions where it is needed, in a back to front ordering. In a postprocess the sample normals are scanned in and the distributions are created. These distributions will be used to find

the redistribution functions and to make the BRDF. The normal distributions are stored in a 3-D table. The first index is the viewing polar angle $\theta_V$. The second and third indices are the $\theta_N, \phi_N$ angles specifying the normal direction. For simplicity a table access is described by $distr[\theta_V, N]$, where $N = (\theta_N, \phi_{N-V})$, and $\phi_{N-V}$ denotes $\phi_N - \phi_V$. The difference between viewing and lighting $\phi$'s is denoted by $\phi_{V-L}$. To improve the statistics of the distribution, the patch is viewed in many $\phi_V$ directions for each $\theta_V$. The result is normal distributions for each $\theta_V$ which account for proper occlusion. To use these distributions in constructing the BRDF, the algorithm in Figure 2 is used.

```
for each level n from highest to lowest frequency
 for each θ_V
  for each θ_L
   for each φ_{V-L}
    {H = (V + L)/|V + L|
     for each θ_N
      for each φ_{N-V}
       if highest frequency BRDF
          { increment BRDF^n_diff[θ_V, θ_L, φ_{V-L}] by
            (L · N) distr^n[θ_V, N]
            increment BRDF^n_spec[θ_V, θ_L, φ_{V-L}] by
            (H · N)^Phong distr^n[θ_V, N]
          }
       else
          { compute θ'_V, θ'_L and φ'_{V-L}
            increment BRDF^n_diff[θ_V, θ_L, φ_{V-L}] by
              BRDF^{n-1}_diff[θ'_V, θ'_L, φ'_{V-L}] distr^n[θ_V, N]
            increment BRDF^n_spec[θ_V, θ_L, φ_{V-L}] by
              BRDF^{n-1}_spec[θ'_V, θ'_L, φ'_{V-L}] distr^n[θ_V, N]
          }
    }
```

**Figure 2.** The algorithm to compute the BRDF using a table of normal distributions.

Note that there are two components to the BRDF, one for the diffuse information and one for the specular. This way the amount of diffusivity and specularity chosen can be used as a parameter later. The $\theta'_V$ and $\theta'_L$ represent the angles between the viewing or lighting direction and the bin normal $N$, rather than with the flat surface patch normal. The angle $\phi'_{V-L}$ is the difference between $L$ and $V$ when projected to the plane perpendicular to the bump normal. It is computed by

$$\phi'_L = \arctan((L \cdot (N \times x)), (L \cdot (y \times N)))$$

$$\phi'_V = \arctan((V \cdot (N \times x)), (V \cdot (y \times N)))$$

$$\phi'_{V-L} = mod((\phi'_V - \phi'_L + \pi), 2\pi) - \pi \qquad (1)$$

where $x = (1,0,0)$ and $y = (0,1,0)$ are the axis directions of the bump table. This technique will give the same BRDF as if the combined displacement maps were used, as long as there is no correlation between the bumps at the different levels.

A smooth surface patch is rendered by interpolating the BRDF trilinearly in the angles $\theta_V, \theta_L$, and $\phi_{V-L}$. The indices for the table are computed from a local coordinate on the patch surface. The smooth surface normal points in the direction of $\theta = 0$. The origin of the azimuthal angle is the projection of the viewing direction onto the surface.

For a given patch parameterization, $P(u,v)$, the partial derivatives, $P_u = \frac{\partial P}{\partial u}$ and $P_v = \frac{\partial P}{\partial v}$, are rarely the same length (causing stretching), and not always perpendicular (causing warping). For these reasons special care must be taken when indexing the BRDF to determine an intensity. The method for computing the difference in azimuthal angle is as follows:

$$V_n = [V \cdot P_u, V \cdot P_v, 0]$$

$$L_n = [L \cdot P_u, L \cdot P_v, 0]$$

$$\Phi_{V-L} = \arccos(\frac{V_n}{|V_n|} \cdot \frac{L_n}{|L_n|}) \qquad (2)$$

The stretching will actually change the normal directions making the BRDF inaccurate. The BRDF would need to be recalculated to yield a theoretically correct result, but equation (2) does get the occlusion correct and gives nice anisotropic highlight effects in places where they would be expected.

### 2.2 Bump-Mapping

In Blinn's bump-mapping[2], the surface is not actually altered from its smooth parametric form, but it is shaded as though it were.

Blinn used a bump height table B to calculate a linear approximation to the bump normal at a point P on an object surface. If $\vec{P}_u$ and $\vec{P}_v$ are the partial derivatives as above, the unnormalized surface normal is $\vec{N} = \vec{P}_u \times \vec{P}_v$. In the bump map $B$, the partial derivatives $B_u$ and $B_v$ at the interpolated point corresponding to $P$ can also be computed using finite differences.

$$B_u = (B[u + \epsilon, v] - B[u - \epsilon, v])/(2 * \epsilon) \qquad (3)$$

and $B_v$ is similar. Each evaluation of $B$ uses bilinear interpolation.

Truncating insignificant terms, Blinn[2] has showed that the new normalized normal is very close to

$$\vec{N}' = \frac{\vec{N} + B_u(\vec{N} \times \vec{P}_v) - B_v(\vec{N} \times \vec{P}_u)}{|\vec{N} + B_u(\vec{N} \times \vec{P}_v) - B_v(\vec{N} \times \vec{P}_u)|} \qquad (4)$$

We have chosen to compute the bump map derivatives by a quadratic rather than linear scheme. Mach bands are eliminated by replacing Blinn's linear formula by a $C^1$ partial derivative formula, defined by taking the derivative of the $C^2$ cubic B spline curve approximation to the bump heights as a function of $u$ or of $v$. Let $du = u - \lfloor u \rfloor$, then

$$B_u = (-du^2/2 + du - .5)B[\lfloor u \rfloor - 1, v] + (3du^2/2 - 2du)B[\lfloor u \rfloor, v]$$

$$+(-3du^2/2 + du + .5)B[\lfloor u \rfloor, v] + (du^2/2)B[\lfloor u \rfloor + 2, v]$$

and $B_v$ is similar. Here each function evaluation requires only a linear interpolation in $v$. This method uses the same eight neighboring values in the height table as does (3), but with quadratic rather than linear weights.

The normals generated by this process do not lie in a distribution consistent with the other two algorithms. As previously discussed, $\vec{N}'$ must be further modified so that on average it will contribute to a normal distribution similar to displacement-map normals. This new algorithm, redistribution bump-mapping, is described in detail in Section 3.

It should also be noted that Perlin's volume textures[12], with the improvement by Max and Becker[10], can be substituted for bump maps when computing height values. The advantage of this is that there is no explicit parameterization to be concerned with, and thus no stretching to cause singularities or anisotropy. If a square patch has an isotropic texture mapped onto it, the texture becomes anisotropic as soon as the patch is stretched unevenly. Many parameterizations have singularities which lead to degenerate patches. If anisotropy is undesirable, then volume textures should be used. Perlin also used volume textures, and redistributed the normals to make them gaussian (personal communication) in his implementation of [11].

### 2.3 Displacement-Mapping

Displacement-mapping is the direct approach to rendering surface detail. For parameterized surfaces, each patch in the object has a $u$ and $v$ parameterization. The $u$ and $v$ coordinates are used as indices to look up height values in the bump height table. The corresponding vertex is then displaced along its normal vector by that height[4]. The normal generated from the bump approximation is also used on the displaced vertices. There is little loss of accuracy in doing this, and continuity during the transition is assured. Occlusion, the main problem with bump-mapping, is accounted for automatically when the vertices are displaced.

Having multiple bump maps for many levels of detail means the displaced bumps will be rendered with the BRDF constructed from the next bump map of higher frequency. To keep combined displacements consistent with BRDFs representing several combined bump maps, surface perturbations for the $i^{th}$ level must be perpendicular to the $(i-1)^{th}$ displaced surface. This means that for each vertex, $P_u$ and $P_v$ vectors must be computed for each level of detail which has been displaced. Since $P_u$ and $P_v$ are not necessarily perpendicular it is recommended that the following formula be used to compute them, given that the surface normal is $N$.

$$P_u[level+1] = P_u[level] + B_u[level]N[level]$$

where $B_u[i]$, $B_v[i]$ are the $i^{th}$ bump map partial derivatives. The equation for $P_v$ is similar.

## 3. REDISTRIBUTION BUMP-MAPPING

### 3.1 Normal Redistribution

The problem of eliminating inconsistencies between the different rendering models lies at the heart of making smooth transitions from one algorithm to another. Primarily we are concerned with keeping the integral of intensities equal over a small area on the surface while the rendering method changes.

Unfortunately, normals from bump-mapping do not yield a distribution similar to that of displacement-mapping or the BRDF. Since the polygon or patch itself is not displaced, it is possible to see normals which ought to be hidden by occluding bumps. In order to overcome this problem a redistribution function $q$ is created. This is a function which accepts as input a normal generated by Blinn's[2] bump approximation, and outputs a normal which is statistically consistent with the distribution used to form the BRDF.

Since the distribution of normals on a displacement-mapped flat patch is different for each viewing angle, it is necessary to have redistribution functions for each one. When the viewing angle is vertical, the identity function is used. When the viewing angle is just above the horizon, the redistribution of bump normals is necessarily quite drastic. The effect is to pull forward normals that might be facing away, and push upward those that might be hidden. This new scheme for doing bump-mapping might appropriately be termed redistribution bump-mapping.

### 3.2 Redistribution Function Construction

Suppose a bumpy surface is viewed from a direction with polar angle $\theta_V$. Let $g$ denote the distribution of normals $distr(\theta, N)$ at this fixed $\theta_V$, computed as above from the displacement map. Let $f$ denote the distribution of normals in a (non-displaced) bump-mapped image. Note that f is the same as $distr(0, N)$. If $q$ is the redistribution function described above, then the requirement that $q$ take the distribution $f$ to the distribution $g$ is that for any region $R$ in the hemisphere $H$ of possible normals,

$$\int_{q(R)} f(\theta,\phi)d\omega = \int_R g(\theta,\phi)d\omega \tag{5}$$

It is easier to explain how to specify $q$ in a 1-D case. So suppose $f(x)$ and $g(x)$ are two distributions on $[0,1]$, such that

$$\int_0^1 f(x)dx = \int_0^1 g(x)dx = 1 \tag{6}$$

The problem is to find $q : [0,1] \rightarrow [0,1]$ such that

$$\int_{q(a)}^{q(b)} f(x)dx = \int_a^b g(x)dx \tag{7}$$

where $a$ and $b \in [0,1]$. It is enough to guarantee that

$$\int_0^{q(b)} f(x)dx = \int_0^b g(x)dx. \tag{8}$$

Let

$$G(b) = \int_0^b g(x)dx$$

and

$$F(b) = \int_0^b f(x)dx.$$

Then

$$G(b) = F(q(b))$$

hence

$$q(b) = F^{-1}(G(b)). \tag{9}$$

The redistribution function $q$ maps a point $b$ so that the area under the curve before $b$ in $g$ is equal to the area under the curve before the point $q(b)$ in $f$.

The problem in 2-D can be handled similarly. One method is to define 1-D redistribution functions separately for $\theta$ and $\phi$. This gives adequate results for most bump maps, whose $\theta$ and $\phi$ distributions are fairly independent. This independence assumption is confirmed by the animation. For a more precise redistribution function, one can first redistribute $\phi$, and then for each fixed $\phi$, establish a separate redistribution function for $\theta$. For details see Becker[1].

## 4. TRANSITIONS

### 4.1 Partial Bump Displacement

For control of appearance and for smooth transitions we want the ability to change the height of the bumps in the bump map. This will alter the normal distribution and occlusion information. By close consideration we can see that the change can be accounted for without having to recalculate the redistribution functions every time the bump heights are altered. If the heights are multiplied by a factor $t$, then the tangent of the angle between the bump normal and the smooth surface normal should also change by a factor $t$; i.e., $\tan(\theta_{N_t}) = t \cdot \tan(\theta_N)$. The normal, $N = (\theta_N, \phi_N)$, needs to be replaced by $N_t = (\arctan(t \cdot \tan(\theta_N)), \phi)$. In order to keep the visibility information the same, the viewing angle, $\theta_V$, must be replaced with $\theta_W = \operatorname{arccot}(\cot(\theta_V)/t)$. See discussion below concerning Figure 3.

The height of the bumps used to calculate the BRDF and redistribution functions must be the same as that of the bumps being rendered. This is because the BRDF is changed in a non-trivial way as the bump heights change. If we were only concerned with bump- and displacement-mapping, we could change the indexing on the redistribution functions to get the occlusion correct for changing bump heights. Unfortunately there is no easy way to re-index the BRDF to account for scale changes. Between the BRDF and redistribution bump-mapping, an intensity is computed for both methods. The resulting intensity is an interpolation of the two.

For the transition between bump- and displacement-mapping, intensity interpolation is not used, since it would cause the bump shading (particularly the highlights) to cross-dissolve rather than correctly adjust in position. As the bumps go from no displacement to full displacement the surface normals do not change, since they are always represented by Blinn's bump normal. The visible subset of bump normals does change, however, due to changing occlusion. Let *disp* be the transition parameter which gives the fraction of the full bump height. With $disp = 0$ all normals are seen, even those on the back of bumps. With $disp = 1$, only the visible subset of these normals are seen. In Figure 3 the segments of the visible surface are shown in bold. The redistribution of normals takes normals from standard bump-mapping into this visible subset. For partially displaced bumps there is a different subset of visible normals, but there is a relationship between the bump height and this subset which can be exploited to give the necessary redistribution.

Different redistribution functions for varying heights are not stored, only different functions for different viewing $\theta$'s. Fortunately the two are equivalent. For the fractional bump height, *disp*, we can determine a new $\theta_W$ for which the same distribution of full height bump normals will be seen. Figure 3 shows that the distribution of normals for this partially displaced surface, viewed from $\theta_V$, is identical to the distribution of visible normals for the fully displaced surface viewed from $\theta_W$. The slope of the line $V$ in Figure 3 is *disp* times the slope of line $W$, so $\cot(\theta_V) = disp \cdot \cot(\theta_W)$ and the formula for finding $\theta_W$ is:

$$\theta_W = \operatorname{arccot}(\cot(\theta_V)/disp).$$

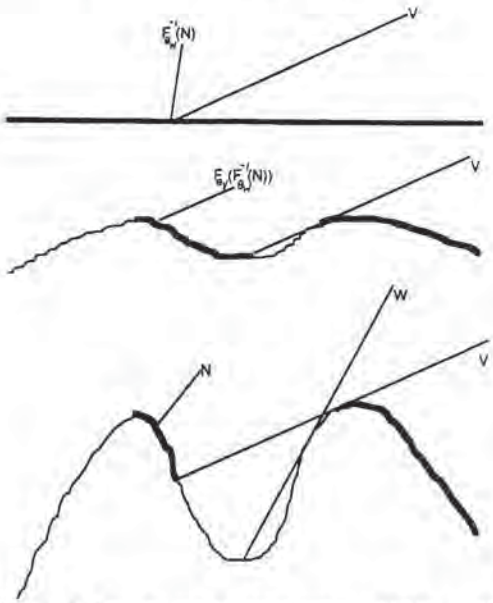



**Figure 3** Top: the non-displaced surface. Middle: surface displaced by bump height fraction *disp*. Bottom: Fully displaced surface.

The inverse redistribution function for $\theta_W$ is applied to take the visible bump normal from the partially displaced surface into a distribution similar to one from a flat bump-mapped surface. Next the redistribution function for $\theta_V$ is applied to that normal to take it all the way forward to match statistically a full displacement-mapped normal. Thus the change from bump-mapping to displacement-mapping is done through two table based function evaluations. Notice that as the bumps decrease in height, the new viewing $\theta_W$ approaches vertical. This means that the inverse function needs to alter the normals less in order to get them back to the bump-map distribution.

### 4.2 Algorithm Selection Criterion

Now that it is known how to modify the algorithms so that they will not deviate from a fundamental reflection model, it must be decided when to apply which algorithm. Clearly displacement-mapping should be applied when the view is close, and the BRDF when the view is far. The relationship is $1/d$, where $d$ is distance, since that is how the projected size of an object relates to distance. Another variable to consider is viewing angle, $\theta_V$. If $f$ is the wavelength of a feature then $f\cos(\theta_V)/d$ is the wavelength of the projected feature (in the direction of maximum foreshortening), and should be no smaller than two pixels. When the object is close, we would like to see a rough silhouette; when it is far, aliasing becomes a problem on the edge so use of the BRDF is desirable. This implies that as the object moves away from the viewer, the transition from displaced bumps to BRDF will be far more rapid on the object silhouette than on that area where the patch normal points toward the viewer. The

threshold at which the switch occurs is determined by a constant $D$. Summarizing these properties, we define a transition parameter

$$T(d, \theta_V) = (1/d - D)/(\cos(\theta_V) + \epsilon). \qquad (10)$$

Here $d$ is the distance from the viewpoint to the surface, $\theta_V$ is the angle between the viewing ray and the surface normal, and $D$ is dependent on individual bump maps. To avoid an instantaneous transition on the silhouette an $\epsilon$ is added to the cosine term in the denominator. The constant $D$ should be large if the highest frequency component of the bump map is large. Note that $D$ controls where the function changes from positive to negative, and thus lies midway between displacement-mapping and the BRDF. The formula for determining $D$ is

$$D = c \cdot freq \cdot S$$

where $freq$ is the highest frequency in the bump map and $S$ is the amount the $u$ and $v$ values are scaled. If $S$ is large, then the bump map will be repeated more times over the same area, and the partial derivatives, $P_u$ and $P_v$, are made shorter by a factor of $S$. The constant $c$ controls computational effort by globally shifting the scene toward more BRDF or alternatively more displacement. If shadows are included, the shadow terminator should be treated just like the silhouette. Areas far from the terminator are likely to be completely illuminated or shadowed, but on the terminator, displacement-mapping will make the shadowing exact. The parameter given by equation (10) determines the algorithm or algorithms used for rendering. Let the threshold values for choice of renderer be $e1 < e2 < 0 < e3 < e4$. If $T < e1$ then use the BRDF, if $T > e4$ then use displacement mapping, and if $e2 < T < e3$ use redistribution bump mapping. Values of $T$ other than these indicate regions where algorithms are blended. Values of -1, -.3, .3, and 1 respectively, were found to give good results.

### 4.3 Multiple Levels of Detail

With multiple levels of detail there are many more than two possible transition points. Many other cases need to be considered. The displacement-mapped image of the $i^{th}$ layer is rendered using the BRDF for the $(i-1)^{th}$ layer. As the camera continues to zoom in, the BRDF will switch to bump-mapping and then again to displacement-mapping.

Since each bump map has its own independent transition regions, some areas may have bump-mapping from two or more different levels. Perlin [11] suggests that each set of bumps be limited to a narrow range of frequencies. The result of implementing two levels of detail is shown in Figure 4. The bump map describing the surface detail is broken up into high and low order band-limited frequencies. The low frequencies compose the first level bump map and the high frequencies compose the second level. The left half of Figure 4 is color coded according to the algorithm used to render the most refined level of detail visible. Hence one can see bumpy sections colored yellow to indicate the BRDF from the next lower level was used to render the displaced bumps.

## 5. RESULTS

### 5.1 Consistency Comparison

In Figure 5 we can see the four rendering methods compared. The difference between the lighting and viewing $\phi$ is zero. Note that since the lighting and viewing directions are in alignment the patch becomes brighter for grazing angles. The rows are rendered with bump-mapping, redistribution bump-mapping, BRDF, and displacement mapping respectively. Note that redistribution bump-mapping is far more consistent with the BRDF and displacement-mapping than is ordinary bump-mapping. Figure 6 is a table which shows quantitative results for viewing angles corresponding to those shown in Figure 5.

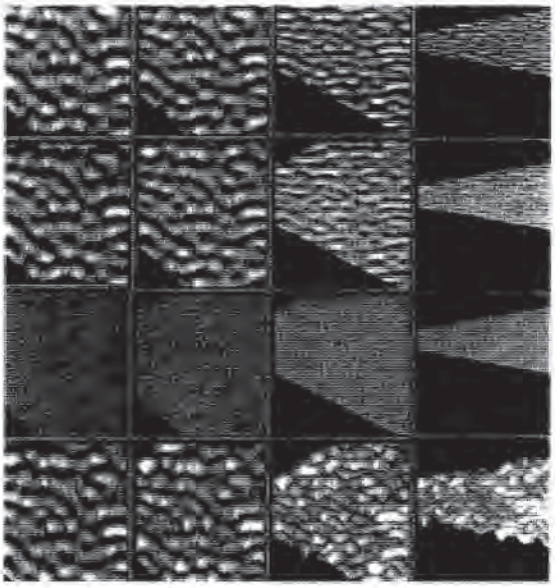


**Figure 5** Intensity comparisons. The lighting direction is consistently $\theta_L = \pi/4$. The rows from top to bottom represent bump-mapping, redistribution bump mapping, BRDF, and displacement mapping.

| | $\theta_v$=0 | $\theta_v$=π/6 | $\theta_v$=π/3 | $\theta_v$=4π/9 |
|---|---|---|---|---|
| Bump | 128 | 129 | 129 | 129 |
| Redistribution | 128 | 143 | 170 | 194 |
| BRDF | 129 | 146 | 172 | 192 |
| Displacement | 128 | 146 | 175 | 194 |

**Figure 6** Area averaged intensities for the diffuse component.

In Figure 7, a single flat patch is drawn in perspective. Regions in the foreground are clearly displacement-mapped. The middle region is redistribution bump-mapped, and the furthest edge is almost completely shaded with the BRDF. It should be apparent that there is no intensity inconsistency between methods and that the transition is smooth.

### 5.2 Conclusions

Combining displacement-mapping, bump-mapping and a BRDF into one algorithm makes it possible to explore great

scale changes, without changing the geometrical data base. Using a series of bump maps we can generate a variety of rough surfaces simulating different material properties. Objects in the scene will have a complex underlying structure but only the minimum amount of effort necessary to give the impression of complete geometrical representation will be expended. Current animations are restricted by the amount of geometrically represented detail. If the view gets too close to a feature, large drab polygons fill the display. With hierarchy of detail, the polygon level need never be reached, no matter how close the viewer gets. Even at intermediate and far distances the light interacts with flat polygonal surfaces as if they were truly composed of millions of smaller micro-polygons. As a result the otherwise drab polygons become alive with texture and interesting highlights. Those smaller micro-polygons may actually get rendered, but only if the viewer zooms in much closer.

### 5.3 Future Research

Shadowing is the main enhancement yet to be considered. One way to do the shadowing of displaced bumps is to use the two-pass z-buffer method developed by Williams[15]. Horizon mapping[9] has been shown to generate shadows for bump-mapped images. It will also work for redistribution bump-mapping since the horizon is determined by the $u$ and $v$ parameterization, not the normal. However, this may cause a problem since the rendering is according to a redistributed normal, and the shadows are according to the parameterization. The shadowing may look inappropriate for the rendered bumps. The shadowing for BRDFs can be done using horizon mapping, as was demonstrated by Cabral[3]. Another possibility is to use only the unshadowed normals from a displaced, rendered, and shadowed flat patch to generate the distributions for the BRDF and the redistribution function. The result should be consistent in terms of average intensity, but may not look qualitatively correct.

### 5.4 Acknowledgements

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.

## BIBLIOGRAPHY

1. Becker, Barry, "Smooth Transitions Between Rendering Algorithms During Animation", Master's thesis, University of California at Davis, Davis, CA, December, 1992.

2. Blinn, James F., "Models of Light Reflection for Computer Synthesized Pictures", *Proceedings of SIGGRAPH '77, Computer Graphics*, Vol. 11, No. 2, July, 1977, pp192-198.

3. Cabral, Brian, Nelson Max, and Rebecca Springmeyer, "Bidirectional Reflection Functions from Surface Bump Maps", *Proceedings of SIGGRAPH '87, Computer Graphics*, Vol. 21, No. 4, July, 1987, pp273-281.

4. Cook, Robert L., "Shade Trees", *Proceedings of SIGGRAPH '84, Computer Graphics*, Vol. 18, No. 3, July, 1984, pp223-231.

5. Cook, Robert L., Loren Carpenter, and Edwin Catmull, "The Reyes Image Rendering Architecture", *Proceedings of SIGGRAPH '87, Computer Graphics*, Vol. 21, No. 4, July, 1987, pp95-102.

6. Cook, Robert L, and Kenneth Torrance,"A Reflectance Model for Computer Graphics",*Proceedings of SIGGRAPH '81, Computer Graphics*, Vol. 15, No. 3, August, 1981, pp307-316.

7. Fournier, Alain, "Normal Distribution Functions and Multiple Surfaces",*GI '92 Workshop on Local Illumination*, 1992, pp 45-52.

8. Kajiya, James, "Anisotropic Reflection Models", *Proceedings of SIGGRAPH '85, Computer Graphics*, Vol. 19, No. 3, July, 1985, pp15-21.

9. Max, Nelson L., "Horizon Mapping: Shadows for Bump-mapped Surfaces", *The Visual Computer*, Springer-Verlag, Vol. 4, No. 2, 1988, pp109-117.

10. Max, Nelson L., and Barry Becker, "Bump Shading for Volume Textures", to appear in *IEEE Computer Graphics and Applications*, 1993.

11. Perlin, Kenneth, "A Unified Textural Reflectance Model", *Advanced Image Synthesis course notes, Proceedings of SIGGRAPH '84, Computer Graphics*, July, 1984.

12. Perlin, Kenneth, "An Image Synthesizer", *Proceedings of SIGGRAPH '85, Computer Graphics*, Vol. 19, No. 3, July, 1985, pp287-296.

13. Torrance, Kenneth, and Ephraim Sparrow, "Theory for Off-Specular Reflection from Roughened Surfaces", *Journal of the Optical Society of America*, 57(9), 1967, pp1105-1114.

14. Westin, Stephen H., James R. Arvo, and Kenneth E. Torrance, "Predicting Reflectance Functions from Complex Surfaces", *Proceedings of SIGGRAPH 92, Computer Graphics*, Vol. 26, No. 2, July, 1992.

15. Williams, Lance, "Casting Curved Shadows on Curved Surfaces",*Proceedings of SIGGRAPH '78, Computer Graphics*, Vol. 12, No. 3, July, 1978, pp270-274.

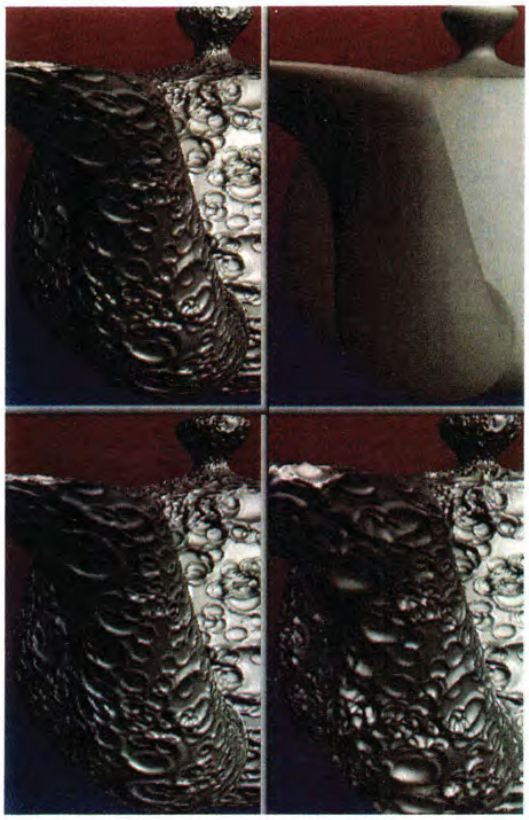


Figure 1. Counter-clockwise from upper left: bump-mapping, redistribution bump-mapping, displacement-mapping, BRDF. The difference between redistribution bump-mapping and plain bump-mapping is apparent near the bottom of the spout.

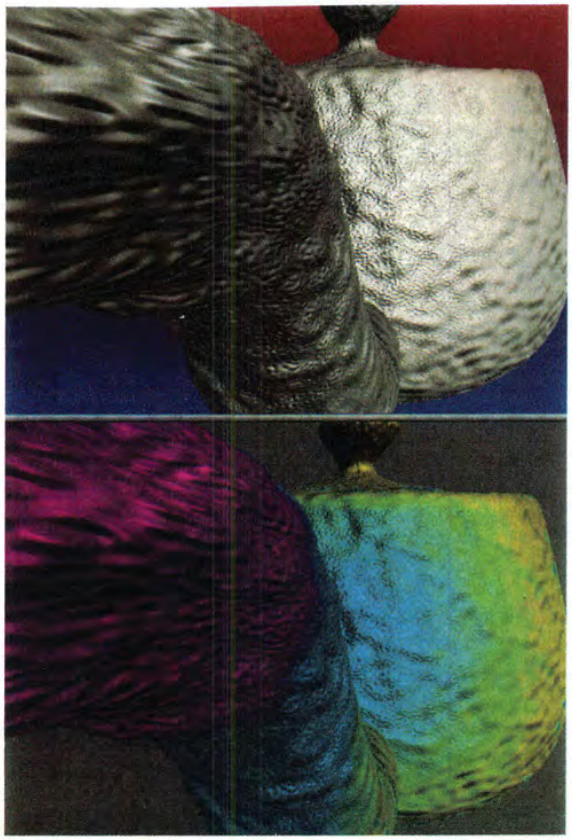


Figure 4. Two levels of bumpy detail. Colors in the bottom half indicate BRDF(yellow), redistribution bump-mapping(blue), and displacement-mapping(red) for the higher frequency bumps.

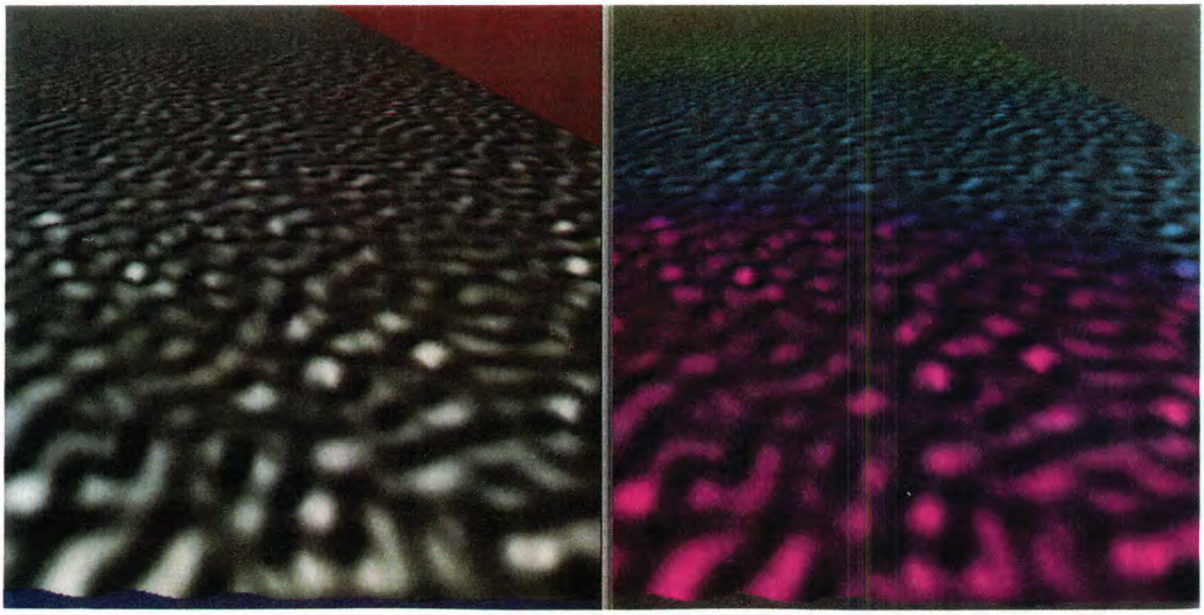


Figure 7. Transitions on a flat surface. BRDF(yellow) in the back, redistribution bump-mapping(blue) in the middle, and displacement-mapping(red) in the foreground.



# Linear Color Representations for Full Spectral Rendering

Mark S. Peercy
Department of Applied Physics
Stanford University

## Abstract

We present a general linear transform method for handling full spectral information in computer graphics rendering. In this framework, any spectral power distribution in a scene is described with respect to a set of fixed orthonormal basis functions. The lighting computations follow simply from this decision, and they can be viewed as a generalization of point sampling. Because any basis functions can be chosen, they can be tailored to the scenes that are to be rendered. We discuss efficient point sampling for scenes with smoothly varying spectra, and we present the use of characteristic vector analysis to select sets of basis functions that deal efficiently with irregular spectral power distributions. As an example of this latter method, we render a scene illuminated with fluorescent light.

**CR Categories and Subject Descriptors:** I.3.3 [**Computer Graphics**]: Picture/Image Generation–Display Algorithms; I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism.

**Additional Keywords:** linear color representations, full spectral rendering, linear models, tristimulus values.

## 1 Introduction

Accurate color rendering in computer graphics must account for the full spectral character of the lights and surfaces within a scene. The rendering procedure must preserve enough spectral information to compute final values for output to some display device, such as an *RGB* monitor. However, one wishes to minimize the computational cost of the rendering to reduce the time required to create an image. Therefore, one desires efficient methods of handling full spectral information during image synthesis.

Author's address: Dept. of Applied Physics, Stanford University
Stanford, CA 94305-4090
peercy@kaos.stanford.edu (415)725-3301



Some suggested techniques in dealing with full spectral information include the use of the tristimulus values for the lights and surfaces [1], the use of polynomial representations of spectra [16], and the use of linear models of surfaces and lights [20] [12]. The typical method employed is point sampling of the surfaces and the lights at a given number of wavelengths. These point samples are used in a numerical integration method to compute approximate tristimulus values before being transformed to values appropriate for display. To minimize the total number of samples, one seeks an efficient integration approximation; one approximation that has been studied in various forms is Gaussian quadrature [14] [19] [2].

In this paper, we consider a more general method for handling full spectral information in synthetic image generation; our technique is closely related to the use of linear models presented in [20]. The principal idea is that we describe the spectral power distribution of the light at every step of the rendering procedure with respect to a single collection of orthonormal basis functions. This formalism encompasses point sampling, which uses delta functions as its basis functions.

The constraint of describing all of the spectral power distributions with respect to the basis functions is advantageous for two reasons. First, it makes the rendering process completely linear. Therefore, this technique can be considered a generalization of point sampling and can be readily incorporated into standard renderers. Second, one has the freedom to select any orthonormal set of basis functions. This freedom can be exploited to increase the efficiency of the rendering process.

The body of this paper is divided into two main sections. In Section 2 we discuss the mathematical formalism of linear color representations of the lights and surfaces, and in Section 3 we address the problem of selecting appropriate basis functions. In this latter section, we discuss Riemann summation for efficient point sampling in scenes with smoothly varying spectra, and we present the use of characteristic vector analysis to provide efficient basis functions for scenes with complex spectra.

 ACM-0-89791-601-8/93/008/0191 $01.50

## 2 Linear Color Representations

During the rendering process, we demand that any spectral power distribution in the scene be described by $m$ orthonormal basis functions $E_i(\lambda); i = 1,...,m$. By any distribution, we mean not only the light coming directly from a light source but also light that has been once, twice, or an arbitrary number of times reflected from surfaces in the scene. In this section, we use this restriction to derive the color representations of both the spectral power distributions and the surfaces, and we discuss the transformation of this color information to values appropriate for display.

### 2.1 Spectral Power Distributions

To obtain a representation for the spectral power distributions in a scene, we can project the spectral power distribution, $I(\lambda)$, of any light source onto the subspace spanned by the basis functions;

$$I(\lambda) = \sum_{i=1}^{m} \epsilon_i E_i(\lambda), \tag{1}$$

where

$$\epsilon_i = \int_\lambda I(\lambda) E_i(\lambda) d\lambda. \tag{2}$$

follows from the orthonormality condition. Thus, any light within the scene can be described with the $m$ elements $\epsilon_i$. These elements are simply the coefficients of the linear transformation defined by the set of basis functions, so we refer to this method as a general linear transform method.

### 2.2 Surface Reflectances

To obtain a representation for the surfaces, we project the spectral power distribution of the light reflected from those surfaces onto the set of basis functions (for clarity and without loss of generality, we neglect transmission and attenuation in this discussion). Lighting models typically divide the reflected light into three terms: ambient, diffuse, and specular ([6] discusses lighting models in detail); the spectral power distribution of light reflected from a surface, $I_o$, is given by

$$\begin{aligned} I_o(\Omega,\lambda) &= R_a(\lambda) I_a(\lambda) + G_d(\Omega) R_d(\lambda) I_s(\lambda) \\ &\quad + R_s(\Omega,\lambda) I_s(\lambda). \end{aligned} \tag{3}$$

Here, $\Omega$ denotes a general dependence on the geometry of the reflection, and $\lambda$ denotes a general dependence on wavelength. $I_a(\lambda)$ is the distribution of the ambient light, $I_s(\lambda)$ is the distribution of directional incoming light, and $G_d(\Omega)$ is the diffuse geometry term. $R_a(\lambda)$, $R_d(\lambda)$, and $R_s(\Omega,\lambda)$ are the ambient, diffuse, and specular reflectances of the surface, respectively. In general, the specular reflectance is a function both of geometry and wavelength. However, empirical models often replace the specular reflectance with a separable term, resulting in a *piecewise separable* lighting model

$$\begin{aligned} I_o(\Omega,\lambda) &= R_a(\lambda) I_a(\lambda) + G_d(\Omega) R_d(\lambda) I_s(\lambda) \\ &\quad + G_s(\Omega) R_s(\lambda) I_s(\lambda). \end{aligned} \tag{4}$$

As described in Section 2.1, the ambient light and directional light are represented by their transform coefficients,

$$I_a(\lambda) = \sum_{i=1}^{m} \epsilon_i^a E_i(\lambda) \tag{5}$$

$$I_s(\lambda) = \sum_{i=1}^{m} \epsilon_i^s E_i(\lambda). \tag{6}$$

By using Equations 5 and 6 in Equation 3, the spectral power distribution reflected from a surface is

$$\begin{aligned} I_o(\Omega,\lambda) &= \sum_{i=1}^{m} \epsilon_i^a R_a(\lambda) E_i(\lambda) + \sum_{i=1}^{m} \epsilon_i^s G_d(\Omega) R_d(\lambda) E_i(\lambda) \\ &\quad + \sum_{i=1}^{m} \epsilon_i^s R_s(\Omega,\lambda) E_i(\lambda). \end{aligned} \tag{7}$$

To obtain the surface representations, we project this result back onto the the basis functions as in Equation 1;

$$I_o(\Omega,\lambda) = \sum_{j=1}^{m} \epsilon_j^o E_j(\lambda). \tag{8}$$

From Equation 2 and Equation 7,

$$\epsilon_j^o = \int_\lambda I_o(\Omega,\lambda) E_j(\lambda) d\lambda \tag{9}$$

$$= \sum_{i=1}^{m} R_{ij}^a \epsilon_i^a + G_d(\Omega) \sum_{i=1}^{m} R_{ij}^d \epsilon_i^s + \sum_{i=1}^{m} R_{ij}^s(\Omega) \epsilon_i^s \tag{10}$$

where

$$R_{ij}^a = \int_\lambda R_a(\lambda) E_i(\lambda) E_j(\lambda) d\lambda \tag{11}$$

$$R_{ij}^d = \int_\lambda R_d(\lambda) E_i(\lambda) E_j(\lambda) d\lambda \tag{12}$$

$$R_{ij}^s(\Omega) = \int_\lambda R_s(\Omega,\lambda) E_i(\lambda) E_j(\lambda) d\lambda. \tag{13}$$

$R_{ij}^a$ is the projection onto the $j^{th}$ basis function of the spectral power distribution obtained from the reflection of the $i^{th}$ basis function from the ambient reflectance of the surface. $R_{ij}^d$ and $R_{ij}^s(\Omega)$ are analogous terms for the diffuse and specular reflections, respectively.

Writing Equation 10 in matrix form, we obtain

$$\begin{aligned} \begin{pmatrix} \epsilon_1^o \\ \vdots \\ \epsilon_m^o \end{pmatrix} &= \begin{pmatrix} R_{11}^a & \cdots \\ \vdots & \ddots \end{pmatrix} \begin{pmatrix} \epsilon_1^a \\ \vdots \\ \epsilon_m^a \end{pmatrix} + \\ & G_d(\Omega) \begin{pmatrix} R_{11}^d & \cdots \\ \vdots & \ddots \end{pmatrix} \begin{pmatrix} \epsilon_1^s \\ \vdots \\ \epsilon_m^s \end{pmatrix} + \\ & \begin{pmatrix} R_{11}^s(\Omega) & \cdots \\ \vdots & \ddots \end{pmatrix} \begin{pmatrix} \epsilon_1^s \\ \vdots \\ \epsilon_m^s \end{pmatrix} \end{aligned} \tag{14}$$

In vector notation, this equation can be written

$$\vec{\epsilon^o} = R^a \vec{\epsilon^a} + G_d(\Omega) R^d \vec{\epsilon^s} + R^s(\Omega) \vec{\epsilon^s}. \tag{15}$$

This final equation reveals the mathematical formalism behind the linear transform method. The spectral power distributions ($I_a(\lambda)$, $I_s(\lambda)$, and $I_o(\Omega,\lambda)$) are represented by

column vectors of length $m$ containing the transform coefficients ($\vec{\epsilon^a}, \vec{\epsilon^s}$, and $\vec{\epsilon^o}$, respectively). Each component of the surface reflectance ($R_a(\lambda), R_d(\lambda)$, and $R_s(\Omega, \lambda)$) is represented by a single $m \times m$ matrix ($R^a, R^d$, and $R^s(\Omega)$, respectively). The interaction of light with a surface component assumes the form of simple matrix multiplication, converting the coefficients of the incoming light into the coefficients of the outgoing light. This result is a generalization of the point sampling case; with point samples, the surface matrices are diagonal, and the matrix product multiplies respective sample values. Because this technique is linear, it can be included without difficulty in standard renderers.

For the general lighting model case, the specular matrix is a function of the geometry. Because the elements of the surface matrices are obtained through integration over the basis functions, this integration must be performed for each geometry configuration. If, however, one uses a piecewise separable lighting model, the geometry and wavelength dependence separate in the specular term,

$$\vec{\epsilon^o} = R^a \vec{\epsilon^a} + G_d(\Omega) R^d \vec{\epsilon^s} + G_s(\Omega) R^s \vec{\epsilon^s}, \tag{16}$$

and the three surface matrices, $R^a, R^d$, and $R^s$, can be precomputed.

The above discussion addresses only surface reflection, but effects such as transmission and attenuation can be included straightforwardly in this framework. As with the reflectance components, these terms take the form of $m \times m$ matrices that act on the coefficients of the incoming light.

### 2.3 Conversion to *RGB*

The rendering algorithm determines the spectral contributions to a pixel by computing multiple reflection paths from each of the light sources to the viewer. These contributions are transform coefficients, and by linearity they can be combined to provide a final set of coefficients for that pixel, $\epsilon_i^p; i = 1, ..., m$. Equation 1 gives the approximation to the spectral power distribution arriving at the pixel,

$$I_p(\lambda) = \sum_{i=1}^{m} \epsilon_i^p E_i(\lambda). \tag{17}$$

To compute appropriate values for display, one first computes the tristimulus values, $XYZ$, for the pixel by integrating the final spectrum over the three color matching functions [21]

$$\begin{aligned} X = \int \bar{x}(\lambda) I_p(\lambda) d\lambda &= \int \sum_{i=1}^{m} \epsilon_i^p \bar{x}(\lambda) E_i(\lambda) d\lambda \\ &= \sum_{i=1}^{m} T_{xi} \epsilon_i^p \end{aligned} \tag{18}$$

$$\begin{aligned} Y = \int \bar{y}(\lambda) I_p(\lambda) d\lambda &= \int \sum_{i=1}^{m} \epsilon_i^p \bar{y}(\lambda) E_i(\lambda) d\lambda \\ &= \sum_{i=1}^{m} T_{yi} \epsilon_i^p \end{aligned} \tag{19}$$

$$\begin{aligned} Z = \int \bar{z}(\lambda) I_p(\lambda) d\lambda &= \int \sum_{i=1}^{m} \epsilon_i^p \bar{z}(\lambda) E_i(\lambda) d\lambda \\ &= \sum_{i=1}^{m} T_{zi} \epsilon_i^p. \end{aligned} \tag{20}$$

In matrix form, this set of equations can be written

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} T_{x1} & T_{x2} & \cdots & T_{xm} \\ T_{y1} & T_{y2} & \cdots & T_{ym} \\ T_{z1} & T_{z2} & \cdots & T_{zm} \end{pmatrix} \begin{pmatrix} \epsilon_1^p \\ \epsilon_2^p \\ \vdots \\ \epsilon_m^p \end{pmatrix}. \tag{21}$$

With $\vec{x} = (X, Y, Z)^T$, this equation yields

$$\vec{x} = T \vec{\epsilon^p}. \tag{22}$$

The elements $T_{xi}, T_{yi}$, and $T_{zi}$ of the matrix $T$ are coefficients that result from integration of the $i^{th}$ basis function over the three color matching functions. For point sampling, these elements are modified based on the method of numerical integration. For example, common Riemann summation over evenly spaced samples includes the distance between the sample points [17], and Gaussian quadrature has its own unique weights [5].

Assuming that an *RGB* display monitor is properly gamma-corrected [4], the color values, $\vec{c} = (R, G, B)^T$, of a given pixel are computed from the tristimulus values by applying a $3 \times 3$ matrix, $M$, derived from the chromaticities of the phosphors of the monitor [6]

$$\begin{aligned} \vec{c} &= M \vec{x} && (23) \\ &= MT \vec{\epsilon^p} && (24) \\ &= C \vec{\epsilon^p}. && (25) \end{aligned}$$

Therefore, the *RGB* values can be obtained directly through a linear transformation of the final coefficient values by a $3 \times m$ matrix $C$. Because this step is linear, it can be applied at any time to the separate contributions to the final pixel values.

## 3 Selection of Basis Functions

It is in the selection of the basis functions that the flexibility of the general transform method is demonstrated. In this section, we describe some factors that determine the effective selection of basis functions, and we present two methods for determining basis functions that are tailored to the spectral power distributions in a scene.

As mentioned in Section 2, the lighting model is a significant influence on the choice of basis functions. If the lighting model is not piecewise separable, the surface matrices must be computed for each geometry configuration, so the most efficient basis functions are most likely point samples. If, however, the lighting model is piecewise separable, we have another consideration. The components of the surface reflectances are represented by $m \times m$ matrices. Therefore, the reflection of light from a surface requires, in general, $m^2$ multiplies. If the basis functions are point samples, though, the surface matrices are diagonal, and the reflection requires only $m$ multiplies. Indeed, only $m$ multiplies are required for any set of non-overlapping basis functions. Consequently, the computational intensiveness of the general transform rises more rapidly than that of point sampling as the number of basis functions increases.



193

A third consideration when selecting basis functions is the nature of the spectral power distributions in the scene to be rendered. For smoothly varying distributions, point sampling can be quite efficient, but for complicated spectra, a set of general basis functions can be more appropriate. We discuss each of these methods in the following sections.

### 3.1 Point Sampling

Point sampling is typically linked to a numerical integration method used in approximating the tristimulus integrals, Equations 18-20. Gaussian quadrature, which is optimal for integrating polynomials over general weighting functions [5], has been applied to this problem [14] [19] [2]. If the spectral power distributions are well described by lower order polynomials, Gaussian quadrature can provide sufficient accuracy with a small number of sample points; it was shown in [14] that as few as four point samples are adequate for many rendering applications.

Here, we discuss the use of simple Riemann summation for approximating the tristimulus integrals. Rather than being efficient for polynomial functions, Riemann summation is efficient when integrating functions that contain a small number of Fourier coefficients.

#### Riemann Summation

Riemann summation is the sum over evenly spaced sample values weighted by the distance between the sample wavelengths [17]. Given $N+2$ evenly spaced sample points $\lambda_0, \lambda_1, ..., \lambda_{N+1}$ separated by a distance $\Delta\lambda = (\lambda_{N+1} - \lambda_0)/(N+1)$ and a spectral power distribution $I(\lambda)$, Riemann summation gives

$$
\begin{aligned}
X &= \int_\lambda \bar{x}(\lambda)I(\lambda)d\lambda \approx \Delta\lambda \sum_{i=0}^{N+1} \bar{x}(\lambda_i)I(\lambda_i) \\
Y &= \int_\lambda \bar{y}(\lambda)I(\lambda)d\lambda \approx \Delta\lambda \sum_{i=0}^{N+1} \bar{y}(\lambda_i)I(\lambda_i) \qquad (26) \\
Z &= \int_\lambda \bar{z}(\lambda)I(\lambda)d\lambda \approx \Delta\lambda \sum_{i=0}^{N+1} \bar{z}(\lambda_i)I(\lambda_i).
\end{aligned}
$$

An appropriate choice of endpoints, $\lambda_0$ and $\lambda_{N+1}$, is the most closely spaced pair of wavelengths that can be chosen such that the color matching functions at these wavelengths can be taken to be zero. We found that $\lambda_0 = 400nm$ and $\lambda_{N+1} = 700nm$ are often reasonable choices; truncation at these limits results in errors significantly smaller than those incurred by undersampling the spectra [17] [18]. Taking $\bar{x}(\lambda_0) = \bar{y}(\lambda_0) = \bar{z}(\lambda_0) = 0$ and $\bar{x}(\lambda_{N+1}) = \bar{y}(\lambda_{N+1}) = \bar{z}(\lambda_{N+1}) = 0$, only the $N$ interior points, $\lambda_1, ..., \lambda_N$, need to be preserved during the rendering process; the basis functions for the spectral power distributions are given by delta functions at these wavelengths.

With the endpoints of the integrands equal to zero, Riemann summation with $N$ points is exact for any linear combination of the first $2N+2$ Fourier functions $1$, $sin(2\pi\frac{\lambda-\lambda_0}{\lambda_{N+1}-\lambda_0})$, $cos(2\pi\frac{\lambda-\lambda_0}{\lambda_{N+1}-\lambda_0})$, ..., $sin(2\pi N\frac{\lambda-\lambda_0}{\lambda_{N+1}-\lambda_0})$, $cos(2\pi N\frac{\lambda-\lambda_0}{\lambda_{N+1}-\lambda_0})$, $sin(2\pi(N+1)\frac{\lambda-\lambda_0}{\lambda_{N+1}-\lambda_0})$. Therefore, if the products of the spectral power distributions with each of the color matching functions are well described by a small number of Fourier coefficients, Riemann summation provides an efficient method for integration. For the set of spectral power distributions obtained from the Macbeth Color Checker [13] under CIE Standard Illuminant C [21], Riemann summation with four point samples at 460nm, 520nm, 580nm, and 640nm results in an average error of less than 5% in the tristimulus values. Rendering with these four sampling points is often sufficient; if it is not, selecting five, six, or more evenly spaced samples is straightforward.

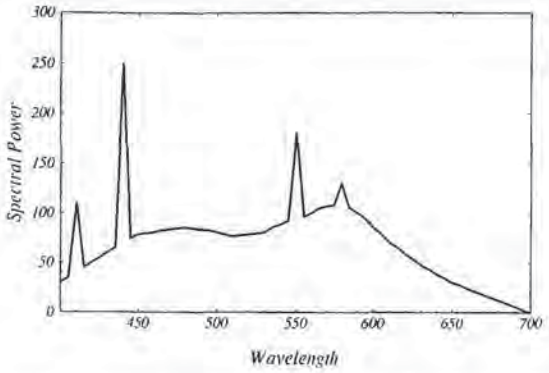



Figure 1: Spectral power distribution of a fluorescent light

### 3.2 General Basis Functions

For scenes with complicated spectral power distributions or surface properties, naive point sampling is insufficient. One notable example is the spectral power distribution of fluorescent light, which is ubiquitous in indoor scenes. Fluorescent light, an example of which is shown in Figure 1 [21], is characterized by narrow emission lines at several wavelengths, a factor leading to aliasing with a small number of point samples. For these complicated cases, one would like to be able to tailor the basis functions to the complex spectra. One attempt in this direction is the use of abutting box functions over the range of wavelengths whose widths are chosen based on the spectra within the scene [7] [6]. Another technique for dealing with these scenes is hand-selecting the basis functions using knowledge of the spectra in the scene. For example, for fluorescent lights, one could ensure that point samples were positioned at the emission lines.

Here, we present an alternative method for the selection of basis functions, gaining insight from studies done on the construction of linear models of surface reflectances and spectral power distributions [3] [8] [9] [15] [12]. Most of these studies have stressed the use of characteristic vector analysis or principal component analysis to characterize lights and surfaces. This technique can be applied to the rendering problem to provide an automated method for selecting an efficient set of basis functions.

#### Characteristic Vector Analysis

Given a set of spectral power distributions, characteristic vector analysis computes an ordered set of functions such that the first $m$ functions are the "best" $m$ functions for approximating the distributions. Here, "best" is measured in terms of least squared error between the actual and the approximating spectra. Formally, for the approximation of

the spectral power distribution

$$I(\lambda) \approx \sum_{i=1}^{m} \epsilon_i E_i(\lambda), \tag{27}$$

the basis functions, $E_i(\lambda)$, are computed such that the sum of the approximation error over all of the lights in the set is minimized

$$Err = \sum_{I} \int [I(\lambda) - \sum_{i=1}^{m} \epsilon_i E_i(\lambda)]^2 d\lambda. \tag{28}$$

In practice, this set can be determined by placing the representative spectra in the columns of a matrix and performing a singular value decomposition [10] [11].

The task is then to find a representative set of spectra on which to perform the analysis. For the rendering problem, the basis functions should describe any spectral power distribution within the scene. The distributions contain contributions from the light sources themselves, from once-reflected light, and from multiply-reflected light. Therefore, an appropriate set of spectra is that set derived from possible interreflections within the scene. Given the spectral power distributions of the lights and the components of the surface reflectances in a scene, one can construct a tree of possible interreflection spectra (disregarding any geometry). The lights themselves would be included, and any number of reflections and interreflections could be included. The basis functions computed from a characteristic vector analysis of this set would then approximate these spectral power distributions.

If the number of spectral power distributions to fit is too large, this technique can become inefficient; the cost of computing the basis functions may exceed the savings in rendering time. Also, this method is inapplicable if one does not know *a priori* the spectral character of the surfaces and lights in the scene. However, for many scenes, this technique can readily be applied.

### 3.3 Examples

To demonstrate the use of characteristic vector analysis in selecting basis functions, we present two related examples. Both examples use the fluorescent light in Figure 1 to show the ability of this technique to handle complex spectra. In the first example, we determine the efficiency in computing the tristimulus values of a set of spectral power distributions, and in the second, we render a simple scene.

#### Tristimulus Values of Test Spectra

We select as sample spectra the twenty-four squares of the Macbeth Color Checker under the fluorescent light. A set of basis functions can be computed by performing a characteristic vector analysis on the set of twenty-five spectral power distributions given by the light itself and the light reflected from the twenty-four samples. Figure 2 shows the first three basis functions for this set; as can be seen, characteristic vector analysis preserves the narrow peaks that are found in the spectral power distribution of the light source.

From these basis functions, we compute the transform coefficients of the fluorescent light with Equation 2. Assuming only diffuse reflection and ignoring geometry, we use Equation 12 to compute a single matrix for each of the twenty-four

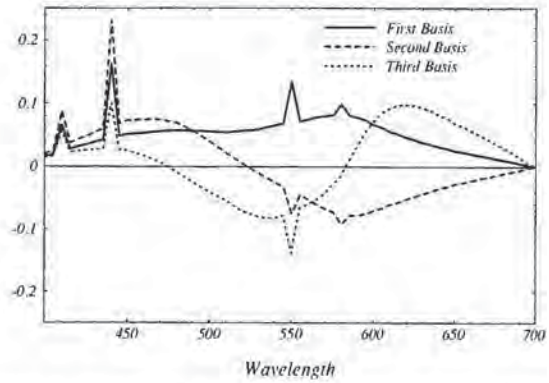



Figure 2: First three basis functions computed with characteristic vector analysis for fluorescent light reflected from the twenty-four squares of the Macbeth Color Checker.

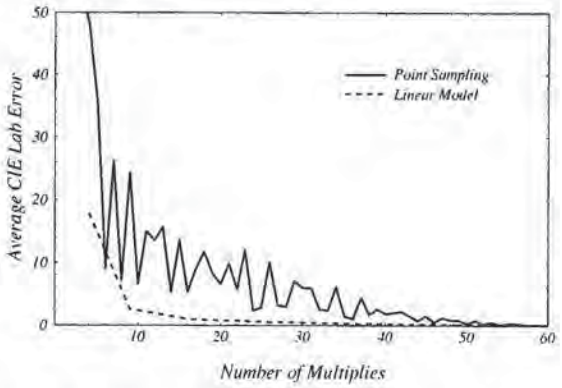



Figure 3: Average CIE Lab Error for set of spectra as a function of the number of multiplies per reflection for evenly spaced point samples and for the general linear transform computed with characteristic vector analysis.

surfaces in the color checker. The product of the vector of coefficients with each of these matrices gives column vectors containing the coefficients of the reflected light. From these vectors, we compute the linear model approximation to the tristimulus values of each of the twenty-four patches with Equations 18-20. The average CIE Lab error in units of $\Delta E$ [21] can then be calculated as a function of the number of basis functions. For reference, we also compute this error as a function of the number of evenly spaced point samples for Riemann summation. To compare the two methods in terms of their computational intensiveness, we plot in Figure 3 the errors as a function of the number of multiplies per reflection.

The general linear model is significantly more efficient than point sampling; the latter shows severe oscillations from the sampling error in computing the narrow peaks in the fluorescent light. Clearly, the point sampling method should (and would) be amended for the fluorescent light case. The most natural method is to ensure point samples lie on the narrow peaks and are weighted appropriately during the integration. This is tantamount to hand-selecting a general

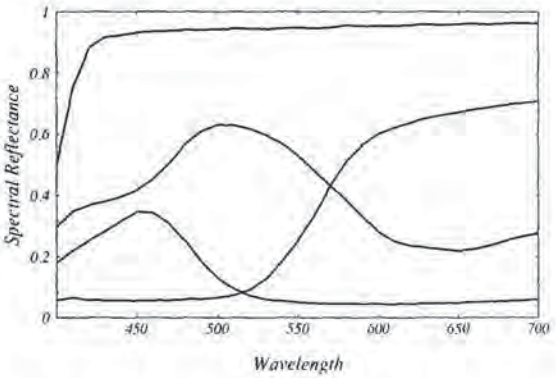



Figure 4: Four surface reflectances from the Macbeth Color Checker used in the example image.

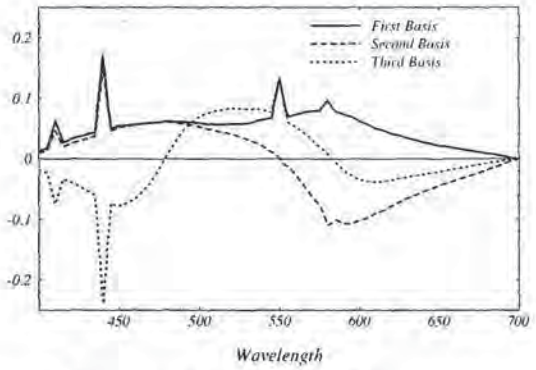



Figure 5: First three basis functions of the general linear model computed with characteristic vector analysis for the example image.

linear model. Characteristic vector analysis is attractive because it matches most anomalies in the spectra without the user being required to address each one distinctly.

### Image Generation

We now apply characteristic vector analysis to select basis functions for ray tracing of a simple scene under fluorescent light. The four distinct surface reflectances in the scene are taken from the Macbeth Color Checker and are shown in Figure 4. To compute the basis functions, we perform a characteristic vector analysis on the set of spectra consisting of the light source itself, all single reflections, and all second interreflections from the four surface samples; the first three basis functions are shown in Figure 5. These functions are used to compute the column vector of the light source and the ambient, diffuse, and specular reflectance matrices for each of the surfaces in the scene.

Figure 6 shows the resultant images for four different numbers of basis functions. The top left image in the figure displays the full resolution rendering of the scene computed at one nanometer intervals. The two columns display the general linear model and evenly spaced point sampling for the same number of multiplies per reflection. The left column shows the general model with 2, 3, 4, and 5 basis functions from top to bottom, and the right column shows 4, 9, 16, and 25 evenly spaced point samples from top to bottom. The linear model based on characteristic vector analysis is superior for all images; with just three basis functions, it is virtually identical to the full resolution image.

## 5 Conclusions

We have presented a general description of the use of linear transform methods in synthetic image generation. This formalism requires that all spectral power distributions be described with respect to a set of orthonormal basis functions. The spectral power distributions are represented by column vectors, and the surfaces are described by matrices. Reflection during the rendering procedure takes the form of matrix multiplication. Because this process is linear, it allows for easy implementation. In addition, this framework guides the choice of basis functions for efficient rendering.

We have discussed two possibilities for the selection of the basis functions, Riemann summation for efficient point sampling and characteristic vector analysis of a representative set of spectra in the scene. Point sampling based on Riemann summation is effective when the spectral power distributions in a scene are well described with low-order Fourier components. The method based on characteristic vector analysis is of comparable efficiency to point sampling techniques when the scenes contain smoothly varying spectra, and it can be significantly more efficient for scenes with complex spectra. We demonstrated this by rendering a scene illuminated by fluorescent light.

A promising direction of future work is the investigation of basis functions that make the rendering procedure more efficient; the techniques in [12] are potentially useful to this end. In addition, we have focussed in this paper on minimizing the cost of full spectral rendering, but the flexibility of the general method might be useful for other issues in computer graphics, such as texturing, that deal with spectral information during rendering.

## Acknowledgements

This material is based upon work supported under a National Science Foundation Graduate Fellowship and partially supported by the National Science Foundation under Grant NSF ECS 88-15815. The author would like to thank Lambertus Hesselink, Marc Levoy, and Paul Ning for helpful discussions. He especially would like to thank Brian Wandell for helpful discussions and for providing the spectral data used in this work.

## References

[1] Borges, Carlos. Trichromatic Approximation for Computer Graphics Illumination Models. Proceedings of SIGGRAPH '91 (Las Vegas, Nevada, July 28-August 2, 1991). In *Computer Graphics 25*,4 (July 1991),101-104.

[2] Borges, Carlos. *Numerical Methods for Illumination Models in Realistic Image Synthesis.* PhD dissertation, University of California, Davis, 1990.

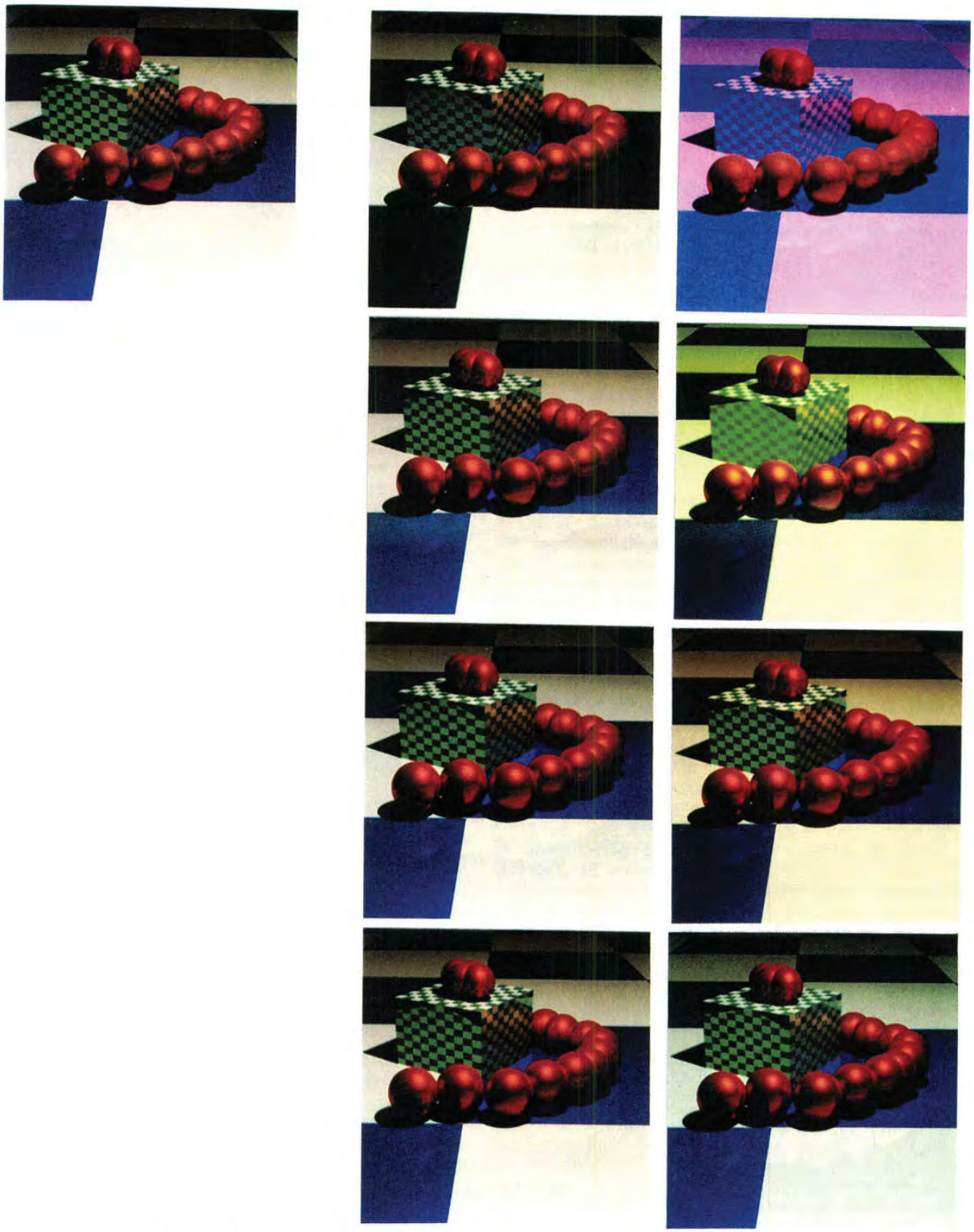


Figure 6: Comparison of general linear model with evenly spaced point sampling. The top left image is a full resolution image computed at one nanometer steps; the left image in each row is the general model with 2, 3, 4, and 5 basis functions from top to bottom; the right image is evenly spaced point sampling with 4, 9, 16, and 25 samples.

[3] Cohen, Jozef. Dependency of the Spectral Reflectance Curves of the Munsell Color Chips. *Psychon. Sci. 1* (1964), 369-370.

[4] Cowan, William. An Inexpensive Scheme for Calibration of a Color Monitor in Terms of CIE Standard Coordinates. Proceedings of SIGGRAPH '83 (Detroit, Michigan, July 25-29, 1983). In *Computer Graphics 17*,3 (July 1983), 315-321.

[5] Davis, P. and Rabinowitz, P. *Methods of Numerical Integration.* Academic Press, New York, 1975.

[6] Hall, Roy. *Illumination and Color in Computer Generated Imagery.* Springer-Verlag, New York, 1989.

[7] Hall, Roy and Greenberg, Donald. A Testbed for Realistic Image Synthesis. *IEEE Computer Graphics and Applications 3* (1983), 10-20.

[8] Judd, Deane, MacAdam, David, and Wyszecki, Gunter. Spectral Distribution of Typical Daylight as a Function of Correlated Color Temperature. *J. Opt. Soc. Am. 54*,8 (1964), 1031-1040.

[9] Maloney, Laurence. Evaluation of linear models of surface spectral reflectance with small numbers of parameters. *J. Opt. Soc. Am. A 3*,10 (1986), 1673-1683.

[10] Maloney, Laurence. *Computational Approaches to Color Constancy.* PhD dissertation, Stanford University, 1985.

[11] Mardia, K., Kent, J., and Bibby, J. *Multivariate Analysis.* Academic, London, 1979.

[12] Marimont, David and Wandell, Brian. Linear models of surface and illuminant spectra. *J. Opt. Soc. Am. A 9*,11 (1992), 1905-1913.

[13] McCamy, C., Marcus, H., and Davidson, J. A Color Rendition Chart. *J. Appl. Photographic Engrg. 11*,3 (1976), 95-99.

[14] Meyer, Gary. Wavelength Selection for Synthetic Image Generation. *Computer Vision, Graphics, and Image Processing 41* (1988), 57-79.

[15] Parkkinen, J., Hallikainen, J., and Jaaskelainen, T. Characteristic Spectra of Munsell Colors. *J. Opt. Soc. Am. A 6*,2 (1989), 318-322.

[16] Raso, Maria, and Fournier, Alain. A Piecewise Polynomial Approach to Shading Using Spectral Distributions. Proceedings of Graphics Interface '91. (Calgary, Alberta, June 3-7, 1991), 40-46.

[17] Smith, Brent, Spiekermann, Charles, and Sember, Robert. Numerical Methods for Colorimetric Calculations: A Comparison of Integration Methods. *COLOR Research and Application 17*,6 (1992), 384-393.

[18] Smith, Brent, Spiekermann, Charles, and Sember, Robert. Numerical Methods for Colorimetric Calculations: Sampling Density Requirements. *COLOR Research and Application 17*,6 (1992), 394-401.

[19] Wallis, Robert. Fast computation of tristimulus values by use of Gaussian quadrature. *J. Opt. Soc. Am. 65*,1 (1975), 91-94.

[20] Wandell, Brian. The Synthesis and Analysis of Color Images. *IEEE Trans. on Pattern Analysis and Machine Intelligence, PAMI-9*,1 (1987), 2-13.

[21] Wyszecki, Gunter and Stiles, W.S. *Color Science: Concepts and Methods, Quantitative Data and Formulae.* John Wiley and Sons, 1982.



# Combining Hierarchical Radiosity and Discontinuity Meshing

*Dani Lischinski* *Filippo Tampieri* *Donald P. Greenberg*

Program of Computer Graphics
Cornell University
Ithaca, NY 14853

## ABSTRACT

We introduce a new approach for the computation of view-independent solutions to the diffuse global illumination problem in polyhedral environments. The approach combines ideas from hierarchical radiosity and discontinuity meshing to yield solutions that are accurate both numerically and visually. First, we describe a modified hierarchical radiosity algorithm that uses a discontinuity-driven subdivision strategy to achieve better numerical accuracy and faster convergence. Second, we present a new algorithm based on discontinuity meshing that uses the hierarchical solution to reconstruct an object-space approximation to the radiance function that is visually accurate. Our results show significant improvements over both hierarchical radiosity and discontinuity meshing algorithms.

**CR Categories and Subject Descriptors:** I.3.3—[**Computer Graphics**]: Picture/Image Generation; I.3.7—[**Computer Graphics**]: Three-Dimensional Graphics and Realism.

**Additional Key Words and Phrases:** diffuse reflector, discontinuity meshing, global illumination, hierarchical radiosity, Mach bands, photorealism, quadratic interpolation, radiance function, radiosity, reconstruction, shadows, view-independence.

## 1 INTRODUCTION

Computing solutions to the global illumination problem is an essential part of photorealistic image synthesis. In this paper, we are interested in computing *view-independent* (or *object-space*) solutions for global illumination. Such solutions provide an approximation to the radiance function across each surface in the environment. Once a solution is computed, images from any viewpoint can be rendered with a relatively small additional effort. These methods are particularly attractive for applications such as architectural design, interior design, lighting design, illumination engineering, and virtual reality, in which the need for multiple views or walk-throughs of static environments arises.

So far, most view-independent methods have been derived from the radiosity method that was originally developed to solve radiative heat transfer problems [23]. Computer graphics researchers adopted this method to compute the global illumination of diffuse polyhedral environments [10, 7, 19]. Radiosity has been extended and improved dramatically since, but there is still much to be done before the method can become a useful tool for its intended users.

The goal of our research is to develop an efficient radiosity system that satisfies the following requirements:

**Objective (numerical) accuracy:** Solutions produced by the system should converge rapidly to the exact solution. This requirement may seem obvious, however, in the computer graphics community results of simulations are too often judged solely by their visual appearance.

**Subjective (visual) accuracy:** While visual appearance should not be used to judge the objective accuracy of the simulation, it is still very important, since the image is the final product. Clearly, accurate visual appearance can be achieved through numerically accurate simulation (if the underlying model is physically accurate.) Unfortunately, experience has shown that the human visual system is extremely sensitive to small perceptual errors that are difficult to quantify. The simulated environments can be very complex and, therefore, the computation of ultra-accurate solutions is generally impractical. Thus, we must have means of producing visually acceptable images even from coarse solutions.

**Ease of control:** (i) The system should be controllable by users who are not necessarily familiar with its inner workings. Therefore, the control parameters should be intuitive and small in number. (ii) In many cases (such as early design stages) the user is interested in a quick solution, even if not exceedingly accurate. At other times, one might be willing to wait overnight for a reliable solution. Therefore, the system should provide the user with the option to trade speed for accuracy.

Most radiosity systems do not satisfy any of these requirements. There are no error bounds on the solutions, because approximations are often used without justifications regarding their impact on the accuracy of the results. The resulting images typically exhibit many visual artifacts such as Mach bands, light and shadow leaks, jagged shadow boundaries, and missing shadows. Radiosity systems are seldom user-friendly and require massive user intervention: typically, a time consuming trial-and-error process is required to produce an image that looks right. Baum *et al.* [1] and Haines [12] provide good discussions of the various pitfalls of radiosity.

In this paper we present a new radiosity method, which comes closer to satisfying our goals. The new method combines two recently developed approaches: hierarchical radiosity [14] and discontinuity meshing [15, 18]. First, we present an improved hierarchical radiosity algorithm that uses a discontinuity-driven subdivision strategy to achieve better numerical accuracy and faster convergence. Second, we describe a new algorithm based on discontinuity meshing that uses the hierarchical solution to reconstruct a visually accurate approximation to the radiance function. Thus, results of

 ACM-0-89791-601-8/93/008/0199 $01.50

high visual quality can be obtained even from coarse global illumination simulations. Previous attempts to improve the visual quality of radiosity solutions were described by Nishita and Nakamae [19], Kok and Jansen [17], Chen *et al.* [4], and Reichert [20]. In all of these cases, however, the improvement takes place in image space, after the view and the resolution have been specified. Our method, instead, operates entirely in object space, and the improved solution is view-independent.

## 2 HIERARCHICAL RADIOSITY

The traditional radiosity approach [10, 7] discretizes the environment into $n$ elements and solves a linear system of $n$ equations, where the radiosities of the elements are the unknowns. The most serious drawback of this approach is the need to compute the $O(n^2)$ coefficients of the linear system, corresponding to the interactions (transfers of light energy) between pairs of elements. In addition to the overwhelming computational complexity, most of these computations are performed to unnecessarily high accuracy, while some are not sufficiently accurate.

Hierarchical radiosity (HR) [14] overcomes these problems by decomposing the matrix of interactions into $O(n)$ blocks, for a given accuracy. These blocks correspond to interactions of roughly equal magnitude, and the same computational effort is required for computing each block. HR operates by constructing a hierarchical subdivision of each input surface. Each node in the hierarchy represents some area on the surface. Two nodes are linked together if the interaction between their corresponding areas can be computed within the required accuracy; otherwise, the algorithm attempts to link their children with each other. Each link corresponds to a block in the interaction matrix.

HR has several important advantages: it is fast, the errors in its approximations are bounded, and it is controlled by only two parameters: the error tolerance and the minimum node area. The smaller the values of these parameters, the more accurate (and expensive) the solution becomes. Thus, HR satisfies our goals of objective accuracy and ease of control.

However, the HR algorithm still suffers from shadow leaks and jagged shadow boundaries. This occurs because surfaces are subdivided regularly, not taking into account the geometry of the shadows. HR uses point sampling to classify the inter-visibility between two surfaces, so it is prone to missing small shadows altogether. Of course, as the user-specified tolerance becomes smaller, the solution becomes more accurate, and the visual artifacts decrease. Nevertheless, images of high visual quality can require solutions of prohibitively high accuracy.

The number of links created by HR is $O(n + m^2)$ where $n$ is the final number of nodes and $m$ is the number of input surfaces. As the complexity of the environment increases, the $m^2$ term eventually becomes dominant, drastically reducing the efficiency of the algorithm. As pointed out by Smits *et al.* [22], this problem could be solved by grouping the input surfaces into higher level clusters. This is an interesting research topic by itself, and it will not be pursued in this paper.

## 3 DISCONTINUITY MESHING

Radiosity methods typically attempt to approximate the radiance function with constant elements and use linear interpolation to display the result. The actual radiance function, however, is neither piecewise constant nor piecewise linear. It is usually smooth, except along certain curves across which discontinuities in value or in derivatives of various order may occur. Discontinuities in radiance functions are discussed in detail elsewhere [16, 15, 18]; what follows is a brief summary of the various types of discontinuity and their causes.

The most significant discontinuities are discontinuities in the radiance function itself (denoted $D^0$). They occur along curves of contact or intersection between surfaces. Discontinuities in the first and the second derivatives ($D^1$ and $D^2$, respectively) occur along curves of intersection between surfaces in the environment and *critical surfaces* corresponding to qualitative changes in visibility, or *visual events*. Visual events in polyhedral environments can be classified into two types [9]: EV events defined by the interaction of an edge and a vertex, where the critical surface is a planar wedge; and EEE events defined by the interaction of three edges, where the critical surface is a part of a quadric. Discontinuities of higher than second order are also possible [16].

Discontinuities are very important both numerically and visually: all the boundaries separating unoccluded, penumbra, and umbra regions correspond to various discontinuities. When a discontinuity curve crosses a mesh element, the approximation to the radiance function over that element becomes less accurate. The resulting errors usually correspond to the most visually distracting artifacts in radiosity images. The traditional radiosity approach uses adaptive subdivision [8] to reduce these errors, however there are several problems with this approach. First, the user must specify an initial mesh that is sufficiently dense, or features will be lost. Second, the shape of the mesh is determined by the geometry of the surface being meshed, and the discontinuities are not resolved exactly. As a result, many small elements are created as the method attempts to converge to shadow boundaries. Furthermore, although the resulting solution may be of adequate visual quality for some views, artifacts may become visible as the view changes (e.g., when we zoom in on a surface.)

Discontinuity meshing (DM) algorithms compute the location of certain discontinuities and represent them explicitly, as boundaries, in the mesh. This leads to solutions which are both numerically and visually more accurate. Another advantage is that higher order elements can be used much more effectively in conjunction with discontinuity meshes [16]. Several algorithms have been described that use the idea of discontinuity meshing to various extents [1, 3, 6, 15].

Recently, a progressive radiosity DM algorithm was described by the authors [18]. The meshing in this algorithm is automatic. Using analytical visibility and form factor computations followed by quadratic interpolation it has produced radiosity solutions of impressive visual accuracy. This algorithm was also shown to be numerically accurate [24].

However, this method is too expensive for computing converged solutions of complex environment and only offers limited user control in trading off speed for accuracy. The main reason for this is that all energy transfers are computed very accurately, regardless of their magnitude.

## 4 A COMBINED APPROACH

Hierarchical radiosity and discontinuity meshing seem to complement each other in their strengths and weaknesses: HR is fast, but the visual appearance of the results can be disappointing; DM, on the other hand, has produced visually accurate results, but so far it has been too expensive for simulation of complex environments. This observation motivated us to look for ways of merging the two methods. Our investigation resulted in the following two-pass approach:

**The global pass** uses a modified HR algorithm to compute a radiosity solution within a prespecified tolerance. Instead of regular quadtree subdivision, the modified algorithm subdivides surfaces along discontinuity segments. This improves the numerical accuracy and results in faster convergence.