are available for checking interference between components as the user interactively assembles the object. (Alternatively, interference detection can be disabled.) The first method is the fastest and least accurate, as it uses only protruding (peg, dovetail, etc.) features and ray casting to determine whether the feature intersects the other component where there is no hole. It is easy to see that surface-surface intersections will be missed unless one of the surfaces is a peg feature. The second method is slightly slower but somewhat more accurate. It looks at the corners of all the surfaces making up the components being connected. For each corner, a ray is cast to determine whether the corner is inside or outside the other component. If any corner is inside the other component, interference is detected. This method may miss some intersections between curved elements. The third method performs a true boolean intersection between every surface in the first component and every surface in the second component. However, this operation is slow and becomes slower as the number of surfaces in the components increases (which is guaranteed to happen as the assembly grows more and more complex), even when bounding box checking is used to eliminate pairs of surfaces which definitely do not interfere.

Determining a part's translational degrees of freedom is closely related to being able to determine whether or not the part is removable from the assembly by a single translation. It is also sometimes important to check that a given part is constricted in such a way that it cannot be removed. For example, if a user is designing a fixture for machining, he may wish to make sure that all the degrees of freedom of the stock being held in the fixture are constrained and the stock cannot accidentally slide out of the fixture while being machined.

The user can choose a component or set of components currently in the assembly for examination (if more than one component is selected, they are examined as a group), and initiate the computation of translational degrees of freedom from the mated features by selecting a menu option. If a component's motion is not constrained, the direction vectors representing the directions in which the component is free to move form a sphere. If the component mates with a single flat surface on some side, that mating reduces the degrees of freedom to half a sphere, because all the direction vectors with a factor in the direction of the constraining surface's normal are eliminated. Similarly, if the component mates with a surface that is not flat, all the direction vectors with a component in the direction of any normal on the constraining surface are eliminated. This situation is illustrated in Figure 6, which shows a two-dimensional analogue to the preceding explanation.

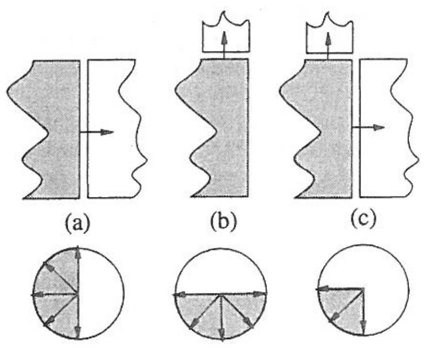

Figure 6. Determining how a two-dimensional component's translational freedom is constrained by two flat-edge matings. (a, b) Each mating constrains the movement of the shaded component to any direction in the shaded semicircle. (c) The final set of directions is determined by intersecting all the semicircles.

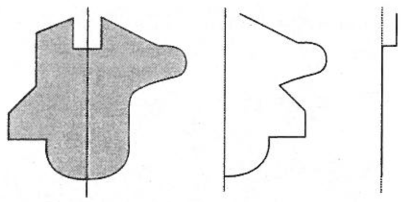

Figure 7. A component, axis of rotation, and the outer and inner contours for determining rotational freedom. Our system computes a polyline approximation to such a contour.

All the feature matings of the component(s) being analyzed are examined. Flat surfaces add a single half-sphere constraint. Hole/hole matings add a single half sphere constraint since a surface is assumed to be present whenever a hole is, even if that surface is not specifically given as a mating feature. Sculptured surfaces' normals are sampled, and each normal adds a half-sphere constraint. A hole/peg mating constrains the two components to move only along the axis line of the peg and hole. All of these constraints are intersected to determine a section of a sphere, a single direction, or two opposing directions which satisfy all the constraints. The set of valid directions of motion is displayed as a collection of vectors.

Similarly, the user may wish to determine if a component (or group of components) can rotate in place. In order to be able to rotate a component, the component must participate in a mating involving a round peg and hole. This condition is necessary to be able to determine an axis of rotation for the component. The axis of rotation is taken to be that of the mated peg and hole. There is no graphical display for the results of rotational freedom analysis, but the user is shown a message summarizing the result.

If the component participates in more than one peg/hole mating, rotation is obviously not possible. Otherwise, a polyline approximation to the outlines which would be swept out by rotating the components about the axis of rotation is found. Each control point of each surface is examined, and its distances from the axis of rotation and from the lowest point on the component are determined. An outer and inner contour are then computed from these points for the rotating component (see Figure 7). Likewise, an outer and inner contour are computed for the stationary component. If the outer contour of the rotating component is everywhere closer to the axis of rotation than the inner contour of the stationary component, we may conclude that free rotation is possible. The same conclusion can be made if the inner contour of the rotating component is outside the outer contour of the stationary component. A more accurate algorithm should be developed for this step, since the existing one involves too much approximation. However, it serves to illustrate the usefulness of rotational analysis.

6. Exploded View Illustration

Exploded view illustrations are common in technical documentation because they are easy to understand even by people without area expertise. Yet, while they are easy to understand, they are deceptively difficult and time consuming to create manually. We use information about part geometries, mated features, and disassembly directions indicated by the mating of features to automate much of the process of creating such illustrations.

As part of the assembly design system, a tool has been developed to create an exploded view of a completed assembly when that

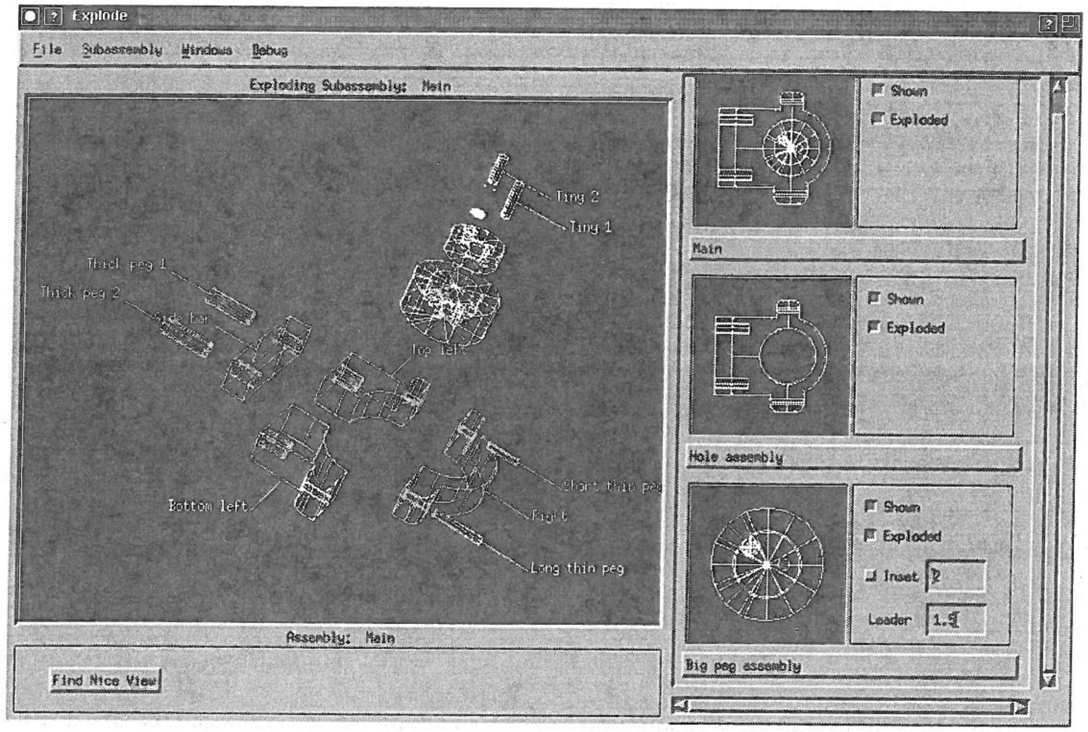

Figure 8. The exploded view tool.

assembly had been created and put together using the other parts of the system. Even though the creation of exploded views is largely automated, a good deal of user control is allowed, because even the most clever algorithm can sometimes generate results which are aesthetically unpleasing or otherwise not exactly what the user wants. The difficult or repetitive operations, such as keeping track of part connections, computing a perspective view of the geometry, and finding the approximate locations for the parts in the final illustration are done by the algorithm. The user decides whether the illustration should be an exploded view of the whole assembly or of a subassembly, and can also alter the distance of explosion of any part or subassembly, hide or show any component in the current illustration, cause any subassembly to be shown exploded or unexploded, request enlarged views of small subassemblies, and specify which parts should have text labels. (A screen capture of the utility is shown in Figure 8.) The explosion of the assembly is generated in three dimensions, but the view can be manipulated by the user until it is satisfactory, then the geometry and view can be saved for output to a rendering program.

Several preprocessing steps are involved in creating an exploded view illustration. Although the geometries of the parts are initially stored in their own coordinate systems (the coordinate systems in which they have been modeled), for the generation of exploded views it is more convenient to store all geometries relative to the coordinate system of the completed assembly. Since each component stores the transformation which assembles it to its parent, this is easily accomplished. The next step involves figuring out the minimum distance each component must be exploded from its parent subassembly to completely remove it from that subassembly. This distance only needs to be computed once for each component, since the basic relationships among the assembly components do not change. Bounding boxes are computed for the component and for its parent subassembly minus the component. Then the minimum distance to separate the bounding boxes along the direction of explosion is found. (The explosion direction is indicated by the mating conditions present between the two components.)

The creation of an exploded view is a recursive operation. Each child of the main assembly is translated out from the assembly's origin along its removal direction by its minimum explosion distance plus some additional value. (This extra distance serves to further separate the parts, giving the "exploded" look.) Each component of each of these subassemblies is, in turn, translated along its removal direction out from the origin of the exploded parent subassembly. The process continues until individual parts are reached. The transformations are accumulated through the levels of recursion, so each component has applied to it all of its ancestor subassemblies' translations, then finally, its own.

However, an exploded view illustration does not consist simply of a set of parts separated in space. Leader lines are drawn between parts, indicating from where each component was exploded. Parts may also exhibit identification labels. Leader lines are computed after the exploded geometries of all the components have been found. A single line per part is not enough to clearly show how that part connects to the other assembly components, so leader lines are drawn between every pair of mated features. Labels help identify parts in the illustration. Each label consists of a text string (the part's name, first assigned in the assembly planner) and a leader line connecting the label to the bounding box of the part.

7. Data Structure

All of the tools described here use a common basic data structure for the assembly. We have already mentioned that this structure stores the assembly hierarchically, as a tree (the approach is similar to that of Lee and Gossard [4]). This section summarizes what data is stored.

Each node representing an assembly component is capable of storing the information below; different pieces of information are added throughout the design process:

- The name of the component, given by the designer.
- A pointer to the subassembly of which the component is a part. The main assembly is the only one which has no parent.
- A direction in which the component can be removed from its parent assembly, determined from mating conditions.
- Textual comments about the component, entered by and useful for the designer.
- A list of the assembly features of the component.
- Information to create the transformation used to move this component from its geometry's local coordinate system into the coordinate system of its immediate parent subassembly.
- A list of component parts, if any. Individual parts have no components.
- The name of the file, if any, where the geometry of the component is stored. Usually, only individual parts have geometry. The geometries of subassemblies are derived from the geometries of their component parts.
- The geometry of the component.

Each assembly feature stores the following information:

- The geometric description of the feature. This includes the feature location and orientation, depth, radius, cross-section curve, number of threads per unit of length, and so on, as appropriate to the type of feature. Hole features also indicate whether or not they go all the way through the material and are open on both ends. The geometry is specified in the coordinate system of the component to which the feature belongs.
- Transformation information used for mating the feature with its matching feature.
- A back pointer to the component whose feature this is.
- The matching feature (if any) on some other component.
- The inheritance history of the feature. This includes the next and previous history links, as mentioned previously, and links to other features (if any) which were coalesced to create this one.

8. Future Work

A number of open issues still remain. First, a better mechanism for making design revisions should be created. Currently, the three clients we have described communicate through data files. It would be useful to be able to make changes in the assembly planner and have these changes propagate to the other clients, automatically updating the assembly components' geometries and connections. This is not an easy problem, however, especially if the topology of the assembly or of any of the parts changes. Next, more accurate and reliable assembly analysis algorithms should be developed. Also, the system currently only tests whether a part is removable along paths consisting of a single translation. Methods for dealing with more complex removal paths should be examined.

We can also envision extensions which would increase the usefulness of the system. For example, incorporating full-fledged kinematic analysis would enable designers to examine mechanisms to see if they perform as expected.

9. Conclusions

A system has been developed which integrates a number of design aids which have not been previously available together in order to help users design assemblies of mechanical parts. This research has shown how, by integrating initial and more detailed design into a single system, the designer's knowledge can be extracted in a natural way during the design process without overburdening the designer. Exploded view illustration has also been explored.

Acknowledgements

We would like to thank the students and staff of the Alpha_1 project, within which this work was developed. Thanks also go to Dr. Richard Riesenfeld, for his help in editing this paper. This work was supported in part by ARPA, under grant N00014-92-J-4113. 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] De Fazio, Thomas L. and Daniel E. Whitney, Simplified Generation of All Mechanical Assembly Sequences, *IEEE Transactions on Robotics and Automation*, RA-3(6), 1987, pp. 640-658.

[2] Eastman, Charles M., The Design of Assemblies, SAE Technical Paper No. 810197, Society of Automotive Engineers, Inc., 1981.

[3] Gui, Jin-Kang and Martti Mantyla, Functional Understanding of Assembly Modelling, *Computer-Aided Design*, June 1994, pp. 435-451.

[4] Lee, Kunwoo and David C. Gossard, A Hierarchical Data Structure for Representing Assemblies: Part 1, *Computer Aided Design*, 17(1), 1985, pp. 15-19.

[5] Lee, Sukhan and Yeong Gil Shin, Assembly Planning Based on Subassembly Extraction, *Proceedings of the 1990 IEEE International Conference on Robotics and Automation*, 1990, pp. 1606-1611.

[6] Lieberman, L. I. and M. A. Wesley, AUTOPASS: An Automatic Programming System for Computer Controlled Mechanical Assembly, *IBM Journal of Research and Development*, 21(4), 1977, pp. 321-333.

[7] Mattikalli, Raju S., Pradeep K. Khosla, and Yangsheng Xu, Subassembly Identification and Motion Generation for Assembly: A Geometric Approach, Engineering Design Research Center, Carnegie Mellon University, preprint.

[8] Shah, Jami J., Conceptual Development of Form Features and Feature Modelers, *Research in Engineering Design*, (2) 1991, pp. 93-108.

[9] Strip, David and Anthony A. Maciejewski, Archimedes: an Experiment in Automating Mechanical Assembly, preprint, to be presented at the International Symposium on Robotics and Automation, July 1990, Vancouver, B.C.

[10] Talukdar, Sarosh N. and Sergio W. Sedas, A Disassembly Planner, Technical Report EDRC-05-10-87, Engineering Design Research Center, Carnegie Mellon University, 1987.

[11] Tilove, Robert B., Extending Solid Modeling Systems for Mechanism Design and Kinematic Simulation, *IEEE Computer Graphics and Applications*, May/June 1983, pp. 9-19.

[12] Woo, Tony C. and Debasish Dutta, Automatic Disassembly and Total Ordering in Three Dimensions, preprint, to appear in *ASME Transactions, Journal of Mechanical Design*, (during or after 1989, exact year unknown).

Hierarchical and Variational Geometric Modeling with Wavelets

Steven J. Gortler[1] and Michael F. Cohen[2]

Department of Computer Science
Princeton University

Abstract

This paper discusses how wavelet techniques may be applied to a variety of geometric modeling tools. In particular, wavelet decompositions are shown to be useful for hierarchical control point or least squares editing. In addition, direct curve and surface manipulation methods using an underlying geometric variational principle can be solved more efficiently by using a wavelet basis. Because the wavelet basis is hierarchical, iterative solution methods converge rapidly. Also, since the wavelet coefficients indicate the degree of detail in the solution, the number of basis functions needed to express the variational minimum can be reduced, avoiding unnecessary computation. An implementation of a curve and surface modeler based on these ideas is discussed and experimental results are reported.

1 Introduction

Wavelet analysis provides a set of tools for representing functions hierarchically. These tools can be used to facilitate a number of geometric modeling operations easily and efficiently. In particular, this paper explores three paradigms for free-form curve and surface construction: control point editing, direct manipulation using least squares, and direct manipulation using variational minimization techniques. For each of these paradigms, the hierarchical nature of wavelet analysis can be used to either provide a more intuitive modeling interface or to provide more efficient numerical solutions.

In control point editing, the user sculpts a free-form curve or surface by dragging a set of control points. A better interface allows the user to directly manipulate the curve or surface itself, which defines a set of constraints. In a *least squares* paradigm, given a current curve or surface, the modeling tool returns the curve or surface that meets the constraints by changing the current control points by the least squares amount [1, 11].

The behavior of the modeling tool is determined by the type of control points and *basis functions* used to describe the curve or surface. With the uniform cubic B-spline basis, for example, the user's actions result in local changes at a predetermined scale. This is not fully desirable; at times the user may want to make fine changes of detail, while at other times he may want to easily make broad changes. Hierarchical B-splines offer a representation that allows both control point and least squares editing to be done at multiple resolutions [9]. Hierarchical B-splines, though, form an over-representation for curves and surface (i.e., any curve has multiple representations using hierarchical B-splines). As a result, the same curve may behave differently to a user depending on the particular underlying representation. In contrast, B-spline wavelets form a hierarchical basis for the space of B-spline curves and surfaces in which every object has a unique representation. Wavelet methods in conjunction with hierarchical B-splines provide a method for constructing a useful geometric modeling interface. This approach is similar to the one described by Finkelstein and Salesin [8]. In this paper we will discuss some of the various issues that are relevant to building such a modeling tool.

Variational modeling is a third general paradigm for geometric modeling[2, 28, 21]. In this setting, a user alters a curve or surface by directly manipulation, as above, defining a set of constraints. The variational modeling paradigm seeks the "best" solution amongst all answers that meet the constraints. The notion of best, which is formally defined as the solution that *minimizes some energy function*, is often taken to mean the *smoothest* solution.

In theory, the desired solution is the curve or surface that has the minimum energy of *all* possible curves or surfaces that meet the constraints. Unfortunately there is little hope to find a closed form solution [1]. Therefore, in practice, the "space" of parametric curves or surfaces is restricted to those represented by a linear combination of a fixed set of basis functions such as cubic B-splines. Given a set of n basis functions, the goal of finding the best curve or surface is then reduced to that of finding the best set of n coefficients. This reduction is referred to as the *finite element method* [27].

The general case requires solving a non-linear optimization problem. In the best case, the energy function is quadratic and the constraints are linear leading to a single linear system to solve. But even this can be costly when n is large since direct methods for matrix inversion require $O(n^3)$ time. To accelerate this process it is tempting to use gradient-type iterative methods to solve the linear system; these methods only take $O(n)$ time per iteration, due to the $O(n)$ matrix sparsity created by the finite element formulation.

[1]Currently at Microsoft Corp. and the Department of C.S., University of Washington. sjg@cs.washington.edu

[2]Currently at Microsoft Corp. mcohen@microsoft.com

[1]But see [20].

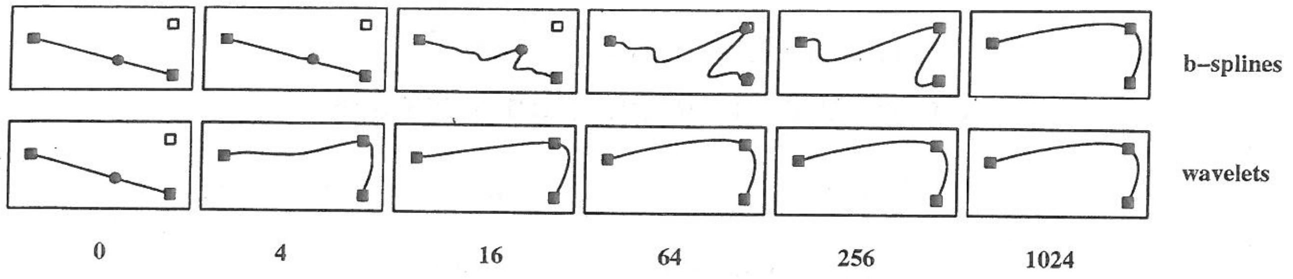

Figure 1: Minimum energy solutions subject to three constraints, found by the B-spline and wavelet methods after various numbers (0-1024) of iterations. (65 variables, 3 constraints). This illustrates the ill conditioning of the B-spline optimization problem.

Unfortunately, the linear systems arising from a finite element formulation are often expensive to solve using iterative methods. This is because the systems are ill-conditioned, and thus require many iterations to converge to a minimum [26, 25]. Intuitively speaking this occurs because each basis function represents a very narrow region of the answer; there is no basis function which can be moved to change the answer in some broad manner. For example, changing one coefficient in a cubic B-spline curve during an iteration alters the curvature in a local region only. In order to produce a broad smooth curve, the coefficients of the neighboring B-splines will move in next few iterations. Over the next many iterations, the solution process will affect wider and wider regions, and the effect will spread out slowly like a wave moving along a string. The result is very slow convergence (see Figure (1)). One method used to combat this problem is multigridding [26, 10], where a sequence of problems at different resolution levels are posed and solved.

An alternative approach, is to use a *wavelet* basis instead of a standard finite element basis [25, 23, 15, 22]. In a wavelet basis, the answer is represented hierarchically. This allows the solution method to alter the answer at any desired resolution by altering the proper basis function, and thus the ill-conditioning is avoided. In this paper we show how to use a wavelet construction, which is based on cubic B-splines, to quickly solve variational modeling problems in an elegant fashion.

Another problem with the finite element approach is choosing the density of the basis functions. If too few basis functions (too few B-spline segments or tensor product B-spline patches) are used then the solution obtained will be far from the actual minimum. If too many basis functions are used then unnecessary computation will be performed during each iteration (n is too big). In order to successfully choose a proper density, one must know how much detail exists in the variational minimum answer. Since, a priori, this is unknown, an efficient solver must be able to adaptively change the basis during the solution process [28], one needs an easy way to detect that too many or too few basis functions are being used. In addition, one needs a basis for which adding more detail, (i.e., refinement), is easy. Wavelets offer a basis where this task can be accomplished quickly and elegantly.

The work presented in this paper combines the wavelet approaches of [25], [12], and [16]. Like [25], this paper uses hierarchical basis functions as a pre-conditioner, so that fewer iterations are needed for convergence. Similar to [12] and [16], wavelets are also used as a method for limiting the solution method to the proper level of detail.

2 Geometric Representation

This paper will restrict itself to parametric representations of curves and surfaces. In this representation, a curve is defined as a 3 dimensional trajectory parameterized by t,

$$\gamma(t) = (X(t), Y(t), Z(t)) \tag{1}$$

and a surface is defined as

$$\gamma(s,t) = (X(s,t), Y(s,t), Z(s,t)) \tag{2}$$

which defines a three dimensional location for every parameter pair (s, t).

The parametric representation of a curve or surface is made up of three functions X, Y, Z, which are represented as a linear combination of basis functions. Just focusing on the X function, for curves this becomes

$$X(t) = \sum_j x_j \phi_{L,j}(t) \tag{3}$$

and for surfaces

$$X(s,t) = \sum_{j,k} x_{j,k} \phi_{L,j,k}(s,t) \tag{4}$$

where the x are scalar coefficients. In geometric modeling the univariate basis $\phi_{L,j}(t)$ is typically some "piecewise" basis, such as a cubic B-spline or the Bernstein (Bézier) basis, and the bivariate basis used for surfaces is the associated tensor product basis $\phi_{L,j,k}(s,t) \equiv \phi_{L,j}(s)\phi_{L,k}(t)$.

3 Hierarchical Geometric Descriptions

In this section we will briefly review some ways that curves and surfaces may be represented hierarchically.

Let us begin by discussing curves. For simplicity we will deal with the uniform cubic B-spline basis over the interval $[0 \dots 2^L]$ made up of translations of a single basis shape denoted $\phi(t)$. The cubic B-spline function $\phi(t)$ is supported over the interval $[0 \dots 4]$ and is made up of 4 cubic polynomial pieces joined with C^2 continuity. The complete uniform cubic B-spline basis is made up of translated copies $\phi_{L,j}(t)$ of the basis shape $\phi(t)$ (see Figure 2).

$$\phi_{L,j}(t) = \phi(t - j) \tag{5}$$

The index j represents the translation of a specific basis from the canonical B-spline left justified at zero, and L is the *level* or resolution of the basis. There are roughly 2^L functions in this basis [2]. In wavelet terminology, the space (or family) of curves spanned by all linear combinations of these basis functions is denoted V_L (e.g., V_L contains all functions that are piecewise cubic, with simple knots at the integers).

[2] A few extra basis functions are needed at the boundary. This paper will not discuss the technical details needed to handle all of the boundary constraints. This is discussed in many places including [4, 16, 8, 13].

3.1 Hierarchical B-splines

Forsey and Bartels [9] introduced hierarchical B-splines as a way of representing and modeling geometric objects hierarchically. Instead of using only B-spline basis functions at a single resolution L, they use a hierarchy of wider and wider B-spline functions

$$\phi_{i,j}(t) = \phi(2^{L-i}t - j) \tag{6}$$

for $0 \leq i \leq L$. For example, the basis functions $\phi_{L-1,j}$ at resolution level $L-1$ (with a support size of 8), are twice as wide as the basis functions $\phi_{L,j}$ at level L (with a support size of 4). These basis functions, $\phi_{L-1,j}$, span the space of piecewise cubic functions with knots at all *even* integers; in wavelet terminology, this space is called V_{L-1}. On each coarser level, the space V_i has half as many basis functions, and they are all twice as wide.

According to the well known B-spline knot insertion algorithm [6, 9, 3] one can define double width B-spline basis functions as linear combinations of single width B-spline basis functions.

$$\phi_{i-1,j} = \sum_k h_{k-2j}\ \phi_{i,k} \tag{7}$$

where the sequence h is

$$h[0..4] = \{\frac{1}{8}, \frac{4}{8}, \frac{6}{8}, \frac{4}{8}, \frac{1}{8}\} \tag{8}$$

(see Figure (2)). As a result of Equation (7) the set of functions in V_{i-1} is a subset of the functions in V_i.

$$V_{i-1} \subset V_i \tag{9}$$

The basic idea of Forsey and Bartels is to allow the user to control the coefficient of each of these basis functions $\phi_{i,j}$ by exposing a control mesh at each level i.

3.2 Wavelets

Hierarchical B-splines $\{\phi_{i,j}\}$ do not form a *basis* for the function space V_L; they form an *overrepresentation* for all the curves in V_L. In other words, there are many linear combinations of the basis functions defining the same curve or surface. Wavelets are a representation related to hierarchical B-splines, that form a basis; in a wavelet basis, all curves in V_L have a unique representation.

Rather than add a new finer set of B-splines at each level of the hierarchy, the idea is to look for a set of functions $\psi_{i,j}$ that "fills in" the space between the adjacent B-spline spaces, V_i and V_{i+1}. These wavelet functions $\psi_{i,j}$ represent the *detail* of the curve that cannot be represented by the double width B-splines, $\phi_{i,j}$. For each i, the space of functions spanned by the $\psi_{i,j}$ is called W_i.

There is actually quite a bit of freedom in choosing these $\psi_{i,j}$ functions, and hence the space W_i, as long as every function in V_{i+1} can be written as a combination of some function in V_i and some function in W_i. This is notated as

$$V_{i+1} = V_i \dot{+} W_i \tag{10}$$

Just like the Hierarchical B-splines are all scales and translates of a single shape $\phi(t)$, (see Equation (5)) in a wavelet basis, the basis functions $\psi_{i,j}$ are all translates and scales of a single function $\psi(t)$.

$$\psi_{i,j}(t) = \psi(2^{L-i}t - j) \tag{11}$$

Also similar to hierarchical B-splines, in a wavelet basis, the basis functions on one level can be defined by linearly combining B-spline functions on the next finer resolution,

$$\psi_{i-1,j} = \sum_k g_{k-2j}\ \phi_{i,k} \tag{12}$$

And as a result $W_{i-1} \subset V_i$. There is some degree of freedom in choosing the sequence g, as long as the property expressed by Equation (10) holds. One such sequence given by Cohen et al. [5] is [3] (see Figure (3)).

$$g[0..10] = \{\frac{5}{256}, \frac{20}{256}, \frac{1}{256}, \frac{-96}{256}, \frac{-70}{256}, \frac{280}{256}, \frac{-70}{256}, \frac{-96}{256}, \frac{1}{256}, \frac{20}{256}, \frac{5}{256}\}$$

Due to the relationships of Equations (7) and (12), if some function $X(t)$ in V_i has been expressed as a linear combination of the B-spline basis function at level $i-1$ and wavelet basis functions at level $i-1$, using coefficients notated by $x_{\phi_{i-1,j}}$ and $x_{\psi_{i-1,j}}$,

$$X(t) = \sum_j x_{\phi_{i-1,j}}\ \phi_{i-1,j}(t) + x_{\psi_{i-1,j}}\ \psi_{i-1,j}(t) \tag{13}$$

then, $x_{\phi_{i,j}}$, the coefficients of the same function, with respect to the B-spline basis at level i may be found with

$$x_{\phi_{i,j}} = \sum_k h_{j-2k}\ x_{\phi_{i-1,k}} + \sum_k g_{j-2k}\ x_{\psi_{i-1,k}} \tag{14}$$

and now $X(t) = \sum_j x_{\phi_{i,j}}\ \phi_{i,j}(t)$

Inversely, if some function has been expressed with respect to B-spline functions at level i, then the representation of Equation (13) may be found using the formula

$$x_{\phi_{i-1,j}} = \sum_k \tilde{h}_{k-2j}\ x_{\phi_{i,k}} \tag{15}$$

$$x_{\psi_{i-1,j}} = \sum_k \tilde{g}_{k-2j}\ x_{\phi_{i,k}} \tag{16}$$

using the proper inverse sequences $\tilde{g}$ and $\tilde{h}$. Equation (15) *projects* the high resolution curve from V_i into the lower resolution space V_{i-1}; this is, in some sense, a smoother approximation of the object in V_i. Equation (16) captures the detail that is lost in this projection, and represents it using a basis for the space W_{i-1}.

When using the h and g sequences given by Cohen et al [5], the proper inverse sequences $\tilde{h}$ and $\tilde{g}$ are

$$\tilde{h}[-3..7] = \{\frac{-5}{256}, \frac{20}{256}, \frac{-1}{256}, \frac{-96}{256}, \frac{70}{256}, \frac{280}{256}, \frac{70}{256}, \frac{-96}{256}, \frac{-1}{256}, \frac{20}{256}, \frac{-5}{256}\}$$

$$\tilde{g}[3..7] = \{\frac{1}{8}, \frac{-4}{8}, \frac{6}{8}, \frac{-4}{8}, \frac{1}{8}\} \tag{17}$$

3.3 The Basis

Every function in V_L, expressed as a combination of the B-spline basis functions $\{\phi_{L,j}\}$, can be expressed uniquely in the *wavelet basis* is made up by the functions

$$\{\phi_{0,j}, \psi_{i,j}\} \quad 0 \leq i \leq L-1 \tag{18}$$

In the wavelet representation, the function is expressed hierarchically.

Transforming a function's representation from B-spline to wavelet coefficients may be done with the pyramid procedure `coef_pyrm_up`. This procedure may be performed in linear time by successively applying the transformation of Equations (15) and (16). This linear transformation may be denoted by the matrix $\mathbf{W}$. The inverse transformation (denoted by the matrix $\mathbf{W}^{-1}$), may be implemented with the procedure `coef_pyrm_down`, which succesively applies the transformation of Equation (14).

If `coef_pyrm_up` is implemented using the h and g sequences instead of the $\tilde{h}$ and $\tilde{g}$ sequences, then the resulting procedure may

[3] A different sequence is given by Chui [3] and generates a semi-orthogonal wavelet.

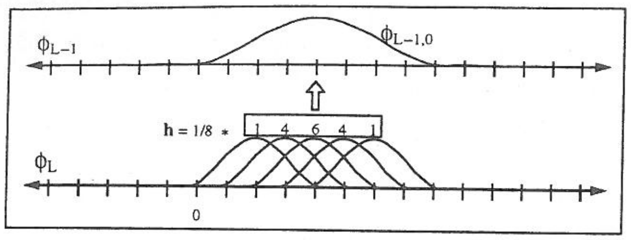

Figure 2: Five B-splines $\phi_{L,j}$ may be combined using the weights h to construct the double width B-spline $\phi_{L-1,0}$

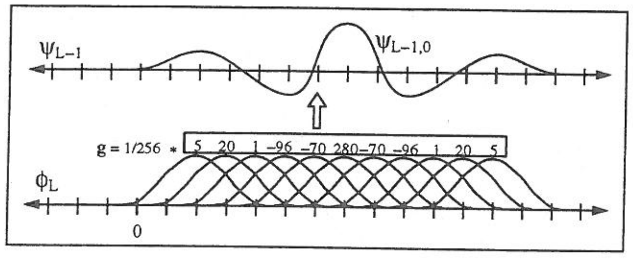

Figure 3: Eleven B-splines $\phi_{L,j}$ may be combined using the weights g to construct the wavelet function $\psi_{L-1,0}$

be called **basis_pyrm_up**, and it is represented by the matrix $\mathbf{W}^{-\mathbf{T}}$. If coef_pyrm_down is implemented using the $\tilde{h}$ and $\tilde{g}$ sequences instead of the h and g sequences, then the resulting procedure may be called **basis_pyrm_down**, and it is represented by the matrix $\mathbf{W}^{\mathbf{T}}$.

3.4 Surfaces

The ideas outlined above are easily extended to tensor product surfaces [3]. The uniform tensor product cubic B-spline basis is made up of the functions $\phi_{L,j}(s)\phi_{L,k}(t)$ The hierarchical uniform tensor product cubic B-spline representation is made up of the functions $\phi_{i,j}(s)\phi_{i,k}(t)$ for $0 \le i \le L$. On each coarser resolution of the hierarchy, there are $1/4$ the amount of ϕ basis functions.

The tensor product B-spline wavelet basis is made up of the functions [4]

$$\begin{array}{cc} \phi_{0,j}(s)\phi_{0,k}(t) & \phi_{i,j}(s)\psi_{i,k}(t) \\ \psi_{i,j}(s)\phi_{i,k}(t) & \psi_{i,j}(s)\psi_{i,k}(t) \end{array} \tag{19}$$

with i in $\{0 \dots L-1\}$.

Just like for curves, there are four pyramid procedures and associated $\mathbf{W}$ matrices.

4 Geometric Modeling with Wavelets

The styles of interactive control discussed in the introduction will be revisited in the context of hierarchical representations. *Multiresolution modeling* allows the user to interactively modify the curve or surface at different resolution levels. This allows the user to make broad changes while maintaining the details, and conversely detailed changes while maintaining the overall shape. Two types of hierarchical manipulation are considered, control point dragging and a direct manipulation involving solving a least squares problem.

In contrast, *variational modeling* allows the user to directly manipulate the curve or surface with the curve or surface maintaining some notion of overall smoothness subject to user imposed constraints. This physically based paradigm provides an intuitive means for shape control. Each of these paradigms will be explored in the context of wavelet bases which will be shown to provide the required hooks for such interaction and/or significant computational savings.

[4]This basis is known as the non-standard basis [3].

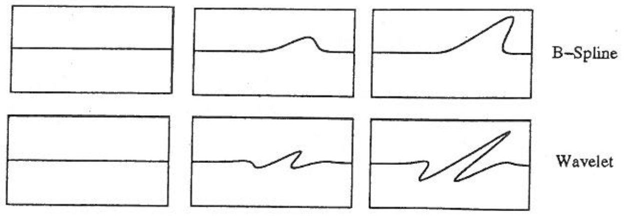

Figure 4: When B-spline coefficients are manipulated, the curve responds in a "hump" like fashion. When wavelet coefficients are manipulated, the curve responds in a "wave" like fashion.

4.1 Multiresolution Modeling

A multiresolution representation such as a hierarchical B-spline or wavelet representation may be used to implement a multiresolution modeling system. This section explores the choices that must be made when designing a multiresolution tool. Two related methods are described; direct control point manipulation and a least squares solver.

In control point modeling, the user is allowed to directly alter the coefficient values, by clicking and dragging on control points. In the least squares scheme [1, 11], the user can click and drag directly on the curve or surface, defining interpolation and tangent constraints. The system returns the curve or surface that satisfies these linear constraints ($\mathbf{A}\mathbf{x}' = \mathbf{b}$), by changing the coefficients by the least squares amount. Least square solutions can be found very inexpensively using the pseudoinverse [11]. The least squared problem can also be posed as a minimization problem [28], whose solution can be found by solving a sparse, well conditioned, linear system.

In multiresolution versions of these two schemes, the user chooses the resolution level i, and then only the quantities of basis functions on level i are altered. The locality of the effect on the curve or surface is directly tied to the chosen level i. In control point modeling, the control polygon at level i is manipulated by the user. In a least squares scheme, the user is provided a direct handle on the curve or surface itself, and the least squares solution is found only using the basis functions on level i. The least-squares approach offers a much more intuitive interface, and (for curves) works at interactive speeds.

One decision to be made is whether to expose the user to hierarchical B-splines or to wavelets. It is easy to see that manipulating wavelet basis functions does not produce an intuitive interface. Moving such a control point, and thus changing the amount of some wavelet basis function used, changes the solution in a "wave" like fashion. In contrast, it is more intuitive to move a B-spline control point which changes the solution in a "hump" like fashion (see Figure 4). Thus the user in this case should manipulate the hierarchical B-spline functions.

4.2 Orientation

In the parametric representation, the curve or surface is represented by three functions X, Y, Z. In the the multi-resolution paradigm, when a user adds fine directional detail, say a fine hump in the X direction, this detail will become locked in the originally chosen direction. If the user later manipulates the broad sweep of the curve, the detail will maintain its original direction (see Figure 5). This is

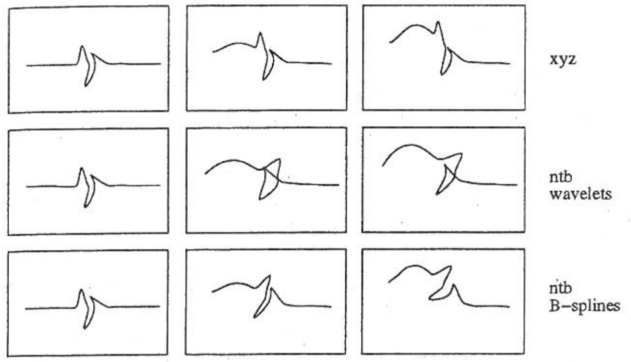

Figure 5: When the (X,Y,Z) frame is used for wavelet multiresolution editing, detail maintains its orientation as the sweep is changed. When the normal, tangent, bi-normal, (N, T, B) frame is used with a wavelet representation, the detail does not maintain its structure as the sweep is changed. When the (N, T, B) frame is used with a B-spline representation, the detail follows the orientation of the curve.

not always desirable, since the user may want the detail's orientation to follow the changing direction of broader curve or surface.

An "orientation" approach first proposed by Forsey and Bartels [9] may be applied to the multiresolution editing scheme. In a multiresolution modeling system all of the information describing the curve or surface lives at some resolution. In an orientation approach, the information at each resolution i is not expressed as three independent functions of (X, Y, Z). Instead the detail at each resolution i is represented with respect to the geometric shape of the lower resolution version of the curve or surface. This lower resolution version is defined by summing all of the information from all the lower resolution levels.

Tangent and normal directions of the lower resolution curve or surface are then computed at a series of sample points. The detail coefficients at level i are then expressed with respect to these tangent and normal directions instead of the (X, Y, Z) directions. If any lower resolution component of the curve is later explicitly altered, then the detail's orientation will change appropriately.

4.2.1 Defining Detail

In order to apply an orientation approach, one must have some method for decomposing the object into components at different resolutions. When one is using hierarchical B-splines, which over-represent objects in V_L, then there is some freedom in defining what information resides at which level of detail.

If the geometric object is being designed with a multiresolution editor, then the user is explicitly manipulating the object at resolutions that he chooses. Therefore, one simple method is to maintain all information at the resolution entered by the user [9]. Using this method, the same geometric object may behave differently depending on the way the object was generated.

An alternative is to use wavelet analysis: begin with the complete resolution object (in V_L), and then successively *project* it to each lower resolution level using Equation (15). This generates a unique smoothed version of the object at each resolution V_i. The object can now be represented as a combination of components from the difference spaces W_i.

In typical wavelet analysis, the components in W_i are represented using some special basis functions $\psi_{i,j}$ that span the difference space W_i. Alternatively, instead of using wavelet functions $\psi_{i,j}$ to represent the difference, one may instead use the B-spline functions on the next finer level $\phi_{i+1,j}$. This can be done because of Equation (12). The choice of whether to use B-spline or wavelets to represent the functions in W_i is an important question that we shall deal with soon.

4.2.2 Projections between Levels

There are many ways to obtain a lower resolution version of some object from V_L. For example, given an object in V_L, one could obtain a lower resolution version in V_{L-1} by throwing away every other control point. Subsampling is not a true projection; starting with a smooth curve in V_{L-1}, and then expressing that smooth curve in the higher resolution B-spline basis basis V_L, and finally subsampling the control points will not return the original smooth curve we began with.

Another way of obtaining a smoothed version of the object is by *orthogonally* projecting the object from V_L into V_{L-1}. The orthogonal projection is the object in V_{L-1} that is closest to object in V_L using the L^2 measure. One may obtain the orthogonal projection by using Equation (15), with the $\tilde{h}$ sequence given for the semi-orthogonal wavelet construction by Chui [3]. This is the approach used in [8]. Although this is a very elegant way of obtaining a lower resolution version of an object, it has a few drawbacks. This particular $\tilde{h}$ sequence is infinite in length (although it does decay rapidly from its centers) and so performing this task efficiently can be troublesome. Also, because these sequences are not local, then a single change to one B-spline coefficient at level L will alter all of the coefficients of the projection at level $L-1$.

One good compromise between these two extremes (subsampling, and orthogonal projection), is to use Equation (15) but to use the $\tilde{h}$ filter given for the non-orthogonal wavelet construction by Cohen et al. [5]. This projection in non-orthogonal, but it is entirely local. This is the choice we have used in our multiresolution modeling tool.

4.2.3 Representing Detail

What set of basis functions should be used to represent the detail. If a wavelet projection Equation (15) is used to define the lower resolution versions of the object, then the detail can be represented by using the corresponding wavelet functions. The other option is to represent the detail using hierarchical B-spline functions. The disadvantage of using hierarchical B-splines is that there are roughly $2n$ B-splines in the hierarchy, and only n wavelets.

The advantage of using hierarchical B-splines however is that they maintain the orientation better. When the user changes the broad sweep of the curve, changing the tangent, normal, and binormal frame at t_j, the detail functions are remixed. If the detail functions are wavelet functions, then changing the normal and tangent frame remixes "wave" shaped functions introducing non-intuitive wiggles. If the detail functions are B-spline basis functions, then "hump" shaped functions get remixed, yieding more intuitive changes. Also if the detail functions are B-splines, then because there are twice as many B-splines than wavelets, the tangent and normal directions are computed at twice as many sample points allowing the detail to follow the orientation with more fidelity (see Figure 5).

5 Variational Modeling

The variational modeling paradigm generalizes the least squares notion to any *objective* function minimization, typically one representing minimizing curvature. The variational problem leads to a non-linear optimization problem over a finite set of variables when cast into a given basis.

There are a variety of objective functions used in geometric modeling [21, 24] In our implementation we have used the *thin-plate* measure which is based on parametric second derivatives [27, 2, 28]. The thin plate minimum may be found by solving the following linear system [28].

$$\left|\begin{array}{cc} \mathbf{H} & \mathbf{A}^T \\ \mathbf{A} & 0 \end{array}\right| \left|\begin{array}{c} \mathbf{x} \\ \lambda \end{array}\right| = \left|\begin{array}{c} 0 \\ \mathbf{b} \end{array}\right| \tag{20}$$

Where $\mathbf{A}$ is the constraint matrix, $\mathbf{H}$ is the Hessian matrix, and λ are Lagrange variables.

5.1 Hierarchical Conditioning

Wavelets can be used in the context of variational modeling so that the solution may be obtained more efficiently.

In the B-spline basis, the optimization procedure resulted in the linear system given by Equation (20). In the wavelet basis, a different linear system results which is given by

$$\left|\begin{array}{cc} \bar{\mathbf{H}} & \bar{\mathbf{A}}^T \\ \bar{\mathbf{A}} & 0 \end{array}\right| \left|\begin{array}{c} \bar{\mathbf{x}} \\ \lambda \end{array}\right| = \left|\begin{array}{c} 0 \\ \mathbf{b} \end{array}\right| \tag{21}$$

where the bars signify that the variables are wavelet coefficients, $\bar{\mathbf{x}} = \mathbf{W}\mathbf{x}$, and the Hessian and constraint matrix are expressed with respect to the wavelet basis. To see the relationship with the B-spline system, the new system can also be written down as

$$\left|\begin{array}{cc} \mathbf{W}^{-T}\mathbf{H}\mathbf{W}^{-1} & \mathbf{W}^{-T}\mathbf{A}^T \\ \mathbf{A}\mathbf{W}^{-1} & 0 \end{array}\right| \left|\begin{array}{c} \bar{\mathbf{x}} \\ \lambda \end{array}\right| = \left|\begin{array}{c} 0 \\ \mathbf{b} \end{array}\right| \tag{22}$$

Although Equation (20) and Equation (21/22) imply each other, they are two distinct linear systems of equations. Because the wavelet system (21/22) is hierarchical it will not suffer from the poor conditioning of the B-spline system of Equation (20). For a rigorous discussion of the relevant theory see [7].

The scaling of the basis functions is very significant for the behavior of the optimizing procedures. Traditionally the wavelet functions are defined with the following scaling [19, 22]:

$$\begin{aligned} \phi_{i,j}(t) &= 2^{(i-L)/2}\,\phi(2^{(i-L)}t - j) \\ \psi_{i,j}(t) &= 2^{(i-L)/2}\,\psi(2^{(i-L)}t - j) \end{aligned} \tag{23}$$

This means that at each level moving up, the basis functions become twice as wide, and are scaled $\frac{1}{\sqrt{2}}$ times as tall. While in many contexts this normalizing may be desirable, for optimization purposes it is counter productive. For the optimization procedure to be well conditioned [15, 7] it is essential to emphasize the coarser levels. The correct theoretical scaling depends on both the energy function used, and the dimension of problem. For a fuller discussion, see the Appendix in [13]. In the experiments described in this paper the following scaling was used

$$\begin{aligned} \phi_{i,j}(t) &= 2^{-(i-L)}\,\phi(2^{(i-L)}t - j) \\ \psi_{i,j}(t) &= 2^{-(i-L)}\,\psi(2^{(i-L)}t - j) \end{aligned} \tag{24}$$

This means that as one goes from level i to level $i-1$ the basis functions become twice as wide, and $1/2$ as tall. In the pyramid code, this is achieved by multiplying all of the h and g entries by 2, and all of the $\tilde{h}$ and $\tilde{g}$ by $1/2$ [5].

[5]The proper scaling is essential to obtain the quick convergence of the wavelet method when steepest descent or conjugate gradient iteration is used. Scaling is not important with Gauss-Seidel iteration, which will perform the same sequence of iterations regardless of scale.

5.1.1 Explicit vs. Implicit

There is now a choice to make. In an iterative conjugate gradient solver, the common operation is multiplication of a vector times the wavelet matrix given in Equations (21/22). There are two ways to implement this.

One approach, the *explicit* approach, is to compute and store the wavelet Hessian matrix $\bar{\mathbf{H}}$ and the wavelet constraint matrix $\bar{\mathbf{A}}$ (Equation (21)). These can be computed directly from a closed form (piecewise polynomial) representation of the wavelet functions $\psi_{i,j}$. Unfortunately, these matrices are not as sparse as the B-spline Hessian and constraint matrices.

Alternatively, there is the *implicit* approach [29, 25] which only computes and stores the entries of the B-spline matrices $\mathbf{H}$ and $\mathbf{A}$ (Equation (22)). Multiplication by the $\mathbf{W}$ matrices is accomplished using the `pyrm` procedures. The advantage of this approach is that the whole multiply remains $O(n)$ in both time and space, since the `pyrm` procedures run in linear time, and the matrices $\mathbf{H}$ and $\mathbf{A}$ are $O(n)$ sparse. Even though one of the methods explicitly uses wavelet terms while the other uses B-spline terms, these two methods are mathematically equivalent, and so both will have the same condition properties.

5.2 Adaptive Oracle

By limiting the possible surfaces to only those that can be expressed as a linear combination of a fixed set of basis functions, one obtains an approximation of the true optimal surface. As more basis functions are added, the space of possible solutions becomes richer and a closer approximation to the true optimal surface can be made. Unfortunately, as the space becomes richer, the number of unknown coefficients increases, and thus the amount of computation required per iteration grows. A priori, it is unknown how many basis functions are needed. Thus, it is desirable to have a solution method that adaptively chooses the appropriate basis functions. This approach was applied using hierarchical B-splines in [28]. When refinement was necessary, "thinner" B-splines basis functions were added, and the redundant original "wider" B-splines were removed. With wavelets, all that must be done is to add in new "thinner" wavelets wherever refinement is deemed necessary. Since the wavelets coefficients correspond directly to local detail, all previously computed coefficients are still valid.

The decision process of what particular wavelets to add and remove is governed by an `oracle` procedure which is called after every fixed number of iterations. The oracle must decide what level of detail is required in each region of the curve or surface.

When some region of the solution does not need fine detail, the corresponding wavelet coefficients are near zero, and so the first thing the `oracle` does is to deactivate the wavelet basis functions whose corresponding coefficients are below some small threshold. The `oracle` then activates new wavelet basis functions where it feels more detail may be needed. There are two criteria used. If a constraint is not being met, then the oracle adds in finer wavelet functions in the region that is closest in parameter space to the unmet constraint. Even if all the constraints are being met, it is possible that more basis functions would allow the freedom to find a solution with lower energy. This is accomplished by activating finer basis functions near those with coefficients above some maximum threshold.

To avoid cycles, a basis function is marked as being `dormant` when it is removed from consideration. Of course, it is possible that later on the solution may really need this basis function, and so periodically there is a `revival` phase, where the `dormant` marks are removed.

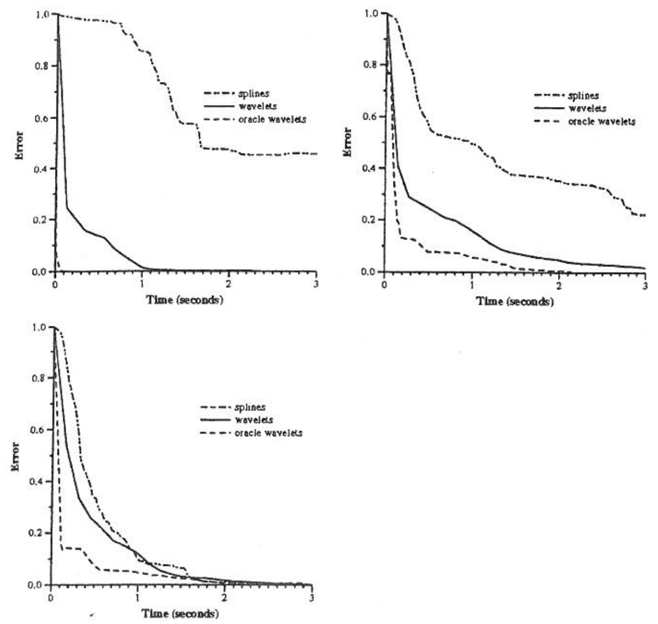

Figure 6: Error per time. Curve with 65 control points, 3, 7, and 13 constraints.

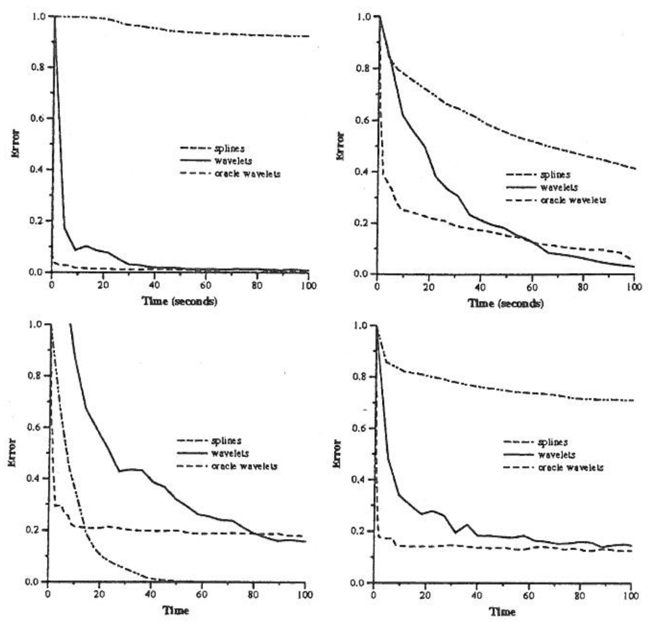

Figure 7: Error per time. Surface with 1089 control points, 11,23,64 evenly space constraints, and 62 constraints along the boundary.

5.3 User Interface

A user of the system is first presented with a default curve or surface. Constraints can then be introduced by clicking on the curve or surface with the mouse. The location of the mouse click defines a parametric position t (and s) on the curve (or surface). The user can then drag this point to a new location to define an interpolation constraint. Tangent constraints at a point can also be defined by orienting "arrow" icons at the point. Once the constraint is set, the solver is called to compute the minimum energy solution that satisfies the constraints placed so far. Resulting curves and surfaces are displayed using SGI GL **nurbscurve** and **nurbssurface** calls [6].

When the solution is completed, the result provides information for not only the curve or surface satisfying the specific value of the new constraint, but for all curves or surfaces with respect to any value of this constraint. Once the linear system (Equation (21/22)) with the newest constraint has been solved, the solver stores the delta vector

$$\frac{\Delta \bar{\mathbf{x}}}{\Delta \mathbf{b_m}} \tag{25}$$

where m is the index of the newest constraint, and b_m is the constraint value (i.e., the position or tangent specified by the user). This vector stores the change of the coefficient vector due to a unit change in the new constraint $\Delta \mathbf{b_m}$, essentially a column of the inverse matrix. The user is now free to interactively move the target location of the constraint without having to resolve the system since, as long as the parameters s, and t of the constraints do not change, the matrix of the system, and thus its inverse, do not change. However, as soon as a new constraint is added (or a change to the parameters s and t is made) there is fresh linear system that must be solved, and all of the delta vectors are invalidated. The ability to interactively change the value of a constraint is indicated to the user by coloring the constraint icon. See Color Plate.

5.4 Variational Modeling Results

[6]One GL call to **nurbssurface** can be more expensive than a complete iteration.

A series of experiments were conducted to examine the performance of the wavelet based system compared to a B-spline basis. In the curve experiments, the number of levels of the hierarchy, L, was fixed to 6, and in the surface experiments, L was fixed as 5. The optimization process was then run on problems with different numbers of constraints. The results of these tests are shown in Figures 6 and 7. These graphs show the convergence behavior of three different methods, solving with the complete B-spline basis, solving with the complete wavelet basis, and solving with an adaptive wavelet basis that uses an oracle. (The wavelet results shown here are using the *implicit* implementation). If $\mathbf{x}^{(\mathbf{m})}$ is the computed solution expressed as B-spline coefficients at time m, and $\mathbf{x}^*$ is the correct solution of the complete linear system [7] (i.e., the complete system with $2^L + 1$ variables, and no adaptive oracle being used) then the error at time m is defined as

$$\frac{\sum_j \mid x_j^* - x_j^{(m)} \mid}{\sum_j \mid x_j^* - x_j^{(0)} \mid} \tag{26}$$

To obtain the starting condition $\mathbf{x}^{(\mathbf{0})}$, two constraints were initialized at the ends of the curve, and the minimal thin plate solution (which in this case is a straight line) was computed. (For surfaces, the four corners were constrained.) All times were taken from runs on an SGI R4000 reality engine. [8]

When the are a large gaps between the constraints, the B-spline method is very poorly conditioned, and converges quite slowly while the wavelet method converges dramatically faster. In these problems, the oracle decides that it needs only a very small active set of wavelets and so the adaptive method converges even faster. As the number of constraints is increased, the solution becomes more tightly constrained, and the condition of the B-spline system improves. (Just by satisfying the constraints, the B-spline solution is very close to minimal energy). Meanwhile the oracle requires a

[7]computed numerically to high accuracy

[8]In the curve experiments, each B-spline iteration took 0.0035 seconds, while each iteration of the implicit wavelet method took 0.011 seconds. For the surface experiments, each B-spline iteration took 0.68 seconds while each iteration of the implicit wavelet method took 0.85 seconds. (The wavelet iterations using the explicit representation took about 10 times as long).

larger active set of wavelets. Eventually, when enough constraints are present, the wavelet methods no longer offer an advantage over B-splines.

Experiments were also run where all the constraints were along the boundary of the surface. In these experiments there are many constraints, but the since the constraints are along the boundary, much of the surface is "distant" from any constraint. In these problems, the wavelets also performed much better than the B-spline method.

6 Conclusion

This paper has explored the use of wavelet analysis in a variety of modeling settings. It has shown how wavelets can be used to obtain multiresolution control point and least squares control. It has shown how wavelets can be used to solve variational problems more efficiently.

Future work will be required to explore the use of higher order functionals like those given in [21, 24]. Because the optimization problems resulting from those functionals are non-linear, they are much more computationally expensive, and it is even more important to find efficient methods. It is also important to study optimization modeling methods where constraint changes only have local effects.

Many of these concepts can be extended beyond the realm of tensor product uniform B-splines. Just as one can create a ladder of nested function spaces V_i satisfying the property of Equation (10) using uniform cubic B-splines of various resolutions, one can also create a nested ladder using non-uniform B-splines [18].

Subdivision surfaces are a powerful technique for describing surfaces with arbitrary topology [14]. A subdivision surface is defined by iteratively refining an input control mesh. As explained by Lounsbery et al. [17], one can develop a wavelet decomposition of such surfaces. Thus, many of the ideas developed in this paper may be applicable to that representation as well.

Acknowledgements

We are grateful to James H. Shaw for developing the graphical interface to the optimization program.

REFERENCES

[1] BARTELS, R., AND BEATTY, J. A Technique for the Direct Manipulation of Spline Curves. In *Graphics Interface 1989* (1989), pp. 33–39.

[2] CELNIKER, G., AND GOSSARD, D. Deformable Curve and Surface Finite-Elements for Free-From Shape Design. *Computer Graphics 25*, 4 (July 1991), 257–266.

[3] CHUI, C. K. *An Introduction to Wavelets*, vol. 1 of *Wavelet Analysis and its Applications*. Academic Press Inc., 1992.

[4] CHUI, C. K., AND QUAK, E. Wavelets on a Bounded Interval. *Numerical Methods of Approximation Theory 9* (1992), 53–75.

[5] COHEN, A., DAUBECHIES, I., AND FEAUVEAU, J. C. Biorthogonal Bases of Compactly Supported Wavelets. *Communication on Pure and Applied Mathematics 45* (1992), 485–560.

[6] COHEN, E., LYCHE, T., AND RIESENFELD, R. Discrete B-Splines and Subdivision Techniques in Computer-Aided Geometric Design and Computer Graphics. *Computer Graphics and Image Processing 14*, 2 (October 1980), 87–111.

[7] DAHMEN, W., AND KUNOTH, A. Multilevel Preconditioning. *Numerische Mathematik 63* (1992), 315–344.

[8] FINKELSTEIN, A., AND SALESIN, D. Multiresolution Curves. In *Computer Graphics, Annual Conference Series, 1994* (1994), Siggraph, pp. 261–268.

[9] FORSEY, D., AND BARTELS, R. Hierarchical B-Spline Refinement. *Computer Graphics 22*, 4 (August 1988), 205–212.

[10] FORSEY, D., AND WENG, L. Multi-resolution Surface Approximation for Animation. In *Graphics Interface* (1993).

[11] FOWLER, B. Geometric Manipulation of Tensor Product Surfaces. In *Proceedings, Symposium on Interactive 3D Graphics* (1992), pp. 101–108.

[12] GORTLER, S., SCHRÖDER, P., COHEN, M., AND HANRAHAN, P. Wavelet Radiosity. In *Computer Graphics, Annual Conference Series, 1993* (1993), Siggraph, pp. 221–230.

[13] GORTLER, S. J. *Wavelet Methods for Computer Graphics*. PhD thesis, Princeton University, January 1995.

[14] HALSTEAD, M., KASS, M., AND DEROSE, T. Efficient, Fair Interpolation using Catmull-Clark Surfaces. In *Computer Graphics, Annual Conference Series, 1993* (1993), Siggraph, pp. 35–43.

[15] JAFFARD, S., AND LAURENÇOT, P. Orthonormal Wavelets, Analysis of Operators, and Applications to Numerical Analysis. In *Wavelets: A Tutorial in Theory and Applications*, C. K. Chui, Ed. Academic Press, 1992, pp. 543–602.

[16] LIU, Z., GORTLER, S. J., AND COHEN, M. F. Hierarchical Spacetime Control. In *Computer Graphics, Annual Conference Series, 1994* (August 1994), pp. 35–42.

[17] LOUNSBERY, M., DEROSE, T., AND WARREN, J. Multiresolution Analysis for Surfaces of Arbitrary Topological Type. Tech. Rep. TR 93-10-05b, Department of Computer Science and Engineering, Princeton University, October 1993.

[18] LYCHE, T., AND MORKEN, K. Spline Wavelets of Minimal Support. In *Numerical Methods in Approximation Theory*, D.Braess and L. L. Schumaker, Eds., vol. 9. Birkhauser Verlag, Basel, 1992, pp. 177–194.

[19] MALLAT, S. G. A Theory for Multiresolution Signal Decomposition: The Wavelet Representation. *IEEE PAMI 11* (July 1989), 674–693.

[20] MEINGUET, J. Multivariate Interpolation at Arbitrary Points Made Simple. *Journal of Applied Mathematics and Physics (ZAMP) 30* (1979), 292–304.

[21] MORETON, H., AND SEQUIN, C. Functional Optimization for Fair Surface Design. *Computer Graphics 26*, 4 (July 1992), 167–176.

[22] PENTLAND, A. Fast Solutions to Physical Equilibrium and Interpolation Problems. *The Visual Computer 8*, 5 (1992), 303–314.

[23] QIAN, S., AND WEISS, J. Wavelets and the Numerical Solution of Partial Differential Equations. *Journal of Computational Physics 106*, 1 (May 1993), 155–175.

[24] RANDO, T., AND ROULIER, J. Designing Faired Parametric Surfaces. *Computer Aided Design 23*, 7 (September 1991), 492–497.

[25] SZELISKI, R. Fast Surface Interpolation Using Hierarchical Basis Functions. *IEEE PAMI 12*, 6 (June 1990), 513–439.

[26] TERZOPOULOS, D. Image Analysis Using Multigrid Relaxation Methods. *IEEE PAMI 8*, 2 (March 1986), 129–139.

[27] TERZOPOULOS, D. Regularization of Inverse Visual Problems Involving Discontinuities. *IEEE PAMI 8*, 4 (July 1986), 413–424.

[28] WELCH, W., AND WITKIN, A. Variational Surface Modeling. *Computer Graphics 26*, 2 (July 1992), 157–166.

[29] YSERENTANT, H. On the Multi-level Splitting of Finite Element Spaces. *Numerische Mathematik 49* (1986), 379–412.

Interactive Shape Metamorphosis

David T. Chen*, Andrei State* and David Banks‡

*Department of Computer Science
University of North Carolina at Chapel Hill

‡Institute for Computer Applications in Science and Engineering

1 INTRODUCTION

Image metamorphosis (morphing) is a powerful and easy-to-use tool for generating new 2D images from existing 2D images. In recent years morphing has become popular as an artistic tool and is used extensively in the entertainment industry. In this paper we describe a new technique for controlled, feature-based metamorphosis of certain types of surfaces in 3-space; it applies well-understood 2D methods to produce shape metamorphosis between 3D models in a 2D parametric space. We also describe an interactive implementation on a parallel graphics multicomputer, which allows the user to define, modify and examine the 3D morphing process in real time.

2 PREVIOUS WORK

Wolberg [4] described a point correspondence technique for morphing 2D images. Consider a pair of 2D source images, A and B. If a feature in image A is meant to match a feature in image B, the user chooses a point within the feature of each image. When the point morphs from A to B, so does a neighborhood surrounding it. By defining such pairs of points for all interesting features, the user can create a metamorphosis sequence between the two static images A and B.

Beier and Neely [1] described a segment correspondence technique for morphing 2D images. When a feature in image A is required to transform to a feature in image B, a line segment is drawn over the feature in each image. As the segment morphs from A to B, so does a neighborhood surrounding it. By judiciously creating line segments, the user can preserve all the important features throughout the morph. This technique is easier to use than the point correspondence method; usually fewer than half as many line segment pairs than point pairs are required to define a morph sequence between two static images.

2D methods provide simple user control for image-based morphing. However, since little or no information about actual 3D geometry is available, it is difficult to create "natural"-looking transformations; morphing animations created with 2D methods often exhibit a subtle "flattening" effect.

Kent, Carlson and Parent [3] described a method for morphing 3D polyhedral objects by merging the topologies of two 3D source polyhedra A and B. New vertices, edges, and faces are added to both A and B so that every polygon of A corresponds to a polygon of B. To morph between them one interpolates between corresponding vertices. The user can exercise some control over how the correspondences are established, but only very indirectly, by selecting a specific method of mapping the two source objects onto a common intermediate mapping surface used for topology merging (for example, a sphere). Kent concludes:

> ... techniques that provide a finer level of control over the transformation are needed. One possibility is to add a warping step ... before the topologies are merged.

We implemented Beier's technique as that warping step for the special case of cylindrical mapping surfaces, warping the model's 2D parameter space instead of a (projected) 2D image.

3 CONTROLLED 2D-3D MORPHING

Our method consists of morphing the common intermediate mapping surface or 2D parameter space of a pair of surface models. We use Beier's techniques to establish correspondences and accomplish the warping. The 2D nature of the process makes interaction easy. While defining correspondences, the user can simultaneously inspect the two parametric images as well as the resulting surface in 3-space .

We begin with a pair of surface models A and B (Figure 1) which have been meshed over some parameter space. Models in other formats (like polygon-lists, NURBS, or implicit surfaces) must be resampled and meshed so that they have similar parameter spaces. This may seem like a relatively harsh restriction,

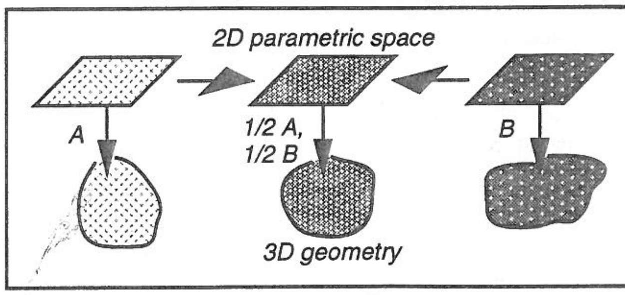

Figure 1. Object A is morphed into Object B. The objects are parametric surfaces. To interpolate between the geometries, interpolate between the 2D parameter spaces.

*Department of Computer Science, University of North Carolina at Chapel Hill, Chapel Hill, North Carolina, 27599-3175. (chen|state)@cs.unc.edu.
‡Institute for Computer Applications in Science and Engineering, Mail Stop 132C, NASA Langley Research Center, Hampton, Virginia, 23681. banks@icase.edu.

making the technique applicable only to convex or star-shaped objects. However, there are physically-based and model-specific projection techniques [3] that can be applied to more complex geometries.

All the surface attributes of the source models must be available in the 2D parameter space so that they may be interpolated. There are map-parameters attached to each sample as well. For example, in the case of a spherically-projected apple, the map-parameter is the radius at each sample point. Knowing the radius, one can reconstruct the surface of the original apple and attach the sample's surface attributes to it.

The surface attributes are interpolated as well as the samples' map-parameters. The interpolated map-parameters serve to construct the morphed target model from a morphed image in the 2D parametric space. The 3D target model is derived from this image by applying the mappings; in doing so, we use the "morphed" values of the map-parameters at each sample point to construct the surface of the target.

4 INTERACTIVE IMPLEMENTATION

We have implemented a prototype system on the Pixel-Planes 5 graphics multicomputer, a heterogeneous system consisting of over 30 Intel i860-based MIMD nodes and a massively parallel array of SIMD pixel processors [2]. We chose Beier's technique for its easy and intuitive control methods. We demonstrate our method on 3D models of human heads generated by a 3D scanner (Cyberware™). These models are represented in cylindrical coordinates (with the mantle of the cylinder serving as the 2D parameter space for the morphing process). Our samples contain the surface attribute color and the map-parameter radius. Traditional morphing between 2D images operates on color as a function of 2D pixel coordinates; here we operate on color and radius as functions of the 2D parametric coordinates angle and height.

The software design of the system is straightforward: the entire 2D parameter space of each of the two source models with surface attributes and map-parameters is replicated on all MIMD nodes. Each node generates a subset of the morphed parametric image. The nodes then apply the morphed colors and map-parameters to generate colored polygons from the morphed parametric image (Plate 1).

Plate 2a shows a pair of 2D parametric images on which a user has marked features. The background images show the color intensity of the models in the parameter space of cylindrical coordinates. Plate 2b shows the radii (essentially height functions) in cylindrical coordinates, mapped to gray intensity values. Note the pairs of line segments: they establish correspondences between various features of the two source models in the Beier-Neely technique. These features may be chosen simply by their similar color (like matching the red regions of lips in a 2D image), but they may also be chosen by their similar 3D geometry (like matching the pointed tip of each nose). This latter ability is crucial for matching features in regions of constant color. These regions are prominent in profile, but not in the general projected views. It would be inefficient to search for corresponding features by continually rotating the objects until their features are identifiable by their colors alone.

Plate 3 shows a sequence of shape metamorphosis images generated by our system. Mapped onto the surfaces of the 3D models, the line segments become surface-following curves. The face rotates as it is morphed to demonstrate how the geometric features are preserved during the interpolation. Notice, for example, how the lips spread open as the morphing progresses. Notice also that one of the eyes is obscured in the left image. Pure image-based morphing cannot interpolate between features when one of them is obscured under a particular viewing projection.

The entire process of matching features and warping between the surfaces in Plate 3 takes only a few minutes for a trained user. The 274-by-222 surface mesh with 33 pairs of line segments for correspondence definition is morphed and rendered on Pixel-Planes 5 at 20 frames per second (4-by-4 decimation) or at 1 frame per second (full resolution).

5 FUTURE WORK

We are currently working on a true 3D interface for our system. This will allow the user to specify correspondence areas directly on the 3D source objects, while continuing to use 2D parametric space morphing techniques. In the future we plan to add support for other types of parametric spaces besides cylindrical projections. Then our system could allow controlled shape metamorphosis for many of the classes of 3D objects described in [3].

6 CONCLUSIONS

We have described how to apply image-based metamorphosis to parametrically defined 3D surfaces or arbitrary surfaces that can be expressed parametrically using projection techniques described in [3]. For such surfaces our method is superior to ordinary image-based warping: the warp is defined only once (rather than frame-by-frame) for an entire animation and can be accomplished in a short interactive session. Since our method provides local correspondence definition, it is superior to previous techniques that automatically map between surfaces in a global manner. The technique is easily parallelizable; on our prototype system the interpolating surfaces can be constructed and displayed at interactive frame rates.

Finally, the animation sequences produced with our method do not exhibit the "flattening" effect typical for image-based morphing. Our sequence has an intuitively three-dimensional "look," noticeable not only in high-resolution animations, but also at lower resolutions, during interactive operation.

We would like to acknowledge the contributions of Andrew Brandt, Anselmo Lastra, Greg Turk and David Addleman.

REFERENCES

1. Beier, Thaddeus, and Shawn Neely. "Feature-Based Image Metamorphosis." Proceedings of SIGGRAPH '92 (Chicago, Illinois, July 26-31, 1992). In Computer Graphics 26, 2 (July 1992), pp 35-42.

2. Fuchs, Henry et. al. "Pixel-Planes 5: A Heterogeneous Multiprocessor Graphics System Using Processor Enhanced Memories." Proceedings of SIGGRAPH '91 (Las Vegas, Nevada, July 29-August 2, 1991). In Computer Graphics 25, 4 (July 1991), pp 79-88.

3. Kent, James R., Wayne E. Carlson, and Richard E. Parent. "Shape Transformation for Polyhedral Objects." Proceedings of SIGGRAPH '92 (Chicago, Illinois, July 26-31, 1992). In Computer Graphics 26, 2 (July 1992), pp 47-54.

4. Wolberg, G. Digital Image Warping. IEEE Computer Society Press, 1990.

Shadow Volume BSP Trees for Computation of Shadows in Dynamic Scenes

Yiorgos Chrysanthou* and Mel Slater*

Queen Mary and Westfield College,
University of London.

ABSTRACT

This paper presents an algorithm for shadow calculation in dynamic polyhedral scenes illuminated by point light sources. It is based on a modification of Shadow Volume Binary Space Partition trees, to allow these be constructed from the original scene polygons in arbitrary order and to support for fast reconstruction after a change in scene geometry. Timings using sample scenes are presented that indicate substantial savings both in terms of computation time and shadows produced.

KEY WORDS: shadows, BSP Trees, SVBSP Trees, dynamic modification.

1 INTRODUCTION

An algorithm is presented for rapid updating of shadows in dynamic environments, where objects are transformed in near real time with induced changes to shadows computed and displayed. The algorithm employs shadow volumes (SV). This was a term used by Crow [6] to denote the semi-infinite volume enclosed by the shadow planes (SP) formed by the triangle of the edge vertices and the light source position, for each edge of a polygon and culled by the polygon plane. Reviews of shadow algorithms for static scenes may be found in [1, 19, 15]. An algorithm for shadows in dynamic scenes is described in [5]. This uses a Shadow Tiling and a Binary Space Partition (BSP) tree. BSP trees were developed by Fuchs, Kedem and Naylor [8] as a visible surface determination, based on Schumacker's results [14]. A BSP tree is a hierarchical subdivision of space into homogeneous regions, using the planes defined by the scene polygons as partitions. It is mainly suitable for static scenes but under the right circumstances it can deal with moving objects [7, 17, 5]. Thibault, Naylor, and Amanatides [11, 16] employed the BSP tree to represent arbitrary polyhedral solids and for set operations on polyhedra in representation and rendering for CSG. They also showed that a BSP tree can be constructed incrementally.

Based on these results Chin and Feiner [3] introduced the Shadow Volume Binary Space Partition (SVBSP) tree algorithm for point light sources that can be used to compute shadows efficiently for polyhedral scenes. The algorithm used a BSP tree to order the input polygons in increasing order of depth from the light source, and to incrementally build a BSP tree representation of a single merged shadow volume for the whole scene. In the process of building the tree, the scene polygons are split and labeled as *lit* or *shadowed*. The algorithm operates in object space so that the shadows need not be regenerated if the viewing parameters are changed. The method is therefore suitable for walk-through applications. However, it is not suitable for interactive modification of objects in the scene, since any change in an object's position could destroy the ordering and may require the reconstruction of the shadow tree. Similar structures were used in [10] for image representation and in [4, 2] for determining illumination discontinuities from area light sources.

The algorithm presented here employs a generalization of the SVBSP tree that does not require the construction of a BSP tree of the original scene polygons, and that does support near real-time incremental changes to the SVBSP tree and therefore to shadows in response to object transformations. This method also results in computation of a smaller number of shadows compared to the original method. An application of the algorithm on a VR system is described in [18].

2 BUILDING THE UNORDERED SVBSP TREE

The standard SVBSP tree is built from an ordered set of polygons so there is no question as to which polygon is closer to the light source when the SVs of two polygons intersect. Building the tree using only the shadow planes is sufficient. For the unordered SVBSP tree the scene polygons themselves must be added to convey that information, so the SV of each individual polygon is complemented with the polygon plane. Since nodes containing shadow planes

*Department of Computer Science, QMW University of London, Mile End Road, London E1 4NS, UK. e-mail: yiorgos@dcs.qmw.ac.uk, mel@dcs.qmw.ac.uk

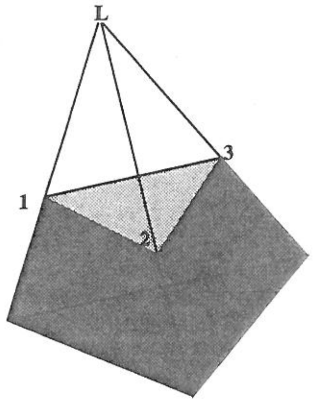

Figure 1: Full shadow volume

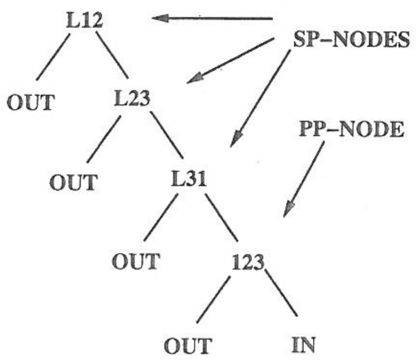

Figure 2: Shadow volume as a BSP tree

and nodes containing polygon planes are treated differently by the algorithm they will be distinguished by calling the former SP-Nodes (Shadow Plane Nodes) and the latter PP-Nodes (Polygon Plane Nodes), Figure 2 .

The algorithm uses a copy of the scene polygons in the tree for calculating the shadows which are then stored as detail ontop of the actual scene polygons.

The tree is built incrementally by inserting the light-facing polygons into an initially empty tree, a single OUT node (Figure 3). The first polygon just replaces that node with its SV (Figure 4). Subsequent scene polygons are filtered into the tree by comparing them at each level against the plane at the root of the tree and recursively inserting them into the appropriate subtree. If they straddle the root plane then they are split and each piece is treated separately. When an OUT node is reached it is replaced by the SV. If the polygon was split its SV is built using the shadow planes of the original polygon (polygon 4 in Figure 6). This is necessary for dynamic modification and it also means that the SV needs to be calculated only once even if a polygon is split into many pieces.

A face onto which a shadow is cast is referred to as a *target* face. When a PP-Node is encountered, if the inserted polygon is classified as behind its plane then it is marked as shadowed and stored there (face 3 in Figure 6). If it is classified as in front then it takes note of the face at the root as a potential target and it is inserted into the front subtree. If it reaches an OUT node then a shadow is cast on the face stored as potential target (face 2 in Figure 6). If it comes in front of more than one potential target, only the last one is remembered and used (polygon 5 in Figure 7 comes in front of 2 and then in front of 4, a shadow is casted only on 4).

To cast a shadow onto a target, the original scene polygon of the target is clipped against the relevant SV.

3 USING THE TREE FOR DYNAMIC SHADOW COMPUTATION

In an interactive application where the scene geometry changes, the tree can be used to maintain the correct shadows.

During the building of the tree, each inserted polygon constructs a list of pointers to the locations it occupies on the tree. When an object is transformed, its polygons and their shadow planes on the tree are found using the location lists and are marked. After all relevant polygons have been marked, a recursive function is called that will iterate through the SVBSP tree once and remove all marked nodes. The result of this will be a valid SVBSP tree for the scene, but now without the transformed object. The object can then be reinserted into the tree using the algorithm described in section 2, to get the shadows at its new position.

3.1 REMOVING THE MARKED NODES

The function used for removing the marked nodes works on the whole SV of polygons rather than on single nodes. There are 3 possible positions for each polygon and its shadow volume to consider:

(a) In the IN region, behind a PP-node (no shadow planes were attached here, just the polygon). This is the simplest case, the polygon is just removed (polygon 3 in Figure 7).

(b) At the leaves, subdividing an empty subspace. Again this is simple, the SV is replaced by an OUT node. Care must be taken if the PP-Node had a non-null target. This occurs when it is in front of some other PP-Node during insertion and it is now casting a shadow on this. In this case the shadow must be removed. For example when deleting polygon 5 in Figure 7 , the front (left) subtree of node labeled 4.2 should be replaced by OUT and the shadow on polygon 4 should be deleted (the arrows there show the target relation).

(c) Splitting a non-empty subspace, the SV forms the root of a larger subtree. This is the only relatively complex case. Removing it would result in unconnected subtrees and these must be put together to form a new tree to replace the old one. If the deleted polygon was casting a shadow then that must be replaced by shadows from polygons that had the deleted one as target. These can only be in the front subtree of the deleted PP-node. For example if polygon 4 in Figure 7 is deleted then the shadow from 4 to to 2 should be replaced by a shadow from 5. Any polygons that were in shadow, in the IN region behind the deleted polygon, must also be inserted into the new unified subtree.

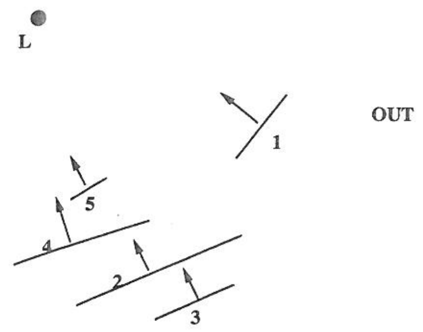

Figure 3: Initial scene

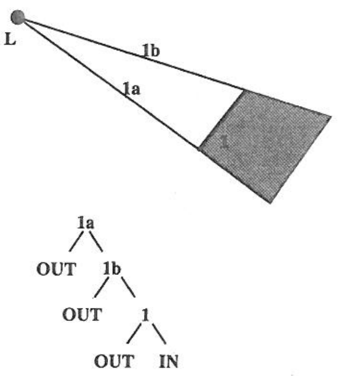

Figure 4: Insert poly 1

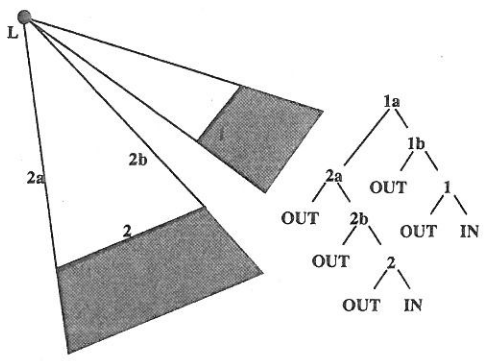

Figure 5: Insert poly 2

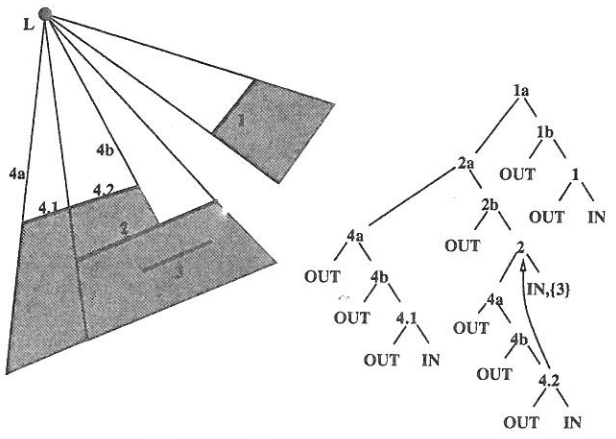

Figure 6: Insert poly 3 and 4

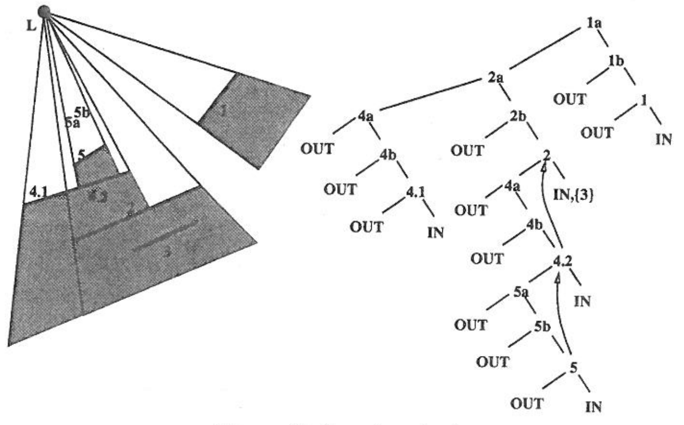

Figure 7: Insert poly 5

3.2 JOINING THE SUBTREES

Naylor in [11] described an algorithm for merging BSP trees that could be used in case (c) above. Given two BSP trees for merging, tree1 is inserted into tree2. To achieve this tree2 is split into tree2.front and tree2.back by filtering the root of tree1 into it. These two new trees are then recursively merged into the corresponding subtrees of tree1 until they reach the leaves. Experimental results have shown this method to be very slow for our purposes. The main reason is that it operates on a closed subspace and it would require the shadow planes involved in the merging to be clipped and bounded. Also it is very general, it doesn't utilise the fact that all the shadow planes emanate from the same point (the light source).

A more specialised algorithm is used here. The largest of the trees to be merged is found, say tree1, and any possible marked nodes on this are removed. The inserted tree, tree2, is then treated as a set of shadow volumes. The polygon node (PP-Node) of the shadow volume forming the root of tree2 is found and filtered down tree1 along with its front and back subtrees. The filtering is done in a similar manner to a polygon. The fact that all shadow planes go through the light source position, ensures that anything enclosed by a polygon's shadow volume can be split by another shadow plane, only if the polygon itself is split. This means that the front and back subtrees need to be checked for intersection with a plane only if the polygon is split by that plane. If the PP-Node meets another fragment of its own original polygon they can join up under its shadow volume (this is possible since the shadow planes used by the fragments are those of the original). When it reaches an OUT node its SV is attached. After the 'root' SV and the subtrees of its PP-Node have been inserted, the algorithm is called recursively to insert the front subtrees of its SP-Nodes.

Note that the subtrees involved here existed in non-intersecting subspaces separated by the deleted planes so there is no shadow relation between them. Also, if polygons split or come together during the merging, the shadows on them or the shadows they cast do not change.

4 FURTHER DISCUSSION

When a target object is being continuously transformed, for example as a result of being dragged during an interactive application, the functions described in sections 3.1 and 3.2 are only relevant for the very first transformation. After the first deletion and re-insertion, the faces will end up at the leaves and in subsequent frames can be deleted in constant time.

In the standard SVBSP tree the smaller objects tend to be higher up the tree because they tend to be closer the light source. This is the order that is obtained from the scene BSP tree traversed from the light position. Also their polygons may be widely distributed in the tree (Figure 10). Moreover