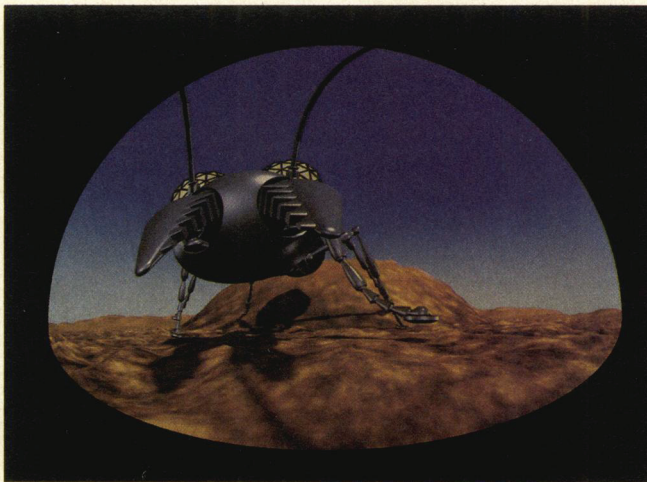# Creating Raster Omnimax Images from Multiple Perspective Views Using the Elliptical Weighted Average Filter

Ned Greene and Paul S. Heckbert*
New York Institute of Technology

© NYIT, 1984

Creation of Omnimax animation by computer opens up fantastic new visual possibilities. Unfortunately, the fish-eye distortion of Omnimax film images complicates synthesis by computer, since most image-synthesis programs can create only perspective views.

As an alternative to modifying existing image-synthesis programs to produce Omnimax projections directly, we present a method for creating them from multiple perspective views. Four perspective views of the environment are created, each a projection onto a face of a cube centered at the camera, and then a mapping program creates an Omnimax projection from them. To minimize aliasing during resampling, the mapping program uses the *elliptical weighted average filter*, a space-variant filter we developed for this application that computes a weighted average over an arbitrarily oriented elliptical area. This filter can also be used for texture mapping 3D surfaces.

The methods described in this article were used to make two animation sequences for *The Magic Egg*, a compilation of computer-generated Omnimax animation sponsored by SIGGRAPH 84.

Rendering programs usually perform a perspective transformation which simulates a pinhole camera and transforms straight lines to straight lines and planes to planes. However, the Omnimax projection of a 3D environment is a 180° fish-eye projection that does not preserve linearity or planarity, so rendering programs must be modified to perform an Omnimax projection rather than a perspective projection.

Rendering programs work in a variety of ways and handle a variety of geometric primitives. Because they are independent from the perspective projection, ray tracers are easily converted for Omnimax[1] projection, but their high computational expense makes them impractical for many applications.

On the other hand, standard polygon renderers rely heavily on the perspective projection to simplify and speed the scan conversion process. Among other things, they take advantage of the fact that the perspective projection preserves the convexity of polygons and the linearity of their edges, properties not preserved by the Omnimax projection. In short, standard polygon renderers are much more difficult to convert for Omnimax projection than ray tracers and conversion reduces their computational efficiency.

The problem of software conversion should be examined in the context of whole image generation systems. Some systems generate the final frames of animation using a single program, while other systems create images piece by piece using a variety of programs. We use the latter approach at NYIT: One program renders polygons, another renders quadric surfaces, and dozens of other programs process rendered images to produce final frames[2].

To convert all the programs in this graphics system to

Creating Raster Omnimax Images from Multiple Perspective Views Using the Elliptical Weighted Average Filter

Ned Greene and Paul S. Heckbert*

New York Institute of Technology

© NYIT, 1984

Creation of Omnimax animation by computer opens up fantastic new visual possibilities. Unfortunately, the fish-eye distortion of Omnimax film images complicates synthesis by computer, since most image-synthesis programs can create only perspective views.

As an alternative to modifying existing image-synthesis programs to produce Omnimax projections directly, we present a method for creating them from multiple perspective views. Four perspective views of the environment are created, each a projection onto a face of a cube centered at the camera, and then a mapping program creates an Omnimax projection from them. To minimize aliasing during resampling, the mapping program uses the *elliptical weighted average filter*, a space-variant filter we developed for this application that computes a weighted average over an arbitrarily oriented elliptical area. This filter can also be used for texture mapping 3D surfaces.

The methods described in this article were used to make two animation sequences for *The Magic Egg*, a compilation of computer-generated Omnimax animation sponsored by SIGGRAPH 84.

Rendering programs usually perform a perspective transformation which simulates a pinhole camera and transforms straight lines to straight lines and planes to planes. However, the Omnimax projection of a 3D environment is a 180° fish-eye projection that does not preserve linearity or planarity, so rendering programs must be modified to perform an Omnimax projection rather than a perspective projection.

Rendering programs work in a variety of ways and handle a variety of geometric primitives. Because they are independent from the perspective projection, ray tracers are easily converted for Omnimax[1] projection, but their high computational expense makes them impractical for many applications.

On the other hand, standard polygon renderers rely heavily on the perspective projection to simplify and speed the scan conversion process. Among other things, they take advantage of the fact that the perspective projection preserves the convexity of polygons and the linearity of their edges, properties not preserved by the Omnimax projection. In short, standard polygon renderers are much more difficult to convert for Omnimax projection than ray tracers and conversion reduces their computational efficiency.

The problem of software conversion should be examined in the context of whole image generation systems. Some systems generate the final frames of animation using a single program, while other systems create images piece by piece using a variety of programs. We use the latter approach at NYIT: One program renders polygons, another renders quadric surfaces, and dozens of other programs process rendered images to produce final frames.[2]

To convert all the programs in this graphics system to

0272-1716/86/0600-0021$01.00 © 1986 IEEE

VALEO EX. 1028_002

Figure 1. An Omnimax film frame.

Figure 2. A three-dimensional scene projected onto four faces of a cube.
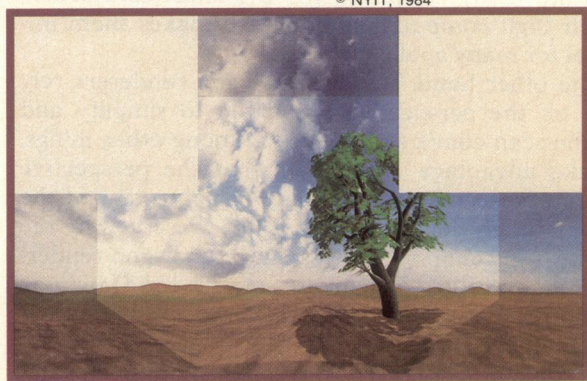
© NYIT, 1984

© NYIT, 1984

Figure 3. The unfolded cube of texture. The brighter area is the Omnimax field of view. The left half of the left panel, the right half of the right panel, and the top half of the top panel lie outside the field of view, so these regions need not be rendered.

handle Omnimax projection would be prohibitively time consuming, would complicate existing software, and would make maintenance more difficult. For these reasons we sought a method for creating Omnimax projections from perspective views.

We were interested in general techniques for creating Omnimax images that would allow high resolution, high scene complexity, and a variety of geometric modeling primitives. We were willing to sacrifice speed for generality and quality. Others have developed more efficient techniques for specific applications. Max has generated Omnimax animation of molecules using a local linear approximation to the distortion,[3] and methods for real-time Omnimax distortion of low-resolution images have also been developed.[4]

The Omnimax projection

Max has described the Omnimax projection lens and theater geometry in detail;[1] the following description has been condensed from his account.

In an Omnimax theater, film is projected through a fish-eye lens onto a hemispherical screen that fills the audience's field of view. This extreme field of view necessitates a large film format to provide sufficient resolution. Omnimax is the largest motion picture film format in use today—frames are oriented lengthwise on 70-mm film stock.

A 180° fish-eye lens (round type) projects half of the environment into a circular image. The Omnimax film image shown in Figure 1 is a 180° fish-eye projection with the bottom of the frame cropped to an approximately elliptical arc. The projected Omnimax image covers a full 180° horizontally but, due to the cropping of the frame, covers less vertically—about 135°. In a typical Omnimax theater the axis of projection (which pierces the film frame at the "projection center" shown in Figure 1) is elevated 11° from the horizontal.

Each pixel in the raster grid of an Omnimax image corresponds to a ray projected through the Omnimax lens. Pixels are mapped to rays by simulating the C430 Omnimax projection lens as follows:

The limiting circle of the 180° fish-eye projection is assigned a unit radius, and its center is chosen as the image origin. Because the projection lens is rotationally symmetric about the axis of projection, the effect of the lens can be described by a single function that relates radial displacement from the center of the circle to the angle of the projected ray from the axis of projection. Points in the image are assigned polar coordinates (r, θ) and are converted to spherical coordinates with angular coordinates θ and ϕ, where θ is longitude and ϕ is the angle from the axis of projection. The transformation from polar to spherical coordinates keeps θ the same and transforms r into ϕ. The expression for ϕ as a function of r, adapted from Max,[1] is

$$\phi(r) = 1.411269r - .094389r^3 + .25674r^5$$

See Max[1] for a more detailed discussion of Omnimax lens and theater geometry, and film frame layout.

Omnimax projection from cube projection

To produce an Omnimax projection, standard image generation programs are used to project the environment onto four faces of a cube centered at the viewpoint,[5] and this cube of texture is then processed to obtain an

Omnimax projection. The cube's orientation corresponds to a coordinate system that has a horizontal left-right axis and another axis coinciding with the axis of projection. Each face of the cube of texture is obtained by pointing the camera down the appropriate axis and rendering with a 90° view angle. The rendered views are the projections on the front, top, left, and right faces of the cube.

Figure 2 shows the projection of a 3D scene onto four faces of a cube. This cube of texture is shown unfolded in Figure 3, wherein the darkened areas correspond to regions of the scene that lie outside the Omnimax field of view. Figure 4 shows the region of the Omnimax frame that each face of the cube contributes to.

Creating the Omnimax projection from the cube of texture can be done in scan-line order. Each pixel in the Omnimax projection corresponds to a ray through the projection lens, the direction of which can be found with some straightforward trigonometry using the lens distortion formula given above. This ray is then intersected with the cube of texture to determine which texture pixel corresponds to the pixel in the Omnimax projection. Actually, this point-sampling approach will cause aliasing. A filtering technique that eliminates visible aliasing is described later in this article.

Figure 5 is an Omnimax projection obtained from the cube of texture in Figure 2. It has twice the horizontal resolution of the cube faces of Figure 2, and this corresponds to a 2:3 ratio of "source pixels" (the bright area of Figure 3) to "destination pixels" in the Omnimax projection. The exact ratio of source pixels to destination pixels isn't important as long as there is rough parity (having far more source pixels than destination pixels wastes source texture resolution; having far fewer source pixels than destination pixels magnifies the source texture unnecessarily). Empirical tests have shown that a resolution of approximately 2000×1500 is sufficient for raster Omnimax frames.[6]

A potential problem with our method is discontinuities in the Omnimax projection at the boundaries between the cube faces shown in Figure 4. For example, if a polygon in the Omnimax projection straddles a seam, it will be clipped and rendered in two separate cube faces. Whether or not the projected polygon will exhibit visible artifacts at the seam depends on the algorithms used for clipping, shading, and so on. In some cases it may be necessary to use more sophisticated algorithms, but in practice seams have never been a problem for us.

The framework for projection need not be a cube. An Omnimax projection can be obtained from any combination of perspective views with viewpoint at the camera that collectively covers the Omnimax field of view. For example, it is possible to use just two perspective views, one covering the left half of the Omnimax field of view, the other covering the right half. This scheme, however, requires very large view angles that result in poor image resolution near the image center. All things considered, the cube appears to be the best framework for projection.

Image filtering

To control aliasing, the cube-face images must be filtered, not point-sampled,[7] as Figure 6 shows. We treat
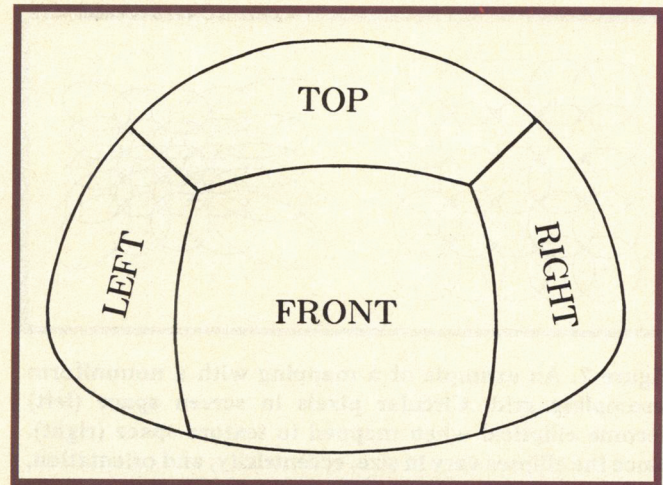

Figure 4. The correspondence between cube faces and regions of the Omnimax projection.

© NYIT, 1984

Figure 5. An Omnimax projection made from the cube of texture of Figures 2 and 3. Scene modeling by Jules Bloomenthal (tree), Paul P. Xander (terrain), Ned Greene (sky), and Dick Lundin (overall scene composition and rendering).
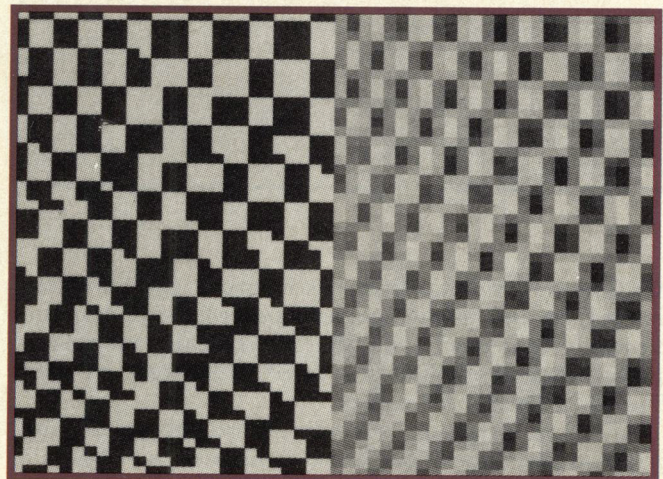

Figure 6. Portions of Omnimax images generated from checkerboard cube faces. Aliasing is evident with point sampling (left) but not with filtering (right).
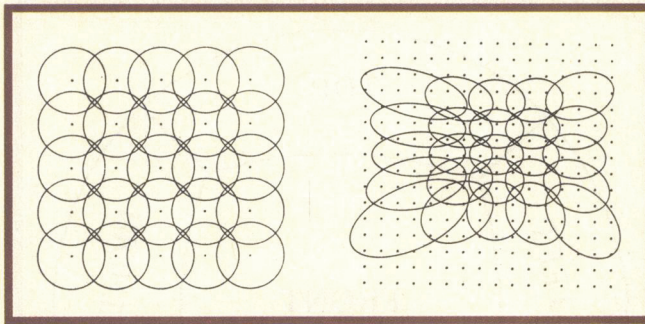
Figure 7. An example of a mapping with a nonuniform resampling grid. Circular pixels in screen space (left) become elliptical when mapped to texture space (right). Since the ellipses vary in size, eccentricity, and orientation, the filter is space variant. Dots mark pixel centers.

Table 1. Classification of texture filters.

FILTER AREA (in texture space)	FILTER CROSS-SECTION	COST	REFERENCE
		constant	(point sampling)
		constant	Dungan, Stenger, and Sutty , Williams
		constant	Crow
		constant	Perlin
		∝ area	Catmull
		∝ area	Blinn and Newell
		∝ area	Feibush, Levoy, and Cook
		∝ area	Gangnet, Perny, and Coueignoux

pixels in the Omnimax frame as square or circular areas instead of points, so their corresponding rays through the projection lens form cones that intersect the cube faces in quadrilateral or elliptical areas.[6,8] These areas are then filtered to obtain pixel values.

Omnimax distortion is nonlinear, so the resampling grids on the cube faces are nonuniform, as Figure 7 illustrates. Consequently a space-variant filter is required instead of the simpler space-invariant filter techniques. In response to the need for generality and speed, we developed a new filtering technique.

Image filtering for Omnimax distortion is very similar to the technique of texture mapping onto curved surfaces in 3D: both require space-variant filters.[9] The filter we present can be used for either application.

Applying the terminology of texture mapping, the cube-face image is called the *texture*, with coordinates u and v. The output picture is called the *screen*, with coordinates x and y. Some methods for texture filtering are listed in Table 1, classified according to kernel shape. The most important features of a filter are the generality of its shape, which determines quality, and the number of samples that must be accessed and the cost of each access, which determine speed.

High-quality filters such as those by Catmull,[10] Blinn et al.,[11] Feibush et al.,[12] and Gangnet et al.[13] operate on arbitrary quadrilaterals or ellipses. Their computational cost is proportional to texture area, so the cost can be very high when large texture areas map to small screen areas.

Fast filters such as those by Dungan et al.,[8] Williams,[14] Crow,[15] and Perlin[16] have a constant cost per screen pixel, but are limited to orthogonally oriented squares, rectangles, or ellipses, so they cannot filter elongated diagonal areas accurately. This limited shape control causes either aliasing or blurring of the texture. These "constant-cost" filters also require extra time and memory to preintegrate or prefilter the texture. Also, when the texture changes from frame to frame, as it does in our method of Omnimax frame creation, the cost per screen pixel (including setup time) is not constant, but includes a term proportional to texture area.

In general, the constant-cost filters are most desirable for static textures with mappings that compress large areas to small. Since the texture areas corresponding to each screen pixel are small in our application (typically nine to 16 pixels), an efficient, high-quality filter may be nearly as fast as a constant-cost filter in this case.

Elliptical weighted average filter

We developed the elliptical weighted average (EWA) filter to attain more efficient, high-quality filtering. The filter area is an arbitrarily oriented ellipse, as in the filters by Feibush et al.[12] and Gangnet et al.[13] These two methods require every texture sample point to be mapped to or from screen space for weighting by the circular kernel residing in screen space. To avoid this cost, EWA distorts the circular screen space kernel into an ellipse in texture space, so texture weighting can be done more directly. This is similar to the method used by Blinn et al., who distorted a screen square into a texture parallelogram.[11]

The radial cross section of the kernel is controlled by a weight lookup table, so it can be any function. Like other high-quality filters, EWA's cost is proportional to the texture space area. However, because it avoids mapping each texture pixel between screen space and texture space, it is faster than filters of similar quality.

Under the perspective transformation, a circular screen space pixel will normally map to an elliptical texture space area, assuming local planarity of the surface (this approximation breaks down near vanishing points, where the circles can map to parabolas or hyperbolas). Point-in-

IEEE CG&A

ellipse testing can be done with one function evaluation (this is faster than point-in-quadrilateral testing, which requires substitution into four line equations). The function for this test is a quadratic in the texture coordinates u and v:

$$Q(u,v) = Au^2 + Buv + Cv^2$$

where $u = 0$, $v = 0$ is the center of the ellipse. This function is an elliptical paraboloid. Points inside the ellipse satisfy $Q(u,v) < F$ for some threshold F. In texture space the contours of Q are concentric ellipses (Figure 8), but when mapped to screen space, they are nearly circular. Since Q is parabolic it is proportional to r^2, where r is the distance from the center of a pixel in screen space. This radius r is just the parameter needed when indexing a kernel, so Q can serve two purposes: inclusion testing and kernel indexing.

The kernel $f(r)$ is stored in a weight lookup table, *WTAB*. Rather than index *WTAB* by r, which would necessitate the calculation of $r = \sqrt{Q}$ at each pixel, we define

$$WTAB[Q] = f(\sqrt{Q})$$

so that the array can be indexed directly by Q.

Warping a lookup table for computational efficiency is a useful trick that has been applied by others.[3,17] A good kernel to use is the Gaussian $f(r) = e^{-\alpha r^2}$, shown in Figure 9, for which $WTAB[Q] = e^{-\alpha Q}$. The Gaussian is preferred to the theoretically optimal *sinc* kernel because it decays much more quickly. By properly scaling A, B, C, and F, the length of the *WTAB* array can be controlled to minimize quantization artifacts (several thousand entries have proven sufficient). The parameters F and α can be tuned to adjust the filter cutoff radius and the degree of pixel overlap.

To evaluate Q efficiently, we employ the method of finite differences. Since Q is quadratic, two additions suffice to update Q from one pixel to the next.[3] The following pseudocode implements the EWA filter for monochrome pictures (it is easily modified for color). Integer variables are lowercase; floating-point variables are uppercase.

```
/* Let texture[v,u] be a 2-dimensional array holding texture */
< Compute texture space ellipse center (U0,V0)
        from screen coordinates (x,y) >

< Compute (Ux,Vx) = [∂u/∂x, ∂v/∂x] and (Uy,Vy) = [∂u/∂y, ∂v/∂y] >

/* Now find ellipse corresponding to a circular pixel: */
A = Vx*Vx+Vy*Vy
B = -2.*(Ux*Vx+Uy*Vy)
C = Ux*Ux+Uy*Uy
F = Ux*Vy-Uy*Vx
F = F*F
< scale A, B, C, and F equally so that F = WTAB length >

/* Ellipse is AU²+BUV+CV²=F,  where U=u-U0, V=v-V0 */

EWA(U0,V0,A,B,C,F)
```

```
begin
    < Find bounding box around ellipse: u1≤u≤u2, v1≤v≤v2 >
    NUM = 0.
    DEN = 0.
    DDQ = 2.*A
    U = u1-U0
    /* scan the box */
    for v=v1 to v2 do begin
        V = v-V0
        DQ = A*(2.*U+1.)+B*V    /* =Q(U+1,V)-Q(U,V) */
        Q = (C*V+B*U)*V+A*U*U
        for u=u1 to u2 do begin
            /* ignore pixel if Q out of range */
            if Q<F then begin
                WEIGHT = WTAB[floor(Q)]
                /* read and weight texture pixel */
                NUM = NUM+WEIGHT*texture[v,u]
                /* DEN is denominator (for normalization) */
                DEN = DEN+WEIGHT
            end
            Q = Q+DQ
            DQ = DQ+DDQ
        end
    end
    return(NUM/DEN)
end
```

This implementation can be optimized further by removing redundant calculations from the v loop and, with proper checking, by using integer variables throughout.

The EWA filter computes the weighted average of elliptical areas incrementally, requiring one floating-point multiply, four floating-point adds, one integerization, and one table lookup per texture pixel. Blinn et al.'s method, which is the most similar to EWA, appears to have
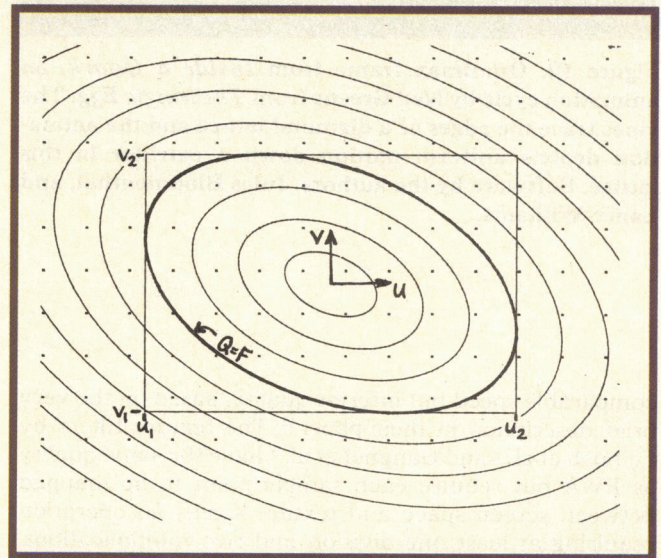

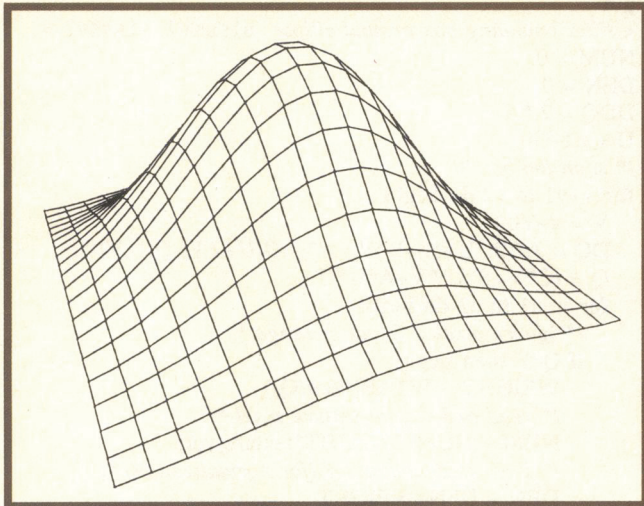**Figure 8.** Contours of elliptical paraboloid Q and box around $Q = F$. Dots are centers of texture space pixels.
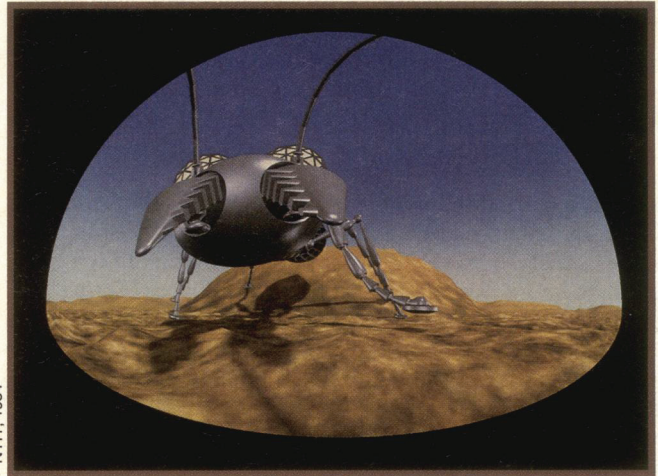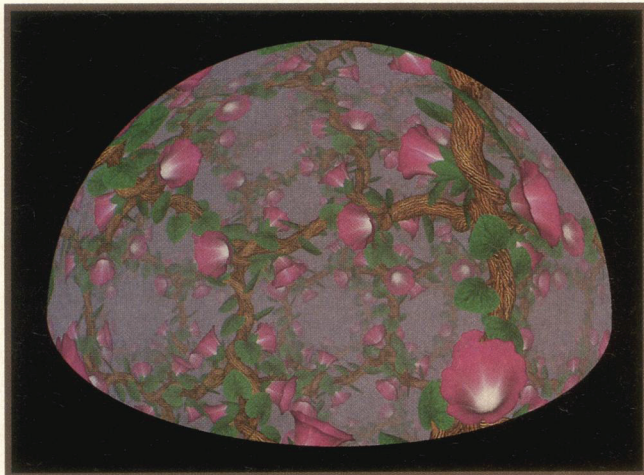
VALEO EX. 1028_006

Figure 9. Gaussian kernel $e^{-\alpha r^2}$.

© NYIT, 1984

Figure 11. Omnimax frame from *Revenge of the Ant*, animation of a mechanical ant by Dick Lundin from *The Magic Egg*. In this 12-second sequence the ant climbs out of the "anthill" and charges the camera. Terrain modeling by Lance Williams and Paul P. Xander.

© NYIT, 1984

Figure 10. Omnimax frame from *Inside a Quark*, an animation cycle by Ned Greene from *The Magic Egg*. The vines trace the edges of a diamond lattice and the animation depicts uniform motion down a corridor in this lattice. Software by the authors, Jules Bloomenthal, and Lance Williams.

comparable speed but inferior quality, based on the very brief description in their paper.[11] The texture filters by Feibush et al.[12] and Gangnet et al.[13] have the same quality as EWA but require each sample point to be mapped between screen space and texture space, an operation requiring at least one division and two multiplications, even when done incrementally.[18] Based on this analysis the EWA filter appears to be about twice as fast as the other methods of similar quality.

Areas for future work

The mapping program to create an Omnimax projection from a cube projection can easily be modified to produce other nonlinear projections such as cylindrical projections (the Mercator projection, for example), or a picture of the environment as it would appear reflected in a chrome ball. The projection formulas for these applications are derived easily. The cube faces required depend on the projection—for example, all six are required to produce a chrome ball reflection.

Because the computational cost of EWA filtering is proportional to texture space area, using it to map large texture areas to small screen areas is very slow compared to constant-cost filters. An untested method of dealing with this problem is to use EWA (or some other high-quality filter) on a prefiltered image pyramid.[8,14] EWA filtering would be performed on small elliptical areas at the appropriate level in the image pyramid. Actually, performing such filtering at two adjacent levels in the image pyramid and interpolating the results, as in Williams,[14] would be preferable. In theory this hybrid approach offers the best of both worlds: high-quality filtering at a constant, relatively low cost per pixel.

An unexplored approach to digital Omnimax distortion is decomposition of the 2D mapping and filtering into two passes of 1D mapping and filtering[19] using least-squares approximations.[20]

Conclusions

This method is completely general in the sense that an Omnimax projection can be produced of any scene that can be rendered in perspective. Only a modest programming effort is required—existing rendering programs need not be modified, and the only new software is the mapping program. All rendering can be done with existing

IEEE CG&A

efficient programs that exploit the linearity of the perspective transformation.

These techniques were used at NYIT to create two sequences of animation for *The Magic Egg*, a compilation of Omnimax animation sponsored by SIGGRAPH 84 and produced by Garrick Films.[21-23] Figures 10 and 11 are frames from these sequences. Cube faces were rendered at a resolution of 1024 × 960; from these 1966 × 1436 × 24-bit Omnimax frames were made. The high resolution requirements of the Omnimax format result in large data volumes (up to 8.5M bytes per frame) and slow frame generation. On a Digital Equipment Corp. VAX 11/780, the four cube-face images needed for each Omnimax frame took from one to six hours to create, followed by mapping and filtering, which took one hour.

To date, about 20 minutes of computer-generated Omnimax animation has been produced in raster format, much of it rendered with ray tracing on supercomputers. Our method provides an alternative to this brute-force approach. ∎

Acknowledgments

References

1. Nelson L. Max, "Computer Graphics Distortion for IMAX and OMNIMAX Projection," *Proc. Nicograph 83*, Dec. 1983, pp.137-159.

2. Tom Duff, "Tools for 3-D Rendering," *Computer Graphics* (*Proc. SIGGRAPH 85*), Vol. 19, No. 3, July 1985, pp.41-44.

3. Nelson L. Max, "ATOMLLL—ATOMS with Shading and Highlights," *Computer Graphics* (*Proc. SIGGRAPH 79*), Vol. 13, No. 3, Aug. 1979, pp.165-173.

4. "News Update," *Electronic Imaging*, Vol. 4 No. 2, Feb. 1985, p.18.

5. Ned Greene, "Applications of World Projections," *Proc. Graphics Interface 86*, May 1986.

6. Charles E. Henderson and Agnis Kaugars, "Satellite Imagery in the Production of an Omnimax Animated Film Sequence," *Proc. Nicograph 83*, Dec. 1983, pp.127-136.

7. Franklin C. Crow, "The Aliasing Problem in Computer Generated Shaded Images," *Comm. ACM*, Vol. 20, Nov. 1977, pp.799-805.

8. William Dungan Jr., Anthony Stenger, and George Sutty, "Texture Tile Considerations for Raster Graphics," *Computer Graphics* (*Proc. SIGGRAPH 78*), Vol. 12, No. 3, Aug. 1978, pp.130-134.

9. Paul Heckbert, "Survey of Texture Mapping," *Proc. Graphics Interface 86*, May 1986.

10. Edwin E. Catmull, *A Subdivision Algorithm for Computer Display of Curved Surfaces*, PhD thesis, Dept. Computer Sci., Univ. of Utah, Dec. 1974.

11. James F. Blinn and Martin E. Newell, "Texture and Reflection in Computer Generated Images," *Comm. ACM*, Vol. 19, No. 10, Oct. 1976, pp.542-547.

12. Eliot A. Feibush, Marc Levoy, and Robert L. Cook, "Synthetic Texturing Using Digital Filters," *Computer Graphics* (*Proc. SIGGRAPH 80*), Vol. 14, No. 3, July 1980, pp.294-301.

13. Michel Gangnet, Didier Perny, and Philippe Coueignoux, "Perspective Mapping of Planar Textures," *Proc. Eurographics 82*, 1982, pp.57-71 (slightly superior to version in *Computer Graphics*, Vol. 16, No. 1, May 1982, pp.70-100).

14. Lance Williams, "Pyramidal Parametrics," *Computer Graphics* (*Proc. SIGGRAPH 83*), Vol. 17, No. 3, July 1983, pp.1-11.

15. Franklin C. Crow, "Summed-Area Tables for Texture Mapping," *Computer Graphics* (*Proc. SIGGRAPH 84*), Vol. 18, No. 3, July 1984, pp.207-212.

16. Kenneth Perlin, "Course Notes," *SIGGRAPH 85 State of the Art in Image Synthesis Seminar*, July 1985.

17. Kenneth Turkowski, "Anti-Aliasing Through the Use of Coordinate Transformations," *ACM Trans. Graphics*, Vol. 1, No. 3, July 1982, pp.215-234.

18. Alvy Ray Smith, "Incremental Rendering of Textures in Perspective," *SIGGRAPH 80 Animation Graphics Seminar Notes*, July 1980.

19. Edwin E. Catmull and Alvy Ray Smith, "3-D Transformations of Images in Scanline Order," *Computer Graphics* (*Proc. SIGGRAPH 80*), Vol. 14, No. 3, July 1980, pp.279-285.

20. Donald Fraser, Robert A. Schowengerdt, and Ian Briggs, "Rectification of Multichannel Images in Mass Storage Using Image Transposition," *Computer Vision Graphics and Image Processing*, Vol. 29, No. 1, Jan. 1985, pp.23-36.

21. Patrice M. Wagner, "Omnimax: Perfecting a Medium for 3-D Digital Animation," *Computer Graphics World*, Oct. 1984, pp.10-16.

22. Cal Kirchhof, "The Making of *The Magic Egg*," *The Big Frame*, Spring 1985, pp.9-12.

23. Paul S. Heckbert, "Making *The Magic Egg*: A Personal Account," *IEEE CG&A*, Vol. 6, No. 6, June 1986, pp.3-8.
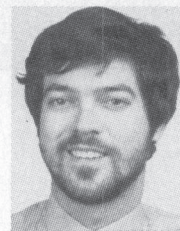
Ned Greene has been a senior member of the research staff at the New York Institute of Technology Computer Graphics Lab since 1980. From 1978 to 1980 he was a programmer in computer-aided design for McDonnell Douglas Automation.

His research interests include graphics software for image synthesis and animation, and computer animation as an artistic medium. His computer images have been widely published, and excerpts from his animation have been televised in the US, Europe, and Japan. Greene holds a BA in mathematics from the University of Kansas. He is a member of ACM SIGGRAPH.

Greene can be contacted at the NYIT Computer Graphics Lab, PO Box 170, Old Westbury, NY 11568. Heckbert can be contacted at Pixar, PO Box 13719, San Rafael, CA 94913.

Paul S. Heckbert is currently doing animation research and development at Pixar in San Rafael, California. From 1980 to 1985 he was software manager at NYIT's Computer Graphics Lab. He received a BS in applied mathematics from the Massachusetts Institute of Technology in 1980, where the Architecture Machine Group introduced him to computer graphics.

His research interests include mathematical modeling, simulation of nature, image synthesis, and image processing. Heckbert is a member of ACM SIGGRAPH.