SYNTHETIC TEXTURING USING DIGITAL FILTERS

by

Eliot A. Feibush, Marc Levoy, Robert L. Cook
Program of Computer Graphics
Cornell University
Ithaca, New York 14853

ABSTRACT

Aliasing artifacts are eliminated from computer generated images of textured polygons by equivalently filtering both the texture and the edges of the polygons. Different filters can be easily compared because the weighting functions that define the shape of the filters are pre-computed and stored in lookup tables. A polygon subdivision algorithm removes the hidden surfaces so that the polygons are rendered sequentially to minimize accessing the texture definition files. An implementation of the texture rendering procedure is described.

COMPUTING REVIEWS CATEGORY: 8.2

KEYWORDS: Computer Graphics, Anti-Aliasing, Sampling, Digital Filtering, Texturing, Hidden Surface Removal

1. INTRODUCTION

Sampling converts a function into a sequence of discrete values so the function can be reproduced at a finite resolution. If the sampling rate is insufficient for the function, then the discrete values will contain aliasing artifacts. The most common aliasing artifacts in computer generated images are jagged edges and Moiré patterns. Animated sequences can also suffer from temporal aliasing artifacts such as strobing and false motion (e.g., wagon wheels that appear to spin backwards).

There are two solutions to the aliasing problem in computer graphics: increasing the sampling rate and filtering the original function. Increasing the sampling rate means computing and displaying the image at a higher resolution. Filtering the original function means blurring the image before sampling. The two approaches are not mutually exclusive. Catmull and Crow point out that if the only goal is to eliminate aliasing, then filtering the original function is better than increasing the sampling rate (4,5). Furthermore, it is often impossible to increase the sampling rate without more costly display technology.

Filtering was introduced to the computer graphics literature by Catmull in (3) and was studied comprehensively by Crow in (5). Since then, several researchers have made filtering an integral part of their synthetic imaging systems. These systems can be classified by the type of data they display:

1. Continuous functions (parametric data):
   1. Crow (5) (parabolas)
2. Objects (polygonal or patch data):
   1. Catmull (3) (patches)
   2. Crow (5) (polygons)
   3. Crow (6) (vectors)
   4. Catmull (4) (polygons)
   5. Whitted (9) (polygons)
3. Textures (pixel data):
   1. Catmull (3) (on patches)
   2. Blinn and Newell (2) (on patches)
   3. Crow (6) (characters)
   4. Blinn (1) (on patches)

Most of the above implementations that filter only edges use an unweighted filter (i.e., a filter with a weighting function that is constant throughout the convolution mask). This is far better than no filter at all, but not as good as a weighted filter. Unweighted filters have been used to avoid the computational expense of weighting functions. Researchers who have used weighted filters for the texture did not use the same filter for the edges of the surfaces.

The texture and the edges of each surface should be filtered separately and equivalently to produce correct renderings. The texture should be filtered first to remove excessively high

frequencies that could cause aliasing in the form of Moiré patterns. Then the edges of surfaces should be filtered to eliminate the excessively high frequencies that could cause aliasing in the form of "jaggies." Of the implementations listed above that include texturing, only Catmull's (3) applies equivalent filters to both the texture and the edges. He uses unweighted filters to display environments of textured patches.

2. IMPLEMENTATION

This paper describes the implementation of two filtering processes used for displaying textured polygons. Both the texture filter and the edge filter are based on a polygon subdivision hidden surface algorithm, and both procedures use pre-computed lookup tables to define any desired filter shape.

2.1 DATA REPRESENTATION

Objects in this texture rendering system are defined by planar polygons. These polygons may be concave, may contain holes, and may be coplanar with other polygons in the environment to create detail faces within larger faces. Each polygon is assigned a texture which completely covers its surface. A texture is a two-dimensional array of texture definition points. The color of each point is represented by either one intensity value, producing a gray scale texture, or three intensity values, producing a full color texture.

The construction of the database, including the creation and assignment of textures to polygons, is handled by an interactive geometric modeling package which is described by Feibush (7). The textures can be generated by any of several methods, including optical scanning or synthetic airbrushing. In practice, several techniques are usually combined for each texture.

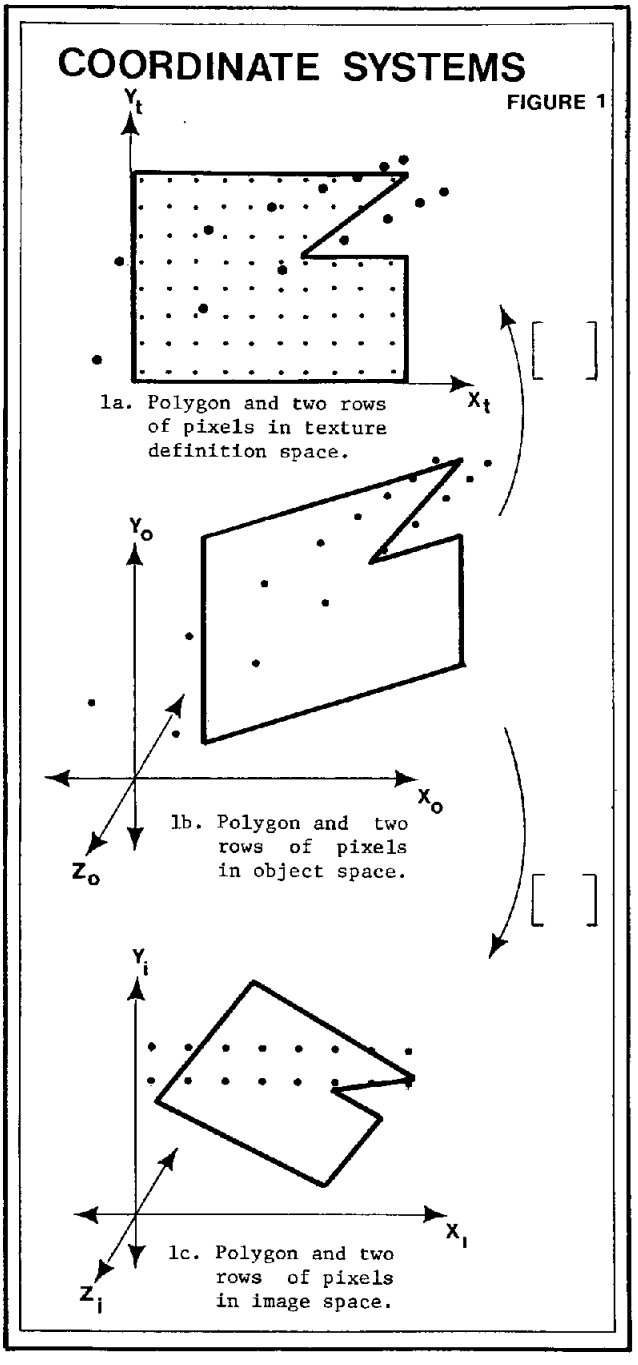
The word "pixel" (picture element) must be clearly defined. A pixel is often thought of as a rectangular block whose width is equal to the distance between centers of adjacent blocks. In this paper, however, a pixel is defined as an infinitesimal point having an intensity value.

2.2 COORDINATE SYSTEMS

Three coordinate systems are used:

1. Texture definition space.
2. Object definition space.
3. Image display space.

The first coordinate system is a two-dimensional space for defining textures. The textures are created and stored on the X - Y plane shown in Figure 1a. The second coordinate system is a three-dimensional space used for defining the polygons. When a texture is assigned to a polygon, a matrix is constructed that transforms the polygon from its location in object space to the X - Y plane in texture space, as shown in Figure 1b. The third coordinate system is a three-dimensional space used for displaying the object. A single matrix is used to transform the polygons from object space to image space and create the perspective distortion, as shown in Figure 1c.



Also shown in the figure are two rows of display pixels which are drawn as points in accordance with the above definition of the term "pixel." These display pixels are initially defined in image space and can be transformed to object space (care must be taken in reversing the perspective distortion), and then to texture definition space, as shown in the figure.

2.3 HIDDEN SURFACE ALGORITHM

Most researchers use a scanline hidden surface algorithm to determine the contribution of each polygon in the object to the display pixels. In a scanline algorithm, all the polygons contributing to the color of a pixel are processed simultaneously. The color of each pixel can therefore be computed in a single pass. An alternative solution presented here is to use a polygon subdivision hidden surface algorithm to compute the visible portions of all the polygons before computing the color of the display pixels. The color of each pixel is built up piecemeal from the visible portions of each contributing polygon. The polygon hidden surface algorithm developed by Weiler (8) has been implemented.

Separating the hidden surface removal from the filtering process has a significant advantage over approaches that do both tasks simultaneously. Rendering a textured polygon involves accessing its texture definition file. A scanline algorithm requires simultaneous access to the texture files of all the polygons that are visible on each scanline. The storage problems this entails can not be taken lightly even in a virtual memory machine, particularly if the textures are high resolution color images. The polygon subdivision hidden surface algorithm produces a list of visible polygons defined at machine precision so that the polygons can be rendered sequentially. Hence only one texture file has to be accessible at a given time, and each texture file is processed completely before another one is needed.

2.4 TEXTURE FILTERING

Whenever a polygon is displayed in perspective and is not parallel to the picture plane, the amount of blurring required to avoid aliasing varies across the polygon and is different in the horizontal and vertical directions. The method described in this paper produces sufficient blurring at each display pixel by selecting specific texture definition points that correspond to the pixel and then filtering the points to determine the color of the pixel. A description of the procedure follows:

1. For a given view of the object, use the polygon subdivision hidden surface algorithm to make a list of the portions of the polygons that are visible in image space. The visible portions are called display polygons.

2. Working with one display polygon at a time, make a list of all the pixels that contribute to the display of the polygon. A convolution mask, whose shape is determined by the weighting function of the filter, is centered at each display pixel. Each pixel has a bounding rectangle, which is the smallest rectangle that completely bounds the pixel's convolution mask. The bounding rectangles may overlap depending on the size and shape of the convolution masks. List every display pixel whose bounding rectangle is completely or partially within the polygon, as shown in Figure 2a. Also save a list of the intersections of each bounding rectangle with the polygon.

3. For each display pixel, transform its bounding rectangle from image space to object space and then to texture definition space. The rectangle can be transformed to texture definition space because its vertices have three-dimensional coordinates coplanar with the display polygon. The rectangle in image space transforms to a quadrilateral in texture space, as shown in Figure 2b. The texture definition points within this quadrilateral contribute to the color of the display pixel. To simplify the selection of these points, a rectangle is constructed around the bounding quadrilateral. This rectangle includes some texture definition points that do not contribute to the color of the pixel, but these extra points will be eliminated from the filtering in step 6.

4. Transform the parent polygon of the current display polygon from object space to texture definition space. Clip the rectangle around the convolution mask quadrilateral against the parent polygon. The texture definition points within this area will be filtered, as in Figure 2c.

5. Transform each texture point that will be filtered to object space and then to image space, as shown in Figure 2d.

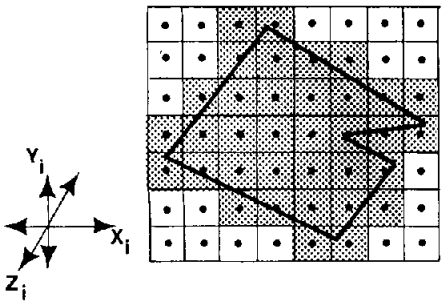
6. Eliminate the extra points selected in step 3 by clipping the transformed texture points against the bounding rectangle of the convolution mask in image space, as shown in Figure 2e.

7. Filter the selected texture points by computing the weighted average of their color values. Points near the center of the convolution mask are weighted more heavily than those near the edge. The cone shown in Figure 2f represents one possible weighting function. The weighting function is computed at a number of locations and the values are stored in a two-dimensional lookup table. The location of each transformed texture point within the convolution mask is used as an index to the lookup table. The color values of all the texture definition points are multiplied by their respective values in the lookup table and summed together in a weighted average. When the transformed texture points do not coincide precisely with the discrete locations at which the weighting function is calculated, the nearest value is used.

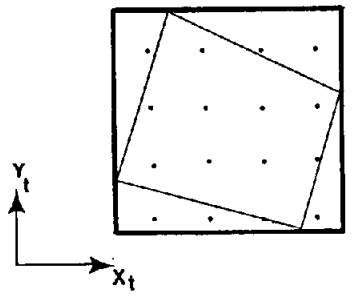
This completes the texture filtering. The edges of the polygons are filtered next to complete the rendering procedure.

TEXTURE FILTERING

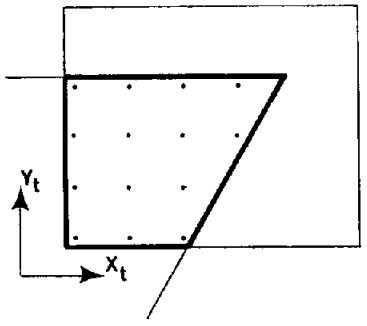
FIGURE 2



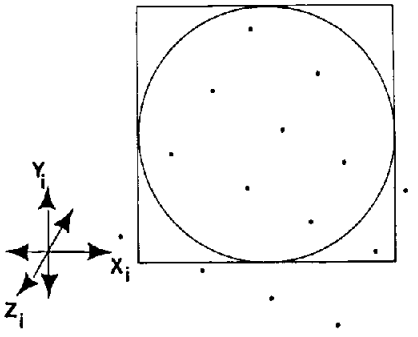
2a. Select the pixels that contribute to the display of the polygon. For clarity, the bounding rectangles shown do not overlap. Figures 2b-2f illustrate the texture calculation for each selected pixel.



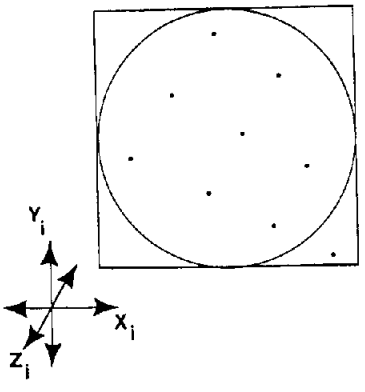
2b. Transform the bounding rectangle to texture space and select texture definition points.



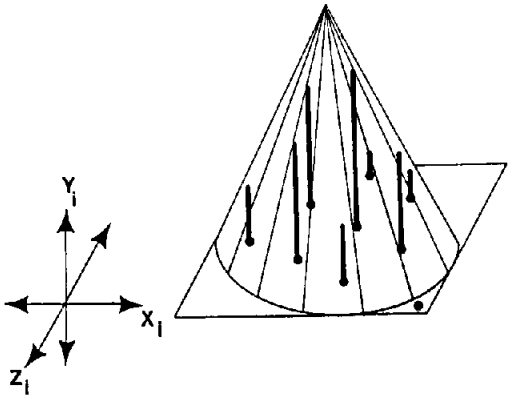
2c. Select the points inside the transformed parent polygon.



2d. Transform the points to image space.



2e. Select the points inside the bounding rectangle.



2f. Compute the weighted average of the selected points.

2.5 EDGE FILTERING

The intensity of a pixel whose convolution mask is completely within one display polygon is determined just by the texture filter. The intensity of a pixel near an edge of a polygon is only partly determined by the texture filter because its convolution mask covers more than one display polygon. The intensity of a pixel computed by the texture filter for one polygon is weighted by the percentage of the total intensity of the pixel that is contributed by that polygon. The total intensity of a display pixel is built up sequentially as each polygon is rendered by accumulating the partial intensities in a frame buffer.

The contribution of a polygon to a pixel is determined by filtering its edges with the same weighting function that was used for the texture filter. But unlike the texture, which is defined by discrete points, the edges of the polygon are defined by a continuous function. Edge filtering is therefore an analytic problem. The cone above the convolution mask shown in Figure 3a represents one possible weighting function for the filter. The value of the weighting function at any point in the convolution mask is the distance from the point to the surface above it. The contribution of a polygon to the pixel is the percentage of the volume of the entire cone that is above the polygon, as shown by the shaded volume in Figure 3a. The calculation of this volume is described below.

1. Clip the display polygon against the bounding rectangle of the convolution mask, as shown in Figure 3b. The points of intersection of each polygon edge with the bounding rectangle are already known from step 2 above. The clipped polygon may be concave and may contain holes.

2. For each vertex of the clipped polygon, construct a triangle with the following sides (as shown in Figure 3c):

    1. BASE is the line segment between the current vertex and the next vertex (going clockwise around the polygon).

    2. SIDE1 is the line segment between the current vertex and the pixel.

    3. SIDE2 is the line segment between the next vertex and the pixel.

3. Calculate the volume above the polygon from the volumes above all the triangles constructed in step 2, as shown in Figure 3c. The volume above a single triangle is added to the total if the cross product of SIDE1 and SIDE2 is negative; it is subtracted from the total if the cross product is positive.

4. The task of finding the volume above an arbitrary polygon has now been simplified to finding the volume above a series of triangles, each having one vertex at the pixel. The problem can be simplified further. For each triangle, the perpendicular from the pixel to BASE (or to its extension) forms two right triangles. The volume above the original triangle is the sum of the volumes above the two right triangles if the perpendicular lies within the triangle, as shown in Figure 3d; it is the difference of the volumes if the perpendicular lies outside of the triangle, as shown in Figure 3e.

5. The problem has now been simplified to finding the volume above a group of right triangles. The base and height of each triangle are used as indices to a lookup table that contains the volume above this triangle for the given weighting function. Care must be taken in computing the lookup table so that areas inside the bounding rectangle but outside the convolution mask have no volume above them. Only the shaded area of Figure 3f has volume above it.

Each filter shape needs only one lookup table, regardless of the filter's absolute size. The filter size can be changed by scaling the indices to the lookup table.

The organization of the lookup table assumes that the filter function is circularly symmetric. For a filter that is not circularly symmetric, one more parameter describing the location of the right triangles (such as a polar sweep angle) is required. A four parameter lookup table would give the volume above the original triangle without constructing the two right triangles. The X and Y positions of the two vertices of BASE of the original triangle would be used as the indices to the four parameter lookup table. This further simplifies the filtering computation but requires significantly more table storage.

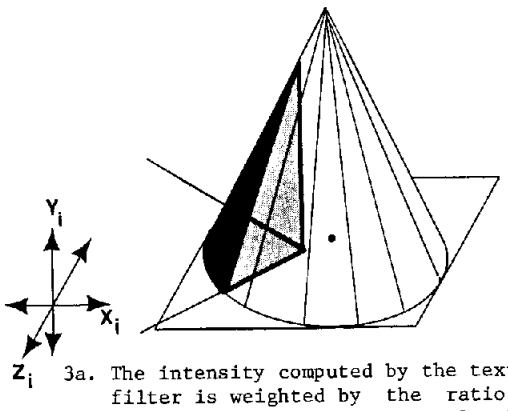
3. EXAMPLES

A polygon textured with alternating red and white vertical stripes has been rendered by the system described in this paper. Due to the rotation and perspective transformations, the number of texture definition points that were filtered for each display pixel varied considerably. The images were computed at a resolution of 512 x 512 and displayed on a 24-bit color frame buffer.

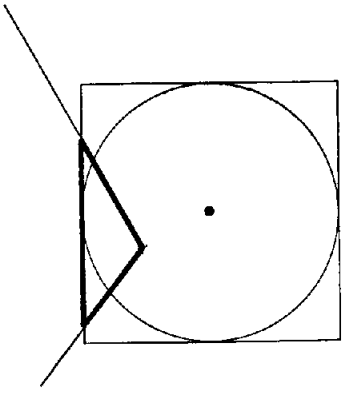
The five images of the polygon demonstrate the effectiveness of different filters, as shown in Figures 4a-e. Figure 4a shows the polygon in texture definition space. In Figure 4b this polygon is displayed in image space with no filtering. In Figure 4c it is displayed using an unweighted filter with a square convolution mask whose sides are equal to the distance between adjacent display pixels. In Figure 4d the polygon is displayed using a filter with a Gaussian weighting function that has a standard deviation equal to the distance between adjacent display pixels. The convolution mask is a circle whose radius is equal to twice the standard deviation of

EDGE FILTERING

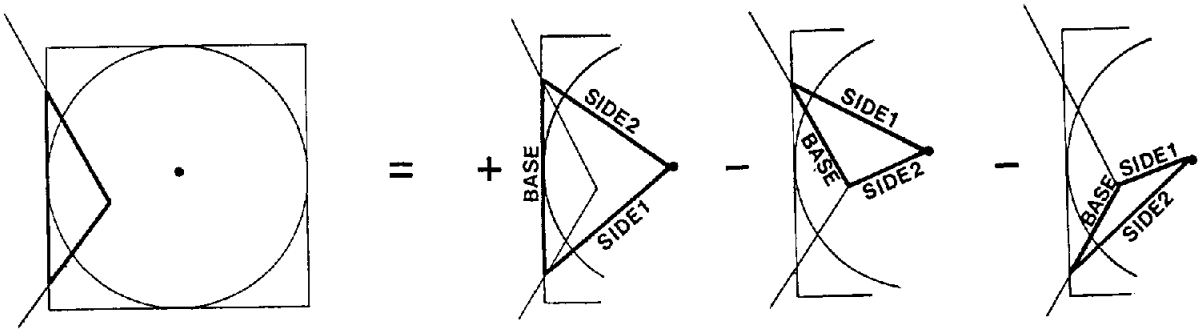
FIGURE 3



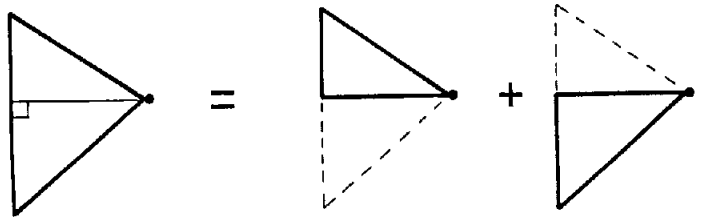
3a. The intensity computed by the texture filter is weighted by the ratio of the shaded volume to the total volume of the cone.



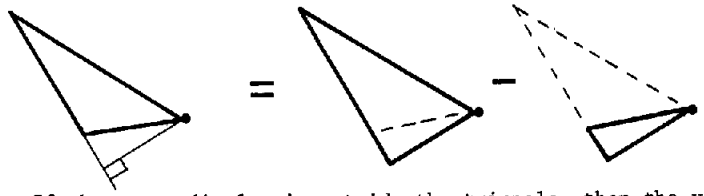
3b. Clip the polygon to the bounding rectangle of the pixel's convolution mask.



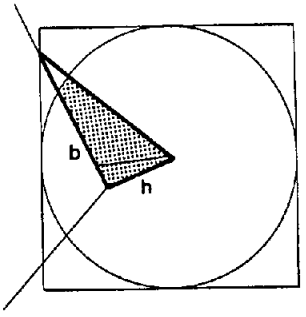
3c. For each vertex of the clipped polygon, construct the triangle formed by the vertex, the next vertex (going clockwise), and the pixel. From the volumes above these triangles, calculate the volume above the clipped polygon as shown in Figures 3d-f.



3d. For each triangle, construct the perpendicular from the pixel to BASE. If the perpendicular is inside the triangle, then the volume above the triangle is the sum of the volumes above the two right triangles formed by the perpendicular.



3e. If the perpendicular is outside the triangle, then the volume above the triangle is the difference of the volumes above the two right triangles formed by the perpendicular.



3f. Find the volume above each right triangle by using its height (h) and base (b) as indices to a lookup table. The value stored in the lookup table includes only the volume above the shaded portion of the triangle.

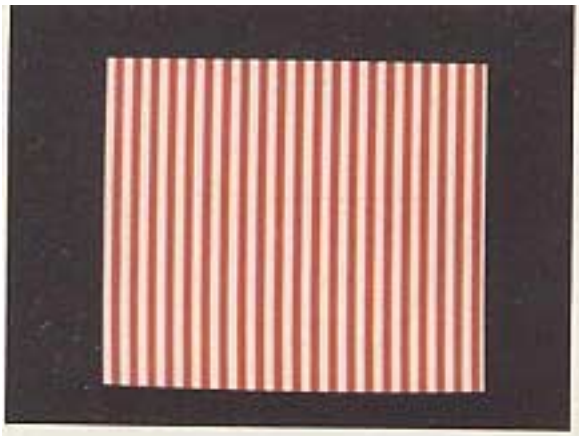


Figure 4a. Texture definition.

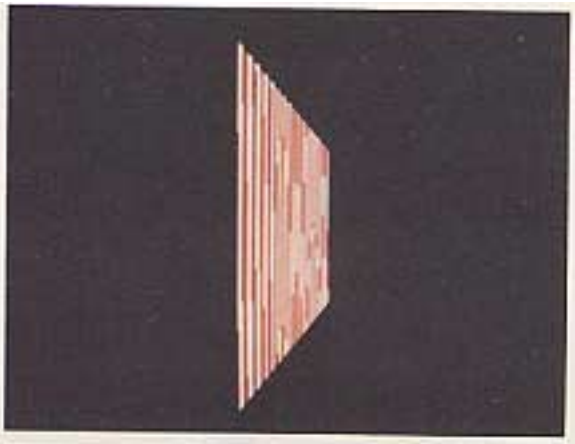


Figure 4b. No filter.

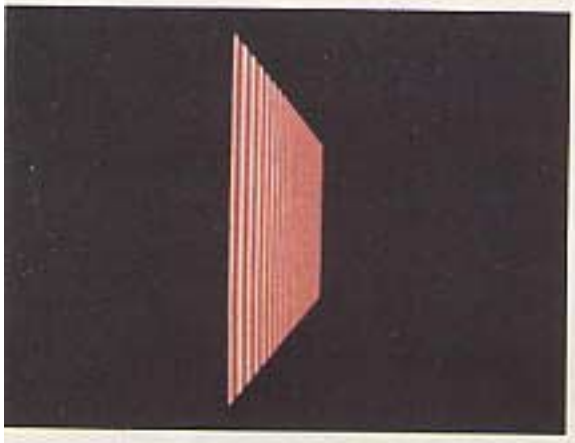


Figure 4c. Unweighted filter.

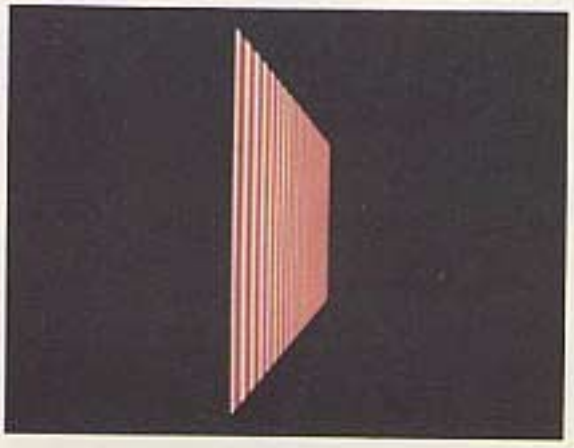


Figure 4d. Gaussian filter.

Figure 4e. Hardware magnification.

Figure 5. House.

the Gaussian.

Displaying the polygon with no filtering is completely unsatisfactory due to the jaggedness of not only the edges of the polygon but also the stripes in the texture. Using an unweighted filter is better and nearly satisfactory along the edges, but Moiré is still evident in the center of the polygon. The weighted filter, however, produces an excellent image. In the hardware magnification shown in Figure 4e, the polygon is inclined slightly more than in Figures 4b-d to enhance the visibility of the filtering. Notice that the filtering along the left edge of the polygon is equivalent to the filtering along the stripes of the texture.

The final image, Figure 5 shows the front facade of an imaginary house that has been rendered by the system described in this paper. It demonstrates an application of the system to a complex database composed of many polygons and textures. The textures were extracted from optically scanned photographs of real objects. The background was created by assigning an optically scanned photograph of a real site to the rearmost polygon in the environment.

4. LIMITATIONS

It is possible to obtain views where textures are magnified beyond their original resolution (i.e., zooming into a texture). During the texture filtering process, the area of the texture definition that corresponds to a display pixel will contain only a few texture definition points. To avoid reproducing these texture definition points as large square areas, the color values of the closest texture definition points are bilinearly interpolated.

Bilinear interpolation of the texture definition points is also necessary when the edges of two polygons are very close to each other, but do not actually touch. The hidden surface algorithm will detect the narrow slot between the polygons, so texture definition points of the polygon seen through the slot should be selected for filtering. If no texture definition points from the background polygon fall within the slot, then the nearby texture definition points are bilinearly interpolated.

More blurring is required to avoid aliasing if there are high frequency components in the texture definition. Aliasing that is not noticeable in a static image may become visible if the image is part of an animated sequence, so that additional blurring is needed.

5. CONCLUSIONS

Two filtering processes, one for the textures and one for the edges, are necessary for displaying textured polygons without introducing aliasing artifacts. A weighted filter, such as the Gaussian used in the examples, produces more realistic images than an unweighted filter or no filter at all.

A polygon subdivision hidden surface algorithm is superior to a scanline hidden surface algorithm for displaying textured polygons. By making a list of all the visible portions of the polygons before computing the color of the display pixels, the polygons can be filtered sequentially to minimize accessing each of the texture definition files.

Complex filters no longer have to be considered prohibitively expensive. If the filter's weighting function is stored in a lookup table instead of being computed at each pixel, an image can be computed in the same amount of time regardless of the complexity of the filter. The filter can be changed just by using a different lookup table.

ACKNOWLEDGEMENTS

This research has been performed at the Program of Computer Graphics at Cornell University and was funded in part by the National Science Foundation.

The authors thank Theodore Crane and Stuart Sechrest for implementing the polygon subdivision hidden surface algorithm.

REFERENCES

1. Blinn, James, "Computer Display of Curved Surfaces", Dissertation, University of Utah, 1978

2. Blinn, James, and Newell, Martin, "Texture and Reflection in Computer Generated Images", Communications of the ACM, Vol. 19, No. 10, Oct., 1976

3. Catmull, Edwin, "A Subdivision Algorithm for Computer Display of Curved Surfaces", Dissertation, University of Utah, 1974

4. Catmull, Edwin, "A Hidden-Surface Algorithm with Anti-Aliasing", Computer Graphics, Vol. 12, No. 3, Aug., 1978 (Siggraph '78)

5. Crow, Franklin, "The Aliasing Problem in Computer Synthesized Shaded Images", Dissertation, University of Utah, 1976

6. Crow, Franklin, "The Use of Grayscale for Improved Raster Display of Vectors and Characters", Computer Graphics, Vol. 12, No. 3, Aug., 1978 (Siggraph '78)

7. Feibush, Eliot, "Texture Rendering for Architectural Design", Computer Aided Design, Vol. 12, No. 2, Mar., 1980

8. Weiler, Kevin, "Hidden Surface Removal Using Polygon Area Sorting", Masters thesis, Cornell University, 1978

9. Whitted, Turner, "An Improved Illumination Model for Shaded Display", Preliminary papers to be published in Communications of the ACM, Aug., 1979