A HIDDEN-SURFACE ALGORITHM WITH ANTI-ALIASING

Edwin Catmull Computer Graphics Lab New York Institute of Technology Old Westbury, New York 11568

ABSTRACT

In recent years we have gained understanding about aliasing in computer generated pictures and about methods for reducing the symptoms of aliasing. The chief symptoms are staircasing along edges and objects that pop on and off in time. The method for reducing these symptoms is to filter the image before sampling at the display resolution. One filter that is easy to understand and that works quite effectively is equivalent to integrating the visible intensities over the area that the pixel covers. There have been several implementations of this method - mostly unpublished - however most algorithms break down when the data for the pixel is complicated. Unfortunately, as the quality of displays and the complexity of pictures increase, the small errors that can occur in a single pixel become quite noticeable. A correct solution for this filter requires a hidden-surface algorithm at each pixel! If the data at the pixel is presented as a depth-ordered list of polygons then the average visible intensity can be found using a polygon clipper in a way similar to that employed by two known hidden-surface algorithms. All of the polygons in a pixel are clipped against some front unclipped edge into two lists of polygons. The algorithm is recursively entered with each new list and halts when the front polygon is clipped on all sides, thereby obscuring the polygons behind. The area weighted colors are then returned as the value to be added to the other pieces in the pizel.

Key words: aliasing, clipping, computer graphics, filtering, hidden-surface removal, sampling.

CR classification: 8.2

INTRODUCTION

Aliasing is now being recognized as an important factor in analysis of image synthesizing algorithms. Attention has turned to anti-aliasing partly because of the need to refine pictures but mostly because the complexity of scenes has increased and with it the need to have pictures free of aliasing symptoms.

A polygon hidden-surface algorithm is presented here with the focus of attention on anti-aliasing. One goal has been to produce a "correct" image for the anti-aliasing technique used. Speed, while important, has played a secondary role. The techniques for hidden-surface elimination have improved in the last few years with the Sutherland et al [7] paper providing coherence to the development. Several new algorithms have come along [3,8,9], each adding new insight into the ways that we can take advantage of coherence for some class of objects to facilitate display.

Progress for anti-aliasing has been slower. In general pictures have not been extremely complicated and the more obvious effects of aliasing, like jagged edges, could be fixed up with ad hoc techniques. Methods for anti-aliasing have been presented in [1,2,4]. Frank Crow's dissertation was devoted to the topic and the results were published in [2].

ANTI-ALIASING

In general, the aliasing problem has been grossly underestimated in computer graphics. Its symptoms include:

- 1. jagged edges
- 2. small objects popping on and off the screen in successive frames
- 3. moire patterns in rendering periodic images
- 4. fine detail breaking up.

The problem occurs chiefly because image space is sampled at discrete points corresponding to the pixels.

There are several unpublished schemes for alleviating the problem for simple cases - in particular the symptom of jagged edges. They are unpublished because either they are incidental to some other algorithm or they are proprietary.

Frank Crow has written about anti-aliasing in [2]. From his study we can extract some key ideas:

- 1. The image space objects have sharpness and detail that cannot possibly be reproduced on a raster display. It is the attempt to sample that detail at discrete points in the image that causes the problem.
- Point sampling of an unfiltered object is never correct at any resolution. It is frequently thought that the symptoms of aliasing will not be noticeable if the resolution is high enough. Unfortunately, this is not true.

3. The image should be filtered to eliminate detail that is too fine. After filtering the image can be sampled.

One simple filter is to integrate the visible intensities over the area of each pixel. In other words we take the average visible intensity over the square area represented by each pixel if the image is divided into a rectangular grid. This corresponds to convolving the continuous image with a two-dimensional Fourier (box) window. While there are better filters, this one is easy to understand and easier to implement analytically than other filters. The use of this filter will be called "area sampling."

The difference between point sampling and area sampling is qualitative while the difference between area sampling and better filters is quanti-The sum of all intensities for a point tative. sampled picture will vary as the object is translated, ie. for a fine picket fence the picture can be all white in one picture and all black in the next. The sum of all intensities for an area sampled picture will be constant under translation because area sampling integrates all the intensities. The difference between area sampling and better filters is quantitative since most reasonable filters would also integrate the intensities. The difference between filtered pictures is lowered as the resolution is increased since the sum of intensities in a local area will be the same or nearly so. We cannot say this when comparing point sampling with sampling of filtered images at high resolution. A line that is much thinner than a pixel will appear dotted using point sampling and jagged using area sampling. As the resolution is increased, point sampling will still produce dots but area sampling will produce a nice line.

In order to truly filter the image before sampling, an analytic continuous solution to both the hidden-surface problem and the filter convolution must be found. The magnitude of this problem grows dramatically with the order of the filter employed. There are several approaches or simplifications that one might take to implement filtering. This paper presents an approach that uses an analytic solution for area sampling.

The problem then is to correctly integrate the intensities of all visible objects at a single pixel. This seems to require some kind of hiddensurface algorithm at every pixel!

As an example where some algorithms might fail see figure 1.



Figure 1

The correct integration would be 25% green, 25% black, 50% blue and no red. A simple minded algorithm that did not properly take into account what was hidden might distribute the intensities incorrectly and may even let some red show through. Unfortunately for computer graphics our eyes are quite capable of seeing errors like these even though they may be only one millionth of the area of the screen.

AN ALGORITHM WITH ANTI-ALIASING

In terms of the Sutherland et al criteria the algorithm presented here:

- 1. sorts all polygons in y.
- sorts all active polygons for a scanline with an x-bucket sort.
- sorts in z by searching a z-list for each entering edge.
- 4. does not use scanline-to-scanline coherence because an x-bucket is used.
- 5. Uses point-to-point coherence since order in the z-list does not change.

While this order of techniques probably has not been used before, it is not new in any spectacular way. However, care has been taken to ensure that everything necessary for anti-aliasing is available and to a much higher precision than the display.

The last step is to determine the intensity at the pixel given the z-list. An integrating algorithm is presented here that determines which pieces of polygons in the pixel are visible and then analytically calculates the average intensity.

Finding which pieces of polygons in the pixel are visible is not unlike the original hiddensurface problem except that we have two simplifications: 1) we are only interested in the sum of the intensities of each piece weighted by its area and 2) the higher level hidden-surface algorithm may have already determined the order of the polygons.

CLIPPING

Clipping is an important part of the algorithm. The clipping algorithm used was originally introduced in [6]. A variation is presented here for completeness.

When a polygon is clipped against a line it is divided into two polygons. See figure 2.



Figure 2

We can determine if a point is on side A or side B by inserting the coordinates of the point into the equation of the line:

d = ax + by + c.

If d is less than zero then the point is on side A, otherwise it is on side B. We are going to generate an A and a B polygon.

The Clipping Algorithm

- I. A polygon is a list of points Pl, P2,...Pn.
- II. Call Pn the previous point. Determine which side it is on.
- III. Loop through each point, called the current point.
 - A. If current point on A side then:
 - 1. If previous point on A side then:

Copy current point to A polygon.

2. If previous point on B side then:

Calculate intersection of line with edge formed from current point and previous point.

Copy calculated point to A and B polygons.

Copy current point to A polygon.

- B. If current point on B side then:
 - 1. If previous point on B side then:

Copy current point to B polygon.

2. If previous point on A side then:

Calculate intersection of line with edge formed from current point and previous point.

Copy calculated point to A and B polygons.

Copy current point to B polygon.

C. Call the current point the previous point.

FINDING VISIBLE SURFACES

The image space polygons handled by this algorithm are of the following form:

- 1. There is a list of vertices on the left and the right.
- The first vertex of each list is the highest in y.
- 3. Each succeeding vertex is lower in y than the preceding one.
- 4. The edge formed by the left vertices does not cross the edge of the right.

This form of polygon definition (see figure 3) is optimized for polygons with large numbers of edges. See figure 8 where the colored areas and the black line are both specified with polygons. The black lines are long thin polygons.



All other polygons in various stages of the algorithm are in the more conventional form of a list of vertices. It is assumed that an edge connects the first and last vertex. This form is necessary for the clipping algorithm presented above.

The purpose of the first level of the algorithm is to find all polygons that overlap a particular scanline and then to clip away everything that doesn't overlap it. Since the scanline has the width of one pixel we are left with a list of very narrow horizontal polygons.

The next step is to find which of those narrow polygons on the scanline overlap a particular pixel and then clip away those not over the pixel. If the closest polygon completely covers the pixel then its intensity value can be put into an array for the scanline, otherwise the list of polygons needs to be passed to the integrater.

Of course one objective is to do the above very quickly. To do so requires that we take advantage of coherence and sorting techniques to quickly reduce the number of items for consideration at each step.

The algorithm proceeds sequentially to each scanline beginning at the highest. At each scanline there is a list of active polygons that overlap that scanline. Note that a scanline is really a strip with width. At each scanline a horizontal strip is clipped off of each active polygon leaving only that part of the polygon which lies below the scanline. (See figure 4.)



Similarly at each pixel, the horizontal strip is clipped at the right edge of the pixel to determine the polygons within the square pixel area. For efficiency it is worth noting that the middle of most horizontal polygons completely covers the respective pixels. It would be wasteful to clip at each pixel in that case. We treat the middle as a solid run or segment and only need to count the pixels that it covers (see figure 5). The ends can be clipped off at the boundary of a solid segment and treated as indicated above.

The depth ordering is maintained with a sorted z-list. The first item in the list is the closest. When a new edge is encountered in the x-bucket it is entered into the z-list in order. If intersections are allowed, each item in the z-list must be checked against the incoming item over its full extent to check for possible intersection which would require splitting a polygon.

The Hidden-surface Algorithm

- I. Sort all polygons on highest y value.
- II. Initialize active polygon list to be empty. III. Repeat for each scanline:
 - A. Add polygons from y-list that enter on this scanline to active polygon list.
 - B. Initialize the x-bucket to be empty and the scanline array to background.
 - C. Loop through each polygon in active polygon list
 - 1. Clip off of each polygon the piece that lies on the current scanline. See figure 5.
 - 2. Replace polygon in list with polygon that has piece clipped off.
 - 3. If there are pixels under the piece that are completely covered, then for efficiency reasons we can break the piece into three pieces: the center solid piece and two polygons clipped off at the ends at the pixel boundaries. The two end polygons are called irregular pieces.
 - 4. The pieces are sorted into the xbucket according to the leftmost pixel covered.
 - D. Initialize the z-list to be empty.
 - E. Repeat for each pixel across the scanline: 1. Sort every entry at the current x position of the x-bucket into the zlist.
 - 2. Evaluate the z-list if not empty:
 - a. If a solid piece, get its color
 - else if an irregular piece is in front of a solid piece then find the area of the irregular piece over the pixel to weight the two colors
 - else call the pixel integrater to get color
 - b. Write the color into scanline array.

THE PIXEL INTEGRATER

Given a list of polygons in the z-list, it is necessary to find the area of each visible polygon piece in order to determine its contribution to the pixel intensity. The polygons in the z-list are in sorted z-order with the first polygon being the closest.

One of the key ideas of this algorithm is that the list of polygons can be divided into two lists with an edge of a polygon being used as the dividing line. A generalization of this idea based on using planes for dividing polygon lists is due to Ivan Sutherland [5] and in fact is part of a complete hidden-surface algorithm that he invented. This technique was used in another hidden-surface algorithm subsequently developed at Cornell[9].

Since the polygons are already in sorted order, we pick an edge of the first polygon to use as the dividing line. If this algorithm is recursively applied to both of the resulting lists of polygons then very shortly the front polygon of a list will cover all polygons behind it since everything else will have been clipped away. The area of the front polygon can then be found to weight the intensity. The sum of the weighted intensities from all the lists gives the final average intensity.



For this algorithm we make the following observations:

- 1. Since z order is implied in the list, there is no need for any z calculations. We may therefore think of the polygons as two-dimensional; they will be clipped against a line and not a vertical plane.
- 2. A pixel polygon for this algorithm is a list of vertices with implied connection of the first and last vertices.
- 3. A vertex consists of x, y, and clipflag, where clipflag is used to indicate whether or not the edge connecting that vertex and the next one has been clipped.
- 4. A pixel polygon that completely covers a pixel will be called a "solid polygon."

To prepare the z-list for the algorithm:

- 1. Each polygon will be transferred to a pixel polygon list in order until a solid polygon is reached. If there is no solid polygon, a dummy solid polygon is added with the background as its color.
- 2. All polygons are clipped to the pixel boundaries.
- 3. All edges that lie concurrent with the pixel boundaries are marked as clipped, ie. the last polygon should cover the pixel and all four edges are marked as clipped.

- 1. Consider the first polygon in the list (which is also the closest).
- 2. Look for the first unclipped edge. If there is no unclipped edge or there is only one polygon in the list then return the color of the polygon weighted by its area.
- 3. Clip all polygons in the list against the edge and put them in two lists, one for each side of the edge. Set clipflag for each edge that lies along the clipping line as it is clipped.
- 4. Reenter this algorithm for each of the two lists.
- 5. Combine and return the two results.

IMPLEMENTATION

The hidden-surface algorithm and pixel integrater were implemented by the author at the Computer Graphics Lab at the New York Institute of Technology. The polygons to be rendered were flat colored with many edges to satisfy the needs of cartoon animation. These pictures are characterized by a large number of pixels that have more than two polygons. See figure 6. The hashed polygon C covers the boundary between polygons A and B. The pixel pointed at by P has four polygons in it, three of which are visible.



Figure 6

The ability to call the pixel integrater is under user control. The user can request jagged edges with the result that the program runs approximately three times faster for complicated pictures. Full anti-aliasing is only required for quality recording. Figure 7 shows a picture with aliasing.

We have been able to use and evaluate the algorithm. See figure 8,9, and 10 which were made at 512x512 resolution with 8 bits each for red, green and blue. Movies generated using this algorithm have not shown any aliasing symptoms for the class of images created. This has made the effort worthwhile.

ACKNOWLEDGMENT

The Computer Graphics Lab was conceived and sponsored by Dr. Alexander Schure, President of New York Institute of Technology. Lance Williams provided critical reading of the first draft.

REFERENCES

- 1. Catmull, Edwin, <u>A</u> subdivision algorithm for computer display of curved surfaces, Technical report UTEC-CSs-74-133 University of Utah, 1974
- 2. Crow, Frank, The aliasing problem in Computergenerated shaded images, CACM November 1977
- Myers, Allan J., An efficient visible surface program, Ohio State University, Computer Graphics Research Group, report to NSF, July 1975
- Shoup, R.G., Some quantization effects in digitally-generated pictures, Society for Information Display, 1973 proceedings.
- Sutherland, I.E., Polygon sorting by subdivision: A solution to the hidden-surface problem, Unpublished manuscript, October 1973, Also public lecture at University of Utah 1973.
- Sutherland, I.E., and Hodgman, G.W., Reentrant polygon clipping, CACM January 1974.
- Sutherland, I.E., Sproull, R.F., and Schumacker, R.A., A characterization of ten hidden-surface algorithms, ACM Computing Surveys, March 1974.
- Hamlin, Griffith Jr., and Gear, C.W., Rasterscan hidden surface algorithm techniques, Siggraph 1977 proceedings.
- Weiler, K. and Atherton, P., Hidden surface removal using polygon area sorting, Siggraph 1977 proceedings.





figure 7

figure 8



figure 9

figure 10