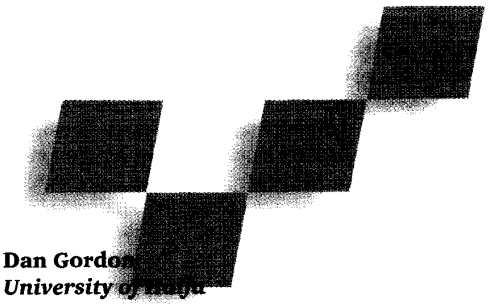
Fast Polygon Scan Conversion with Medical Applications



Dan Gordon
University of Haifa

Michael A. Peterson
Picker International

R. Anthony Reynolds
Royal Postgraduate Medical School, Hammersmith Hospital, London

This new algorithm for polygon scan conversion is very fast and can replace the standard version in any application. Special requirements from medical data sets illustrate its practicality and versatility.

The polygon scan conversion problem refers to filling the interior of a region represented by one or more polygons. The polygons are represented by the coordinates of their vertices in cyclic order. This problem shows up in many areas of computer graphics. For example, medical imaging must often identify, outline, and extract structures of interest from a set of 2D cross-sectional "slice" images of the human body. The contours that result from this segmentation process are, in fact, polygons—often with quite irregular shapes and a very large number of vertices. To render them in 3D requires filling multiple and often overlapping polygons correctly. Medical data sets are therefore ideal to demonstrate the power and flexibility of a new polygon scan conversion algorithm.

The standard approach to polygon scan conversion creates two data structures, an edge table and an active edge table (AET).[1-2] The edge table contains a bucket for each scan line. Each bucket consists of a linked list containing all the edges starting on that scan line (or straddling it for the first time). The AET contains only the active edges, that is, those that straddle the current scan line. After the algorithm creates the edge table, the AET is initialized and swept through the scene one scan line at a time. At each scan line, the AET is updated by adding newly active edges and deleting inactive ones, and the polygons are scan converted by filling pixels between alternate pairs of edges.

Creation of the edge table is an *image-space* step, since it depends on the number of scan lines existing in image space (that is, on the resolution of the target display device). In this respect, it is a device-dependent step. The algorithm we develop here uses only an *object-space* data structure instead of the edge table and is therefore less device dependent. In particular, it is suitable for both raster and vector graphics devices.

We base our new algorithm on the preliminary identification and sorting of all polygon vertices that are local minima with respect to the y coordinate. We call these vertices *critical points*. Sechrest and Greenberg[3] used a similar approach for determining the visible polygons of a 3D scene composed of planar, nonintersecting polygons. They obtained the visible polygons in object space with object-space precision. Thus, their algorithm could be used both for hidden surface removal on raster devices and for hidden line removal on vector devices. The algorithm's output was a rather complex data structure that coded the visible polygons and required further operations prior to hidden surface removal.

Our technique, initially developed by Gordon[4] independently of Sechrest and Greenberg, does not require complex data structures. Rather than using the edge table of the standard algorithm, we use the sorted local minima together with the initial input (the array or list of polygon vertices). In the sweep stage, we use a linked list for the AET as in the standard algorithm, but instead of inserting every edge of the polygon into the AET (and eventually deleting it), only edges starting at a local minimum are inserted (and deleted at a local maximum).

The resulting critical points algorithm is easy to implement, correct in its handling of special cases, and very fast. It can replace the regular algorithm in any of its various applications, including hidden surface removal. In this article, we first describe the new algorithm in detail, presenting two versions: one for vector graphics and one incorporating the changes required for the pixel model. We include some implementation details. Then we present some practical applications using medical data sets with special requirements. We conclude with a discussion of further applications and a brief complexity analysis.

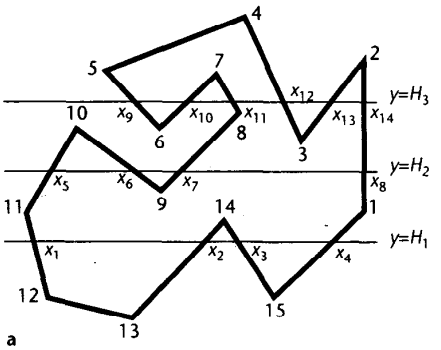
The critical points method

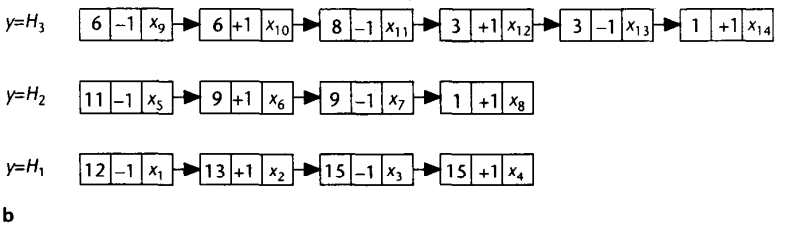
Assume that one or more polygons represent the boundary of the region to be filled. Each polygon is rep-

0272-17-16/94/$4.00 © 1994 IEEE

resented by an array of its vertices, and each vertex is given by its (x, y) coordinates. Our algorithm begins by going around each polygon and determining the set CR of critical points. A critical point is either a vertex whose y coordinate is at a local minimum or a representative of a set of consecutive vertices that together form a local minimum. Moving along the polygon in either direction from a critical point can lead only upwards until a local maximum is encountered. Figure 1a shows this for five critical points.

As in standard scan conversion, we use a linked list, the AET, whose elements follow monotonically increasing sections of the boundary, starting from the critical points (by a "section" we mean a connected part of the boundary). The AET elements represent the intersections of a horizontal line $y = H$ with the polygon boundaries; the elements are ordered by their x coordinates. The AET performs two functions. First, it holds the x values of the intersections of the line $y = H$ with the polygons. Second, it represents the polygon's boundary sections as they move away from the critical points (see Figure 1a and b). At each horizontal level, the algorithm fills the polygon by traversing the AET and filling between alternate pairs of AET elements.





1 (a) A polygon with 15 vertices. The critical points (in increasing order of Y) are 13, 15, 9, 3, 6. Intersections with the scan lines $y = H_1$, $y = H_2$, and $y = H_3$ are shown. (b) The AET at $y = H_1$, $y = H_2$, and $y = H_3$. Each element has three fields: IND, DIR, and XS.

Filling a single polygon in vector graphics

It is customary to distinguish between vector and raster graphics.[1] For our purposes, the distinction is that we can ignore the thickness of the boundary in the former, whereas lines in the latter are composed of discrete pixels that might have to be filled.

We first present the critical points method for filling a single polygon P, given by two arrays of its vertices

$$X[1], Y[1], \ldots, X[n], Y[n].$$

The critical points method also works when the vertices are represented by a two-way linked list. (We later extend the method to fill several regions, each defined by several polygons.) We assume that the edges of P are the line segments

$$(\,(X[i], Y[i]), (X[i+1], Y[i+1])\,).$$

The arrays are considered in circular order, that is, $i + 1 = 1$ when $i = n$, and $i - 1 = n$ when $i = 1$.

Our first procedure, DETCR, determines the set CR of critical points by going around P. It is implemented as shown in Figure 2. After DETCR, we initialize the AET to empty and determine the first horizontal line $y = H_1$. The AET is updated to "meet" $y = H_1$ as follows: From every critical point whose y value is less than H_1, each of the two polygon secions ascending from it is followed upwards until either it turns down before reaching H_1 or it passes H_1. In the first case, the section is abandoned; in the second case, a new element is added to the AET.

The new element contains the following information fields: IND is the index of the highest of the vertices on the section that is less than or equal to H_1, DIR = ±1 is

```
Procedure DETCR
/* CAND is a Boolean variable that indicates
whether the current vertex is a "candidate" for a
critical point. CAND is set True whenever Y decreas-
es but is unchanged when the Y values are equal.
Whenever Y increases and CAND is true, we have a
critical point. */

begin
  CAND := False; set CR to empty;

  for I := 1 to n do
    if Y[I] > Y[I+1]
      then /* I+1 is a candidate */ CAND := True
    else if Y[I] < Y[I+1] and CAND
          then /* I is a critical point *
          begin add I to CR; CAND := False end;

  if CAND
    then /* 1 is a candidate */

    for I := 1 to n do
      if Y[I] > Y[I+1]
        then exit /* from loop */
      else if Y[I] < Y[I+1]
            then begin add I to CR; exit end;
```

2 Procedure DETCR.

```
Procedure MOVEUP(H,J,DI,S);
/* y = H is a horizontal line; J is the current index
from which to start the search. J will (possibly)
change to become the index of the highest vertex with
y ≤ H. If the polygon section turns down before
reaching H, J is set to 0. DI = ±1 and indicates the
direction of advance from J along the polygon arrays
X and Y. S will become the x coordinate of the inter-
section point. MOVEUP assumes Y[J] ≤ H. */

begin

   J1 := J + DI /*index of next vertex on polygon*/;

   while Y[J1]≤ H do
     if  Y[J1] < Y[J]
       then /* polygon turns down */
       begin J:=0; return; end
     else /* move up */
       begin J:J1; J1:=J+DI; end;

     S:= x coordinate of intersection of edge
       ( (X[J], Y[J], (X{J1], Y[J1]) ) with y=H;

end /* MOVEUP */
```

3 Procedure MOVEUP.

the direction of advance along the arrays *X* and *Y* from the critical point; *XS* is the *x* coordinate of the intersection of the polygon section with $y = H_1$. It might also be desirable to store the inverse of the current edge's slope.[1]

The AET is maintained in nondecreasing order, sorted by the values of *XS*. The polygon is filled at level H_1 by traversing the AET and drawing lines from (x_1, H_1) to (x_2, H_1), from (x_3, H_1) to (x_4, H_1), and so on (where x_i is the *XS* of the AET's *i*th element).

The AET is updated to meet the next horizontal line $y = H_2$ as follows: From every point currently on the AET, the polygon boundary is followed upwards in the direction specified by DIR until either it turns down before reaching H_2 or it passes H_2. In the first case, the element is deleted from the AET; in the second case, the information in the element is updated.

If the initial input contains intersecting edges, then at this point the algorithm must check the AET and, if necessary, reorder it by nondecreasing *XS* values. After that, all critical points less than H_2 are handled the same as those previously described for H_1.

The procedure MOVEUP determines whether (and if so, where) a section of the polygon boundary ascends to meet a horizontal line $y = H$. It is implemented as shown in Figure 3.

We can now list the entire program for the critical points method, as shown in Figure 4. For this program, we assume that *c* is the number of critical points and that CR is implemented as an array of the indices of the critical points, sorted into nondecreasing order by *Y*. We assume that appropriate procedures INSERT and DELETE are available for the AET and that a procedure FILL is available for drawing horizontal lines between alternate pairs of AET elements. H1, H2, and HI are respectively the first, last, and incremental value of the horizontal lines to be drawn. The procedure REORDER sorts the AET elements into nondecreasing values of *XS*. It can be skipped if there are no intersecting edges (see the sidebar "Some implementation details").

```
Program Critical Points

begin

   DETCR; initialize AET to empty;
   M := 1 /* 1 ≤ M ≤ c is index of current critical
             point */;

   for H := H1 to H2 step HI do begin
     for each AET element K do begin
       /* update K to meet y=H */
       J := K.IND; DI := K.DIR; MOVEUP(H,J,DI,S);
       if J = 0
         then DELETE(K)
       else begin K.IND := J; K.XS := S end;
     end /* for */;

     REORDER;
     /* insert new critical points */
     while (M ≤ c and Y[CR[M]] ≤ H) do begin
       /* both directions from critical point */
       for DI := -1 to +1 step 2 do begin
         J := CR[M]; MOVEUP(H,J,DI,S);
         if J > 0
           then INSERT(J, DI, S);
       end /* for */;
       M := M + 1;
     end /* while */;
     FILL;
   end /* for */;
end /* CP */.
```

4 Complete program for the critical points method.

The critical points method is very simple to implement. The program length is about the same as a conventional method based on an edge table, but it uses much less data space and runs much faster.

Filling multiple polygons in vector graphics

We wish to extend the critical points method to "color" regions defined by Boolean operations (union, intersection, and so forth) on multiple polygons, allowing the polygons to intersect themselves and each other if need be. This

allows us to fill disconnected regions, regions with holes, and arbitrary nestings of regions. As in the standard algorithm, we maintain bitvectors holding the necessary information. The following details are standard practice. We include them here because they support special requirements that our medical applications make on the raster version.

Assume we have a primitive command of the form DRAW(g, x, y) whose action is to draw a line of color (or gray-level) g from the current position to vertex (x, y), where $g = 0$ is a background color or a MOVE operation). Suppose we have a set of polygons $\{P_0, \ldots, P_{m-1}\}$ and a set of "coloring rules" of the form $B_i \leftarrow g_i$, $1 \leq i \leq k$, where each B_i is a Boolean expression in disjunctive normal form over the variables $p_0, \ldots, p_{m-1}$. (We can replace a conjunctive assignment of the form $B_1 \vee B_2 \leftarrow g_i$ with the two assignments $B_1 \leftarrow g_i$, $B_2 \leftarrow g_i$). The problem is to fill the region defined by B_i with color g_i. For example, if $B_i = p_3 \vee (p_5 \wedge \sim p_8)$ then the region $P_3 \cup (P_5 \cap \sim P_8)$ should be colored with g_i (where $\sim P$ is the complement of polygon P).

For each i, we construct a string S_i of length m over the alphabet $\{0, 1, d\}$ so that the jth character of S_i is

1 if p_j appears in B_i
0 if $\sim p_j$ appears in B_i
d otherwise

where d is a "don't care" symbol.

S_i can be implemented with two bitvectors, each with m bits. FILL initializes another bitvector MEM to 0 and, when it meets a polygon P_j in traversing the AET, sets MEM := MEM XOR 2^j (that is, it toggles MEM_j, which is the jth bit of MEM). For each interval between consecutive AET elements, the interval is in P_j if and only if $\text{MEM}_j = 1$, so the algorithm checks whether MEM matches any S_i (ignoring the "don't care" symbol d). If MEM does not match any S_i, the interval is colored 0. If MEM matches only one S_i, the interval is colored g_i. If MEM matches more than one S_i, then we have conflicting coloring instructions. We resolve this conflict on a priority basis by arranging every S_i in decreasing order of priority, then coloring according to the first match. An alternative is to set up a color lookup table with an entry for each possible value of MEM (2^m entries).

Raster version and special problems

We can extend CP to raster graphics in a straightforward manner by setting the step size HI to 1 and using a raster version of FILL. This will take care of all the usual applications and extensions of scan conversion. However, for our medical applications, we need to consider every pixel intersected by a polygon edge as belonging to the region to be filled (such pixels will be called boundary pixels). For convenience, we assign integer coordinates (I, J) to the lower left corner of a pixel, with the pixel itself consisting of $\{(x, y) \mid I \leq x < I + 1, J \leq y < J + 1\}$. This causes two difficulties, described below and illustrated in Figure 5.

1. When coloring a boundary pixel, we can use the bitvector MEM as in the previous section when entering a polygon, but on leaving the polygon MEM changes value before coloring the pixel. Therefore, the last pixel does not get colored correctly.
2. A polygon boundary can enter a raster scan line and oscillate there, going to and fro several times before leaving it. In that case, all the pixels on the scan line encountered by the boundary belong to the region and must be colored accordingly.

We can solve these problems as follows: When updating the AET to meet a new scan line, the polygon is followed from each AET element into the scan line until it

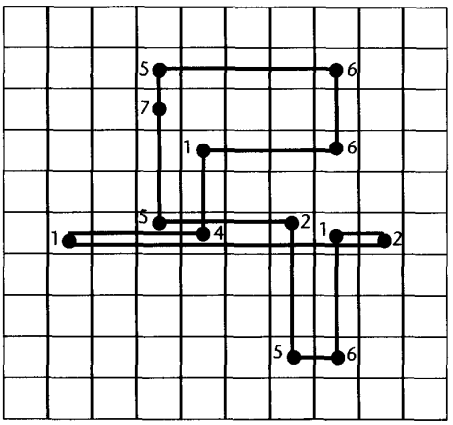
Some implementation details

1. When there are no intersecting edges, it is possible to avoid reordering the AET. Every critical point causes two elements to be inserted in the AET. If these have identical values for XS, then they are ordered along the AET according to increasing values of the inverses of their slopes.

2. In the pixel model, boundaries can be represented efficiently by chain codes.[1,5] This format gives the coordinates of a starting pixel and a string of direction numbers in the range 0 to 7 (indicating which pixel to advance to next). We can perform scan conversion on the chain code as follows: Store the true coordinates of every critical point; the coordinates of the next pixel in either direction can always be found from the chain code. As the AET advances, the adjacent pixels are found from the coordinates of the AET pixels and the chain code.

3. An alternative to the chain code is a sequence of pixel edges (a *pixel edge* is the line segment separating two adjacent pixels). The advantage of this method is that it lets us use the simpler version of CP presented under "Filling multiple polygons in vector graphics." We can also represent such boundaries by a variable-length chain code (for example, 0 means the next pixel edge in the same direction as the current one; 10 means turn left; 11 means turn right).

4. Inconsistencies must be avoided in finding the intersections of the scan lines with the polygon edges. Note how this is coded in MOVEUP: We first determine that an intersection must exist, then we calculate its location. We can always calculate the latter with sufficient accuracy even though the floating-point arithmetic is not perfectly accurate.



5 Polygon scan conversion in the raster graphics case. The region to be filled includes the boundary pixels (that is, the pixels through which the heavy line passes). The type codes of some AET pixels are illustrated.

```
1. Initialize MEMj := False,
   BOUNDj := 0, for 0 ≤ j ≤ m - 1;
2. Let K1 := first AET element;
3. Let IX := [K1.XS];
   /* integer x coordinate of pixel */
4. Let E := { K | K is an AET element
   such that [K.XS] = IX };
   /* Note that K1 belongs to E. Also,
   because the AET is ordered by the
   x coordinate, the other elements
   of E will be bunched together on
   the AET following K1. */
5. For every K in E do:
   5a. if (K.type = 1 or 5) then
       BOUNDK.p := BOUNDK.p + 1;
   5b. if (K.type ≥ 4) then MEMK.p :=
       NOT MEMK.p;
6. Color pixel (IX, J) according to
   the values of {MEMj OR BOUNDj | 0 ≤
   j ≤ m-1}
   (Consider BOUNDj as True if it is
   positive);
7. For every K in E do:
   if (K.type = 2 or 6) then
   BOUNDK.p := BOUNDK.p - 1;
8. Let K2 be the first AET element
   such that [K2.XS] > IX.
   If there is no such K2 then end
   /* FILL */.
9. For every integer I,
   IX < I < [K2.XS], color pixel
   (I,J) according to the current
   values of {MEMj OR BOUNDj | 0 ≤ j ≤
   m-1}.
   /* Note that this set may differ
   from what it was in step 6,
   because of possible changes in
   step 7. Note also the strict
   inequalities surrounding I. In
   step 9, we are only coloring pix-
   els that are not AET pixels. AET
   pixels were colored in step 6. */
10. Set K1 := K2 and goto step 3.
```

6 Procedure FILL for raster graphics.

leaves the scan line (turning upwards or downwards as before). Thus we know the minimum and maximum x values of the polygon section within the scan line.

We also need to know the last pixel on the scan line to be visited by the polygon section, since this will be used for the next update. Three different types of AET elements can be defined:

1. An element of type 1 represents the minimum x value.
2. An element of type 2 represents the maximum x value.
3. An element of type 4 represents the last pixel to be visited.

The same AET elements may have several types, in which case the type numbers are added (see Figure 5).

Each existing AET element can thus generate up to two new AET elements. These new elements are brought to their correct position in the AET by reordering the AET according to the x coordinate as before. If K is an AET element, then pixel $([K.XS], J)$ is called an AET pixel, where $y = J$ is the current scan line and $[K.XS]$ is the integer value of $K.XS$. Several consecutive elements can have the same (integer) x value and thus correspond to the same AET pixel. This occurs when boundaries overlap or intersect at a particular pixel.

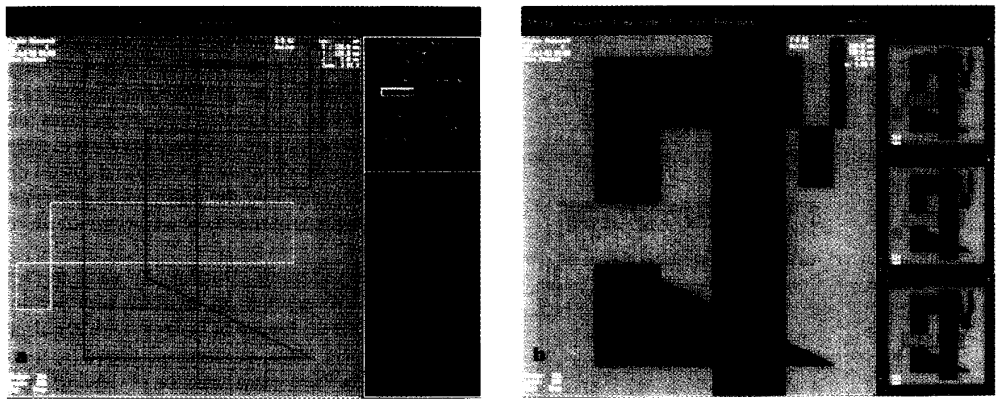
Assume we are given the polygons $P_0, \ldots, P_{m-1}$ and coloring rules $B_i \leftarrow g_i$ as before. We need a bitvector MEM as before, and for each polygon P_j, we also need a counter $BOUND_j$. MEM_j (the jth bit of MEM) is a Boolean variable with values True or False. $BOUND_j$ counts the number of horizontal stretches of the boundary of P_j for the scan line we are on (a horizontal stretch is the portion of the scan line between elements of types 1 and 2). Assume that every AET element K has fields $K.p$ and $K.type$, holding the polygon number and coding the element type, respectively. Let $y = J$ be the current scan line. The FILL procedure operates as shown in Figure 6. It deletes elements with type codes 1 or 2 after their information is used, since they are not needed for the next update.

Note that when a critical point is entered into a scan line, it contributes two elements and will therefore cause the statement MEM_j := NOT MEM_j to be executed twice. This is equivalent to ensuring that a scan line meets a critical point twice or not at all. The same occurs at a scan line that has a local maximum: Two ascending branches of the polygon meet and both create elements of type 4 before turning down. Therefore, the statement MEM_j := NOT MEM_j will again be executed twice. Note that a single horizontal stretch at the maximum is counted twice by $BOUND_j$. This is not a problem because $BOUND_j$ will be incremented by two and later decremented by two at the very same pixels.

Medical applications

Segmentation is the process of identifying, outlining, and extracting structures of interest. It plays an important role in generating 3D reconstructions from medical data sets[5] and making quantitative volume measurements of 3D anatomical structures.[6] There are basically two approaches to segmentation: *boundary-based* methods, which identify the boundary of the structure directly, and *region-based* methods, which identify the pixels or voxels within the region occupied by the structure.[7] In the former case, the output is usually the set of "contours" delineating the intersection of the boundary with each input slice. In the latter case, the output is usually a copy of the input slices in which voxels are "tagged" if they belong to the region represented by the structure.[8] A fast, accurate method of scan conversion allows for conversion between the contour and the tagged voxel representation.

A body organ is typically a single, connected structure; however, it can bifurcate, or branch, and therefore intersect a given slice in more than one polygon. The polygons resulting from these organ systems can over-



7 (a) Arbitrary regions outlined using the manual segmentation option and (b) the same regions filled on a priority basis.

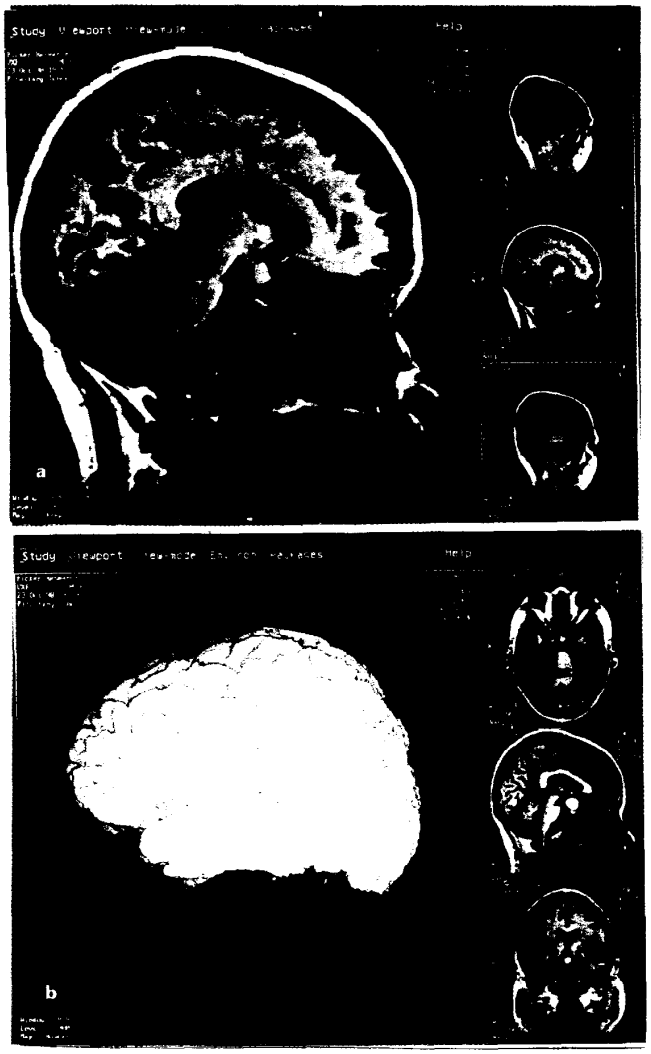
lap in several complex ways. For instance, a tumor can overlap several healthy organs. We can model structures such as blood vessels as thick-walled hollow tubes and represent them by overlapping concentric polygons. The critical points method handles all these cases correctly and efficiently, since it can fill multiple polygons simultaneously and implement Boolean operations on polygons via the coloring rules.

The VoxelQ (formerly known as the VoxelScope), a medical imaging workstation produced by Picker International, illustrates the use of the critical points algorithm to generate tagged voxels from contours. The VoxelQ supports a collection of semiautomatic contouring tools for segmentation. With these tools, users can interactively trace, erase, and stretch contours around structures of interest, as well as automatically trace areas of a specified gray-level range. After a collection of slices has been contoured, the voxels inside the contours are labeled, or tagged, using the critical points algorithm. The tag is then used to selectively render or count voxels (for volume measurements) during later processing.

CP processes all the contours from all the structures on each slice in a single pass. In areas of overlap, it uses a simple priority method to overcome tagging conflicts. Structures of a higher priority are tagged preferentially, thus resolving the overlapping contour problem, as shown in Figure 7.

8 (a) A CT image of the heart showing tagged aortic lumen and (b) a cutaway 3D view of the structures tagged in (a).

Figures 8 and 9 illustrate the use of CP to tag structures from contours in medical data sets. Figure 8a is a computed tomography (CT) image of the heart, showing a bifurcated descending aorta. The true aor-



9 (a) An MR image of the brain segmented and tagged to show a pituitary tumor in red, the cerebrum in yellow, and the cerebellum in blue, and (b) a 3D view of the cerebrum and cerebellum tagged in (a).

tic lumen is tagged in red and the false aortic lumen is tagged in yellow. Figure 8b shows a cutaway 3D view of the tagged lumens, illustrating their relationship in space. Figure 9a shows the contoured and tagged results from a series of overlapping contours representing a segmentation of a magnetic resonance (MR) image of the brain. Note the pituitary tumor in red. Figure 9b presents a 3D view of the segmented brain, showing the temporal lobe in yellow and the cerebellum in blue. These examples illustrate the usefulness and robustness of our new scan conversion algorithm.

Discussion and analysis

Conventional scan conversion algorithms are frequently used for hidden surface removal, and our algorithm can be used for that purpose also. In this case, the arrays of critical points and polygon vertices take the place of the edge table.[1-3] Some rotations in space (for example, rotations about a vertical axis) do not alter CR, so preprocessing the polygons to determine CR just once might help speed the surface rendering of rotating bodies.

Voxel arrays represent 3D scenes in many applications. Segmentation produces contours representing the intersections of objects of interest with regularly spaced transverse slices, as discussed above for medical applications. For a given direction of viewing, these slices have a known priority. At display time, the contours can be projected onto the screen first and scan converted later, using the priority rules to accomplish hidden surface removal appropriately.

For the analysis, we compare the time and space requirements of CP with standard scan-conversion algorithms for the raster model only, although results are similar for the vector model. In this comparison, let n be the total number of polygon edges, c the number of critical points, and h the number of scan lines. For $1 \le i \le h$, let n_i be the number of edges starting at (or straddling for the first time) scan line i.

Both CP and existing algorithms have the following properties in common: Space for the initial input is $O(n)$, and the time needed to fill pixels between AET elements is simply proportional to the total number of pixels that are set.

In comparing aspects of the algorithms that differ, we found that to build the edge table, a standard algorithm takes $O(n)$ time for the bucket sort and an additional $O(\sum_{i=1}^{h} n_i \log n_i)$ to sort the edges within the buckets. The AET contains $O(c)$ elements at any one time, but all n elements are inserted into and deleted from the AET before the algorithm completes. The merge operation at every scan line (between the AET and the ith bucket) takes $O(n_i + c)$ time per scan line. Summing this over h scan lines gives $O(n + hc)$ for the total merge time. The total deletion time is $O(n)$ since the edge table is not involved. The edge table requires $O(n + h)$ space.

For CP, the time to find and sort the critical points is clearly $O(n + c \log c)$, and the space requirement is $O(c)$. At the sweep stage, $2c$ edges starting at the critical points are inserted into the AET (whose size is $O(c)$) and $2c$ edges are removed, so the time for this is $O(c^2)$.

Table 1 summarizes these differences, which are particularly relevant when c is small compared to n or h. Note that the time in Table 1 does not include the time to fill pixels. In practice, the savings can be quite substantial (consider filling a single convex polygon, where $c = 1$, or a smooth heart-shaped figure, where $c = 2$, while n and h can run into the hundreds). To find the worst-case behavior, substitute n for c, because in the worst case, $c = n/2$.

The critical points algorithm is faster than the standard scan con-

Table 1. Summary of performance differences between a standard algorithm and the critical points method.

	Time	Extra space
Standard algorithm	$O(hc + \sum_{i=1}^{h} (n_i \log n_i))$	$O(n + h)$
Critical points method	$O(c^2)$	$O(c)$

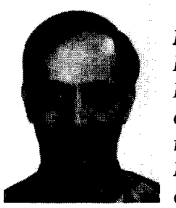
version method and uses less space. It is also versatile in handling special requirements, as the examples from medical graphics show. Because it can replace the standard method in any application—from single-polygon scan conversion to hidden surface removal—we expect that practitioners in computer graphics will gradually come to prefer it. ■

Acknowledgments

Algorithm CP was implemented on the VoxelScope II and optimized for medical applications while two of the authors, M.A. Peterson and R.A. Reynolds, were with Dynamic Digital Displays (now a subsidiary of Picker International). We are grateful to our colleagues for many helpful contributions.

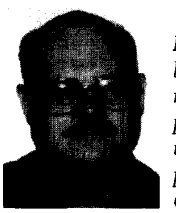
References

1. J.D. Foley et al., *Computer Graphics Principles and Practice*, 2nd ed., Addison-Wesley, Reading, 1990.
2. F.S. Hill, Jr., *Computer Graphics*, Macmillan, New York, 1990.
3. S. Sechrest and D.P. Greenberg, "A Visible Polygon Reconstruction Algorithm," *ACM Trans. Graphics*, Vol. 1, No. 1, Jan. 1982, pp. 25-42.
4. D. Gordon, "Scan-Conversion Based on a Concept of Critical Points," Tech. Report No. CSD83-1, Dept. of Computer Studies, Univ. of Haifa, Israel, May 1983.
5. J.K. Udupa, "Interactive Segmentation and Boundary Surface Formation for 3D Digital Images," *Computer Graphics and Image Processing*, Vol. 18, 1982, pp. 213-235.
6. D.N. Kennedy, P.A. Filipek, and V.S. Caviness, "Anatomic Segmentation and Volumetric Calculations in Nuclear Magnetic Resonance Imaging," *IEEE Trans. Medical Imaging*, Vol. 8, No. 7, March 1989, pp. 1-7.
7. K.S. Fu and J.K. Mui, "A Survey on Image Segmentation," *Pattern Recognition*, Vol. 13, 1981, pp. 3-16.
8. R.A. Reynolds, E.D. Wyatt, and M.A. Peterson, "Segmentation for 3D Display," *Colloquium on Image Processing in Medicine*, Digest No. 1991/084, IEE, London, April 1991, pp. 8/1-8/7.



Dan Gordon *received the BSc and MSc degrees in mathematics from the Hebrew University and the DSc degree in mathematics from the Technion-Israel Institute of Technology. He is currently a senior lecturer of computer science at the University of Haifa and has taught previously at Texas A&M and other universities. His research interests include computer graphics, data structures and algorithms, and reconfigurable processor arrays. Gordon is a member of the ACM and EATCS (European Association for Theoretical Computer Science).*

Mike Peterson *is a staff scientist in the Visualization Group of the Computed Tomography Division at Picker International, currently working on the development of visualization software for a variety of clinical applications to increase the diagnostic efficacy of computed tomography. He received his BS degree in electrical engineering from the University of Illinois in Urbana-Champaign and his MS degree in biomedical engineering from Case Western Reserve University in Cleveland, Ohio. Peterson is a member of IEEE, Eta Kappa Nu, and Tau Beta Pi.*



R. Anthony Reynolds *is a senior lecturer in the Department of Diagnostic Radiology, Hammersmith Hospital, London, where he is part of a team implementing a large-scale hospital-wide Picture Archiving and Communications System (PACS). After taking a BA and MSc in physics and a PhD in computer and information science, he cofounded Dynamic Digital Displays, a 3D workstation company that is now a subsidiary of Picker International. He is interested in all aspects of computerized medical imaging.*

Readers may contact Gordon at Dept. of Mathematics and Computer Science, University of Haifa, Haifa 31905, Israel, e-mail gordon@mathcs2. haif.ac.il. Readers can contact Reynolds at e-mail areynold@rpms.ac.uk.