
2011 INTRODUCTION TO GRAPHICS NOTES

ADDITIONAL NOTES AND EXERCISES

JAN KAUTZ

DEPARTMENT OF COMPUTER SCIENCE, UNIVERSITY COLLEGE LONDON

LECTURE 9: SCAN CONVERSION

EDGE TABLES

The following algorithm makes use of scan-line coherence, and is based on the approach of [APPE 68; BOUK 70; WATK 70]. Consider an edge (x_1, y_1) to (x_2, y_2) with the labels chosen so that the y -value of the first point is always less than that of the second ($y_1 < y_2$). *Horizontal edges are ignored throughout.* This edge is represented by a structure:

```
struct Edge {int y2; float x1; float Dx;}
with Dx = dx/dy
where dx = x2-x1, dy = y2-y1.
```

All edges are bucket sorted according to their smaller y -values (y_1). This sorted sequence is called the Edge Table (ET). The ET therefore contains all the edges in the original polygon, but now ordered according to the height at which they start (i.e., the lower y -value). Therefore, associated with every bucket in ET there will be a set of edges which have their lower y -value equal to the bucket index.

More specifically, represent ET as an array with bounds 0 to the maximum possible scan-line on the display (YMAX). Each array entry is a sequence of triples representing the edges, using the Edge structure. In order to make the Edge Table,

```
Initialise ET[i] =  $\emptyset$  (the empty sequence), for  $i = 0$  to YMAX;
```

For each edge (x_1, y_1) to (x_2, y_2) ensure that $y_2 > y_1$ (which may involve swapping values, and ignoring horizontal edges):

```
append(ET[y1], {y2, x1, Dx})
```

which puts the new triple at the end of this list (or alternatively it could be put at the head, or even insert sorted with respect to x_1 into the list). Also during construction of the ET, keep track of the maximum and minimum y -vertices (y_{min} and y_{max}).

Next, process the ET as follows, constructing a new object AET (Active Edge Table) which is of type 'sequence of Edges' (i.e., the same type as any particular ET[i]).

```
processET(void) {
    AET =  $\emptyset$ ;
    for (i = ymin; i <= ymax) {
        update(AET, i);
        append(AET, ET[i]);
    }
    //initialise to the empty sequence
    // for each scan-line
    //delete from AET entries with  $y=i$ ,
    //and compute  $x_1 += Dx$  for remainder
    //join ET[i] to the AET
```

```

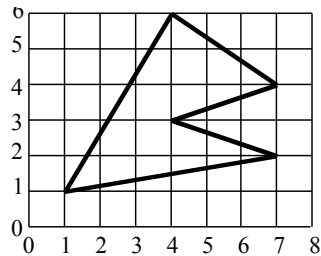
        sort(AET);                //sort the entries by x1
        JoinLines(AET,i);        //join horizontal lines between
                                //pairs of x1, at height i
    }
}

```

The Active Edge Table is the sequence of edges that intersect the current scan-line sorted according to increasing x_1 value (the intersections between the polygon edges and the scan-line). Initially it is empty - any scan-line below the lowest vertex (at $y=y_{\min}$) of the polygon obviously intersects none of its edges. The update function deletes any entries in the AET which have their y -coordinate equal to the current scan-line at height i (since these edges have now been completely processed and are below the current scan-line). For the edges that are not deleted, it increments their x_1 -coordinate by Dx , implementing the scan-line coherence idea shown in Figure 2.4. These x_1 -coordinates are the x -coordinates of the intersections of the active edges with the current scan-line. The append function appends to the AET those edges which start at height i . The sort function now sorts the edges in the AET according to their x_1 -values. The reason for this is that the next function, `JoinLines`, must join horizontal spans determined by pairs of x_1 values. This relies on the odd-even inside test to assure correctness. (It is assumed that update has no effect when the AET is empty).

EXERCISES

1. What simplifications can be made to the AET algorithm if the polygon is a) convex b) triangular?
2. Work through the AET for the following polygon:



3. Does the AET algorithm work for non-simple polygons? Can it be fixed with suitable modifications?
4. How would you do anti-aliasing when using AET?