Computer Graphics 2004p3q8

Though this is a single part question, it actually splits into a number of parts:

- specifying the polygon data set
- projecting a polygon to 2D
- clipping a polygon to the screen boundaries
- drawing a 2D clipped polygon
- how to handle depth values so that the correct polygons appear on top (this is best handled by the z-buffer modification of the 2D polygon drawing algorithm but can also be done by depth-sorting, at the expense of a longer answer).

Model Answer

Polygons, P_k are each specified as an ordered set of 3D vertices

$P^k = (V_1^k, V_2^k, V_3^k, \ldots, V_{n_k}^k)$ $\qquad V_i^k = (x_i^k, y_i^k, z_i^k)$

Each polygon has a colour, C^k

Step ① – project polygons to 2D, preserving depth information for use later:

~~for ex~~ Assume that the eye (camera) is at the origin, that the screen is parallel to the xy plane, of size $2a \times 2b$, centred on the z-axis at $(0,0,d)$. This makes projection easy and matches what is taught in the notes. [Some students may make the assumption that the eye and screen are in an arbitrary location, which makes this a lot harder and is not necessary.]

The algorithm for create projected polygon $\hat{P}^k$ from polygon P^k is:

for each polygon, P^k
  for each vertex, V_i^k
    let $\hat{V}_i^k = \left(x_i^k \cdot \frac{d}{z_i^k}, y_i^k \cdot \frac{d}{z_i^k}, z_i^k \right)$

~~If $z_i^k \leq 0$ then we need to do something clever as these are vertices behind the eyepoint.~~

~~Generally, we set some $\varepsilon > 0$ and say:~~

If $z_i^k = 0$, then set z_i^k to be some small value before projecting. Such vertices will be clipped out in the next step.

Step 2 – clipping

we need to clip the polygon against six clipping planes:

$x = -a$; $x = +a$

$y = -b$; $y = +b$

$z = z_{front}$; $z = z_{back}$ ← these values are > 0 and defined by the programmer or user as appropriate for the application.

~~the algorithm to clip polygon $\hat{P}$ against an edge is:~~

~~let $\hat{V}_{prev} = \hat{V}_n$ (the last polygon in the list)~~
~~for each $\hat{V}_i$ from $i=1$ to $i=n$~~
~~IN_{prev} = (is $\hat{V}_{prev}$ on the "IN" side of~~

The algorithm to clip a polygon against a clipping plane uses the concept of IN and OUT. A vertex is IN if it is on the side of the clipping plane which contains the clip volume; otherwise it is OUT.

A new list of vertices is produced by marching around the polygon vertices and outputting new vertices as shown below:

let $\hat{V}_{prev}$ = the final vertex in the polygon list, $\hat{V}_n$
find IN_{prev} = is $\hat{V}_{prev}$ IN?
for each vertex $\hat{V}_i$ from $i=1$ to $i=n$
    find IN_i = is $\hat{V}_i$ IN?
    there are four cases:
        $IN_{prev} \wedge IN_i \longrightarrow$ output $\hat{V}_i$
        $IN_{prev} \wedge \neg IN_i \longrightarrow$ output interpolated point ~~between on the plane~~ where the line $\overline{\hat{V}_{prev}\hat{V}_i}$ intersects the clip plane
        $\neg IN_{prev} \wedge IN_i \longrightarrow$ output the interpolated point (as above) followed by $\hat{V}_i$
        $\neg IN_{prev} \wedge \neg IN_i \longrightarrow$ output nothing
    let $\hat{V}_{prev} = \hat{V}_i$

Pass the ~~new~~ output from clipping against one plane as the input to the next clipping.
After clipping against all six planes, we have the clipped polygon.

When doing the interpolation of $\hat{P} = (\hat{x}, \hat{y}, z)$ values you must interpolate $\hat{x}$ and $\hat{y}$ normally, but interpolate $\frac{1}{z}$ values.

Step 3 – drawing the polygons ~~(overview)~~.

Use Z-buffer algorithm

For each pixel $[i,j]$

Set $\text{depth}[i,j] = \infty$ and $\text{colour}[i,j] = \text{black}$

For each clipped polygon, $\hat{P}$,

For each pixel in the polygon

```
if z[i,j] < depth [i,j] {
    depth [i,j] = z [i,j];
    colour [i,j] = colour of P̂
}
```

How we find $z[i,j]$ and "each pixel in the polygon" is the final algorithm.

Standard 2D polygon scan conversion.

~~Sort all vertices $\hat{V}_i$ into order on $\hat{y}_i$ and store in an Edge List (EL)~~

Create an Edge List (EL) containing all edges in the polygon, sorted on their lowest $\hat{y}$ value.

Create an empty Active Edge List (AEL)

Set y to be the y-value of the row of pixels just below the lowest $\hat{y}$ value in the EL.

Loop {

Increment y.

[Remove from AEL any edges whose highest $\hat{y}$ value is less than y

[Move to AEL from EL any edges where lowest $\hat{y}$ value is less than y

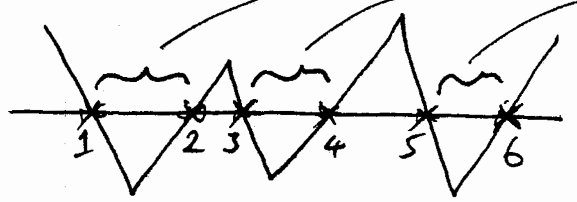
these must be the other way round

Find intersection point (in x-y) of all edges in AEL with scan line y

Sort these points in increasing order of $\hat{x}$.

Pixels "in the polygon" are those between pairs of points in this sorted list

e.g.



these three spans, between points 1&2, 3&4, and 5&6 are in the polygon

} Until AEL is empty

z values can be obtained by linear interpolation in $\frac{1}{z}$

[Some students may mention that there are special cases when a vertex, $\hat{v} = (\hat{x}, \hat{y}, z)$ has $\hat{y}$ exactly equal to a scan-line y]

2004 p3q8 Marking Scheme

specify polygon as ordered list of vertices 1 | 1

PROJECTION

project to 2D by $\hat{x} = x \cdot \frac{d}{z}$ $\hat{y} = y \cdot \frac{d}{z}$ 1

state necessary assumptions to make this work (eye at (0,0,0), screen centre at (0,0,d), screen parallel to xy plane) 1

mention the "divide by zero" special case 1 | 3

~~remember the original z-values (which are used later)~~

CLIPPING

clip against SIX planes 1

specify what these planes are 1

correct output for each of the four cases in the algorithm 4

mention that the algorithm is run on each clip plane in turn with output from one stage going to next 1

* mention that z must be interpolated in $\frac{1}{z}$ space 1 | 8

DRAWING

initialise depth and colour buffers 1

for each pixel check z against current depth 1

replace colour and depth if z < depth 1

create <u>edge</u> list 1

correct edges put into Active Edge List

correct edges taken out of Active Edge List } 1

find intersection points between scan line and all edges in Active Edge List 1

sort into increasing order on x 1

[* ~~remember to mention that z can be interpolated~~]

fill in between pairs of intersection points 1 | 8

8/20