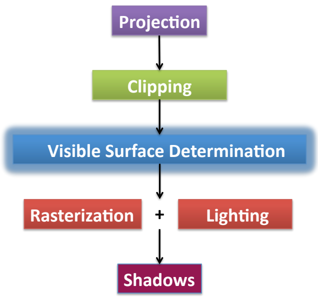
Coping with Depth

Jan Kautz

Rasterization Pipeline Reminder



Introduction

- One way is with ray tracing:
  - Trace a ray from each pixel, and intersect it with the polygons of the scene
  - Problem: This is expensive
- We are looking for methods
  - that project the polygons on the window
  - then decisions on the pixel colour are made in the image space

Overview

- AET and depth
- Z-Buffering
- Z-Correct Depth Interpolation

TODO:

- Add painter's algorithm!

Coping with depth

- When all polygons are projected on the screen, they may overlap
- Need to know which one is in front of the others
- There are several methods to treat this
  - Scan-line depth buffering
  - Z-buffer
  - Trade-offs
- Z stands for the depth

Scan-line depth buffering: Extending the AET

- Easy enough if polygons do NOT intersect
- Put all polygon edges into ET with extra depth information and proceed as before except …
- … now consider overlapping ranges of edges

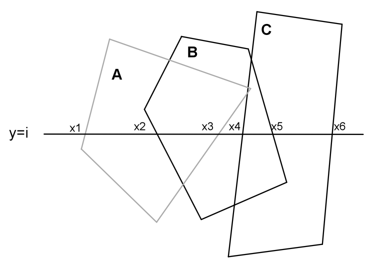
Representation of the edges

- Each edge is represented by four elements now:

$$(y_2, x_1, \frac{dx}{dy}, pt)$$

- pt is a pointer to information about the polygon (plane equation, shading, …)

AET Example



Polygons A,B,C are such that A is in front of B which is in front of C.

AET Example Notes

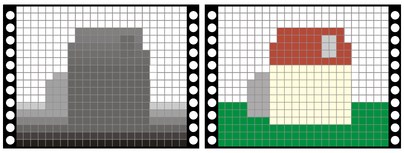
- Our edge table must contain data which enables us to look up z at an intersection point
- From x1 to x2 only A is considered.
- At x2, A and B are considered
  - plane equations are solved to get depth at (x2, i)
  - A is closest so x2 to x3 is filled as A
- At x3 A finishes so we draw B from x3 to x4
- At x4 B and C are considered, B in front etc…

Method analysis

- Drawbacks
  - Sorting is required and added to the AET
  - If the number of polygons is high, the z-computation will be costly
- Acceleration
  - Would usually store the scanning in a 1D BSP tree for large numbers of polygons!
  - Exploit coherence (assume similar overlapping at y +1)
  - Pre-order the polygons (no need to compute the depth calculations)

Z-Buffer

- Don't bother with per span tests - just test every pixel
- In addition to frame buffer (R, G, B)
- Store distance to camera (z-buffer)

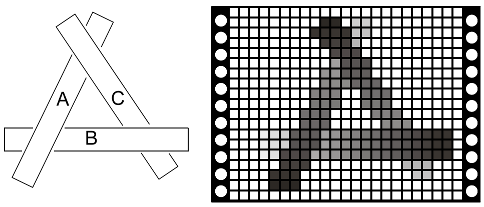


Z-Buffer

- Most common usage is a full window sized array ZBUF (M*N) of 16, 24 or 32 bit “depth” values
- Basic idea:
  – Initialise Z-Buffer to Z_MAX
  – For each polygon
    - Point (x,y,z) of the polygon projects on pixel (xs,ys) and has colour col associated
    - If z < ZBUF[x,y] then set CBUF[x,y] = col, ZBUF[x,y] = z else do nothing

Z-Buffer

- Works for hard cases



Polygon scan-line renderer

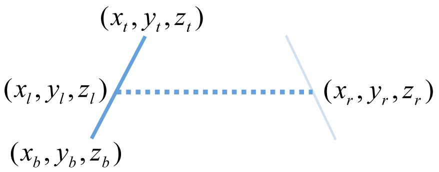
- We can do this in several ways
- 1D z-buffer re-used on each scan line
  – Process *each* polygon with separate AET
  – Or use as adjunct to extended AET for multiple polygons
- Problems with z-buffer…
  – Aliasing on depth (z-buffer tearing)

Scanning Depth into the Z-Buffer

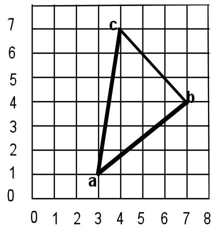
- Now we have to write a z-value for each point
  - directly from plane equation (re-calculate for each point)
  - incremental across the scan-line (store z_start and dz)
  - Interpolate
    - We will look at this in more detail!

Interpolating Depth

- Interpolate z along edges $dz_{tb} = \frac{z_t - z_b}{y_t - y_b}$
- AND interpolate between edges on each scan-line (bi-linear interpolation) $dz_{lr} = \frac{z_r - z_l}{x_r - x_l}$



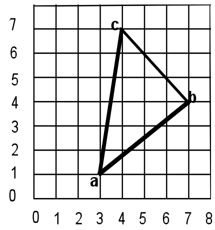
Z-Buffer Fill Example



- General form of ET
  - (y2,x1,dx/dy, z1,dz/dy)
- ET[1] =
  - ac (7,3,1/6, 1,3/6)
  - ab (4,3,4/3, 1,1/3)
- ET[4] =
  - cb (7,7,-3/3, 2,2/3)

a=(3,1,1) b=(7,4,2) c=(4,7,4)

…Contents of AET

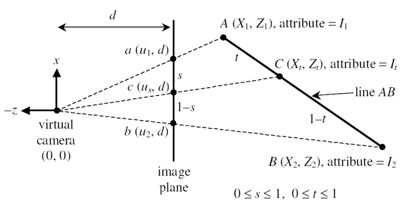


- Scanline y=1
  - ac (7,3,1/6, 1,3/6)
  - ab (4,3,4/3, 1,1/3)
  - zspans 1 to 1
- y=2
  - ac (7,3.166,1/6, 1.5,3/6)
  - ab (4,4.333,4/3, 1.333, 1/3)
  - zspans 1.5 to 1.333
- y=3
  - ac (7,3.333,1/6, 2.0,3/6)
  - ab (4,5.666,4/3, 1.666,1/3)
  - zspans 2 to 1.666

Interpolating Depth

- Interpolating z *linearly* along scan-line is **incorrect**!
- Why is that?
  - Projection of a point onto screen is done with non-linear projection matrix (remember: 1/(z+1) factor)
  - **Must** take that into account

Perspective Correct Depth Interpolation



- Given this scenario, can write down equations and see that we need to linearly interpolate 1/z (and not z)

Perspective Correct Depth Interpolation

- By similar triangles we have:

$$\frac{X_1}{Z_1} = \frac{u_1}{d} \Rightarrow X_1 = \frac{u_1 Z_1}{d}, \quad (1)$$

$$\frac{X_2}{Z_2} = \frac{u_2}{d} \Rightarrow X_2 = \frac{u_2 Z_2}{d}, \quad (2)$$

$$\frac{X_t}{Z_t} = \frac{u_s}{d} \Rightarrow Z_t = \frac{dX_t}{u_s}. \quad (3)$$

Perspective Correct Depth Interpolation

By linearly interpolating in the image plane (or screen space), we have

$$u_s = u_1 + s(u_2 - u_1). \quad (4)$$

By linearly interpolating across the primitive in the camera coordinate system, we have

$$X_t = X_1 + t(X_2 - X_1), \quad (5)$$

$$Z_t = Z_1 + t(Z_2 - Z_1), \quad (6)$$

Substituting (4) and (5) into (3),

$$Z_t = \frac{d\,(X_1 + t(X_2 - X_1))}{u_1 + s(u_2 - u_1)}. \quad (7)$$

Perspective Correct Depth Interpolation

Substituting (1) and (2) into (7),

$$Z_t = \frac{d\left(\frac{u_1 Z_1}{d} + t\left(\frac{u_2 Z_2}{d} - \frac{u_1 Z_1}{d}\right)\right)}{u_1 + s(u_2 - u_1)}$$
$$= \frac{u_1 Z_1 + t(u_2 Z_2 - u_1 Z_1)}{u_1 + s(u_2 - u_1)}. \quad (8)$$

Substituting (6) into (8),

$$Z_1 + t(Z_2 - Z_1) = \frac{u_1 Z_1 + t(u_2 Z_2 - u_1 Z_1)}{u_1 + s(u_2 - u_1)}, \quad (9)$$

Perspective Correct Depth Interpolation

which can be simplified into

$$t = \frac{sZ_1}{sZ_1 + (1-s)Z_2}. \quad (10)$$

Substituting (10) into (6), we have

$$Z_t = Z_1 + \frac{sZ_1}{sZ_1 + (1-s)Z_2}(Z_2 - Z_1), \quad (11)$$

which can be simplified to

$$Z_t = \frac{1}{\frac{1}{Z_1} + s\left(\frac{1}{Z_2} - \frac{1}{Z_1}\right)}. \quad (12)$$

Perspective Correct Depth Interpolation

- Thus we only need to linearly interpolate between 1/z values:

$$Z_t = \frac{1}{\frac{1}{Z_1} + s\left(\frac{1}{Z_2} - \frac{1}{Z_1}\right)}. \quad (12)$$

Trade-Offs

- Z-Buffer can be inaccurate with few bits
  - really simple to implement though!
- Scan-line AET good for large polygons
  - good coherency across lines
  - requires non-intersecting polygons
- Z-Buffer good for small, sparse polygons
  - AET more time consuming to maintain

Recap

- Rasterization
  - Z-Buffer for visibility
  - Need to do perspective correct rasterization