CS559 – Lecture 14
Visibility, Projection

These are course notes (not used as slides)
Written by Mike Gleicher, Oct. 2006

Basic Perspective

- Similar Triangles
- Warning = using d for focal length (like book)
  - F will be “far plane”

\[
\frac{y}{z} = \frac{y'}{d} \quad y' = \frac{d}{z}y
\]

D = focal length

Use Homogeneous coordinates!

- Use divide by w to get perspective divide
- Issues with simple version:
  - Font / back of viewing volume
  - Need to keep some of Z in Z (not flatten)

\[
\begin{bmatrix}
x' \\
y' \\
z' \\
w'
\end{bmatrix} =
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 1 & 1
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z \\
w
\end{bmatrix} =
\begin{bmatrix}
x/z \\
y/z \\
z/z = 1 \\
1
\end{bmatrix}
\]

The real perspective matrix

- N = near distance, F = far distance
- Z = n put on front plane, z=f put on far plane

\[
P =
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & n + f & -f \\
0 & 0 & n & 0
\end{bmatrix}
\]

Shirley’s Perspective Matrix

- After we do the divide, we get an unusual thing for z – it does preserve the order and keeps n&f

\[
P_{x} = P
\begin{bmatrix}
x \\
y \\
z \\
1
\end{bmatrix} =
\begin{bmatrix}
x \frac{n}{z} \\
y \frac{n}{z} \\
n + f - \frac{fn}{z} \\
1
\end{bmatrix}
\]

Camera Model

- The “window coordinate” system is all the we really know
- In a sense, it is the camera coordinate system
- Easiest to think about it as a camera taking a picture of the work
- Transform world coordinates into camera coordinates
  - Or, think about it the other way…
How to describe cameras?

- Rotate and translate (and scale) the world to be in view
- The camera is a physical object (that can be rotated and translated in the world)
- Easier ways to specify cameras
  - Lookfrom/at/vup

A Hack: Painted Shadows

- Use projection to squash objects onto floor
- Paint a copy of them in black on the floor
- Useful for UI
- Drop Straight onto floor = set Y to zero
- Beware – might want to have things float above floor

Projective Shadows – point light

- Position of light Lx, Ly, Lz
- Position of point x,y,z
- Position of Shadow Sx,0,Sz
- Assume ground (y) = 0

Visibility

- A topic for later in the class:
  How to get objects to occlude each other
- Give polygons in any order (even back ones last)
- Use a Z-Buffer to store depth at each pixel
- Things that can go wrong:
  - Near and far planes DO matter
  - Backface culling and other tricks can be problematic
  - You may need to turn the Z-buffer on
  - Don’t forget to clear the Z-Buffer!

How to make objects solid

- So far, just curves (outlines of things)
- Can fill regions (polygons)
  - But how to get stuff in front to occlude stuff in back
- General categories
  - Re-think drawing
    - From eye (pixels) not objects
    - Analytically compute what can be seen
    - Hidden line drawing (hard)
  - Hidden Surface Removal

Painter’s Algorithm

- Simplest hidden surface algorithm
- Draw farthest objects first
  - Nearer object cover further ones
- Problems
  - Cycles / intersections (no order possible)
    - Fix by splitting triangles
  - Need all triangles ahead of time
  - O(n log n) sort
  - Must resort for every view direction
- Depth Complexity (amount of time each pixels is drawn)
Binary Space Partitions

- Fancy data structure to help painters algorithm
- Stores order from any viewpoint
- A plane (one of the triangles) divides other triangles
- Things on same side as eye get drawn last

Using a BSP tree

- Recursively divide up triangles
- Traverse entire tree
  - Draw farther from eye subtree
  - Draw root
  - Draw closer to eye subtree
- Always O(n) to traverse
  - (since we explore all nodes)
  - No need to worry about it being balanced

Building a BSP tree

- Each triangle must divide other triangles
  - Cut triangles if need be (like painters alg)
- Goal in building tree: minimize cuts

Z-Buffer

- Throw memory at the problem
- A hardware visibility solution
  - Useful in software, but a real win for hardware
- For every pixel, store depth that pixel came from
- No object? Store $\infty$
- When you draw a pixel, only write the pixel if you pass the “z-test”

Things to notice about Z-Buffer

- Pretty much order independent
  - Same Z-values
  - Transparent objects
- Z-fighting
  - Objects have same Z-value, ordering is “random”
  - Bucketing (finite resolution) causes more things to be same
  - As things move, they may flip order
- Anti-Aliasing
  - Things done per-pixel, so sampling issues

Resolution of Z-Buffer

- Old days: big deal
  - Integer Z-buffers, limited resolution
- Future: floating point z-buffer
  - Still have resolution issues, not as bad
- Need to bucket things from near to far
  - Don’t set near too near or far to far
- Non-linear nature of post-divide Z
  - Remember that perspective divide gives fn/z