CS 559 Lecture 18
Visibility

Mike Gleicher
October 2007
Notes – not for display

Visibility: What objects do you see?

- What objects are offscreen?
  - To avoid drawing them
- What objects are blocked?
  - Need to make things look solid
- Assumes we have "filled" primitives
  - Triangles, not lines

How to make objects solid

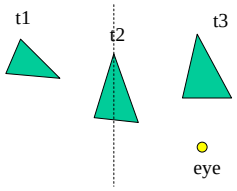
- Can fill regions (polygons)
  - But how to get stuff in front to occlude stuff in back
- General categories
  - Re-think drawing
    - From eye (pixels) not objects
    - "light-based rendering" (like ray tracing) later in course
  - Analytically compute what can be seen
    - Hidden line drawing (hard)
  - 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



T2 divides into groups
T3 is on same side of eye

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 ∞
- 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

Using the Z buffer

- 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!