Today

- Clipping
  - Clipping to view volume
  - Clipping arbitrary polygons
- Hidden Surface Removal
  - Z-Buffer
  - BSP Trees
  - Others

Clipping

- Stuff outside view volume should not be drawn
  - Too close: obscures view
  - Too far:
    - Complexity
    - Z-buffer problems
  - Too high/low/right/left:
    - Memory errors
    - Broken algorithms
    - Complexity
Clipping Line to Line/Plane

Line segment to be clipped
\[ x(t) = a + t(b - a) \]

Line/plane that clips it
\[ \hat{n} \cdot x - \hat{n} \cdot r = 0 \]

Clipping Line to Line/Plane

Line segment to be clipped
\[ x(t) = a + t(b - a) \]

Line/plane that clips it
\[ \hat{n} \cdot x - f = 0 \]
\[ \hat{n} \cdot (a + t(b - a)) - f = 0 \]
\[ \hat{n} \cdot a + t(\hat{n} \cdot (b - a)) - f = 0 \]

Segment may be on one side
\[ t \notin [0...1] \]

Lines may be parallel
\[ \hat{n} \cdot d = 0 \]

Convex domain defined by collection of planes (or lines or hyper-planes)

Planes have outward pointing normals

Clip against each plane in turn

Check for early/trivial rejection

\[ |\hat{n} \cdot d| \leq \varepsilon \quad \text{(Recall comments about numerical issues)} \]
Polygon Clip to Convex Domain

- Sutherland-Hodgman algorithm
  - Basically edge walking
- Clipping done often... should be efficient
  - Liang-Barsky parametric space algorithm
  - See text for clipping in 4D homogenized coordinates
General Polygon Clipping
- Weiler Algorithm
  - Double edges

Hidden Surface Removal
- True 3D to 2D projection would put everything overlapping into the view plane.
- We need to determine what’s in front and display only that.

Z-Buffers
- Add extra depth channel to image
- Write Z values when writing pixels
- Test Z values before writing

Z-Buffers
- Benefits
  - Easy to implement
  - Works for most any geometric primitive
  - Parallel operation in hardware
- Limitations
  - Quantization and aliasing artifacts
  - Overfill
  - Transparency does not work well
**Z-Buffers**

- Transparency requires partial sorting:

<table>
<thead>
<tr>
<th>Partially transparent</th>
<th>3rd</th>
<th>Front</th>
</tr>
</thead>
<tbody>
<tr>
<td>Opaque</td>
<td>2nd</td>
<td></td>
</tr>
<tr>
<td>Opaque</td>
<td>1st</td>
<td></td>
</tr>
</tbody>
</table>

Good

Not Good

**Z-Buffers**

Recall depth-value distortions.

It's a feature...!

More resolution near viewer!

Best use of limited precision

---

**A-Buffers**

- Store sorted list of “fragments” at each pixel
- Draw all opaque stuff first then transparent
- Stuff behind full opacity gets ignored

- Nice for antialiasing...

**Scan-line Algorithm**

- Assume polygons don’t intersect
- Each time an edge is crossed determine who’s on top
**Painter’s Algorithm**

- Sort Polygons Front-to-Back
  - Draw in order
  - Back-to-Front works also, but wasteful
- How to sort quickly?
- Intersecting polygons?
- Cycles?

**BSP-Trees**

- Binary Space Partition Trees
  - Split space along planes
  - Allows fast queries of some spatial relations
- Simple construction algorithm
  - Select a plane as sub-tree root
  - Everything on one side to one child
  - Everything on the other side to other child
  - Use random polygon for splitting plane
Visibility Traversal

- Variation of in-order-traversal
  - Child one
  - Sub-tree root
  - Child two
- Select “child one” based on location of viewpoint
  - Child one on same side of sub-tree root as viewpoint

BSP-Trees