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
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

\[ \hat{n} \cdot x - f = 0 \]

\[ \hat{n} \cdot (a + t(b - a)) - f = 0 \]

\[ \hat{n} \cdot a + t(\hat{n} \cdot (b - a)) - f = 0 \]
Clipping Line to Line/Plane

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

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

\[ t = \frac{f - \hat{n} \cdot a}{\hat{n} \cdot d} \]

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

- 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
Polygon Clip to Convex Domain
Polygon Clip to Convex Domain

Inside | Outside | Inside | Outside | Inside | Outside | Inside | Outside

Output p | Output i | No output | Output i and p
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

$A \cap B$

$A \cup B$

$A - B$

$B - A$

$A \cup B$

$A \cap B$
General Polygon Clipping

- Weiler Algorithm
  - Double edges
Hidden Surface Removal

- True 3D to 2D projection would put every thing 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

Images from Okan Arikan
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></th>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
<th>Front</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partially透明</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>不透明</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>不透明</td>
<td></td>
<td></td>
<td></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
BSP-Trees

a, b, c, d, e, f, g

a, b, c, d, e, f, g
BSP-Trees

The diagram illustrates the structure of a BSP-Trees, which is a type of binary tree used in computer science for partitioning a space. The tree is rooted at node 'a' and has two children: 'b, c_1, d' and 'c_2, e, f, g'. Each node represents a partition of the space, with branches indicating further divisions.
BSP-Trees
BSP-Trees

[Diagram of BSP-Trees with labels a, b, c, d, e, f, g and a tree structure with nodes a, b, c, d, e, f, g and edges connecting them.]

27
BSP-Trees
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

c₁:b:d:a:f:e₁:c₂:g:e₂
BSP-Trees

\[
g : e_2 : c_2 : f : e_1 : a : c_1 : b : d
\]