CS-184: Computer Graphics

Lecture #10: Clipping and Hidden Surfaces

Prof. James O'Brien
University of California, Berkeley

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

<table>
<thead>
<tr>
<th>Line segment to be clipped</th>
<th>Line/plane that clips it</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x(t) = a + t(b - a)$</td>
<td>$\hat{n} \cdot x - \hat{n} \cdot r = 0$</td>
</tr>
</tbody>
</table>

The solution for $t$ is given by:

$$t = \frac{-\hat{n} \cdot a}{\hat{n} \cdot d}$$

### Clipping Line to Line/Plane

<table>
<thead>
<tr>
<th>Line segment to be clipped</th>
<th>Line/plane that clips it</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x(t) = a + t(b - a)$</td>
<td>$\hat{n} \cdot x - f = 0$</td>
</tr>
<tr>
<td></td>
<td>$\hat{n} \cdot (a + t(b - a)) - f = 0$</td>
</tr>
<tr>
<td></td>
<td>$\hat{n} \cdot a + t(\hat{n} \cdot (b - a)) - f = 0$</td>
</tr>
</tbody>
</table>
### Clipping Line to Line/Plane

- Segment may be on one side
  \[ t \notin [0 \ldots 1] \]
- Lines may be parallel

\[
\hat{n} \cdot d = 0
\]
\[
|\hat{n} \cdot d| \leq \varepsilon \quad \text{(Recall comments about numerical issues)}
\]

### Triangle Clip/Split

Double vertices if you want separation...
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 Clipping

- Find the part of a polygon inside the clip window?
Polygon Clipping

- Find the part of a polygon inside the clip window?

After Clipping

Sutherland-Hodgman Clipping

- Clip to each window boundary one at a time
Sutherland-Hodgman Clipping

- Clip to each window boundary one at a time
Sutherland-Hodgman Clipping

- Clip to each window boundary one at a time
Polygon Clip to Convex Domain

Note double edges.

17

18
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

- $A - B$
- $B - A$
- $A \cup B$
- $A \cap B$
General Polygon Clipping

• Weiler Algorithm
  • Double edges

Hidden Surface Removal

• True 3D to 2D projection would put 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

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

Good

Not Good

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
- Draw Front-to-Back
  - Draw same-side polygons first
  - Draw root node polygon (if any)
  - Draw other-side polygons last