Clipping

Line Clipping
Polygon Clipping
Clipping in Three Dimensions
[Angel 8.3-8.7]

March 11, 2003
Frank Pfenning
Carnegie Mellon University

http://www.cs.cmu.edu/~fp/courses/graphics/
The Graphics Pipeline, Revisited

- Must eliminate objects outside viewing frustum
- Tied in with projections
  - **Clipping**: object space (eye coordinates)
  - **Scissoring**: image space (pixels in frame buffer)
- Introduce **clipping** in stages
  - 2D (for simplicity)
  - 3D (as in OpenGL)
- In a later lecture: **scissoring**
Transformations and Projections

- Sequence applied in many implementations
  1. Object coordinates to
  2. Eye coordinates to
  3. Clip coordinates to
  4. Normalized device coordinates to
  5. Screen coordinates
Clipping Against a Frustum

• General case of frustum (truncated pyramid)

• Clipping is tricky because of frustum shape
Perspective Normalization

• Solution:
  – Implement perspective projection by perspective normalization and orthographic projection
  – Perspective normalization is a homogeneous tfm.

See [Angel Ch. 5.8]
The Normalized Frustum

- OpenGL uses \(-1 \leq x, y, z \leq 1\) (others possible)
- Clip against resulting cube
- Clipping against programmer-specified planes is different and more expensive
- Often a useful programming device
The Viewport Transformation

- Transformation sequence again:
  1. **Camera**: From object coordinates to eye coords
  2. **Perspective normalization**: to clip coordinates
  3. **Clipping**
  4. **Perspective division**: to normalized device coords.
  5. **Orthographic projection** (setting $z_p = 0$)
  6. **Viewport transformation**: to screen coordinates

- Viewport transformation can distort
- Often in OpenGL: resize callback
Line-Segment Clipping

• General: 3D object against cube
• Simpler case:
  – In 2D: line against square or rectangle
  – Before scan conversion (rasterization)
  – Later: polygon clipping
• Several practical algorithms
  – Avoid expensive line-rectangle intersections
  – Cohen-Sutherland Clipping
  – Liang-Barsky Clipping
  – Many more [see Foley et al.]
Clipping Against Rectangle

- **Line-segment clipping**: modify endpoints of lines to lie within clipping rectangle
- Could calculate intersections of line (segments) with clipping rectangle (expensive)
Cohen-Sutherland Clipping

• Clipping rectangle as intersection of 4 half-planes

\[ \text{interior} = \bigcap \begin{align*} 
    x > \text{xmin} \\
    x < \text{xmax} \\
    \text{y} < \text{ymax} \\
    \text{y} > \text{ymin} 
\end{align*} \]

• Encode results of four half-plane tests
• Generalizes to 3 dimensions (6 half-planes)
Outcodes

• Divide space into 9 regions
• 4-bit outcode determined by comparisons

\[
\begin{align*}
\text{o}_1 &= \text{outcode}(x_1, y_1) \text{ and } \text{o}_2 = \text{outcode}(x_2, y_2) \\
\text{b}_0 &= y > y_{\text{max}} \\
\text{b}_1 &= y < y_{\text{min}} \\
\text{b}_2 &= x > x_{\text{max}} \\
\text{b}_3 &= x < x_{\text{min}}
\end{align*}
\]
Cases for Outcodes

- Outcomes: accept, reject, subdivide

\[ \begin{align*}
0_1 = 0_2 = 0000 & : \text{accept} \\
0_1 \& 0_2 \neq 0000 & : \text{reject} \\
0_1 = 0000, \ 0_2 \neq 0000 & : \text{subdiv} \\
0_1 \neq 0000, \ 0_2 = 0000 & : \text{subdiv} \\
0_1 \& 0_2 = 0000 & : \text{subdiv}
\end{align*} \]
Cohen-Sutherland Subdivision

• Pick outside endpoint \((o \neq 0000)\)
• Pick a crossed edge \((o = b_0b_1b_2b_3 \text{ and } b_k \neq 0)\)
• Compute intersection of this line and this edge
• Replace endpoint with intersection point
• Restart with new line segment
  – Outcodes of second point are unchanged
• Must converge (roundoff errors?)
Liang-Barsky Clipping

- Starting point is parametric form
  \[ p(\alpha) = (1 - \alpha)p_1 + \alpha p_2, \quad 0 \leq \alpha \leq 1 \]
  \[ x(\alpha) = (1 - \alpha)x_1 + \alpha x_2 \]
  \[ y(\alpha) = (1 - \alpha)y_1 + \alpha y_2 \]

- Compute four intersections with extended clipping rectangle
- Will see that this can be avoided
Ordering of intersection points

- Order the intersection points
- Figure (a): $1 > \alpha_4 > \alpha_3 > \alpha_2 > \alpha_1 > 0$
- Figure (b): $1 > \alpha_4 > \alpha_2 > \alpha_3 > \alpha_1 > 0$
Liang-Barsky Efficiency Improvements

• Efficiency improvement 1:
  – Compute intersections one by one
  – Often can reject before all four are computed

• Efficiency improvement 2:
  – Equations for $\alpha_3$, $\alpha_2$

\[
y_{max} = (1 - \alpha_3)y_1 + \alpha_3 y_2
\]
\[
x_{min} = (1 - \alpha_2)x_1 + \alpha_2 x_2
\]
\[
\alpha_3 = \frac{y_{max} - y_1}{y_2 - y_1}
\]
\[
\alpha_2 = \frac{x_{min} - x_1}{x_2 - x_1}
\]

  – Compare $\alpha_3$, $\alpha_2$ without floating-point division
Line-Segment Clipping Assessment

• Cohen-Sutherland
  – Works well if many lines can be rejected early
  – Recursive structure (multiple subdiv) a drawback

• Liang-Barsky
  – Avoids recursive calls (multiple subdiv)
  – Many cases to consider (tedious, but not expensive)
  – Used more often in practice (?)
Outline

• Line-Segment Clipping
  – Cohen-Sutherland
  – Liang-Barsky
• Polygon Clipping
  – Sutherland-Hodgeman
• Clipping in Three Dimensions
Polygon Clipping

- Convert a polygon into **one or more** polygons
- Their union is intersection with clip window
- Alternatively, we can first tessellate concave polygons (OpenGL supported)
Concave Polygons

• **Approach 1:** clip and join to a single polygon

  ![Diagram](a)

  ![Diagram](b)

• **Approach 2:** tesselate and clip triangles

  ![Diagram](c)
Sutherland-Hodgeman I

• Subproblem:
  – Input: polygon (vertex list) and single clip plane
  – Output: new (clipped) polygon (vertex list)

• Apply once for each clip plane
  – 4 in two dimensions
  – 6 in three dimensions
  – Can arrange in pipeline
Sutherland-Hodgeman II

• To clip vertex list (polygon) against half-plane:
  – Test first vertex. Output if inside, otherwise skip.
  – Then loop through list, testing transitions
    • In-to-in: output vertex
    • In-to-out: output intersection
    • out-to-in: output intersection and vertex
    • out-to-out: no output
  – Will output clipped polygon as vertex list

• May need some cleanup in concave case
• Can combine with Liang-Barsky idea
Other Cases and Optimizations

- **Curves and surfaces**
  - Analytically if possible
  - Through approximating lines and polygons otherwise

- **Bounding boxes**
  - Easy to calculate and maintain
  - Sometimes big savings
Outline

• Line-Segment Clipping
  – Cohen-Sutherland
  – Liang-Barsky

• Polygon Clipping
  – Sutherland-Hodgeman

• Clipping in Three Dimensions
Clipping Against Cube

- Derived from earlier algorithms
- Can allow right parallelepiped
Cohen-Sutherland in 3D

• Use 6 bits in outcode
  – $b_4$: $z > z_{\text{max}}$
  – $b_5$: $z < z_{\text{min}}$
• Other calculations as before
Liang-Barsky in 3D

- Add equation \( z(\alpha) = (1- \alpha) z_1 + \alpha z_2 \)
- Solve, for \( p_0 \) in plane and normal \( n \):

\[
p(\alpha) = (1 - \alpha)p_1 + \alpha p_2
\]
\[
n \cdot (p(\alpha) - p_0) = 0
\]

- Yields

\[
\alpha = \frac{n \cdot (p_0 - p_1)}{n \cdot (p_2 - p_1)}
\]

- Optimizations as for Liang-Barsky in 2D
Perspective Normalization

- Intersection simplifies for orthographic viewing
  - One division only (no multiplication)
  - Other Liang-Barsky optimizations also apply
- Otherwise, use perspective normalization
  - Reduces to orthographic case
  - Applies to oblique and perspective viewing

Normalization of oblique projections
Summary: Clipping

• Clipping line segments to rectangle or cube
  – Avoid expensive multiplications and divisions
  – Cohen-Sutherland or Liang-Barsky

• Clipping to viewing frustum
  – Perspective normalization to orthographic projection
  – Apply clipping to cube from above

• Client-specific clipping
  – Use more general, more expensive form

• Polygon clipping
  – Sutherland-Hodgeman pipeline
Preview and Announcements

• Scan conversion
• Anti-aliasing
• Other pixel-level operations
• Assignment 5 due a week from Thursday!
• Start early!
• Sriram’s office hours now Mon 4:30-6:30
• Movie
• Returning Midterm