The Graphics Pipeline: Line Clipping & Line Rasterization

Last Time?
- Ray Tracing vs. Scan Conversion
- Overview of the Graphics Pipeline
- Projective Transformations

Today: Line Clipping & Rasterization
- Portions of the object outside the view frustum are removed
- Rasterize objects into pixels

Today
- Why Clip?
- Line Clipping
- Overview of Rasterization
- Line Rasterization
- Circle Rasterization
- Antialiased Lines

Clipping
- Eliminate portions of objects outside the viewing frustum
- View Frustum
  - boundaries of the image plane projected in 3D
  - a near & far clipping plane
- User may define additional clipping planes

Questions?

MIT EECS 6.837, Durand and Cutler
Why clip?
- Avoid degeneracies
  - Don’t draw stuff behind the eye
  - Avoid division by 0 and overflow
- Efficiency
  - Don’t waste time on objects outside the image boundary
- Other graphics applications (often non-convex)
  - Hidden surface removal, Shadows, Picking, Binning, CSG (Boolean) operations (2D & 3D)

Clipping strategies
- Don’t clip (and hope for the best)
- Clip on-the-fly during rasterization
- Analytical clipping: alter input geometry

Questions?

Today
- Why Clip?
- Point & Line Clipping
  - Plane – Line intersection
  - Segment Clipping
  - Acceleration using outcodes
- Overview of Rasterization
- Line Rasterization
- Circle Rasterization
- Antialiased Lines

Implicit 3D Plane Equation
- Plane defined by:
  - point \( p \) & normal \( n \)  OR
  - normal \( n \) & offset \( d \)  OR
  - 3 points
- Implicit plane equation
  \[ Ax + By + Cz + D = 0 \]

Homogeneous Coordinates
- Homogenous point: \((x,y,z,w)\)
  - infinite number of equivalent homogenous coordinates:
  \((sx, sy, sz, sw)\)
- Homogenous Plane Equation:
  \[ Ax + By + Cz + D = 0 \]  \(\rightarrow\)  \(H = (A,B,C,D)\)
  - Infinite number of equivalent plane expressions:
  \[ sAx + sBy + sCz + sD = 0 \]  \(\rightarrow\)  \(H = (sA,sB,sC,sD)\)
Point-to-Plane Distance

- If \((A, B, C)\) is normalized:
  \[ d = H \cdot p = H^T p \]
  (the dot product in homogeneous coordinates)

- \(d\) is a signed distance
  - positive = "inside"
  - negative = "outside"

Clipping a Point with respect to a Plane

- If \(d = H \cdot p \geq 0\):
  Pass through
- If \(d = H \cdot p < 0\):
  Clip (or cull or reject)

Clipping with respect to View Frustum

- Test against each of the 6 planes
  - Normals oriented towards the interior
- Clip (or cull or reject) point \(p\) if any \(H \cdot p < 0\)

What are the View Frustum Planes?

- \(H_{\text{near}} = (0, 0, -1, -\text{near})\)
- \(H_{\text{far}} = (0, 0, 1, \text{far})\)
- \(H_{\text{bottom}} = (0, \text{near bottom}, 0, 0)\)
- \(H_{\text{top}} = (0, -\text{near}, -\text{top}, 0)\)
- \(H_{\text{left}} = (-1, \text{near}, 0, 0)\)
- \(H_{\text{right}} = (-1, -\text{near}, 0, 0)\)

Clipping & Transformation

- Transform \(M\) (e.g. from world space to NDC)
  \[ (1, 1, 1) \]
- \((-1, -1, -1)\)
- The plane equation is transformed with \((M^{-1})^T\)

Segment Clipping

- If \(H \cdot p > 0\) and \(H \cdot q < 0\)
- If \(H \cdot p < 0\) and \(H \cdot q > 0\)
- If \(H \cdot p > 0\) and \(H \cdot q > 0\)
- If \(H \cdot p < 0\) and \(H \cdot q < 0\)
Segment Clipping

- If $\mathbf{H} \cdot \mathbf{p} > 0$ and $\mathbf{H} \cdot \mathbf{q} < 0$
  - clip q to plane
- If $\mathbf{H} \cdot \mathbf{p} < 0$ and $\mathbf{H} \cdot \mathbf{q} > 0$
  - clip p to plane
- If $\mathbf{H} \cdot \mathbf{p} > 0$ and $\mathbf{H} \cdot \mathbf{q} > 0$
  - pass through
- If $\mathbf{H} \cdot \mathbf{p} < 0$ and $\mathbf{H} \cdot \mathbf{q} < 0$
  - clipped out

Clipping against the frustum

- For each frustum plane $\mathbf{H}$
  - If $\mathbf{H} \cdot \mathbf{p} > 0$ and $\mathbf{H} \cdot \mathbf{q} < 0$, clip q to H
  - If $\mathbf{H} \cdot \mathbf{p} < 0$ and $\mathbf{H} \cdot \mathbf{q} > 0$, clip p to H
  - If $\mathbf{H} \cdot \mathbf{p} > 0$ and $\mathbf{H} \cdot \mathbf{q} > 0$, pass through
  - If $\mathbf{H} \cdot \mathbf{p} < 0$ and $\mathbf{H} \cdot \mathbf{q} < 0$, clipped out

Line – Plane Intersection

- Explicit (Parametric) Line Equation
  \[ \mathbf{L}(t) = \mathbf{P}_0 + t \ast (\mathbf{P}_1 - \mathbf{P}_0) \]
  \[ \mathbf{L}(t) = (1 - t) \ast \mathbf{P}_0 + t \ast \mathbf{P}_1 \]
- How do we intersect?
  - Insert explicit equation of line into implicit equation of plane
- Parameter $t$ is used to interpolate associated attributes (color, normal, texture, etc.)
Is this Clipping Efficient?
• For each frustum plane $H$
  – If $H \cdot p > 0$ and $H \cdot q < 0$, clip $q$ to $H$
  – If $H \cdot p < 0$ and $H \cdot q > 0$, clip $p$ to $H$
  – If $H \cdot p > 0$ and $H \cdot q > 0$, pass through
  – If $H \cdot p < 0$ and $H \cdot q < 0$, clipped out

What is the problem?
The computation of the intersections, and any corresponding interpolated values is unnecessary
Can we detect this earlier?

Improving Efficiency: Outcodes
• Compute the sidedness of each vertex with respect to each bounding plane (0 = valid)
• Combine into binary outcode using logical AND

Outcode of $p$ : 1010
Outcode of $q$ : 0110
Outcode of $pq$ : 0010
Clipped because there is a 1

Improving Efficiency: Outcodes
• When do we fail to save computation?

Outcode of $p$ : 1000
Outcode of $q$ : 0010
Outcode of $pq$ : 0000
Not clipped
Questions?

Today

• Why Clip?
• Line Clipping
• Overview of Rasterization
• Line Rasterization
• Circle Rasterization
• Antialiased Lines

Framebuffer Model

• Raster Display: 2D array of picture elements (pixels)
• Pixels individually set/cleared (greyscale, color)
• Window coordinates: pixels centered at integers

2D Scan Conversion

• Geometric primitives (point, line, polygon, circle, polyhedron, sphere...)
• Primitives are continuous; screen is discrete
• Scan Conversion: algorithms for efficient generation of the samples comprising this approximation

Brute force solution for triangles

• For each pixel
  – Compute line equations at pixel center
  – “clip” against the triangle

Problem?
If the triangle is small, a lot of useless computation
Brute force solution for triangles

• Improvement:
  – Compute only for the screen bounding box of the triangle
  – Xmin, Xmax, Ymin, Ymax of the triangle vertices

Can we do better? Yes!

• More on polygons next week.
• Today: line rasterization

Questions?

Today

• Why Clip?
• Line Clipping
• Overview of Rasterization
• Line Rasterization
  – naive method
  – Bresenham's (DDA)
• Circle Rasterization
• Antialiased Lines

Scan Converting 2D Line Segments

• Given:
  – Segment endpoints (integers x1, y1; x2, y2)
• Identify:
  – Set of pixels (x, y) to display for segment

Line Rasterization Requirements

• Transform continuous primitive into discrete samples
• Uniform thickness & brightness
• Continuous appearance
• No gaps
• Accuracy
• Speed
Algorithm Design Choices

- Assume:
  - \( m = \frac{dy}{dx} \), \( 0 < m < 1 \)
- Exactly one pixel per column
  - fewer \( \rightarrow \) disconnected, more \( \rightarrow \) too thick

Naive Line Rasterization Algorithm

- Simply compute \( y \) as a function of \( x \)
  - Conceptually: move vertical scan line from \( x_1 \) to \( x_2 \)
  - What is the expression of \( y \) as function of \( x \)?
  - Set pixel \((x, \text{round}(y(x)))\)

Efficiency

- Computing \( y \) value is expensive
  - Observe: \( y' = m + m(x-x_1) \) at each \( x \)

Line Rasterization

- It's like marching a ray through the grid
- Also uses DDA (Digital Difference Analyzer)

Grid Marching vs. Line Rasterization

- Ray Acceleration:
  - Must examine every cell the line touches
- Line Rasterization:
  - Best discrete approximation of the line

Algorithm Design Choices

- Note: brightness can vary with slope
  - What is the maximum variation? \( \sqrt{2} \)
- How could we compensate for this?
  - Answer: antialiasing
Bresenham's Algorithm (DDA)

- Select pixel vertically closest to line segment
  - intuitive, efficient,
  pixel center always within 0.5 vertically
- Same answer as naive approach

Bresenham Step

- Which pixel to choose: E or NE?
  - Choose E if segment passes below or through middle point M
  - Choose NE if segment passes above M

Bresenham's Algorithm (DDA)

- Decision Function:
  \[ D(x, y) = y - mx - b \]
- Initialize:
  error term \( e = -D(x, y) \)
- On each iteration:
  update \( x \):
  \[ x' = x + 1 \]
  update \( e \):
  \[ e' = e + m \]
  if \( (e \leq 0.5) \):
  \[ y' = y \] (choose pixel E)
  if \( (e > 0.5) \):
  \[ y' = y + 1 \] (choose pixel NE) \( e' = e - 1 \)

Bresenham Step

- Use decision function \( D \) to identify points underlying line \( L \):
  \[ D(x, y) = y - mx - b \]
  - positive above \( L \)
  - zero on \( L \)
  - negative below \( L \)
  \[ D(p_x, p_y) = \text{vertical distance from point to line} \]

Summary of Bresenham

- initialize \( x, y, e \)
- for \( (x = x1; x \leq x2; x++) \)
  - plot \( (x, y) \)
  - update \( x, y, e \)
- Generalize to handle all eight octants using symmetry
- Can be modified to use only integer arithmetic
Today

- Why Clip?
- Line Clipping
- Overview of Rasterization
- Line Rasterization
  - naive method
  - Bresenham's (DDA)
- Circle Rasterization
- Antialiased Lines

Circle Rasterization

- Decision Function:
  \[ D(x, y) = x^2 + y^2 - R^2 \]
- Initialize:
  \[ e = -D(x, y) \]
- On each iteration:
  \[
  \begin{align*}
  &x' = x + 1 \\
  &e' = e + 2x + 1 \\
  &\text{if } (e \geq 0.5): \ y' = y \quad \text{(choose pixel E)} \\
  &\text{if } (e < 0.5): \ y' = y - 1 \quad \text{(choose pixel SE)}, \quad e' = e + 1
  \end{align*}
  \]
Antialiased Line Rasterization

Next Week:

Polygon Rasterization & Polygon Clipping