Overview

- Cohen-Sutherland Line Clipping
- Parametric Line Clipping
- 2D Affine transformations
- Homogeneous coordinates
- Discussion of homework #1

Lecture Credits: Most pictures are from Foley/VanDam; Additional and extensive thanks also goes to those credited on individual slides

Scissoring Clipping
Performed during scan conversion of the line (circle, polygon)

Compute the next point (x,y)
If \(x_{\text{min}} \leq x \leq x_{\text{max}}\) and \(y_{\text{min}} \leq y \leq y_{\text{max}}\)
Then output the point.
Else do nothing

• Issues with scissoring:
  – Too slow
  – Does more work then necessary
• Better to clip lines to window, than “draw” lines that are outside of window

The Cohen-Sutherland Line Clipping Algorithm

- How to clip lines to fit in windows?
  – easy to tell if whole line falls w/in window
  – harder to tell what part falls inside
- Consider a straight line \(P_0 = (x_0, y_0)\) and \(P_1 = (x_1, y_1)\).
- And window: \(WT, WB, WL\) and \(WR\)

Cohen-Sutherland

Basic Idea:
- **First**, do easy test
  – completely inside or outside the box?
- **If no**, we need a more complex test
• Note: we will also need to figure out how line intersects the box

Cohen-Sutherland

Perform trivial accept and reject

- Assign each end point a location code
- Perform bitwise logical operations on a line’s location codes
- Accept or reject based on result
- Frequently provides no information
  – Then perform more complex line intersection
Cohen-Sutherland

- The Easy Test:
  - Compute 4-bit code based on endpoints \( P_1 \) and \( P_2 \)

<table>
<thead>
<tr>
<th>Bit 1</th>
<th>Bit 2</th>
<th>Bit 3</th>
<th>Bit 4</th>
<th>Code Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>9</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>11</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>12</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>13</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>14</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>15</td>
</tr>
</tbody>
</table>

Bit 1: 1 if point is above window, i.e. \( y > WT \).
Bit 2: 1 if point is below window, i.e. \( y < WR \).
Bit 3: 1 if point is right of window, i.e. \( x > WR \).
Bit 4: 1 if point is left of window, i.e. \( x < WL \).

Otherwise, we clip the lines:

- We know that there is a bit flip, w.o.l.g. assume its \((x_0, x_1)\).
- Which bit? Try ‘em all!
  - suppose it’s bit 4
  - Then \( x_0 < WT \) and we know that \( x_0 > WL \)
  - We need to find the point: \((x_c, y_c)\)

- Replacing \((x_0, y_0)\) with \((x_c, y_c)\)
- Re-compute codes
- Continue until all bit flips (clip lines) are processed, i.e. all points are inside the clip window

Cohen-Sutherland

- If line segments are completely outside the window, then \( C_0 \cap C_1 \neq 0 \)

- Clearly: \( x_c = WL \)
- Using similar triangles
  \[
  \frac{y_c - y_0}{y_1 - y_0} = \frac{WL - x_0}{x_1 - x_0}
  \]
- Solving for \( y_c \) gives
  \[
  y_c = WL - x_0 \left( \frac{y_1 - y_0}{x_1 - x_0} \right) + y_0
  \]
Parametric Line Clipping

- Developed by Cyrus and Beck in 1978
- Used to clip 2D/3D lines against convex polygon/polyhedron
- Liang and Barsky (1984) algorithm efficient in clipping upright 2D/3D clipping regions
- Cyrus-Beck may be reduced to more efficient Liang-Barsky case
- Based on parametric form of a line
  - Line: $P(t) = P_0 + t(P_1 - P_0)$

Parametric Line Equation

- Line: $P(t) = P_0 + t(P_1 - P_0)$
- $t$ value defines a point on the line going through $P_0$ and $P_1$
- $0 \leq t \leq 1$ defines line segment between $P_0$ and $P_1$
- $P(0) = P_0$, $P(1) = P_1$
The Cyrus-Beck Technique

- Cohen-Sutherland algorithm computes \((x,y)\) intersections of the line and clipping edge
- Cyrus-Beck finds a value of parameter \(t\) for intersections of the line and clipping edges
- Simple comparisons used to find actual intersection points
- Liang-Barsky optimizes it by examining \(t\) values as they are generated to reject some line segments immediately

Finding the Intersection Points

Line \(P(t) = P_0 + t(P_1 - P_0)\)

Point on the edge \(P_{ei}\)

\(N_i\) is normal to edge \(i\)

\[\begin{align*}
N_i \cdot (P(t) - P_0) &= 0 \\
N_i \cdot (P_1 - P_0) &= 0 \\
N_i \cdot (P - P_0) + N_i \cdot (t(P_1 - P_0)) &= 0
\end{align*}\]

Let \(D = (P_0 - P_1)\)

\[t = \frac{D \cdot N_i}{D \cdot D}\]

Make sure
1. \(D \neq 0\), or \(P_0 \neq P_1\)
2. \(N_i \cdot D \neq 0\), lines are not parallel

Calculating \(N_i\)

\(N_i\) for window edges
- WT: (0,1) WB: (0, -1) WL: (-1,0) WR: (1,0)

\(N_i\) for arbitrary edges
- Calculate edge direction
  - \(E = (V_1 - V_0)\) \[|V_1 - V_0|\]
  - Be sure to process edges in CCW order
- Rotate direction vector \(-90^\circ\)
  - \(N_x = E_y\)
  - \(N_y = -E_x\)

Finding the Line Segment

- Calculate intersection points between line and every window line
- Classify points as potentially entering (PE) or leaving (PL)
- PE if crosses edge into inside half plane \(\Rightarrow\) angle \(P_0\) and \(N_i\) greater \(90^\circ\) \(\Rightarrow\) \(N_i \cdot D < 0\)
- PL otherwise.
- Find \(T_e = \max(t_E)\)
- Find \(T_l = \min(t_L)\)
- Discard if \(T_e > T_l\)
- If \(T_e < 0\), \(T_e = 0\)
- If \(T_l > 1\), \(T_l = 1\)
- Use \(T_e, T_l\) to compute intersection coordinates \((x_e, y_e), (x_l, y_l)\)

2D Transformations

All represented as matrix operations on vectors!
Parallel lines preserved, angles/lengths not

- Scale
- Rotate
- Translate
- Reflect
- Shear
2D Affine Transformations

- **Example 1:** rotation and non uniform scale on unit cube
- **Example 2:** shear first in x, then in y

Note:
- Preserves parallels
- Does not preserve lengths and angles

2D Transforms: Translation

- Rigid motion of points to new locations
  \[ x' = x + dx, \quad y' = y + dy \]
- Defined with column vectors:
  \[ \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} dx \\ dy \end{bmatrix} \]

2D Transforms: Scale

- Stretching of points along axes:
  \[ x' = s_x \cdot x, \quad y' = s_y \cdot y \]

In matrix form:
\[
\begin{bmatrix}
  x' \\
  y'
\end{bmatrix} =
\begin{bmatrix}
  s_x & 0 \\
  0 & s_y
\end{bmatrix}
\begin{bmatrix}
  x \\
  y
\end{bmatrix}
\]

or just: \[ P' = S \cdot P \]

2D Transforms: Rotation

- Rotation of points about the origin
  \[ x' = x \cdot \cos \theta - y \cdot \sin \theta, \quad y' = x \cdot \sin \theta + y \cdot \cos \theta \]
  Positive Angle: CCW
  Negative Angle: CW

Matrix form:
\[
\begin{bmatrix}
  x' \\
  y'
\end{bmatrix} =
\begin{bmatrix}
  \cos \theta & -\sin \theta \\
  \sin \theta & \cos \theta
\end{bmatrix}
\begin{bmatrix}
  x \\
  y
\end{bmatrix}
\]

or just: \[ P' = R \cdot P \]

2D Transforms: Rotation

- Substitute the 1st two equations into the 2nd two to get the general equation
  \[ x = r \cdot \cos \phi, \quad y = r \cdot \sin \phi \]
  \[ x' = r \cdot \cos (\theta + \phi) = r \cdot \cos \phi \cdot \cos \theta - r \cdot \sin \phi \cdot \sin \theta \]
  \[ y' = r \cdot \sin (\theta + \phi) = r \cdot \cos \phi \cdot \sin \theta + r \cdot \sin \phi \cdot \cos \theta \]
  \[ x' = x \cos(\theta) - y \sin(\theta) \]
  \[ y' = x \sin(\theta) + y \cos(\theta) \]

Homogeneous Coordinates

- Observe: translation is treated differently from scaling and rotation
- **Homogeneous coordinates:** allows all transformations to be treated as matrix multiplications

Example: A 2D point \((x, y)\) is the line \((x, y, w)\), where \(w\) is any real \(w\), in 3D homogeneous coordinates.

To get the point, homogenize by dividing by \(w\) (i.e., \(w = 1\))
Recall our Affine Transformations

Matrix Representation of 2D Affine Transformations

Composition of 2D Transforms

Composition of 2D Transforms

Composition of 2D Transforms

Composition of 2D Transforms

Composition of 2D Transforms

• Be sure to multiple transformations in proper order!

\[ P' = (T \circ (R \circ S \circ (T \circ P))) \]

\[ P' = ((T \circ (R \circ (S \circ T))) \circ P) \]

\[ P' = T \circ P \]
Programming assignment 1

- Implement Simplified Postscript reader
- Implement 2D transformations
- Implement Cohen-Sutherland clipping
  – Generalize edge intersection formula
- Generalize DDA or Bresenham algorithm
- Implement XPM image writer