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 within 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></th>
<th>1001</th>
<th>1000</th>
<th>1010</th>
<th>Window</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>0000</td>
<td>0010</td>
<td>WR</td>
<td></td>
</tr>
<tr>
<td>0101</td>
<td>0100</td>
<td>0110</td>
<td></td>
<td></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. \( y < WB \).
Bit 3: 1 if point is right of window, i.e. \( x > WR \).
Bit 4: 1 if point is left of window, i.e. \( x < WL \).

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

Cohen-Sutherland

- Line is completely visible iff both code values of endpoints are 0, i.e. \( C_0 \lor C_1 = 0 \)

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

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

- 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

\[
y_c = \frac{WL - x_0}{x_1 - x_0} (y_1 - y_0) + y_0
\]

Cohen-Sutherland

- Replace \((x_0, y_0)\) 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
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 $P_1$
- $P(0) = P_0 \quad P(1) = P_1$
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 \(P_{ei}\)

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 \(P_0 - P_1\) and \(N_i\) greater 90° \(\Rightarrow\) \(N_i \cdot D < 0\)
- PL otherwise.
- Find \(T_e = \max(t)\)
- Find \(T_l = \min(t)\)
- 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 \((x_e, y_e), (x_l, y_l)\)

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°  
    \(-N_y = E_x\)
    \(-N_x = E_y\)

2D Affine Transformations

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

- Scale
- Rotate
- Translate
- Reflect
- Shear

2D Transformations
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
  - $x' = x + d_x$
  - $y' = y + d_y$
- Defined with column vectors:
  - $P' = P + T$

2D Transforms: Scale

- Stretching of points along axes:
  - $x' = s_x \cdot x$
  - $y' = s_y \cdot y$
- 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
  - Positive Angle: CCW
  - Negative Angle: CW
- In 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: $P' = R \cdot P$

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 #, 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

- Translation:
  \[
  \begin{bmatrix}
  x' \\
  y'
  \end{bmatrix} =
  \begin{bmatrix}
  1 & d_x \\
  0 & 1
  \end{bmatrix}
  \begin{bmatrix}
  x \\
  y
  \end{bmatrix}
  \]

- Scale:
  \[
  \begin{bmatrix}
  x' \\
  y'
  \end{bmatrix} =
  \begin{bmatrix}
  s_x & 0 \\
  0 & s_y
  \end{bmatrix}
  \begin{bmatrix}
  x \\
  y
  \end{bmatrix}
  \]

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

- Shear:
  \[
  \begin{bmatrix}
  x' \\
  y'
  \end{bmatrix} =
  \begin{bmatrix}
  1 & s_x \\
  0 & 1
  \end{bmatrix}
  \begin{bmatrix}
  x \\
  y
  \end{bmatrix}
  \]

Composition of 2D Transforms

- Rotate about a point \( P1 \)
  - Translate \( P1 \) to origin
  - Rotate
  - Translate back to \( P1 \)

\[
T(x_1, y_1) \cdot R(\theta) \cdot T(-x_1, -y_1)
\]

\[
\begin{bmatrix}
T_{x_1, y_1} & R(\theta) & T_{-x_1, -y_1}
\end{bmatrix}
\]

Composition of 2D Transforms

- Scale + rotate object around point \( P1 \) and move to \( P2 \)
  - \( P1 \) to origin
  - Scale
  - Rotate
  - Translate to \( P2 \)

\[
T(x_2, y_2) \cdot R(\theta) \cdot S(s_x, s_y) \cdot T(-x_1, -y_1)
\]

\[
\begin{bmatrix}
T_{x_1, y_1} & R(\theta) & S(s_x, s_y) & T_{-x_1, -y_1}
\end{bmatrix}
\]
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 or PBM image writer

HW1 Steps

• Read in line segments from file
• Apply transformations
• Clip against world window
• Translate lines into screen/image coordinates
• Draw clipped lines into software frame buffer
• Output frame buffer pixels to standard out either in XPM or PBM format