Lecture 6
- clipping
- windowing and viewport
- scan conversion/ rasterization

Last class
- projective transform followed by normalization
- normalized view volume

Last lecture (clip coordinates):
A vertex \((w x, w y, w z, w)\)
is in the
normalized view volume if:
\[-w \leq w x \leq w\]
\[-w \leq w y \leq w\]
\[-w \leq w z \leq w\]

Any object that lies entirely outside the view volume
doesn't need to be drawn. Such objects can "culled".
Any object that lies partly outside the view volume
needs to be "clipped".
Today, "clipping" refers to both of these.

Terminology: clipping vs. culling

3D Line Clipping
Q: Given endpoints \((x_0, y_0, z_0), (x_1, y_1, z_0)\), how to check if the line segment needs to be clipped?
i.e. either discarded, or modified to lie in volume

3 cases of interest: the line may be:
- entirely outside of view volume
- entirely in view volume
- partly in view volume

To check if a line segment intersects a boundary e.g. \(x=1\), solve for \(t\):
\[t (x_0, y_0) + (1-t) (x_1, y_1) = (1, 1)\]
and check if \(0 \leq t \leq 1\).

2D Line Clipping (simpler to discuss)
Q: Given endpoints \((x_0, y_0), (x_1, y_1)\), how to check if the line segment needs to be clipped?

This line can be "trivially rejected" since the endpoint \(x\) values are both less than -1.
This line can be "trivially accepted" since the endpoint x and y values are all between -1 and 1.

Cohen-Sutherland (1965) encoded the above rules:

For each vertex, compute the outcode.

Trivially reject a line segment if bitwiseAND ( _____ ,  _____ ) contains a 1.

Trivially accept a line segment if bitwiseOR ( _____ ,  ______ ) == 0000.

In both cases below, we can neither trivially accept nor reject.

Outcodes are the same in the two cases.

What if we cannot trivially accept or reject?

Q: what is the logic condition for this general case?

A: bitwiseXOR( _____ ,  _____ ) is

If we cannot trivially accept or reject, then the line must cross one of x=1, x=-1, y=1, or y = -1.

Cohen-Sutherland: consider the bits b3, b2, b1, b0 such that XOR( b, b' ) = 1.

Modify/clip the line segment to remove the offending part.

Example:

First clip line segment so that b3 = 0 for both outcodes.

Then, clip line segment so that b2= 0 for both outcodes.

Then, clip line segment so that b1= 0 for both outcodes.
Then, clip line segment so that $b_0 = 0$ for both outcodes.

And we’re done.... trivial accept!

Typically we don't need to do all four clips before trivially rejecting.

Cohen-Sutherland line clipping in 3D:
(exactly the same idea but the outcodes have 6 bits)

By the way.....

If we didn't do a projective transformation and map to normalized view volume, we could still compute outcodes and do line clipping, but it wouldn't be as easy.

Algorithms for clipping polygons (SKIP !)

Recall:
OpenGL clips in (4D) 'clip coordinates'
\[(w, x, w, y, w, z, w)\]
\textit{not} in (3D) 'normalized device coordinates'
\[(x, y, z)\].

We can compute outcodes in clip coordinates easily.

But the line clipping is tricky in clip coordinates. Why?

Exercise (surprising):
Clipping based on 4D interpolation works!

Recall from lecture 2:
The above was an abuse of notation. It was meant to express that:
The issue for clipping is whether the following interpolation scheme can be used.

\[
\begin{bmatrix}
  a \\
  b \\
  c \\
  d
\end{bmatrix} + \left(1- \right) \begin{bmatrix}
  a' \\
  b' \\
  c' \\
  d'
\end{bmatrix}
\]

The answer is yes, but it requires some thought to see why.

Lecture 6

clipping

windowing and viewport

scan conversion / rasterization

What is a "window"?

Two meanings:

- region of display screen (pixels) that you can drag and resize. Also known as "display window".
- region of the near plane in camera coordinates. Also known as "viewing window".

glutCreateWindow('COMP557 A1')
glutInitWindowSize(int width, int height)
glutInitWindowPosition(int x, int y)
glutReshapeWindow(int width, int height)
glutPositionWindow(int x, int y)

What is a "viewport"?

glViewport(int x, int y, int width, int height)

A viewport is a region within a display window. (The default viewport is the whole window.)

"window to viewport" transformation

display window
two viewports

We've finally arrived at pixels!

How do we convert our floating point (continuous) primitives into integer locations (pixels)?
"Scan Conversion" ("Rasterization")

- convert a continuous representation of an object such as a point, line segment, curve, triangle, etc into a discrete (pixel) representation on a pixel grid

- why "scan"?

---

**Algorithm:**

- scan convert a line segment from \((x_0, y_0)\) to \((x_1, y_1)\)

  \[
m = \frac{y_1 - y_0}{x_1 - x_0} \quad \text{// slope of line}
\]

  \[
y = y_0
\]

  for \(x = \text{round}(x_0)\) to \(\text{round}(x_1)\)

  writepixel\((x, \text{Round}(y), \text{rgbValue})\)

  \(y = y + m\)

---

**Scan converting a Polygon (Sketch only)**

- \(y_{\text{min}} = \text{round} \left( \min \text{ of } y \text{ values of vertices} \right)\)
- \(y_{\text{max}} = \text{round} \left( \max \text{ of } y \text{ values of vertices} \right)\)

  for \(y = y_{\text{min}}\) to \(y_{\text{max}}\)

  compute intersection of polygon edges with row \(y\)

  fill in pixels between adjacent pairs of edges

  i.e. \((x, y)\) to \((x', y)\), \((x'', y)\) to \((x''', y)\), ...

  where \(x < x' < x'' < x''' < ...\)