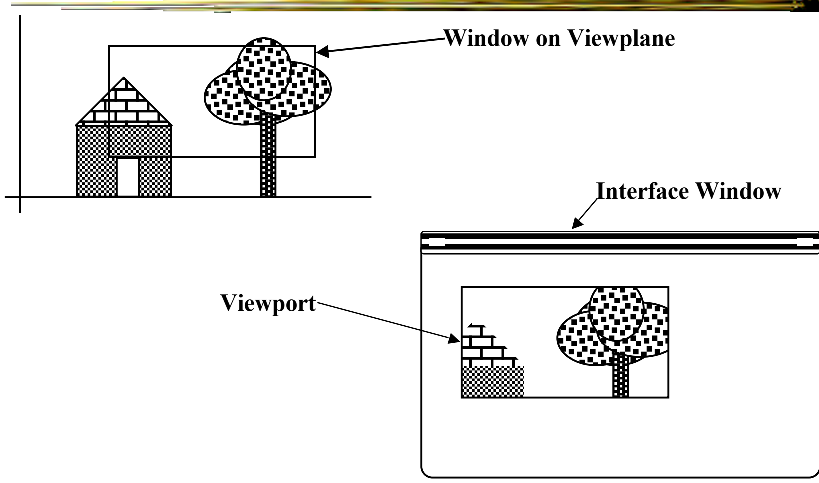
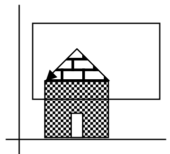
3-D Clipping and other things

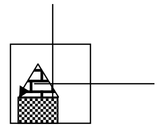
Aside: Windows and Viewports



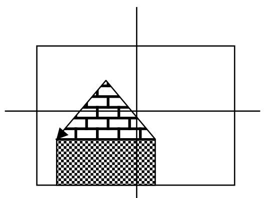
Window to Viewport Xform



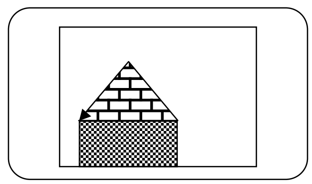
Choose Window in
World Coordinates



Xform window to Canonical View
& Clip



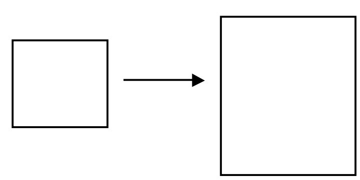
Scale to size of
Viewport



Translate to proper position
on screen (Interface Window)

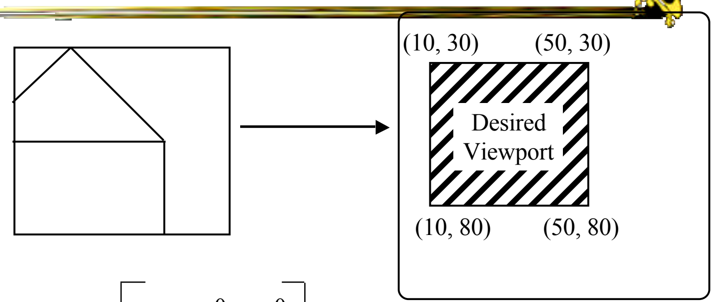
Notes on Window to Viewport

- Inverse relationship between window and viewport
  - As the window increases in size, the image in the viewport decreases in size and vice versa



- Beware of aspect ratio

Example



$$\begin{bmatrix} 1 & 0 & \\ 0 & 1 & \\ 0 & 0 & 1 \end{bmatrix}$$

2) Translate to proper coordinates

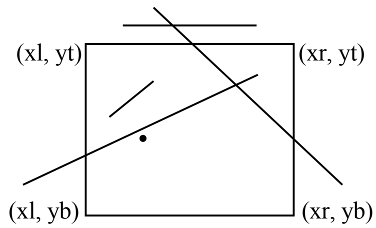
$$\begin{bmatrix} & 0 & 0 \\ 0 & & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

1) Scale to correct size

Culling Revisited

- Trivial accept/reject of whole objects
  - Fast, simple approach
    - Maintain bounding sphere
    - Fast, simple check against canonical volume
- Only clip polygons of remaining objects
  - Can't skip clipping of trivially accepted objects in practice

2D Clipping



A **point** is visible if

$xl < X < xr$
and
$yb < Y < yt$

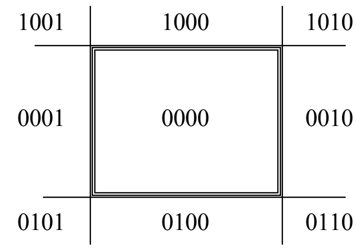
- Line visible if both endpoints in window
- "Brute Force Method"
  - Solve simultaneous equations for intersections of lines and edges

Cohen-Sutherland Algorithm

- Region Checks: Trivially reject or accept lines and points
- Fast for large windows (everything is inside) and for small windows (everything is outside)
- 4-bit outcodes:
  - Bit 1 <-- sign bit of (yt-Y) -- point is above window
  - Bit 2 <-- sign bit of (Y-yb) -- point is below window
  - Bit 3 <-- sign bit of (xr-X) -- point is to right of window
  - Bit 4 <-- sign bit of (X-xl) -- point is to left of window

Cohen-Sutherland Clipping (cont.)

Bit 1: Above
Bit 2: Below
Bit 3: Right
Bit 4: Left

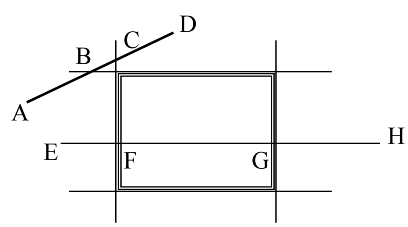


- Trivially accept a line if:

- Trivially reject a line if:

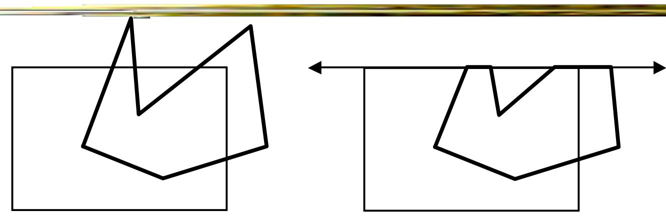
Clipping Lines Not Accepted or Rejected ("divide and conquer")

Example:



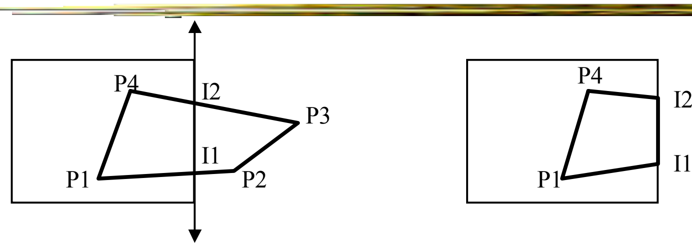
Line AD:
1) Test outcodes of A and D --> can't accept or reject.
2) Calculate intersection point B, which is conceptually on the window side of the dividing line. Form new line segment AB and discard the rest of the line because it is above the window.
3) Test outcodes of A and B. Reject.

Sutherland-Hodgman Polygon Clipping



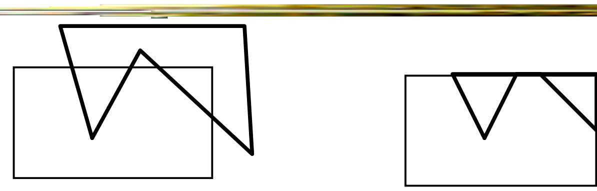
- Clip against each edge of the window one edge at a time
- New set of vertices after each clip
  - The number of vertices usually changes and will often increase.

Polygon Clipping Algorithm



- Window determines a visible and invisible region
- Edge from i to i+1 one of four types:
  - Exit visible region - save the intersection
  - Wholly outside visible region - save nothing
  - Enter visible region - save intersection and endpoint
  - Wholly inside visible region - save endpoint

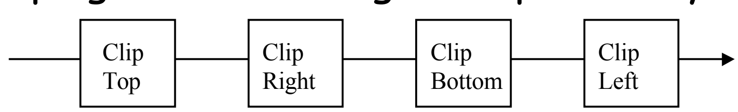
Polygon clipping issues



- Final output, if any, is always considered a single polygon
  - Might be multiple pieces
- Extra edge may not be a problem
  - Always occurs on a window boundary
  - Can be eliminated if necessary

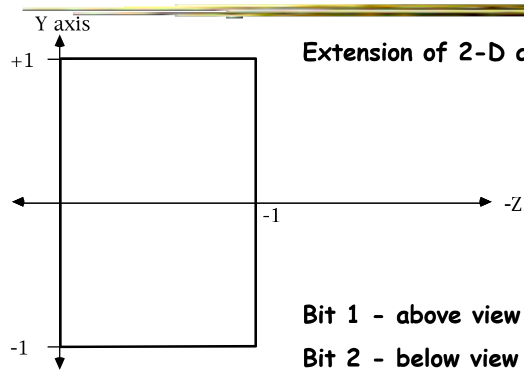
Pipelined Polygon Clipping

- Clip against each edge independently



- Arrange clipping stages in a pipeline
  - Input polygon clipped against one edge
  - Retained points passed to next stage
- Can avoid intermediate storage

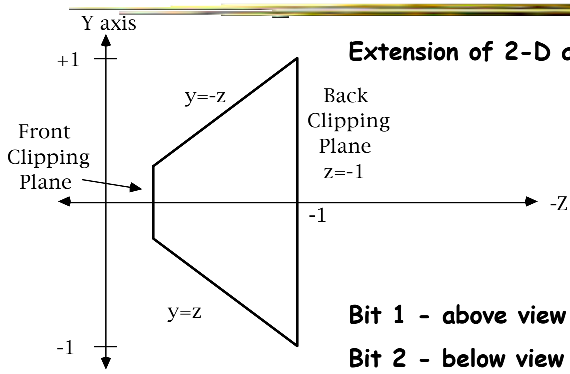
3D Canonical Parallel View Volume



Extension of 2-D algorithm, 6-bit outcode

Bit 1 - above view volume	**y > 1**
Bit 2 - below view volume	**y < -1**
Bit 3 - right of view volume	**x > 1**
Bit 4 - left of view volume	**x < -1**
Bit 5 - behind view volume	**z < -1**
Bit 6 - front of view volume	**z > 0**

3D Canonical Perspective View Volume



Extension of 2-D algorithm, 6-bit outcode

Bit 1 - above view volume	**y > -z**
Bit 2 - below view volume	**y < z**
Bit 3 - right of view volume	**x > -z**
Bit 4 - left of view volume	**x < z**
Bit 5 - behind view volume	**z < -1**
Bit 6 - front of view volume	**z > zmin**

Canonical View Volume

- Trivially accept
  - Both endpoints have a code of all zeros
- Trivially rejected
  - logical AND of the codes is not all zeros.
- Otherwise Calculate intersections.

Intersection Calculation (Perspective Volume)

On the y = z plane

From parametric equation of the line:

$$y0 + t(y1 - y0) = z0 + t(z1 + z0)$$

Solve for t

$$t = (z0 - y0) / ((y1 - y0) - (z1 - z0))$$

Calculate x and y

Already know z = y

Clipping in Homogeneous Coordinates

- Two reasons:
  - Efficiency
  - Correctness

Picking

- Goal: To use the mouse (2D) to select 3D objects
- Analytical method
  - gluUnproject
  - expensive

What are we trying to find?

- The objects that lie on the line that projects to the mouse position

Screen corresponds to Canonical View Volume

- What sliver lies under the mouse?

Scale Sliver to Screen: gluPickMatrix

- After Viewing Transform
- Before Clipping

How to know what gets drawn?

- OpenGL Selection Modes (Picking and Feedback) (chapter 13)
  - glRenderMode, glSelectBuffer
- Add "names" to rendering stream
  - glInitNames, glLoadName, glPushName, glPopName