

## Windows, Viewports, and Clipping

---

---


---

---

---

---

---



## Terminology

- **World Coordinate System** (Object Space): The space in which the application model is defined. The representation of an object is measured in some physical or abstract units.
- **Screen Coordinate System** (Image Space): The space in which the image is displayed. Usually measured in pixels, but could use any units.

(C) Doug Bowman, Virginia Tech, 2008 2

---

---


---

---

---

---

---



## Terminology (cont.)

- **(World) Window**: The rectangle defining the part of the world we wish to display
- **(Screen) Window**: The visual representation of the screen coordinate system for "windowed" displays (coordinate system moves with screen window)
- **Viewport**: The rectangle within the screen window defining where the image will appear (Default is usually entire interface window)

(C) Doug Bowman, Virginia Tech, 2008 3

---

---

---

---

---

---

---



## Terminology (cont.)

- **Window-Viewport Transformation:** The process of mapping from a window in world coordinates to a viewport in screen coordinates

(C) Doug Bowman, Virginia Tech, 2008

4

---

---

---

---

---

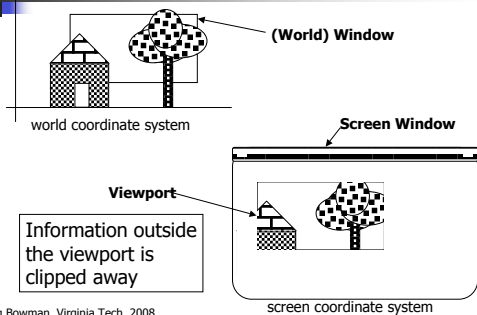
---

---

---



## Windows and Viewports



(C) Doug Bowman, Virginia Tech, 2008

5

---

---

---

---

---

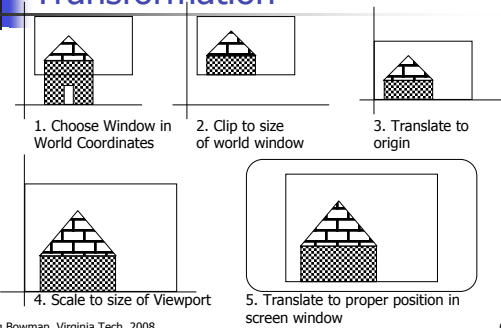
---

---

---



## Window-Viewport Transformation



(C) Doug Bowman, Virginia Tech, 2008

6

---

---

---

---

---

---

---

---

## Window-Viewport Transformation Example

For a point  $(x, y)$ :

3. Translate window to origin:  $x' = x - 10$ ;  $y' = y - 5$
4. Scale to size of viewport:  $x'' = (5/4)x'$ ;  $y'' = 2y'$
5. Translate to viewport pos.:  $x''' = x'' + 25$ ;  $y''' = y'' + 25$

(C) Doug Bowman, Virginia Tech, 2008 7

---

---

---

---

---

---

---

---

## Window-Viewport Transformation Example

For a point  $(x, y)$ :

$$x''' = (5/4)(x - 10) + 25$$

$$y''' = 2(x - 5) + 25$$

$(10, 5)$	-->	$(25, 25)$
$(50, 30)$	-->	$(75, 75)$
$(30, 17.5)$	-->	$(50, 50)$

(C) Doug Bowman, Virginia Tech, 2008 8

---

---

---

---

---

---

---

---

## Notes on W-V Transformation

- **Panning:** Moving the window about the world
- **Zooming:** Reducing or increasing the window size
- As the window increases in size, the image in the viewport decreases in size and vice versa
- Beware of aspect ratio.

4:3

→

2:3

(C) Doug Bowman, Virginia Tech, 2008 9

---

---

---

---

---

---

---

---

## W-V transformation in OpenGL

- `gluOrtho2d(left, right, bottom, top)` takes care of it for you!
  - ex: if my window is 5 meters by 4 meters and has a lower-left corner at (2, 0) in world coordinates:  
`gluOrtho2D(2, 7, 0, 4);` will automatically do the transformation for me
- For generality, also need `glViewport(x, y, width, height)`, but the default viewport uses the whole interface window.

(C) Doug Bowman, Virginia Tech, 2008

10

---

---

---

---

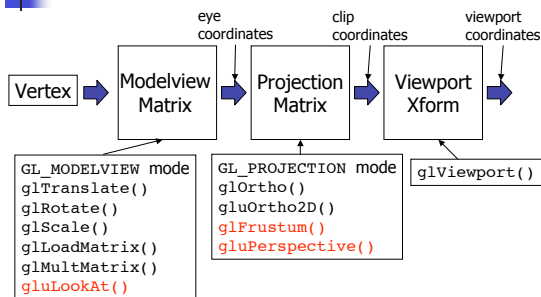
---

---

---

---

## Sequence of OpenGL transformations



(C) Doug Bowman, Virginia Tech, 2008

11

---

---

---

---

---

---

---

---

## Reshaping the window

- We can reshape and resize the OpenGL display window by pulling the corner of the window
- What happens to the display?
  - Must redraw from application
  - Two possibilities
    - Display part of world
    - Display whole world but force to fit in new window
      - **Can alter aspect ratio**

(C) Doug Bowman, Virginia Tech, 2008

12

---

---

---

---

---

---

---

---

## Reshape possibilities

The diagram illustrates the concept of window reshaping. On the left, a square window labeled 'original' contains a green smiley face. Two yellow arrows point from this window to a group of three rectangular windows on the right labeled 'reshaped'. The top rectangle is tall and narrow, the middle one is wide and short, and the bottom one is very short and wide. Each reshaped window contains a portion of the smiley face, demonstrating how the aspect ratio of the window affects the visible content.

(C) Doug Bowman, Virginia Tech, 2008 13

---

---

---

---

---

---

---

---

## The Reshape callback

```
glutReshapeFunc (myreshape)
void myreshape (int w, int h)
```

- Gets width and height of new window (in pixels) as parameters
- A redisplay is posted automatically at end of execution of the callback
- GLUT has a default reshape callback but you probably want to define your own
- The reshape callback is good place to put viewing functions because it is invoked when the window is first opened

(C) Doug Bowman, Virginia Tech, 2008 14

---

---

---

---

---

---

---

---

## Example Reshape

- This reshape preserves shapes by making the viewport and world window have the same aspect ratio

```
void myReshape(int w, int h)
{
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION); /* switch matrix mode */
    glLoadIdentity();
    if (w <= h)
        gluOrtho2D(-2.0, 2.0, -2.0 * (GLfloat) h / (GLfloat) w,
            2.0 * (GLfloat) h / (GLfloat) w);
    else gluOrtho2D(-2.0 * (GLfloat) w / (GLfloat) h, 2.0 *
        (GLfloat) w / (GLfloat) h, -2.0, 2.0);
    glMatrixMode(GL_MODELVIEW); /* return to modelview mode */
}
```

(C) Doug Bowman, Virginia Tech, 2008 15

---

---

---

---

---

---

---

---

## OpenGL example for windows, viewports, and reshaping



---

---

---

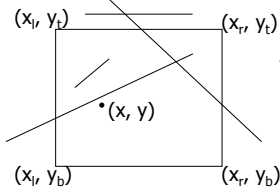
---

---

---

---

## Clipping Lines



A **point** is visible if

$$x_l < x < x_r \\ \text{and} \\ y_b < y < y_t$$

- A **line** is completely visible if both of its end points are in the window.
- **Brute Force Method:** Solve simultaneous equations for intersections of lines with window edges.

(C) Doug Bowman, Virginia Tech, 2008

17

---

---

---

---

---

---

---

## 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)
- Each vertex is assigned a four-bit **outcode**

(C) Doug Bowman, Virginia Tech, 2008

18

---

---

---

---

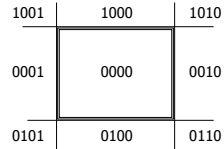
---

---

---

## Cohen-Sutherland Clipping (cont.)

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



- Bit 1: 1 if  $y > y_{tr}$  else 0
- Bit 2: 1 if  $y < y_{br}$  else 0
- Bit 3: 1 if  $x > x_{rr}$  else 0
- Bit 4: 1 if  $x < x_{lr}$  else 0

(C) Doug Bowman, Virginia Tech, 2008

19

---

---

---

---

---

---

---

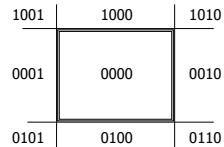
---

---

---

## Cohen-Sutherland Clipping (cont.)

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



- A line can be trivially **accepted** if both endpoints have an outcode of 0000.
- A line can be trivially **rejected** if any corresponding bits in the two outcodes are both equal to 1. (This means that both endpoints are to the right, to the left, above, or below the window.)
- if (outcode 1 & outcode 2) != 0000, trivially reject!

(C) Doug Bowman, Virginia Tech, 2008

20

---

---

---

---

---

---

---

---

---

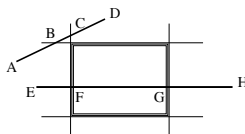
---

## Clipping Lines Not Accepted or Rejected

- In the case where a line can be neither trivially accepted nor rejected, the algorithm uses a "divide and conquer" method.

Line AD:

- Test outcodes of A and D --> can't accept or reject.
- Calculate intersection point B, which is on the dividing line between the window and the "above" region. Form new line segment AB and discard BD because above the window.
- Test outcodes of A and B. Reject.



Line EH: ??

(C) Doug Bowman, Virginia Tech, 2008

21

---

---

---

---

---

---

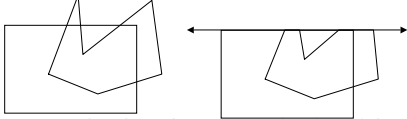
---

---

---

---

## Polygon Clipping



- Polygons can be clipped against each edge of the window one edge at a time. Window/edge intersections, if any, are easy to find since the X or Y coordinates are already known.
- Vertices which are kept after clipping against one window edge are saved for clipping against the remaining edges. Note that the number of vertices usually changes and will often increase.

(C) Doug Bowman, Virginia Tech, 2008

22

---

---

---

---

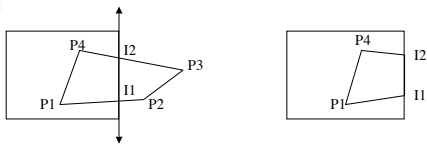
---

---

---

---

## Polygon Clipping Algorithm



- The window boundary determines a visible and invisible region.
- The edge from vertex  $i$  to vertex  $i+1$  can be 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

(C) Doug Bowman, Virginia Tech, 2008

23

---

---

---

---

---

---

---

---

## Polygon clipping issues



- The final output, if any, is always considered a single polygon.
- The spurious edge may not be a problem since it always occurs on a window boundary, but it can be eliminated if necessary.

(C) Doug Bowman, Virginia Tech, 2008

24

---

---

---

---

---

---

---

---



## Pipelined Polygon Clipping



- Because polygon clipping does not depend on any other polygons, it is possible to arrange the clipping stages in a **pipeline**. the input polygon is clipped against one edge and any points that are kept are passed on as input to the next *stage* of the pipeline.
- This way four polygons can be at different *stages* of the clipping process simultaneously. This is often implemented in hardware.

---

---

---

---

---

---

---

---