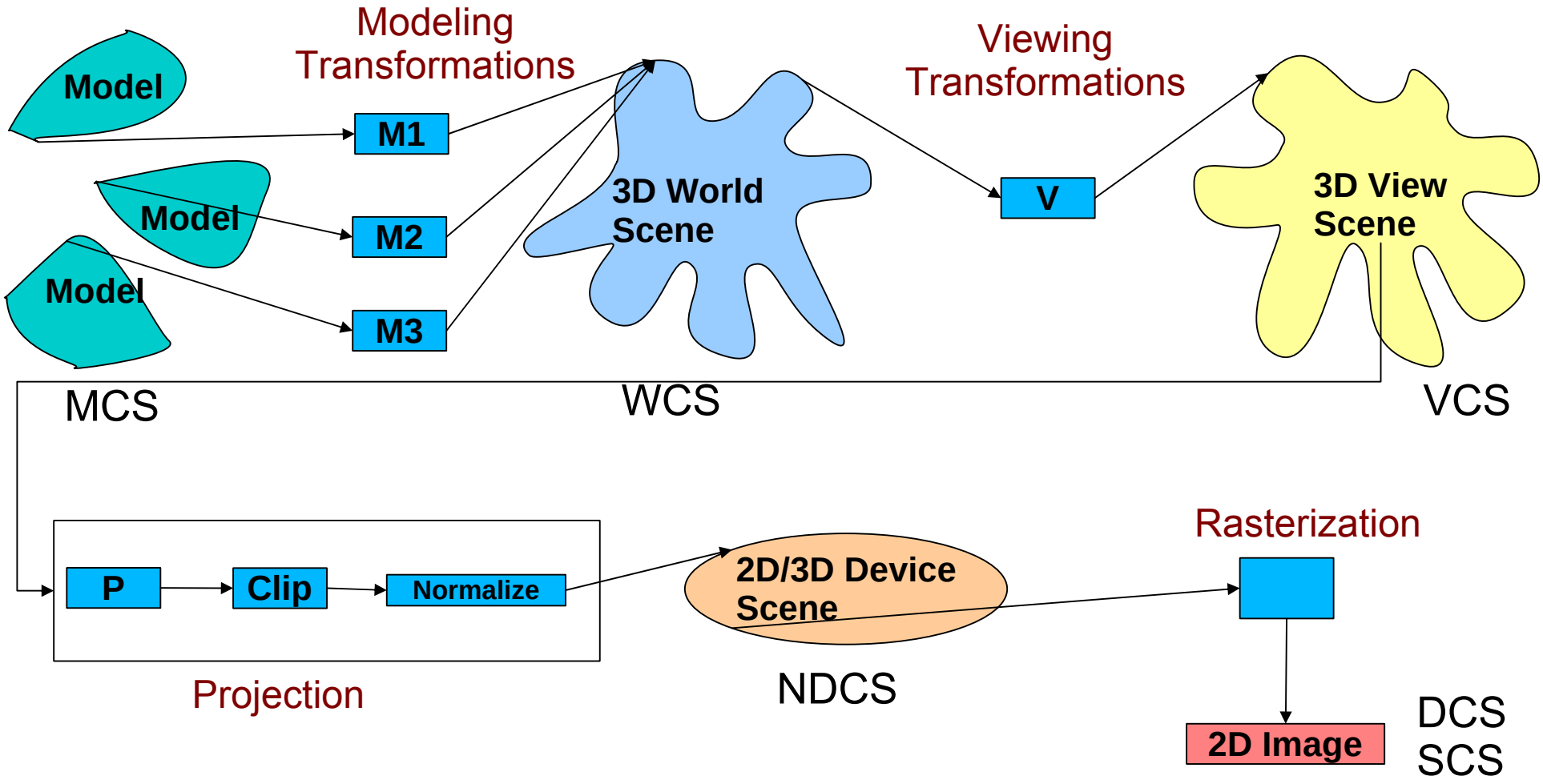
2 DIMENSIONAL VIEWING

Ceng 477
Introduction to Computer Graphics

Computer Engineering
METU

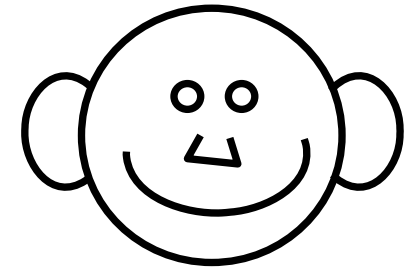
Viewing Pipeline Revisited



- Model coordinates to World coordinates: Modelling transformations

Model coordinates:
1 circle (head),
2 circles (eyes),
1 line group (nose),
1 arc (mouth),
2 arcs (ears).
With their relative coordinates and sizes

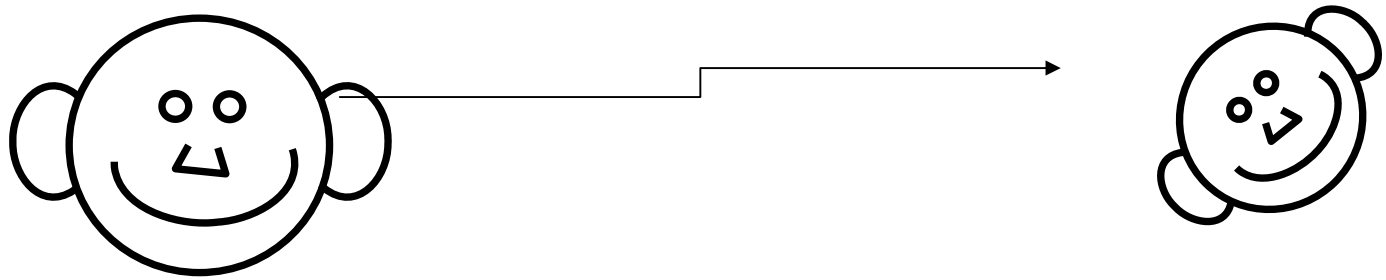
World coordinates:
All shapes with their absolute coordinates and sizes.
circle(0,0,2)
circle(-.6,.8,.3) circle(.6,.8,.3)
lines[(-.4,0),(-.5,-.3),(.5,.3),(.4,0)]
arc(-.6,0,.6,0,1.8,180,360)
arc(-2.2,.2,-2.2,-.2,.8,45,315)
arc(2.2,.2,2.2,-.2,.8,225,135)



- World coordinates to Viewing coordinates: Viewing transformations

World coordinates

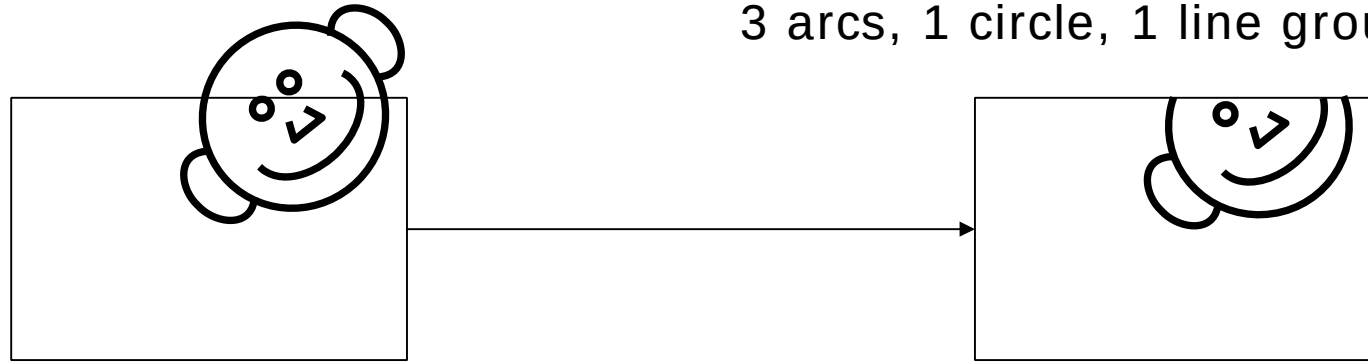
Viewing coordinates:
Viewers position and view angle. i.e. rotated/translated



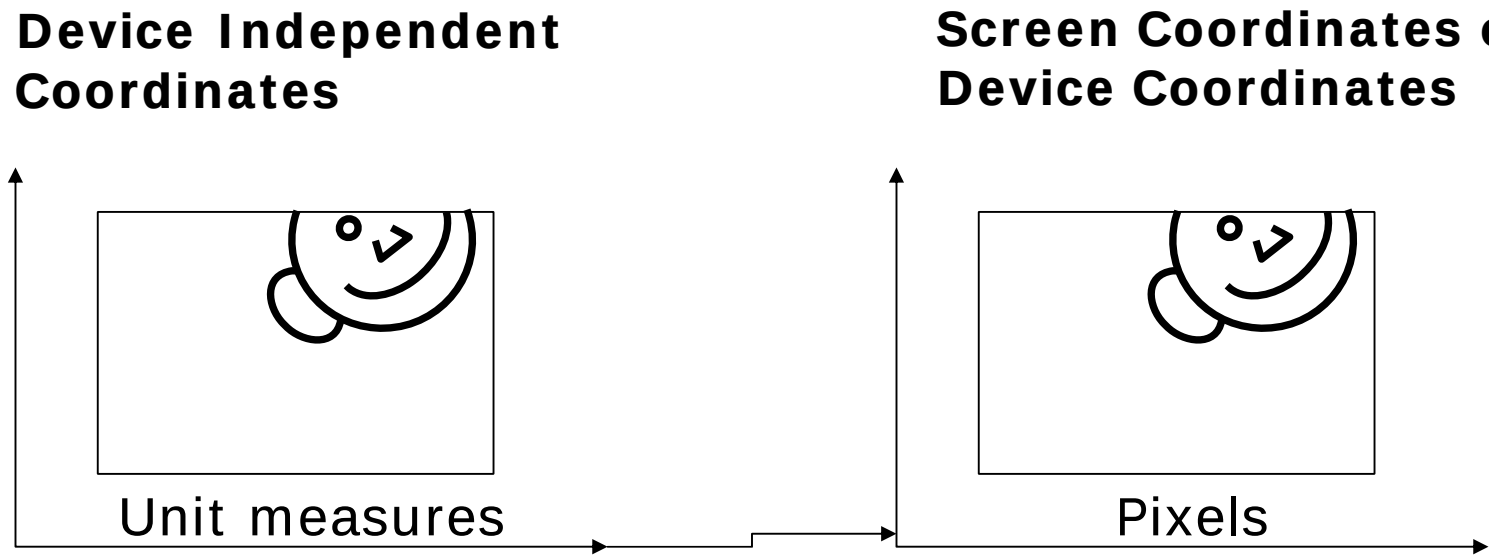
- Projection: 3D to 2D. Clipping depends on viewing frame/volume. Normalization: device independent coordinates

Viewing coordinates:

Device Independent Coordinates:
Invisible shapes deleted, others reduced to visible parts.
3 arcs, 1 circle, 1 line group

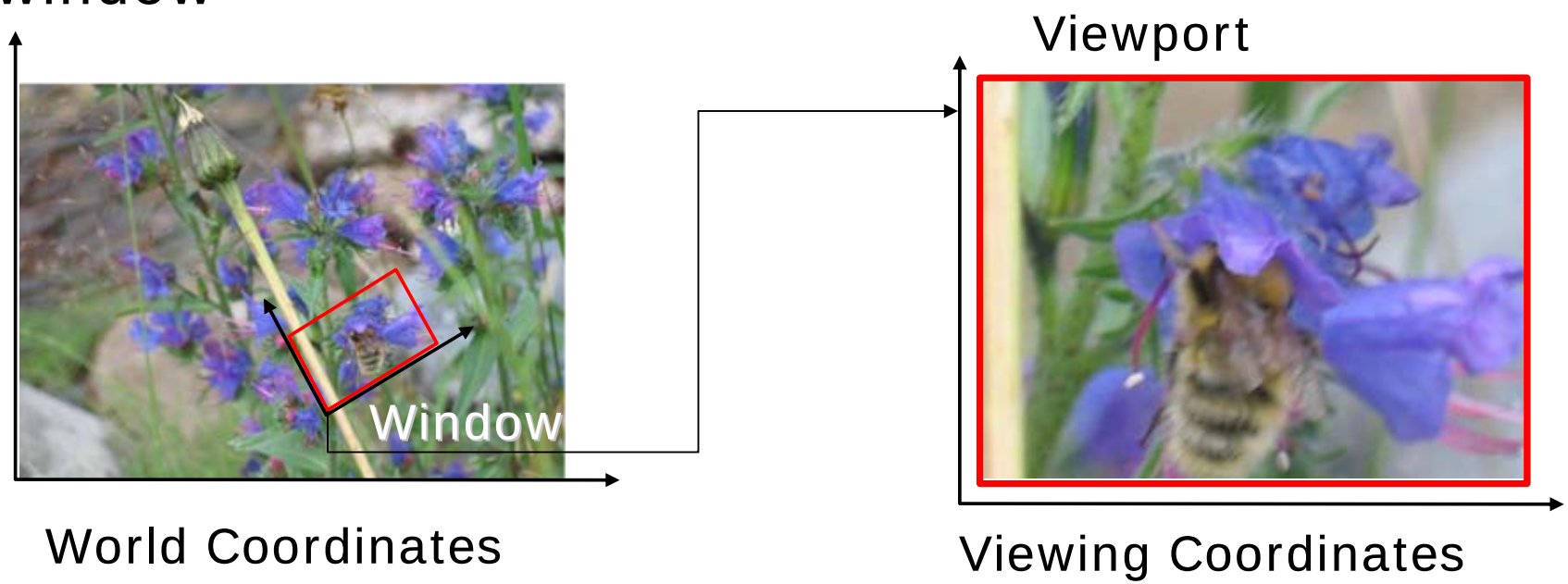


- Device Independent Coordinates to Device Coordinates. Rasterization



2D Viewing

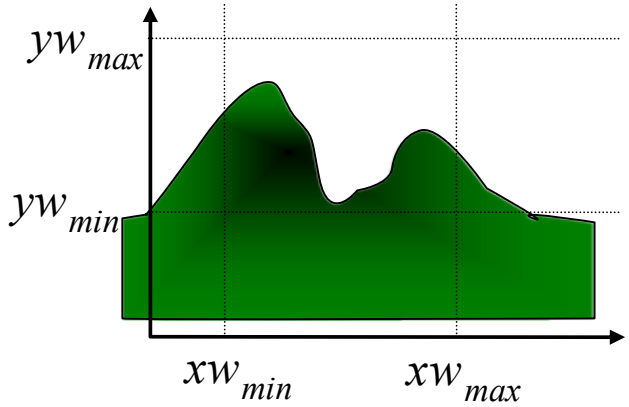
- World coordinates to Viewing coordinates
- Window to Viewport.
  Window: A region of the scene selected for viewing (also called *clipping window*)
  Viewport: A region on display device for mapping to window

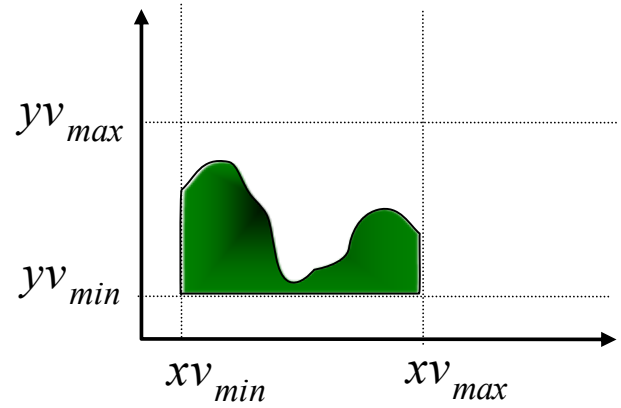


Clipping Window vs. Viewport

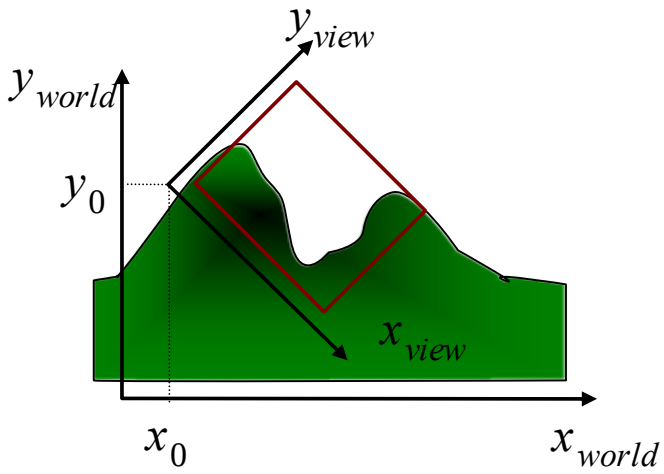
- The clipping window selects ***what*** we want to see in our virtual 2D world.
- The *viewport* indicates where it is to be viewed on the output device (or within the display window)
- By default the *viewport* have the same location and dimensions of the GLUT display window you create
  - But it can be modified so that only a part of the display window is used for OpenGL display

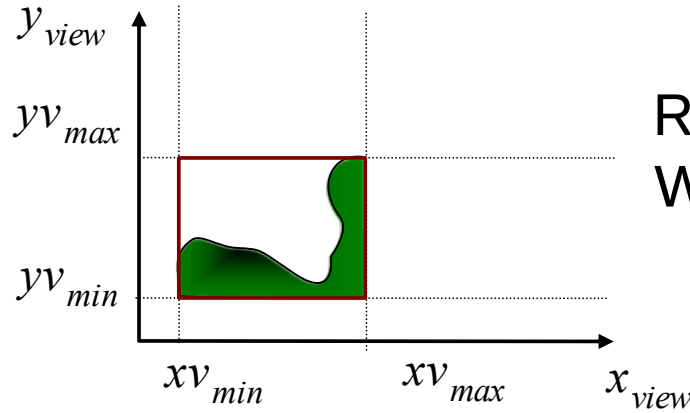
The clipping window





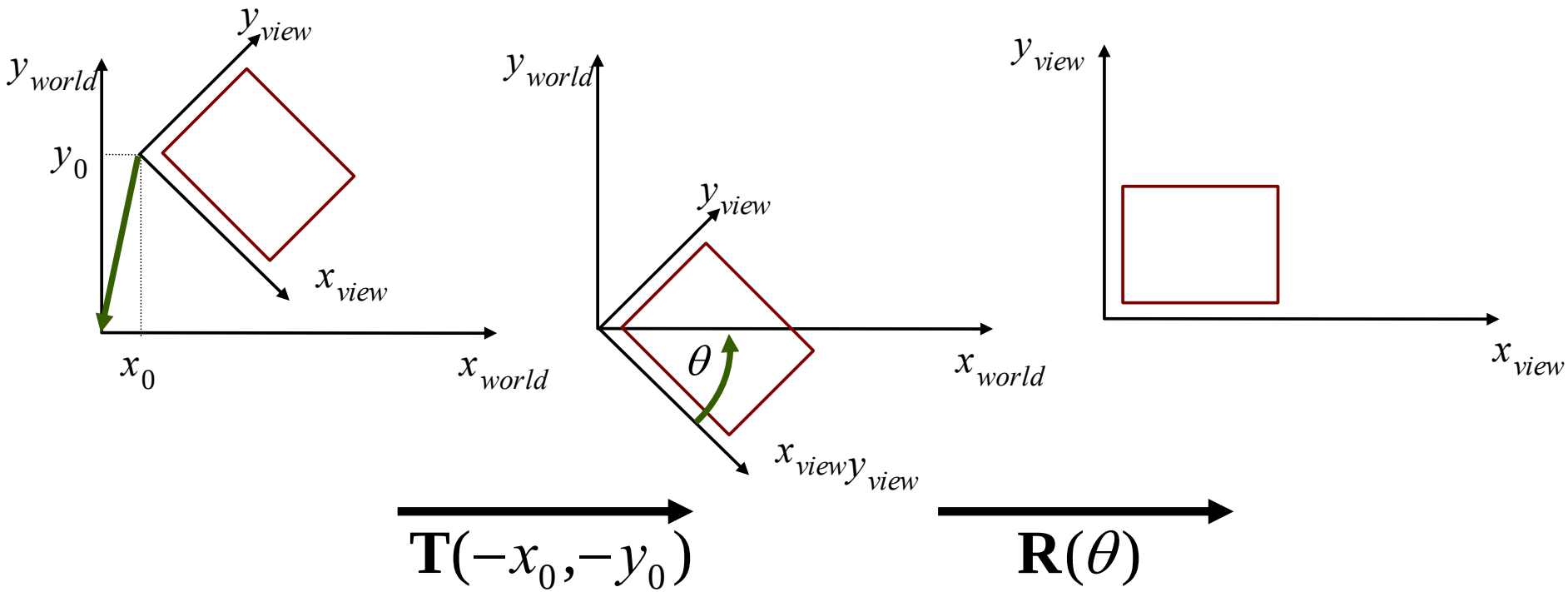
Rectangular Window





Rotated Window

World-coordinates to Viewing Coordinates



- $M_{wc,vc} = R \cdot T$

Normalization

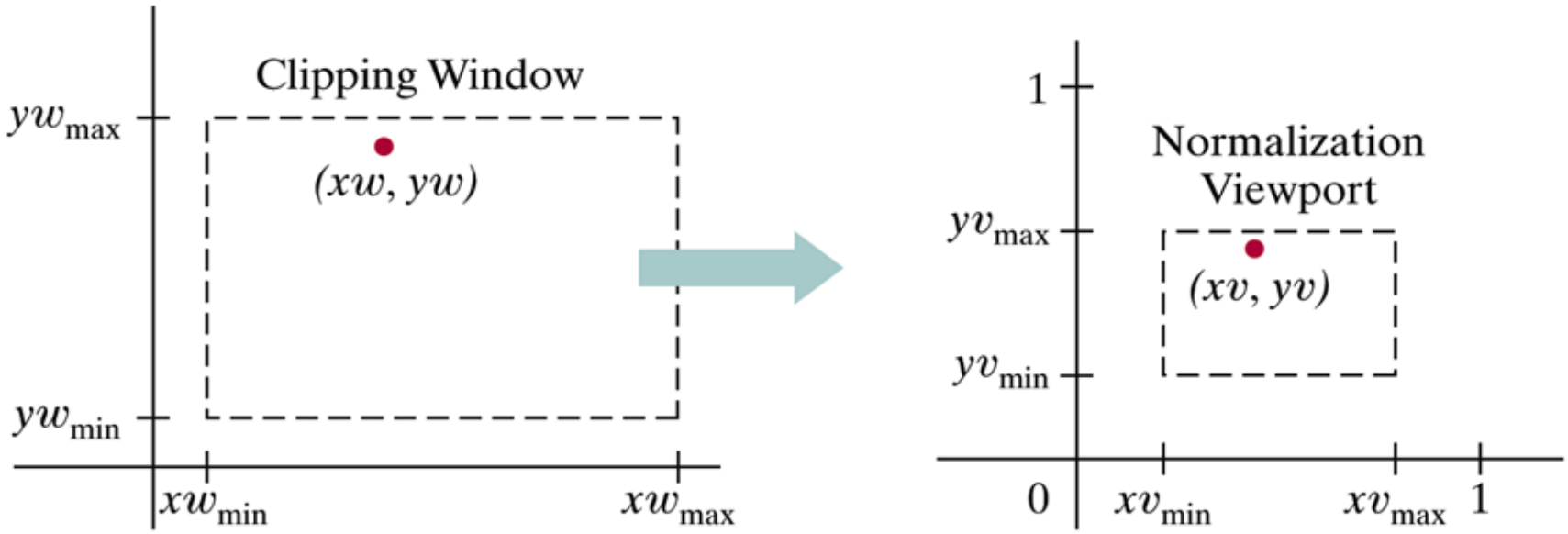
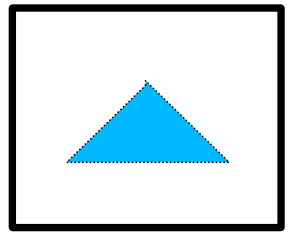
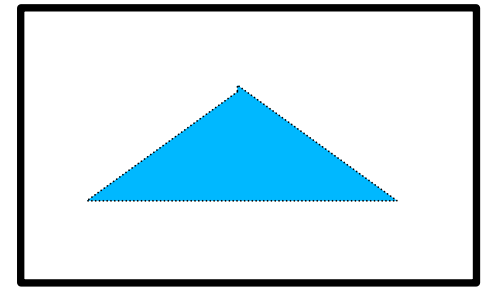


Figure 6-7

A point (xw, yw) in a world-coordinate clipping window is mapped to viewport coordinates (xv, yv), within a unit square, so that the relative positions of the two points in their respective rectangles are the same.

- Coordinate transformation: Different sizes and/or height width ratios?





- For any point:

$$\frac{xv - xv_{min}}{xv_{max} - xv_{min}} = \frac{xw - xw_{min}}{xw_{max} - xw_{min}}$$

$$\frac{yv - yv_{min}}{yv_{max} - yv_{min}} = \frac{yw - yw_{min}}{yw_{max} - yw_{min}}$$

should hold.

$$xv = xv_{min} + (xw - xw_{min})\frac{(xv_{max} - xv_{min})}{xw_{max} - xw_{min}}$$

$$yv = yv_{min} + (yw - yw_{min})\frac{(yv_{max} - yv_{min})}{yw_{max} - yw_{min}}$$

- This can also be accomplished in 2 steps:

1. Scale over the fixed point:

$$\mathbf{S}(xw_{min}, yw_{min}, s_x, s_y)$$

2. Translate lower-left corner of the clipping window to the lower-left corner of the viewport

$$\mathbf{T}(xv_{min} - xw_{min}, yv_{min} - yw_{min})$$

OpenGL 2D Viewing Functions

- OpenGL, GLU, and GLUT provide functions to specify clipping windows, viewports, and display windows within a video screen.

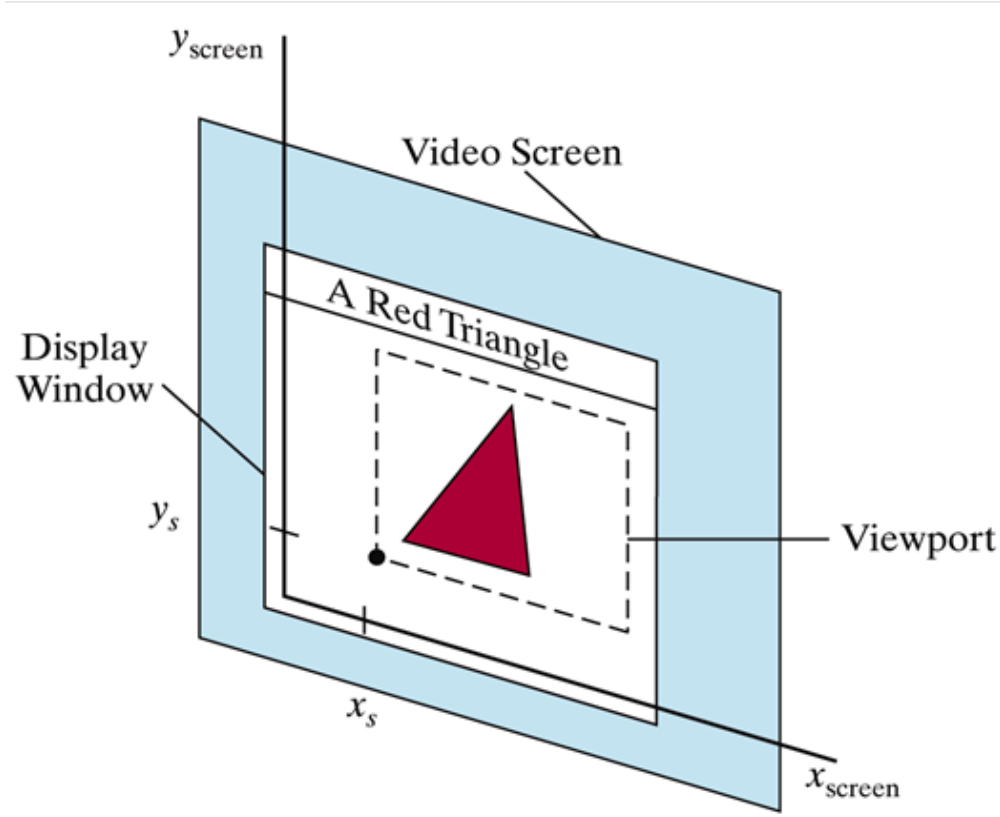


Figure 6-9

A viewport at coordinate position (x_s, y_s) within a display window.

Setting up a 2D Clipping-Window

- glMatrixMode (GL_PROJECTION)
- glLoadIdentity (); // reset, so that new viewing parameters are not combined with old ones (if any)
- gluOrtho2D (xwmin, xwmax, ywmin, ywmax);

or

- glOrtho (xwmin, xwmax, ywmin, ywmax, zwmin, zwmax);

- Objects within the clipping window are transformed to normalized coordinates (-1,1)

Setting up a Viewport

- glViewport (xvmin, yvmin, vpWidth, vpHeight);
- All the parameters are given in integer screen coordinates relative to the lower-left corner of the display window.
- If we do not invoke this function, by default, a viewport with the same size and position of the display window is used (i.e., all of the GLUT window is used for OpenGL display)

Creating a GLUT Display Window

- glutInitWindowPosition (xTopLeft, yTopLeft);
  - the integer parameters are relative to the top-left corner of the screen
- glutInitWindowSize (dwWidth, dwHeight);
- glutCreateWindow (“Title of Display Window”);
- glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB)
  - Specification of the buffer that will be used
- glClearColor (red, green, blue, alpha)
  - Specify the background color

Multiple GLUT windows

- Multiple windows may be created within an OpenGL program
  - Need *window id*s to manage multiple windows
  - windowID = glutCreateWindow(“Window1”);
  - glutDestroyWindow (windowID)

    // to distroy the window
- General functions (like glutInitDisplayMode) are applied to the current display window. We can set the current window to a specific window with:
  - glutSetWindow (windowID);

Other functions

- GLUT provide functions to relocate, resize, minimize, resize to fullscreen, change window title, hide, show, bring to front, or send to back, select a specific cursor for the current display window. (pages 309-311 in the textbook)

GLUT Subwindows

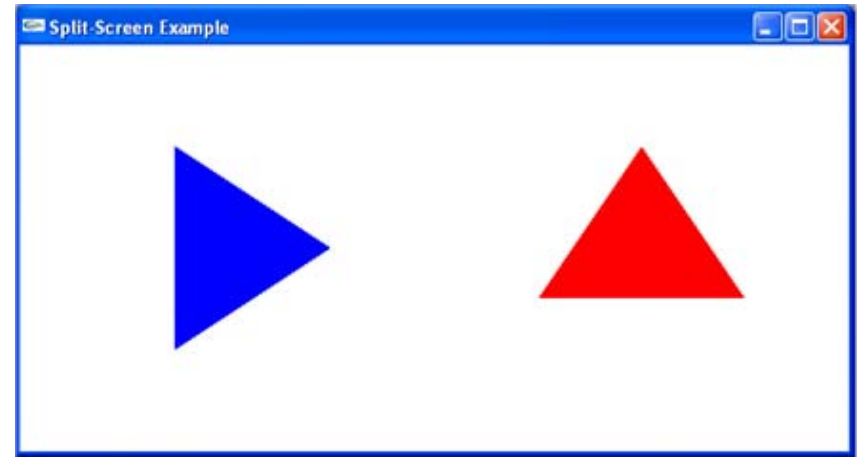
- glutCreateSubWindow (windowID, xBottomLeft, yBottomLeft, width, height);
  - the parameters are given relative to the lower left corner of the display window specified by windowID
- Subwindows also have window ids and we can create subwindows within subwindows

Display callback function

- Each separate GLUT window can have its own function to specify what will be drawn inside. E.g.,
  - glutSetWindow (w1);
  - glutDisplayFunction (wireframeDisplay);
  - glutSetWindow (w2);
  - glutDisplayFunction (solidDisplay);
- Display callback functions are called only when GLUT determines that the display content should be renewed. To update the display manually call: glutPostRedisplay();
- glutIdleFunc (functionName) could be used in animations

OpenGL 2D Viewing Example

- 2 Viewports
- One triangle is displayed in two colors and orientations in 2 viewports



glutInitWindowSize (600, 300);

```
glClear (GL_COLOR_BUFFER_BIT);
glColor3f(0.0, 0.0, 1.0);
glViewport(0, 0, 300, 300);
drawCenteredTriangle();

glColor3f(1.0, 0.0, 0.0);
glViewport(300, 0, 300, 300);
glRotatef(90.0, 0.0, 0.0, 1.0);
drawCenteredTriangle();
```

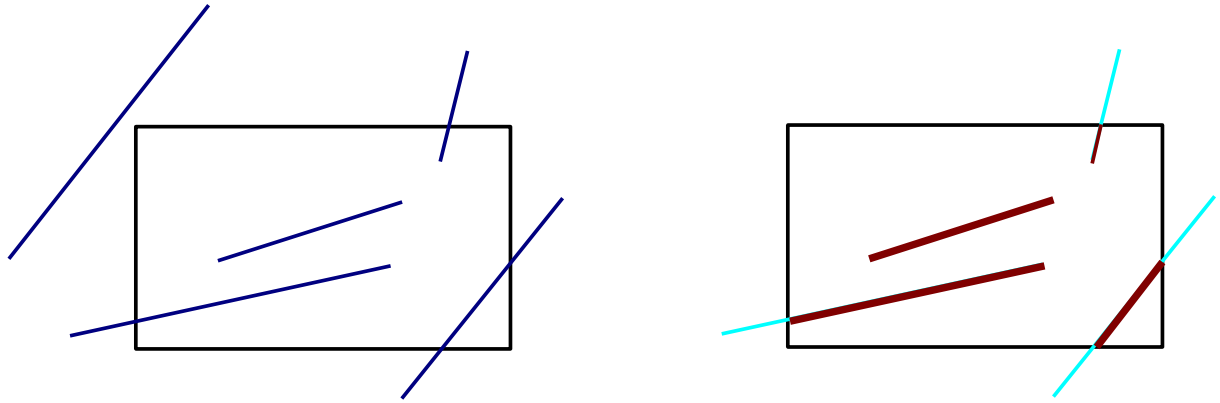
Clipping Algorithms

- Clipping: identifying the parts of the objects that will be inside of the window.
- Everything outside the clipping window is eliminated from the scene description (i.e., not scan converted) for efficiency.
- Point clipping:

$$xw_{min} \le x \le xw_{max}$$

$$yw_{min} \le y \le yw_{max}$$

Line Clipping



- When both end points are inside all four boundaries (completely inside) or both end points are outside any one of the boundaries (completely outside) no extra processing need to be done. Otherwise:
  - For each line determine the intersection with boundaries. Parametric line equations:

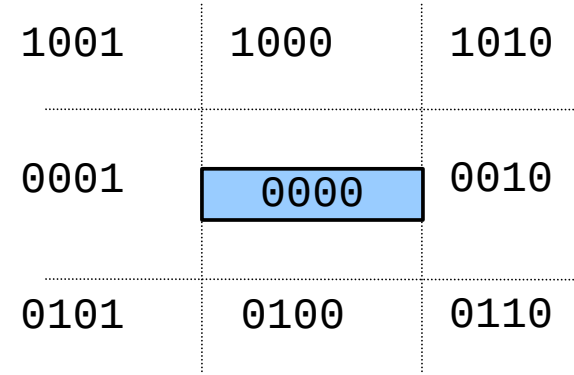
$$x = x_1 + u(x_2 - x_1),$$

$$y = y_1 + u(y_2 - y_1), \quad 0 \le u \le 1$$

Find u for all boundary lines. Not very efficient.

Cohen-Sutherland Line Clipping

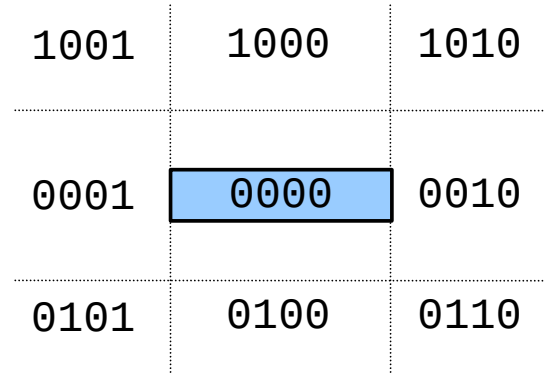
- Based on determination of completely invisible line segments by doing more tests before intersection tests.
- Define test bits for a point:
  bit 1: left of the left border
  bit 2: right of the right border
  bit 3: below the bottom border
  bit 4: above the top border



- When bits (also called *out bits*) are defined for start and end point of the line segment, a single bitwise operation defines the visibility of the line segment. How?

- ```
  #define bits(x,y) ((x<xmin)|(x>xmax)<<1|(y<ymin)<<2|(y>ymax)<<3)

  b1=bits(x1,y1) ; b2=bits(x2,y2);
  if (b1==0 && b2==0) {
  /* both end points inside
     trivial accept          */
  } else if (b1 & b2) {
  /* line is completely outside
     ignore it               */
  } else {
  /* needs further calculation*/
  }
  ```



- Process clipping boundaries in order and clip a section of the line during the process
- Use the bits calculated before for crossing tests (if one end is 1 and the other is 0 then the line crosses the boundary)
- Use slope-intercept line representation for intersection
  ```

Cohen-Sutherland Algorithm

$$m = \frac{(y_2 - y_1)}{(x_2 - x_1)}$$

repeat for $border = \{\text{LEFT, RIGHT, BOTTOM, TOP}\}$

if $\neg(bits(x_1, y_1) \vee bits(x_2, y_2))$

Both inside, accept line and terminate

else if $bits(x_1, y_1) \wedge bits(x_2, y_2)$

Both outside same region, reject line and terminate

if $border = \text{LEFT} \vee border = \text{RIGHT}$

$$y_p = y_1 + m(x_{border} - x_1), x_p = x_{border}$$

else

$$x_p = x_1 + \frac{(y_{border} - y_1)}{m}, y_p = y_{border}$$

if $bordertest(x_1, y_1)$

$$x_1 = x_p; y_1 = y_p$$

else if $bordertest(x_2, y_2)$

$$x_2 = x_p; y_2 = y_p$$

Example

$bits(0,0) = 0101 \quad bits(8,5) = 0010$

$bits(0,0) \wedge bits(8,5) = 0000$

$$m = \frac{(5-0)}{(8-0)} = \frac{5}{8}$$

LEFT : $y_p = y_1 + \frac{5}{8}(1-0) = \frac{5}{8}$

$\text{LEFT}(P_1) \rightarrow P_1' = (1,5/8)$

RIGHT : $y_p = y_1 + \frac{5}{8}(7-1) = \frac{5}{8} + \frac{5}{8}6 = \frac{35}{8}$

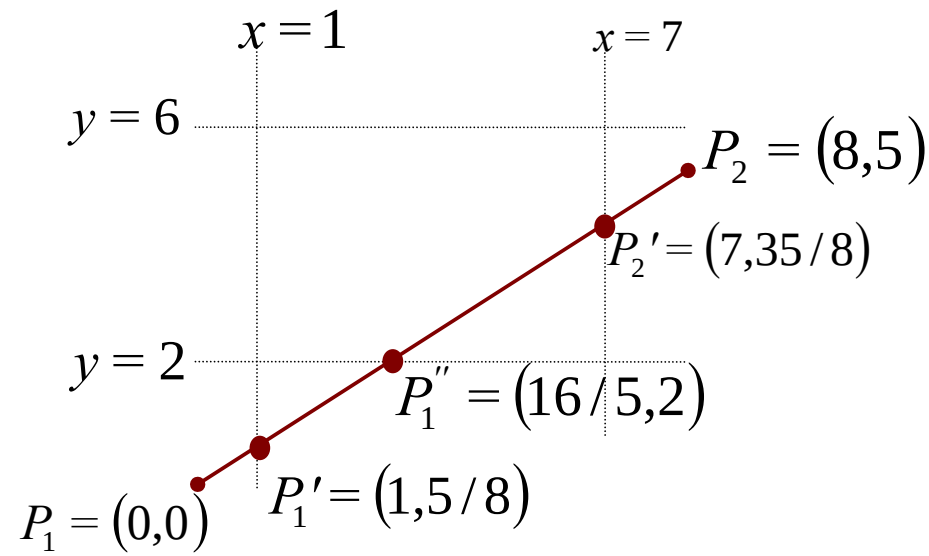
$\text{RIGHT}(P_2) \rightarrow P_2' = (7,35/8)$

BOTTOM : $x_p = 1 + \left(2 - \frac{5}{8}\right)\frac{8}{5} = 1 + \frac{16}{5} - 1 = \frac{16}{5}$

$\text{BOTTOM}(P_1') \rightarrow P_1'' = (16/5,2)$

TOP : P_1'' inside, P_2' inside, so terminate

$L = (16/5,2) \text{ to } (7,35/8)$



Liang-Barsky Line Clipping

- Use parametric equations for efficiency

$$xw_{min} \le x_1 + u\Delta x \le xw_{max}$$
$$yw_{min} \le y_1 + u\Delta y \le yw_{max}, \quad 0 \le u \le 1$$

rewrite these inequalities:

$$up_k \le q_k, \quad k = 1,2,3,4$$

$$p_1 = -\Delta x, \quad q_1 = x_1 - xw_{min}$$
$$p_2 = \Delta x, \quad q_2 = xw_{max} - x_1$$
$$p_3 = -\Delta y, \quad q_3 = y_1 - yw_{min}$$
$$p_4 = \Delta y, \quad q_4 = y_{max} - y_1$$

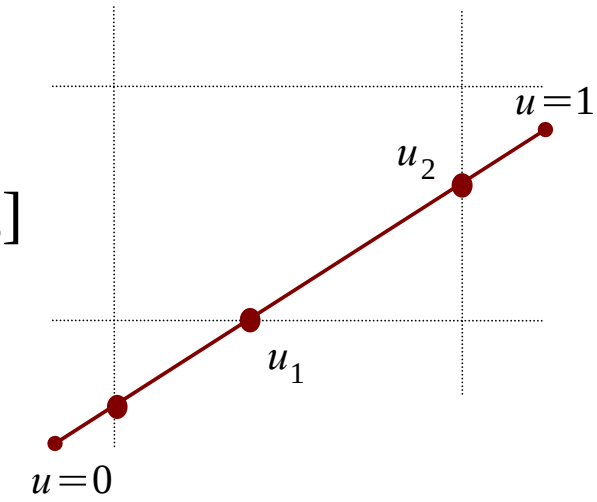
$u = \frac{q_k}{p_k}$ The point where the line intersects the borders

$up_k \le q_k, \quad 0 \le u \le 1$

if $p_k < 0$, outside to inside transition, candidate for u_1

if $p_k > 0$, inside to outside transition, candidate for u_2

- Problem: find the clipping interval $[u_1, u_2]$
- If $p_k<0$ $u_1 = \max(u_1, p_k/q_k)$
  If $p_k>0$ $u_2 = \min(u_2, p_k/q_k)$



Liang-Barsky Algorithm

$u_1 = 0, u_2 = 1$

for $k = 1,2,3,4$

find p_k

if $p_k < 0 \quad r_k = \dfrac{q_k}{p_k}$

$u_1 = max(u_1, r_k)$

else if $p_k > 0 \quad r_k = \dfrac{q_k}{p_k}$

$u_2 = min(u_2, r_k)$

else *parallel line, treat specially*

if $u_1 > u_2$

line is outside, totally reject

$x_2 = x_1 + u_2 \Delta x, y_2 = y_1 + u_2 \Delta y$

$x_1 = x_1 + u_1 \Delta x, y_1 = y_1 + u_1 \Delta y$

Example

$\Delta x = 8, \Delta y = 5$

$k = 1$

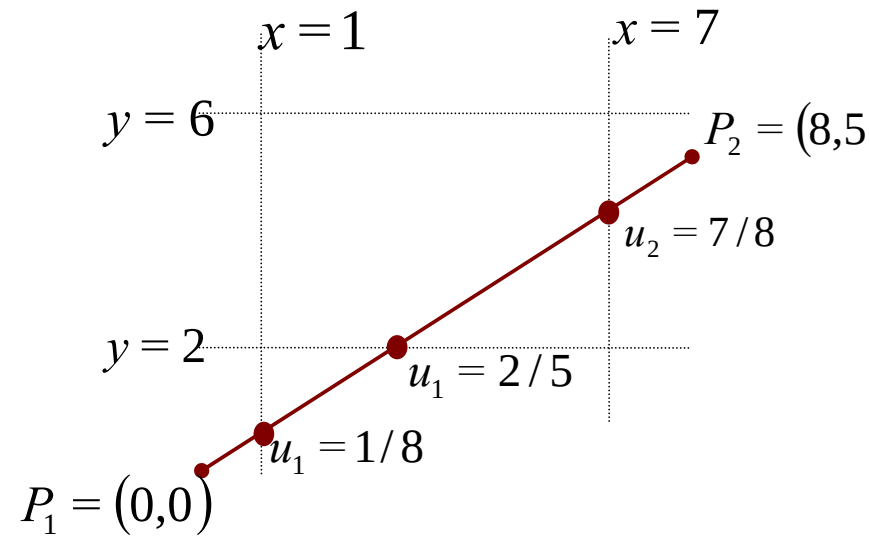
$$p_1 = -8, \quad q_1 = 0 - 1 = -1, \quad r_1 = \frac{1}{8}$$

$$u_1 = max\left\{\frac{1}{8}, 0\right\} = \frac{1}{8}$$

$k = 2$

$$p_2 = 8, \quad q_2 = 7 - 0 = 7, \quad r_2 = \frac{7}{8}$$

$$u_2 = min\left\{\frac{7}{8}, 1\right\} = \frac{7}{8}$$



$k = 3$

$$p_3 = -5, \quad q_3 = 0 - 2 = -2, \quad r_3 = \frac{-2}{-5}$$

$$u_1 = max\left\{\frac{1}{8}, \frac{2}{5}\right\} = \frac{2}{5}$$

$k = 4$

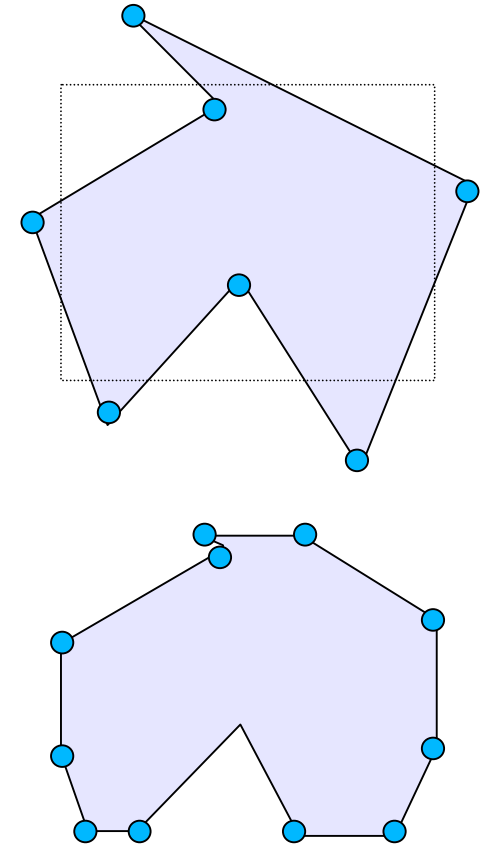
$$p_4 = 5, \quad q_4 = 7 - 0 = 7, \quad r_4 = \frac{7}{5}$$

$$u_2 = min\left\{\frac{7}{5}, \frac{7}{8}\right\} = \frac{7}{8}$$

$$P_1 = (0 + 2/5 \cdot 8, 0 + 2/5 \cdot 5) = (16/5, 2) \quad P_2 = (0 + 7/8 \cdot 8, 0 + 7/8 \cdot 5) = (7, 35/8)$$

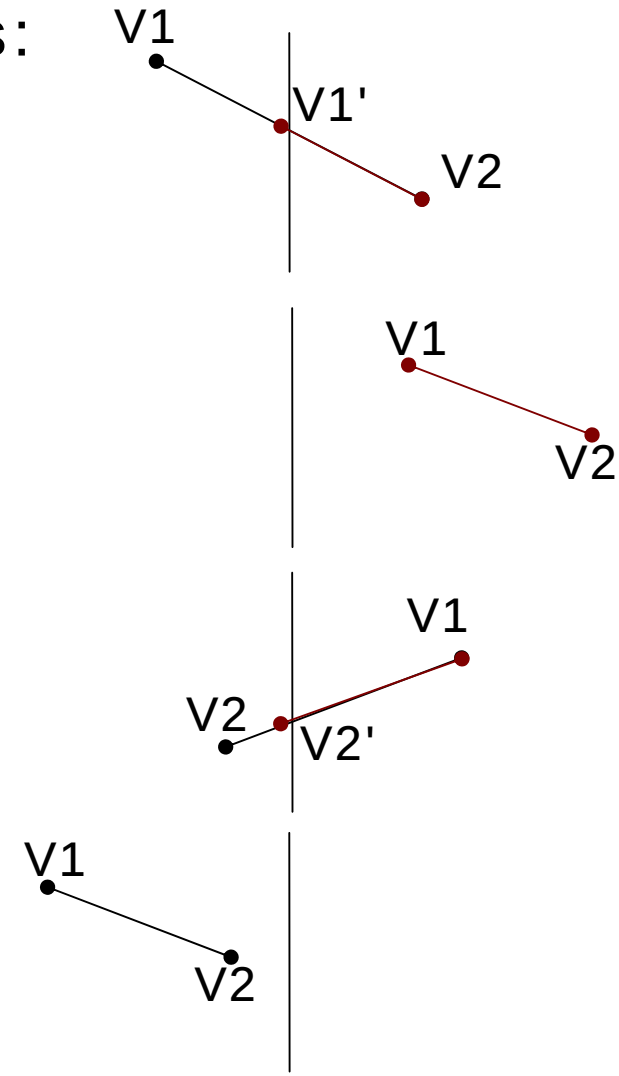
Polygon Clipping

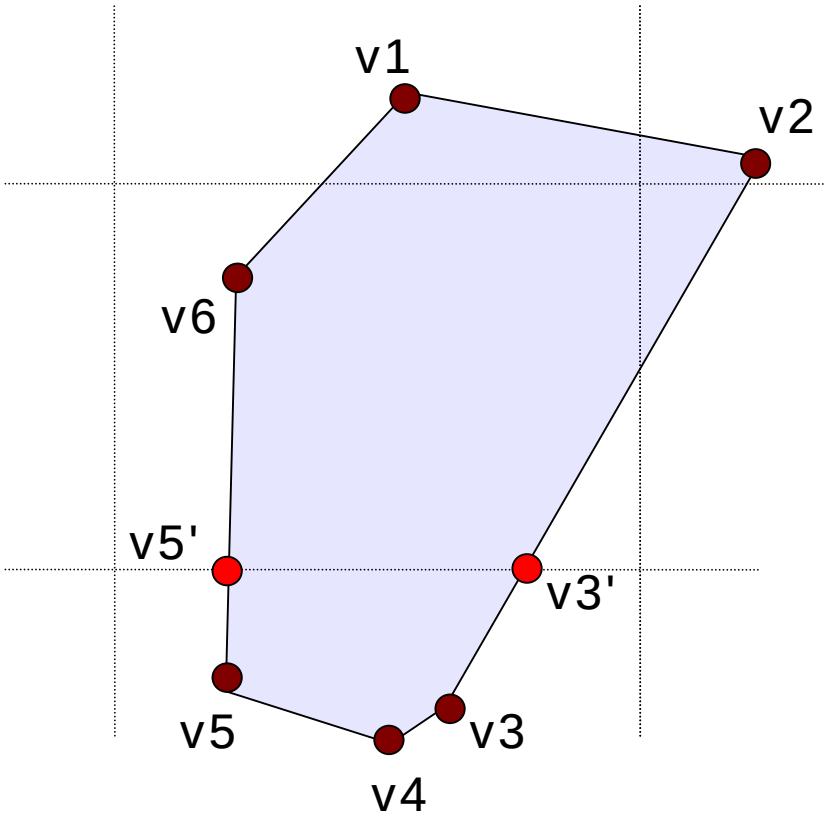
- Find the vertices of the new polygon(s) inside the window.
- Sutherland-Hodgman Polygon Clipping:
  Check each edge of the polygon against all window boundaries. Modify the vertices based on transitions. Transfer the new edges to the next clipping boundary.



Shutherland-Hodgman Polygon Clipping

- Traverse edges for borders; 4 cases:
  - V1 outside, V2 inside: take V1' and V2
  - V1 inside, V2 inside: take V1 and V2
  - V1 inside, V2 outside: take V1 and V2'
  - V1 outside, V2 outside: take none





- Left border:

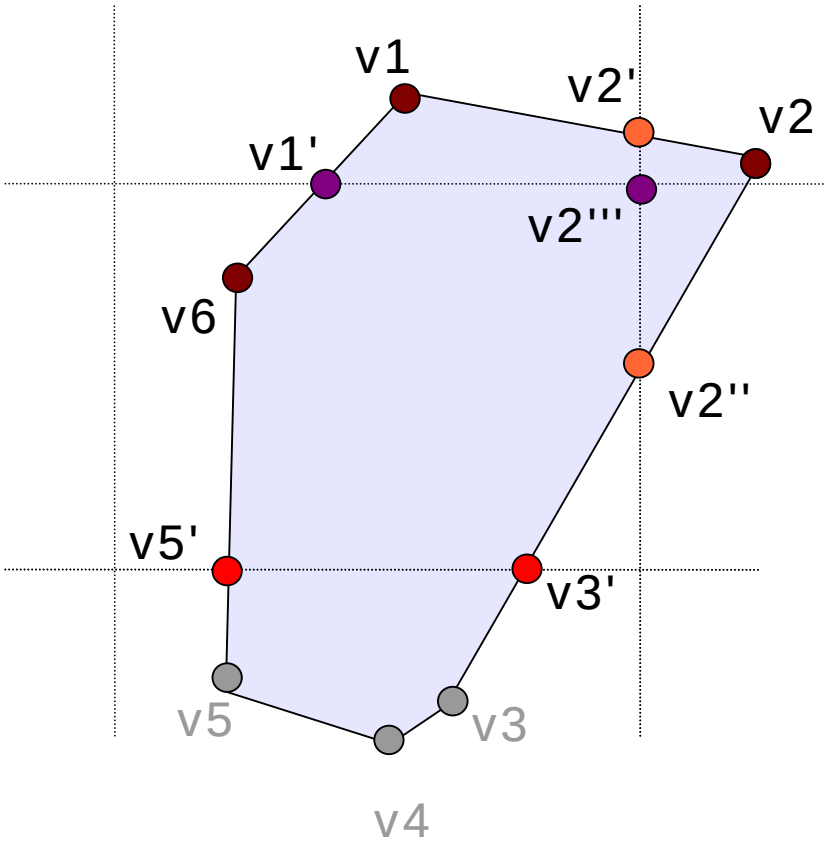
v1 v2	both inside	v1 v2
v2 v3	both inside	v2 v3
.....	" "	.....

v1,v2,v3,v4v5,v6,v1

- Bottom Border:

v1 v2	both inside	v1 v2
v2 v3	v2 i, v3 o	v2 v3'
v3 v4	both outside	none
v4 v5	both outside	none
v5 v6	v5 o, v6 i	v5' v6
v6 v1	both inside	v6 v1

v1,v2,v3',v5',v6,v1



- v1,v2,v3',v5',v6,v1
- Right border:

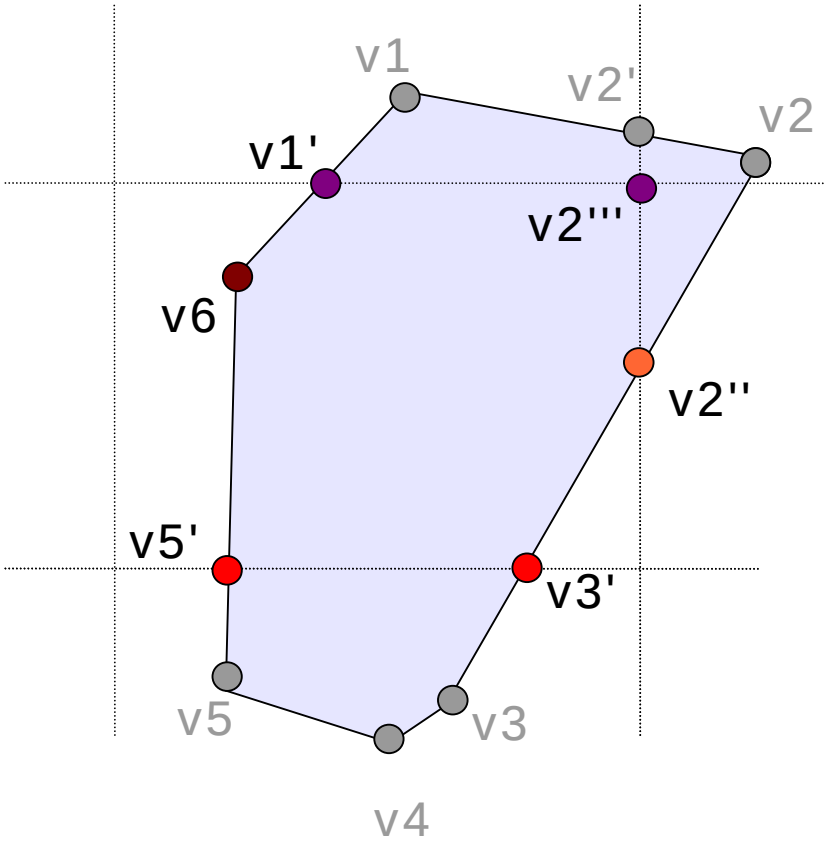
v1 v2	v1 i, v2 o	v1 v2'
v2 v3'	v2 o, v3'i	v2'' v3'
v3' v5'	both inside	v3' v5'
v5' v6	both inside	v5' v6
v6 v1	both inside	v6 v1

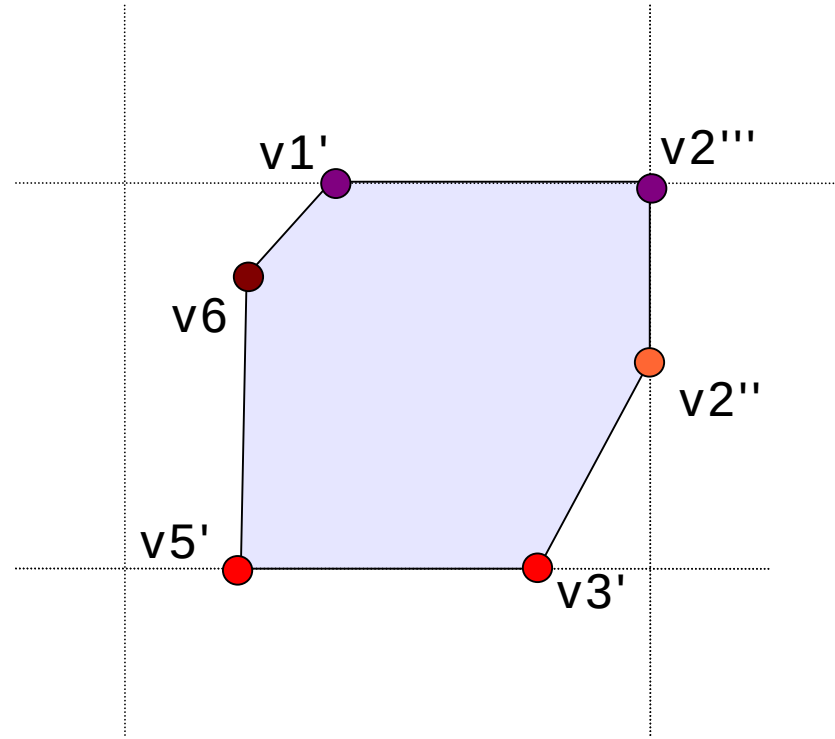
v1,v2',v2'',v3',v5',v6,v1

- Top Border:

v1 v2'	both outside	none
v2' v2''	v2' o, v2'' i	v2''' v2''
v2'' v3'	both inside	v2'' v3'
v3' v5'	both inside	v3' v5'
v5' v6	both inside	v5' v6
v6 v1	v6 i, v1 o	v6 v1'

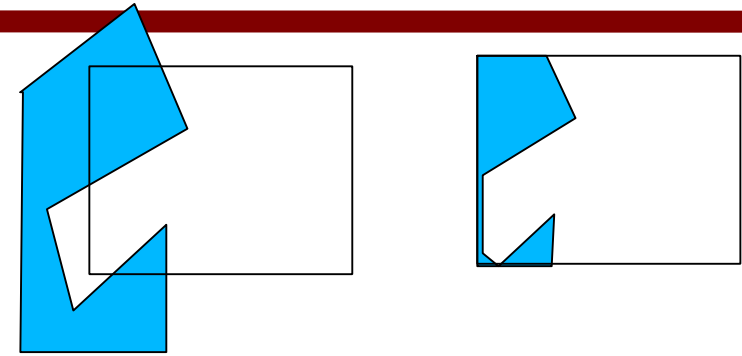
v2''',v2'',v3',v5',v6,v1'


v1
v2'
v2
v1'
v2'''
v6
v2''
v5'
v3'
v5
v3
v4


v1'
v2'''
v6
v2''
v5'
v3'

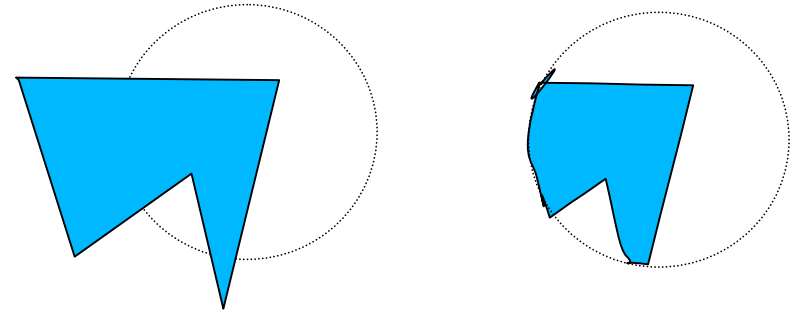
Other Issues in Clipping

- Problem in Shutherland-Hodgman. Weiler-Atherton has a solution



- Clipping other shapes: Circle, Ellipse, Curves.

- Clipping a shape against another shape



- Clipping the interior.