# CMPE 466 COMPUTER GRAPHICS

Chapter 8

2D Viewing

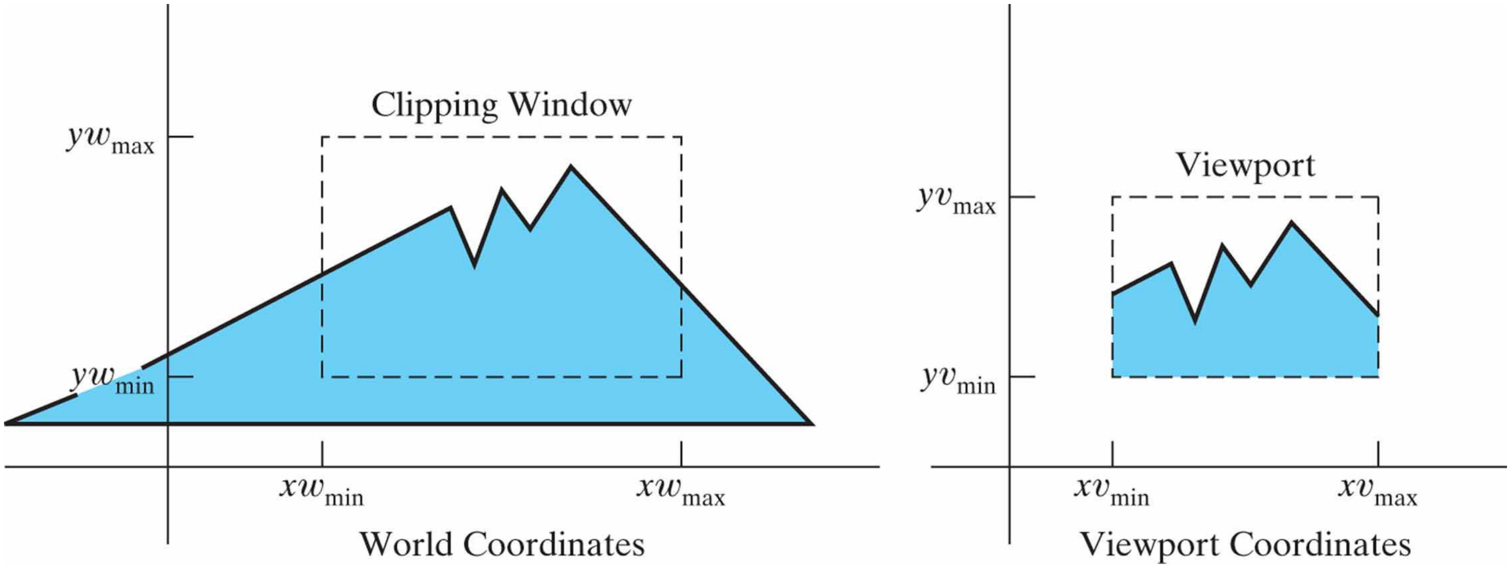
Instructor: D. Arifler

Material based on
- *Computer Graphics with OpenGL®,* Fourth Edition by Donald Hearn, M. Pauline Baker, and Warren R. Carithers
- *Fundamentals of Computer Graphics*, Third Edition by by Peter Shirley and Steve Marschner
- *Computer Graphics* by F. S. Hill

# Window-to-viewport transformation

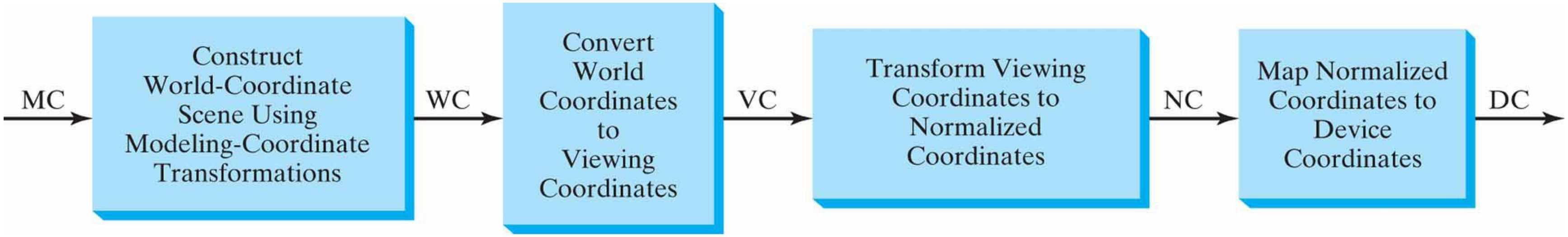
- Clipping window: section of 2D scene selected for display
- Viewport: window where the scene is to be displayed on the output device

**Figure 8-1** A clipping window and associated viewport, specified as rectangles aligned with the coordinate axes.



# Viewing pipeline

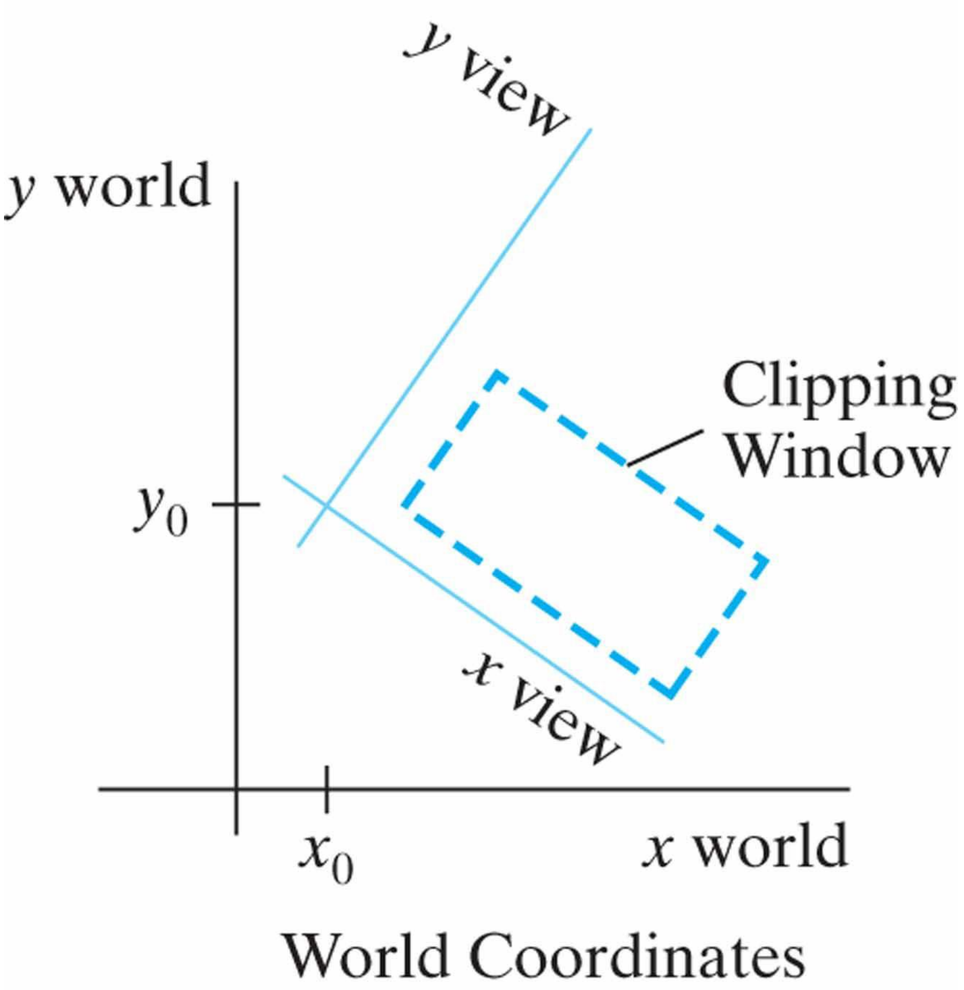
**Figure 8-2** Two-dimensional viewing-transformation pipeline.


MC
Construct World-Coordinate Scene Using Modeling-Coordinate Transformations
WC
Convert World Coordinates to Viewing Coordinates
VC
Transform Viewing Coordinates to Normalized Coordinates
NC
Map Normalized Coordinates to Device Coordinates
DC

Normalization makes viewing device independent
Clipping can be applied to object descriptions in normalized coordinates

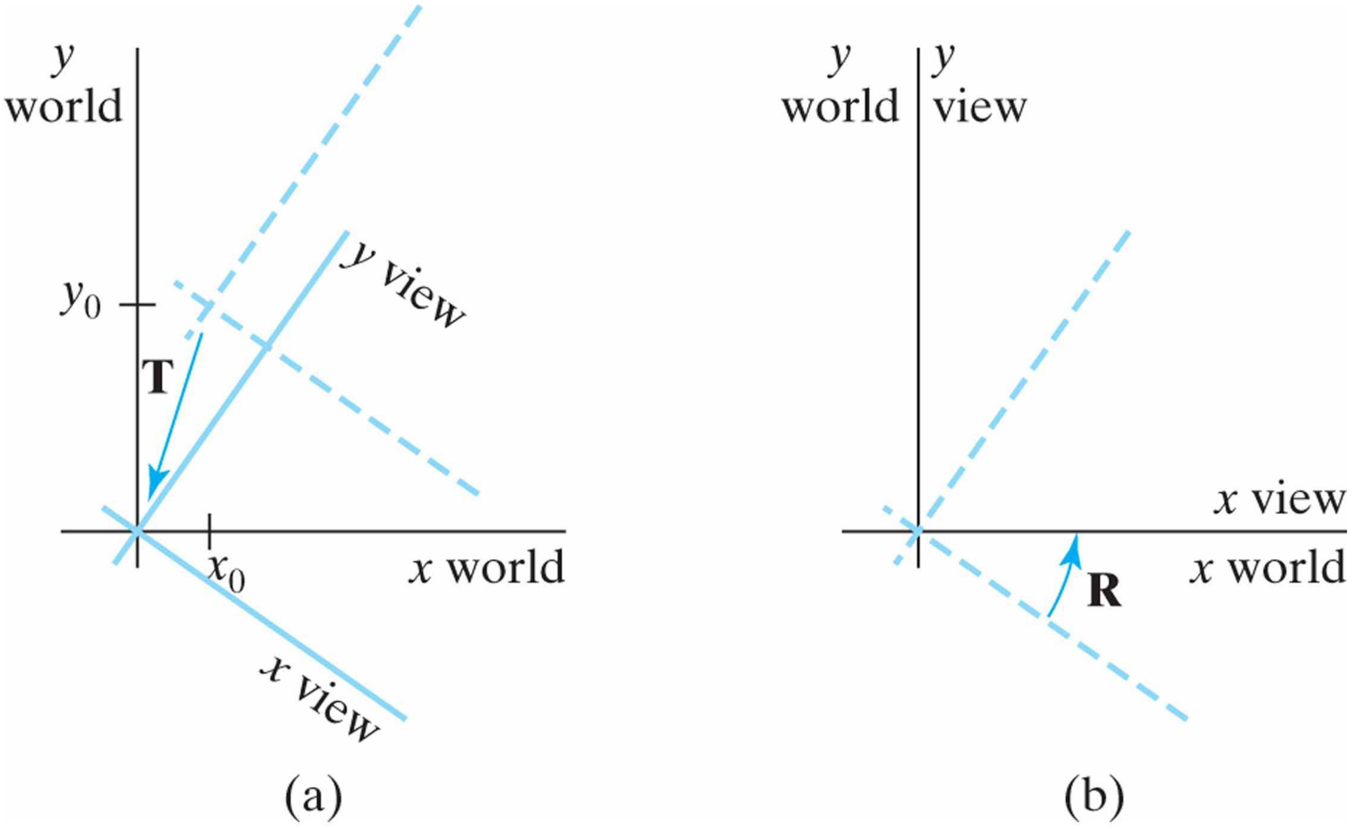
# Viewing coordinates

**Figure 8-3** A rotated clipping window defined in viewing coordinates.



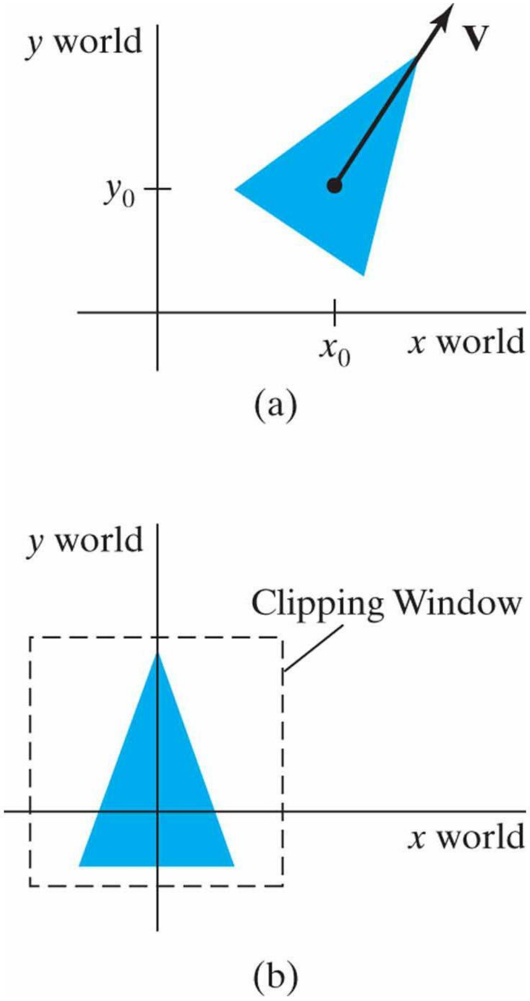
# Viewing coordinates

**Figure 8-4** A viewing-coordinate frame is moved into coincidence with the world frame by (a) applying a translation matrix **T** to move the viewing origin to the world origin, then (b) applying a rotation matrix **R** to align the axes of the two systems.



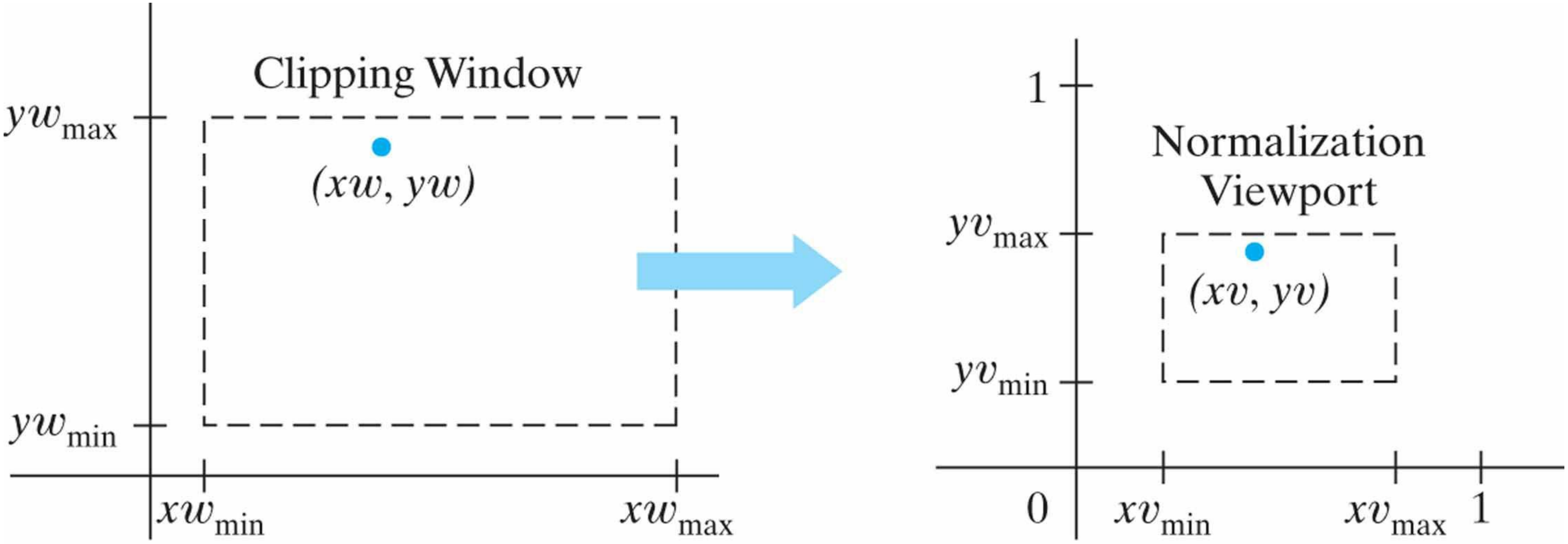
# View up vector

**Figure 8-5** A triangle (a), with a selected reference point and orientation vector, is translated and rotated to position (b) within a clipping window.



# Mapping the clipping window into normalized viewport

**Figure 8-6** 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.



# Window-to-viewport mapping

To transform the world-coordinate point into the same relative position within the viewport, we require that

$$\frac{xv - xv_{\min}}{xv_{\max} - xv_{\min}} = \frac{xw - xw_{\min}}{xw_{\max} - xw_{\min}}$$
$$\frac{yv - yv_{\min}}{yv_{\max} - yv_{\min}} = \frac{yw - yw_{\min}}{yw_{\max} - yw_{\min}} \tag{2}$$

Solving these expressions for the viewport position $(xv, yv)$, we have

$$xv = s_x xw + t_x$$
$$yv = s_y yw + t_y \tag{3}$$

# Window-to-viewport mapping

where the scaling factors are

$$s_x = \frac{xv_{\text{max}} - xv_{\text{min}}}{xw_{\text{max}} - xw_{\text{min}}} \qquad \textbf{(4)}$$
$$s_y = \frac{yv_{\text{max}} - yv_{\text{min}}}{yw_{\text{max}} - yw_{\text{min}}}$$

and the translation factors are

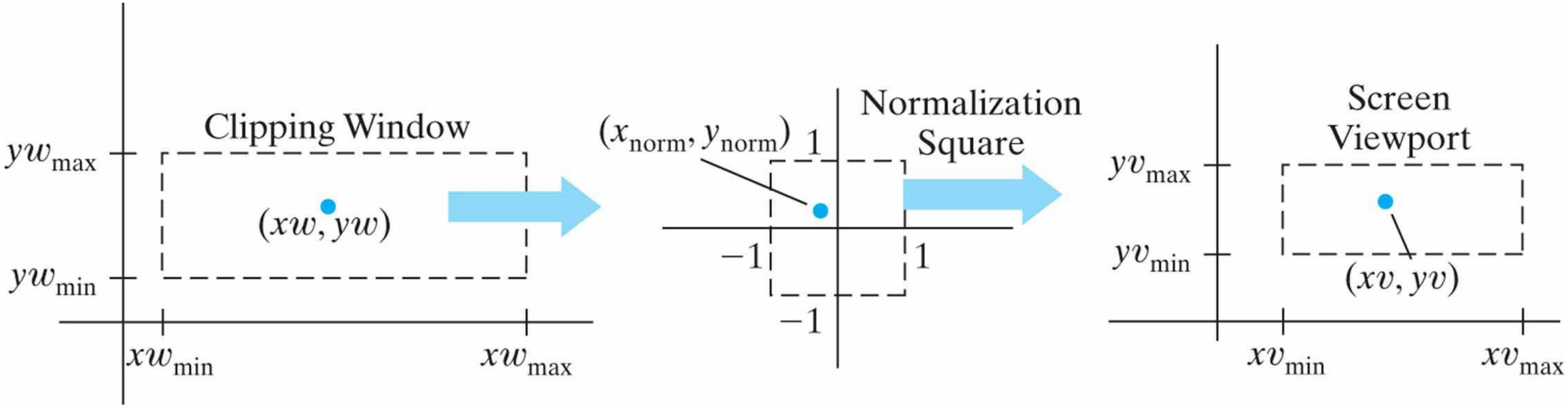
$$t_x = \frac{xw_{\text{max}}xv_{\text{min}} - xw_{\text{min}}xv_{\text{max}}}{xw_{\text{max}} - xw_{\text{min}}} \qquad \textbf{(5)}$$
$$t_y = \frac{yw_{\text{max}}yv_{\text{min}} - yw_{\text{min}}yv_{\text{max}}}{yw_{\text{max}} - yw_{\text{min}}}$$

$$\mathbf{M}_{\text{window, normviewp}} = \mathbf{T} \cdot \mathbf{S} = \begin{bmatrix} s_x & 0 & t_x \\ 0 & s_y & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

# Alternative: mapping clipping window into a normalized square

- Advantage: clipping algorithms are standardized (see more later)
- Substitute $xv_{min}=yv_{min}=-1$ and $xv_{max}=yv_{max}=1$

**Figure 8-7** A point (*xw, yw*) in a clipping window is mapped to a normalized coordinate position ($x_{norm}$, $y_{norm}$), then to a screen-coordinate position (*xv, yv*) in a viewport. Objects are clipped against the normalization square before the transformation to viewport coordinates occurs.



# Mapping to a normalized square

Making these substitutions in the expressions for $t_x$, $t_y$, $s_x$, and $s_y$, we have

$$\mathbf{M}_{\text{window, normsquare}} = \begin{bmatrix} \dfrac{2}{xw_{\text{max}} - xw_{\text{min}}} & 0 & -\dfrac{xw_{\text{max}} + xw_{\text{min}}}{xw_{\text{max}} - xw_{\text{min}}} \\ 0 & \dfrac{2}{yw_{\text{max}} - yw_{\text{min}}} & -\dfrac{yw_{\text{max}} + yw_{\text{min}}}{yw_{\text{max}} - yw_{\text{min}}} \\ 0 & 0 & 1 \end{bmatrix} \quad (9)$$

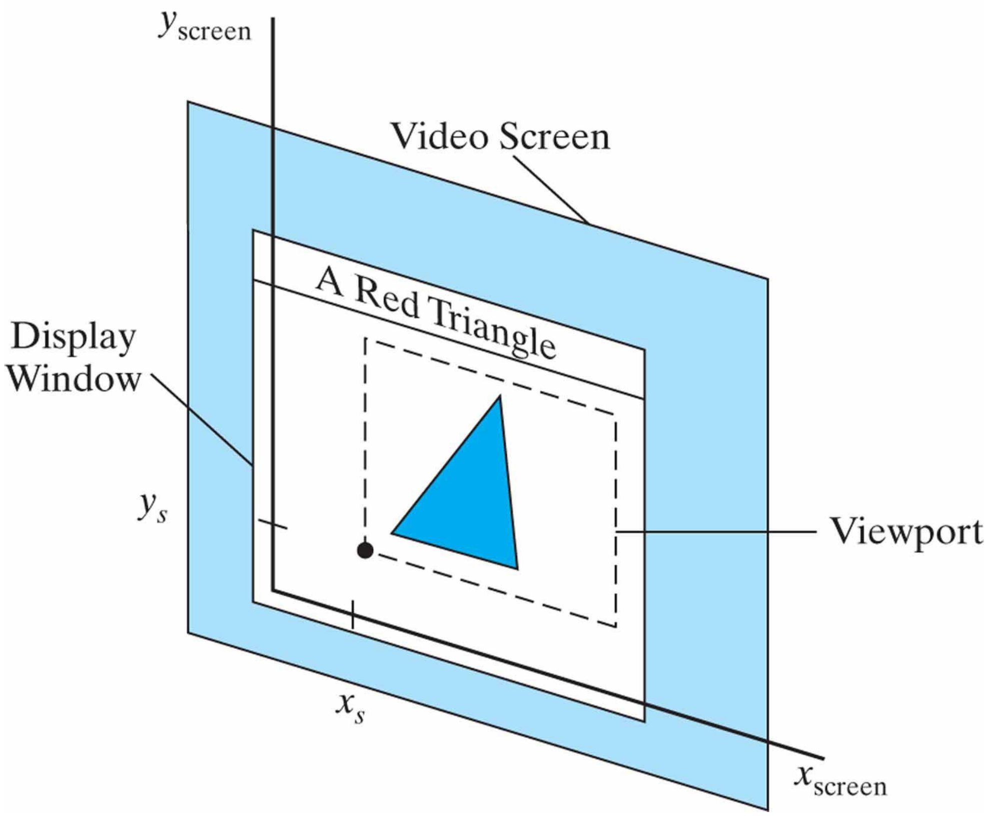
# Finally, mapping to viewport

Similarly, after the clipping algorithms have been applied, the normalized square with edge length equal to 2 is transformed into a specified viewport. This time, we get the transformation matrix from Equation 8 by substituting −1 for $xw_{\text{min}}$ and $yw_{\text{min}}$ and substituting +1 for $xw_{\text{max}}$ and $yw_{\text{max}}$:

$$\mathbf{M}_{\text{normsquare, viewport}} = \begin{bmatrix} \dfrac{xv_{\text{max}} - xv_{\text{min}}}{2} & 0 & \dfrac{xv_{\text{max}} + xv_{\text{min}}}{2} \\ 0 & \dfrac{yv_{\text{max}} - yv_{\text{min}}}{2} & \dfrac{yv_{\text{max}} + yv_{\text{min}}}{2} \\ 0 & 0 & 1 \end{bmatrix} \tag{10}$$

# Screen, display window, viewport

**Figure 8-8** A viewport at coordinate position ($x_s$ , $y_s$ ) within a display window.



# OpenGL 2D viewing functions

```
glMatrixMode (GL_PROJECTION);

glLoadIdentity ( );
```

- GLU clipping-window function

```
gluOrtho2D (xwmin, xwmax, ywmin, ywmax);
```

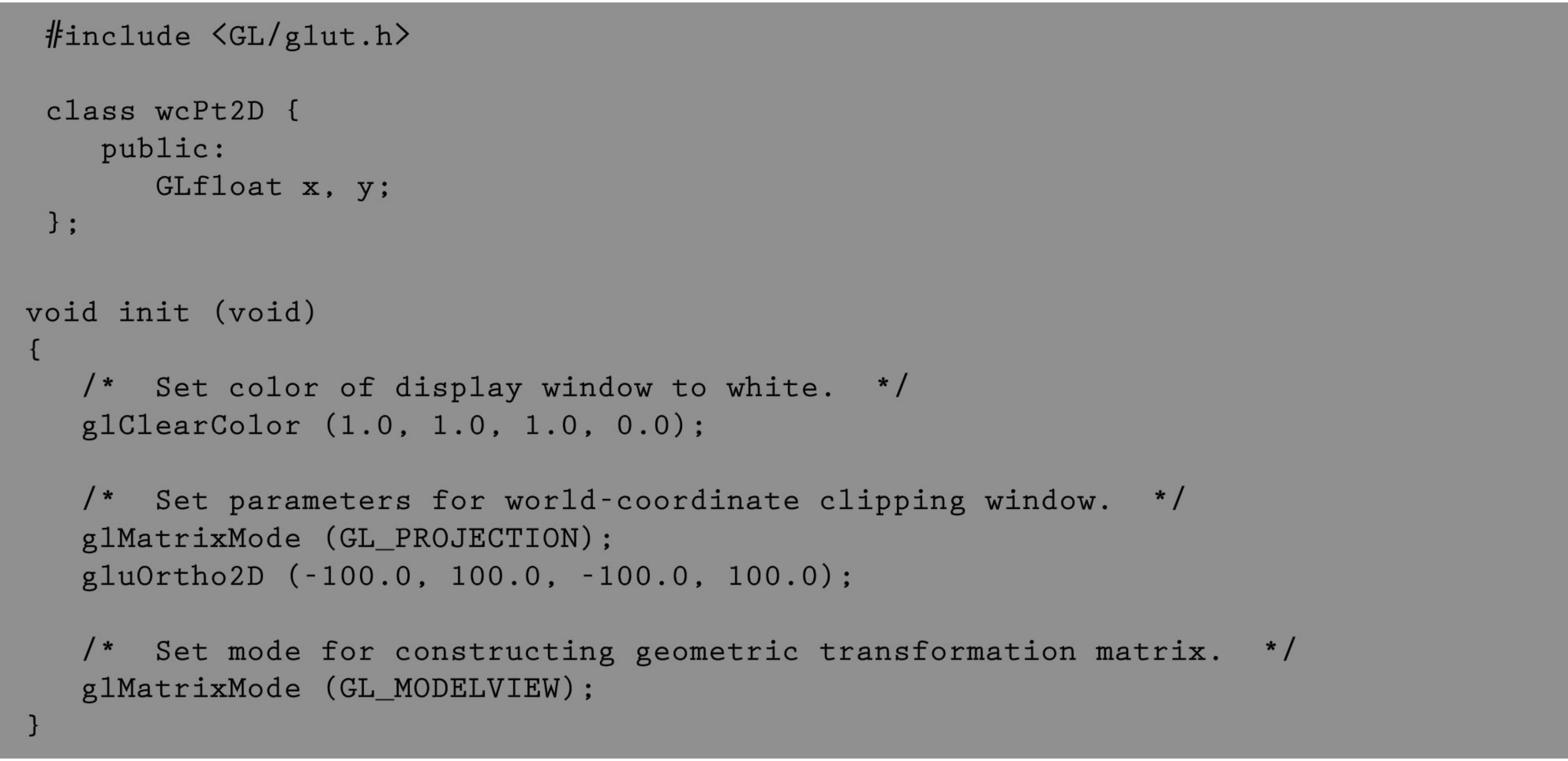
- OpenGL viewport function

```
glViewport (xvmin, yvmin, vpWidth, vpHeight);
```

# Creating a GLUT display window

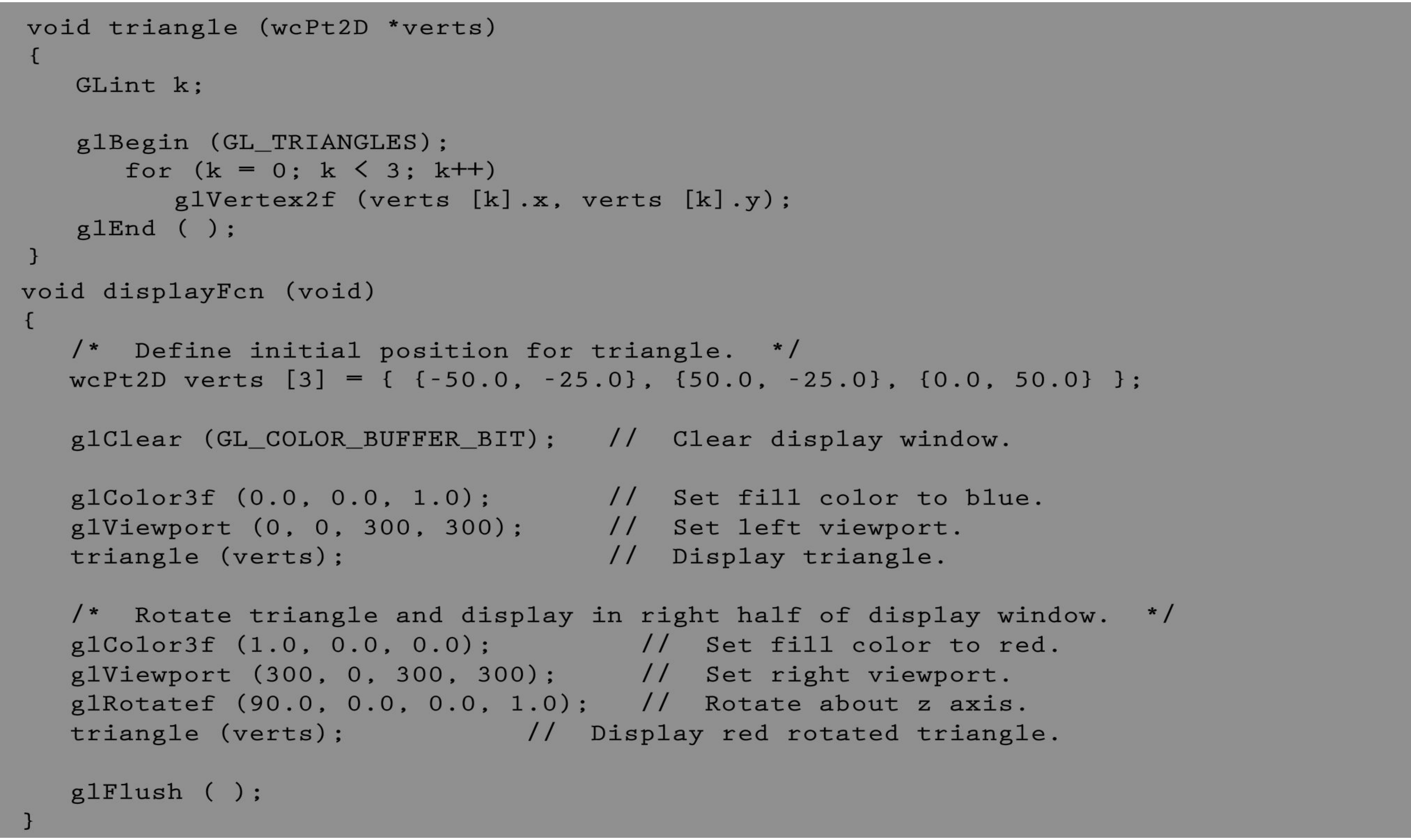
```
glutInitWindowPosition (xTopLeft, yTopLeft);
glutInitWindowSize (dwWidth, dwHeight);
glutCreateWindow ("Title of Display Window");
```

# Example



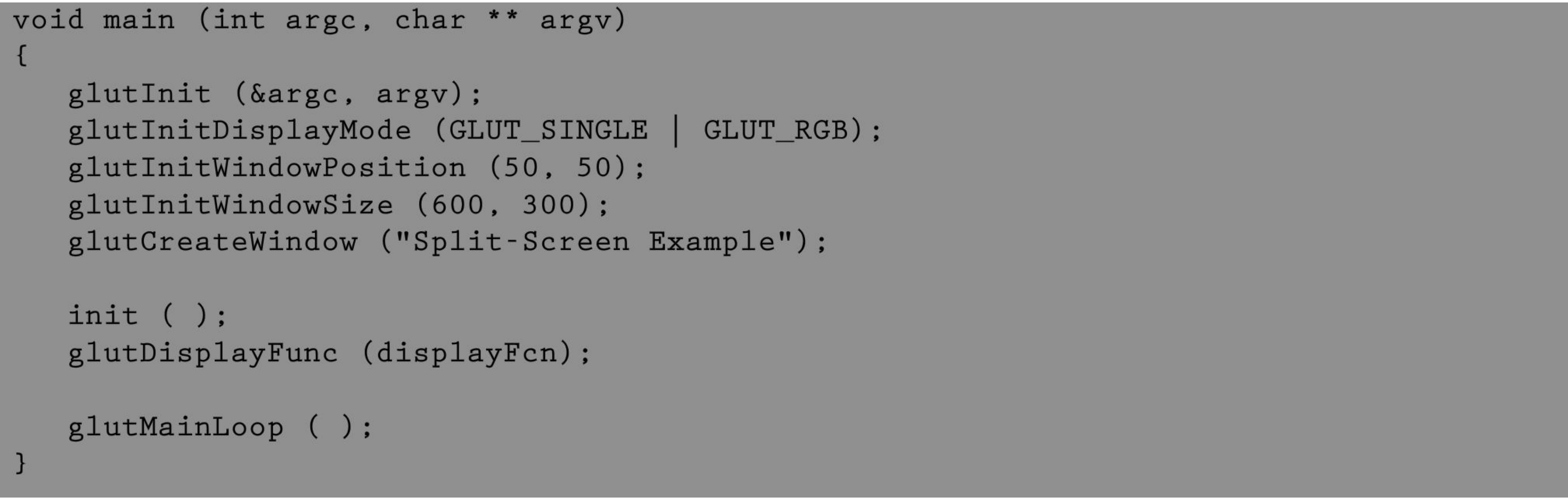
```
 #include <GL/glut.h>

 class wcPt2D {
    public:
       GLfloat x, y;
 };


void init (void)
{
   /*  Set color of display window to white.  */
   glClearColor (1.0, 1.0, 1.0, 0.0);

   /*  Set parameters for world-coordinate clipping window.  */
   glMatrixMode (GL_PROJECTION);
   gluOrtho2D (-100.0, 100.0, -100.0, 100.0);

   /*  Set mode for constructing geometric transformation matrix.  */
   glMatrixMode (GL_MODELVIEW);
}
```

# Example



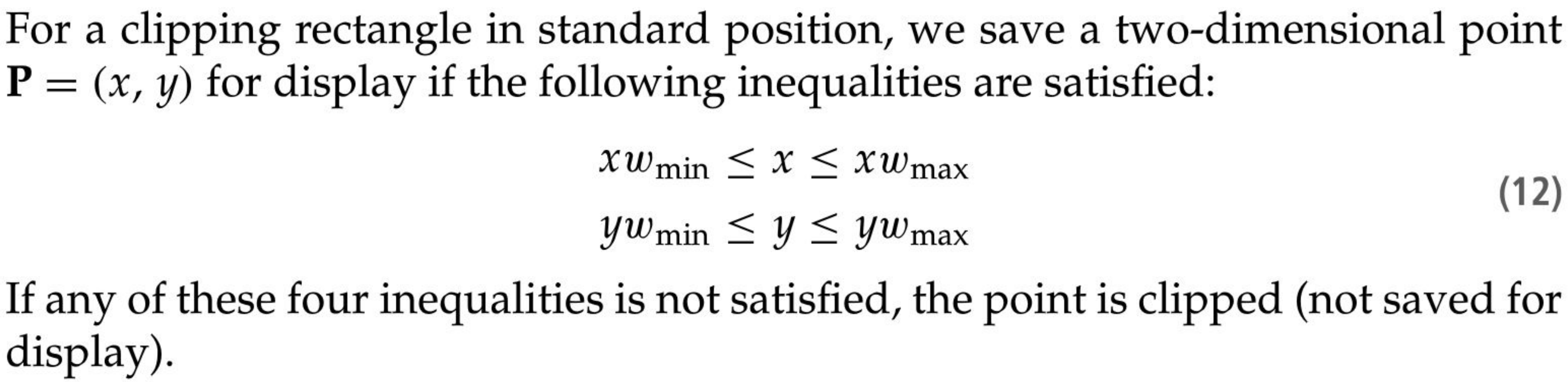
```
void triangle (wcPt2D *verts)
{
    GLint k;

    glBegin (GL_TRIANGLES);
        for (k = 0; k < 3; k++)
            glVertex2f (verts [k].x, verts [k].y);
    glEnd ( );
}
void displayFcn (void)
{
    /*  Define initial position for triangle.  */
    wcPt2D verts [3] = { {-50.0, -25.0}, {50.0, -25.0}, {0.0, 50.0} };

    glClear (GL_COLOR_BUFFER_BIT);   //  Clear display window.

    glColor3f (0.0, 0.0, 1.0);       //  Set fill color to blue.
    glViewport (0, 0, 300, 300);     //  Set left viewport.
    triangle (verts);                //  Display triangle.

    /*  Rotate triangle and display in right half of display window.  */
    glColor3f (1.0, 0.0, 0.0);         //  Set fill color to red.
    glViewport (300, 0, 300, 300);     //  Set right viewport.
    glRotatef (90.0, 0.0, 0.0, 1.0);   //  Rotate about z axis.
    triangle (verts);           //  Display red rotated triangle.

    glFlush ( );
}
```

# Example



```
void main (int argc, char ** argv)
{
   glutInit (&argc, argv);
   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
   glutInitWindowPosition (50, 50);
   glutInitWindowSize (600, 300);
   glutCreateWindow ("Split-Screen Example");

   init ( );
   glutDisplayFunc (displayFcn);

   glutMainLoop ( );
}
```

# 2D point clipping



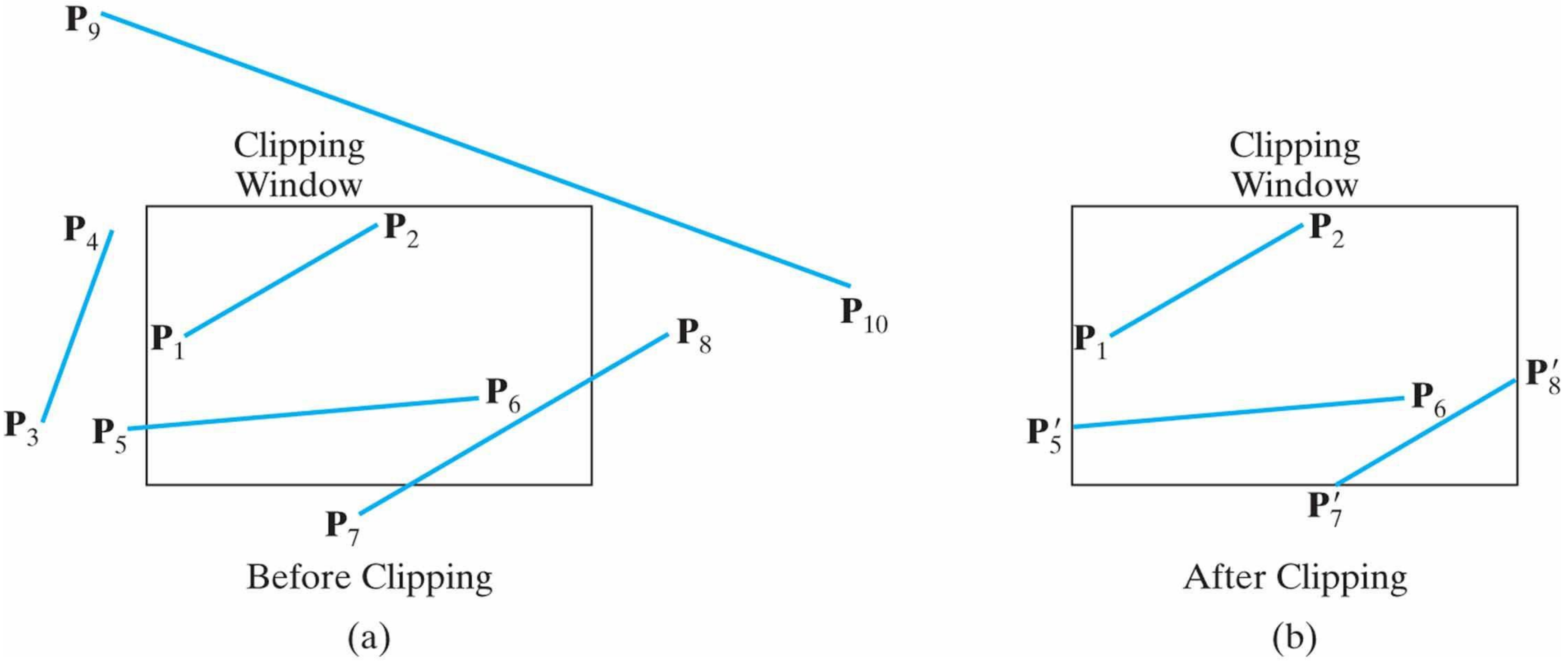
For a clipping rectangle in standard position, we save a two-dimensional point $\mathbf{P} = (x, y)$ for display if the following inequalities are satisfied:

$$\begin{aligned} xw_{\min} \leq x \leq xw_{\max} \\ yw_{\min} \leq y \leq yw_{\max} \end{aligned} \tag{12}$$

If any of these four inequalities is not satisfied, the point is clipped (not saved for display).

# 2D line clipping

**Figure 8-9** Clipping straight-line segments using a standard rectangular clipping window.



# 2D line clipping: basic approach

- Test if line is completely inside or outside
- When both endpoints are inside all four clipping boundaries, the line is completely inside the window
- Testing of outside is more difficult: When both endpoints are outside any one of four boundaries, line is completely outside
- If both tests fail, line segment intersects at least one clipping boundary and it may or may not cross into the interior of the clipping window

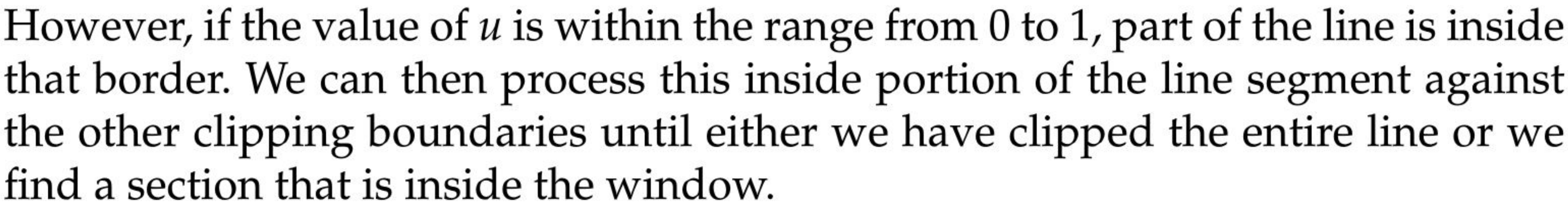
# Finding intersections and parametric equations

One way to formulate the equation for a straight-line segment is to use the following parametric representation, where the coordinate positions $(x_0, y_0)$ and $(x_{\text{end}}, y_{\text{end}})$ designate the two line endpoints:

$$\begin{aligned} x &= x_0 + u(x_{\text{end}} - x_0) \\ y &= y_0 + u(y_{\text{end}} - y_0) \qquad 0 \le u \le 1 \end{aligned} \tag{13}$$

We can use this parametric representation to determine where a line segment crosses each clipping-window edge by assigning the coordinate value for that edge to either $x$ or $y$ and solving for parameter $u$. For example, the left window boundary is at position $xw_{\text{min}}$, so we substitute this value for $x$, solve for $u$, and calculate the corresponding $y$-intersection value. If this value of $u$ is outside the range from 0 to 1, the line segment does not intersect that window border line.

# Parametric equations and clipping



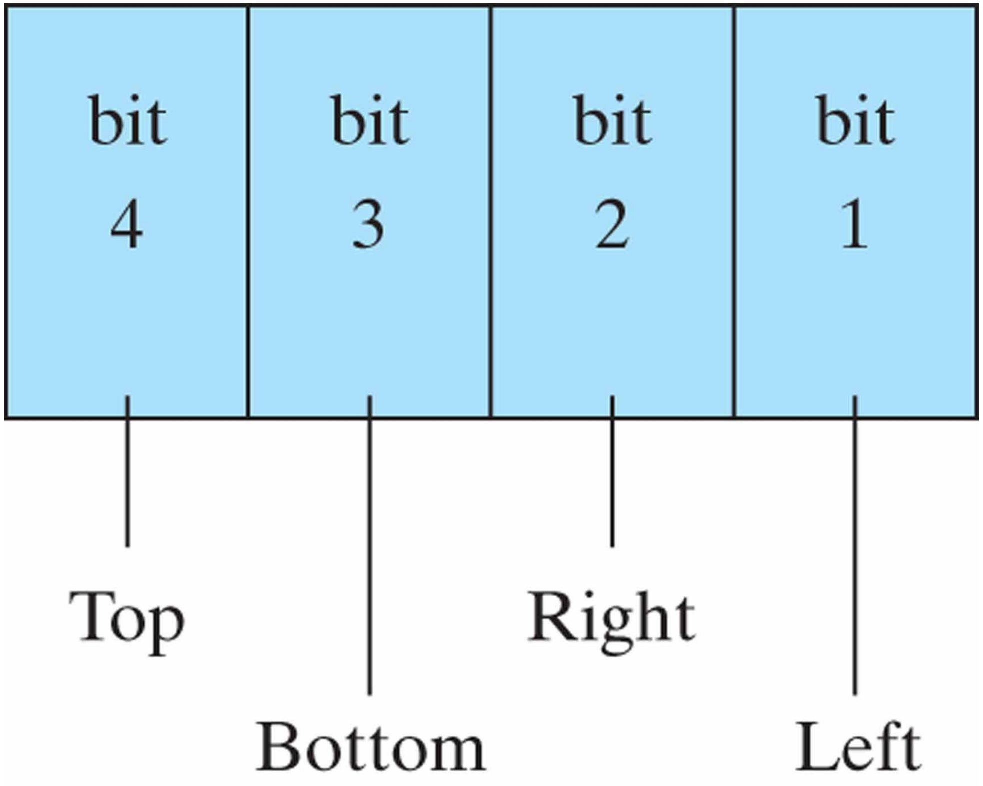
However, if the value of $u$ is within the range from 0 to 1, part of the line is inside that border. We can then process this inside portion of the line segment against the other clipping boundaries until either we have clipped the entire line or we find a section that is inside the window.

# Cohen-Sutherland line clipping

- Perform more tests before finding intersections
- Every line endpoint is assigned a 4-digit binary value (region code or out code), and each bit position is used to indicate whether the point is inside or outside one of the clipping-window boundaries
- E.g., suppose that the coordinate of the endpoint is (x, y). Bit 1 is set to 1 if $x<xw_{min}$

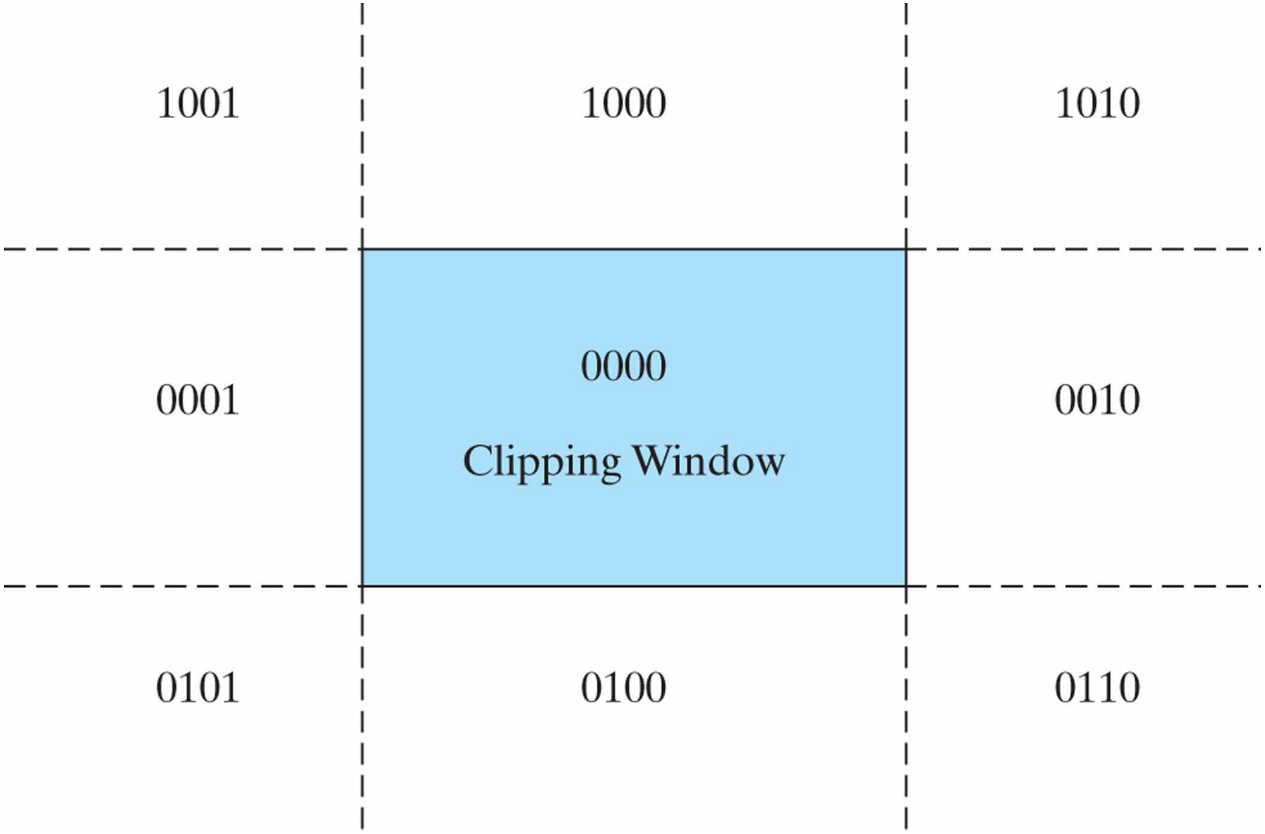
# Region codes

**Figure 8-10** A possible ordering for the clipping window boundaries corresponding to the bit positions in the Cohen-Sutherland endpoint region code.



# Region codes

**Figure 8-11** The nine binary region codes for identifying the position of a line endpoint, relative to the clipping-window boundaries.



# Cohen-Sutherland line clipping: steps

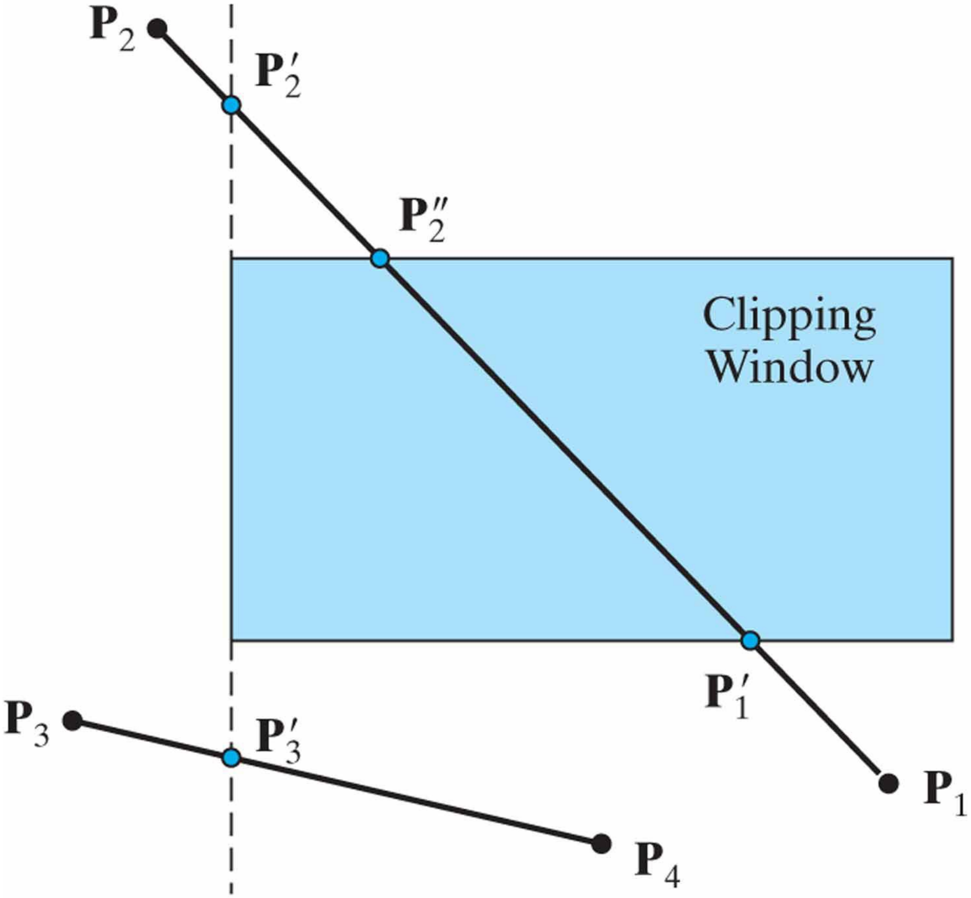
- Calculate differences between endpoint coordinates and clipping boundaries
- Use the resultant sign bit of each difference to set the corresponding value in the region code
  - Bit 1 is the sign bit of $x\text{-}xw_{min}$
  - Bit 2 is the sign bit of $xw_{max}\text{-}x$
  - Bit 3 is the sign bit of $y\text{-}yw_{min}$
  - Bit 4 is the sign bit of $yw_{max}\text{-}y$
- Any lines that are completely inside have a region code 0000 for both endpoints (save the line segment)
- Any line that has a region code value of 1 in the same bit position for each endpoint is completely outside (eliminate the line segment)

# Cohen-Sutherland line clipping: inside-outside tests

- For performance improvement, first do inside-outside tests
- When the OR operation between two endpoint region codes for a line segment is FALSE (0000), the line is inside the clipping region
- When the AND operation between two endpoint region codes for a line is TRUE (not 0000), then line is completely outside the clipping window
- Lines that cannot be identified as being completely inside or completely outside are next checked for intersection with the window border lines

# CS clipping: completely inside-outside?

**Figure 8-12** Lines extending from one clipping-window region to another may cross into the clipping window, or they could intersect one or more clipping boundaries without entering the window.



# CS clipping

- To determine whether the line crosses a selected clipping boundary, we check the corresponding bit values in the two endpoint region codes
  - If one of these bit values is 1 and the other is 0, the line segment crosses that boundary
- To determine a boundary intersection for a line segment, we use the slope-intercept form of the line equation
- For a line with endpoint coordinates (x0, y0) and (xEnd, yEnd), the y coordinate of the intersection point with a vertical clipping border line can be obtained with the calculation

$$y=y0+m(x-x0)$$

# CS clipping

where x value is set to either $xw_{min}$ or $xw_{max}$, and the slope $m=(yEnd-y0)/(xEnd-x0)$

- Similarly, if we are looking for the intersection with a horizontal border, $x=x0+(y-y0)/m$ with y value set to $yw_{min}$ or $yw_{max}$

# Liang-Barsky line clipping

For a line segment with endpoints $(x_0, y_0)$ and $(x_{\text{end}}, y_{\text{end}})$, we can describe the line with the parametric form

$$\begin{aligned} x &= x_0 + u\Delta x \\ y &= y_0 + u\Delta y \qquad 0 \leq u \leq 1 \end{aligned} \tag{16}$$

where $\Delta x = x_{\text{end}} - x_0$ and $\Delta y = y_{\text{end}} - y_0$. In the Liang-Barsky algorithm, the parametric line equations are combined with the point-clipping conditions 12 to obtain the inequalities

$$\begin{aligned} xw_{\min} \leq x_0 + u\Delta x \leq xw_{\max} \\ yw_{\min} \leq y_0 + u\Delta y \leq yw_{\max} \end{aligned} \tag{17}$$

which can be expressed as

$$u\ p_k \leq q_k, \qquad k = 1, 2, 3, 4 \tag{18}$$

# Liang-Barsky line clipping

where parameters $p$ and $q$ are defined as

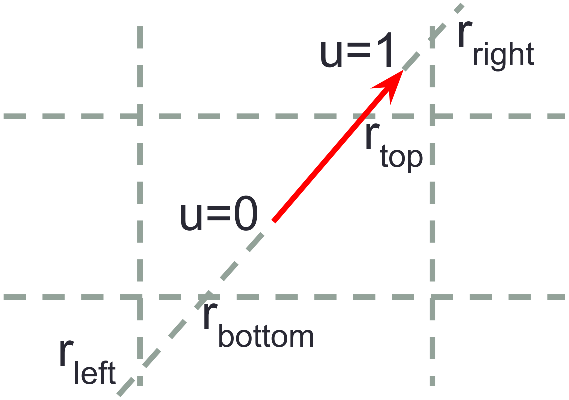
$$\begin{array}{lll} p_1 = -\Delta x, & q_1 = x_0 - xw_{\min} & \text{(left)} \\ p_2 = \Delta x, & q_2 = xw_{\max} - x_0 & \text{(right)} \\ p_3 = -\Delta y, & q_3 = y_0 - yw_{\min} & \text{(bottom)} \\ p_4 = \Delta y, & q_4 = yw_{\max} - y_0 & \text{(top)} \end{array} \qquad (19)$$

# Liang-Barsky line clipping

- If $p_k=0$ (line parallel to clipping window edge)
  - If $q_k<0$, the line is completely outside the boundary (clip)
  - If $q_k\geq 0$, the line is completely inside the parallel clipping border (needs further processing)
- When $p_k<0$, infinite extension of line proceeds from outside to inside of the infinite extension of this particular clipping window edge
- When $p_k>0$, line proceeds from inside to outside
- For non-zero $p_k$, we can calculate the value of u that corresponds to the point where the infinitely extended line intersects the extension of the window edge k as $u=q_k/p_k$

# LB algorithm

- If $p_k=0$ and $q_k<0$ for any k, clip the line and stop. Otherwise, go to next step
- For all k such that $p_k<0$ (outside-inside), calculate $r_k=q_k/p_k$. Let u1 be the max of $\{0, r_k\}$
- For all k such that $p_k>0$ (inside-outside), calculate $r_k=q_k/p_k$. Let u2 be the min of $\{r_k, 1\}$
- If u1>u2, clip the line since it is completely outside. Otherwise, use u1 and u2 to calculate the endpoints of the clipped line
- Example: (u1<u2)
- $u1=\max\{0, r_{left}, r_{bottom}\}$
- $u2=\min\{r_{top}, r_{right}, 1\}$



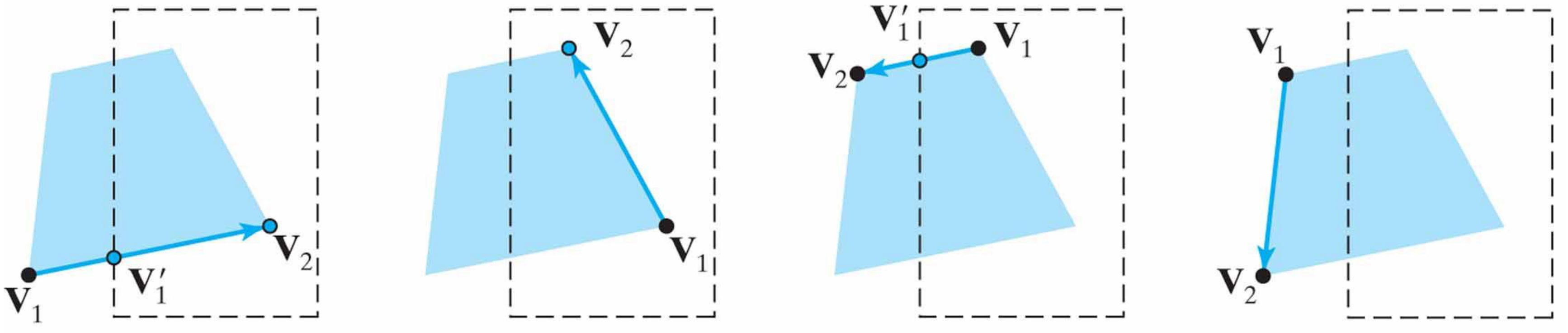
# Notes

- LB is more efficient than CS
- Both CS and LB can be extended to 3D

# Polygon Fill-Area Clipping

- Sutherland-Hodgman polygon clipping

**Figure 8-24** The four possible outputs generated by the left clipper, depending on the position of a pair of endpoints relative to the left boundary of the clipping window.



(1) out ⟶ in Output: $\mathbf{V}'_1, \mathbf{V}_2$

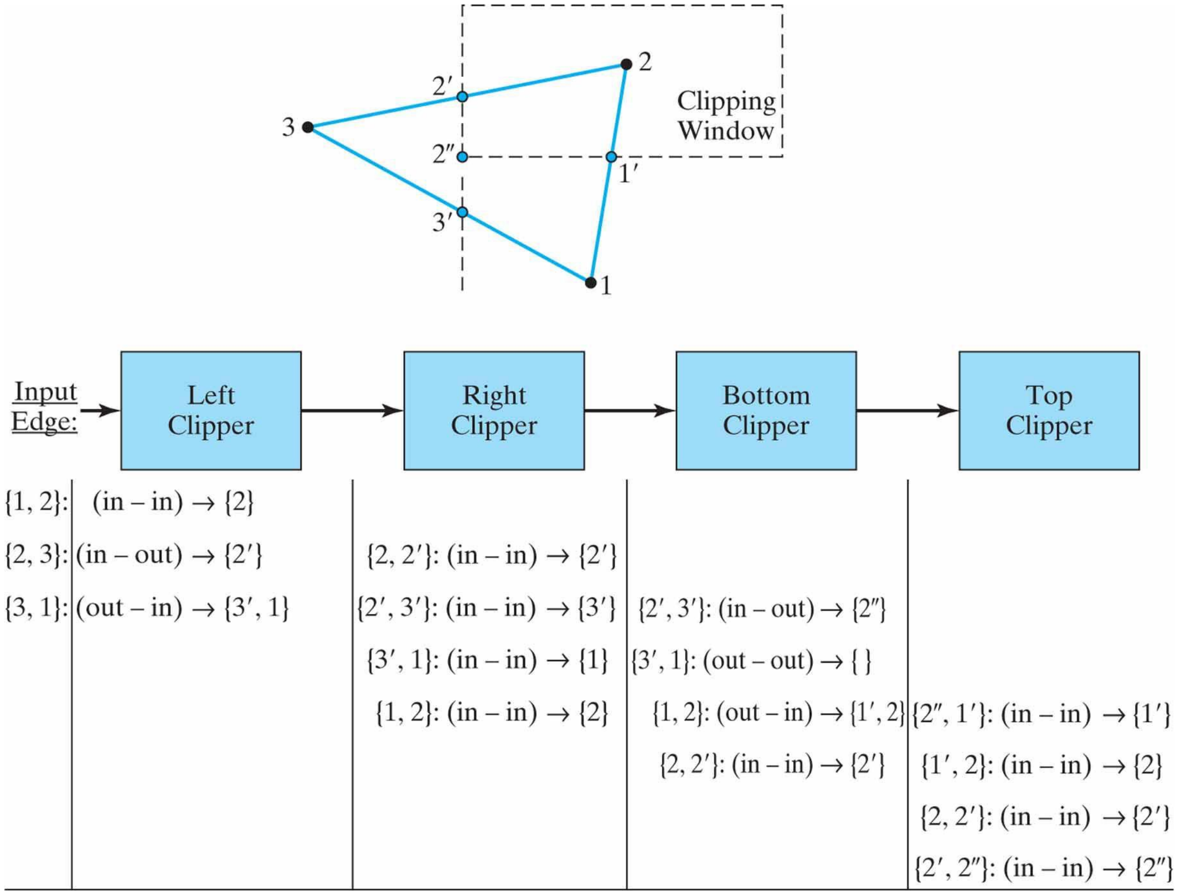
(2) in ⟶ in Output: $\mathbf{V}_2$

(3) in ⟶ out Output: $\mathbf{V}'_1$

(4) out ⟶ out Output: none

# Sutherland-Hodgman polygon clipping

**Figure 8-25** Processing a set of polygon vertices, {1, 2, 3}, through the boundary clippers using the Sutherland-Hodgman algorithm. The final set of clipped vertices is {1′, 2, 2′, 2″}.



| Input Edge: | Left Clipper | Right Clipper | Bottom Clipper | Top Clipper |
|---|---|---|---|---|
| {1, 2}: | (in – in) → {2} | | | |
| {2, 3}: | (in – out) → {2′} | {2, 2′}: (in – in) → {2′} | | |
| {3, 1}: | (out – in) → {3′, 1} | {2′, 3′}: (in – in) → {3′} | {2′, 3′}: (in – out) → {2″} | |
| | | {3′, 1}: (in – in) → {1} | {3′, 1}: (out – out) → { } | |
| | | {1, 2}: (in – in) → {2} | {1, 2}: (out – in) → {1′, 2} | {2″, 1′}: (in – in) → {1′} |
| | | | {2, 2′}: (in – in) → {2′} | {1′, 2}: (in – in) → {2} |
| | | | | {2, 2′}: (in – in) → {2′} |
| | | | | {2′, 2″}: (in – in) → {2″} |

# Sutherland-Hodgman polygon clipping

- Send pair of endpoints for each successive polygon line segment through the series of clippers. Four possible cases:

1. If the first input vertex is outside this clipping-window border and the second vertex is inside, both the intersection point of the polygon edge with the window border and the second vertex are sent to the next clipper
2. If both input vertices are inside this clipping-window border, only the second vertex is sent to the next clipper
3. If the first vertex is inside and the second vertex is outside, only the polygon edge intersection position with the clipping-window border is sent to the next clipper
4. If both input vertices are outside this clipping-window border, no vertices are sent to the next clipper

# Sutherland-Hodgman polygon clipping

- The last clipper in this series generates a vertex list that describes the final clipped fill area
- When a concave polygon is clipped, extraneous lines may be displayed. Solution is to split a concave polygon into two or more convex polygons

# Concave polygons

**Figure 8-26** Clipping the concave polygon in (a) using the Sutherland-Hodgman algorithm produces the two connected areas in (b).