Unit 4

Viewing and Clippning

Mrs. Deepali K. Jaadhav, KITCoEK

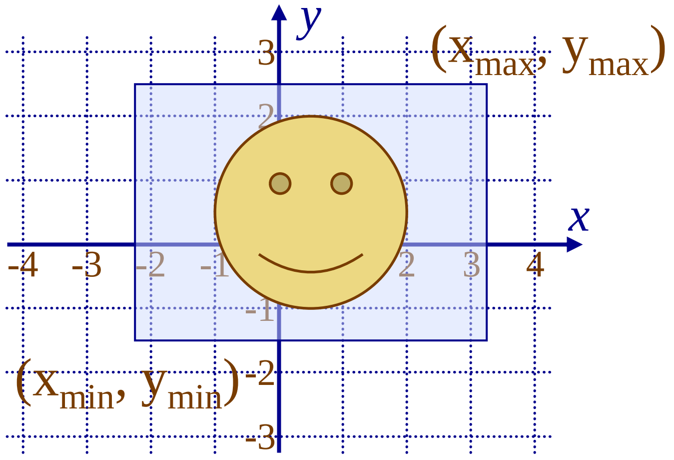
Viewing and Clippning

- Windowing and View-porting,
- Two Dimensional Clipping
- Simple Visibility Algorithm
- End Point Coding Algorithm
- Sutherland - Cohen line clipping algorithm

Window

- "Window" refers to the area in "world space" or "world coordinates" that you wish to project onto the screen
- Units could be inches, feet, meters, kilometers, light years, etc.
- The window is often centered around the origin, but need not be
- Specified as (x,y) coordinates

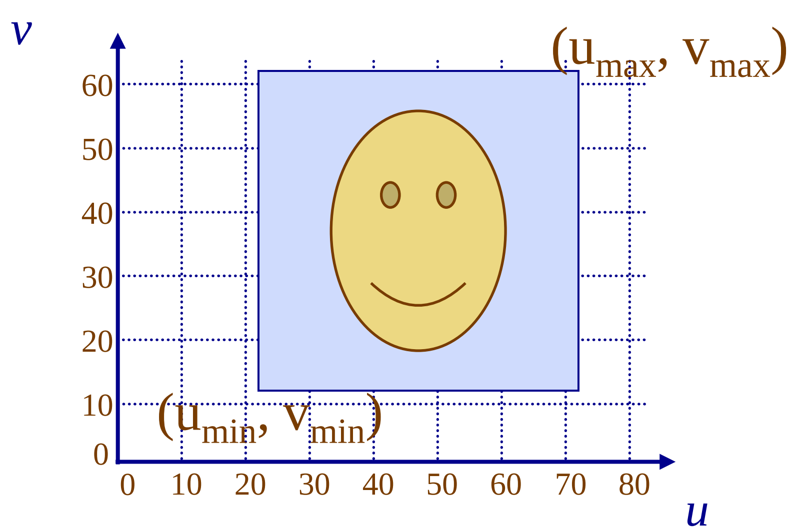
Window ("world")



Viewport

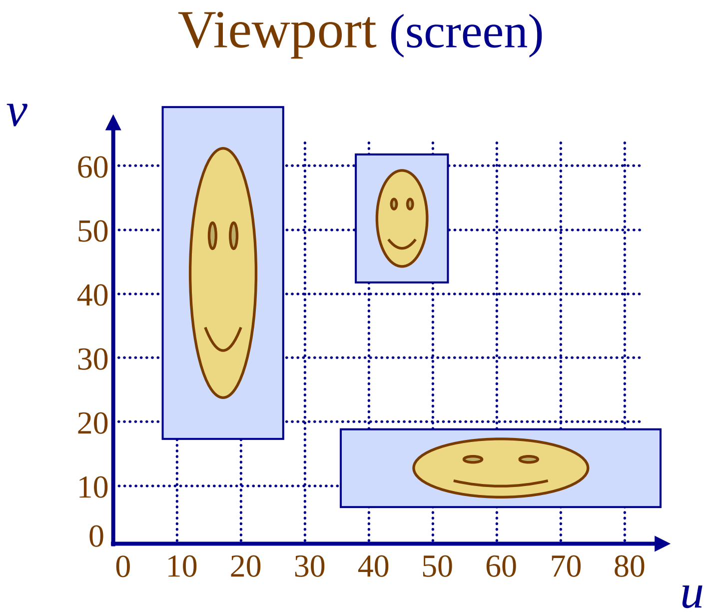
- The area on the screen that you will map the window.
- Specified in “screen coordinates” - (u,v) coordinates
- The viewport can take up the entire screen, or just a portion of it.

Viewport (screen)



Viewport (cont)

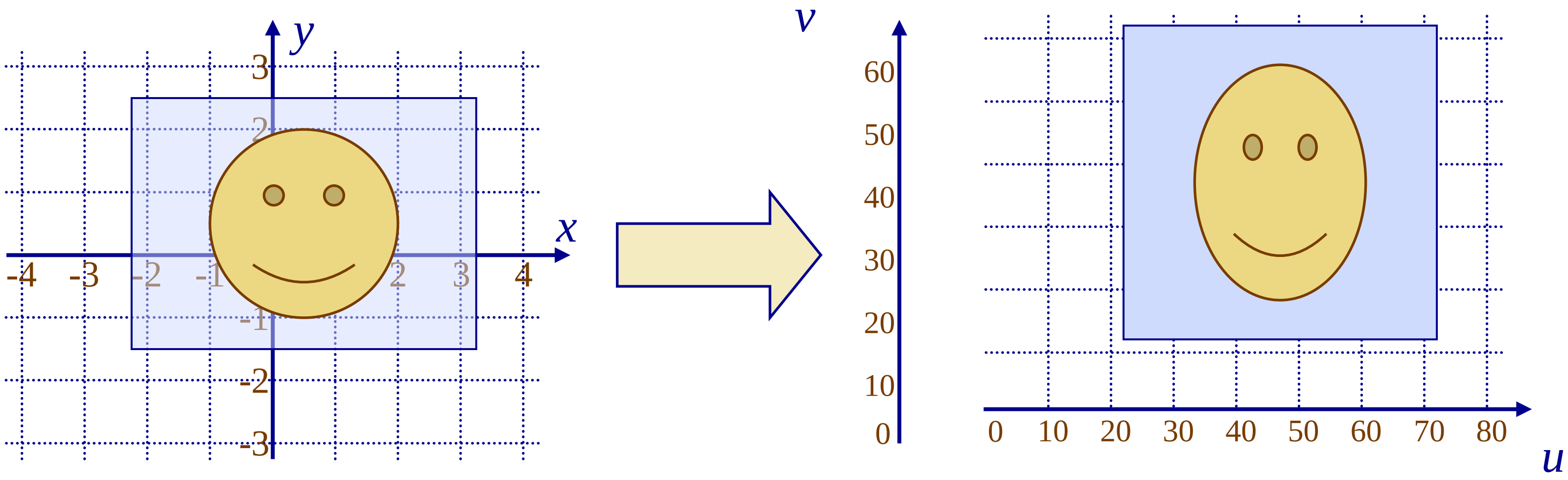
- You can have multiple viewports
  - They can contain the same view of a window, different views of the same window, or different views of different windows



Window-to-Viewport Transform

- Need to transform points from "world" view (*window*) to the screen view (*viewport*)
- Window-to-Viewport mapping is the process of mapping or transforming a two-dimensional, world coordinate scene to device coordinates. In particular, objects inside the world or clipping window are mapped to the viewport.
  - Can be done with a translate-scale-translate sequence

Window ("world") Viewport (screen)



Window-to-Viewport Transform (cont.)

Translate lower-left corner of window to origin

→

Scale width and height of window to match with the viewport

→

Translate corner at origin to lower-left corner in viewport

Window-to-Viewport Transform (cont.)

1. Translate lower-left corner of window to origin:
   - To shift window towards origin, lower left corner of window will become (-).
   - Hence translation factor will become negative (-tx,-ty).
   - $(-x_{min},-y_{min})$ – When origin is lower left corner of the screen.

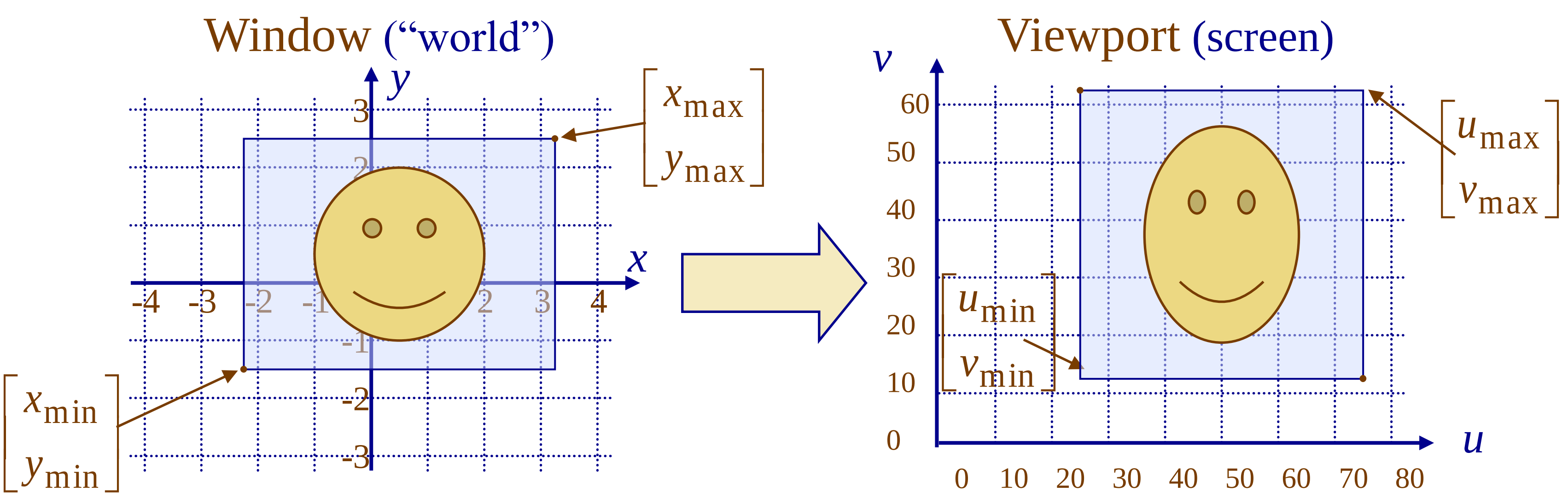
$$t_x = -x_{min} \text{ and } t_y = -y_{min}$$

Window-to-Viewport Transform (cont.)

2. Scale width and height of window to match with viewport.

- To convert window size in to view port size following computation is required.

$$s_x = \frac{u_{max} - u_{min}}{x_{max} - x_{min}} \text{ and } s_y = \frac{v_{max} - v_{min}}{y_{max} - y_{min}}$$



Window-to-Viewport Transform (cont.)

3. Translate corner at origin to lower-left corner in viewport

- If lower left corner of viewport is (0,0) we don't need to take step 3 because window lower left corner is already shifted on origin after taking first step.
- If lower left corner is not (0,0) we have to take translation factor (+).

$$t_x = u_{min} \text{ and } t_y = v_{min}$$

Window-to-Viewport Transform (cont.)

- The final window-to-viewport transform is:

$$M_{WV} = T(-x_{\min}, -y_{\min}) \bullet S\left(\frac{u_{\max} - u_{\min}}{x_{\max} - x_{\min}}, \frac{v_{\max} - v_{\min}}{y_{\max} - y_{\min}}\right) \bullet T(u_{\min}, v_{\min})$$

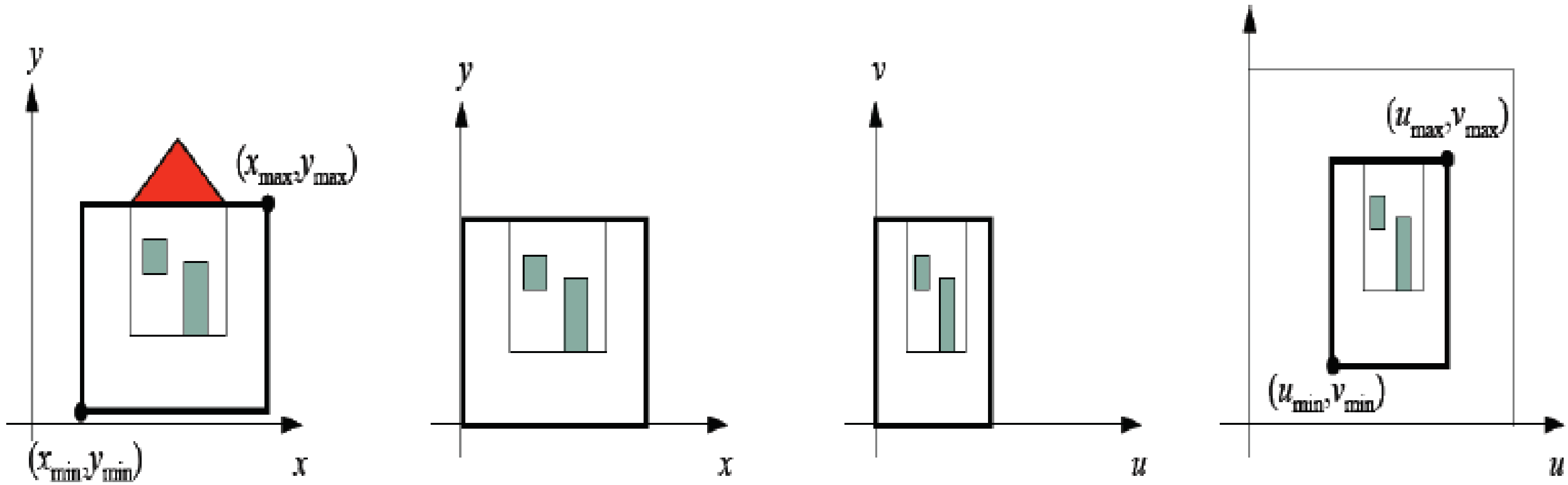
$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x_{\min} & -y_{\min} & 1 \end{bmatrix} \begin{bmatrix} \frac{u_{\max} - u_{\min}}{x_{\max} - x_{\min}} & 0 & 0 \\ 0 & \frac{v_{\max} - v_{\min}}{y_{\max} - y_{\min}} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ u_{\min} & v_{\min} & 1 \end{bmatrix}$$

Window-to-Viewport Transform (cont.)

- Multiplying the matrix M_{wv} by the point $p(x,y)$ gives:

$$p' = \begin{bmatrix} (x - x_{\min}) \bullet \dfrac{u_{\max} - u_{\min}}{x_{\max} - x_{\min}} + u_{\min} \\ (y - y_{\min}) \bullet \dfrac{v_{\max} - v_{\min}}{y_{\max} - y_{\min}} + v_{\min} \\ 1 \end{bmatrix}$$

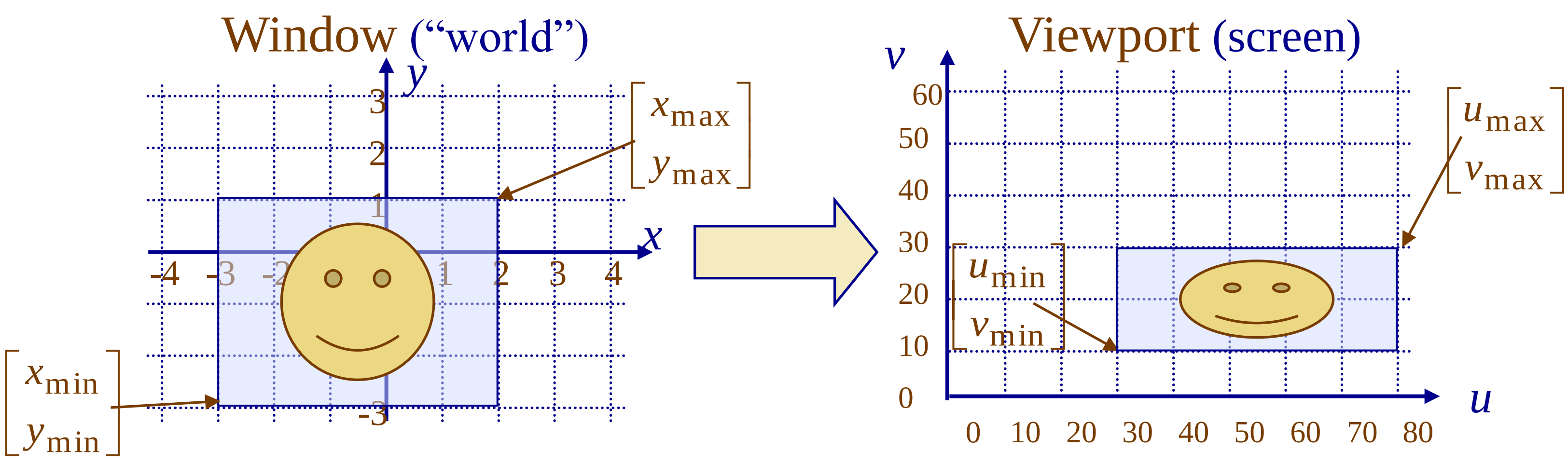
Window-to-Viewport Transform (cont.)



window in world coordinates	window translated to origin	window scaled to size of viewport	translated by (u_{min}, v_{min}) to final position

Window-to-Viewport Example

- Window: $(x_{min}, y_{min}) = (-3, -3)$, $(x_{max}, y_{max}) = (2, 1)$
- Viewport: $(u_{min}, v_{min}) = (30, 10)$, $(u_{max}, v_{max}) = (80, 30)$



Window-to-Viewport Example

- **Plugging the values into the equation:**
  - Window: $(x_{min}, y_{min}) = (-3, -3)$, $(x_{max}, y_{max}) = (2, 1)$
  - Viewport: $(u_{min}, v_{min}) = (30, 10)$, $(u_{max}, v_{max}) = (80, 30)$

$$p' = \begin{bmatrix} (x - x_{\min}) \bullet \dfrac{u_{\max} - u_{\min}}{x_{\max} - x_{\min}} + u_{\min} \\ (y - y_{\min}) \bullet \dfrac{v_{\max} - v_{\min}}{y_{\max} - y_{\min}} + v_{\min} \\ 1 \end{bmatrix} \quad p' = \begin{bmatrix} (x - (-3)) \bullet \dfrac{80 - 30}{2 - (-3)} + 30 \\ (y - (-3)) \bullet \dfrac{30 - 10}{1 - (-3)} + 10 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} (x + 3) \bullet \dfrac{50}{5} + 30 \\ (y + 3) \bullet \dfrac{20}{4} + 10 \\ 1 \end{bmatrix} = \begin{bmatrix} (x + 3) \bullet 10 + 30 \\ (y + 3) \bullet 5 + 10 \\ 1 \end{bmatrix} = \begin{bmatrix} 10x + 60 \\ 5y + 25 \\ 1 \end{bmatrix}$$

Window-to-Viewport Example

- So:

$$p' = \begin{bmatrix} 10x + 60 \\ 5y + 25 \\ 1 \end{bmatrix}$$

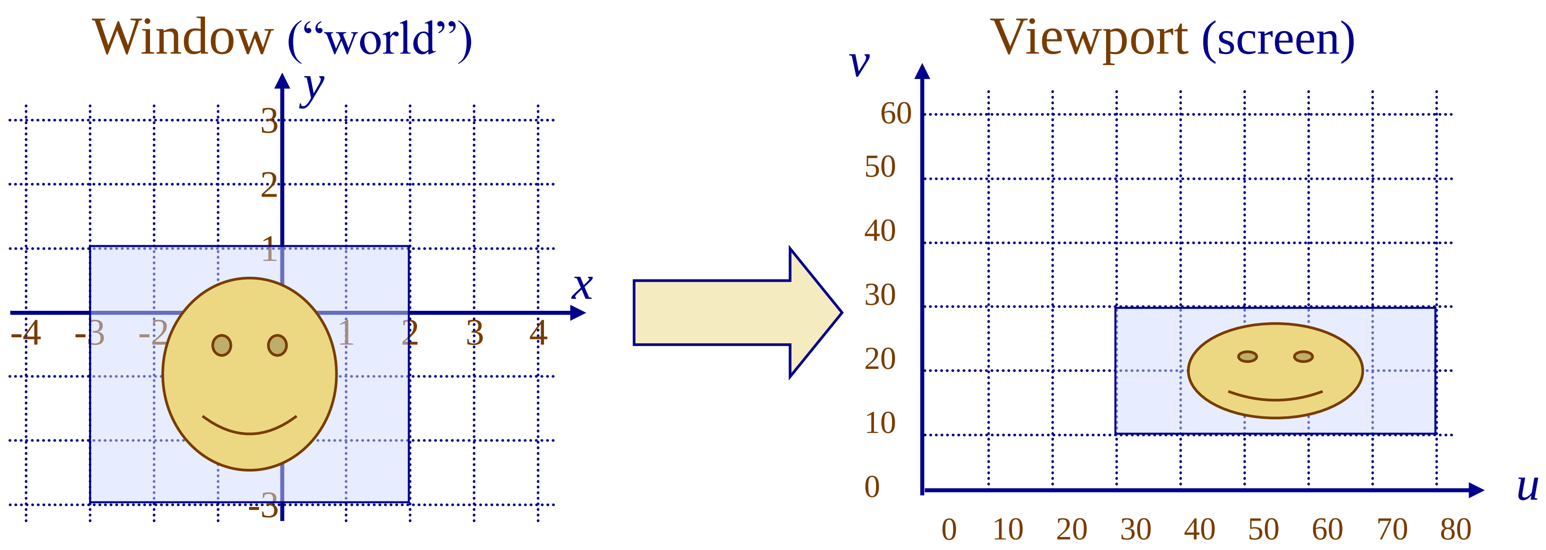
Trying some points:

$(x_{min}, y_{min}) = (-3, -3) \rightarrow (30, 10)$

$(x_{max}, y_{max}) = (2, 1) \rightarrow (80, 30)$

Left eye = (-1, -.8) –> (50, 21)

Top of head = (-0.5, 0.5) –> (55, 27.5)



Advantage of Viewing Transformation

- We can display picture at device or display system according to our need and choice.

Clipping

- When we have to display a large portion of the picture, then not only scaling & translation is necessary, the visible part of picture is also identified.
- This process is not easy. Certain parts of the image are inside, while others are partially inside.
- The lines or elements which are partially visible will be omitted.
- The process used for deciding the visible and invisible portion is called clipping.
- Clipping determines each element into the visible and invisible portion.
- Visible portion is selected. An invisible portion is discarded.
- Clipping is also useful for copying, moving or deleting a portion of a scene or picture e.g. 'cut' and 'paste' operation.

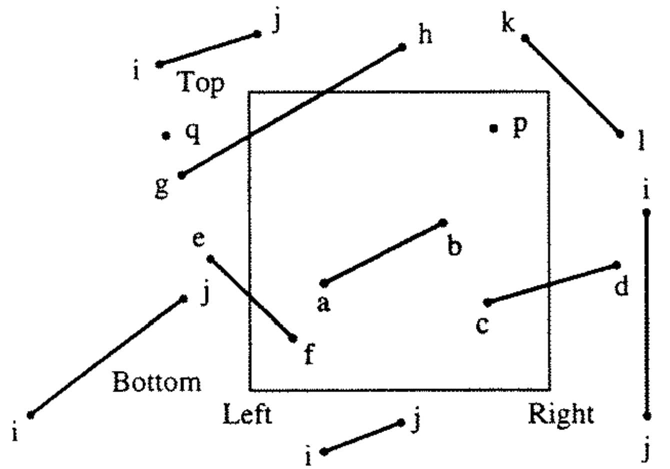
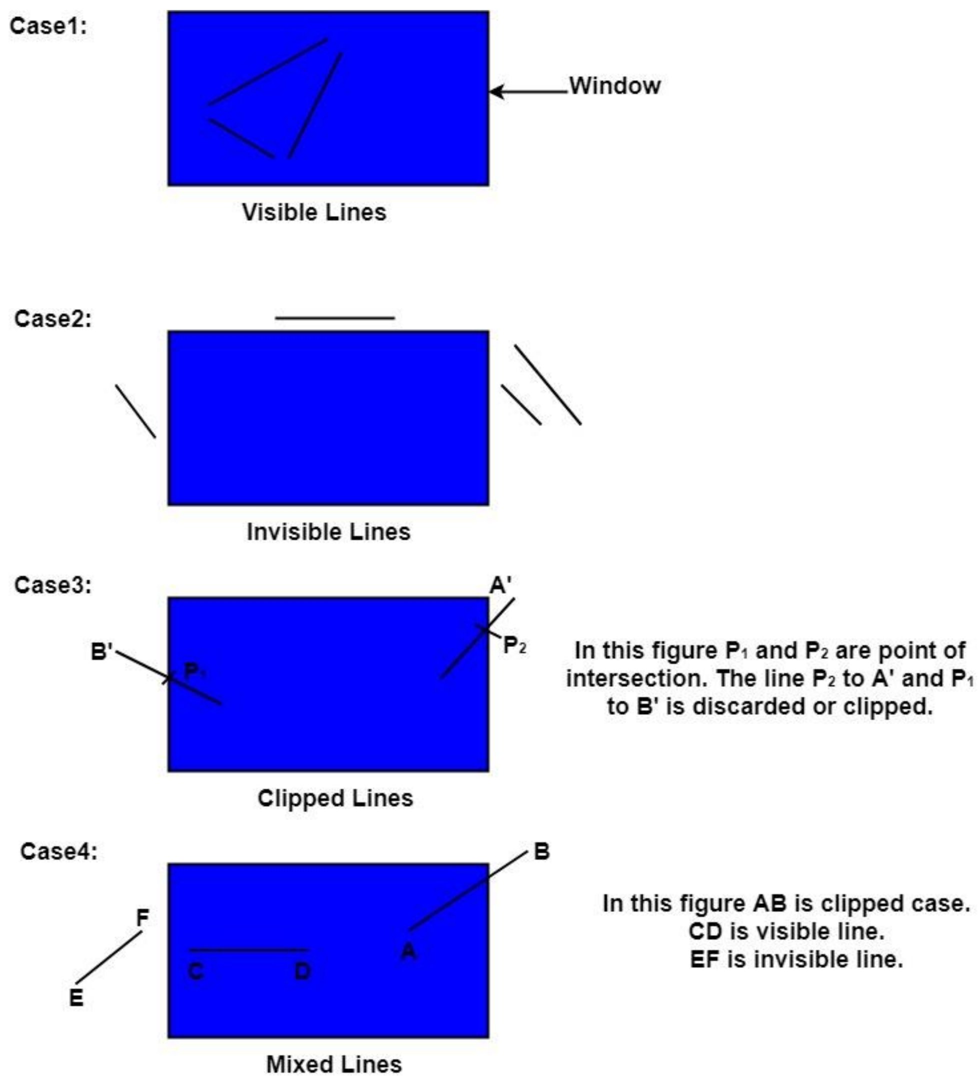


Figure 3–1 Two-dimensional clipping window.

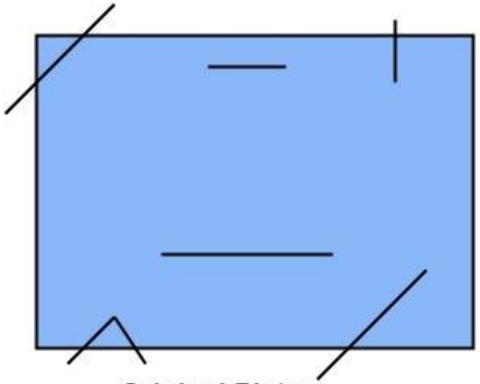
Two Dimensional Clipping

- Types of Lines: Lines are of three types:
  1. **Visible:** A line or lines entirely inside the window is considered visible
  2. **Invisible:** A line entirely outside the window is considered invisible
  3. **Clipped:** A line partially inside the window and partially outside is clipped. For clipping, point of intersection of a line with the window is determined.

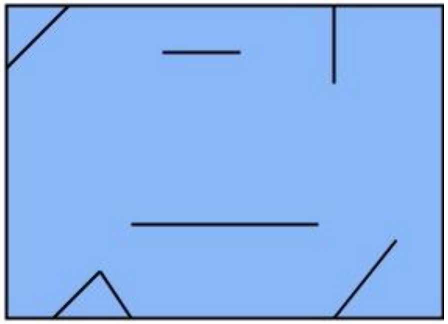


Two Dimensional Clipping

- Clipping can be applied through hardware as well as software.
- In some computers, hardware devices automatically do work of clipping.
- In a system where hardware clipping is not available software clipping applied.
- Following figure show before and after clipping



Original Picture
or
Before Clipping



After Clipping

- The window against which object is clipped called a clip window. It can be curved or rectangle in shape.

Line Clipping Algorithms

- Simple Visibility Algorithm
- End Point Coding Algorithm
- Sutherland - Cohen line clipping algorithm
- Midpoint Subdivision Algorithm

Simple Visibility Algorithm

- A point P(x, y) is interior to the clipping window if

$$x_L \le x \le x_R \qquad \text{and} \qquad y_B \le y \le y_T$$

- Consider a line with end points a and b.
- The visibility algorithm can be written as follows:

simple visibility algorithm

a *and* b *are the end points of the line, with components* x *and* y

for each line

Visibility = True

check for totally invisible lines

if both end points are left, right, above or below the window, the line is trivially invisible

if $x_a < x_L$ **and** $x_b < x_L$ **then** Visibility = False
if $x_a > x_R$ **and** $x_b > x_R$ **then** Visibility = False
if $y_a > y_T$ **and** $y_b > y_T$ **then** Visibility = False
if $y_a < y_B$ **and** $y_b < y_B$ **then** Visibility = False

Simple Visibility Algorithm

if Visibility <> False **then** *avoid the totally visible calculation*

check if the line is totally visible

if any coordinate of either end point is outside the window, then the line is not totally visible

if $x_a < x_L$ **or** $x_a > x_R$ **then** Visibility = Partial **Point a is outside**

if $x_b < x_L$ **or** $x_b > x_R$ **then** Visibility = Partial **Point b is outside**

if $y_a < y_B$ **or** $y_a > y_T$ **then** Visibility = Partial **Point a is outside**

if $y_b < y_B$ **or** $y_b > y_T$ **then** Visibility = Partial **Point b is outside**

end if

if Visibility = Partial **then**

the line is partially visible or diagonally crosses a corner invisibly

determine the intersections and the visibility of the line

end if

if Visibility = True **then**

line is totally visible — draw line

end if

line is invisible

next line

finish

Simple Visibility Algorithm

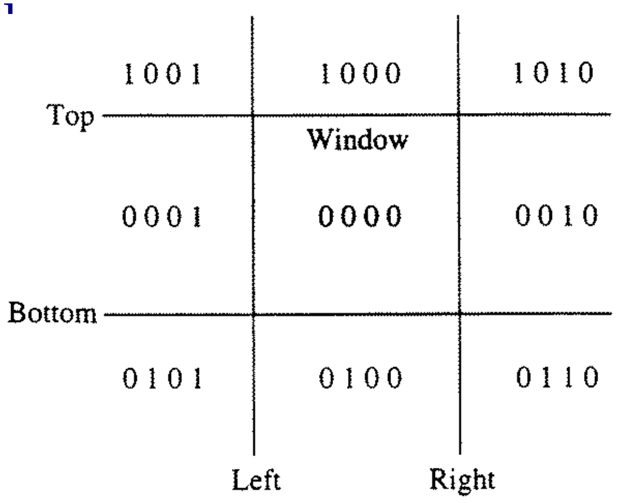
- Here, x_L, x_R, y_T and y_B are the x and y coordinates, respectively of the left, right, top and bottom of the window edges.
- The order in which test for visibility and invisibility are performed is immaterial.
- Some lines require all 4 tests, while some requires only one test.

End Point Coding Algorithm

- **End Point Codes:** The Visibility tests described by the simple visibility algorithm is improved further by coding the end points with 4 bit coding. This was given by **Dan Cohen** and **Ivan Sutherland**
- The scheme uses a 4 bit code (fig) to indicate which of the 9 regions contains the end point of a line. The rightmost bit is the first bit.
- The bit is set to 1, based on the following scheme.
  - **First bit** – If the end point is to the **left** of the window
  - **Second bit** – If the end point is to the **right** of the window
  - **Third bit** – If the end point is **below** the window
  - **Fourth bit** – If the end point is **above** the win

Otherwise, the bit is set to 0.

- If both end point codes are 0, then both ends of the line lie inside the window; the line is totally visible.



Codes for line end point regions.

End Point Coding – An Example

Truth Table for AND

1	0	0
0	1	0
0	0	0
1	1	1

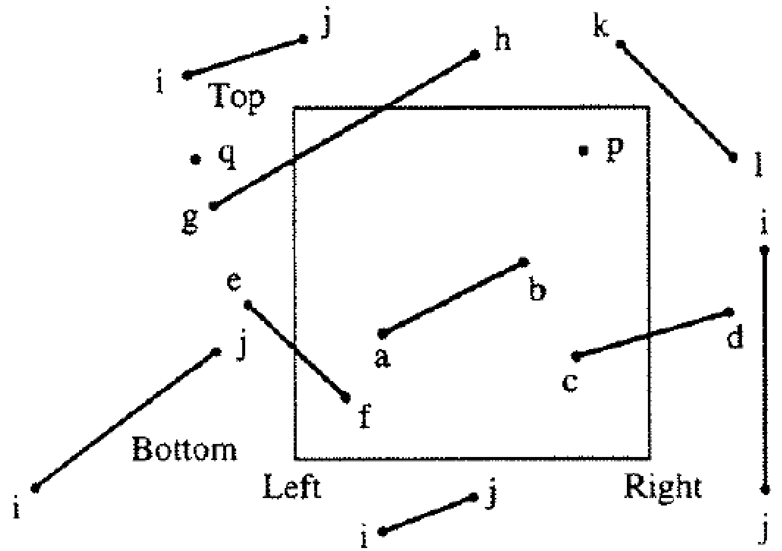


Table 3–1 End Point Codes

Line (see Fig. 3–1)	End point codes (see Fig. 3–2)		Logical and	Comments
ab	0000	0000	0000	Totally visible
ij	0010	0110	0010	Totally invisible
ij	1001	1000	1000	Totally invisible
ij	0101	0001	0001	Totally invisible
ij	0100	0100	0100	Totally invisible
cd	0000	0010	0000	Partially visible
ef	0001	0000	0000	Partially visible
gh	0001	1000	0000	Partially visible
kl	1000	0010	0000	Totally invisible

- If the bit-by-bit logical AND of the two end point codes is not zero, then the line is totally invisible and rejected.
- When the logical AND is zero, then line may be totally visible, partially visible or totally invisible.

End Point Coding Algorithm

- One possible software implementation that does not use bit manipulation routine is:
- P1 and P2 are the end points of the given line x_L, x_R, y_T, y_B are the left, right, top and bottom window coordinates Calculate the end point codes.

Store the endpoint codes in 1x4 arrays called P1code and P2code

first end point: P_1

if $x_1 < x_L$ **then** P1code(4) = 1 **else** P1code(4) = 0
if $x_1 > x_R$ **then** P1code(3) = 1 **else** P1code(3) = 0
if $y_1 < y_B$ **then** P1code(2) = 1 **else** P1code(2) = 0
if $y_1 > y_T$ **then** P1code(1) = 1 **else** P1code(1) = 0

second end point: P_2

if $x_2 < x_L$ **then** P2code(4) = 1 **else** P2code(4) = 0
if $x_2 > x_R$ **then** P2code(3) = 1 **else** P2code(3) = 0
if $y_2 < y_B$ **then** P2code(2) = 1 **else** P2code(2) = 0
if $y_2 > y_T$ **then** P2code(1) = 1 **else** P2code(1) = 0

finish

End Point Coding Algorithm

The intersection between two lines can be determined either parametrically or nonparametrically. Explicitly, the equation of the infinite line through $P_1(x_1, y_1)$ and $P_2(x_2, y_2)$ is

$$y = m(x - x_1) + y_1 \qquad \text{or} \qquad y = m(x - x_2) + y_2$$

where

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

is the slope of the line. The intersections with the window edges are given by

$$\begin{array}{lll}
\text{Left:} & x_L, y = m(x_L - x_1) + y_1 & m \neq \infty \\
\text{Right:} & x_R, y = m(x_R - x_1) + y_1 & m \neq \infty \\
\text{Top:} & y_T, x = x_1 + (1/m)(y_T - y_1) & m \neq 0 \\
\text{Bottom:} & y_B, x = x_1 + (1/m)(y_B - y_1) & m \neq 0
\end{array}$$

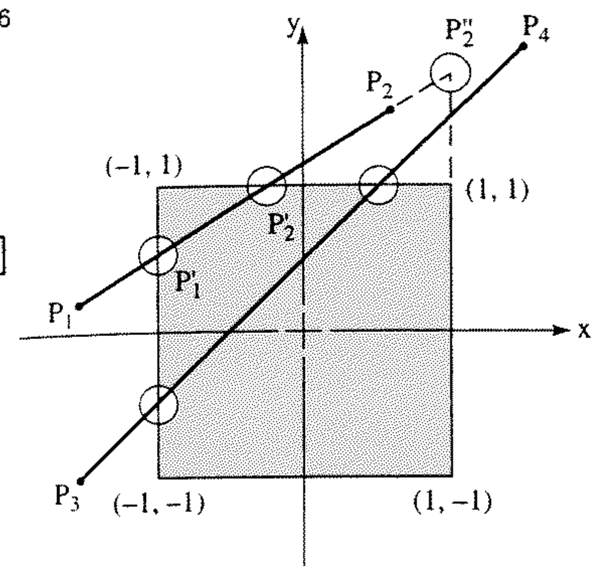
End Point Coding Algorithm - Example

- Consider the clipping window and the lines shown in Fig. 3–3. For the line from $P_1(3/2, 1/6)$ to $P_2(1/2, 3/2)$ the slope is

$$m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{3/2 - 1/6}{1/2 - (-3/2)} = \frac{2}{3}$$

and the intersections with the window edge are

$$\begin{array}{llll}
\text{Left:} & x = -1 & y = & 2/3\,[-1 - (-3/2)] + 1/6 \\
 & & = & 1/2 \\
\text{Right:} & x = 1 & y = & 2/3\,[1 - (-3/2)] + 1/6 \\
 & & = & 11/6 \\
\text{Top:} & y = 1 & x = & -3/2 + 3/2\,[1 - 1/6] \\
 & & = & -1/4 \\
\text{Bottom:} & y = -1 & x = & -3/2 + 3/2\,[-1 - 1/6] \\
 & & = & -13/4
\end{array}$$



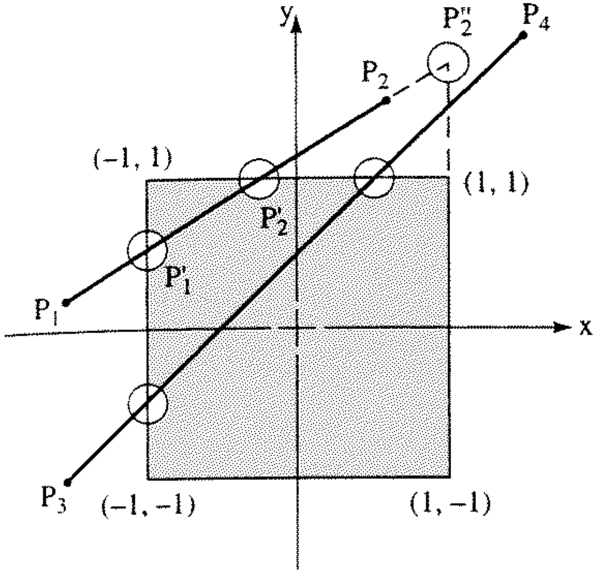
End Point Coding Algorithm - Example

Similarly, for the line from $P_3(-3/2, -1)$ to $P_4(3/2, 2)$

$$m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{2 - (-1)}{3/2 - (-3/2)} = 1$$

and

$$\begin{array}{llll}
\text{Left:} & x = -1 & y = & (1)[-1 - (-3/2)] + (-1) \\
 & & = & -1/2 \\
\text{Right:} & x = 1 & y = & (1)[1 - (-3/2)] + (-1) \\
 & & = & 3/2 \\
\text{Top:} & y = 1 & x = & -3/2 + (1)[1 - (-1)] \\
 & & = & 1/2 \\
\text{Bottom:} & y = -1 & x = & -3/2 + (1)[-1 - (-1)] \\
 & & = & -3/2
\end{array}$$



Sutherland - Cohen line clipping algorithm

- From previous discussion:
  - If the slope of the line is infinite, it is parallel to the left and right edges; and only the top and bottom edges need to be checked for intersections.
  - Similarly, if the slope is zero, the line is parallel to the top and bottom edges; and only the right and left edges need to be checked for intersections.
  - Finally, if either end point code is zero, one end point is interior the window and only one intersection can occur.

Sutherland - Cohen line clipping algorithm

For each window edge:

For the line P_1P_2, determine if the line is totally visible or can be trivially rejected as invisible.

If P_1 is outside the window, continue; otherwise, swap P_1 and P_2.

Replace P_1 with the intersection of P_1P_2 and the window edge.

Sutherland - Cohen line clipping algorithm - Example

Again consider the line P_1P_2 clipped against the window shown in Fig. 3–3. The end point codes for $P_1(-3/2, 1/6)$ and $P_2(1/2, 3/2)$ are (0001) and (1000), respectively. The end point codes are not simultaneously zero, and the logical **and** of the end point codes is zero. Consequently, the line is neither totally visible nor trivially invisible. Comparing the first bits of the end point codes, the line crosses the left edge; and P_1 is outside the window.

The intersection with the left edge ($x = -1$) of the window is $P_1'(-1, 1/2)$. Replace P_1 with P_1' to yield the new line, $P_1(-1, 1/2)$ to $P_2(1/2, 3/2)$.

The end point codes for P_1 and P_2 are now (0000) and (1000), respectively. The line is neither totally visible nor trivially invisible.

Comparing the second bits, the line does not cross the right edge; skip to the bottom edge.

The end point codes for P_1 and P_2 are still (0000) and (1000), respectively. The line is neither totally visible nor trivially invisible.

Comparing the third bits, the line does not cross the bottom edge. Skip to the top edge.

Sutherland - Cohen line clipping algorithm - Example

The end point codes for P_1 and P_2 are still (0000) and (1000), respectively. The line is neither totally visible nor trivially invisible.

Comparing the fourth bits, the line crosses the top edge. P_1 is not outside. Swap P_1 and P_2 to yield the new line, $P_1(1/2, 3/2)$ to $P_2(-1, 1/2)$.

The intersection with the top edge ($y = 1$) of the window is $P_1'(-1/4, 1)$. Replace P_1 with P_1' to yield the new line, $P_1(-1/4, 1)$ to $P_2(-1, 1/2)$.

The end point codes for P_1 and P_2 are (0000) and (0000), respectively. The line is totally visible.

The procedure is complete.

Draw the line ($P_1'P_2'$ in Fig. 3–3).

Midpoint Subdivision Algorithm

- The Sutherland Cohen subdivision line clipping algorithm requires the calculation of the intersection of the line with the window edge.
- These calculations can be avoided by repetitively subdividing the line at its midpoint.
- This algorithm is special case of Sutherland - Cohen line clipping algorithm was proposed by Sproull and Sutherland
- There should be following categories of the line-
  - **Visible line**
  - **Invisible line**
  - **Partially visible**
- We can calculate the midpoint of the line by the following formula- $\mathbf{p_m = (p_1 + p_2)/2}$

Midpoint Subdivision Algorithm

For each end point:

If the end point is visible, then it is the farthest visible point. The process is complete. If not, continue.

If the line is trivially invisible, no output is generated. The process is complete. If not, continue.

Guess at the farthest visible point by dividing the line P_1P_2 at its midpoint, P_m. Apply the previous tests to the two segments P_1P_m and P_mP_2. If P_mP_2 is rejected as trivially invisible, the midpoint is an overestimation of the farthest visible point. Continue with P_1P_m. Otherwise, the midpoint is an underestimation of the farthest visible point. Continue with P_2P_m. If the segment becomes so short that the midpoint corresponds to the accuracy of the machine or, as specified, to the end points, evaluate the visibility of the point and the process is complete.

Midpoint Subdivision Algorithm

- **Step1:** Calculate the position of both endpoints of the line
- **Step2:** Perform OR operation on both of these endpoints
- **Step3:** If the OR operation gives 0000 then

  Line is guaranteed to be visible

  else if

  Perform AND operation on both endpoints.

  If AND $\neq$ 0000 then

  the line is invisible

  else

  the line is clipped case – partially visible.

Midpoint Subdivision Algorithm

- **Step4:** For the line to be clipped. Find midpoint

  $X_m=(x_1+x_2)/2$

  $Y_m=(y_1+y_2)/2$

  X_m is midpoint of X coordinate.

  Y_m is midpoint of Y coordinate.
- **Step5:** Check each midpoint, whether it nearest to the boundary of a window or not.
- **Step6:** If the line is not totally visible or totally rejected then repeat step 1 to 5.
- **Step7:** Stop algorithm.

Midpoint Subdivision Algorithm - Example

- **Example:** A window contains the size (0, 50, 0, 50). A line PQ has the coordinates (-10, 40) and (30, -20). Find the visible point of the line using midpoint subdivision.
- **Solution:** We have,

  The coordinates for x and y = P (-10, 40)

  The coordinates for x and y = Q (30, -20)

  Now,
- **Step 1:** We have to compute the midpoint of the line segment PQ.

  Q' = [(-10 + 30) /2, (40 - 20)/2]

  = (10, 10)

  Now the new coordinates of Q' = (10, 10)

End of Unit 4