Computer Graphics

2D Viewing

Dr. Mousumi Dutt

CSE, STCET

Introduction

- A graphics package allows a user to specify **which part of a defined picture is to be displayed** and **where that part is to be placed** on the display device
- World coordinate reference frame is used to define picture
- Any convenient Cartesian coordinate system is referred to as world coordinate system
- View in 2D: by specifying a subarea of the total picture area
- To display:
  - A single area
  - Selected several areas for simultaneous display or for animation

Introduction

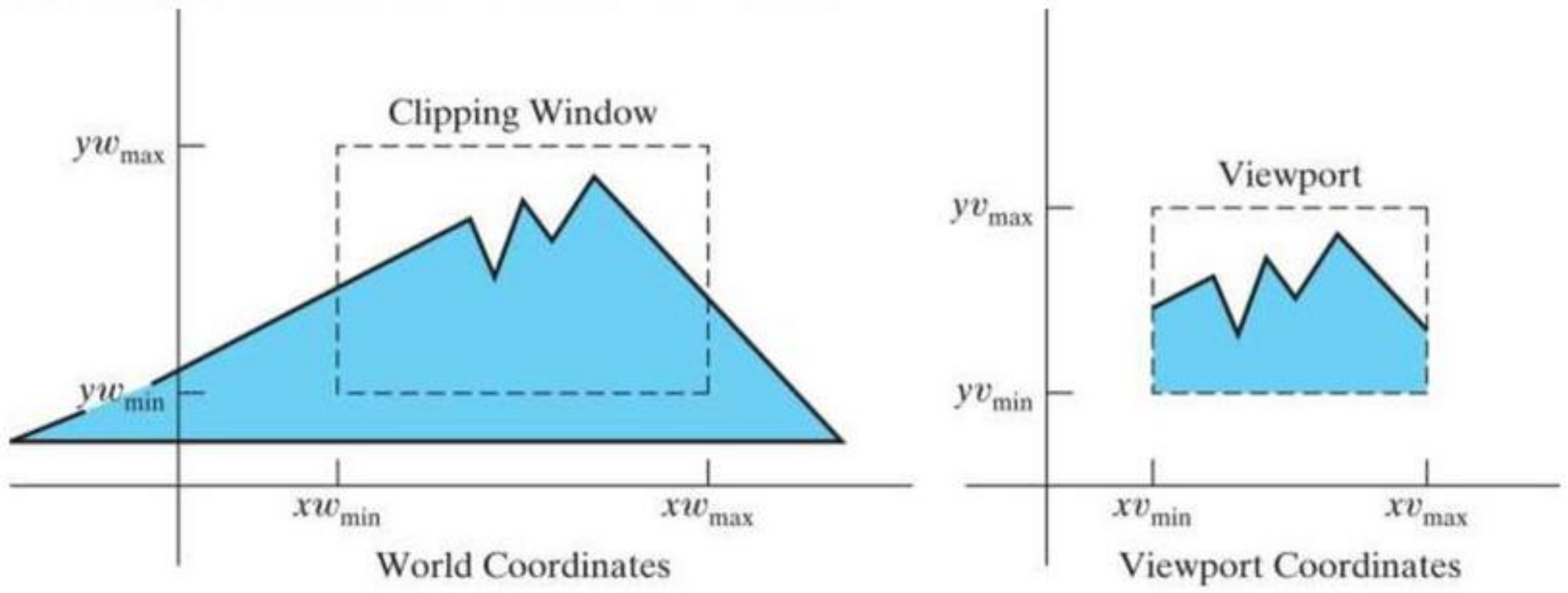
- Picture parts within the selected areas are then mapped onto specified areas of the device coordinates
- When multiple view areas are selected,
- these areas can be placed in separate display locations, or some areas could be inserted into other, larger display areas
- Transformations from world to device coordinates involve **translation, rotation, and scaling operations**, as well as procedures for deleting those parts of the picture that are outside the limits of a selected display area (clipping)

Viewing Pipeline

- A world-coordinate area selected for display is called a **window** (what to be viewed)
- An area on a display device to which a window is mapped is called a **viewport** (where to be displayed)
- Shape:
  - windows and viewports are rectangles in standard position, with the rectangle edges parallel to the coordinate axes
  - other window or viewport geometries, such as general polygon shapes and circles, are used in some applications, but these shapes take longer to process
- the mapping of a part of a world-coordinate scene to device coordinates is referred to as a **viewing transformation** or **windowing transformation** or **window-to-viewport transformation**

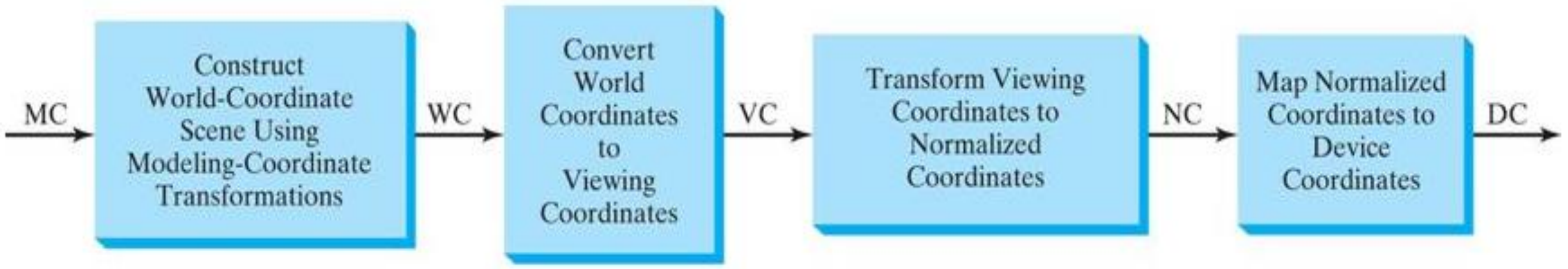
Viewing Pipeline

- Some graphics packages that provide window and viewport operations allow only standard rectangles, but a more general approach is to allow the rectangular window to have any orientation



Viewing Pipeline

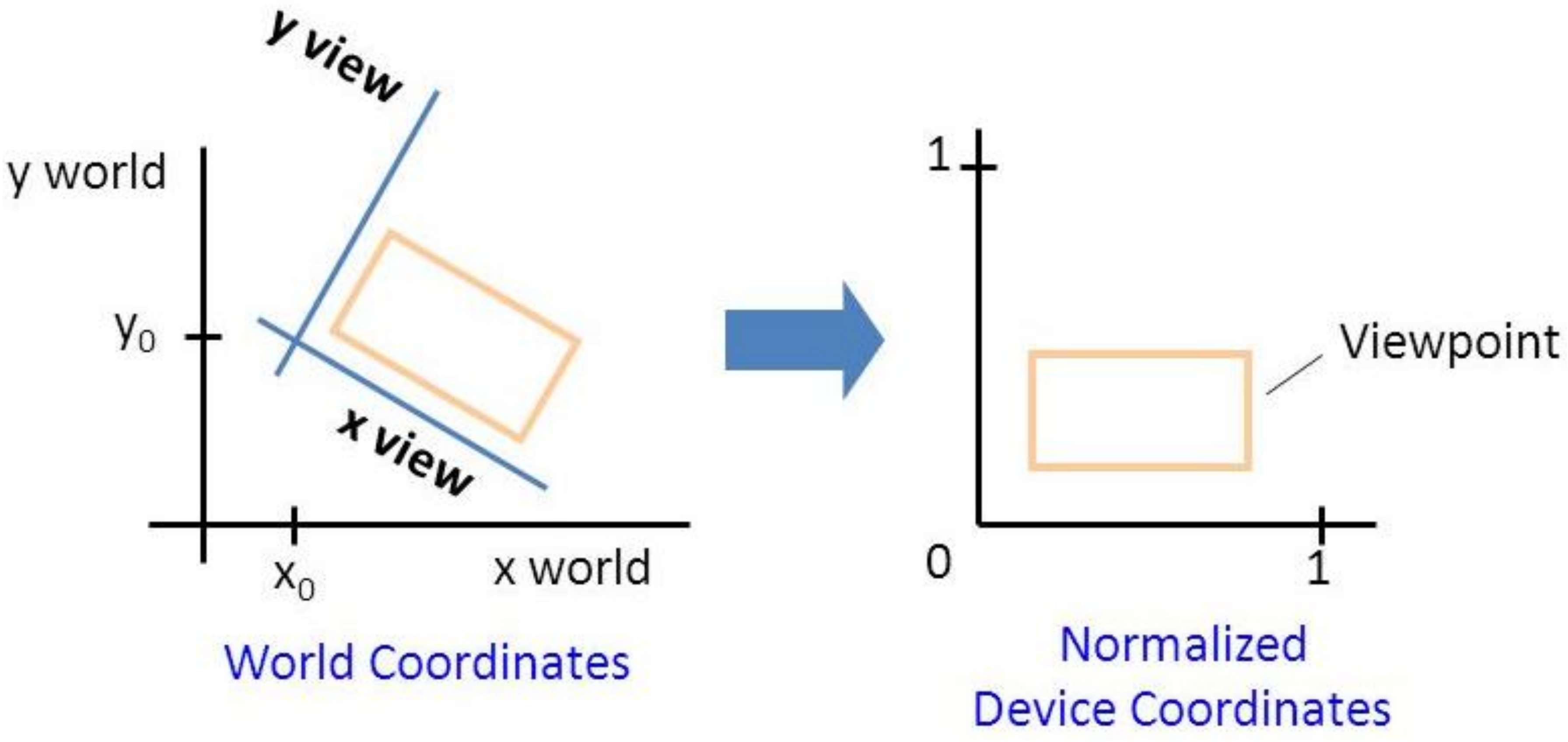
Two-dimensional viewing-transformation pipeline


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 Pipeline

Setting up a rotated world window in viewing coordinates and the corresponding normalized-coordinate viewport



Viewing and Related Effects

- By changing the position of the viewport, we can view objects **at different positions** on the display area of an output device
- By varying the size of viewports, we can change the **size and proportions of displayed objects**
- We achieve **zooming effects** by successively mapping different-sized windows on a fixed-size viewport
- As the windows are made smaller, we zoom in on some part of a scene **to view details** that are not shown with larger windows
- Similarly, more overview is obtained by **zooming out** from a section of a scene with successively larger windows
- **Panning effects** are produced by moving a **fixed-size window** across the various objects in a scene.

Importance of Normalized Coordinates

- Viewports are typically defined within the unit square **(normalized coordinates)**
- This provides a means for separating the viewing and other transformations from specific output-device requirements, so that the graphics package is largely **device-independent**
- Once the scene has been transferred to normalized coordinates, the unit square is simply mapped to the display area for the particular output device in use at that time
- Different output devices can be used by providing the appropriate device drivers

Importance of Clipping

- When all coordinate transformations are completed, viewport clipping can be performed in normalized coordinates or in device coordinates
- This allows us to **reduce computations** by concatenating the various transformation matrices
- Clipping procedures are of fundamental importance in computer graphics
- They are used not only in **viewing transformations**, but also in **window-manager systems**, in **painting and drawing** packages to eliminate parts of a picture inside or outside of a designated screen area, and in many other applications

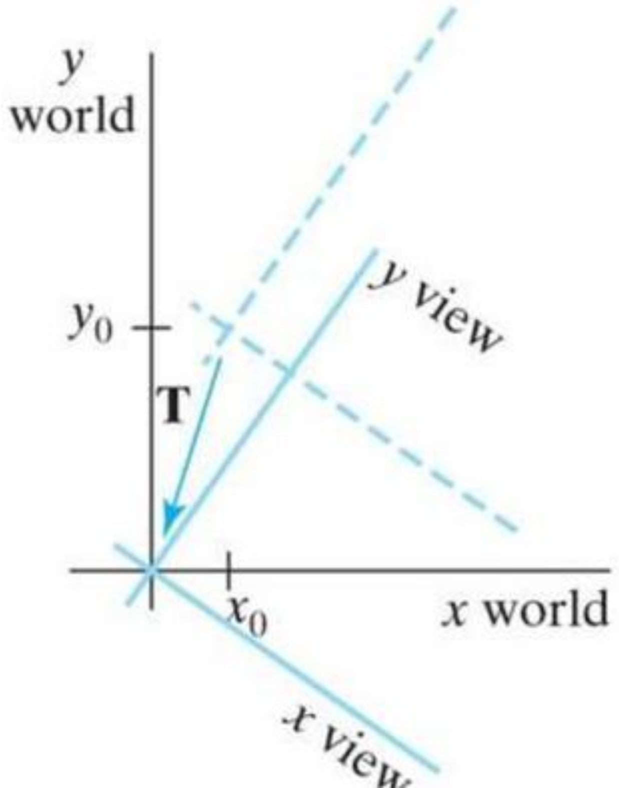
Viewing Coordinate Reference Frame

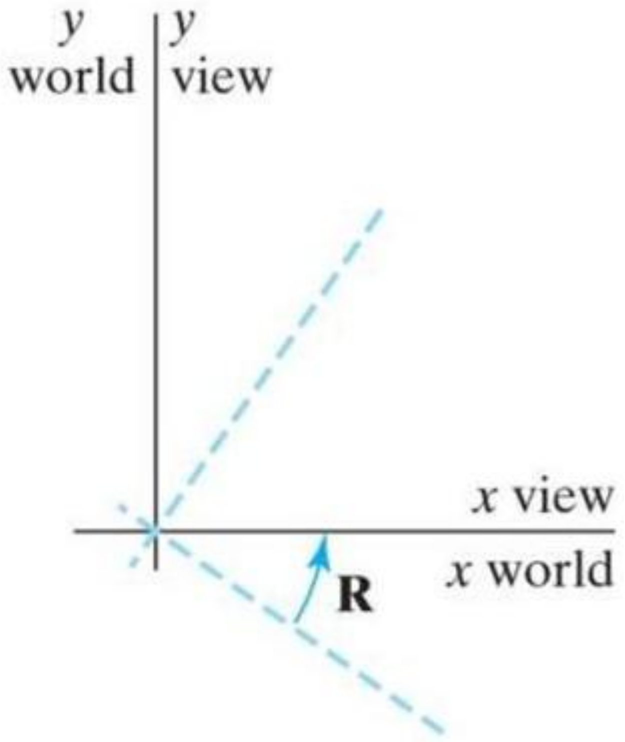
- This coordinate system provides the reference frame for specifying the world coordinate window
- A viewing-coordinate origin is selected at some world position: $P_0 = (x_0, y_0)$
- To establish the orientation, or rotation, of this reference frame
- One way to do this is to specify world vector **V** that defines the viewing y_v, direction
- Vector **V** is called the view up vector

Given V, we can calculate the components of unit vectors $v = (v_x, v_y)$ and $u = (u_x, u_y)$ for the viewing y_v, and x_v, axes, respectively. These unit vectors are used to form the first and second rows of the rotation matrix R that aligns the viewing $x_v y_v$ axes with the world $x_w y_w$ axes

Viewing Coordinate Reference Frame

- This coordinate system provides the reference frame for specifying the world coordinate window





Viewing Coordinate Reference Frame

- We obtain the matrix for converting world coordinate positions to viewing coordinates as a two-step composite transformation:
- First, we translate the viewing origin to the world origin, then we rotate to align the two coordinate reference frames
- The composite two-dimensional transformation to convert world coordinates to viewing coordinate is

$$\mathbf{M}_{WC,VC}=\mathbf{R}.\mathbf{T}$$

- where **T** is the translation matrix that takes the viewing origin point $\mathbf{P}_0$ to the world origin, and **R** is the rotation matrix that aligns the axes of the two reference frames

Window to Viewport Coordinate Transformation

- Once object descriptions have been transferred to the viewing reference frame, we choose the window extents in viewing coordinates and select the viewport limits in normalized coordinates
- Object descriptions are then transferred to normalized device coordinates
- We do this using a transformation that maintains the same relative placement of objects in normalized space as they had in viewing coordinates
- Example:

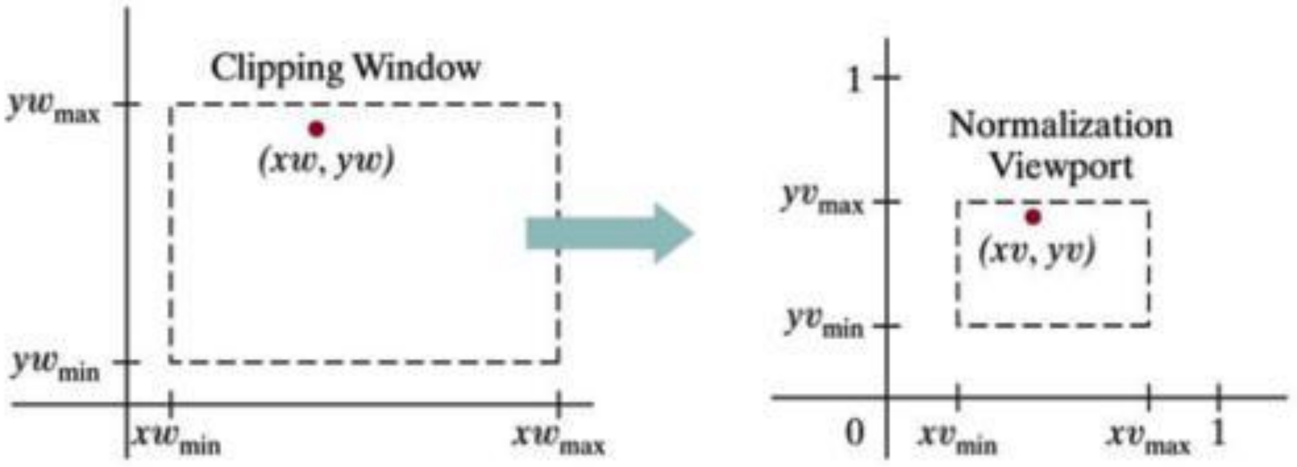
If a coordinate position is at the center of the viewing window, for instance, it will be displayed at the center of the viewport.

Window to Viewport Coordinate Transformation

- A point at position (x_w,y_w) in the window is mapped into position (x_v, y_v) in the associated viewport
- To maintain the same relative placement in the viewport as in the window, we require that

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

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



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 Coordinate Transformation

Solving the expression for the viewport position (xv, yv)

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

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

- Equations can also be derived with a set of transformations that converts the window area into the viewport area
- This conversion is performed with the following sequence of transformations:

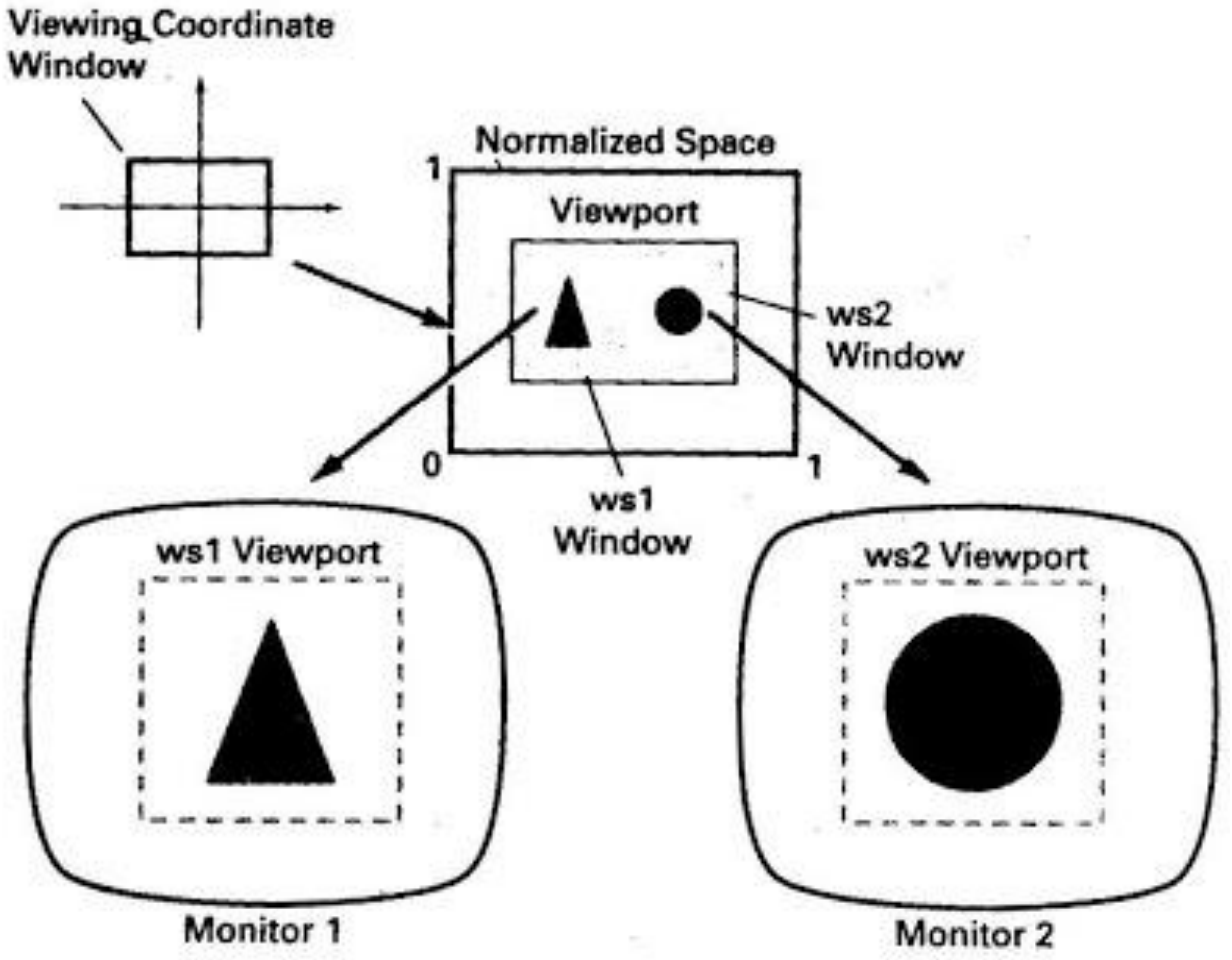
1. Perform a scaling transformation using a fixed-point position of (xw_{min},yw_{min}) that scales the window area to the size of the viewport
2. Translate the scaled window area to the position of the viewport.

Window to Viewport Coordinate Transformation

- Relative proportions of objects are maintained if the scaling factors are the same (sx = sy)
- Otherwise, world objects will be stretched or contracted in either the x or y direction when displayed on the output device
- Character strings can be handled in two ways when they are mapped to a viewport
- For characters formed with line segments, the mapping to the viewport can be carried out as a sequence of line transformations
- **Workstation Transformation: From normalized coordinates, object descriptions are mapped to the various display devices. Any number of output devices can be open in a particular application, and another window-to-viewport transformation can be performed for each open output device**
- **accomplished by selecting a window area in normalized space and a viewport area in the coordinates of the display device**

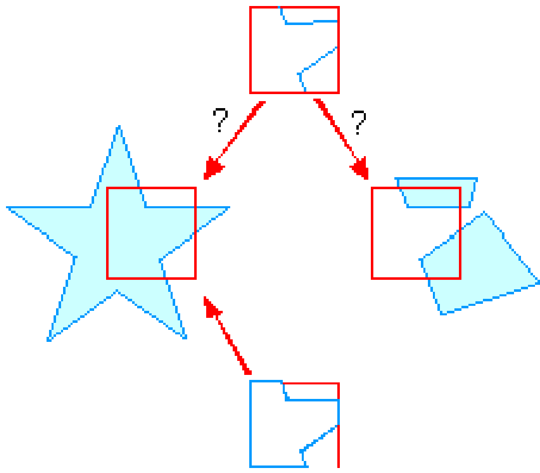
Window to Viewport Coordinate Transformation

With the workstation transformation, we gain some additional control over the positioning of parts of a scene on individual output devices



Clipping Operation

- **Clipping Algorithm/ Clipping:** any procedure that identifies those portions of a picture that are either inside or outside of a specified region of space
- **Clip Window:** The region against which an object is to clipped



Application:

➢extracting part of a detined scene for viewing

➢identifying visible surfaces in three-dimensional views

➢antialiasing line segments or object boundaries

➢creating objects using solid-modeling procedures

➢displaying a multiwindow environment

➢drawing and painting operations that allow parts of a picture to be selected for copying, moving, erasing, or duplicating

Clipping Operation

Clip Window:

- Polygon or curved boundary depending on the application

Viewing Transformation:

- To display those picture parts that are within the window area
- Everything outside the window is discarded
- Clipping algorithms can be applied in world coordinates, so that only the contents of the window interior are mapped to device coordinates
- Alternatively, the complete world-coordinate picture can be mapped first to device coordinates, or normalized device coordinates, then clipped against the viewport boundaries

Clipping Operation

- Viewport clipping, on the other hand, can reduced calculations by allowing concatenation of viewing and geometric transformation matrices
- viewport clipping does require that the transformation to device coordinates be performed for all objects, including those outside the window area
- On raster systems, clipping algorithms are often combined with scan conversion
- Clipping of primitive elements are easier
- To handle curved objects is to approximate them with straight-line segments and apply the line- or polygon clipping procedure

Point Clipping

- **Assumption:** Clip window is a rectangle
- The point P = (x, y) is saved for display if the following inequalities are satisfied:

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

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

- the edges of the clip window (xw_{min}, xw_{max}, yw_{min}, yw_{max},) can be either the world-coordinate window boundaries or viewport boundaries
- If any one of these four inequalities is not satisfied, the point is clipped (not saved for display)
- **Application:** scenes involving explosions or sea foam that are modelled with particles (points) distributed in some region of the scene

Line Clipping

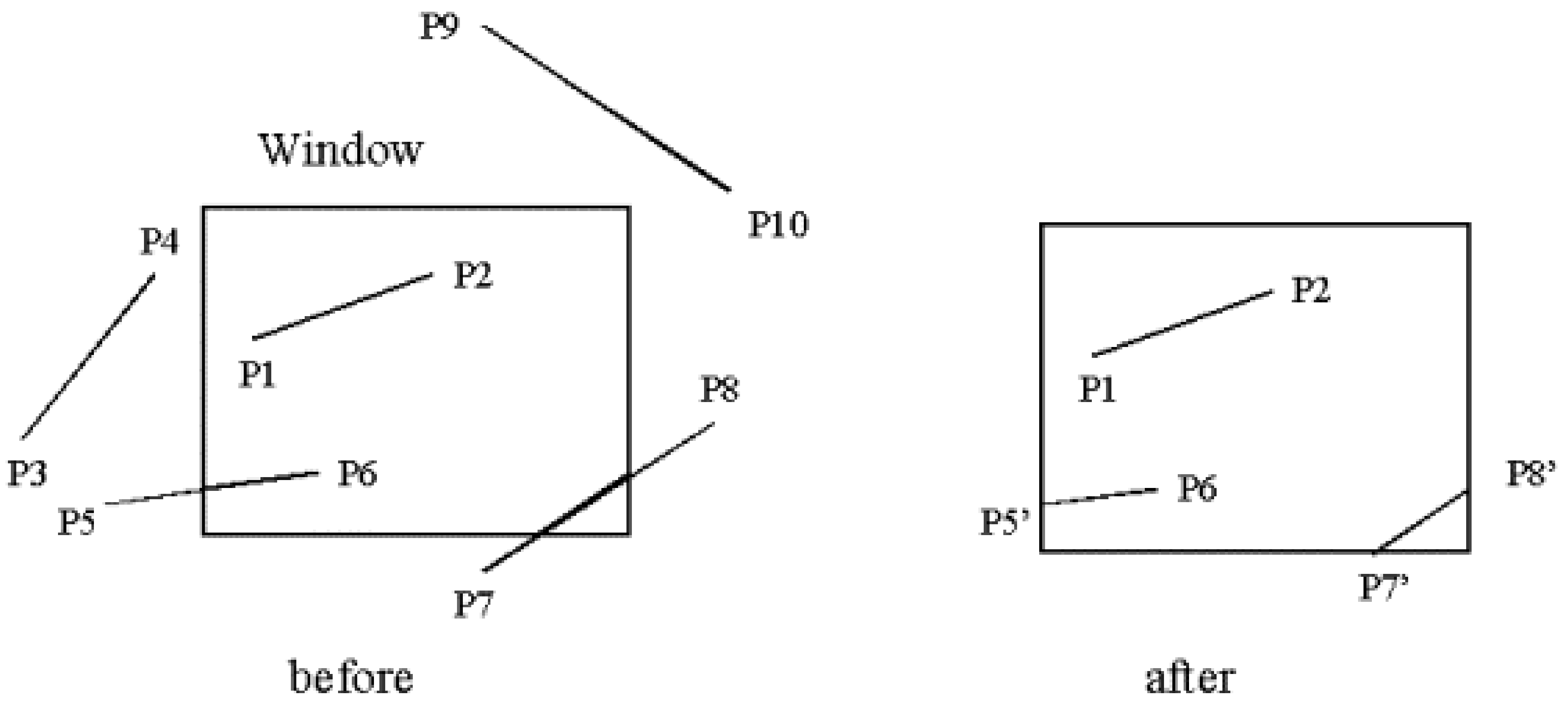
- To find line position w.r.t. a standard rectangular clipping region

Steps:

1) Test the line whether it is completely inside the clipping window
2) If not, then test whether it is completely outside the clipping window
3) Otherwise perform intersection calculations of line with one or more clipping boundaries

- We process lines through the "inside-outside" tests by checking the line endpoints

Line Clipping and Requirement for Clipping Algorithm



❖Lines cross one or more clipping boundaries may require calculation of multiple intersection points

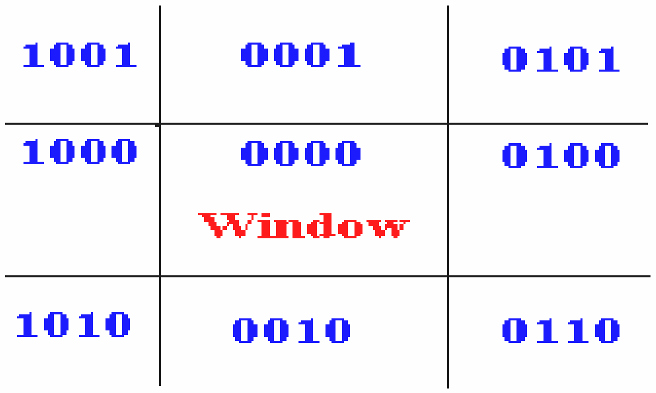
❖To minimize calculations, clipping algorithms are devised that can efficiently identify outside lines and redraw intersection calculations

Line Clipping

- **The parametric representation of clipping boundary is**
- **$x=x_1+u(x_2-x_1)$**
- **$y=y_1+u(y_2-y_1)$, $0<= u<= 1$**
- To determine values of parameter u for intersections with the clipping boundary coordinates
- The line is outside when the value of u is beyond the range
- Otherwise, find intersections
- **Special handling for the lines parallel to window edges**
- Parametric test: good deal of computation, faster approaches to clipping is possible

Cohen-Sutherland Line Clipping

- **Oldest**
- **Most popular**
- speeds up the processing of line segments
  - by performing initial tests
  - reduces the number of intersections that must be calculated
- Every line end-point in a picture is assigned a four-digit binary code, called a region code, that identifies the location of the point relative to the boundaries of the clipping rectangle



Cohen-Sutherland Line Clipping

- Each bit position in the region code is used to indicate one of the four relative coordinate positions of the point with respect to the clip window: to the left, right, top, or bottom
- By numbering the bit positions in the region code as 1 through 4 from right to left, the coordinate regions can be correlated with the bit positions as
  - bit 1: left
  - bit 2: right
  - bit 3: below
  - bit 4: above
- A value of 1 in any bit position indicates that the point is in that relative position; otherwise, the bit position is set to 0
- **If a point is within the clipping rectangle the region code is 0000**
- **A point that is below and to the left of the rectangle has a region code of 0101**

Cohen-Sutherland Line Clipping

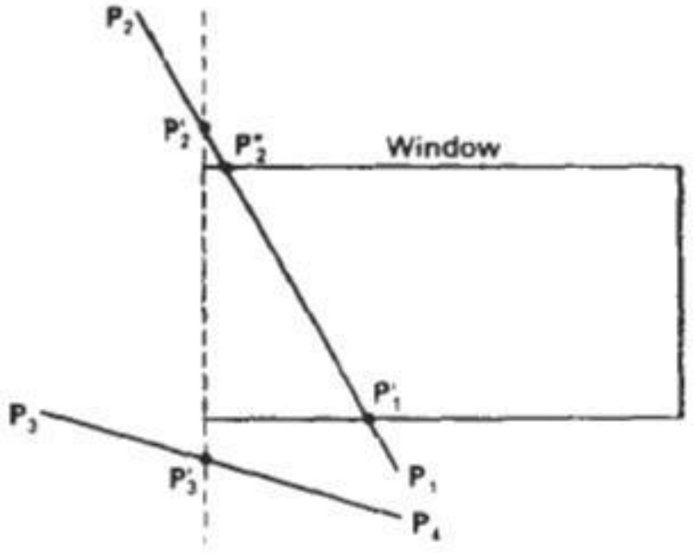
- Calculate differences between endpoint coordinates
- and clipping boundaries
- Use the resultant sign bit of each difference calculation to set the corresponding value in the region code
- **Bit 1 is the sign bit of $x – xw_{min}$**
- **Bit 2 is the sign bit of $xw_{max} - x$**
- **Bit 3 is the sign bit of $y – yw_{min}$**
- **Bit 4 is the sign bit of $yw_{max} – y$**
- Any lines that are completely contained within the window boundaries have a region code of 0000 for both endpoints, and we trivially accept these lines

Cohen-Sutherland Line Clipping

- Any lines that have a 1 in the same bit position in the region codes for each endpoint are completely outside the clipping rectangle, and we trivially reject these lines
- We would discard the line that has a region code of 1001 for one endpoint and a code of 0101 for the other endpoint
- Both endpoints of this line are left of the clipping rectangle, as indicated by the 1 in the first bit position of each region code
- **A method that can be used to test lines for total clipping is to perform the logical and operation with both region codes**
- If the result is not 0000, the line is completely outside the clipping region
- Lines that cannot be identified as completely inside or completely outside a clip window by these tests are checked for intersection with the window boundaries.

Cohen-Sutherland Line Clipping

- We begin the clipping process for a line by comparing an outside endpoint to a clipping boundary to determine how much of the line can be discarded
- Then the remaining part of the line is checked against the other boundaries, and we continue until either the line is totally discarded or a section is found inside the window
- We set up our algorithm to check line endpoints against clipping boundaries in the order left, right, bottom, top



Cohen-Sutherland Line Clipping

- Intersection points with a clipping boundary can be calculated using the slope-intercept form of the line equation
- For a line with the endpoint coordinates (x_1, y_1) and (x_2, y_2), the y-coordinate of the intersection point with a vertical boundary can be obtained with the calculation, $y = y_1 + m(x - x_1)$, where x is either xw_{min} or xw_{max}
- For the intersection with a horizontal boundary, the x coordinate can be calculated as $x = x_1 + (y - y_1)/m$, where y is either yw_{min} or yw_{max}

Liang-Barsky Line Clipping

- Faster line clippers have been developed that are based on analysis of the parametric equation of a line segment, which we can write in the form

$$x = x_1 + u\Delta x$$
$$y = y_1 + u\Delta y, \qquad 0 \le u \le 1$$

where $\Delta x = x_2 - x_1$ and $\Delta y = y_2 - y_1$

- Using these parametric equations, Cyrus and Beck developed an algorithm that is generally more efficient than the Cohen-Sutherland algorithm
- Later, Liang and Barsky independently devised an even faster parametric line-clipping algorithm
- Following the Liang-Barsky approach, we first write the point-clipping conditions in the parametric form

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

Liang-Barsky Line Clipping

Each of these four inequalities can be expressed as

$$u p_k \le q_k, \qquad k = 1, 2, 3, 4$$

where parameters p and q are defined as

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

Liang-Barsky Line Clipping

- Any line that is parallel to one of the clipping boundaries has $p_k = 0$ for the value of k corresponding to that boundary (k = 1, 2, 3, and 4 correspond to the left, right, bottom, and top boundaries, respectively)
- If, for that value of k, we also find $q_k < 0$, then the line is completely outside the boundary and can be eliminated from further consideration
- If $q_k >= 0$, the line is inside the parallel clipping boundary
- When $p_k < 0$, the infinite extension of the line proceeds from the outside to the inside of the infinite extension of this particular clipping boundary
- When $p_k > 0$ the line proceeds from the inside to the outside

Liang-Barsky Line Clipping

- For a nonzero value of p_k, we can calculate the value of u that corresponds to the point where the infinitely extended line intersects the extension of boundary k as

$$u = \frac{q_k}{p_k}$$

- For each line, we can calculate values for parameters u_1 and u_2 that define that part of the line that lies within the clip rectangle
- The value of u_1 is determined by looking at the rectangle edges for which the line proceeds from the outside to the inside ($p < 0$)
- For these edges, we calculate $r_k = q_k/p_k$
- The value of u_1 is taken as the largest of the set consisting of 0 and the various values of r

Liang-Barsky Line Clipping

- The value of u_2 is determined by examining the boundaries for which the line proceeds from inside to outside ($p > 0$)
- A value of r_k , is calculated for each of these boundaries, and the value of u_2, is the minimum of the set consisting of 1 and the calculated r values

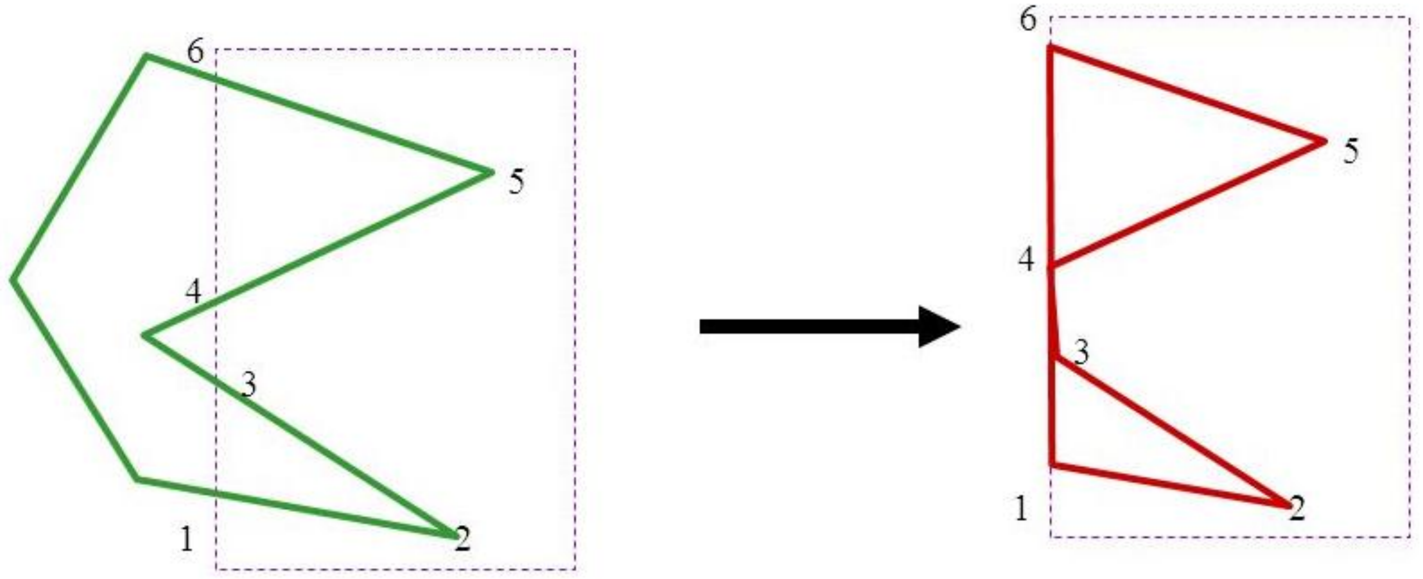
- If $u_1 > u_2$, the line is completely outside the clip window and it can be rejected
- Otherwise, the endpoints of the clipped line are calculated from the two values of parameter u

Liang-Barsky Line Clipping: Summary

- Line intersection parameters are initialized to the values $u_1 = 0$ and $u_2 = 1$
- For each clipping boundary, the appropriate values for p and q are calculated
- When $p < 0$, the parameter r is used to update u_1
- When $p > 0$, parameter r is used to update u_2
- If updating u_1 or u2 results in $u_1 > u_2$, we reject the line
- Otherwise, we update the appropriate u parameter only if the new value results in a shortening of the line
- When $p = 0$ and $q < 0$, we can discard the line since it is parallel to and outside of this boundary
- If the line has not been rejected after all four values of p and q have been tested, the endpoints of the clipped line are determined from values of u_1 and u_2

Polygon Clipping

- A polygon boundary processed with a line clipper may be displayed as a series of unconnected line segments depending on the orientation of the polygon to the clipping window
- For polygon clipping, we require an algorithm that will generate one or more closed areas that are then scan converted for the appropriate area fill
- The output of a polygon clipper should be a sequence of vertices that defines the clipped polygon boundaries



Sutherland-Hodgeman Polygon Clipping

- We can correctly clip a polygon by processing the polygon boundary as a whole against each window edge
- Beginning with the initial set of polygon vertices, we could first clip the polygon against the left rectangle boundary to produce a new sequence of vertices
- The new set of vertices could then be successively passed to a right boundary clipper, a bottom boundary clipper, and a top boundary clipper
- At each step, a new sequence of output vertices is generated and passed to the next window boundary clipper

Polygon Clipping

Computer Graphics

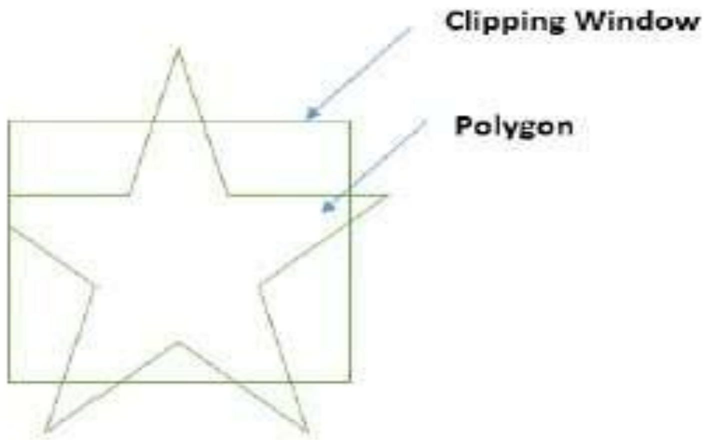


Figure: Clipping Left Edge

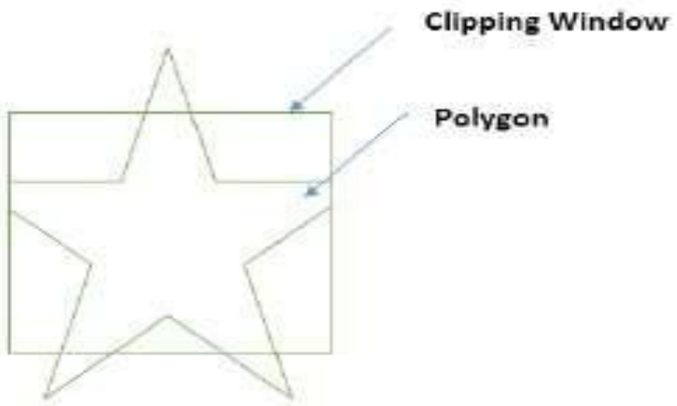


Figure: Clipping Right Edge

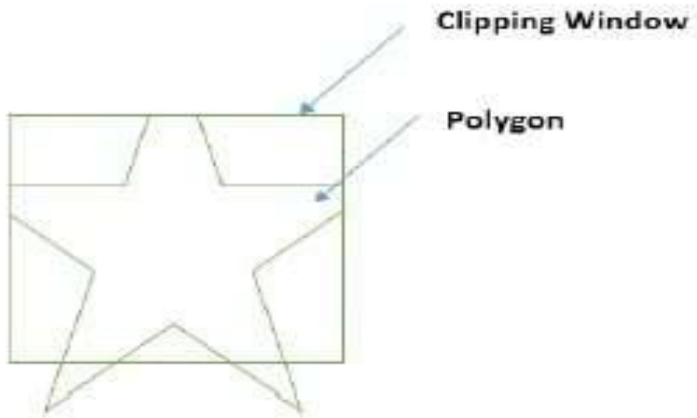


Figure: Clipping Top Edge

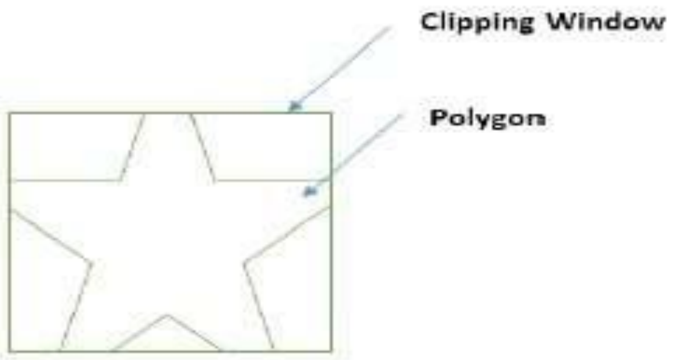


Figure: Clipping Bottom Edge

Sutherland-Hodgeman Polygon Clipping

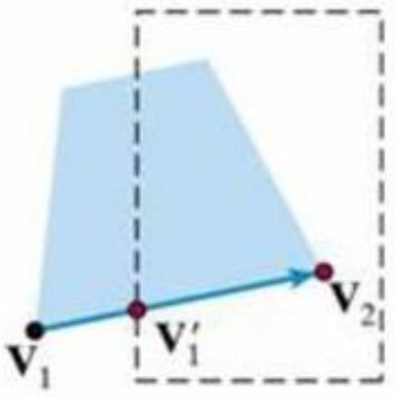
- There are four possible cases when processing vertices in sequence around the perimeter of a polygon
- As each pair of adjacent polygon vertices is passed to a window boundary clipper, we make the following tests:

1. If the first vertex is outside the window boundary and the second vertex is inside, both the intersection point of the polygon edge with the window boundary and the second vertex are added to the output vertex list
2. If both input vertices are inside the window boundary, only the second vertex is added to the output vertex list
3. If the first vertex is inside the window boundary and the second vertex is outside, only the edge intersection with the window boundary is added to the output vertex list

Sutherland-Hodgeman Polygon Clipping

4. If both input vertices are outside the window boundary, nothing is added to the output list

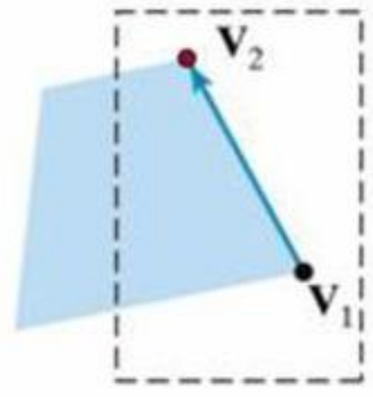
Once all vertices have been processed for one clip window boundary, the output list of vertices is clipped against the next window boundary



(1)

out $\longrightarrow$ in

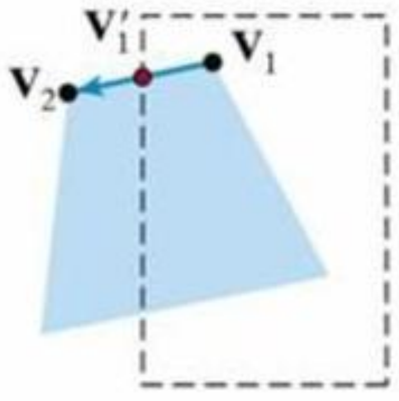
Output: V_1', V_2



(2)

in $\longrightarrow$ in

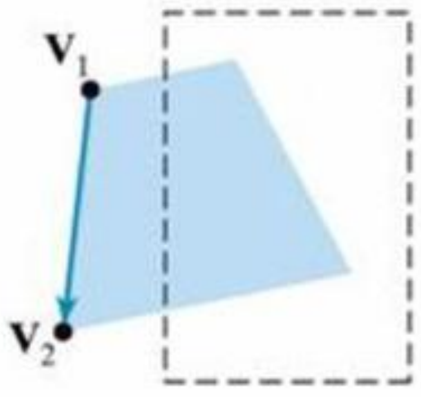
Output: V_2



(3)

in $\longrightarrow$ out

Output: V_1'



(4)

out $\longrightarrow$ out

Output: none

Sutherland-Hodgeman Polygon Clipping

Storage for an output list of vertices as a polygon is clipped against each window boundary

The intermediate output vertex lists can be eliminated by simply clipping individual vertices at each step and passing the clipped vertices on to the next boundary clipper

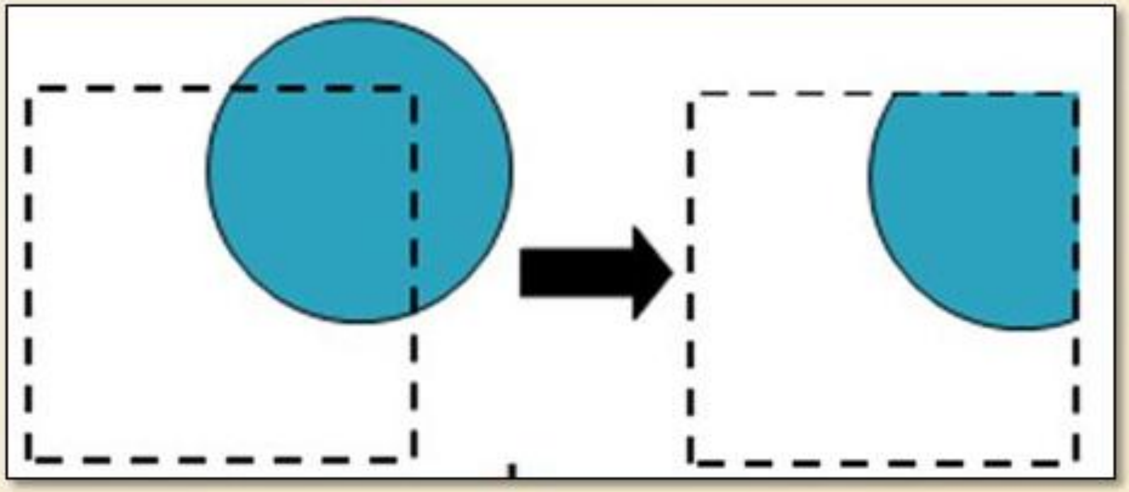
Can be done with parallel processors or a single processor and a pipeline with clipping routine

A point (either an input vertex or a calculated intersection point) is added to the output vertex list only after it has been determined to be inside or on a window boundary by all four boundary clippers

Curve Clipping

- ➢Curve clipping procedures will involve non-linear equations (so requires more processing than for objects with linear boundaries In general, methods depend on how characters are represented).
- ➢Clipping curves requires more work

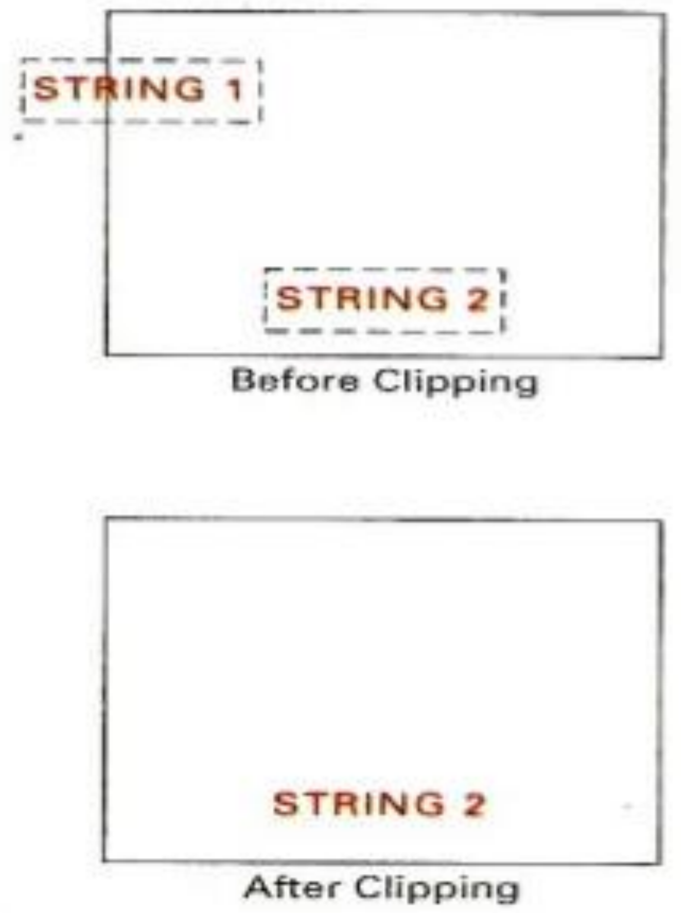
  For circles we must find the two intersection points on the window boundary



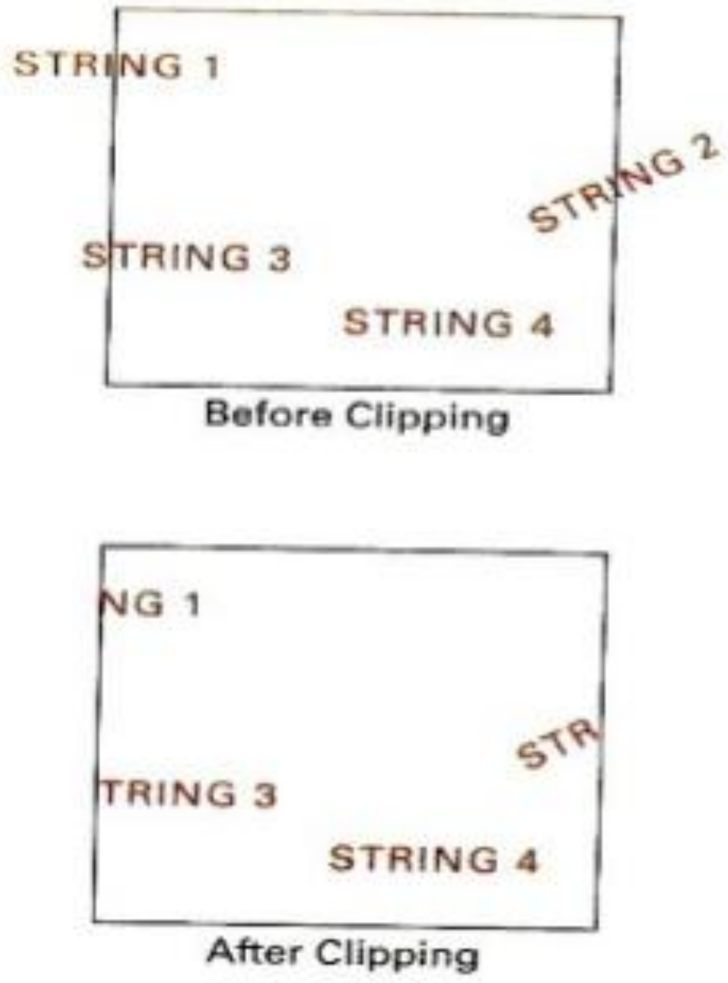
Curve Clipping

- Preliminary test (Test for overlapping)
  - The bounding rectangle for a circle or other curved object is used to test for overlap with a rectangular clip window.
  - If the bounding rectangle is completely inside (save object), completely outside (discard the object)
  - Both cases- no computation is necessary.
  - If bounding rectangle test fails, use computation-saving approaches
- Circle – coordinate extents of individual quadrants & then octants are used for preliminary testing before calculating curve–window intersections
- Ellipse – coordinate extents of individual quadrants are used.
- If 2 regions overlap, solve the simultaneous line-curve equations to obtain the clipping intersection points.

Text Clipping



Text clipping using a bounding rectangle about the entire string.

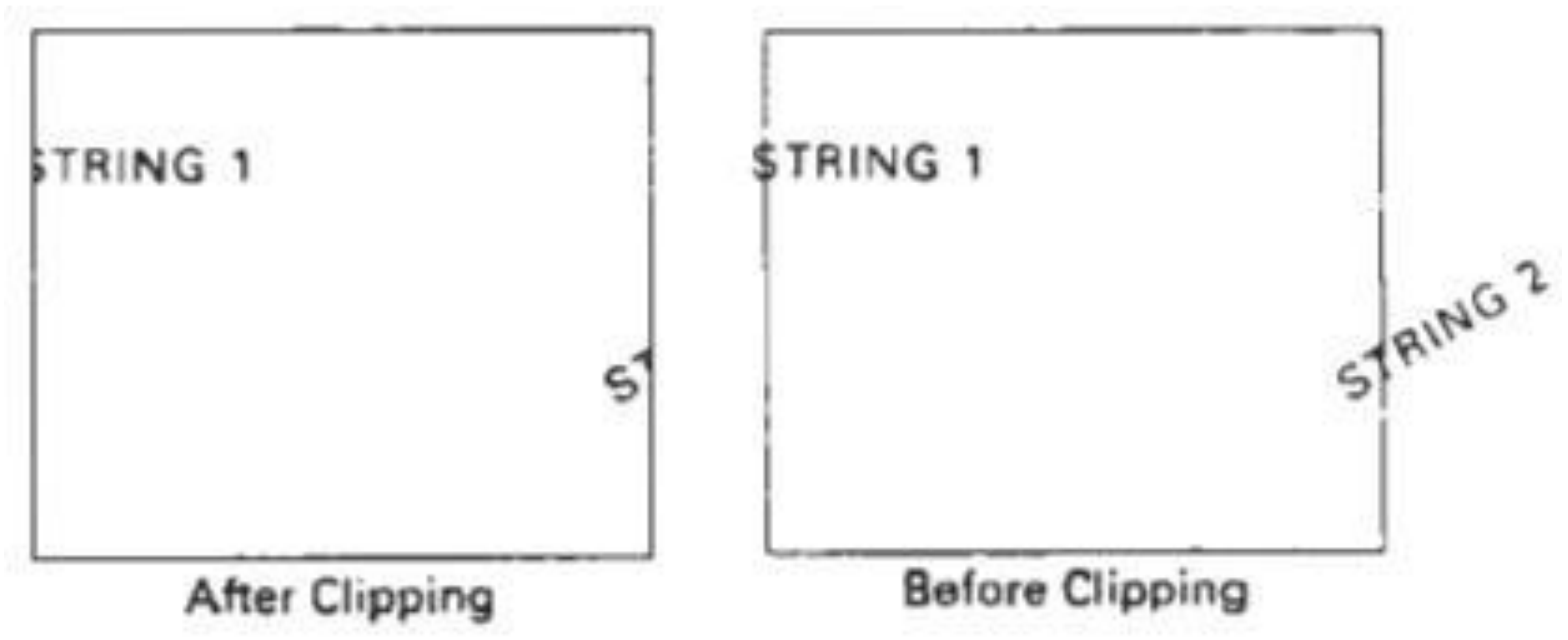


Text clipping using a bounding rectangle about individual characters.

Text Clipping

- **All-or-none string clipping strategy**
  - Simplest method, fastest text clipping
  - All string - inside clip window, keep it, otherwise discard.
  - Bounding rectangle considered around the text pattern
  - If bounding position of rectangle overlap with window boundaries, string is rejected.
- **All or none character clipping strategy**
  - Discard or reject an entire character string that overlaps a window boundary i.e, discard those characters that are not completely inside the window.
  - Compare boundary limits of individual characters with the window.
  - Any character which is outside or overlapping the window boundary are clipped.

Text Clipping



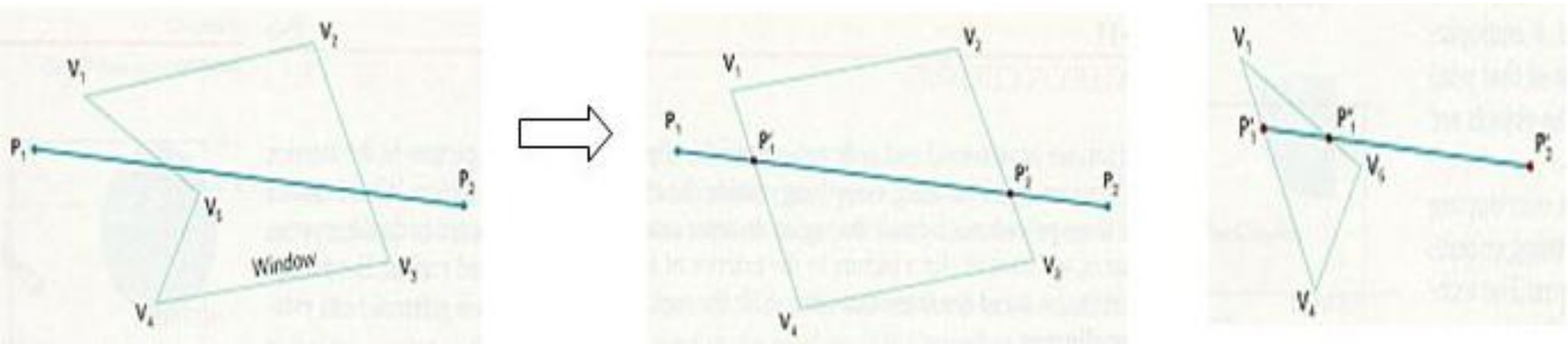
Exterior Clipping

Save the outside region

Applications

- Multiple window systems
- The design of page layouts in advertising or publishing
- Adding labels or design patterns to a picture

Procedures for clipping objects to the interior of concave polygon windows



V_1
V_2
P_1
V_5
P_2
Window
V_3
V_4
V_1
V_2
P_1
P'_1
P'_2
P_2
V_3
V_4
V_1
P'_1
P''_1
P'_2
V_5
V_4