Two-Dimensional Viewing & Clipping

COMPUTER GRAPHICS

Contents

Viewing Coordinate System

- Much like what we see in real life through a small window on the wall or the viewfinder of a camera, a computer-generated image often depicts a partial view of a large scene.
- Objects are placed into the scene by modeling transformations to a master coordinate system, commonly referred to as the ***world coordinate system (WCS)***.
- A rectangular window with its edges parallel to the axes of the ***WCS*** is used to select the portion of the scene for which an image is to be generated.
- Sometimes an additional coordinate system called the ***viewing coordinate system*** is introduced to simulate the effect of moving and/or tilting the camera.

Viewing Coordinate System

- An image representing a view often becomes part of a larger image.
- Since monitor sizes differ from one system to another, we want to introduce a device-independent tool to describe the display area. This tool is called the ***normalized device coordinate system (NDCS)*** in which a unit ***(1 x 1)*** square whose lower left corner is at the origin of the coordinate system defines the display area of a virtual display device.
- A rectangular viewport with its edges parallel to the axes of the NDCS is used to specify a sub-region of the display area that embodies the image.

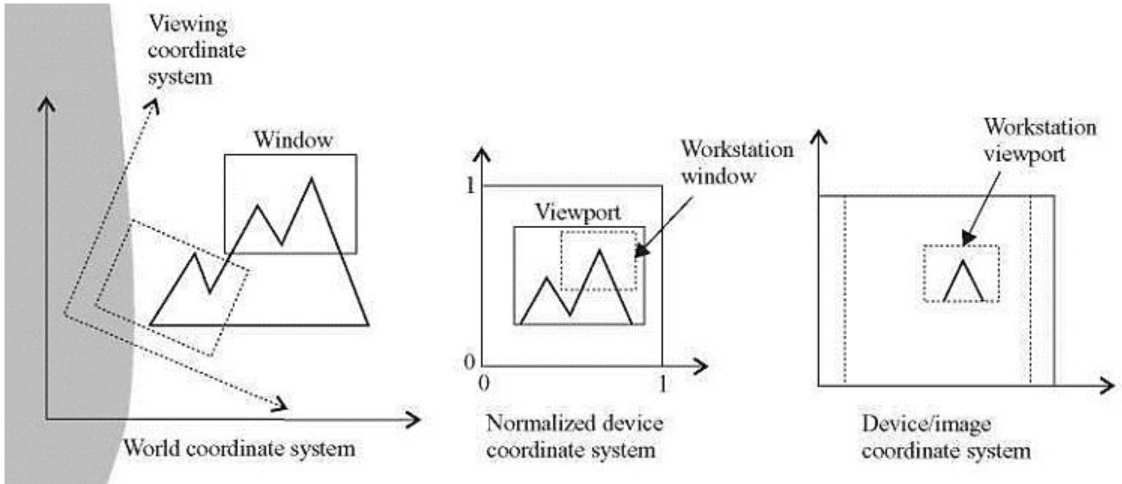

Viewing coordinate system
Window
World coordinate system
Viewport
Workstation window
1
0
0
1
Normalized device coordinate system
Workstation viewport
Device/image coordinate system

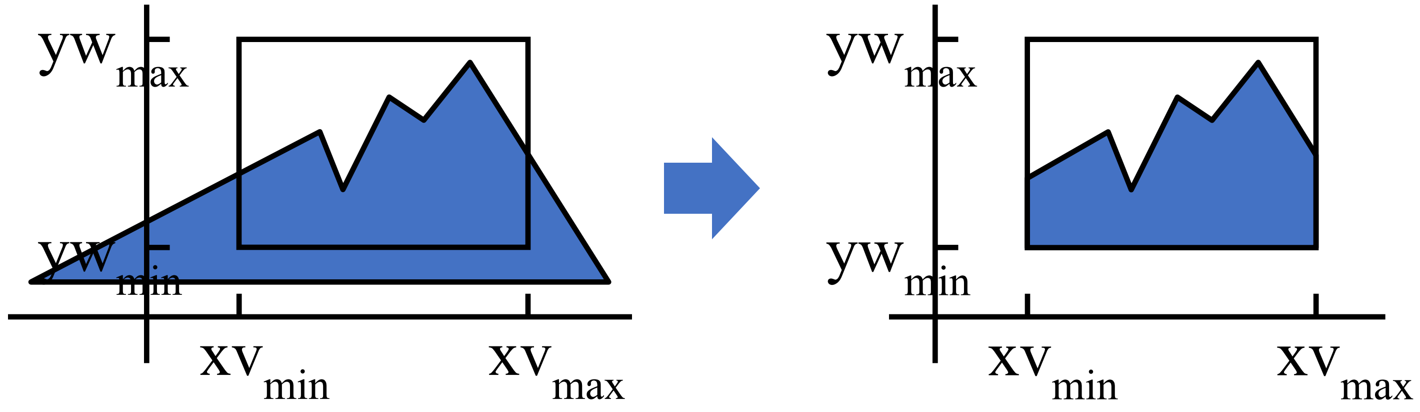
Fig. 5-1 Viewing transformation.

Viewing Coordinate System

- The process that converts object coordinates in WCS to normalized device coordinates is called ***window-to-viewport mapping* *or normalization transformation***.
- The process that maps normalized device coordinates to discrete device/image coordinates is called ***workstation transformation***,
- Collectively, these two coordinate-mapping operations are referred to as ***viewing transformation***.

The Viewing Pipeline

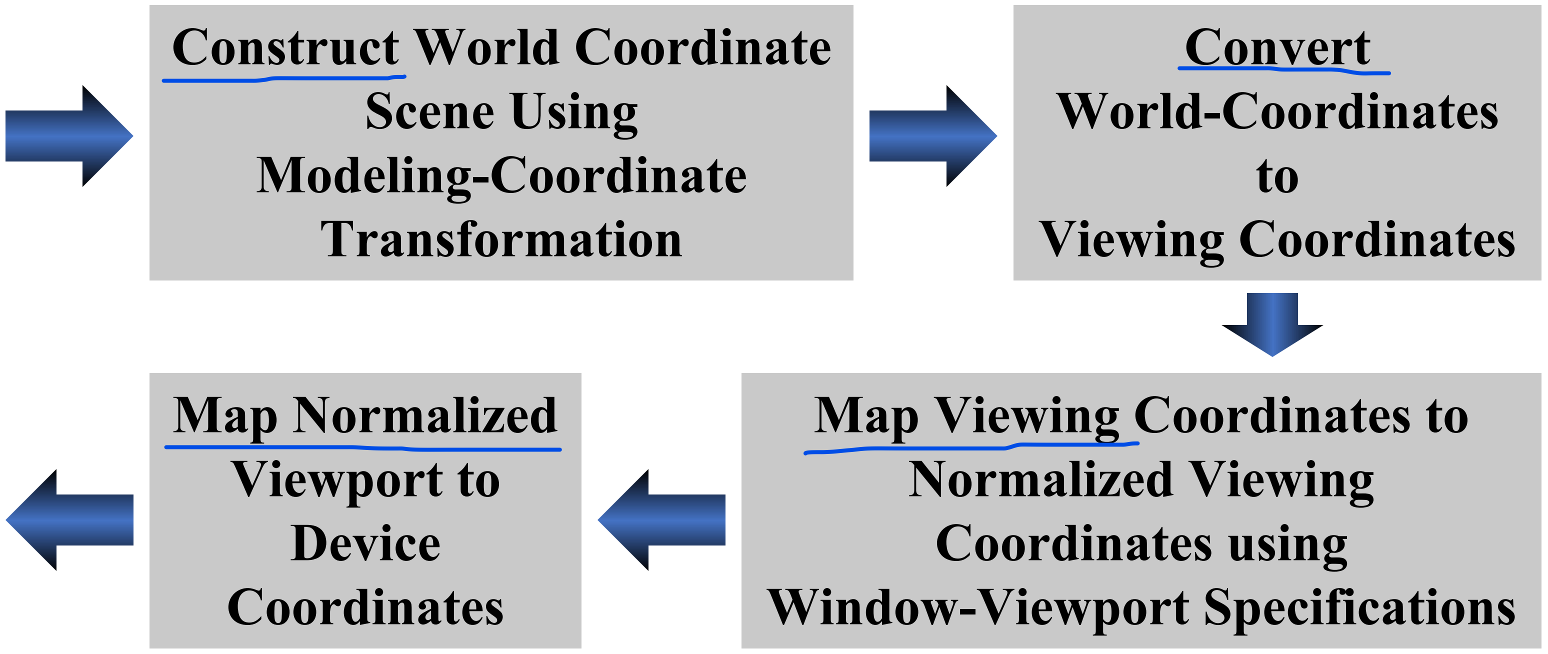
- What's the viewing pipe line??
  - Viewing transformation in several steps
- A viewing transformation using standard rectangles for the window and viewport

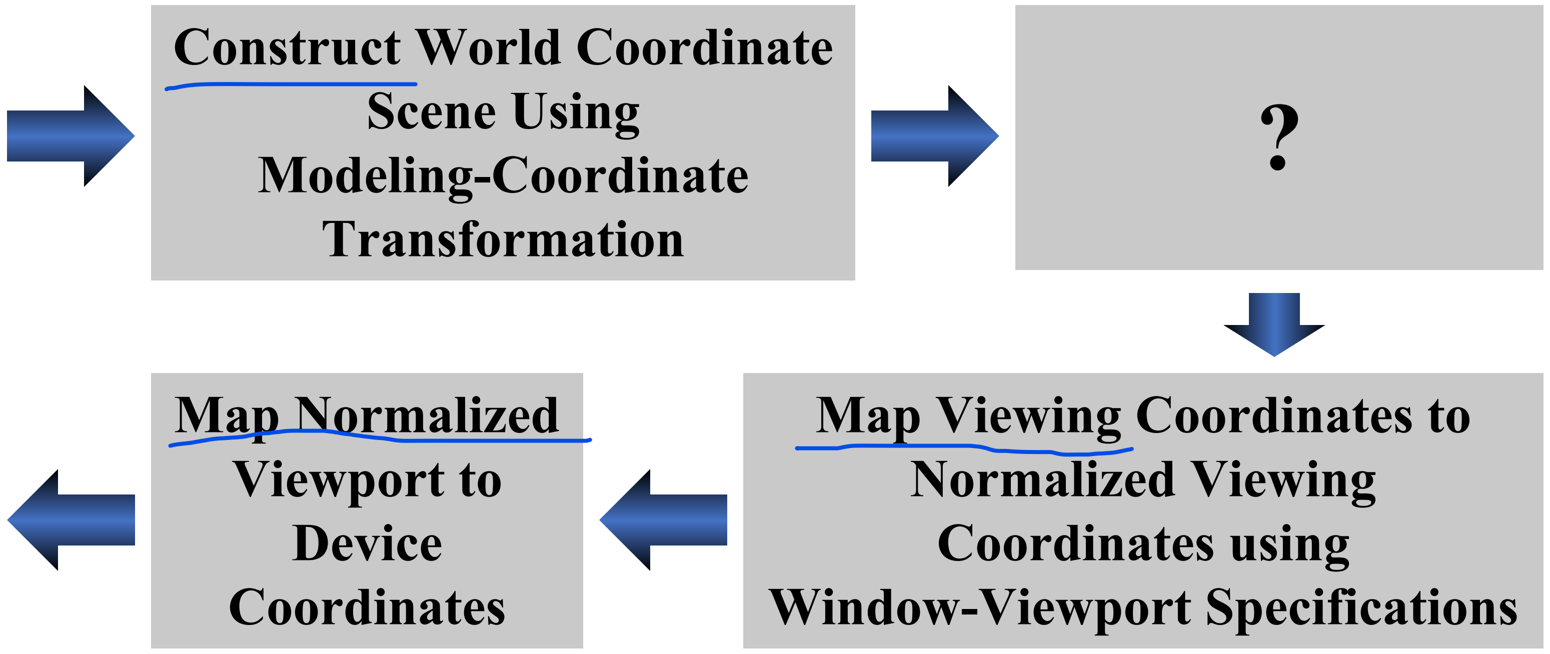


World Coordinates   Device Coordinate

The Viewing Pipeline

- The two-dimensional viewing-transformation pipeline


Construct World Coordinate Scene Using Modeling-Coordinate Transformation
Convert World-Coordinates to Viewing Coordinates
Map Viewing Coordinates to Normalized Viewing Coordinates using Window-Viewport Specifications
Map Normalized Viewport to Device Coordinates


Construct World Coordinate Scene Using Modeling-Coordinate Transformation
?
Map Viewing Coordinates to Normalized Viewing Coordinates using Window-Viewport Specifications
Map Normalized Viewport to Device Coordinates

Window-To-Viewport Coordinate Transformation

- A window is specified by four world coordinates: wx_min, wx_max, wy_min, and wy_max
- Similarly, a viewport is described by four normalized device coordinates: vx_min, vx_max, vy_min, and vy_max
- The objective of window-to-viewport mapping is to convert the world coordinates *(wx, wy)* of an arbitrary point to its corresponding normalized device coordinates *(vx, vy)*.
- In order to maintain the same relative placement of the point in the viewport as in the window, we require:

$$\frac{wx - wx_{\min}}{wx_{\max} - wx_{\min}} = \frac{vx - vx_{\min}}{vx_{\max} - vx_{\min}} \quad \text{and} \quad \frac{wy - wy_{\min}}{wy_{\max} - wy_{\min}} = \frac{vy - vy_{\min}}{vy_{\max} - vy_{\min}}$$

Window-To-Viewport Coordinate Transformation

$$\begin{cases} vx = \dfrac{vx_{\max} - vx_{\min}}{wx_{\max} - wx_{\min}}(wx - wx_{\min}) + vx_{\min} \\ vy = \dfrac{vy_{\max} - vy_{\min}}{wy_{\max} - wy_{\min}}(wy - wy_{\min}) + vy_{\min} \end{cases}$$

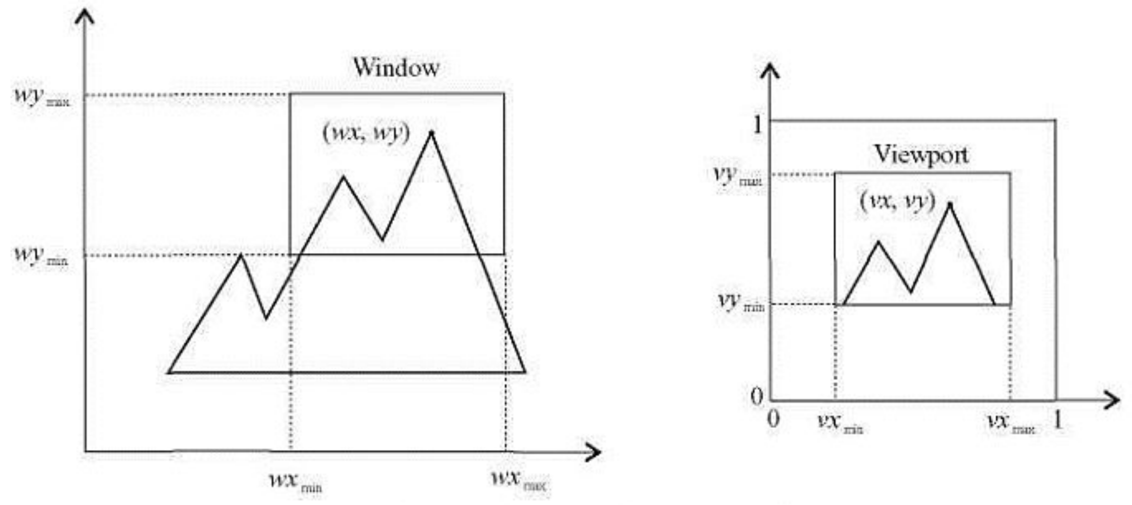


Fig. 5-2 Window-to-viewport mapping.

Clipping Operations

- Clipping
  - Any procedure that identifies those portions of a picture that are either inside or outside of a specified region of space
- Applied in World Coordinates
- Adapting Primitive Types
  - Point ✓
  - Line ✓
  - Area (or Polygons)
  - Curve, Text

Point Clipping

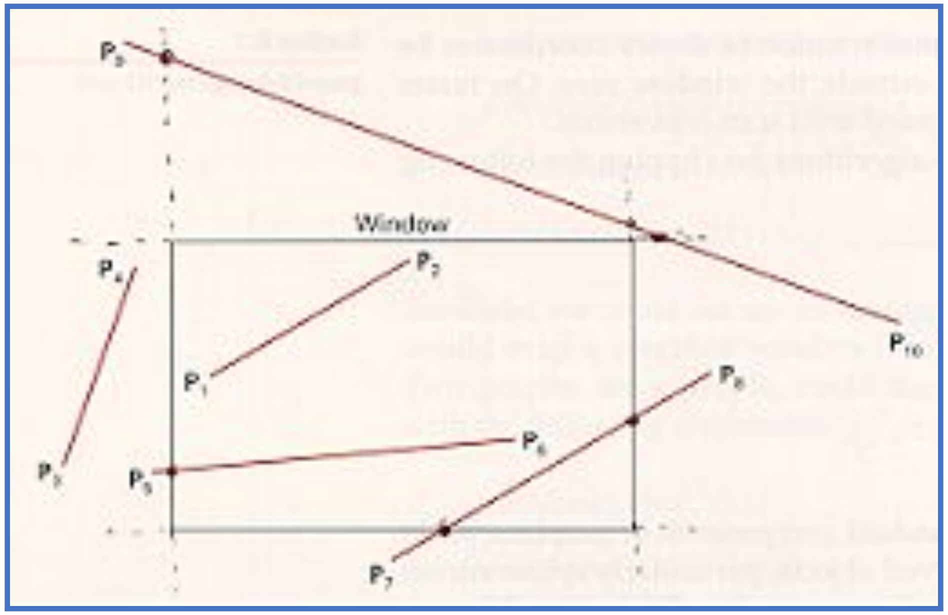
- Assuming that the clip window is a rectangle in standard position
- Point clipping is essentially the evaluation of the following inequalities:

$$x_{\min} \le x \le x_{\max} \quad \text{and} \quad y_{\min} \le y \le y_{\max}$$

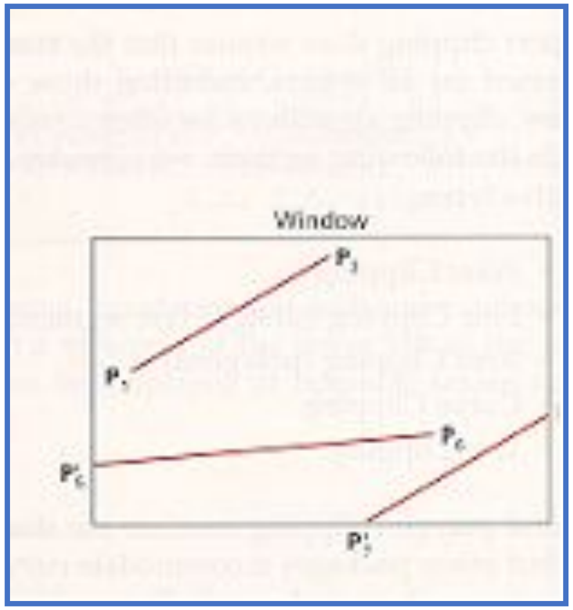
- where x_min, x_max, y_min and y_max define the clipping window.
- A point (x,y) is considered inside the window when the inequalities all evaluate to true.

Line Clipping

- Line clipping against a rectangular clip window



a) Before Clipping



b) After Clipping

Line Clipping

- Lines that do not intersect the clipping window are either completely inside the window or completely outside the window.
- On the other hand, a line that intersects the clipping window is divided by the intersection point(s) into segments that are either inside or outside the window.

Line Clipping

- Cohen-Sutherland Line Clipping
- Liang-Barsky Line Clipping
- NLN(Nicholl-Lee-Nicholl) Line Clipping
- Line Clipping Using Nonrectangular Clip Windows
- Splitting Concave Polygons

Cohen-Sutherland Line Clipping

- In this algorithm we divide the line clipping process into two phases: *(1)* identify those lines which intersect the clipping window and so need to be clipped and *(2)* perform the clipping.
- All lines fall into one of the following clipping categories:

1. Visible—both endpoints of the line lie within the window.

2. Not visible—the line definitely lies outside the window. This will occur if the line from (x1, y1) to (x2,y2) satisfies any one of the following four inequalities:

$$x_1, x_2 > x_{\max} \qquad y_1, y_2 > y_{\max}$$
$$x_1, x_2 < x_{\min} \qquad y_1, y_2 < y_{\min}$$

3. Clipping candidate—the line is in neither category 1 nor 2.

Cohen-Sutherland Line Clipping

- In Fig. 5-3, line AB is in category 1 (visible); lines CD and EF are in category 2 (not visible); and lines GH, IJ, and KL are in category 3 (clipping candidate).

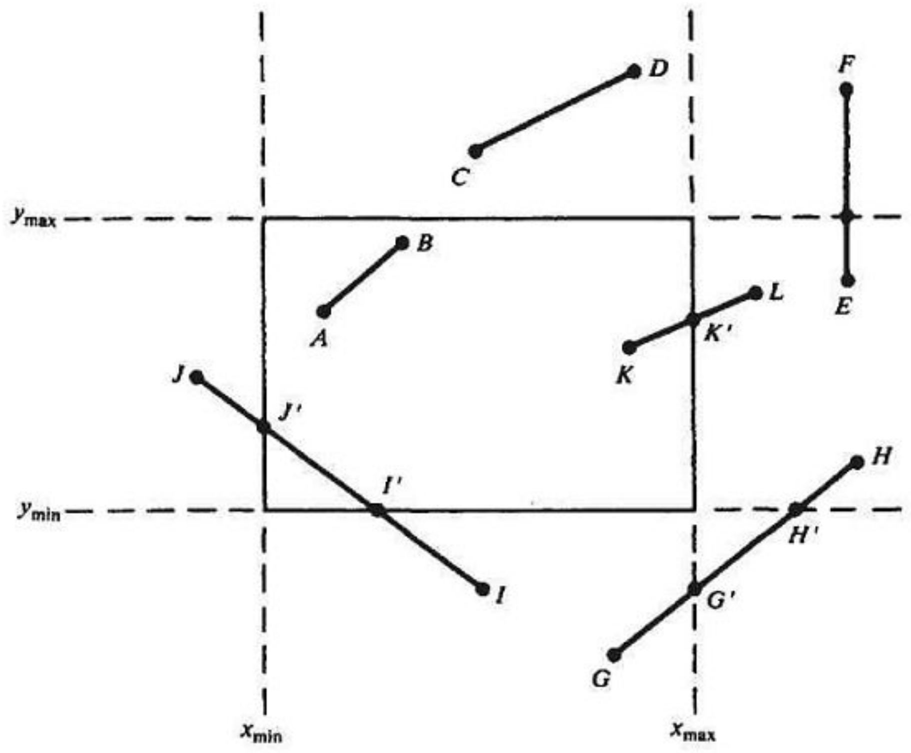
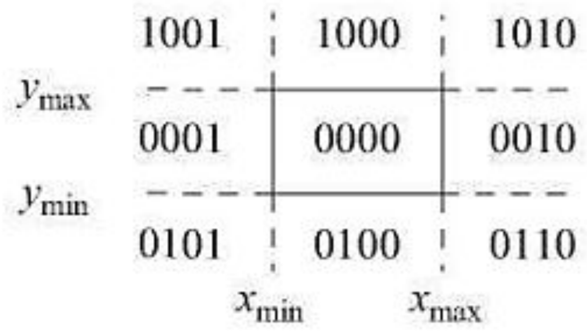


Fig. 5-3

Cohen-Sutherland Line Clipping

The algorithm employs an efficient procedure for finding the category of a line. It proceeds in two steps:

1. Assign a 4-bit region code to each endpoint of the line. The code is determined according to which of the following nine regions of the plane the endpoint lies in



Starting from the leftmost bit, each bit of the code is set to true (1) or false (0) according to the scheme

Bit 1 $\equiv$ endpoint is above the window $=$ sign $(y - y_{\max})$
Bit 2 $\equiv$ endpoint is below the window $=$ sign $(y_{\min} - y)$
Bit 3 $\equiv$ endpoint is to the right of the window $=$ sign $(x - x_{\max})$
Bit 4 $\equiv$ endpoint is to the left of the window $=$ sign $(x_{\min} - x)$

Cohen-Sutherland Line Clipping

2. The line is visible if both region codes are 0000, and not visible if the bitwise logical AND of the codes is not 0000, and a candidate for clipping if the bitwise logical AND of the region codes is 0000 (see Prob. 5.8).

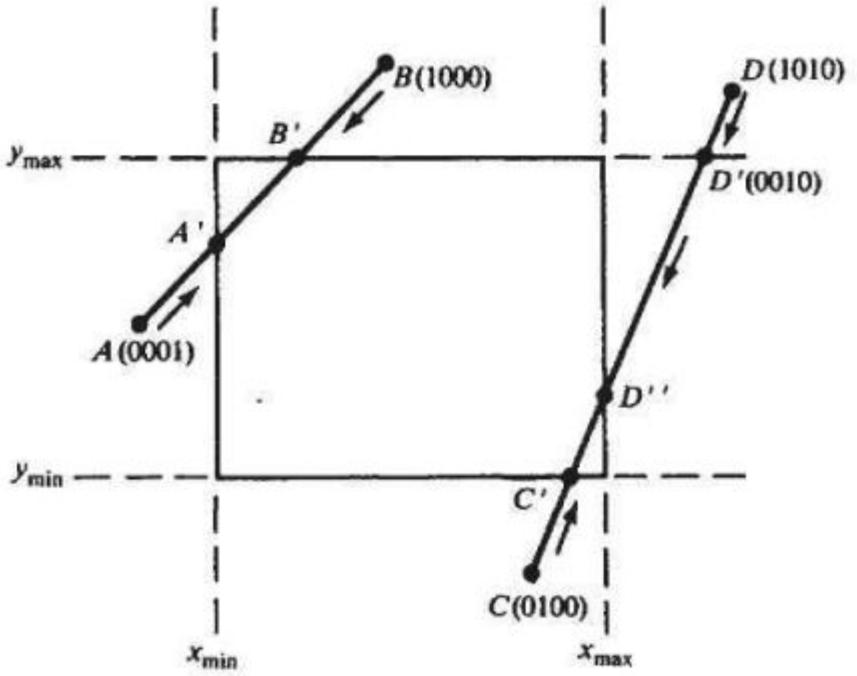
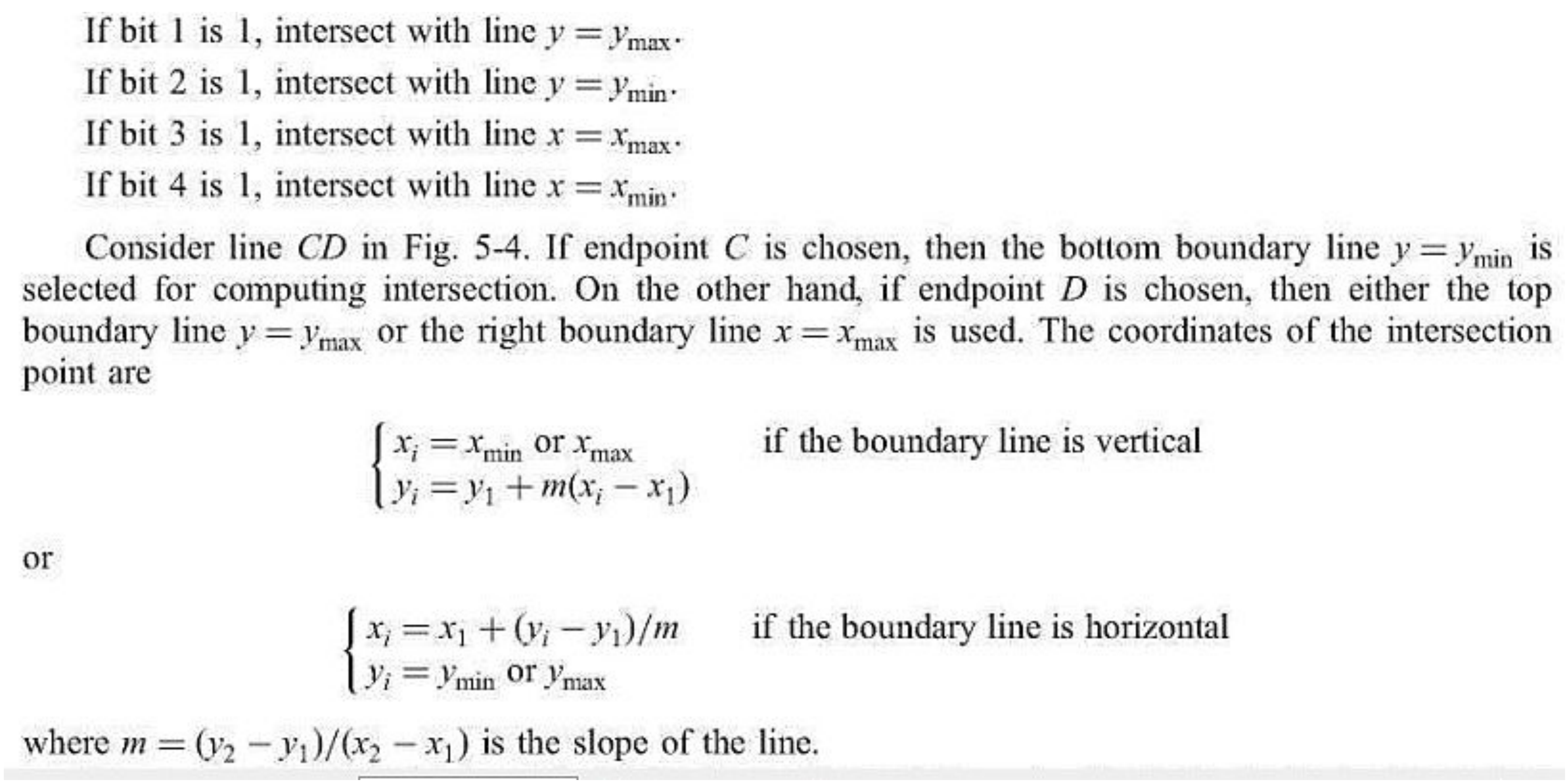


Fig. 5-4

Cohen-Sutherland Line Clipping



If bit 1 is 1, intersect with line $y = y_{\max}$.

If bit 2 is 1, intersect with line $y = y_{\min}$.

If bit 3 is 1, intersect with line $x = x_{\max}$.

If bit 4 is 1, intersect with line $x = x_{\min}$.

Consider line *CD* in Fig. 5-4. If endpoint *C* is chosen, then the bottom boundary line $y = y_{\min}$ is selected for computing intersection. On the other hand, if endpoint *D* is chosen, then either the top boundary line $y = y_{\max}$ or the right boundary line $x = x_{\max}$ is used. The coordinates of the intersection point are

$$\begin{cases} x_i = x_{\min} \text{ or } x_{\max} \\ y_i = y_1 + m(x_i - x_1) \end{cases} \qquad \text{if the boundary line is vertical}$$

or

$$\begin{cases} x_i = x_1 + (y_i - y_1)/m \\ y_i = y_{\min} \text{ or } y_{\max} \end{cases} \qquad \text{if the boundary line is horizontal}$$

where $m = (y_2 - y_1)/(x_2 - x_1)$ is the slope of the line.

Midpoint Subdivision

- An alternative way to process a line in category 3 is based on binary search.
- The line is divided at its midpoint into two shorter line segments.
- The clipping categories of the two new line segments are then determined by their region codes.
- Each segment in category 3 is divided again into shorter segments and categorized.
- This bisection and categorization process continues until each line segment that spans across a window boundary reaches a threshold for line size and all other segments are either in category 1 (visible) or in category 2 (invisible).
- The midpoint coordinates (xm,ym) of a line joining (x1, y1) and (x2,y2) are given by

$$x_m = \frac{x_1 + x_2}{2} \qquad y_m = \frac{y_1 + y_2}{2}$$

Midpoint Subdivision

- The example in Fig. 5-5 illustrates how midpoint subdivision is used to zoom in onto the two intersection points I_1 and I_2 with 10 bisections.
- The process continues until we reach two line segments that are, say, pixel-sized, i.e., mapped to one single pixel each in the image space.
- If the maximum number of pixels in a line is M, this method will yield a pixel-sized line segment in N subdivisions, where 2N = M or N = log2(M). For instance, when M = 1024 we need at most N = log2 1024 = 10 subdivisions.

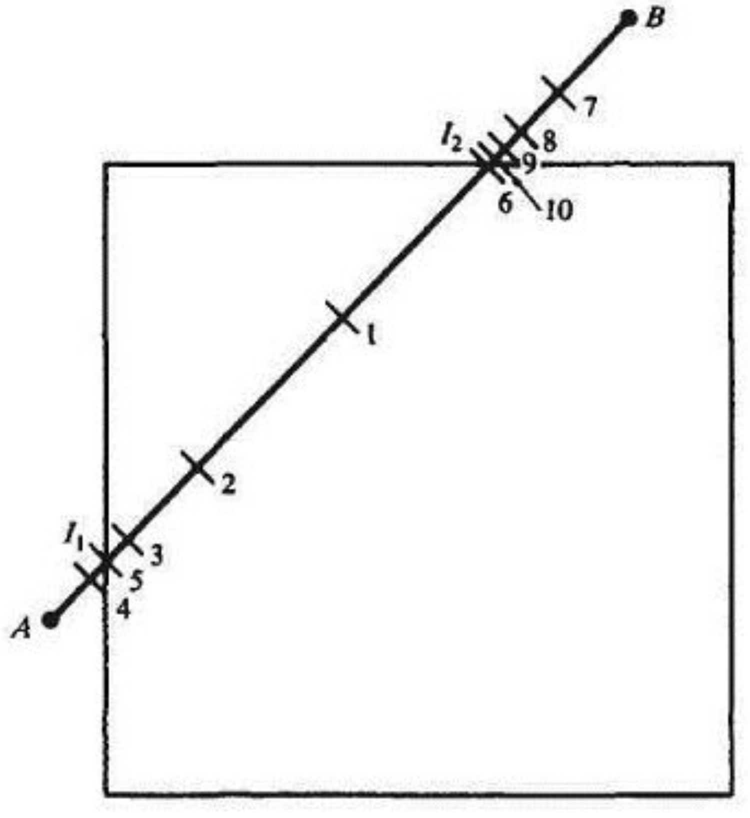


Fig. 5-5