COMPUTER GRAPHICS AND MULTIMEDIA - (18MCA41C)

UNIT-I

'Basics & Two-Dimensional Concepts'

FACULTY:

Dr. R. A. Roseline, M.Sc., M.Phil., Ph.D.,
Associate Professor and Head,
Post Graduate and Research Department of Computer Applications,
Government Arts College (Autonomous), Coimbatore – 641 018.

COMPUTER GRAPHICS AND MULTIMEDIA-(18MCA41C)

Syllabus :

- UNIT I: Basics & Two-Dimensional Concepts: Overview of Graphics Systems - Output primitives: Points and Lines - Line Drawing Algorithms - Circle generating Algorithms - Ellipse generating Algorithms - Two Dimensional Geometric Transformations: Basic Transformations, Matrix Representations, Composite Transformations - Two Dimensional Viewing: Line Clipping - Cohen-Sutherland algorithm.
- UNIT II: Three-Dimensional Concepts: Three Dimensional Concepts, Three Dimensional Object Representations: Polygon Surfaces - Curved Lines and Surfaces - Quadric Surfaces - Spline Representation - Three Dimensional Geometric and Modelling Transformations - Three Dimensional Viewing, Viewing Pipeline, Projections - Clipping. (Chapters 9, 10, 11 & 12).

- UNIT III: Visible Surface Detection Methods & Color Models: Visible Surface Detection Methods: Classification of Visible - Surface Detection Algorithms- Depth - Buffer Method, Scan line Method, BSPTree Method - Color Models and Color Applications.
- UNIT IV: Fundamentals Of Multimedia: Multimedia-Overview - digital representation - text - image - audio - video.
- UNIT V: Multimedia Architecture & Virtual Reality: Animation - compression - multimedia architecture - multimedia documents - multimedia application development - virtual reality.

- TEXT BOOKS: 1. Donald Hearn & M. Pauline Baker, “Computer Graphics - C version” Second Edition, Pearson Education, 2006. 2. Ranjan Parekh, “Principles of Multimedia”, Tata McGraw Hill Publications, 2008.
- REFERENCE BOOKS: 1. Foley James D., Vandam Andries and Hughes John F., "Computer Graphics : Principles and Practice", Pearson Education, 2013. 2. Ralf Steinmetz, Klara Steinmetz , “Multimedia Computing , Communications and Applications”, Pearson Education, 2012.

What is Computer Graphics? (

- Definition
  - Producing pictures or images using a computer
- Example
  - Starship Troopers: Tango-Urilla, Death From Above

What is Computer Graphics?

- Definition
  - Producing pictures or images using a computer
- Example
  - Starship Troopers: Tango-Urilla, Death From Above
    - On set: Pyrotechnics

Courtesy of Tippet Studio

What is Computer Graphics?

- Definition
  - Producing pictures or images using a computer
- Example
  - Starship Troopers: Tango-Urilla, Death From Above
    - Bugs:

Courtesy of Tippet Studio

What is Computer Graphics?

- Definition
  - Producing pictures or images using a computer
- Example
  - Starship Troopers: Tango-Urilla, Death From Above
    - The Final Stage: Burning the bugs



Courtesy of Tippet Studio

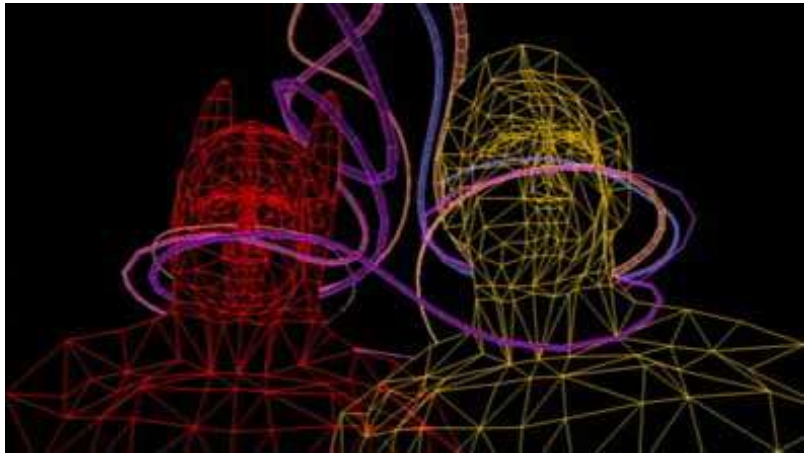
What is Computer Graphics?

- Definition
  - Producing pictures or images using a computer
- Example
  - Starship Troopers: Tango-Urilla, Death From Above
  - Batman & Robin: The Love Dust

Courtesy of Buf Compagnie

What is Computer Graphics?

- Definition
  - Producing pictures or images using a computer
- Example
  - Starship Troopers: Tango-Urilla, Death From Above
  - Batman & Robin: The Love Dust



What is Computer Graphics?

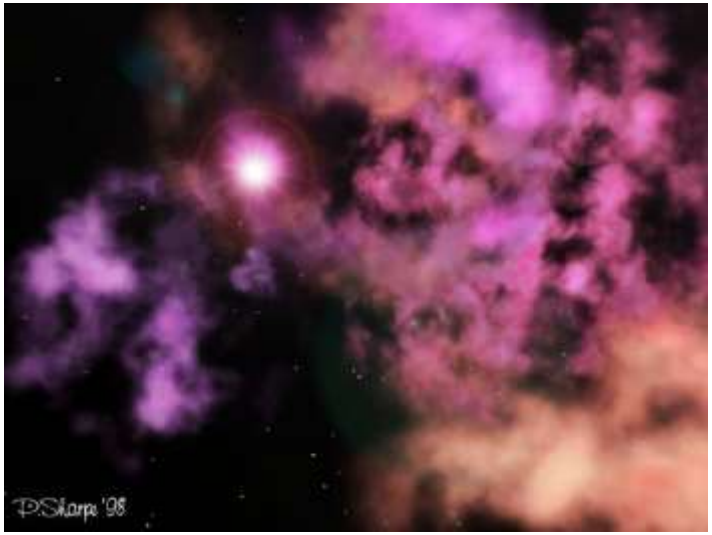
- Imaging
  - Representing 2D images
- Modeling
  - Representing 3D objects
- Rendering
  - Constructing 2D images from 3D models
- Animation
  - Simulating changes over time

Applications

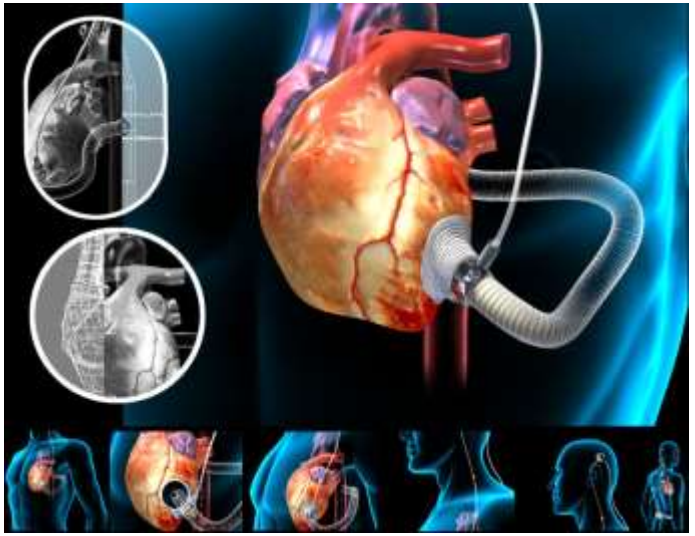
- Display of Information
- Design
- Simulation
- Computer Art
- Entertainment

Display of Information

- Graphics for Scientific, Engineering, and Medical Data



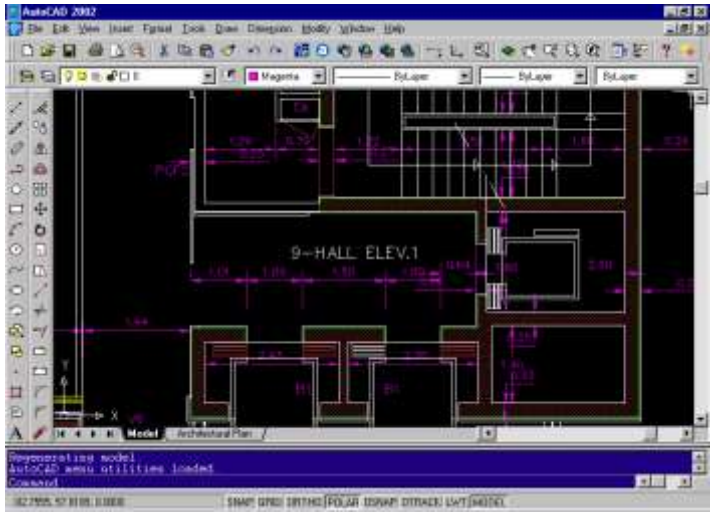
Nebula



Medical Image

Design

- Graphics for Engineering and Architectural System
- Design of Building, Automobile, Aircraft, Machine etc.



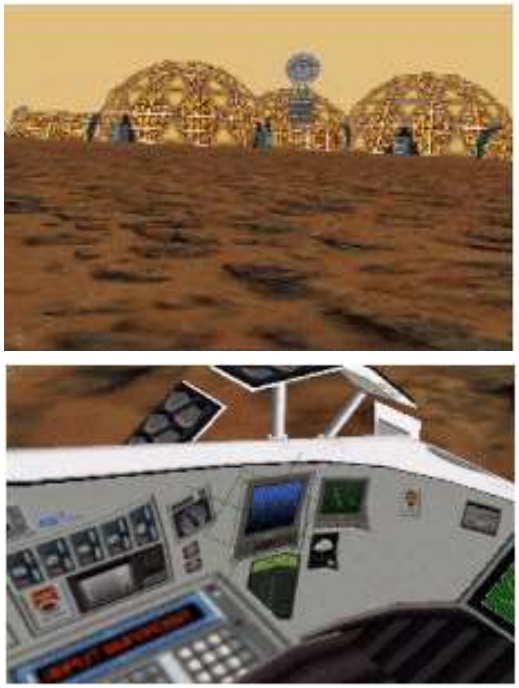
AutoCAD 2002

Interior Design

Simulation

- Computer-Generated Models of Physical, Financial and Economic Systems for Educational Aids

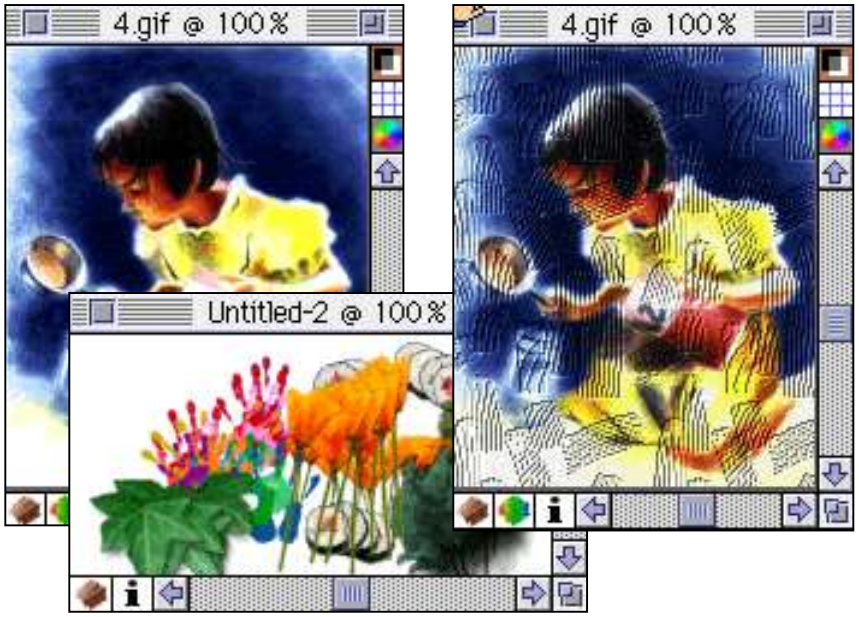
Flight Simulator



Mars Rover Simulator

Computer Art

- Graphics for Artist



Metacreation Painter

Entertainment

- Graphics for Movie, Game, VR etc.

Final Fantasy

Online Game

Contents

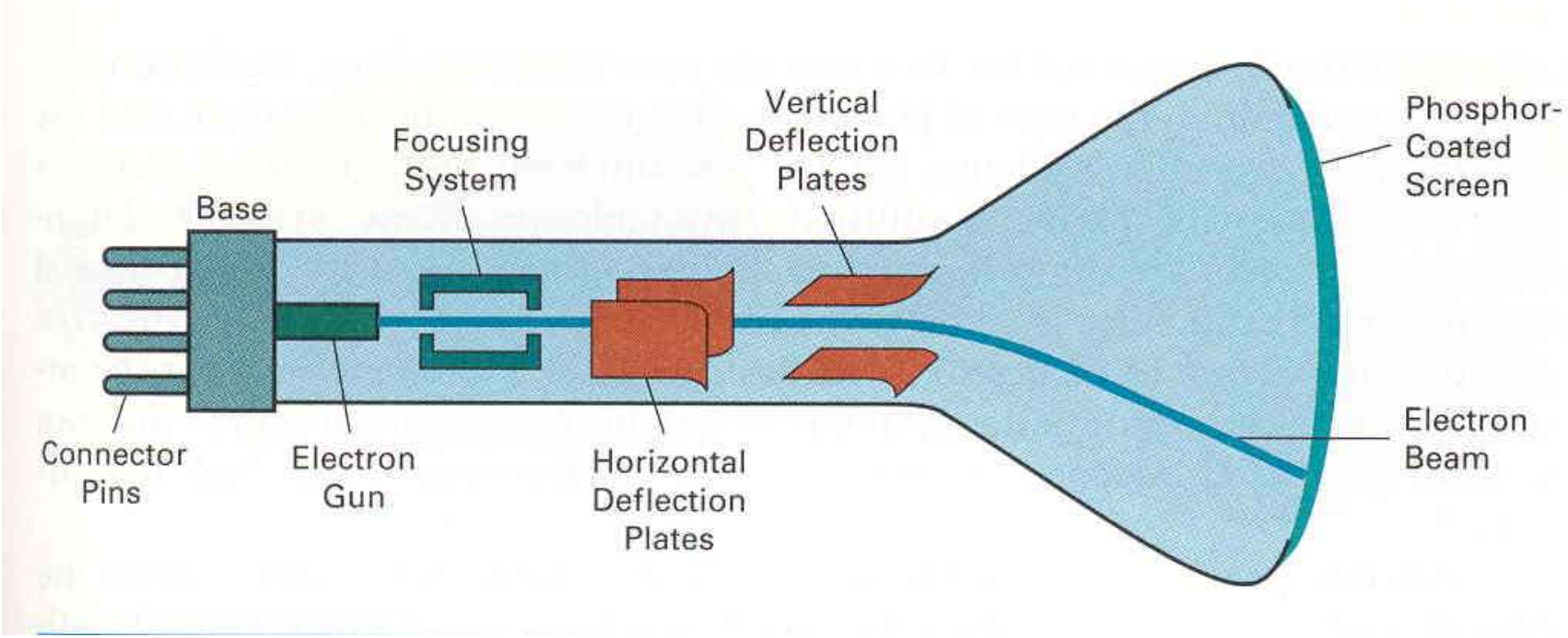
- Display Hardware
  - How are images display?
- Raster Graphics Systems
  - How are imaging system organized?
- Output Primitives
  - How can we describe shapes with primitives?
- Color Models
  - How can we describe and represent colors?

Display Hardware

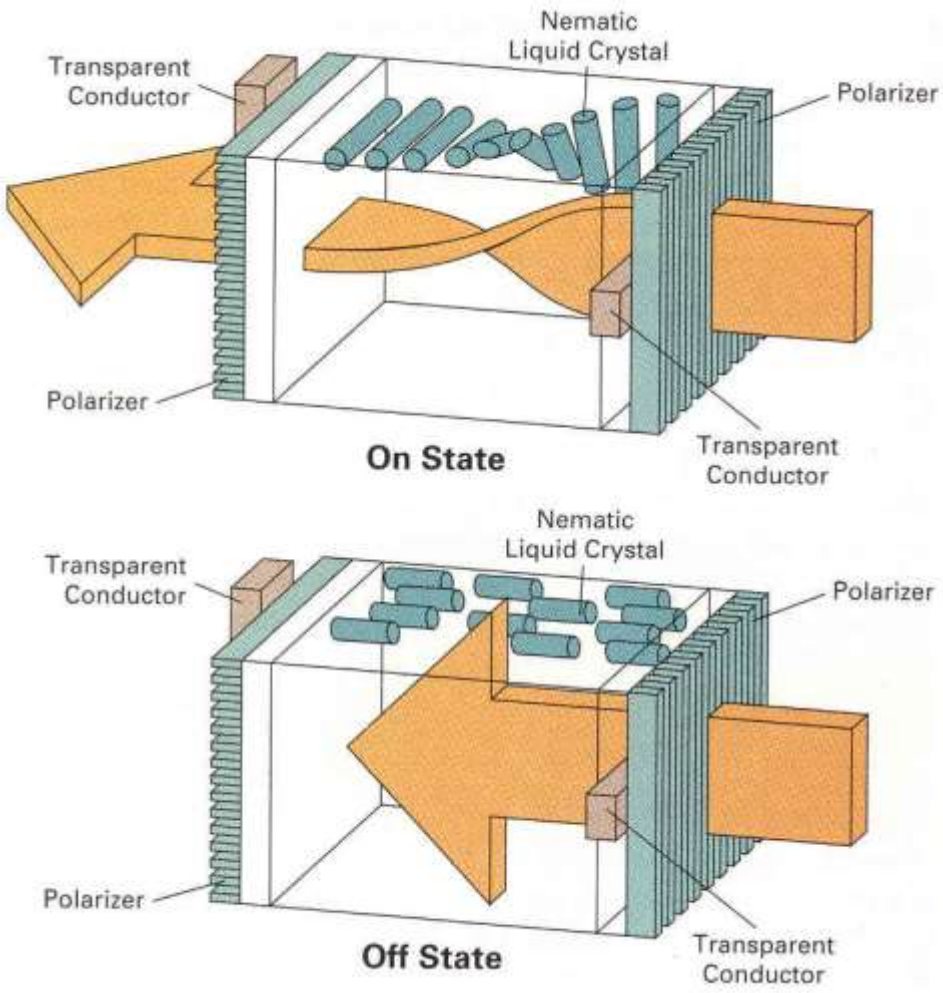
- Video Display Devices
  - Cathode Ray Tube (CRT)
  - Liquid Crystal Display (LCD)
  - Plasma panels
  - Thin-film electroluminescent display
  - Light-emitting diodes (LED)

- Hard-Copy Devices
  - Ink-jet printer
  - Laser printer
  - Film recorder
  - Electrostatic printer
  - Pen plotter

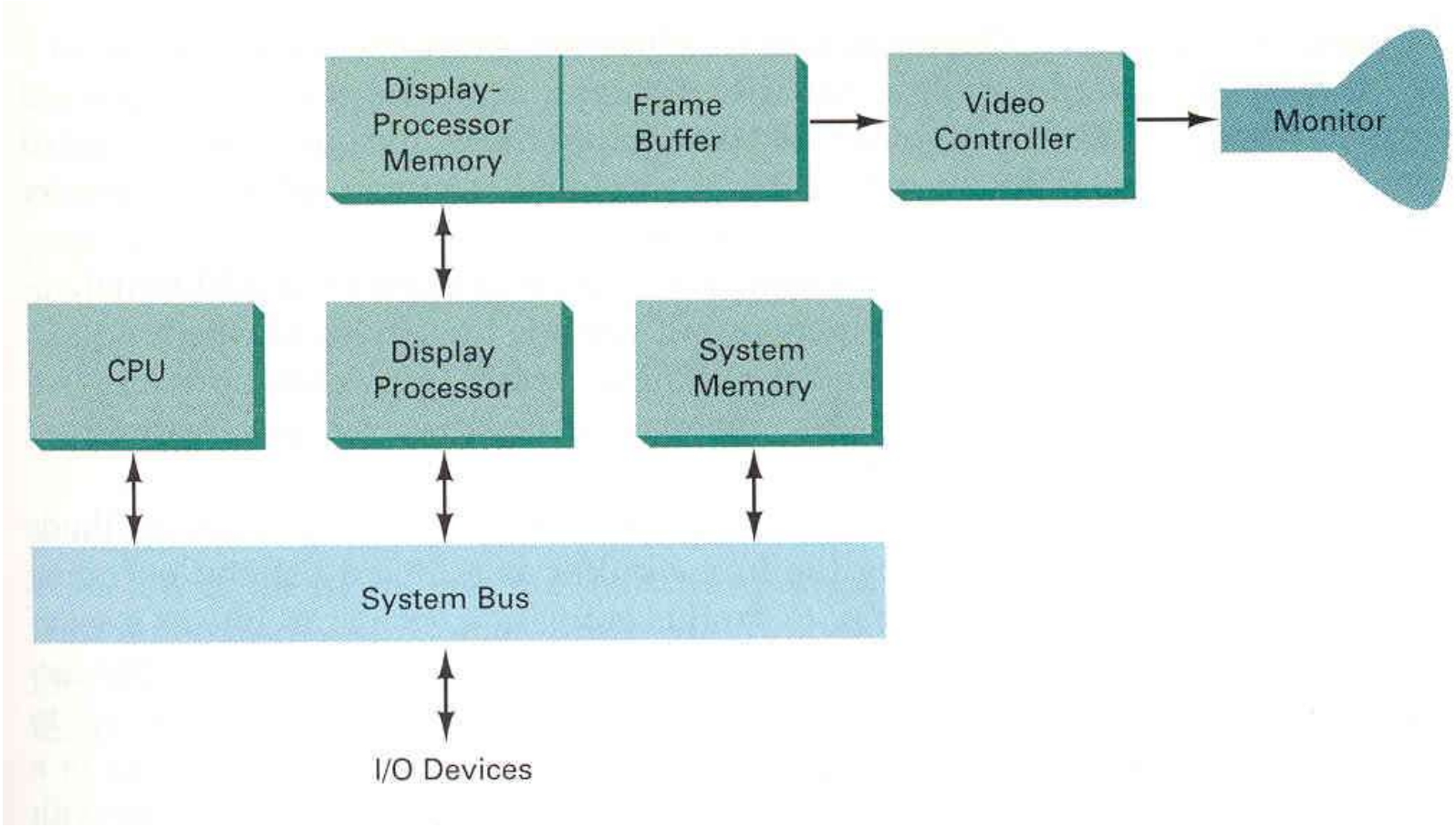
Cathode Ray Tube (CRT)



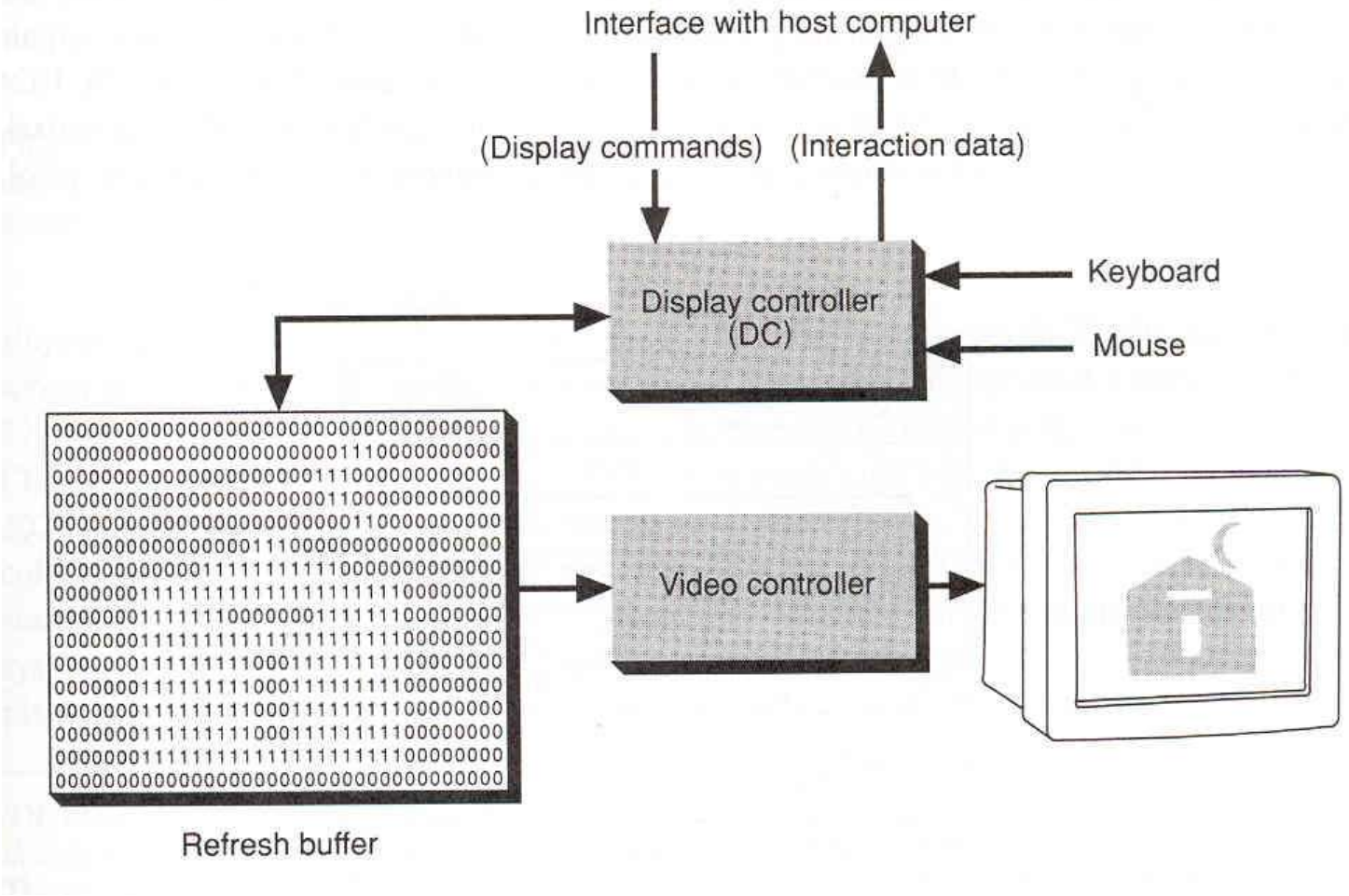
Liquid Crystal Display (LCD)



Raster Graphics

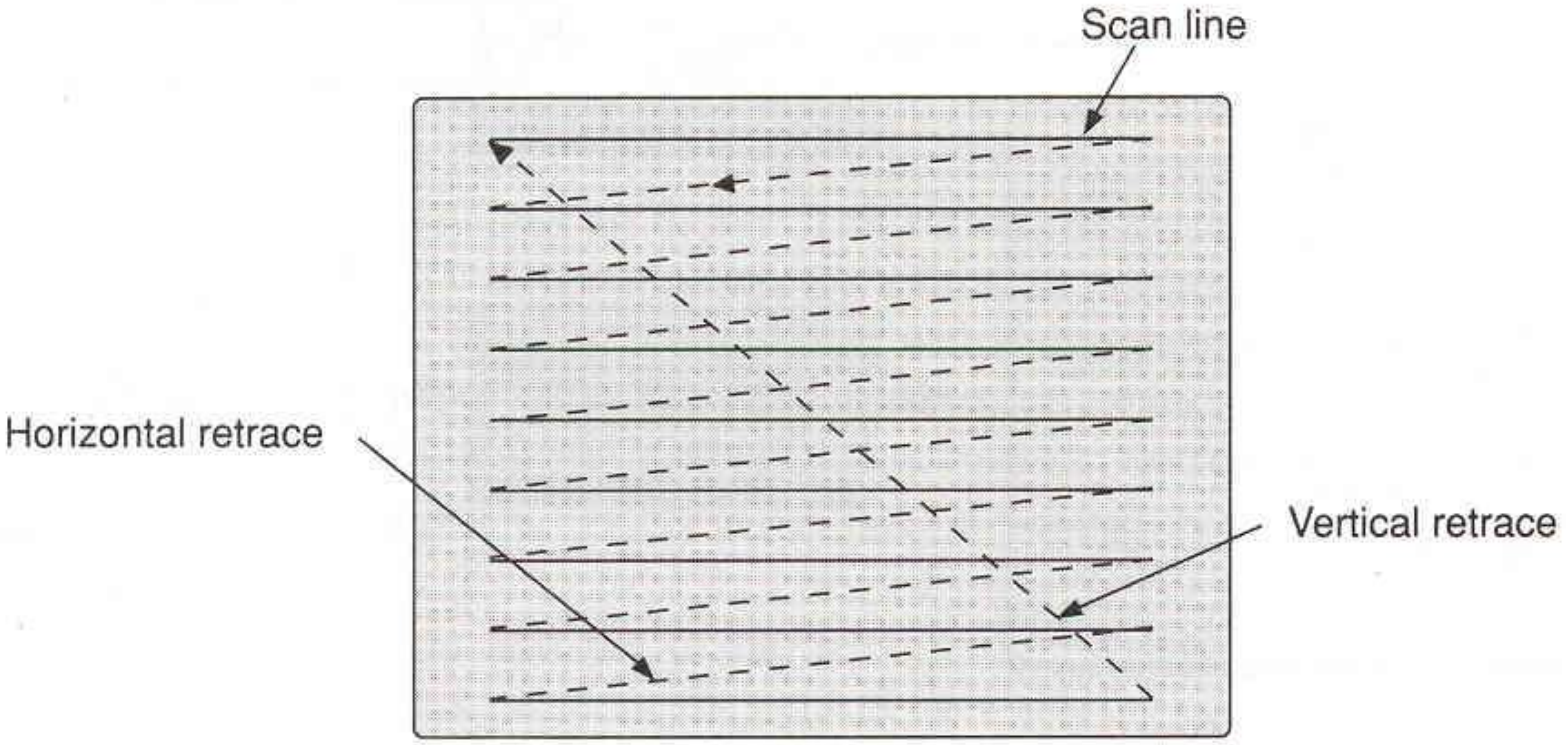


Frame Buffer

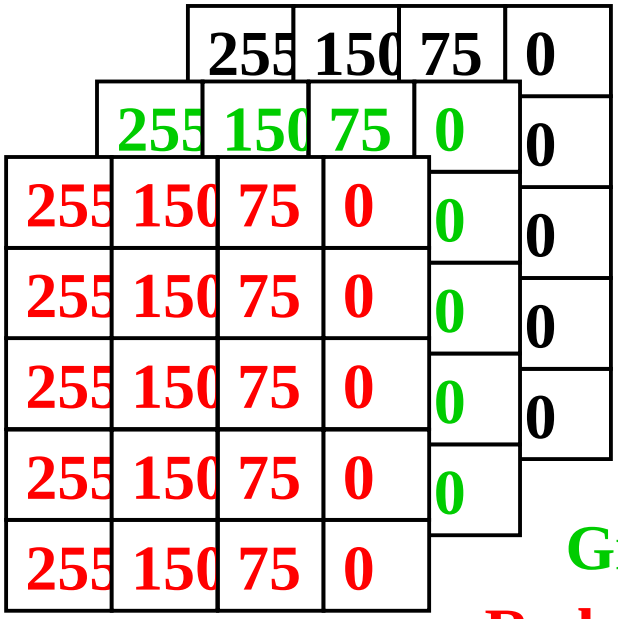


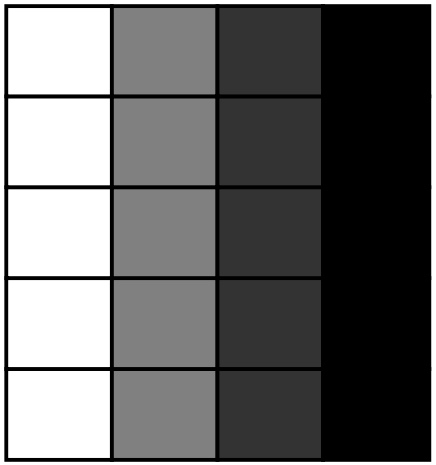
Frame Buffer Refresh

- Refresh Rate
  - Usually 30~75 Hz

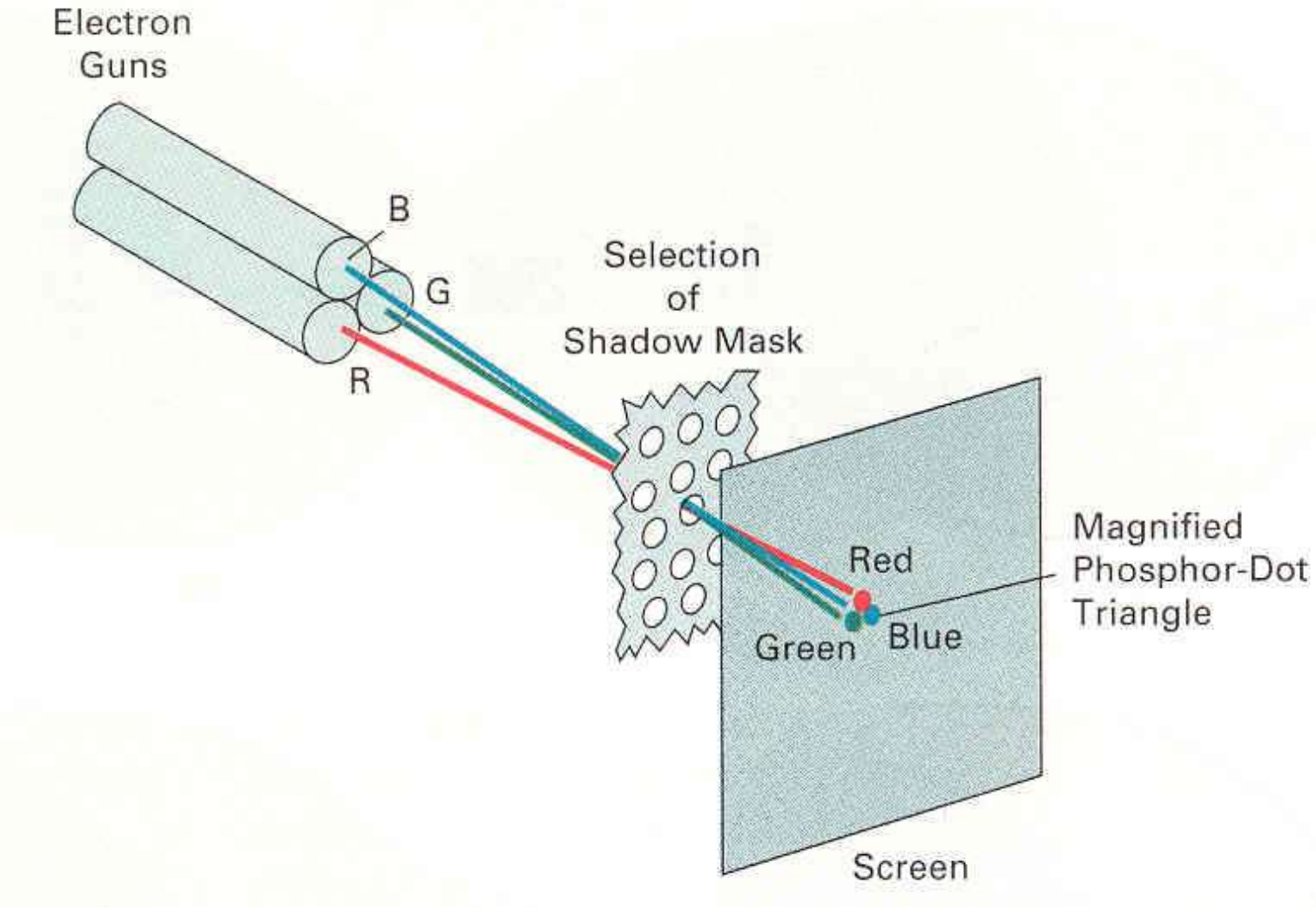


Color Frame Buffer





Color CRT


Electron
Guns
B
G
R
Selection
of
Shadow Mask
Red
Green
Blue
Magnified
Phosphor-Dot
Triangle
Screen

Introduction

- For a raster display, a picture is completely specified by:
  - intensity and position of pixels, or/and
  - set of complex objects

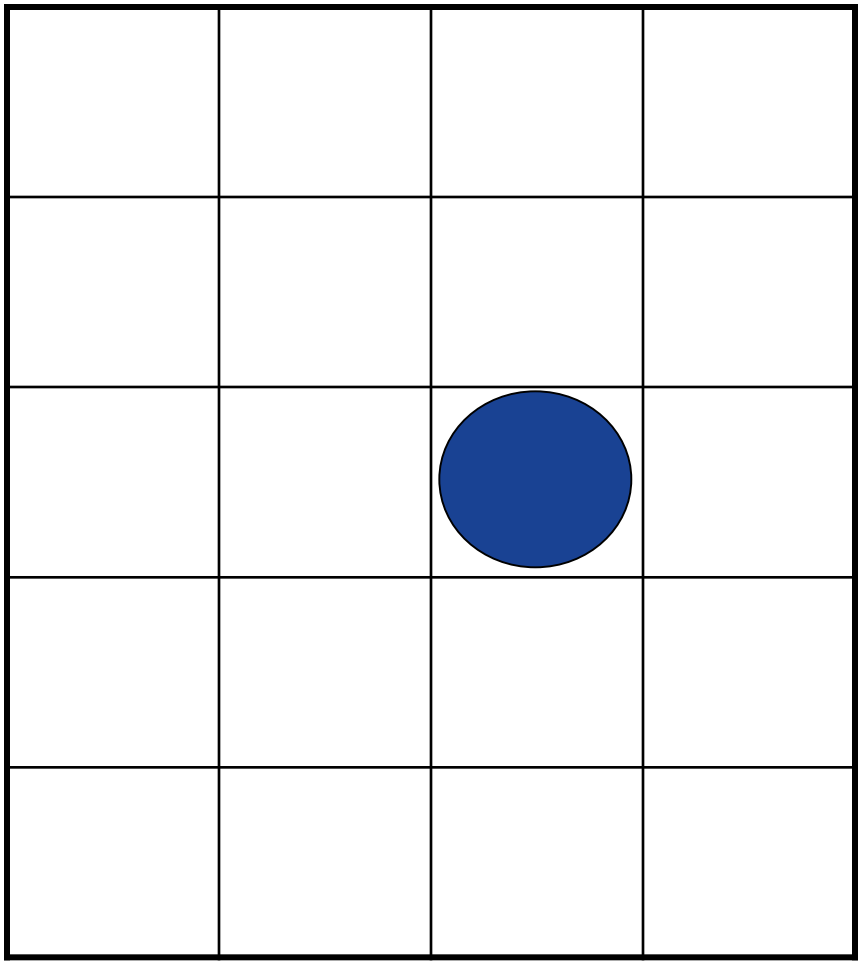
- Shapes and contours can either be stored in terms of pixel patterns (bitmaps) or as a set of basic geometric structures (for example, line segments).

Introduction

- Output primitives are the basic geometric structures which facilitate or describe a scene/picture. Example of these include:
  - *points, lines, curves (circles, conics etc), surfaces, fill colour, character string etc.*

Points

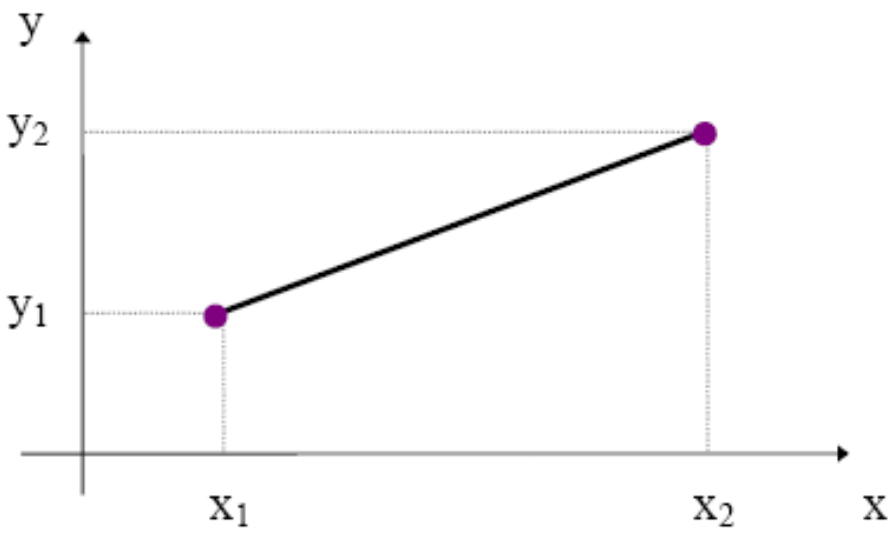
- A point is shown by illuminating a pixel on the screen



Lines

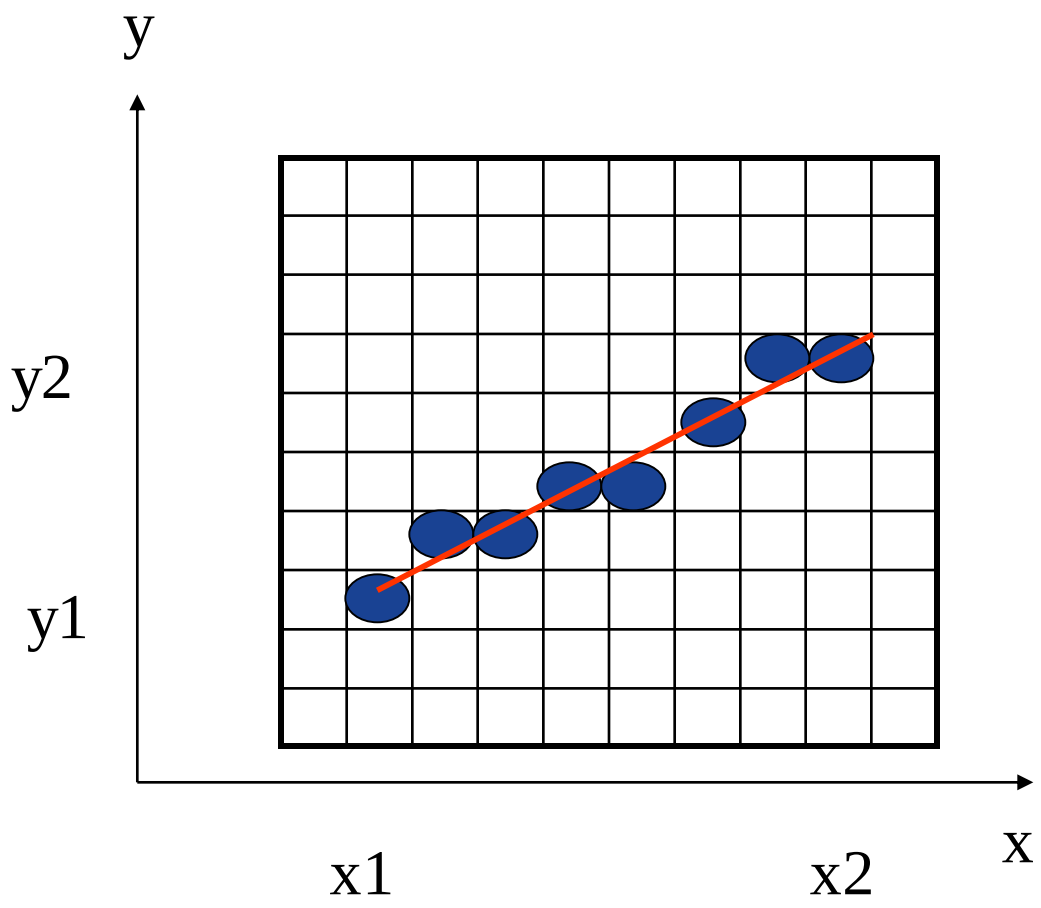
- A line segment is completely defined in terms of its two endpoints.
- A line segment is thus defined as:

  Line_Seg = { $(x_1, y_1), (x_2, y_2)$ }



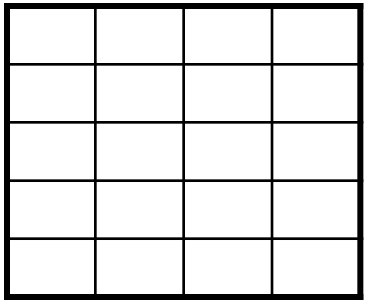
Lines

- A line is produced by means of illuminating a set of intermediary pixels between the two endpoints.



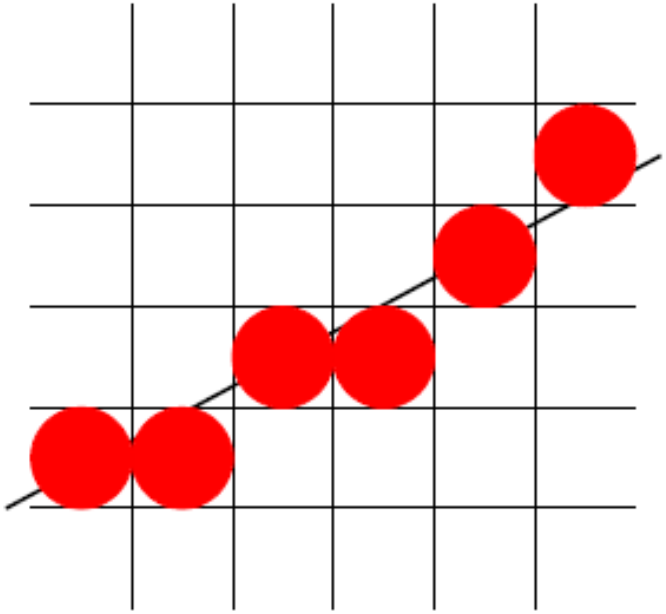
Lines

- Lines is digitized into a set of discrete integer positions that approximate the actual line path.
- Example: A computed line position of (10.48, 20.51) is converted to pixel position (10, 21).



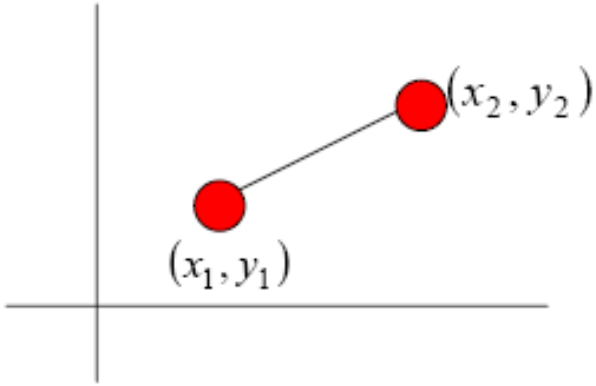
Line

- The rounding of coordinate values to integer causes all but horizonatal and vertical lines to be displayed with a stair step appearance “the jaggies”.



Line Drawing Algorithms

- A straight line segment is defined by the coordinate position for the end points of the segment.
- Given Points (x_1, y_1) and (x_2, y_2)



Line

- All line drawing algorithms make use of the fundamental equations:

- Line Eqn. $y = m.x + b$
- Slope $m = y_2 - y_1 / x_2 - x_1 = \Delta y / \Delta x$
- y-intercept $b = y_1 - m.x_1$
- x-interval$\rightarrow \Delta x = \Delta y / m$
- y-interval$\rightarrow \Delta y = m\, \Delta x$

DDA Algorithm (Digital Differential Analyzer)

- A line algorithm Based on calculating either Δy or Δx *using the above equations.*
- *There are two cases:*
  - *Positive slop*
  - *Negative slop*

DDA- Line with positive Slope

If $m \leq 1$ then take $\Delta x = 1$

- Compute successive y by

$$y_{k+1} = y_k + m \qquad (1)$$

- Subscript k takes integer values starting from 1, for the first point, and increases by 1 until the final end point is reached.
- Since $0.0 < m \leq 1.0$, the calculated y values must be rounded to the nearest integer pixel position.

DDA

- If $m > 1$, reverse the role of x and y and take $\Delta y = 1$, calculate successive x from

$$x_{k+1} = x_k + 1/m \qquad (2)$$

- In this case, each computed x value is rounded to the nearest integer pixel position.
- The above equations are based on the assumption that lines are to be processed from left endpoint to right endpoint.

DDA

- In case the line is processed from Right endpoint to Left endpoint, then

$$\Delta x = -1,\ y_{k+1} = y_k - m \quad \text{for } m \le 1 \qquad (3)$$

or

$$\Delta y = -1,\ x_{k+1} = x_k - 1/m \quad \text{for } m > 1 \qquad (4)$$

DDA- Line with negative Slope

- If $m < 1$,
  - use(1) [provided line is calculated from left to right] and
  - use(3) [provided line is calculated from right to left].
- If $m \geq 1$
  - use (2) or (4).

Merits + Demerits

- Faster than the direct use of line Eqn.
- It eliminates the multiplication in line Eqn.
- For long line segments, the true line Path may be mislead due to round off.
- Rounding operations and floating-point arithmetic are still time consuming.
- The algorithm can still be improved.
- Other algorithms, with better performance also exist.

Code for DDA Algorithm

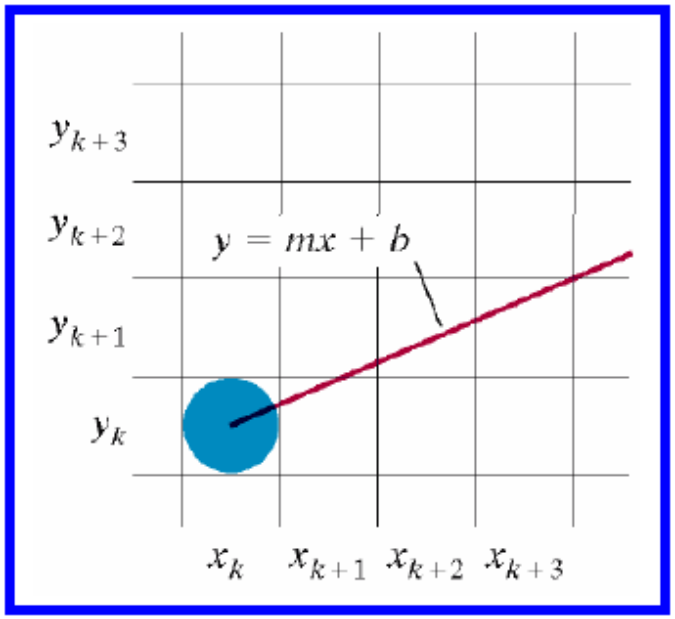
```
Procedure lineDDA(xa,ya,xb,yb:integer);
Var
    dx,dy,steps,k:integer
    xIncrement,yIncrement,x,y:real;
begin
    dx:=xb-xa;
    dy:=yb-ya;
    if abs(dx)>abs(dy) then steps:=abs(dx)
    else steps:=abs(dy);
    xIncrement:=dx/steps;
    yIncrement:=dy/steps;
    x:=xa;
    y:=ya;
    setPixel(round(x),round(y),1);
    for k:=1 to steps do
        begin
            x:=x+xIncrement;
            y:=y+yIncrement;
            setPixel(round(x),round(y),1)
        end
end; {lineDDA}
```

Bresenham's Line Algorithm

- It is an efficient raster line generation algorithm.
- It can be adapted to display circles and other curves.
- The algorithm
  - After plotting a pixel position (x_k, y_k) , what is the next pixel to plot?
- Consider lines with positive slope.

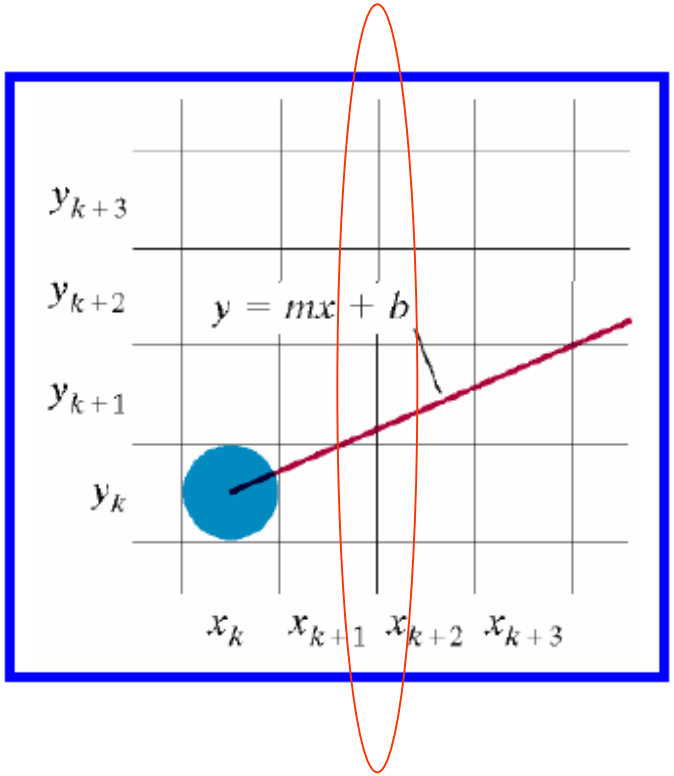
Bresenham's Line

- For a positive slope, 0 < m < 1 and line is starting from left to right.
- After plotting a pixel position (x_k, y_k) we have two choices for next pixel:
  - (x_k +1, y_k)
  - (x_k +1, y_k+1)



Bresenham's Line

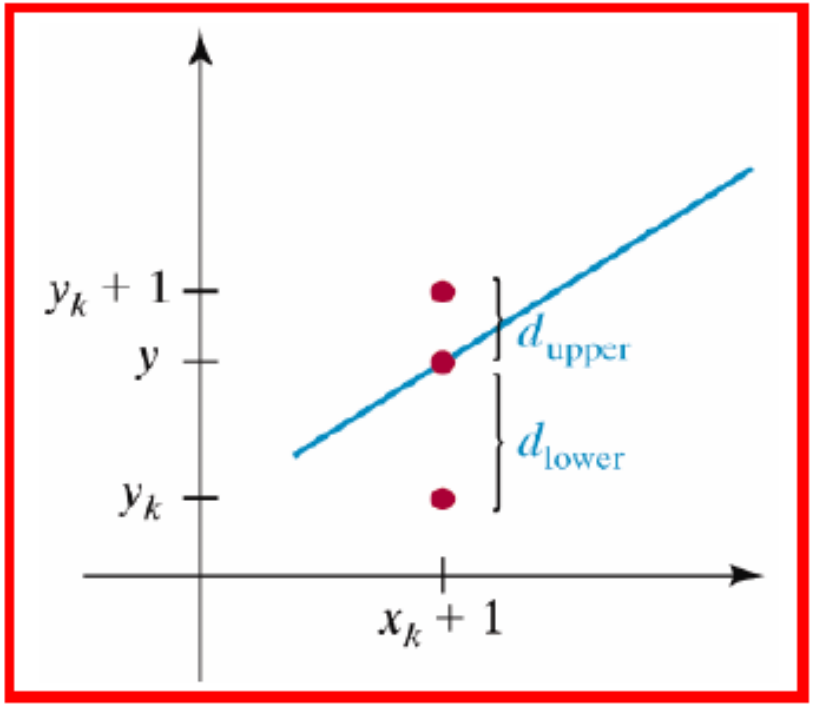
- At position x_k +1, we pay attention to the intersection of the vertical pixel and the mathematical line path.



Bresenham's Line

- At position x_k +1, we label vertical pixel separations from the mathematical line path as

  d_{lower} , d_{upper}.



Bresenham's Line

- The y coordinate on the mathematical line at x_k+1 is calculated as

$$y = m(x_k + 1) + b$$

then

$$d_{lower} = y - y_k$$
$$= m(x_k + 1) + b - y_k$$

and

$$d_{upper} = (y_k + 1) - y$$
$$= y_k + 1 - m(x_k + 1) - b$$

Bresenham's Line

- To determine which of the two pixels is closest to the line path, we set an efficient test based on the difference between the two pixel separations

$$d_{lower} - d_{upper} = 2m(x_k + 1) - 2y_k + 2b - 1$$
$$= 2(\Delta y / \Delta x)(x_k + 1) - 2y_k + 2b - 1$$

- Consider a decision parameter p_k such that

$$p_k = \Delta x (d_{lower} - d_{upper})$$
$$= 2\Delta y.x_k - 2\Delta x.y_k + c$$

- where

$$c = 2\Delta y + \Delta x(2b - 1)$$

Bresenham's Line

- Since $\Delta x > 0$, Comparing (d_{lower} and d_{upper}), would tell which pixel is closer to the line path; is it y_k or $y_k + 1$

- If ($d_{lower} < d_{upper}$)
  - Then p_k is negative
    - Hence plot lower pixel.
  - Otherwise
    - Plot the upper pixel.

Bresenham's Line

- We can obtain the values of successive decision parameter as follows:

$$p_k = 2\Delta y.x_k - 2\Delta x.y_k + c$$

$$p_{k+1} = 2\Delta y.x_{k+1} - 2\Delta x.y_{k+1} + c$$

- Subtracting these two equations

$$p_{k+1} - p_k = 2\Delta y\,(x_{k+1} - x_k) - 2\Delta x\,(y_{k+1} - y_k)$$

- But $x_{k+1} - x_k = 1$, Therefore

$$p_{k+1} = p_k + 2\Delta y - 2\Delta x\,(y_{k+1} - y_k)$$

Bresenham's Line

- ($y_{k+1} - y_k$) is either 0 or 1, depending on the sign of p_k (plotting lower or upper pixel).
- The recursive calculation of p_k is performed at integer x position, starting at the left endpoint.
- p_0 can be evaluated as:

$$p_0 = 2\Delta y - \Delta x$$

Bresenham's Line-Drawing Algorithm for m < 1

1. Input the two line end points and store the left end point in (x_0 , y_0).
2. Load (x_0 , y_0) into the frame buffer; that is, plot the first point.
3. Calculate the constants Δx, Δy, $2\Delta y$, and $2\Delta y - 2\Delta x$, and obtain the starting value for the decision parameter as

   $$p_0 = 2\Delta y - \Delta x$$
4. At each x_k along the line, starting at $k = 0$, perform the following test: If $p_k < 0$,the next point to plot is $(x_{k+1}, y_k$ and

   $$p_{k+1}=p_k+2\Delta y$$

   Otherwise, the next point to plot is (x_{k+1}, y_{k+1}) and

   $$p_{k+1}=p_k+2\Delta y-2\Delta x$$

5. Repeat step 4, $\Delta x-1$ times.

Summary

- The constants $2\Delta y$ and $2\Delta y - 2\Delta x$ are calculated once for each line to be scan converted.
- Hence the arithmetic involves only integer addition and subtraction of these two constants.

Example

- To illustrate the algorithm, we digitize the line with endpoints (20,10) and (30,18). This line has slope of 0.8, with

  $\Delta x = 10$

  $\Delta y = 8$

- The initial decision parameter has the value

  $p_0 = 2\Delta y - \Delta x = 6$

- and the increments for calculating successive decision parameters are

  $2\,\Delta y = 16$

  $2\,\Delta y - 2\,\Delta x = -4$

Example

- We plot the initial point (x_0 , y_0)=(20,10) and determine successive pixel positions along the line path from the decision parameter as

K	p_k	(x_{k+1}, y_{k+1})	K	p_k	(x_{k+1}, y_{k+1})
0	6	(21,11)	5	6	(26,15)
1	2	(22,12)	6	2	(27,16)
2	-2	(23,12)	7	-2	(28,16)
3	14	(24,13)	8	14	(29,17)
4	10	(25,14)	9	10	(30,18)

Example

A plot of the pixels generated along this line path is shown in Fig.

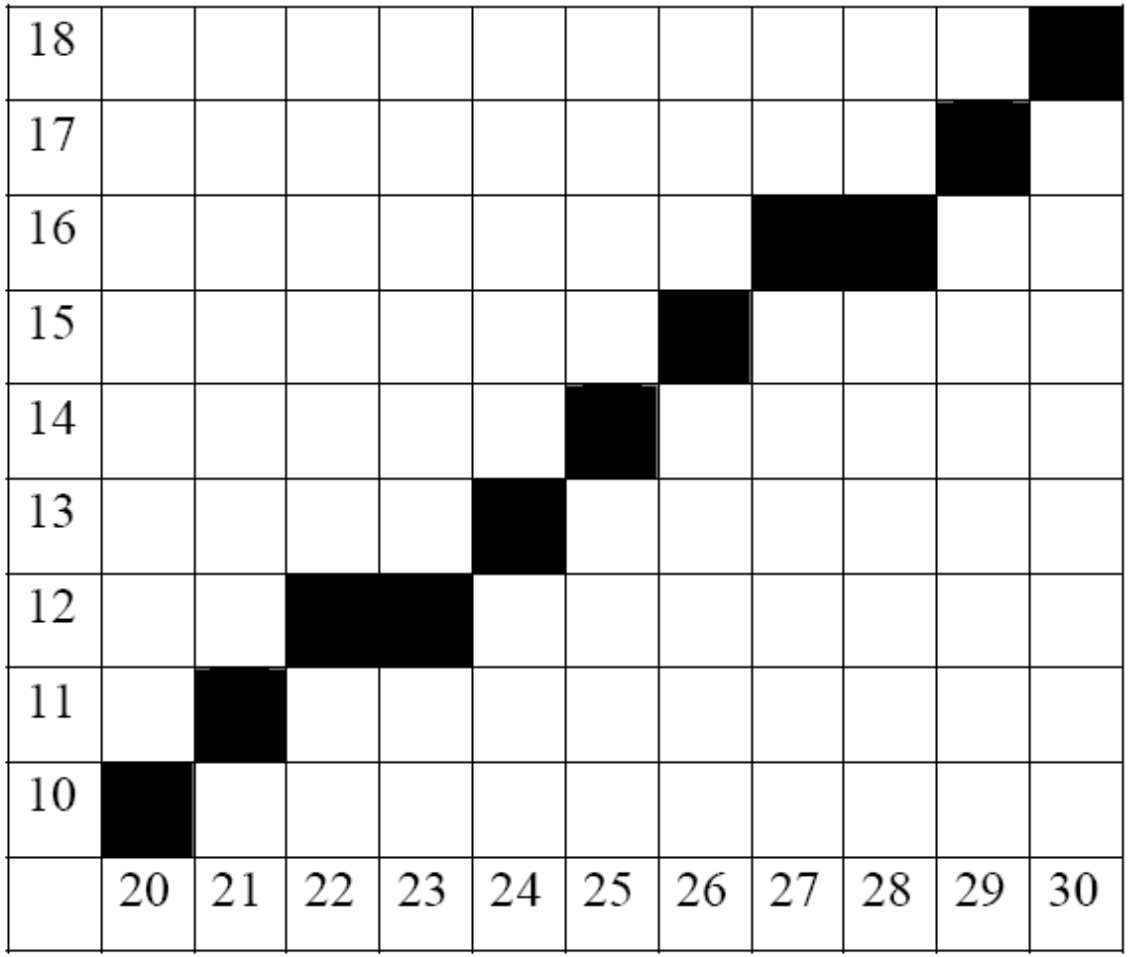
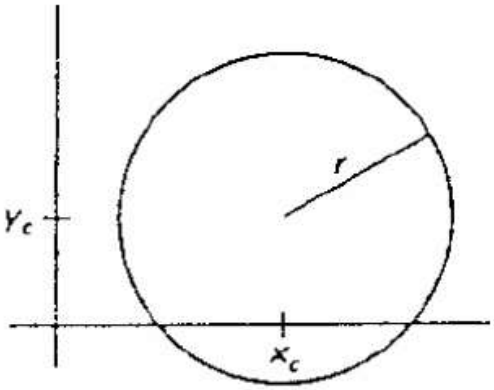


Figure: The Bresenham line from point (20,10) to point (30,18)

Circle Generating Algorithms

- A circle is defined as the set of points that are all at a given distance r from a center point (x_c, y_c).



- For any circle point (x, y), this distance is expressed

$$(x - x_c)^2 + (y - y_c)^2 = r^2$$

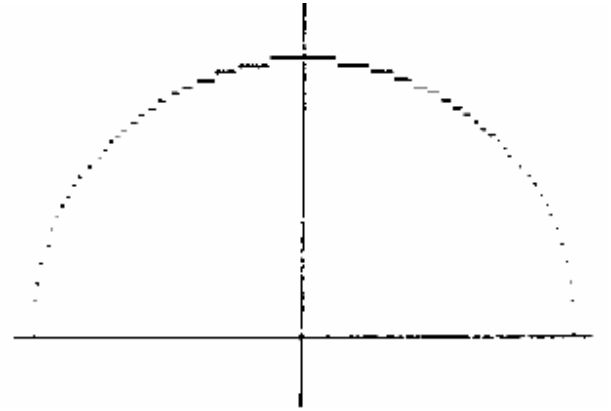
- We calculate the points by stepping along the x-axis in unit steps from x_c-r to x_c+r and calculate y values as

$$y = y_c \pm \sqrt{r^2 - (x_c - x)^2}$$

Circle Generating Algorithms

- **There are some problems with this approach:**

1. Considerable computation at each step.
2. Non-uniform spacing between plotted pixels as in this Figure.



Circle Generating Algorithms

- Problem 2 can be removed using the polar form:

$$x = x_c + r \cos \theta$$

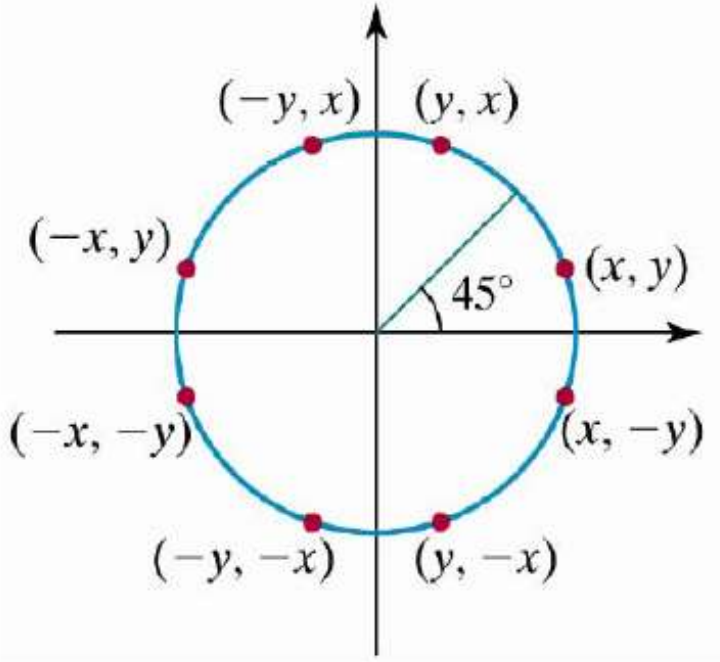
$$y = y_c + r \sin \theta$$

- using a fixed angular step size, a circle is plotted with equally spaced points along the circumference.

Circle Generating Algorithms

- Problem 1 can be overcome by considering the symmetry of circles as in Figure 3.
- But it still requires a good deal of computation time.

- **Efficient Solutions**
  - Midpoint Circle Algorithm



Mid point Circle Algorithm

- To apply the midpoint method, we define a circle function:

$$f_{circle}(x,y) = x^2 + y^2 - r^2 = 0 \qquad (2)$$

- Any point (x,y) on the boundary of the circle with radius r satisfies the equation $f_{circle}(x, y)= 0$.

Mid point Circle Algorithm

- If the points is in the interior of the circle, the circle function is negative.
- If the point is outside the circle, the circle function is positive.

- To summarize, the relative position of any point (x,y) can be determined by checking the sign of the circle function:

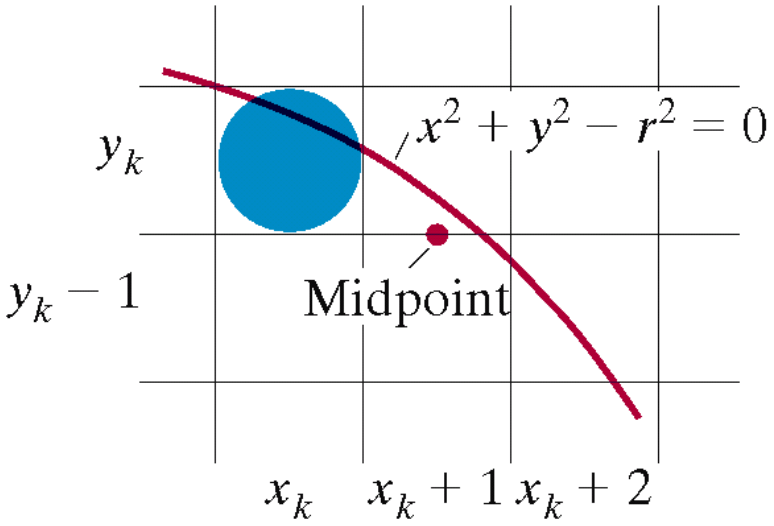
Mid point Circle Algorithm

$$f_{circle}(x,y) \begin{cases} < 0 & \text{if } (x,y) \text{ is inside the circle boundary} \\ = 0 & \text{if } (x,y) \text{ is on the circle boundary} \\ > 0 & \text{if } (x,y) \text{ is outside the circle boundary} \end{cases} \quad (3)$$

- The circle function tests in (3) are performed for the mid positions between pixels near the circle path at each sampling step. Thus, the circle function is the decision parameter in the midpoint algorithm, and we can set up incremental calculations for this function as we did in the line algorithm.

Mid point Circle Algorithm

- Figure 4 shows the midpoint between the two candidate pixels at sampling position x_k +1. Assuming we have just plotted the pixel at (x_k , y_k), we next need to determine whether the pixel at position $(x_k$ +1, $y_k)$ or the one at position $(x_k$ +1, y_k −1) is closer to the circle.



Mid point Circle Algorithm

- Our decision parameter is the circle function (2) evaluated at the midpoint between these two pixels:

$$p_k = f_{circle}\left(x_k + 1, y_k - \frac{1}{2}\right)$$
$$= (x_k + 1)^2 + \left(y_k - \frac{1}{2}\right)^2 - r^2 \qquad (4)$$

Mid point Circle Algorithm

- If $p_k < 0$, this midpoint is inside the circle and the pixel on scan line y_k is closer to the circle boundary.
- Otherwise, the midpoint is outside or on the circle boundary, and we select the pixel on scan line $y_k - 1$.
- Successive decision parameters are obtained using incremental calculations.

Mid point Circle Algorithm

- We obtain a recursive expression for the next decision parameter by evaluating the circle function at sampling position $x_{k+1} + 1 = x_k + 2$

$$p_{k+1} = f_{circle}\left(x_{k+1}+1, y_{k+1}-\frac{1}{2}\right)$$

$$= \left[(x_k+1)+1\right]^2 + \left(y_{k+1}-\frac{1}{2}\right)^2 - r^2$$

or

$$p_{k+1} = p_k + 2(x_k+1) + \left(y^2{}_{k+1} - y^2{}_k\right) - (y_{k+1} - y_k) + 1$$

- where y_{k+1} is either y_k or y_{k-1},depending on the sign of p_k.

Mid point Circle Algorithm

- Increments for obtaining p_{k+1} are either
  - $2x_{k+1}$ +1 (if p_k is negative) or
  - $2x_{k+1}$ +1− $2y_{k+1}$ (if p_k is positive)
- Evaluation of the terms $2x_{k+1}$ and $2y_{k+1}$ can also be done incrementally as:

$$2x_{k+1} = 2x_k + 2$$

$$2y_{k+1} = 2y_k - 2$$

Mid point Circle Algorithm

- At the start position (0, *r*), these two terms (2x, 2y) have the values 0 and 2*r*, respectively.
- Each successive value is obtained by adding 2 to the previous value of *2x* and subtracting 2 from the previous value of 2*y.*

Mid point Circle Algorithm

- The initial decision parameter is obtained by evaluating the circle function at the start position $(x_0 , y_0)=(0, r)$:

$$p_0 = f_{circle}\left(1, r - \frac{1}{2}\right)$$

$$= 1 + \left(r - \frac{1}{2}\right)^2 - r^2$$

or

$$p_0 = \frac{5}{4} - r$$

Mid point Circle Algorithm

- If the radius r is specified as an integer, we can simply round p_0 to

$$p_0 = 1 - r \quad (\text{for } r \text{ an integer})$$

- since all increments are integers.

Summary of the Algorithm

- As in Bresenham's line algorithm, the midpoint method calculates pixel positions along the circumference of a circle using integer additions and subtractions, assuming that the circle parameters are specified in screen coordinates. We can summarize the steps in the midpoint circle algorithm as follows.

Algorithm

1. Input radius r and circle center (x_c, y_c),and obtain the first point on the circumference of a circle centered on the origin as $(x_0, y_0) = (0, r)$
2. Calculate the initial value of the decision parameter as

$$p_0 = \frac{5}{4} - r$$

3. At each x_k position, starting at $k = 0$, perform the following test: If $p_k < 0$, the next point along the circle centered on $(0,0)$ is (x_{k+1}, y_k) and

$$p_{k+1} = p_k + 2x_{k+1} + 1$$

Otherwise, the next point along the circle is $(x_k + 1, y_k - 1)$ and

$$p_{k+1} = p_k + 2x_{k+1} + 1 - 2y_{k+1}$$

where $2x_{k+1} = 2x_k + 2$ and $2y_{k+1} = 2y_k - 2$.

4. Determine symmetry points in the other seven octants.
5. Move each calculated pixel position (x, y) onto the circular path centered on (x_0, y_0) and plot the coordinate values: $x = x + x_c, y = y + y_c$

6. Repeat steps 3 through 5 until $x \geq y$.

Example

- Given a circle radius $r = 10$, we demonstrate the midpoint circle algorithm by determining positions along the circle octant in the first quadrant from $x = 0$ to $x = y$. The initial value of the decision parameter is

$$p_0 = 1 - r = -9$$

Example

- For the circle centered on the coordinate origin, the initial point is (x_0 , y_0) =(0,10), and initial increment terms for calculating the decision parameters are

$$2x_0 = 0, 2y_0 = 20$$

- Successive decision parameter values and positions along the circle path are calculated using the midpoint method as shown in the table.

Example

k	p_k	(x_{k+1}, y_{k+1})	$2x_{k+1}$	$2y_{k+1}$
0	-9	(1,10)	2	20
1	-6	(2,10)	4	20
2	-1	(3,10)	6	20
3	6	(4,9)	8	18
4	-3	(5,9)	10	18
5	8	(6,8)	12	16
6	5	(7,7)	14	14

Example

- A plot of the generated pixel positions in the first quadrant is shown in Figure 5.

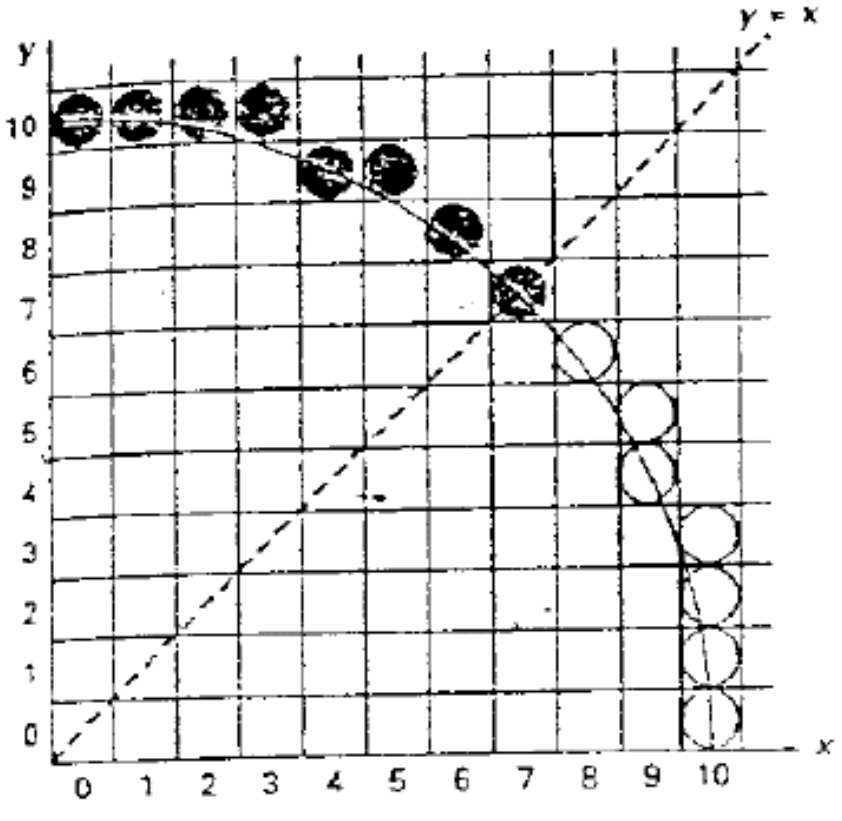


Figure 5

Midpoint Ellipse Algorithm

- Ellipse equations are greatly simplified if the major and minor axes are oriented to align with the coordinate axes.
- In Fig. 3-22, we show an ellipse in “standard position” with major and minor axes oriented parallel to the *x* and *y* axes.
- Parameter r_x for this example labels the semimajor axis, and parameter r_y labels the semiminor axis.

Midpoint Ellipse Algorithm

- The equation for the ellipse shown in Fig. 3-22 can be written in terms of the ellipse center coordinates and parameters r_x and r_y as

$$\left(\frac{x - x_c}{r_x}\right)^2 + \left(\frac{y - y_c}{r_y}\right)^2 = 1 \qquad (3\text{-}37)$$

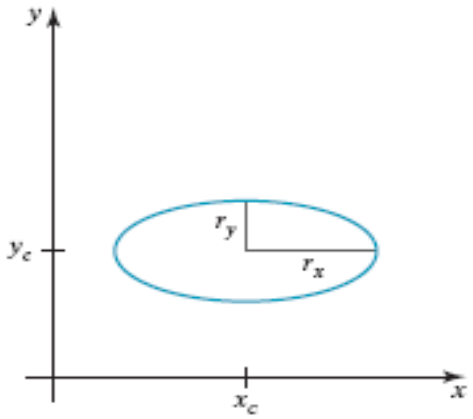


FIGURE 3–22 Ellipse centered at (x_c, y_c) with semimajor axis r_x and semiminor axis r_y.

Midpoint Ellipse Algorithm

- Using polar coordinates r and θ, we can also describe the ellipse in standard position with the parametric equations

$$x = x_c + r_x \cos\theta$$
$$y = y_c + r_y \sin\theta \qquad (3\text{-}38)$$

Midpoint Ellipse Algorithm

- The midpoint ellipse method is applied throughout the first quadrant in two parts.
- Figure 3-25 shows the division of the first quadrant according to the slope of an ellipse with $r_x < r_y$.

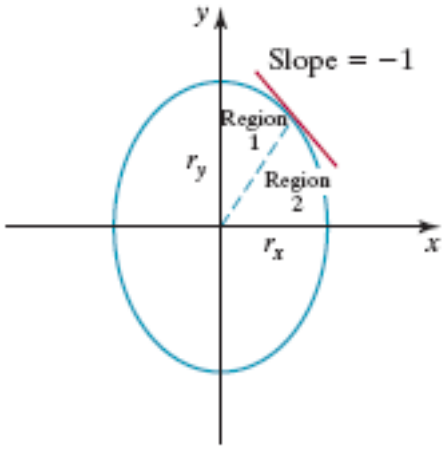


FIGURE 3-25 Ellipse processing regions. Over region 1, the magnitude of the ellipse slope is less than 1.0; over region 2, the magnitude of the slope is greater than 1.0.

Midpoint Ellipse Algorithm

- Regions 1 and 2 (Fig. 3-25) can be processed in various ways.
- We can start at position $(0, r_y)$ and step clockwise along the elliptical path in the first quadrant, shifting from unit steps in x to unit steps in y when the slope becomes less than −1.0.
- Alternatively, we could start at $(r_x, 0)$ and select points in a counterclockwise order, shifting from unit steps in y to unit steps in x when the slope becomes greater than −1.0.

Midpoint Ellipse Algorithm

- We define an ellipse function from Eq. 3-37 with $(x_c , y_c) = (0, 0)$ as
- which has the following properties:

$$f_{\text{ellipse}}(x, y) = r_y^2 x^2 + r_x^2 y^2 - r_x^2 r_y^2 \qquad (3\text{-}39)$$

$$f_{\text{ellipse}}(x, y) \begin{cases} < 0, & \text{if } (x, y) \text{ is inside the ellipse boundary} \\ = 0, & \text{if } (x, y) \text{ is on the ellipse boundary} \\ > 0, & \text{if } (x, y) \text{ is outside the ellipse boundary} \end{cases} \qquad (3\text{-}40)$$

Midpoint Ellipse Algorithm

- Starting at $(0, r_y)$, we take unit steps in the x direction until we reach the boundary between region 1 and region 2 (Fig. 3-25).
- Then we switch to unit steps in the y direction over the remainder of the curve in the first quadrant.
- At each step we need to test the value of the slope of the curve.

Midpoint Ellipse Algorithm

- The ellipse slope is calculated from Eq. 3-39 as

$$\frac{dy}{dx} = -\frac{2r_y^2 x}{2r_x^2 y} \qquad (3\text{-}41)$$

- At the boundary between region 1 and region 2, $dy/dx = -1.0$ and

- Therefore, we move out of region 1 whenever

$$2r_y^2 x = 2r_x^2 y$$

$$2r_y^2 x \geq 2r_x^2 y \qquad (3\text{-}42)$$

Midpoint Ellipse Algorithm

- Figure 3-26 shows the midpoint between the two candidate pixels at sampling position x_k +1 in the first region.
- Assuming position (x_k , y_k) has been selected in the previous step, we determine the next position along the ellipse path by evaluating the decision parameter (that is, the ellipse function 3-39) at this midpoint:

$$p1_k = f_{\text{ellipse}}\left(x_k + 1, y_k - \frac{1}{2}\right)$$

$$= r_y^2(x_k + 1)^2 + r_x^2\left(y_k - \frac{1}{2}\right)^2 - r_x^2 r_y^2 \qquad (3\text{-}43)$$

Midpoint Ellipse Algorithm

- If $p1_k < 0$, the midpoint is inside the ellipse and the pixel on scan line y_k is closer to the ellipse boundary.
- Otherwise, the midposition is outside or on the ellipse boundary, and we select the pixel on scan line $y_k - 1$.

Midpoint Ellipse Algorithm

- At the next sampling position (x_k+1 + 1 = x_k + 2), the decision parameter for region 1 is evaluated as

$$p1_{k+1} = f_{\text{ellipse}}\left(x_{k+1} + 1, y_{k+1} - \frac{1}{2}\right)$$

$$= r_y^2[(x_k + 1) + 1]^2 + r_x^2\left(y_{k+1} - \frac{1}{2}\right)^2 - r_x^2 r_y^2$$

or

$$p1_{k+1} = p1_k + 2r_y^2(x_k + 1) + r_y^2 + r_x^2\left[\left(y_{k+1} - \frac{1}{2}\right)^2 - \left(y_k - \frac{1}{2}\right)^2\right] \qquad (3\text{-}44)$$

where y_{k+1} is either y_k or $y_k - 1$, depending on the sign of $p1_k$.

Midpoint Ellipse Algorithm

- Decision parameters are incremented by the following amounts:

$$\text{increment} = \begin{cases} 2r_y^2 x_{k+1} + r_y^2, & \text{if } p1_k < 0 \\ 2r_y^2 x_{k+1} + r_y^2 - 2r_x^2 y_{k+1}, & \text{if } p1_k \geq 0 \end{cases}$$

Midpoint Ellipse Algorithm

- At the initial position $(0, r_y)$, these two terms evaluate to

$$2r_y^2 x = 0 \qquad (3\text{-}45)$$

$$2r_x^2 y = 2r_x^2 r_y \qquad (3\text{-}46)$$

- As x and y are incremented, updated values are obtained by adding $2r^2_y$ to the current value of the increment term in Eq. 3-45 and subtracting $2r^2_x$ from the current value of the increment term in Eq. 3-46.
- The updated increment values are compared at each step, and we move from region 1 to region 2 when condition 3-42 is satisfied.

Midpoint Ellipse Algorithm

- In region 1, the initial value of the decision parameter is obtained by evaluating the ellipse function at the start position $(x_0, y_0) = (0, r_y)$:

$$p1_0 = f_{\text{ellipse}}\left(1, r_y - \frac{1}{2}\right)$$
$$= r_y^2 + r_x^2\left(r_y - \frac{1}{2}\right)^2 - r_x^2 r_y^2$$

or

$$p1_0 = r_y^2 - r_x^2 r_y + \frac{1}{4} r_x^2 \qquad (3\text{-}47)$$

Midpoint Ellipse Algorithm

- Over region 2, we sample at unit intervals in the negative *y* direction, and the midpoint is now taken between horizontal pixels at each step (Fig. 3-27).
- For this region, the decision parameter is evaluated as

$$p2_k = f_{\text{ellipse}}\left(x_k + \frac{1}{2}, y_k - 1\right)$$

$$= r_y^2\left(x_k + \frac{1}{2}\right)^2 + r_x^2(y_k - 1)^2 - r_x^2 r_y^2 \qquad (3\text{-}48)$$

Midpoint Ellipse Algorithm

- If $p2_k > 0$, the midposition is outside the ellipse boundary, and we select the pixel at x_k.
- If $p2_k <= 0$, the midpoint is inside or on the ellipse boundary, and we select
- pixel position x_{k+1}.

Midpoint Ellipse Algorithm

- To determine the relationship between successive decision parameters in region 2,we evaluate the ellipse function at the next sampling step $y_k+1 -1 = y_k -2$:

$$p2_{k+1} = f_{\text{ellipse}}\left(x_{k+1} + \frac{1}{2}, y_{k+1} - 1\right)$$

$$= r_y^2\left(x_{k+1} + \frac{1}{2}\right)^2 + r_x^2[(y_k - 1) - 1]^2 - r_x^2 r_y^2 \qquad (3\text{-}49)$$

or

$$p2_{k+1} = p2_k - 2r_x^2(y_k - 1) + r_x^2 + r_y^2\left[\left(x_{k+1} + \frac{1}{2}\right)^2 - \left(x_k + \frac{1}{2}\right)^2\right] \qquad (3\text{-}50)$$

with x_{k+1} set either to x_k or to $x_k + 1$, depending on the sign of $p2_k$.

Midpoint Ellipse Algorithm

- When we enter region 2, the initial position (x_0, y_0) is taken as the last position selected in region 1 and the initial decision parameter in region 2 is then

$$p2_0 = f_{\text{ellipse}}\left(x_0 + \frac{1}{2}, y_0 - 1\right)$$

$$= r_y^2\left(x_0 + \frac{1}{2}\right)^2 + r_x^2(y_0 - 1)^2 - r_x^2 r_y^2 \qquad (3\text{-}51)$$

Algorithm

Midpoint Ellipse Algorithm

1. Input r_x, r_y, and ellipse center (x_c, y_c), and obtain the first point on an ellipse centered on the origin as

$$(x_0, y_0) = (0, r_y)$$

2. Calculate the initial value of the decision parameter in region 1 as

$$p1_0 = r_y^2 - r_x^2 r_y + \frac{1}{4} r_x^2$$

3. At each x_k position in region 1, starting at $k = 0$, perform the following test. If $p1_k < 0$, the next point along the ellipse centered on $(0, 0)$ is (x_{k+1}, y_k) and

$$p1_{k+1} = p1_k + 2r_y^2 x_{k+1} + r_y^2$$

Otherwise, the next point along the ellipse is $(x_k + 1, y_k - 1)$ and

$$p1_{k+1} = p1_k + 2r_y^2 x_{k+1} - 2r_x^2 y_{k+1} + r_y^2$$

with

$$2r_y^2 x_{k+1} = 2r_y^2 x_k + 2r_y^2, \qquad 2r_x^2 y_{k+1} = 2r_x^2 y_k - 2r_x^2$$

and continue until $2r_y^2 x \geq 2r_x^2 y$.

4. Calculate the initial value of the decision parameter in region 2 as

$$p2_0 = r_y^2 \left(x_0 + \frac{1}{2} \right)^2 + r_x^2 (y_0 - 1)^2 - r_x^2 r_y^2$$

where (x_0, y_0) is the last position calculated in region 1.

5. At each y_k position in region 2, starting at $k = 0$, perform the following test. If $p2_k > 0$, the next point along the ellipse centered on $(0, 0)$ is $(x_k, y_k - 1)$ and

$$p2_{k+1} = p2_k - 2r_x^2 y_{k+1} + r_x^2$$

Otherwise, the next point along the ellipse is $(x_k + 1, y_k - 1)$ and

$$p2_{k+1} = p2_k + 2r_y^2 x_{k+1} - 2r_x^2 y_{k+1} + r_x^2$$

using the same incremental calculations for x and y as in region 1. Continue until $y = 0$.

6. For both regions, determine symmetry points in the other three quadrants.

7. Move each calculated pixel position (x, y) onto the elliptical path centered on (x_c, y_c) and plot the coordinate values:

$$x = x + x_c, \qquad y = y + y_c$$

Example

- Given input ellipse parameters r_x =8 and r_y = 6, we illustrate the steps in the midpoint ellipse algorithm by determining raster positions along the ellipse path in the first quadrant.
- Initial values and increments for the decision parameter calculations are

$$2r_y^2 x = 0 \qquad (\text{with increment } 2r_y^2 = 72)$$
$$2r_x^2 y = 2r_x^2 r_y \qquad (\text{with increment } -2r_x^2 = -128)$$

Example

- For region 1, the initial point for the ellipse centered on the origin is $(x_0, y_0) = (0, 6)$, and the initial decision parameter value is

$$p1_0 = r_y^2 - r_x^2 r_y + \frac{1}{4} r_x^2 = -332$$

- Successive midpoint decision parameter values and the pixel positions along the ellipse are listed in the following table.

Example

k	$p1_k$	(x_{k+1}, y_{k+1})	$2r_y^2 x_{k+1}$	$2r_x^2 y_{k+1}$
0	−332	(1, 6)	72	768
1	−224	(2, 6)	144	768
2	−44	(3, 6)	216	768
3	208	(4, 5)	288	640
4	−108	(5, 5)	360	640
5	288	(6, 4)	432	512
6	244	(7, 3)	504	384

Example

- We now move out of region 1, since

$$2r^2{}_y\, x > 2r^2{}_x\, y.$$

- For region 2, the initial point is

$$(x_0, y_0) = (7, 3)$$

- and the initial decision parameter is

$$p2_0 = f_{\text{ellipse}}\left(7 + \frac{1}{2}, 2\right) = -151$$

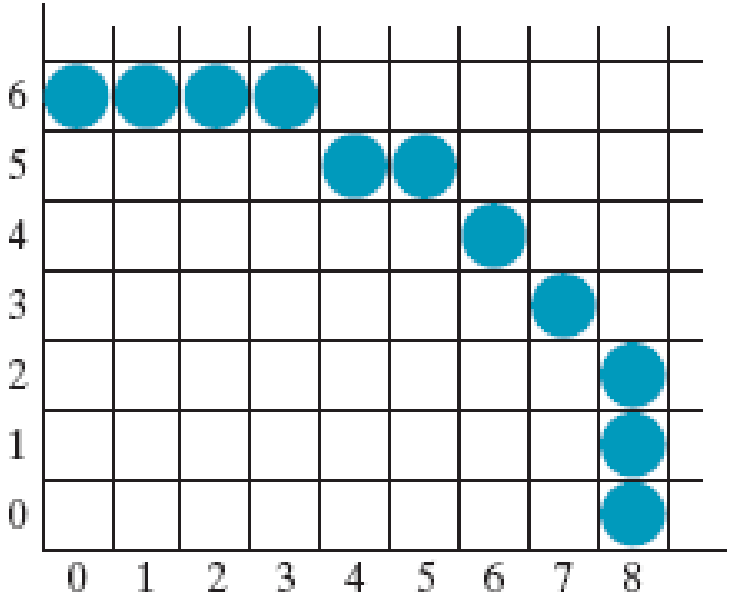
Example

- The remaining positions along the ellipse path in the first quadrant are then calculated as

k	$p1_k$	(x_{k+1}, y_{k+1})	$2r_y^2 x_{k+1}$	$2r_x^2 y_{k+1}$
0	−151	(8, 2)	576	256
1	233	(8, 1)	576	128
2	745	(8, 0)	—	—

Example

- A plot of the calculated positions for the ellipse within the first quadrant is shown bellow:

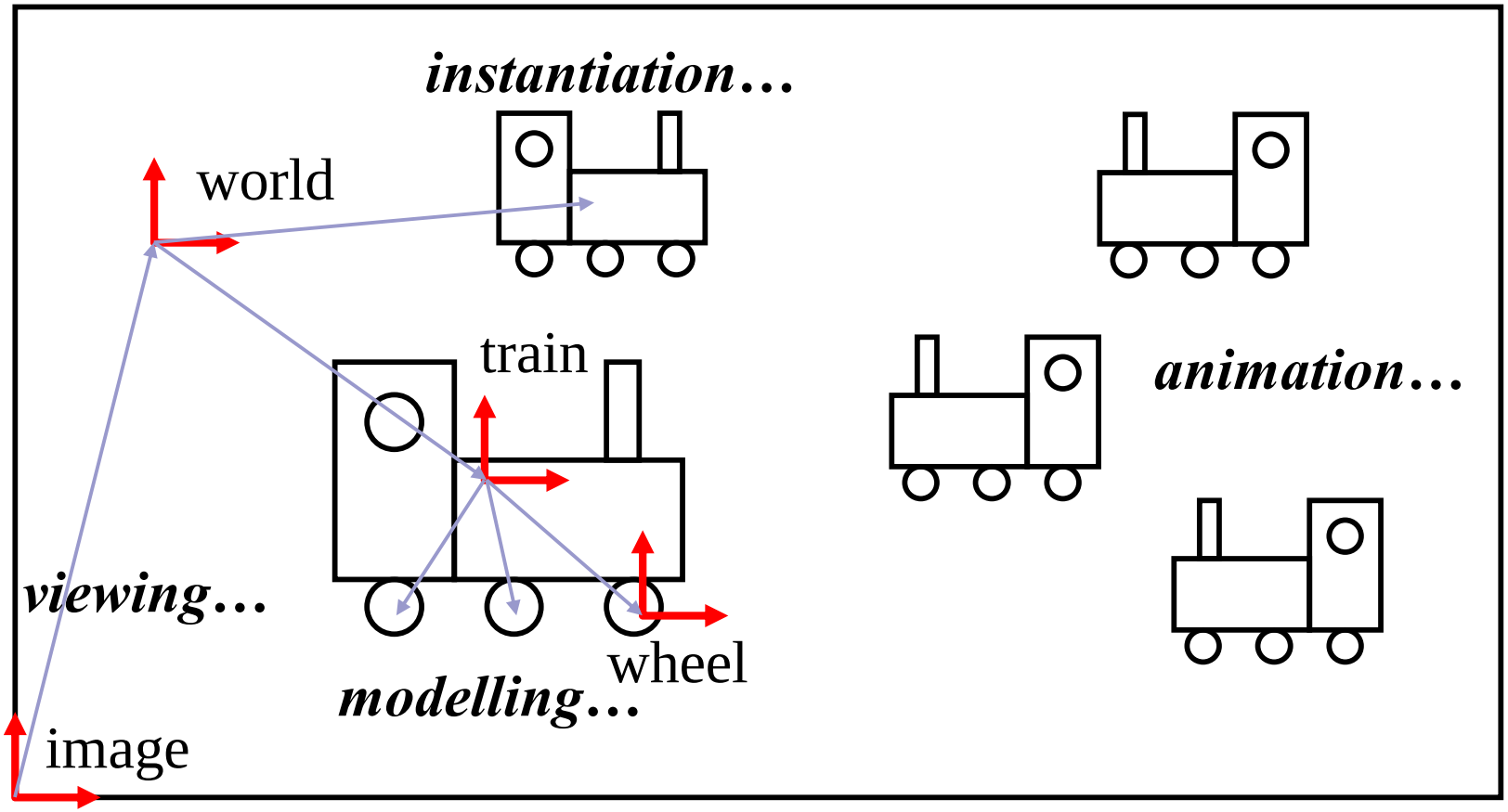


Computer Graphics
2D transformations

Overview

- Why transformations?
- Basic transformations:
  - translation, rotation, scaling
- Combining transformations
  - homogenous coordinates, transform. Matrices

Transformations



Why transformation?

- Model of objects
  world coordinates: *km, mm, etc.*
  Hierarchical models::
  human = torso + arm + arm + head + leg + leg
  arm = upperarm + lowerarm + hand …
- Viewing
  zoom in, move drawing, etc.
- Animation

Translation

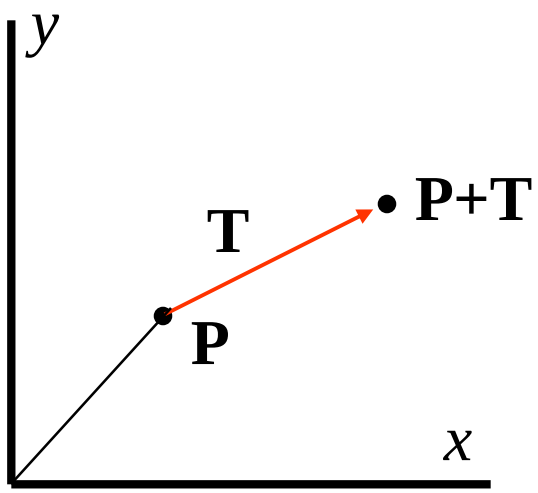
Translate over vector (t_x, t_y)

$x'=x+t_x$, $y'=y+t_y$

or

$$\mathbf{P}' = \mathbf{P} + \mathbf{T}, \quad \text{with}$$

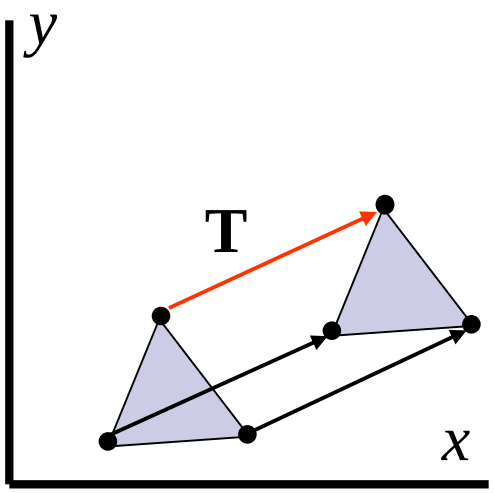
$$\mathbf{P}' = \begin{pmatrix} x' \\ y' \end{pmatrix}, \mathbf{P} = \begin{pmatrix} x \\ y \end{pmatrix} \text{ and } \mathbf{T} = \begin{pmatrix} t_x \\ t_y \end{pmatrix}$$



Translation polygon

Translate polygon:

- Apply the same operation on all points.
- Works always, for all transformations of objects defined as a set of points.



Rotation

Rotate over an angle α :

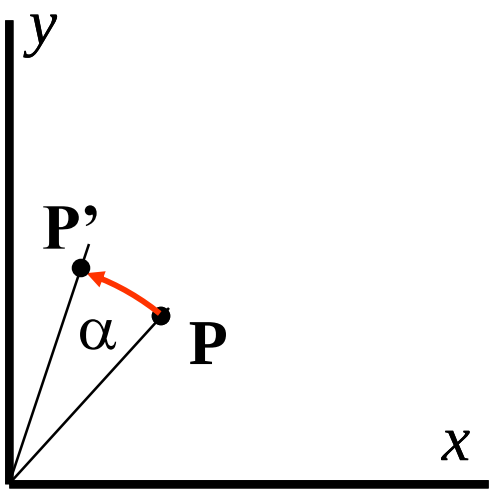
$$x' = x\cos\alpha - y\sin\alpha$$

$$y' = x\sin\alpha + y\cos\alpha$$

Or

$$\mathbf{P}' = \mathbf{R}\mathbf{P}, \text{ with}$$

$$\mathbf{P}' = \begin{pmatrix} x' \\ y' \end{pmatrix}, \mathbf{R} = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix} \text{ and } \mathbf{P} = \begin{pmatrix} x \\ y \end{pmatrix}$$

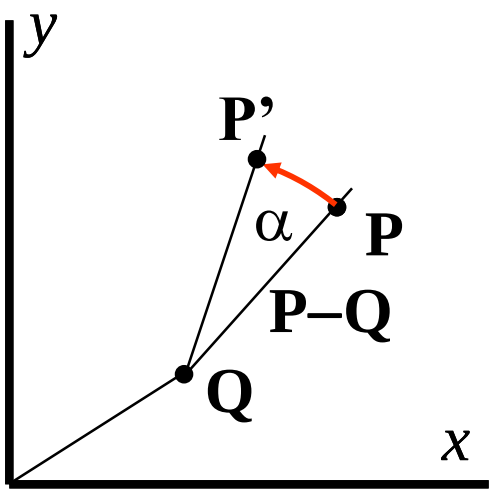


Rotation around a point Q

Rotate around origin :

$$P_x' = P_x \cos\alpha - P_y \sin\alpha$$

$$P_y' = P_x \sin\alpha + P_y \cos\alpha$$



Rotate around **Q** over an angle α :

$$P_x' = Q_x + (P_x - Q_x)\cos\alpha - (P_y - Q_y)\sin\alpha$$

$$P_y' = Q_y + (P_x - Q_x)\sin\alpha + (P_y - Q_y)\cos\alpha$$

Scaling

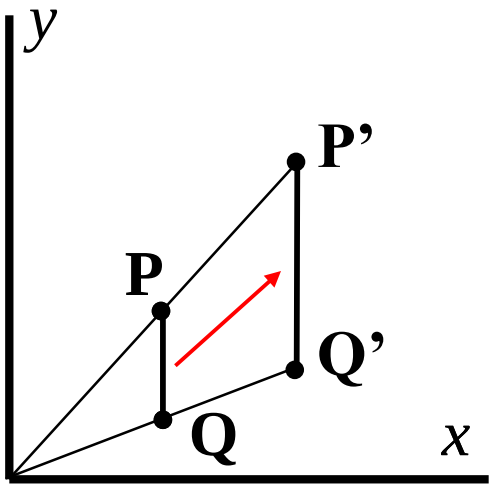
Schale with factor s_x and s_y:

$x' = s_x\, x, \quad y' = s_y\, y$

or

$$\mathbf{P}' = \mathbf{SP}, \text{ with}$$

$$\mathbf{P}' = \begin{pmatrix} x' \\ y' \end{pmatrix}, \mathbf{S} = \begin{pmatrix} s_x & 0 \\ 0 & s_y \end{pmatrix} \text{ and } \mathbf{P} = \begin{pmatrix} x \\ y \end{pmatrix}$$



Scaling with respect to a point F

Scale with factors s_x and s_y:

$$P_x' = s_x P_x, \quad P_y' = s_y P_y$$

With respect to **F**:

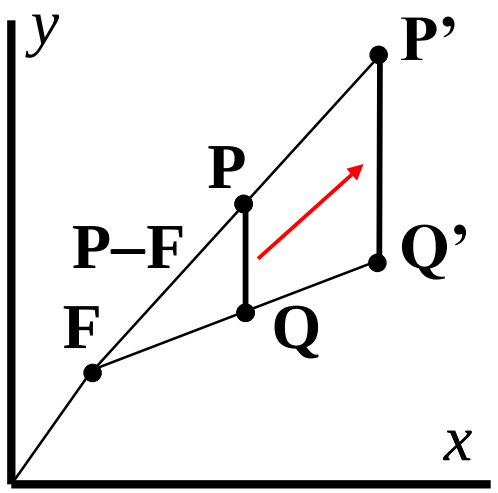
$$P_x' - F_x = s_x (P_x - F_x),$$

$$P_y' - F_y = s_y (P_y - F_y)$$

or

$$P_x' = F_x + s_x (P_x - F_x),$$

$$P_y' = F_y + s_y (P_y - F_y)$$



Transformations

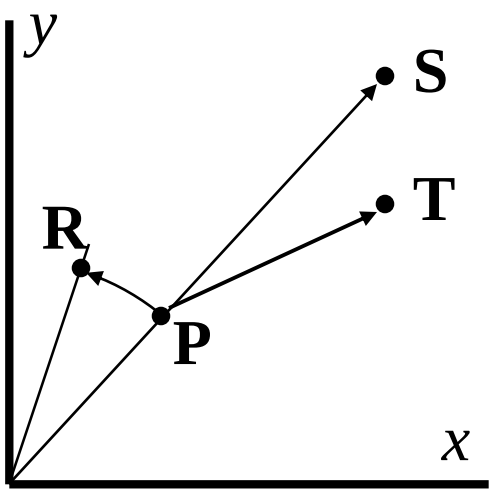
- Translate with **V**:

  T = **P** + **V**
- Schale with factor $s_x = s_y = s$:

  S = s**P**
- Rotate over angle α:

  $R'_x = \cos\alpha\, P_x - \sin\alpha\, P_y$

  $R'_y = \sin\alpha\, P_x + \cos\alpha\, P_y$



Transformations…

- Messy!
- Transformations with respect to points: even more messy!
- How to combine transformations?

Homogeneous coordinates 1

- Uniform representation of translation, rotation, scaling
- Uniform representation of points and vectors
- Compact representation of sequence of transformations

Homogeneous coordinaten 2

- Add extra coordinate:

  $\mathbf{P} = (p_x , p_y , p_h)$ or

  $\mathbf{x} = (x, y, h)$
- Cartesian coordinates: divide by h

  $\mathbf{x} = (x/h, y/h)$
- Points: $h = 1$ (for the time being…),

  vectors: $h = 0$

Translation matrix

Translation:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

or

$$\mathbf{P}' = \mathbf{T}(t_x, t_y)\mathbf{P}$$

Rotation matrix

Rotation :

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

or

$$\mathbf{P}' = \mathbf{R}(\theta)\mathbf{P}$$

Scaling matrix

$$\text{Scaling}:$$

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

or

$$\mathbf{P}' = \mathbf{S}(s_x, s_y)\mathbf{P}$$

Inverse transformations

Translation :

$$\mathbf{T}^{-1}(t_x, t_y) = \mathbf{T}(-t_x, -t_y)$$

Rotation :

$$\mathbf{R}^{-1}(\theta) = \mathbf{R}(-\theta)$$

Scaling :

$$\mathbf{S}^{-1}(s_x, s_y) = \mathbf{S}(\frac{1}{s_x}, \frac{1}{s_y})$$

Combining transformations 1

$\mathbf{P}' = \mathbf{M}_1\mathbf{P}$ first transformation…

$\mathbf{P}'' = \mathbf{M}_2\mathbf{P}'$ second transformation…

Combined :

$$\mathbf{P}'' = \mathbf{M}_2(\mathbf{M}_1\mathbf{P})$$
$$= \mathbf{M}_2\mathbf{M}_1\mathbf{P}$$
$$= \mathbf{M}\mathbf{P} \quad \text{with } \mathbf{M} = \mathbf{M}_2\mathbf{M}_1$$

Combining transformations 2

$$\mathbf{P}' = \mathbf{T}(t_{1x}, t_{1y})\mathbf{P} \qquad \text{first translation}$$

$$\mathbf{P}'' = \mathbf{T}(t_{2x}, t_{2y})\mathbf{P}' \qquad \text{second translation}$$

Combined :

$$\mathbf{P}'' = \mathbf{T}(t_{2x}, t_{2y})\mathbf{T}(t_{1x}, t_{1y})\mathbf{P}$$

$$= \begin{pmatrix} 1 & 0 & t_{2x} \\ 0 & 1 & t_{2y} \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & t_{1x} \\ 0 & 1 & t_{1y} \\ 0 & 0 & 1 \end{pmatrix} \mathbf{P} = \begin{pmatrix} 1 & 0 & t_{1x} + t_{2x} \\ 0 & 1 & t_{1y} + t_{2y} \\ 0 & 0 & 1 \end{pmatrix} \mathbf{P}$$

$$= \mathbf{T}(t_{1x} + t_{2x}, t_{1x} + t_{2y})\mathbf{P}$$

Combining transformations 3

Composite translations :

$$\mathbf{T}(t_{2x}, t_{2y})\mathbf{T}(t_{1x}, t_{1y}) = \mathbf{T}(t_{1x} + t_{2x}, t_{1x} + t_{2y})$$

Composite rotations :

$$\mathbf{R}(\theta_2)R(\theta_1) = \mathbf{R}(\theta_1 + \theta_2)$$

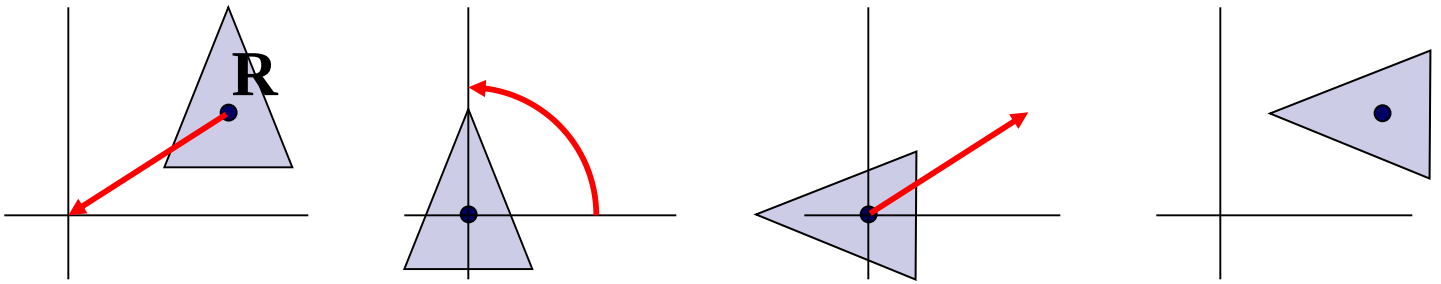
Composite scaling :

$$\mathbf{S}(s_{2x}, s_{2y})\mathbf{S}(s_{1x}, s_{1y}) = \mathbf{S}(s_{1x}s_{2x}, s_{1y}s_{2y})$$

Rotation around a point 1

Rotate over angle θ around point **R**:

1) Translate such that **R** coincides with origin;
2) Rotate over angle θ around origin;
3) Translate back.



1) 2) 3)

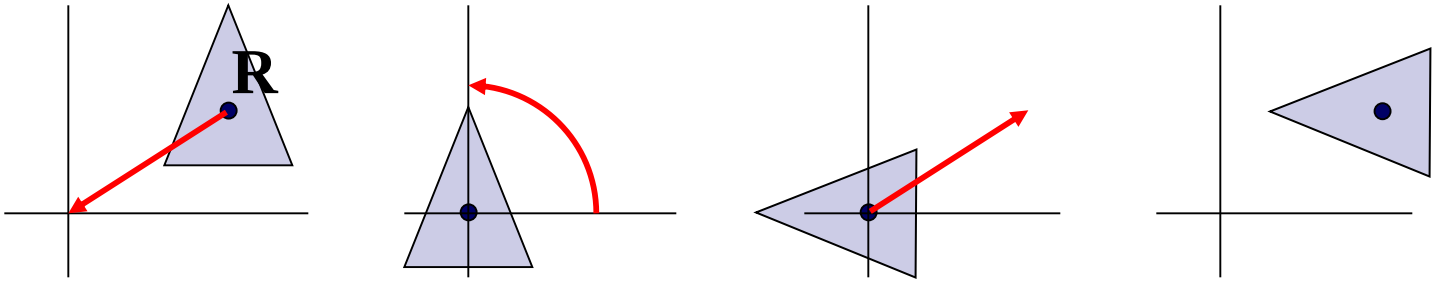
Rotation around a point 2

Rotate over angle θ around point $\mathbf{R}$:

1) $\mathbf{P}' = \mathbf{T}(-R_x, -R_y)\mathbf{P}$

2) $\mathbf{P}'' = \mathbf{R}(\theta)\mathbf{P}'$

3) $\mathbf{P}''' = \mathbf{T}(R_x, R_y)\mathbf{P}''$



1) 2) 3)

Rotation around point 3

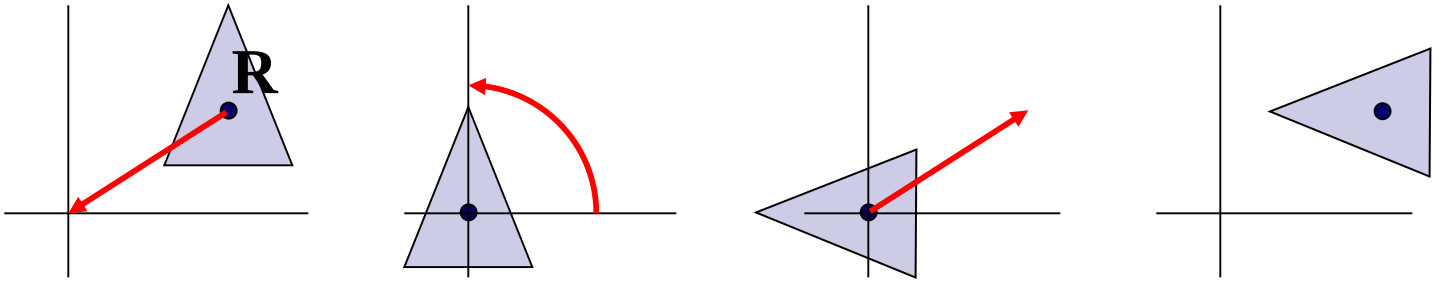
1) $\mathbf{P}' = \mathbf{T}(-R_x, -R_y)\mathbf{P}$

2) $\mathbf{P}'' = \mathbf{R}(\theta)\mathbf{P}' = \mathbf{R}(\theta)\mathbf{T}(-R_x, -R_y)\mathbf{P}$

3) $\mathbf{P}''' = \mathbf{T}(R_x, R_y)\mathbf{P}''$

$= \mathbf{T}(R_x, R_y)\mathbf{R}(\theta)\mathbf{P}'$

$= \mathbf{T}(R_x, R_y)\mathbf{R}(\theta)\mathbf{T}(-R_x, -R_y)\mathbf{P}$



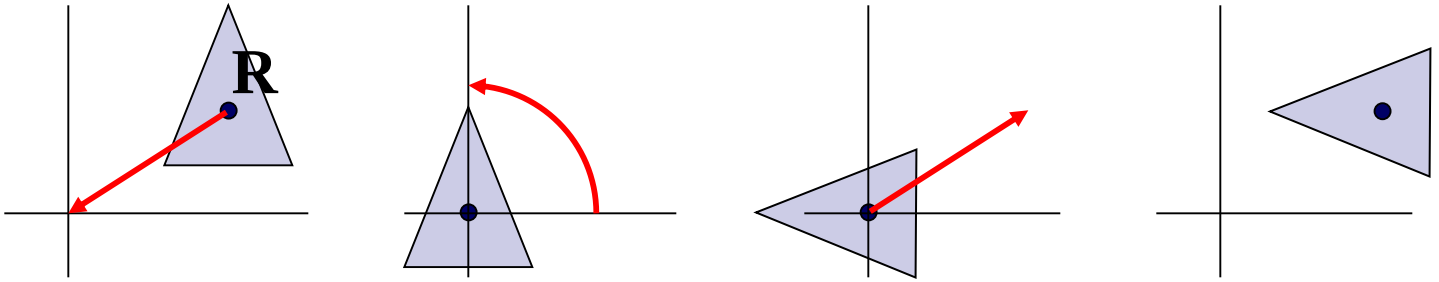
1) 2) 3)

Rotation around point 4

$$1\text{-}3)\ \mathbf{P}''' = \mathbf{T}(R_x, R_y)\mathbf{R}(\theta)\mathbf{T}(-R_x, -R_y)\mathbf{P}$$

or

$$\mathbf{P}''' = \begin{pmatrix} \cos\theta & -\sin\theta & R_x(1-\cos\theta) + R_y \sin\theta \\ \sin\theta & \cos\theta & R_y(1-\cos\theta) - R_x \sin\theta \\ 0 & 0 & 1 \end{pmatrix} \mathbf{P}$$

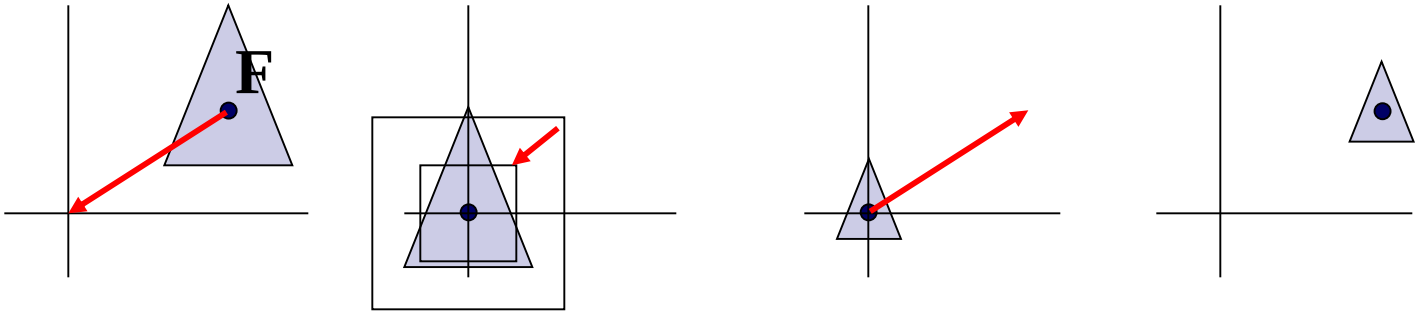


1) 2) 3)

Scaling w.r.t. point 1

Scale with factors s_x and s_x w.r.t. point $\mathbf{F}$:

1) Translate such that $\mathbf{F}$ coincides with origin;
2) Schale w.r.t. origin;
3) Translate back again.



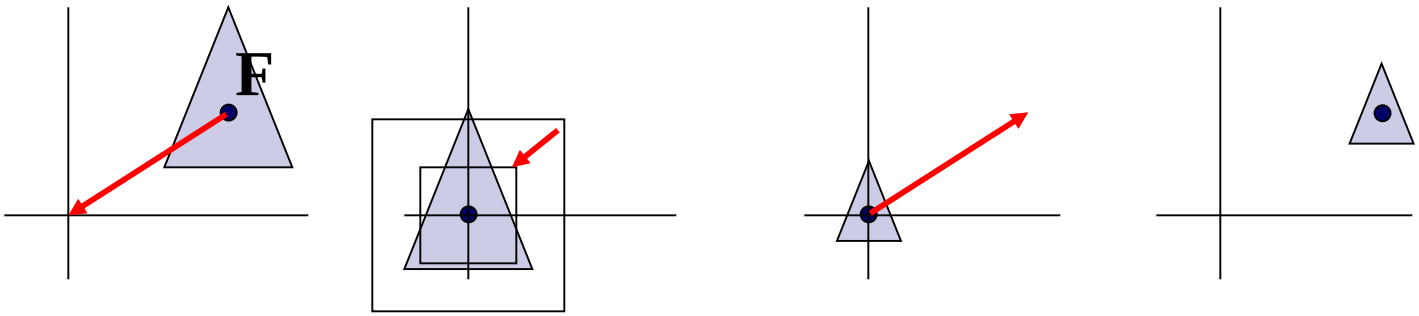
Scaling w.r.t.point 2

Schale w.r.t.point $\mathbf{F}$:

1) $\mathbf{P}' = \mathbf{T}(-F_x, -F_y)\mathbf{P}$

2) $\mathbf{P}'' = \mathbf{S}(s_x, s_y)\mathbf{P}'$

3) $\mathbf{P}''' = \mathbf{T}(F_x, F_y)\mathbf{P}''$

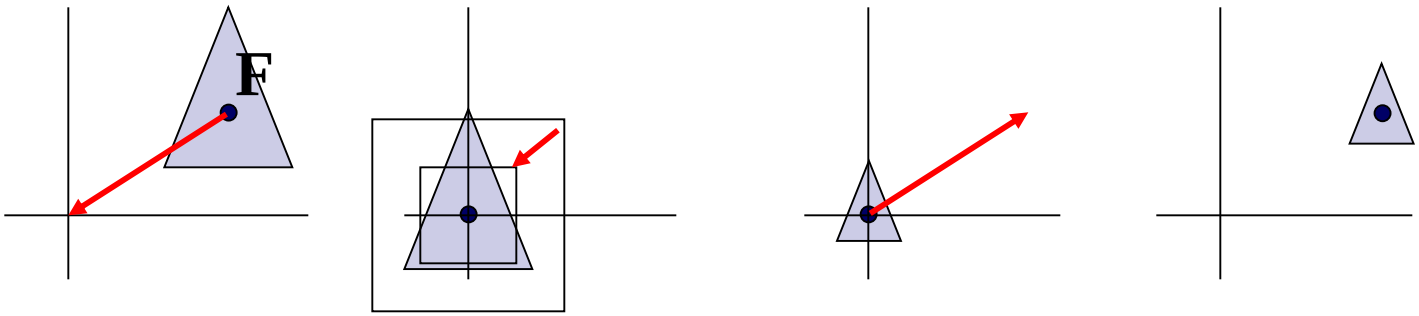


Scaling w.r.t.point 3

$$1\text{-}3)\ \mathbf{P}''' = \mathbf{T}(F_x, F_y)\mathbf{S}(s_x, s_y)\mathbf{T}(-F_x, -F_y)\mathbf{P}$$

or

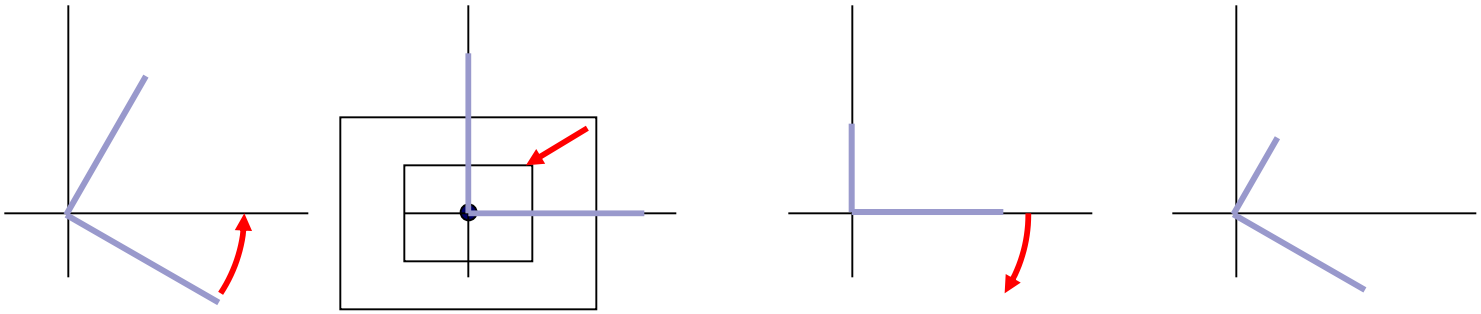
$$\mathbf{P}''' = \begin{pmatrix} s_x & 0 & F_x(1-s_x) \\ 0 & s_y & F_y(1-s_y) \\ 0 & 0 & 1 \end{pmatrix}\mathbf{P}$$



Scale in other directions 1

Scale with factors s_1 and s_2 w.r.t.rotated frame:

1) Rotate such that frame coincides with standard xy-frame;
2) Scale w.r.t.origin;
3) Rotate back again.



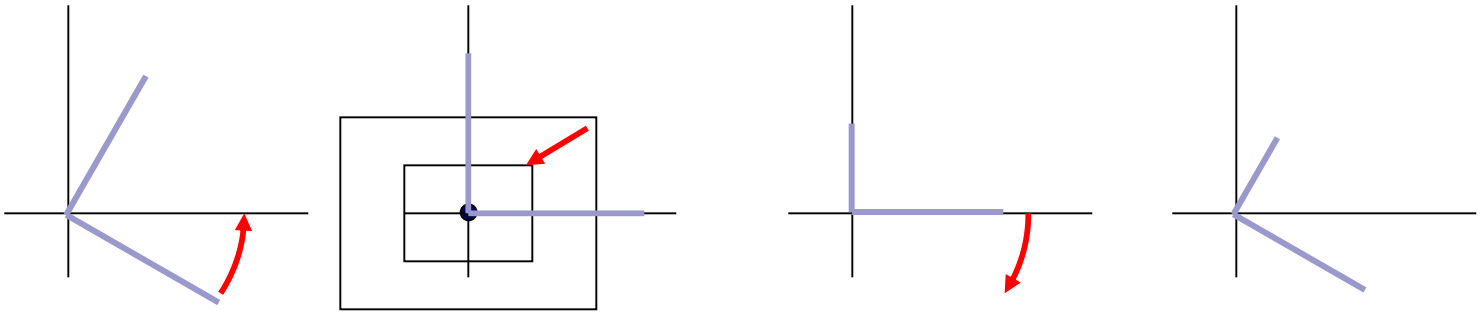
Scale in other directions 2

Scale in other direction :

1) $\mathbf{P}' = \mathbf{R}(\theta)\mathbf{P}$

2) $\mathbf{P}'' = \mathbf{S}(s_1, s_2)\mathbf{P}'$

3) $\mathbf{P}''' = \mathbf{R}(-\theta)\mathbf{P}''$

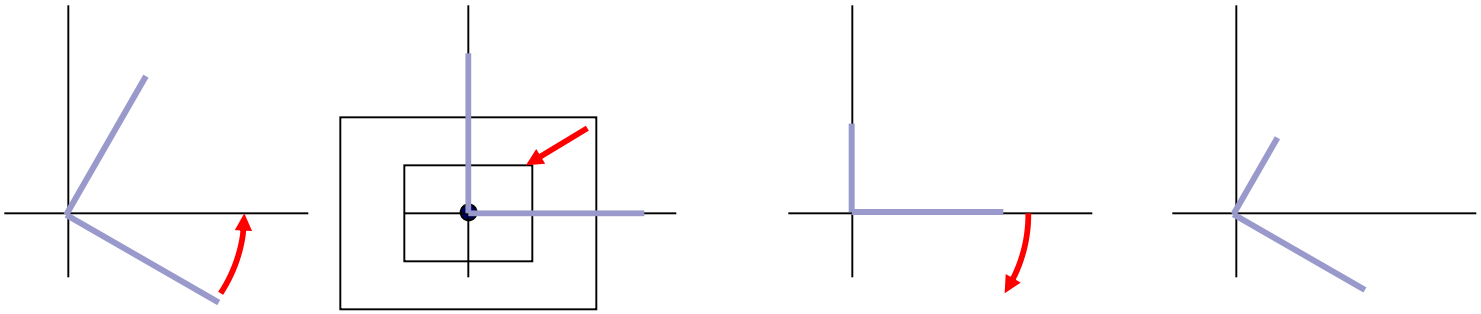


Scale in other directions 3

1-3) $\mathbf{P}''' = \mathbf{R}(-\theta)\mathbf{S}(s_1, s_2)\mathbf{R}(\theta)\mathbf{P}$

or

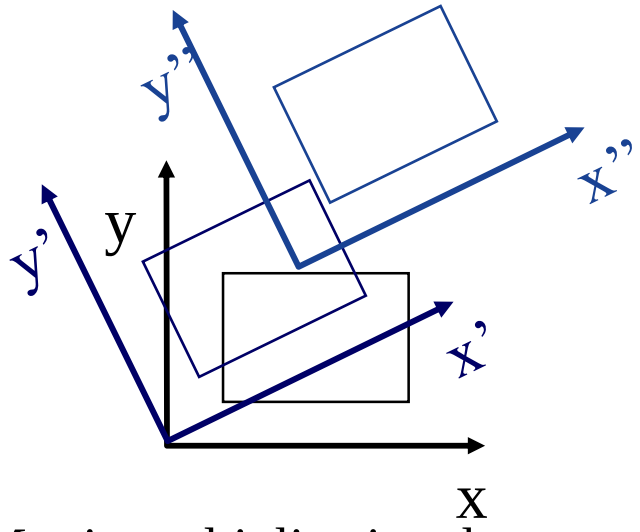
$$\mathbf{P}''' = \begin{pmatrix} s_1\cos^2\theta + s_2\sin^2\theta & (s_2 - s_1)\cos\theta\sin\theta & 0 \\ (s_2 - s_1)\cos\theta\sin\theta & s_1\sin^2\theta + s_2\cos^2\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}\mathbf{P}$$



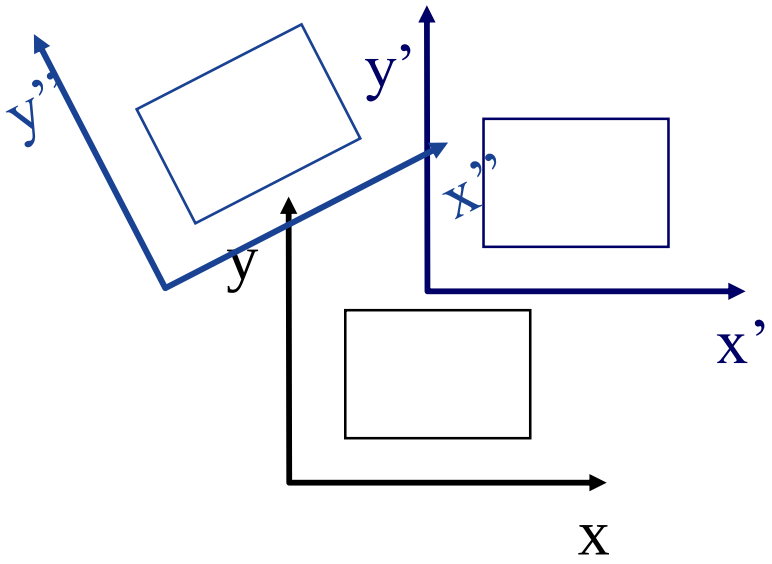
1) 2) 3)

Order of transformations 1

Rotation, translation…



Translation, rotation…



Matrix multiplication does **not** commute.

The order of transformations makes a difference!

Order of transformations 2

- Pre-multiplication:

  $\mathbf{P' = M}_n \mathbf{M}_{n-1} \ldots \mathbf{M}_2 \mathbf{M}_1 \mathbf{P}$

  Transformation $\mathbf{M}_n$ in global coordinates
- Post-multiplication:

  $\mathbf{P' = M}_1 \mathbf{M}_2 \ldots \mathbf{M}_{n-1} \mathbf{M}_n \mathbf{P}$

  Transformation $\mathbf{M}_n$ in local coordinates: the coordinate system after application of $\mathbf{M}_1 \mathbf{M}_2 \ldots \mathbf{M}_{n-1}$

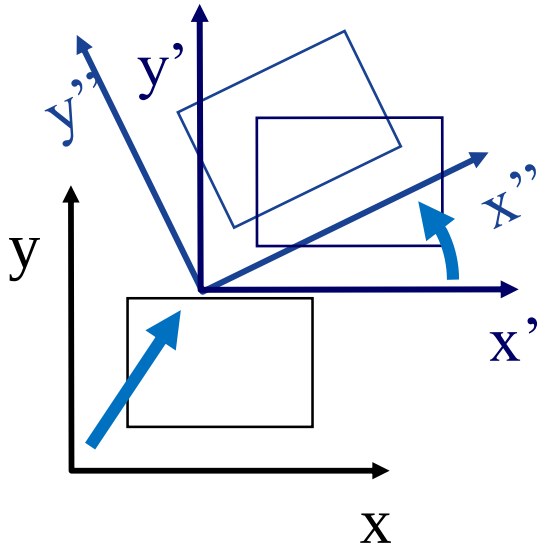
Order of transformations 3

OpenGL: **`glRotate, glScale`**, etc.:

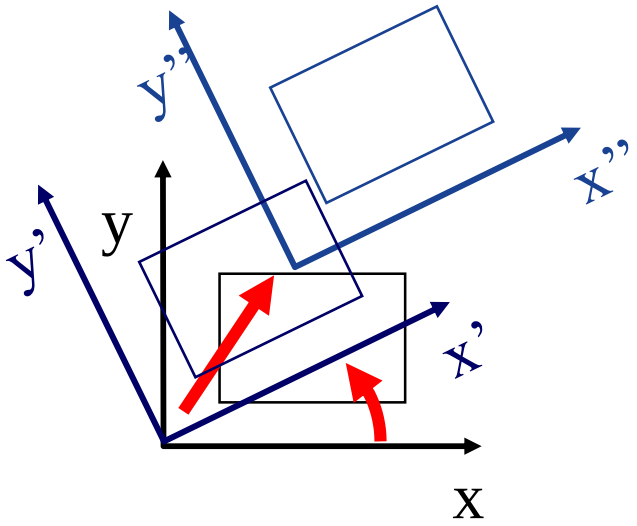
- Post-multiplication of current transformation matrix
- Always transformation in local coordinates
- Global coordinate version: read in reverse order

Order of transformations 4

***Local** transformations:*



***Global** transformations:*



***Local** trafo*

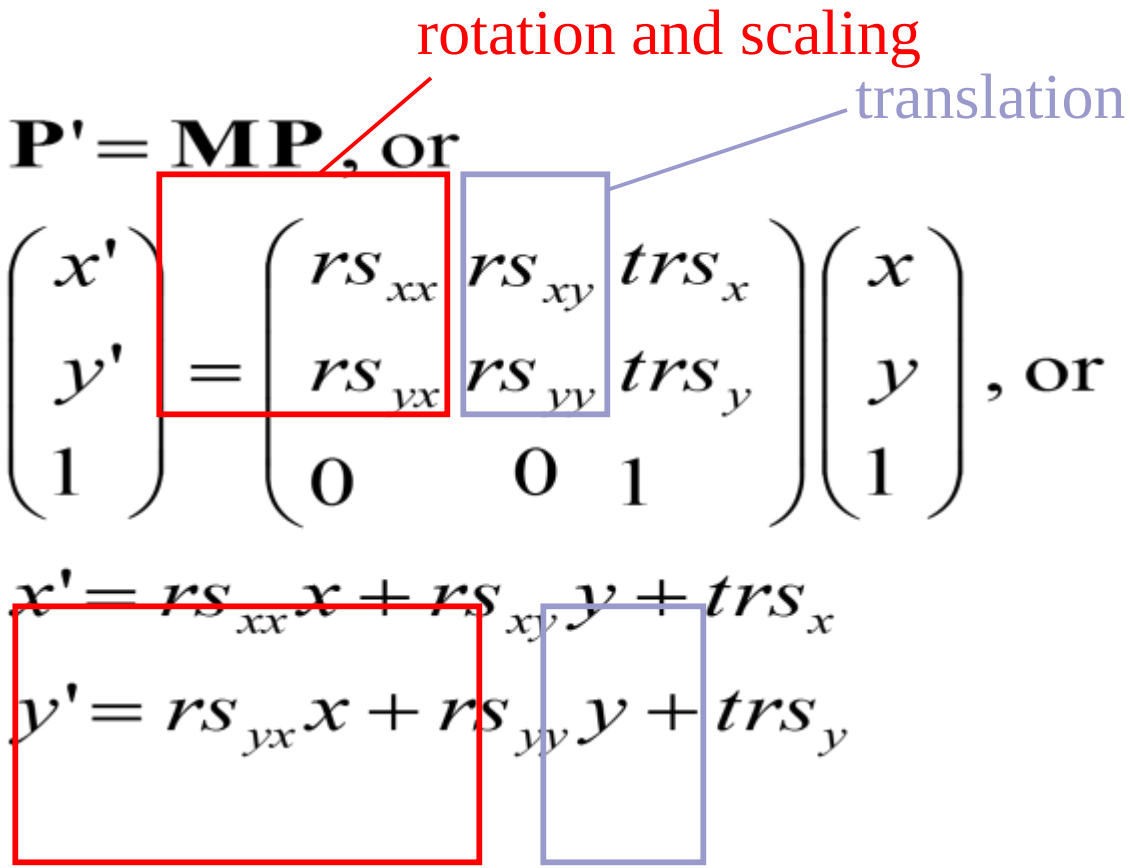
interpretation

```
glTranslate(…);
glRotate(…);
```

***Global** trafo*

interpretation

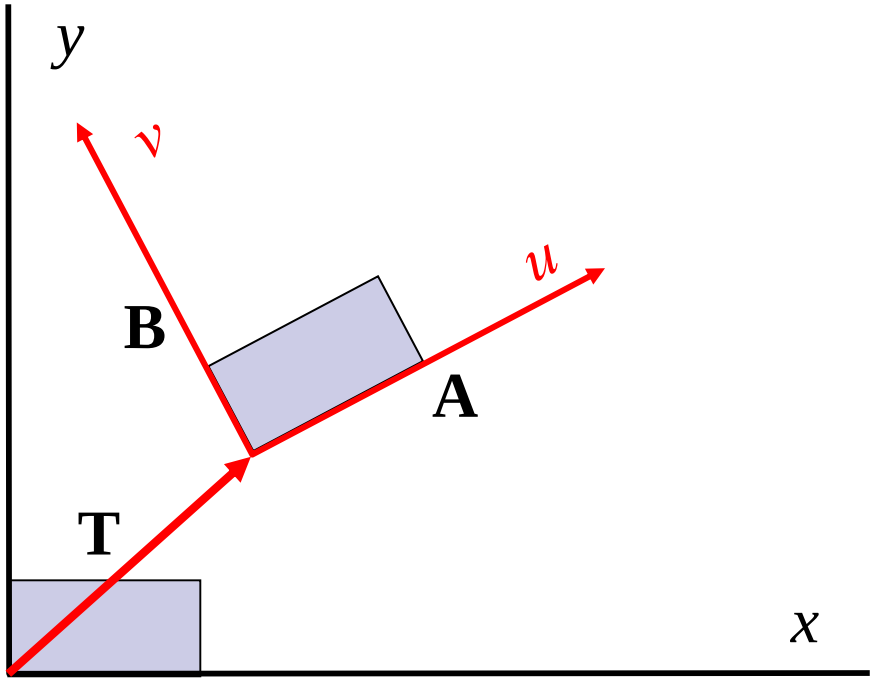
Matrices in general



Direct construction of matrix

If you know the target frame:
Construct matrix directly.

Define shape in nice local *u*,*v* coordinates, use matrix transformation to put it in *x*,*y* space.



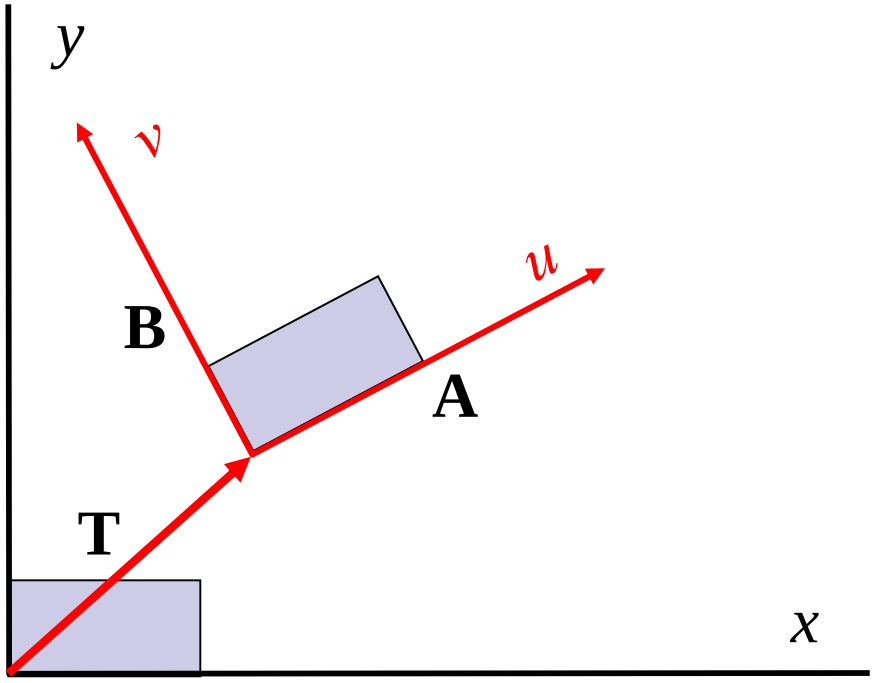
Direct construction of matrix

If you know the target frame:
Construct matrix directly.

$$\mathbf{P'} = \mathbf{A}u + \mathbf{B}v + \mathbf{T}, \text{ or}$$

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} \mathbf{A} & \mathbf{B} & \mathbf{T} \end{pmatrix} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix}, \text{ or}$$

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} A_x & B_x & T_x \\ A_y & B_y & T_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix}$$



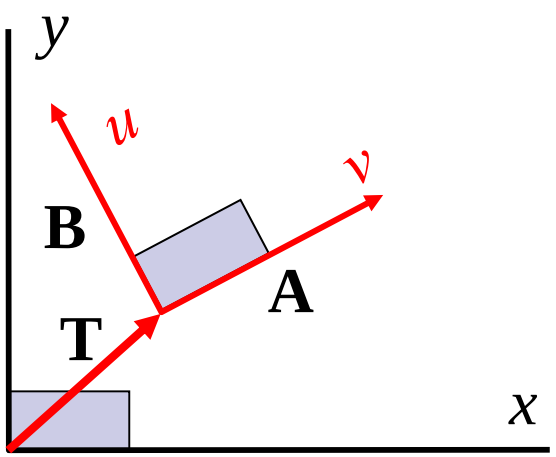
Rigid body transformation

$\mathbf{P}' = \mathbf{M}\mathbf{P}$, of

only rotation

translation

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} r_{xx} & r_{xy} & tr_x \\ r_{yx} & r_{yy} & tr_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix}$$



$\begin{pmatrix} r_{xx} & r_{xy} \\ r_{yx} & r_{yy} \end{pmatrix}$: orthonormal submatrix

$$\mathbf{A} = \begin{pmatrix} r_{xx} \\ r_{yx} \end{pmatrix} \text{ and } \mathbf{B} = \begin{pmatrix} r_{xy} \\ r_{yy} \end{pmatrix}, \ |\mathbf{A}| = 1, |\mathbf{B}| = 1, \mathbf{A} \cdot \mathbf{B} = 0$$

Other 2D transformations

- Reflection
- Shear

Can also be combined

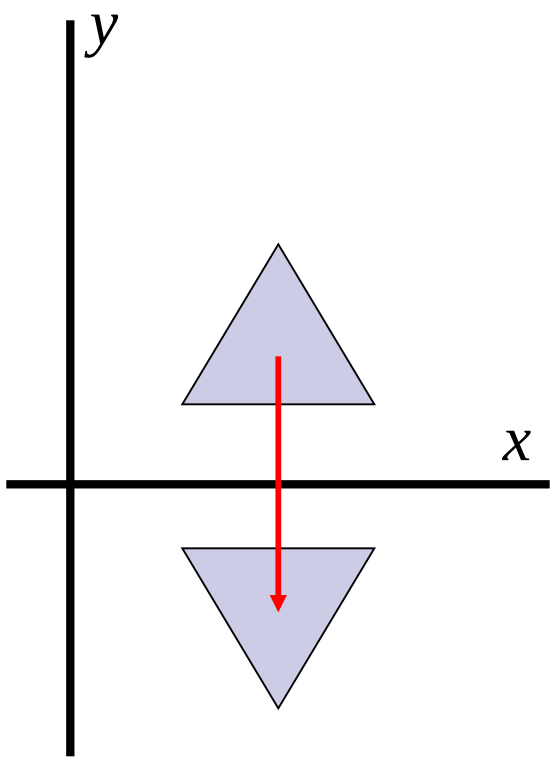
Reflection over axis

Reflext over *x*-axis:

$x' = x,\ \ y' = -y$

or

$$\mathbf{P}' = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \mathbf{P}$$



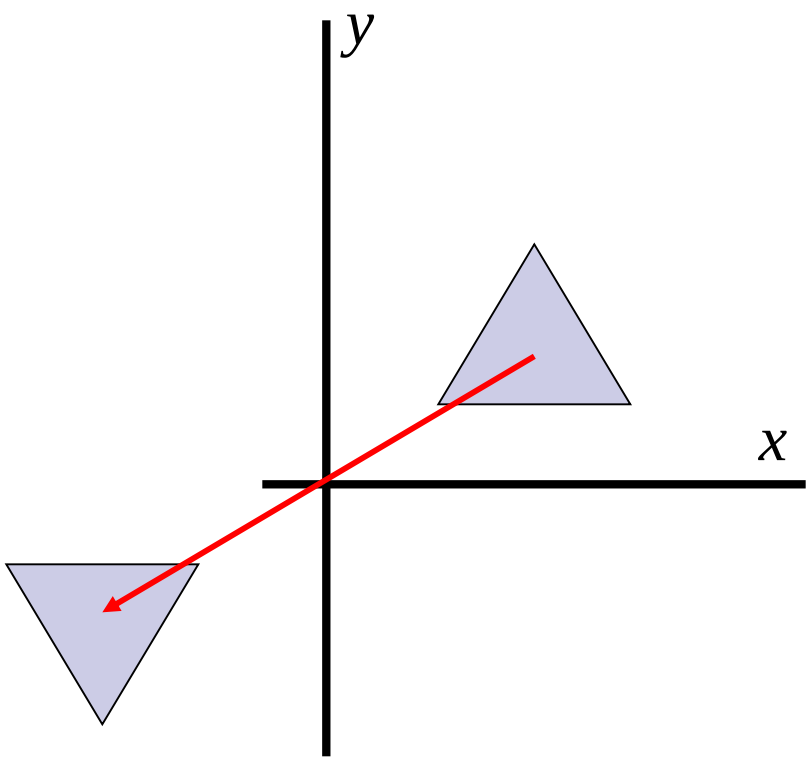
Reflect over origin

Reflect over origin:

$x' = -x, \quad y' = -y$

or

$$\mathbf{P}' = \begin{pmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \mathbf{P}$$



Same as $\mathbf{P}' = \mathbf{R}(180)\mathbf{P}$

Shear

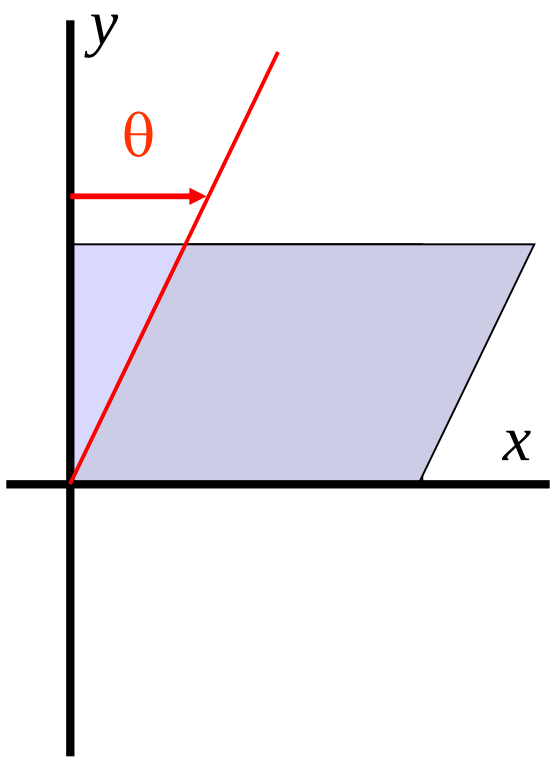
Shear the *y*-as:

$x'=x+fy,\quad y'=y$

or

$$\mathbf{P'}=\begin{pmatrix}1 & f & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{pmatrix}\mathbf{P}$$

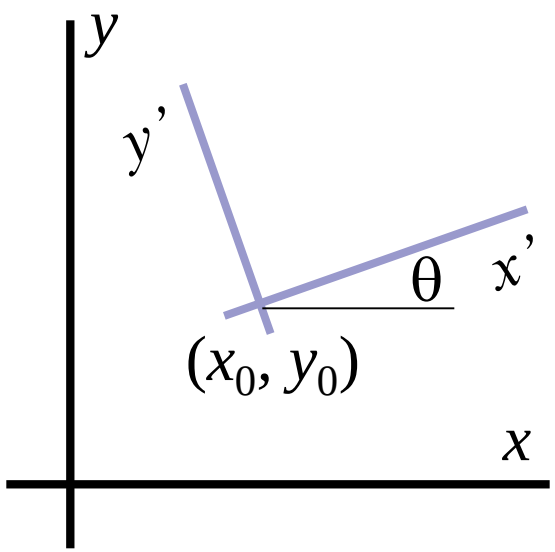
with $f=\tan\theta$



Transformations coordinates

Given (*x*,*y*)-coordinates,
Find (*x'*,*y'*)-coordinates.

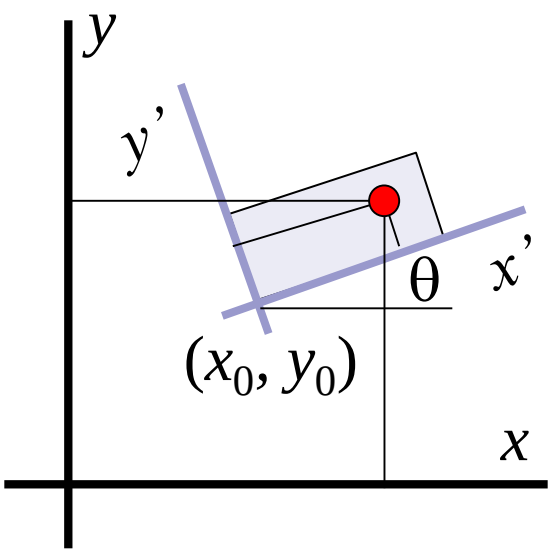
Reverse route as
object transformaties.



Transformations coordinates

Given (*x*,*y*)-coordinates,
Find (*x'*,*y'*)-coordinates.

Example: user points at (*x*,*y*), what's the position in local coordinates?



Transformations coordinates

Given **X**: (*x*,*y*)-coordinates,

Find **X'**: (*x'*,*y'*)-coordinates.

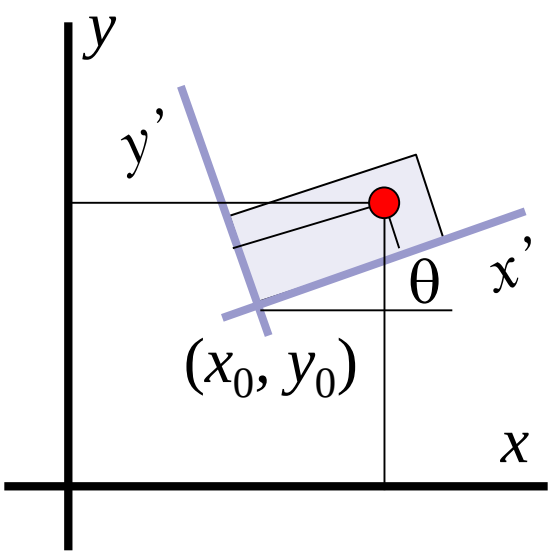
Standard:

X=MX' (object trafo:

from local to global)

Here:

X'=M^{-1}X (from global to local)



Transformations coordinates

Given **X**: (*x*,*y*)-coordinates,

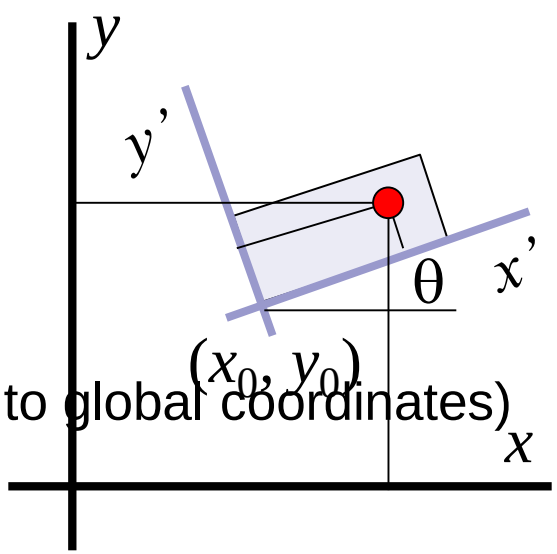
Find **X'**: (*x'*,*y'*)-coordinates.

Here:

X'=M^{-1}X (from global to local)

Approach 1:

- Determine "standard matrix" **M** (from local to global coordinates) and invert



Transformations coordinates

Given **X**: (*x*,*y*)-coordinates,

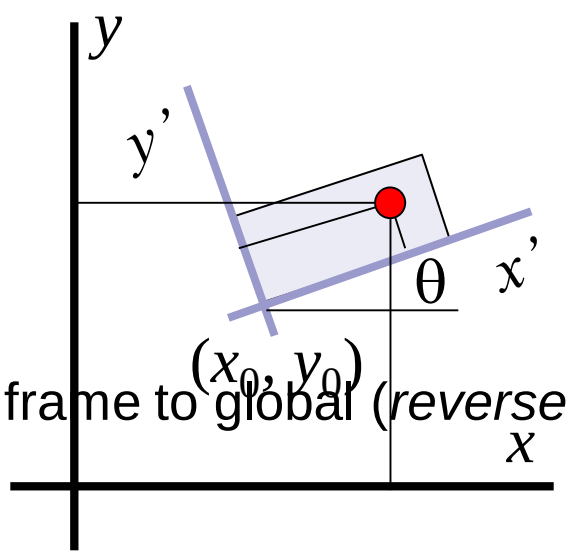
Find **X'**: (*x'*,*y'*)-coordinates.

Here:

X'=M^{-1}X (from global to local)

Approach 2:

- construct transformation that maps local frame to global (*reverse of usual*).



Transformations coordinates

Given **X**: (*x*,*y*)-coordinates,

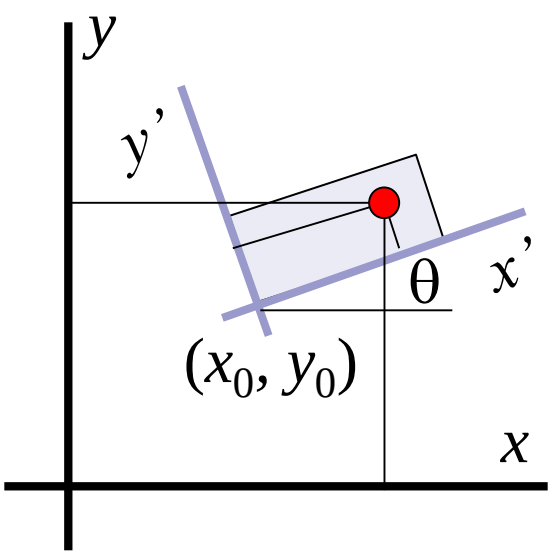
Find **X'**: (*x'*,*y'*)-coordinates.

Here:

X'=M^{-1}X (from global to local)

Approach 2:

1. Translate (x_0, y_0) to origin;
2. Rotate *x'*-axis to *x*-axis.



Transformations coordinates

Given **X**: (*x*,*y*)-coordinates,

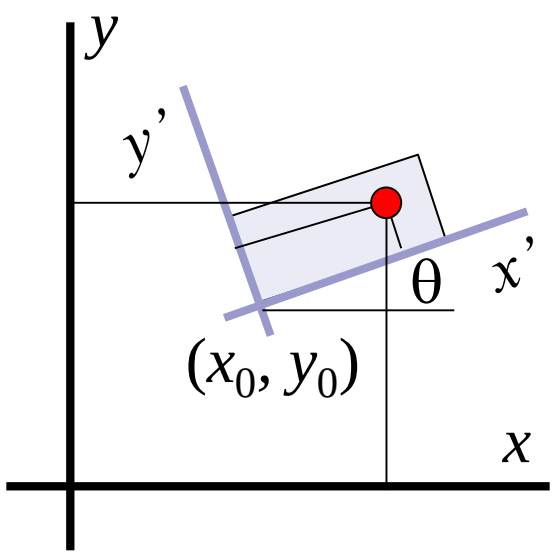
Find **X'**: (*x'*,*y'*)-coordinates.

Here:

X'=M^{-1}X (from global to local)

Approach 2:

M^{-1} = T$(-x_0, -y_0)$ **R**$(-\theta)$



OpenGL 2D transformations 1

Internally:

- Coordinates are four-element row vectors
- Transformations are 4×4 matrices

2D trafo's: Ignore z-coordinates, set $z = 0$.

OpenGL 2D transformations 2

OpenGL maintains two matrices:

- **`GL_PROJECTION`**
- **`GL_MODELVIEW`**

Transformations are applied to the current matrix, to be selected with:

- **`glMatrixMode(GL_PROJECTION)`** or
- **`glMatrixMode(GL_MODELVIEW)`**

OpenGL 2D transformations 3

Initializing the matrix to **I**:

- `glLoadIdentity();`

Replace the matrix with **M**:

- `GLfloat M[16]; fill(M);`
- `glLoadMatrix*(M);`

Matrices are specified in *column-major* order:

$$\mathbf{M} = \begin{bmatrix} m[0] & m[4] & m[8] & m[12] \\ m[1] & m[5] & m[9] & m[13] \\ m[2] & m[6] & m[10] & m[14] \\ m[3] & m[7] & m[11] & m[15] \end{bmatrix}$$

Multiply current matrix with **M**:

- `glMultMatrix*(M);`

OpenGL 2D transformations 4

Basic transformation functions: generate matrix and post-multiply this with current matrix.

Translate over [tx, ty, tz]:

```
glTranslate*(tx, ty, tz);
```

Rotate over theta degrees (!) around axis [vx, vy, vz]:

```
glRotate*(theta, vx, vy, vz);
```

Scale axes with factors sx, sy, sz:

```
glScale*(sx, sy, sz);
```

OpenGL 2D Transformations 5

OpenGL maintains *stacks* of transformation matrices.

Two operations:

- **`glPushMatrix():`**

  Make copy of current matrix and put that on top of the stack;

- **`glPopMatrix():`**

  Remove top element of the stack.

Handy for dealing with hierarchical models

Handy for “undoing” transformations

OpenGL 2D Transformations 6

Standard:

```
glRotate(10, 1, 2, 0);
glScale(2, 1, 0.5);
glTranslate(1, 2, 3);

glutWireCube(1);

glTranslate(–1, –2, –3);
glScale(0.5, 1, 2);
glRotate(–10, 1, 2, 0);
```

Undo transformation

Using the stack:

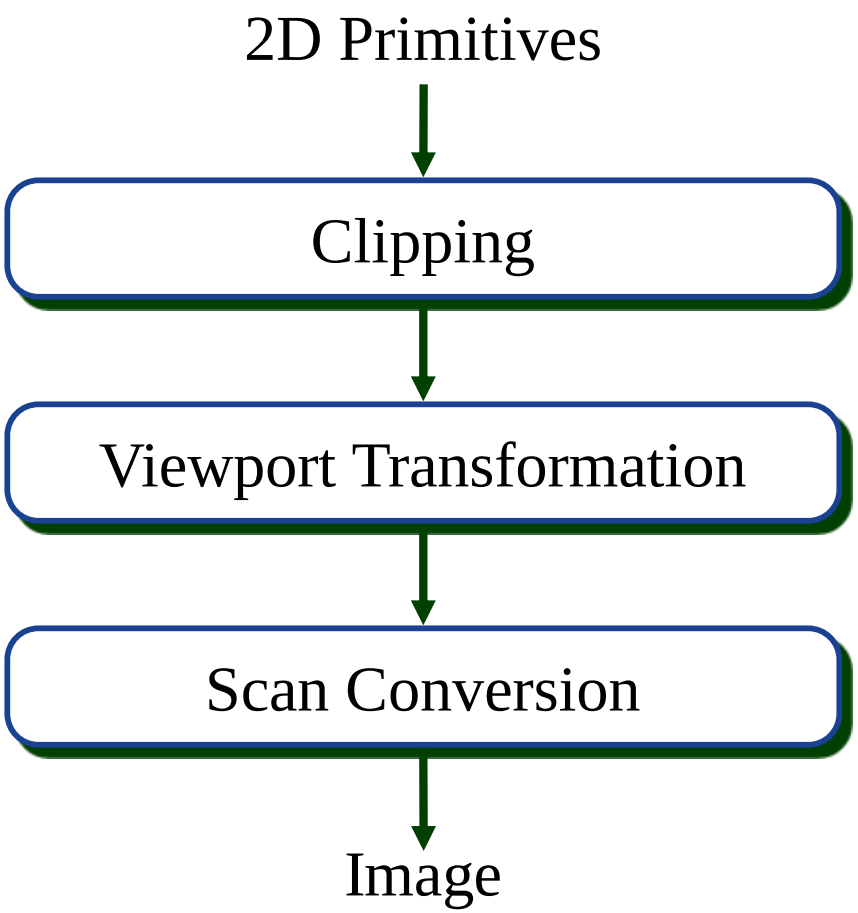
```
glPushMatrix();
glRotate(10, 1, 2, 0);
glScale(2, 1, 0.5);
glTranslate(1, 2, 3);

glutWireCube(1);

glPopMatrix();
```

Shorter, more robust

2D transformations summarized

- Transformations: modeling, viewing, animation;
- Several kinds of transformations;
- Homogeneous coordinates;
- Combine transformations using matrix multiplication.

2D Rendering Pipeline

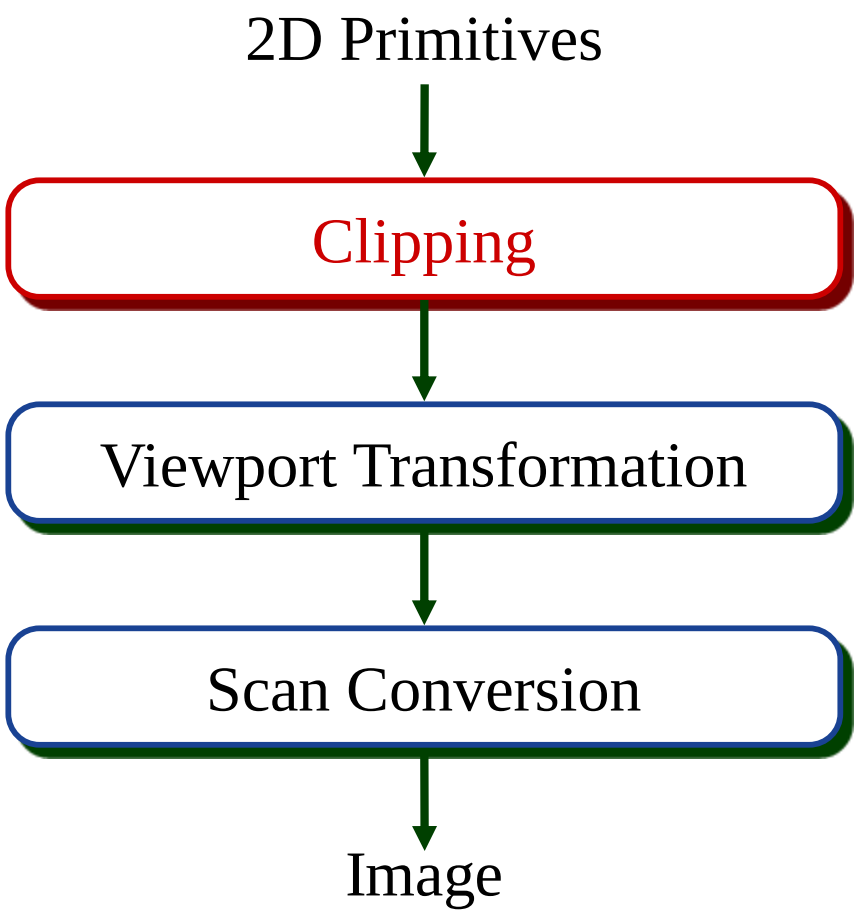


Clip portions of geometric primitives residing outside window

Transform the clipped primitives from screen to image coordinates

Fill pixel representing primitives in screen coordinates

2D Rendering Pipeline



Clip portions of geometric primitives residing outside window

Transform the clipped primitives from screen to image coordinates

Fill pixel representing primitives in screen coordinates

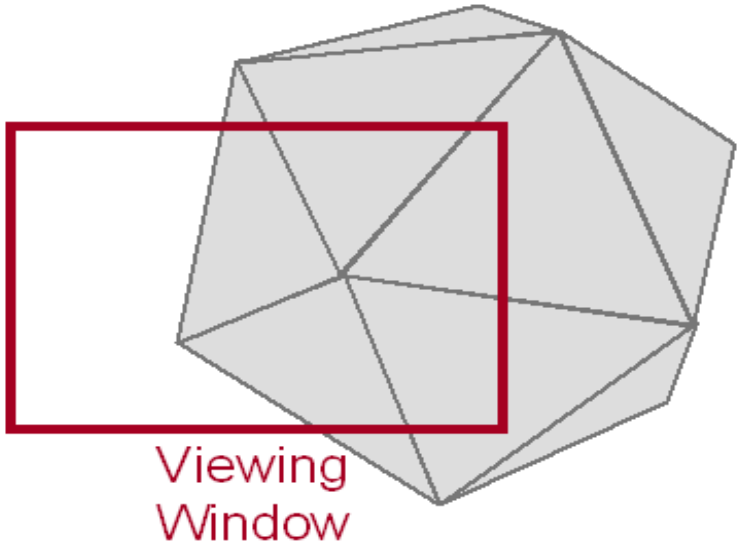
Clipping

- Avoid Drawing Parts of Primitives Outside Window
  - Window defines part of scene being viewed
  - Must draw geometric primitives only inside window

World Coordinates

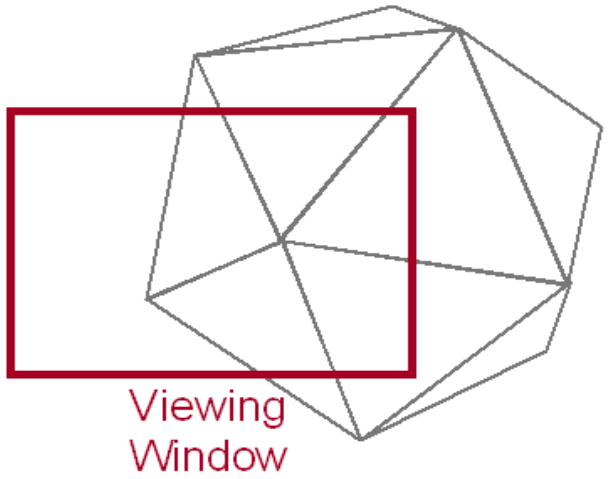
Clipping

- Avoid Drawing Parts of Primitives Outside Window
  - Window defines part of scene being viewed
  - Must draw geometric primitives only inside window



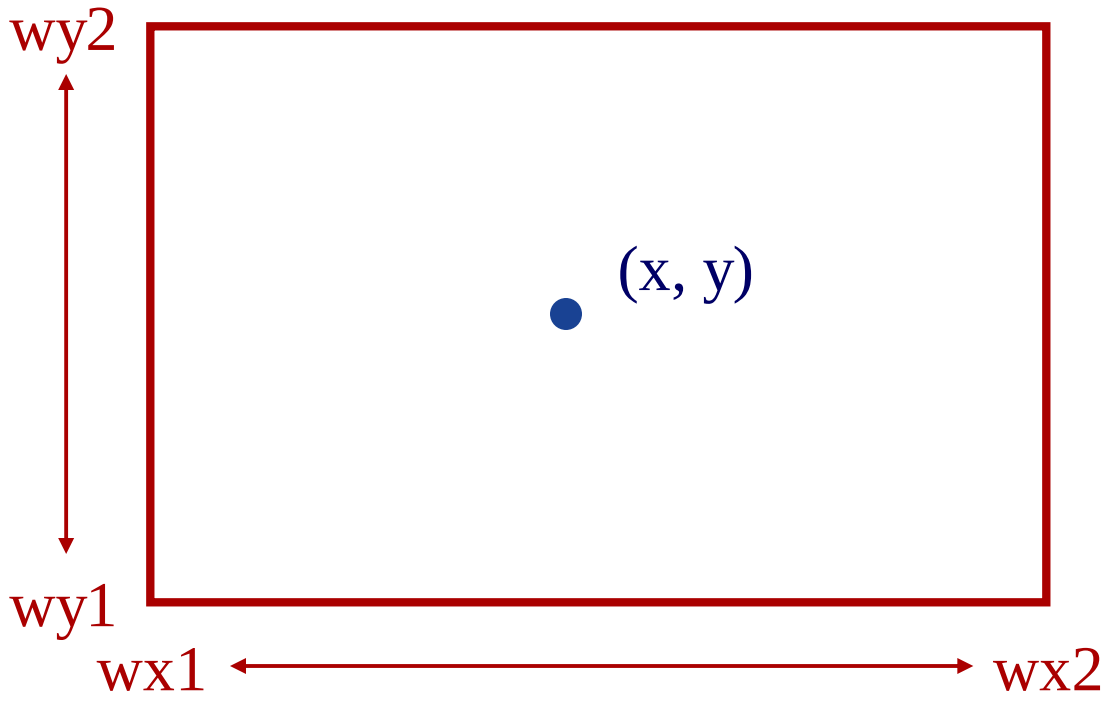
Clipping

- Avoid Drawing Parts of Primitives Outside Window
  - Points
  - Lines
  - Polygons
  - Circles
  - etc.



Point Clipping

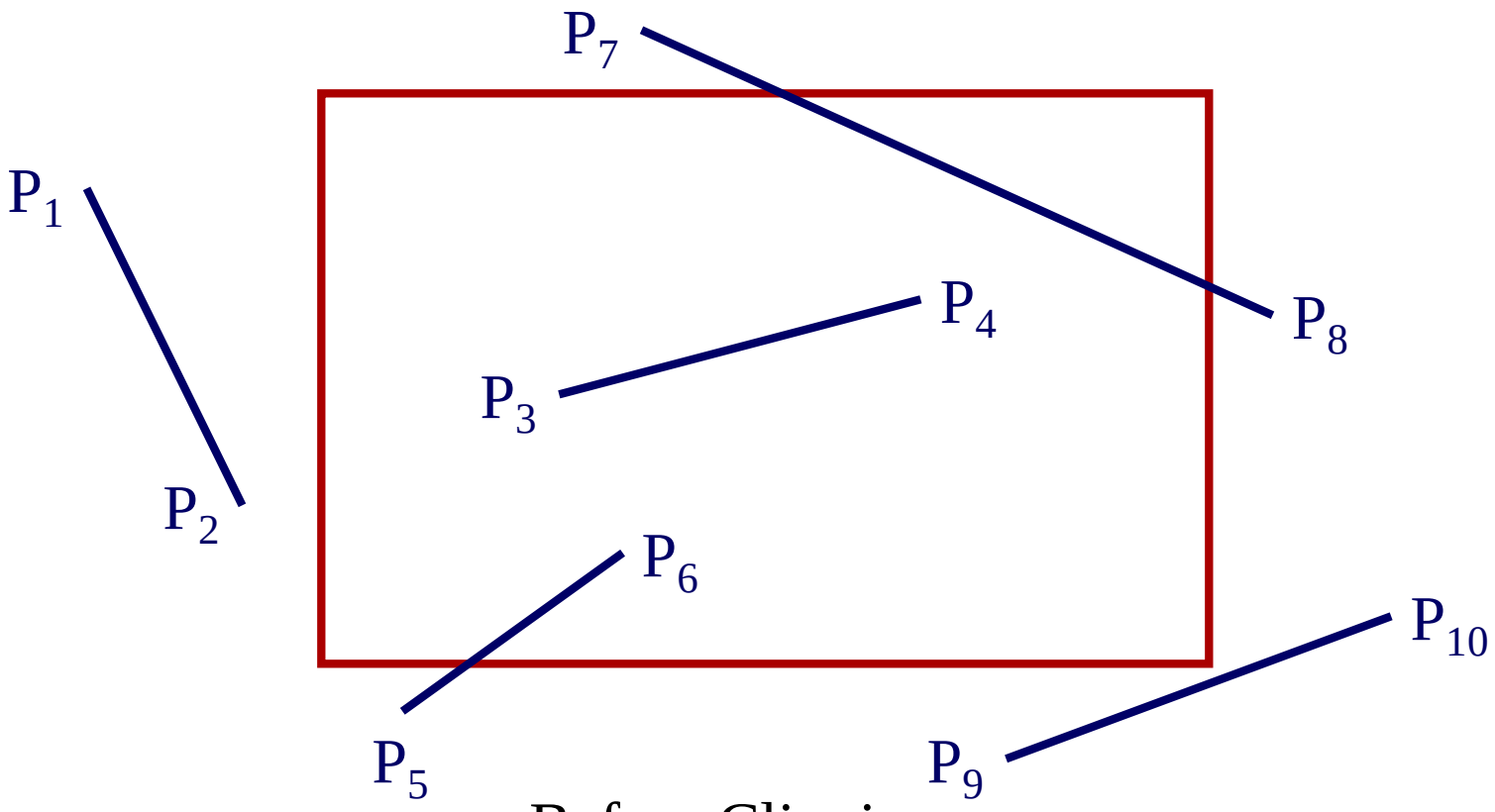
- Is Point(x,y) Inside the Clip Window?




```
Inside =
     (x>=wx1) &&
     (x<=wx2) &&
     (y>=wy1) &&
     (y<=wy2);
```

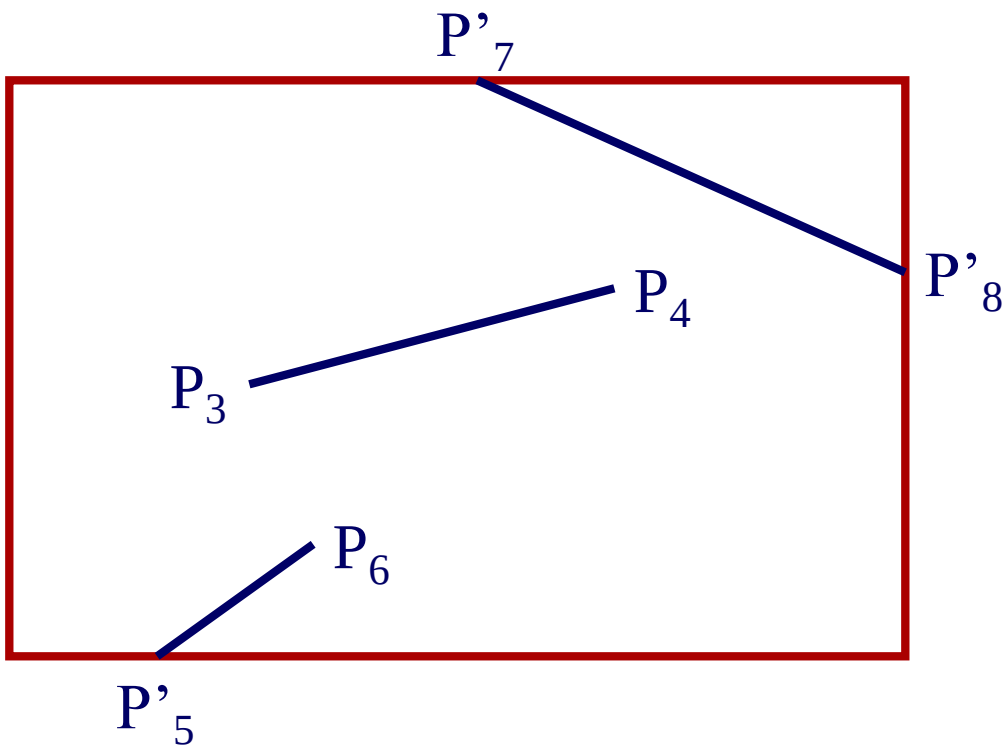
Line Clipping

- Find the Part of a Line Inside the Clip Window



Line Clipping

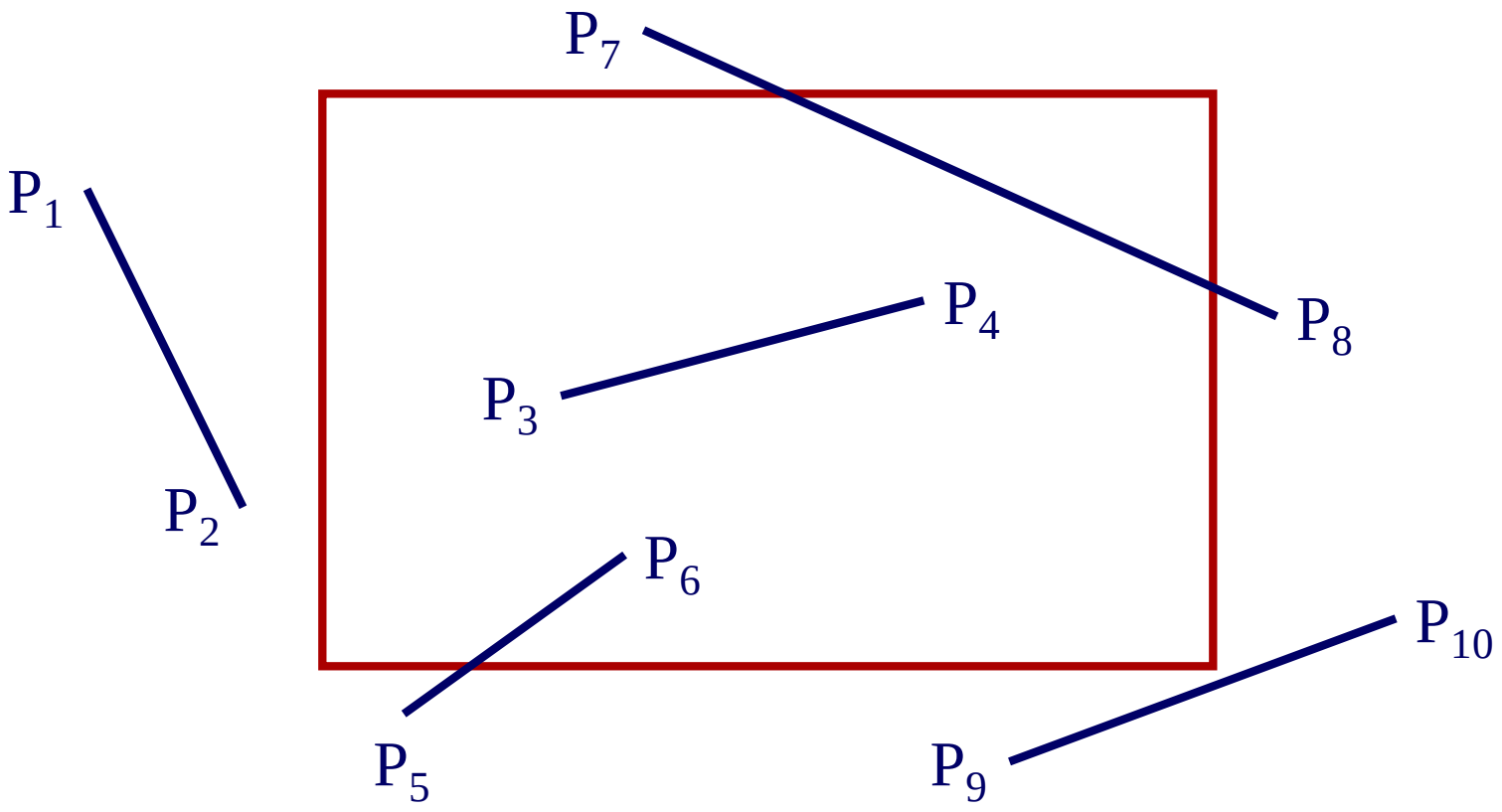
- Find the Part of a Line Inside the Clip Window



After Clipping

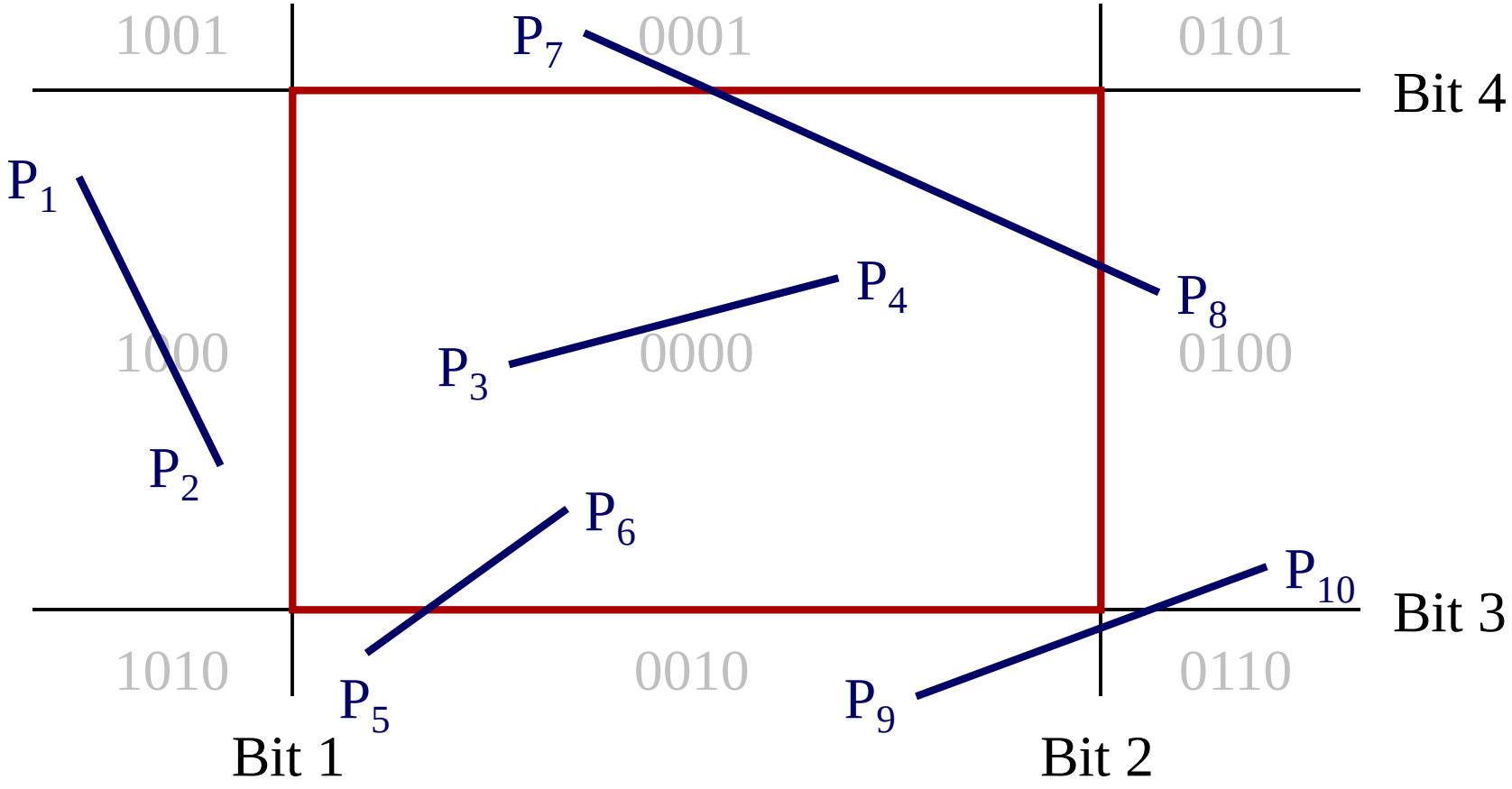
Cohen-Sutherland Line Clipping

- Use Simple Tests to Classify Easy Cases First



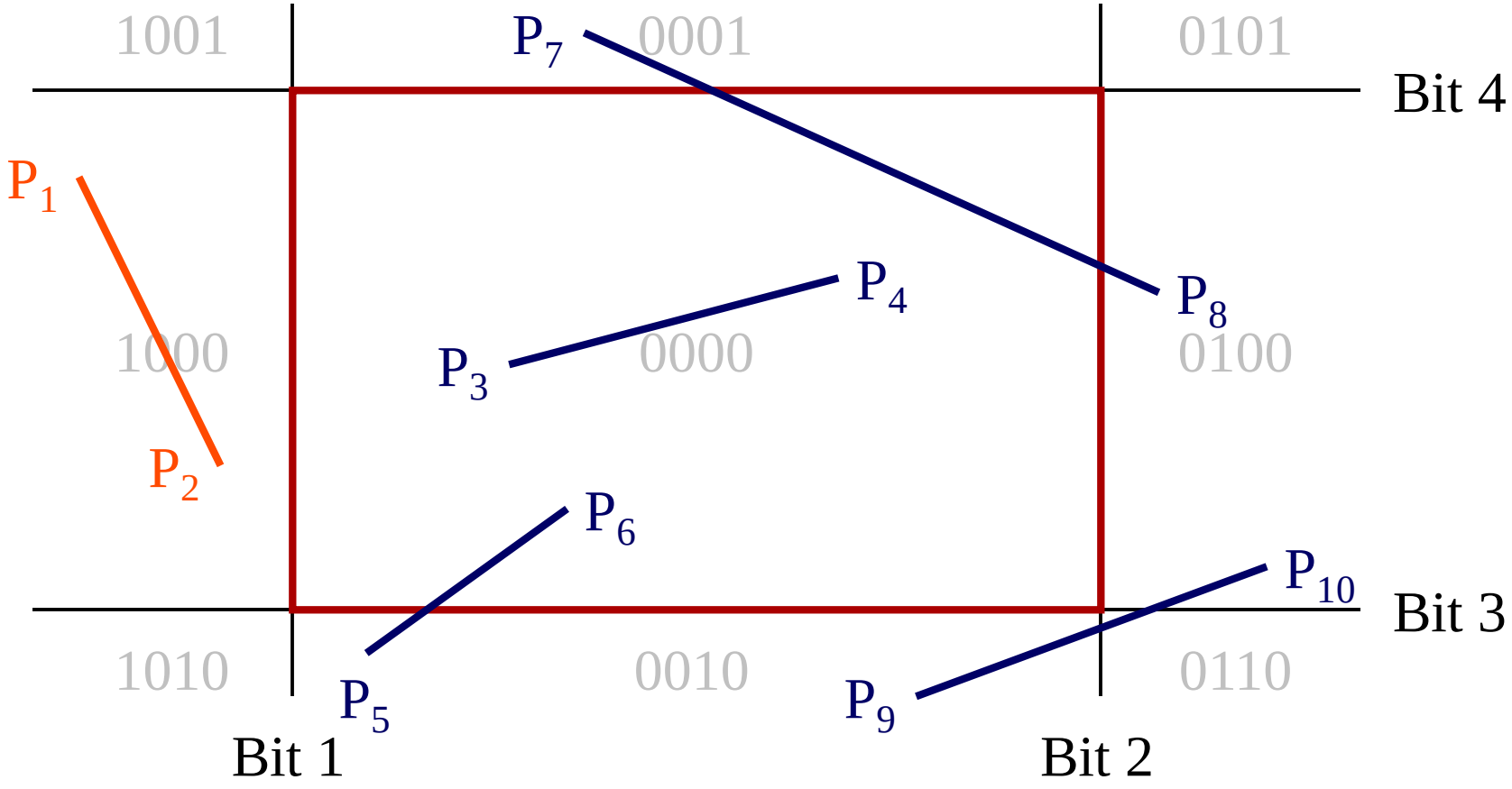
Cohen-Sutherland Line Clipping

- Classify Some Lines Quickly by AND of Bit Codes Representing Regions of Two Endpoints (Must Be 0)



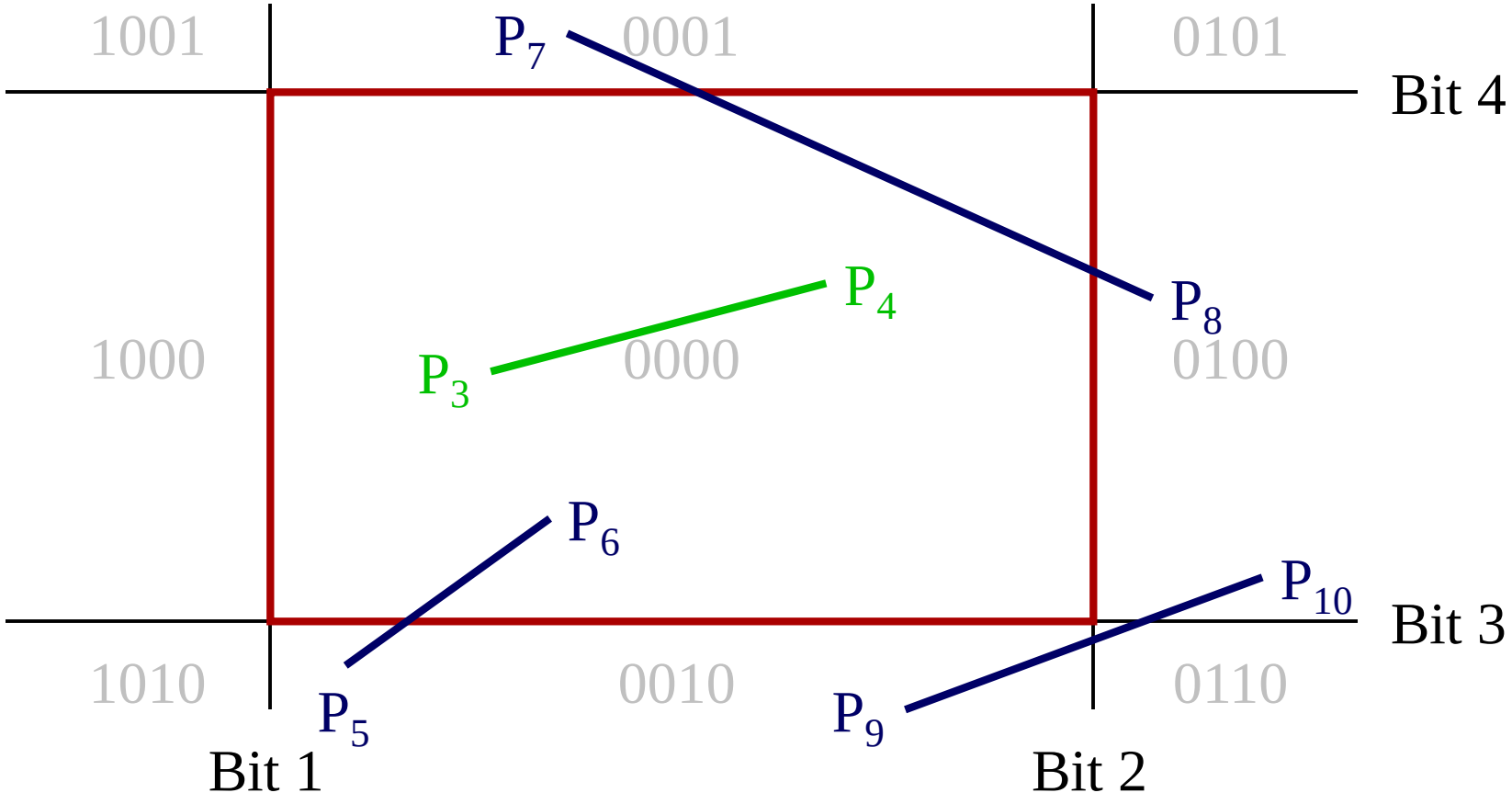
Cohen-Sutherland Line Clipping

- Classify Some Lines Quickly by AND of Bit Codes Representing Regions of Two Endpoints (Must Be 0)



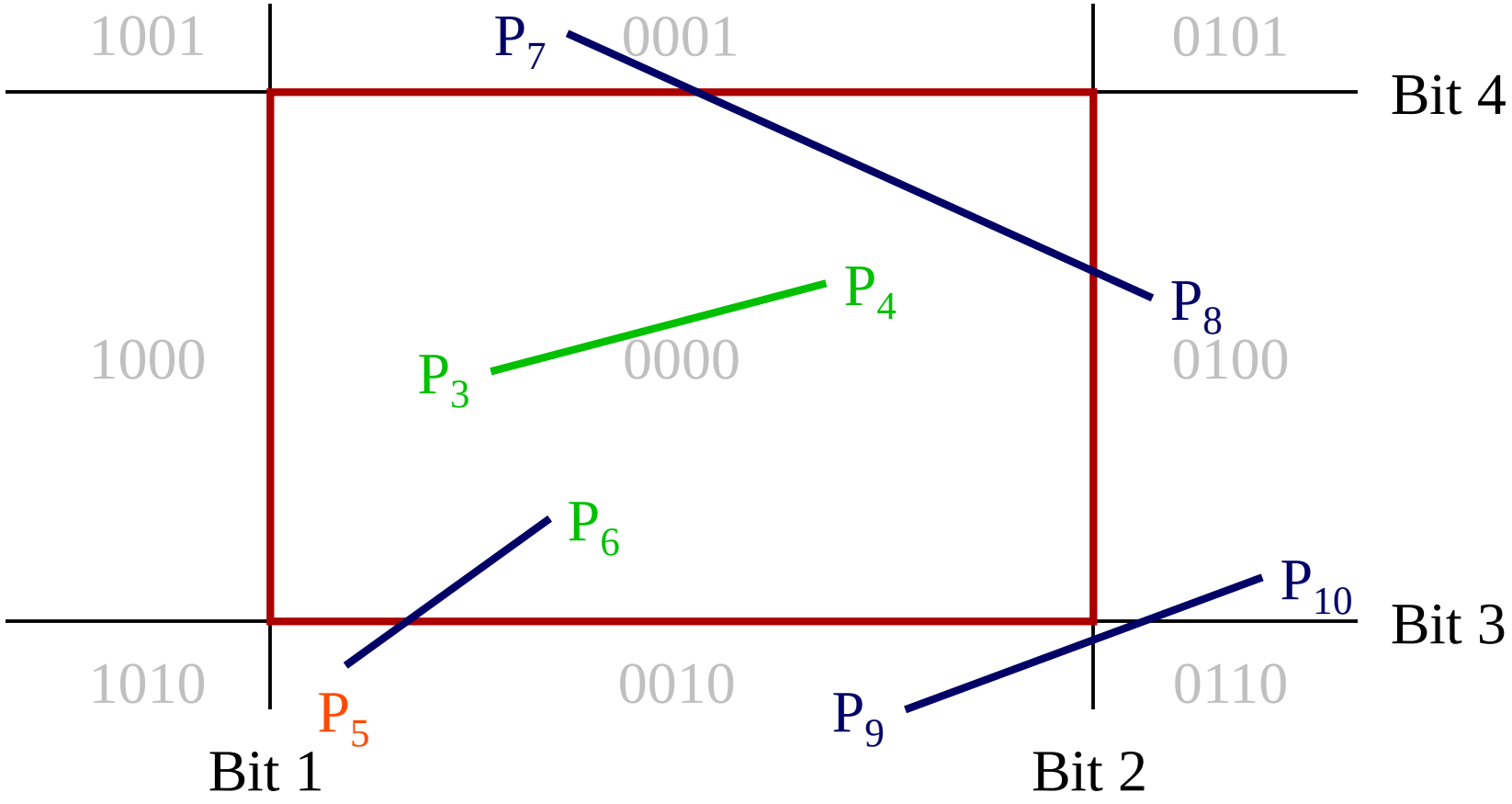
Cohen-Sutherland Line Clipping

- Classify Some Lines Quickly by AND of Bit Codes Representing Regions of Two Endpoints (Must Be 0)



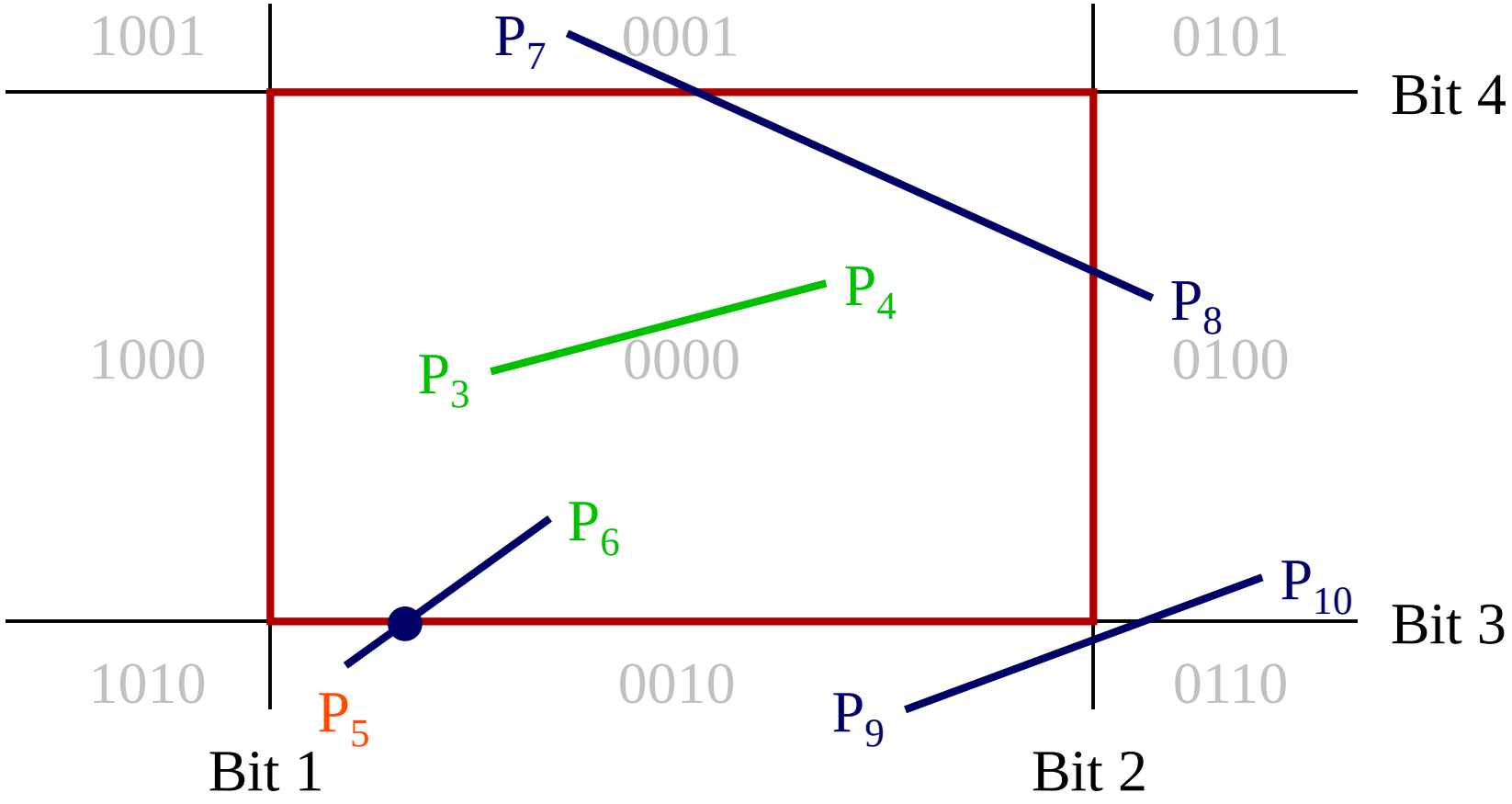
Cohen-Sutherland Line Clipping

- Compute Intersections with Window Boundary for Lines That Can't be Classified Quickly



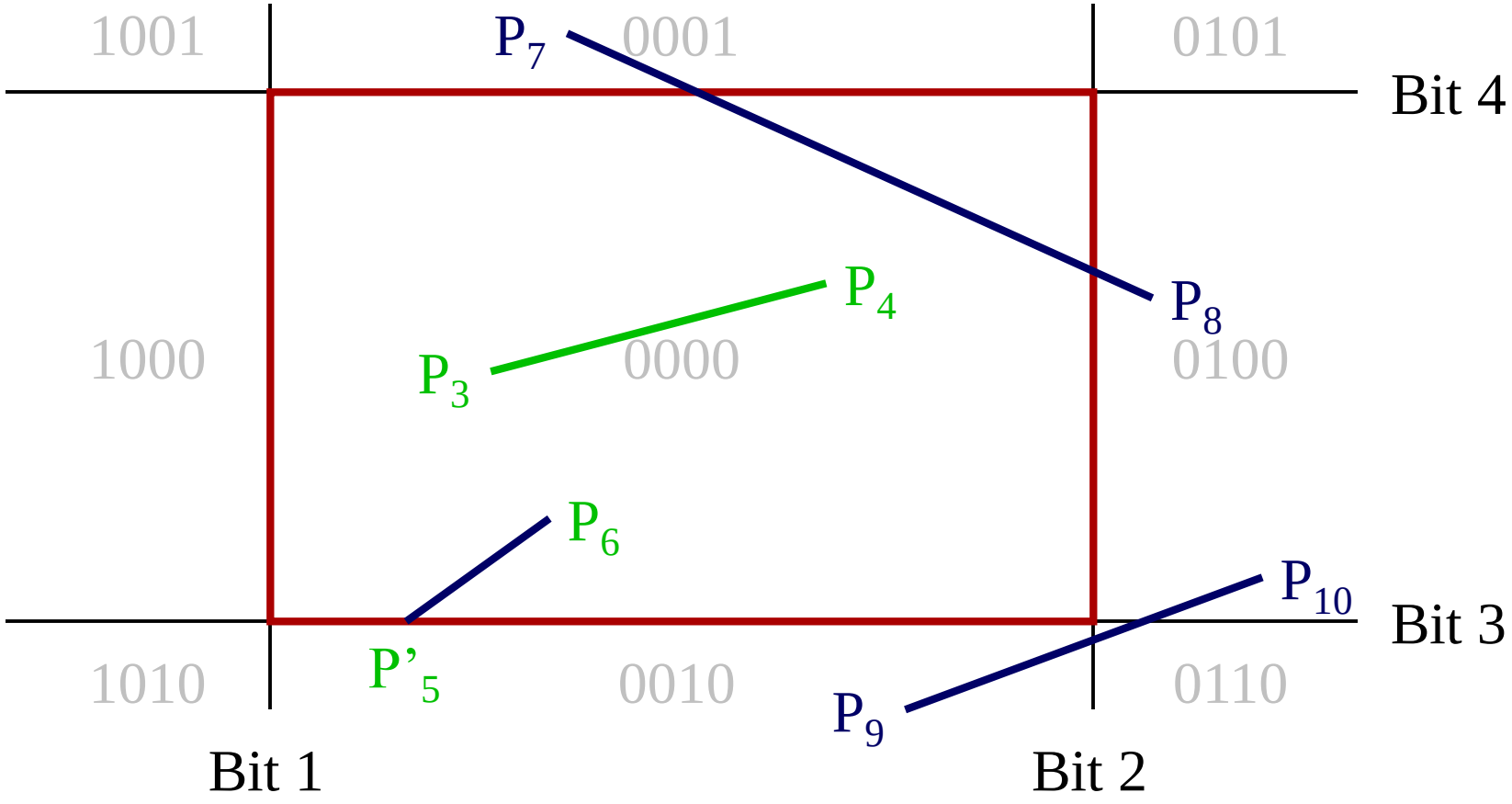
Cohen-Sutherland Line Clipping

- Compute Intersections with Window Boundary for Lines That Can't be Classified Quickly



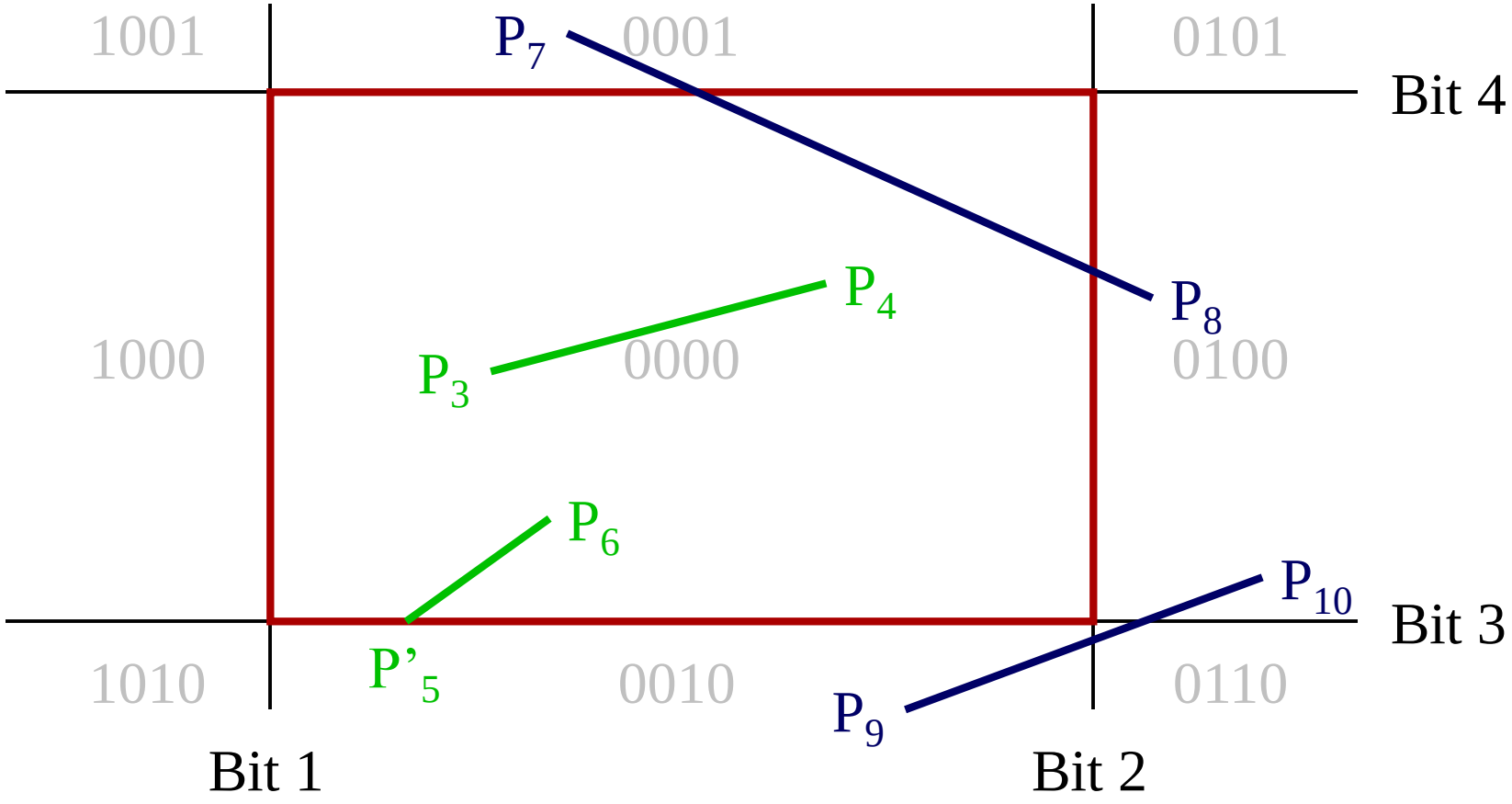
Cohen-Sutherland Line Clipping

- Compute Intersections with Window Boundary for Lines That Can't be Classified Quickly



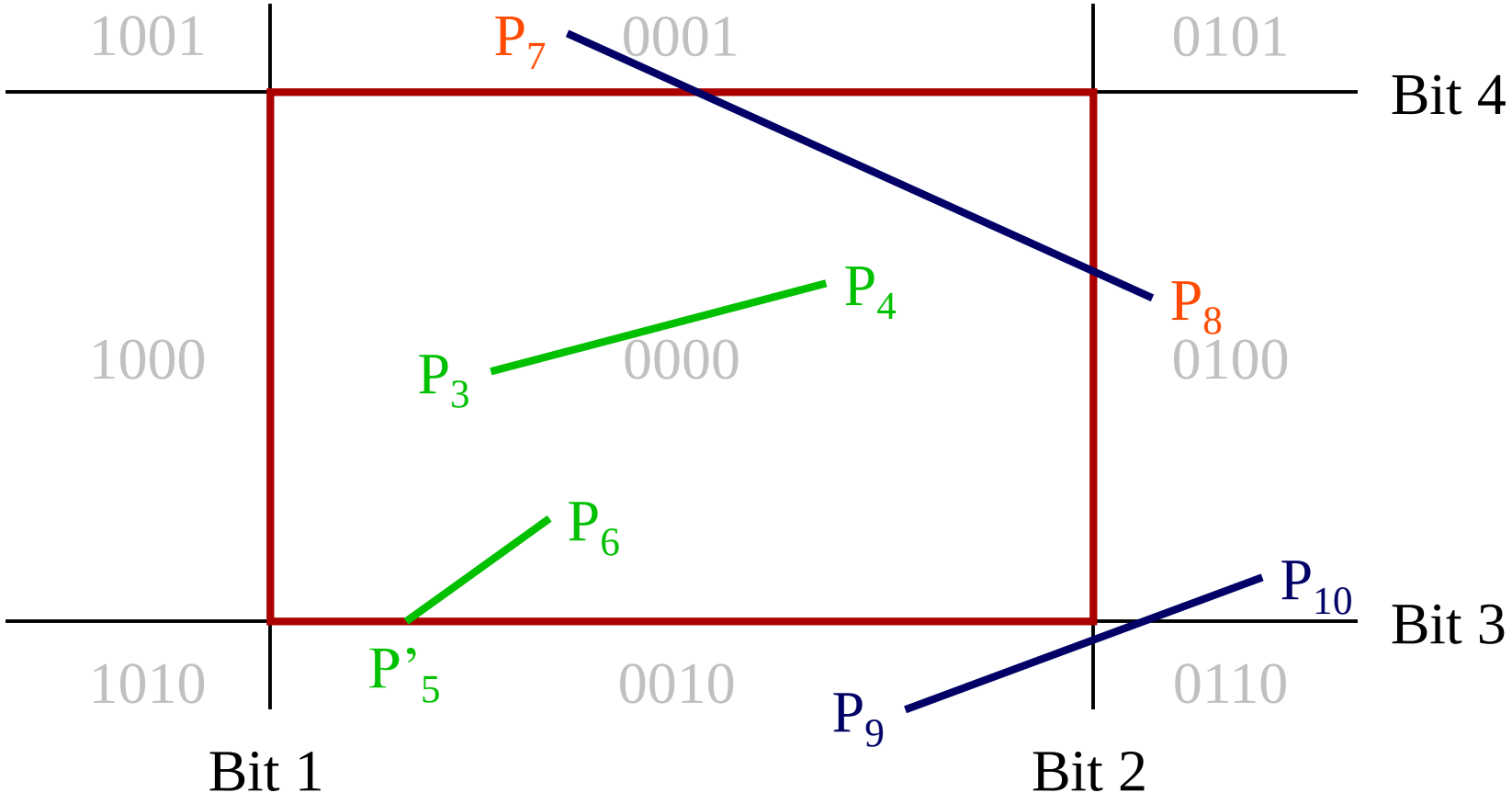
Cohen-Sutherland Line Clipping

- Compute Intersections with Window Boundary for Lines That Can't be Classified Quickly



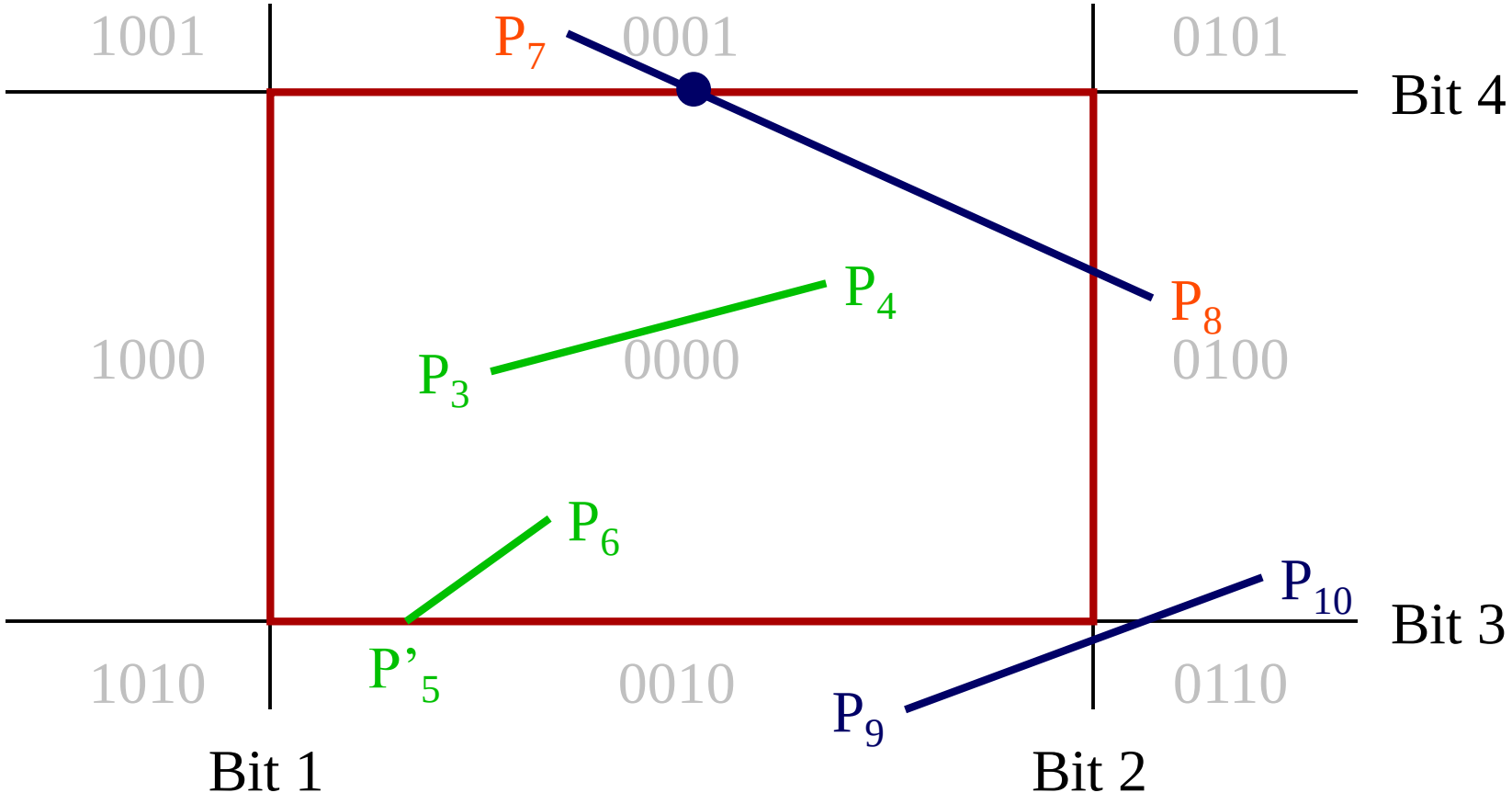
Cohen-Sutherland Line Clipping

- Compute Intersections with Window Boundary for Lines That Can't be Classified Quickly



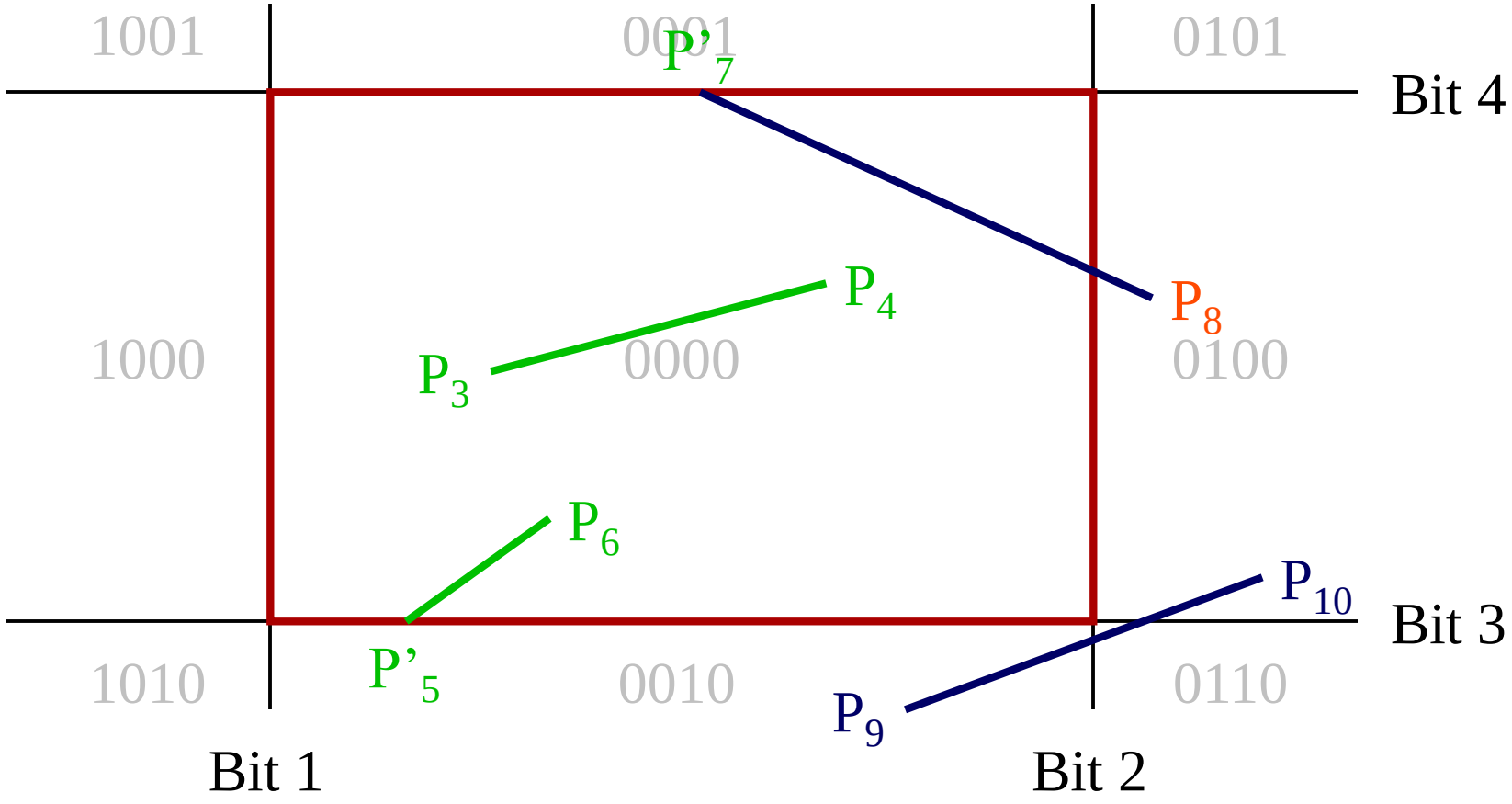
Cohen-Sutherland Line Clipping

- Compute Intersections with Window Boundary for Lines That Can't be Classified Quickly



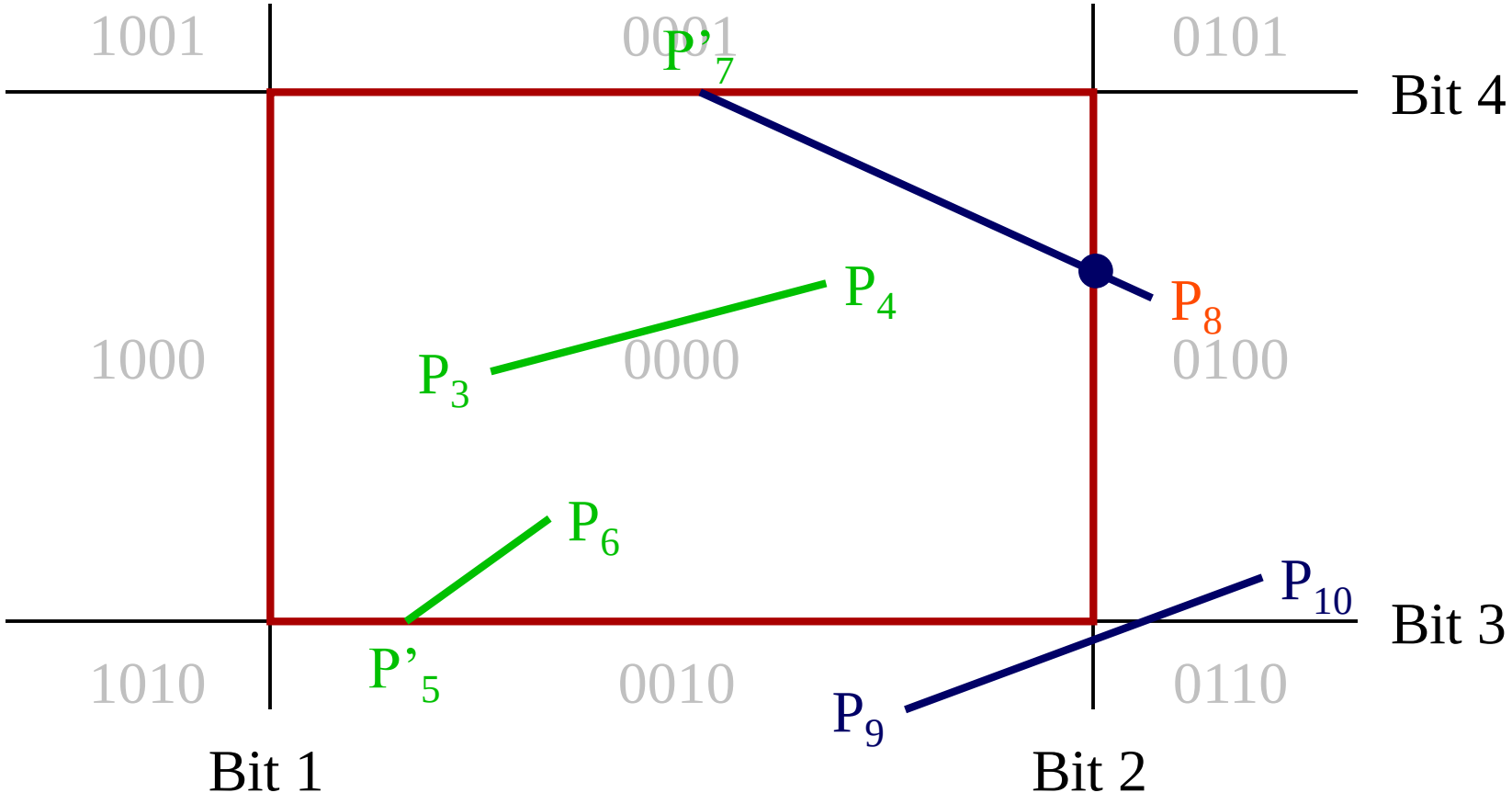
Cohen-Sutherland Line Clipping

- Compute Intersections with Window Boundary for Lines That Can't be Classified Quickly



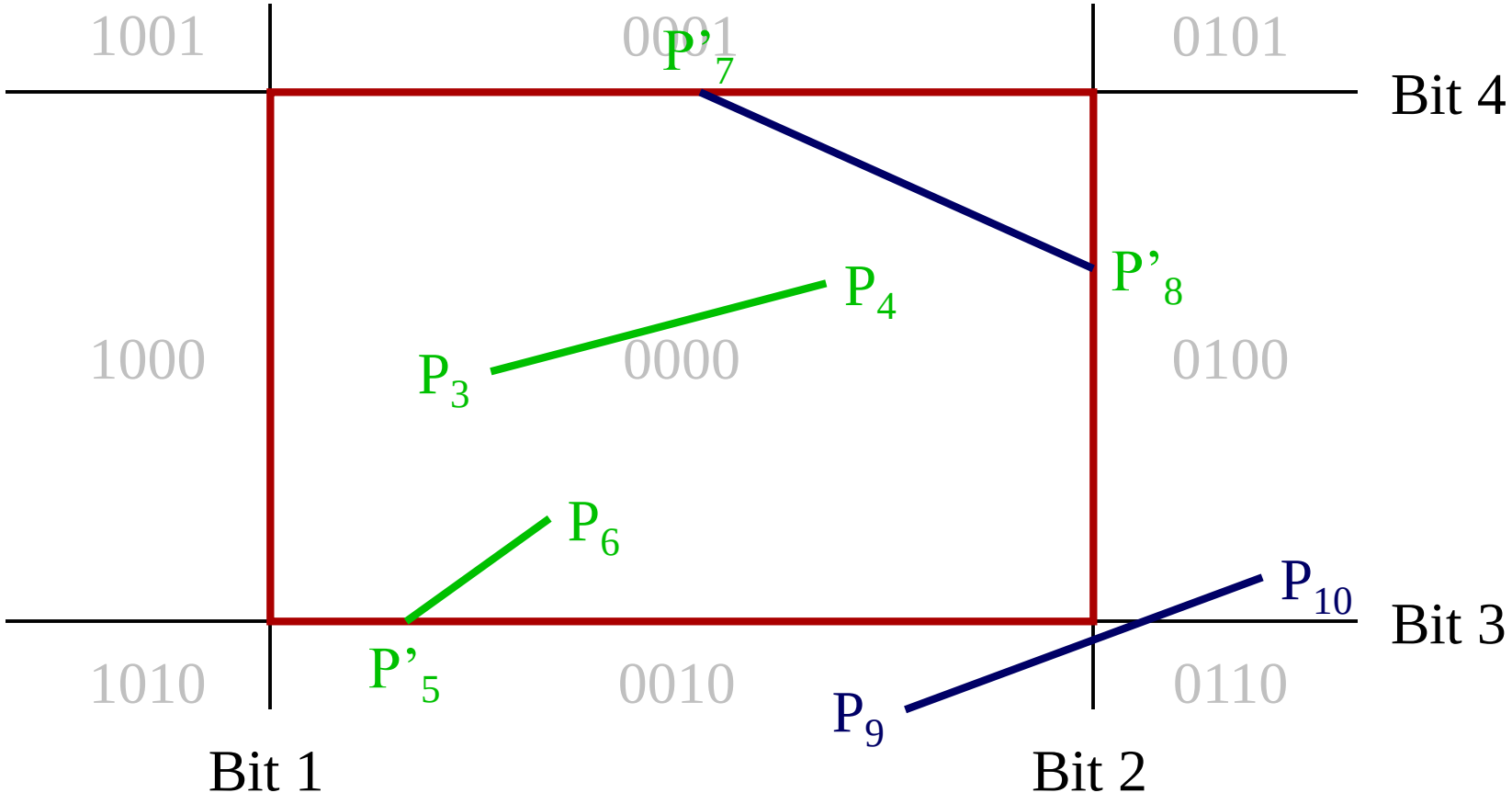
Cohen-Sutherland Line Clipping

- Compute Intersections with Window Boundary for Lines That Can't be Classified Quickly



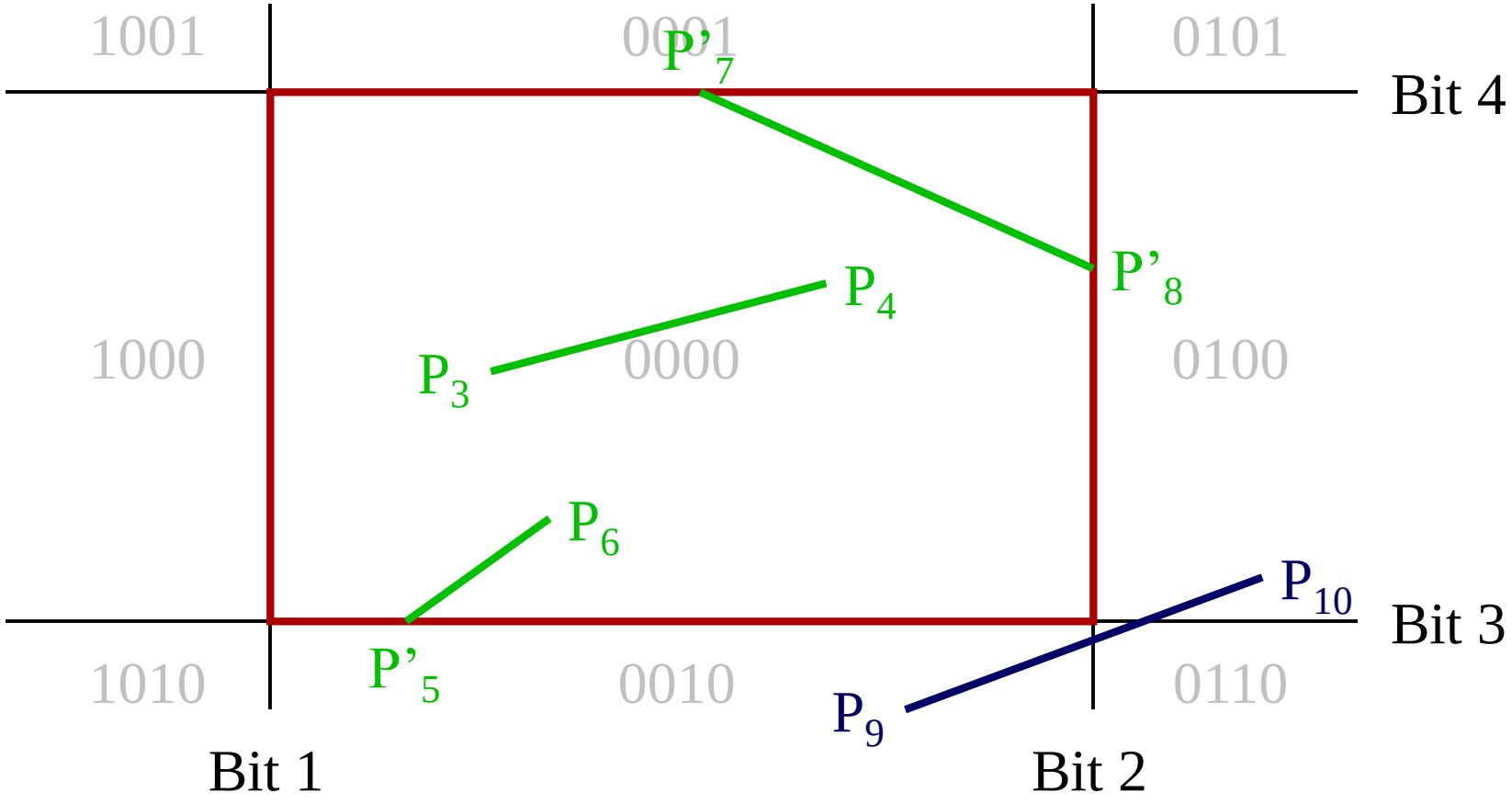
Cohen-Sutherland Line Clipping

- Compute Intersections with Window Boundary for Lines That Can't be Classified Quickly



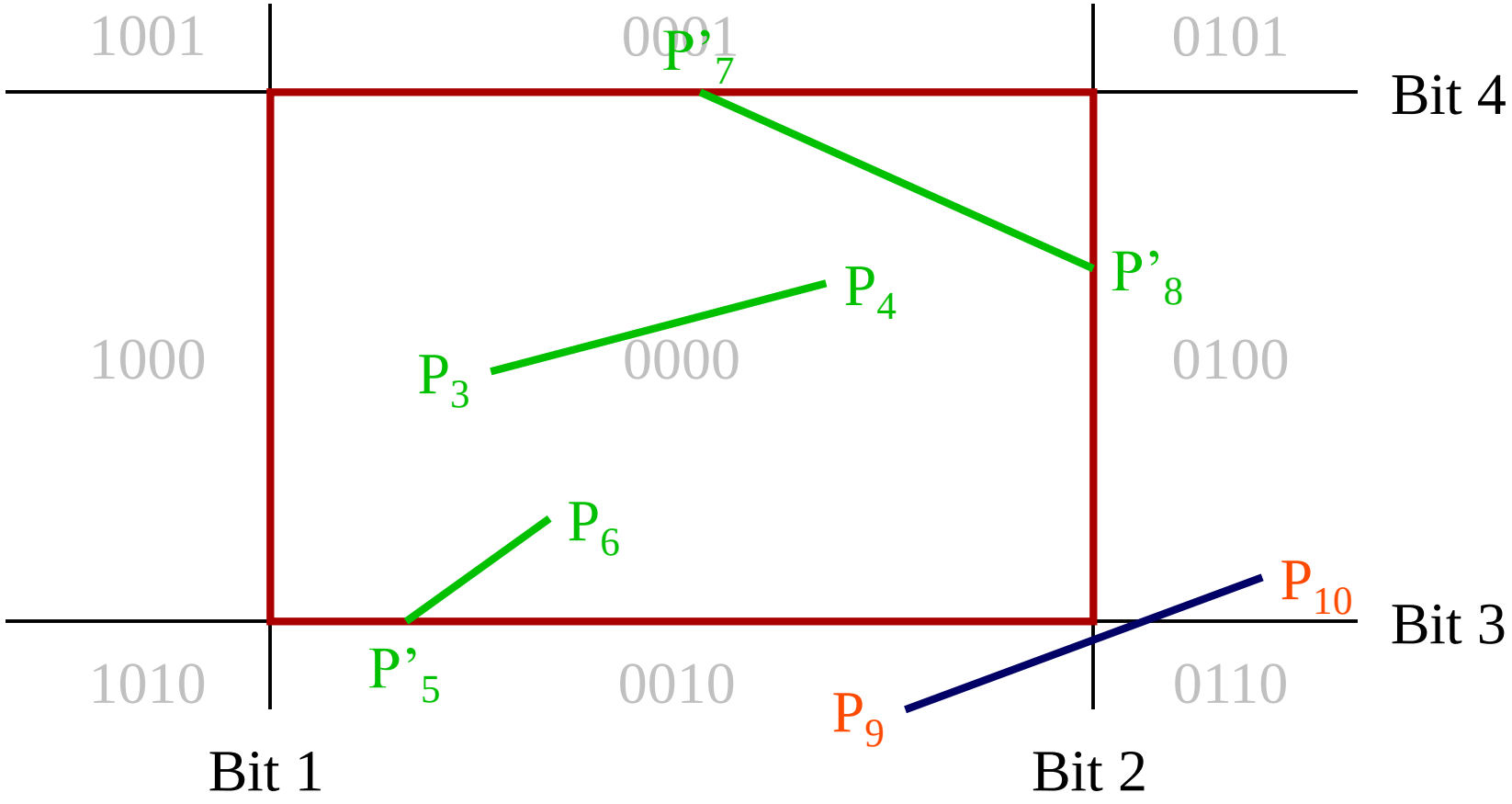
Cohen-Sutherland Line Clipping

- Compute Intersections with Window Boundary for Lines That Can't be Classified Quickly



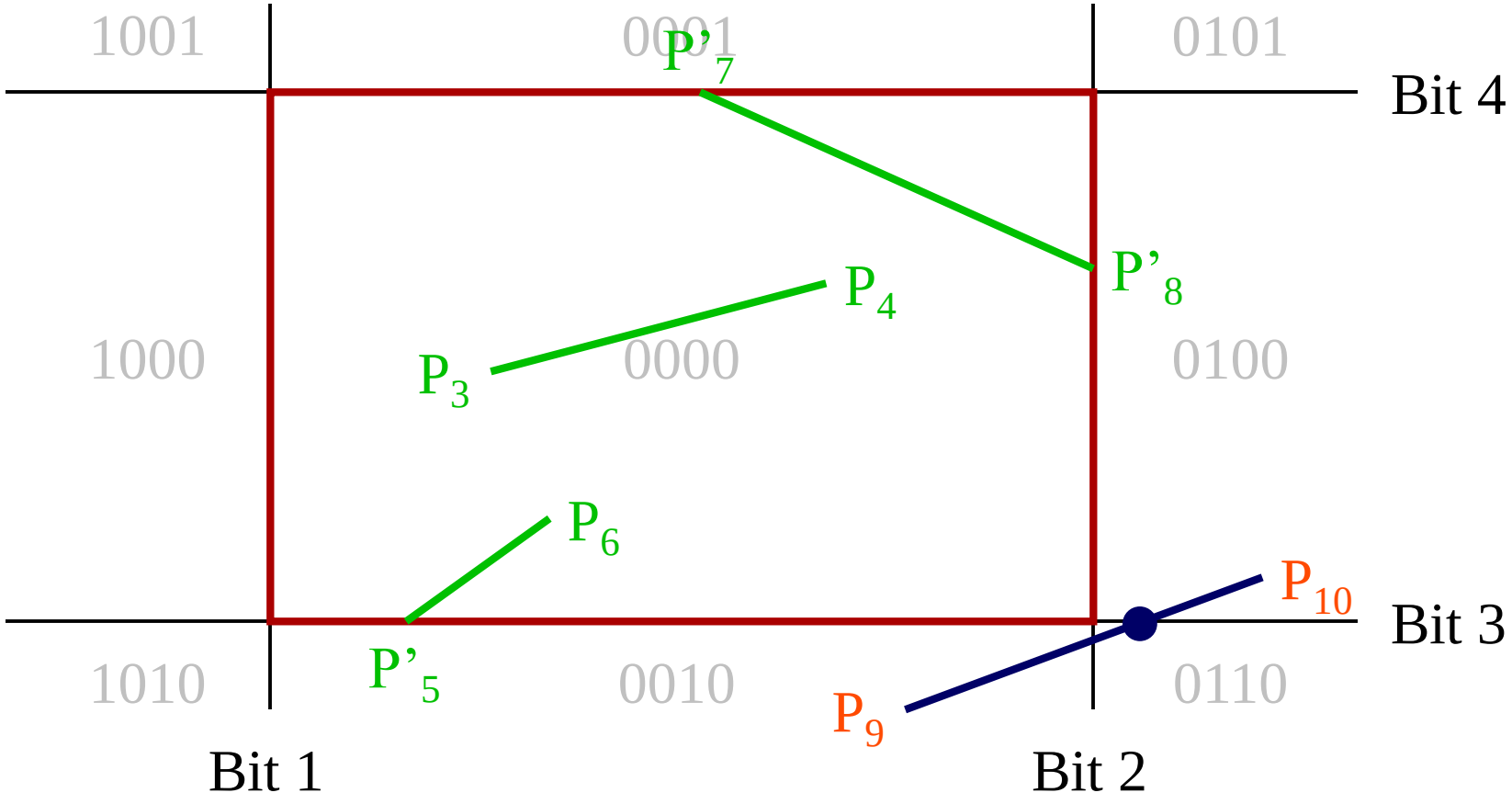
Cohen-Sutherland Line Clipping

- Compute Intersections with Window Boundary for Lines That Can't be Classified Quickly



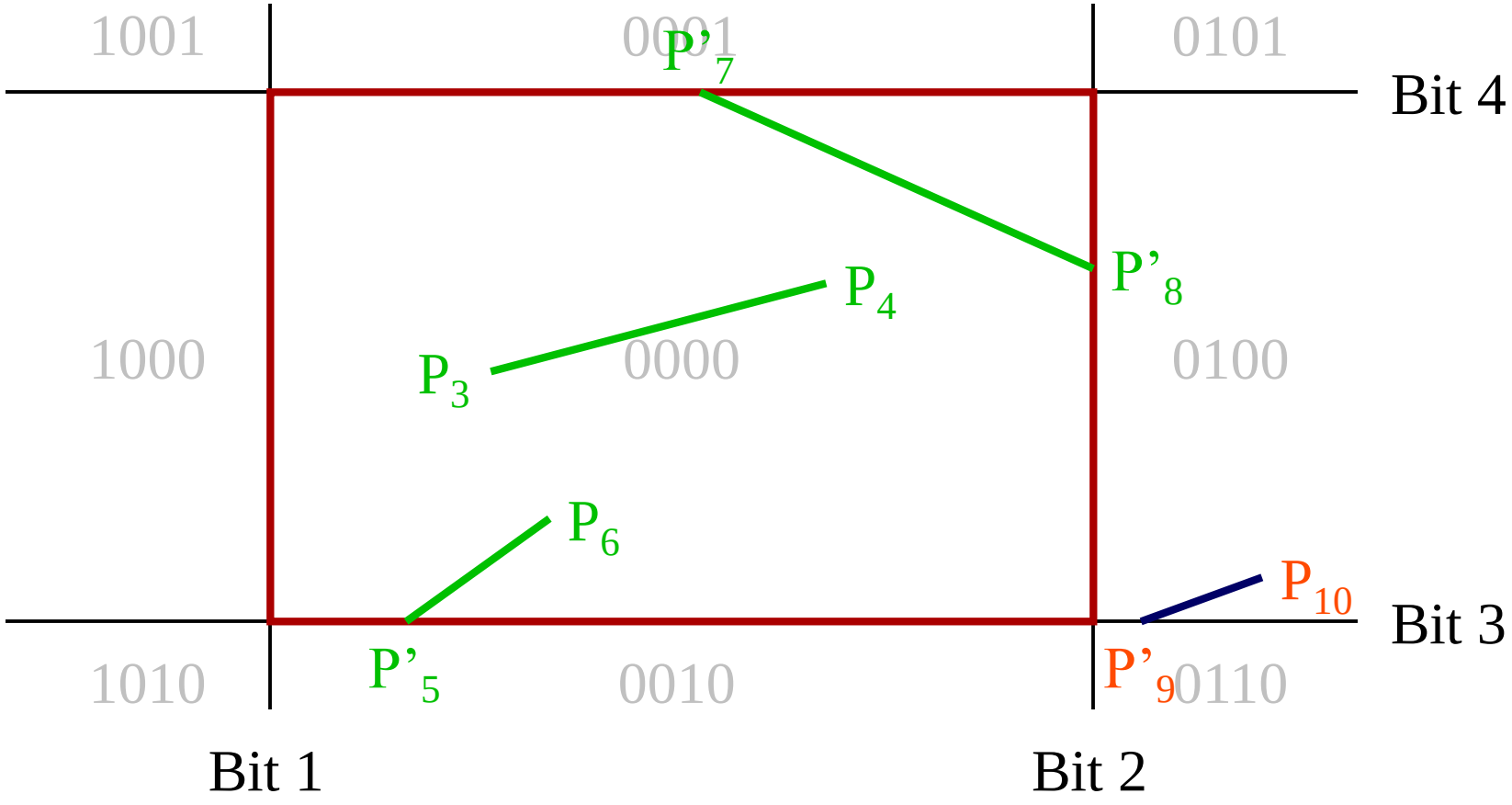
Cohen-Sutherland Line Clipping

- Compute Intersections with Window Boundary for Lines That Can't be Classified Quickly



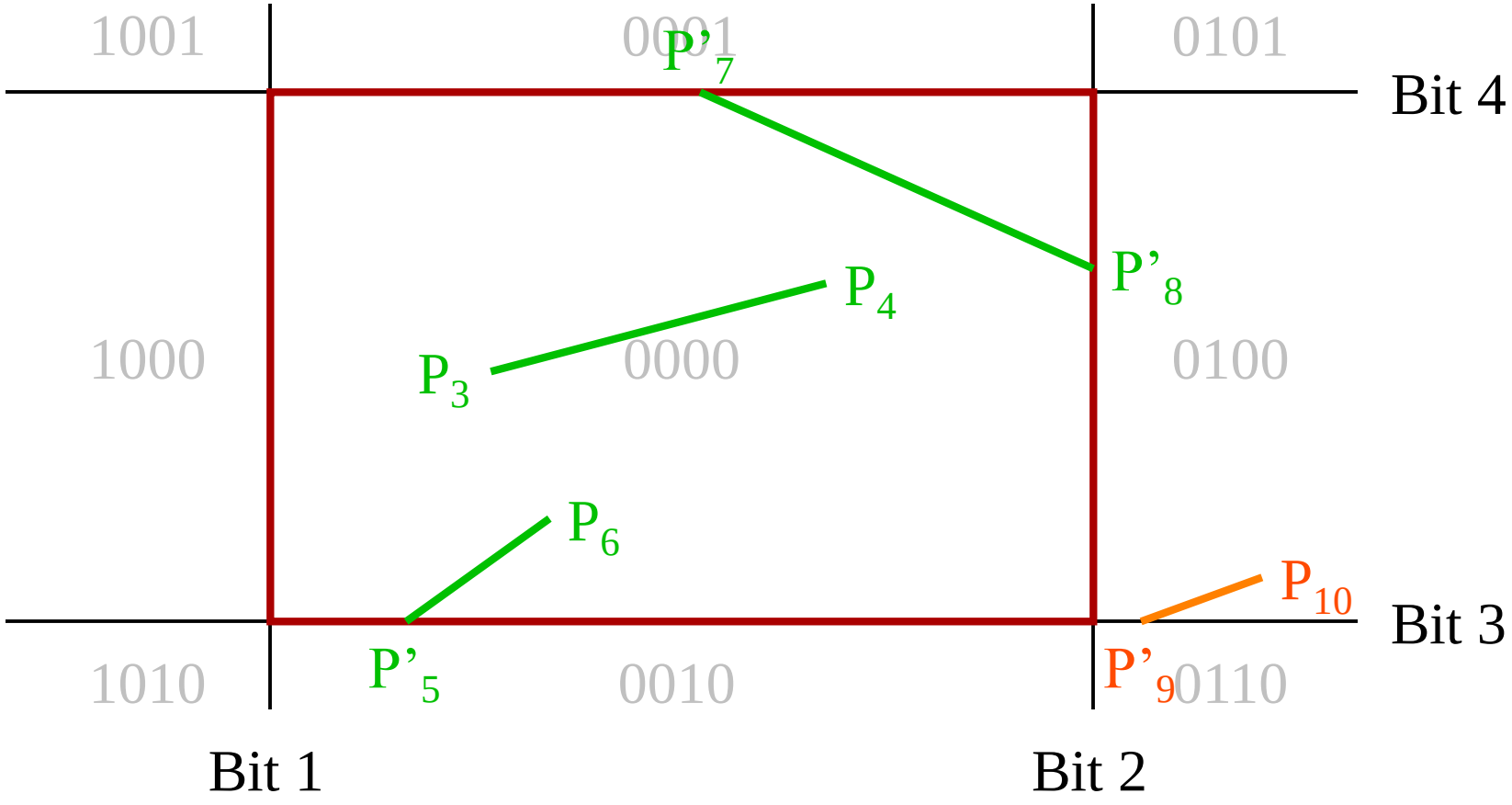
Cohen-Sutherland Line Clipping

- Compute Intersections with Window Boundary for Lines That Can't be Classified Quickly



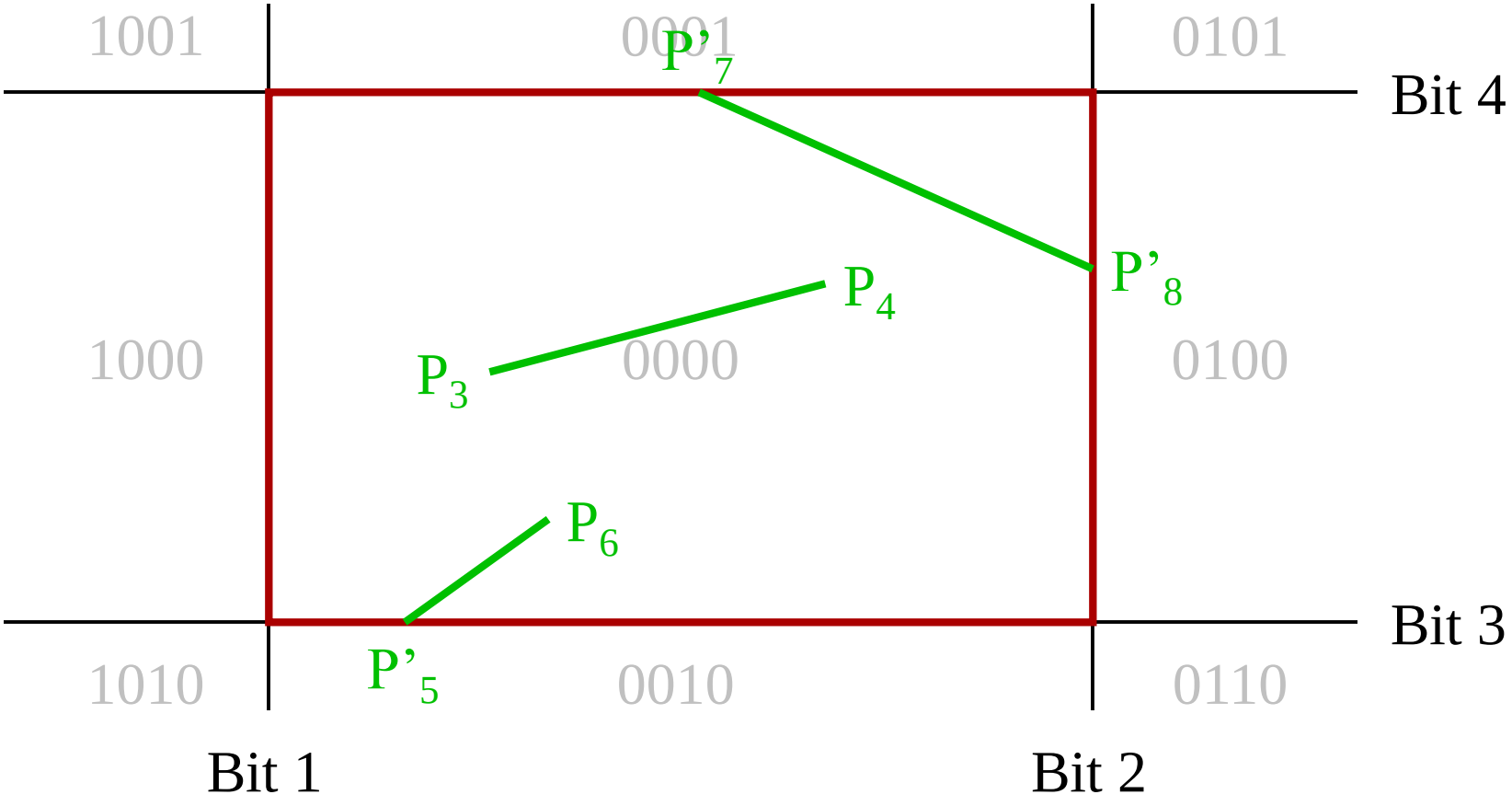
Cohen-Sutherland Line Clipping

- Compute Intersections with Window Boundary for Lines That Can't be Classified Quickly



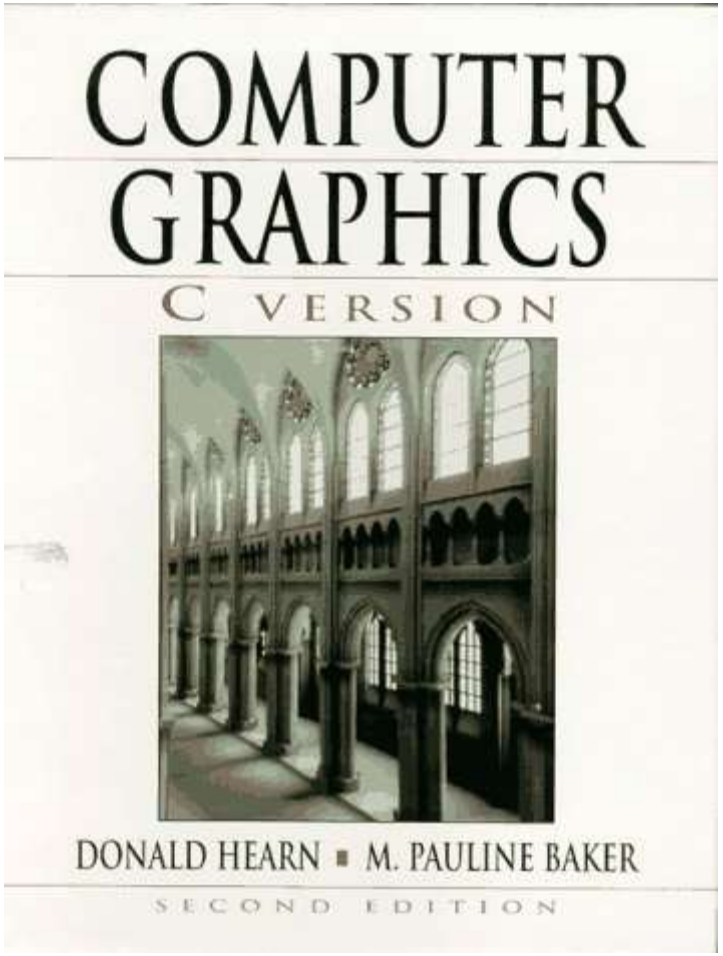
Cohen-Sutherland Line Clipping

- Compute Intersections with Window Boundary for Lines That Can't be Classified Quickly



Textbook

- Computer Graphics
  C Version
  - D. Hearn and M. P. Baker
  - 2nd Edition
  - PRENTICE HALL



Thank you

The Content in this Material are from the Textbooks and Reference books given in the Syllabus