Two Dimensional Viewing

Basic Interactive Programming

Basic Interactive Programming

- User controls contents, structure, and appearance of objects and their displayed images via rapid visual feedback.

Model

- **Model:** a pattern, plan, representation, or description designed to show the structure or working of an object, system, or concept.

Modeling

- **Modeling** is the process of creating, storing and manipulating a model of an object or a system.

Modeling

- In Modeling, we often use a geometric model
  - i.e.. A description of an object that provides a numerical description of its shape, size and various other properties.
- Dimensions of the object are usually given in units appropriate to the object:
  - meters for a ship
  - kilometres for a country

Modeling

- The shape of the object is often described in terms of sub-parts, such as circles, lines, polygons, or cubes.
- **Example:** Model of a house units are in meters

Instances of Objects

- Instances of this object may then be placed in various positions in a scene, or world, scaled to different sizes, rotated, or deformed.
- Each house is created with instances of the same model, but with different parameters.

2D Viewing

2D Viewing

Viewing is the process of drawing a view of a model on a 2-dimensional display.

2D Viewing

- The geometric description of the object or scene provided by the model, is converted into a set of graphical primitives, which are displayed where desired on a 2D display.
- The same abstract model may be viewed in many different ways:
  - e.g. faraway, near, looking down, looking up

Real World Coordinates

- It is logical to use dimensions which are appropriate to the object e.g.
  - meters for buildings
  - nanometers or microns for molecules, cells, atoms
  - light years for astronomy
- The **objects** are described with respect to their actual physical size in the **real world**, and then mapped onto screen co-ordinates.
- It is therefore possible to view an object at various sizes by zooming in and out, *without* actually having to change the model.

2D Viewing

- ***How much*** of the model should be **drawn**?
- ***Where*** should it appear on the **display**?

How do we **convert** Real-world coordinates into screen co-ordinates?

- We could have a model of a whole room, full of objects such as chairs, tablets and students.
- We may want to view the whole room in one go, or zoom in on one single object in the room.
- We may want to display the object or scene on the full screen, or we may only want to display it on a portion of the screen.

2D Viewing

- Once a model has been constructed, the programmer can specify a view.

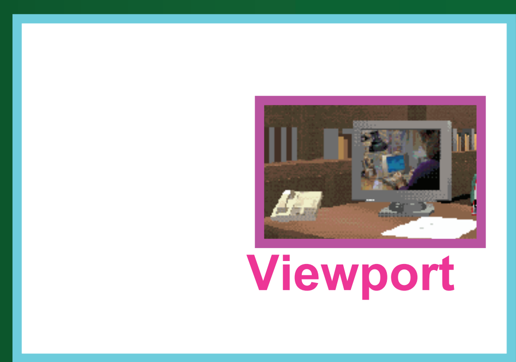
A 2-Dimensional view consists of *two* rectangles:

- A ***Window***, given in **real-world** co-ordinates, which defines the portion of the model that is to be drawn
- A ***Viewport*** given in **screen** co-ordinates, which defines the portion of the screen on which the contents of the window will be displayed

Basic Interactive Programming

- ***Window***: What is to be viewed
- ***Viewport***: Where is to be displayed

Scene



Image

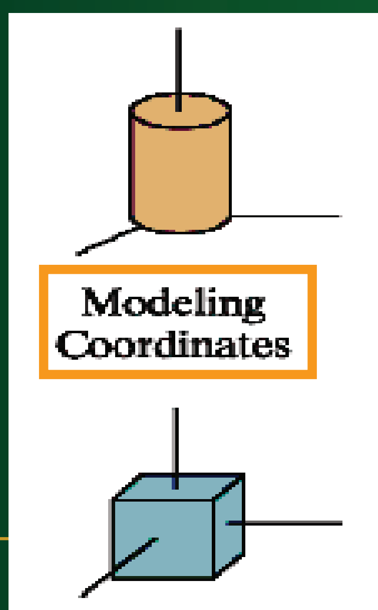
Coordinate Representations

Coordinate Representations

- General graphics packages are designed to be used with Cartesian coordinate specifications.
- Several different Cartesian reference frame are used to construct and display a scene.

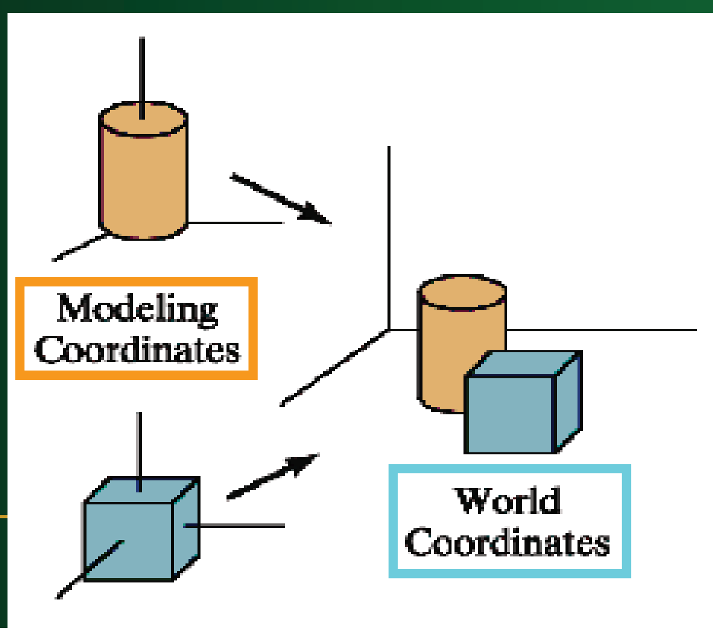
Coordinate Representations

- **Modeling coordinates:** We can construct the shape of individual objects in a scene within separate coordinate reference frames called modeling (local) coordinates.



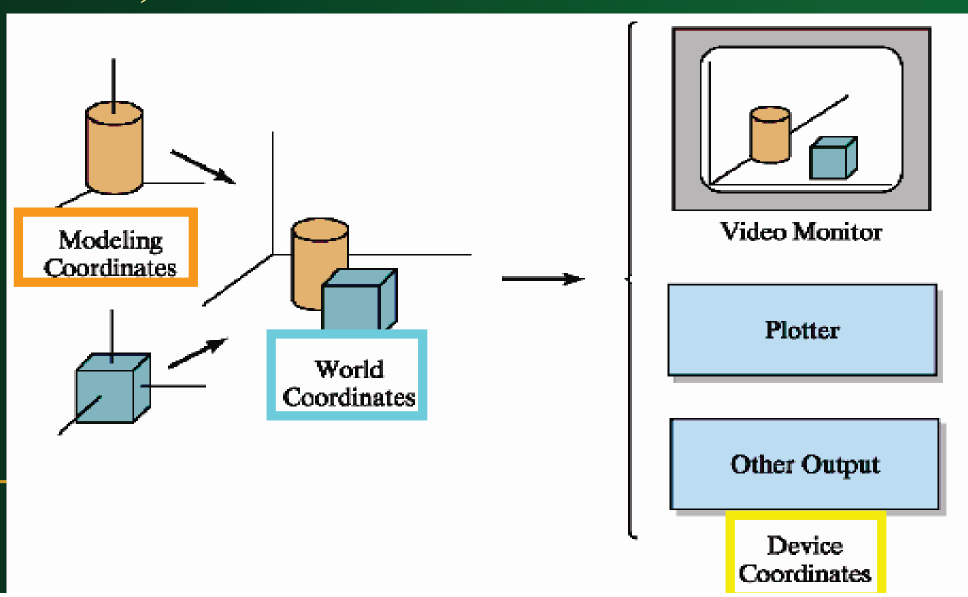
Coordinate Representations

- **World coordinates:** Once individual object shapes have been specified, we can place the objects into appropriate positions within the scene using reference frame called world coordinate.



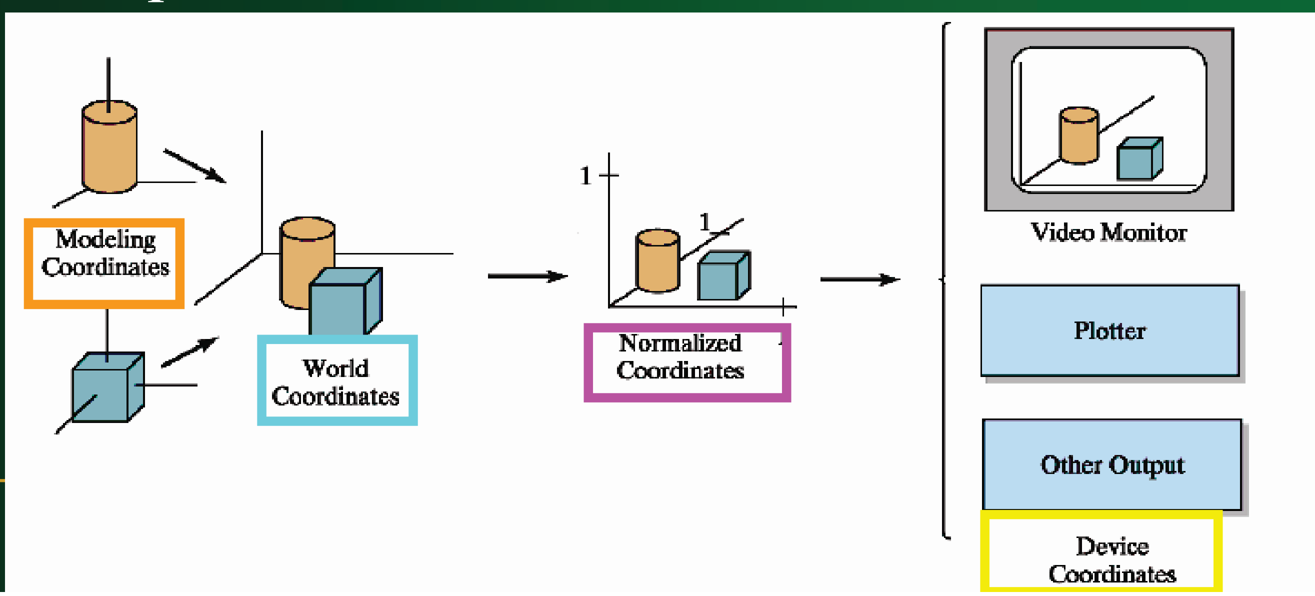
Coordinate Representations

- **Device Coordinates:** Finally, the world coordinates description of the scene is transferred to one or more output-device reference frames for display, called device (screen) coordinates.



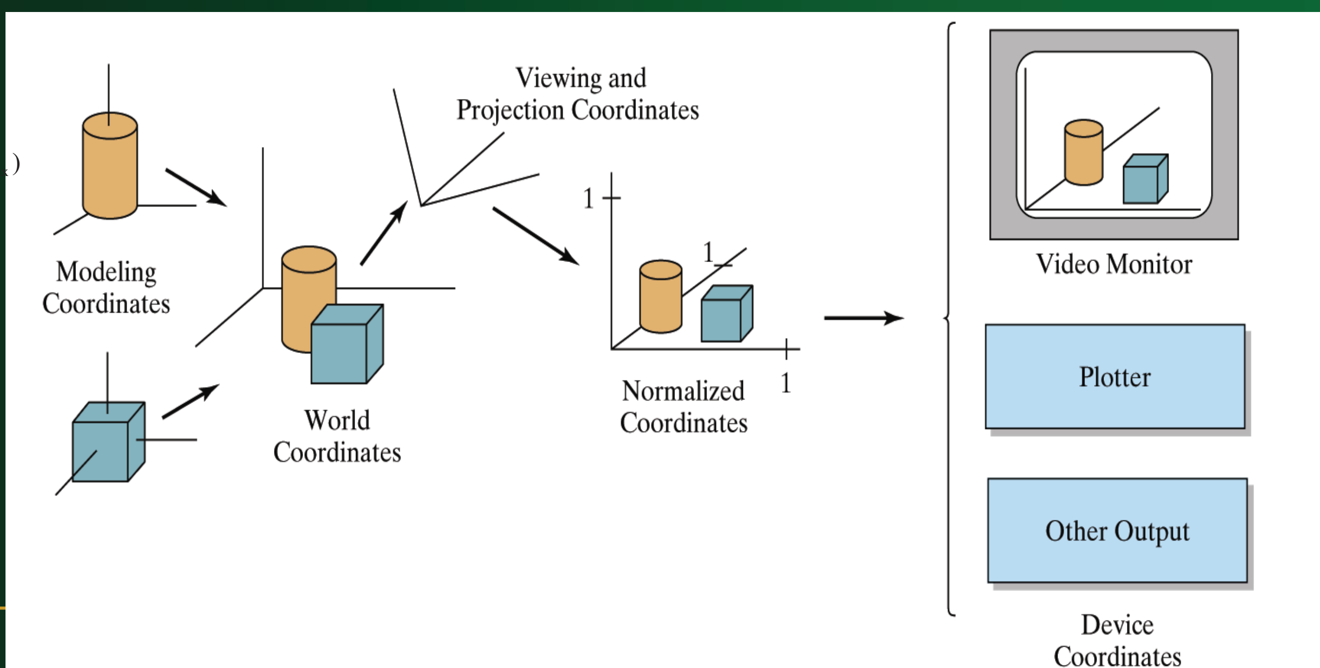
Coordinate Representations

- **Normalized Coordinates:** A graphic system first converts world coordinate positions to normalized device coordinates, in the range **0** to **1**.This makes the system independent of the output-devices.



Coordinate Representations

- Modeling coordinates: We can construct the shape of individual objects in a scene within separate coordinate reference frames called modeling (local) coordinates.



Coordinate Representations

- An initial modeling coordinate position is transferred to a device coordinate position with the sequence:

$$(x_{mc}, y_{mc}) \rightarrow (x_{wc}, y_{wc}) \rightarrow (x_{nc}, y_{nc}) \rightarrow (x_{dc}, y_{dc})$$

- The modeling and world coordinate positions in this transformation can be any floating values; normalized coordinates satisfy the inequalities:

$$0 \le x_{nc} \le 1 \qquad 0 \le y_{nc} \le 1$$

- The device coordinates are integers within the range (0,0) to $(x_{\max}, y_{\max})$ for a particular output device.

The Viewing Pipeline

The Viewing Pipeline

- A world coordinate area selected for display is called **window**.
- An area on a display device to which a window is mapped a **viewport**.
- Windows and viewports are rectangular in standard position.

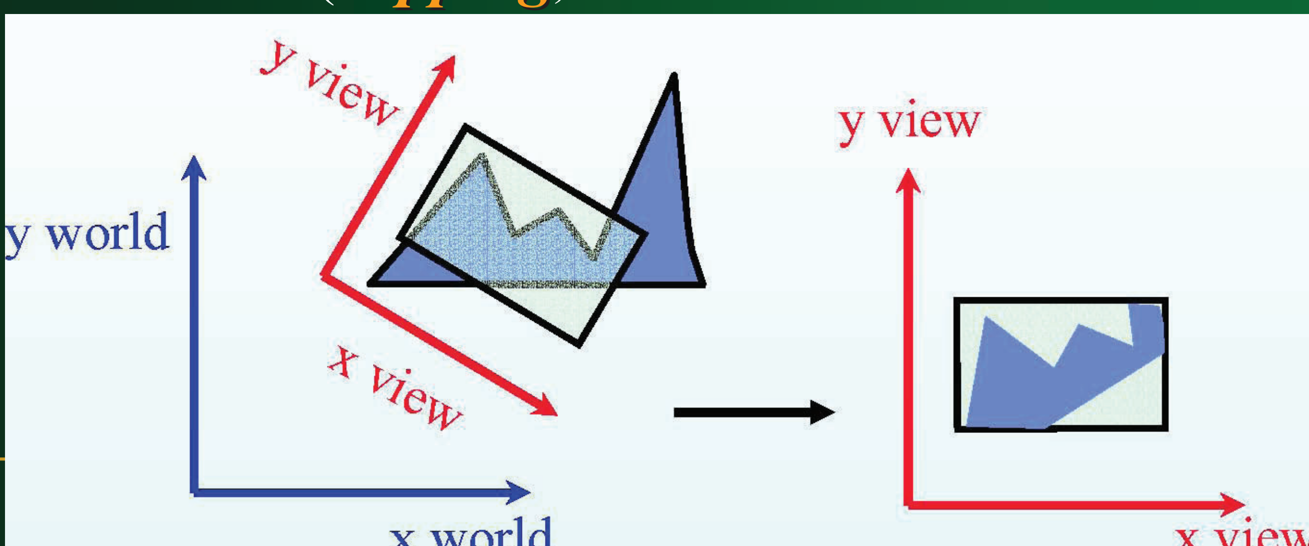
The Viewing Pipeline

- The mapping of a part of a world coordinate scene to device coordinate is referred to as **viewing transformation** or **window-to-viewport transformation** or **windowing transformation**.

window-to-viewport transformation

The Viewing Pipeline

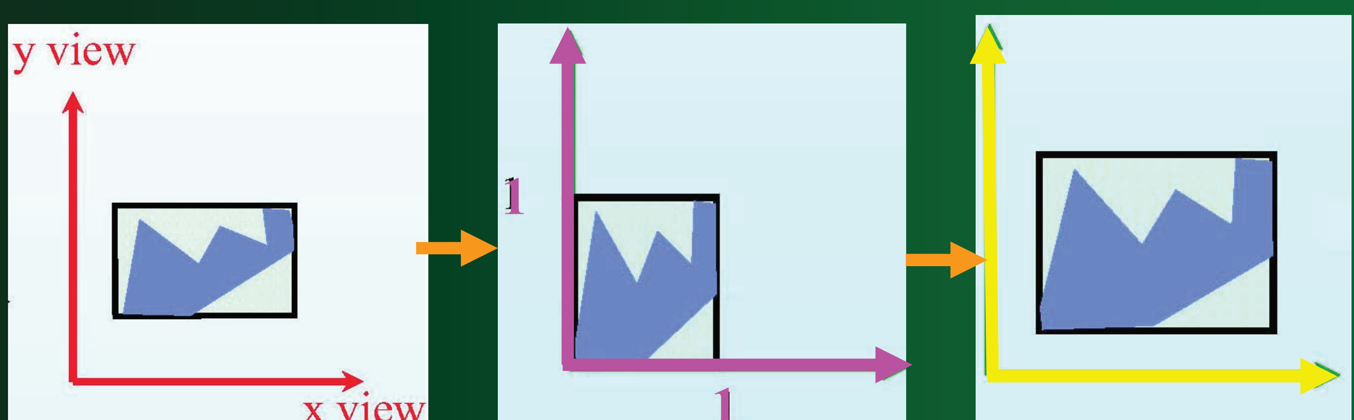
1. Construct the scene in world coordinate using the output primitives.
2. Obtain a particular orientation for the window by set up a two dimensional ***viewing coordinate*** system in the world coordinate, and define a window in the viewing coordinate system. Transform descriptions in world coordinates to viewing coordinates (***clipping***).



The Viewing Pipeline

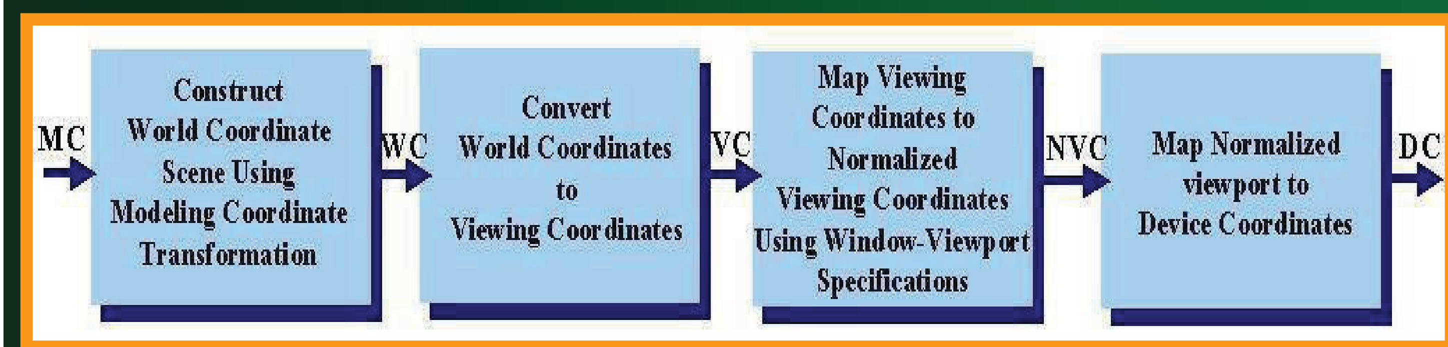
3. Define a viewport in normalized coordinate, and map the viewing coordinate description of the scene to normalized coordinate

4. (All parts lie outside the viewport are ***clipped***), and contents of the viewport are transferred to device coordinates.



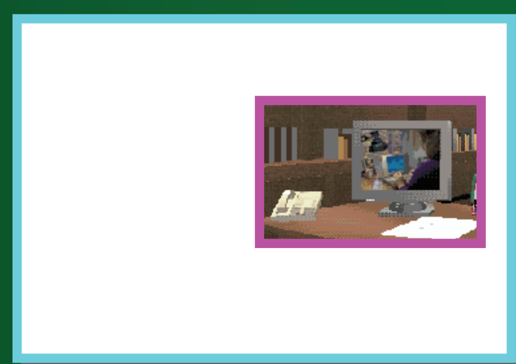
Viewing Coordinate **Normalized Coordinate** **Device Coordinate**

The Viewing Pipeline



The Viewing Pipeline

- By Changing the position of the viewport, we can view objects at different position on the display area of an output device.



The Viewing Pipeline

- By varying the size of viewport, we can change the size of displayed objects (zooming).

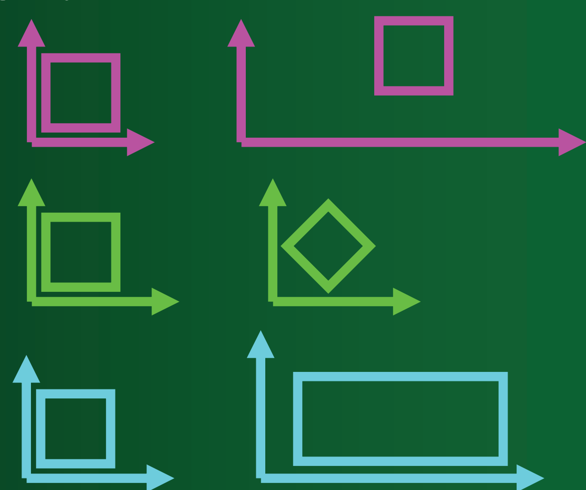
2D Geometric Transformations

2D Geometric Transformations

- Operations that are applied to the geometric description of an object to change its position, orientation or size.

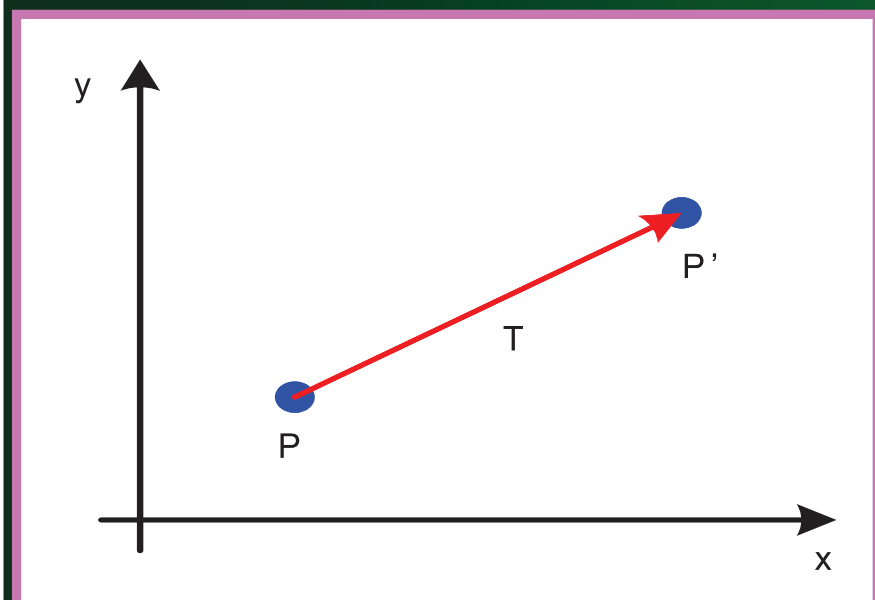
Basic transformation:

- **Translation**
- **Rotation**
- **Scaling**



2D Translation

- **2D Translation:** Move a point along a straight-line path to its new location.



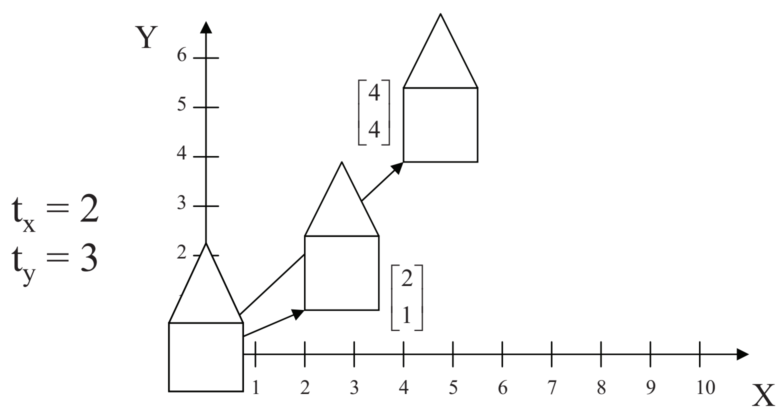
$$x' = x + t_x, \qquad y' = y + t_y$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

$$\mathbf{P}' = \mathbf{P} + \mathbf{T}$$

2D Tranlation

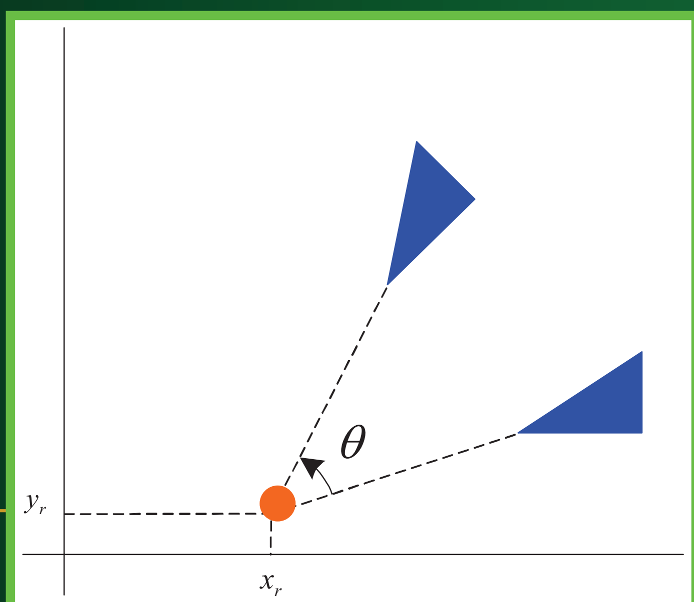
- **Rigid-body translation**: moves objects without deformation (every point of the object is translated by the same amount)



Note: House shifts position relative to origin

2D Rotation

- **2D Rotation:** Rotate the points a specified rotation angle about the rotation axis.
- Axis is perpendicular to xy plane; specify only rotation point (**pivot point** (x_r, y_r))



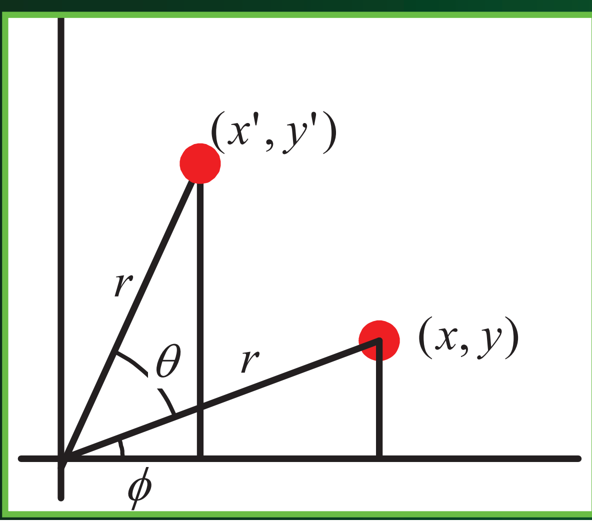
2D Rotation

- **Simplify:** rotate around origin: $x_r = 0, \quad y_r = 0$

$$x' = r\cos(\varphi + \theta) = r\cos\varphi\cos\theta - r\sin\varphi\sin\theta$$
$$y' = r\sin(\varphi + \theta) = r\cos\varphi\sin\theta + r\sin\varphi\cos\theta$$

$$x = r\cos\varphi, \qquad y = r\sin\varphi$$

$$x' = x\cos\theta - y\sin\theta$$
$$y' = x\sin\theta + y\cos\theta$$



$$\mathbf{R} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

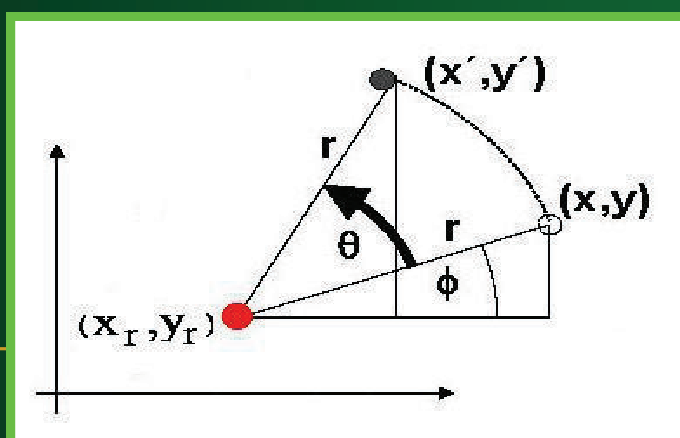
$$\mathbf{P}' = \mathbf{R} \cdot \mathbf{P}$$

2D Rotation

- Rotation of a point about an arbitrary pivot position:

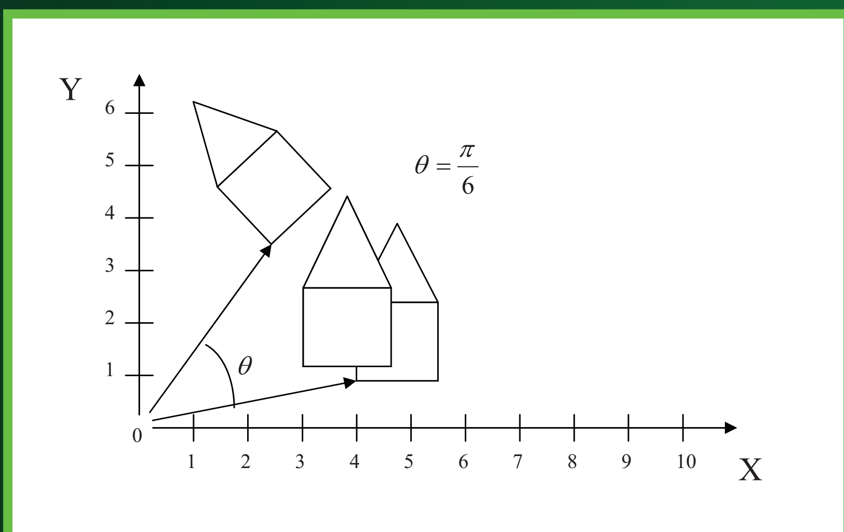
$$\boxed{\begin{aligned} x' &= x_r + (x - x_r)\cos\theta - (y - y_r)\sin\theta \\ y' &= y_r + (x - x_r)\sin\theta + (y - y_r)\cos\theta \end{aligned}}$$

- The matrix expression could be modified to include pivot coordinates by matrix addition of a column vector whose elements contain the additive (translational) term.



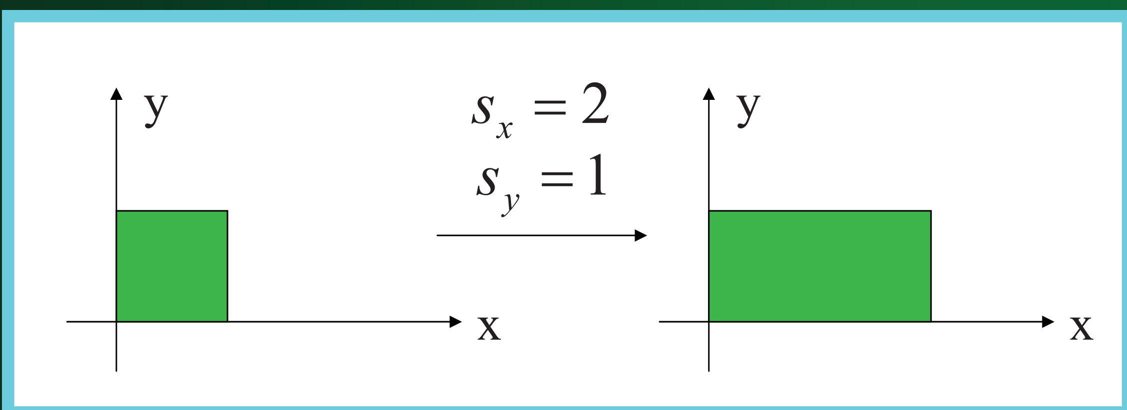
2D Rotation

- **Rigid-body translation**: Rotates objects without deformation (every point of the object is rotated through the same angle.



2D Scaling

- **2D Scaling:** Alters the size of an object.
- This operation can be carried out for polygons by multiplying the coordinate values (x, y) of each vertex by scaling factors S_x and S_y to produce the transformed coordinates

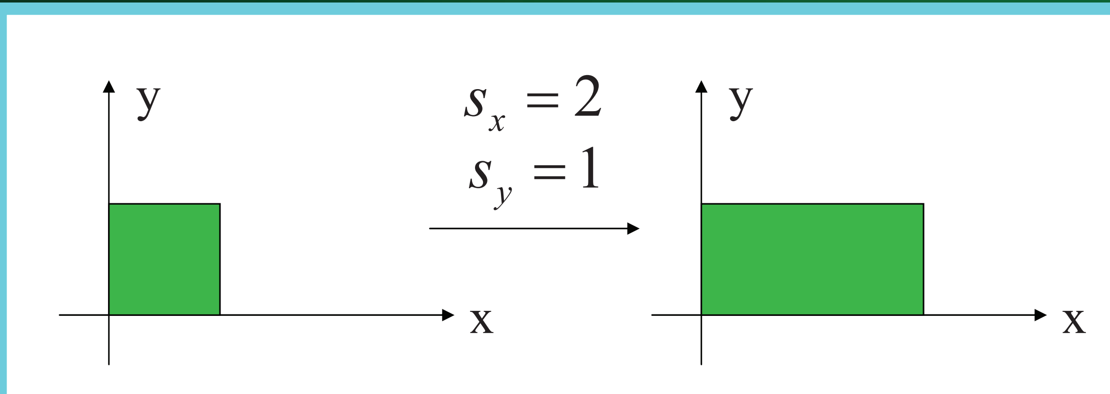


2D Scaling

$$\boxed{x' = x \cdot s_x, \quad y' = y \cdot s_y}$$

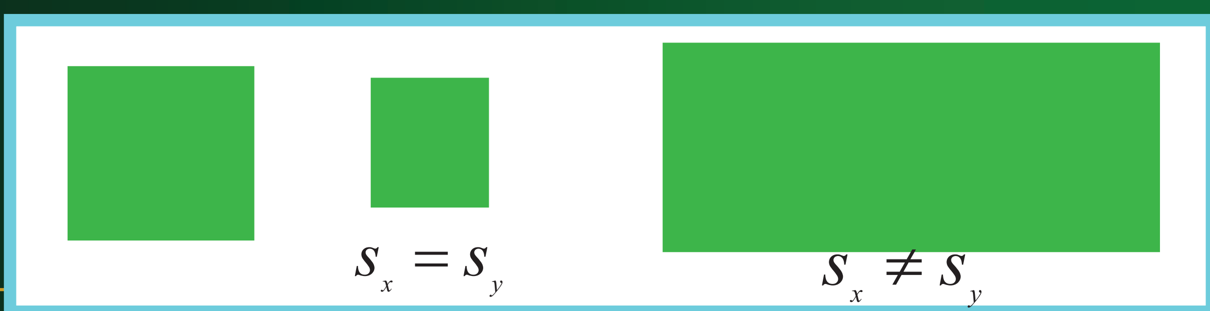
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

$$\mathbf{P'} = \mathbf{S} \cdot \mathbf{P}$$



2D Scaling

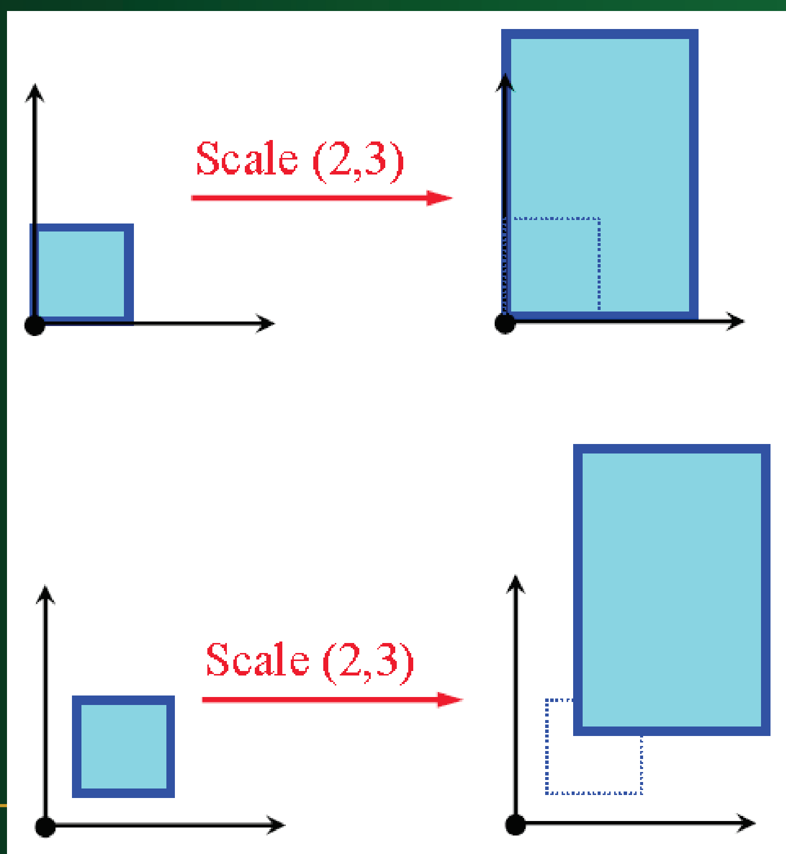
- An positive numeric values can be assigned to the scaling factors.
- Values less than 1 reduce the size of objects, and greater than 1 produce an enlargement.
- ***Uniform Scaling:*** $s_x = s_y$
- ***Differential Scaling:*** $s_x \neq s_y$, used in modeling applications.



original **Uniform scaling** **Differential scaling**

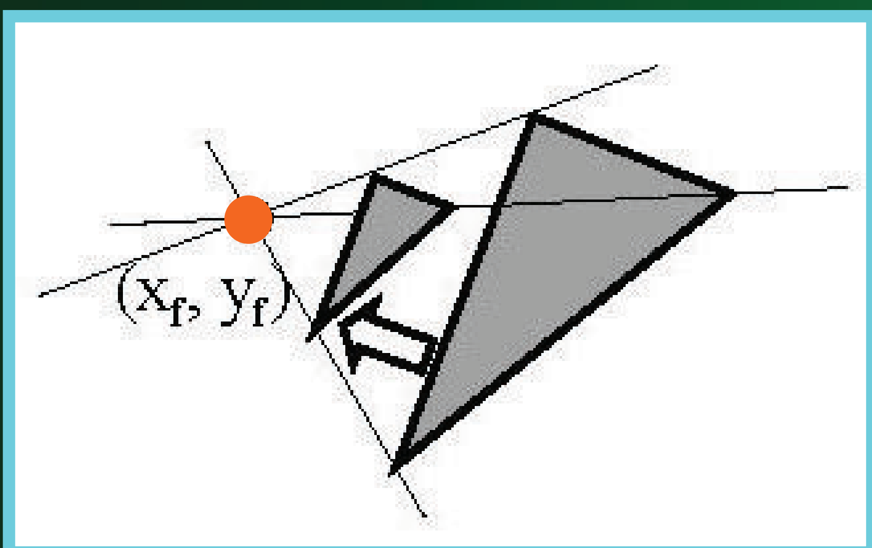
2D Scaling

- Scale an object moving its origin (upper right)



2D Scaling

- We can control the location of a scaled object by choosing a position, called **fixes point** (x_f, y_f)
- **Fixes point** can be chosen as one of the vertices, the object centroid, or any other position



$$x' = x_f + (x - x_f) \cdot s_x$$

$$y' = y_f + (y - y_f) \cdot s_y$$

$$x' = x \cdot s_x + x_f(1 - s_x)$$

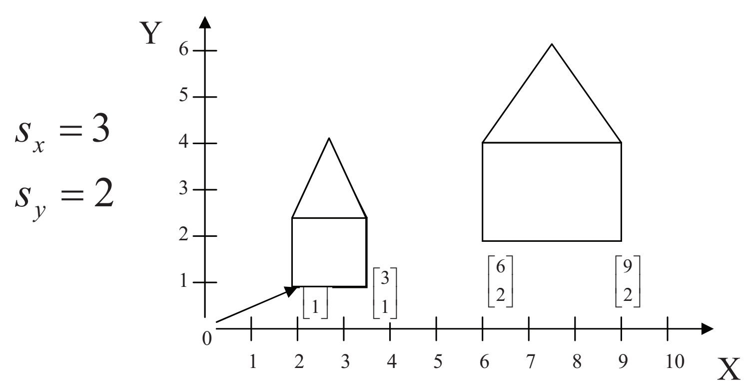
$$y' = y \cdot s_y + y_f(1 - s_y)$$

2D Scaling

$$x' = x \cdot s_x + x_f(1 - s_x)$$

$$y' = y \cdot s_y + y_f(1 - s_y)$$

- The matrix expression could be modified to include fixed coordinates.



Note: House shifts position relative to origin

Matrix Representations

And

Homogeneous Coordinates

Matrix Representations

- In Modeling, we perform sequences of geometric transformation: translation, rotation, and scaling to model components into their proper positions.

- ***How*** the matrix representations can be **reformulated** so that transformation sequences can be efficiently processed?

Matrix Representations

- We have seen:

Rotation:

$$x' = x_r + (x - x_r)\cos\theta - (y - y_r)\sin\theta$$
$$y' = y_r + (x - x_r)\sin\theta + (y - y_r)\cos\theta$$

Scaling:

$$x' = x \cdot s_x + x_f(1 - s_x)$$
$$y' = y \cdot s_y + y_f(1 - s_y)$$

- The basic transformations can be expressed in the general matrix form:

$$\boxed{\mathbf{P}' = \mathbf{M}_1 \cdot \mathbf{P} + \mathbf{M}_2}$$

Matrix Representations

$$\boxed{\mathbf{P}' = \mathbf{M}_1 \cdot \mathbf{P} + \mathbf{M}_2}$$

- $\mathbf{M_1}$ is a 2×2 array containing multiplicative factors.
- $\mathbf{M_2}$ is a two element column matrix containing translation terms.

- Translation: M_1 is the identity matrix.
- Rotation: M_2 contains the translation terms associated with the pivot point.
- Scaling: M_2 contains the translation terms associated with the fixed point.

Matrix Representations

$$\boxed{\mathbf{P}' = \mathbf{M}_1 \cdot \mathbf{P} + \mathbf{M}_2}$$

To produce a sequence of transformations, we must **calculate** the transformed coordinates **one step at a time**.

- We need to **eliminate** the matrix addition associated with the translation terms in $\mathbf{M_2}$.

Matrix Representations

We can combine the multiplicative and translation terms for 2D transformation into a **single matrix** representation

by

expanding 2×2 matrix to 3×3 matrix.

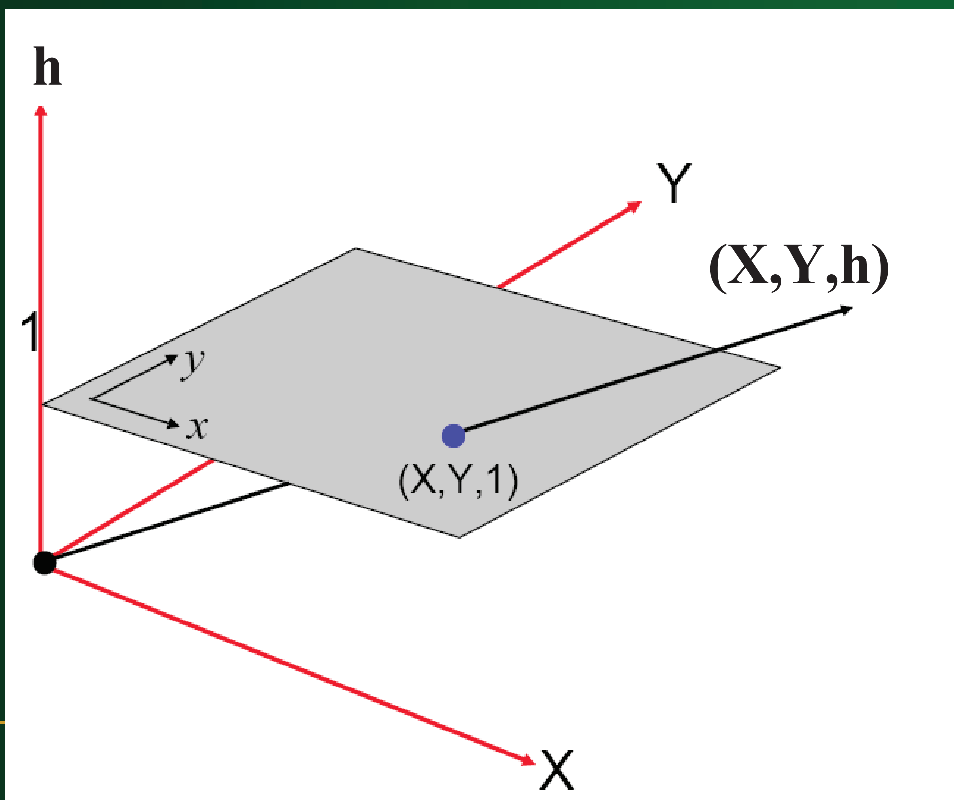
Matrix Representations
and
Homogeneous Coordinate

- **Homogeneous Coordinate:** To express any 2D transformation as a matrix multiplication, we represent each Cartesian coordinate position **(x,y)** with the homogeneous Coordinate triple **(x_h,y_h,h):**

$$\boxed{x = \frac{x_h}{h}, \qquad y = \frac{y_h}{h}}$$

Simply: h=1

Matrix Representations and Homogeneous Coordinate



Matrix Representations and Homogeneous Coordinate

Expressing position in homogeneous Coordinates, **(x,y,1)** allows us to represent all geometric transformation as matrix multiplication

Matrix Representations and Homogeneous Coordinate

- Basic 2D transformations as 3x3 matrices:

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

Translation

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

Rotation

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

Scaling

Composite Transformation

Composite Transformation

- **Combined transformations**
  - By matrix multiplication

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \left(\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \right) \cdot \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

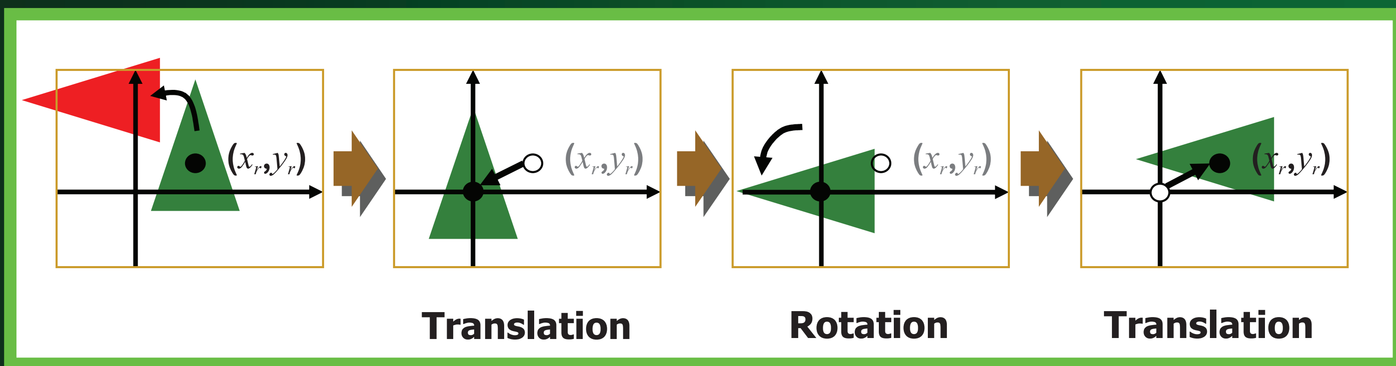
$$\mathbf{p}' = \mathbf{T}(t_x, t_y) \bullet \mathbf{R}(\theta) \bullet \mathbf{S}(s_x, s_y) \bullet \mathbf{p}$$

- Efficiency with pre-multiplication

$$\mathbf{p}' = (\mathbf{T} \times (\mathbf{R} \times (\mathbf{S} \times \mathbf{p}))) \Rightarrow \mathbf{p}' = (\mathbf{T} \times \mathbf{R} \times \mathbf{S}) \times \mathbf{p}$$

General Pivot Point Rotation

General Pivot Point Rotation

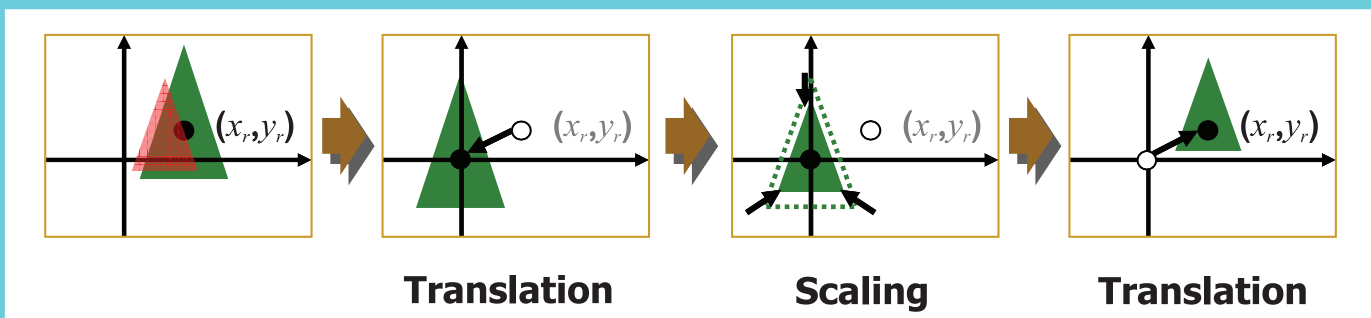


$$\mathbf{T}(x_r, y_r) \cdot \mathbf{R}(\theta) \cdot \mathbf{T}(-x_r, -y_r) = \mathbf{R}(x_r, y_r, \theta)$$

$$\begin{bmatrix} 1 & 0 & x_r \\ 0 & 1 & y_r \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -x_r \\ 0 & 1 & -y_r \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & x_r(1-\cos\theta)+y_r\sin\theta \\ \sin\theta & \cos\theta & y_r(1-\cos\theta)-x_r\sin\theta \\ 0 & 0 & 1 \end{bmatrix}$$

General Fixed Point Scaling

General Fixed Point Scaling



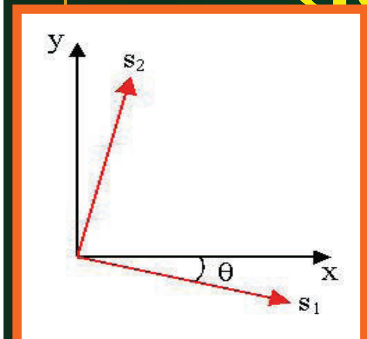
$$\mathbf{T}(x_r,y_r)\cdot\mathbf{S}(s_x,s_y)\cdot\mathbf{T}(-x_r,-y_r)=\mathbf{S}(x_r,y_r,s_x,s_y)$$

$$\begin{bmatrix}1 & 0 & x_r\\ 0 & 1 & y_r\\ 0 & 0 & 1\end{bmatrix}\cdot\begin{bmatrix}s_x & 0 & 0\\ 0 & s_y & 0\\ 0 & 0 & 1\end{bmatrix}\cdot\begin{bmatrix}1 & 0 & -x_r\\ 0 & 1 & -y_r\\ 0 & 0 & 1\end{bmatrix}=\begin{bmatrix}s_x & 0 & x_r(1-s_x)\\ 0 & s_y & y_r(1-s_y)\\ 0 & 0 & 1\end{bmatrix}$$

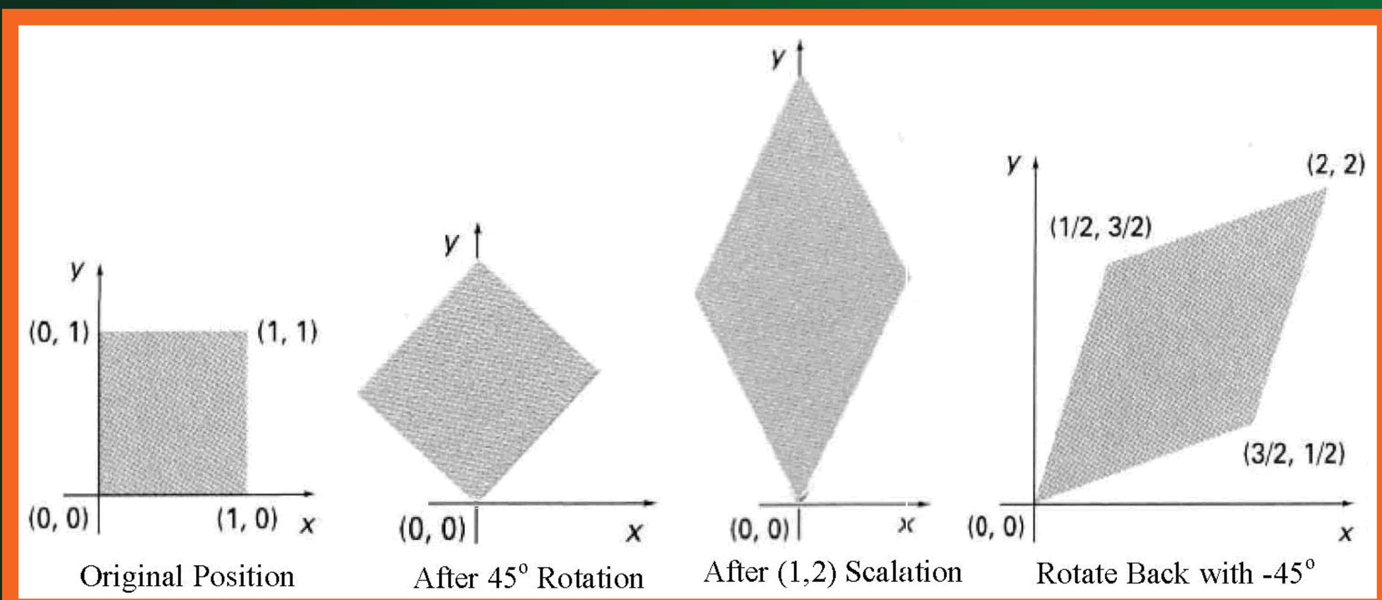
General Scaling

Direction

General Scaling Direction



$$\mathbf{R}^{-1}(\theta)\cdot\mathbf{S}(s_1,s_2)\cdot\mathbf{R}(\theta)=\begin{bmatrix} s_1\cos^2\theta+s_2\sin^2\theta & (s_2-s_1)\cos\theta\sin\theta & 0\\ (s_2-s_1)\cos\theta\sin\theta & s_1\sin^2\theta+s_2\cos^2\theta & 0\\ 0 & 0 & 1\end{bmatrix}$$



Reflection

Reflection

- **Reflection:** Produces a mirror image of an object.

X Axis

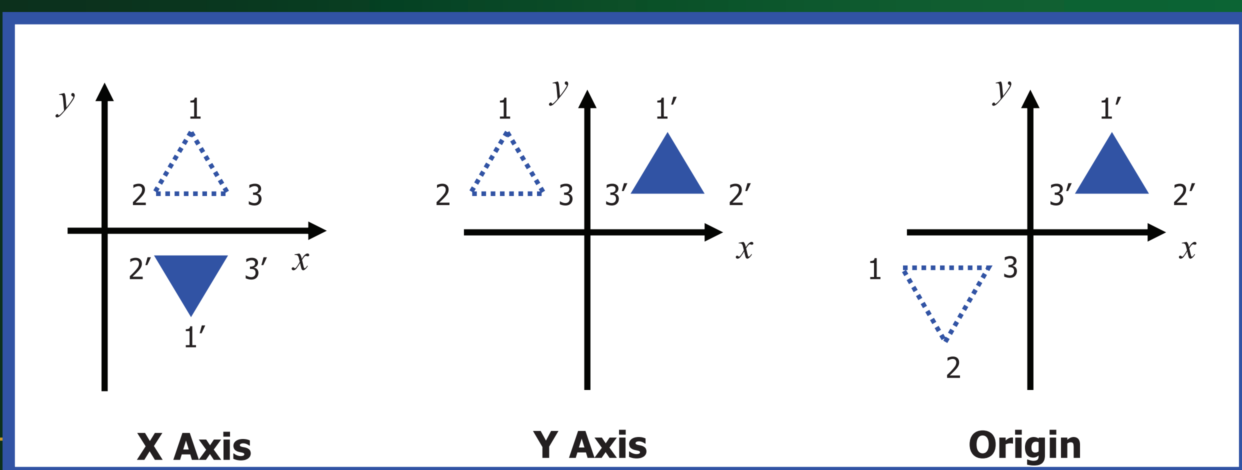
$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Y Axis

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Origin

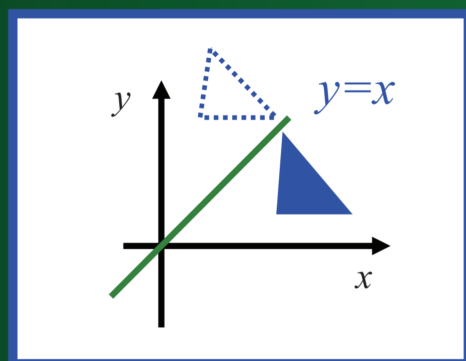
$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



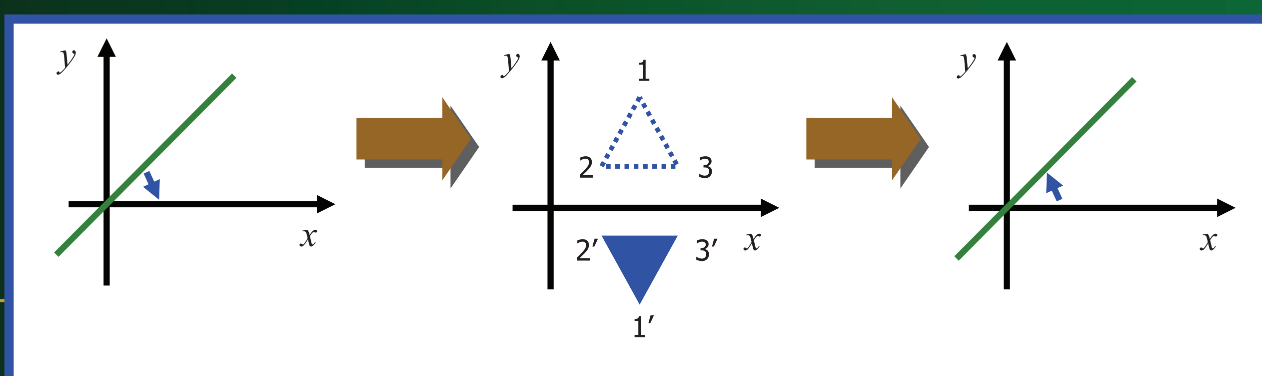
Reflection

- Reflection with respect to a line **y=x**

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



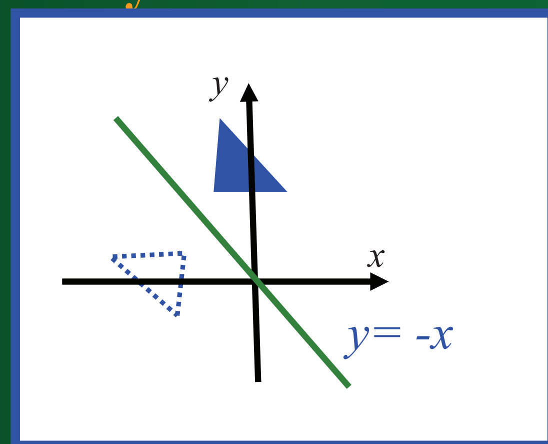
- We can drive this matrix by : Clockwise rotation of 45° → Reflection about the x axis → Counterclockwise rotation of 45°



Reflection

- Reflection with respect to a line **y=-x**

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



Reflection with respect to a line y=mx+b:

1. Translate the line so that it passes through the origin.
2. Rotate the onto one of the coordinate axes
3. Reflect about that axis
4. Inverse rotation
5. Inverse translation

Shear

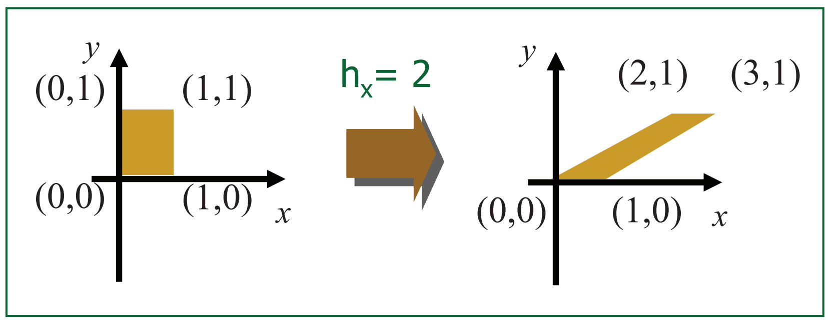
Shear

- **Shear:** A transformation that distorts the shape of an object such that transformed shape appears as if the object were composed of internal layers that had been caused to slide over each other is called a shear.

Shear

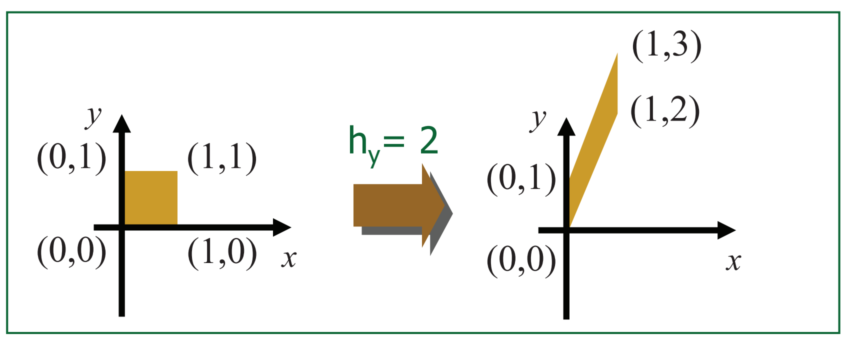
- x-direction: $x' = x + h_x \cdot y, \quad y' = y$

$$\begin{bmatrix} 1 & h_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



- y-direction: $x' = x, \quad y' = y + h_y \cdot x$

$$\begin{bmatrix} 1 & 0 & 0 \\ h_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

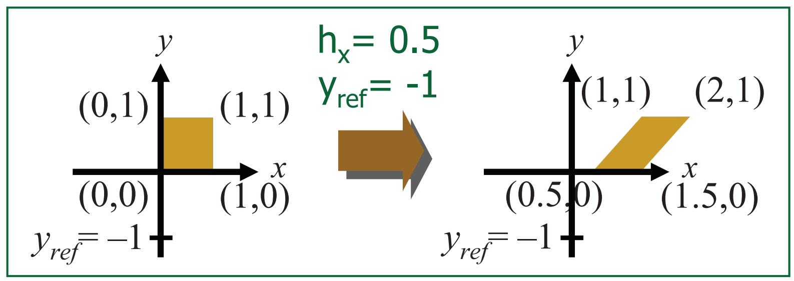


Shear

- x-direction shears relative to a reference line $y = y_{ref}$

$$x' = x + h_x \cdot (y - y_{ref}),\quad y' = y$$

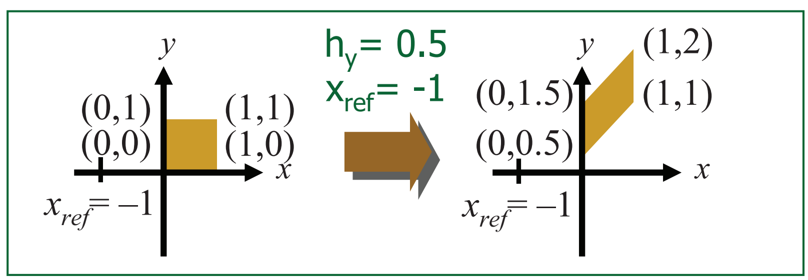
$$\begin{bmatrix} 1 & h_x & -h_x \cdot y_{ref} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



- y-direction shears relative to a reference line $x = x_{ref}$

$$x' = x,\quad y' = y + h_y \cdot (x - x_{ref})$$

$$\begin{bmatrix} 1 & 0 & 0 \\ h_y & 1 & -h_y \cdot x_{ref} \\ 0 & 0 & 1 \end{bmatrix}$$



Transformations Between Coordinates Systems

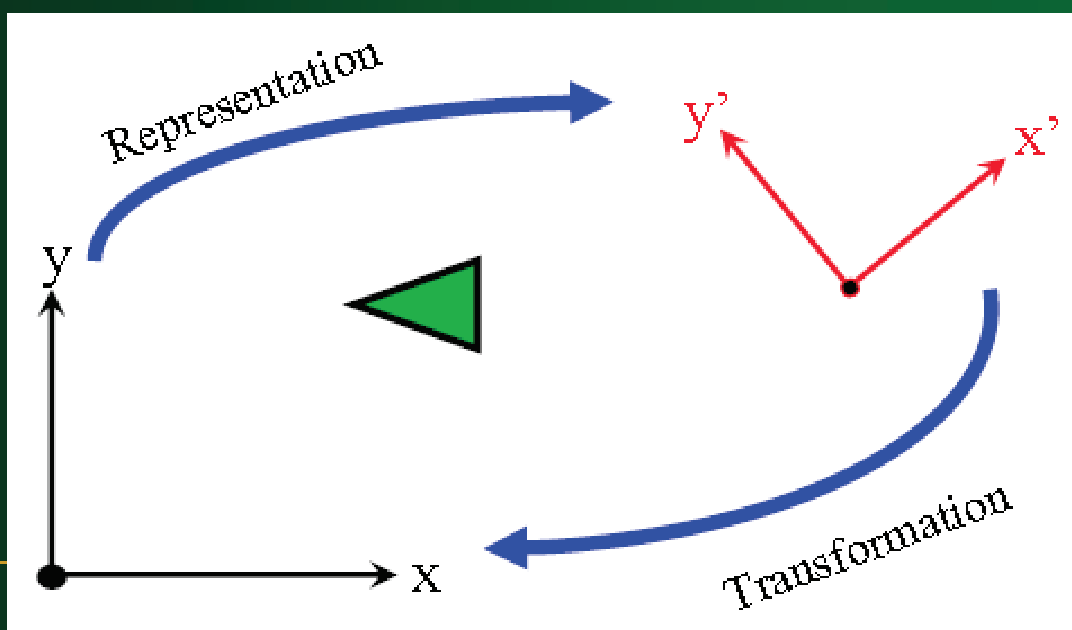
Transformations Between Coordinates Systems

- It is often requires the transformation of object description from one coordinate system to another.

How do we transform between two Cartesian coordinate systems?

Transformations Between Coordinates Systems

- **Rule:** Transform one coordinate frames towards the other in the opposite direction of the representation change.

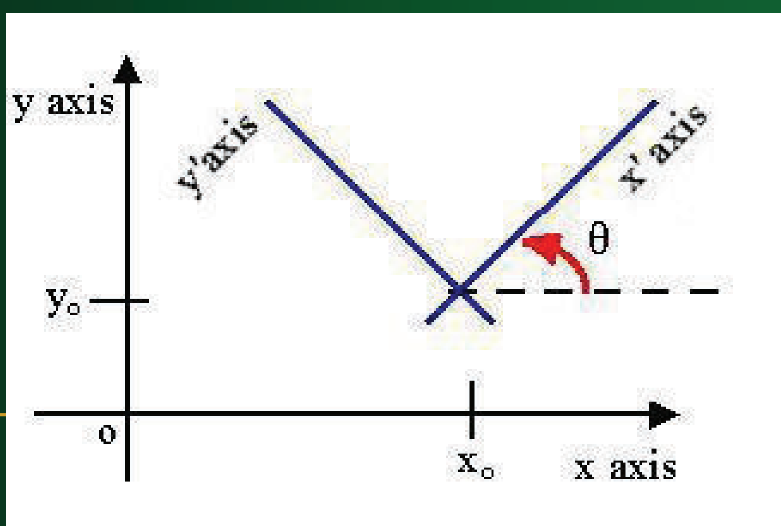


Transformations Between Coordinates Systems

- **Two Steps**:

1. Translate so that the origin (x_0,y_0) of the $x'y'$ system is moved to the origin of the xy system.

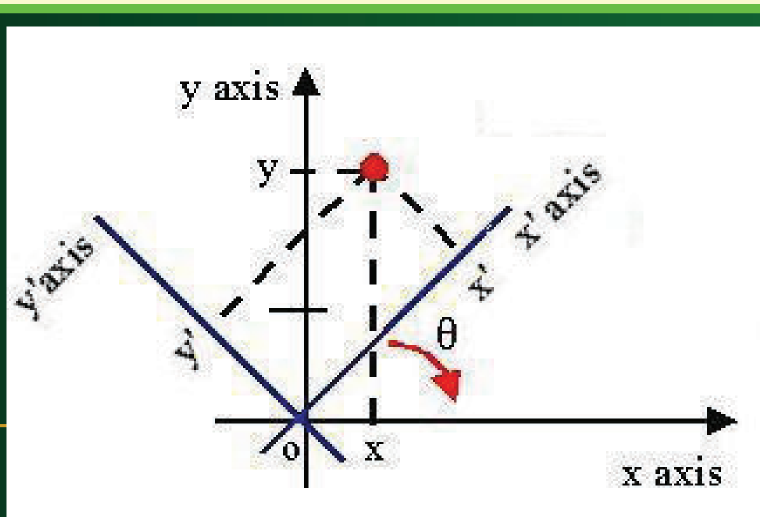
2. Rotate the x' axis onto the x axis.



Transformations Between Coordinates Systems

$$T(-x_0,-y_0)=\begin{bmatrix}1 & 0 & -x_0\\ 0 & 1 & -y_0\\ 0 & 0 & 1\end{bmatrix} \qquad R(-\theta)=\begin{bmatrix}\cos\theta & \sin\theta & 0\\ -\sin\theta & \cos\theta & 0\\ 0 & 0 & 1\end{bmatrix}$$

$$\boxed{\mathbf{M}_{xyx'y'}=\mathbf{R}(-\theta)\cdot\mathbf{T}(-x_0,-y_0)}$$



Transformations Between Coordinates Systems

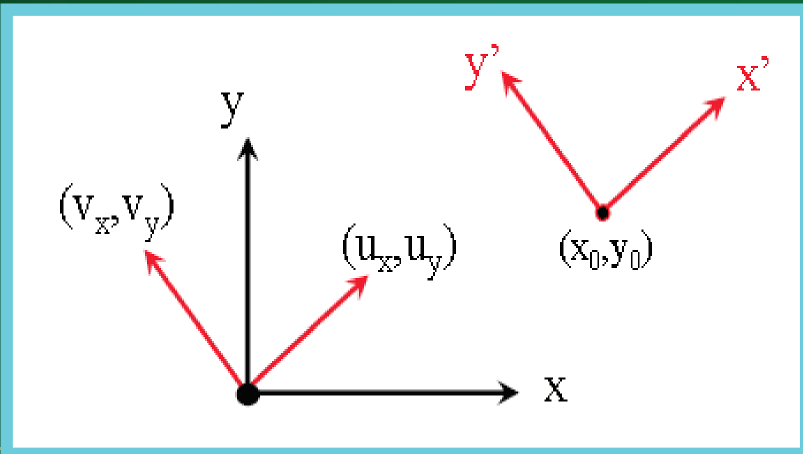
Alternative Method:

- Assume $x'=(u_x,,u_y)$ and $y'=(v_x,v_y)$ in the (x,y) coordinate systems:

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

$$M = \begin{bmatrix} u_x & u_y & 0 \\ v_x & v_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -x_0 \\ 0 & 1 & -y_0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{M} = \mathbf{R} \cdot \mathbf{T}$$



Transformations Between Coordinates Systems

Example:

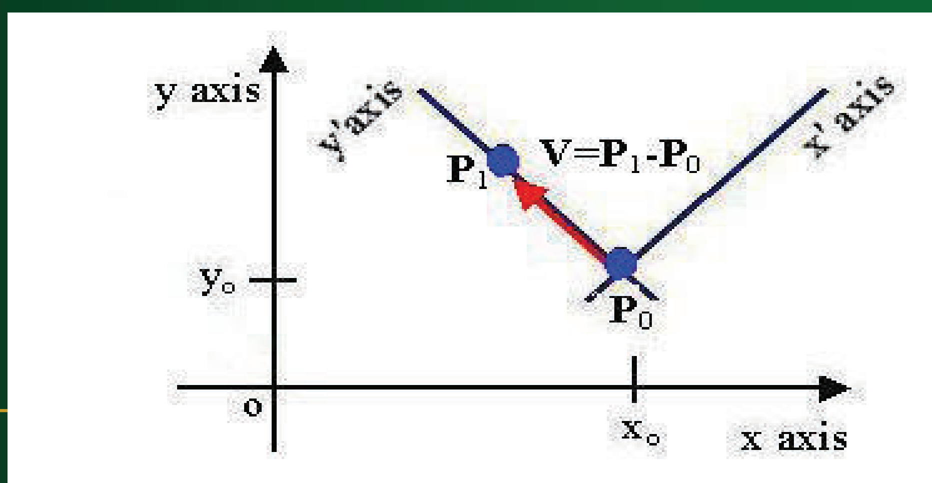
- If **V**=(-1,0) then the x′ axis is in the positive direction y and the rotation transformation matrix is:

$$R = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Transformations Between Coordinates Systems

- **In an interactive application,** it may be more convenient to choose the direction for **V** relative to position $\mathbf{P}_0$ than it is to specify it relative to the xy coordinate origin.

$$\mathbf{v} = \frac{\mathbf{P}_1 - \mathbf{P}_0}{\left|\mathbf{P}_1 - \mathbf{P}_0\right|}$$



Viewing Coordinate Reference Frame

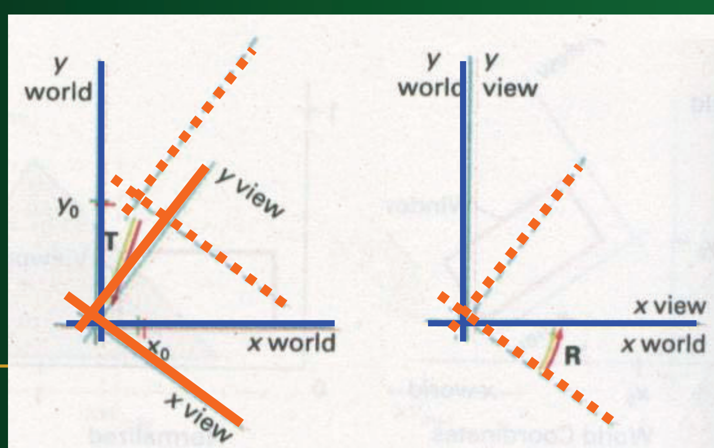
Viewing Coordinate Reference Frame

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

Viewing Coordinate Reference Frame

Set up the viewing coordinate:

1. Origin is selected at some world position: $\mathbf{P}_0 = (x_0, y_0)$
2. Established the orientation. Specify a world vector V that defines the viewing y direction.
3. Obtain the matrix for converting world to viewing coordinates (Translate and rotate) $\mathbf{M}_{WC,VC} = \mathbf{R} \cdot \mathbf{T}$



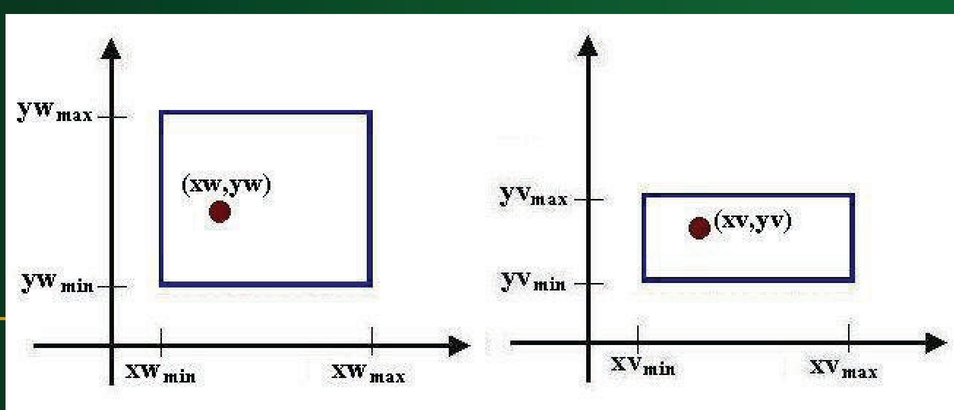
Window to Viewport Coordinate Transformation

Window to Viewport Coordinate Transformation

- Select the viewport in normalized coordinate, and then object description transferred to normalized device coordinate.
- To maintain the same relative placement in the viewport as in the window:

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

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



Window to Viewport Coordinate Transformation

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

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

$$\boxed{\begin{aligned} xv &= xv_{\min} + (xw - xw_{\min})s_x \\ yv &= yv_{\min} + (yw - yw_{\min})s_y \end{aligned}}$$

1. Perform a scaling transformation that scales the window area to the size of the viewport.
2. Translate the scaled window area to the position of the vieport.

Clipping

Clipping

Clipping

- **Clipping Algorithm** or **Clipping**: Any procedure that identifies those portion of a picture that are either inside or outside of a specified region of space.

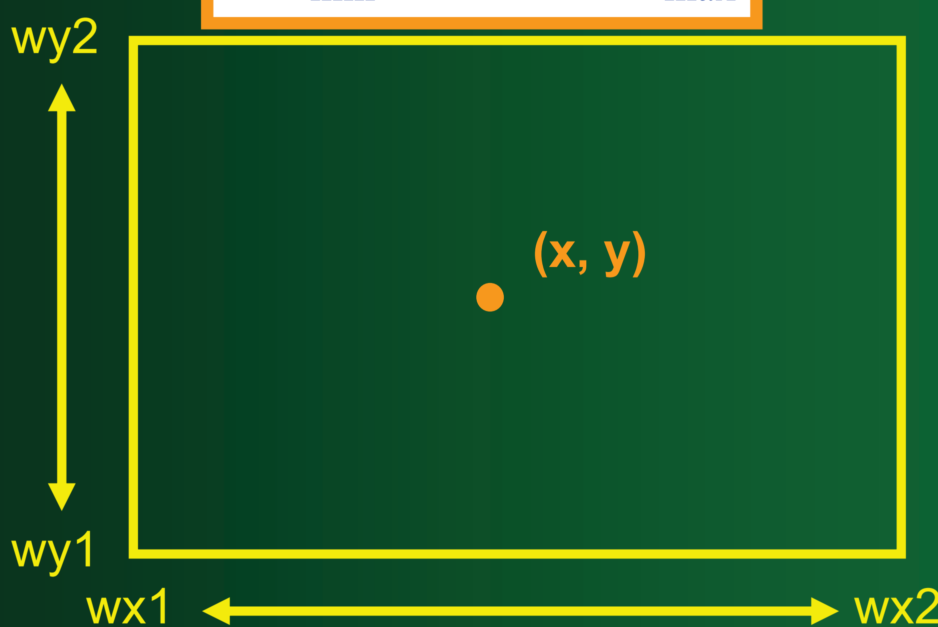
- The region against which an object is to clipped is called a ***clip window***.

Point Clipping

Point Clipping

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

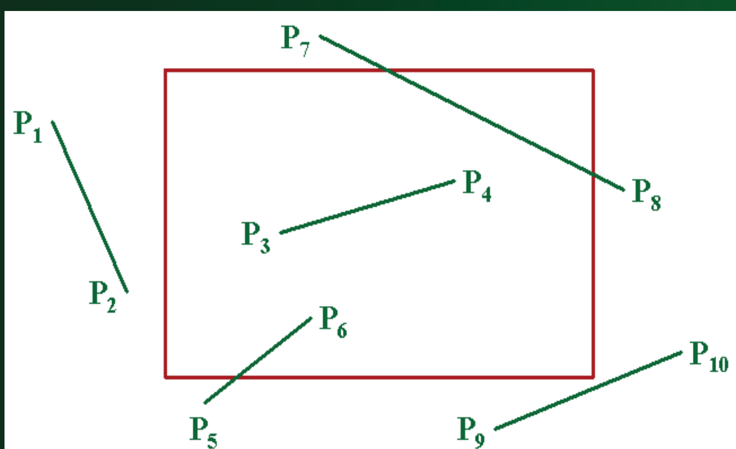
$$yw_{\min} \leq y \leq yw_{\max}$$

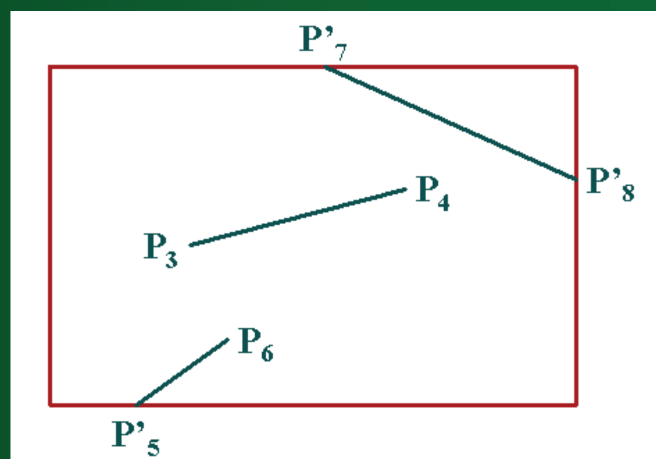


Line Clipping

Line Clipping

- Possible relationship between line position and a standard clipping region.





Before Clipping

After Clipping

Line Clipping

- **A line clipping procedure involves several parts:**

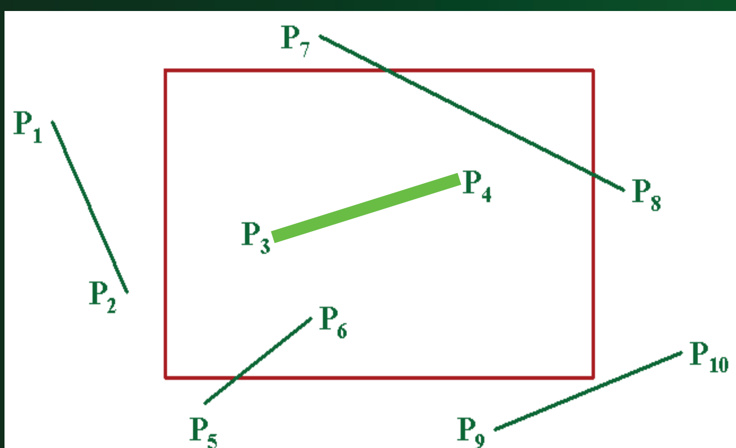
1. Determine whether line lies completely inside the clipping window.

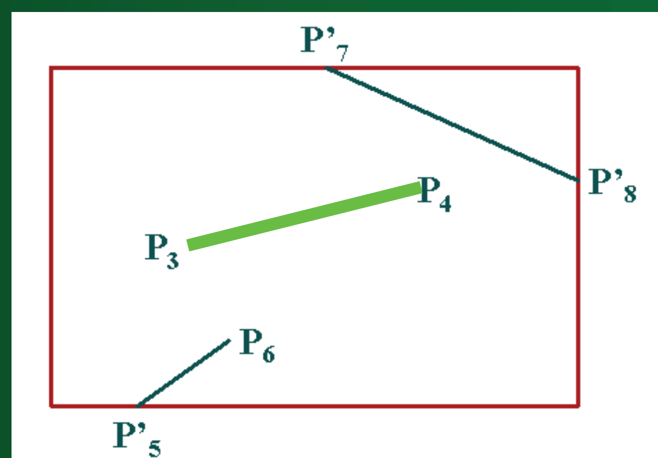
2. Determine whether line lies completely outside the clipping window.

3. Perform intersection calculation with one or more clipping boundaries.

Line Clipping

- A line with both endpoints inside all clipping boundaries is **saved** ($\overline{P_3 P_4}$)

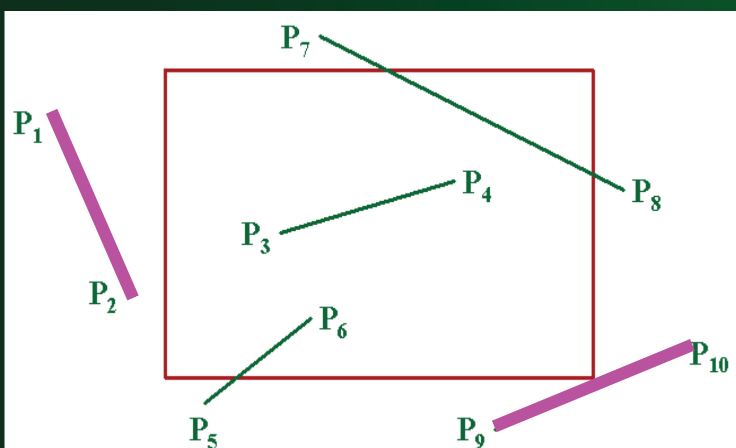


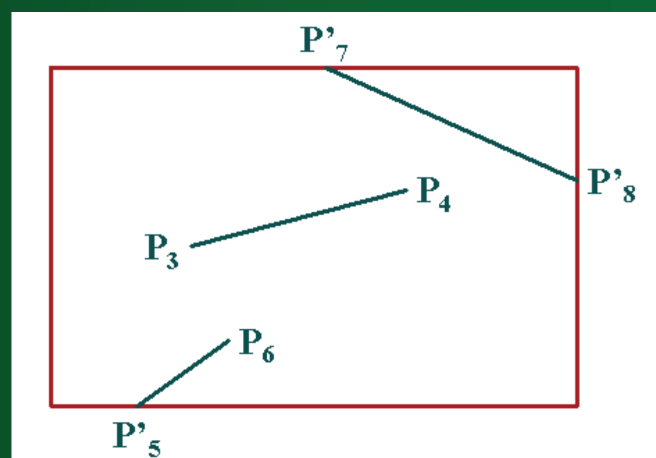


Before Clipping **After Clipping**

Line Clipping

- A line with both endpoints outside all clipping boundaries is **reject** ($\overline{P_1P_2}$ & $\overline{P_9P_{10}}$)

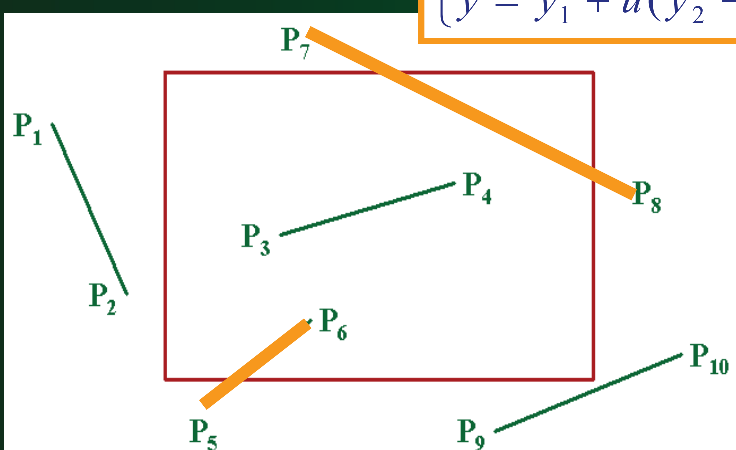




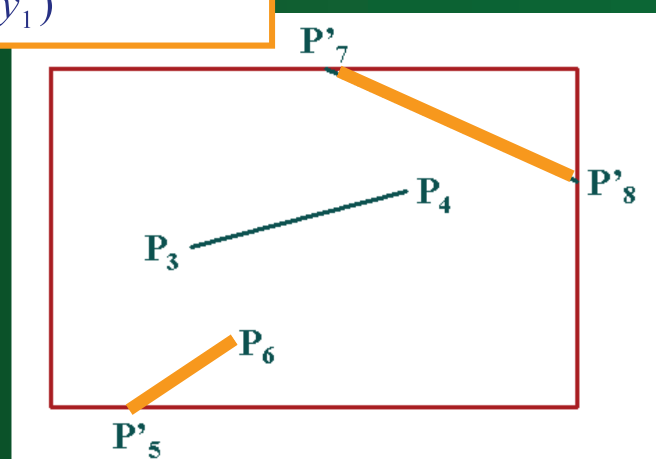
Before Clipping **After Clipping**

- If one or both endpoints outside the clipping rectangular, the **parametric representation** could be used to determine values of parameter **u** for intersection with the clipping boundary coordinates.

$$\begin{cases} x = x_1 + u(x_2 - x_1) \\ \qquad\qquad\qquad\qquad 0 \le u \le 1 \\ y = y_1 + u(y_2 - y_1) \end{cases}$$



Before Clipping



After Clipping

Line Clipping

1. If the value of u is outside the range 0 to 1: The line dose not enter the interior of the window at that boundary.
2. If the value of u is within the range 0 to 1, the line segment does cross into the clipping area.

- Clipping line segments with these parametric tests requires a good deal of computation, and faster approaches to clipper are possible.

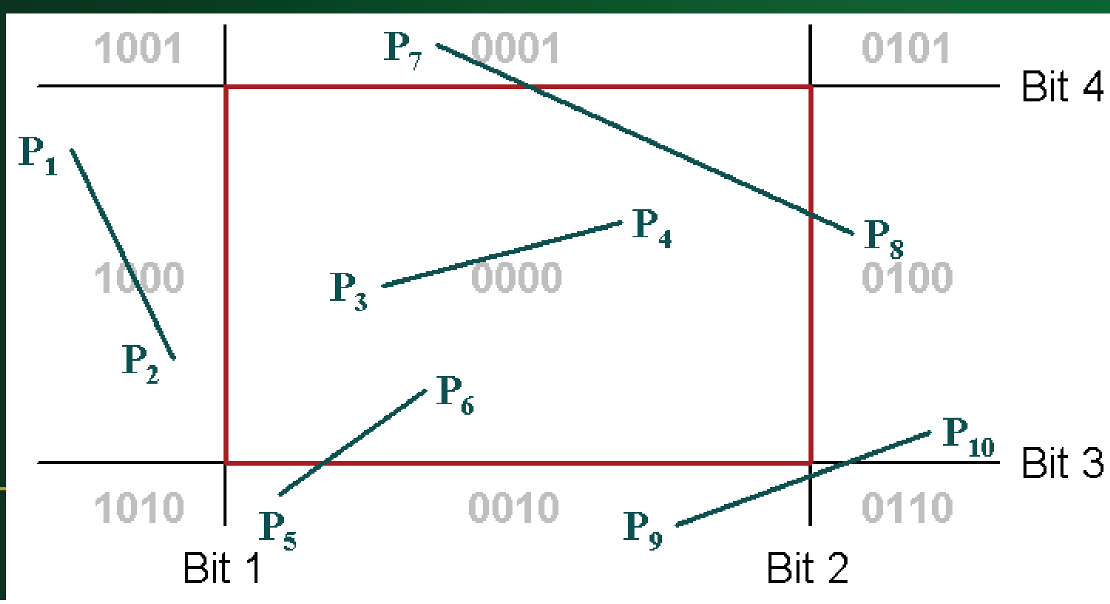
Cohen Sutherland Line Clipping

Cohen Sutherland Line Clipping

- The method speeds up the processing of line segments by performing **initial tests** that reduce the number of intersections that must be calculated.

Cohen Sutherland Line Clipping

- Every line endpoint is assigned a four digit binary code, called ***region code***, that identifies the location of the point relative to the boundaries of the clipping rectangle.



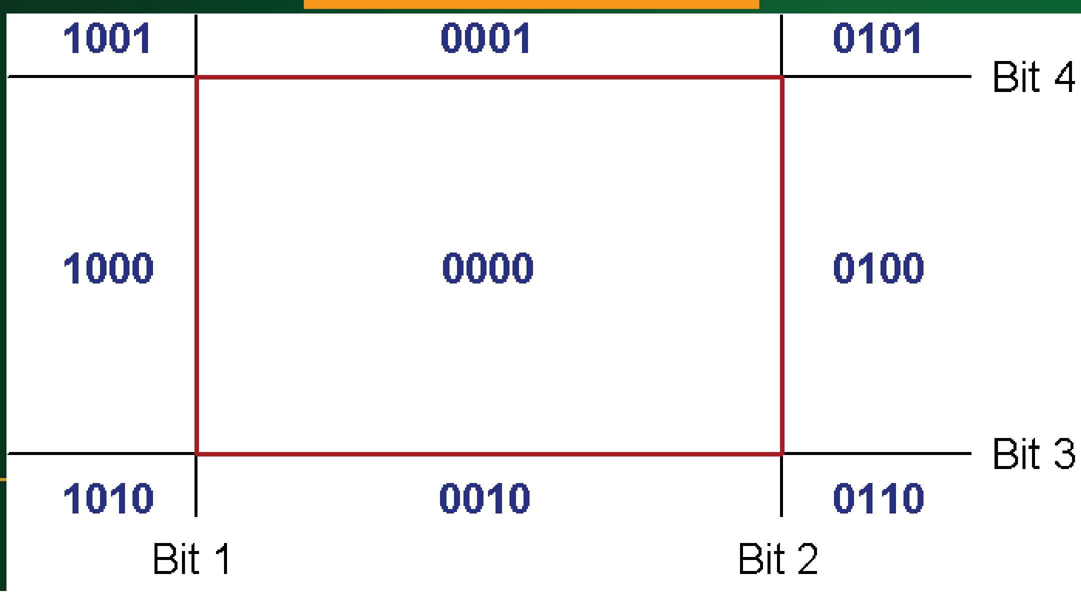
- Each bit position in the region code is used to indicate one of the four relative coordinate positions of the point with respect to the clip window.

Bit 1: Left

Bit 2: Right

Bit 3: Below

Bit 4: Above



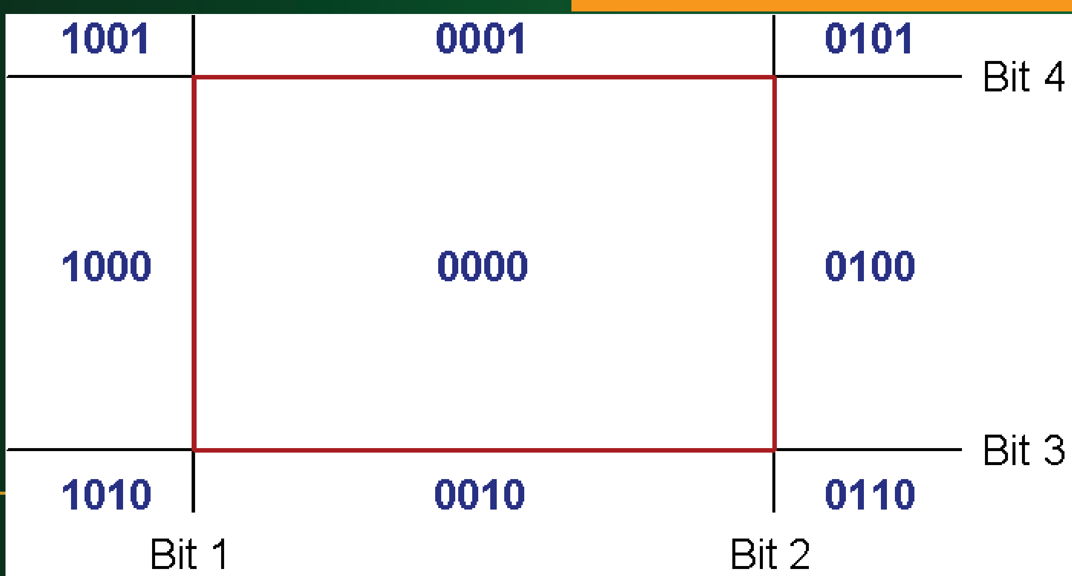
- Bit values in the region code are determined by comparing endpoint coordinates values (x,y) to the clip boundaries. Bit 1 is set to 1 if $x < xw_{\min}$

Bit 1: $sign(xw_{\min} - x)$

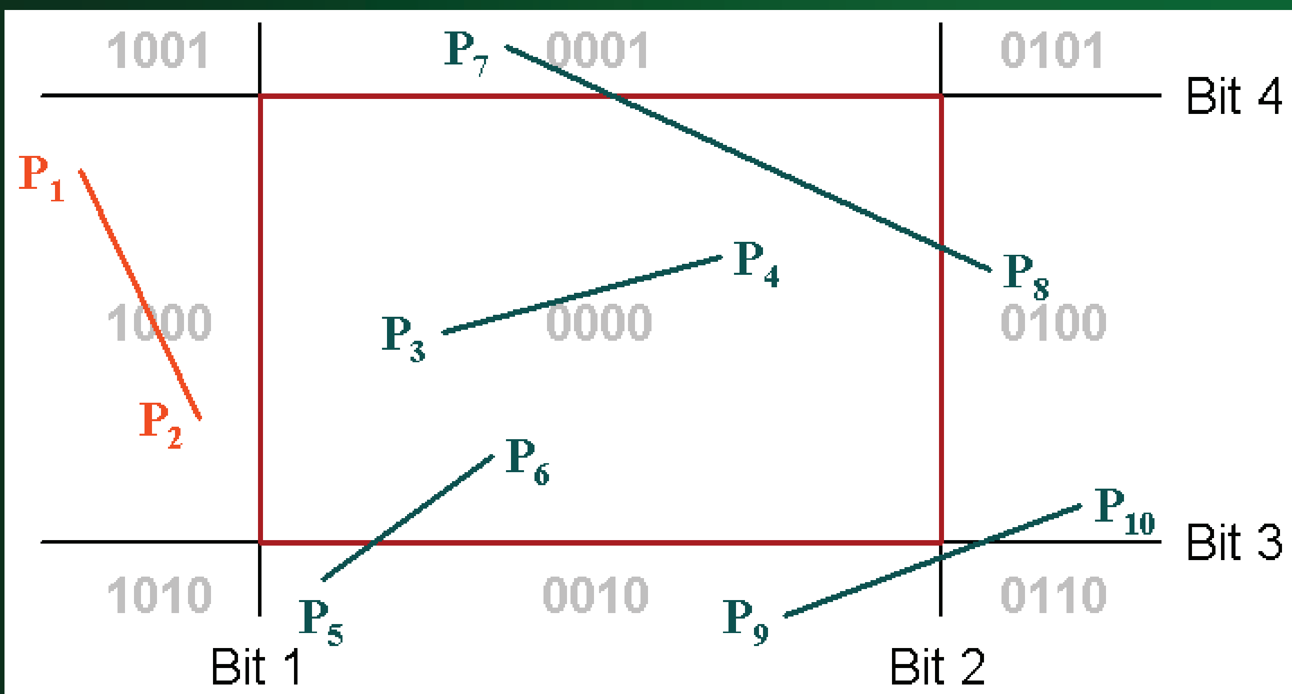
Bit 2: $sign(x - xw_{\max})$

Bit 3: $sign(yw_{\min} - y)$

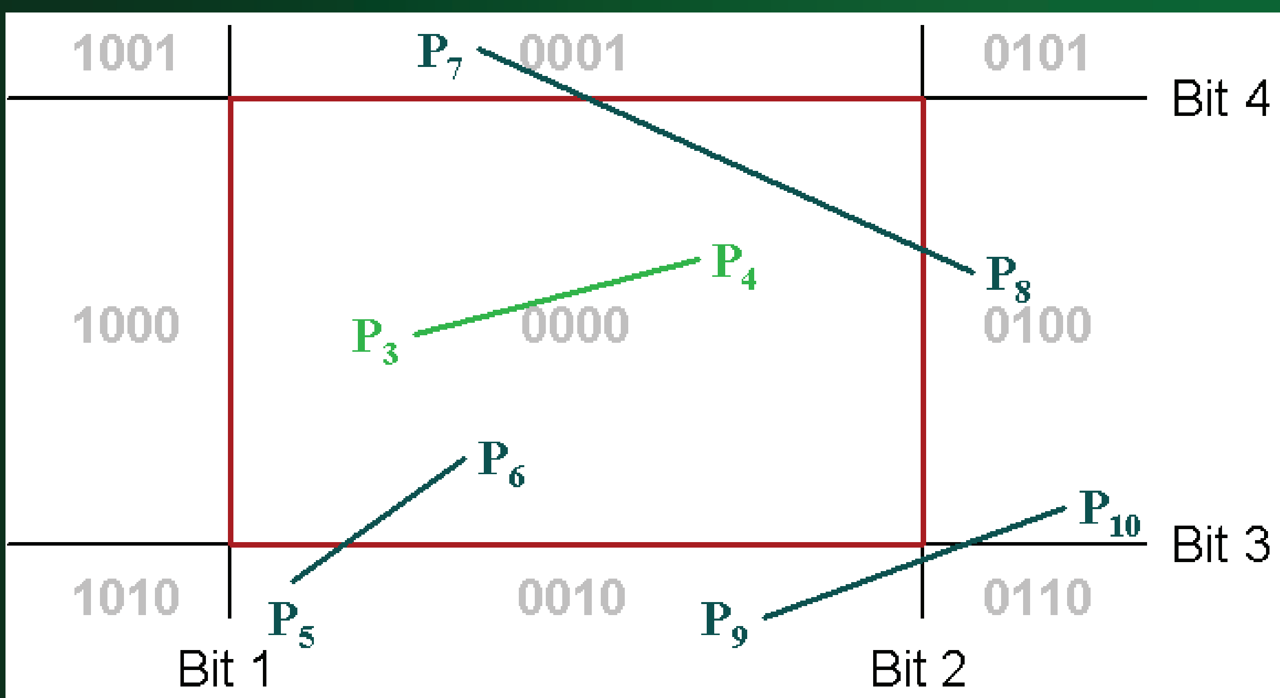
Bit 4: $sign(y - yw_{\max})$



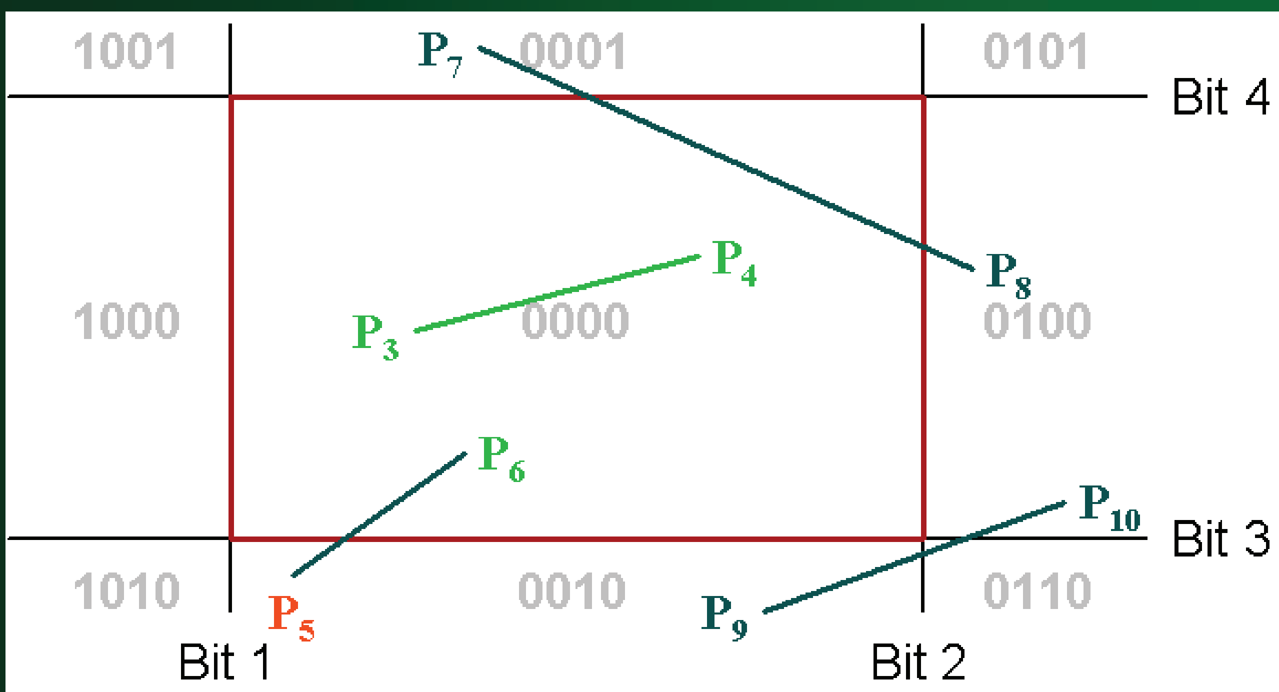
- Once we have established region codes for all line endpoints, we can quickly determine lines are completely outside or inside the clip window.



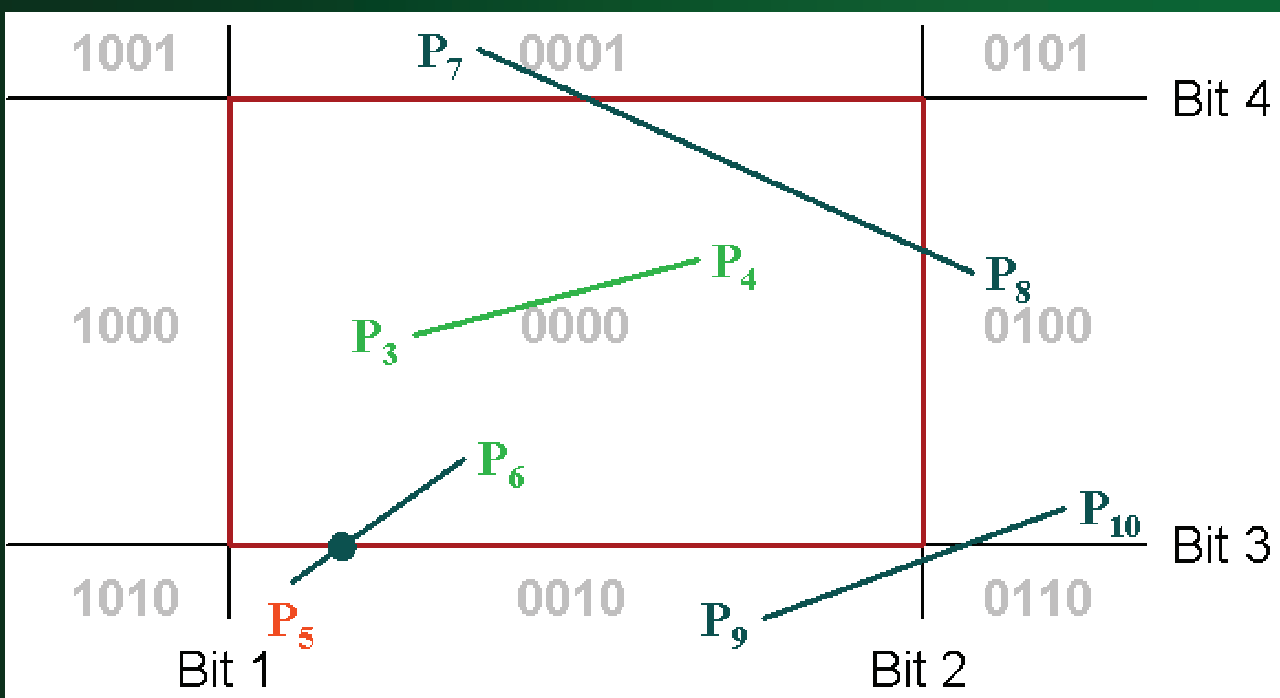
- Once we have established region codes for all line endpoints, we can quickly determine lines are completely outside or inside the clip window.



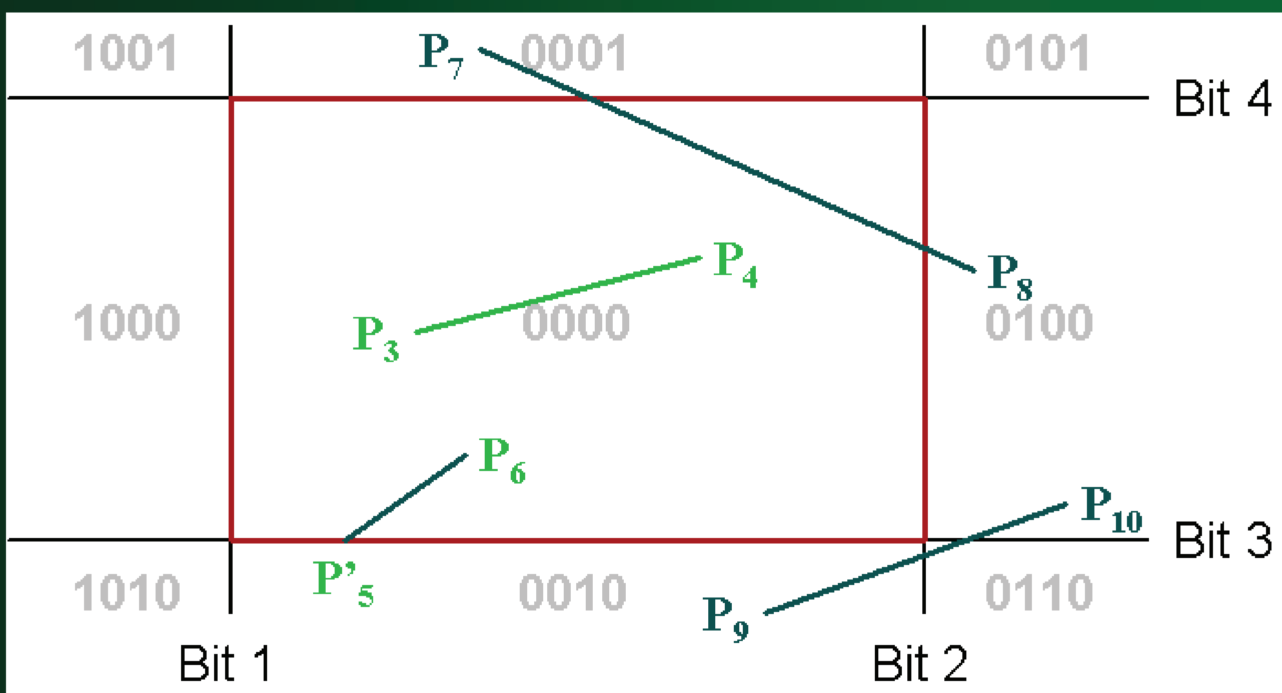
- Lines that cannot be identified as completely inside or outside a clip window are checked for intersection with boundaries.



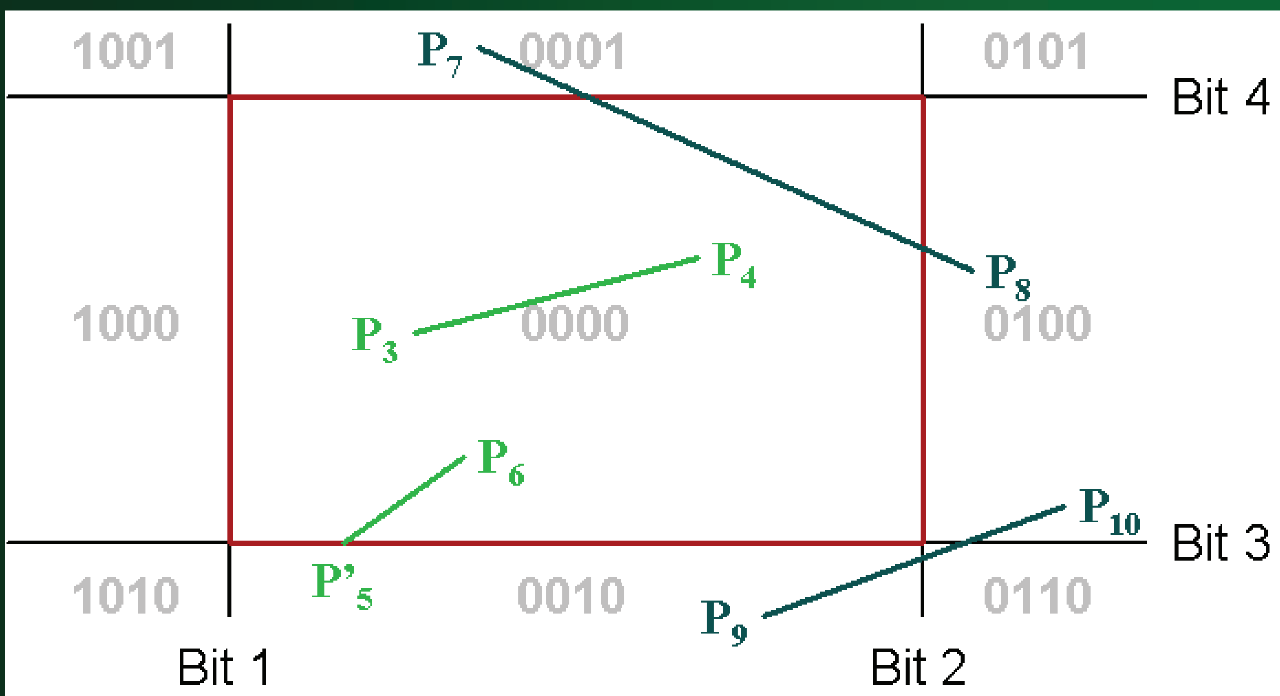
- Lines that cannot be identified as completely inside or outside a clip window are checked for intersection with boundaries.



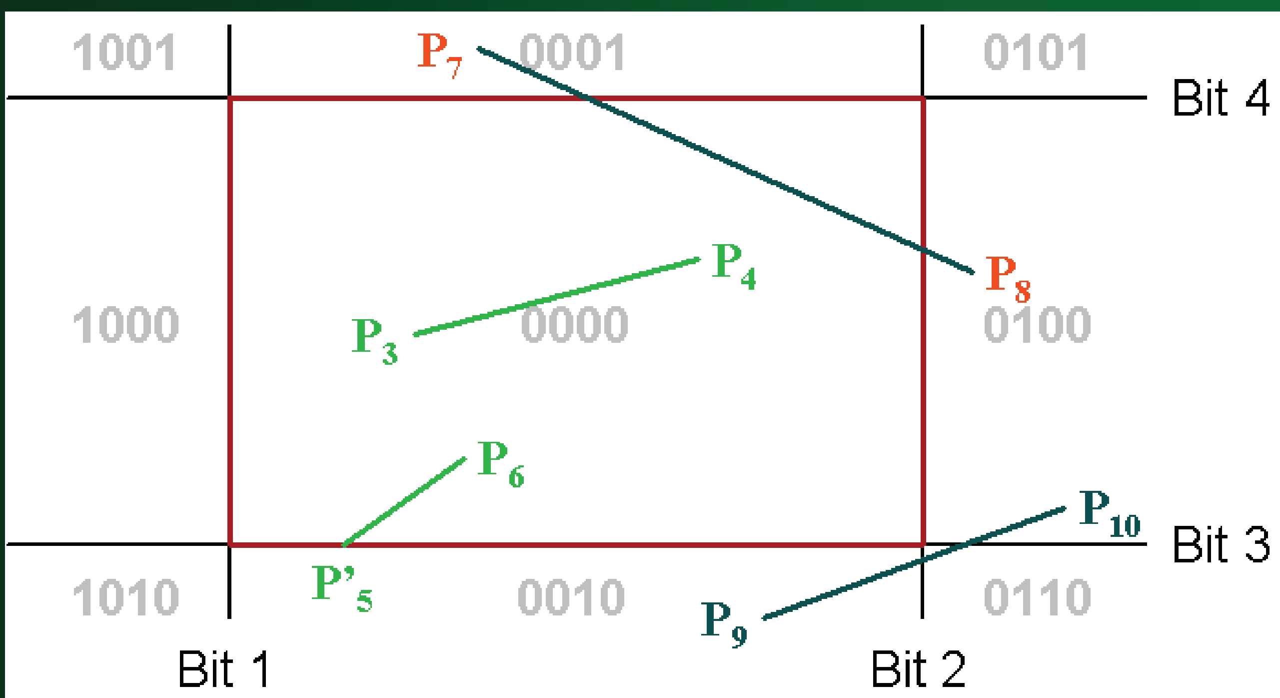
- Lines that cannot be identified as completely inside or outside a clip window are checked for intersection with boundaries.



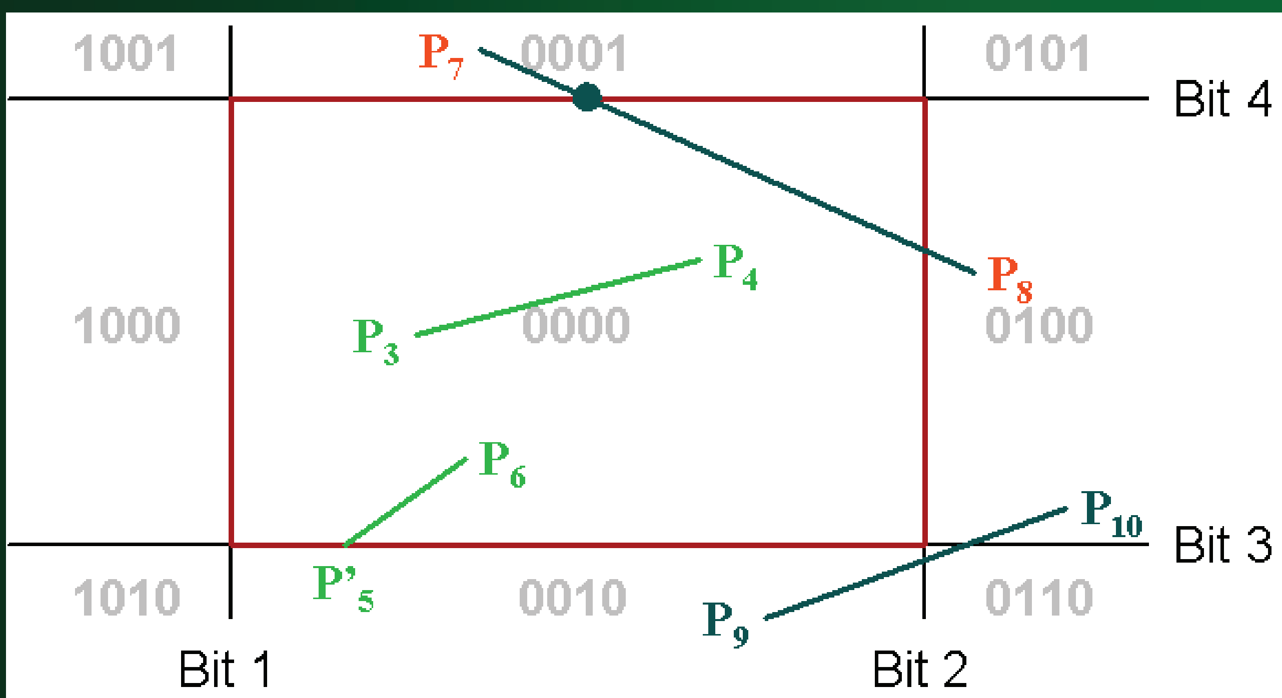
- Lines that cannot be identified as completely inside or outside a clip window are checked for intersection with boundaries.



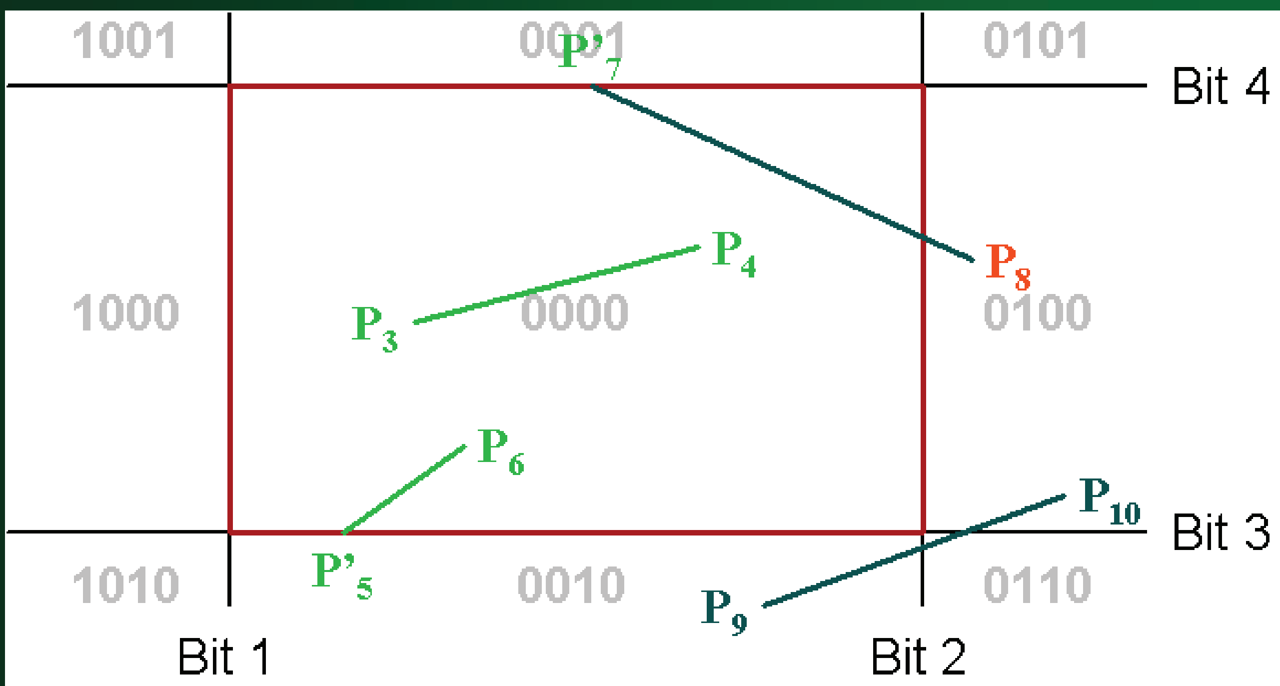
- Lines that cannot be identified as completely inside or outside a clip window are checked for intersection with boundaries.



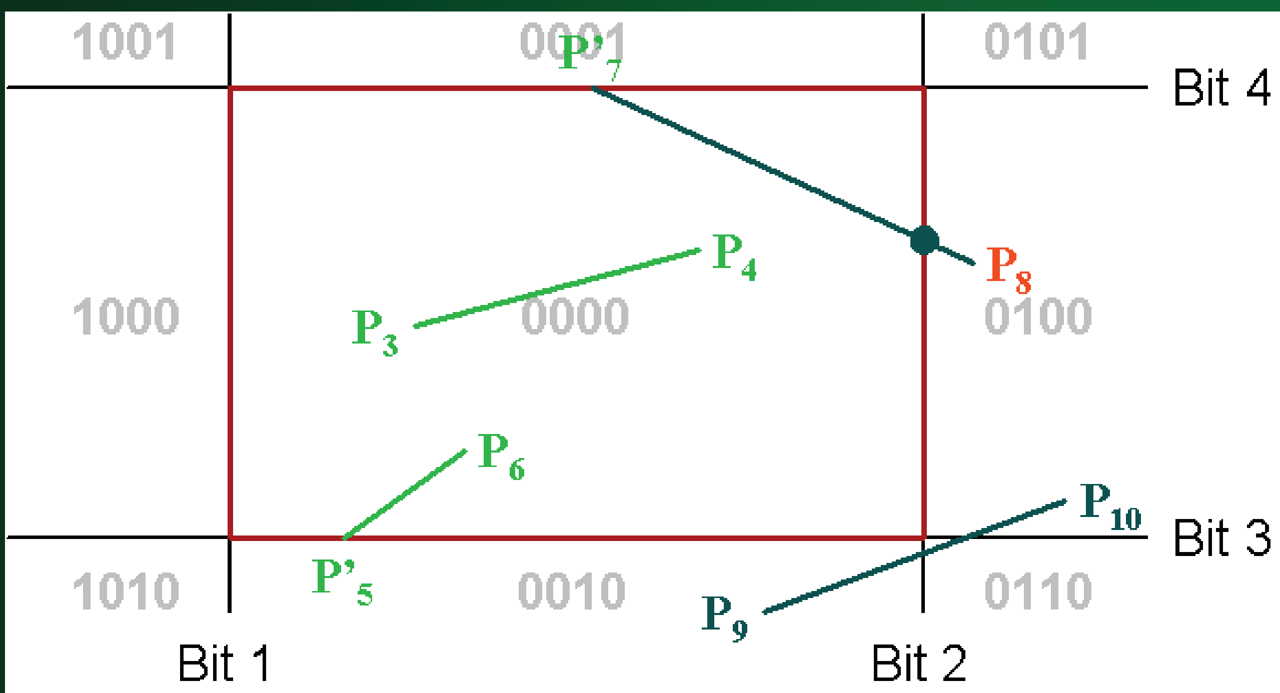
- Lines that cannot be identified as completely inside or outside a clip window are checked for intersection with boundaries.



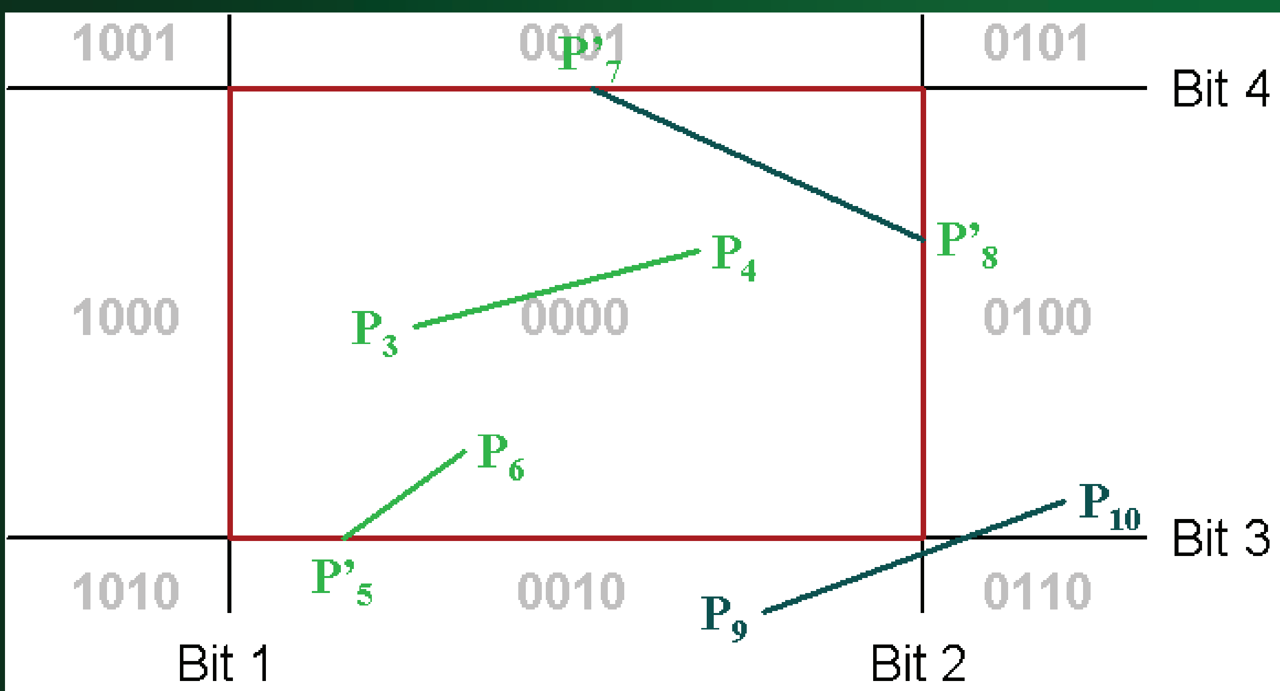
- Lines that cannot be identified as completely inside or outside a clip window are checked for intersection with boundaries.



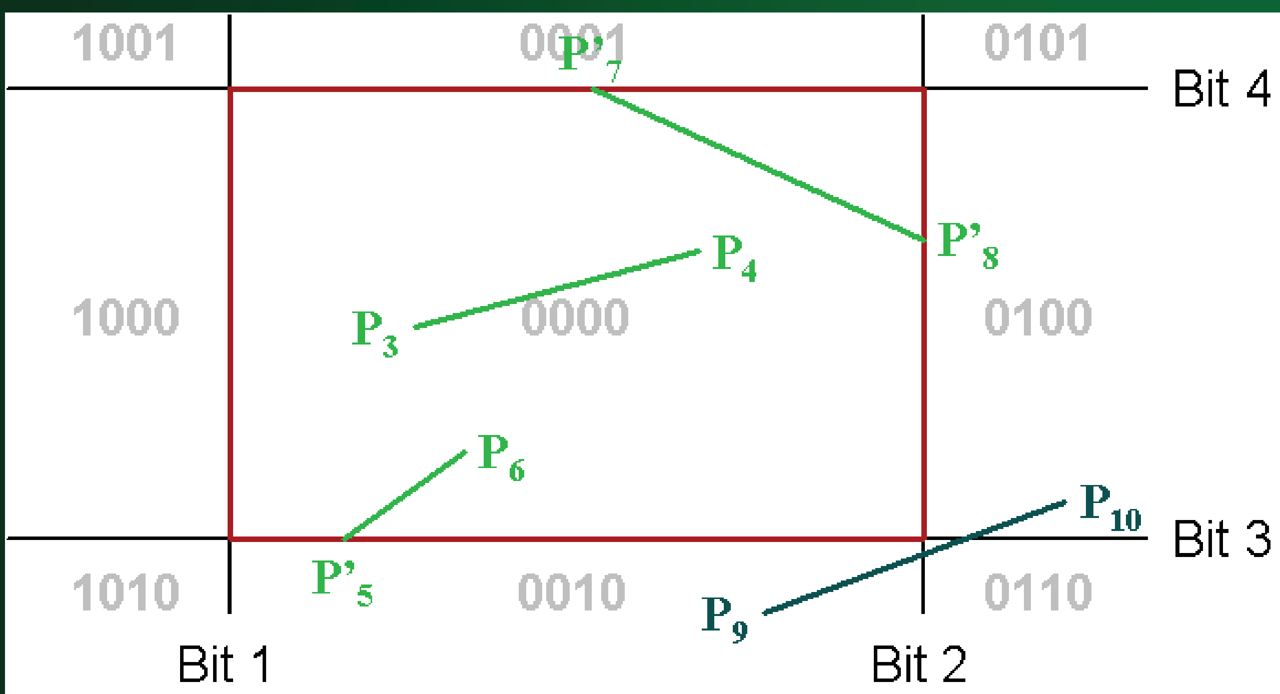
- Lines that cannot be identified as completely inside or outside a clip window are checked for intersection with boundaries.



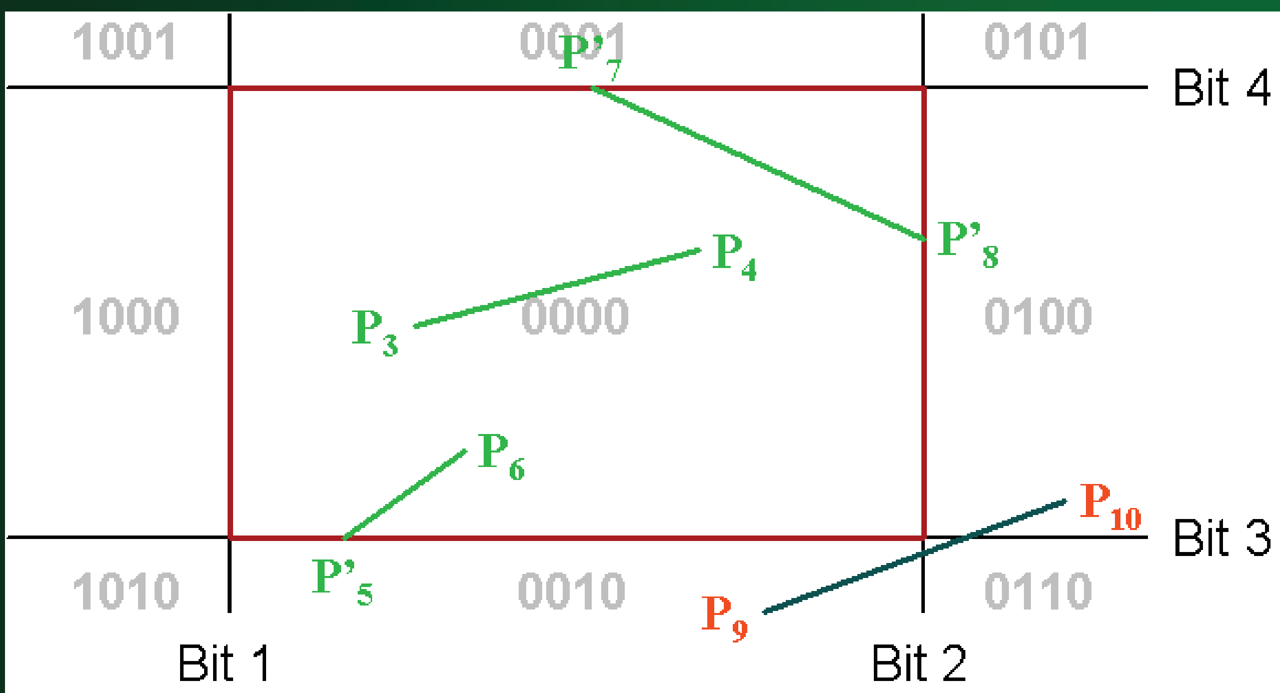
- Lines that cannot be identified as completely inside or outside a clip window are checked for intersection with boundaries.



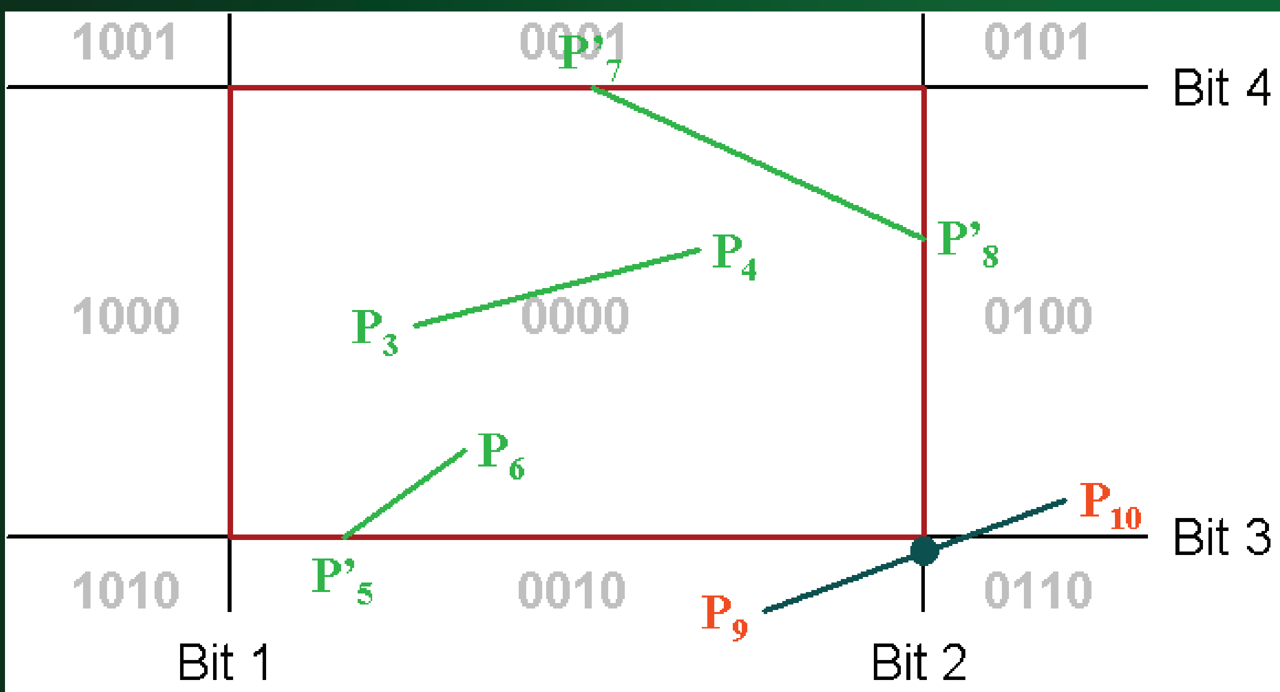
- Lines that cannot be identified as completely inside or outside a clip window are checked for intersection with boundaries.



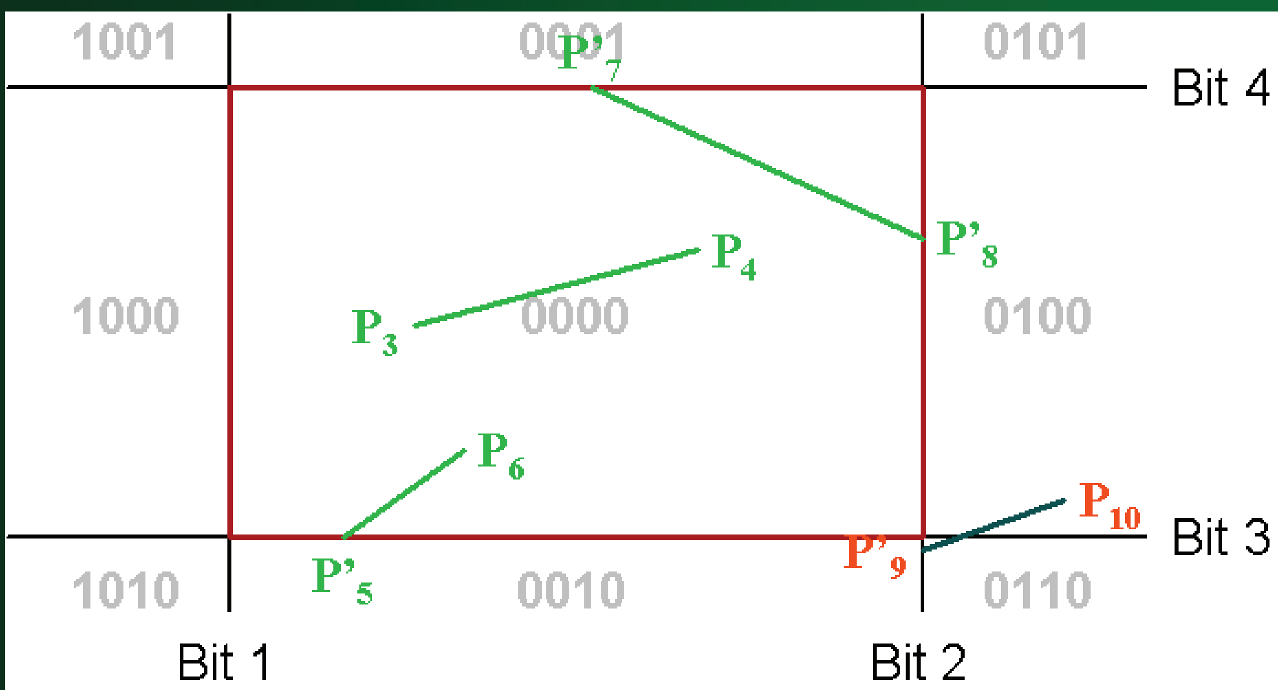
- Lines that cannot be identified as completely inside or outside a clip window are checked for intersection with boundaries.



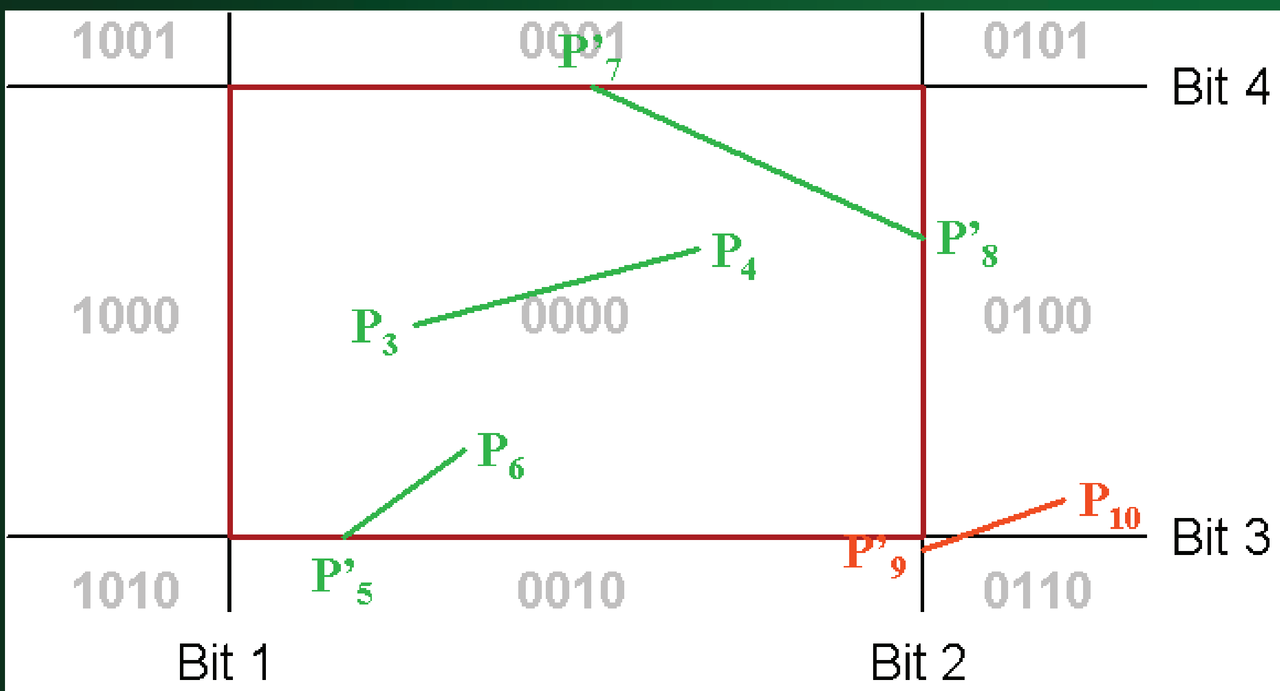
- Lines that cannot be identified as completely inside or outside a clip window are checked for intersection with boundaries.



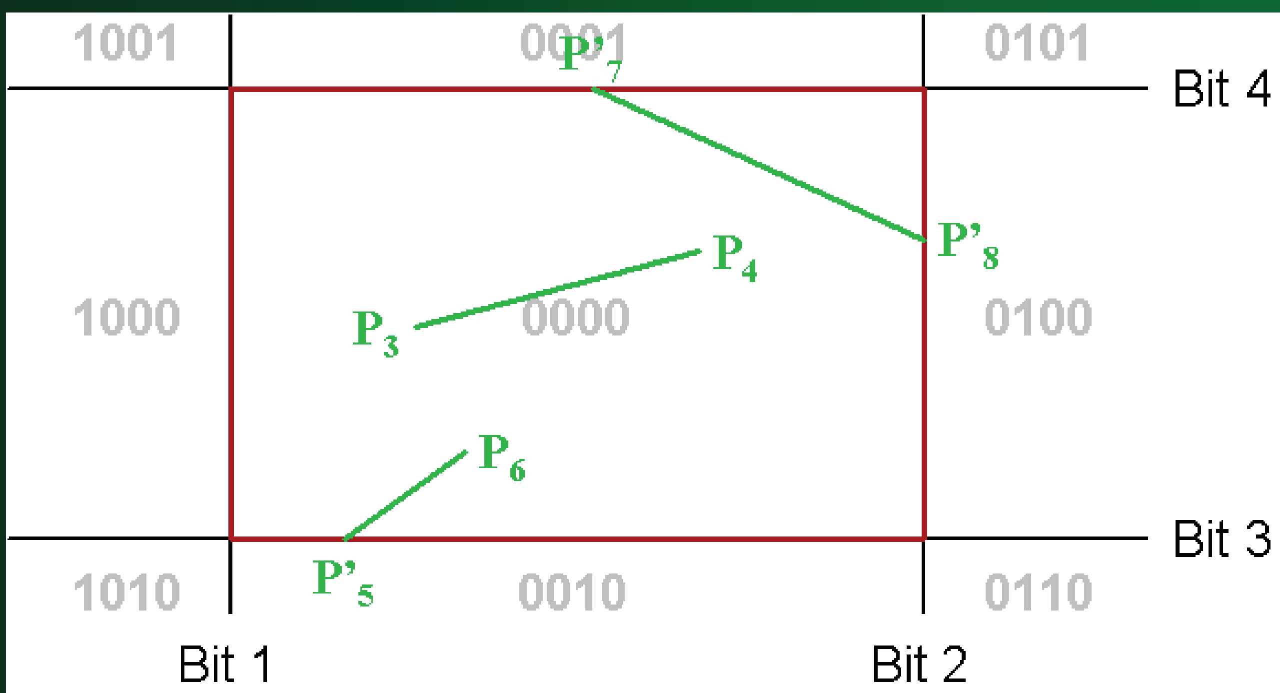
- Lines that cannot be identified as completely inside or outside a clip window are checked for intersection with boundaries.



- Lines that cannot be identified as completely inside or outside a clip window are checked for intersection with boundaries.



- Lines that cannot be identified as completely inside or outside a clip window are checked for intersection with boundaries.



Cohen Sutherland Line Clipping

- Intersection points with a clipping boundary can be calculated using the slope-intercept form of the line equation.

$$m = (y_2 - y_1)/(x_2 - x_1)$$

$$y = y_1 + m(x - x_1) \qquad x = xw_{\min} \quad x = xw_{\max}$$

$$x = x_1 + \frac{y - y_1}{m} \qquad y = yw_{\min} \quad y = yw_{\max}$$

Liang barsky Clipping

Liang barsky Clipping

- **Liang barsky Clipping:** Faster line clippers, that are based on analysis of the parametric of a line segment:

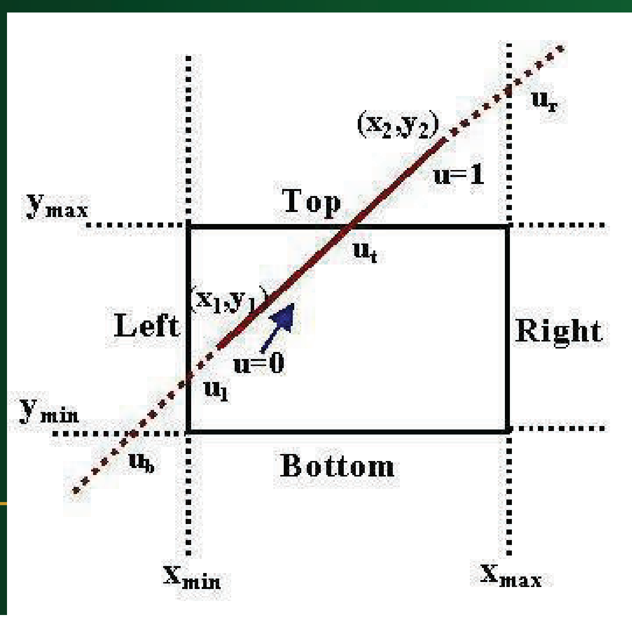
$$\begin{cases} x = x_1 + u\Delta x \\ \\ y = y_1 + u\Delta y \end{cases} \qquad 0 \le u \le 1$$

$$\Delta x = x_2 - x_1$$

$$\Delta y = y_2 - y_1$$

Liang barsky Clipping

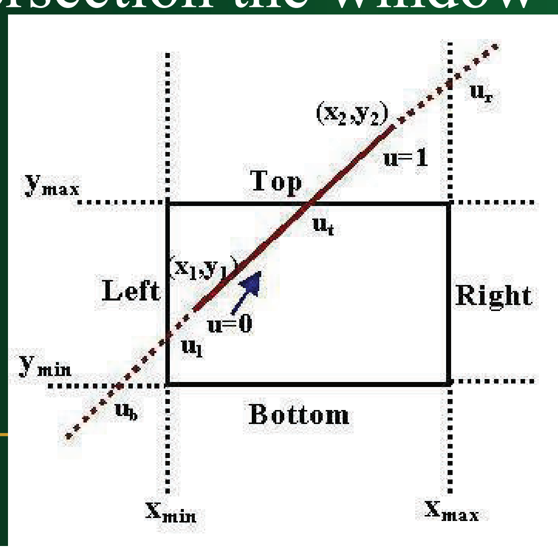
- When we traverse along the extended line with u increasing from $-\infty$ to ∞ ,
- we first move from the outside to the inside of the clipping window's two boundary lines (bottom and left)
- Then move from the inside to the outside of the other two boundary lines (top and right)



Liang barsky Clipping

$$u_1 \le u_2 \qquad u_1 = Max(0, u_1, u_b) \qquad u_2 = Min(1, u_t, u_r)$$

- **u_l:** intersection the window's **left**
- **u_b:** intersection the window's **bottom**
- **u_l:** intersection the window's **right**
- **u_r:** intersection the window's **top**



Liang barsky Clipping

- Point (x,y) inside the clipping window

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

Rewrite the four inequalities as: $up_k \le q_k, \quad k = 1,2,3,4$

$$\begin{array}{lll}
p_1 = -\Delta x, & q_1 = x_1 - x_{\min} & (left) \\
p_2 = \Delta x, & q_2 = xw_{\max} - x_1 & (right) \\
p_3 = -\Delta y, & q_3 = y_1 - y_{\min} & (bottom) \\
p_4 = \Delta y, & q_4 = yw_{\max} - y_1 & (top)
\end{array}$$

$$
\begin{array}{lll}
p_1 = -\Delta x, & q_1 = x_1 - x_{\min} & (left) \\
p_2 = \Delta x, & q_2 = xw_{\max} - x_1 & (right) \\
p_3 = -\Delta y, & q_3 = y_1 - y_{\min} & (bottom) \\
p_4 = \Delta y, & q_4 = yw_{\max} - y_1 & (top)
\end{array}
$$

- If $p_k = 0$, the line is parallel to the boundary:

if $q_k < 0$ the line is completely outside (can be eliminated)

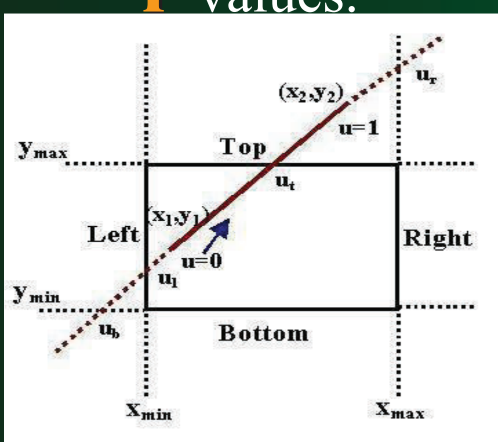
if $q_k \geq 0$ the line is completely inside (need further consideration)

- If $p_k < 0$ the extended line proceeds from the outside to the inside.
- If $p_k > 0$ the extended line proceeds from the inside to the outside.
- When $p_k \neq 0$, u corresponding to the intersection point is q_k / p_k

Liang barsky Clipping

- **A four step process** for finding the visible portion of the line:

1. If $p_k = 0$ and $q_k < 0$ for any **k**, eliminate the line and stop, Otherwise proceed to the next step.
2. For all k such that $p_k < 0$ calculate $r_k = q_k / p_k$. Let u_1 be the maximum of the set containing **0** and the calculated **r** values.

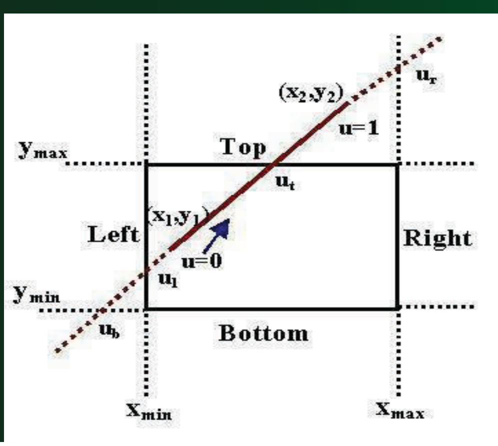


$$
\begin{array}{lll}
p_1 = -\Delta x, & q_1 = x_1 - x_{\min} & (left) \\
p_2 = \Delta x, & q_2 = xw_{\max} - x_1 & (right) \\
p_3 = -\Delta y, & q_3 = y_1 - y_{\min} & (bottom) \\
p_4 = \Delta y, & q_4 = yw_{\max} - y_1 & (top)
\end{array}
$$

Liang barsky Clipping

- **A four step process …**

3. For all **k** such that $p_k > 0$, calculate $r_k = q_k / p_k$. Let u_2 be the minimum of the set containing **1** and the calculated **r** values.
4. If $u_1 > u_2$, eliminate the line since it is completely outside the clipping window, Otherwise, use u_1 and u_2 to calculate the endpoints of the clipped line.



$$
\begin{array}{lll}
p_1 = -\Delta x, & q_1 = x_1 - x_{min} & (left) \\
p_2 = \Delta x, & q_2 = xw_{max} - x_1 & (right) \\
p_3 = -\Delta y, & q_3 = y_1 - y_{min} & (bottom) \\
p_4 = \Delta y, & q_4 = yw_{max} - y_1 & (top)
\end{array}
$$