ICT 4203

Computer Graphics and Animation

Lecture 09

2D Transformation

Md. Mahmudur Rahman
Lecturer
Institute of Information Technology

Lecture Outlines

- Transformation
- Types of 2D Transformation
  - ✓Geometric
  - ✓Coordinate
  - ✓Composite
  - ✓Instance
- Matrix Revisit
  - ✓Use of Matrix in 2D Transformation

What is Transformation?

- The geometrical changes of an object from a current state to modified state is referred to as Transformation. It allows us to change the -
  - ✓ Position;
  - ✓ Size;
  - ✓ Orientation of the objects.
- Why it is needed?
  - ✓ To manipulate the initially created object;
  - ✓ To display the modified object without having to redraw it.

Two-Dimensional Transformation

- There are two complementary points of view for describing object movement -
  - ✓The first is that the object itself is moved relative to a stationary coordinate system or background [Geometric Transformations].
  - ✓The second point of view holds that the object is held stationary while the coordinate system is moved relative to the object [Coordinate Transformations].

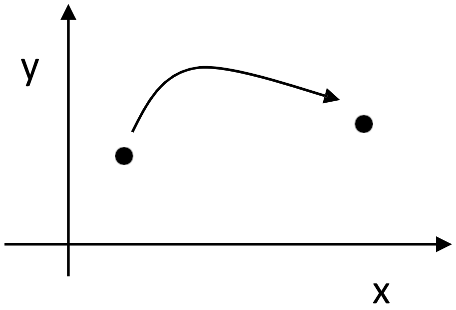
Two Dimensional Transformation

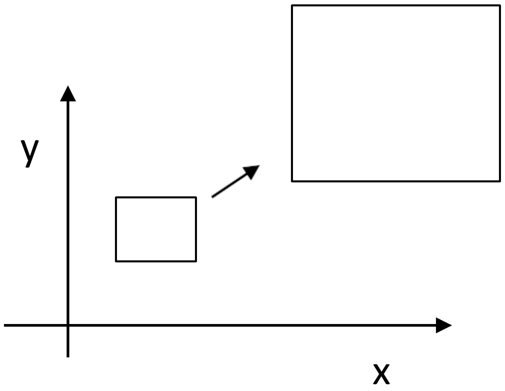
- An example involves the motion of an automobile against a scenic background.
  - ✓We can simulate this by moving the automobile while keeping the background fixed **[Geometric Transformations].**
  - ✓We can also keep the automobile fixed while moving the background scenery **[Coordinate Transformations].**

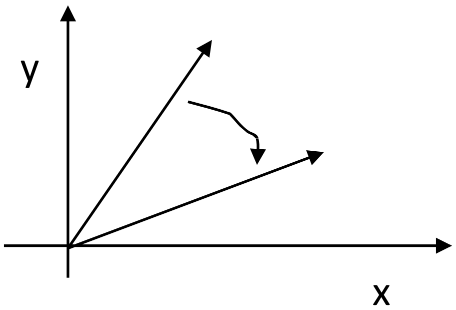
2D Transformation

- Two ways -
  - ❑ Object Transformation -
    - ✓Alter the coordinate of an object;
    - ✓Translation, rotation, scaling etc.
    - ✓Coordinate system unchanged.
  - ❑ Coordinate Transformation -
    - ✓Produce a different coordinate system.

Examples of 2D Transformations







Geometric Transformations

- Let us impose a coordinate system on a plane.
- An object *Obj* in the plane can be considered as a set of points.
- Every object point `P` has coordinates `(x, y)`, and so the object is the sum total of all its coordinate points.
- If the object is moved to a new position, it can be regarded as a new object `Obj'`, all of whose coordinate point `P'` can be obtained from the original points `P` by the application of a geometric transformation.

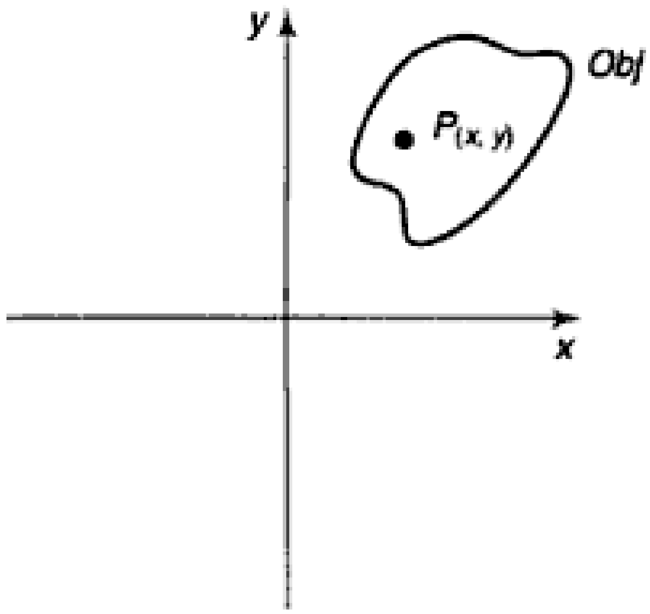


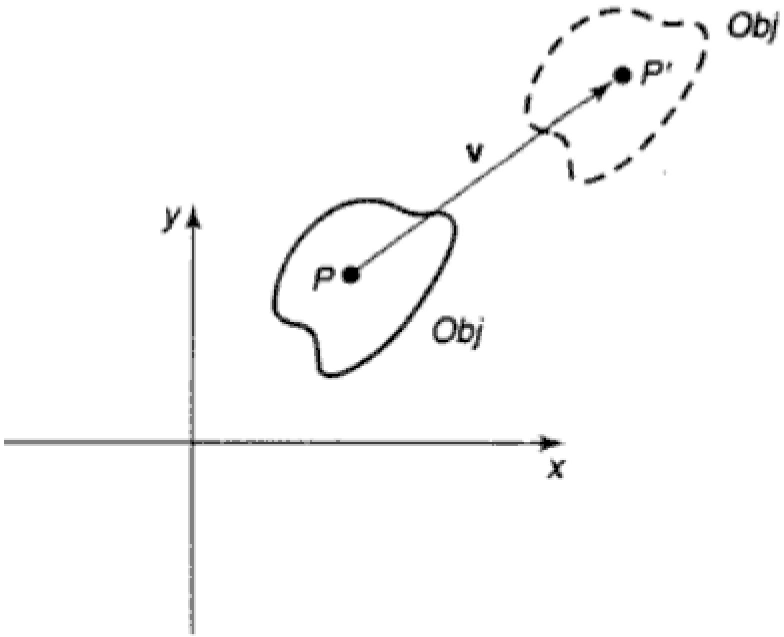
Fig. 4.1

Geometric Transformations

- Translation;
- Rotation about the Origin;
- Scaling with Respect to the Origin;
- Mirror Reflection about an Axis.

Translation

- In translation, an object is displaced a given distance and direction from its original position.
- If the displacement is given by the vector $v = t_x\mathbf{I} + t_y\mathbf{j}$ the new object point $P'(x', y')$ can be found by applying the transformation T_v to $P(x, y)$



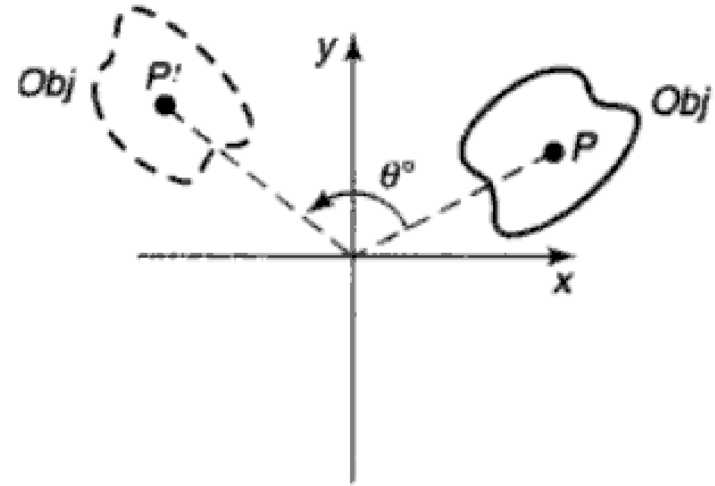
Now,

$$P' = T_v(P)$$

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

Rotation about the Origin

- In rotation, the object is rotated $\theta°$ about the origin.
- The convention is that the direction of rotation is counterclockwise if θ is a positive angle and clockwise if θ is a negative angle.
- The transformation of rotation R_θ is -



$$P' = R_\theta(P)$$

where $x' = x\cos(\theta) - y\sin(\theta)$

and $y' = x\sin(\theta) + y\cos(\theta)$

$\cos(\alpha) = \frac{x}{r}$

$x = r\cos(\alpha)$ -----------(i)

$\cos(\alpha+\beta) = \frac{x'}{r}$

$x' = r\cos(\alpha+\beta)$

$x' = r\,[\cos(\alpha)\cos(b) - \sin a \sin b]$

$x' = r\cos(\alpha)\cos(b) - r\sin a \sin b$

$x' = x\cos(b) - y\sin(b)$

$\sin(\alpha) = \frac{y}{r}$

$y = r\sin(\alpha)$ -----------(ii)

$\sin(a+b) = \frac{y'}{r}$

$Y' = r\sin(\alpha+\beta)$

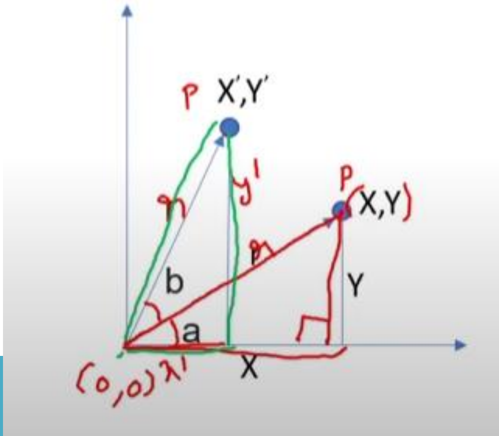
$y' = r\,[\sin a \cos(b) - \cos a \sin b]$

$y' = r\sin a \cos(b) - r\cos a \sin b$

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

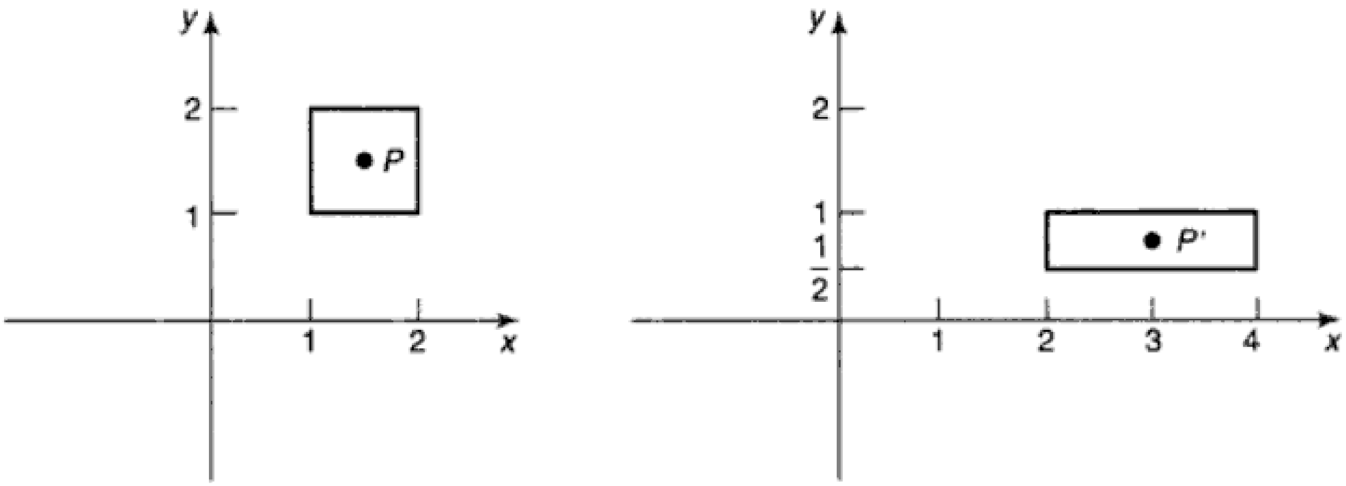
where $x' = x\cos(\theta) - y\sin(\theta)$

and $y' = x\sin(\theta) + y\cos(\theta)$



Scaling with Respect to the origin

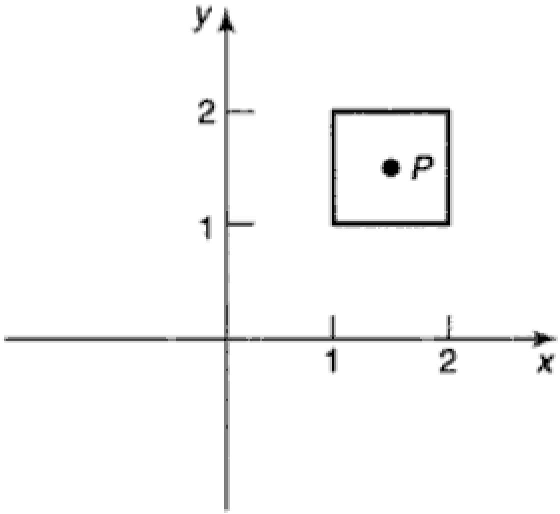
- Scaling is the process of expanding or compressing the dimension of an object.
- Positive scaling constants s_x and s_y are used to describe changes in length with respect to the x direction and y direction, respectively.
- A scaling constant greater than one indicates an expansion of length, and less than one, compression of length.



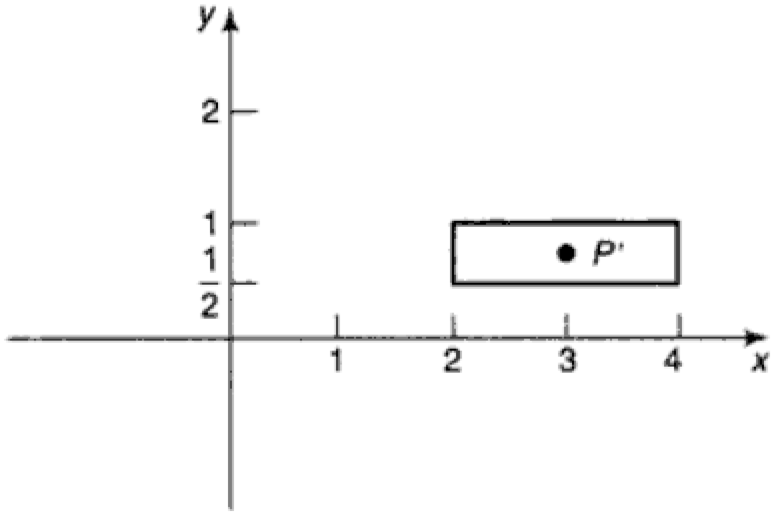
(a) (b)

Scaling with Respect to the origin

- The scaling transformation $S_{Sx,Sy}$ is given by $P' = S_{Sx,Sy}(P)$ where, $x' = s_x x$ and $y' = s_y y$.
- After a scaling transformation is performed, the new object is located at a different position relative to the origin.
- In fact, in a scaling transformation, the only point that remains fixed is the origin.



(a) Original Object



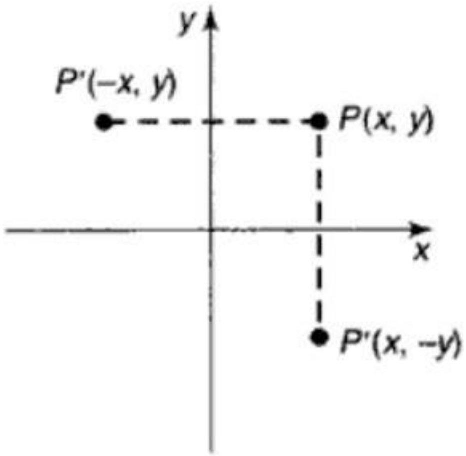
(b) Scaling factors s_x =2
Scaling factors s_y =1/2

Mirror Reflection about an Axis

- If either the x and y axis is treated as a mirror, the object has a mirror image or reflection.
- Since the reflection P' of an object point P is located the same distance from the mirror as P, the mirror reflection transformation M_x about the x-axis is given by $P' = M_x(P)$

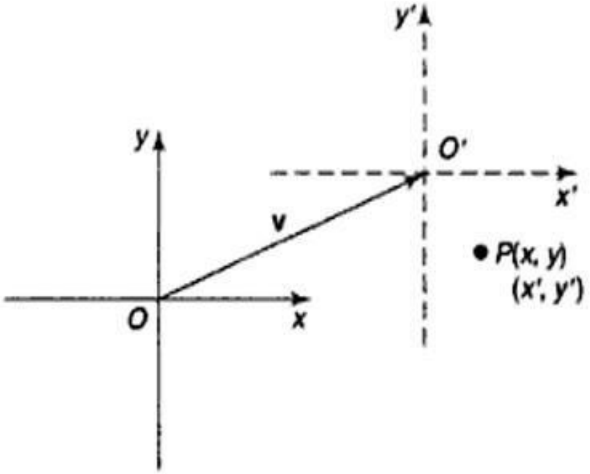
  where $x' = x$ and $y' = -y$.
- Similarly, the mirror reflection about the y-axis is $P' = M_y(P)$ y

  where, $x' = -x$ and $y' = y$.



Coordinate Transformations

- Suppose that we have two coordinate systems in the plane. The first system is located at origin O and has coordinates axes xy.
- The second coordinate system is located at origin O' and has coordinate axes x'y'.
- Now each point in the plane has two coordinate descriptions: (x,y) or (x',y'), depending on which coordinate system is used.
- If we think of the second system x'y' as arising from a transformation applied to the first system xy, we say that a coordinate transformation has been applied. We can describe this transformation by determining how the (x',y') coordinates of a point P are related to the (x,y) coordinates of the same point.



Coordinate Transformations

- Translation;
- Rotation about the Origin;
- Scaling with Respect to the Origin;
- Mirror Reflection about an axis.

Translation

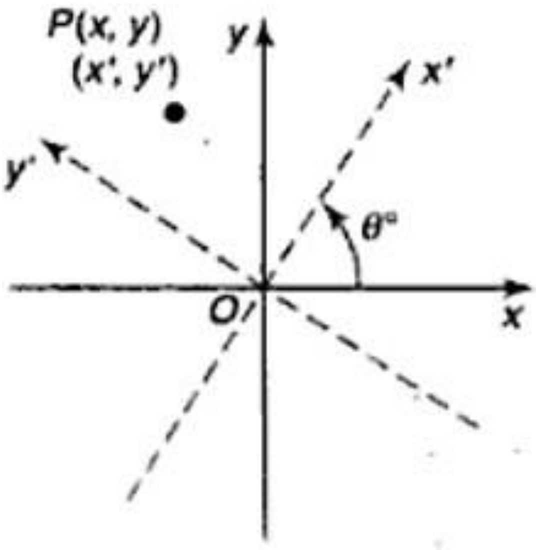
- If the xy coordinate system is displaced to a new position, where the direction and distance of the displacement is given by the vector v = t_x**I** + t_y**J**, the coordinates of a point in both systems are related by the translation transformation $\bar{T}_v$:

$$(x', y') = \bar{T}_v (x, y)$$

where $x' = x - t_x$ and $y' = y - t_y$

Rotation about the Origin

- The xy system is rotated θ° about the origin.
- Then the coordinates of a point in both systems are related by the rotation transformation $\bar{R}_\theta$:
- $(x', y') = \bar{R}_\theta(x, y)$
- $x' = x \cos(\theta) + y \sin(\theta)$
- $y' = - x \sin(\theta) + y \cos(\theta)$.

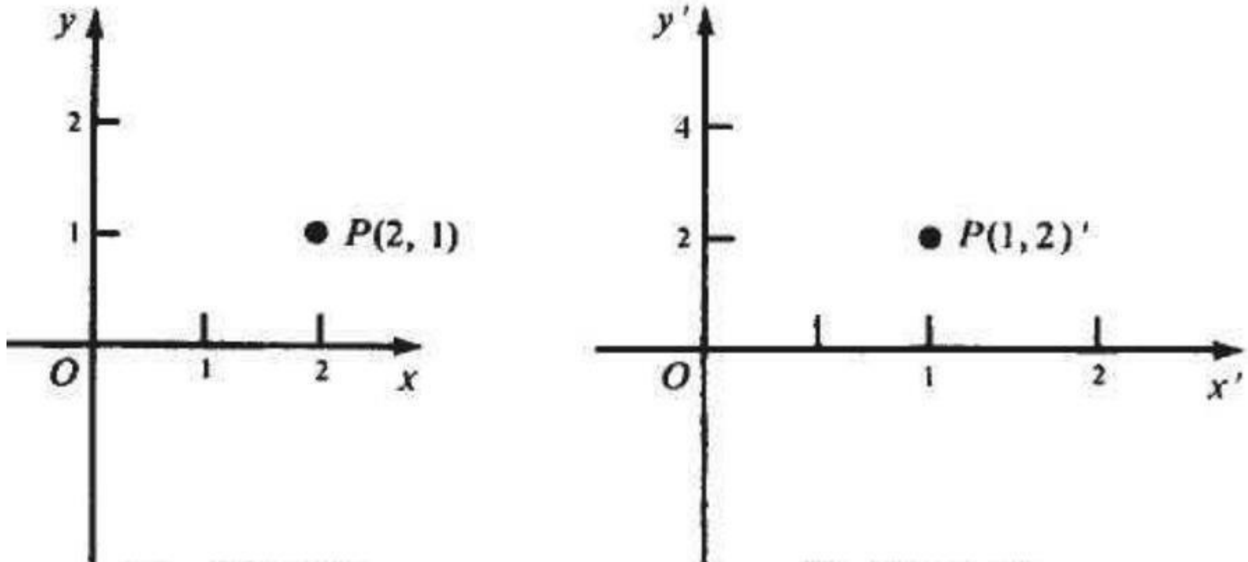


Scaling with Respect to the Origin

- Suppose that a new coordinate system is formed by leaving the origin and coordinate axes unchanged, but introducing different units of measurement along the x and y axes.
- If the new units are obtained from the old units by a scaling of s_x along the x axis and s_y along the y axis, the coordinates in the new system are related to coordinates in the old system through the scaling transformation Ss_xs_x :
- where $x' = \{1/s_x)x$ and $y' = \{1/s_y)y$.

Continue…

- Figure shows coordinate scaling transformation using scaling factors $s_x = 2$ and $s_y = ½$.



(a) Old units. (b) New units.

Mirror Reflection about an Axis

- If the new coordinate system is obtained by reflecting the old system about either x or y axis, the relationship between coordinates is given by the coordinate transformations M_x and M_y.

- Reflection about the x axis [Fig. (a)]:

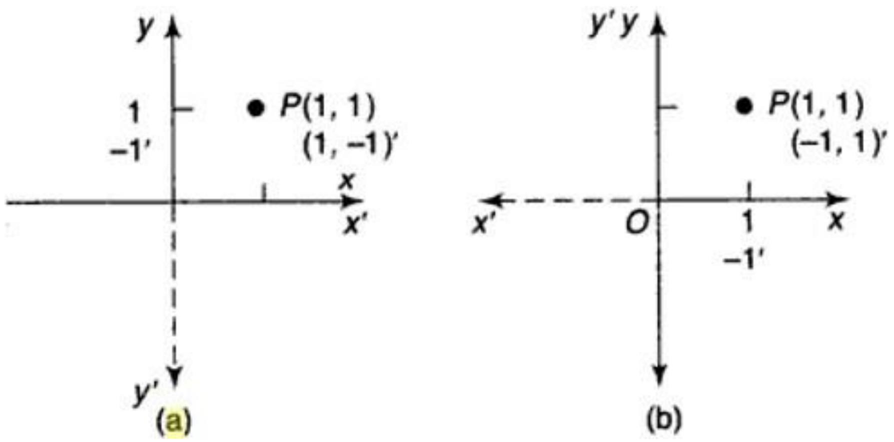
$$(x', y') = M_x(x, y);$$

  - where x' = x and y' = —y.

- Reflection about the y axis [Fig. (b)]:

$$(x',y') = M_y(x,y);$$

  - where x' = —x and y' = y.



Composite Transformation

- More complex geometric and coordinate transformations can be built from the basic transformations described above by using the process of composition of functions.
- For example, such operations as rotation about a point other than the origin or reflection about lines other than the axes can be constructed from the basic transformations.

Continue…

- If we want to apply a series of transformation T_1, T_2, T_3 to a set of points, we can do it like below-
  - Calculate, $\mathbf{T} = T_1 \times T_2 \times T_3$

    then $\mathbf{P'} = \mathbf{T} \times \mathbf{P}$
- This method saves large number of adds and multiplications.

Example - 01

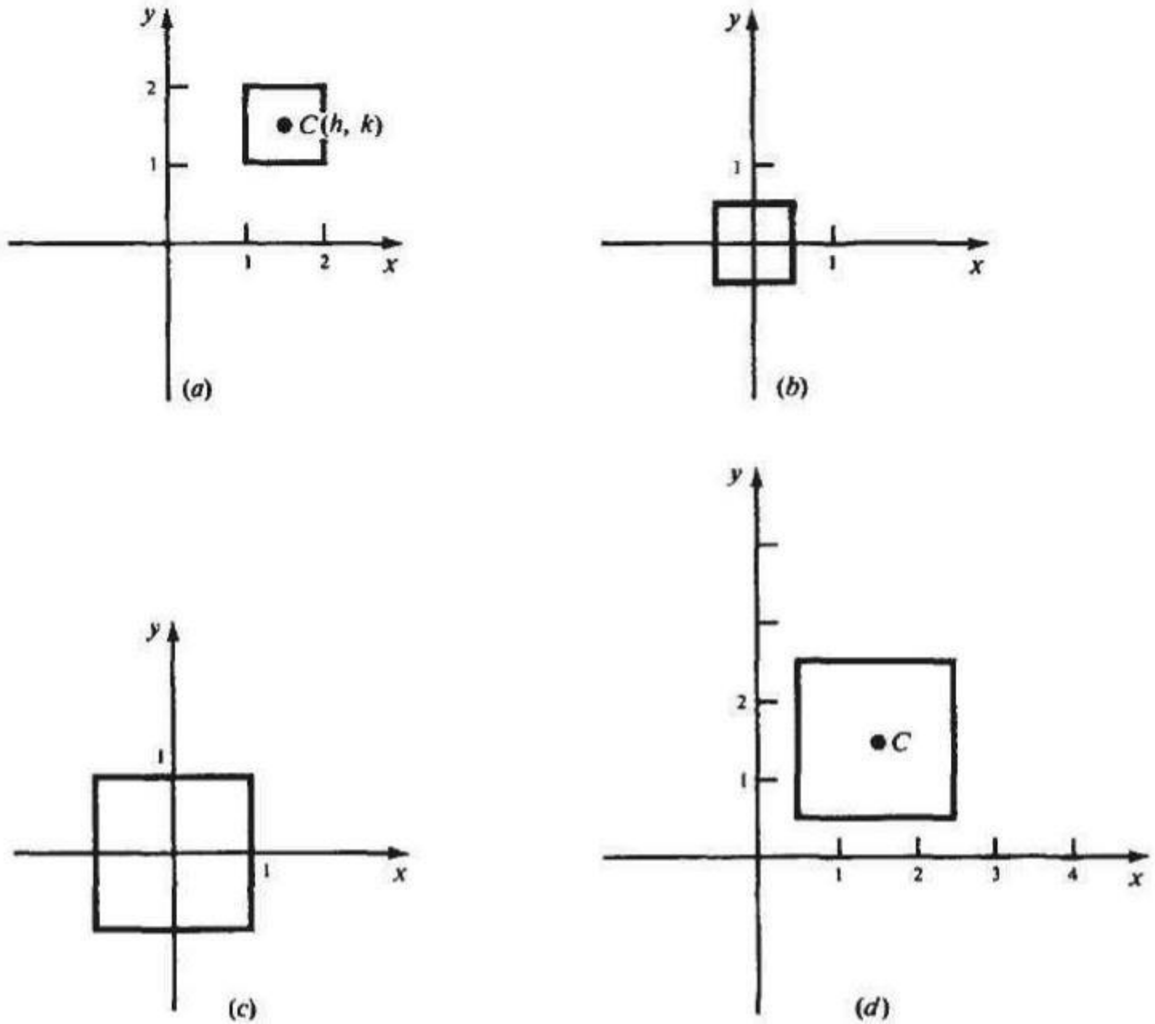
- Magnification of an object while keeping its center fixed:
- Let the geometric center be located at C(h, k). Choosing a magnification factor $s > 1$, we construct the transformation by performing the following sequence of basic transformations:

  (1) Translate the object so that its center coincides with the origin;

  (2) Scale the object with respect to the origin;

  (3) Translate the scaled object back to the original position.

Continue...



Continue...

- The required transformation $S_{S,C}$ can be formed by compositions:

  $S_{S,C} = T_v . S_{S,S} . T_v^{-1}$,where v = **h**I + **k**J.

- By using composition, we can build more general scaling, rotation, and reflection transformations.

- For these transformations, we shall use the following notations:

  (1) $S_{Sx,\, Sy,\, P}$ —scaling with respect to a fixed point P;

  (2) $R_{\theta,\, p}$—rotation about a point P;

  (3) M_L—reflection about a line L.

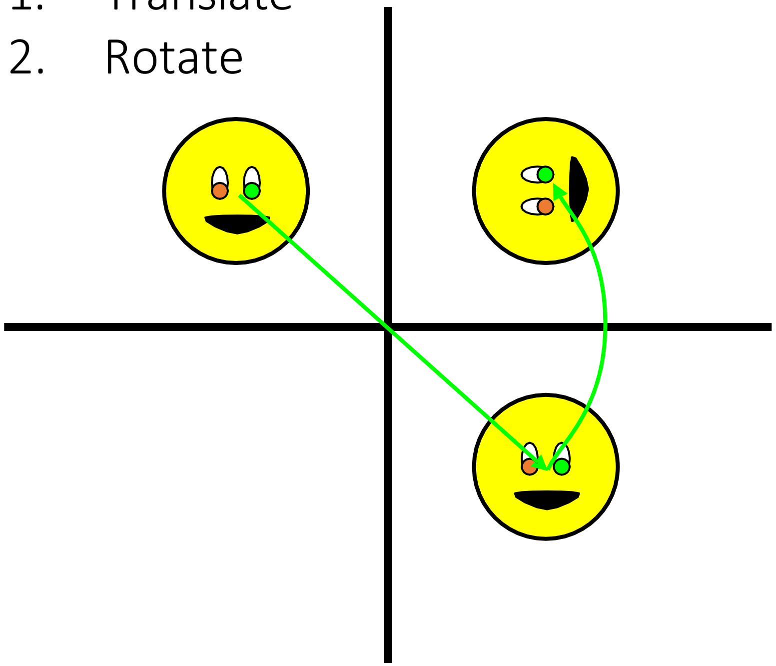
Transformations are NOT Commutative

- If we scale /rotate and then translate is that equivalent to translate first and then scale/rotate?
- No, because in general case result of matrix multiplication depends on the order.
- So, the order of transformation has to be maintained .

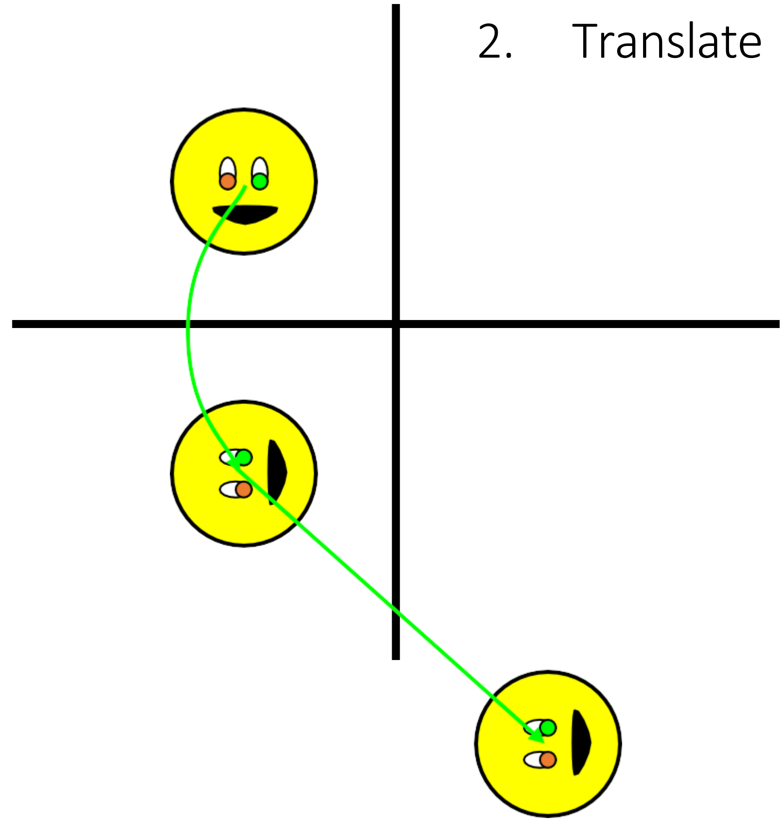
Order of operations

It does matter. Let's look at an example:

1. Translate
2. Rotate



1. Rotate
2. Translate



Matrix Description of the Basic Transformations

- The transformations of rotation, scaling, and reflection can be represented as matrix functions:

Geometric transformations	**Coordinate transformations**
$R_\theta = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix}$	$\bar{R}_\theta = \begin{pmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{pmatrix}$
$S_{s_x,s_y} = \begin{pmatrix} s_x & 0 \\ 0 & s_y \end{pmatrix}$	$\bar{S}_{s_x,s_y} = \begin{pmatrix} \frac{1}{s_x} & 0 \\ 0 & \frac{1}{s_y} \end{pmatrix}$
$M_x = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$	$\bar{M}_x = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$
$M_y = \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix}$	$\bar{M}_y = \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix}$

Matrix Review

Why do we use matrix?

- More convenient organization of data.
- More efficient processing
- Enable the combination of various concatenations

Matrix addition and subtraction

$$\begin{bmatrix} a \\ b \end{bmatrix} \pm \begin{bmatrix} c \\ d \end{bmatrix} = \begin{pmatrix} a \pm c \\ b \pm d \end{pmatrix}$$

Matrix Review

- Matrix Multiplication
  - Dot product

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \cdot \begin{pmatrix} e & f \\ g & h \end{pmatrix} = \begin{pmatrix} a.e + b.g & a.f + b.h \\ c.e + d.g & c.f + d.h \end{pmatrix}$$

Matrix Review

What about this?

$$\begin{pmatrix} 1 & 2 \end{pmatrix} \cdot \begin{pmatrix} 1 & 2 \\ 3 & 1 \end{pmatrix} = \begin{pmatrix} 7 & 4 \end{pmatrix}$$

$$\begin{pmatrix} 2 \\ 3 \end{pmatrix} \cdot \begin{pmatrix} 1 & 2 \\ 3 & 1 \end{pmatrix} = \text{No!!}$$

Type of matrix

$$\begin{pmatrix} \mathbf{a} & \mathbf{b} \end{pmatrix}$$

Row-Matrix

$$\begin{pmatrix} \mathbf{a} \\ \mathbf{b} \end{pmatrix}$$

Column-Matrix

Use of Matrix in Transformations

Vector or matrix representation of any point is a 2x1 matrix like below:

$$[P] = \begin{bmatrix} x \\ y \end{bmatrix}$$

General formula for transformation is like below:

$$[P'] = [T][P] \quad \ldots\ldots\ldots\ldots\ldots eq.1$$

here T describes the nature of transformation and known as geometric or affine transformation matrix. $[P']$ represents the transformed matrix where- $[P'] = \begin{bmatrix} x' \\ y' \end{bmatrix}$

Use of Matrix in Transformations

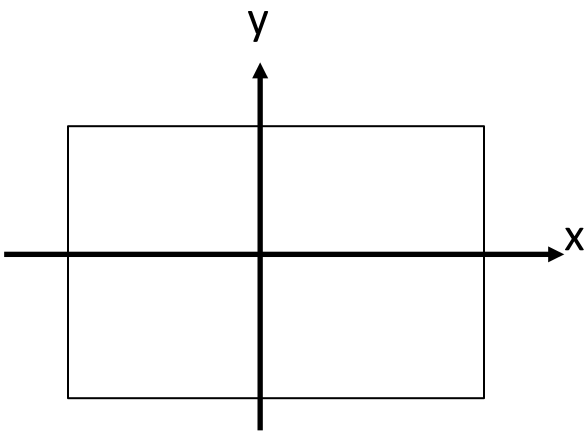
Lets know some more details before going to different types of transformations. From eq.1 we can write-

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & c \\ b & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

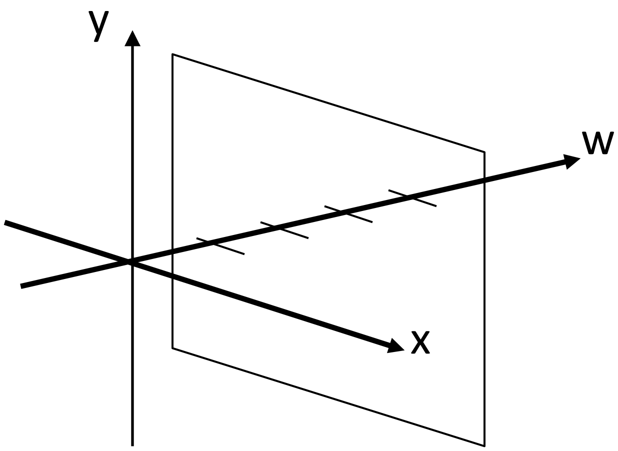
And we get two equations using matrix multiplication-

$$x' = ax + cy$$
$$y' = bx + dy$$

Homogenous Coordinates



➔



- Let's move our problem into 3D.
- Let point (x, y) in 2D be represented by point (x, y, 1) in the new space.
- Scaling our new point by any value a puts us somewhere along a particular line: (ax, ay, a).
- A point in 2D can be represented in many ways in the new space.
- (2, 4) ---------→ (8, 16, 4) or (6, 12, 3) or (2, 4, 1) or etc.

Continue…

- We can always map back to the original 2D point by dividing by the last coordinate
- (15, 6, 3) ---→ (5, 2).
- (60, 40, 10) -→ ?.

- Why do we use 1 for the last coordinate?

- The fact that all the points along each line can be mapped back to the same point in 2D gives this coordinate system its name – homogeneous coordinates.

Matrix Representation

- Point (x, y) in column matrix:

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

- Our point now has three coordinates. So our matrix is needs to be 3x3.
- Translation:

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

Continue…

- Rotation:

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

- Scaling:

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

Rotation about an Arbitrary Point P

- To rotate an object about a point $P(x, y)$ we need to follow the following steps:
  - Step 1: Translate by $(-x, -y)$
  - Step 2: Rotate
  - Step 3: Translate by (x, y)

Continue...

- From Step 1 we get-

$$T_3(-x,-y) = \begin{bmatrix} 1 & 0 & -x \\ 0 & 1 & -y \\ 0 & 0 & 1 \end{bmatrix}$$

- From Step 2 we get-

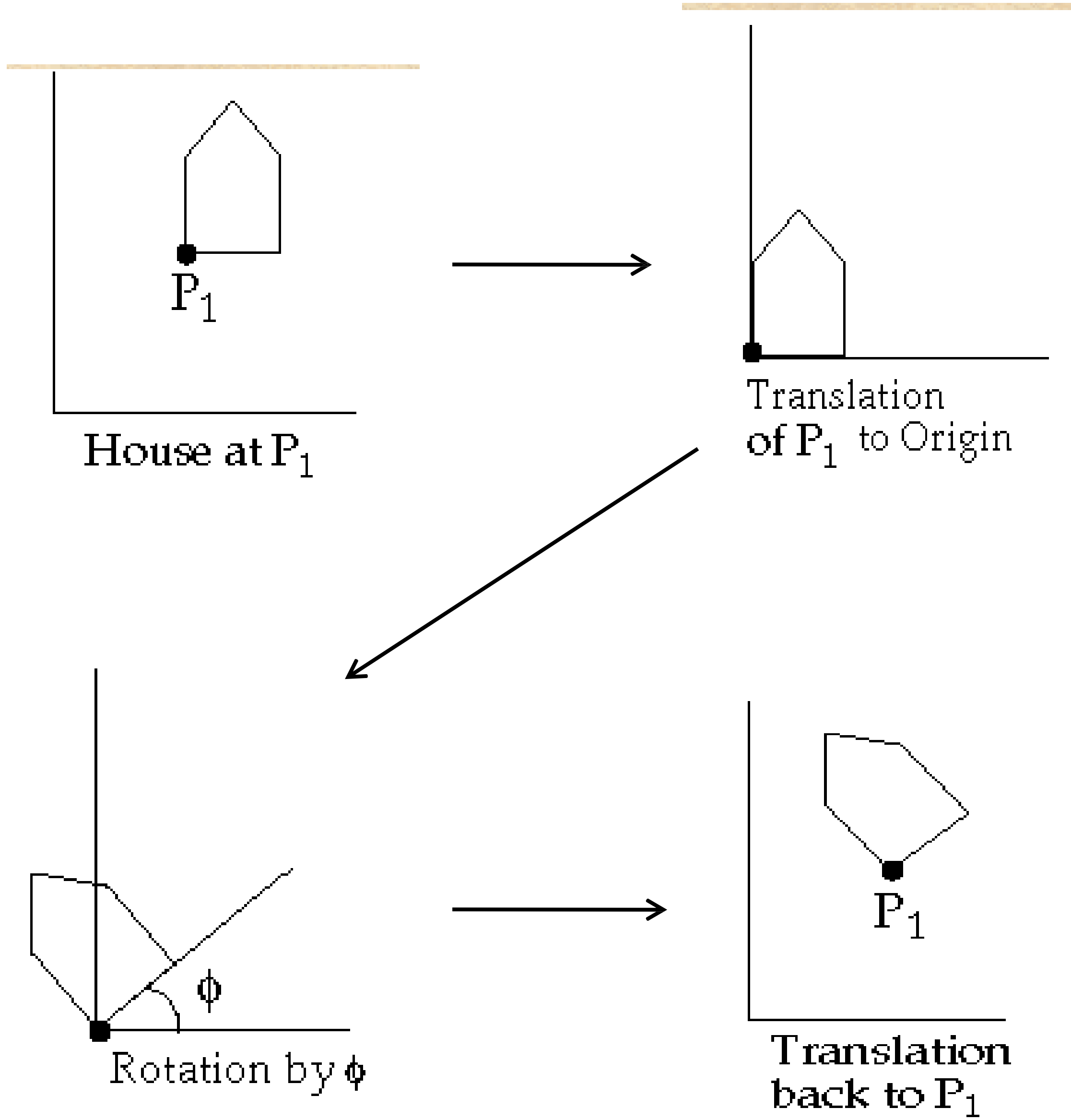
$$R(\Theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- From Step 3 we get-

$$T_1(x, y) = \begin{bmatrix} 1 & 0 & x \\ 0 & 1 & y \\ 0 & 0 & 1 \end{bmatrix}$$

- So, $T = T_1\ (x,y) * R(\theta) * T_3\ (-x,-y)$

Continue…



P_1
House at P_1
Translation of P_1 to Origin
ϕ
Rotation by ϕ
P_1
Translation back to P_1

Example - 02

- Perform a 45° rotation of triangle A (0, 0), B (1, 1), C (5, 2)
  - **(a) about the origin, and (b) about P(-1, -1).**

SOLUTION

We represent the triangle by a matrix formed from the homogeneous coordinates of the vertices:

$$\begin{array}{c} \begin{matrix} A & B & C \end{matrix} \\ \begin{pmatrix} 0 & 1 & 5 \\ 0 & 1 & 2 \\ 1 & 1 & 1 \end{pmatrix} \end{array}$$

(a) The matrix of rotation is

$$R_{45^\circ} = \begin{pmatrix} \cos 45^\circ & -\sin 45^\circ & 0 \\ \sin 45^\circ & \cos 45^\circ & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} & 0 \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

So the coordinates $A'B'C'$ of the rotated triangle ABC can be found as

$$[A'B'C'] = R_{45^\circ} \cdot [ABC] = \begin{pmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} & 0 \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 & 5 \\ 0 & 1 & 2 \\ 1 & 1 & 1 \end{pmatrix} = \begin{array}{c} \begin{matrix} A' & B' & C' \end{matrix} \\ \begin{pmatrix} 0 & 0 & \frac{3\sqrt{2}}{2} \\ 0 & \sqrt{2} & \frac{7\sqrt{2}}{2} \\ 1 & 1 & 1 \end{pmatrix} \end{array}$$

Thus $A' = (0, 0)$, $B' = (0, \sqrt{2})$, and $C' = (\frac{3}{2}\sqrt{2}, \frac{7}{2}\sqrt{2})$.

(b) about P(-1, -1).

From Prob. 4.4, the rotation matrix is given by $R_{45^\circ,P} = T_{\mathbf{v}} \cdot R_{45^\circ} \cdot T_{-\mathbf{v}}$, where $\mathbf{v} = -\mathbf{I} - \mathbf{J}$. So

$$R_{45^\circ,P} = \begin{pmatrix} 1 & 0 & -1 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} & 0 \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} & -1 \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & (\sqrt{2}-1) \\ 0 & 0 & 1 \end{pmatrix}$$

Now

$$[A'B'C'] = R_{45^\circ,P} \cdot [ABC] = \begin{pmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} & -1 \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & (\sqrt{2}-1) \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 & 5 \\ 0 & 1 & 2 \\ 1 & 1 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} -1 & -1 & (\frac{3}{2}\sqrt{2}-1) \\ (\sqrt{2}-1) & (2\sqrt{2}-1) & (\frac{9}{2}\sqrt{2}-1) \\ 1 & 1 & 1 \end{pmatrix}$$

So $A' = (-1, \sqrt{2}-1)$, $B' = (-1, 2\sqrt{2}-1)$, and $C' = (\frac{3}{2}\sqrt{2}-1, \frac{9}{2}\sqrt{2}-1)$.

Practice

1.Perform 60^0 rotation of a point P(2, 5) about a pivot point (1,2). Find P'?

Step 1: Translate the system so that the pivot point becomes the origin.

- The pivot point is $(1, 2)$, so we translate $P(2, 5)$ by subtracting the coordinates of the pivot point:

$$P_{translated} = (2 - 1, 5 - 2) = (1, 3)$$

Step 2: Perform the 60° rotation around the origin.

The rotation matrix for an angle $\theta = 60^\circ$ is:

$$\begin{pmatrix} \cos(60^\circ) & -\sin(60^\circ) \\ \sin(60^\circ) & \cos(60^\circ) \end{pmatrix} = \begin{pmatrix} \frac{1}{2} & -\frac{\sqrt{3}}{2} \\ \frac{\sqrt{3}}{2} & \frac{1}{2} \end{pmatrix}$$

To rotate the translated point $(1, 3)$, multiply the rotation matrix by the point's coordinates:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \frac{1}{2} & -\frac{\sqrt{3}}{2} \\ \frac{\sqrt{3}}{2} & \frac{1}{2} \end{pmatrix} \begin{pmatrix} 1 \\ 3 \end{pmatrix}$$

Carrying out the matrix multiplication:

$$x' = \left(\frac{1}{2} \cdot 1\right) + \left(-\frac{\sqrt{3}}{2} \cdot 3\right) = \frac{1}{2} - \frac{3\sqrt{3}}{2}$$

$$y' = \left(\frac{\sqrt{3}}{2} \cdot 1\right) + \left(\frac{1}{2} \cdot 3\right) = \frac{\sqrt{3}}{2} + \frac{3}{2}$$

So, the new coordinates after rotation are:

$$(x', y') = \left(\frac{1 - 3\sqrt{3}}{2}, \frac{\sqrt{3} + 3}{2}\right)$$

Step 3: Translate the system back to the original position.

Now, we translate the rotated point back by adding the pivot point $(1, 2)$ to the result:

$$P' = \left(\frac{1 - 3\sqrt{3}}{2} + 1, \frac{\sqrt{3} + 3}{2} + 2\right)$$

The coordinates of the point P' after rotating P(2, 5) by 60° about the pivot point

(1, 2) are approximately (-1.10, 4.37).

P' = (-1, 4)

Composite Transformation Matrix

General Fixed-Point Scaling

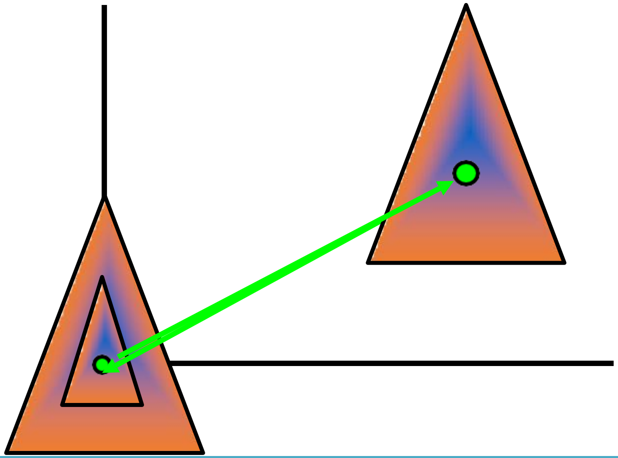
Operation :-

1. Translate (fixed point is moved to origin)
2. Scale with respect to origin
3. Translate (fixed point is returned to original position)

T(fixed) • S(scale) • T(–fixed)

Find the matrix that represents scaling of an object with respect to any fixed point?

Given P(6, 8) , Sx = 2, Sy = 3 and fixed point (2, 2). Use that matrix to find P'?



Answer

$$\begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & -t_x \\ 0 & 1 & -t_y \\ 0 & 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} Sx & 0 & -t_x Sx \\ 0 & Sy & -t_y Sy \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} Sx & 0 & -t_x Sx + t_x \\ 0 & Sy & -t_y Sy + t_y \\ 0 & 0 & 1 \end{pmatrix}$$

$x = 6$, $y = 8$, $Sx = 2$, $Sy = 3$, $t_x = 2$, $t_y = 2$

$$\begin{pmatrix} 2 & 0 & -2(2) + 2 \\ 0 & 3 & -2(3) + 2 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 6 \\ 8 \\ 1 \end{pmatrix} = \begin{pmatrix} 10 \\ 20 \\ 1 \end{pmatrix}$$

Practice Problem

Solved problems from Chapter-4:

- 4.2, 4.4 to 4.9.

Thank You!

ICT 4203
Computer Graphics and Animation

Lecture 10
2D Viewing & Clipping

Md. Mahmudur Rahman
Lecturer
Institute of Information Technology

Lecture Outlines

- Coordinate Systems
- 2D Viewing Transformation
- Window to Viewport Mapping

Coordinate Systems

- **Cartesian** – along the x and y axis from (0,0).
- **Polar** – rotation around the angle θ e.g. (r, θ).
- Graphic libraries mostly uses Cartesian coordinates.
- Any polar coordinates must be converted to Cartesian coordinates.
- **Four Cartesian coordinates systems in computer Graphics:**
    1. Modeling coordinates;
    2. World coordinates;
    3. Normalized device coordinates;
    4. Device coordinates.

Modeling Coordinates

- A coordinate system used for three-dimensional modeling in which each object possesses its own set of coordinates which can then be converted to a set of world coordinates.
- Also known as local coordinate.
- Each object has an origin (0,0).
- So the part of the objects are placed with reference to the object’s origin.
- In terms of scale it is user defined; so coordinate values can be any size.

World Coordinates

- The world coordinate system describes the relative positions and orientations of every generated objects.
- The scene has an origin (0,0).
- The object in the scene are placed with reference to the scenes origin.
- World coordinate scale may be the same as the modeling coordinate scale or it may be different.
- However, the coordinates values can be any size (similar to MC).

Normalized Device Coordinates

- Output devices have their own coordinates.

Coordinate values:

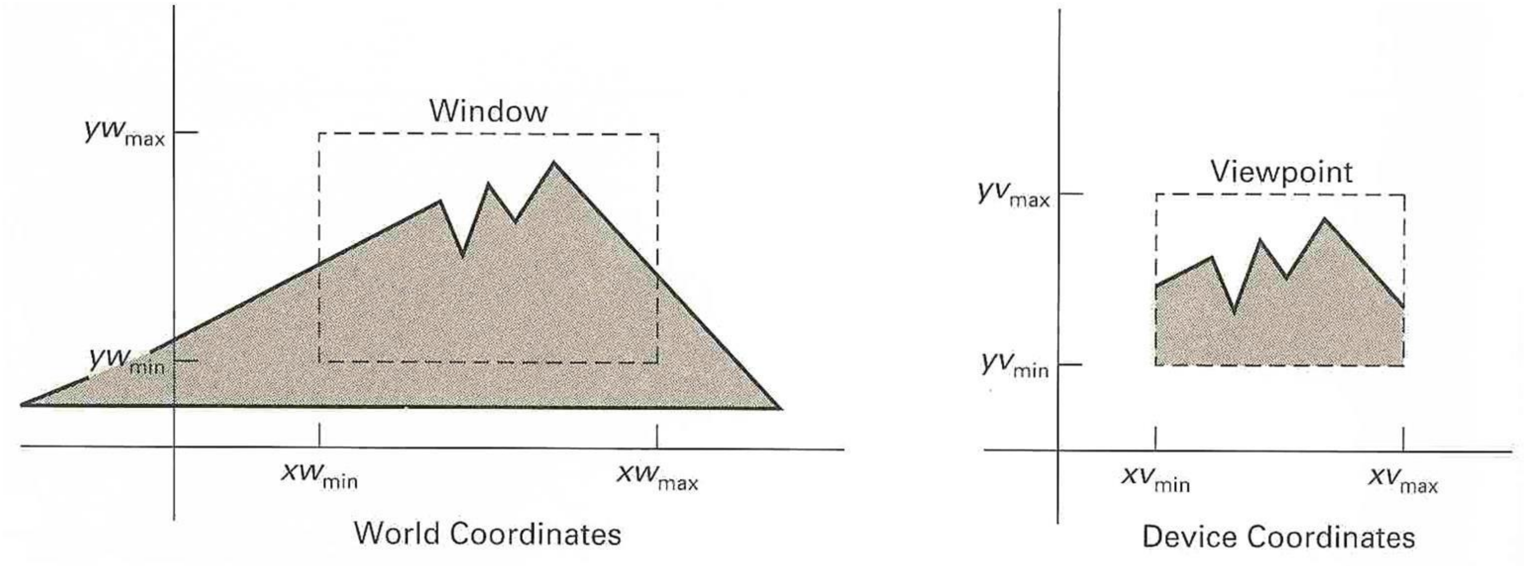
- The x and y axis range from 0 to 1.
- All the x and y coordinates are floating point numbers in the range of 0 to 1.
- This makes the system independent of the various devices coordinates.
- This is handled internally by graphic system without user awareness.

Device Coordinates

- Specific coordinates used by a device.
  - Pixels on a monitor
  - Points on a laser printer.
  - mm on a plotter.
- The transformation based on the individual device is handled by computer system without user concern.

2D Viewing Transformation

- The mapping of a part of a world-coordinate scene to device coordinates.
- 2D viewing transformation = window-to-viewport, windowing transformation.



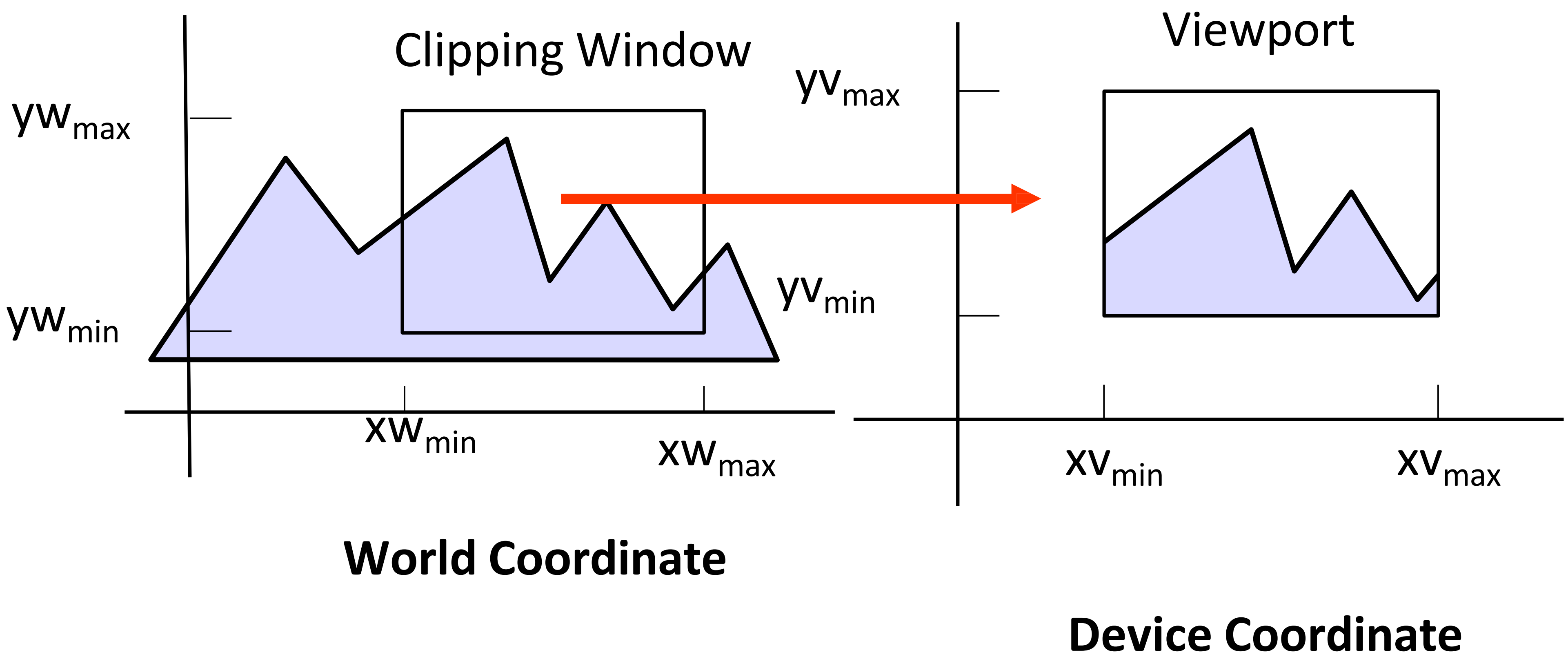
Continue…

- **window**
  - a world-coordinate area selected for display.
  - define what is to be viewed.
- **view port**
  - an area on a display device to which a window is mapped.
  - define where it is to be displayed.
  - define within the unit square.
  - the unit square is mapped to the display area for the particular output device in use at that time.
- **windows & viewport**
  - be rectangles in standard position, with the rectangle edges parallel to the coordinate axes.

2D Viewing Transformation Steps

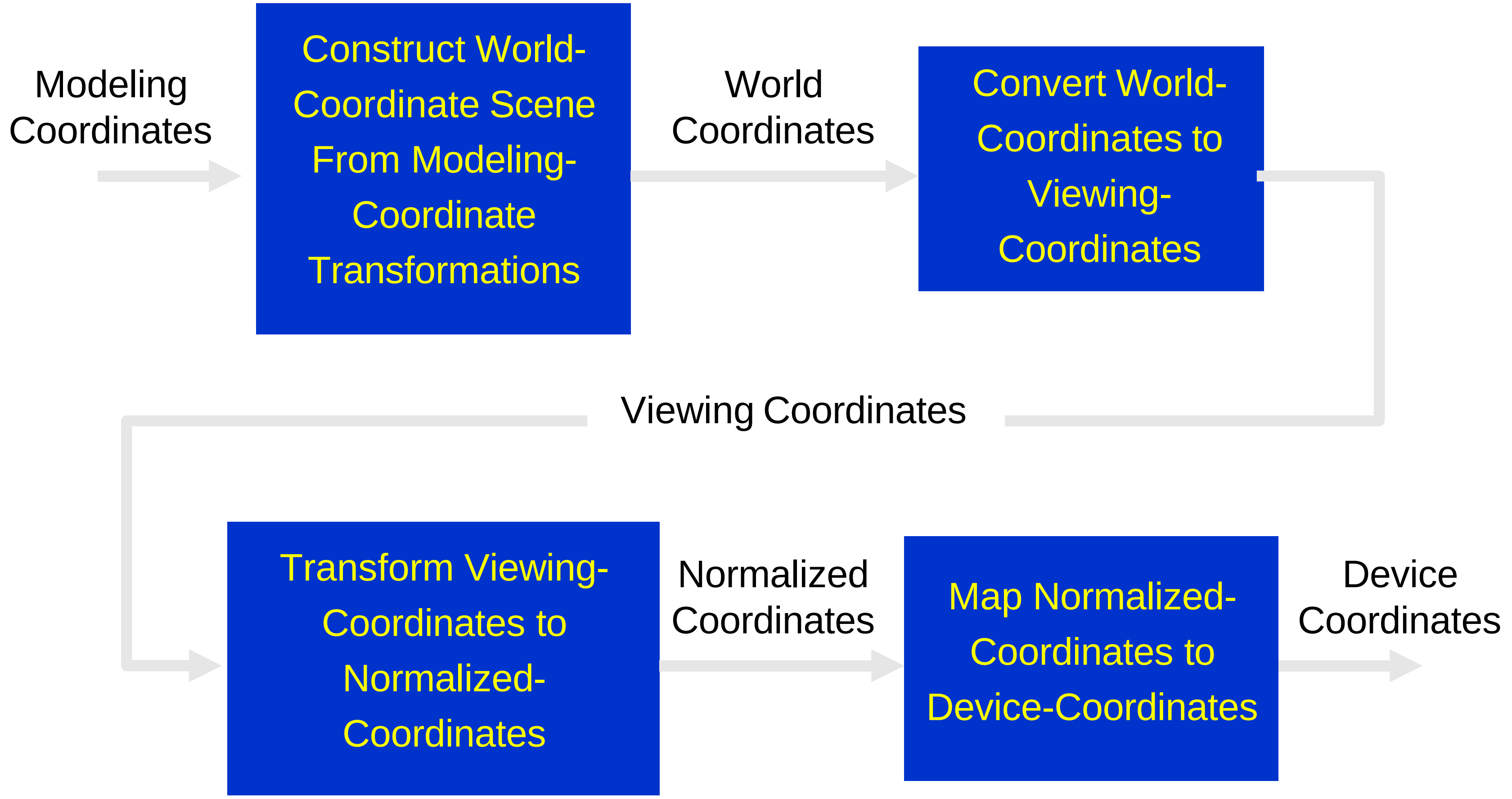
1. Construct the world-coordinate scene.
2. Transform descriptions in world coordinates to viewing coordinates.
3. Map the viewing-coordinate description of the scene to normalized coordinates.
4. Transfer to device coordinates.

Example


Clipping Window
ywmax
ywmin
xwmin
xwmax
World Coordinate
Viewport
yvmax
yvmin
xvmin
xvmax
Device Coordinate

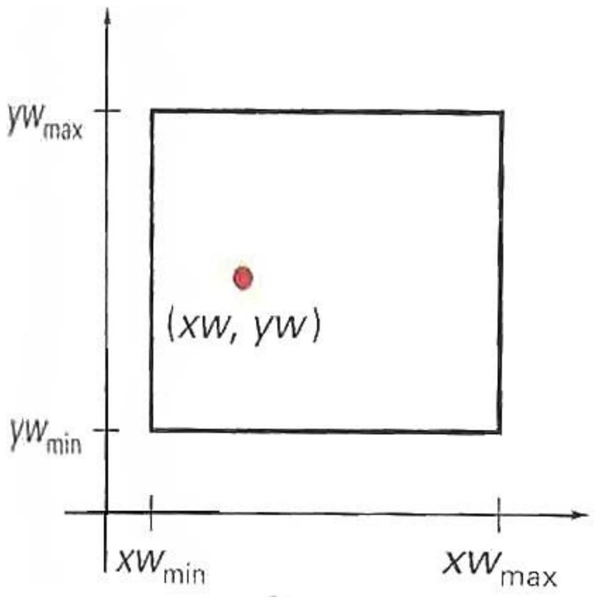
Viewing Transformation Pipeline

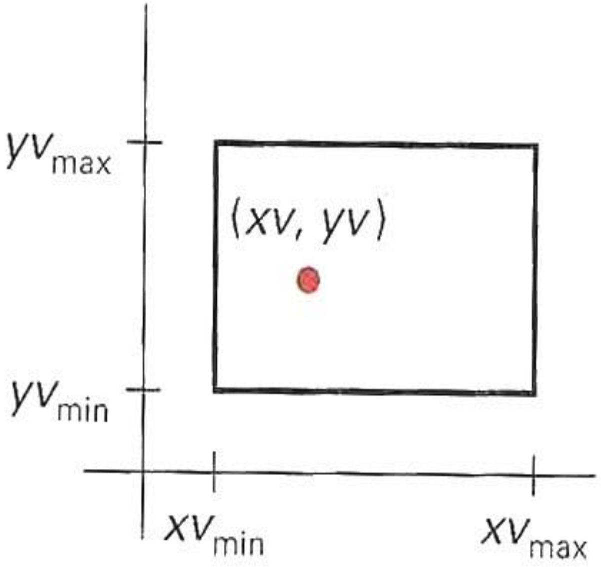
- The two-dimensional viewing-transformation pipeline



Window to Viewport Mapping

A point at position (xw, yw) in a designated window is mapped to viewport coordinates (xv, yv) so that relative positions in the two areas are the same.





Continue...

In order to maintain the same relative placement of the point in the viewport as in the window, we require:

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

and

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

Thus-

$$xv = xv_{\min} + (xw - xw_{\min})sx$$
$$yv = yv_{\min} + (yw - yw_{\min})sy$$

where

$$sx = \frac{xv_{\max} - xv_{\min}}{xw_{\max} - xw_{\min}}$$
$$sy = \frac{yv_{\max} - yv_{\min}}{yw_{\max} - yw_{\min}}$$

Continue...

Now we can express those two formulas for computing (vx, vy) from (wx, wy) in terms of a **translate-scale-translate** transformation N.

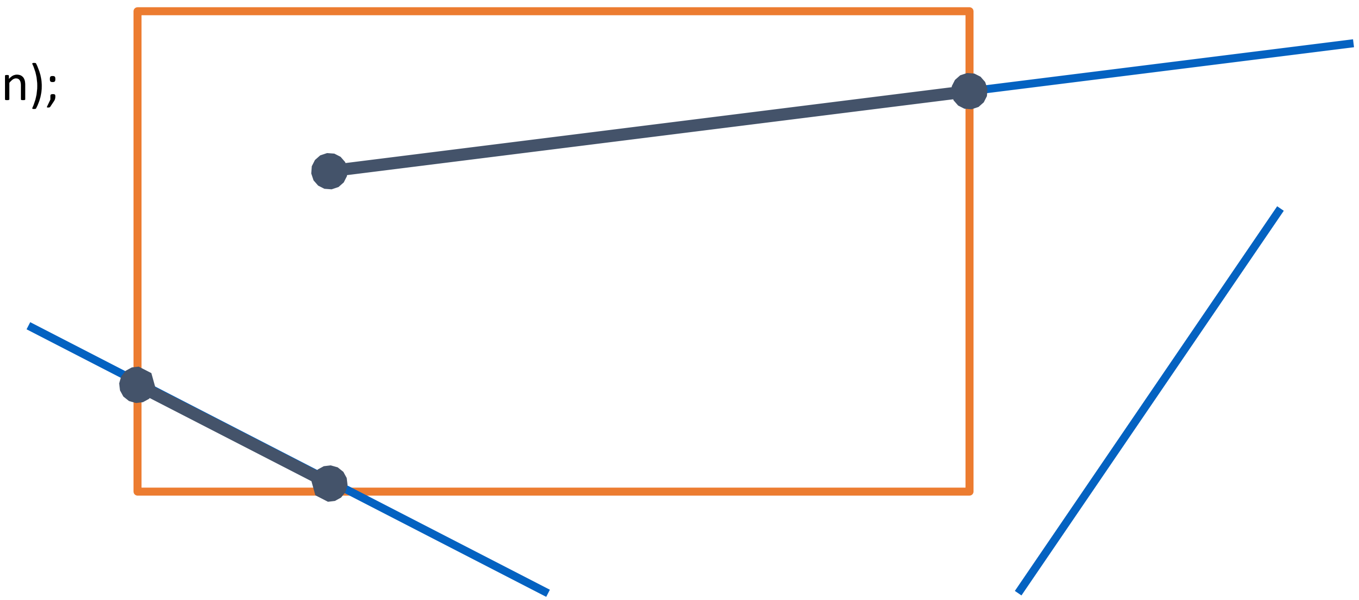
$$\begin{pmatrix} vx \\ vy \\ 1 \end{pmatrix} = N.\begin{pmatrix} wx \\ wy \\ 1 \end{pmatrix}$$

where

$$N = \begin{bmatrix} 1 & 0 & xv_{\min} \\ 0 & 1 & yv_{\min} \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} \dfrac{xv_{\max} - xv_{\min}}{xw_{\max} - xw_{\min}} & 0 & 0 \\ 0 & \dfrac{yv_{\max} - yv_{\min}}{yw_{\max} - yw_{\min}} & 0 \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} 1 & 0 & -xw_{\min} \\ 0 & 1 & -yw_{\min} \\ 0 & 0 & 1 \end{bmatrix}$$

Clipping

- Analytically calculating the portions of primitives within the viewport.
- Adaptive primitive types:
  - Point;
  - Line;
  - Area (e.g. Polygon);



Point Clipping

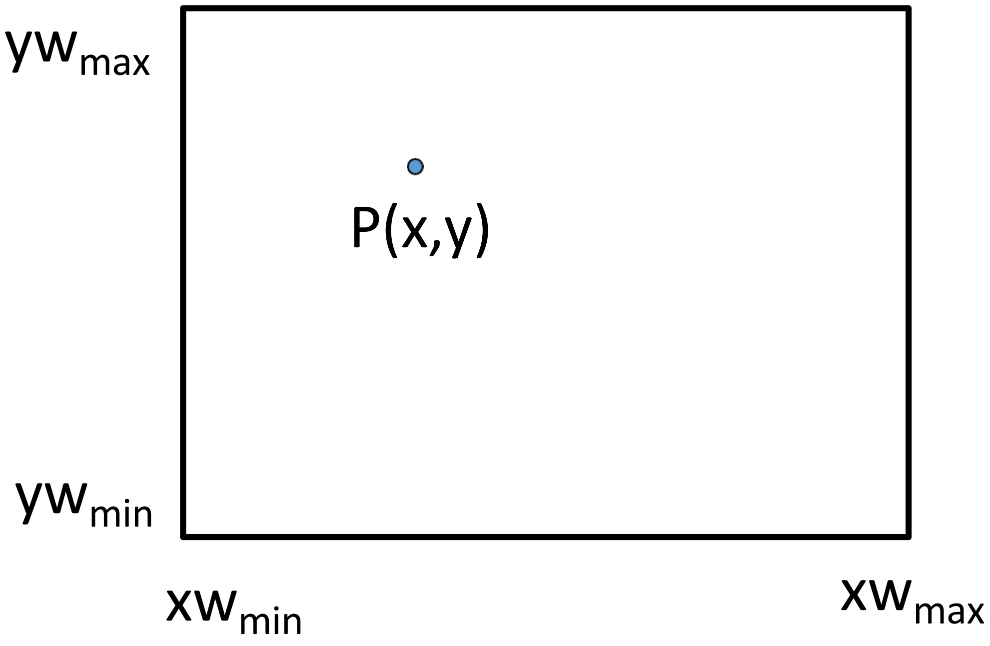
- Assuming that the clip window is a rectangle in standard position.
- For a clipping rectangle in standard position, we save a 2-D point P(x, y) for display if the following inequalities are satisfied:

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

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

- If any one of these four inequalities is not satisfied, the point is clipped (not saved for display).
- Where $x_{\min}, x_{\max}, y_{\min}, y_{\max}$ define the clipping window.

Point Clipping



If P(x,y) is inside the window?

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

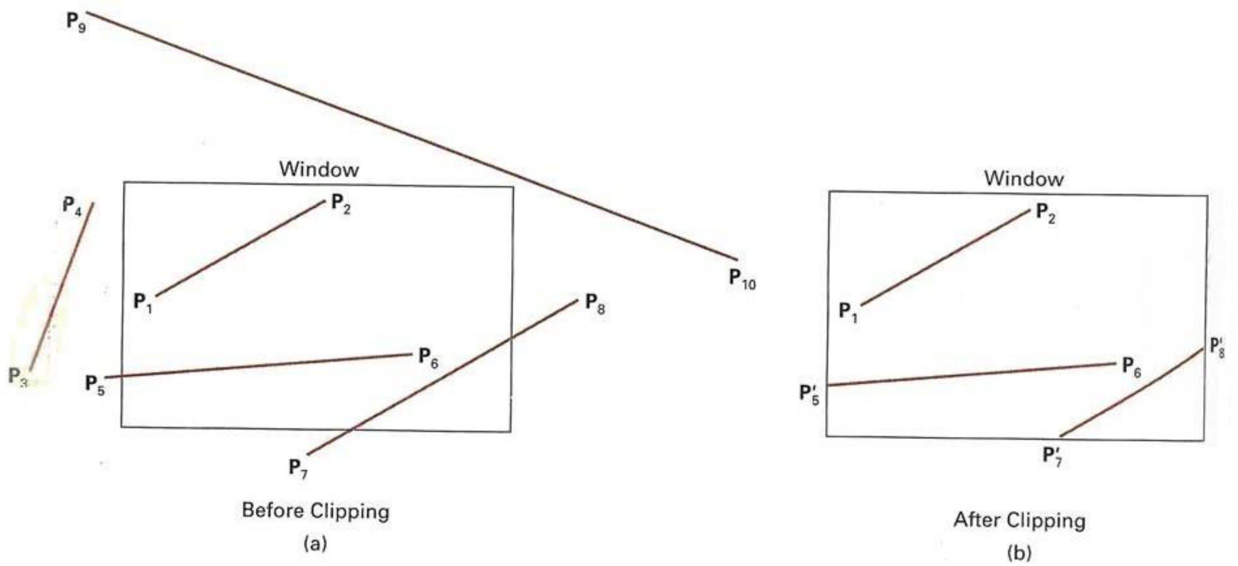
$$yw_{\min} \leq y \leq yw_{\max}$$

Line clipping

- **Line clipping procedure**
  - test a given line segment to determine whether it lies completely inside the clipping window.
  - if it doesn’t, we try to determine whether it lies completely outside the window.
  - if we can’t identify a line as completely inside or completely outside, we must perform intersection calculations with one or more clipping boundaries.

Line clipping

- Checking the line endpoints ⇒ inside-outside test.



- Line clipping Algorithm:
  - Cohen-Sutherland line clipping Algorithm;
  - Liang-Barsky line clipping Algorithm.

Thank you

ICT 4203
Computer Graphics and Animation

Lecture 11
Line clipping Algorithm

Md. Mahmudur Rahman
Lecturer
Institute of Information Technology

Lecture Outlines

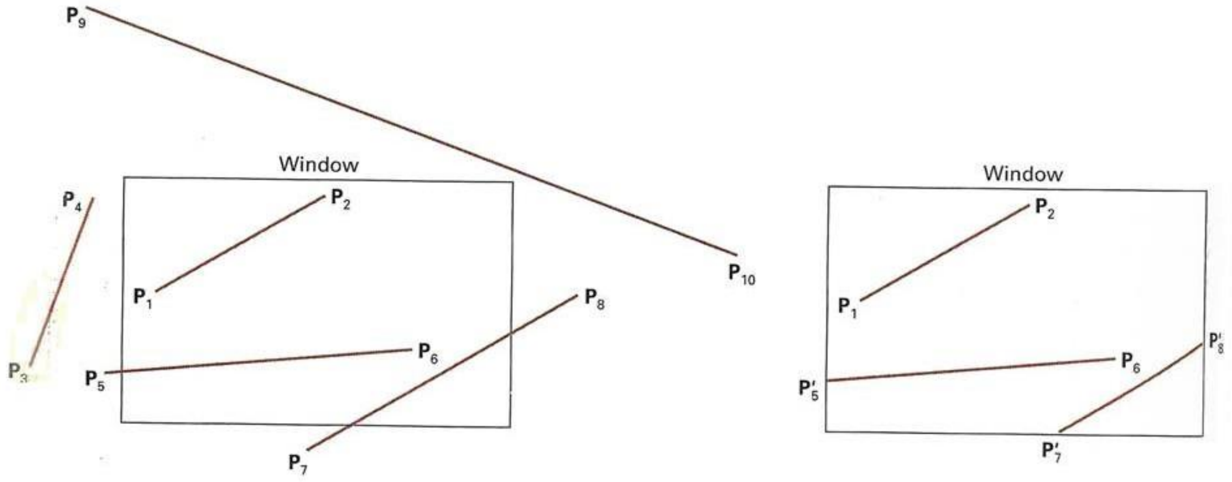
- Line Clipping Algorithms -
  - ✓ Cohen-Sutherland Algorithm
  - ✓ Midpoint Subdivision Algorithm

Line Clipping

- **Line clipping procedure -**
  - Test a given line segment to determine whether it lies completely inside the clipping window.
  - If it doesn’t, we try to determine whether it lies completely outside the window.
  - If we can’t identify a line as completely inside or completely outside, we must perform intersection calculations with one or more clipping boundaries.

Continue...

- Checking the line endpoints ⇒ inside-outside test.



Before Clipping
(a)

After Clipping
(b)

- Line clipping Algorithm:
  - Cohen-Sutherland Algorithm;
  - Midpoint Subdivision Algorithm;
  - Liang-Barsky Algorithm.

Cohen-Sutherland Algorithm

- Divide the line clipping process into two phases:
  1) Identify those lines which intersect the clipping window and so need to be clipped;
  2) Perform the clipping.
- All lines fall into one of the following clipping categories:
  1) **Visible:** Both end points of the line lie within the window.
  2) **Not visible:** The line definitely lies outside the window. This will occur if the line from (x1,y1) to (x2,y2) satisfies any one of the following inequalities:

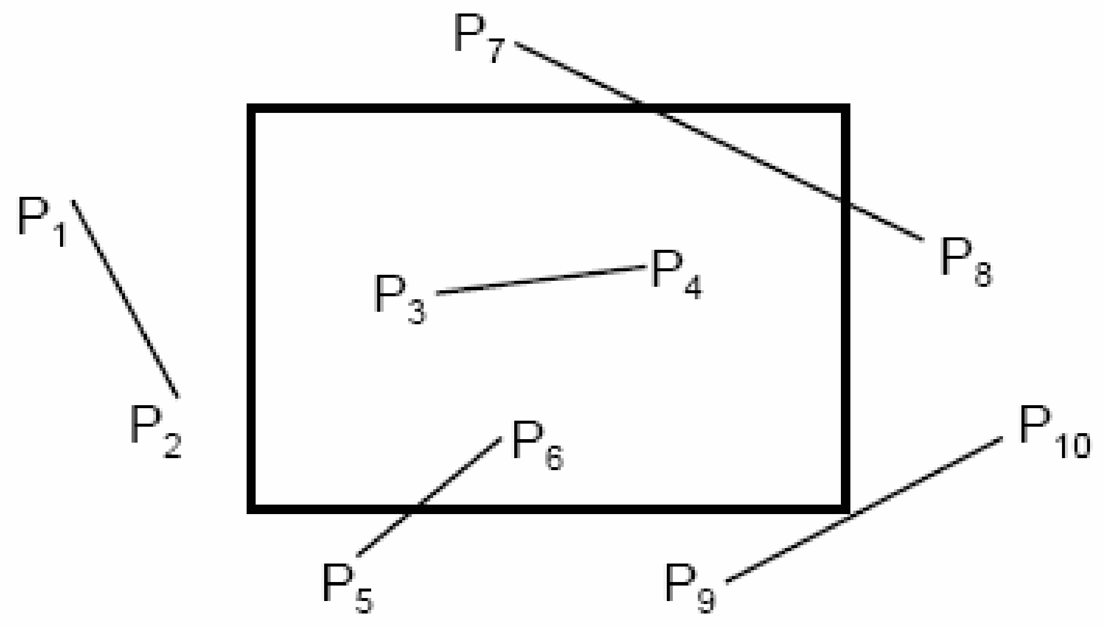
$$x_{1,}x_2 > x_{\max} \qquad y_1, y_2 > y_{\max}$$

$$x_{1,}x_2 < x_{\min} \qquad y_1, y_2 < y_{\min}$$

  3) **Clipping candidate:** the line is in neither category 1 nor 2.

Continue...

Find the part of a line inside the clip window

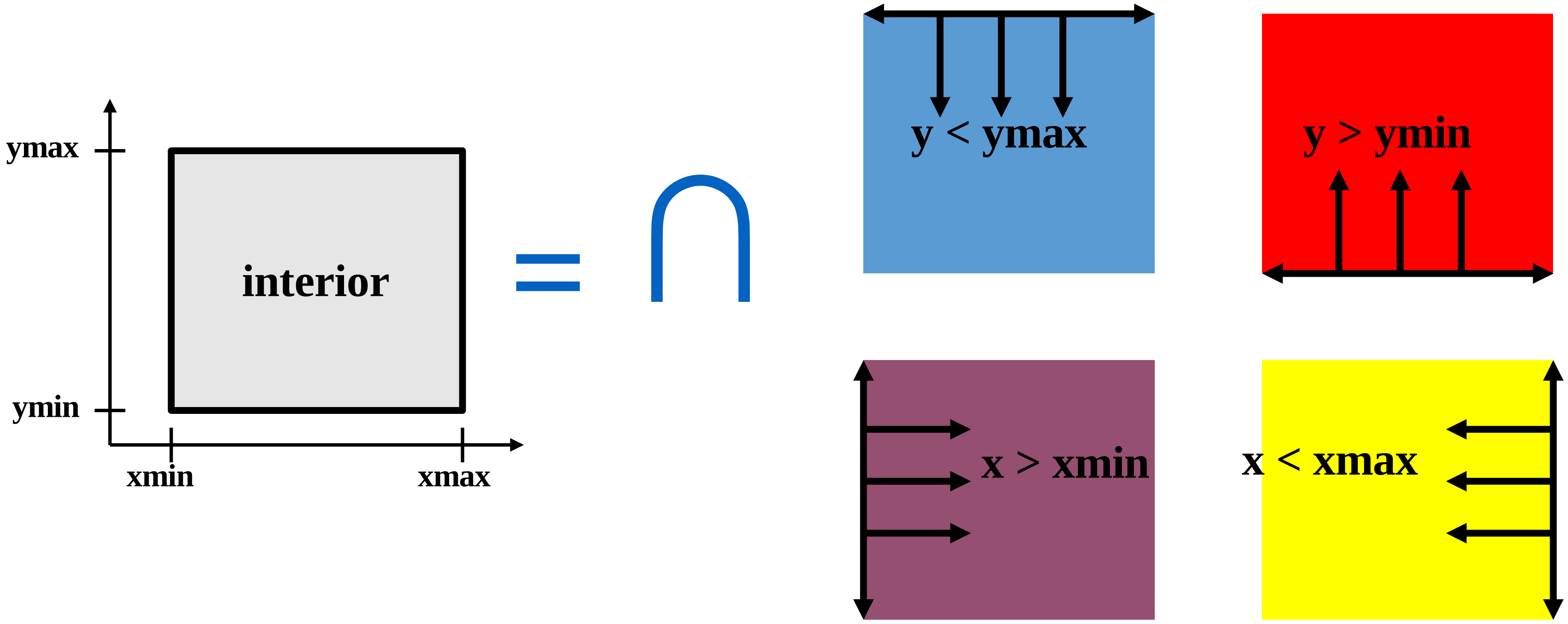


p_3p_4 is in category 1(Visible)

p_1p_2 is in category 2(Not Visible)

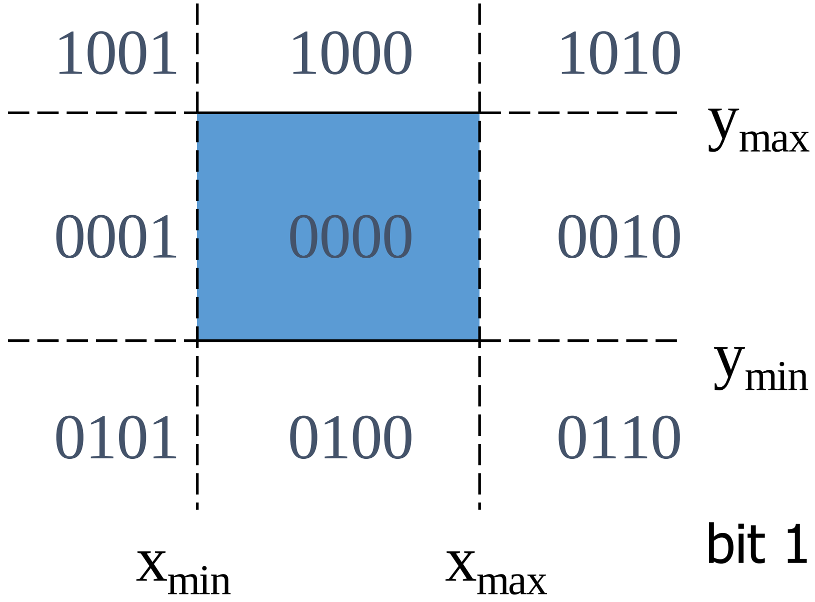
$p_5p_6, p_7p_8, p_9p_{10}$ is in category 3(Clipping candidate)

Continue...



Continue…

- Assign a four-bit pattern (Region Code) to each endpoint of the given segment. The code is determined according to which of the following nine regions of the plane the endpoint lies in.

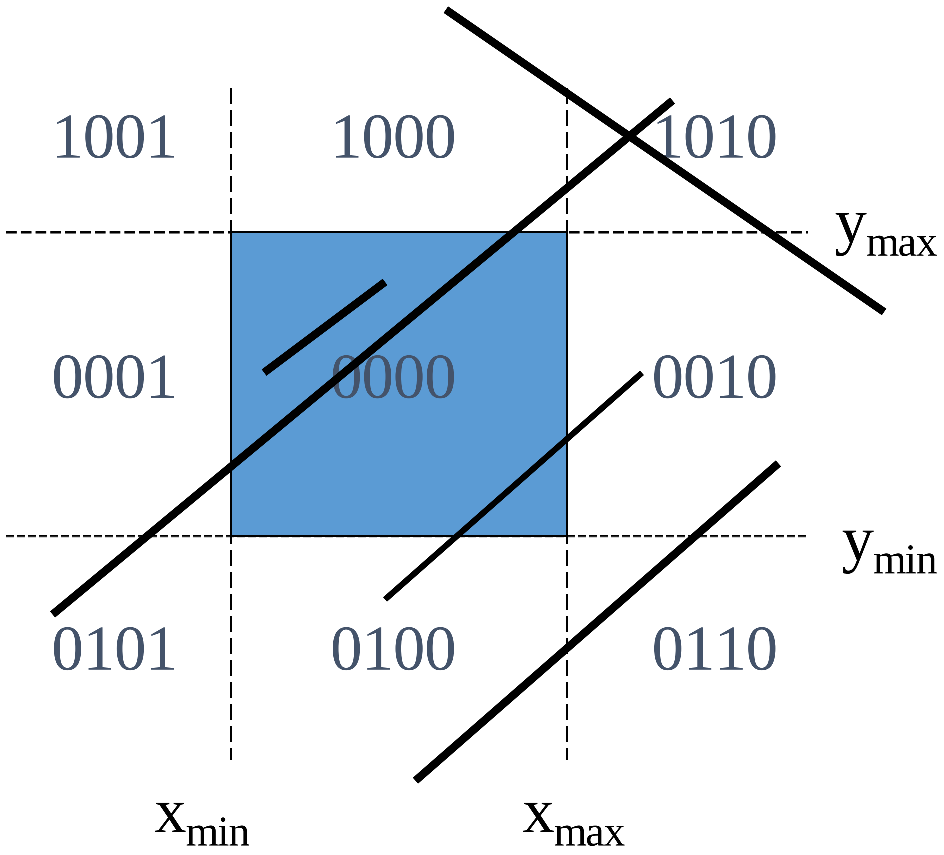


bit 1 : bit 2 : bit 3 : bit 4

Top : Bottom : Right : Left:

- Of course, a point with code 0000 is inside the window.

Continue...

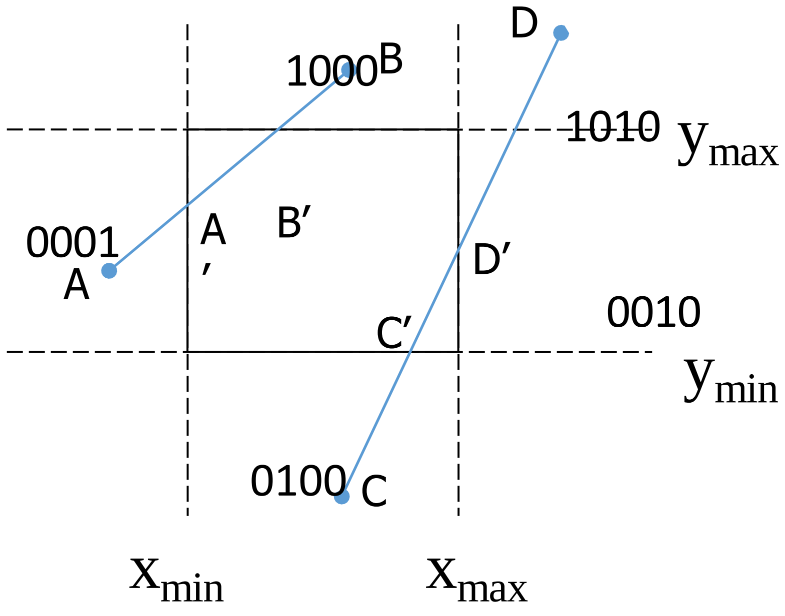


1001
1000
1010
y_{max}
0001
0000
0010
y_{min}
0101
0100
0110
x_{min}
x_{max}

Continue...

- If both endpoint codes are 0000, the line segment is visible (inside).
- The logical AND of the two endpoint codes -
  - not completely 0000 , the line segment is not visible (outside).
  - completely 0000, the line segment maybe inside (and outside).
- Lines that cannot be identified as being completely inside or completely outside, a clipping window are then checked for intersection with the window border lines.

Continue...



- Consider code of an end point
  - if bit 1 is 1, intersect with line y = Ymax
  - if bit 2 is 1, intersect with line y = Ymin
  - if bit 3 is 1, intersect with line x = Xmax
  - if bit 4 is 1, intersect with line x = Xmin
- Consider line CD.
  - If endpoint C is chosen, then the bottom boundary line Y=Ymin is selected for computing intersection
  - If endpoint D is chosen, then either the top boundary line Y=Ymax or the right boundary line X=Xmax is used.
  - The coordinates of the intersection point are:
    - $\begin{cases} x_i = x_{min} \text{ or } x_{max} \\ y_i = y_1 + m(x_i - x_1) \end{cases}$ if the boundary line is vertical
    - $\begin{cases} x_i = x_1 + (y_i - y_1)/m \\ y_i = y_{min} \text{ or } y_{max} \end{cases}$ if the boundary line is horizontal
    - where $m = \dfrac{y_{end} - y_0}{x_{end} - x_0}$

Cohen Sudenland line Clipping:

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

horizontal intersection:

$$y_i = y_{max} \text{ or } y_{min}$$

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

$$\Rightarrow x_2 = x_1 + \frac{(y_i - y_1)}{m}$$

$$\text{or, } x_i = x_1 + (y_i - y_1)/m$$

Vertical Intersection

$$x_i = x_{max} \text{ or } x_{min}$$

$$m = \frac{y_2 - y_1}{x_{max} - x_1}$$

$$\Rightarrow y_2 = y_1 + m(x_{max} - x_1)$$

$$\Rightarrow y_i = y_1 + m(x_{max} - x_1)$$

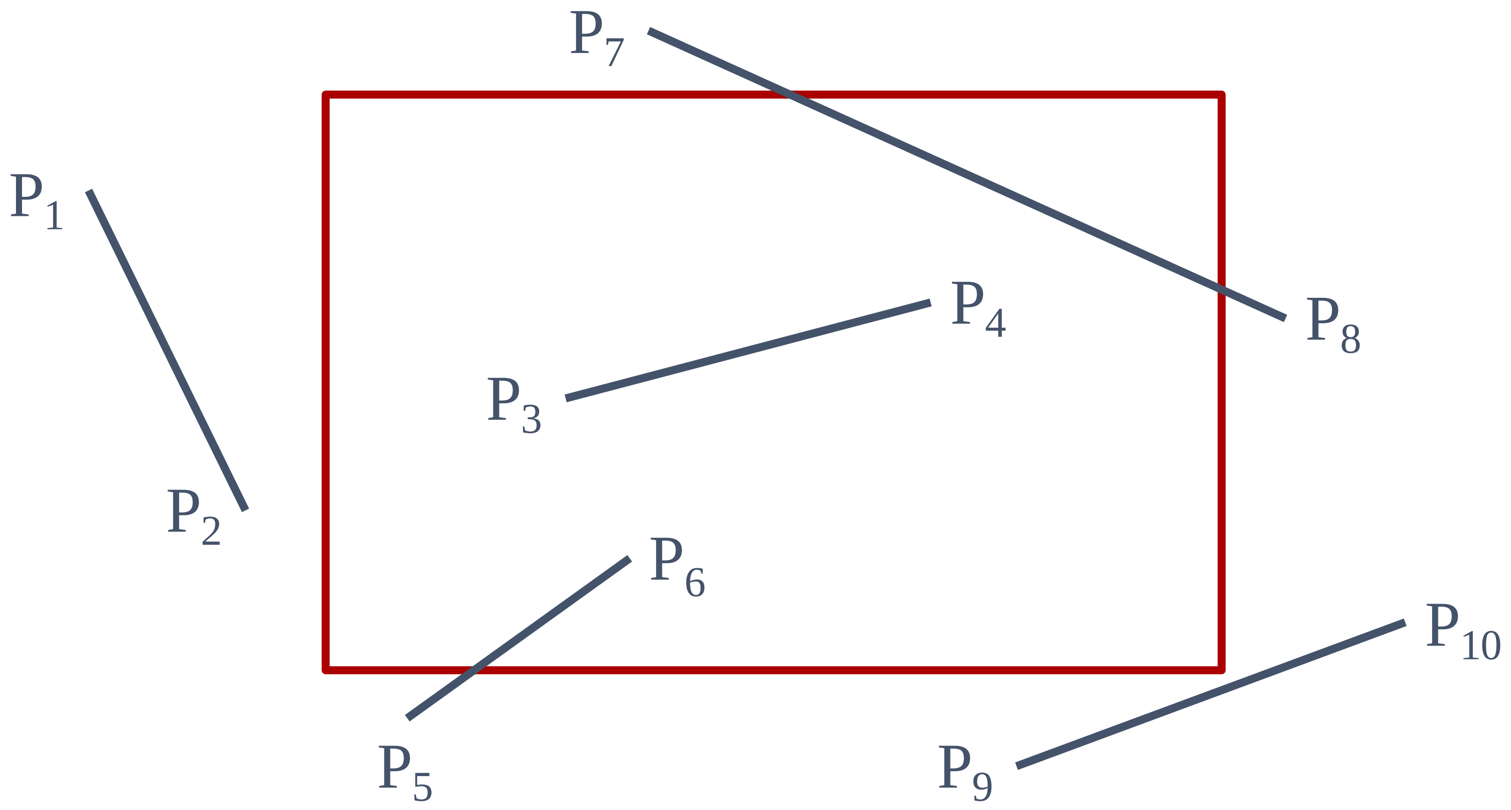
i

Continue…

- Replace endpoint (x1,y1) with the intersection point(xi,yi), effectively eliminating the portion of the original line that is on the outside of the selected window boundary.
- The new endpoint is then assigned an updated region code and the clipped line re-categoriged and handled in the same way.
- This iterative process terminates when we finally reach a clipped line that belongs to either category 1(visible) or category 2(not visible).

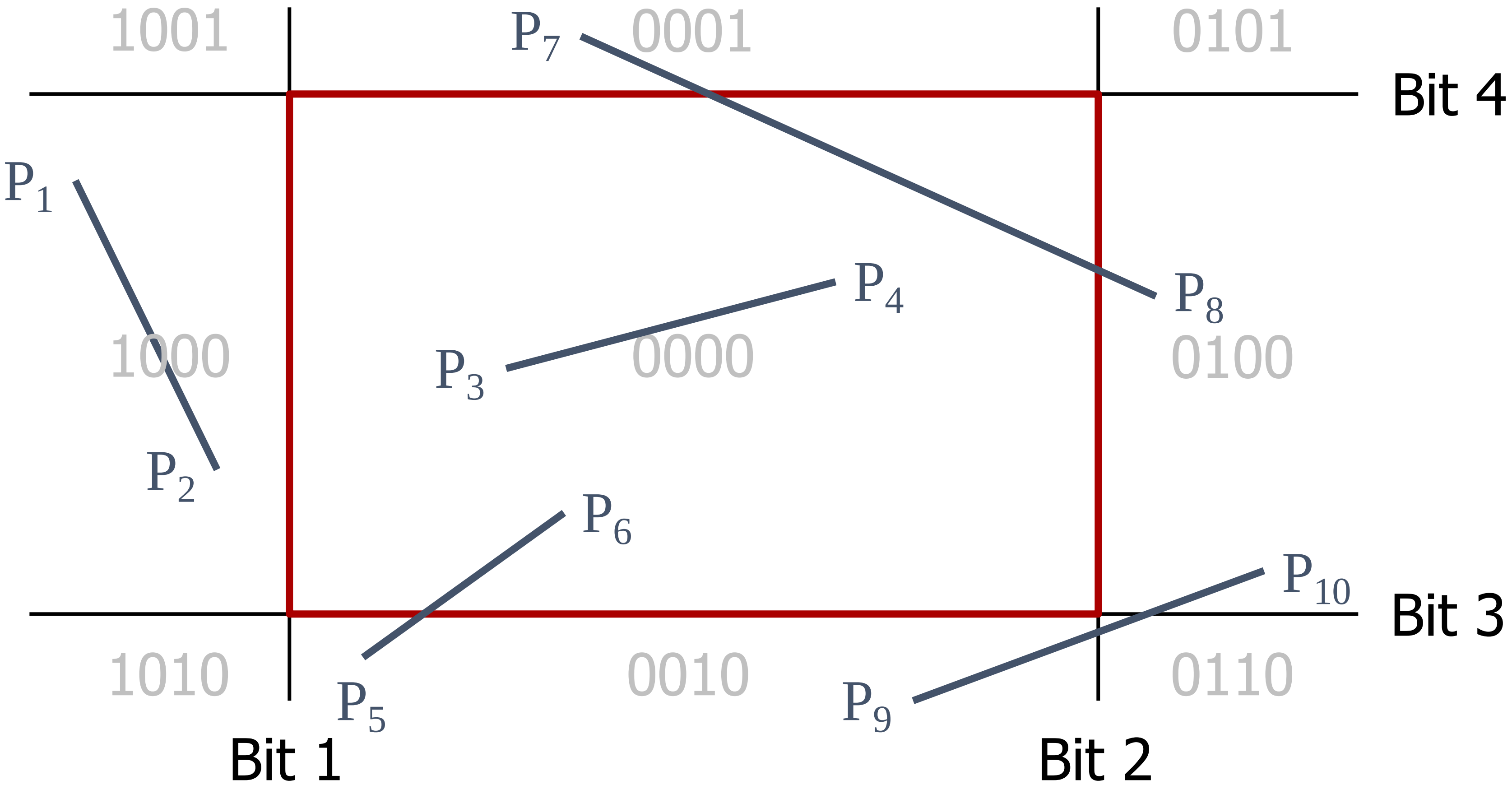
Continue…

- Use simple tests to classify easy cases first:

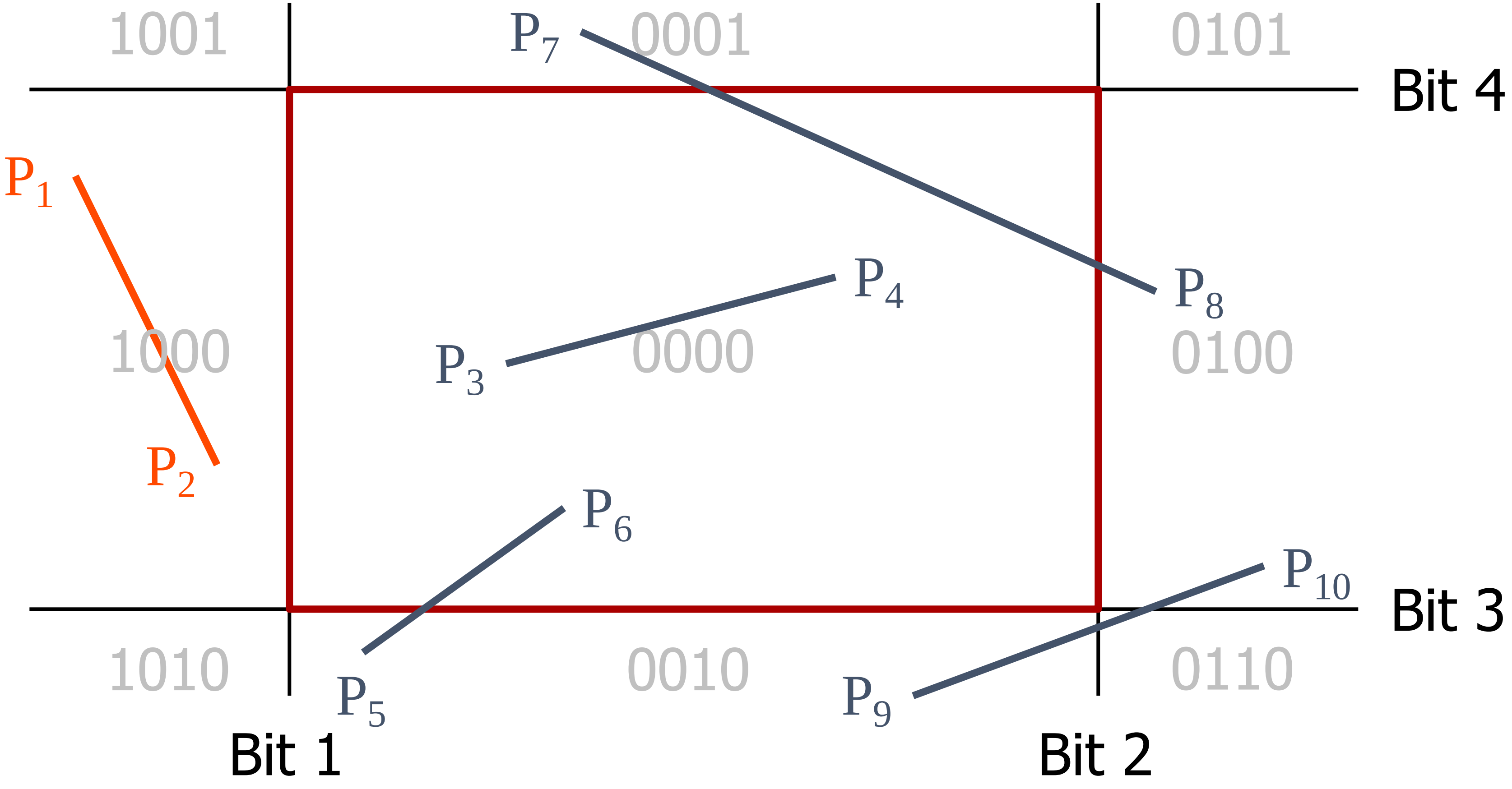


Continue...

- Classify some lines quickly by AND of bit-codes representing regions of two endpoints (must be 0).

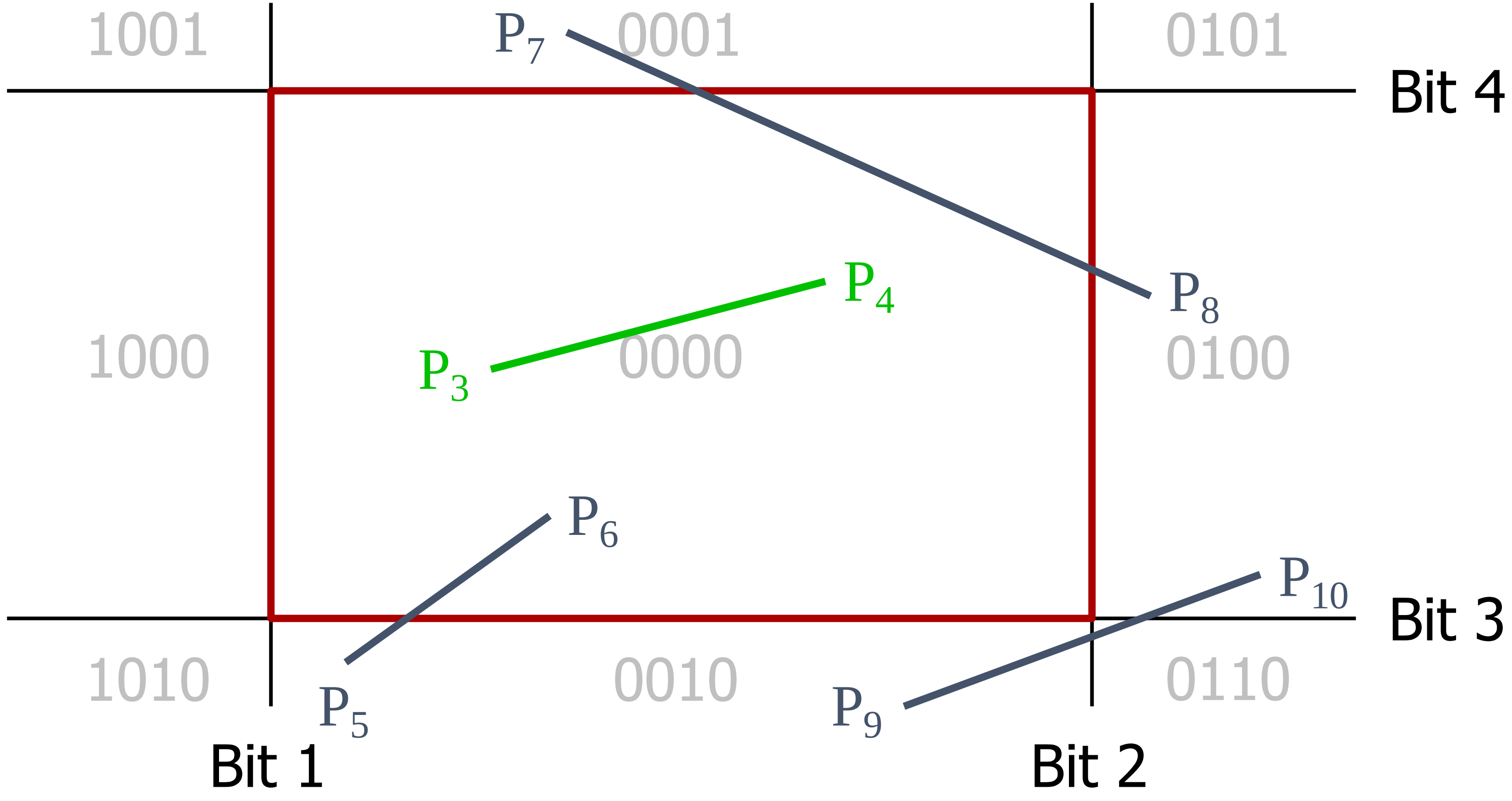


Continue...



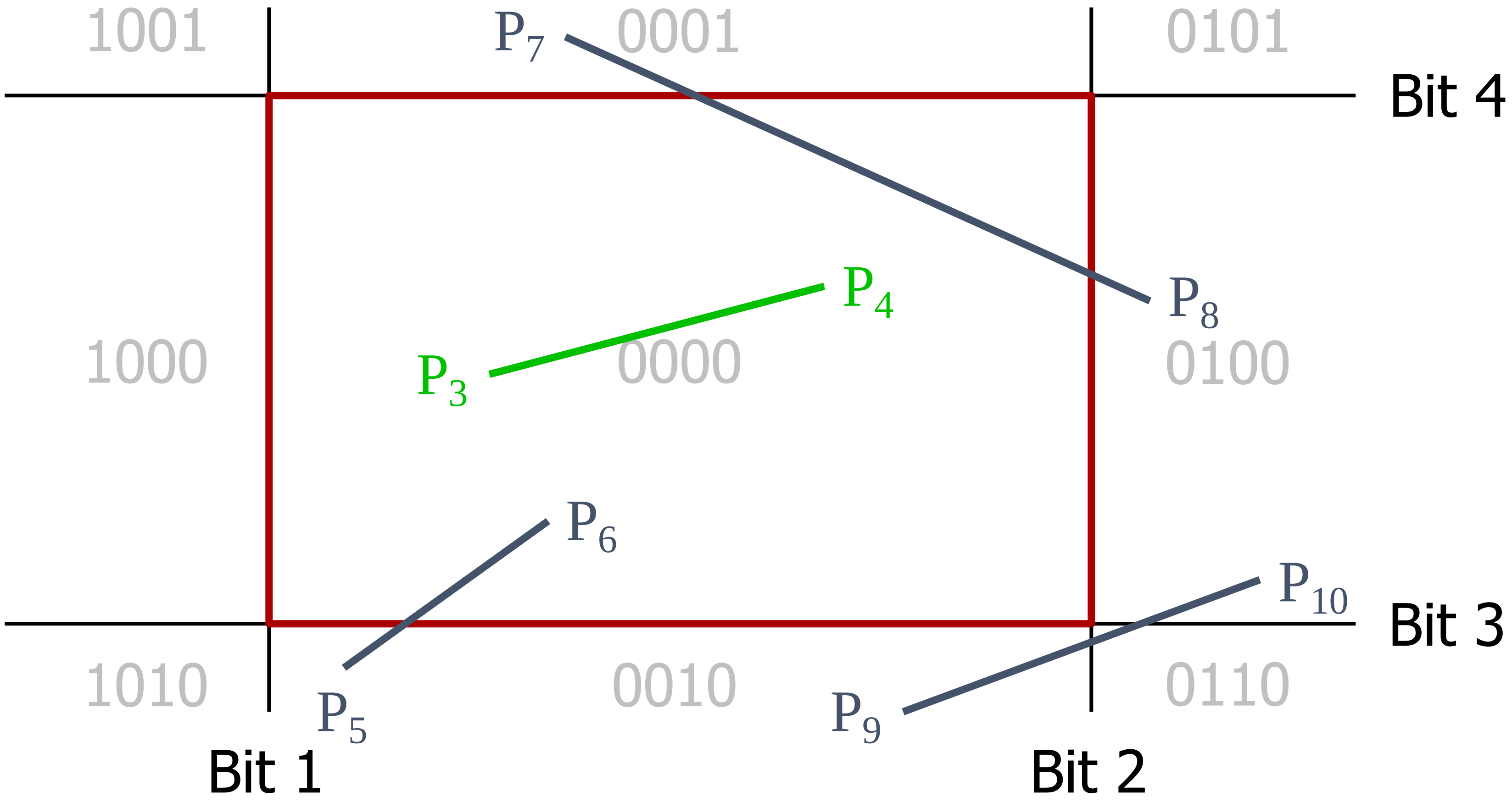
1001
P7
0001
0101
Bit 4
P1
P4
P8
1000
P3
0000
0100
P2
P6
P10
Bit 3
1010
P5
0010
P9
0110
Bit 1
Bit 2

Continue...

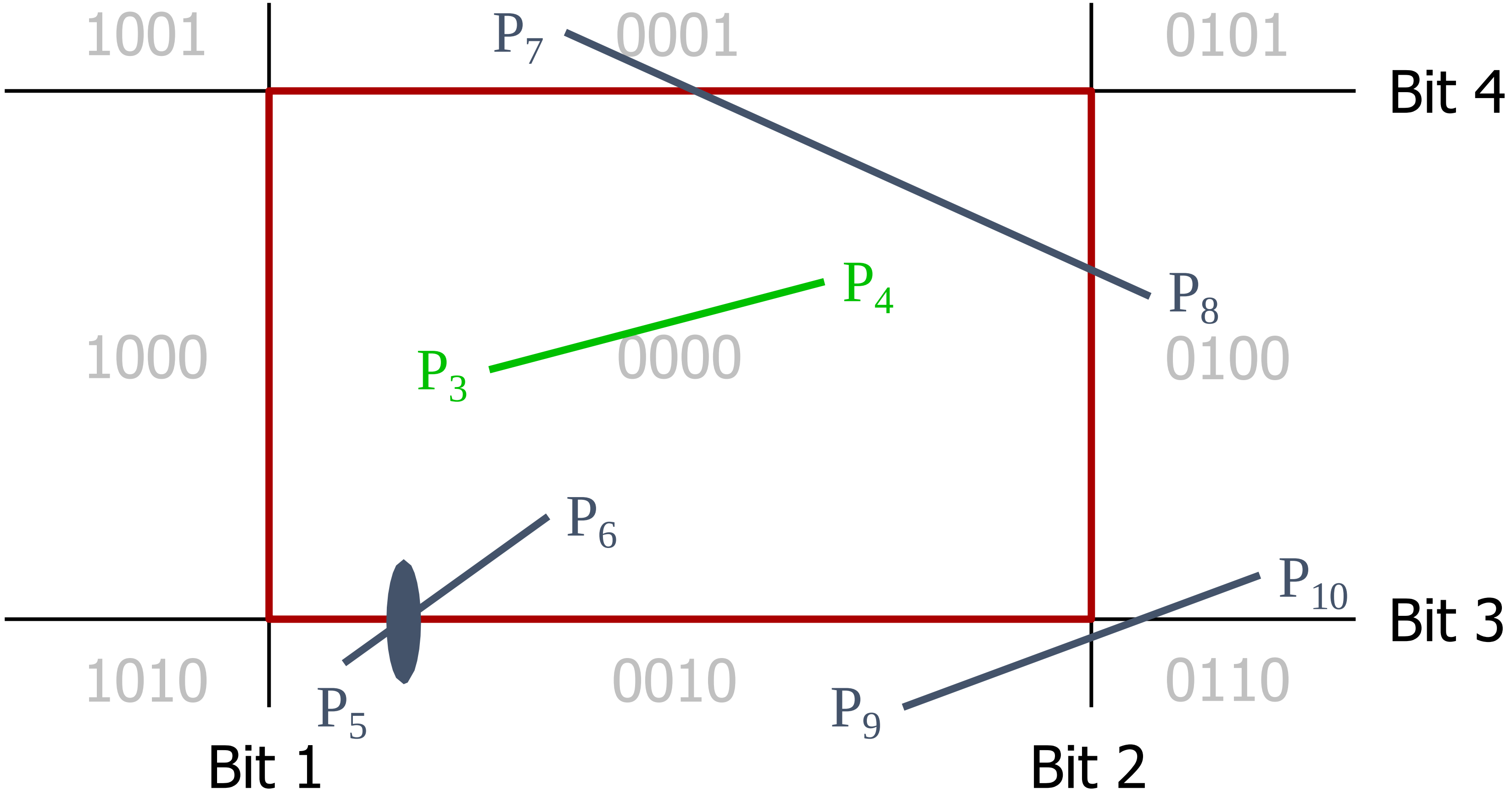

1001
P7
0001
0101
Bit 4
P4
P8
1000
P3
0000
0100
P6
P10
Bit 3
1010
P5
0010
P9
0110
Bit 1
Bit 2

Continue…

- Compute intersections with window boundary for lines that can't be classified quickly:

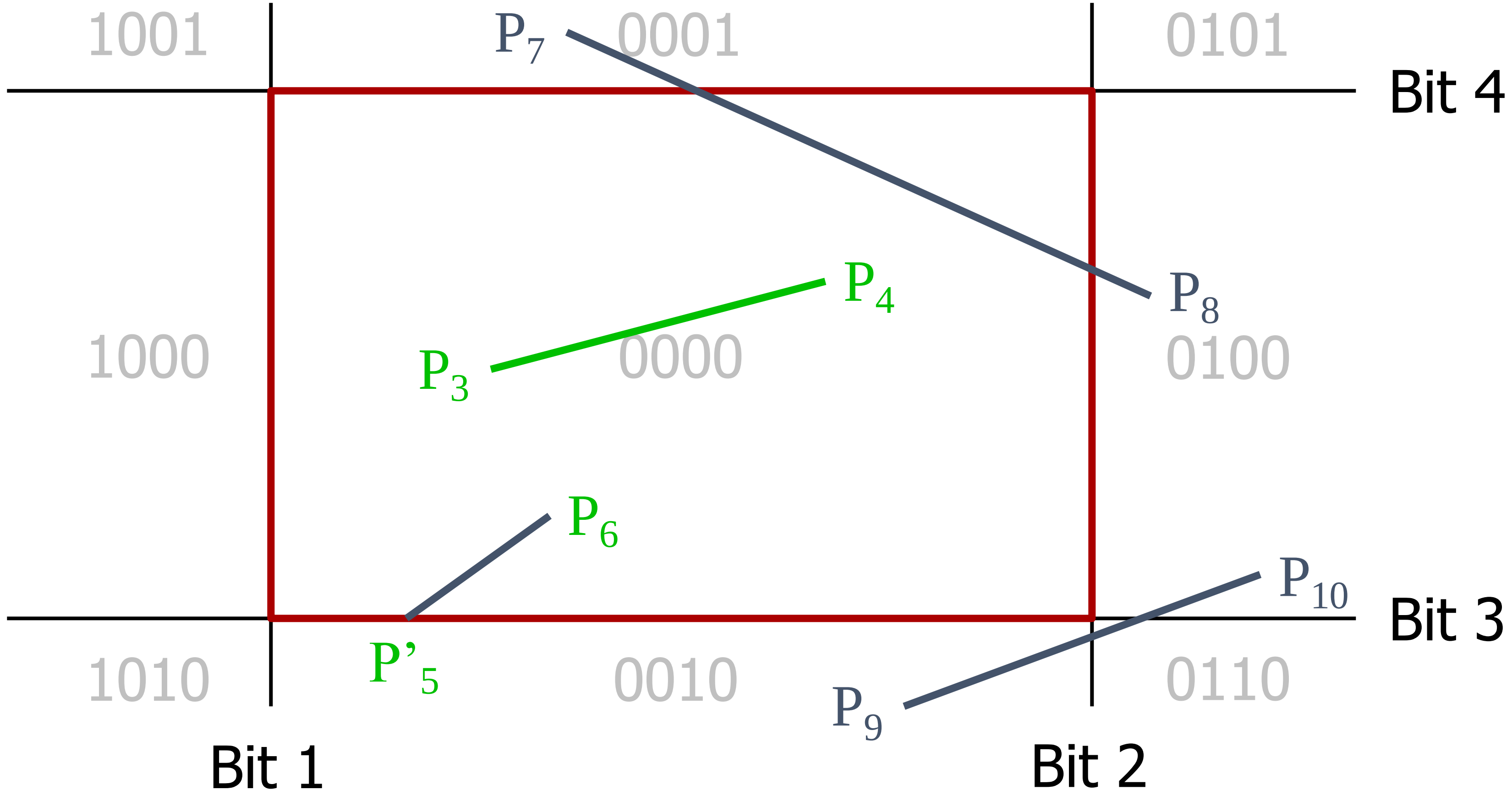


Continue...



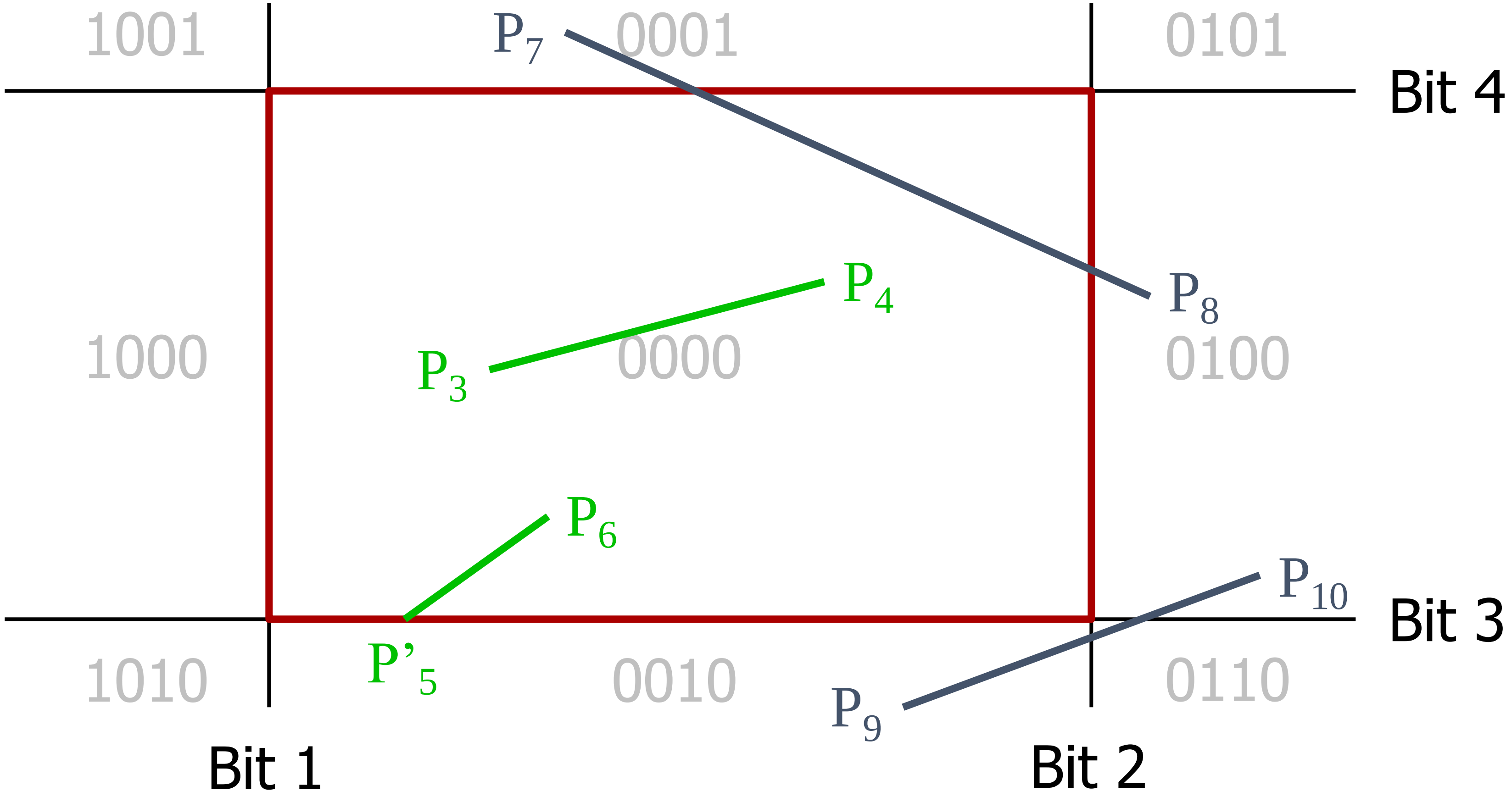
1001
P_7
0001
0101
Bit 4
P_4
P_8
1000
P_3
0000
0100
P_6
P_{10}
Bit 3
1010
0010
0110
P_5
P_9
Bit 1
Bit 2

Continue...



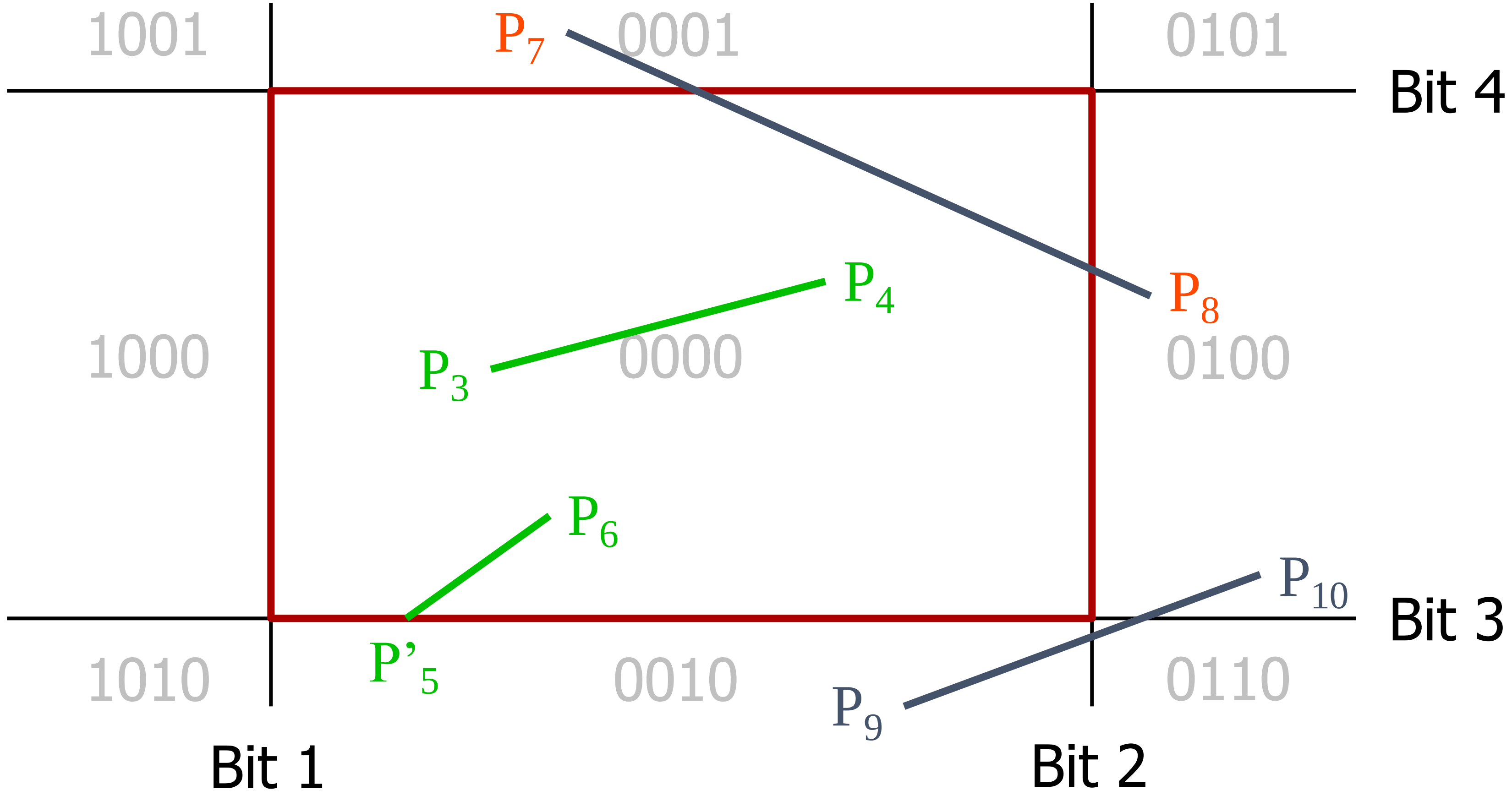
1001
P7
0001
0101
Bit 4
P4
P8
1000
P3
0000
0100
P6
P10
Bit 3
P'5
1010
0010
P9
0110
Bit 1
Bit 2

Continue...



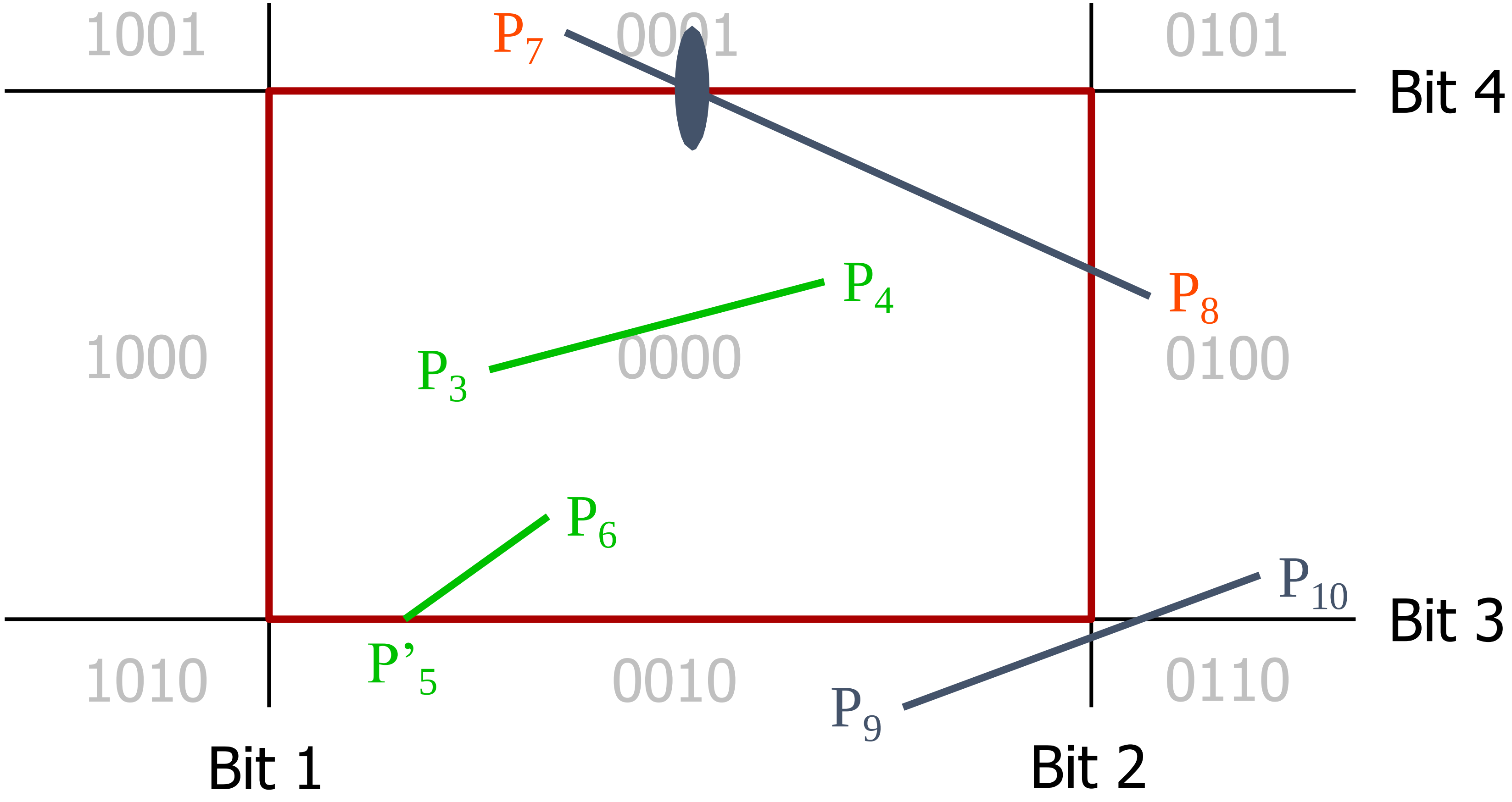
1001
P7
0001
0101
Bit 4
P4
P8
1000
P3
0000
0100
P6
P10
Bit 3
P'5
1010
0010
P9
0110
Bit 1
Bit 2

Continue...



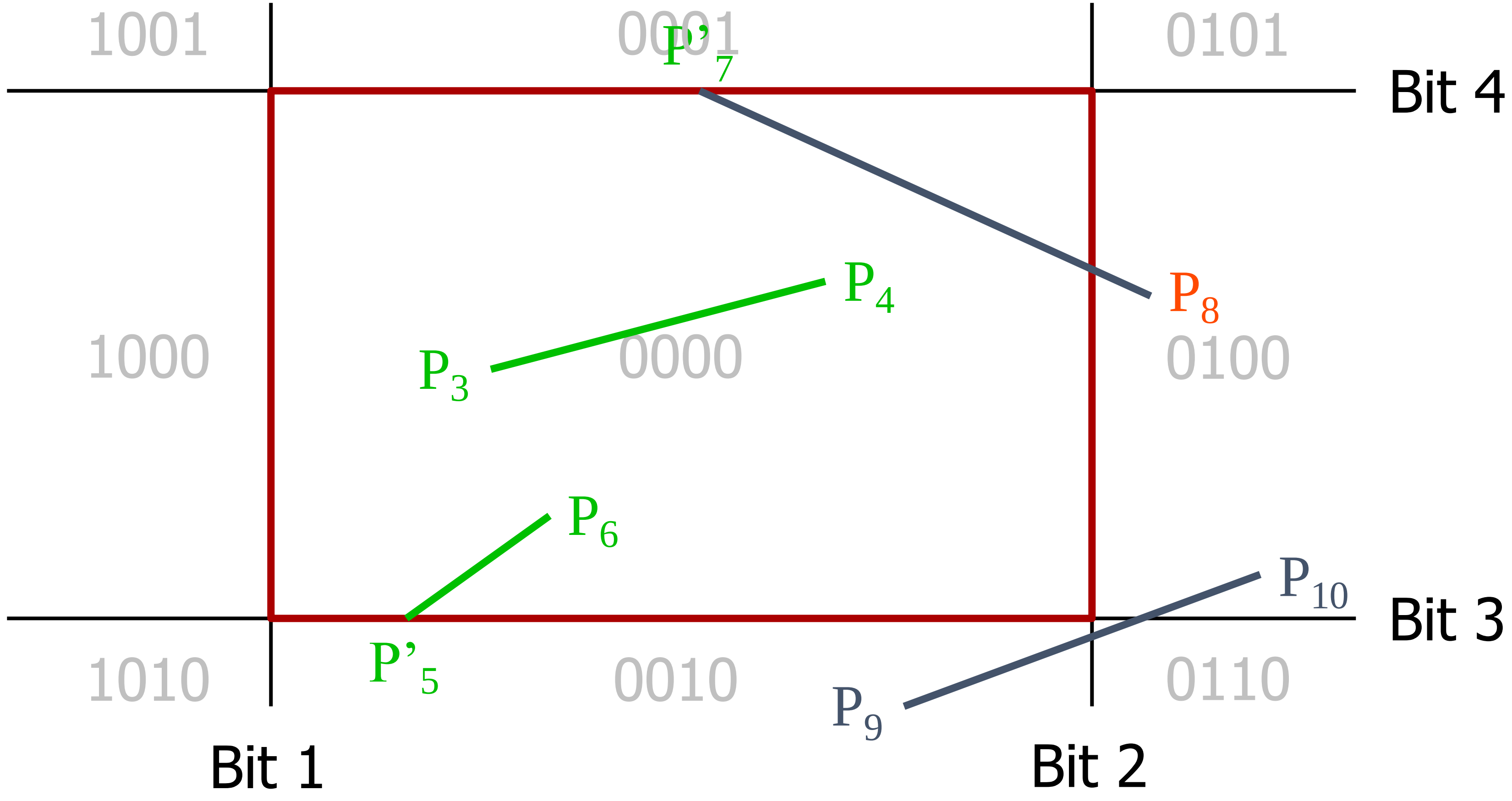
1001
P7
0001
0101
Bit 4
P4
P8
1000
P3
0000
0100
P6
P10
Bit 3
P'5
1010
0010
P9
0110
Bit 1
Bit 2

Continue...



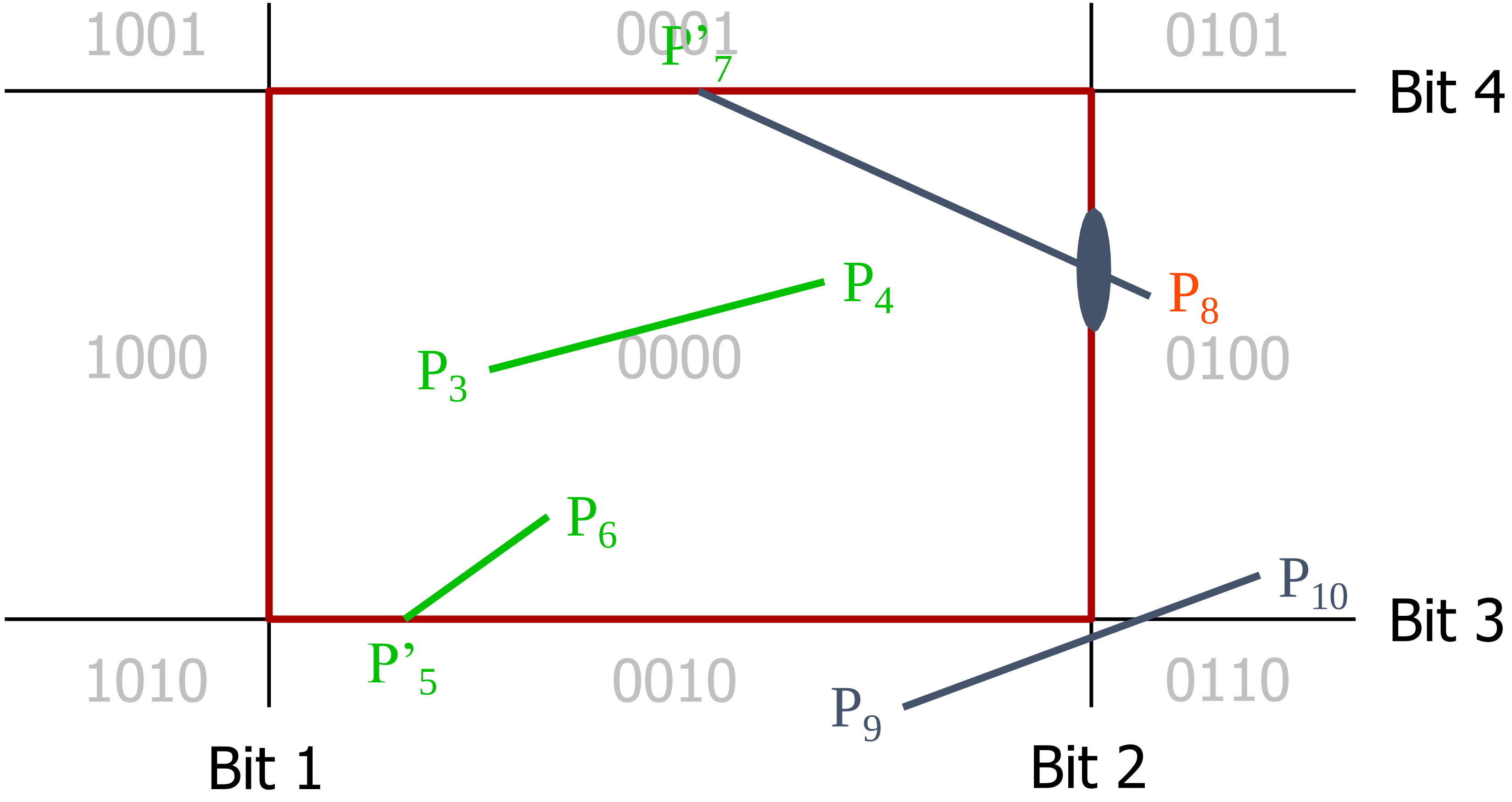
1001
P7
0001
0101
Bit 4
P4
P8
1000
P3
0000
0100
P6
P10
Bit 3
P'5
1010
0010
P9
0110
Bit 1
Bit 2

Continue…



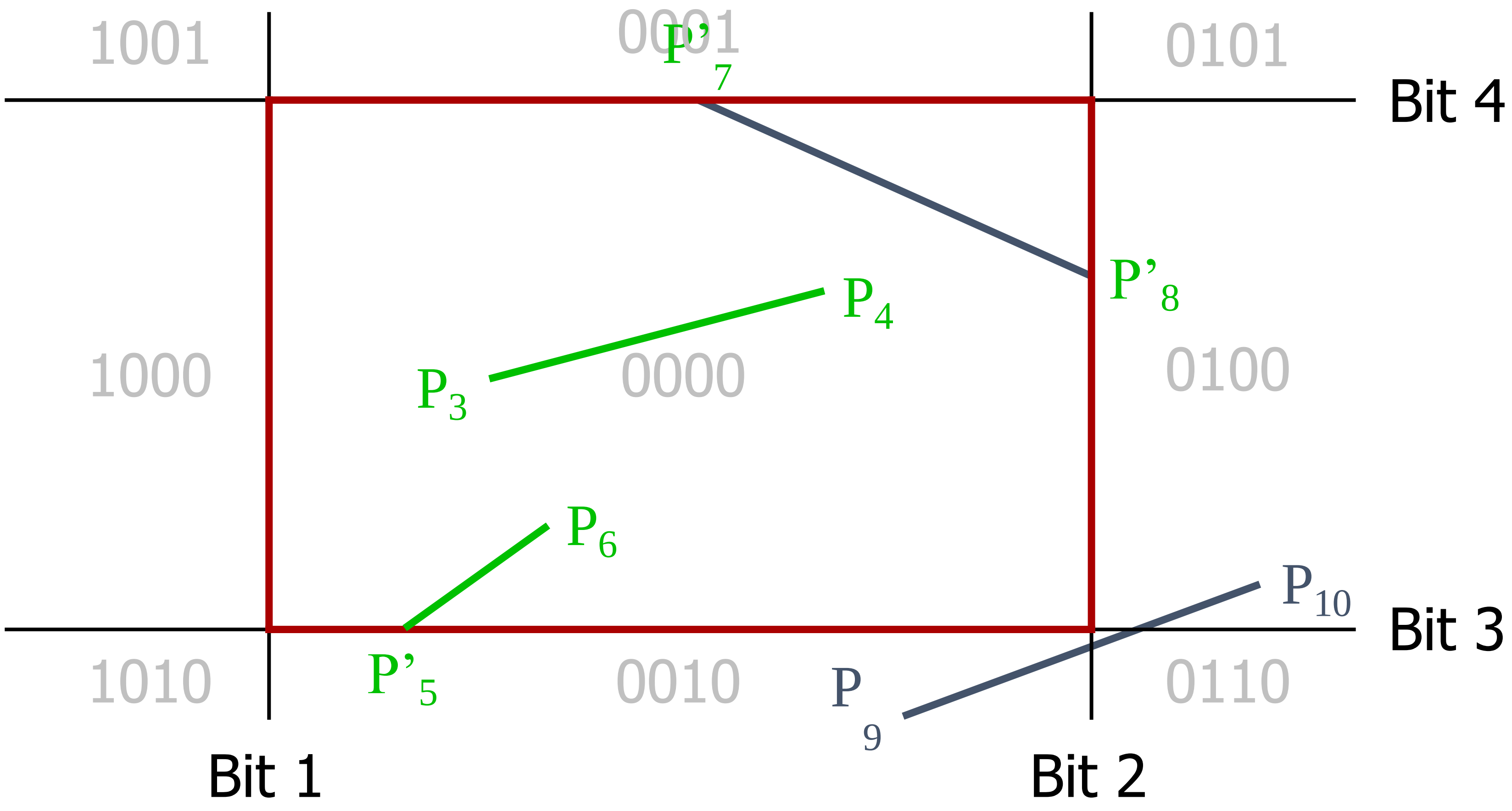
1001
0001
0101
P'7
Bit 4
P4
P8
1000
P3
0000
0100
P6
P10
Bit 3
P'5
1010
0010
0110
P9
Bit 1
Bit 2

Continue...



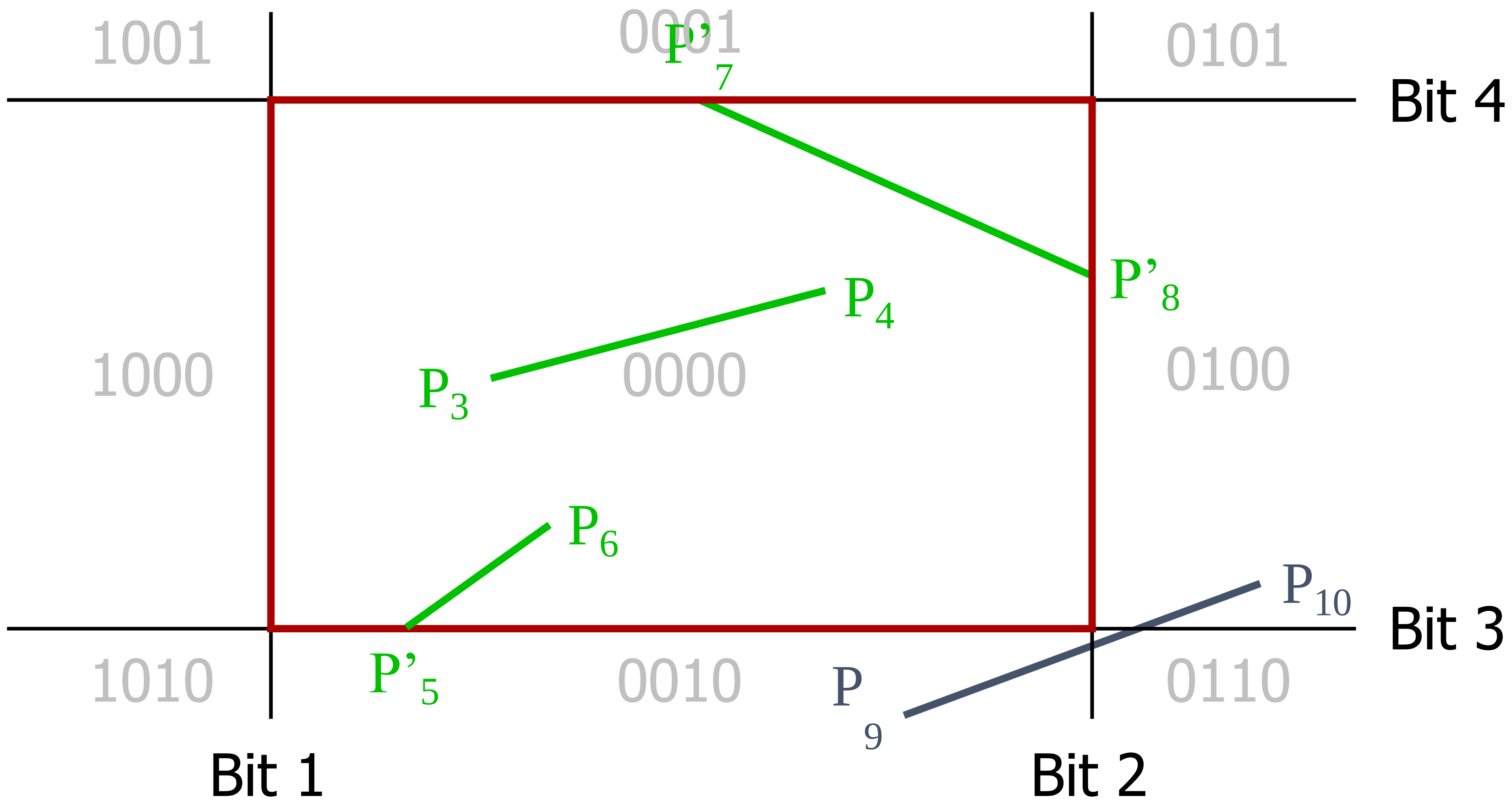
1001
0001
0101
P'7
Bit 4
P4
P8
1000
P3
0000
0100
P6
P10
Bit 3
P'5
1010
0010
P9
0110
Bit 1
Bit 2

Continue...

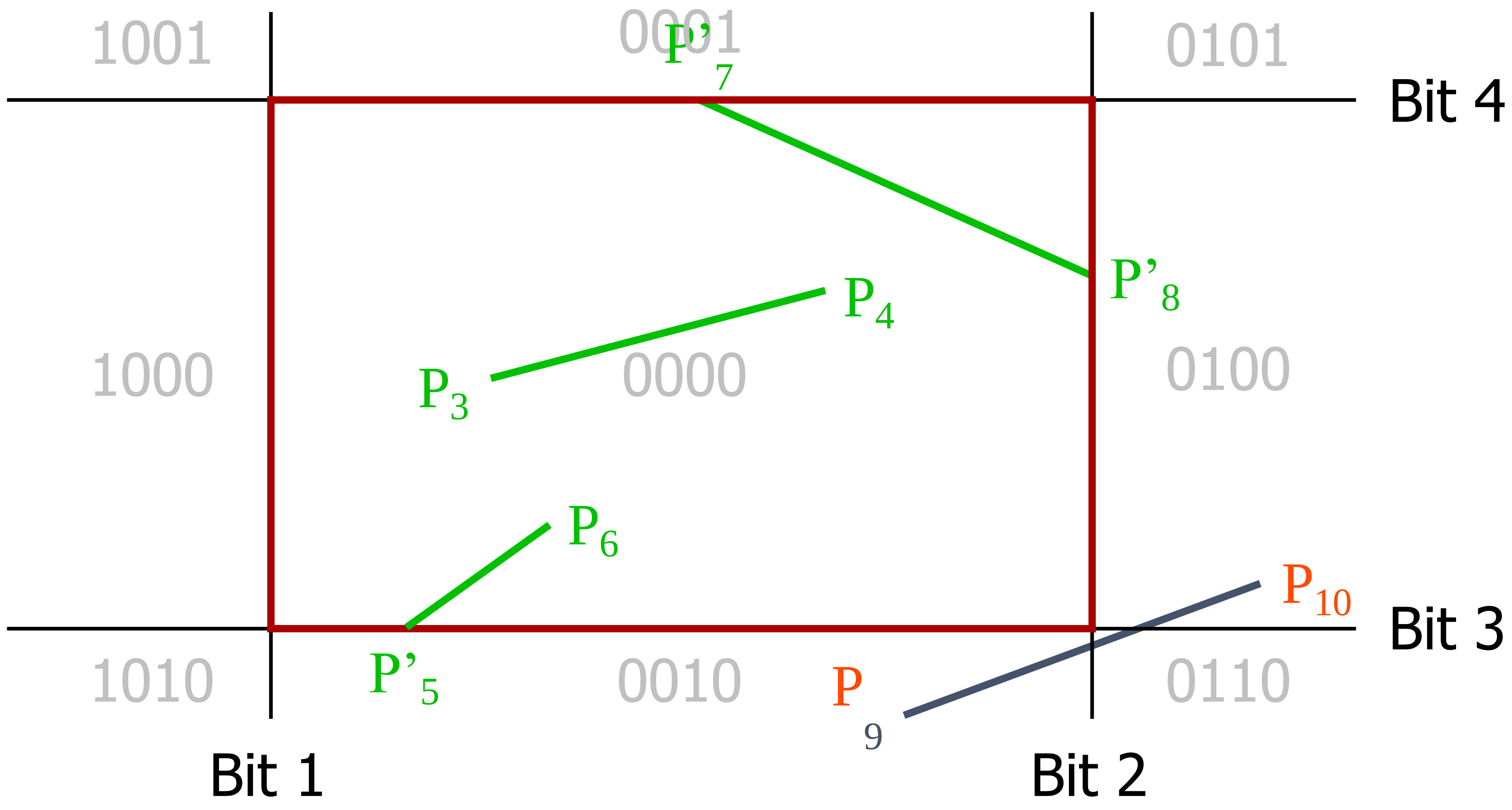


1001
0001
0101
P'7
Bit 4
P'8
P4
1000
P3
0000
0100
P6
P10
Bit 3
1010
P'5
0010
P
9
0110
Bit 1
Bit 2

Continue...

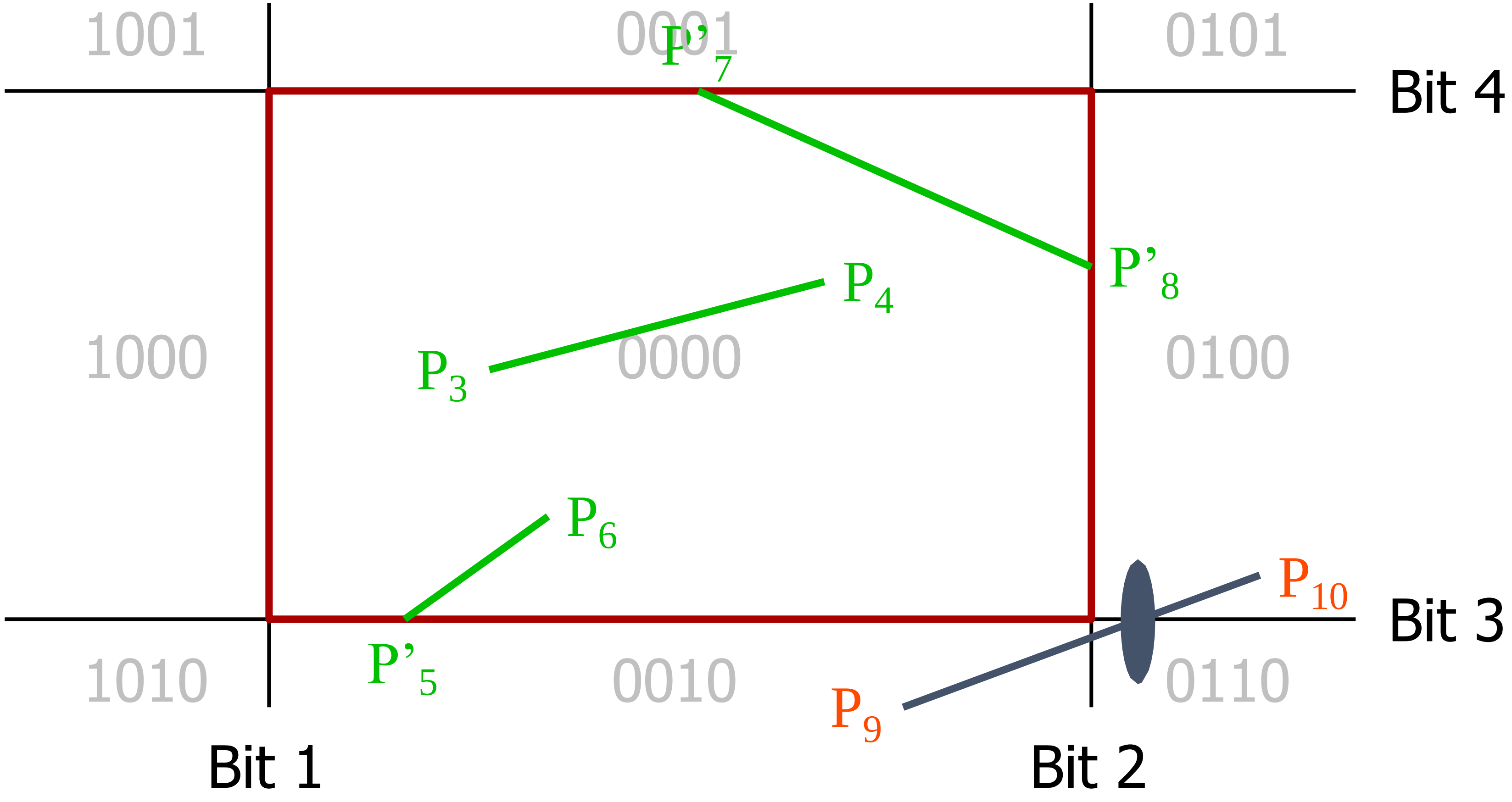

1001
0001
0101
Bit 4
P'7
P'8
P4
1000
P3
0000
0100
P6
P10
Bit 3
P'5
1010
0010
P
9
0110
Bit 1
Bit 2

Continue...



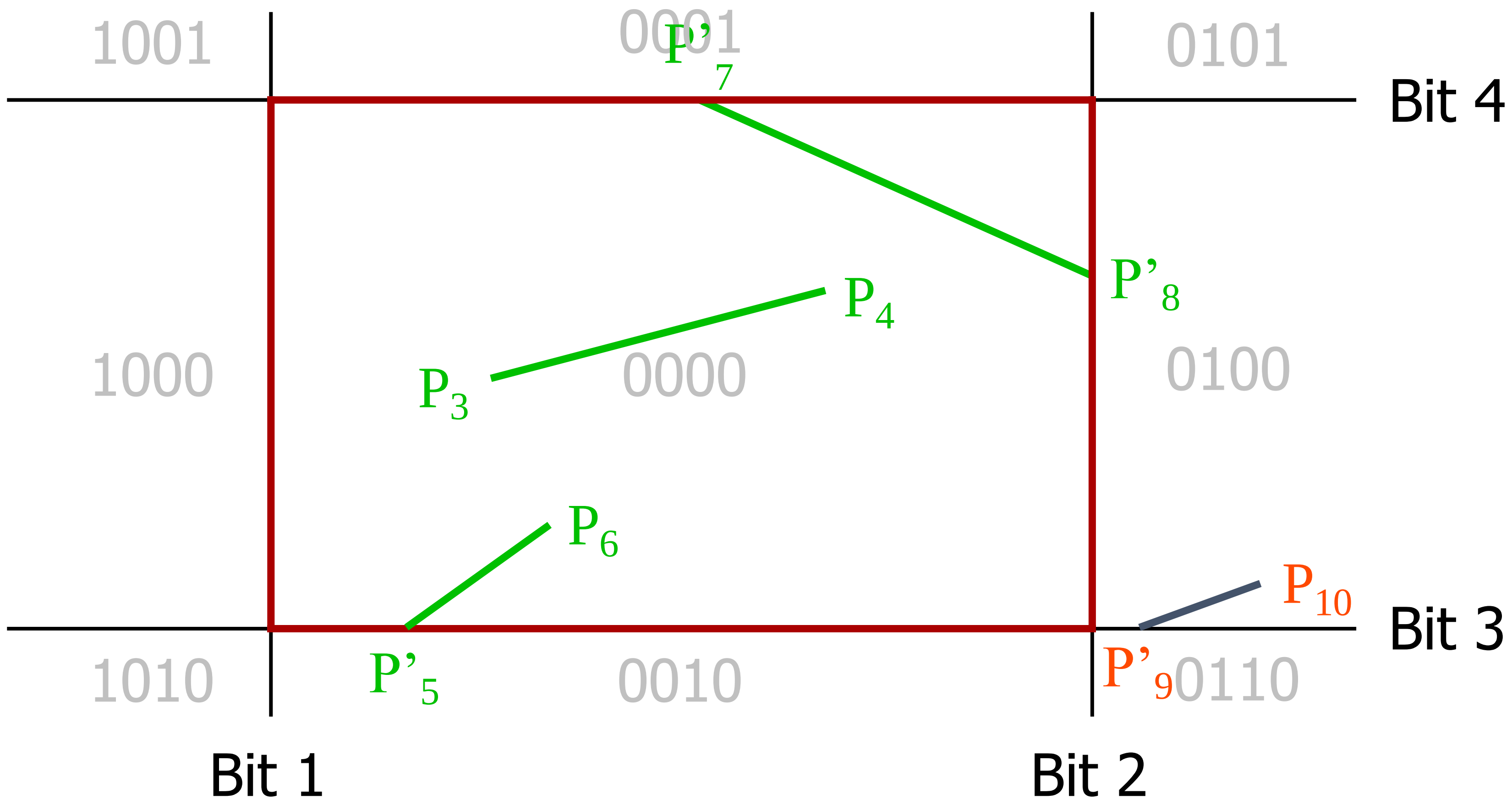
1001
0001
P'7
0101
Bit 4
P'8
P4
1000
P3
0000
0100
P6
P10
Bit 3
P'5
1010
0010
P9
0110
Bit 1
Bit 2

Continue…



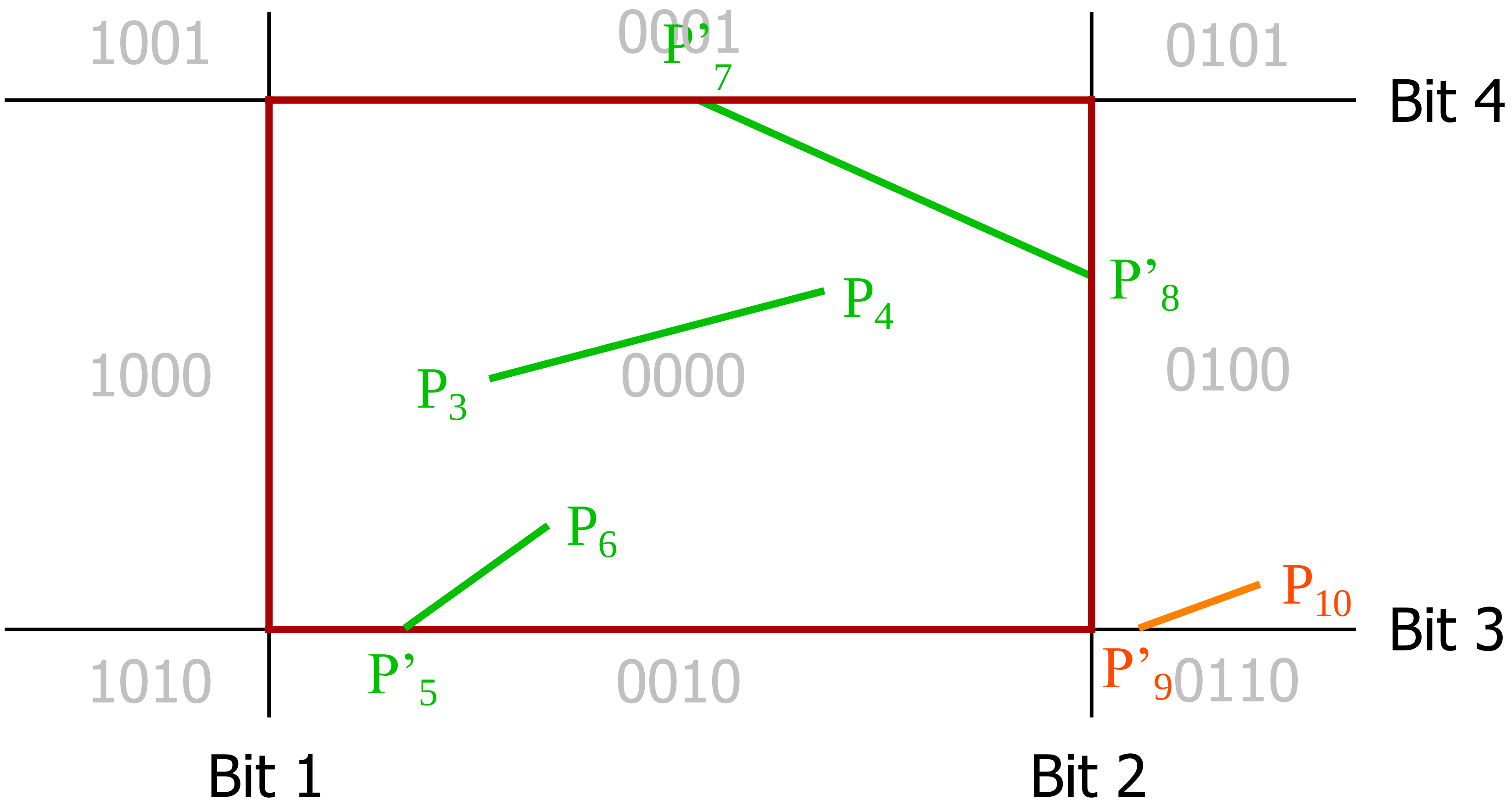
1001
0001
0101
P'7
Bit 4
P'8
P4
1000
P3
0000
0100
P6
P10
Bit 3
1010
P'5
0010
P9
0110
Bit 1
Bit 2

Continue…

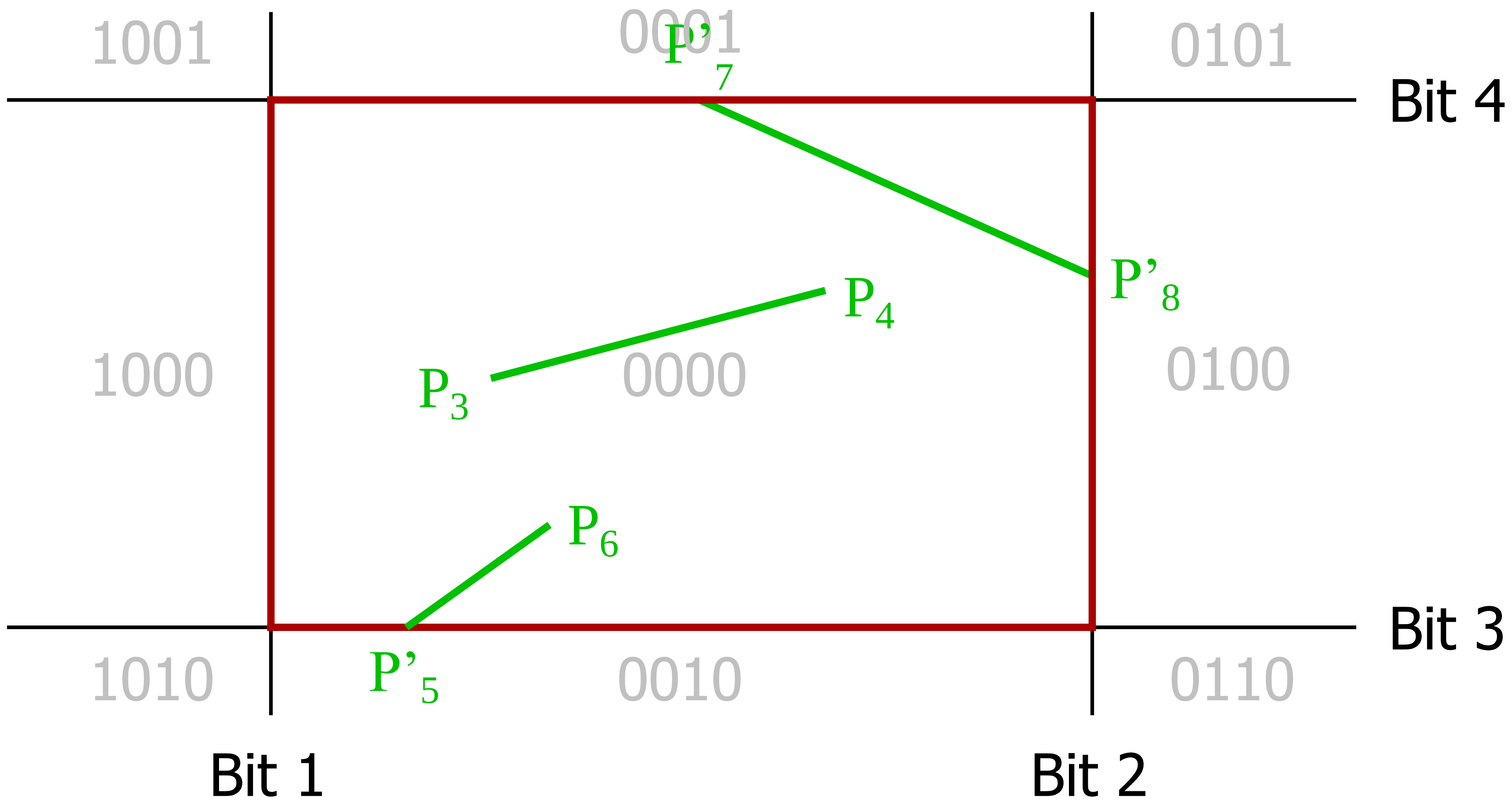


1001
0001
P'7
0101
Bit 4
P'8
P4
1000
P3
0000
0100
P6
P10
Bit 3
P'5
1010
0010
P'9
0110
Bit 1
Bit 2

Continue...


1001
0001
P'7
0101
Bit 4
P'8
P4
1000
P3
0000
0100
P6
P10
Bit 3
P'5
1010
0010
P'9
0110
Bit 1
Bit 2

Continue...



1001
0001
0101
P'7
Bit 4
P'8
P4
1000
P3
0000
0100
P6
Bit 3
P'5
1010
0010
0110
Bit 1
Bit 2

Midpoint Subdivision

- An alternative way to process a line in category 3 is based on binary search.
- The line is divided at its midpoint into two shorter line segments.
- The clipping categories of the two new line segments are determined by their region codes.
- Each segment in category 3 is divided again into shorter segments and categorized.
- This bisection & categorization process continues until each line segment that spans across a window boundary reaches a threshold for line size and all other segments are either in category 1 or in category 2.

Midpoint Subdivision Algorithm

- Step-1: Calculate the position of both endpoints of the line.
- Step-2: Perform OR operation on both of these endpoints.
- Step-3: If the OR operation gives 0000:

  then-

  Line is guaranteed to be visible;

  else-

  Perform AND operation on both endpoints.

  If AND ≠ 0000-

  the line is invisible;

  else

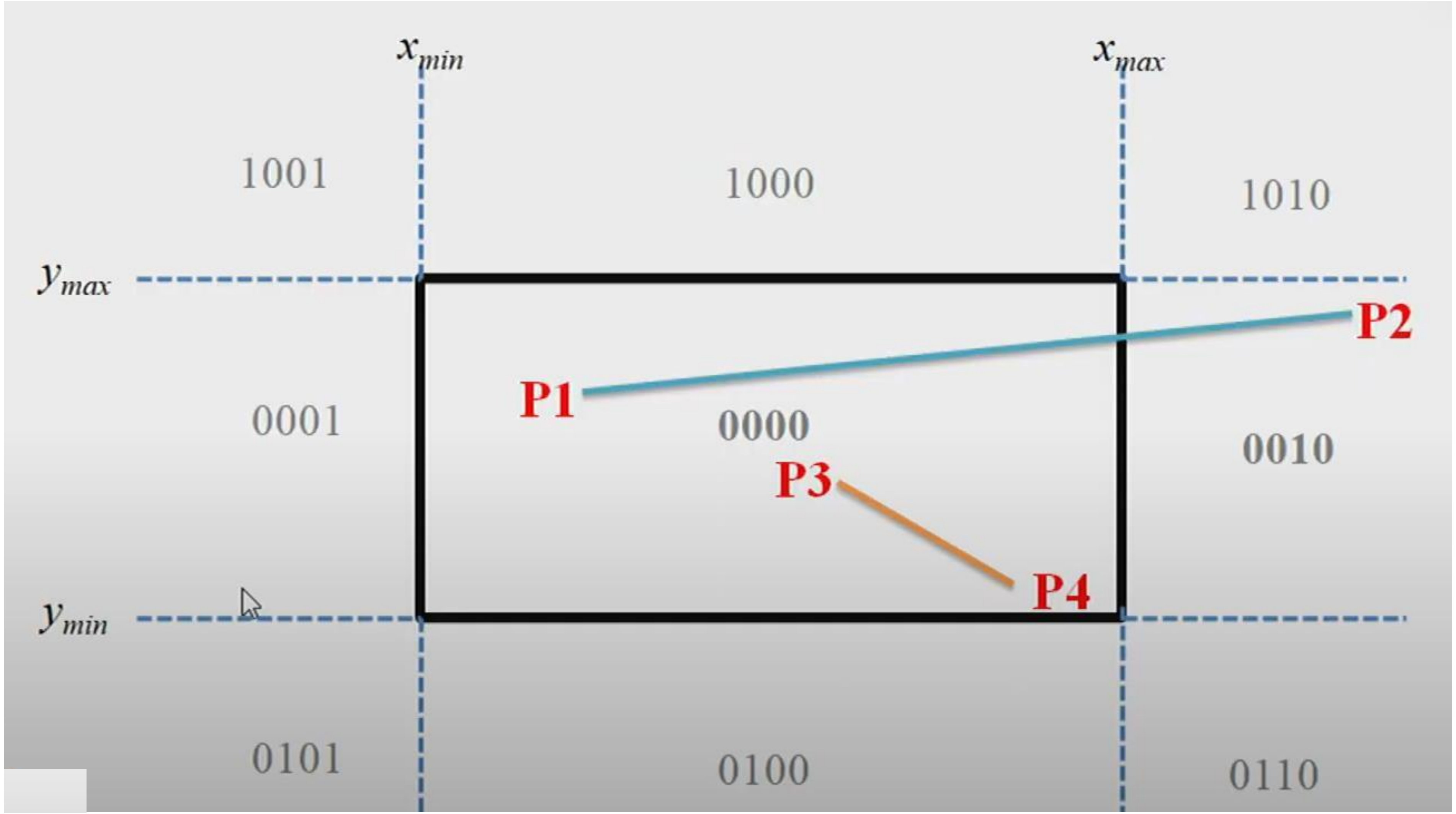
  the line is clipped case;
- Step-4: For the line to be clipped. Find midpoint.

  $X_m=(x_1+x_2)/2$

  $Y_m=(y_1+y_2)/2$
- Step-5: Check each midpoint, whether it nearest to the boundary of a window or not.
- Step-6: If the line is totally visible or totally rejected not found:

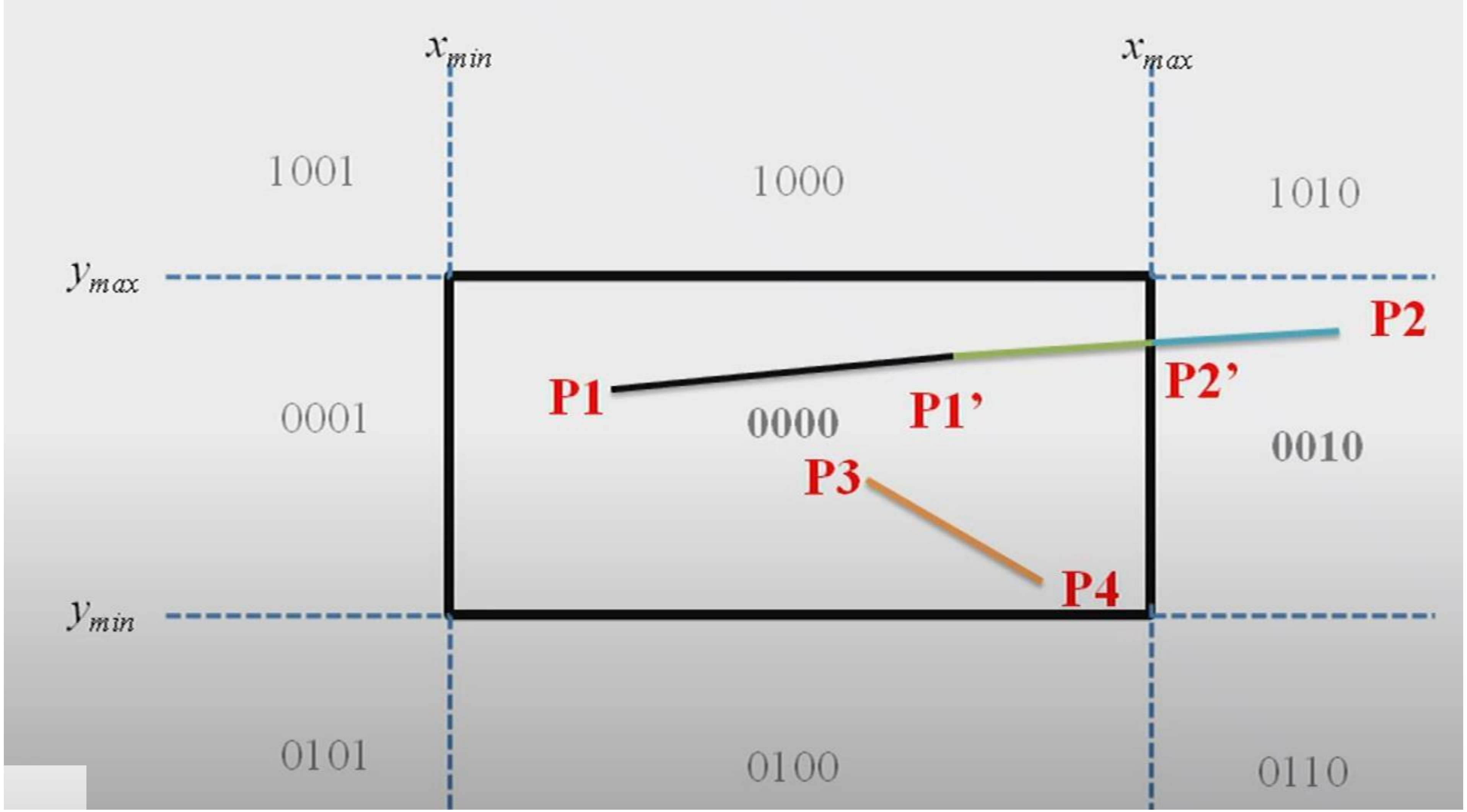
  Repeat step 1 to 5.
- Step-7: Stop algorithm.

Example - 01



x_{min}
x_{max}
y_{max}
y_{min}
1001
1000
1010
0001
0000
0010
0101
0100
0110
P1
P2
P3
P4

Continue…


x_{min}
x_{max}
1001
1000
1010
y_{max}
P2
P1
P2'
P1'
0001
0000
0010
P3
P4
y_{min}
0101
0100
0110

Thank you