Two Dimensional and Three Dimensional Transformations & Viewing

By

Hari Prashad Pant

Two Dimensional Viewing

Two Dimensional Viewing

Basic Interactive Programming

Henry Ford Int'l College, Kalanki, Kathmandu
By: Hari Prashad Pant

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

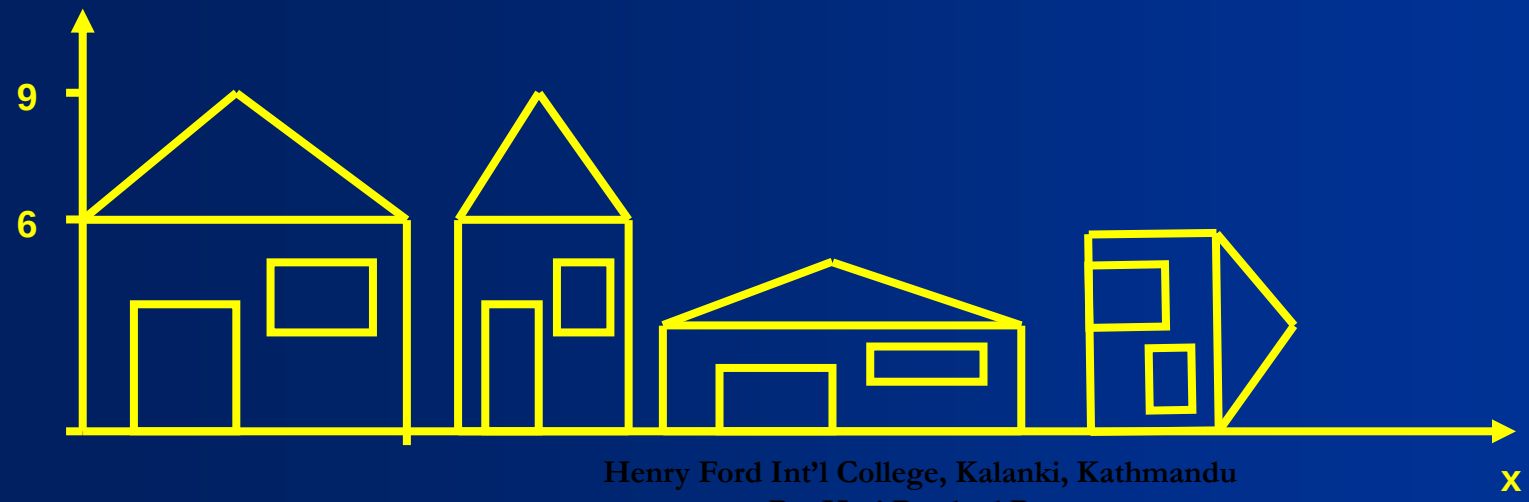
- In Modeling, we often use a geometric model
  - i.e.. A description of an object that provides a numerical description of its shape, size and various other properties.
- Dimensions of the object are usually given in units appropriate to the object:
  - meters for a ship
  - kilometres for a country

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 Geometric Transformations

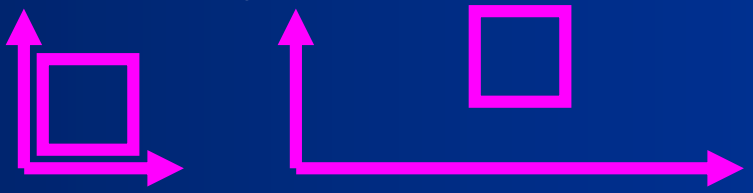
Henry Ford Int'l College, Kalanki, Kathmandu
By: Hari Prashad Pant

2D Geometric Transformations

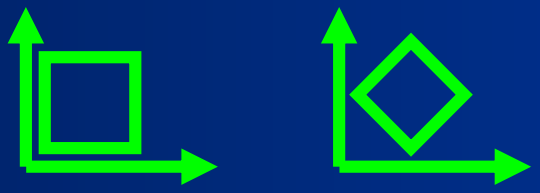
- Operations that are applied to the geometric description of an object to change its position, orientation or size.

Basic transformation:

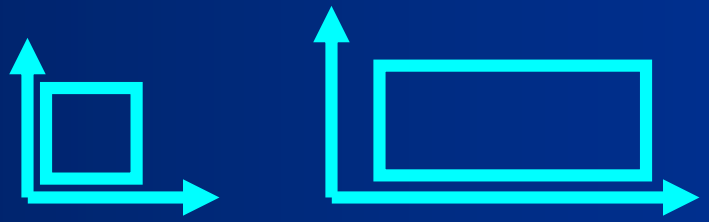
- **Translation**



- **Rotation**

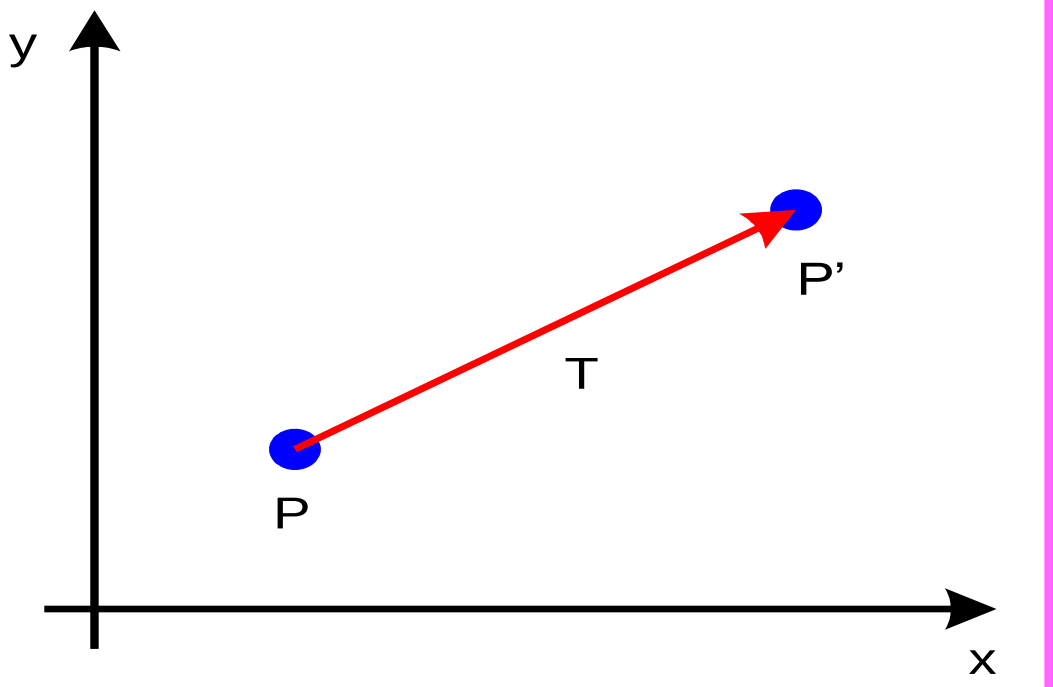


- **Scaling**



2D Translation

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



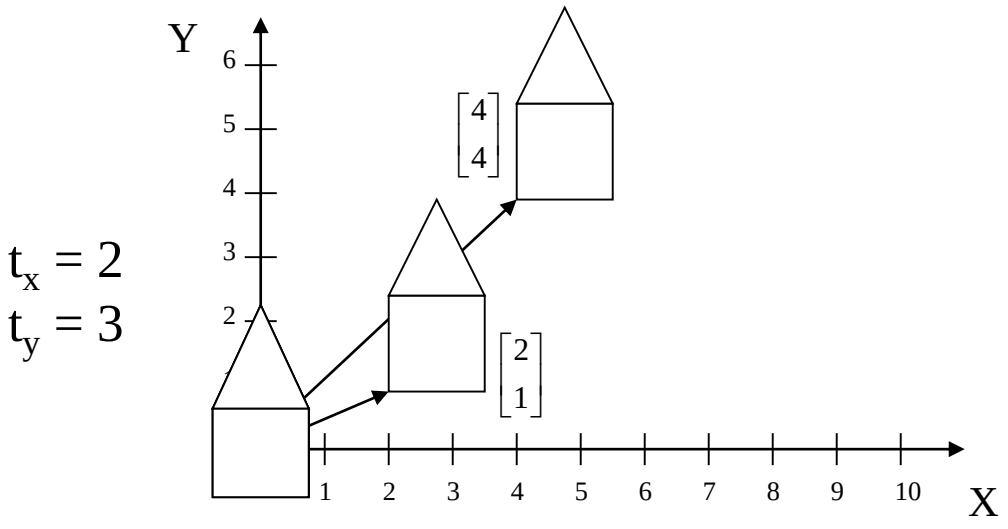
$$x' = x + t_x, \quad y' = y + t_y$$

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

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

2D Tranlation

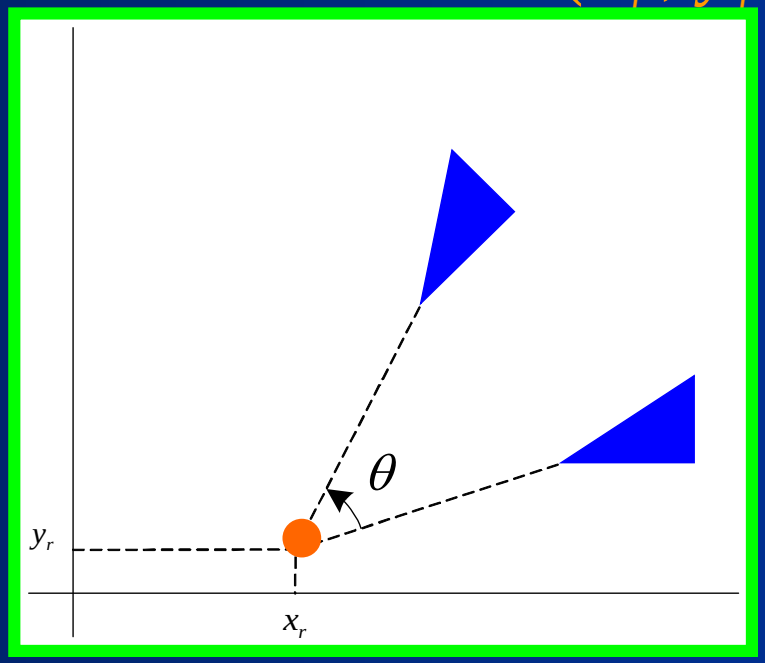
- **Rigid-body translation**: moves objects without deformation (every point of the object is translated by the same amount)



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))

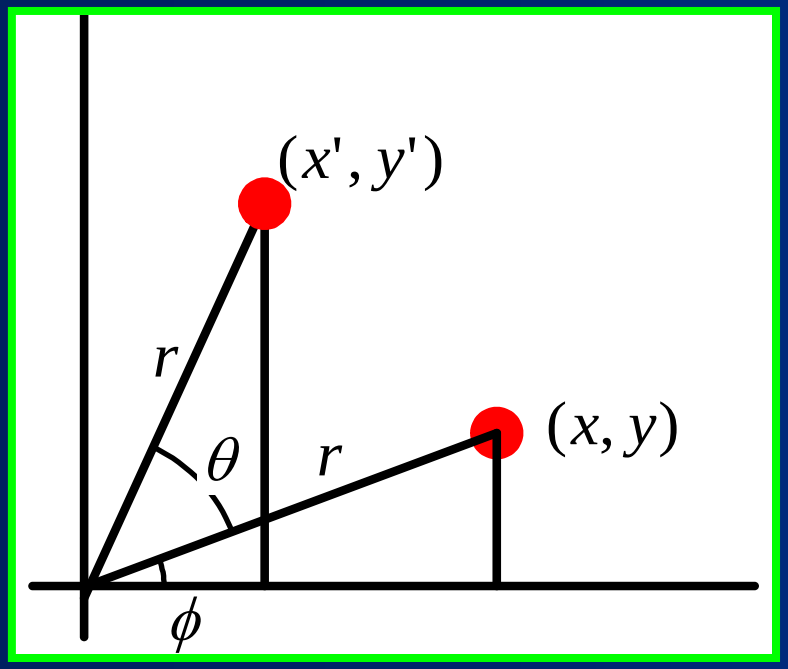


2D Rotation

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

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

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



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

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

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

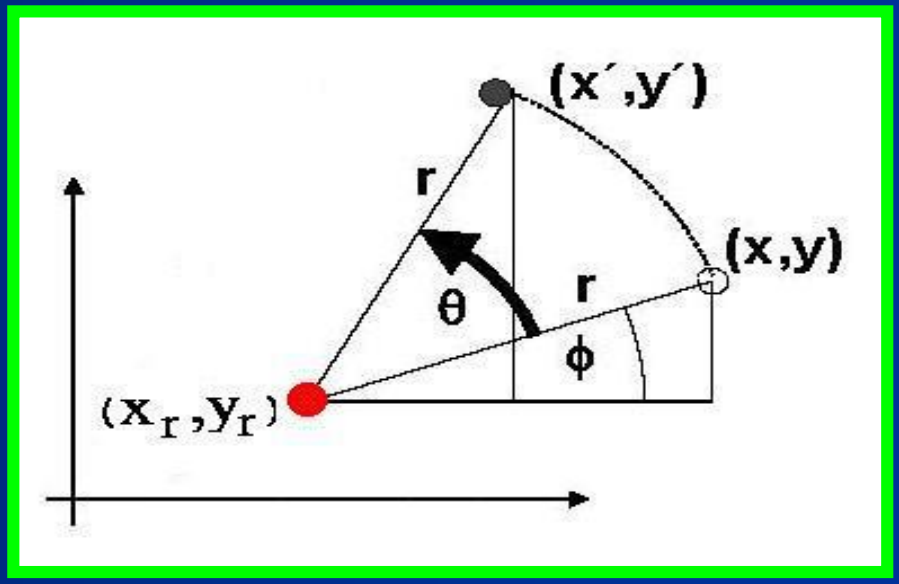
2D Rotation

- Rotation of a point about an arbitrary pivot position:

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

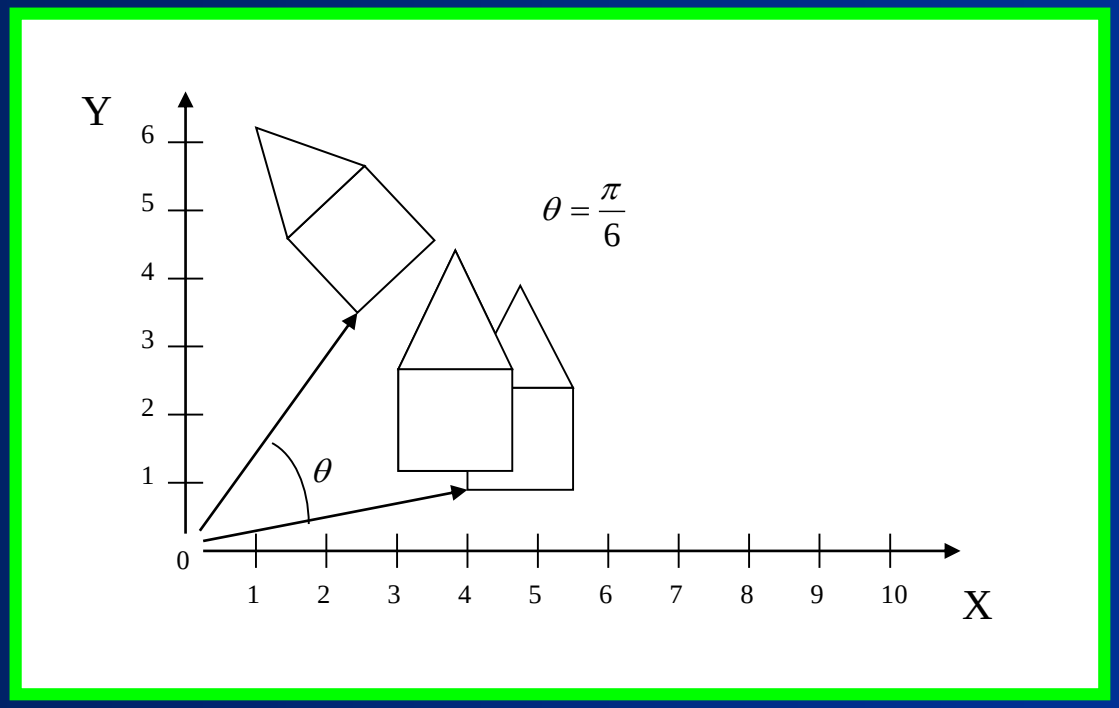
$$y' = y_r + (x - x_r)\sin\theta + (y - y_r)\cos\theta$$

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



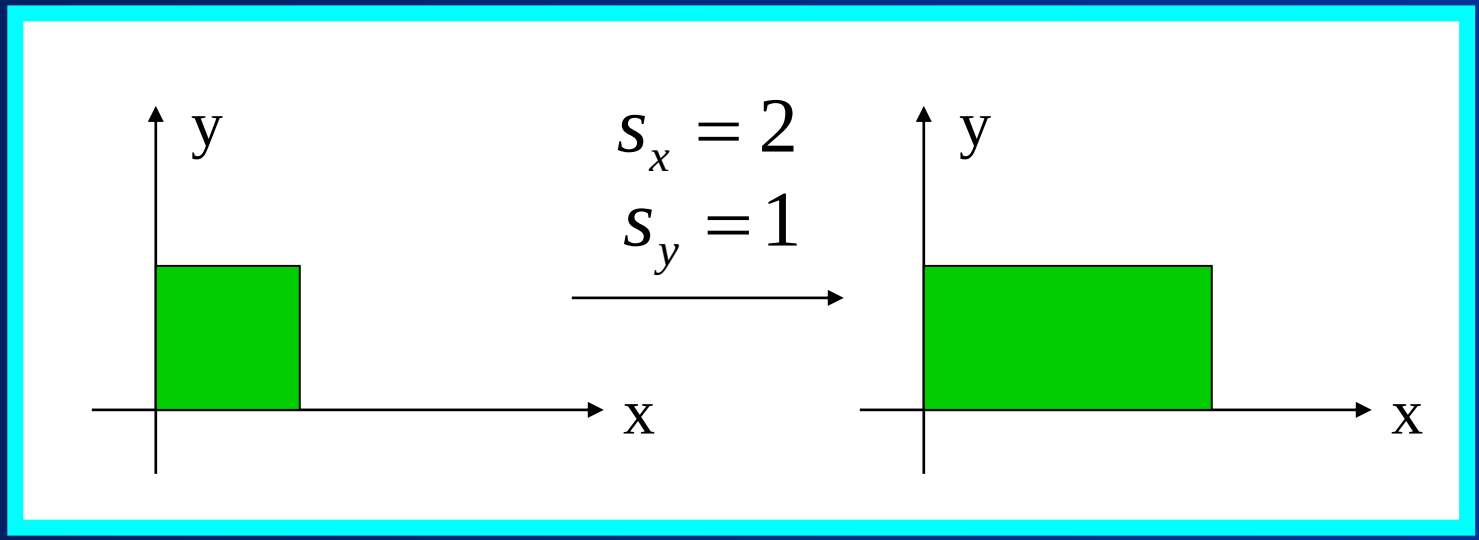
2D Rotation

- **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

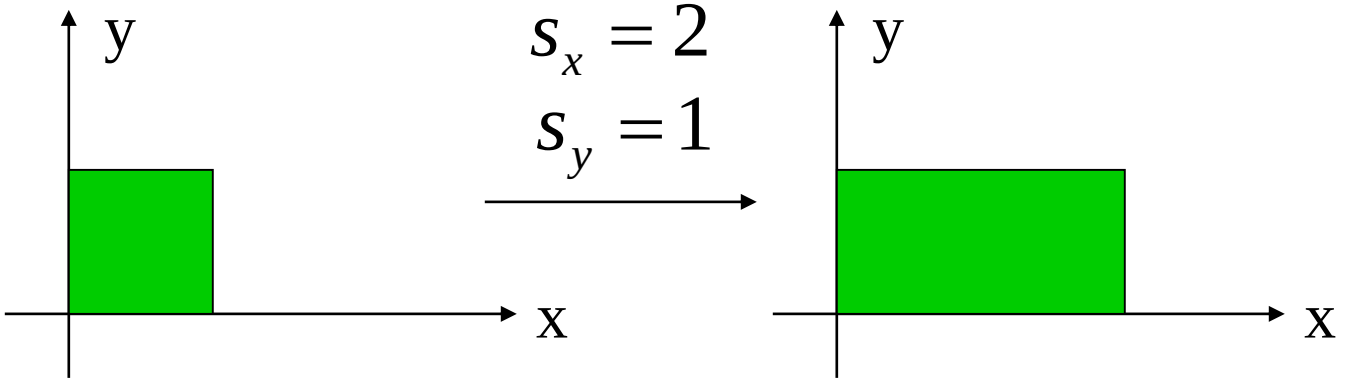


2D Scaling

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

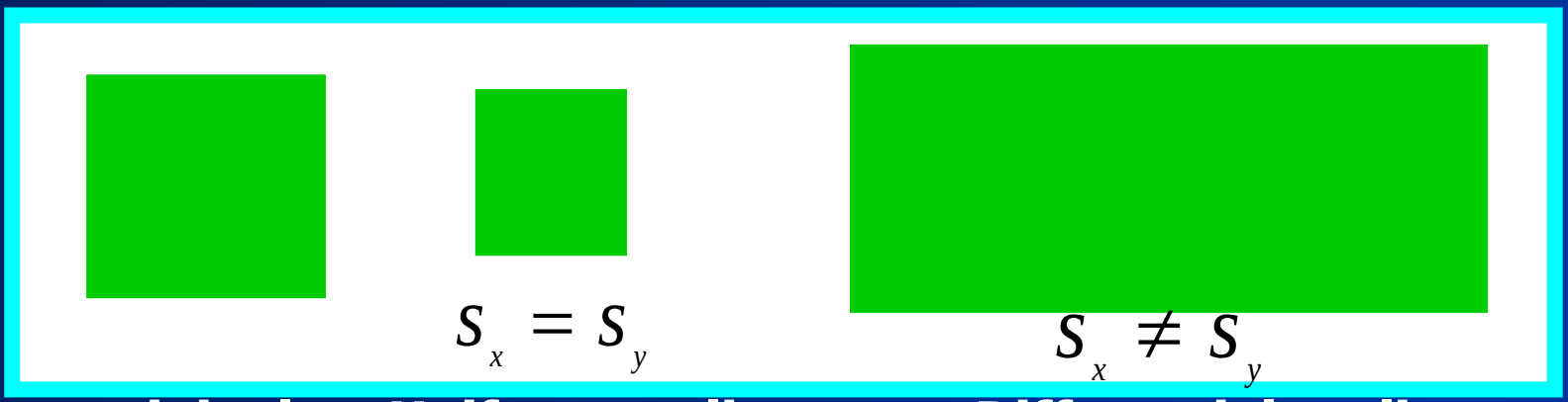
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

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



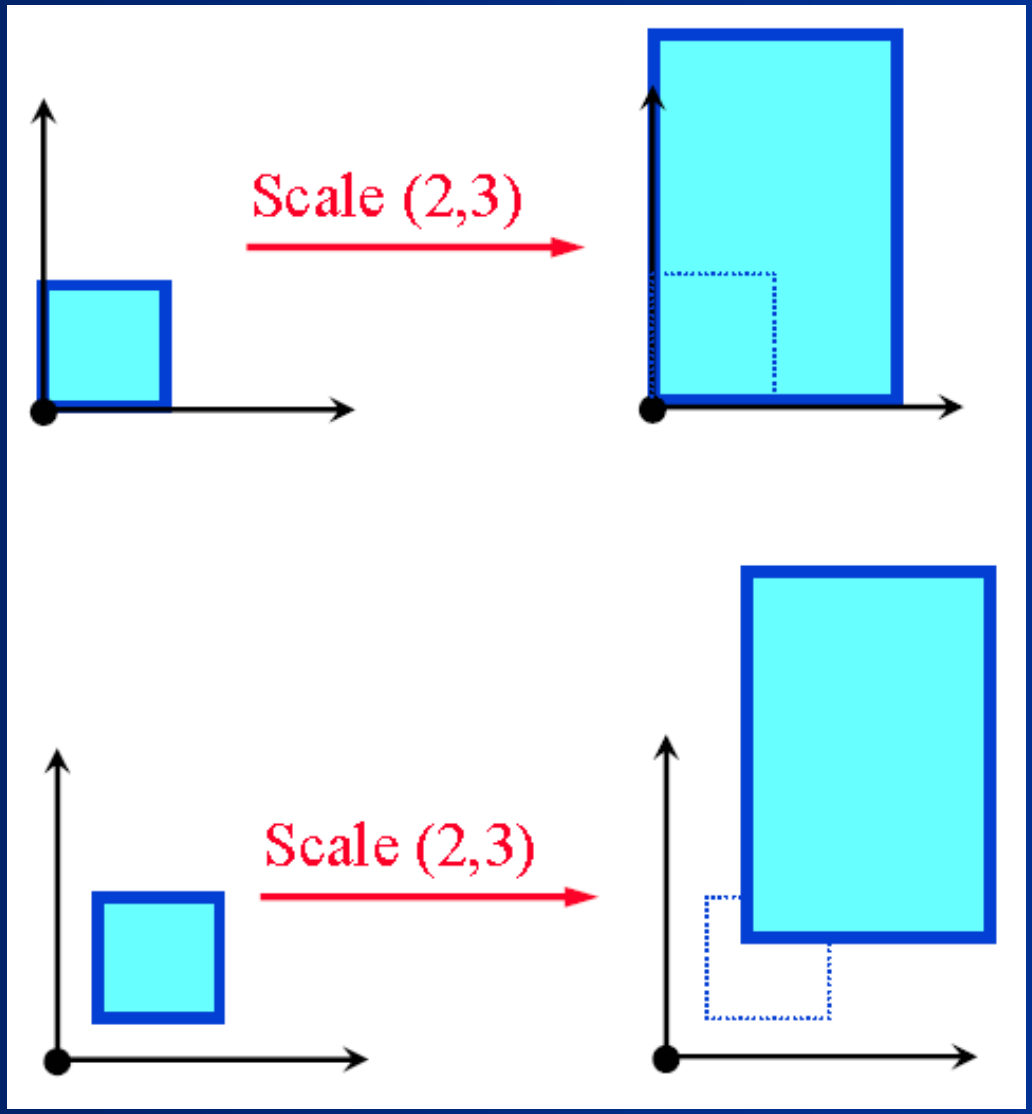
2D Scaling

- 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.



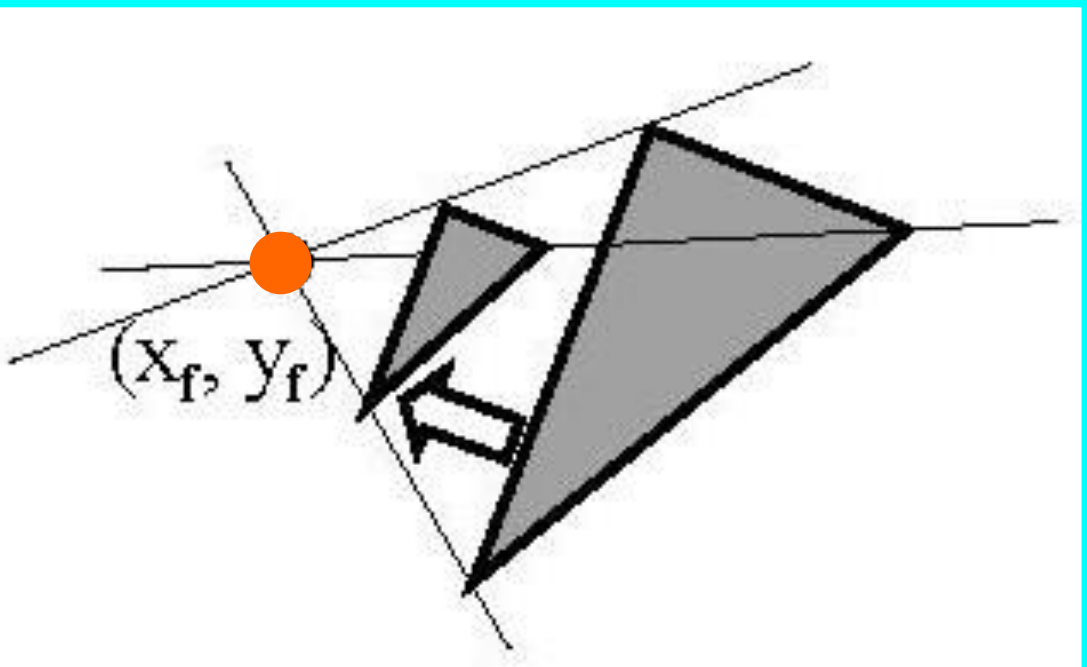
2D Scaling

- 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



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

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

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

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

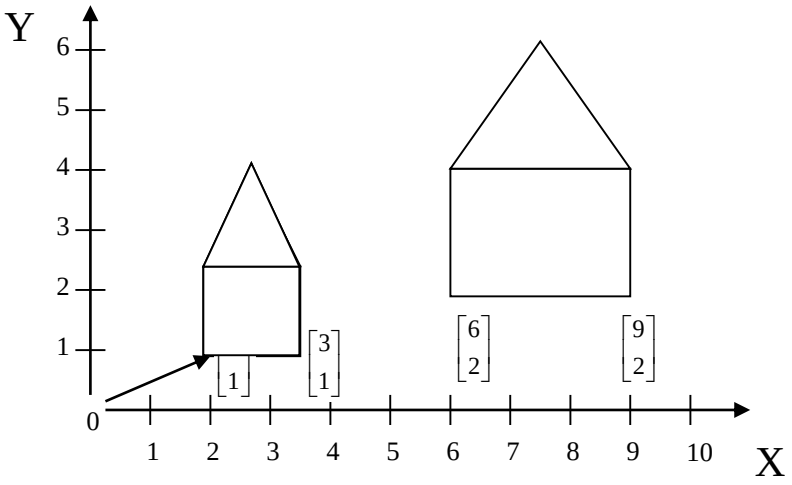
2D Scaling

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

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

$s_x = 3$

$s_y = 2$



Note: House shifts position relative to origin

Matrix Representations

And

Homogeneous Coordinates

Henry Ford Int'l College, Kalanki, Kathmandu
By: Hari Prashad Pant

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:

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

Matrix Representations

$$P' = M_1 \cdot P + M_2$$

- M_1 is a 2×2 array containing multiplicative factors.
- 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

$$P' = M_1 \cdot P + M_2$$

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 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.

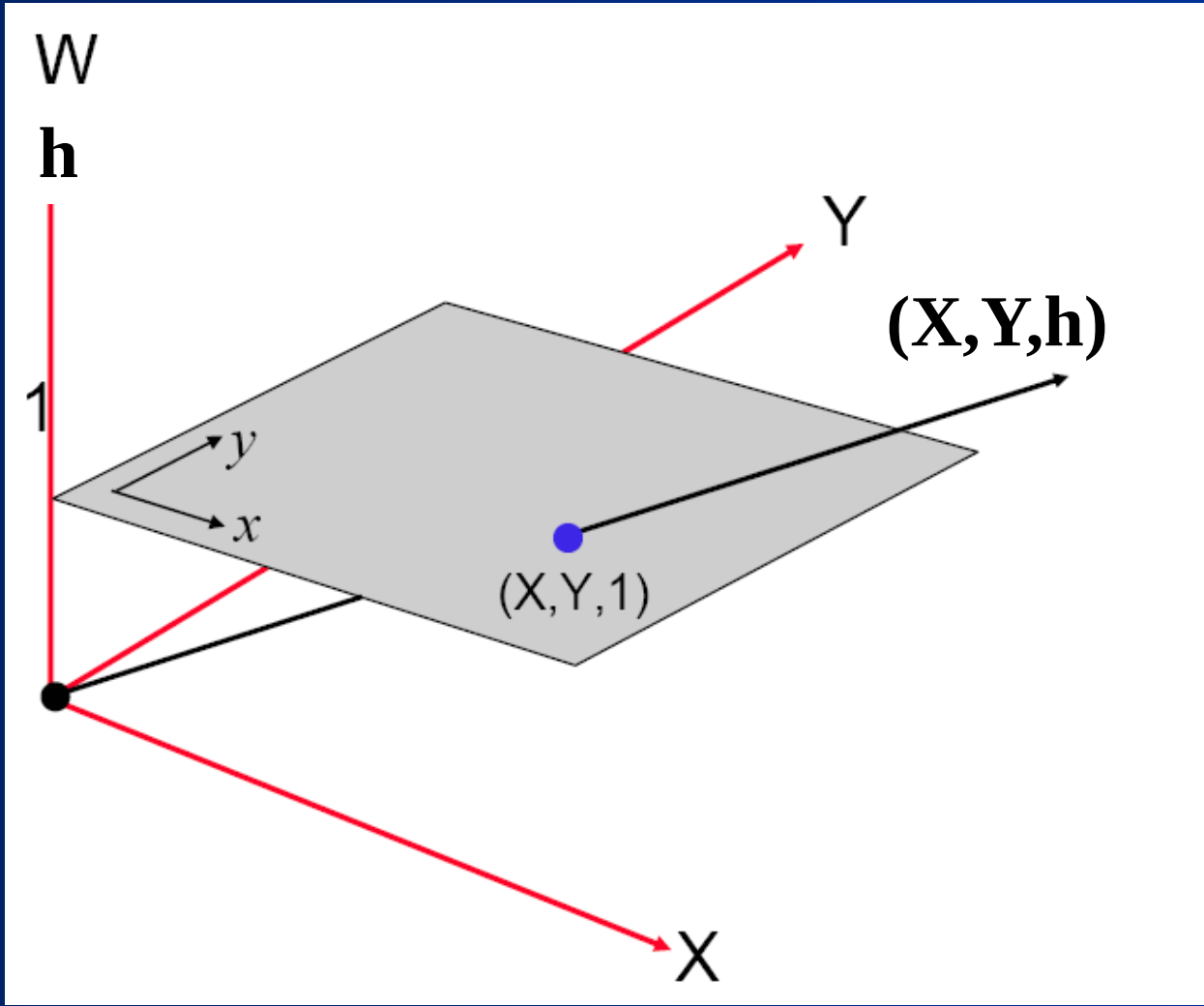
Matrix Representations and Homogeneous Coordinate

- **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):**

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

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

Henry Ford Int'l College, Kalanki, Kathmandu
By: Hari Prashad Pant

Composite Transformation

- **Combined transformations**
  - By matrix multiplication

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

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

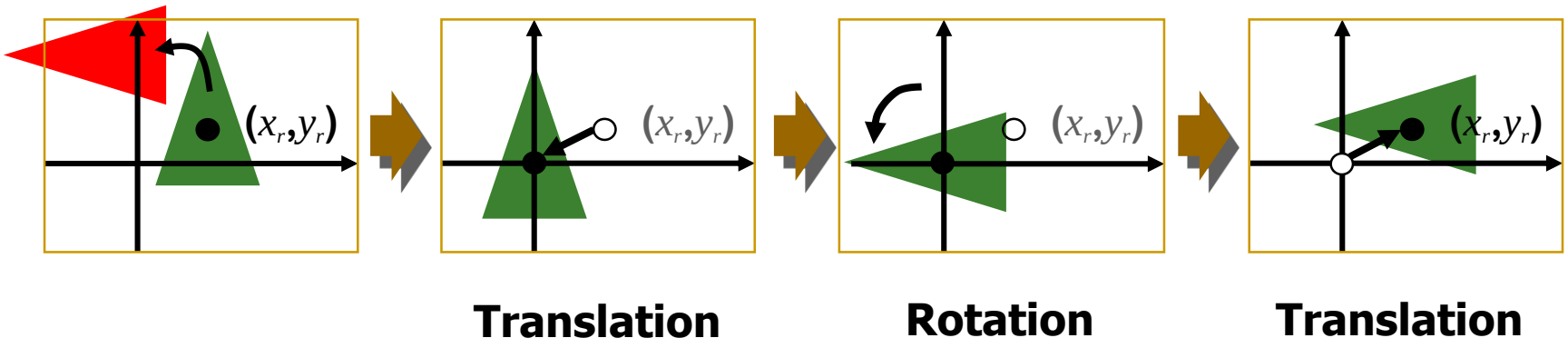
- Efficiency with pre-multiplication

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

General Pivot Point Rotation

Henry Ford Int'l College, Kalanki, Kathmandu
By: Hari Prashad Pant

General Pivot Point Rotation



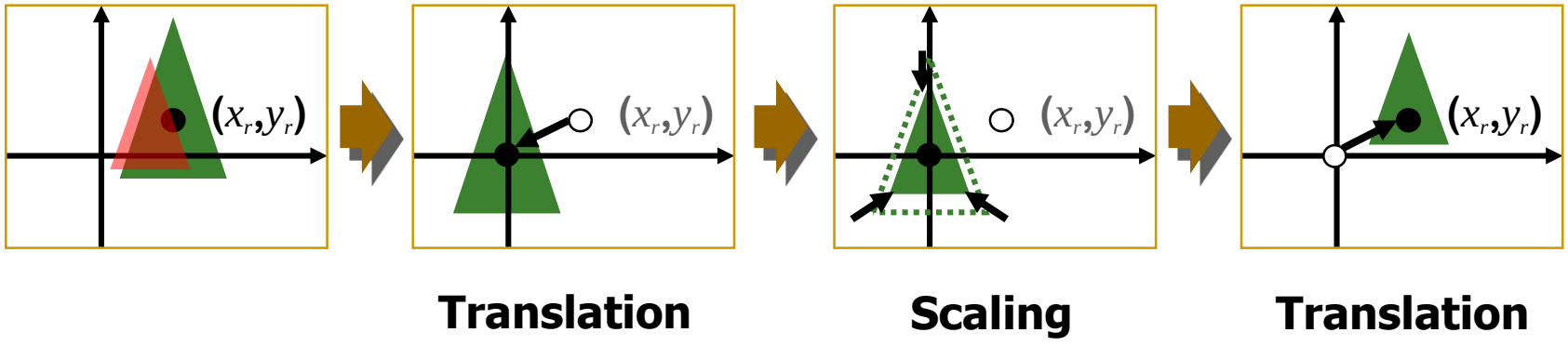
$$\mathbf{T}(x_r, y_r) \cdot \mathbf{R}(\theta) \cdot \mathbf{T}(-x_r, -y_r) = \mathbf{R}(x_r, y_r, \theta)$$

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

General Fixed Point Scaling

Henry Ford Int'l College, Kalanki, Kathmandu
By: Hari Prashad Pant

General Fixed Point Scaling



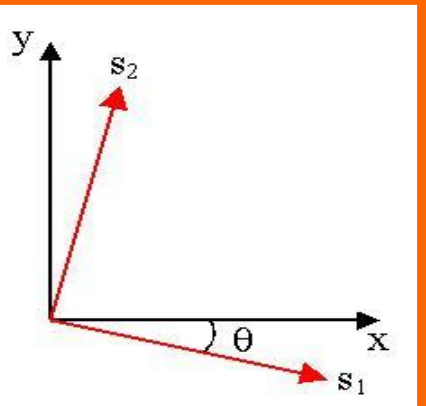
$$\mathbf{T}(x_r, y_r) \cdot \mathbf{S}(s_x, s_y) \cdot \mathbf{T}(-x_r, -y_r) = \mathbf{S}(x_r, y_r, s_x, s_y)$$

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

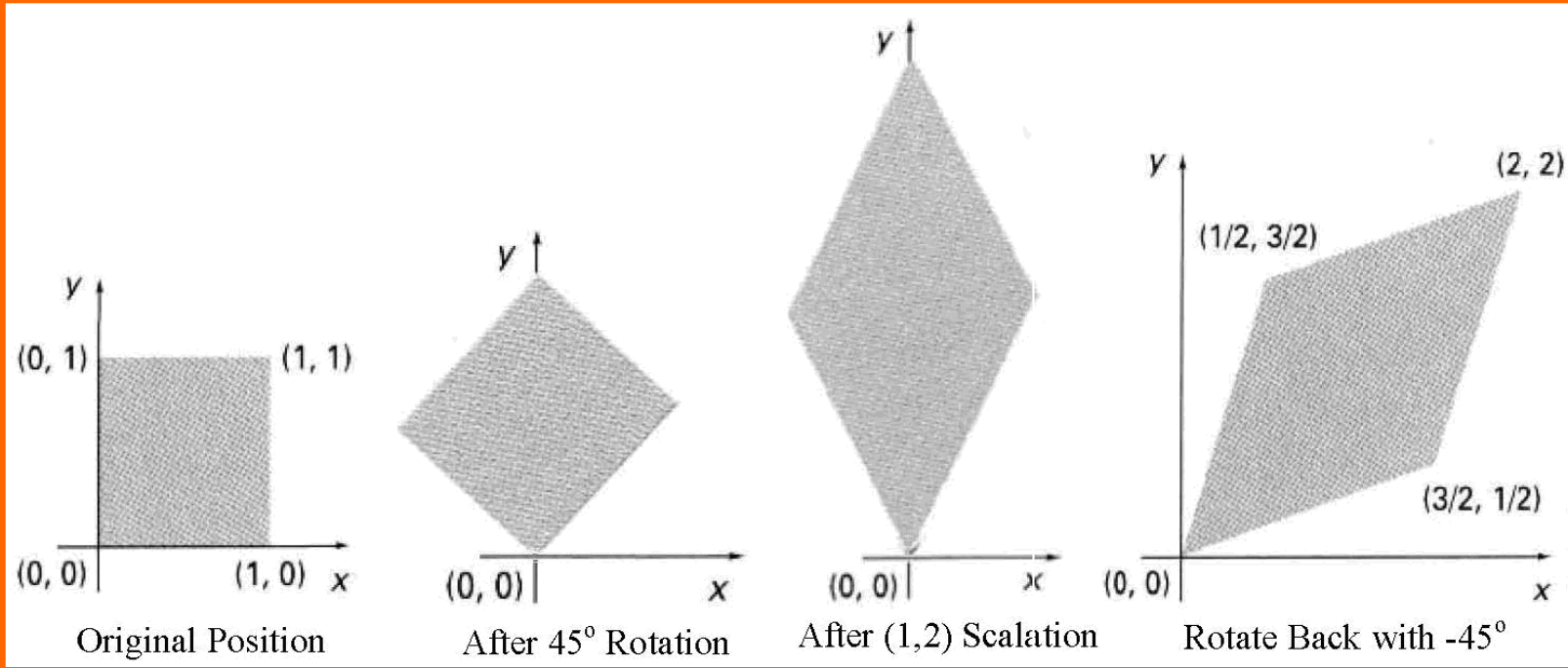
General Scaling Direction

Henry Ford Int'l College, Kalanki, Kathmandu
By: Hari Prashad Pant

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

Henry Ford Int'l College, Kalanki, Kathmandu
By: Hari Prashad Pant

Reflection

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

X Axis

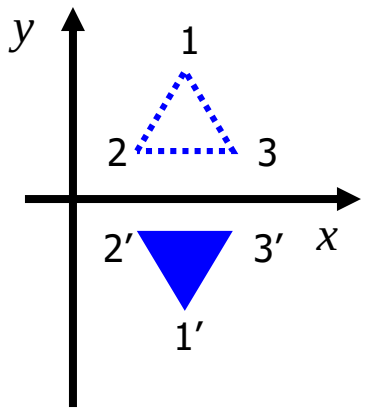
$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Y Axis

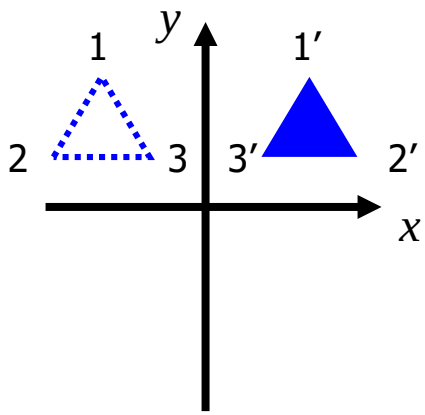
$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Origin

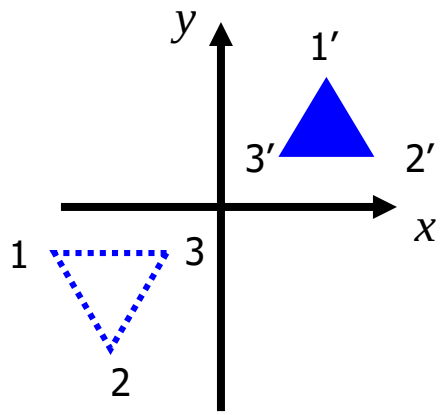
$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



X Axis



Y Axis

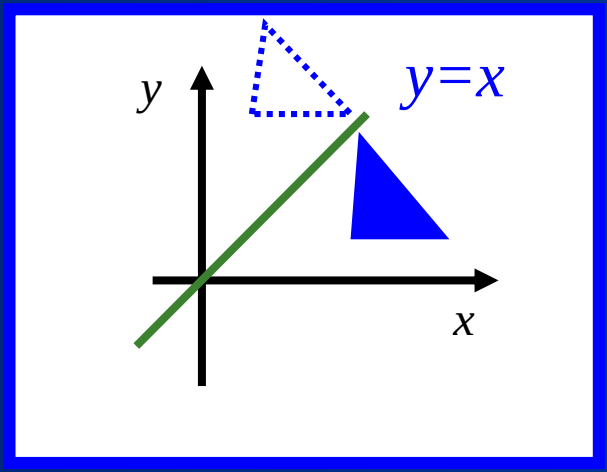


Origin

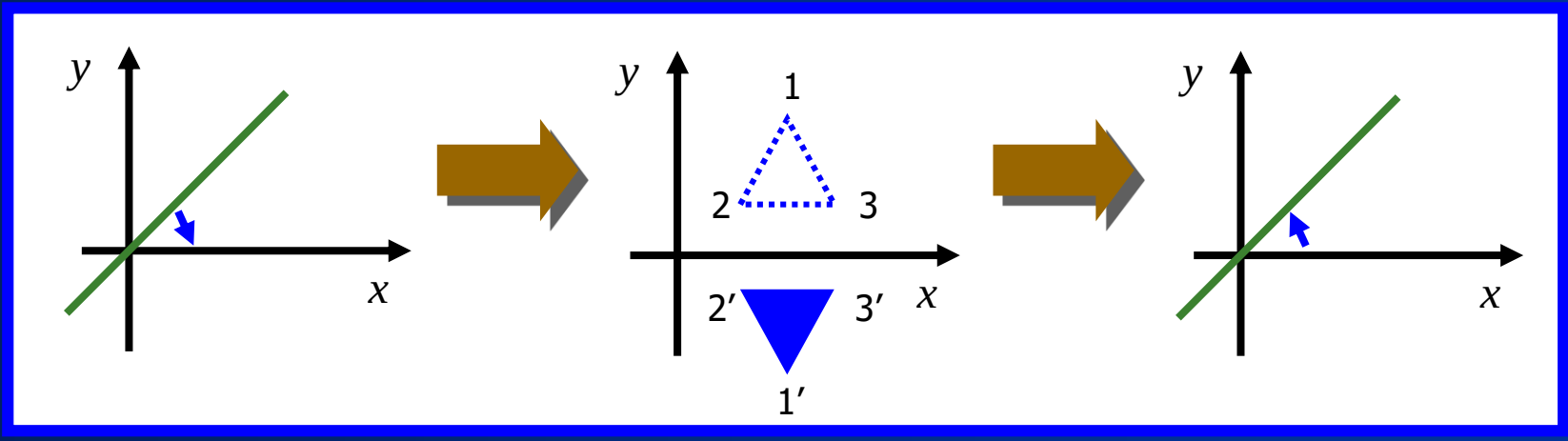
Reflection

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

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



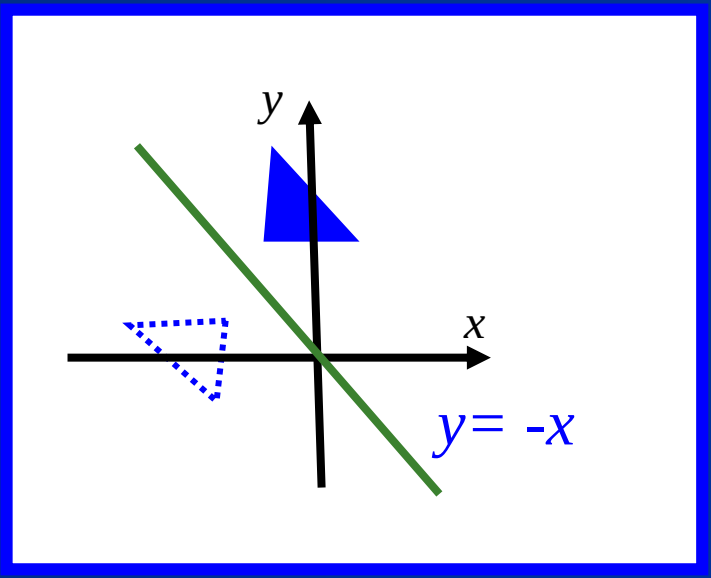
- We can drive this matrix by : Clockwise rotation of 45° → Reflection about the x axis → Counterclockwise rotation of 45°



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

Henry Ford Int'l College, Kalanki, Kathmandu
By: Hari Prashad Pant

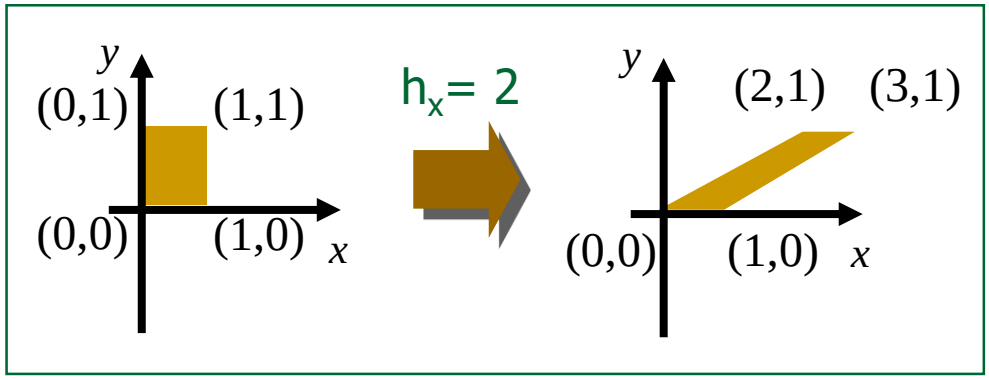
Shear

- **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

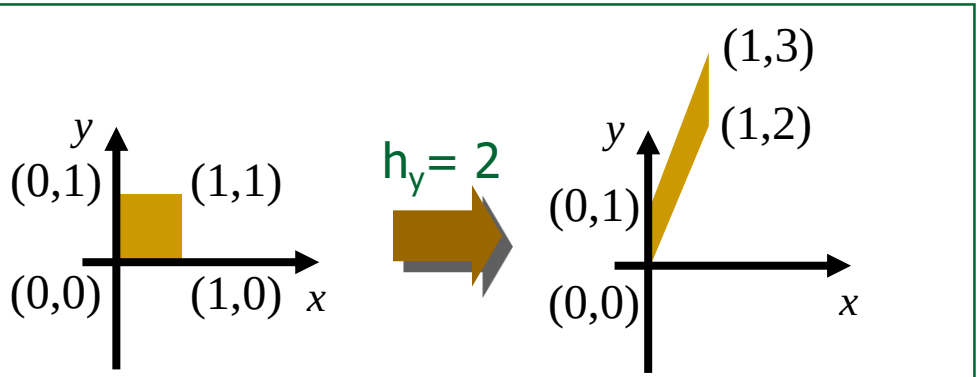
- x-direction: $x' = x + h_x \cdot y, \quad y' = y$

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



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

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

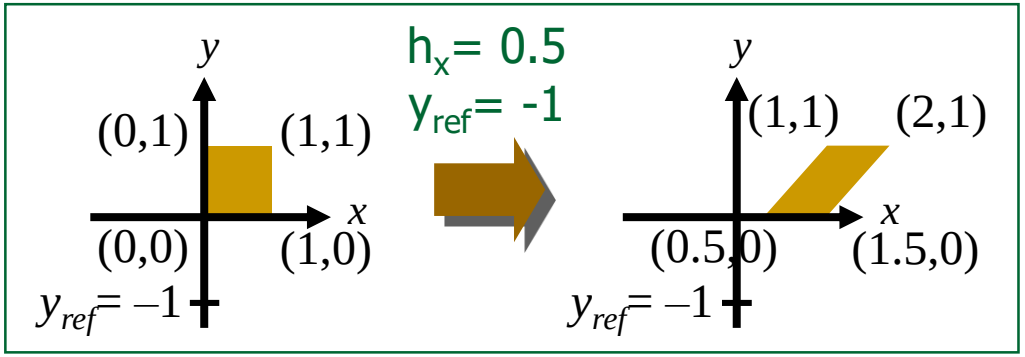


Shear

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

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

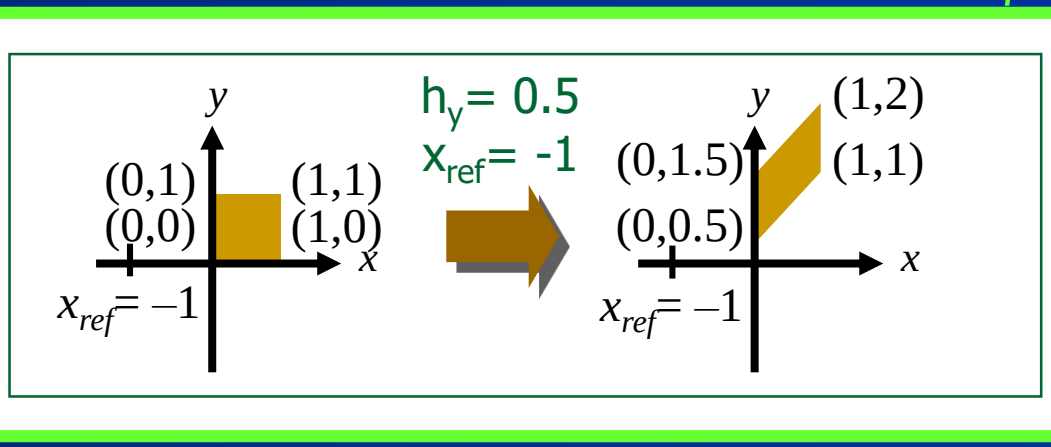
$$\begin{bmatrix} 1 & h_x & -h_x \cdot y_{ref} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



- 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

Henry Ford Int'l College, Kalanki, Kathmandu
By: Hari Prashad Pant

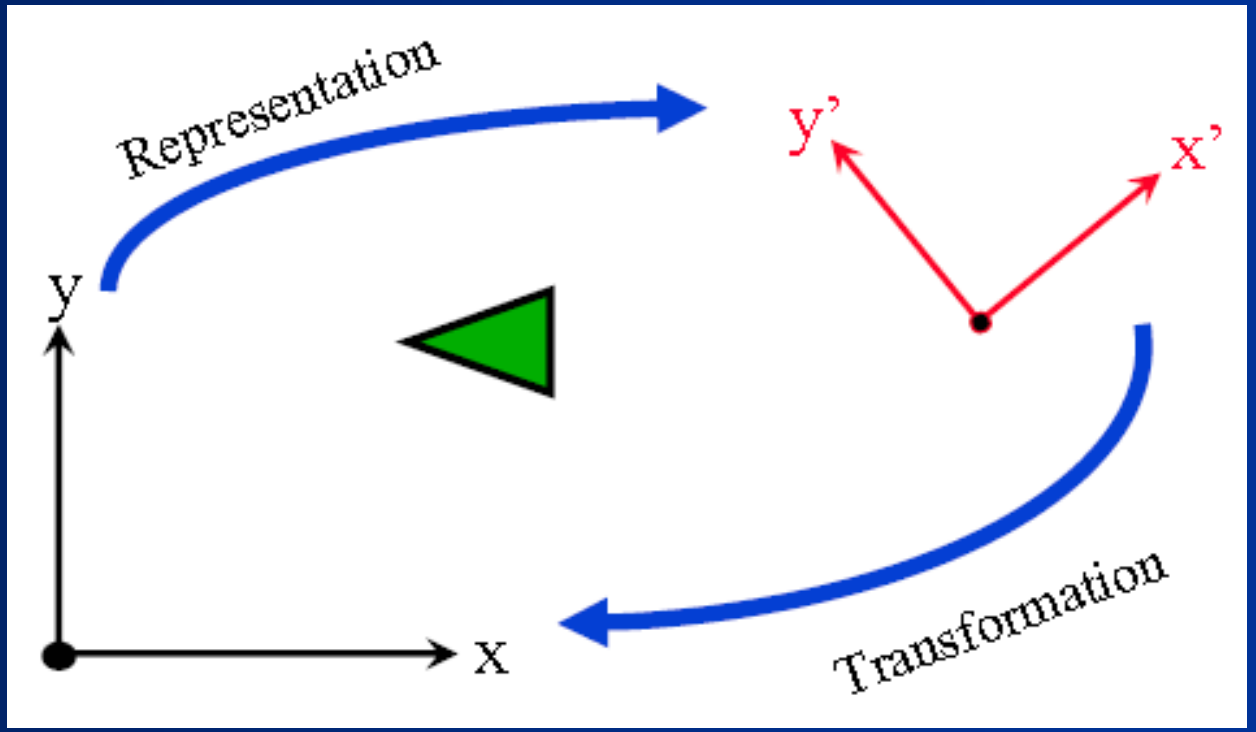
Transformations Between Coordinates Systems

- 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

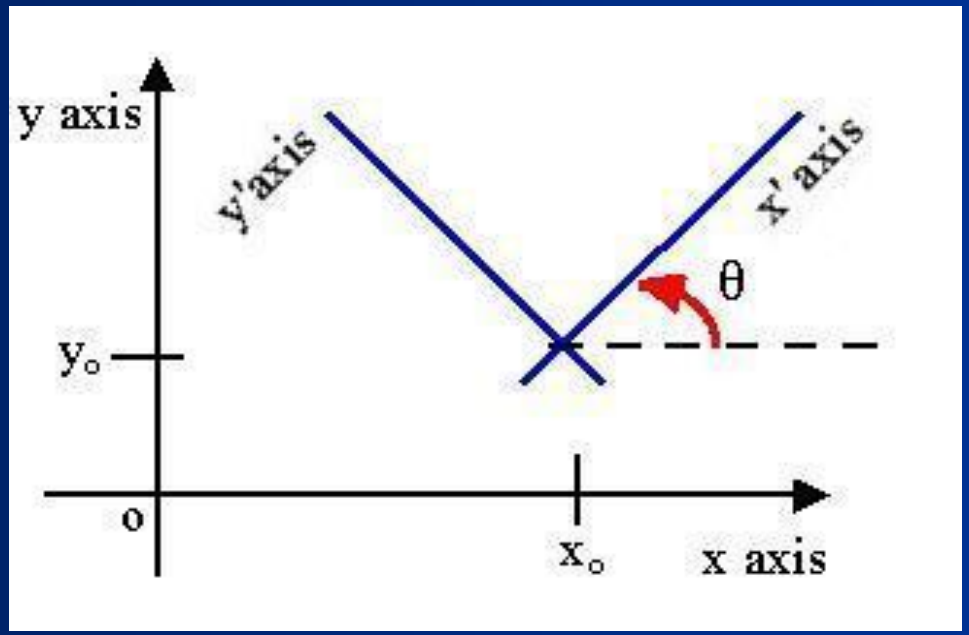
- **Rule:** Transform one coordinate frames towards the other in the opposite direction of the representation change.



Transformations Between Coordinates Systems

- **Two Steps**:

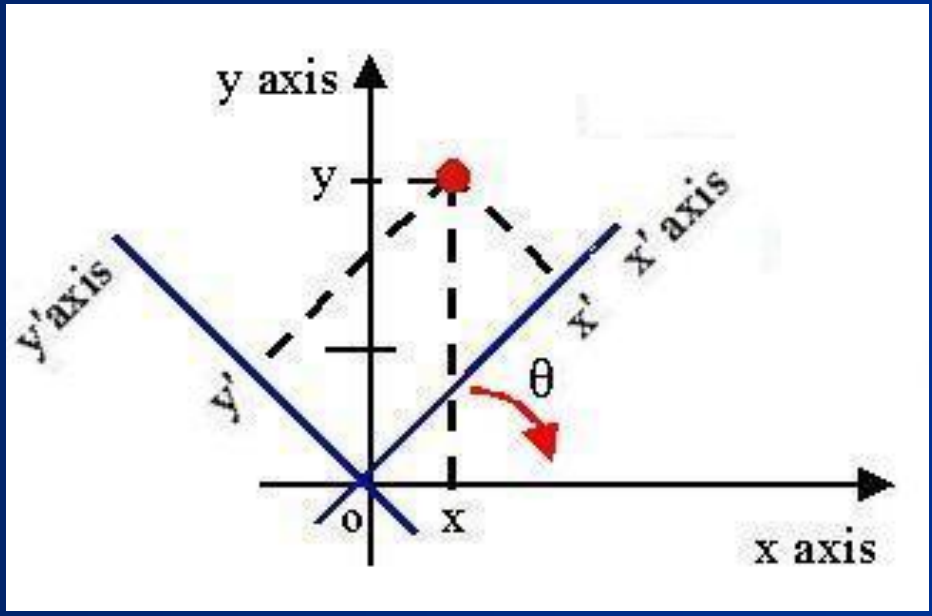
1. Translate so that the origin (x_0,y_0) of the x´y´ system is moved to the origin of the xy system.
2. Rotate the x´ axis onto the x axis.



Transformations Between Coordinates Systems

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

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



Transformations Between Coordinates Systems

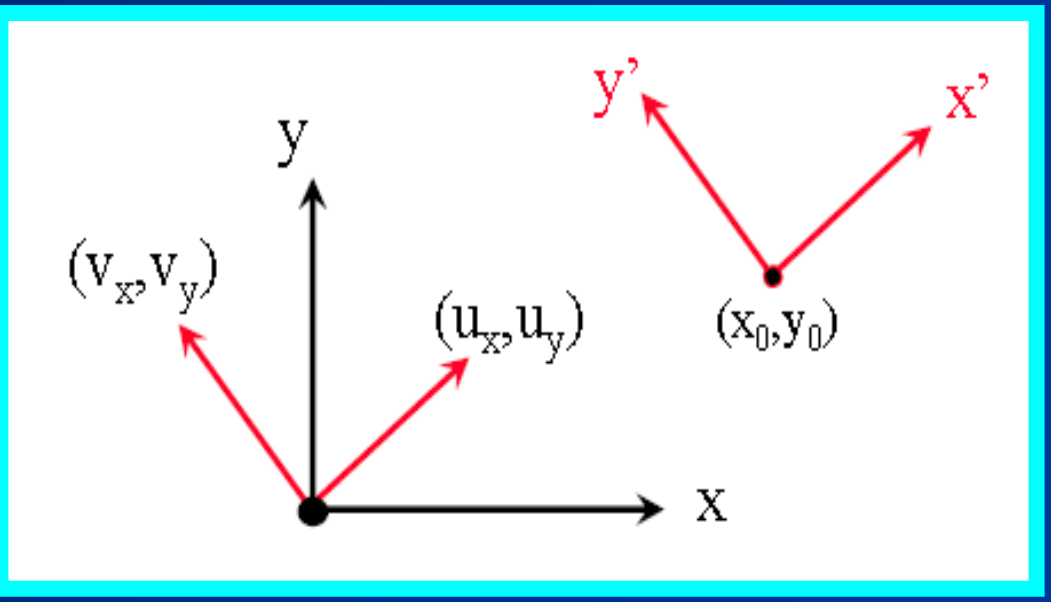
Alternative Method:

- 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:

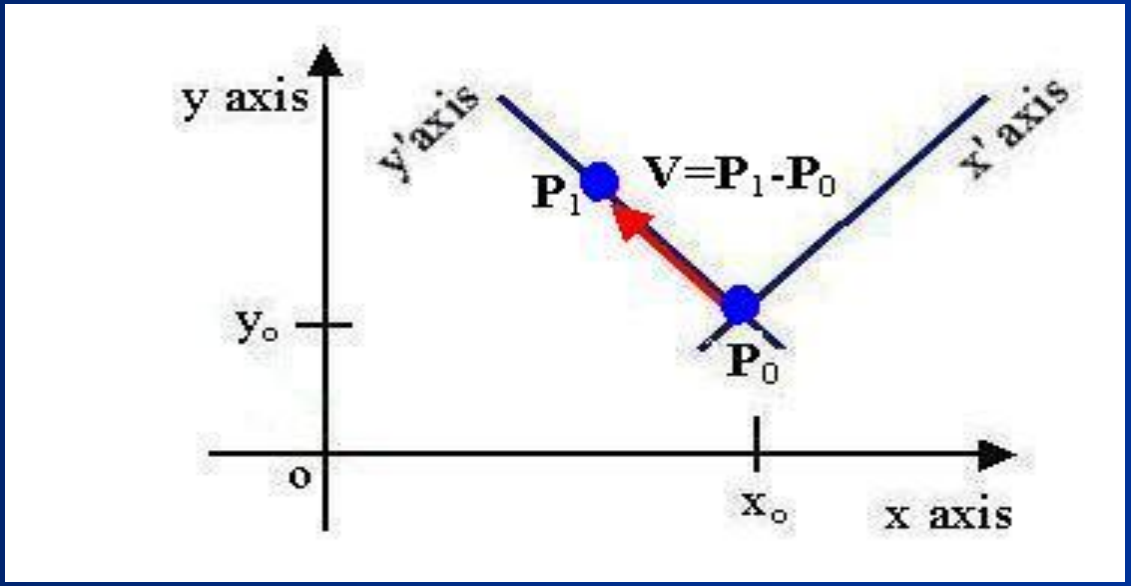
- If **V**=(-1,0) then the x′ axis is in the positive direction y and the rotation transformation matrix is:

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

Transformations Between Coordinates Systems

- **In an interactive application,** it may be more convenient to choose the direction for **V** relative to position P_0 than it is to specify it relative to the xy coordinate origin.

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



2D Viewing

Henry Ford Int'l College, Kalanki, Kathmandu
By: Hari Prashad Pant

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

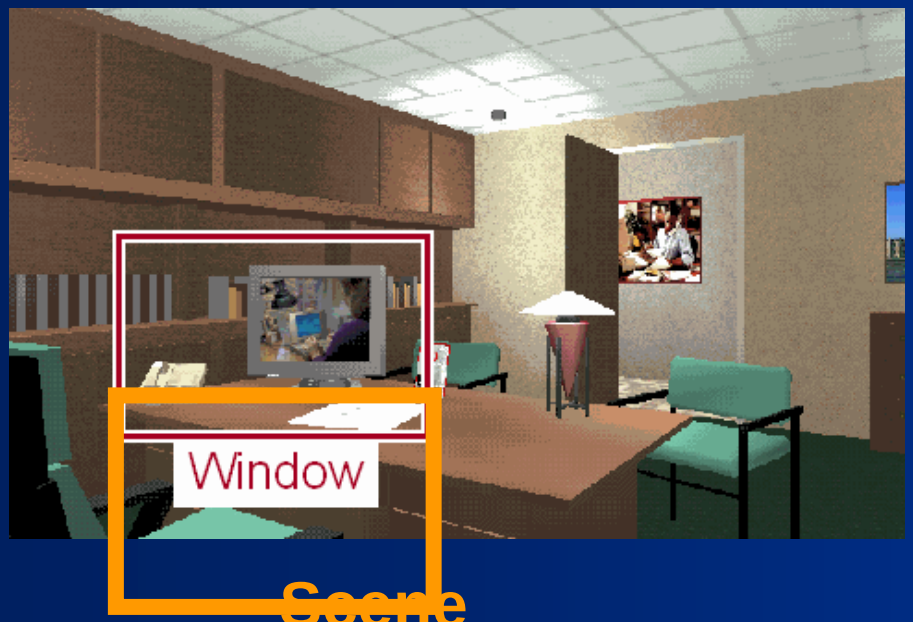
- Once a model has been constructed, the programmer can specify a view.

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



Coordinate Representations

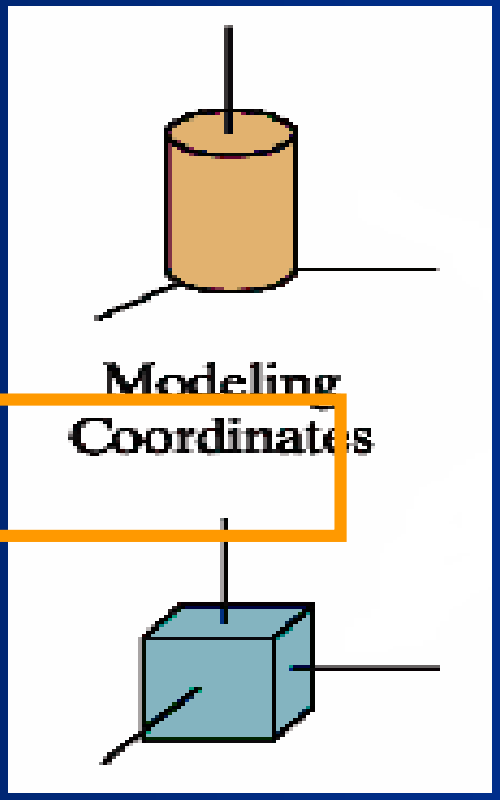
Henry Ford Int'l College, Kalanki, Kathmandu
By: Hari Prashad Pant

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.

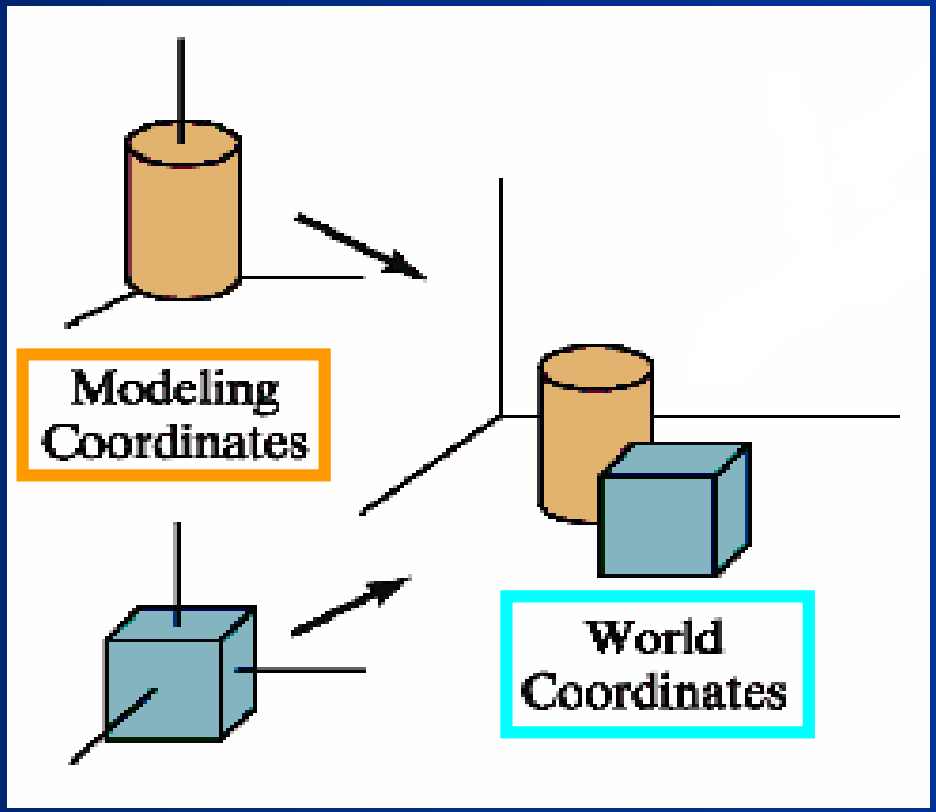
Coordinate Representations

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



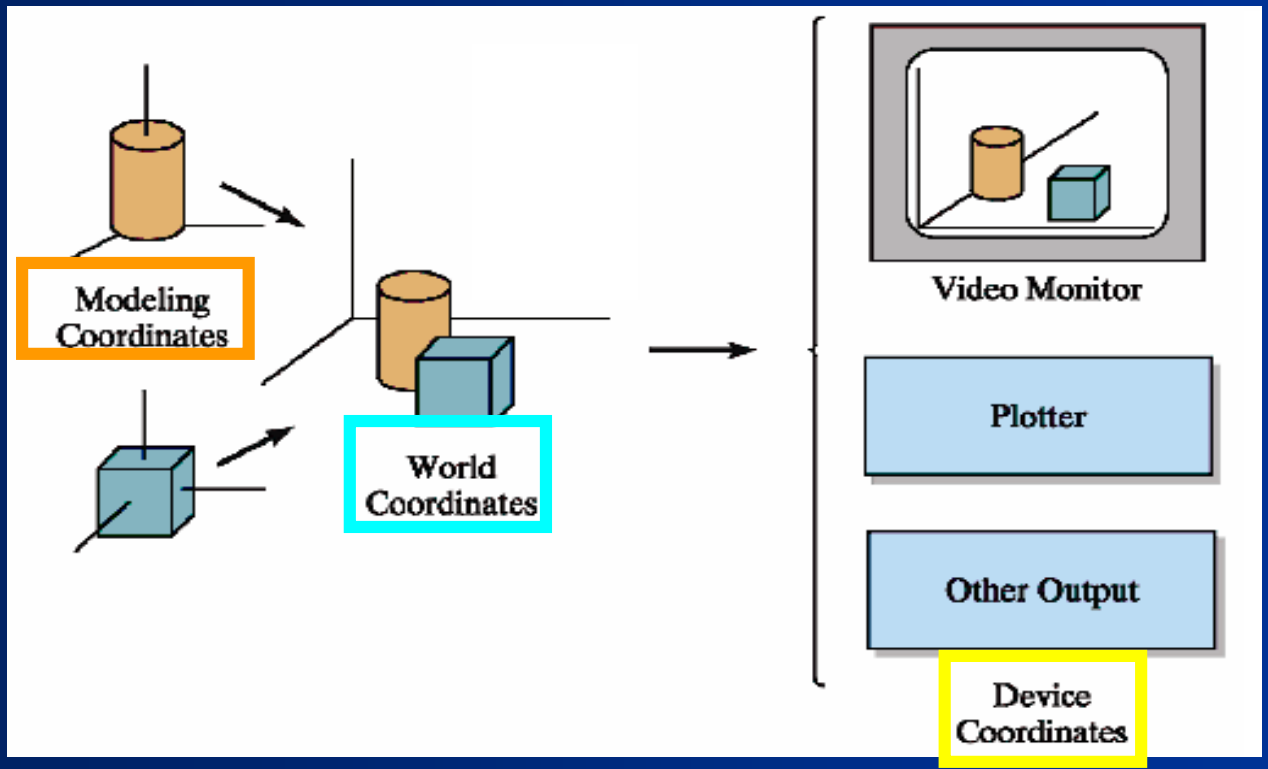
Coordinate Representations

- **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.



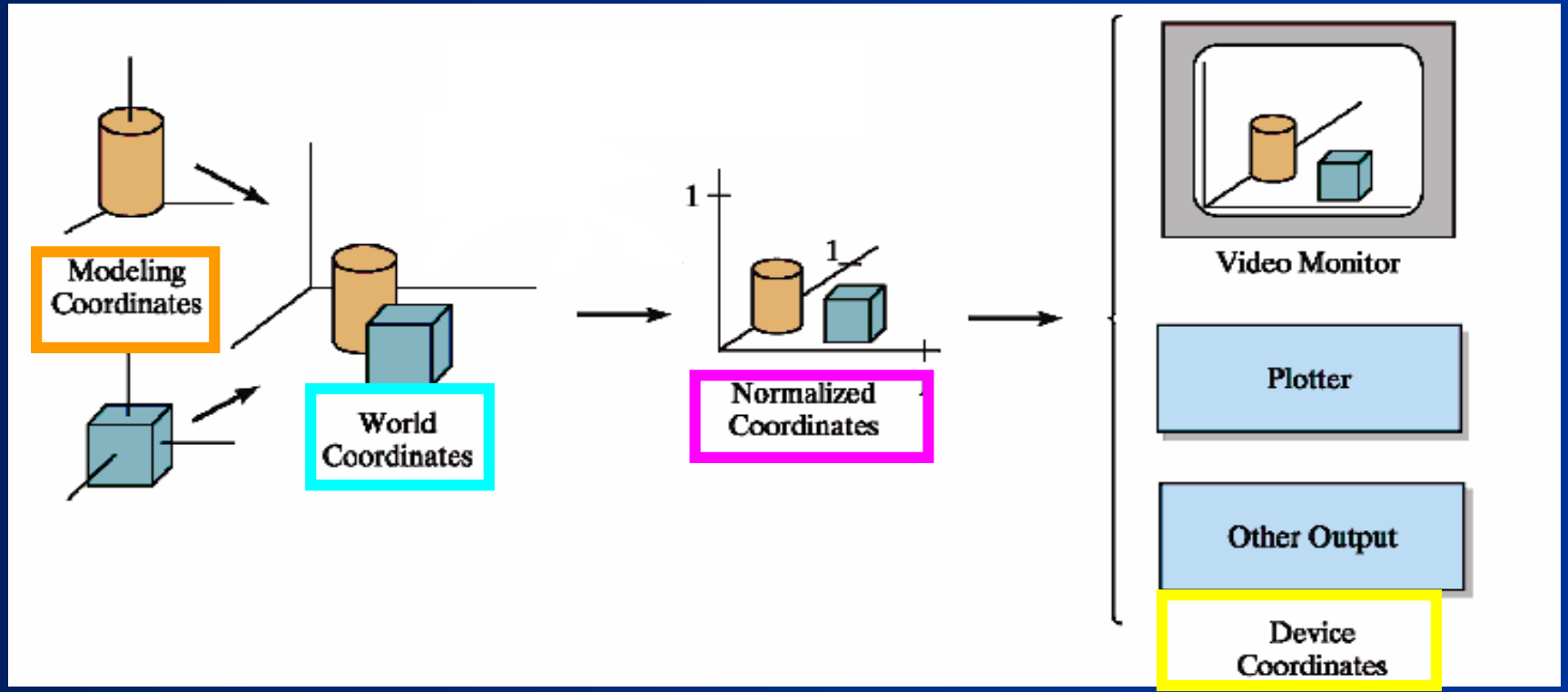
Coordinate Representations

- **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

- 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:

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

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

The Viewing Pipeline

Henry Ford Int'l College, Kalanki, Kathmandu
By: Hari Prashad Pant

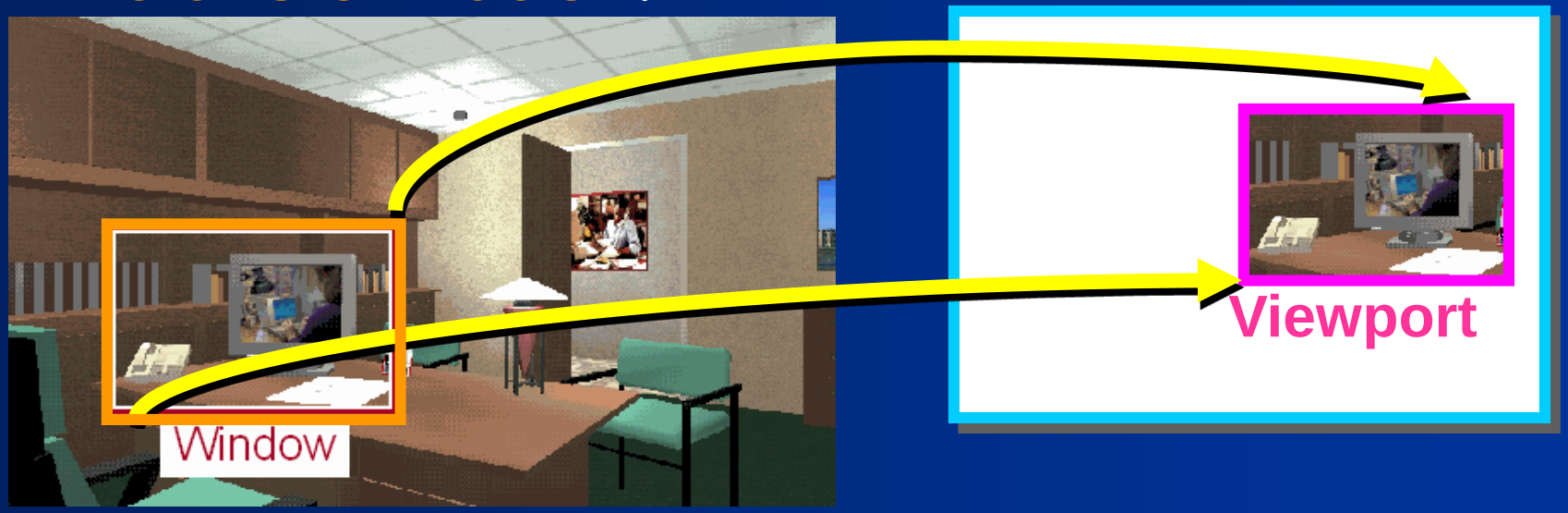
The Viewing Pipeline

- 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

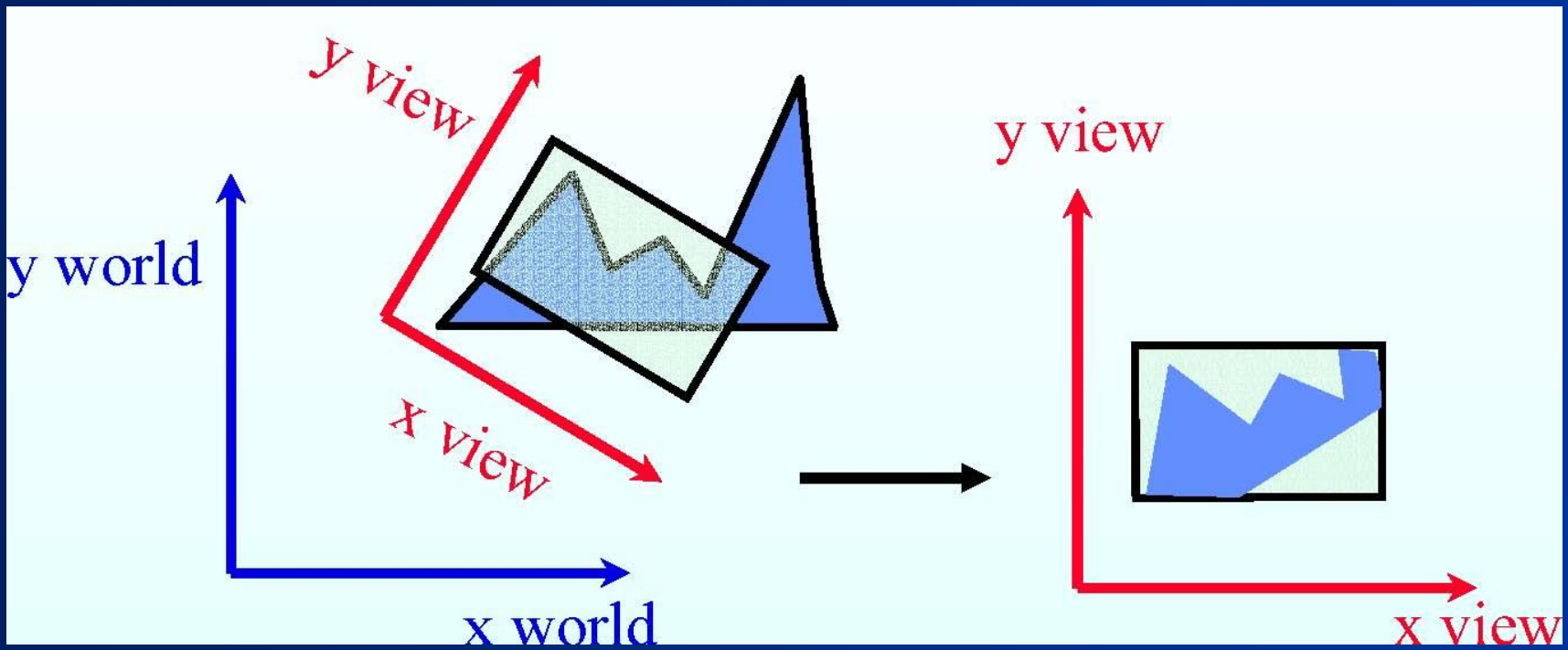
- The mapping of a part of a world coordinate scene to device coordinate is referred to as **viewing transformation** or **window-to-viewport transformation** or **windowing transformation**.



window-to-viewport transformation

The Viewing Pipeline

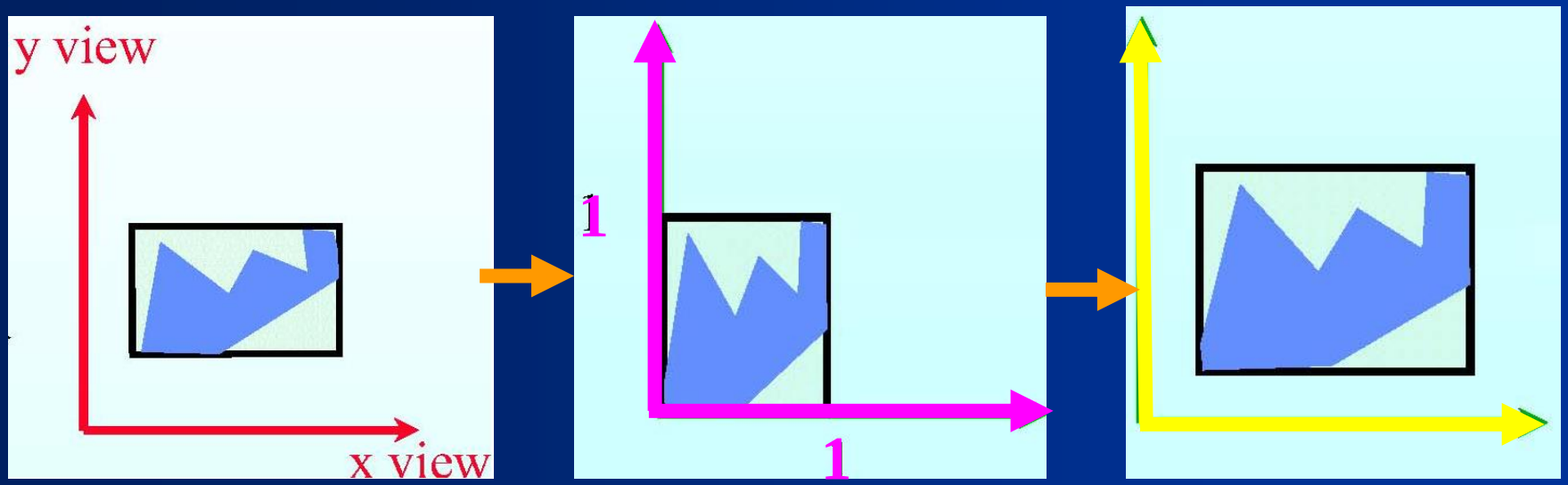
1. Construct the scene in world coordinate using the output primitives.
2. Obtain a particular orientation for the window by set up a two dimensional ***viewing coordinate*** system in the world coordinate, and define a window in the viewing coordinate system. Transform descriptions in world coordinates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



The Viewing Pipeline

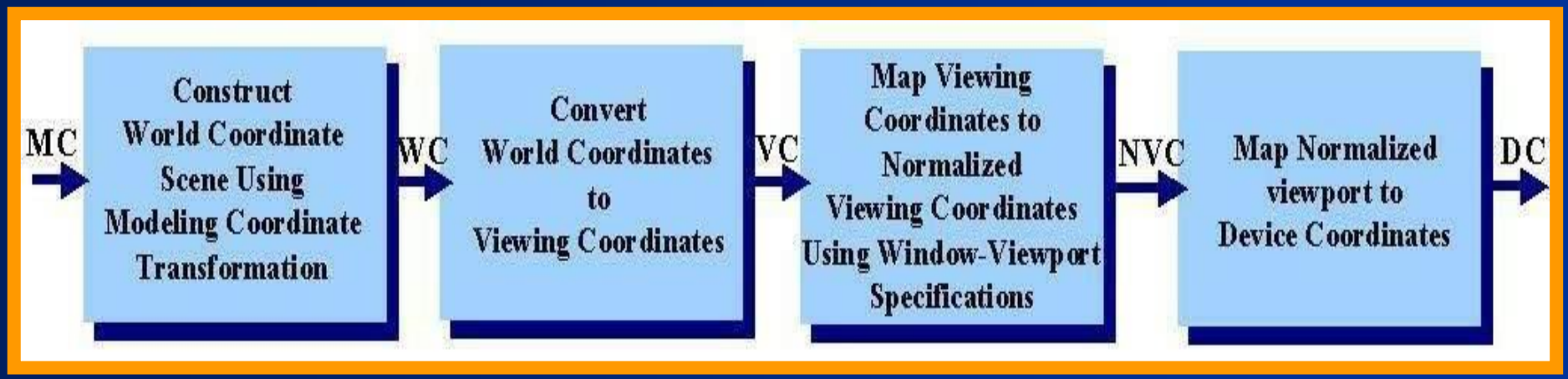
3. Define a viewport in normalized coordinate, and map the viewing coordinate description of the scene to normalized coordinate

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).

Viewing Coordinate Reference Frame

Henry Ford Int'l College, Kalanki, Kathmandu
By: Hari Prashad Pant

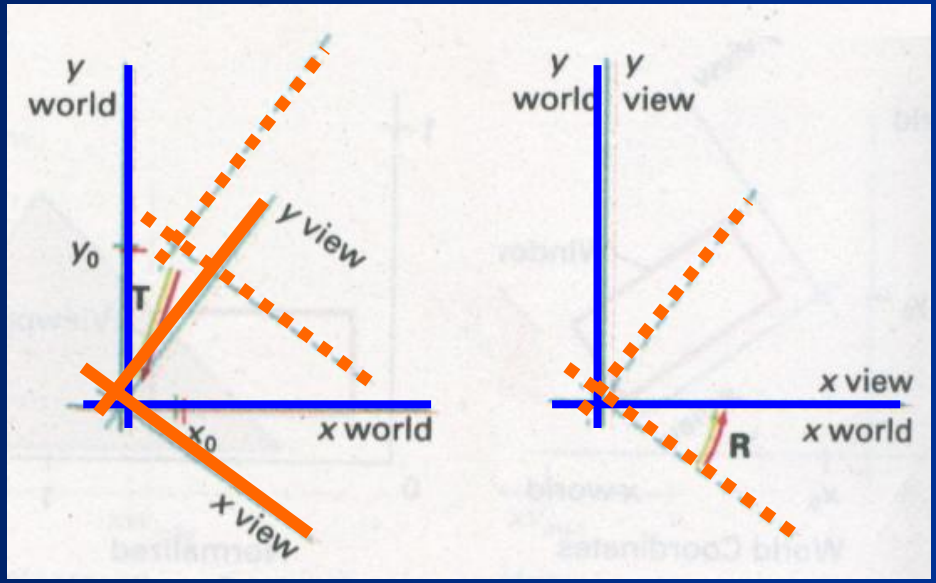
Viewing Coordinate Reference Frame

- 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

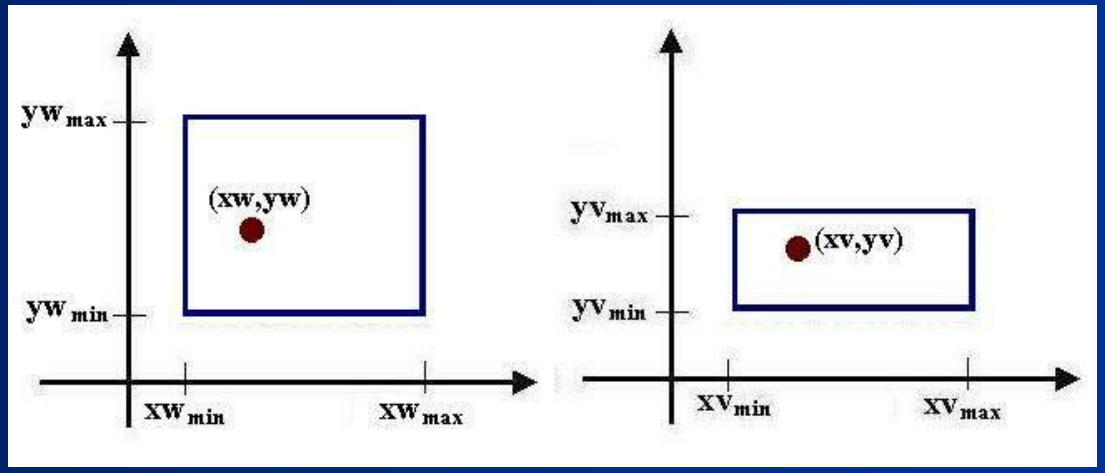
Henry Ford Int'l College, Kalanki, Kathmandu
By: Hari Prashad Pant

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

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

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

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

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

$$xv = xv_{min} + (xw - xw_{min})s_x$$

$$yv = yv_{min} + (yw - yw_{min})s_y$$

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

Henry Ford Int'l College, Kalanki, Kathmandu
By: Hari Prashad Pant

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***.

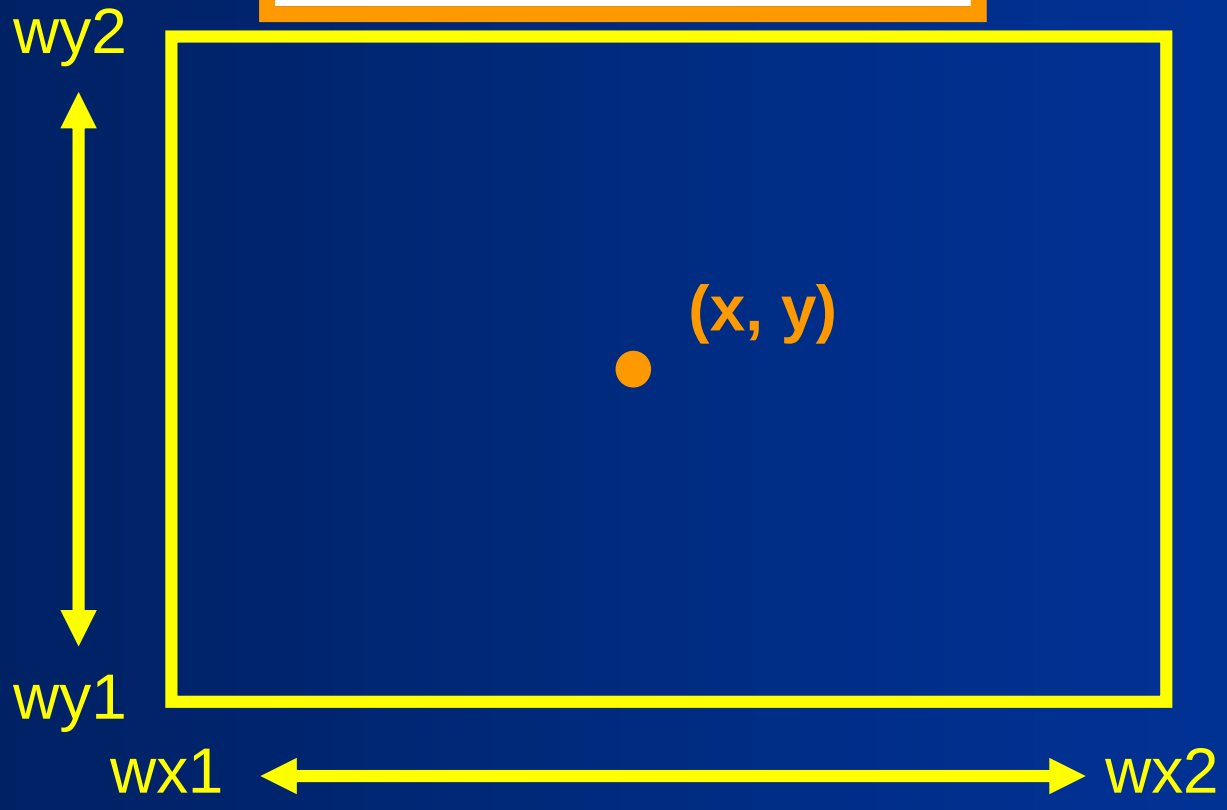
Point Clipping

Henry Ford Int'l College, Kalanki, Kathmandu
By: Hari Prashad Pant

Point Clipping

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

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

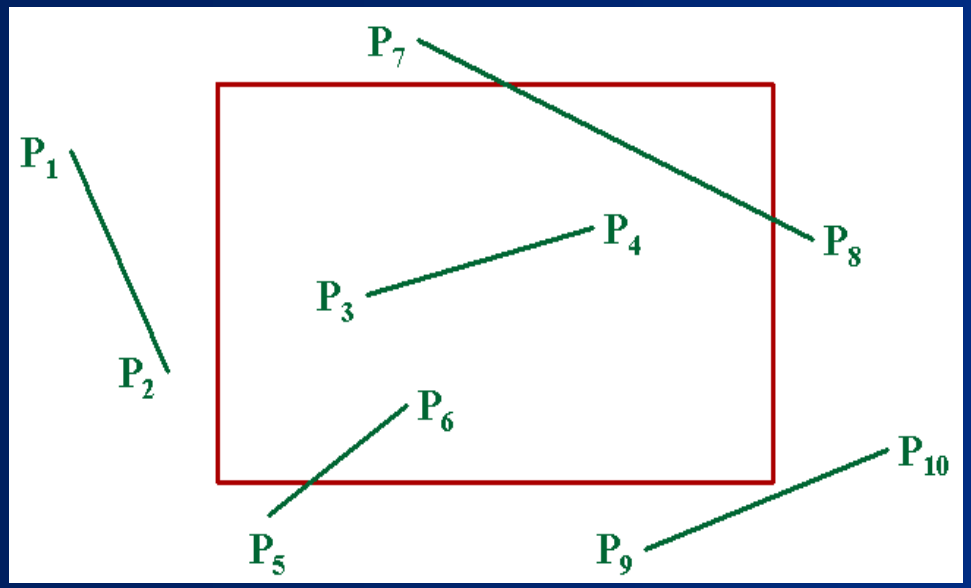


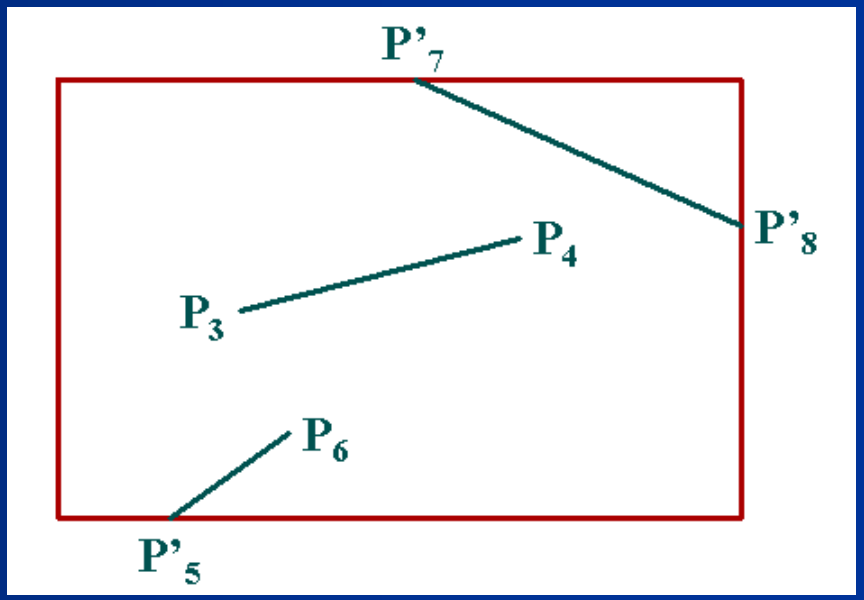
Line Clipping

Henry Ford Int'l College, Kalanki, Kathmandu
By: Hari Prashad Pant

Line Clipping

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





Before Clipping **After Clipping**

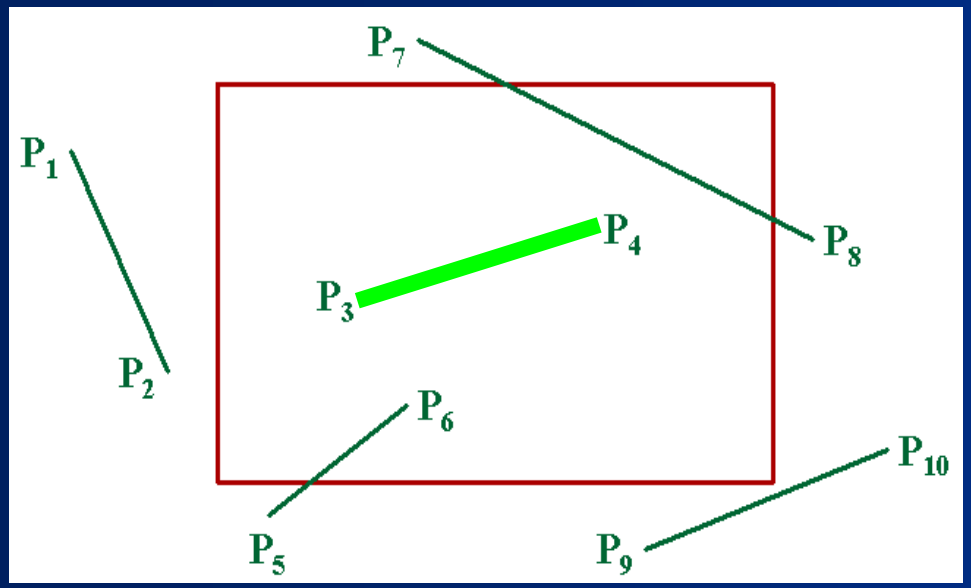
Line Clipping

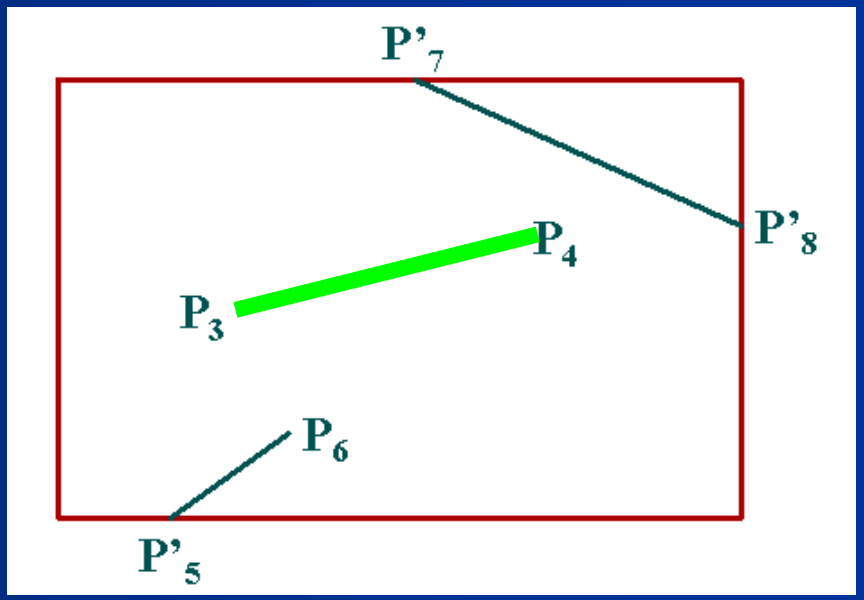
- **A line clipping procedure involves several parts:**

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** ($\overline{P_3 P_4}$)

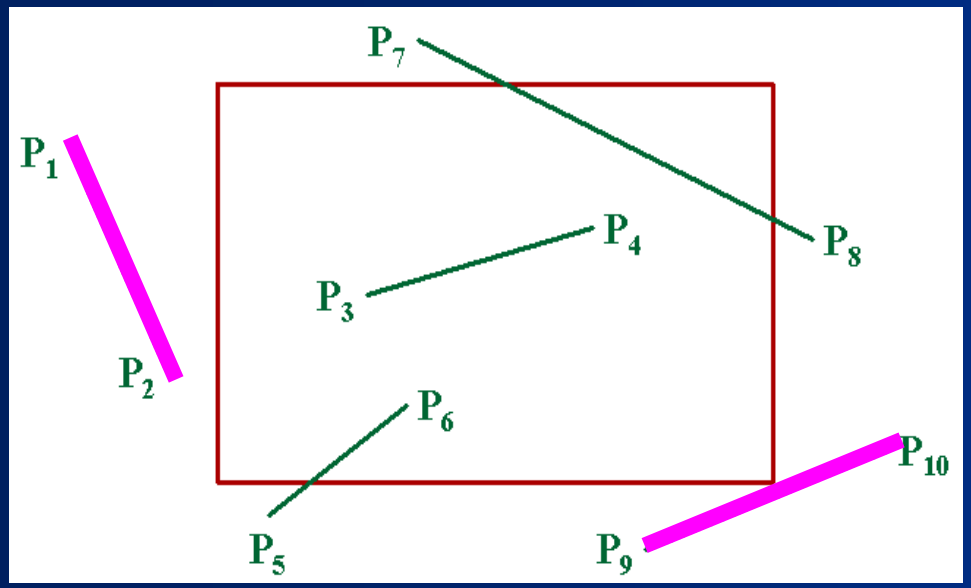


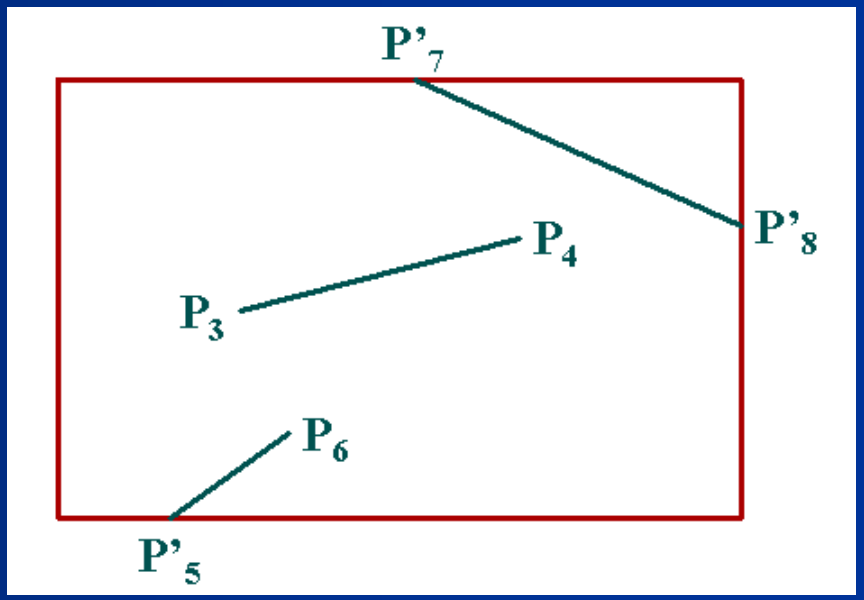


Before Clipping **After Clipping**

Line Clipping

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

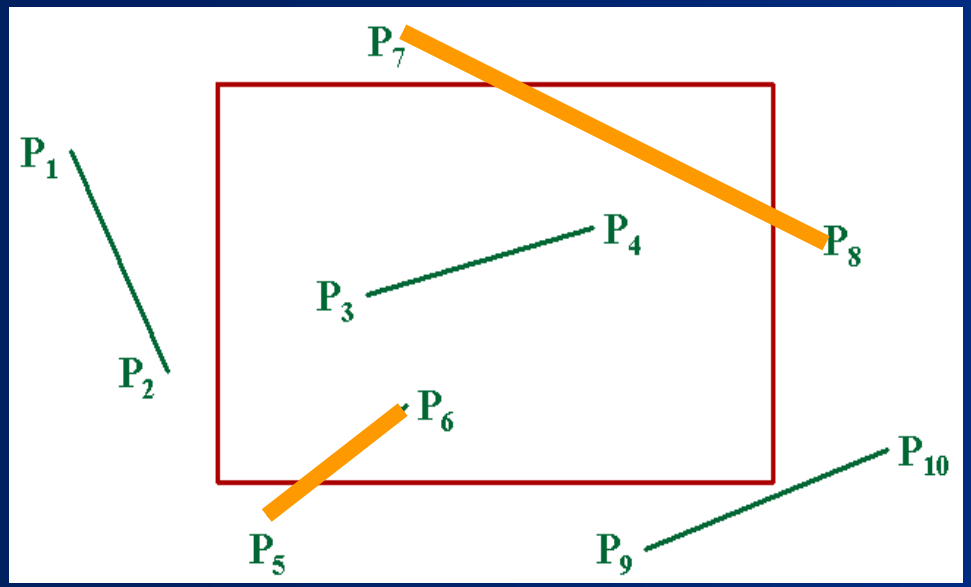


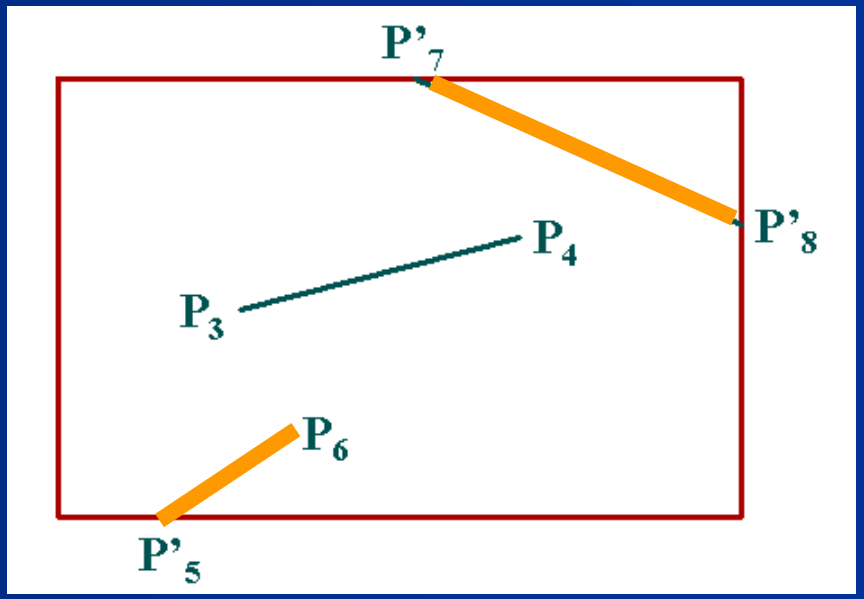


Before Clipping **After Clipping**

- 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.

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





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

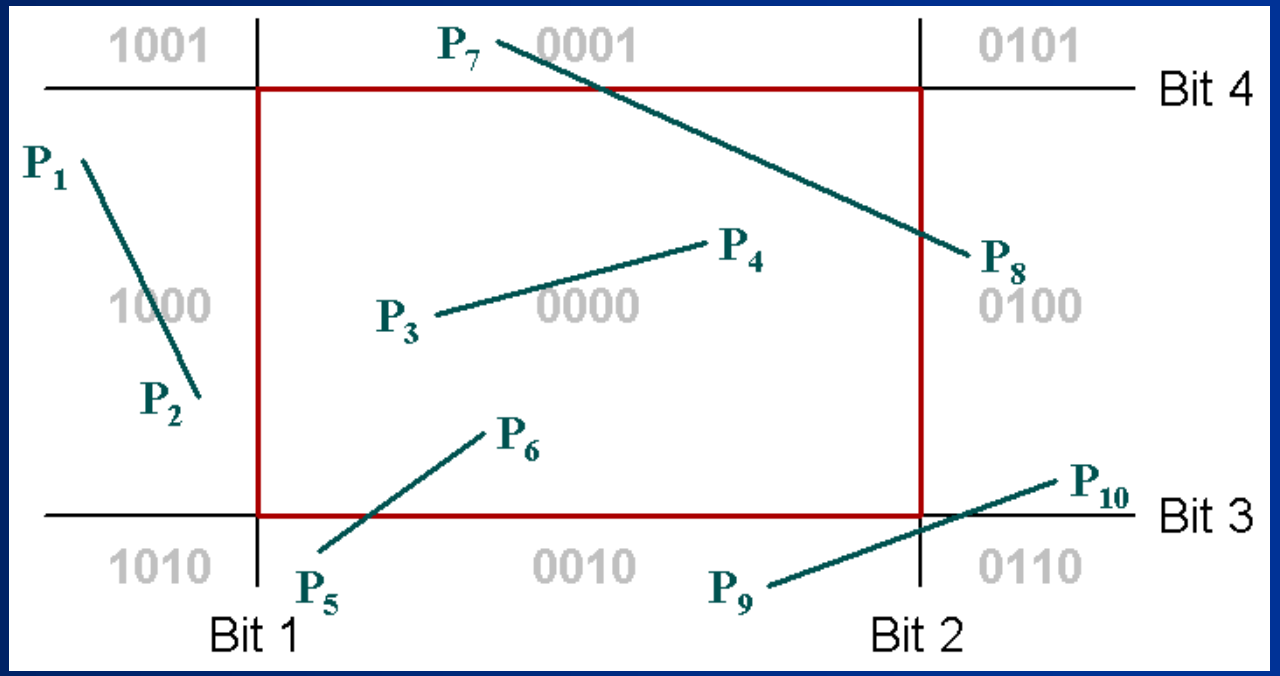
Henry Ford Int'l College, Kalanki, Kathmandu
By: Hari Prashad Pant

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.



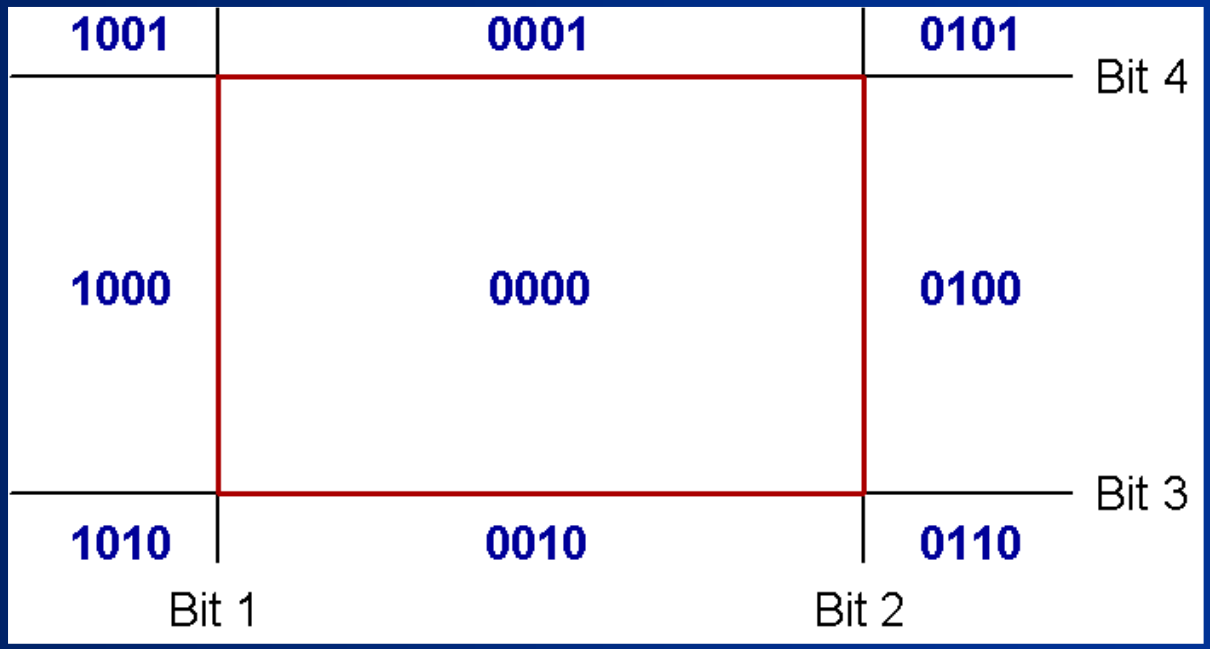
- Each bit position in the region code is used to indicate one of the four relative coordinate positions of the point with respect to the clip window.

Bit 1: Left

Bit 2: Right

Bit 3: Below

Bit 4: Above



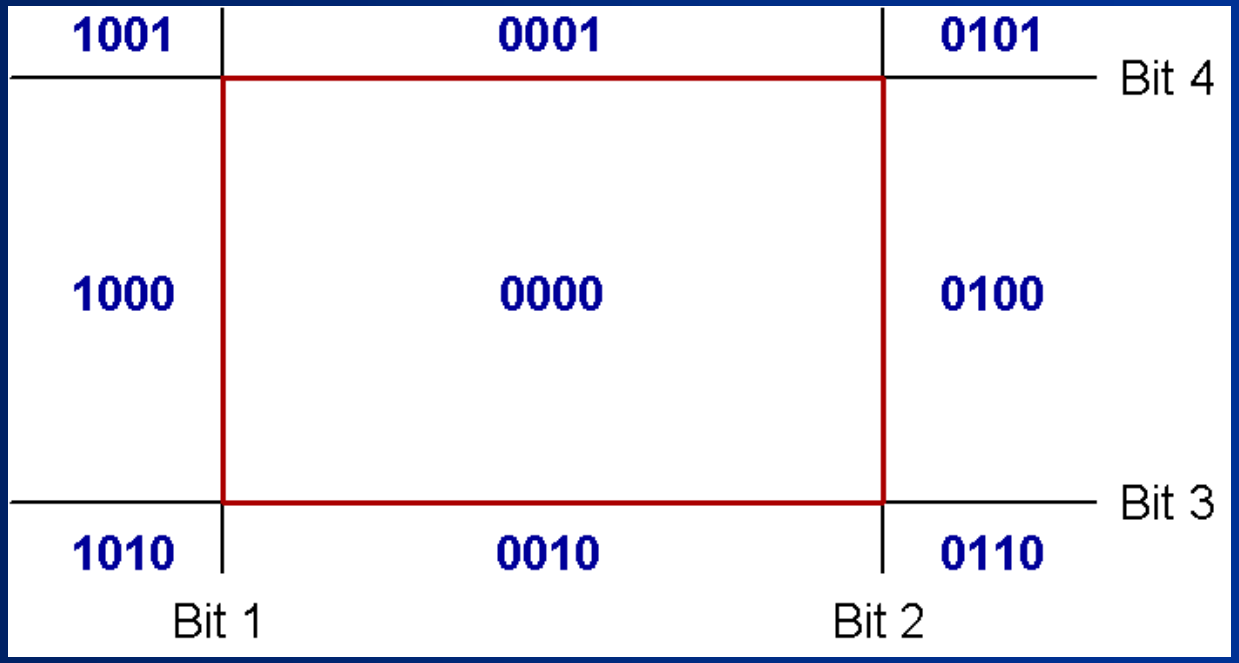
- Bit values in the region code are determined by comparing endpoint coordinates values (x,y) to the clip boundaries. Bit 1 is set to 1 if $x < xw_{min}$

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

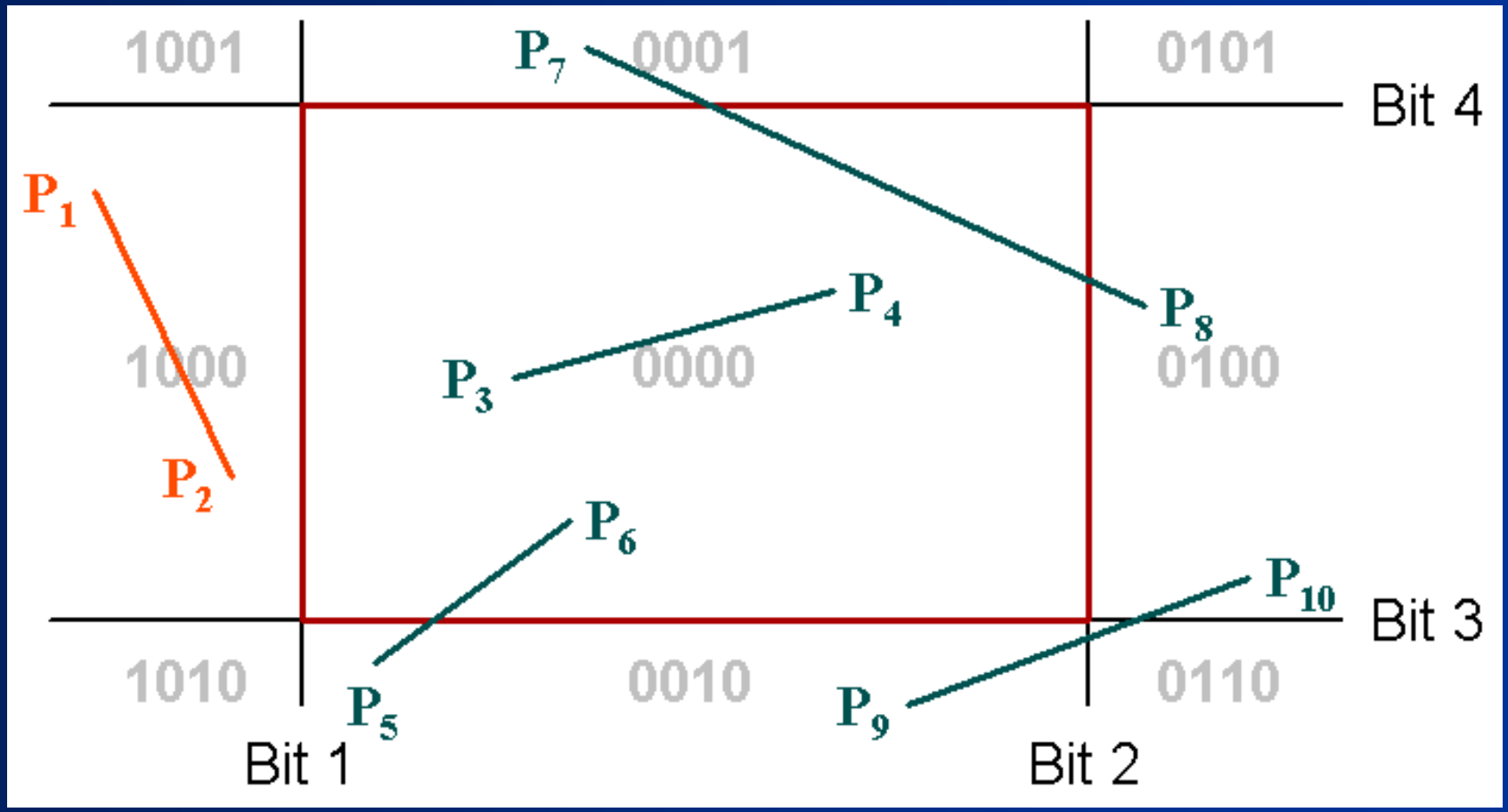
Bit 2: $sign(x - xw_{max})$

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

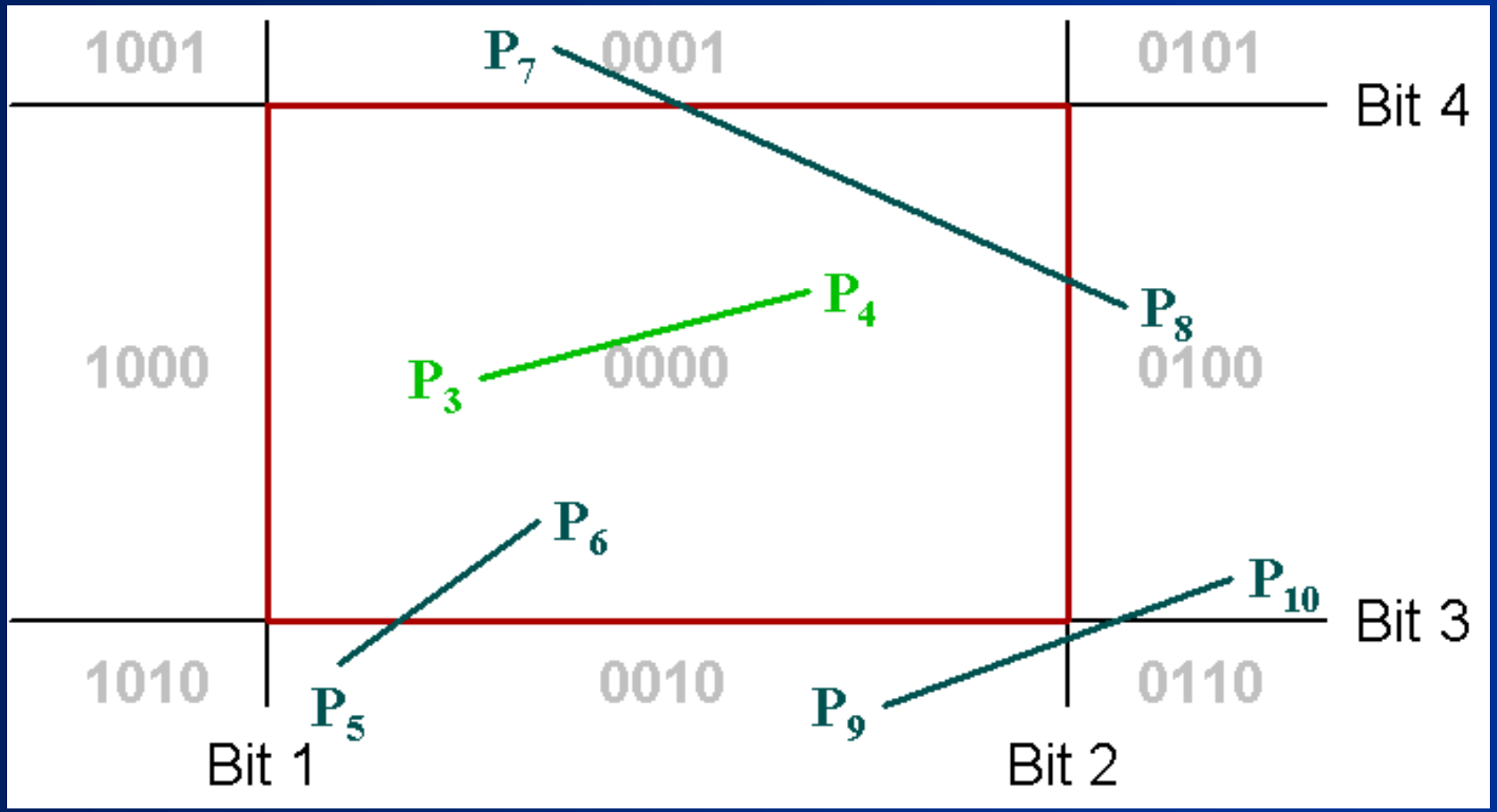
Bit 4: $sign(y - yw_{max})$



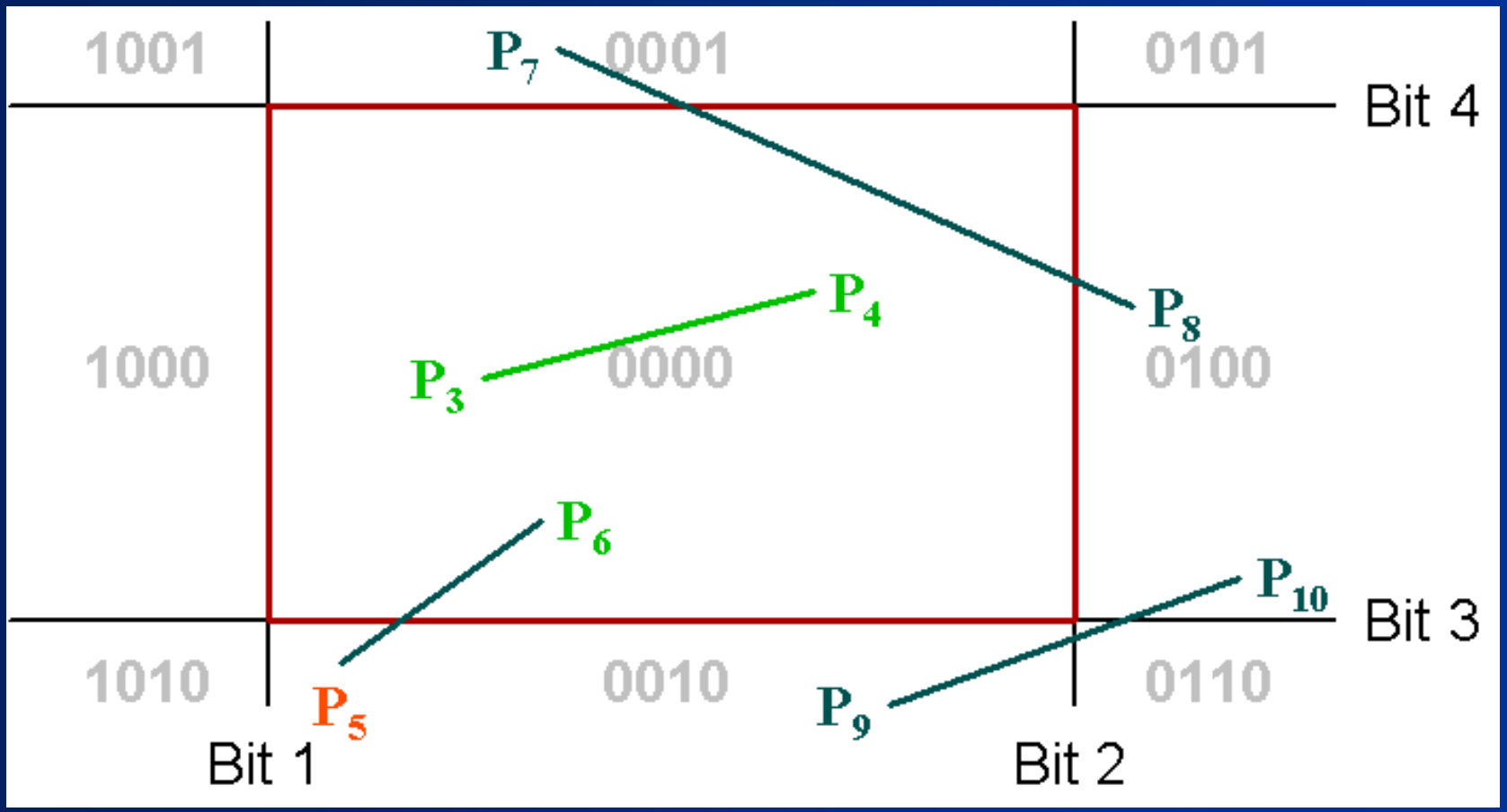
- Once we have established region codes for all line endpoints, we can quickly determine lines are completely outside or inside the clip window.



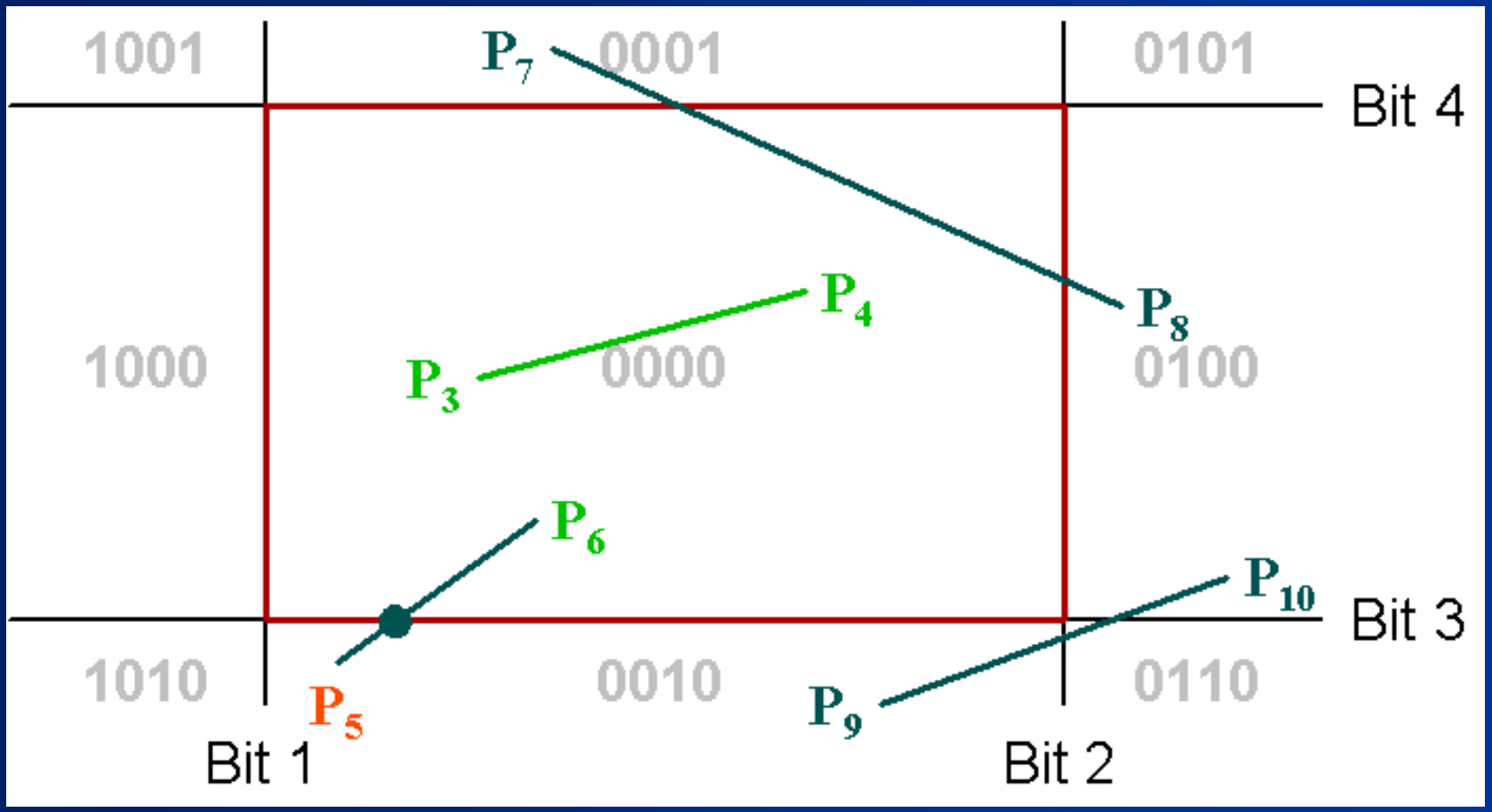
- Once we have established region codes for all line endpoints, we can quickly determine lines are completely outside or inside the clip window.



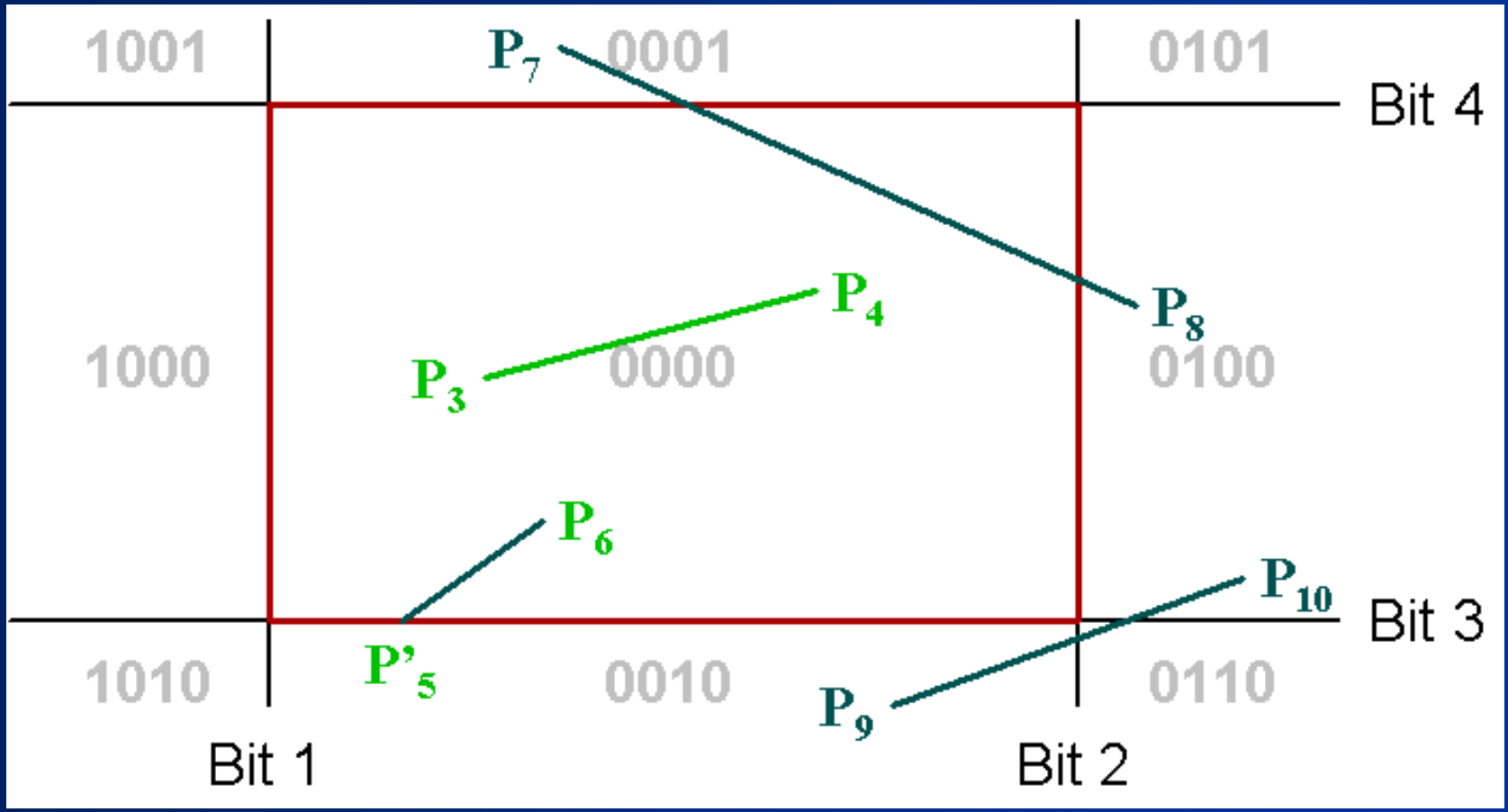
- Lines that cannot be identified as completely inside or outside a clip window are checked for intersection with boundaries.



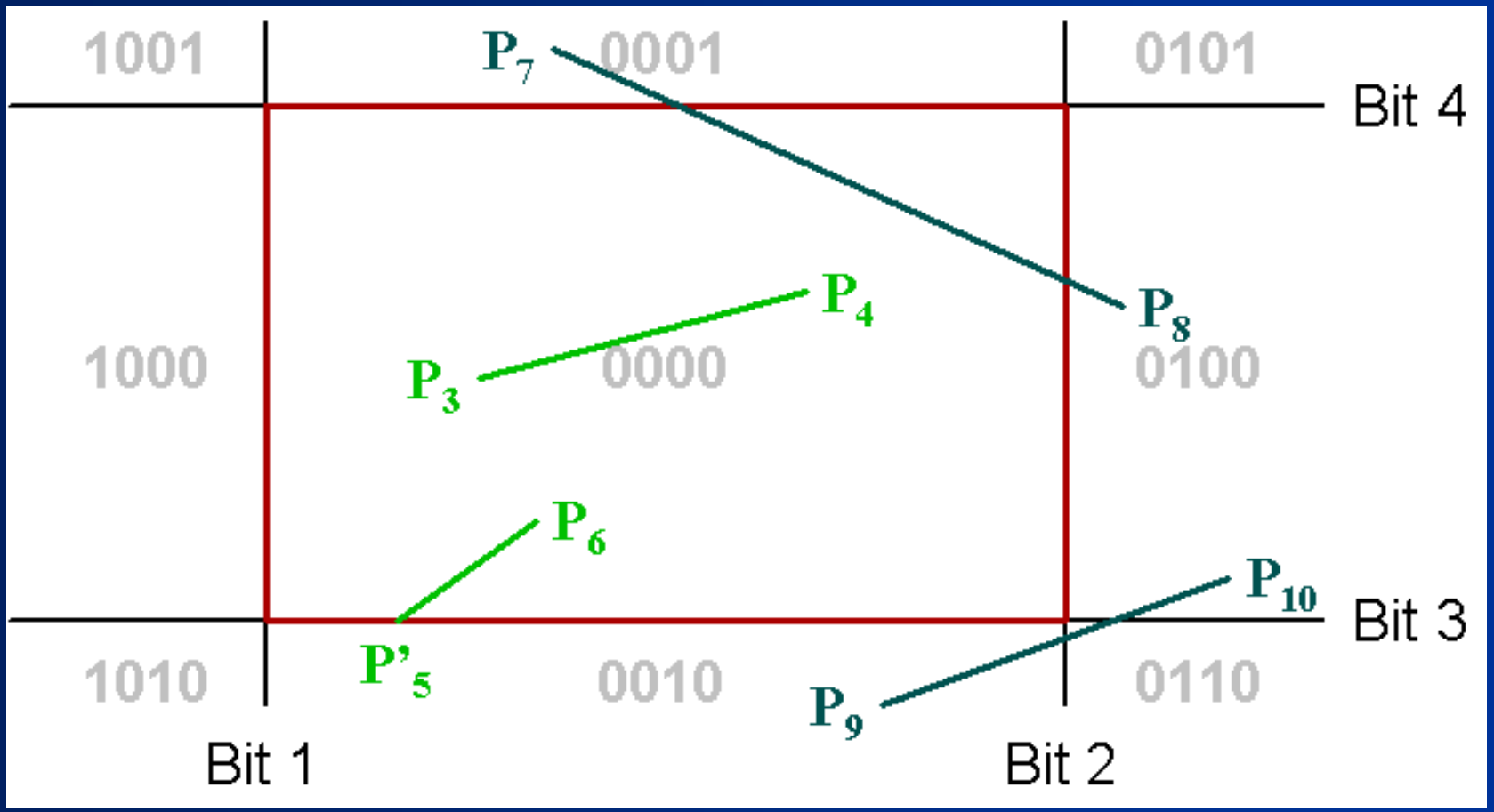
- Lines that cannot be identified as completely inside or outside a clip window are checked for intersection with boundaries.



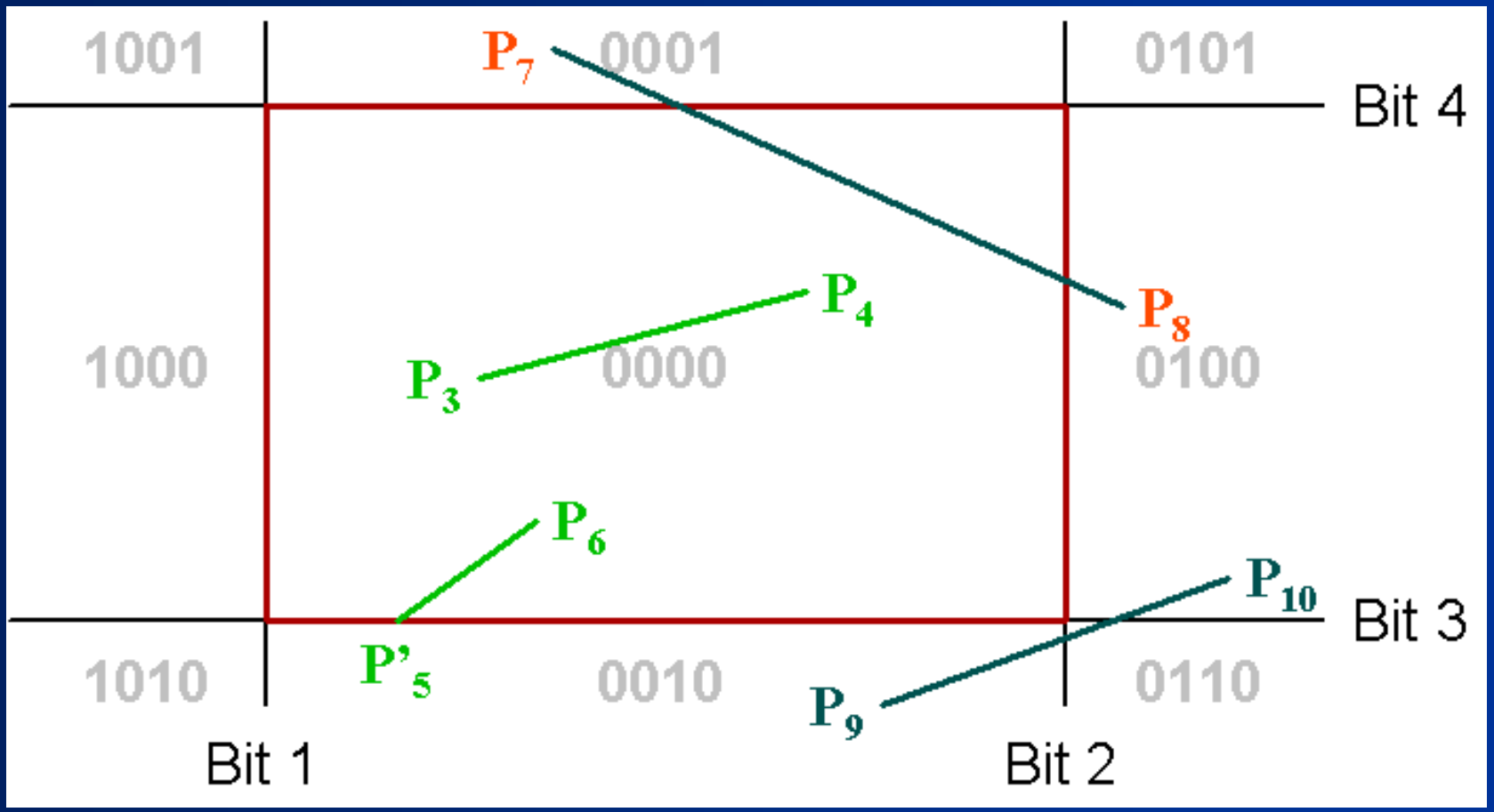
- Lines that cannot be identified as completely inside or outside a clip window are checked for intersection with boundaries.



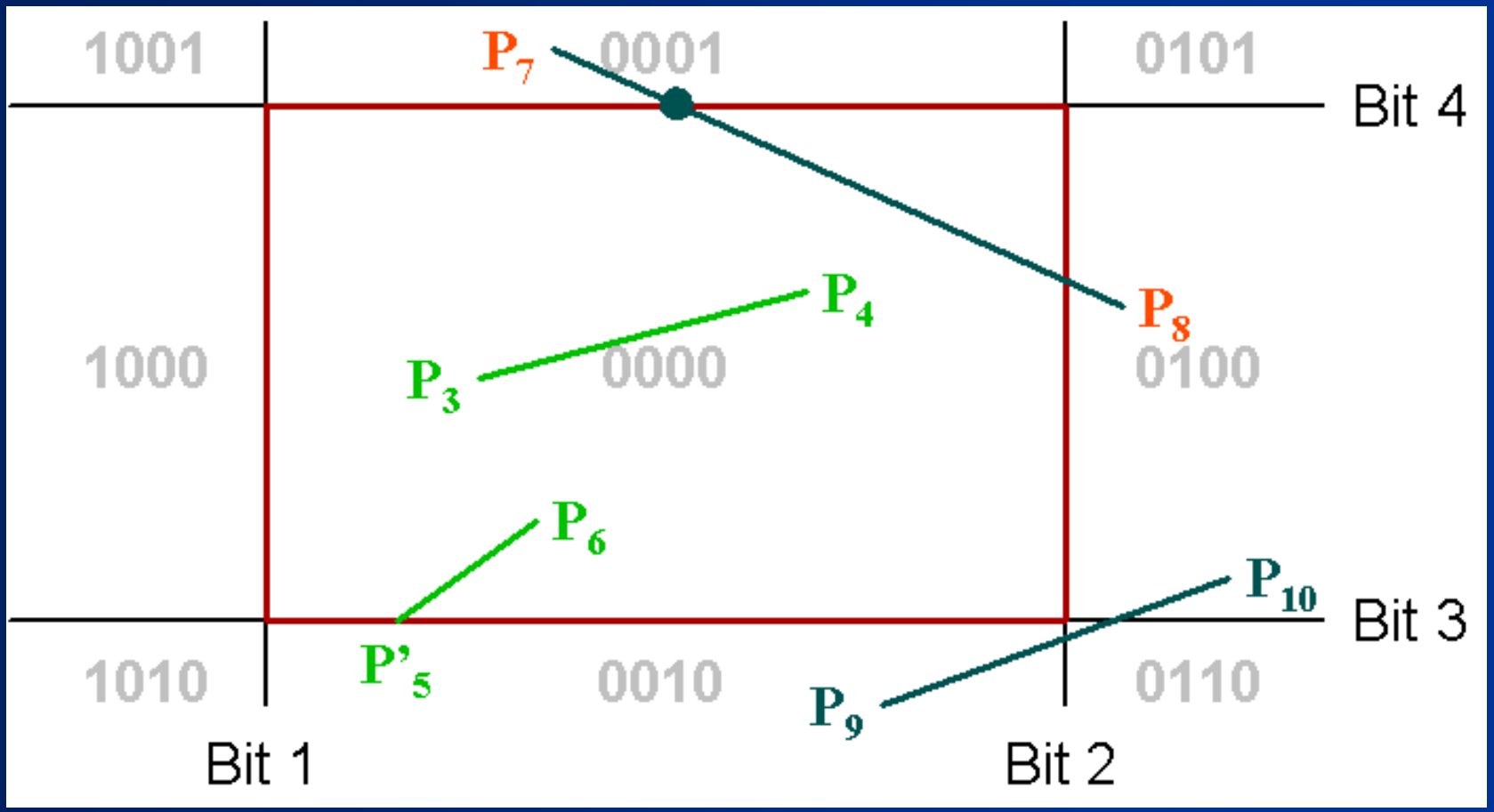
- Lines that cannot be identified as completely inside or outside a clip window are checked for intersection with boundaries.



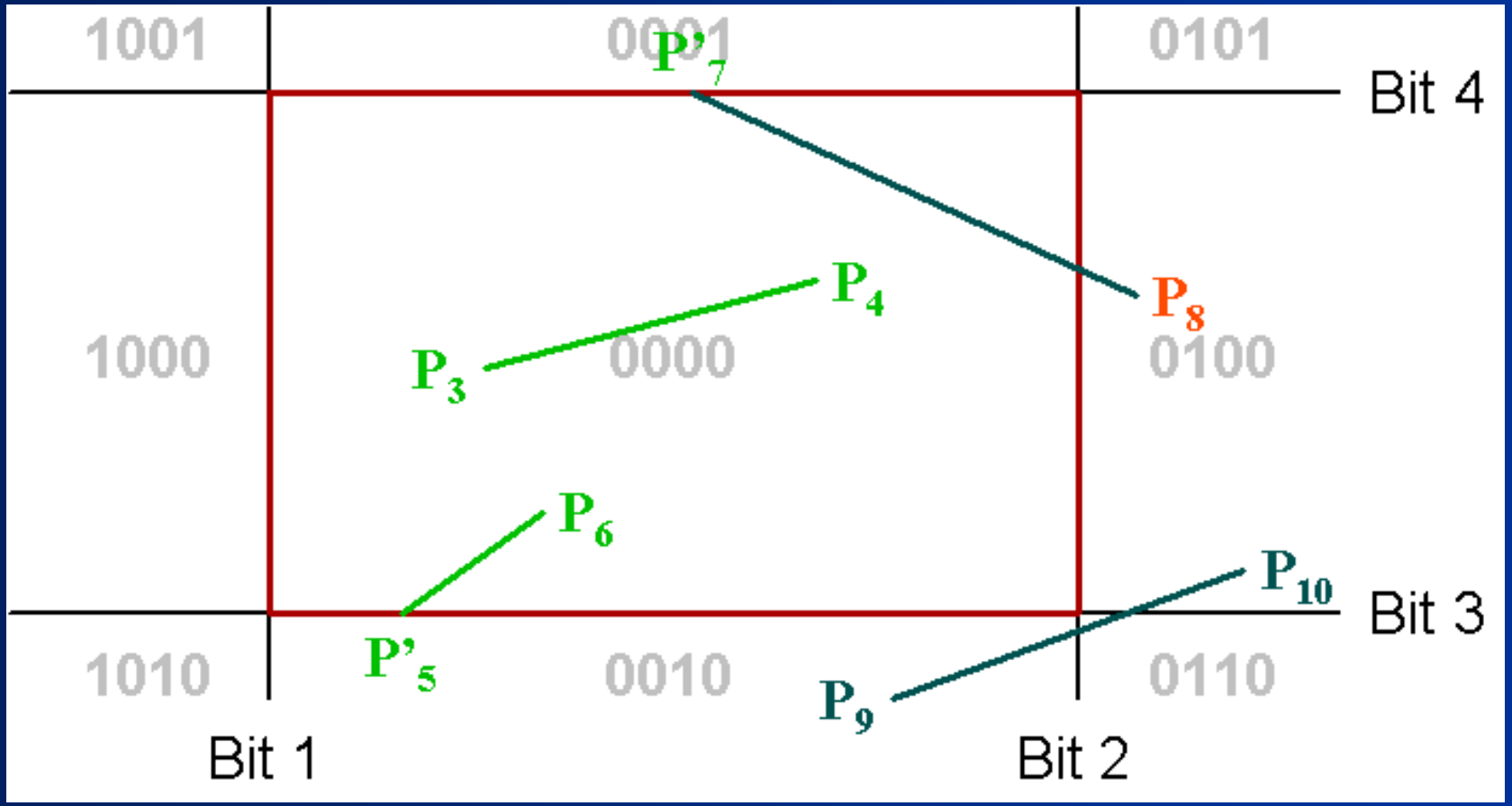
- Lines that cannot be identified as completely inside or outside a clip window are checked for intersection with boundaries.



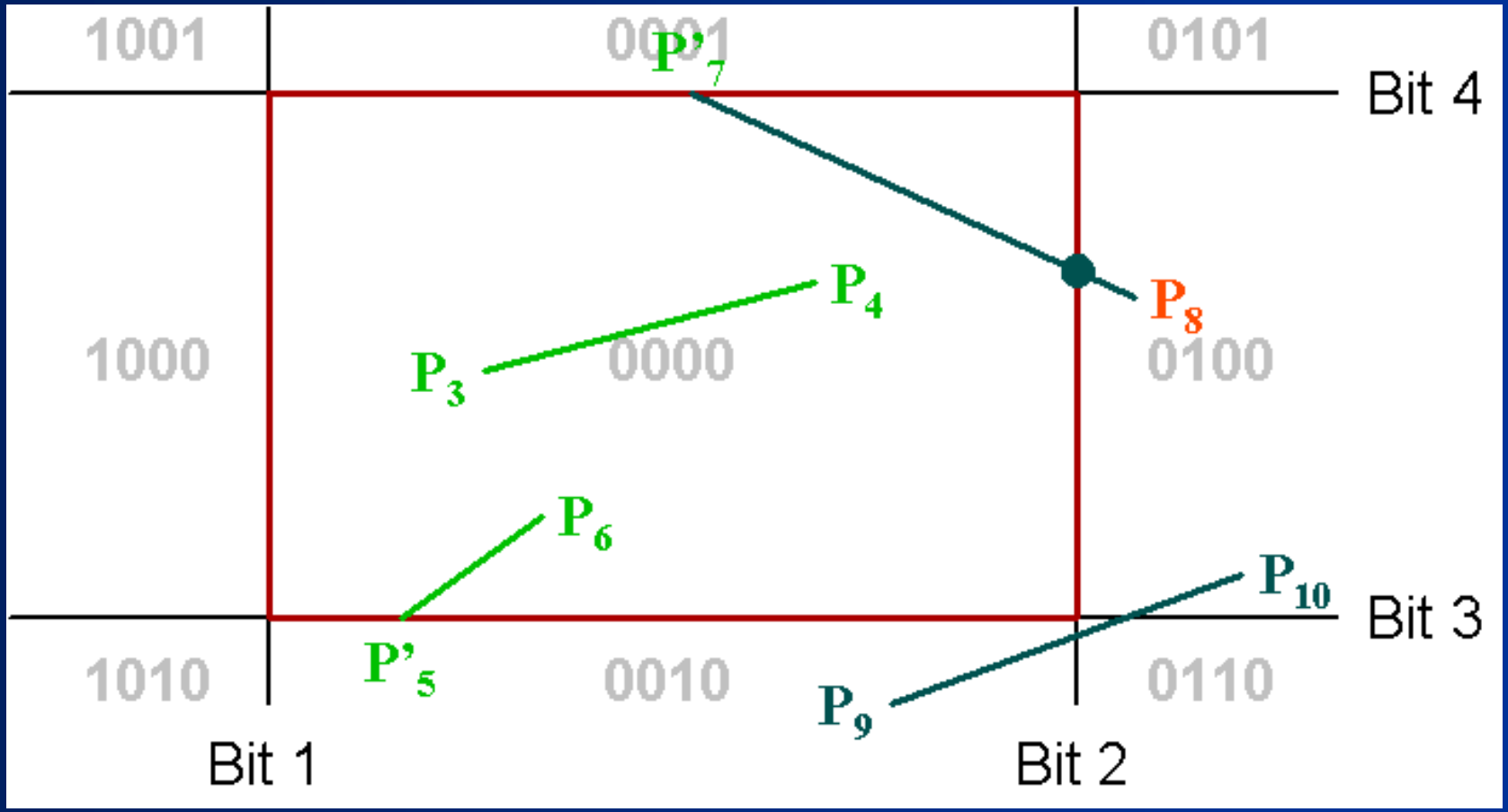
- Lines that cannot be identified as completely inside or outside a clip window are checked for intersection with boundaries.



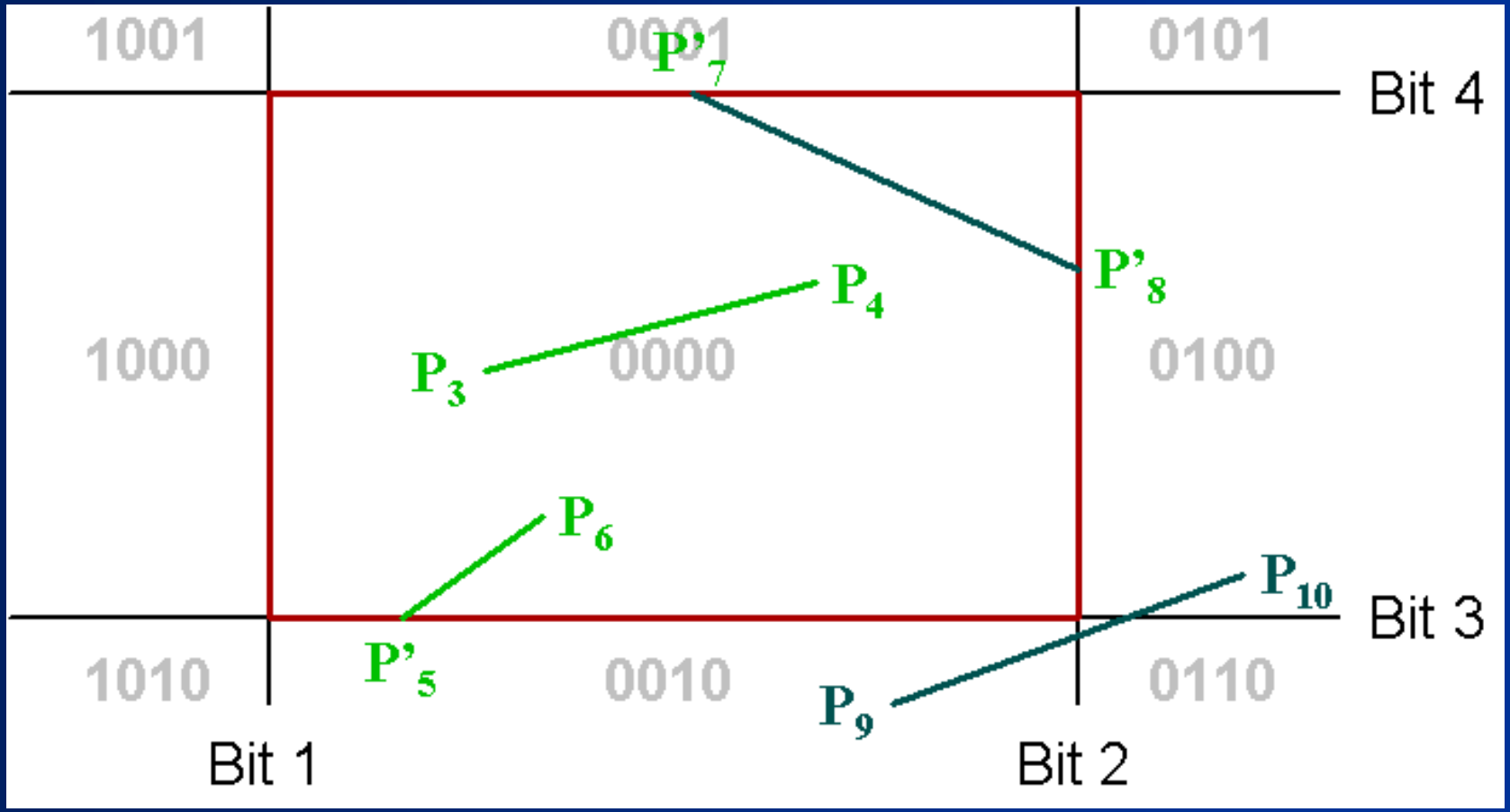
- Lines that cannot be identified as completely inside or outside a clip window are checked for intersection with boundaries.



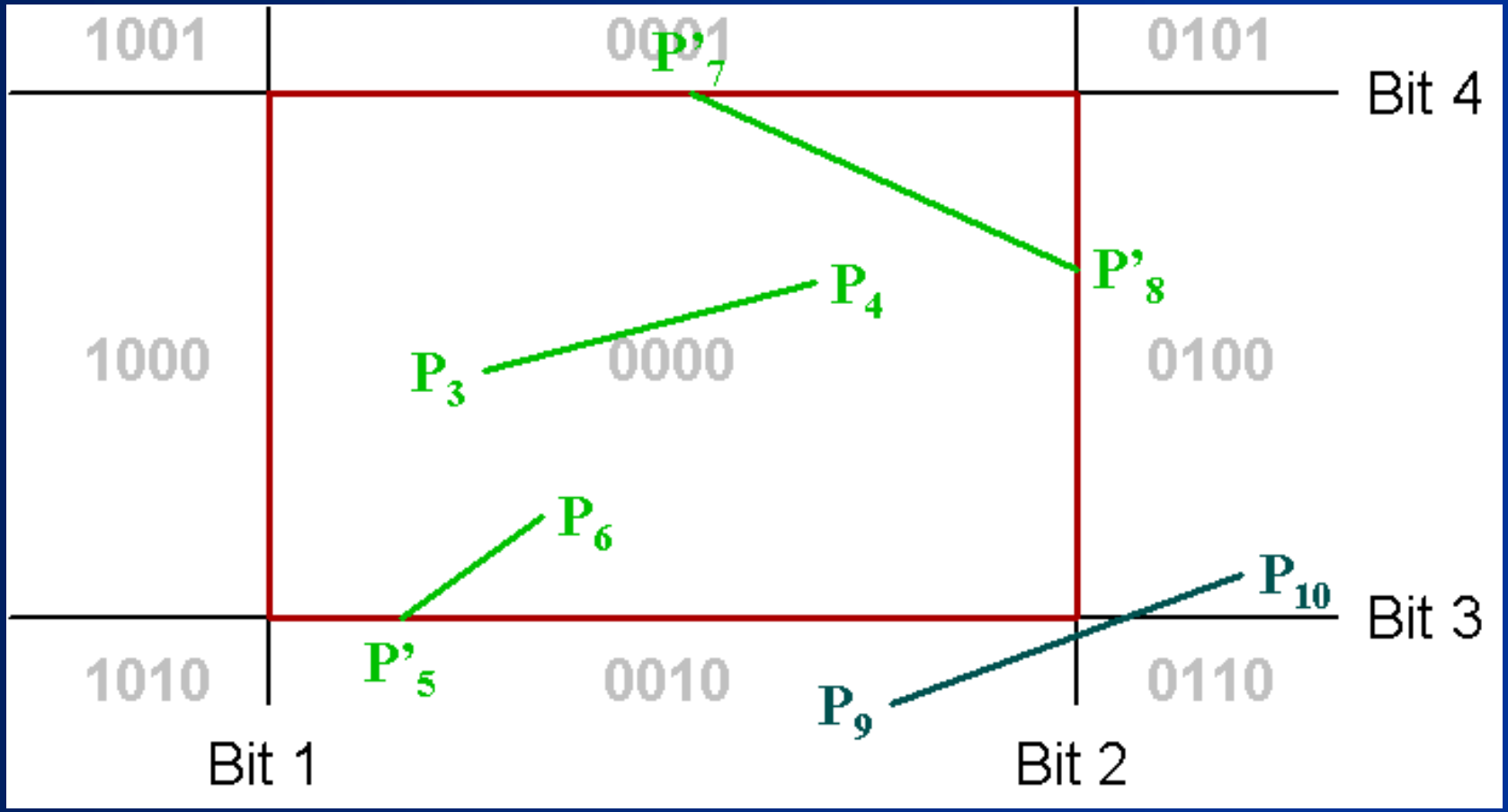
- Lines that cannot be identified as completely inside or outside a clip window are checked for intersection with boundaries.



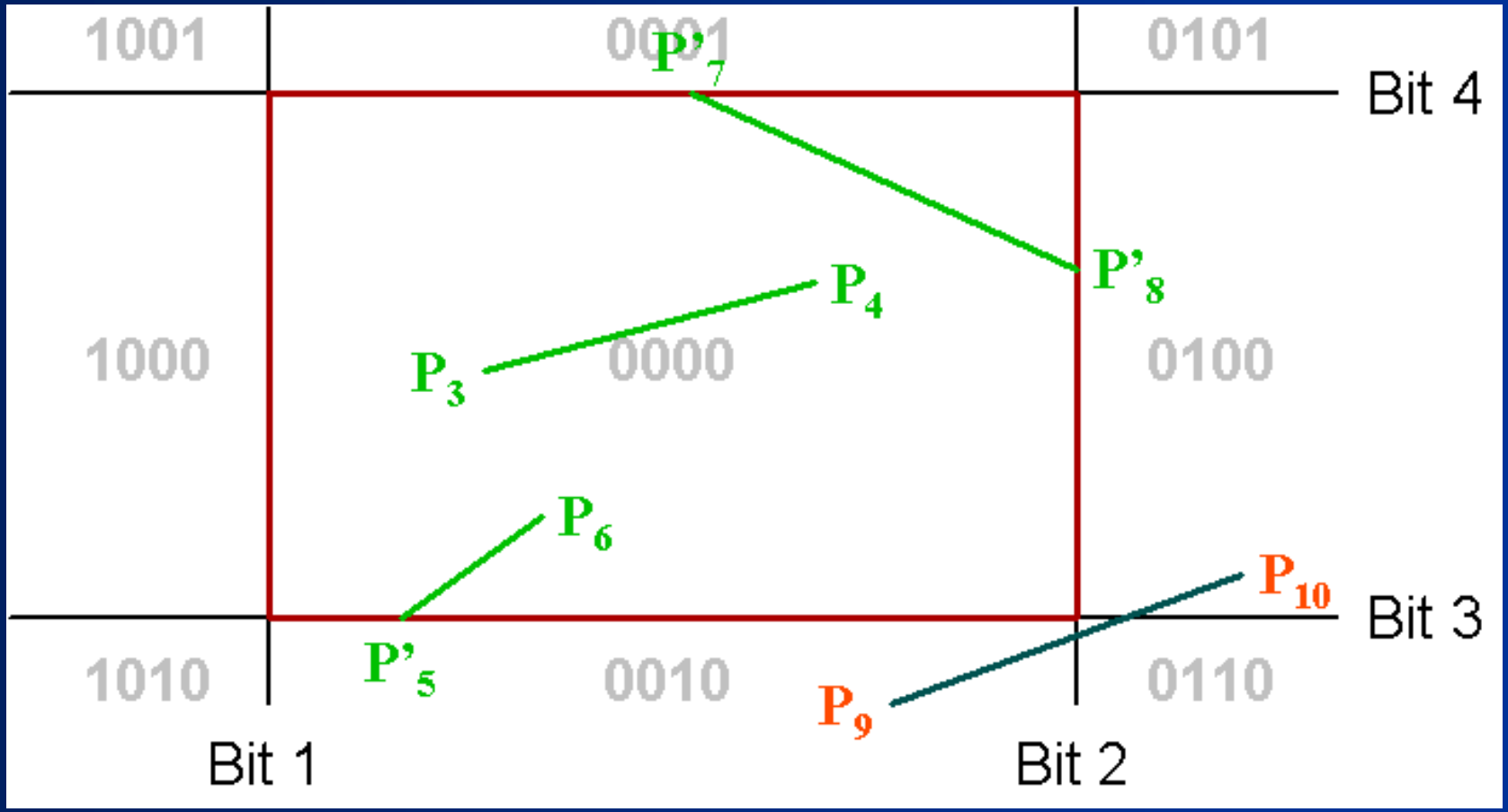
- Lines that cannot be identified as completely inside or outside a clip window are checked for intersection with boundaries.



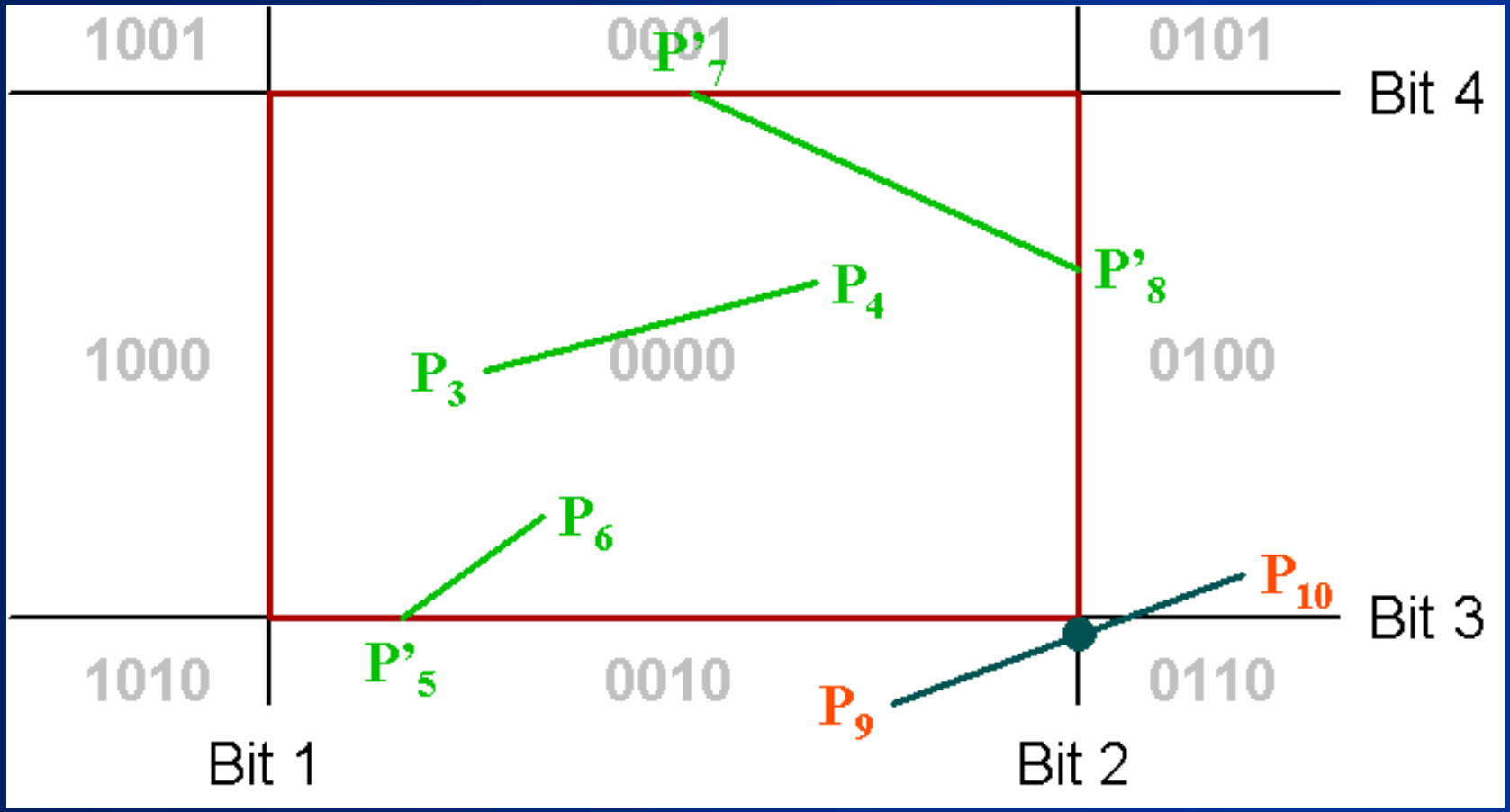
- Lines that cannot be identified as completely inside or outside a clip window are checked for intersection with boundaries.



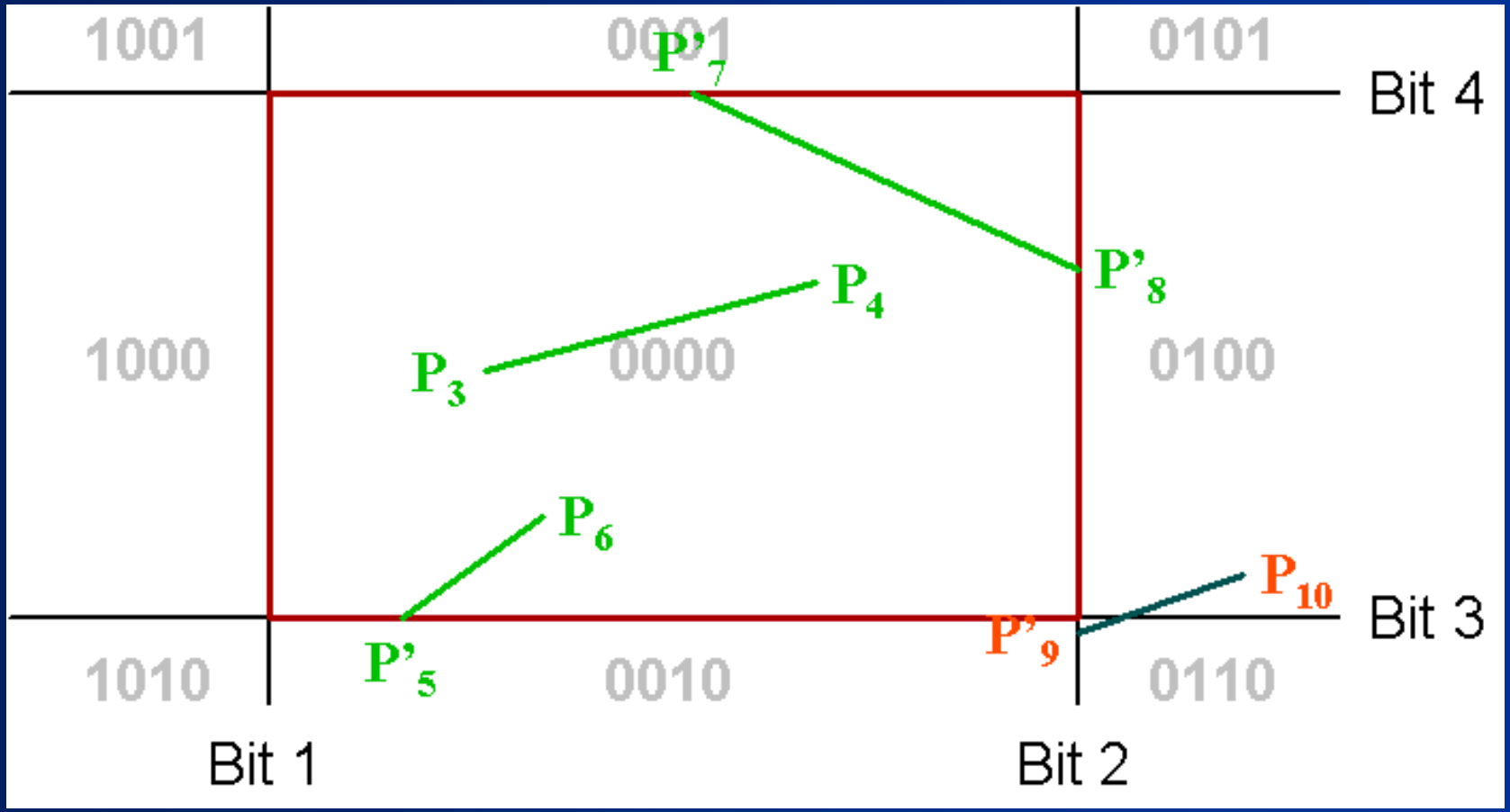
- Lines that cannot be identified as completely inside or outside a clip window are checked for intersection with boundaries.



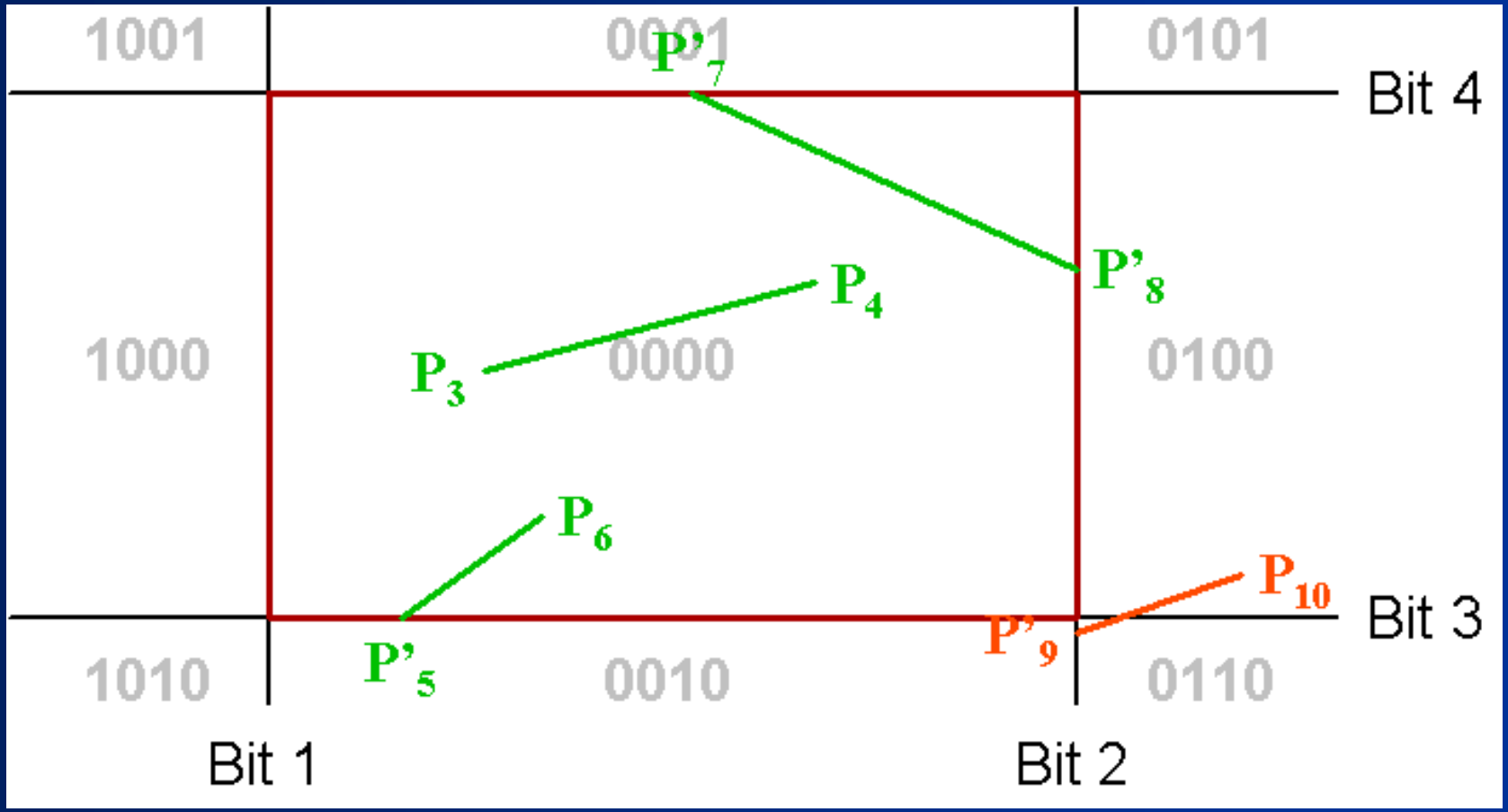
- Lines that cannot be identified as completely inside or outside a clip window are checked for intersection with boundaries.



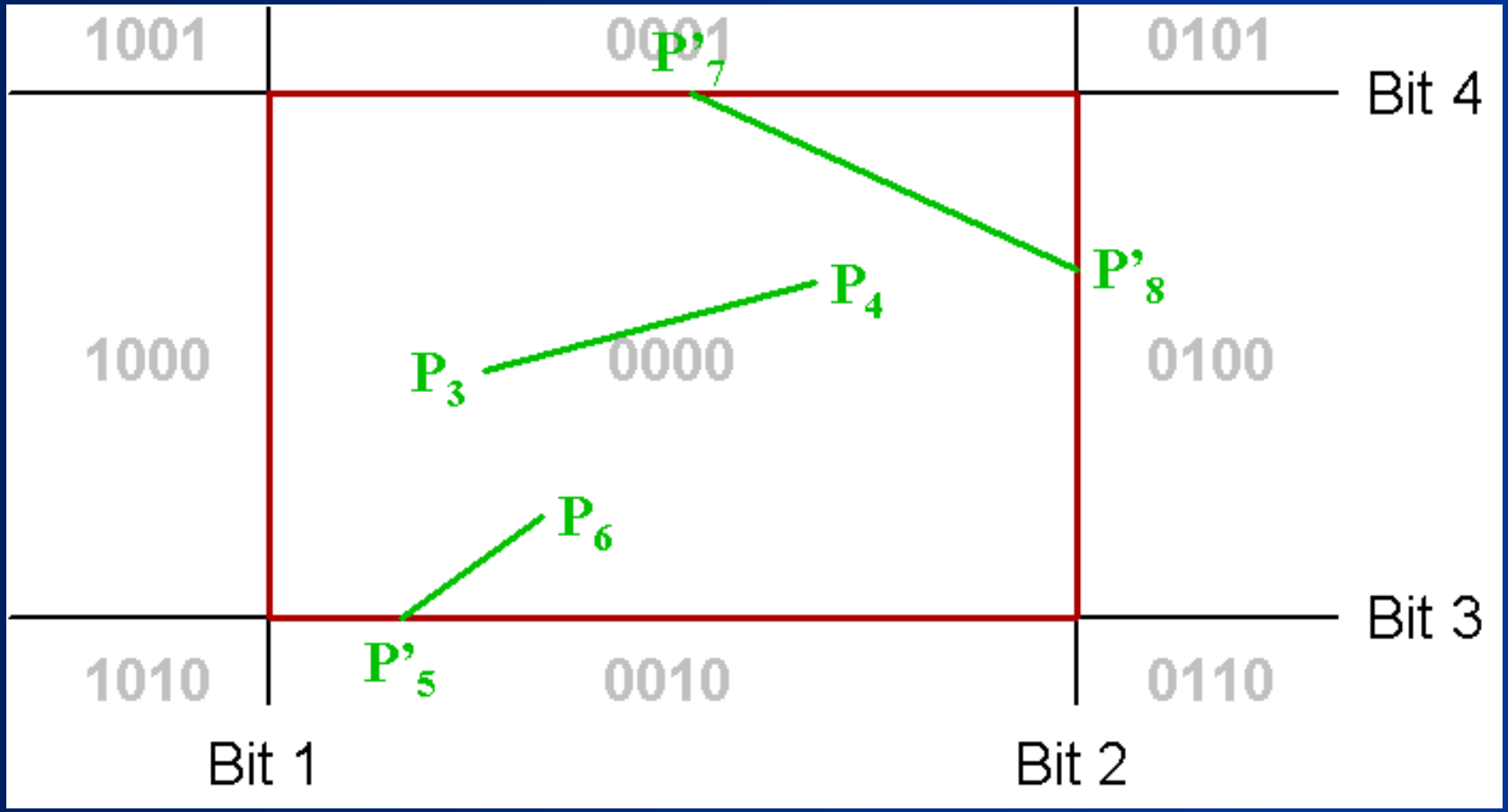
- Lines that cannot be identified as completely inside or outside a clip window are checked for intersection with boundaries.



- 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

Henry Ford Int'l College, Kalanki, Kathmandu
By: Hari Prashad Pant

Liang barsky Clipping

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

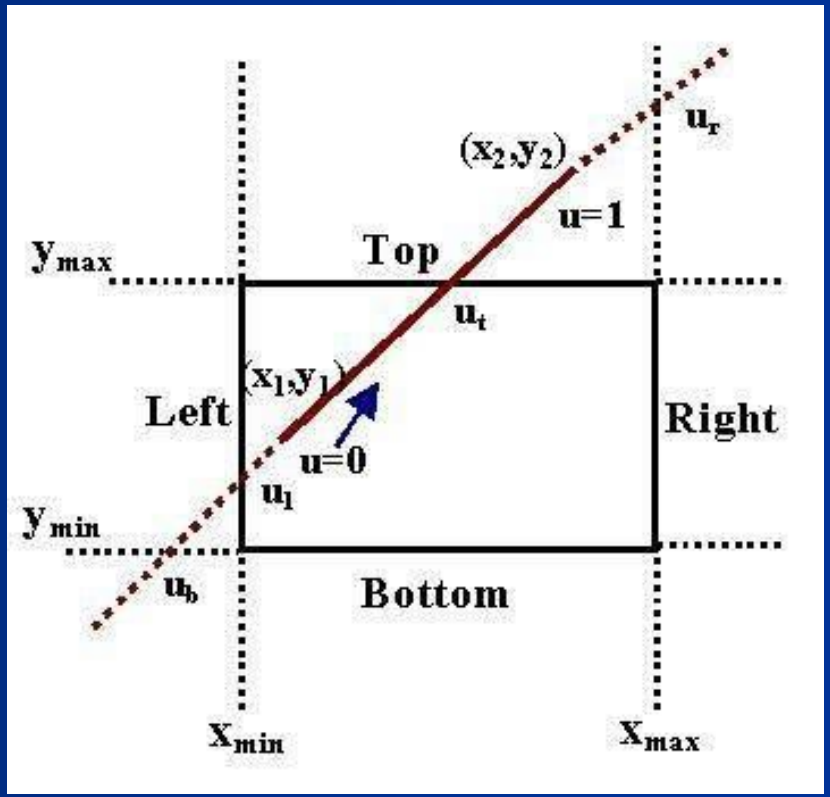
$$\begin{cases} x = x_1 + u\Delta x \\ \qquad\qquad 0 \le u \le 1 \\ y = y_1 + u\Delta y \end{cases}$$

$$\Delta x = x_2 - x_1$$

$$\Delta y = y_2 - y_1$$

Liang barsky Clipping

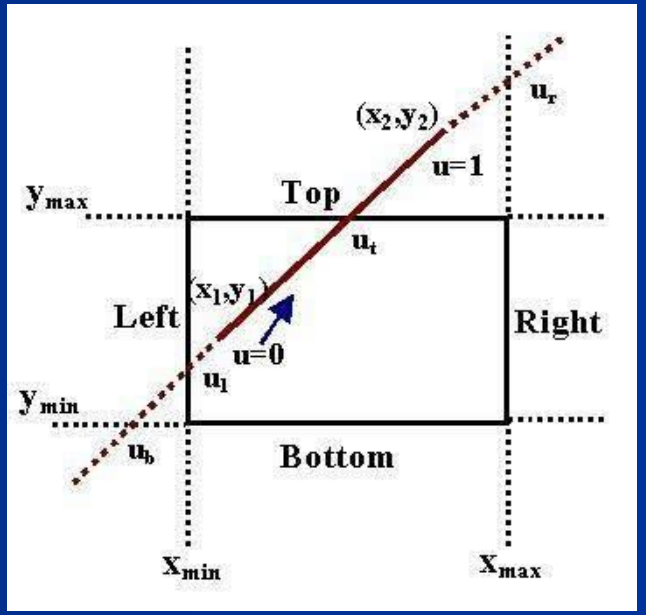
- When we traverse along the extended line with u increasing from $-\infty$ to ∞ ,
- we first move from the outside to the inside of the clipping window's two boundary lines (bottom and left)
- Then move from the inside to the outside of the other two boundary lines (top and right)



Liang barsky Clipping

$$u_1 \leq u_2 \quad u_1 = Max(0, u_l, u_b) \quad u_2 = Min(0, u_t, u_r)$$

- u_l: intersection the window's **left**
- u_b: intersection the window's **bottom**
- u_r: intersection the window's **right**
- u_t: 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: $up_k \le q_k, \quad k = 1,2,3,4$

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

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

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

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

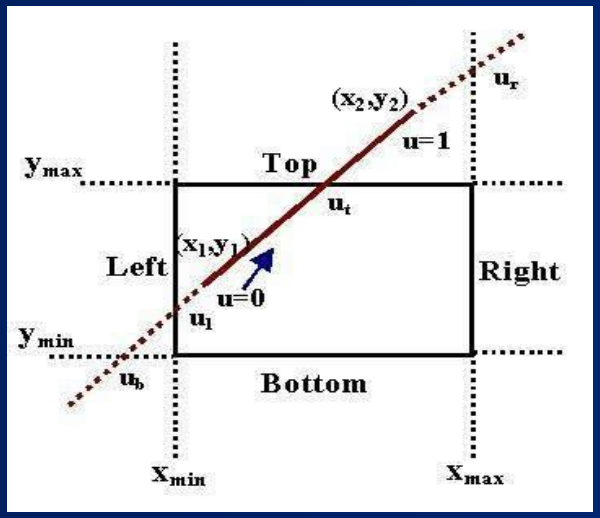
if $q_k \geq 0$ the line is completely inside (need further consideration)

- 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.

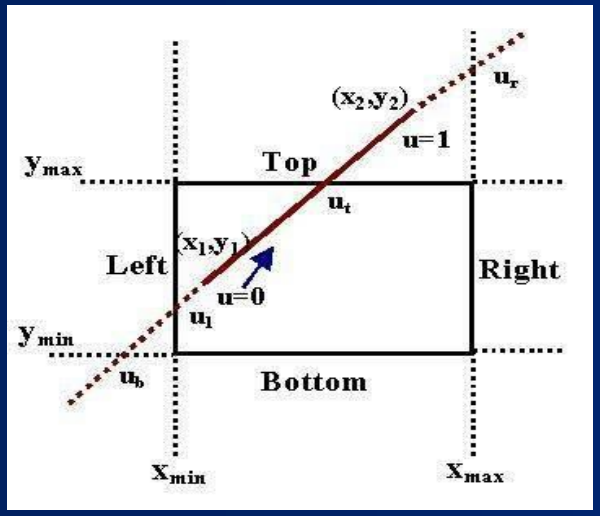


$$p_1 = -\Delta x, \quad q_1 = x_1 - x_{\min} \quad (left)$$
$$p_2 = \Delta x, \quad q_2 = xw_{\max} - x_1 \quad (right)$$
$$p_3 = -\Delta y, \quad q_3 = y_1 - y_{\min} \quad (bottom)$$
$$p_4 = \Delta y, \quad q_4 = yw_{\max} - y_1 \quad (top)$$

Liang barsky Clipping

- **A four step process …**

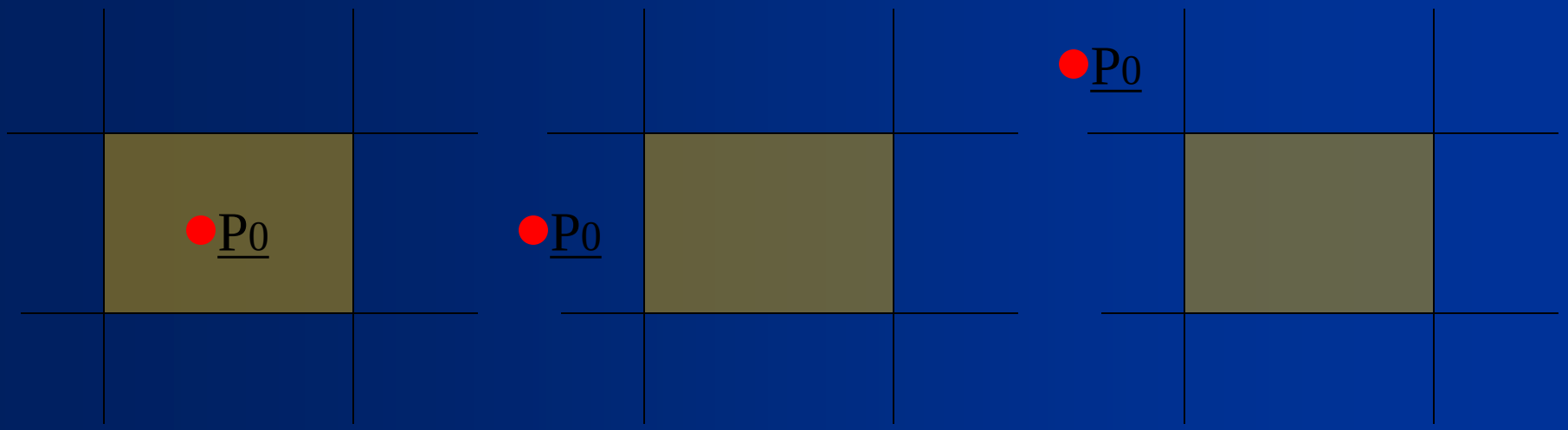
3. For all **k** such that $p_k > 0$, calculate $r_k = q_k / p_k$. Let u_2 be the minimum of the set containing **1** and the calculated **r** values.
4. If $u_1 > u_2$, eliminate the line since it is completely outside the clipping window, Otherwise, use u_1 and u_2 to calculate the endpoints of the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



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

Nicholl-Lee-Nicholl Line Clipping

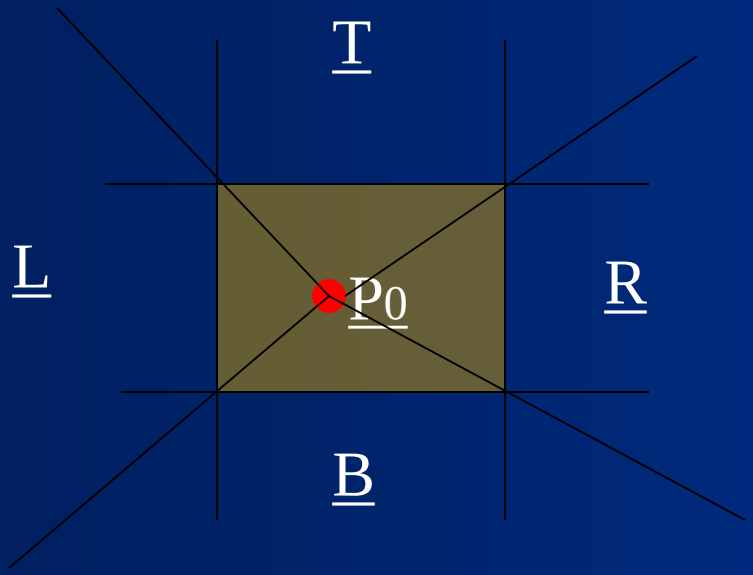
- Creates more testing regions around the clipping window
  - Avoids multiple line-intersection calculations
- Initial testing to determine if a line segment is completely inside the clipping window can be done using previous methods
- If trivial acceptance or rejection is not possible the NLN algorithm sets up additional regions



For line with endpoints P_0P_{end}, there are three different positions to consider - all others can be derived from these by symmetry considerations

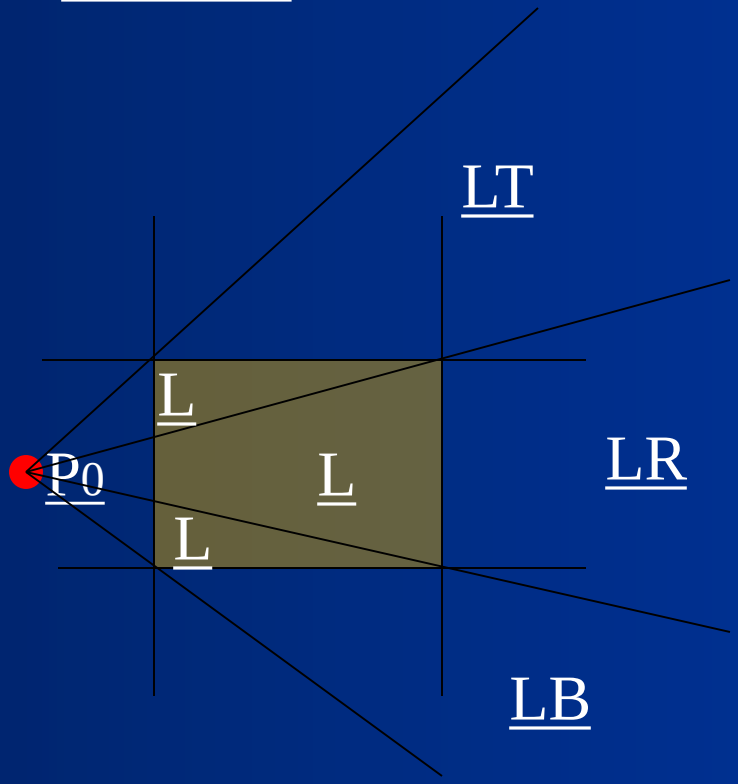
For each case, we generate specialized test regions for other endpoint P_{end}, which use simple tests (slope, >, <), and tells us which edges to clip against.

Case 1



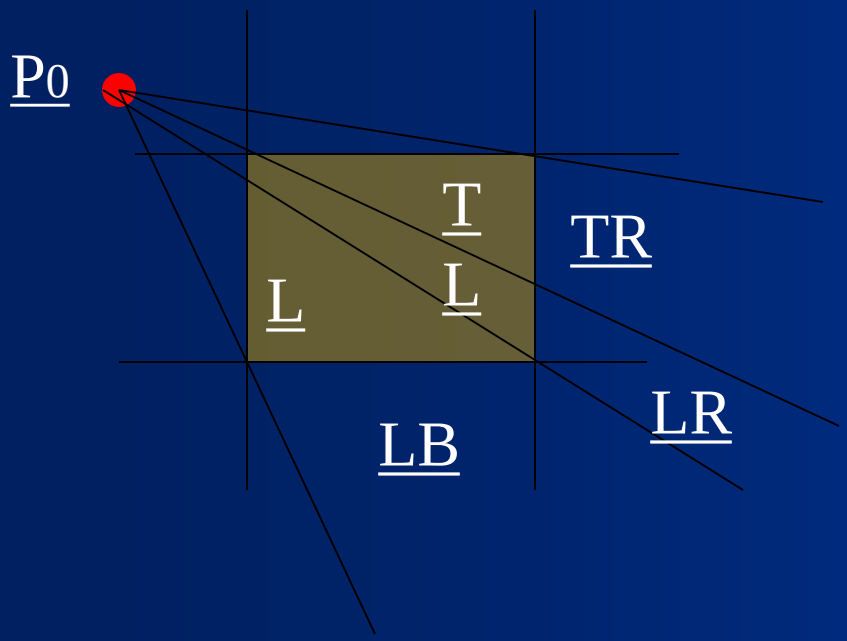
Find which of the four regions Pend lies in, then calculate the line intersection with the corresponding boundary

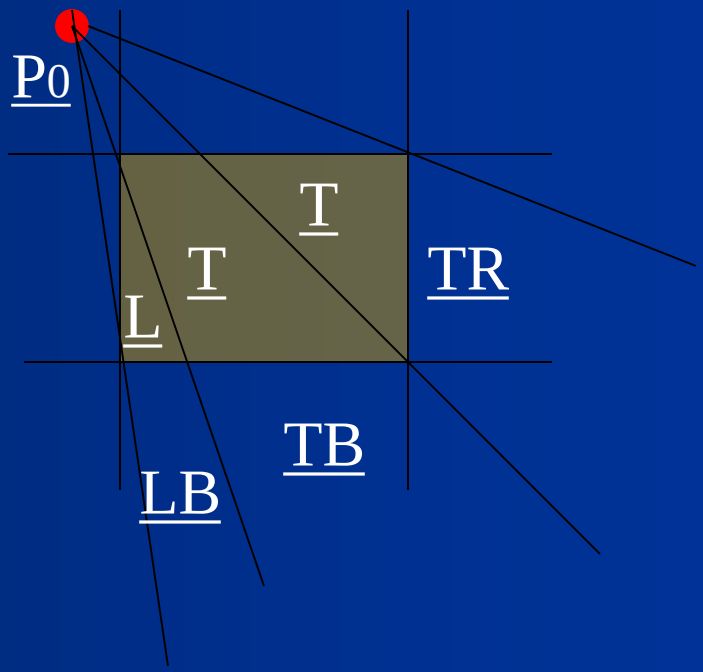
Case 2



Find which of the four regions Pend lies in, then calculate the line intersection with the corresponding boundary

Case 3 : 2 possibilities

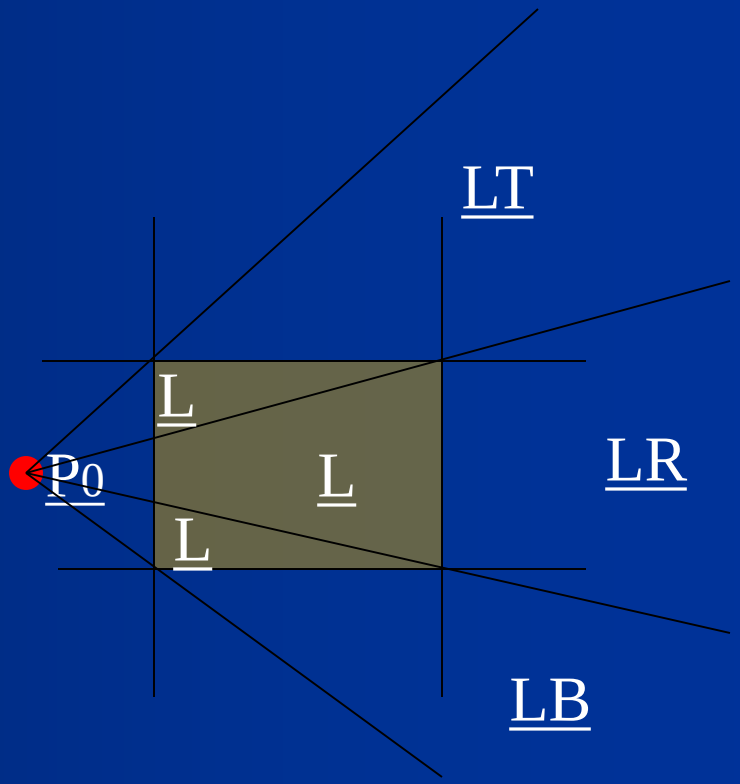




Find which of the five regions Pend lies in, then calculate the line intersection with the corresponding boundary

N-L-N Line clipping

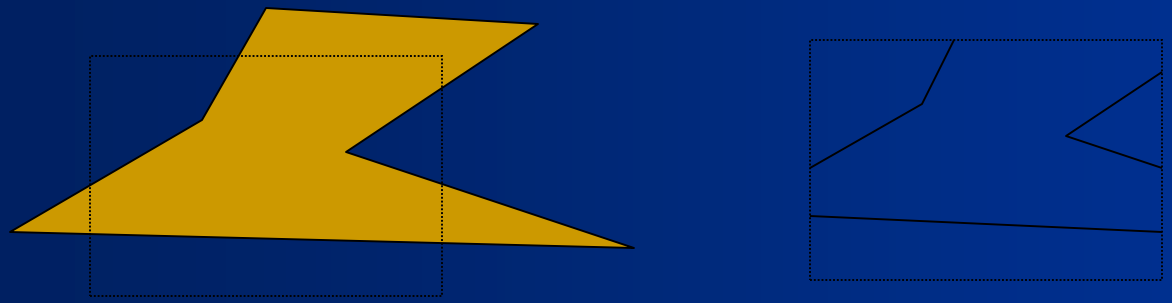
- To determine in which region P_{end} lies we compare the slope of $P_{end}P_o$ to the slopes of the boundaries of the NLN regions



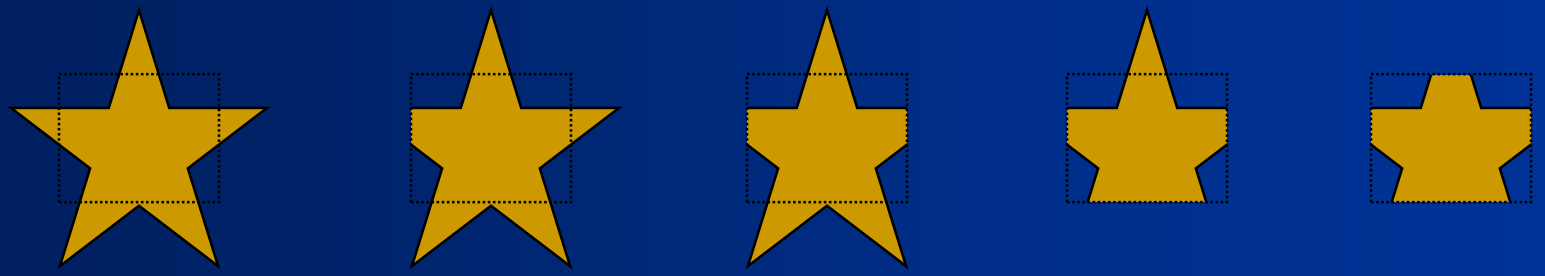
Number of cases explodes in 3D, making algorithm unsuitable

Polygon clipping

- Clipping a polygon fill area needs more than line-clipping of the polygon edges
  - would produce and unconnected set of lines
- Must generate one or more closed polylines, which can be filled with the assigned colour or pattern



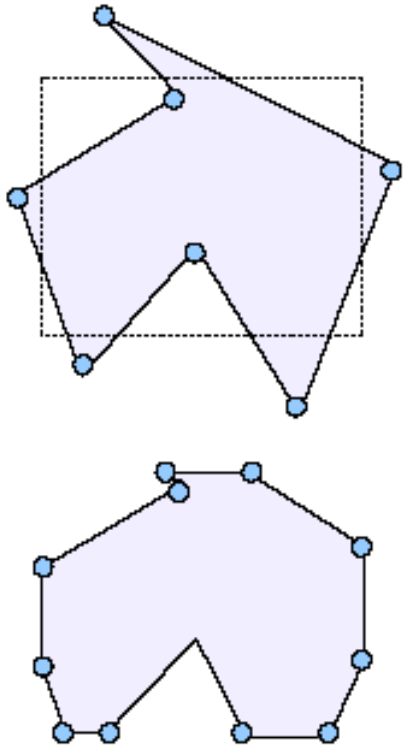
Polygon Clipping



Original Polygon | Clip Left | Clip Right | Clip Bottom | Clip Top

Polygon Clipping

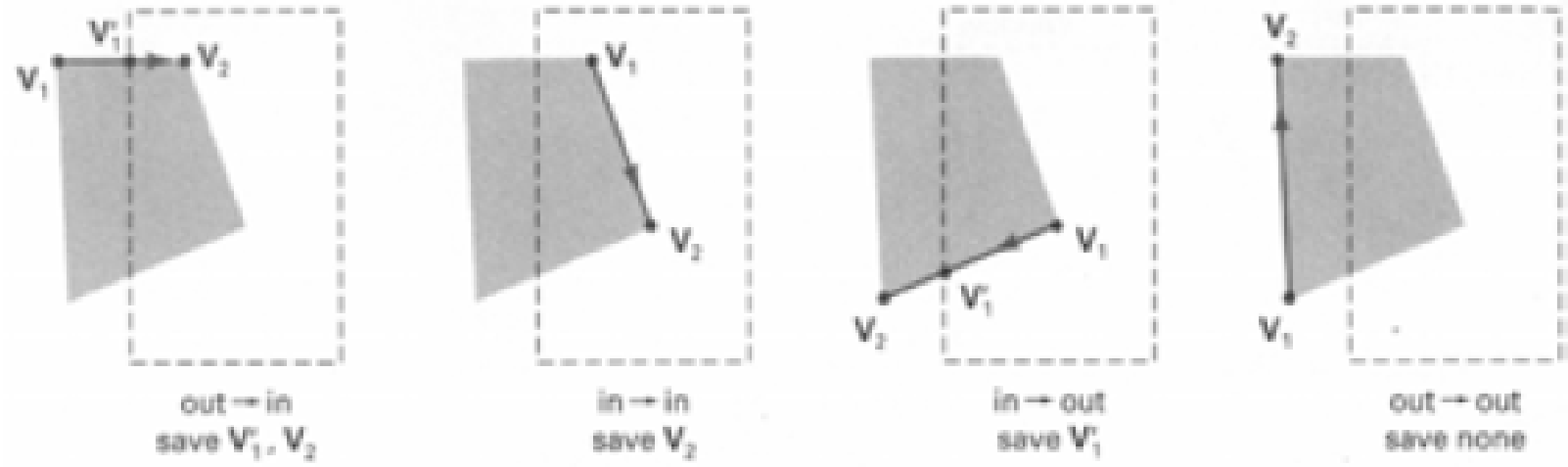
- Find the vertices of the new polygon(s) inside the window.
- Sutherland-Hodgeman Polygon Clipping:
  Check each edge of the polygon against all window boundaries. Modify the vertices based on transitions. Transfer the new edges to the next clipping boundary.



Sutherland-Hodgeman Polygon Clipping

Four test cases:

1. First vertex inside and the second outside (in-out pair)
2. Both vertices inside clip window
3. First vertex outside and the second inside (out-in pair)
4. Both vertices outside the clip window

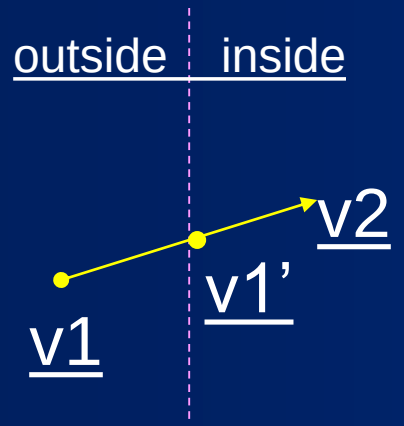


- Concave polygons may be displayed with extra lines.

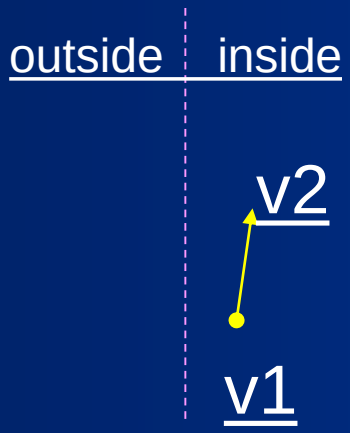
Sutherland-Hodgman Polygon Clipping

- Input each edge (vertex pair) successively.
- Output is a new list of vertices.
- Each edge goes through 4 clippers.
- The rule for each edge for each clipper is:
  - If first input vertex is outside, and second is inside, output the intersection and the second vertex
  - If first both input vertices are inside, then just output second vertex
  - If first input vertex is inside, and second is outside, output is the intersection
  - If both vertices are outside, output is nothing

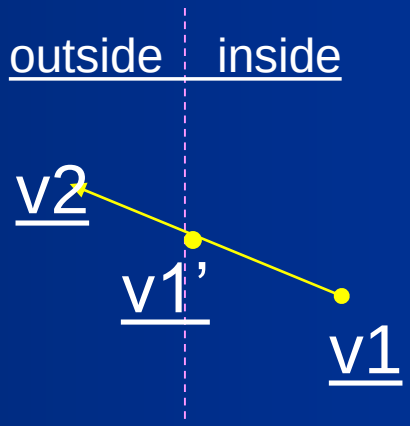
Sutherland-Hodgman Polygon Clipping: Four possible scenarios at each clipper



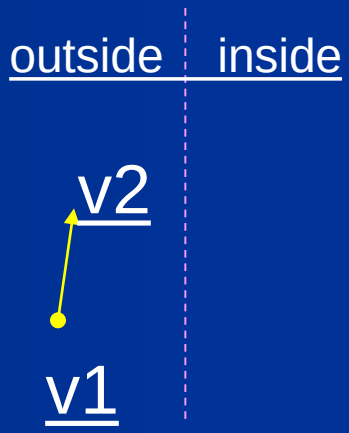
Outside to inside:
Output: v1' and v2



Inside to inside:
Output: v2



Inside to outside:
Output: v1'



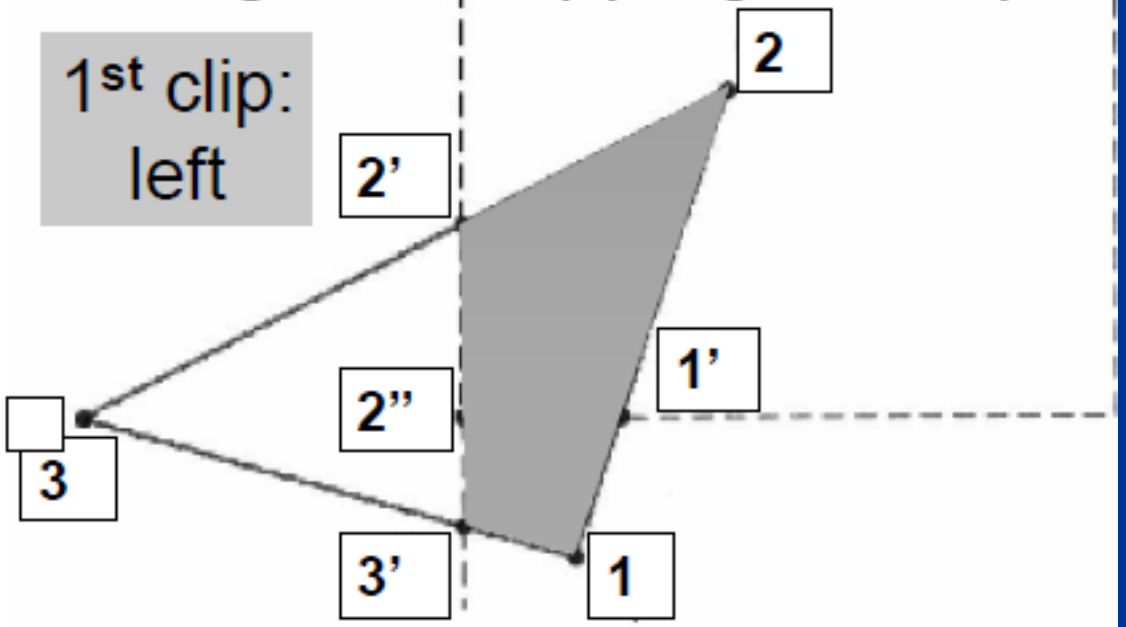
Outside to outside:
Output: nothing

Sutherland-Hodgeman Algorithm: Combination of the 4 Passes

- the polygon is clipped against each of the 4 borders separately,
  that would produce 3 intermediate results.
- by calling the 4 tests *recursively*,
  (or by using a clipping pipeline)
  every result point is immediately processed on, so that only *one* result list is produced

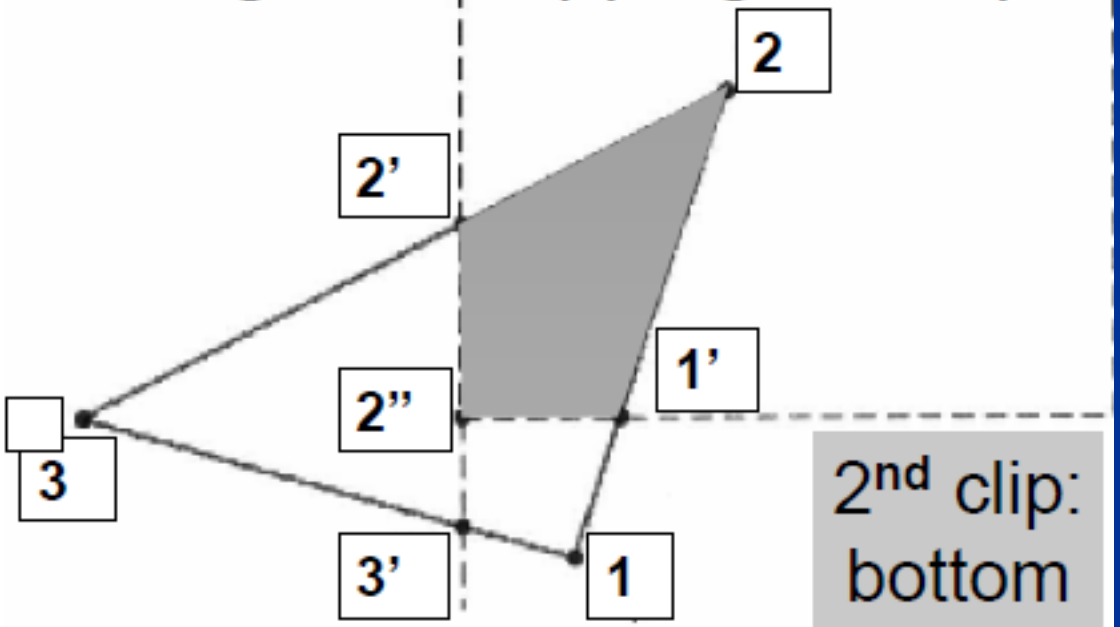
Sutherland-Hodgman Clipping Example

- pipeline of boundary clippers to avoid intermediate vertex lists



Processing the vertices of the polygon through a boundary-clipping pipeline.

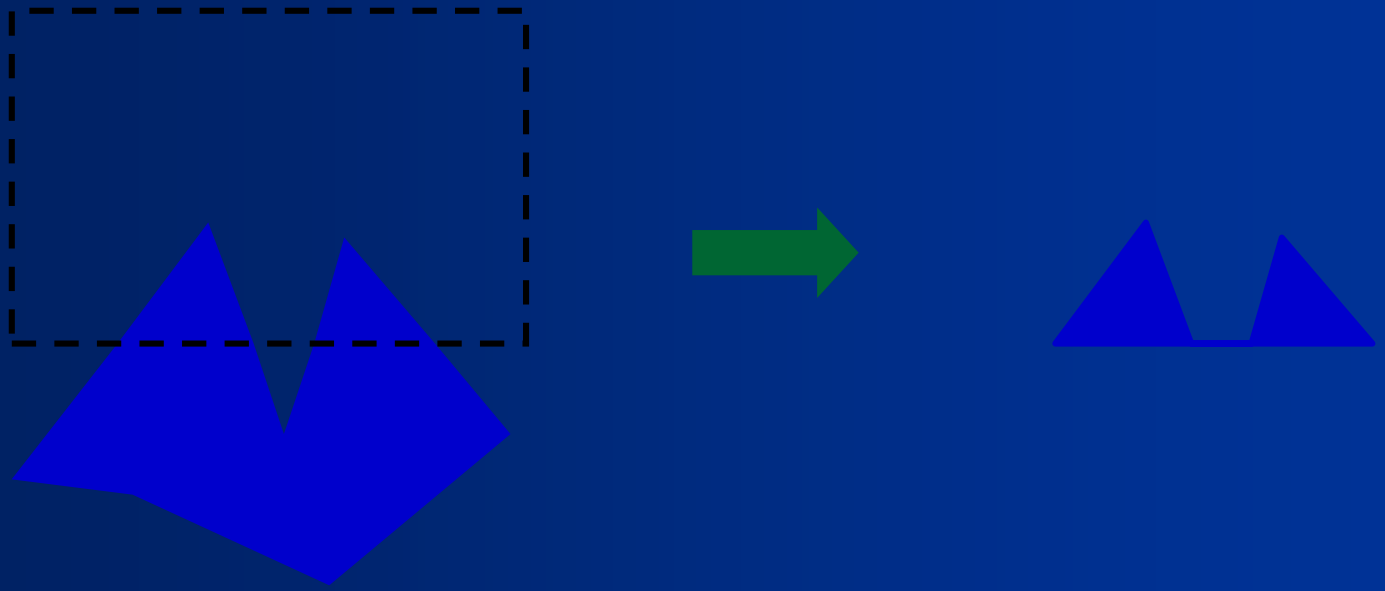
Sutherland-Hodgman Clipping Example



After all vertices are processed through the pipeline, the vertex list for the clipped polygon is [1', 2, 2', 2"]

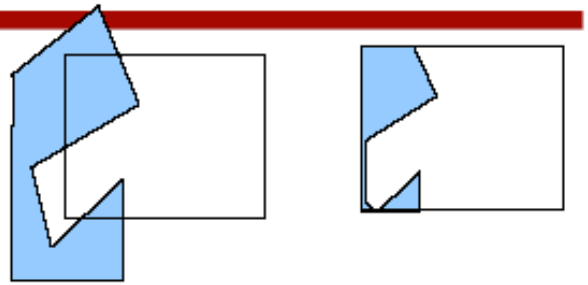
Sutherland-Hodgman Polygon Clipping

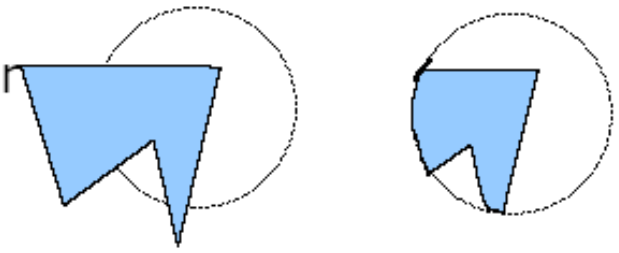
- The algorithm correctly clips convex polygons, but may display extraneous lines for concave polygons

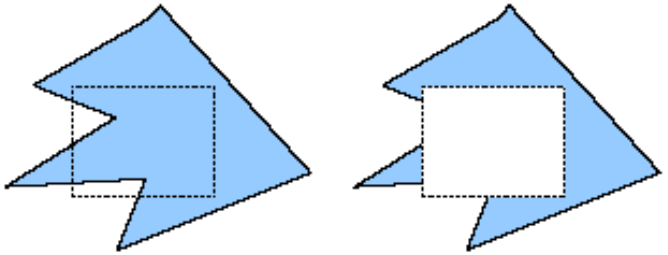


Other Issues in Clipping

- Problem in Sutherland-Hodges. Weiler-Atherton has a solution
- Clipping other shapes: Circle, Ellipse, Curves.
- Clipping a shape against another shape
- Clipping the exteriors.







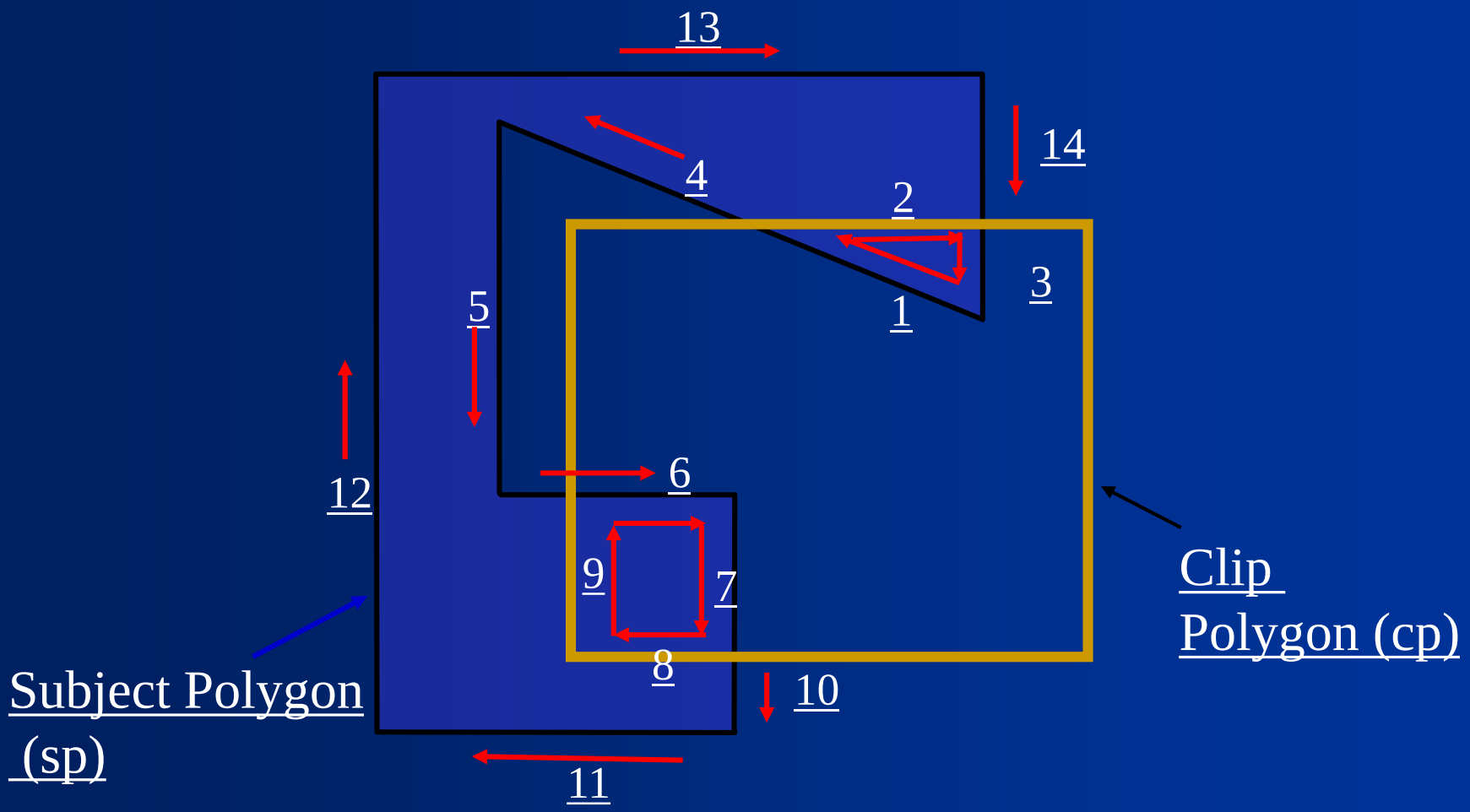
Weiler-Atherton Polygon Clipping

- Another approach to polygon clipping
- No extra clipping outside window
- Works for arbitrary shapes
- Avoids degenerate polygons

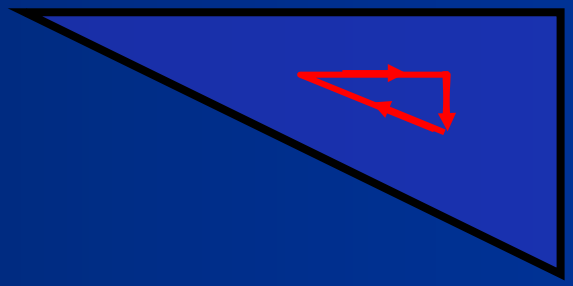
Outline of Weiler algorithm:

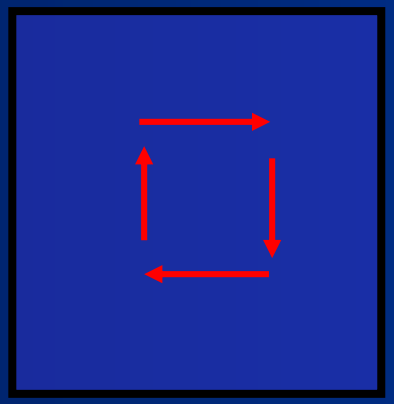
- Replace crossing points with vertices
- Form linked lists of edges
- Change links at vertices
- Enumerate polygon patches

Weiler-Atherton Clipping
clockwise orientation of subject polygon



Gives “Right” Answer





Weiler-Atherton Clipping (clockwise orientation of polygon)

- Start at first (inside) vertex
- Traverse polygon until hitting a window boundary
- Output intersection point *i*
- Turn *right*
- Follow window boundary until next intersection

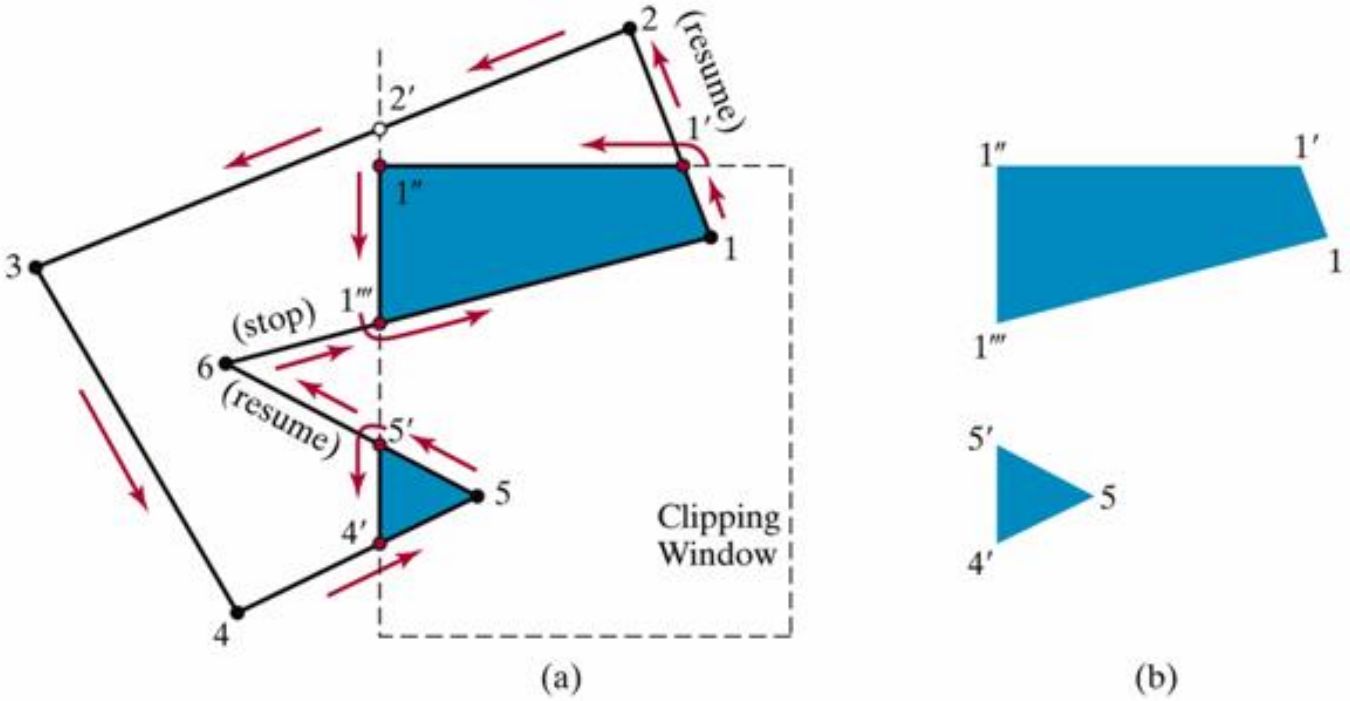
Weiler-Atherton Clipping

- Output second intersection
- Turn *right*, again, and follow subject polygon until closed
- Continue on subject polygon from first intersection point.
- Repeat processing until complete

Generalizations of W-A

- Can be extended to complex situations, arbitrary windows
- Stability issues can arise for such cases

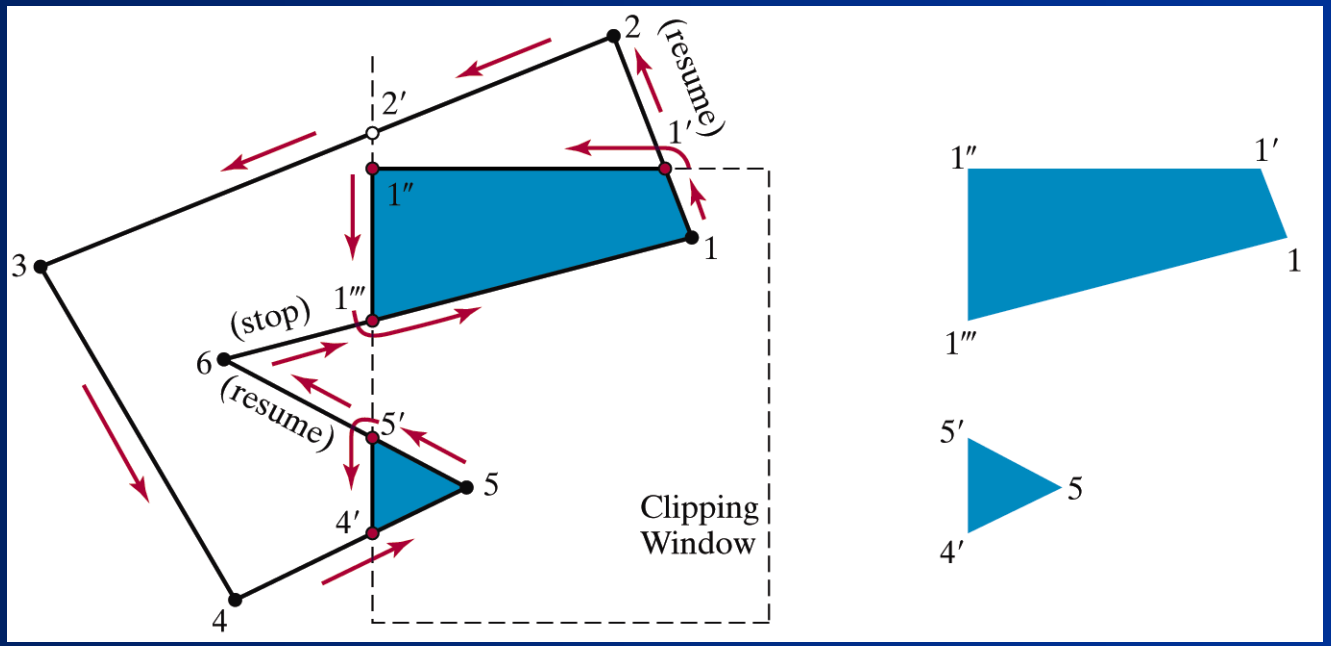
Weiler-Atherton Polygon Clipping counter-clockwise orientation of subject polygon



A concave polygon (a), defined with the vertex list {1, 2, 3, 4, 5, 6 }, is clipped using the Weiler-Atherton algorithm to generate the two lists {1, 1′,1″,1‴} and {4′,5,5′}, which represent the separate polygon fill areas shown in (b).

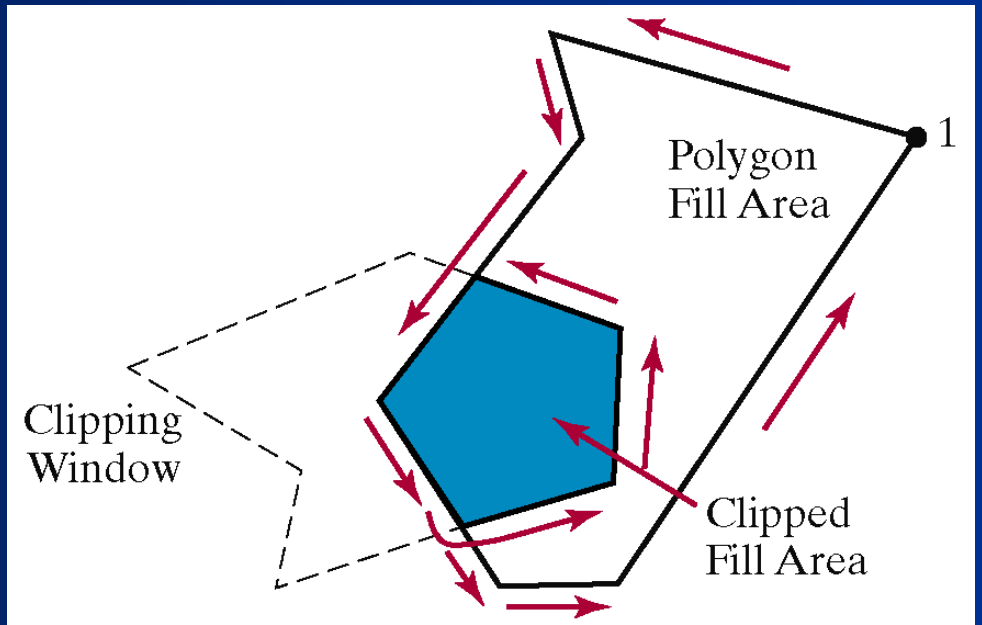
Weiler-Atherton Polygon Clipping counter-clockwise orientation of subject polygon

- For an outside-to-inside pair of vertices, follow the polygon boundary
- For an inside-to-outside pair of vertices, follow the window boundary in a counter-clockwise direction



Weiler-Atherton Polygon Clipping

- Polygon clipping using nonrectangular polygon clip windows



Clipping a polygon fill area against a concave-polygon clipping window using the Weiler-Atherton algorithm.

Curve Clipping

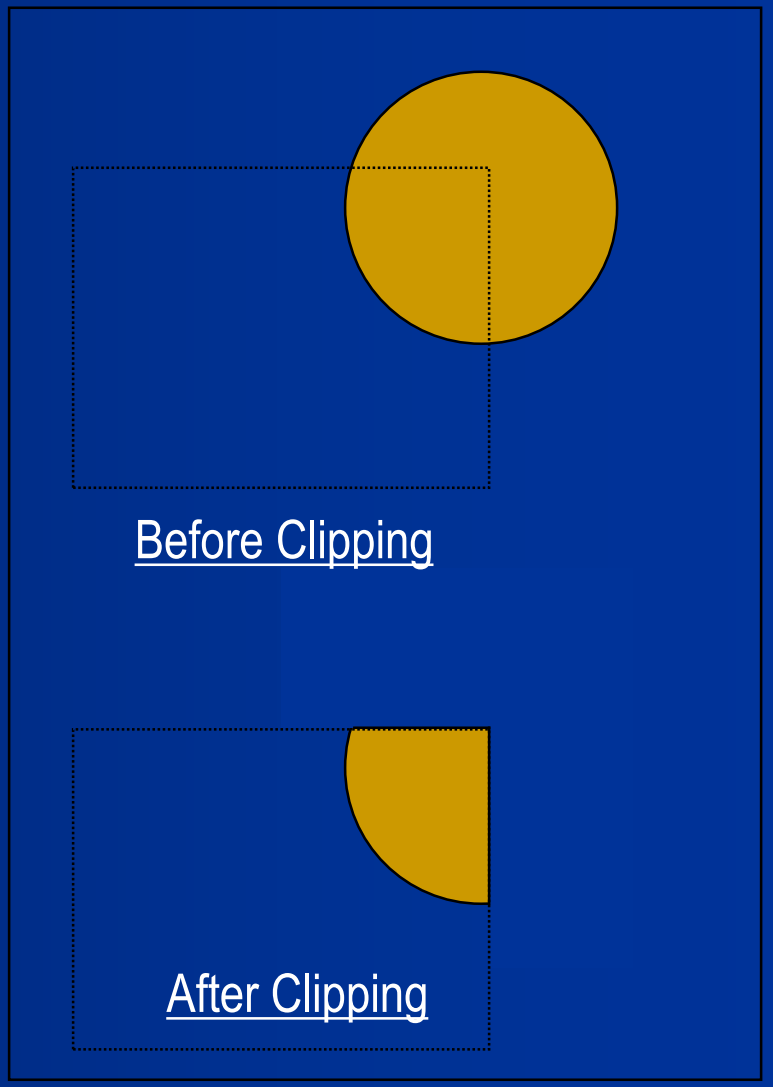
- Areas with curved boundaries can be clipped with methods similar to those discussed in the previous sections. If the objects are approximated with straight-line boundary sections, we use a polygon-clipping method. Otherwise, the clipping procedures involve nonlinear equations, and this requires more processing than for objects with linear boundaries.

Curve Clipping

- We can first test the coordinate extents of an object against the clipping boundaries to determine whether it is possible to accept or reject the entire object trivially. If not, we could check for object symmetries that we might be able to exploit in the initial accept/reject tests. For example, circles have symmetries between quadrants and octants, so we could check the coordinate extents of these individual circle regions.

Curve Clipping

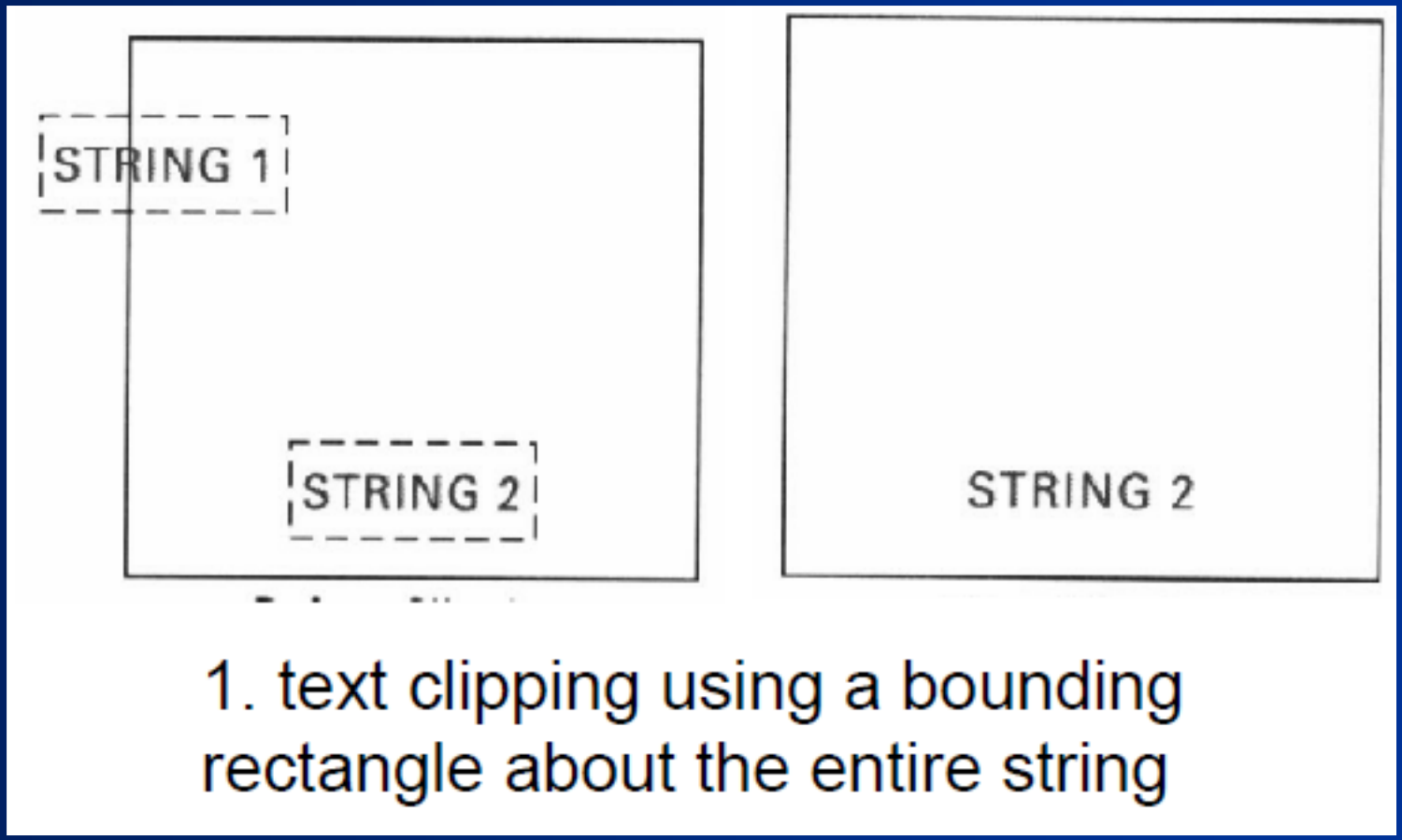
- An intersection calculation involves substituting a clipping-boundary position **(*xw*min, *xw*max, *yw*min, or *yw*max)** in the nonlinear equation for the object boundary and solving for the other coordinate value. Once all intersection positions have been evaluated, the defining positions for the object can be stored for later use by the scan-line fill procedures.



Text Clipping

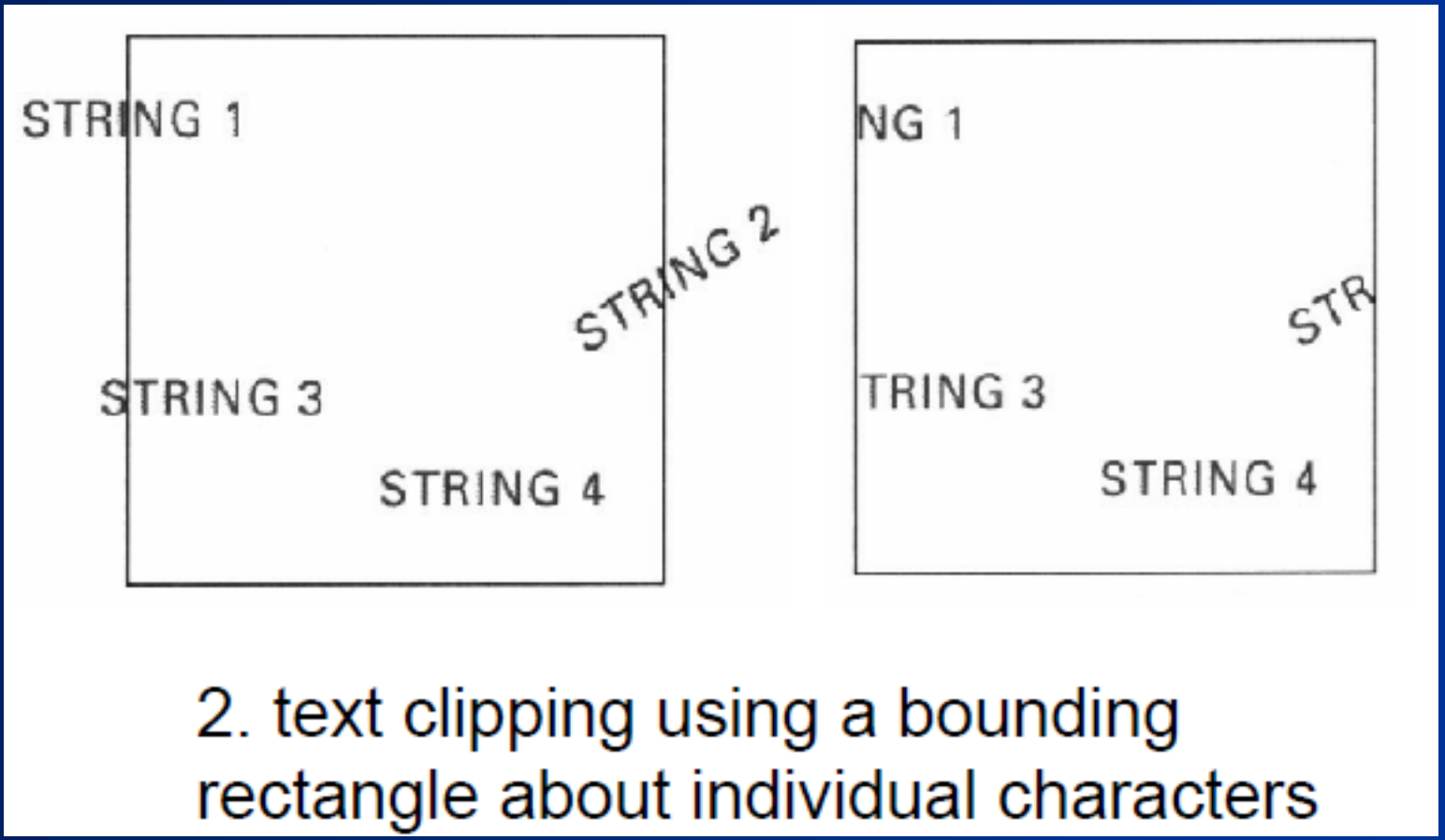
- All-or-none text clipping
  - Using boundary box for the entire text
- All-or-non character clipping
  - Using boundary box for each individual character
- Character clipping
  - Vector font: Clip boundary polygons or curves
  - Bitmap font: Clip individual pixels

Text Clipping (1)



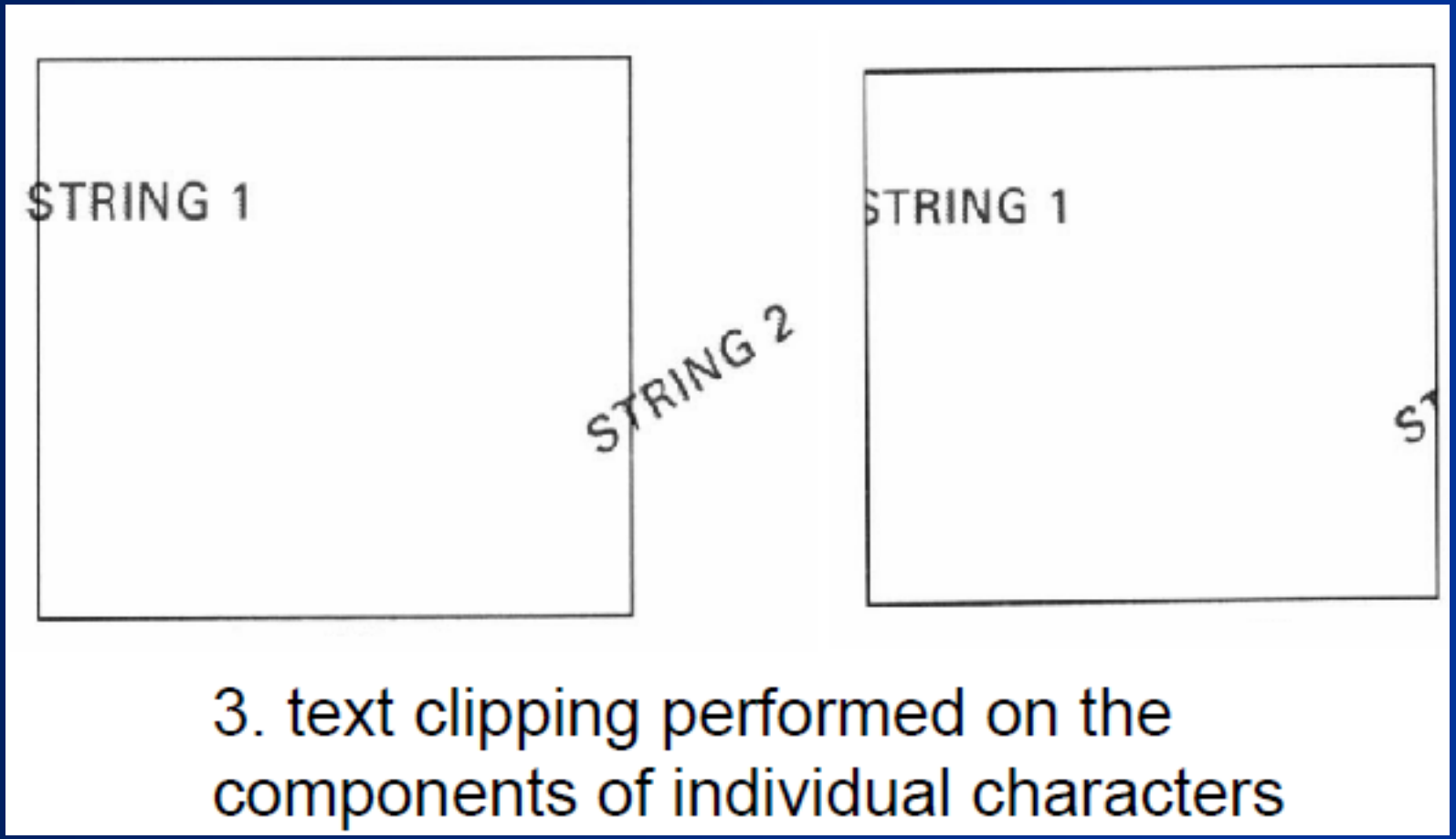
1. text clipping using a bounding rectangle about the entire string

Text Clipping (2)



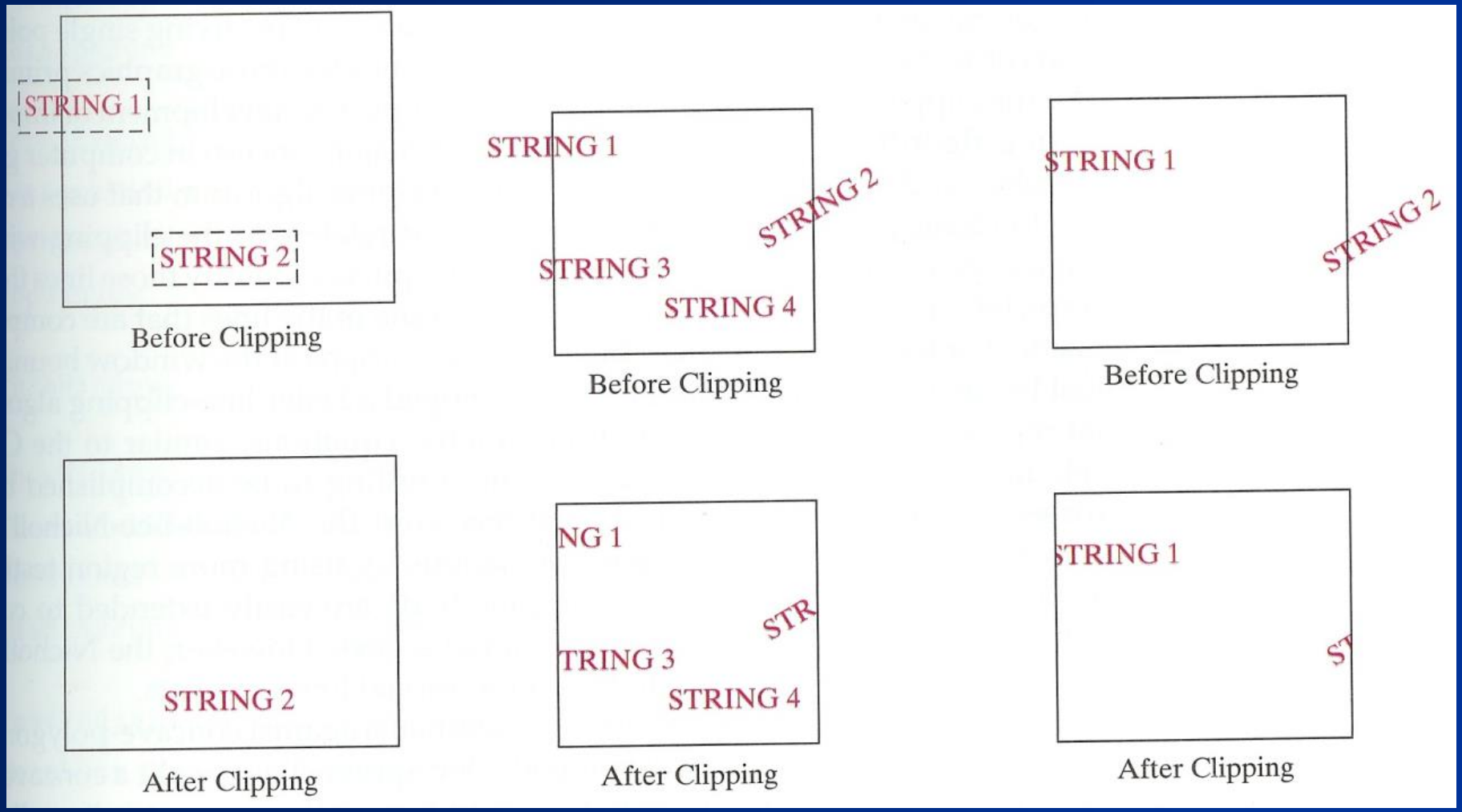
2. text clipping using a bounding rectangle about individual characters

Text Clipping (3)



3. text clipping performed on the components of individual characters

Text Clipping



3D Modeling Transformations

Henry Ford Int'l College, Kalanki, Kathmandu
By: Hari Prashad Pant

Three Dimensional Modeling Transformations

- Methods for object modeling transformation in three dimensions are extended from two dimensional methods by including consideration for the z coordinate.

Three Dimensional Modeling Transformations

- Generalize from 2D by including **z** coordinate
- Straightforward for translation and scale, rotation more difficult
- Homogeneous coordinates: 4 components
- Transformation matrices: 4×4 elements

3D Point

- We will consider points as column vectors. Thus, a typical point with coordinates (x, y, z) is represented as:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

3D Point Homogenous Coordinate

- A 3D point **P** is represented in homogeneous coordinates by a 4-dim. Vect:

$$\mathbf{P} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

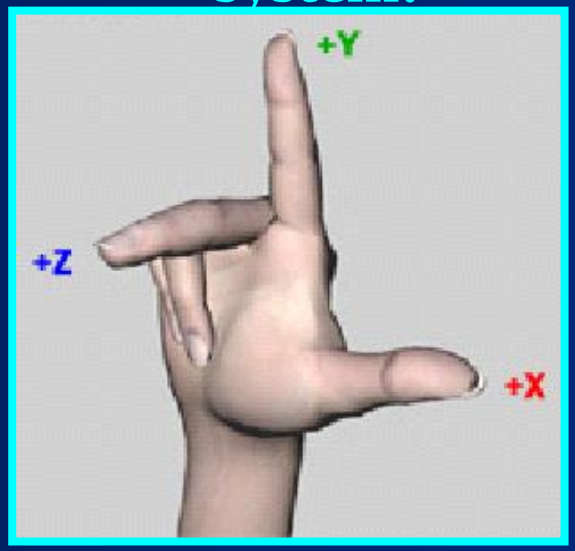
3D Point Homogenous Coordinate

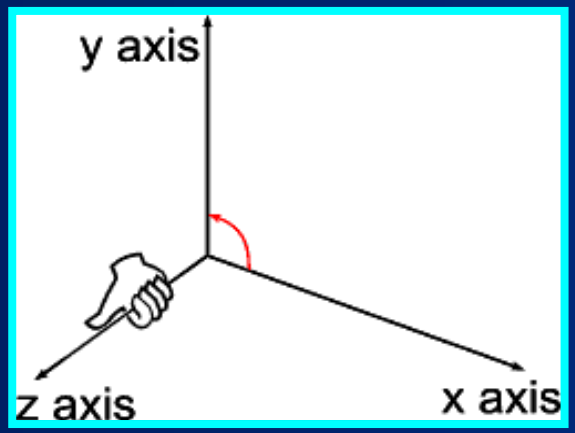
- We don't lose anything
- The main advantage: it is easier to compose translation and rotation
- Everything is matrix multiplication

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

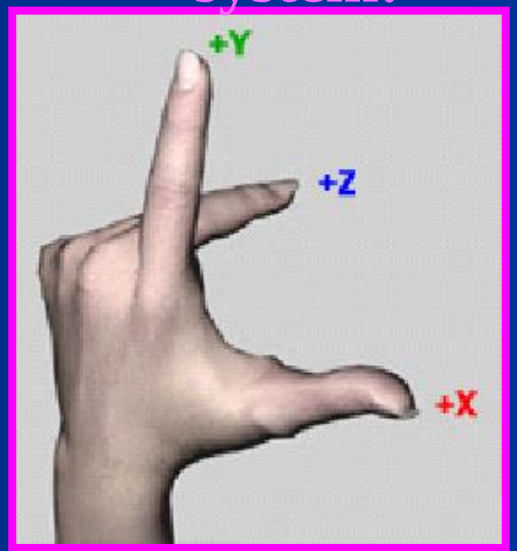
3D Coordinate Systems

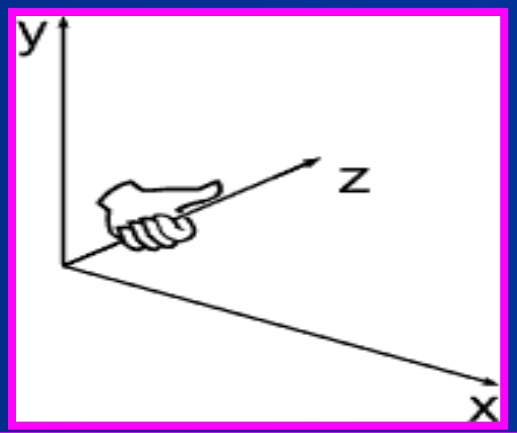
- ***Right Hand*** **coordinate system:**





- ***Left Hand*** **coordinate system:**





3D Transformation

- In homogeneous coordinates, 3D transformations are represented by 4×4 matrixes:

$$\begin{bmatrix} a & b & c & t_x \\ d & e & f & t_y \\ g & h & i & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3D Translation

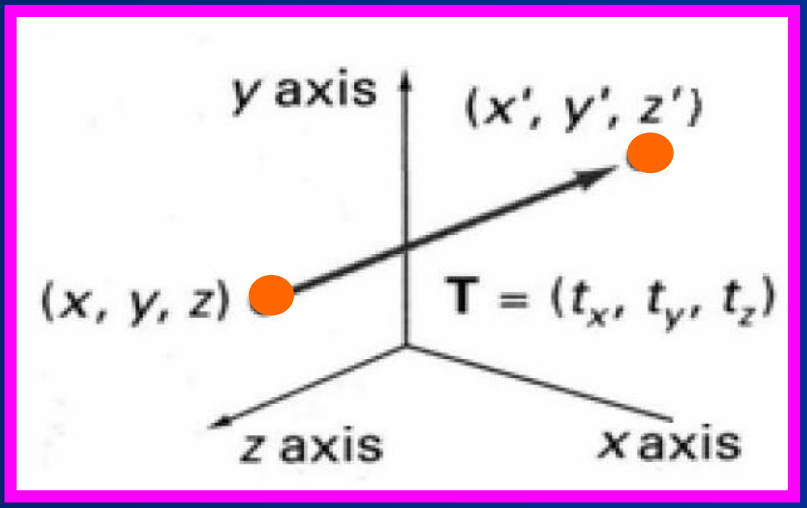
Henry Ford Int'l College, Kalanki, Kathmandu
By: Hari Prashad Pant

3D Translation

- **P** is **translated** to **P'** by:

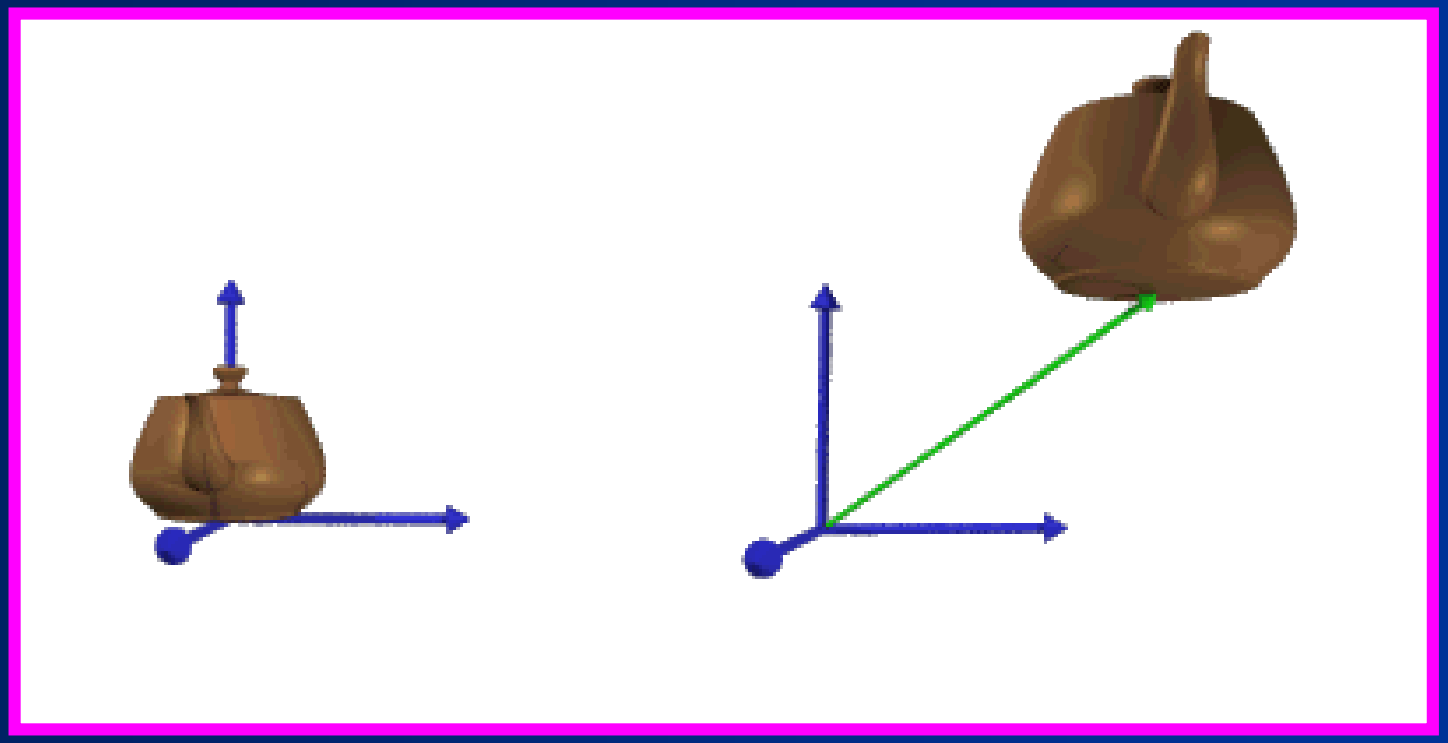
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

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



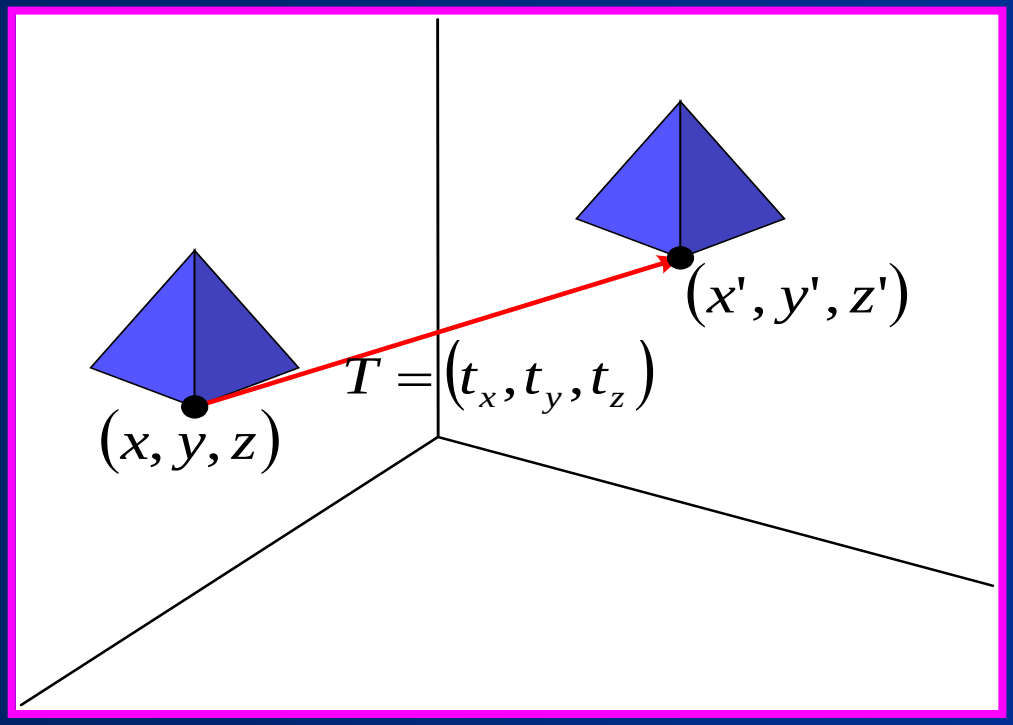
3D Translation

- An object is translated in 3D space by transforming each of the defining points of the objects .



3D Translation

- An Object represented as a set of polygon surfaces, is translated by translating each vertex of each surface and redraw the polygon facets in the new position.



- **Inverse Translation:** $T^{-1}(t_x, t_y, t_z) = T(-t_x, -t_y, -t_z)$

3D Rotation

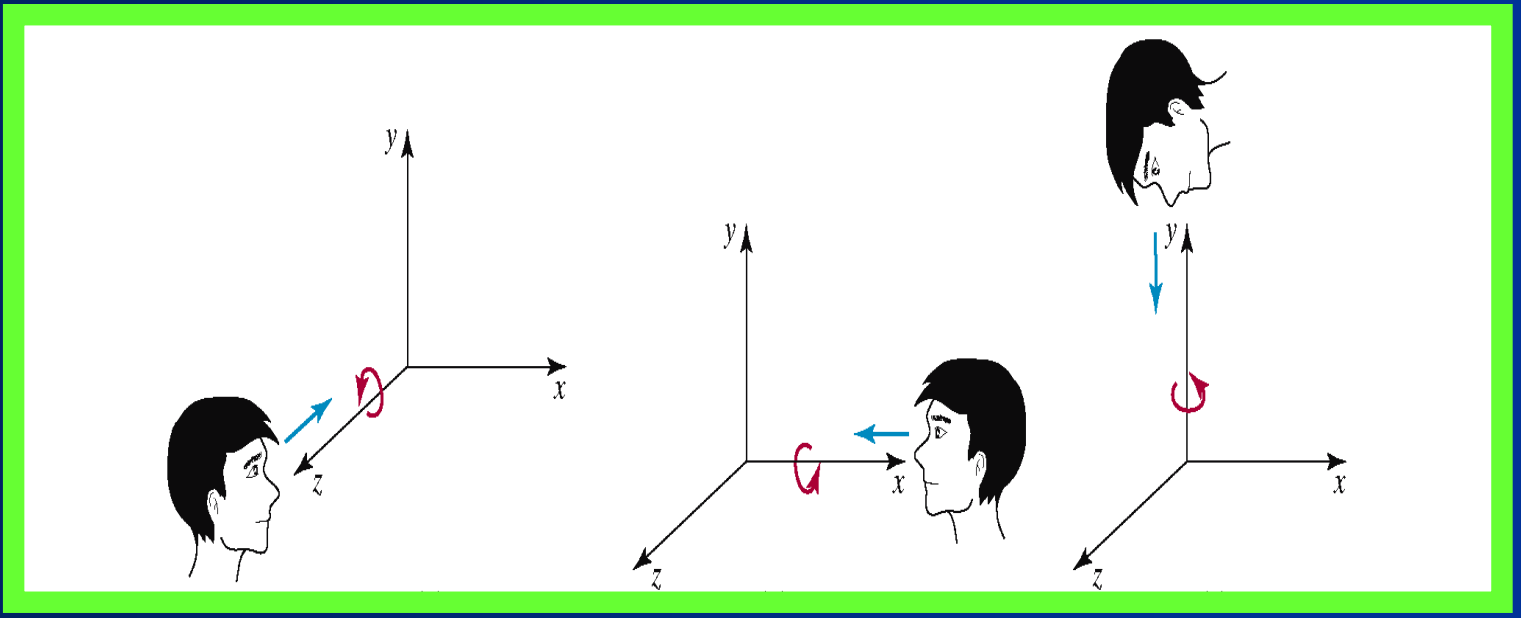
Henry Ford Int'l College, Kalanki, Kathmandu
By: Hari Prashad Pant

3D Rotation

- In general, rotations are specified by a ***rotation axis*** and an ***angle***. In two-dimensions there is only one choice of a rotation axis that leaves points in the plane.

3D Rotation

- The easiest **rotation axes** are those that parallel to the coordinate axis.
- Positive rotation **angles** produce counterclockwise rotations about a coordinate axis, if we are looking along the positive half of the axis toward the coordinate origin.



Coordinate Axis Rotations

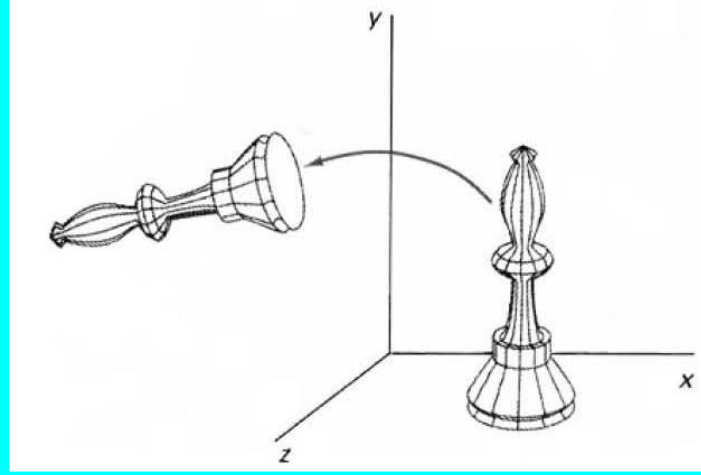
Henry Ford Int'l College, Kalanki, Kathmandu
By: Hari Prashad Pant

Coordinate Axis Rotations

- **Z-axis rotation:** For z axis same as 2D rotation:

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

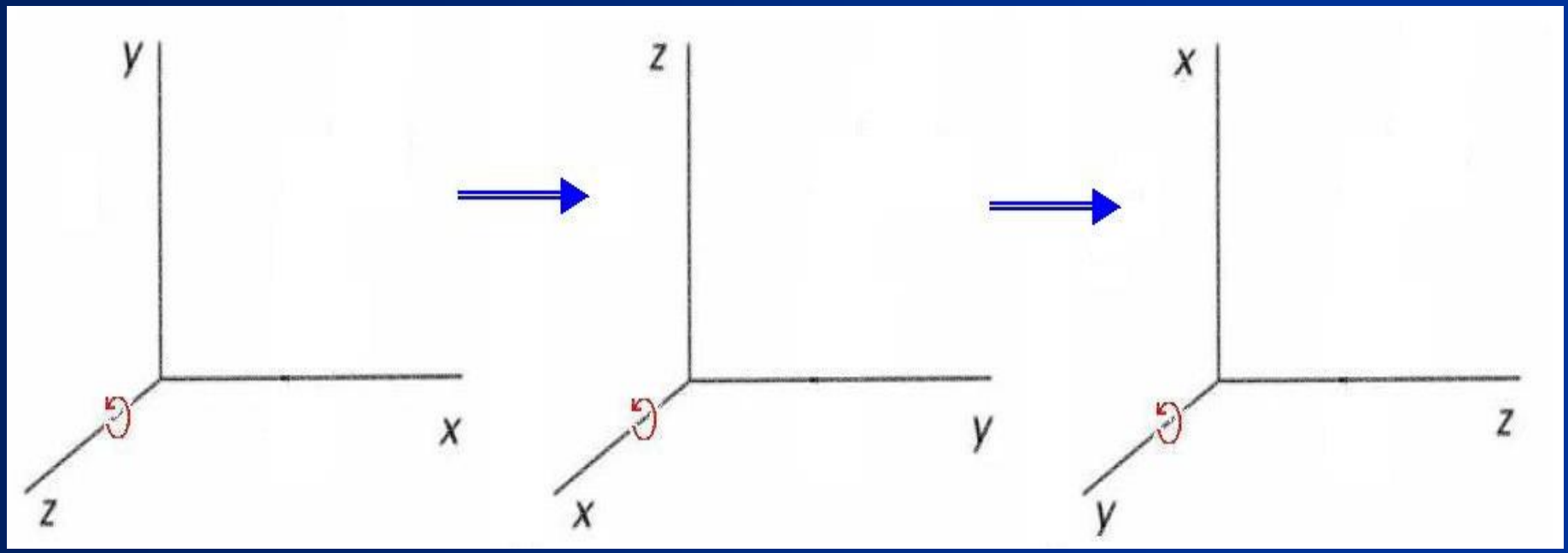
$$\mathbf{P}' = \mathbf{R}_z(\theta) \cdot \mathbf{P}$$



Coordinate Axis Rotations

- Obtain rotations around other axes through cyclic permutation of coordinate parameters:

$$x \rightarrow y \rightarrow z \rightarrow x$$

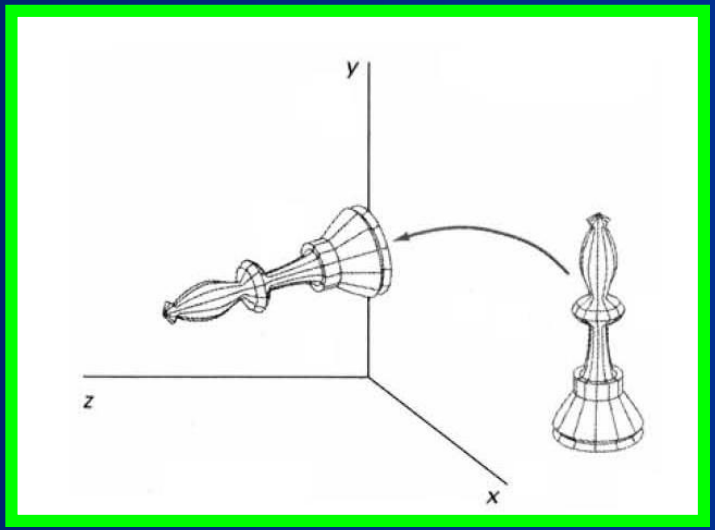


Coordinate Axis Rotations

- **X-axis rotation:**

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

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

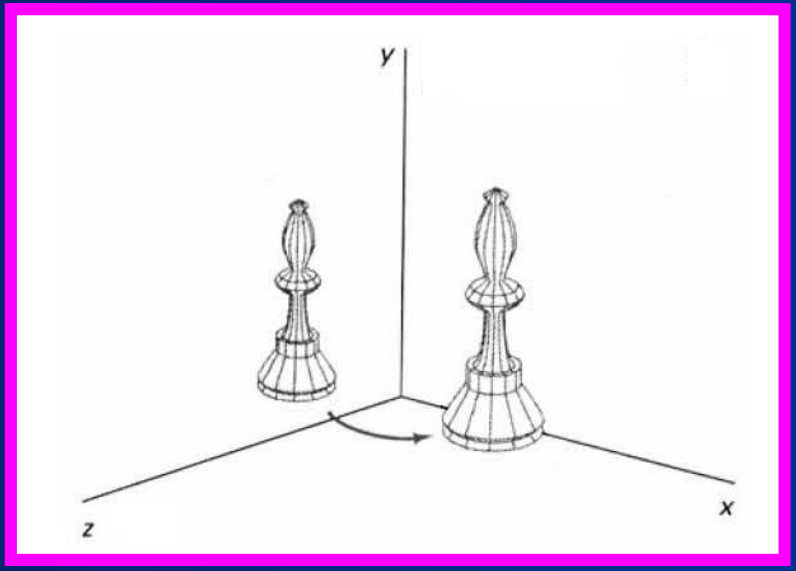


Coordinate Axis Rotations

- **Y-axis rotation:**

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

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

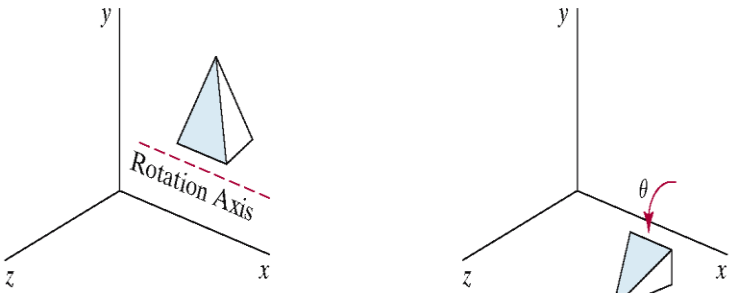


General Three Dimensional Rotations

Henry Ford Int'l College, Kalanki, Kathmandu
By: Hari Prashad Pant

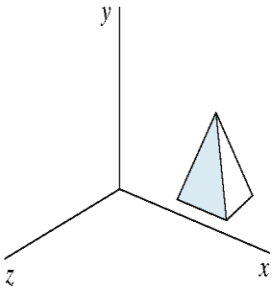
General Three Dimensional Rotations

Rotation axis parallel with coordinate axis (Example x axis):

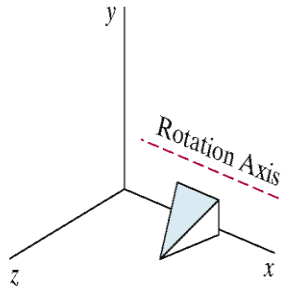


(a)
Original Position of Object

(c)
Rotate Object Through Angle θ



(b)
Translate Rotation Axis onto x Axis



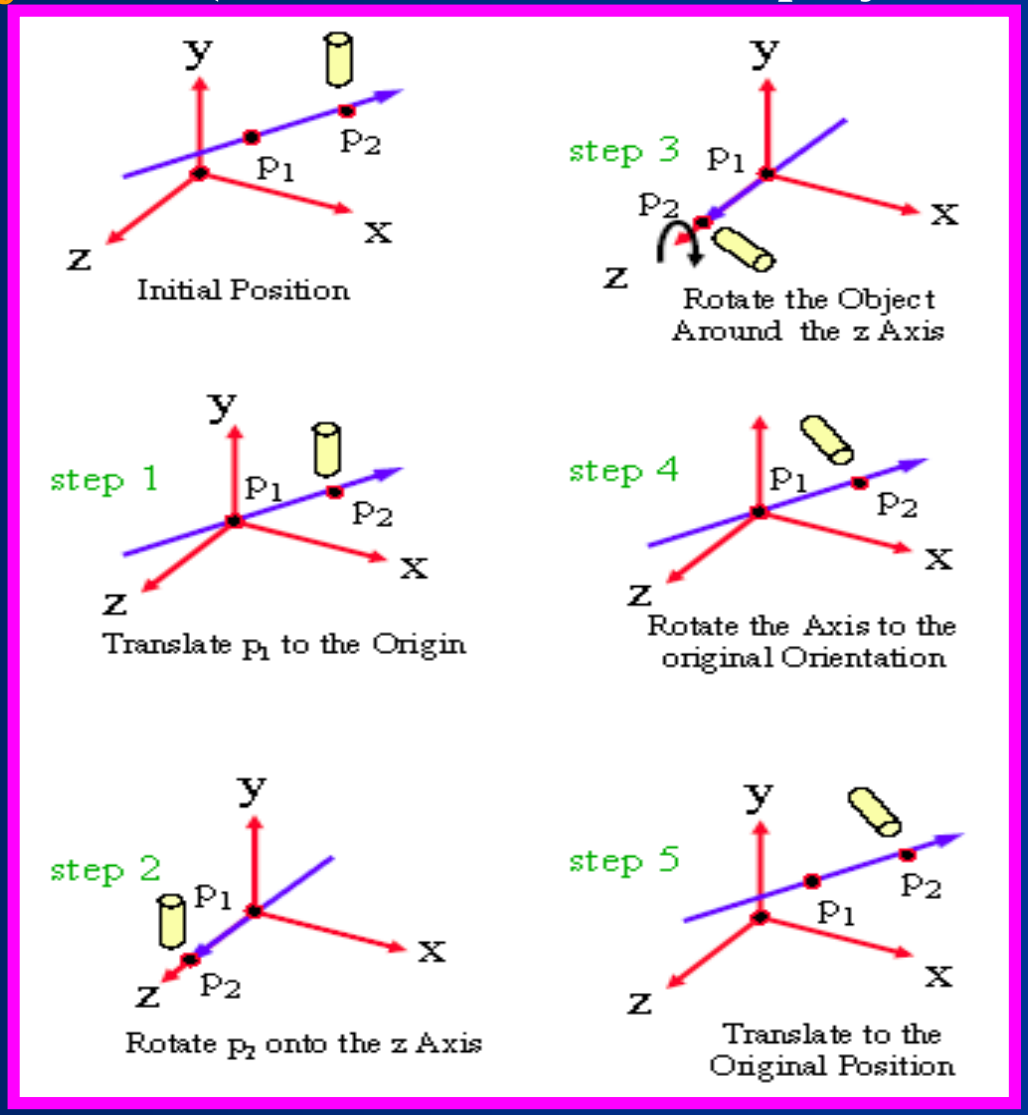
(d)
Translate Rotation
Axis to Original Position

$$\mathbf{P}' = \mathbf{T}^{-1} \cdot \mathbf{R}_x(\theta) \cdot \mathbf{T} \cdot \mathbf{P}$$

Henry Ford Int'l College, Kalanki, Kathmandu
By: Hari Prashad Pant

General Three Dimensional Rotations

An arbitrary axis (with the rotation axis projected onto the **Z** axis):



$$\mathbf{R}(\theta) = \mathbf{T}^{-1} \cdot \mathbf{R}_x^{-1}(\alpha) \cdot \mathbf{R}_y^{-1}(\beta) \cdot \mathbf{R}_z(\theta) \cdot \mathbf{R}_y(\beta) \cdot \mathbf{R}_x(\alpha) \cdot \mathbf{T}$$

Henry Ford Int'l College, Kalanki, Kathmandu

By: Hari Prashad Pant

General Three Dimensional Rotations

$$\mathbf{R}(\theta) = \mathbf{T}^{-1} \cdot \mathbf{R}_x^{-1}(\alpha) \cdot \mathbf{R}_y^{-1}(\beta) \cdot \mathbf{R}_z(\theta) \cdot \mathbf{R}_y(\beta) \cdot \mathbf{R}_x(\alpha) \cdot \mathbf{T}$$

- A rotation matrix for any axis that does not coincide with a coordinate axis can be set up as a composite transformation involving combination of translations and the coordinate-axes rotations:

1. Translate the object so that the rotation axis passes through the coordinate origin
2. Rotate the object so that the axis rotation coincides with one of the coordinate axes
3. Perform the specified rotation about that coordinate axis
4. Apply inverse rotation axis back to its original orientation
5. Apply the inverse translation to bring the rotation axis back to its original position

3D Scaling

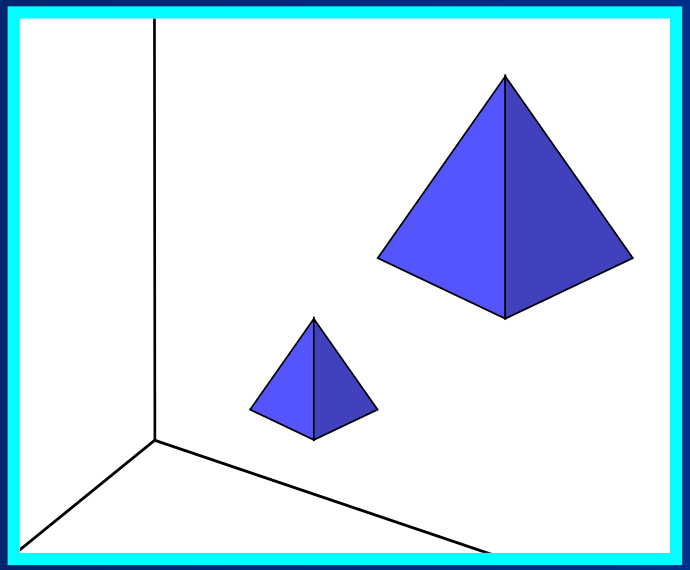
Henry Ford Int'l College, Kalanki, Kathmandu
By: Hari Prashad Pant

3D Scaling

- **About origin:** Changes the size of the object and repositions the object relative to the coordinate origin.

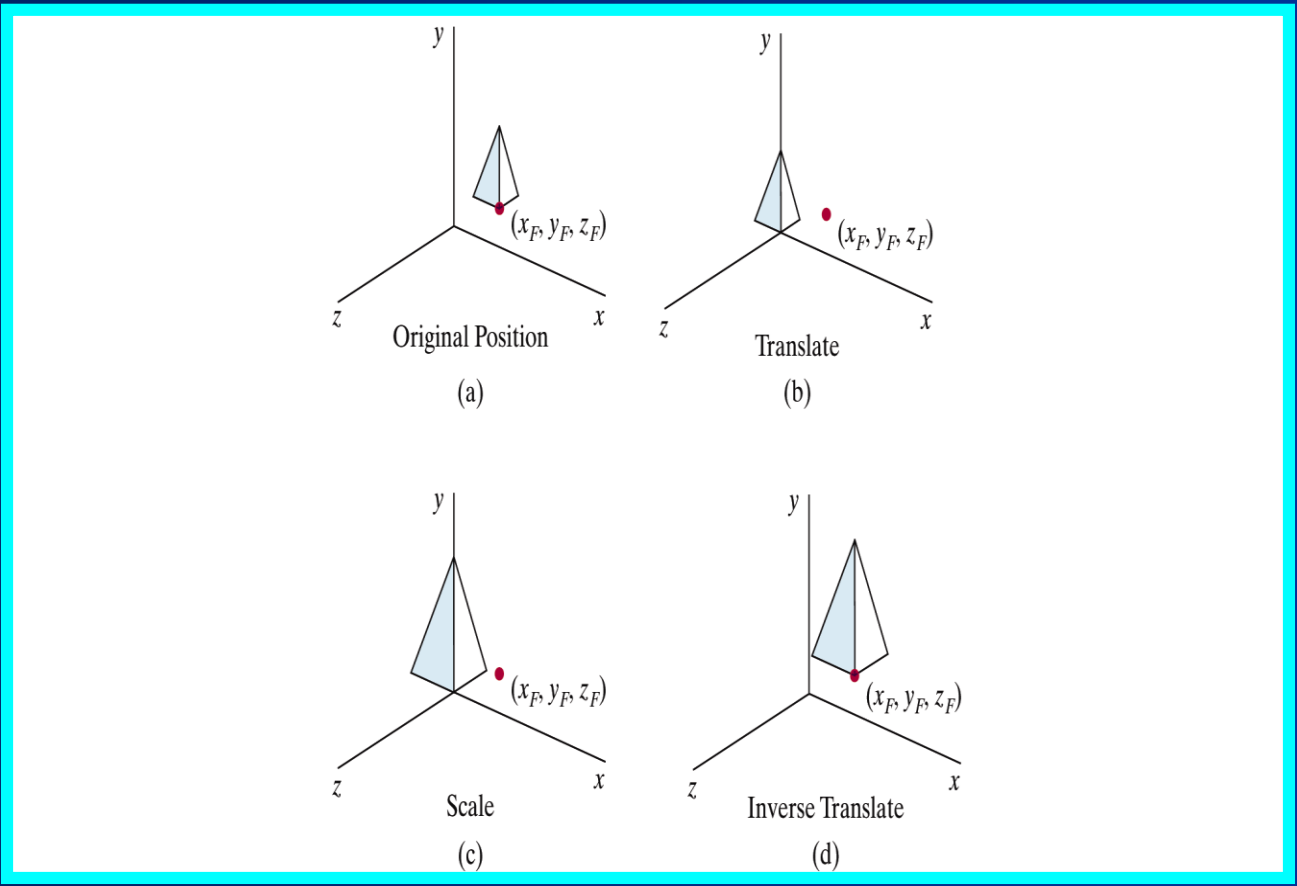
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

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



3D Scaling

- **About any fixed point:**



$$\mathbf{T}(x_f, y_f, z_f) \cdot \mathbf{S}(s_x, s_y, s_z) \cdot \mathbf{T}(-x_f, -y_f, -z_f) = \begin{bmatrix} s_x & 0 & 0 & (1-s_x)x_f \\ 0 & s_y & 0 & (1-s_y)y_f \\ 0 & 0 & s_z & (1-s_z)z_f \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Composite 3D Transformations

Henry Ford Int'l College, Kalanki, Kathmandu
By: Hari Prashad Pant

Composite 3D Transformations

- **Same way as in two dimensions:**
  - Multiply matrices
  - Rightmost term in matrix product is the first transformation to be applied

3D Reflections

Henry Ford Int'l College, Kalanki, Kathmandu
By: Hari Prashad Pant

3D Reflections

- **About an axis:** equivalent to 180˚rotation about that axis

3D Reflections

About a plane:

- A reflection through the **xy** plane:

$$\begin{bmatrix} x \\ y \\ -z \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

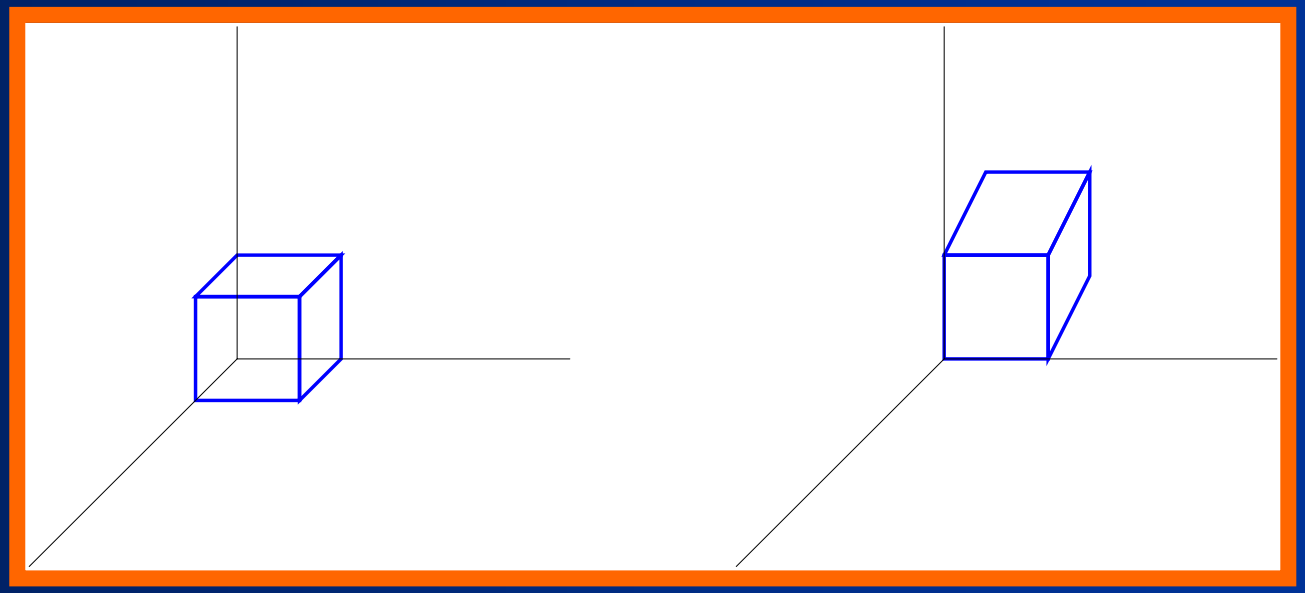
- A reflections through the **xz** and the **yz** planes are defined similarly.

3D Shearing

Henry Ford Int'l College, Kalanki, Kathmandu
By: Hari Prashad Pant

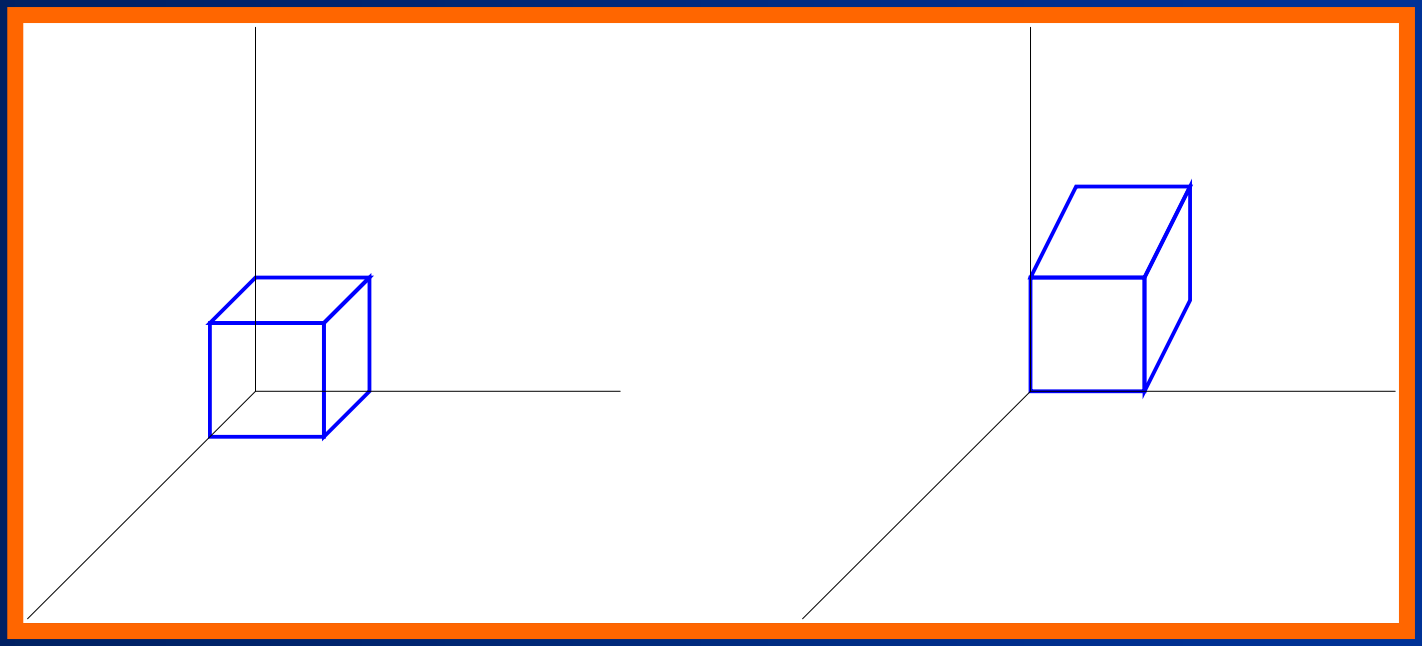
3D Shearing

- **Modify object shapes**
- **Useful for perspective projections:**
  - E.g. draw a cube (3D) on a screen (2D)
  - Alter the values for **x** and **y** by an amount proportional to the distance from z_{ref}



3D Shearing

$$M_{zshear} = \begin{bmatrix} 1 & 0 & sh_{zx} & -sh_{zx} \cdot z_{ref} \\ 0 & 1 & sh_{zy} & -sh_{zy} \cdot z_{ref} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

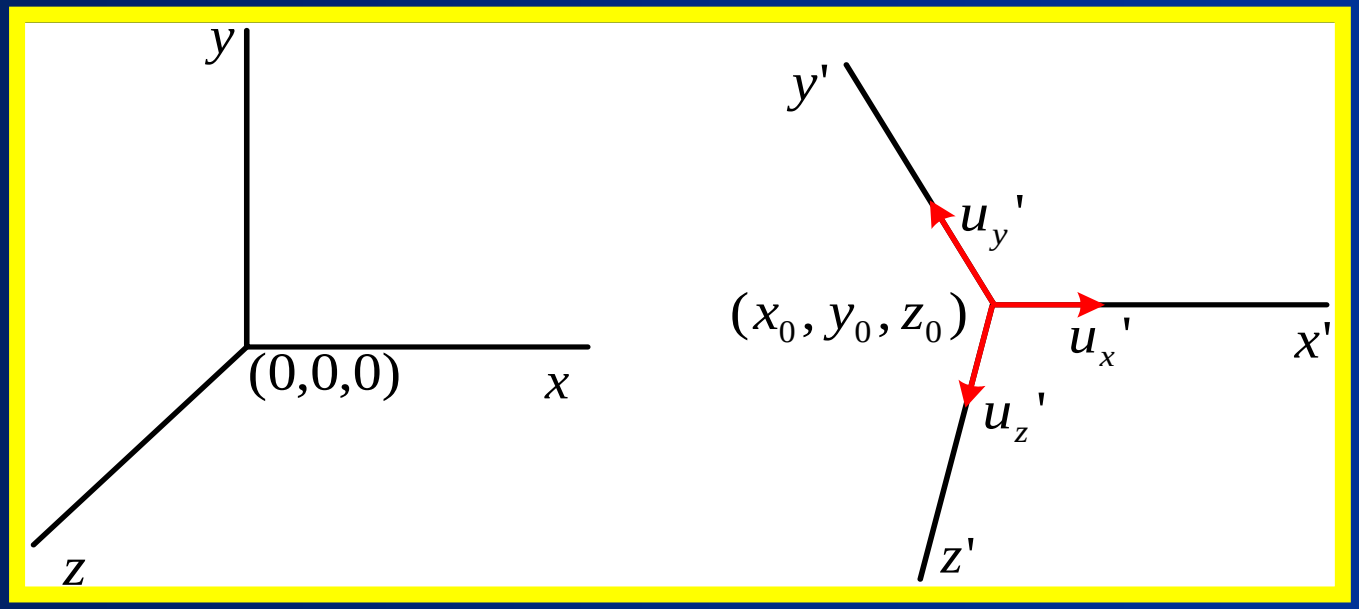


Transformations Between Coordinate Systems

Henry Ford Int'l College, Kalanki, Kathmandu
By: Hari Prashad Pant

Transformations Between Coordinate ystems

- Translate such that the origins overlap
- Rotate such that the axes overlap



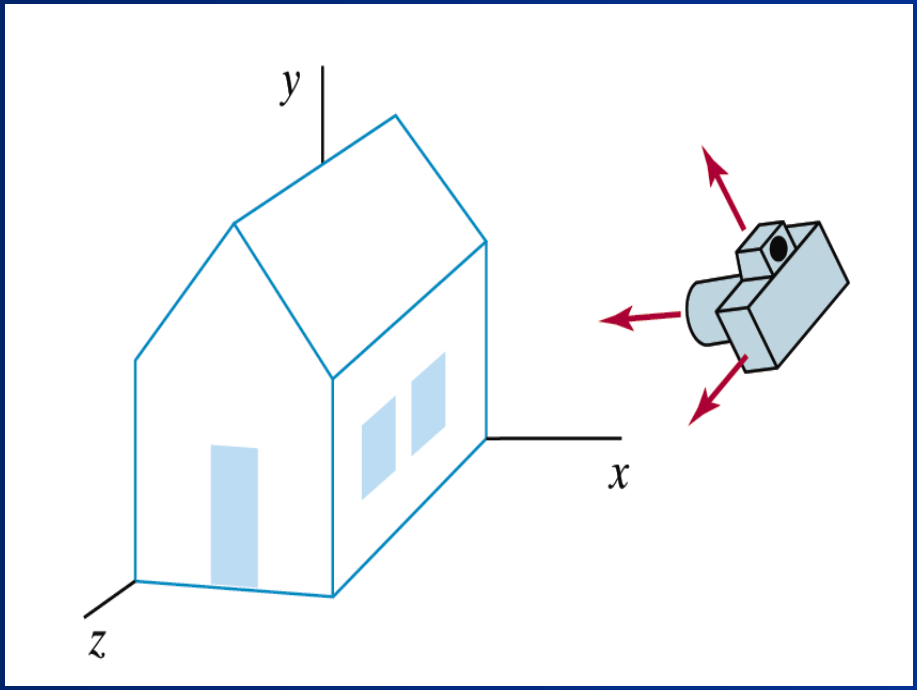
$$R \cdot T = \begin{bmatrix} u'_{x1} & u'_{x2} & u'_{x3} & 0 \\ u'_{y1} & u'_{y2} & u'_{y3} & 0 \\ u'_{z1} & u'_{z2} & u'_{z3} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot T(-x_0, -y_0, -z_0)$$

Three Dimensional Viewing

Henry Ford Int'l College, Kalanki, Kathmandu
By: Hari Prashad Pant

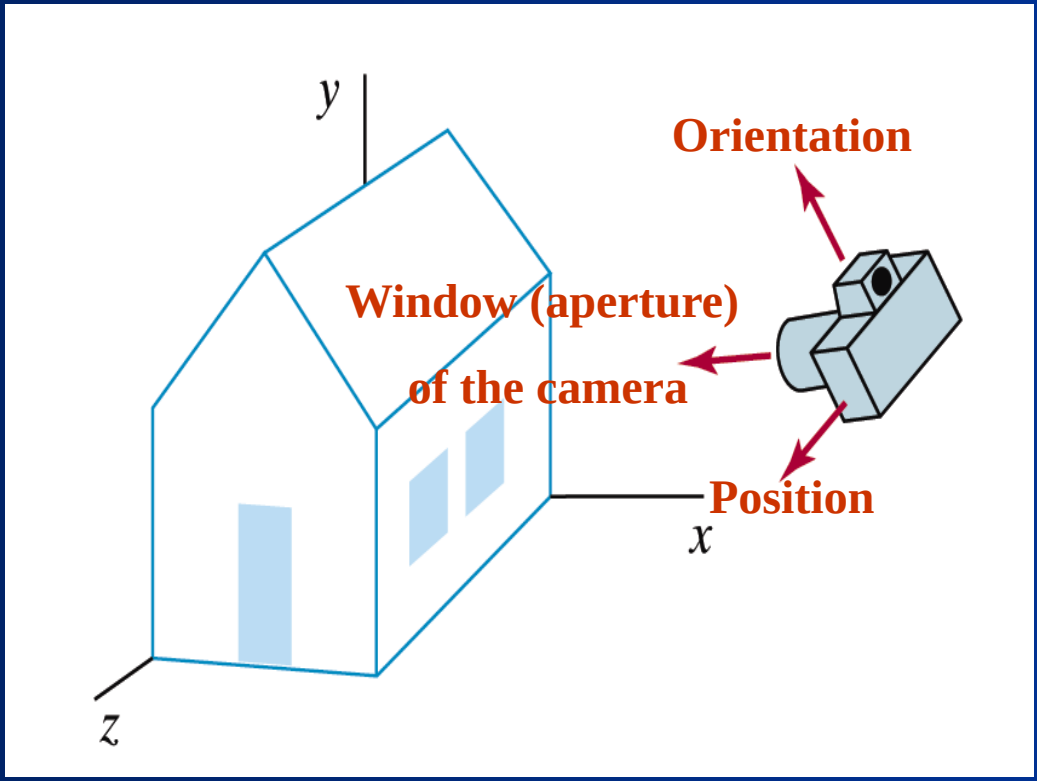
3D Viewing

- The steps for computer generation of a view of a three dimensional scene are somewhat analogous to the processes involved in taking a photograph.



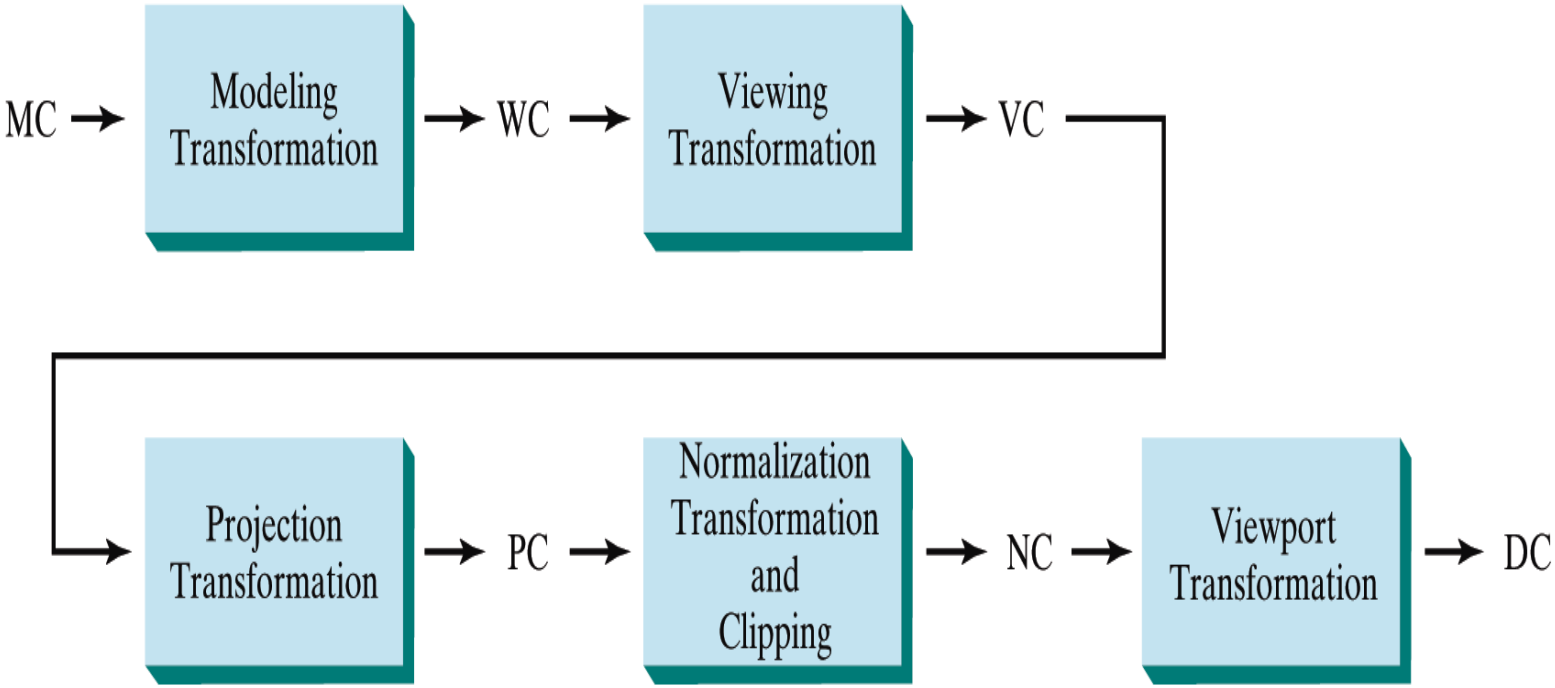
Camera Analogy

1. Viewing position
2. Camera orientation
3. Size of clipping window



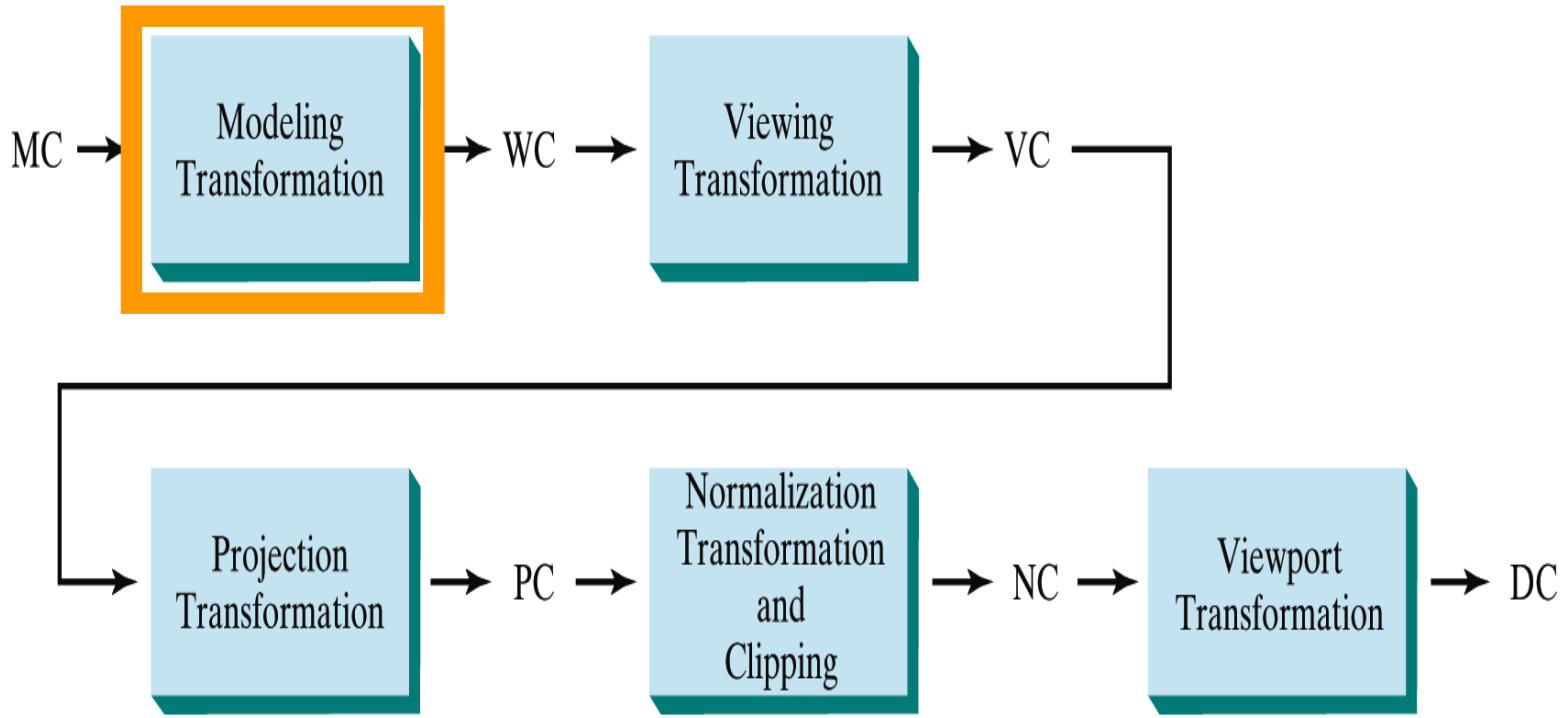
Viewing Pipeline

- The general processing steps for modeling and converting a world coordinate description of a scene to device coordinates:



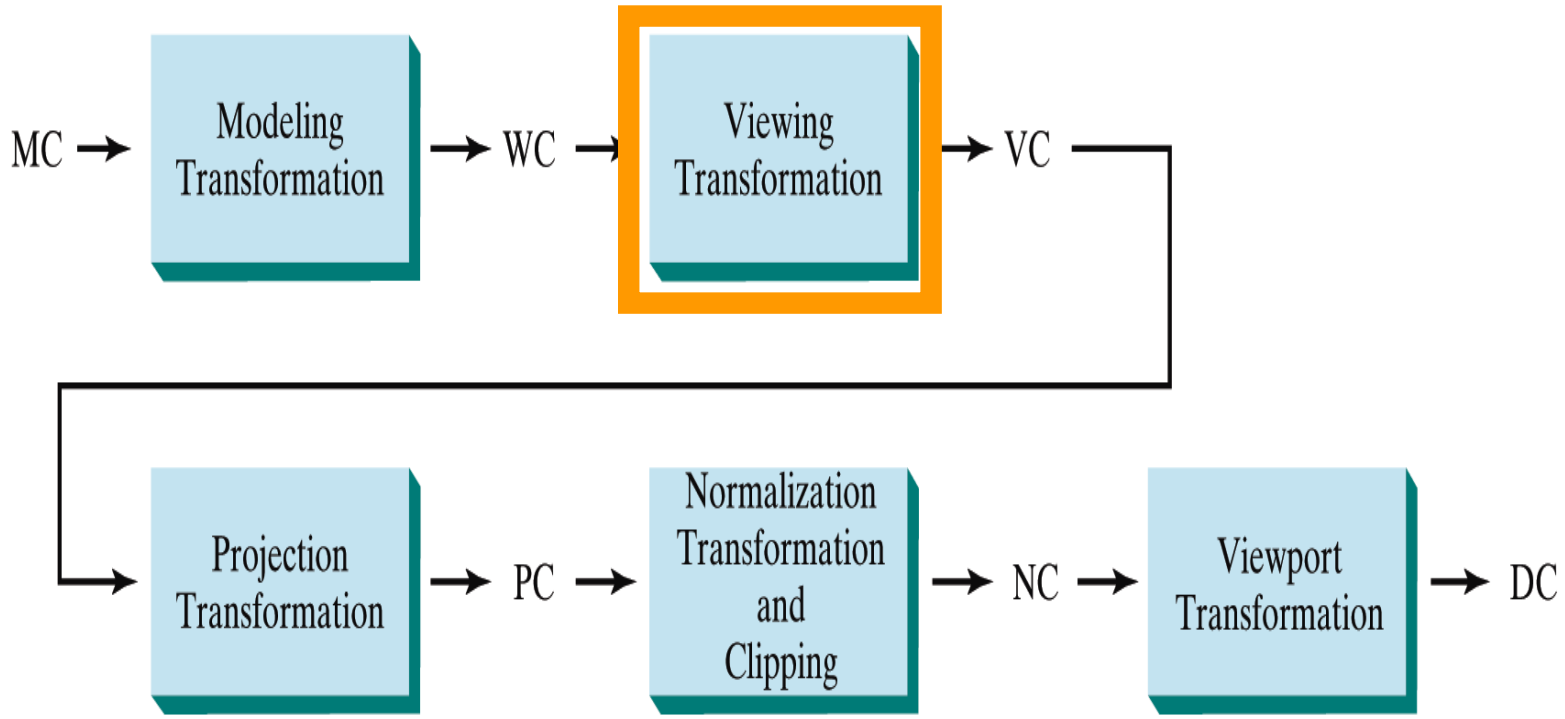
Viewing Pipeline

1. Construct the shape of individual objects in a scene within modeling coordinate, and place the objects into appropriate positions within the scene (world coordinate).



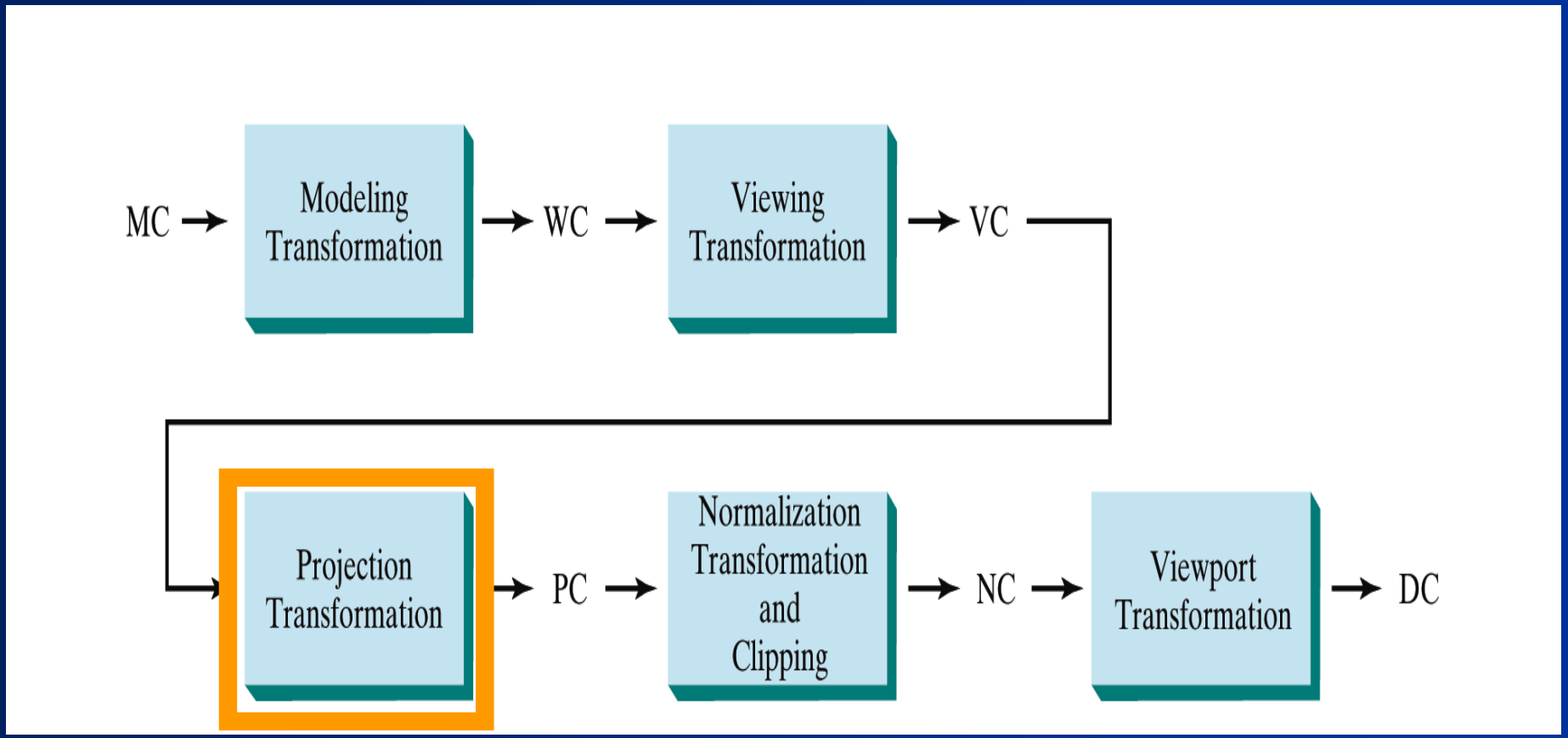
Viewing Pipeline

2. World coordinate positions are converted to viewing coordinates.



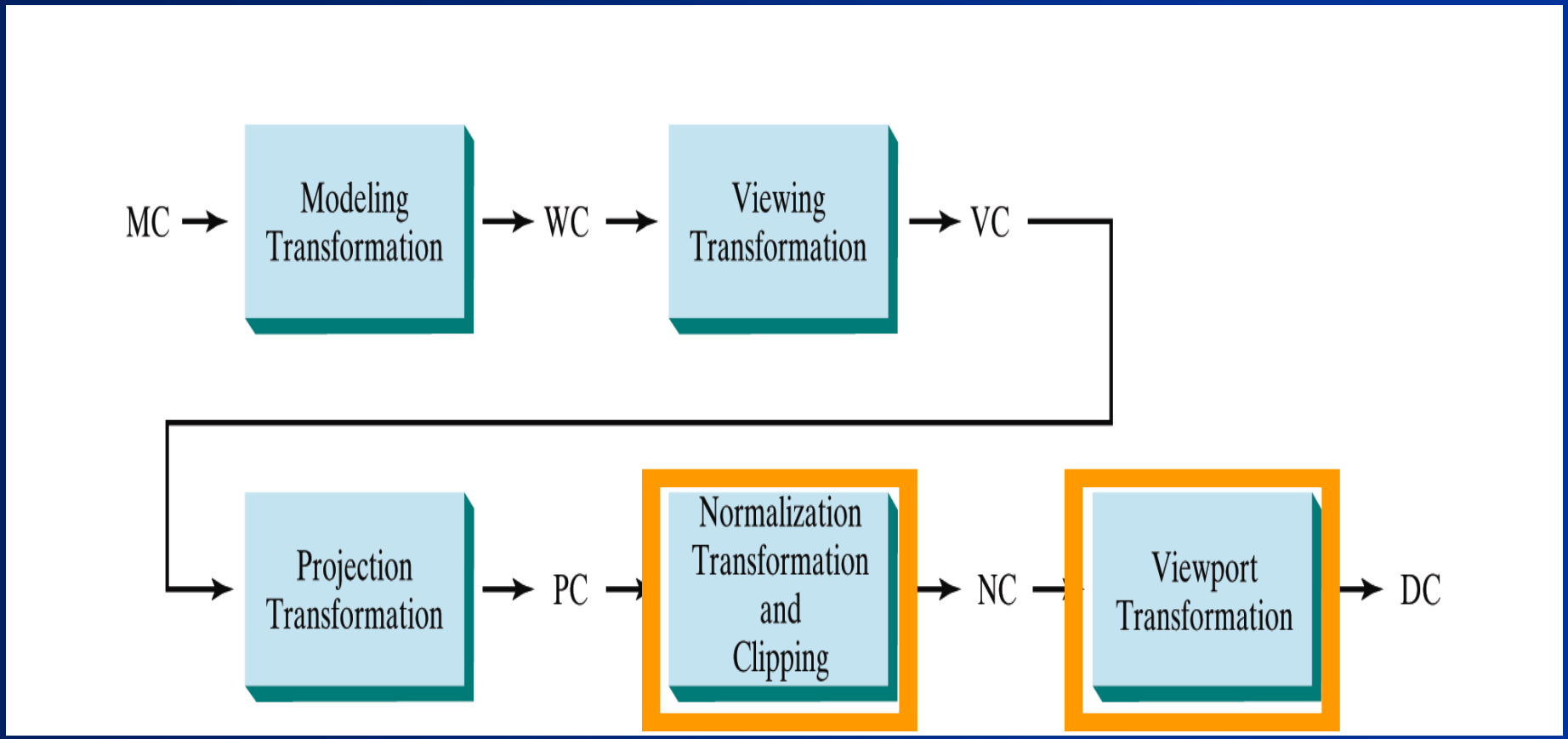
Viewing Pipeline

3. Convert the viewing coordinate description of the scene to coordinate positions on the projection plane.

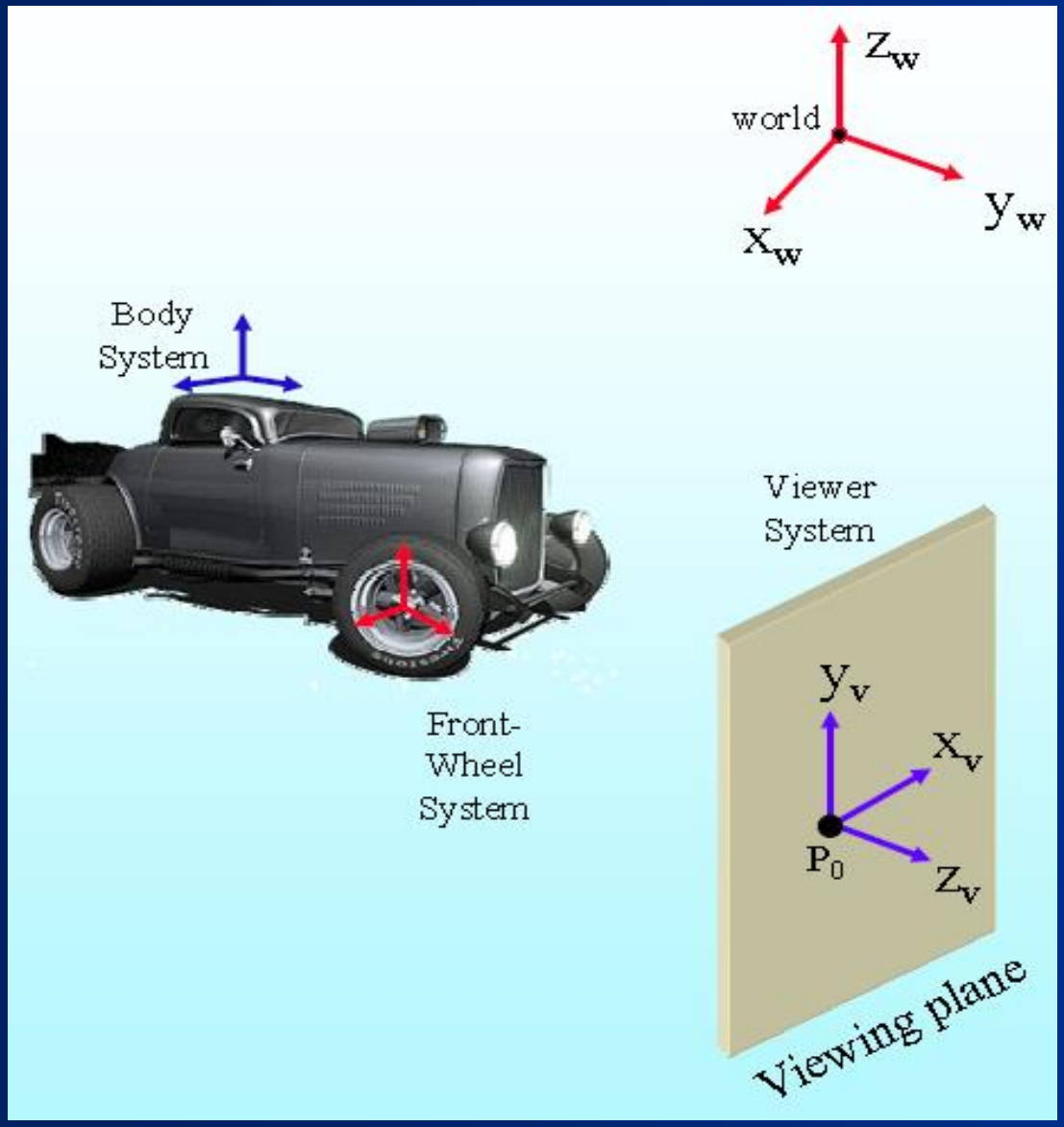


Viewing Pipeline

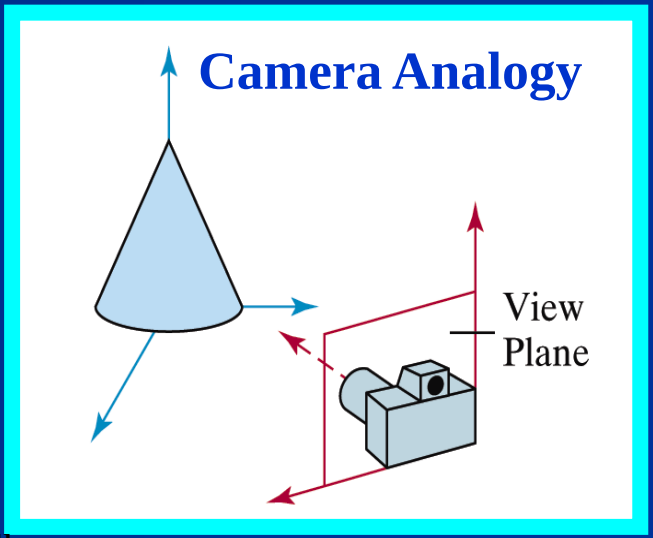
4. Positions on the projection plane, will then mapped to the Normalized coordinate and output device.



Viewing Coordinates

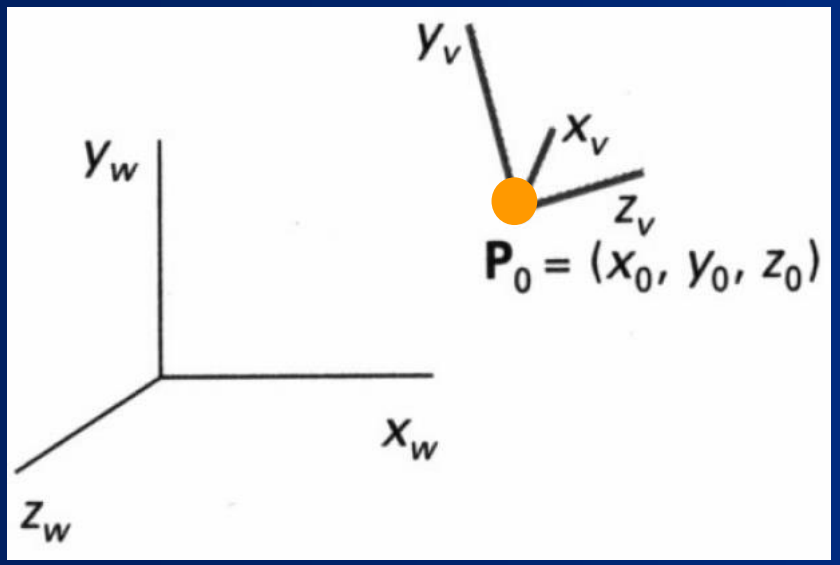


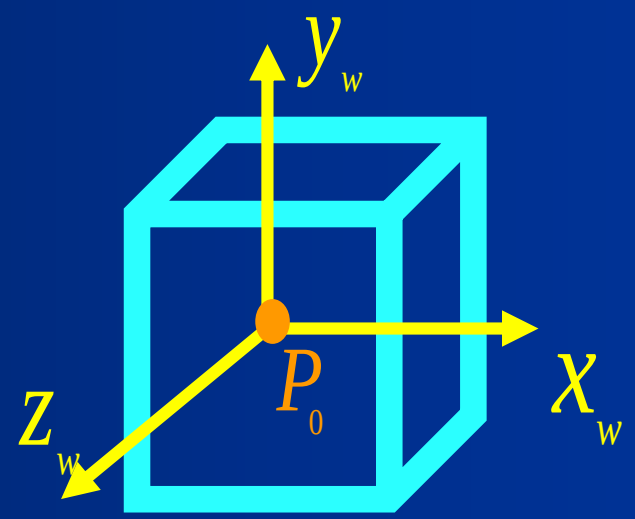
- Viewing coordinates system describes 3D objects with respect to a viewer.
- A Viewing (Projector) plane is set up perpendicular to z_v and aligned with (x_v,y_v).

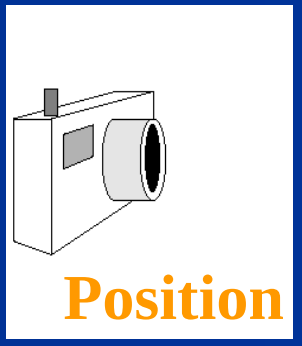


Specifying the Viewing Coordinate System (View Reference Point)

- We first pick a world coordinate position called **view reference point** (origin of our viewing coordinate system).
- P_0 is a point where a camera is located.
- The view reference point is often chosen to be close to or on the surface of some object, or at the center of a group of objects.

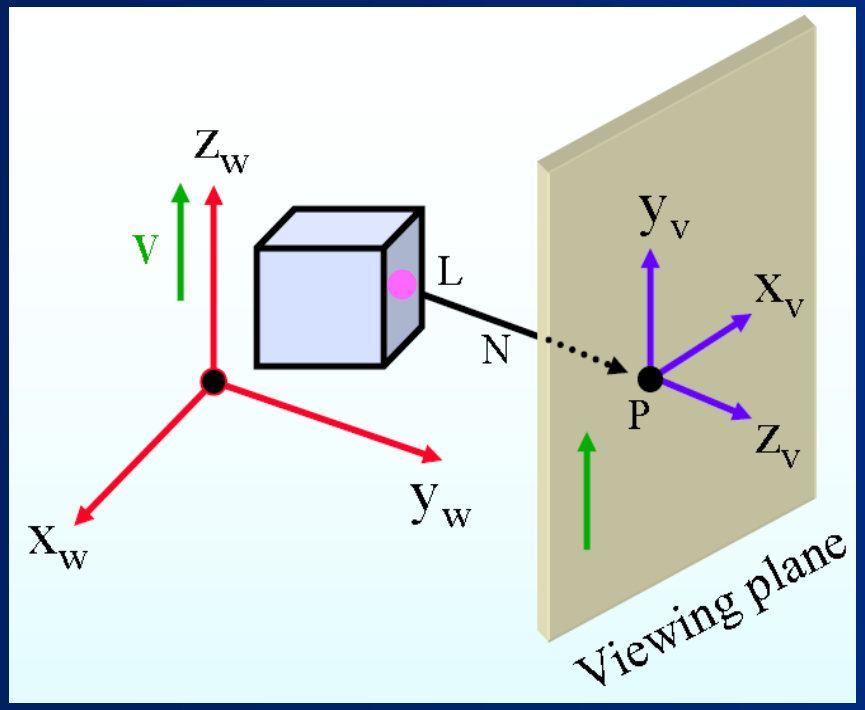


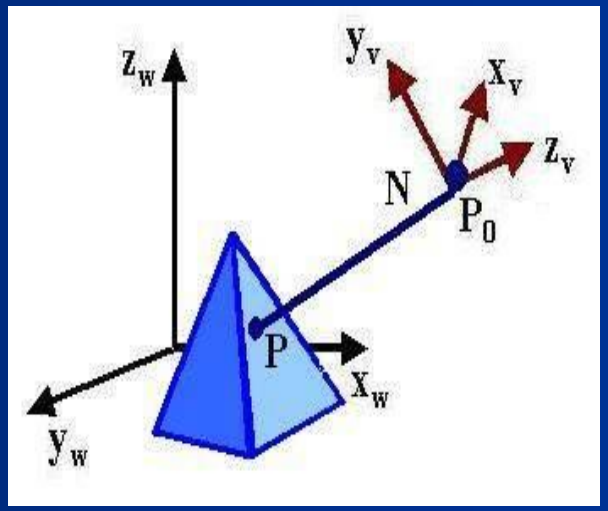


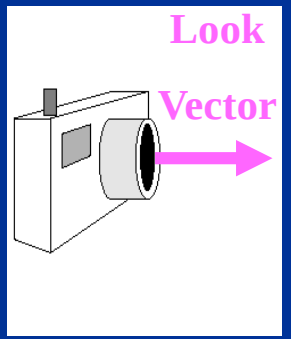


Specifying the Viewing Coordinate System (Z_v Axis)

- Next, we select the positive direction for the viewing z_v axis, by specifying the **view plane normal vector**, **N**.
- The direction of **N**, is from the **look at point** (L) to the view reference point.

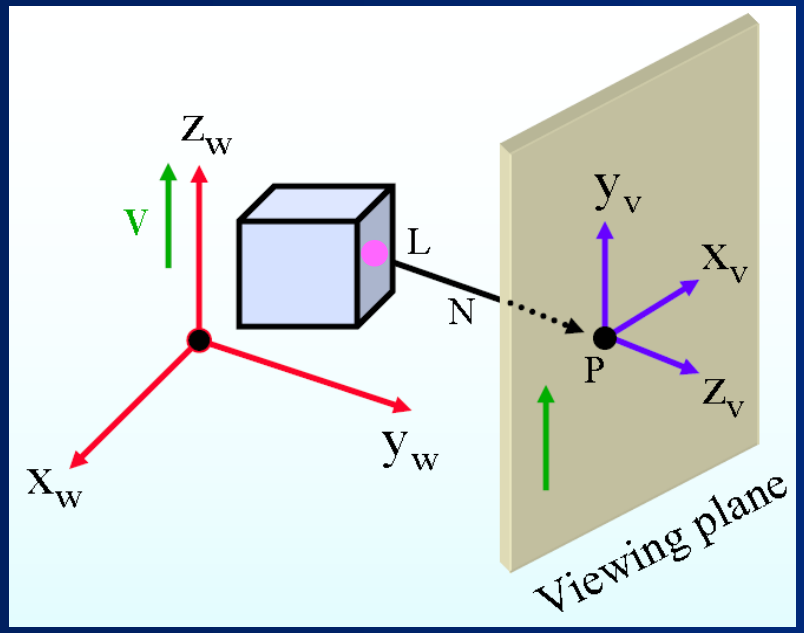


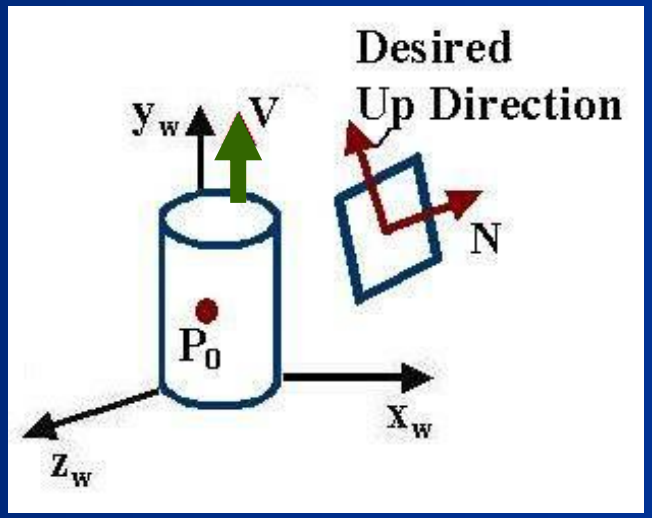


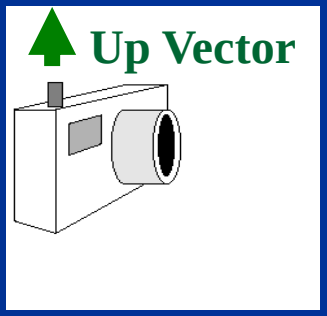


Specifying the Viewing Coordinate System (y_v Axis)

- Finally, we choose the ***up direction*** for the view by specifying a vector ***V***, called the ***view up vector***.
- This vector is used to establish the positive direction for the y_v axis.
- ***V*** is projected into a plane that is perpendicular to the normal vector.

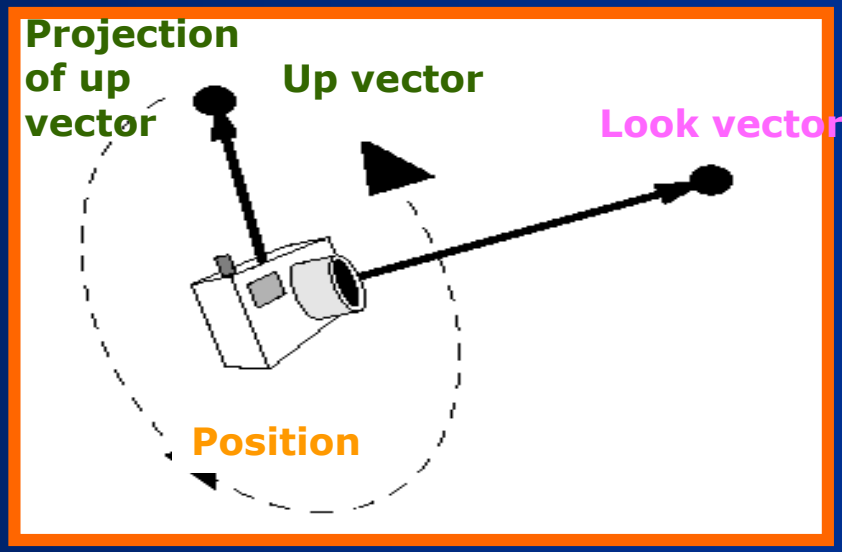






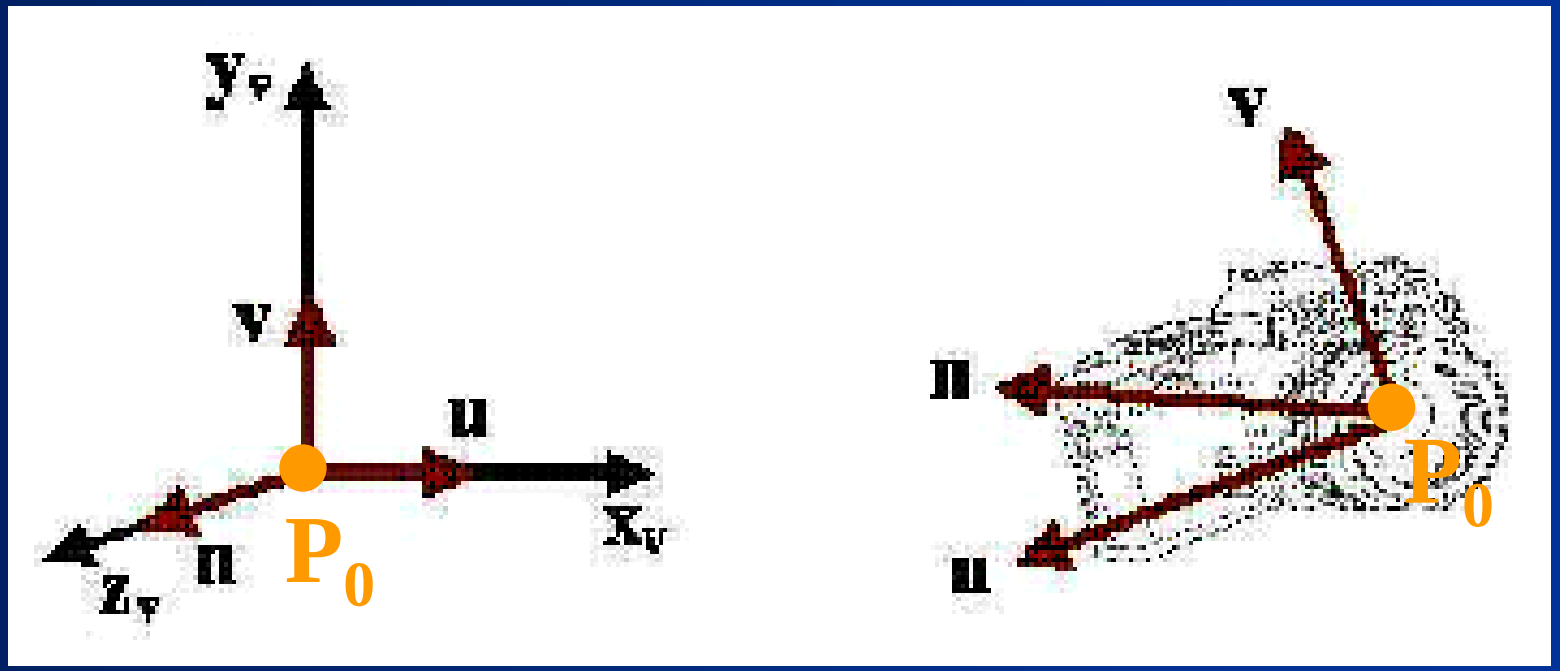
Look and Up Vectors

- ***Look Vector***
  - the direction the camera is pointing
  - three degrees of freedom; can be any vector in 3D-space
- ***Up Vector***
  - determines how the camera is rotated around the *Look vector*
  - for example, whether you're holding the camera horizontally or vertically (or in between)
  - projection of *Up vector* must be in the plane perpendicular to the look vector (this allows *Up vector* to be specified at an arbitrary angle to its *Look vector*)



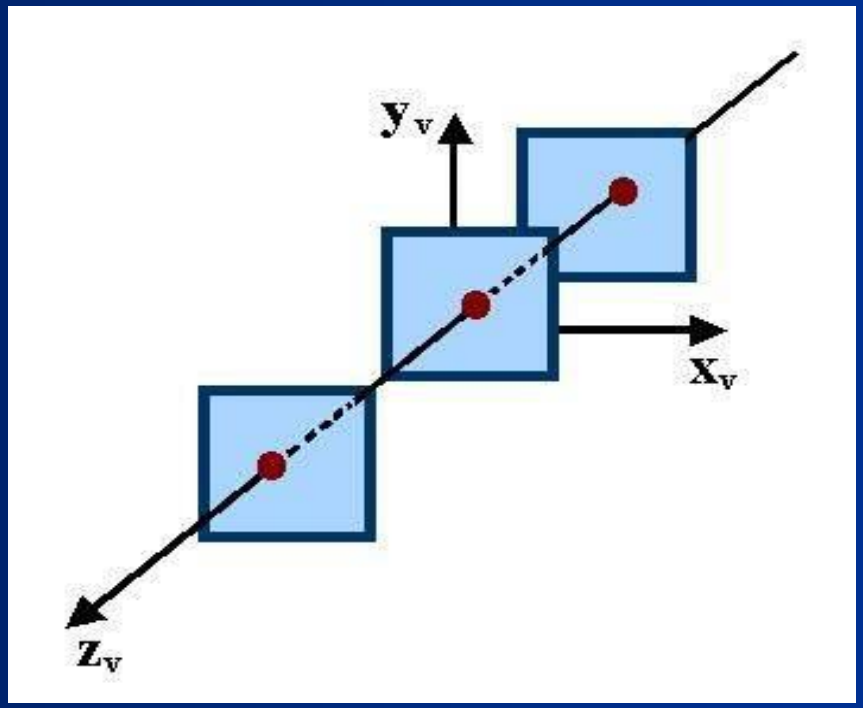
Specifying the Viewing Coordinate System (x_v Axis)

- Using vectors **N** and **V**, the graphics package computer can compute a third vector **U**, perpendicular to both **N** and **V**, to define the direction for the x_v axis.



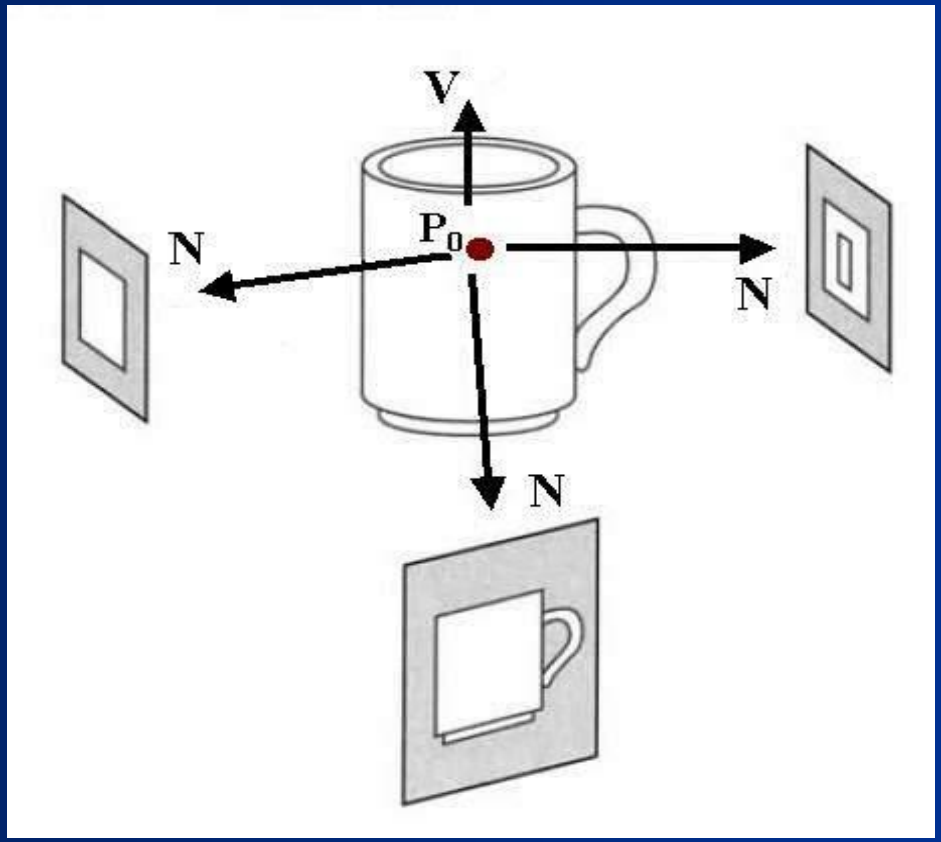
The View Plane

- Graphics package allow users to choose the position of the view plane along the z_v axis by specifying the **view plane distance** from the viewing origin.
- The view plane is always parallel to the $x_v y_v$ plane.



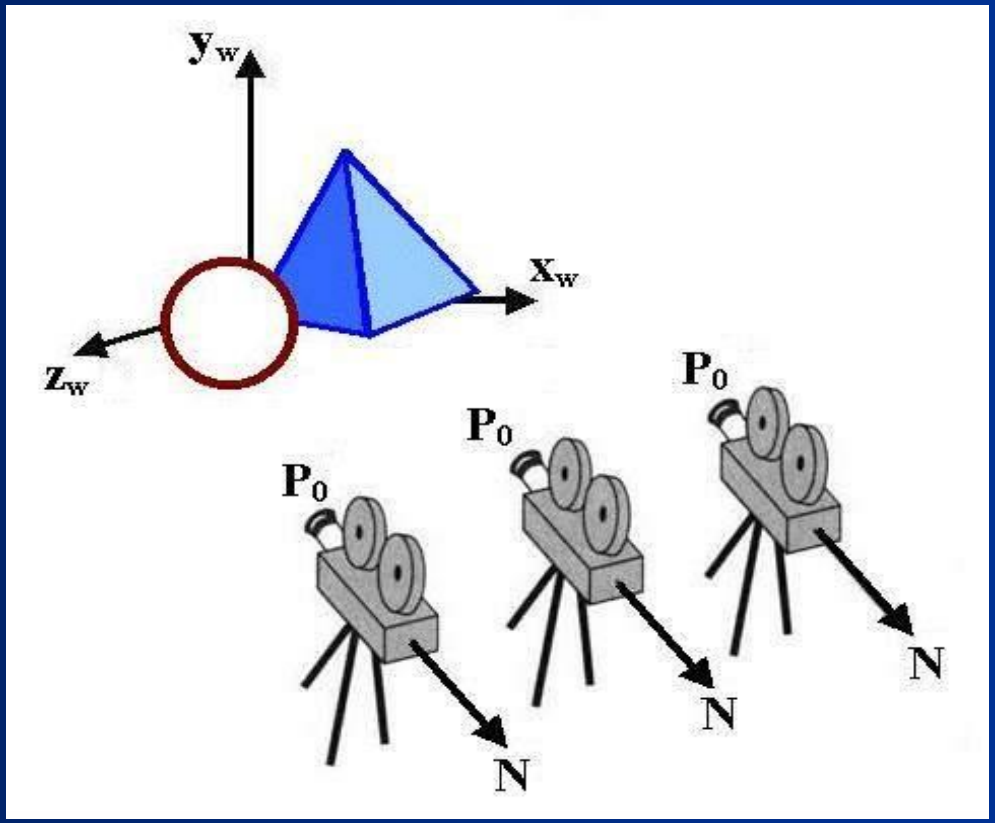
Obtain a Series of View

- To obtain a series of view of a scene, we can keep the view reference point fixed and **change** the direction of **N**.



Simulate Camera Motion

- To simulate camera motion through a scene, we can keep **N fixed** and **move** the view reference **point** around.

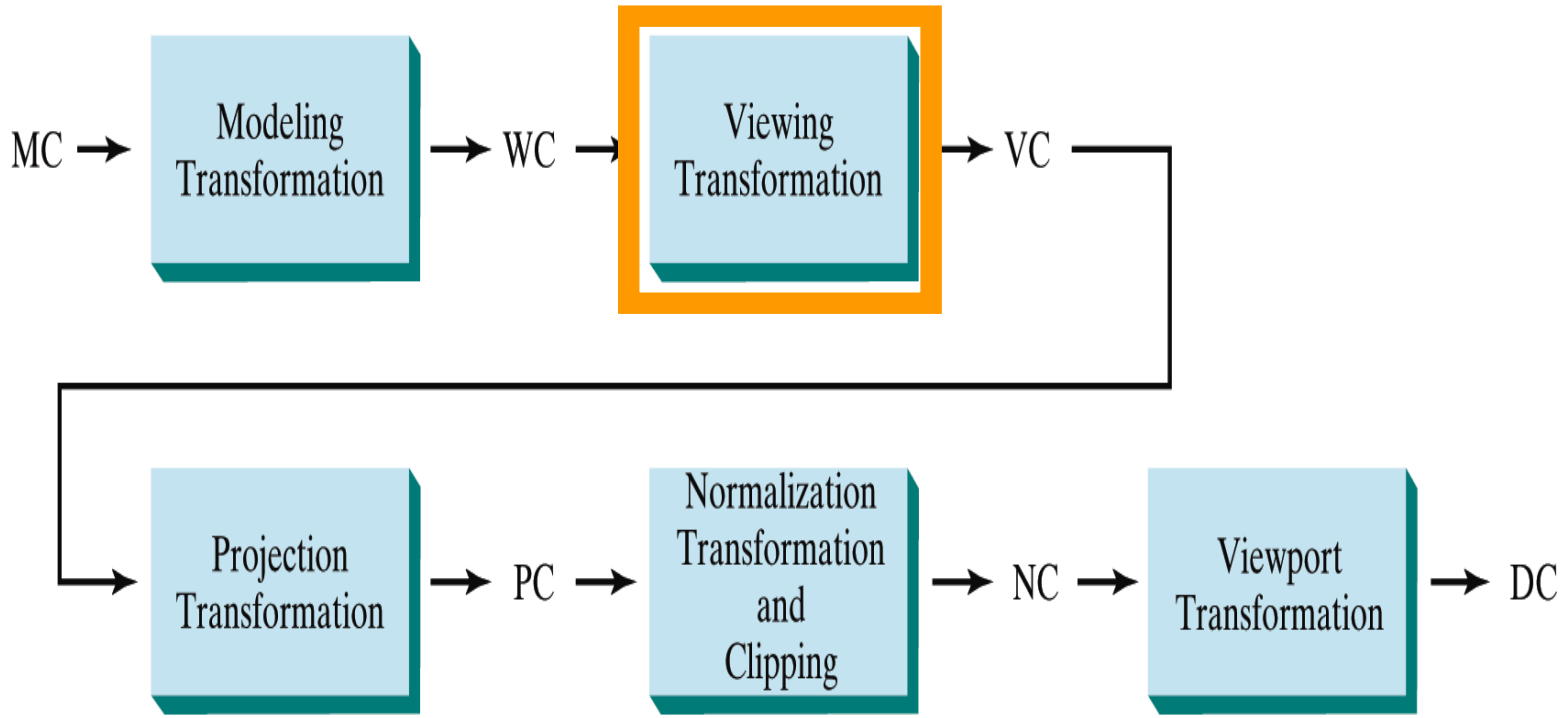


Transformation from World to Viewing Coordinates

Henry Ford Int'l College, Kalanki, Kathmandu
By: Hari Prashad Pant

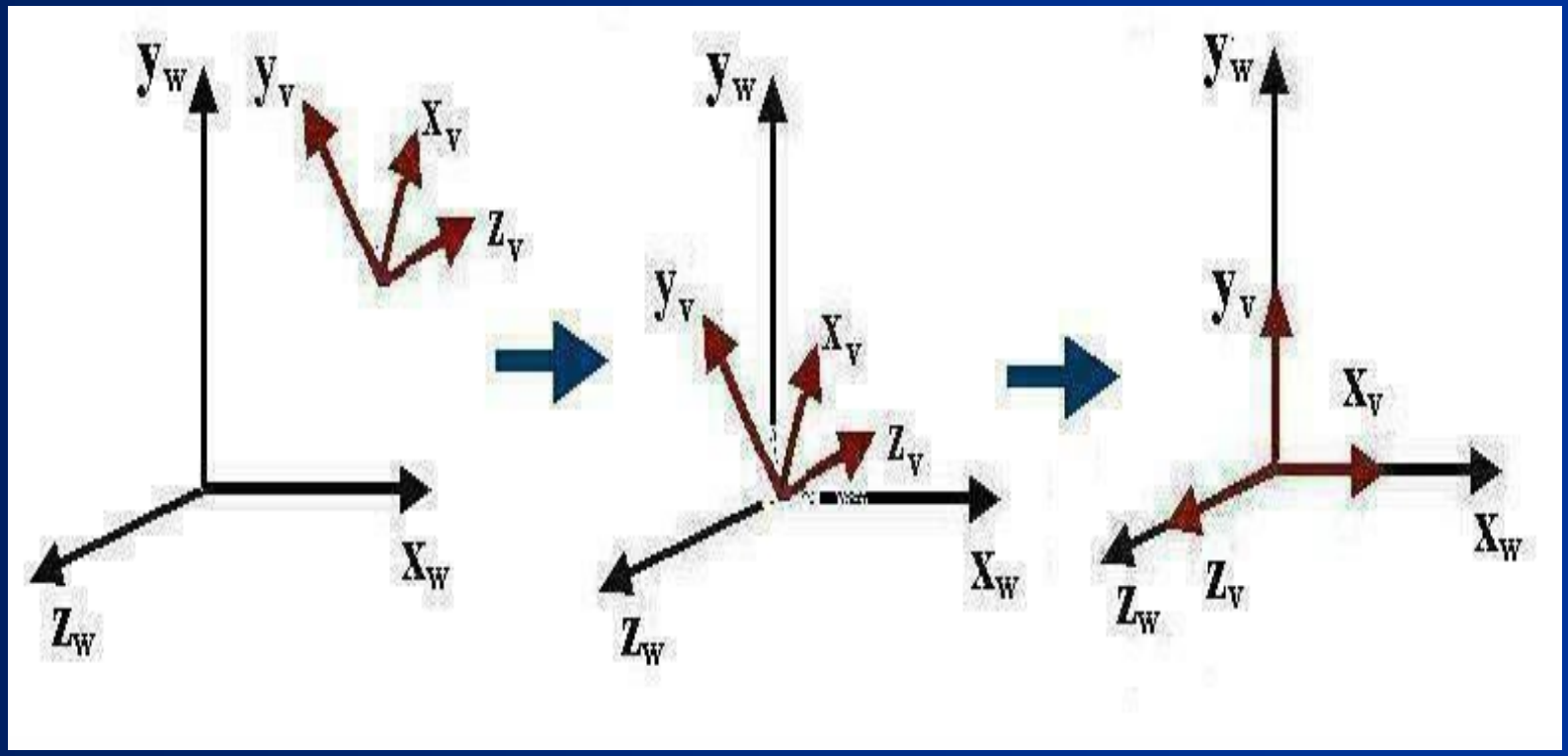
Viewing Pipeline

- Before object description can be projected to the view plane, they must be transferred to viewing coordinates.
- World coordinate positions are converted to viewing coordinates.



Transformation from World to Viewing Coordinates

- Transformation sequence from world to viewing coordinates:



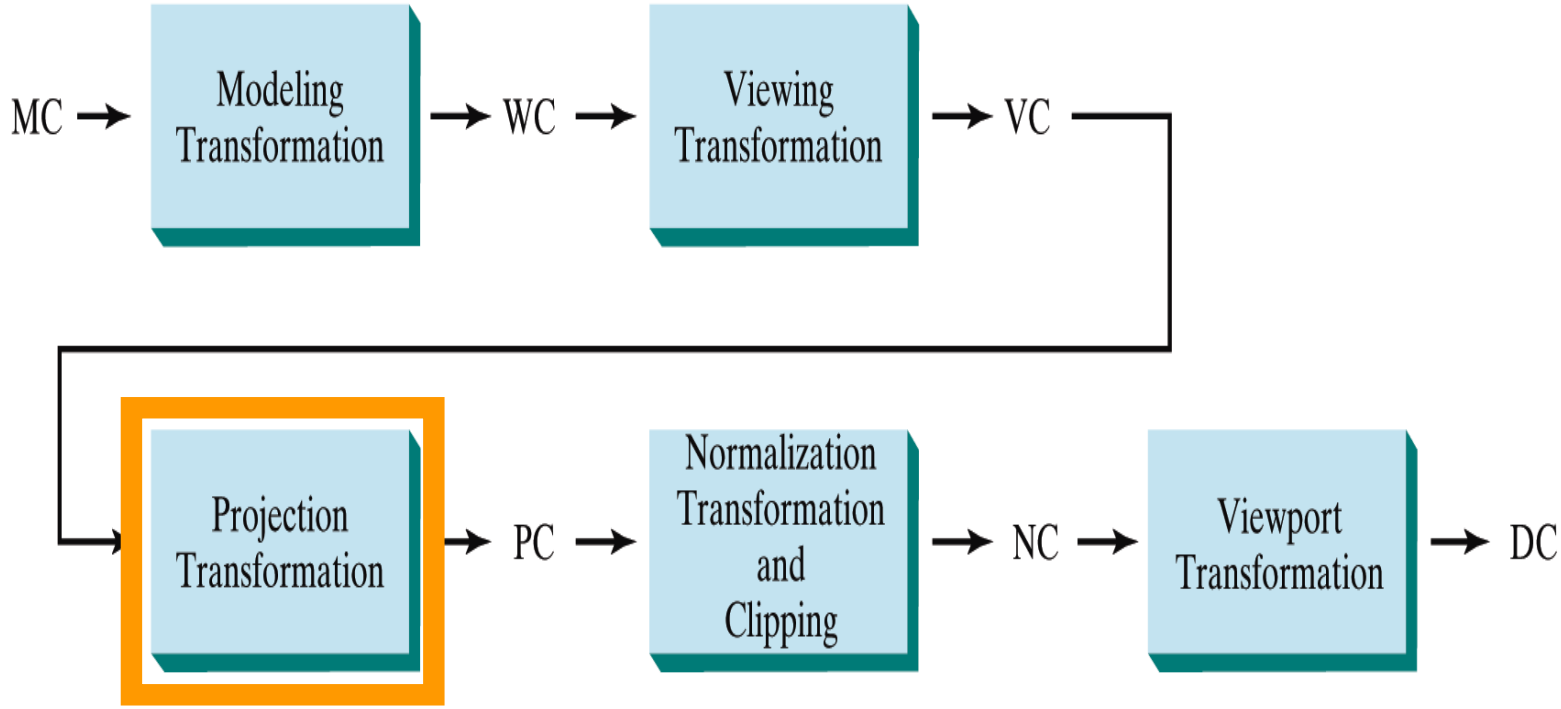
$$\mathbf{M}_{wc,vc} = \mathbf{R}_z . R_y . R_x . \mathbf{T}$$

Projection

Henry Ford Int'l College, Kalanki, Kathmandu
By: Hari Prashad Pant

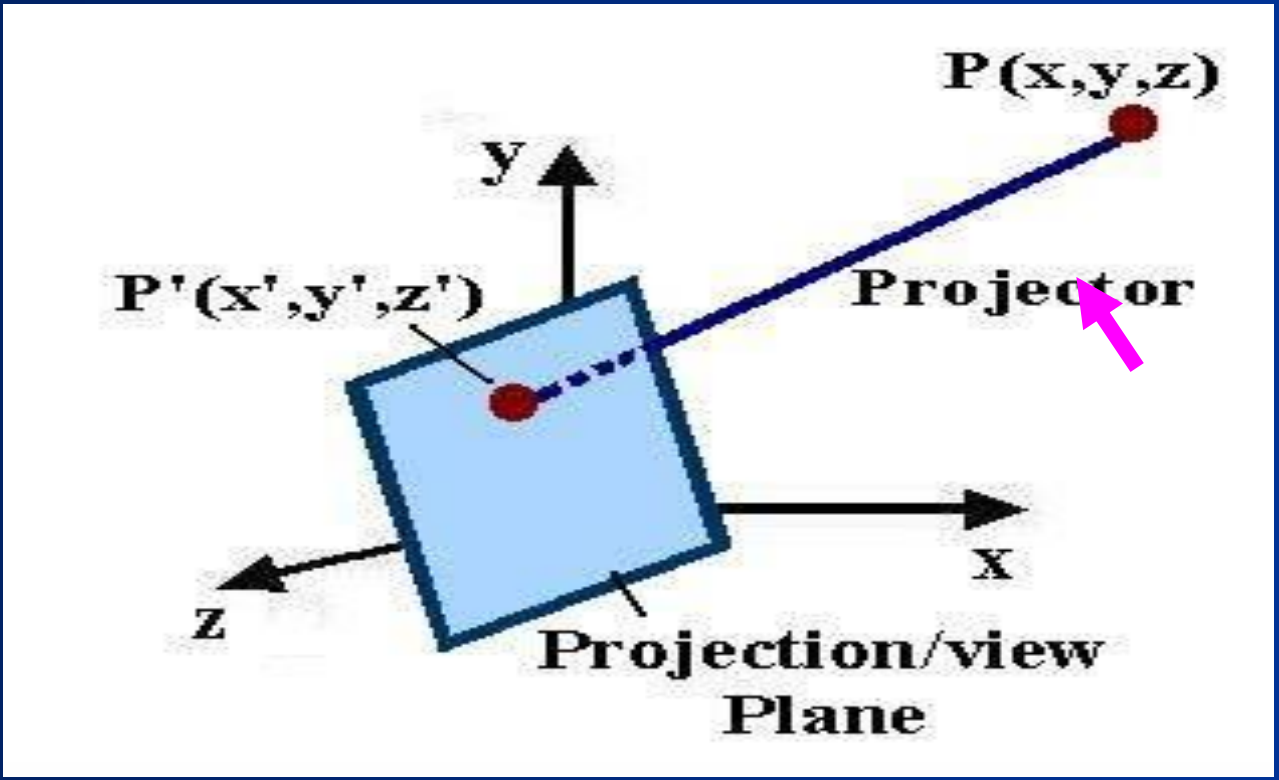
Viewing Pipeline

- Convert the viewing coordinate description of the scene to coordinate positions on the projection plane.
- Viewing 3D objects on a 2D display requires a mapping from 3D to 2D.



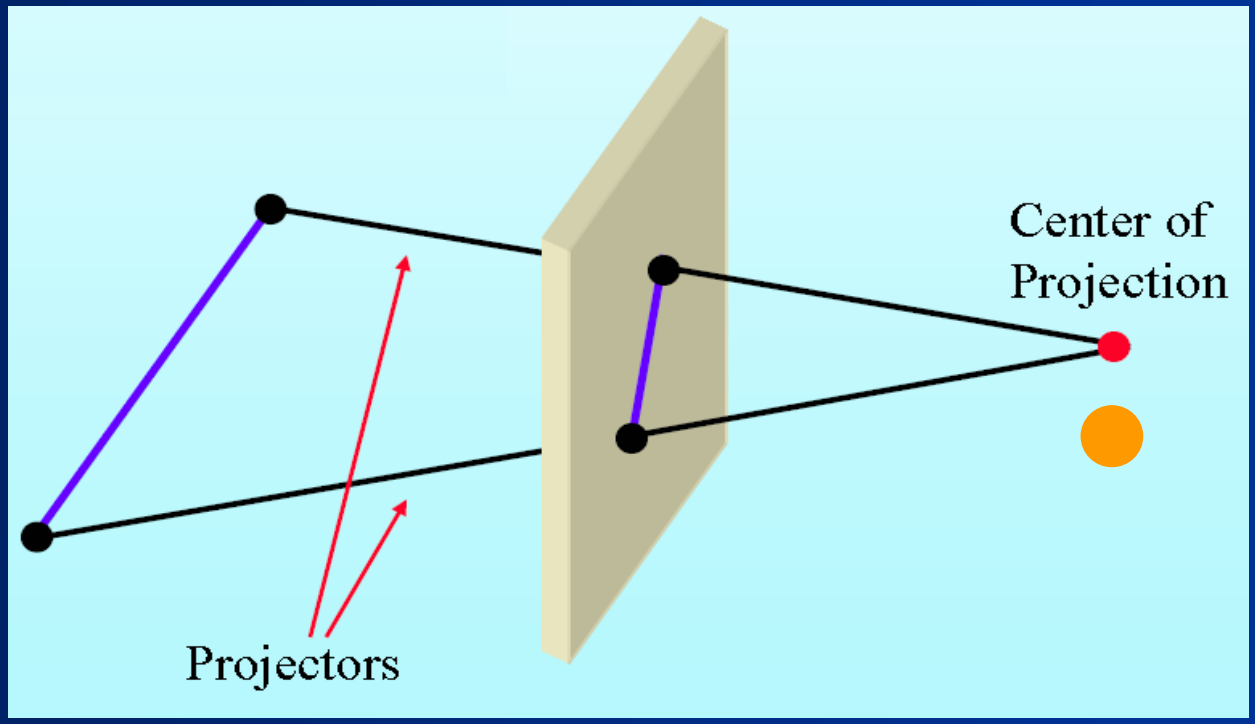
Projection

- **Projection** can be defined as a mapping of point P(x,y,z) onto its image $P'(x', y', z')$ in the projection plane.
- The mapping is determined by a ***projector*** that passes through P and intersects the view plane (P').

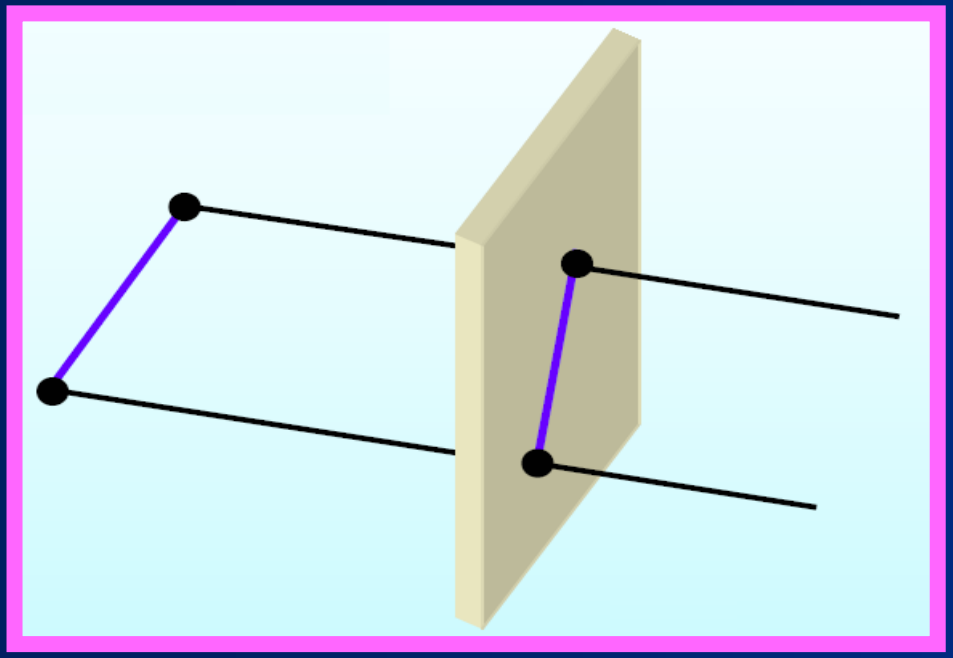


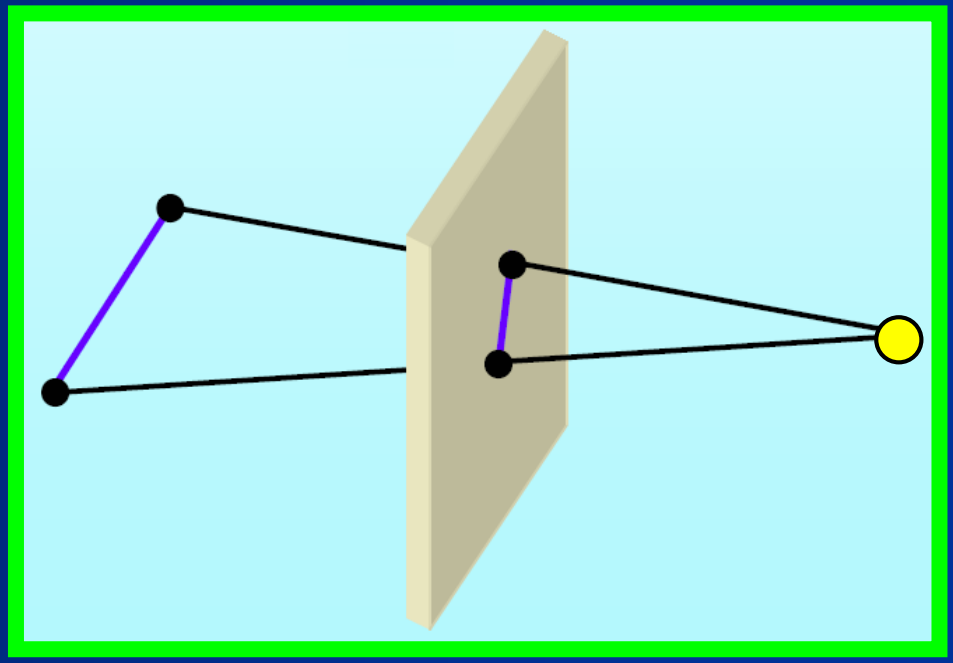
Projection

- Projectors are lines from **center (reference) of projection** through each point in the object.
- The result of projecting an object is dependent on the spatial relationship among the projectors and the view plane.



Projection



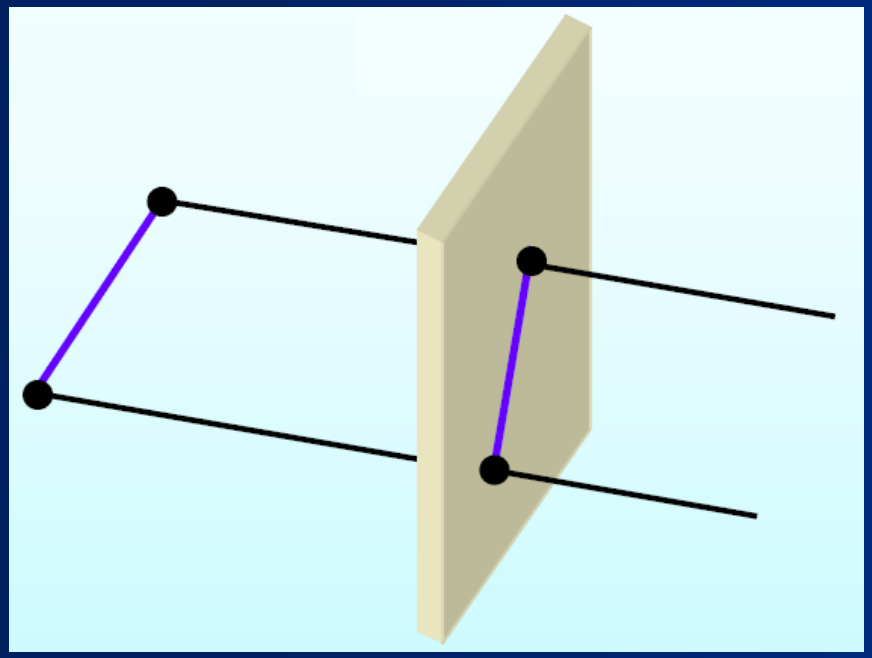


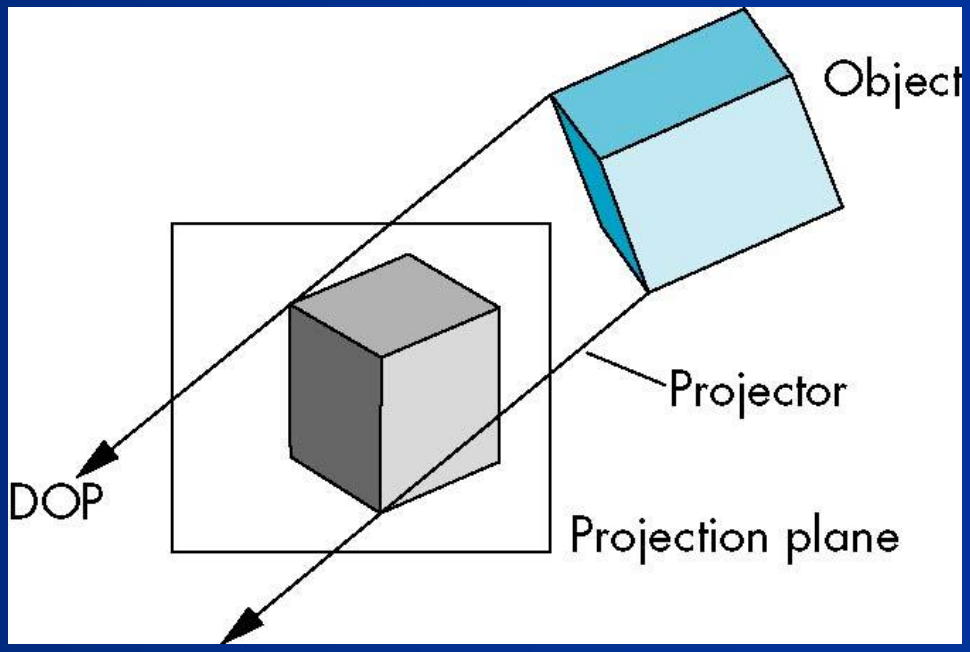
Parallel Projection :
Coordinate position are transformed to the view plane along **parallel lines**.

Perspective Projection:
Object positions are transformed to the view plane along lines that converge to the **projection reference (center) point**.

Parallel Projection

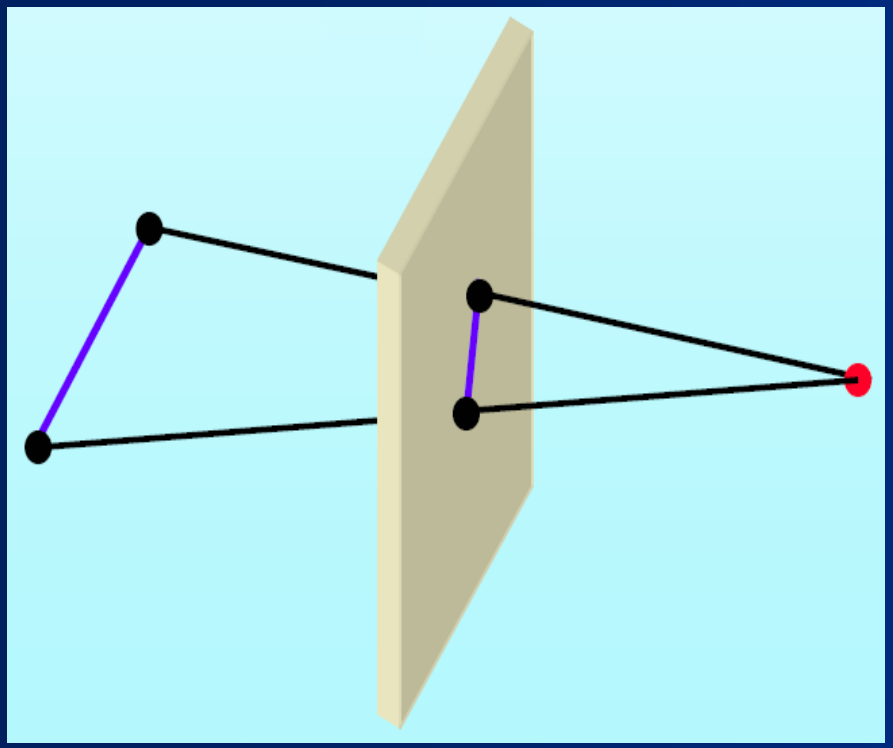
- Coordinate position are transformed to the view plane along parallel lines.
- Center of projection at infinity results with a parallel projection.
- A parallel projection preserves relative proportion of objects, but dose not give us a realistic representation of the appearance of object.

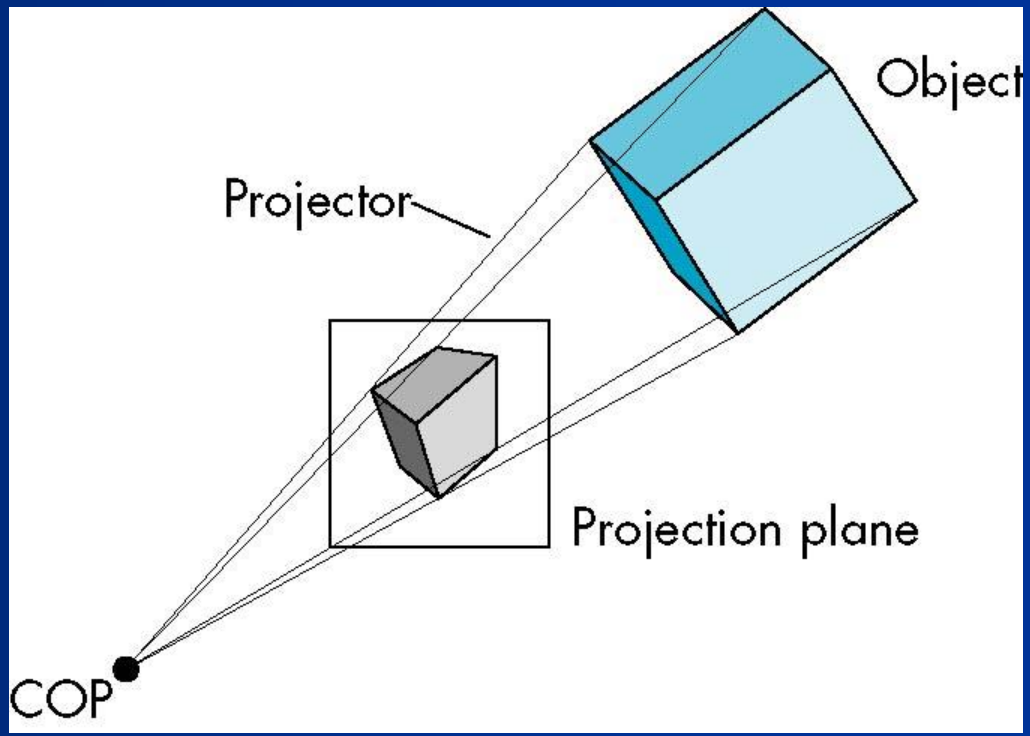




Perspective Projection

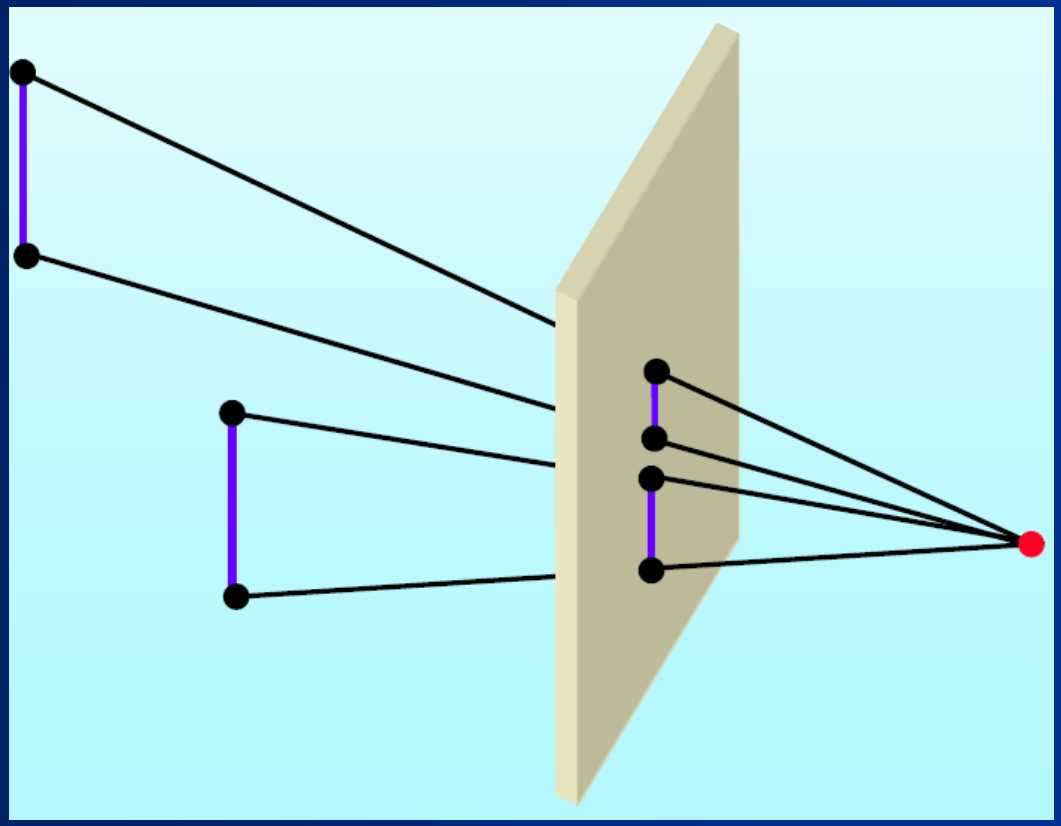
- Object positions are transformed to the view plane along lines that converge to the **projection reference (center) point.**
- Produces realistic views but does not preserve relative proportion of objects.



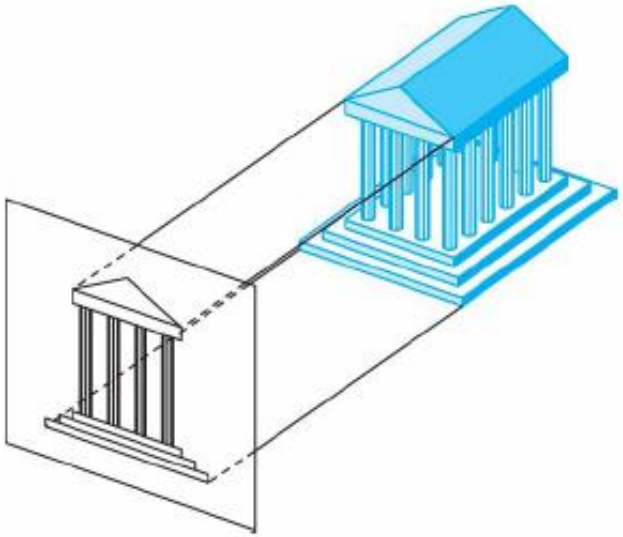


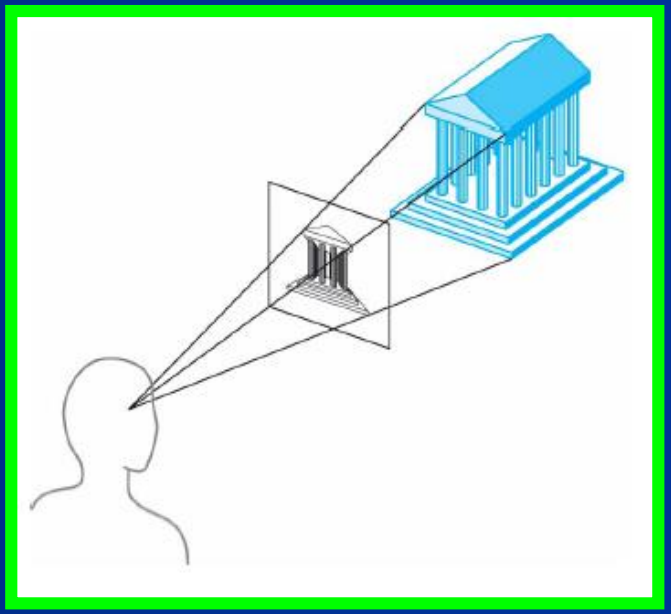
Perspective Projection

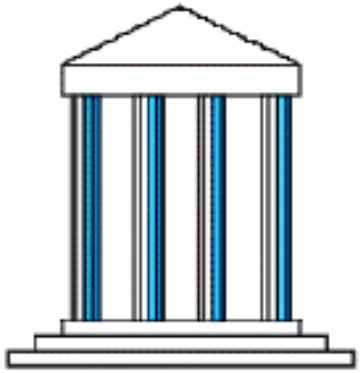
- Projections of distant objects are smaller than the projections of objects of the same size that are closer to the projection plane.



Parallel and Perspective Projection







parallel
(orthographic)

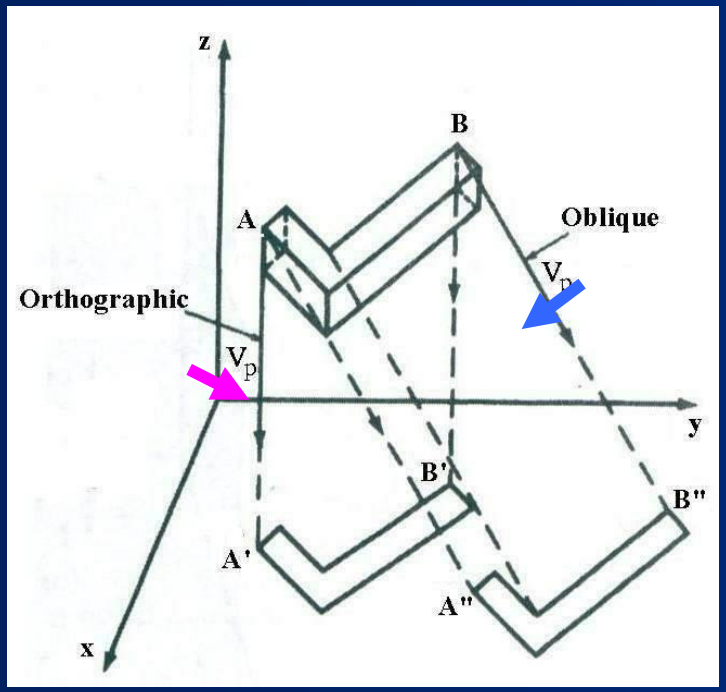
perspective

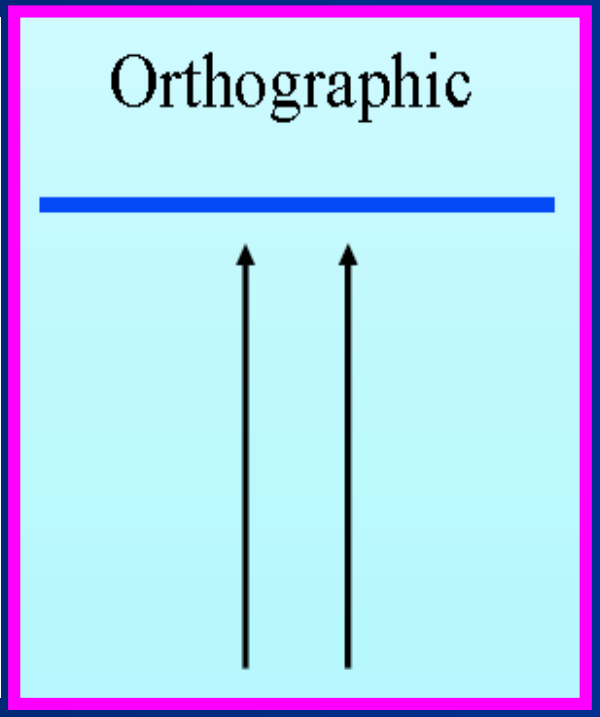
Parallel Projection

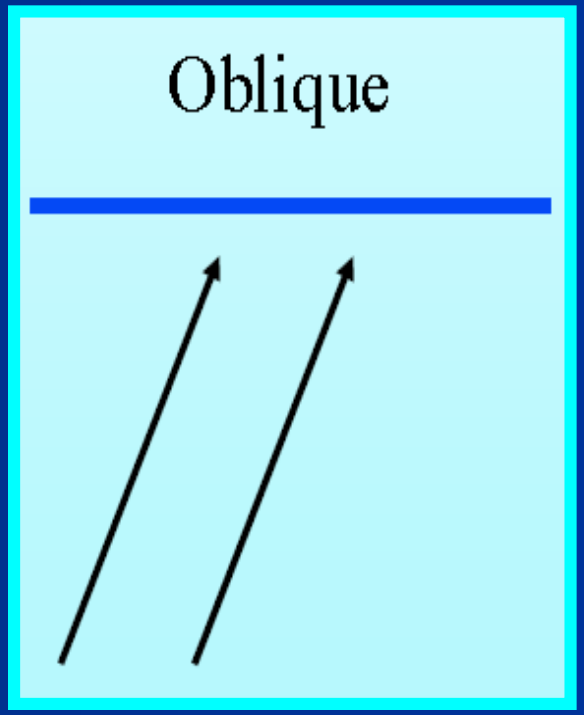
Henry Ford Int'l College, Kalanki, Kathmandu
By: Hari Prashad Pant

Parallel Projection

- **Projection vector**: Defines the direction for the projection lines (projectors).
- ***Orthographic Projection***: Projectors (projection vectors) are **perpendicular** to the projection plane.
- ***Oblique Projection***: Projectors (projection vectors) are ***not*** perpendicular to the projection plane.





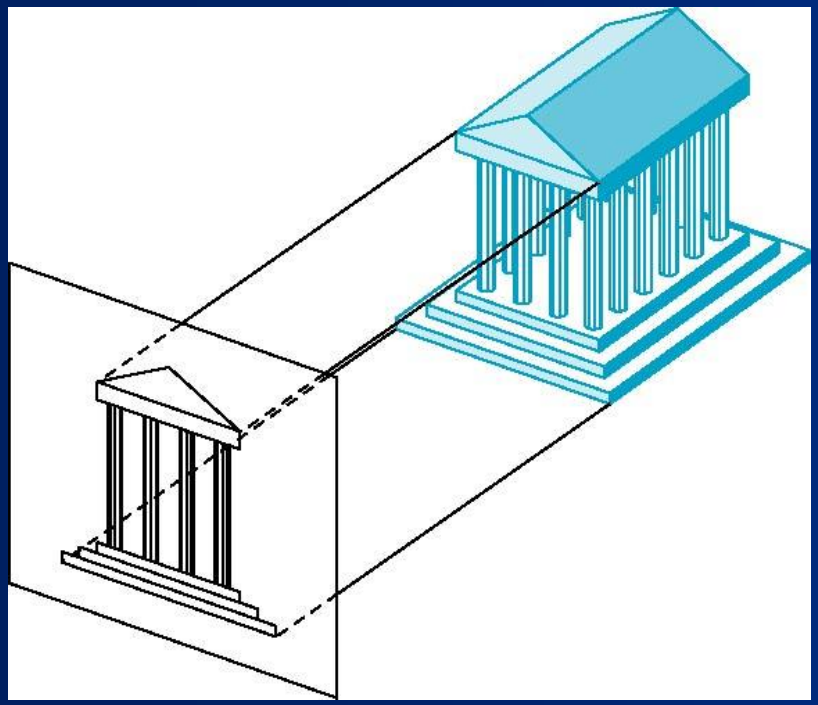


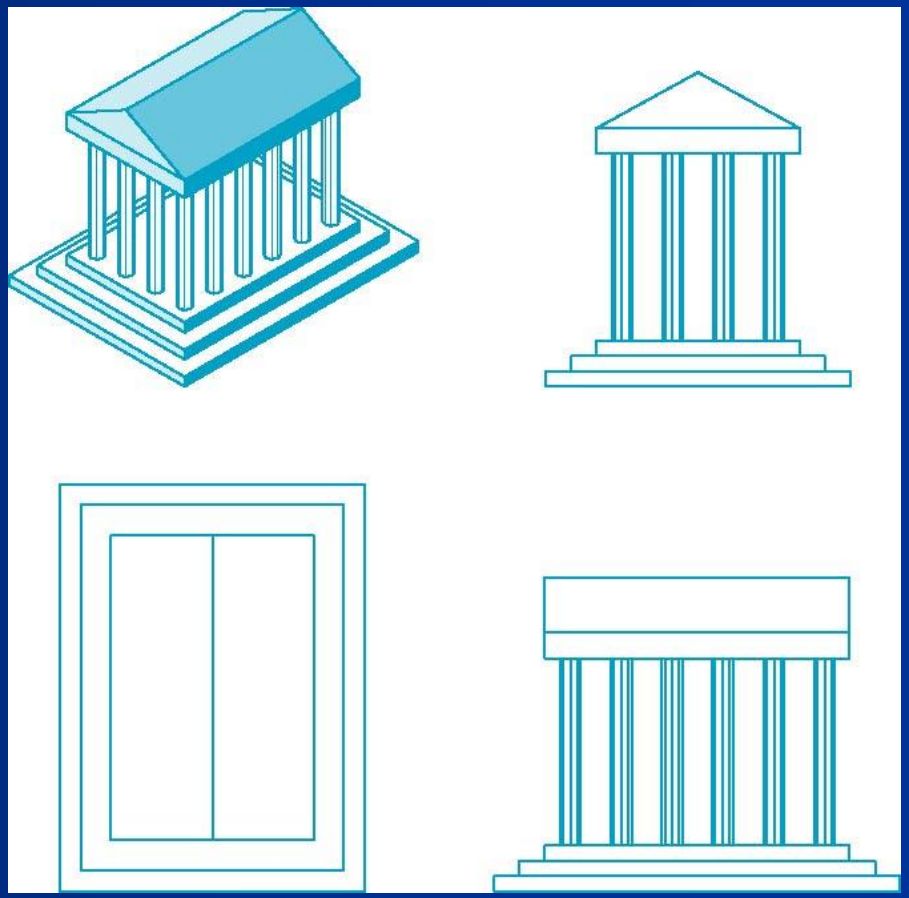
Orthographic Parallel Projection

Henry Ford Int'l College, Kalanki, Kathmandu
By: Hari Prashad Pant

Orthographic Parallel Projection

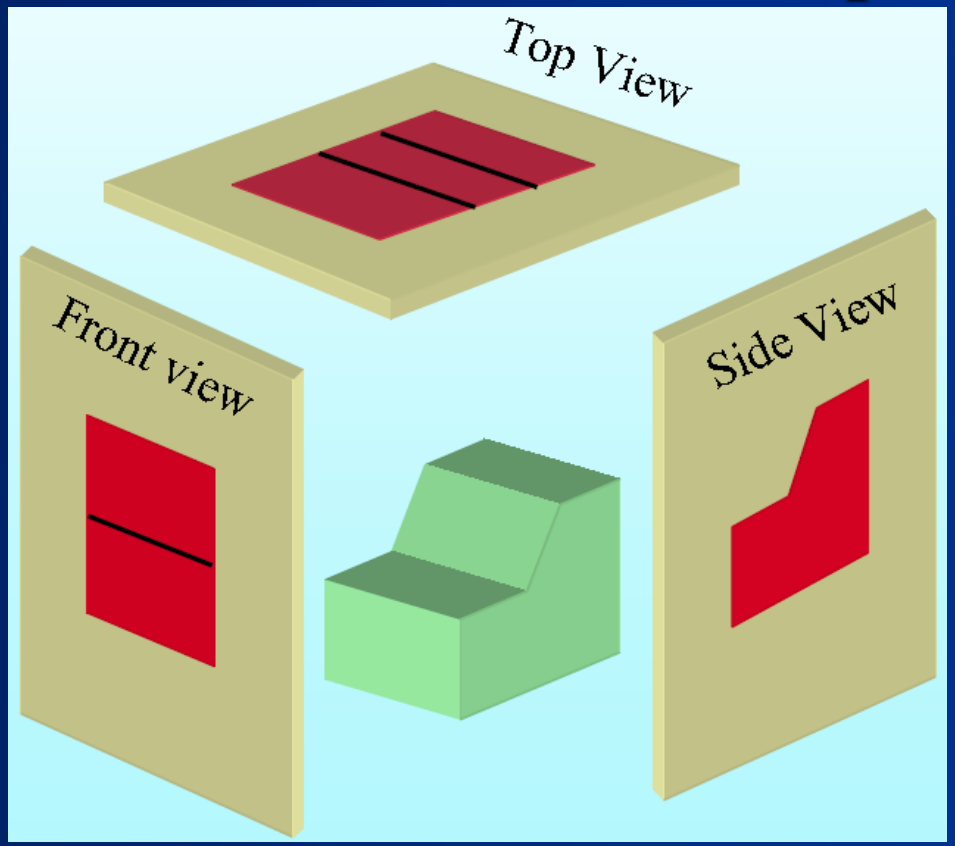
- Orthographic projection used to produce the **front**, **side**, and **top** views of an object.



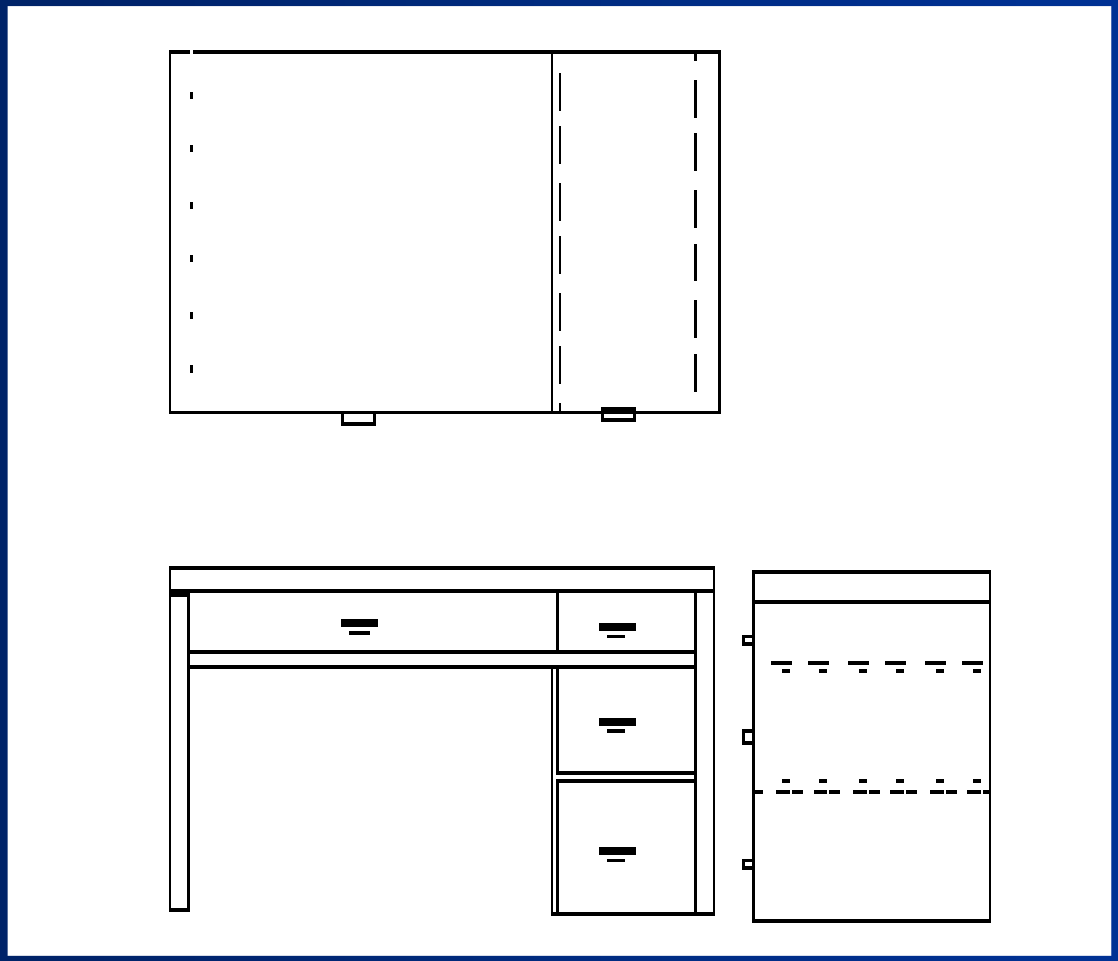


Orthographic Parallel Projection

- ***Front***, ***side***, and ***rear*** orthographic projections of an object are called ***elevations***.
- ***Top*** orthographic projection is called a ***plan*** view.



Orthographic Parallel Projection

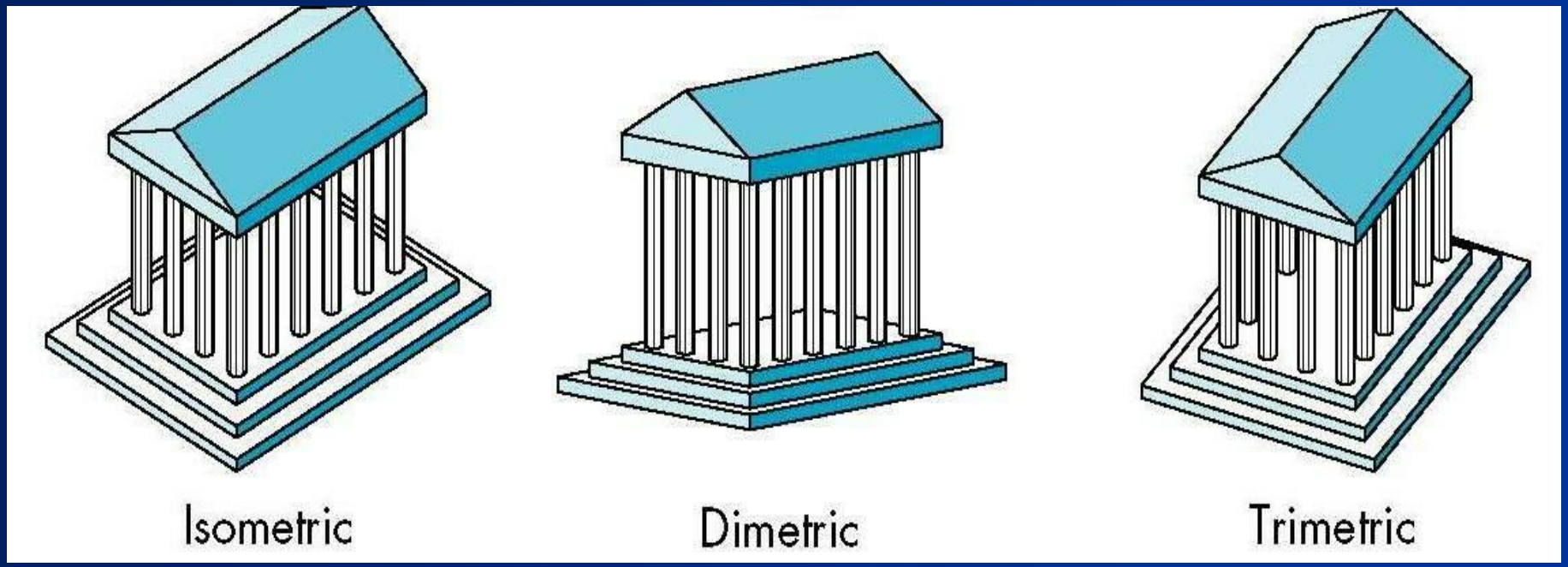


Multi View Orthographic

Henry Ford Int'l College, Kalanki, Kathmandu
By: Hari Prashad Pant

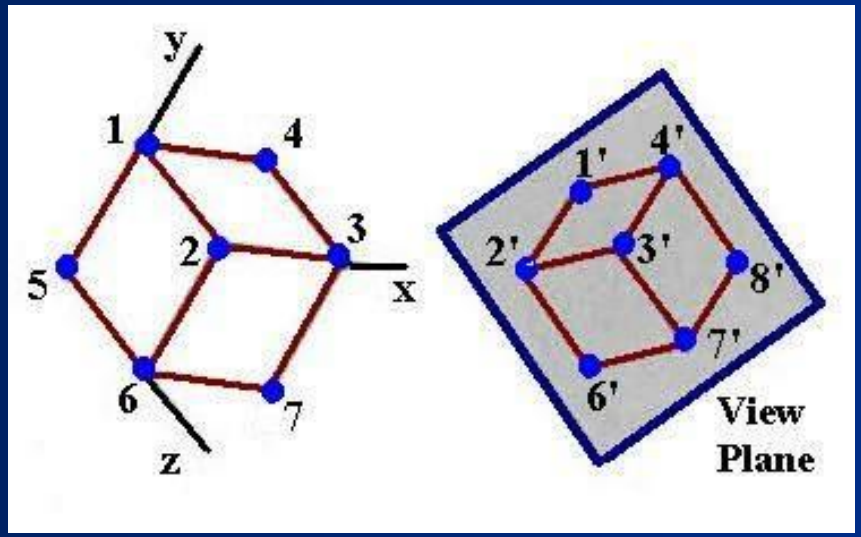
Orthographic Parallel Projection

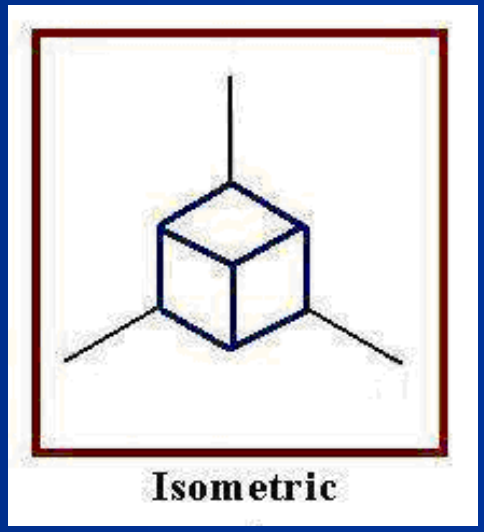
- ***Axonometric orthographic*** projections display more than one face of an object.



Orthographic Parallel Projection

- ***Isometric Projection***: Projection plane intersects each coordinate axis in which the object is defined (principal axes) at the same distant from the origin.
- Projection vector makes equal angles with all of the **three principal axes**.

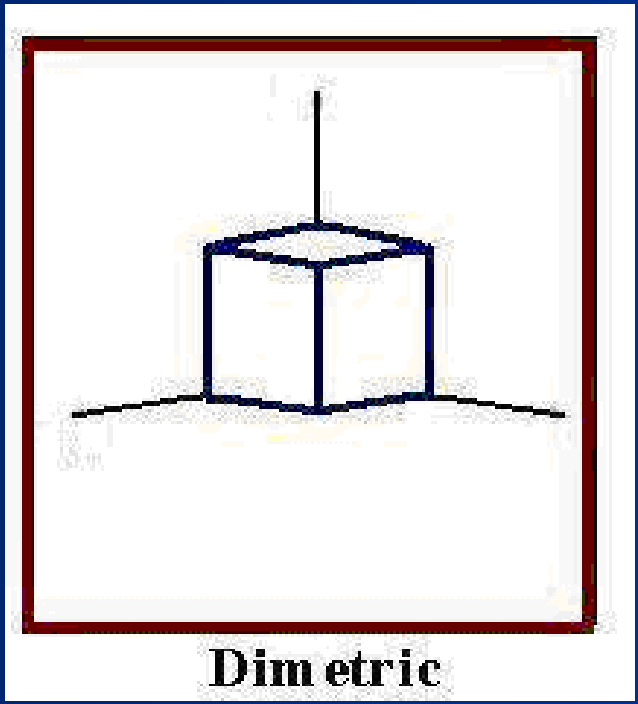




Isometric projection is obtained by **aligning** the **projection vector** with the **cube diagonal**.

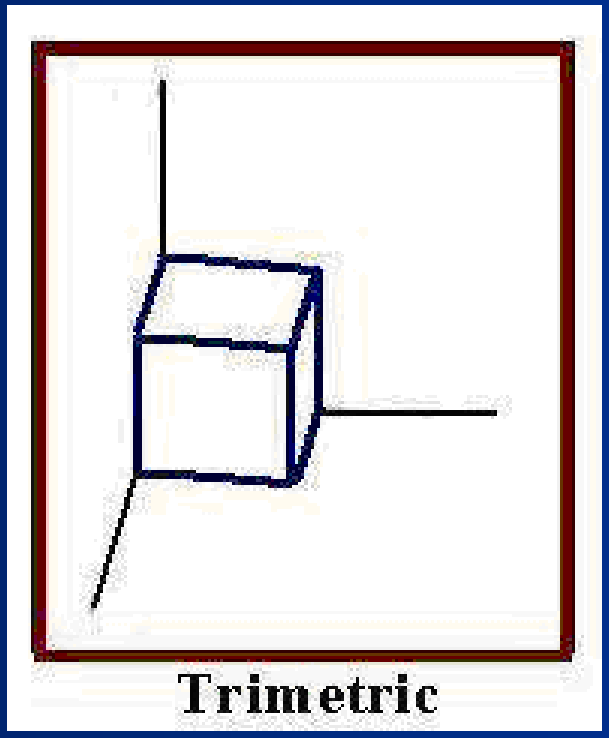
Orthographic Parallel Projection

- ***Dimetric Projection***: Projection vector makes **equal angles** with exactly **two** of the principal axes.

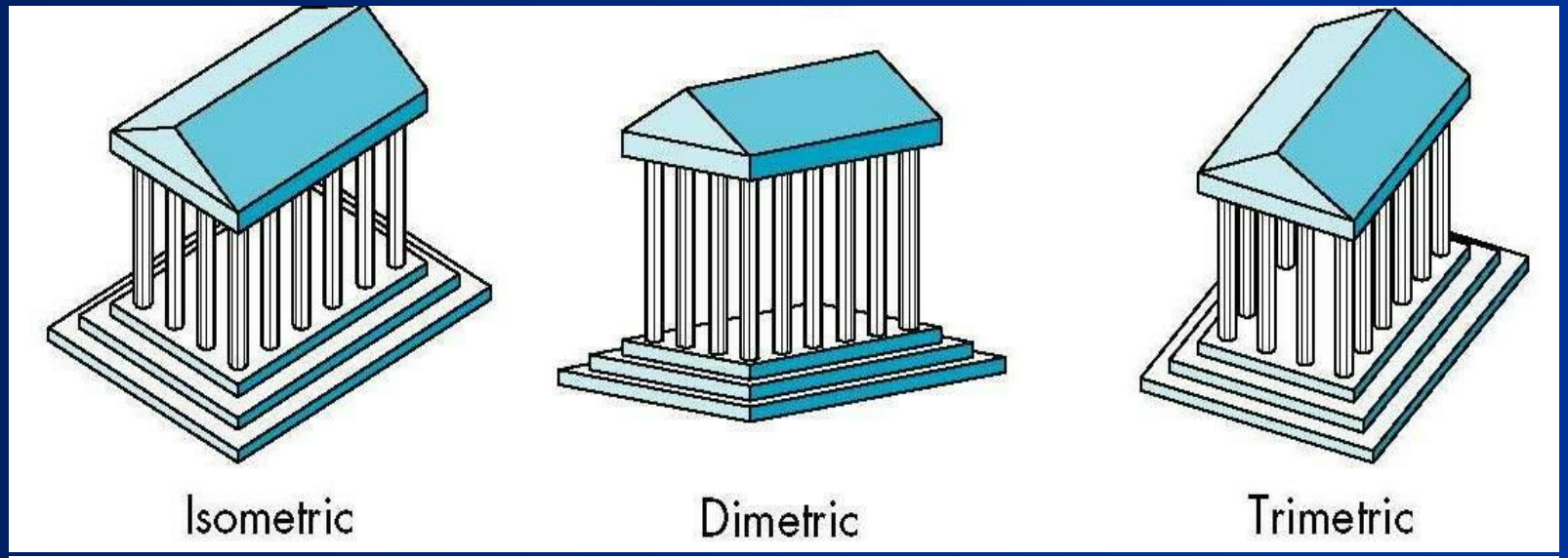


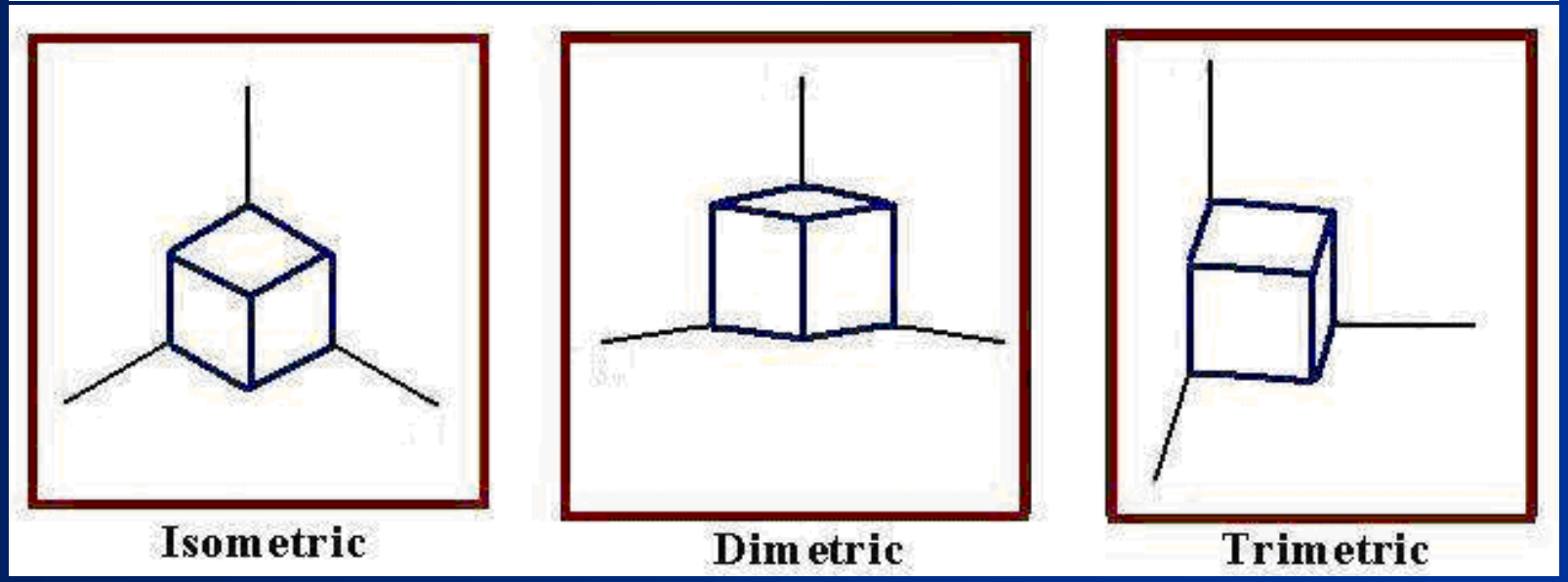
Orthographic Parallel Projection

- ***Trimetric Projection***: Projection vector makes **unequal angles** with the three principal axes.



Orthographic Parallel Projection



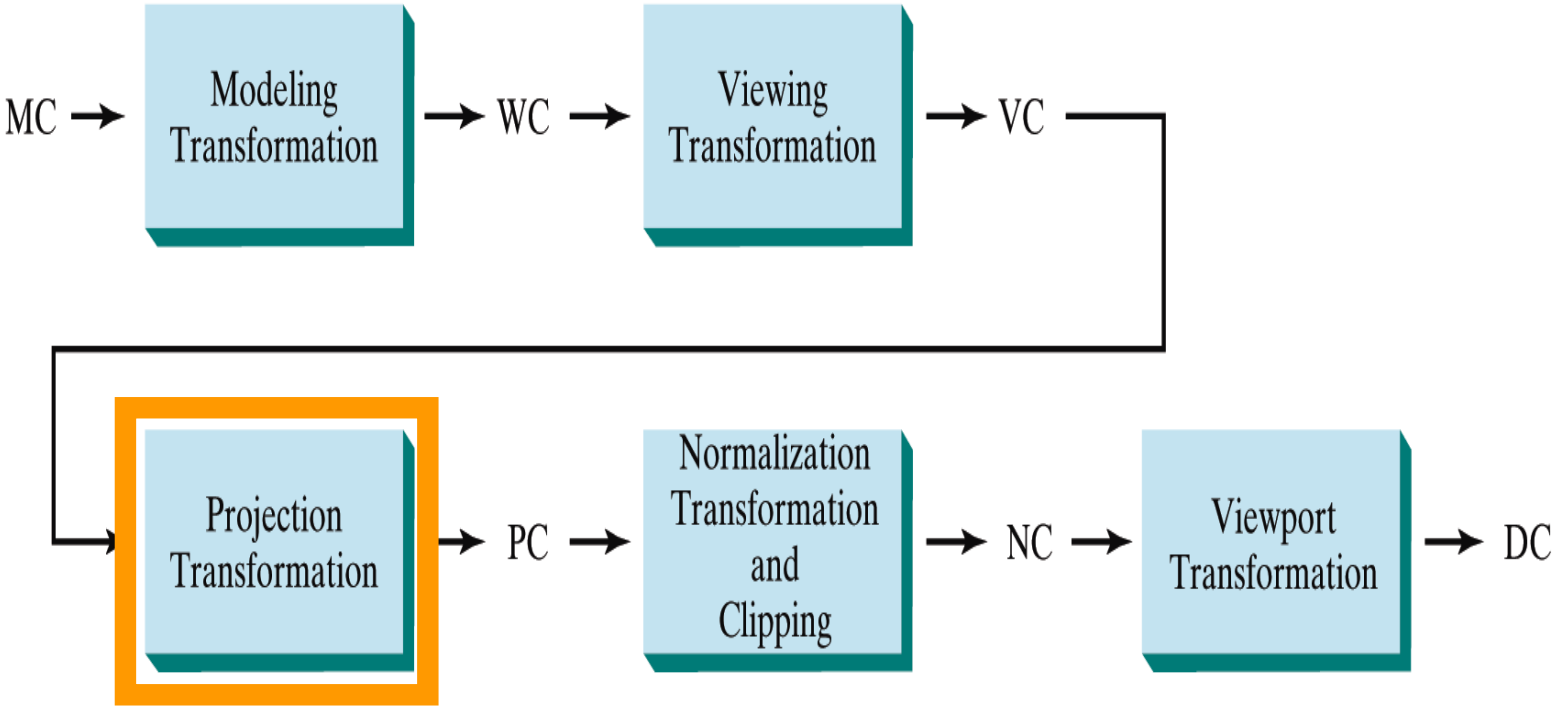


Orthographic Parallel Projection Transformation

Henry Ford Int'l College, Kalanki, Kathmandu
By: Hari Prashad Pant

Orthographic Parallel Projection Transformation

- Convert the **viewing coordinate** description of the scene to coordinate positions on the **Orthographic parallel projection plane**.

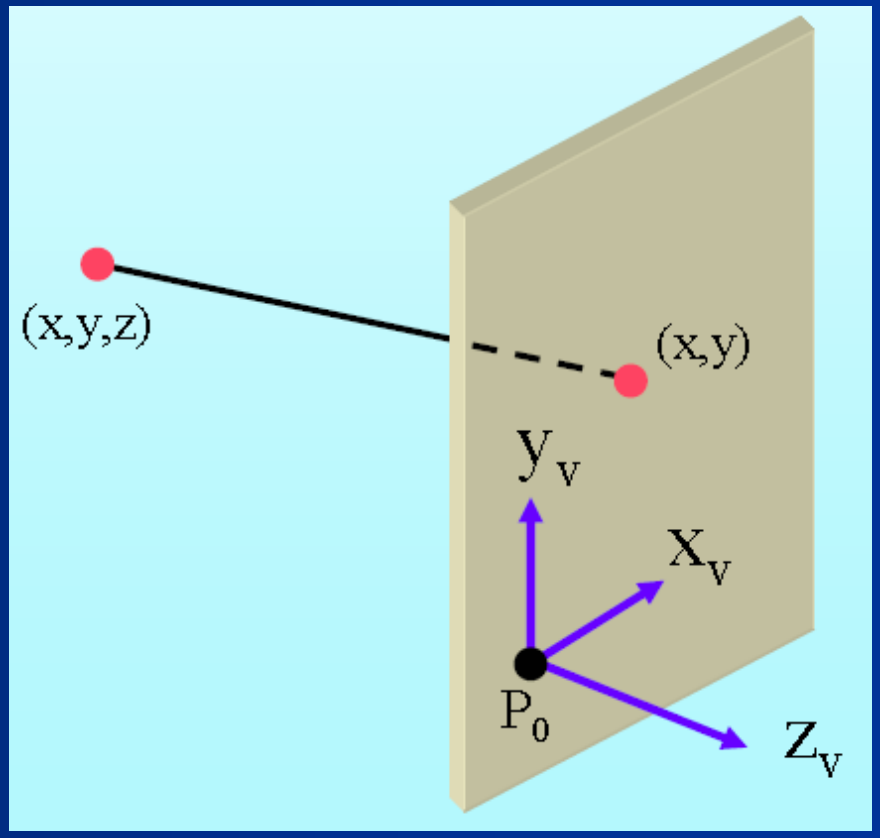


Orthographic Parallel Projection Transformation

- Since the view plane is placed at position z_{vp} along the z_v axis. Then any point (x,y,z) in viewing coordinates is transformed to projection coordinates as:

$$x_p = x, \quad y_p = y$$

$$\mathbf{M}_{Orthographic\ Parallel} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

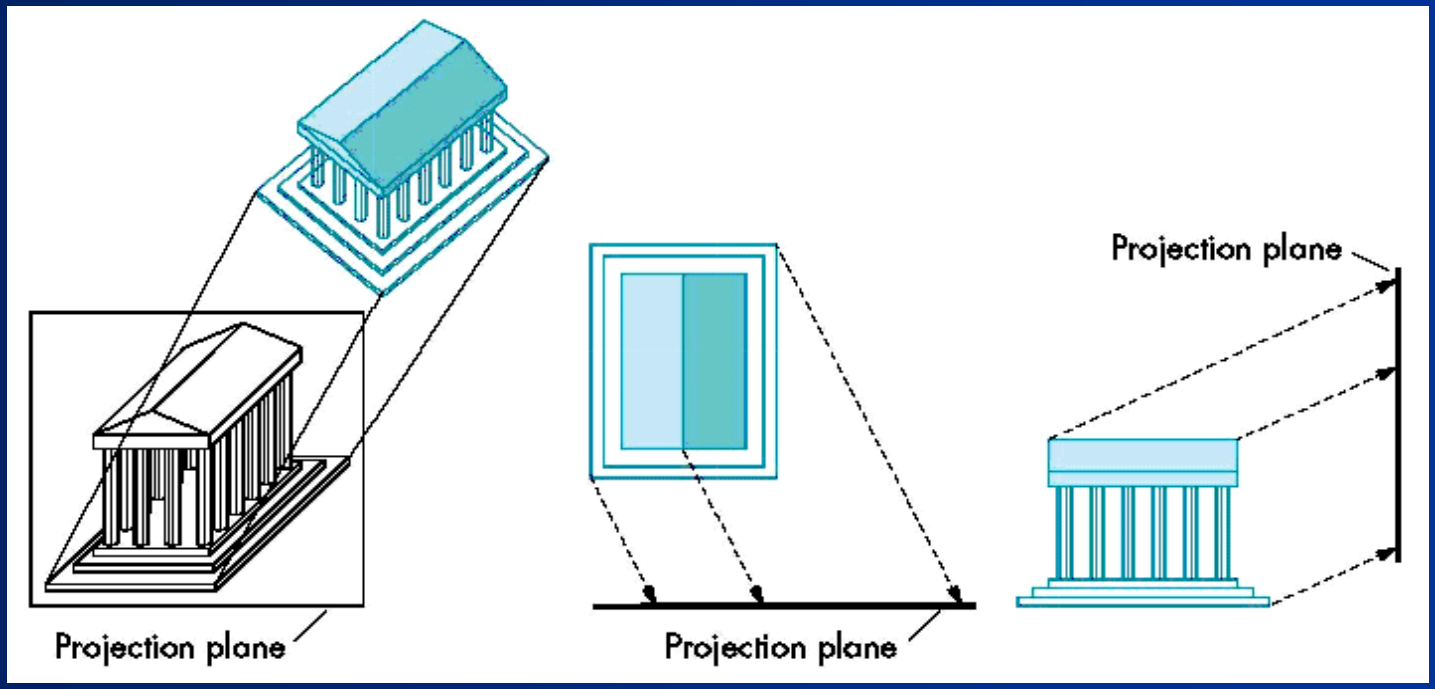


Oblique
Parallel Projection

Henry Ford Int'l College, Kalanki, Kathmandu
By: Hari Prashad Pant

Oblique Parallel Projection

- Projection are **not** perpendicular to the viewing plane.
- Angles and lengths are preserved for faces parallel the plane of projection.
- Preserves 3D nature of an object.

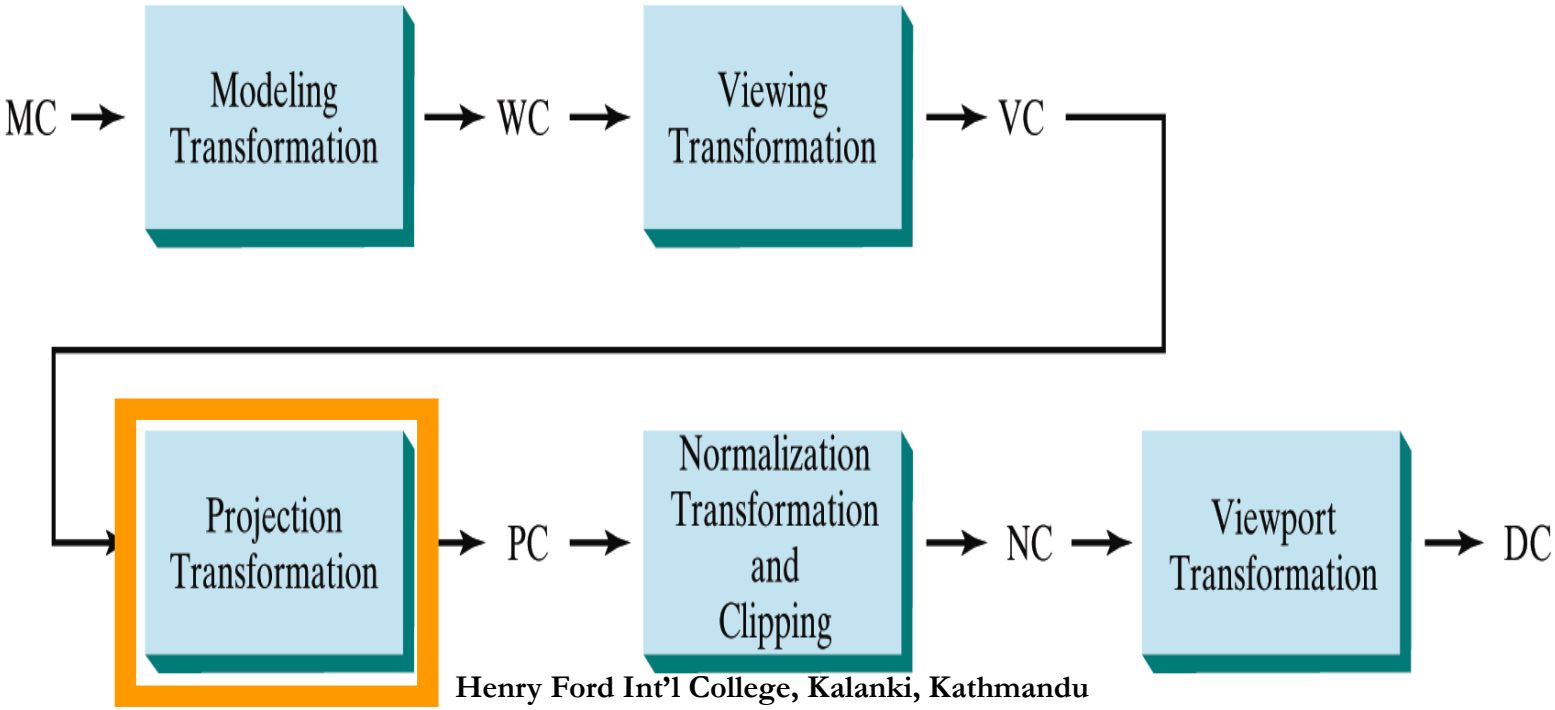


Oblique Parallel Projection Transformation

Henry Ford Int'l College, Kalanki, Kathmandu
By: Hari Prashad Pant

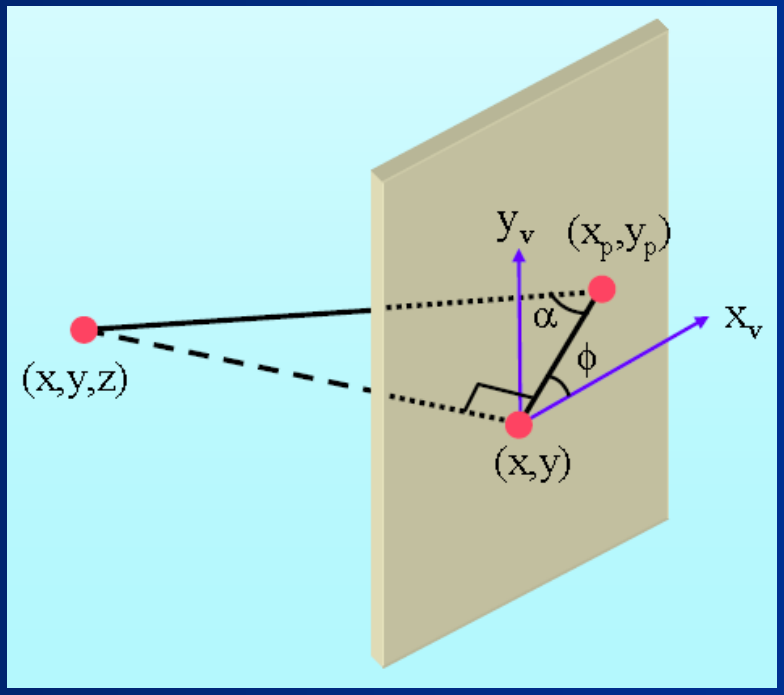
Oblique Parallel Projection Transformation

- Convert the **viewing coordinate** description of the scene to coordinate positions on the **Oblique parallel projection plane**.



Oblique Parallel Projection

- Point (x,y,z) is projected to position (x_p,y_p) on the view plane.
- Projector (oblique) from (x,y,z) to (x_p,y_p) makes an angle α with the line (L) on the projection plane that joins (x_p,y_p) and (x,y).
- Line L is at an angle φ with the horizontal direction in the projection plane.



Oblique Parallel Projection

$$x_p = x + L\cos\varphi$$

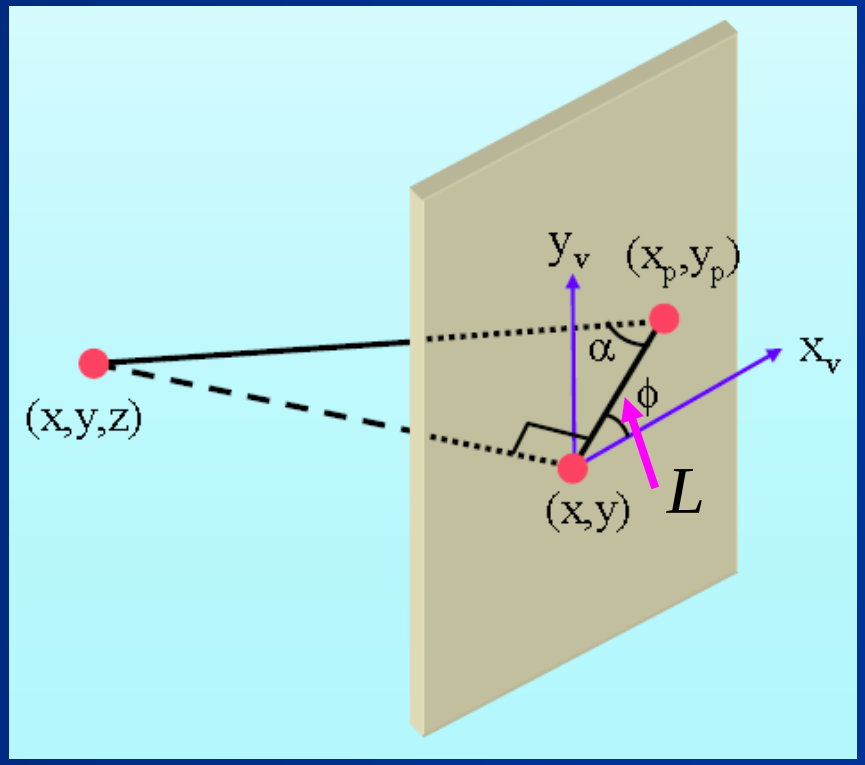
$$y_p = y + L\sin\varphi$$

$$\tan\alpha = \frac{z}{L} \qquad L = \frac{z}{\tan\alpha} = zL_1$$

$$x_p = x + z(L_1\cos\varphi)$$

$$y_p = y + z(L_1\sin\varphi)$$

$$\mathbf{M}_{Parallel} = \begin{bmatrix} 1 & 0 & L_1\cos\varphi & 0 \\ 0 & 1 & L_1\sin\varphi & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Oblique Parallel Projection

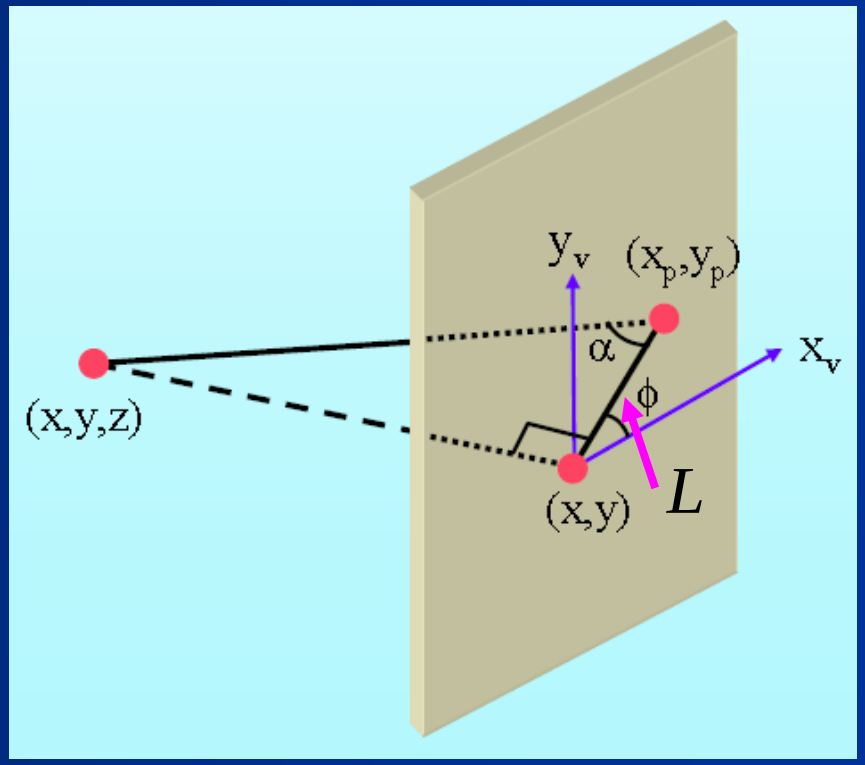
Orthographic Projection:

$$L_1 = 0$$

$$\alpha = 90^\circ$$

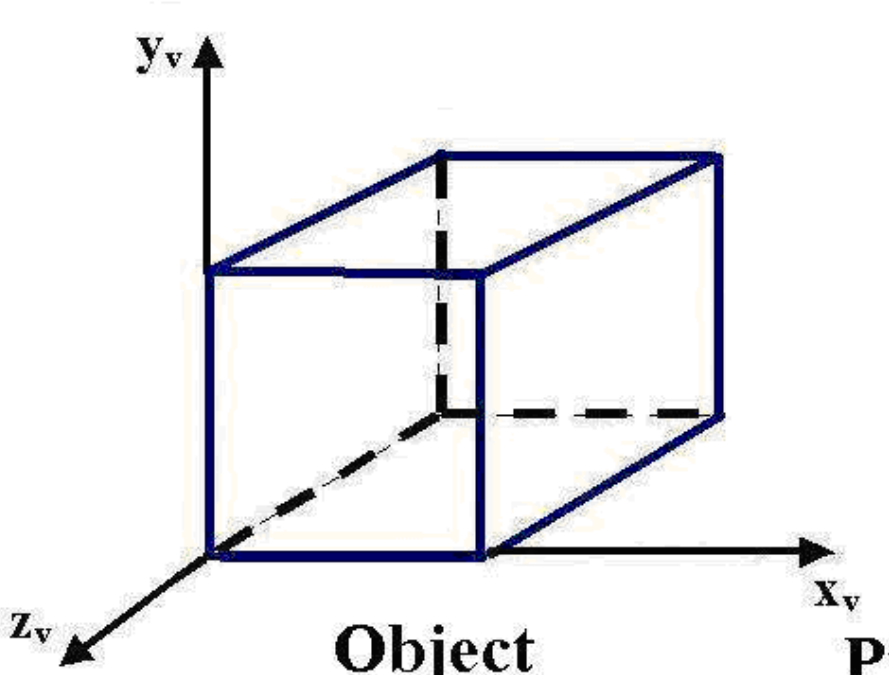
$$x_p = x, \quad y_p = y$$

$$\mathbf{M}_{Orthographic\ Parallel} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

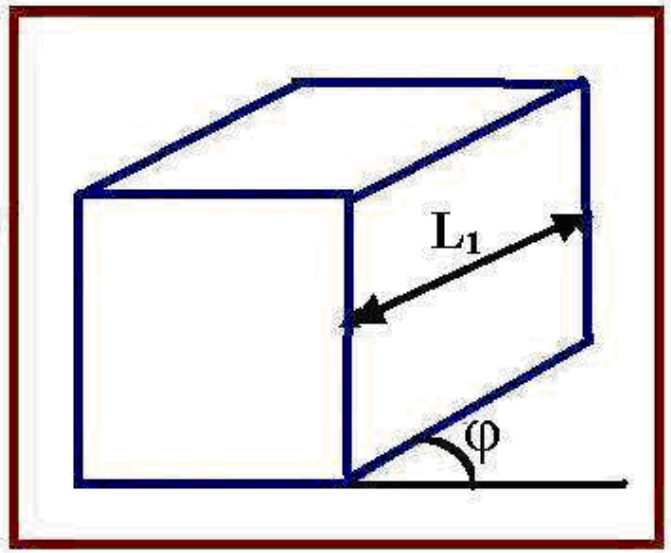


Oblique Parallel Projection

- **Angles**, **distances**, and **parallel lines** in the plane are projected accurately.



Object



Projection on the Viwing Plane

Cavalier Projection

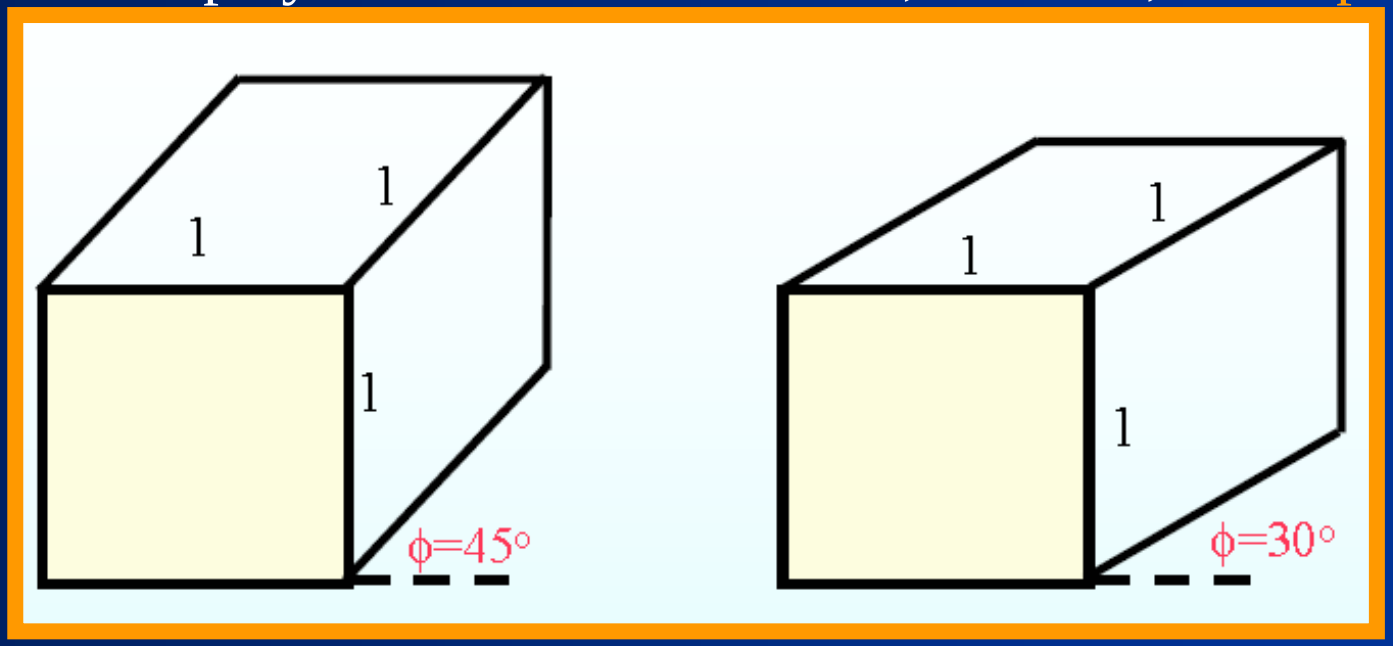
Cavalier Projection:

$\phi = 30^\circ \quad and \quad 45^\circ$

$$\tan\alpha = 1$$

$$\alpha = 45^\circ$$

- Preserves lengths of lines perpendicular to the viewing plane.
- 3D nature can be captured but shape seems distorted.
- Can display a combination of front, and side, and top views.



Cabinet Projection

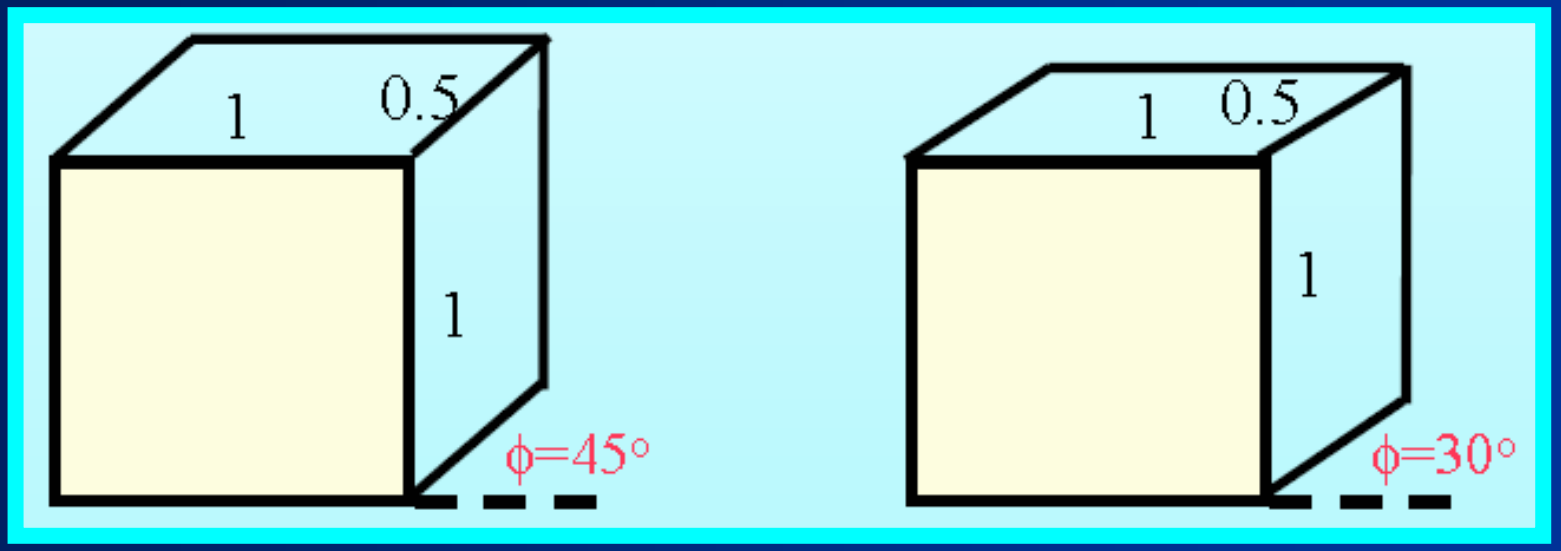
Cabinet Projection:

$\tan\alpha = 2$

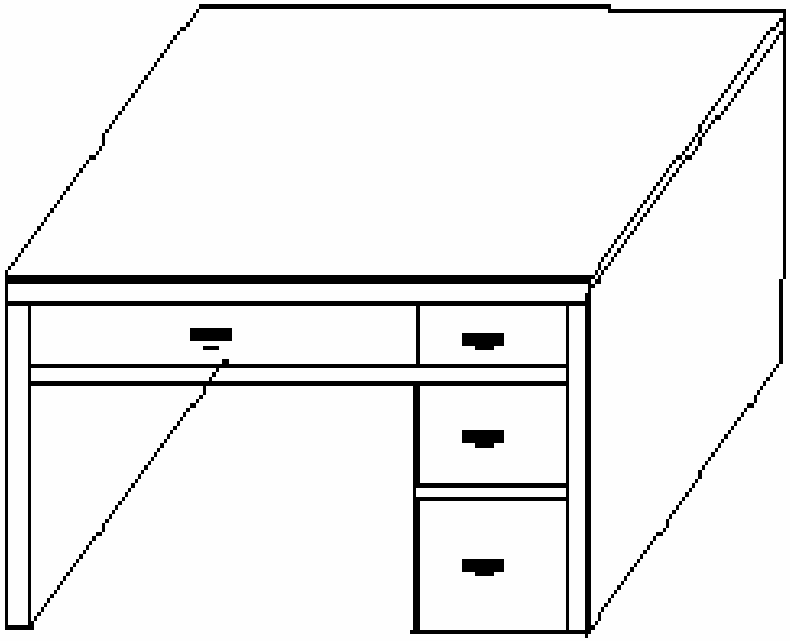
$\alpha \approx 63.4^\circ$

$\phi = 30^\circ$ *and* 45°

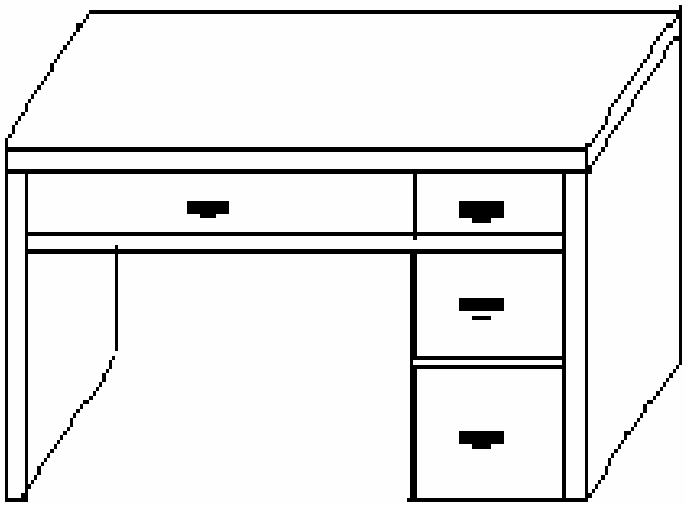
- Lines perpendicular to the viewing plane project at ½ of their **length**.
- A more realistic view than the cavalier projection.
- Can display a combination of front, and side, and top views.



Cavalier & Cabinet Projection



Cavalier

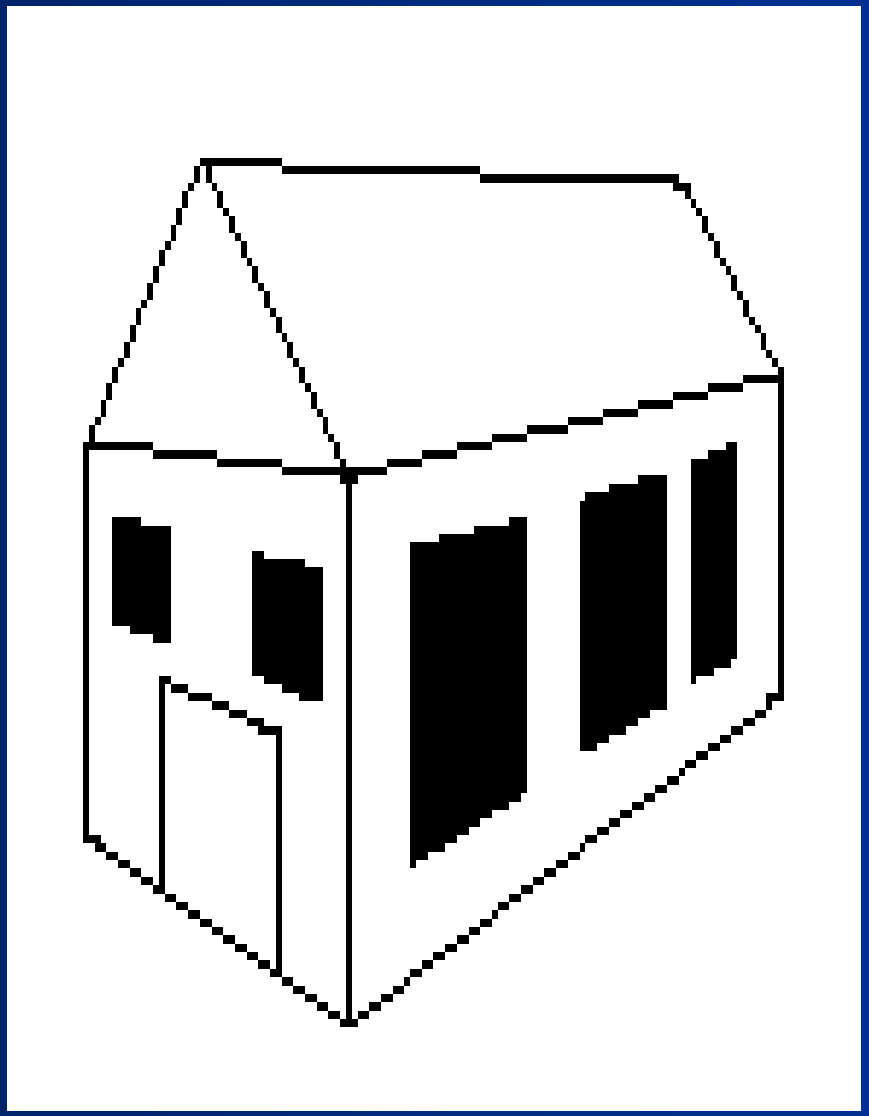


Cabinet

Perspective Projection

Henry Ford Int'l College, Kalanki, Kathmandu
By: Hari Prashad Pant

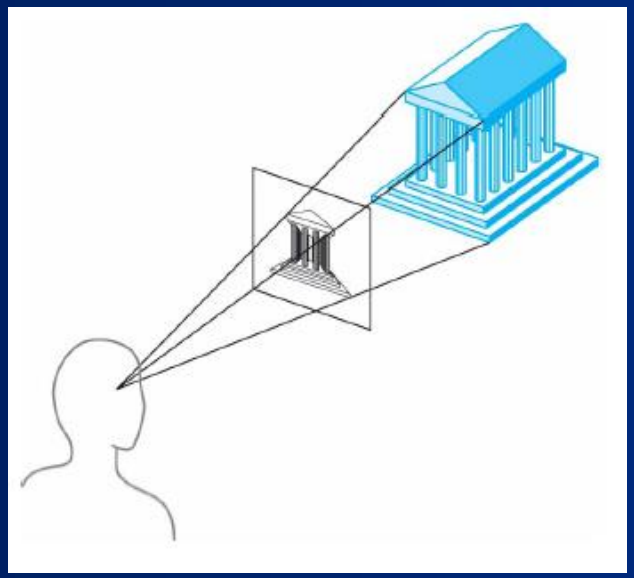
Perspective Projection

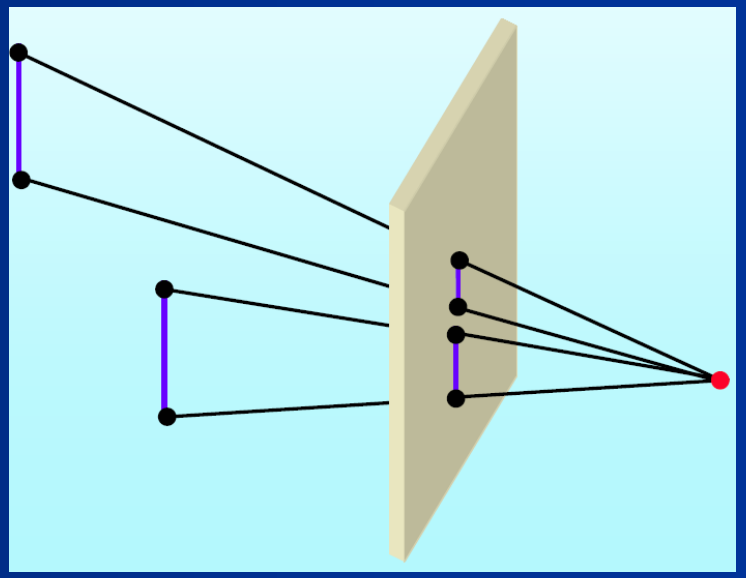


Henry Ford Int'l College, Kalanki, Kathmandu
By: Hari Prashad Pant

Perspective Projection

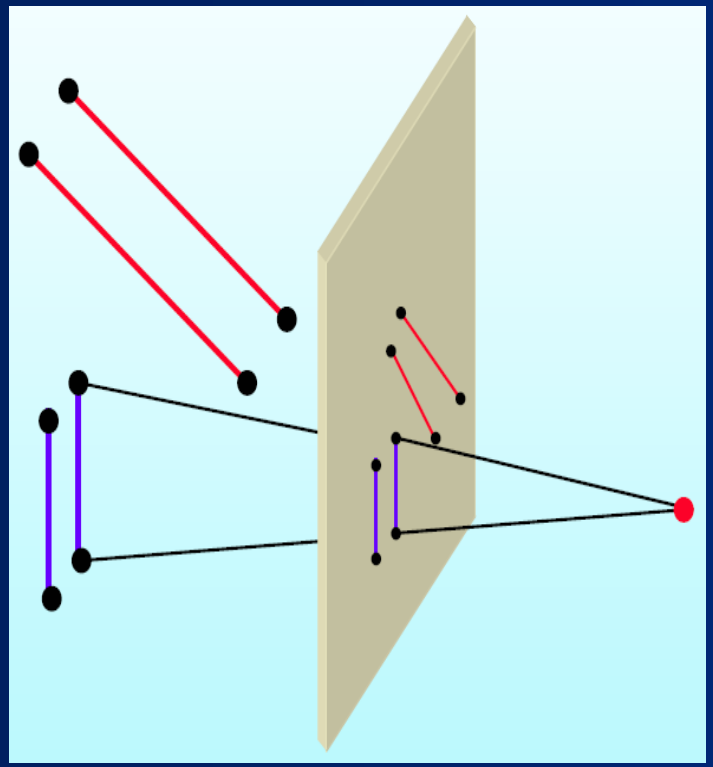
- In a perspective projection, the center of projection is at a finite distance from the viewing plane.
- Produces realistic views but does not preserve relative proportion of objects
- The size of a projection object is inversely proportional to its distance from the viewing plane.

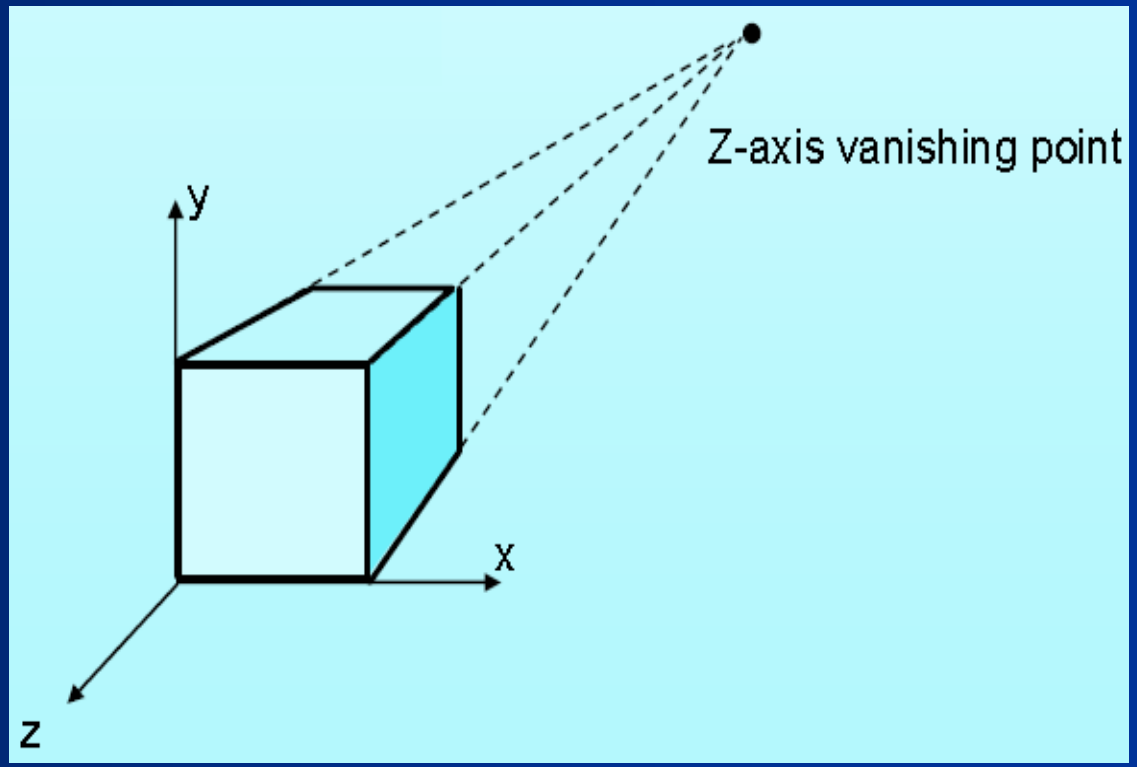




Perspective Projection

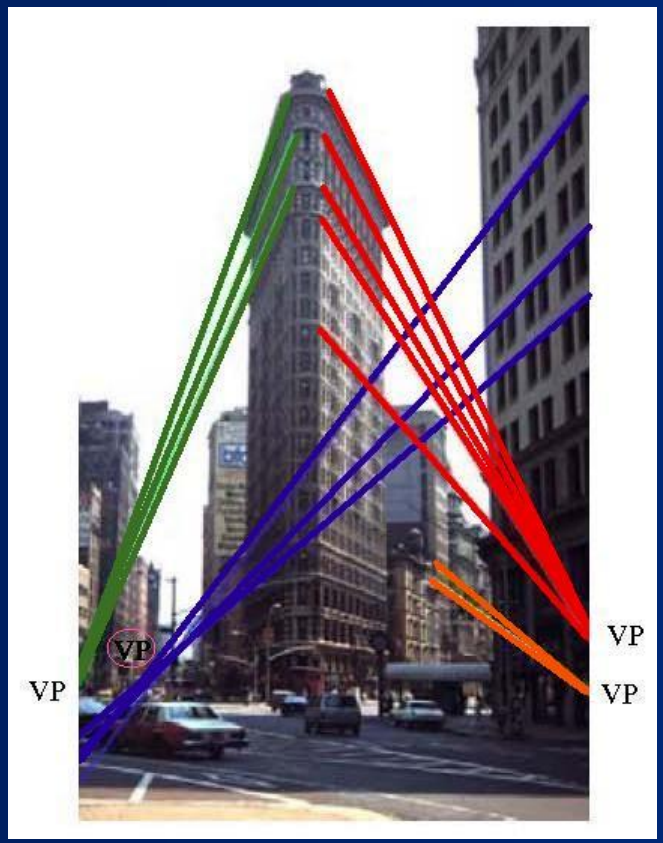
- Parallel lines that are not parallel to the viewing plane, **converge** to a ***vanishing point***.
- A vanishing point is the projection of a point at infinity.

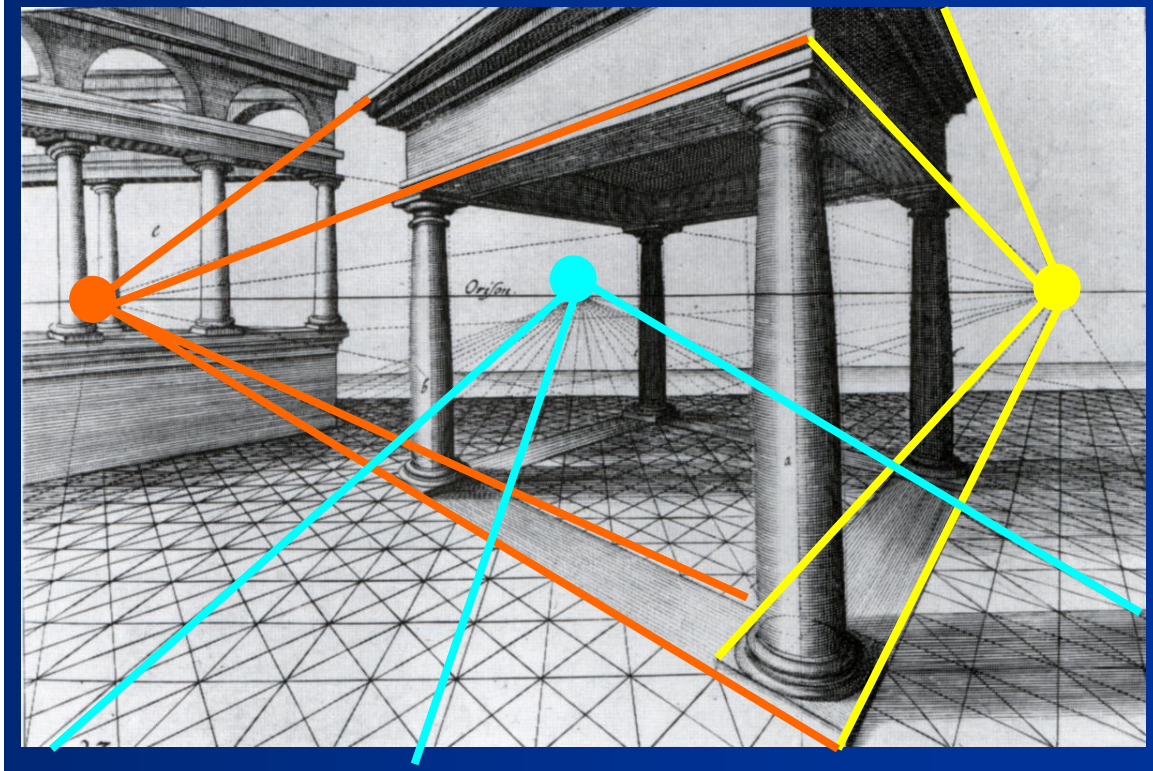




Vanishing Points

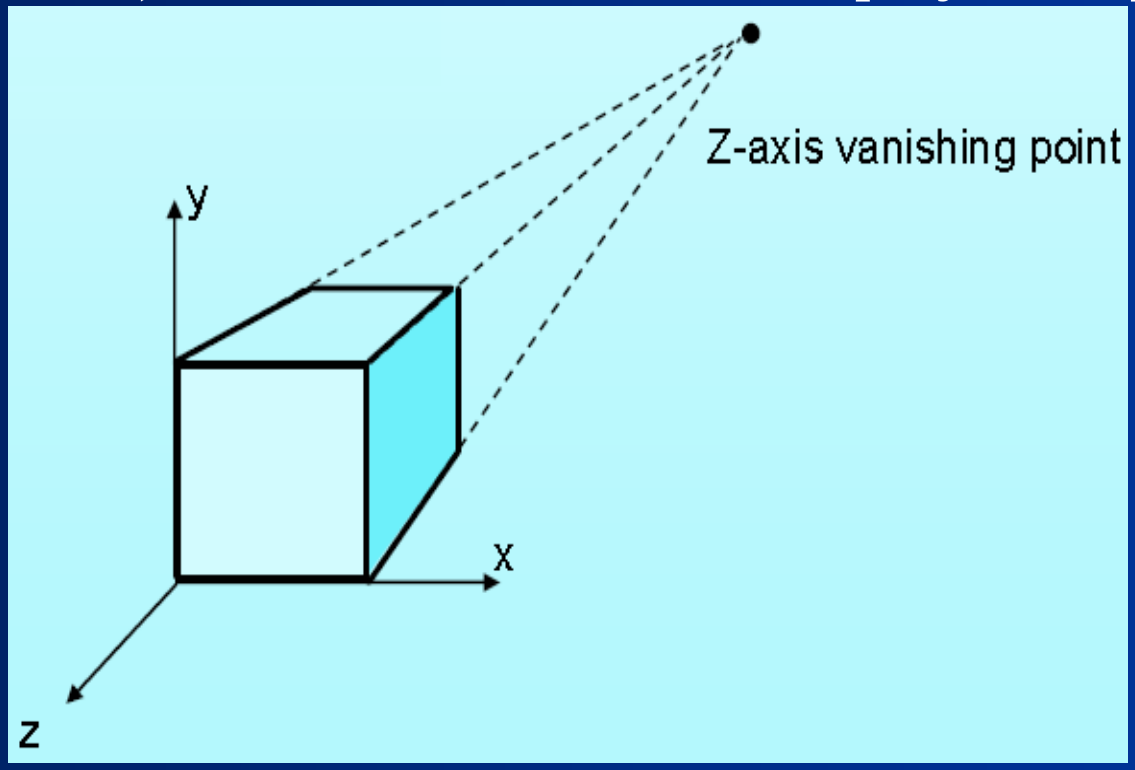
- Each set of projected parallel lines will have a separate vanishing points.
- There are infinity many **general** vanishing points.





Perspective Projection

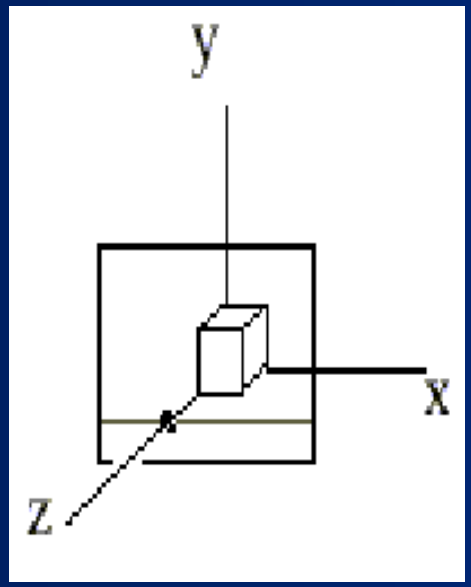
- The vanishing point for any set of lines that are parallel to one of the principal axes of an object is referred to as a **principal vanishing point**.
- We control the number of principal vanishing points (one, two, or three) with the orientation of the projection plane.

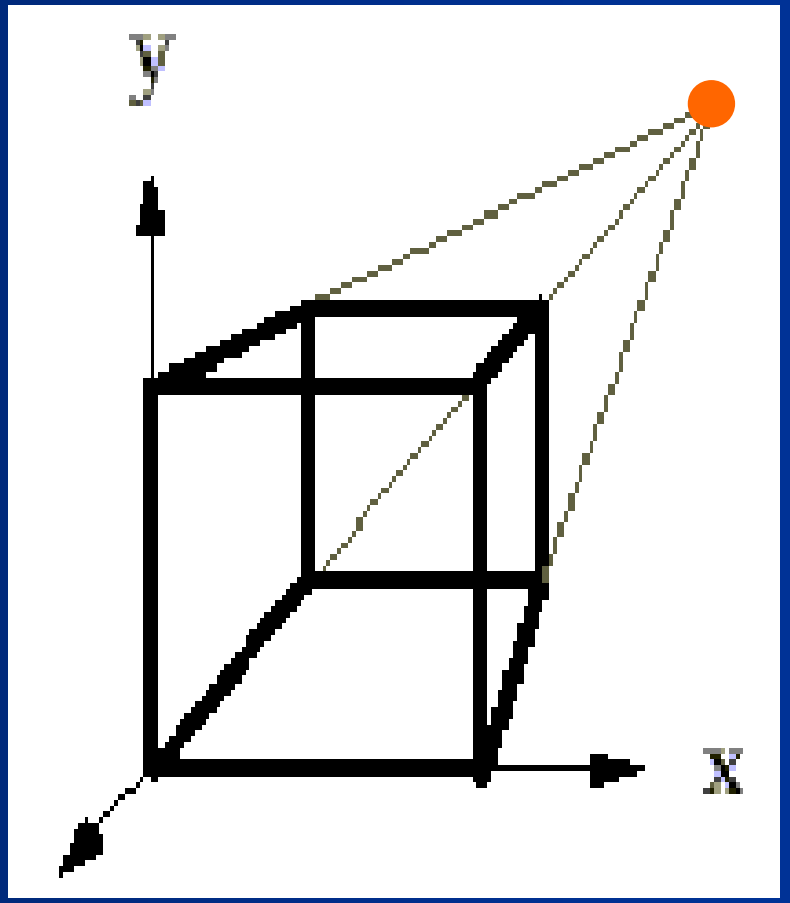


Perspective Projection

- The number of principal vanishing points in a projection is determined by the number of principal axes **intersecting** the view plane.

Perspective Projection

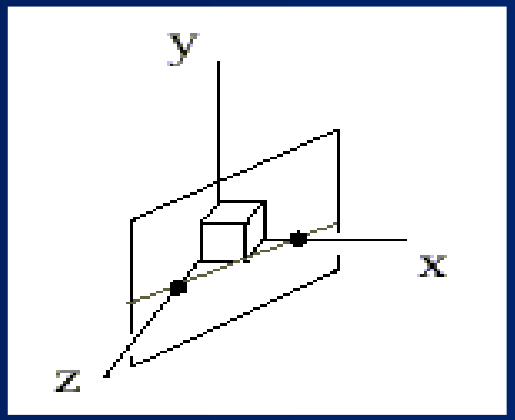


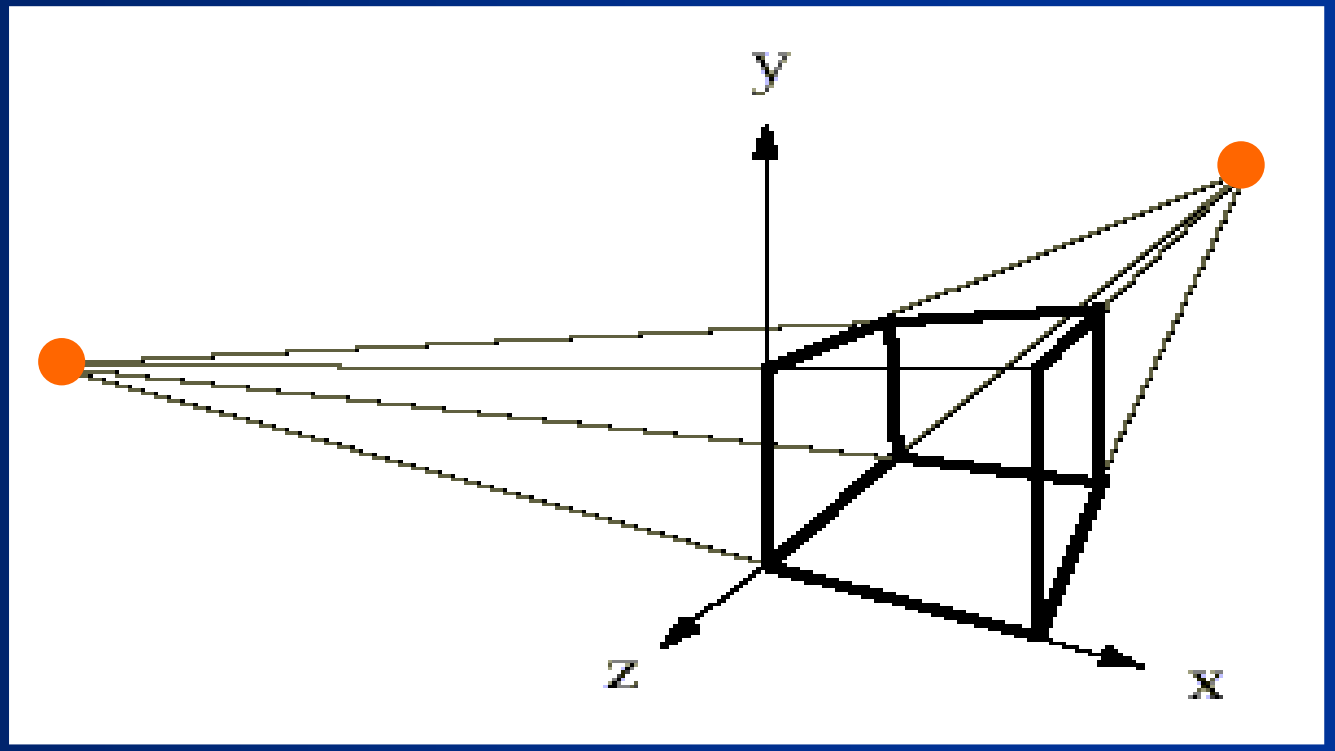


One Point Perspective
(*z*-axis vanishing point)

Henry Ford Int'l College, Kalanki, Kathmandu
By: Hari Prashad Pant

Perspective Projection



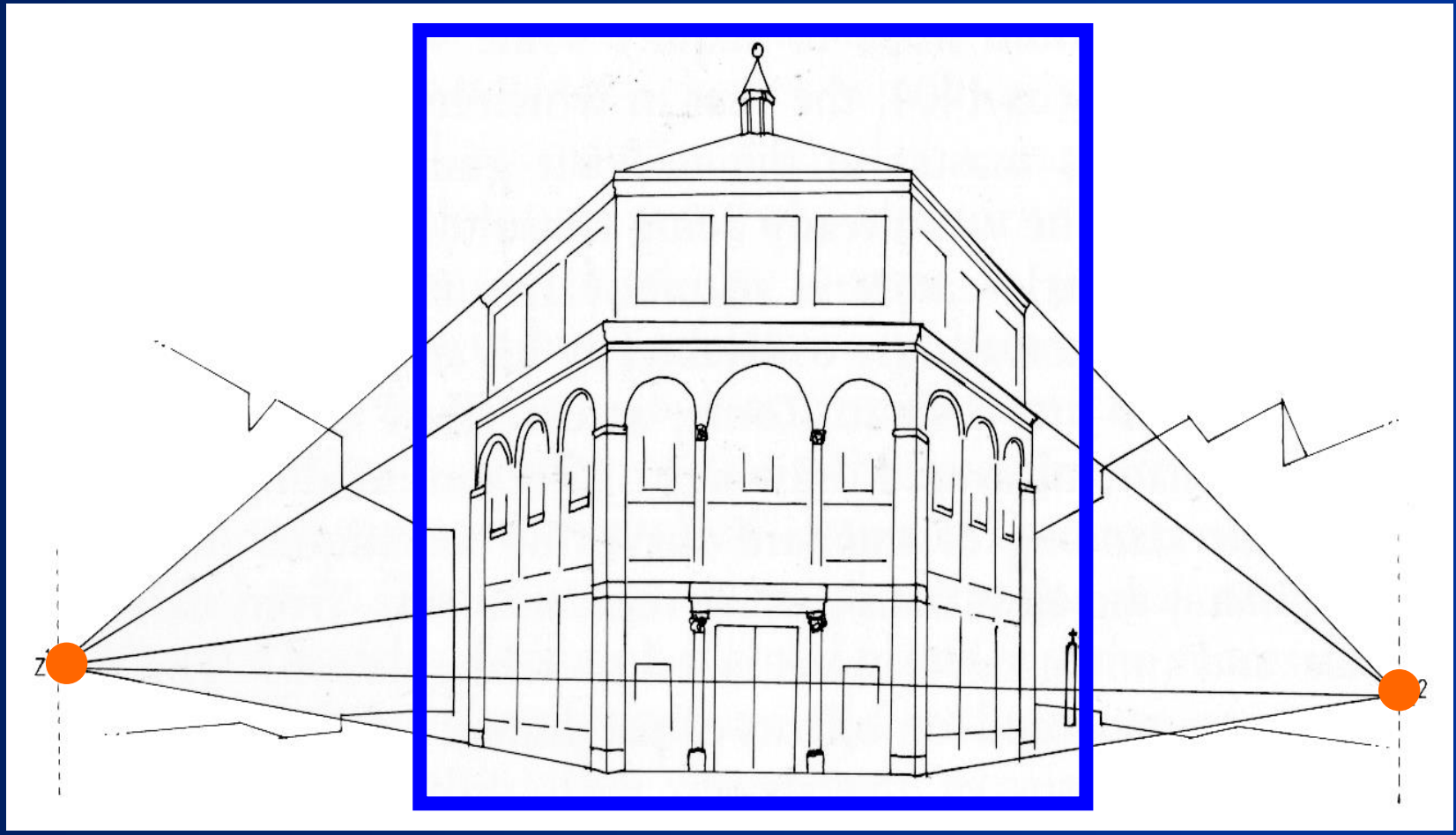


Two Point Perspective

(z, and x-axis vanishing points)

Henry Ford Int'l College, Kalanki, Kathmandu
By: Hari Prashad Pant

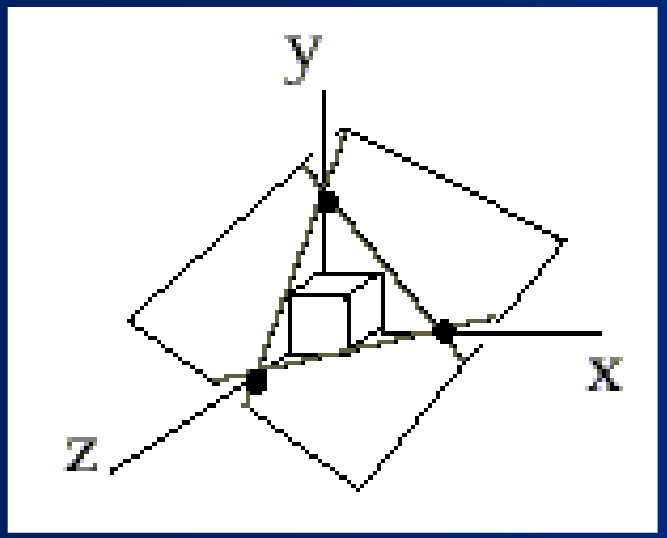
Perspective Projection

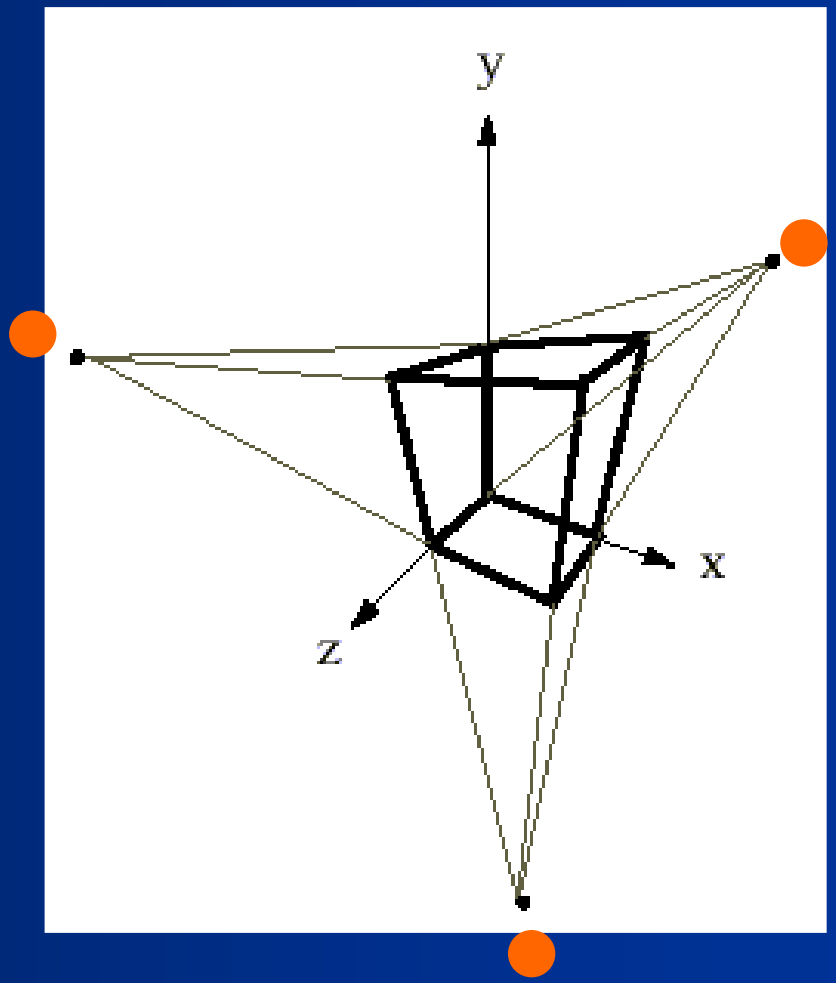


Two Point Perspective

Henry Ford Int'l College, Kalanki, Kathmandu
By: Hari Prashad Pant

Perspective Projection



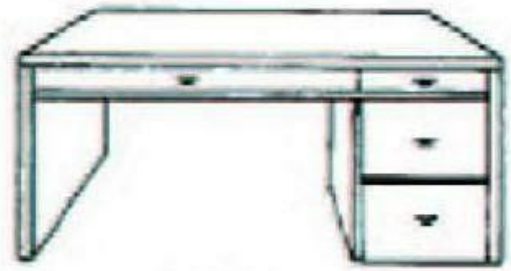


Three Point Perspective

(z, x, and y-axis vanishing points)

Henry Ford Int'l College, Kalanki, Kathmandu
By: Hari Prashad Pant

Perspective Projection



One-Point Perspective Projection

Two-Point Perspective Projection

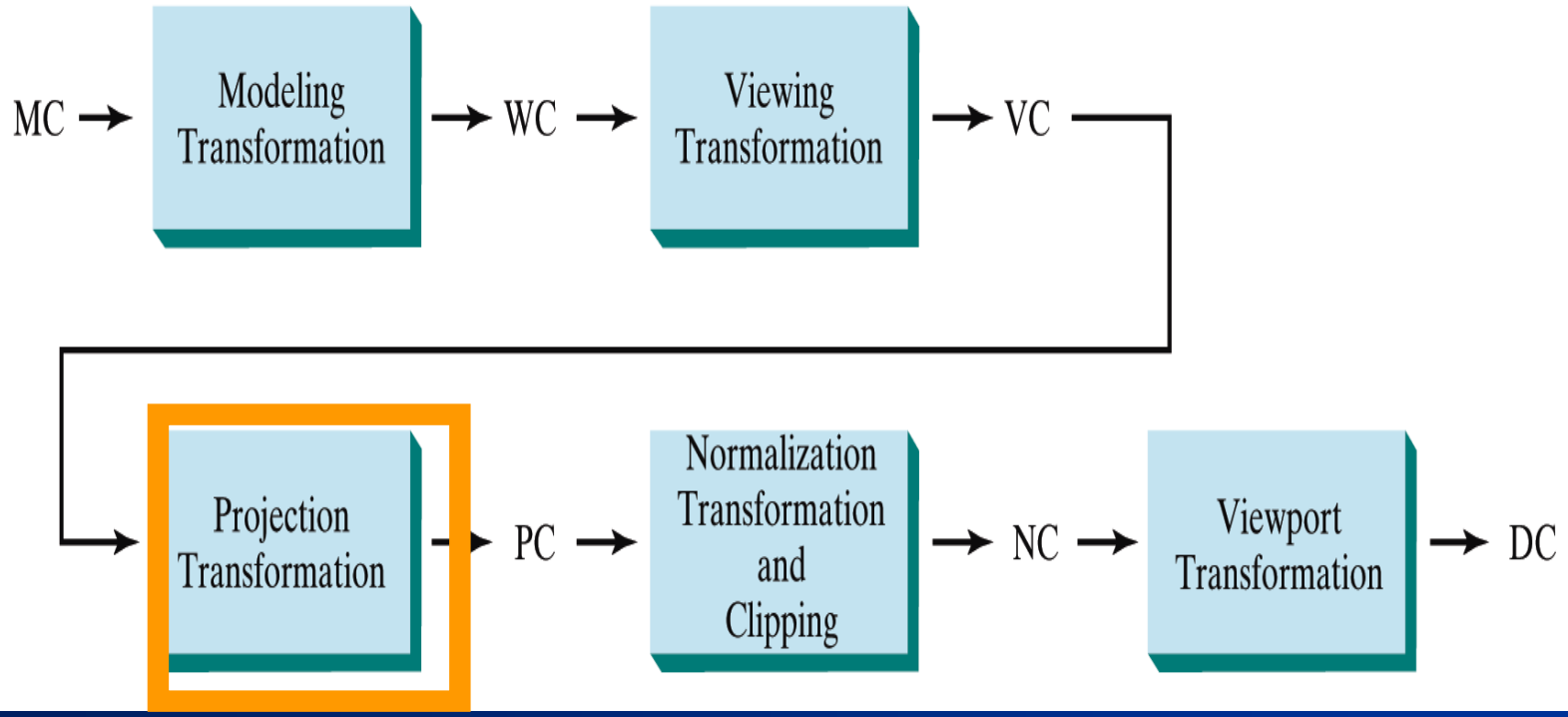
Tree-Point Perspective Projection

Perspective Projection Transformation

Henry Ford Int'l College, Kalanki, Kathmandu
By: Hari Prashad Pant

Perspective Projection Transformation

- Convert the **viewing coordinate** description of the scene to coordinate positions on the **perspective projection plane**.

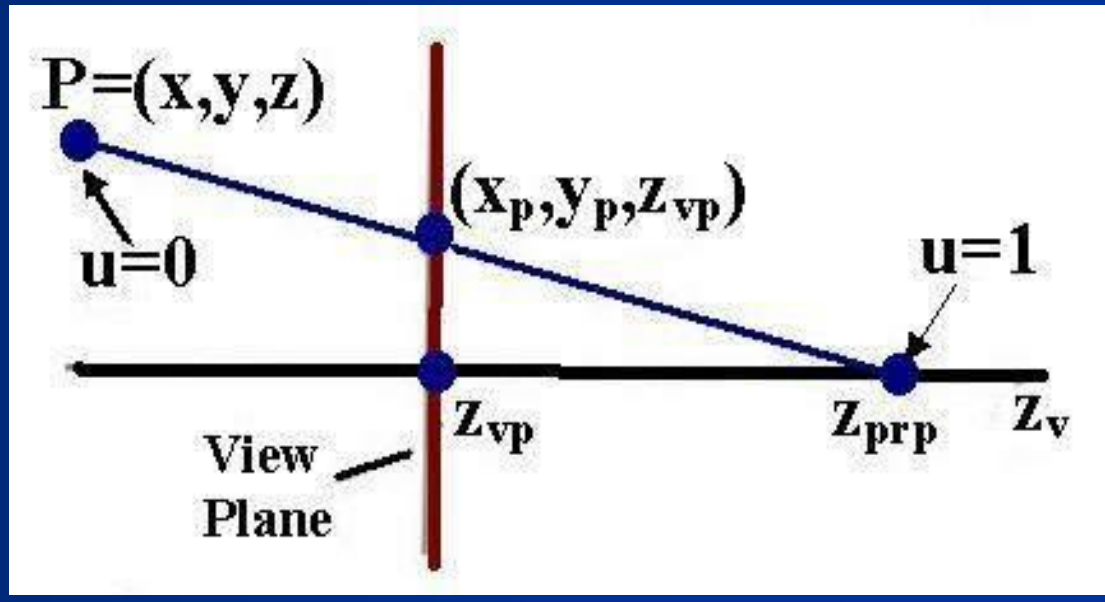


Perspective Projection Transformation

- Suppose the projection reference point at position z_{prp} along the z_v axis, and the view plane at z_{vp}.

$$x' = x - xu$$
$$y' = y - yu$$
$$z' = z - (z - z_{prp})u$$

$$0 \le u \le 1$$



Perspective Projection Transformation

On the view plane: $z' = z_{vp}$

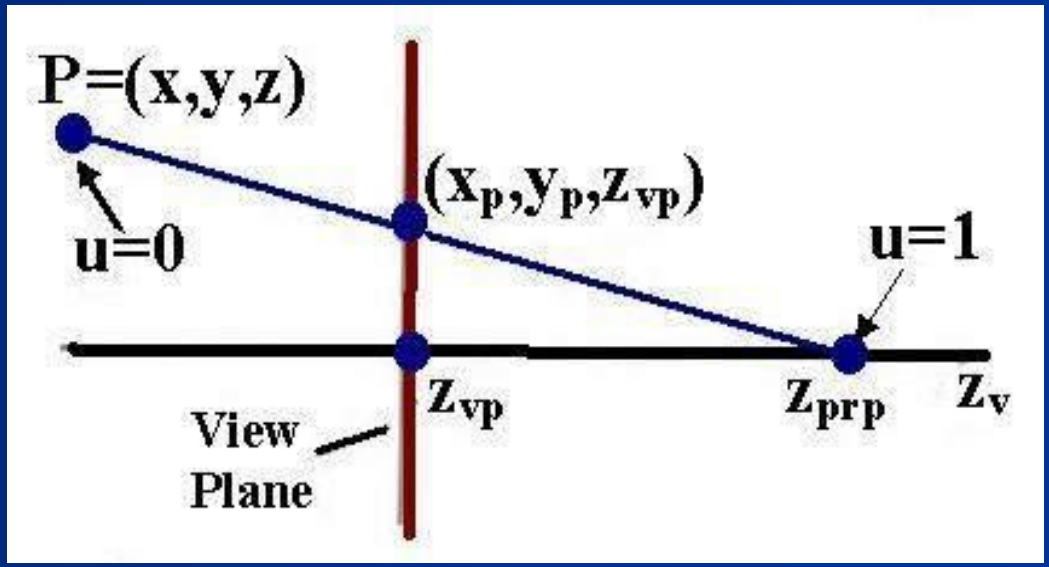
$$u = \frac{z_{vp} - z}{z_{prp} - z}$$

$$x' = x - xu$$
$$y' = y - yu$$
$$z' = z - (z - z_{prp})u$$

$$x_p = x\left(\frac{z_{prp} - z_{vp}}{z_{prp} - z}\right) = x\left(\frac{d_p}{z_{prp} - z}\right)$$

$$y_p = y\left(\frac{z_{prp} - z_{vp}}{z_{prp} - z}\right) = y\left(\frac{d_p}{z_{prp} - z}\right)$$

$$d_p = z_{prp} - z_{vp}$$



Perspective Projection Transformation

On the view plane: $z' = z_{vp}$

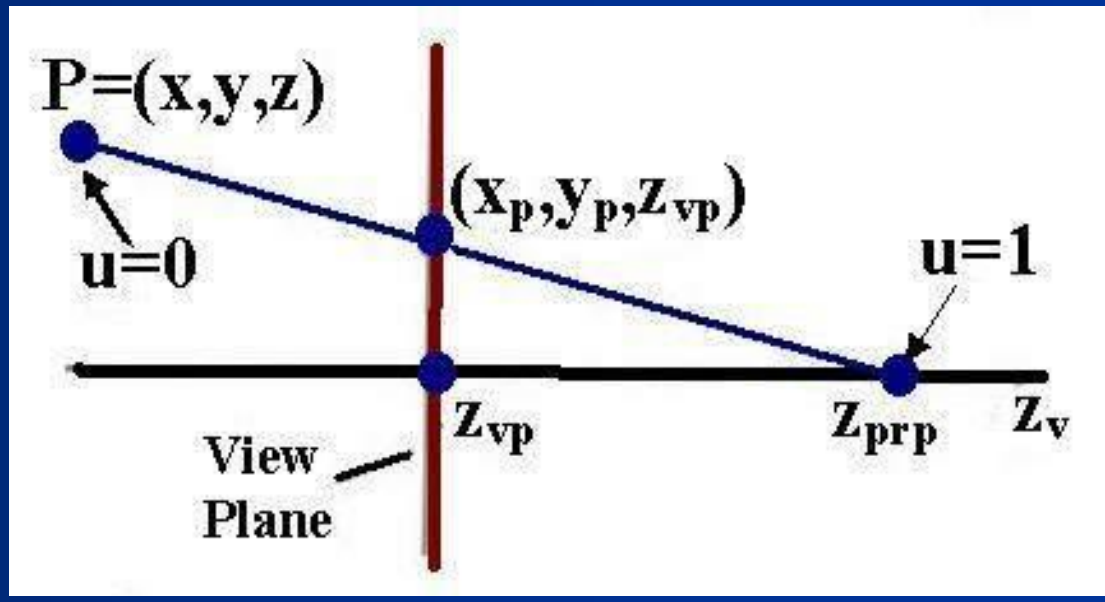
$$\begin{bmatrix} x_h \\ y_h \\ z_h \\ h \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -z_{vp}/d_p & z_{vp}(z_{prp}/d_p) \\ 0 & 0 & -1/d_p & z_{prp}/d_p \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$x_p = x\left(\frac{z_{prp}-z_{vp}}{z_{prp}-z}\right) = x\left(\frac{d_p}{z_{prp}-z}\right)$$

$$y_p = y\left(\frac{z_{prp}-z_{vp}}{z_{prp}-z}\right) = y\left(\frac{d_p}{z_{prp}-z}\right)$$

$$h = \frac{z_{prp} - z}{d_p}$$

$$x_p = x_h/h, \quad y_p = y_h/h$$



Perspective Projection Transformation

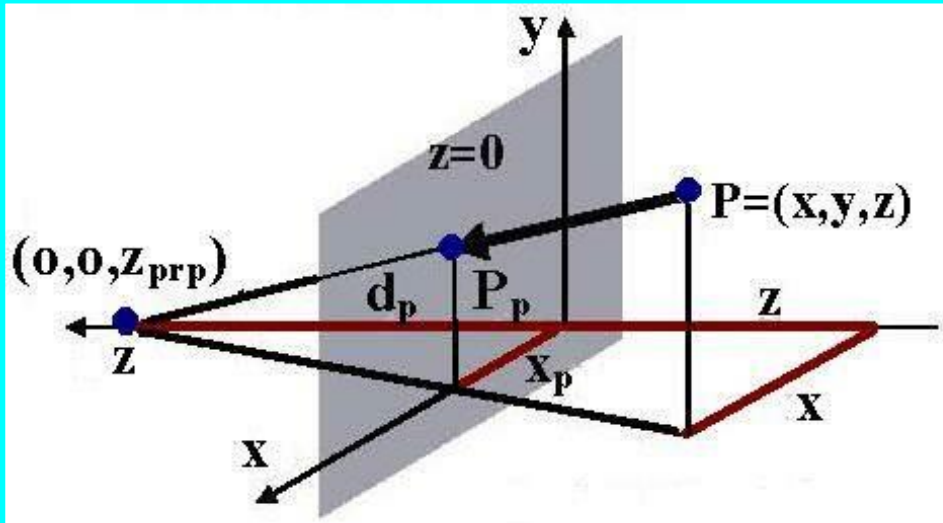
Special Cases: $z_{vp} = 0$

$$x_p = x\left(\frac{z_{prp}}{z_{prp}-z}\right) = x\left(\frac{1}{1-z/z_{prp}}\right)$$

$$y_p = y\left(\frac{z_{prp}}{z_{prp}-z}\right) = y\left(\frac{1}{1-z/z_{prp}}\right)$$

$$x_p = x\left(\frac{z_{prp}-z_{vp}}{z_{prp}-z}\right) = x\left(\frac{d_p}{z_{prp}-z}\right)$$

$$y_p = y\left(\frac{z_{prp}-z_{vp}}{z_{prp}-z}\right) = y\left(\frac{d_p}{z_{prp}-z}\right)$$



Perspective Projection Transformation

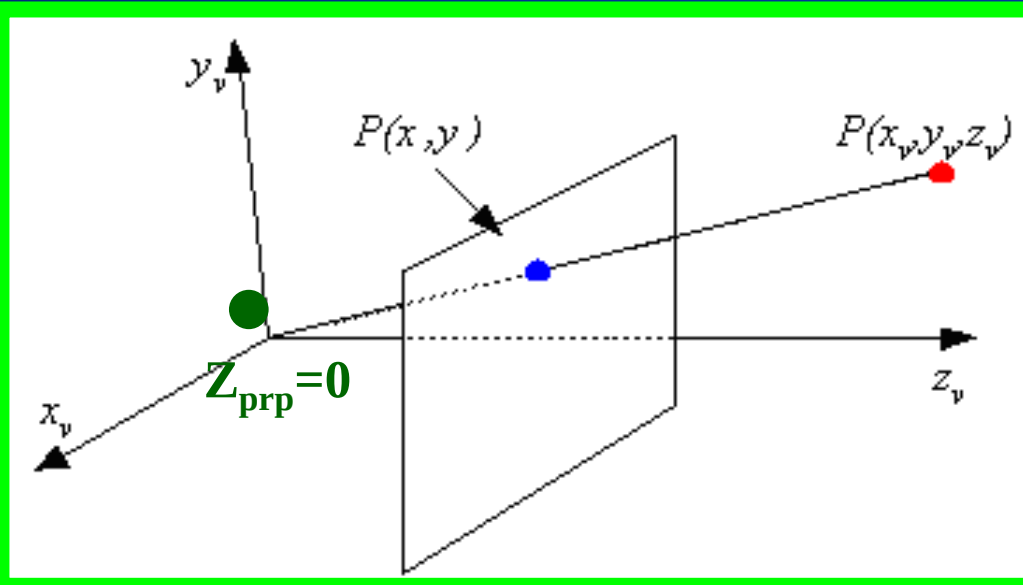
Special Cases: The projection reference point is at the viewing coordinate origin: $z_{prp} = 0$

$$x_p = x\left(\frac{z_{vp}}{z}\right) = x\left(\frac{1}{z/z_{vp}}\right)$$

$$y_p = y\left(\frac{z_{vp}}{z}\right) = y\left(\frac{1}{z/z_{vp}}\right)$$

$$x_p = x\left(\frac{z_{prp}-z_{vp}}{z_{prp}-z}\right) = x\left(\frac{d_p}{z_{prp}-z}\right)$$

$$y_p = y\left(\frac{z_{prp}-z_{vp}}{z_{prp}-z}\right) = y\left(\frac{d_p}{z_{prp}-z}\right)$$



References

- **Foley, J. D., A. V. Dam, S. K. Feiner, J. F. Hughes**, ***"Computer Graphics Principle and Practices"***, Addison Wesley Longman, Singapore Pvt. Ltd., 1999
- **Hearn Donald, M. P. Baker**, ***"Computer Graphics, 2nd Ed."***, Prentice Hall of India Private Limited, New Delhi, 2000
- **Hearn, Baker & Carithers**, ***"Computer Graphics with OpenGL, 4th Ed."***, Pearson Education Limited 2014