2D Representations and Transformations

Overview

Coordinate Representations

Definition

2 Dimensional Geometric Transformations

- Translation
- Rotation
- Scaling

Homogeneous Coordinates

Reflection

Shearing

Coordinate Reprsentations

A **Cartesian coordinate system** specifies each point uniquely in a plane by a pair of numerical **coordinates**, which are the signed distances from the point .

Graphics package are designed to use with Cartesian coordinate specifications.

Several different Cartesian reference frames are used to construct and display a scene.

The geometric part of the rendering process is that it consists of the application of a series of coordinate transformations that takes an object database through a series of coordinate systems.

Coordinate Representations

Local or Modelling Coordinate system :For ease of modeling store the vertices of an object with respect to some point conveniently located in or near the object.

Ex:Construct the individual objects such as trees or furniture in a scene within separate coordinate reference frames .

Once an object has been modeled, the next stage is to place it in the scene that we wish to render

The global coordinate system of the scene is known as the *world coordinate system*

Coordinate Representations

The world coordinate description of the scene is transferred to one or more output device reference frames for display called device coordinates.

Modeling and world coordinate definitions allow us to any convenient dimensions.

Graphics system first converts the world coordinate positions to normalized device coordinates in the range of 0 to 1 before final conversion to specific device coordinates.

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

Geometric Transformation Definition

In many applications there is need for altering and manipulating displays.

Geometric Transformations: Operations that are applied to change the geometric description of an object by changing its position, orientation, or size.

The basic transformations are translation, rotation and scaling.

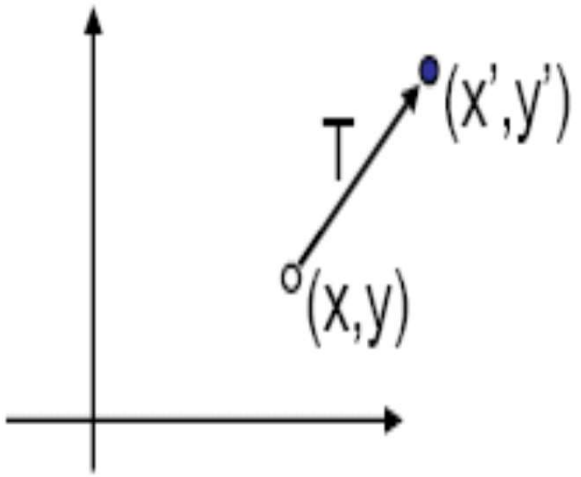
TRANSLATION

Translation is applied to an object by repositioning it along a line path from one coordinate location to another.

Translate a two dimensional point by adding translation distances t_x , t_y

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

The translation distance pair (tx, ty) is the translation vector or shift vector.



TRANSLATION

Translation equations can be expressed as a single matrix equation

$$P = \begin{bmatrix} x \\ y \end{bmatrix} P' = \begin{bmatrix} x' \\ y' \end{bmatrix} T = \begin{bmatrix} tx \\ ty \end{bmatrix}$$

2D translation equation

$$P' = P + T$$

TRANSLATION

Rigid body transformation → moves object without deformation

Every point is translated by the same amount

Straight line segment is translated by applying the transformations to each of the line endpoints and redrawing the line between the new endpoint positions.

A triangle with position (10,2), (20,2) and (15,5) is translated with the translation vector (-5.5,3.75). Determine the new positions of the triangle.

2D ROTATION

A rotation transformation of an object is generated by specifying a rotation axis and a rotation angle.

All points of the object are then transformed to new positions by rotating the points through the specified angle about the rotation axis.

2D rotation is obtained by repositioning the object along a circular path in the *xy* plane.

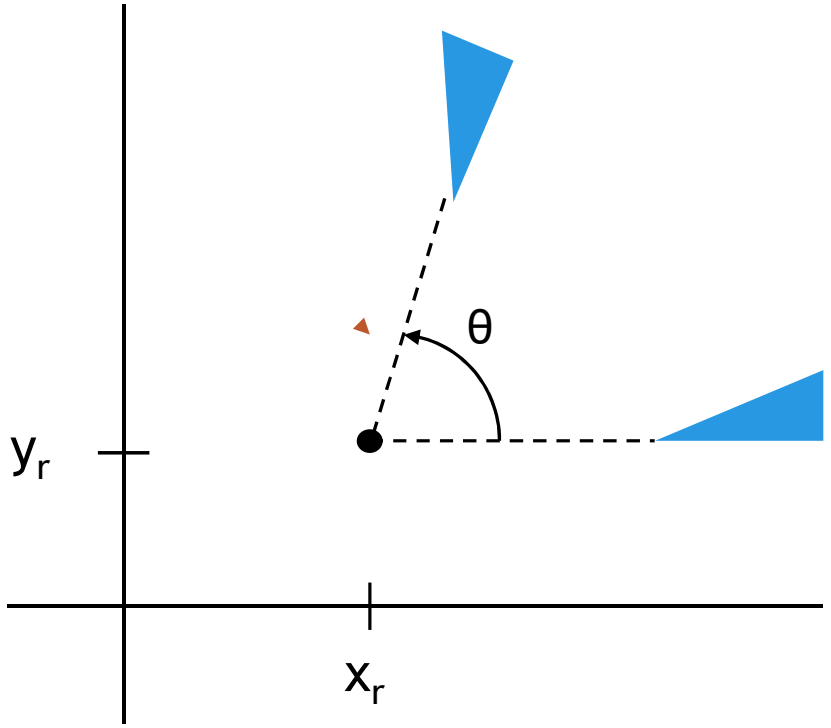
2D ROTATION

Parameter for 2D rotation:

- Rotation angle, θ
- Rotation point (pivot point), (xr, yr)

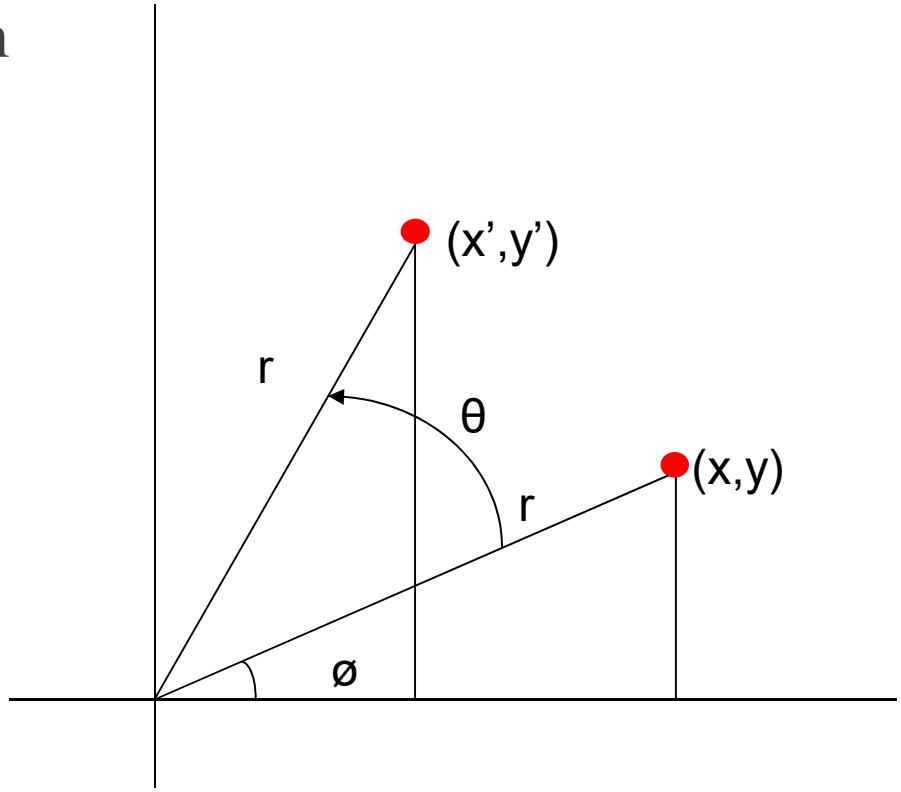
Positive θ >> counterclockwise rotation about the pivot point

Negative θ >> clockwise rotation about the pivot point



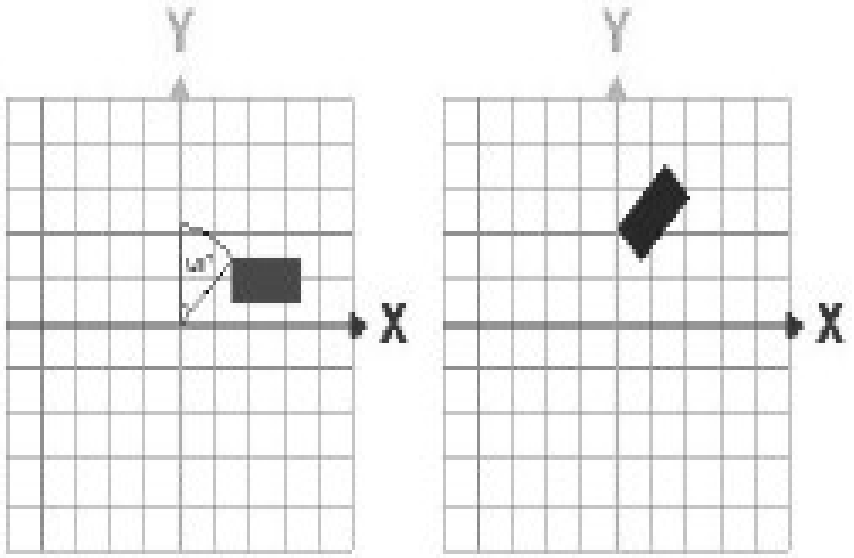
2D ROTATION

- Rotation of a point from position (x, y) to position (x', y') through an angle θ relative to the coordinate origin. The original angular displacement of the point from the x axis is Φ



2D ROTATION

Rotation by 45° counter-clockwise about origin.



2D ROTATION

Using standard trigonometric identities, transformed coordinates can be expressed in terms of angles θ and Φ as

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

$$y' = r\sin(\Phi + \theta)$$
$$= r\cos\Phi\sin\theta + r\sin\Phi\cos\theta$$

The original coordinates of the point in polar coordinates are

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

2D ROTATION

Substituting expression (5) into (4), we obtain the transformation equations for rotating a point at position (x, y) through an angle θ about the origin:

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

Rotation equation in matrix form, P' = R . P where the rotation matrix is

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

2D ROTATION

- Rotation of a point about an arbitrary pivot point.

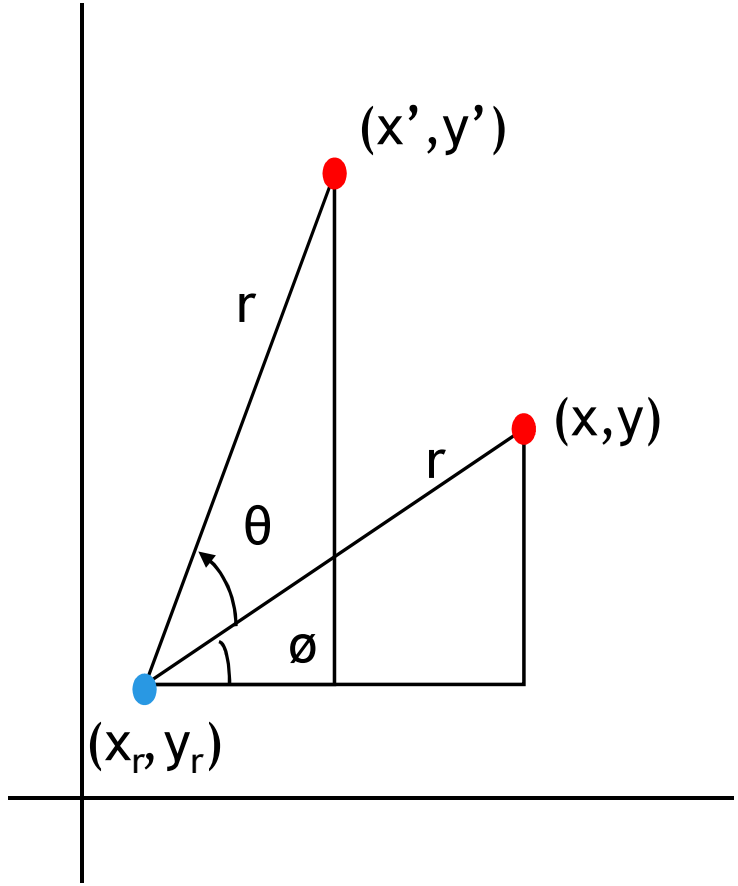
  For rotation of a point about any specified rotation position (x_r, y_r):

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

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

  Rotations are also rigid body transformations that move object without deformation

  Every point in the object is rotated through the same angle



2D SCALING

To change the size of an object.

A simple operation is by multiplying object positions (x, y) by scaling factors s_x and s_y to produce the transformed coordinates (x',y') :

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

can also be written in matrix form

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

or $\quad P' = S \,.\, P$

2D SCALING

Any positive value can be assigned to scaling factors s_x and s_y

Values less than 1 reduce the size and greater than 1 enlarge it.

Specifying a value of 1 for both s_x and s_y leaves the size unchanged.

Uniform scaling: maintain relative object proportions (size) when s_x and s_y is assigned same value.

2D SCALING

Differential scaling: applying unequal values for s_x and s_y.

- Often use in design applications, where pictures are constructed from few basic shape that can be adjusted by scaling and positioning transformations .

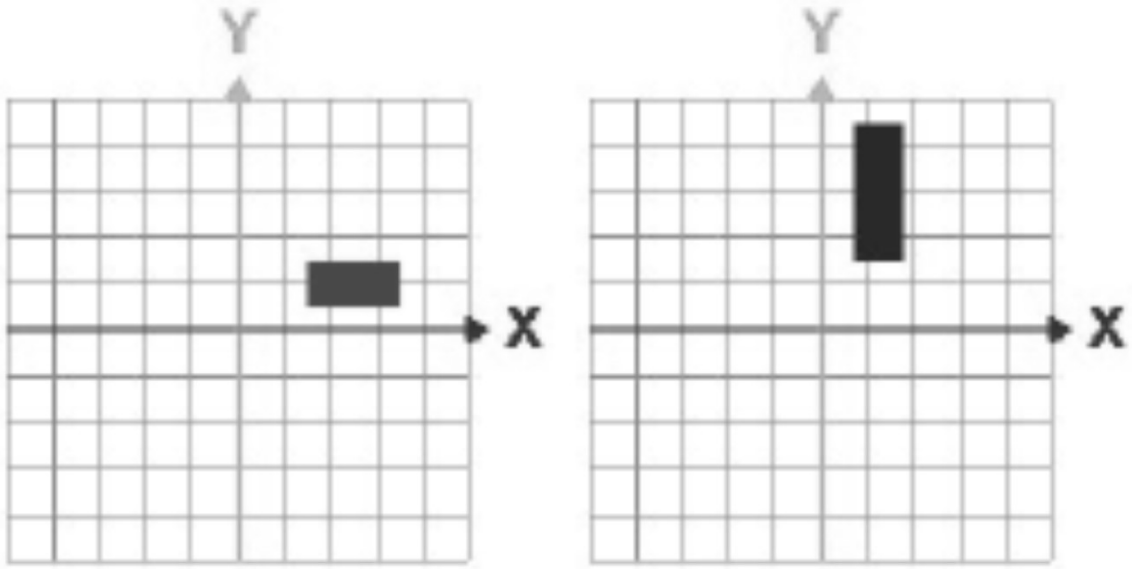
Objects transformed are BOTH scaled and repositioned.

Scaling factor:

- |<1| - move objects closer to origin
- |>1| - move objects farther from the origin

2D SCALING

- Scaling vector : (0.5, 3.0) about origin.



2D SCALING

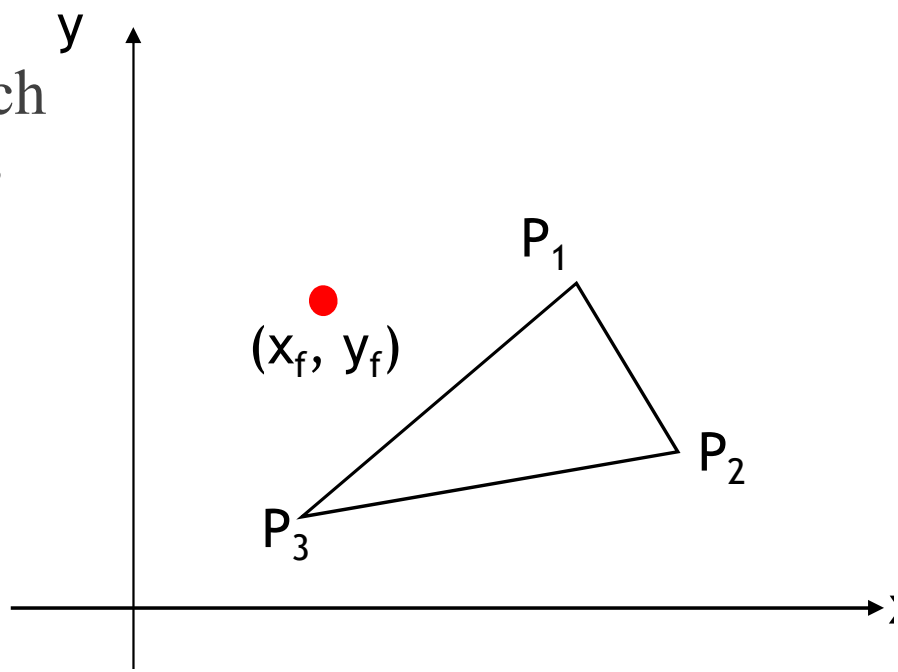
The location of the scaled object can be controlled by choosing a position, fixed point, that is to remain unchanged after the transformation.

The coordinate for fixed point (x_f, y_f) are often chosen at some object position, but any other position can be selected.

Objects are now resized by scaling the distances between object points and the fixed point.

2D SCALING

- Scaling relative to a chosen fixed point (xf,yf). The distance from each polygon vertex to the fixed point is scaled by transformation equation (13).



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

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

Summary

Local and World Coordinates

Cartesian and Polar coordinate system

Geometric transformations

- Rigid body transformations

2D Transformations - Reflection and Shearing
2D Composite Transformations

Overview

Homogeneous Coordinates

Reflection

Shearing

HOMOGENEOUS CO-ORDINATES

Graphics applications involves sequences of geometric transformations.

Efficient approach is needed to combine the transformations so that the final coordinates are obtained directly.

Combine the multiplicative and the translational terms for 2d geometric transformations into single matrix multiplication by homogenous coordinates.

Homogeneous coordinates seem unintuitive, but they make graphics operations much easier

Represent each 2D coordinate position (x, y) with the homogenous coordinate triple (x_h, y_h, h).

HOMOGENEOUS CO-ORDINATES

Represent each 2D coordinate position (x, y) with the homogenous coordinate triple (x_h, y_h, h). Where

$$x = \frac{x_h}{h} \quad y = \frac{y_h}{h} \quad P = \begin{bmatrix} x_h \\ y_h \\ h \end{bmatrix} = \begin{bmatrix} h \cdot x \\ h \cdot y \\ h \end{bmatrix}$$

General homogeneous representation can also written as (h.x,h.y,h) set h=1.

Transformations of translation, scaling and rotation can be represented using Homogeneous coordinates.

Homogeneous Transformation Coordinates

Translation

$$T(t_x, t_y) = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

$$P' = T(t_x, t_y) \cdot P$$

Rotation

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

$$P' = R(\theta) \cdot P$$

Scaling

$$S(s_x, s_y) = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$P' = S(s_x, s_y) \cdot P$$

Composite Transformations

Application of a sequence of transformations to a point:

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

Composite transformations is formed by calculating the matrix product of the individual transformations and forming products of transformation matrix.

Composite Transformations- Translation

First: composition of similar type transformations

If we apply to successive translations to a point:

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

P AND P' are represented as homogenous coordinate values.

$$T(t_{2x}, t_{2y}) \cdot T(t_{1x}, t_{1y}) = \begin{bmatrix} 1 & 0 & t_{2x} \\ 0 & 1 & t_{2y} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & t_{1x} \\ 0 & 1 & t_{1y} \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_{1x} + t_{2x} \\ 0 & 1 & t_{1y} + t_{2y} \\ 0 & 0 & 1 \end{bmatrix} = T(t_{1x} + t_{2x}, t_{1y} + t_{2y})$$

Successive translations are additive

Composite Transformations-Rotation

Two successive rotations applied to the point p produce the transformed position

P'=R(θ) {R (Φ).P}={R(θ) .R (Φ)}.P

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

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

Two successive rotations are additive.

Composite Transformations-Scaling

Two successive scaling operations produces the following composite scaling matrix

$$\mathbf{S}(s_{2x}, s_{2y}) \cdot \mathbf{S}(s_{1x}, s_{1y}) = \begin{bmatrix} s_{2x} & 0 & 0 \\ 0 & s_{2y} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} s_{1x} & 0 & 0 \\ 0 & s_{1y} & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} s_{1x} \cdot s_{2x} & 0 & 0 \\ 0 & s_{1y} \cdot s_{2y} & 0 \\ 0 & 0 & 1 \end{bmatrix} = \mathbf{S}(s_{1x} \cdot s_{2x}, s_{1y} \cdot s_{2y})$$

•The resulting matrix indicates the successive operations are multiplicative.

Composite Transformations

Combining transformations reduces to matrix multiplication, e.g.

- R(r,) = T(r)* R(θ)* T(-r)

In general: multiplication of a 3x3 with another 3x3 matrix requires 3*3*3 = 27 multiplications and 2*3*3 additions.

In 2D transformations, the third row of the matrices is always [0 0 1] and should never be calculated.

In addition, in homogeneous coordinates the third component of the vectors is always one: (x,y,1).

Composite converts all to matrix multiplications thus improving computational efficiency

Rotation around a pivot point

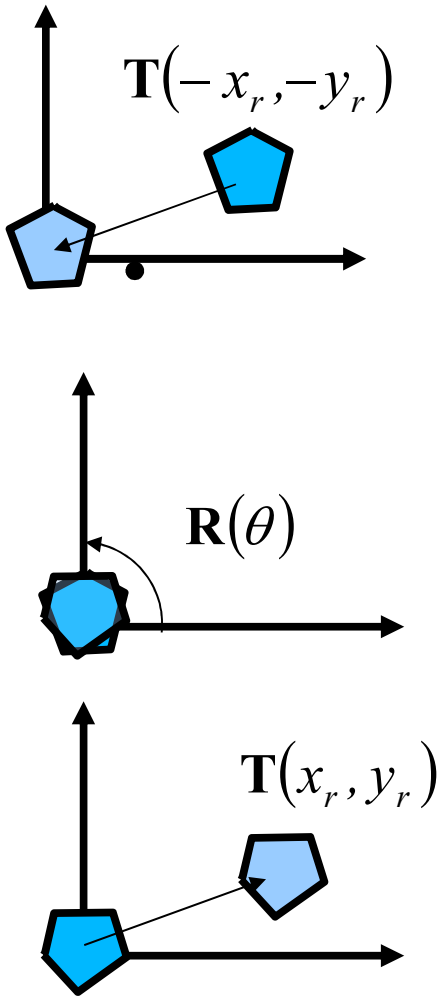
Rotations about any selected pivot point (x_r, y_r) by performing the following sequence:

- Translate the object so that the pivot point moves to the origin
- Rotate around origin
- Translate the object so that the pivot point is back to its original position

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

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

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



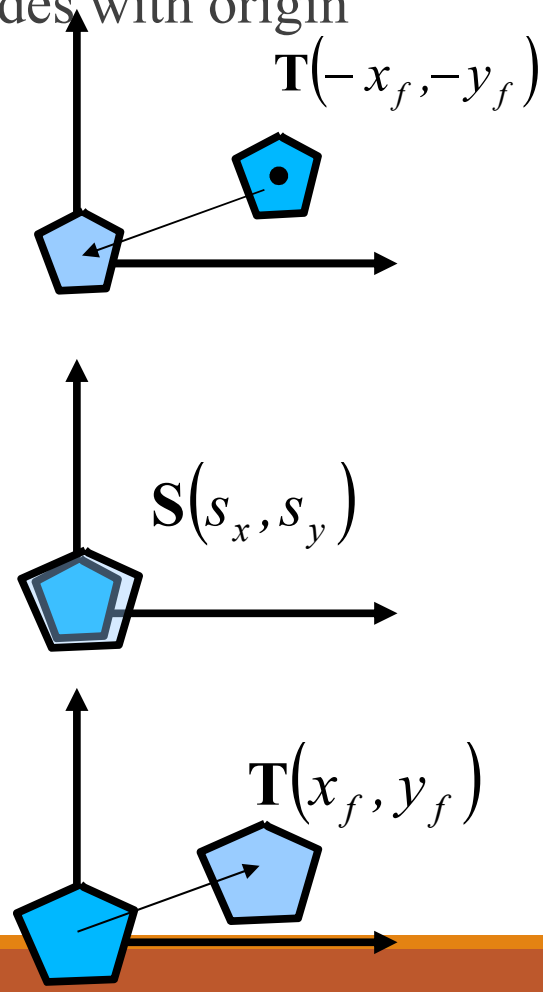
Scaling with respect to a Fixed Point

- Translate object to origin so fixed point coincides with origin
- Scale the object with respect to origin
- Translate back by inverse translation.

$$\mathbf{T}(x_f, y_f) \cdot \mathbf{S}(s_x, s_y) \cdot \mathbf{T}(-x_f, -y_f) =$$

$$\begin{bmatrix} 1 & 0 & x_f \\ 0 & 1 & y_f \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -x_f \\ 0 & 1 & -y_f \\ 0 & 0 & 1 \end{bmatrix} =$$

$$\begin{bmatrix} s_x & 0 & x_f(1-s_x) \\ 0 & s_y & y_f(1-s_y) \\ 0 & 0 & 1 \end{bmatrix}$$

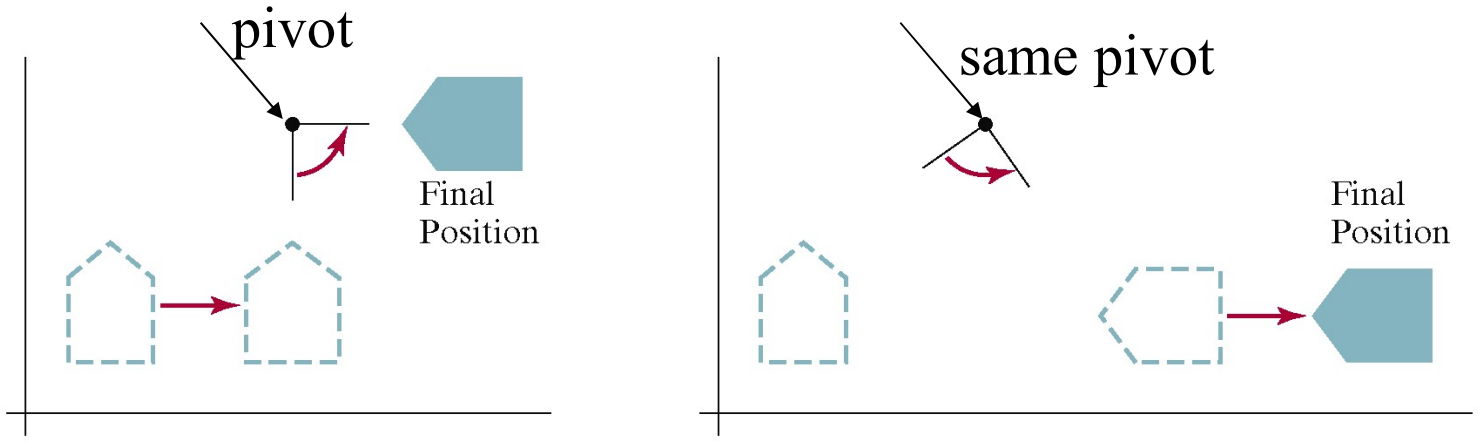


Concatenation Properties

Matrix multiplication is associative, evaluate matrix products using left-to-right or right-to-left associative grouping.

Matrix composition is not commutative. So careful when applying a sequence of transformations.

Reversing the order in which the sequence of transformations is performed may affect the transformed position of an object.



REFLECTION

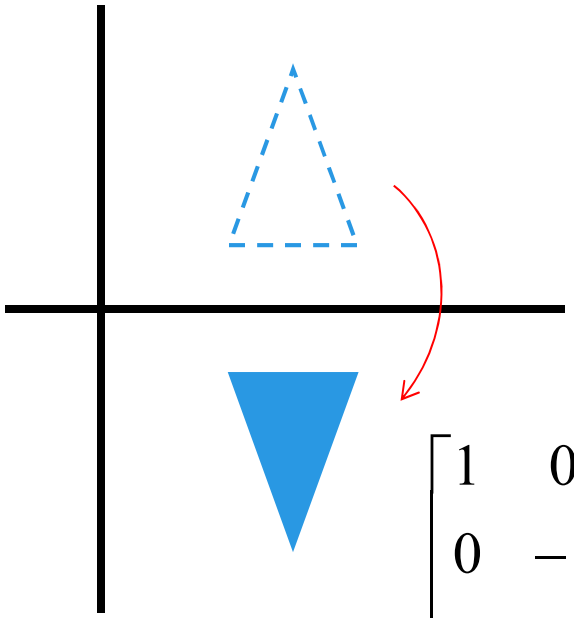
A transformation that produces a mirror image of an object

Image is generated relative to an axis of reflection by rotating the object 180° about the reflection axis

- *Reflection axis is* *xy* plane – rotation path about the axis is in the plane perpendicular to *xy* plane
- Reflection axis perpendicular to *xy* plane – rotation path is in the *xy* plane

2D REFLECTION

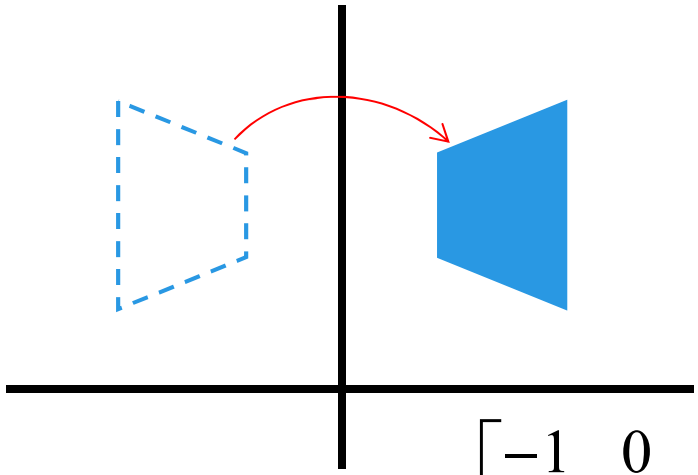
x-axis
Reflection about the line
y=0



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

Transformation keeps x values but flips the y values

y-axis
Reflection about the line
x=0

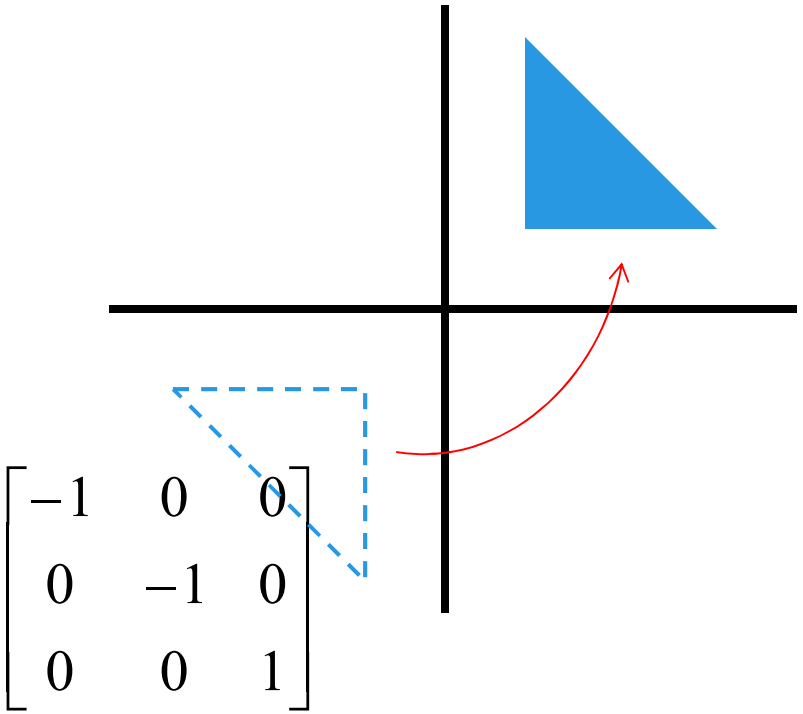


$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Transformation keeps y values but flips the x values

2D REFLECTION

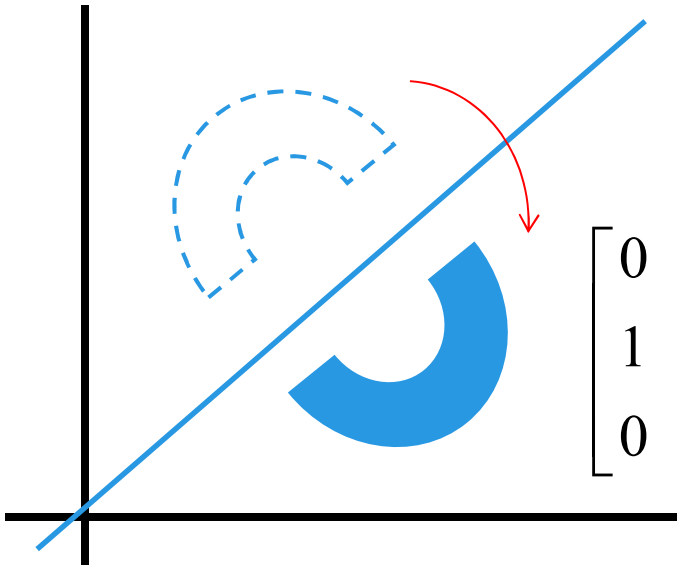
Reflection relative to the coordinate origin



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

Transformation flips both x values and y values by Reflecting relative to the coordinate origin

Reflection axis as the diagonal line x=y



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

2D Reflection

Elements of the reflection matrix can be set to values other than ±1.

Reflection parameter:

- >1 – shifts the mirror image of a point farther from the reflection axis.
- <1 – brings the mirror image of a point closer to the reflection axis.

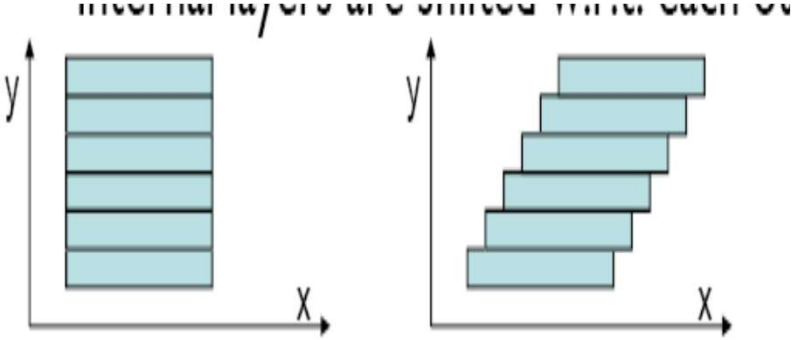
Thus, a reflected object can also be enlarged, reduced or distorted.

2D SHEAR

Transformation that distort the shape of an object.

Slide to another shape

Internal layers are shifted w.r.t. each other



2 common shearing transformation

- Shift coordinate *x* values
- Shift coordinate *y* values

2D Shear

An x-direction shear relative to the x axis is produced with the transformation matrix

$$\begin{pmatrix} 1 & sh_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

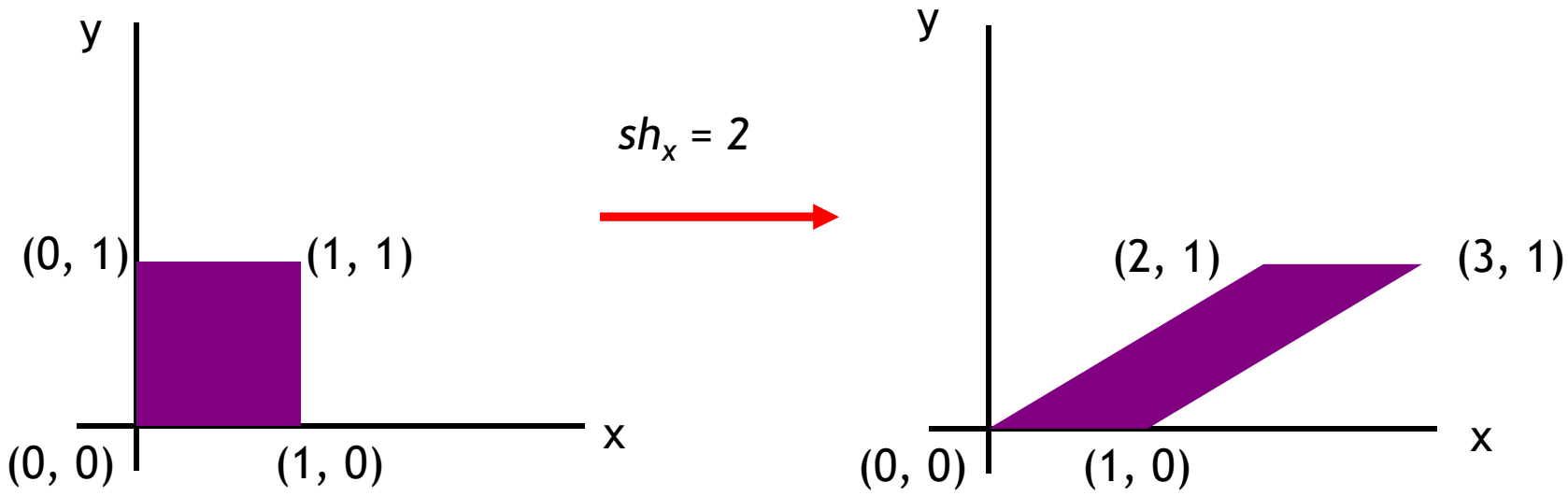
which transforms coordinate positions as

$$x' = x + sh_x \cdot y, \qquad y' = y$$

2D SHEAR

Any real number can be assigned to the shear parameter sh_x.

A coordinate position (x, y) is then shifted horizontally by an amount proportional to its perpendicular distance (y value) from the x axis.



2D Shear

We can generate *x*-direction shears relative to other reference lines with

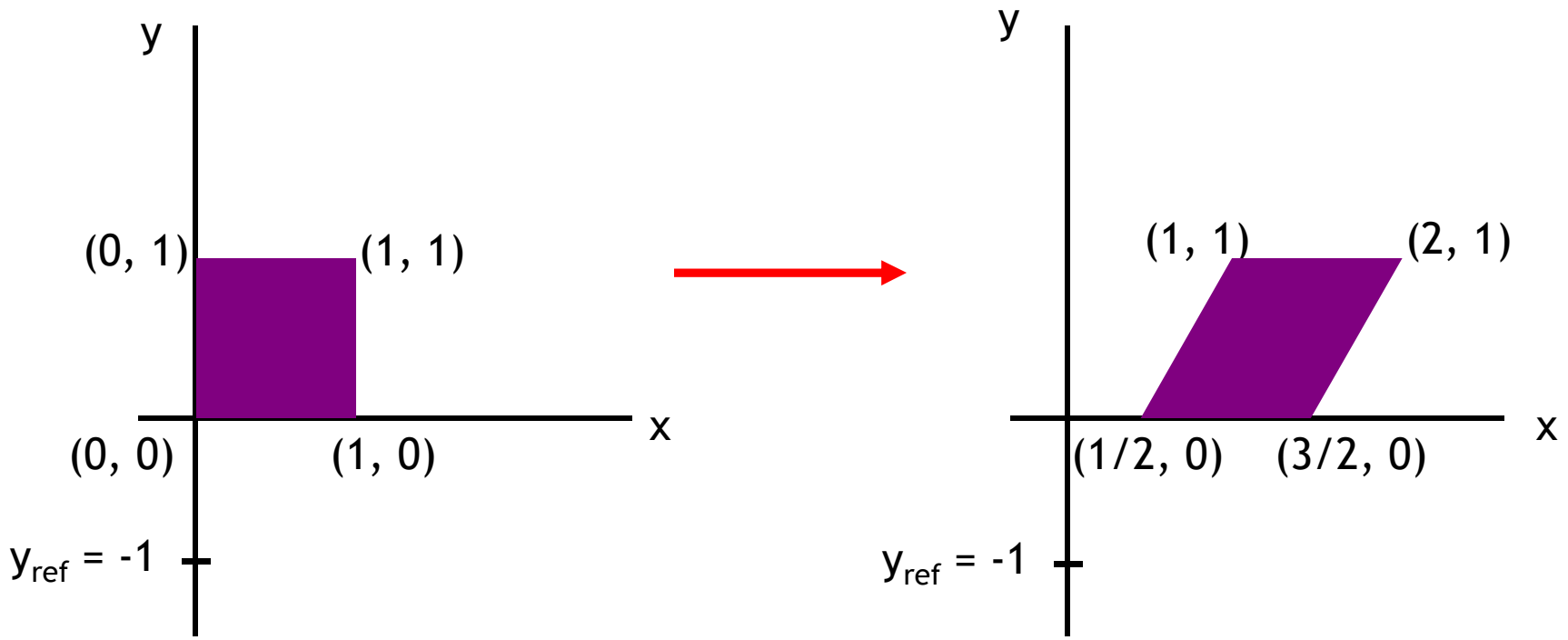
$$\begin{pmatrix} 1 & sh_x & -sh_x \cdot y_{ref} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Now, coordinate positions are transformed as

$$x' = x + sh_x(y - y_{ref}), \qquad y' = y$$

EXAMPLE

sh_x=0.5 and y_{ref} = -1



2D SHEAR

A y-direction shears relative to other reference lines can generate with

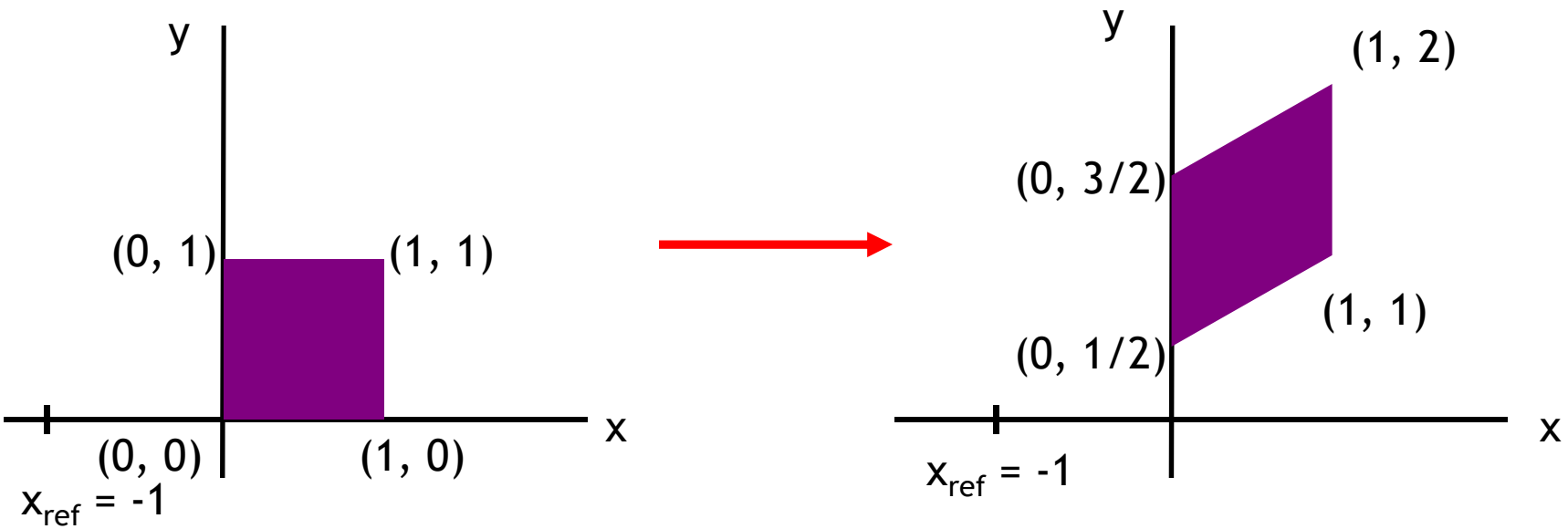
$$\begin{pmatrix} 1 & 0 & 0 \\ sh_y & 1 & -sh_y \cdot x_{ref} \\ 0 & 0 & 1 \end{pmatrix}$$

Now, coordinate positions are transformed as

$$x' = x, \qquad y' = y + sh_y(x - x_{ref})$$

EXAMPLE

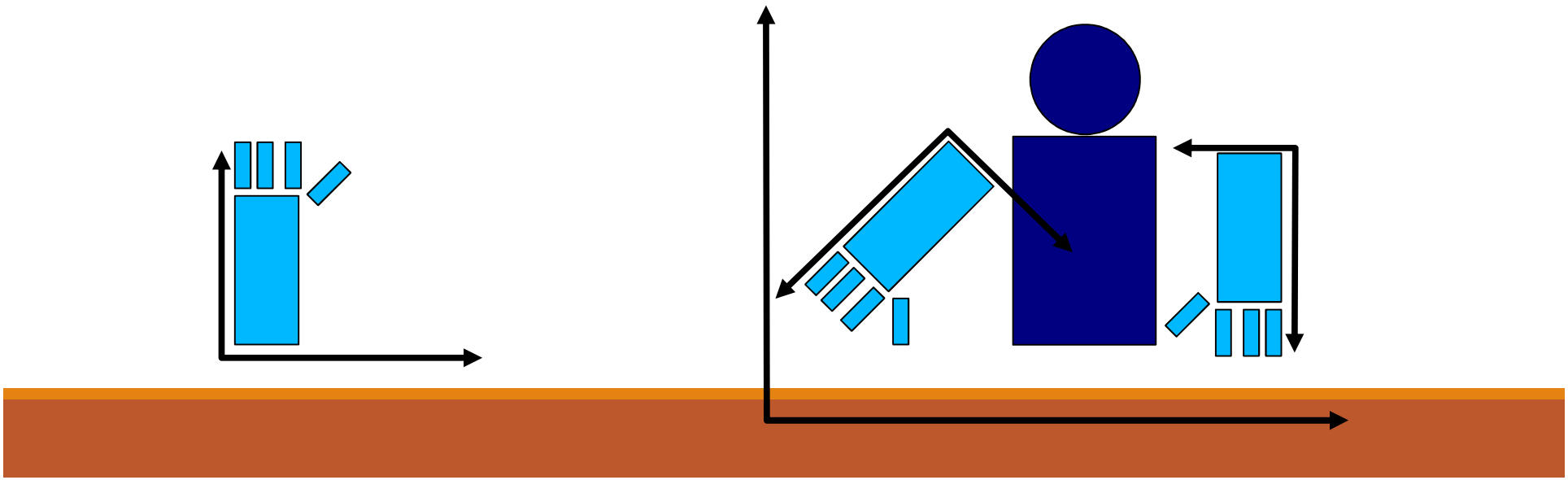
sh_y=0.5 and x_{ref} = -1



Transformations Between the Coordinate Systems

Between different systems: Polar coordinates to cartesian coordinates

Between two cartesian coordinate systems. For example, relative coordinates or window to viewport transformation.



Transformations Between the Coordinate Systems

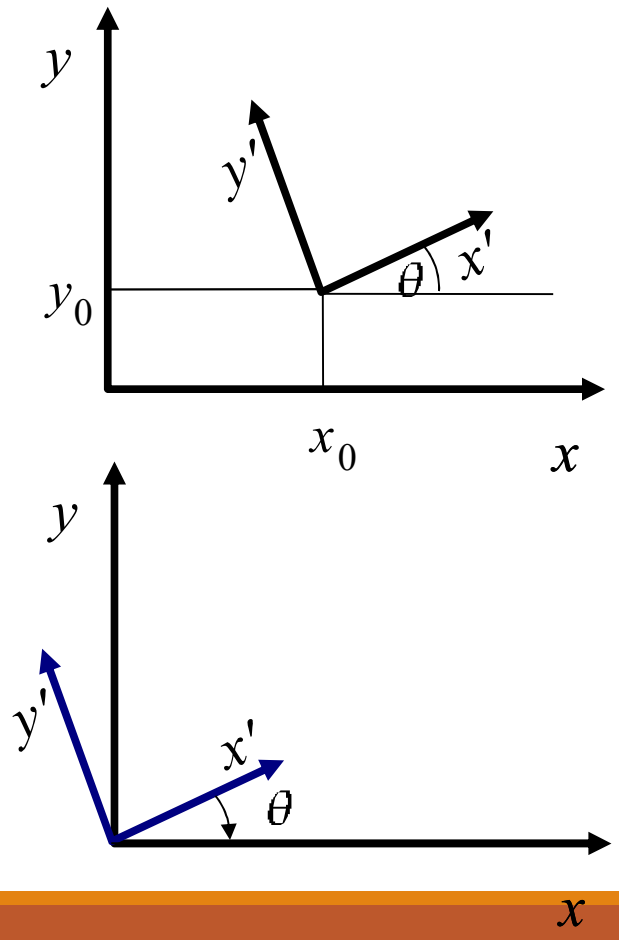
How to transform from *x,y* *to x',y' ?*

Superimpose x',y' to x,y

Transformation:

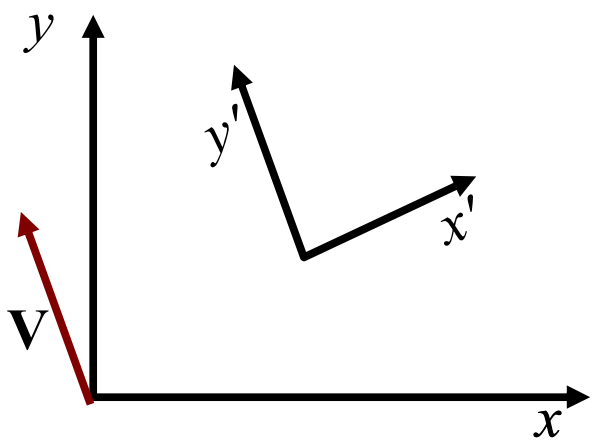
- *Translate so that (x_0,y_0) moves to (0,0) of x,y*
- *Rotate x' axis onto x axis*

$$R(-\theta)\cdot T(-x_{0,}-y_0)$$



Transformations Between the Coordinate Systems

Alternate method for rotation:
Specify a vector **V** for positive y' axis:



unit vector in the y' direction :

$$\mathbf{v} = \frac{\mathbf{V}}{|\mathbf{V}|} = (v_x, v_y)$$

unit vector in the x' direction, rotate **v** clockwise 90°

$$\mathbf{u} = (v_y, -v_x) = (u_x, u_y)$$

Transformations Between the Coordinate Systems

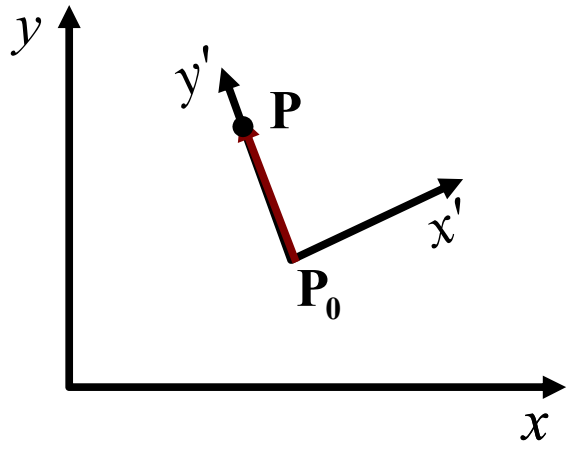
Elements of any rotation matrix can be expressed as elements of a set of orthogonal unit vectors:

$$\mathbf{R} = \begin{bmatrix} u_x & u_y & 0 \\ v_x & v_y & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} v_y & -v_x & 0 \\ v_x & v_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Choose the directions for v relative to position P0 .

The components of v calculated as

$$\mathbf{v} = \frac{\mathbf{P} - \mathbf{P_0}}{|\mathbf{P} - \mathbf{P_0}|}$$



U is obtained as perpendicular to v

Affine Transformations

- An affine transformation is an important class of linear 2-D geometric transformations which maps variables (*e.g.* pixel **intensity values** located at position (x,y) in an input image) into new variables (*e.g.* in an output image (x',y') by applying a linear combination **of translation, rotation, scaling** and/or shearing (*i.e.* non-uniform scaling in some directions) operations.

- Coordinate transformations of the form:

$$x' = a_{xx}x + a_{xy}y + b_x$$

$$y' = a_{yx}x + a_{yy}y + b_y$$

Translation, rotation, scaling, reflection, shear. Any affine transformation can be expressed as the combination of these.

Summary

Homogeneous coordinates

Reflection and shearing – w.r.t origin and fixed point

Transformation between systems

Affine transformations

Two Dimensional Viewing

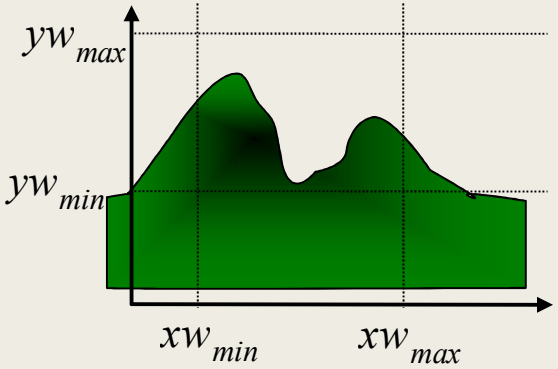
The Viewing Pipeline

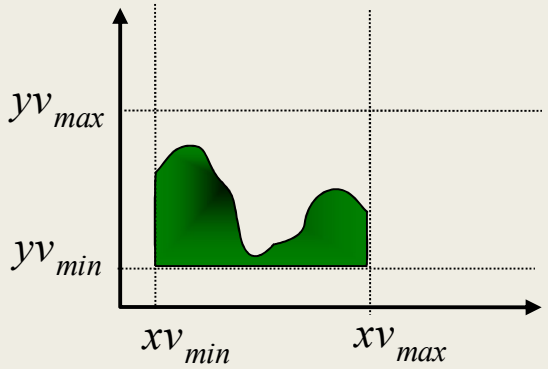
- Graphics package allows the user to specify which part of a defined picture to be displayed where that part is to displayed on the display device
- **Window**
  - *A world-coordinate area selected for display.*

    defines what is to be viewed
- **Viewport**
  - *An area on a display device to which a window is mapped.*

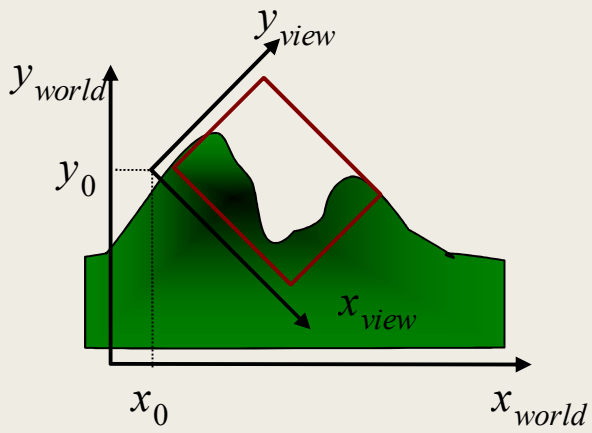
    defines where it is to be displayed
- **Viewing transformation**
  - *The mapping of a part of a world-coordinate scene to device coordinates.*
  - *The two dimensional viewing transformation is referred as windowing transformations.*

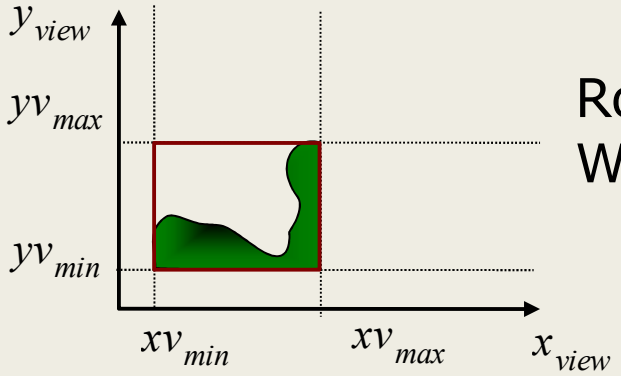
Two-Dimensional Viewing





Rectangular Window





Rotated Window

Two-Dimensional Viewing

- Rectangular window of different orientation viewing transformations follows the following steps
- Construct the scene in world coordinates using the output primitives and attributes.
- Obtain the particular orientation for the window, set up a two dimensional viewing coordinate system in the world coordinate plane and define a window in the viewing coordinate system.
- Once the reference frame is established transform the descriptions in world coordinates to viewing coordinates.
- We then define a viewport in normalized coordinates and map the viewing coordinate of the scene to normalized coordinates

1. Construct the scene in world coordinates with primitives and attributes.
2. Set up a 2D viewing coordinate system, defining a window in it.
3. Transform descriptions from world to viewing coordinates.
4. Define a viewport in normalized coordinates and map the scene to it.
5. Map the scene from normalized coordinates to various display devices.
6. Perform window-to-viewport transformations for each output device.
7. This mapping process is called workstation transformation.
8. The window is in normalized space, and the viewport is in display device coordinates.
9. This process controls how different parts of the scene appear on individual output devices.

The Viewing Pipeline

- All parts of the picture that lie outside the viewport are clipped .
- Contents of the viewport are transferred to device coordinates.

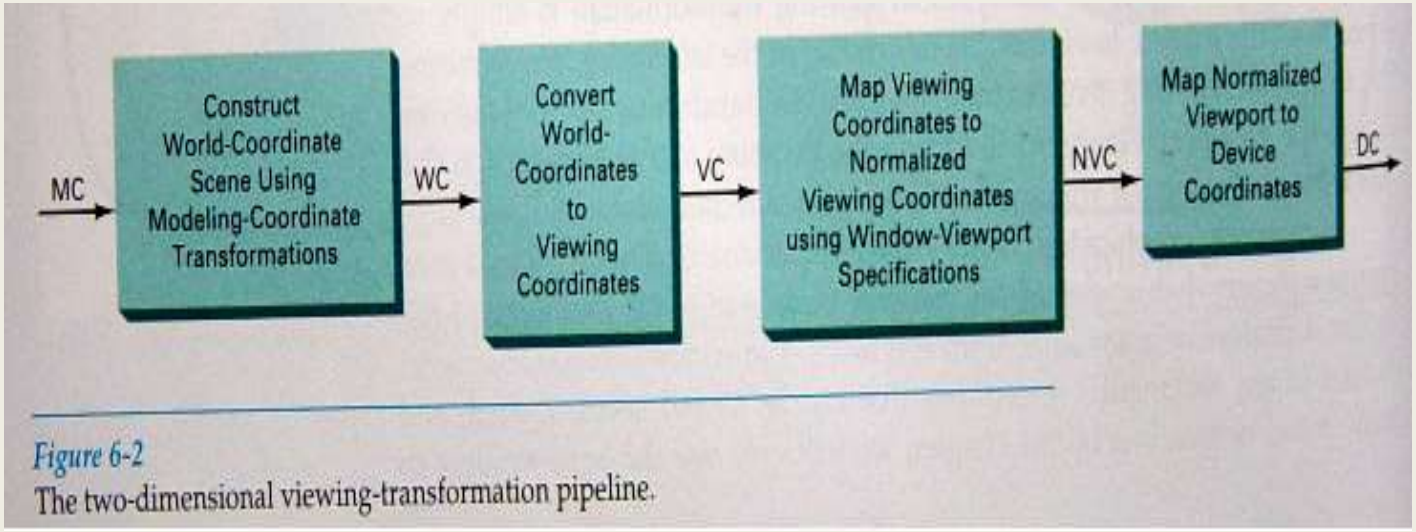


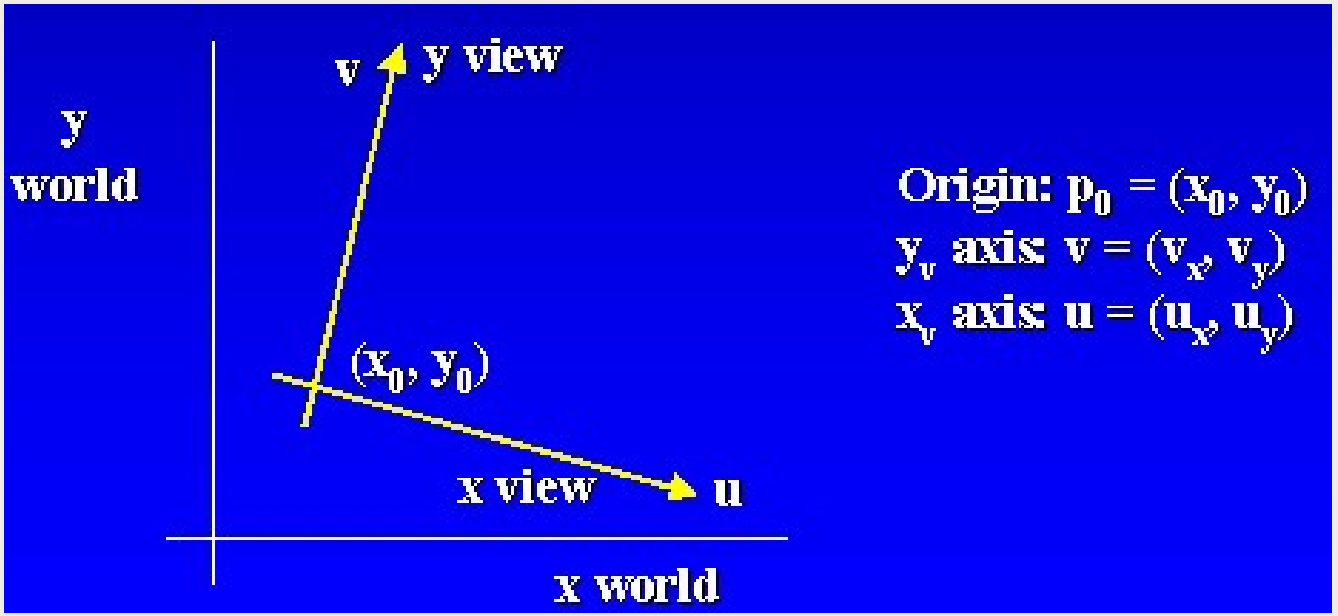
Figure 6-2
The two-dimensional viewing-transformation pipeline.

Two-Dimensional Viewing

- By changing the position of the viewport we can view objects at different positions on the display area of the output device.
- **Zooming effects**
  - *Successively mapping different-sized windows on a fixed-sized viewports.*
- **Panning effects**
  - *Moving a fixed-sized window across the various objects in a scene.*
- **Device independent**
  - *Viewports are typically defined within the unit square. (normalized coordinates)*
  - *This provides means for separating the viewing and other transformations from specific output device requirements.*

Viewing Coordinate Reference Frame

- The reference frame for specifying the world-coordinate window.
  - *Viewing-coordinate origin: P0 = (x0, y0)*
  - *Establish the orientation or rotation of this reference frame.*
  - *Specify View up vector V: Define the viewing y_v direction*

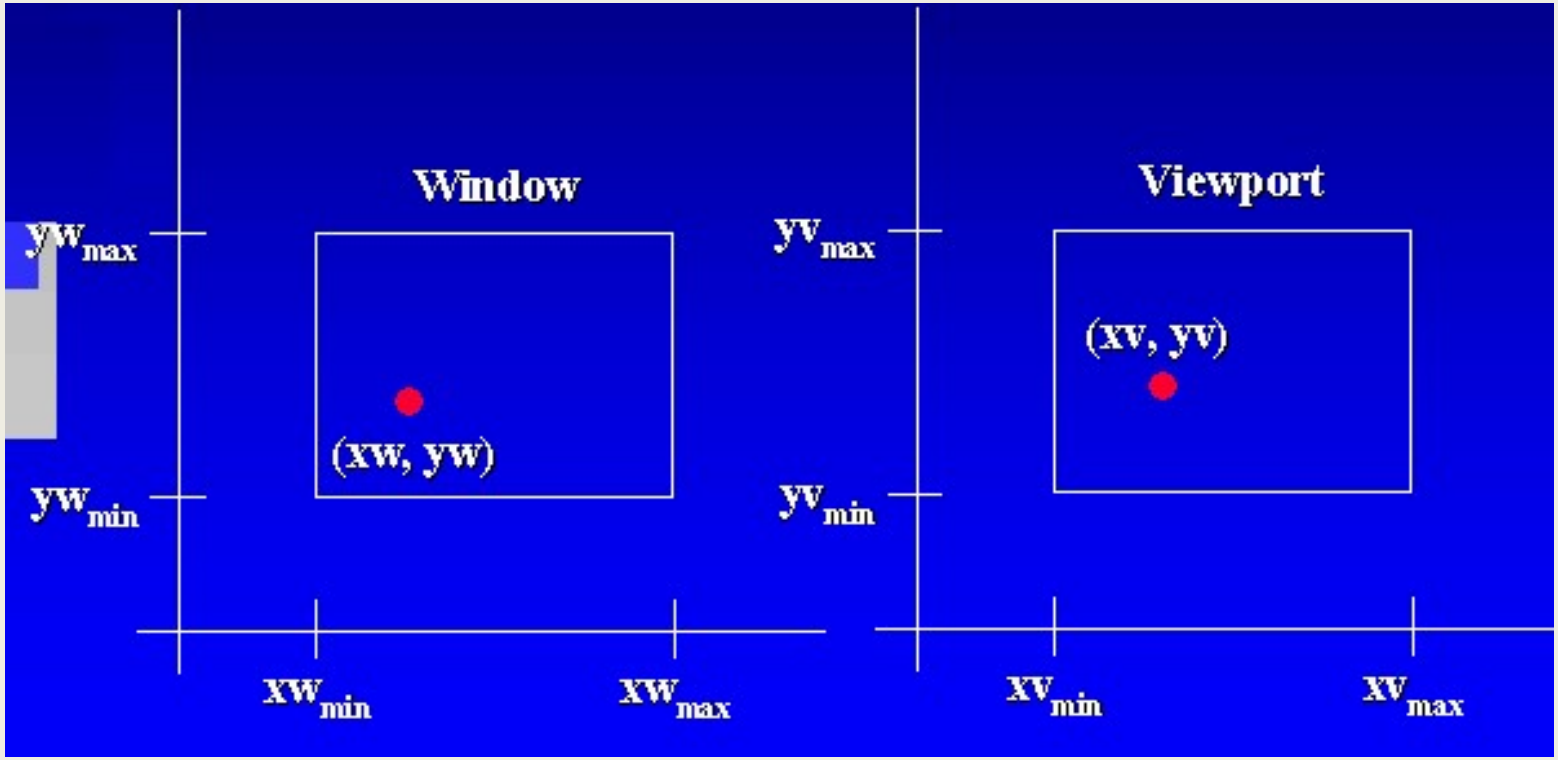


Viewing Coordinate Reference Frame

- Given V calculate the components of uint vectors v_x,v_y and u_x,u_y

  for the viewing y_v and X_v .
- These unit vectors aligns the viewing x_v,y_v axes with world axes x_w,y_w
- The composite two dimensional transformation to convert world coordinates to viewing coordinates is
- Translate the viewing origin to the world origin
- Rotate to align the two coordinate reference frames.
  - $M_{wc,vc}=R.T$

Window-to-Viewport Coordinate Transformation



Window-to-Viewport Coordinate Transformation

$$\frac{xv - xv_{\min}}{xv_{\max} - xv_{\min}} = \frac{xw - xw_{\min}}{xw_{\max} - xw_{\min}} \qquad \frac{yv - yv_{\min}}{yv_{\max} - yv_{\min}} = \frac{yw - yw_{\min}}{yw_{\max} - yw_{\min}}$$

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

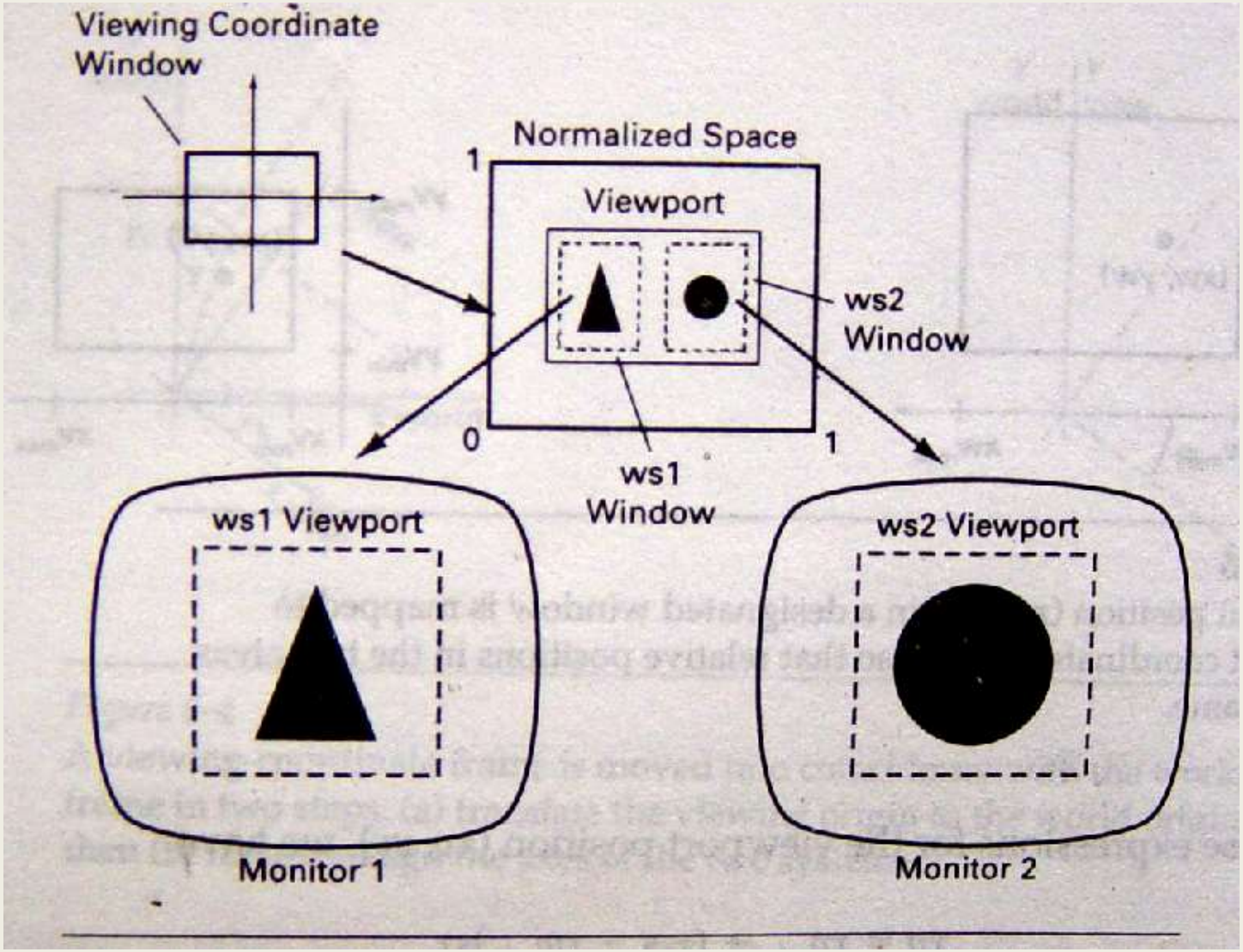
$$yv = yv_{\min} + (yw - yw_{\min})sy$$

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

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

A point at poisition (x_w,y_w) is mapped into position (x_v,y_v)

Workstation transformtion



Workstation Transformtion

- From normalized coordinates , object description are mapped to various display devices.
- Any no of output devices can be used and window to viewport transformation can be performed for each open output device.
- This mapping called the **workstation transformation.**
- Window area in normalized space and viewport area in the coordinates of the display device.
- Controls the positioning of parts of a scene on individual output devices.

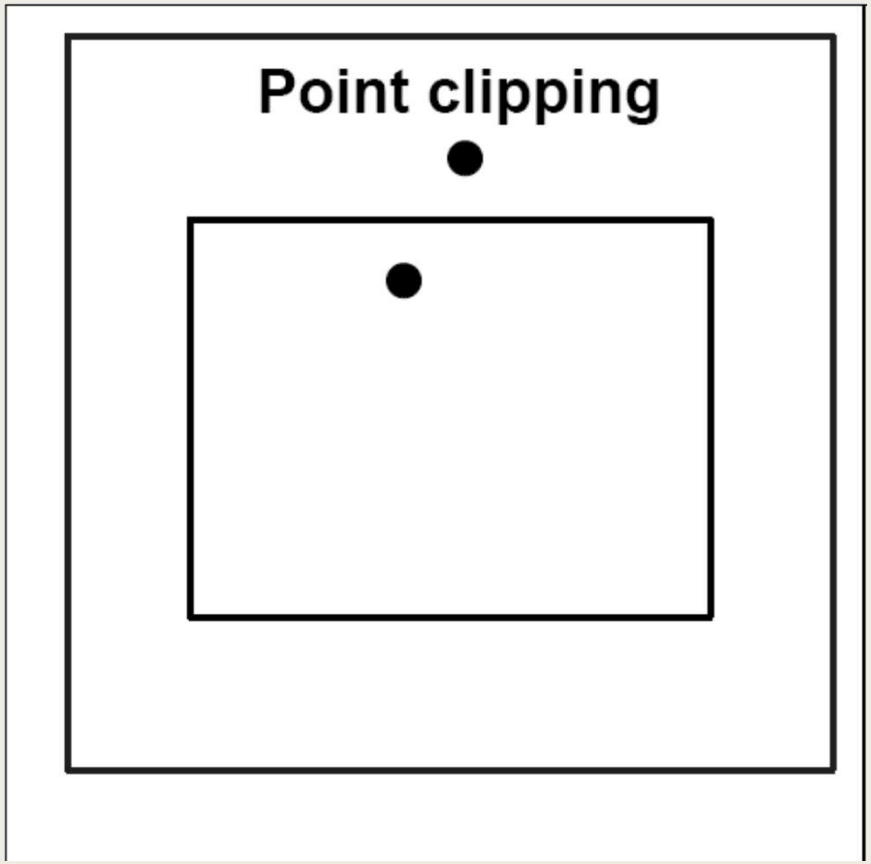
Clipping Operations

- Clipping
  - *Identify those portions of a picture that are either inside or outside of a specified region of space.*
- Clip window
  - *The region against which an object is to be clipped.*
  - *The shape of clip window*
- Applications of clipping :extracting part of the defined scene for viewing , identifying visible surfaces in 3d views etc.
- World-coordinate clipping: Clipping algorithm can be applied to the world coordinate , so the contents of the window are mapped to device coordinates.

Clipping Operations

- Viewport clipping
  - *It can reduce calculations by allowing concatenation of viewing and geometric transformation matrices.*
- Types of clipping
  - *Point clipping*
  - *Line clipping*
  - *Area (Polygon) clipping*
  - *Curve clipping*
  - *Text clipping*
- Point clipping (Rectangular clip window)

Point clipping



Point clipping

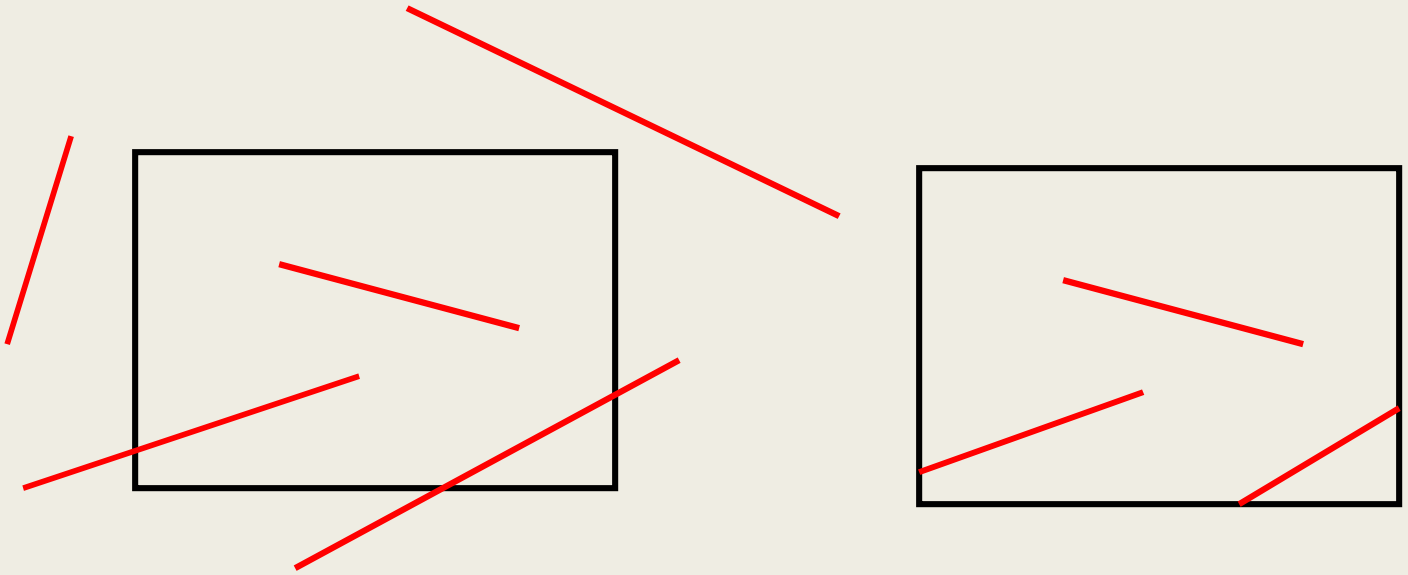
- Is point (x,y) inside the clip window?
- Considering the clip window is a rectangular window.
- A pint is inside if it satisfies the following inequalities are satisfied
  - $XW_{win} <= x <= XW_{max}$
  - $YW_{win} <= x <= YW_{max}$

where the edges of the clip window can be either world coordinate window boundaries or viewport boundaries

- If any of the inequalities are not satisfied the point is clipped.

Line Clipping

- **Possible relationships between line positions and a standard rectangular clipping region**



Before clipping

after clipping

Line Clipping

- Possible relationships
  - *Completely inside the clipping window*
  - *Completely outside the window*
  - *Partially inside the window*
- Parametric representation of a line segment with endpts (x1,y1) and (x2,y2)

  $x = x_1 + u(x_2 - x_1)$

  $y = y_1 + u(y_2 - y_1)$ $0 <= u <= 1$
- The value of u for an intersection with a rectangle boundary edge
  - *Outside the range 0 to 1 ,line does not enter the interior of window*
  - *Within the range from 0 to 1,the line segment crosses the clipping area.*

2D Viewing

Part II

Clipping Algorithms

Types of Clipping

Point Clipping

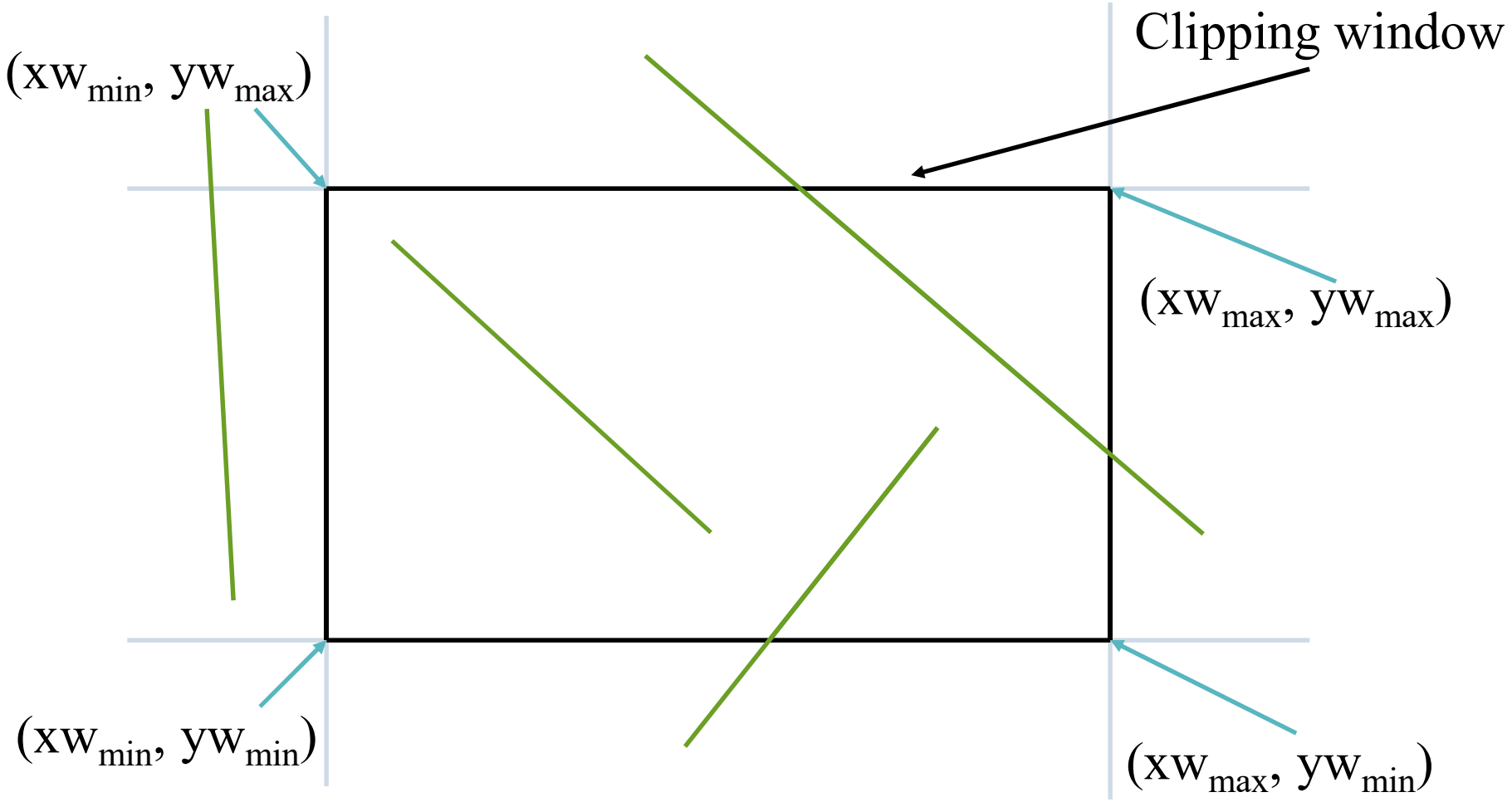
Line Clipping

Polygon Clipping

Curve Clipping

Text Clipping

Line Clipping



Line Clipping Algorithms

Simple Line Clipping

Cohen-Sutherland Algorithm

Liang-Barsky Algorithm

Simple Line Clipping

Using **point-clipping test** to determine completely INSIDE or OUTSIDE line.

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

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

Using **parametric equations** to determine partially INSIDE or OUTSIDE line.

$$x = x_0 + u(x_{end} - x_0)$$

$$y = y_0 + u(y_{end} - y_0) \quad 0 \le u \le 1$$

Cohen-Sutherland Line Clipping Algorithm

Oldest and most popular line-clipping algorithms.

Method speeds up the processing of line segments by performing initial tests

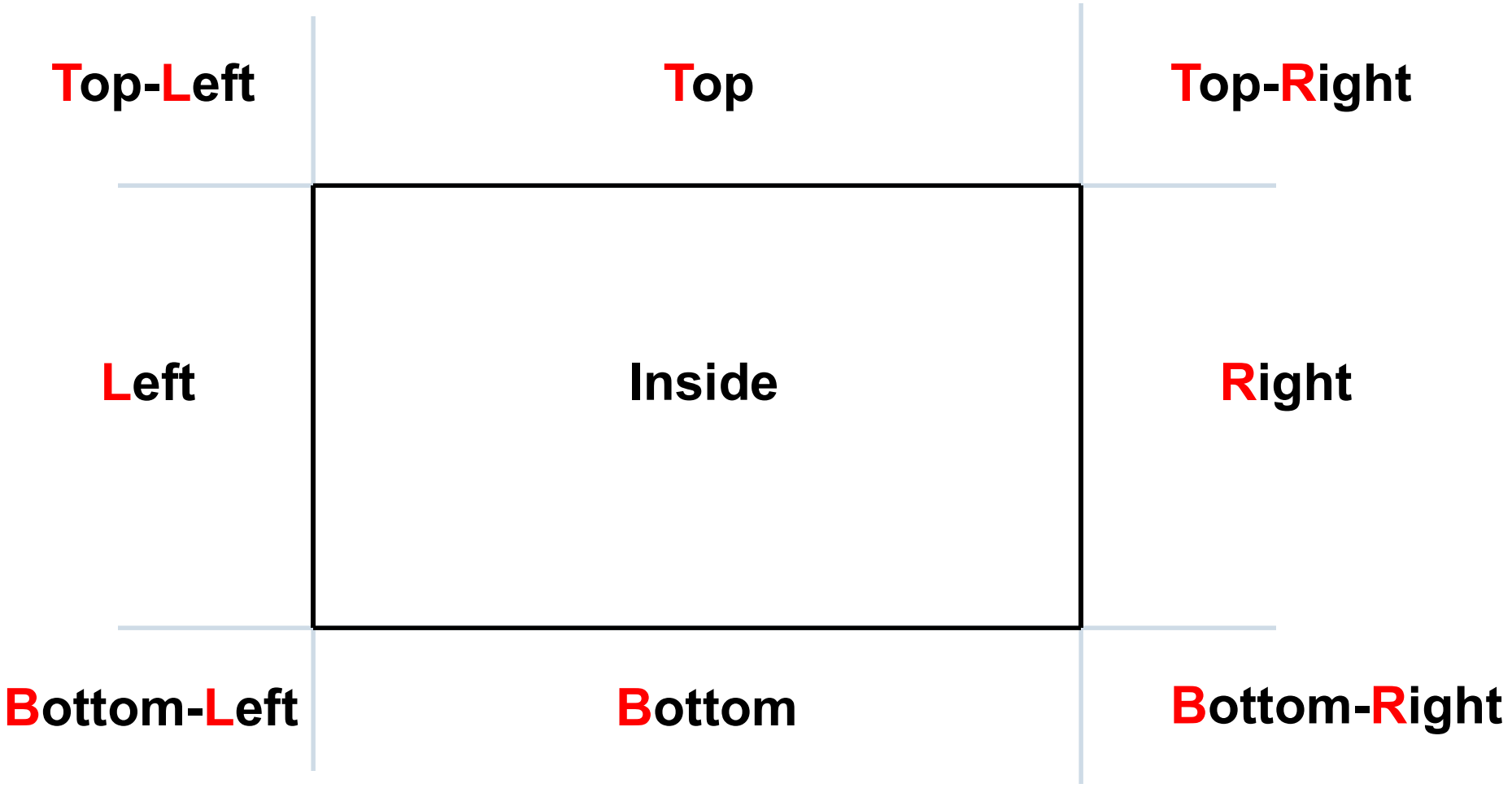
Reduces the no of intersections that must be calculated.

Each line endpoint in picture is assigned a four digit binary code called **region code** .

Region code identifies the location of the point relative to the boundaries of the clip window.

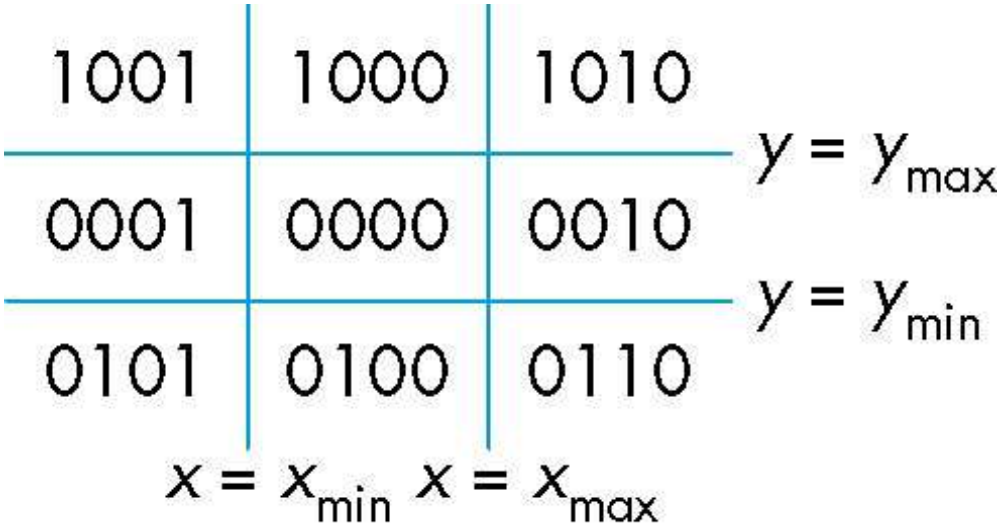
TBRL

Cohen-Sutherland Algorithm



Region Codes

Bit	T	B	R	L
	4	3	2	1



Each region is represented by a 4-bit **region code** TBRL:

$$T = \begin{cases} 1 \text{ if } y > y_{\max} \\ 0 \text{ otherwise} \end{cases} \quad B = \begin{cases} 1 \text{ if } y < y_{\min} \\ 0 \text{ otherwise} \end{cases} \quad R = \begin{cases} 1 \text{ if } x > x_{\max} \\ 0 \text{ otherwise} \end{cases} \quad L = \begin{cases} 1 \text{ if } x < x_{\min} \\ 0 \text{ otherwise} \end{cases}$$

Trivial accept cases

The trivial accept case corresponds to ensuring that no region code bits are set for both endpoints. This can be nicely accomplished with a bitwise OR operation.

RC (A) 0101

RC (B) 0100

bitwise OR 0101 → fails trivial accept

RC(C) 0000

RC (D) 0000 → trivial accept

Trivial reject cases

The trivial reject case can be nicely accomplished with a bitwise AND operation. We can trivially reject a line segment which has a result not equal to 0000

RC (A) 0101

RC (B) 0100

bitwise AND 0100 → trivial reject

RC (E) 1000

RC (F) 0010

bitwise AND 0000 →fails trivial reject

Cohen-Sutherland Algorithm (cont.)

Lines that cannot be identified as completely inside or outside by above tests can be test as follows

- Choose an endpoint of the line that is outside the window.
- Find the intersection point at the window boundary (base on region code).
- Replace endpoint with the intersection point and update the region code.
- Repeat steps until we find a clipped line either trivially accepted or trivially rejected.

Repeat step for other lines.

How to check for intersection?

if bit 4 = 1 → there is intersection on TOP boundary.
if bit 3 = 1 → .. .. BOTTOM ..
if bit 2 = 1 → .. .. RIGHT ..
if bit 1 = 1 → .. .. LEFT ..

How to find intersection point?

- using slope intercept of the line equation

m=(y2-y1)/(x2 –x1)

intersection with LEFT or RIGHT boundary.

$x = xw_{min}$ (LEFT) $x = xw_{max}$ (RIGHT)

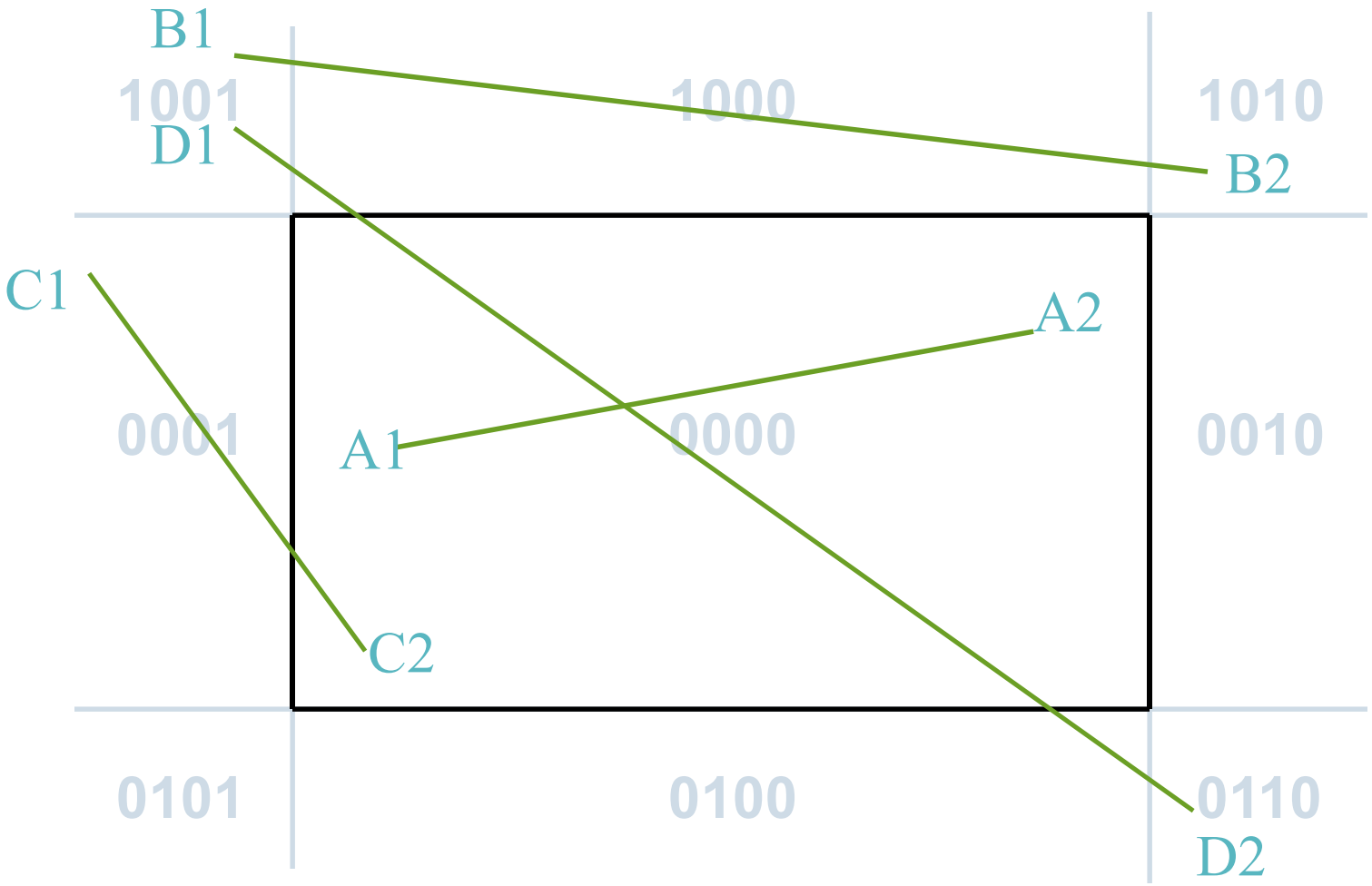
y = y1 + m(x –x1)

intersection with BOTTOM or TOP boundary.

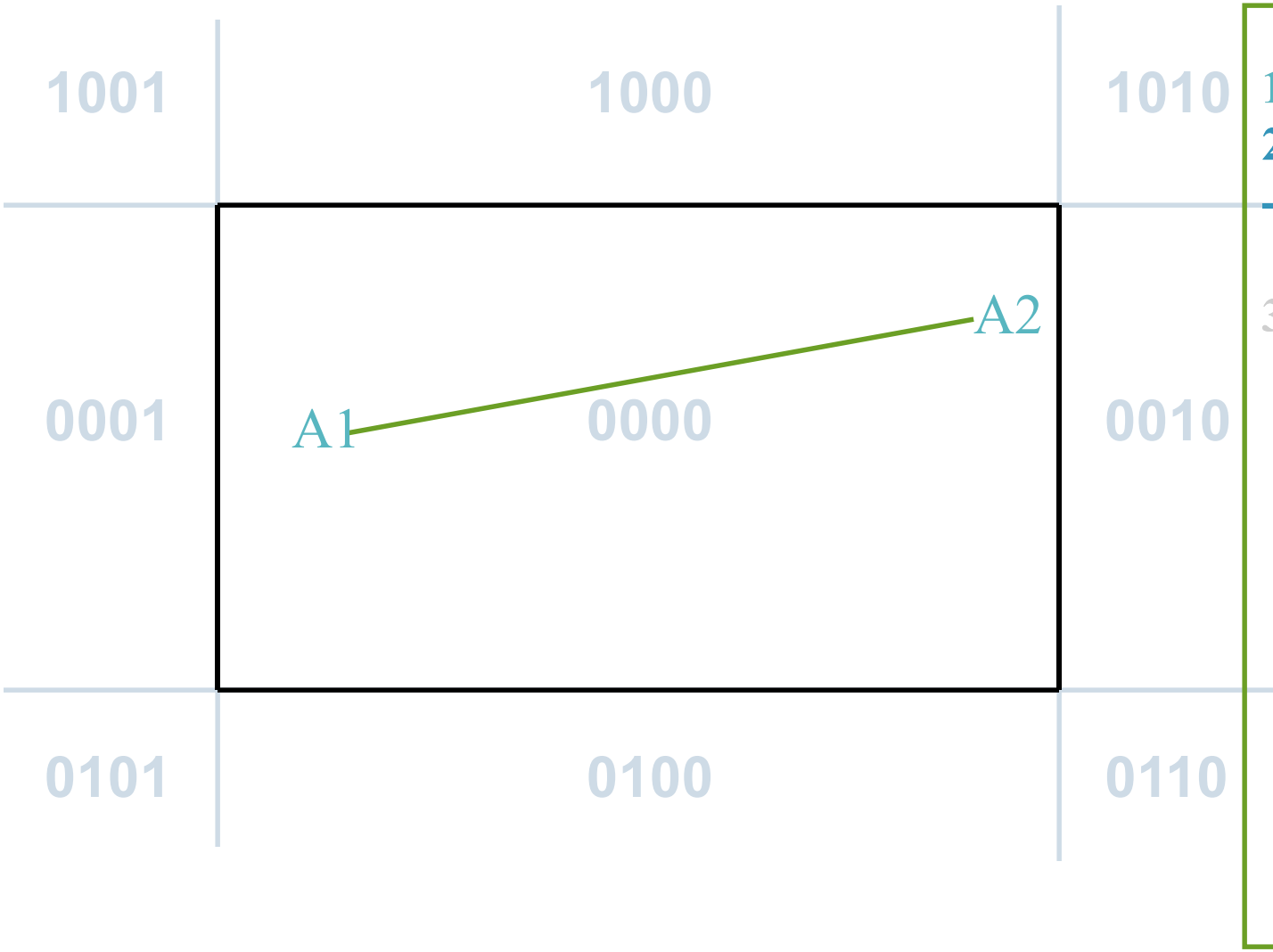
$y = yw_{min}$ (BOTTOM) $y = yw_{max}$ (TOP)

x = x1 + (y –y1)/m

Trivial accept & reject



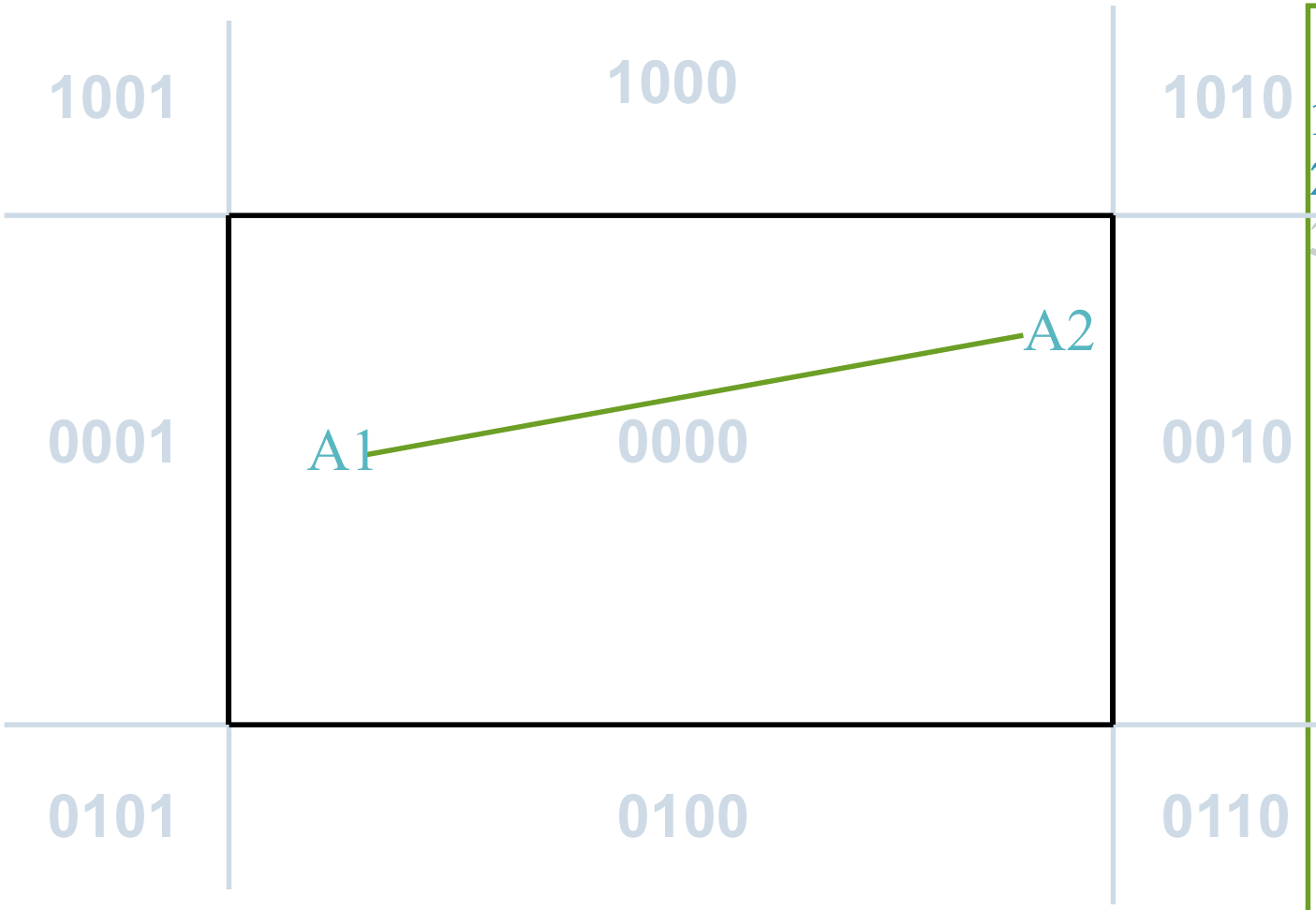
Example



algorithm

1. A1=0000,A2=0000
2. (both 0000) – Yes -> accept & draw
3.
   - 3.1
   - 3.2
     - 3.2.1
     - 3.2.2
     - 3.2.3
     - 3.2.4

Example



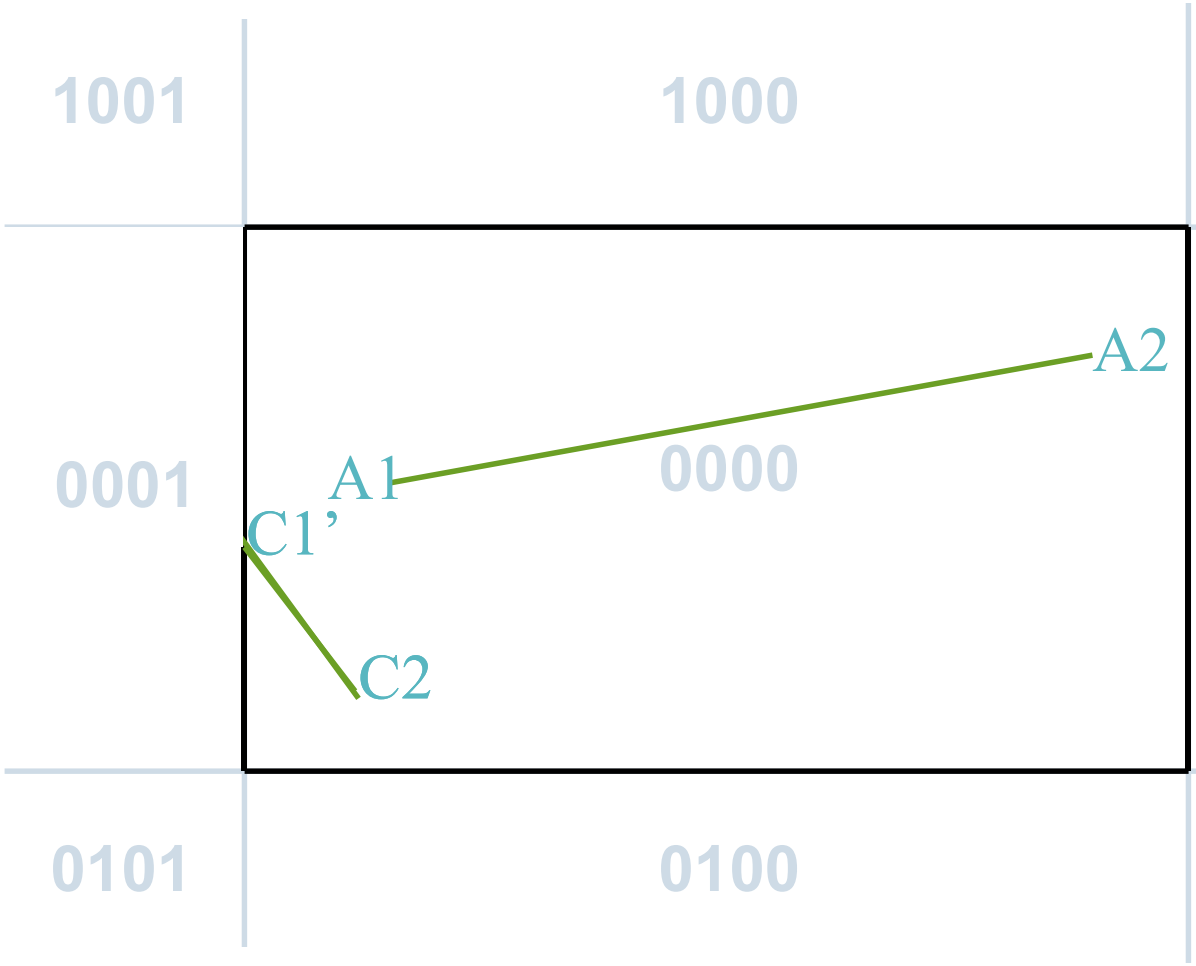
algorithm

1. B1=1001,B2=1010
2. (both 0000) – No
3. AND Operation
   - B1 → 1001
   - B2 → 1010
   - Result 1000

3.1 (not 0000) – Yes
→ reject

3.2
- 3.2.1
- 3.2.2
- 3.2.3
- 3.2.4

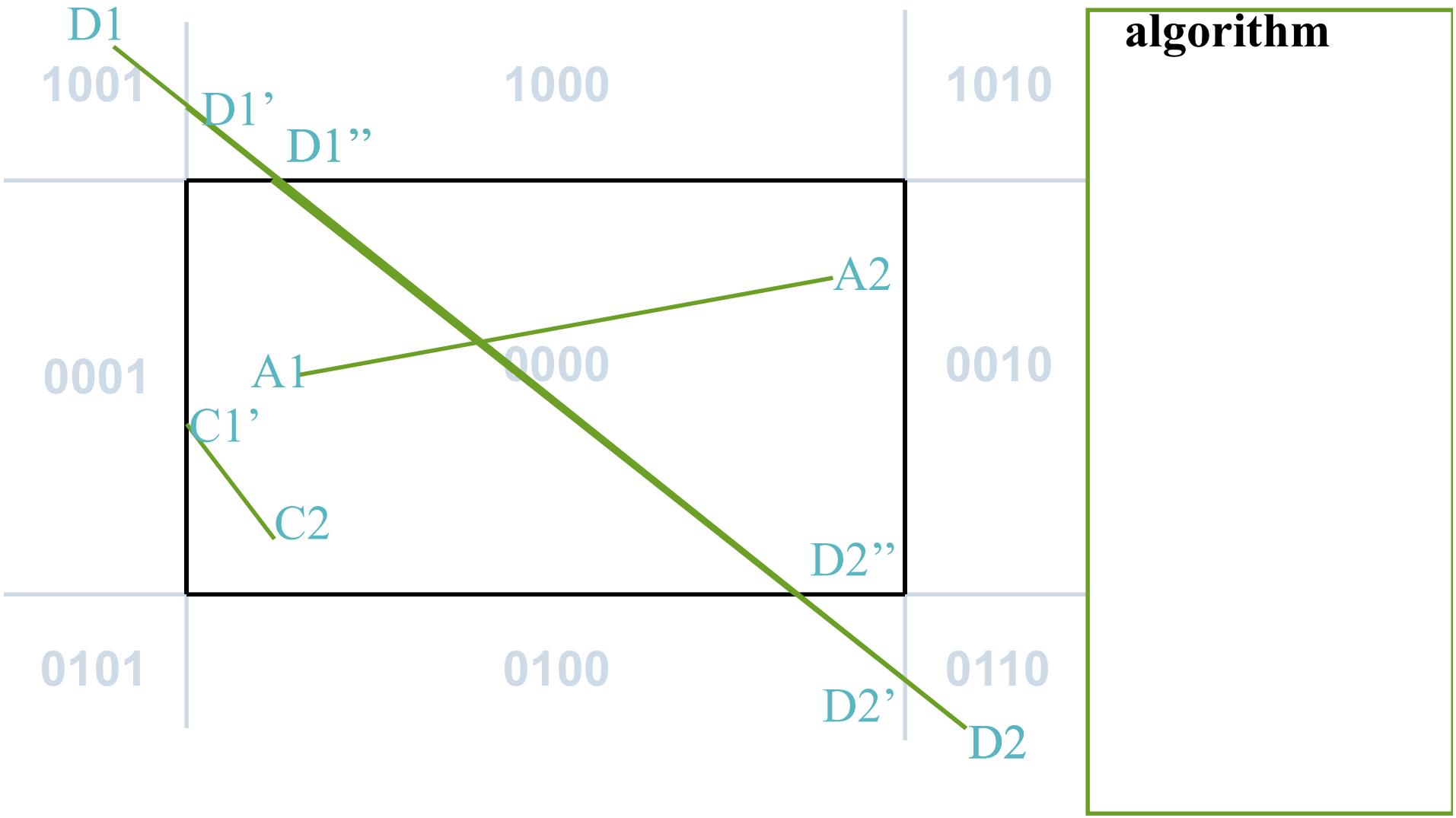
Example



algorithm

1. C1’=0000,C2=0000
2. (both 0000) – Yes -> accept &
   draw
3.
   3.1
   3.2
      3.2.1
      3.2.2
      3.2.3
      3.2.4

Example


D1
1001
D1’
D1”
1000
1010
A2
A1
0000
0001
0010
C1’
C2
D2”
0101
0100
0110
D2’
D2
algorithm

Advantages & Disadvantages of Cohen-Sutherland Algorithm

Easy to implement

Early accept/reject tests

Slow for many clipped lines

Exercise

$(xw_{min}, xw_{max}) = (10, 150)$

$(yw_{min}, yw_{max}) = (10, 100)$

$P_0 = (0, 120)$

$P_1 = (130, 5)$

Liang-Barsky Line Clipping

This algorithm uses the parametric equations for a line

Solves four inequalities to find the range of the parameter for which the line is in the viewport (window).

Consider the parametric definition of a line:

- $x = x_1 + u\Delta x$
- $y = y_1 + u\Delta y$
- $\Delta x = (x_2 - x_1)$, $\Delta y = (y_2 - y_1)$, $0 \leq u \leq 1$

Liang-Barsky Line Clipping

Mathematically, this can be written using point clipping conditions in the parametric form

- $x_{min} \leq x_1 + u\Delta x \leq x_{max}$
- $y_{min} \leq y_1 + u\Delta y \leq y_{max}$

Rearranging, we get

- $-u\Delta x \leq (x_1 - x_{min})$
- $u\Delta x \leq (x_{max} - x_1)$
- $-u\Delta y \leq (y_1 - y_{min})$
- $u\Delta y \leq (y_{max} - y_1)$

Liang-Barsky Line Clipping

In general: u * pk ≤ qk

Where the parameters p and q are defined as

$$\overline{P}_1 = -\Delta x, \quad \overline{q}_1 = x_1 - x_{\min}$$
$$\overline{P}_2 = \Delta x, \quad \overline{q}_2 = x_{\max} - x_1$$
$$\overline{P}_3 = -\Delta y, \quad \overline{q}_3 = y_1 - y_{\min}$$
$$\overline{P}_4 = \Delta y, \quad \overline{q}_4 = y_{\max} - y_1$$

Liang-Barsky Line Clipping

Cases: pk = 0

- Line is parallel to boundaries

If for the same k, qk < 0,

- The line is completely outside , so reject it

Else,

The line is inside the parallel clipping boundary accept the line

For a non zero value of **pk,**

- Calculate the value of u that corresponds to the point where the infinitely extended line intersects with the extension of boundary k as

$$u = qk / pk$$

Calculate the parameters u1 and u2 the part of the line within clip rectangle

Liang-Barsky Line Clipping

Initialize u1=0,u2=1

Case **pk < 0**

Line proceeds from outside to inside of the infinite extension

- rk = qk / pk
- u1 = max(rk, u1)

Case **p_k > 0**

Line proceeds from inside to outside boundary

rk = qk / pk

u_2 = min(r_k, u_2)

If $u_1 > u_2$, the line is completely outside and can rejected

Else

The endpoints of the clipped line is calculated from the two values of parameters u

If u1 < u2 then draw a line from:

(x0 + Δx · u1, y0 + Δy · u1) to

(x0 + Δx · u2, y0 + Δy · u2)

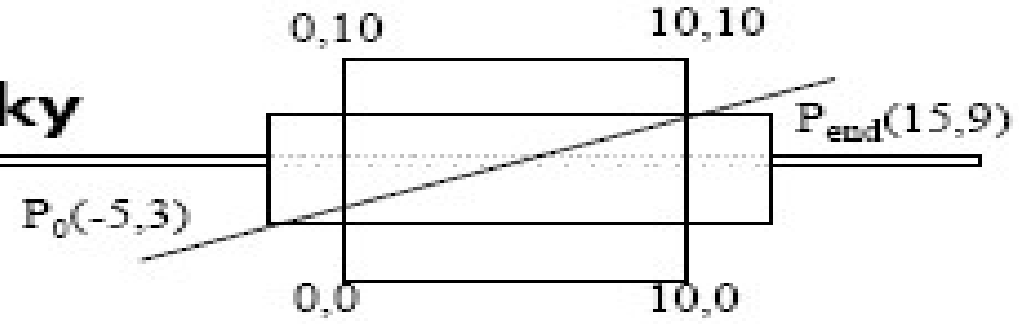
Liang-Barsky Line Clipping

In most cases, Liang-Barsky is slightly more efficient

- Intersection calculations are reduced
- Avoids multiple shortenings of line segments
- Window intersection of the line is computed only once

Liang-Barsky Line Clipping

Example Liang-Barsky



$$u_{left} = \frac{q_1}{p_1} = \frac{x_0 - xw_{min}}{-\Delta x} = \frac{-5-0}{-(15-(-5))} = \frac{1}{4}$$

Entering ⇒ $u_{min} = 1/4$

$$u_{right} = \frac{q_2}{p_2} = \frac{xw_{max} - x_0}{\Delta x} = \frac{10-(-5)}{15-(-5)} = \frac{3}{4}$$

Exiting ⇒ $u_{max} = 3/4$

$$u_{bottom} = \frac{q_3}{p_3} = \frac{y_0 - yw_{min}}{-\Delta y} = \frac{3-0}{-(9-3)} = -\frac{1}{2}$$

u < 0 then ignore

$$u_{top} = \frac{q_4}{p_4} = \frac{yw_{max} - y_0}{\Delta y} = \frac{10-3}{9-3} = \frac{7}{6}$$

u > 1 then ignore

Fredrik Johansson
Department of Computing Science, Umeå University

Liang-Barsky Line Clipping

Liang-Barsky Line-Clipping

- We have $u_{min} = 1/4$ and $u_{max} = 3/4$

$$P_{end} - P_0 = (15+5, 9-3) = (20, 6)$$
$$\Delta x \quad \Delta y$$

- If $u_{min} < u_{max}$, there is a line segment
  - compute endpoints by substituting u values
- Draw a line from

  $(-5+(20)\cdot(1/4),\ 3+(6)\cdot(1/4))$

  to

  $(-5+(20)\cdot(3/4),\ 3+(6)\cdot(3/4))$

Peter Johansson
Department of Computing Science, Umeå University

Nicholl-Lee-Nicholl Line Clipping

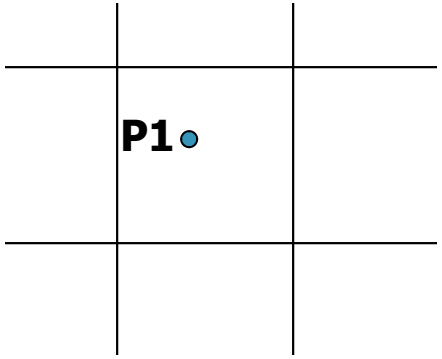
Create more regions around the clip window.

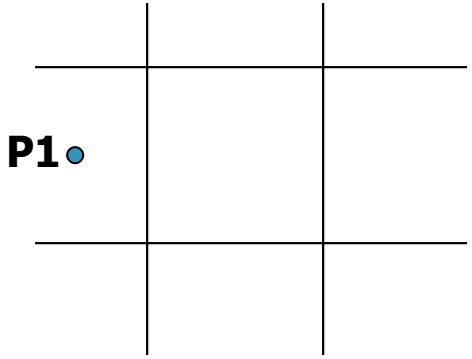
Avoids multiple clipping of an line segment

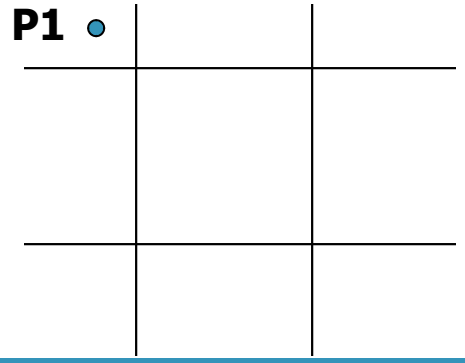
Performs fewer comparisons and divisions of the three.

Disadv: applied only to 2 D Clipping.

NLN

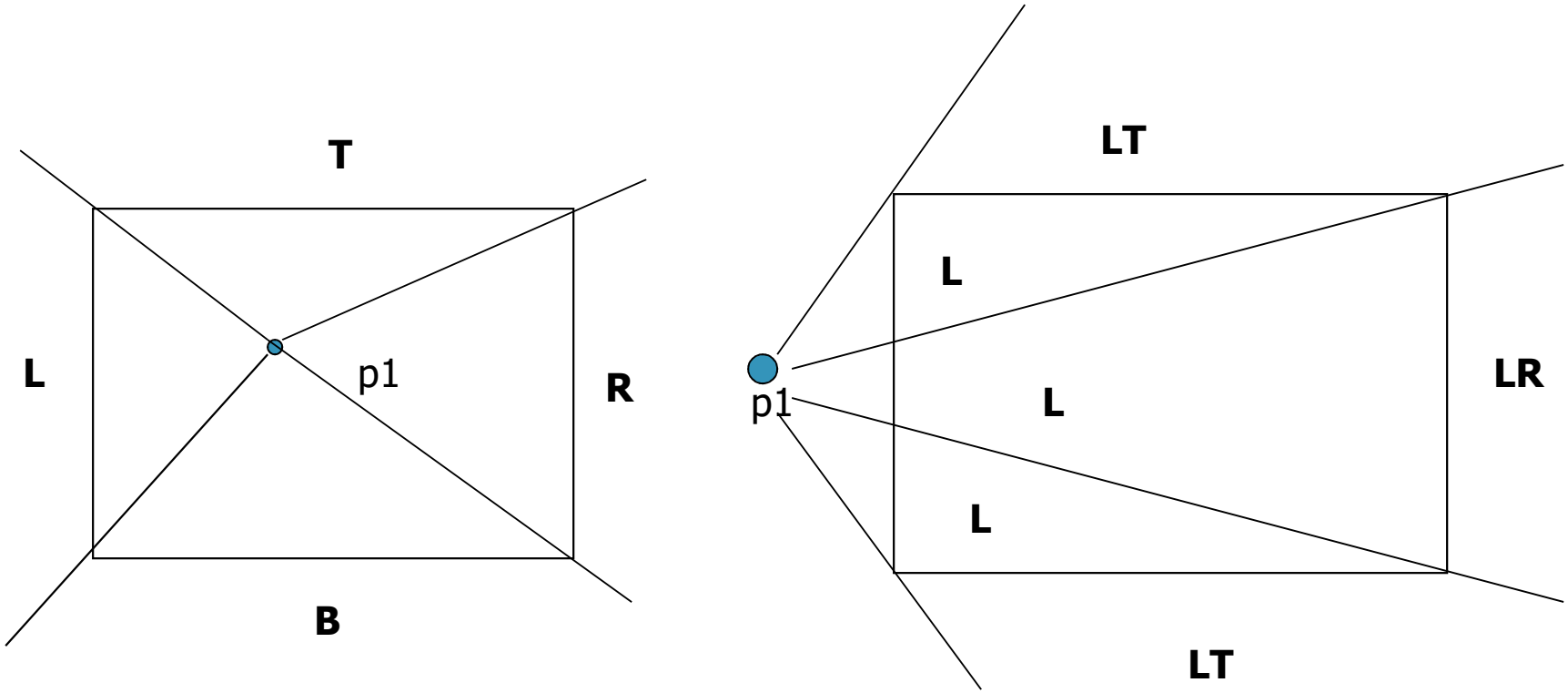




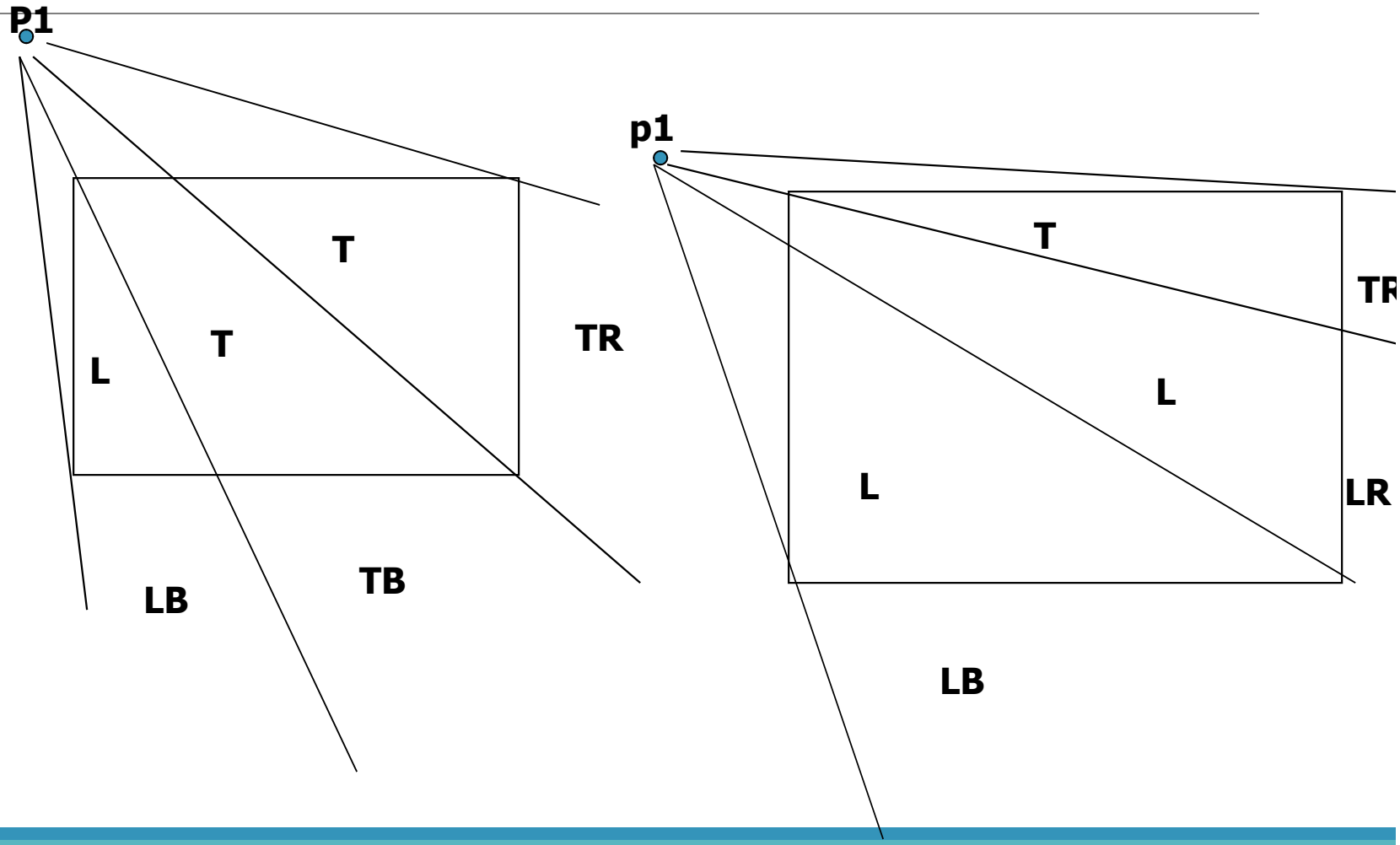


NLN

Find p2 relative to p1.



NLN



NLN

To determine where p2 is located,

- Slope p1ptr < slope p1p2 < slope p1ptl

Find x and y intersection points.

Thank You

2D Viewing

Polygon and Text Clipping

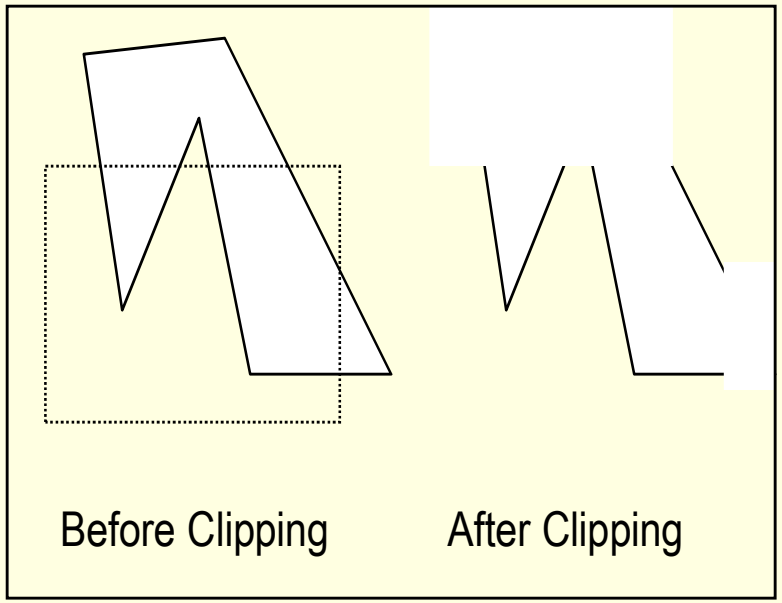
Overview

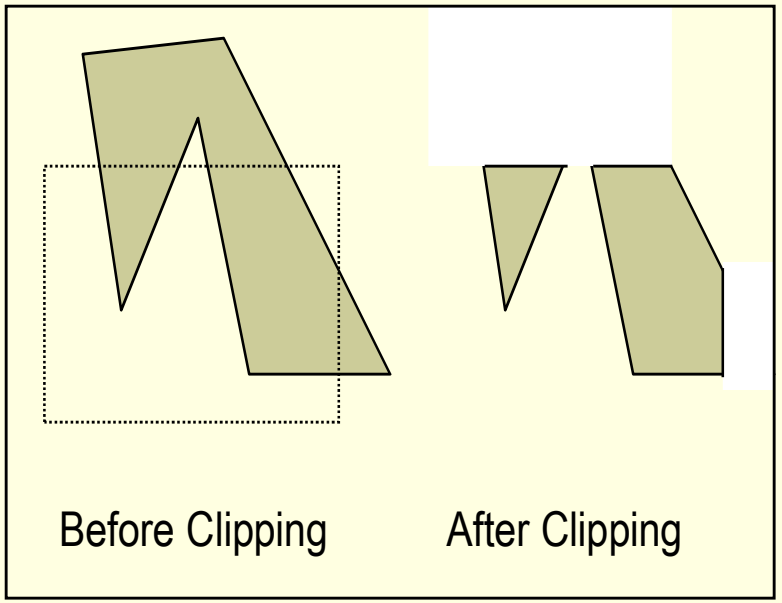
- Sutherland-Hodgeman Polygon Clipping
- Weiler-Atherton Polygon Clipping
- Text Clipping
- Exterior Clipping

Polygon Clipping

- To clip a polygon fill area, we cannot directly apply a line clipping method to the individual polygon edges.
- Because this approach would not, in general, produce a closed polyline.
- Instead, a line clipper would often produce a disjoint set of lines with no complete information about how we might form a closed boundary around the clipped fill area.

Sutherland-Hodgman Polygon-Clipping





Sutherland-Hodgeman Polygon Clipping Algorithm (A divide-and-conquer strategy)

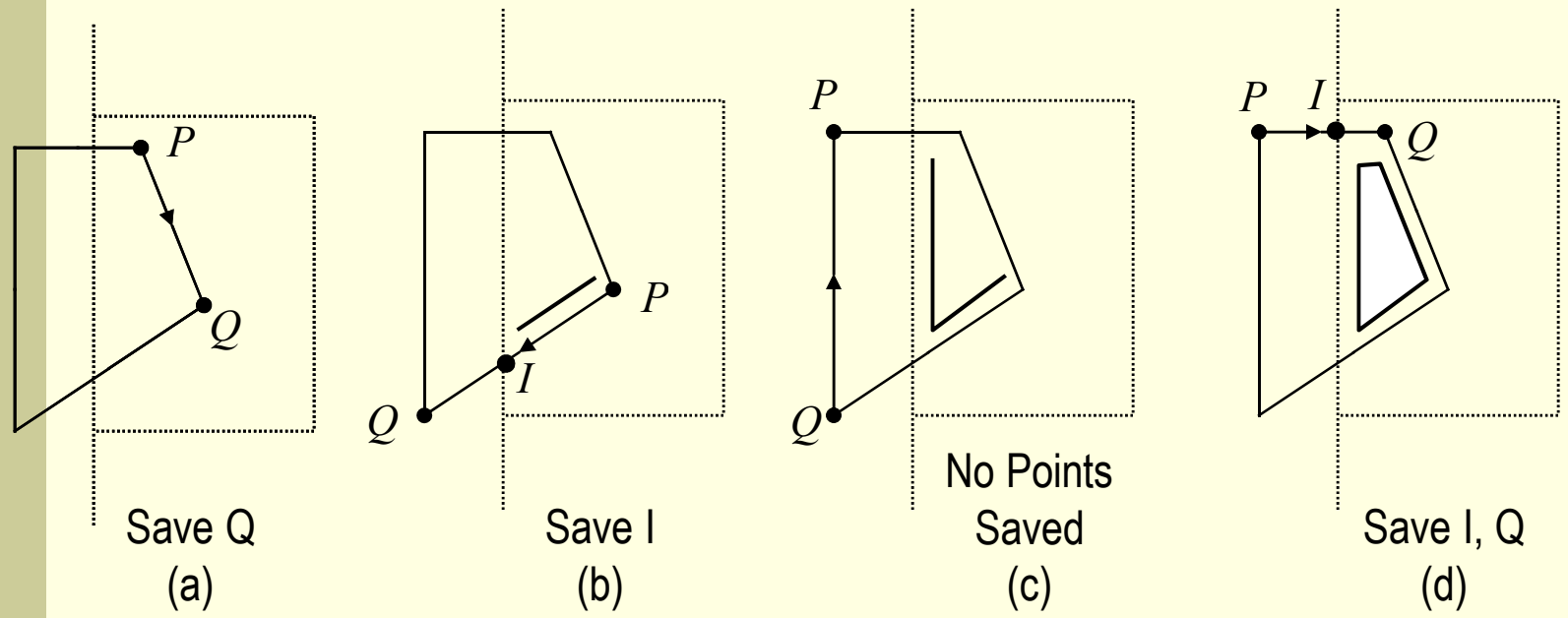
- Polygons can be clipped against each edge of the window one at a time.
- Clip the output polygon against the next edge.
- Repeat for all edges
- Edge intersections, if any, are easy to find since the *X* or *Y* coordinates are already known.
- Note that the number of vertices usually changes and will often increases.

Sutherland-Hodgeman Polygon Clipping Algorithm

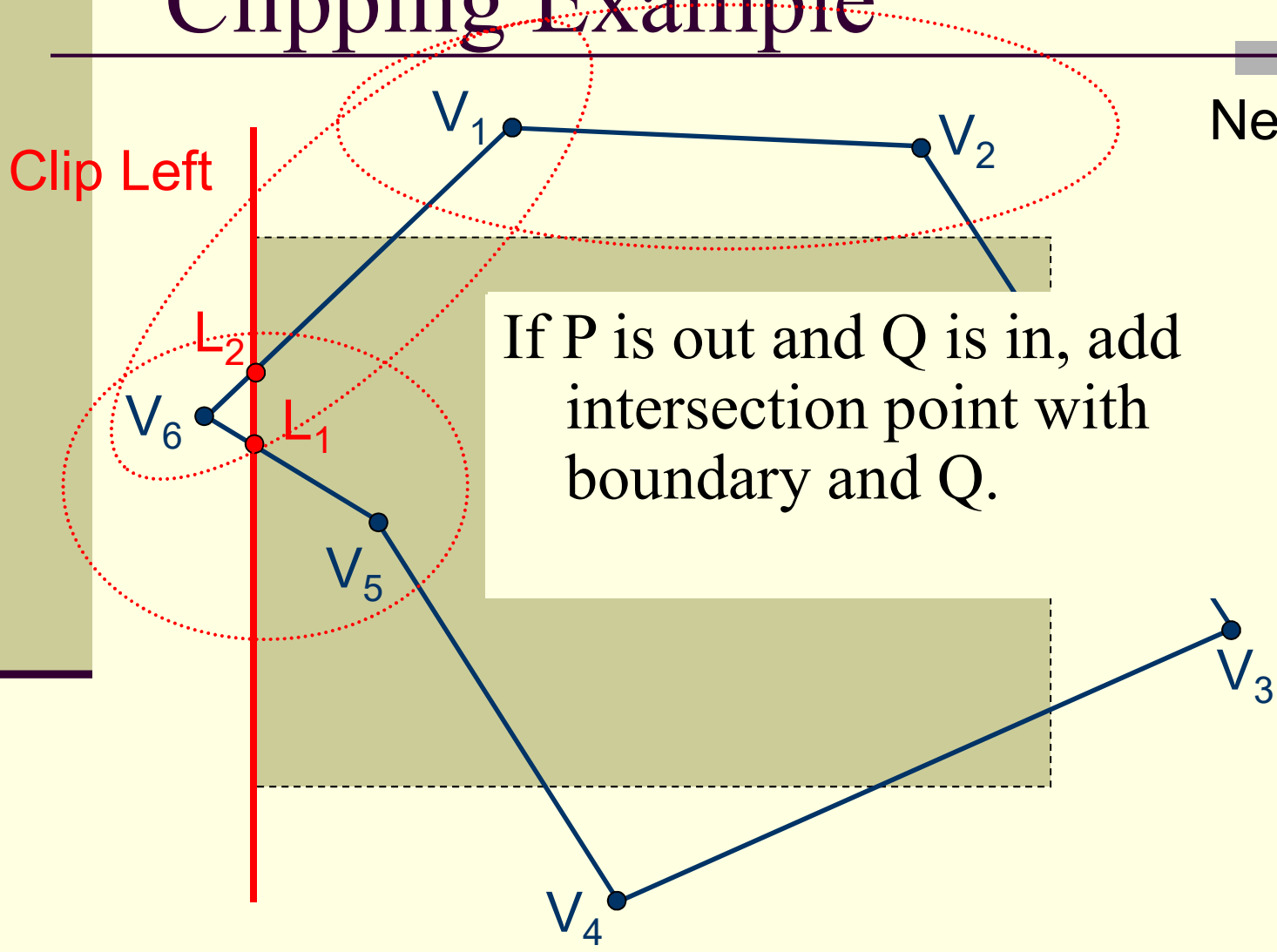
- Clip one boundary at a time: left, top, right, bottom.
- Check each adjacent pair of vertices (P,Q), in order to make a new vertex list.

1. If P and Q are in, add Q.
2. If P is in and Q is out, add the intersection point with boundary only.
3. If P and Q are both out, add nothing.
4. If P is out and Q is in, add intersection point with boundary and Q.

Sutherland-Hodgeman Algorithm(*cont.*)



Sutherland-Hodgeman Clipping Example



If P is out and Q is in, add intersection point with boundary and Q.

New Vertex List

V_2

V_3

V_4

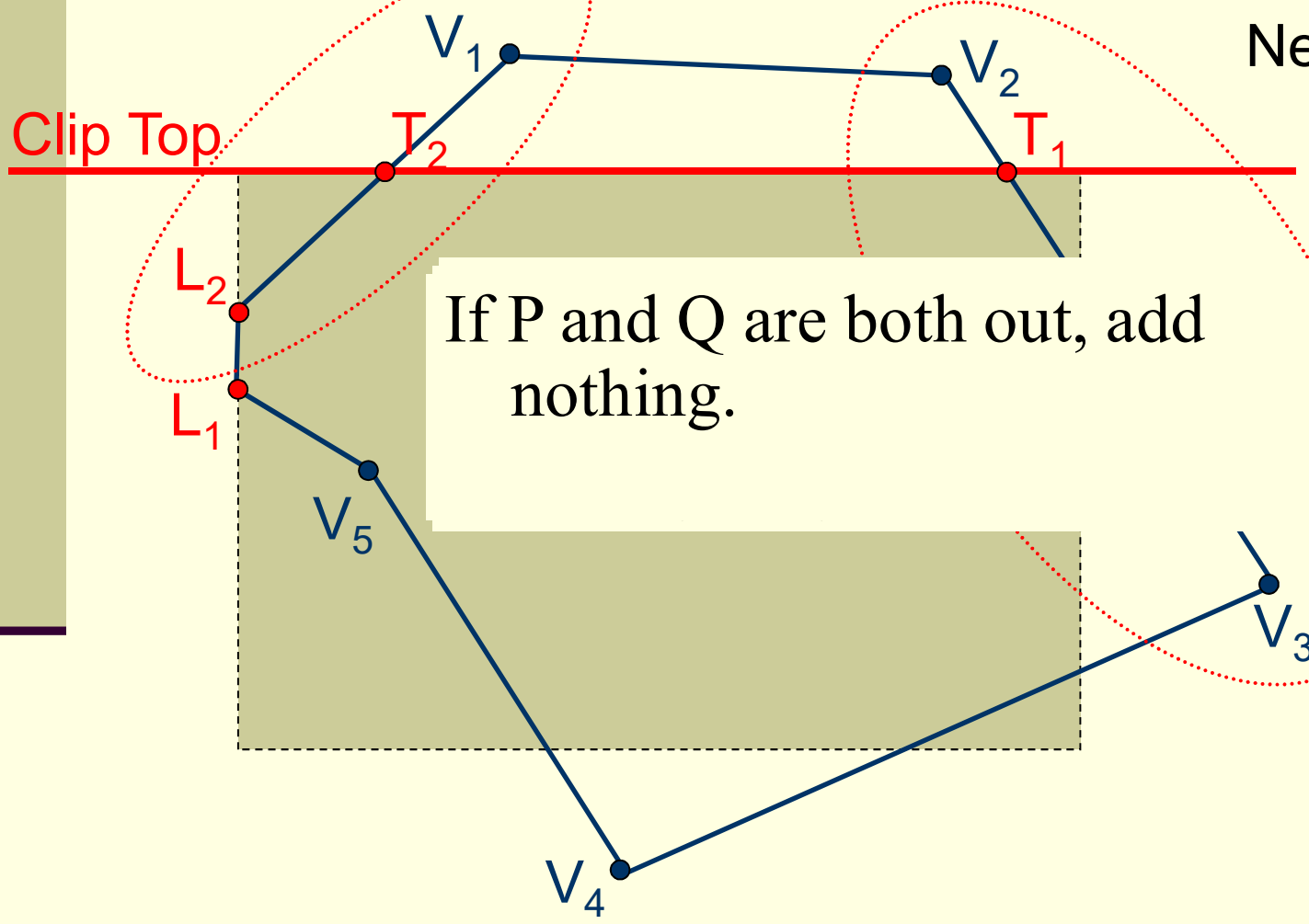
V_5

L_1

L_2

V_1

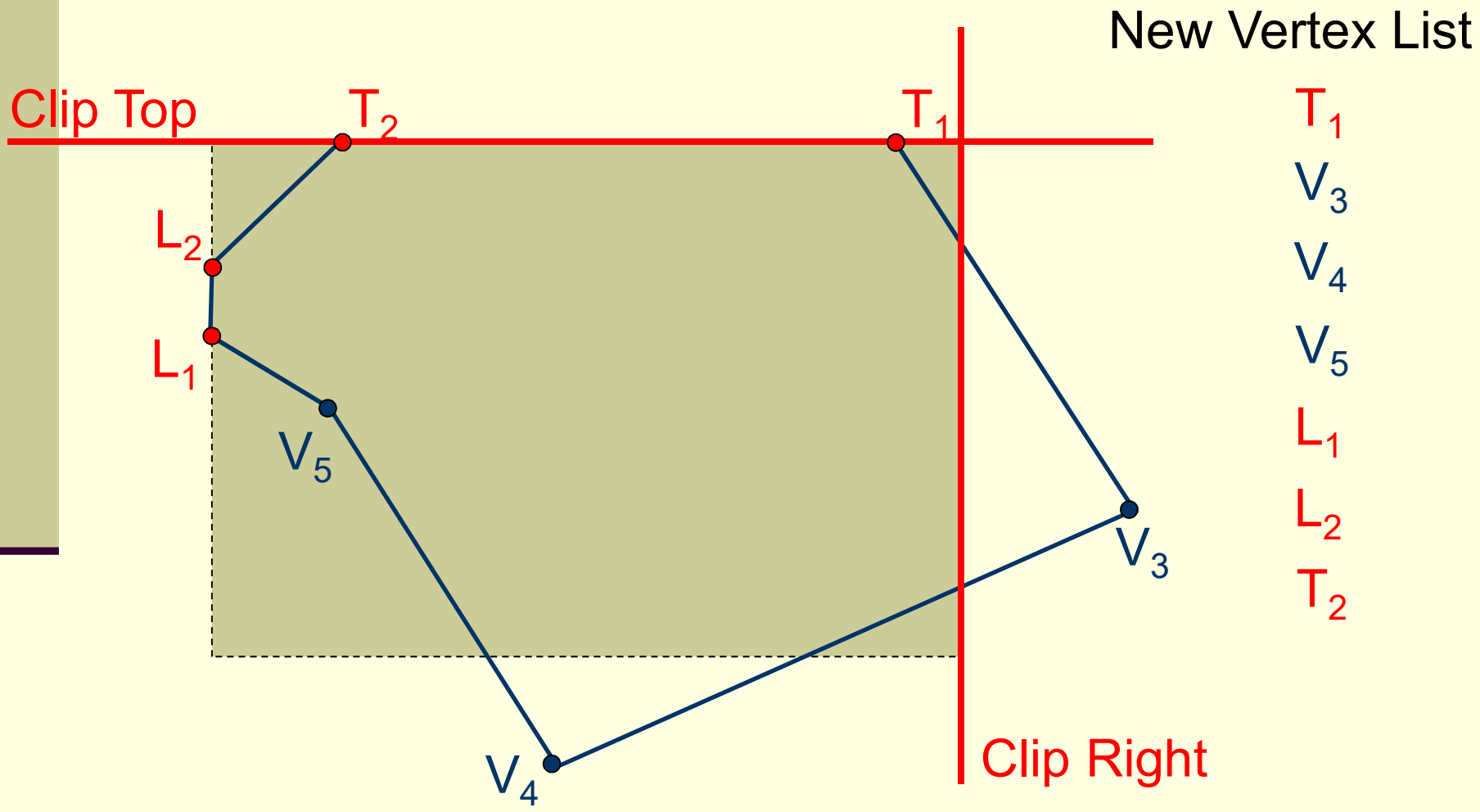
Sutherland-Hodgeman Clipping Example



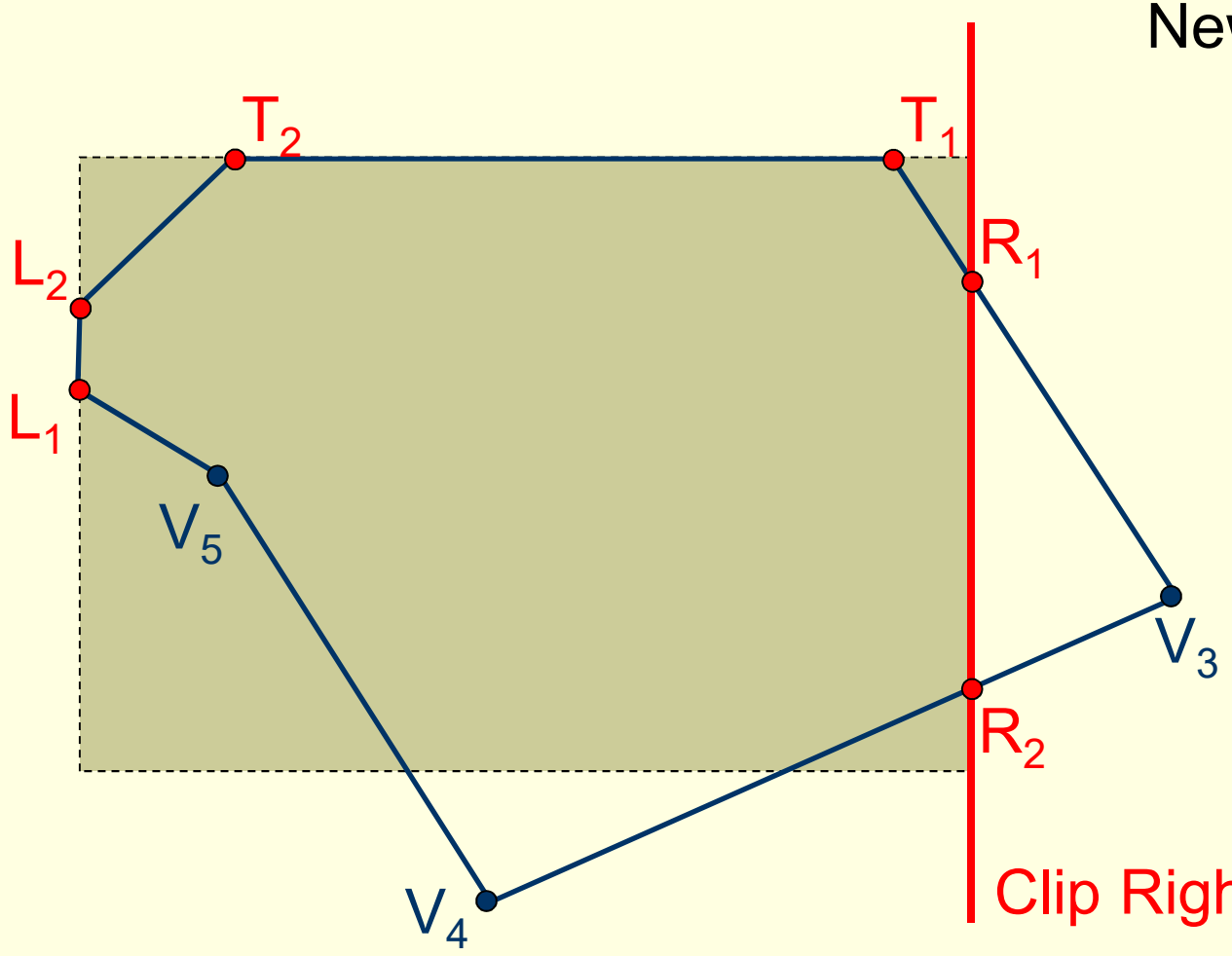
New Vertex List

T_1
V_3
V_4
V_5
L_1
L_2
T_2

Sutherland-Hodgeman Clipping Example



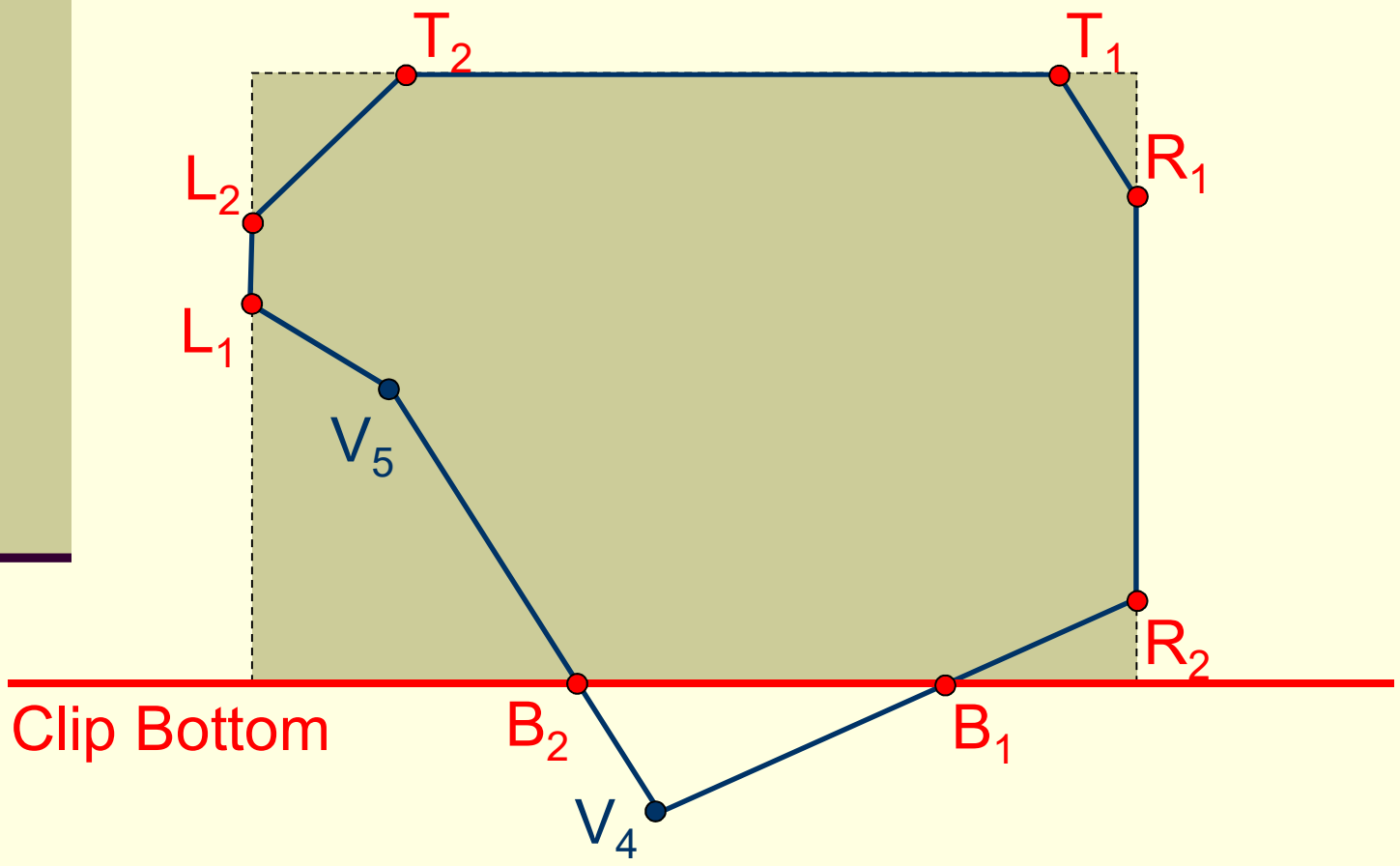
Sutherland-Hodgeman Clipping Example



New Vertex List

R_1
R_2
V_4
V_5
L_1
L_2
T_2
T_1

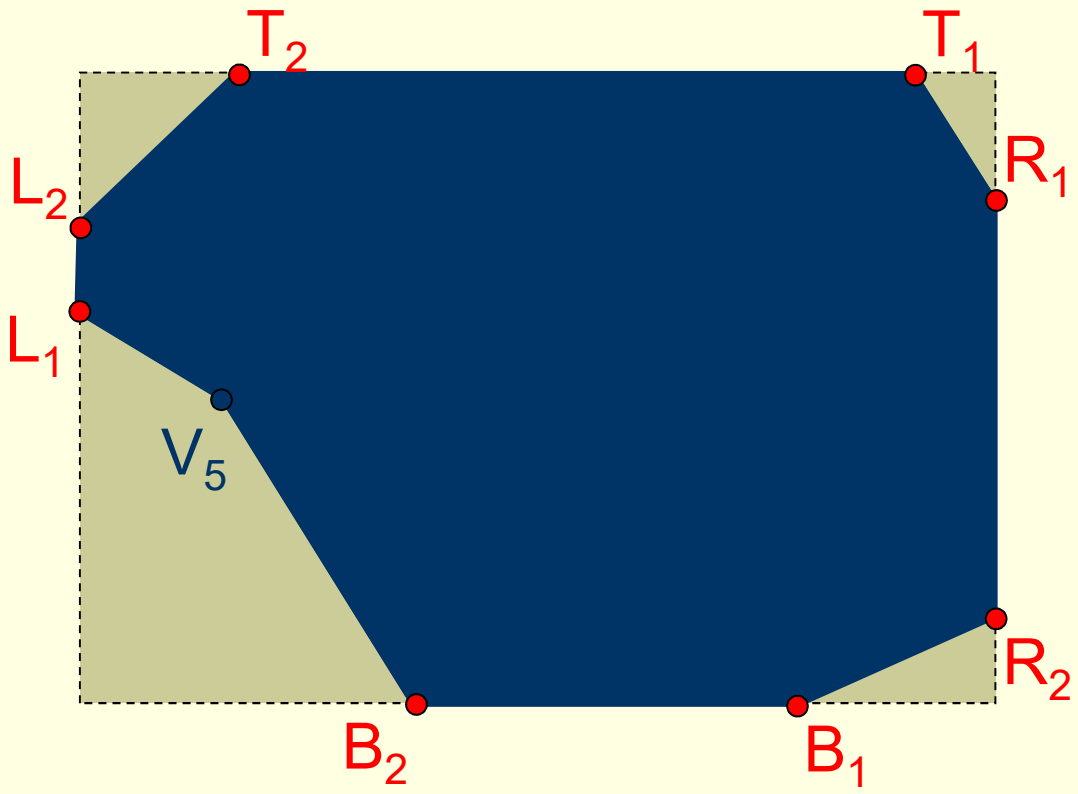
Sutherland-Hodgeman Clipping Example



New Vertex List

R_2
B_1
B_2
V_5
L_1
L_2
T_2
T_1
R_1

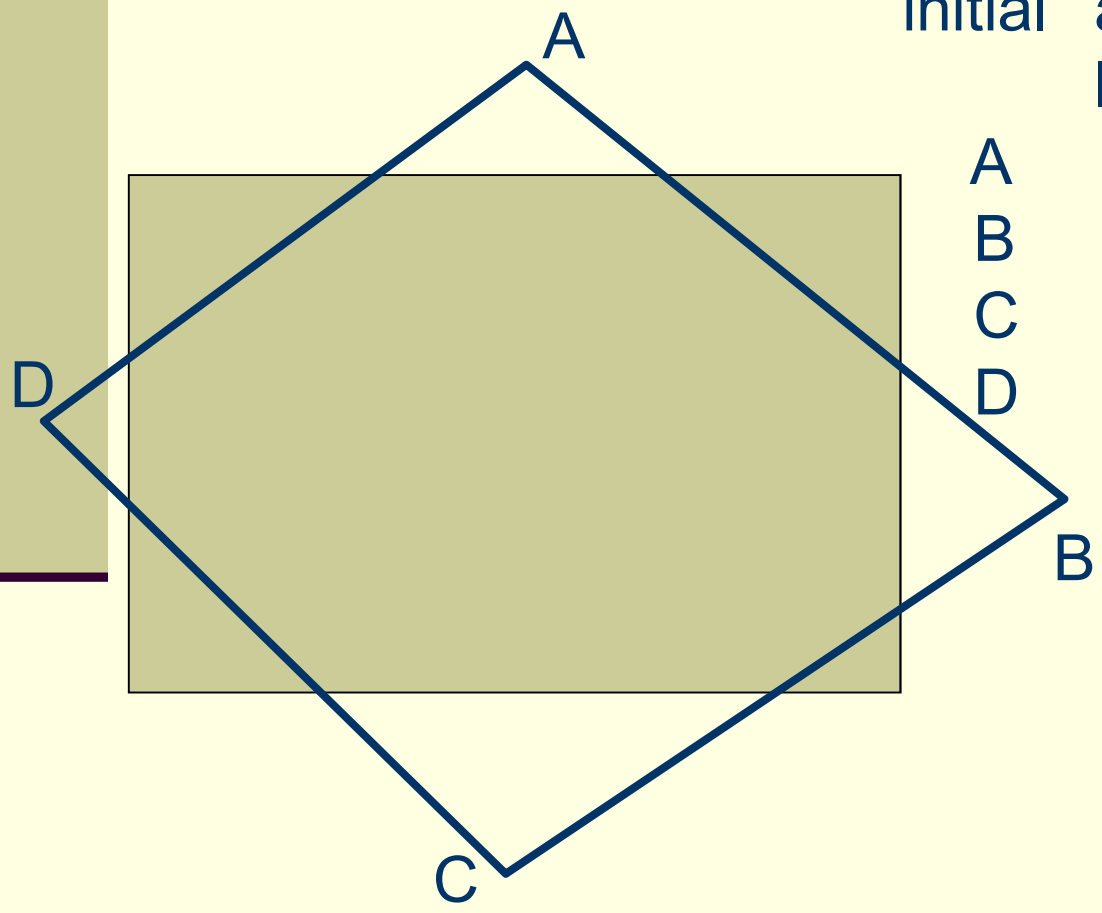
Sutherland-Hodgeman Clipping Example



New Vertex List

R_2
B_1
B_2
V_5
L_1
L_2
T_2
T_1
R_1

Sutherland-Hodgeman Exercise 1



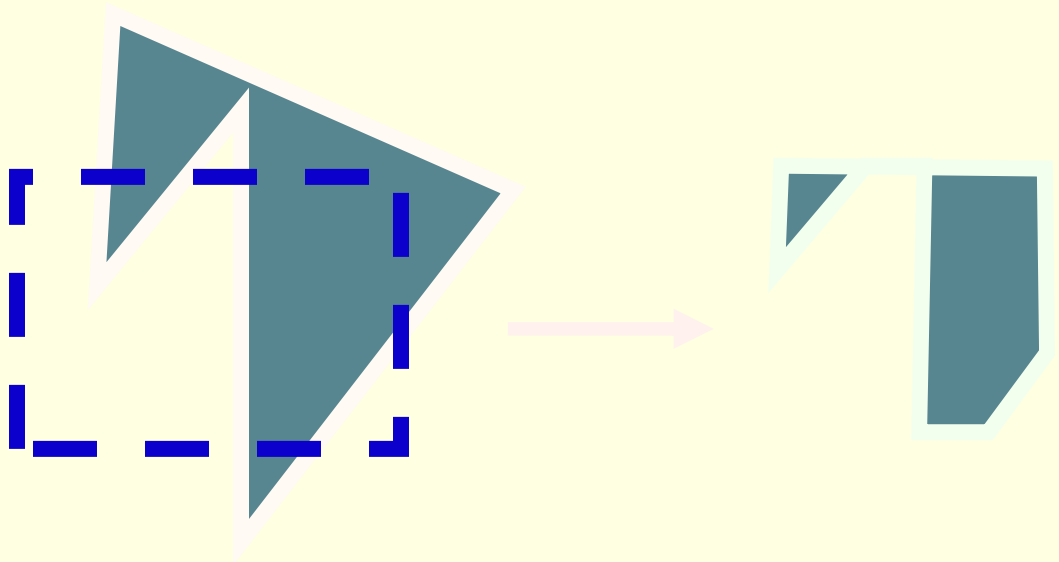
Vertex List

initial	after left	after top	after right	after bottom
A				
B				
C				
D				

Sutherland-Hodgeman Polygon Clipping Algorithm

- Convex polygons are correctly clipped by sutherland hodgeman algorithm
- Concave polygons may be displayed with extraneous lines.
- Occurs when clipped polygon have two separate sections.
- Only one output vertex list, the last vertex in the list is always joined to the first vertex

Weiler-Atherton Clipping



- A different clipping algorithm, the Weiler-Atherton algorithm, creates separate polygons

Weiler-Atherton Clipping

- The vertex processing procedures for window boundaries are modified so that concave polygons are displayed.
- Consider the window boundaries along with the polygon edges.
- Which path to follow depends on the polygon processing direction.
- For clockwise processing of polygon vertices, use the following rules:
  - 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 clockwise direction

Weiler Atherton Clipping

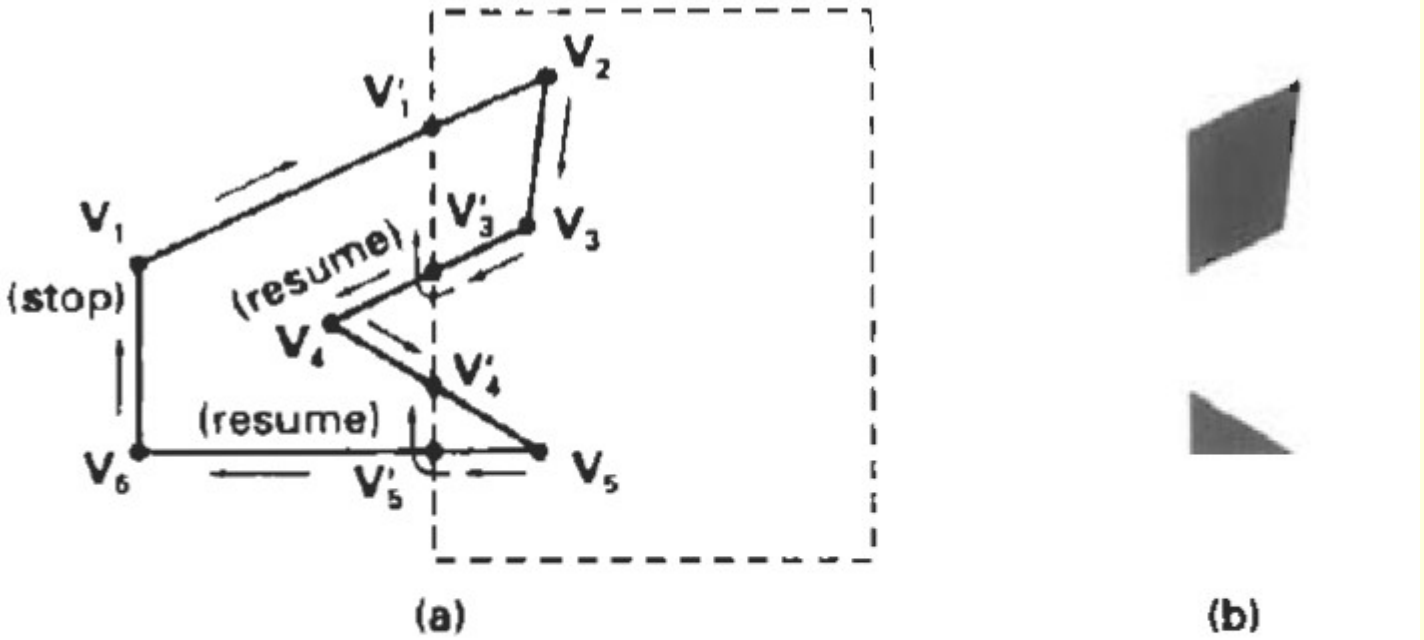
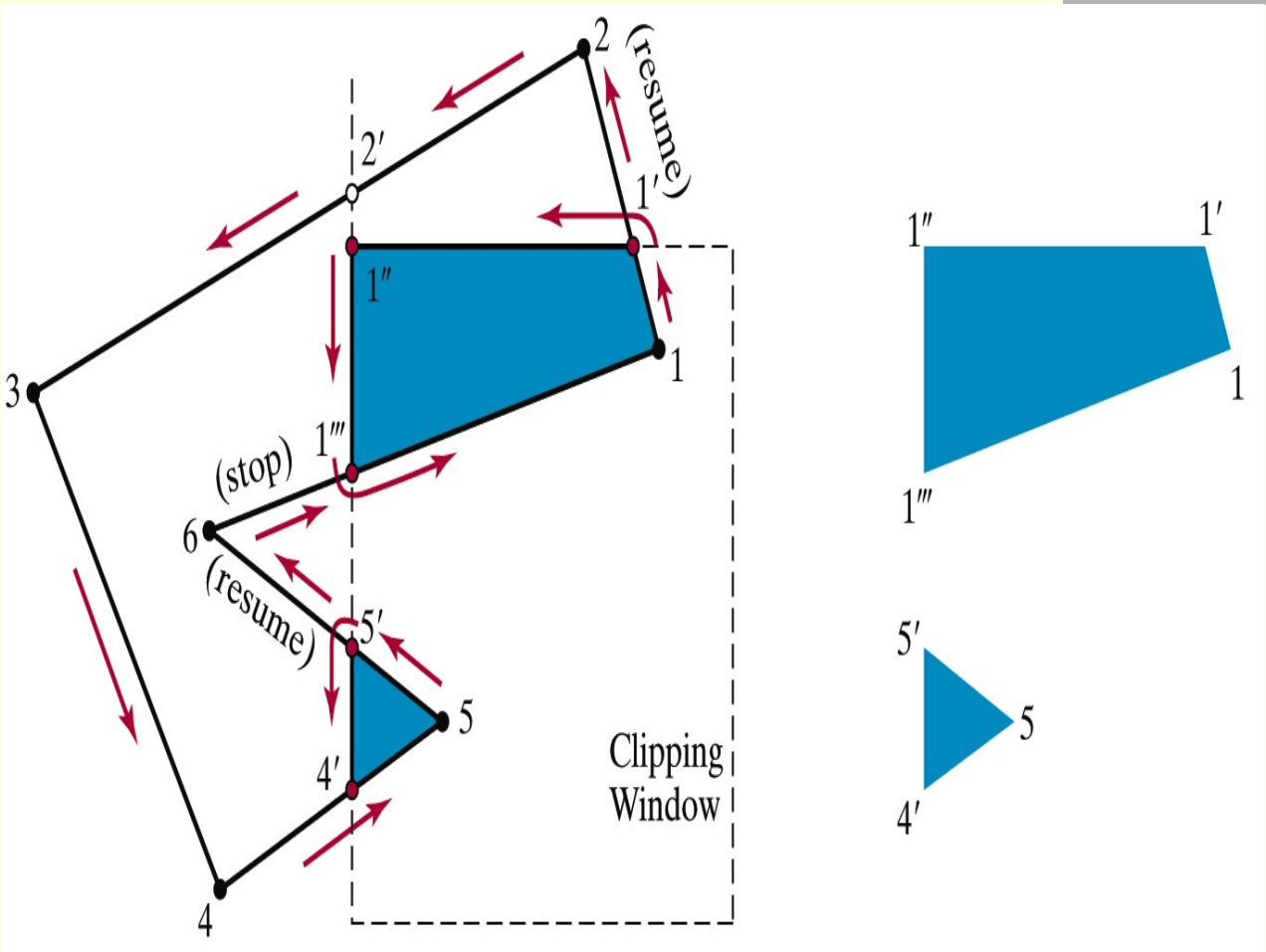


Figure 6-25
Clipping a concave polygon (a) with the Weiler–Atherton algorithm generates the two separate polygon areas in (b).

Weiler-Atherton Clipping



Weiler-Atherton Clipping

- Polygon clipping using nonrectangular polygon clip windows

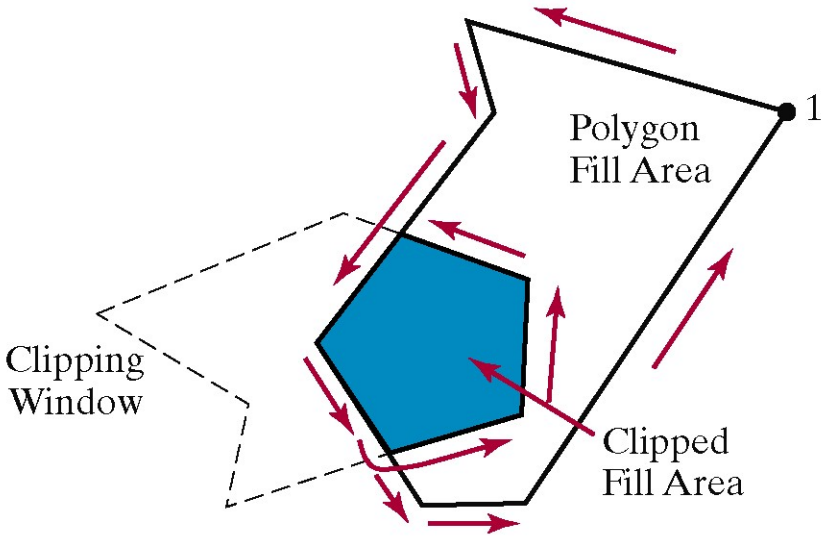


Figure 6-30

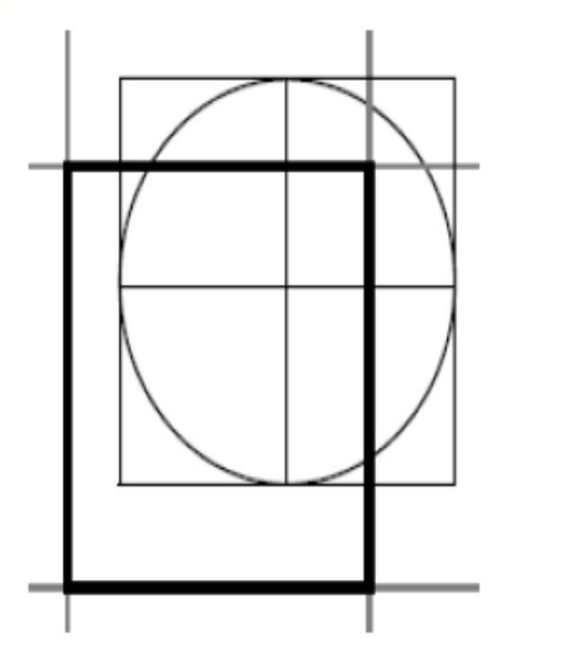
Clipping a polygon fill area against a concave-polygon clipping window using the Weiler-Atherton algorithm.

Curve Clipping

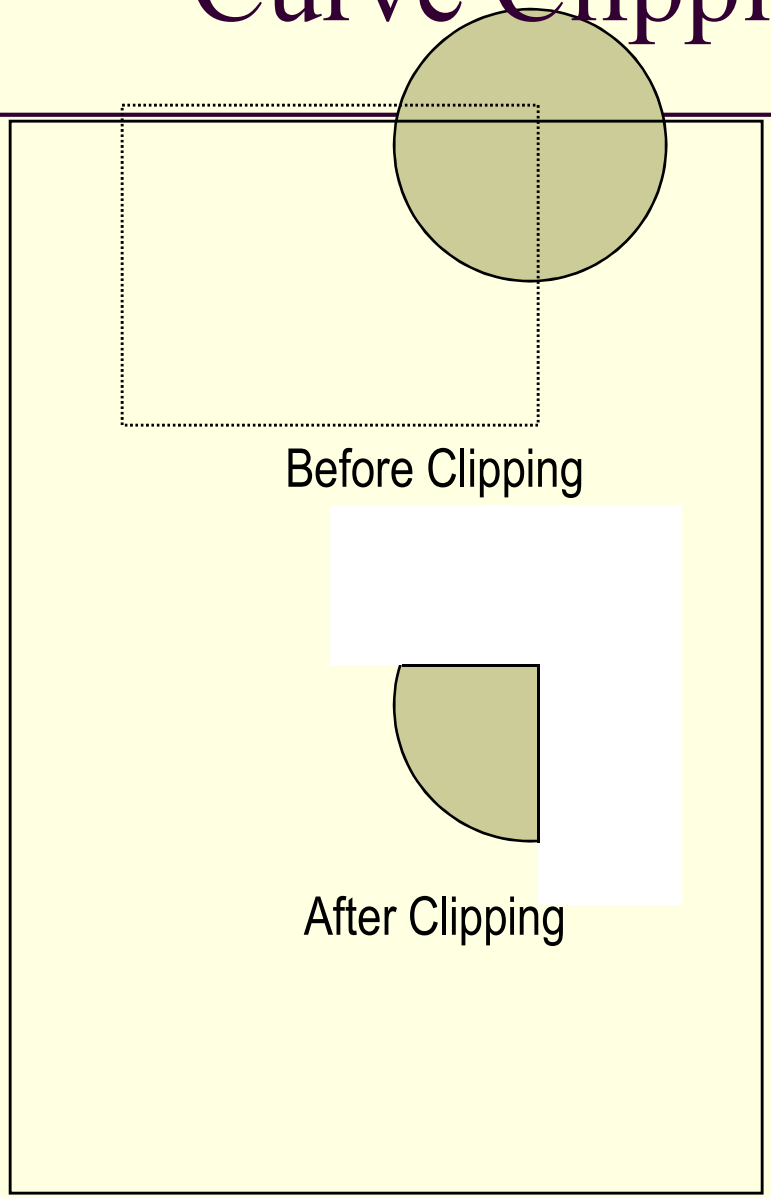
- A good strategy to deal with areas with curved boundaries is to utilize bounding information, e.g.:
- Check bounding box for trivial accept/reject
- Either case , no computation is required.

Before calculating the intersection test analytically :

- Use coordinate extension of individual quadrants.
- Use coordinate extension of individual octants



Curve Clipping



Text clipping

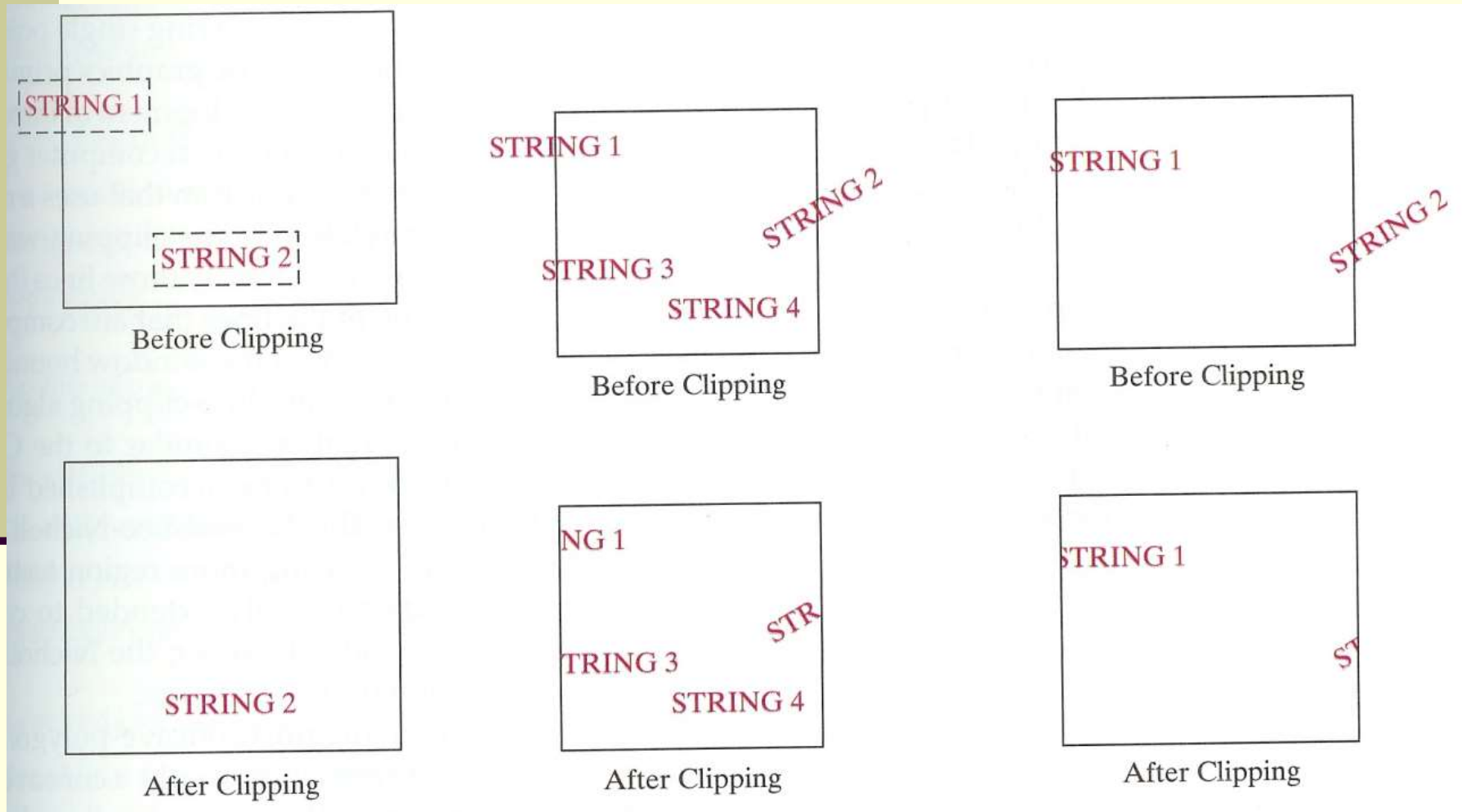
Text Clipping

- Several techniques used to provide text clipping in a graphics package.
- Technique depends on the methods used to generate characters and applications
- **All-or-none text clipping**
  - Using boundary box for the entire text
  - If all text inside the clip window , keep it
  - String is discarded if there is any overlap of bounding rectangle with window boundary.

Text Clipping

- **All-or-none character clipping**
  - Using boundary box for each individual character
  - The boundary limits of the individual characters are compared to the window.
  - Any character which is outside or overlaps a window boundary is clipped.
- **Character Component Clipping**
  - If individual character overlaps a clip window boundary clip off the parts of the character that are outside the window.
  - Vector font: Clip boundary polygons or curves
  - Bitmap font: Clip individual pixels

Text Clipping



Exterior Clipping

- When we want to clip a picture to the exterior of a specified region .
- The picture to be saved are those that are outside the region, known as exterior clipping.
- Example : Multiple window systems
- Objects within the window are clipped.
- Other high priority windows overlap these objects , they are clipped to the exterior of the overlapping windows.

- Thank You