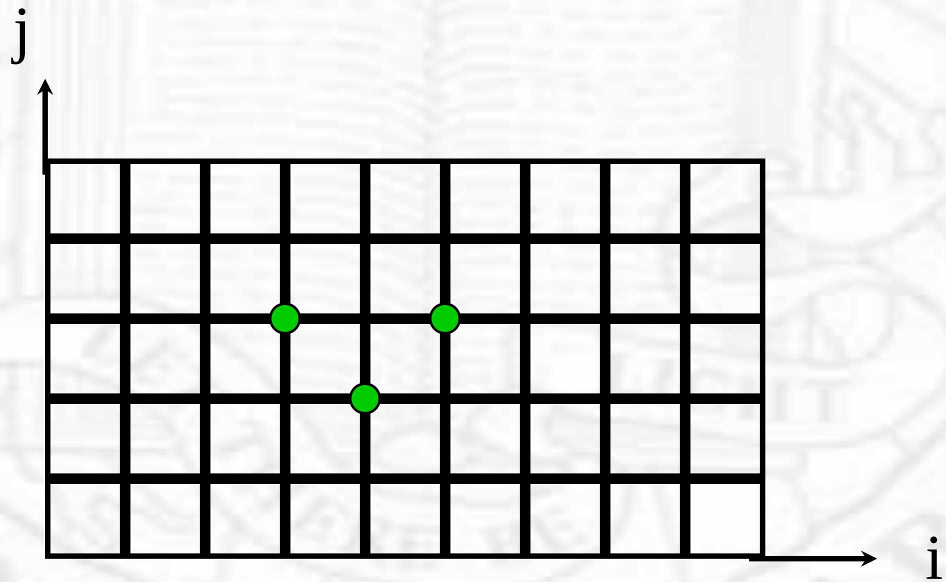
2D Graphics

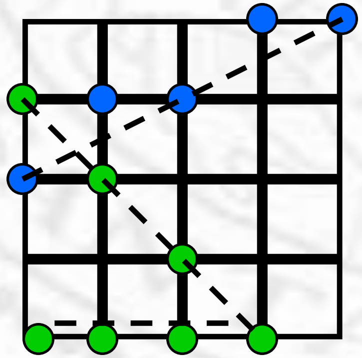
2D Raster Graphics

- Integer grid
- Sequential (left-right, top-down) scan



Line drawing

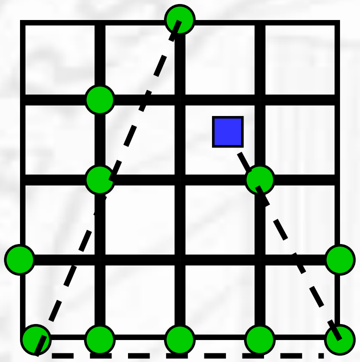
- A very important operation
  - used frequently, block diagrams, bar charts, engineering drawing, architecture plans, etc.
  - curves as concatenation of small line segments
- Criteria
  - line should appear straight



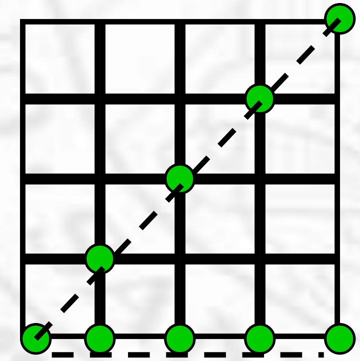
illuminate nearest grid point

Line drawing

- Line should terminate correctly
- Line should have a constant intensity



specify both end points
instead of end point + slope +length



$5/4\sqrt{2}$

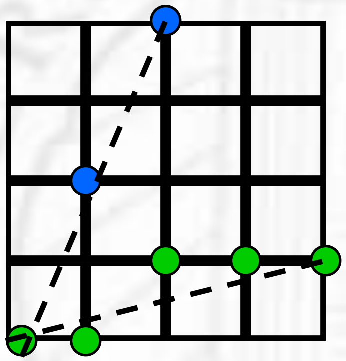
brightness adjustment
(antialiasing)

5/4

intensity ~ # of dots/unit length

Line drawing

- Line should not have “gaps”



y=f(x)

y=f(x) if |slope|<1
x=f(y) if |slope|>1

Line drawing

- Line should be drawn as fast as possible
  - Brute-force method
  - DDA (digital differential analyzer)

$$y = mx + b \Rightarrow$$ 1 fp *

$$for(i = x_o; i < x_n; i++)$$ 1 fp +

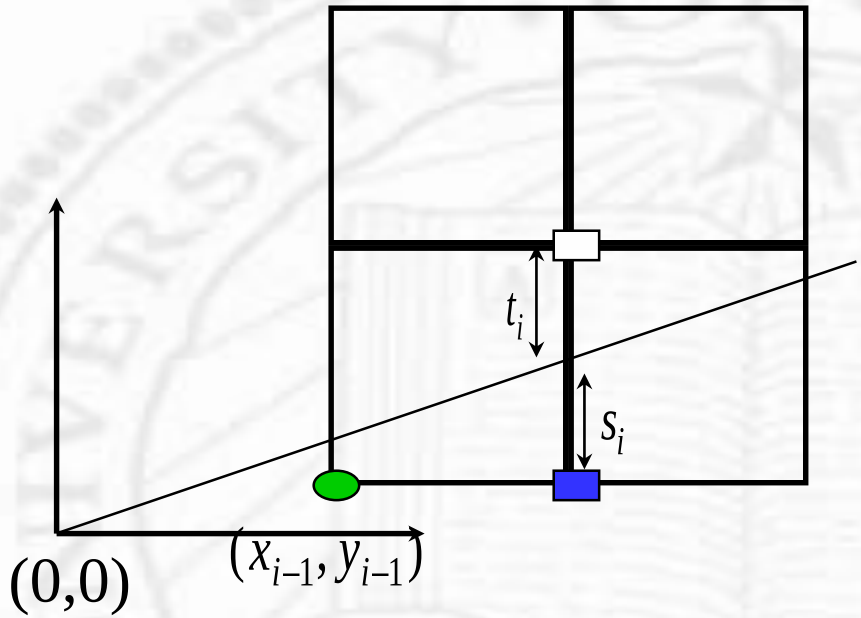
$$y_i = m \cdot i + b$$

$$\begin{aligned} y_{i+1} &= mx_{i+1} + b \\ &= m(x_i + 1) + b \\ &= mx_i + b + m \\ &= y_i + m \end{aligned}$$

1 fp +

Bresenham's Line Algorithm

integer operations only

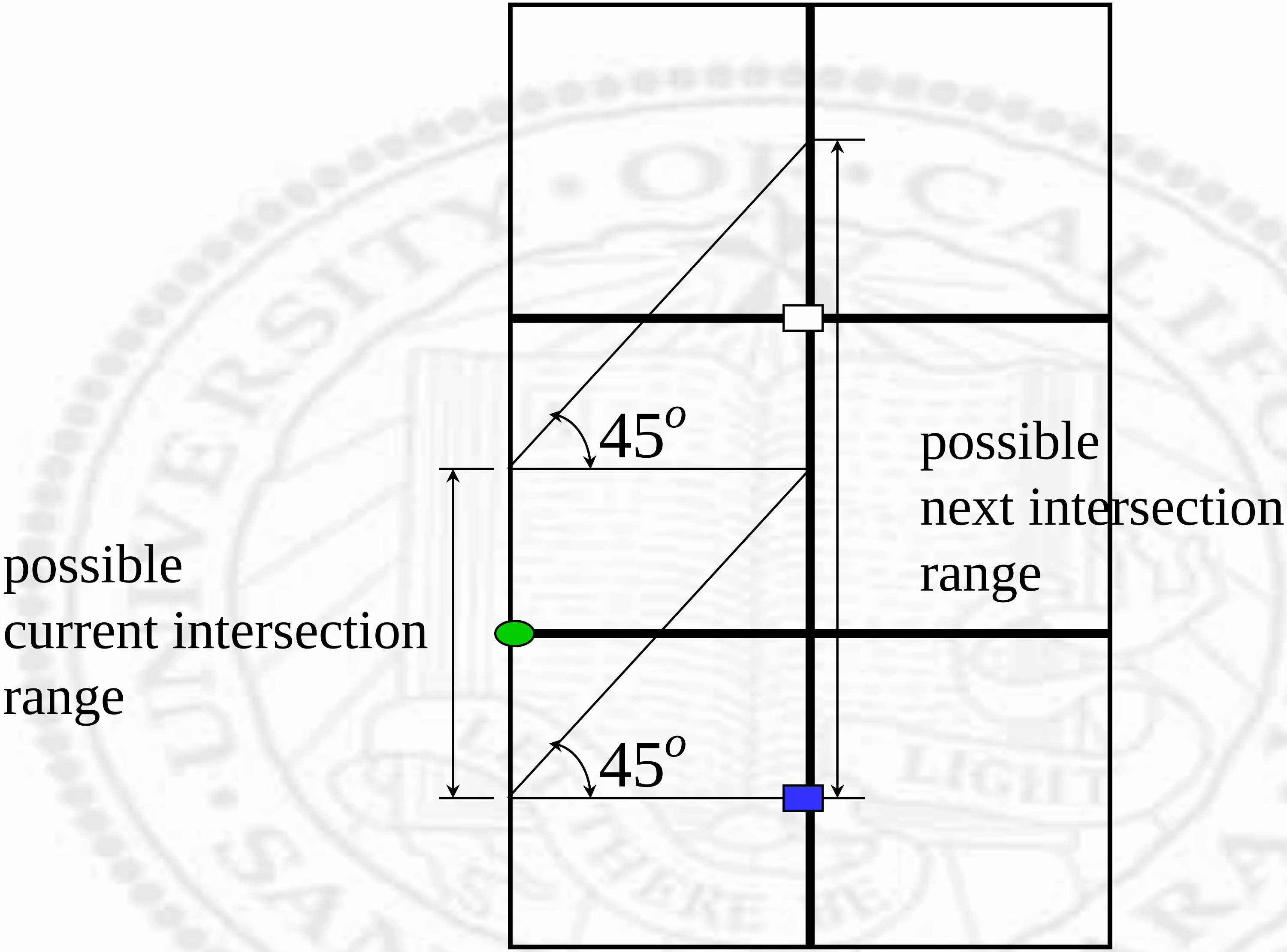


$$0 < slope < 1$$

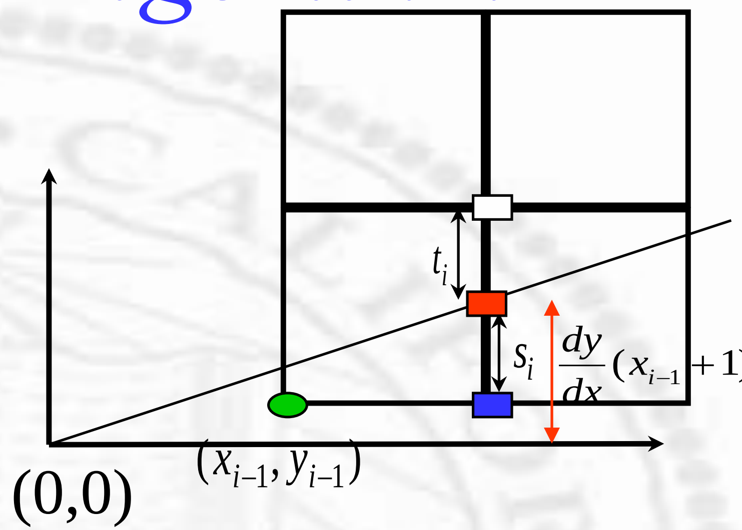
$$(0,0) \rightarrow (x_2 - x_1, y_2 - y_1)$$

$$if\ \ s > t \qquad or\ s - t > 0 \qquad \Rightarrow \square\ t_i$$

$$else\ s < t \qquad or\ s - t < 0 \qquad \Rightarrow \blacksquare\ s_i$$


45°
45°
possible
next intersection
range
possible
current intersection
range

Bresenham's Line Algorithm



$$s_i = \frac{dy}{dx}(x_{i-1}+1) - y_{i-1}$$

$$t_i = (y_{i-1}+1) - \frac{dy}{dx}(x_{i-1}+1)$$

$$s_i - t_i = 2\frac{dy}{dx}(x_{i-1}+1) - 2y_{i-1} - 1$$

floating point

$$\underbrace{dx(s_i - t_i)}_{d_i} = 2(x_{i-1}dy - y_{i-1}dx) + 2dy - dx$$

Integer!!

Bresenham's Line Algorithm

$$d_i = 2(x_{i-1}dy - y_{i-1}dx) + 2dy - dx$$

$$d_{i+1} = 2(x_i dy - y_i dx) + 2dy - dx$$

$$\Rightarrow d_{i+1} - d_i = 2dy(x_i - x_{i-1}) - 2dx(y_i - y_{i-1})$$

$$\Rightarrow d_{i+1} - d_i = 2dy - 2dx(y_i - y_{i-1})$$

$$\Rightarrow d_{i+1} = d_i + 2dy - 2dx(y_i - y_{i-1})$$

Bresenham's Line Algorithm

$$d_{i+1} = d_i + 2dy - 2dx(y_i - y_{i-1})$$

$$if\ \ d_i \geq 0\ choose\ t_i$$

$$\Rightarrow y_i = y_{i-1} + 1$$

$$\Rightarrow d_{i+1} = d_i + 2(dy - dx)$$

$$if\ \ d_i < 0\ choose\ s_i$$

$$\Rightarrow y_i = y_{i-1}$$

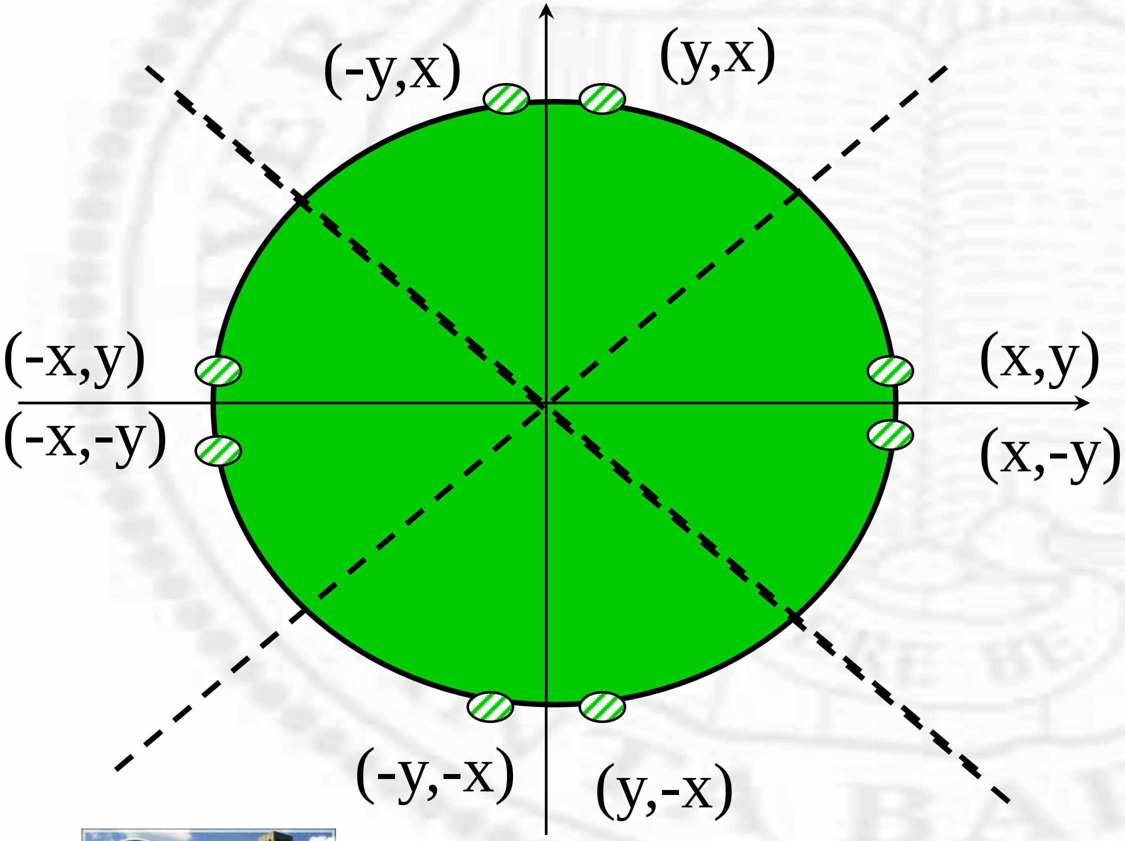
$$\Rightarrow d_{i+1} = d_i + 2dy$$

$$initial\ condition\ d_1 = 2dy - dx\ (x_0, y_0) = (0,0)$$

- Complexity: 1 left shift + 2 integer additions

Circle Drawing

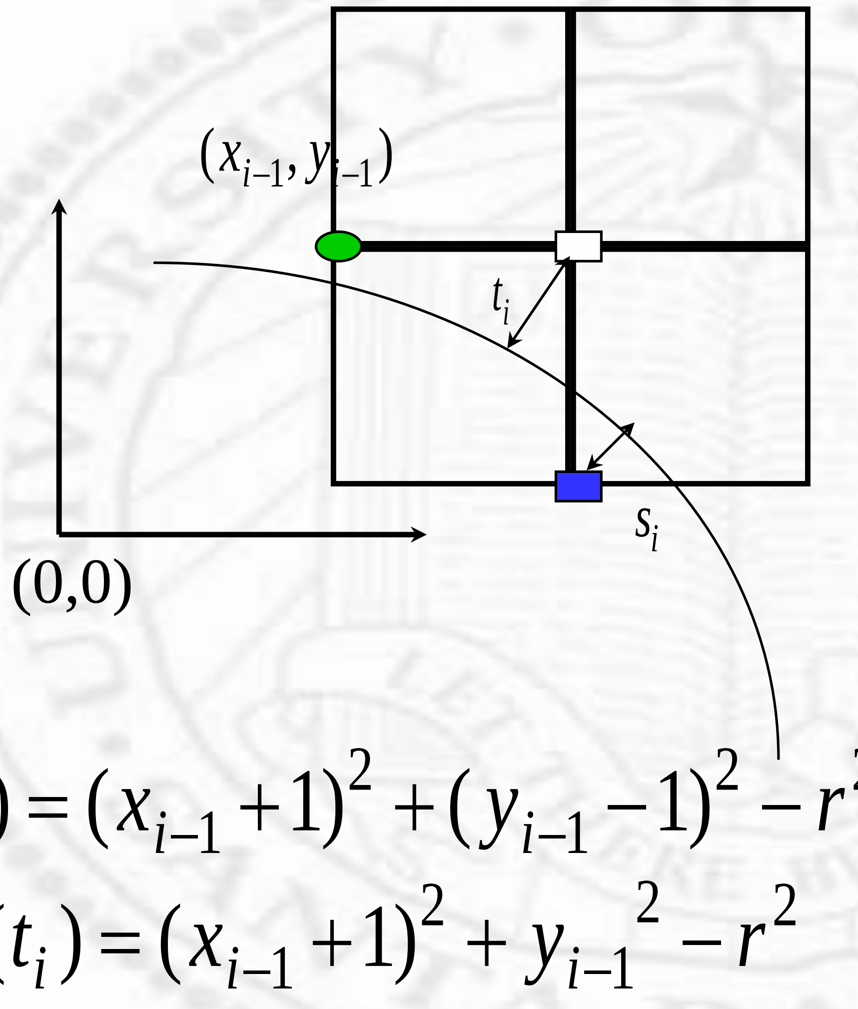
❖ Symmetry reduces drawing to 1/8



$$x = x_c + r\cos\theta$$
$$y = y_c + r\sin\theta$$

Bresenham's Circle Algorithm

integer operations only



$$(x_c, y_c) = (0,0)$$

$$45^o < \theta < 90^o$$

$$D(s_i) = (x_{i-1}+1)^2 + (y_{i-1}-1)^2 - r^2 \qquad |D(s_i)| > |D(t_i)| \Rightarrow t_i$$

$$D(t_i) = (x_{i-1}+1)^2 + {y_{i-1}}^2 - r^2 \qquad |D(s_i)| < |D(t_i)| \Rightarrow s_i$$

Bresenham's Circle Algorithm

$$d_i = |D(s_i)| - |D(t_i)| = -D(s_i) + D(t_i)$$

$$d_i = 2r^2 - 2(x_{i-1} + 1)^2 - (y_{i-1} - 1)^2 - y_{i-1}^2$$

$$d_{i+1} = 2r^2 - 2(x_{i-1} + 2)^2 - (y_i - 1)^2 - y_i^2$$

$$\Rightarrow d_{i+1} - d_i = -4x_{i-1} - 6 - 2(y_i^2 - y_{i-1}^2) - 2(y_i - y_{i-1})$$

Bresenham's Circle Algorithm

$$d_1 = -3 + 2r \quad (x_0, y_0) = (0, r)$$

$$if \;\; d_i \geq 0 \; choose \; t_i$$

$$\Rightarrow y_i = y_{i-1}, d_{i+1} = d_i - 4x_{i-1} - 6$$

$$if \;\; d_i < 0 \; choose \; s_i$$

$$\Rightarrow y_i = y_{i-1} - 1, d_{i+1} = d_i - 4x_i + 4y_i - 6$$

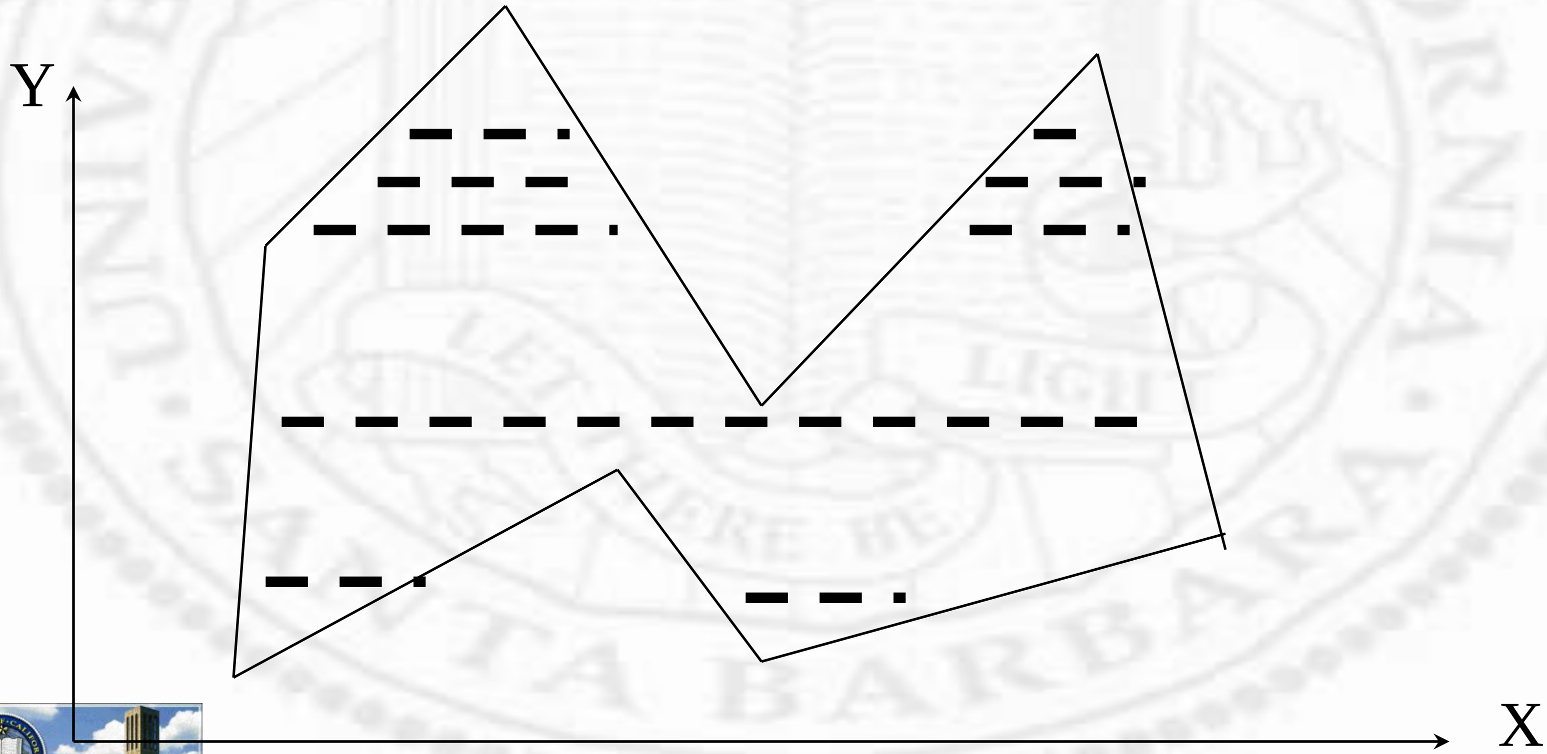
- Complexity: only integer and shift operations

Other primitives

- Ellipse
  - symmetry reduces to 1/4
  - Bresenhem's ellipse algorithm
- Curve
  - difficult
  - approximation using short line segments
  - general curve forms (Bezier, B-spline, etc.)
- Characters
  - rectangular grid patterns

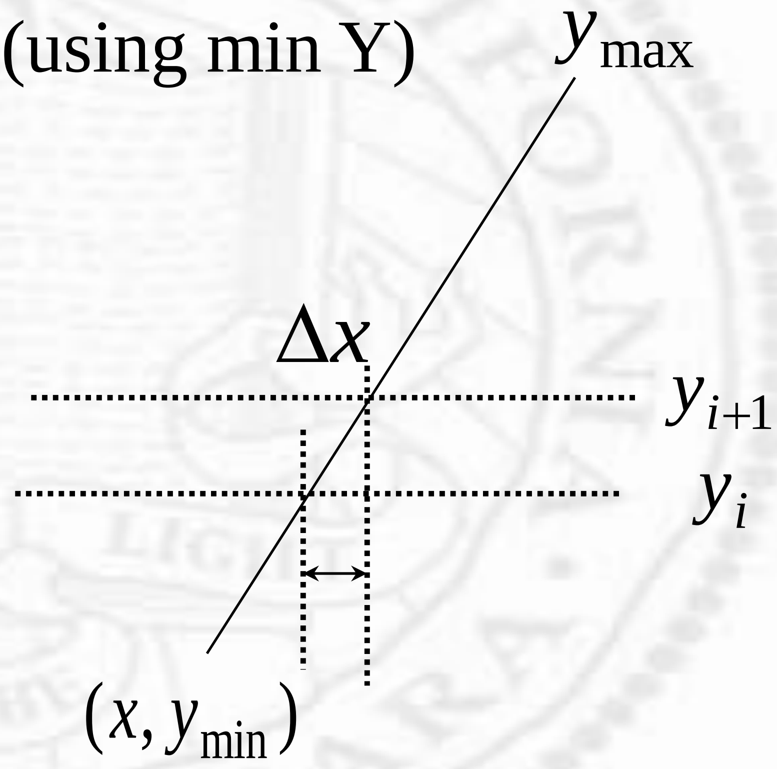
Polygon Filing

- Arbitrary # of sides
- Convex or concave
- Holes



Scan Line Algorithm

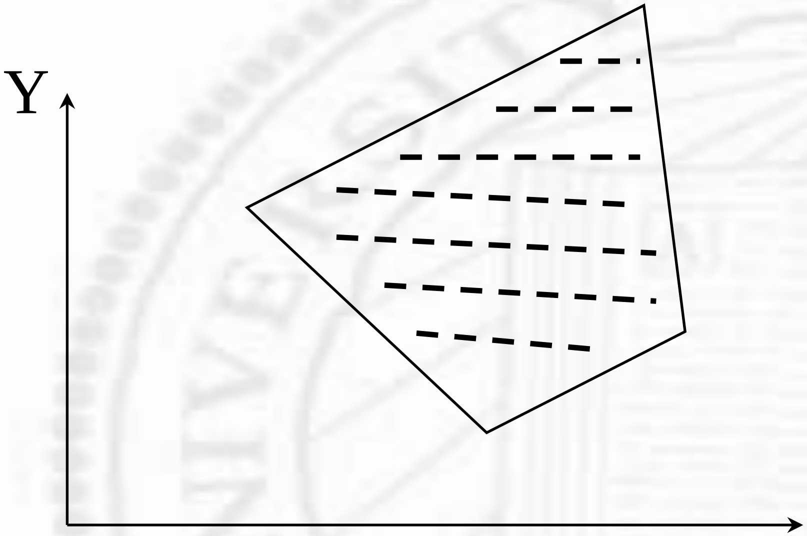
- Edge table
  - sort edges by scanline (using min Y)
  - record
    - x coordinate of ymin
    - ymax
    - Δx to be added



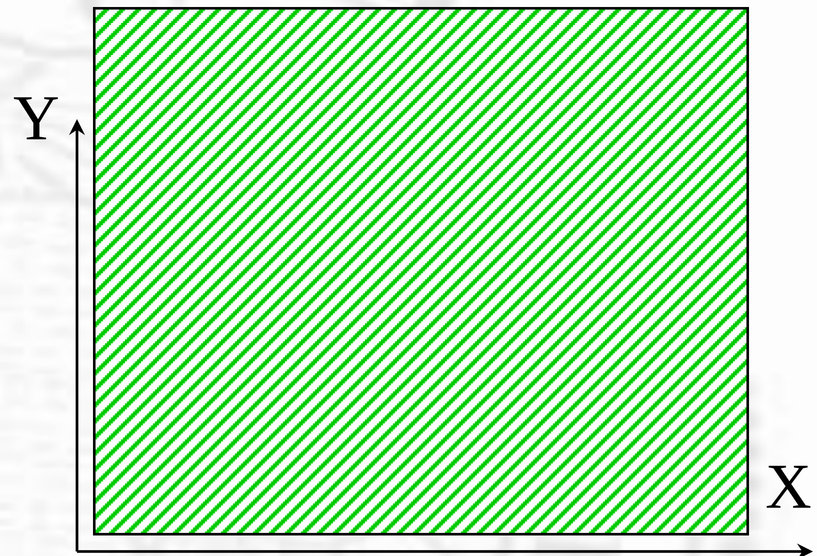
Scan Line Algorithm

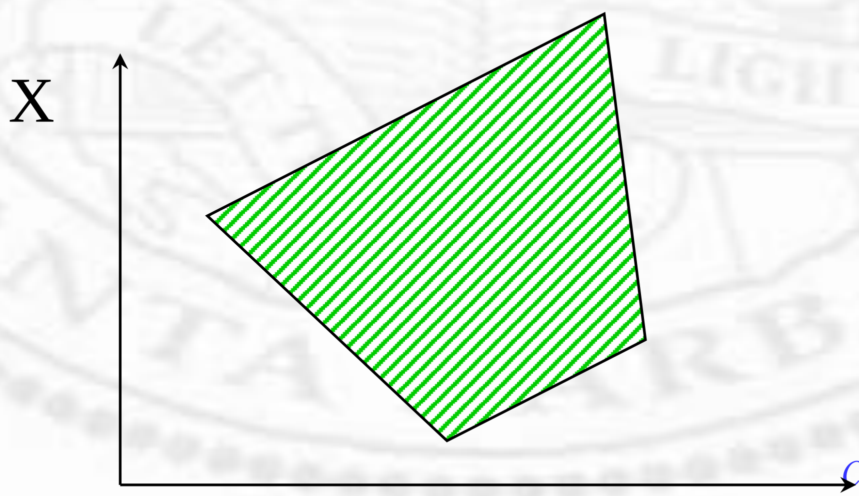
- Set y to smallest y in ET
- Initialize AET to be Null
- Repeat until AET and ET are empty
  - move from ET bucket y to AET those edges whose ymin=y
  - sort edges in AET by x (insertion sort)
  - fill in pixel values in between x pairs
  - remove from AET those edges whose ymax = y
  - increment y by 1
  - update x for all edges in AET $x \leftarrow x + \Delta x$

Polygon Patterned Filling



logical
and





Polygon Patterned Filling

- Pattern can be anchored at
  - a fixed point: transparent object moves on a patterned background
  - a polygon corner: patterned object

2D Transformation

- For animation, manipulation, user interaction
- translation, rotation, scaling

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} T_x \\ T_y \end{bmatrix}$$

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

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

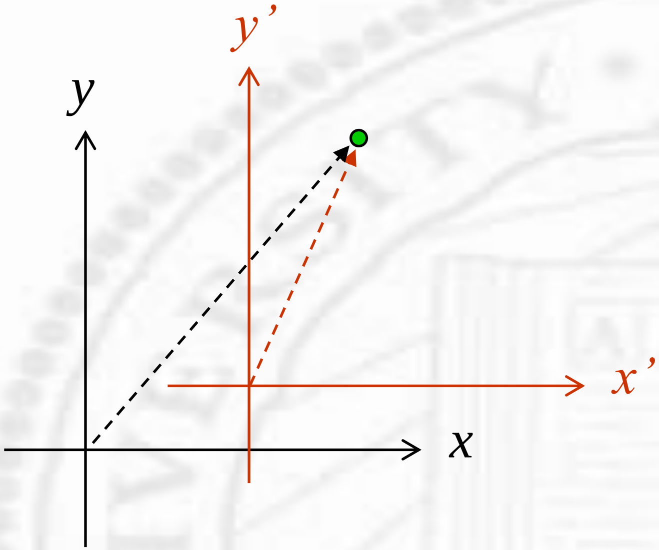
A Very Common Confusion

- What is being transformed? Points or coordinate system?
- For CG, pipeline operations are always applied to features (points, lines, curves, planes)
- But you can think in either way:
  - Points are physically moved in a fixed coordinate system (e.g., in modeling transform), or
  - A coordinate system is moved, while points stay stationary (e.g., in viewing transform)
  - Both interpretations are useful

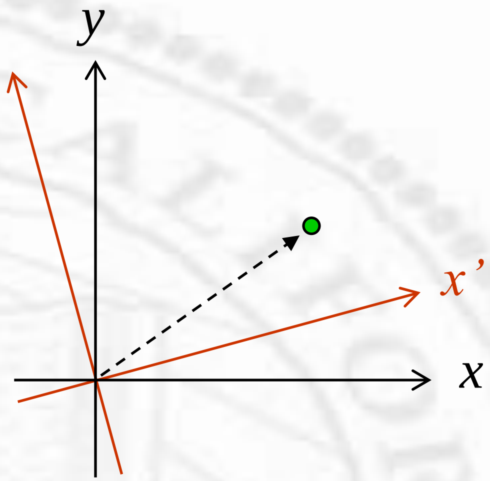
2D Rigid Transformations

- A rigid transformation maps one coordinate system into another
  - Preserves distances and angles
- To transform points from one coordinate frame to another, find the rigid transformation that brings the two coordinate frames in alignment
  - **Translate** so that their origins coincide
  - **Rotate** so that their axes coincide (x with x, y with y, and z with z)

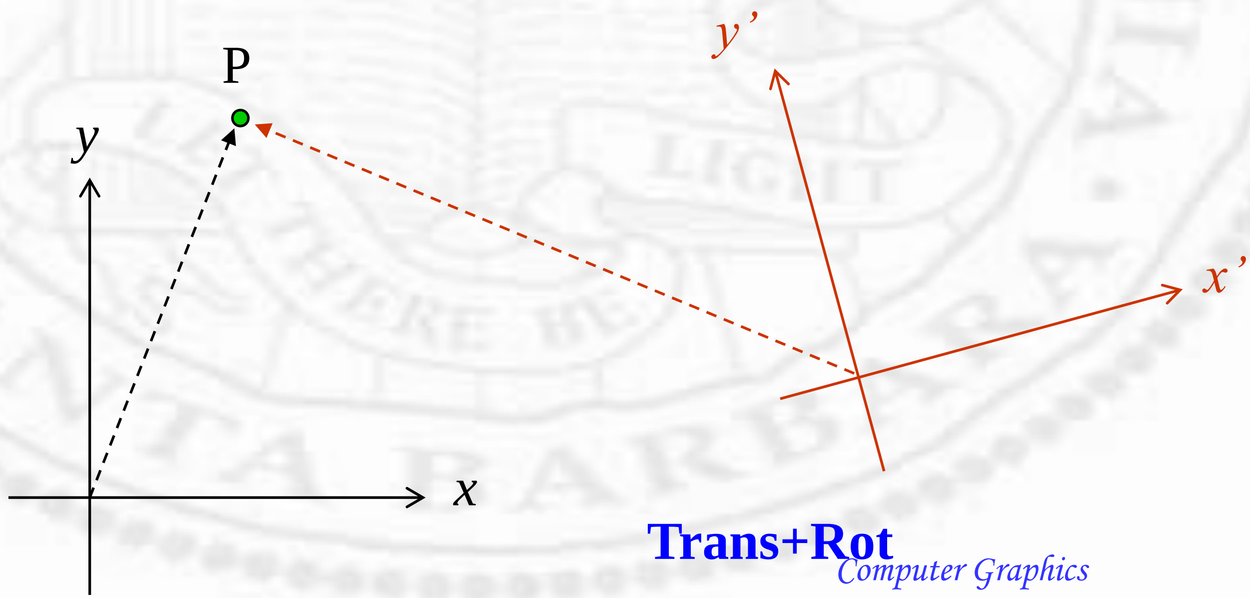
2D examples



Trans



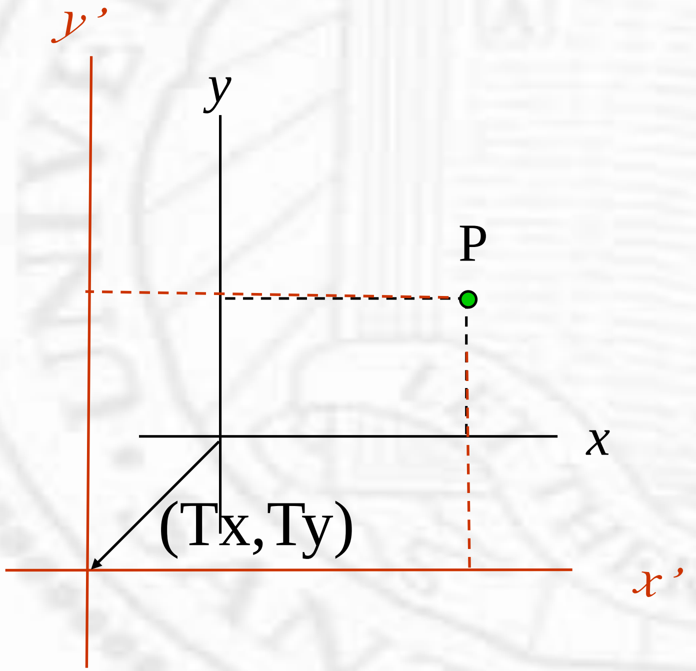
Rot



Trans+Rot

2D Translation

- Translate the coordinate system by (Tx,Ty)
  - What is the translated point?

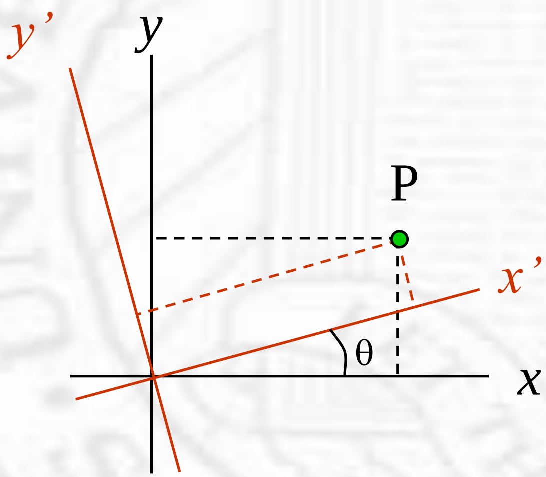


$$P' = P + \begin{bmatrix} -T_x \\ -T_y \end{bmatrix}$$

2D rotation matrix

- Rotate θ counterclockwise
  - What is the transformation R?

$$P' = RP$$

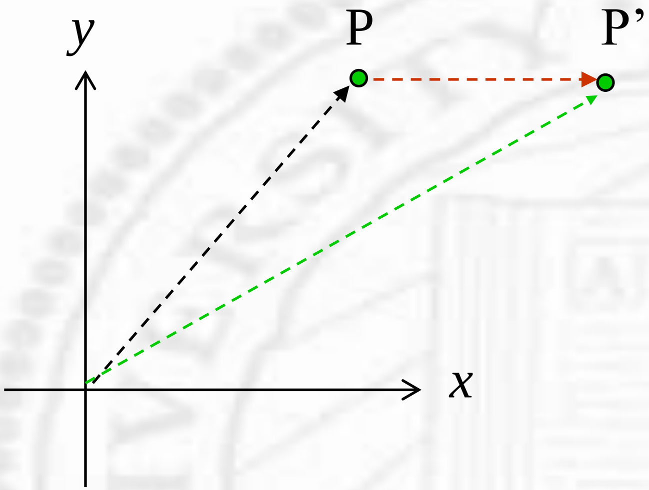


$$P' = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} P$$

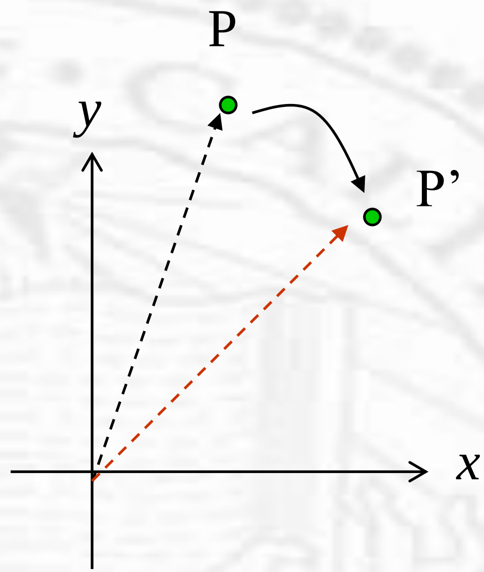
2D Rigid Transformations

- A rigid transformation moves an object from one location to another location
- Preserves distances and angles
- To transform points from one place to another, find the rigid transformation that
  - **Translate** so that the object moves
  - **Rotate** so that the object reorients

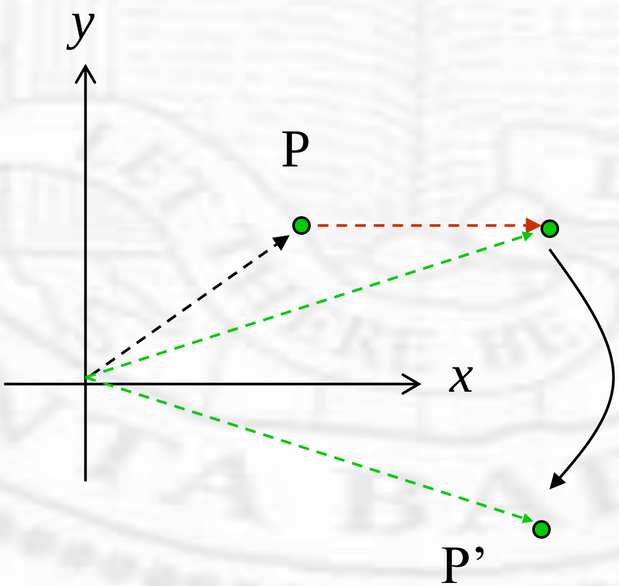
2D examples



Trans



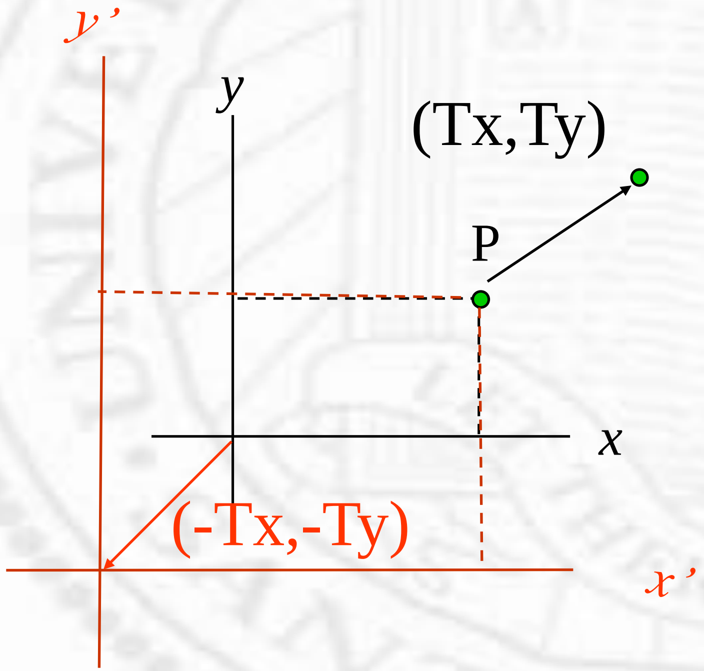
Rot



Trans+Rot

2D Translation

- Translate the coordinate system by (Tx,Ty)
  - What is the translated point?

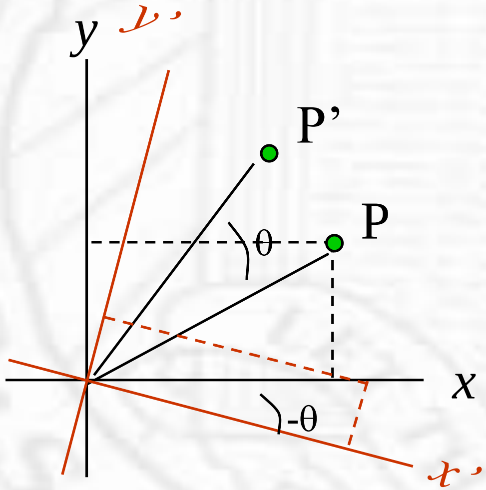


$$P' = P + \begin{bmatrix} T_x \\ T_y \end{bmatrix}$$

2D rotation matrix

- Rotate θ counterclockwise
  - What is the transformation *R*?

$$P' = RP$$



$$P' = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} P$$

2D Transformation

- For animation, manipulation, user interaction
- translation, rotation, scaling

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} T_x \\ T_y \end{bmatrix}$$

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

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

2D Transformation *(cont.)*

- Inconsistent representation for translation
- Cannot be concatenated
- Troublesome for
  - Hierarchical transforms
  - Interactive, incremental display

$$\mathbf{P} = \mathbf{R}_n \cdots (\mathbf{R}_3(\mathbf{R}_2(\mathbf{R}_1\mathbf{P} + \mathbf{T}_1) + \mathbf{T}_2) + \mathbf{T}_3) \cdots + \mathbf{T}_n$$

Homogeneous Coordinates

- consistent representation for all three
- can be concatenated & pre-computed

$$(x, y) \rightarrow (wx, wy, w), w \neq 0$$

$$(wx, wy, w) \rightarrow (wx / w, wy / w)$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

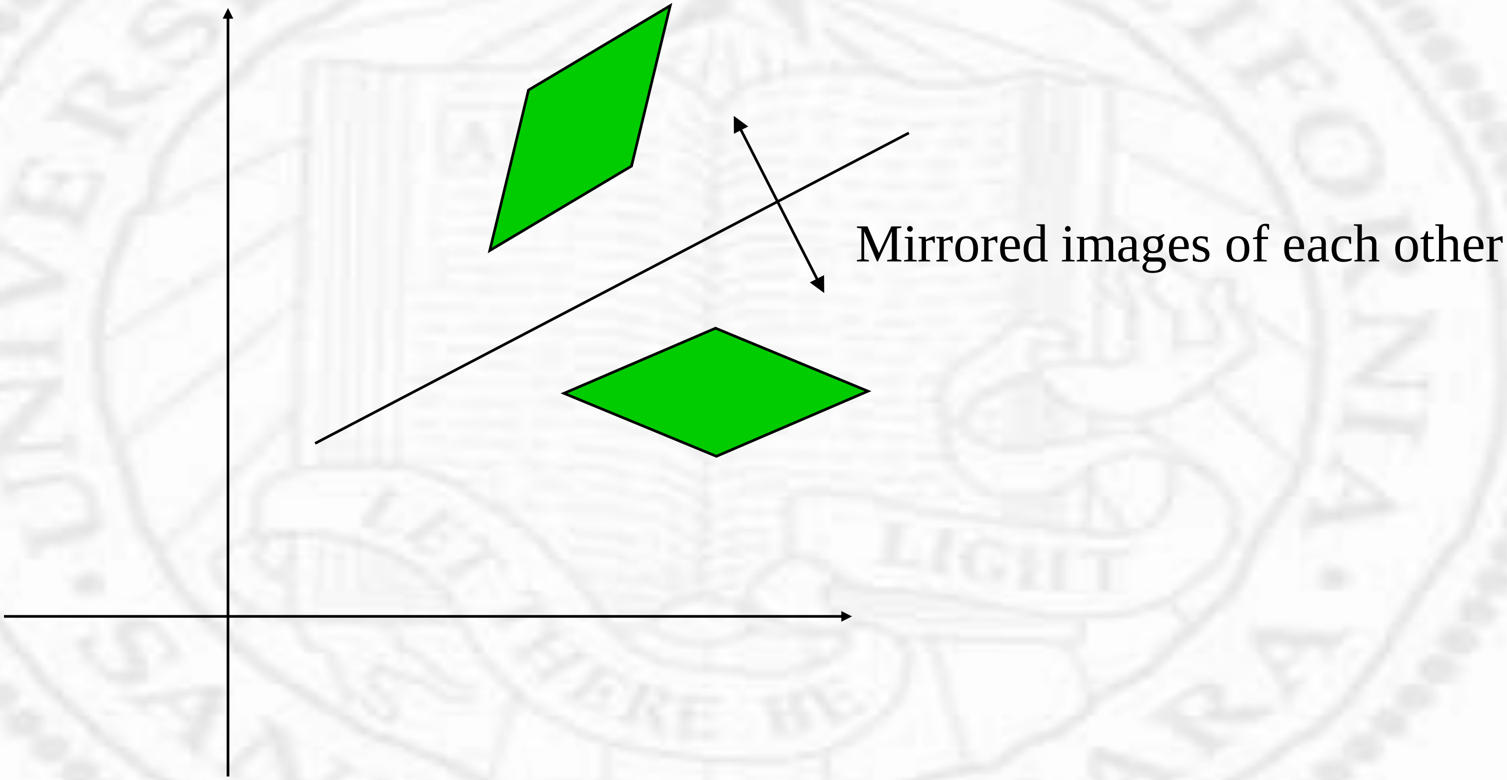
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

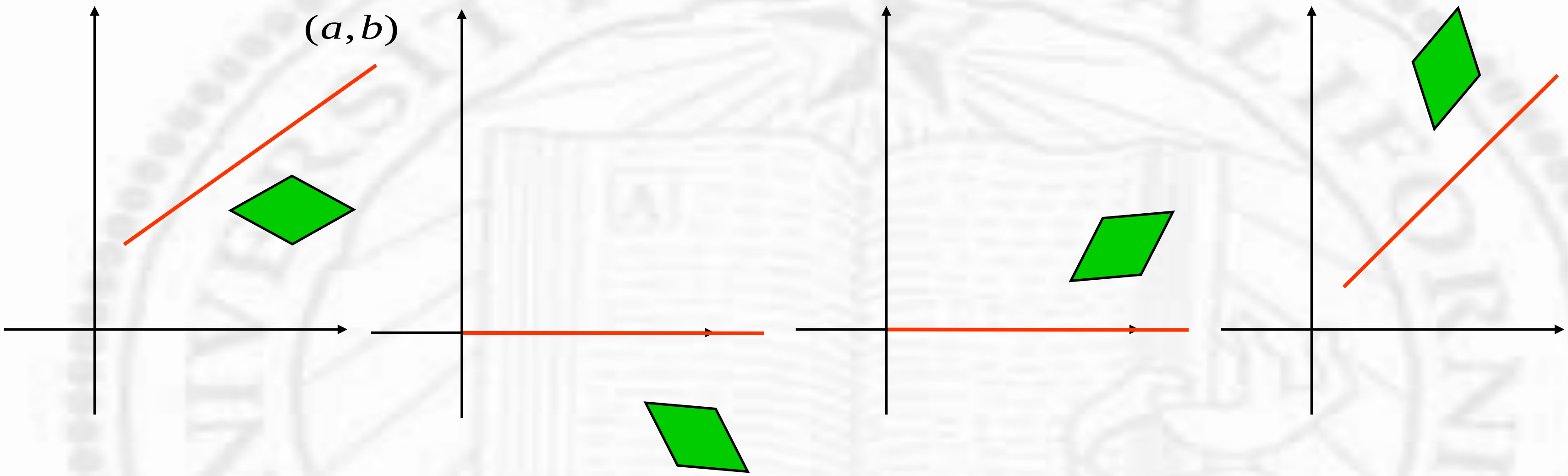
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = (TRS) \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

How about other transforms?

❖ For example, reflection



- Try to represent the new transform as a composite of T, R, S



$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\theta = -\tan^{-1}\left(\frac{b}{a}\right) \qquad \theta = \tan^{-1}\left(\frac{b}{a}\right)$$

Clipping Against Upright Rectangular Window

❖ Points

$$\text{if} \quad x_{\min} \le x \le x_{\max} \ \& \ y_{\min} \le y \le y_{\max}$$

then accept otherwise reject

Clipping Against Upright Rectangular Window

- Lines
  - trivially accepted if both end points inside
  - otherwise Points

$$x_1 + t(x_2 - x_1) = x'_1 + t'(x'_2 - x'_1)$$

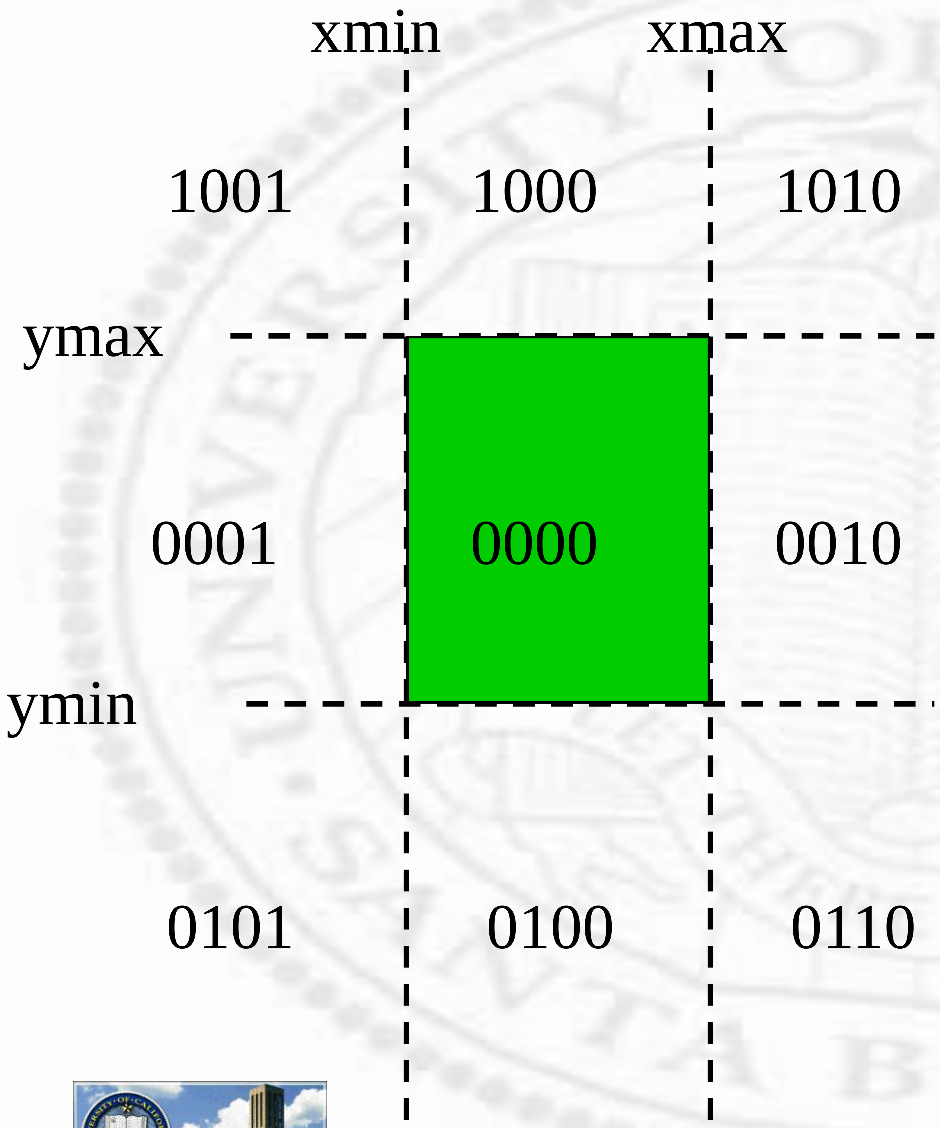
$$y_1 + t(y_2 - y_1) = y'_1 + t'(y'_2 - y'_1)$$

$$0 \le t, t' \le 1$$

$(x_1, y_1), (x_2, y_2)$: end points of line

$(x'_1, y'_1), (x'_2, y'_2)$: end points of window boundary

Cohen-Sutherland Line-Clipping Algorithm



- Outcodes

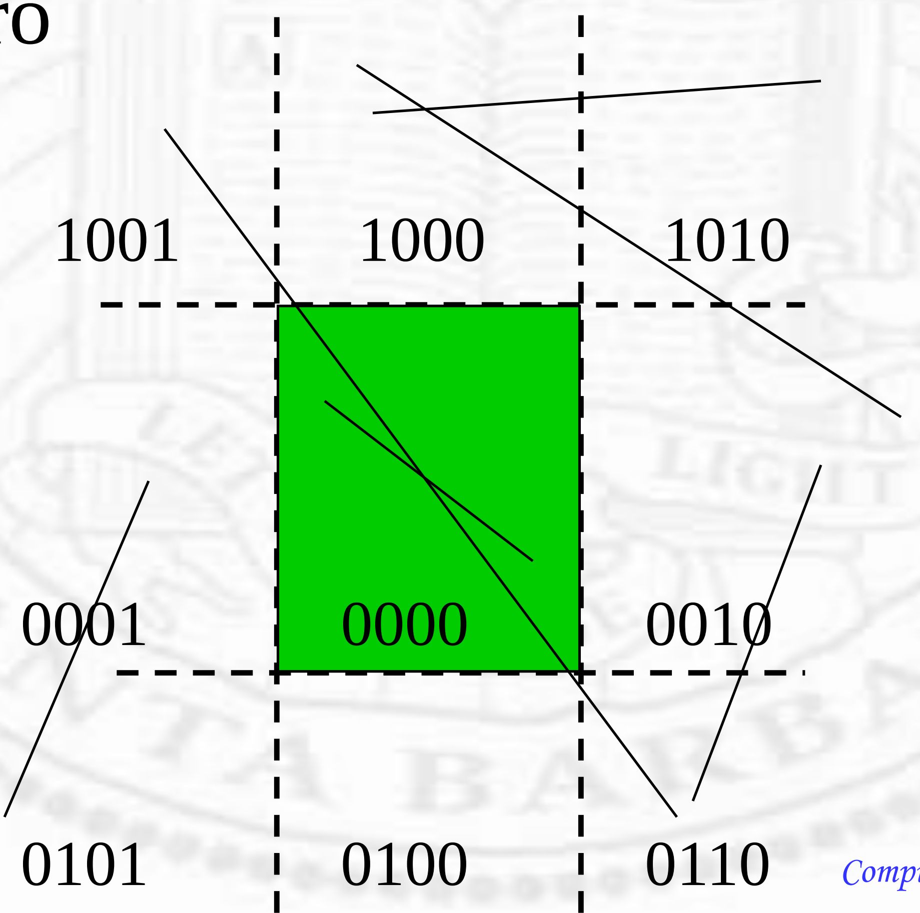
bit1 --above: ymax-y

bit2 --below: y-ymin

bit3 --right of: xmax-x

bit4 --left of: x-xmin

- Trivially-accept: both end points having outcode 0000
- Trivially-reject: corresponding bits in two outcodes are set, or outcode1 & outcode2 nonzero



- Neither: need more testing
- E.g. mid-point algorithm
  - divide a line segment $(x_1, y_1),(x_2, y_2)$ into two line segments

$$(x_1, y_1),((x_1 + x_2)/2,(y_1 + y_2)/2)$$

$$((x_1 + x_2)/2,(y_1 + y_2)/2),(x_2, y_2)$$

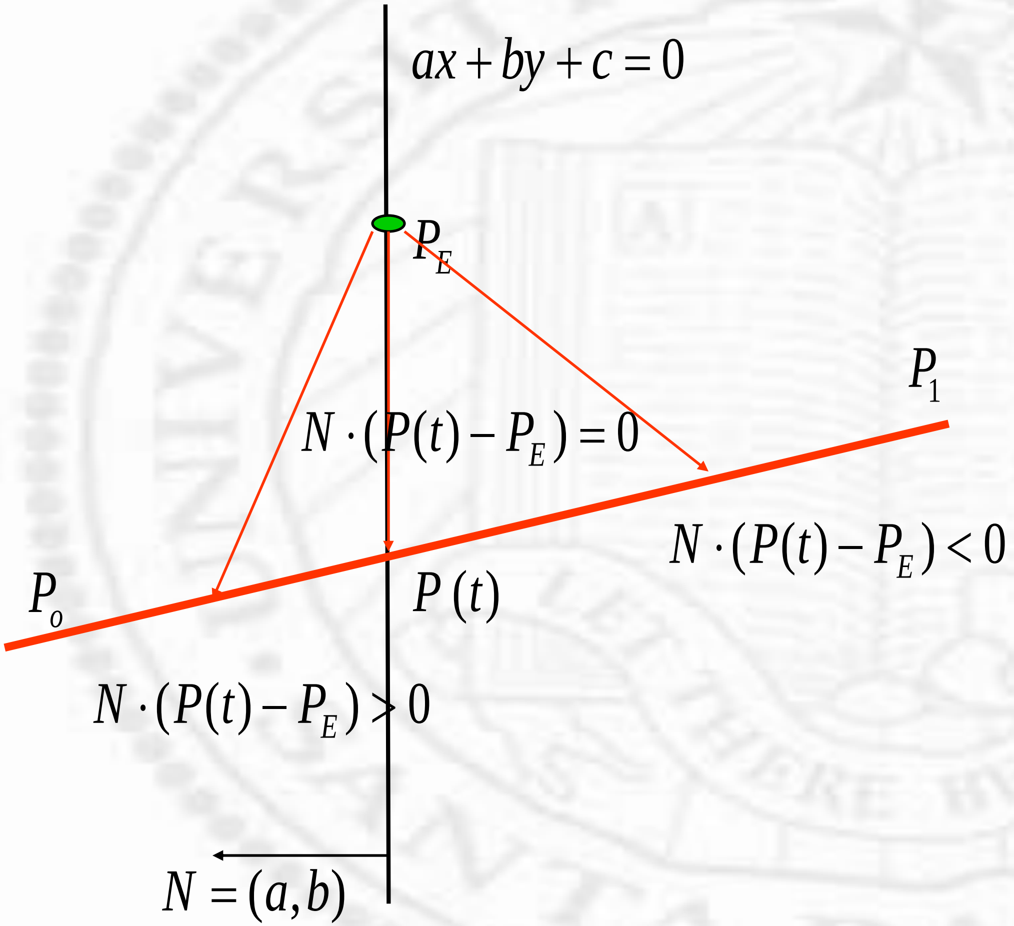
  - test each line independently
  - recursive division if necessary
  - guarantee to stop in O(logn) steps

Cyrus-Beck (Liang-Basky) Line Clipping

- Can be more efficient when intersection tests are unavoidable
- Work in the parameter (*t*) space to locate true intersections before calculating 2D coordinates
- Work for all kinds of clipping polygons and in 3D
- Two basic steps:
  - find intersections (t)
  - classify intersections

Cyrus-Beck (Liang-Basky) Line Clipping

- Find intersections

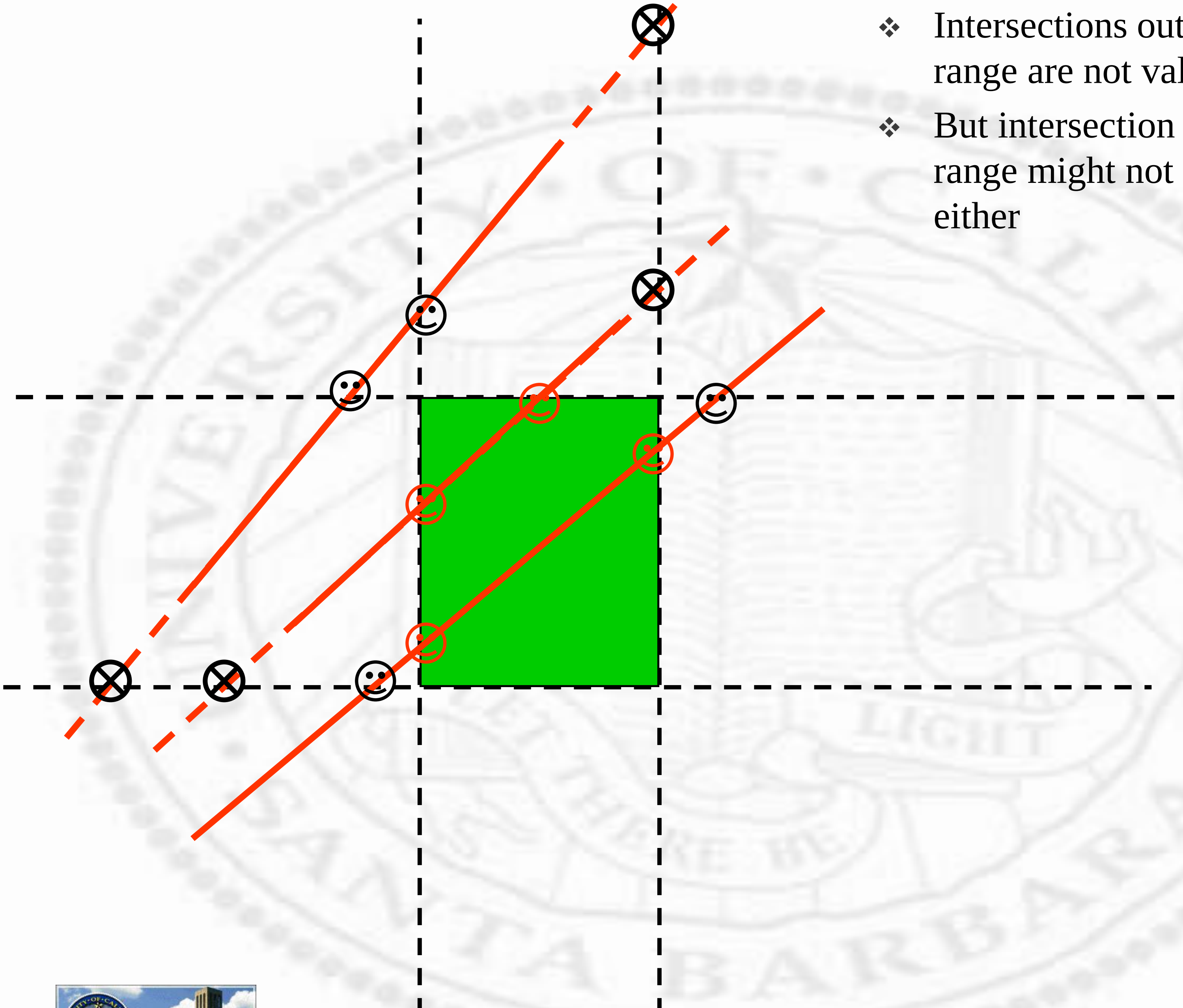


$$N \cdot (P(t) - P_E) = 0$$

$$N \cdot (P_o + t(P_1 - P_o) - P_E) = 0$$

$$t\,N \cdot (P_1 - P_o) + N \cdot (P_o - P_E) = 0$$

$$t = \frac{N \cdot (P_o - P_E)}{-N \cdot (P_1 - P_o)}$$



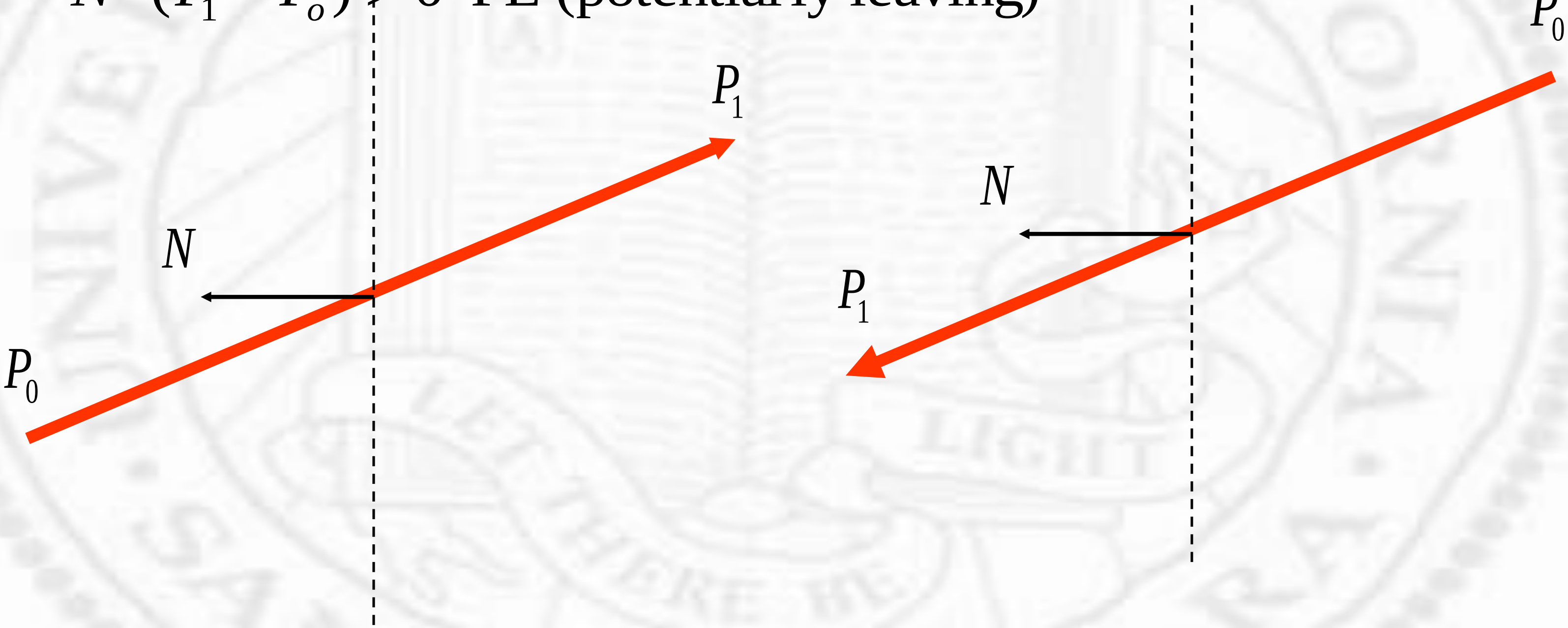
- Intersections outside (0,1) range are not valid ⊗
- But intersection inside (0,1) range might not be valid either ☺

Cyrus-Beck (Liang-Basky) Line Clipping

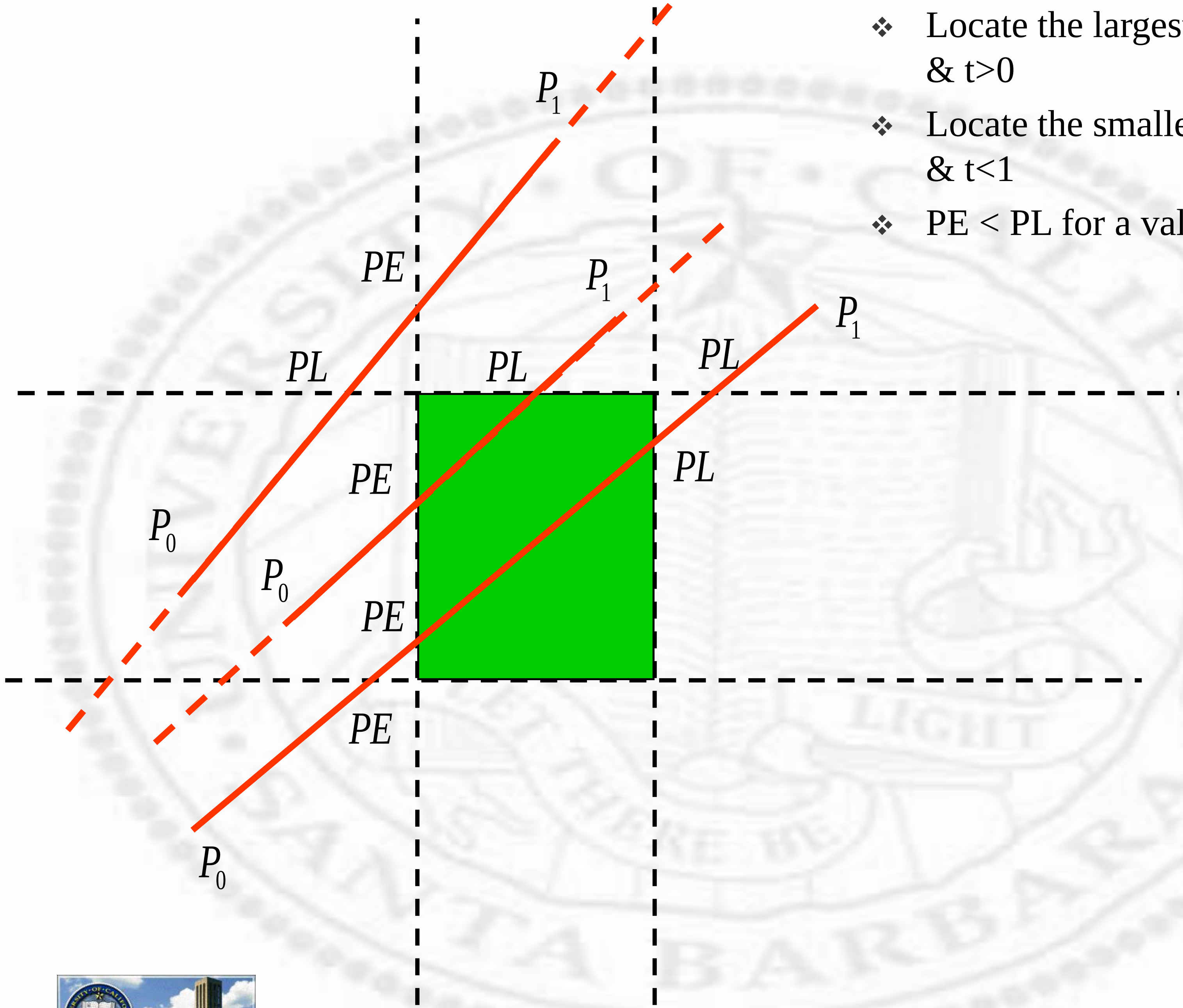
- Classify intersections

$N \cdot (P_1 - P_o) < 0$ PE (potentially entering)

$N \cdot (P_1 - P_o) > 0$ PL (potentially leaving)

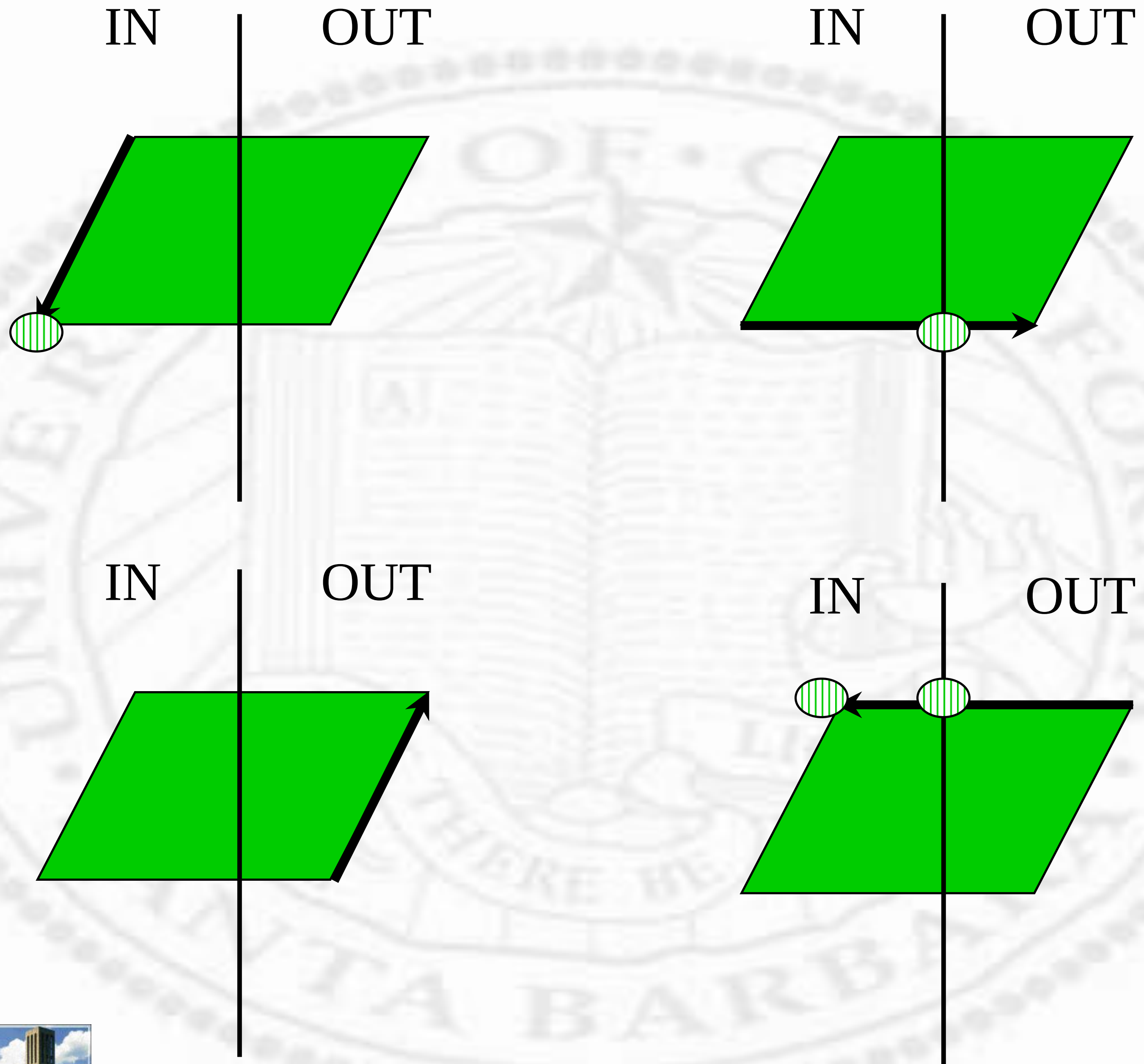


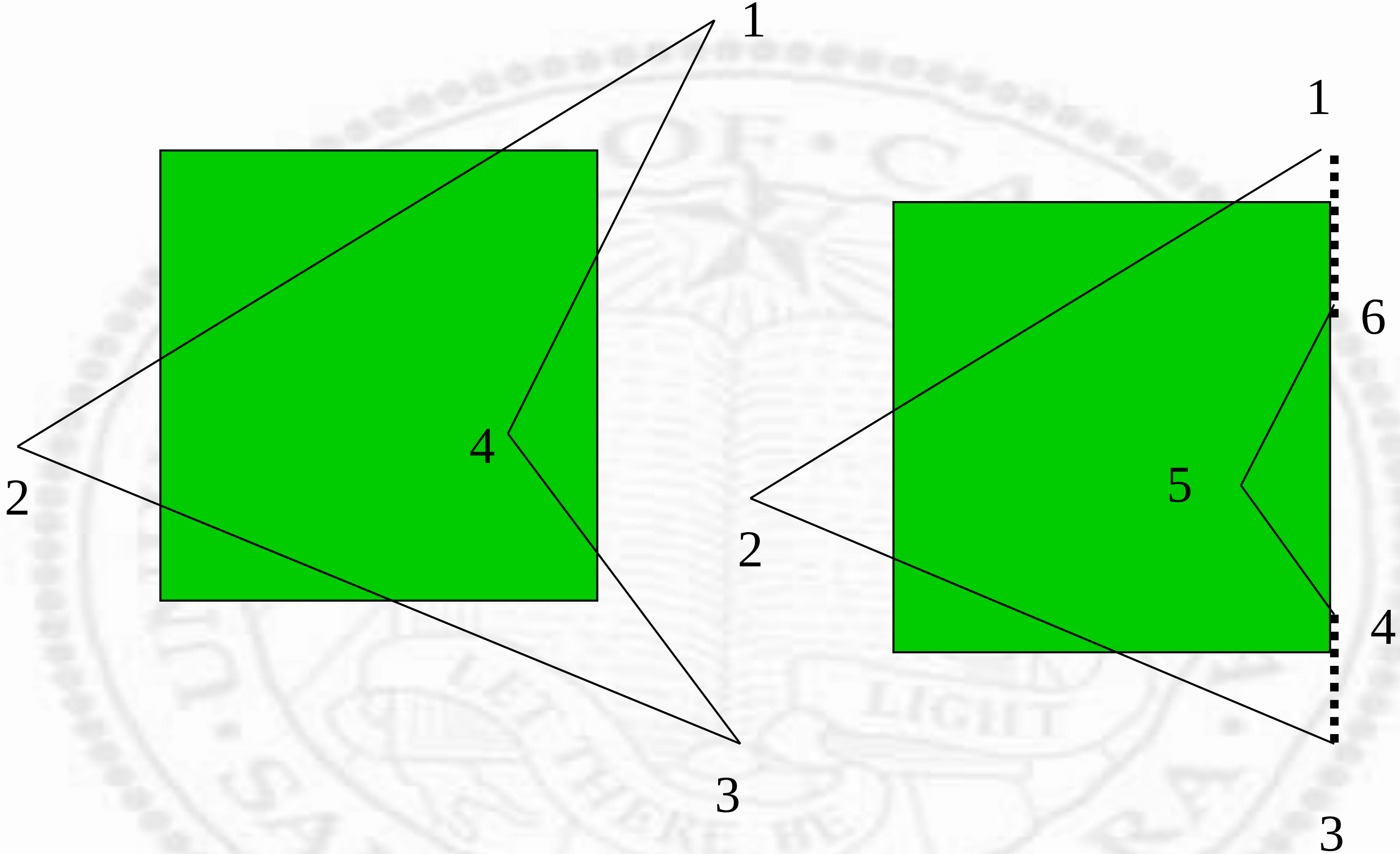
- Locate the largest PE point & t>0
- Locate the smallest PL point & t<1
- PE < PL for a valid line



Polygon Clipping (Sutherland-Hodgman)

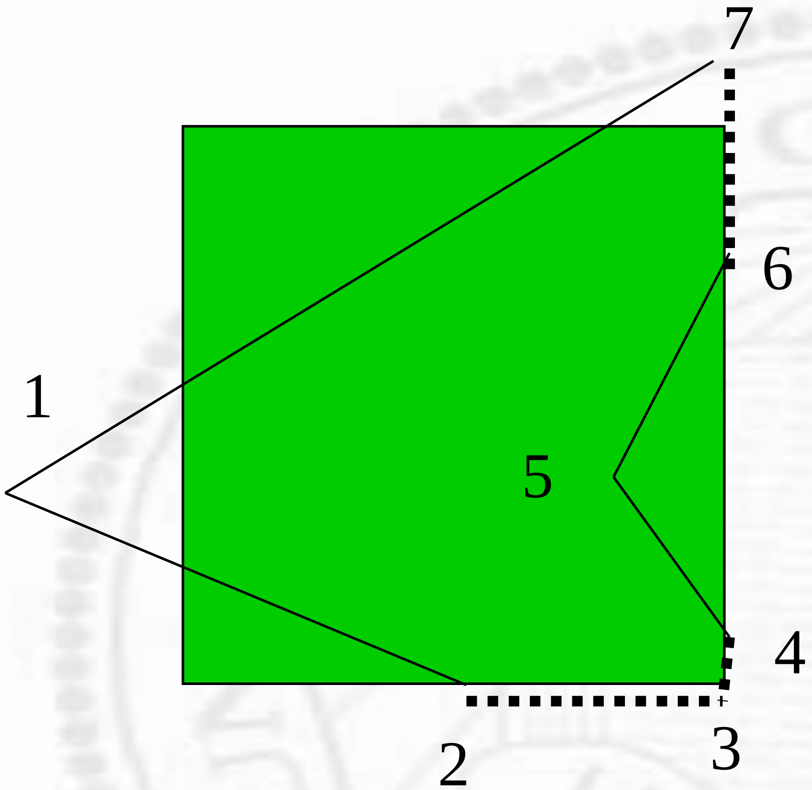
- Given an ordered sequence of polygon vertices
- And a *convex* clipping polygon
- Output ordered clipped polygon vertices
- Using divide-and-conquer, one clipping edge at a time


IN
OUT
IN
OUT
IN
OUT
IN
OUT

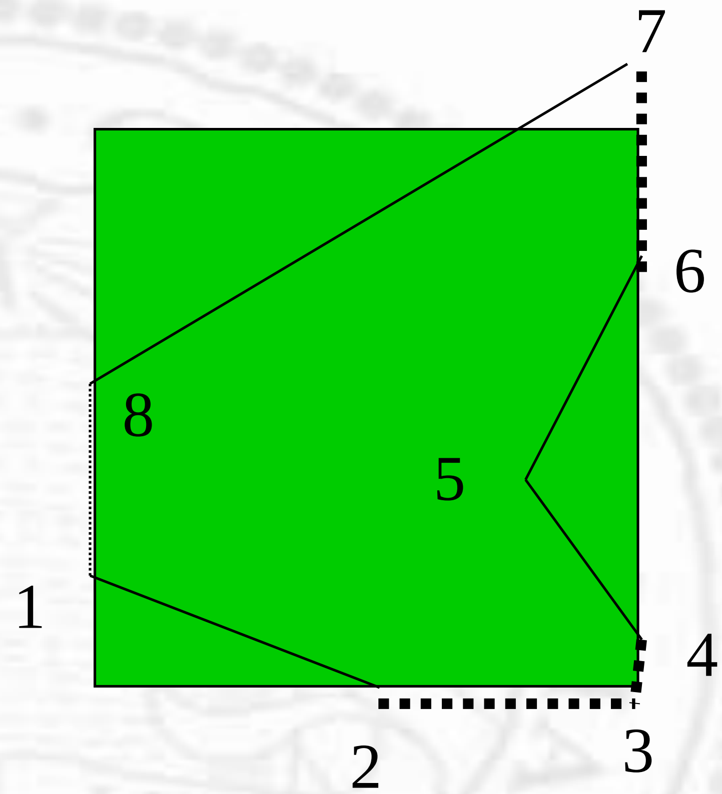


Original

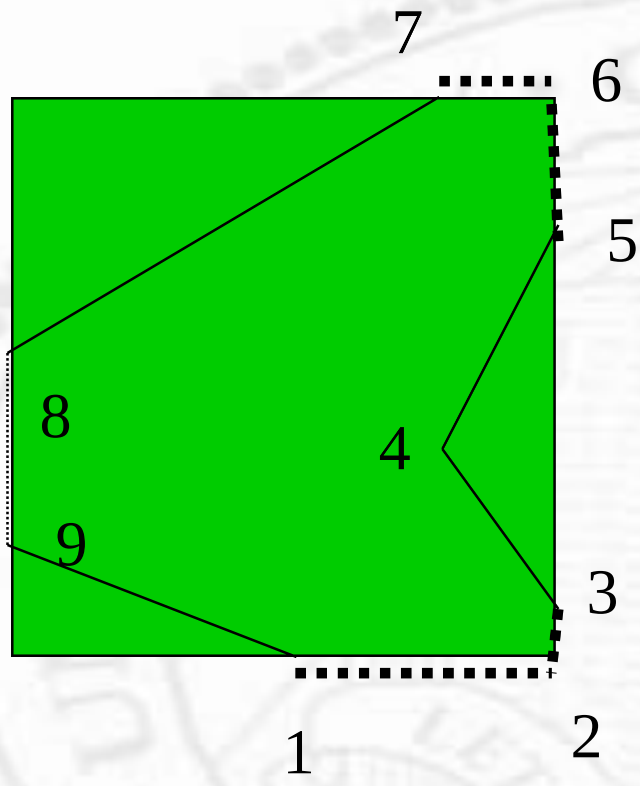
Right boundary clipping



Bottom boundary clipping



Left boundary clipping



Top boundary clipping

Other Primitives

- Use of extents (extents for a whole string, words, individual characters)
- Divide and Conquer