实验4: 几何变换&裁剪算法
Transformation & Clipping Algorithm

华东师范大学计算机科学与技术学院

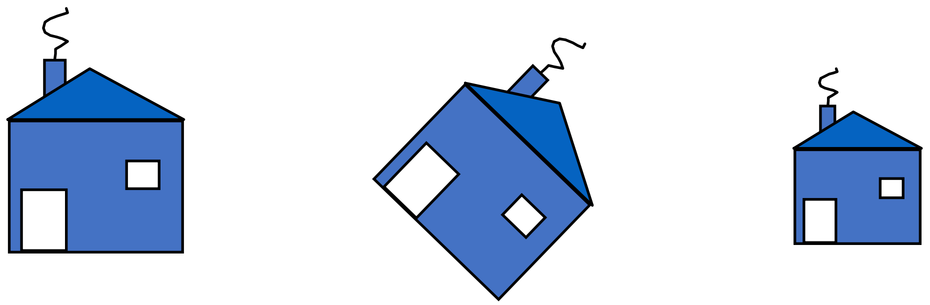
李晨 副研究员

cli@cs.ecnu.edu.cn

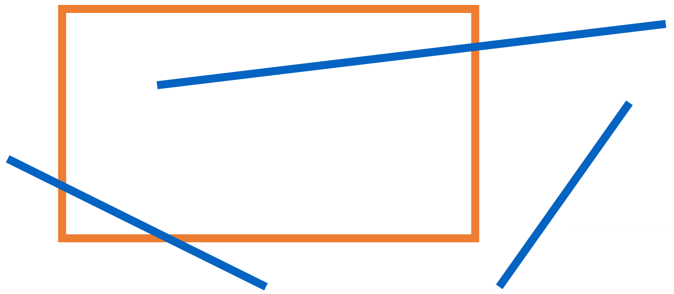
华东师范大学计算机科学与技术学院
School of Computer Science and Technology

Contents

- 2D Transformation



- Polygon Clipping



Matrix Representation

- Represent 2D transformation with matrix

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \qquad \begin{aligned} x' &= ax + by \\ y' &= cx + dy \end{aligned}$$

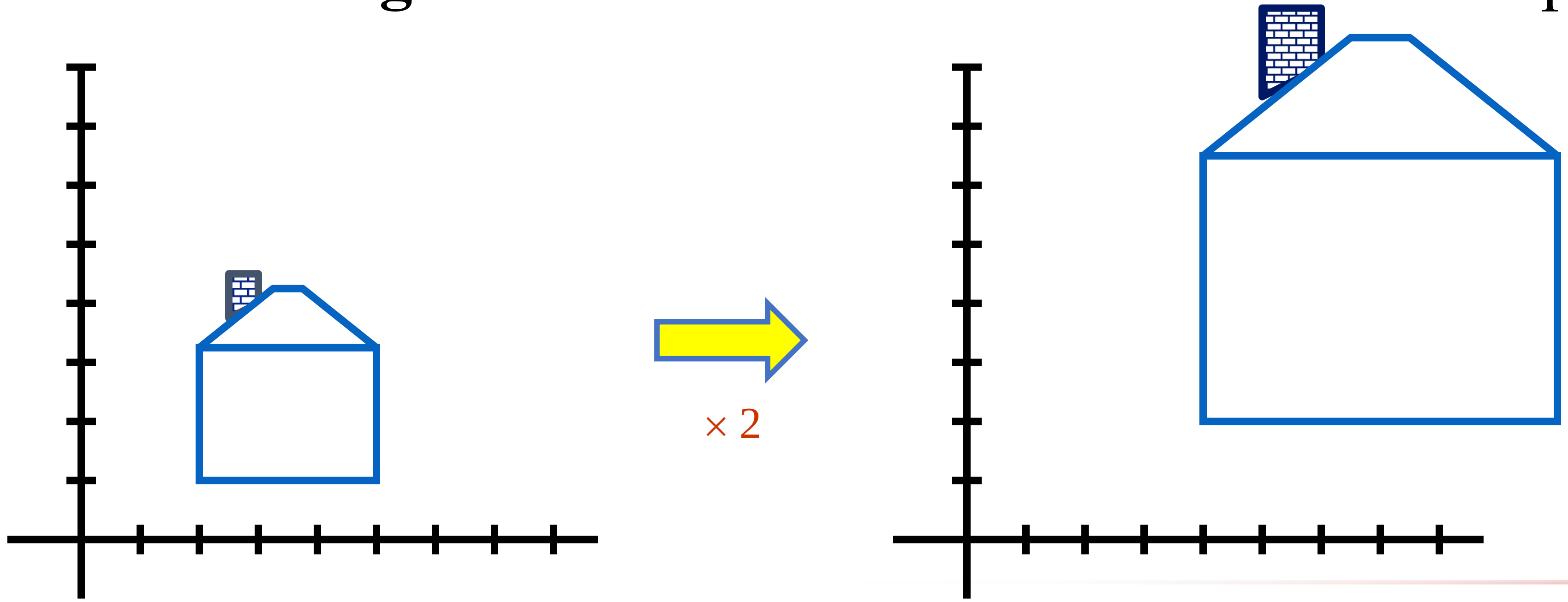
- Transformations combined by multiplication

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} d & e \\ d & f \end{bmatrix} \begin{bmatrix} h & i \\ g & j \end{bmatrix} \begin{bmatrix} i & x \\ k & y \end{bmatrix}$$

  - Matrices are efficient, convenient way to represent sequence of transformations!

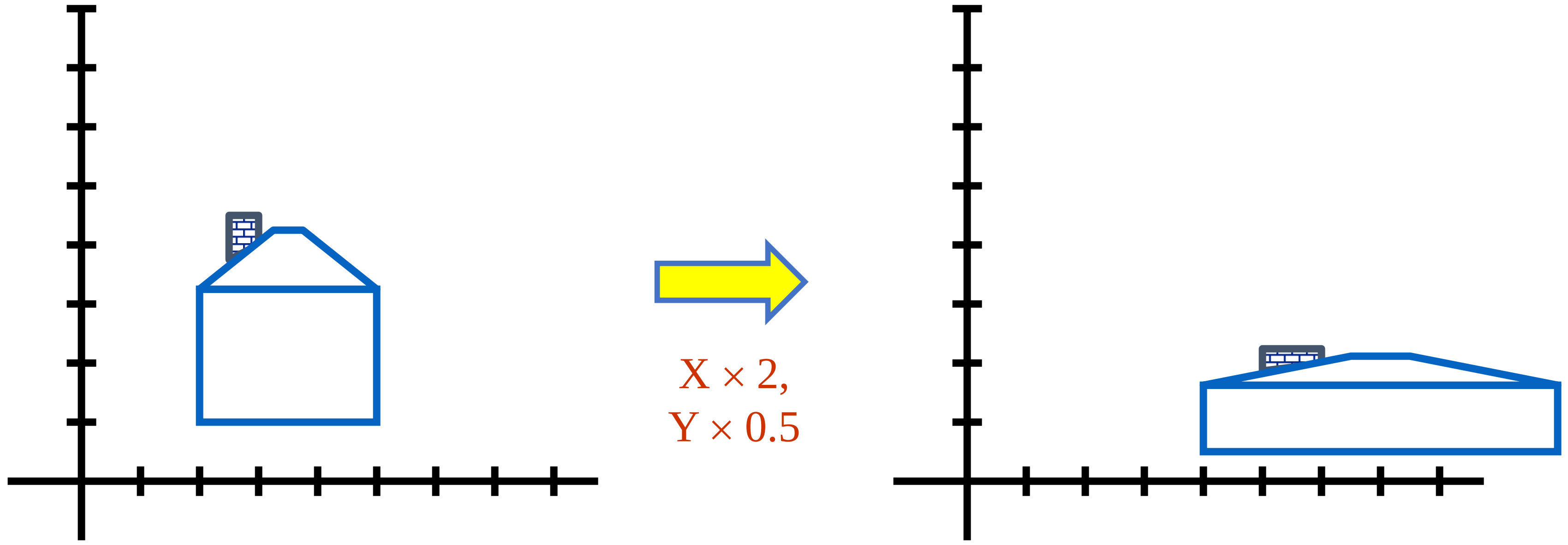
Scaling

- Scaling a coordinate means multiplying each of its components by a scalar
- **Uniform scaling** means this scalar is the same for all components:



Scaling

- **Non-uniform scaling**: different scalars per component:

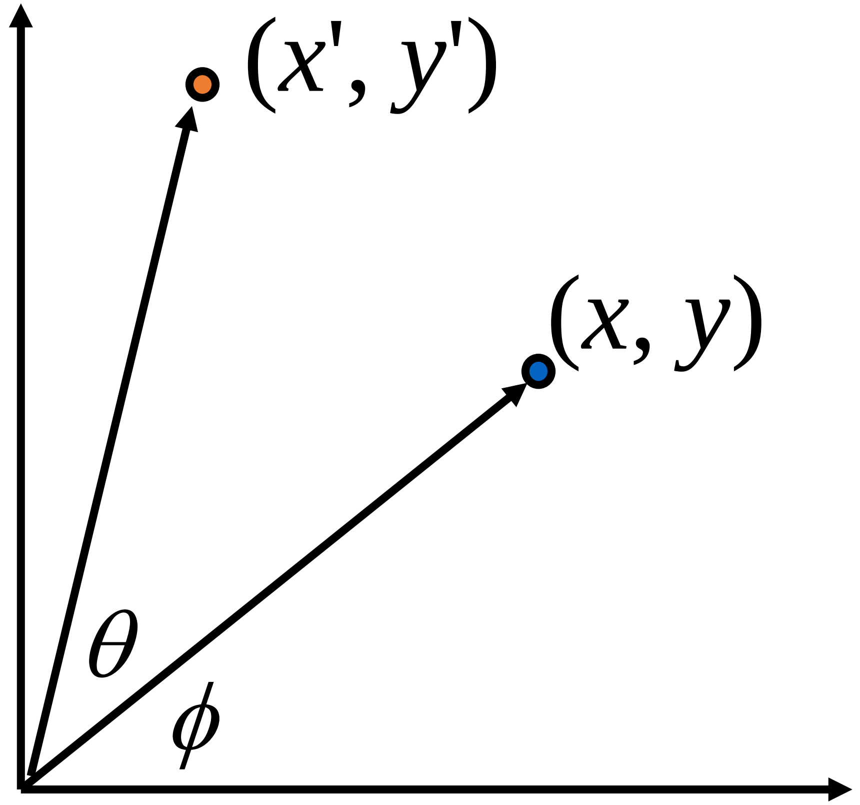


Scaling

- Scaling operation: $\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} ax \\ by \end{bmatrix}$

- or, in matrix form: $$\begin{bmatrix} x' \\ y' \end{bmatrix} = \underbrace{\begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix}}_{\textit{scaling matrix}} \begin{bmatrix} x \\ y \end{bmatrix}$$

Rotation



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

Trig Identity…

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

Substitute…

$x' = x\, \mathbf{cos}(\theta) - y\, \mathbf{sin}(\theta)$
$y' = x\, \mathbf{sin}(\theta) + y\, \mathbf{cos}(\theta)$

Rotation

- Easy to capture in matrix form:

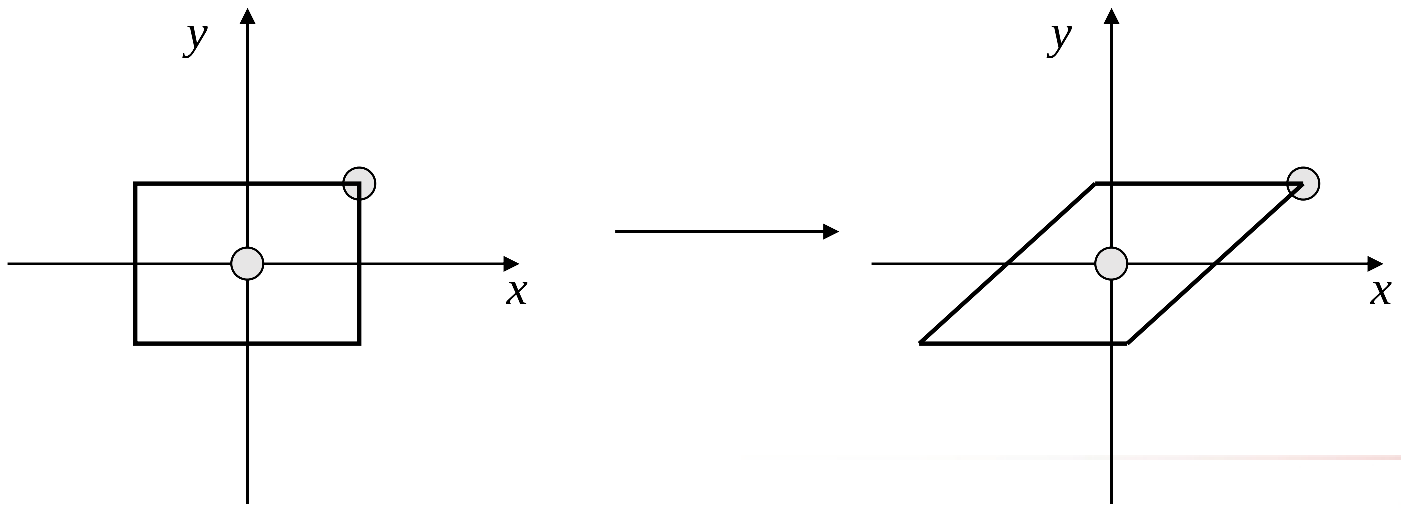
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos(q) & -\sin(q) \\ \sin(q) & \cos(q) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

- Even though sin() and cos() are nonlinear functions
  - x' is a linear combination of x and y
  - y' is a linear combination of x and y

Shear

- Shear along *x*-axis
  - push points to right in proportion to height

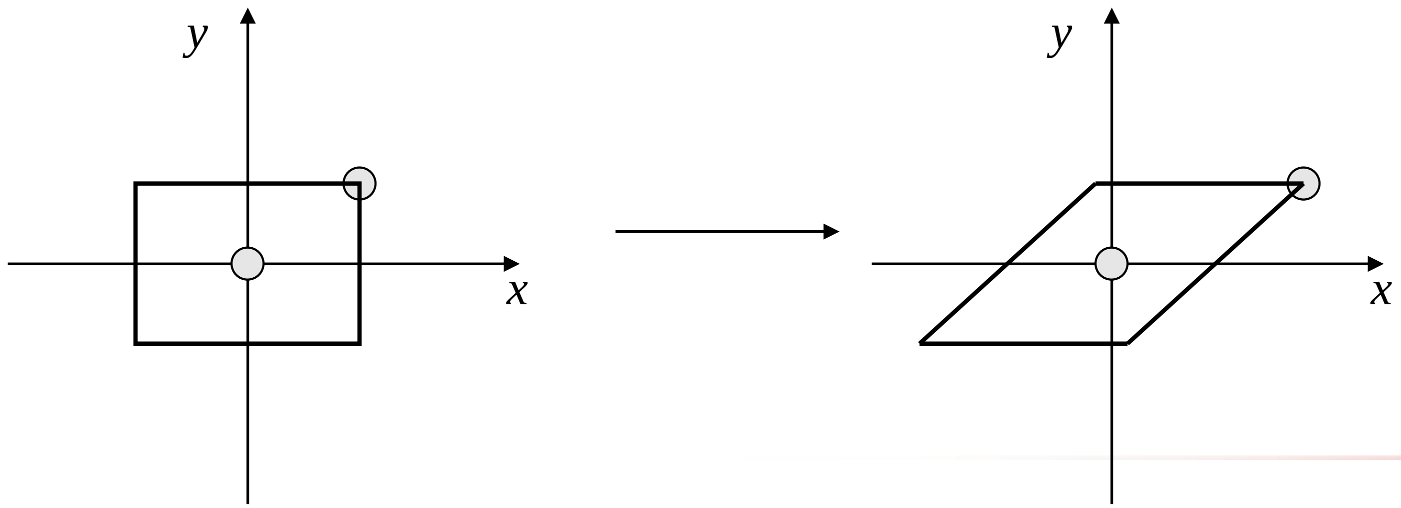
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} ? & ? \\ ? & ? \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} ? \\ ? \end{bmatrix}$$



Shear

- Shear along *x*-axis
  - push points to right in proportion to height

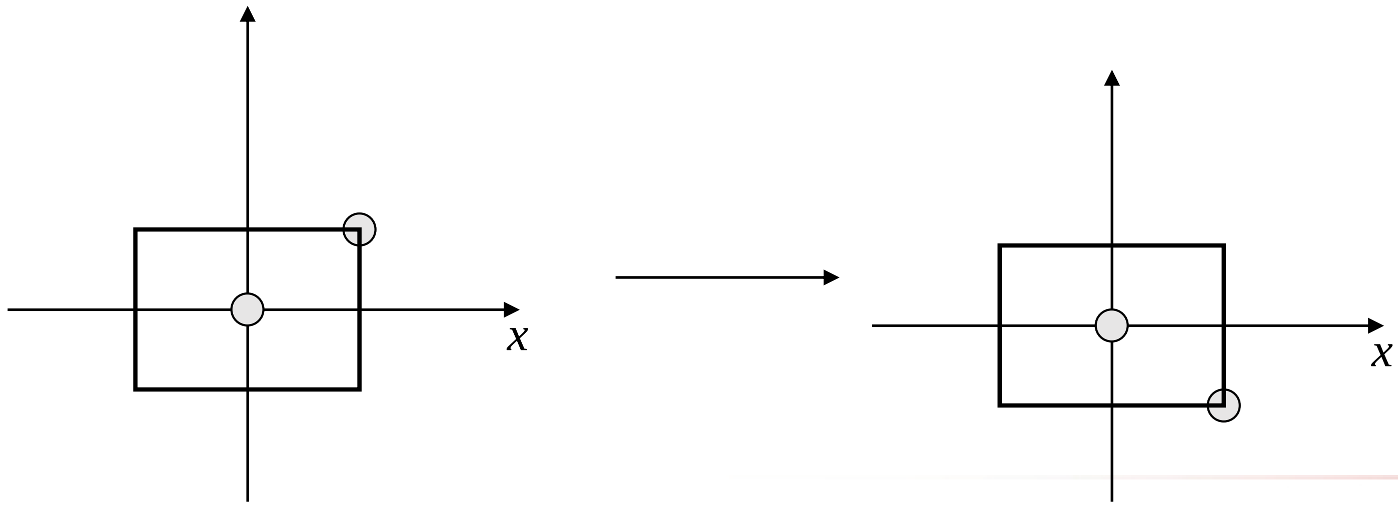
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & sh_x \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$



Reflection

- Reflect across *x*-axis

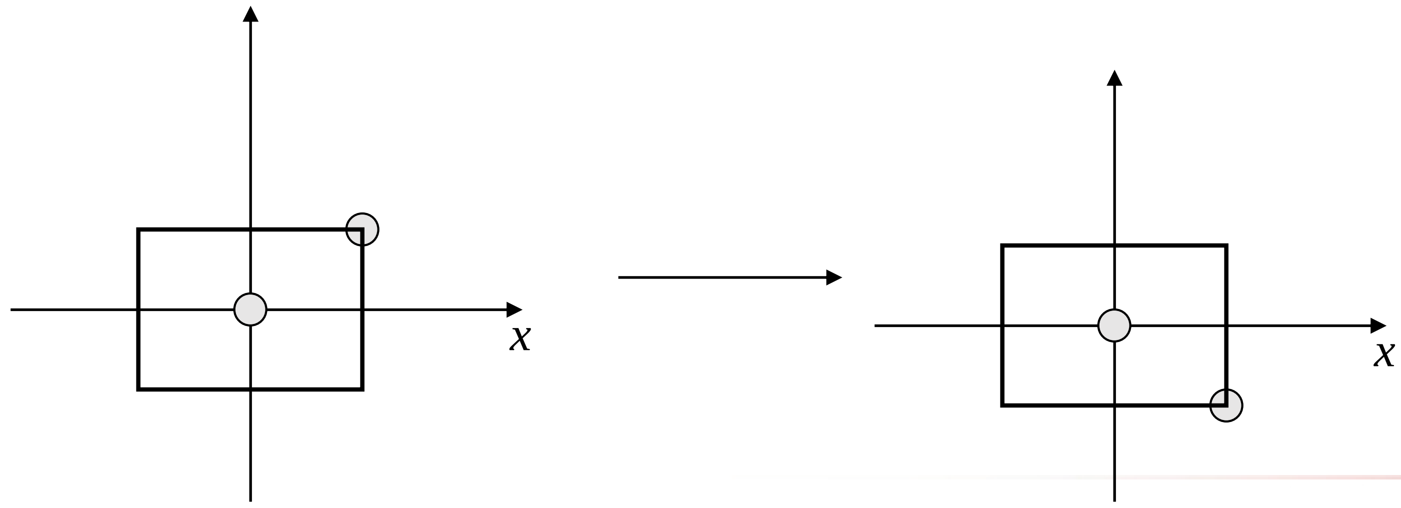
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} ? & ? \\ ? & ? \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} ? \\ ? \end{bmatrix}$$



Reflection

- Reflect across *x*-axis

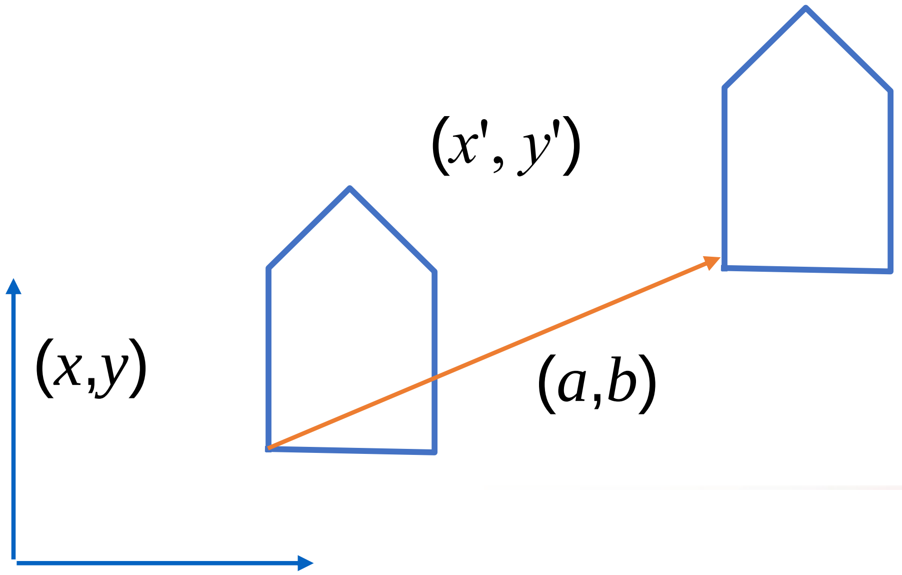
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$



Translation

- Translate by (a,b)

$$\begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} x+a \\ y+b \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix}$$



Linear & Affine Transformations

- **Linear transformations** are combinations of
  - shear
  - scale
  - rotate
  - reflect
- Properties of linear transformations
  - satisfies $T(sx+ty) = sT(x) + tT(y)$
  - origin maps to origin
  - lines map to lines
  - parallel lines remain parallel

Linear & Affine Transformations

- **Affine transformations** are combinations of
  - linear transformations
  - translations
- Properties of affine transformations
  - **origin does not necessarily map to origin**
  - lines map to lines
  - parallel lines remain parallel

Challenge

- Matrix multiplication
  - for everything except translation
  - how to do everything with multiplication?
    - then just do composition, no special cases
- Homogeneous coordinates trick
  - represent 2D coordinates (x,y) with 3D vector (x,y,1)

matrix multiplication

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \underbrace{\begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix}}_{\textit{scaling matrix}} \begin{bmatrix} x \\ y \end{bmatrix}$$

matrix multiplication

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \underbrace{\begin{bmatrix} \cos(q) & -\sin(q) \\ \sin(q) & \cos(q) \end{bmatrix}}_{\textit{rotation matrix}} \begin{bmatrix} x \\ y \end{bmatrix}$$

vector addition

$$\begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} x + a \\ y + b \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix}$$

Homogeneous Coordinates

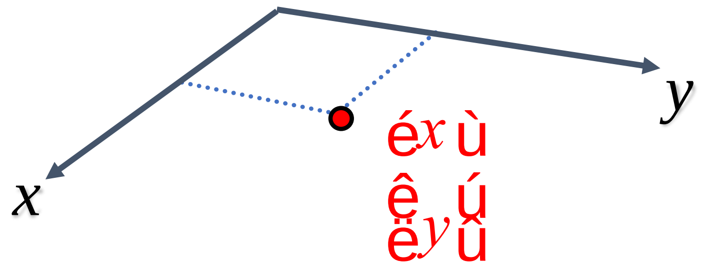
- Our 2D transformation matrices are now 3x3:

$$\mathbf{R}otation = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{S}cale = \begin{bmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{T}ranslation = \begin{bmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \\ 0 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} 1 & 0 & a \\ 0 & 1 & b \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x*1+a*1 \\ y*1+b*1 \\ 1 \end{bmatrix} = \begin{bmatrix} x+a \\ y+b \\ 1 \end{bmatrix}$$

Homogeneous Coordinates Geometrically

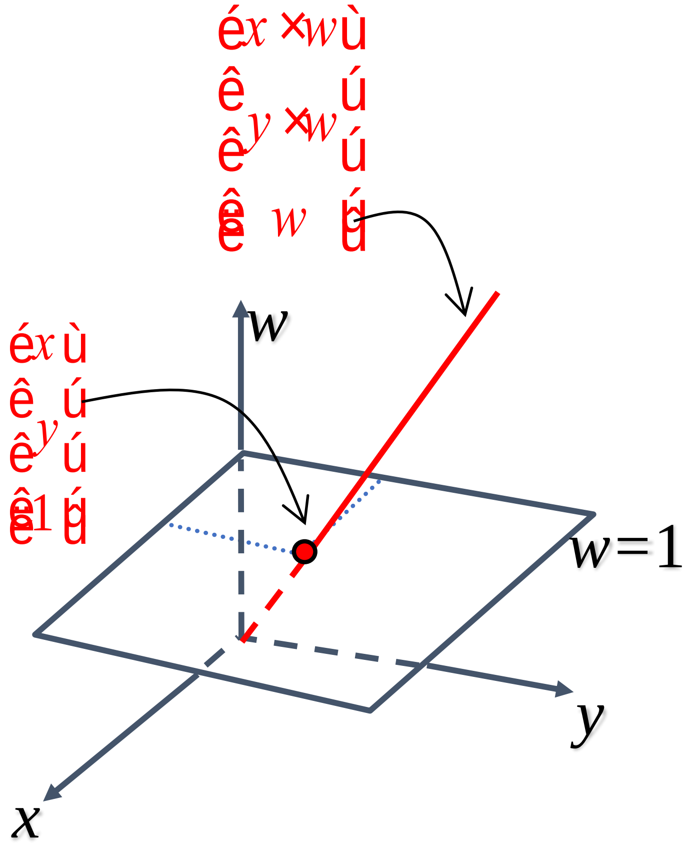
- point in 2D cartesian



Homogeneous Coordinates Geometrically

homogeneous **cartesian**

$$(x, y, w) \xrightarrow{/\,w} \left(\frac{x}{w}, \frac{y}{w}\right)$$

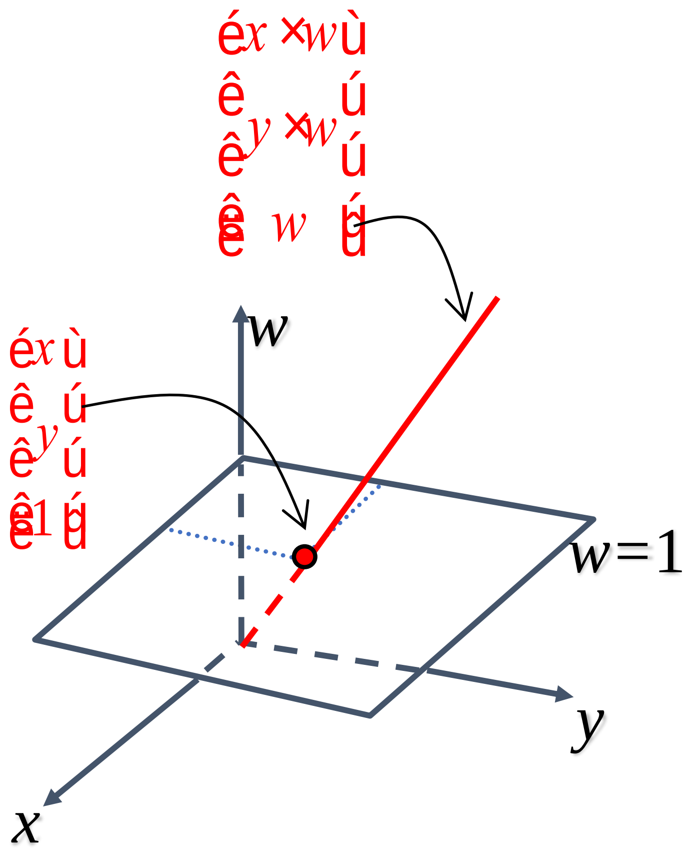


$\begin{bmatrix} x \cdot w \\ y \cdot w \\ w \end{bmatrix}$
$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$
w
$w=1$
y
x

- point in 2D cartesian + weight *w* = point *P* in 3D homogeneous coordinates
- multiples of (*x*,*y*,*w*)
  - form a line *L* in 3D
  - all homogeneous points on *L* represent same 2D cartesian point
  - example: (2,2,1) = (4,4,2) = (1,1,0.5)

Homogeneous Coordinates Geometrically

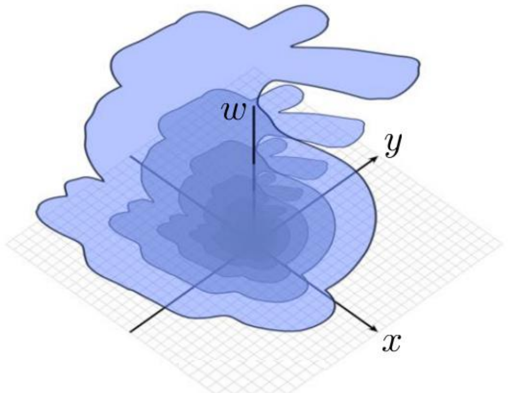
$$\begin{array}{ccc} \textbf{homogeneous} & & \textbf{cartesian} \\ (x, y, w) & \xrightarrow{/\ w} & \left(\frac{x}{w}, \frac{y}{w}\right) \end{array}$$



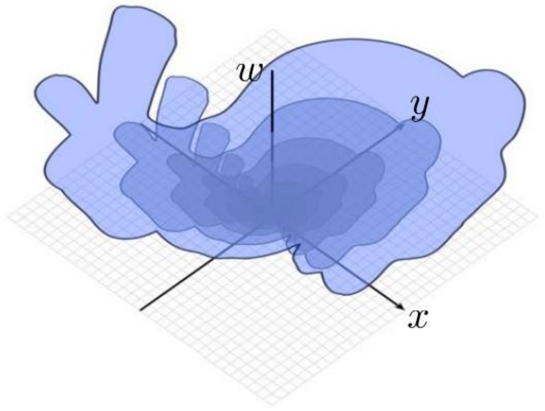
$\begin{bmatrix} x \cdot w \\ y \cdot w \\ w \end{bmatrix}$
$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$
w
$w=1$
y
x

- homogenize to convert homogeneous 3D point to cartesian 2D point:
  - divide by *w* to get (*x*/*w*, *y*/*w*, 1)
  - projects line to point onto *w*=1 plane
- when *w*=0, consider it as direction
  - points at infinity
  - these points cannot be homogenized
  - lies on *x*-*y* plane
- (0,0,0) is undefined

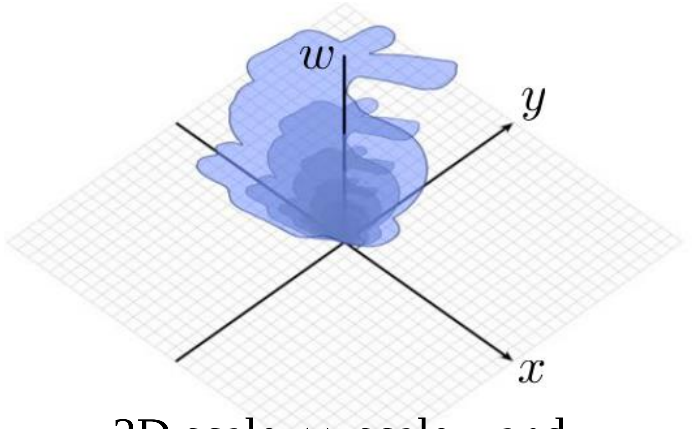
Visualizing 2D transformations



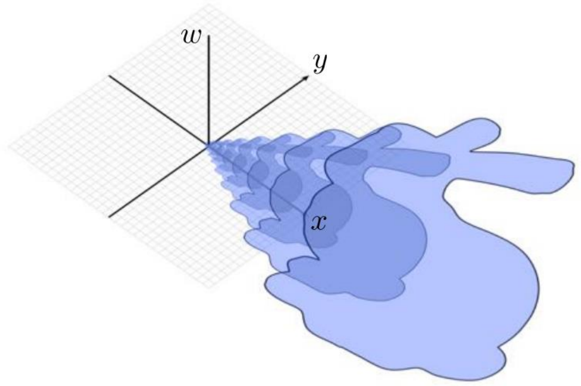
Original shape in 2D can be viewed as many copies, uniformly scaled by *w*



2D rotation ↔ rotate around *w*



2D scale ↔ scale *x* and *y* preserve *w*



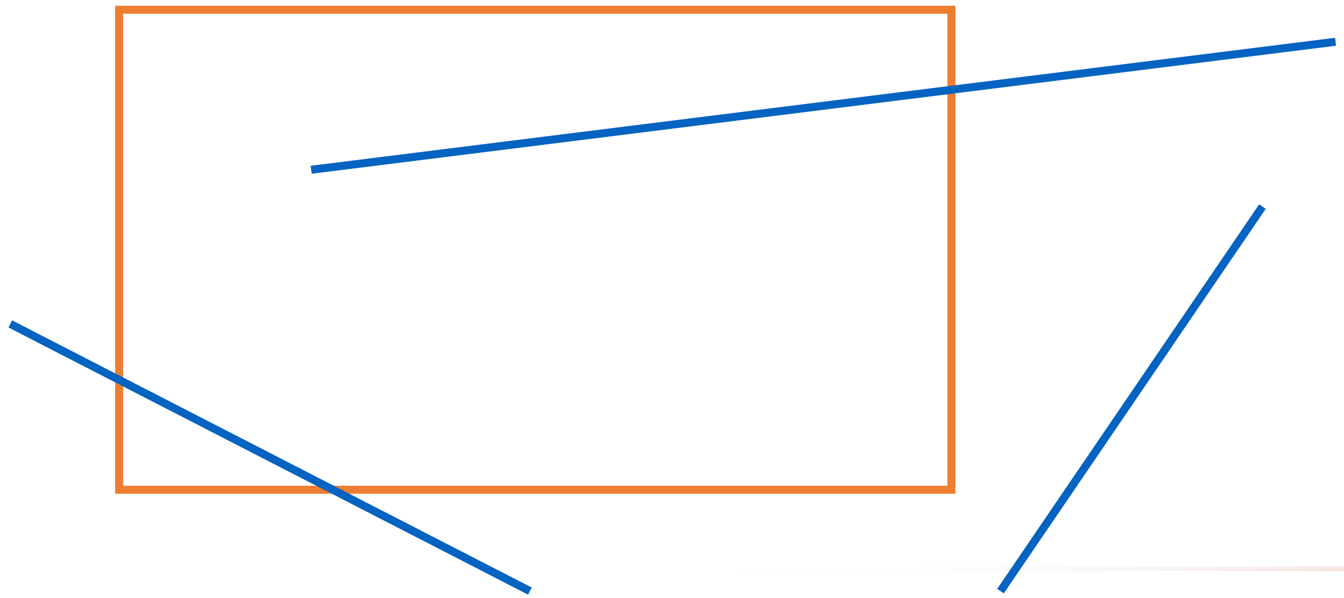
2D translate ↔ shear in 2D-H

Transformations Summary

- Transformations can be interpreted as operations that move points in space
  - e.g., for modeling, animation
- Or as a change of coordinate system
  - e.g., screen and view transforms
- Construct complex transformations as compositions of basic transforms
- Homogeneous coordinate representation allows for expression of non-linear transforms as matrix operations (linear transforms) in higher-dimensional space
  - Matrix representation affords simple implementation and efficient composition

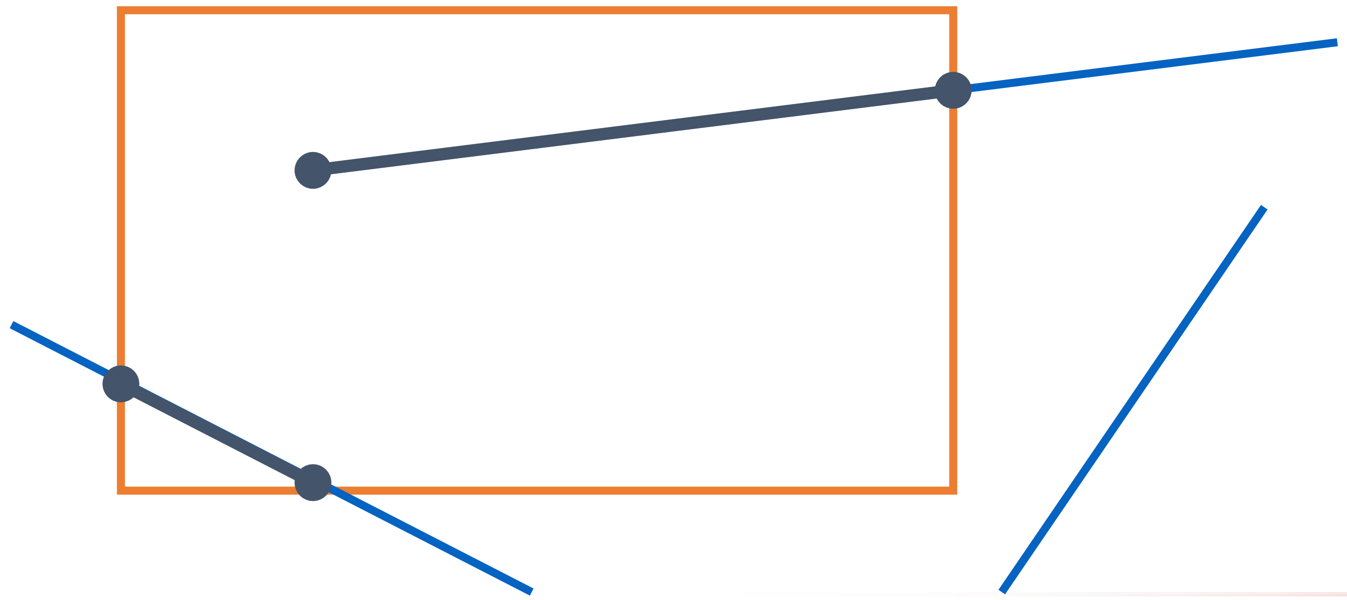
Next Topic: Clipping

- We’ve been assuming that all primitives (lines, triangles, polygons) lie entirely within the viewport
- In general, this assumption will not hold:



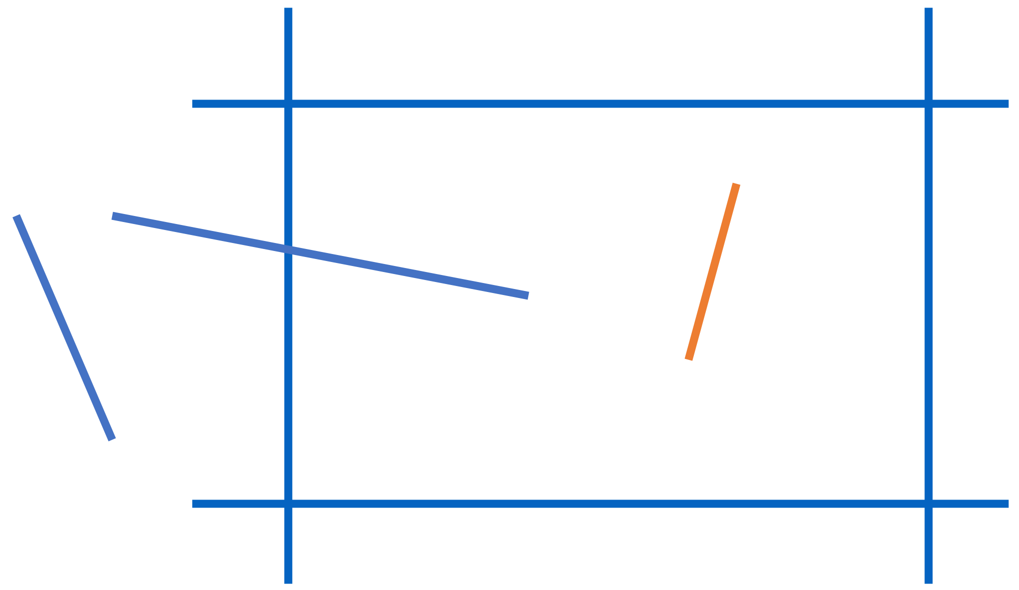
Why Clip?

- Bad idea to rasterize outside of framebuffer bounds
- Also, don’t waste time scan converting pixels outside window



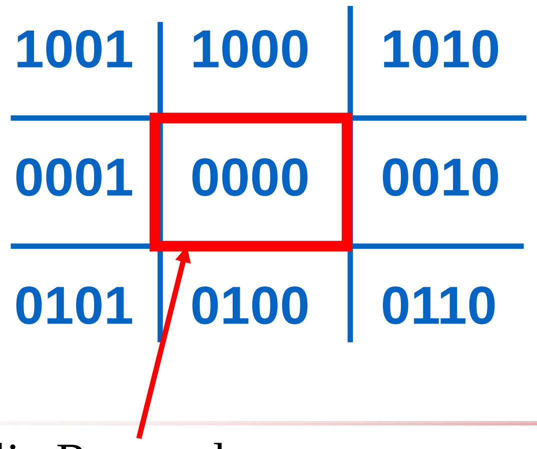
Line Clipping

- Trivially accept lines with both endpoints inside all edges of the viewport
- Trivially reject lines with both endpoints outside the same edge of the viewport
- Otherwise, reduce to trivial cases by splitting into two segments



Cohen-Sutherland Line Clipping

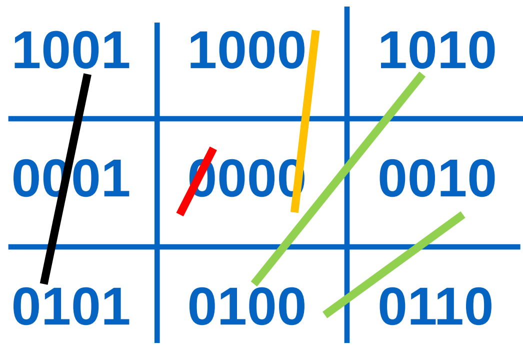
- Extend the edges of the clip rectangle to divide the plane of the clip rectangle into nine regions
- Each region is assigned a 4-bit code (outcode) determined by where the region lies with respect to the clip edges
- Each bit in the outcode is set to either 1 (true) or 0 (false), depending on the following conditions:
  - Bit 1: above top edge $y > y_{max}$
  - Bit 2: below bottom edge $y < y_{min}$
  - Bit 3: right of right edge $x > x_{max}$
  - Bit 4: left of left edge $x < x_{min}$



Clip Rectangle

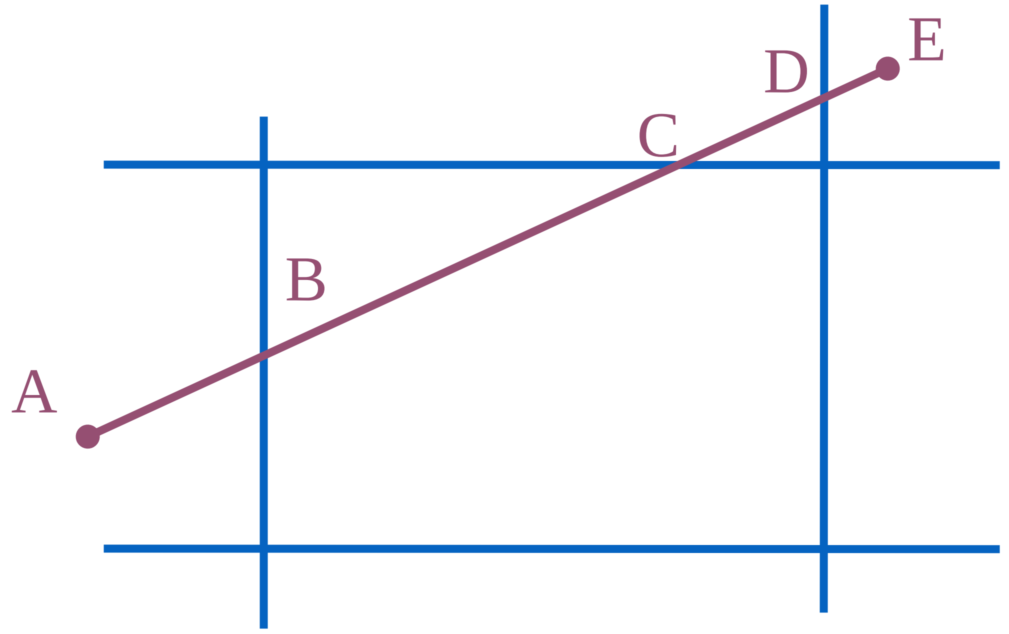
Cohen-Sutherland Line Clipping

- Say code1=outcode (P1), code2=outcode (P2)
- If **code1 = code2 = 0 (code1|code2=0)** then both ends inside so line inside – **trivial accept**
- If **(code1 | code2) = 0**, then one inside one outside – **inconclusive** - compute intersection point and check outcode for the intersection point
- If **code1 & code2 != 0** then both ends on the same side of the window – **trivial reject**
- If **code1 & code2 = 0** both ends are outside, but on the outside of different edges of the window - **inconclusive** - compute intersection point and check outcode for the intersection point



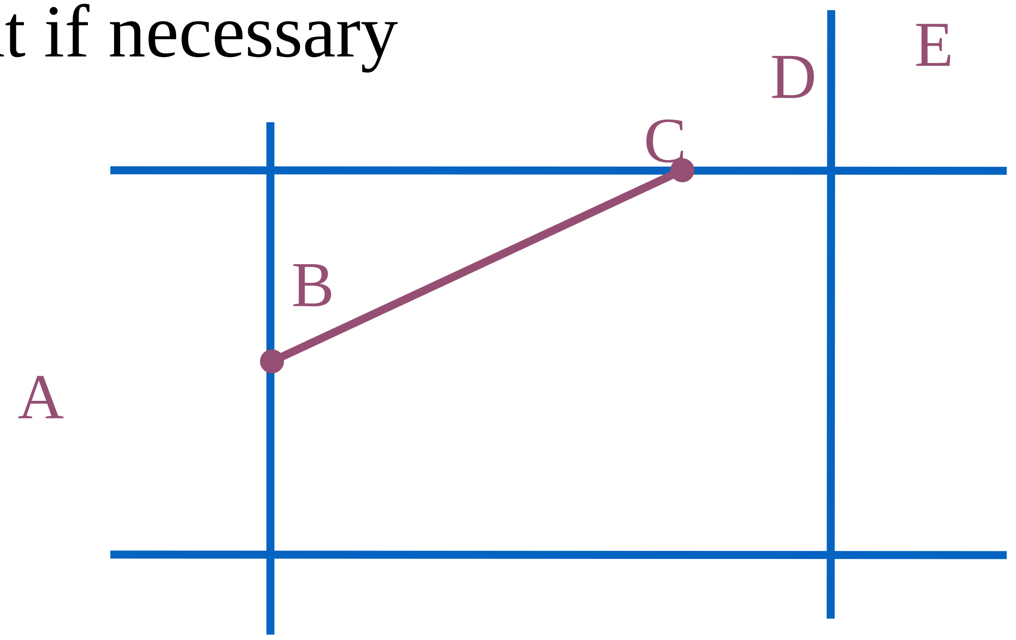
Cohen-Sutherland Line Clipping

- If line cannot be trivially accepted or rejected, subdivide so that one or both segments can be discarded
- Pick an edge that the line crosses
    - check against edges in same order each time



Cohen-Sutherland Line Clipping

- Discard portion on wrong side of edge and assign outcode to new vertex
- Apply trivial accept/reject tests and repeat if necessary

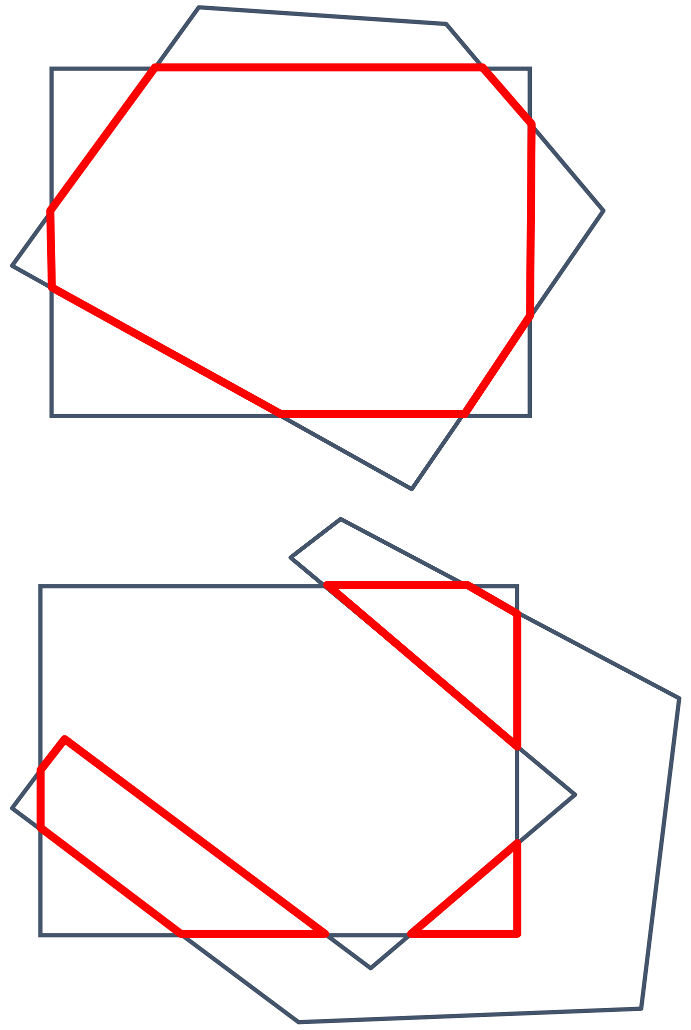


Cohen-Sutherland Discussion

- Key concepts
  - use outcodes to quickly eliminate/include lines
    - best algorithm when trivial accepts/rejects are common
  - must compute viewport clipping of remaining lines
    - non-trivial clipping cost
    - redundant clipping of some lines
- Basic idea, more efficient algorithms exist
  - Liang-Barsky

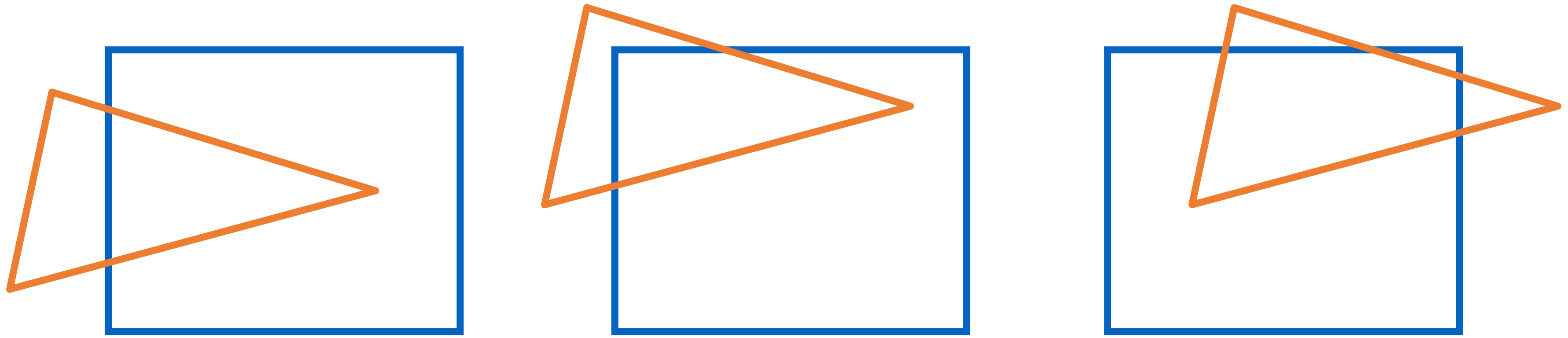
Polygon Clipping

- Objective
  - 2D: clip polygon against rectangular window
    - or general convex polygons
    - extensions for non-convex or general polygons
  - 3D: clip polygon against parallelpiped
- Not just clipping all boundary lines
  - may have to introduce new line segments

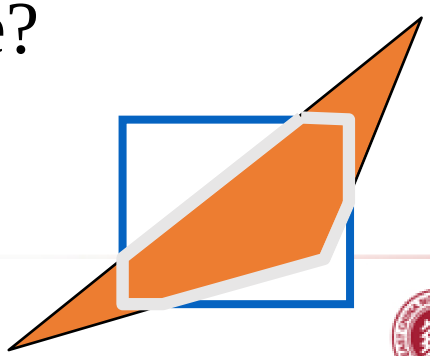


Why Is Clipping Hard?

- What happens to a triangle during clipping?
  - some possible outcomes:

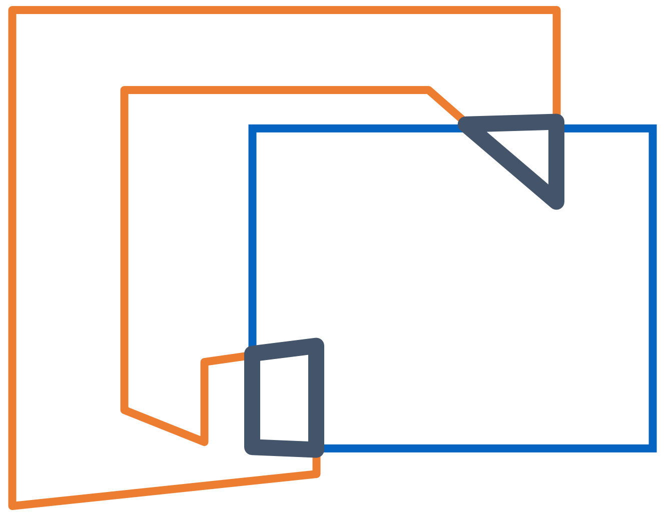


- How many sides can result from a triangle?
  - seven



Why Is Clipping Hard?

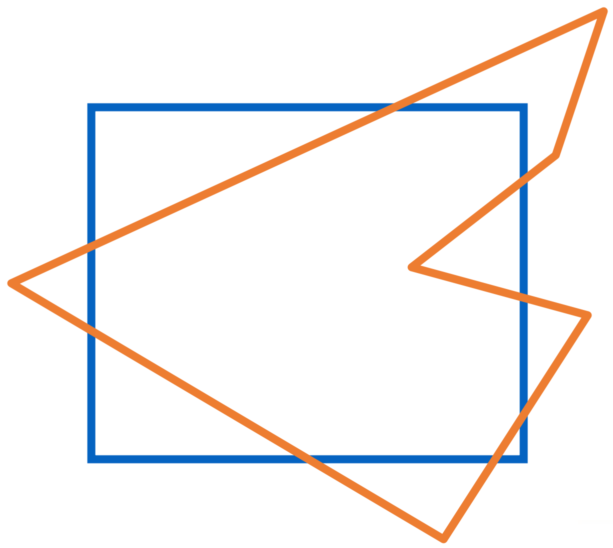
- A really tough case:



concave polygon to multiple polygons

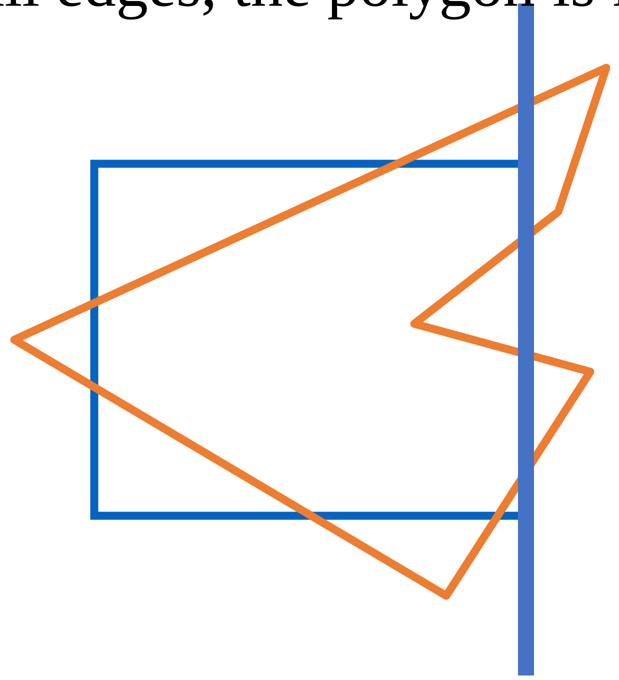
Sutherland-Hodgeman Clipping

- Basic idea:
  - consider each edge of the viewport individually
  - clip the polygon against the edge equation
  - after doing all edges, the polygon is fully clipped



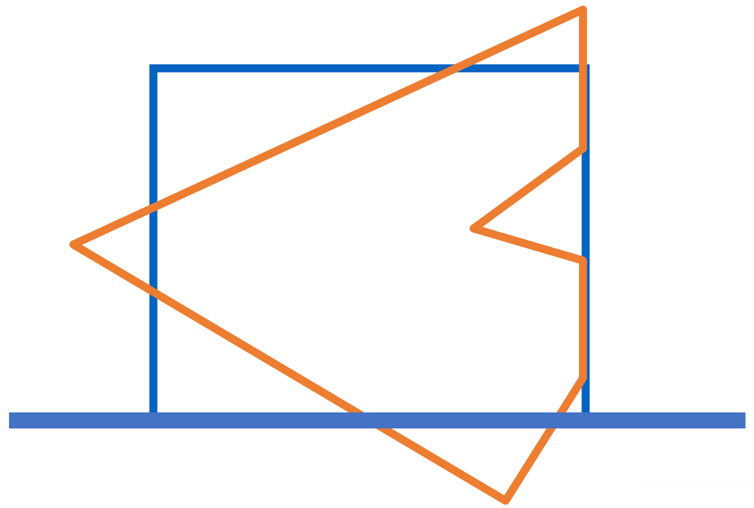
Sutherland-Hodgeman Clipping

- Basic idea:
  - consider each edge of the viewport individually
  - clip the polygon against the edge equation
  - after doing all edges, the polygon is fully clipped



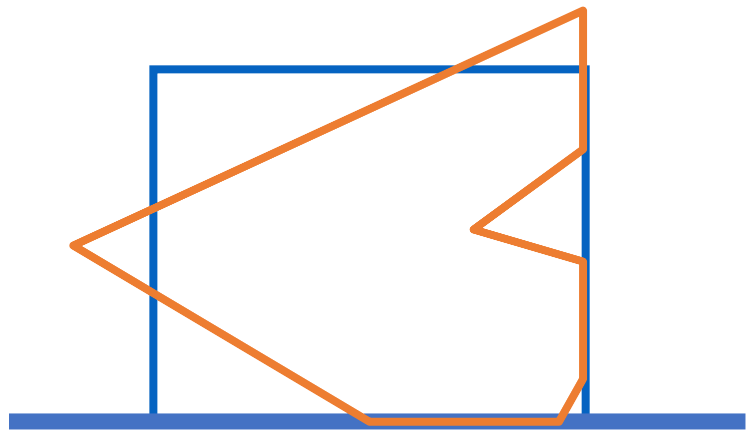
Sutherland-Hodgeman Clipping

- Basic idea:
  - consider each edge of the viewport individually
  - clip the polygon against the edge equation
  - after doing all edges, the polygon is fully clipped



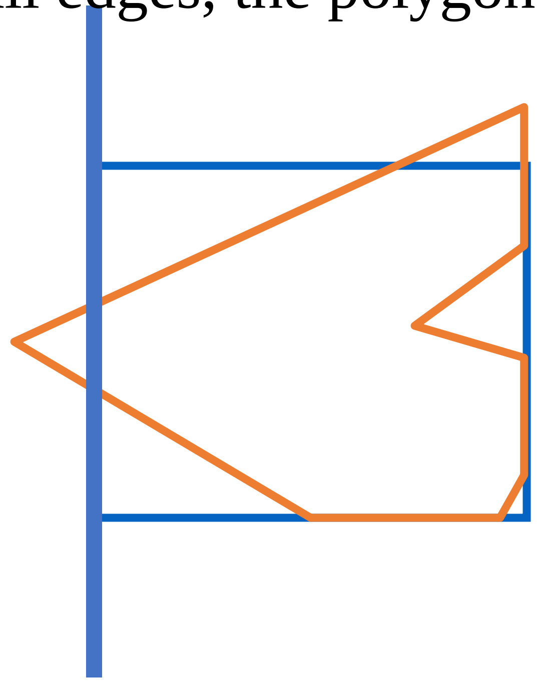
Sutherland-Hodgeman Clipping

- Basic idea:
  - consider each edge of the viewport individually
  - clip the polygon against the edge equation
  - after doing all edges, the polygon is fully clipped



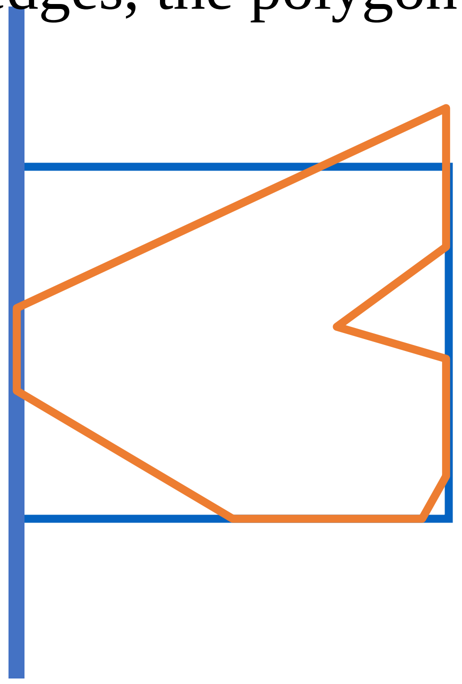
Sutherland-Hodgeman Clipping

- Basic idea:
  - consider each edge of the viewport individually
  - clip the polygon against the edge equation
  - after doing all edges, the polygon is fully clipped



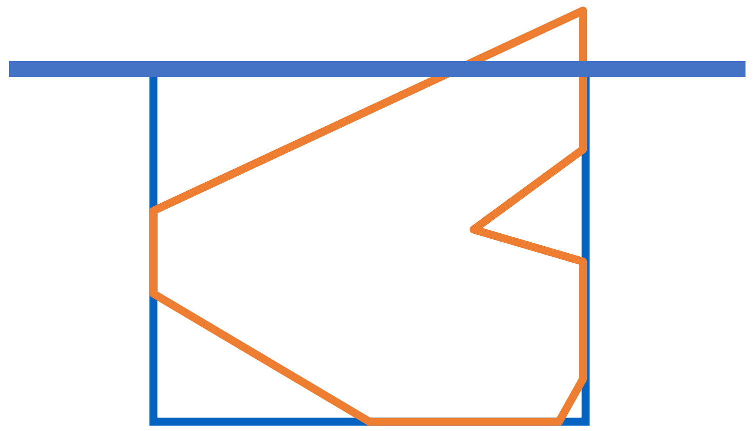
Sutherland-Hodgeman Clipping

- Basic idea:
  - consider each edge of the viewport individually
  - clip the polygon against the edge equation
  - after doing all edges, the polygon is fully clipped



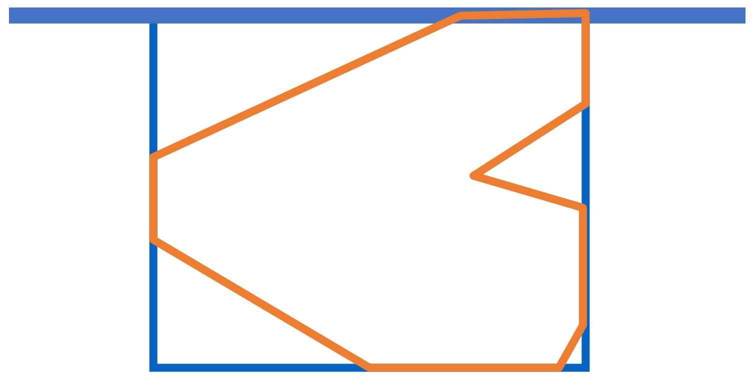
Sutherland-Hodgeman Clipping

- Basic idea:
  - consider each edge of the viewport individually
  - clip the polygon against the edge equation
  - after doing all edges, the polygon is fully clipped



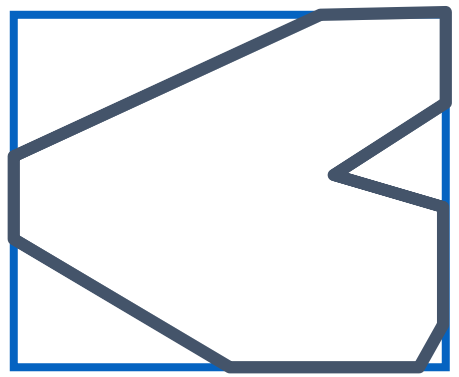
Sutherland-Hodgeman Clipping

- Basic idea:
  - consider each edge of the viewport individually
  - clip the polygon against the edge equation
  - after doing all edges, the polygon is fully clipped



Sutherland-Hodgeman Clipping

- Basic idea:
  - consider each edge of the viewport individually
  - clip the polygon against the edge equation
  - after doing all edges, the polygon is fully clipped

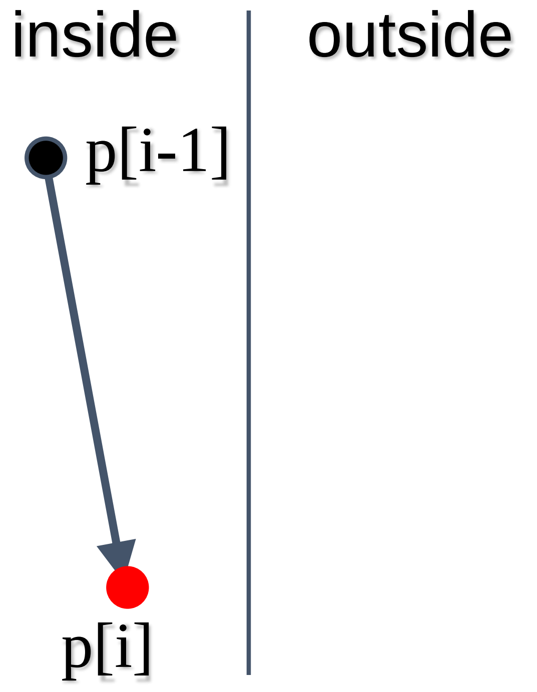


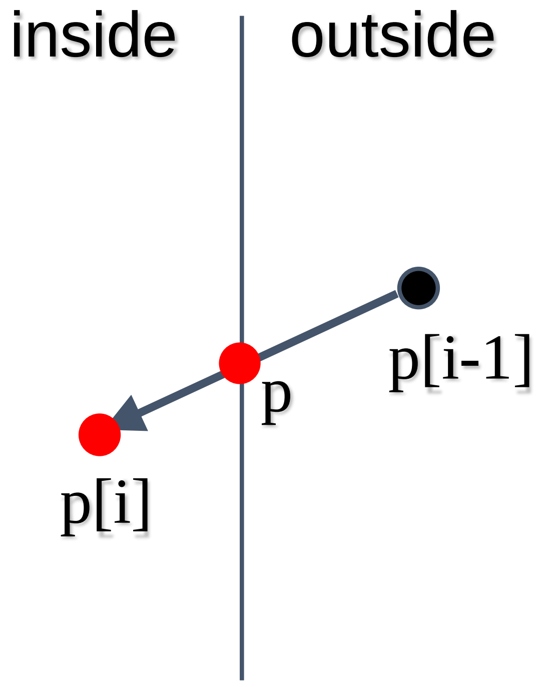
Sutherland-Hodgeman Clipping

- Consider the polygon as a list of vertices
- Clip the polygon against each edge of the clip region in turn
- Rewrite the polygon one vertex at a time – the rewritten polygon will be the clipped polygon
    - decide what to do based on 4 possibilities
        - is vertex inside or outside?
        - is previous vertex inside or outside?

Clipping Against One Edge

- p[i] inside: 2 cases



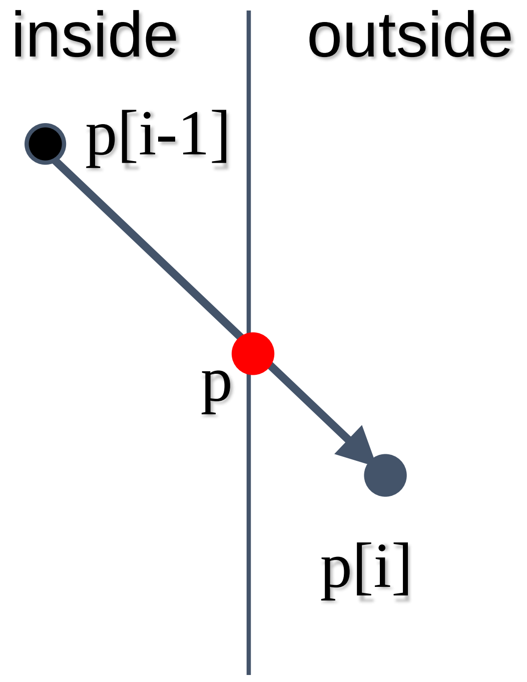


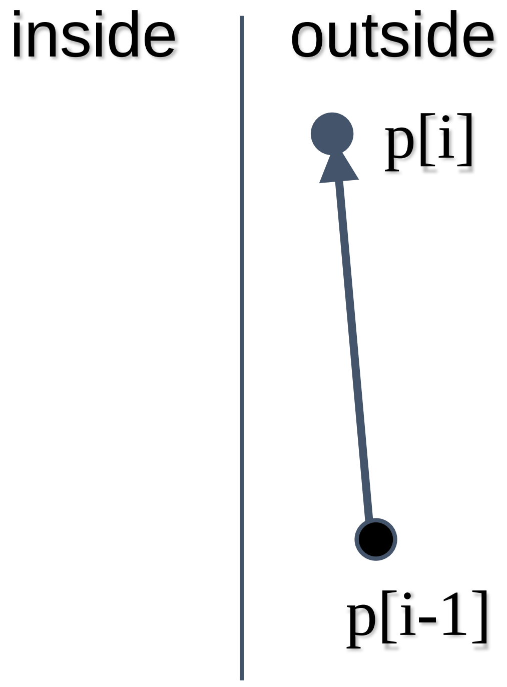
output: p[i]

output: p, p[i]

Clipping Against One Edge

- p[i] outside: 2 cases





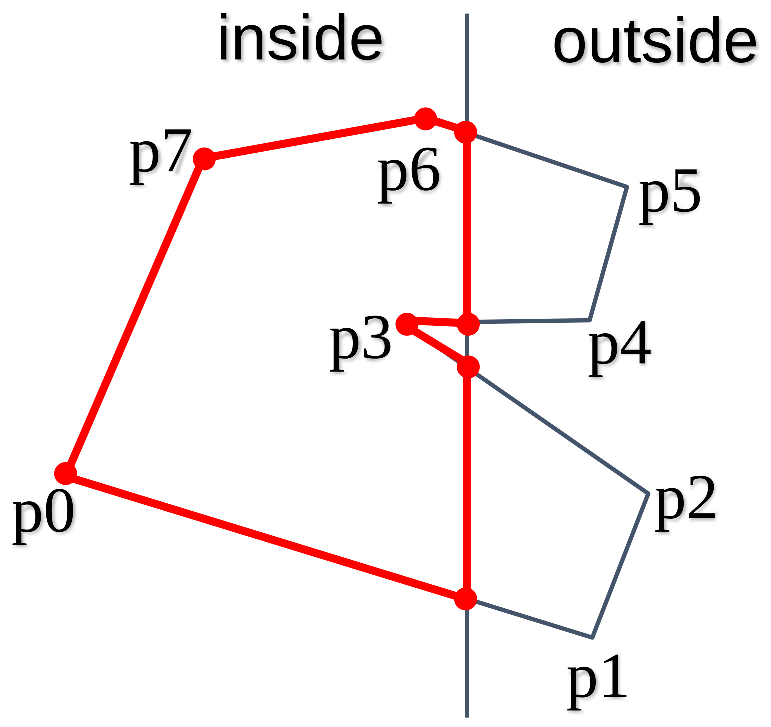
output: p

output: nothing

Clipping Against One Edge

```
clipPolygonToEdge( p[n], edge ) {
    for( i= 0 ; i< n ; i++ ) {
        if( p[i] inside edge ) {
          if( p[i-1] inside edge ) output p[i];     // p[-1]= p[n-1]
          else {
            p= intersect( p[i-1], p[i], edge ); output p, p[i];
          }
        } else {                                   // p[i] is outside edge
        if( p[i-1] inside edge ) {
          p= intersect(p[i-1], p[I], edge ); output p;
        }
    }
}
```

Sutherland-Hodgeman Example

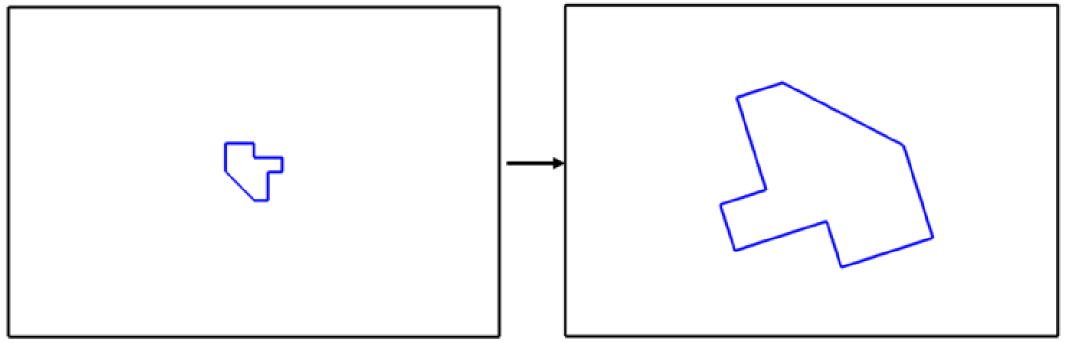


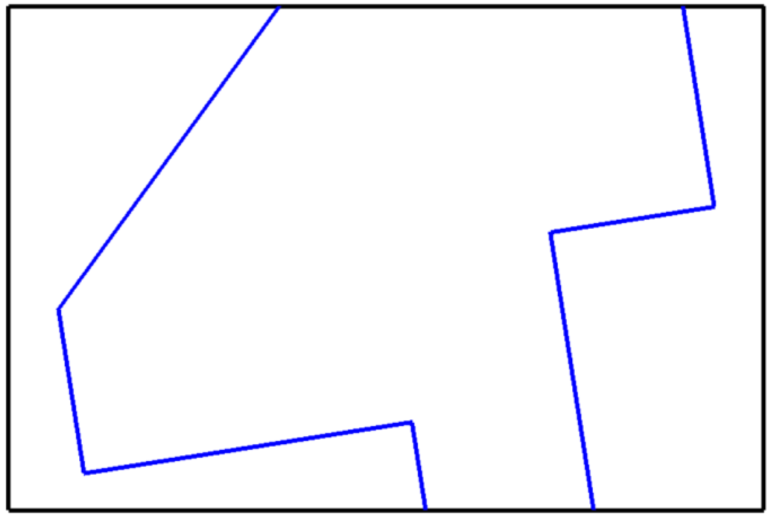
Sutherland-Hodgeman Discussion

- Similar to Cohen Sutherland line clipping
    - inside/outside tests: outcodes
    - intersection of line segment with edge: window-edge coordinates
- Clipping against individual edges independent
    - great for hardware (pipelining)
    - all vertices required in memory at same time
        - not so good, but unavoidable
        - another reason for using triangles only in hardware rendering

Assignment

- 实验编号：4
- 实验名称：几何变换与裁剪算法
- 实验内容
  - 实现基本二维图形变换操作
    - 平移变换
    - 缩放变换
    - 旋转变换
  - 实现Cohen-Sutherland裁剪算法





Extra Credit

- Could you clip a circle against a rectangle region?
  - How to trivially accept/reject a circle?
  - If the two regions overlap, you will need to solve the simultaneous line-curve equations to obtain the clipping intersection points.

Reference

- https://en.wikipedia.org/wiki/Transformation_matrix
- https://en.wikipedia.org/wiki/Clipping_(computer_graphics)
- https://en.wikipedia.org/wiki/Sutherland%E2%80%93Hodgman_algorithm