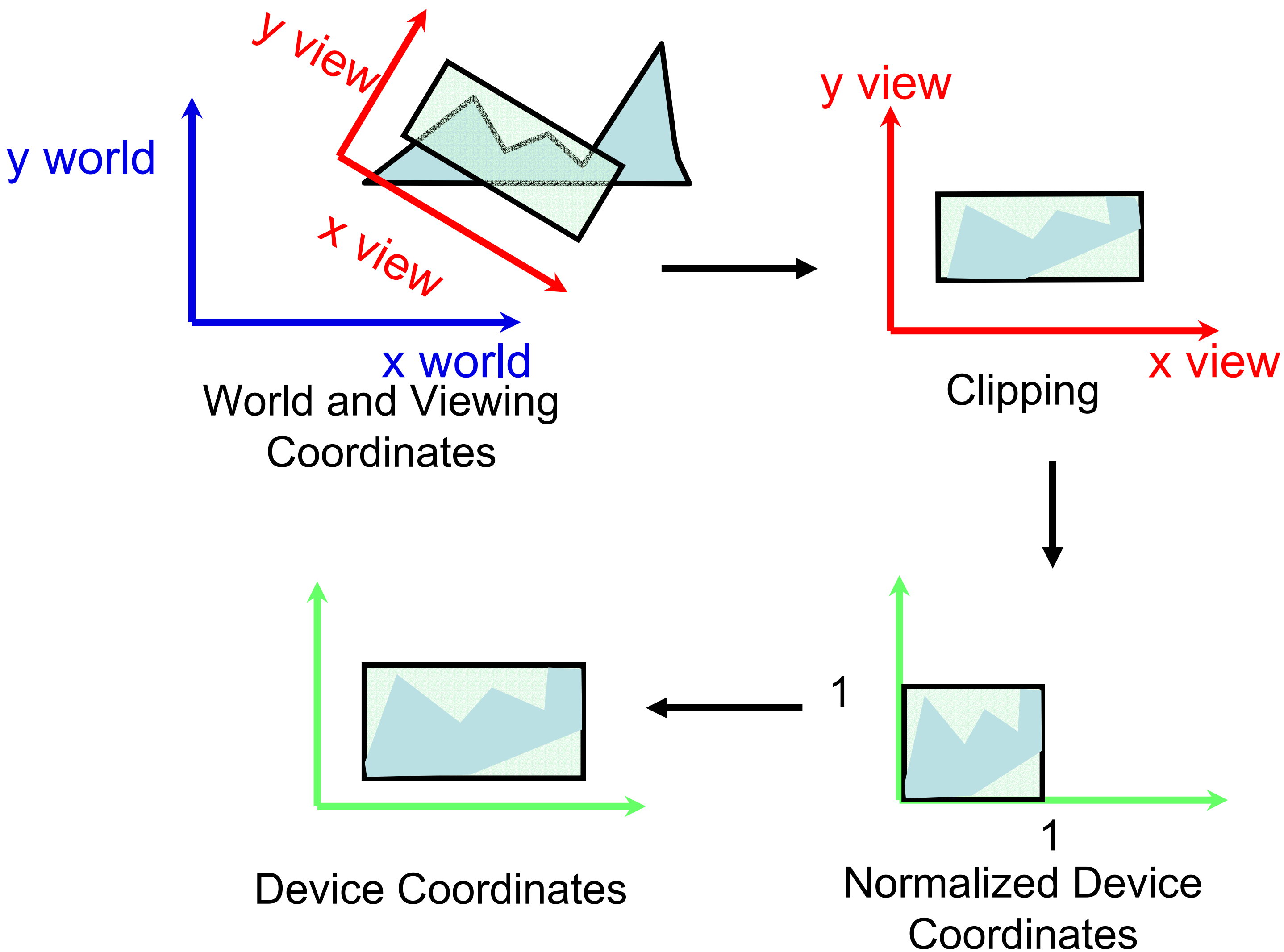
Line and Polygon Clipping

Foley & Van Dam, Chapter 3

Topics

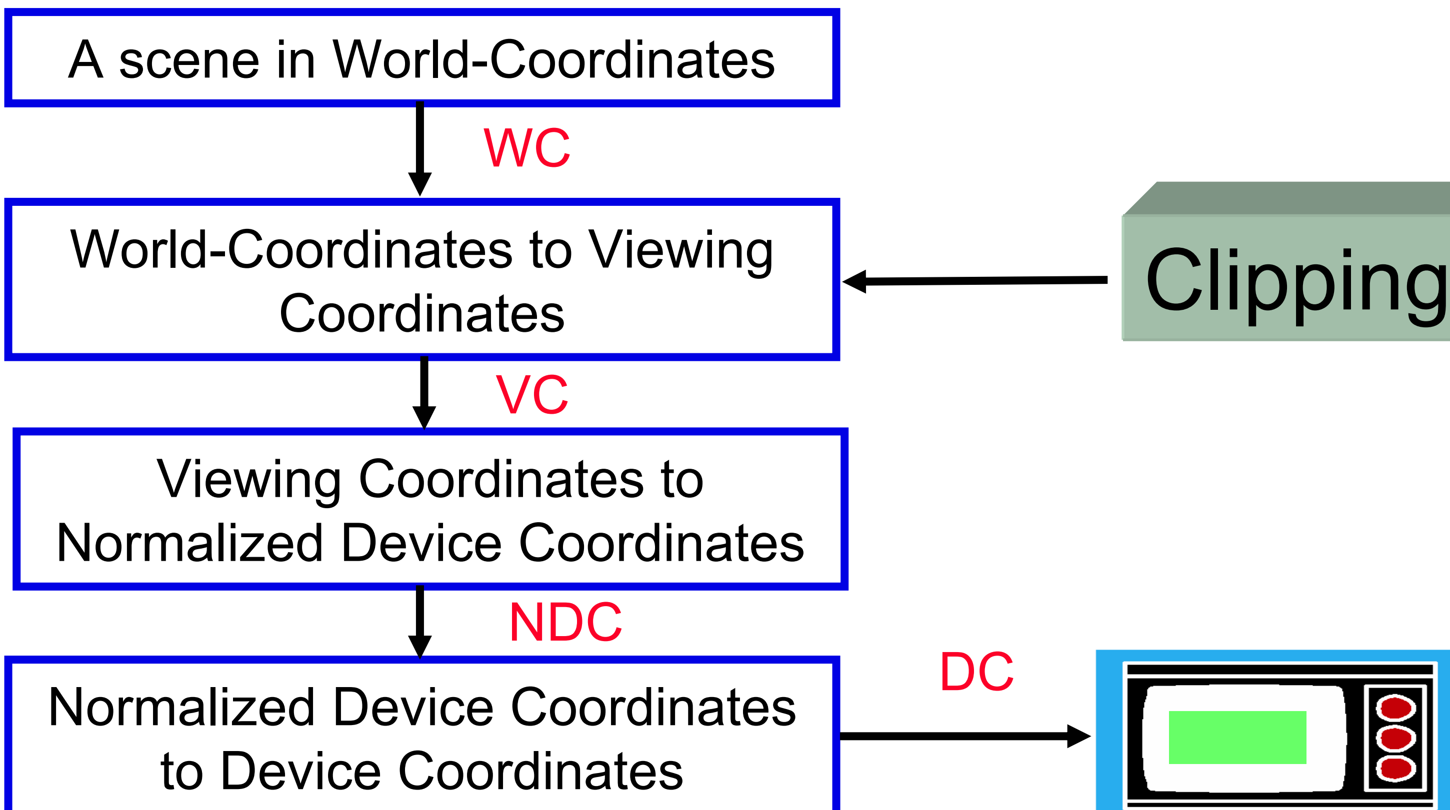
- Viewing Transformation Pipeline in 2D
- Line and polygon clipping
- Brute force analytic solution
- Cohen-Sutherland Line Clipping Algorithm
- Cyrus-Beck Line Clipping Algorithm
- Sutherland-Hodgman Polygon Clipping
- Sampling Theorem (Nyquist Frequency)

Viewing Transformation in 2D

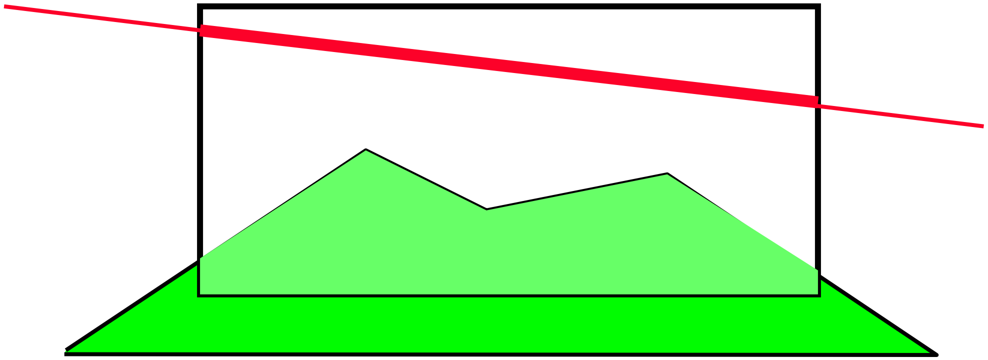


Viewing Transformation in 2D

- Objects are given in *world coordinates*
- The world is viewed through a *window*
- The window is mapped onto a *device viewport*



Line and Polygon Clipping

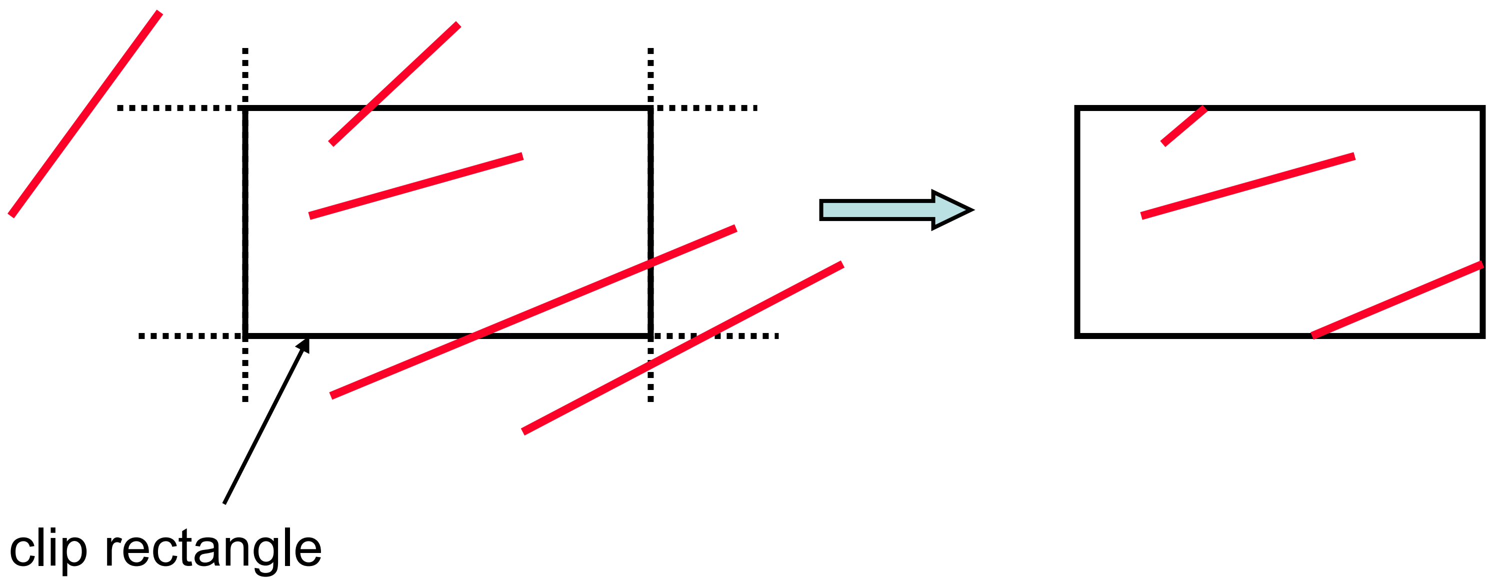


<u>The problem:</u>

Given a set of 2D lines or polygons and a window, clip the lines or polygons to their regions that are *inside* the window

Motivations

- Efficiency
- Display in portion of a screen
- Occlusions



Line Clipping

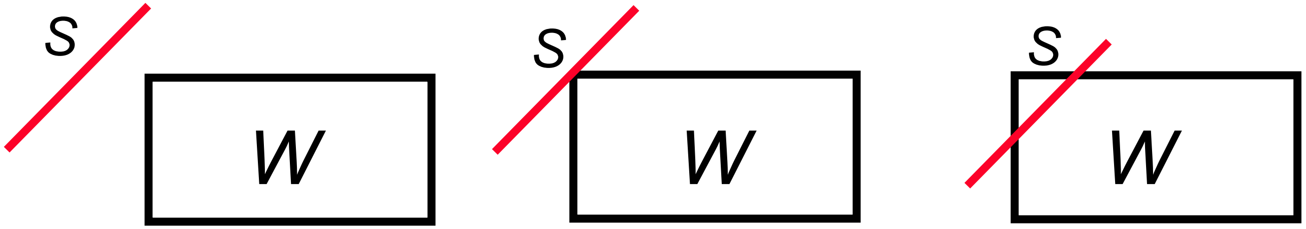
- We will deal only with lines (segments)
- Our window can be described by two extreme points:

$$(x_{min},y_{min}) \text{ and } (x_{max},y_{max})$$

- A point (x,y) is in the window iff:

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

Brute Force Analytic Solution



0, 1, or 2 intersections between a line and a window

- The intersection of convex regions is always convex
- Since both *W* and *S* are convex, their intersection is convex, i.e a single connected segment of *S*

Question: Can the boundary of two convex shapes intersect more than twice?

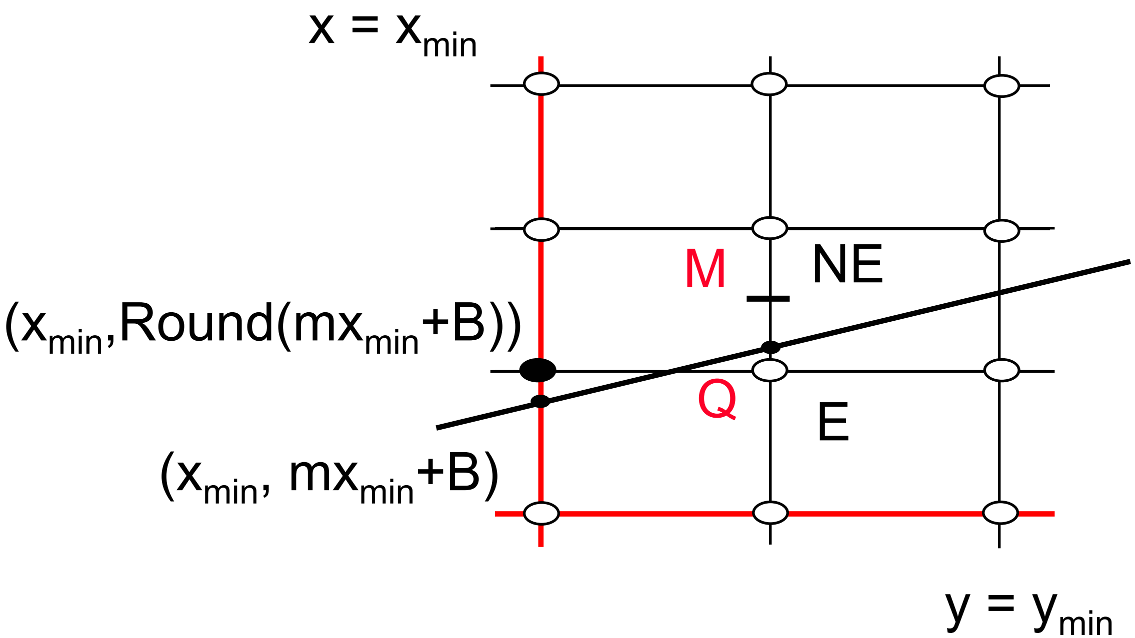
Pseudo Code for Midpoint Line Drawing

```
Line(x0,y0,x1,y1)
begin
    int dx, dy, x, y, d, ΔE, ΔNE ;
    x:= x0;    y=y0;
    dx := x1-x0;    dy := y1-y0;
    d := 2*dy-dx;
    ΔE := 2*dy;    ΔNE := 2*(dy-dx);
    PlotPixel(x,y);
    while(x < x1) do
        if (d < 0) then
          d:=d+ ΔE;
          x:=x+1;
        end;
        else
          d:=d+ ΔNE;
          x:=x+1;
          y:=y+1;
        end;
        PlotPixel(x,y);
    end;
end;
```

Assume $x_1 > x_0$ and $0 < \text{slope} \leq 1$

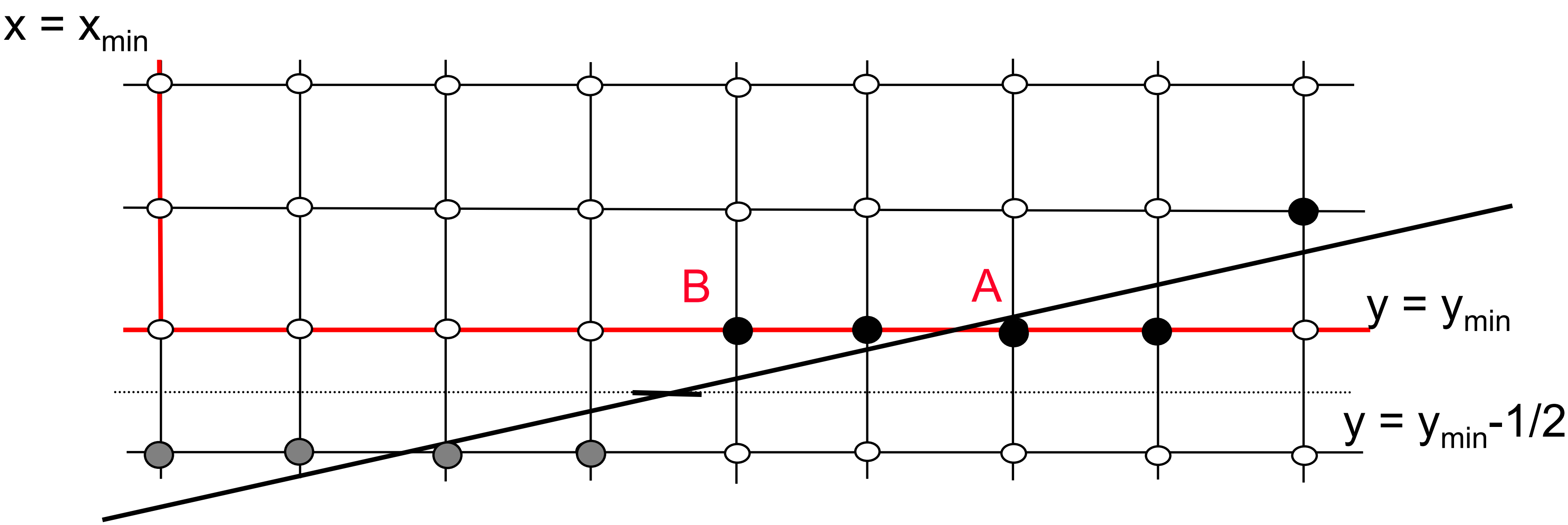
Line Clipping

Midpoint Algorithm: Intersection with a vertical edge



Line Clipping

Midpoint Algorithm: Intersection with a horizontal edge



Cohen-Sutherland for Line Clipping

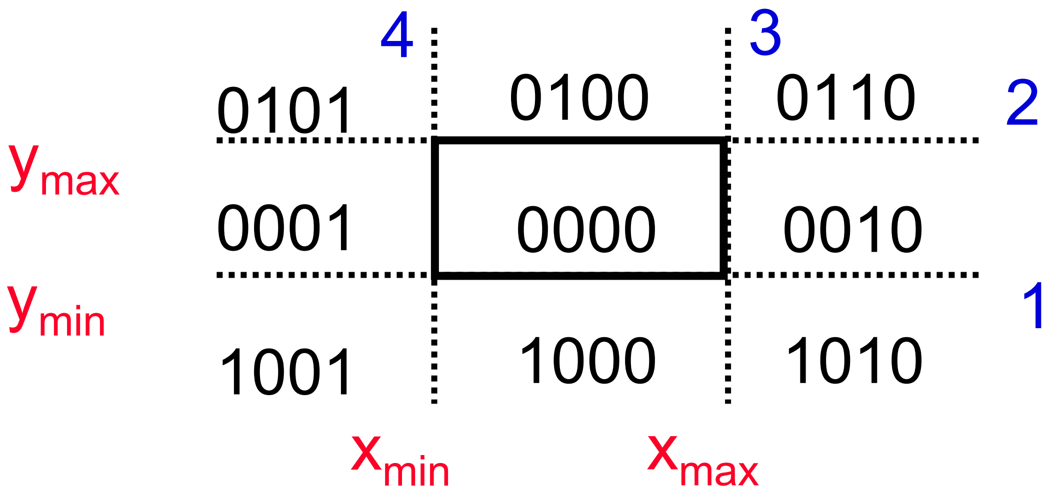
- Clipping is performed by computing intersections with four boundary segments of the window:

$$L_i, \quad i=1,2,3,4$$

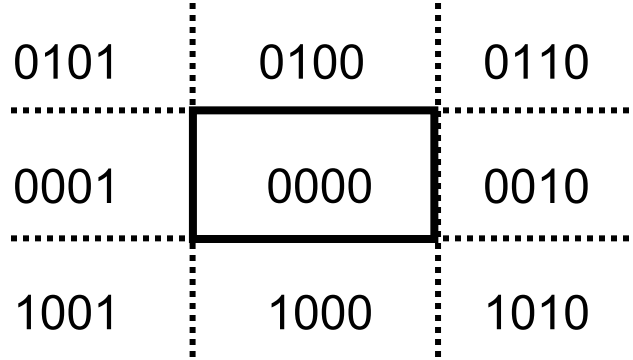
- **Purpose**: Fast treatment of lines that are trivially inside/outside the window
- Let $P=(x,y)$ be a point to be classified against window W
- **Idea**: Assign P a binary code consisting of a bit for each edge of W. The bit is 1 if the pixel is in the half-plane that does not contain W

Cohen-Sutherland for Line Clipping

bit	1	0
1	$y < y_{min}$	$y \geq y_{min}$
2	$y > y_{max}$	$y \leq y_{max}$
3	$x > x_{max}$	$x \leq x_{max}$
4	$x < x_{min}$	$x \geq x_{min}$



Cohen-Sutherland for Line Clipping



Given a line segment S from $p_0=(x_0,y_0)$ to $p_1=(x_1,y_1)$ to be clipped against a window W

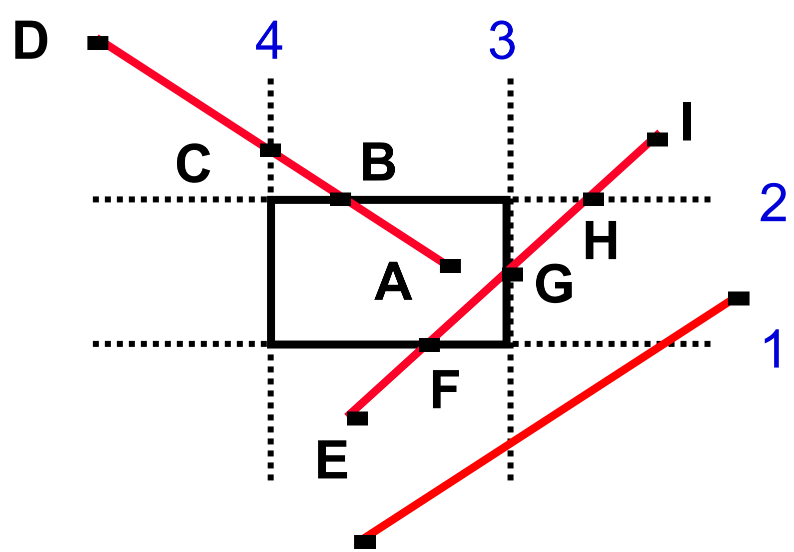
If code(p_0) **AND** code(p_1) is not zero, then S is *trivially rejected*

If code(p_0) **OR** code(p_1) is zero, then S is *trivially accepted*

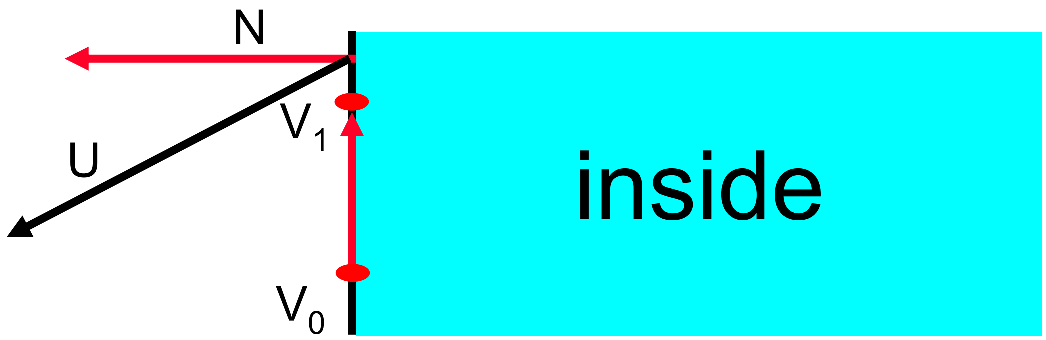
Cohen-Sutherland for Line Clipping

Otherwise: let assume w.l.o.g. that p_0 is outside W

- Find the intersection of S with the edge corresponding to the MSB in code(p_0) that is equal to 1. Call the intersection point p_2.
- Run the procedure for the new segment (p_1 , p_2).



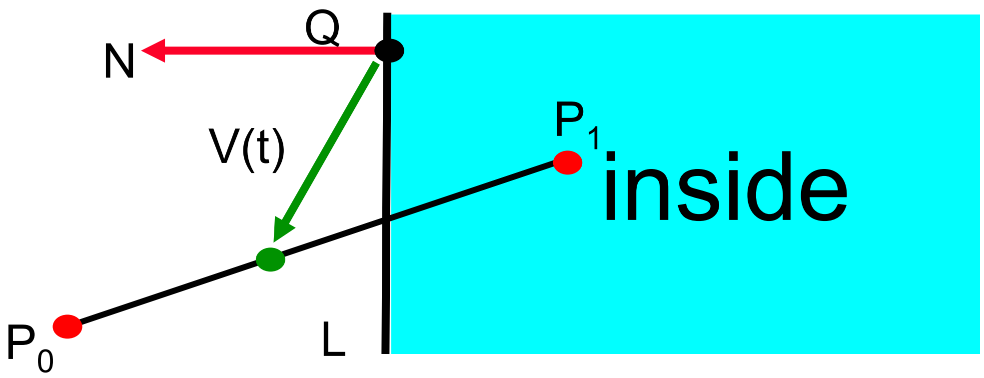
Cyrus-Beck Line Clipping



Inside/Outside Test:

- Assume WLOG that $V=(V_1-V_0)$ is the border vector where "inside" is to its right
- If $V=(V_x,V_y)$, N is the normal to V, pointing outside, defined by $N=(-V_y,V_x)$
- Vector U points "outside" if $N \cdot U > 0$
- Otherwise U points "inside"

Cyrus-Beck Line Clipping



The parametric line $P(t)=P_0+(P_1-P_0)t$

The parametric vector $V(t)=P(t)-Q$

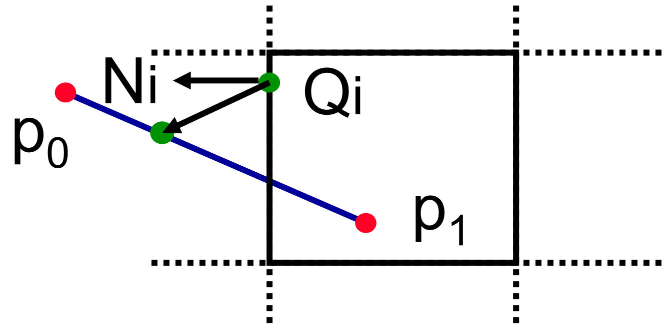
The segment P_0P_1 intersects the line L at t_0 satisfying $V(t_0)\cdot N=0$

The intersection point is $P(t_0)$

$\Delta=P_1-P_0$ points inside if $(P_1-P_0)\cdot N<0$. Otherwise it points outside

If L is vertical, intersection can be computed using the explicit equation

Cyrus-Beck Line Clipping



- Denote $p(t)=p_0+(p_1-p_0)t \quad t\in[0..1]$
- Let Q_i be a point on the edge L_i with outside pointing normal N_i
- $V(t) = p(t)-Q_i$ is a parameterized vector from Q_i to the segment $P(t)$
- $N_i \cdot V(t) = 0$ iff $V(t) \perp N_i$
- We are looking for *t* satisfying $N_i \cdot V(t) = 0$

Cyrus-Beck Line Clipping

$$
\begin{aligned}
0 &= N_i \cdot V(t) \\
&= N_i \cdot (p(t)-Q_i) \\
&= N_i \cdot (p_0+(p_1-p_0)t-Q_i) \\
&= N_i \cdot (p_0-Q_i) + N_i \cdot (p_1-p_0)t
\end{aligned}
$$

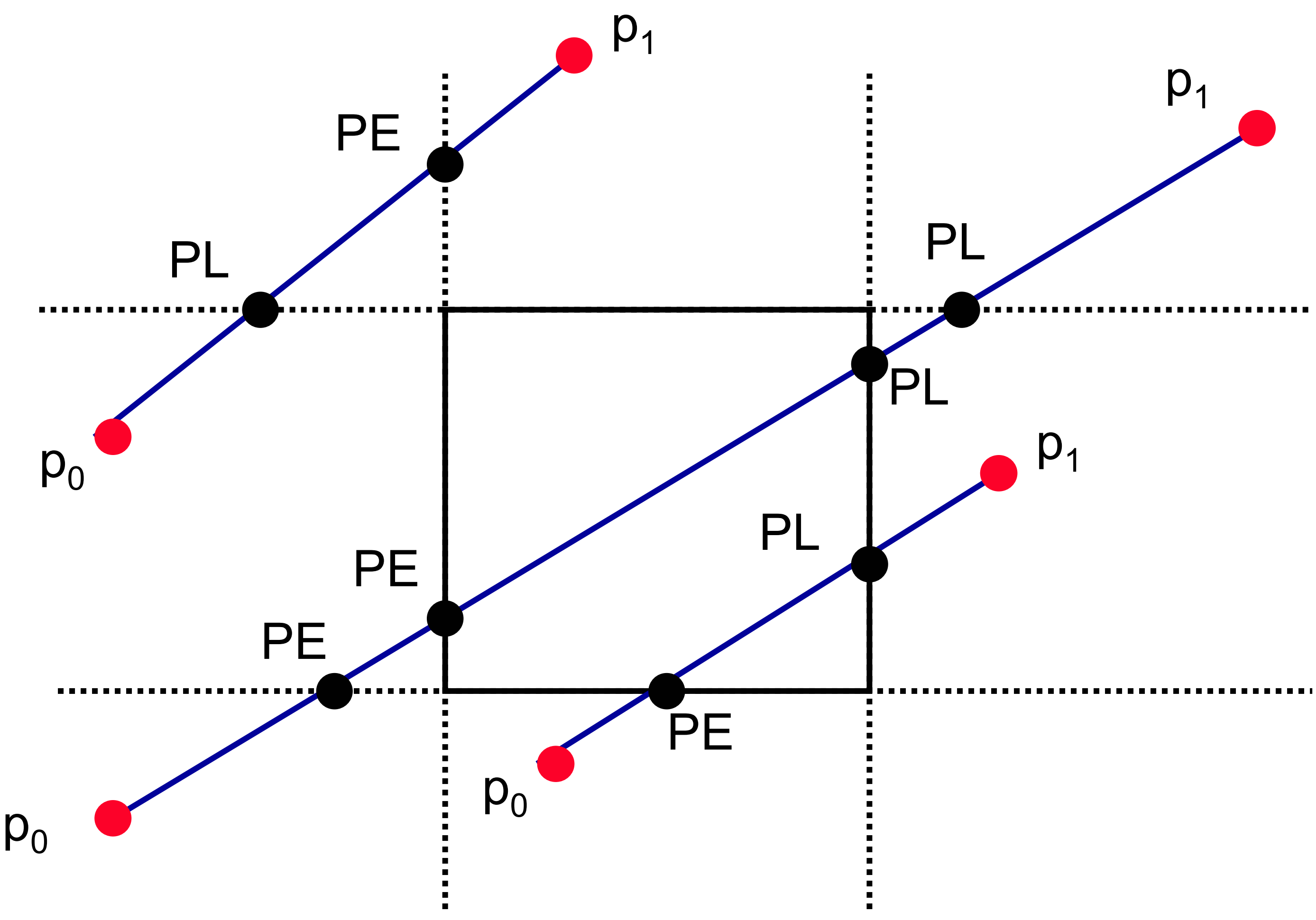
Solving for t we get:

$$
t = \frac{N_i \cdot (p_0-Q_i)}{-N_i \cdot (p_1-p_0)} = \boxed{\frac{N_i \cdot (p_0-Q_i)}{-N_i \cdot \Delta}}
$$

where $\Delta=(p_1-p_0)$

Comment: If $N_i \cdot \Delta=0$, t has no solution ($V(t) \perp N_i$)

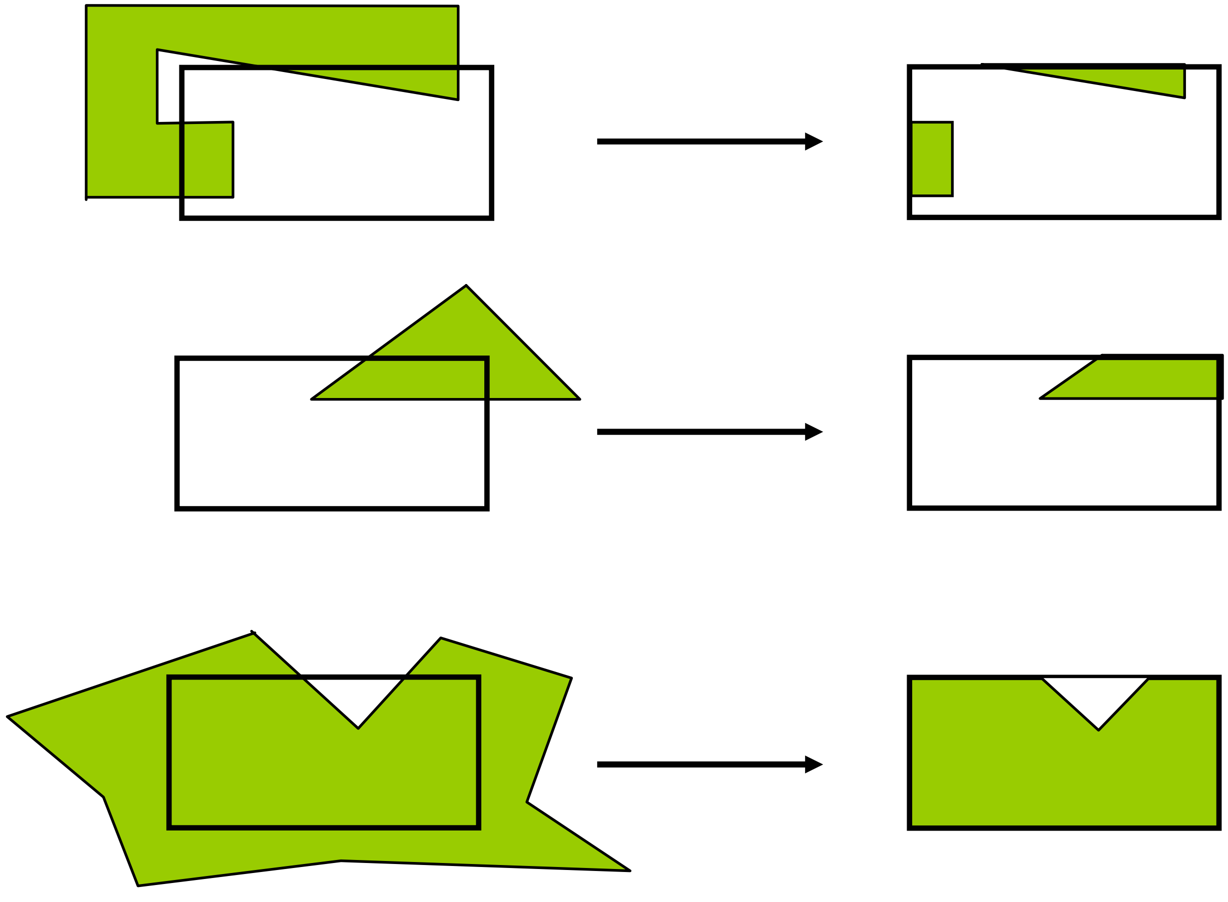
Cyrus-Beck Line Clipping



Cyrus-Beck Line Clipping

- The intersection of *p(t)* with all four edges L_i is computed, resulting in up to four t_i values
- If $t_i<0$ *or* $t_i>1$, t_i can be discarded
- Based on the sign of $N_i \cdot \Delta$, each intersection point is classified as ***PE*** (potentially entering) or ***PL*** (potentially leaving)
- ***PE*** with the largest *t* and ***PL*** with the smallest *t* provide the domain of p(t) inside *W*
- The domain, if inverted, signals that p(t) is totally outside

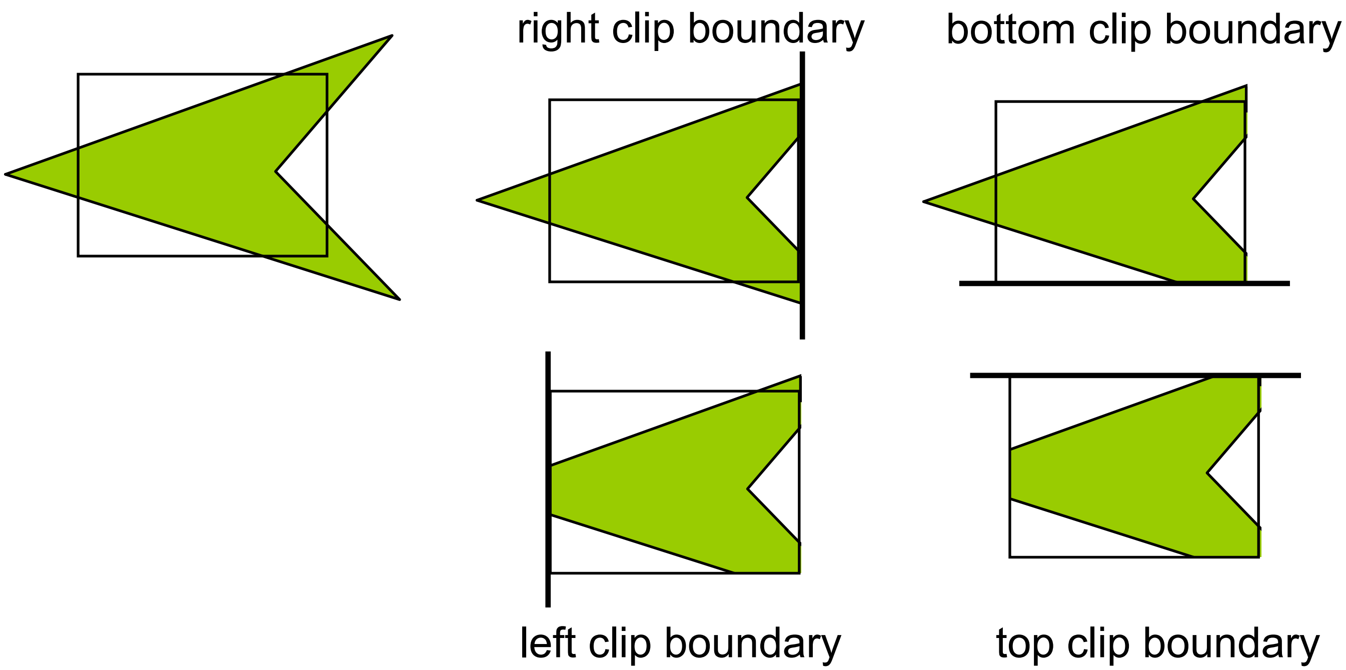
Sutherland-Hodgman Polygon-Clipping Algorithm



Sutherland-Hodgman Polygon-Clipping Algorithm

Idea: Clip a polygon by successively clipping against each (infinite) clip edge

After each clipping a new set of vertices is produced.

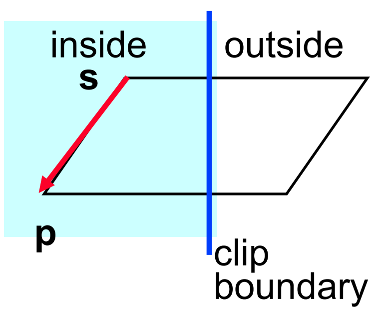


Sutherland-Hodgman Polygon-Clipping Algorithm

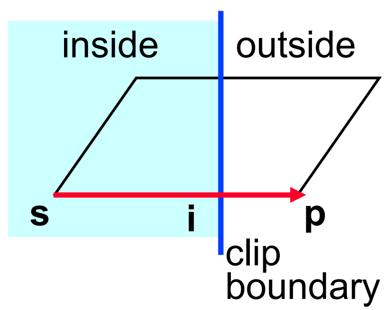
For each clip edge - scan the polygon and consider the relation between successive vertices of the polygon

Each iteration adds 0, 1 or 2 new vertices

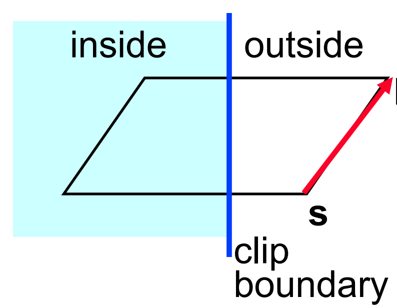
Assume vertex **s** has been dealt with, vertex **p** follows:



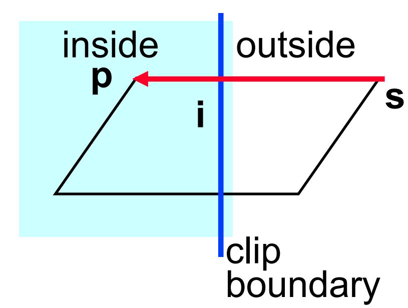
p added to output list



i added to output list

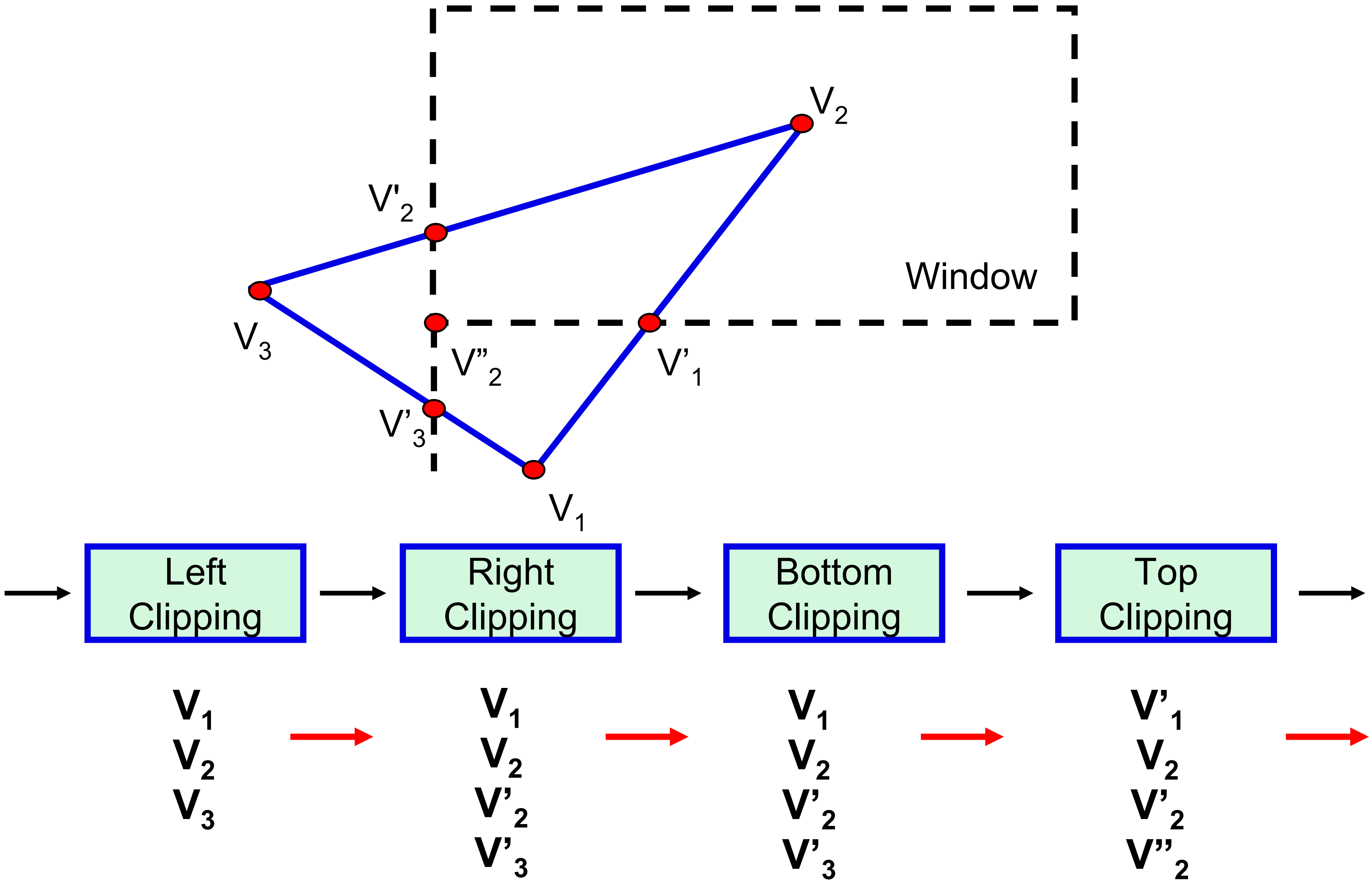


no output



i and **p** added to output list

Sutherland-Hodgman Polygon-Clipping Algorithm



Sampling Theorem

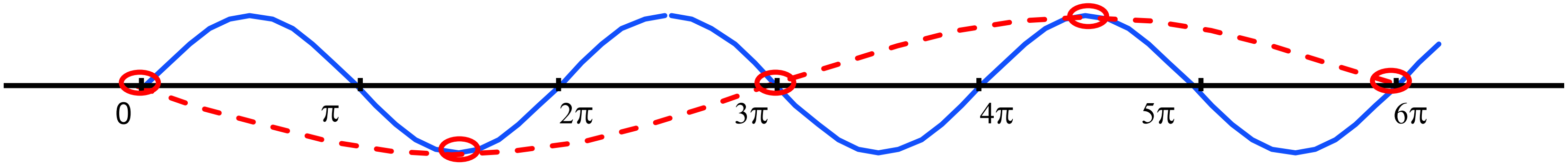
Question: How dense should be the pixel grid in order to draw properly a drawn object?

Given a sampling at intervals equal to **d** then one may recover frequencies of wavelength **> 2d**

Aliasing: If the sampling interval is **more** than **1/2** the wavelength, erroneous frequencies may be produced

Sampling Theorem

1D Example:



Rule of Thumb: To observe details of size **d** one must sample at **d/2** intervals

To observe details at frequency **f** (=1/d) one must sample at frequency **2f**. The Frequency **2f** is the **NYQUIST frequency**

Sampling Theorem

2D Example: Moire' Effect