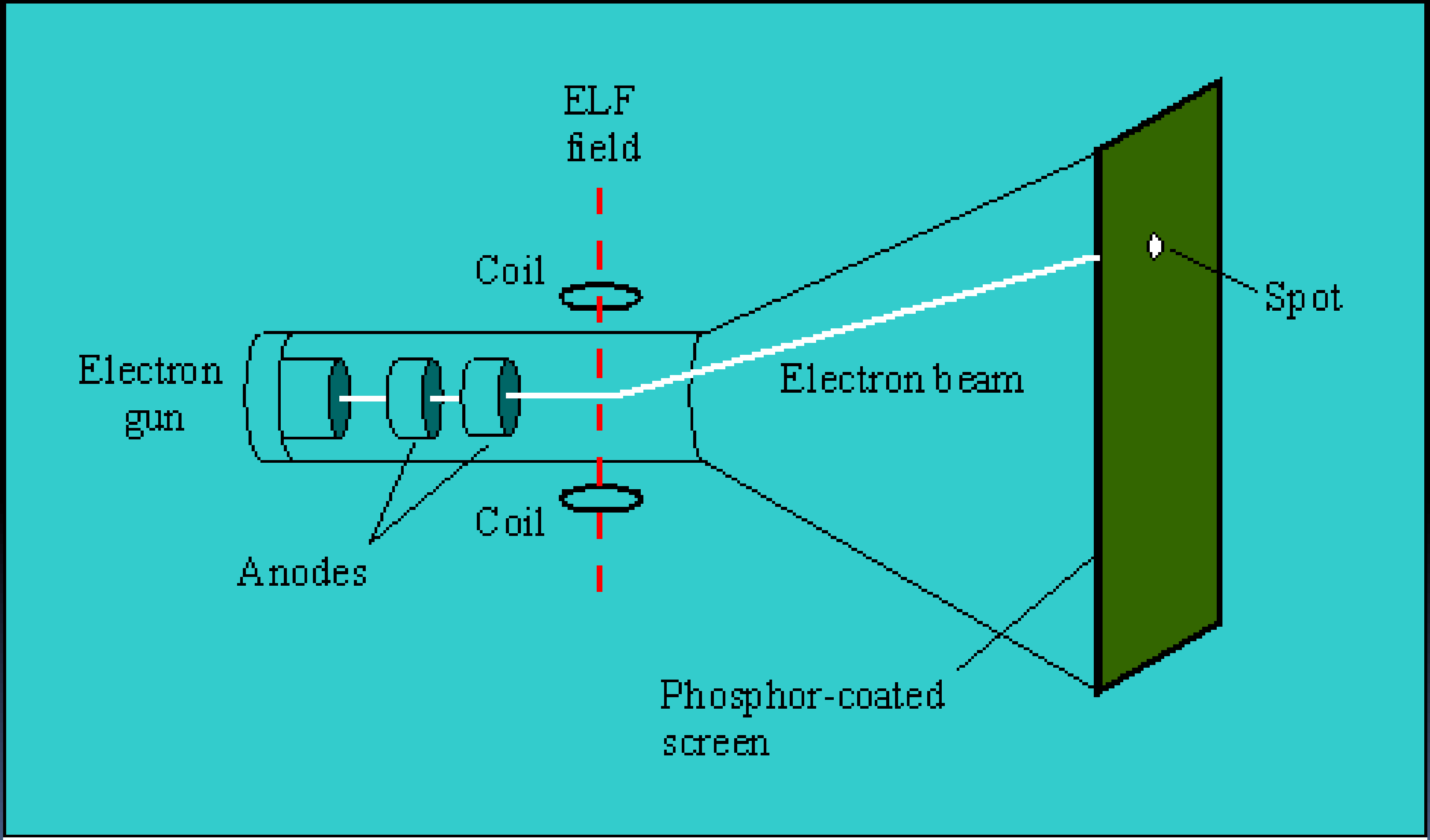
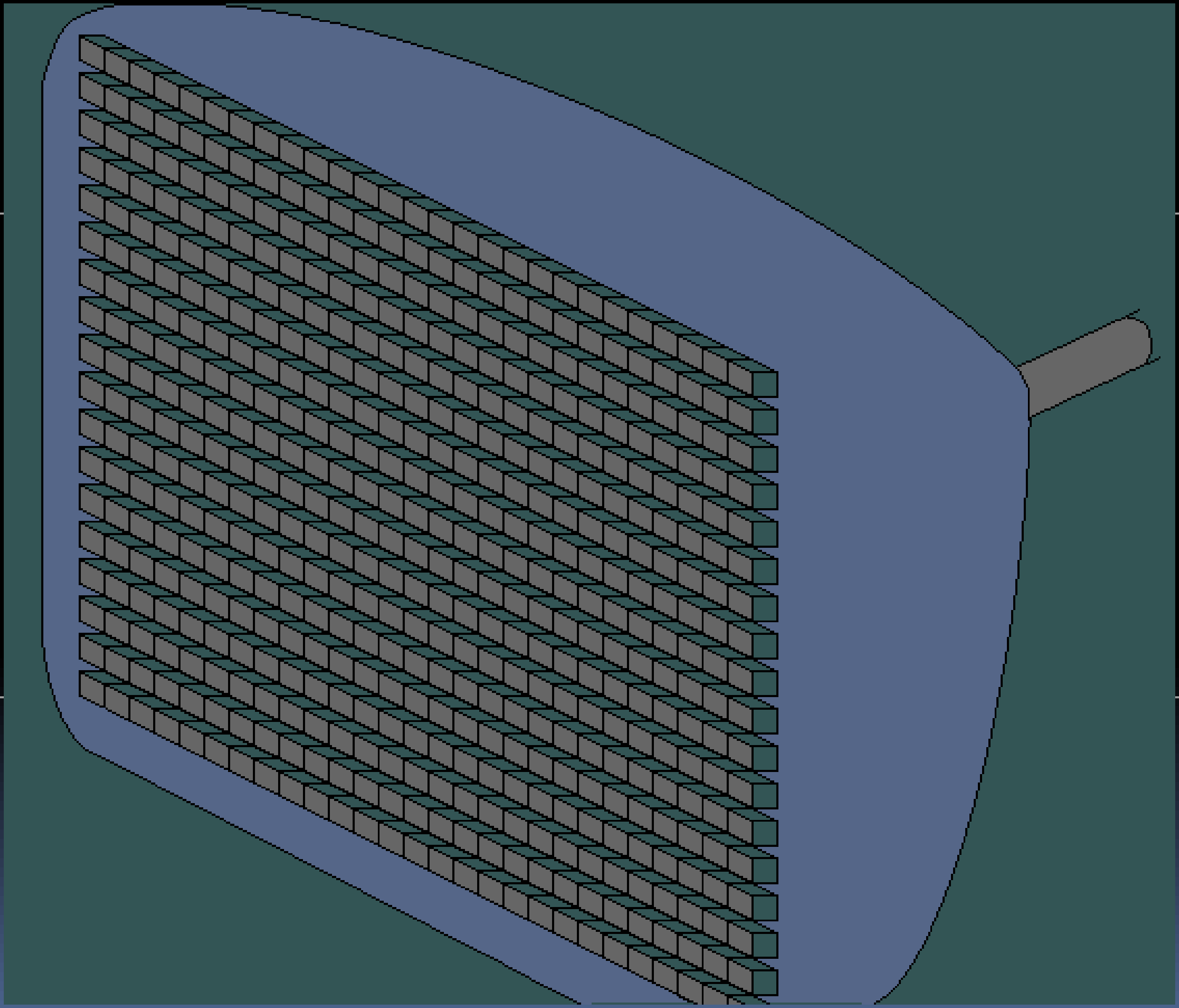
RASTERIZATION

Raster Scan - Monitors





Monitors

- Raster scan
- Progressice/interlaced scanning
- Phosphor triad - RGB pixel group
- Refresh rate 60 NTSC, 50 PAL
- Flicker - low refresh rate

Aspect Ratio

- Screen width / screen height

4/3 = 1.33

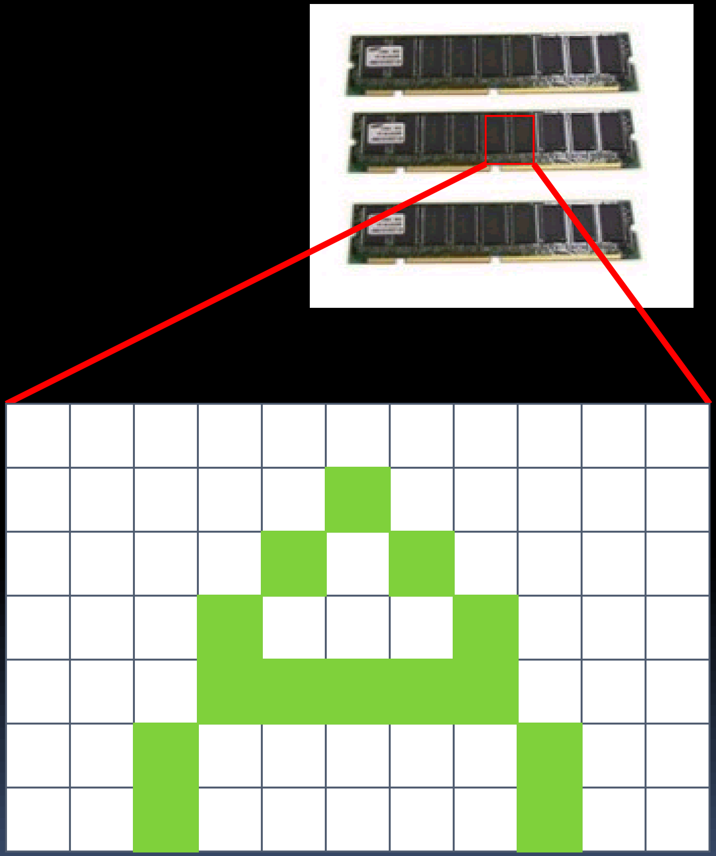
Digital Images: pixels

Frame Buffer

Frame Buffer

A block of memory, dedicated that contains the pixel array to be passed to the optical display system

Each pixel encodes color or other properties (e.g., opacity)



Frame Buffer Concepts

Pixel: One element of frame buffer
- uniquely accessible point in image

Resolution: Width x Height (in pixels)
- 640x480, 1280x1024, 1920x1080

Color depth: Number of bits per-pixel in the buffer
- 8, 16, 24, 32-bits for RGBA

Buffer size: Total memory allocated for buffer

Frame Buffer Opacity

Alpha

- Used for compositing or merging images

Alpha channel – added to color

- Holds the alpha value for every pixel
  - 8 bit range: 0 (transparent) – 255 (opaque)

How Much Memory?

Buffer size = width * height *color depth

For example:

If width=640, height=480, color depth=24 bits

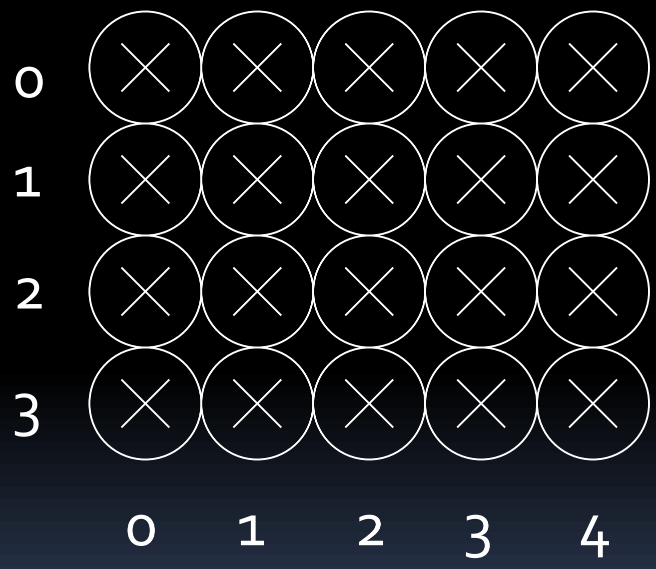
Buffer size = 640 * 480 * 3 = 921,600 bytes

If width=1920, height=1080, color depth=24 bits

Buffer size = 1920 * 1080 * 3 = 6,220,800 bytes

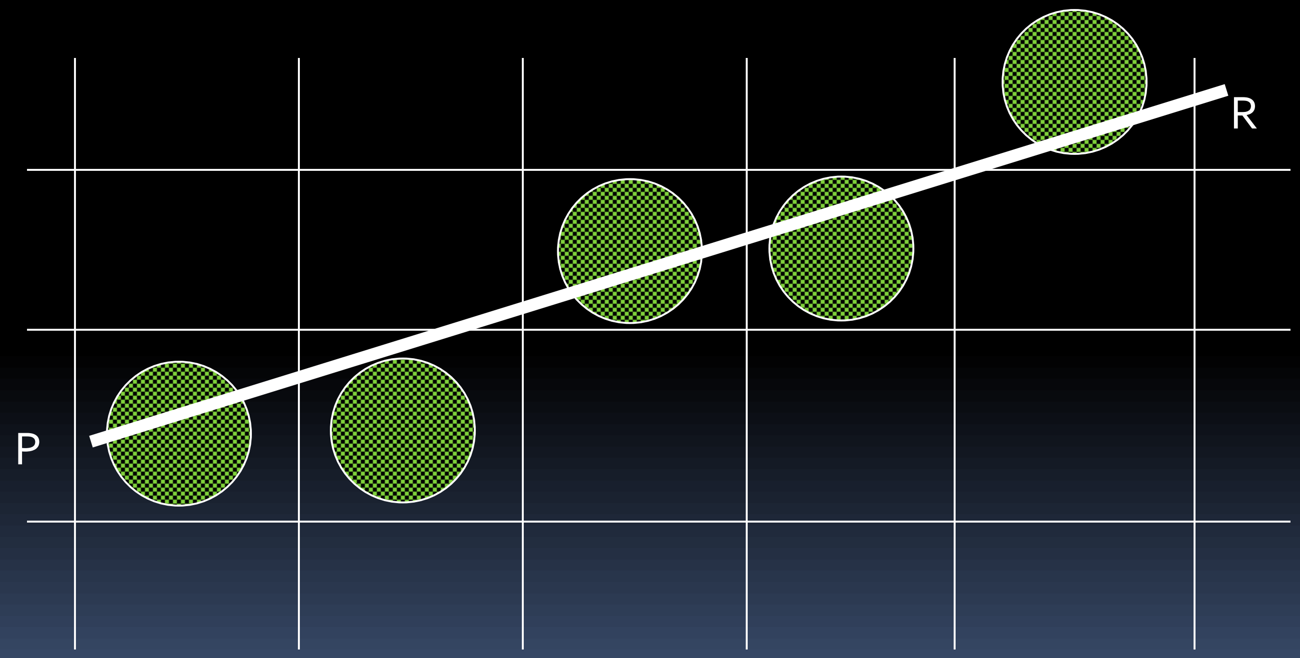
Rasterization

Array of pixels



Rasterizing Lines

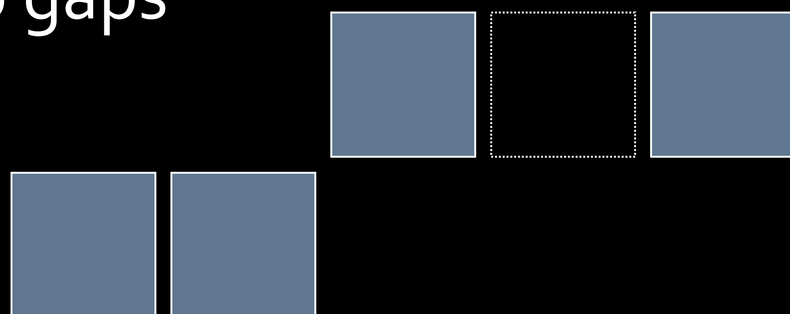
Given two endpoints, P= (x_0, y_0), R = (x_1, y_1) find the pixels that make up the line



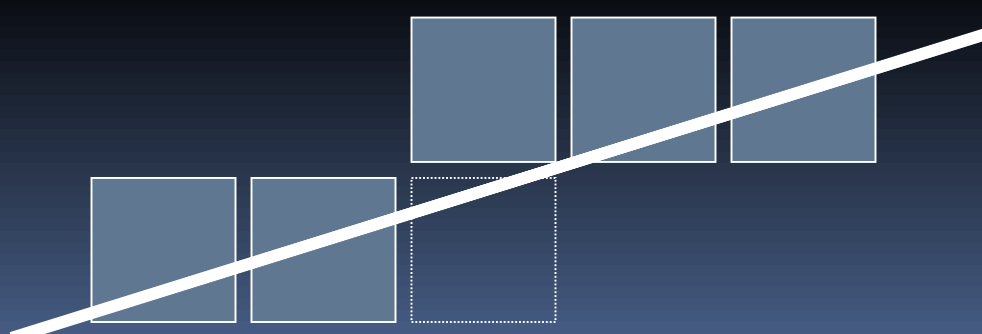
Rasterizing Lines

Requirements

1. No gaps



2. Minimize error (distance to true line)

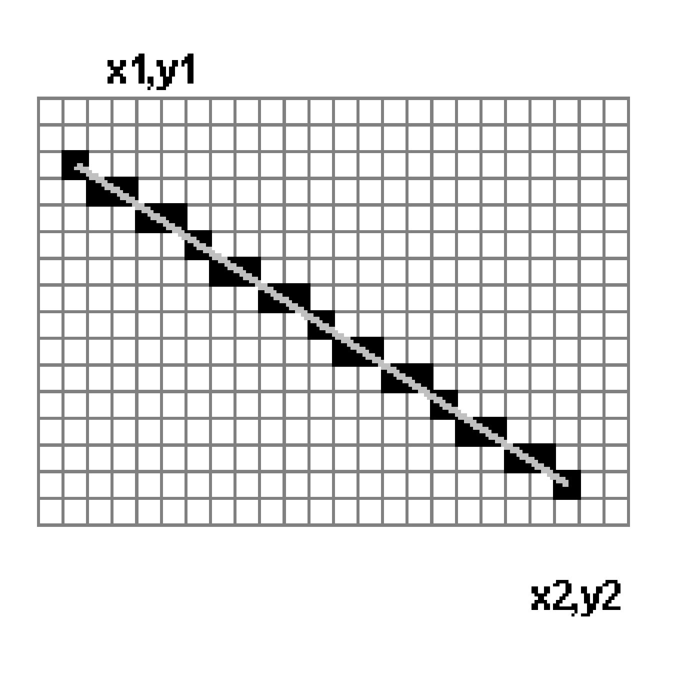


Ideal Lines

- From the equation of a line

$$y = y_1 + \left(\frac{y_2 - y_1}{x_2 - x_1} \right) (x - x_1)$$

- Find a discretization



Scan Converting Line

```
for (x=x1;x<=x2;x++) {
    y = round(y1+(dy/dx)*(x-x1))
    setPixel(x,y,colour)
}
```

- Problems
  - one divide, one round, two adds, one multiply per pixel
  - no coherence at all

First Speed Up - An Obvious Thing

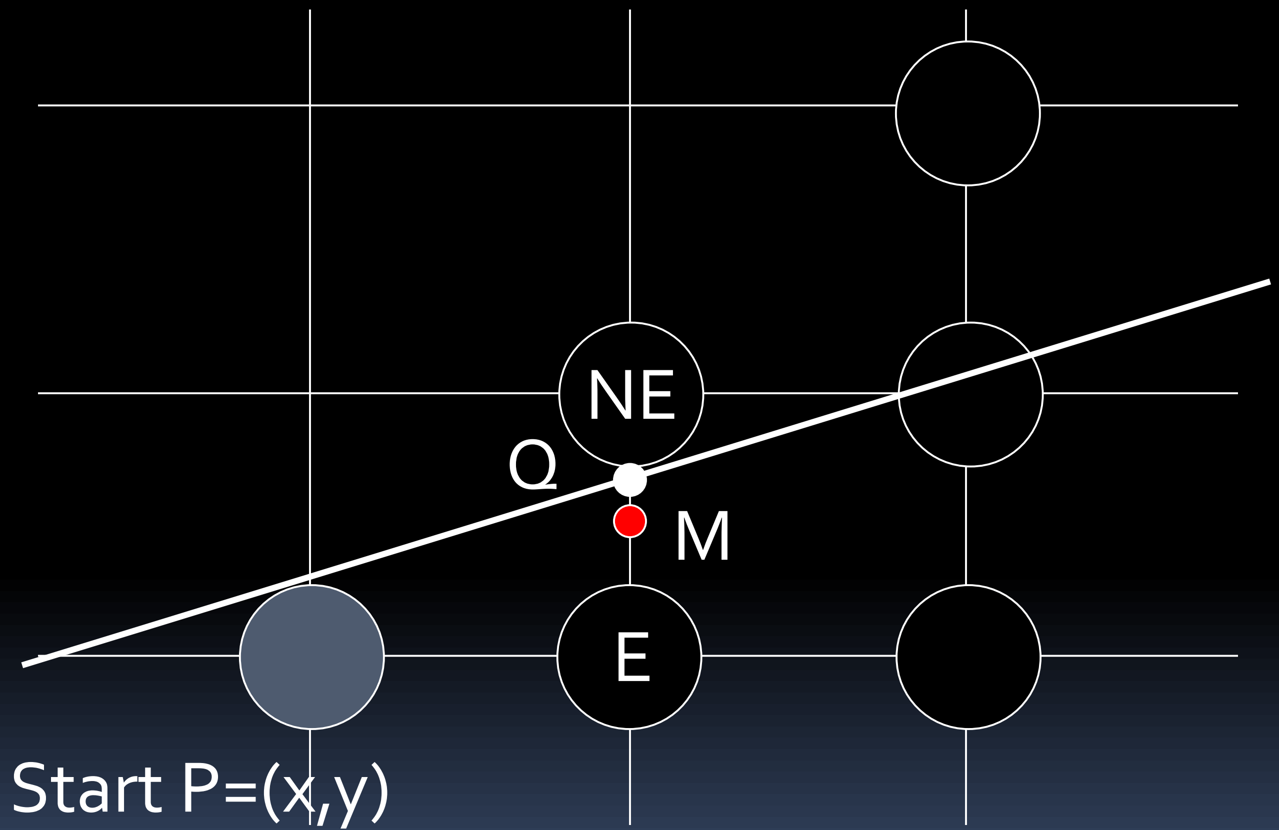
- Obviously the gradient does not change each time through the loop
- Calculate $r = dy/dx = (y2-y1)/x2-x1)$ **once**
- Note that
  - $y(x) = y1+r*(x-x1)$
  - $y(x+1) = y1+r*((x+1)-x1)$
  - $y(x+1)-y(x) = r$

…gives us

```
r = (y2-y1)/(x2-x1)
y=y1
for (x=x1;x<x2;x++) {
    y += r
    setPixel(x,round(y),colour)
}
```

- Problems?
  - the round which is expensive
  - floating point math is relatively expensive

Midpoint Algorithm



If Q <= M, choose East. If Q > M, choose NorthEast

Implicit Form of a Line

Implicit form — Explicit form

$$ax + by + c = 0 \qquad y = \frac{dy}{dx}x + B$$

$$dy\;x - dx\;y + B\;dx = 0$$

$$a = dy \quad b = -dx \quad c = B\;dx$$

Positive below the line
Negative above the line
Zero on the line

Decision Function

$$d = F(x, y) = a\,x + b\,y + c$$

$$d = F(x+1, y+\frac{1}{2}) = a\,(x+1) + b\,(y+\frac{1}{2}) + c$$

Choose NE if d > 0
Choose E if d <= 0

Incrementing d

If choosing E:

$$d_{new} = F(x+2, y+\tfrac{1}{2}) = a(x+2) + b(y+\tfrac{1}{2}) + c$$

But:

$$d_{old} = F(x+1, y+\tfrac{1}{2}) = a(x+1) + b(y+\tfrac{1}{2}) + c$$

So:

$$d_{inc} = d_{new} - d_{old} = a = \Delta E$$

Incrementing d

If choosing NE:

$$d_{new} = F(x+2, y+\tfrac{3}{2}) = a(x+2) + b(y+\tfrac{3}{2}) + c$$

But:

$$d_{old} = F(x+1, y+\tfrac{1}{2}) = a(x+1) + b(y+\tfrac{1}{2}) + c$$

So:

$$d_{inc} = d_{new} - d_{old} = a + b = \Delta NE$$

Initializing d

$$d = F(x_0 + 1, y_0 + \frac{1}{2}) = a\,(x_0 + 1) + b\,(y_0 + \frac{1}{2}) + c$$

$$= a\,x_0 + b\,y_0 + c + a + b\frac{1}{2}$$

$$= a + b\frac{1}{2}$$

Multiply everything by 2 to remove fractions (doesn't change the sign)

Midpoint Algorithm

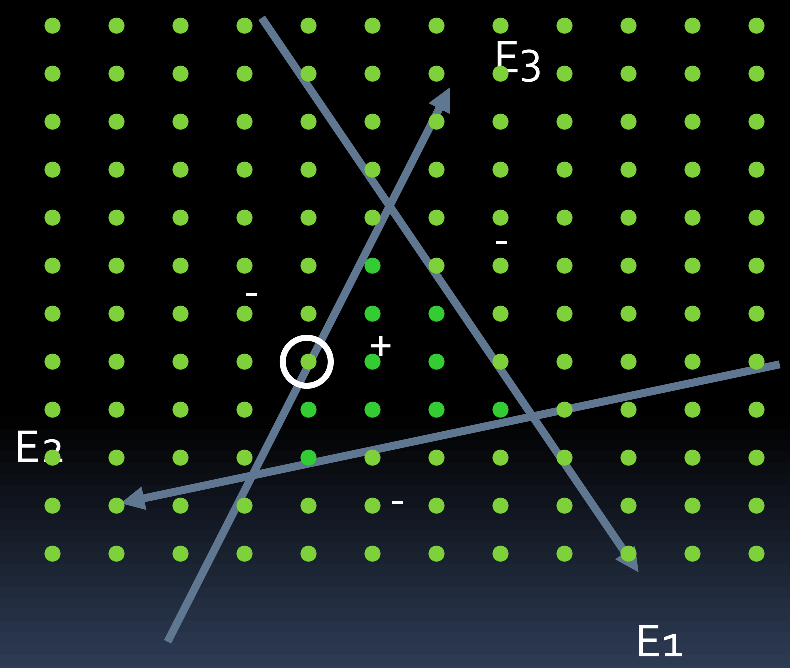
Assume 0 < m < 1, x0 < x1

```
Line(int x0, int y0, int x1, int y1)
  int dx = x1 – x0, dy = y1 – y0;
  int d = 2*dy-dx;
  int delE = 2*dy, delNE = 2*(dy-dx);
  int x = x0, y = y0;
  setPixel(x,y);

  while(x < x1)
     if(d<=0)
        d += delE; x = x+1;
     else
        d += delNE; x = x+1; y = y+1;
     setPixel(x,y);
```

Only integer arithmetic

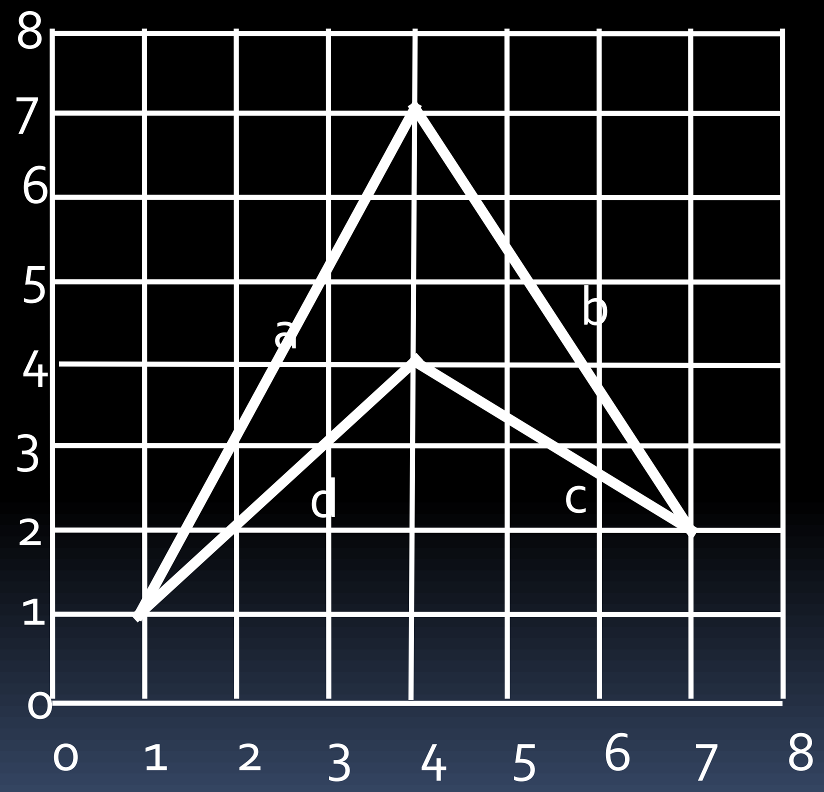
Polygon Fill



Active Edge Table

- For each scan-line in a polygon only certain edges need considering
- Keep an ***ACTIVE*** edge table
  - Update this edge table based upon the vertical extent of the edges
- From the AET extract the required spans

Example



Setting Up

- “fix” edges
  - make sure y1<y2 for each (x1,y1) (x2,y2)
- Form an ET
  - Bucket sort all edges on minimum y value
  - 1 bucket might contain several edges
  - Each edge element contains
    - (max Y, start X, X increment)

Setup

- Edges are

Edge Label	Coordinates	y1	Structure
a	(1,1) to (4,7)	1	(7,1,0.5)
b	(7,2) to (4,7)	2	(7,7,-0.6)
c	(7,2) to (4,4)	2	(4,7,-1.5)
d	(1,1) to (4,4)	1	(4,1,1)

- Edge Table Contains

y1	Sequence of Edges
1	(7,1,0.5), (4, 1, 1)
2	(7,7,-0.6), (4, 7,-1.5)

Maintaining the AET

- For each scan line
  - Remove all edges whose y2 is equal to current line
  - Update the x value for each remaining edge
  - Add all edges whose y1 is equal to current line

On Each Line

Line	Active Edge Table	Spans
0	empty	
1	(7,1,0.5), (4,1,1)	1 to 1
2	(7,1.5,0.5), (4,2,1), (7,7,-0.6), (4,7,-1.5)	1.5 to 2, 7 to 7
3	(7,2.0,0.5), (4,3,1), (4,5.5,-1.5), (7,6.4,-0.6)	2.0 to 3, 5.5 to 6.4
4	(7,2.5,0.5), (7,5.8,-0.6)	2.5 to 5.8
5	(7,3.0,0.5), (7,5.2,-0.6)	3.0 to 5.2
6	(7,3.5,0.5), (7,4.6,-0.6)	3.5 to 4.6
7	empty	
8	empty	

Drawing the AET

- Sort the active edges on x intersection
- Pairs of edges are the spans we require

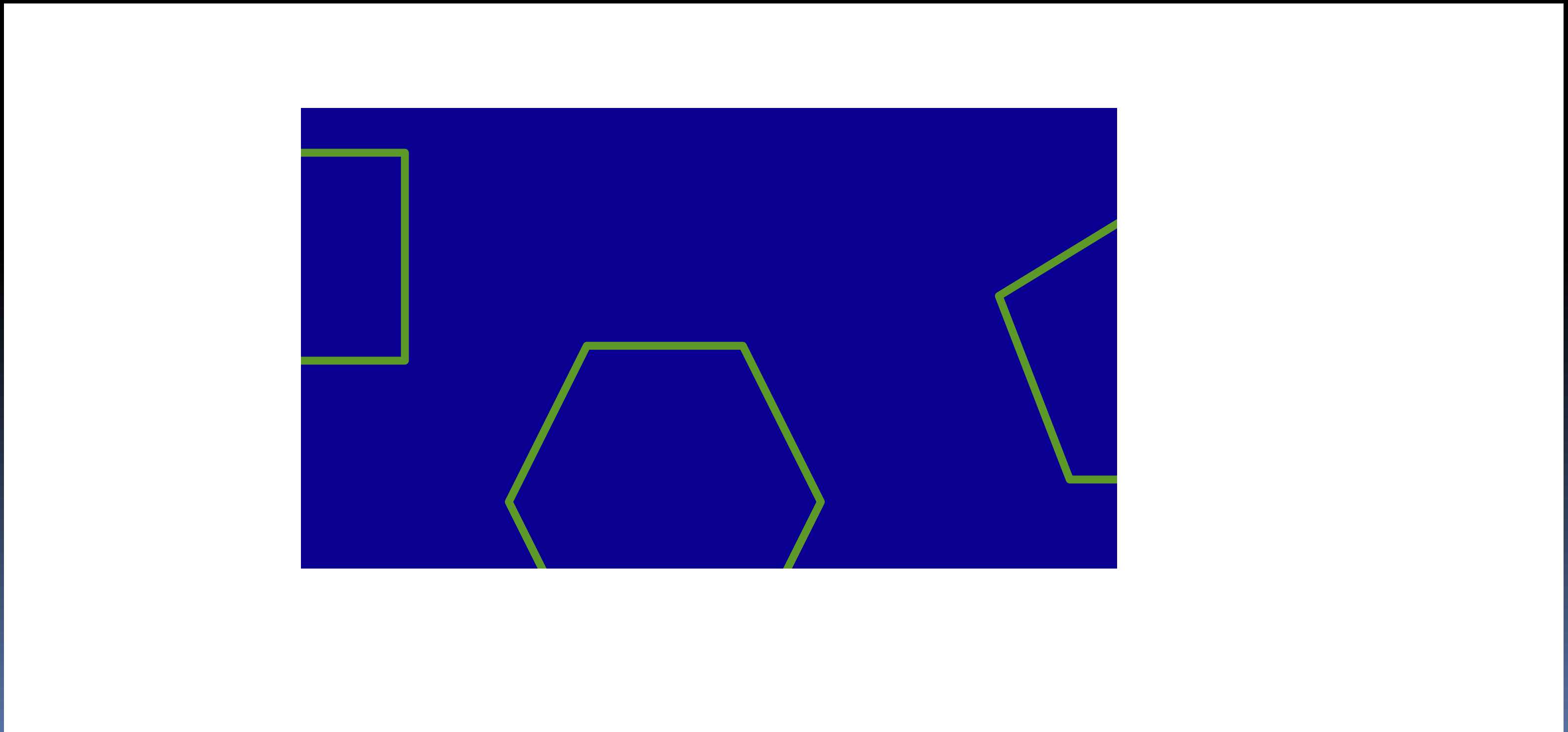
- Caveats (discussed in the notes)
  - Don’t consider horizontal lines
  - Maximum vertices are not drawn
  - Plenty of special cases when polygons share edges

On Each Line

Line	Active Edge Table	Spans
0	empty	
1	(7,1,0.5), (4,1,1)	1 to 1
2	(7,1.5,0.5), (4,2,1), (7,7,-0.6), (4,7,-1.5)	1.5 to 2, 7 to 7
3	(7,2.0,0.5), (4,3,1), (4,5.5,-1.5), (7,6.4,-0.6)	2.0 to 3, 5.5 to 6.4
4	(7,2.5,0.5), (7,5.8,-0.6)	2.5 to 5.8
5	(7,3.0,0.5), (7,5.2,-0.6)	3.0 to 5.2
6	(7,3.5,0.5), (7,4.6,-0.6)	3.5 to 4.6
7	empty	
8	empty	

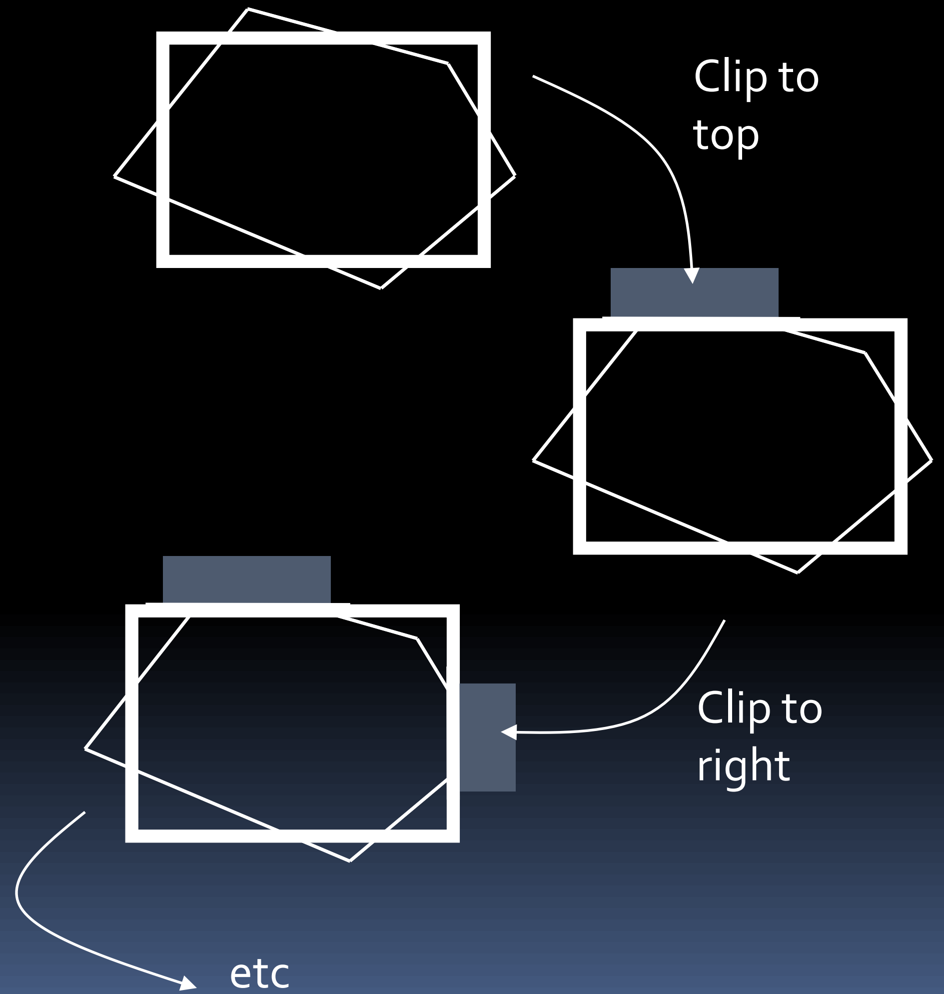
Clipping Problem (2D)

- Once we start projecting polygons we have to cope with cases where only some of the vertices project to the view window

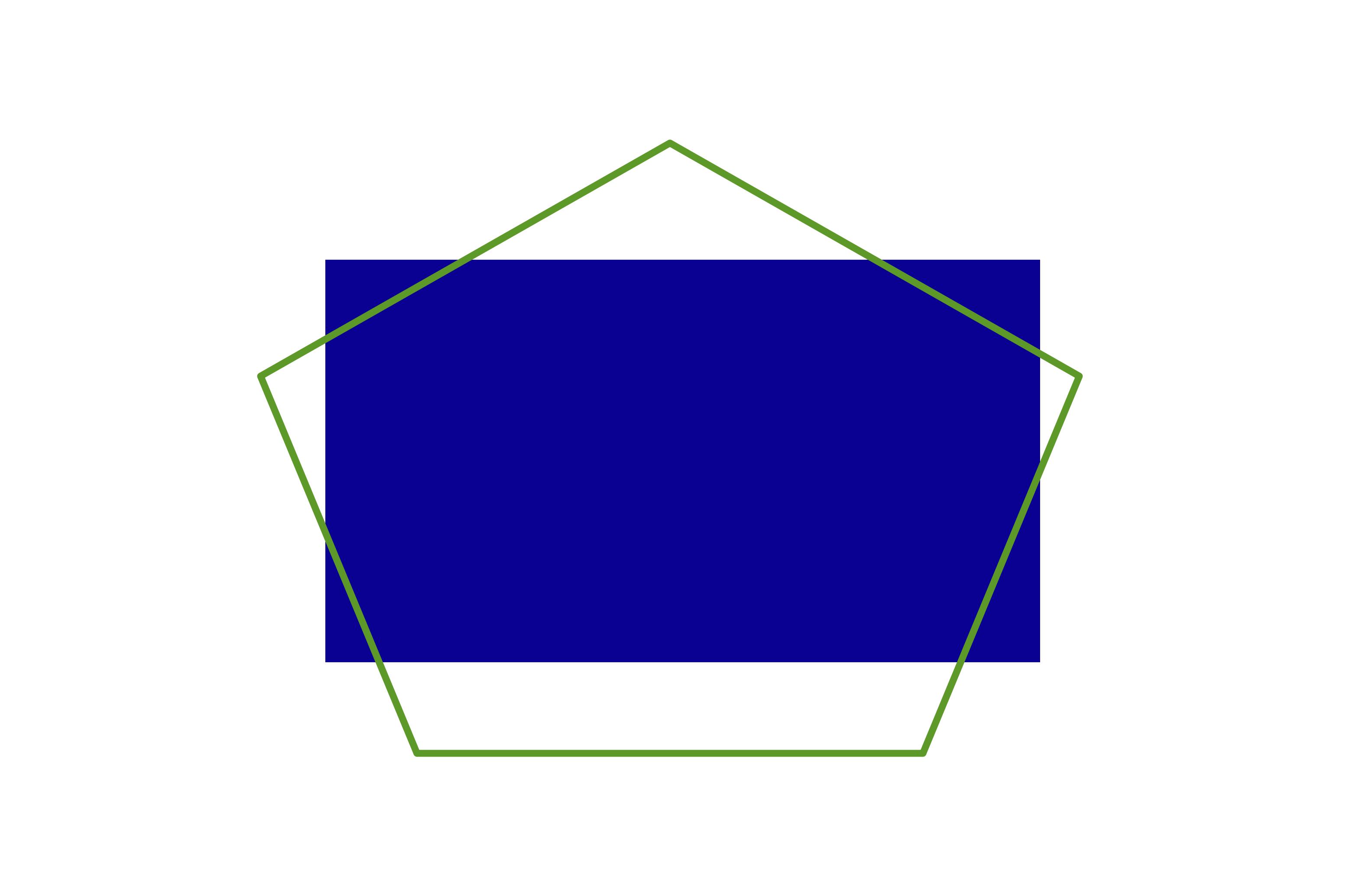


Sutherland-Hodgman Algorithm (2D)

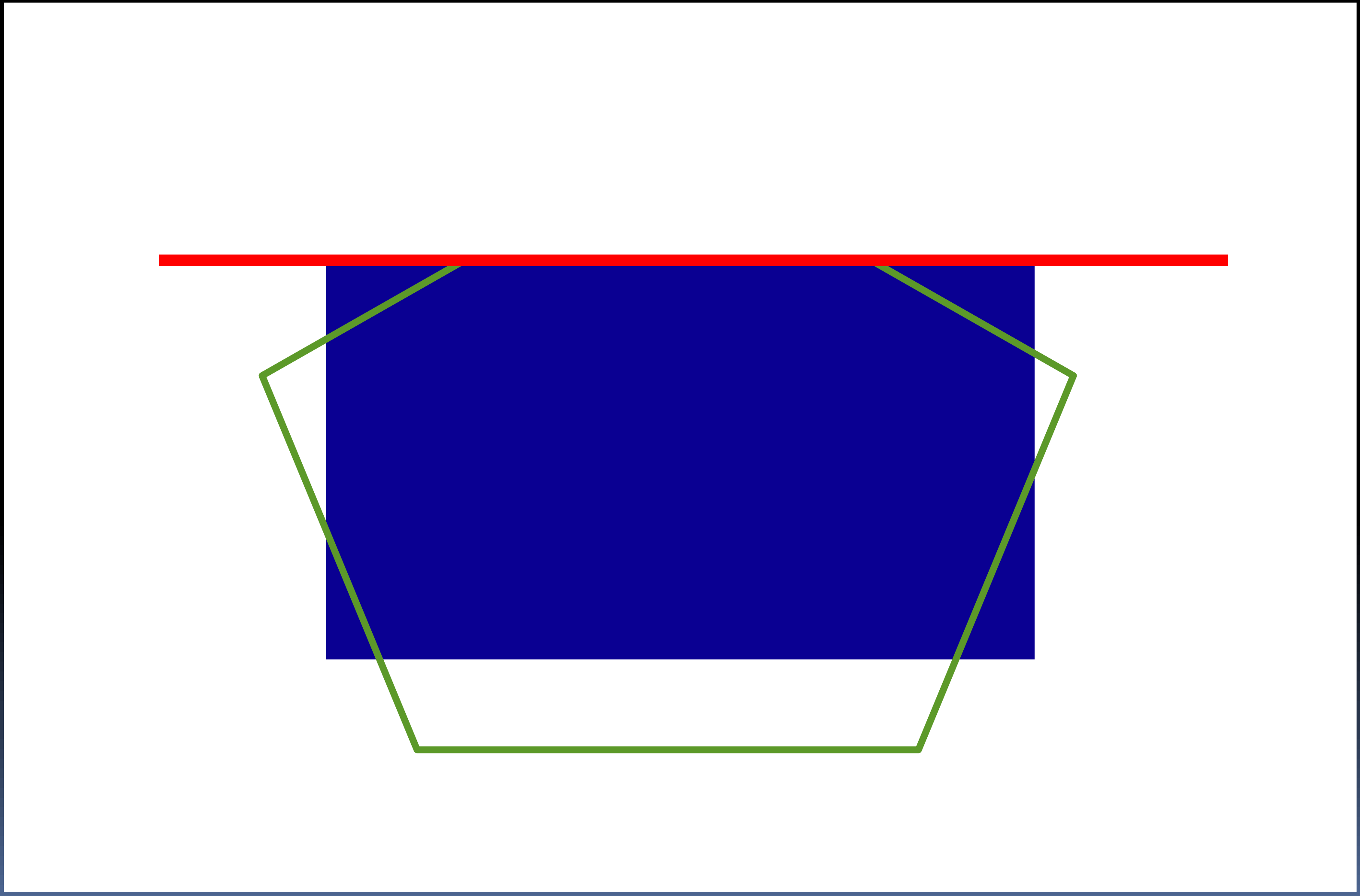
- Clip the polygon against each boundary of the clip region successively
- Result is possibly NUL if polygon is outside
- Can be generalised to work for any polygonal clip region, not just rectangular



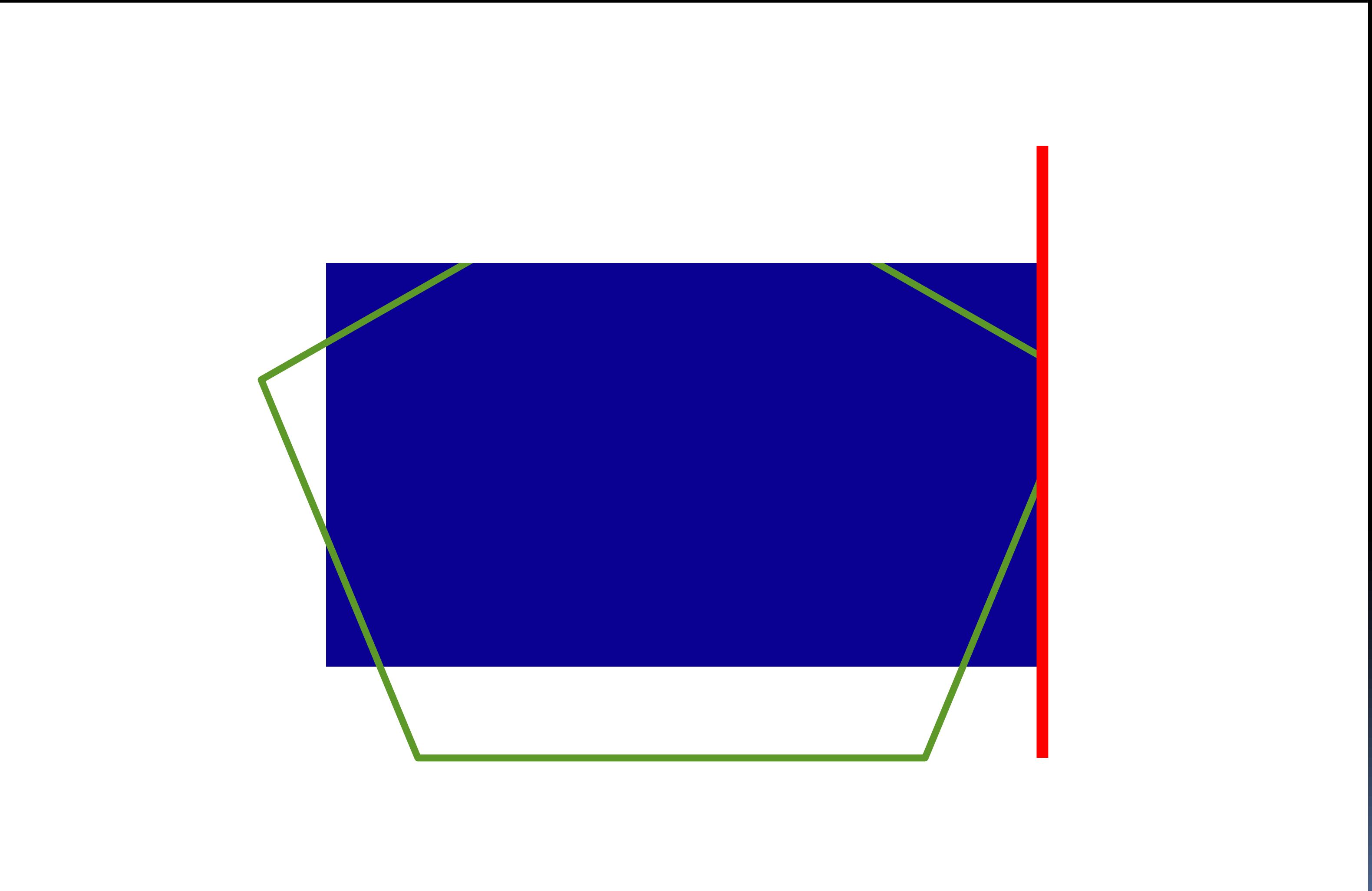
4 Passes to Clip



1st Pass

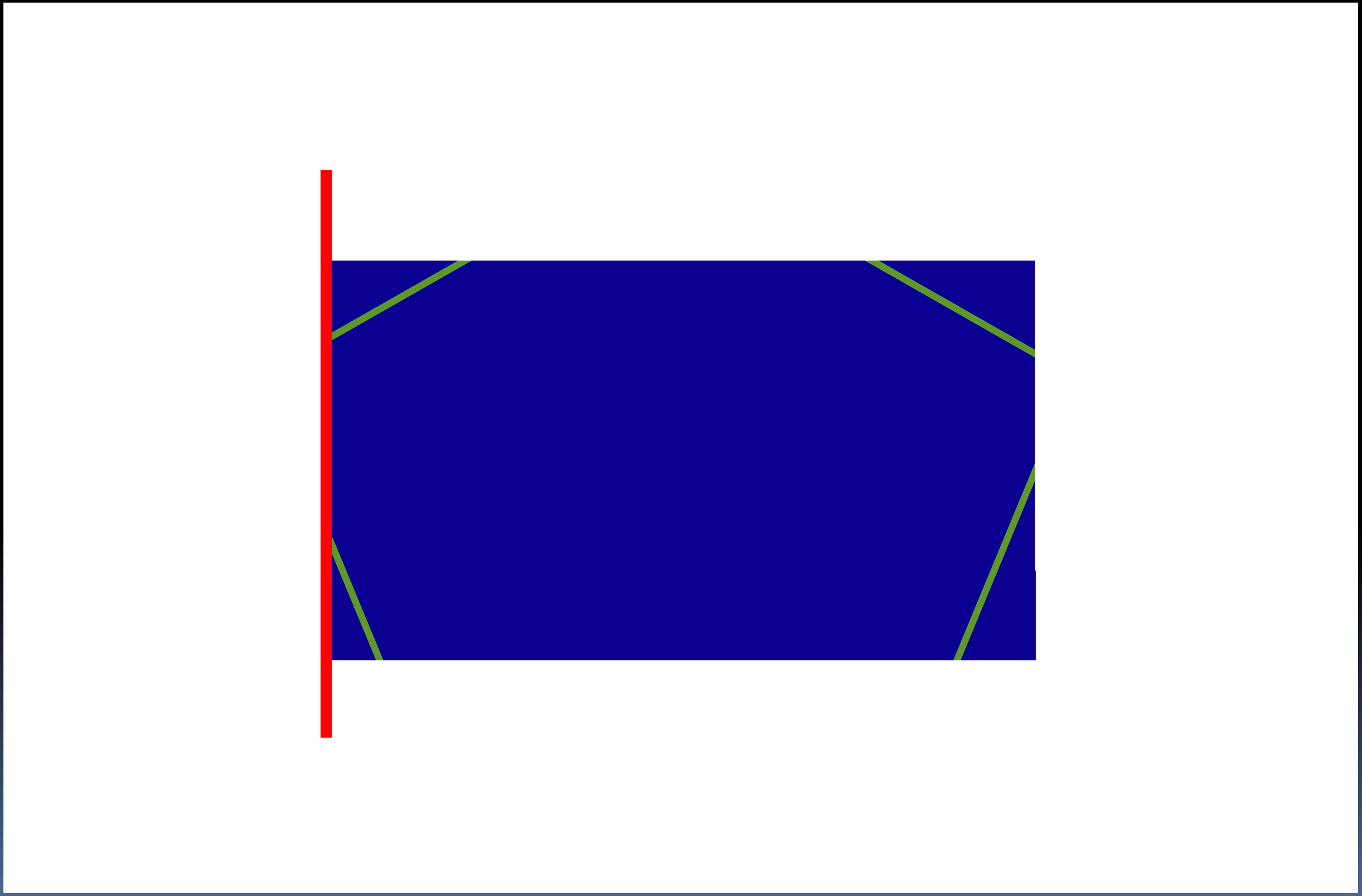


2nd Pass



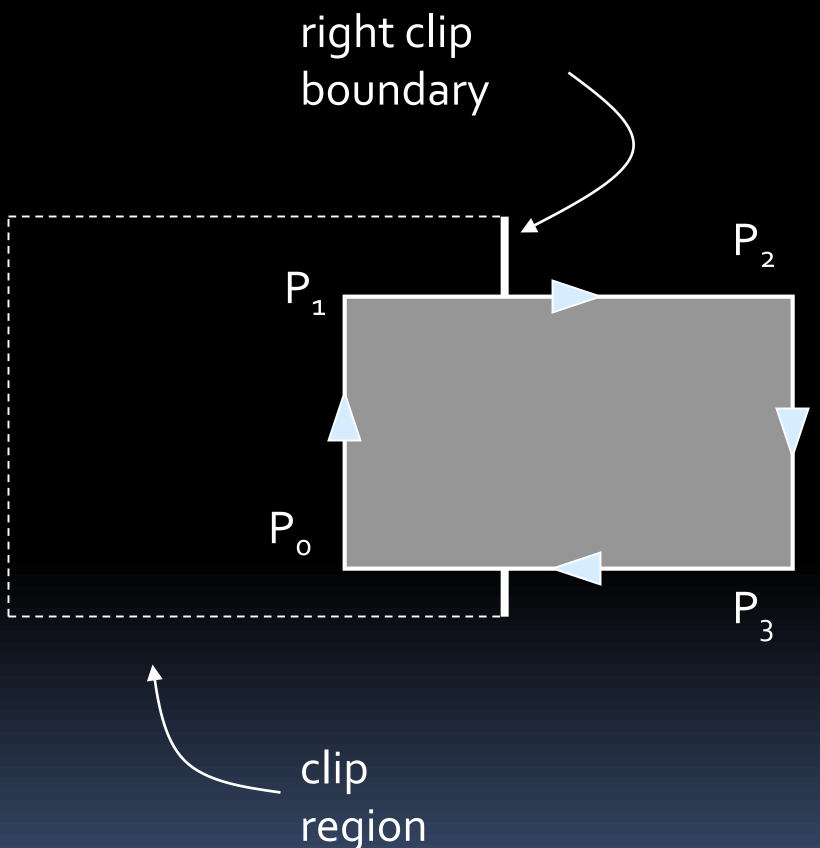
3rd Pass

4th Pass



Clipping to a Region Boundary

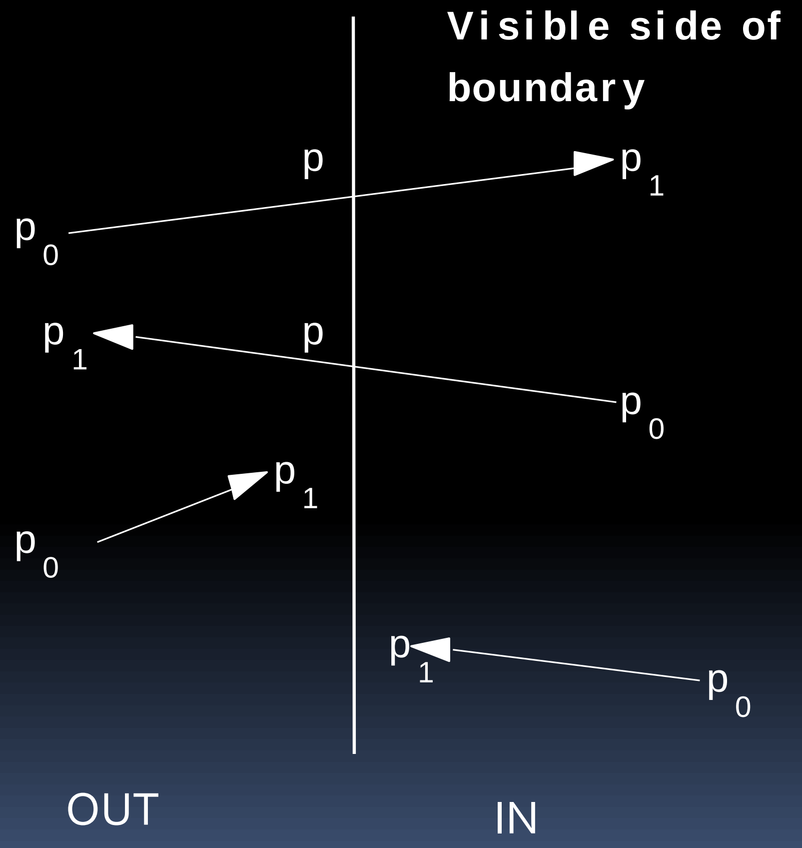
- To find the new polygon
  - iterate through each of the polygon edges and construct a new sequence of points
  - starting with an empty sequence
  - for each edge there are 4 possible cases to consider （see next page）



P0 → P1 → P3 → P4

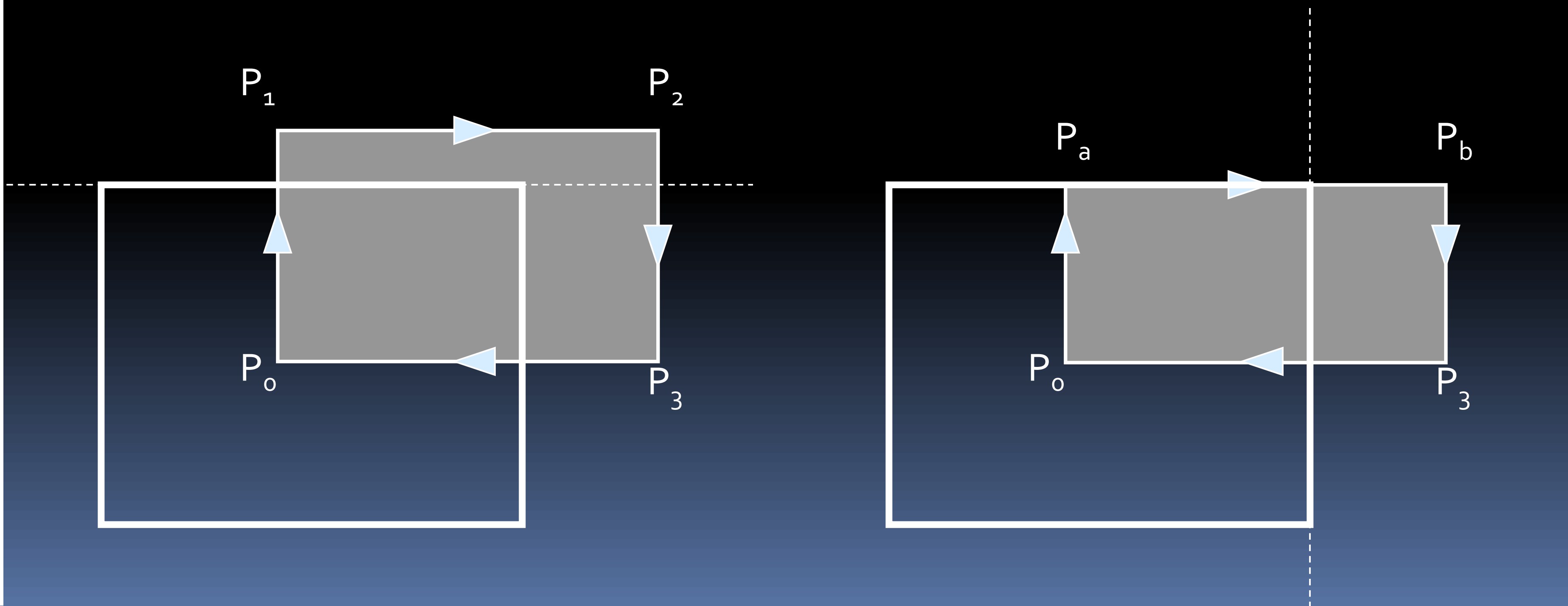
Clipping the Polygon Edge

- Given an edge P_0,P_1 we have 4 case. It can be:
  - entering the clip region, add *P* and P_1
  - leaving the region, add only *P*
  - entirely outside, do nothing
  - entirely inside, add only *P1*
- Where *P* is the point of intersection



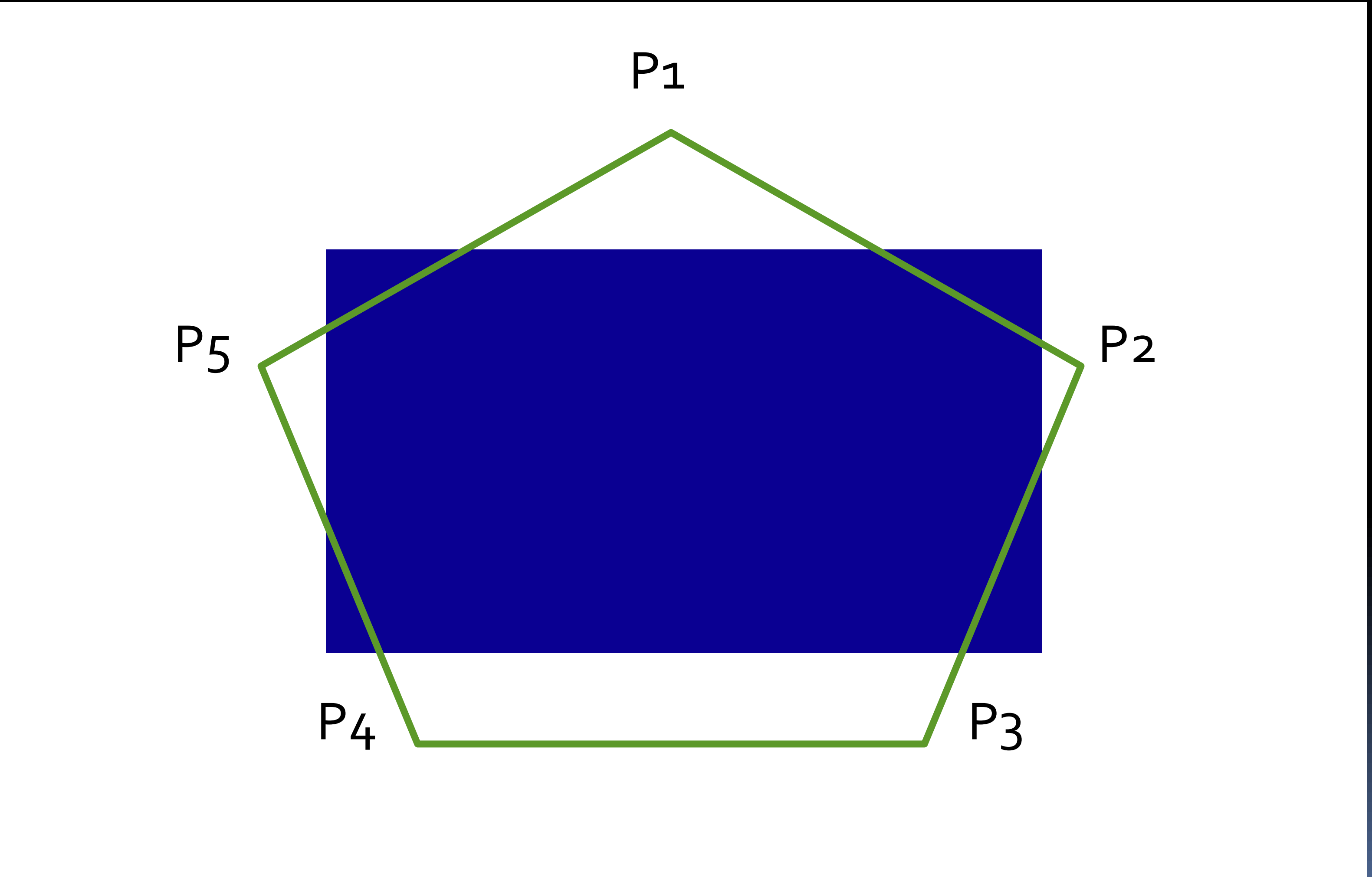
Wrapping Up

- We can determine which of the 4 cases and also the point of intersection with just if statements
- To sum it up, an example:



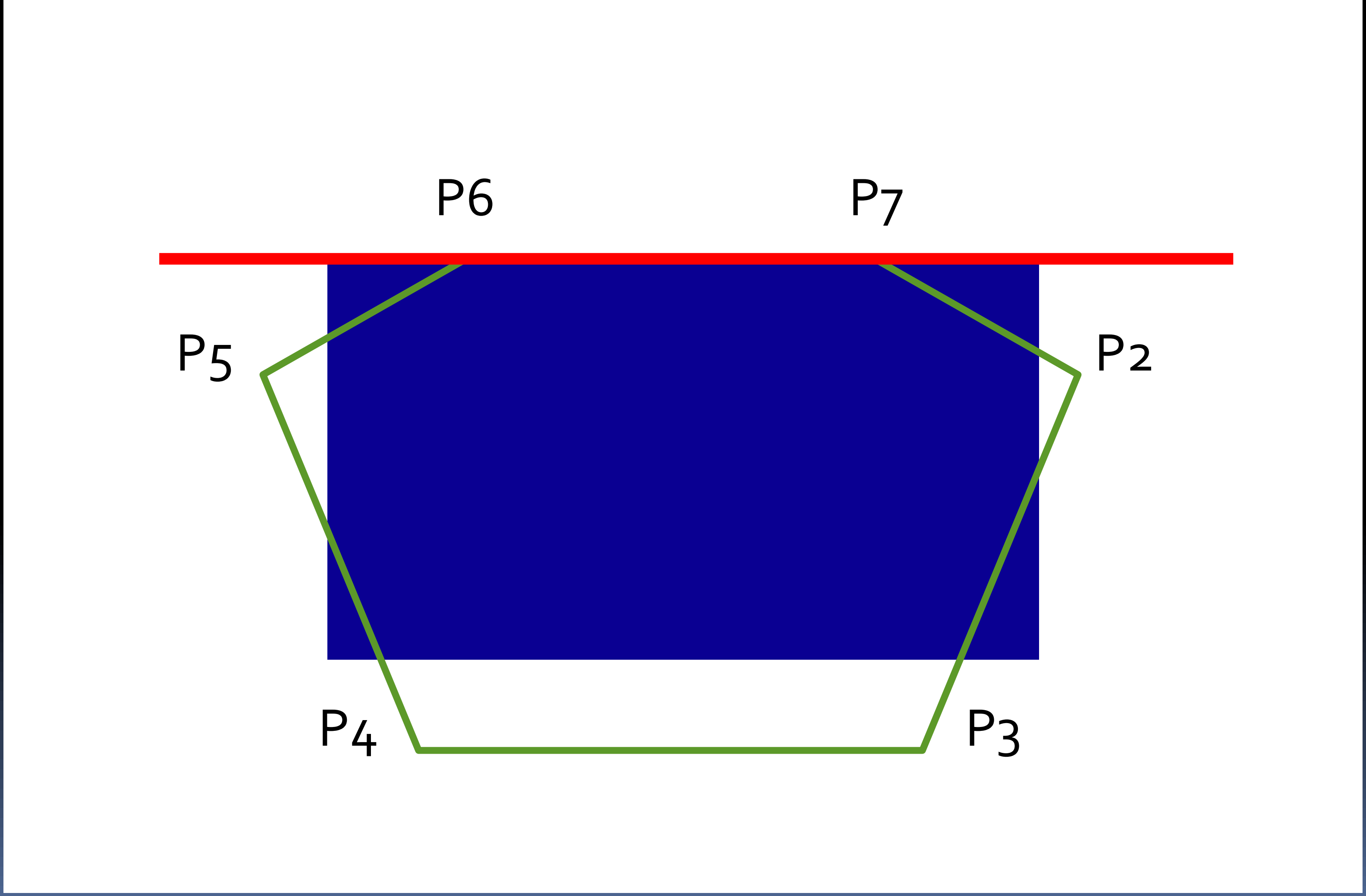
4 Passes to Clip

$P_1 \rightarrow P_2 \rightarrow P_3 \rightarrow P_4 \rightarrow P_5$



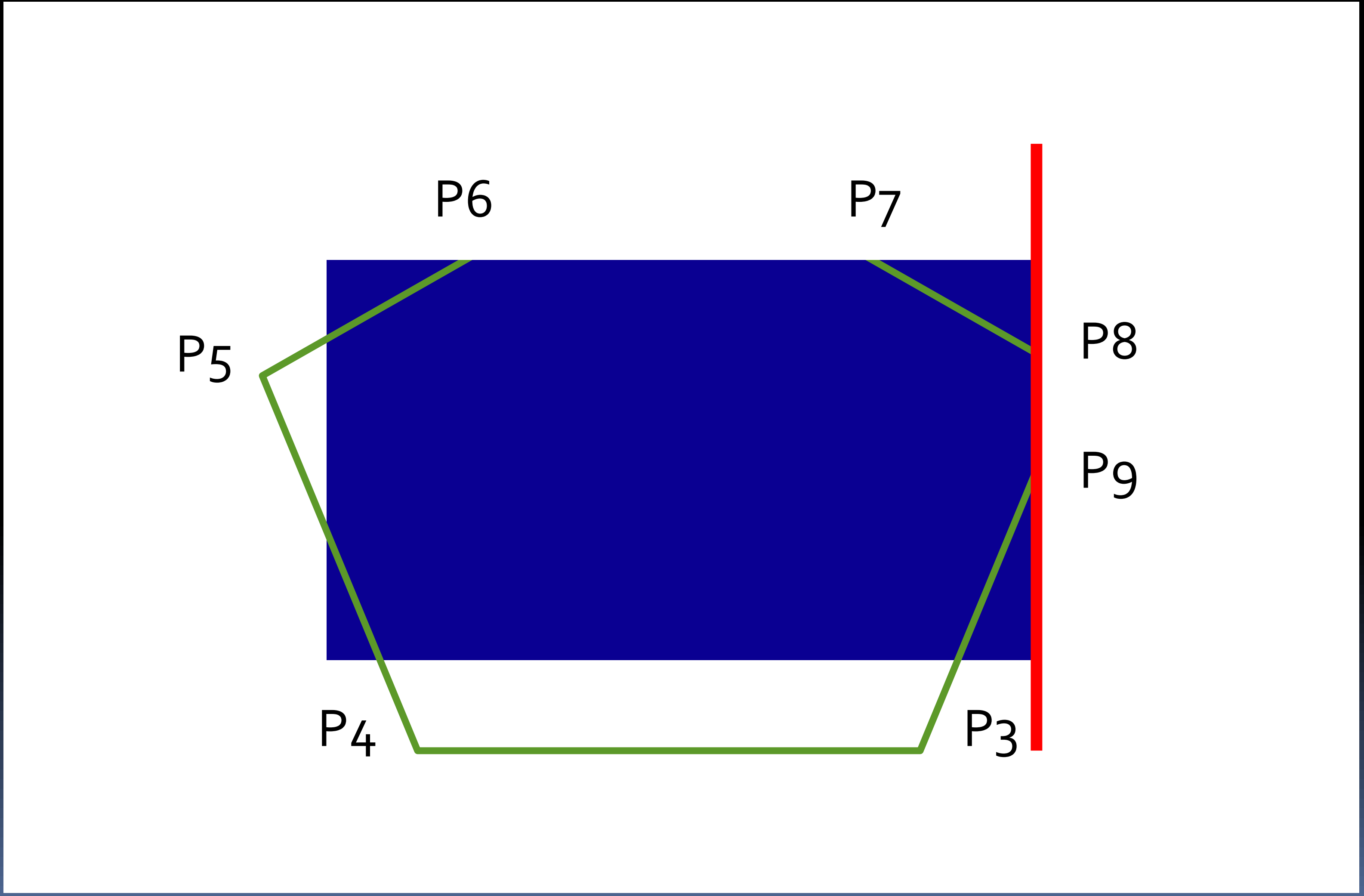
1st Pass

P6 → P7 → P2 → P3 → P4 → P5



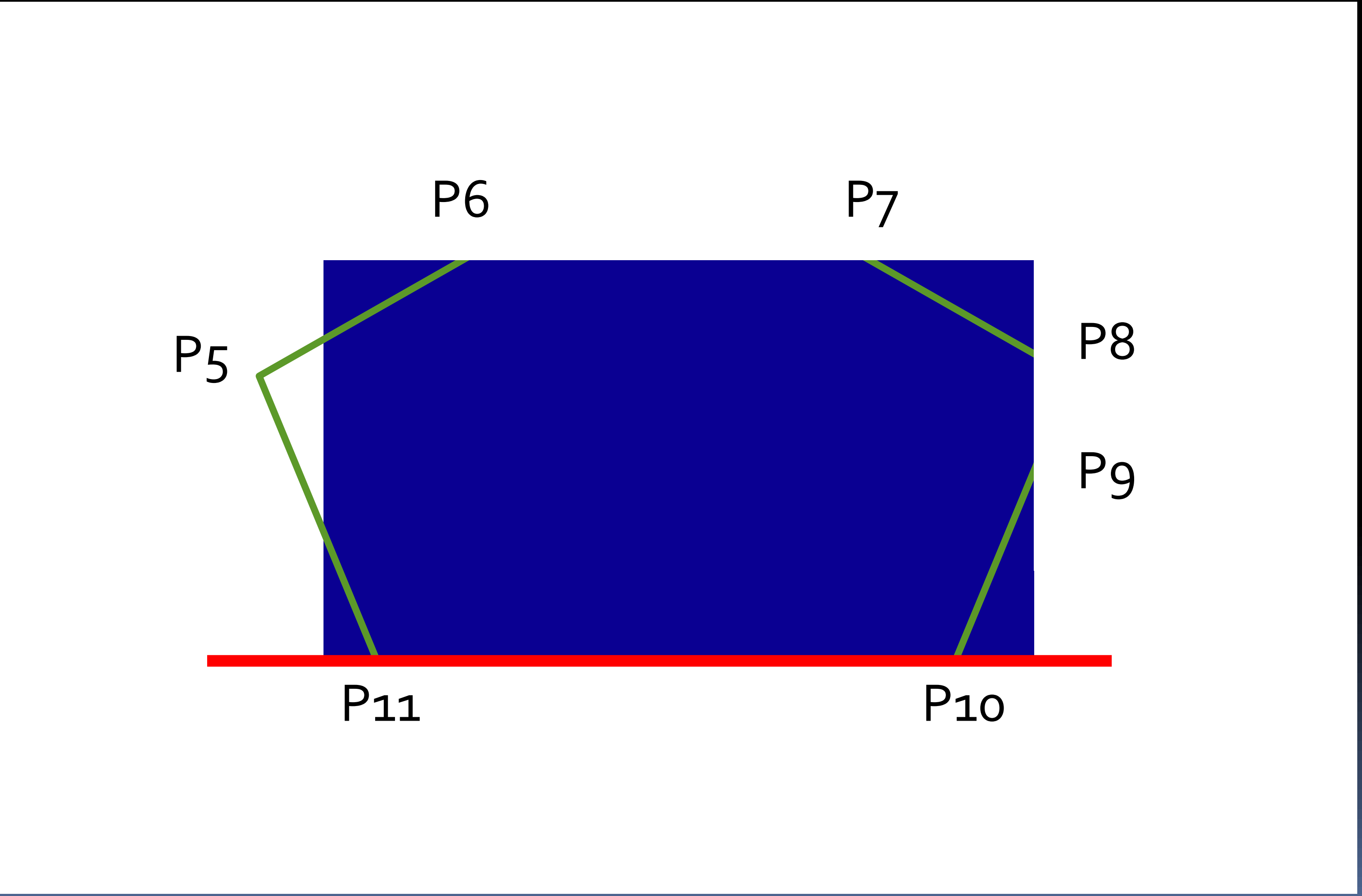
2nd Pass

P6 → P7 → P8 → P9 → P3 → P4 → P5



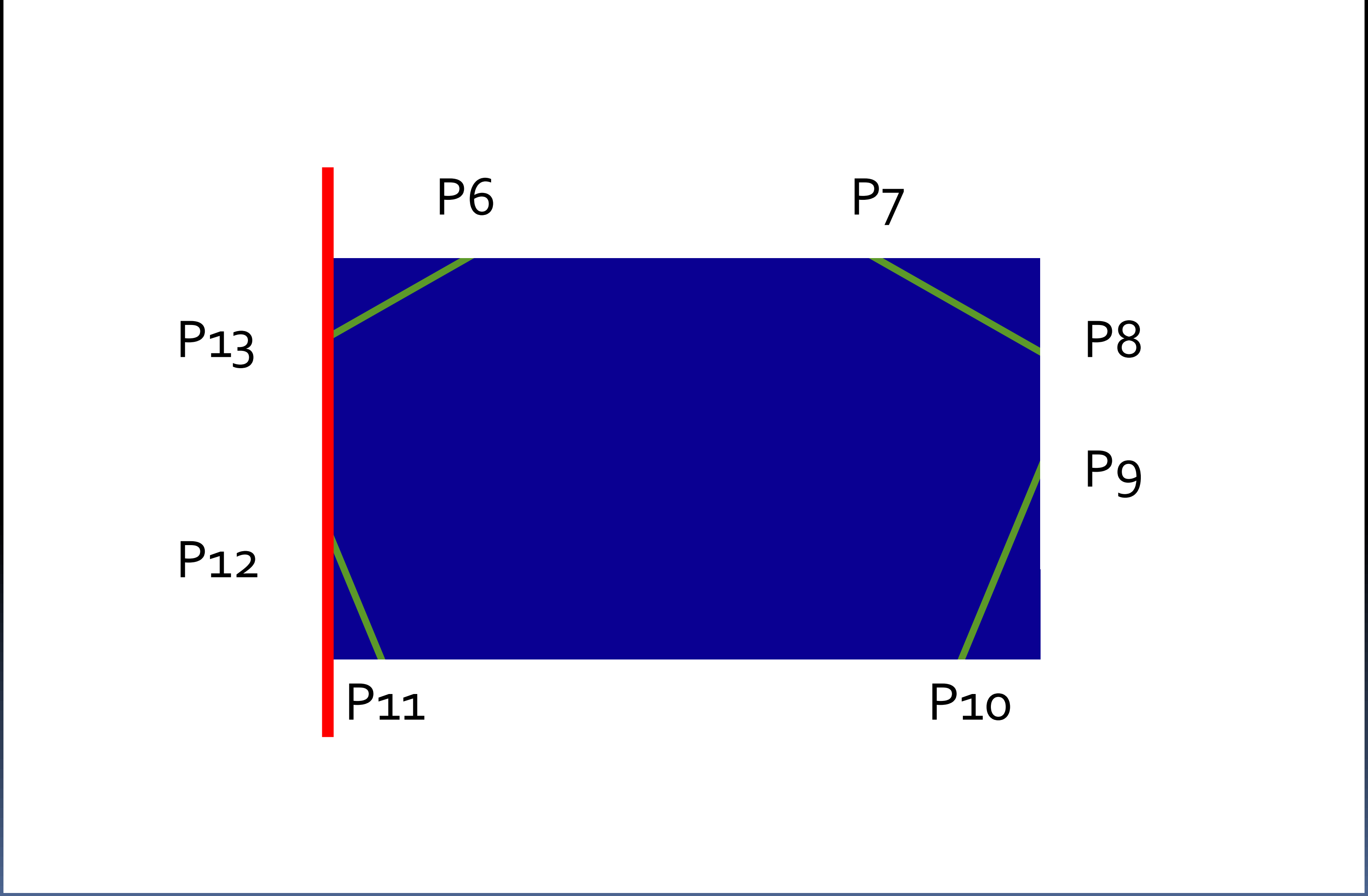
3rd Pass

P6 → P7 → P8 → P9 → P10 → P11 → P5



4th Pass

P6 → P7 → P8 → P9 → P10 → P11 → P12 → P13



P6 → P7 → P8 → P9 → P10 → P11 → P12 → P13