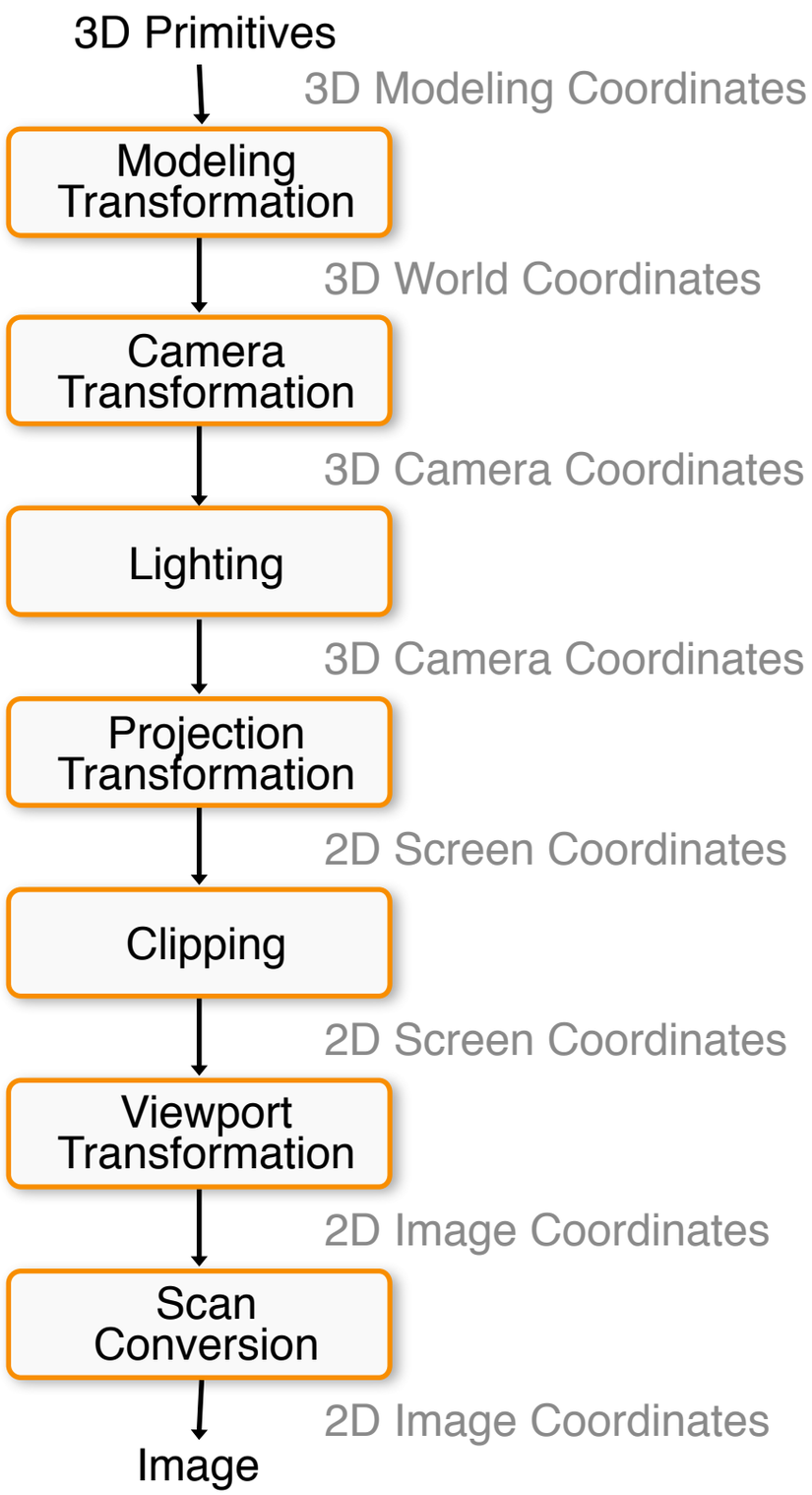
Clipping and Scan Conversion

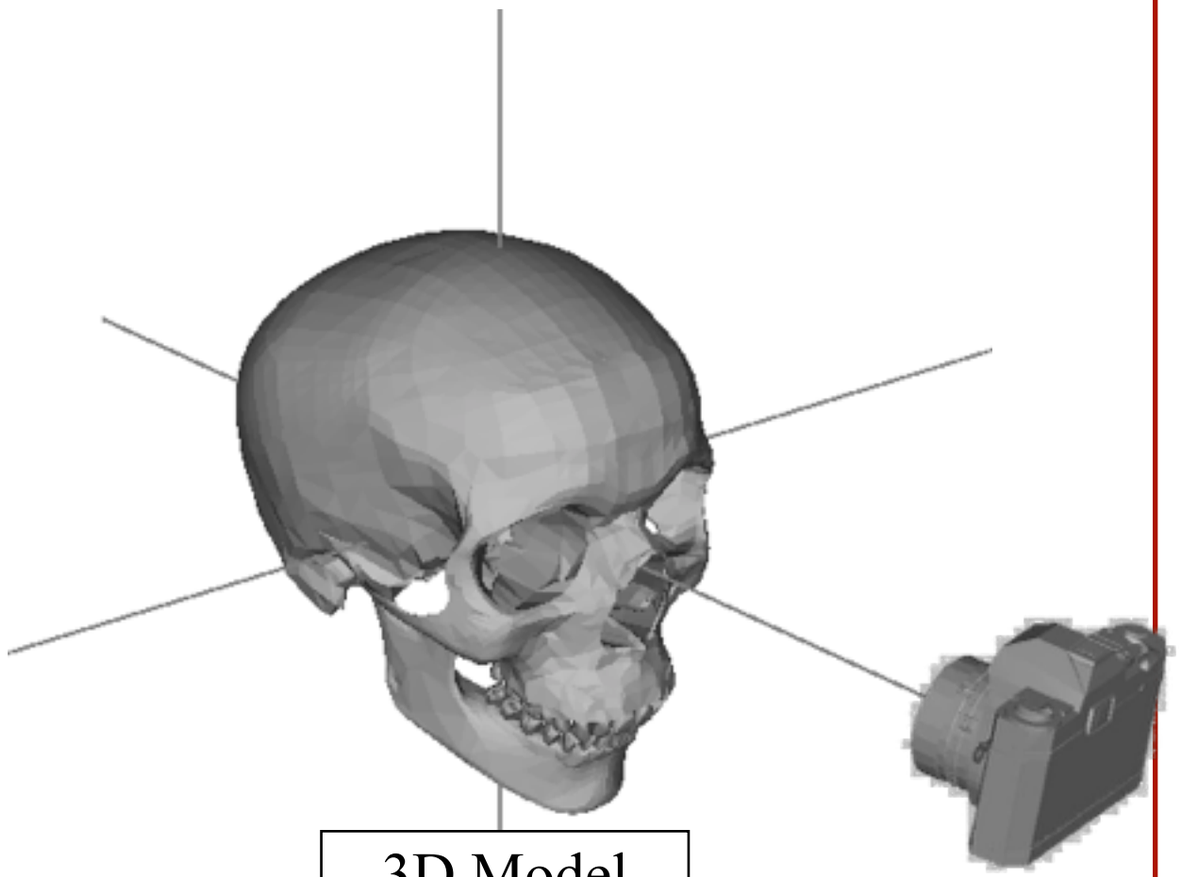
Connelly Barnes

CS 4810: Graphics

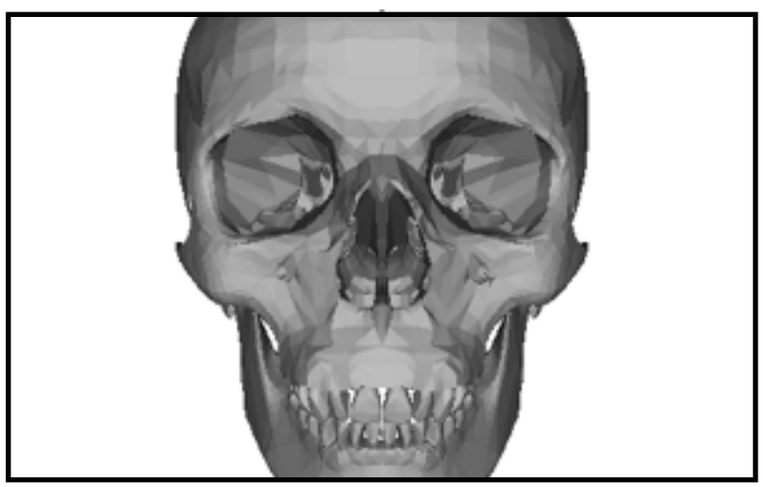
Acknowledgment: slides by Jason Lawrence, Misha Kazhdan, Allison Klein,
Tom Funkhouser, Adam Finkelstein and David Dobkin

3D Rendering Pipeline (for direct illumination)





3D Model



2D Image

Transformations

p(x,y,z)

3D Object Coordinates

Modeling Transformation

3D World Coordinates

Camera Transformation

3D Camera Coordinates

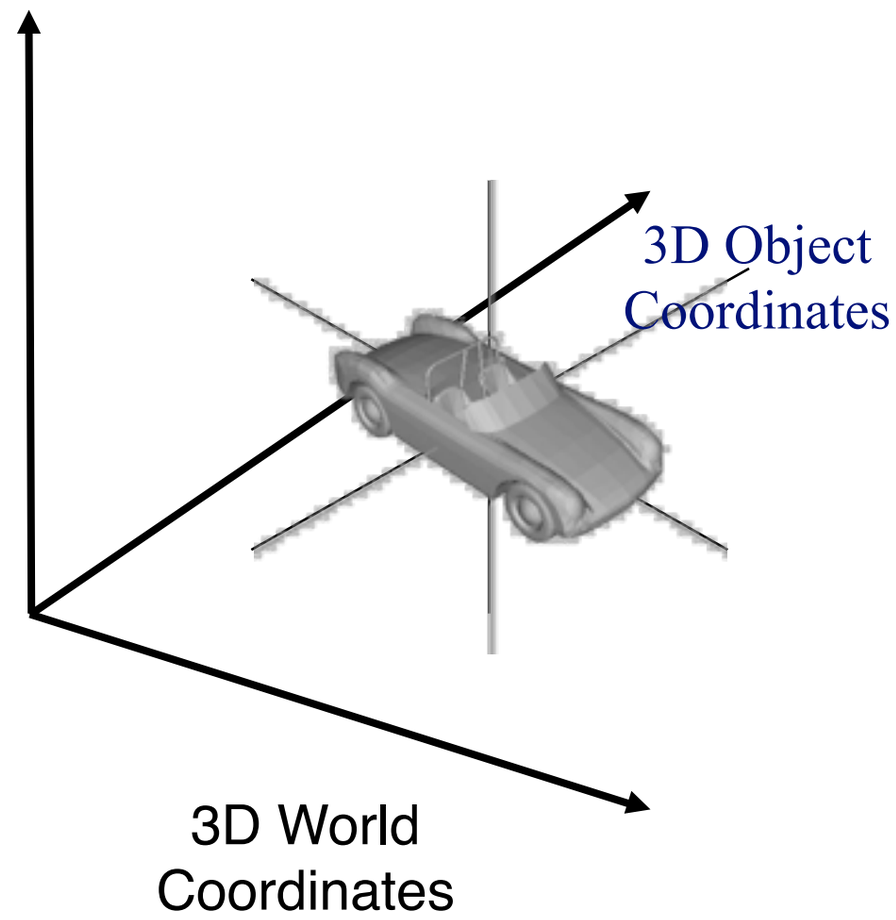
Projection Transformation

2D Screen Coordinates

Window-to-Viewport Transformation

2D Image Coordinates

p’(x’,y’)



Transformations

p(x,y,z)

3D Object Coordinates

Modeling Transformation

3D World Coordinates

Camera Transformation

3D Camera Coordinates

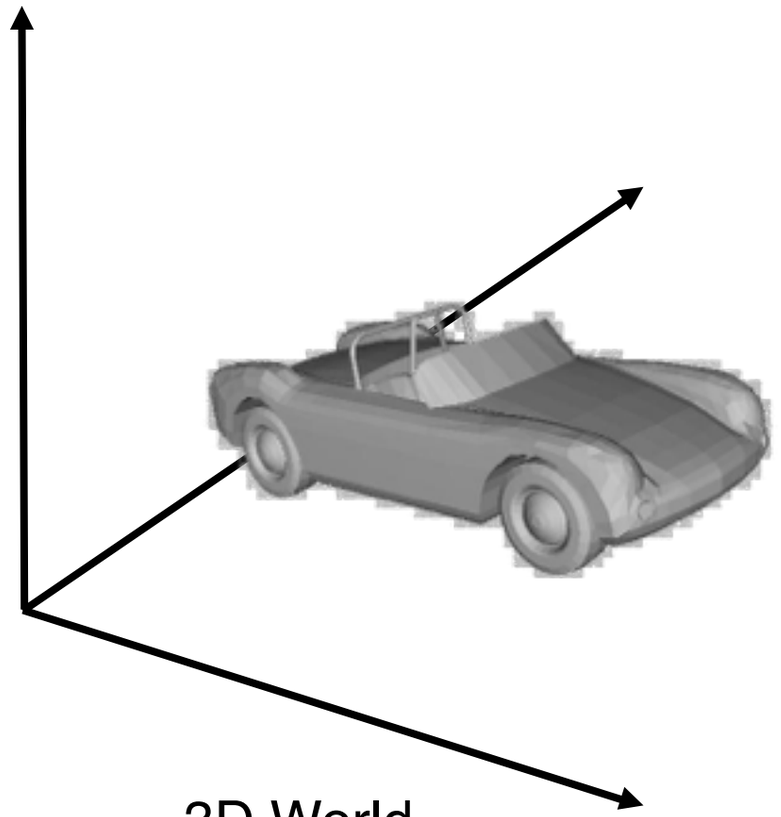
Projection Transformation

2D Screen Coordinates

Window-to-Viewport Transformation

2D Image Coordinates

p’(x’,y’)

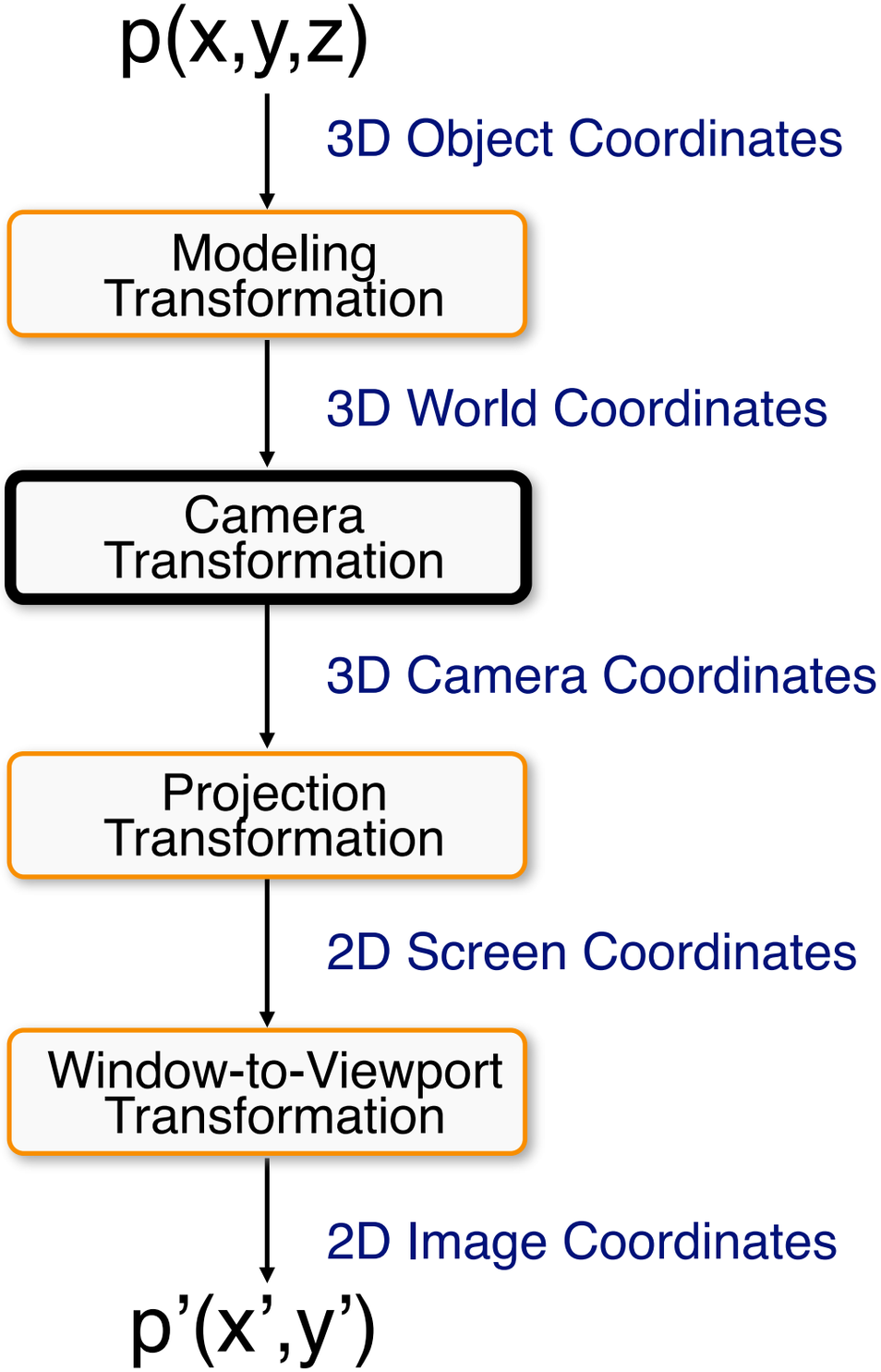


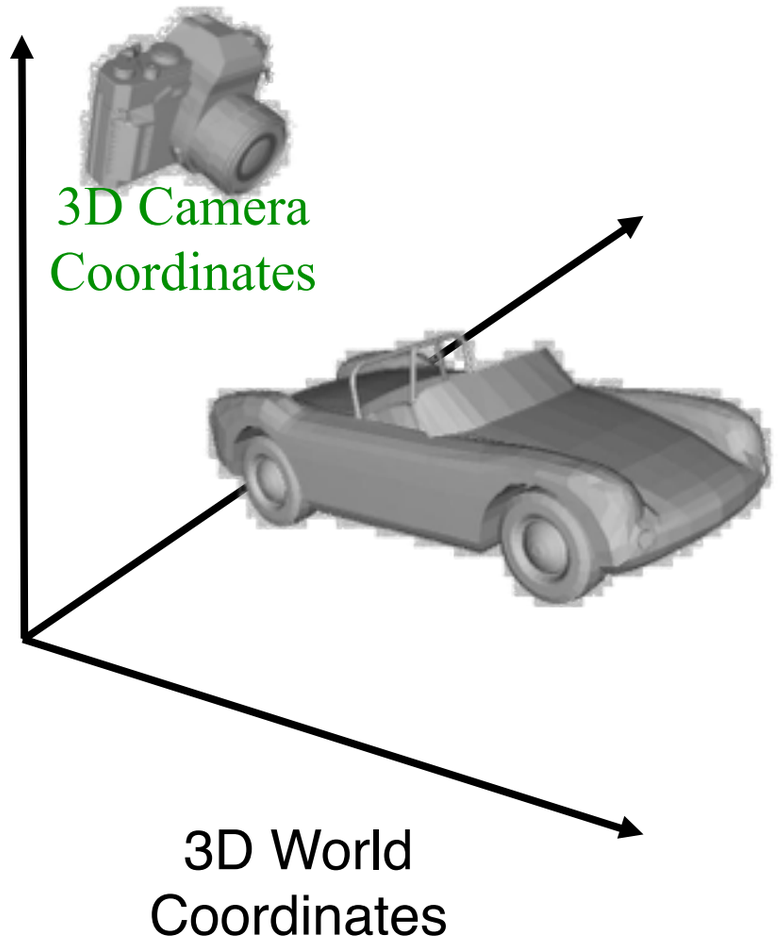
3D World Coordinates

Transform=M

M := local to world transform

Transformations





Transform=M

M := local to world transform

Transformations

p(x,y,z)

3D Object Coordinates

Modeling Transformation

3D World Coordinates

Camera Transformation

3D Camera Coordinates

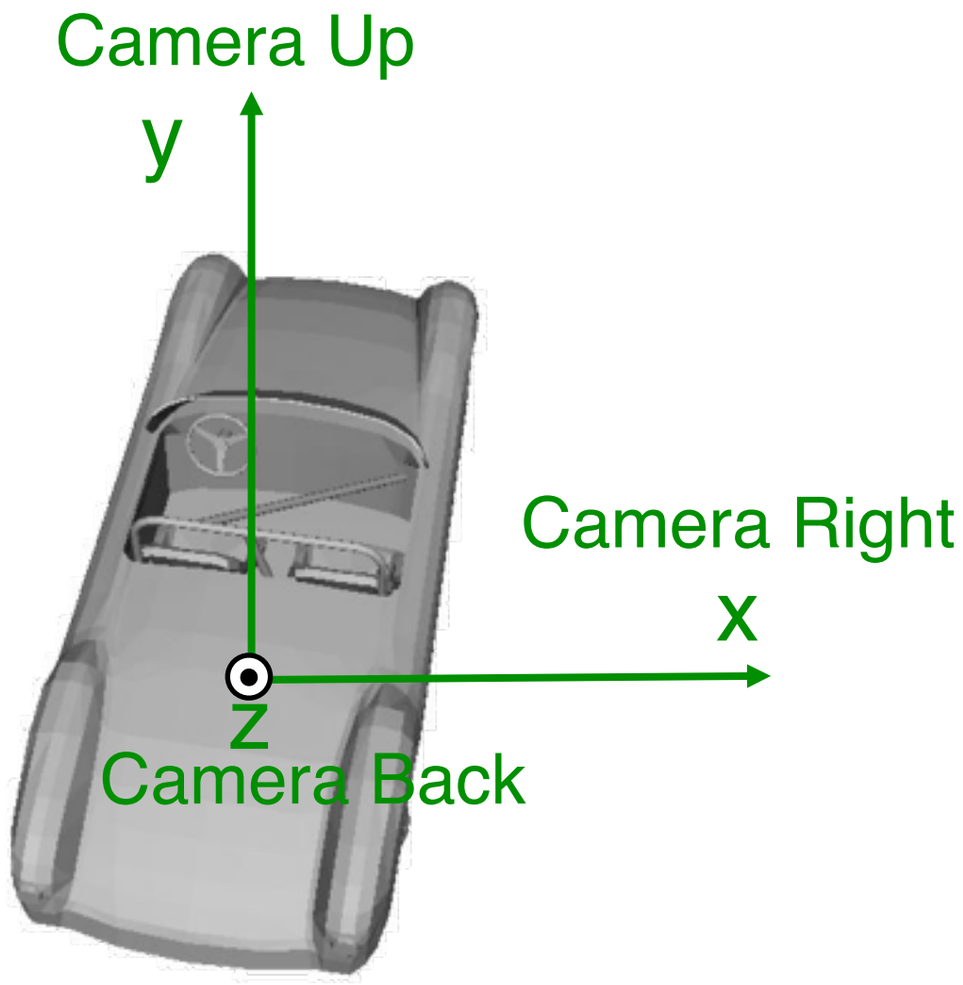
Projection Transformation

2D Screen Coordinates

Window-to-Viewport Transformation

2D Image Coordinates

p’(x’,y’)

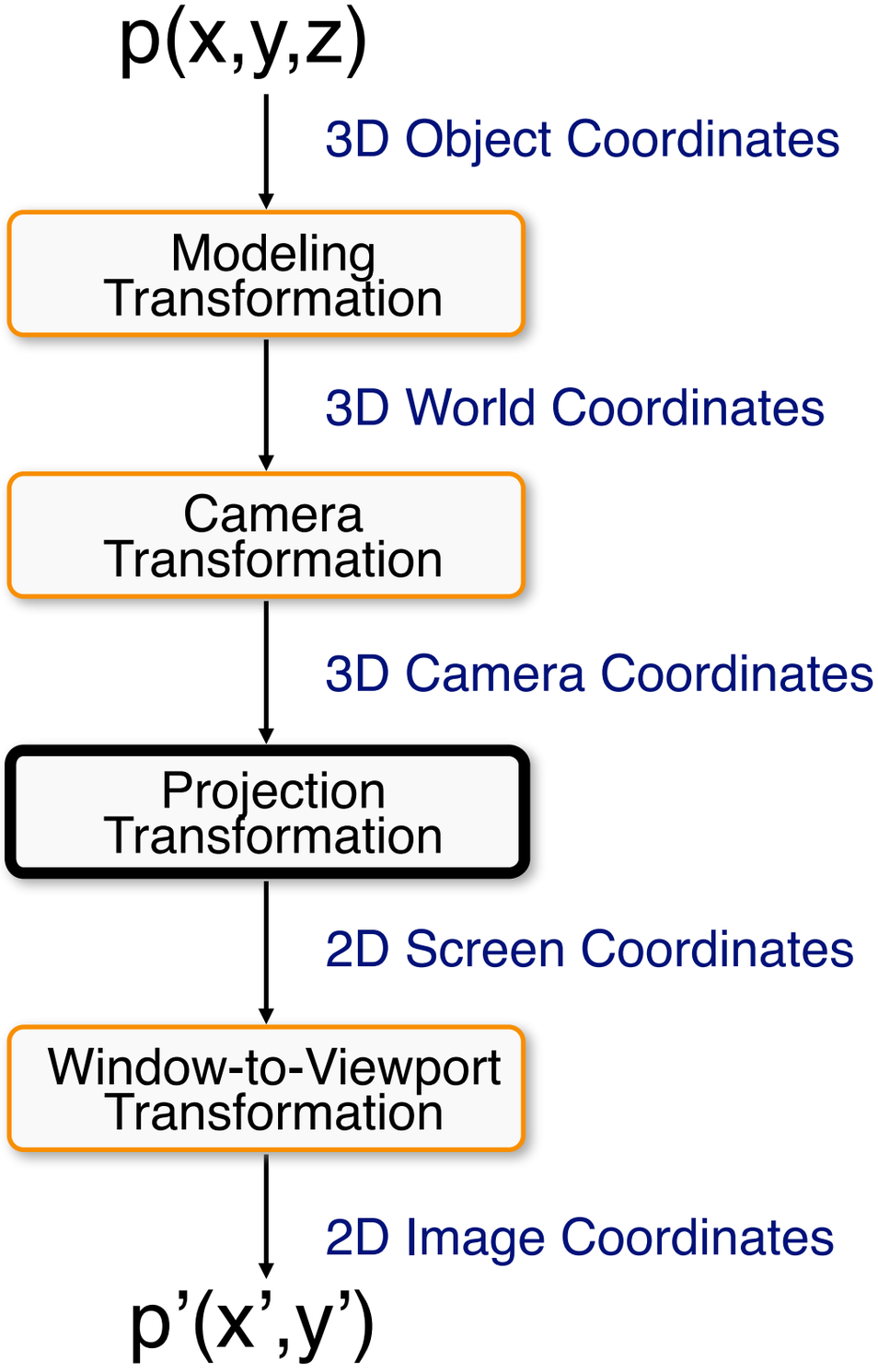


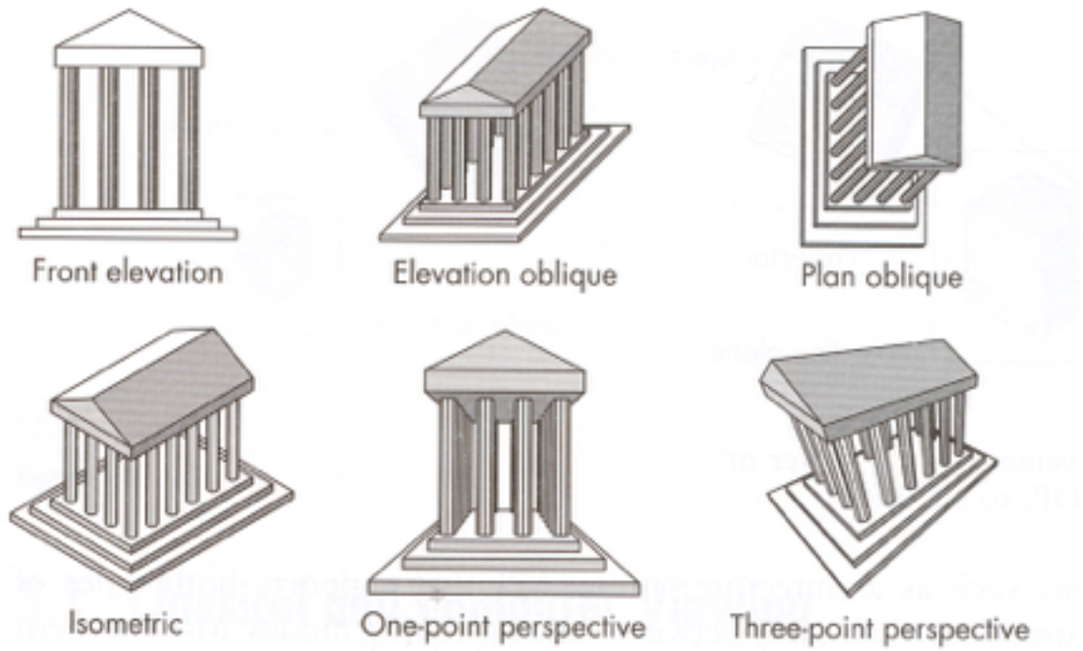
$$\text{Transform} = C^{-1}M$$

C := camera transform

$$C = \begin{bmatrix} R_x & U_x & B_x & E_x \\ R_y & U_y & B_y & E_y \\ R_z & U_z & B_z & E_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Transformations





$$\text{Transform} = PC^{-1}M$$

P := projection transform

$$P_o = \begin{bmatrix} 1 & 0 & L\cos\phi & 0 \\ 0 & 1 & L\sin\phi & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad P_p = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{D} & 0 \end{bmatrix}$$

Transformations

p(x,y,z)

3D Object Coordinates

Modeling Transformation

3D World Coordinates

Camera Transformation

3D Camera Coordinates

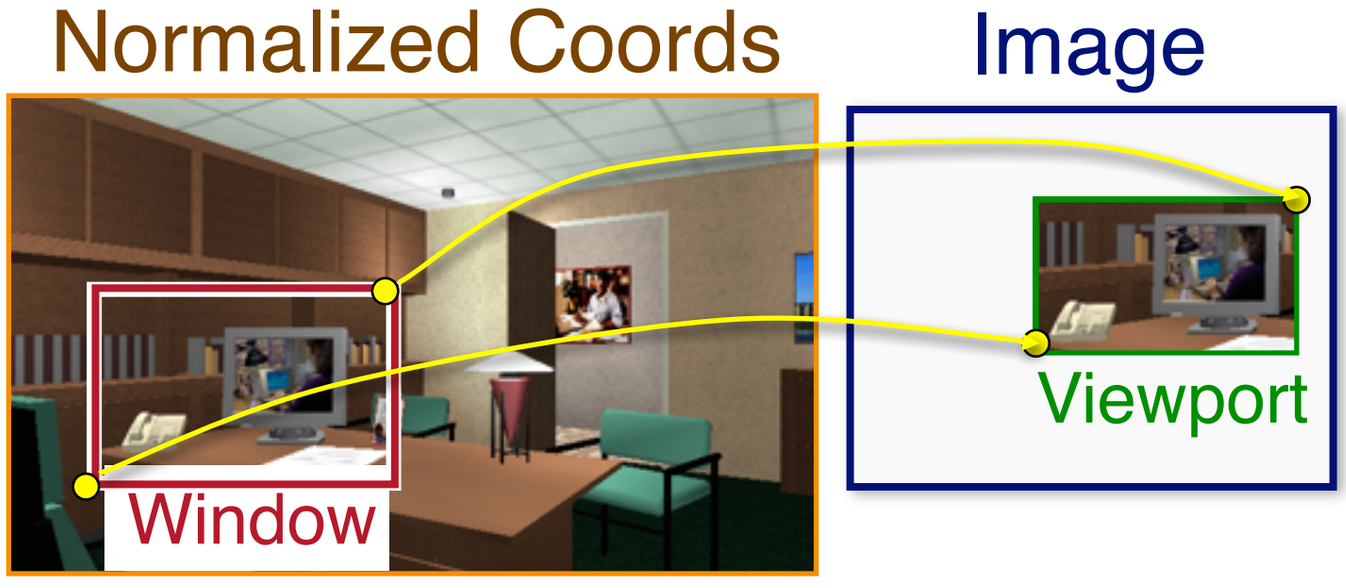
Projection Transformation

2D Screen Coordinates

Window-to-Viewport Transformation

2D Image Coordinates

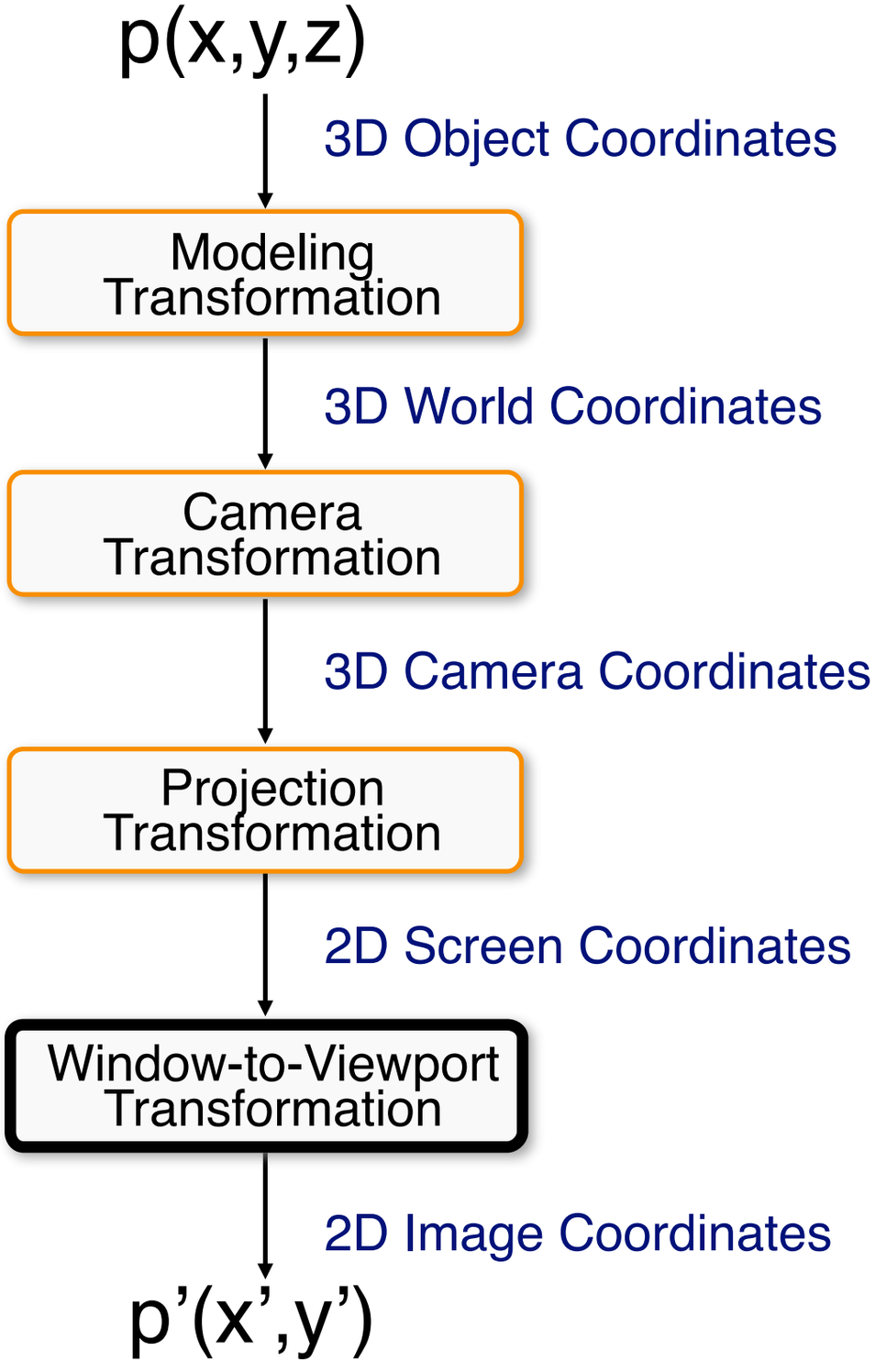
p’(x’,y’)

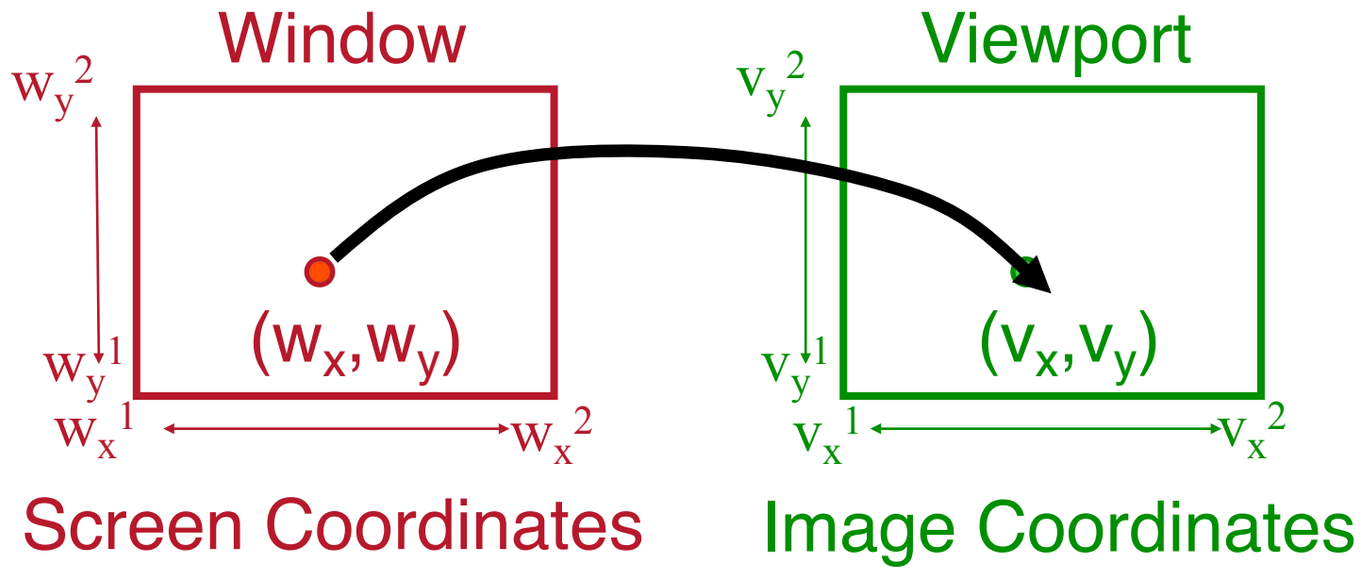


$$\text{Transform} = VPC^{-1}M$$

V := viewport transform

Transformations





$$\text{Transform} = VPC^{-1}M$$

V := viewport transform

$$V = \begin{bmatrix} 1 & 0 & 0 & v_x^1 \\ 0 & 1 & 0 & v_y^1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{v_x^2 - v_x^1}{w_x^2 - w_x^1} & 0 & 0 & 0 \\ 0 & \frac{v_y^2 - v_y^1}{w_y^2 - w_y^1} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -w_x^1 \\ 0 & 1 & 0 & -w_y^1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3D Rendering Pipeline (for direct illumination)

p(x,y,z)

3D Object Coordinates

Modeling Transformation

3D World Coordinates

Camera Transformation

3D Camera Coordinates

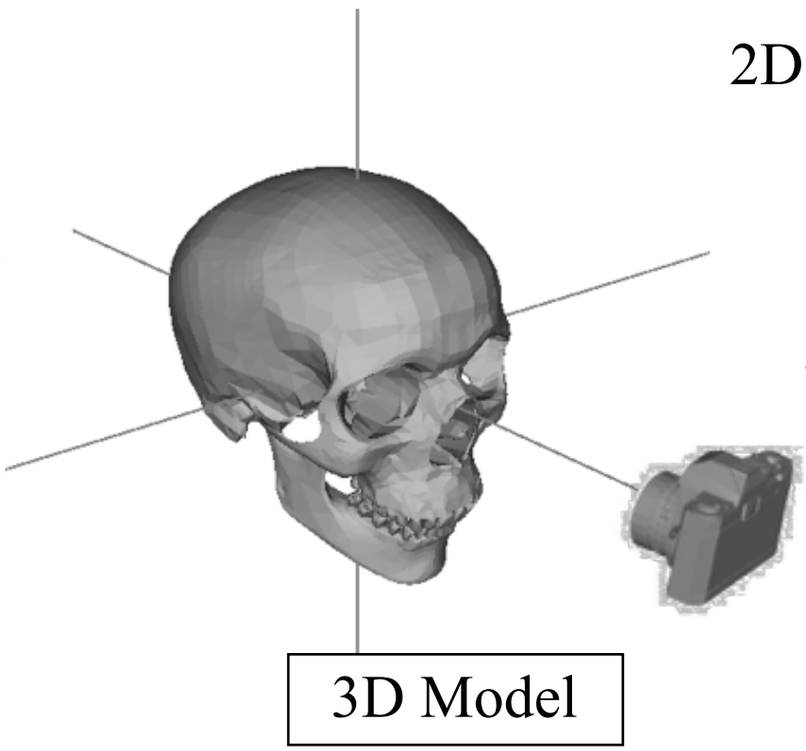
Projection Transformation

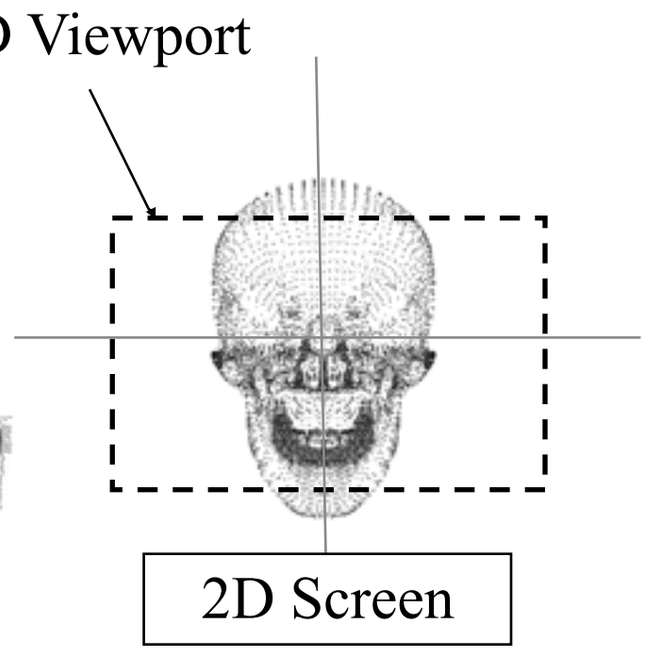
2D Screen Coordinates

Window-to-Viewport Transformation

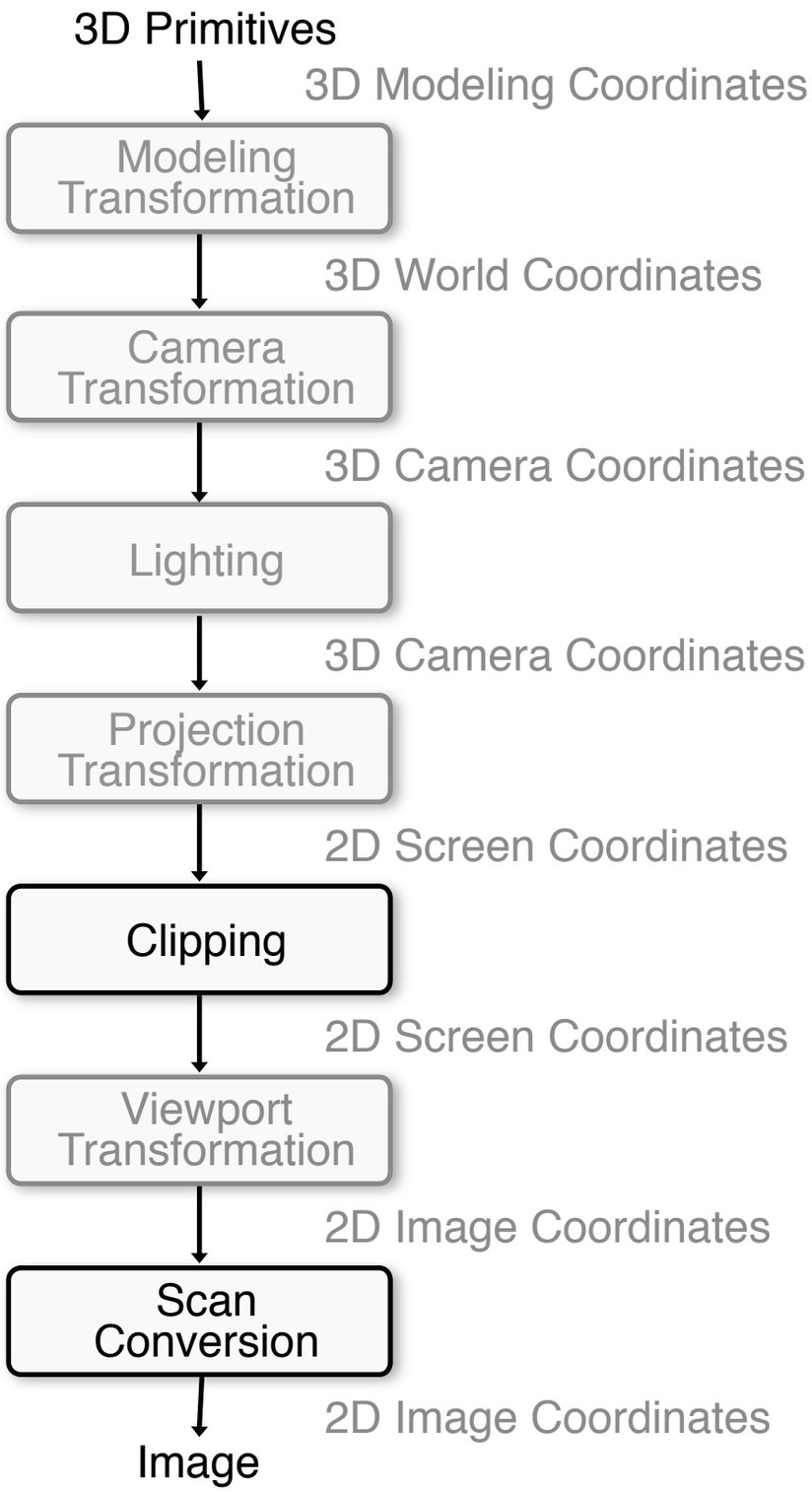
2D Image Coordinates

p’(x’,y’)





3D Rendering Pipeline (for direct illumination)



Clipping

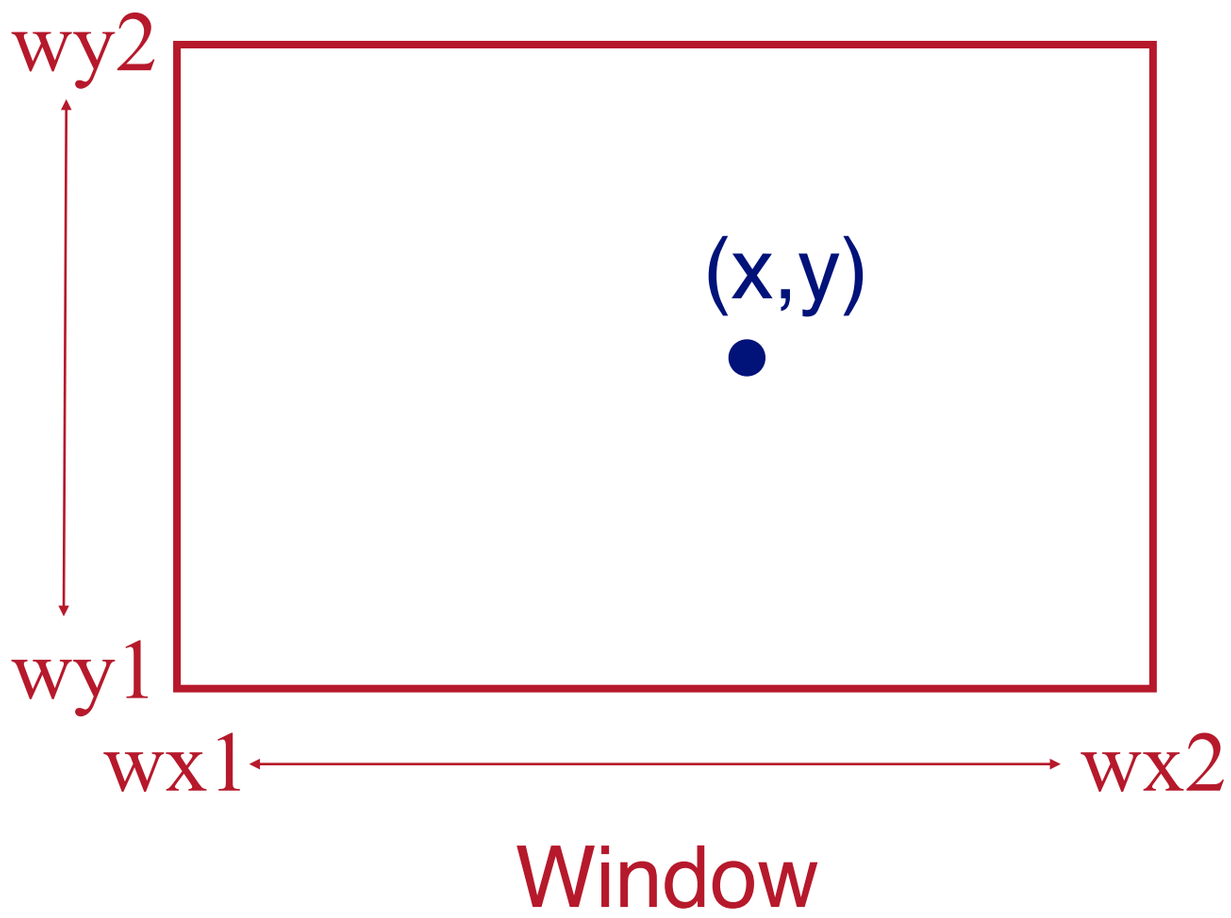
- Avoid drawing parts of primitives outside window
  - Window defines part of scene being viewed
  - Must draw geometric primitives only inside window

Clipping

- Avoid drawing parts of primitives outside window
  - **Points**
  - Line Segments
  - Polygons

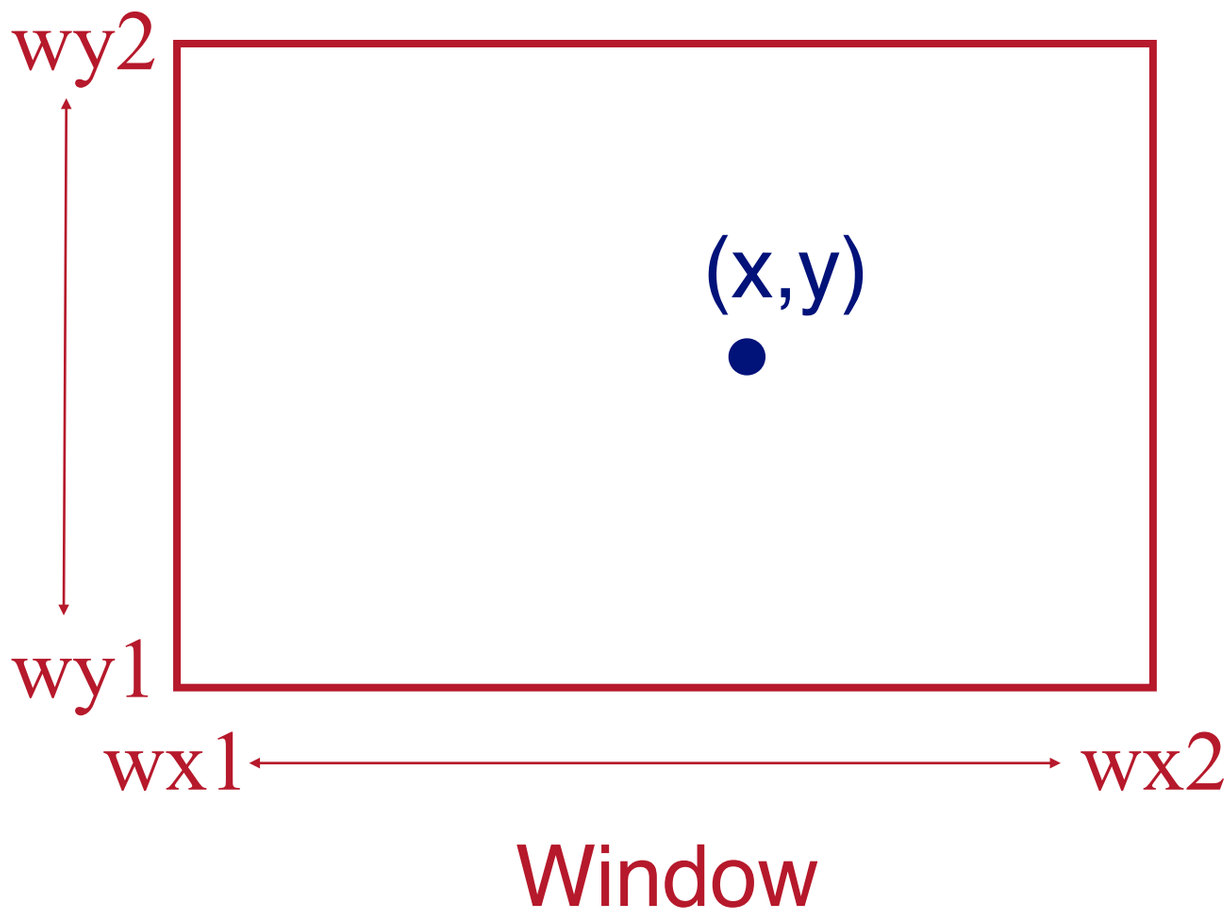
Point Clipping

- Is point (x,y) inside the clip window?



Point Clipping

- Is point (x,y) inside the clip window?



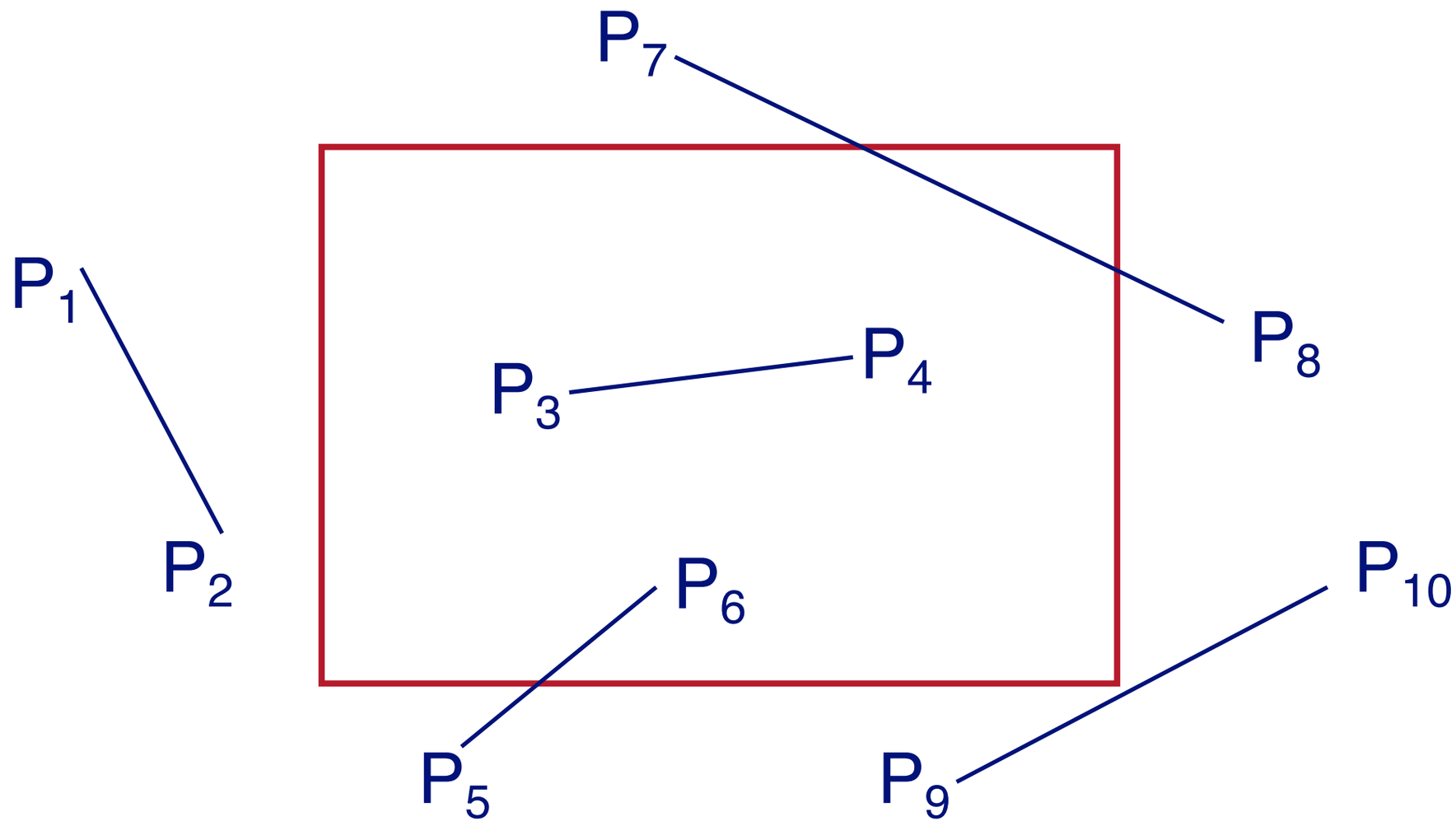

```
inside =
   (x >= wx1)  &&
   (x <= wx2)  &&
   (y >= wy1)  &&
   (y <= wy2);
```

Clipping

- Avoid drawing parts of primitives outside window
  - Points
  - **Line Segments**
  - Polygons

Line Segment Clipping

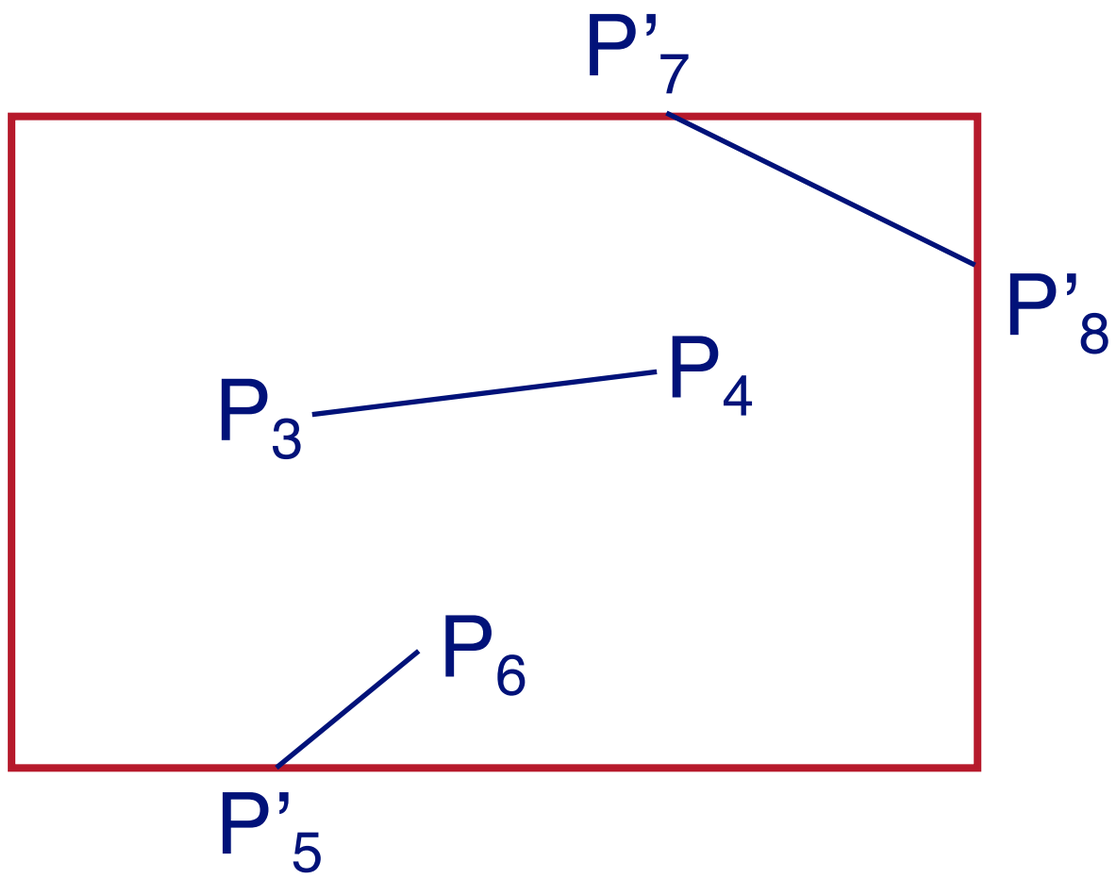
- Find the part of a line inside the clip window



Before Clipping

Line Segment Clipping

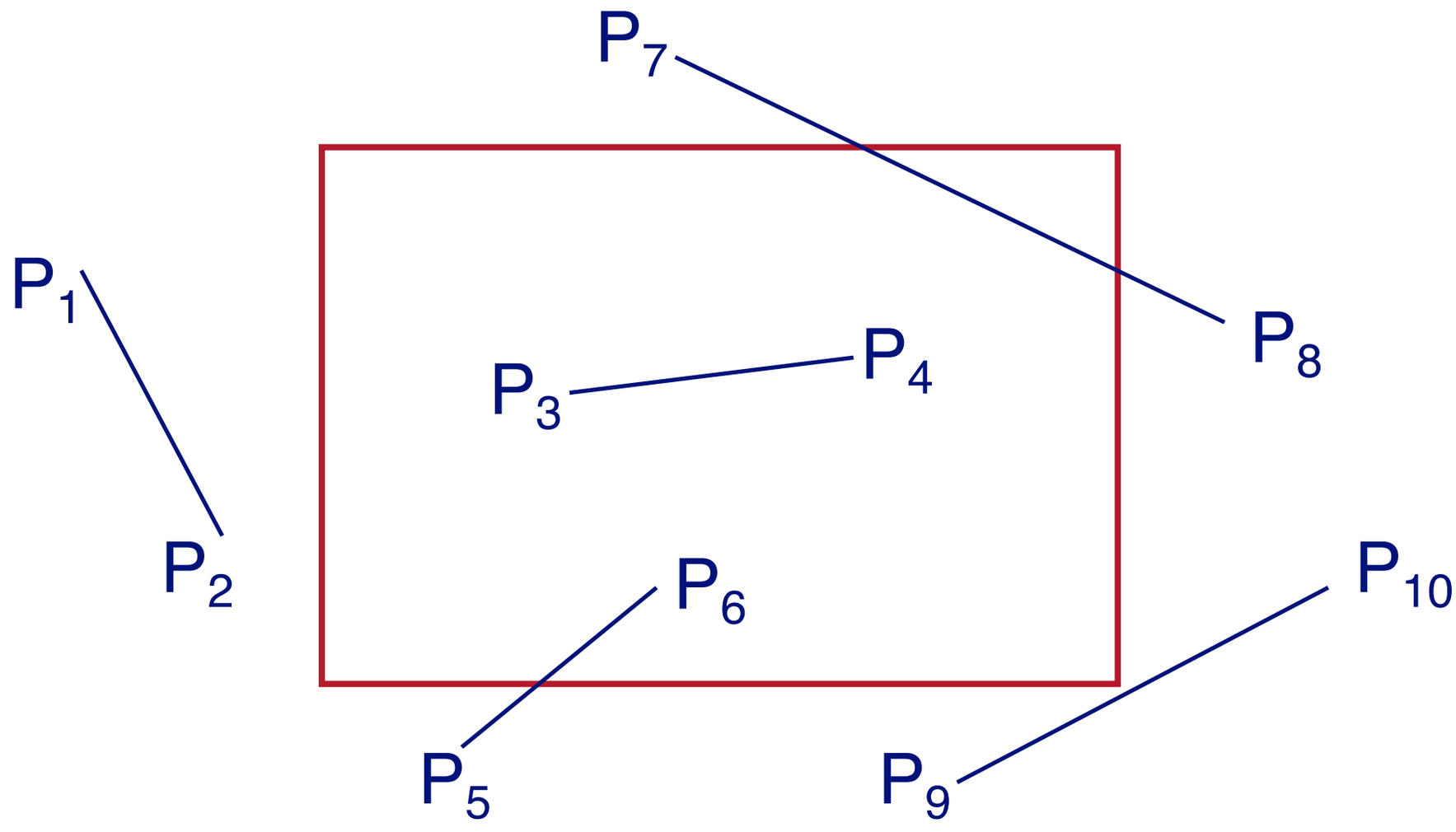
- Find the part of a line inside the clip window



After Clipping

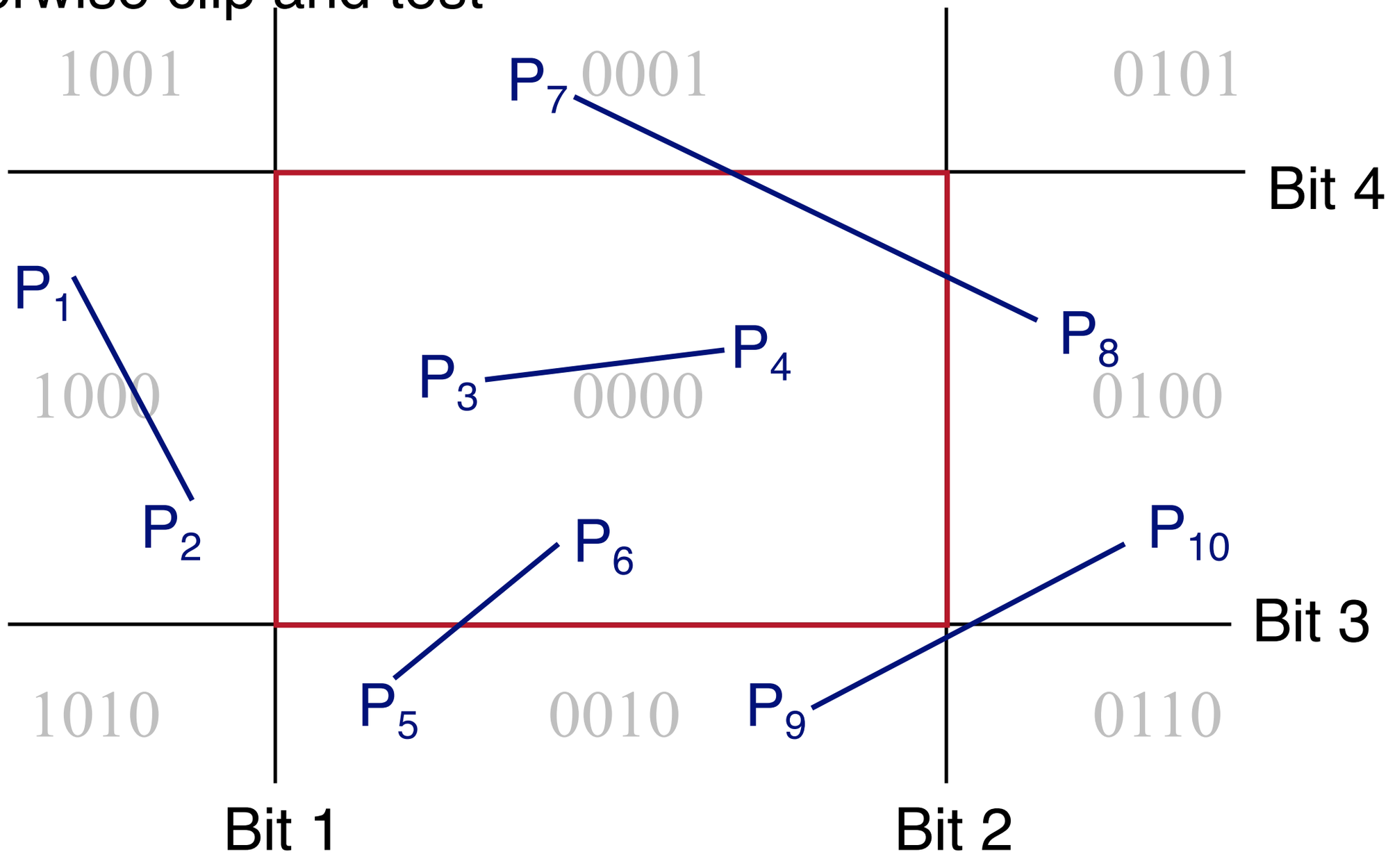
Cohen-Sutherland Line Clipping

- Use simple tests to classify easy cases first



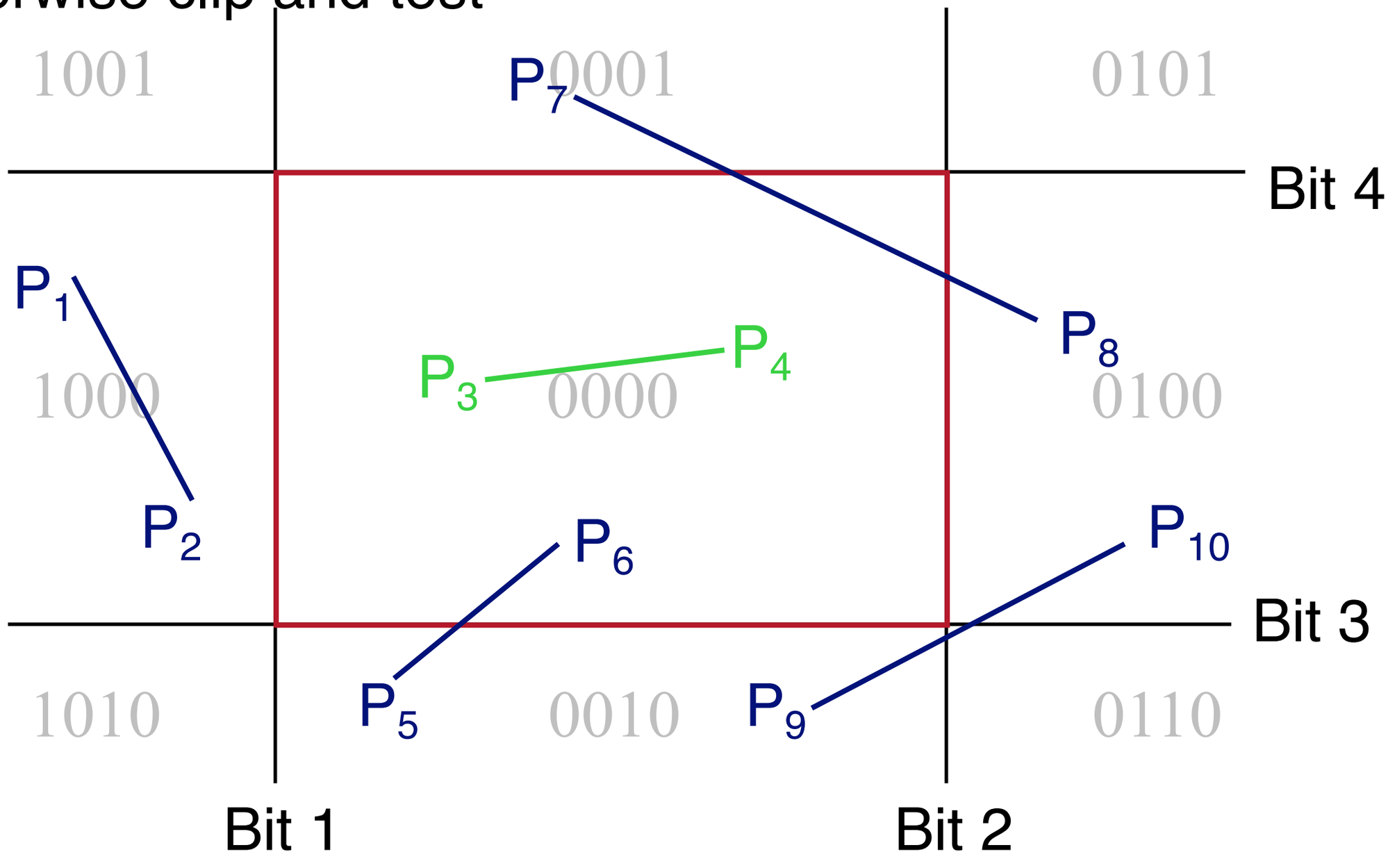
Cohen-Sutherland Line Clipping

- If both outcodes are 0, line segment is inside
- If AND of outcodes not 0, line segment is outside
- Otherwise clip and test



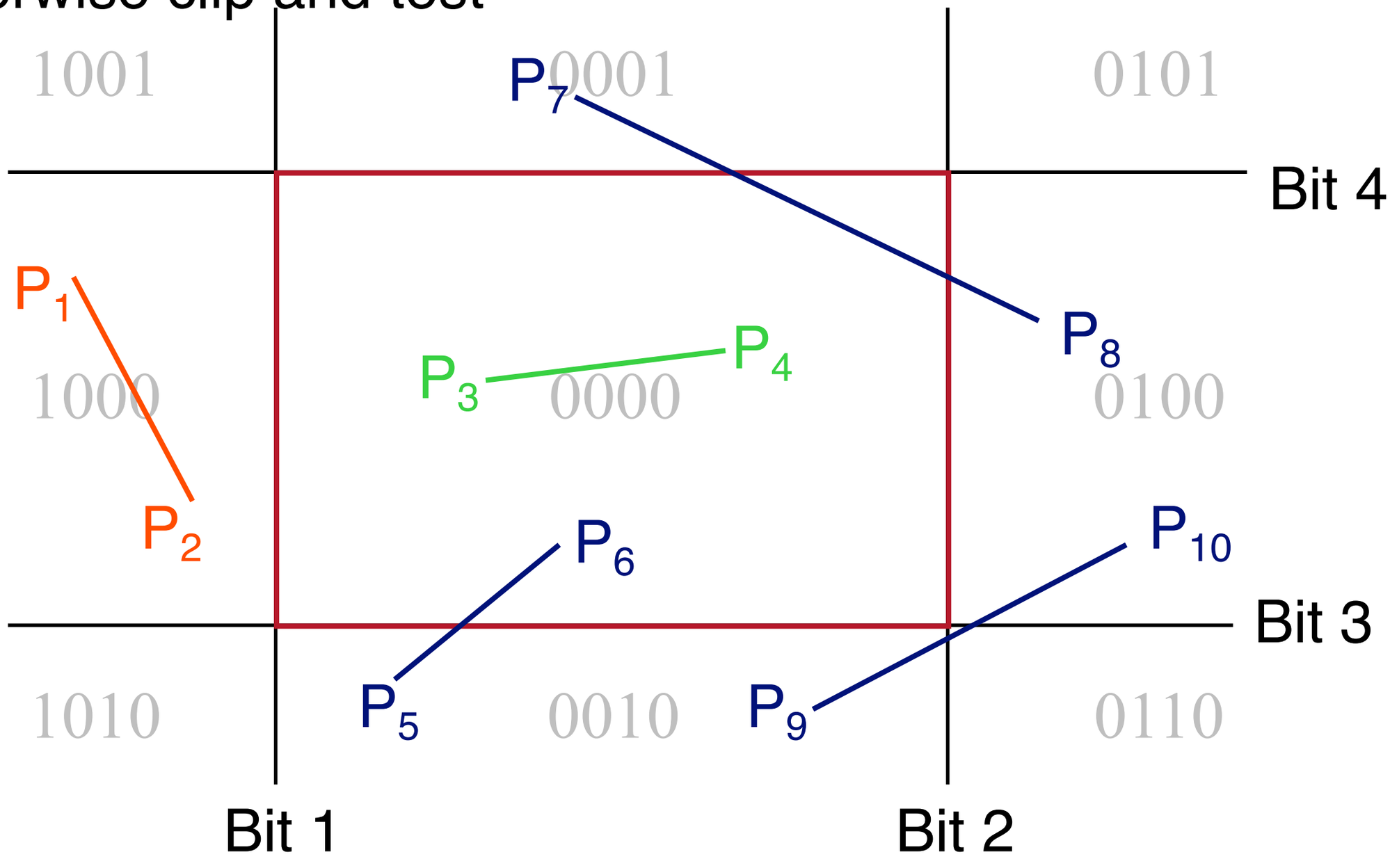
Cohen-Sutherland Line Clipping

- **If both outcodes are 0, line segment is inside**
- If AND of outcodes not 0, line segment is outside
- Otherwise clip and test



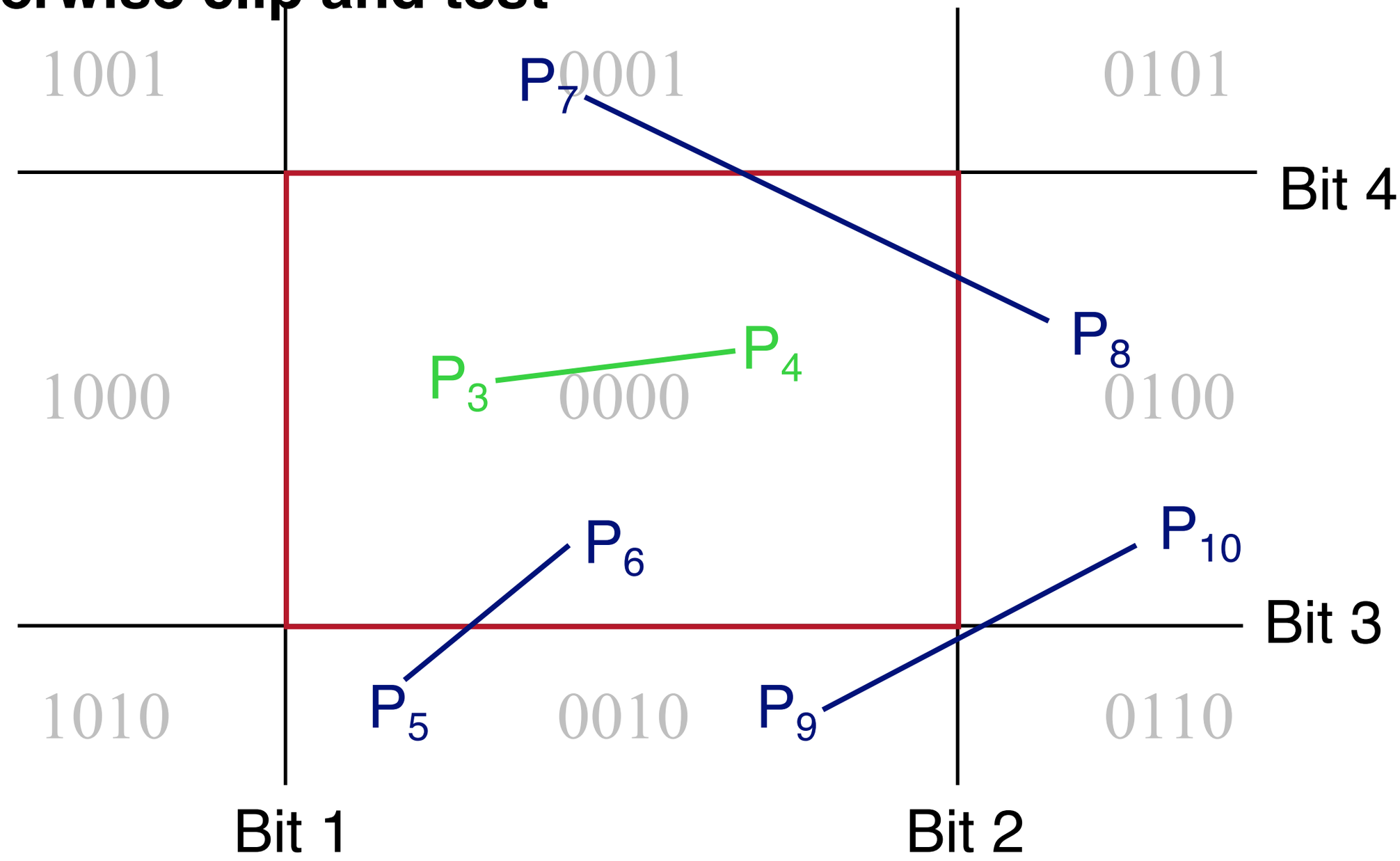
Cohen-Sutherland Line Clipping

- If both outcodes are 0, line segment is inside
- **If AND of outcodes not 0, line segment is outside**
- Otherwise clip and test



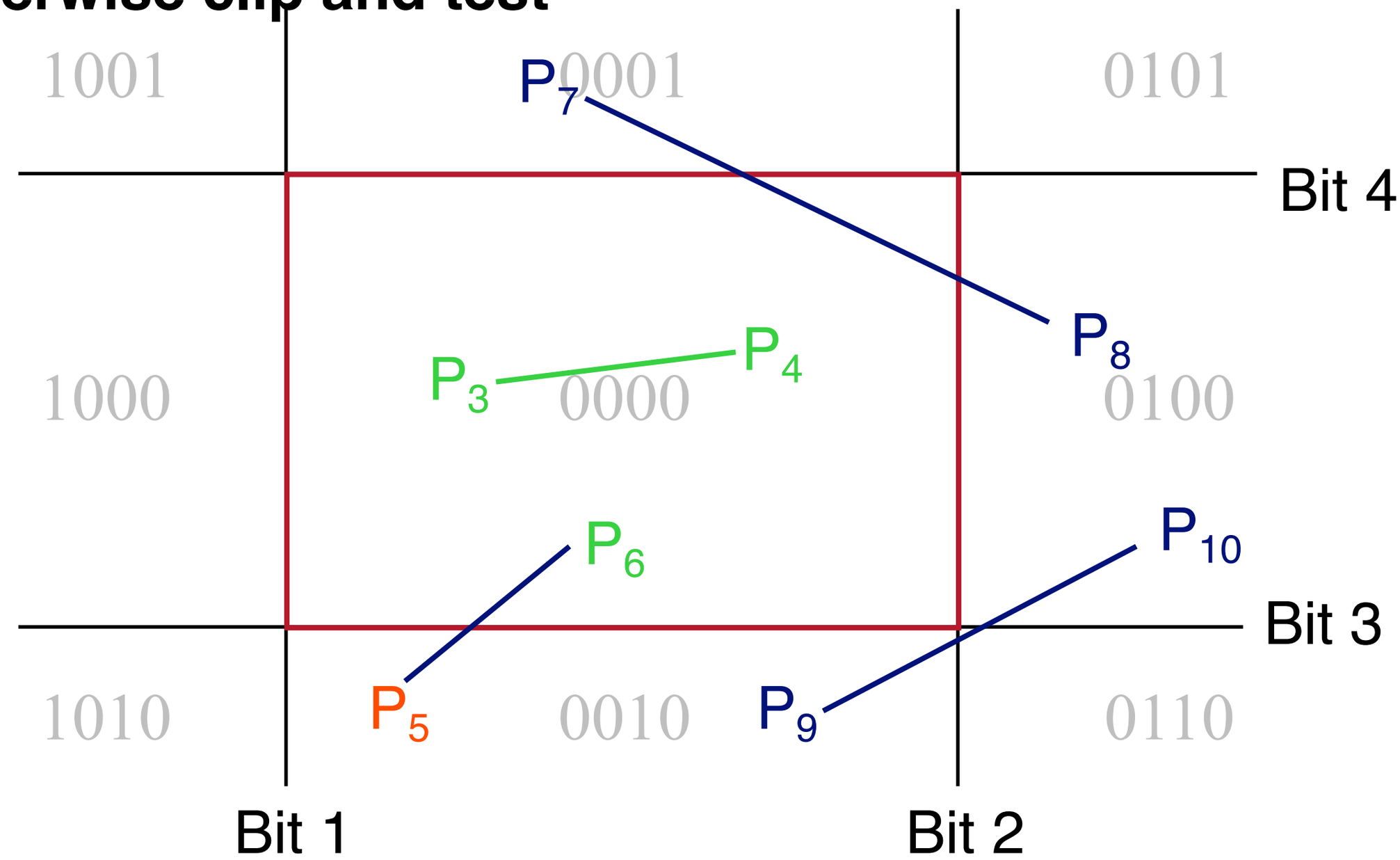
Cohen-Sutherland Line Clipping

- If both outcodes are 0, line segment is inside
- If AND of outcodes not 0, line segment is outside
- **Otherwise clip and test**



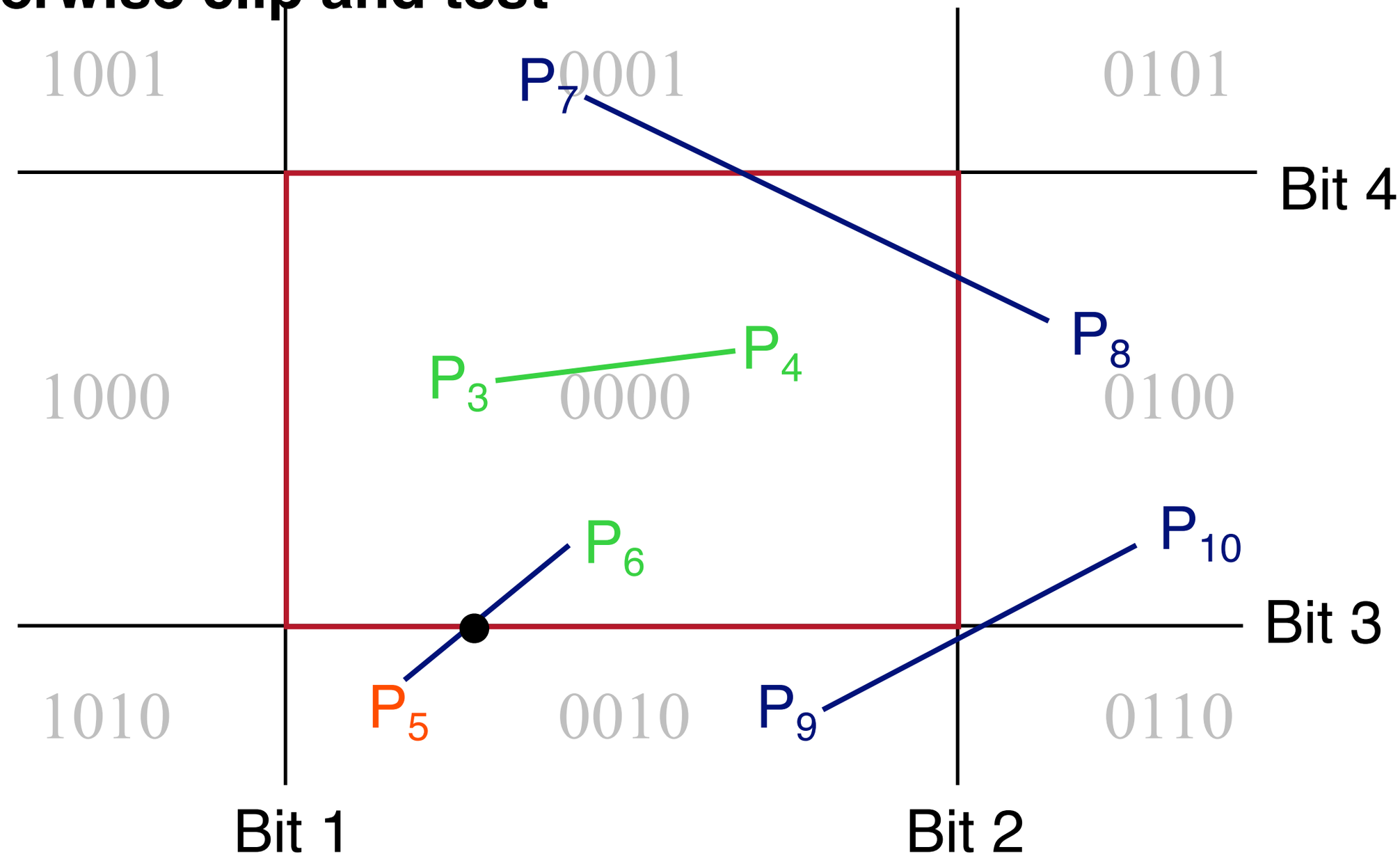
Cohen-Sutherland Line Clipping

- If both outcodes are 0, line segment is inside
- If AND of outcodes not 0, line segment is outside
- **Otherwise clip and test**



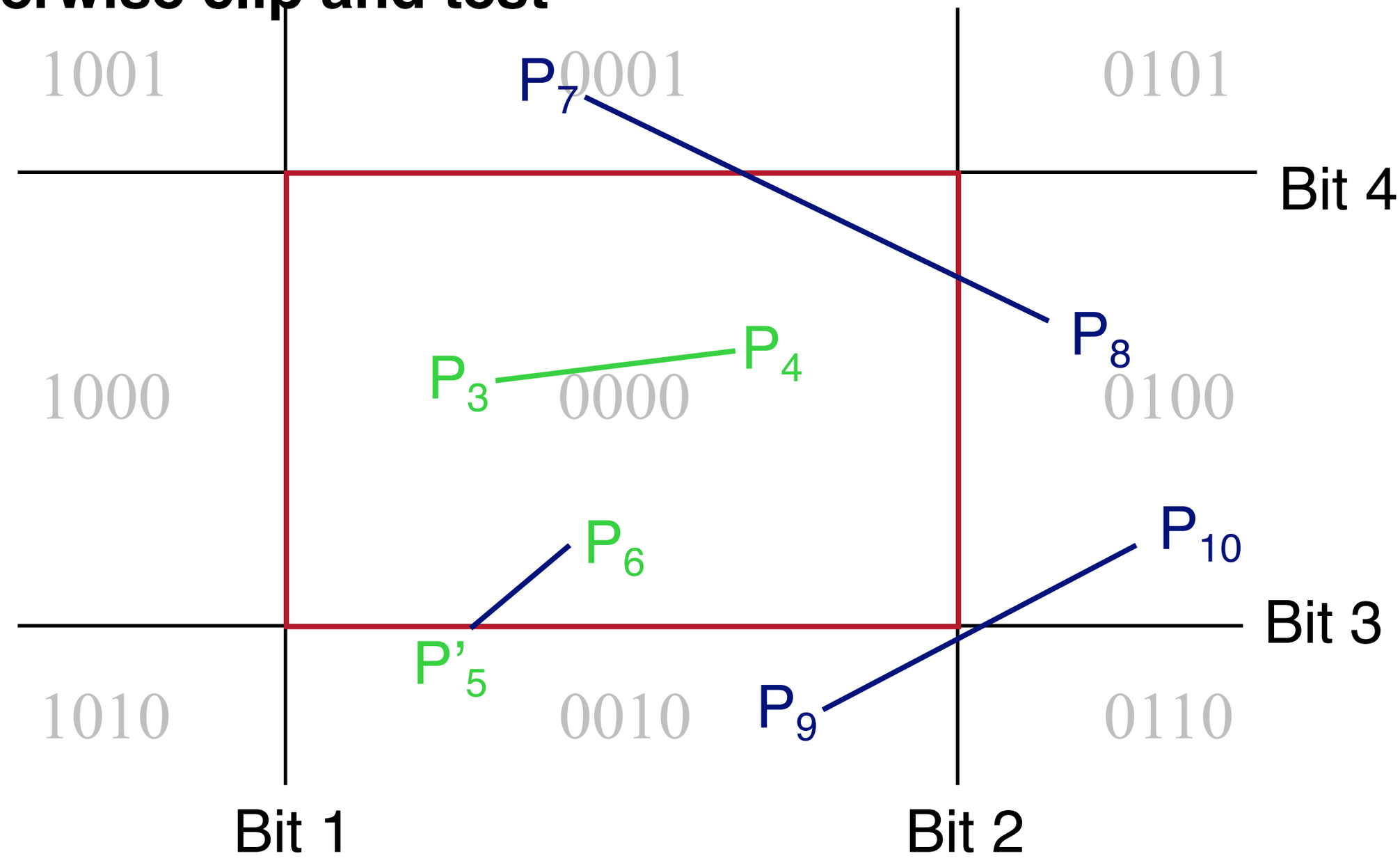
Cohen-Sutherland Line Clipping

- If both outcodes are 0, line segment is inside
- If AND of outcodes not 0, line segment is outside
- **Otherwise clip and test**



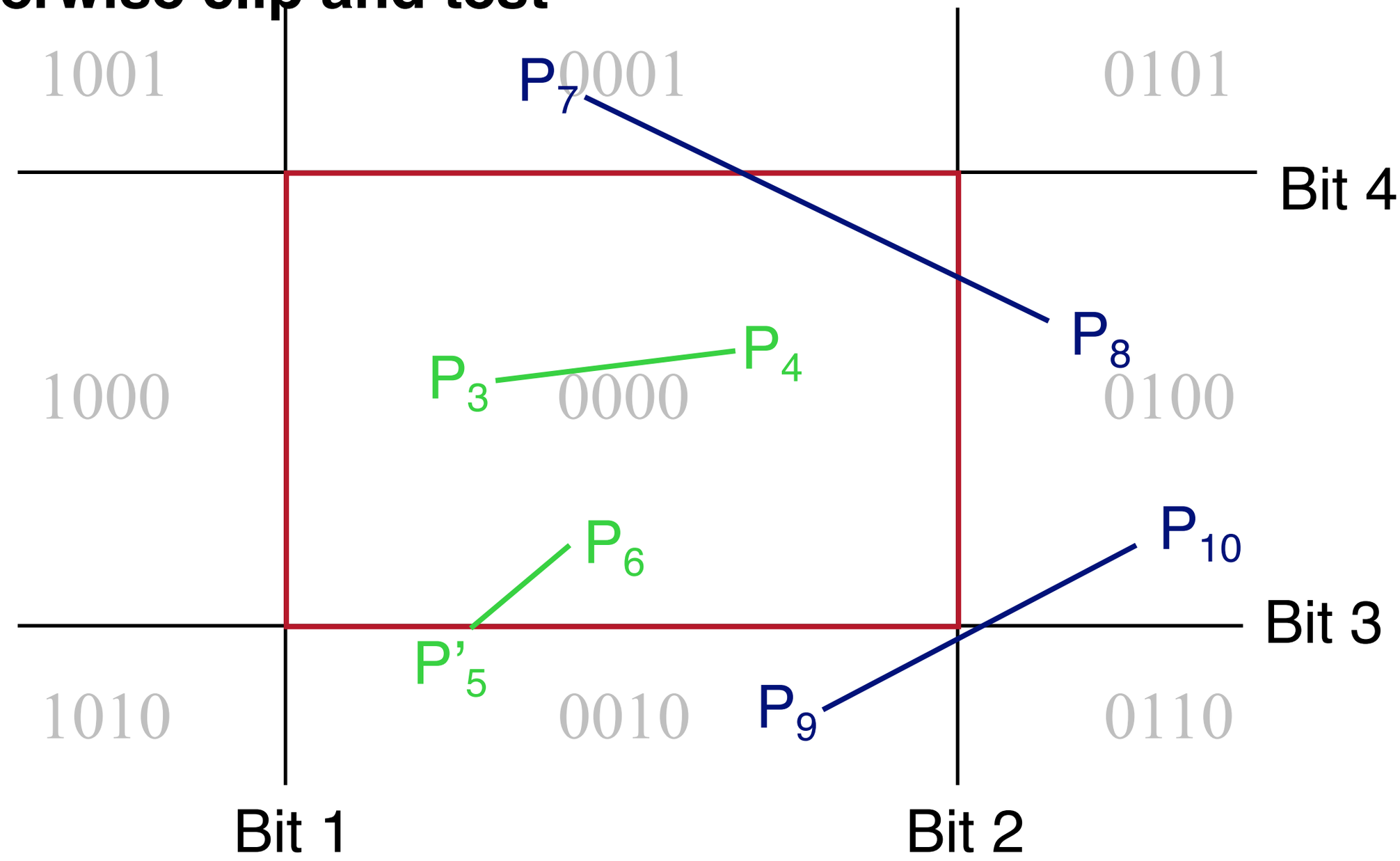
Cohen-Sutherland Line Clipping

- If both outcodes are 0, line segment is inside
- If AND of outcodes not 0, line segment is outside
- **Otherwise clip and test**



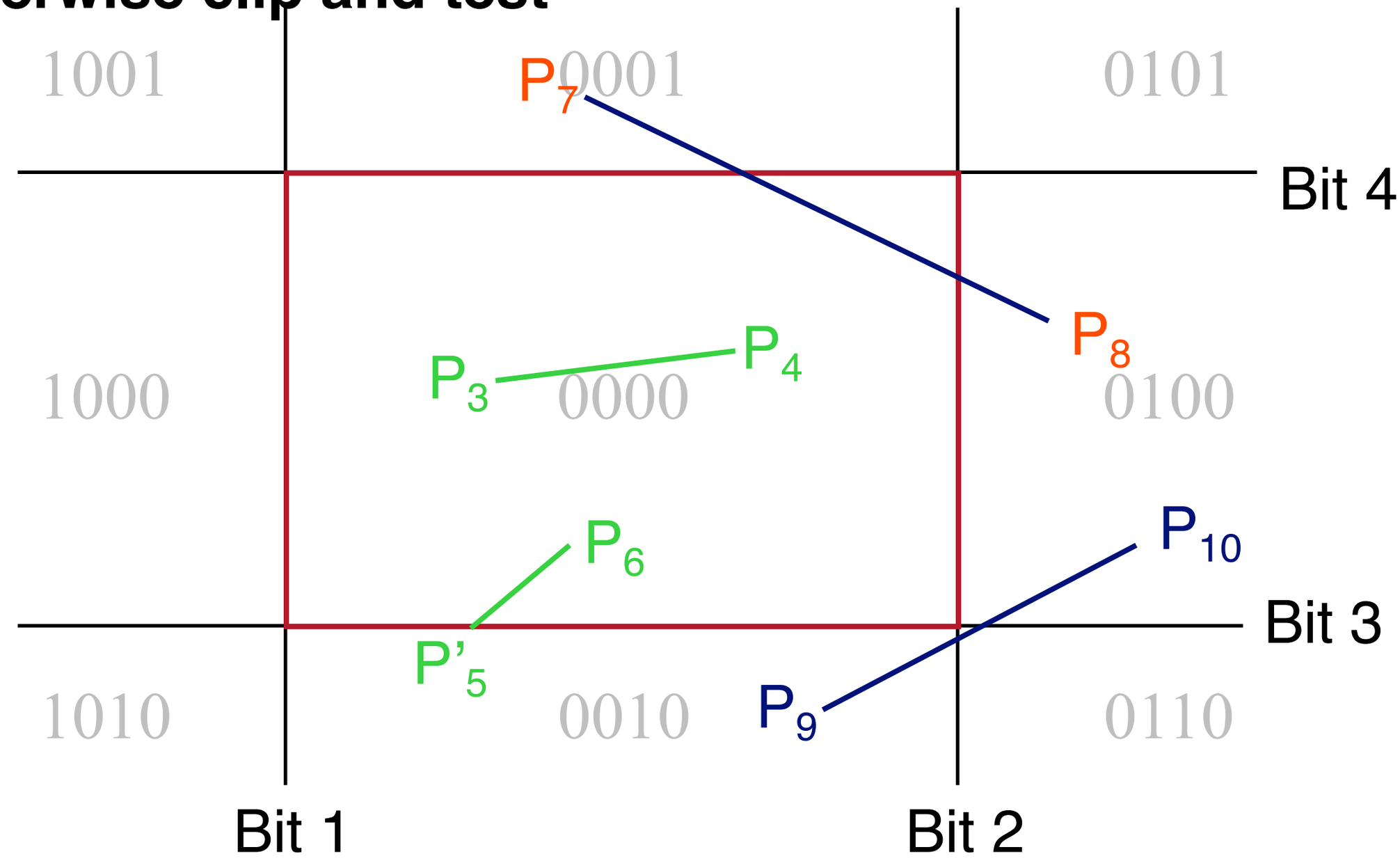
Cohen-Sutherland Line Clipping

- If both outcodes are 0, line segment is inside
- If AND of outcodes not 0, line segment is outside
- **Otherwise clip and test**



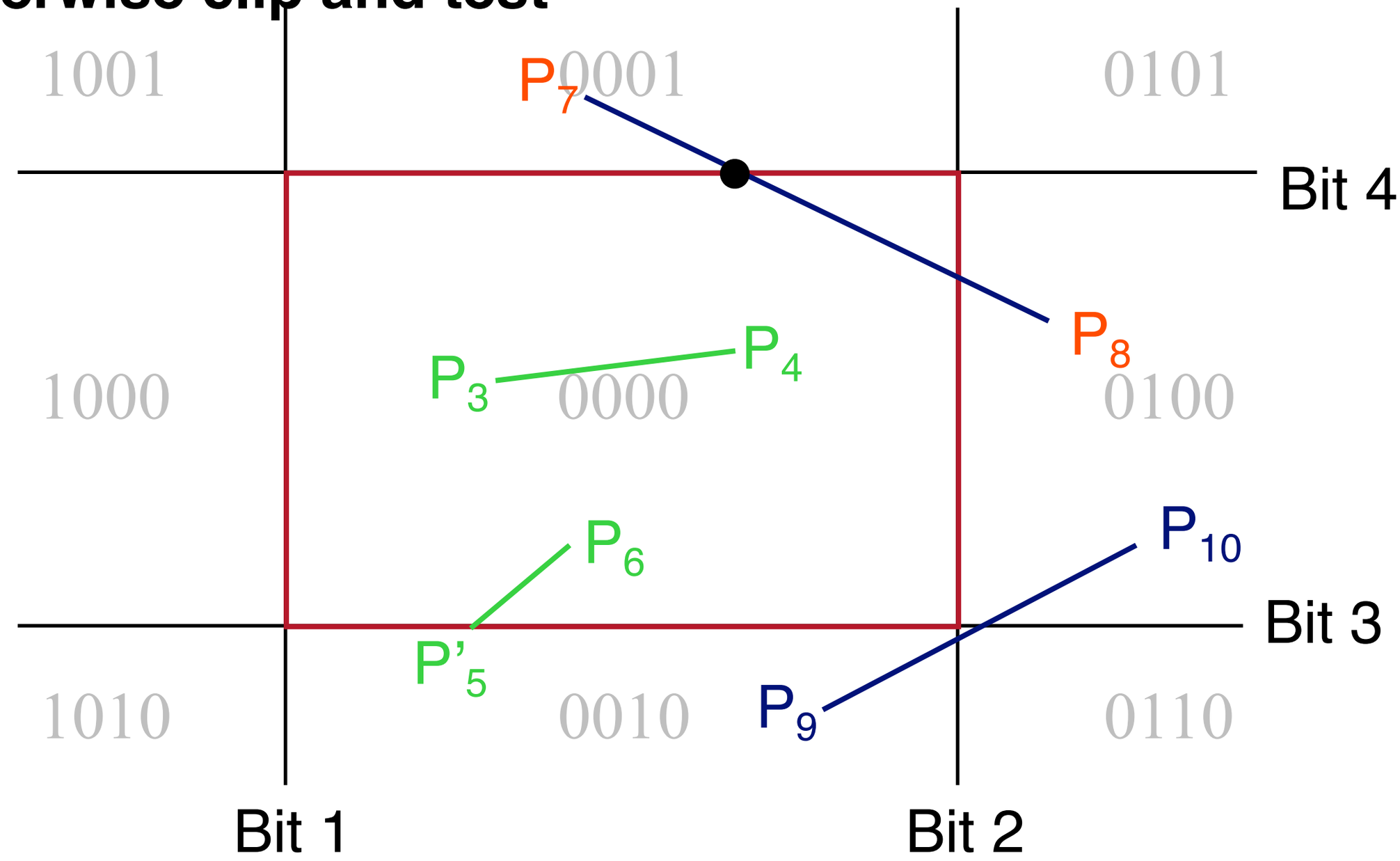
Cohen-Sutherland Line Clipping

- If both outcodes are 0, line segment is inside
- If AND of outcodes not 0, line segment is outside
- **Otherwise clip and test**



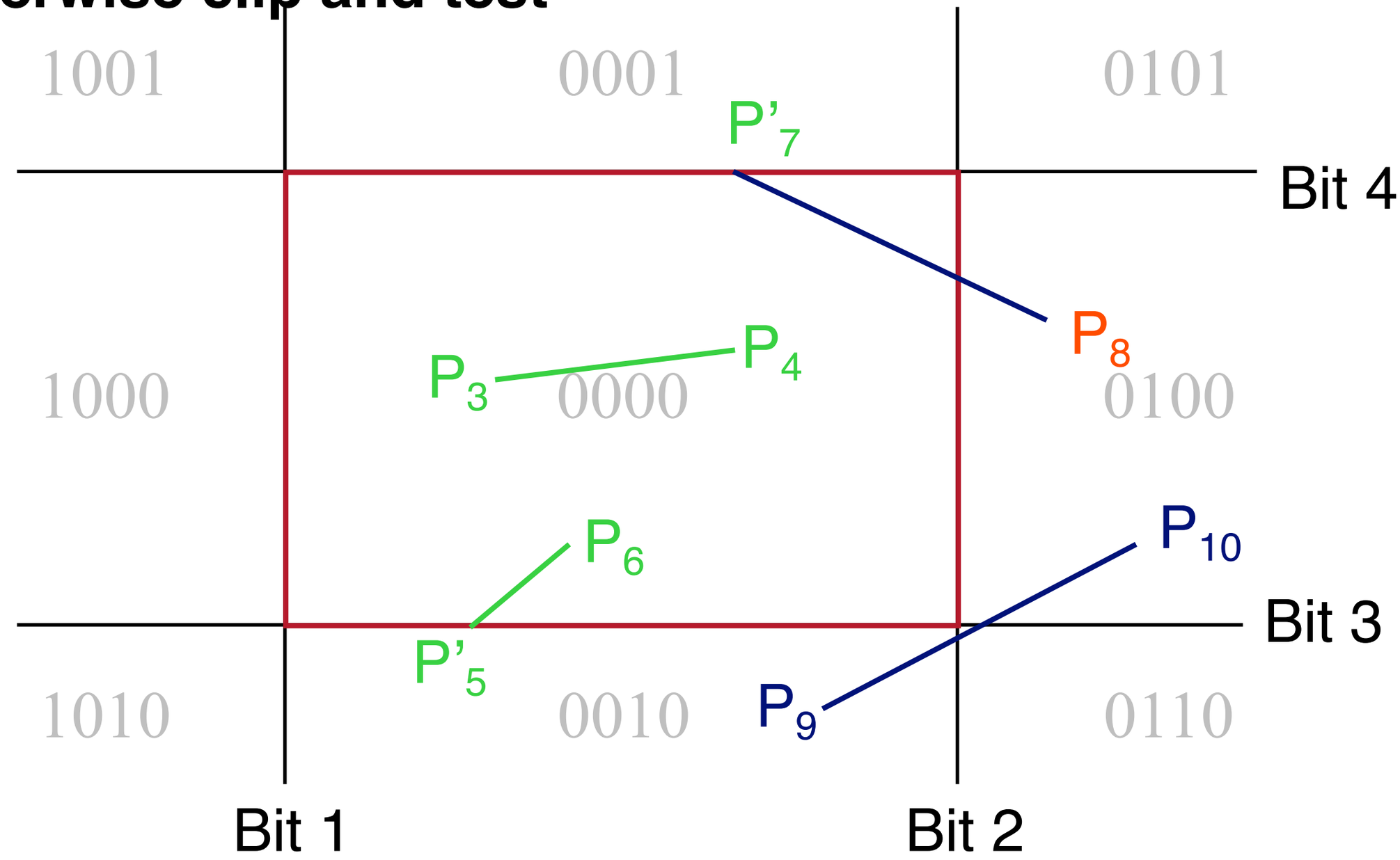
Cohen-Sutherland Line Clipping

- If both outcodes are 0, line segment is inside
- If AND of outcodes not 0, line segment is outside
- **Otherwise clip and test**



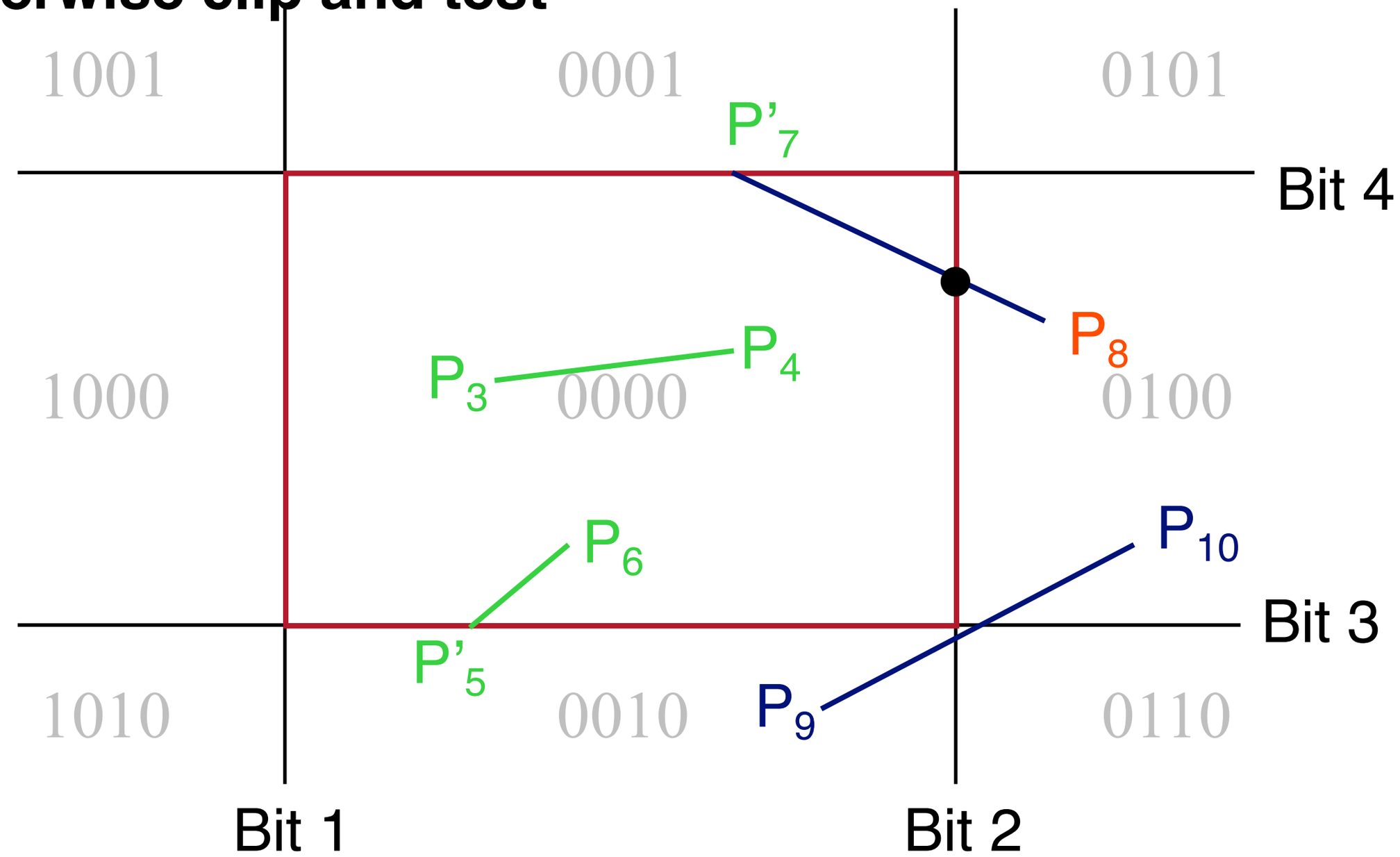
Cohen-Sutherland Line Clipping

- If both outcodes are 0, line segment is inside
- If AND of outcodes not 0, line segment is outside
- **Otherwise clip and test**



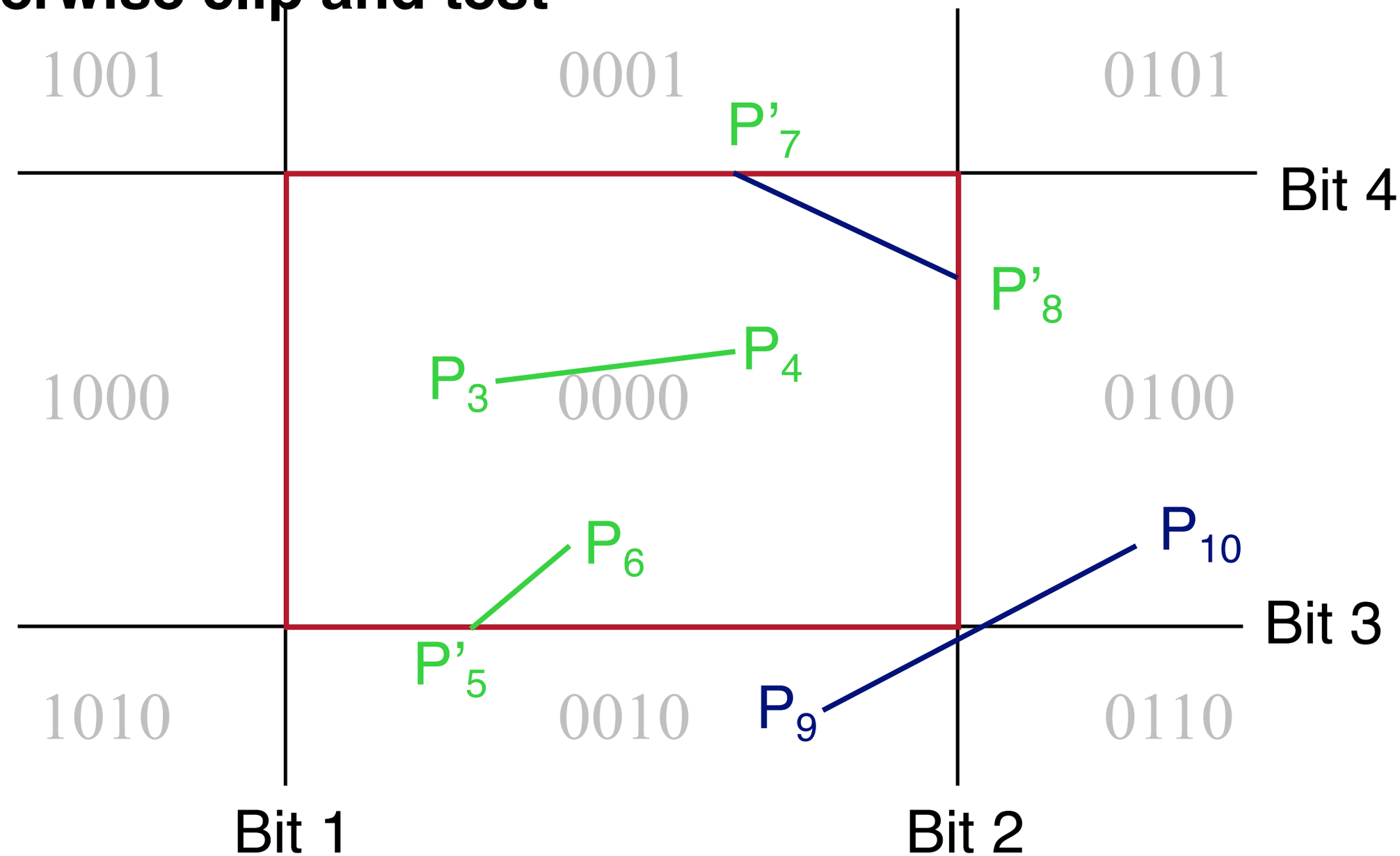
Cohen-Sutherland Line Clipping

- If both outcodes are 0, line segment is inside
- If AND of outcodes not 0, line segment is outside
- **Otherwise clip and test**



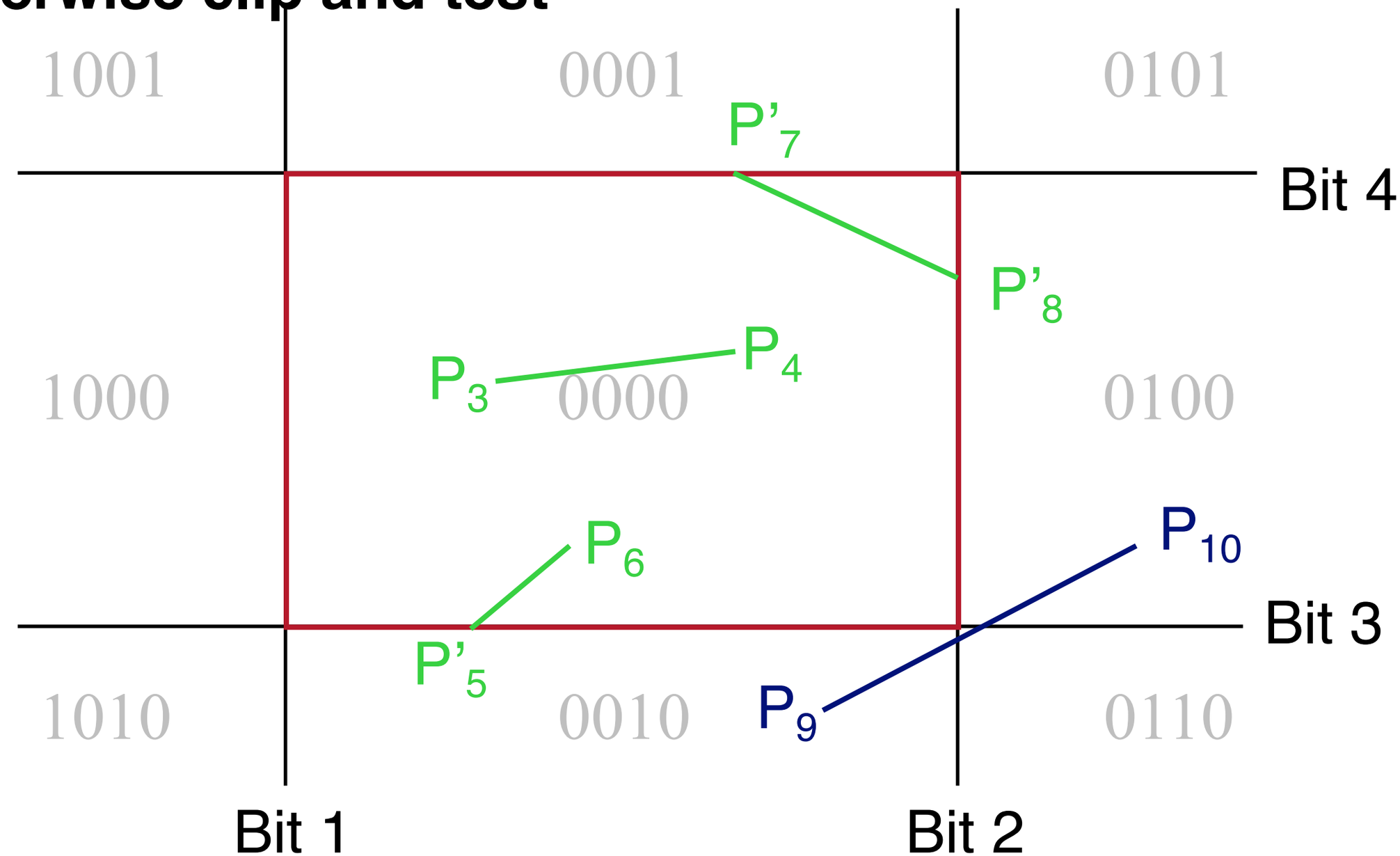
Cohen-Sutherland Line Clipping

- If both outcodes are 0, line segment is inside
- If AND of outcodes not 0, line segment is outside
- **Otherwise clip and test**



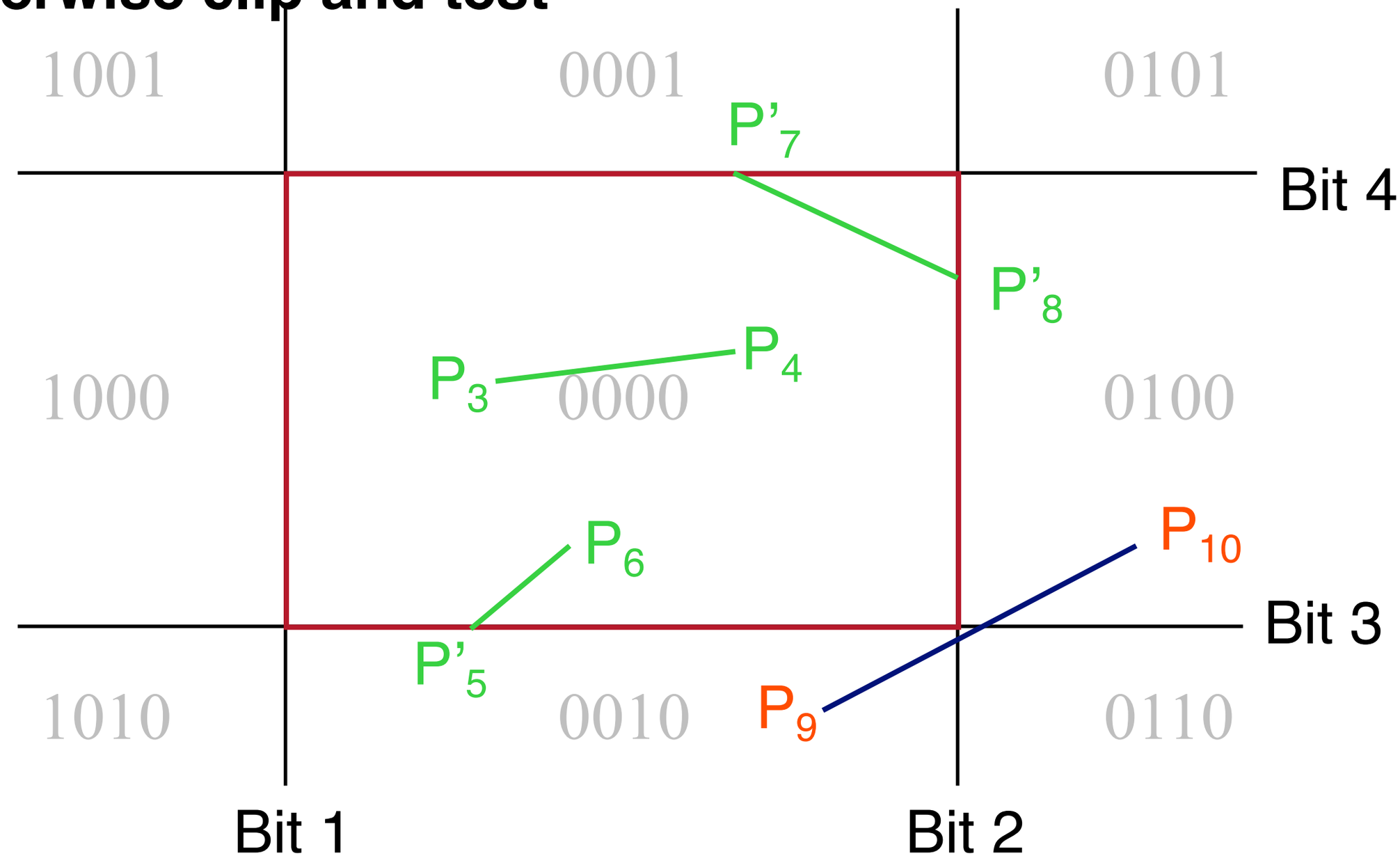
Cohen-Sutherland Line Clipping

- If both outcodes are 0, line segment is inside
- If AND of outcodes not 0, line segment is outside
- **Otherwise clip and test**



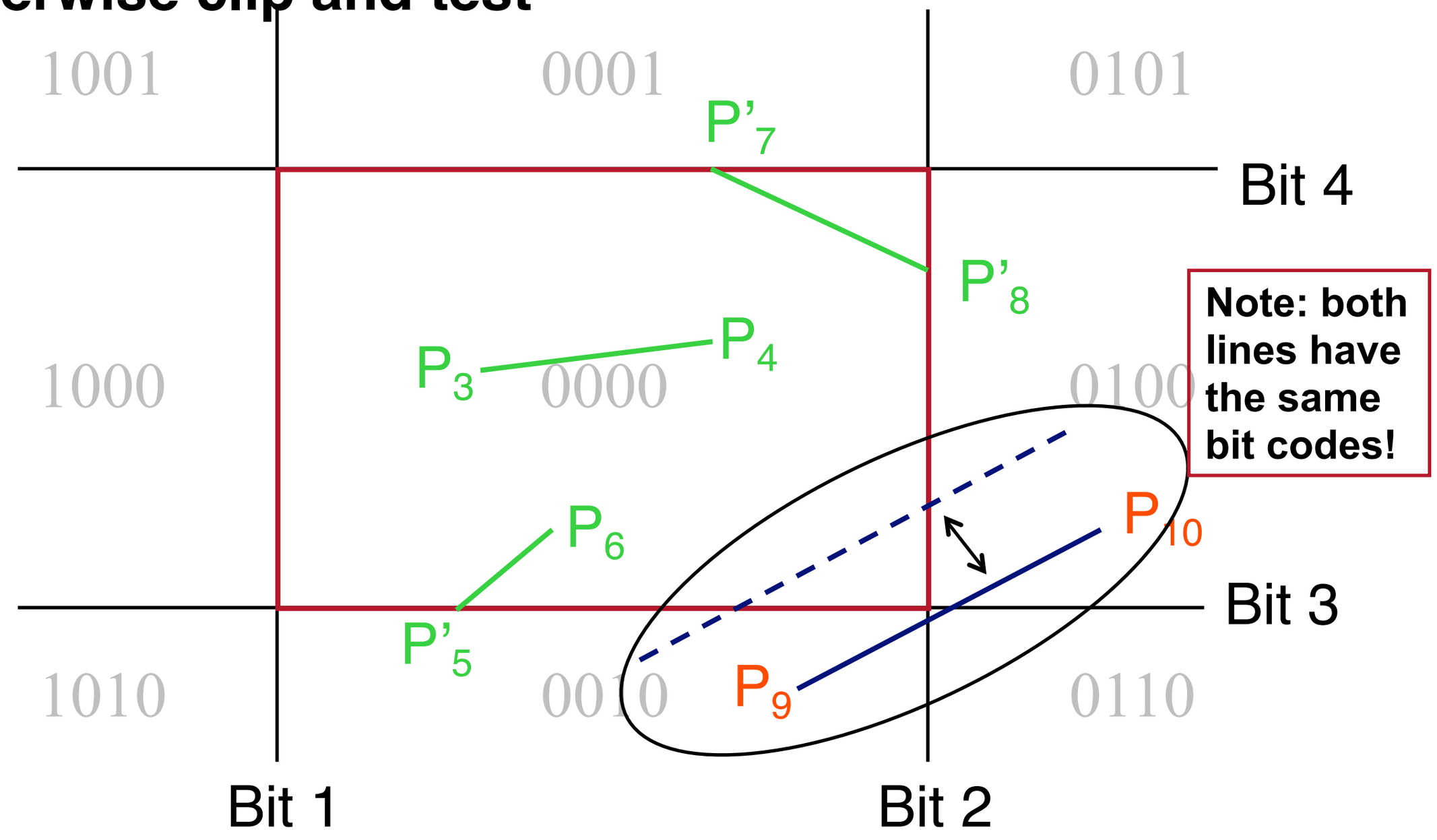
Cohen-Sutherland Line Clipping

- If both outcodes are 0, line segment is inside
- If AND of outcodes not 0, line segment is outside
- **Otherwise clip and test**



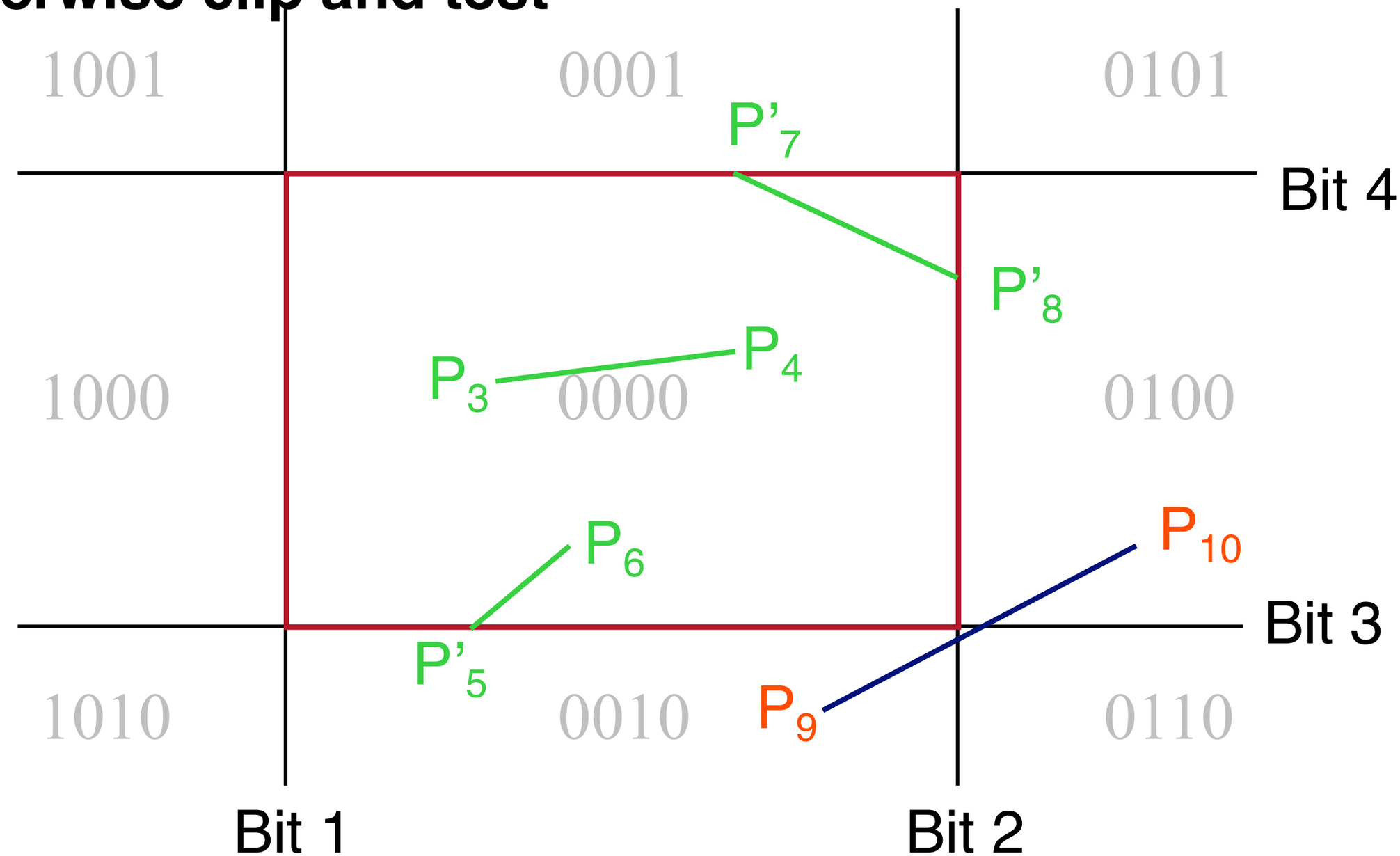
Cohen-Sutherland Line Clipping

- If both outcodes are 0, line segment is inside
- If AND of outcodes not 0, line segment is outside
- **Otherwise clip and test**



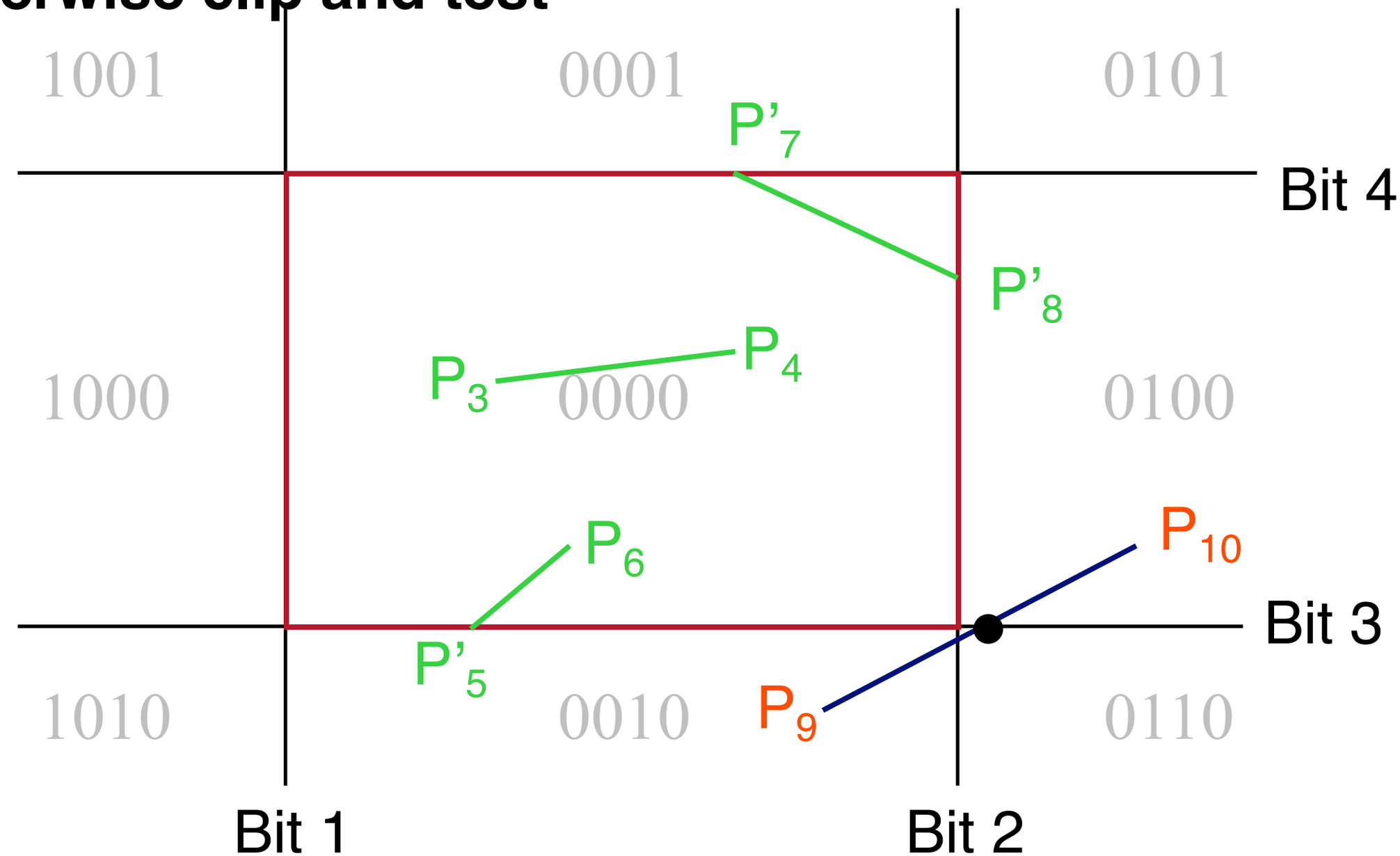
Cohen-Sutherland Line Clipping

- If both outcodes are 0, line segment is inside
- If AND of outcodes not 0, line segment is outside
- **Otherwise clip and test**



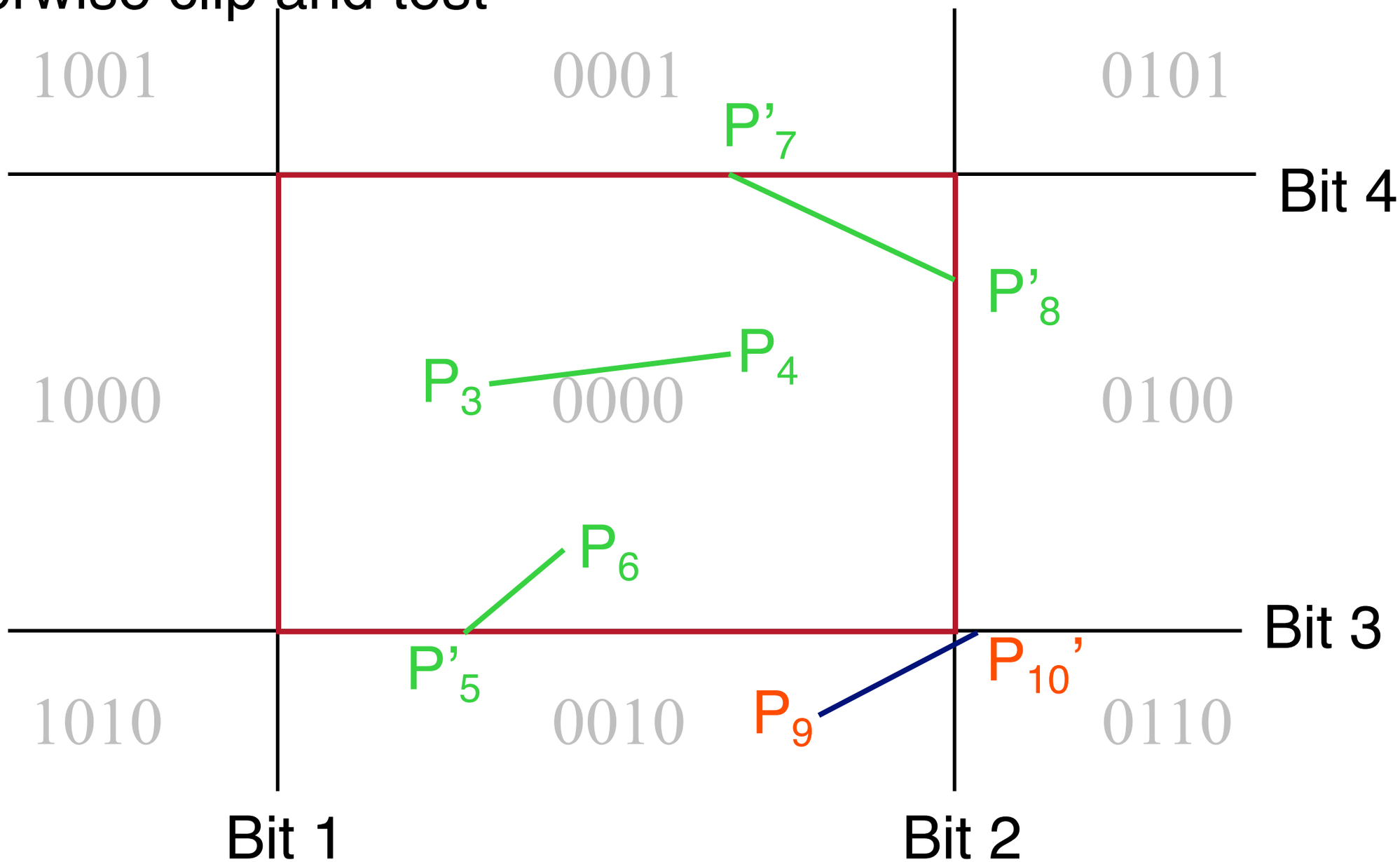
Cohen-Sutherland Line Clipping

- If both outcodes are 0, line segment is inside
- If AND of outcodes not 0, line segment is outside
- **Otherwise clip and test**



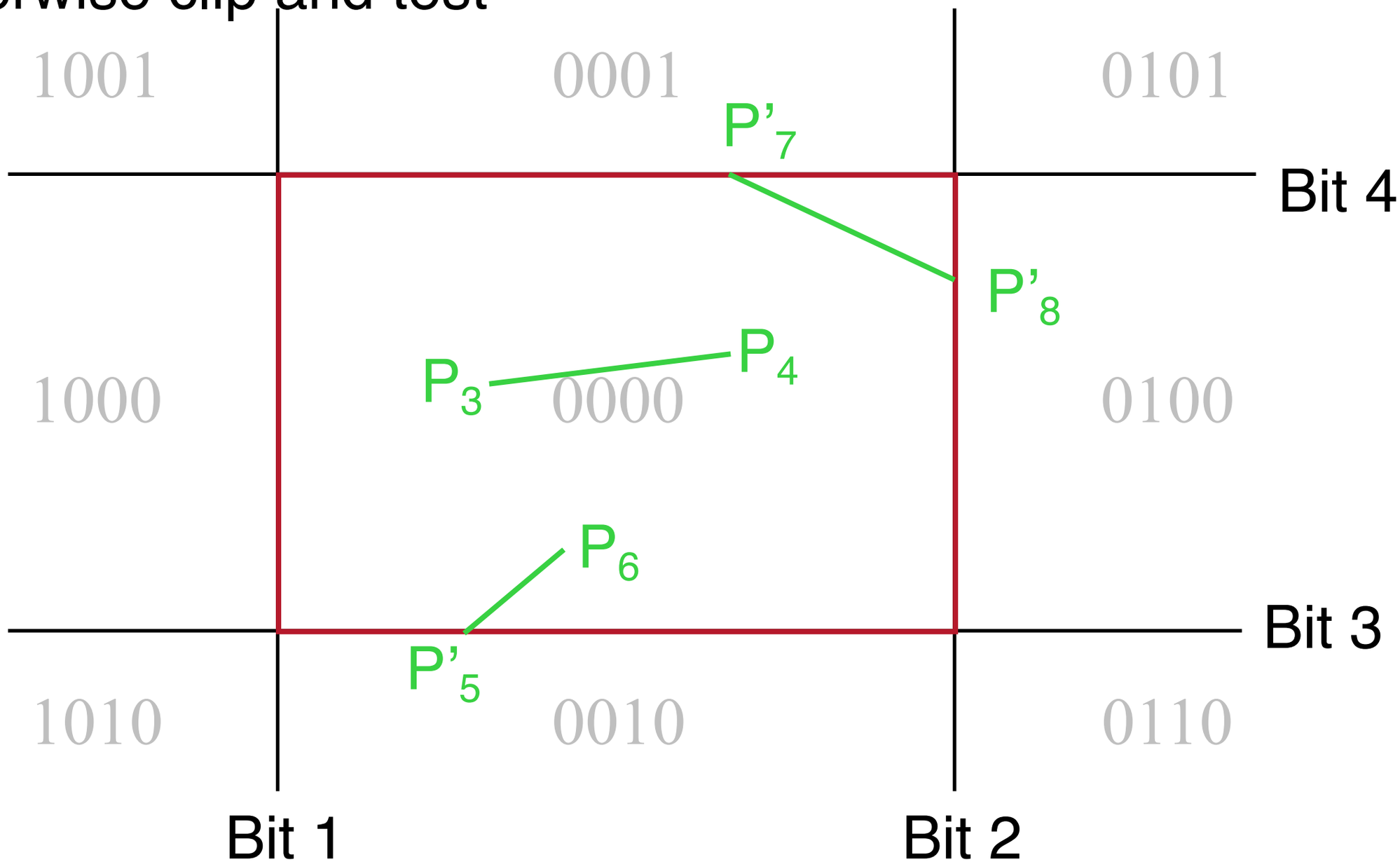
Cohen-Sutherland Line Clipping

- If both outcodes are 0, line segment is inside
- **If AND of outcodes not 0, line segment is outside**
- Otherwise clip and test



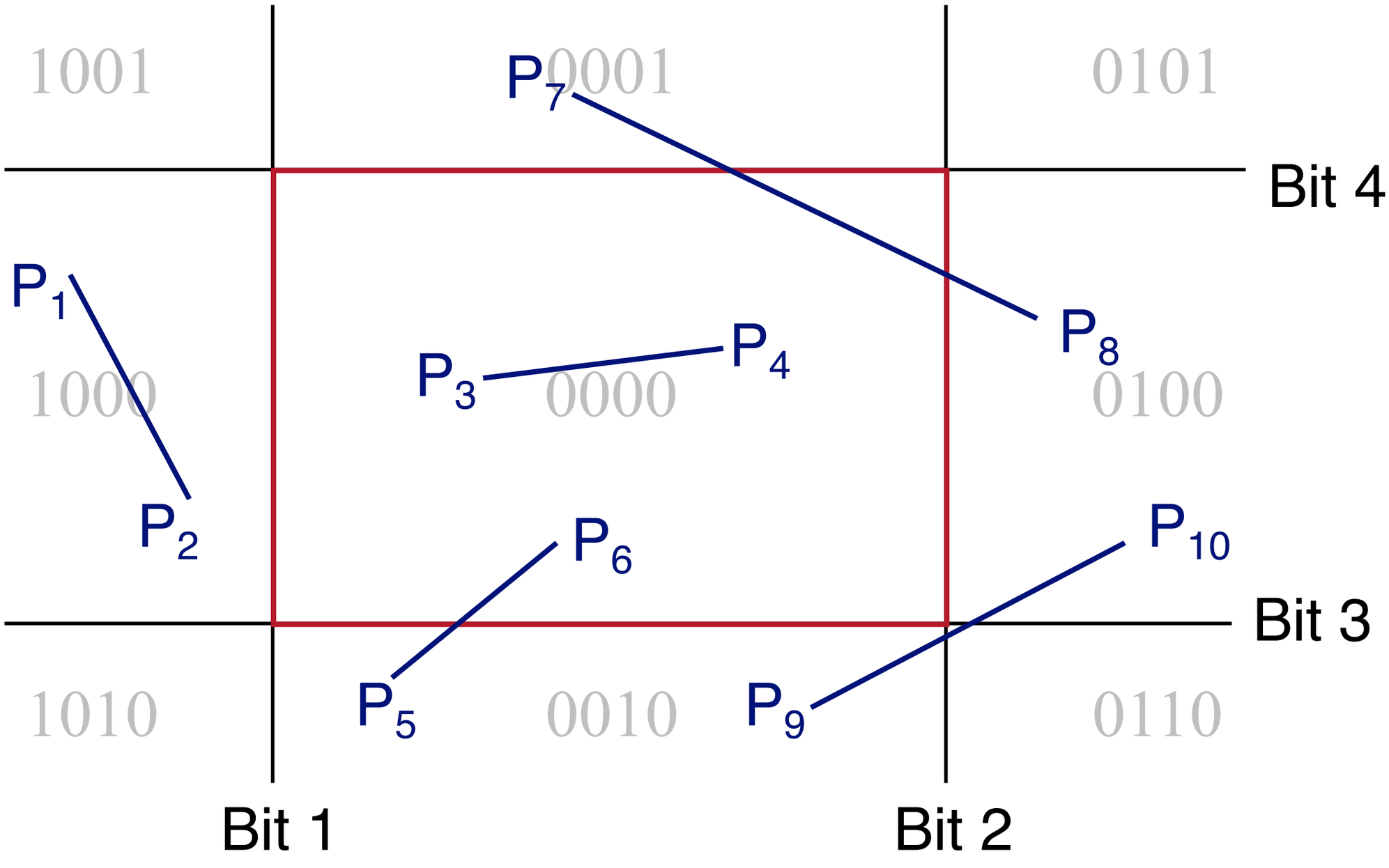
Cohen-Sutherland Line Clipping

- If both outcodes are 0, line segment is inside
- **If AND of outcodes not 0, line segment is outside**
- Otherwise clip and test



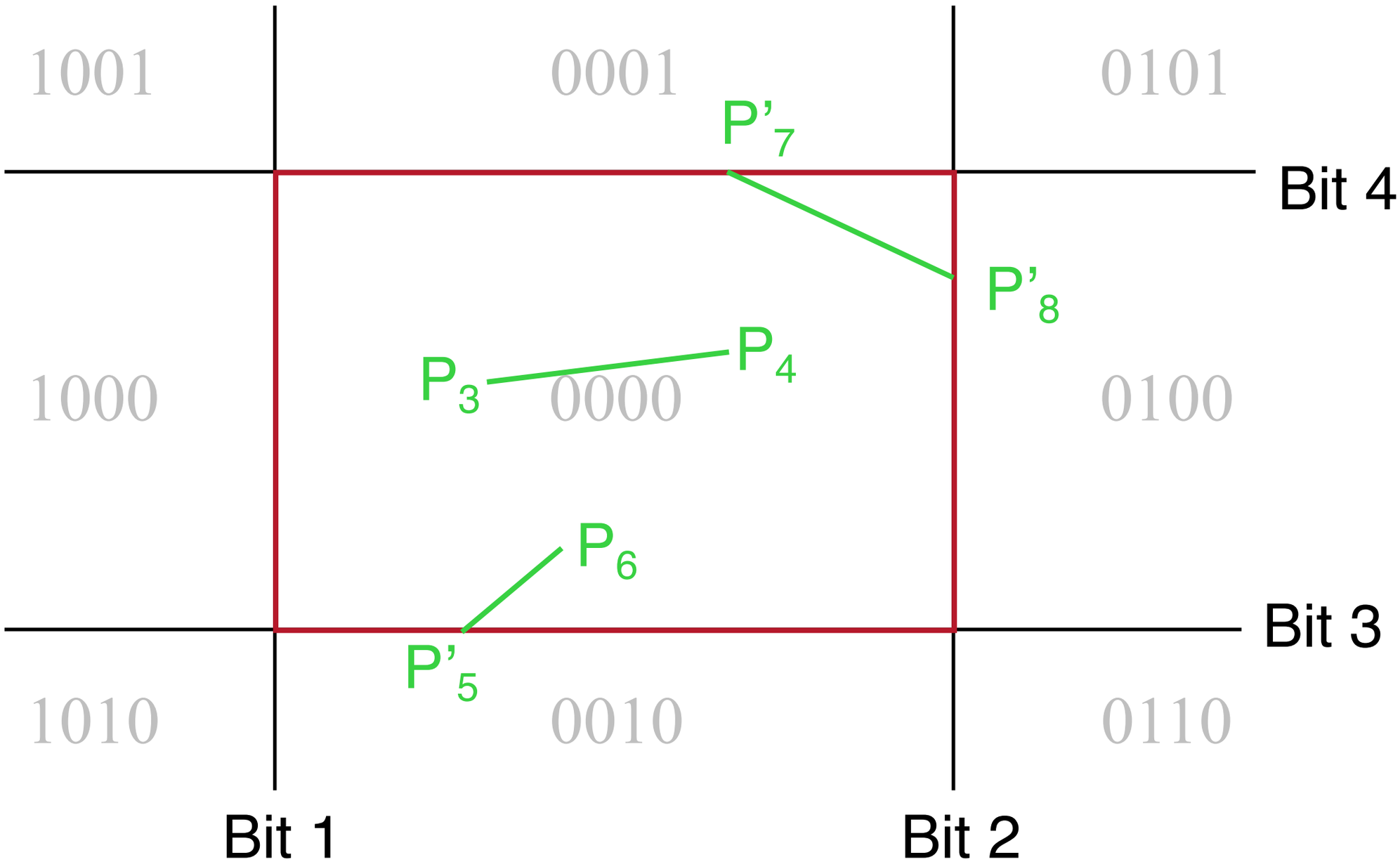
Cohen-Sutherland Line Clipping

- Before clipping



Cohen-Sutherland Line Clipping

- After clipping

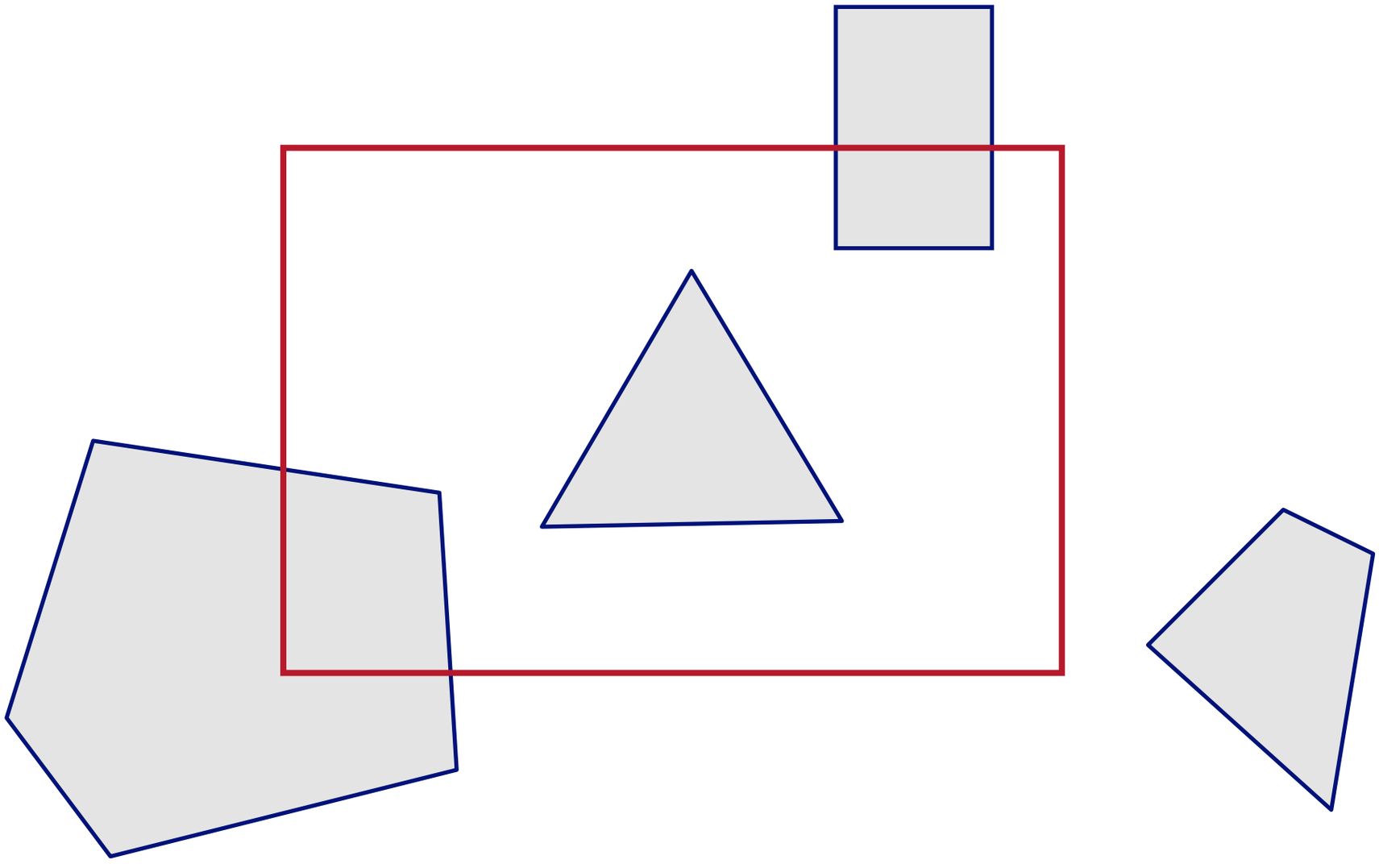


Clipping

- Avoid drawing parts of primitives outside window
  - Points
  - Line Segments
  - **Polygons**

Polygon Clipping

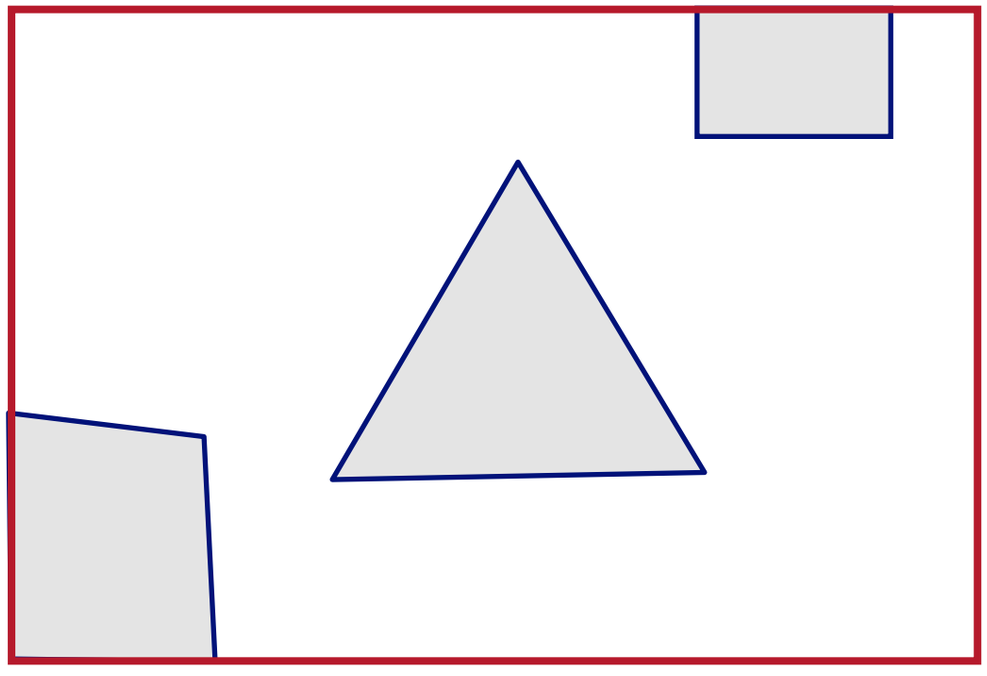
- Find the part of a polygon inside the clip window



Before Clipping

Polygon Clipping

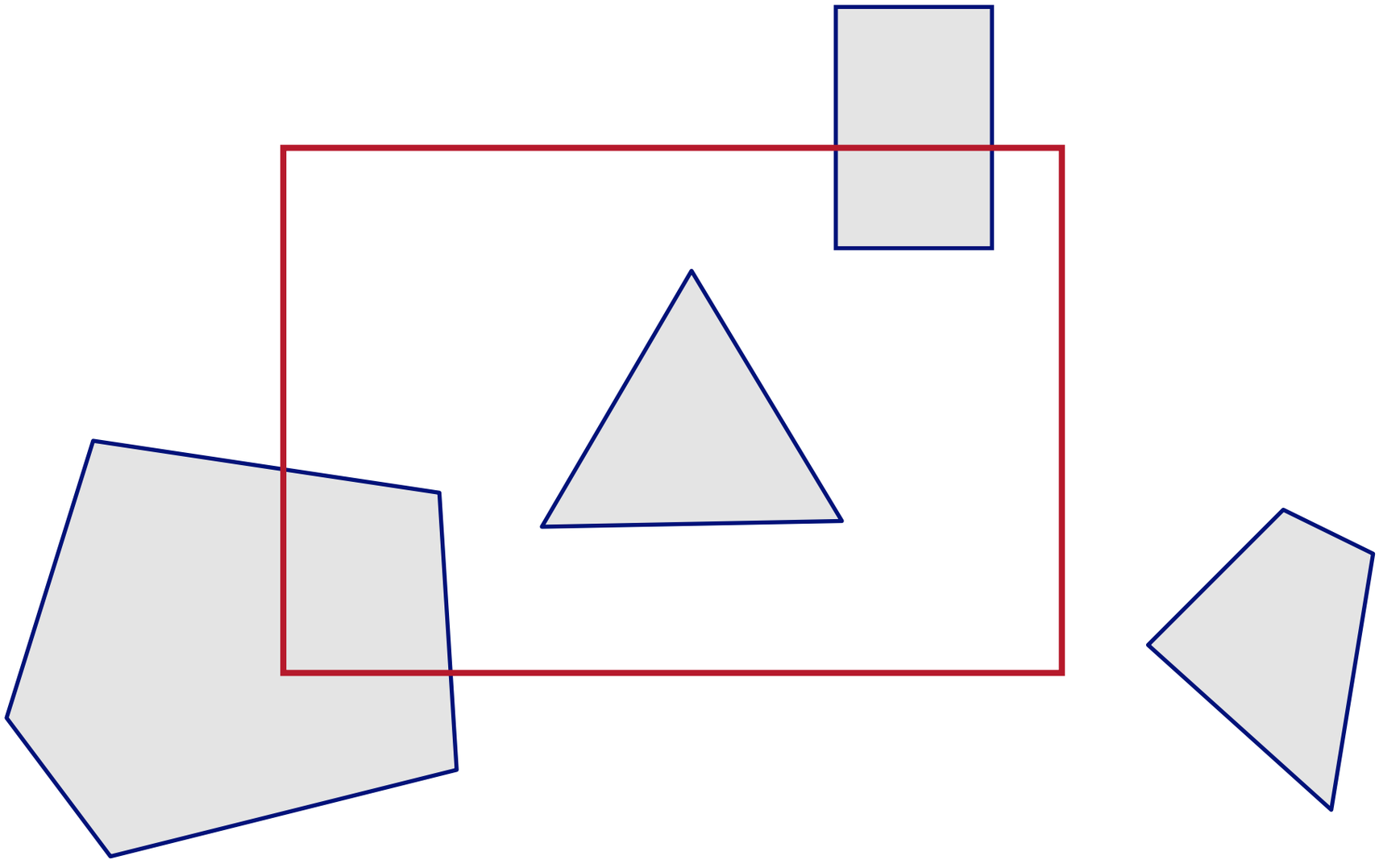
- Find the part of a polygon inside the clip window



After Clipping

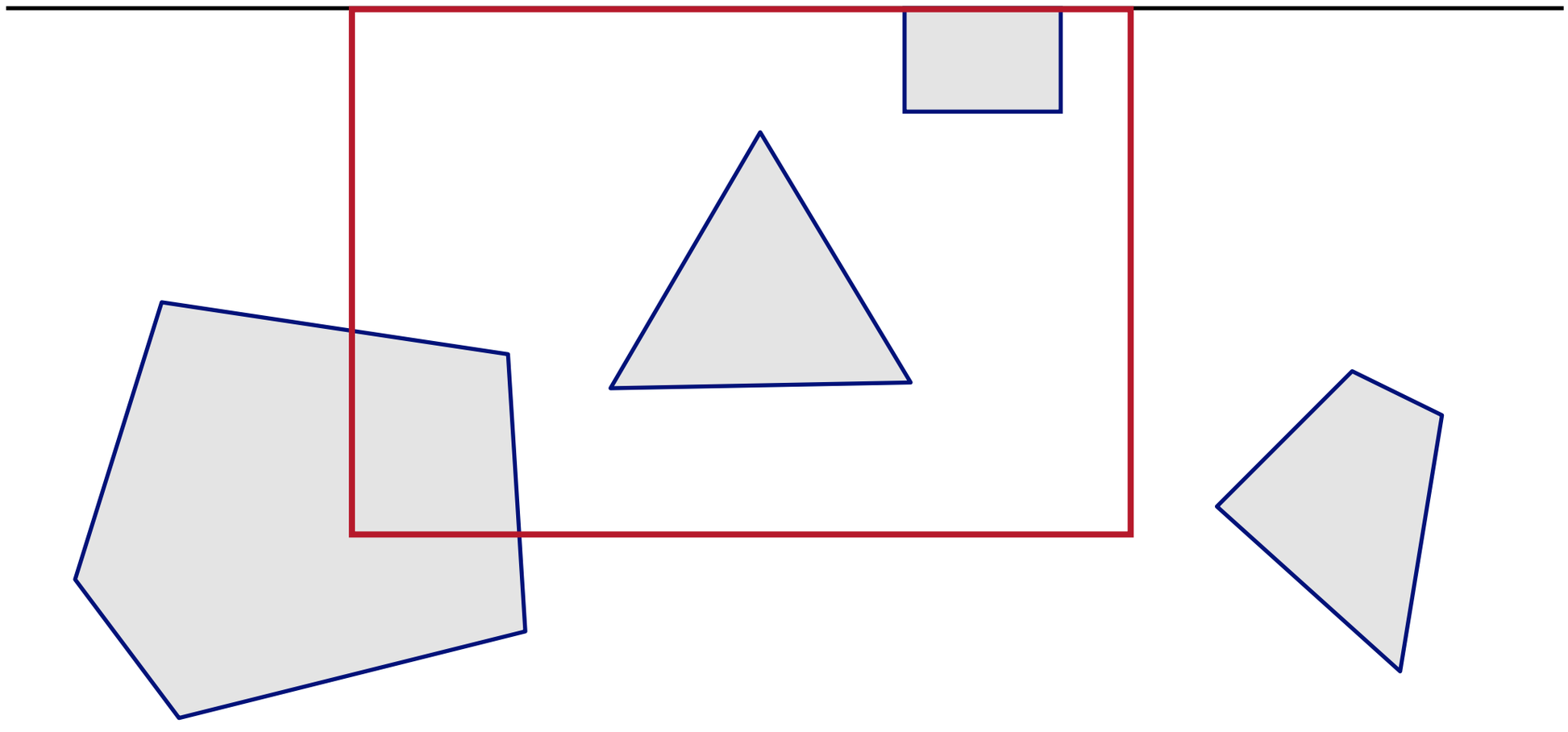
Sutherland-Hodgeman Clipping

- Clip to each window boundary one at a time



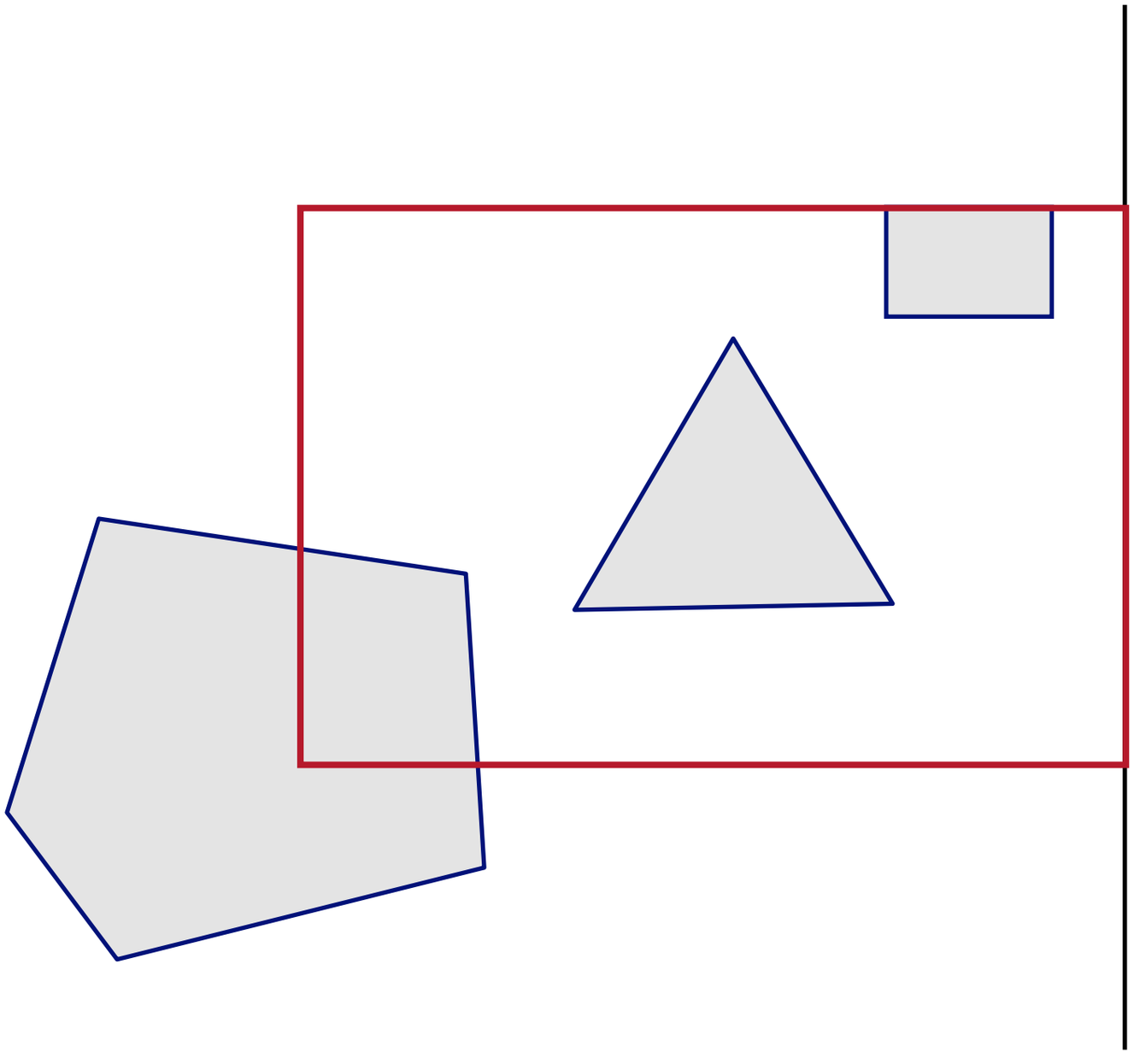
Sutherland-Hodgeman Clipping

- Clip to each window boundary one at a time



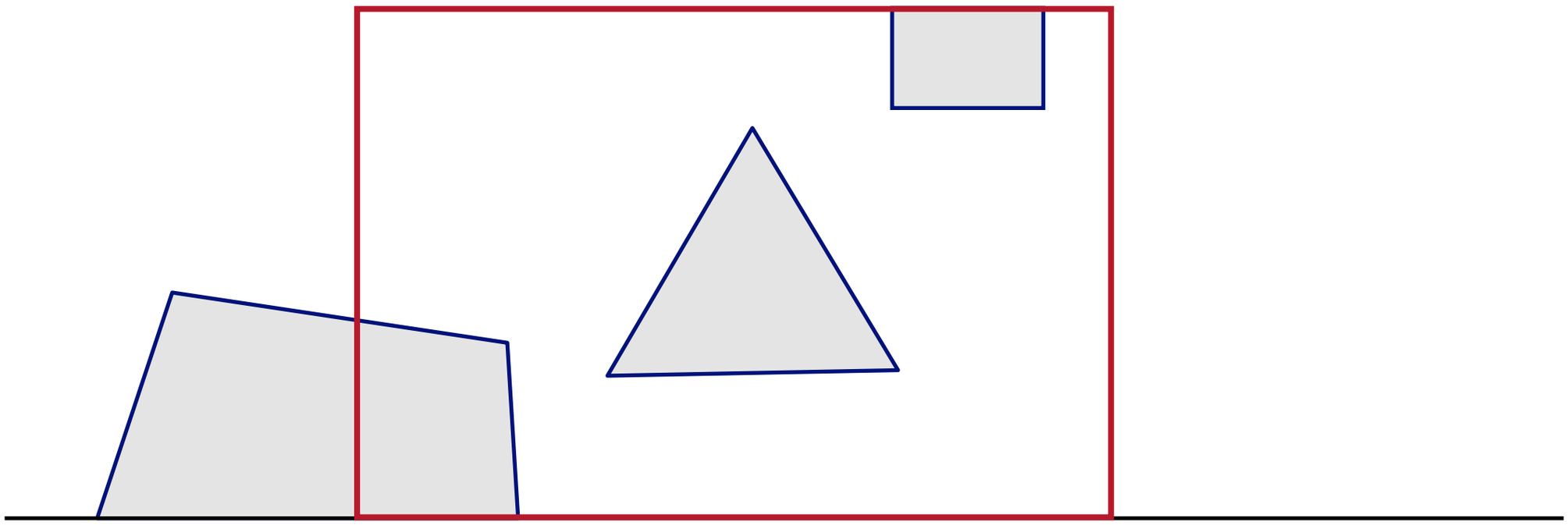
Sutherland-Hodgeman Clipping

- Clip to each window boundary one at a time



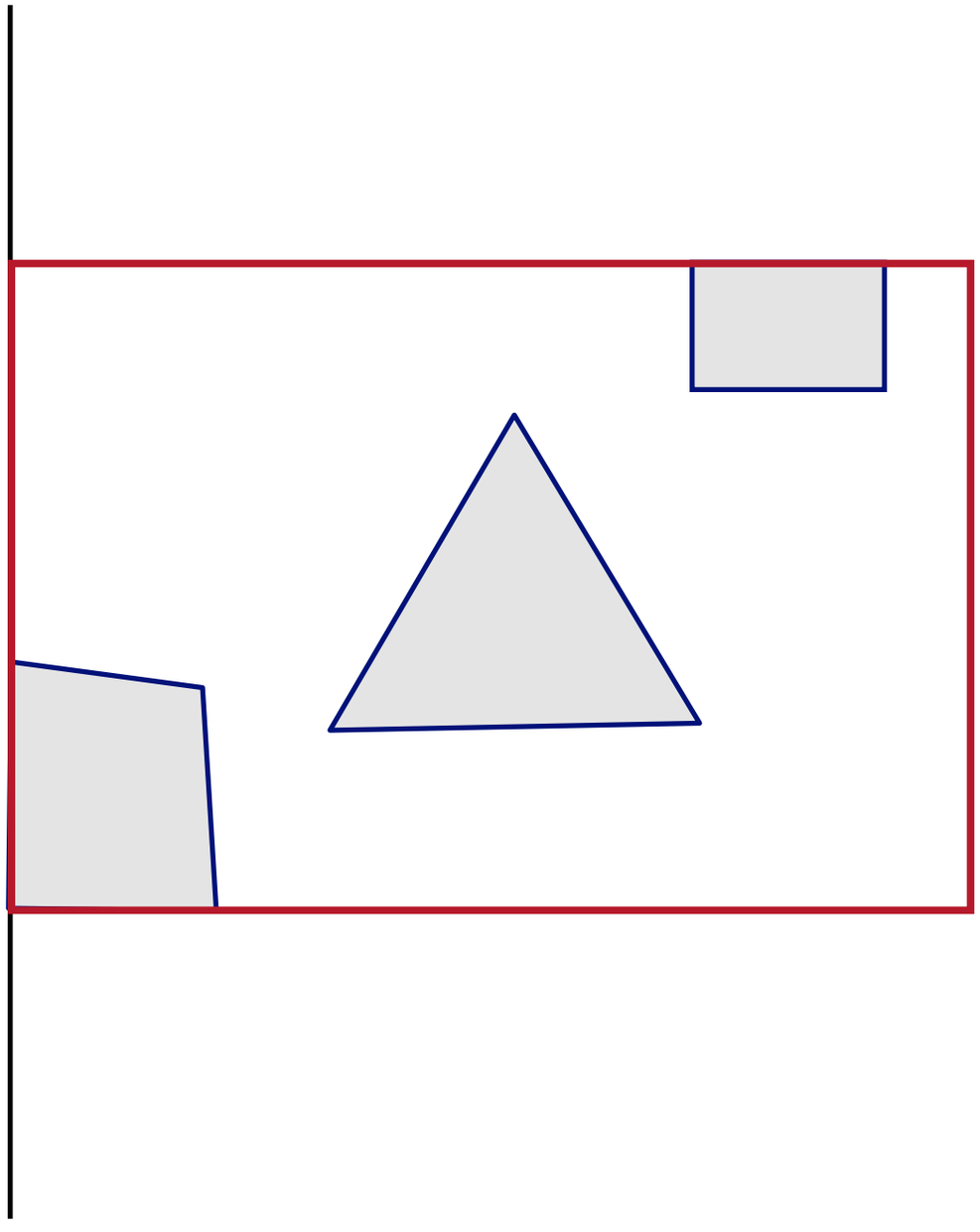
Sutherland-Hodgeman Clipping

- Clip to each window boundary one at a time



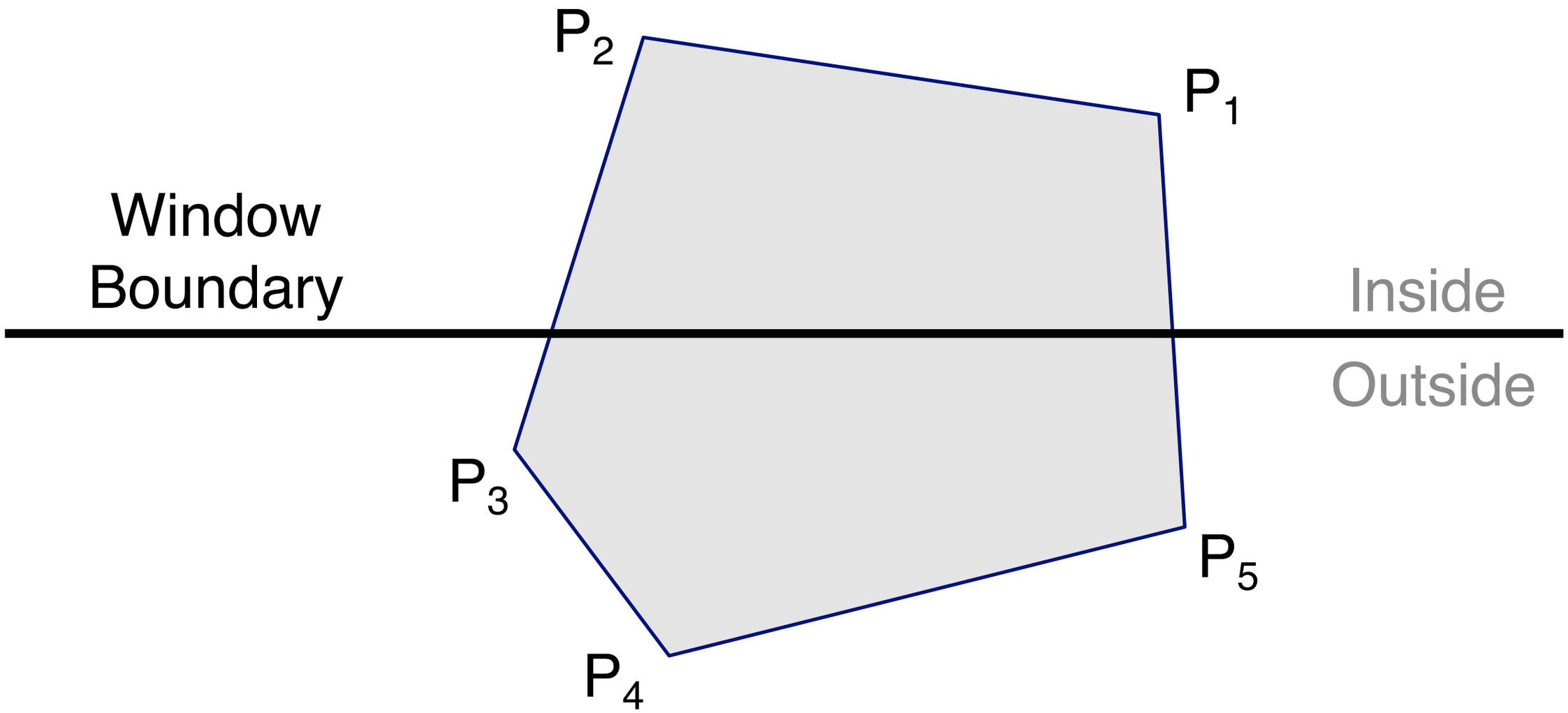
Sutherland-Hodgeman Clipping

- Clip to each window boundary one at a time



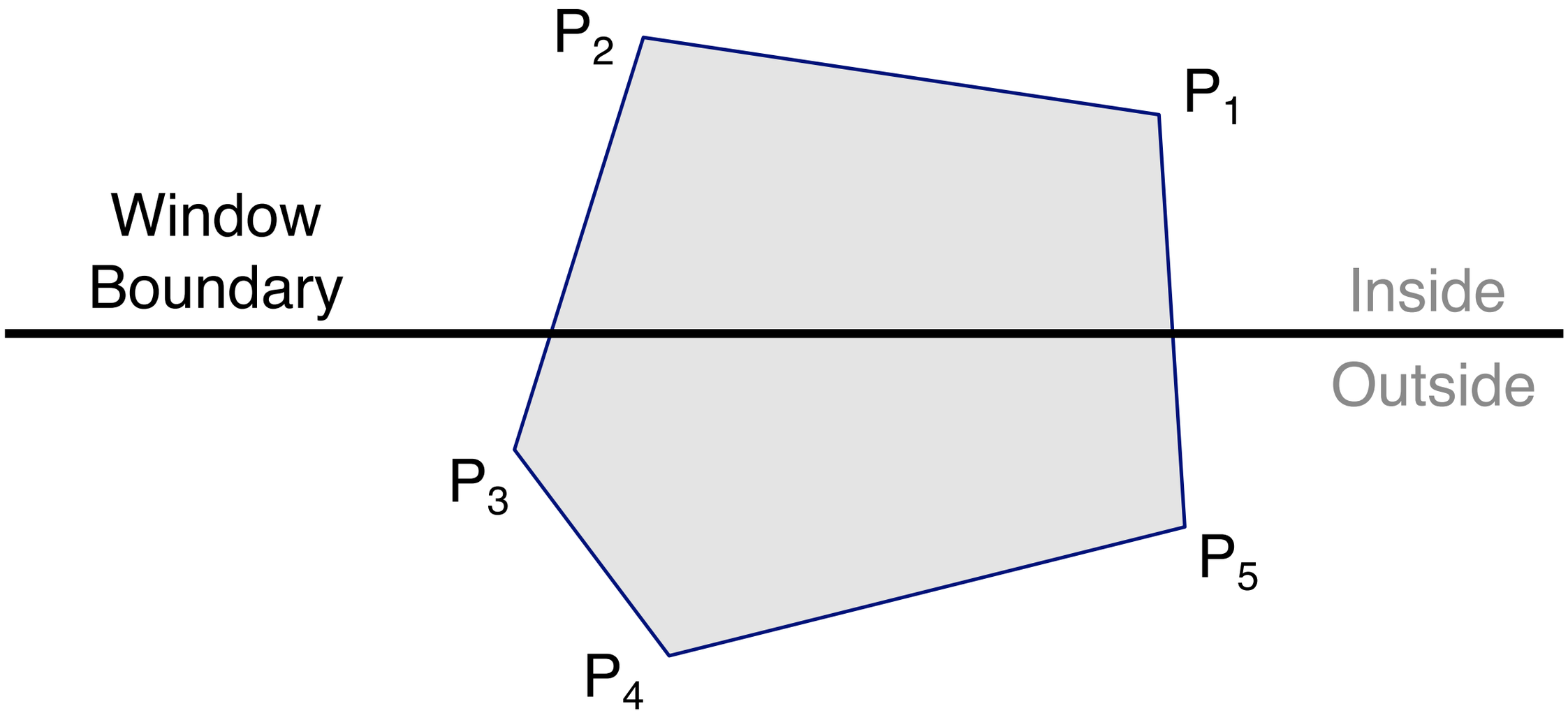
Sutherland-Hodgeman Clipping

- How do we clip a polygon with respect to a line?



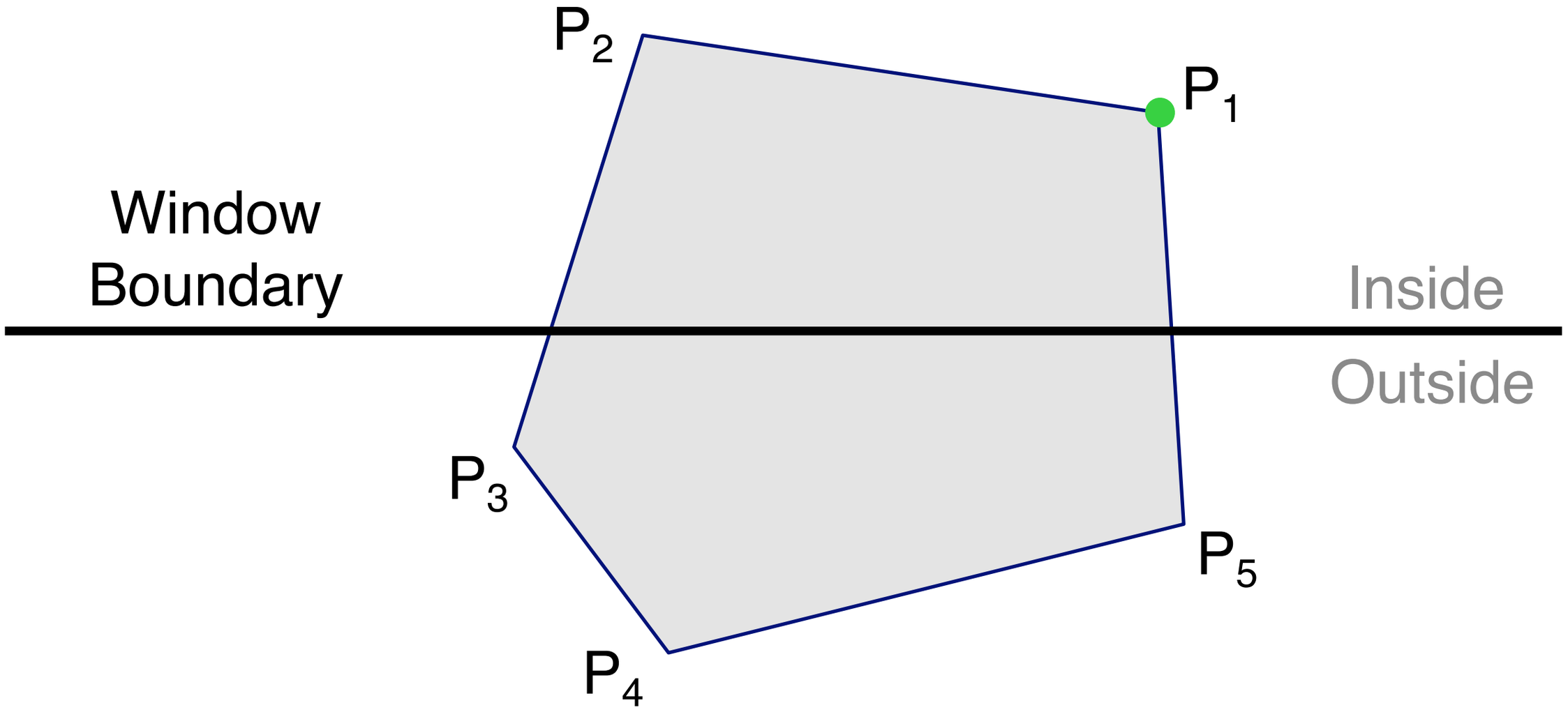
Sutherland-Hodgeman Clipping

- Do inside test for each point in sequence,
  Insert new points when cross window boundary,
  Remove points outside window boundary



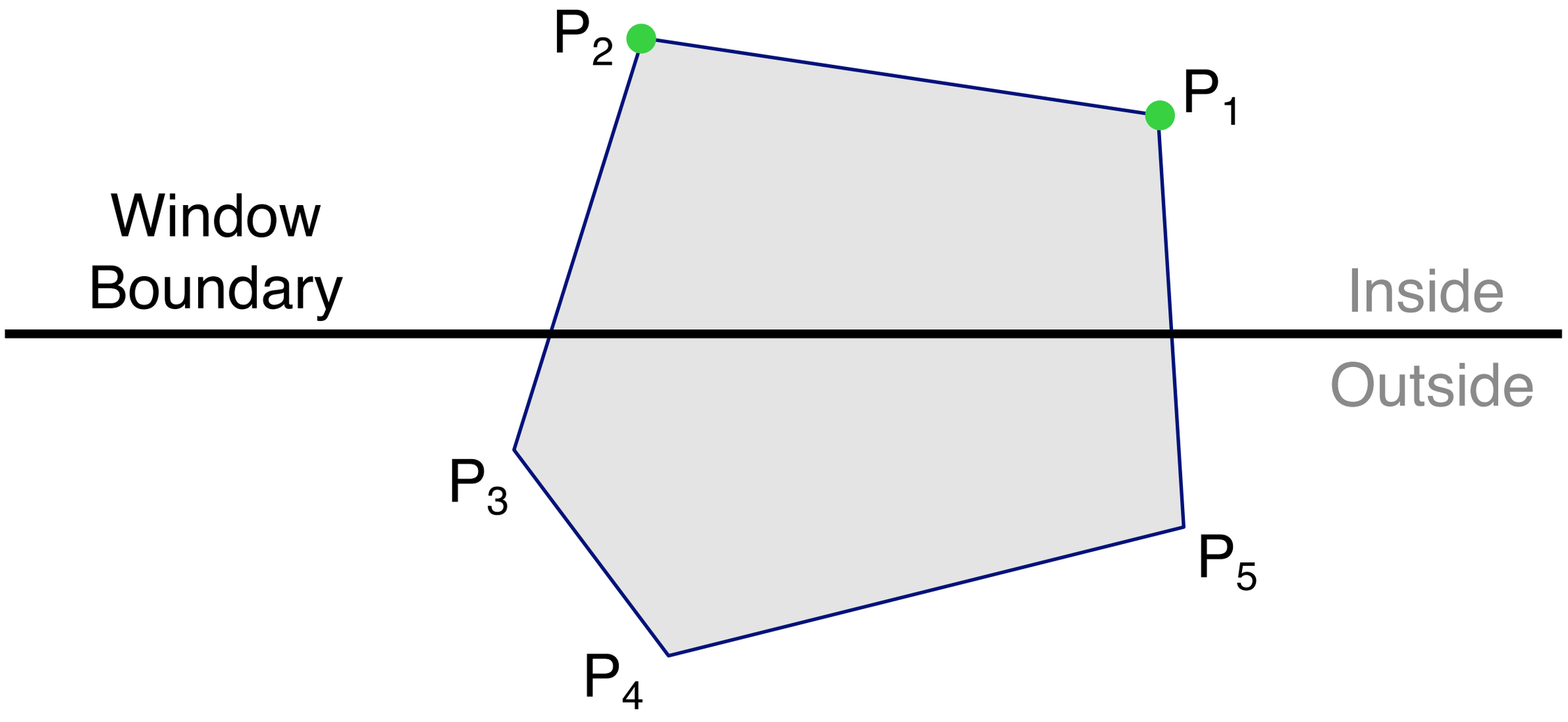
Sutherland-Hodgeman Clipping

- Do inside test for each point in sequence,
  Insert new points when cross window boundary,
  Remove points outside window boundary



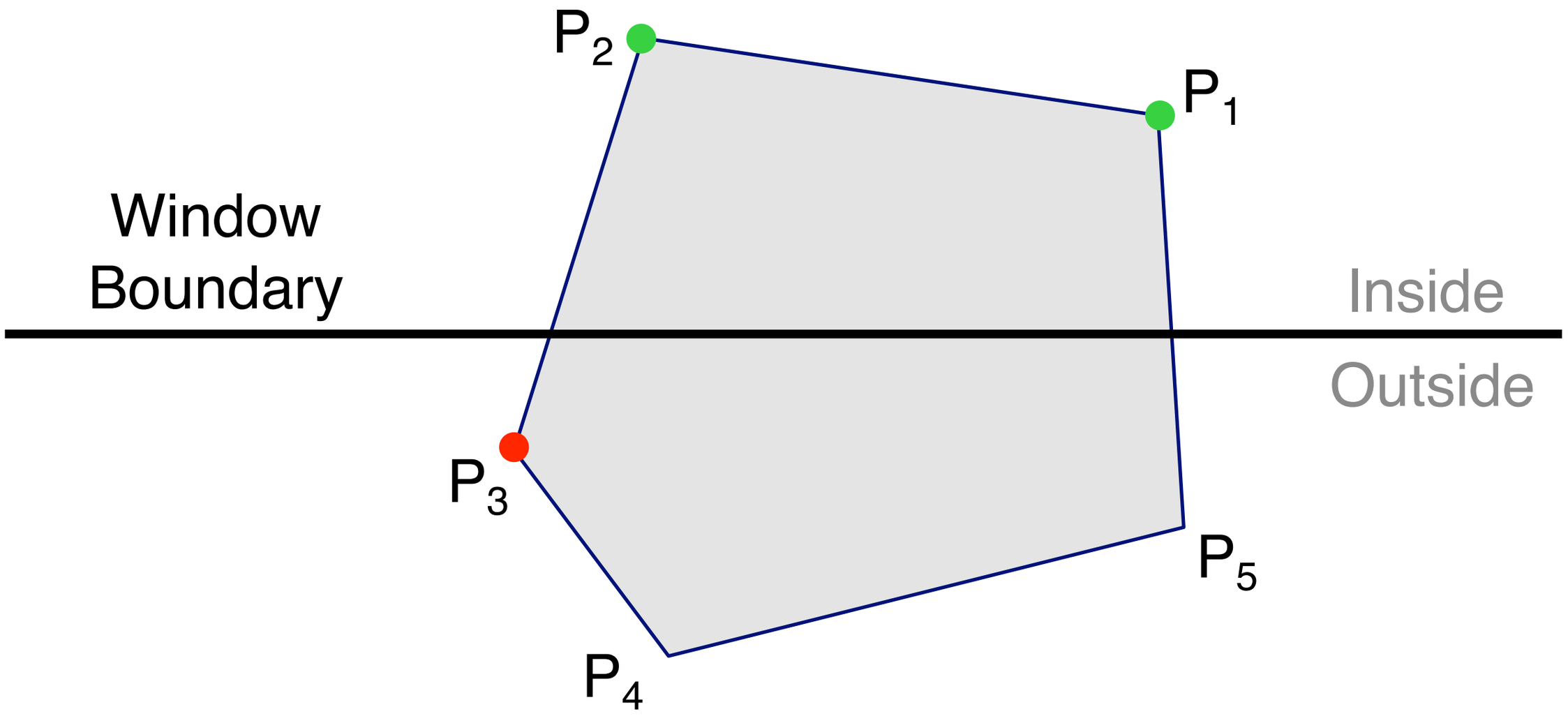
Sutherland-Hodgeman Clipping

- Do inside test for each point in sequence,
  Insert new points when cross window boundary,
  Remove points outside window boundary



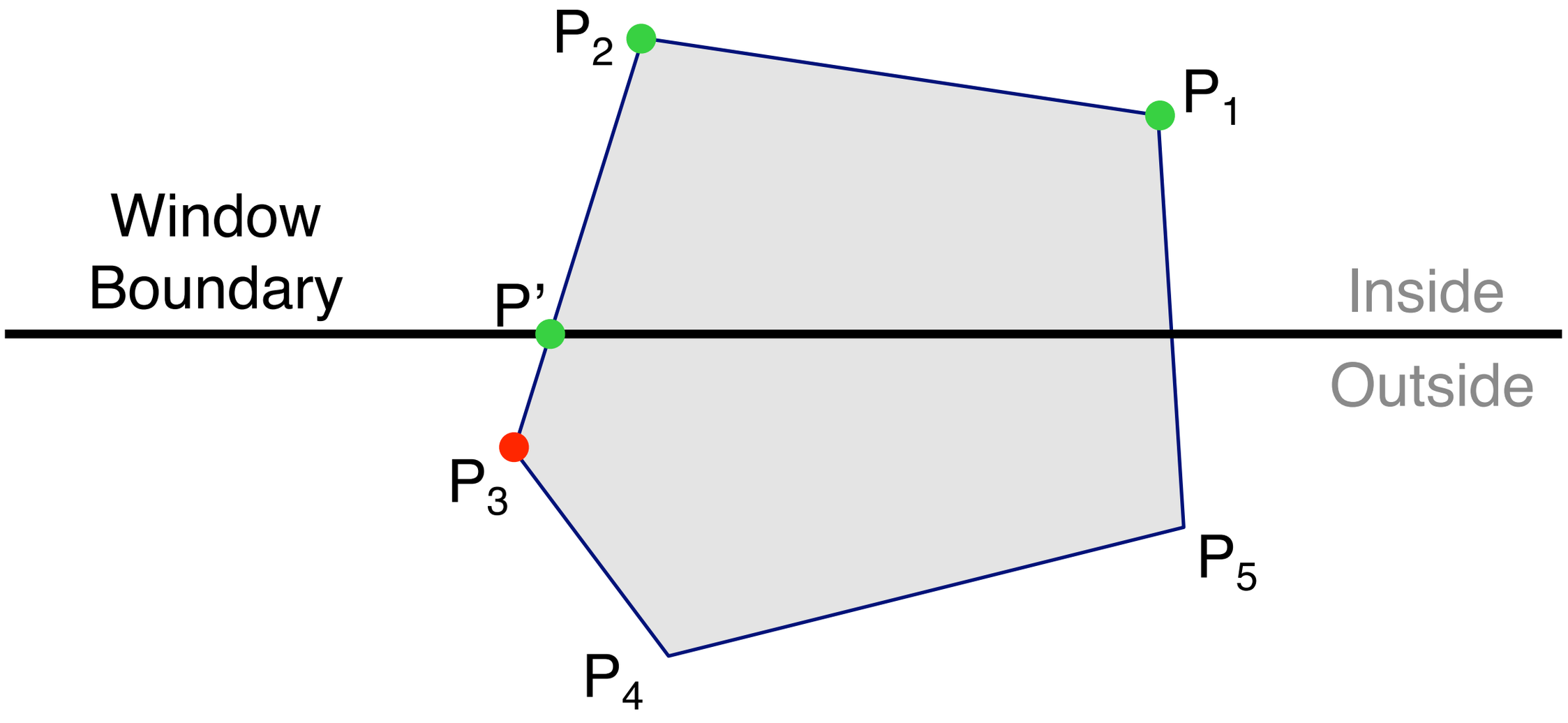
Sutherland-Hodgeman Clipping

- Do inside test for each point in sequence,
  Insert new points when cross window boundary,
  Remove points outside window boundary



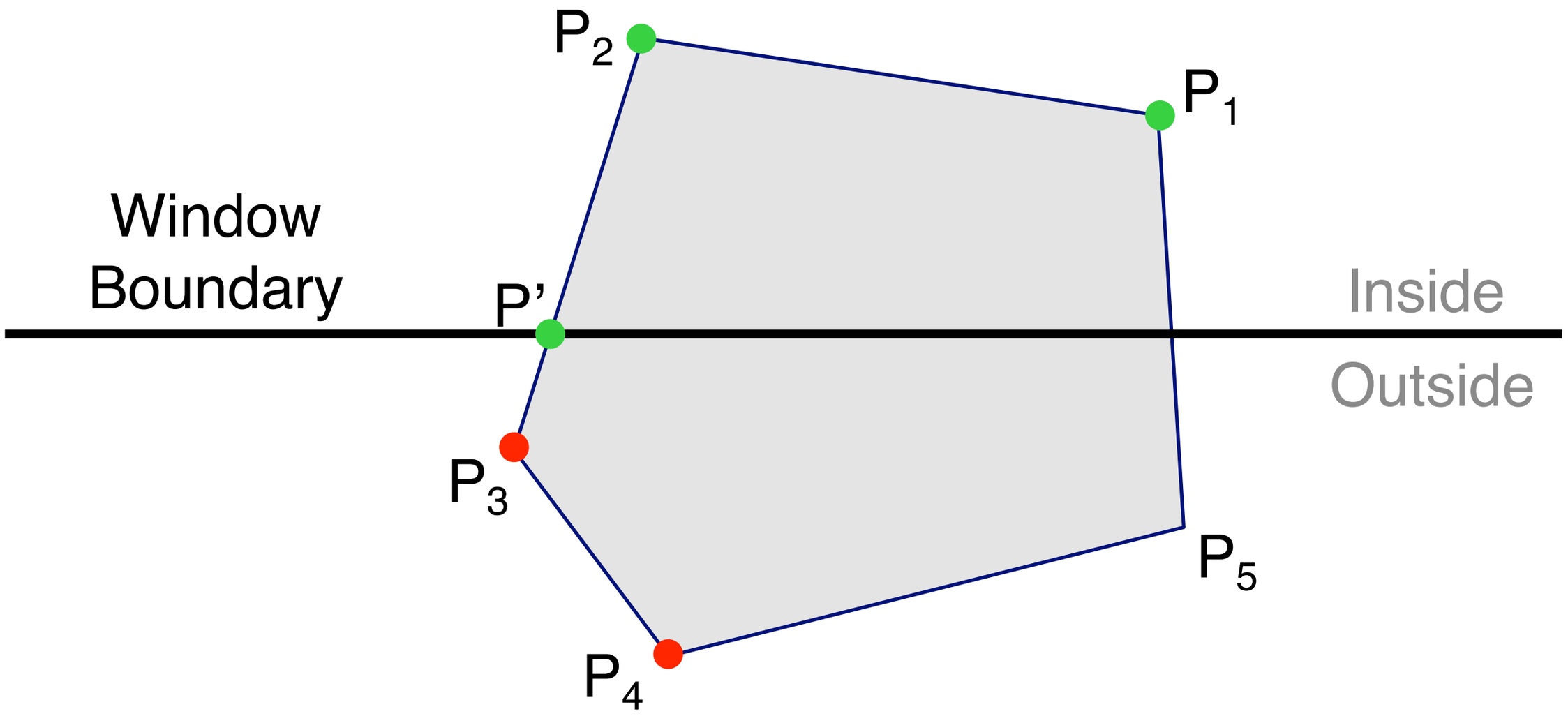
Sutherland-Hodgeman Clipping

- Do inside test for each point in sequence,
  Insert new points when cross window boundary,
  Remove points outside window boundary



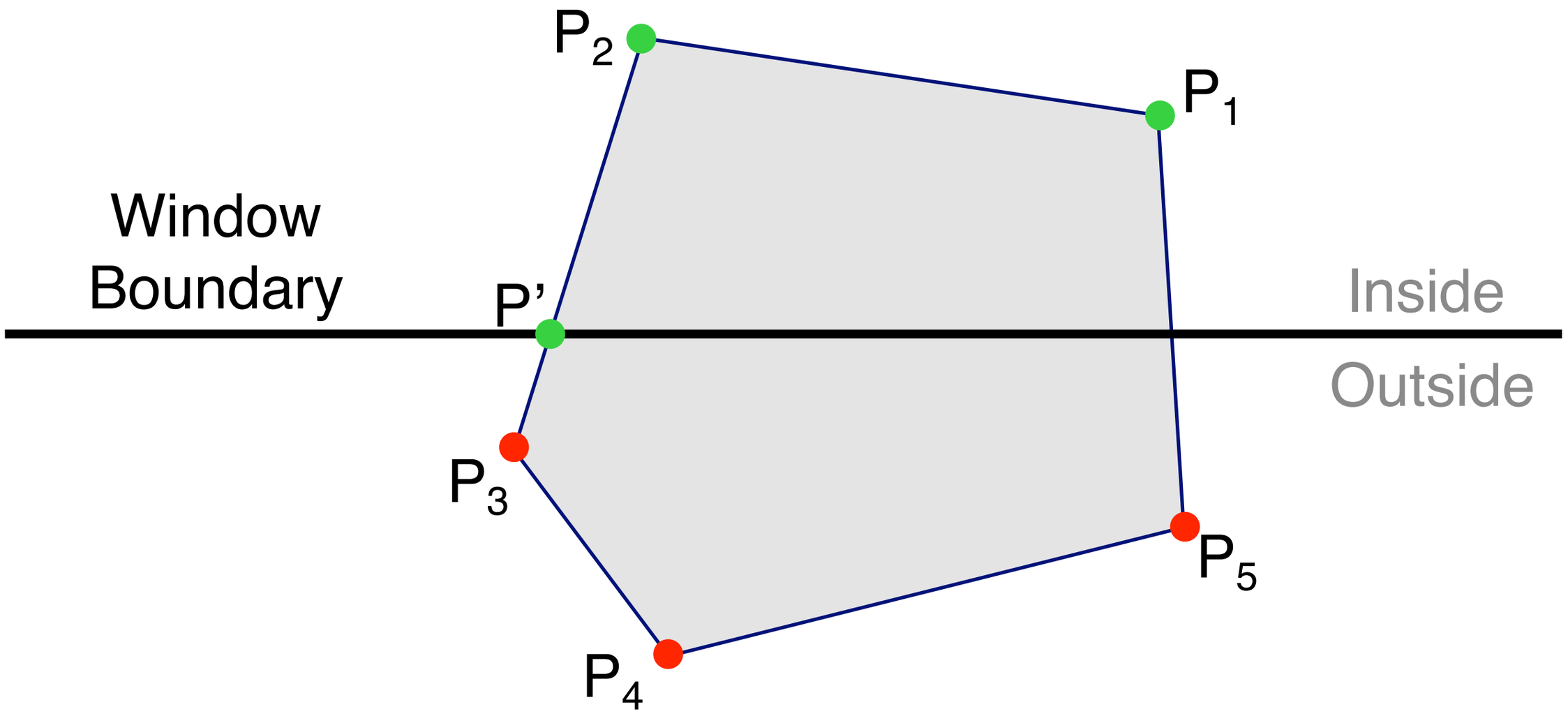
Sutherland-Hodgeman Clipping

- Do inside test for each point in sequence,
  Insert new points when cross window boundary,
  Remove points outside window boundary



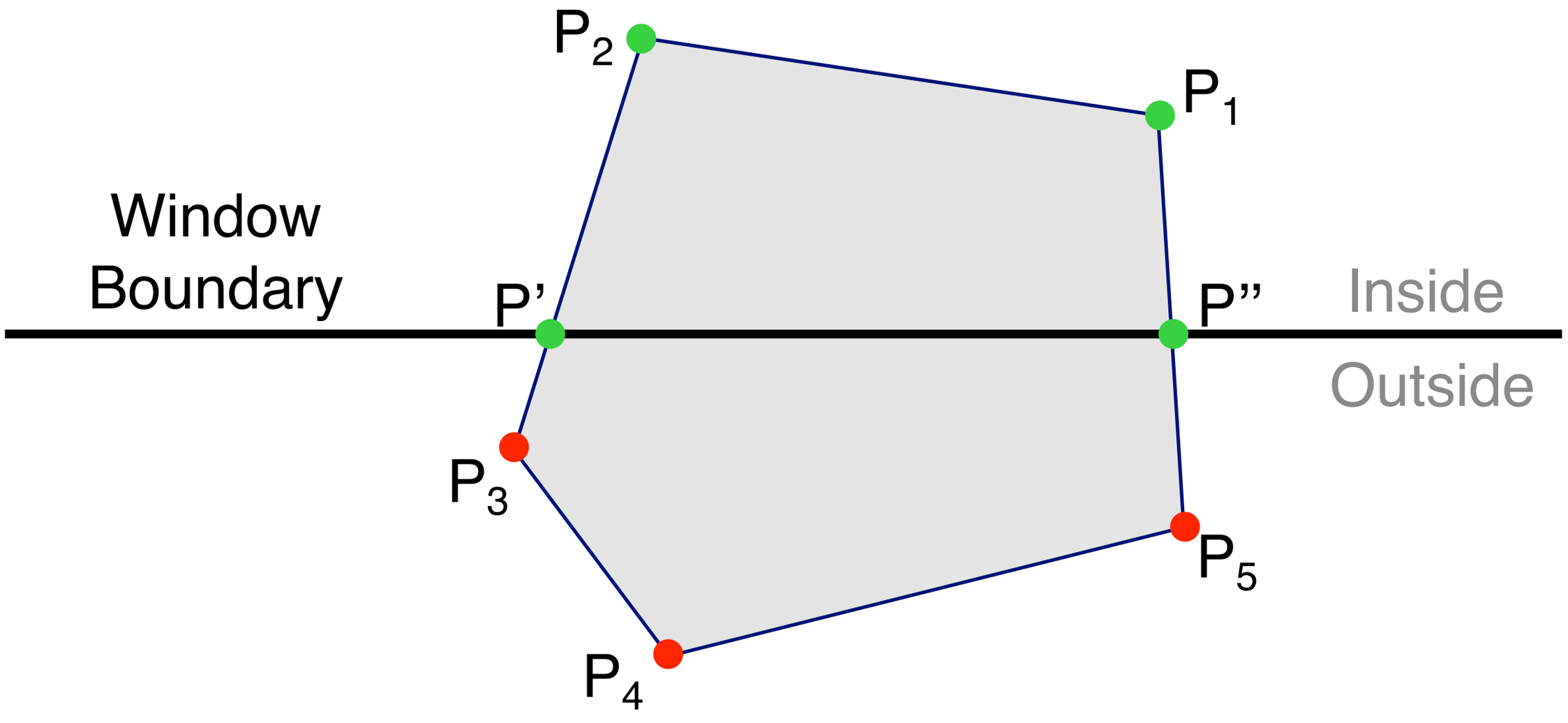
Sutherland-Hodgeman Clipping

- Do inside test for each point in sequence,
  Insert new points when cross window boundary,
  Remove points outside window boundary



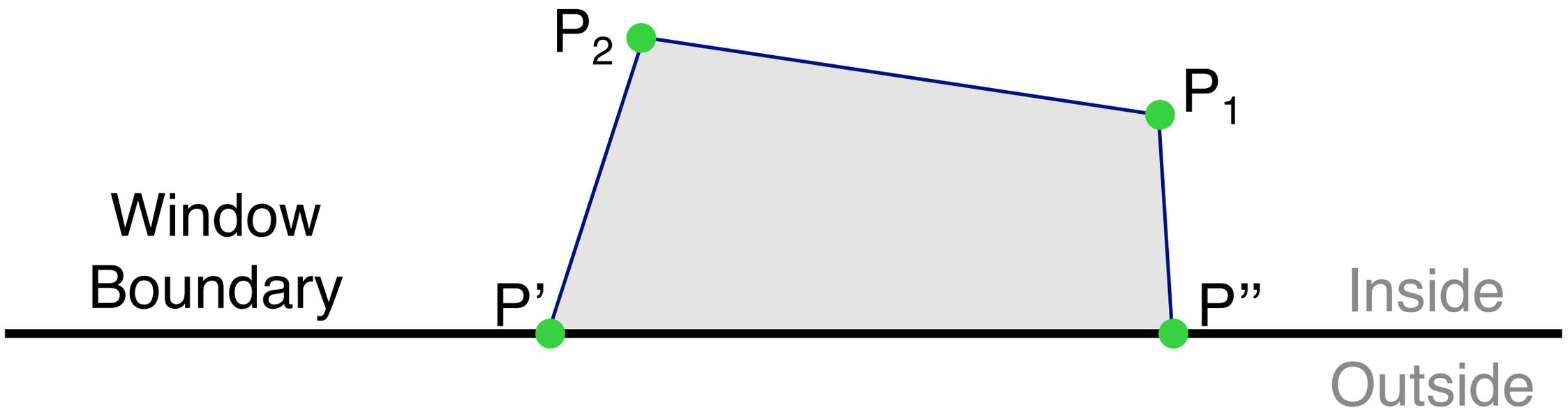
Sutherland-Hodgeman Clipping

- Do inside test for each point in sequence,
  Insert new points when cross window boundary,
  Remove points outside window boundary

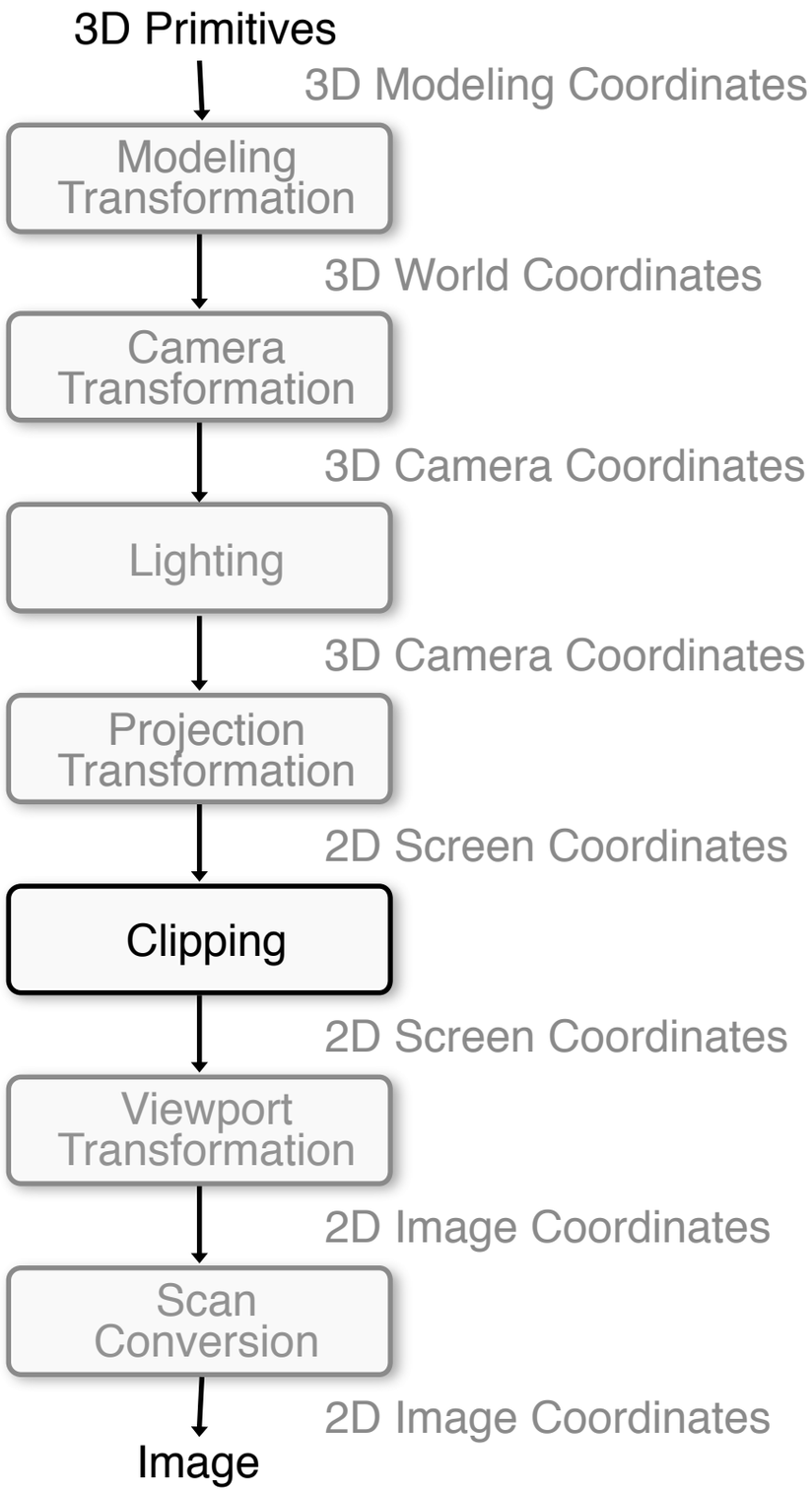


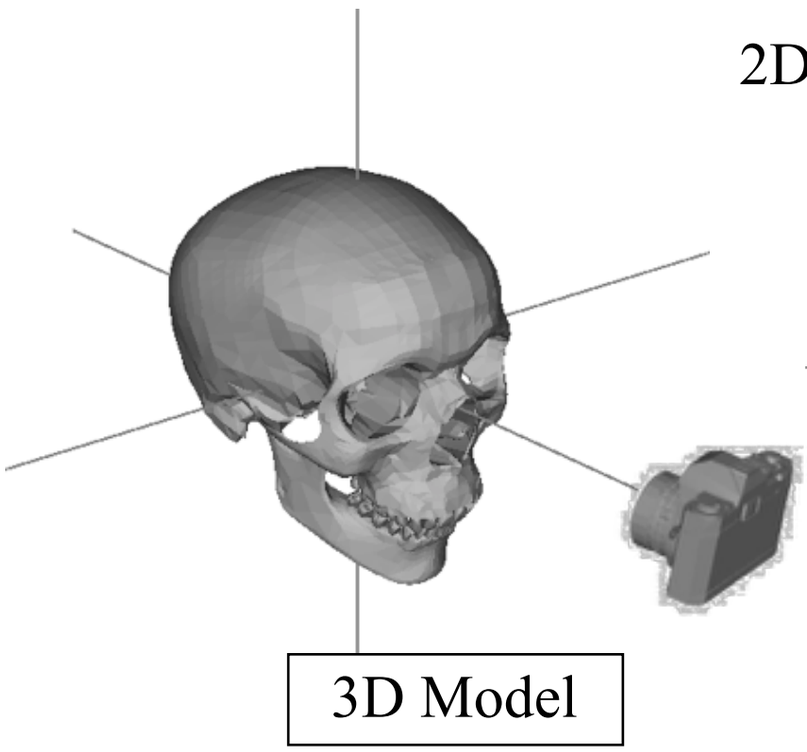
Sutherland-Hodgeman Clipping

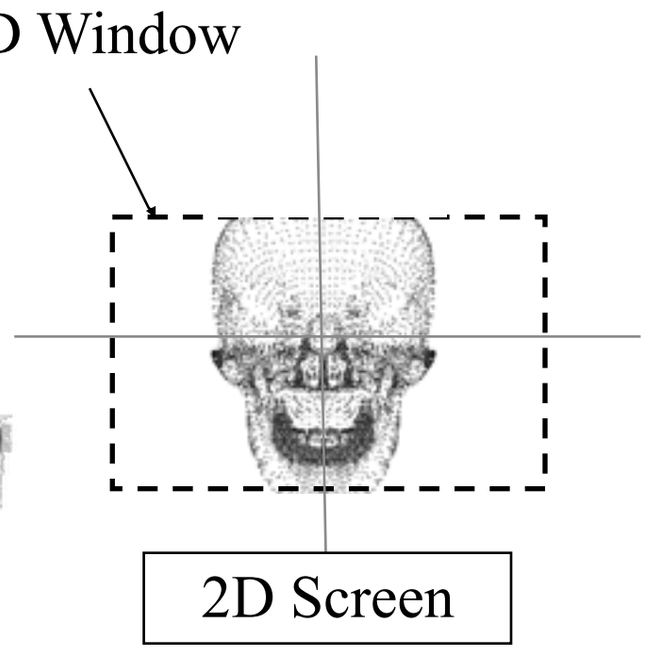
- Do inside test for each point in sequence,
  Insert new points when cross window boundary,
  Remove points outside window boundary



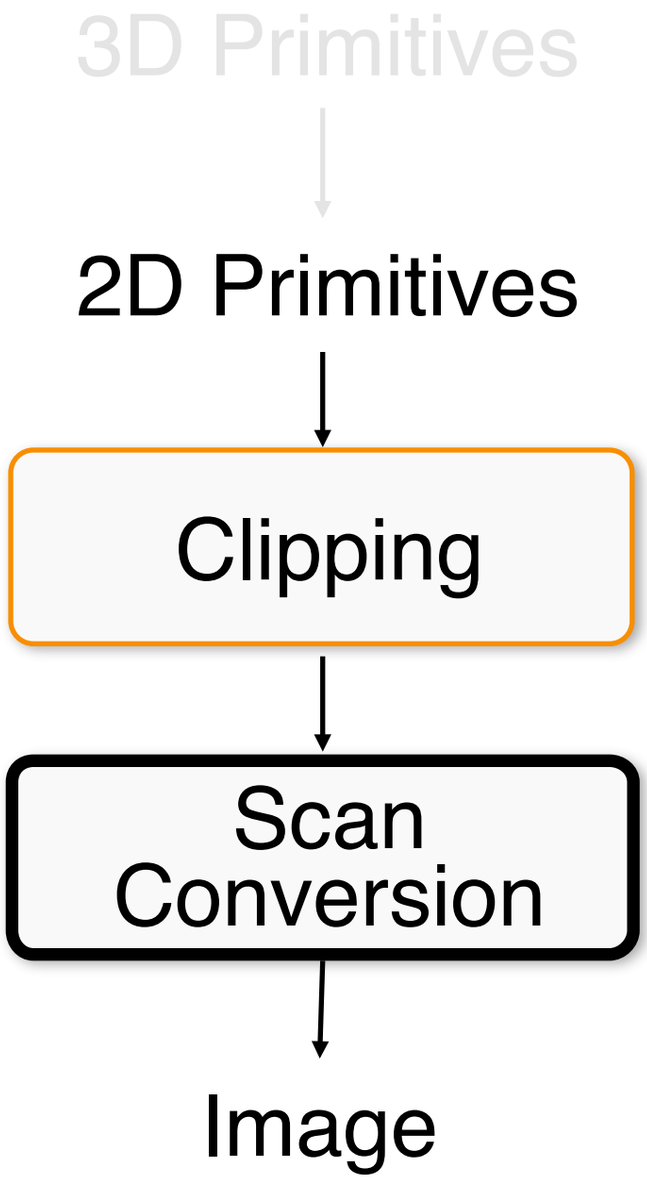
3D Rendering Pipeline (for direct illumination)







2D Rendering Pipeline



Clip portions of geometric primitives residing outside the window

Fill pixels representing primitives in screen coordinates

Overview

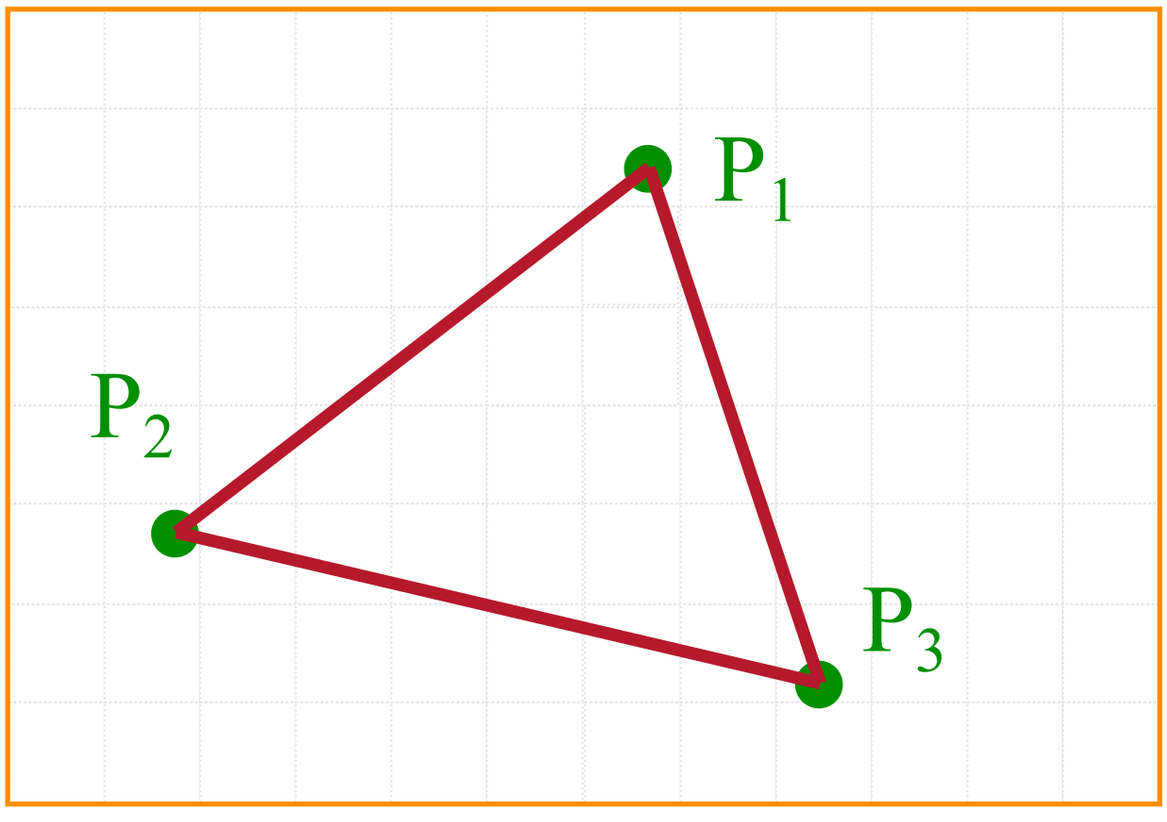
- Scan conversion
  - Figure out which pixels to fill
- Shading
  - Determine a color for each filled pixel
- Depth test
  - Determine when the color of a pixel should be overwritten

Scan Conversion

- Render an image of a geometric primitive by setting pixel colors

```
void SetPixel(int x, int y, Color rgba)
```

- Example: Filling the inside of a triangle

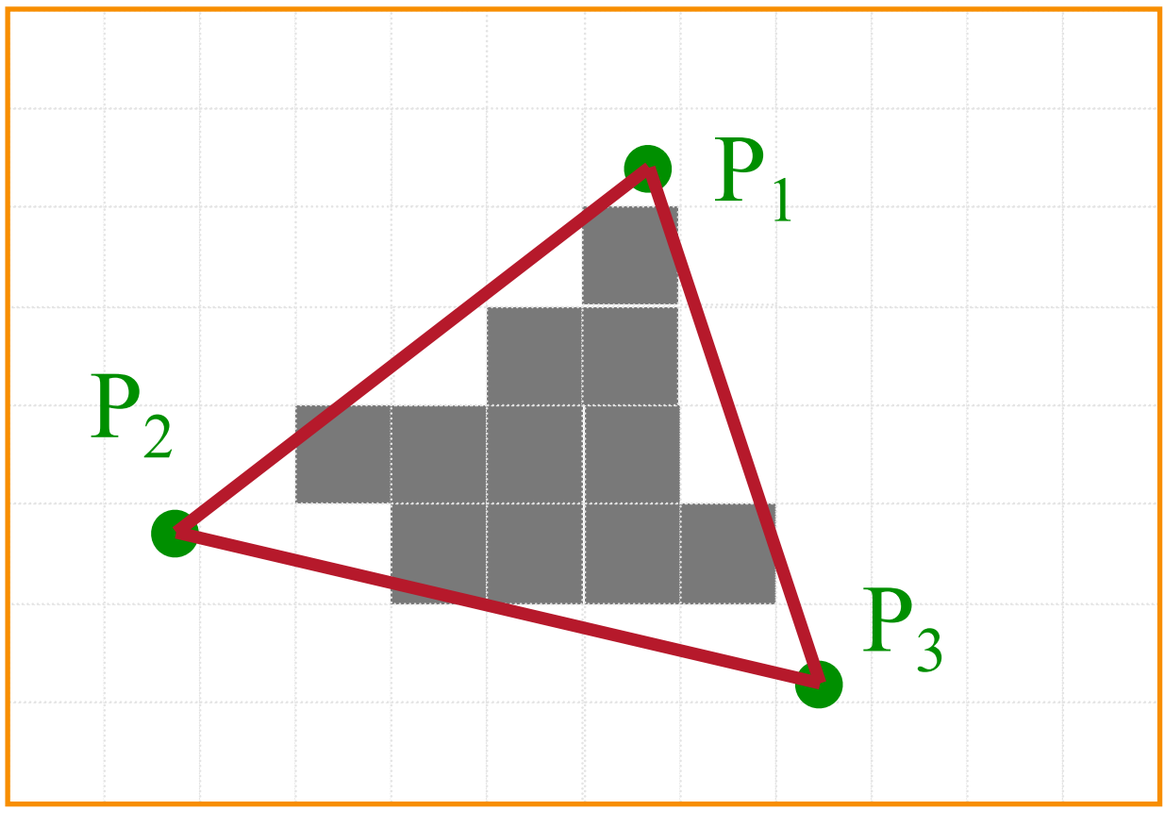


Scan Conversion

- Render an image of a geometric primitive by setting pixel colors

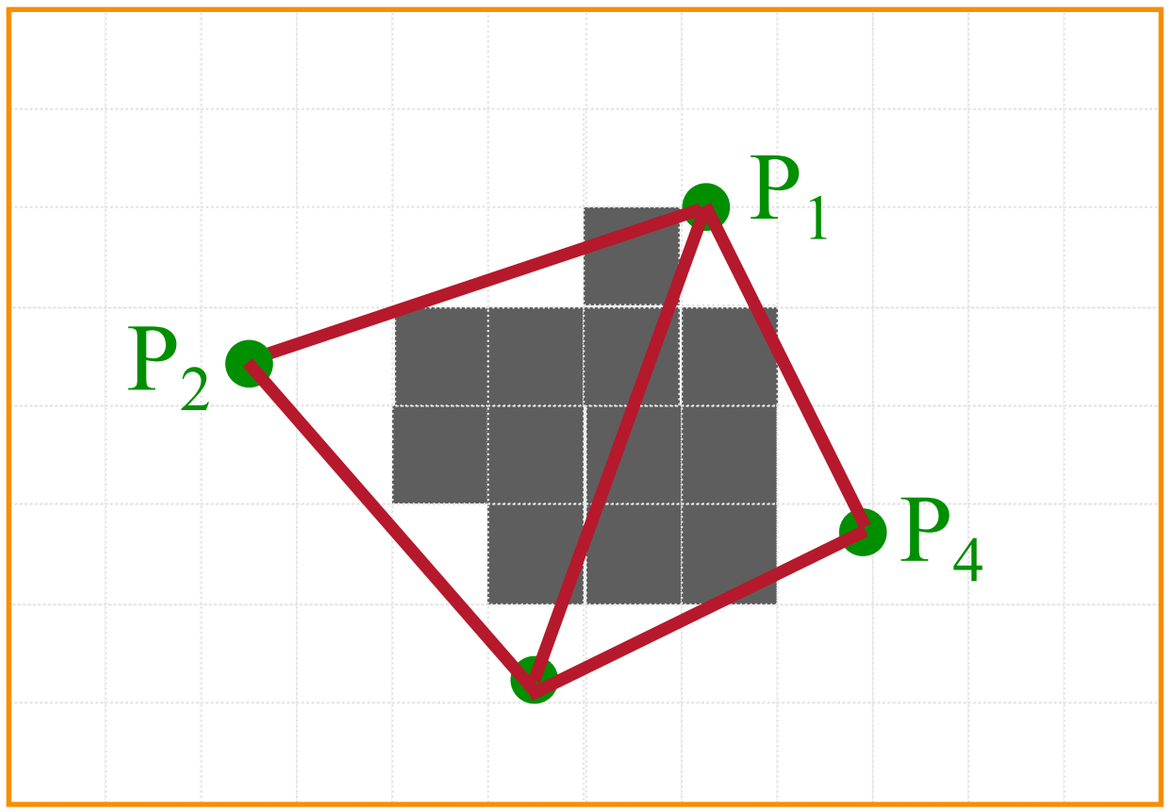
```
void SetPixel(int x, int y, Color rgba)
```

- Example: Filling the inside of a triangle



Triangle Scan Conversion

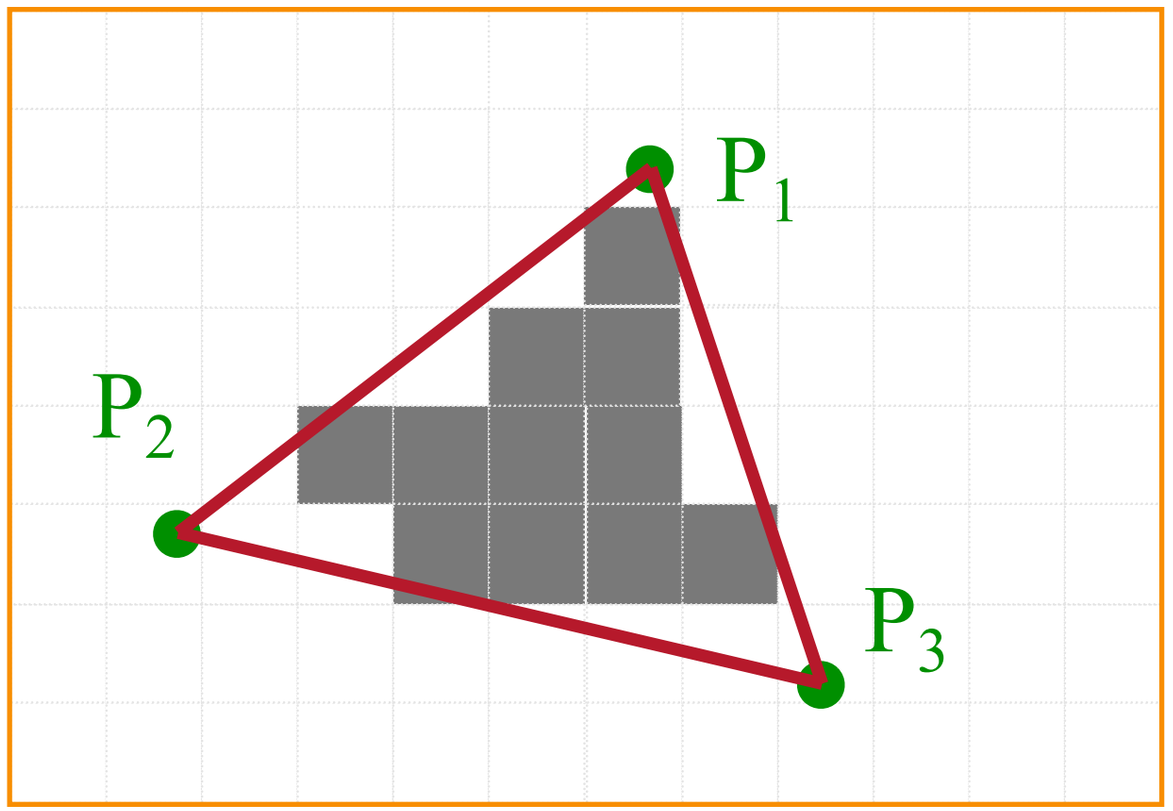
- Properties of a good algorithm
  - MUST BE FAST!
  - No cracks between adjacent primitives



Simple Algorithm

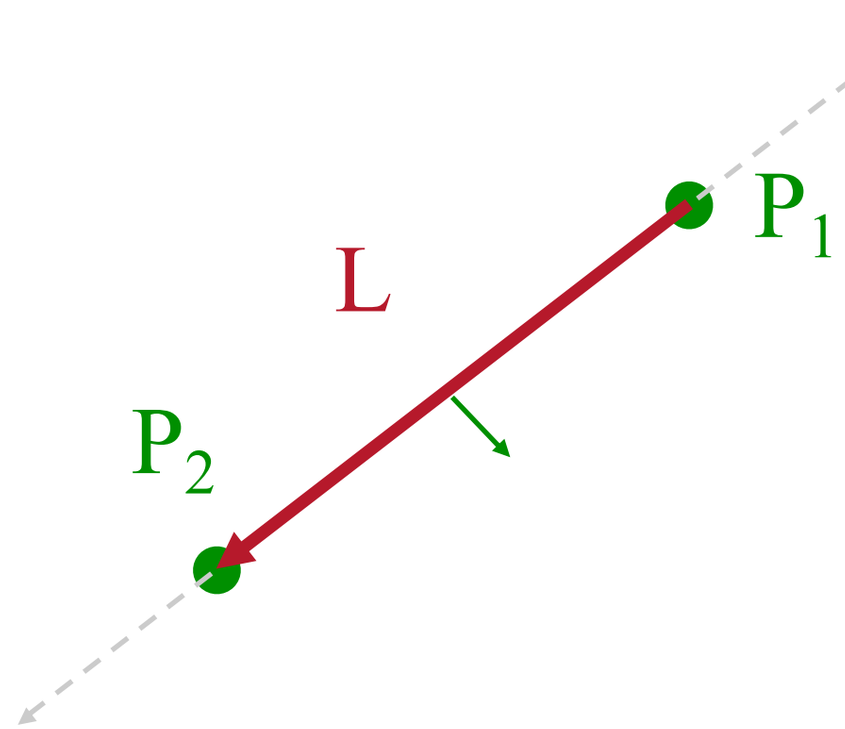
- Color all pixels inside triangle

```
void ScanTriangle(Triangle T, Color rgba){
   for each pixel P at (x,y){
      if (Inside(T, P))
         SetPixel(x, y, rgba);
   }
}
```



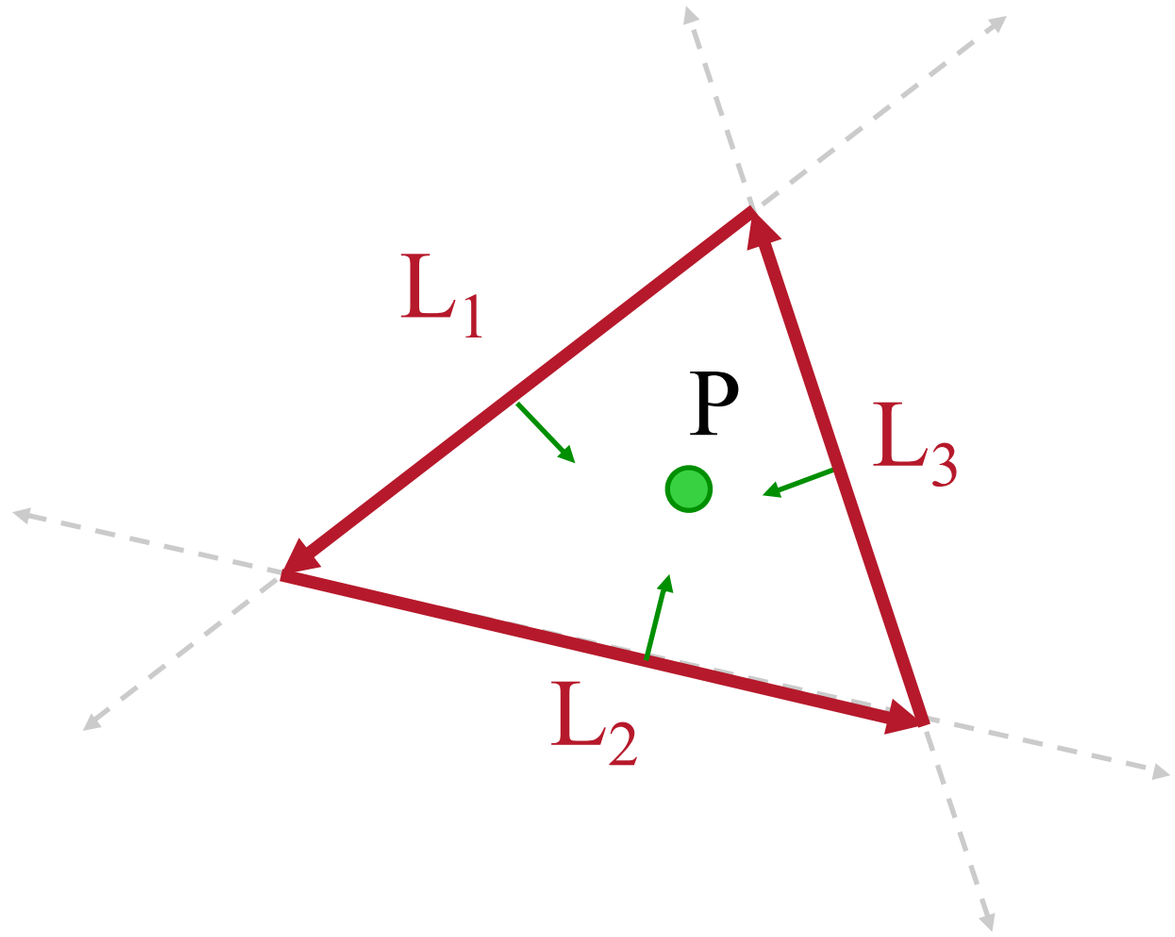
Line defines two halfspaces

- Test: use implicit equation for a line
  - On line: $ax + by + c = 0$
  - On right: $ax + by + c < 0$
  - On left: $ax + by + c > 0$



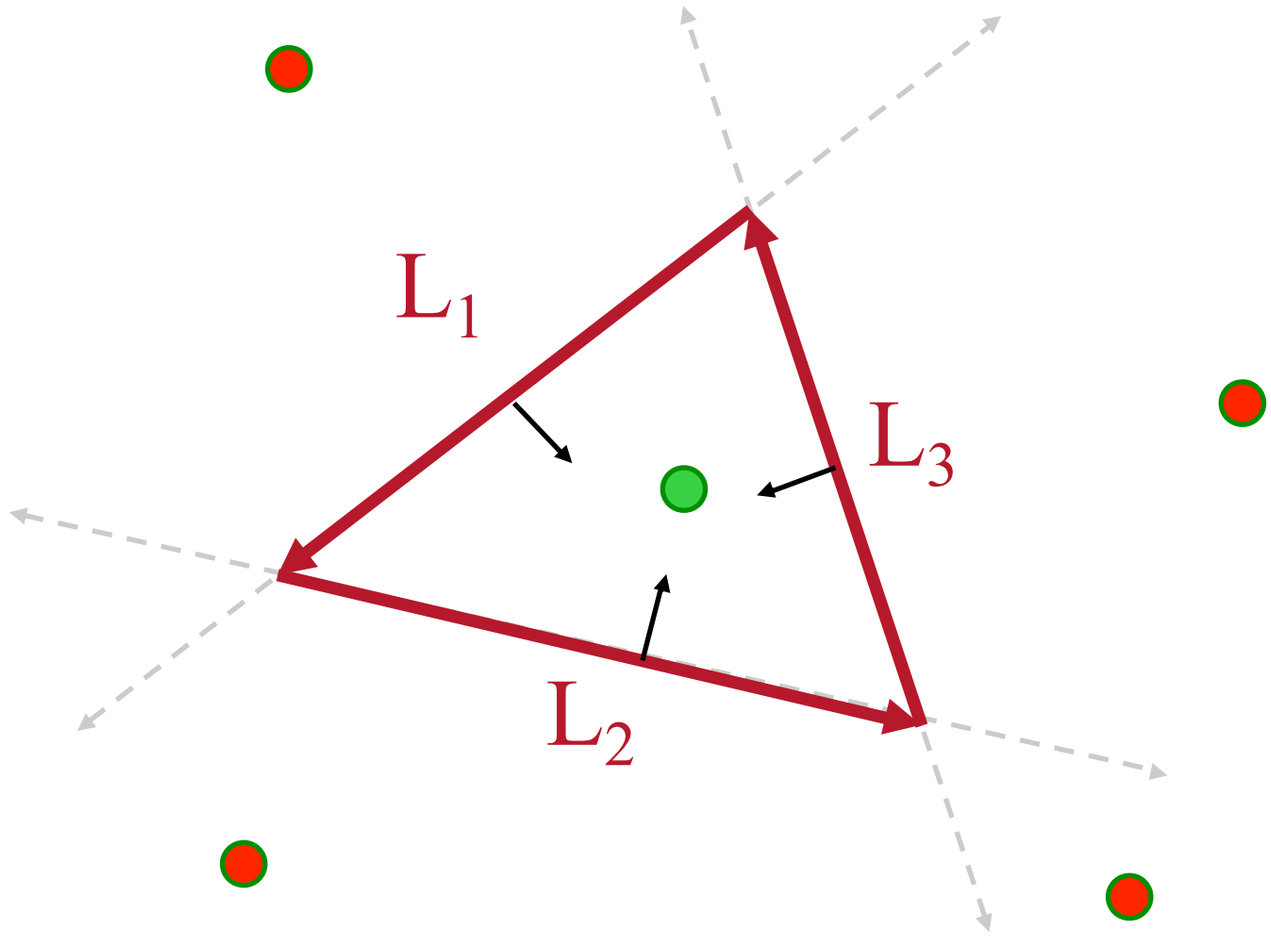
Inside Triangle Test

- A point is inside a triangle if it is in the positive half-space of all three boundary lines
  - Triangle vertices are ordered counter-clockwise
  - Point must be on the left side of every boundary line



Inside Triangle Test

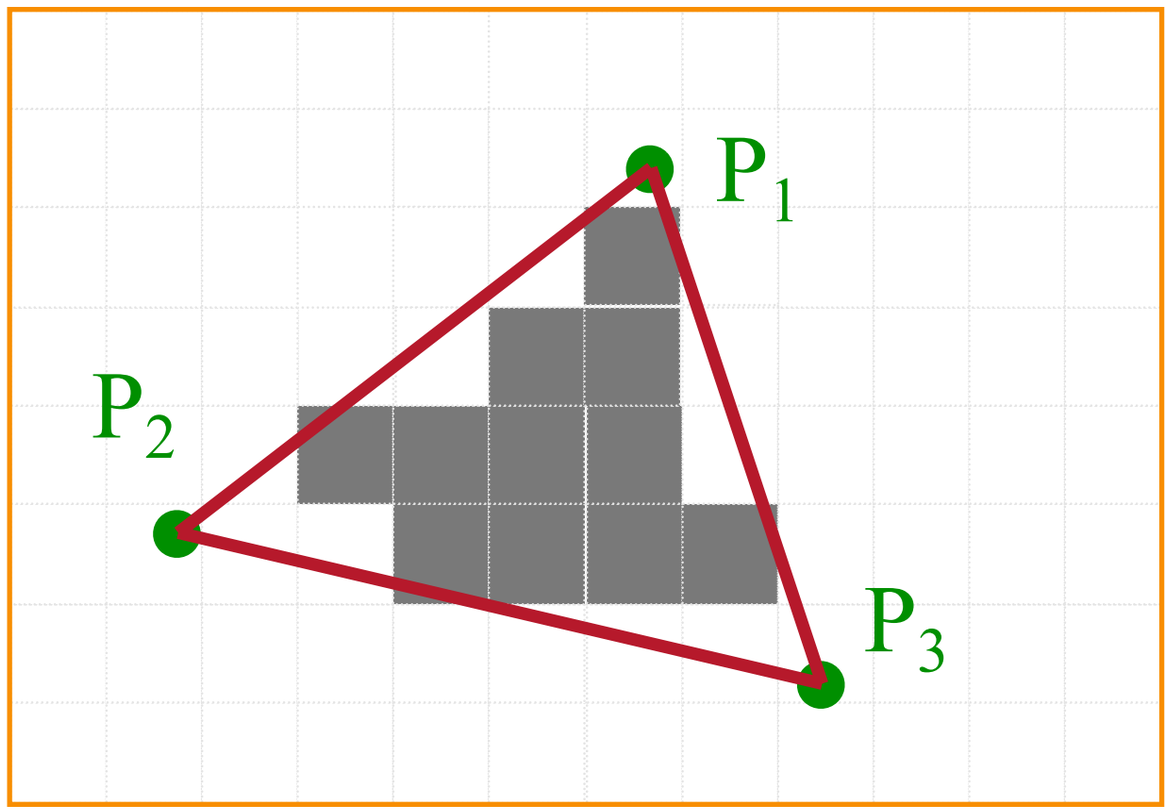
```
Boolean Inside(Triangle T, Point P)
{
   for each boundary line L of T {
      Scalar d = L.a*P.x + L.b*P.y + L.c;
      if (d < 0.0) return FALSE;
   }
   return TRUE;
}
```



Simple Algorithm

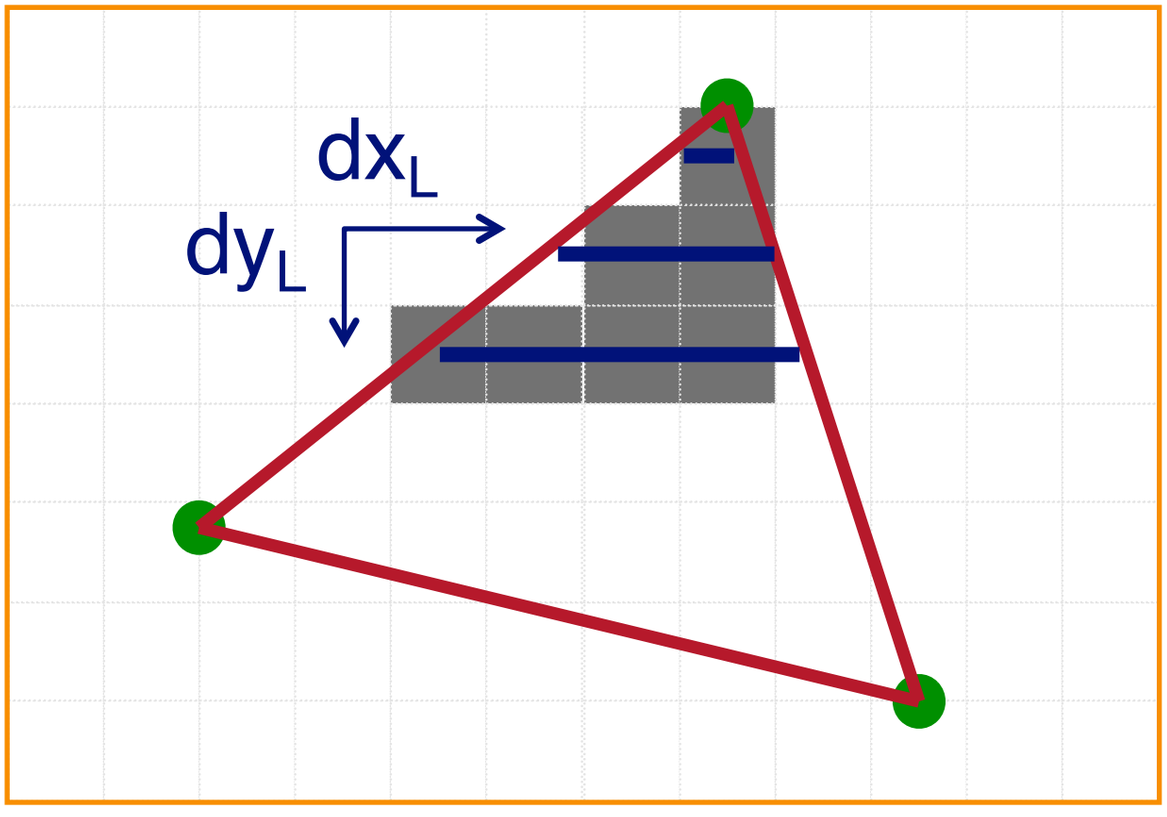
- What is bad about this algorithm?

```
void ScanTriangle(Triangle T, Color rgba){
   for each pixel P at (x,y){
      if (Inside(T, P))
         SetPixel(x, y, rgba);
   }
}
```



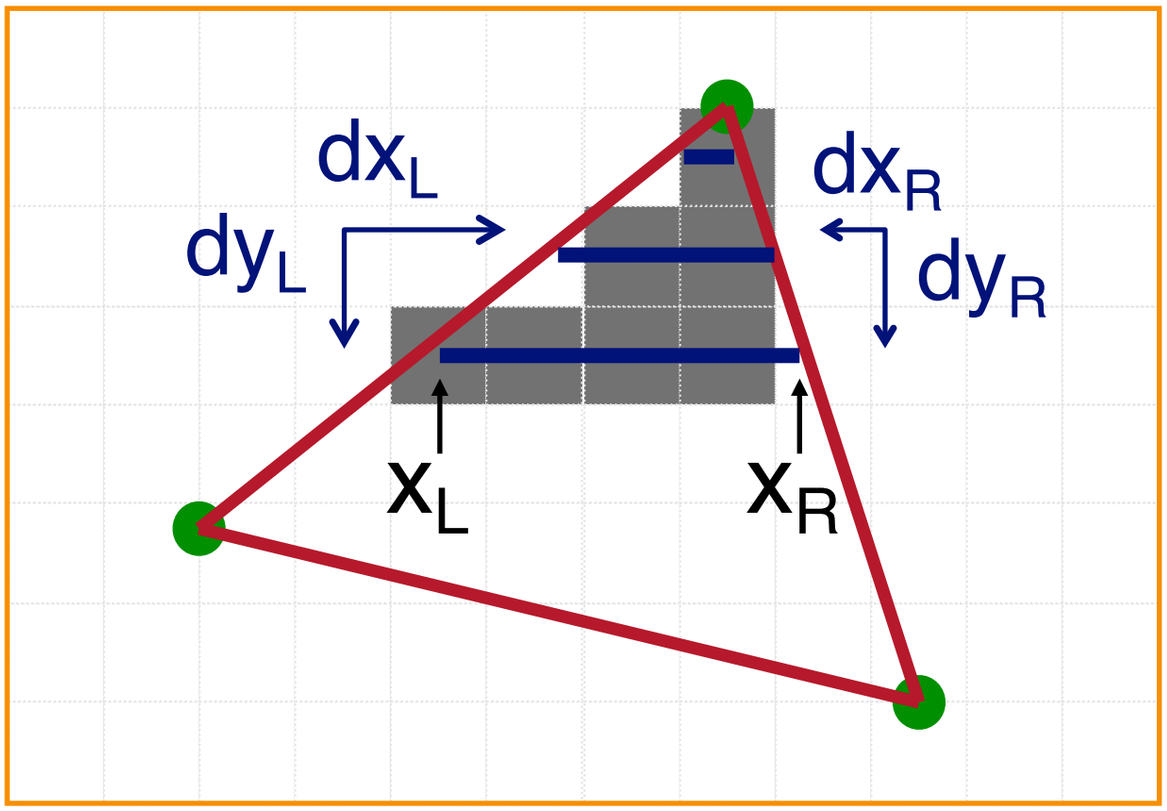
Triangle Sweep-Line Algorithm

- Take advantage of spatial coherence
  - Compute which pixels are inside using horizontal spans
  - Process horizontal spans in scan-line order
- Take advantage of edge linearity
  - Use edge slopes to update coordinates incrementally



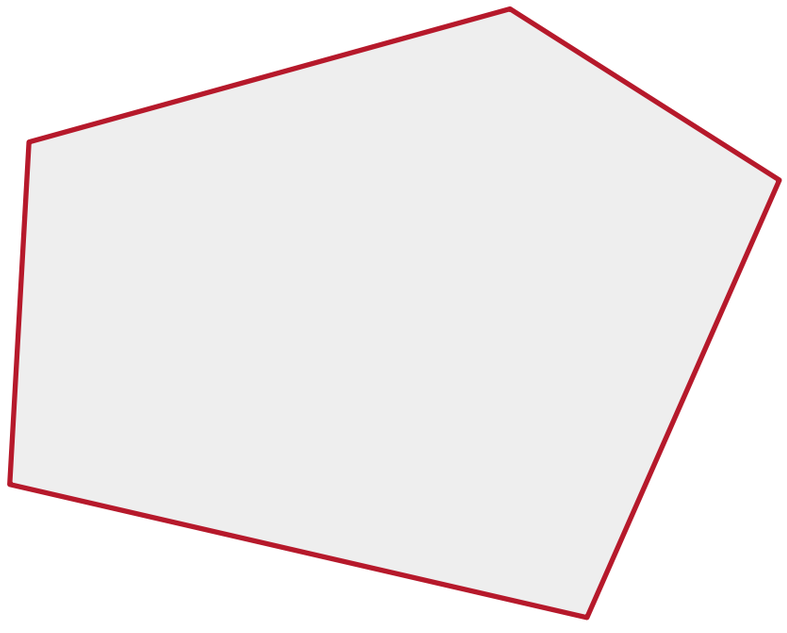
Triangle Sweep-Line Algorithm

```
void ScanTriangle(Triangle T, Color rgba){
   for both edge pairs {
      initialize xL, xR;
      compute dxL/dyL and dxR/dyR;
      for each scanline at y
       for (int x = xL; x <= xR; x++)
            SetPixel(x, y, rgba);
         xL += dxL/dyL;
         xR += dxR/dyR;
   }
}
```



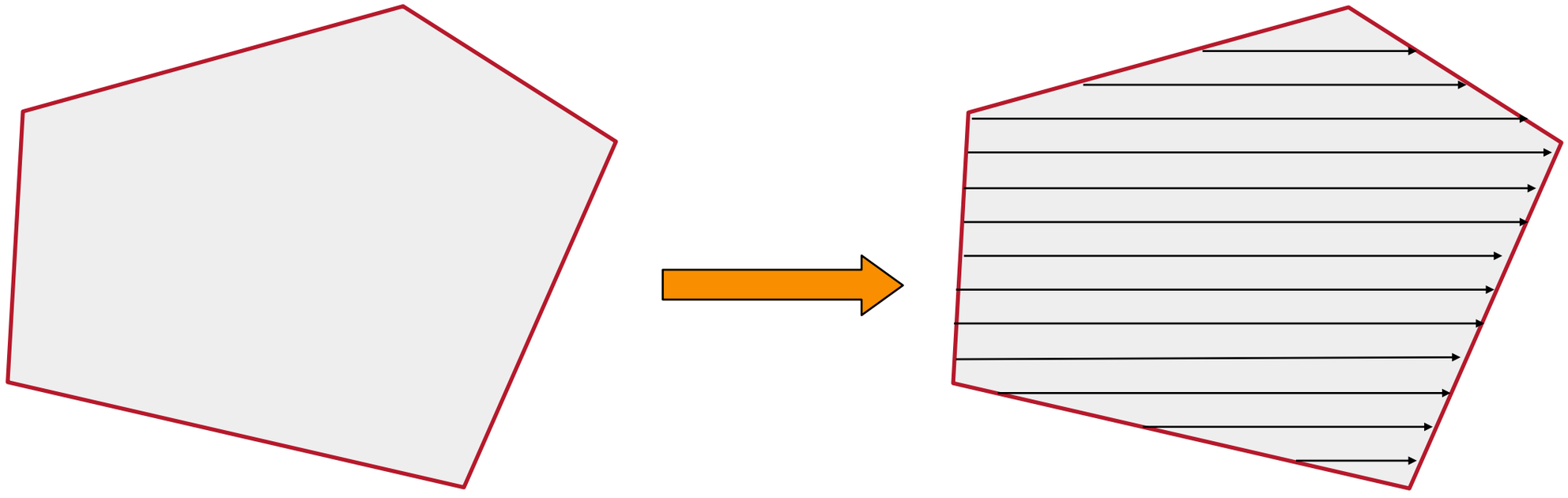
Polygon Scan Conversion

- Will this method work for convex polygons?



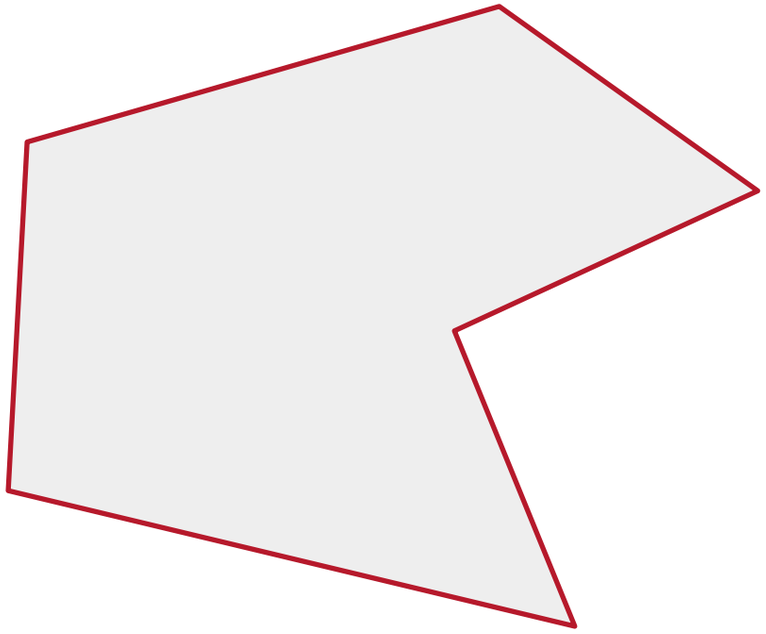
Polygon Scan Conversion

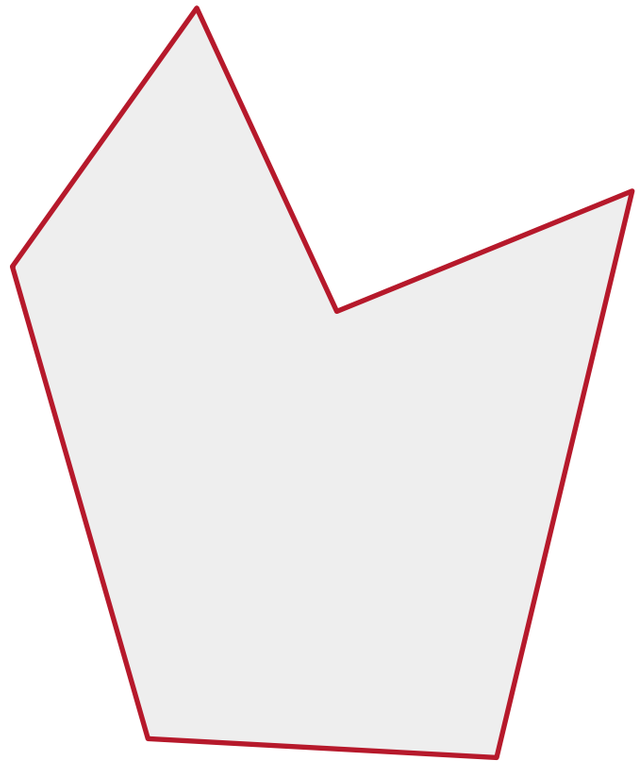
- Will this method work for convex polygons?
  - Yes, since each scan line will only intersect the polygon at two points.



Polygon Scan Conversion

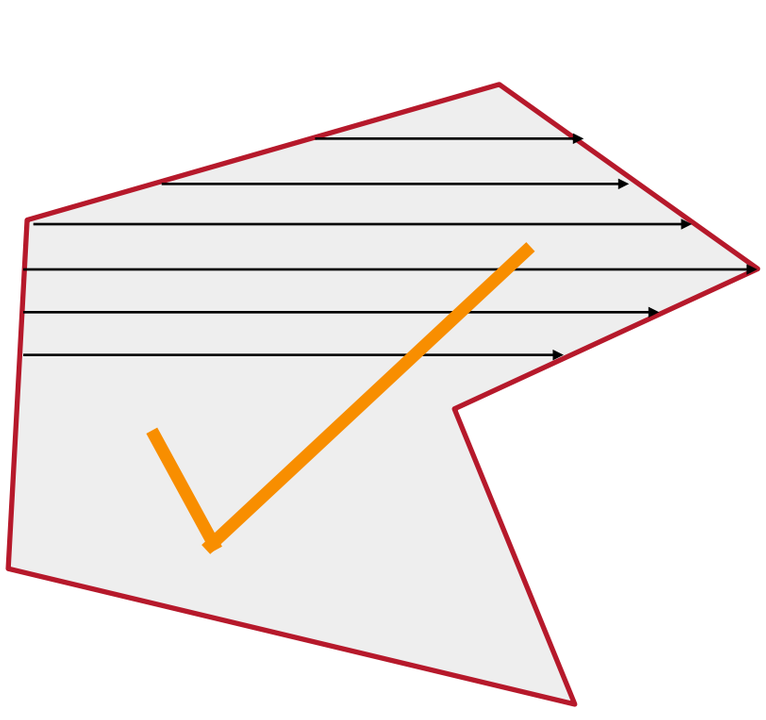
- How about these polygons?

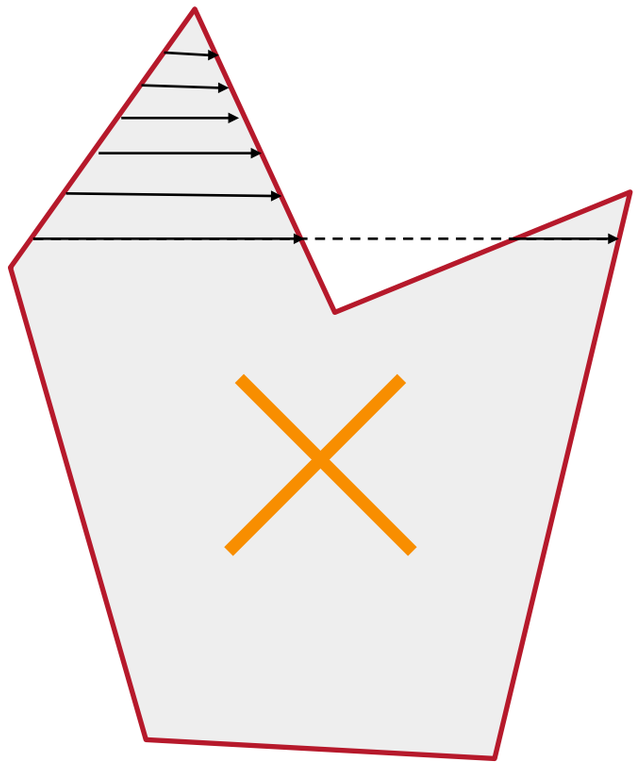




Polygon Scan Conversion

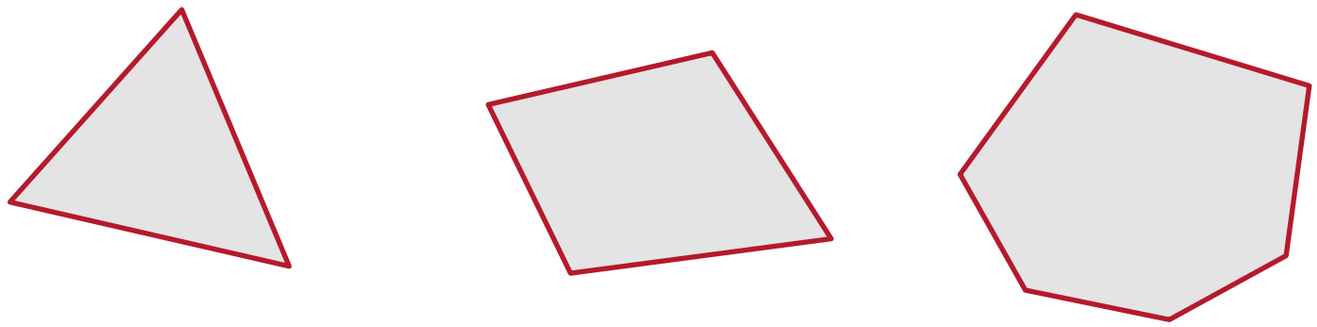
- How about these polygons?

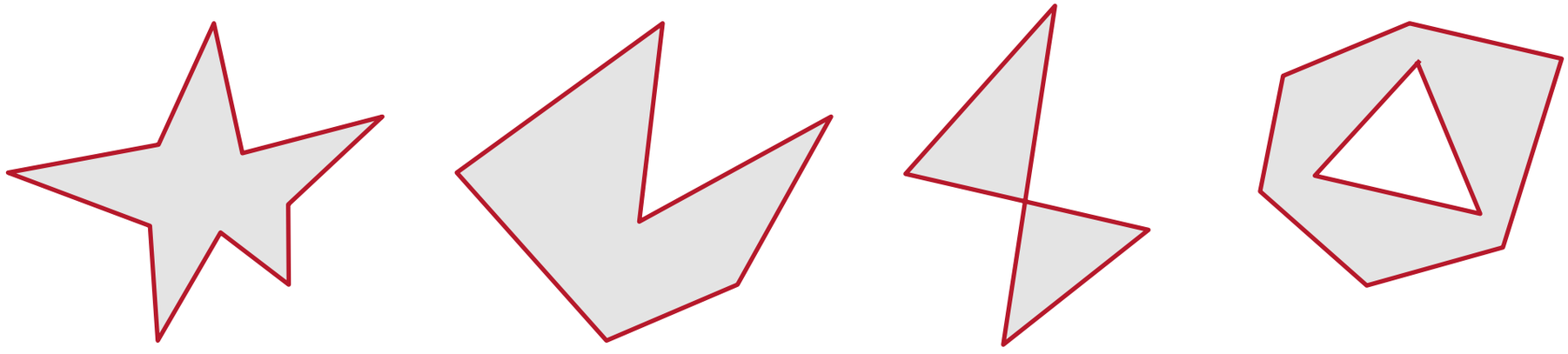




Polygon Scan Conversion

- Fill pixels inside a polygon
  - Triangle
  - Quadrilateral
  - Convex
  - Star-shaped
  - Concave
  - Self-intersecting
  - Holes

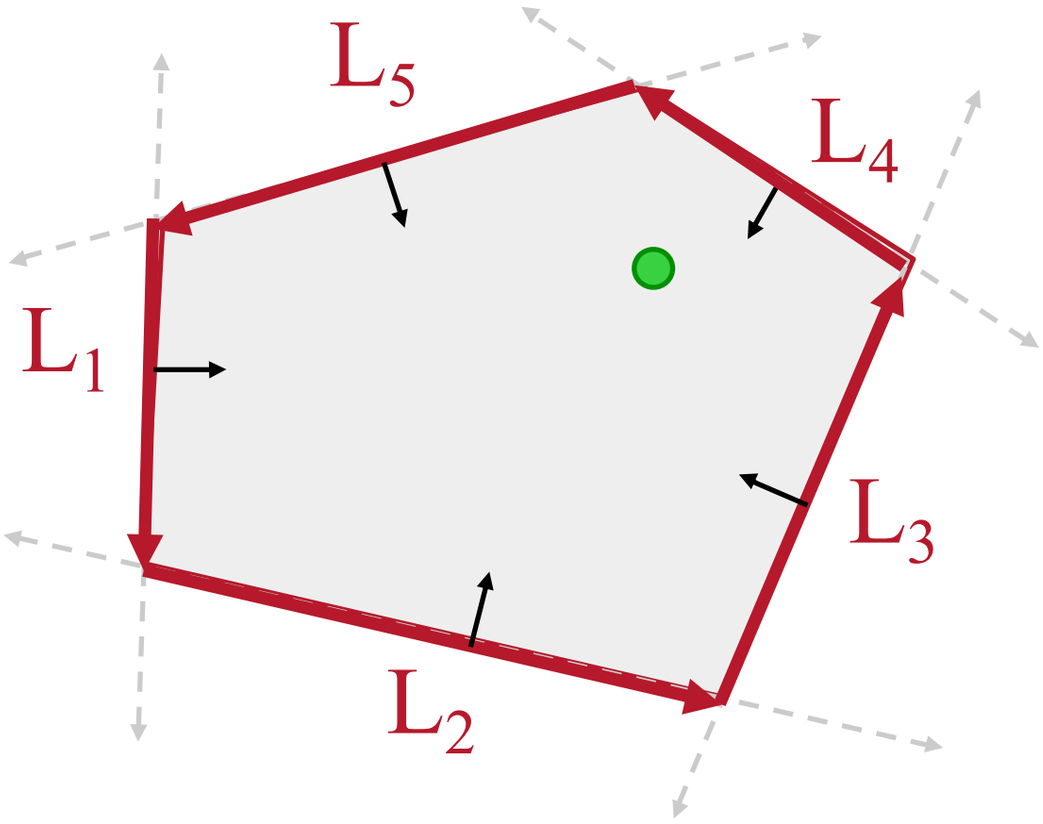




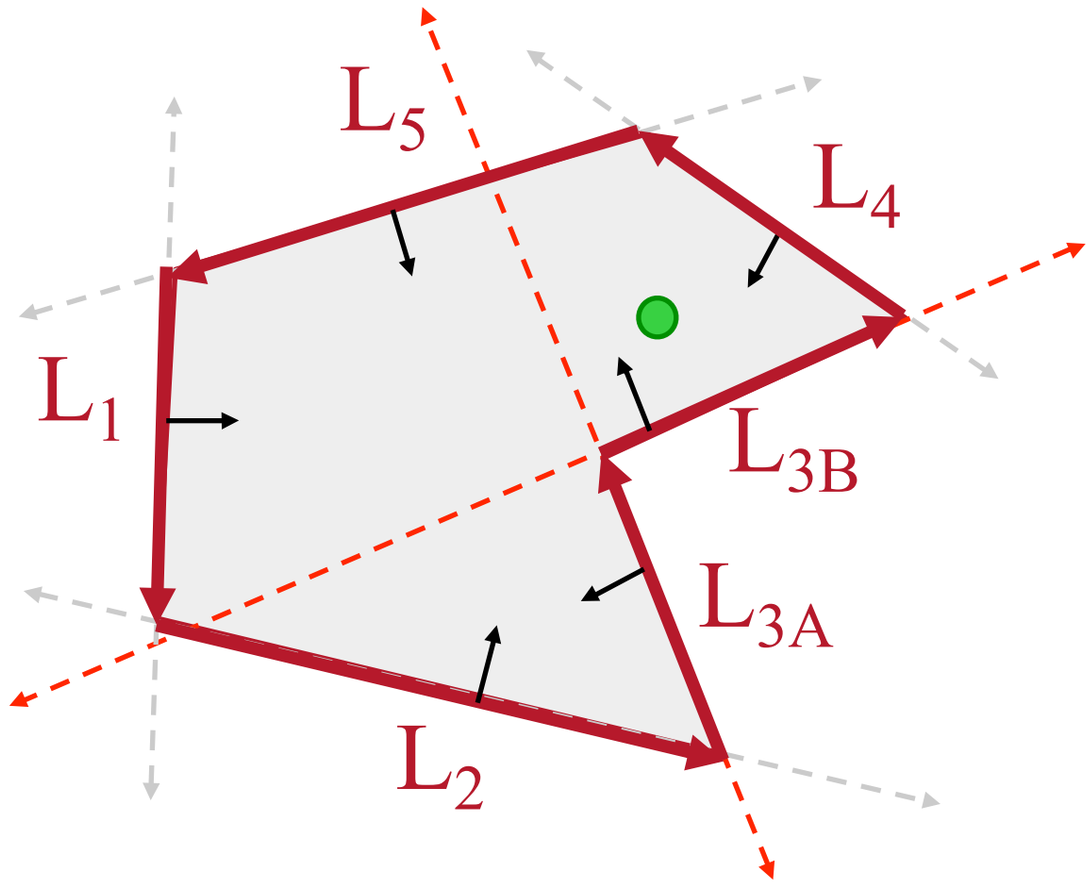
What problems do we encounter with arbitrary polygons?

Polygon Scan Conversion

- Need better test for points inside polygon
  - Triangle method works only for convex polygons



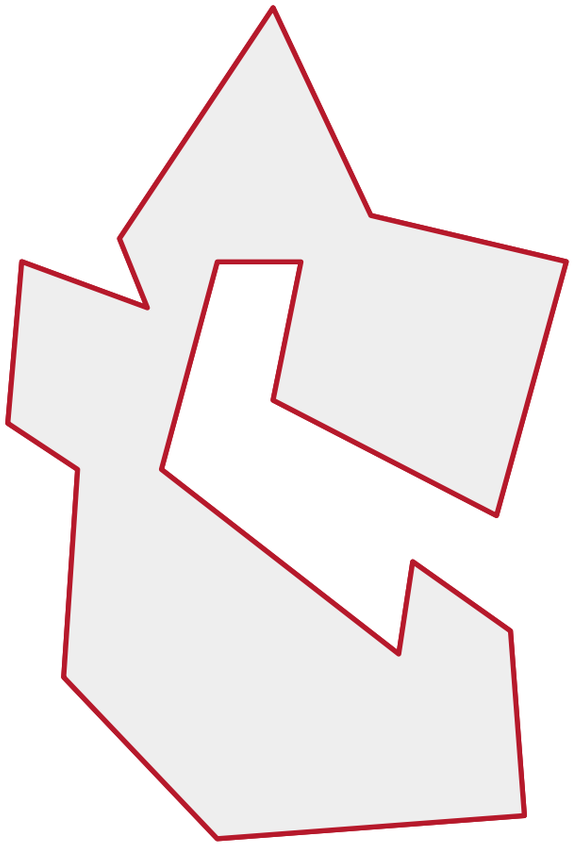
Convex Polygon



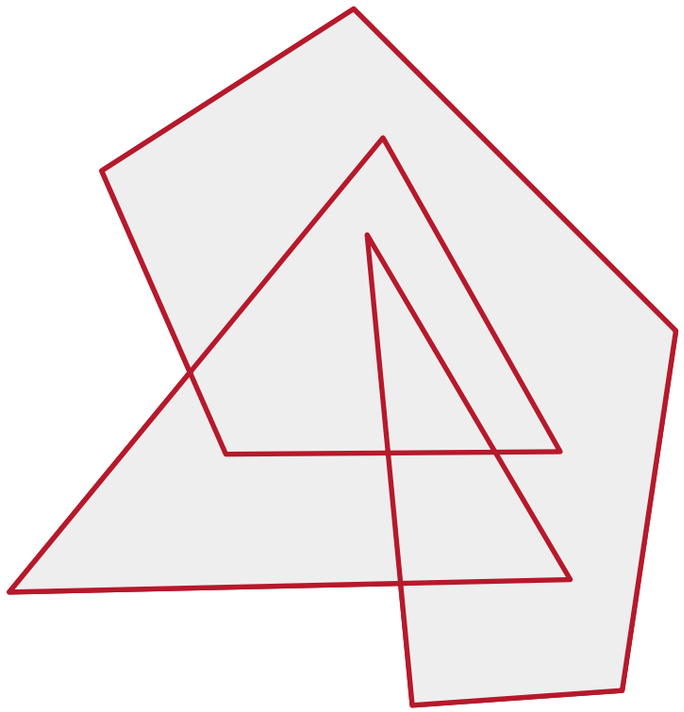
Concave Polygon

Inside Polygon Rule

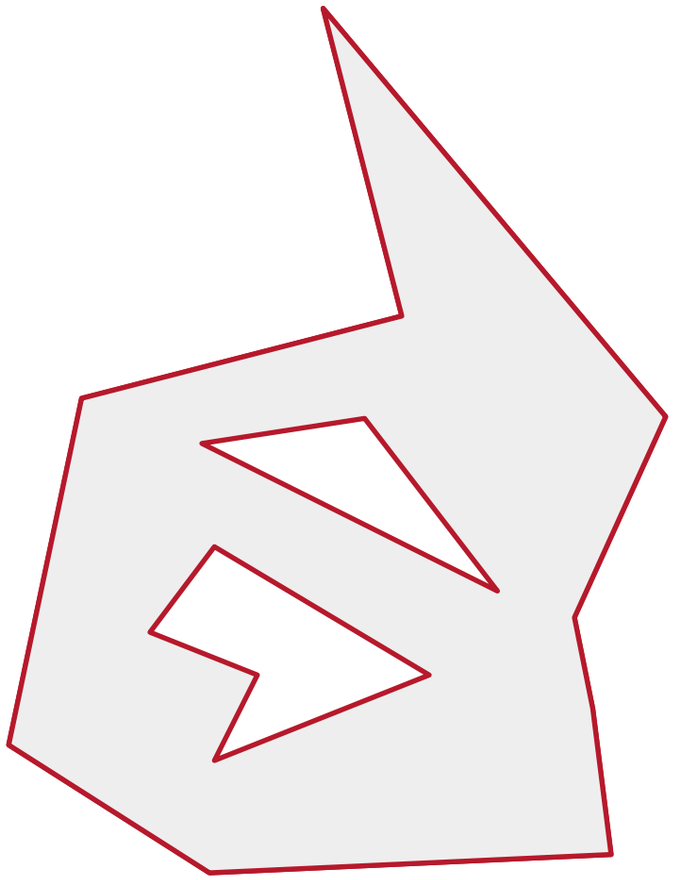
- What is a good rule for which pixels are inside?



Concave



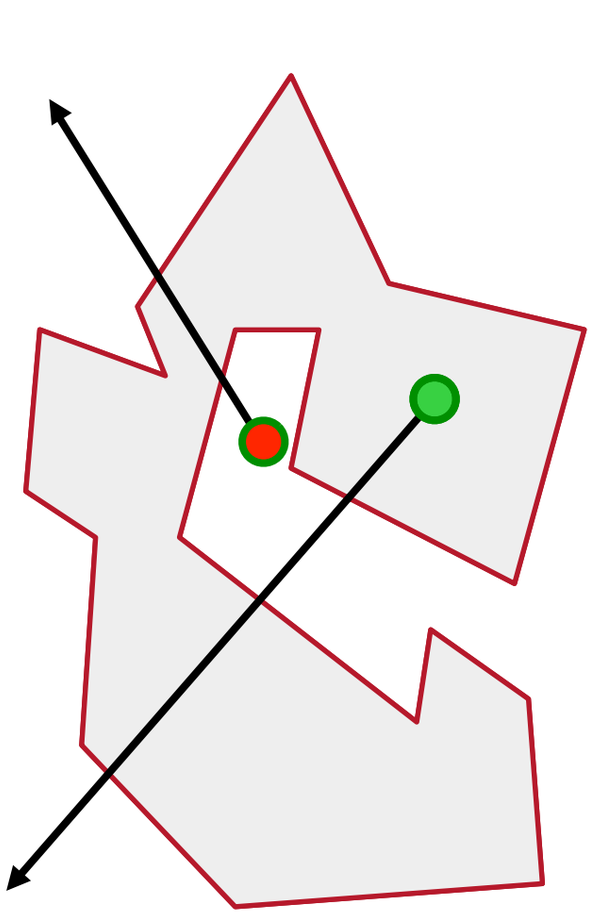
Self-Intersecting



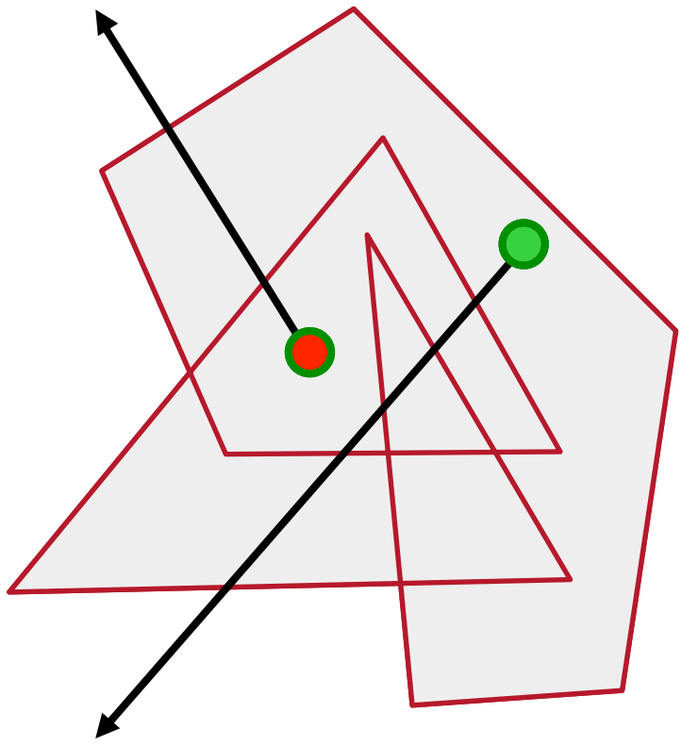
With Holes

Inside Polygon Rule

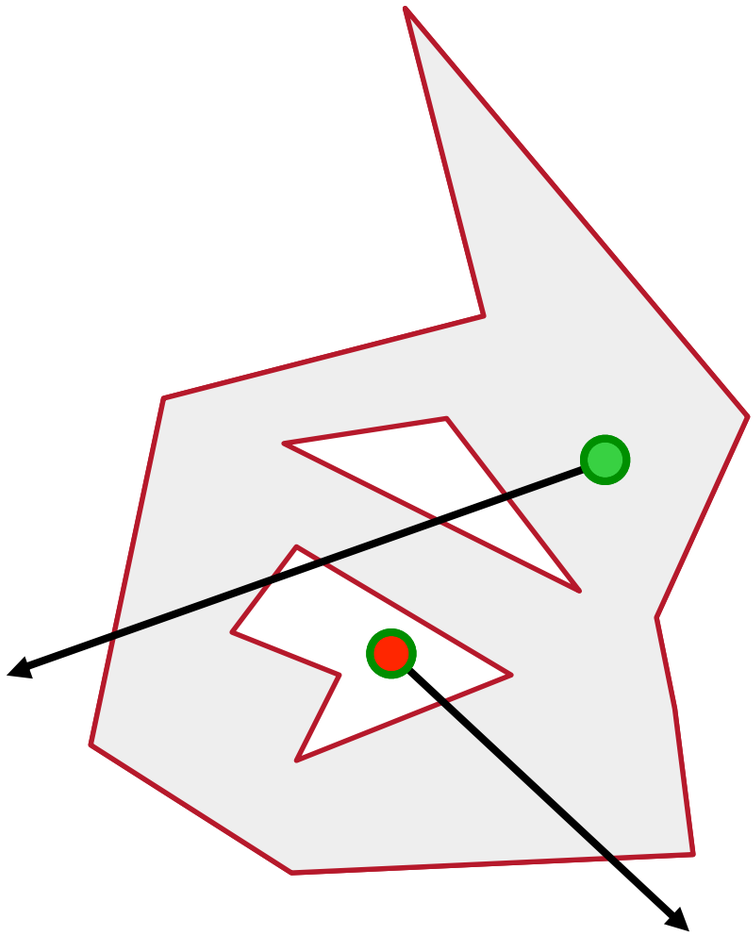
- Odd-parity rule
  - Any ray from P to infinity crosses odd number of edges



Concave



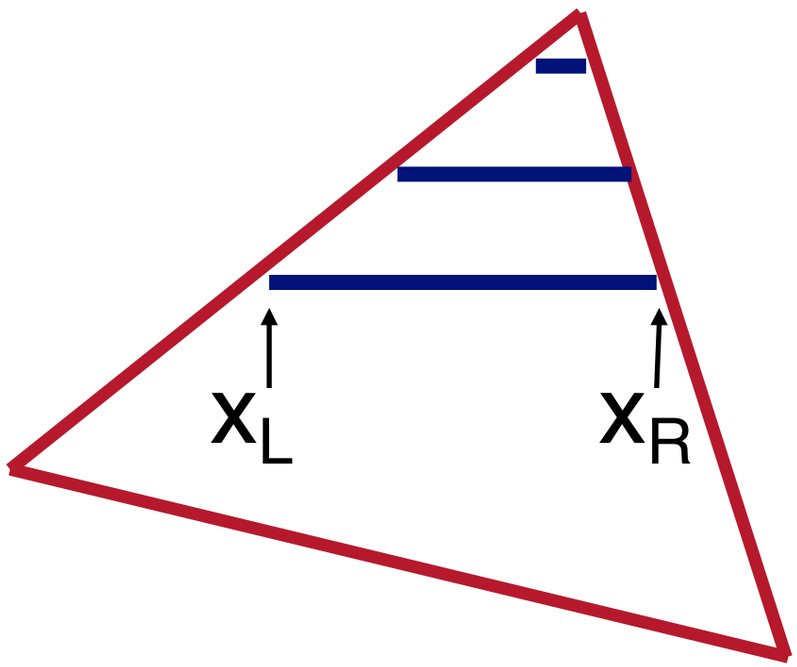
Self-Intersecting



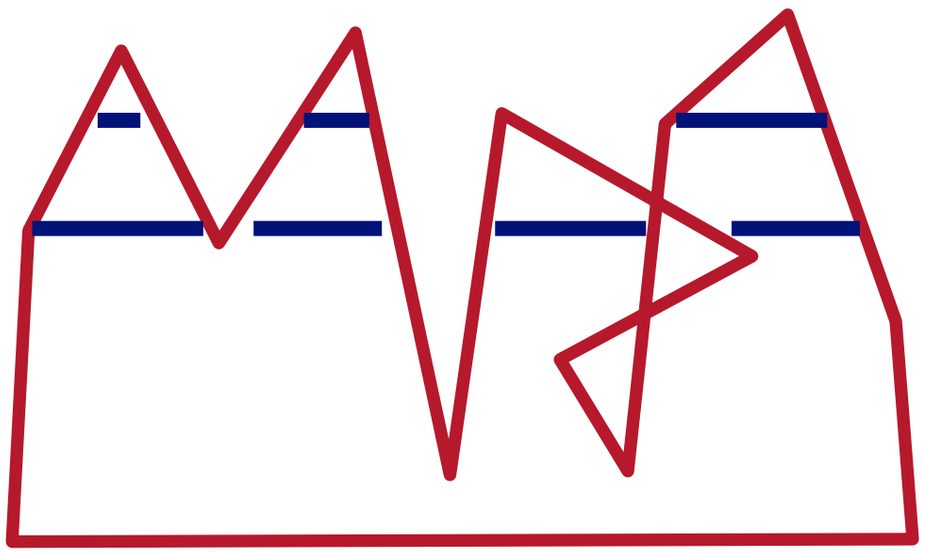
With Holes

Polygon Sweep-Line Algorithm

- Incremental algorithm to find spans, and determine “insideness” with odd parity rule
  - Takes advantage of scan line coherence



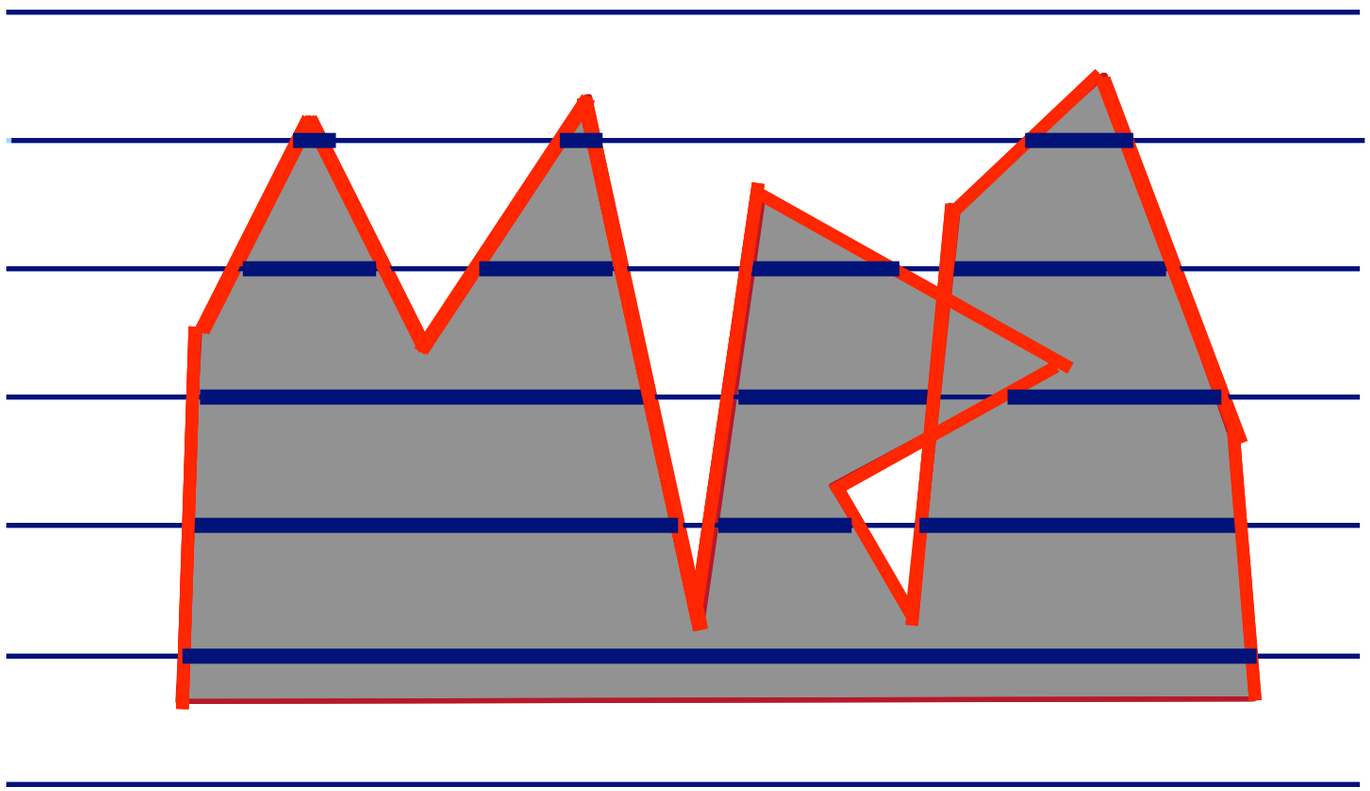
Triangle



Polygon

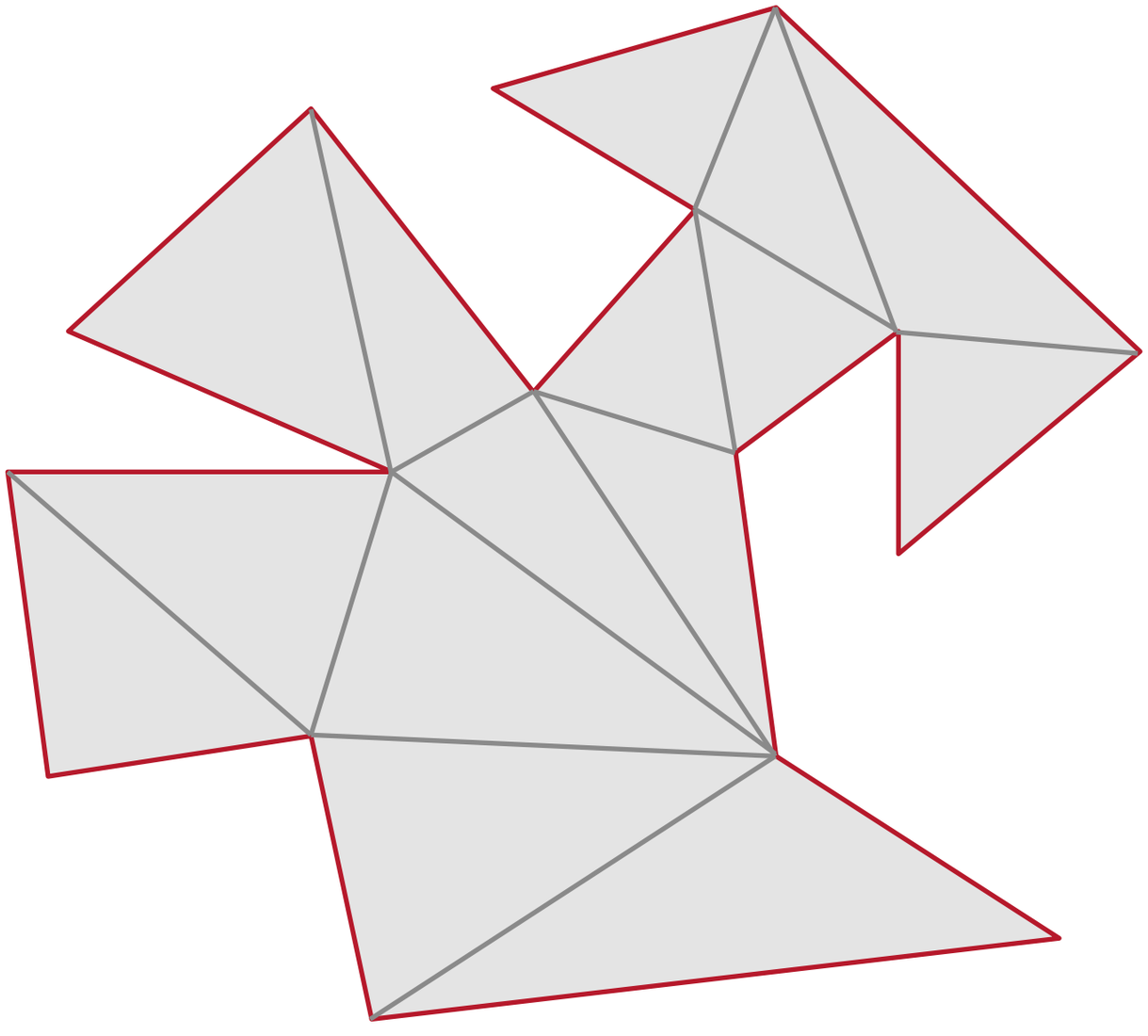
Polygon Sweep-Line Algorithm

```
void ScanPolygon(Polygon P, Color rgba){
   sort edges by maxy
   make empty "active edge list"
   for each scanline (top-to-bottom) {
      insert/remove edges from "active edge list"
      update x coordinate of every active edge
      sort active edges by x coordinate
      for each pair of active edges (left-to-right)
         SetPixels(x_i, x_i+1, y, rgba);
      }
}
```



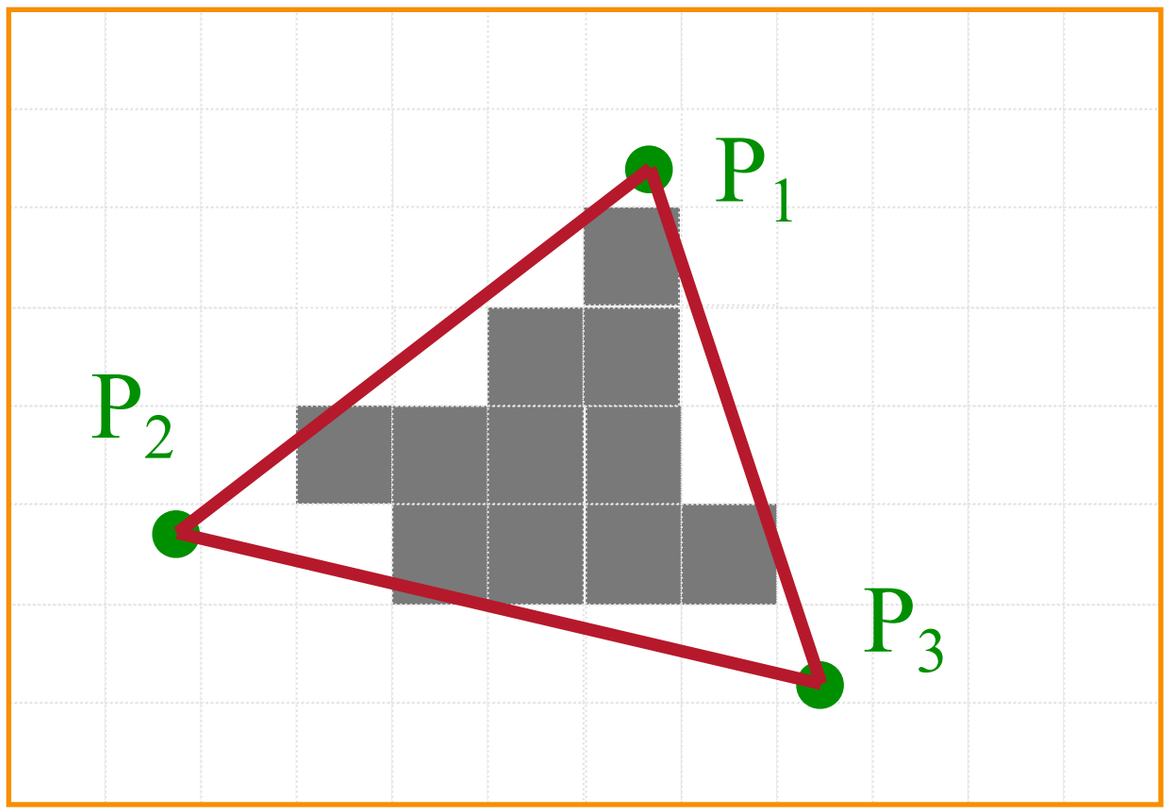
Hardware Scan Conversion

- Convert everything into triangles
  - Scan convert the triangles



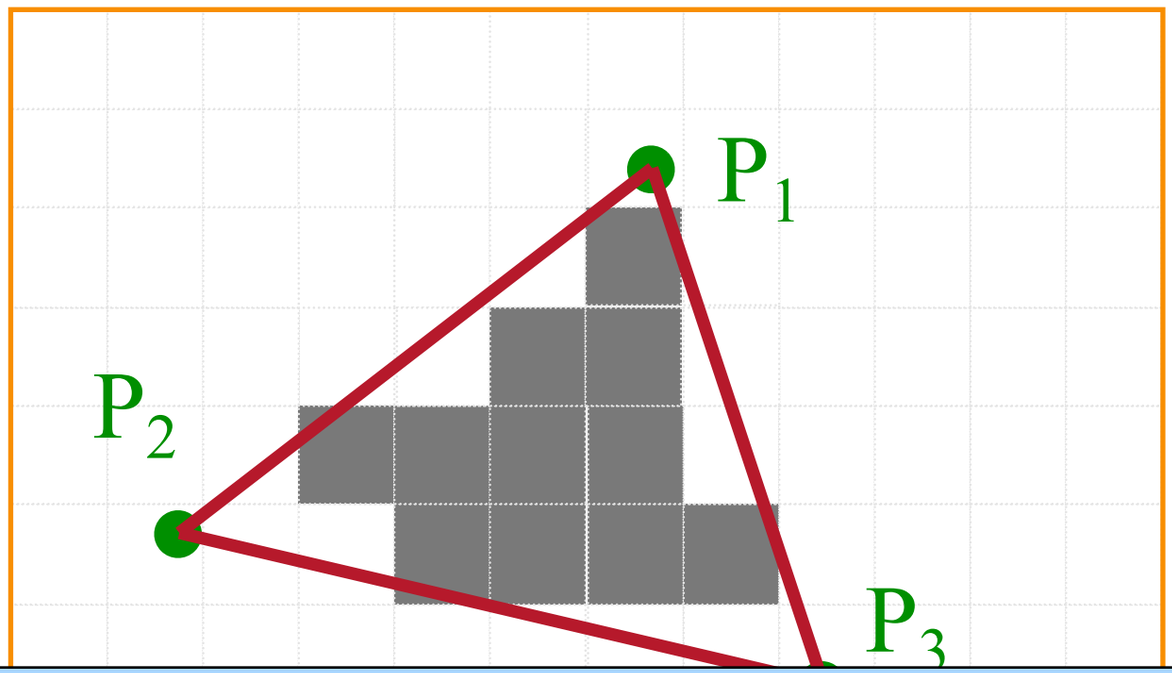
Scan Conversion

- What about pixels on edges?
  - If we set them either “on” or “off” we get aliasing or “jaggies”



Scan Conversion

- What about pixels on edges?
  - If we set them either “on” or “off” we get aliasing or “jaggies”



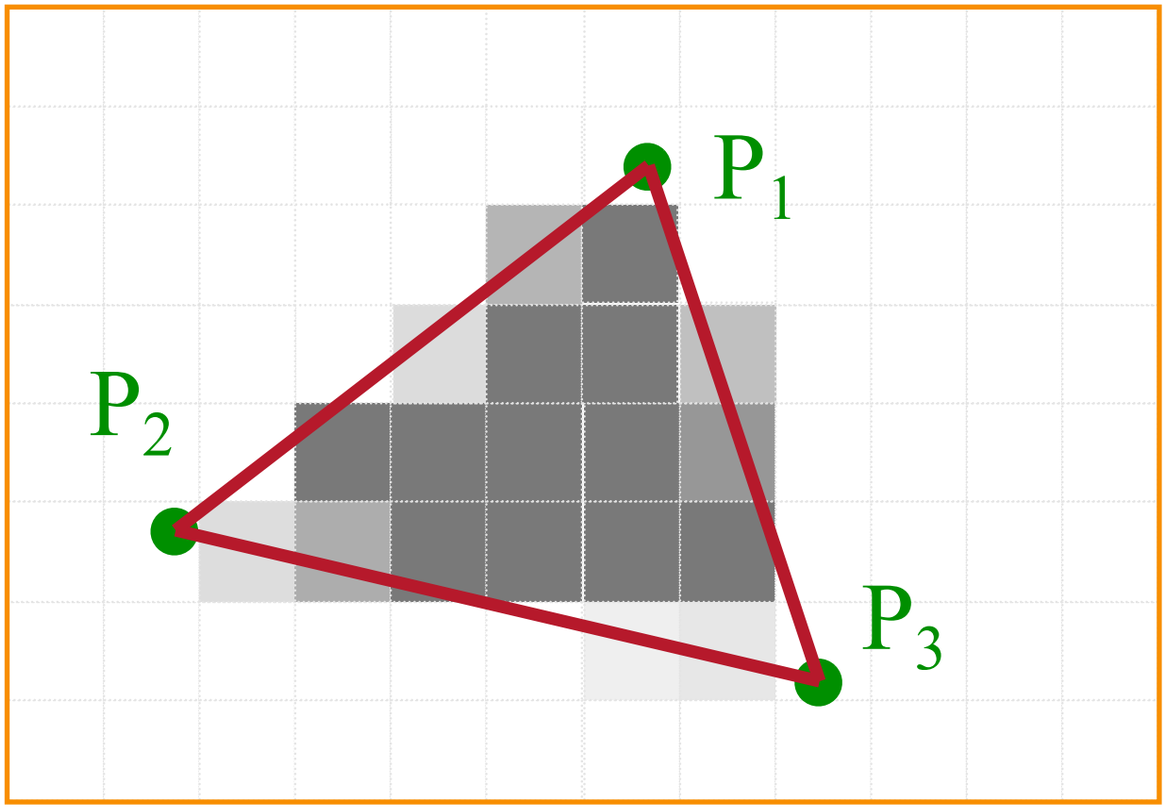
This amounts to using a “nearest” interpolation filter!

Antialiasing Techniques

- Display at higher resolution
  - Corresponds to increasing sampling rate
  - Not always possible (fixed size monitors, fixed refresh rates, etc.)
- Modify pixel intensities
  - Vary pixel intensities along primitive boundaries for antialiasing
  - Must have more than bi-level display

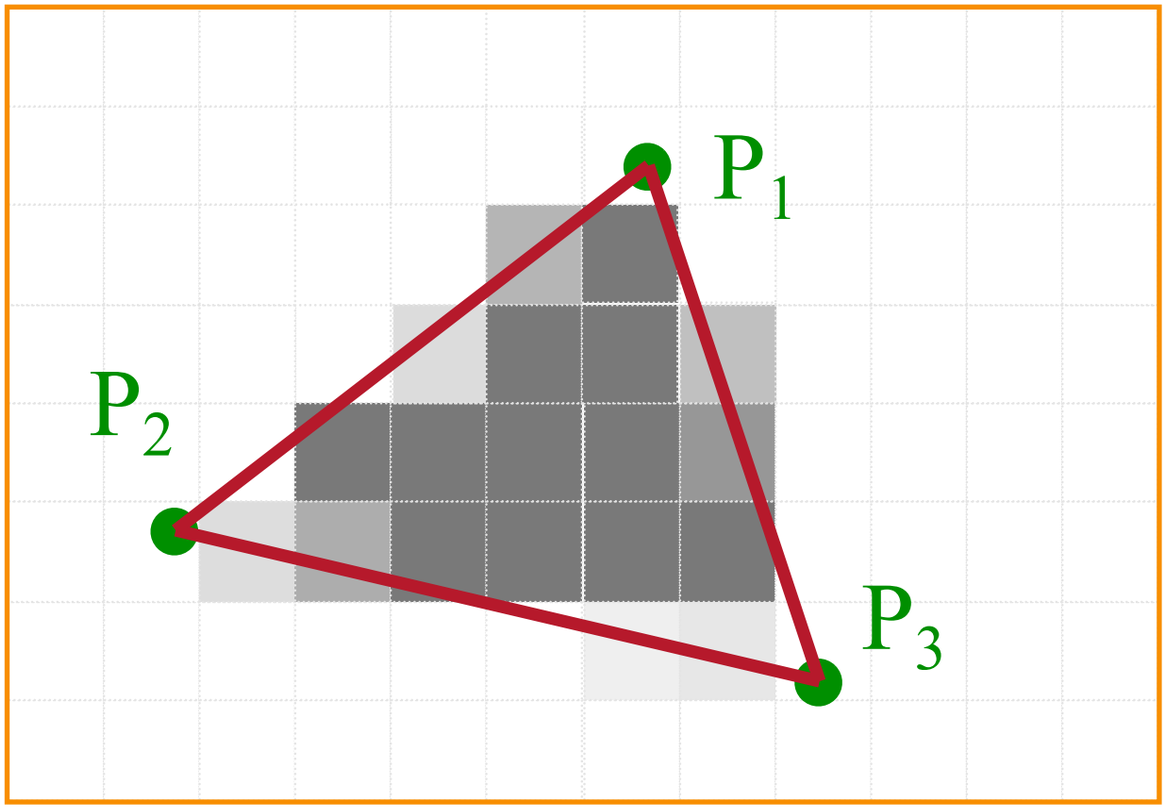
Scan Conversion

- What about pixels on edges?
  - If we set them either “on” or “off” we get aliasing or “jaggies”
  - Vary pixel intensities along primitive boundaries for antialiasing



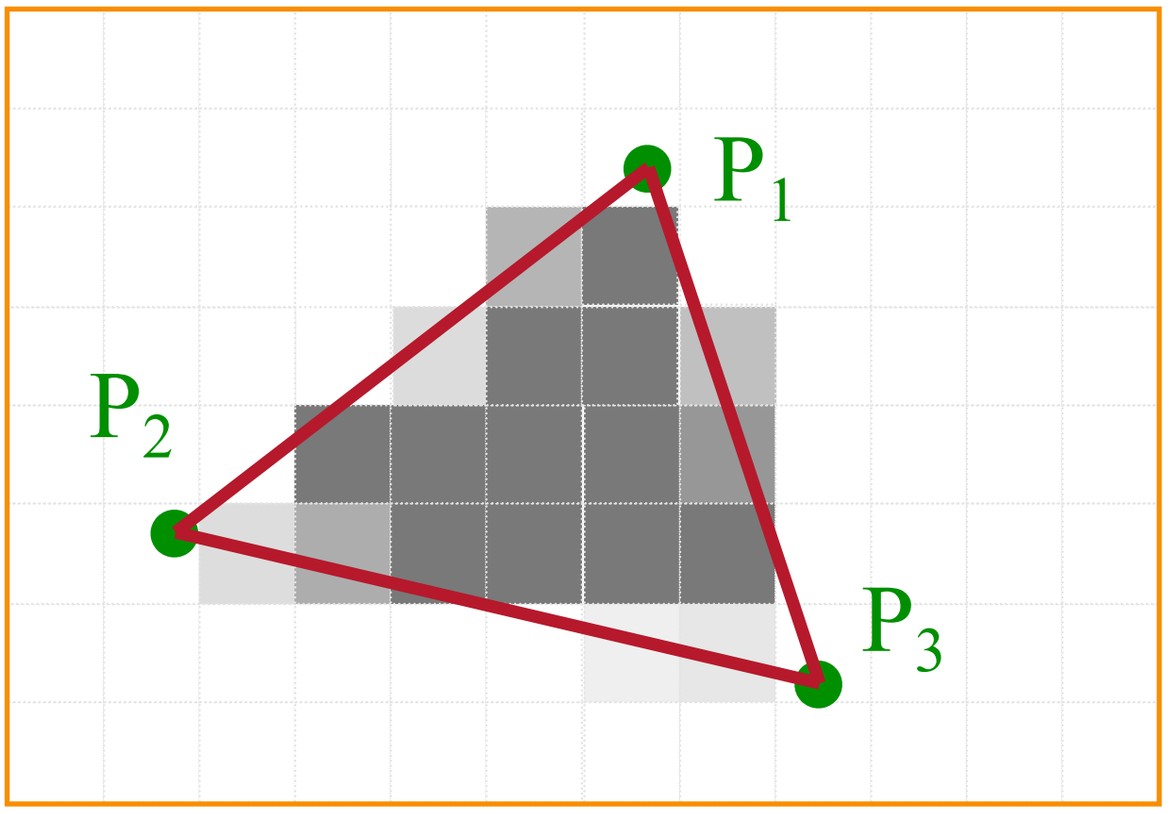
Antialiasing

- Method 1: Area sampling (aka prefiltering)
  - Calculate percent of pixel covered by primitive
  - Multiply this percentage by desired intensity/color
  - Set resulting pixel to closest available display level



Antialiasing

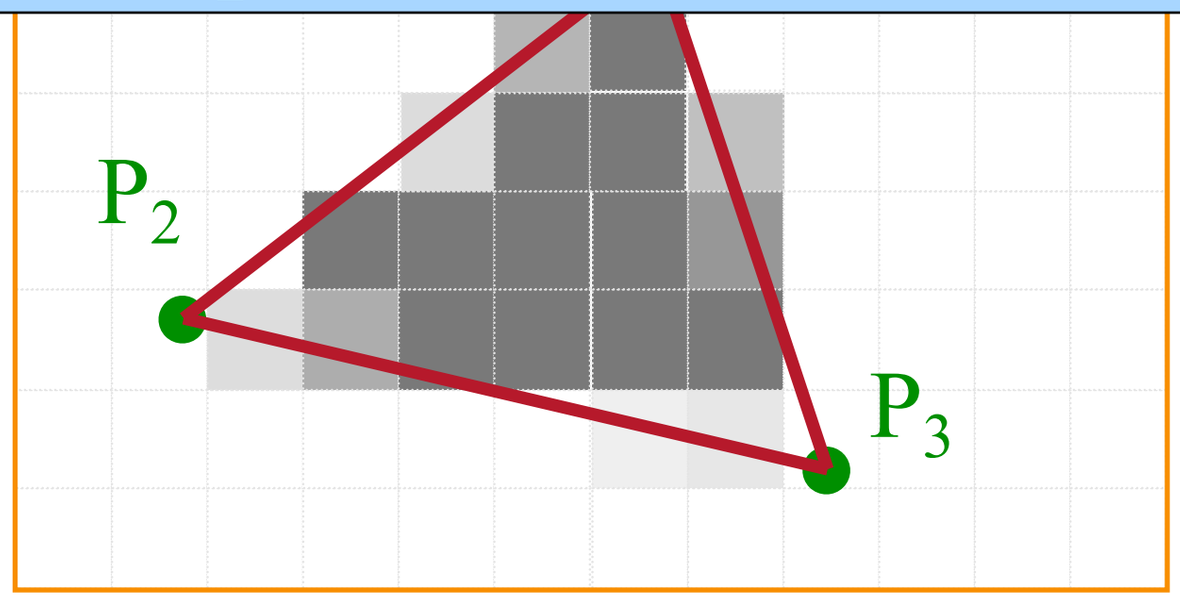
- Method 2: Supersampling (aka postfiltering)
  - Sample as if screen were higher resolution
  - Average multiple samples to get final intensity



Antialiasing

- Method 2: Supersampling (aka postfiltering)
  - Sample as if screen were higher resolution
  - Average multiple samples to get final intensity

This amounts to using a “bilinear” interpolation filter!

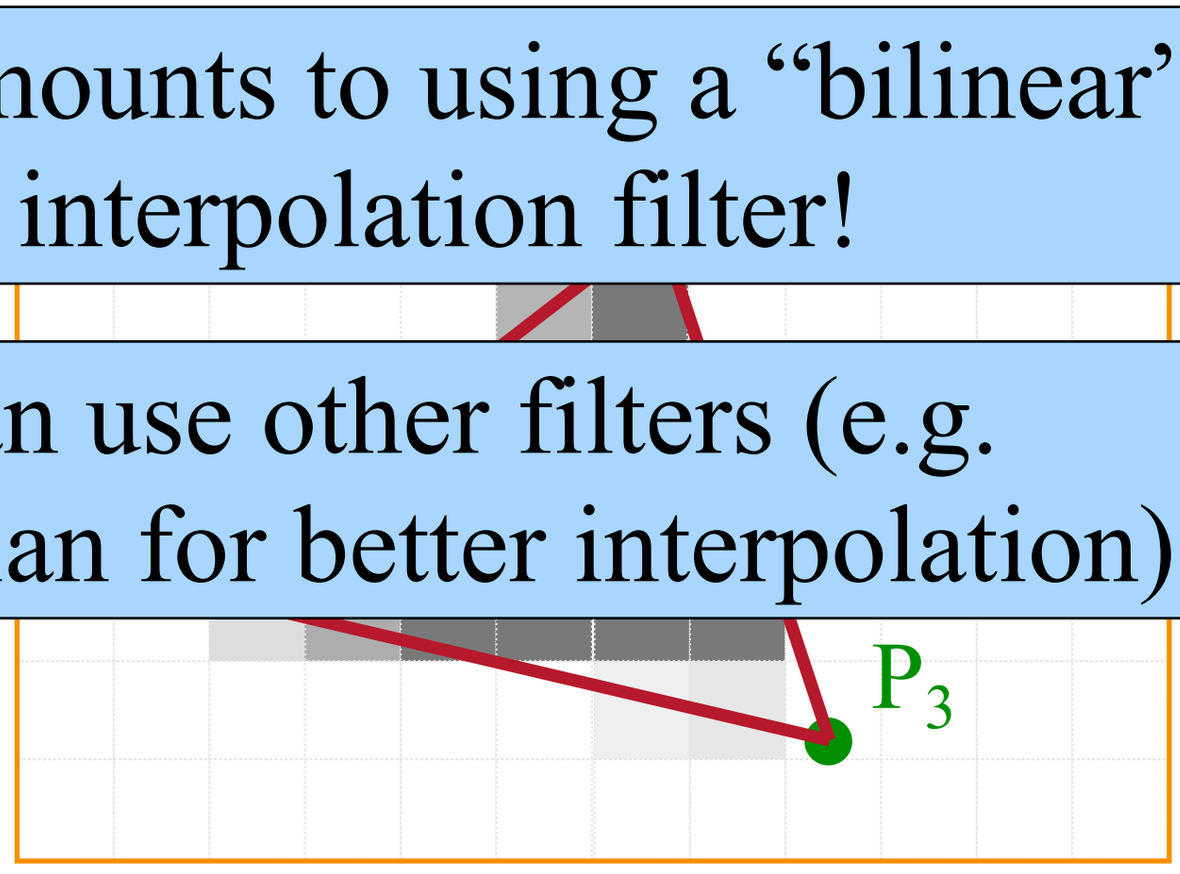


Antialiasing

- Method 2: Supersampling (aka postfiltering)
  - Sample as if screen were higher resolution
  - Average multiple samples to get final intensity

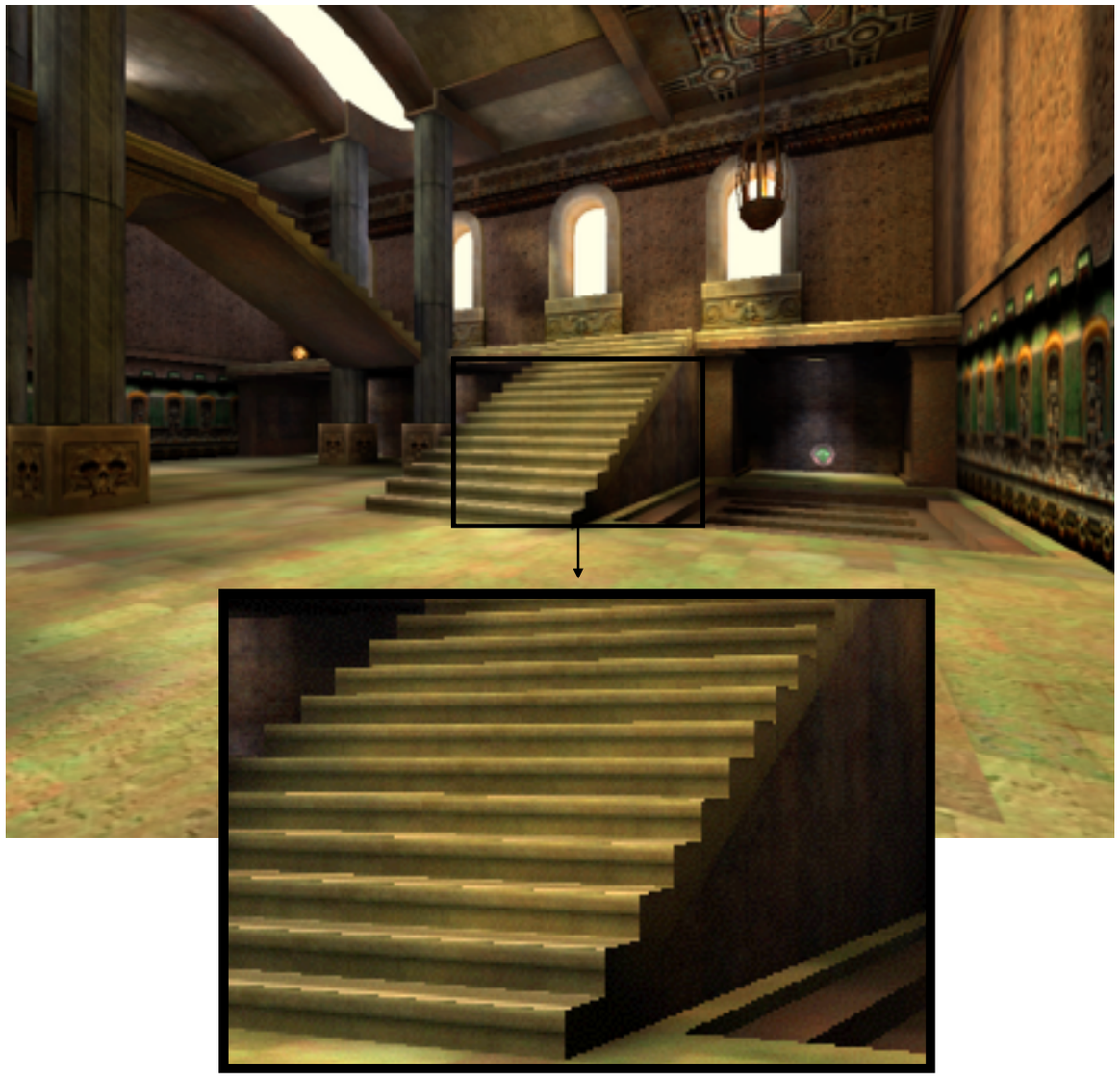
This amounts to using a “bilinear” interpolation filter!

Can use other filters (e.g. Gaussian for better interpolation)



Scan Conversion

- Example:



No Anti-Aliasing

4 x Anti-Aliasing

Images courtesy of NVIDIA