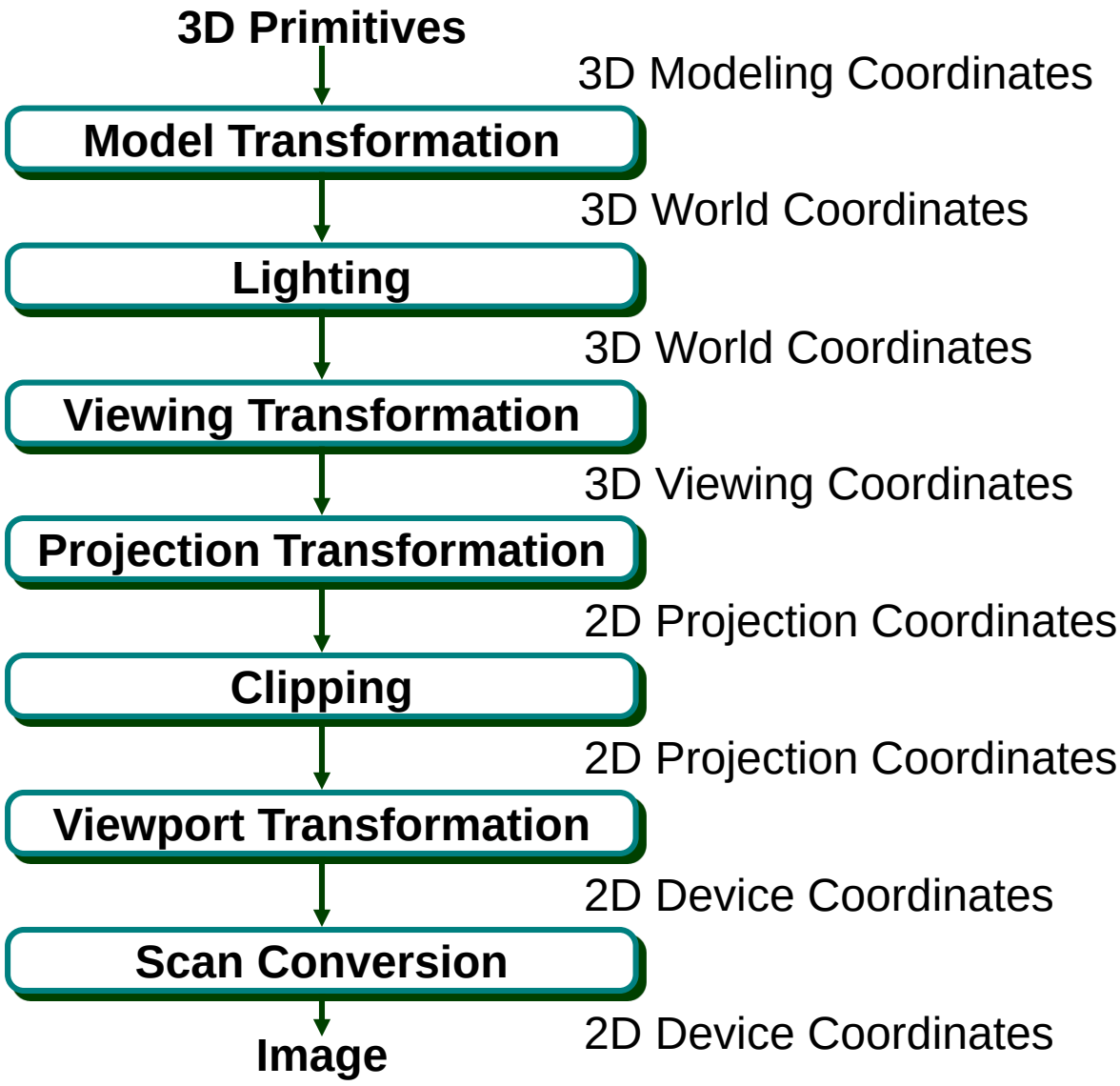
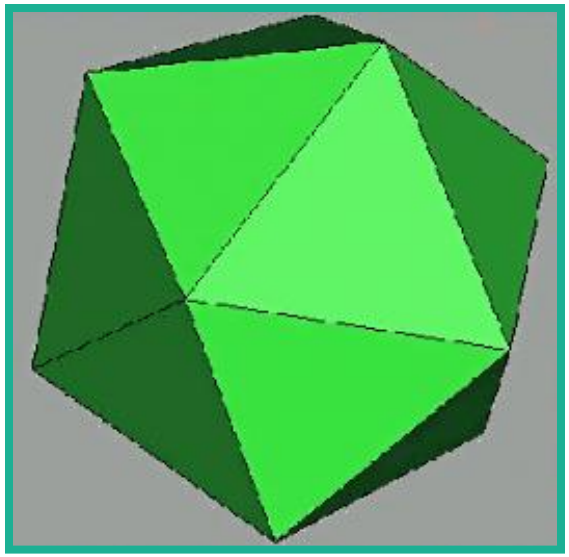
2D Viewing

Contents

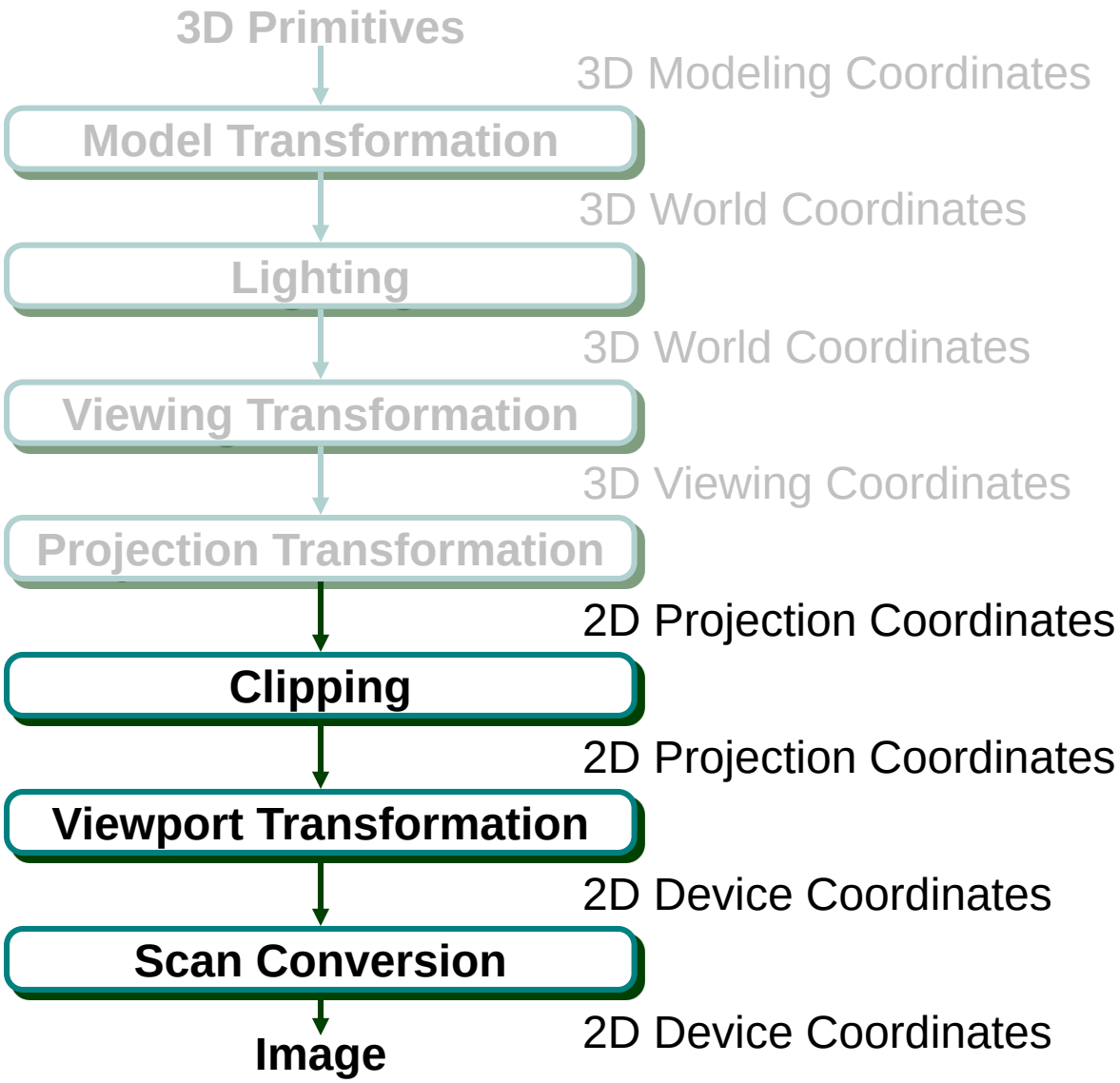
- **3D Rendering Pipeline**
- **2D Rendering Pipeline**
- **Clipping**
  - Cohen-Sutherland Line Clipping
  - Sutherland-Hodgeman Polygon Clipping
- **Viewport Transformation**
- **Scan Conversion**
- **Summary of Transformation**

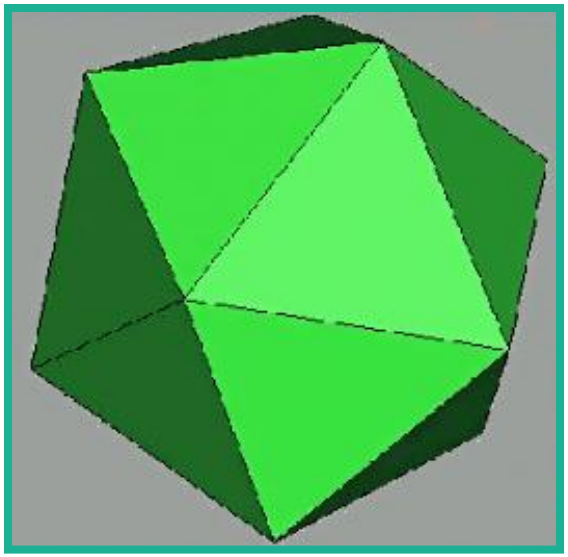
3D Rendering Pipeline



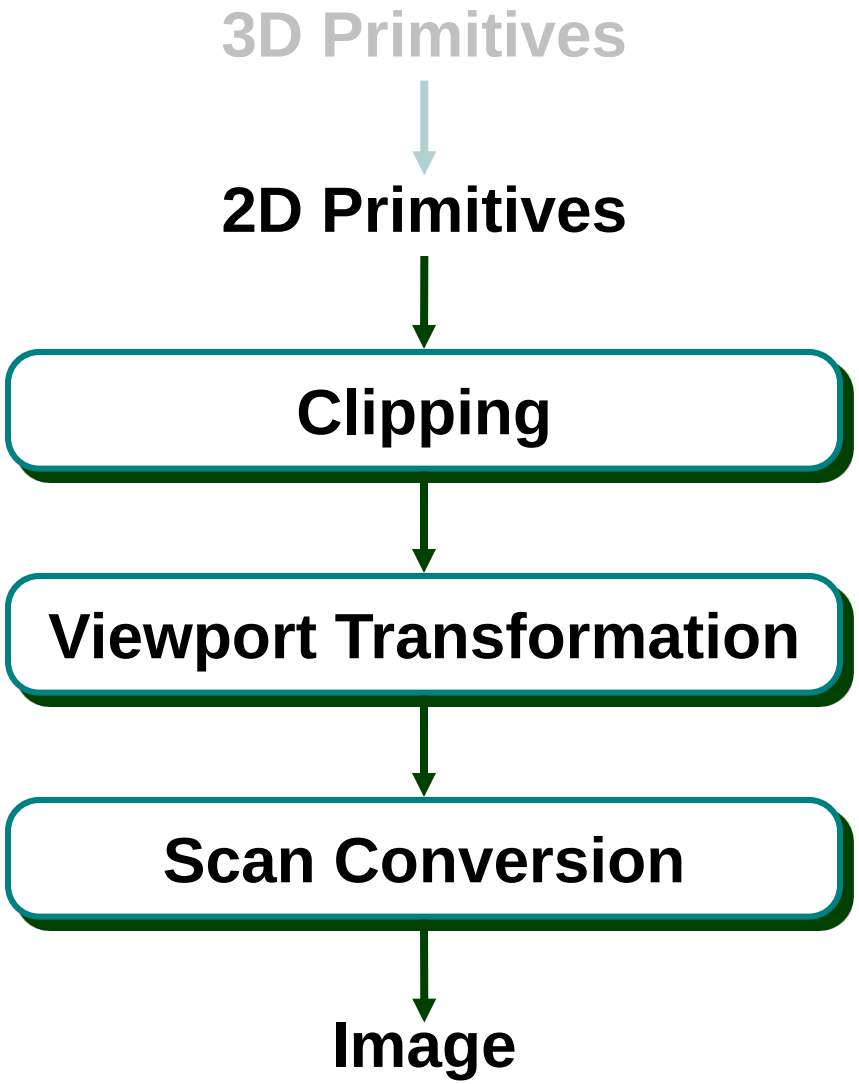


3D Rendering Pipeline





2D Rendering Pipeline

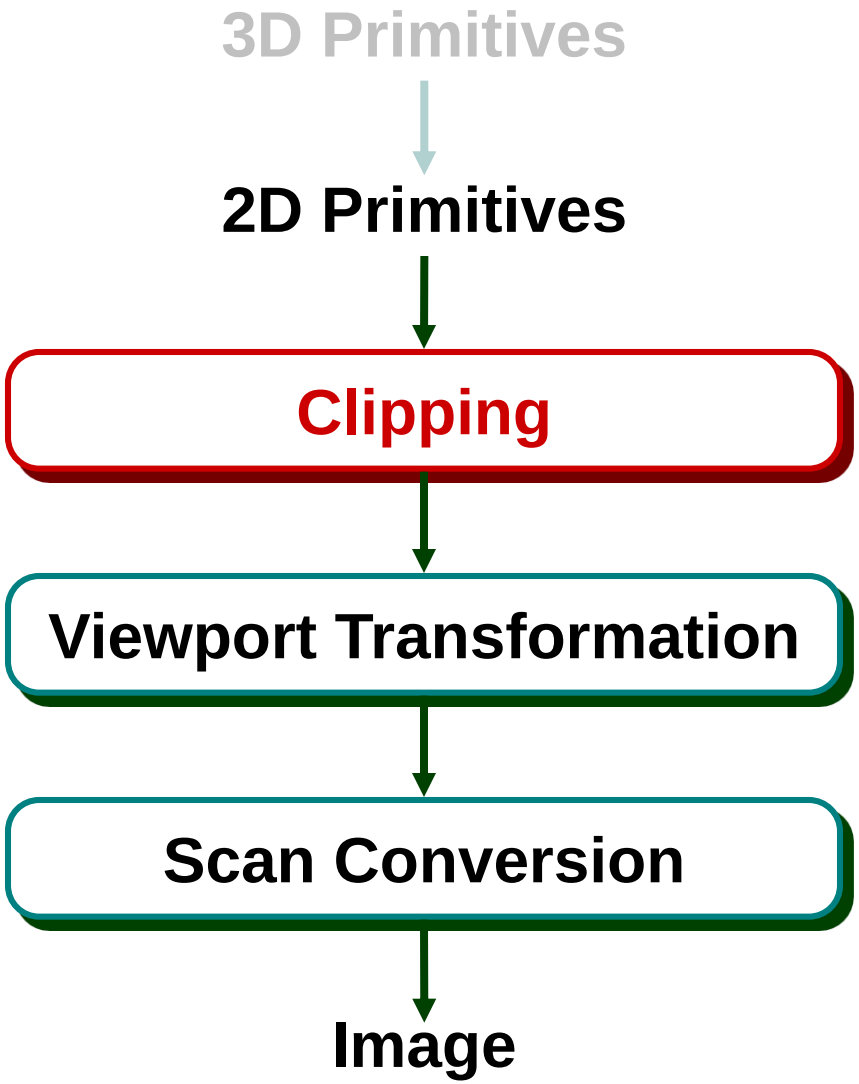


Clip portions of geometric primitives residing outside window

Transform the clipped primitives from screen to image coordinates

Fill pixel representing primitives in screen coordinates

2D Rendering Pipeline



Clip portions of geometric primitives residing outside window

Transform the clipped primitives from screen to image coordinates

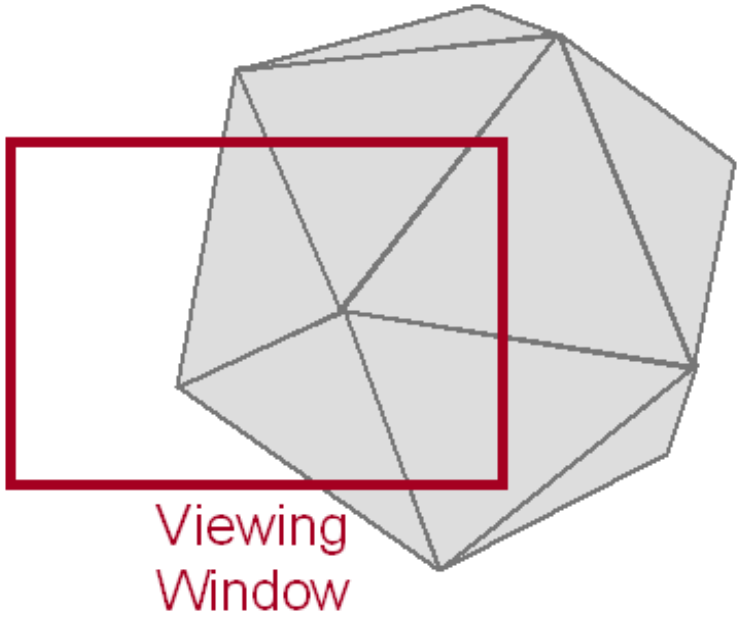
Fill pixel representing primitives in screen coordinates

Clipping

- **Avoid Drawing Parts of Primitives Outside Window**
  - Window defines part of scene being viewed
  - Must draw geometric primitives only inside window

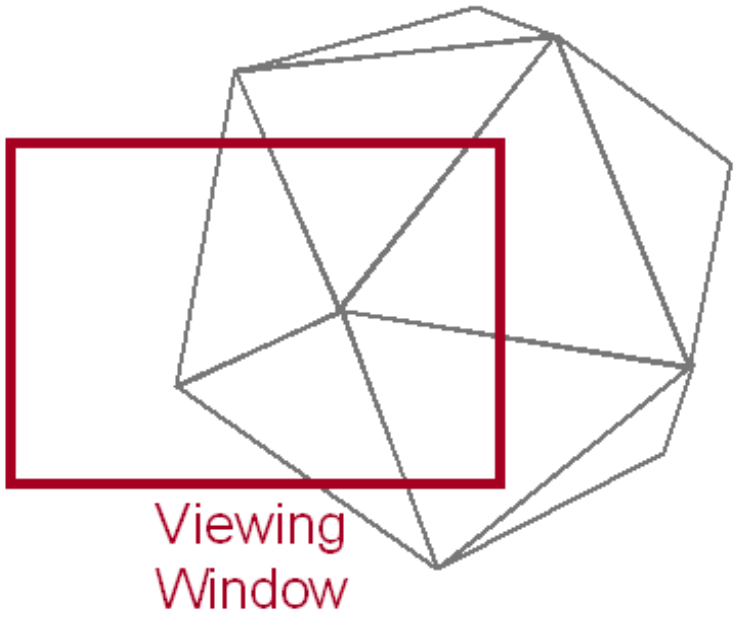
Clipping

- **Avoid Drawing Parts of Primitives Outside Window**
  - Window defines part of scene being viewed
  - Must draw geometric primitives only inside window



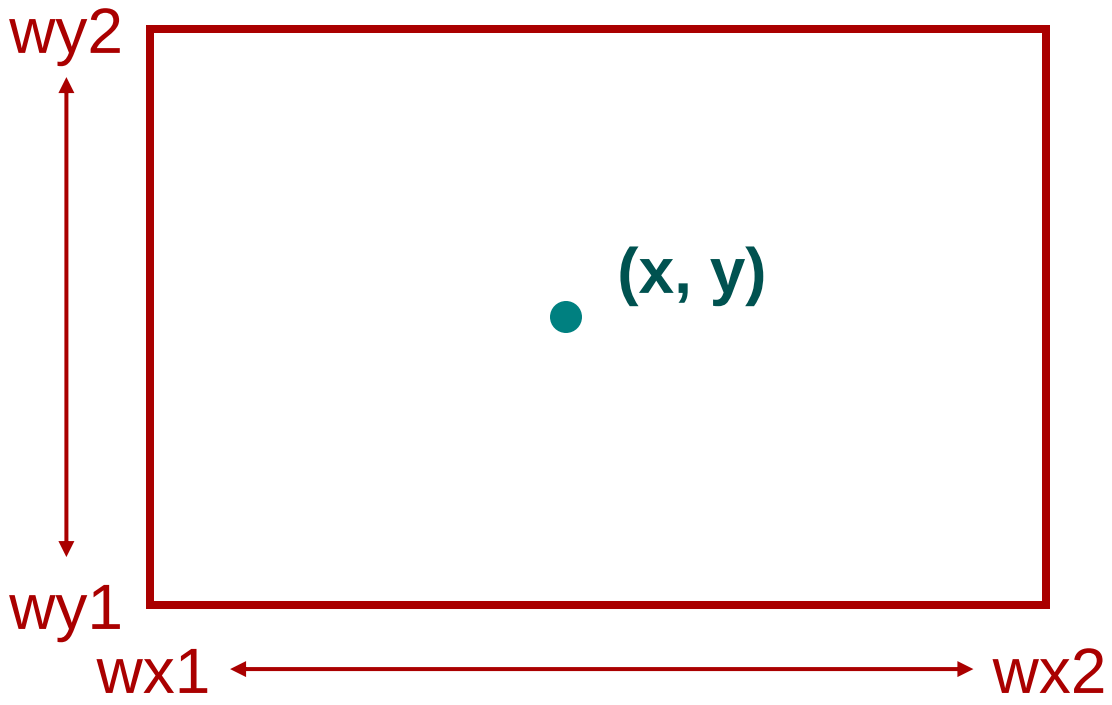
Clipping

- **Avoid Drawing Parts of Primitives Outside Window**
  - Points
  - Lines
  - Polygons
  - Circles
  - etc.



Point Clipping

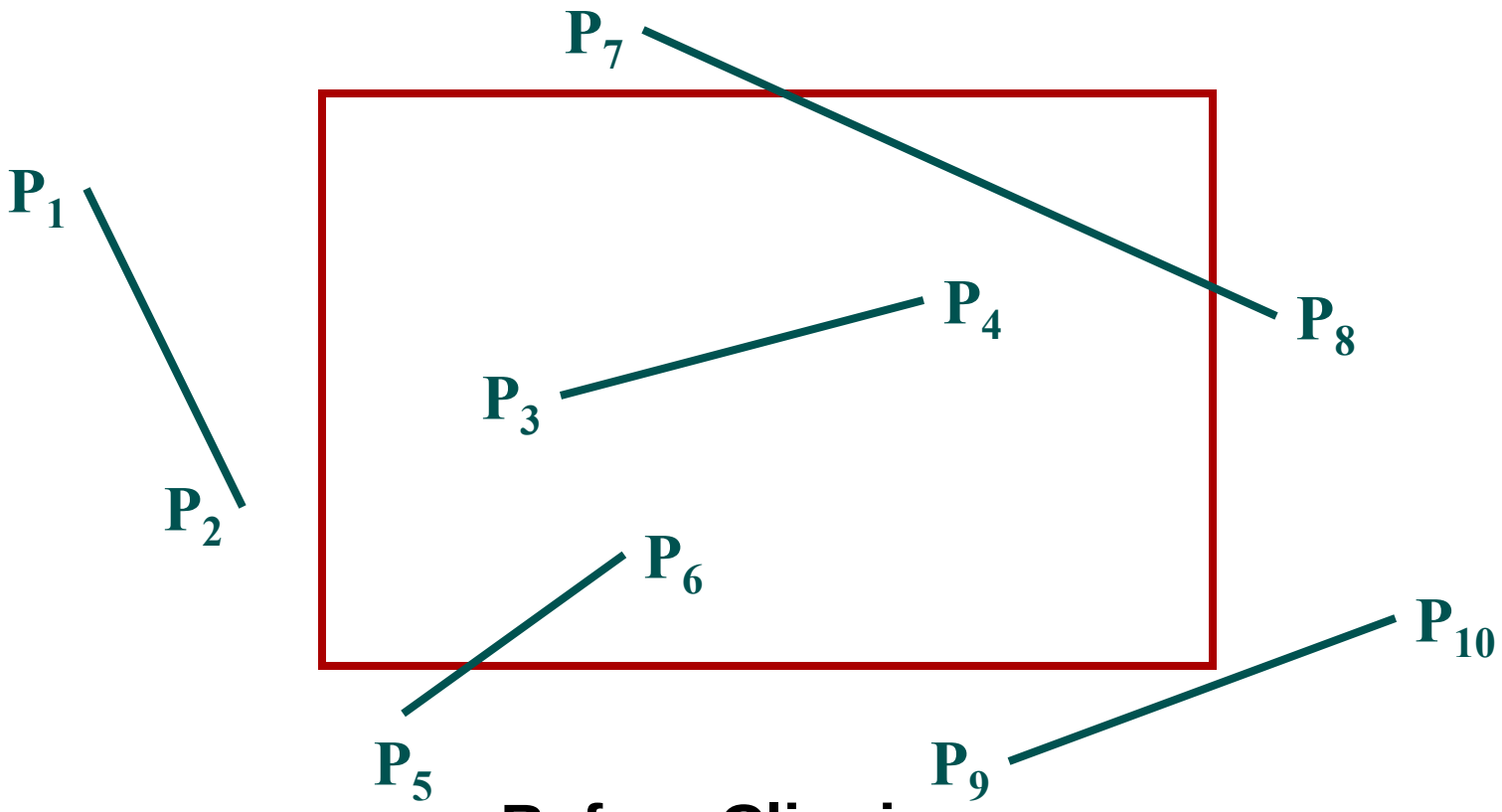
- **Is Point(x,y) Inside the Clip Window?**




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

Line Clipping

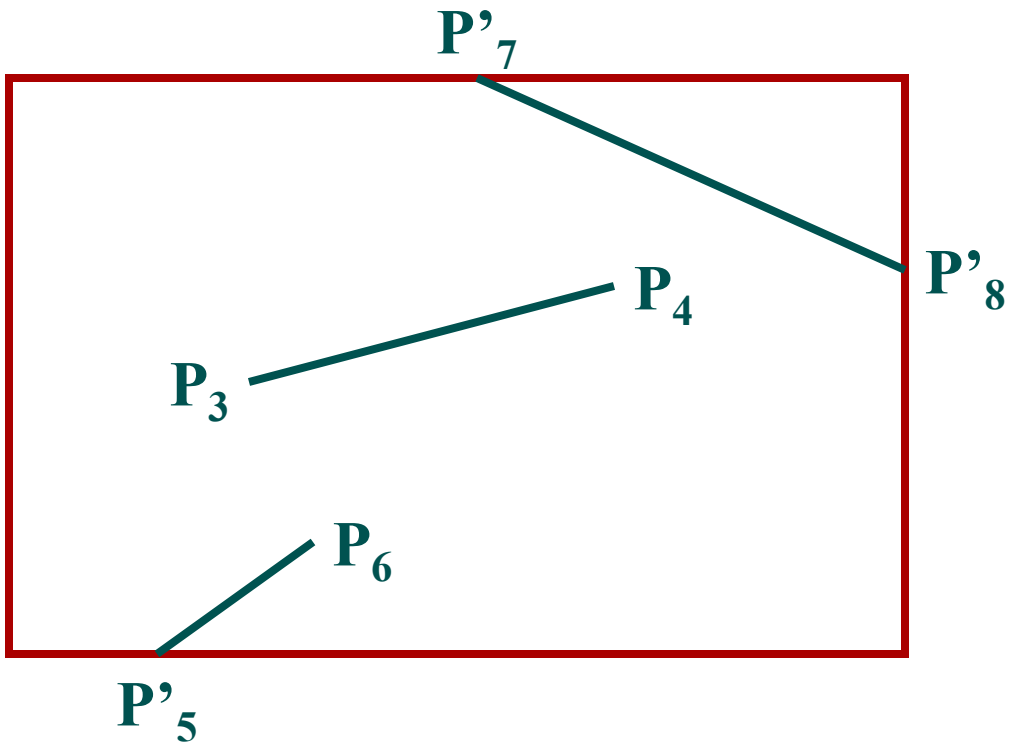
- **Find the Part of a Line Inside the Clip Window**



Before Clipping

Line Clipping

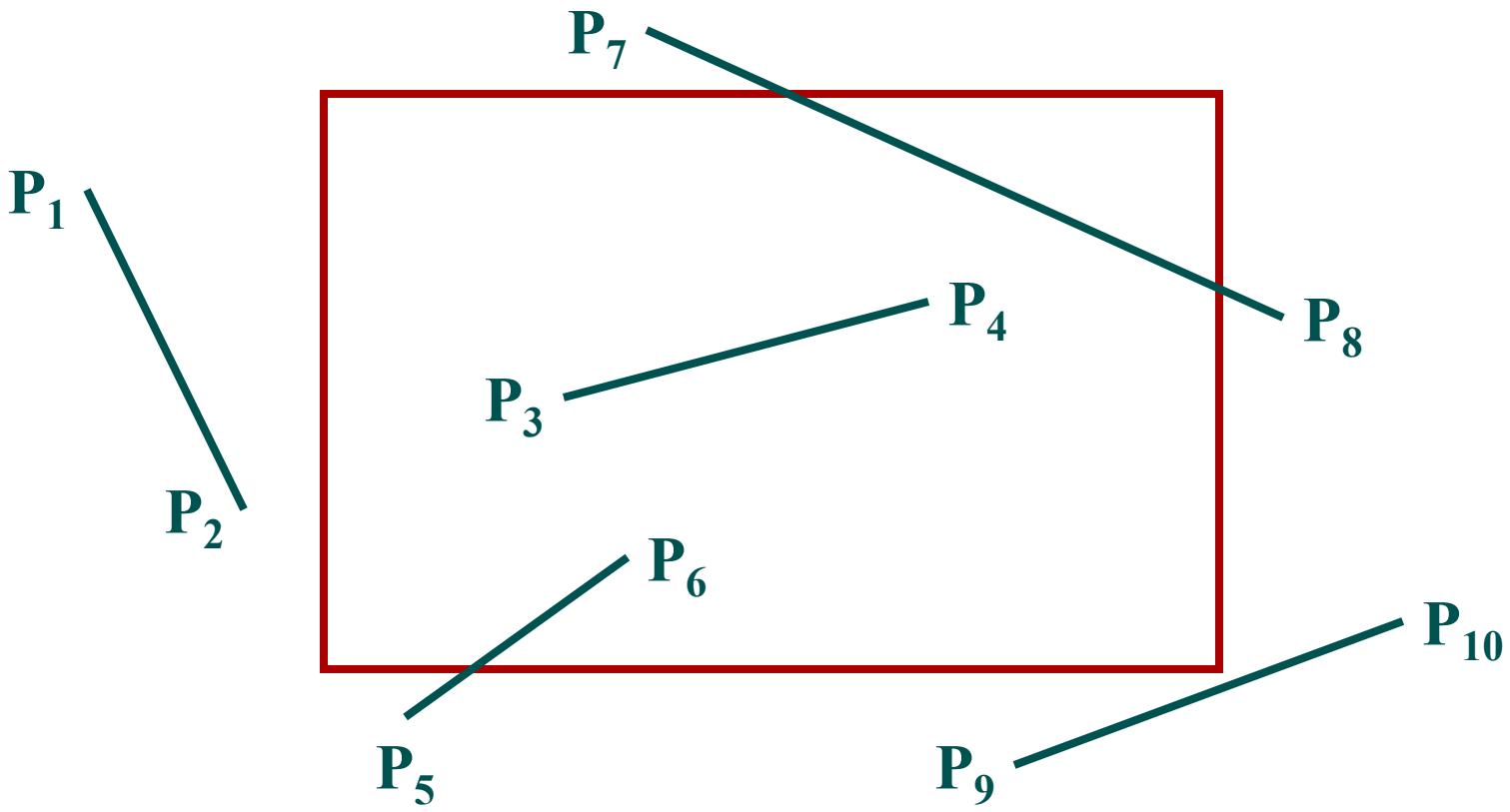
- **Find the Part of a Line Inside the Clip Window**



After Clipping

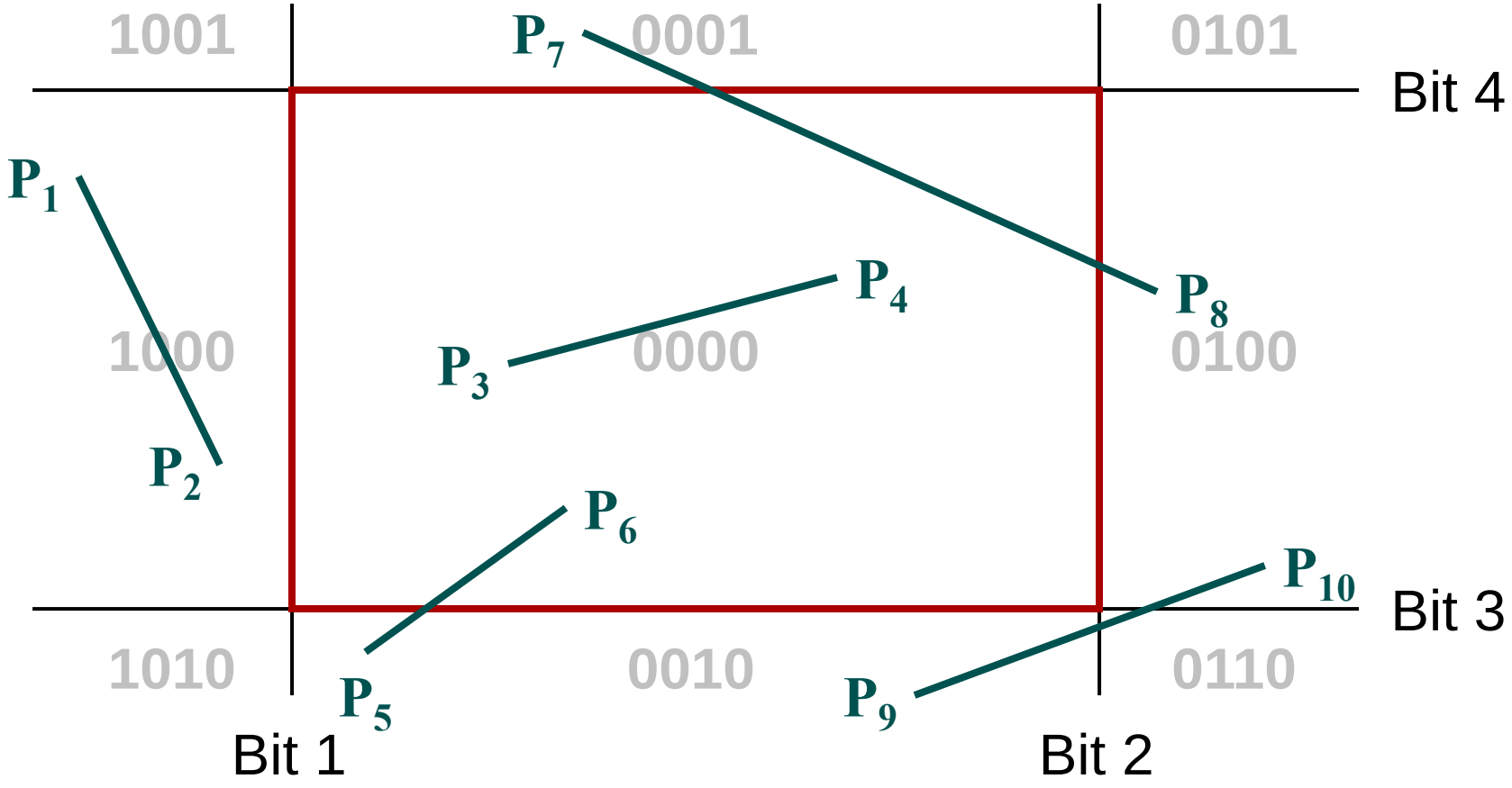
Cohen-Sutherland Line Clipping

- Use Simple Tests to Classify Easy Cases First



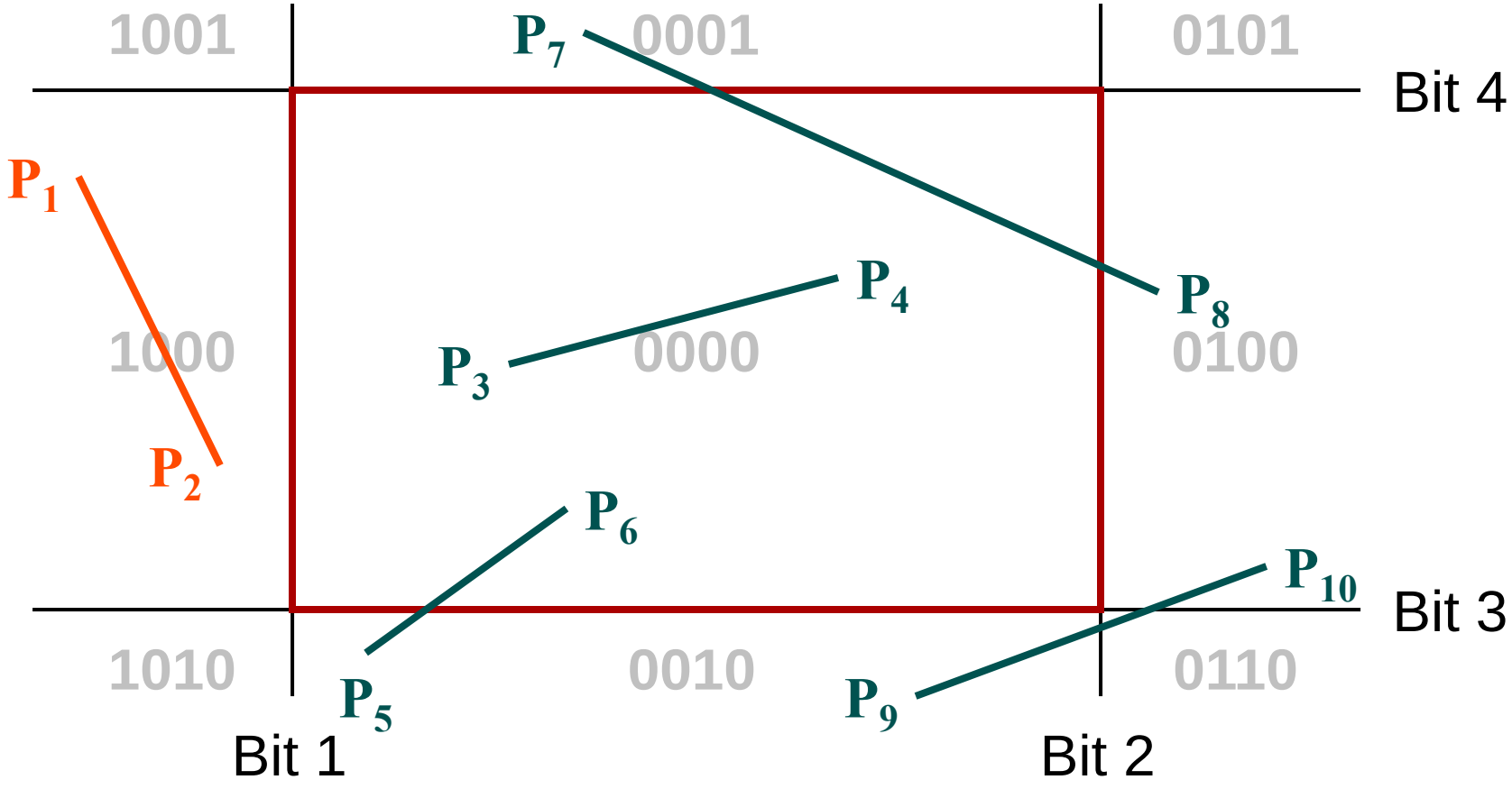
Cohen-Sutherland Line Clipping

- **Classify Some Lines Quickly by AND of Bit Codes Representing Regions of Two Endpoints (Must Be 0)**



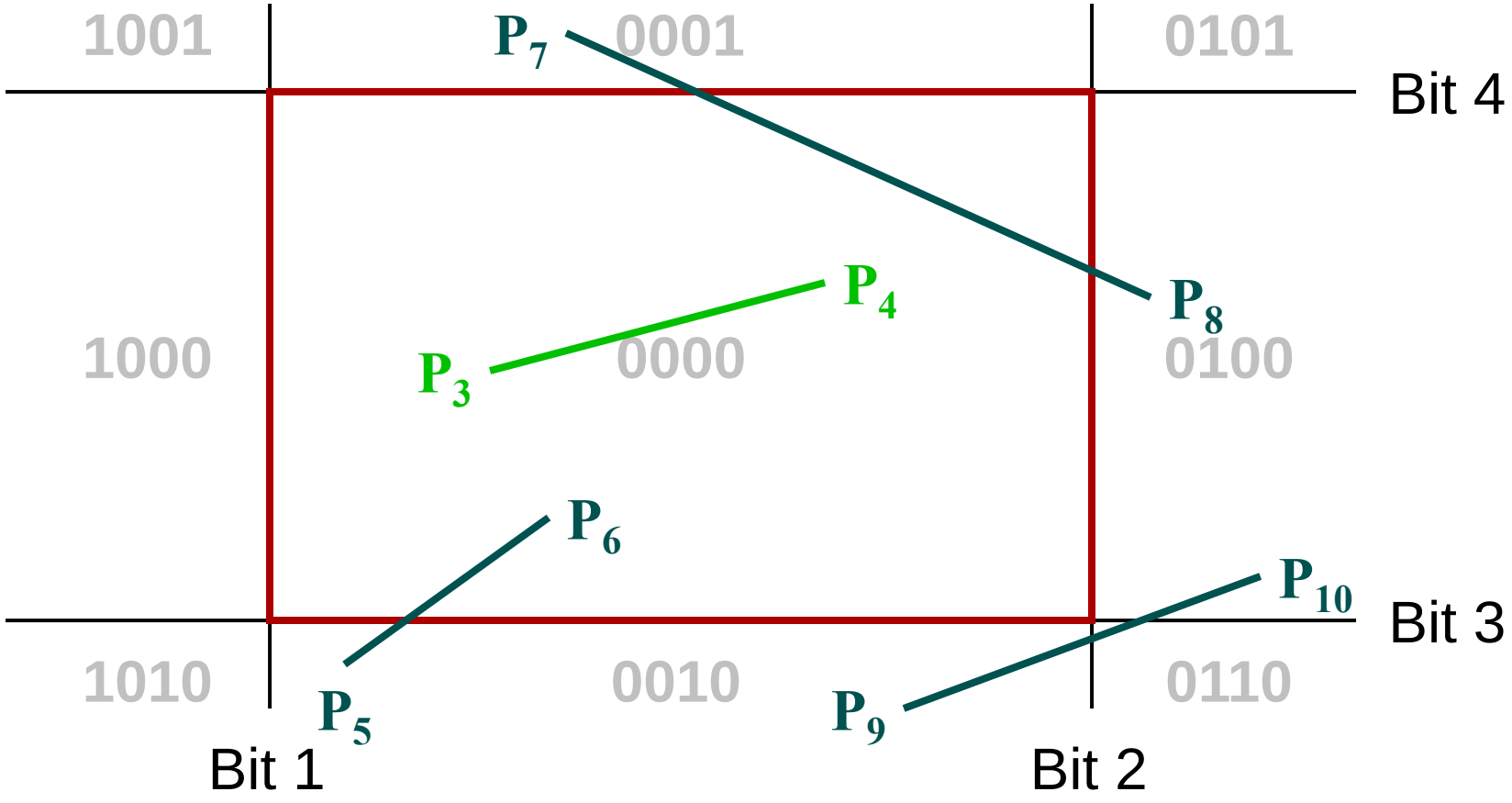
Cohen-Sutherland Line Clipping

- **Classify Some Lines Quickly by AND of Bit Codes Representing Regions of Two Endpoints (Must Be 0)**



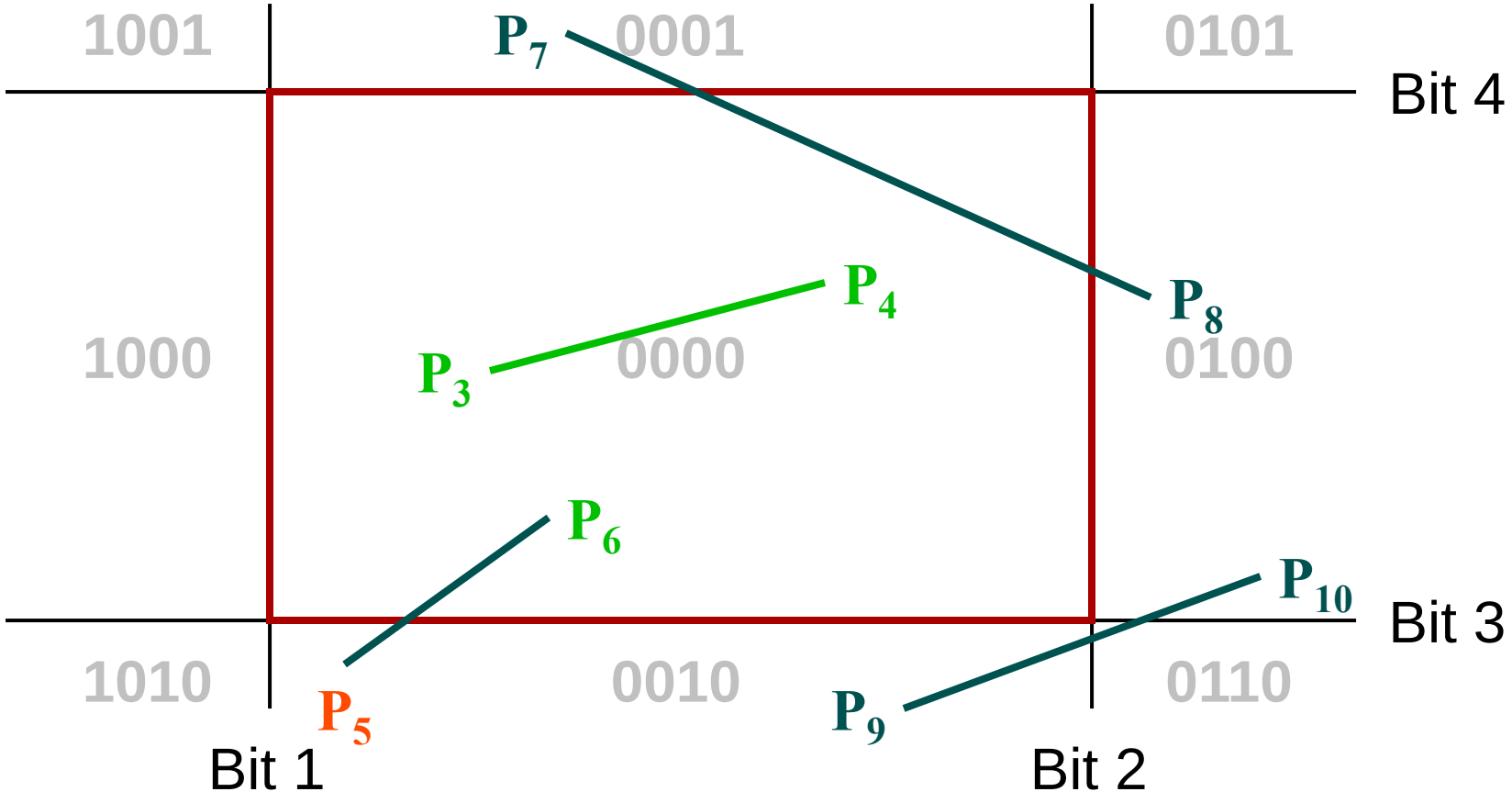
Cohen-Sutherland Line Clipping

- **Classify Some Lines Quickly by AND of Bit Codes Representing Regions of Two Endpoints (Must Be 0)**



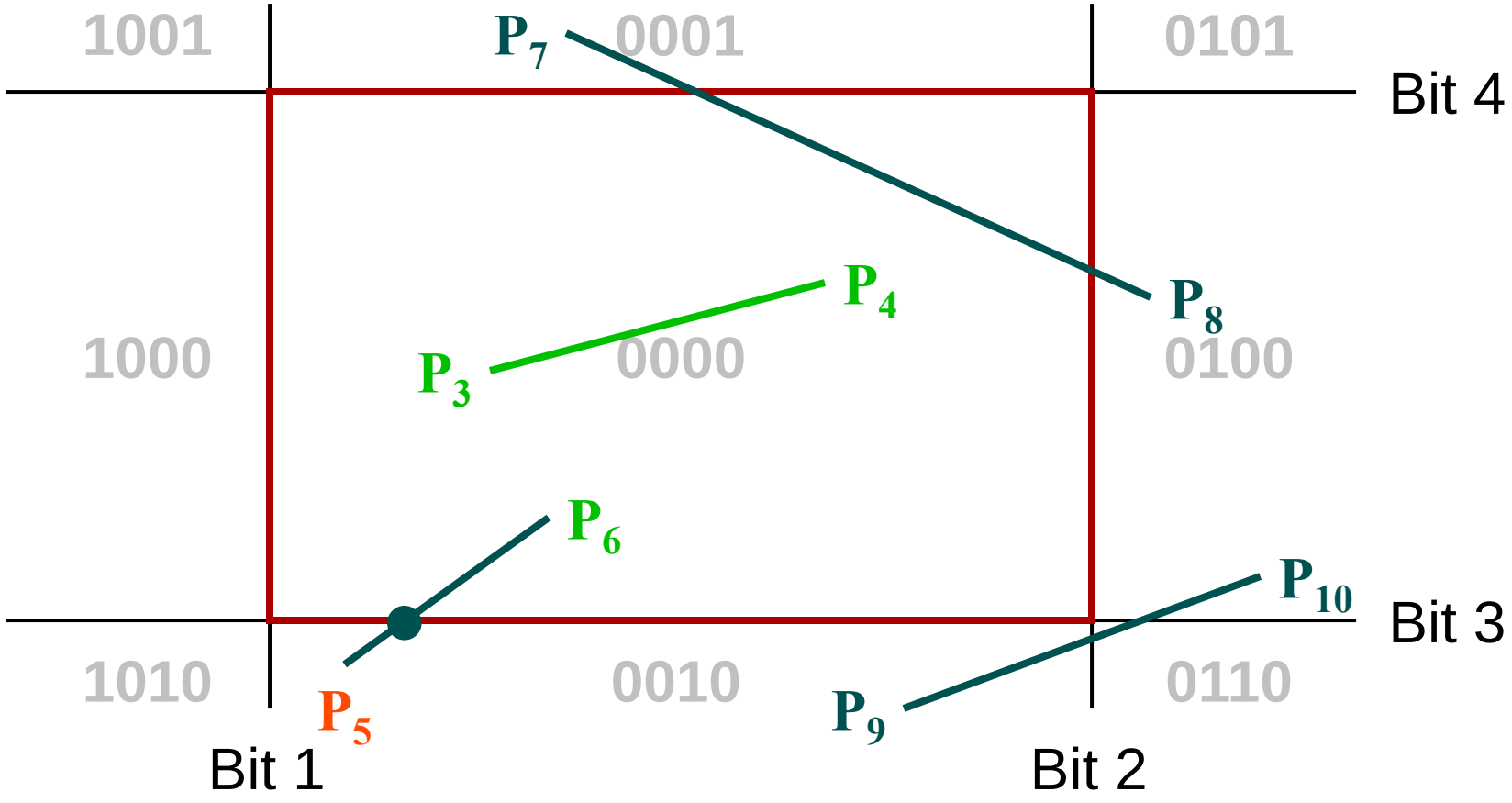
Cohen-Sutherland Line Clipping

- Compute Intersections with Window Boundary for Lines That Can't be Classified Quickly



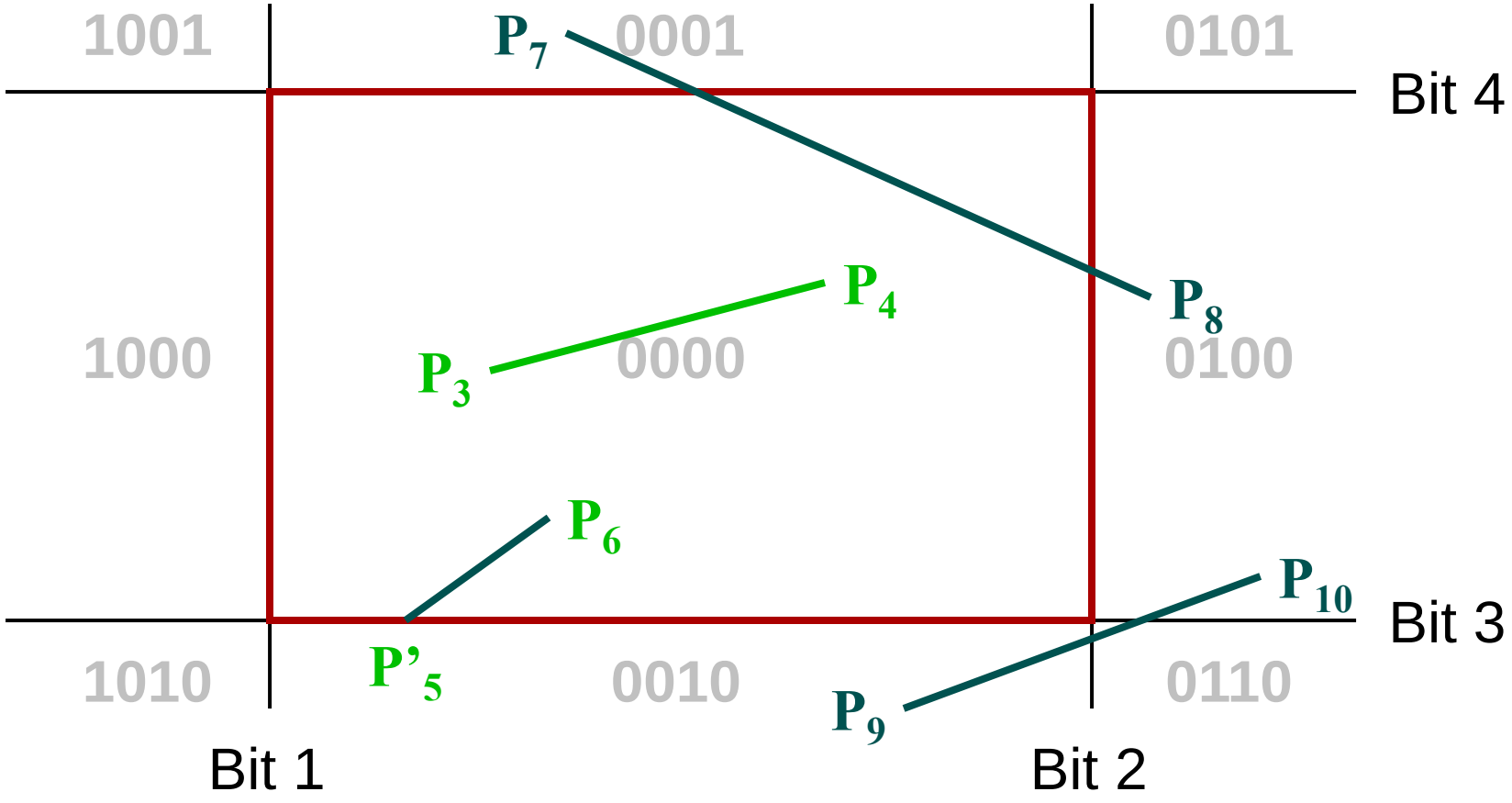
Cohen-Sutherland Line Clipping

- Compute Intersections with Window Boundary for Lines That Can't be Classified Quickly



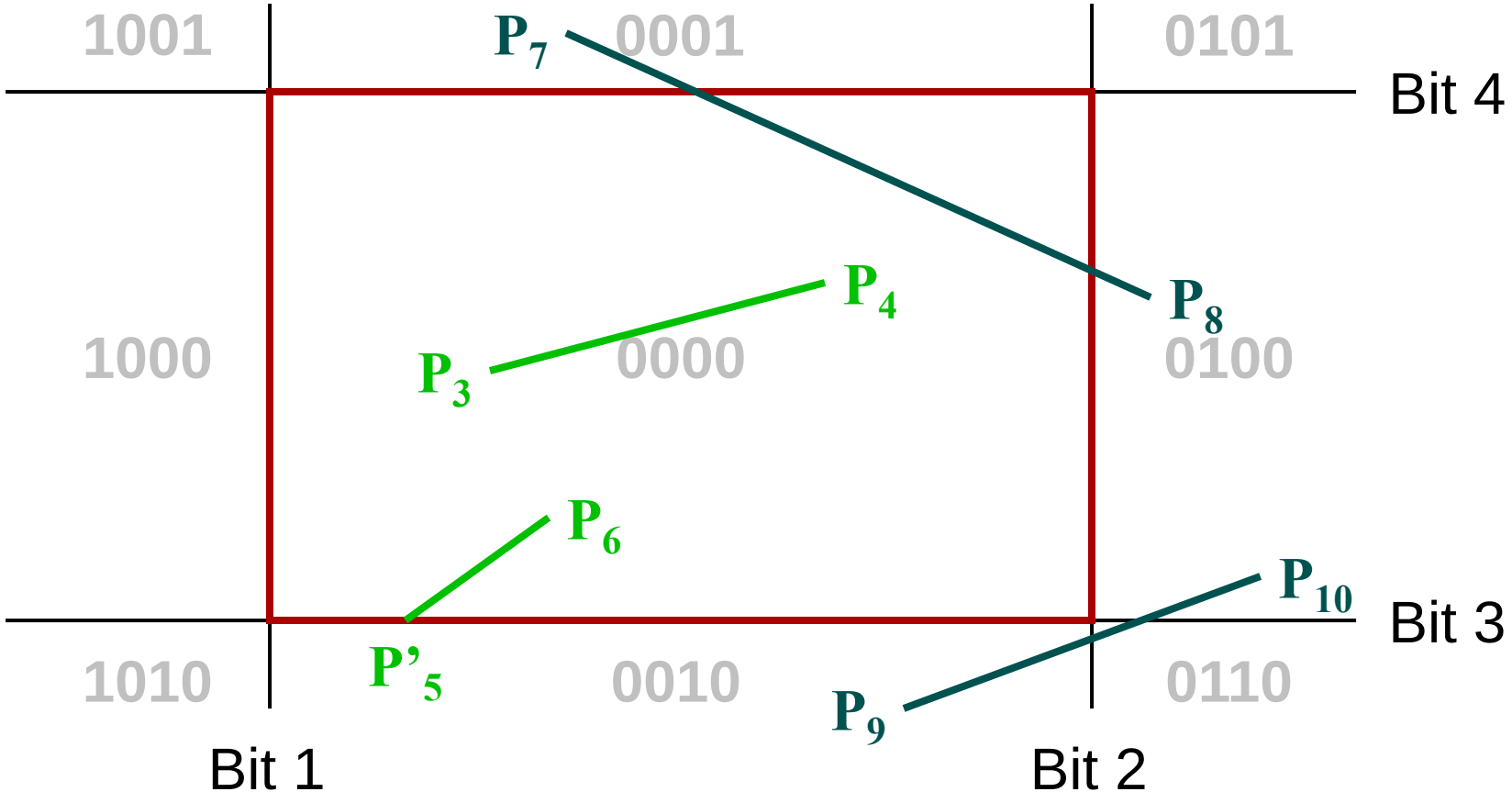
Cohen-Sutherland Line Clipping

- **Compute Intersections with Window Boundary for Lines That Can't be Classified Quickly**



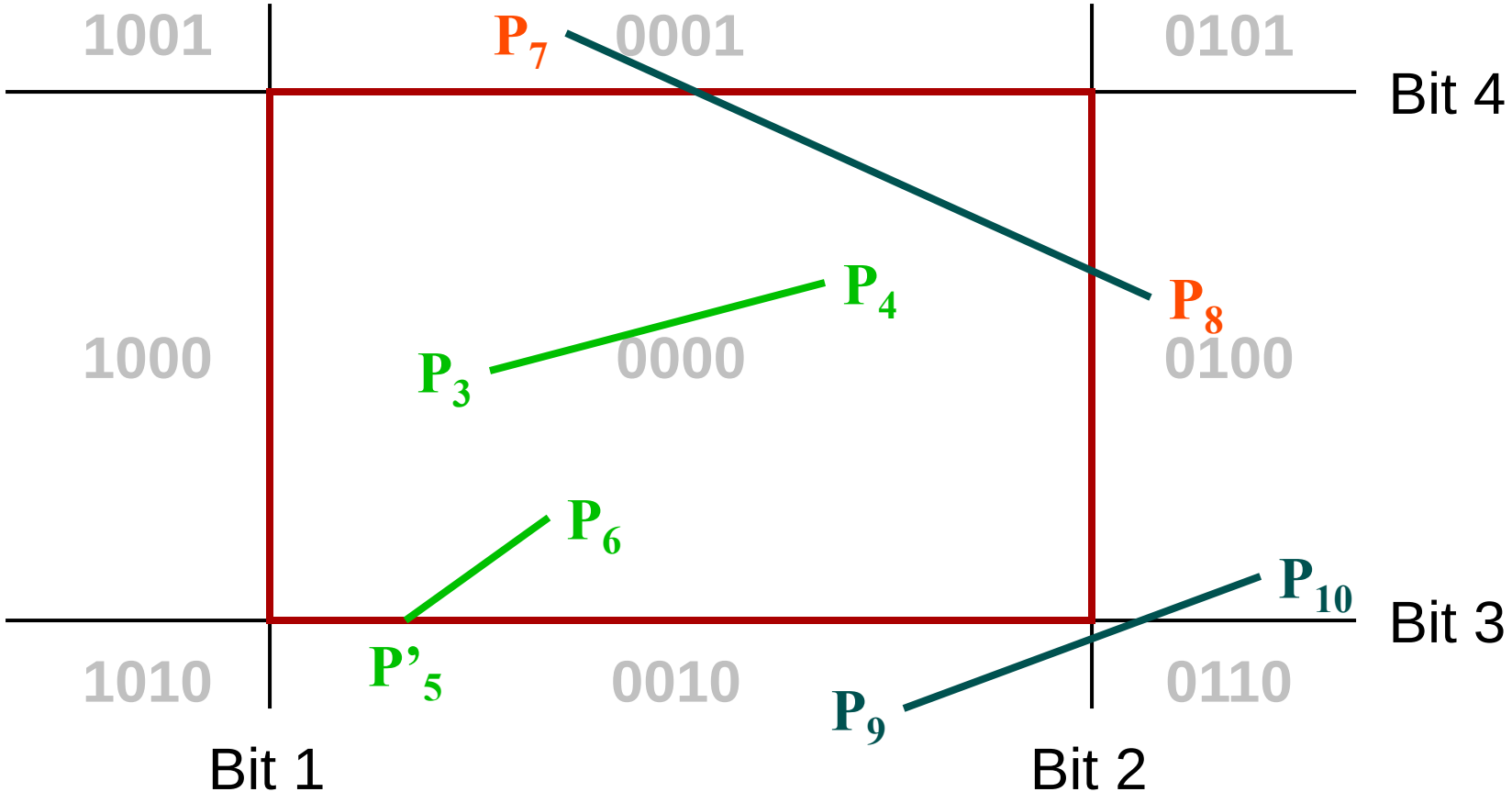
Cohen-Sutherland Line Clipping

- **Compute Intersections with Window Boundary for Lines That Can't be Classified Quickly**



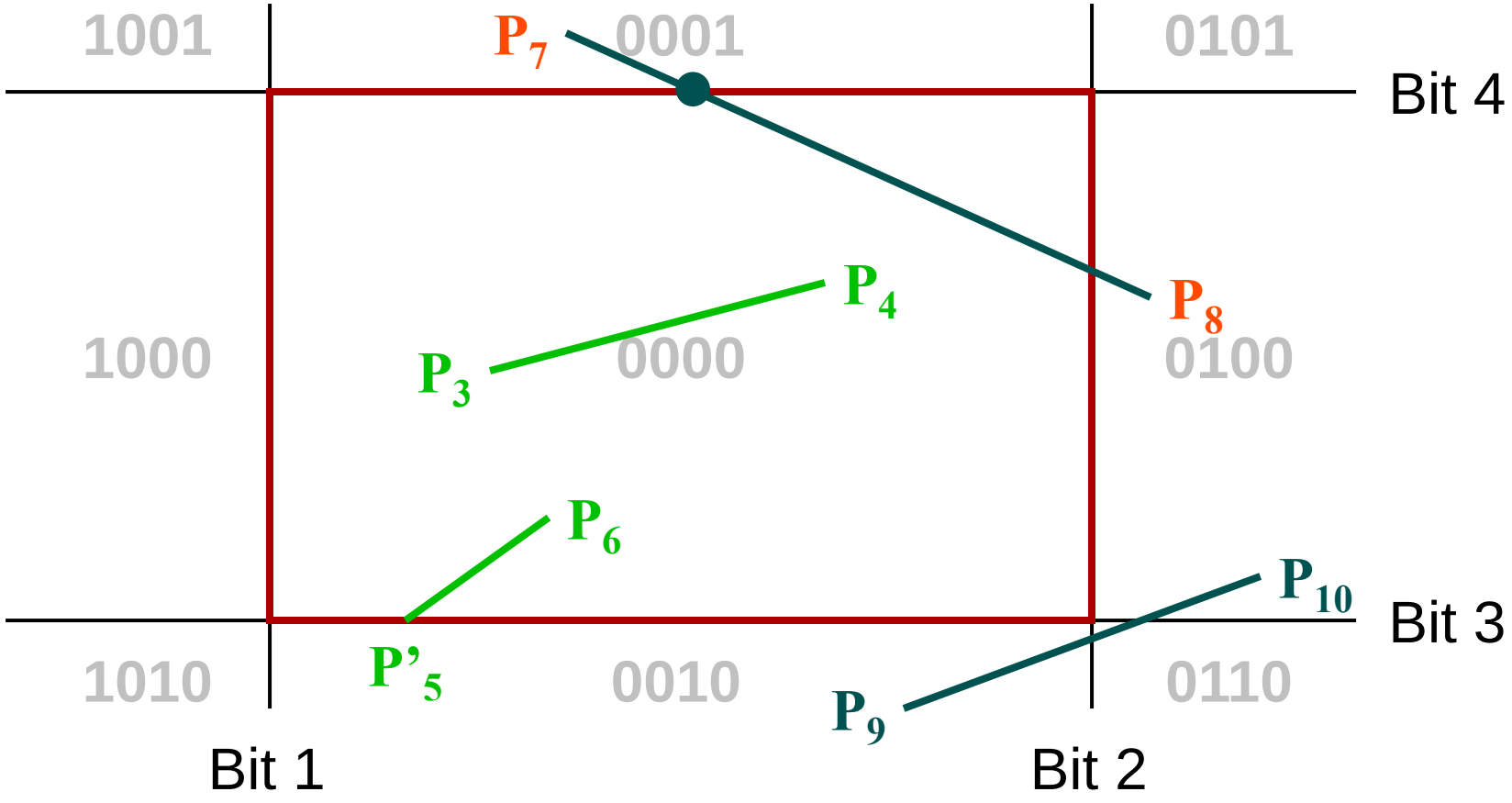
Cohen-Sutherland Line Clipping

- Compute Intersections with Window Boundary for Lines That Can't be Classified Quickly



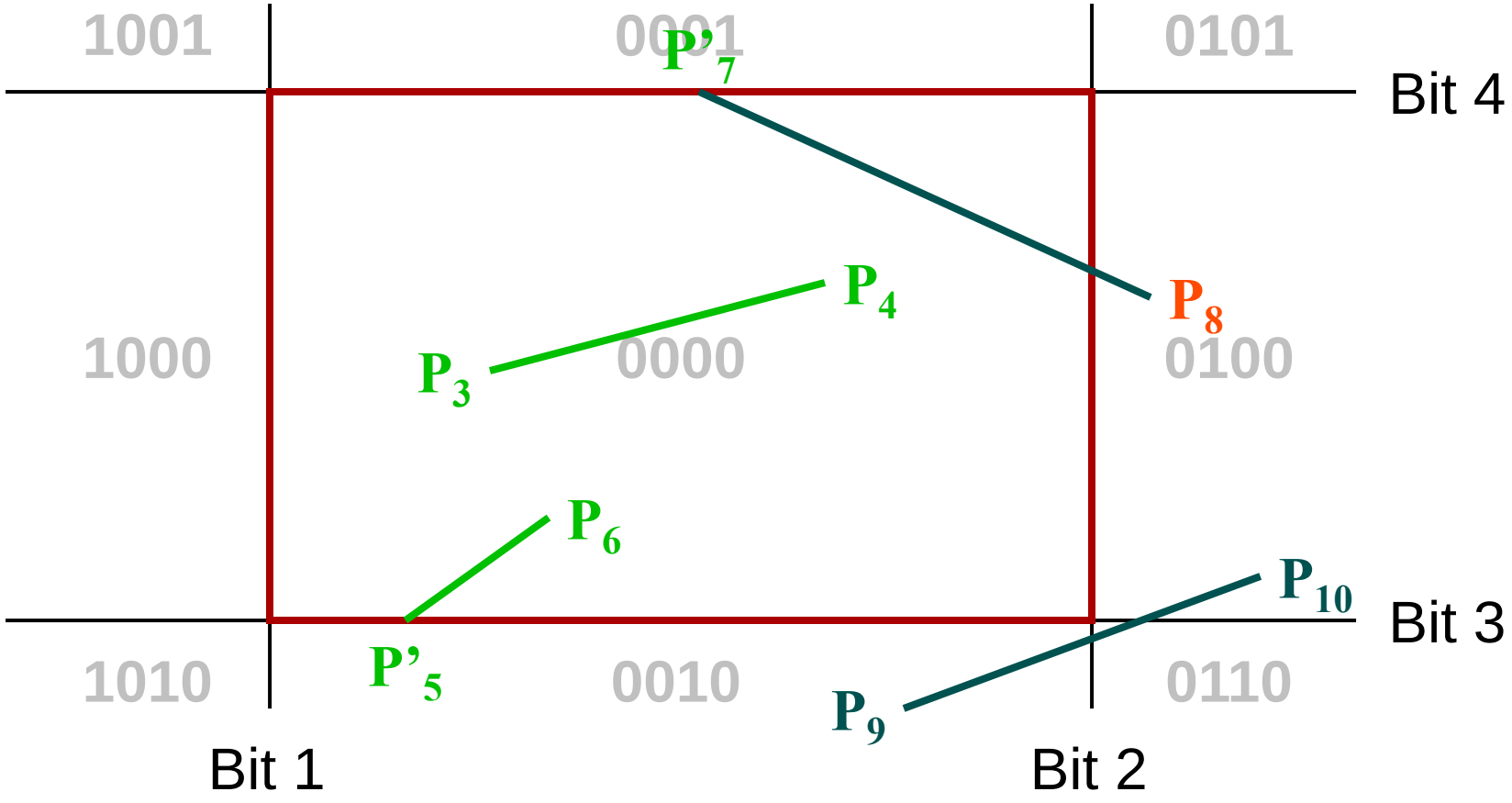
Cohen-Sutherland Line Clipping

- Compute Intersections with Window Boundary for Lines That Can't be Classified Quickly



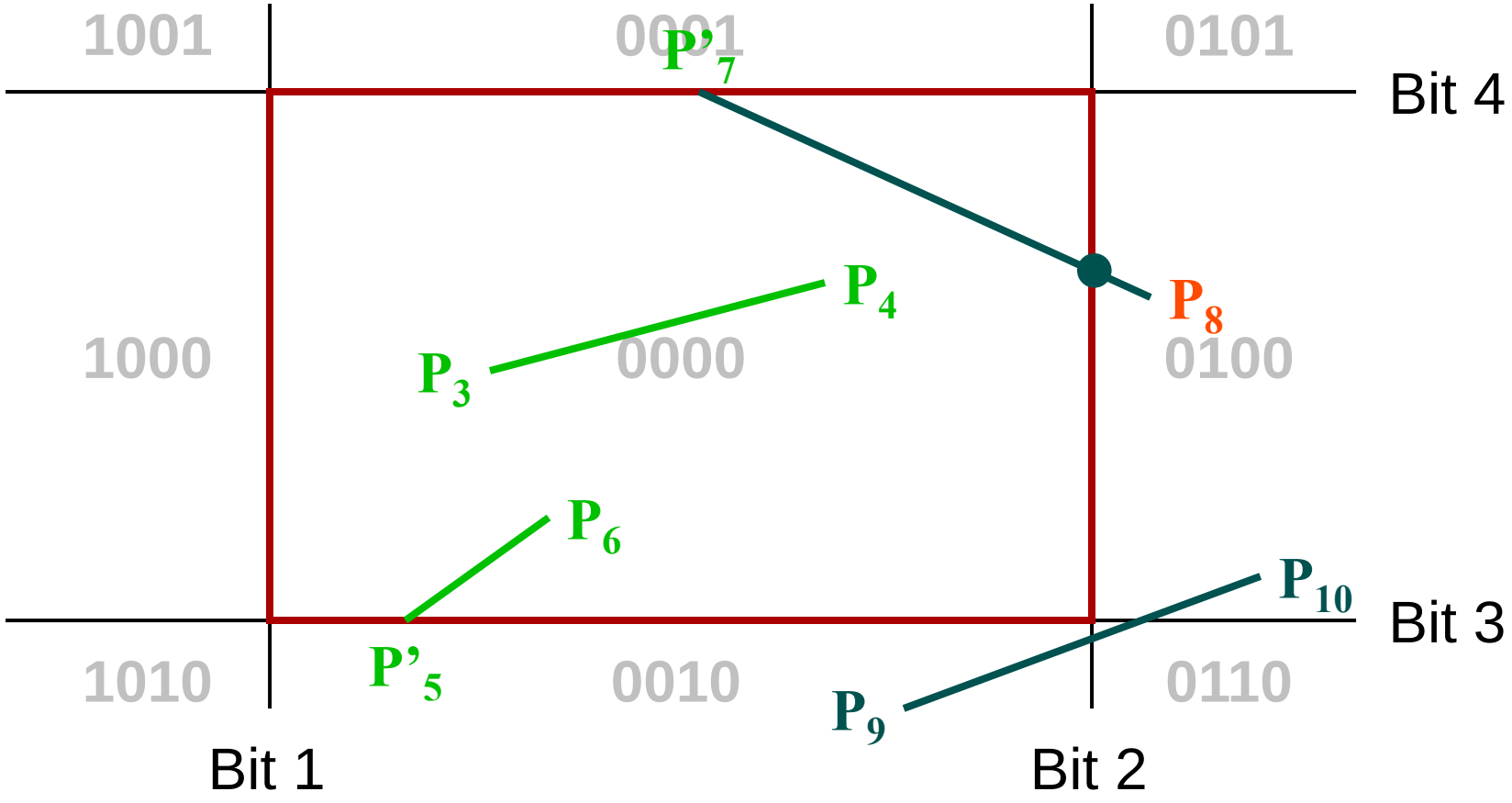
Cohen-Sutherland Line Clipping

- Compute Intersections with Window Boundary for Lines That Can't be Classified Quickly



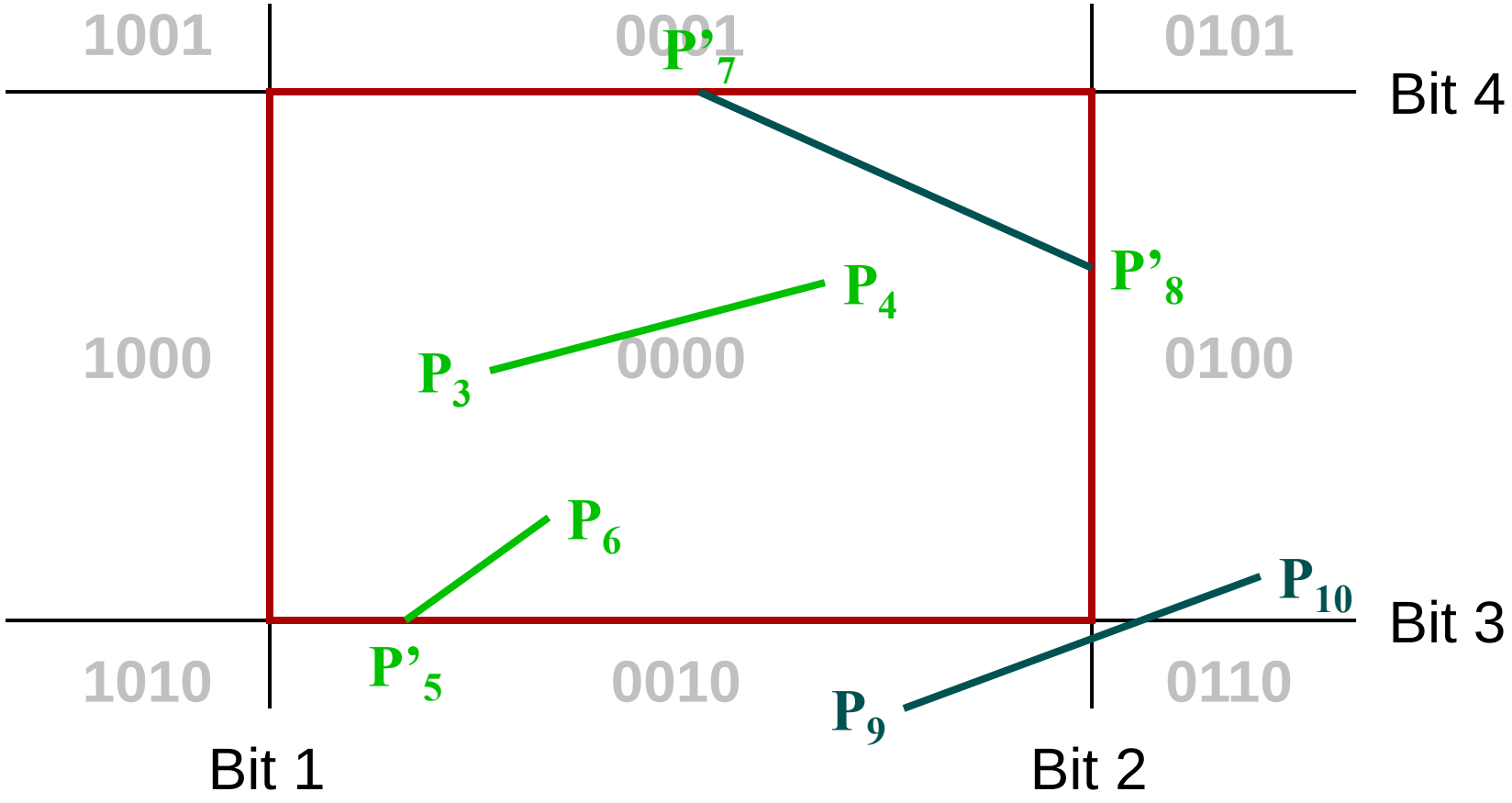
Cohen-Sutherland Line Clipping

- Compute Intersections with Window Boundary for Lines That Can't be Classified Quickly



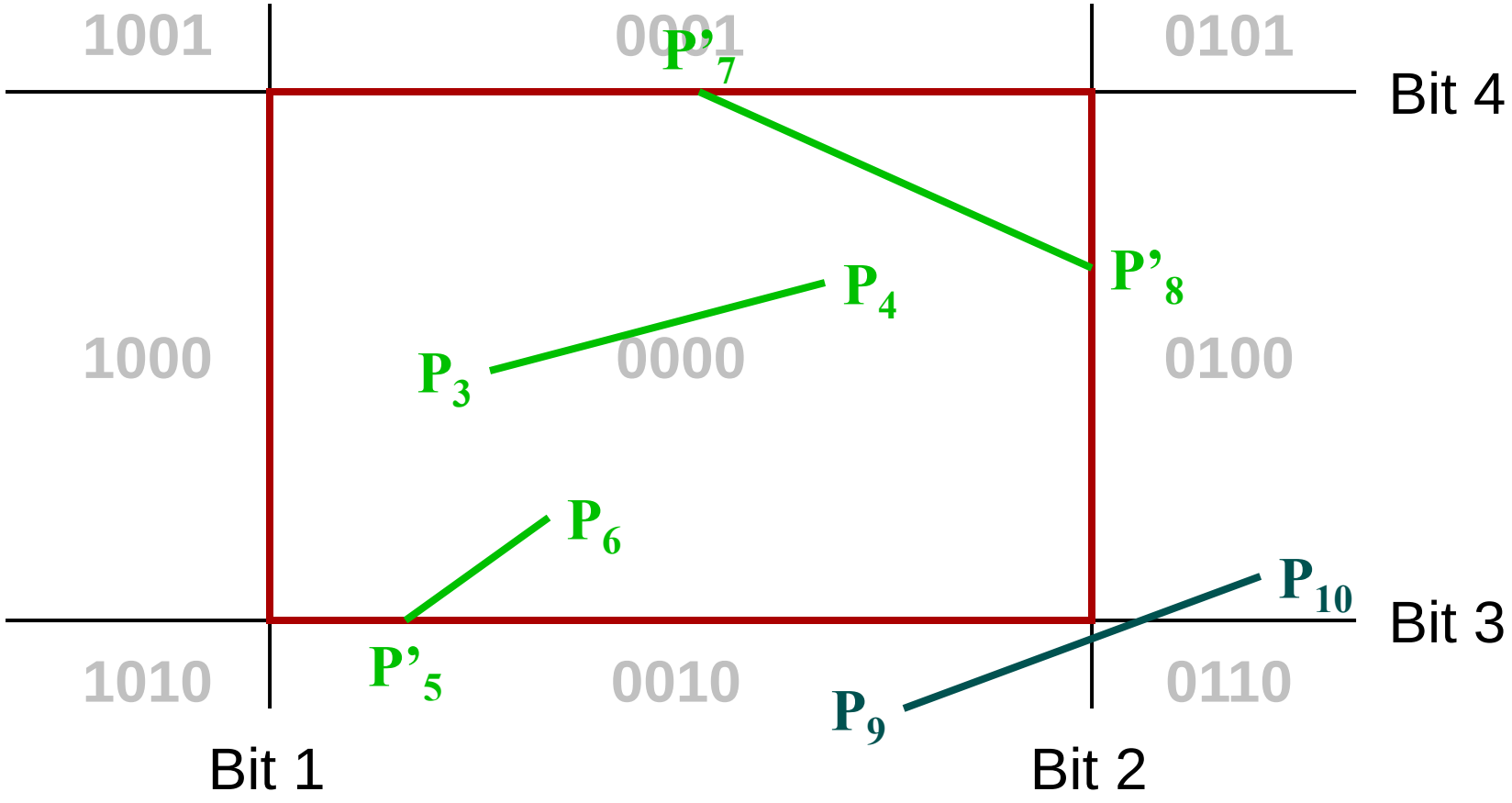
Cohen-Sutherland Line Clipping

- Compute Intersections with Window Boundary for Lines That Can't be Classified Quickly



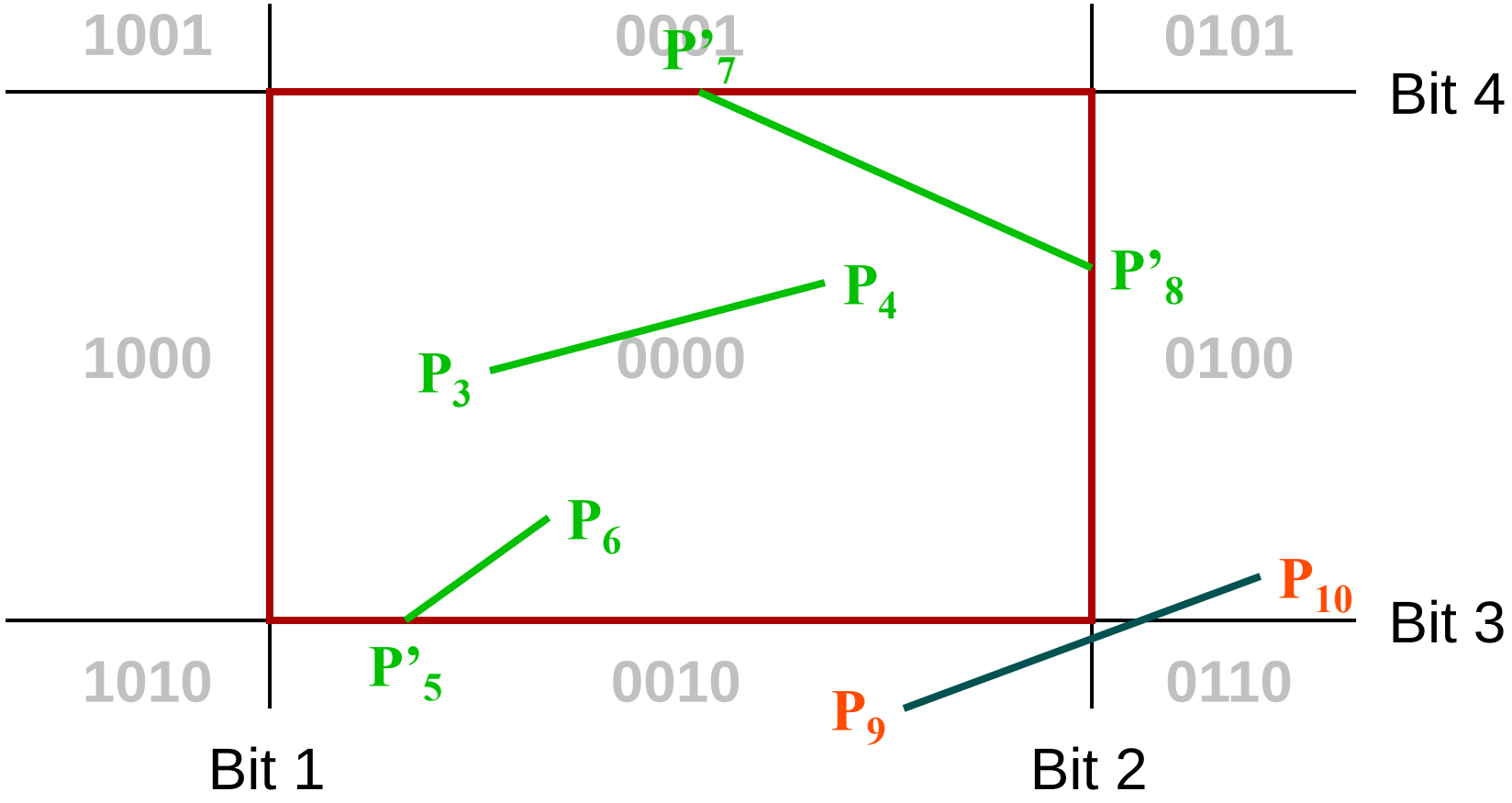
Cohen-Sutherland Line Clipping

- Compute Intersections with Window Boundary for Lines That Can't be Classified Quickly



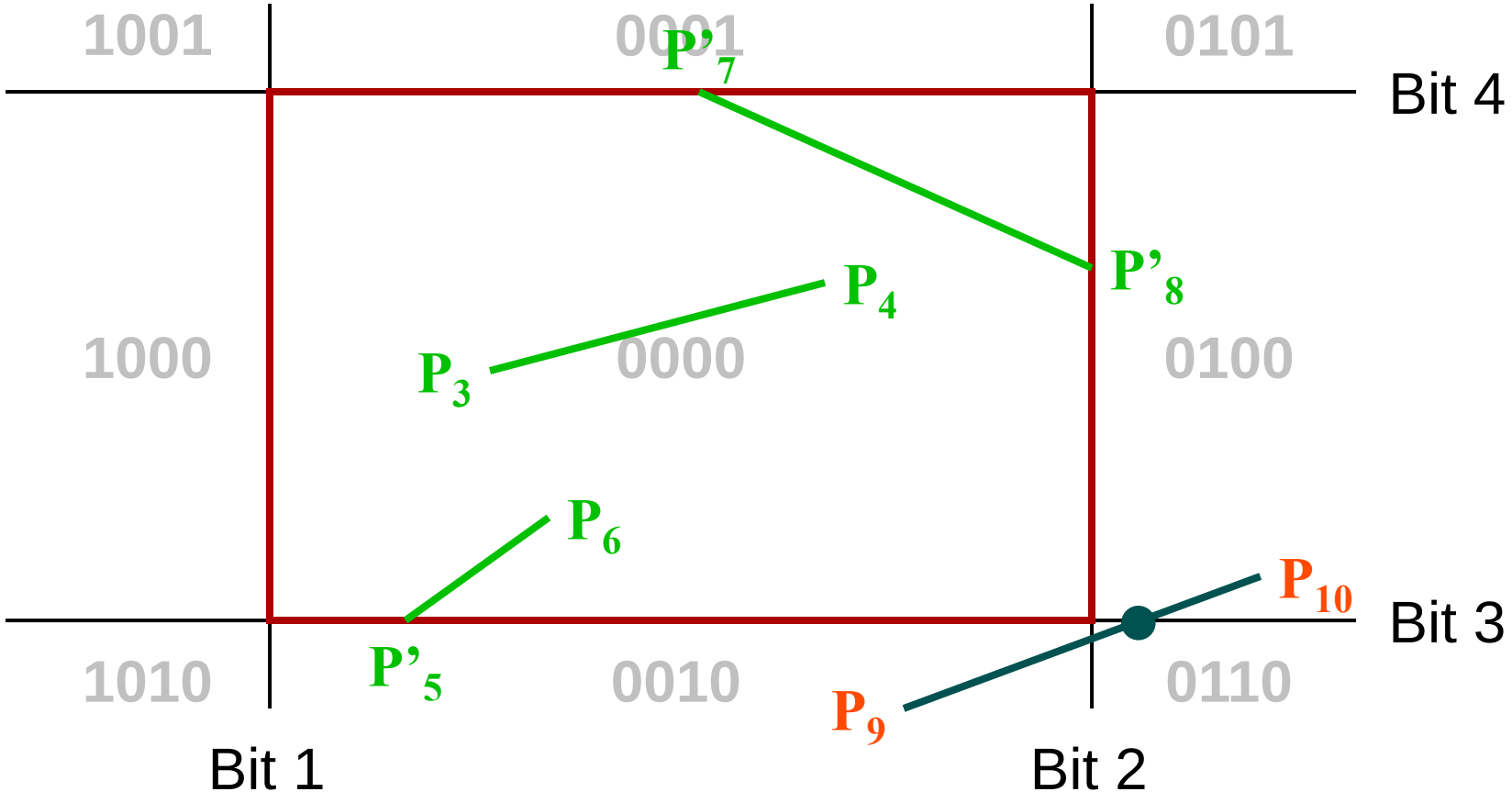
Cohen-Sutherland Line Clipping

- Compute Intersections with Window Boundary for Lines That Can't be Classified Quickly



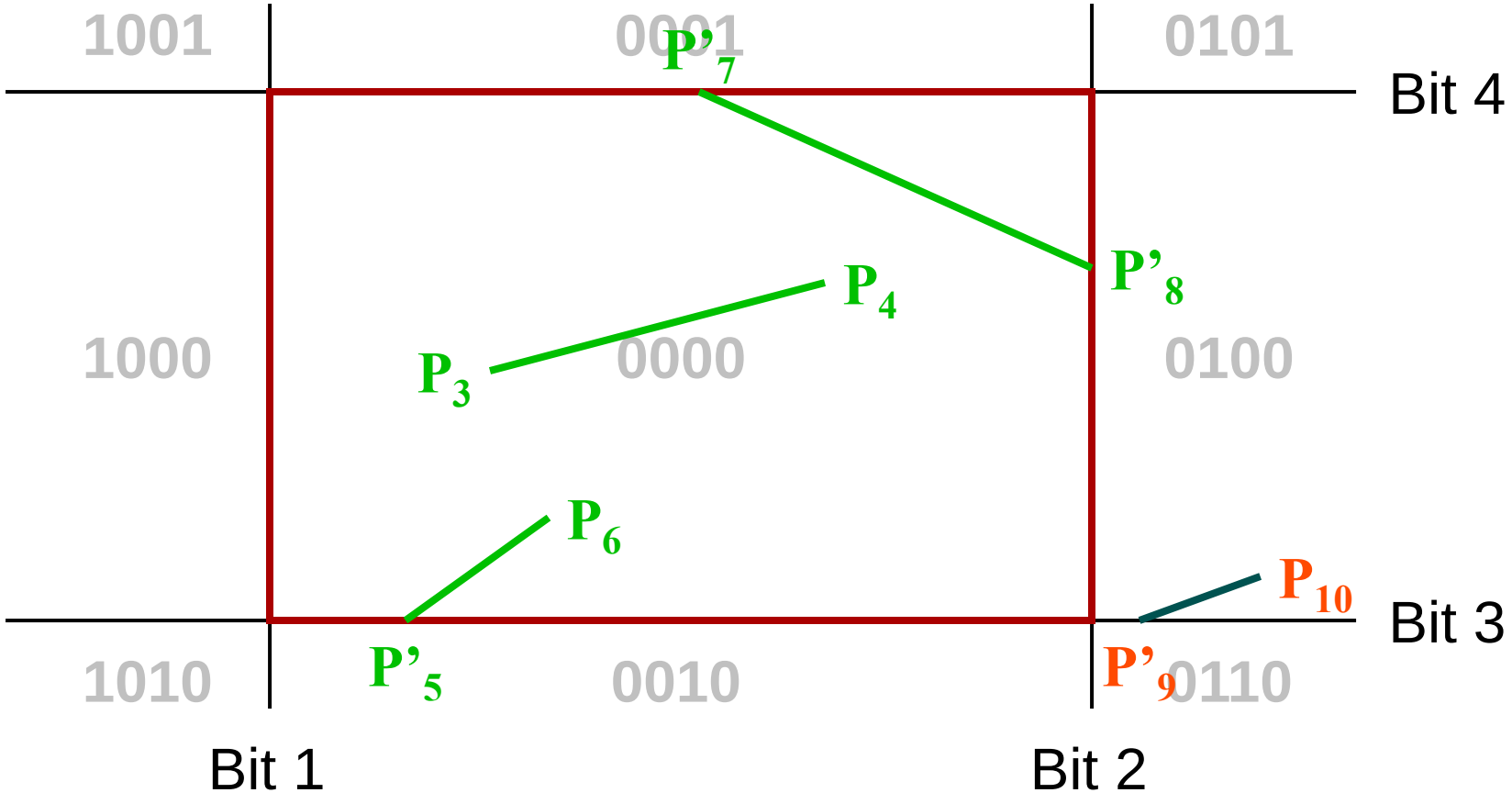
Cohen-Sutherland Line Clipping

- Compute Intersections with Window Boundary for Lines That Can't be Classified Quickly



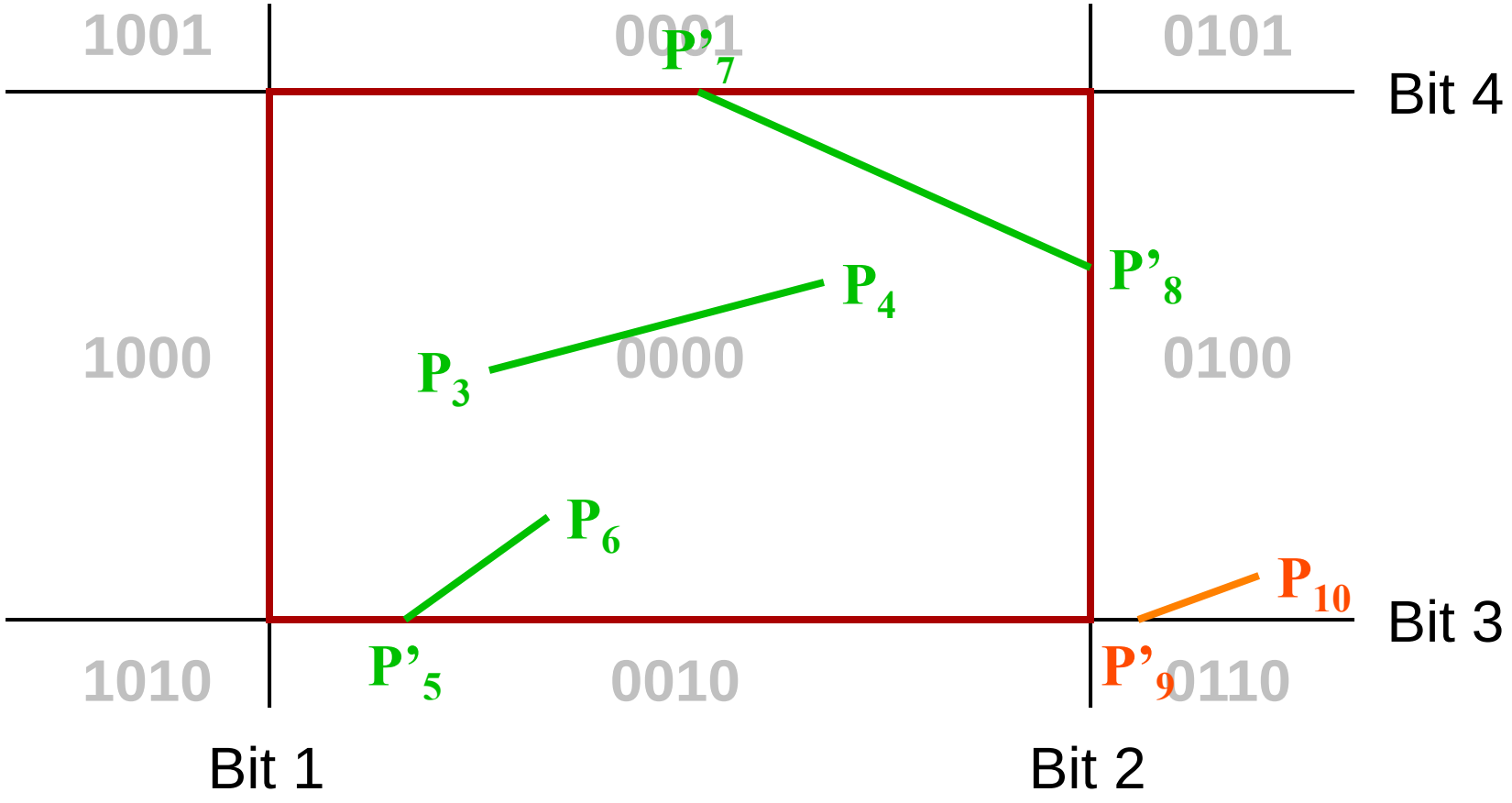
Cohen-Sutherland Line Clipping

- Compute Intersections with Window Boundary for Lines That Can't be Classified Quickly



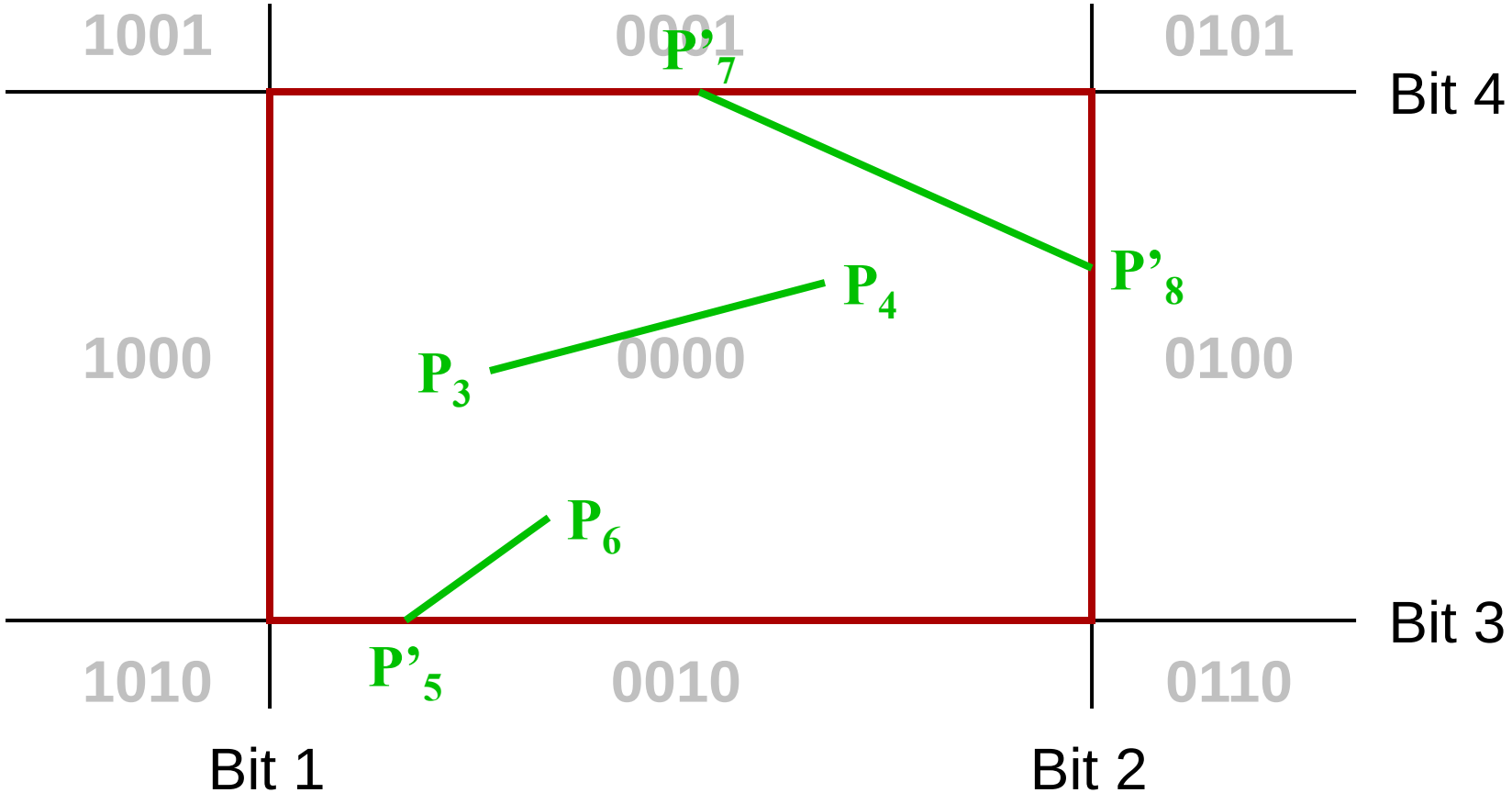
Cohen-Sutherland Line Clipping

- **Compute Intersections with Window Boundary for Lines That Can't be Classified Quickly**



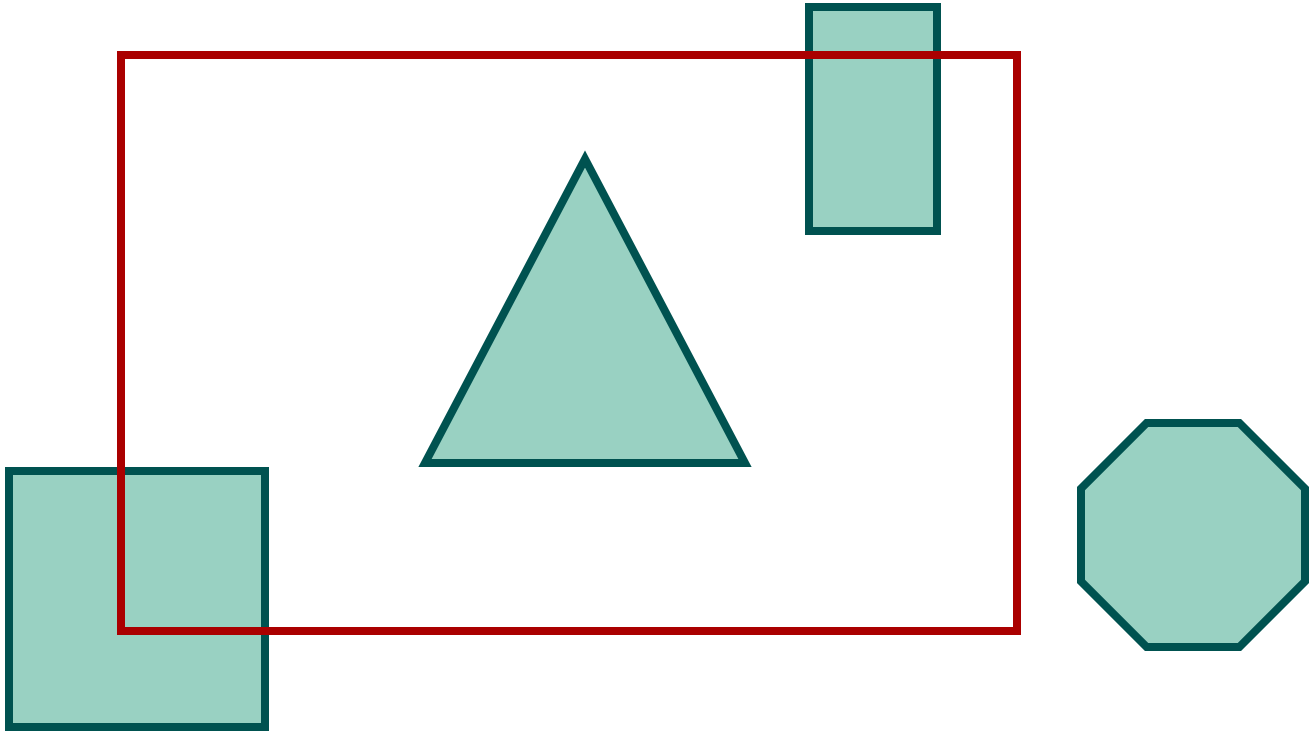
Cohen-Sutherland Line Clipping

- **Compute Intersections with Window Boundary for Lines That Can't be Classified Quickly**



Polygon Clipping

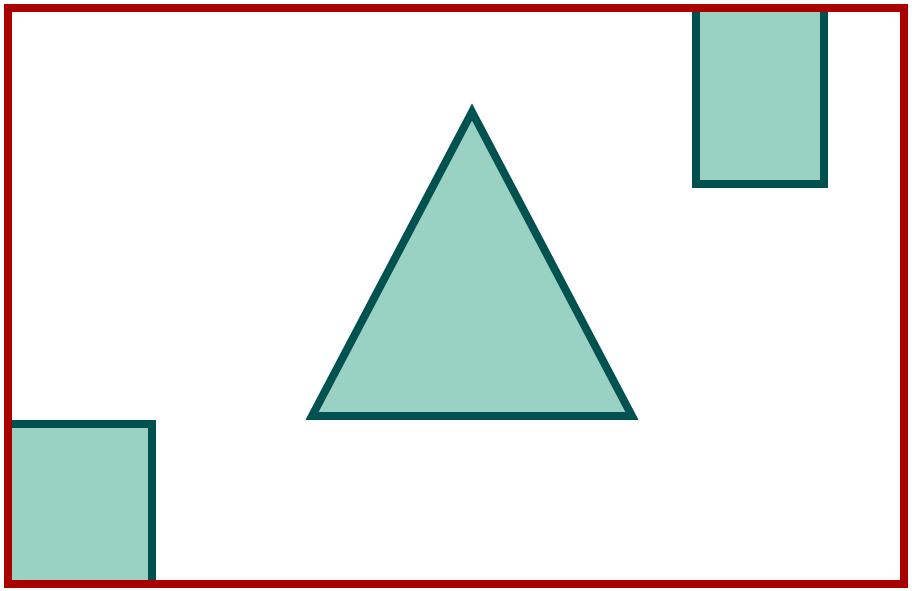
- **Find the Part of a Polygon Inside the Clip Window?**



Before Clipping

Polygon Clipping

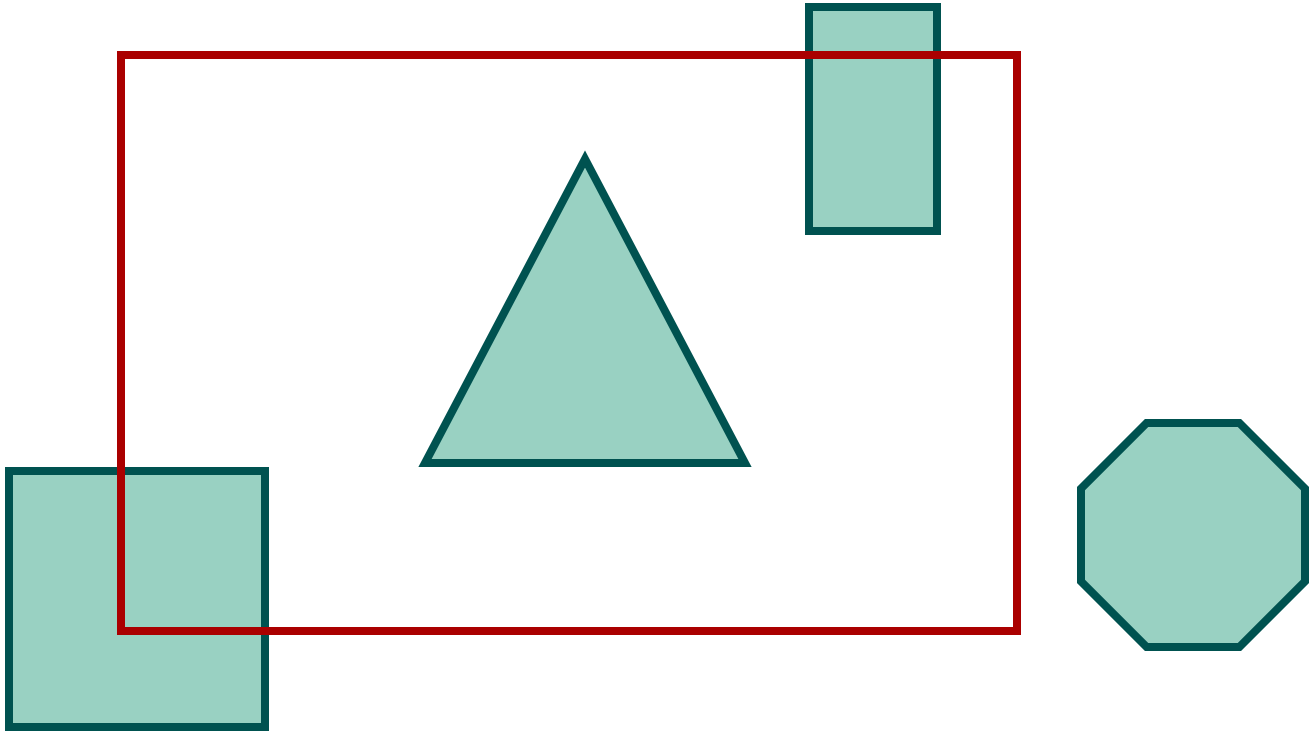
- **Find the Part of a Polygon Inside the Clip Window?**



After Clipping

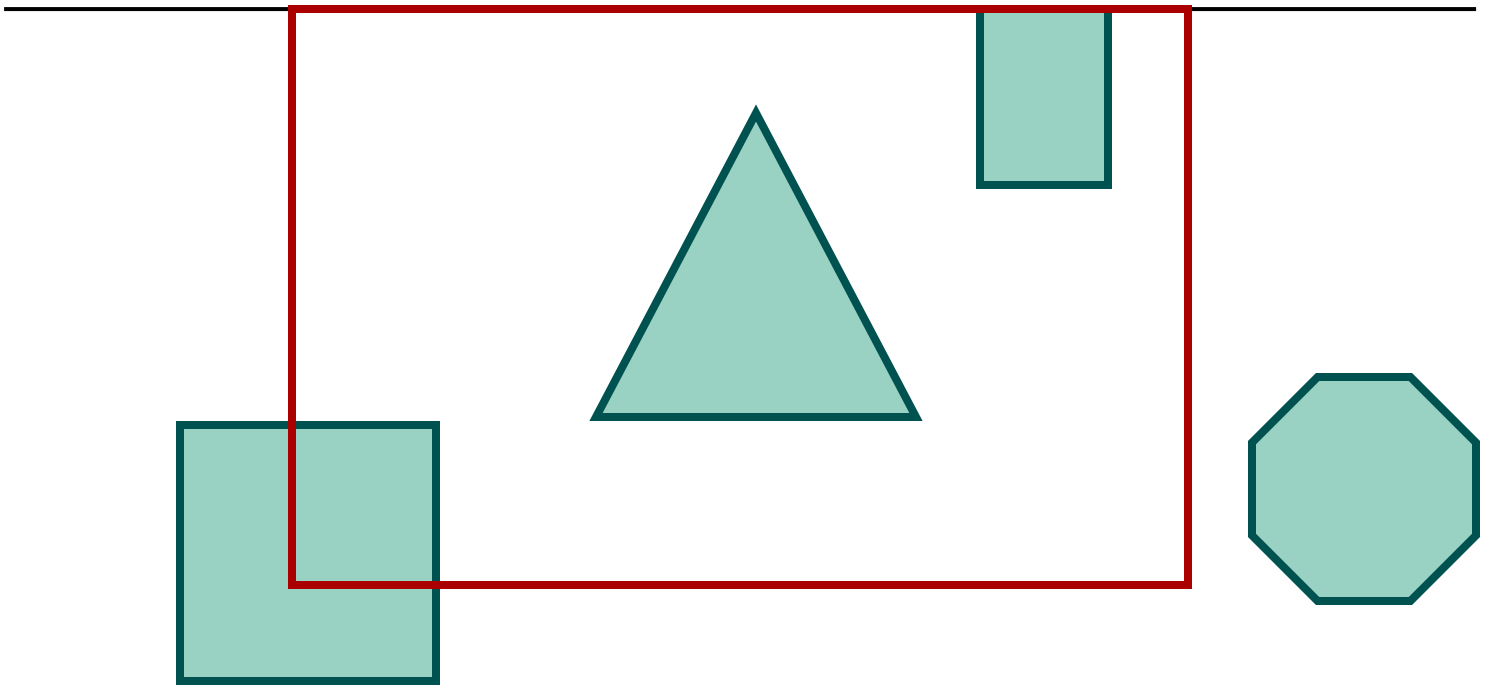
Sutherland-Hodgeman Polygon Clipping

- **Clip to Each Window Boundary One at a Time**



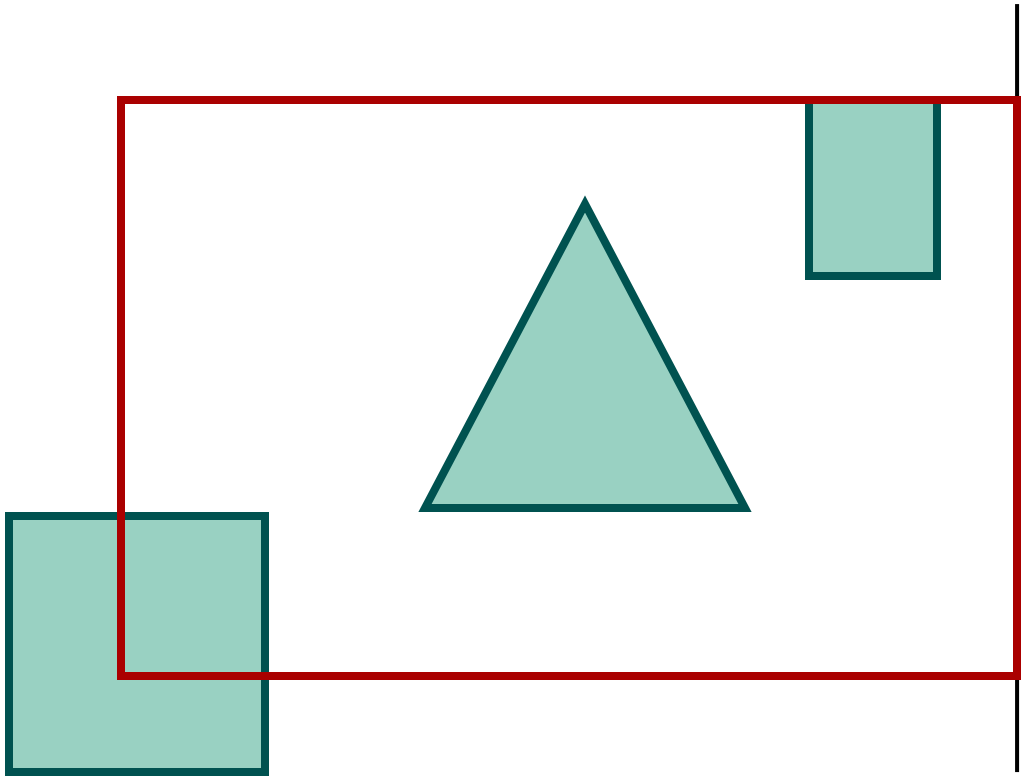
Sutherland-Hodgeman Polygon Clipping

- **Clip to Each Window Boundary One at a Time**



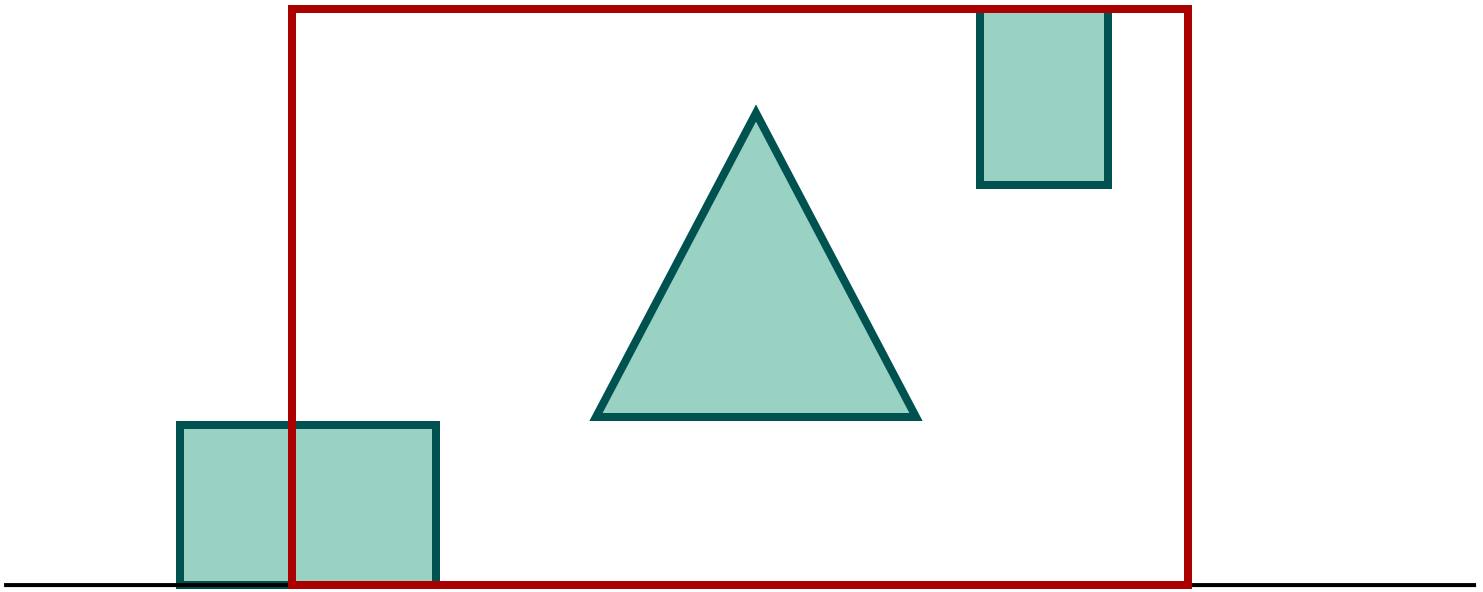
Sutherland-Hodgeman Polygon Clipping

- Clip to Each Window Boundary One at a Time



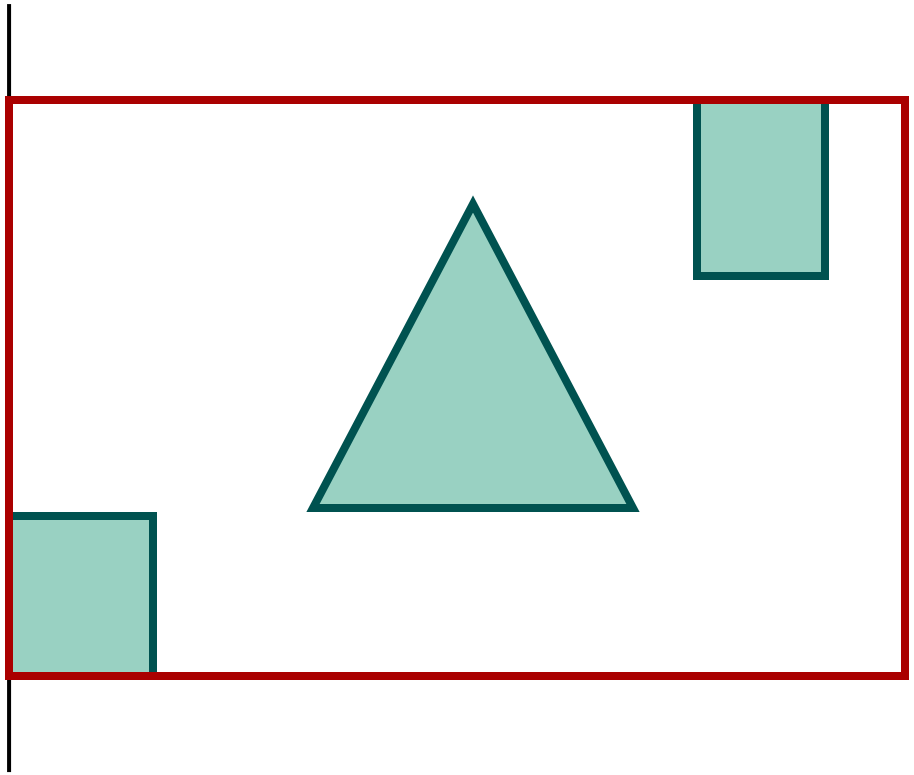
Sutherland-Hodgeman Polygon Clipping

- **Clip to Each Window Boundary One at a Time**



Sutherland-Hodgeman Polygon Clipping

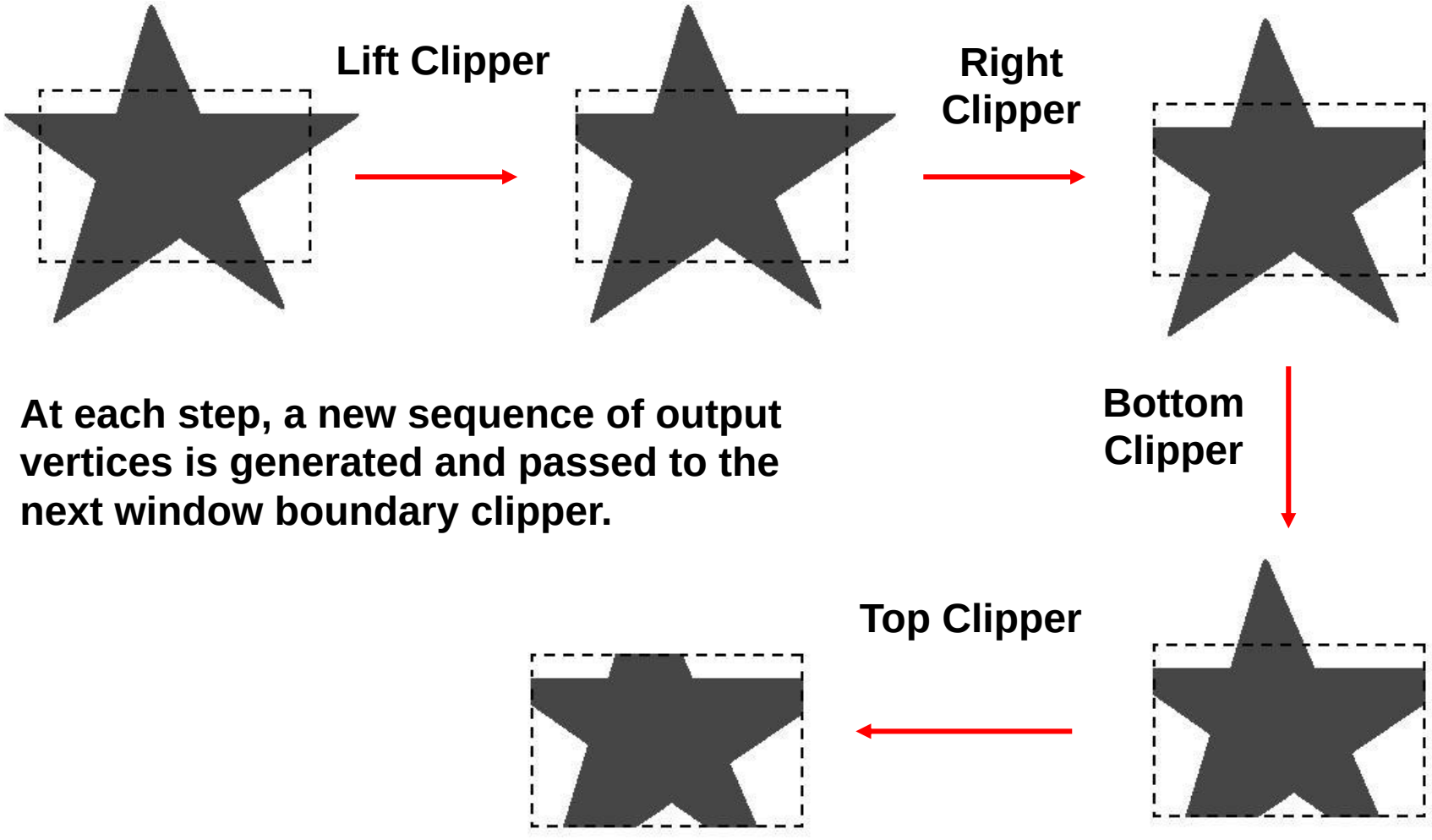
- **Clip to Each Window Boundary One at a Time**



Sutherland-Hodgman Polygon Clipping

- Clip a polygon by processing the polygon boundary as a whole against each window edge.
- Processing all polygon vertices against each clip rectangle boundary in turn.
- Beginning with the initial set of polygon vertices, we could first clip the polygon against the left rectangle boundary to produce a new sequence of vertices.
- The new set of vertices could be successively passed to a right boundary clipper, a bottom boundary clipper, and a top boundary clipper, a right boundary clipper.

Sutherland-Hodgman Polygon Clipping



At each step, a new sequence of output vertices is generated and passed to the next window boundary clipper.

Sutherland-Hodgman Polygon Clipping

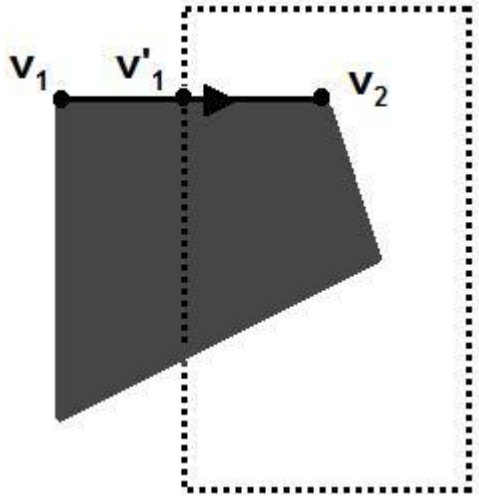
There are four possible cases when processing vertices in sequence around the perimeter of a polygon.

As each pair of adjacent polygon vertices is passed to a next window boundary clipper, we make the following tests:

Sutherland-Hodgman Polygon Clipping

1. If the first vertex is outside the window boundary and the second vertex is inside

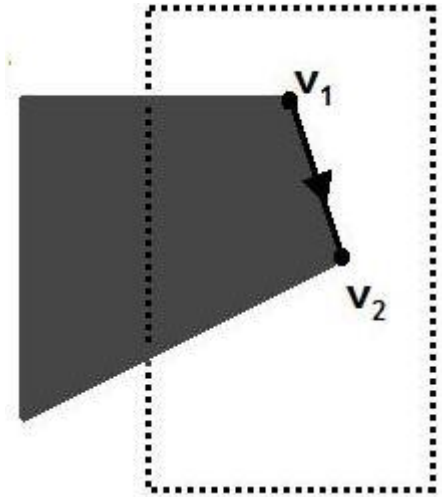
Then , both the intersection point of the polygon edge with the window boundary and the second vertex are added to the output vertex list.



Sutherland-Hodgman Polygon Clipping

2. If both input vertices are inside the window boundary.

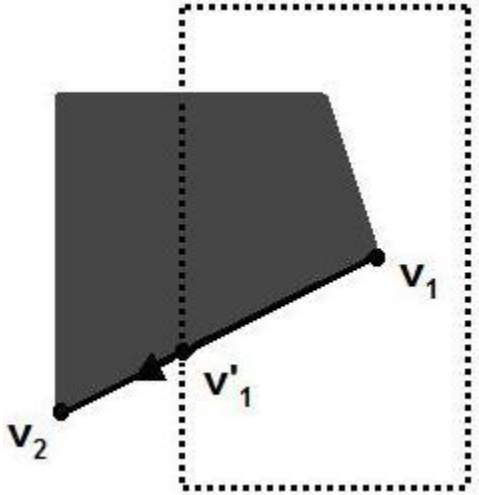
Then, only the second vertex is added to the output vertex list.



Sutherland-Hodgman Polygon Clipping

3. If the first vertex is inside the window boundary and the second vertex is outside.

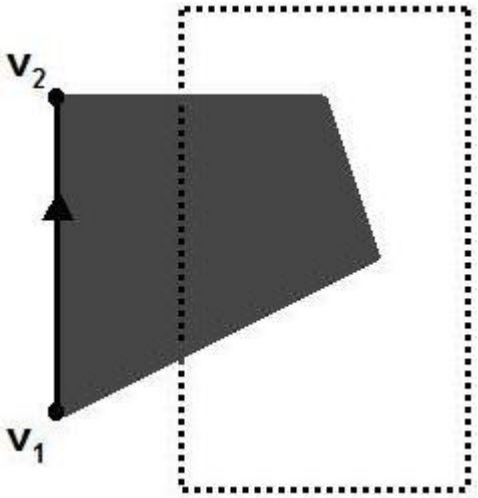
Then, only the edge intersection with the window boundary is added to the output vertex list.



Sutherland-Hodgman Polygon Clipping

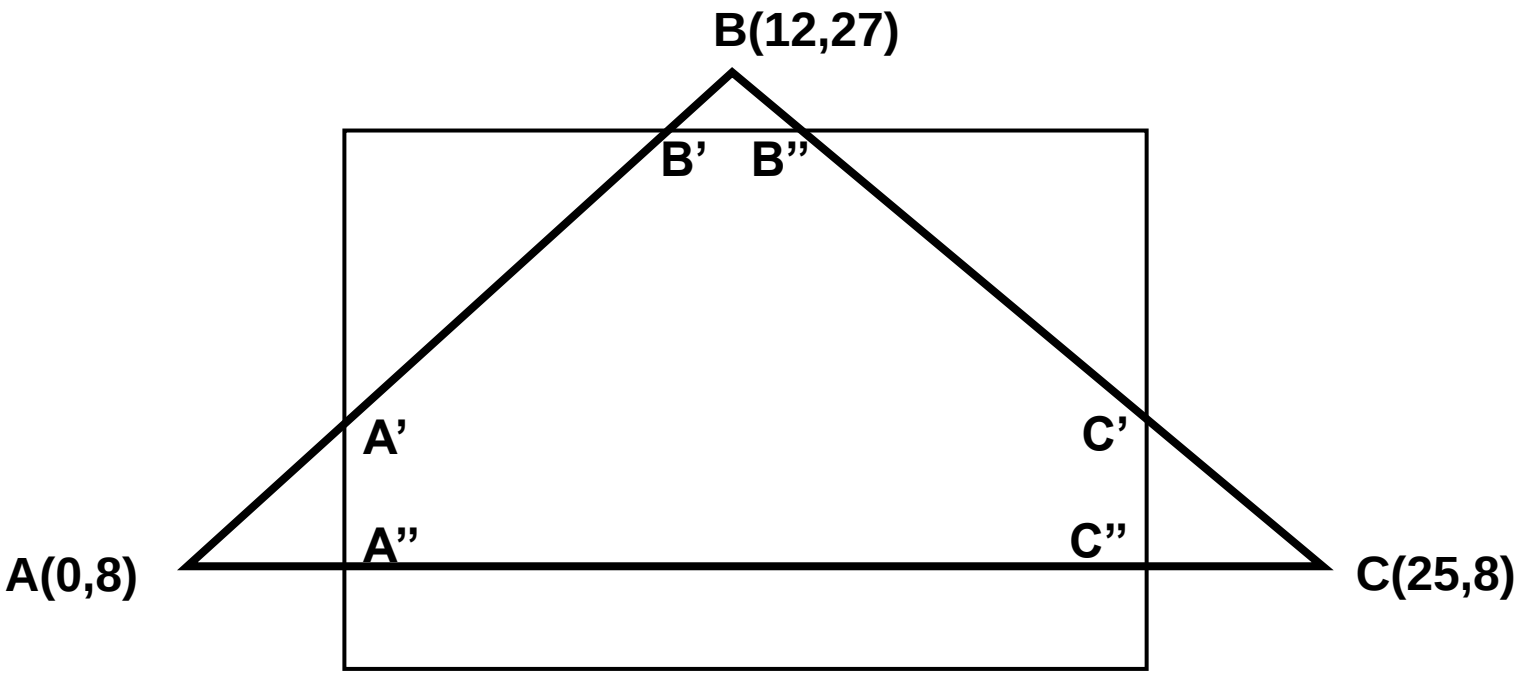
4. If both input vertices are outside the window boundary.

Then, nothing is added to the output vertex list.



Sutherland-Hodgman Polygon Clipping

Example:- **Apply Polygon clipping for following Triangle with Xmin=5, Xmax=20 , Ymin=5 , Ymax=25**



Sutherland-Hodgman Polygon Clipping

Solution:-

First Step:- we want to check the points if its in or out

A=1000 its out

B= 0001 its out

C=0100 its out

Second Step:- We should find intersection points by using line clipping.

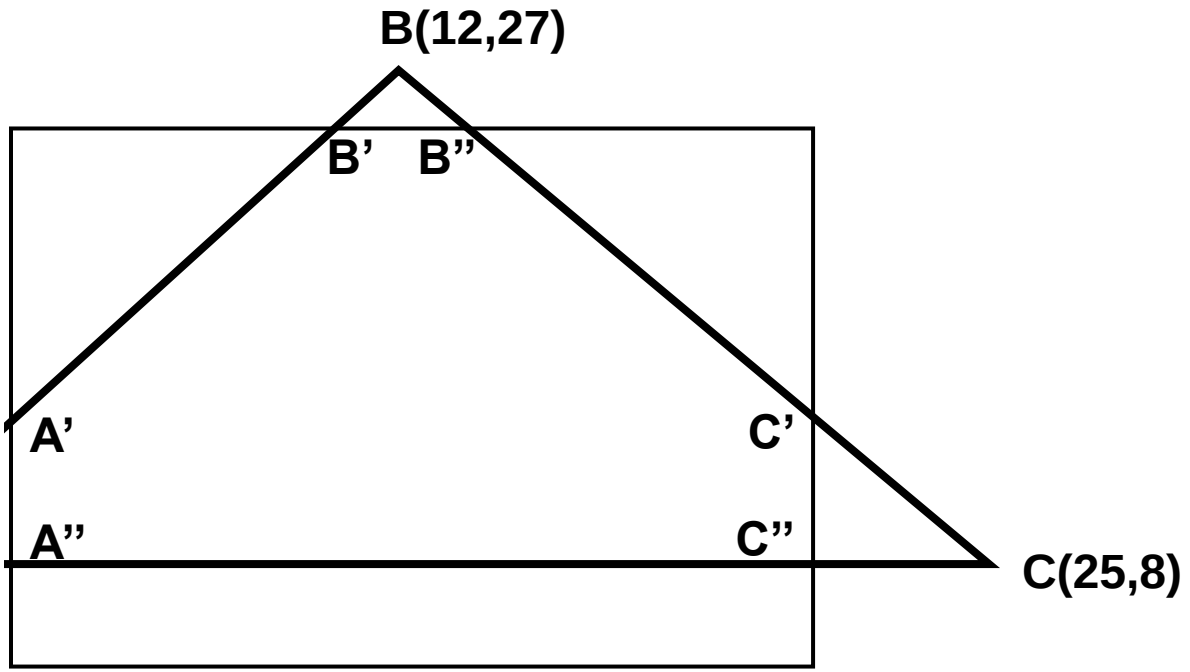
1. For line A ⟶ B we have two intersection A’ and B’
2. For line B ⟶ C we have two intersection B’’ and C’
3. For Line C ⟶ A we have two intersection C’’ and A’’

Sutherland-Hodgman Polygon Clipping

Solution:-

Third Step:- Then we will apply Sutherland-Hodgman algorithm (Left, Right, Bottom, Top sequence) to get the clipped polygon.

1. For Left the output is [A’,B],[C],[A’’]

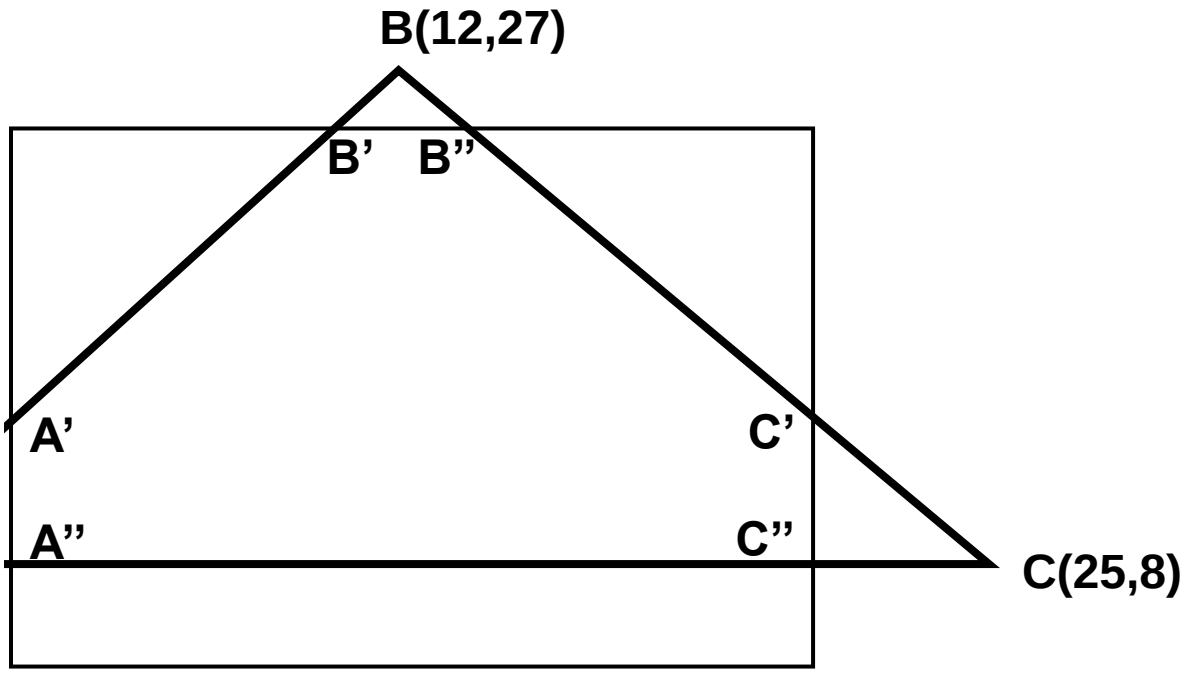


Sutherland-Hodgman Polygon Clipping

Solution:-

Third Step:- Then we will apply Sutherland-Hodgman algorithm (Left, Right, Bottom, Top sequence) to get the clipped polygon.

2. For Right the output is [B],[C'],[C'',A'']

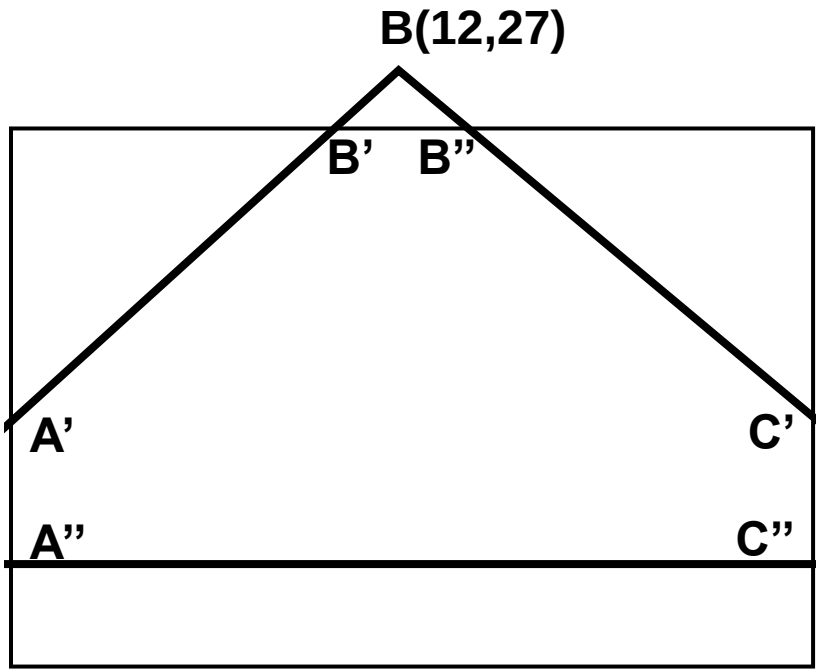


Sutherland-Hodgman Polygon Clipping

Solution:-

Third Step:- Then we will apply Sutherland-Hodgman algorithm (Left, Right, Bottom, Top sequence) to get the clipped polygon.

3. For Bottom the output is [B],[C'],[C''],[A''],[A']

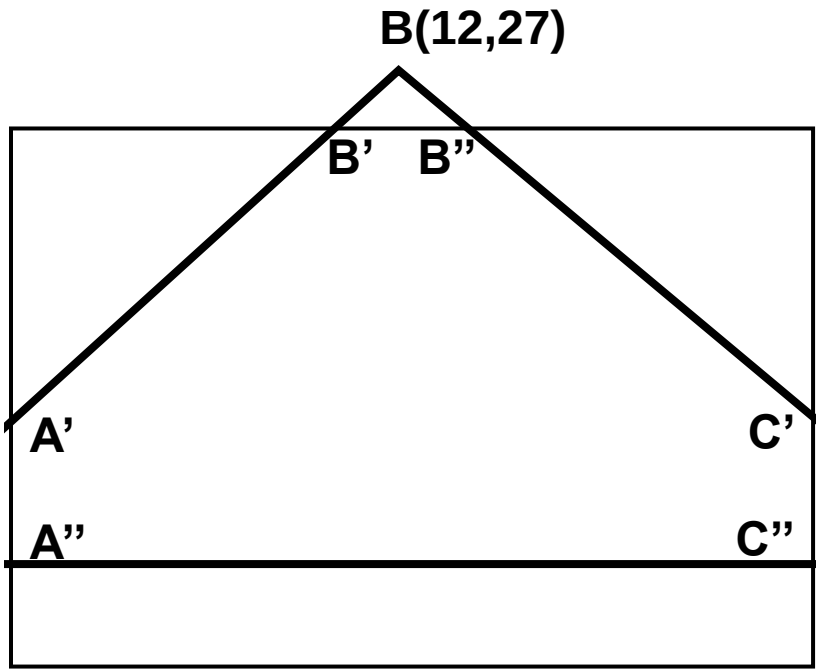


Sutherland-Hodgman Polygon Clipping

Solution:-

Third Step:- Then we will apply Sutherland-Hodgman algorithm (Left, Right, Bottom, Top sequence) to get the clipped polygon.

4. For Top the output is [B’],[B’’,C’],[C’’],[A’’],[A’]

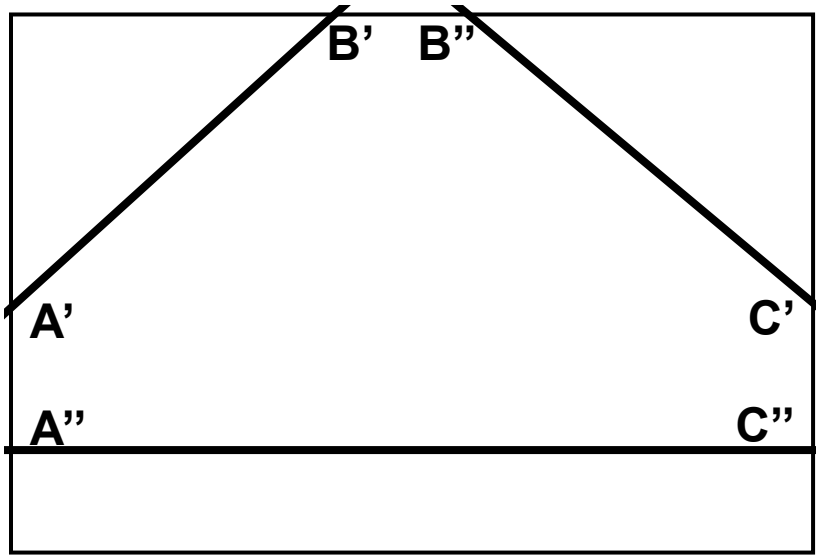


Sutherland-Hodgman Polygon Clipping

Solution:-

Third Step:- Then we will apply Sutherland-Hodgman algorithm (Left, Right, Bottom, Top sequence) to get the clipped polygon.

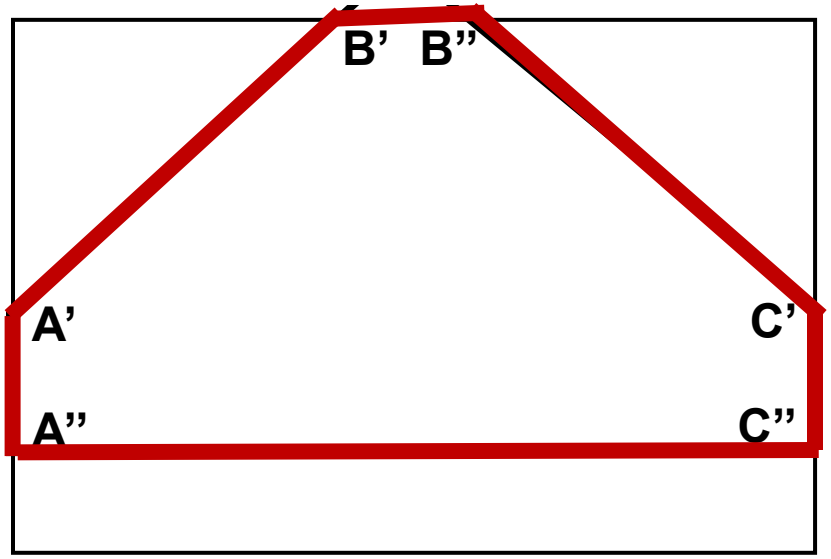
Final Points [B’],[B’’,C’],[C’’],[A’’],[A’]



Sutherland-Hodgman Polygon Clipping

Solution:-

Final Polygon



Sutherland-Hodgman Polygon Clipping

Example:-

We illustrate this algorithm by processing the area in figure against the left window boundary.

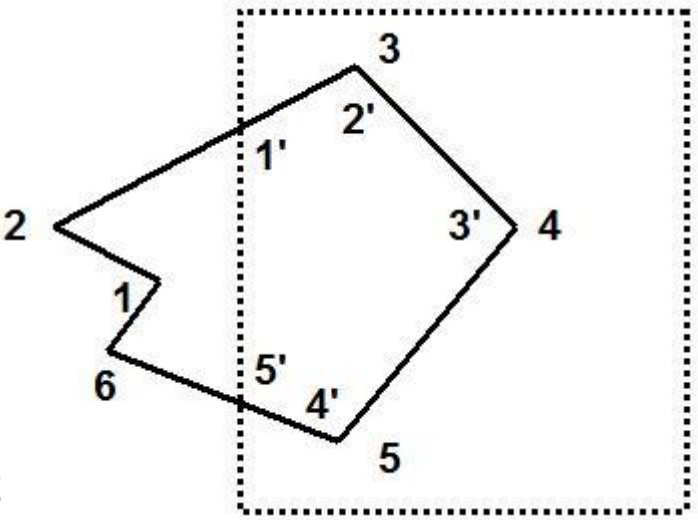
Vertices 1 and 2 are outside of the boundary.

Vertex 3, which is inside, 1' and vertex 3 are saved.

Vertex 4 and 5 are inside, and they also saved.

Vertex 6 is outside, 5' is saved.

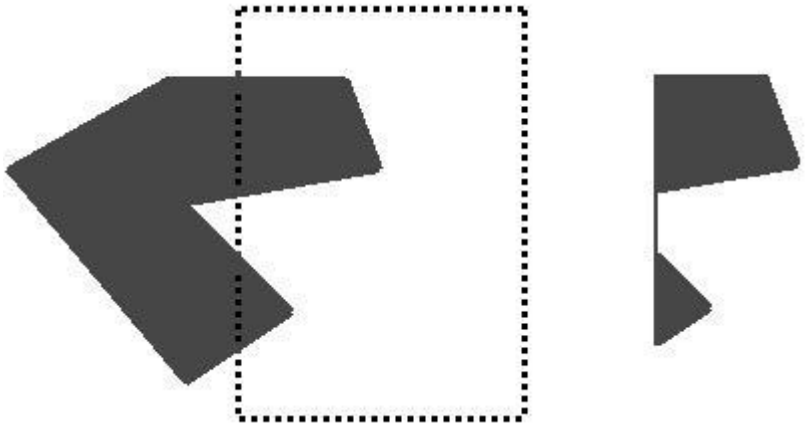
Using the five saved points, we would repeat the process for the next window boundary.



Sutherland-Hodgman Polygon Clipping

The Sutherland-Hodgman algorithm correctly clips convex polygons, but concave polygons may be displayed with extraneous lines as demonstrated in figure.

Since there is only one output vertex list, the last vertex in the list is always joined to the first vertex.



Weiler-Atherton Polygon Clipping

This algorithm was developed for identifying visible surfaces, and can be used to clip a fill area that is either a convex polygon or a concave polygon.

The basic idea of this algorithm is that instead of proceeding around the polygon edges as vertices are processed, we will follow the window boundaries.

The path we follow depends on:

- **polygon-processing direction (clockwise or counterclockwise)**
- **The pair of polygon vertices outside-to-inside or an inside-to-outside.**

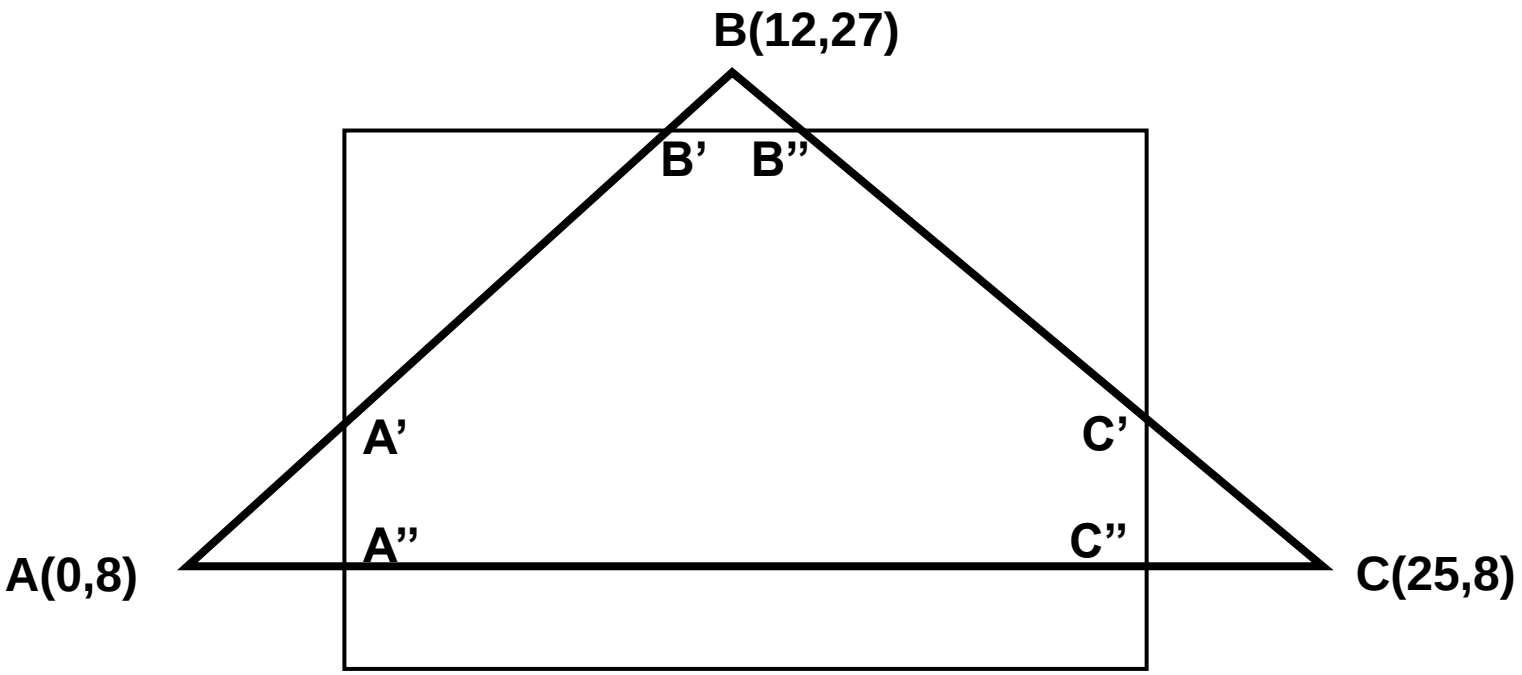
Weiler-Atherton Polygon Clipping

- **For clockwise processing of polygon vertices, we use the following rules:**

- **For an outside-to-inside pair of vertices, follow polygon boundaries.**
- **For an inside-to-outside pair of vertices, follow window boundaries in a clockwise direction.**

Weiler-Atherton Polygon Clipping

Example:- **Apply Polygon clipping for following Triangle with Xmin=5, Xmax=20 , Ymin=5 , Ymax=25**



Weiler-Atherton Polygon Clipping

Solution:-

First Step:- we want to check the points if its in or out

A=1000 its out

B= 0001 its out

C=0100 its out

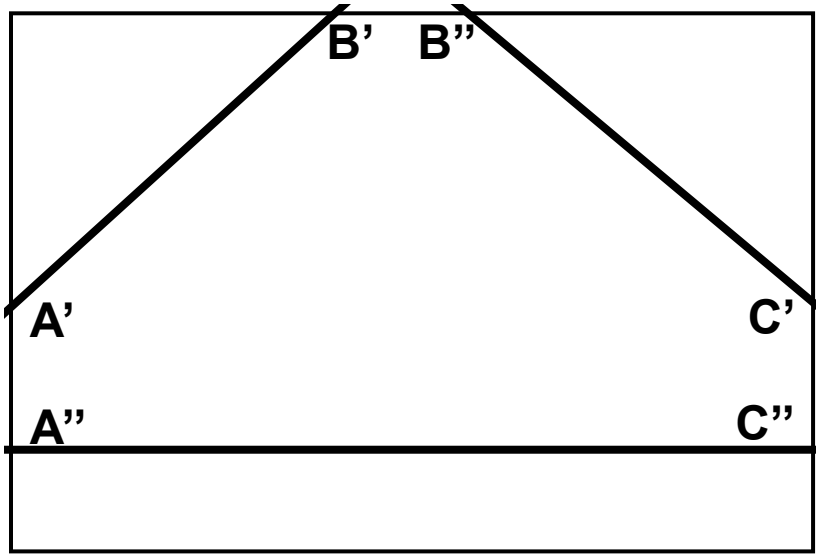
Second Step:- We should find intersection points by using line clipping.

1. For line A ⟶ B we have two intersection A’ and B’
2. For line B ⟶ C we have two intersection B’’ and C’
3. For Line C ⟶ A we have two intersection C’’ and A’’

Weiler-Atherton Polygon Clipping

Solution:-

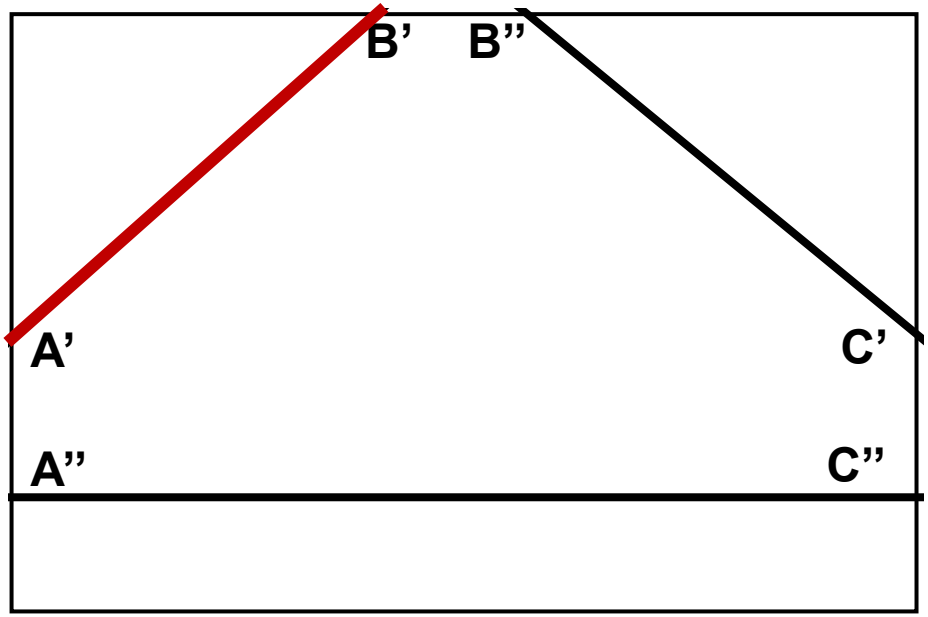
Third Step:- Then we will apply **Weiler-Atherton Polygon Clipping** to get the clipped polygon.



Weiler-Atherton Polygon Clipping

Solution:-

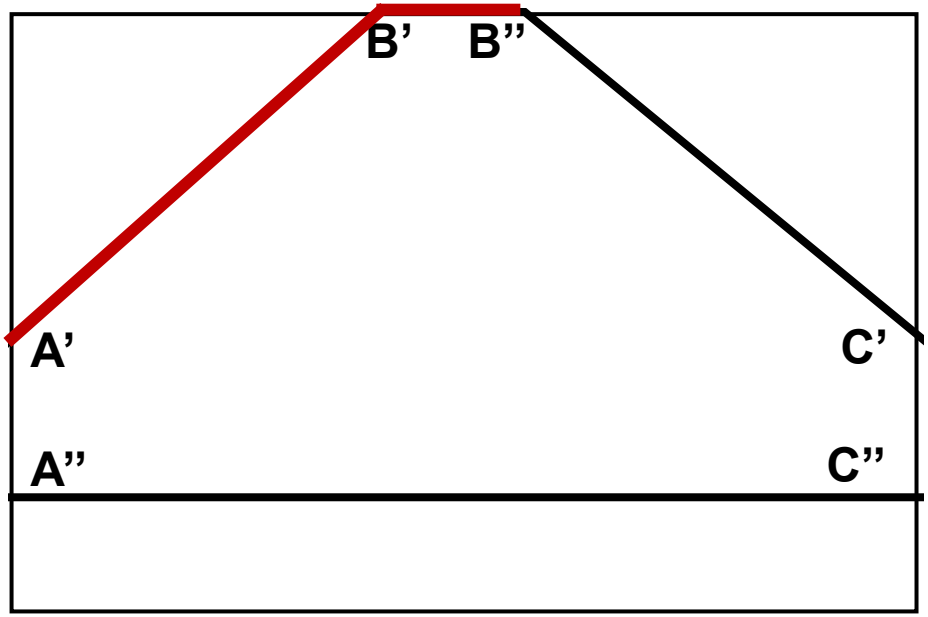
1. **A** OUT to **B'** IN Follow Polygon



Weiler-Atherton Polygon Clipping

Solution:-

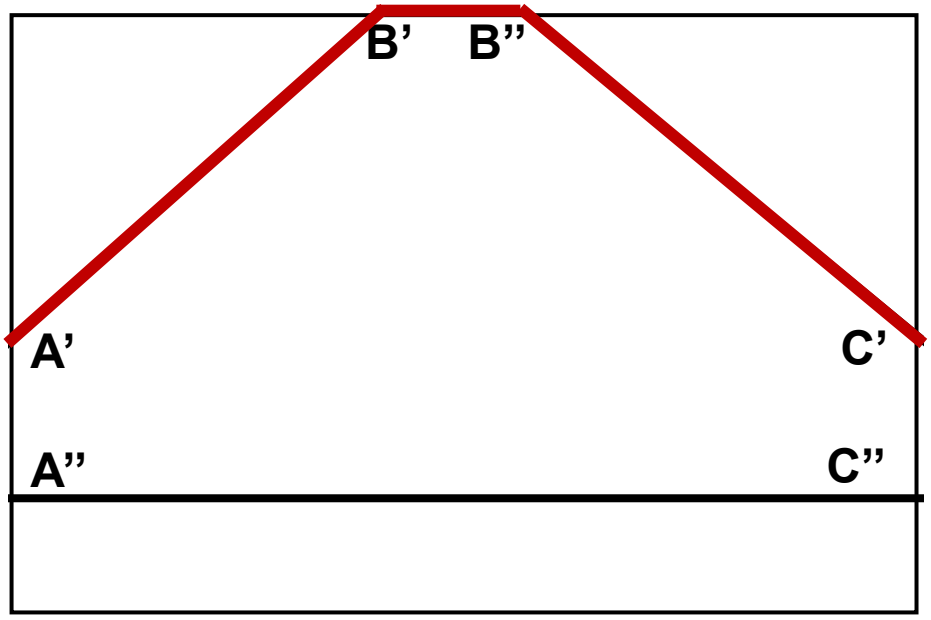
2. **A’** IN to **B** OUT Follow window boundaries



Weiler-Atherton Polygon Clipping

Solution:-

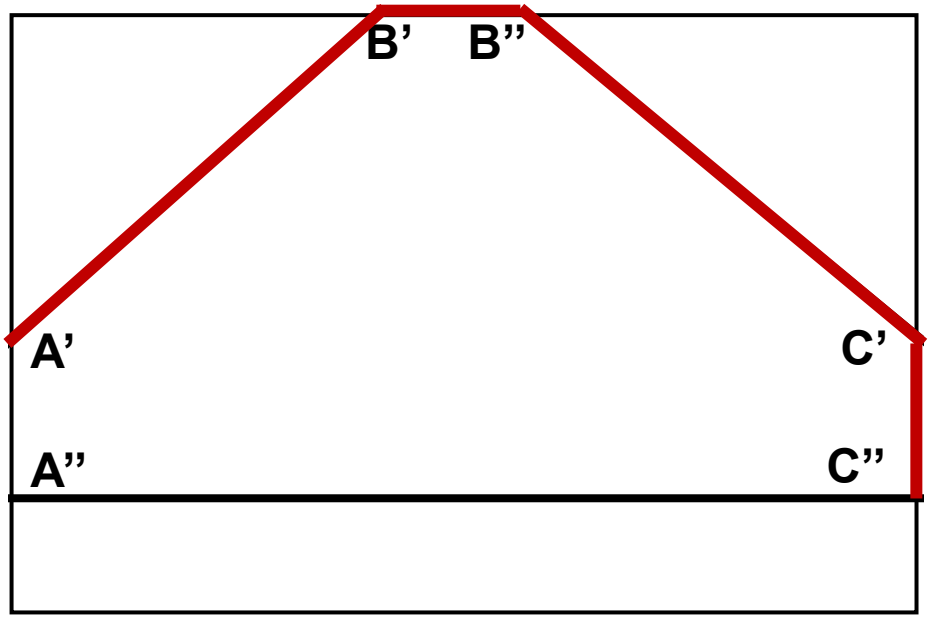
3. **B** OUT to **C'** IN Follow Polygon



Weiler-Atherton Polygon Clipping

Solution:-

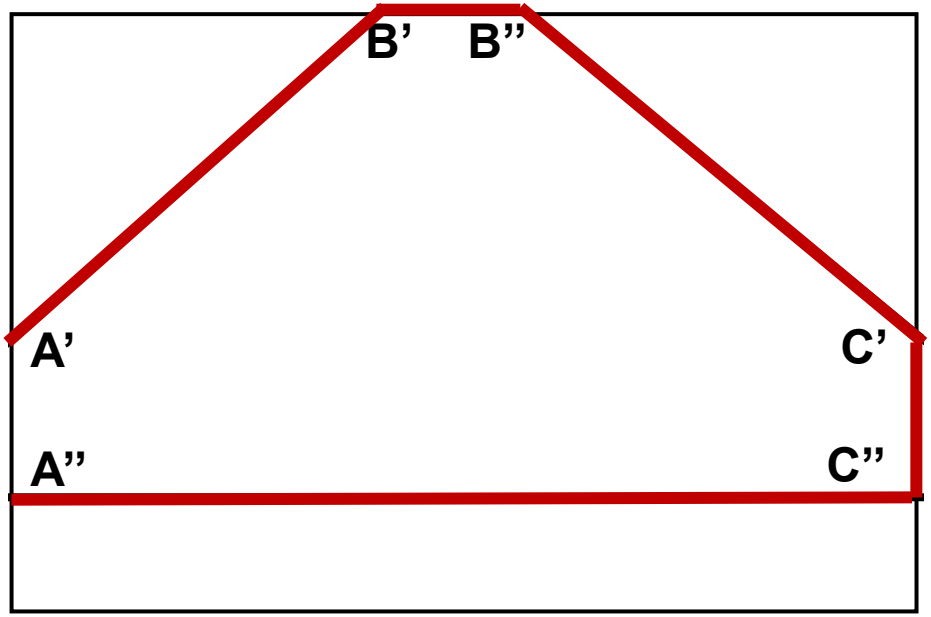
4. B” IN to **C** OUT Follow window boundaries



Weiler-Atherton Polygon Clipping

Solution:-

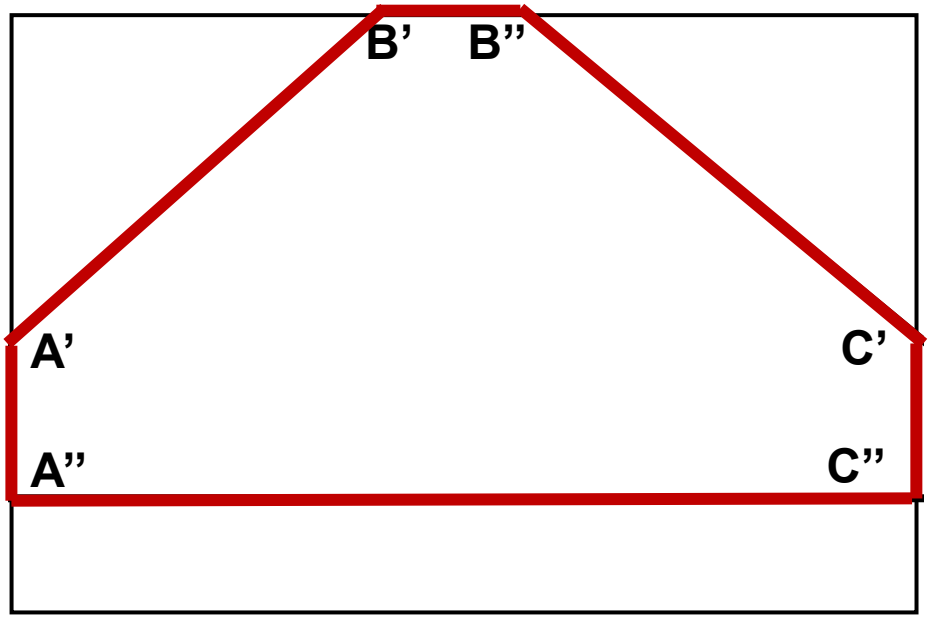
5. C OUT to **A”** IN Follow Polygon



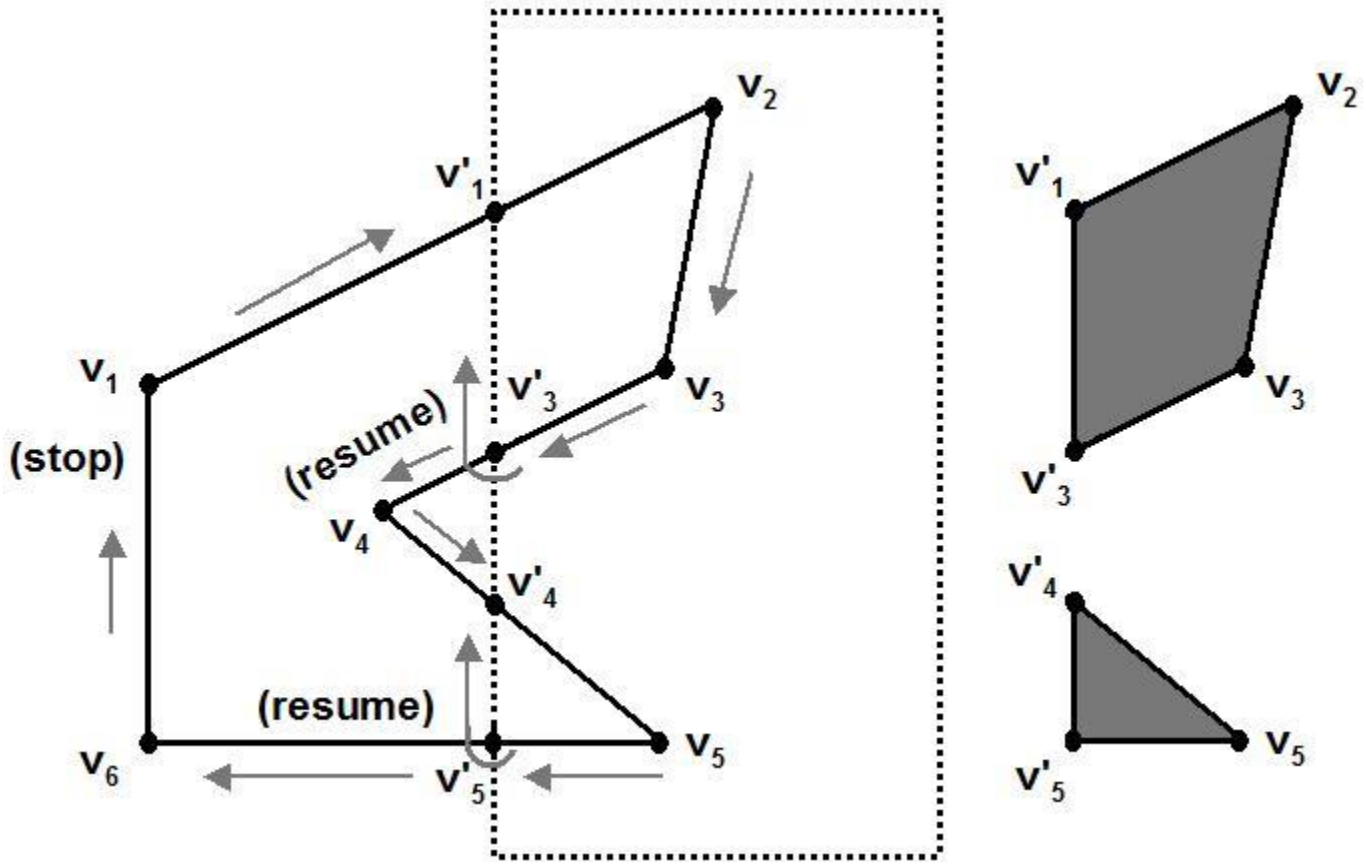
Weiler-Atherton Polygon Clipping

Solution:-

6. **C”** IN to **A** OUT Follow window boundaries



Weiler-Atherton Polygon Clipping



Text Clipping

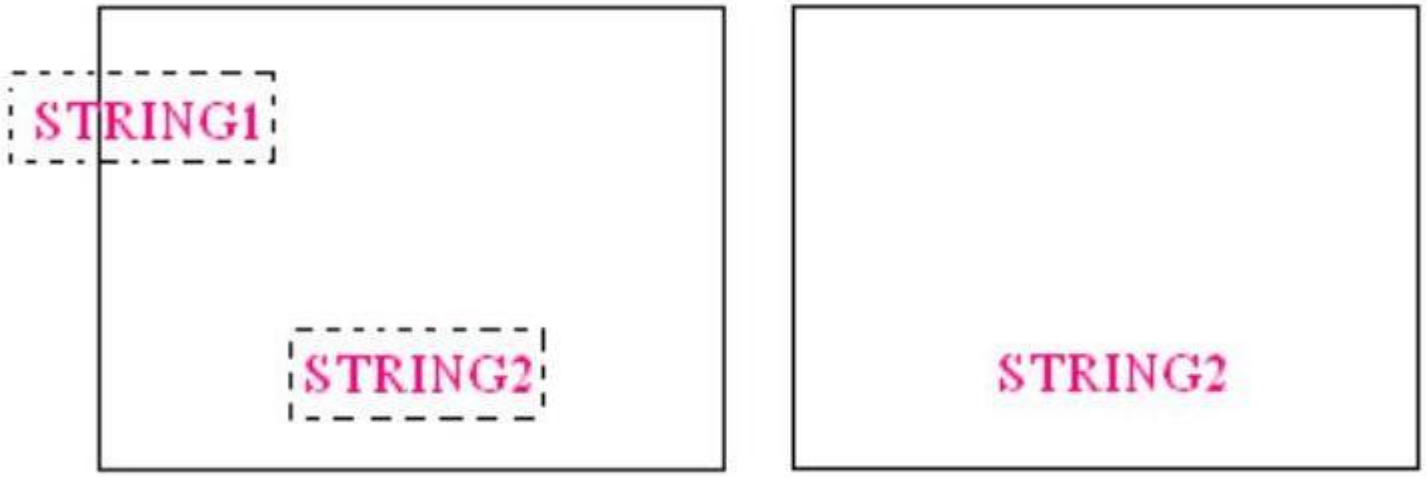
There are several techniques that can be used to provide text clipping in a graphics packages.

The choice of clipping method depends on how characters are generated and what requirements we have for displaying character strings.

Text Clipping

All-or-none string-clipping

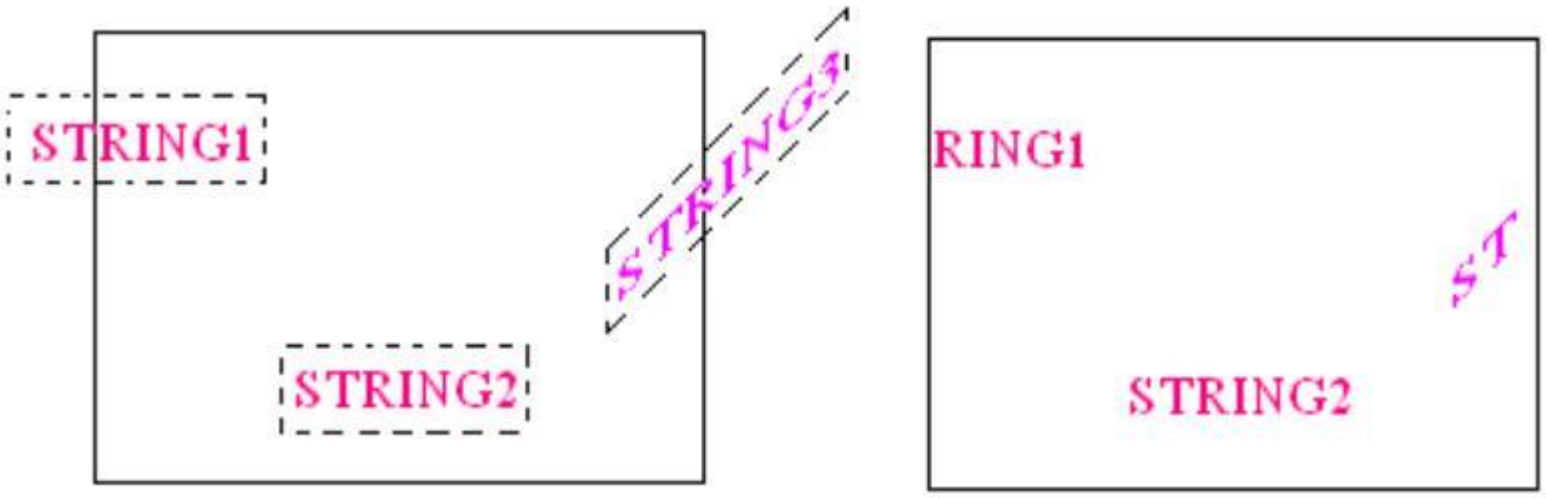
- If all of the string is inside a clip window, we keep it.
- Otherwise the string is discarded.



Text Clipping

All-or-none character-clipping

Here we discard only those characters that are not completely inside the window

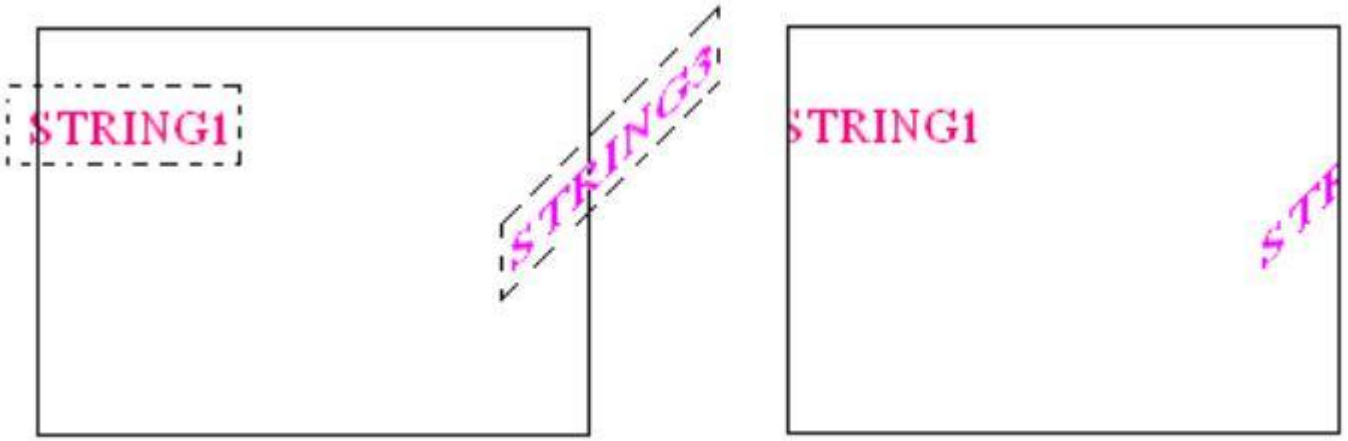


Text Clipping

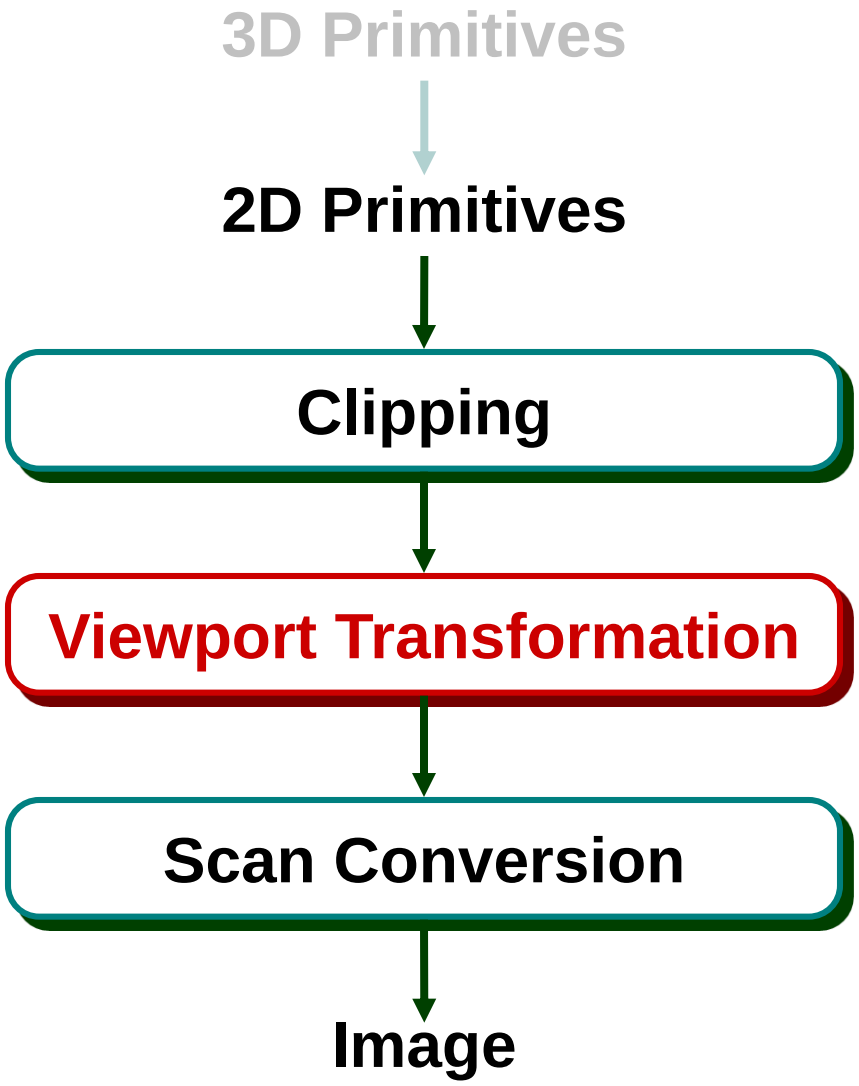
Clip the components of individual characters

We treat characters in much the same way that we treated lines.

If an individual character overlaps a clip window boundary, we clip off the parts of the character that are outside the window



2D Rendering Pipeline



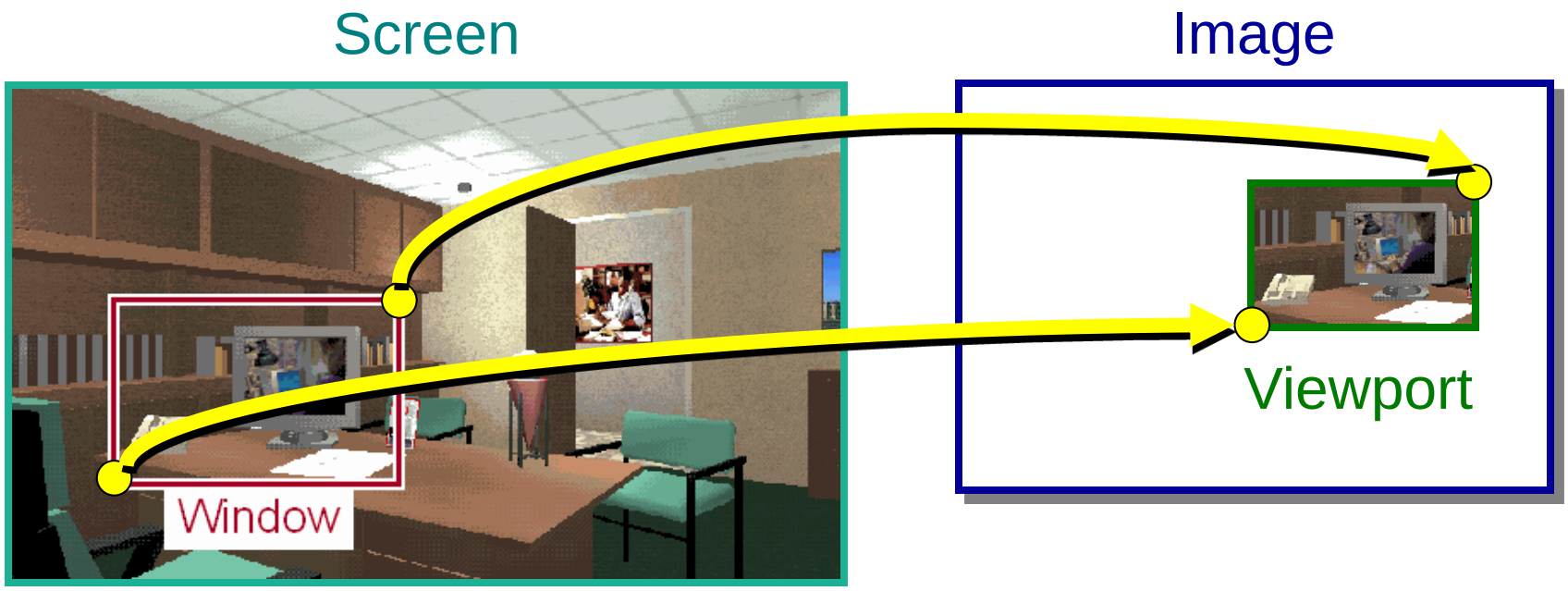
Clip portions of geometric primitives residing outside window

Transform the clipped primitives from screen to image coordinates

Fill pixel representing primitives in screen coordinates

Viewport Transformation

- **Transform 2D Geometric Primitives from Screen Coordinate System (Projection Coordinates) to Image Coordinate System (Device Coordinates)**

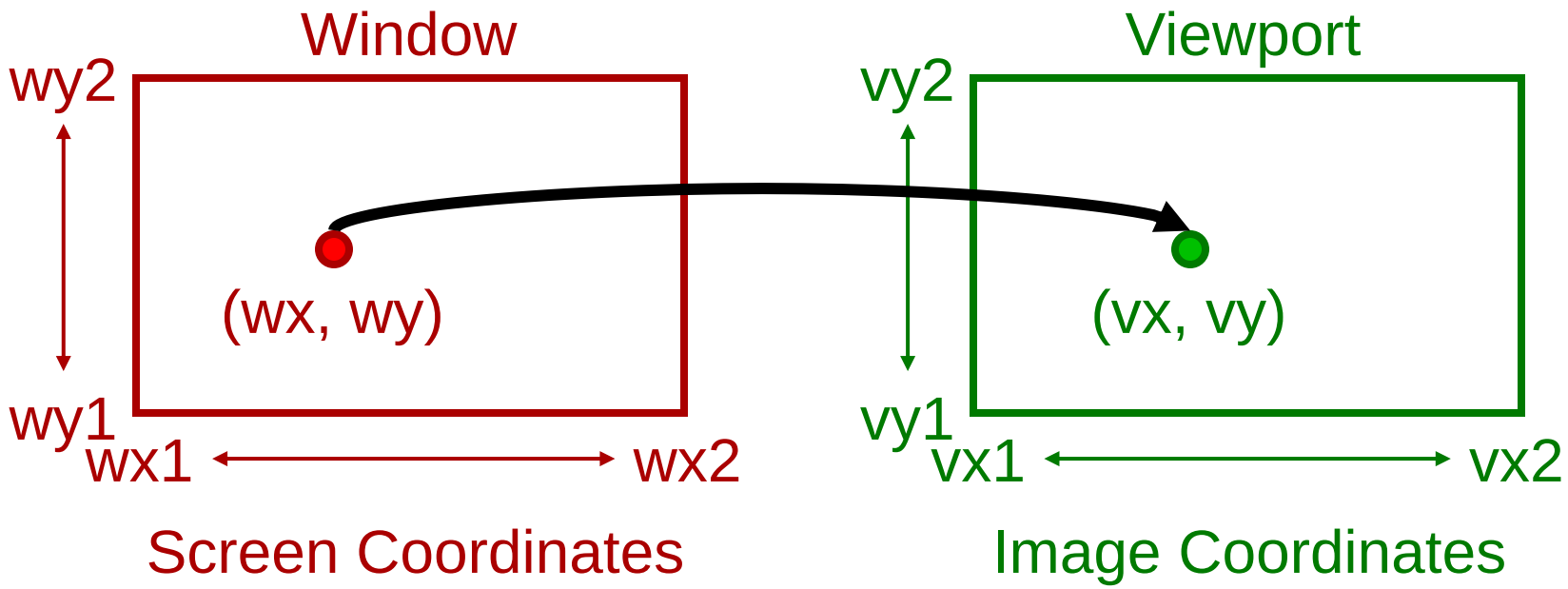


Window *vs.* Viewport

- **Window**
  - World-coordinate area selected for display
  - What is to be viewed
- **Viewport**
  - Area on the display device to which a window is mapped
  - Where it is to be displayed

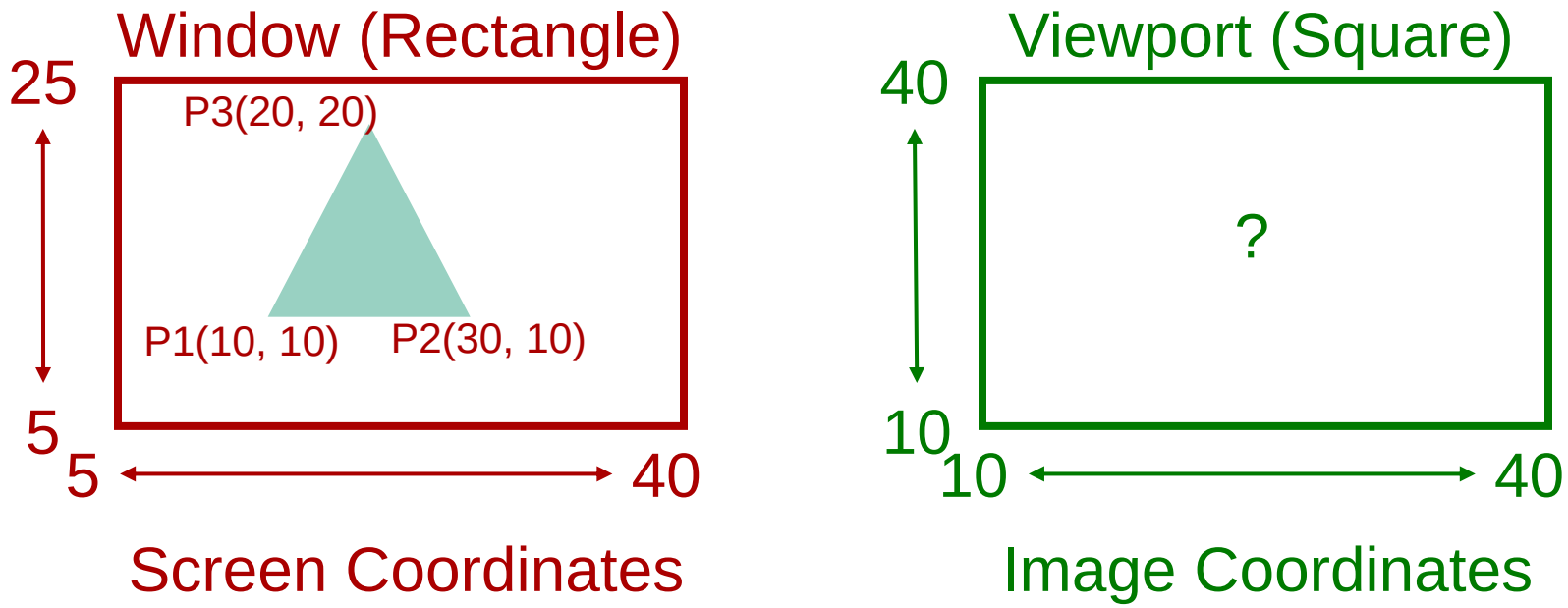
Viewport Transformation

- **Window-to-Viewport Mapping**



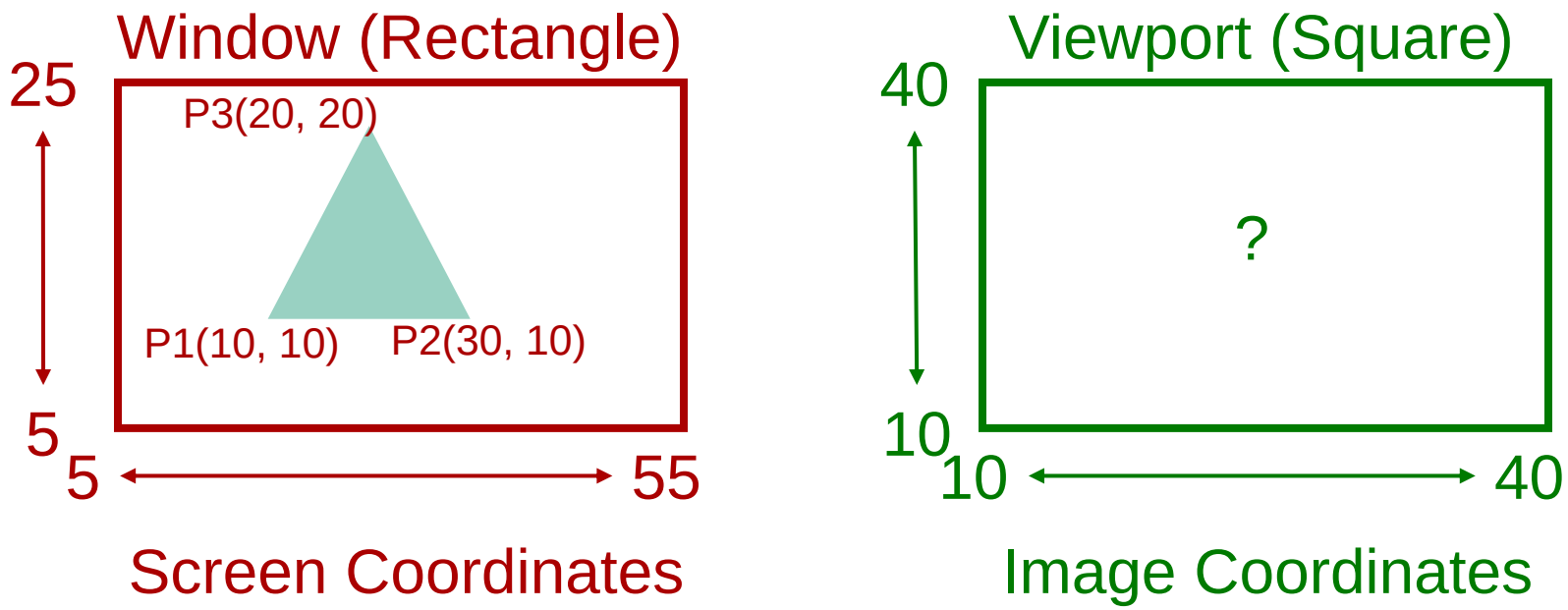

```
vx = vx1 + (wx – wx1) * (vx2 – vx1) / (wx2 – wx1);
vy = vy1 + (wy – wy1) * (vy2 – vy1) / (wy2 – wy1);
```

Viewport Transformation




```
vx = vx1 + (wx - wx1) * (vx2 - vx1) / (wx2 - wx1);
vy = vy1 + (wy - wy1) * (vy2 - vy1) / (wy2 - wy1);
```

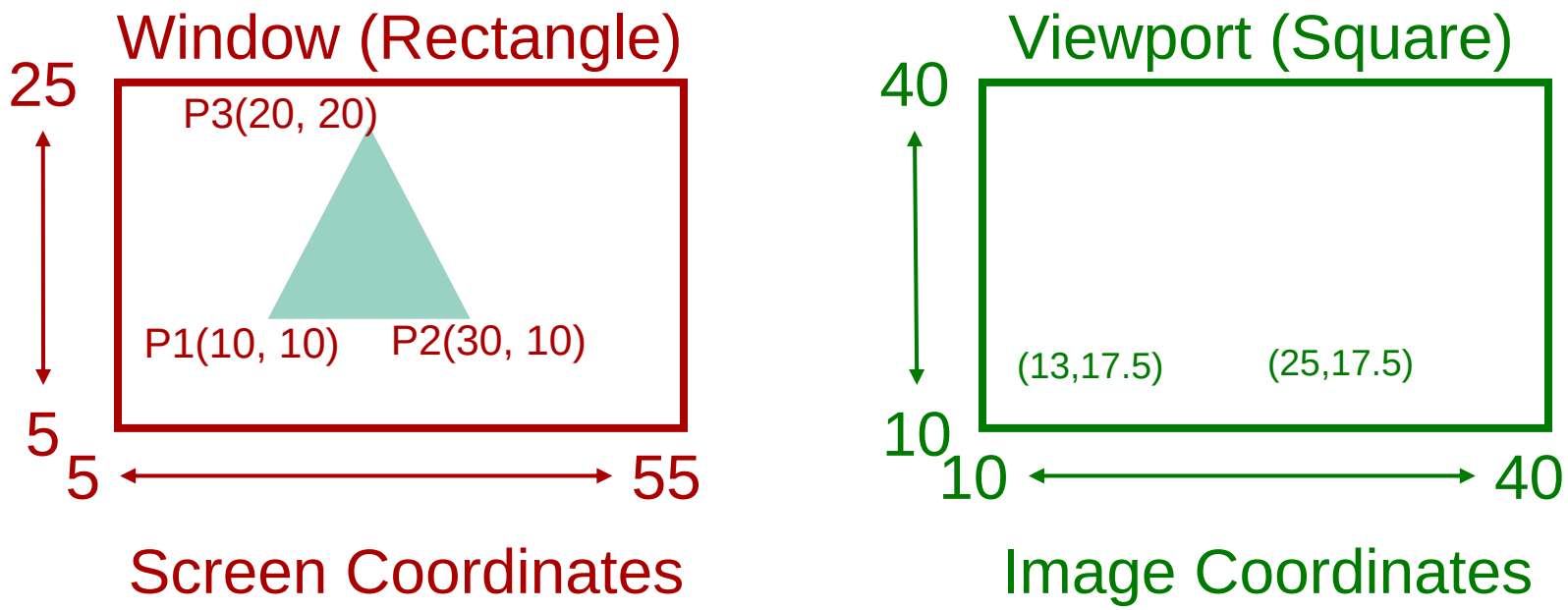
Viewport Transformation



$$vx_{p1} = 10 + (10 - 5) * (40 - 10) / (55 - 5) = 13;$$

$$vy_{p1} = 10 + (10 - 5) * (40 - 10) / (25 - 5) = 17.5;$$

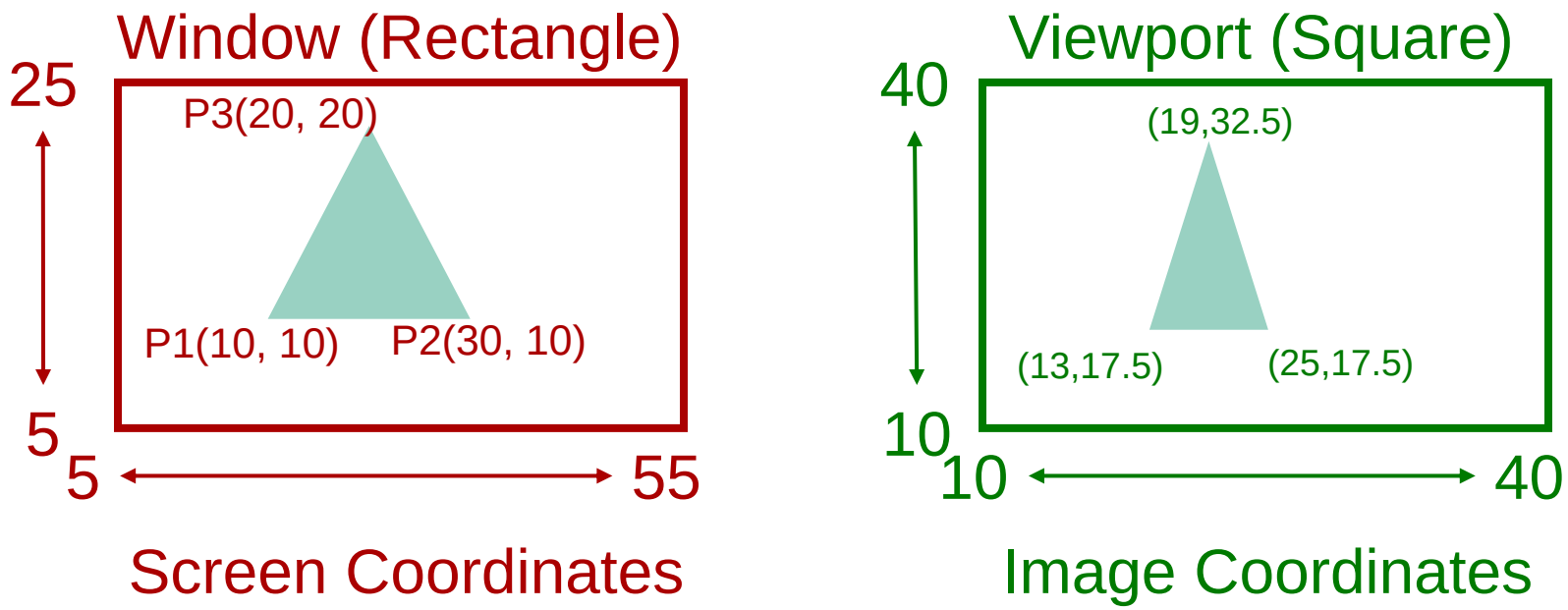
Viewport Transformation



$$vx_{p2} = 10 + (30 - 5) * (40 - 10) / (55 - 5) = 25;$$

$$vy_{p2} = 10 + (10 - 5) * (40 - 10) / (25 - 5) = 17.5;$$

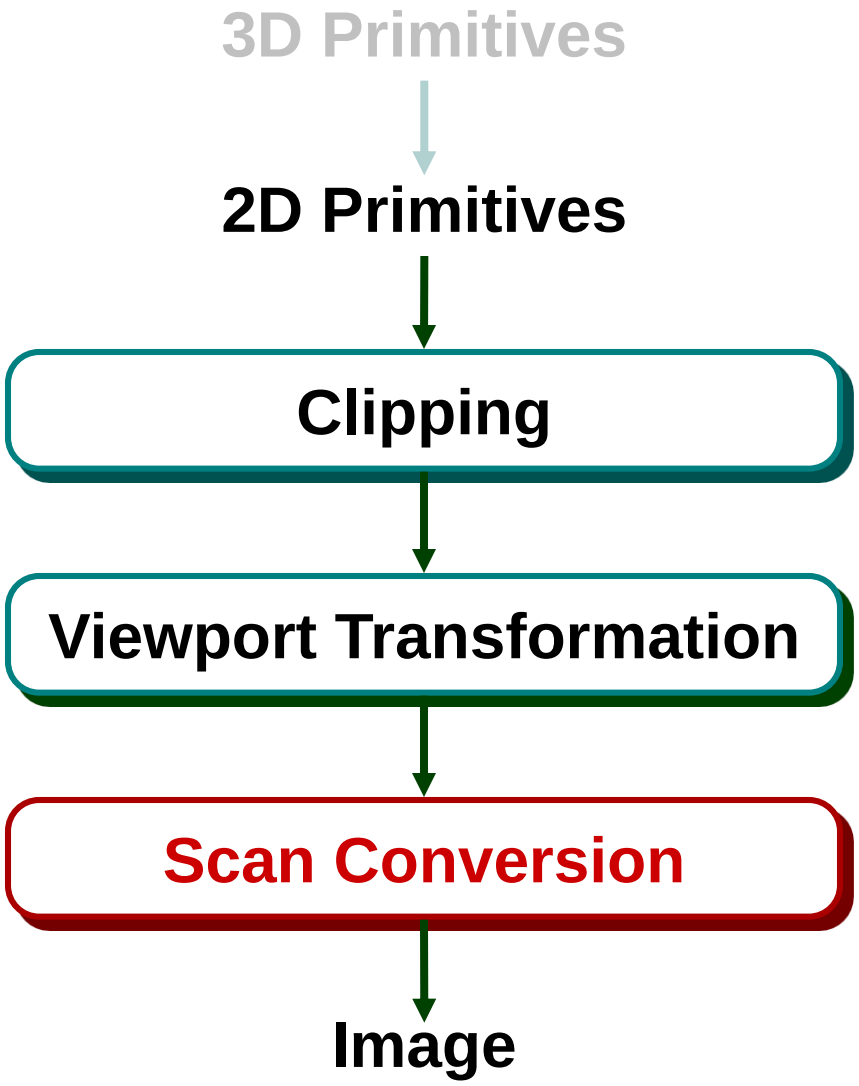
Viewport Transformation



$$vx_{p3} = 10 + (20 - 5) * (40 - 10) / (55 - 5) = 19;$$

$$vy_{p3} = 10 + (20 - 5) * (40 - 10) / (25 - 5) = 32.5;$$

2D Rendering Pipeline



Clip portions of geometric primitives residing outside window

Transform the clipped primitives from screen to image coordinates

Fill pixel representing primitives in screen coordinates

Summary of Transformation