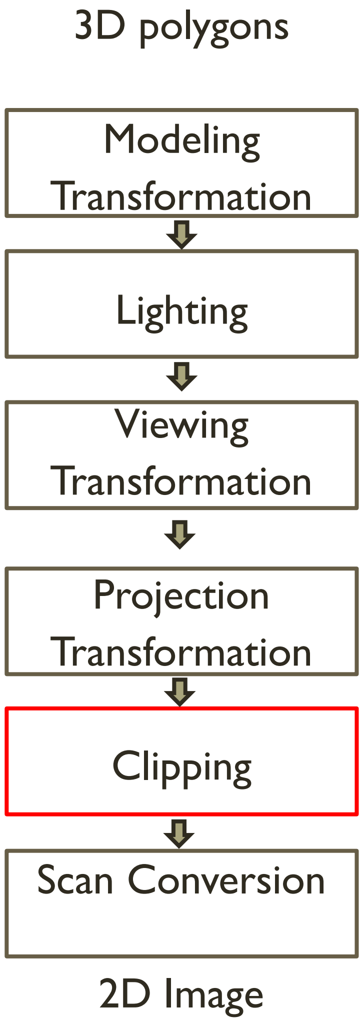
Fundamentals of Computer Graphics and Image Processing
Visibility, Culling, Clipping (05)

doc. RNDr. Martin Madaras, PhD.
martin.madaras@fmph.uniba.sk

Overview

- **Clipping**
  - Point Clipping
  - Line Clipping
  - Polygon Clipping
- Hidden Surface Removal

3D rendering pipeline



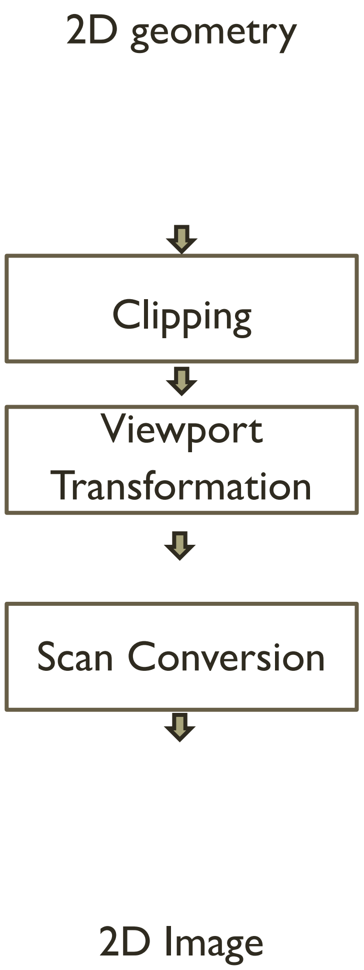
Rasterization (05)

Clip polygons outside of camera's view

How the lectures should look like #1

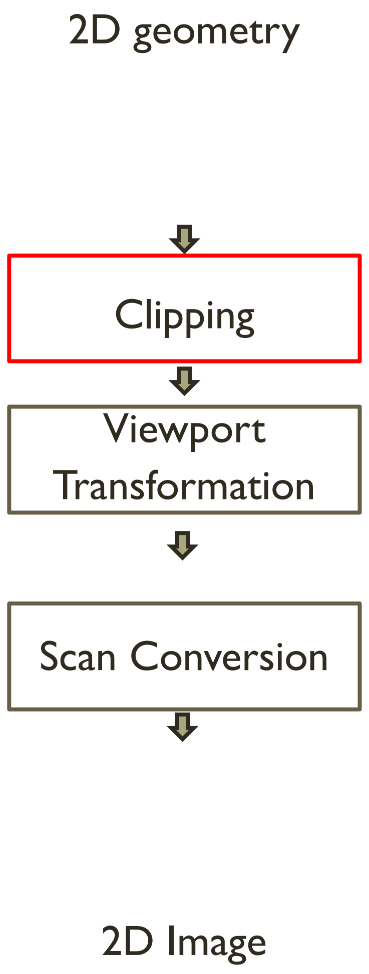
- Ask questions, please!!!
- Be communicative
- More active you are, the better for you!

2D rendering pipeline



- Clip and remove geometry outside of the window
- Transform from screen coordinates to image coordinates
- Fill pixels on the screen

2D rendering pipeline



- Clip and remove geometry outside of the window
- Transform from screen coordinates to image coordinates
- Fill pixels on the screen

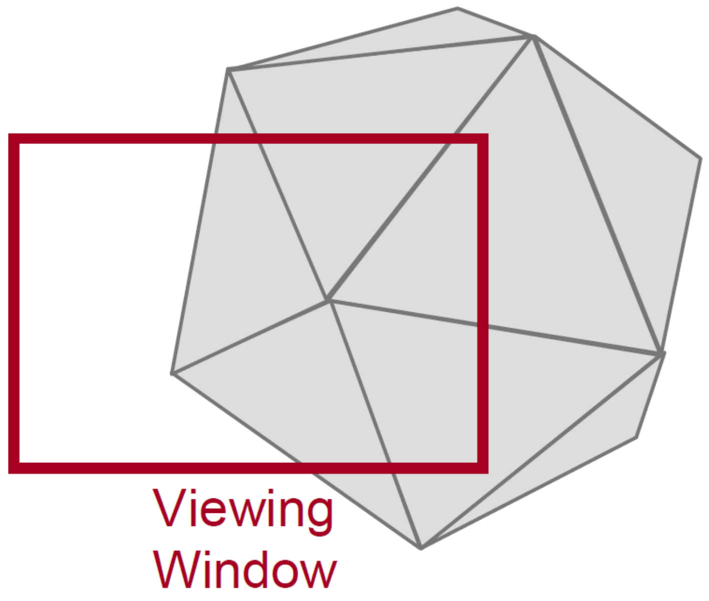
Clipping

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

Screen Coordinates

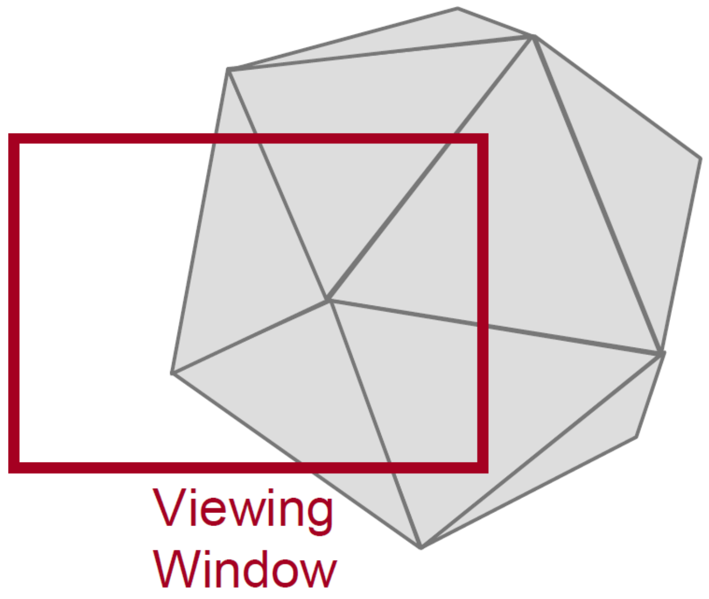
Clipping

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



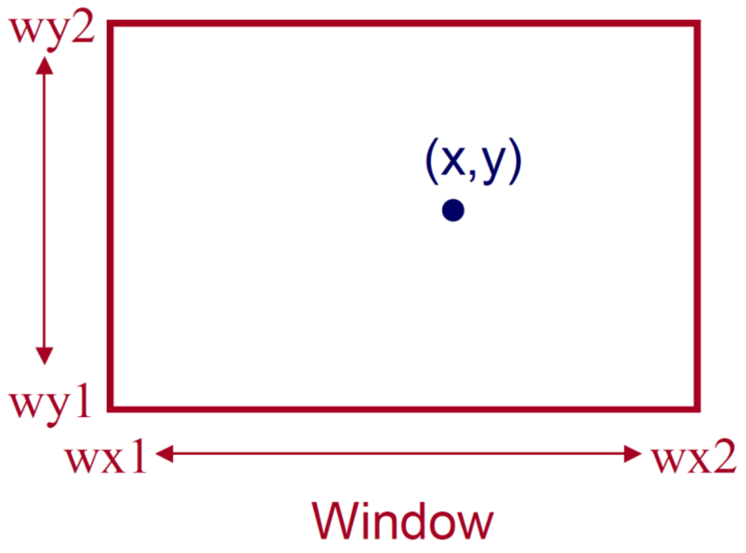
Clipping

- Avoid drawing parts of primitives outside window
  - Points, Lines, Polygons, Circles etc.



Point Clipping

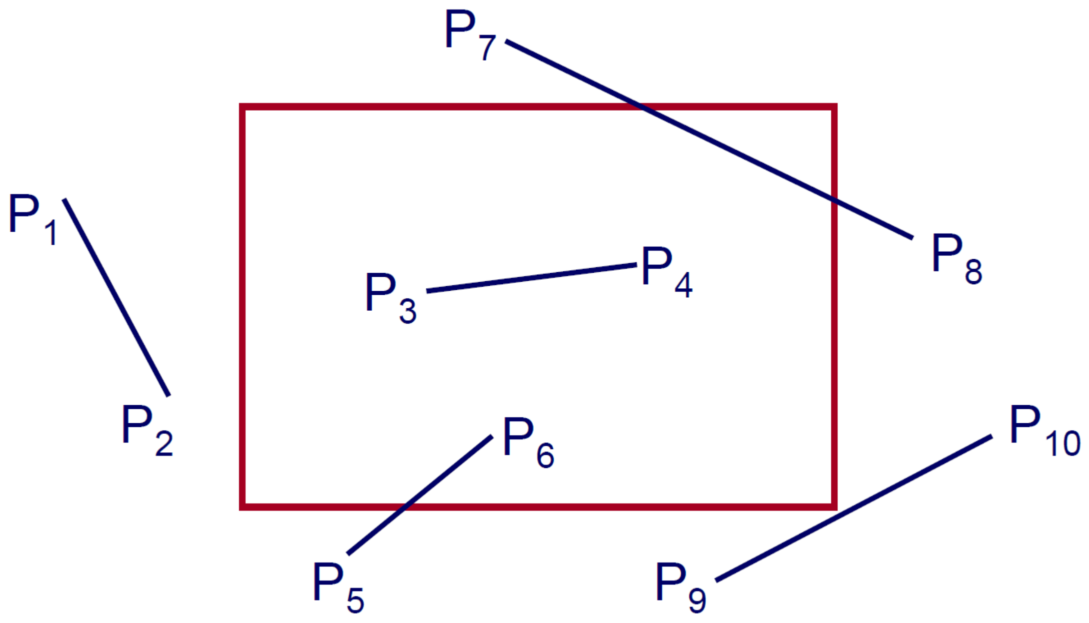
- Is point (x,y) inside clip window ?



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

Line Clipping

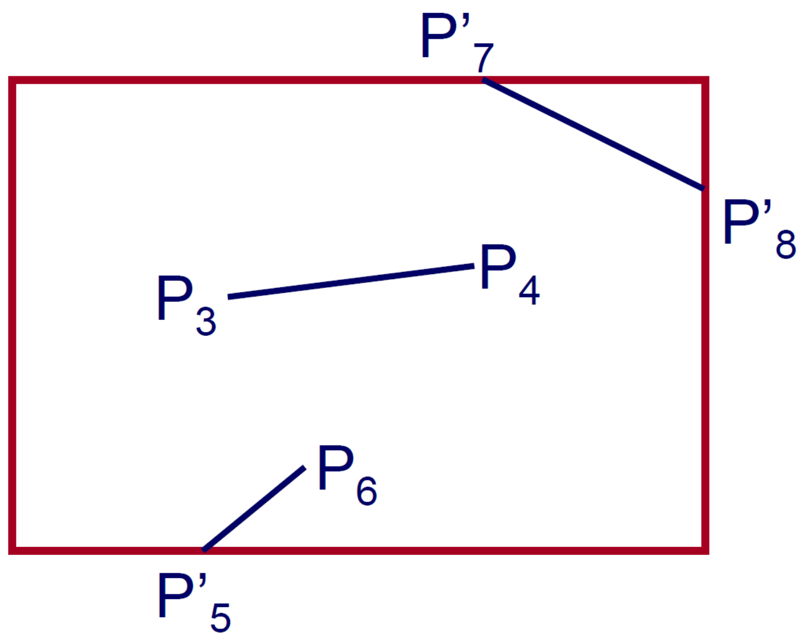
- Find the part of a line inside the clip window



Before Clipping

Line Clipping

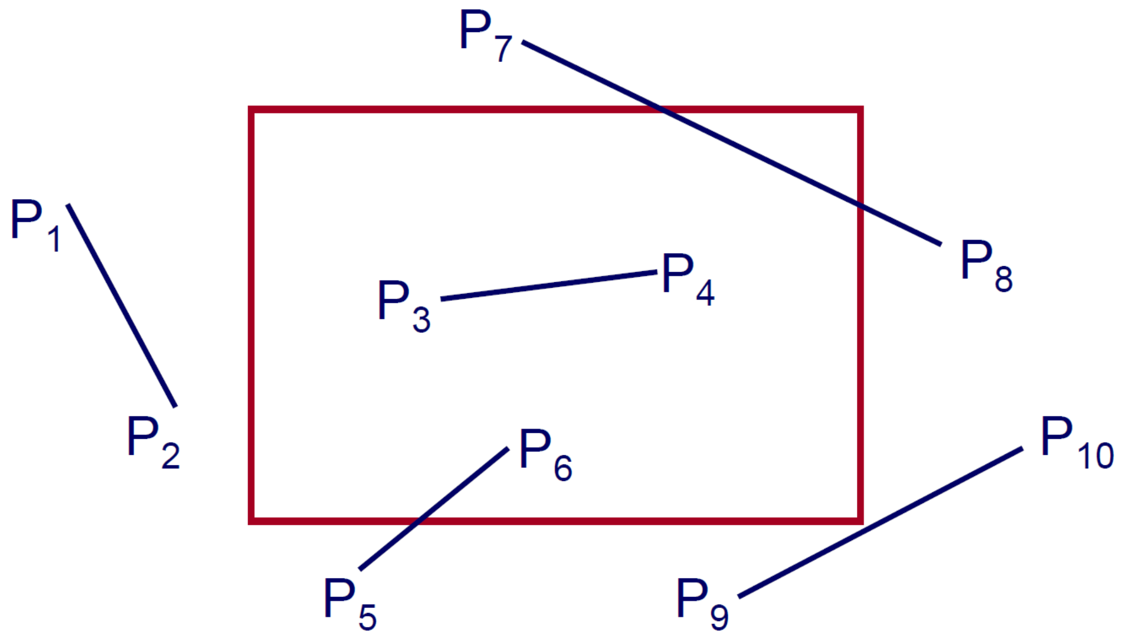
- Find the part of a line inside the clip window



After Clipping

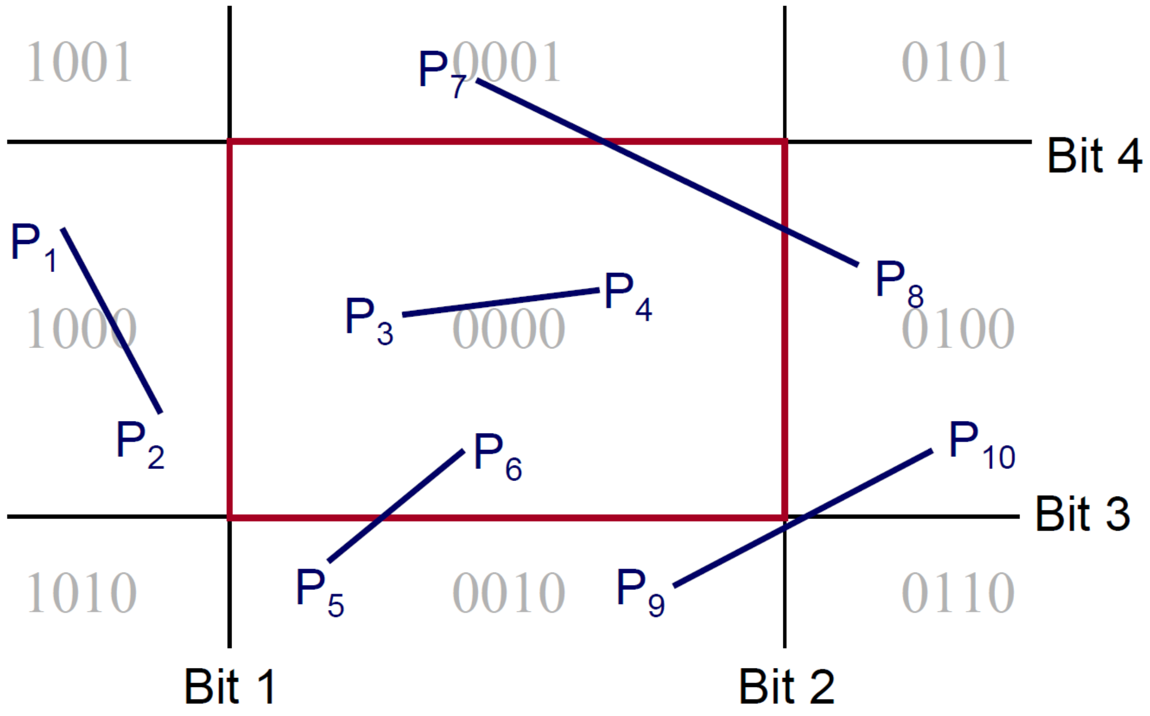
Cohen-Shutherland Line Clipping

- Use simple test to classify easy cases first
- Danny Cohen, Ivan Sutherland 1967



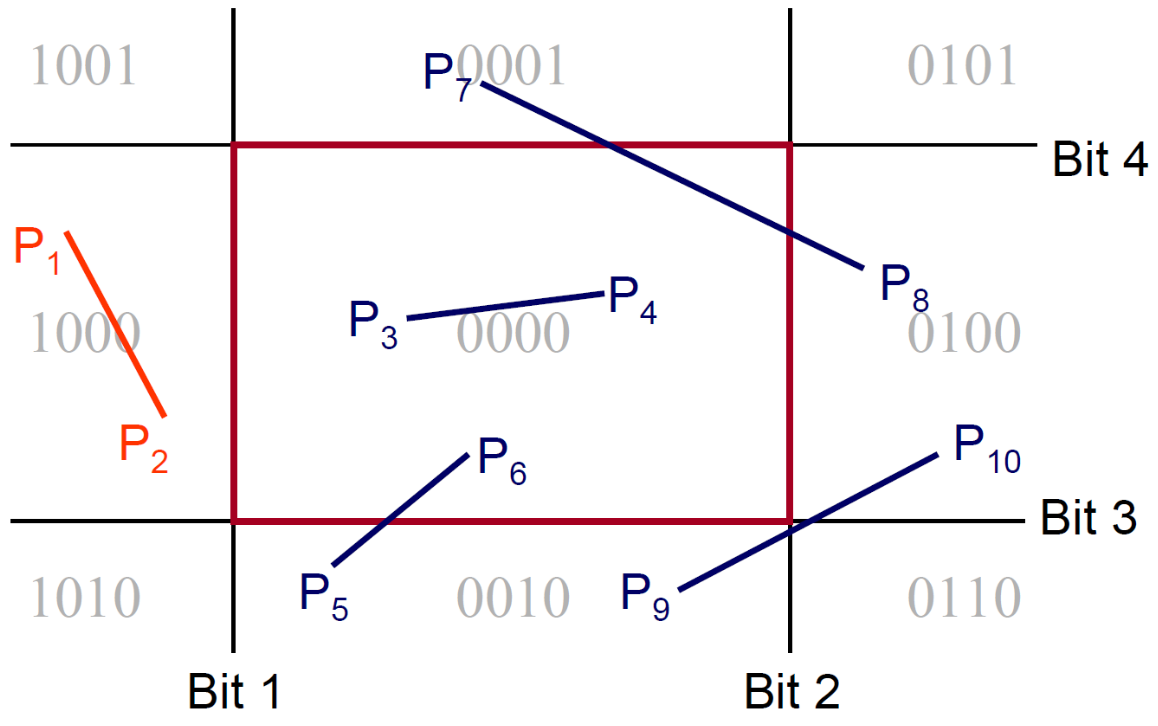
Cohen-Shutherland Line Clipping

- Classify lines quickly by AND of bit codes representing regions of two endpoints (test for 0: inside or clipping, 1: outside)



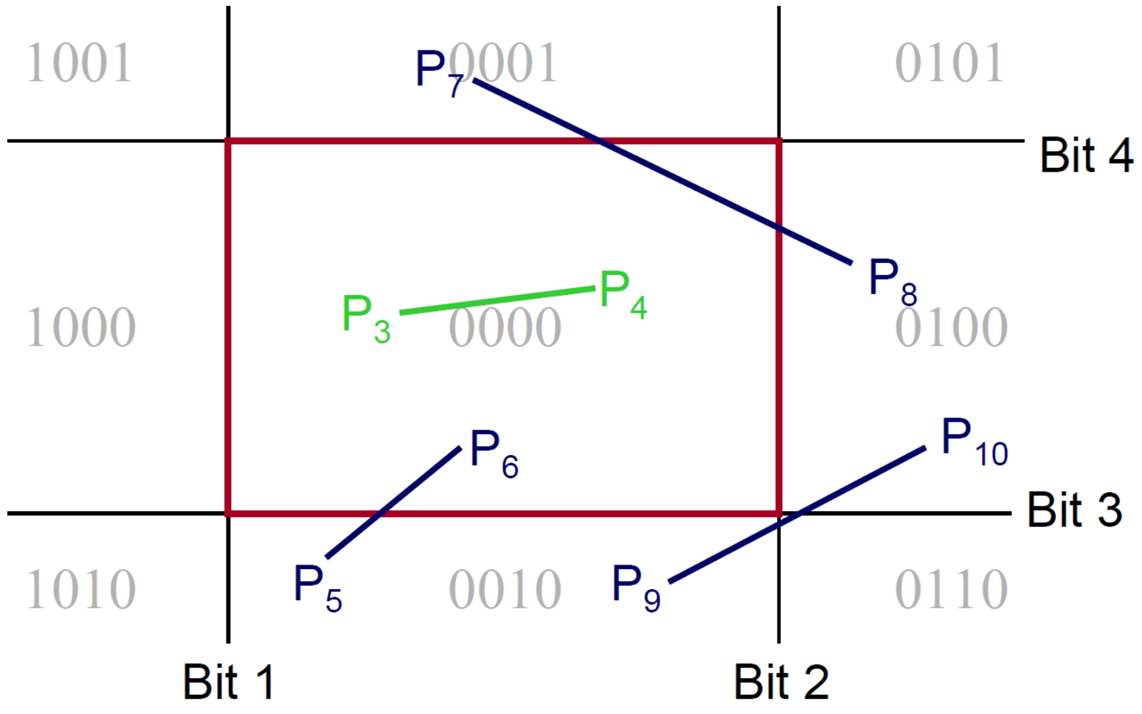
Cohen-Shutherland Line Clipping

- Classify lines quickly by AND of bit codes representing regions of two endpoints (test for 0: inside or clipping, 1: outside)



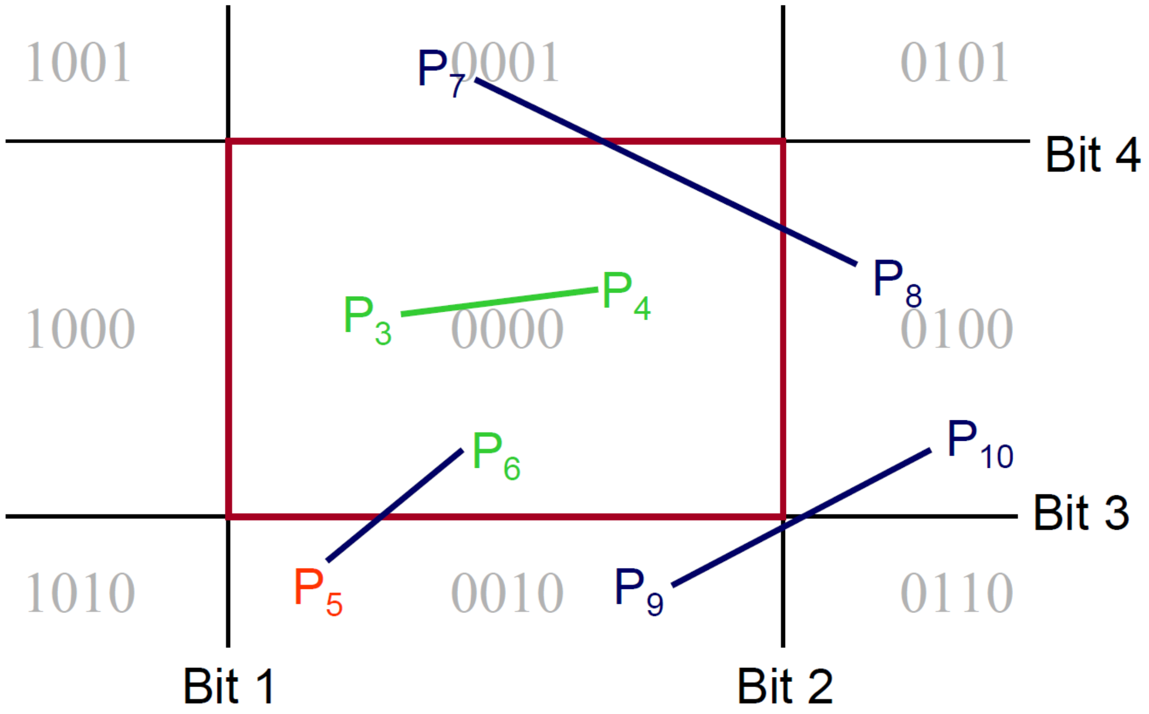
Cohen-Shutherland Line Clipping

- Classify possible clipping lines by OR of bit codes representing regions of two endpoints (test for 0: inside)



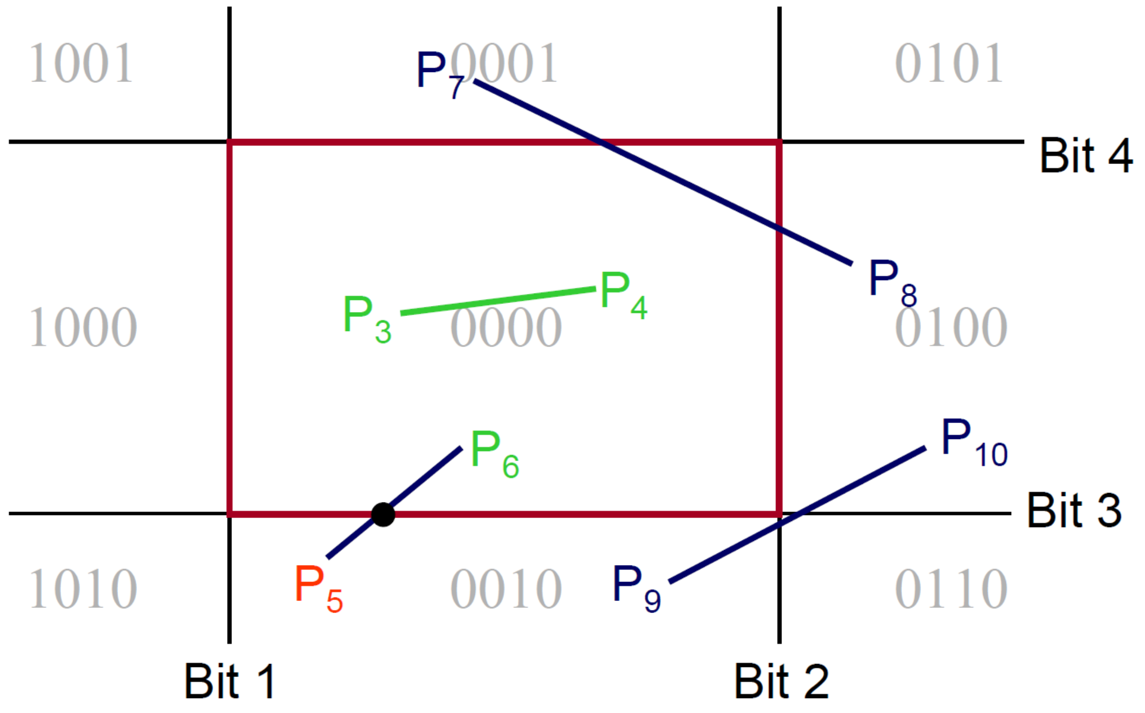
Cohen-Shutherland Line Clipping

- Compute intersections with window boundary for remaining lines, OR of bit codes representing the boundary



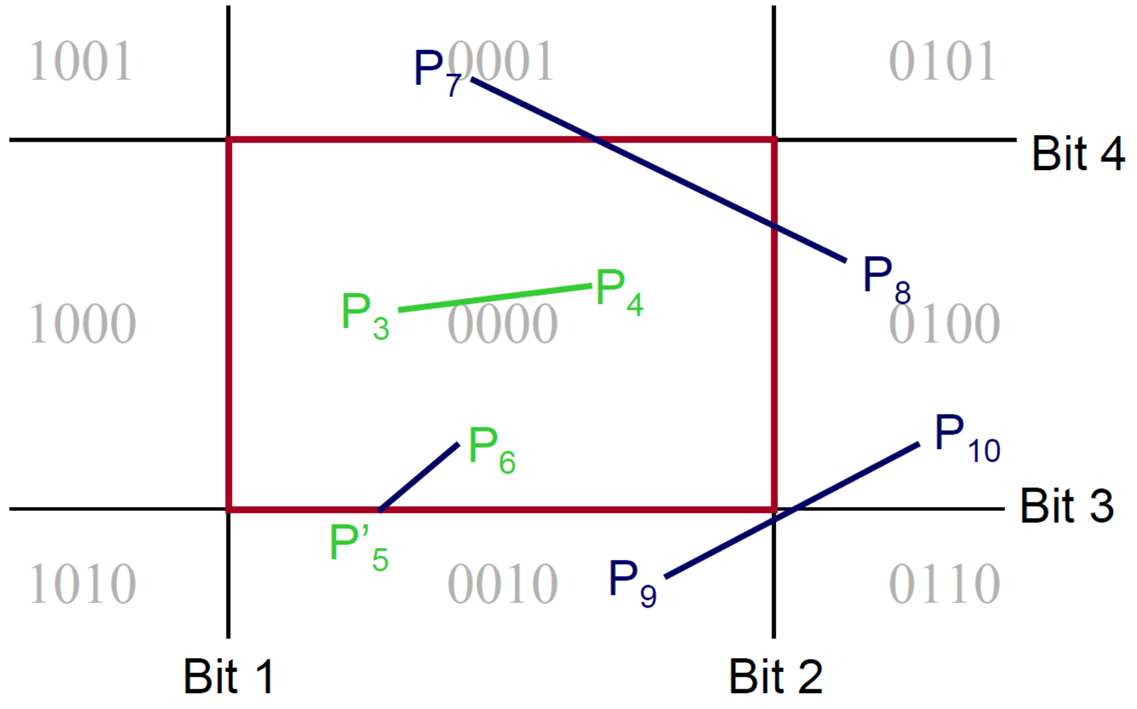
Cohen-Shutherland Line Clipping

- Intersect with boundary determined by the bits of the non zero point and set 0000 for the new point



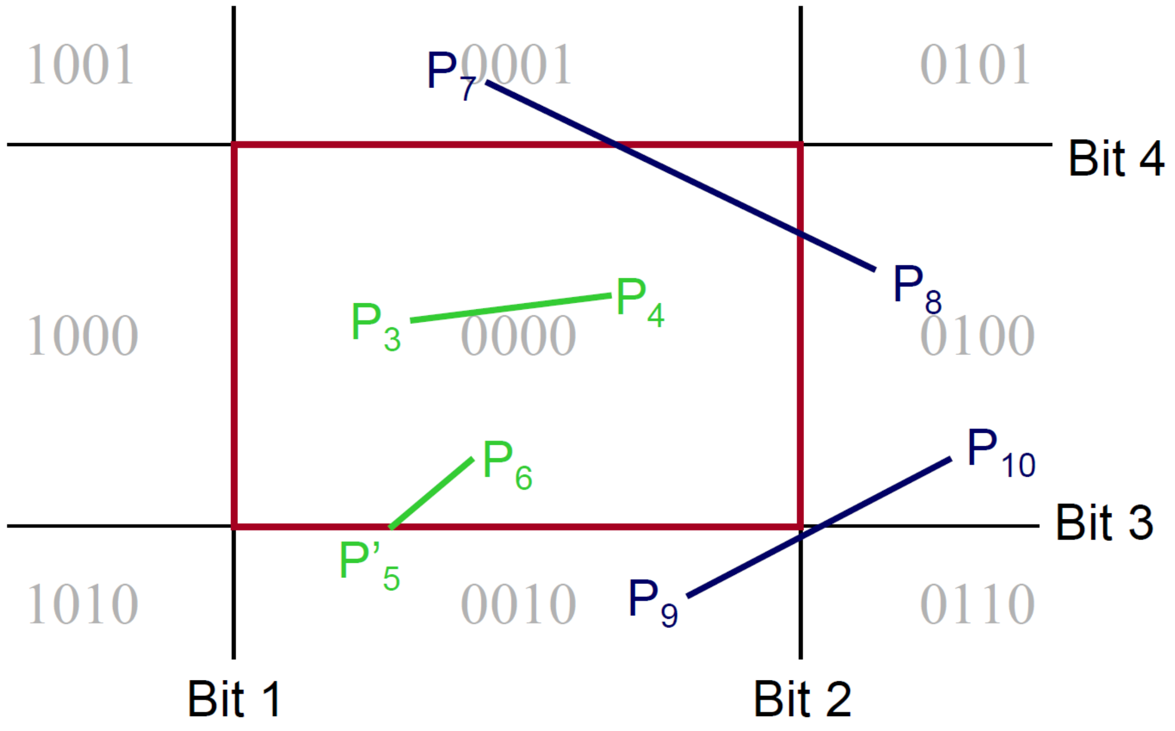
Cohen-Shutherland Line Clipping

- Create new point on the boundary



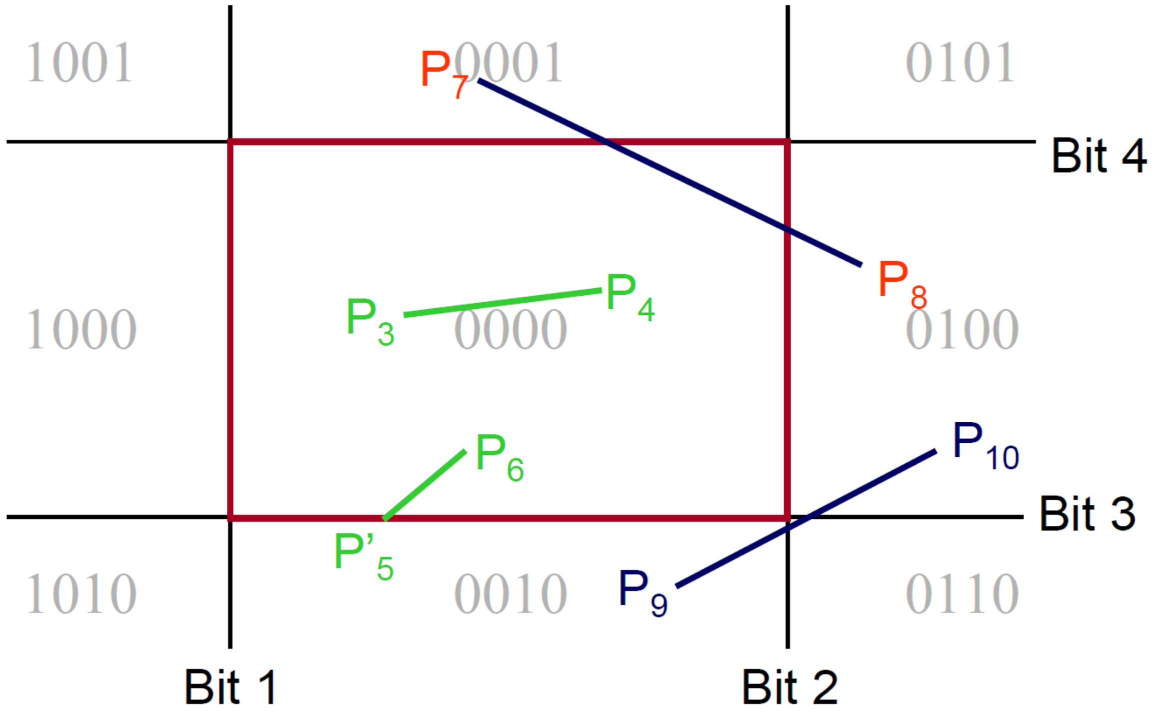
Cohen-Shutherland Line Clipping

- Check using the AND operation again



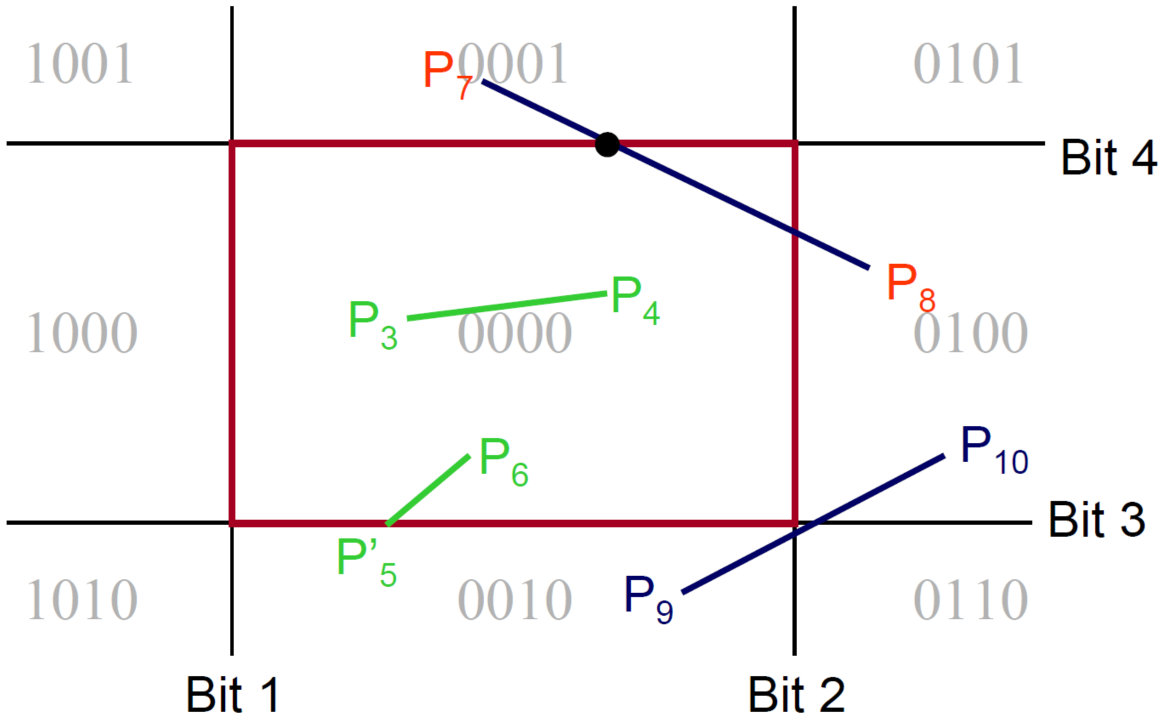
Cohen-Shutherland Line Clipping

- Do the same for the next line



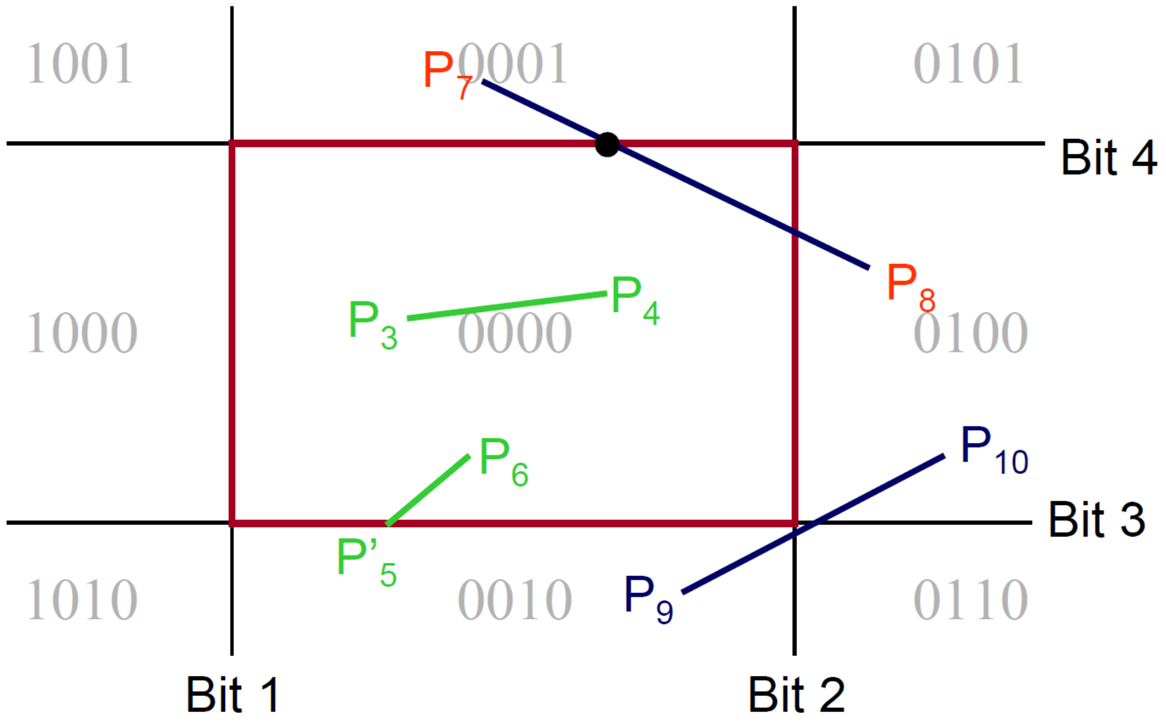
Cohen-Shutherland Line Clipping

- Clip using the boundary determined by P7



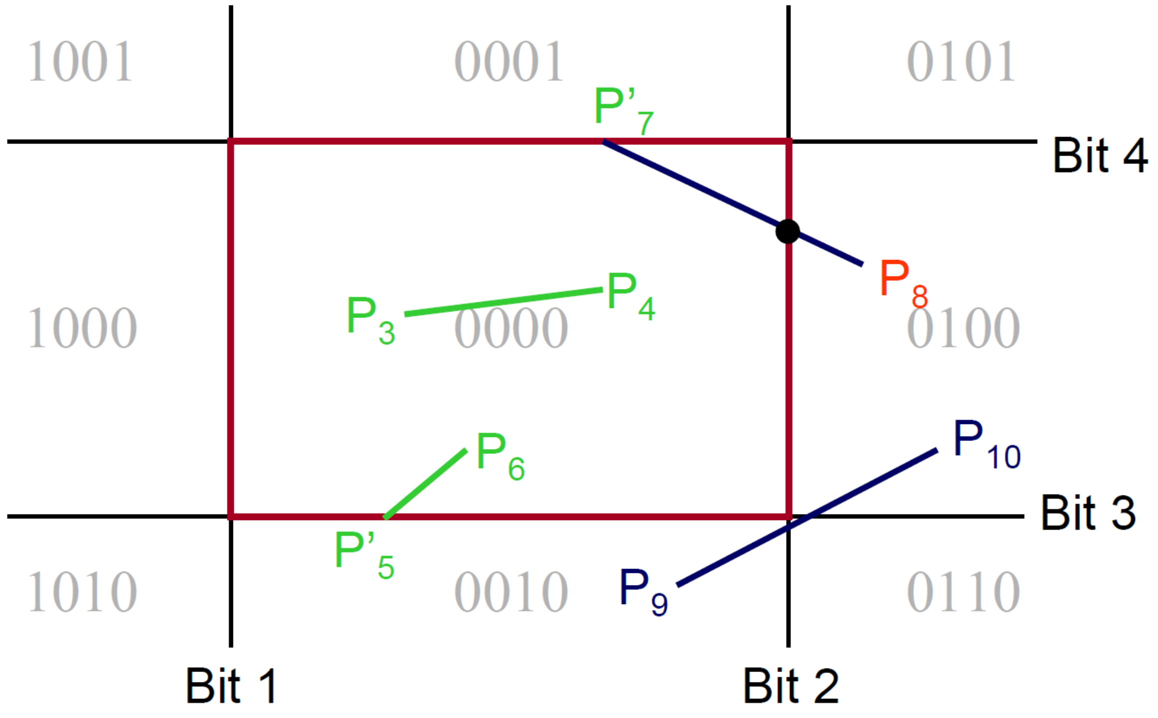
Cohen-Shutherland Line Clipping

- Clip using the boundary determined by P7



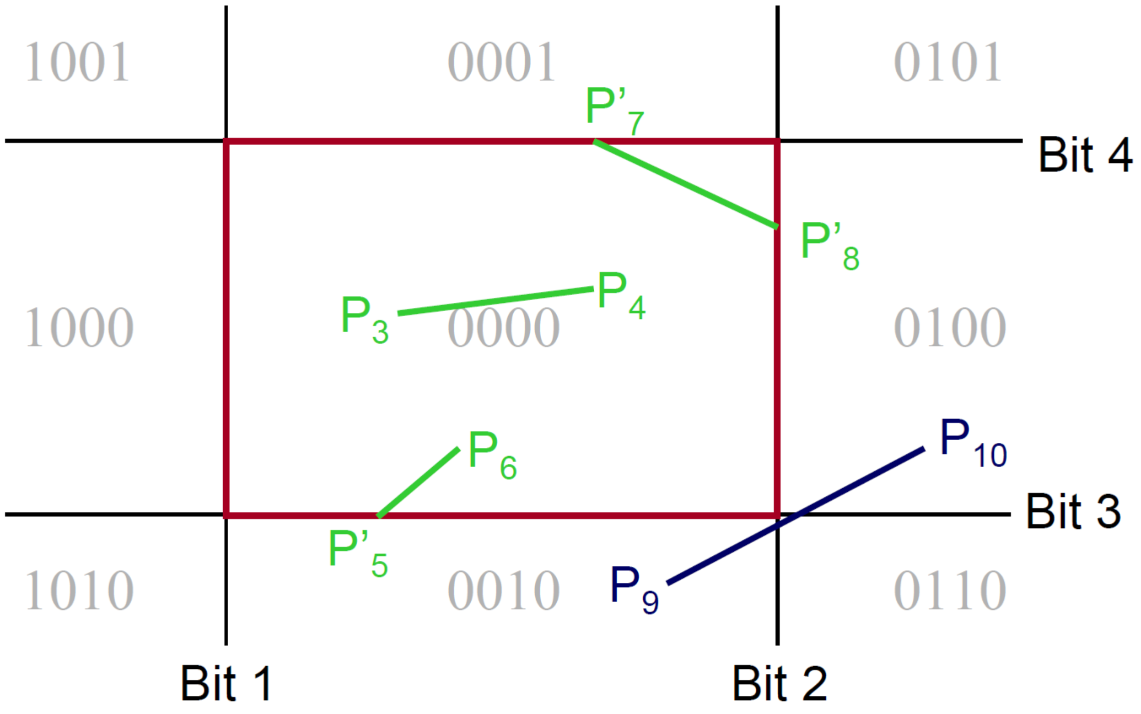
Cohen-Shutherland Line Clipping

- Clip using the boundary determined by P8



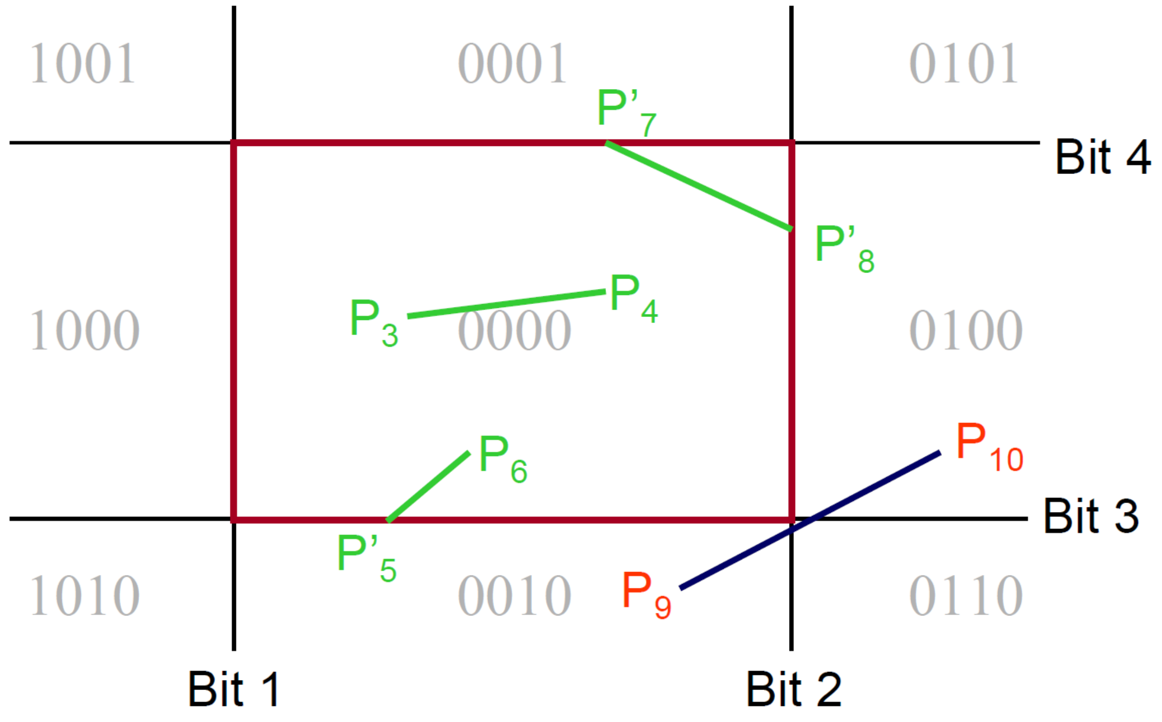
Cohen-Shutherland Line Clipping

- Test the line again using AND



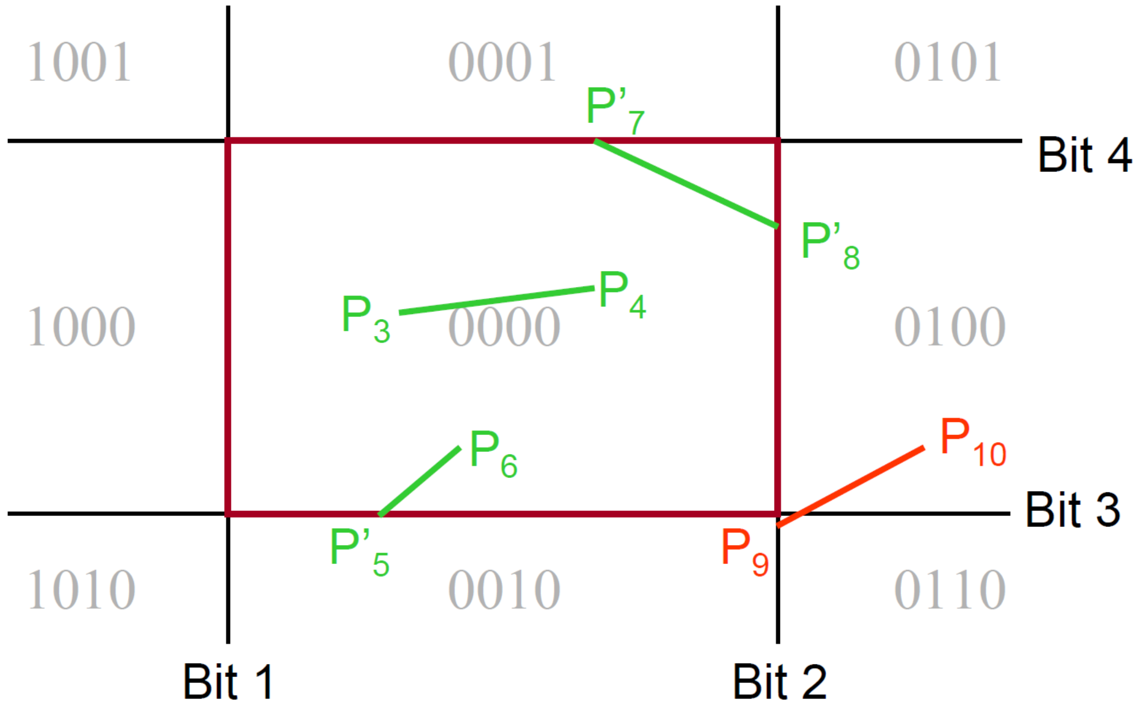
Cohen-Shutherland Line Clipping

- Again for the last line



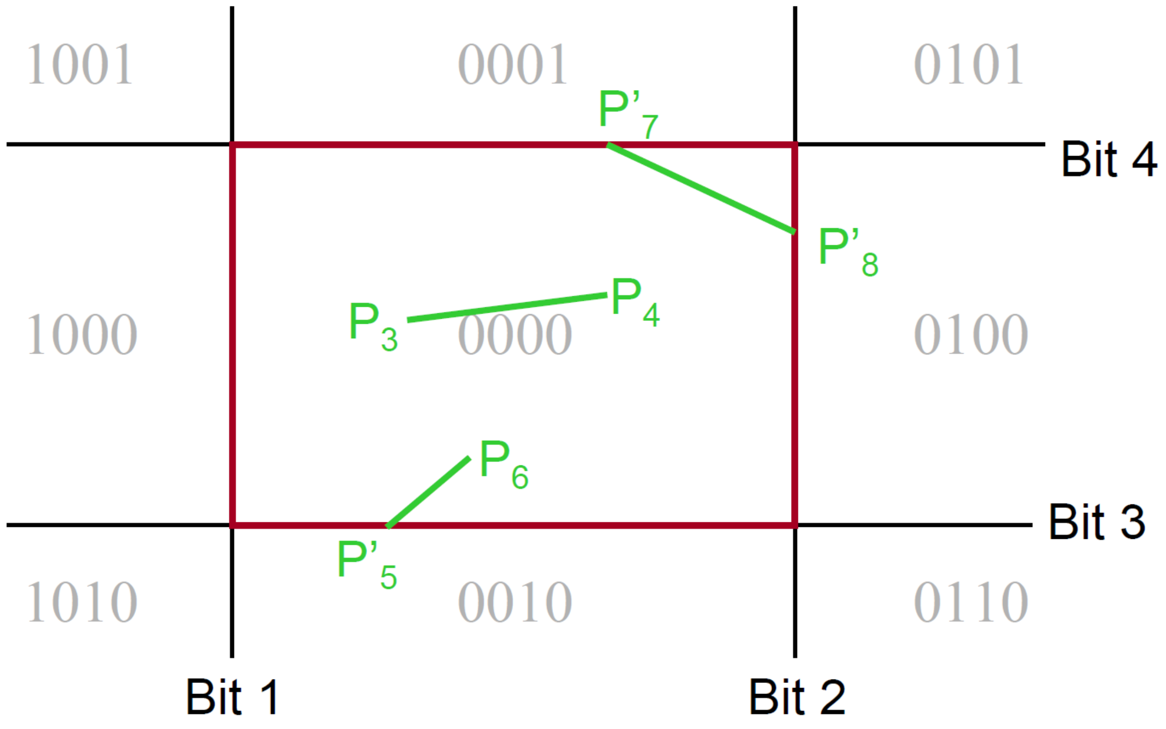
Cohen-Shutherland Line Clipping

- P9 AND P10 no longer zero



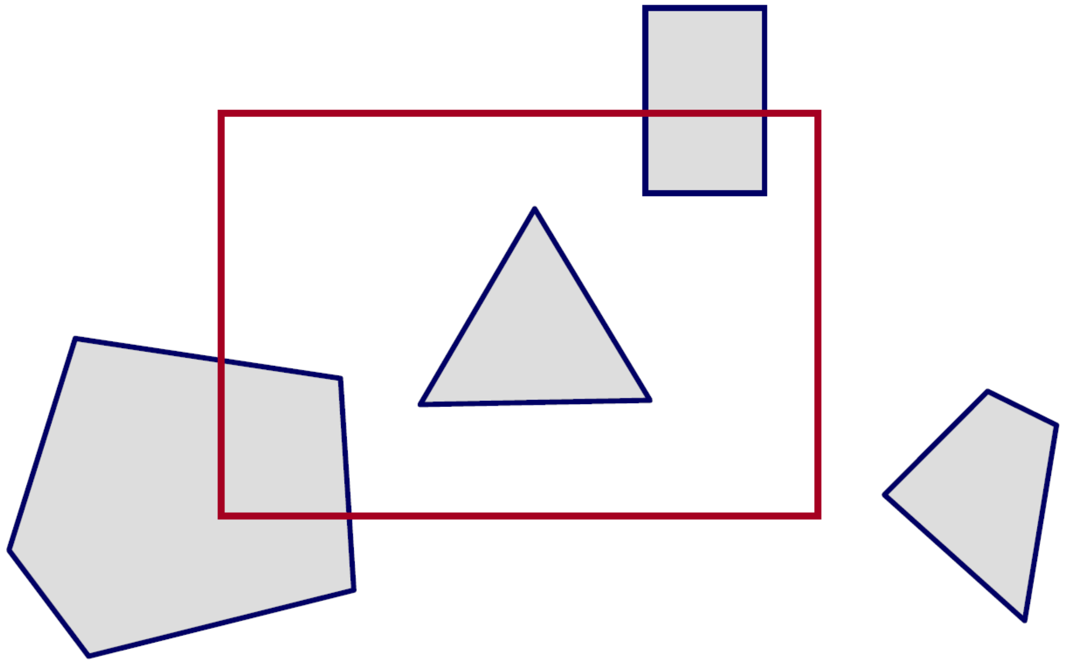
Cohen-Shutherland Line Clipping

- Final result



Polygon Clipping

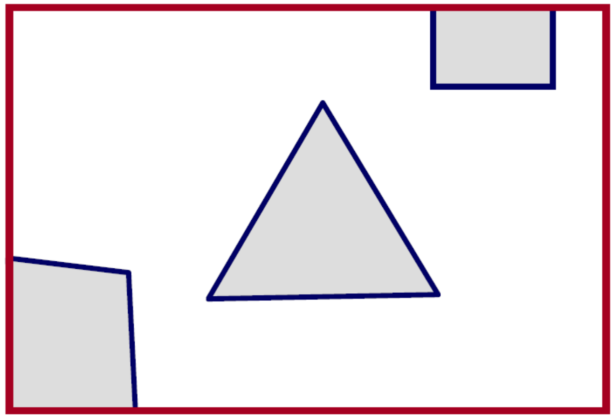
- Find the part of a polygon inside the clip window?



Before Clipping

Polygon Clipping

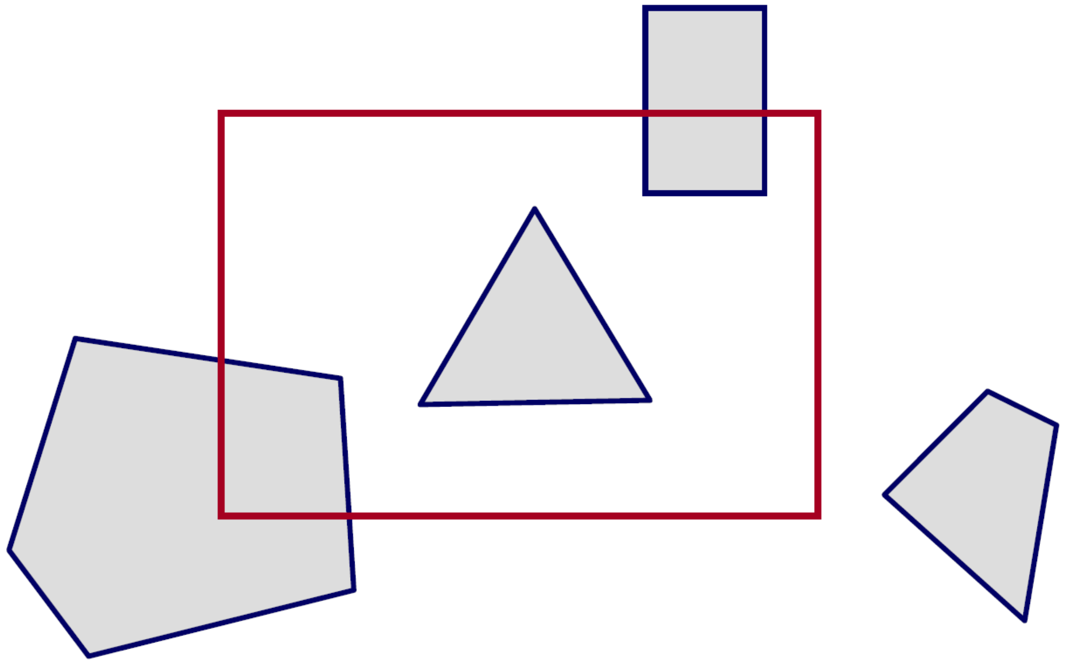
- Find the part of a polygon inside the clip window?



After Clipping

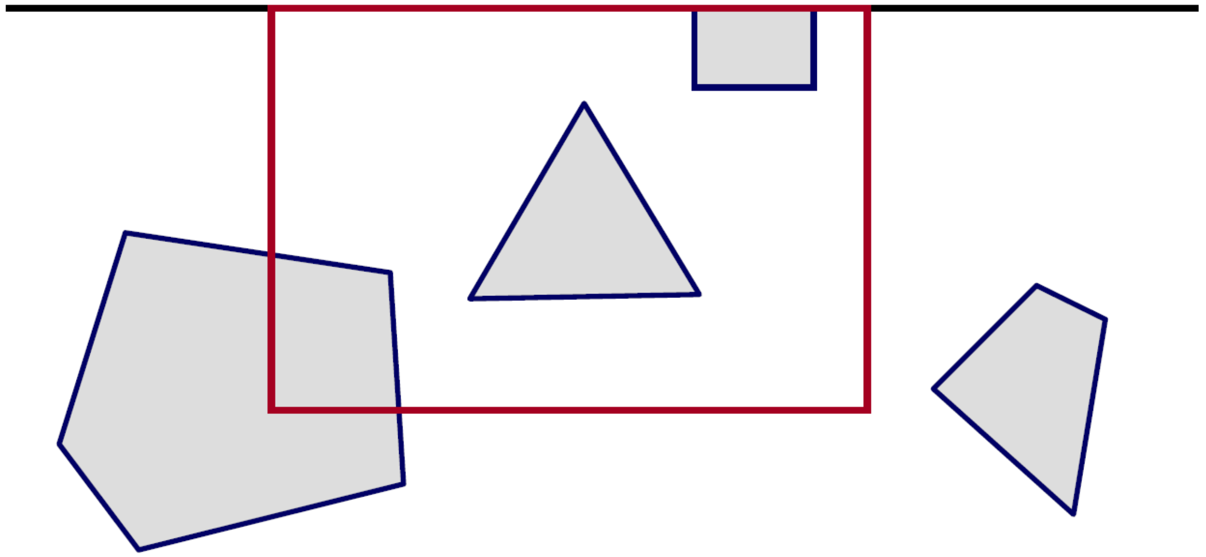
Sutherland–Hodgman Clipping

- Clip to each window boundary one at a time



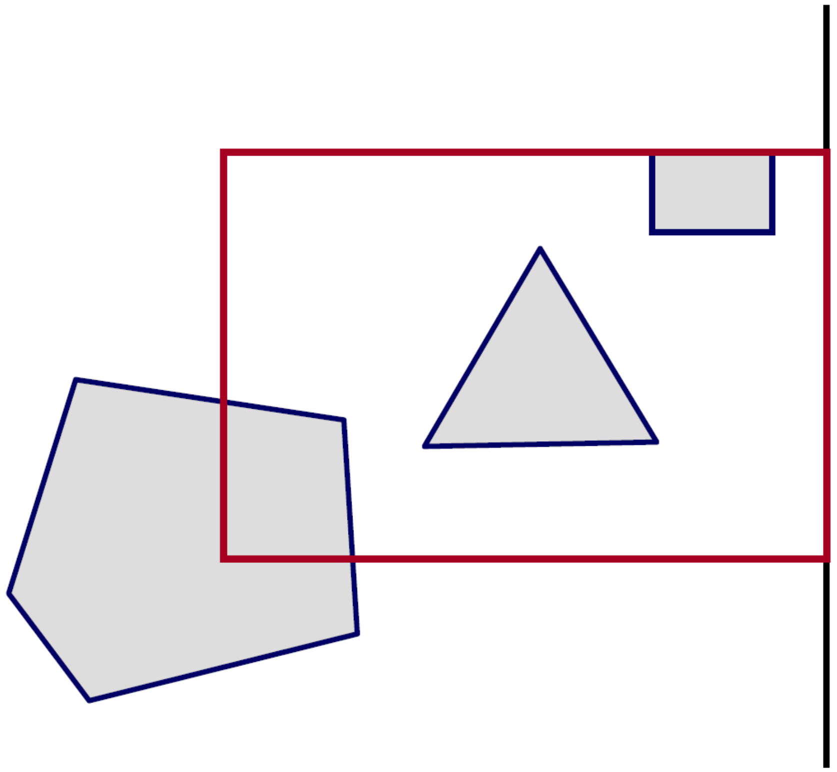
Sutherland–Hodgman Clipping

- Clip to each window boundary one at a time



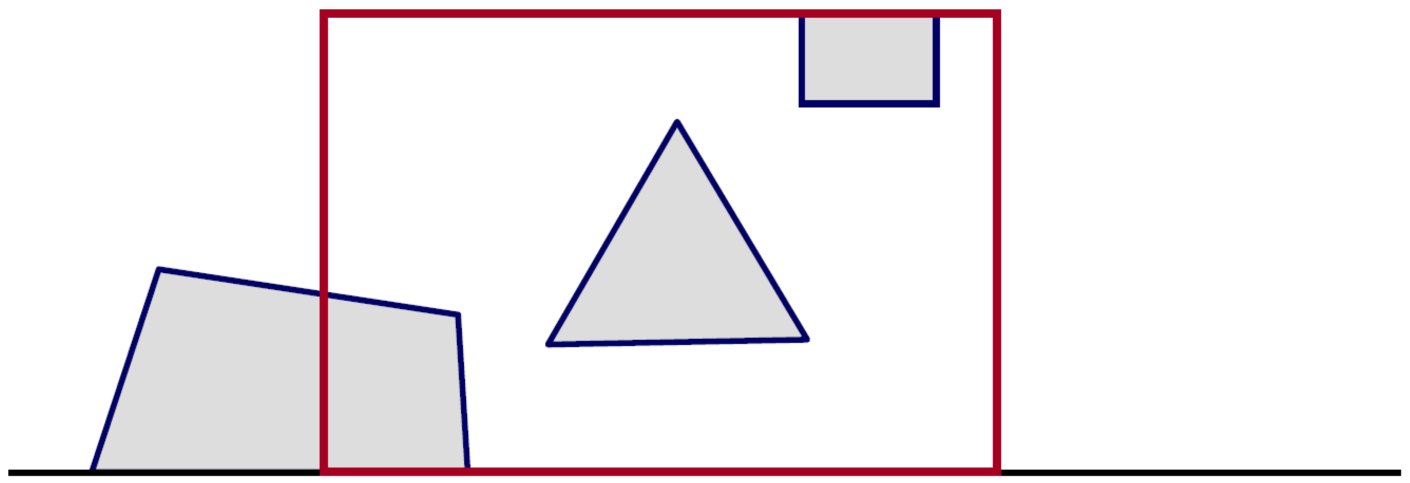
Sutherland–Hodgman Clipping

- Clip to each window boundary one at a time



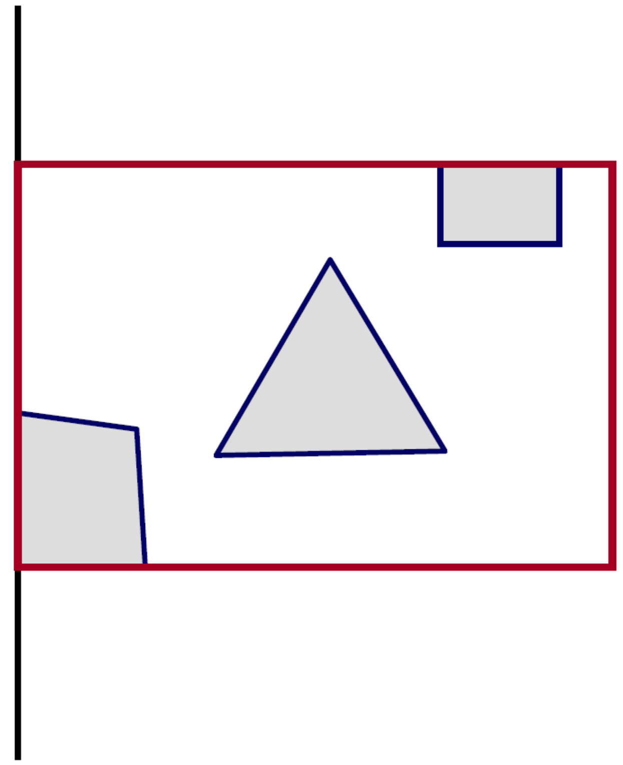
Sutherland–Hodgman Clipping

- Clip to each window boundary one at a time



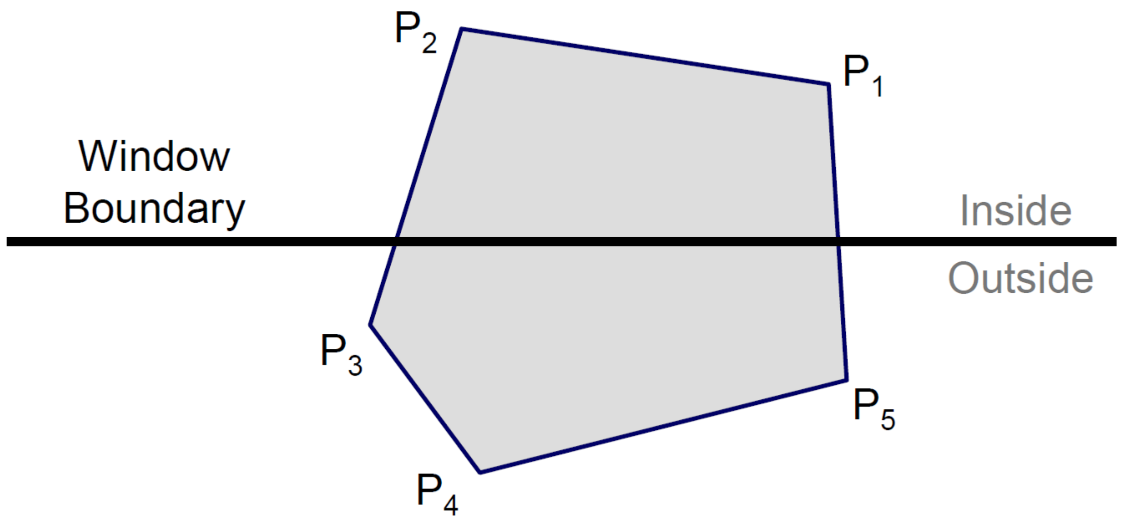
Sutherland–Hodgman Clipping

- Clip to each window boundary one at a time



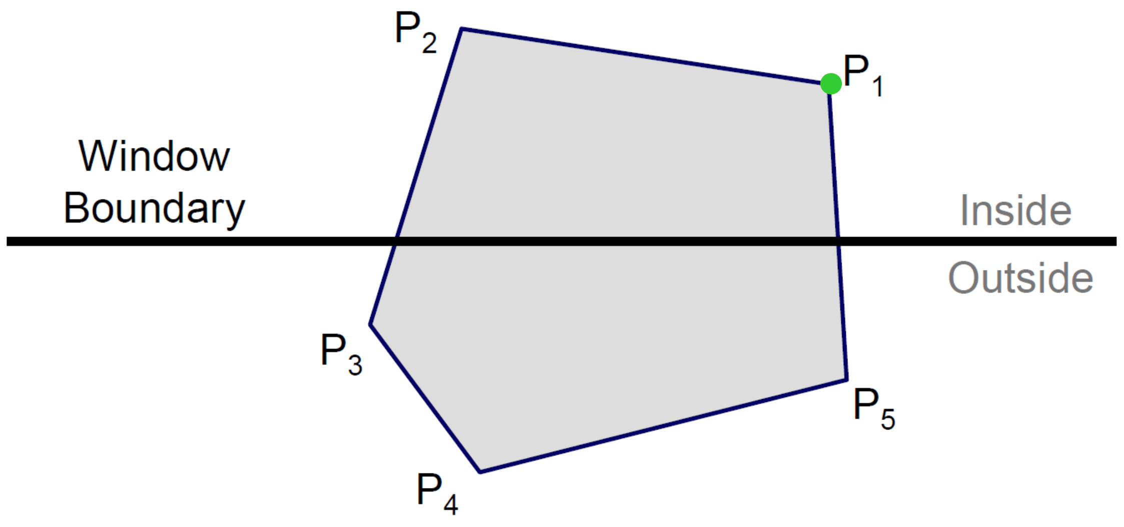
Clipping to a Boundary

- Do inside test for each point in sequence
- Insert new points when crossing the boundary
- Remove points outside of boundary



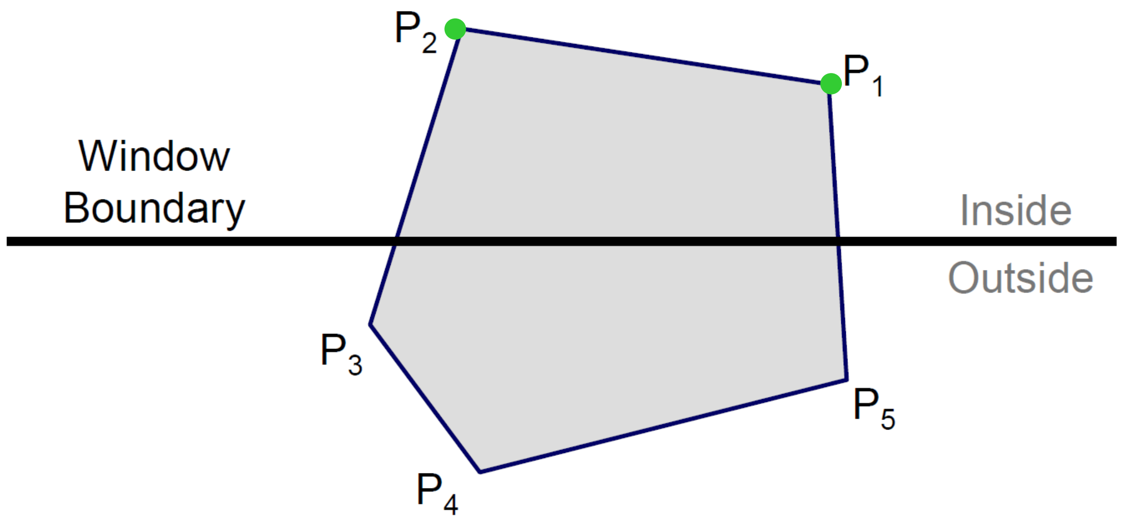
Clipping to a Boundary

- Do inside test for each point in sequence
- Insert new points when crossing the boundary
- Remove points outside of boundary



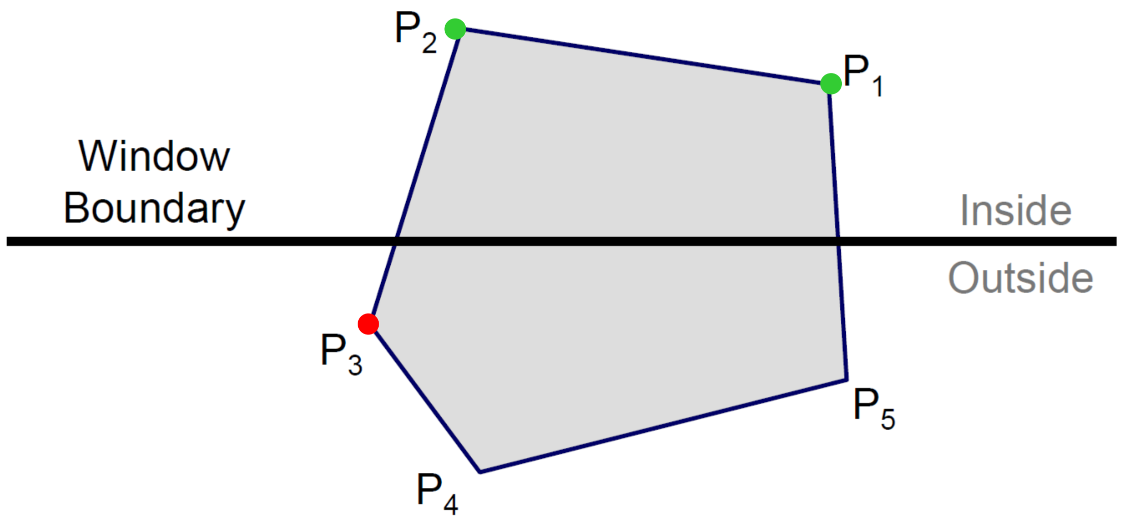
Clipping to a Boundary

- Do inside test for each point in sequence
- Insert new points when crossing the boundary
- Remove points outside of boundary



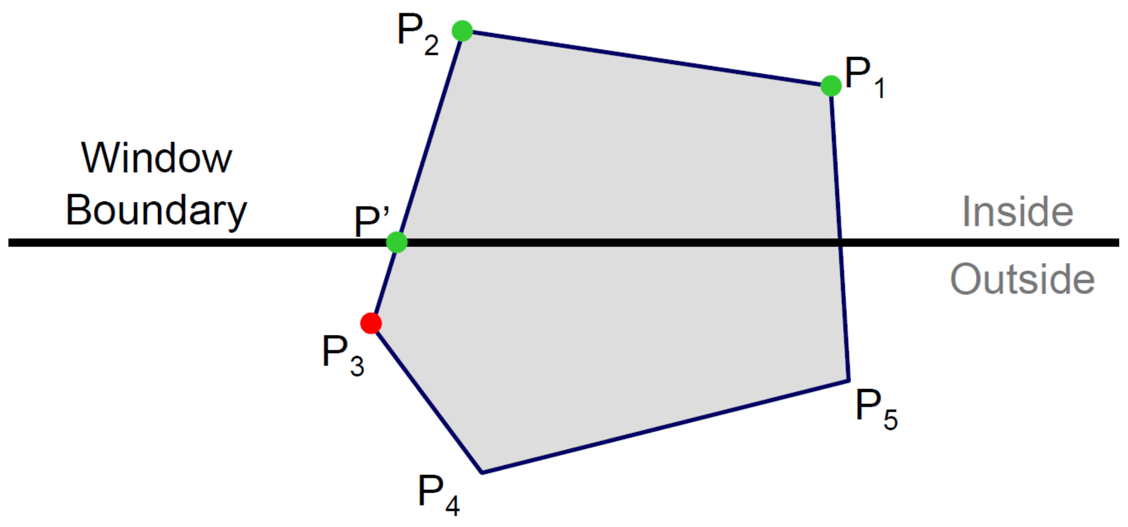
Clipping to a Boundary

- Do inside test for each point in sequence
- Insert new points when crossing the boundary
- Remove points outside of boundary



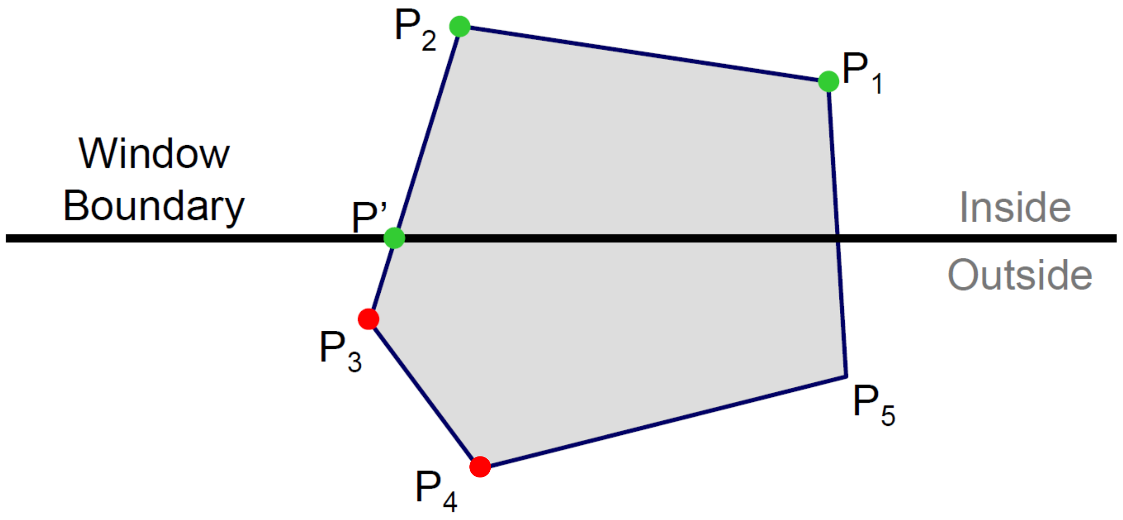
Clipping to a Boundary

- Do inside test for each point in sequence
- Insert new points when crossing the boundary
- Remove points outside of boundary



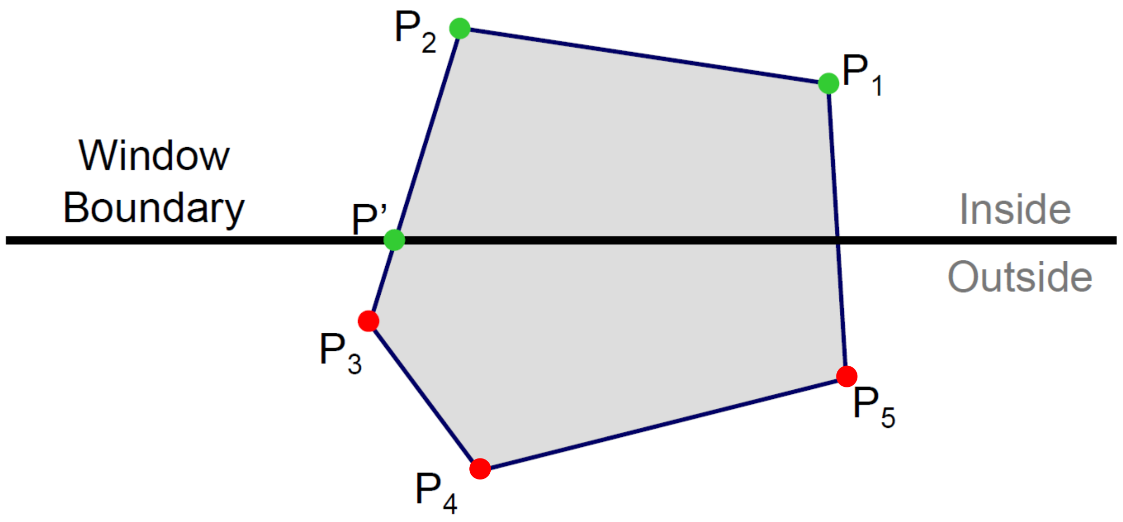
Clipping to a Boundary

- Do inside test for each point in sequence
- Insert new points when crossing the boundary
- Remove points outside of boundary



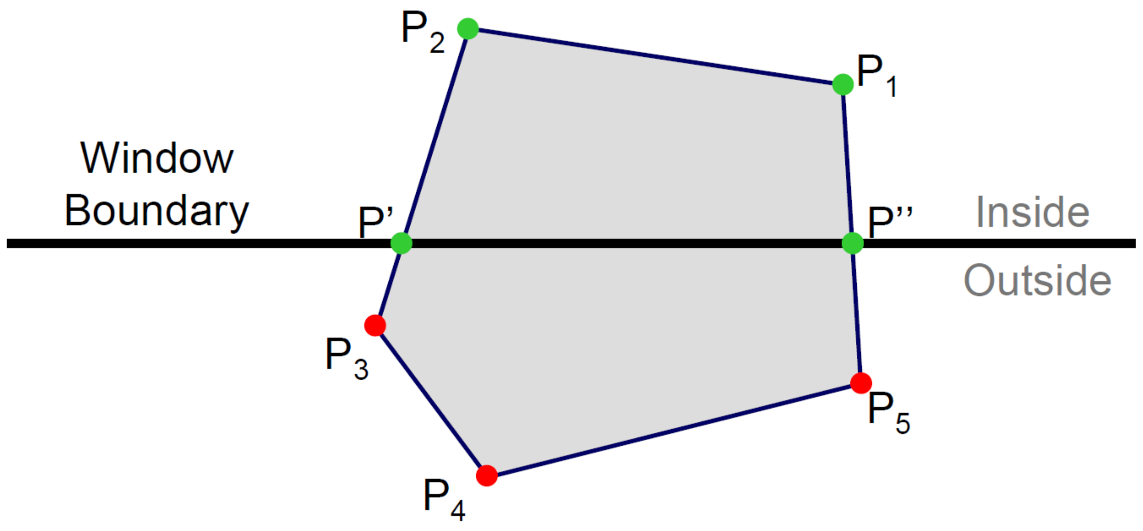
Clipping to a Boundary

- Do inside test for each point in sequence
- Insert new points when crossing the boundary
- Remove points outside of boundary



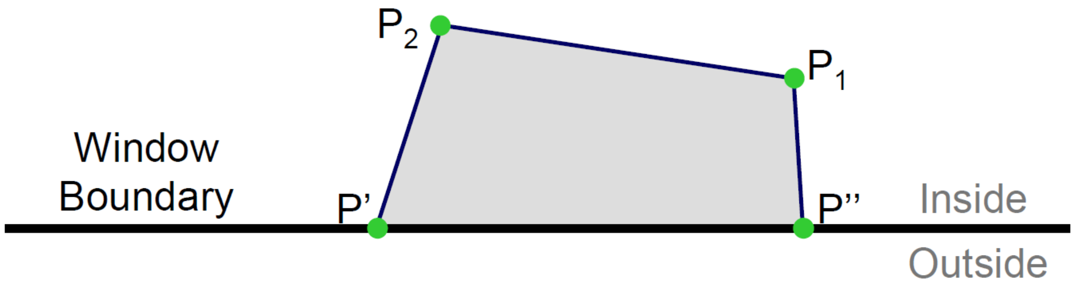
Clipping to a Boundary

- Do inside test for each point in sequence
- Insert new points when crossing the boundary
- Remove points outside of boundary

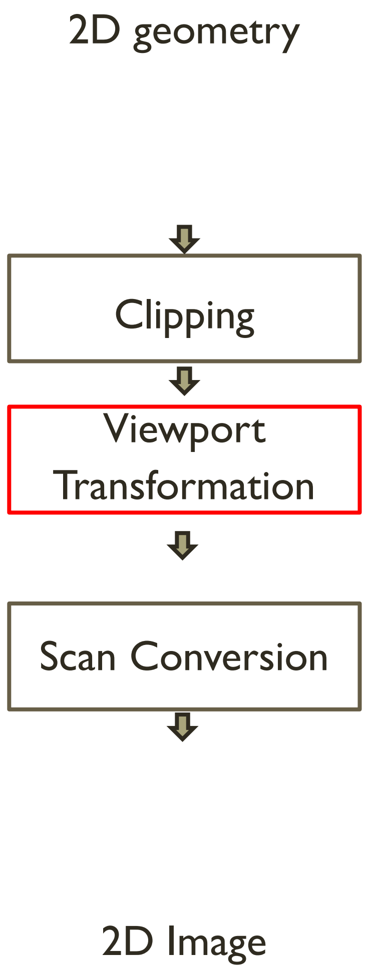


Clipping to a Boundary

- Do inside test for each point in sequence
- Insert new points when crossing the boundary
- Remove points outside of boundary



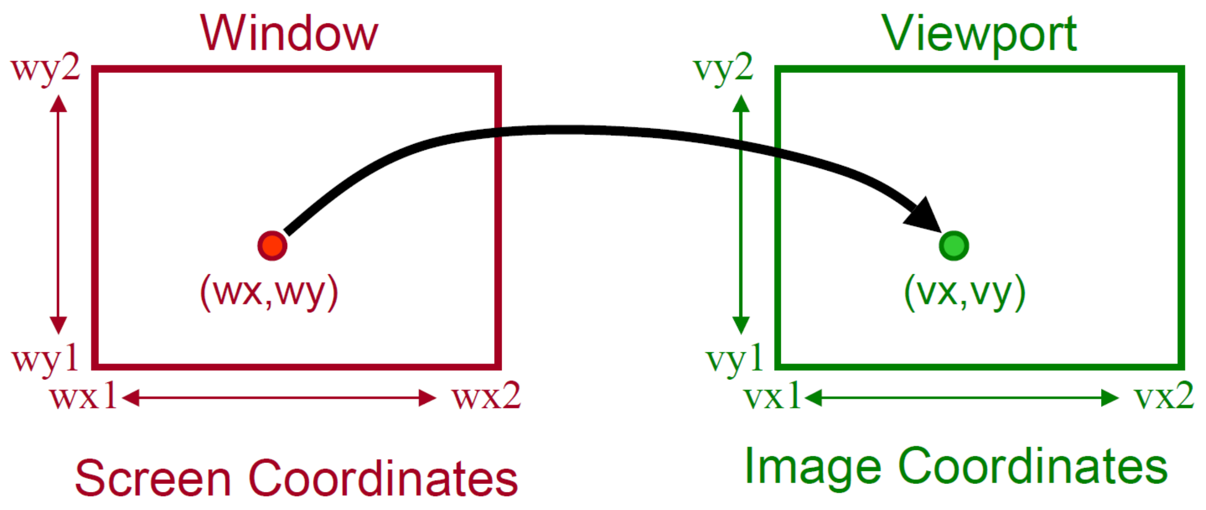
2D rendering pipeline



- Clip and remove geometry outside of the window
- Transform from screen coordinates to image coordinates
- Fill pixels on the screen

Viewport Transformation

- Window to viewport mapping

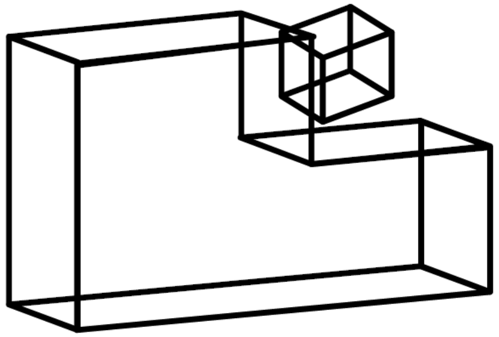


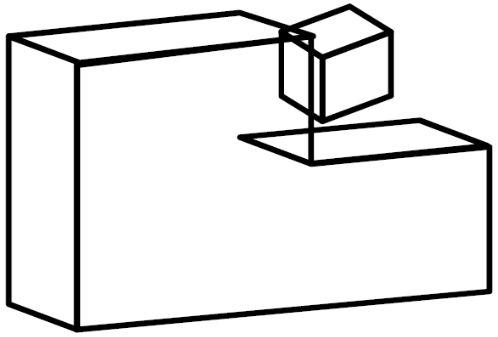
```
vx = vx1 + (wx - wx1) * (vx2 - vx1) / (wx2 - wx1);
vy = vy1 + (wy - wy1) * (vy2 - vy1) / (wy2 - wy1);
```

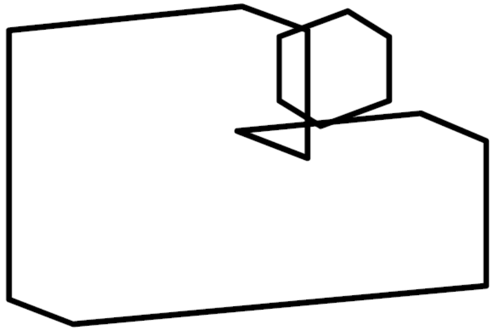
Overview

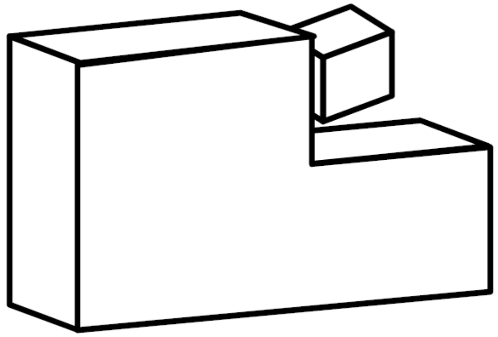
- Clipping
  - Point Clipping
  - Line Clipping
  - Polygon Clipping
- **Hidden Surface Removal**

Visibility


wireframe model

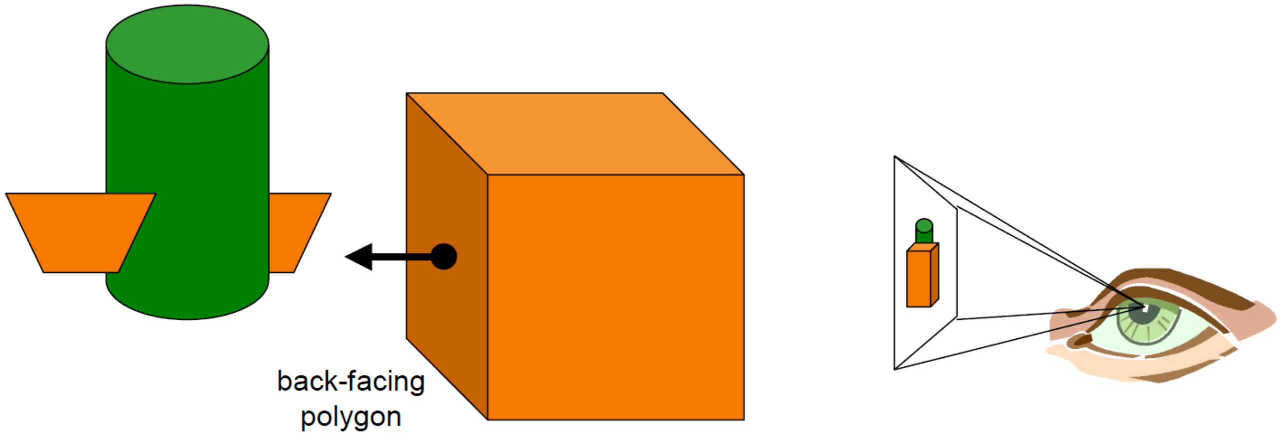

front faces


silhouette

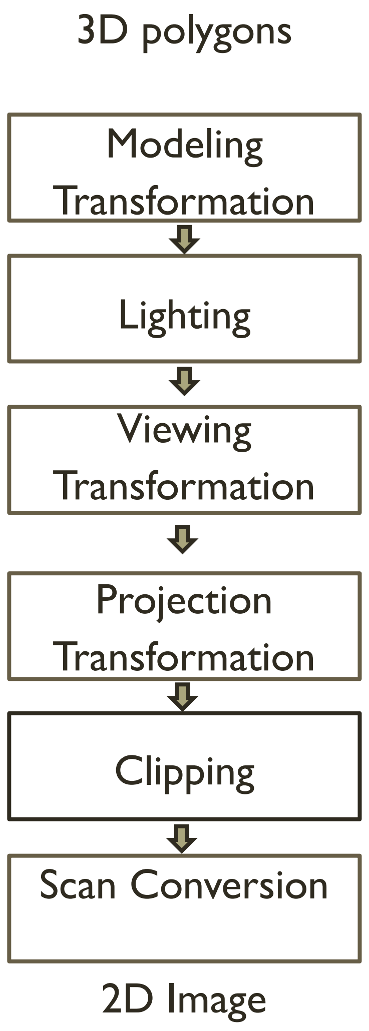

visible faces, edges

Motivation

- Surfaces may be back-facing
- Surfaces may be occluded
- Surfaces may overlap in the image plane
- Surfaces may intersect



3D rendering pipeline



- Somewhere here we have to determine which objects are visible and which are hidden

Basic algorithms for HSR

- Clipping
  - Point Clipping
  - Line Clipping
  - Polygon Clipping
- **Hidden Surface Removal**

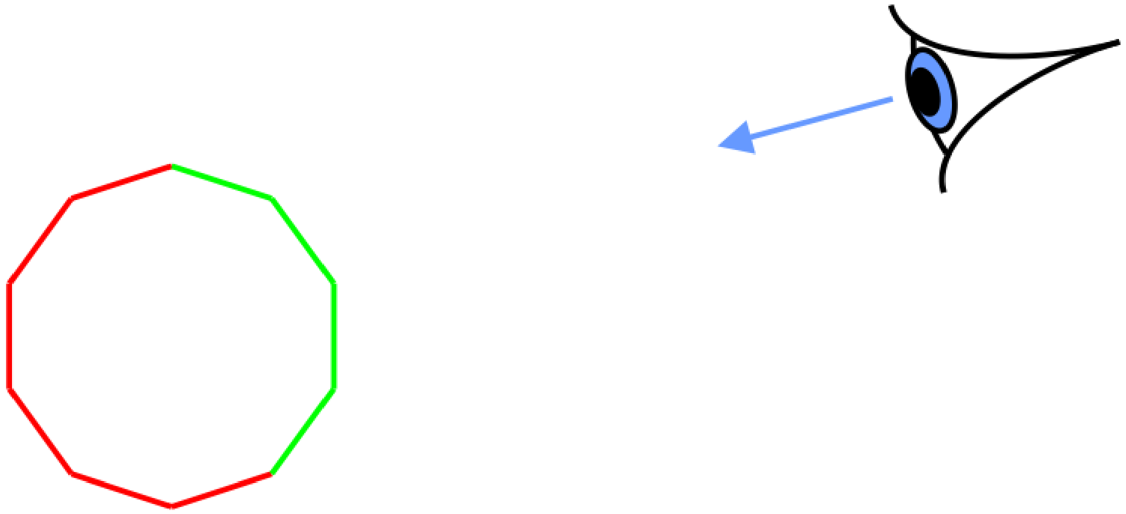
Optimizing visibility

- Get rid of objects that are surely not visible

- Frustum culling
- Occlusion culling
- Back-face culling

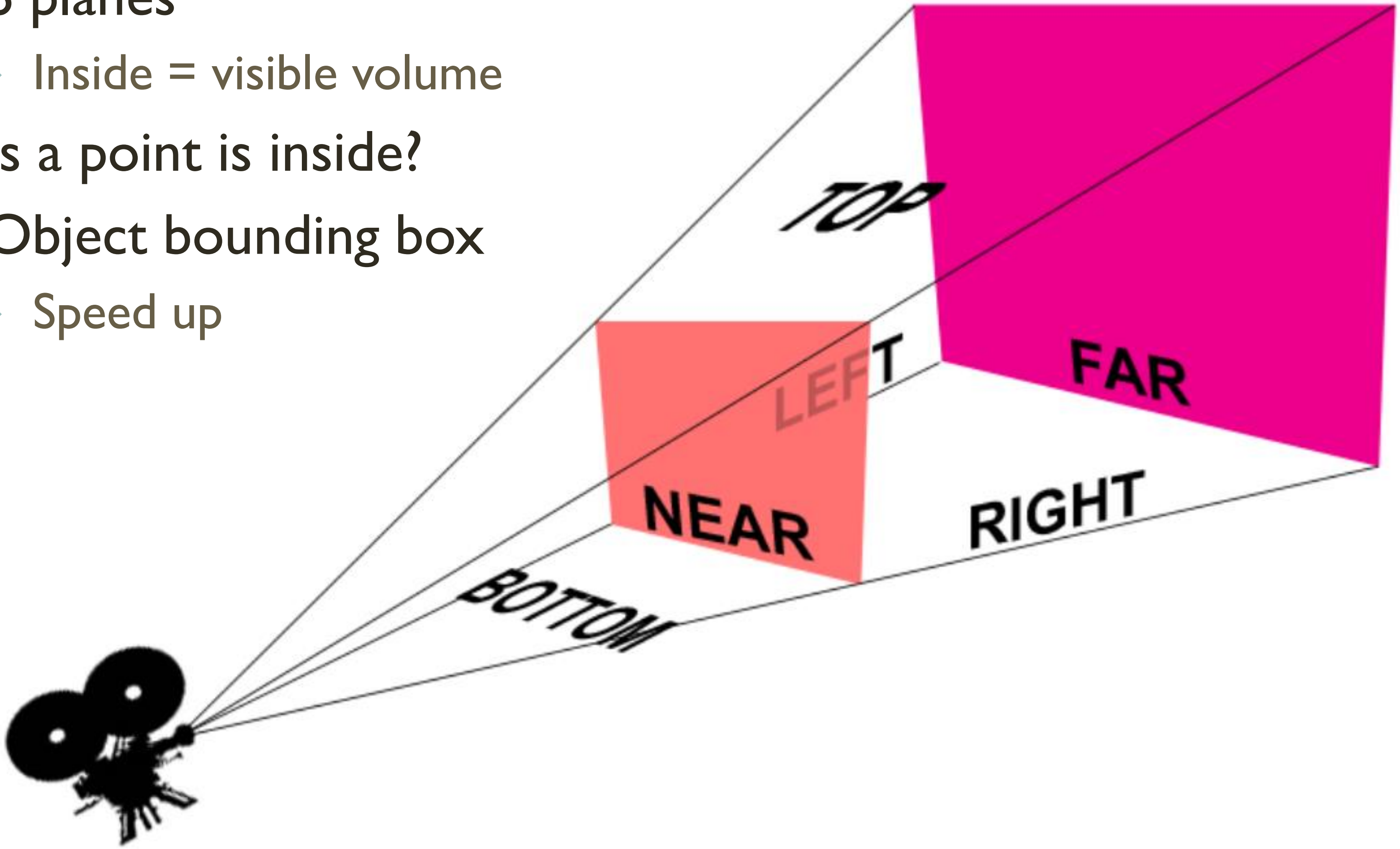
Back-face culling

- Which object faces are visible?
- Remember normal vector (face orientation)



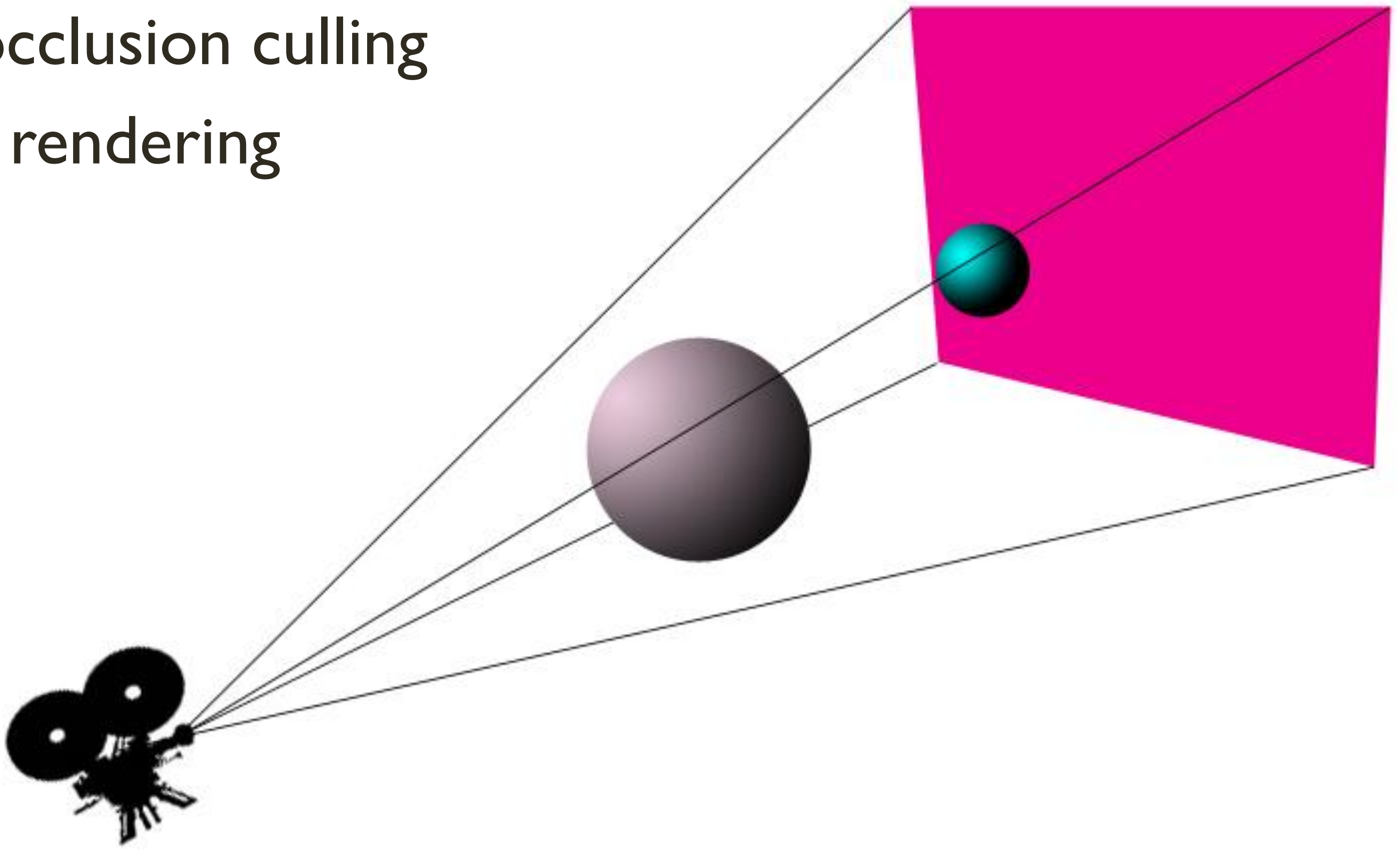
Frustum culling

- 6 planes
  - Inside = visible volume
- Is a point is inside?
- Object bounding box
  - Speed up



Occlusion culling

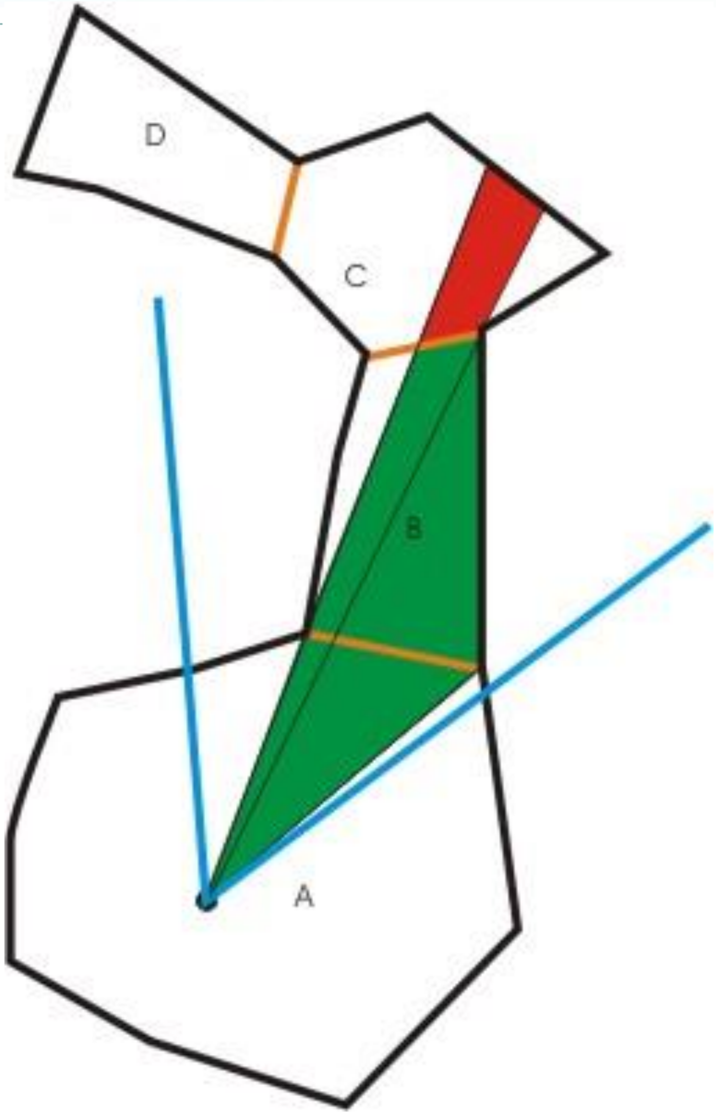
- Some objects are fully occluded by others
- Spatial relations between objects
- Portals, occlusion culling
- Realtime rendering



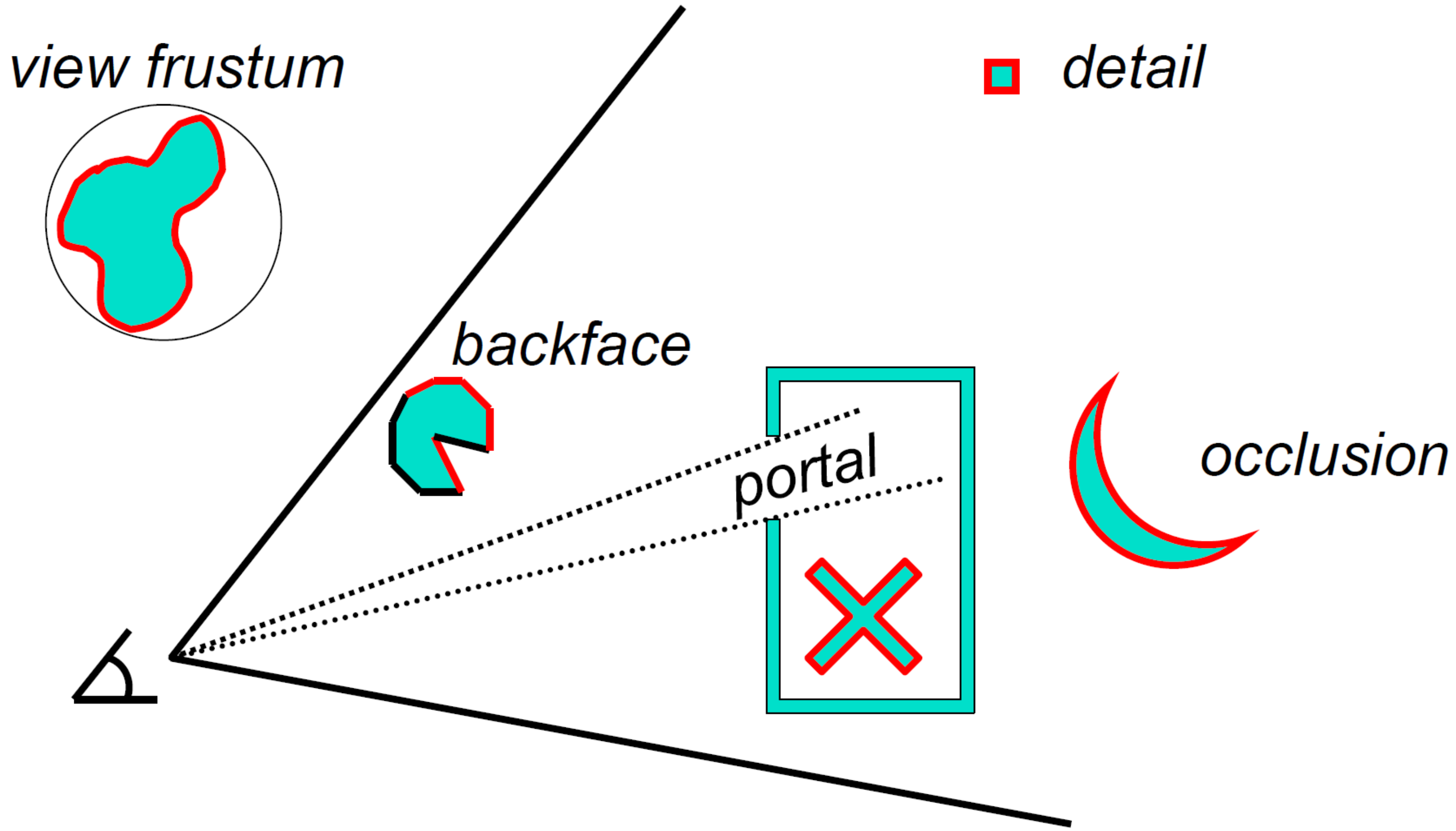
Portal culling

- Some parts of the scene are not visible from some other parts of the scene

1
2
3
4
5
6
tra-Luxe Casino



Optimizing visibility

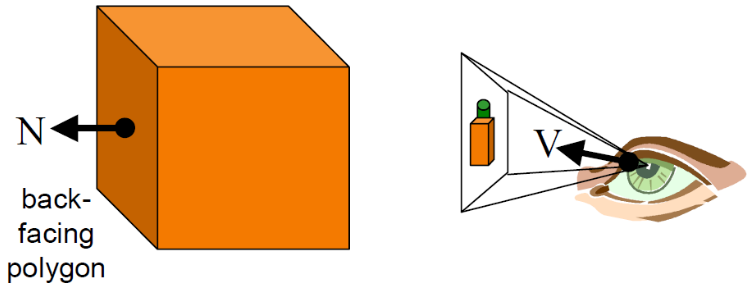


Basic algorithms for HSR

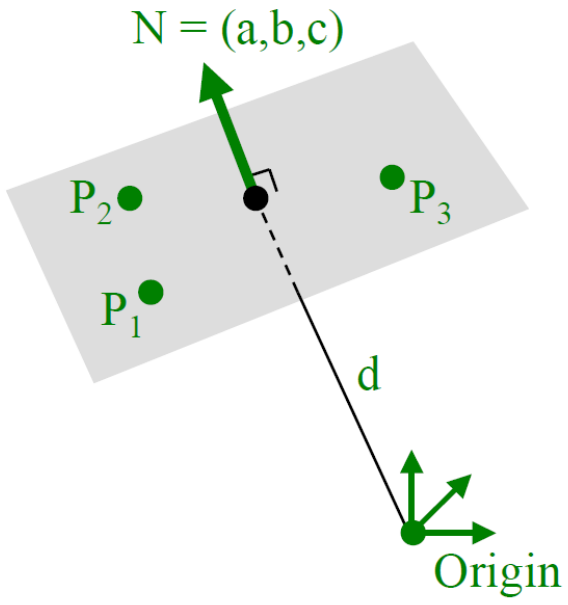
- Back-face culling
- Depth sort
- Z-Buffer

Back-face culling

- How do we test back-facing polygons ?
- Dot product the normal and view direction

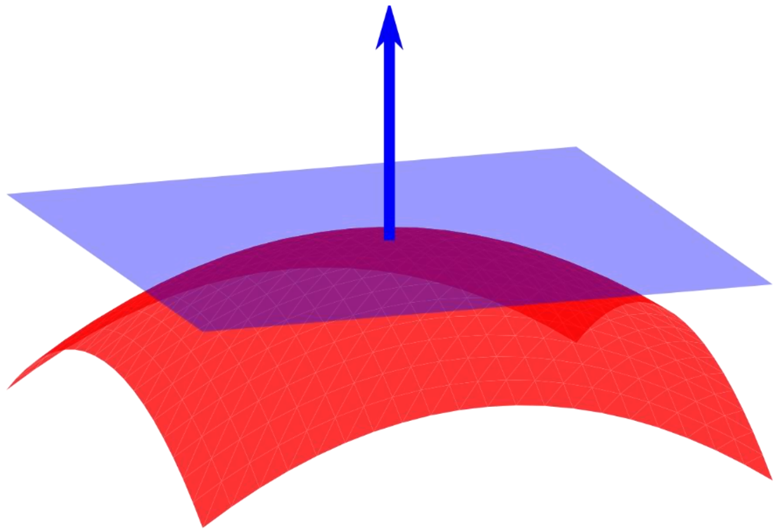


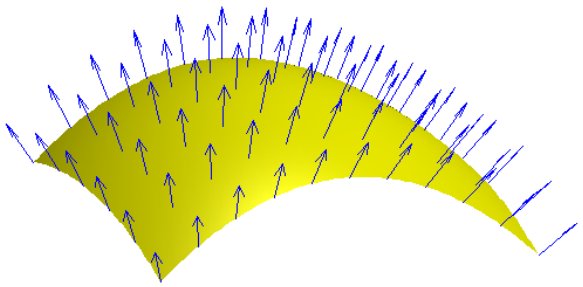
$N \cdot V > 0$

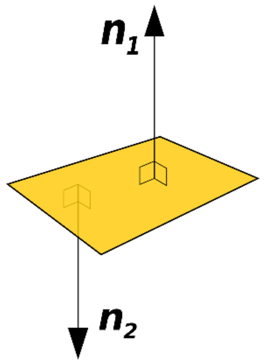


Surface Normals

- Normal
  - Cross product of surface tangent vectors
  - Length normalized to 1

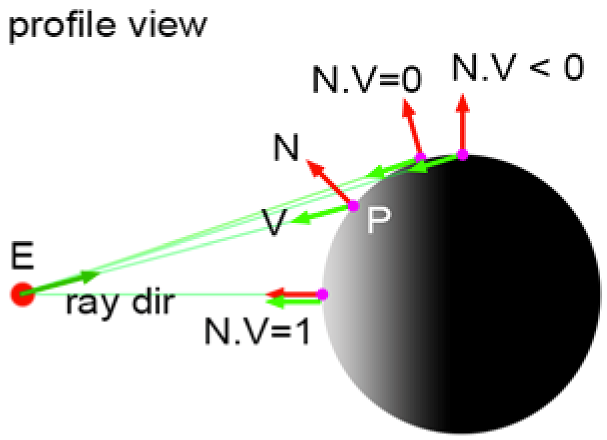


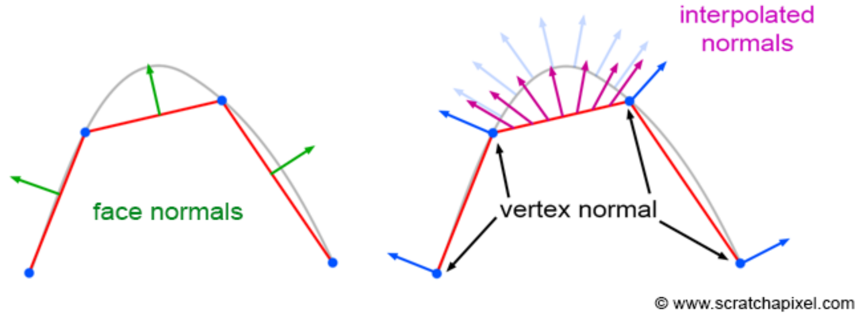




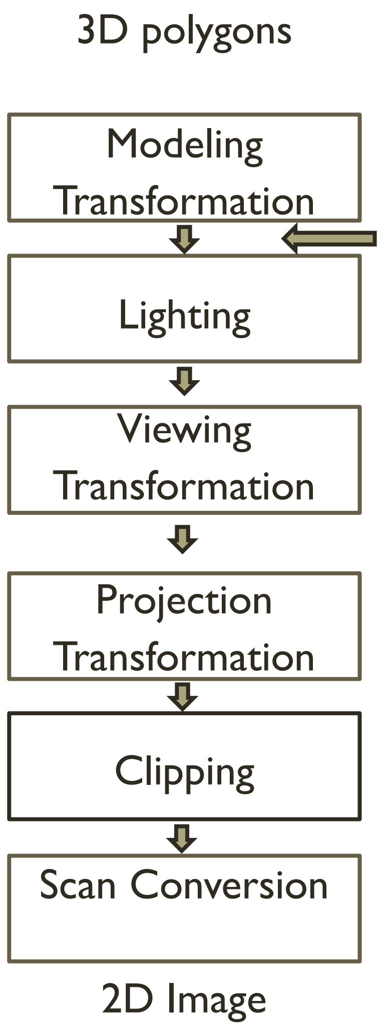
Vertex / Fragment Normals

- Dot product the normal and view direction
- Fragment normals can be interpolated from vertex normals

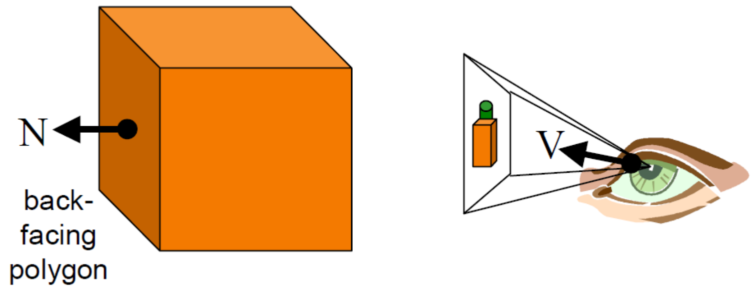




3D rendering pipeline

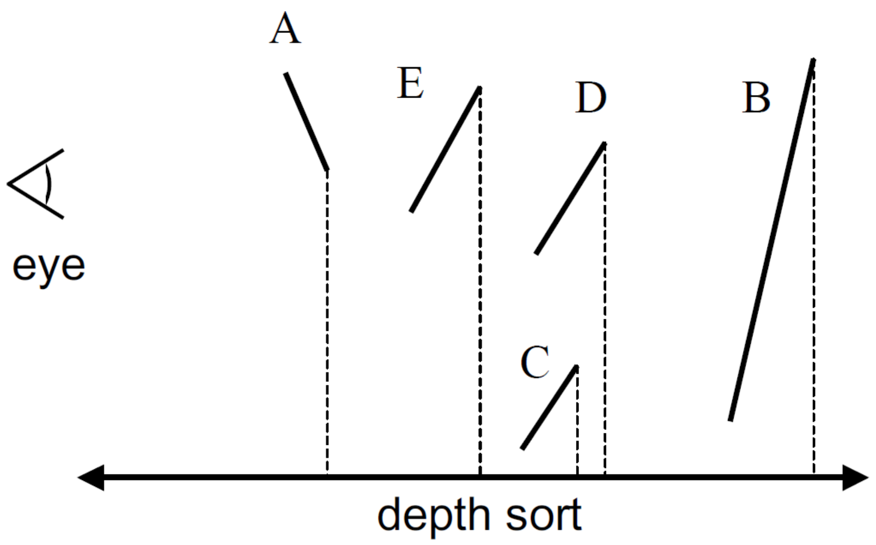


- Back-face culling
- Remove all polygons that are back-facing



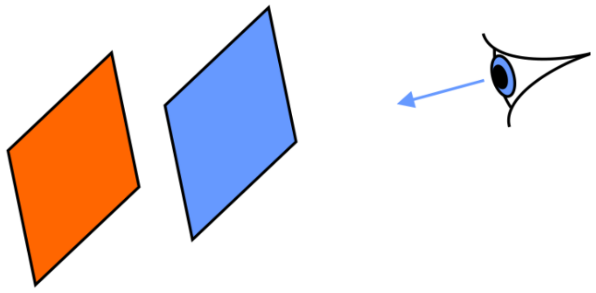
Depth sort

- "Painter's algorithm"
- Sort surfaces by maximum depth
- Draw surfaces in back to front order

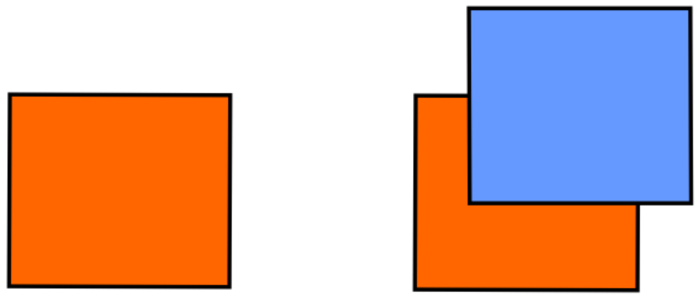


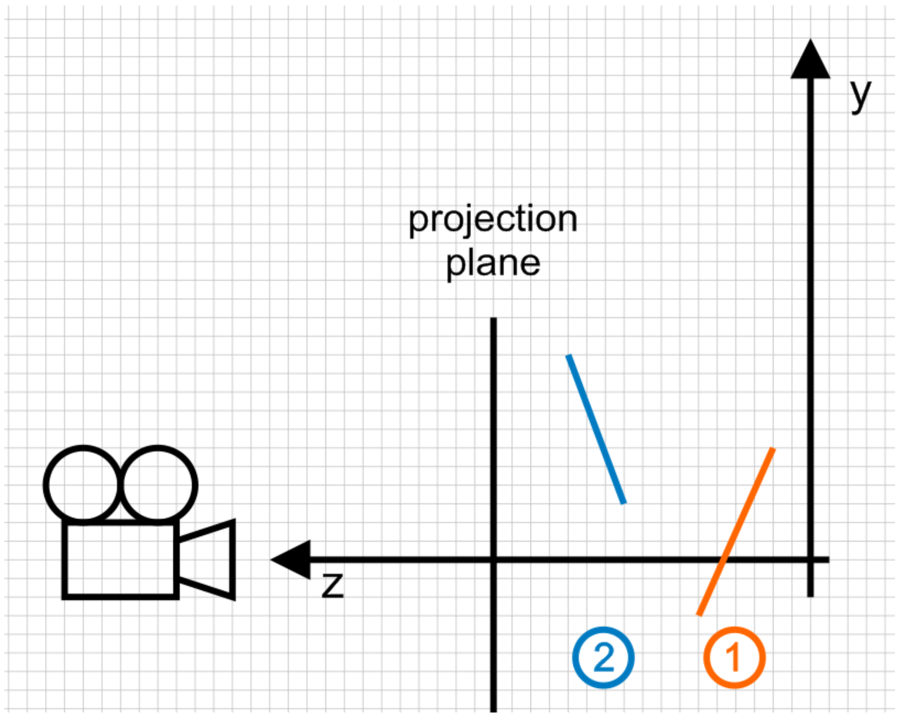
Painter's algorithm

- Sort faces in a back-to-front order, render



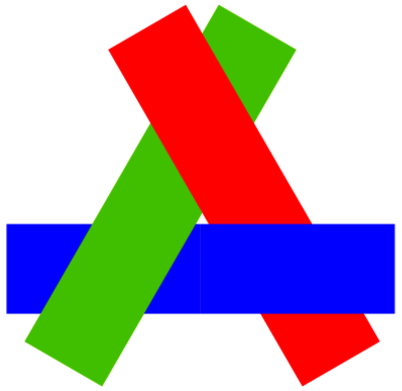
- New pixels over-write old pixels

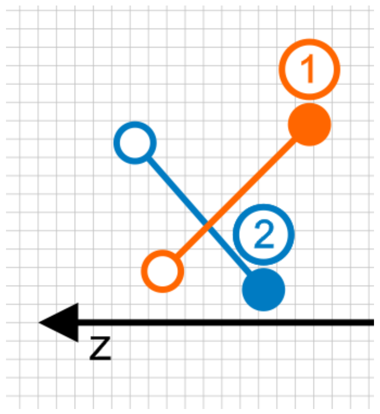




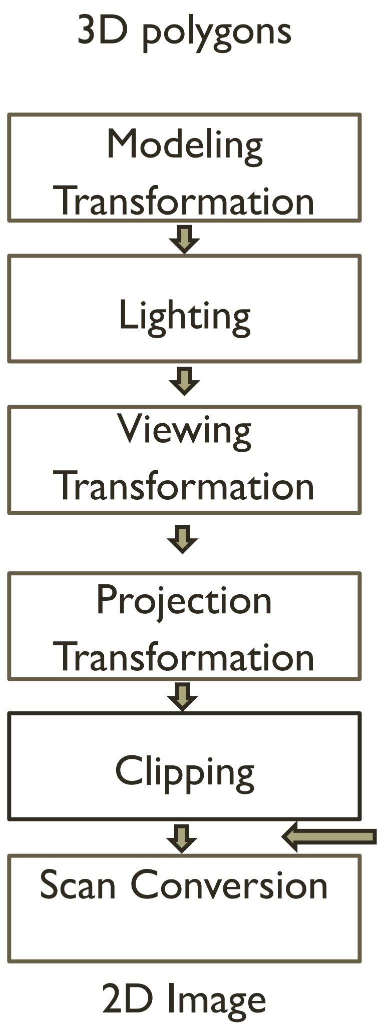
Painter's algorithm problems

- Intersecting faces
- Cyclically overlapping faces
- Redundant rendering

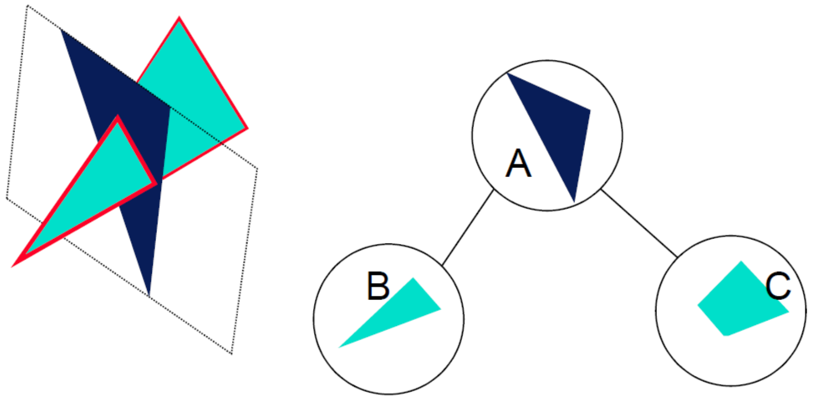




3D rendering pipeline



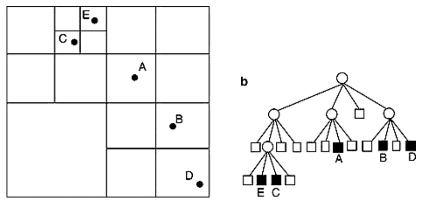
- Sorting is often O(n log n)
- Usually, software implementation only
- Mostly using BSP-trees



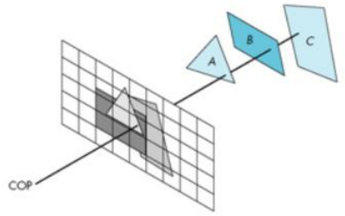
Depth sort

Other algorithms

- Warnock algorithm
  - subdivide screen into a quadtree until whole cell empty or whole cell inside polygons

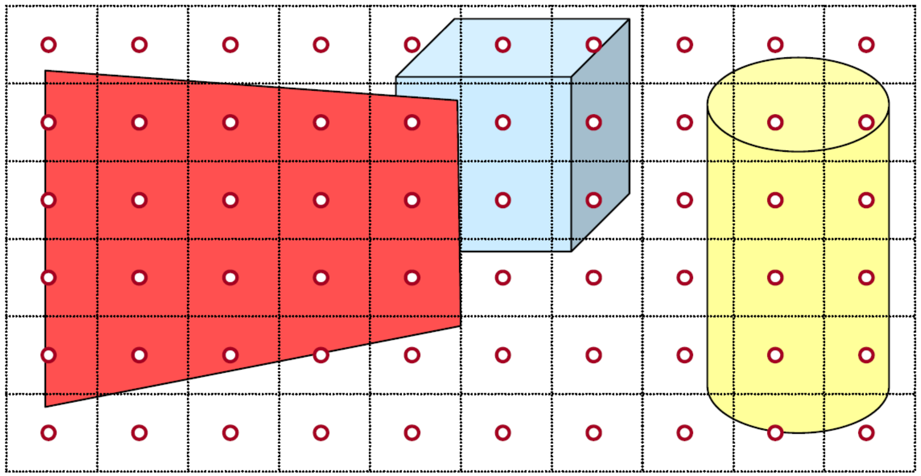


- Reversed painter's algorithm
  - paint front-to-back and paint only empty areas
- Z-buffer
  - remember z-value for each pixel and only paint when new z is higher



Z-Buffer

- Also known as depth buffering
- Stores closest depth of objects for every pixel
  - Draw only pixels with less depth
  - Depths are interpolated between vertices



Z-Buffer

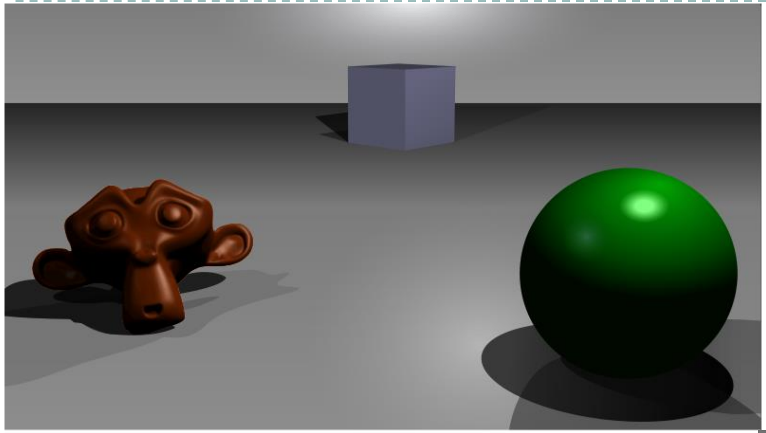
- works in screen space
- z-buffer w×h

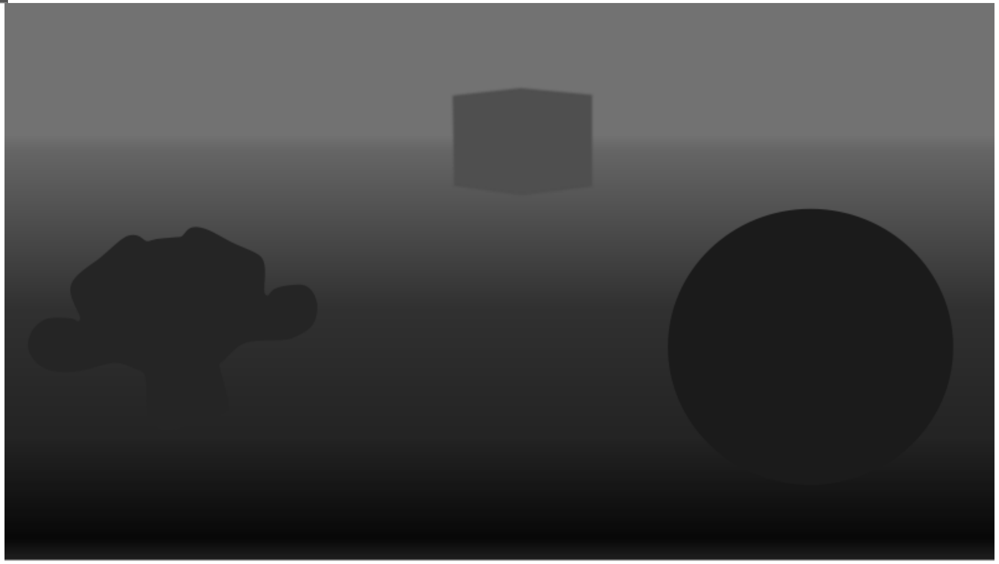
```
for each $0 \le x \le w, 0 \le y \le h$: z-buffer[x,y] $\leftarrow z_{max}$
for each face:
    rasterize it into pixels {x,y,z}
        for each face's pixel (x,y,z):
            if z < z-buffer[x,y]
            then :
                z-buffer[x,y] $\leftarrow$ z
                and screen[x,y] $\leftarrow$ color
```

Z-buffer pros and cons

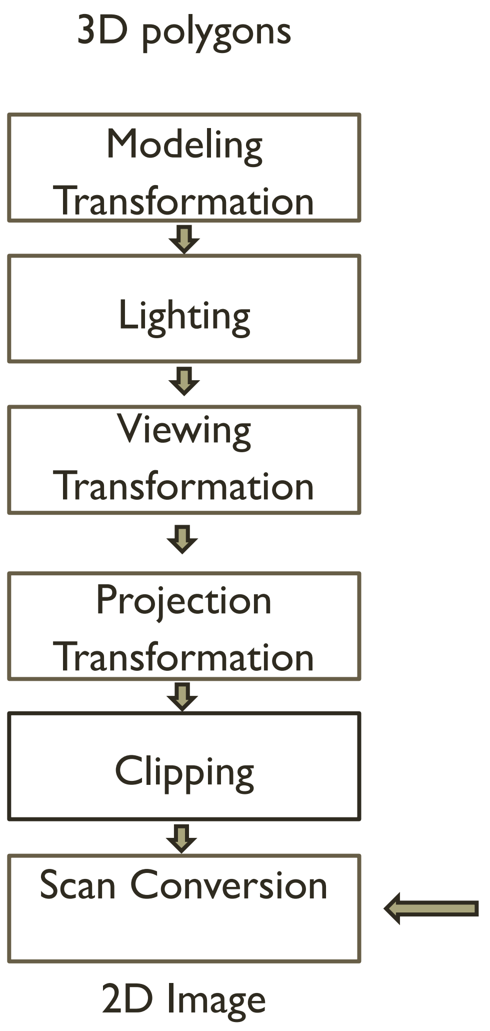
- GPU support
- precision issues might occur
- z-buffer test before per-pixel-lighting or pixel shading saves a lot of redundant work
- memory demands (width×height×precision)
  - can be reduced by scanline (width×1×precision)

Z-Buffer


A simple three-dimensional scene


Z-buffer representation

3D rendering pipeline



- Sorting not needed
- Excellent for hardware
- Requires additional memory to store the depth values
- Subject to aliasing

- Z-Buffer

Visibility

- Can be solved in different ways
  - Painter's algorithm / Depth sort
    - Binary space partitioning (BSP)
    - Warnock algorithm (Quadtree)
  - Z-buffering
  - Raycasting / Raytracing

Culling

- Viewing-frustum culling
- Back-face culling
- Contribution culling (LoD)
- Occlusion culling
  - Potentially visible set (PVS)
  - Portal rendering

Next Lecture

Textures and Mappings

Acknowledgements

- Thanks to all the people, whose work is shown here and whose slides were used as a material for creation of these slides:

	Matej Novotný, GSVM lectures at FMFI UK
	Peter Drahoš, PPGSO lectures at FIIT STU
	Output of all the publications and great team work
	Very best data from 3D cameras

Questions ?!

Skeletex
RESEARCH

www.skeletex.xyz

madaras@skeletex.xyz

martin.madaras@fmph.uniba.sk