3D Polygon Rendering Pipeline

Michael Kazhdan

(601.457/657)

3D Polygon Rendering

- Many applications require *interactive* rendering of 3D polygons with direct illumination

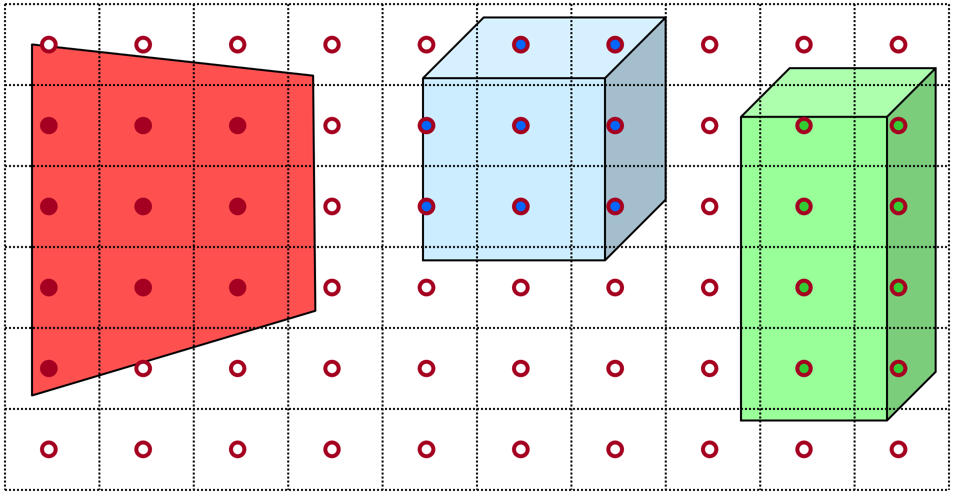
God of War
(Santa Monica Studio, 2018)

Ray Casting

- For each sample:
  - Construct ray from the camera into the scene
  - Find first surface intersected by ray through pixel
  - Compute color of sample based on surface radiance

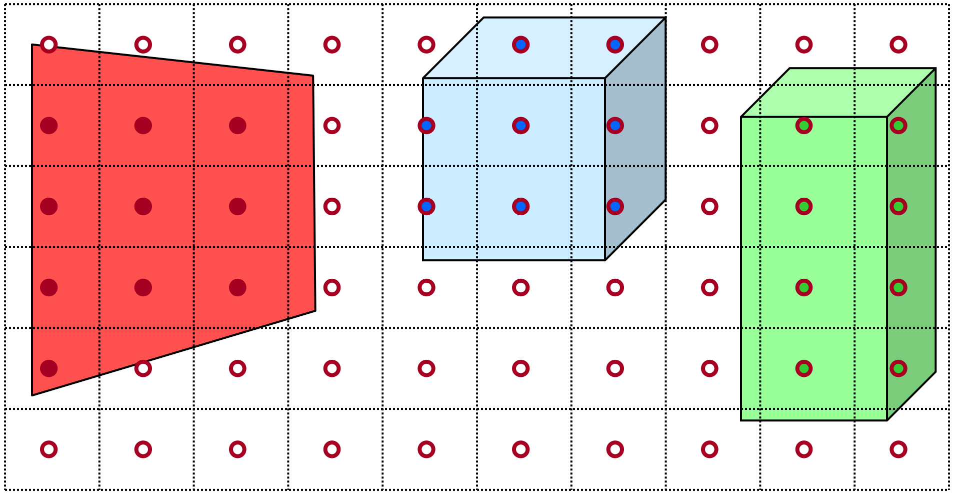
$\Downarrow$

  - Send 2D pixels into the scene and get color



3D Polygon Rendering

- For each primitive:
  - Send 3D points to the camera and set the pixel color



3D Rendering Pipeline (for direct illumination)

3D Primitives

↓ 3D Modeling Coordinates

Modeling Transformation

↓ 3D World Coordinates

Viewing Transformation

↓ 3D Camera Coordinates

Lighting

↓ 3D Camera Coordinates

Projection Transformation

↓ 2D Window Coordinates

Clipping

↓ 2D Window Coordinates

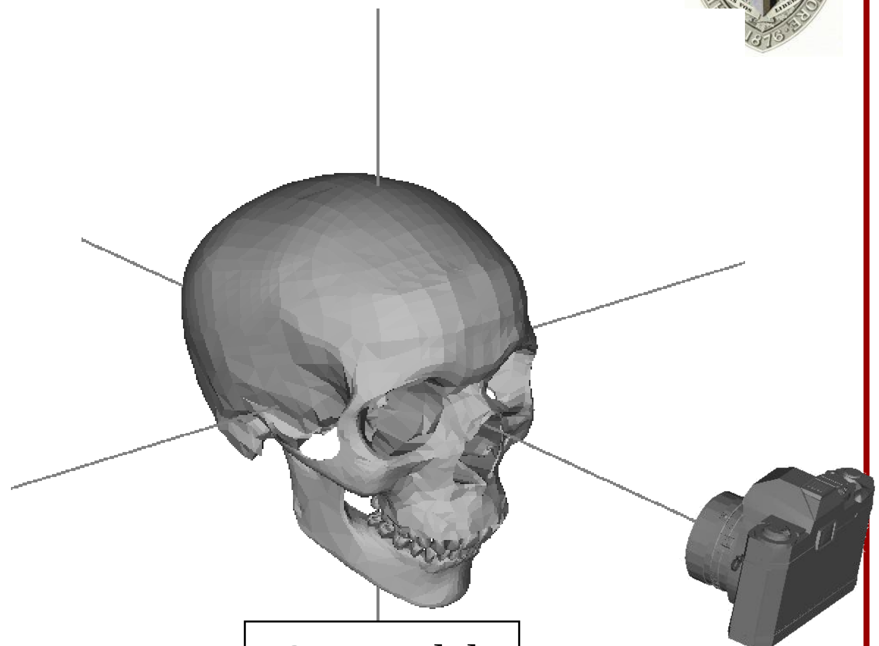
Viewport Transformation

↓ 2D Viewport Coordinates

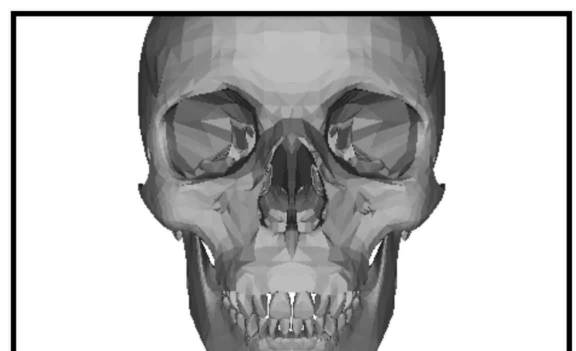
Scan Conversion

↓ 2D Viewport Coordinates

Image

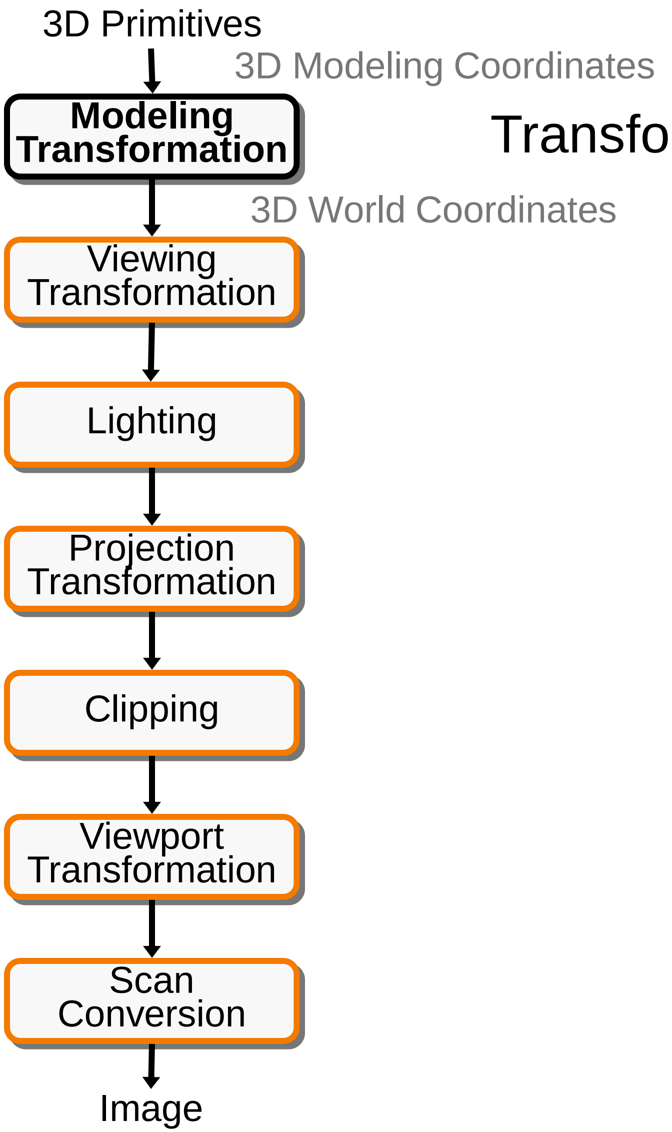


3D Model



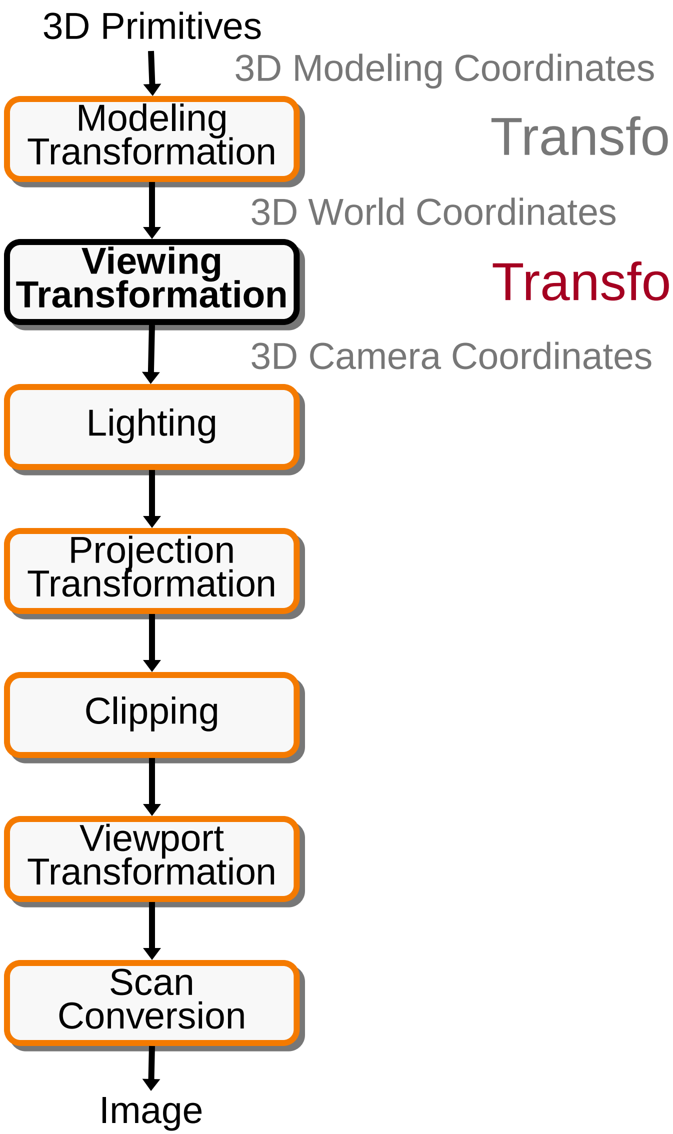
2D Viewport

3D Rendering Pipeline (for direct illumination)



Transform into 3D world coordinate system

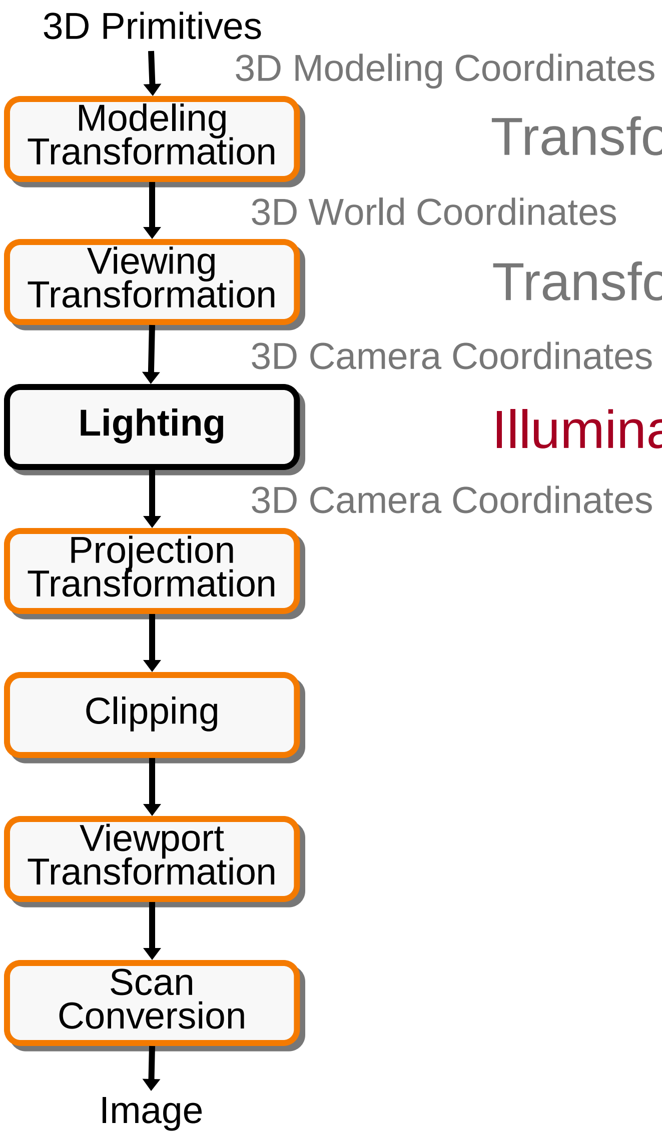
3D Rendering Pipeline (for direct illumination)



Transform into 3D world coordinate system

Transform into 3D camera coordinate system

3D Rendering Pipeline (for direct illumination)



Transform into 3D world coordinate system

Transform into 3D camera coordinate system

Illuminate **vertices** using lighting and reflectance

3D Rendering Pipeline (for direct illumination)

3D Primitives

↓ 3D Modeling Coordinates

Modeling Transformation — Transform into 3D world coordinate system

↓ 3D World Coordinates

Viewing Transformation — Transform into 3D camera coordinate system

↓ 3D Camera Coordinates

Lighting — Illuminate vertices using lighting and reflectance

↓ 3D Camera Coordinates

Projection Transformation — Transform into 2D window coordinate system

↓ 2D Window Coordinates

Clipping

↓

Viewport Transformation

↓

Scan Conversion

↓

Image

3D Rendering Pipeline (for direct illumination)

3D Primitives

↓ 3D Modeling Coordinates

Modeling Transformation — Transform into 3D world coordinate system

↓ 3D World Coordinates

Viewing Transformation — Transform into 3D camera coordinate system

↓ 3D Camera Coordinates

Lighting — Illuminate vertices using lighting and reflectance

↓ 3D Camera Coordinates

Projection Transformation — Transform into 2D window coordinate system

↓ 2D Window Coordinates

Clipping — Clip (parts of) primitives outside camera's view

↓ 2D Window Coordinates

Viewport Transformation

↓

Scan Conversion

↓

Image

3D Rendering Pipeline (for direct illumination)

3D Primitives

↓ 3D Modeling Coordinates

Modeling Transformation — Transform into 3D world coordinate system

↓ 3D World Coordinates

Viewing Transformation — Transform into 3D camera coordinate system

↓ 3D Camera Coordinates

Lighting — Illuminate vertices using lighting and reflectance

↓ 3D Camera Coordinates

Projection Transformation — Transform into 2D window coordinate system

↓ 2D Window Coordinates

Clipping — Clip (parts of) primitives outside camera’s view

↓ 2D Window Coordinates

Viewport Transformation — Transform into 2D viewport coordinate system

↓ 2D Viewport Coordinates

Scan Conversion

↓

Image

3D Rendering Pipeline (for direct illumination)

3D Primitives	3D Modeling Coordinates	
↓ Modeling Transformation		Transform into 3D world coordinate system
↓	3D World Coordinates	
Viewing Transformation		Transform into 3D camera coordinate system
↓	3D Camera Coordinates	
Lighting		Illuminate vertices using lighting and reflectance
↓	3D Camera Coordinates	
Projection Transformation		Transform into 2D window coordinate system
↓	2D Window Coordinates	
Clipping		Clip (parts of) primitives outside camera's view
↓	2D Window Coordinates	
Viewport Transformation		Transform into 2D viewport coordinate system
↓	2D Viewport Coordinates	
Scan Conversion		Draw pixels (includes texturing, hidden surface, etc.)
↓	2D Viewport Coordinates	
Image		

Transformations

3D Primitives

3D Modeling Coordinates

Modeling Transformation — **Transform** into 3D world coordinate system

3D World Coordinates

Viewing Transformation — **Transform** into 3D camera coordinate system

3D Camera Coordinates

Lighting — Illuminate vertices using lighting and reflectance

3D Camera Coordinates

Projection Transformation — **Transform** into 2D window coordinate system

2D Window Coordinates

Clipping — Clip (parts of) primitives outside camera's view

2D Window Coordinates

Viewport Transformation — **Transform** into 2D viewport coordinate system

2D Viewport Coordinates

Scan Conversion — Draw pixels (includes texturing, hidden surface, etc.)

2D Viewport Coordinates

Image

Recall: Homogeneous Coordinates

- Add a 4th coordinate to every 3D point
  - (x, y, z, w) represents a point at location $\left(\frac{x}{w}, \frac{y}{w}, \frac{z}{w}\right)$
  - $(x, y, z, 0)$ represents the (unsigned) direction $\frac{\pm(x,y,z)}{\sqrt{x^2+y^2+z^2}}$
  - $(0, 0, 0,0)$ is not allowed

Recall: 3D Transformations

- Using homogenous coordinates, we have two types of transformations:
  - Affine

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

  - Projective

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

Transformations

(x, y, z)

↓ 3D Object Coordinates

Modeling Transformation

↓ 3D World Coordinates

Viewing Transformation

↓ 3D Camera Coordinates

Projection Transformation

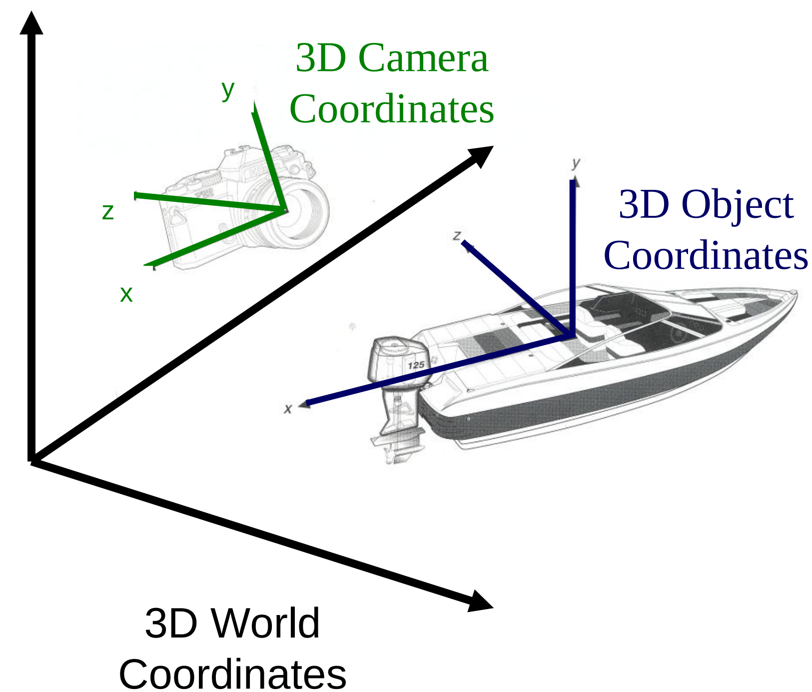
↓ 2D Window Coordinates

Window-to-Viewport Transformation

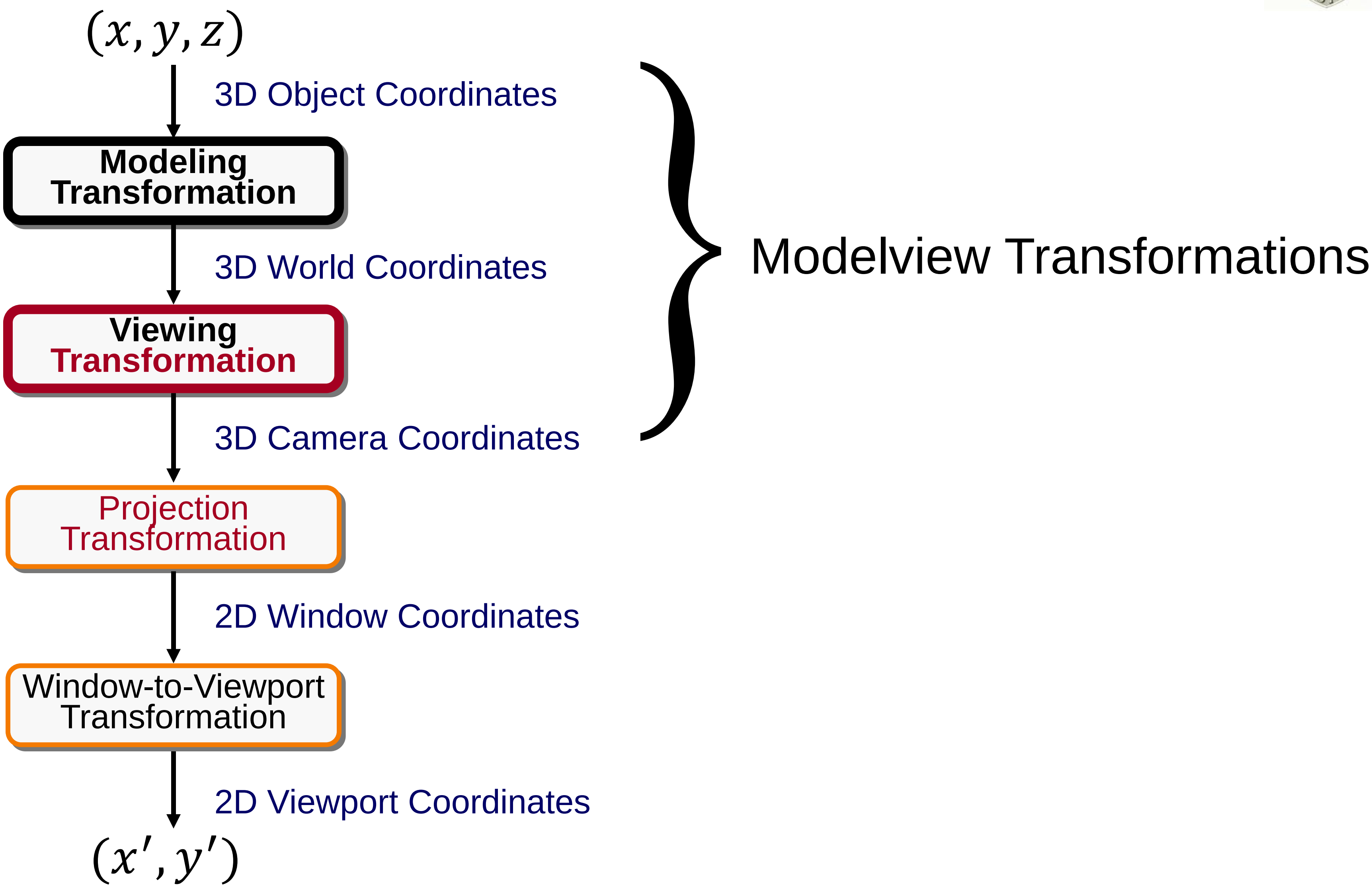
↓ 2D Viewport Coordinates

(x', y')

Transformations map points from one coordinate system to another



Transformations



Transformations

(x, y, z)

3D Object Coordinates

Modeling Transformation

3D World Coordinates

Viewing Transformation

3D Camera Coordinates

Projection Transformation

2D Window Coordinates

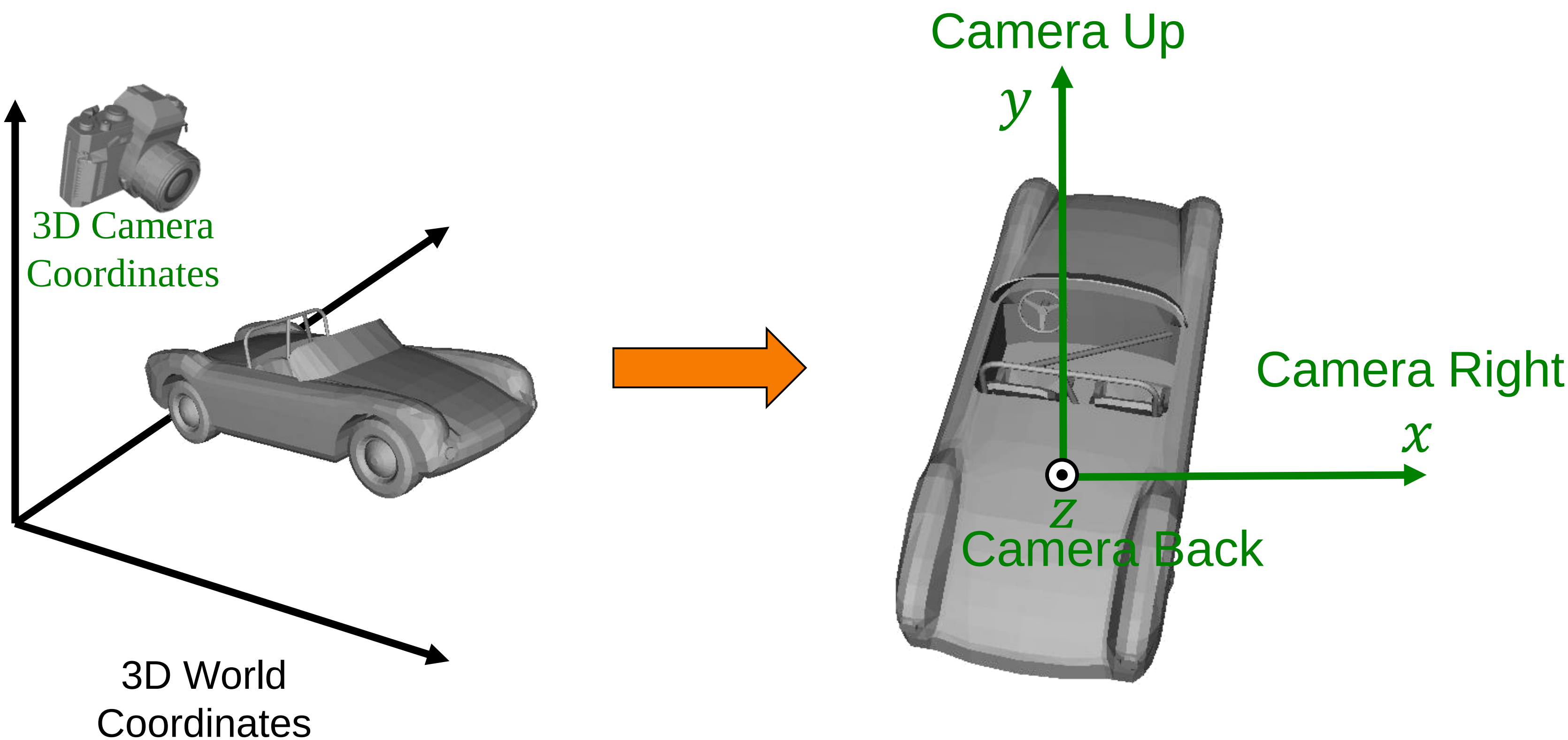
Window-to-Viewport Transformation

2D Viewport Coordinates

(x', y')

Viewing Transformation

- Canonical coordinate system
  - Convention is right-handed (looking down $-z$ axis)
  - Convenient for projection, clipping, etc.



Viewing Transformation

- The transformation, $\mathbf{T}_{W\to C}$, taking us from world coordinates to camera coordinates should map:
  - The right vector to the x-axis:
    $$\left(R_x, R_y, R_z, 0\right) \to (1,0,0,0)$$
  - The up vector to the y-axis:
    $$\left(U_x, U_y, U_z, 0\right) \to (0,1,0,0)$$
  - The back vector to the z-axis:
    $$\left(B_x, B_y, B_z, 0\right) \to (0,0,1,0)$$
  - The eye position to the origin:
    $$\left(E_x, E_y, E_z, 1\right) \to (0,0,0,1)$$

How should we define this transformation/matrix?

Viewing Transformation

- Consider the inverse transformation, $\mathbf{T}_{C\to W}$, taking us from camera coordinates to world coordinates:

$$\left(R_x, R_y, R_z, 0\right) \leftarrow (1,0,0,0)$$
$$\left(U_x, U_y, U_z, 0\right) \leftarrow (0,1,0,0)$$
$$\left(B_x, B_y, B_z, 0\right) \leftarrow (0,0,1,0)$$
$$\left(E_x, E_y, E_z, 1\right) \leftarrow (0,0,0,1)$$

- This is described by the matrix:

$$\begin{pmatrix} x^w \\ y^w \\ z^w \\ 1 \end{pmatrix} = \underbrace{\begin{pmatrix} R_x & U_x & B_x & E_x \\ R_y & U_y & B_y & E_y \\ R_z & U_z & B_z & E_z \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{\mathbf{T}_{C\to W}} \begin{pmatrix} x^c \\ y^c \\ z^c \\ 1 \end{pmatrix}$$

Finding the Viewing Transformation

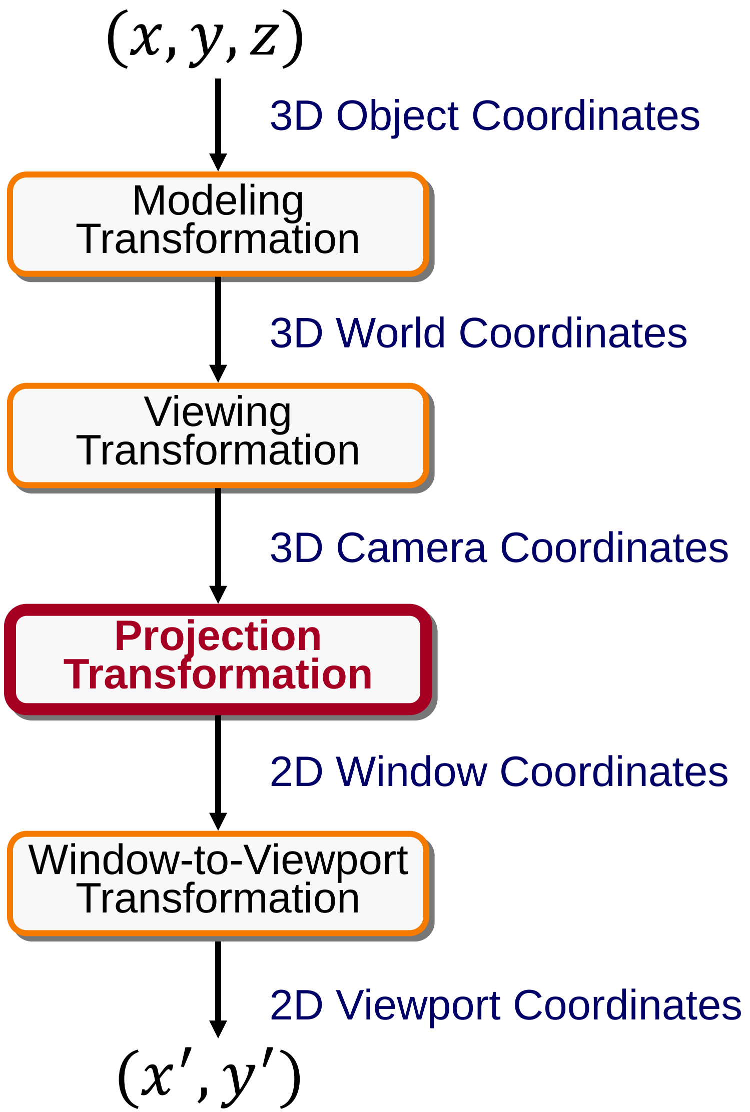
- The camera-to-world matrix:

$$\begin{pmatrix} x^w \\ y^w \\ z^w \\ 1 \end{pmatrix} = \underbrace{\begin{pmatrix} R_x & U_x & B_x & E_x \\ R_y & U_y & B_y & E_y \\ R_z & U_z & B_z & E_z \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{\mathbf{T}_{C\to W}} \begin{pmatrix} x^c \\ y^c \\ z^c \\ 1 \end{pmatrix}$$

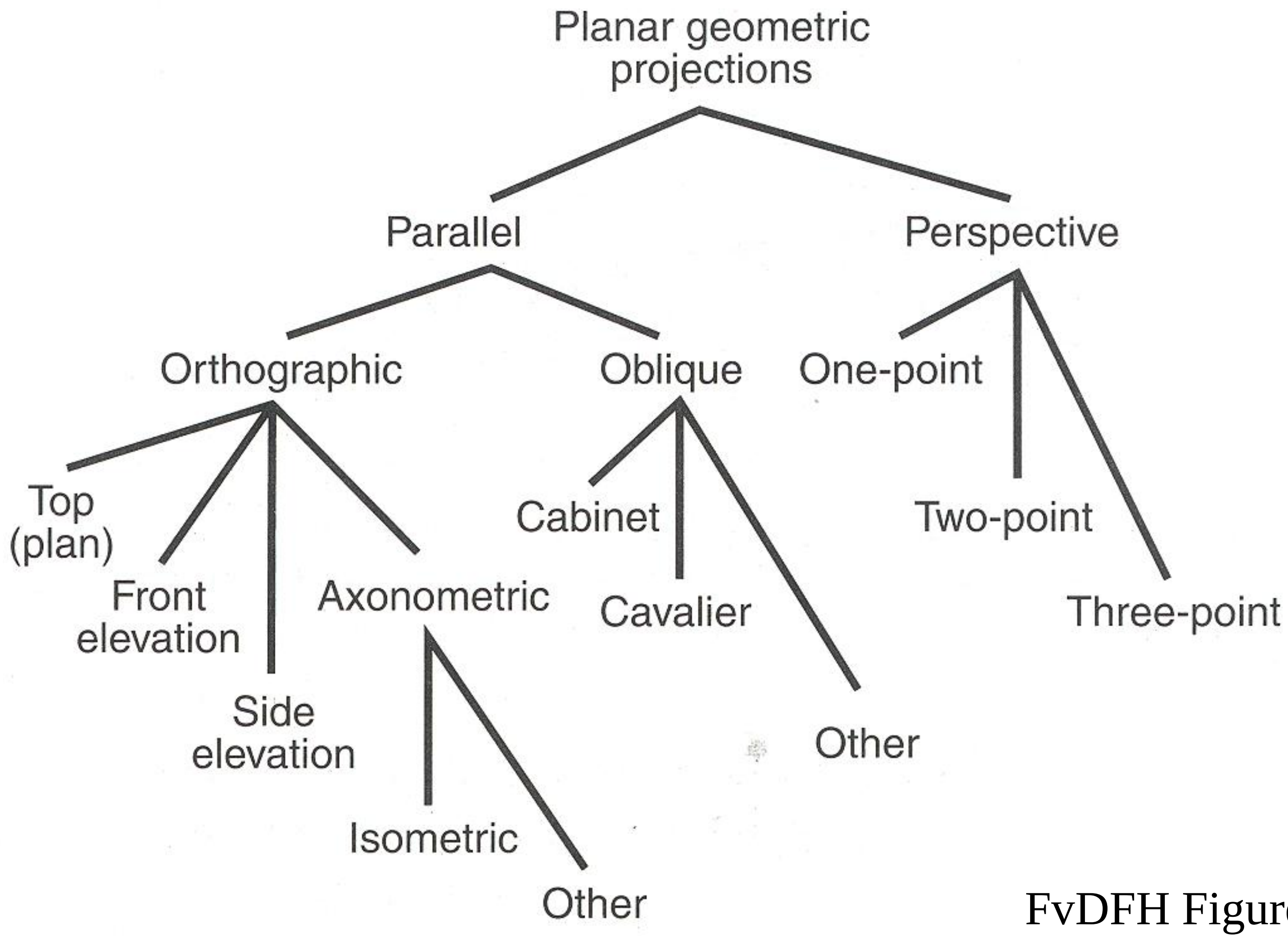
- The world-to-camera matrix is its inverse:

$$\begin{pmatrix} x^c \\ y^c \\ z^c \\ 1 \end{pmatrix} = \underbrace{\begin{pmatrix} R_x & U_x & B_x & E_x \\ R_y & U_y & B_y & E_y \\ R_z & U_z & B_z & E_z \\ 0 & 0 & 0 & 1 \end{pmatrix}^{-1}}_{\mathbf{T}_{W\to C} = \mathbf{T}_{C\to W}^{-1}} \begin{pmatrix} x^w \\ y^w \\ z^w \\ 1 \end{pmatrix}$$

Transformations



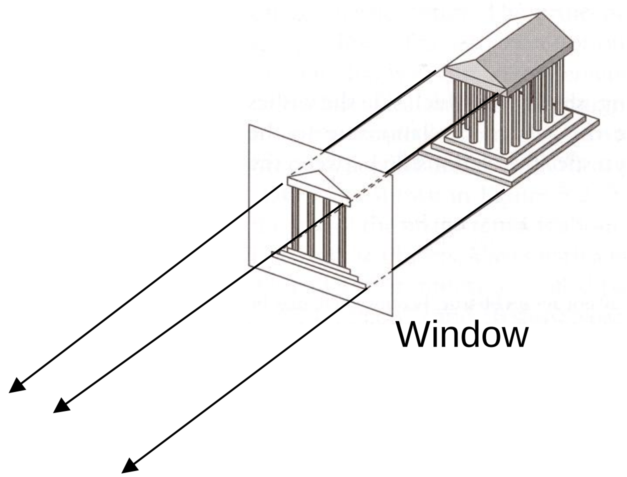
Taxonomy of Projections

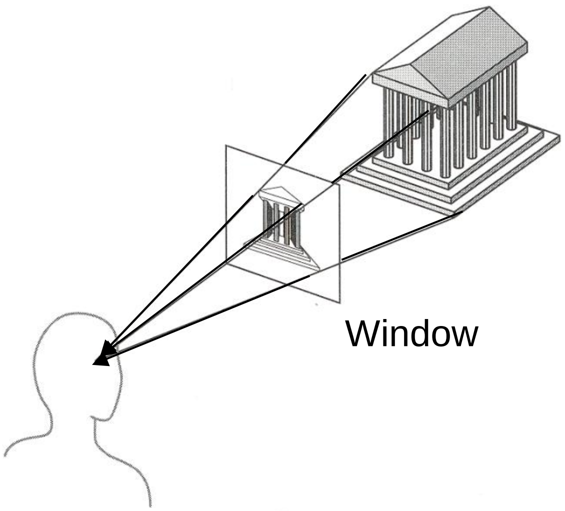


FvDFH Figure 6.13

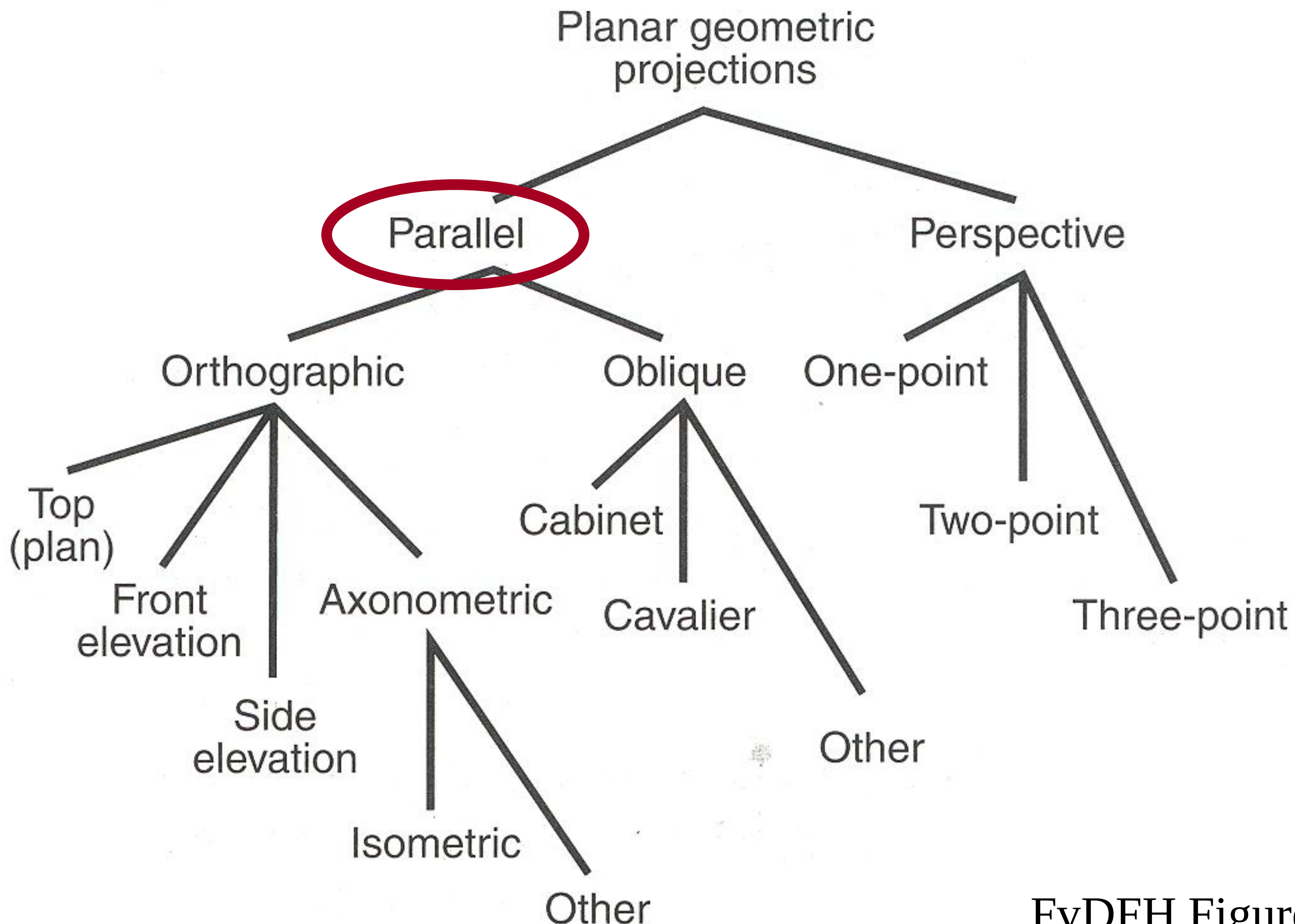
Projection

- Two general classes of projections, which shoot rays from the 3D scene, through the 2D window:
  - Parallel Projection:
    - » Rays converge at a point at infinity and are parallel
  - Perspective "Projection":
    - » Rays converge at a finite point, giving rise to perspective distortion





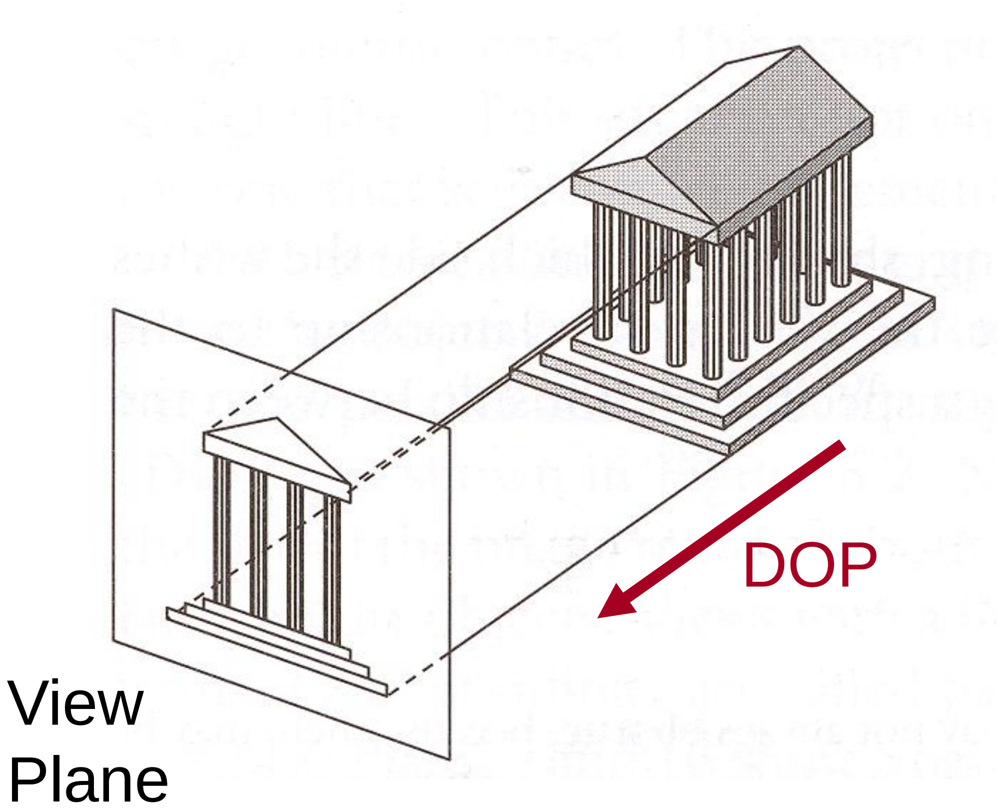
Taxonomy of Projections



FvDFH Figure 6.13

Parallel Projection

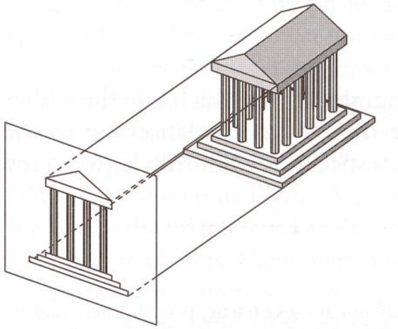
- Center of projection is at infinity
  - Direction of projection (DoP) same for all points



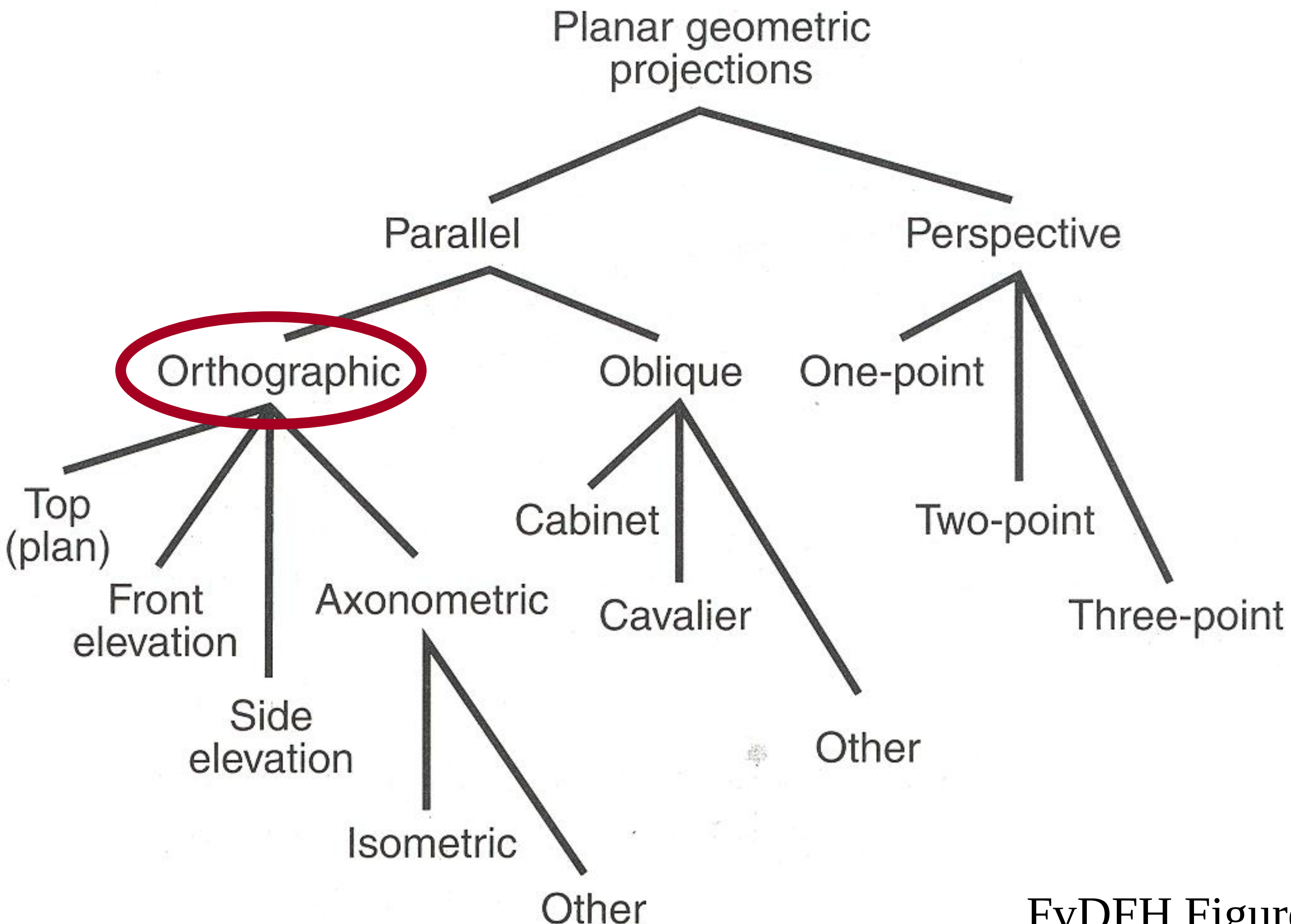
Angel Figure 5.4

Parallel Projection

- ✓ Parallel lines remain parallel
- ✓ Proportions are preserved (no foreshortening)
- ✗ (Some) angles are not preserved
- ✗ Less realistic looking



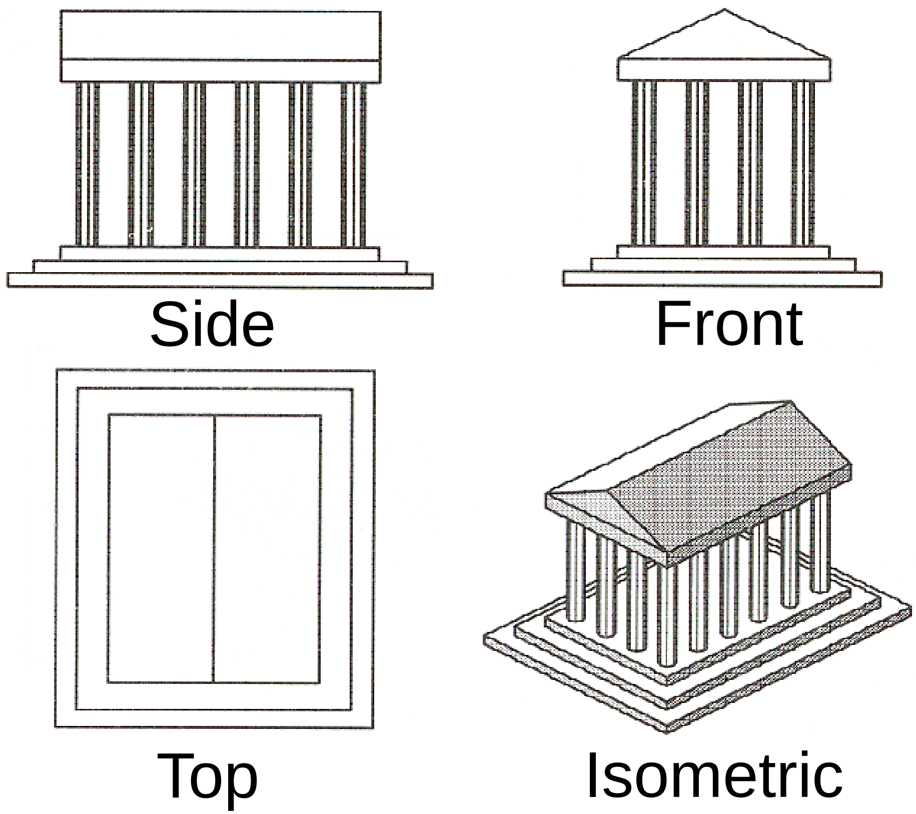
Taxonomy of Projections



FvDFH Figure 6.13

Orthographic Projections

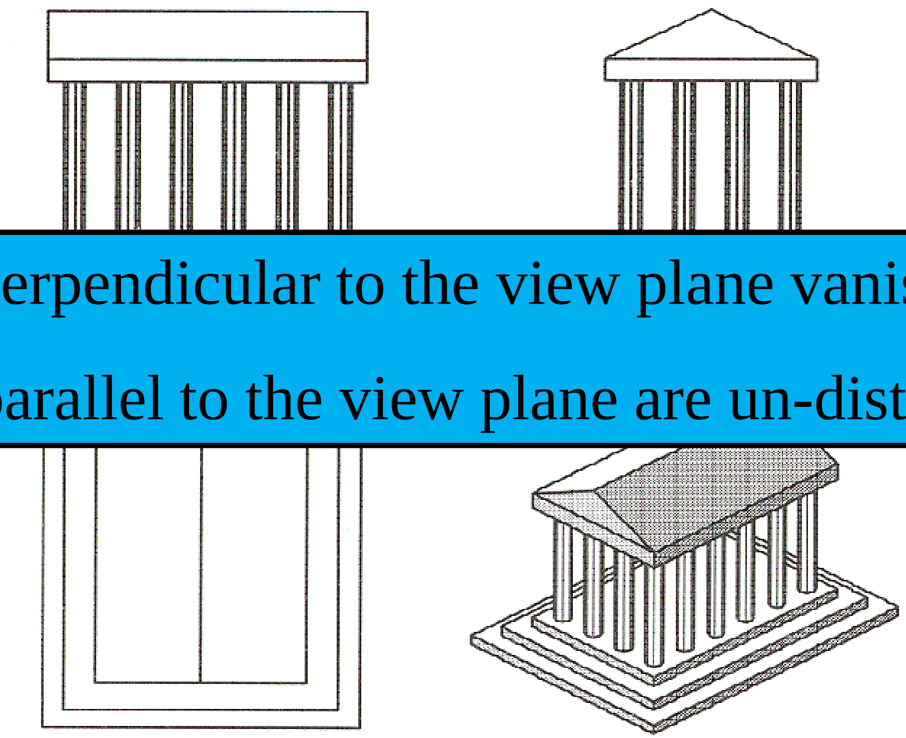
- DoP perpendicular to view plane



Angel Figure 5.5

Orthographic Projections

- DoP perpendicular to view plane



• Lines perpendicular to the view plane vanish

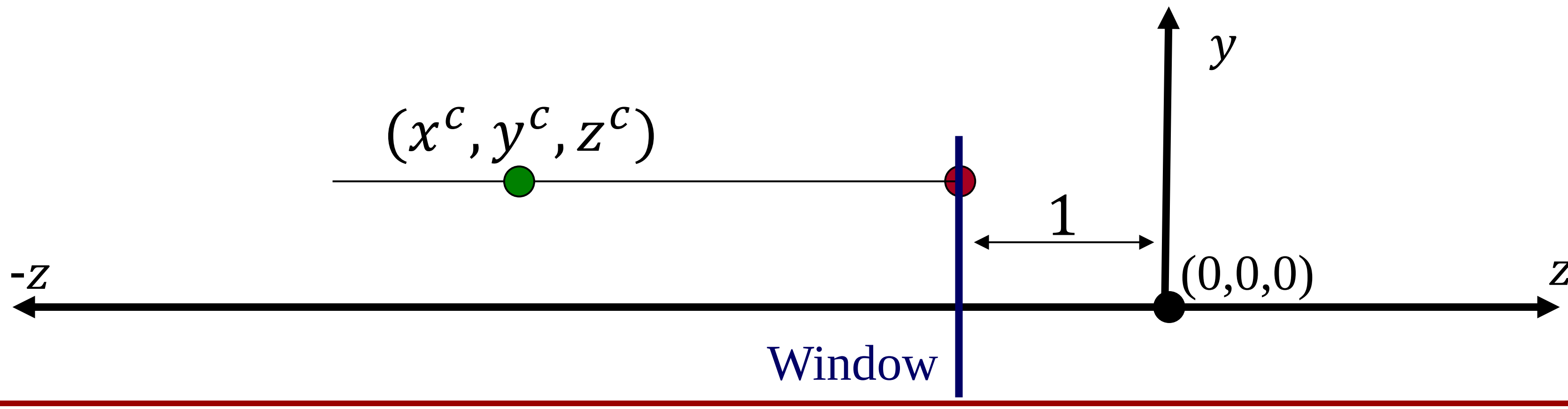
• Faces parallel to the view plane are un-distorted.

Top   Isometric

Angel Figure 5.5

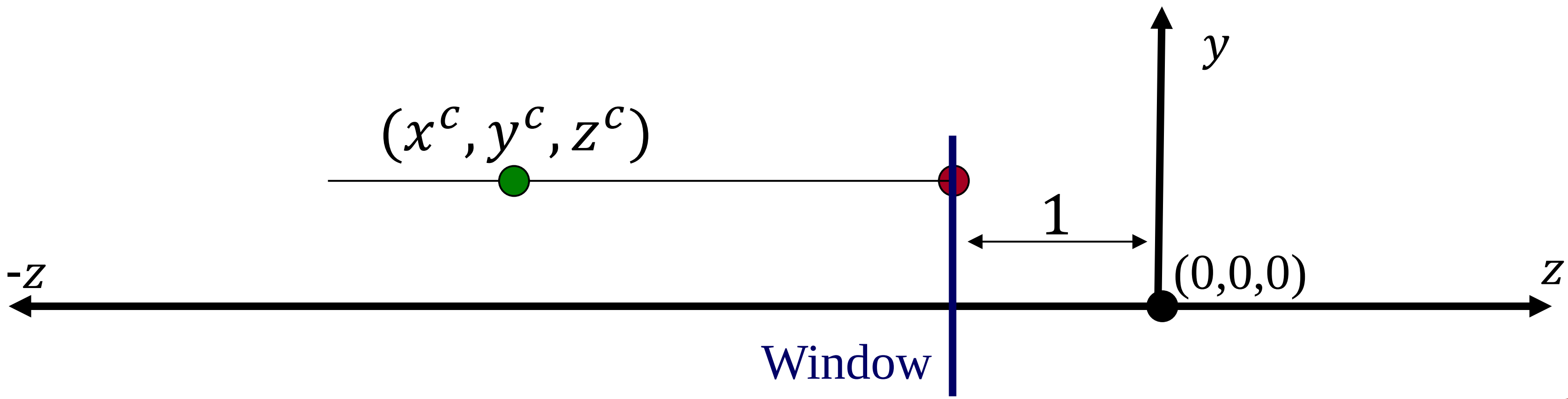
Orthographic Projections

- DoP perpendicular to view plane
  - Maps a point in 3D space to the $(x, y, -1)$-plane, by projecting out the z-component:
  $$(x^c, y^c, z^c) \to (x^c, y^c, -1)$$



Orthographic Projections

- DoP perpendicular to view plane
  - Maps a point in 3D space to the $(x, y, -1)$-plane, by projecting out the z-component:
$$(x^c, y^c, z^c, 1) \to (x^c, y^c, -1, 1)$$
  - In terms of homogenous coordinates:
$$\begin{bmatrix} x^c \\ y^c \\ -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x^c \\ y^c \\ z^c \\ 1 \end{bmatrix}$$

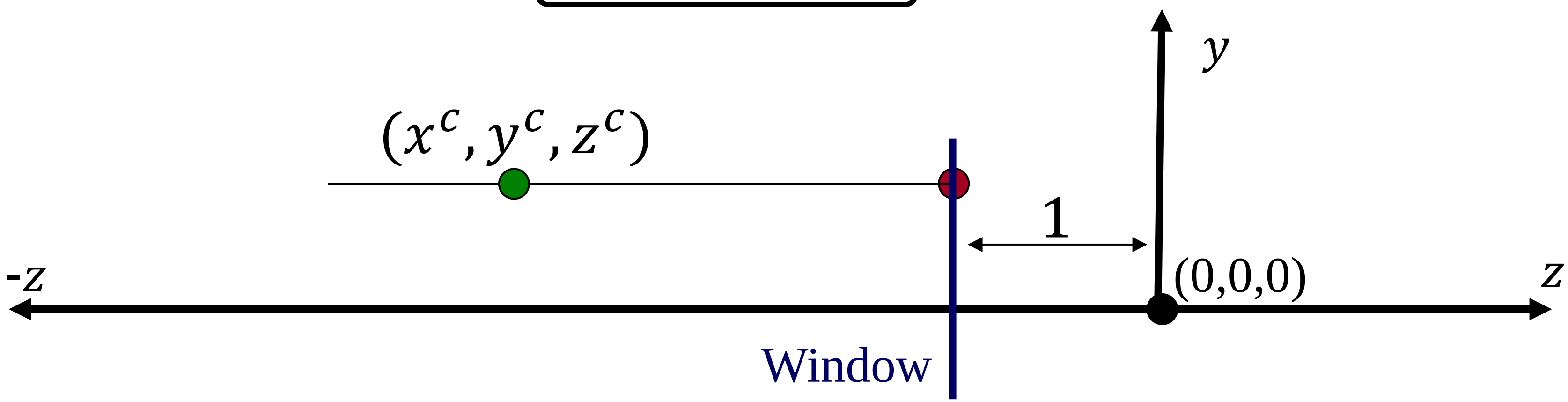


Orthographic Projections

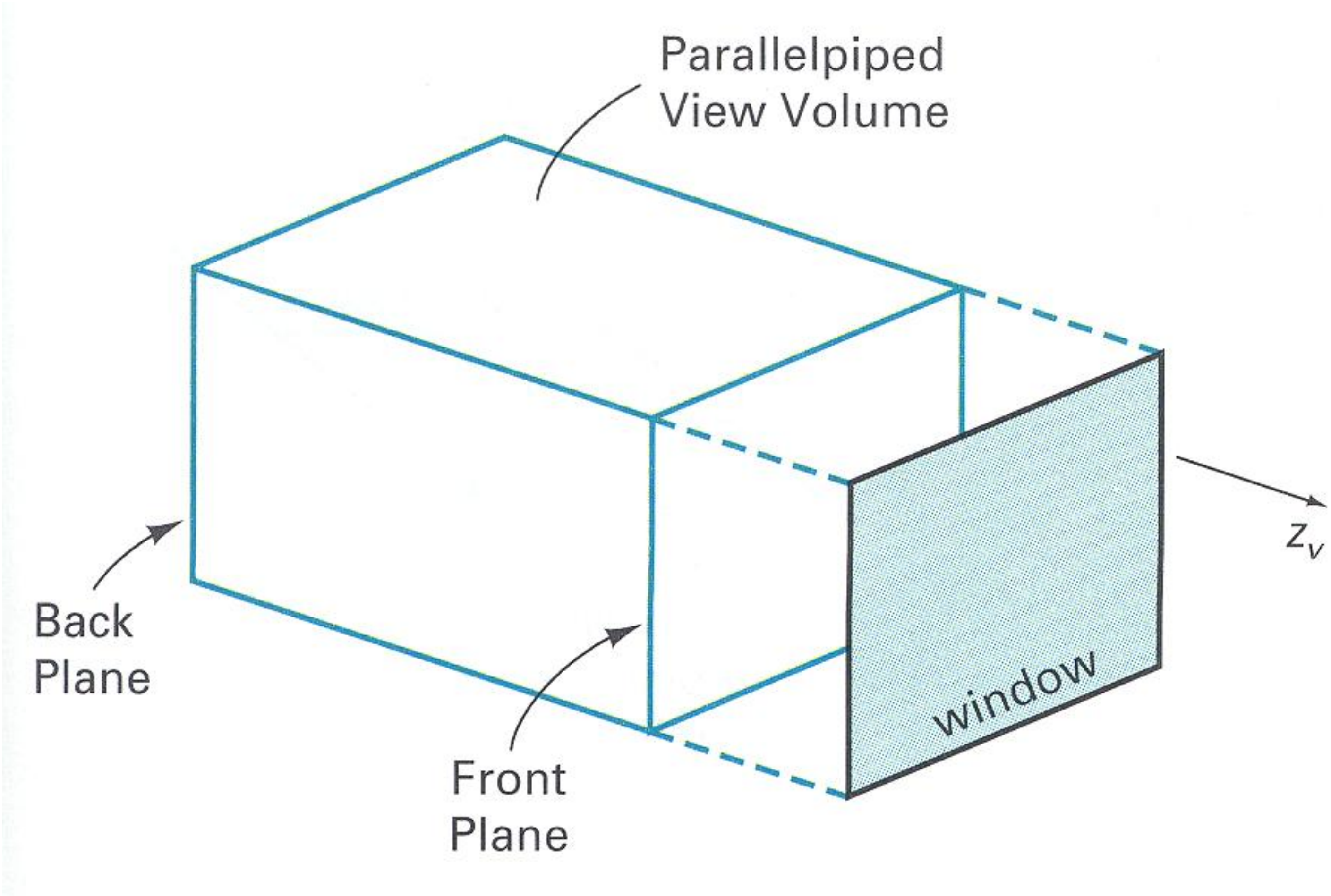
- DoP perpendicular to view plane
  - Maps a point in 3D space to the $(x, y, -1)$-plane, by projecting out the z-component:
    $$(x^c, y^c, z^c, 1) \rightarrow (x^c, y^c, -1, 1)$$

Note:
This matrix describes an affine transformation

$$\begin{bmatrix} -1 \\ 1 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} z^c \\ 1 \end{bmatrix}$$

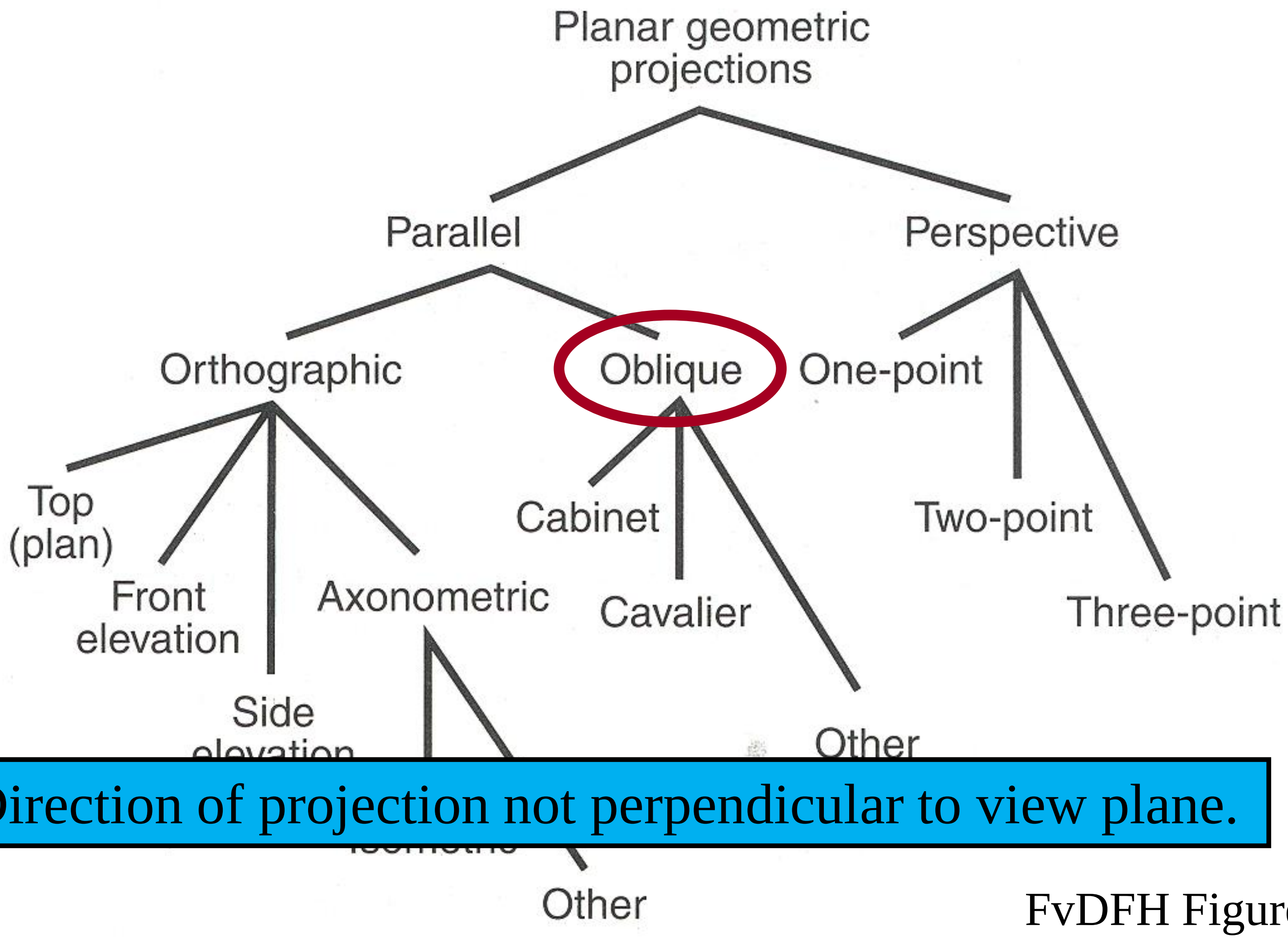


Parallel Projection View Volume



H&B Figure 12.30

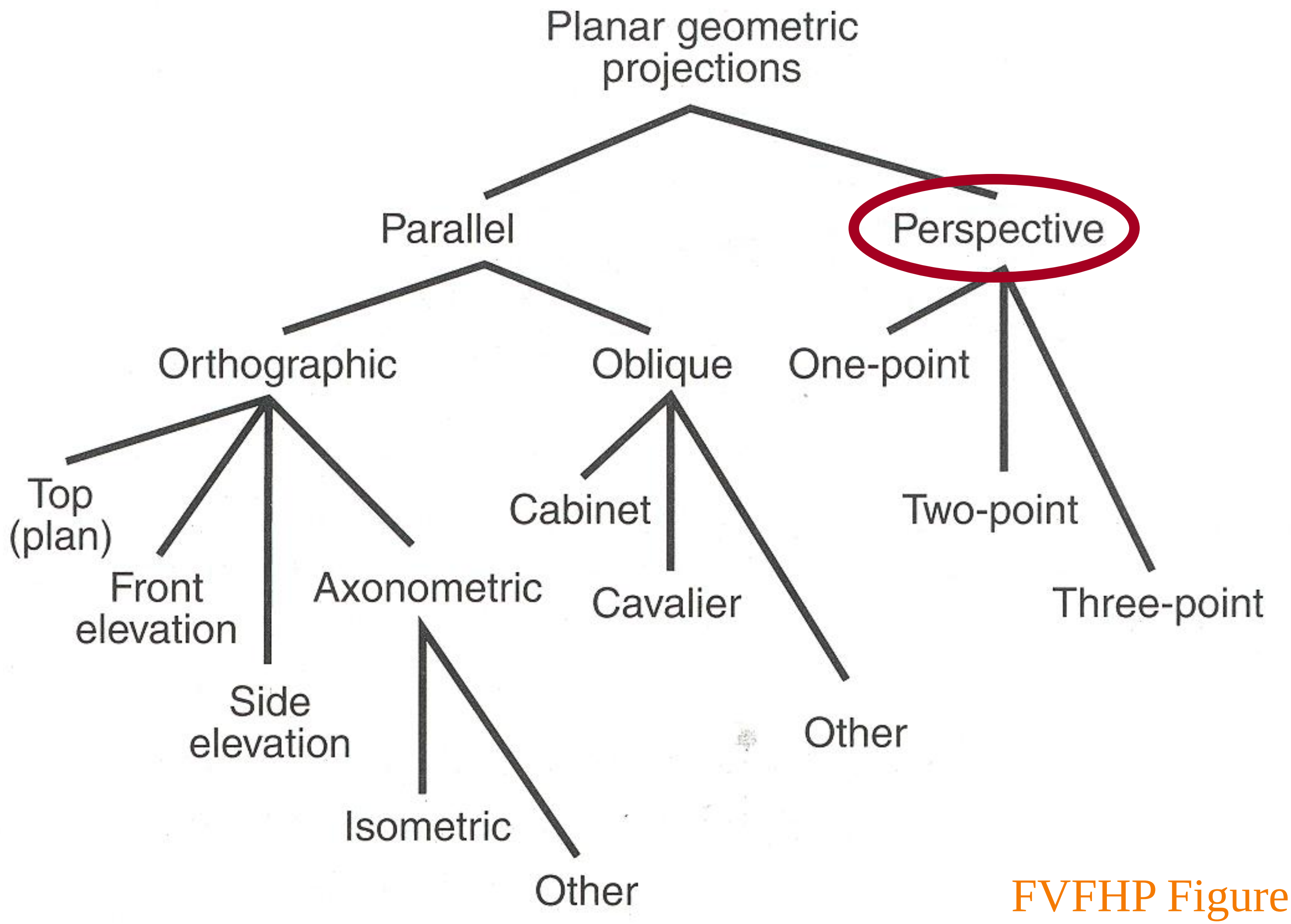
Taxonomy of Projections



Direction of projection not perpendicular to view plane.

FvDFH Figure 6.13

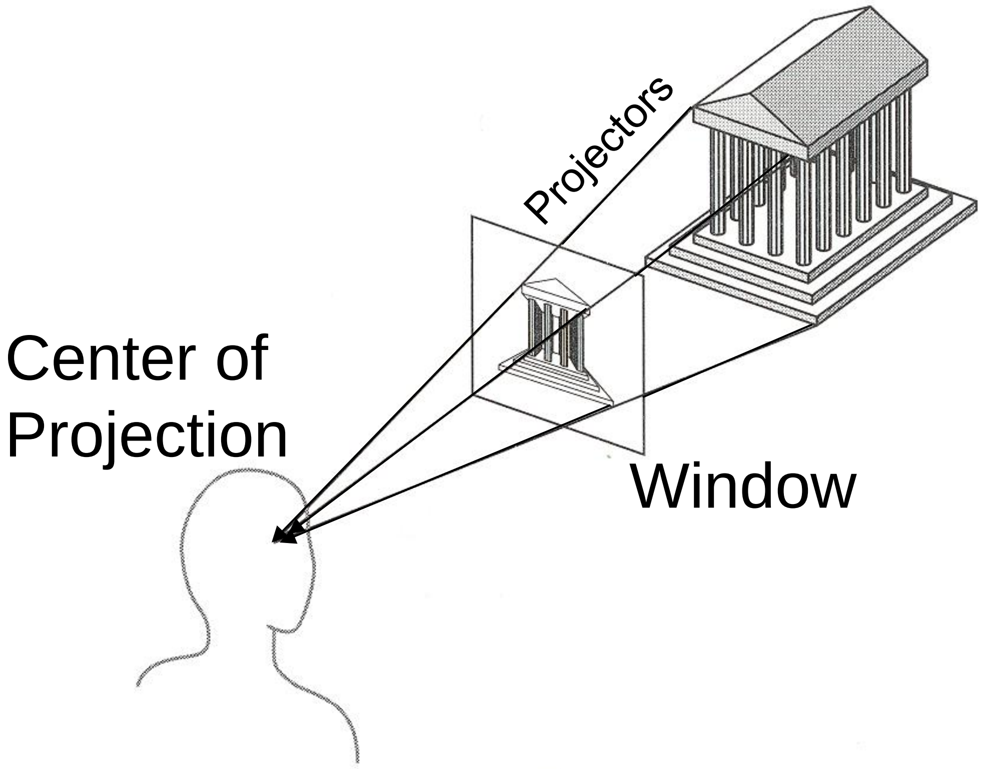
Taxonomy of Projections



FVFHP Figure 6.10

Perspective "Projection"

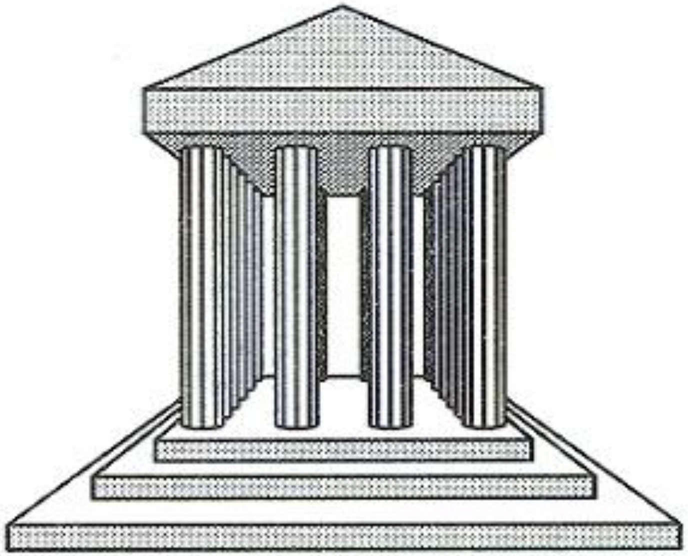
- Map points onto "view plane" along "projectors" emanating from "center of projection" (CoP)



Angel Figure 5.9

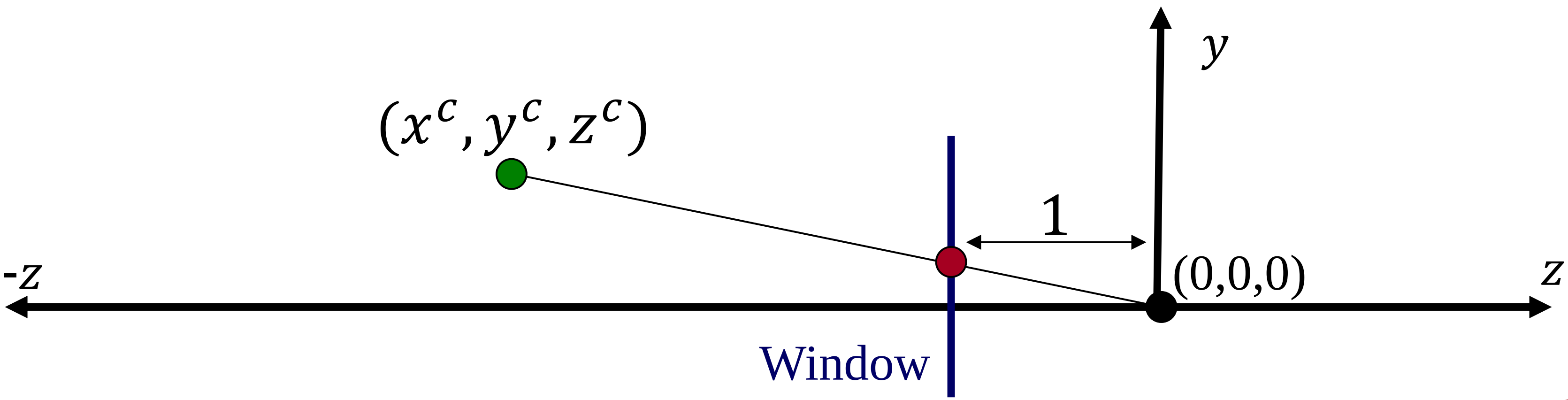
Perspective Projection

- Not all parallel lines remain parallel!



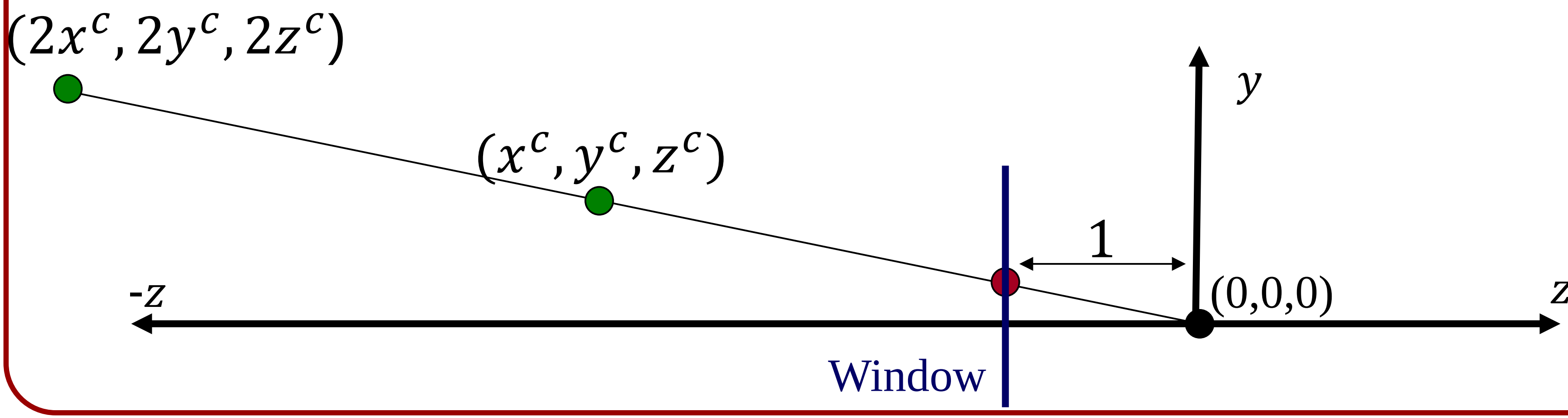
Perspective Projection

- What are the coordinates of the point resulting from projection of (x^c, y^c, z^c) onto the camera screen a unit distance back along the z-axis?



Perspective Projection

- For any point (x^c, y^c, z^c) and any scalar α, the points (x^c, y^c, z^c) and $(\alpha x^c, \alpha y^c, \alpha z^c)$ map to the same location.

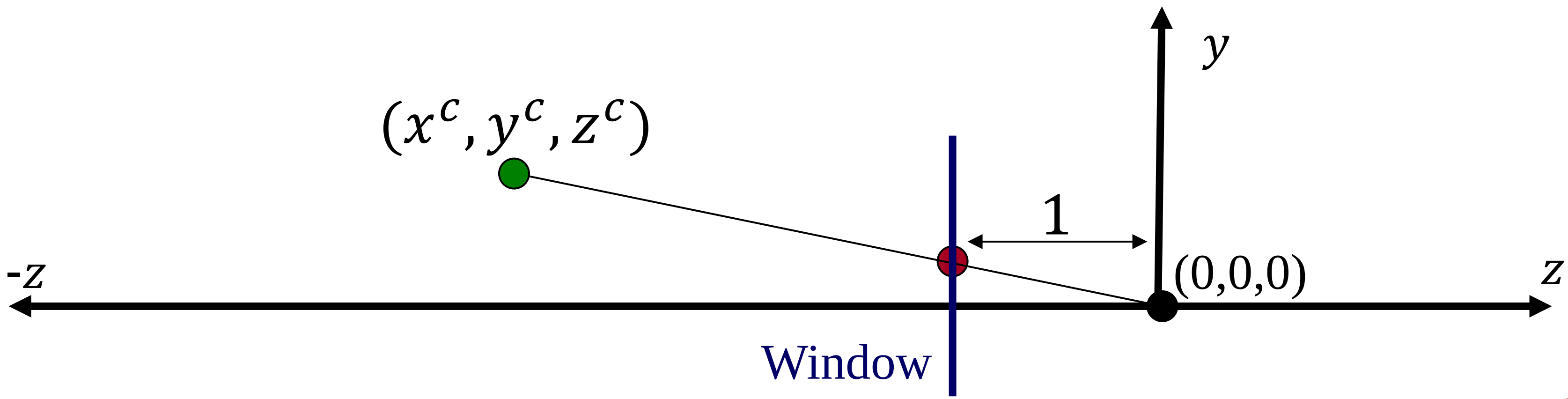


Perspective Projection

- For any point (x^c, y^c, z^c) and any scalar α, the points (x^c, y^c, z^c) and $(\alpha x^c, \alpha y^c, \alpha z^c)$ map to the same location.

- Since we want the position on the window that intersects the line from (x^c, y^c, z^c) to the origin:

$$(x^c, y^c, z^c) \rightarrow \left(\frac{x^c}{-z^c}, \frac{y^c}{-z^c}, -1\right)$$

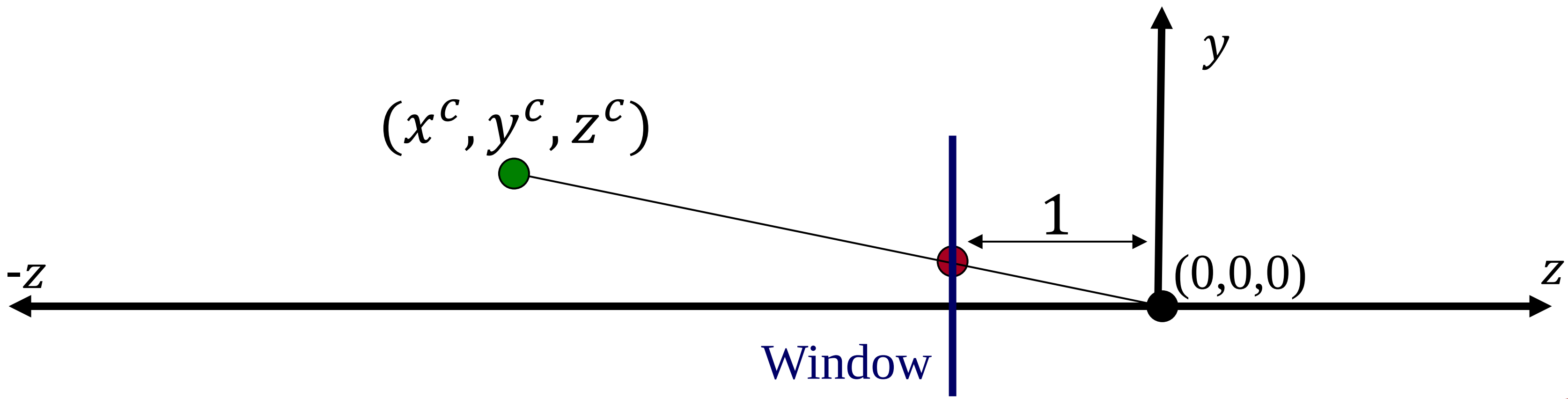


Perspective Projection

- For any point (x^c, y^c, z^c) and any scalar α, the points (x^c, y^c, z^c) and $(\alpha x^c, \alpha y^c, \alpha z^c)$ map to the same location.

- Since we want the position on the window that intersects the line from (x^c, y^c, z^c) to the origin:

$$(x^c, y^c, z^c, 1) \rightarrow \left(\frac{x^c}{-z^c}, \frac{y^c}{-z^c}, -1, 1\right)$$



Perspective Projection Matrix

$$(x^c, y^c, z^c, 1) \to \left(\frac{x^c}{-z^c}, \frac{y^c}{-z^c}, -1, 1\right)$$

Division by z^c can’t represented with a 3×3 matrix!

In homogenous coordinates, we can write this as:

$$(x^c, y^c, z^c, 1) \to (x^c, y^c, z^c, -z^c)$$

In matrix form, this gives:

$$\begin{bmatrix} -x^c/z^c \\ -y^c/z^c \\ -1 \\ 1 \end{bmatrix} \equiv \begin{bmatrix} x^c \\ y^c \\ z^c \\ -z^c \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x^c \\ y^c \\ z^c \\ 1 \end{bmatrix}$$

Perspective Projection Matrix

$$(x^c, y^c, z^c, 1) \to \left(\frac{x^c}{-z^c}, \frac{y^c}{-z^c}, -1, 1\right)$$

Division by z^c can't represented with a 3×3 matrix!

In homogenous coordinates, we can write this as:

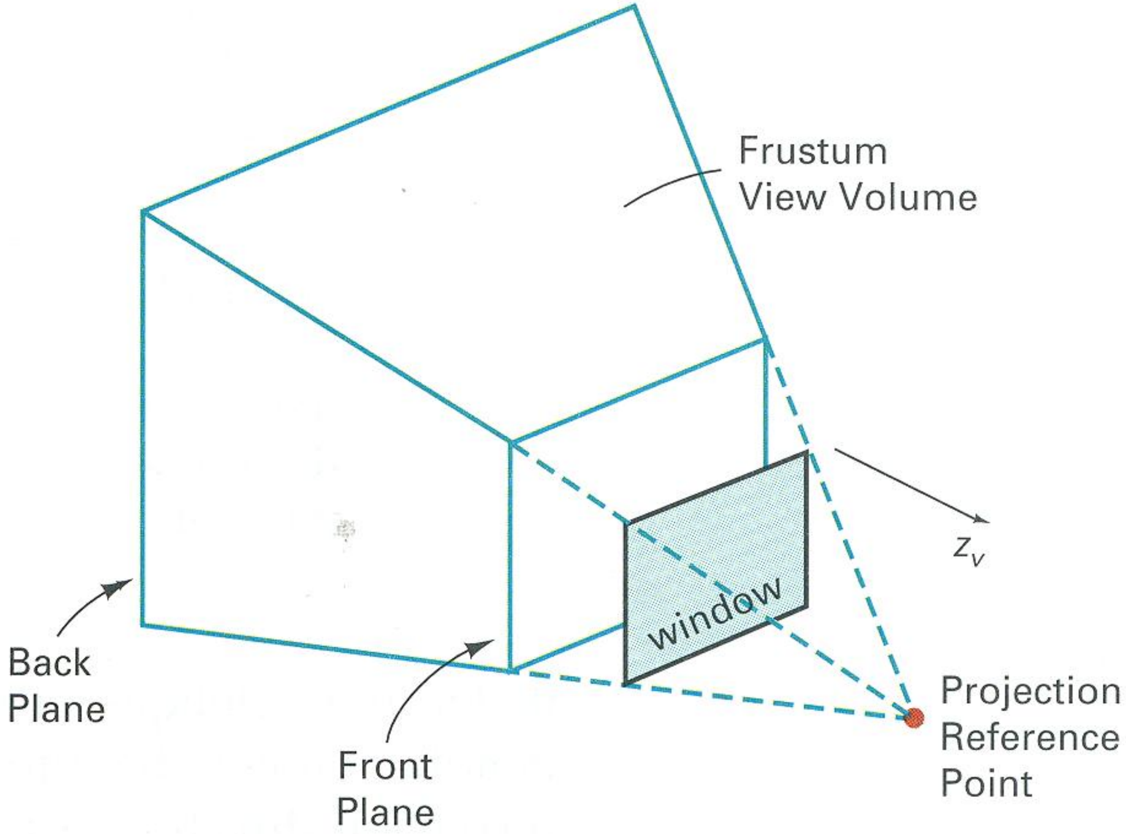
$$(x^c, y^c, z^c, 1) \to (x^c, y^c, z^c, -z^c)$$

In

<u>Note</u>:
This matrix describes a projective transformation

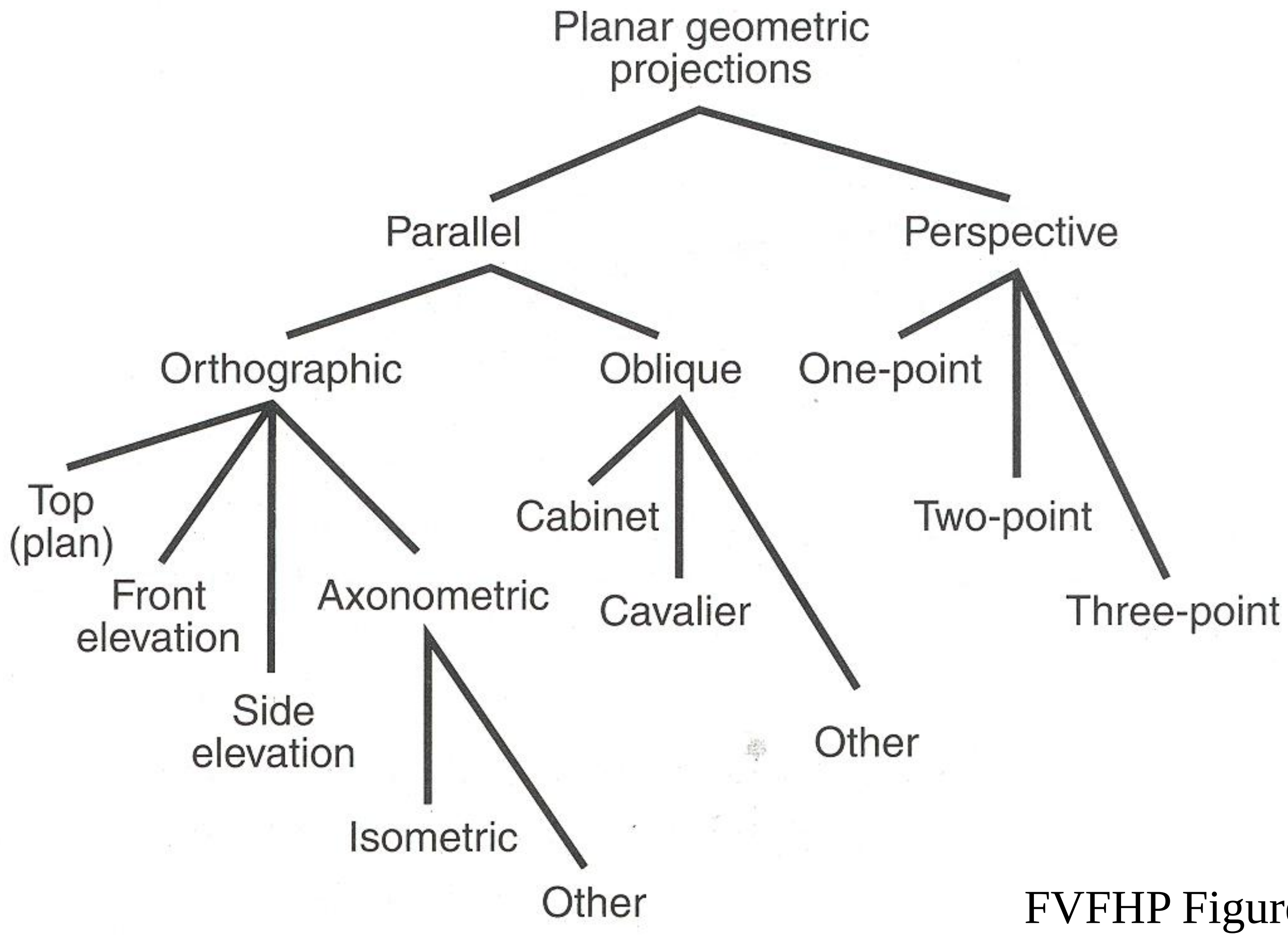
$$\begin{bmatrix} -y^c/z^c \\ -1 \\ 1 \end{bmatrix} \equiv \begin{bmatrix} y^c \\ z^c \\ -z^c \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} y^c \\ z^c \\ 1 \end{bmatrix}$$

Perspective Projection View Volume



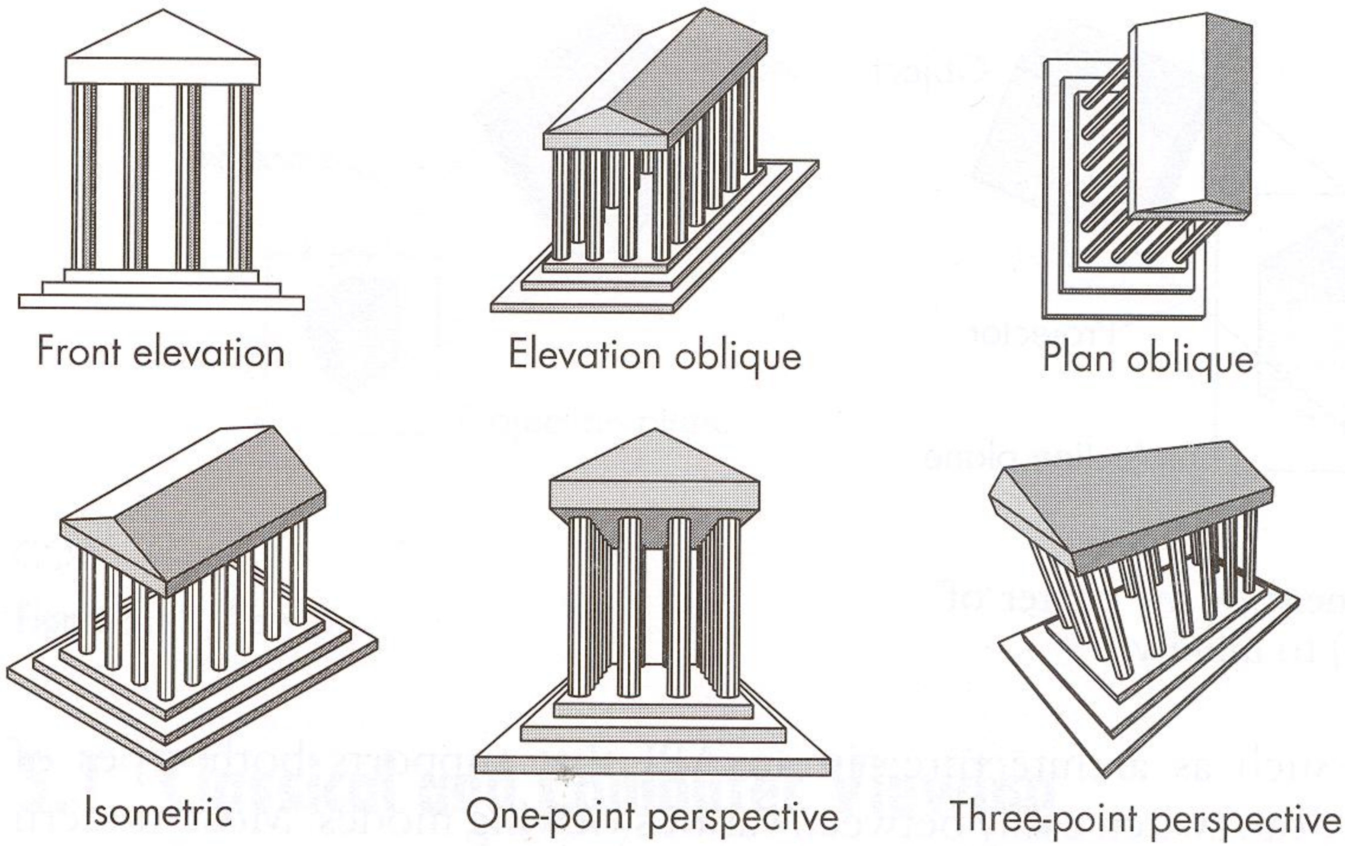
H&B Figure 12.30

Taxonomy of Projections



FVFHP Figure 6.10

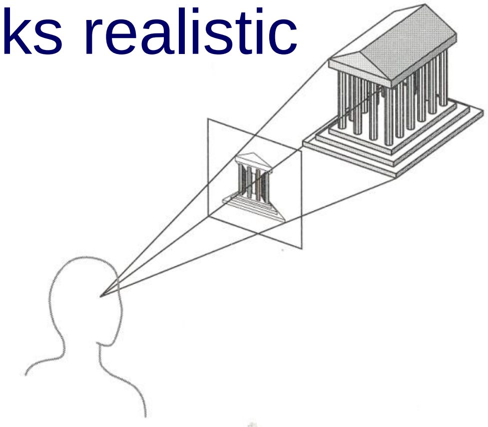
Classical Projections



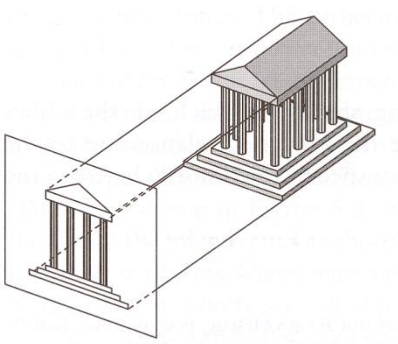
Angel Figure 5.3

Perspective vs. Parallel

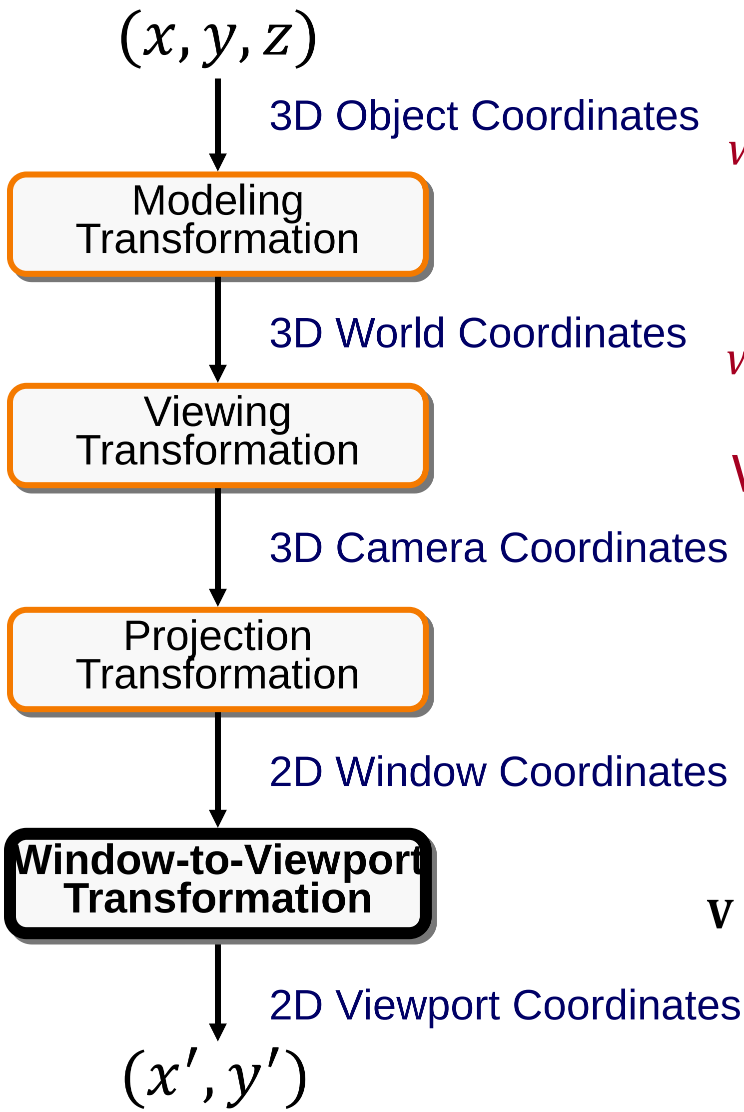
- Perspective projection
  - ✓ Size varies inversely with distance - looks realistic
  - ✓ Angles are preserved on faces parallel to the view plane
  - ✗ Distance are not preserved

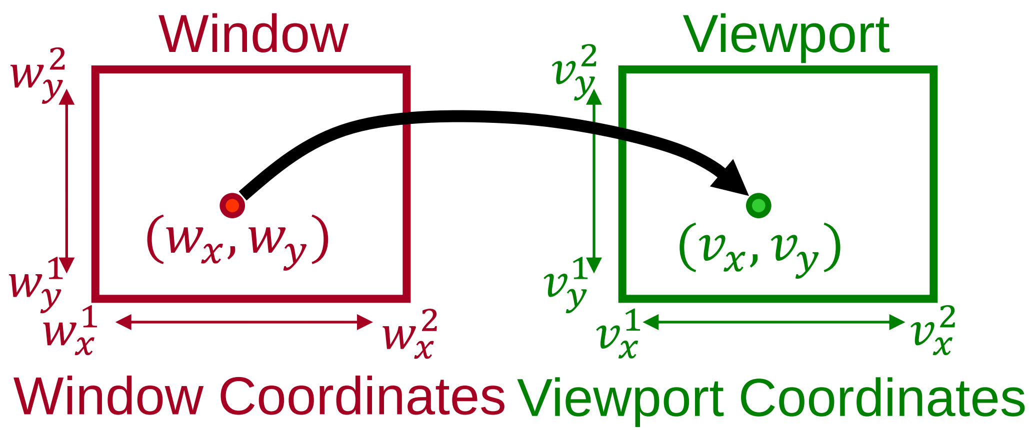


- Parallel (orthographic) projection
  - ✓ Parallel lines remain parallel
  - ✓ Angles **and distances** are preserved on faces parallel to the view plane
  - ✗ Less realistic looking
  - ✓ Good for exact measurements



Transformations





$\mathbf{V}$ = viewport transform

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

Note that this may scale non-uniformly.

3D Rendering Pipeline (for direct illumination)

(x, y, z)

↓ 3D Object Coordinates

Modeling Transformation

↓ 3D World Coordinates

Viewing Transformation

↓ 3D Camera Coordinates

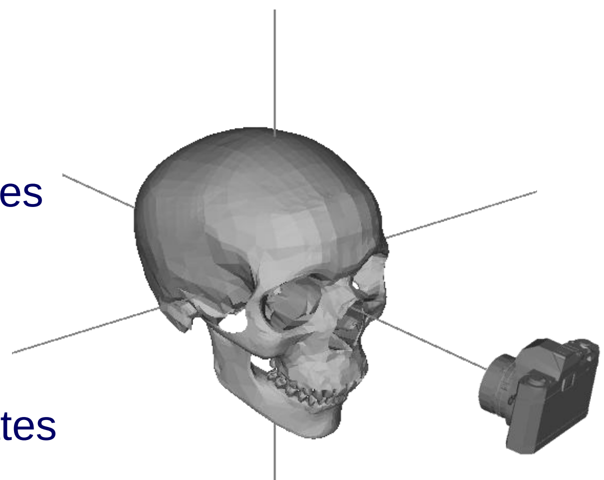
Projection Transformation

↓ 2D Window Coordinates

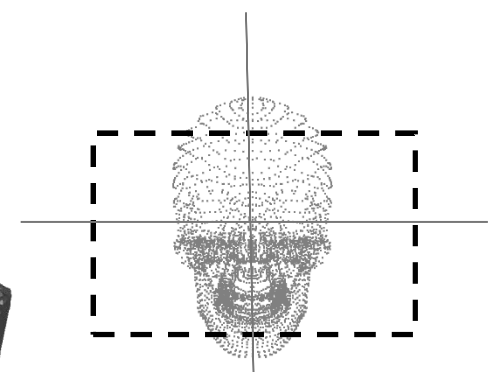
Window-to-Viewport Transformation

↓ 2D Viewport Coordinates

(x', y')



3D Model



2D Viewport

Transformations

3D Primitives

↓ 3D Modeling Coordinates

Modeling Transformation

↓ 3D World Coordinates

Viewing Transformation

↓ 3D Camera Coordinates

Lighting

↓ 3D Camera Coordir

Projection Transformation

↓ 2D Window Coordinates

Clipping

↓ 2D Window Coordinates

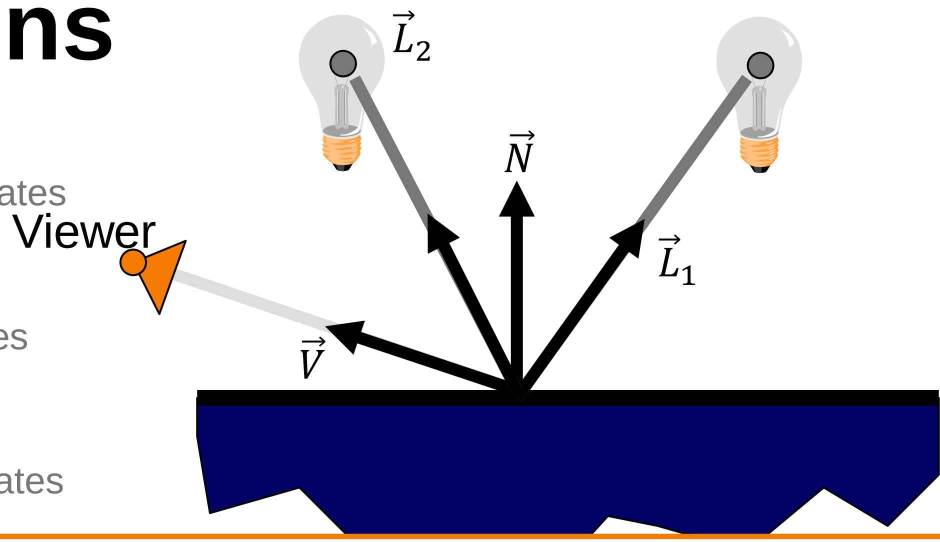
Viewport Transformation

↓ 2D Viewport Coordinates

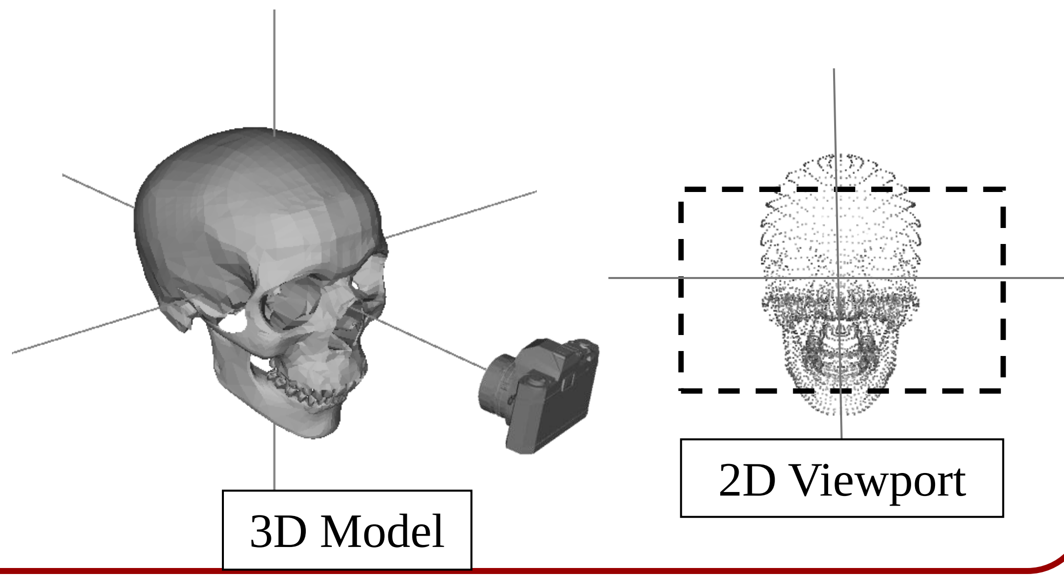
Scan Conversion

↓ 2D Viewport Coordinates

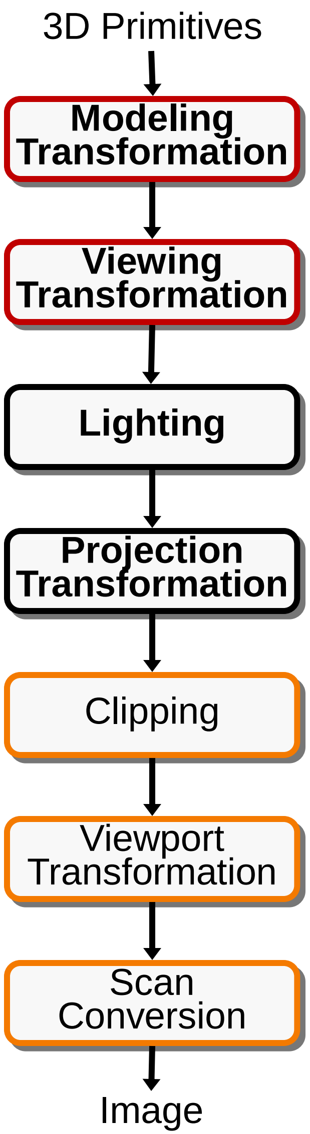
Image



$$I = I_E + \sum_L \left[K_A \cdot I_L^A + \left(K_D \cdot \langle \vec{N},\vec{L}\rangle + K_S \cdot \langle \vec{V},\vec{R}\rangle^n\right) \cdot I_L\right]$$

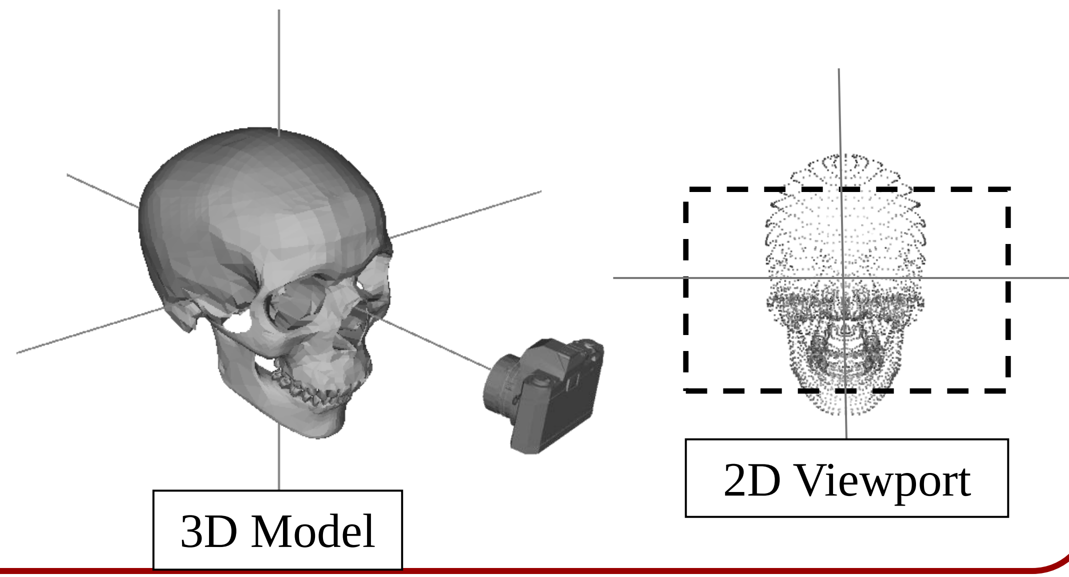


Transformations

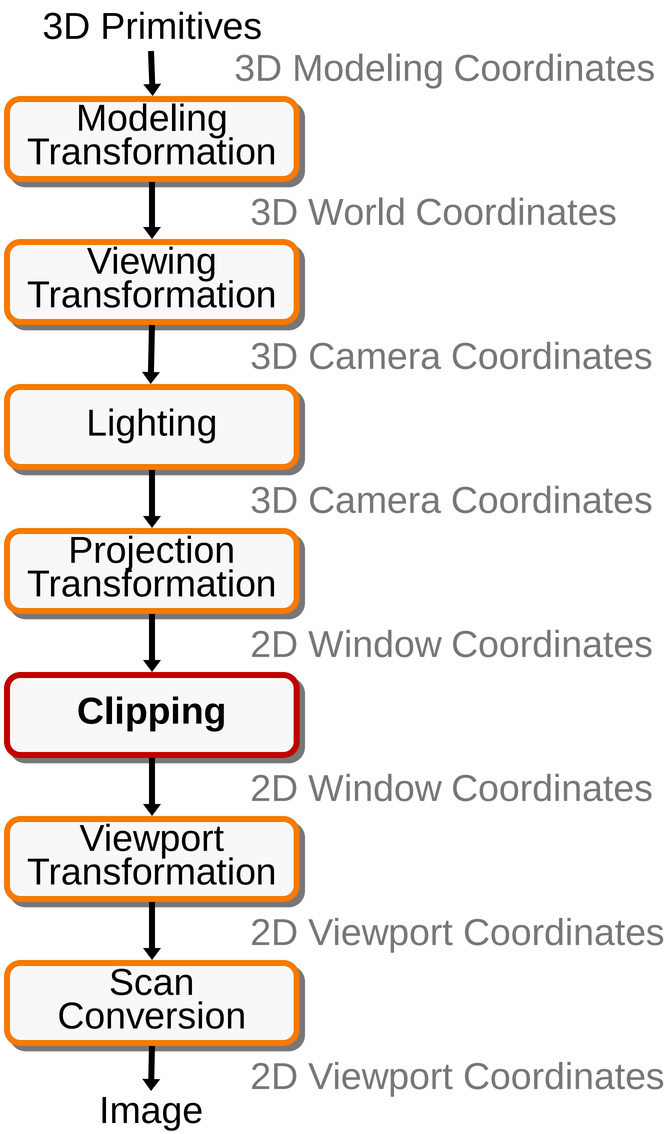


Vertex processing

- Originally, vertex processing was fixed
- Now this is programmable in the vertex shader



3D Rendering Pipeline (for direct illumination)



Clipping

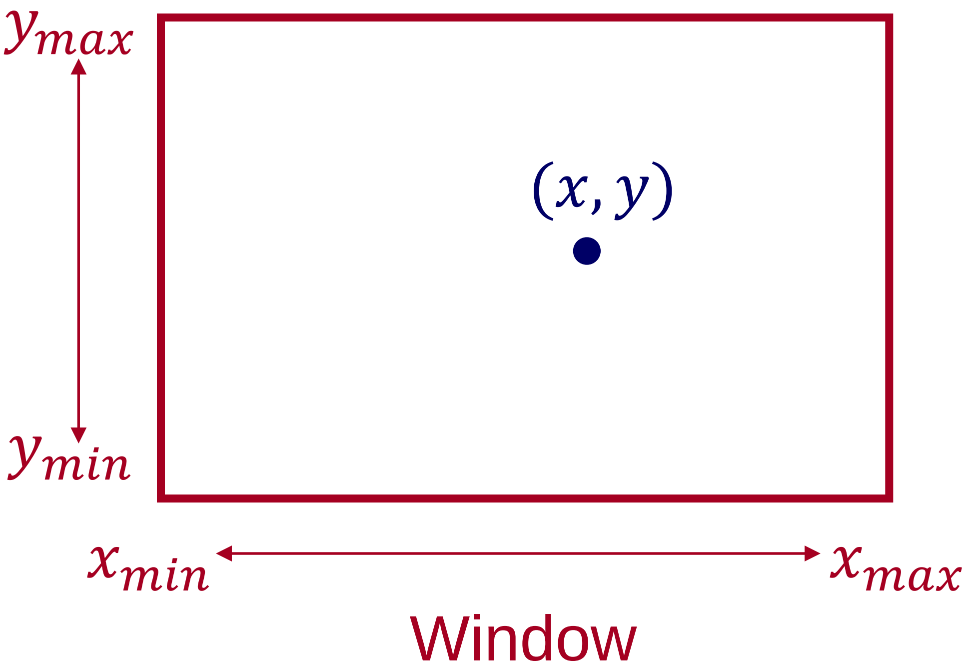
- Avoid drawing parts of primitives outside window
  - Window defines the subset of the scene being viewed
  - Must draw geometric primitives only inside window

Clipping

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

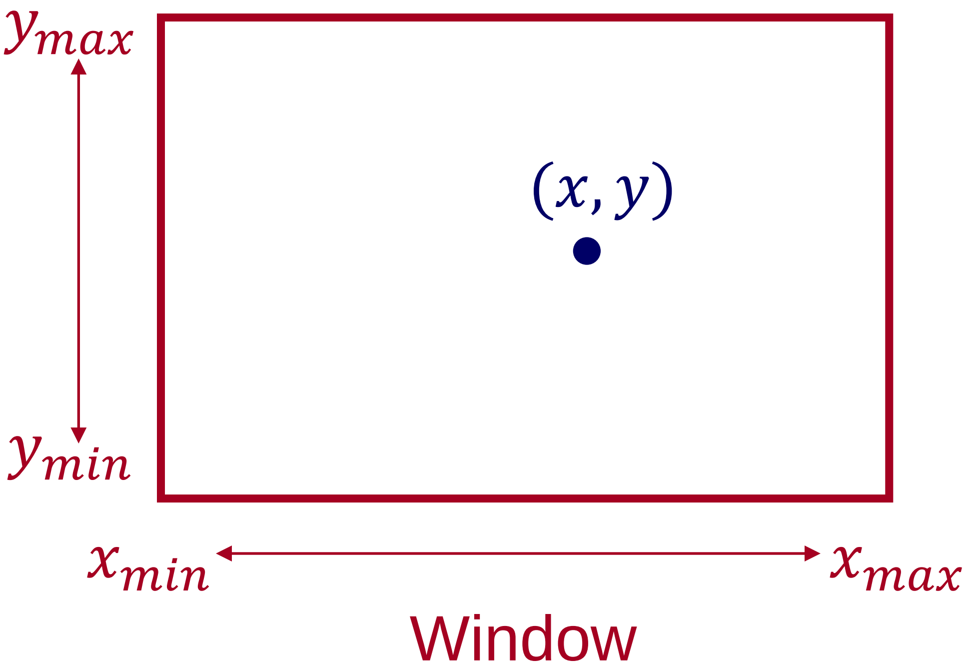
Point Clipping

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



Point Clipping

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



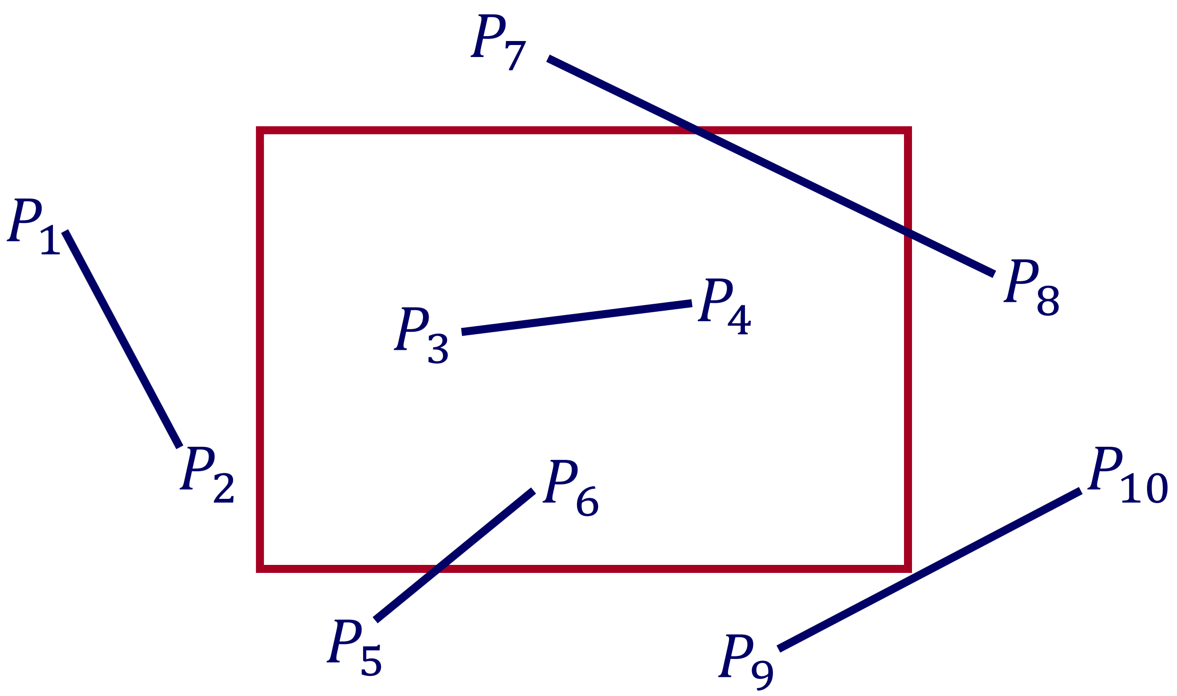

```
inside =
  (x >= x_min) &&
  (x <  x_max) &&
  (y >= y_min) &&
  (y <  y_max);
```

Clipping

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

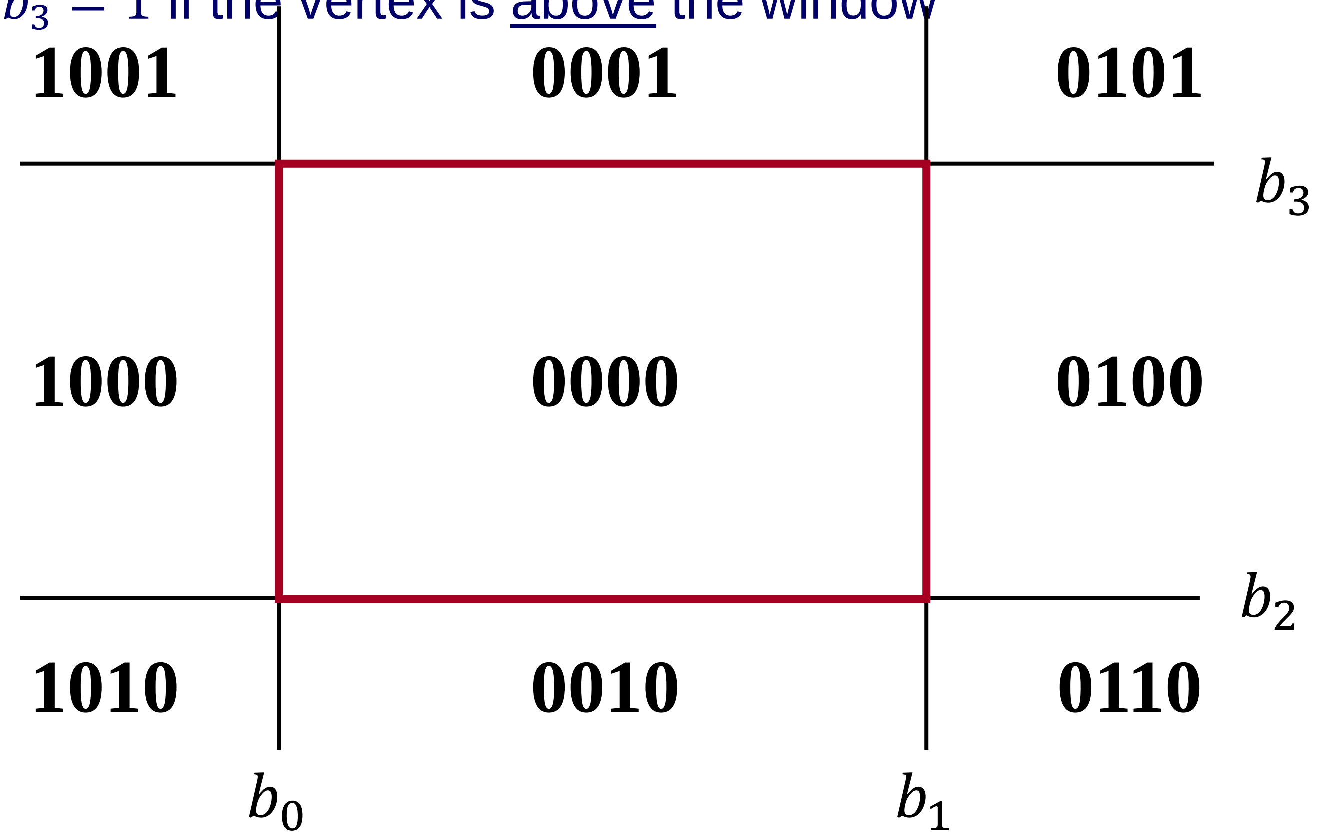
Line Segment Clipping

- Find the part of a line inside the clip window
  - Do this as <u>efficiently</u> as possible by identifying the easiest cases first



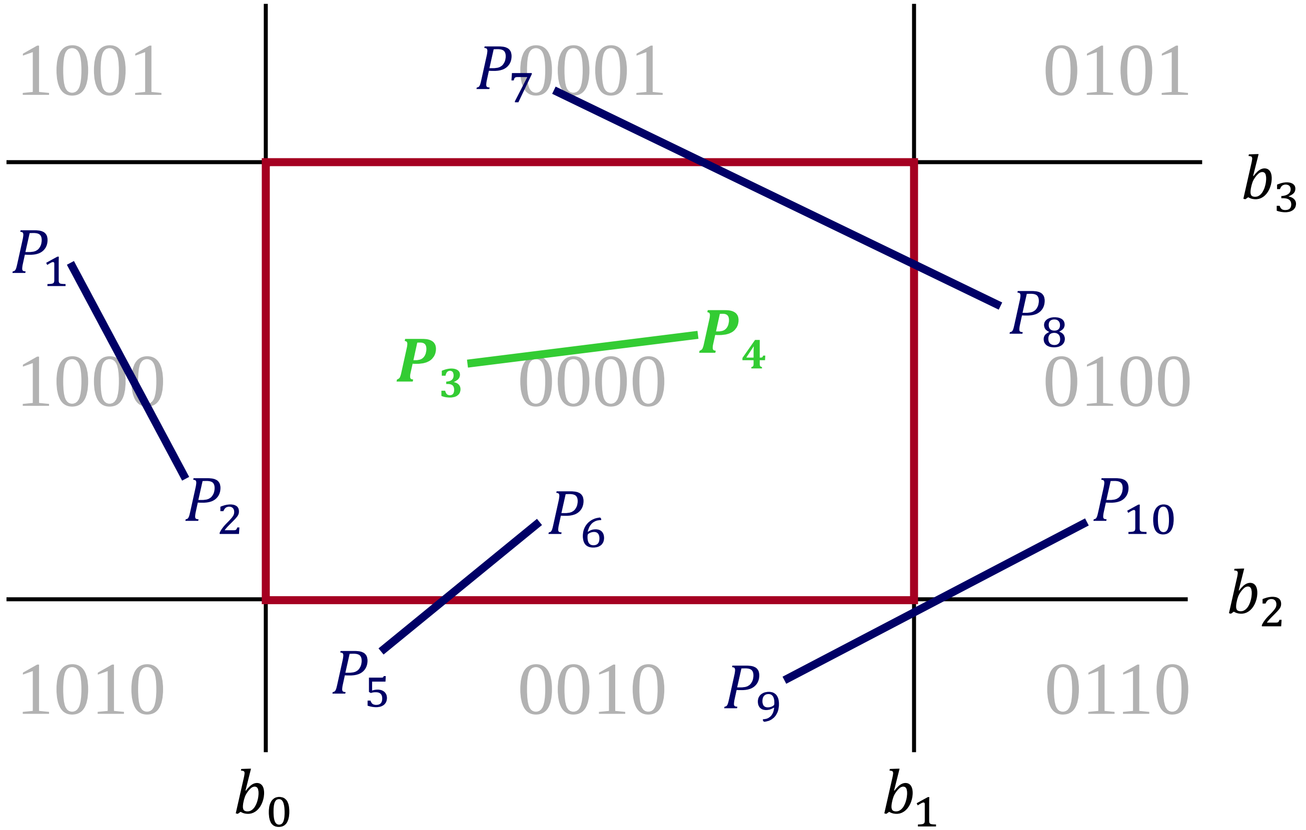
Cohen-Sutherland Line Clipping

- **Associate a 4-bit <u>outcode</u> $b_0 b_1 b_2 b_3$ to each vertex**
  - $b_0 = 1$ if the vertex is <u>left of</u> the window
  - $b_1 = 1$ if the vertex is <u>right of</u> the window
  - $b_2 = 1$ if the vertex is <u>below</u> the window
  - $b_3 = 1$ if the vertex is <u>above</u> the window



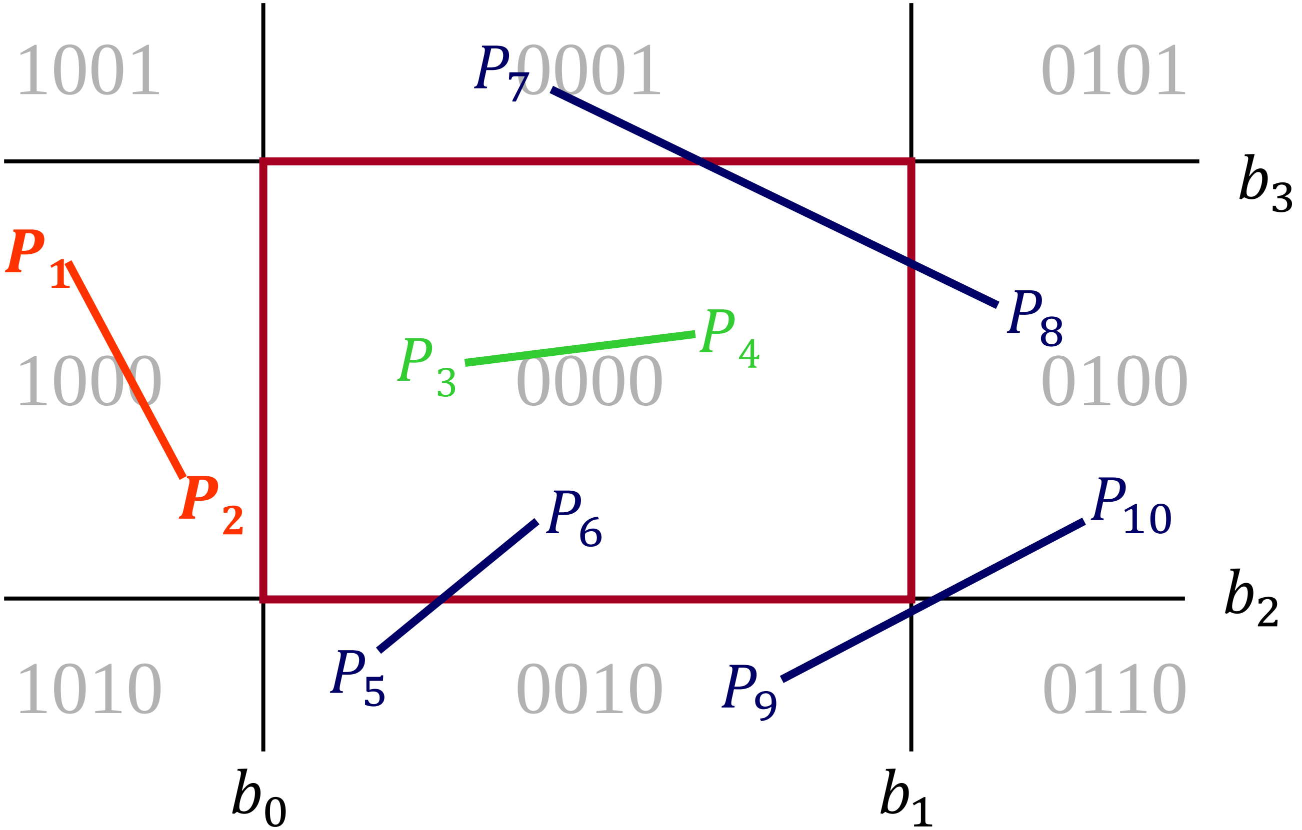
Cohen-Sutherland Line Clipping

- Associate a 4-bit <u>outcode</u> $b_0 b_1 b_2 b_3$ to each vertex
- **If both points' outcodes are 0, line segment is inside**



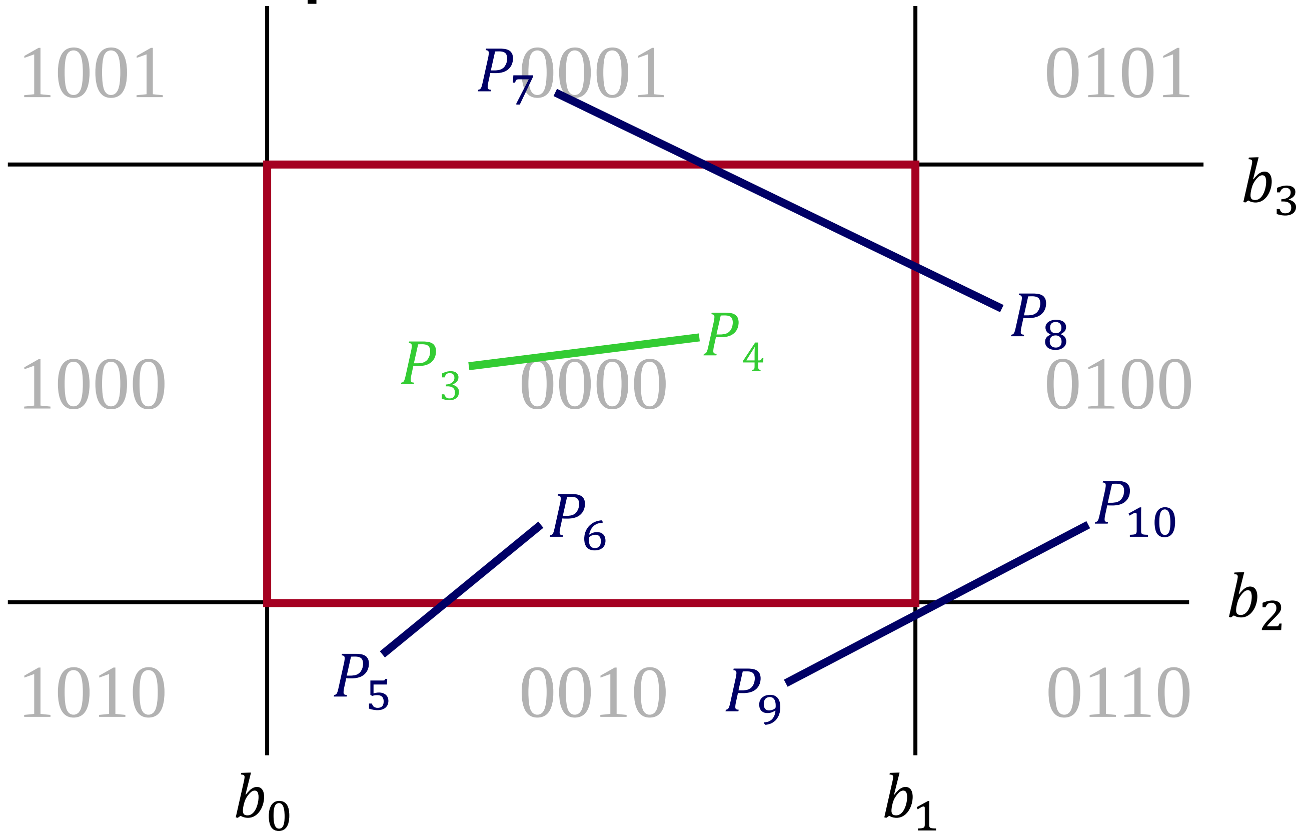
Cohen-Sutherland Line Clipping

- Associate a 4-bit outcode $b_0 b_1 b_2 b_3$ to each vertex
- If both points' outcodes are 0, line segment is inside
- **If AND of outcodes is not 0, line segment is outside**



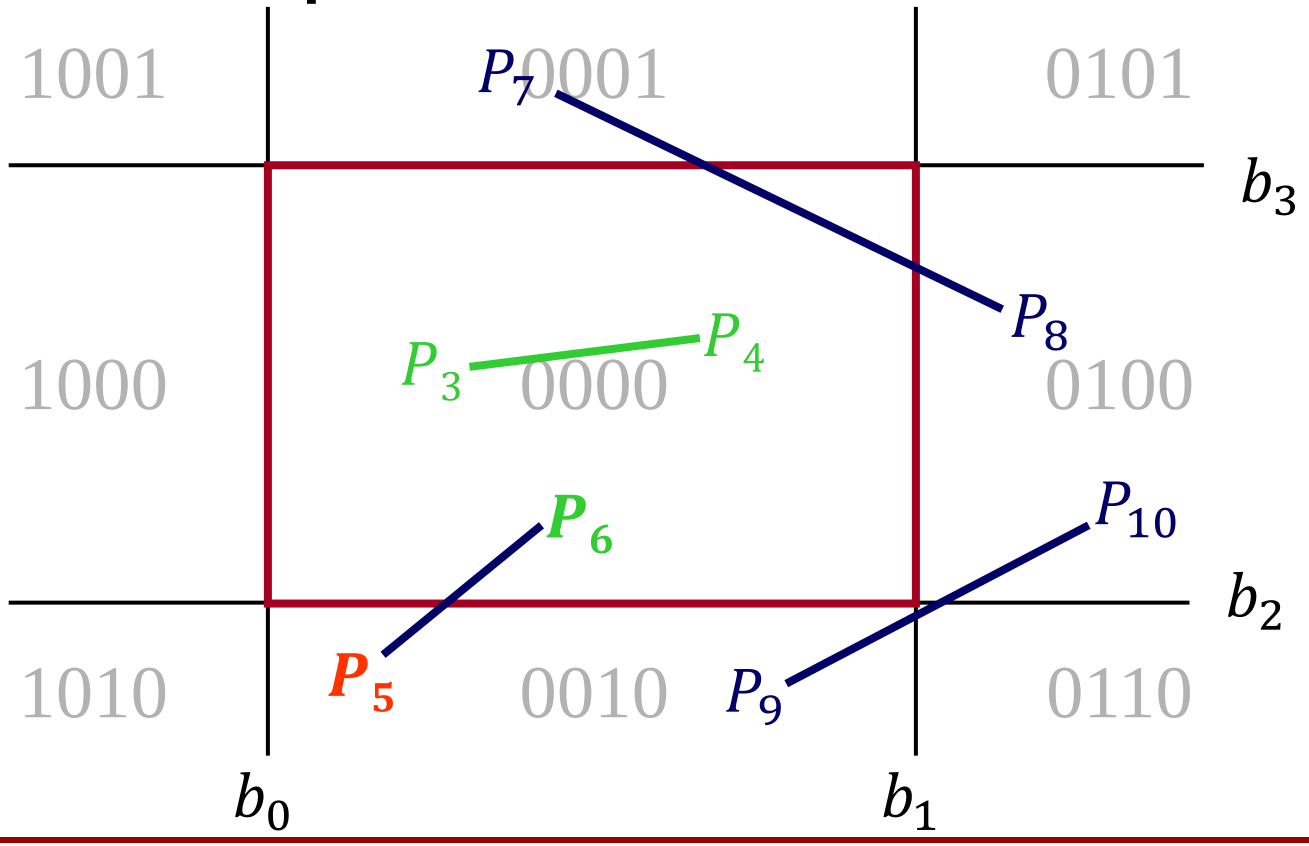
Cohen-Sutherland Line Clipping

- Associate a 4-bit <u>outcode</u> $b_0 b_1 b_2 b_3$ to each vertex
- If both points' outcodes are 0, line segment is inside
- If AND of outcodes is not 0, line segment is outside
- **Otherwise clip and test**



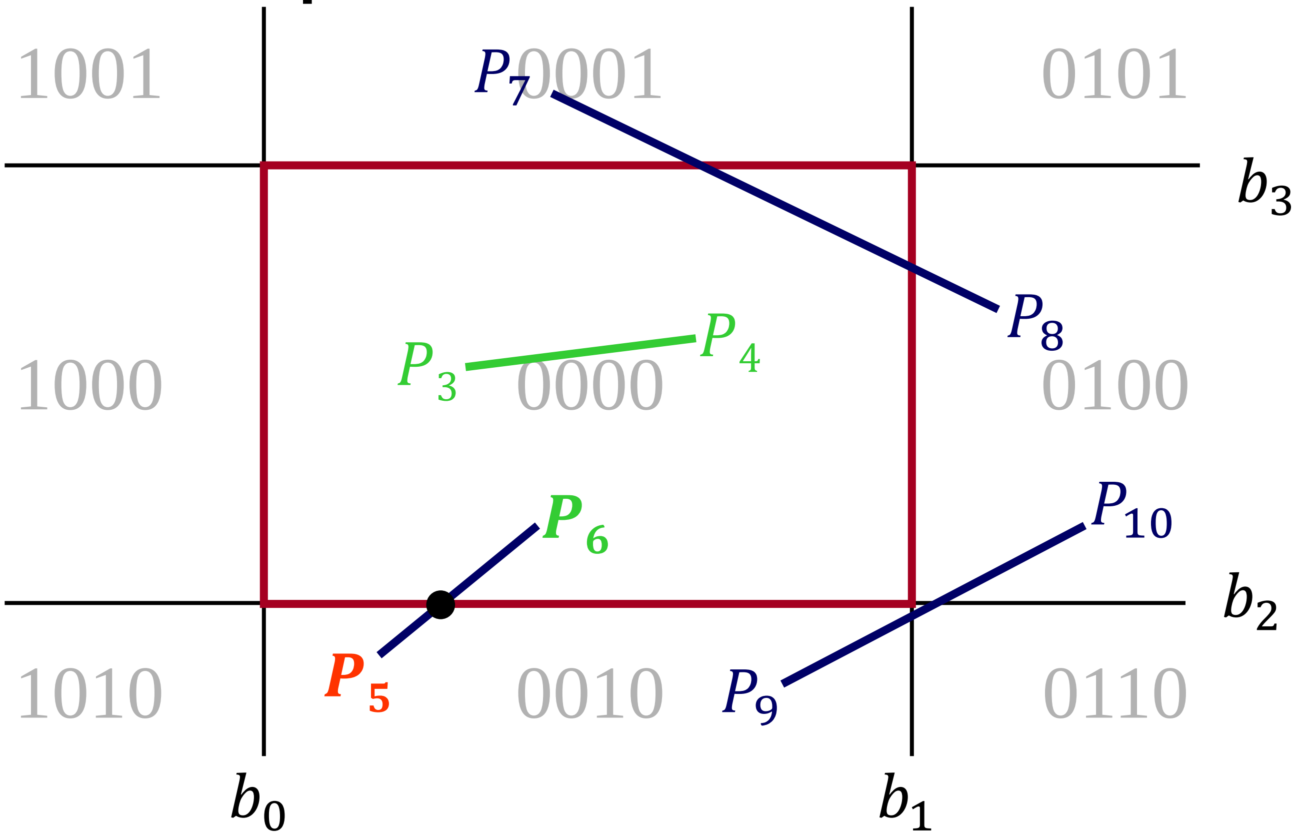
Cohen-Sutherland Line Clipping

- Associate a 4-bit outcode $b_0 b_1 b_2 b_3$ to each vertex
- If both points' outcodes are 0, line segment is inside
- If AND of outcodes is not 0, line segment is outside
- **Otherwise clip and test**



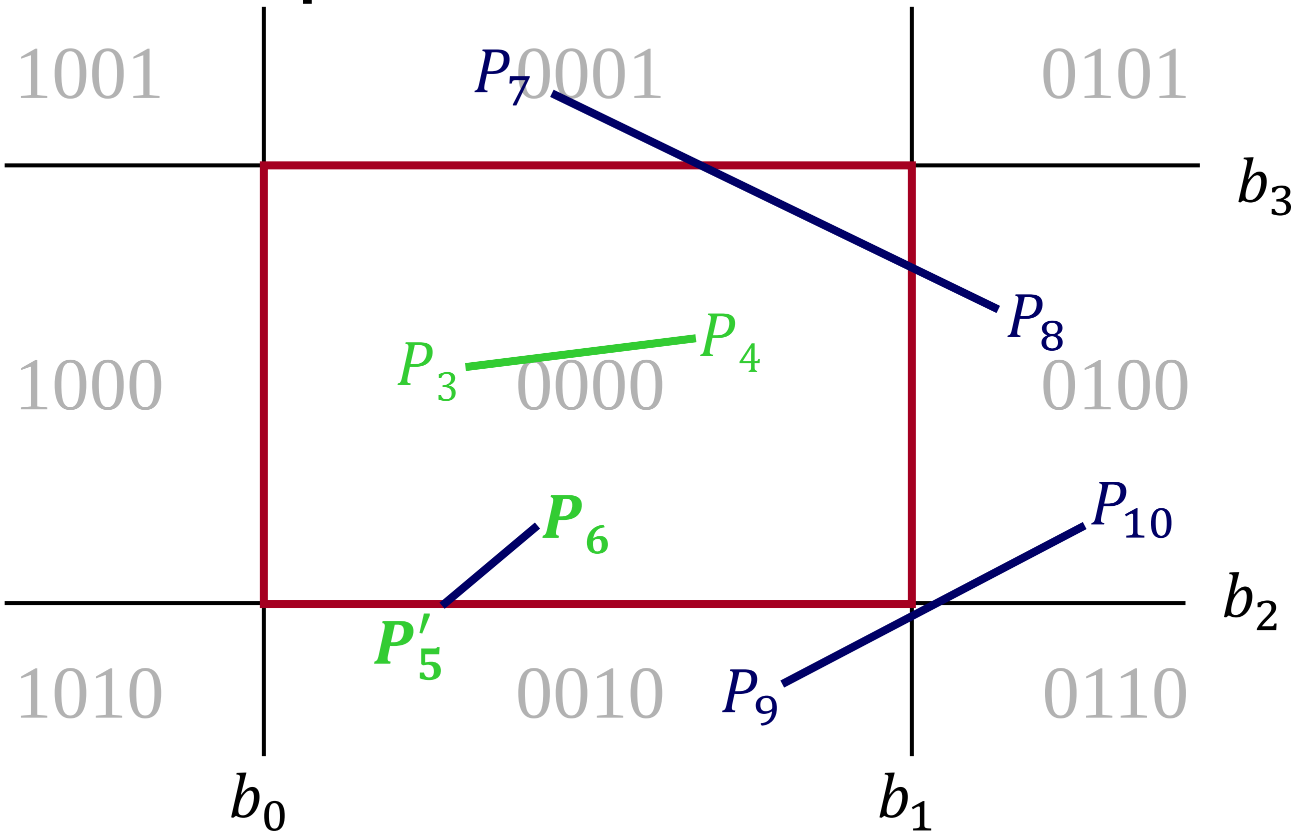
Cohen-Sutherland Line Clipping

- Associate a 4-bit <u>outcode</u> $b_0 b_1 b_2 b_3$ to each vertex
- If both points' outcodes are 0, line segment is inside
- If AND of outcodes is not 0, line segment is outside
- **Otherwise clip and test**



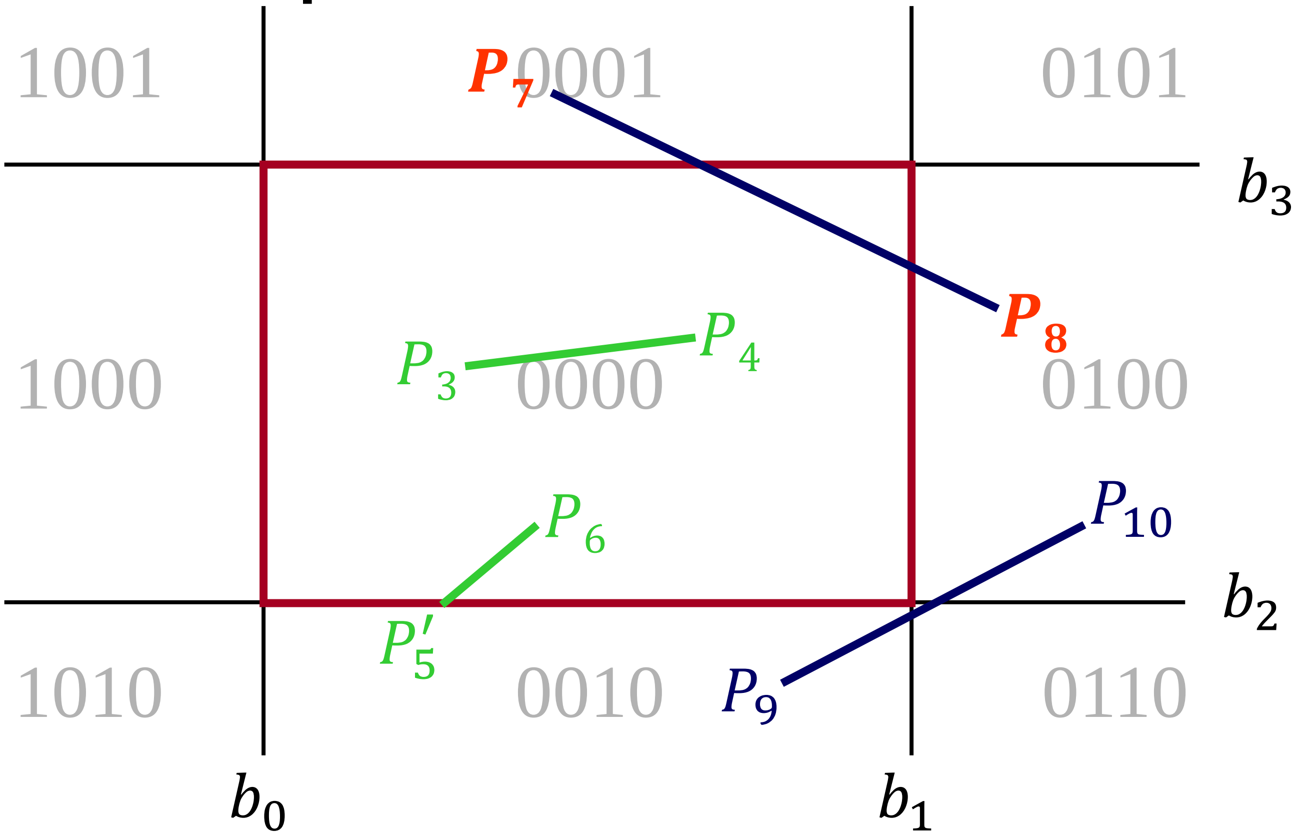
Cohen-Sutherland Line Clipping

- Associate a 4-bit outcode $b_0 b_1 b_2 b_3$ to each vertex
- If both points' outcodes are 0, line segment is inside
- If AND of outcodes is not 0, line segment is outside
- **Otherwise clip and test**



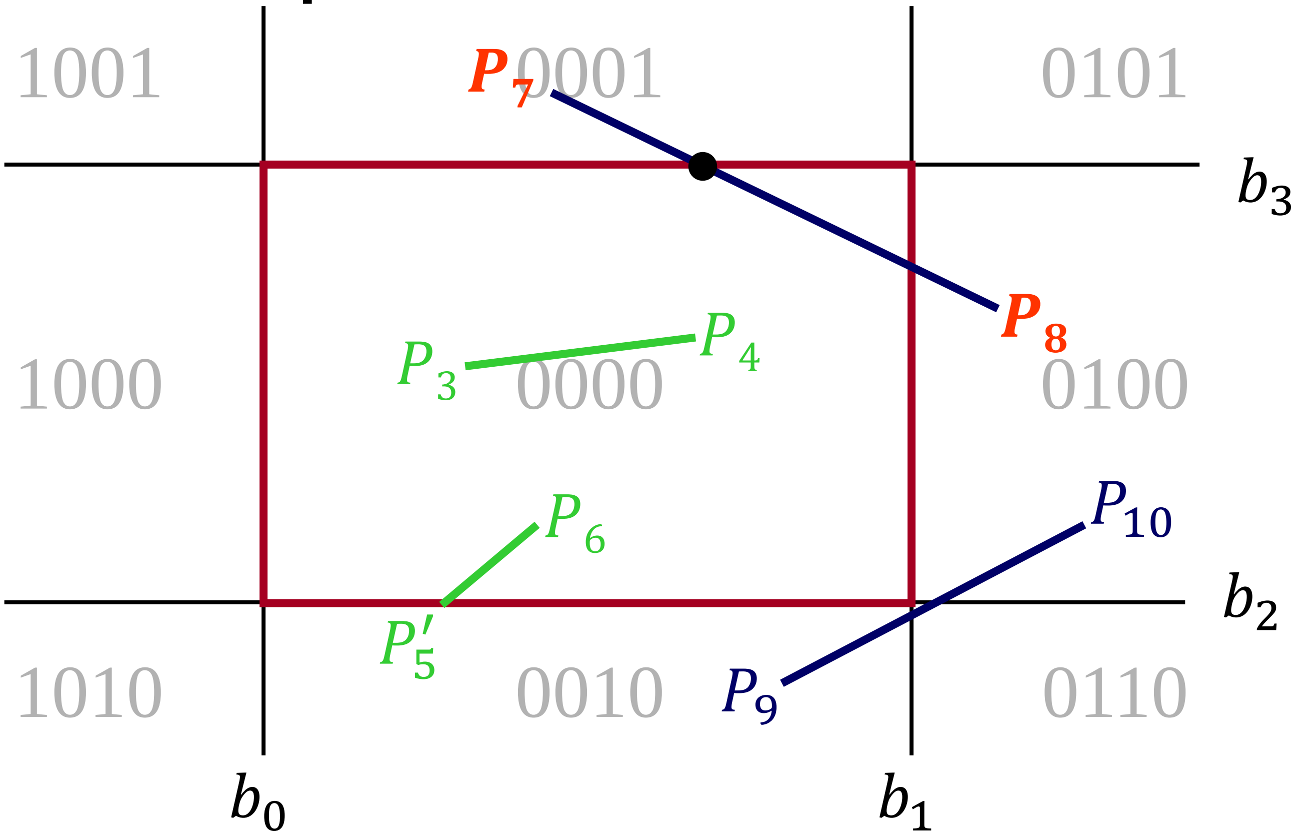
Cohen-Sutherland Line Clipping

- Associate a 4-bit <u>outcode</u> $b_0 b_1 b_2 b_3$ to each vertex
- If both points' outcodes are 0, line segment is inside
- If AND of outcodes is not 0, line segment is outside
- **Otherwise clip and test**



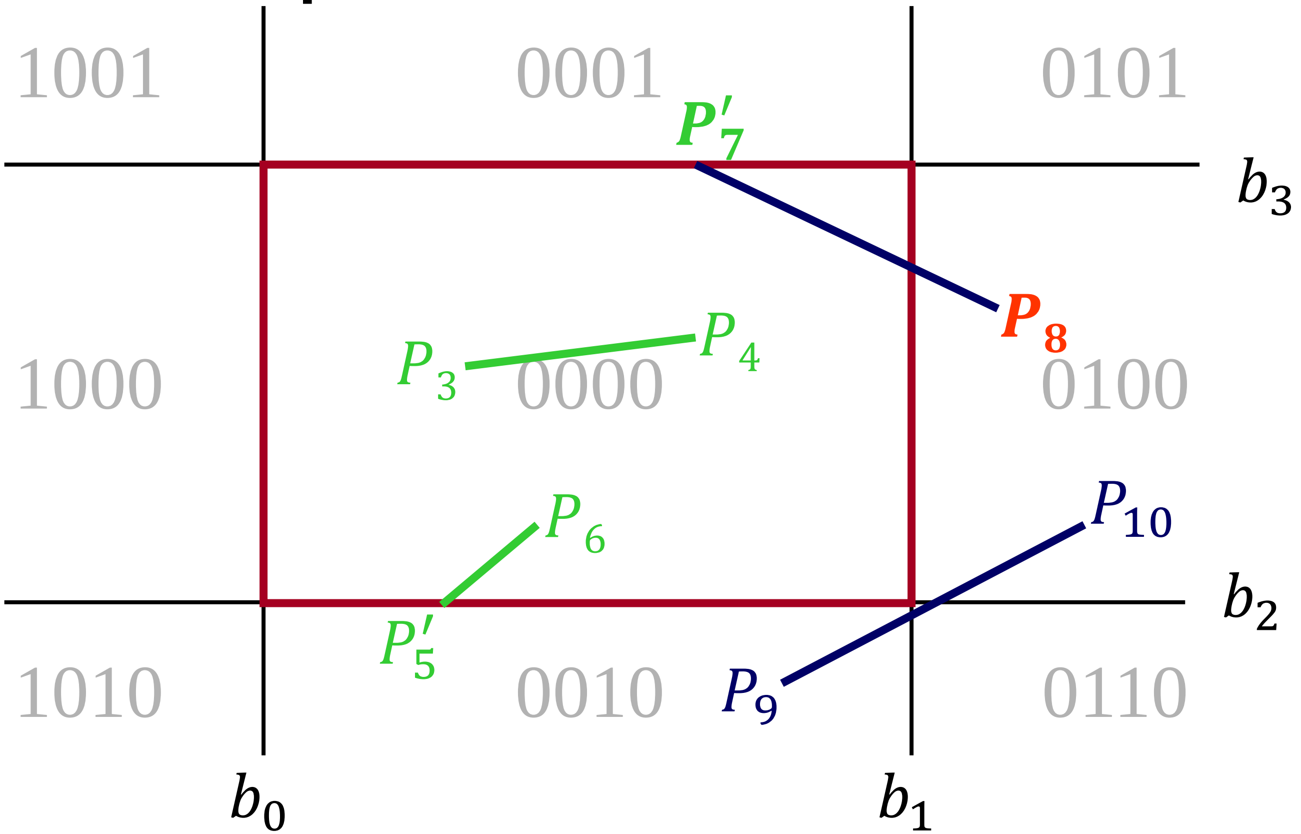
Cohen-Sutherland Line Clipping

- Associate a 4-bit <u>outcode</u> $b_0 b_1 b_2 b_3$ to each vertex
- If both points' outcodes are 0, line segment is inside
- If AND of outcodes is not 0, line segment is outside
- **Otherwise clip and test**



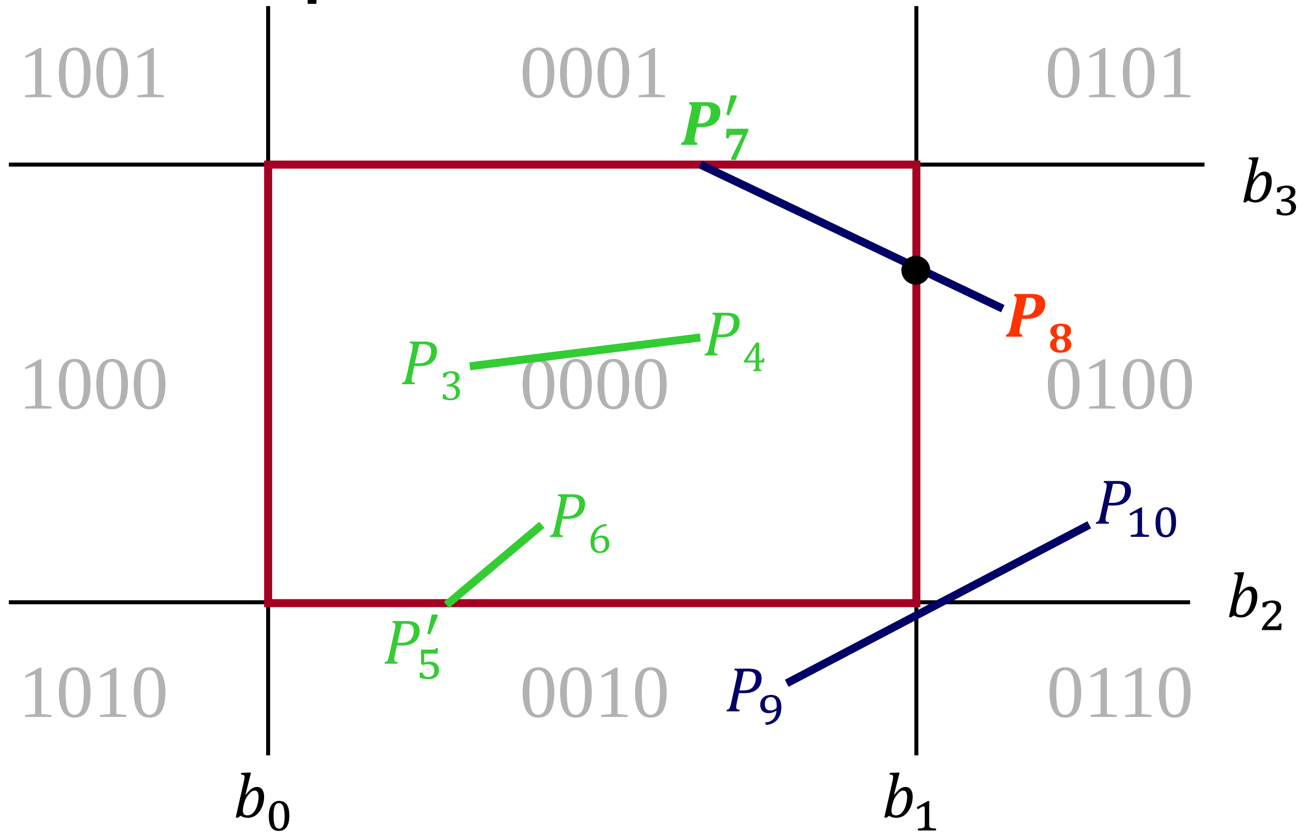
Cohen-Sutherland Line Clipping

- Associate a 4-bit <u>outcode</u> $b_0 b_1 b_2 b_3$ to each vertex
- If both points' outcodes are 0, line segment is inside
- If AND of outcodes is not 0, line segment is outside
- **Otherwise clip and test**



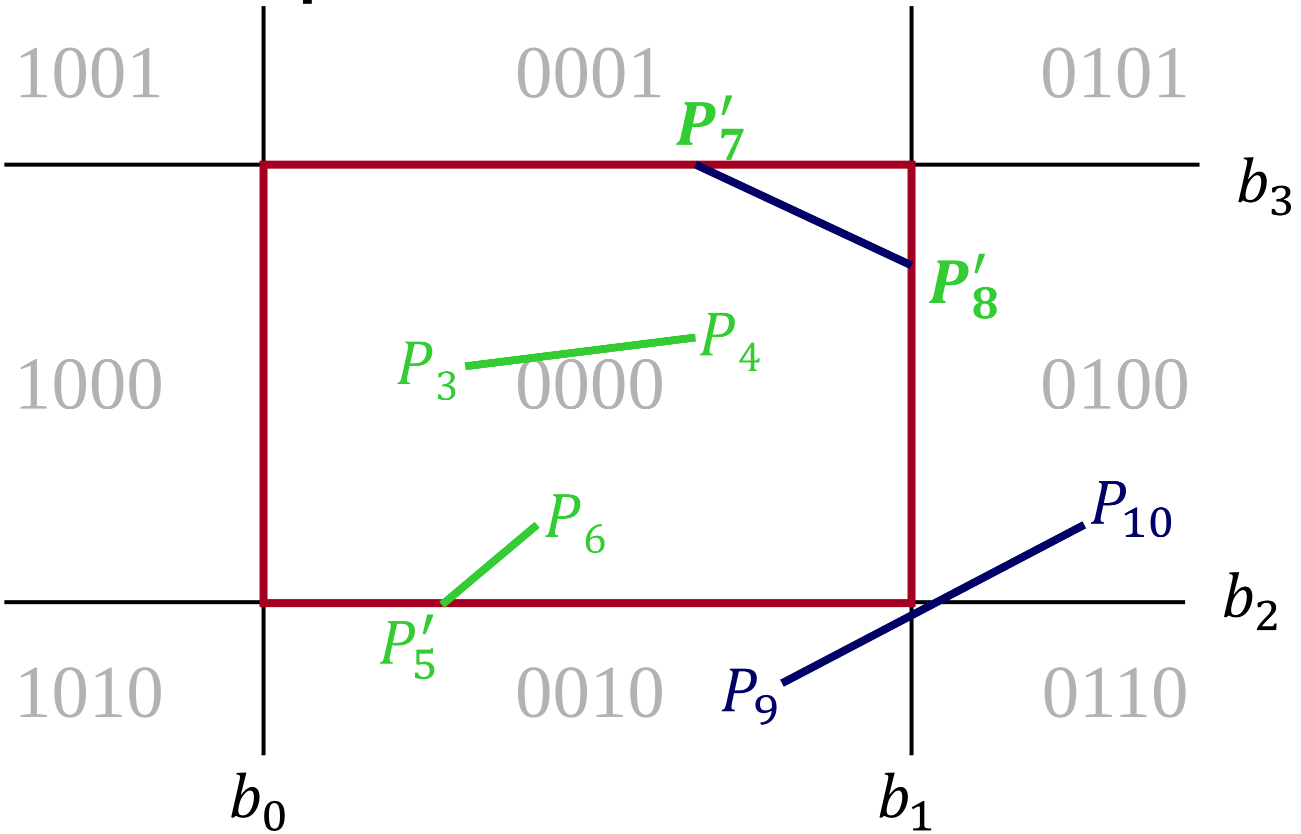
Cohen-Sutherland Line Clipping

- Associate a 4-bit <u>outcode</u> $b_0 b_1 b_2 b_3$ to each vertex
- If both points' outcodes are 0, line segment is inside
- If AND of outcodes is not 0, line segment is outside
- **Otherwise clip and test**



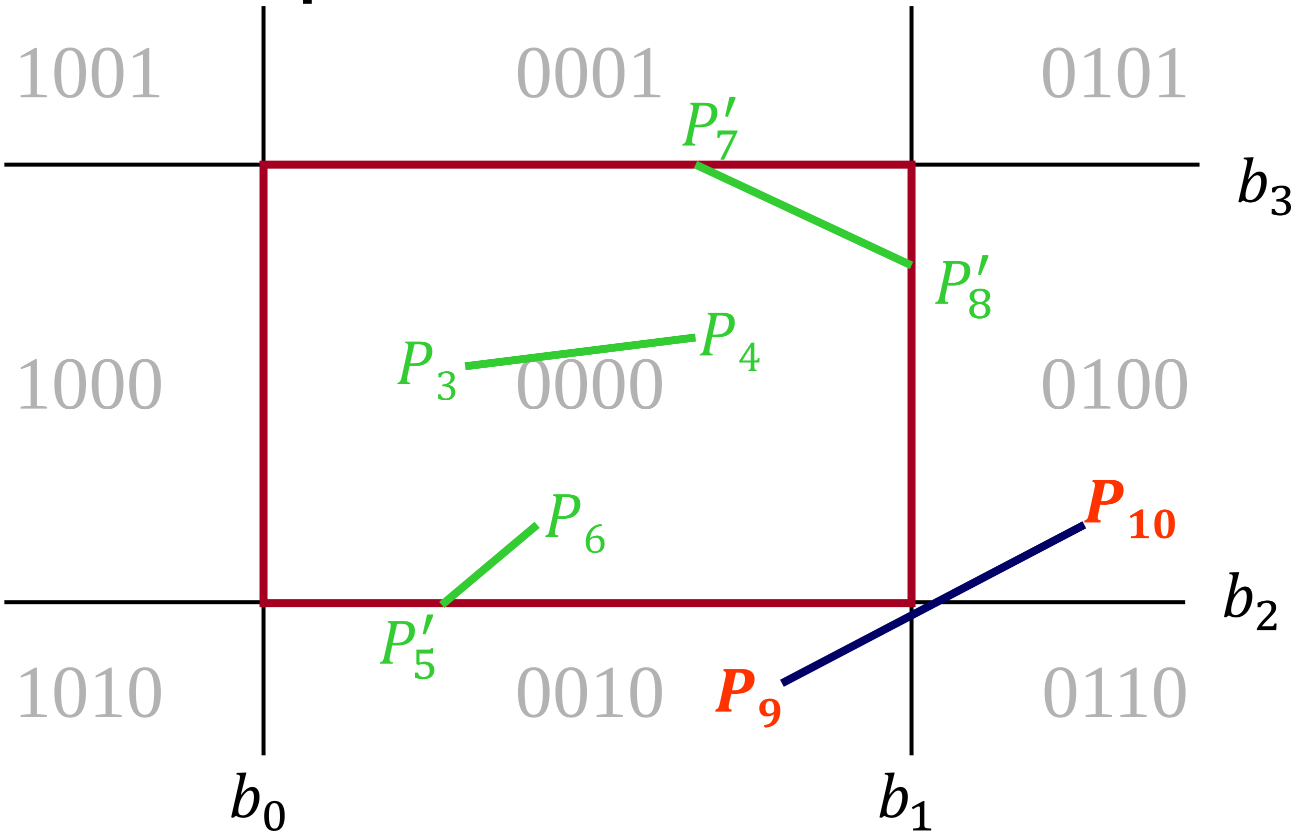
Cohen-Sutherland Line Clipping

- Associate a 4-bit outcode $b_0 b_1 b_2 b_3$ to each vertex
- If both points' outcodes are 0, line segment is inside
- If AND of outcodes is not 0, line segment is outside
- **Otherwise clip and test**



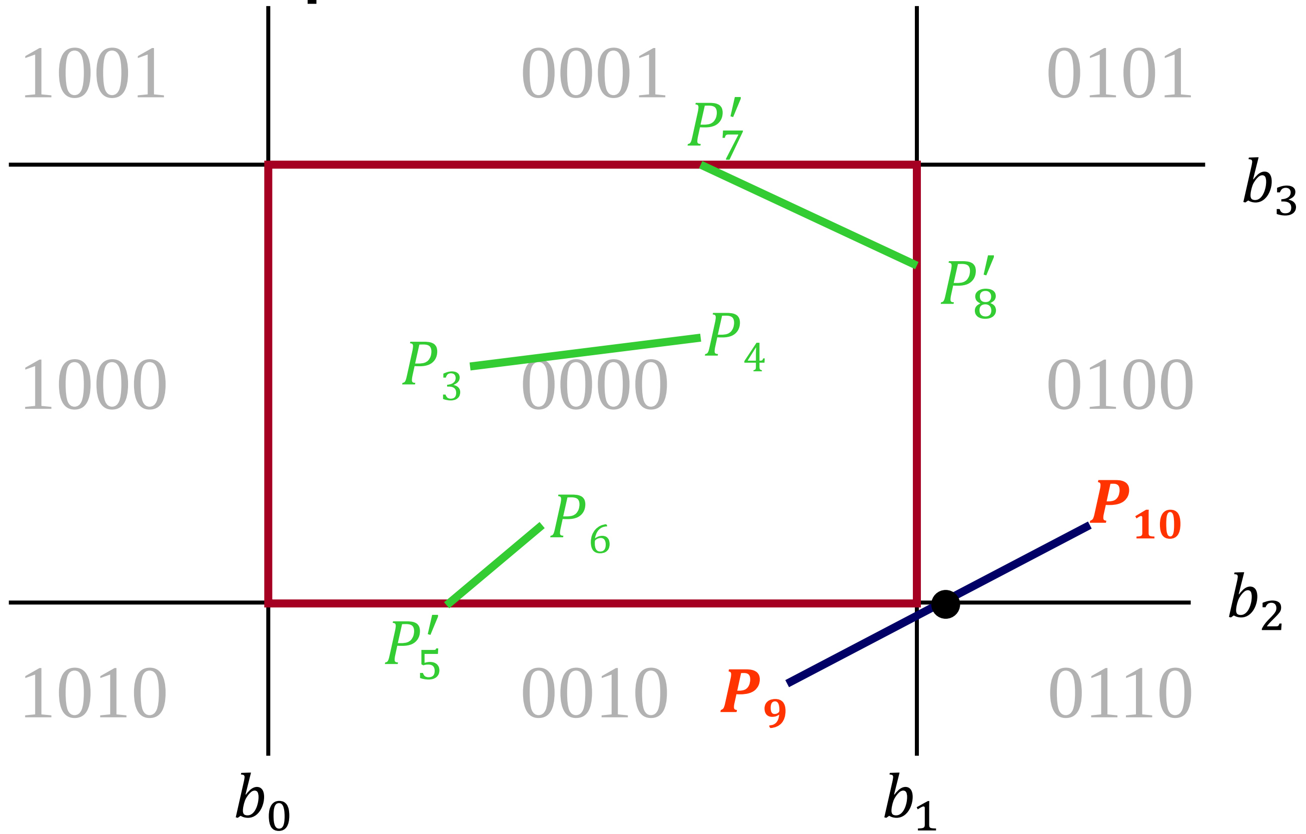
Cohen-Sutherland Line Clipping

- Associate a 4-bit outcode $b_0 b_1 b_2 b_3$ to each vertex
- If both points' outcodes are 0, line segment is inside
- If AND of outcodes is not 0, line segment is outside
- **Otherwise clip and test**



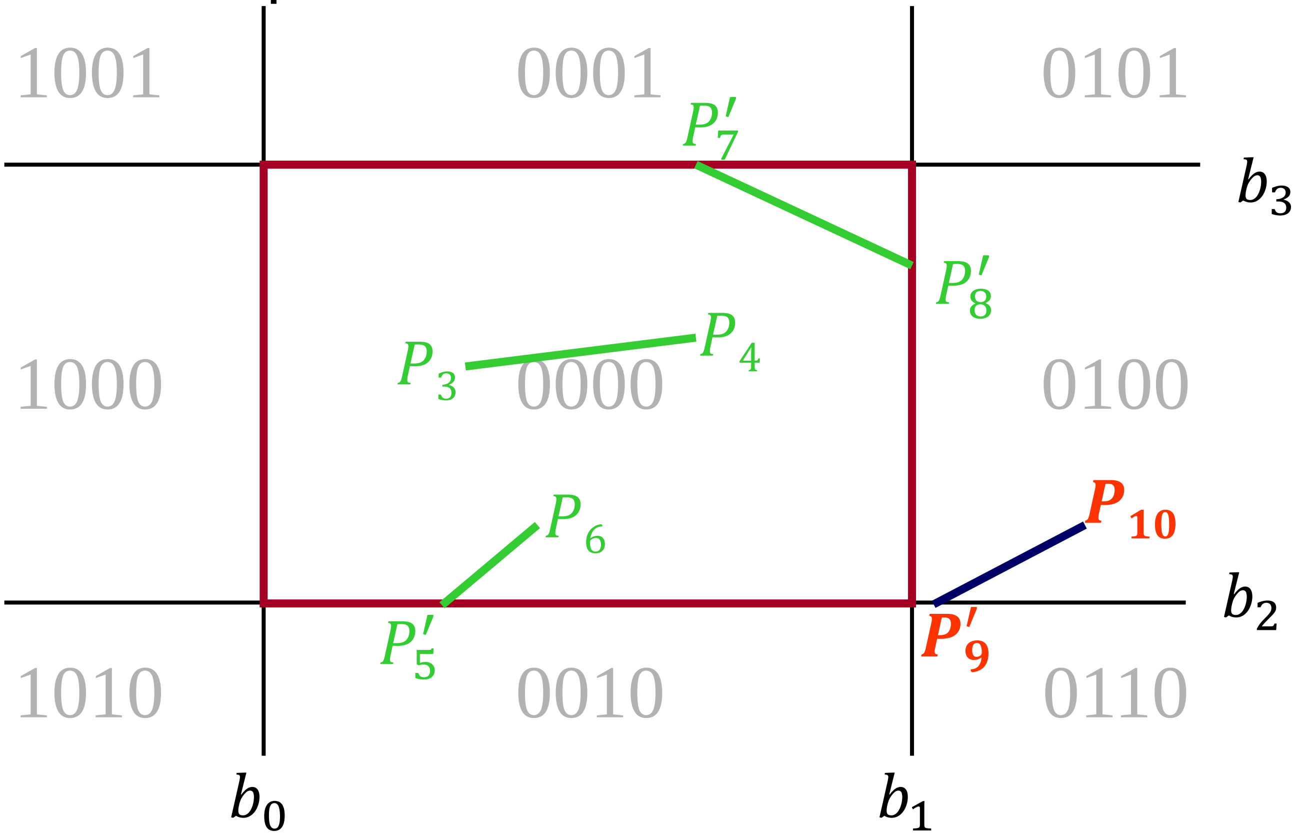
Cohen-Sutherland Line Clipping

- Associate a 4-bit <u>outcode</u> $b_0 b_1 b_2 b_3$ to each vertex
- If both points' outcodes are 0, line segment is inside
- If AND of outcodes is not 0, line segment is outside
- **Otherwise clip and test**



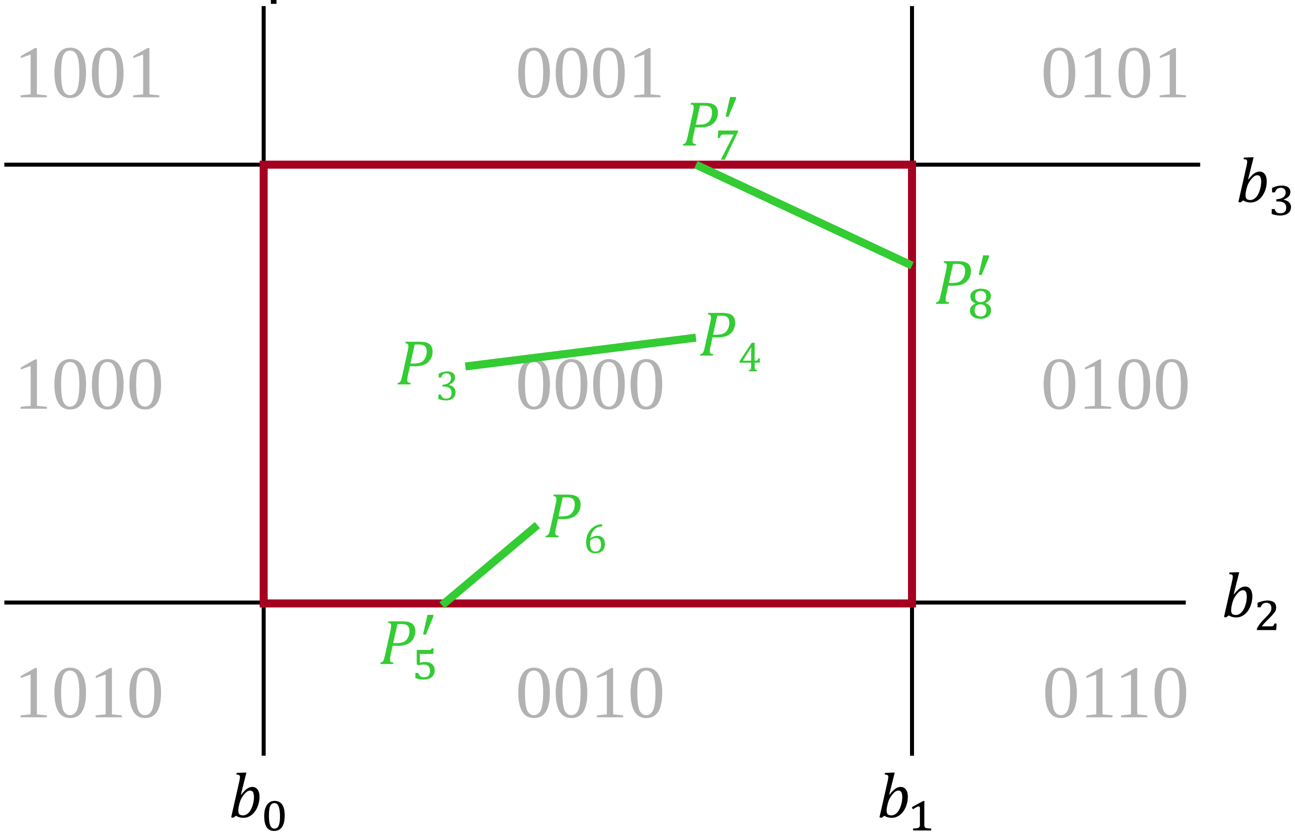
Cohen-Sutherland Line Clipping

- Associate a 4-bit <u>outcode</u> $b_0 b_1 b_2 b_3$ to each vertex
- If both points' outcodes are 0, line segment is inside
- **If AND of outcodes is not 0, line segment is outside**
- Otherwise clip and test



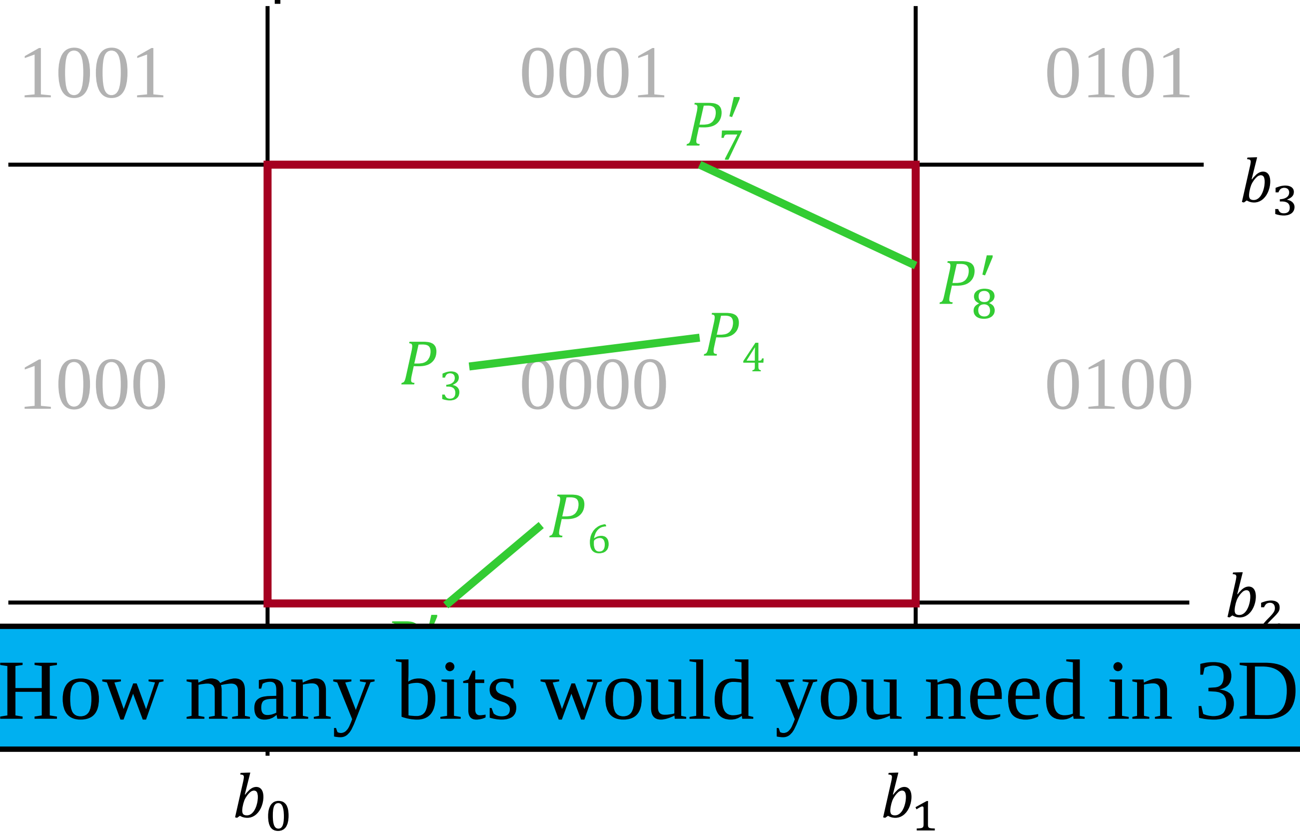
Cohen-Sutherland Line Clipping

- Associate a 4-bit <u>outcode</u> $b_0 b_1 b_2 b_3$ to each vertex
- If both points' outcodes are 0, line segment is inside
- **If AND of outcodes is not 0, line segment is outside**
- Otherwise clip and test



Cohen-Sutherland Line Clipping

- Associate a 4-bit outcode $b_0 b_1 b_2 b_3$ to each vertex
- If both points' outcodes are 0, line segment is inside
- If AND of outcodes is not 0, line segment is outside
- Otherwise clip and test



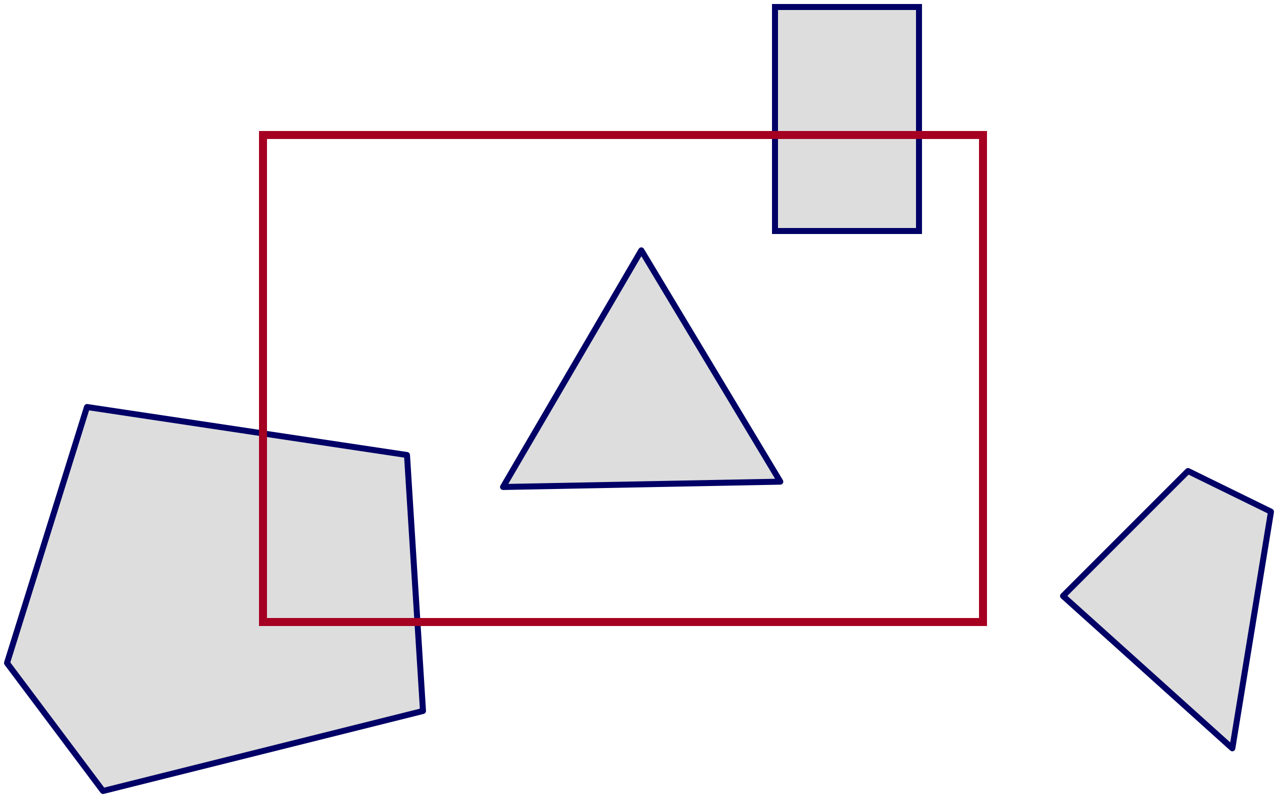
How many bits would you need in 3D?

Clipping

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

Polygon Clipping

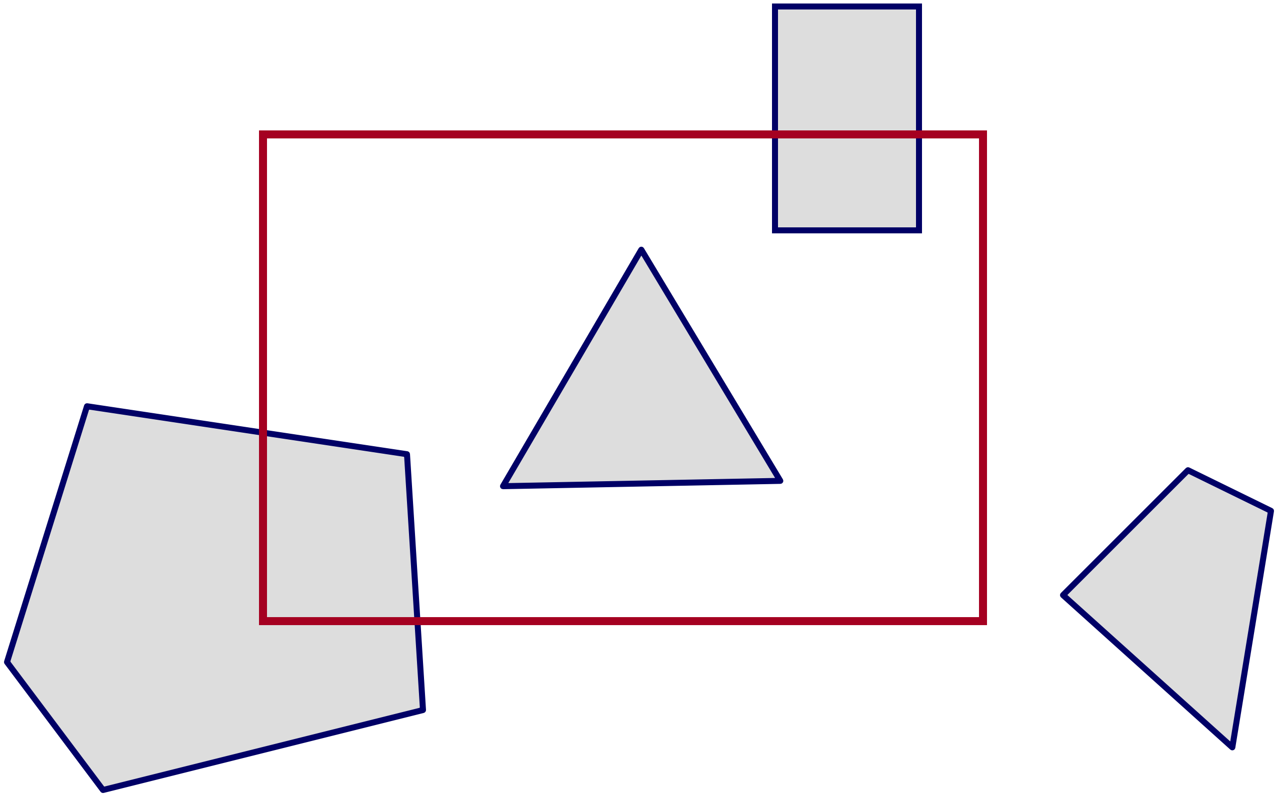
- Find the part of a polygon inside the clip window



Before Clipping

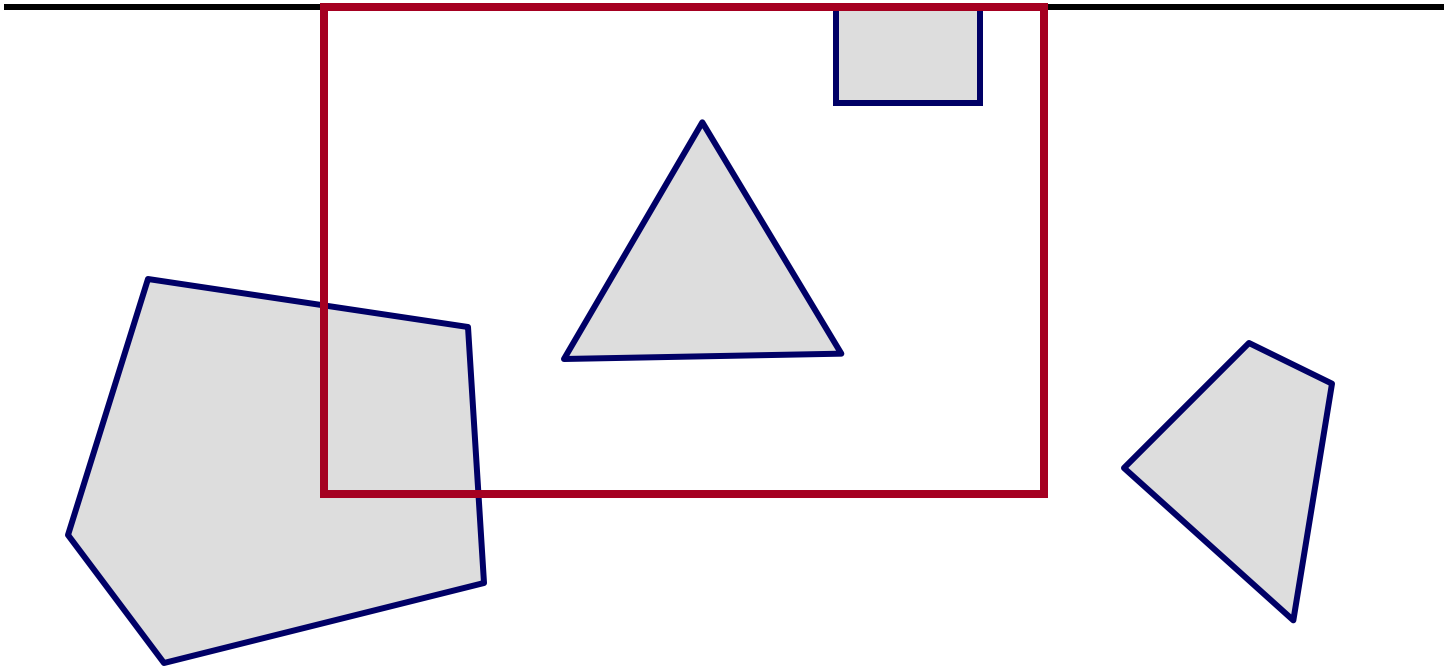
Sutherland-Hodgeman Clipping

- Clip to each window boundary, one at a time



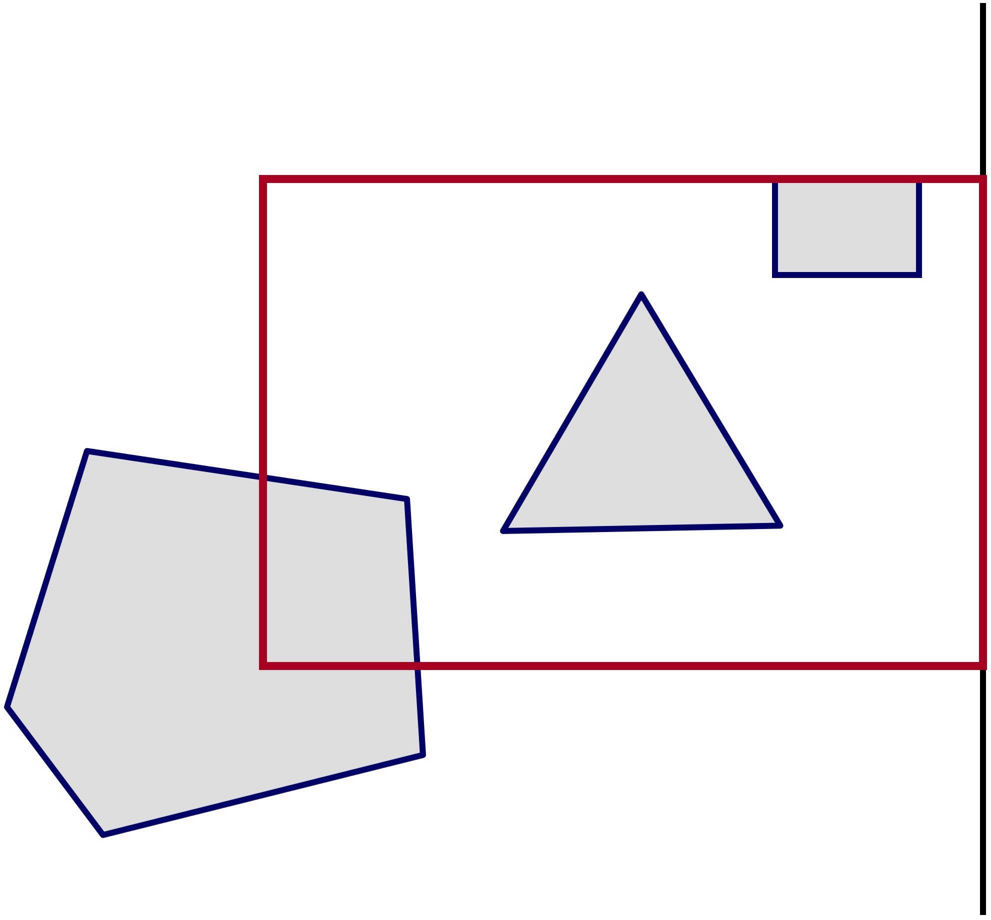
Sutherland-Hodgeman Clipping

- Clip to each window boundary, one at a time



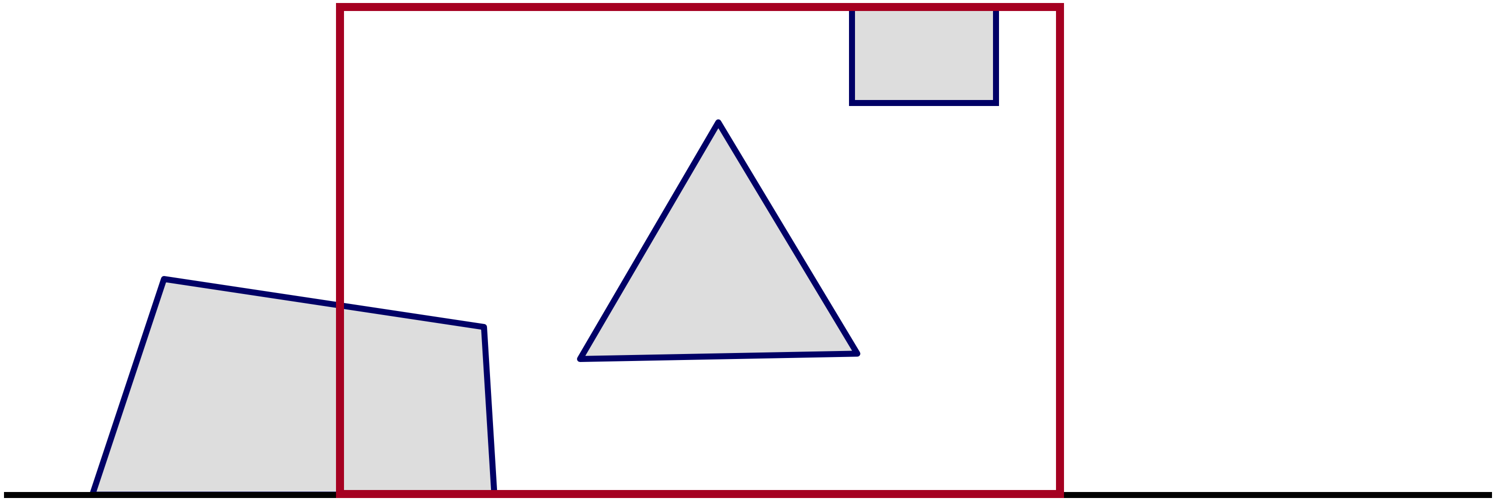
Sutherland-Hodgeman Clipping

- Clip to each window boundary, one at a time



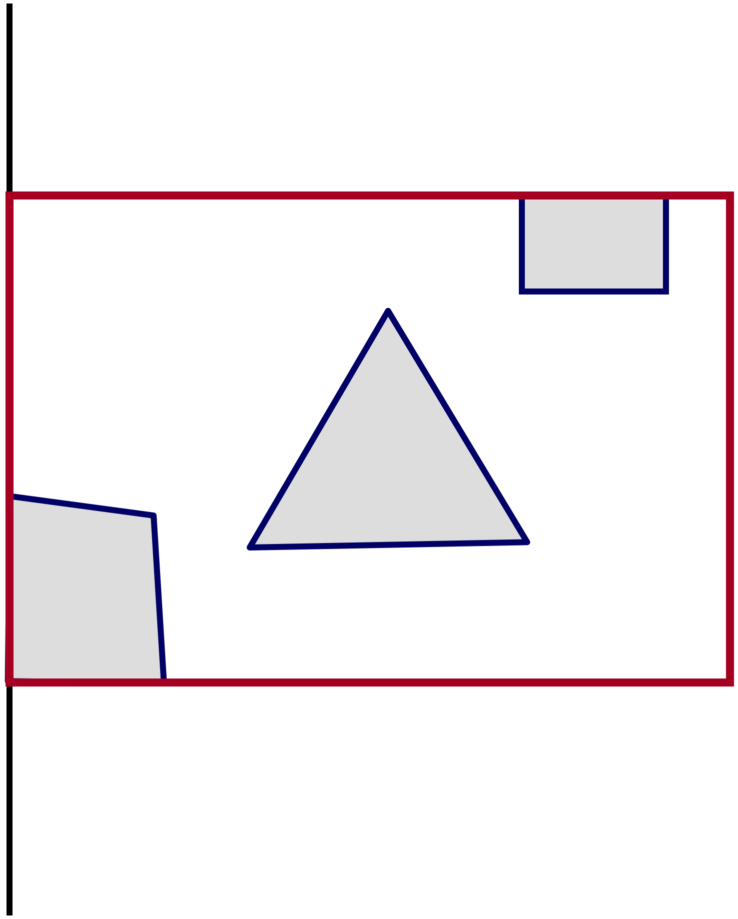
Sutherland-Hodgeman Clipping

- Clip to each window boundary, one at a time



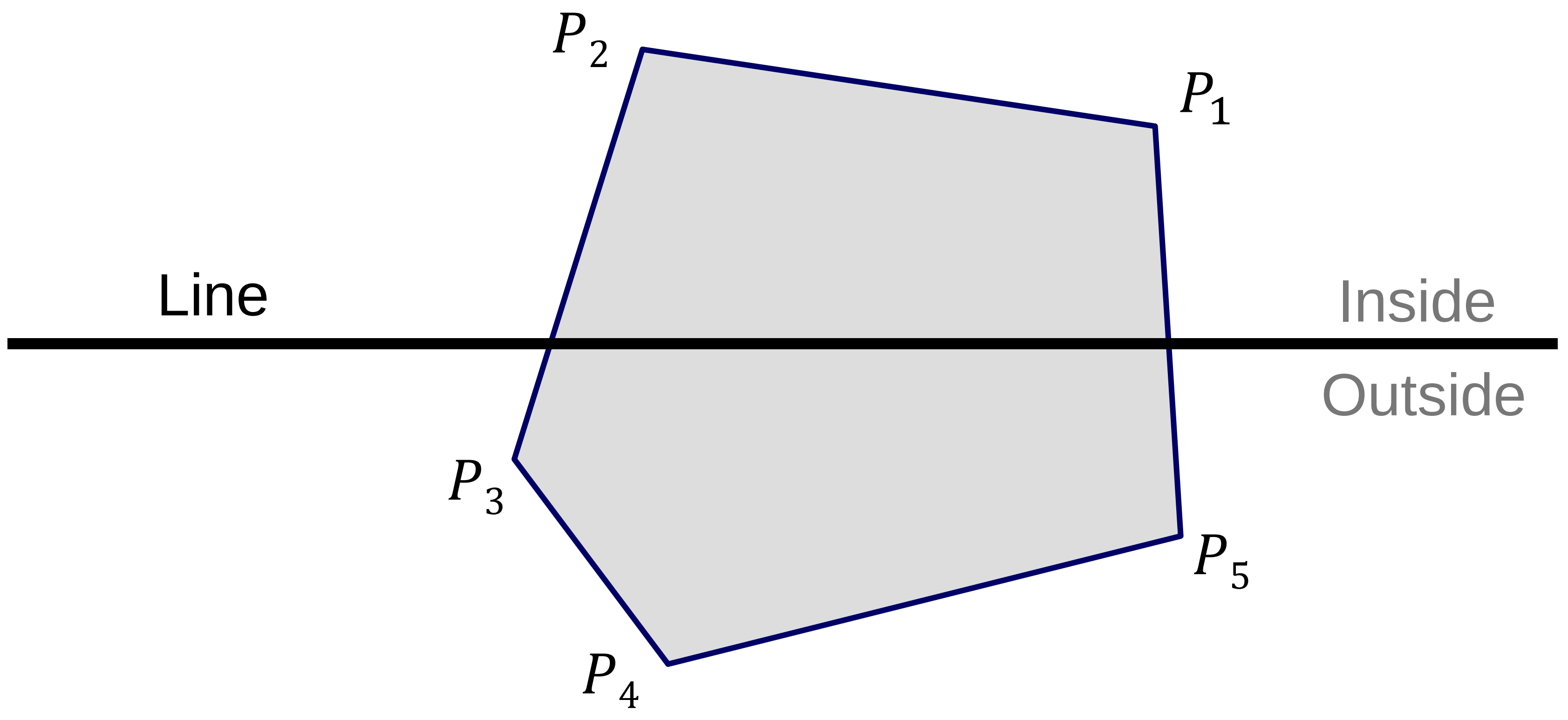
Sutherland-Hodgeman Clipping

- Clip to each window boundary, one at a time



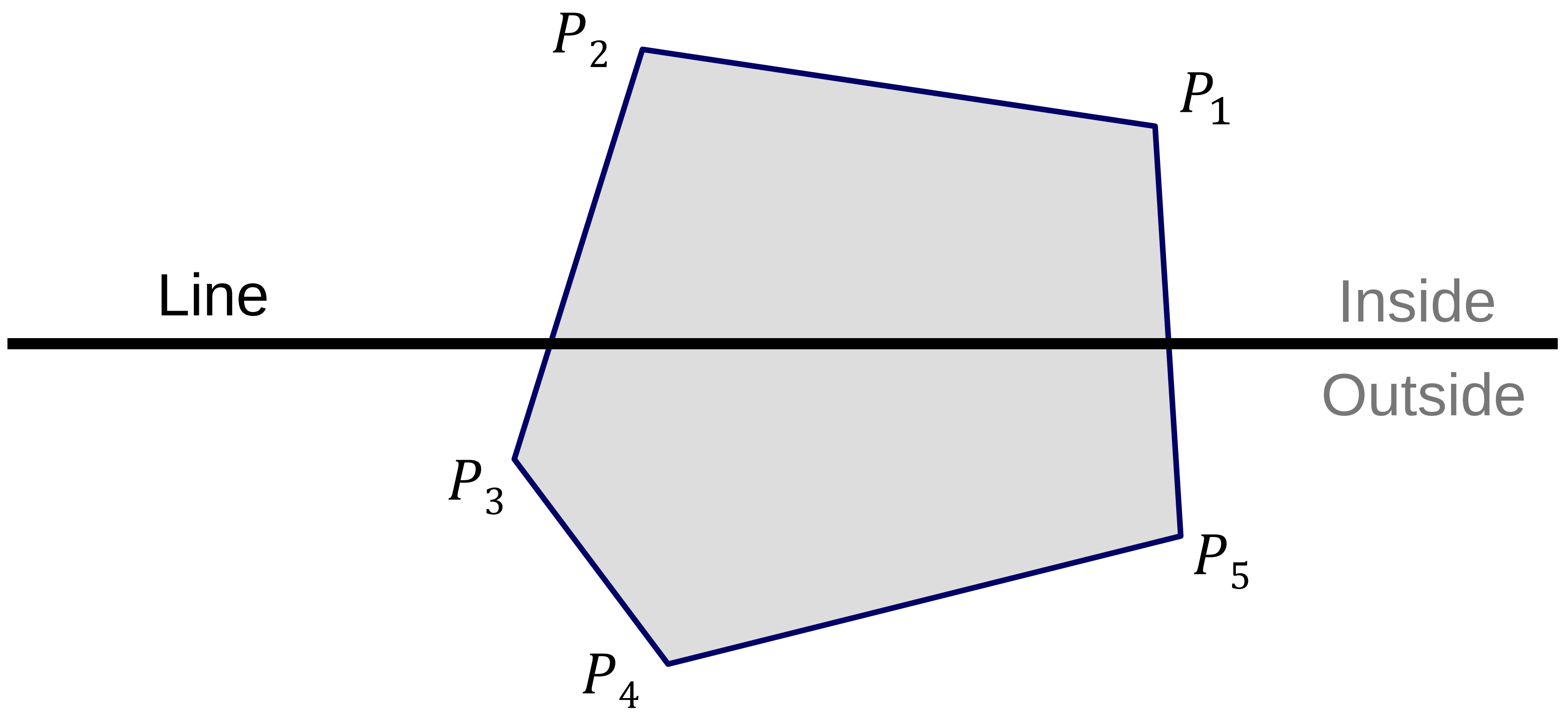
Sutherland-Hodgeman Clipping

- How do we clip a <u>convex</u> polygon with respect to a (window boundary) line?



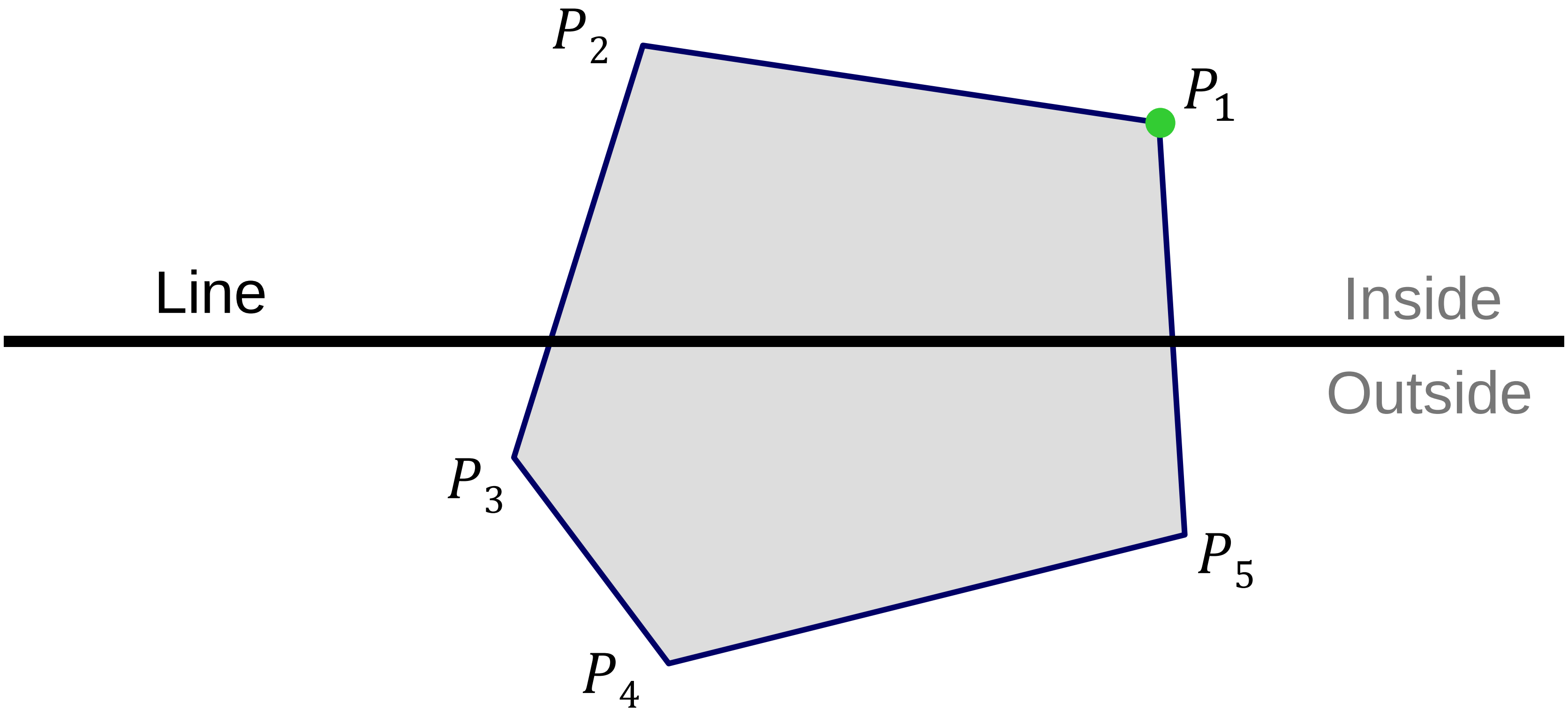
Sutherland-Hodgeman Clipping

- Do interior test for each point in sequence.
- Insert a new point when crossing the line.
- Remove points outside the line.



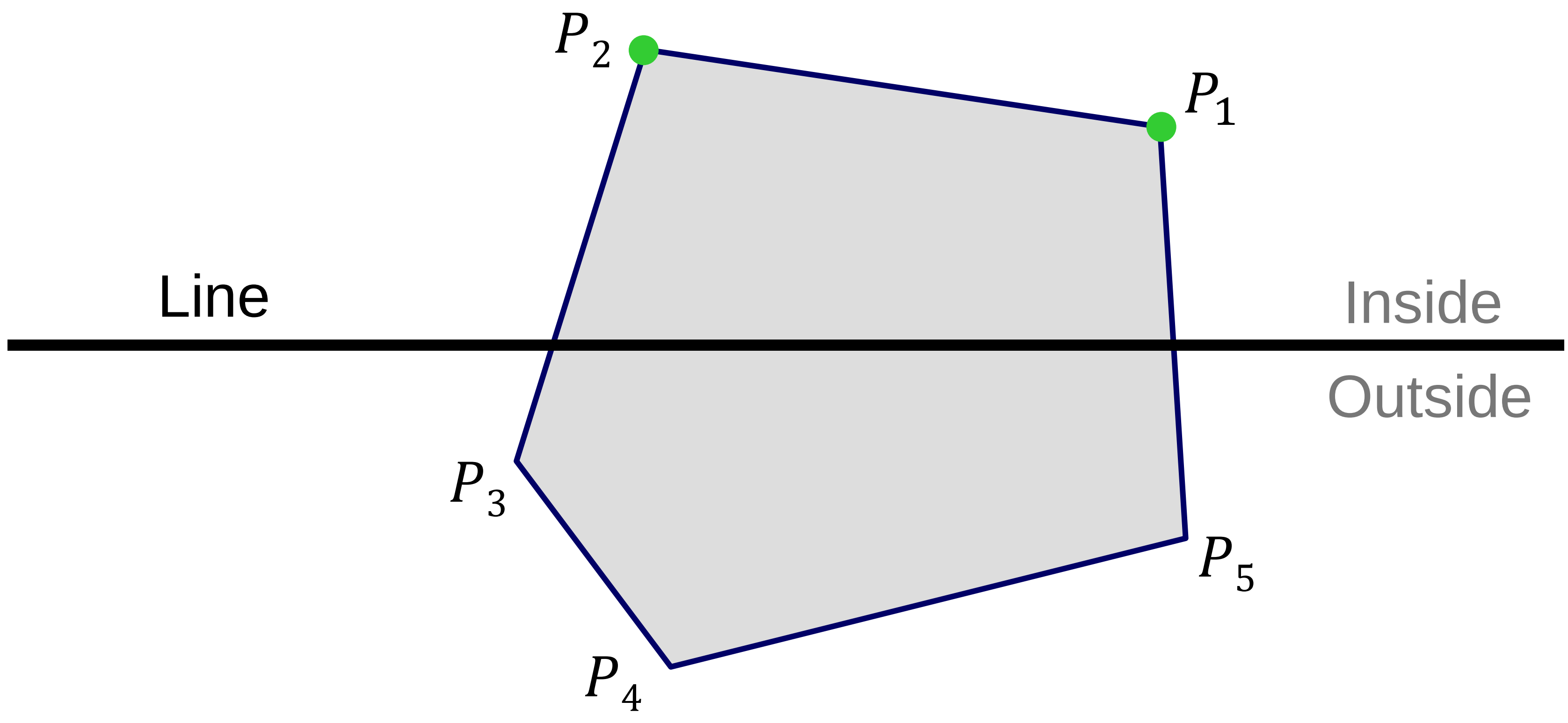
Sutherland-Hodgeman Clipping

- Do interior test for each point in sequence.
- Insert a new point when crossing the line.
- Remove points outside the line.



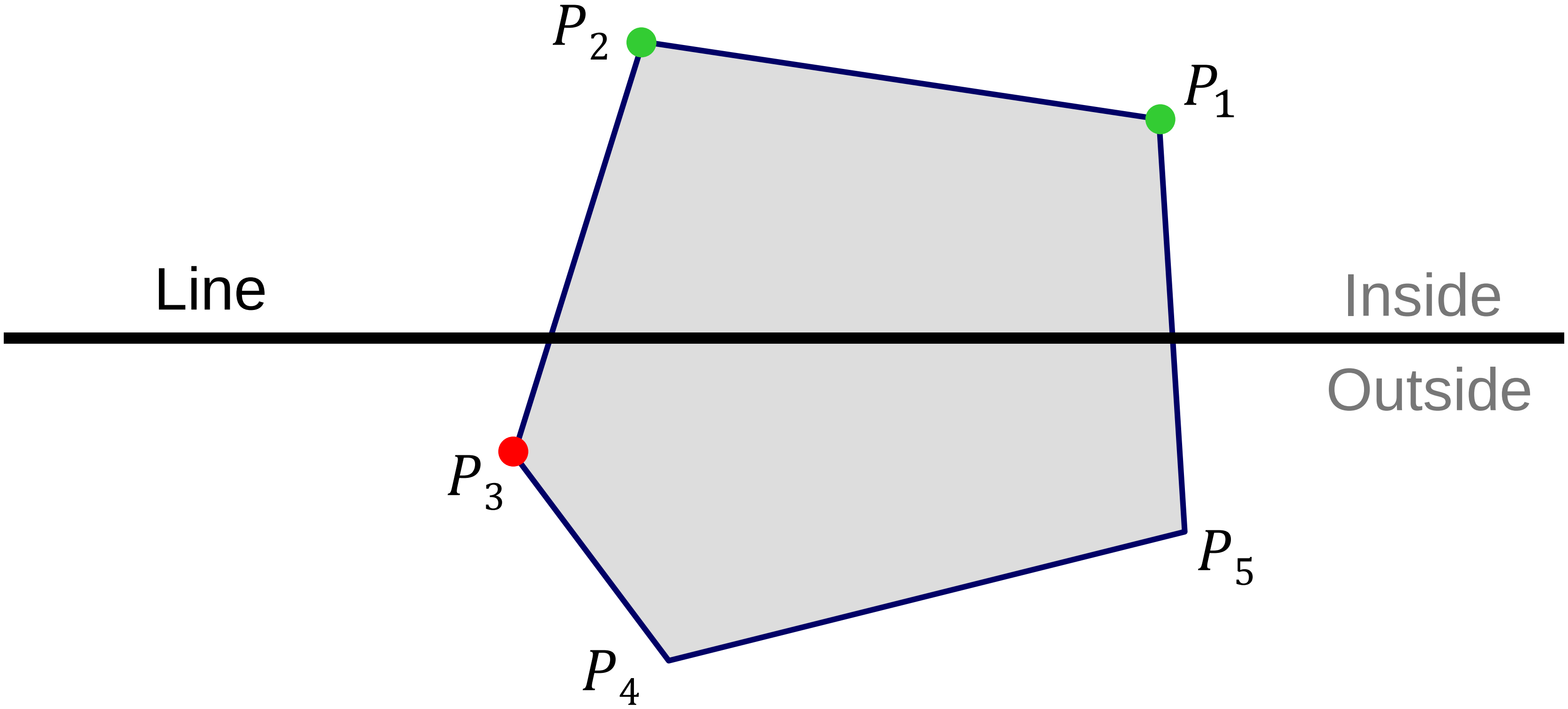
Sutherland-Hodgeman Clipping

- Do interior test for each point in sequence.
- Insert a new point when crossing the line.
- Remove points outside the line.



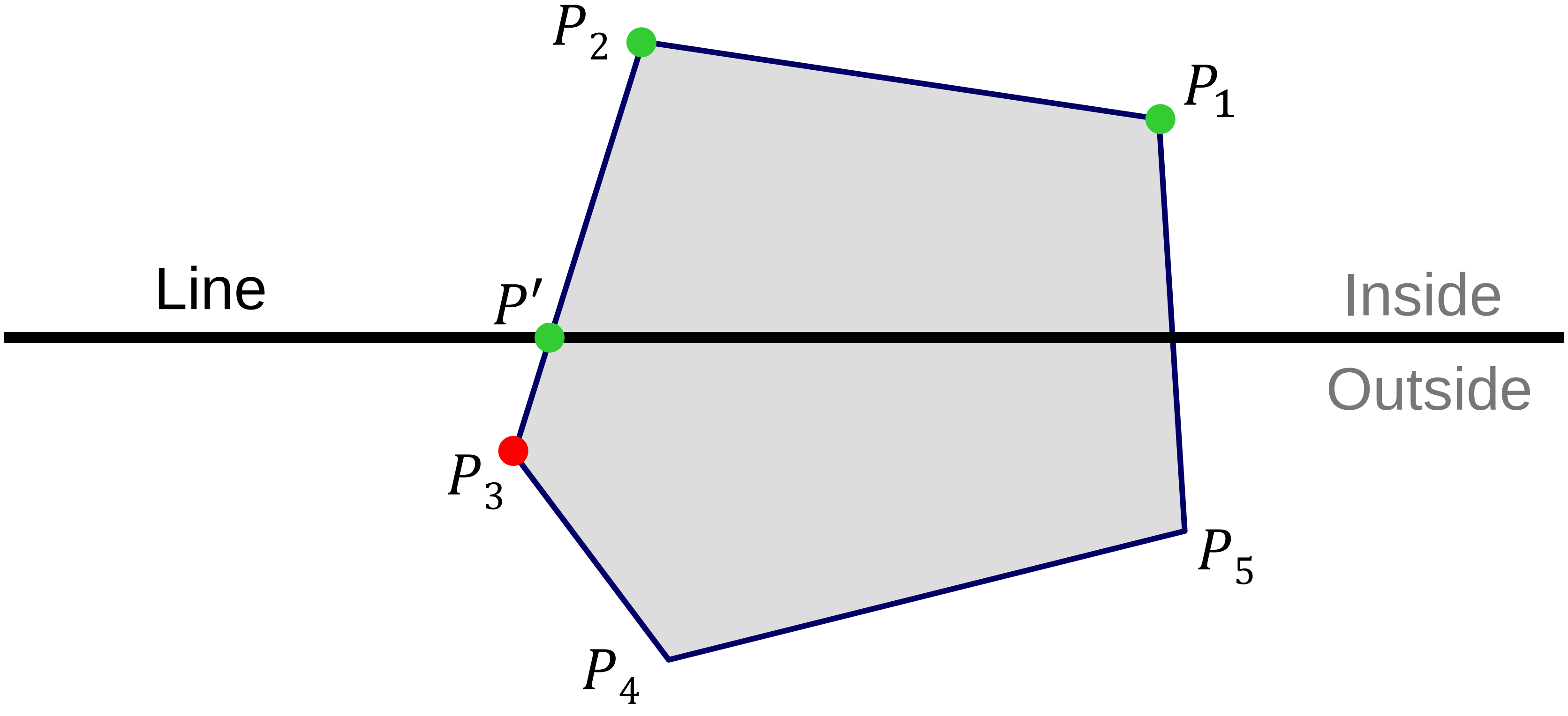
Sutherland-Hodgeman Clipping

- Do interior test for each point in sequence.
- Insert a new point when crossing the line.
- Remove points outside the line.



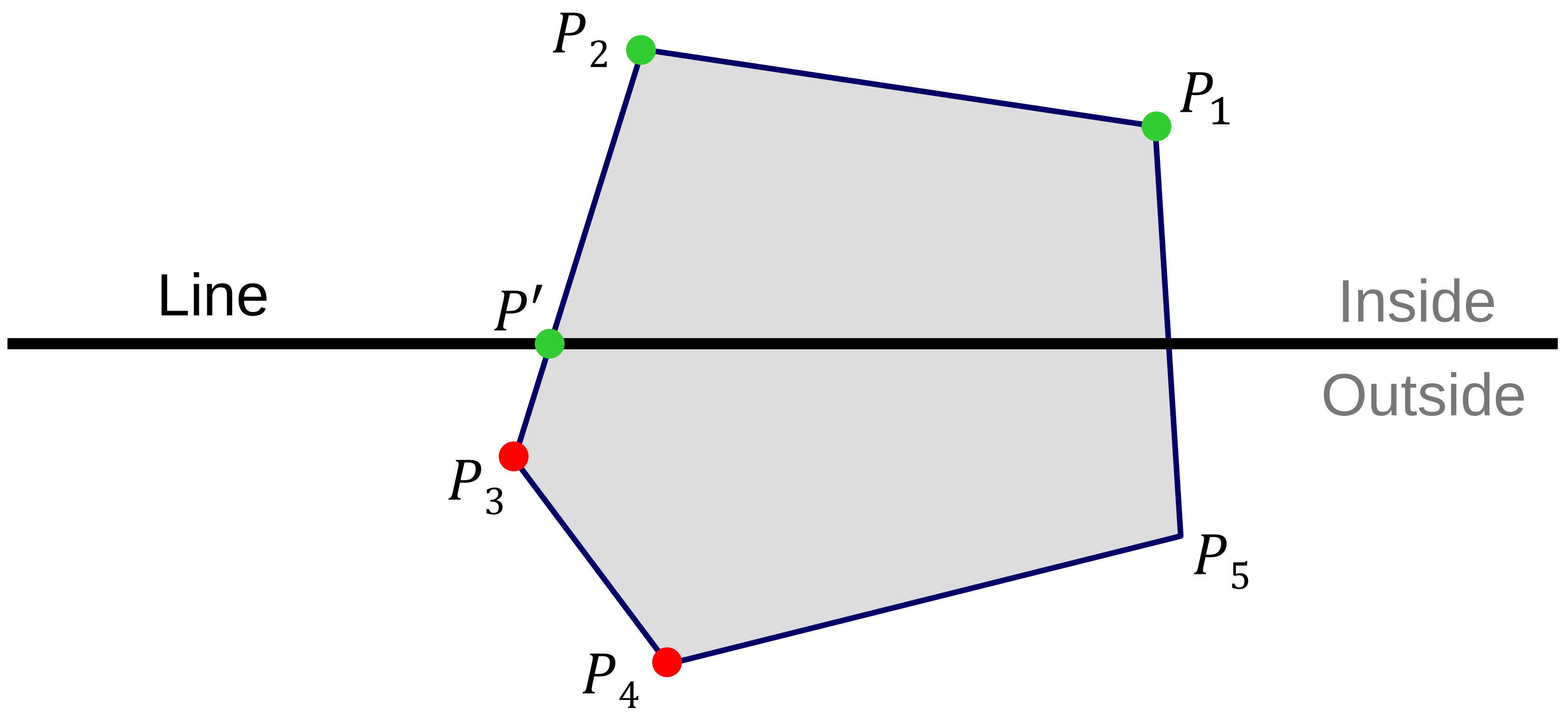
Sutherland-Hodgeman Clipping

- Do interior test for each point in sequence.
- Insert a new point when crossing the line.
- Remove points outside the line.



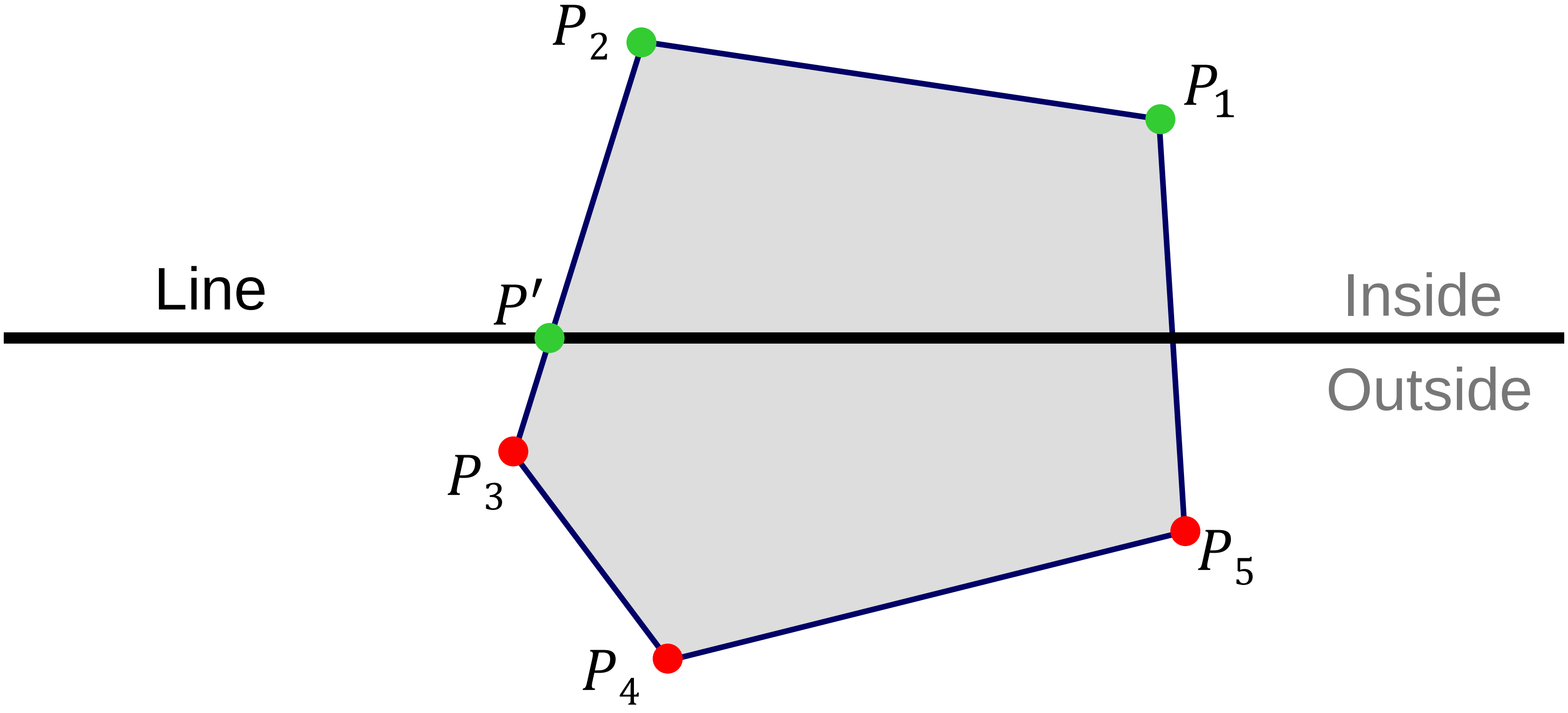
Sutherland-Hodgeman Clipping

- Do interior test for each point in sequence.
- Insert a new point when crossing the line.
- Remove points outside the line.



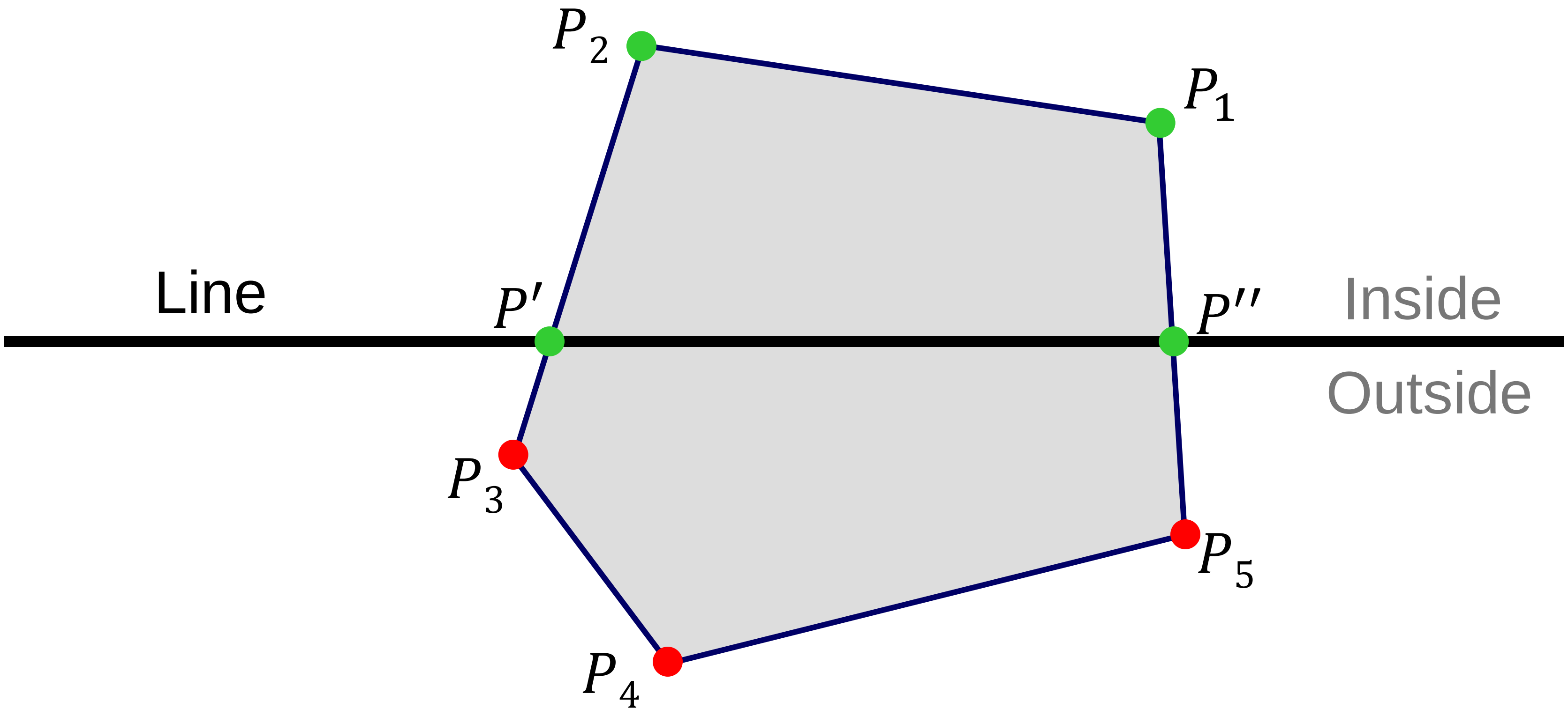
Sutherland-Hodgeman Clipping

- Do interior test for each point in sequence.
- Insert a new point when crossing the line.
- Remove points outside the line.



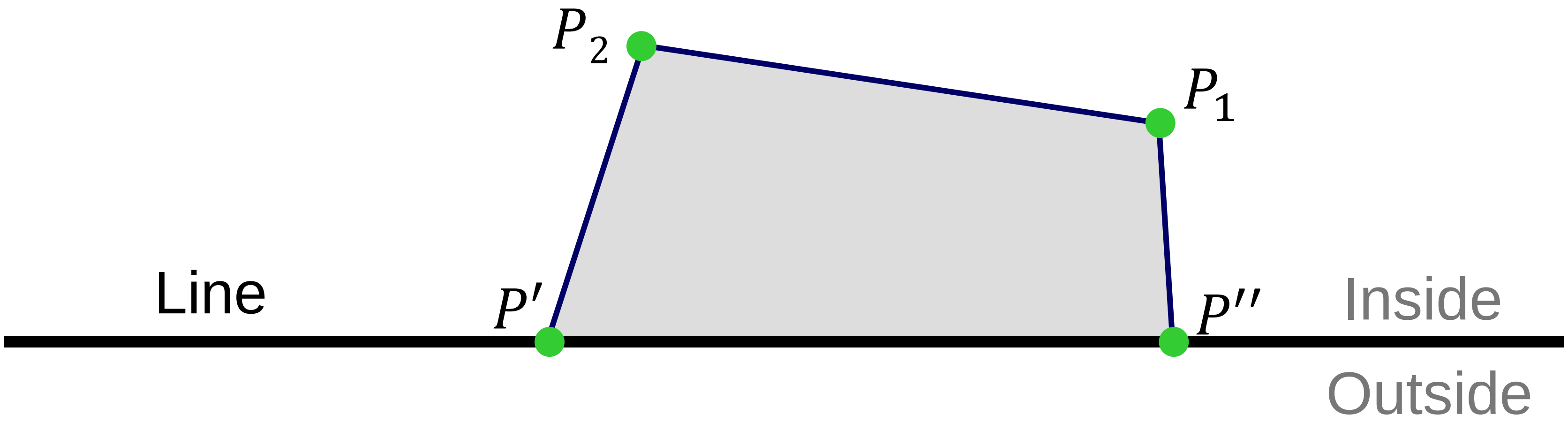
Sutherland-Hodgeman Clipping

- Do interior test for each point in sequence.
- Insert a new point when crossing the line.
- Remove points outside the line.



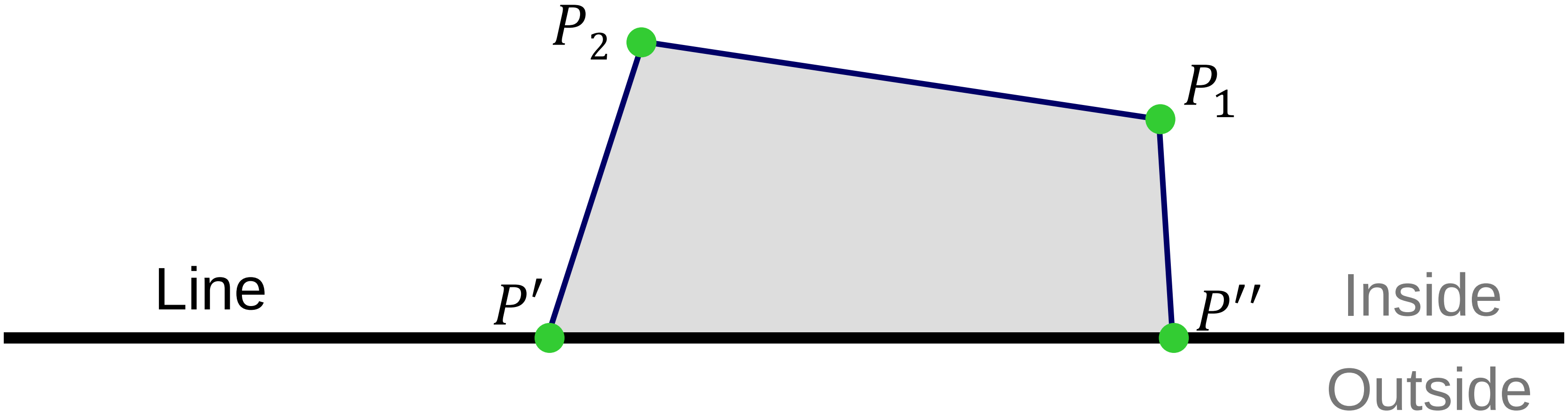
Sutherland-Hodgeman Clipping

- Do interior test for each point in sequence.
- Insert a new point when crossing the line.
- Remove points outside the line.



Sutherland-Hodgeman Clipping

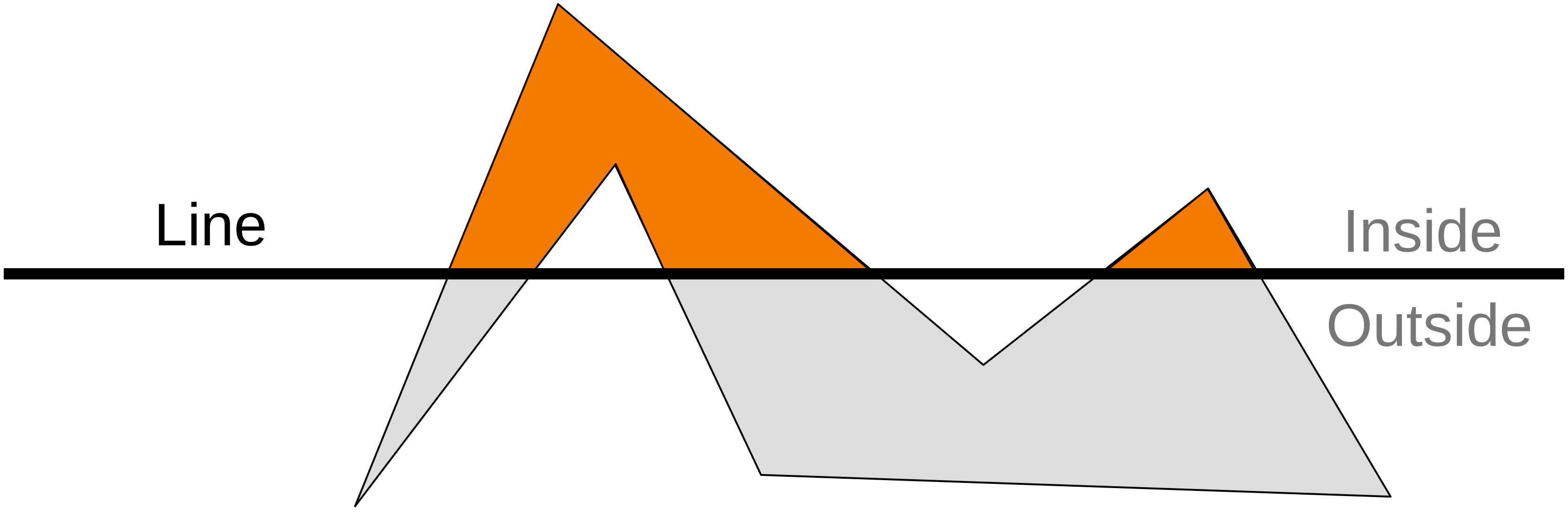
- Do interior test for each point in sequence.
- Insert a new point when crossing the line.
- Remove points outside the line.



When polygons are clipped, per-vertex properties (e.g. lighting) is interpolated to the new vertices.

Sutherland-Hodgeman Clipping

- Do interior test for each point in sequence.
- Insert a new point when crossing the line.
- Remove points outside the line.



[WARNING] If the polygon is not convex, we may end up with more than one polygon!

3D Rendering Pipeline (for direct illumination)

3D Primitives

↓ 3D Modeling Coordinates

Modeling Transformation

↓ 3D World Coordinates

Viewing Transformation

↓ 3D Camera Coordinates

Lighting

↓ 3D Camera Coordinates

Projection Transformation

↓ 2D Window Coordinates

Clipping

↓ 2D Window Coordinates

Viewport Transformation

↓ 2D Viewport Coordinates

Scan Conversion

↓ 2D Viewport Coordinates

Image

At this point we have the:

- Positions of the mesh vertices (including new vertices obtained through clipping)
- Color information at each vertex.
- A list of (possibly clipped) polygons describing the intersection of the projected 3D polygons with the window.