



3D Polygon Rendering Pipeline

Misha Kazhdan

(601.457/657)



3D Polygon Rendering

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



Assassin's Creed Shadows

(Ubisoft Quebec, 2025)



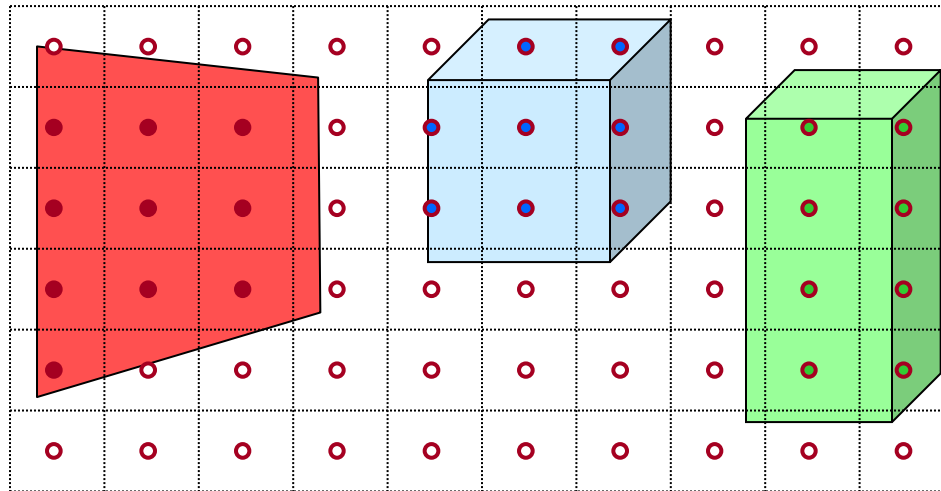
Ray Casting

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



- Send 2D pixels into the scene and get color

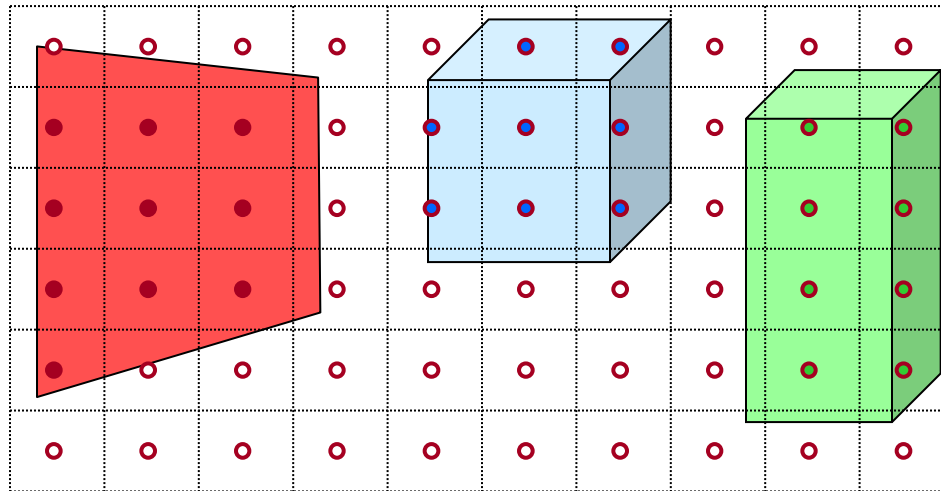




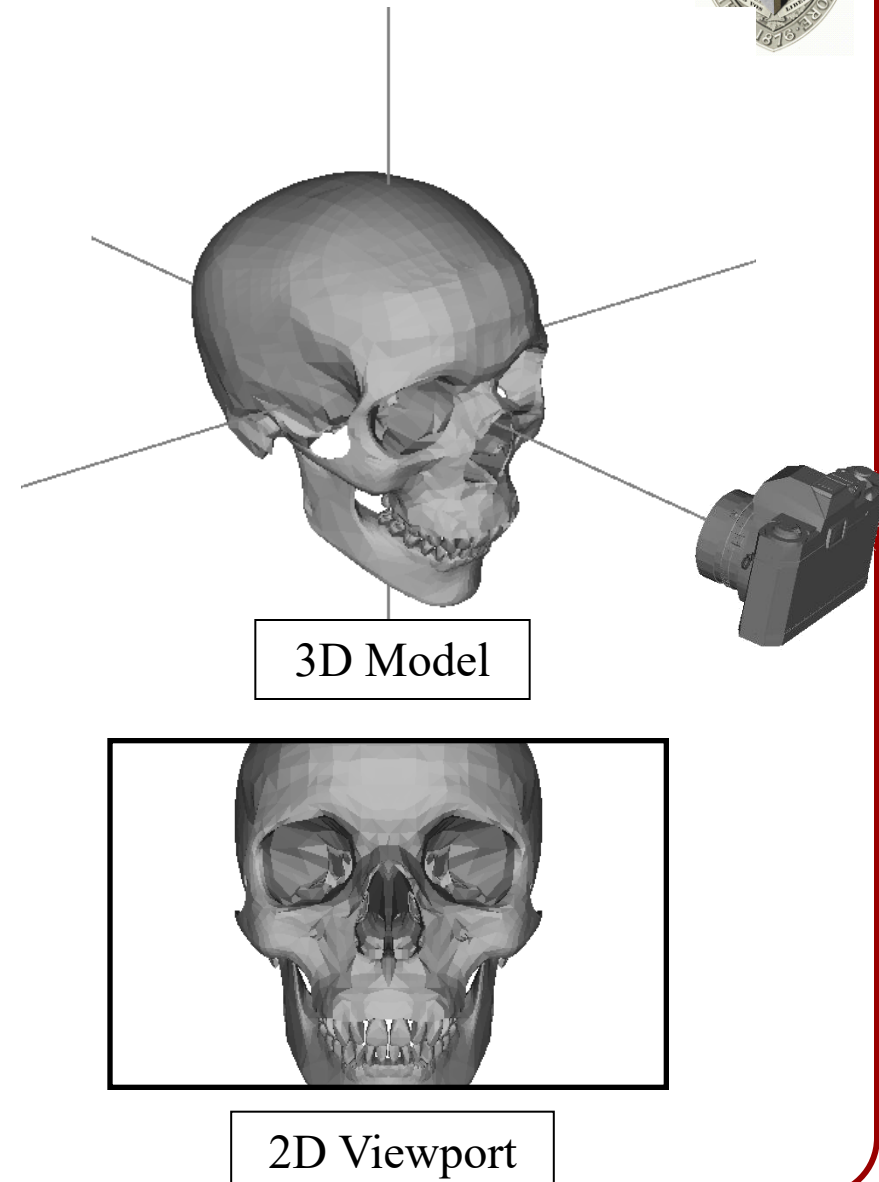
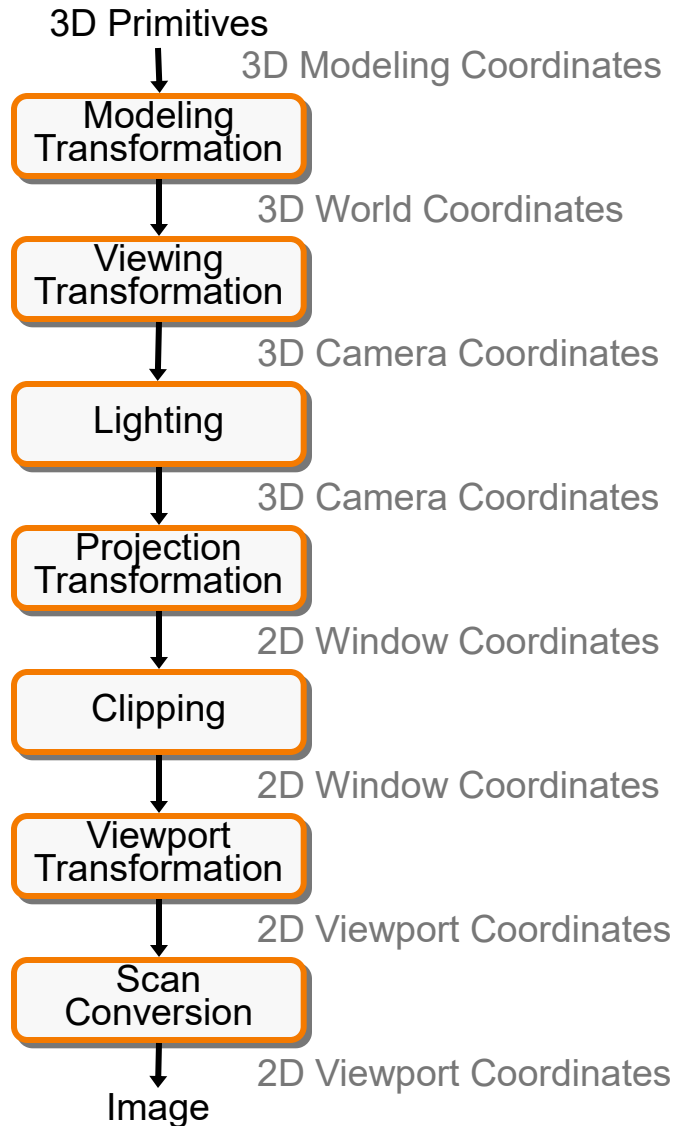
3D Polygon Rendering

For each primitive:

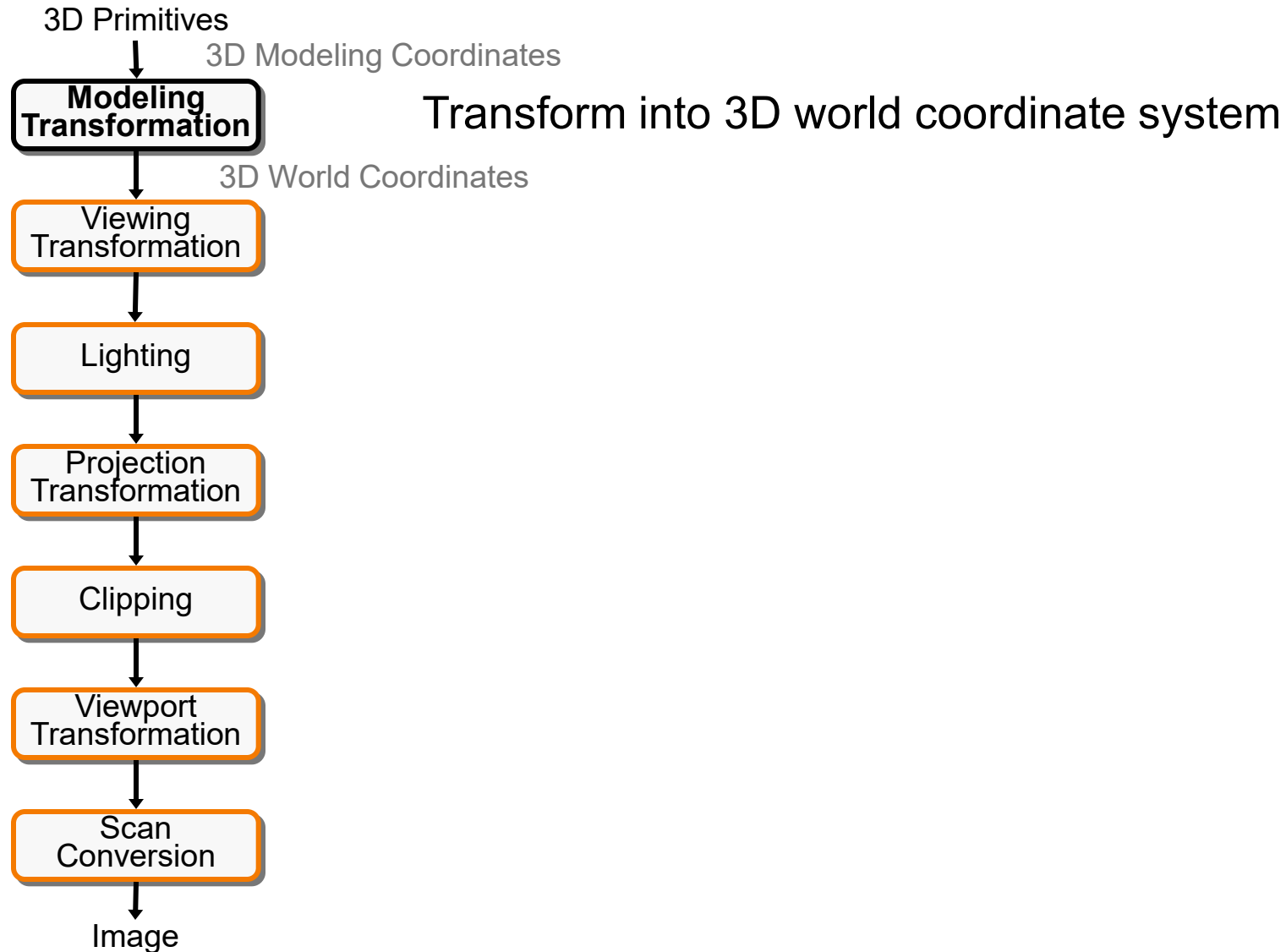
- Send 3D points to the camera and set the pixel color



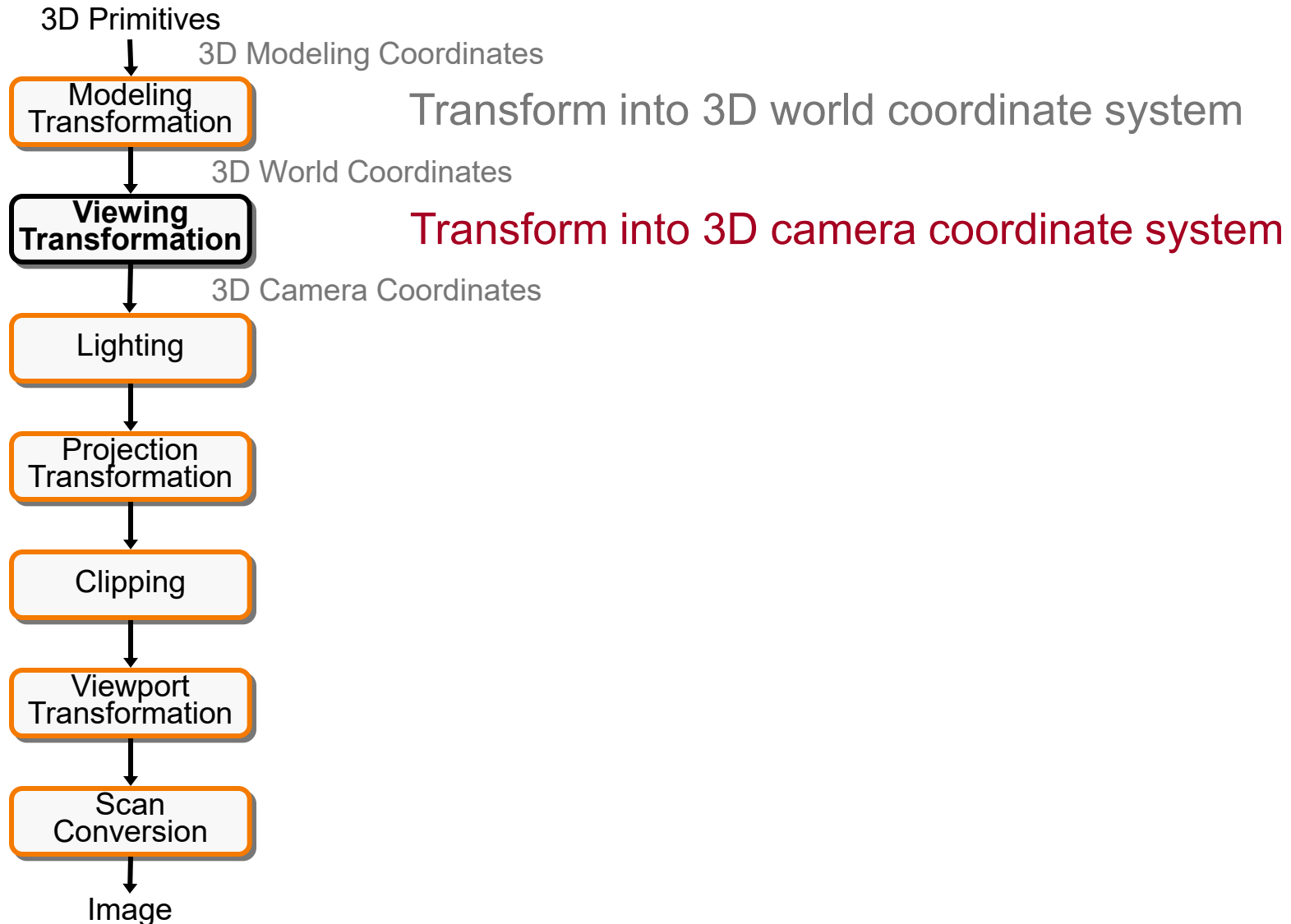
3D Rendering Pipeline (for direct illumination)



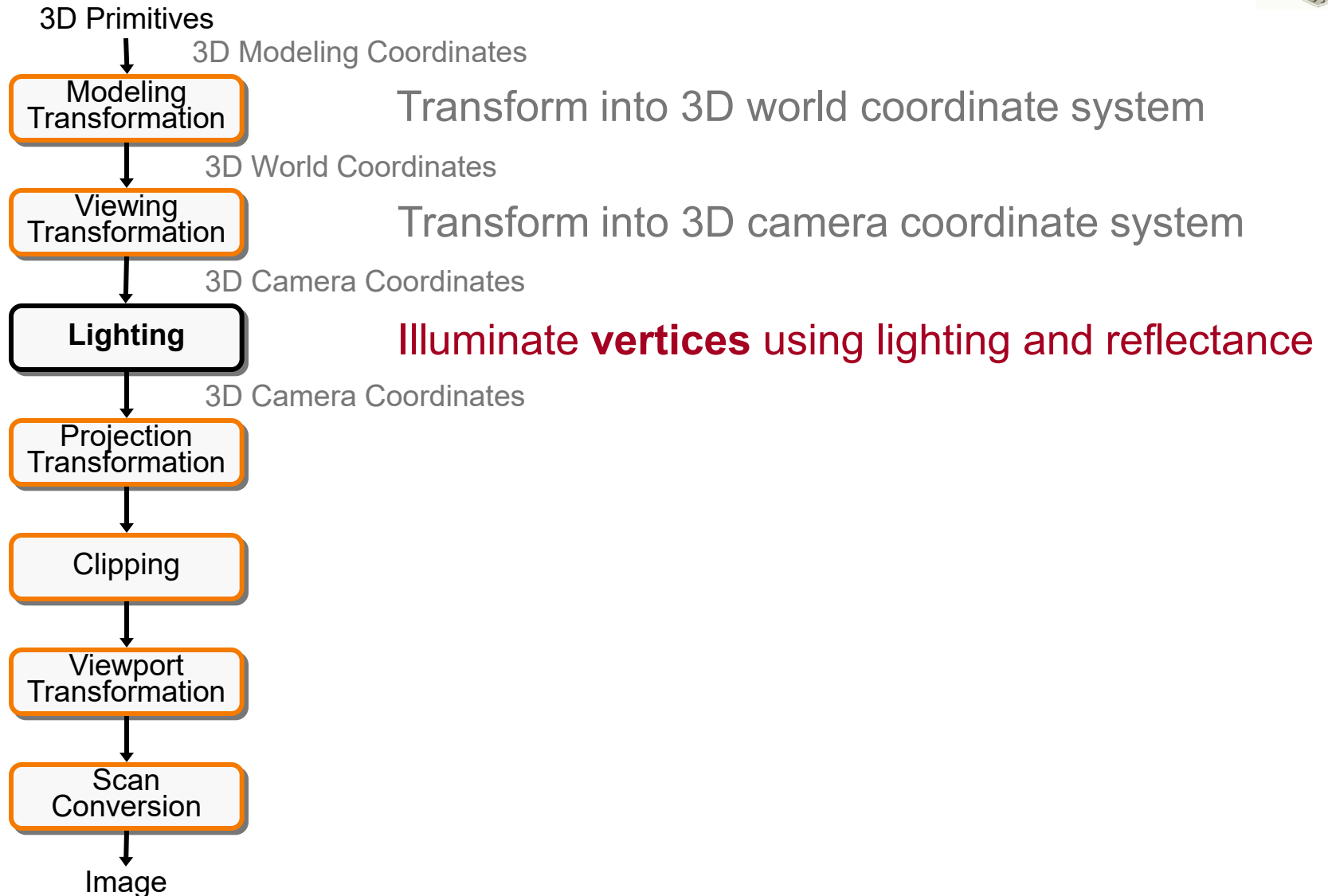
3D Rendering Pipeline (for direct illumination)



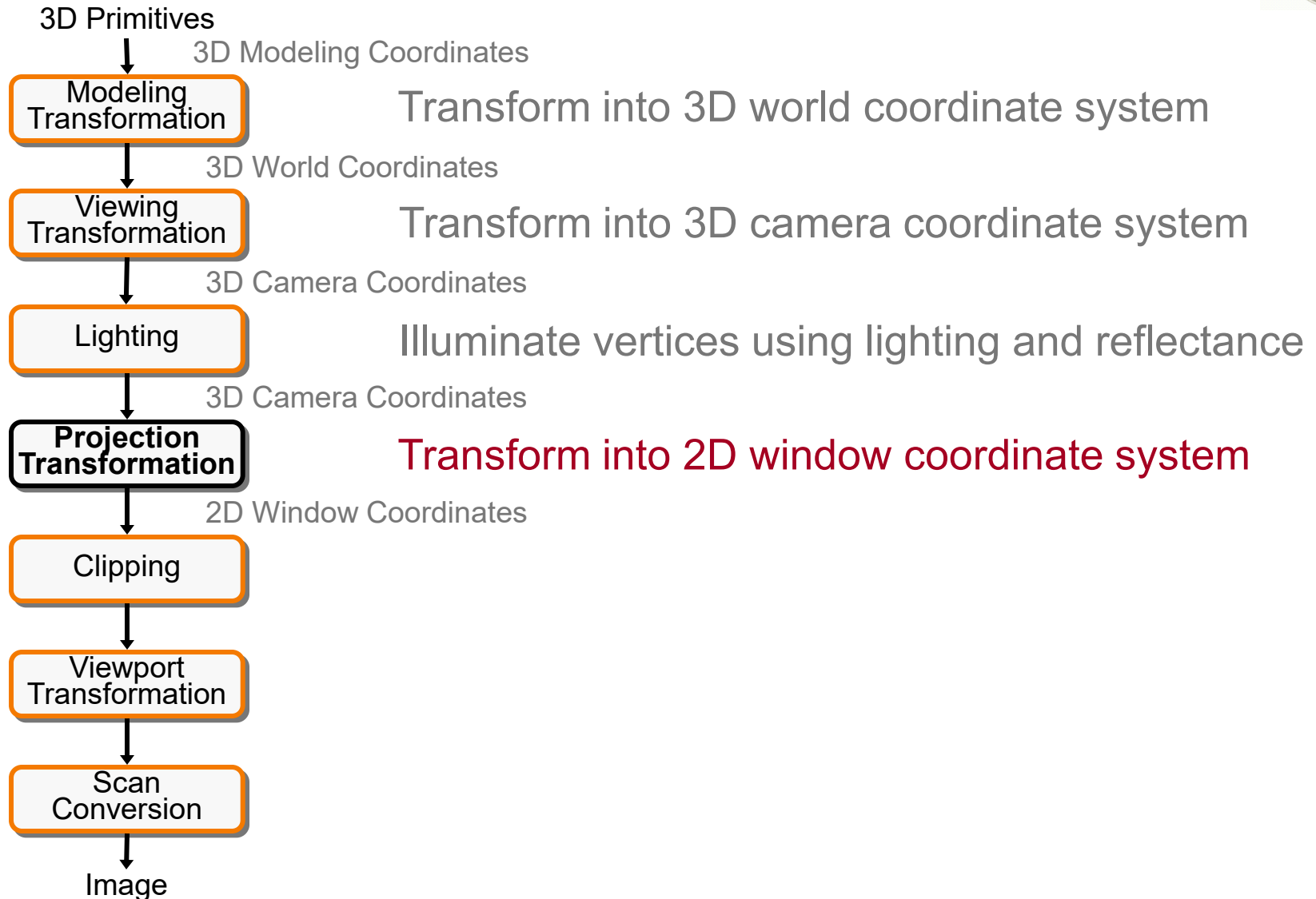
3D Rendering Pipeline (for direct illumination)



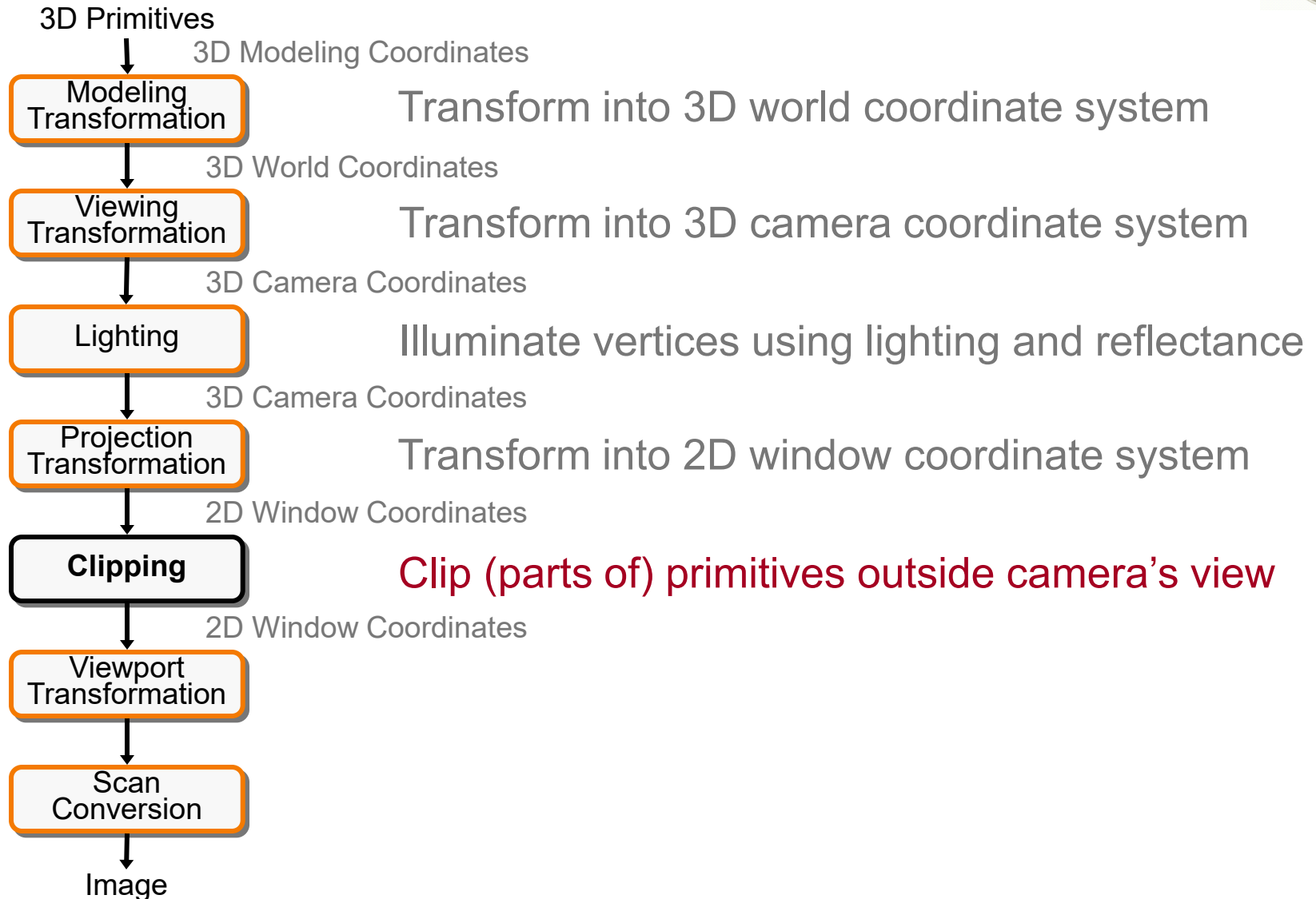
3D Rendering Pipeline (for direct illumination)



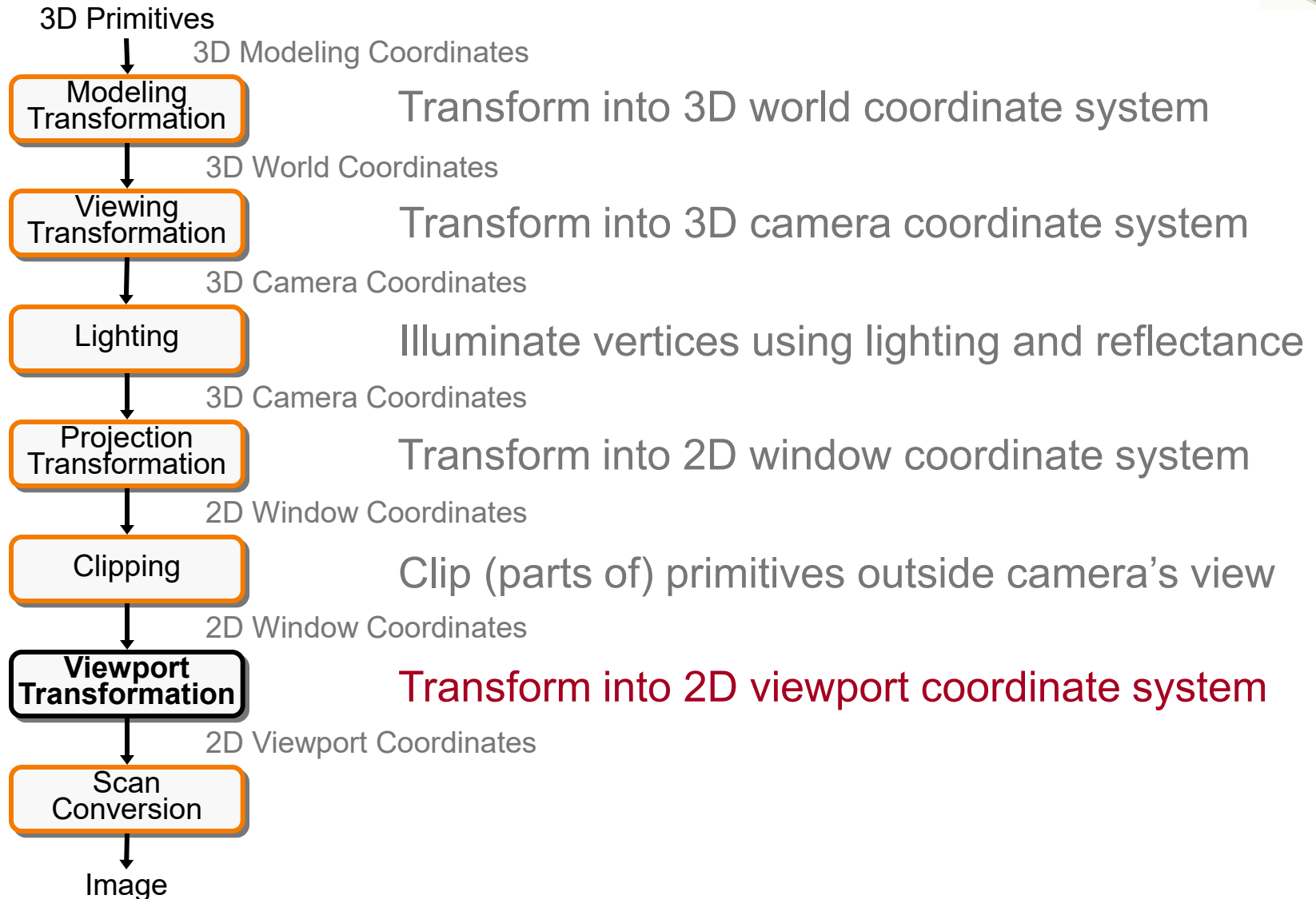
3D Rendering Pipeline (for direct illumination)



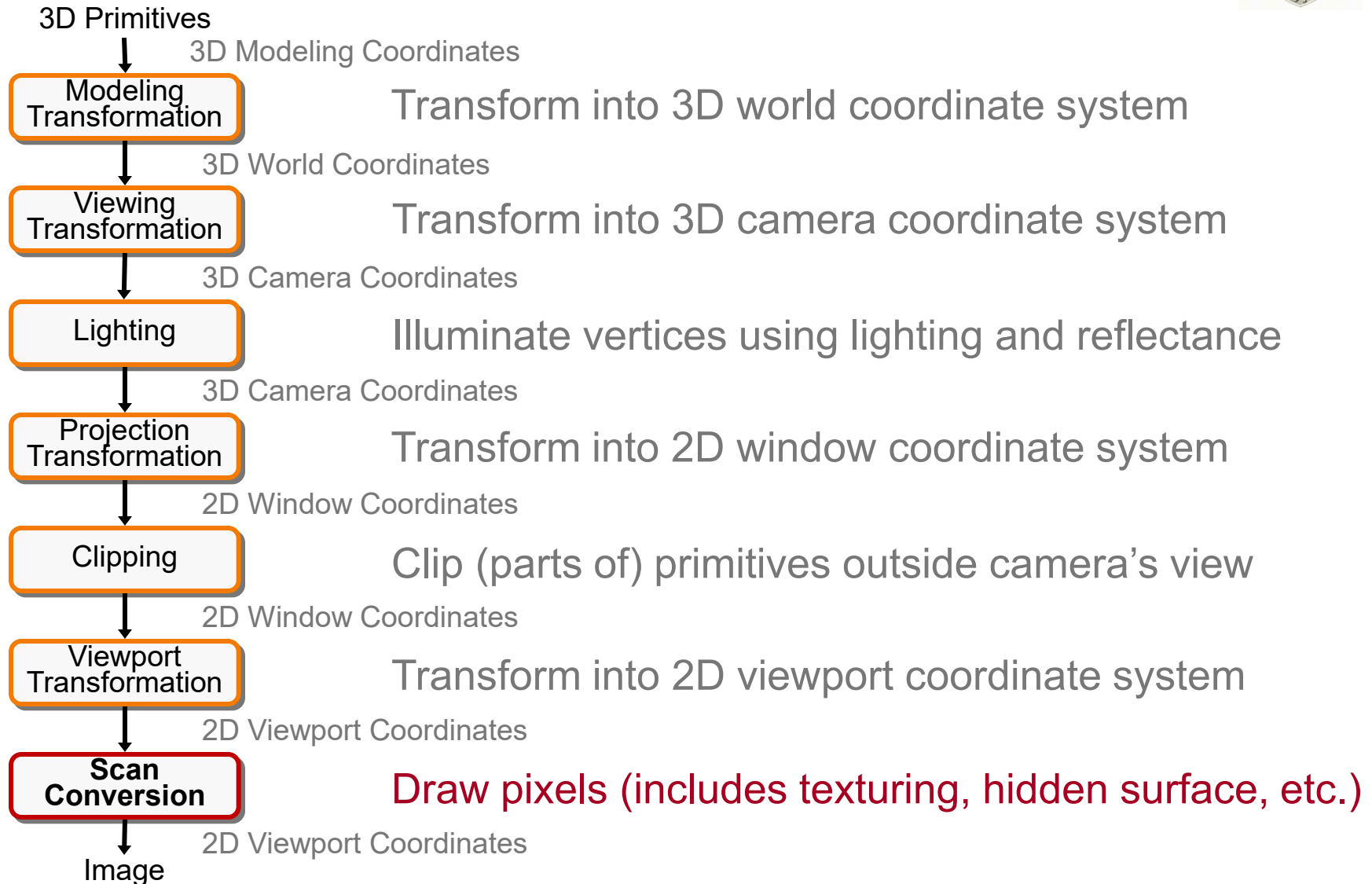
3D Rendering Pipeline (for direct illumination)



3D Rendering Pipeline (for direct illumination)

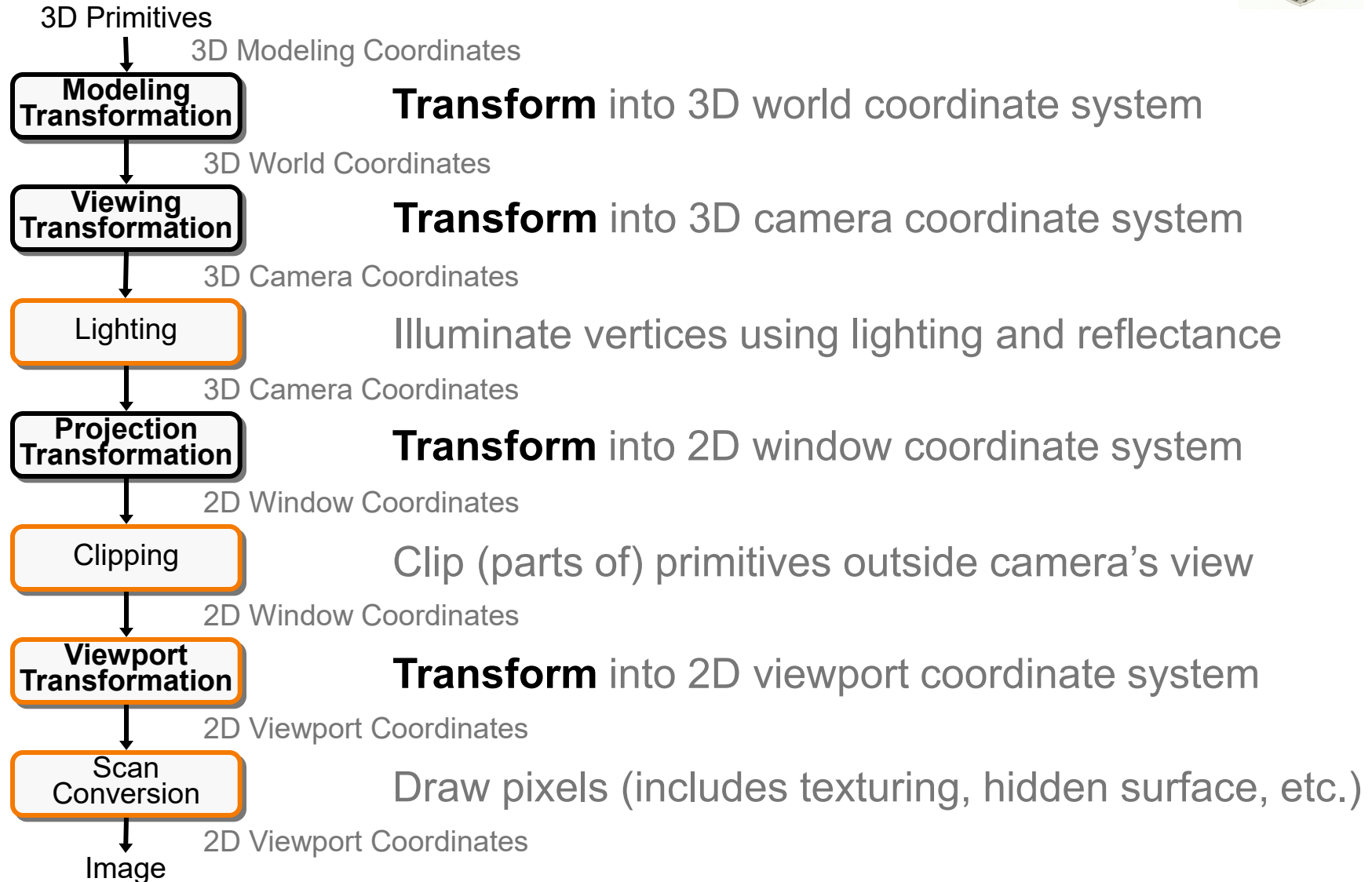


3D Rendering Pipeline (for direct illumination)





Transformations



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

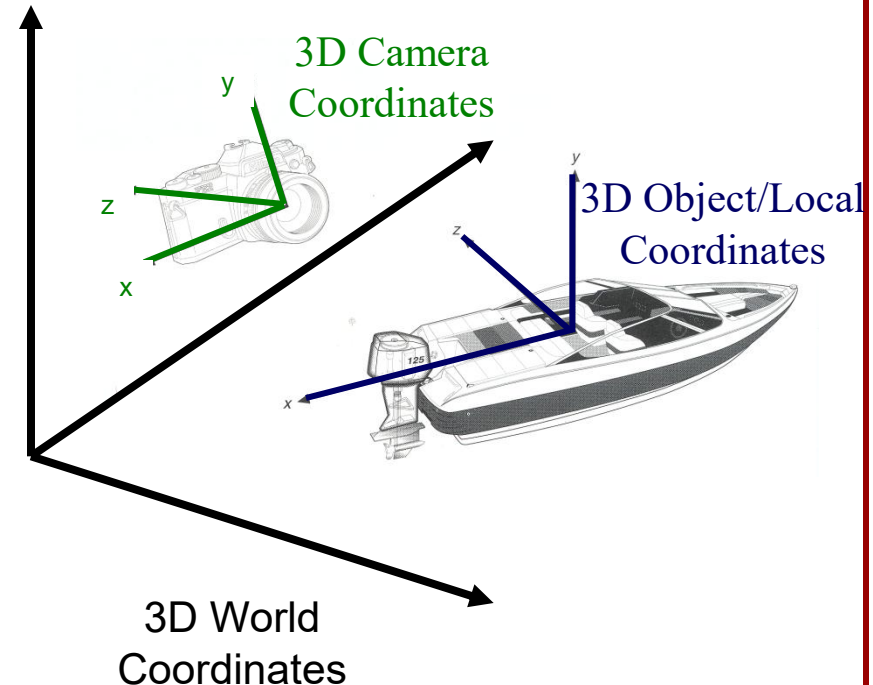
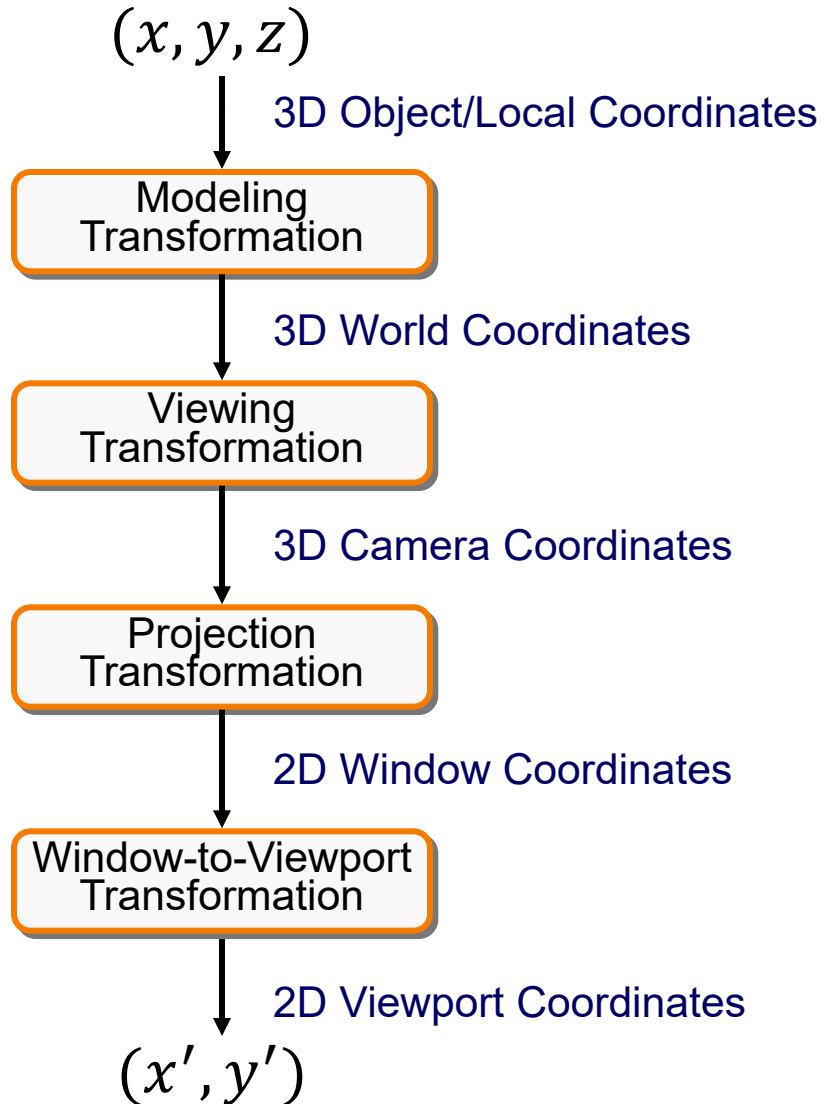
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ \boxed{0 \quad 0 \quad 0 \quad 1} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- Projective

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

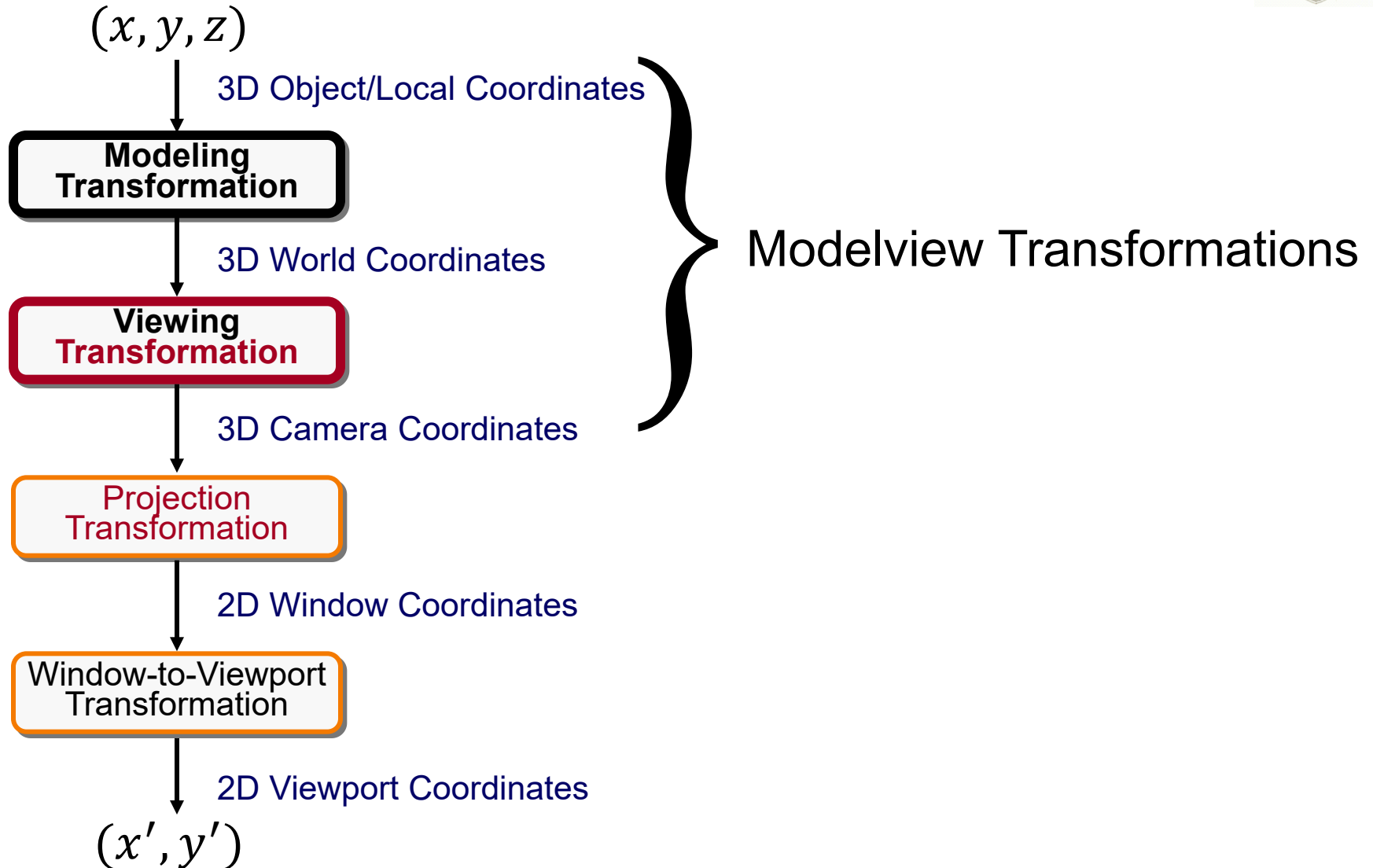


Transformations



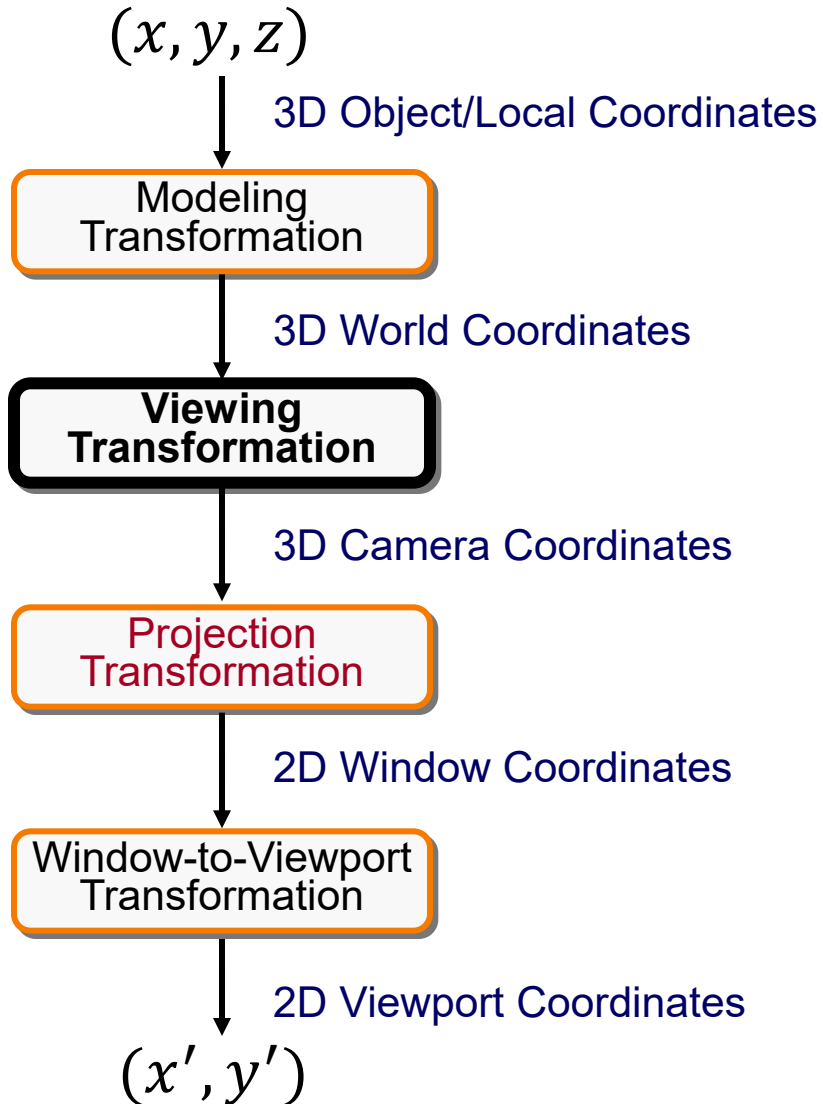


Transformations





Transformations

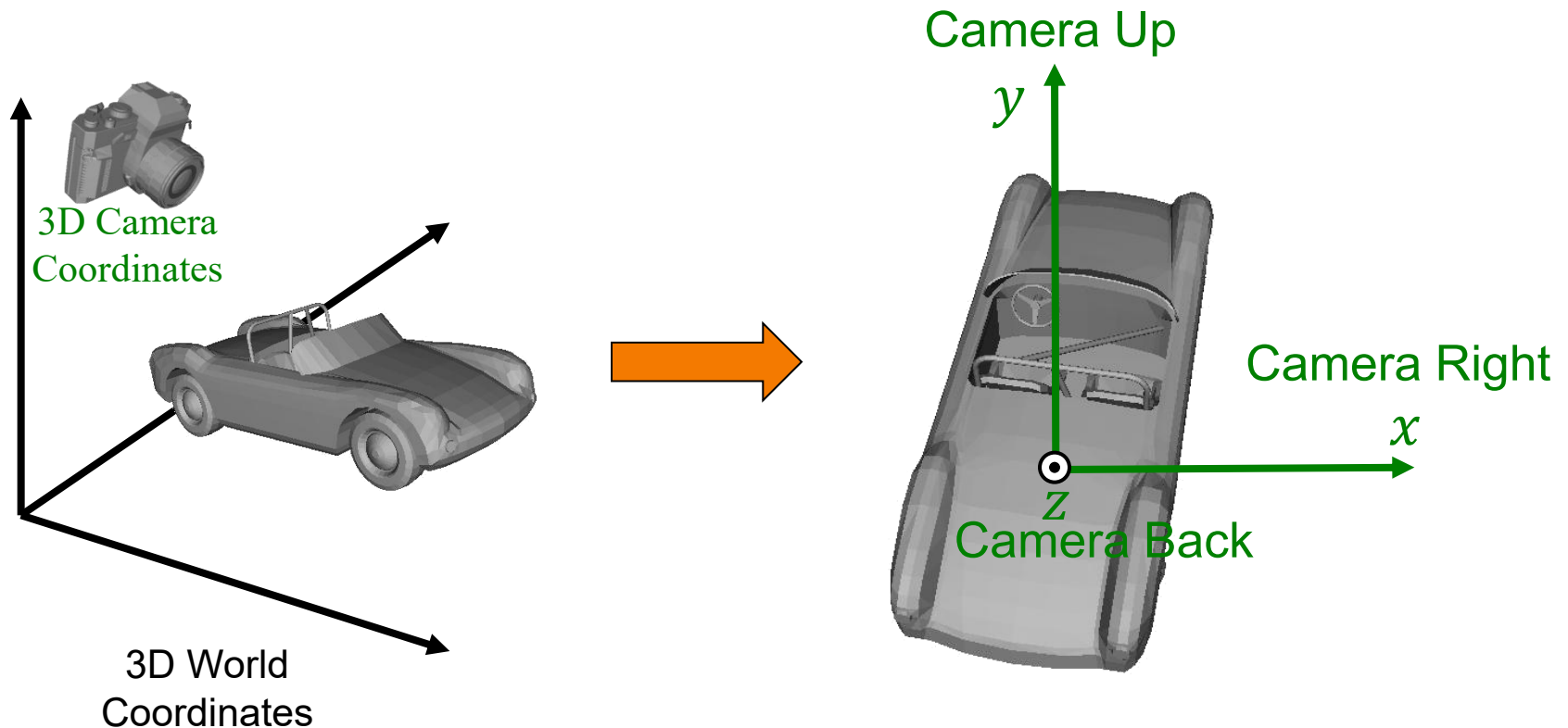




Viewing Transformation

Canonical viewing coordinate system

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





Viewing Transformation

The transformation, $\mathbf{T}_{W \rightarrow C}$, taking us from world coordinates to camera coordinates should map:

- The right vector to the x -axis:

$$(R_x, R_y, R_z, 0) \rightarrow (1, 0, 0, 0)$$

- The up vector to the y -axis:

$$(U_x, U_y, U_z, 0) \rightarrow (0, 1, 0, 0)$$

- The back vector to the z -axis:

$$(B_x, B_y, B_z, 0) \rightarrow (0, 0, 1, 0)$$

- The eye position to the origin:

$$(E_x, E_y, E_z, 1) \rightarrow (0, 0, 0, 1)$$

How should we define this transformation/matrix?



Viewing Transformation

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

$$(R_x, R_y, R_z, 0) \leftarrow (1, 0, 0, 0)$$

$$(U_x, U_y, U_z, 0) \leftarrow (0, 1, 0, 0)$$

$$(B_x, B_y, B_z, 0) \leftarrow (0, 0, 1, 0)$$

$$(E_x, E_y, E_z, 1) \leftarrow (0, 0, 0, 1)$$

This is described by the camera-to-world matrix:

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



Viewing Transformation

The world-to-camera matrix is its inverse:

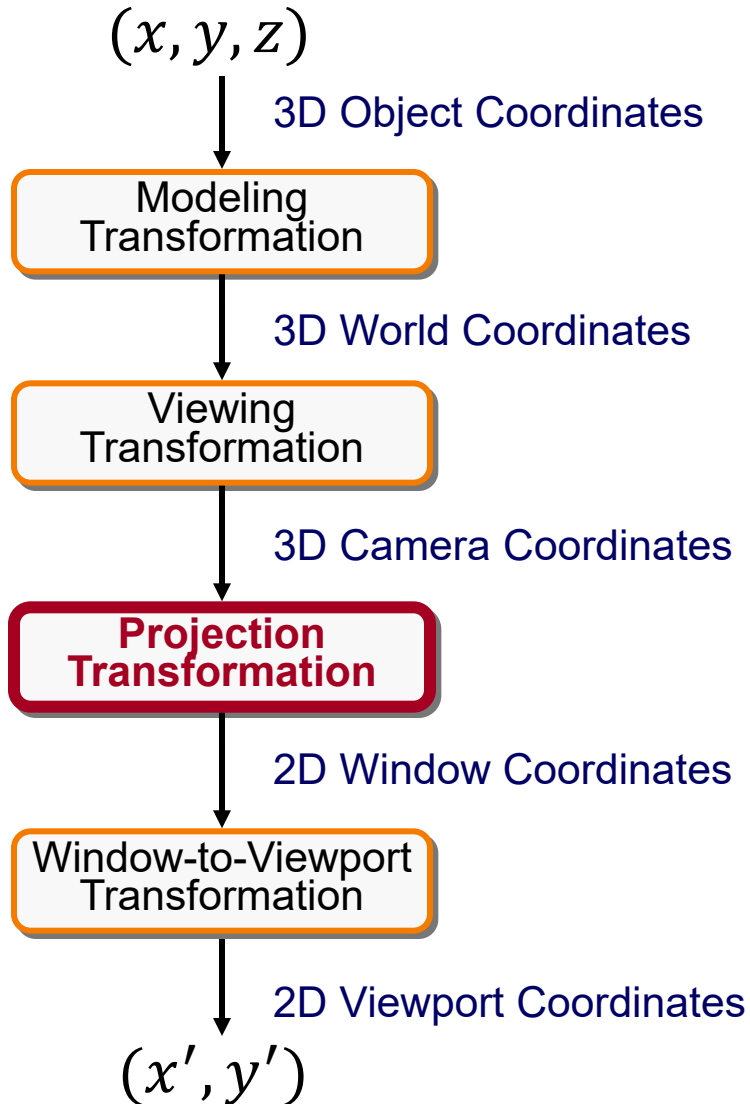
$$\begin{pmatrix} x^c \\ y^c \\ z^c \\ 1 \end{pmatrix} = \underbrace{\begin{pmatrix} R_x & U_x & B_x & E_x \\ R_y & U_y & B_y & E_y \\ R_z & U_z & B_z & E_z \\ 0 & 0 & 0 & 1 \end{pmatrix}^{-1}}_{\mathbf{T}_{W \rightarrow C} = \mathbf{T}_{C \rightarrow W}^{-1}} \begin{pmatrix} x^w \\ y^w \\ z^w \\ 1 \end{pmatrix}$$

This is described by the camera-to-world matrix:

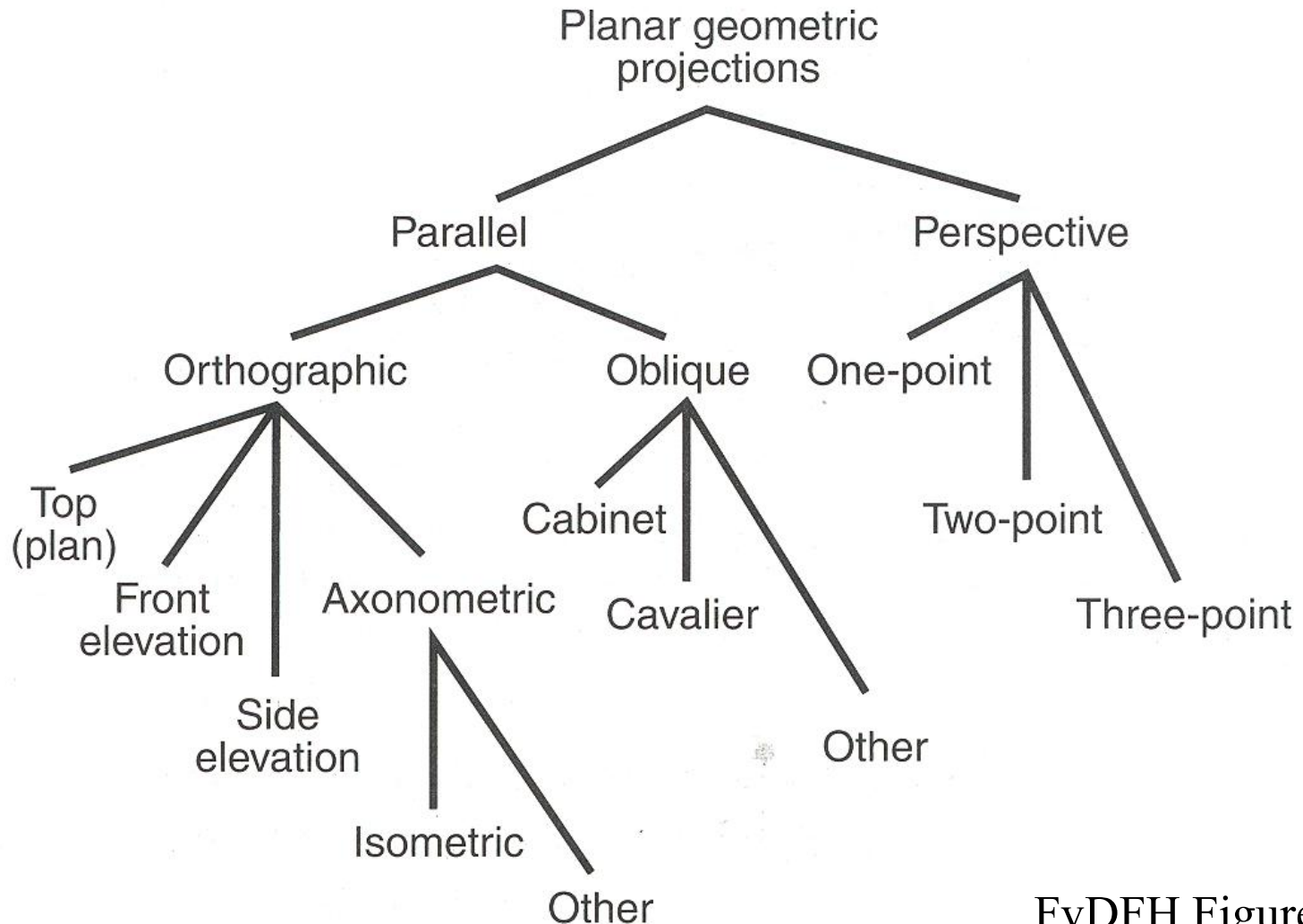
$$\begin{pmatrix} x^w \\ y^w \\ z^w \\ 1 \end{pmatrix} = \underbrace{\begin{pmatrix} R_x & U_x & B_x & E_x \\ R_y & U_y & B_y & E_y \\ R_z & U_z & B_z & E_z \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{\mathbf{T}_{C \rightarrow W}} \begin{pmatrix} x^c \\ y^c \\ z^c \\ 1 \end{pmatrix}$$



Transformations



Taxonomy of Projections



FvDFH Figure 6.13



Projection

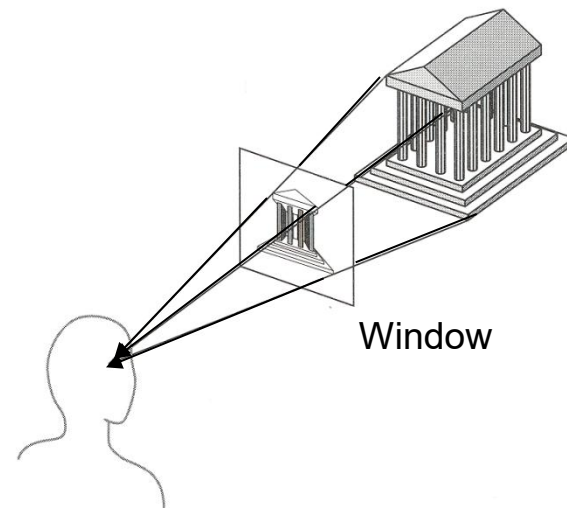
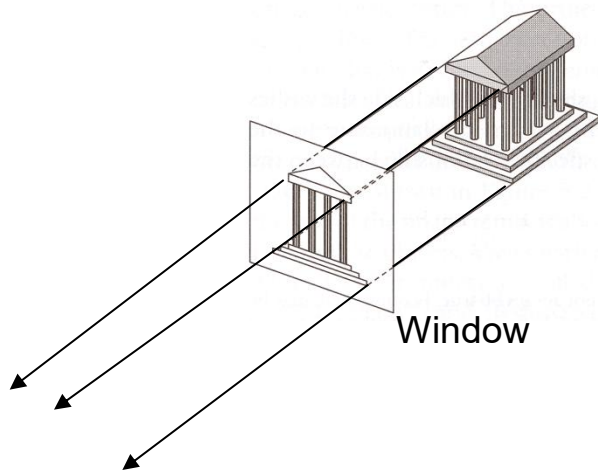
Two general classes. Both correspond to shooting rays *from* the 3D scene, *through* the 2D window:

Parallel Projection:

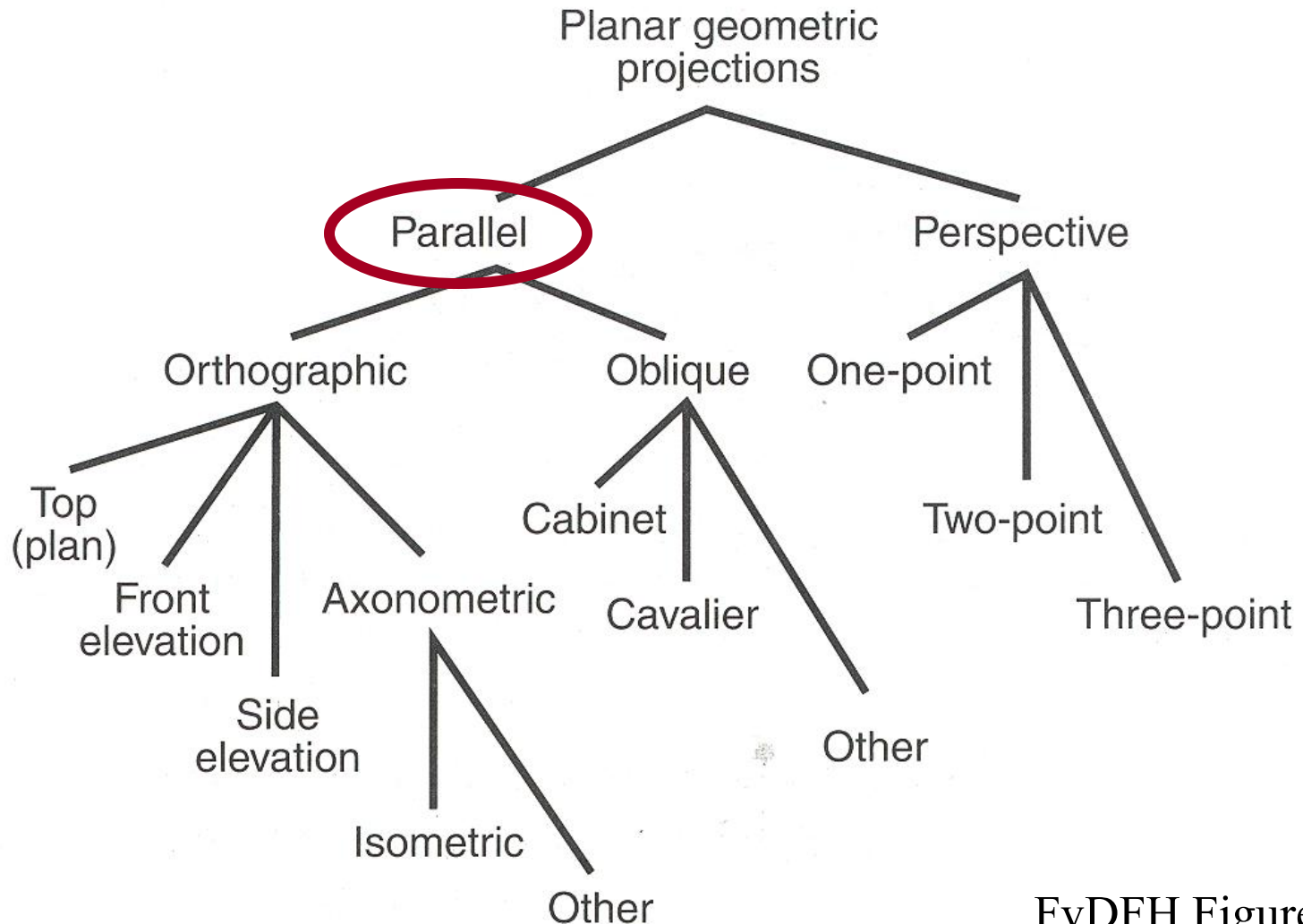
Rays converge at a point at infinity and are parallel

Perspective “Projection”:

Rays converge at a finite point, giving rise to perspective distortion



Taxonomy of Projections



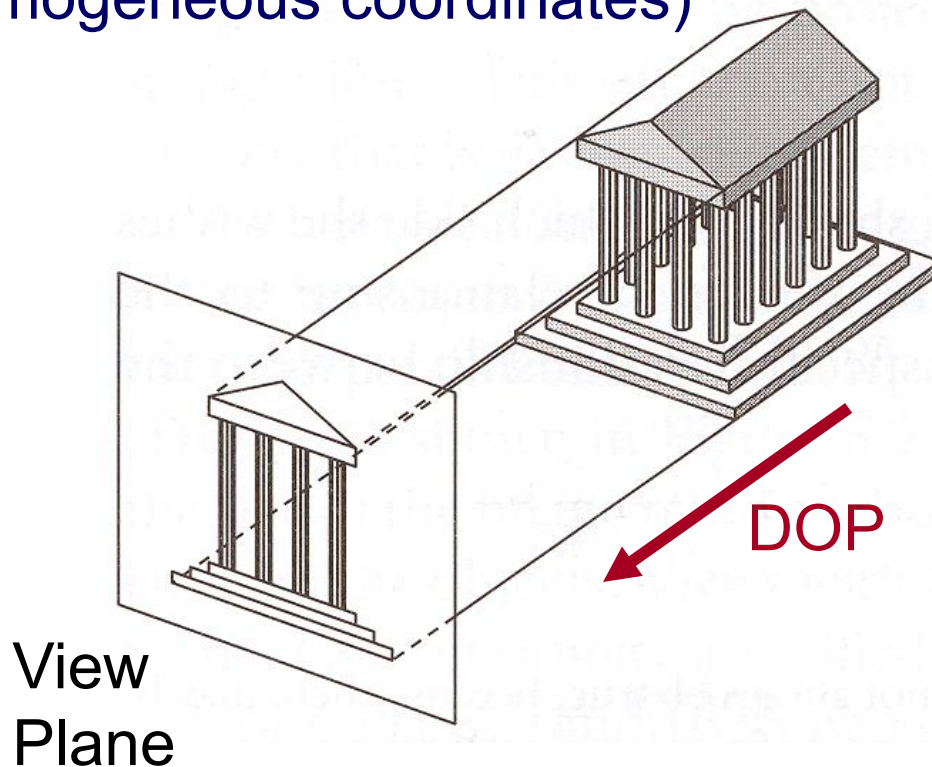
FvDFH Figure 6.13



Parallel Projection

Center of projection is at infinity

Direction of projection (DoP) same for all points
(up to sign, can be represented as a center with homogeneous coordinates)

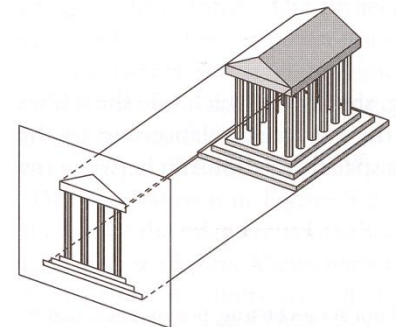


Angel Figure 5.4

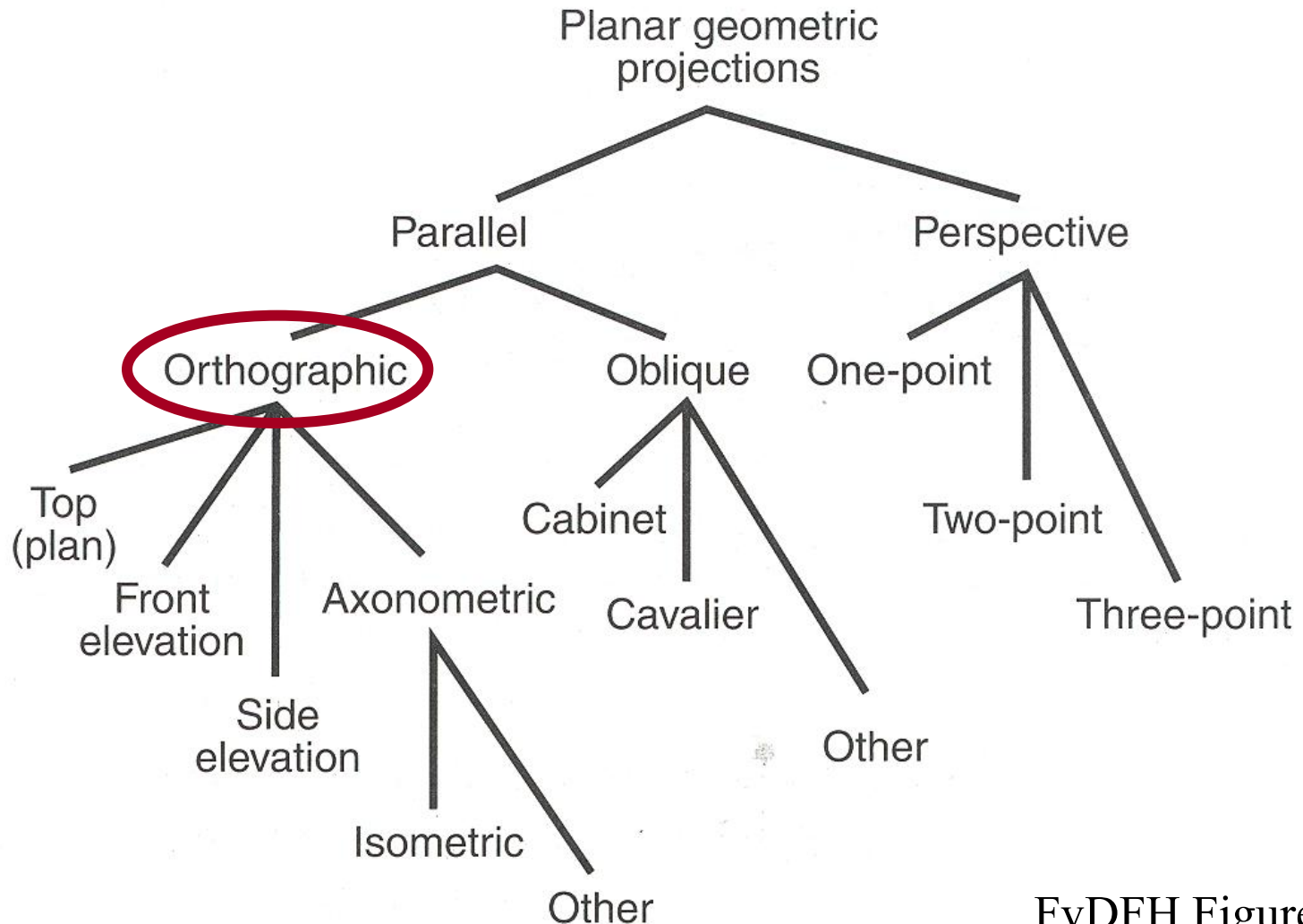


Parallel Projection

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



Taxonomy of Projections

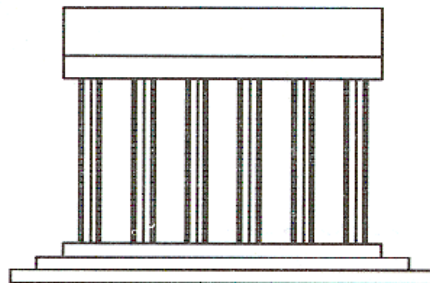


FvDFH Figure 6.13

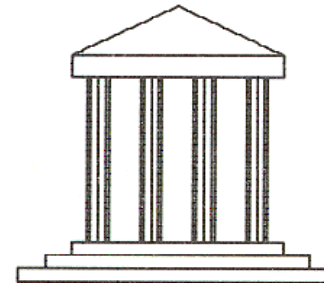


Orthographic Projections

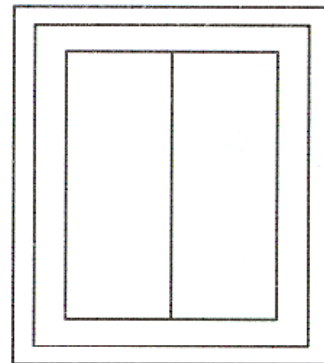
DoP perpendicular to view plane



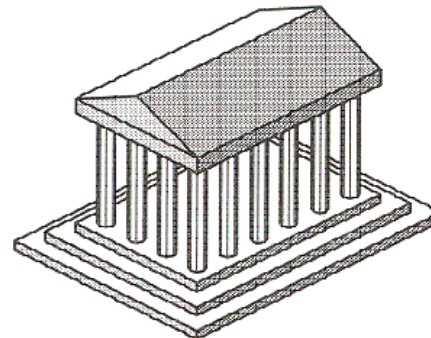
Side



Front



Top

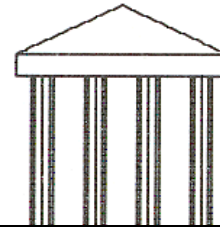
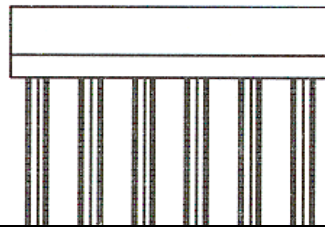


Isometric

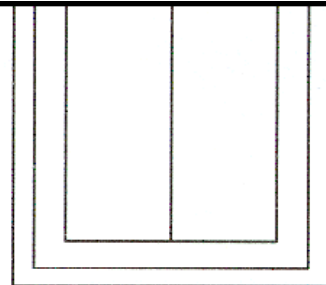


Orthographic Projections

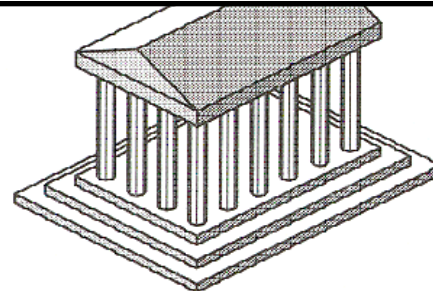
DoP perpendicular to view plane



- Lines perpendicular to the view plane vanish
- Faces parallel to the view plane are un-distorted.



Top



Isometric

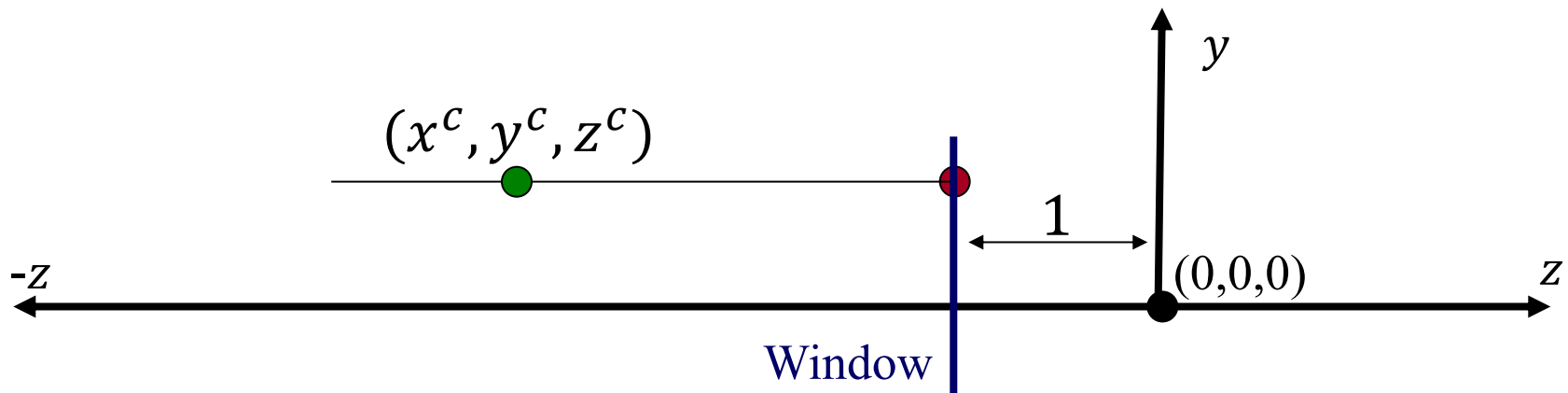


Orthographic Projections

DoP perpendicular to view plane

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

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





Orthographic Projections

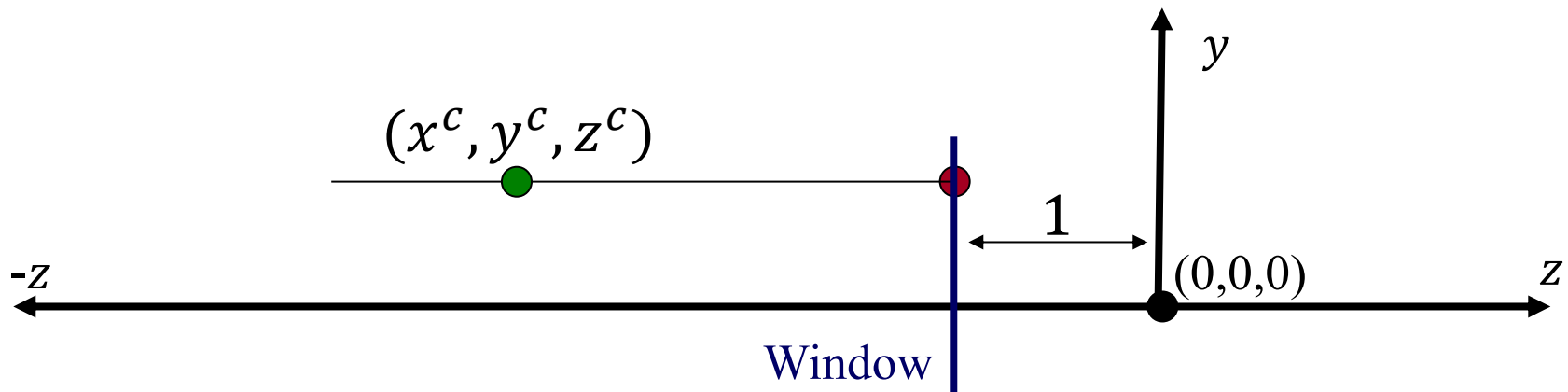
DoP perpendicular to view plane

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

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

- 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:

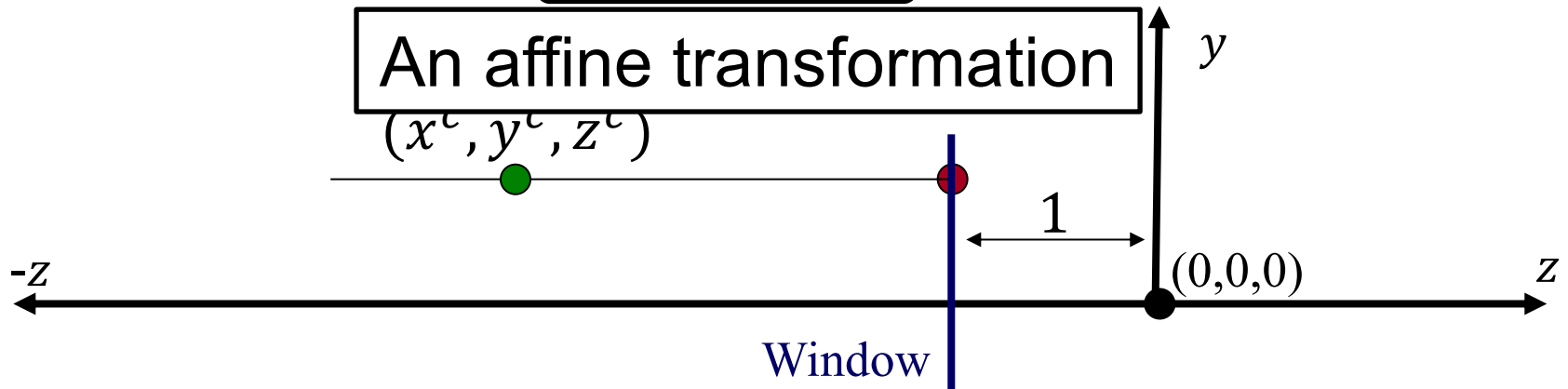
$$(x^c, y^c, z^c, 1) \rightarrow (x^c, y^c, -1, 1)$$

- In terms of homogenous coordinates:

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

An affine transformation

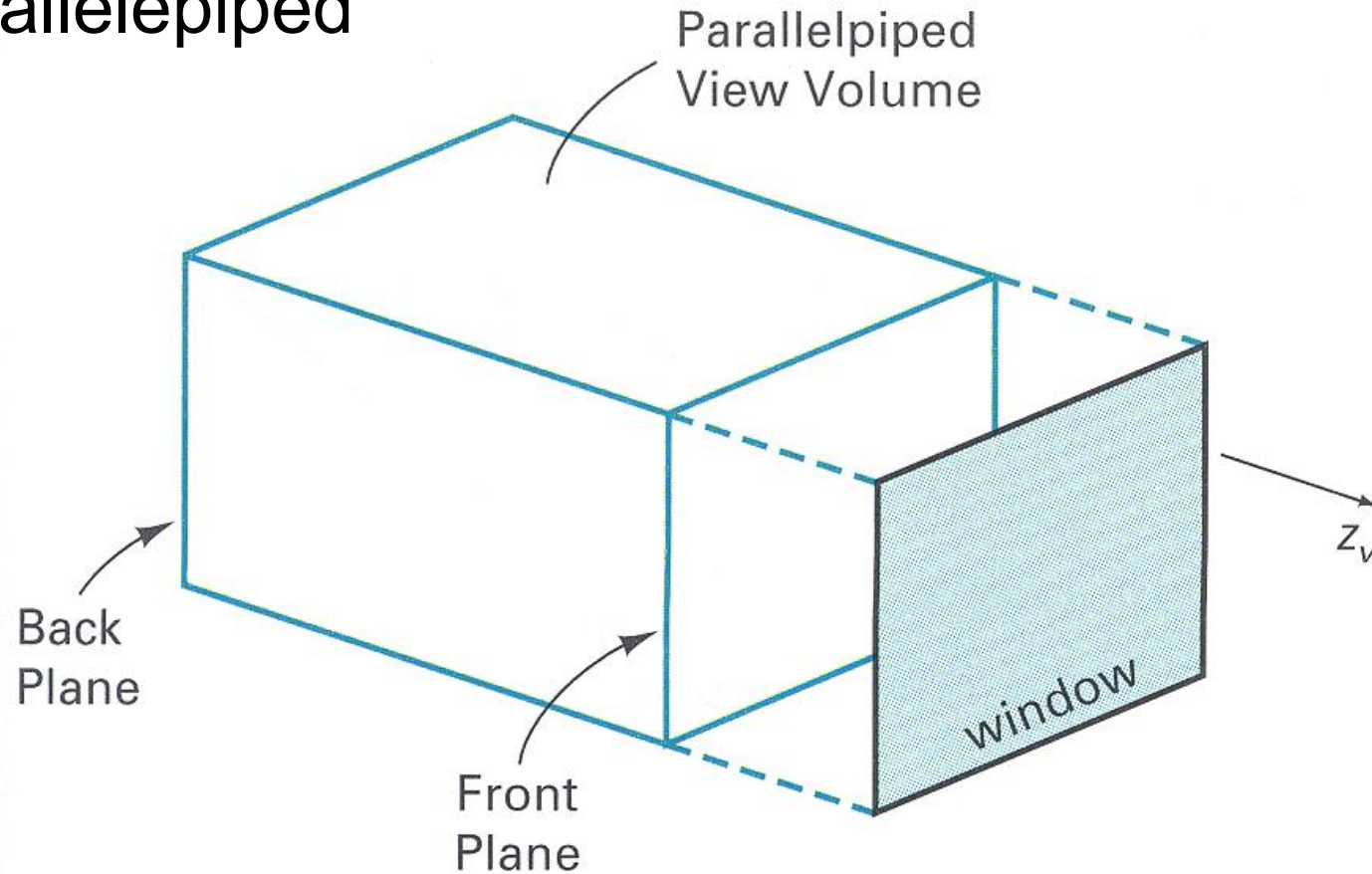
(x^c, y^c, z^c)



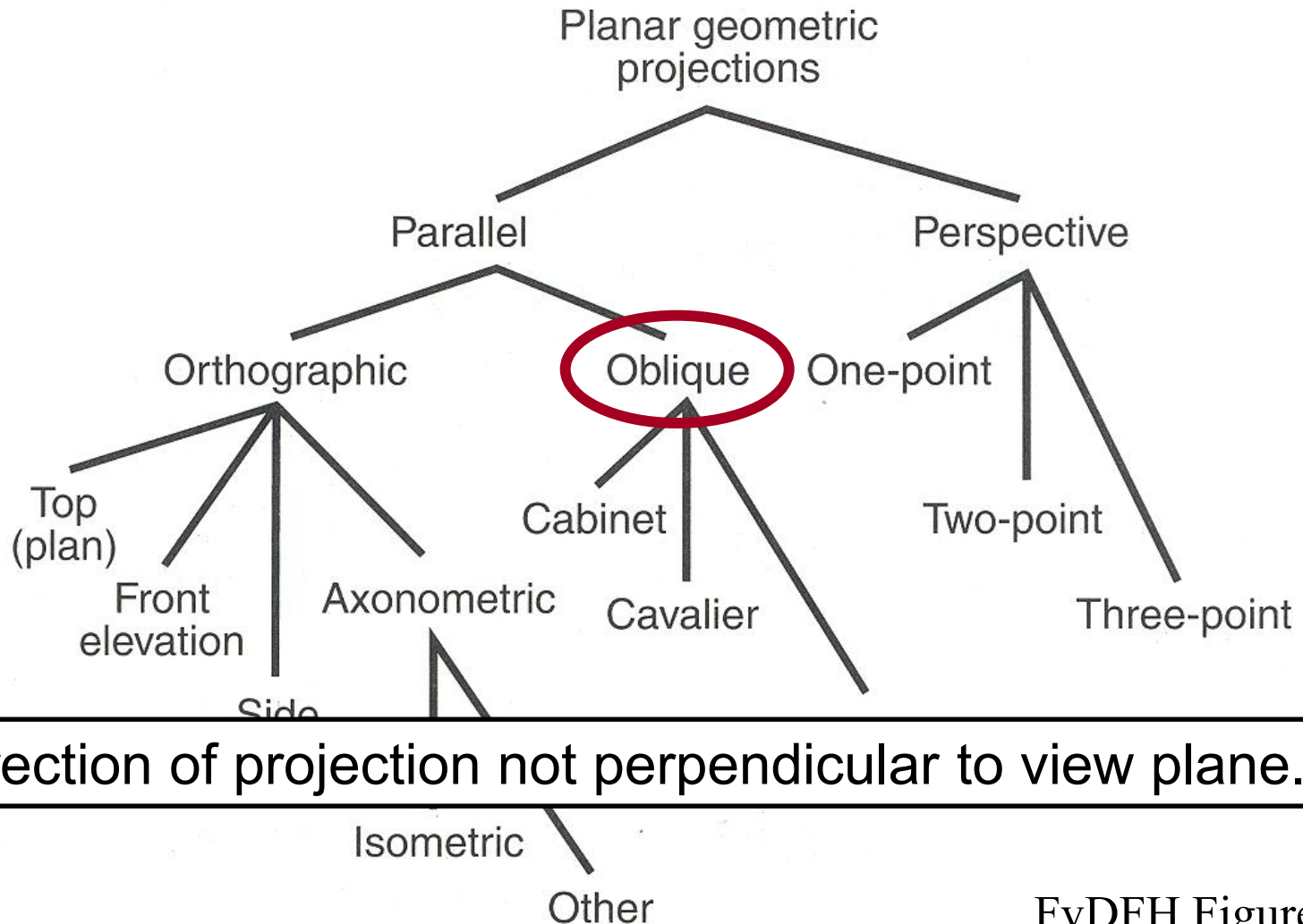


Parallel Projection View Volume

The volume projecting onto the window is a parallelepiped

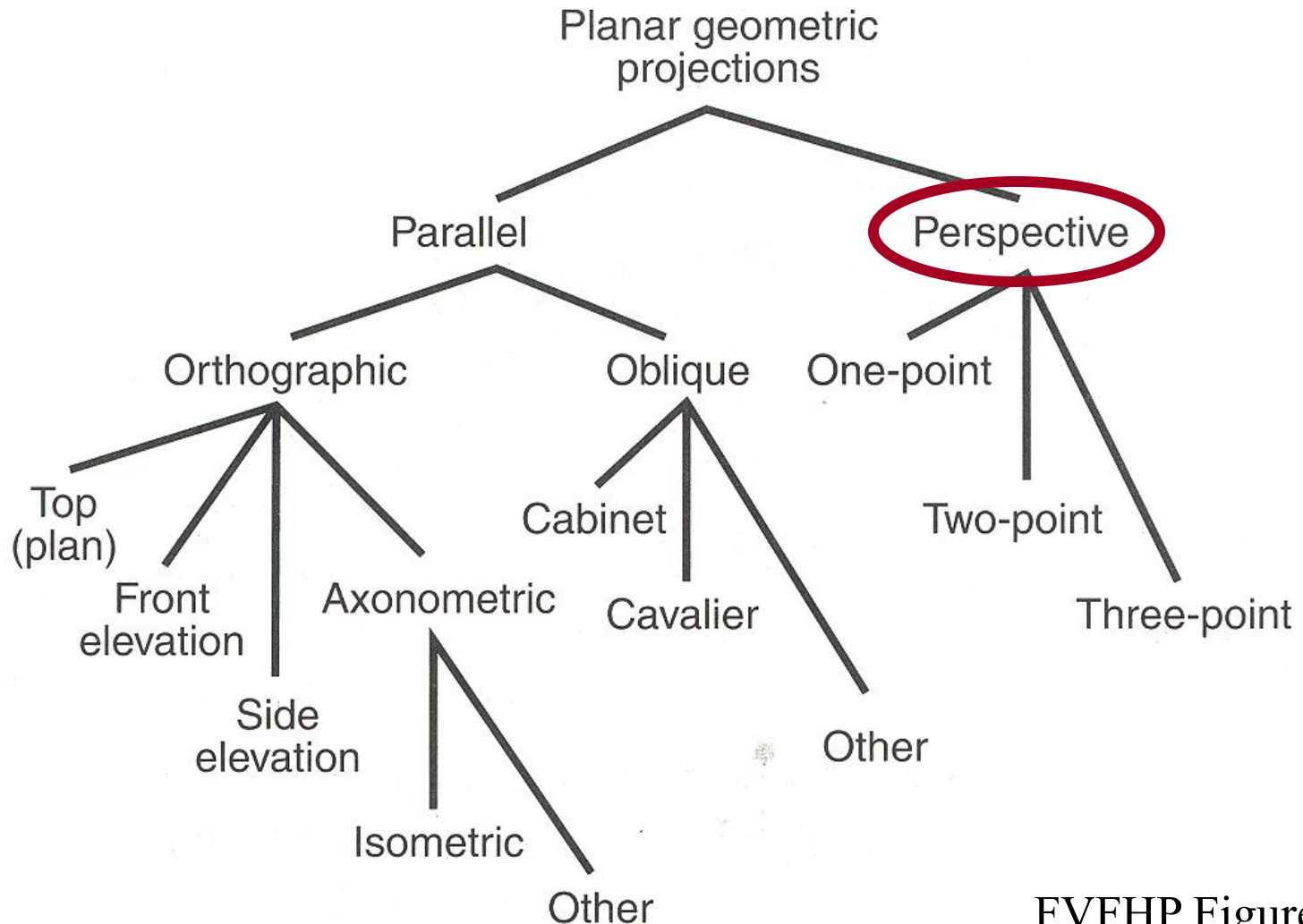


Taxonomy of Projections





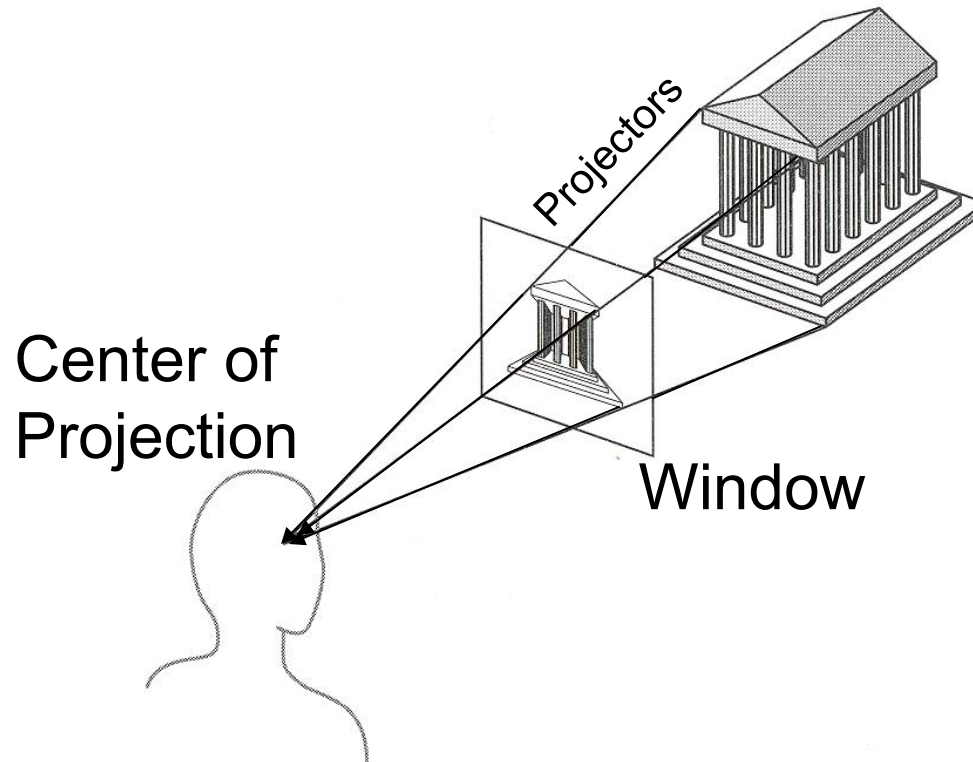
Taxonomy of Projections





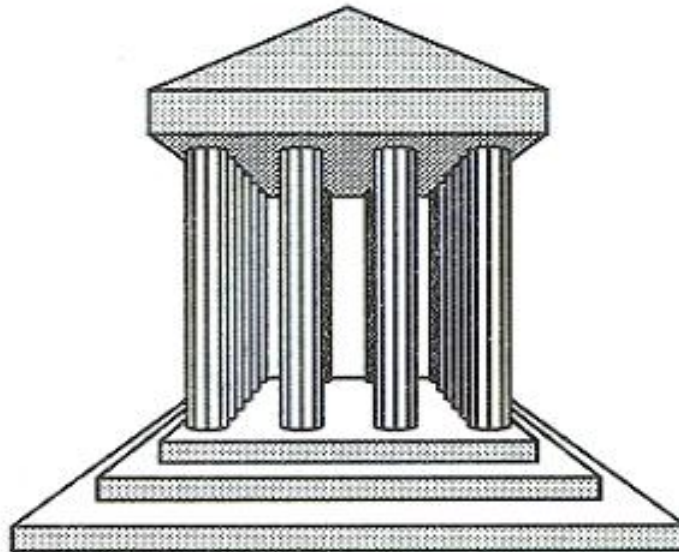
Perspective “Projection”

Map points onto view plane along “projectors” emanating from the **center of projection** (CoP)



Perspective Projection

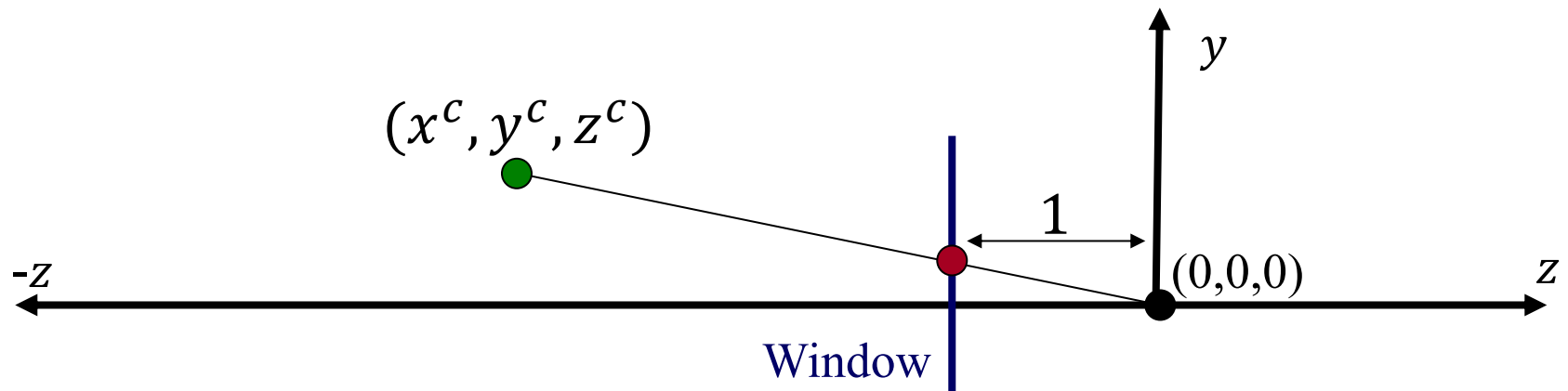
Not all parallel lines remain parallel!





Perspective Projection

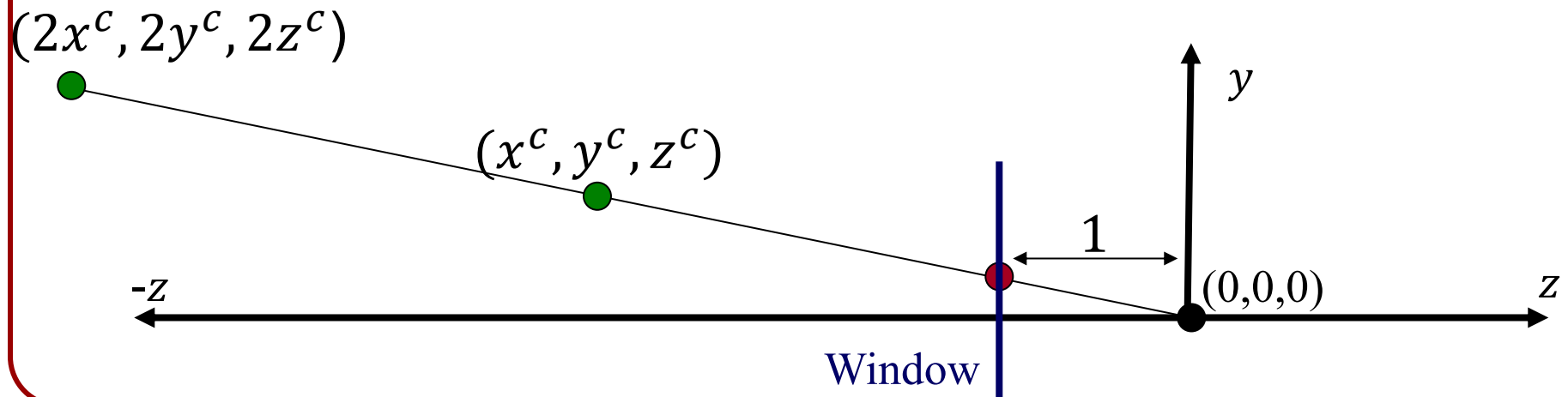
Q: What are the coordinates of the point resulting from projection of (x^c, y^c, z^c) onto the camera screen a unit distance back along the z -axis?





Perspective Projection

A: 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.



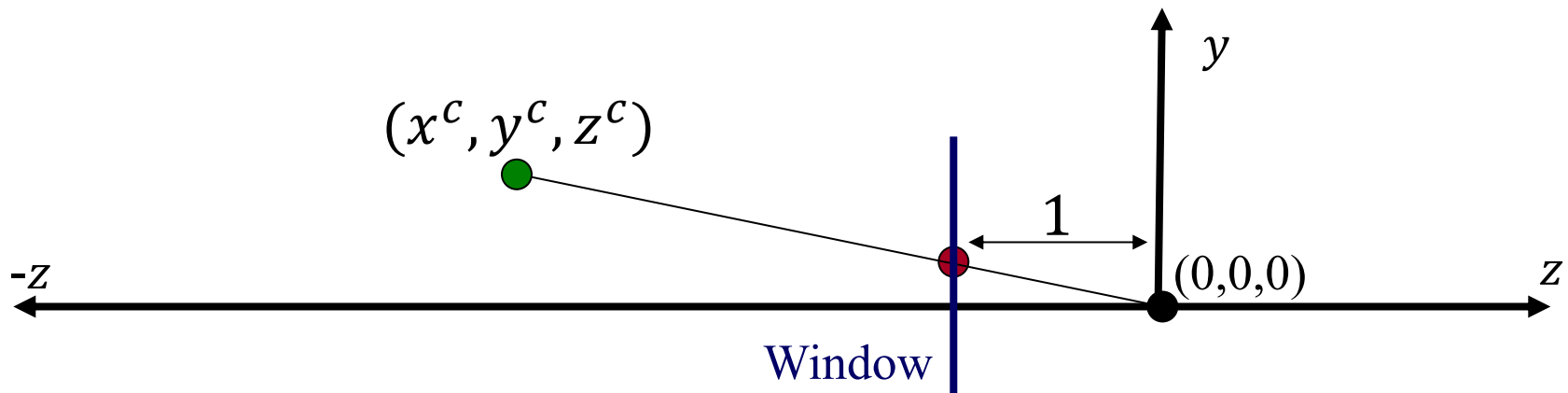


Perspective Projection

A: 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:

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



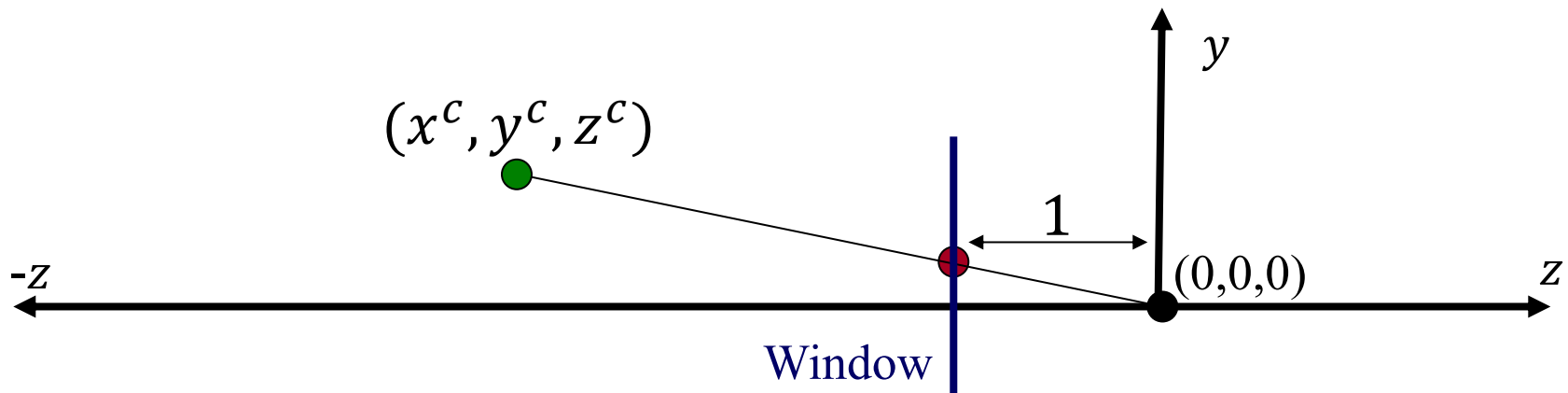


Perspective Projection

A: 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

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

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

In homogeneous coordinates, we can write this as:

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

In matrix form, this gives:

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



Perspective Projection Matrix

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

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

In homogeneous coordinates, we can write this as:

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

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

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

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

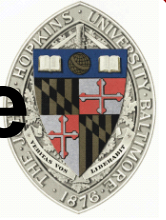
In homogeneous coordinates, we can write this as:

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

In matrix form, this gives:

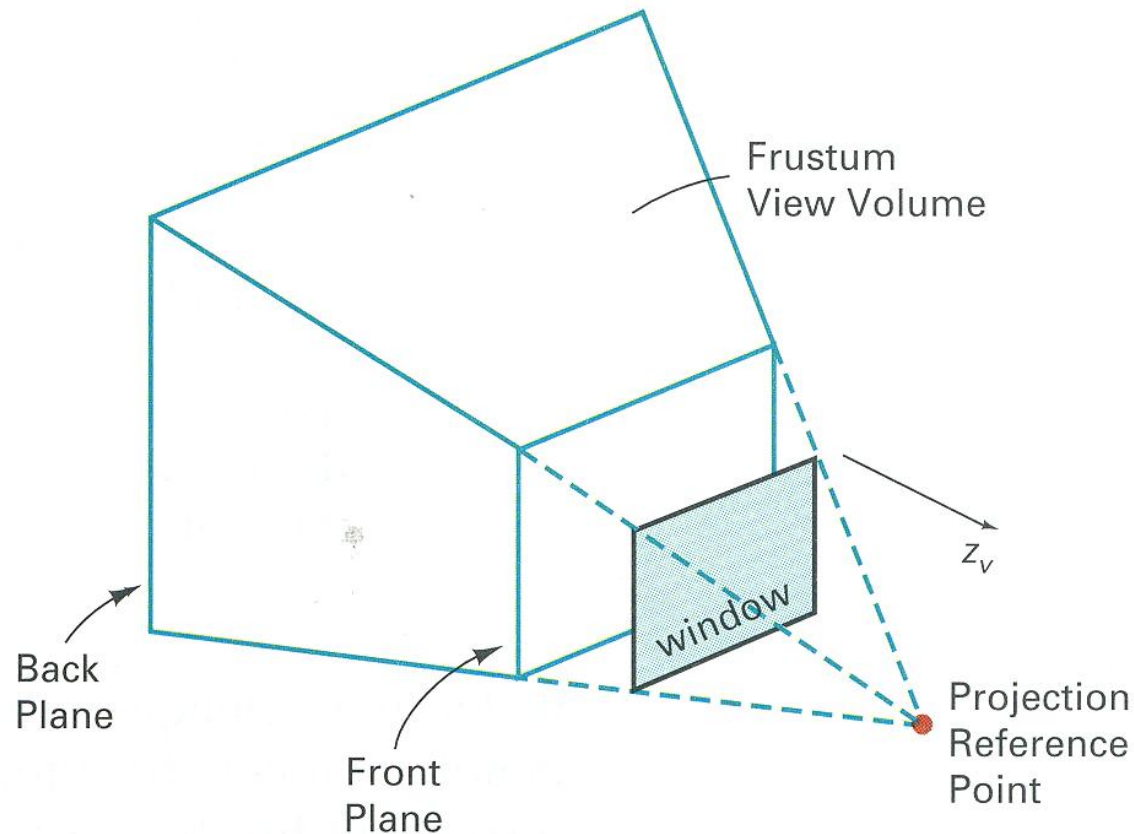
$$\begin{bmatrix} -x^c/z^c \\ -y^c/z^c \\ -1 \\ 1 \end{bmatrix} \equiv \begin{bmatrix} x^c \\ y^c \\ z^c \\ -z^c \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x^c \\ y^c \\ z^c \\ 1 \end{bmatrix}$$

Not an affine transformation.

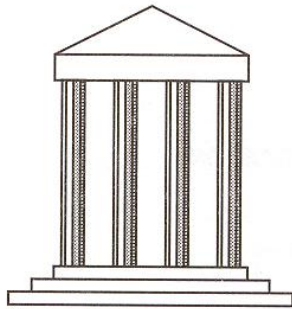


Perspective Projection View Volume

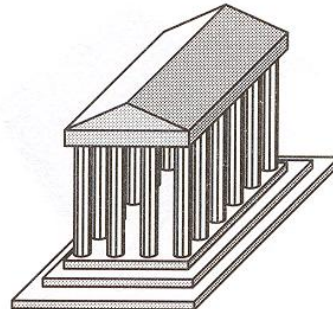
The volume projecting onto the window is a pyramid



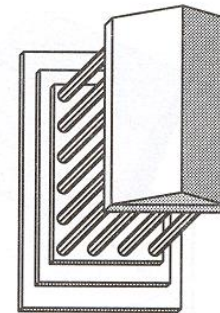
Classical Projections



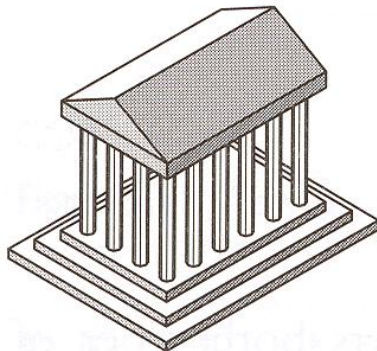
Front elevation



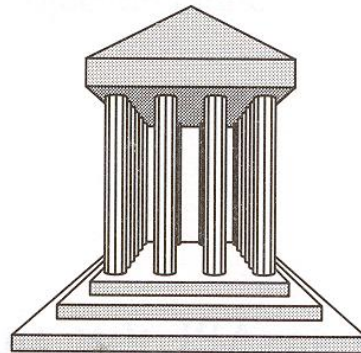
Elevation oblique



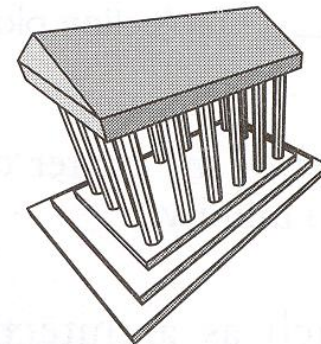
Plan oblique



Isometric



One-point perspective



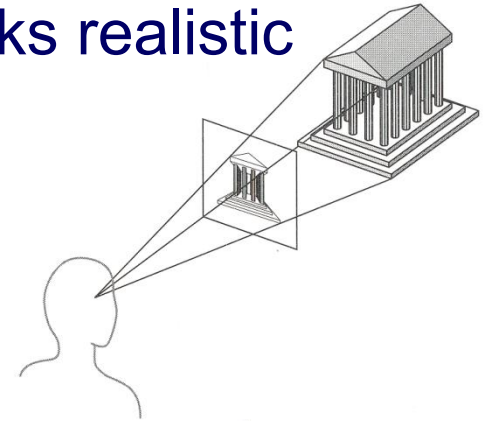
Three-point perspective



Perspective vs. Parallel

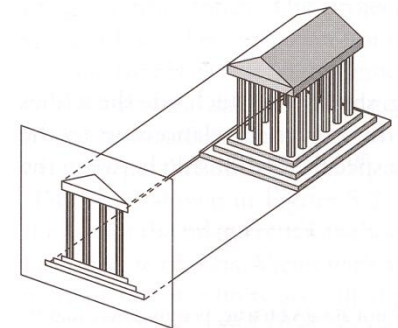
Perspective projection

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

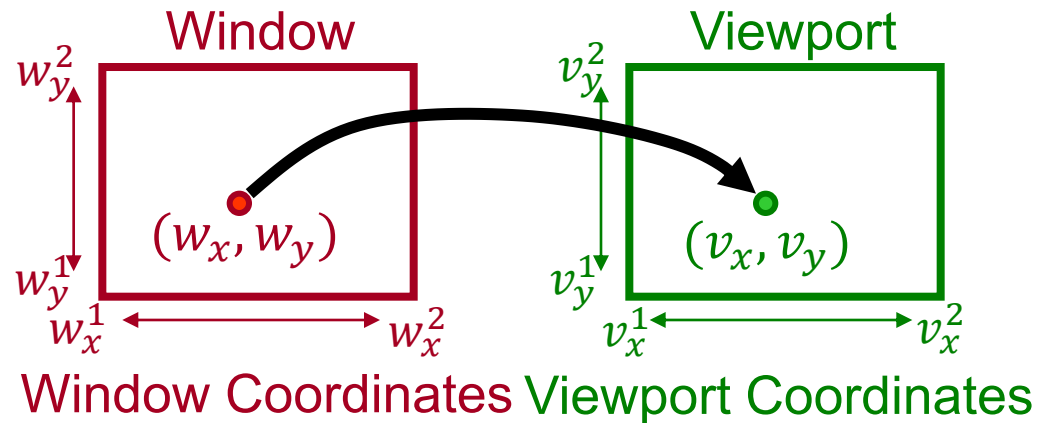
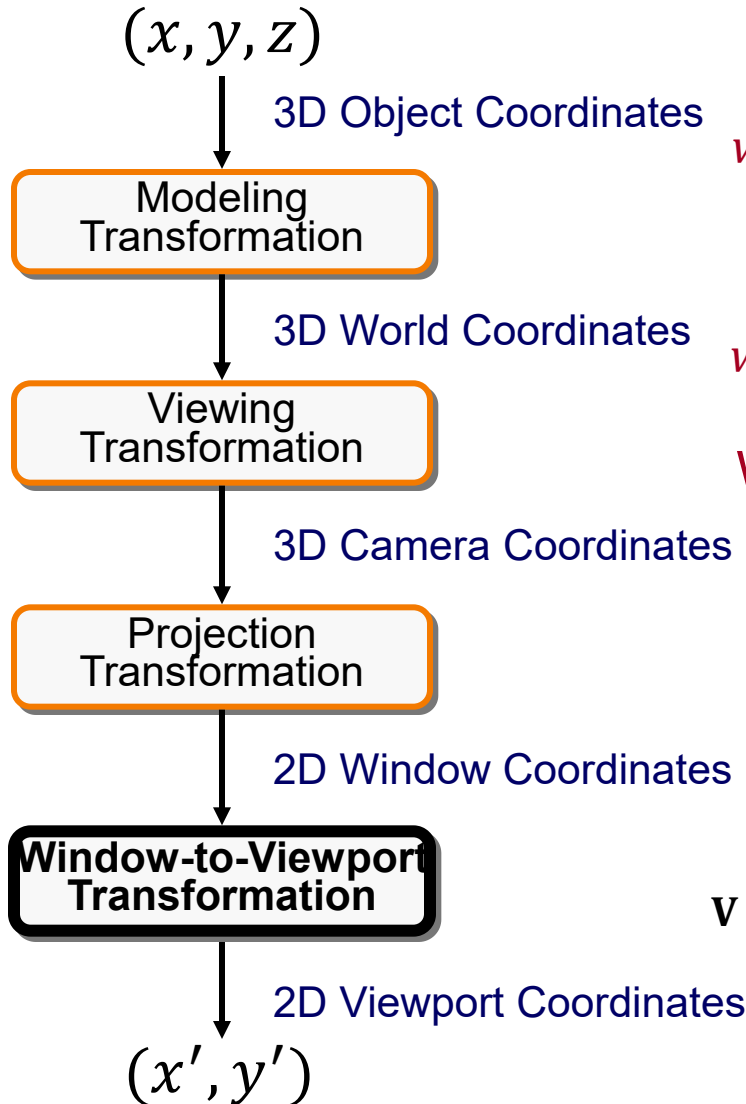


Parallel (orthographic) projection

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



Transformations

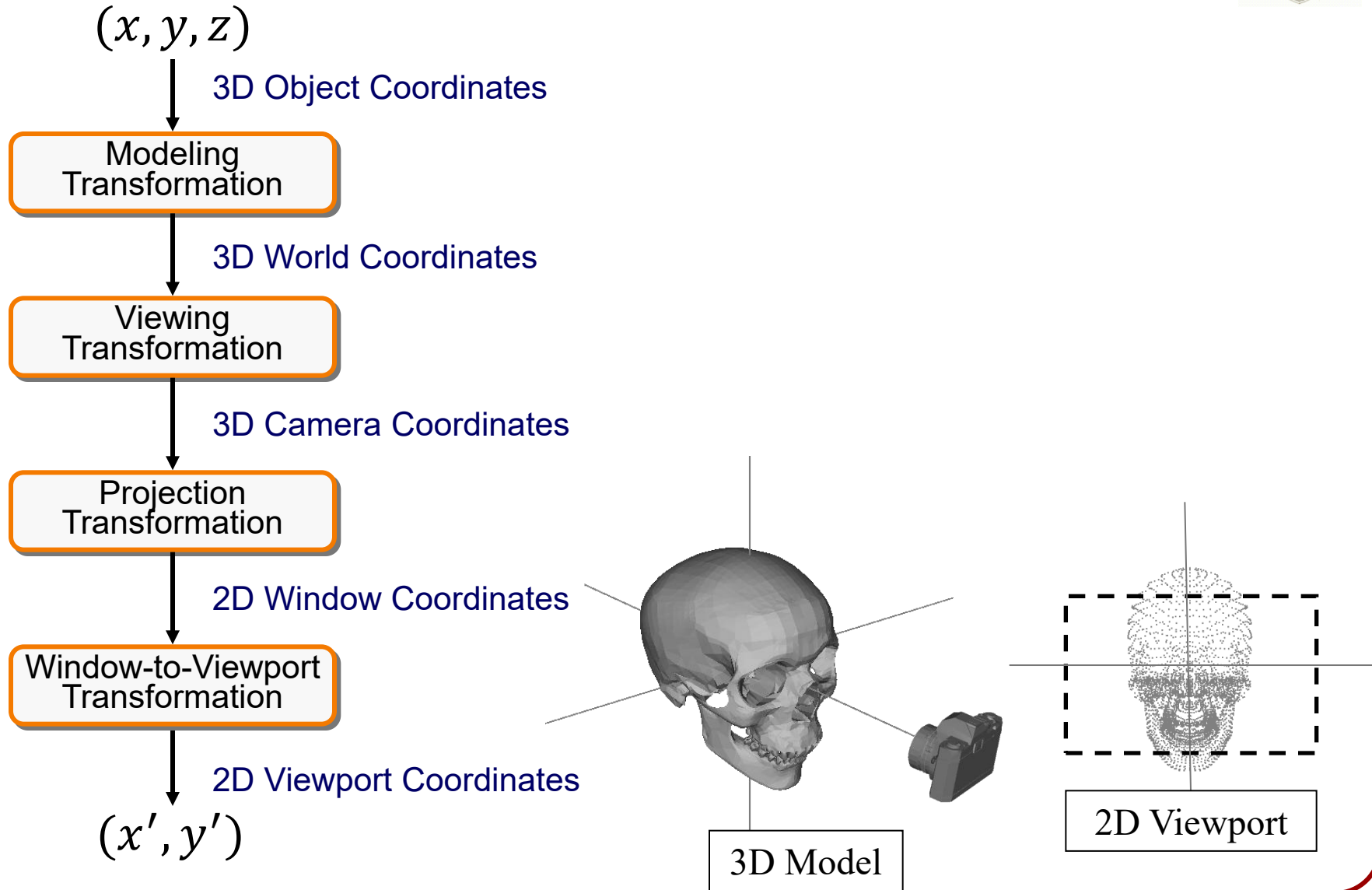


V = viewport transform

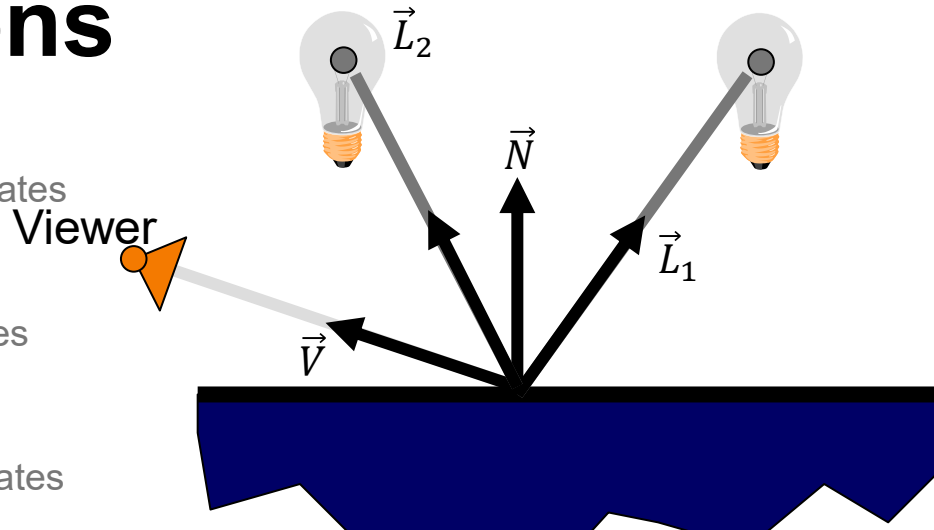
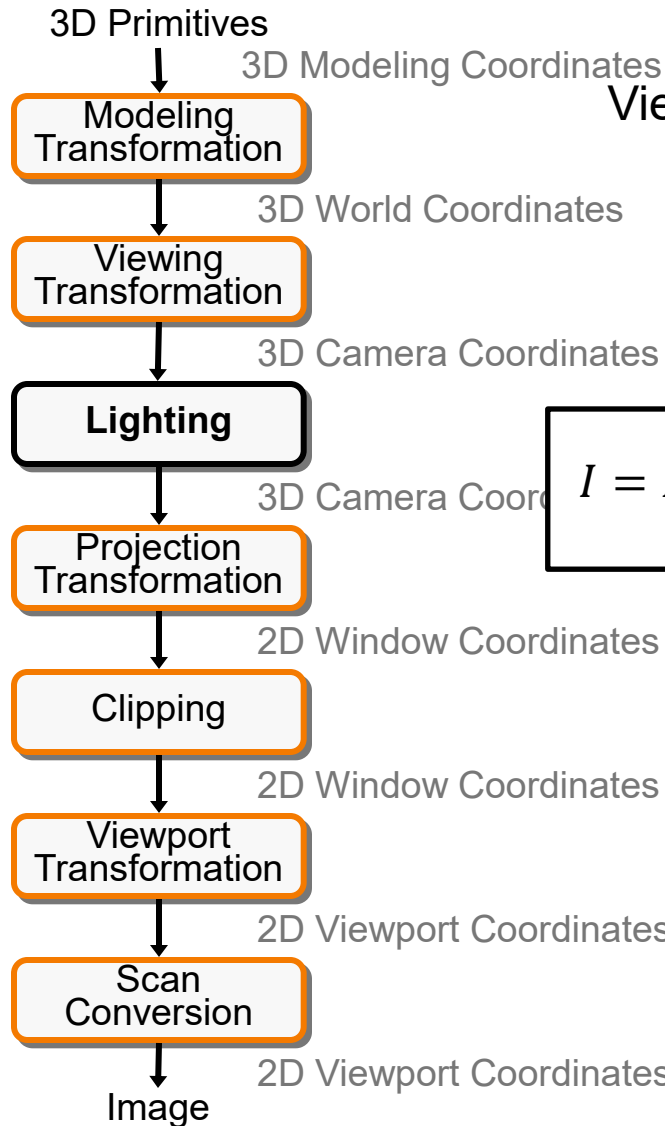
$$V = \begin{bmatrix} 1 & 0 & v_x^1 \\ 0 & 1 & v_y^1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{v_x^2 - v_x^1}{w_x^2 - w_x^1} & 0 & 0 \\ 0 & \frac{v_y^2 - v_y^1}{w_y^2 - w_y^1} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -w_x^1 \\ 0 & 1 & -w_y^1 \\ 0 & 0 & 1 \end{bmatrix}$$

Note that this may scale non-uniformly.

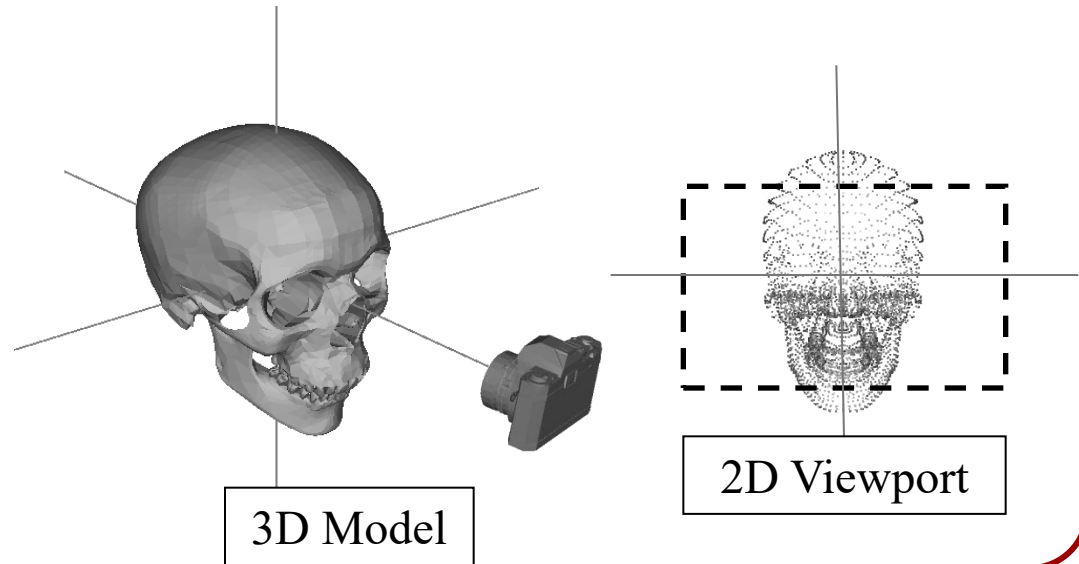
3D Rendering Pipeline (for direct illumination)



Transformations

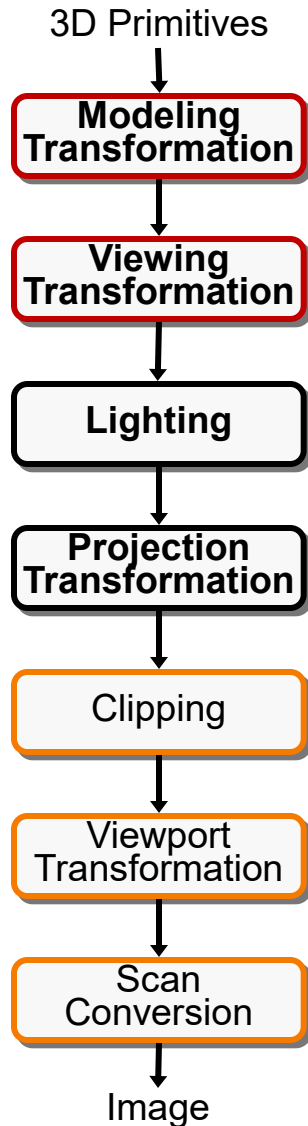


$$I = I_E + \sum_L [K_A \cdot I_L^A + (K_D \cdot \langle \vec{N}, \vec{L} \rangle + K_S \cdot \langle \vec{V}, \vec{R}(\vec{L}) \rangle^{K_n}) \cdot I_L]$$



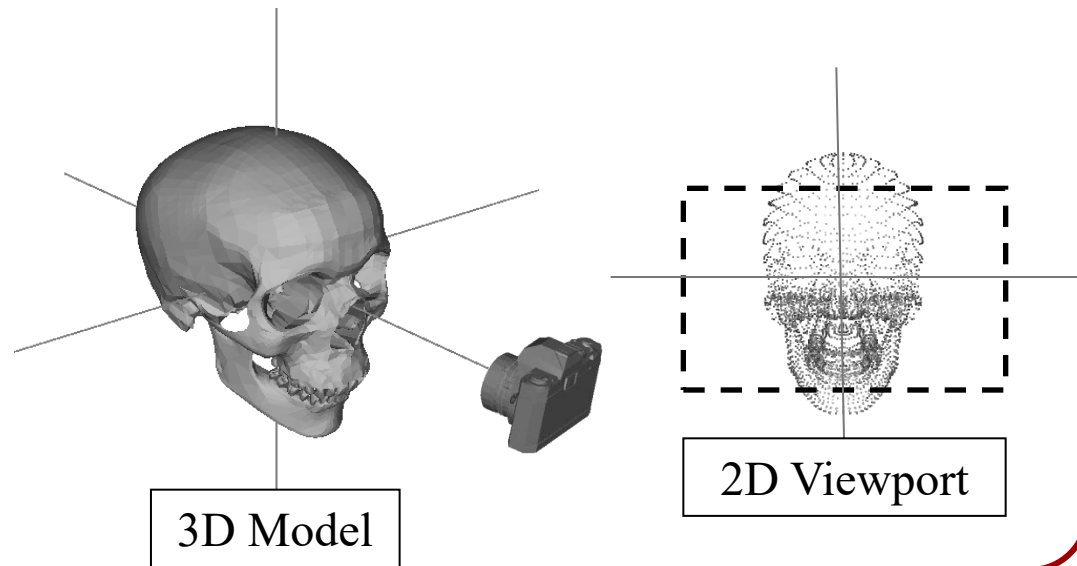


Transformations

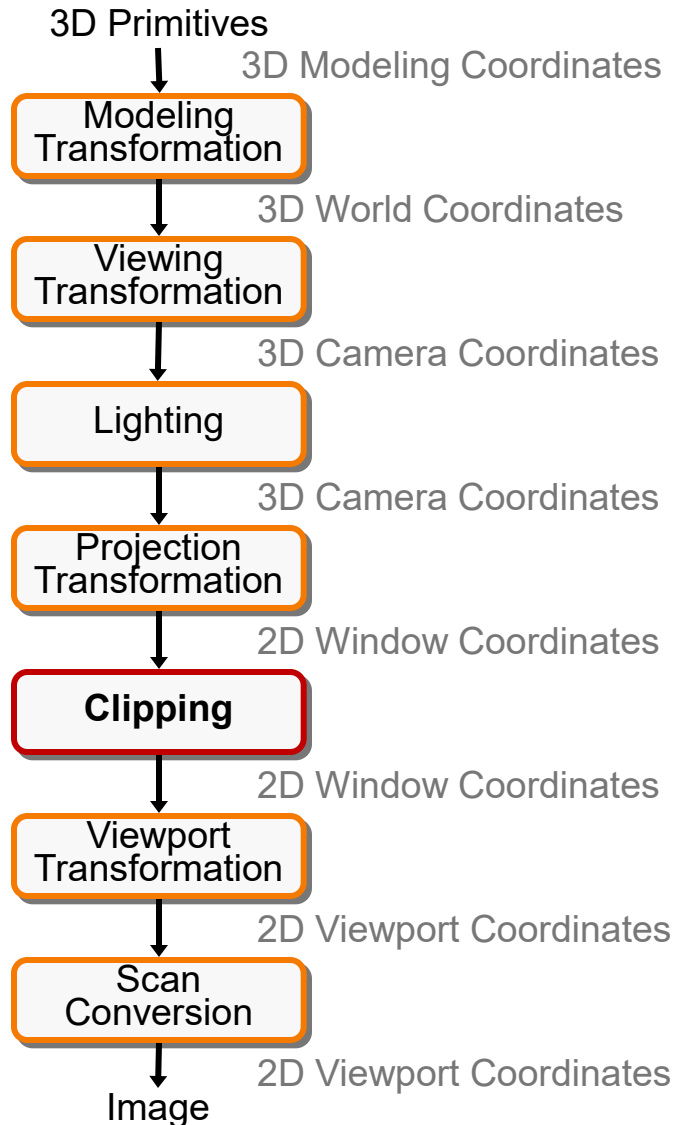


Vertex processing

- 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

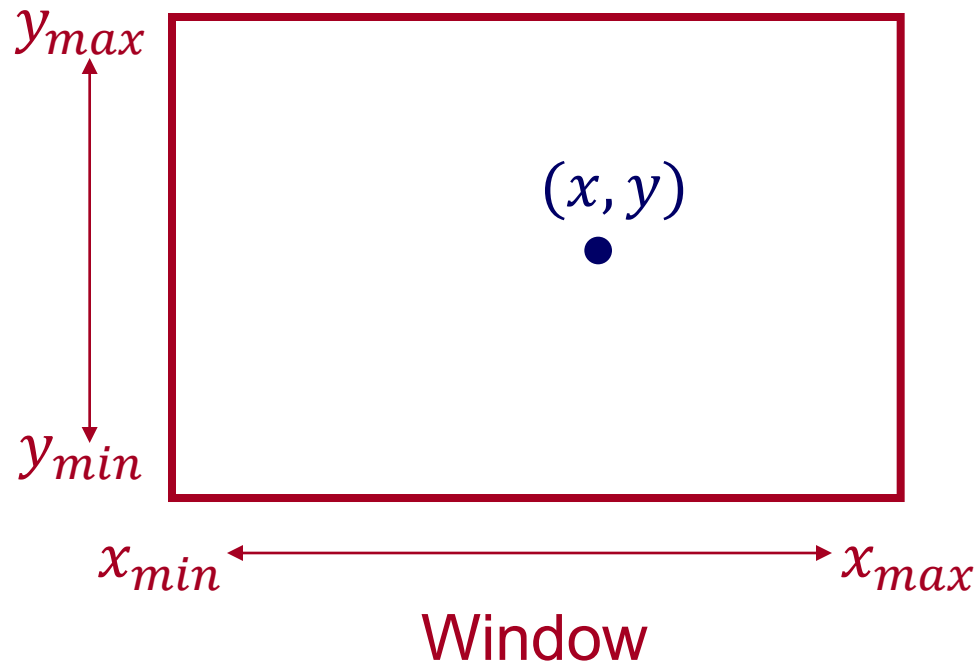
- **Points**
- Line Segments
- Polygons





Point Clipping

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



```
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

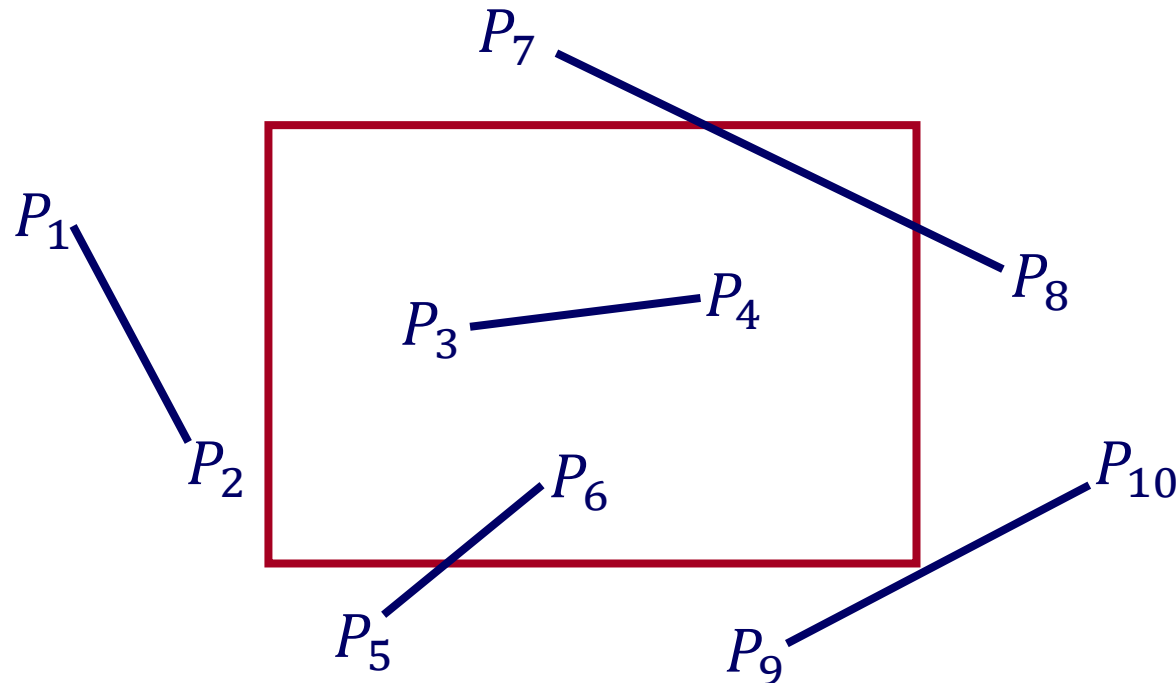




Line Segment Clipping

Find the part of a line inside the clip window

Do this efficiently by identifying the easiest cases first

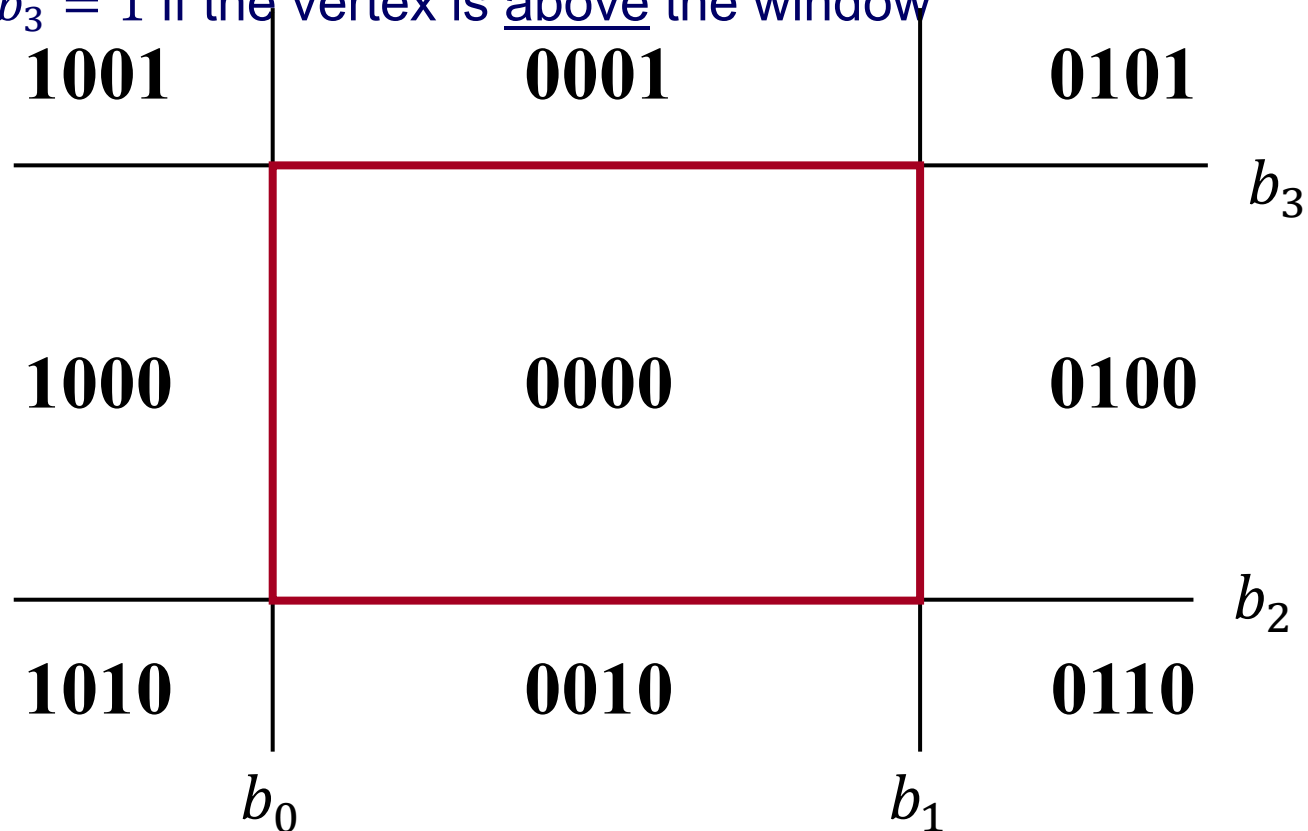




Cohen-Sutherland Line Clipping

Associate a 4-bit outcode $b_0b_1b_2b_3$ to each vertex

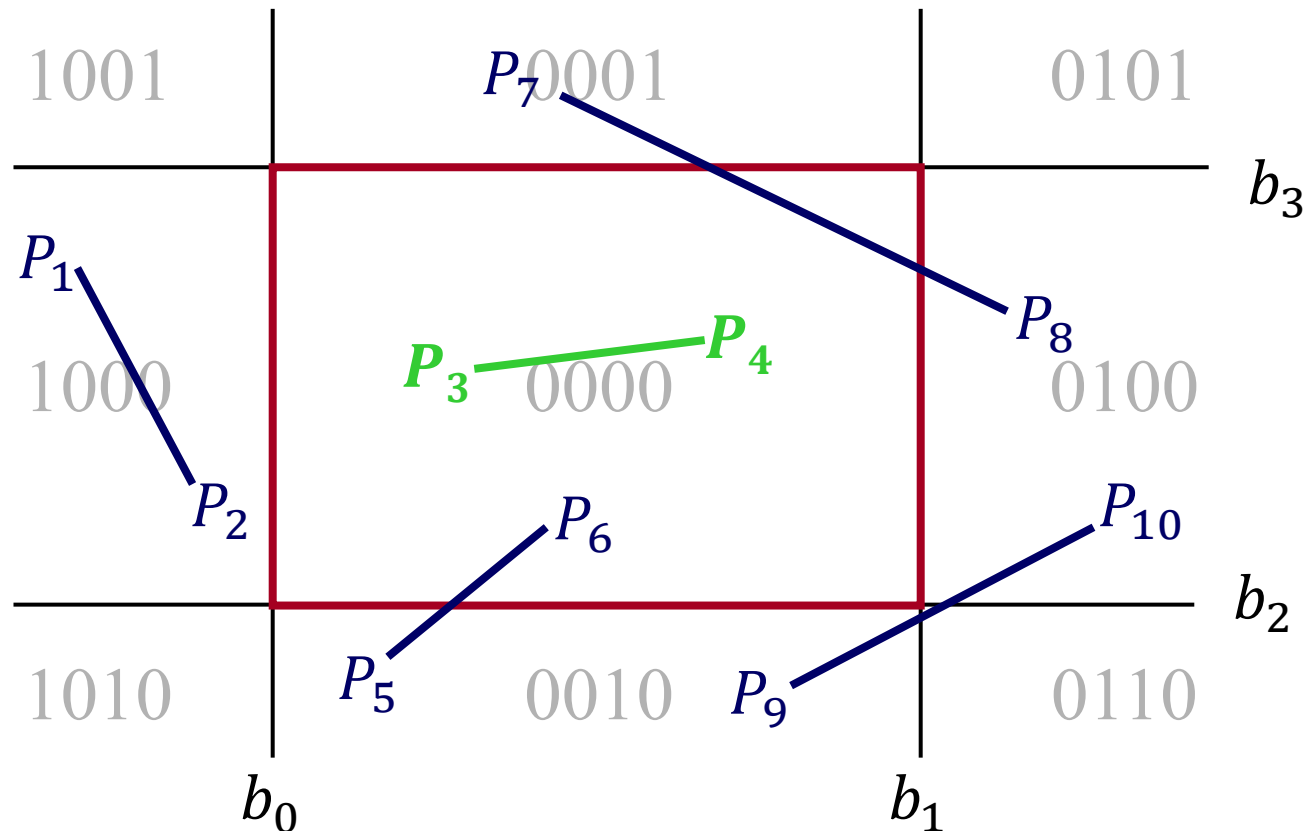
- $b_0 = 1$ if the vertex is left of the window
- $b_1 = 1$ if the vertex is right of the window
- $b_2 = 1$ if the vertex is below the window
- $b_3 = 1$ if the vertex is above the window





Cohen-Sutherland Line Clipping

Associate a 4-bit outcode $b_0b_1b_2b_3$ to each vertex
If both points' outcodes are 0, line segment is inside



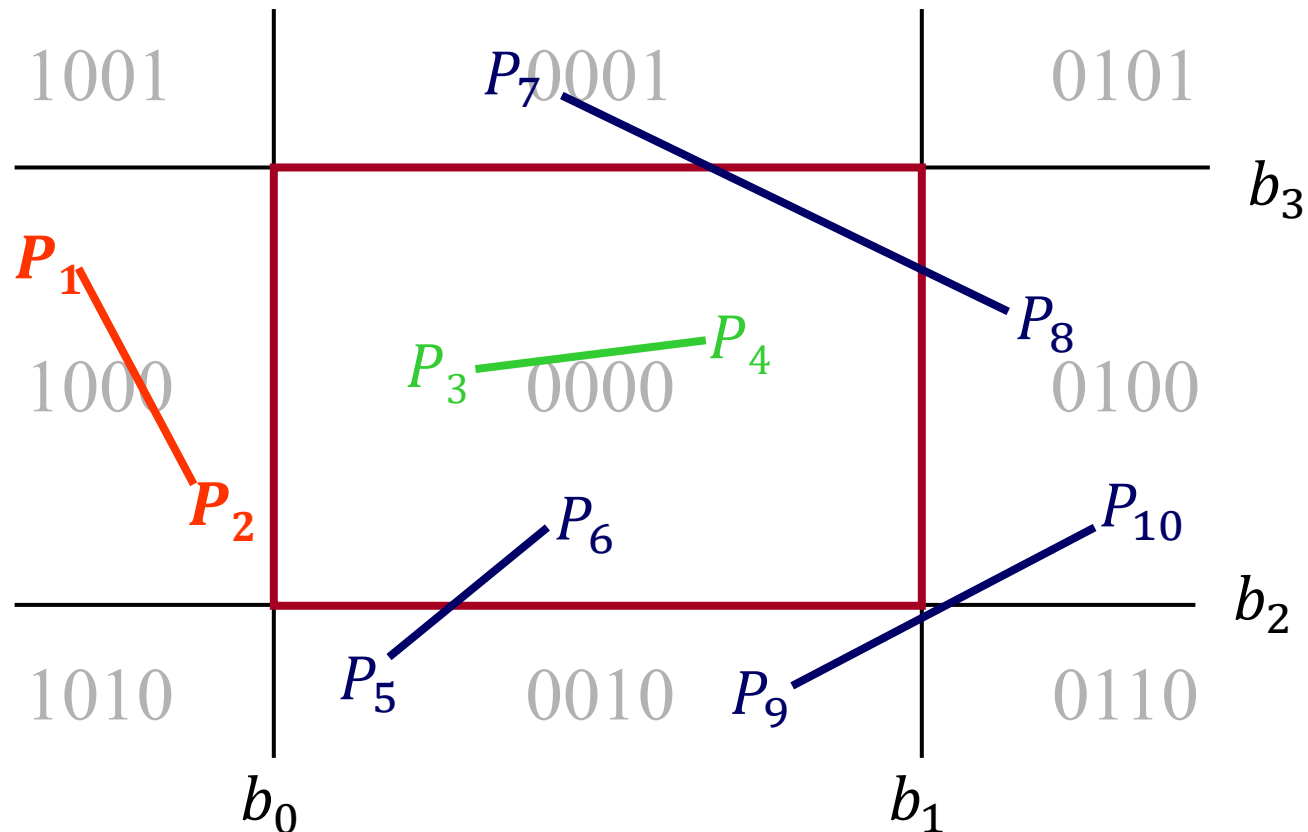


Cohen-Sutherland Line Clipping

Associate a 4-bit outcode $b_0b_1b_2b_3$ to each vertex

If both points' outcodes are 0, line segment is inside

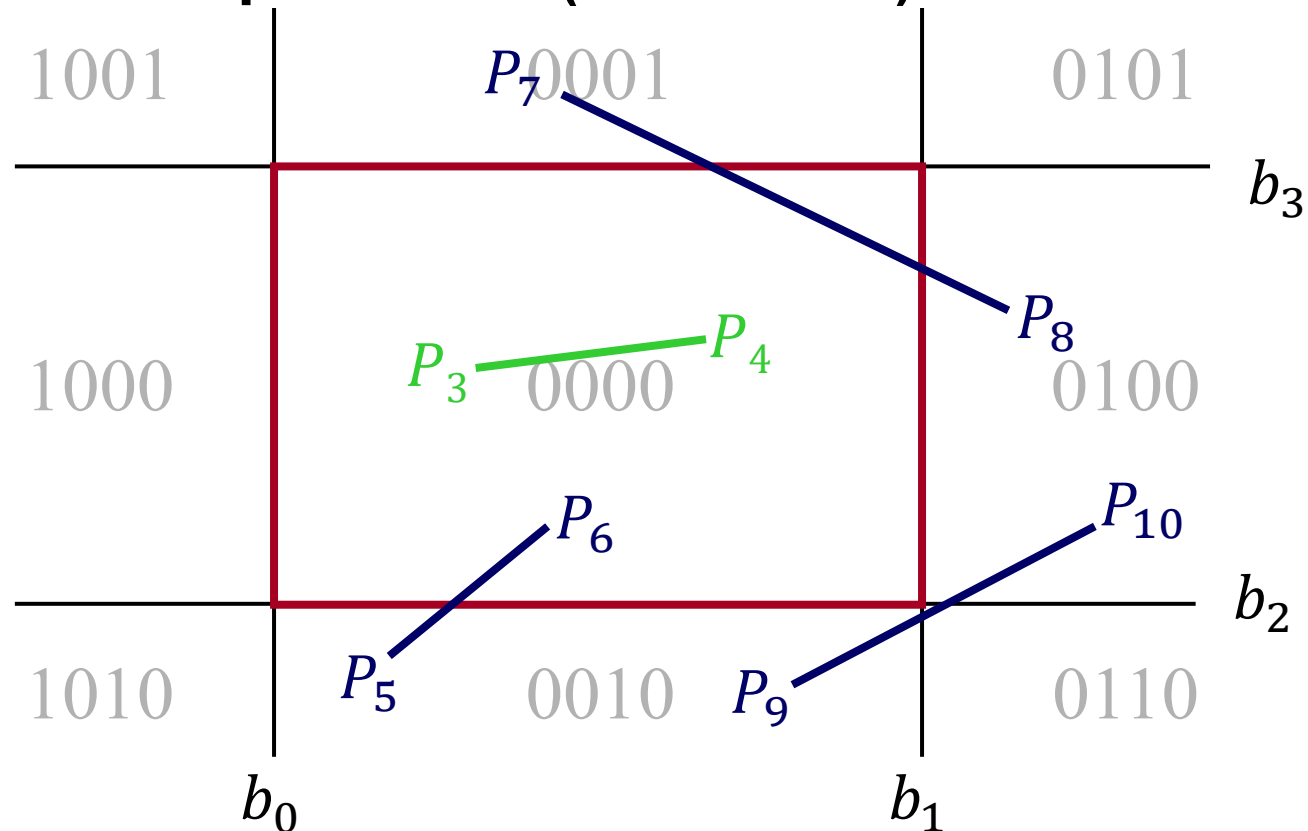
If AND of outcodes is not 0, line segment is outside





Cohen-Sutherland Line Clipping

Associate a 4-bit outcode $b_0b_1b_2b_3$ 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 (inside half)





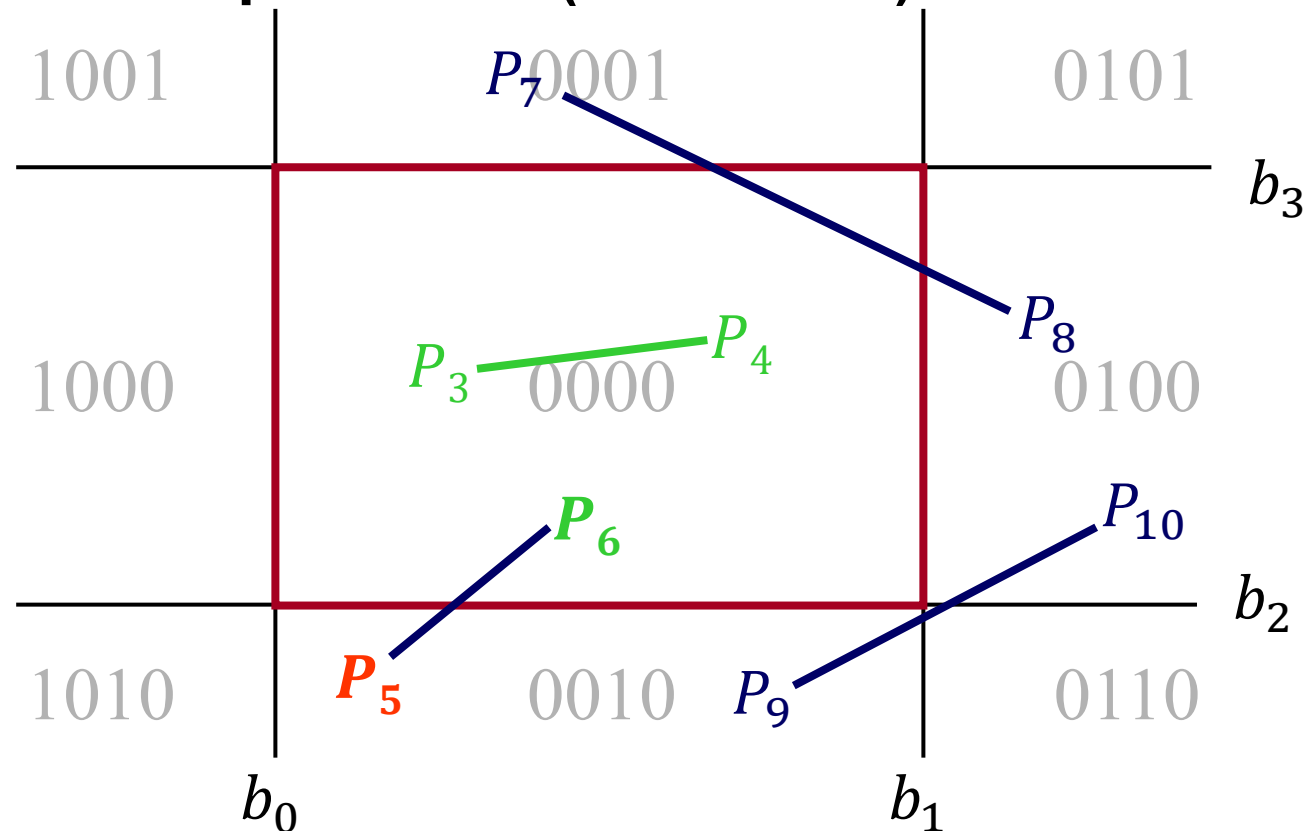
Cohen-Sutherland Line Clipping

Associate a 4-bit outcode $b_0b_1b_2b_3$ to each vertex

If both points' outcodes are 0, line segment is inside

If AND of outcodes is not 0, line segment is outside

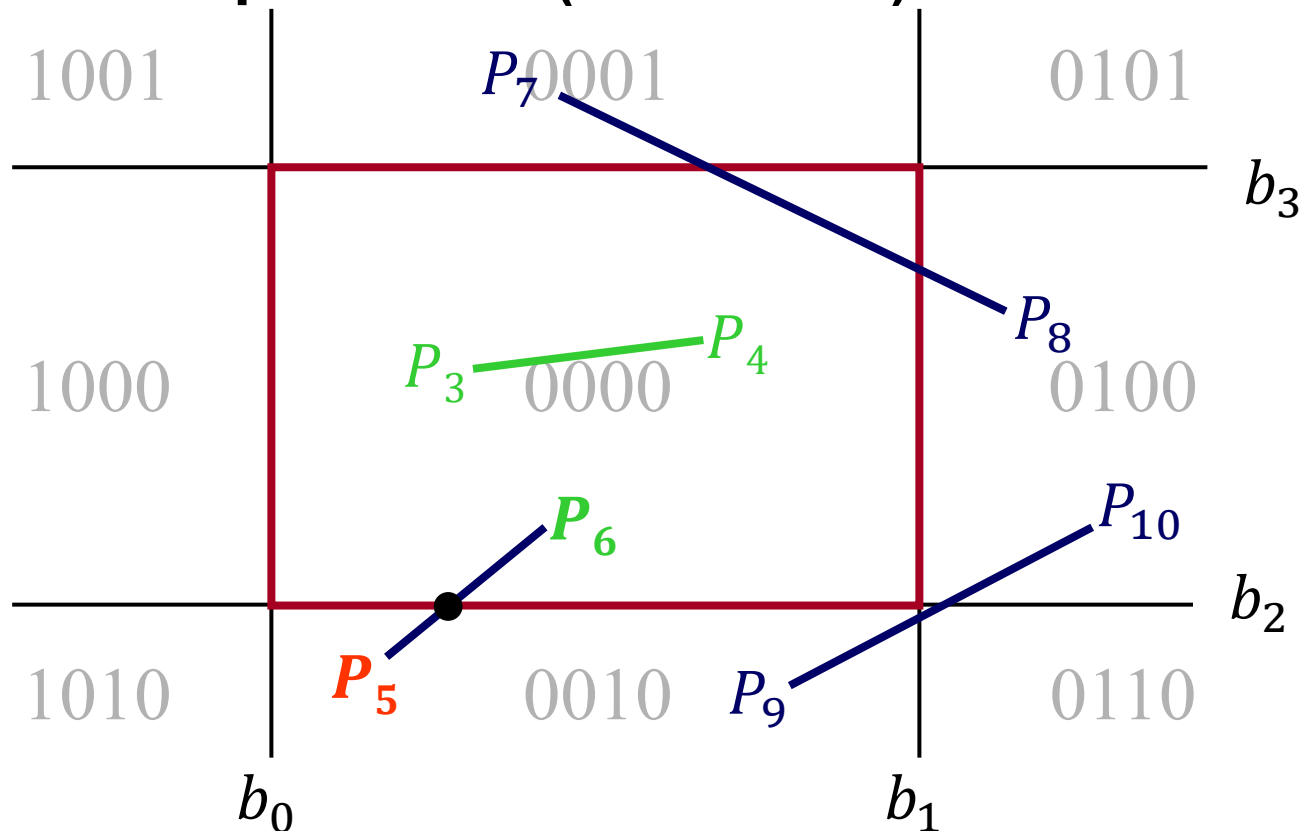
Otherwise clip and test (inside half)





Cohen-Sutherland Line Clipping

Associate a 4-bit outcode $b_0b_1b_2b_3$ 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 (inside half)





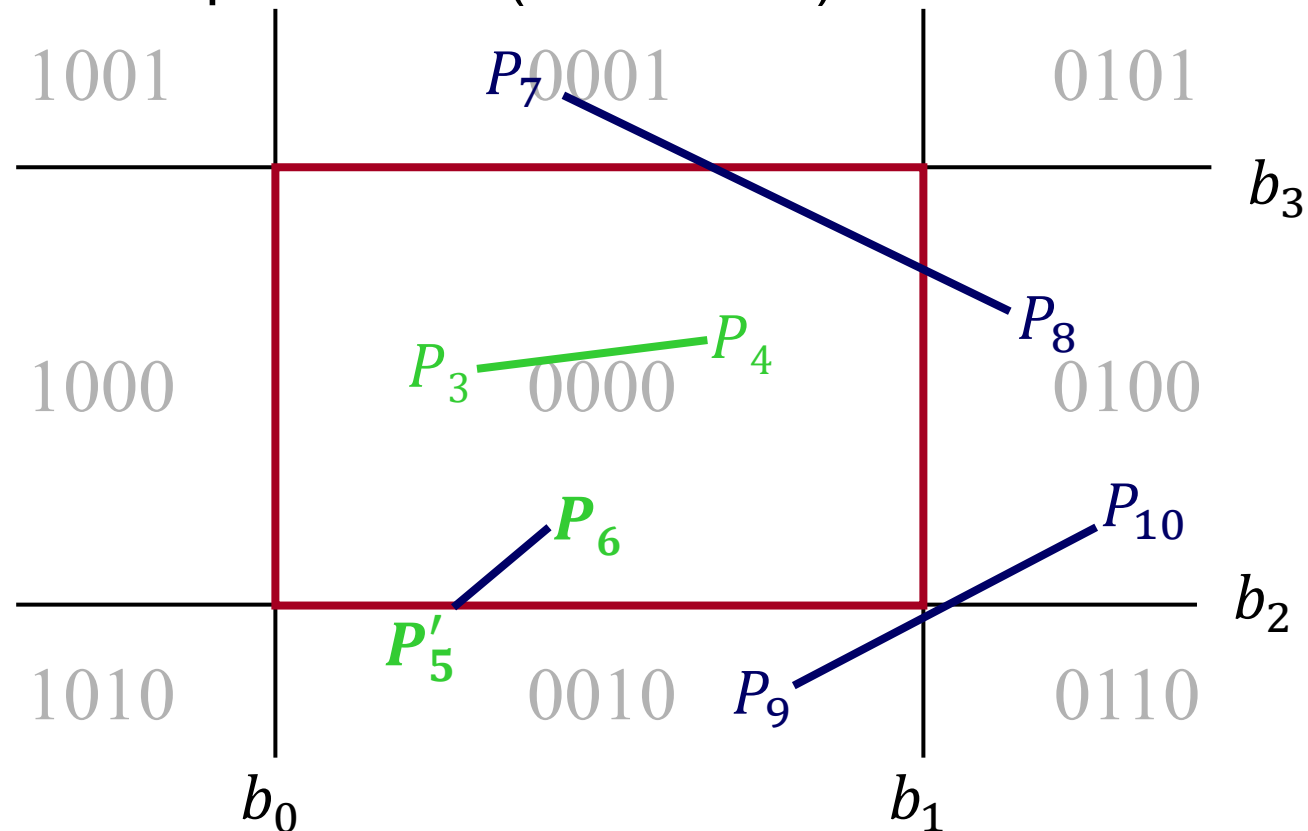
Cohen-Sutherland Line Clipping

Associate a 4-bit outcode $b_0b_1b_2b_3$ to each vertex

If both points' outcodes are 0, line segment is inside

If AND of outcodes is not 0, line segment is outside

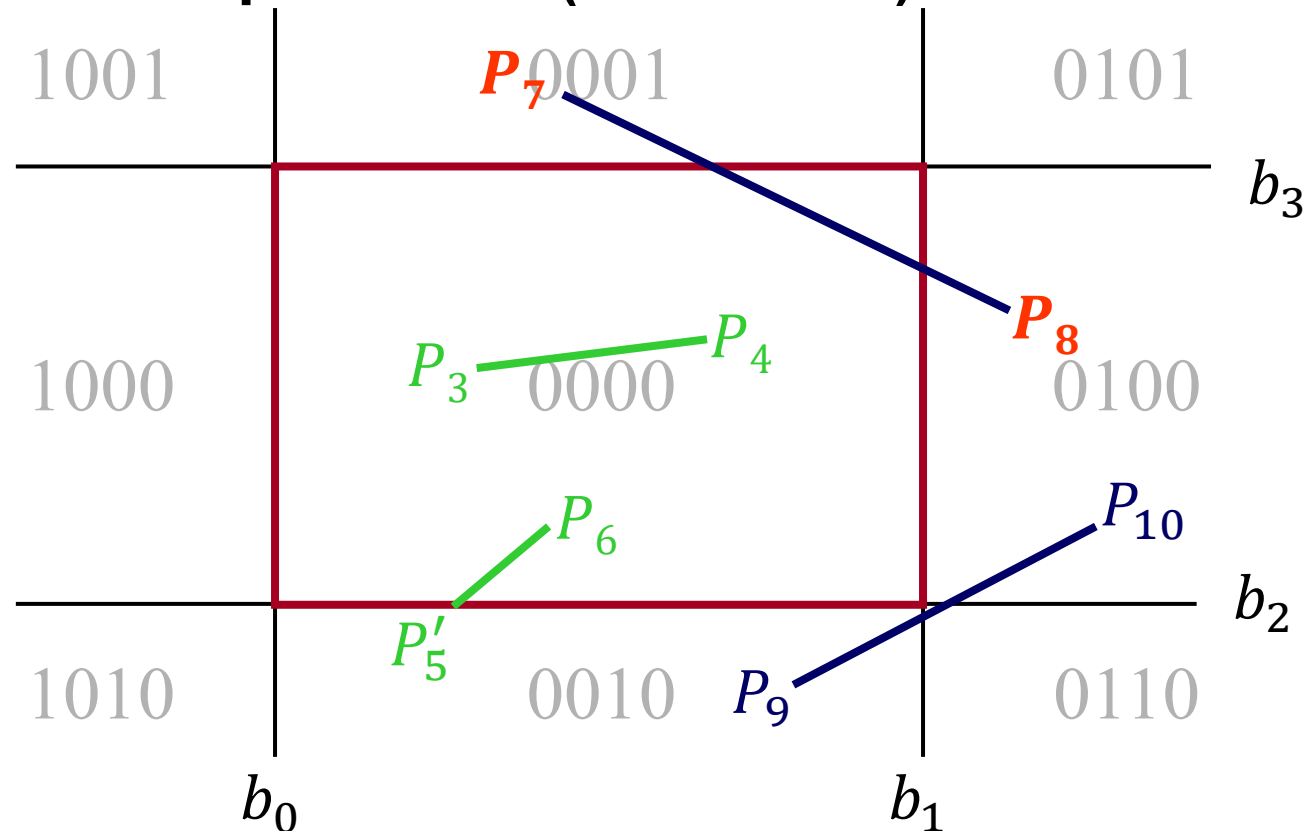
Otherwise clip and test (inside half)





Cohen-Sutherland Line Clipping

Associate a 4-bit outcode $b_0b_1b_2b_3$ 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 (inside half)





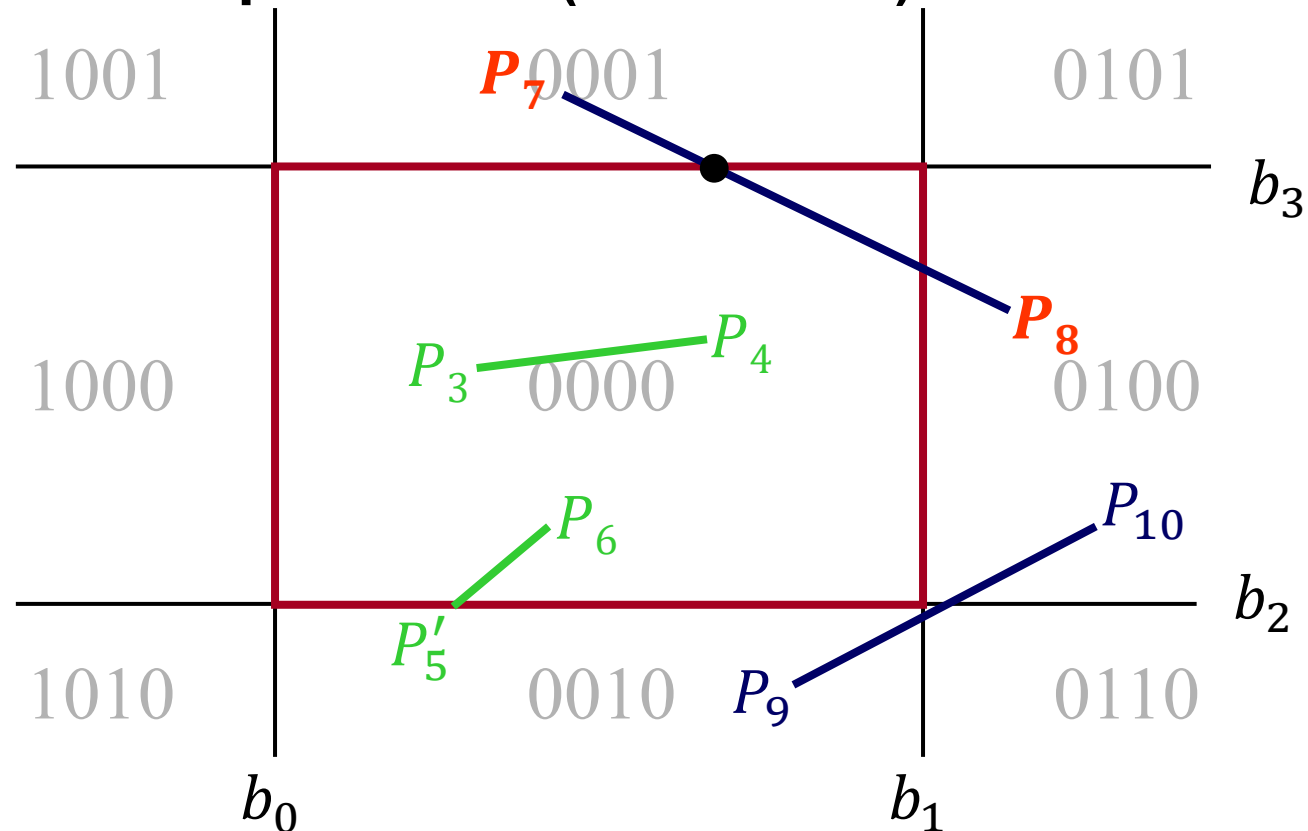
Cohen-Sutherland Line Clipping

Associate a 4-bit outcode $b_0b_1b_2b_3$ 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 (inside half)





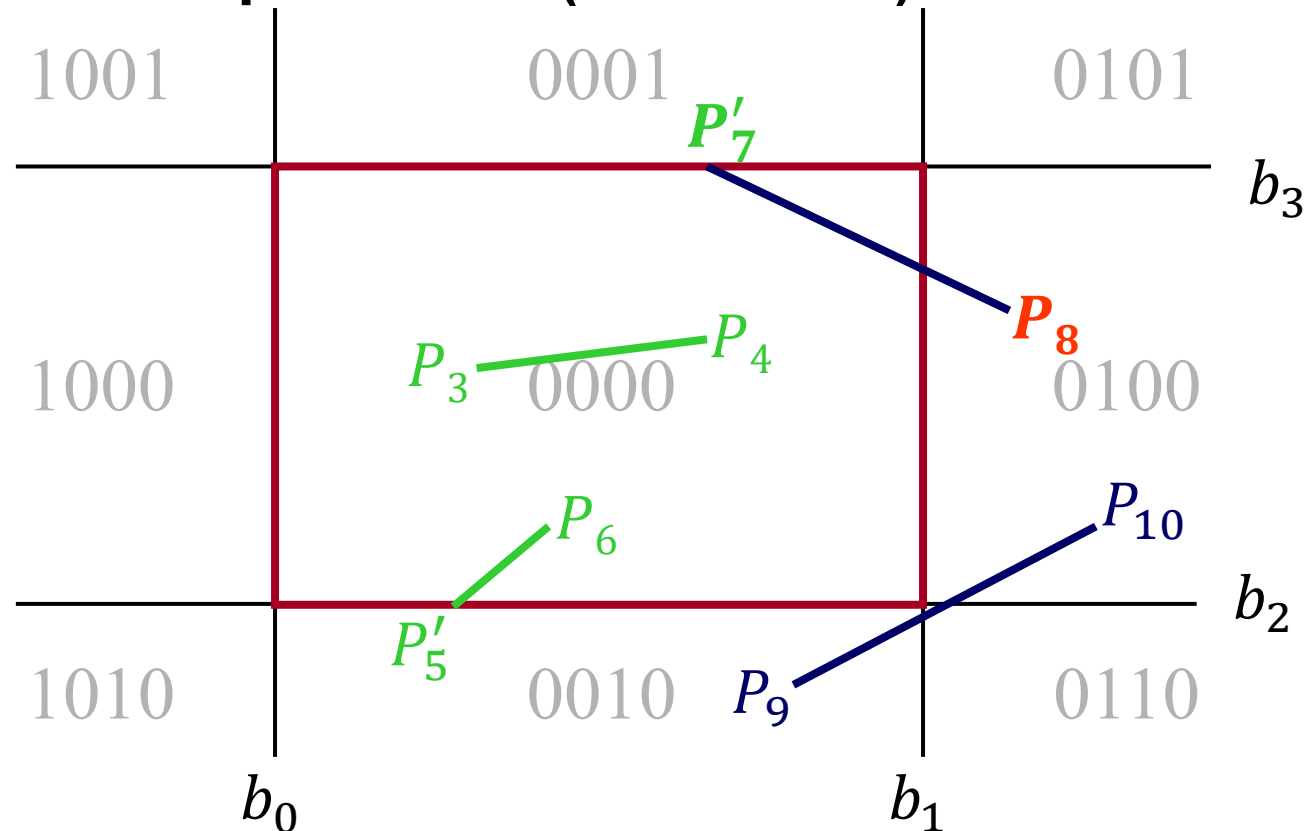
Cohen-Sutherland Line Clipping

Associate a 4-bit outcode $b_0b_1b_2b_3$ 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 (inside half)





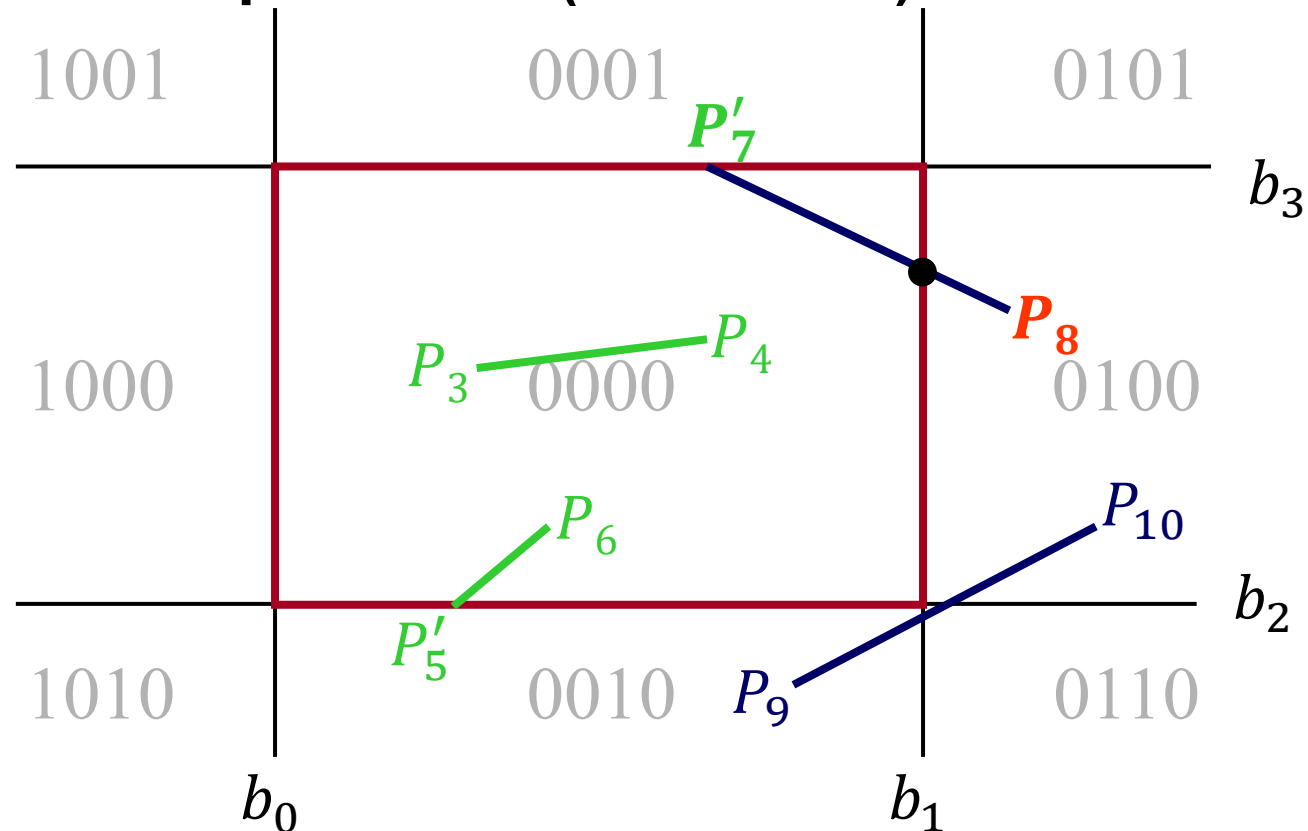
Cohen-Sutherland Line Clipping

Associate a 4-bit outcode $b_0b_1b_2b_3$ 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 (inside half)





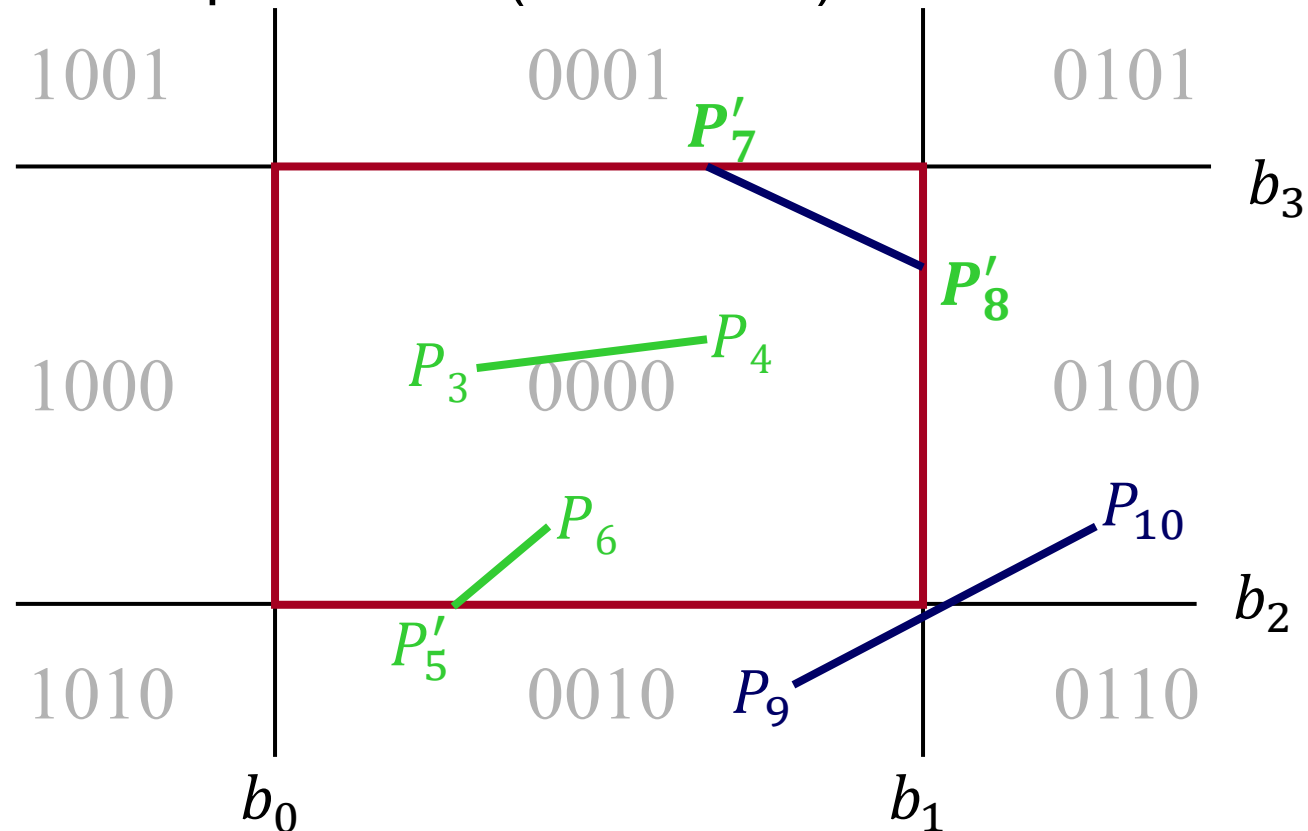
Cohen-Sutherland Line Clipping

Associate a 4-bit outcode $b_0b_1b_2b_3$ to each vertex

If both points' outcodes are 0, line segment is inside

If AND of outcodes is not 0, line segment is outside

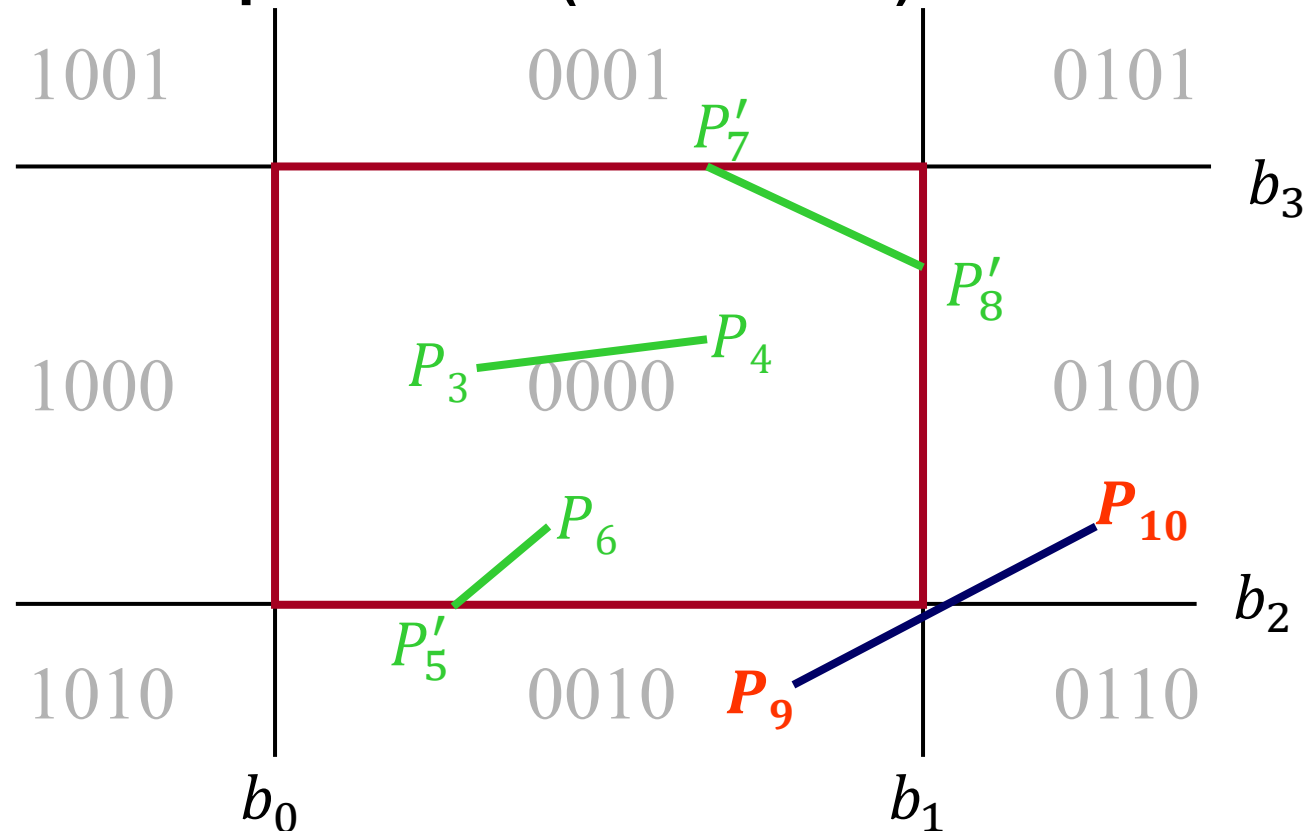
Otherwise clip and test (inside half)





Cohen-Sutherland Line Clipping

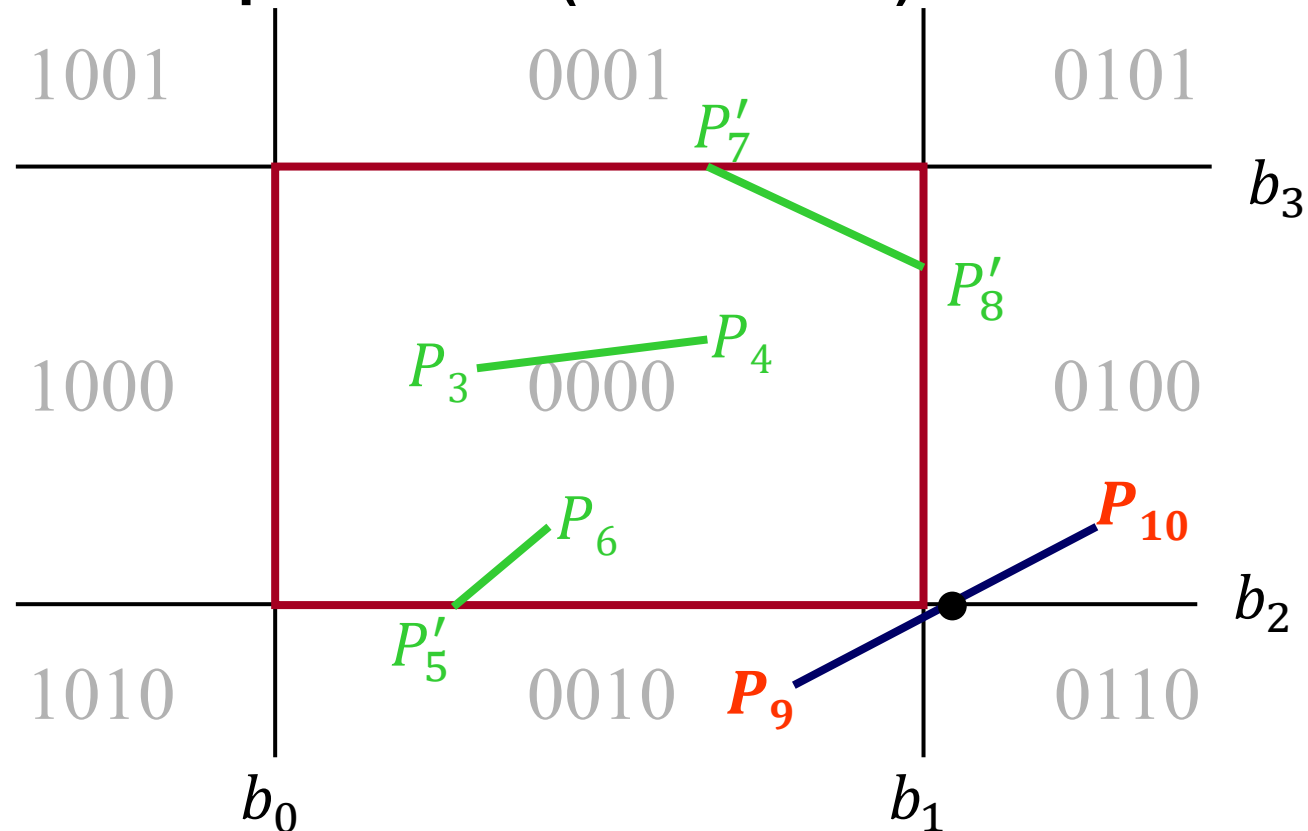
Associate a 4-bit outcode $b_0b_1b_2b_3$ to each vertex
If both points' outcodes are 0, line segment is inside
If AND of outcodes is not 0, line segment is outside
Otherwise clip and test (inside half)





Cohen-Sutherland Line Clipping

Associate a 4-bit outcode $b_0b_1b_2b_3$ 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 (inside half)





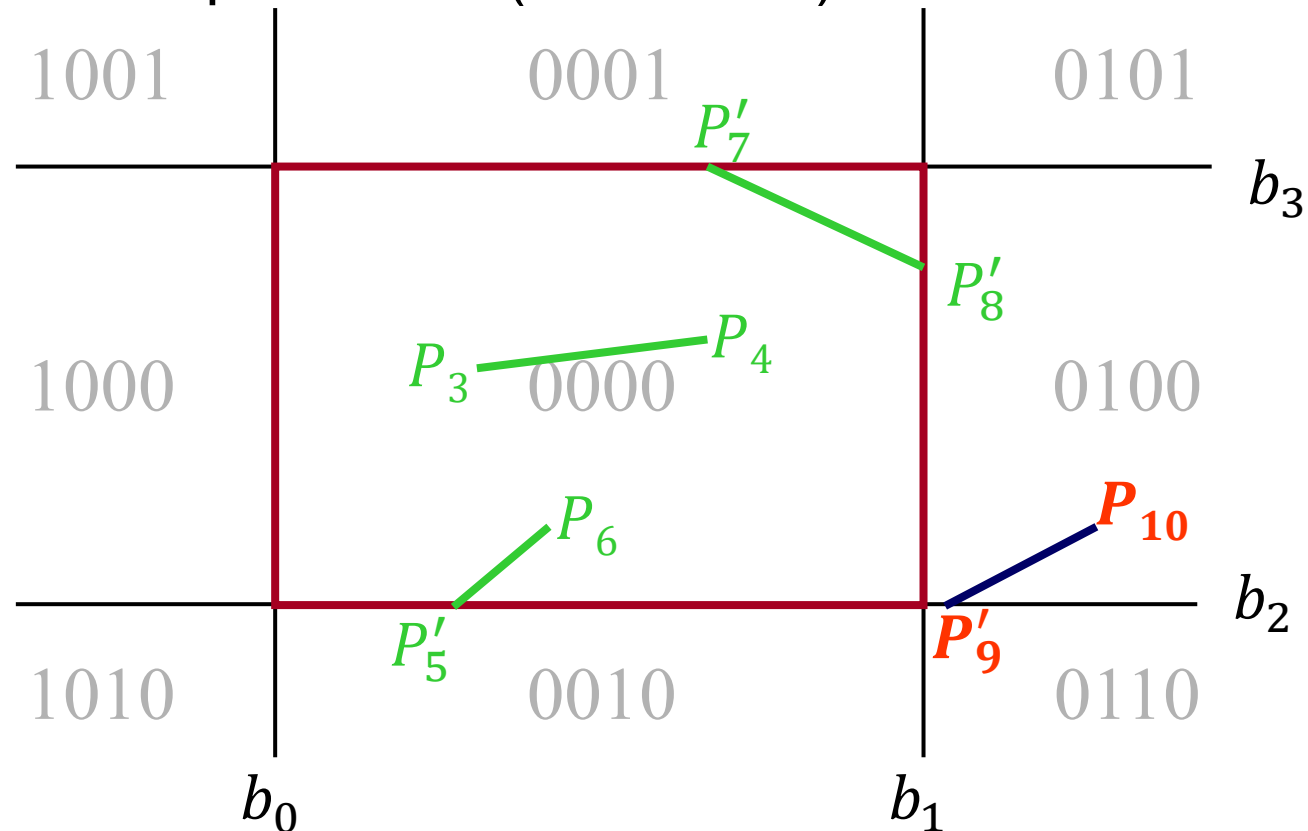
Cohen-Sutherland Line Clipping

Associate a 4-bit outcode $b_0b_1b_2b_3$ 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 (inside half)





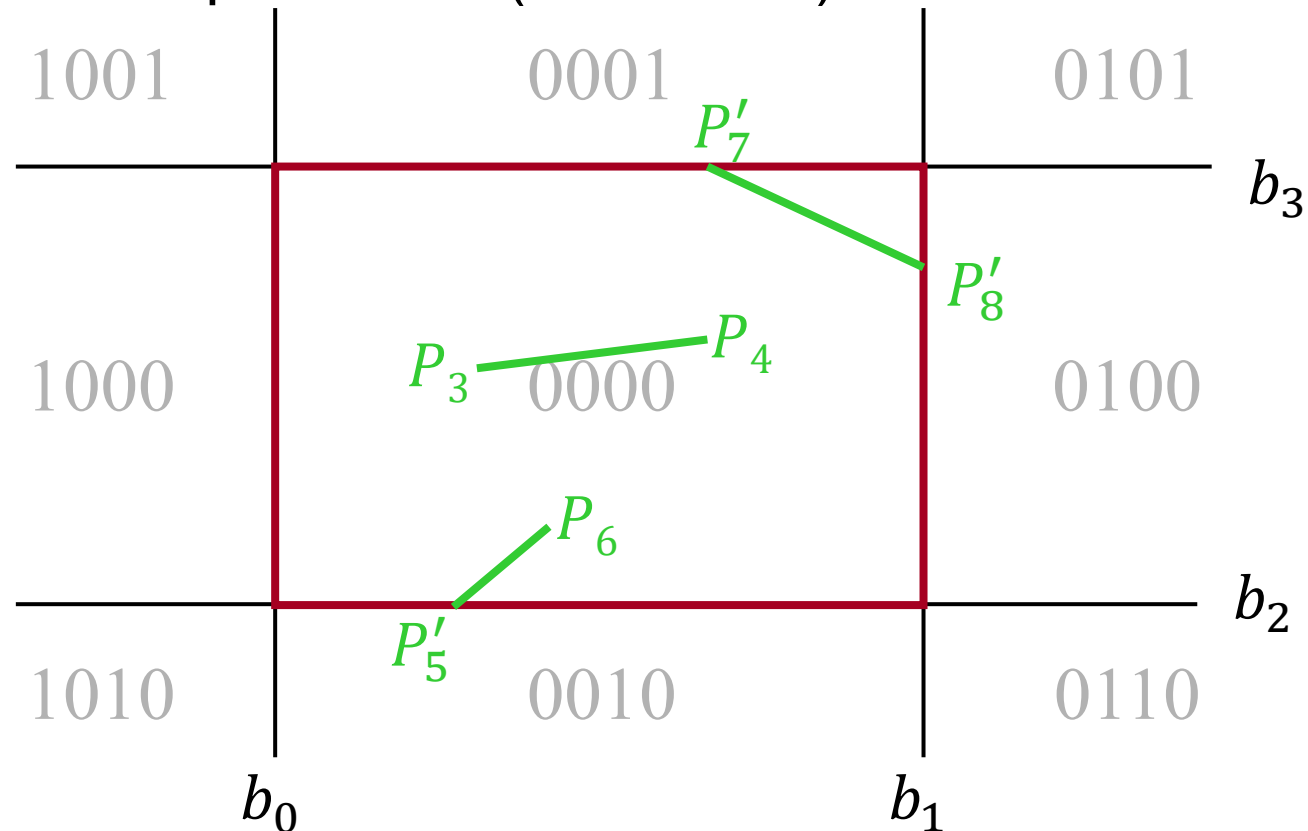
Cohen-Sutherland Line Clipping

Associate a 4-bit outcode $b_0b_1b_2b_3$ to each vertex

If both points' outcodes are 0, line segment is inside

If AND of outcodes is not 0, line segment is outside

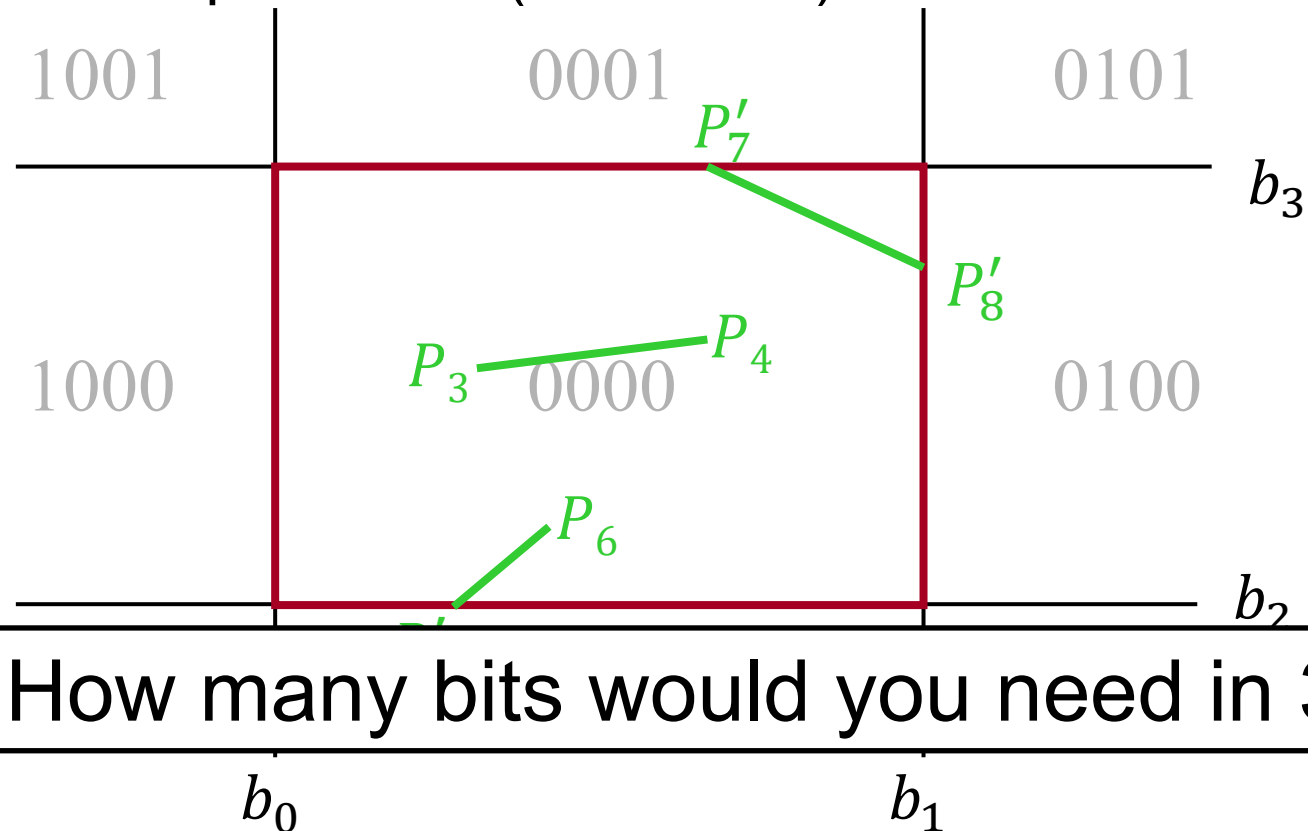
Otherwise clip and test (inside half)





Cohen-Sutherland Line Clipping

Associate a 4-bit outcode $b_0b_1b_2b_3$ 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 (inside half)





Clipping

Avoid drawing parts of primitives outside window

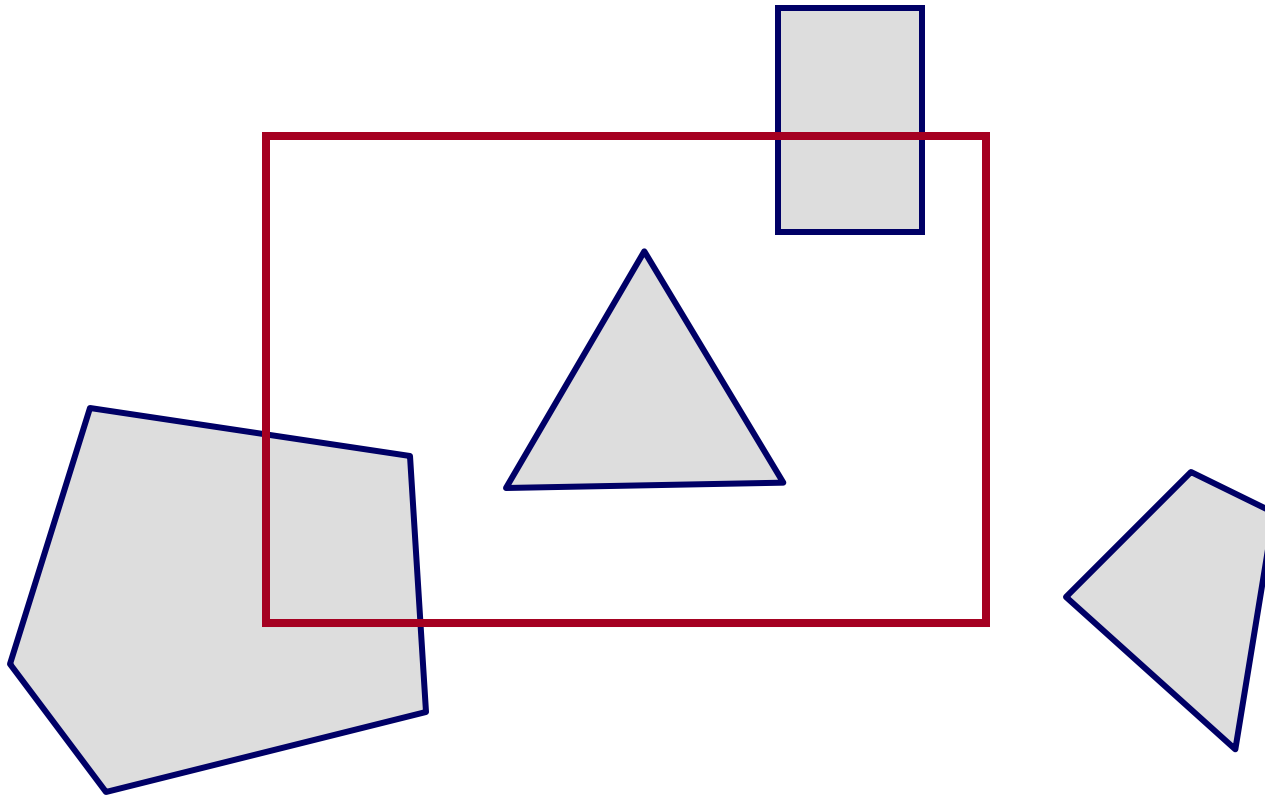
- Points
- Line Segments
- **Polygons**





Polygon Clipping

Find the part of a polygon inside the clip window

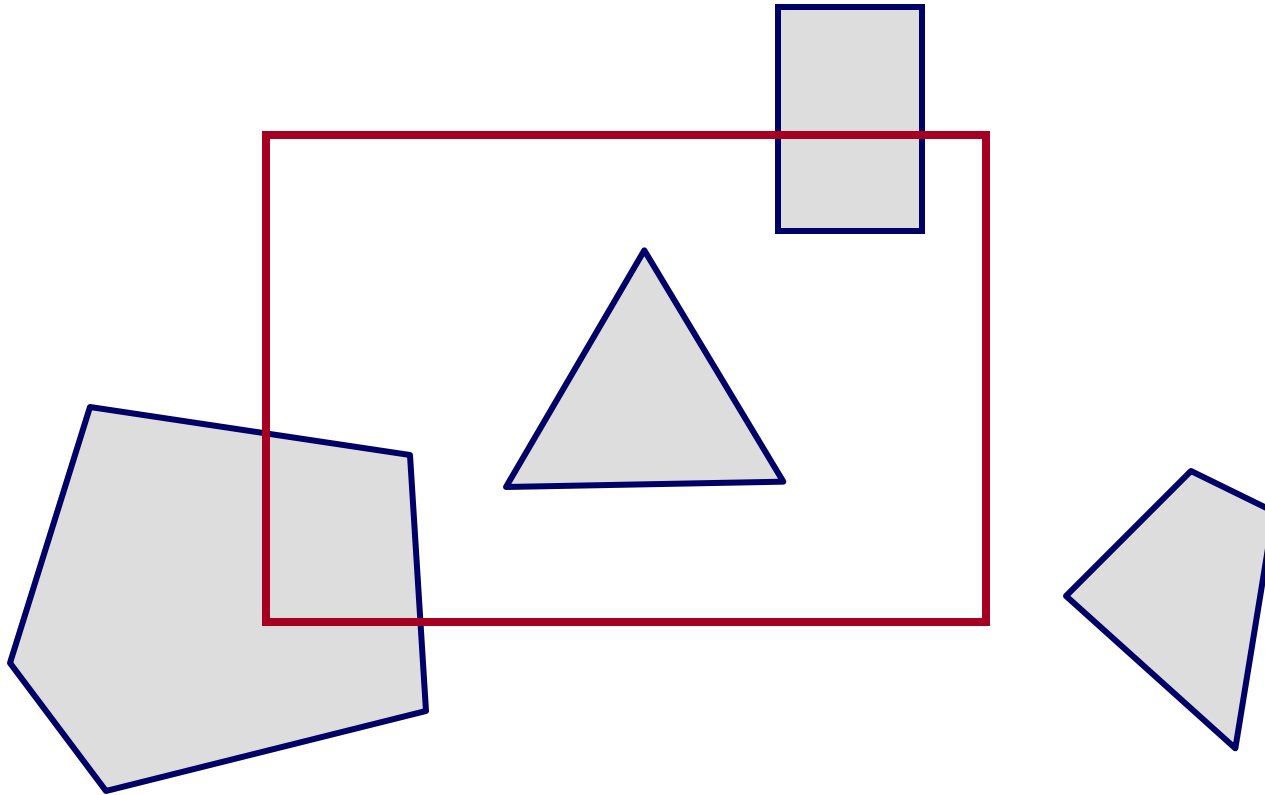


Before Clipping



Sutherland-Hodgeman Clipping

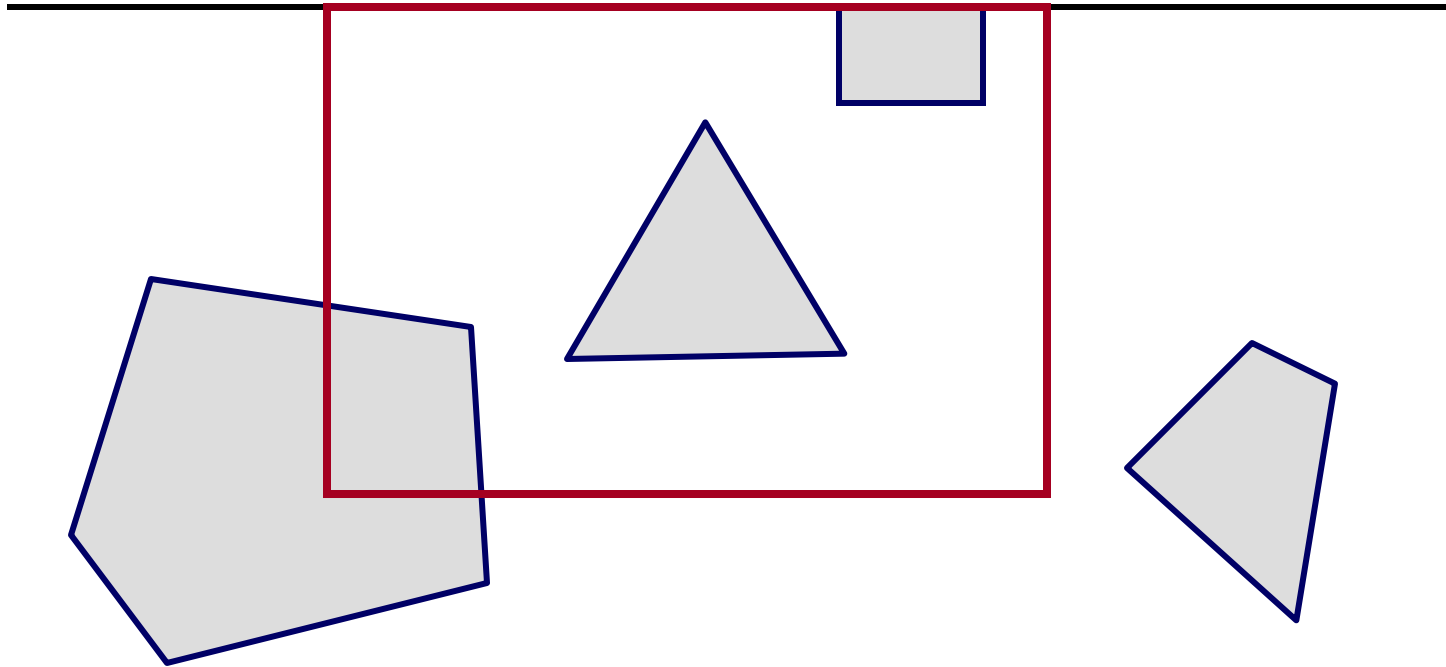
Clip to each window boundary, one at a time





Sutherland-Hodgeman Clipping

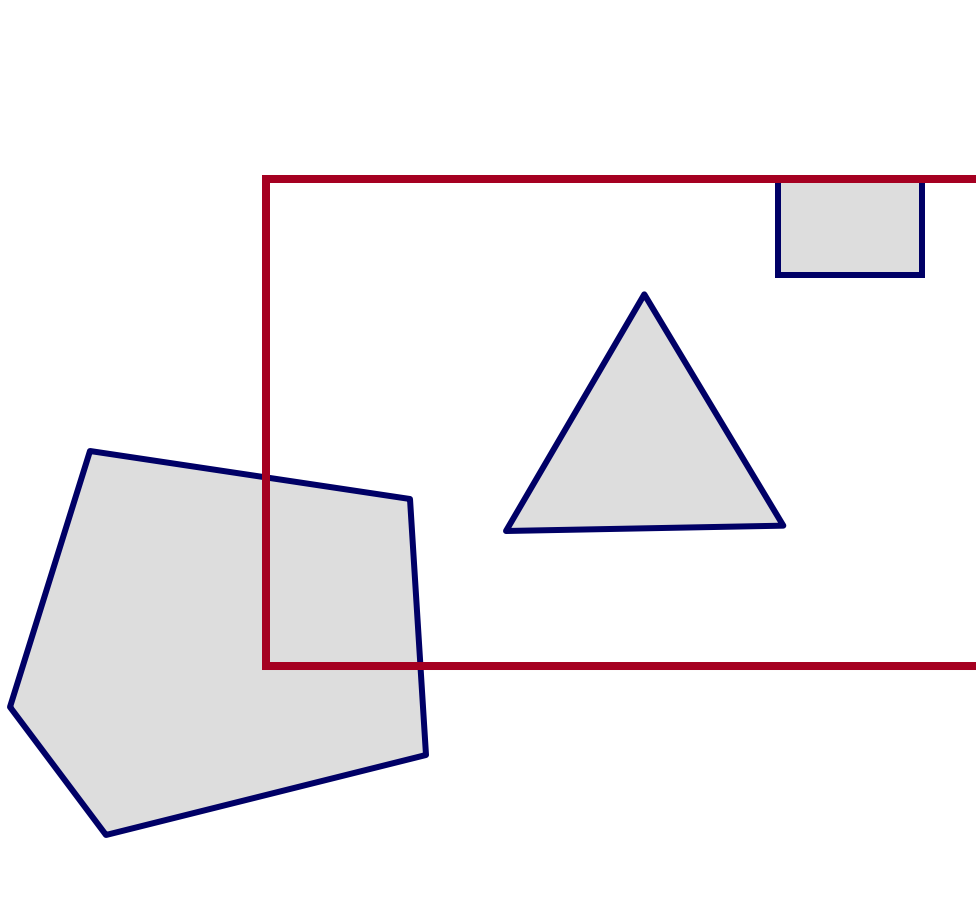
Clip to each window boundary, one at a time





Sutherland-Hodgeman Clipping

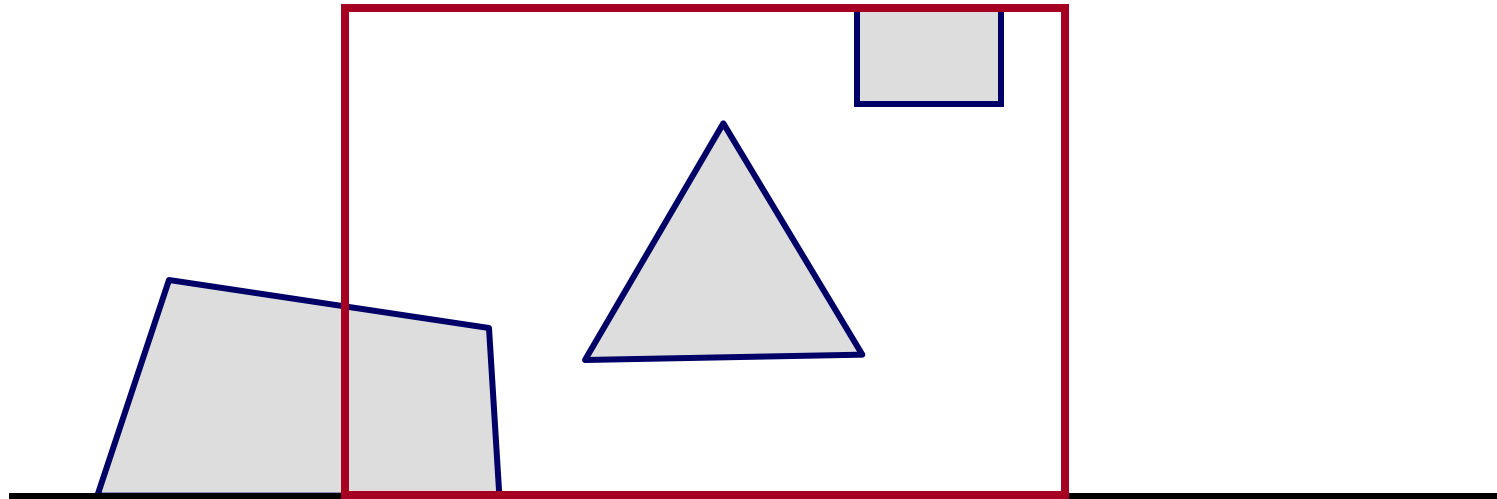
Clip to each window boundary, one at a time





Sutherland-Hodgeman Clipping

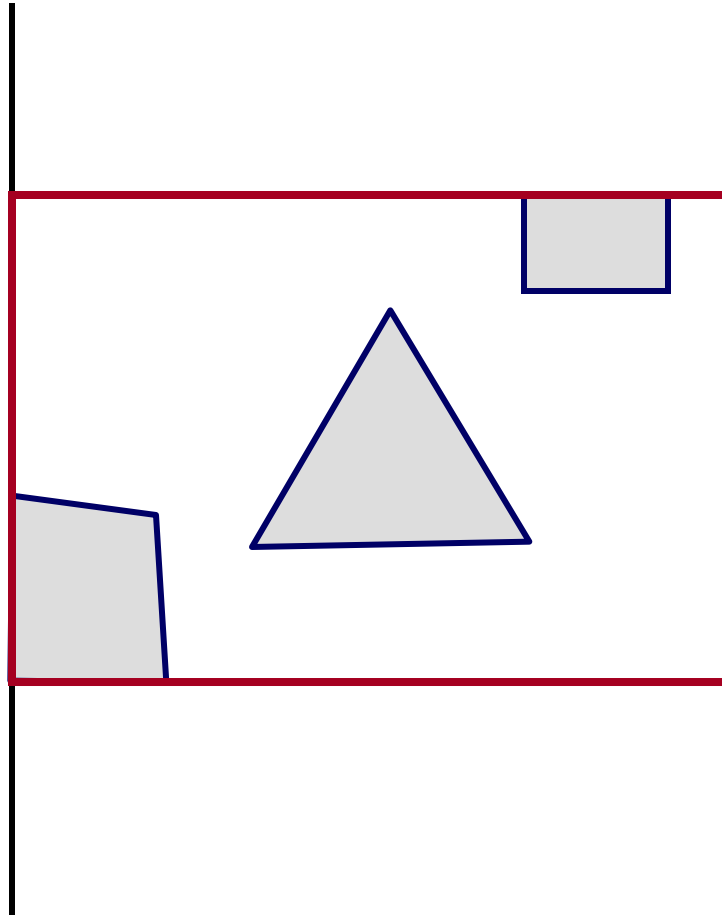
Clip to each window boundary, one at a time





Sutherland-Hodgeman Clipping

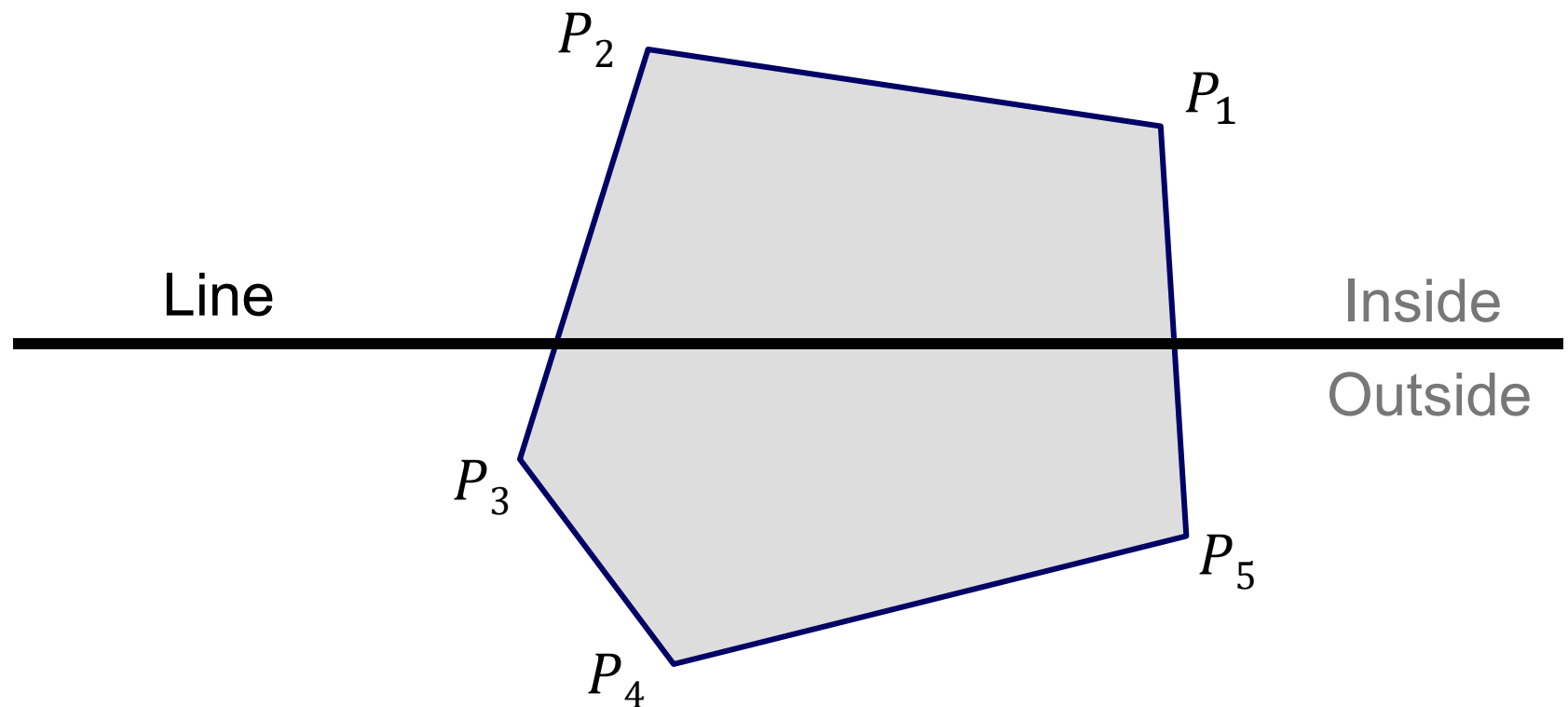
Clip to each window boundary, one at a time





Sutherland-Hodgeman Clipping

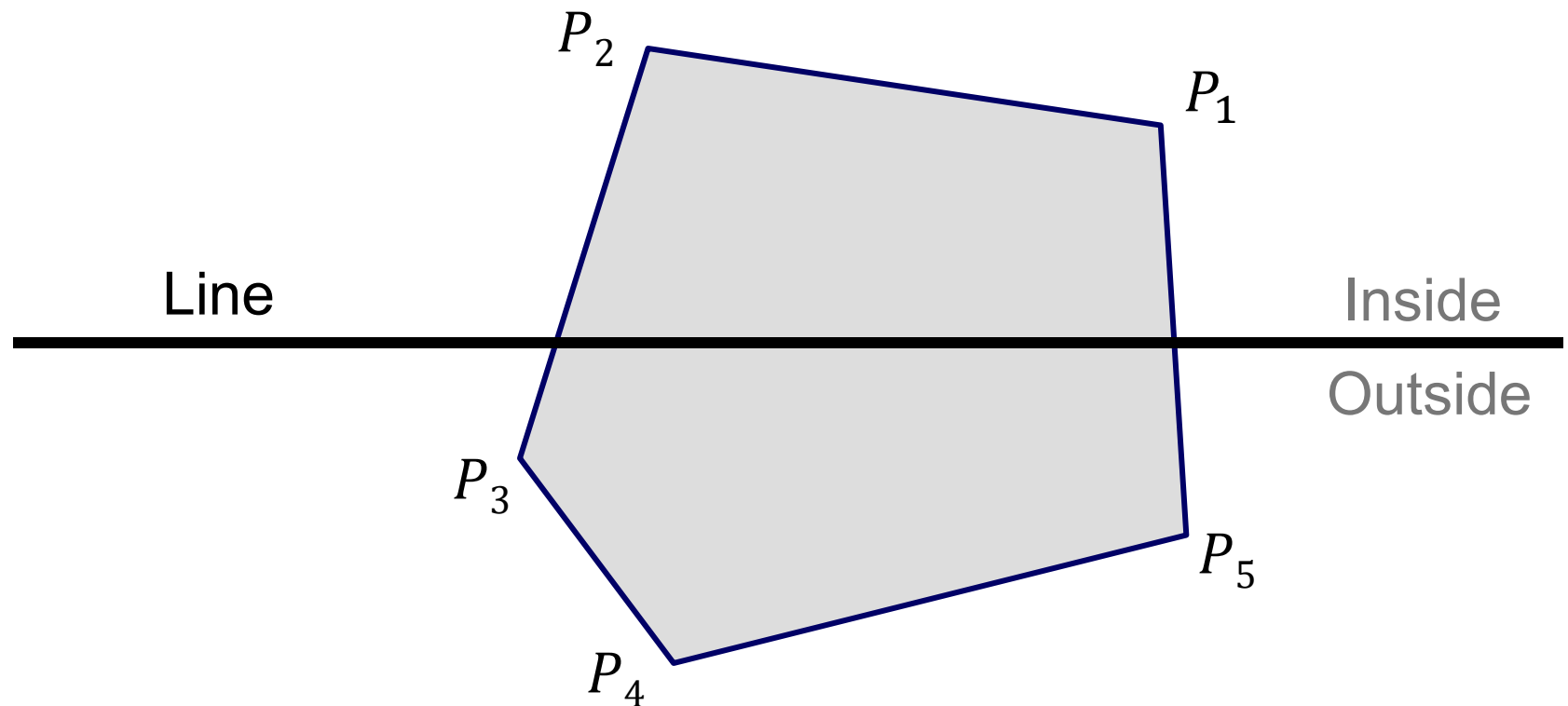
How do we clip a convex polygon with respect to a (window boundary) line?





Sutherland-Hodgeman Clipping

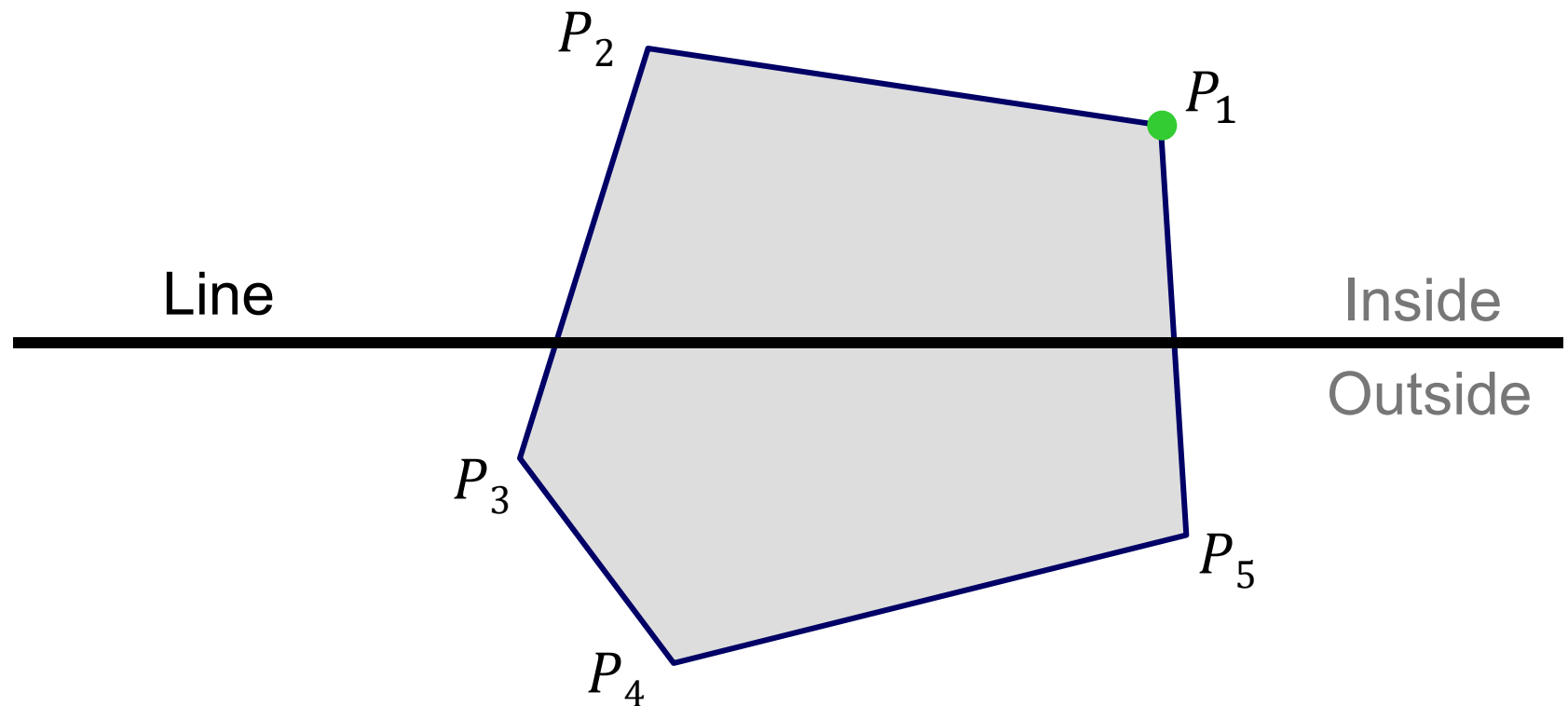
- In sequence, do interior test for each point
- Insert a new point when crossing the line.
- Remove points outside the line.





Sutherland-Hodgeman Clipping

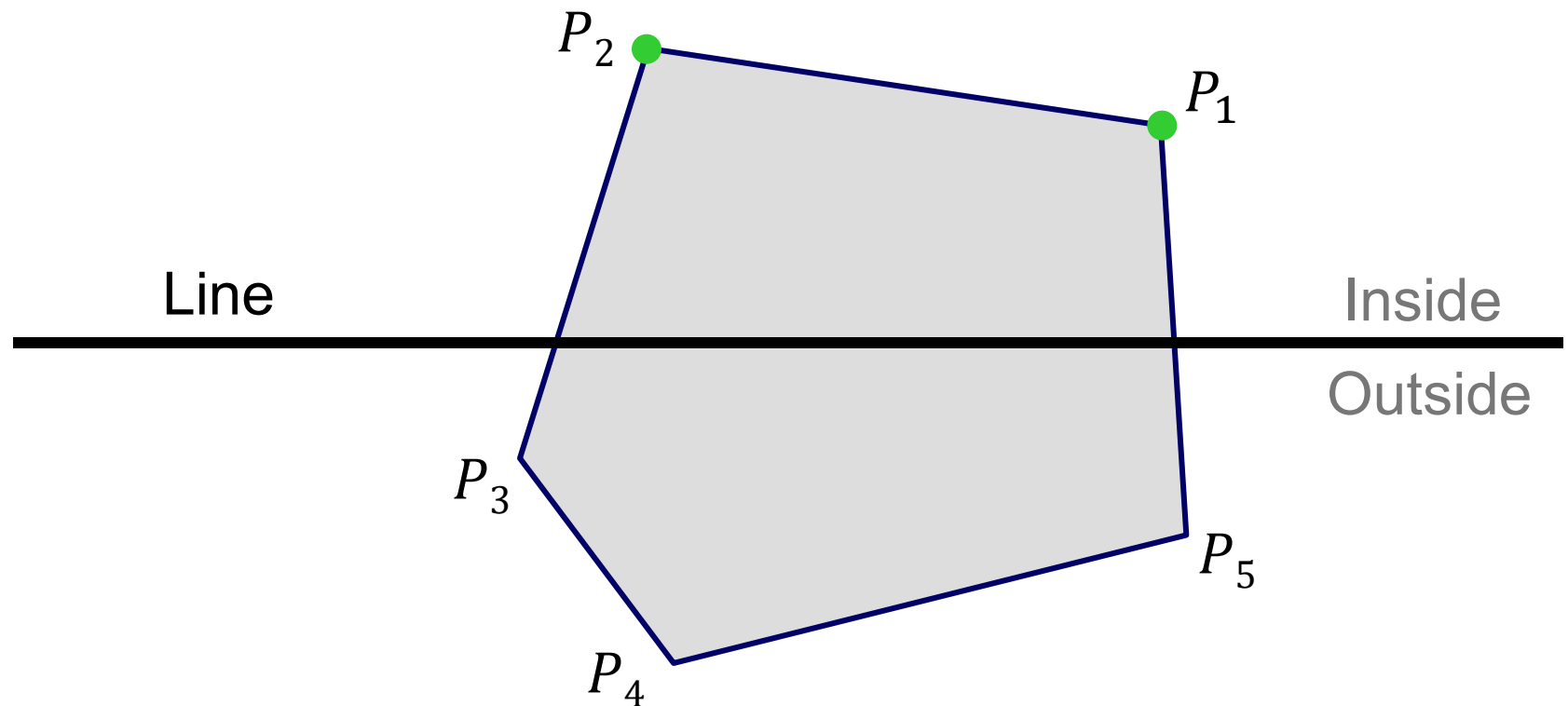
- In sequence, do interior test for each point
- Insert a new point when crossing the line.
- Remove points outside the line.





Sutherland-Hodgeman Clipping

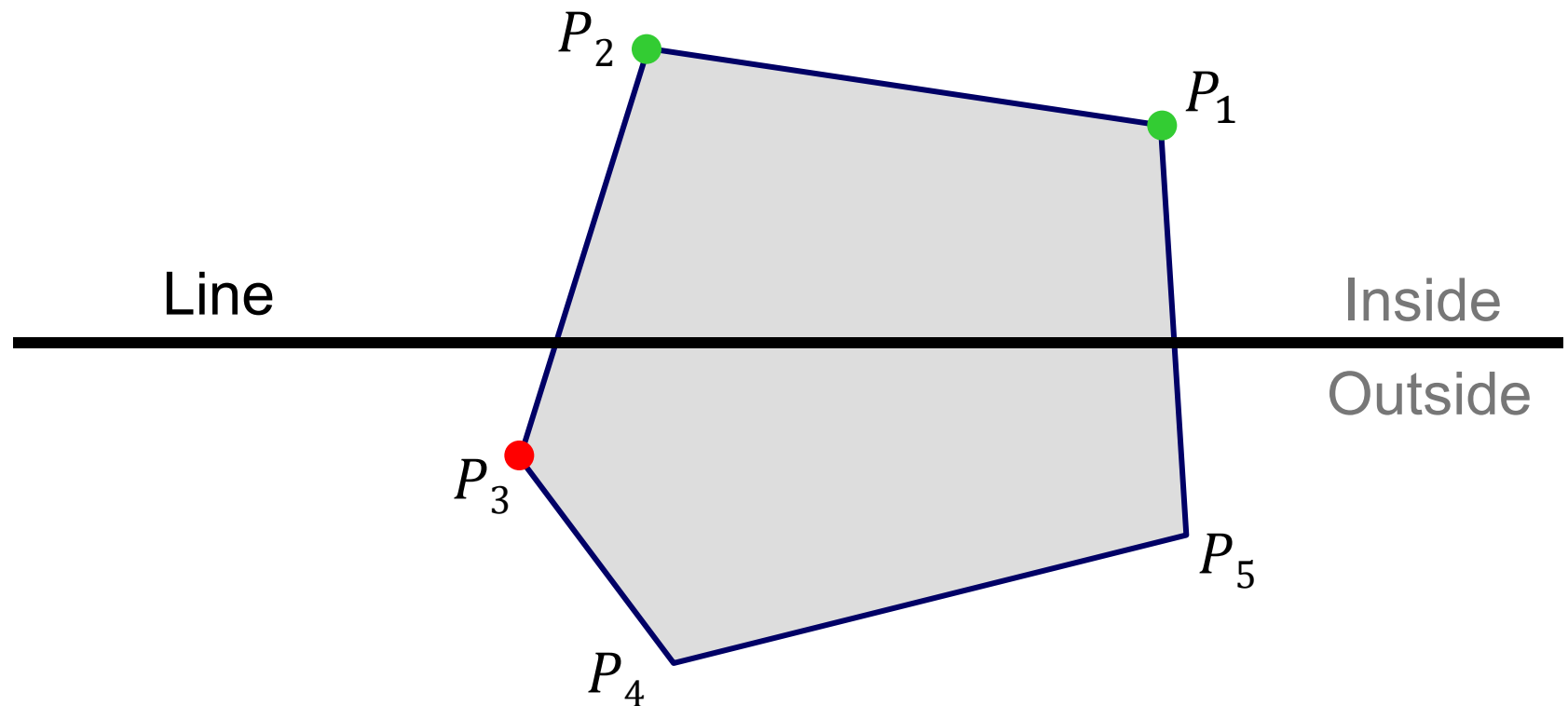
- In sequence, do interior test for each point
- Insert a new point when crossing the line.
- Remove points outside the line.





Sutherland-Hodgeman Clipping

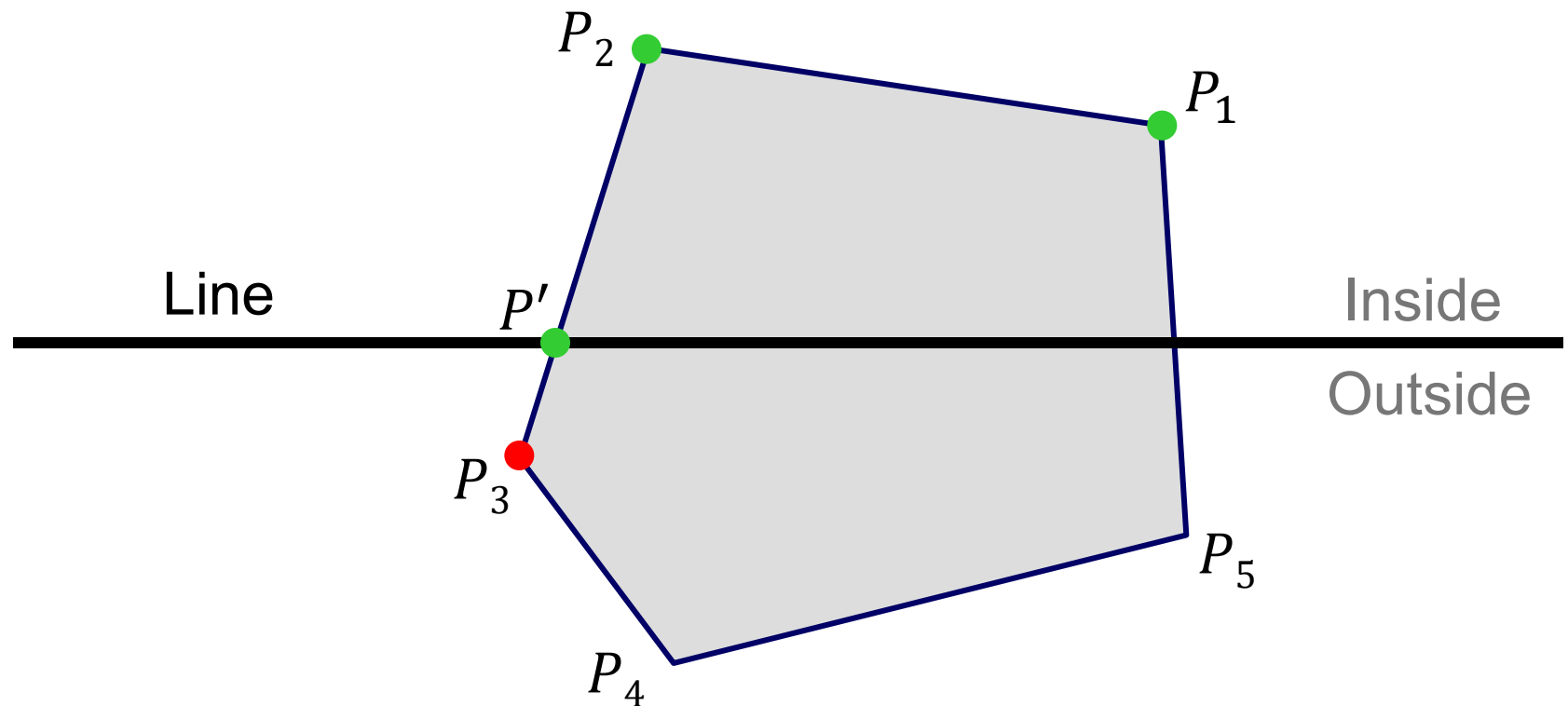
- In sequence, do interior test for each point
- Insert a new point when crossing the line.
- Remove points outside the line.





Sutherland-Hodgeman Clipping

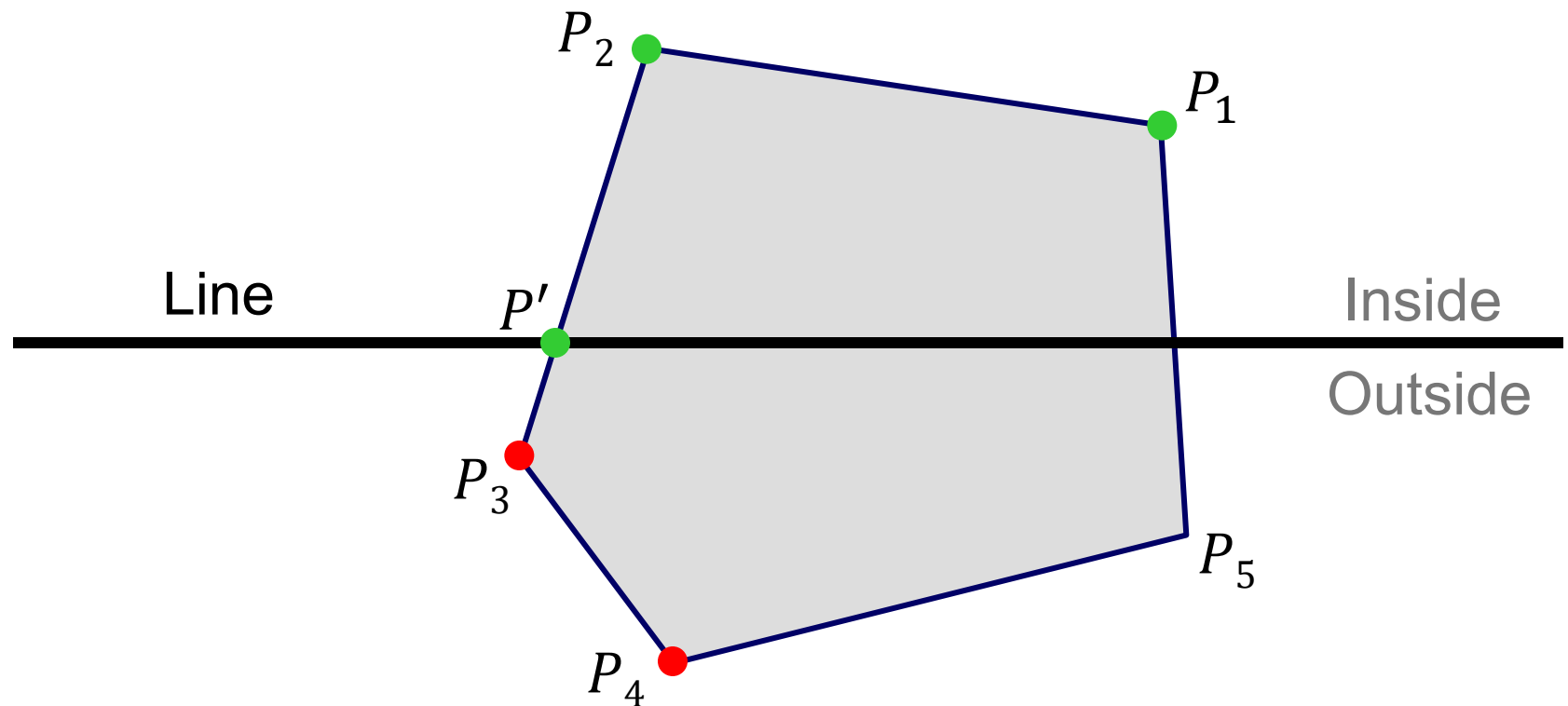
- In sequence, do interior test for each point
- Insert a new point when crossing the line.
- Remove points outside the line.





Sutherland-Hodgeman Clipping

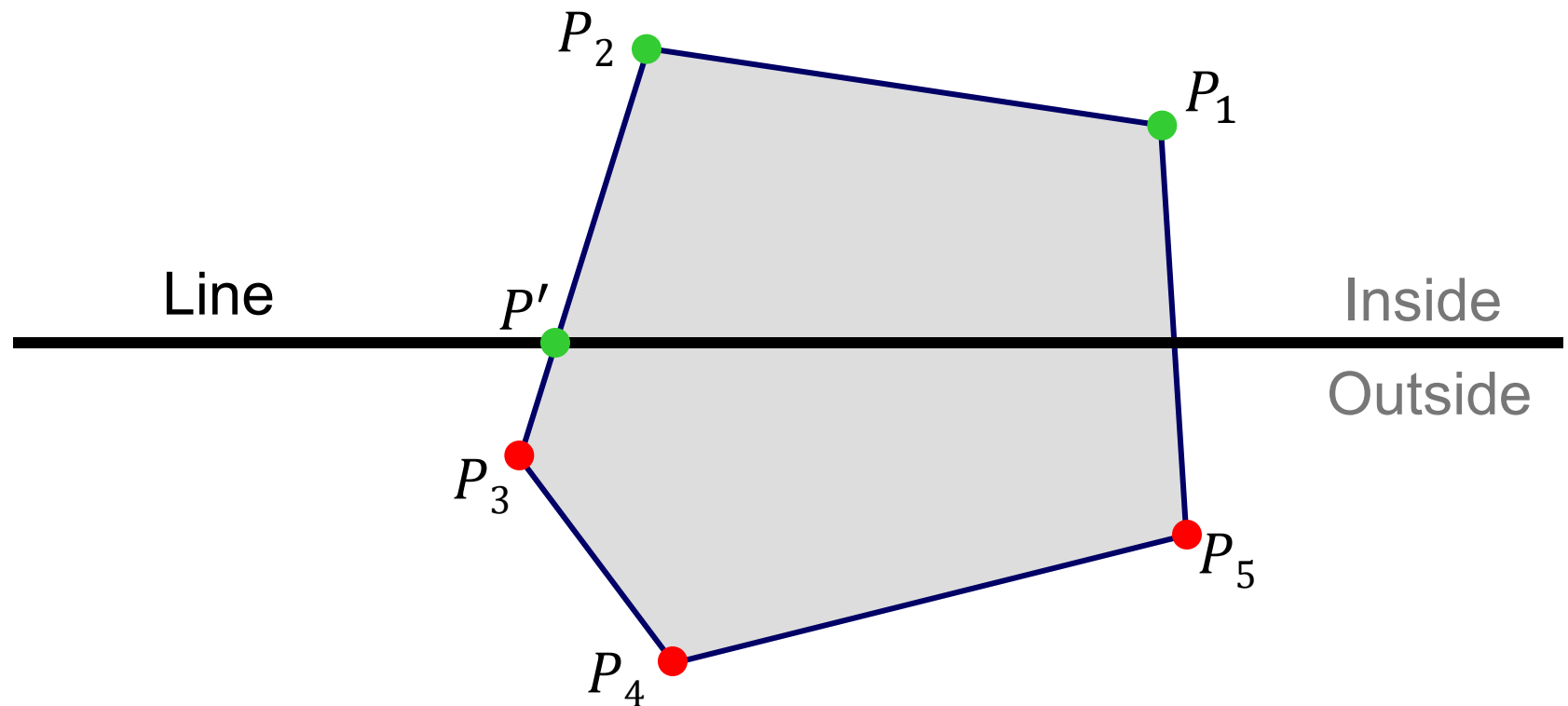
- In sequence, do interior test for each point
- Insert a new point when crossing the line.
- Remove points outside the line.





Sutherland-Hodgeman Clipping

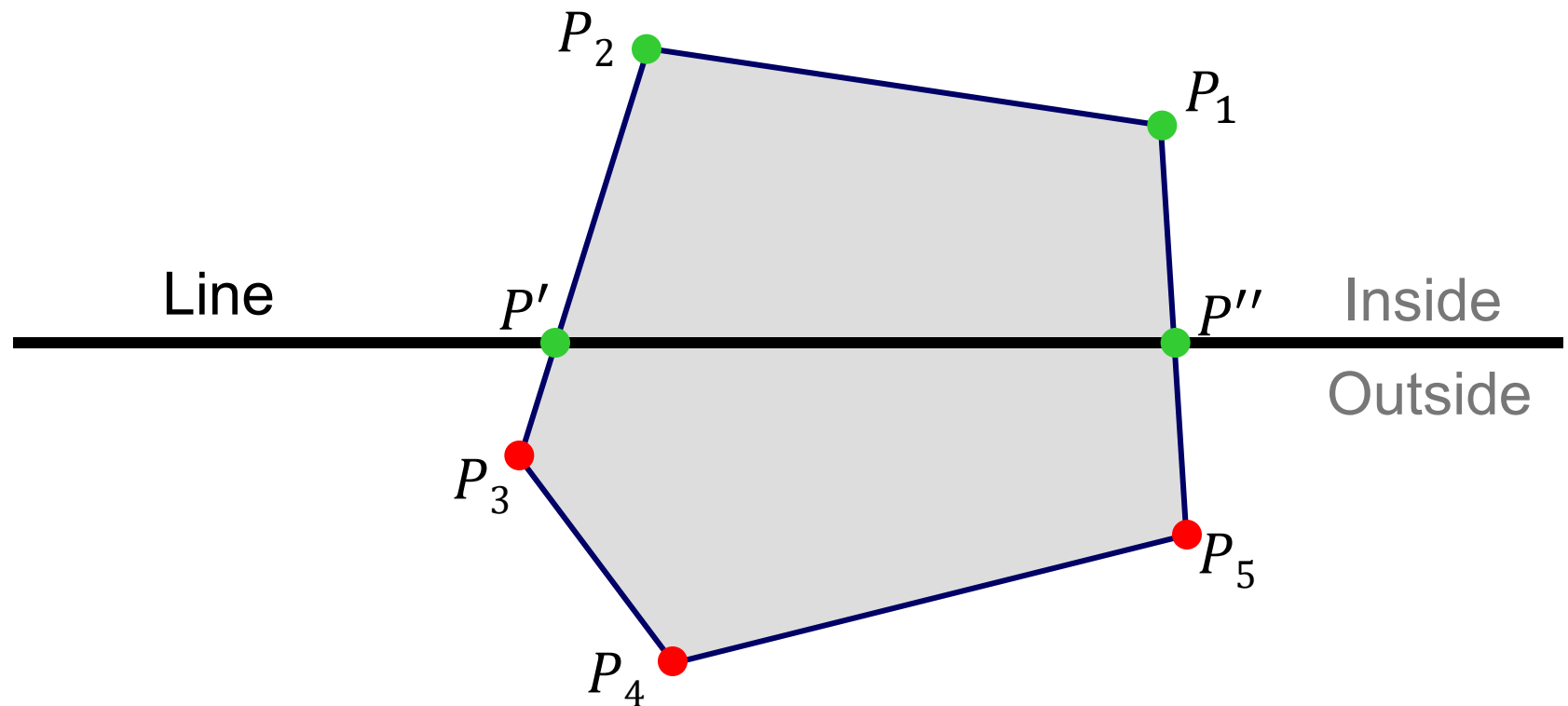
- In sequence, do interior test for each point
- Insert a new point when crossing the line.
- Remove points outside the line.





Sutherland-Hodgeman Clipping

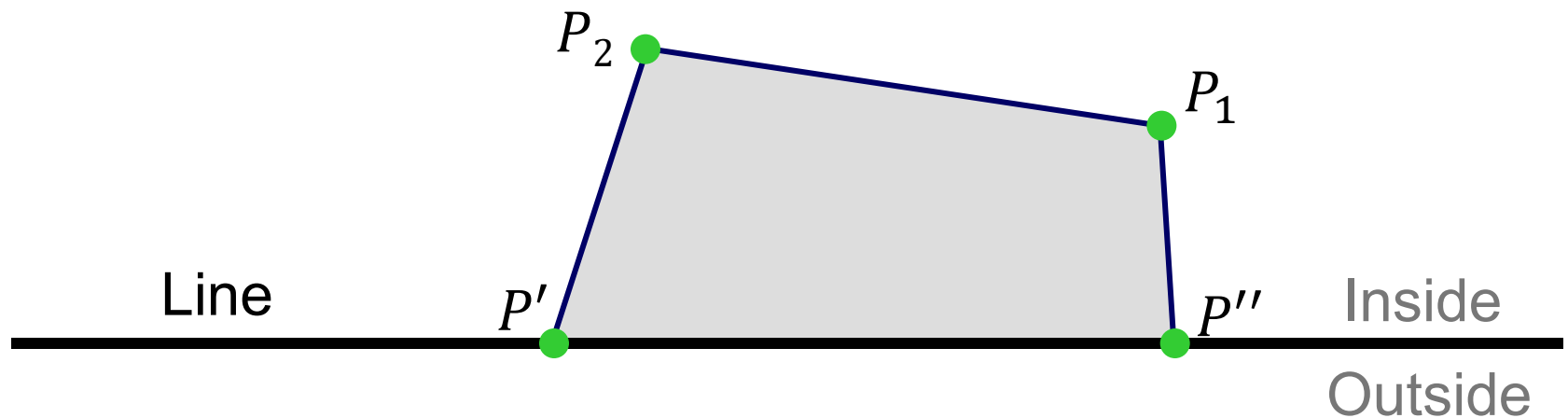
- In sequence, do interior test for each point
- Insert a new point when crossing the line.
- Remove points outside the line.





Sutherland-Hodgeman Clipping

- In sequence, do interior test for each point
- Insert a new point when crossing the line.
- Remove points outside the line.

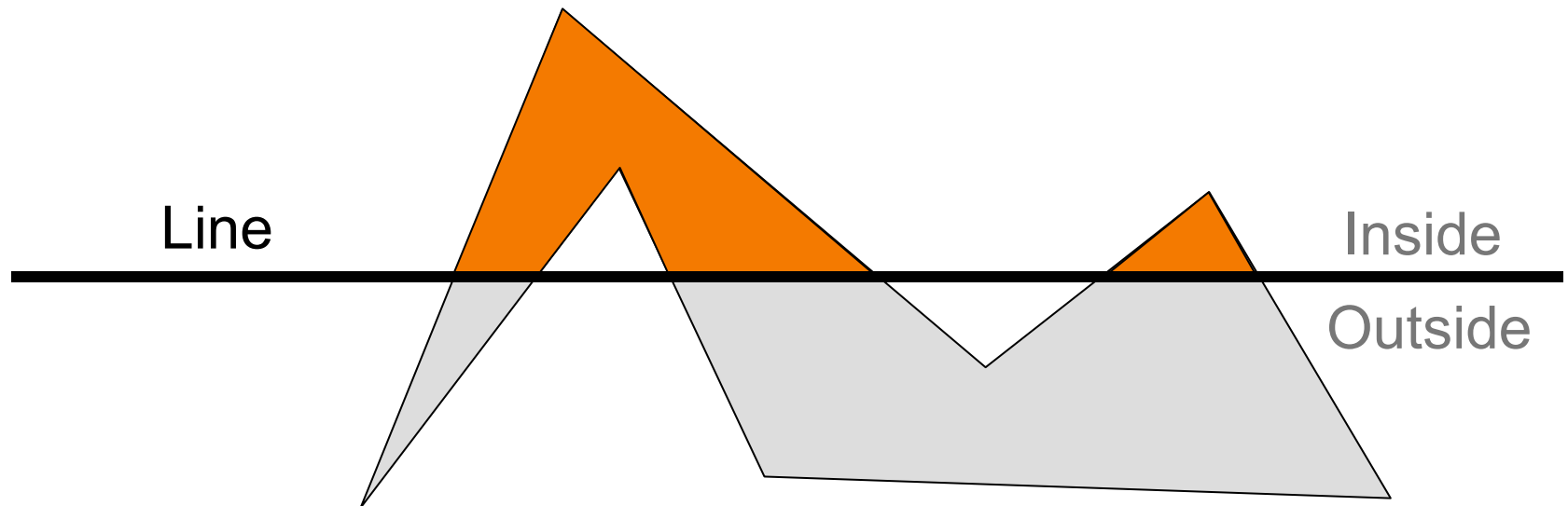


When polygons are clipped, per-vertex properties (e.g. lighting) are interpolated to the new vertices.



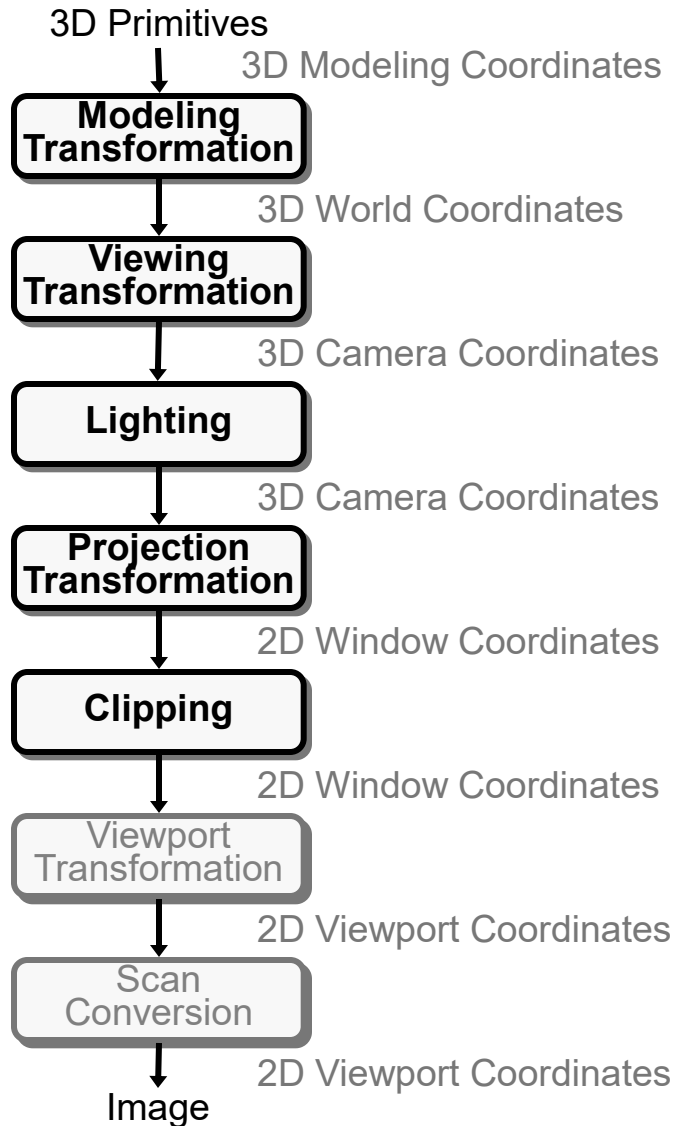
Sutherland-Hodgeman Clipping

- In sequence, do interior test for each point
- 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)



At this point we have the:

- 2D positions of the mesh vertices (with new vertices obtained through clipping)
 - Color at each vertex.
 - A list of clipped projected 2D polygons.
- Need to “fill in” the polygons.

