The 3D Rasterization Pipeline

COS 426, Spring 2021

Felix Heide

Princeton University

3D Rendering Scenarios

- Offline
  - One image generated with as much quality as possible for a particular set of rendering parameters
    - Take as much time as is needed (minutes)
    - Targets photorealistism, movies, etc.
- Interactive
  - Images generated dynamically, in fraction of a second (e.g., 1/30) as user controls rendering parameters (e.g., camera)
    - Achieve highest quality possible in given time
    - Visualization, games, etc.

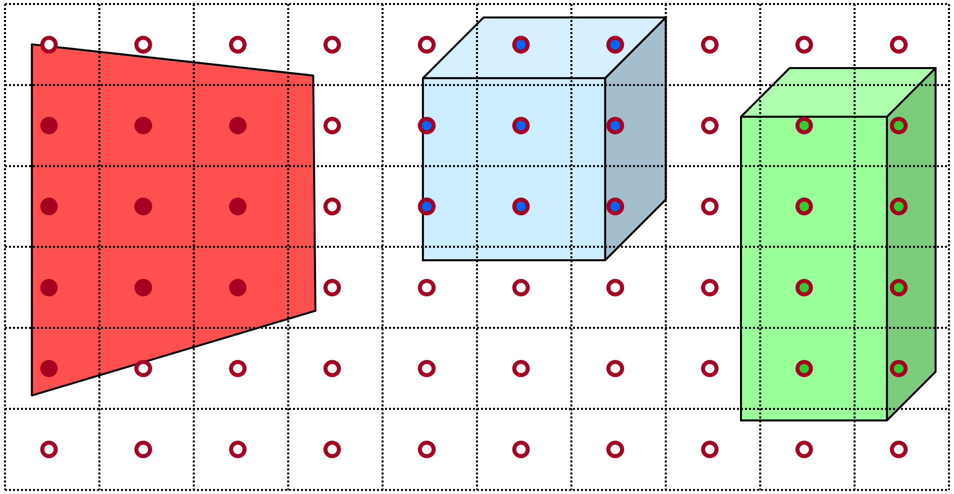
3D Polygon Rendering

- Many applications use rendering of 3D polygons with direct illumination

Valve

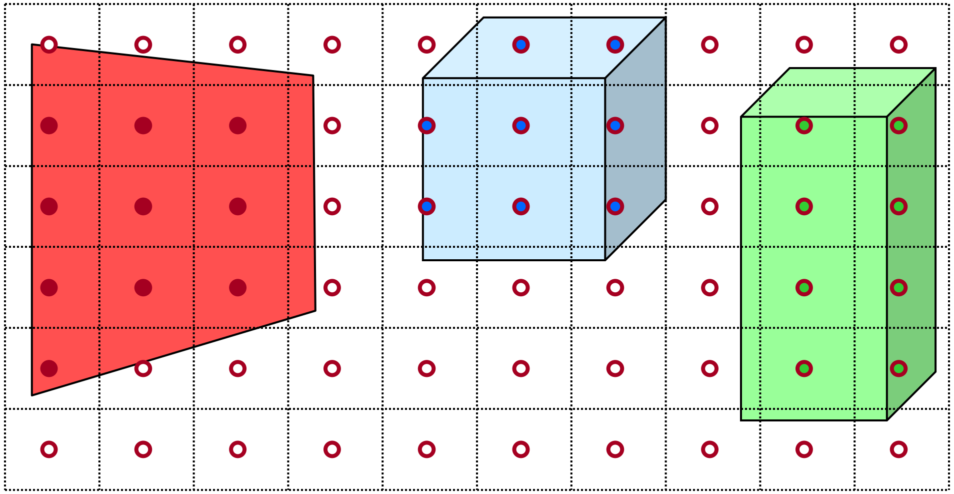
Ray Casting Revisited

- For each sample …
  - Construct ray from eye position through view plane
  - Find first surface intersected by ray through pixel
  - Compute color of sample based on illumination



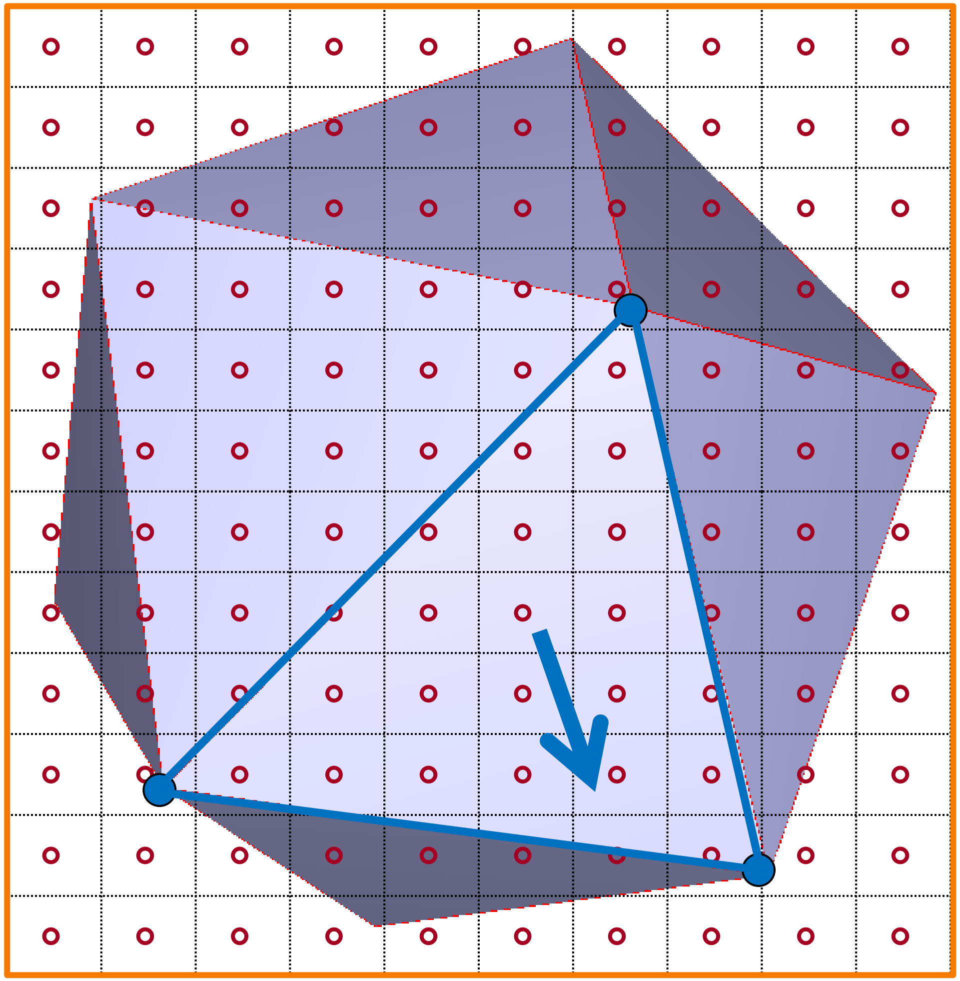
3D Polygon Rasterization

- We can render polygons faster if we take advantage of **spatial coherence**



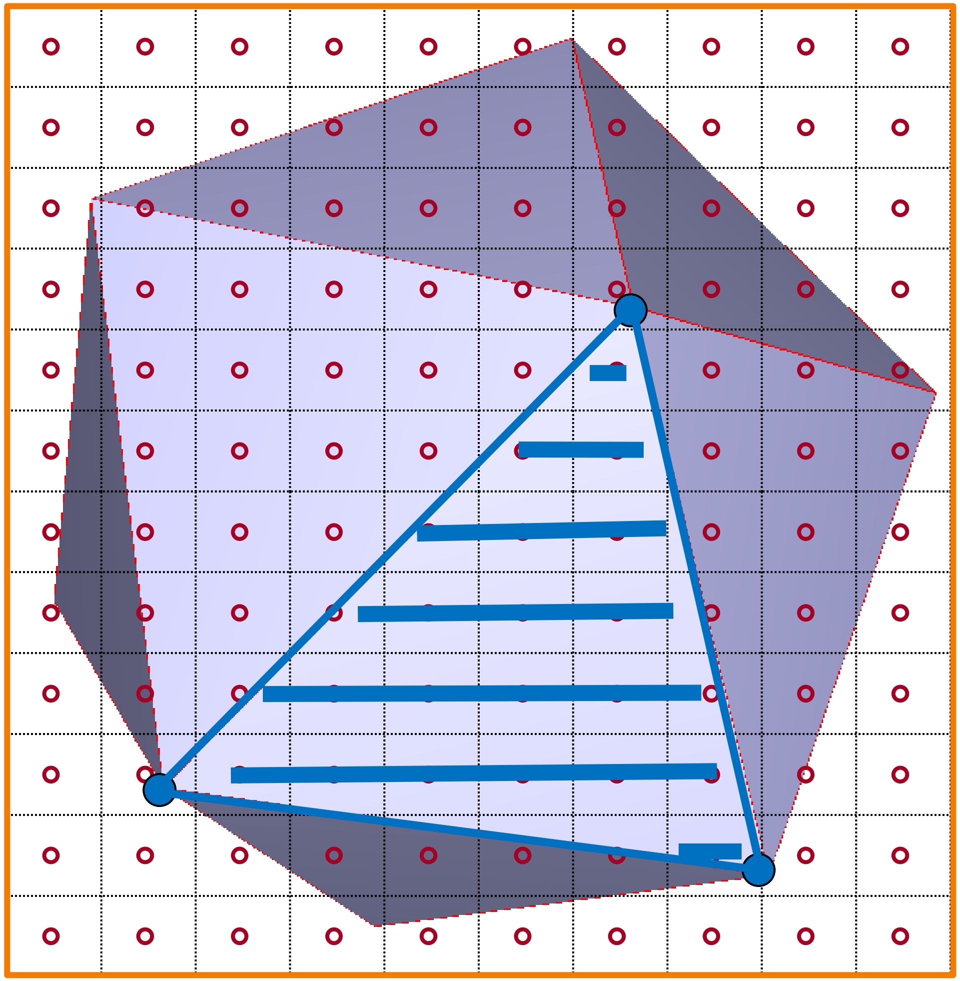
3D Polygon Rasterization

- How?

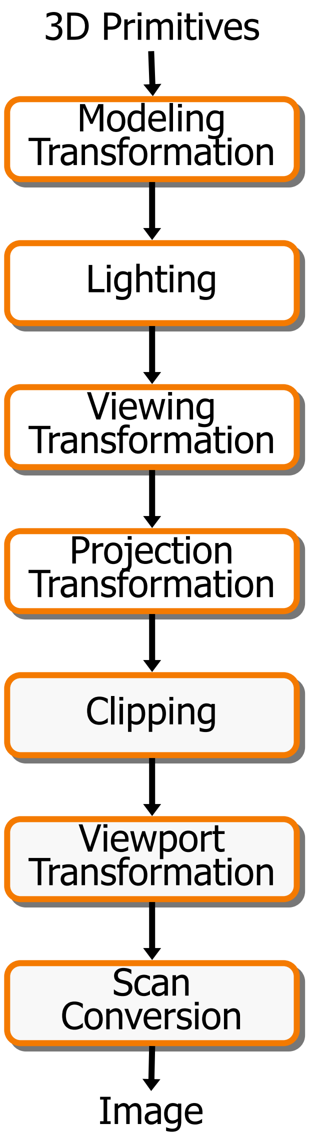


3D Polygon Rasterization

- How?

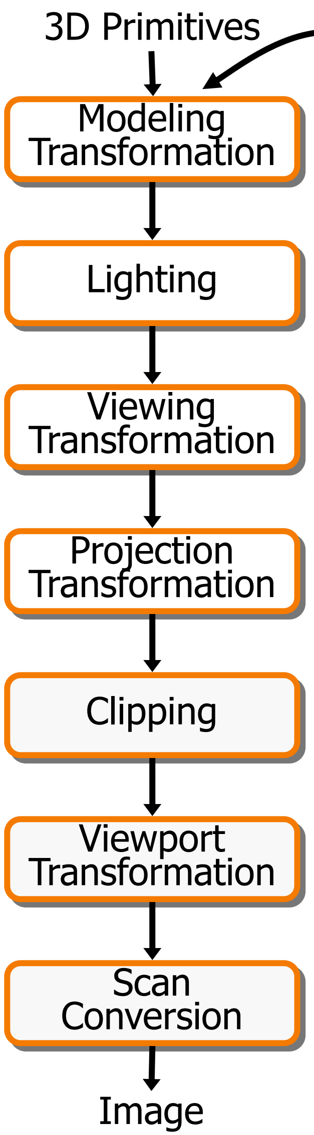


Rasterization Pipeline (for direct illumination)



This is a pipelined sequence of operations to draw 3D primitives into a 2D image

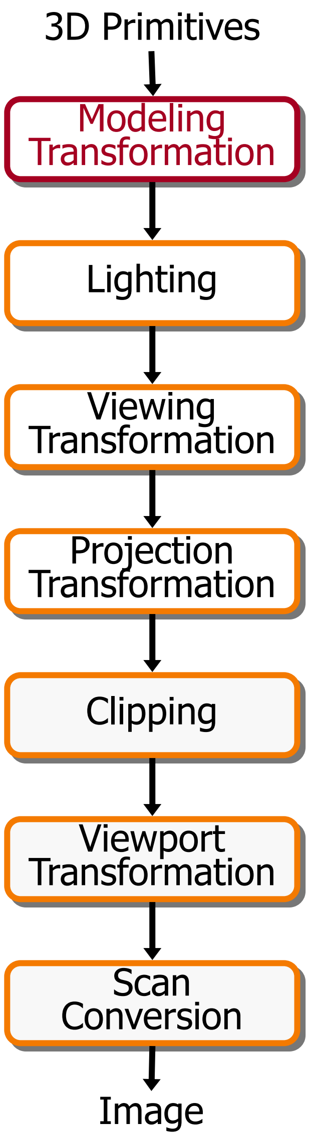
Rasterization Pipeline (for direct illumination)




```
glBegin(GL_POLYGON);
glVertex3f(0.0, 0.0, 0.0);
glVertex3f(1.0, 0.0, 0.0);
glVertex3f(0.0, 1.0, 0.0);
glEnd();
```

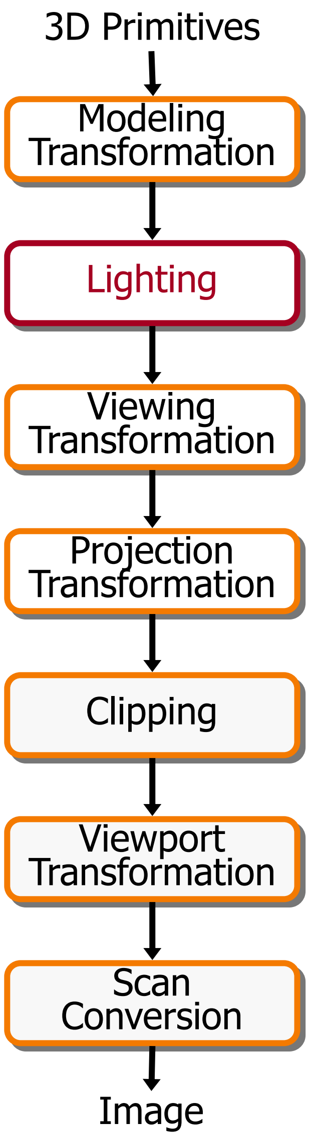
OpenGL executes steps of 3D rendering pipeline for each polygon

Rasterization Pipeline (for direct illumination)



Transform into 3D world coordinate system

Rasterization Pipeline (for direct illumination)

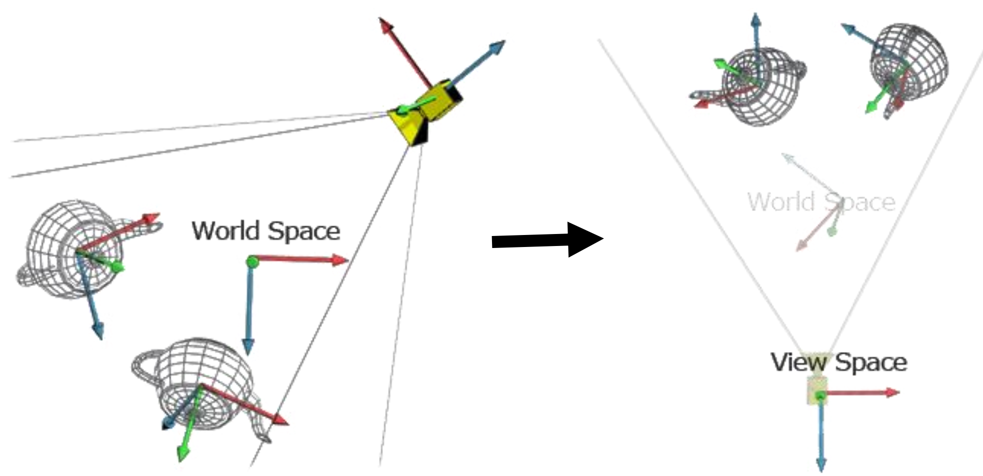


Transform into 3D world coordinate system

Illuminate according to lighting and reflectance

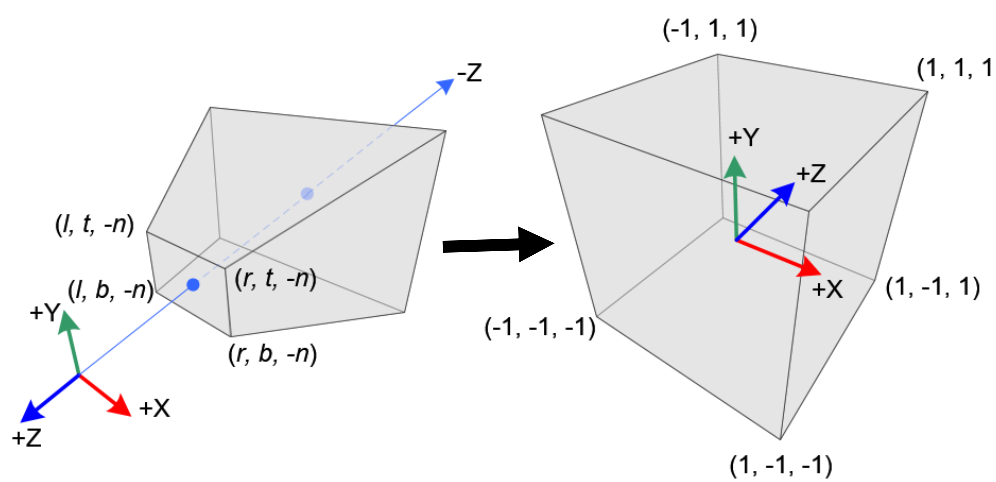
Rasterization Pipeline (for direct illumination)

3D Primitives
↓
Modeling Transformation — Transform into 3D world coordinate system
↓
Lighting — Illuminate according to lighting and reflectance
↓
Viewing Transformation — Transform into 3D camera coordinate system
↓
Projection Transformation
↓
Clipping
↓
Viewport Transformation
↓
Scan Conversion
↓
Image



Rasterization Pipeline (for direct illumination)

3D Primitives
↓
Modeling Transformation — Transform into 3D world coordinate system
↓
Lighting — Illuminate according to lighting and reflectance
↓
Viewing Transformation — Transform into 3D camera coordinate system
↓
Projection Transformation — Transform into 2D camera coordinate system
↓
Clipping
↓
Viewport Transformation
↓
Scan Conversion
↓
Image



Rasterization Pipeline (for direct illumination)

3D Primitives

↓

Modeling Transformation — Transform into 3D world coordinate system

↓

Lighting — Illuminate according to lighting and reflectance

↓

Viewing Transformation — Transform into 3D camera coordinate system

↓

Projection Transformation — Transform into 2D camera coordinate system

↓

Clipping — Clip primitives outside camera's view

↓

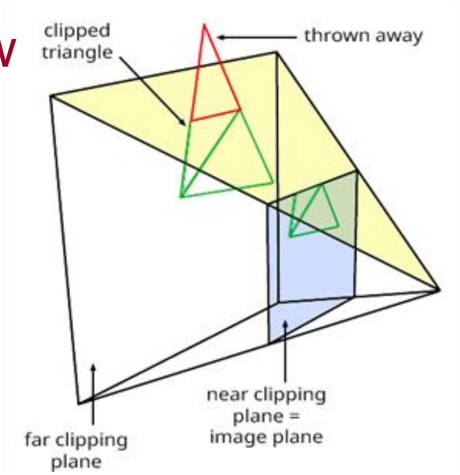
Viewport Transformation

↓

Scan Conversion

↓

Image



Rasterization Pipeline (for direct illumination)

3D Primitives

↓

Modeling Transformation — Transform into 3D world coordinate system

↓

Lighting — Illuminate according to lighting and reflectance

↓

Viewing Transformation — Transform into 3D camera coordinate system

↓

Projection Transformation — Transform into 2D camera coordinate system

↓

Clipping — Clip primitives outside camera's view … in clip space

↓

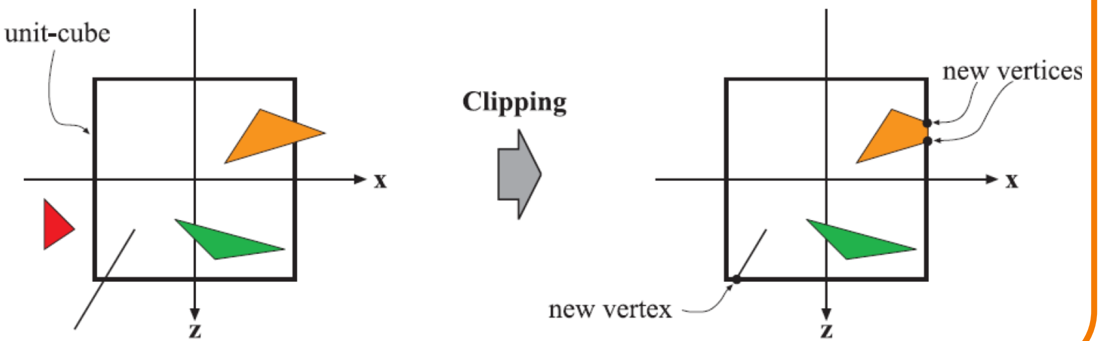
Viewport Transformation

↓

Scan Conversion

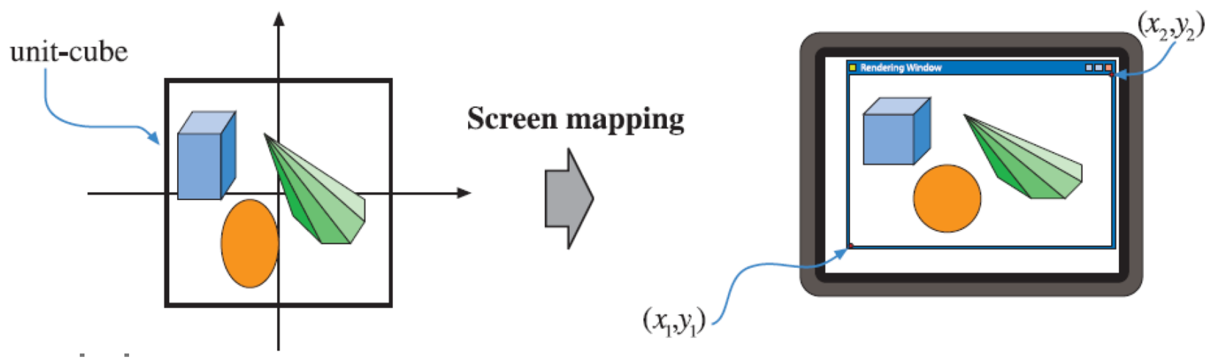
↓

Image



Rasterization Pipeline (for direct illumination)

3D Primitives
↓
Modeling Transformation — Transform into 3D world coordinate system
↓
Lighting — Illuminate according to lighting and reflectance
↓
Viewing Transformation
↓
Projection Transformation
↓
Clipping
↓
Viewport Transformation — Transform into image coordinate system
↓
Scan Conversion
↓
Image



Rasterization Pipeline (for direct illumination)

3D Primitives

↓

Modeling Transformation — Transform into 3D world coordinate system

↓

Lighting — Illuminate according to lighting and reflectance

↓

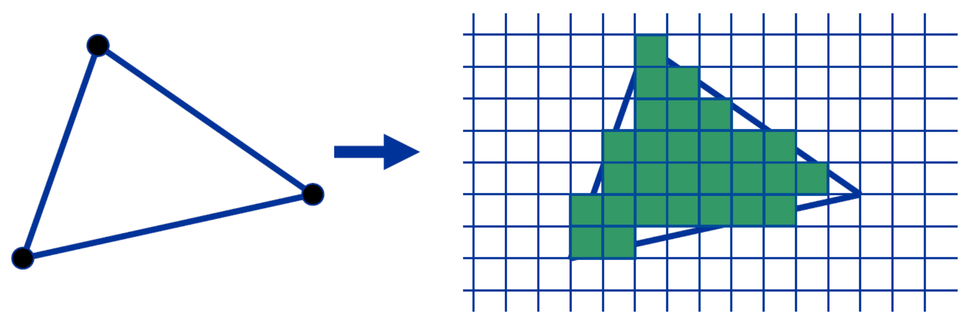
Viewing Transformation — Transform into 3D camera coordinate system

↓

Projection Transformation — Transform into 2D camera coordinate system

↓

Clipping



↓

Viewport Transformation

↓

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

↓

Image

Rasterization Pipeline (for direct illumination)

3D Primitives

↓

Modeling Transformation	Transform into 3D world coordinate system
Lighting	Illuminate according to lighting and reflectance
Viewing Transformation	Transform into 3D camera coordinate system
Projection Transformation	Transform into 2D camera coordinate system
Clipping	Clip primitives outside camera's view
Viewport Transformation	Transform into image coordinate system
Scan Conversion	Draw pixels (includes texturing, hidden surface, ...)

↓

Image

Transformations

p(x,y,z)

↓ 3D Object Coordinates

Modeling Transformation

↓ 3D World Coordinates

Viewing Transformation

↓ 3D Camera Coordinates

Projection Transformation

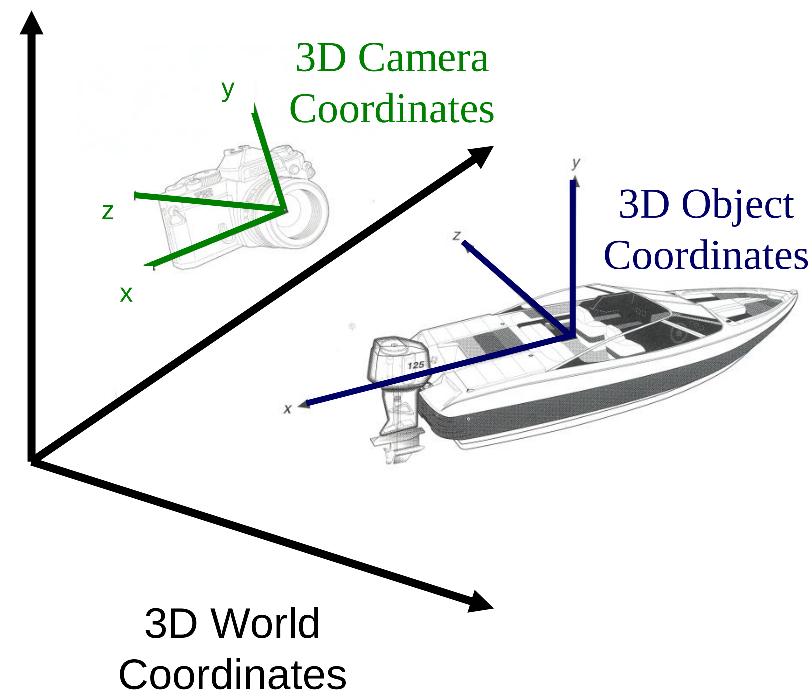
↓ 2D Screen Coordinates

Viewport Transformation

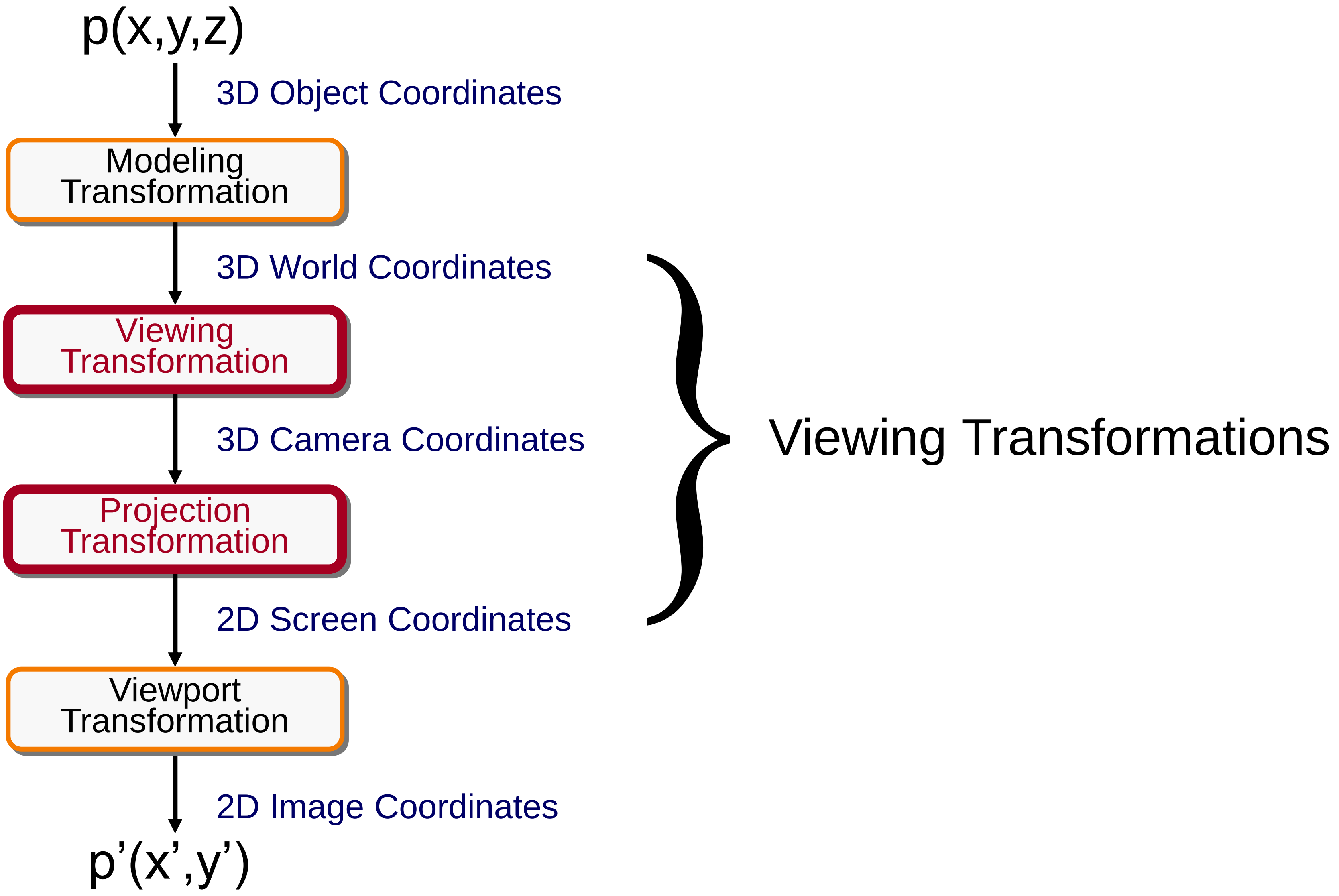
↓ 2D Image Coordinates

p'(x',y')

> Transformations map points from one coordinate system to another

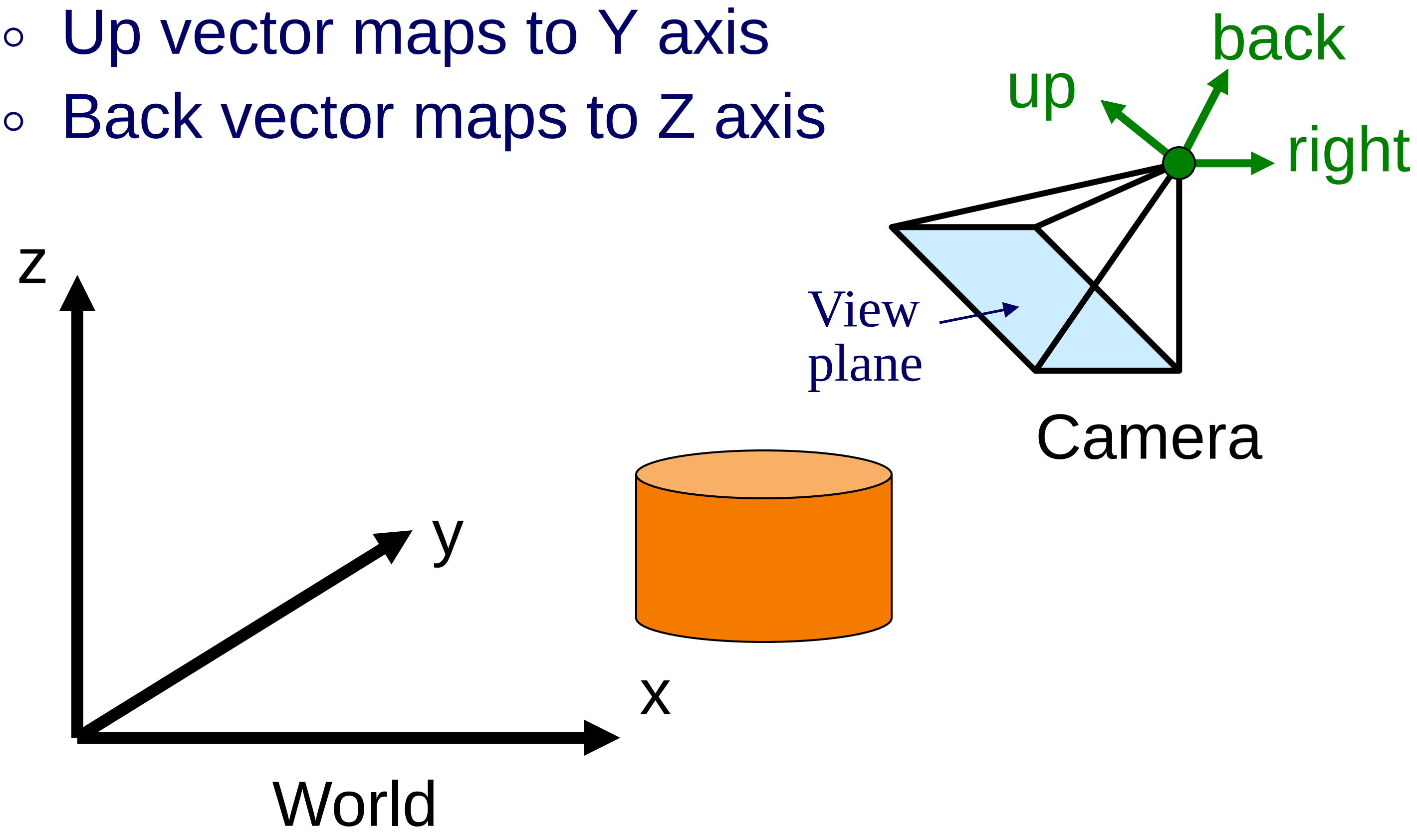


Viewing Transformations



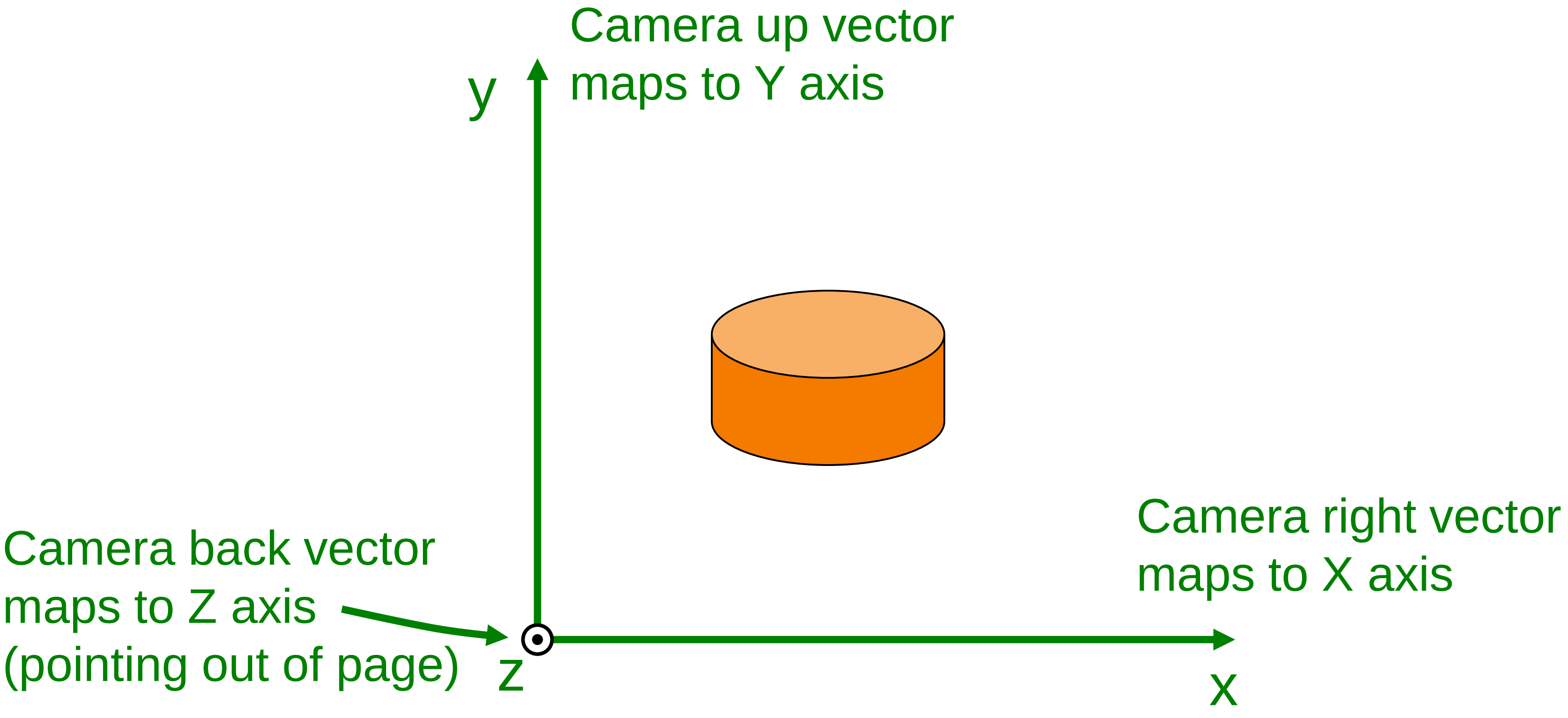
Review: Viewing Transformation

- Mapping from world to camera coordinates
  - Eye position maps to origin
  - Right vector maps to X axis
  - Up vector maps to Y axis
  - Back vector maps to Z axis



Review: Camera Coordinates

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



Finding the Viewing Transformation

- Trick: map from camera coordinates to world
  - Origin maps to eye position
  - Z axis maps to Back vector
  - Y axis maps to Up vector
  - X axis maps to Right vector

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} R_x & U_x & B_x & E_x \\ R_y & U_y & B_y & E_y \\ R_z & U_z & B_z & E_z \\ R_w & U_w & B_w & E_w \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

- This matrix is T^{-1} so we invert it to get T … easy!

Finding the viewing transformation

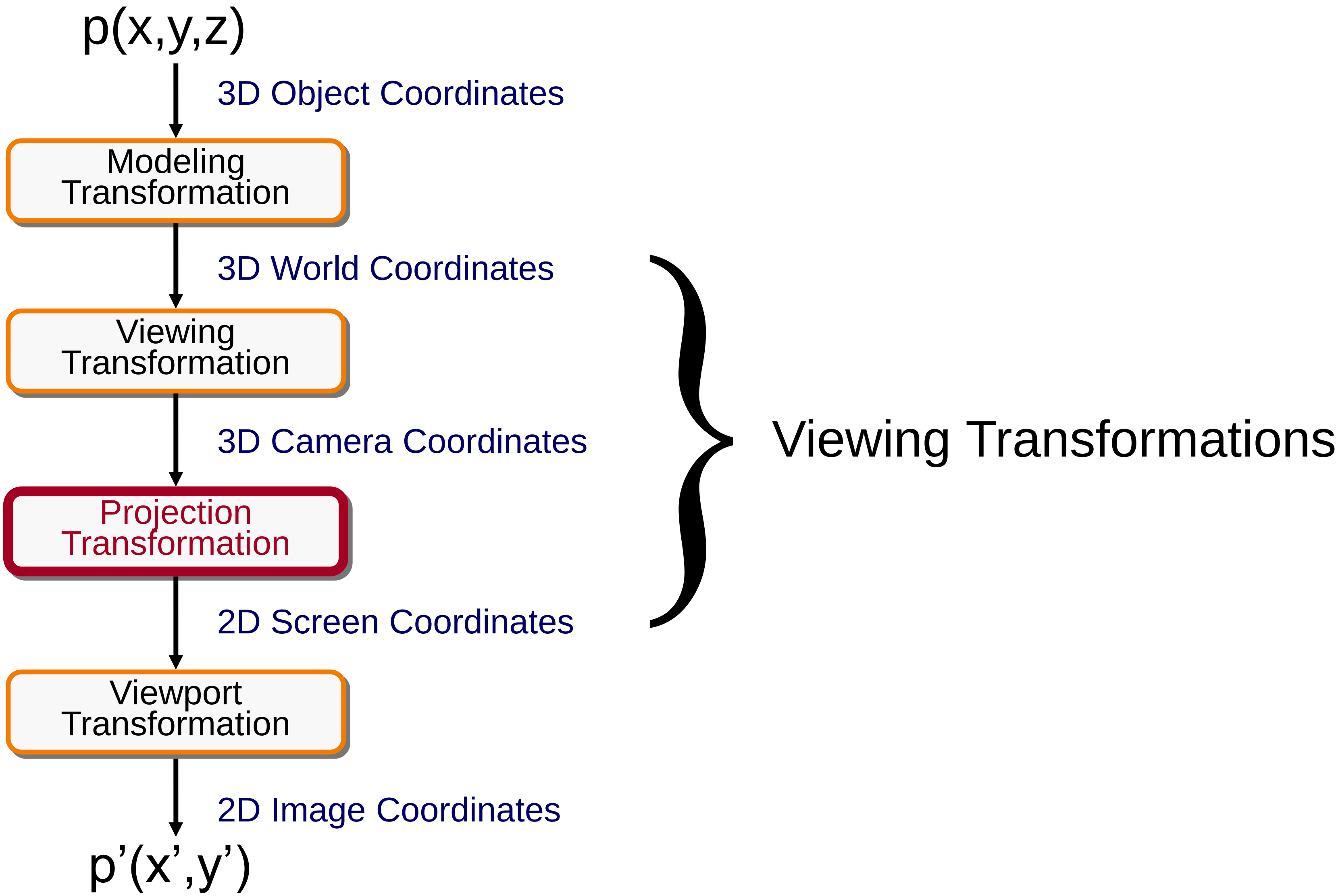
- We have the camera (in world coordinates)
- We want T taking objects from world to camera

$$p^C = T\ p^W$$

- Trick: find T^{-1} taking objects in camera to world

$$p^W = T^{-1} p^C$$

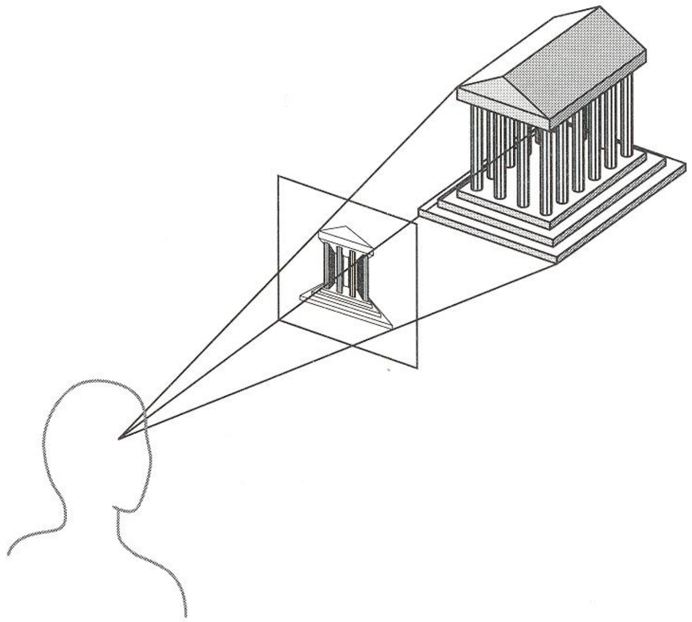
Viewing Transformations



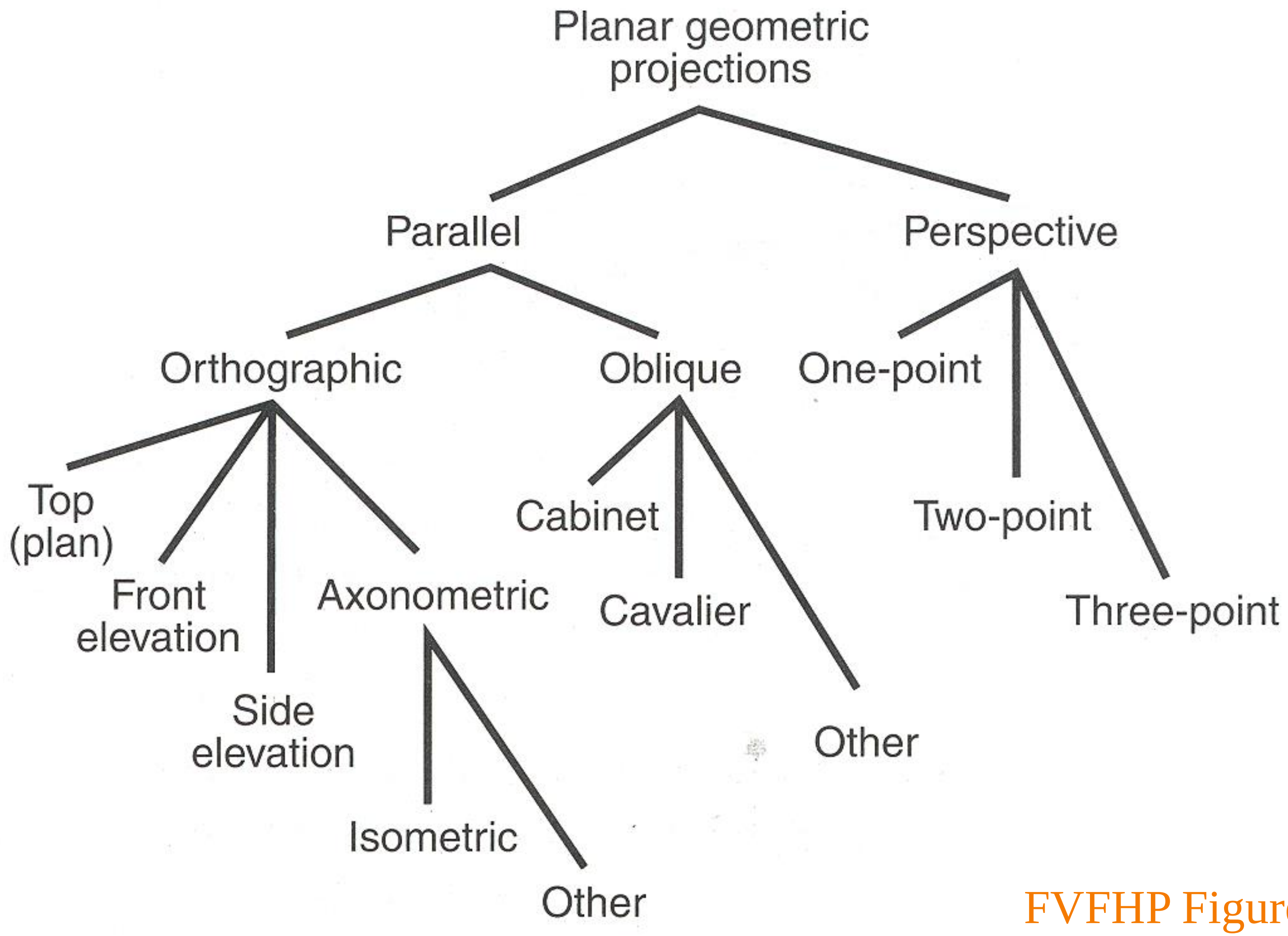
Projection

- General definition:
  - Transform points in *n*-space to *m*-space (*m*<*n*)

- In computer graphics:
  - Map 3D camera coordinates to 2D screen coordinates

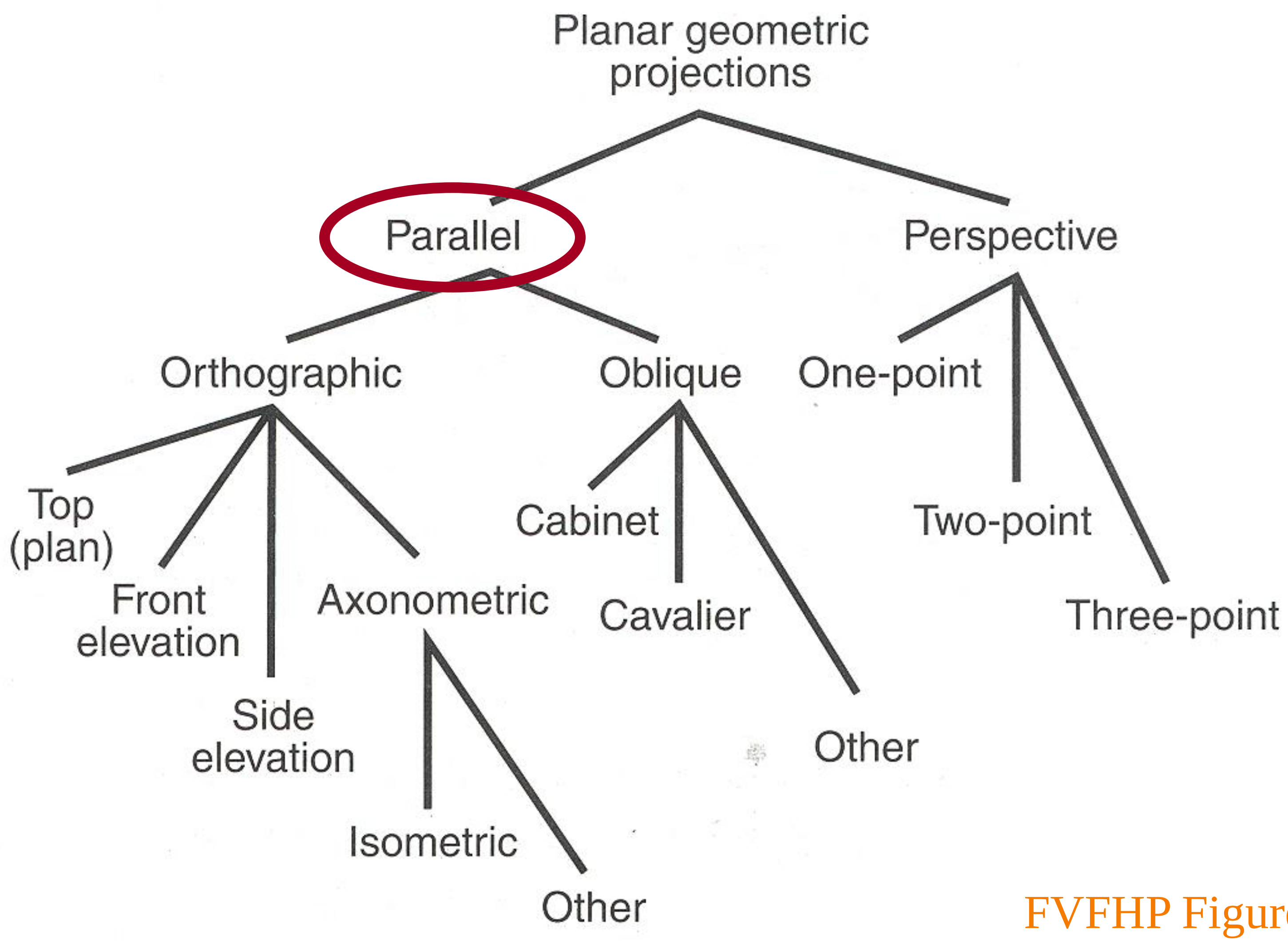


Taxonomy of Projections



FVFHP Figure 6.10

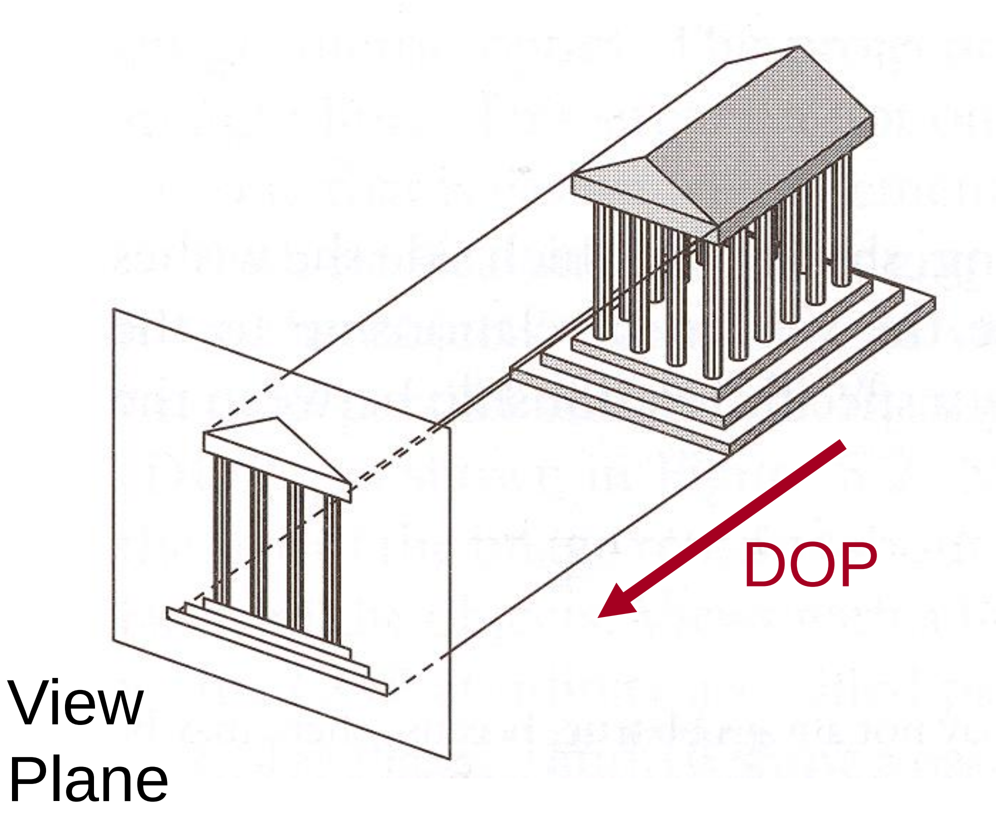
Taxonomy of Projections



FVFHP Figure 6.10

Parallel Projection

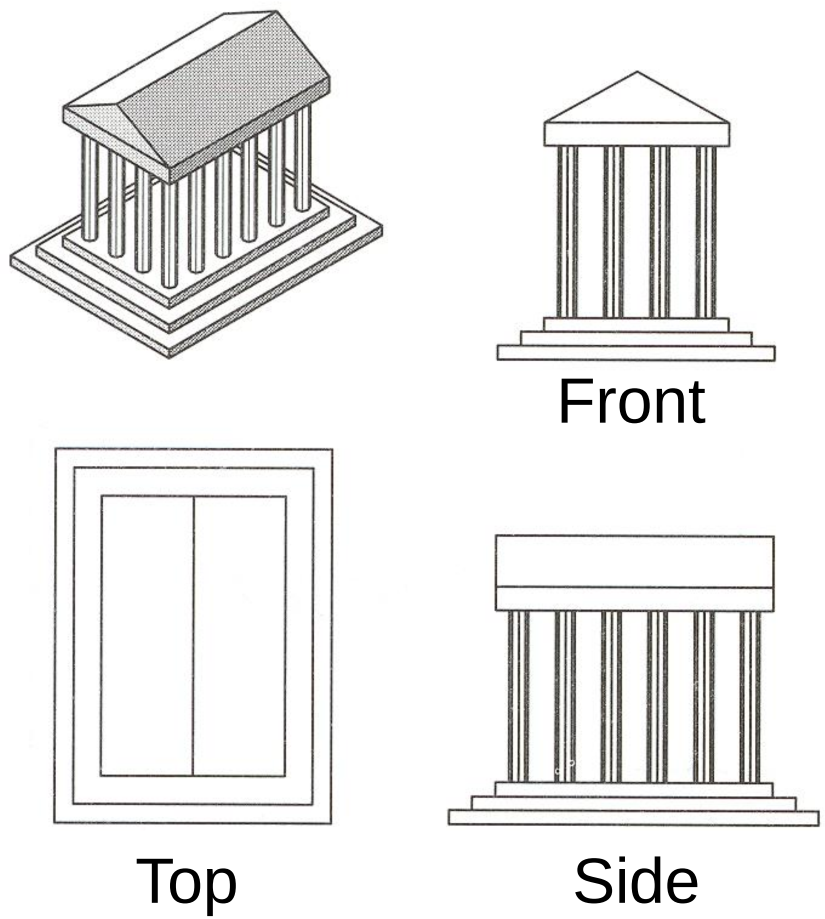
- Center of projection is at infinity
  - Direction of projection (DOP) same for all points



Angel Figure 5.4

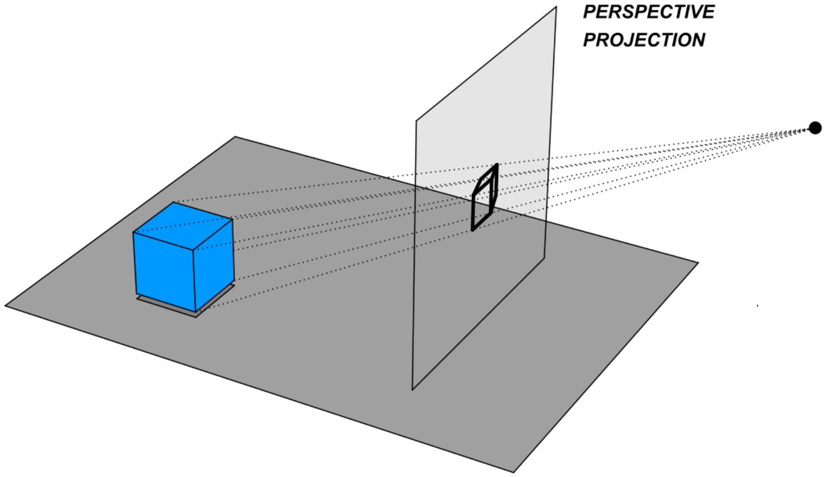
Orthographic Projections

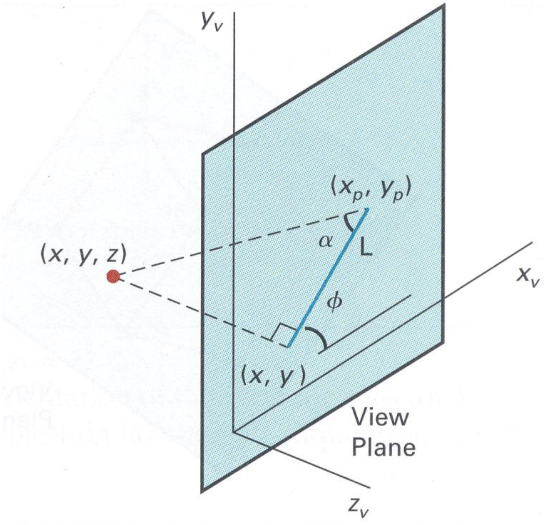
- DOP perpendicular to view plane

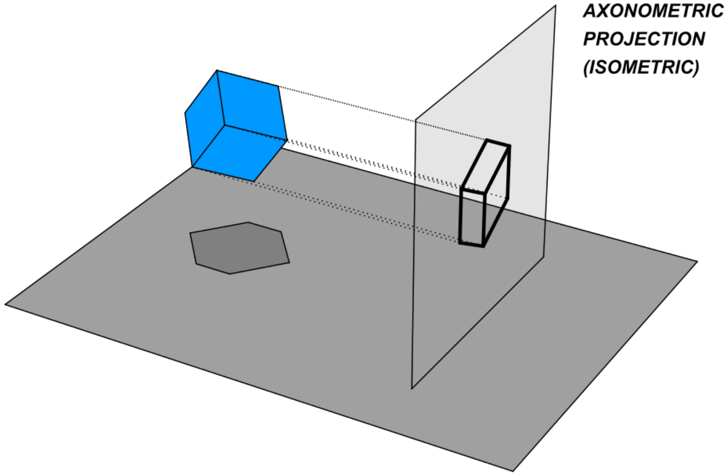


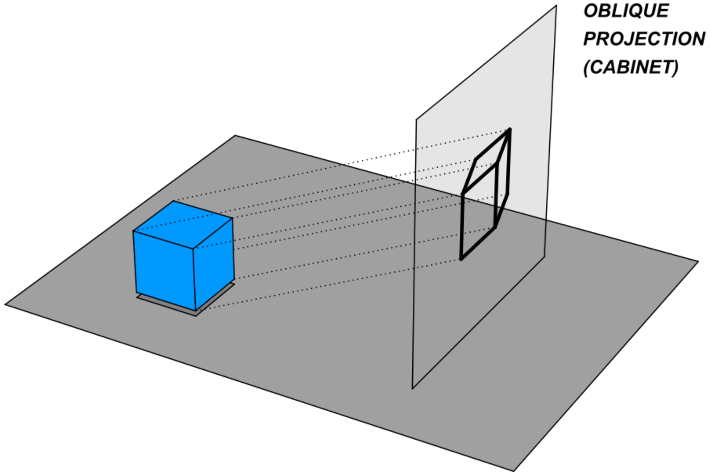
Angel Figure 5.5

Parallel Projection Matrix



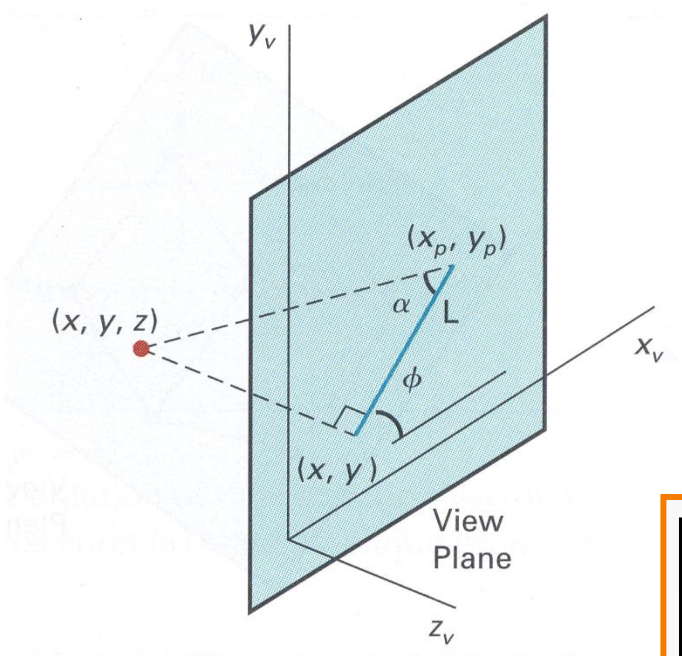






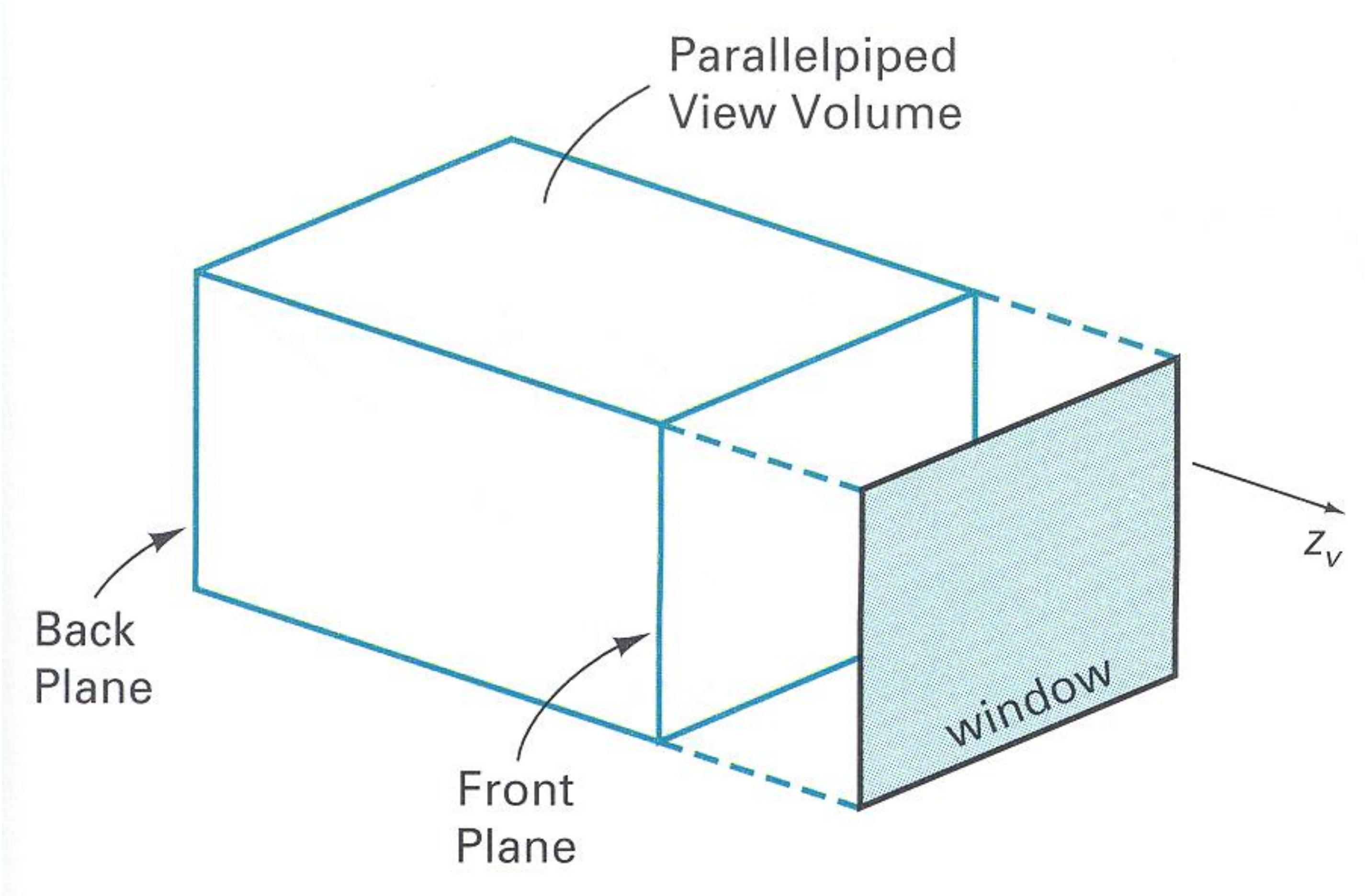
Parallel Projection Matrix

- General parallel projection transformation:



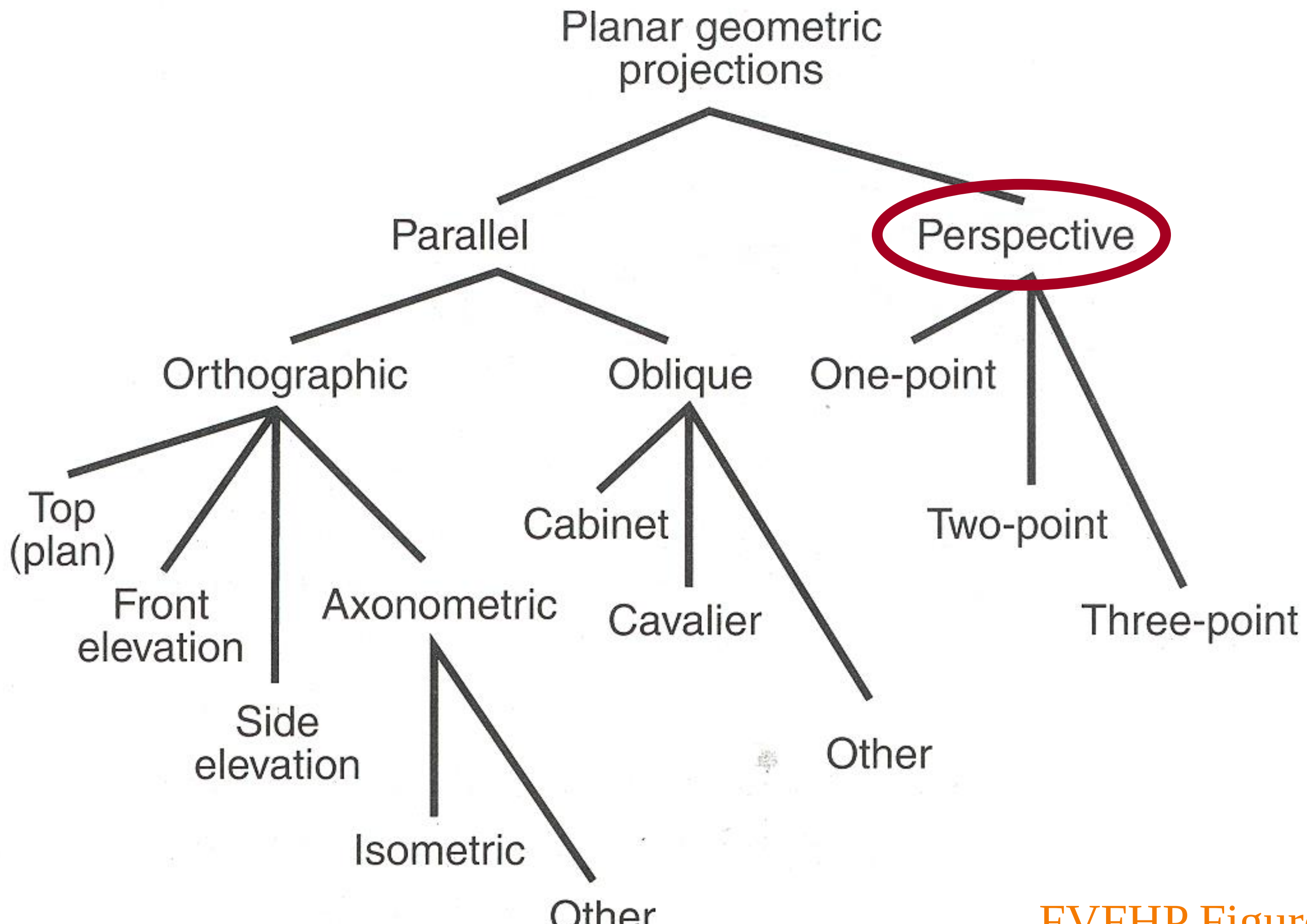
$$\begin{bmatrix} x_s \\ y_s \\ z_s \\ w_s \end{bmatrix} = \begin{bmatrix} 1 & 0 & L\cos\phi & 0 \\ 0 & 1 & L\sin\phi & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix}$$

Parallel Projection View Volume



H&B Figure 12.30

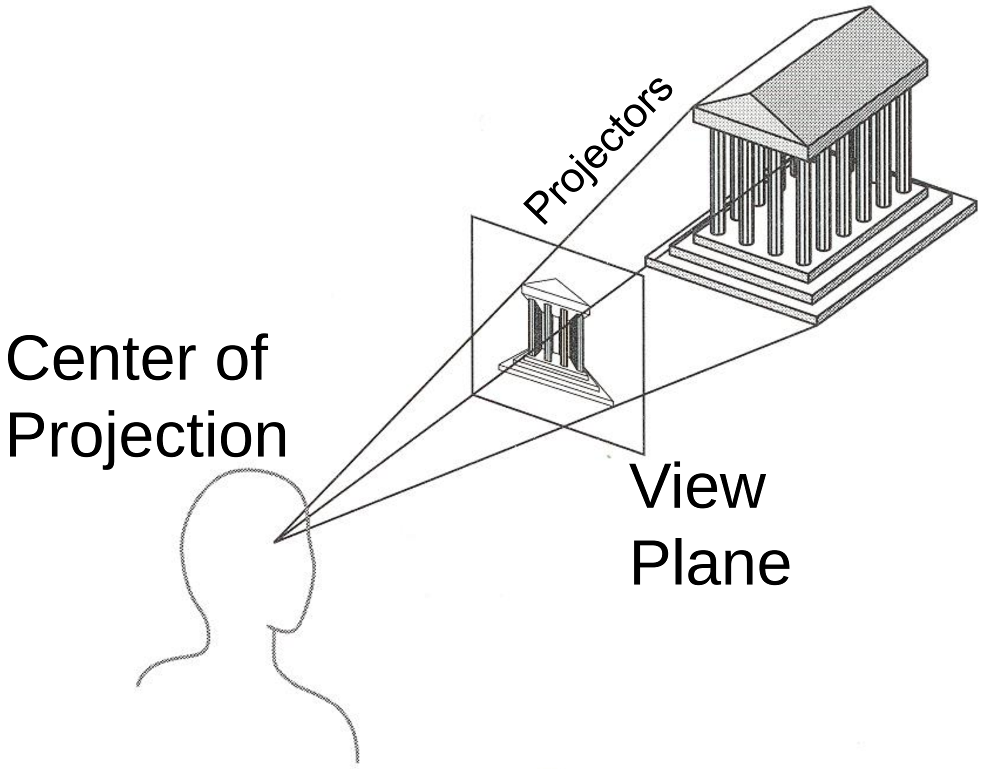
Taxonomy of Projections



FVFHP Figure 6.10

Return to Perspective Projection

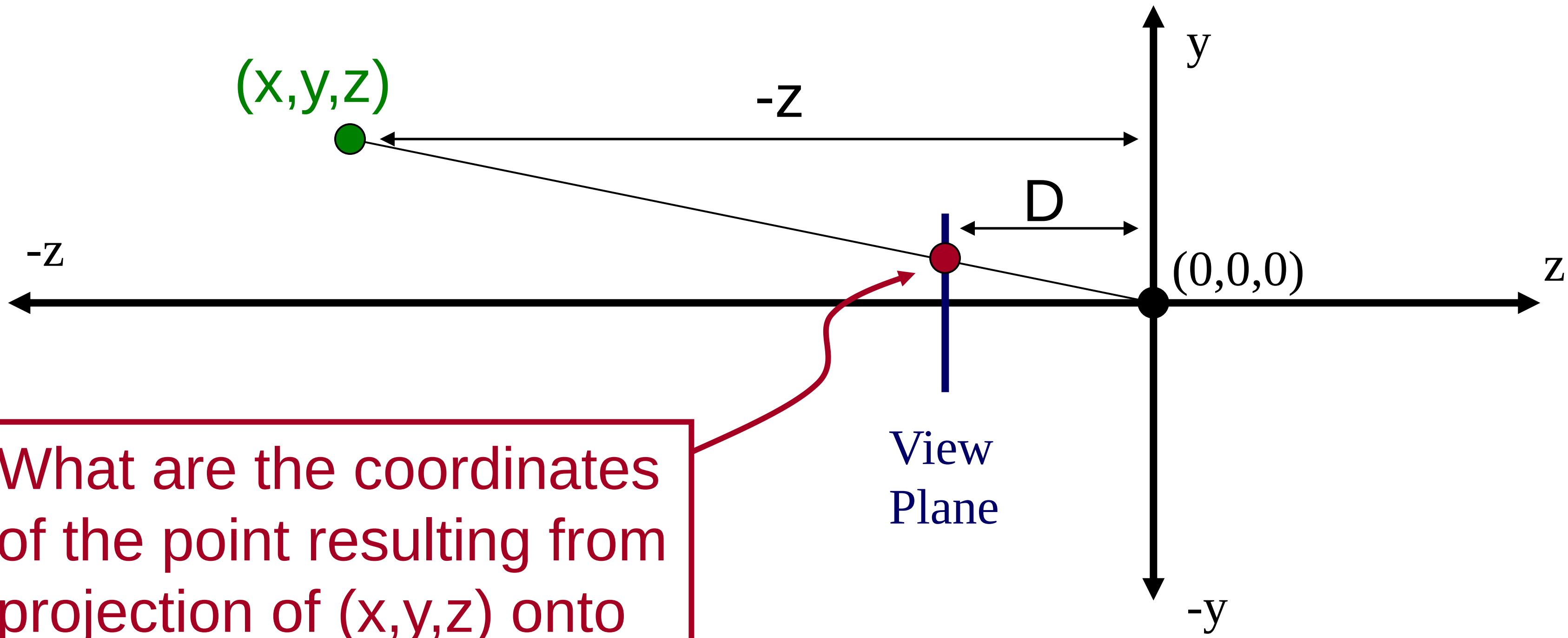
- Map points onto “view plane” along “projectors” emanating from “center of projection” (COP)



Angel Figure 5.9

Perspective Projection

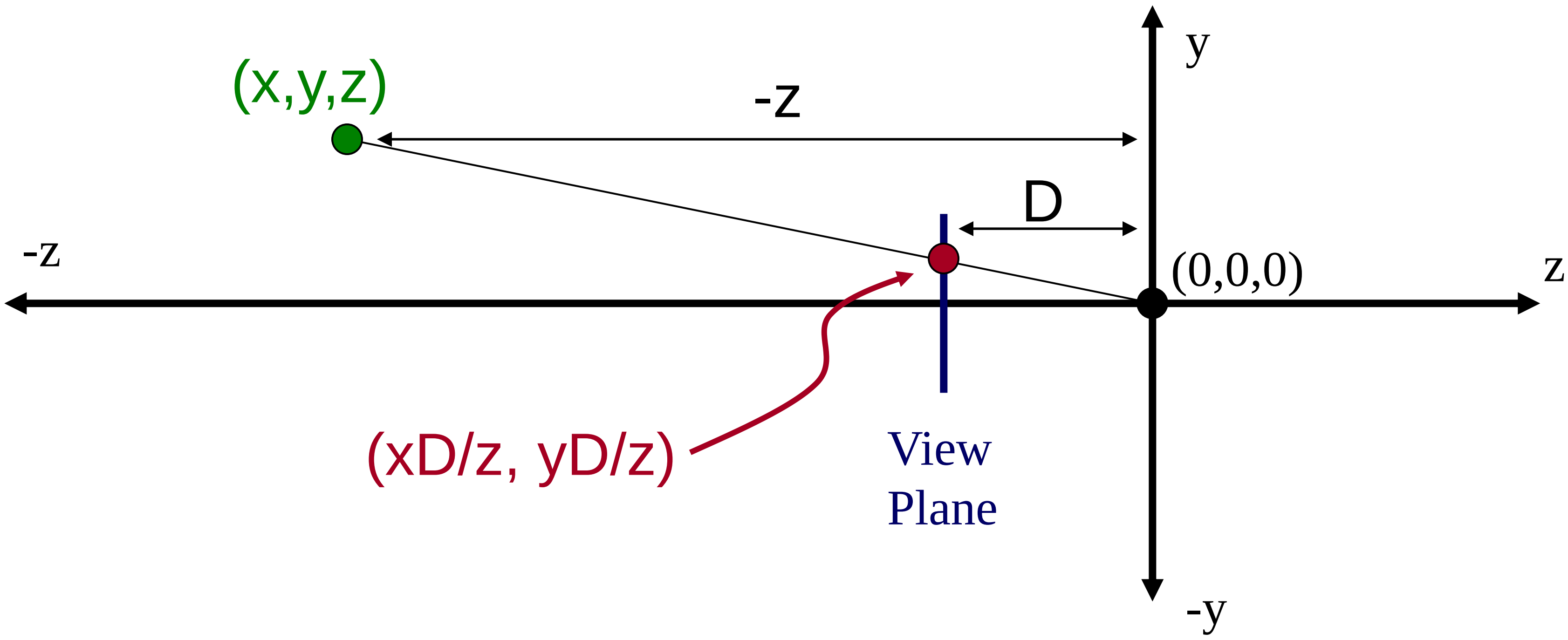
- Compute 2D coordinates from 3D coordinates with similar triangles



What are the coordinates of the point resulting from projection of (x,y,z) onto the view plane?

Perspective Projection

- Compute 2D coordinates from 3D coordinates with similar triangles



Perspective Projection Matrix

- 4x4 matrix representation?

$$x_s = x_c D / z_c$$
$$y_s = y_c D / z_c$$
$$z_s = D$$
$$w_s = 1$$

$$\begin{bmatrix} x_s \\ y_s \\ z_s \\ w_s \end{bmatrix} = \begin{bmatrix} ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix}$$

Perspective Projection Matrix

- 4x4 matrix representation?

$$x_s = x_c D / z_c \qquad x_s = x' / w' \qquad x' = x_c$$
$$y_s = y_c D / z_c \qquad y_s = y' / w' \qquad y' = y_c$$
$$z_s = D \qquad z_s = z' / w' \qquad z' = z_c$$
$$w_s = 1 \qquad w' = z_c / D$$

$$\begin{bmatrix} x_s \\ y_s \\ z_s \\ w_s \end{bmatrix} = \begin{bmatrix} ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix}$$

Perspective Projection Matrix

- 4x4 matrix representation?

$$
\begin{aligned}
x_s &= x_c D / z_c \\
y_s &= y_c D / z_c \\
z_s &= D \\
w_s &= 1
\end{aligned}
\qquad
\begin{aligned}
x_s &= x' / w' \\
y_s &= y' / w' \\
z_s &= z' / w'
\end{aligned}
\qquad
\begin{aligned}
x' &= x_c \\
y' &= y_c \\
z' &= z_c \\
w' &= z_c / D
\end{aligned}
$$

$$
\begin{bmatrix} x_s \\ y_s \\ z_s \\ w_s \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/D & 0 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix}
$$

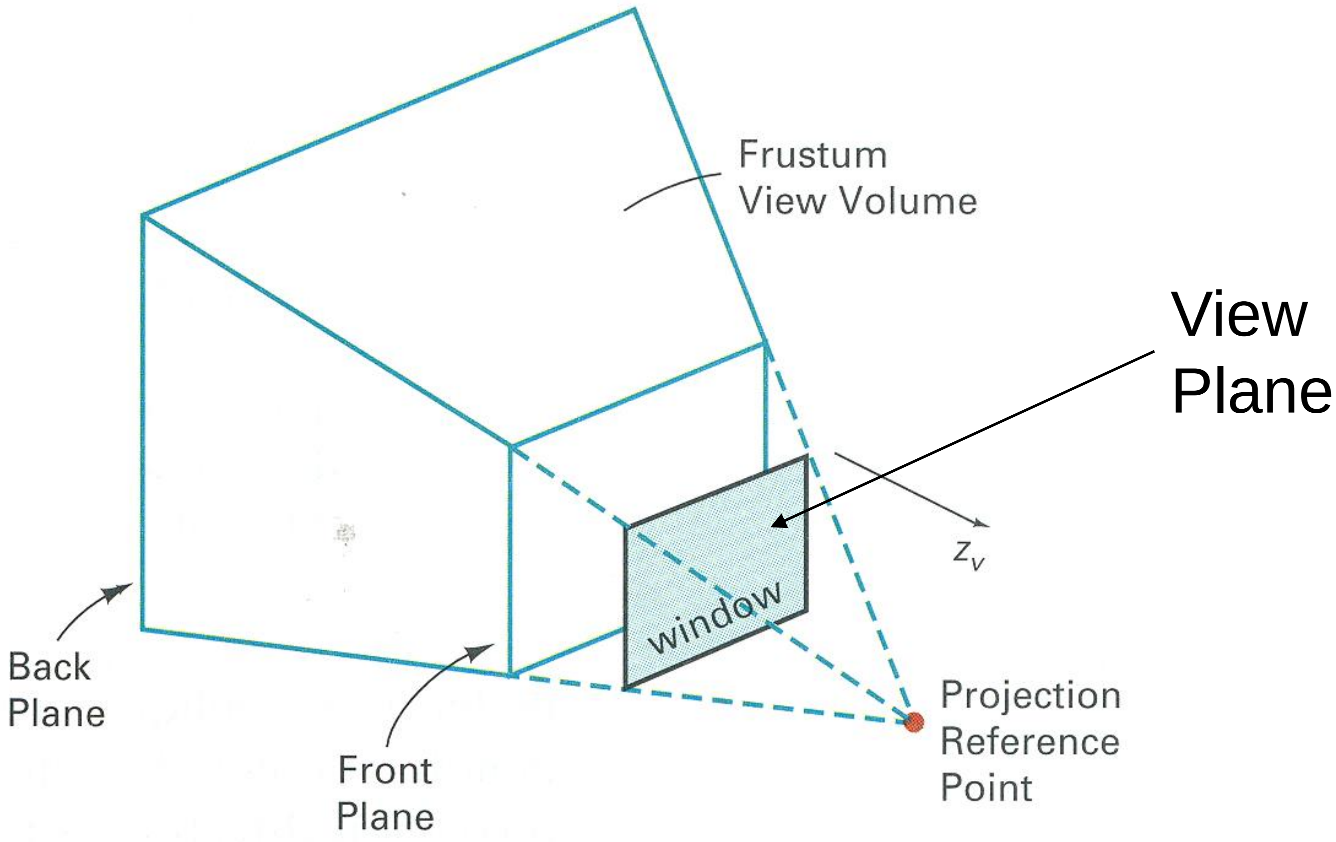
Perspective Projection Matrix

- In practice, want to compute a value related to depth to include in *z*-buffer

$$\begin{aligned} x_s &= x_c D / z_c \\ y_s &= y_c D / z_c \\ z_s &= -D / z_c \\ w_s &= 1 \end{aligned} \qquad \begin{aligned} x_s &= x' / w' \\ y_s &= y' / w' \\ z_s &= z' / w' \end{aligned} \qquad \begin{aligned} x' &= x_c \\ y' &= y_c \\ z' &= -1 \\ w' &= z_c / D \end{aligned}$$

$$\begin{bmatrix} x_s \\ y_s \\ z_s \\ w_s \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & 1/D & 0 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix}$$

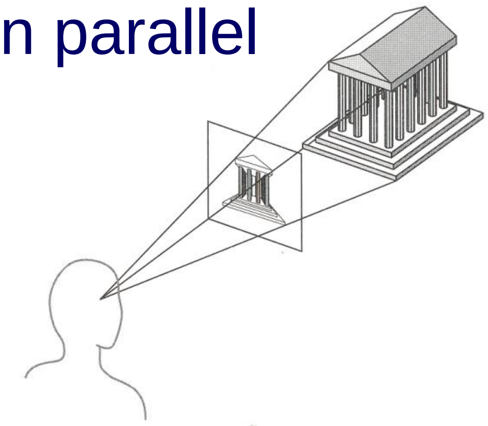
Perspective Projection View Volume



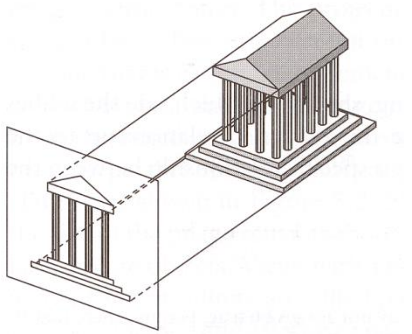
H&B Figure 12.30

Perspective vs. Parallel

- Perspective projection
  - \+ Size varies inversely with distance - looks realistic
  - – Distance and angles are not (in general) preserved
  - – Parallel lines do not (in general) remain parallel



- Parallel projection
  - \+ Good for exact measurements
  - \+ Parallel lines remain parallel
  - – Angles are not (in general) preserved
  - – Less realistic looking



Transformations

p(x,y,z)

↓ 3D Object Coordinates

Modeling Transformation

↓ 3D World Coordinates

Viewing Transformation

↓ 3D Camera Coordinates

Projection Transformation

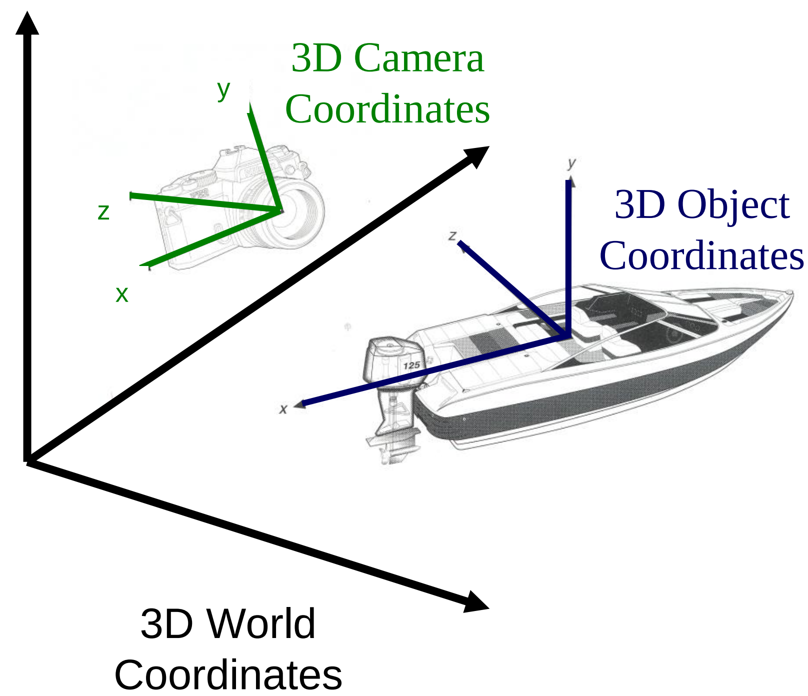
↓ 2D Screen Coordinates

Viewport Transformation

↓ 2D Image Coordinates

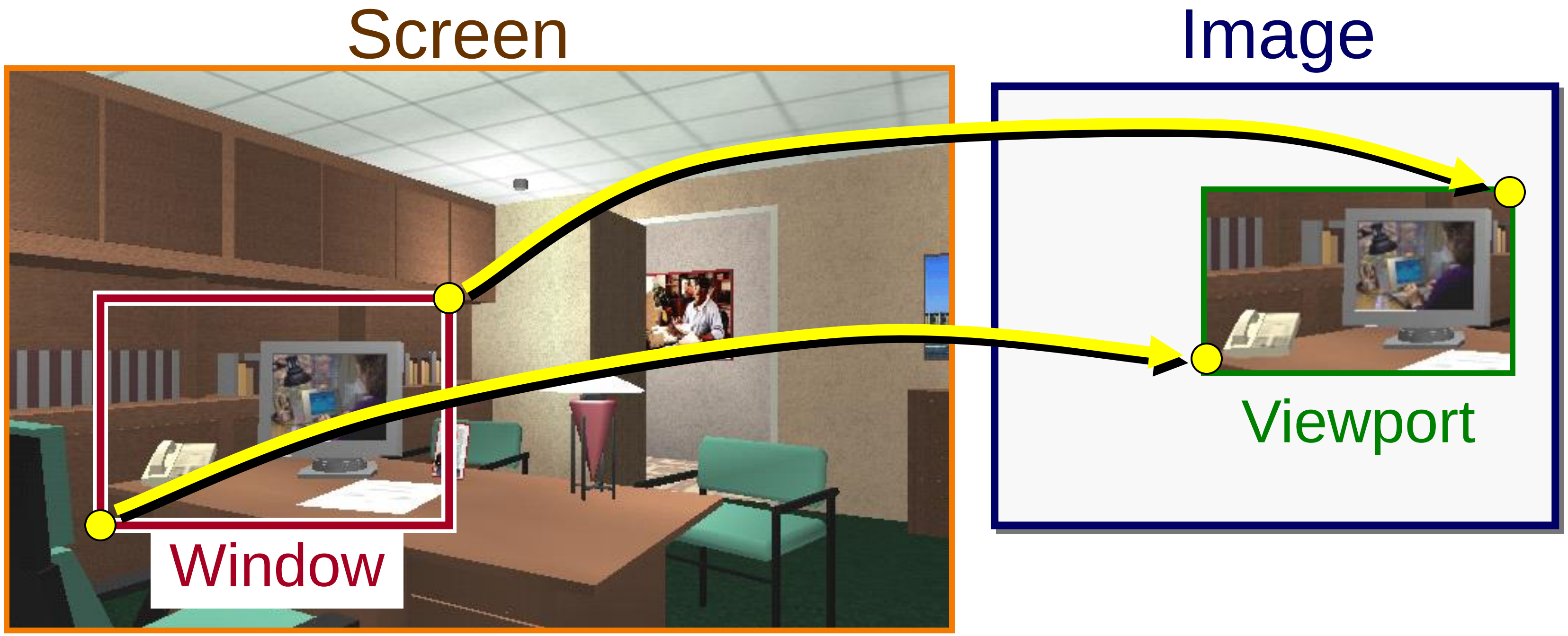
p'(x',y')

> Transformations map points from one coordinate system to another



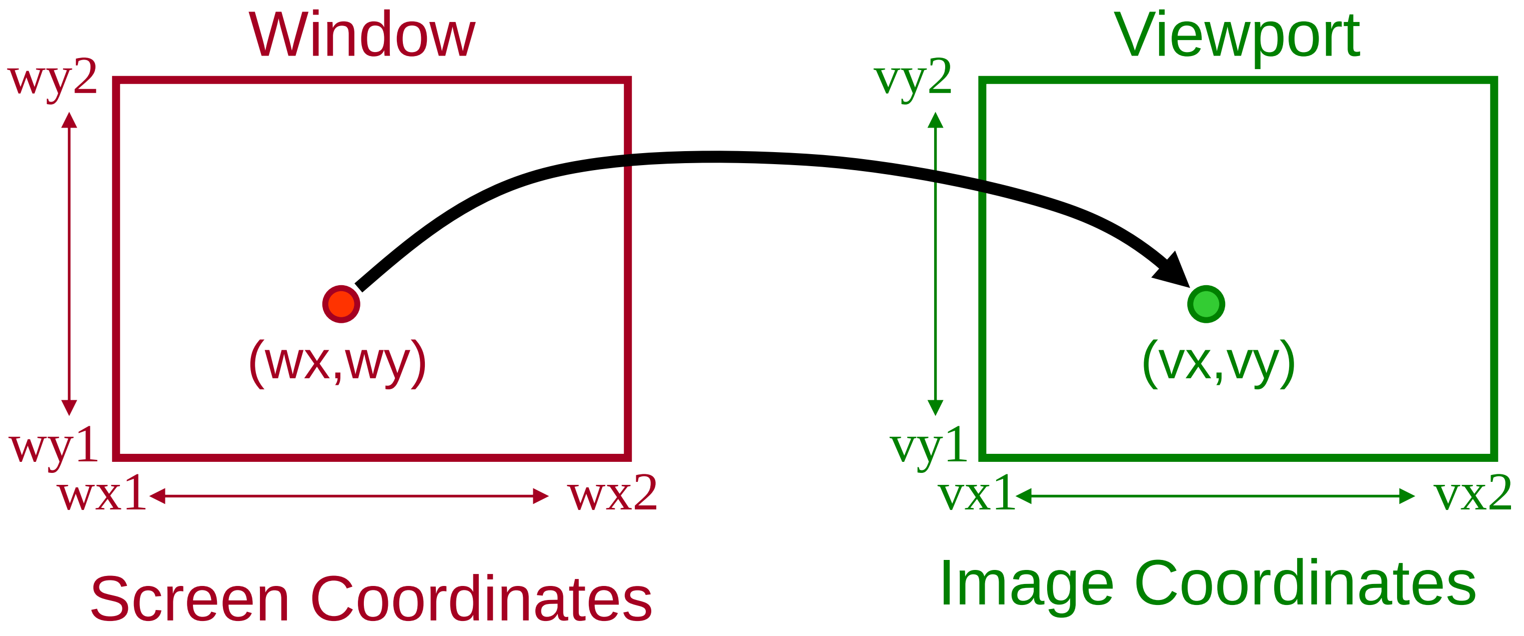
Viewport Transformation

- Transform 2D geometric primitives from screen coordinate system (normalized device coordinates) to image coordinate system (pixels)



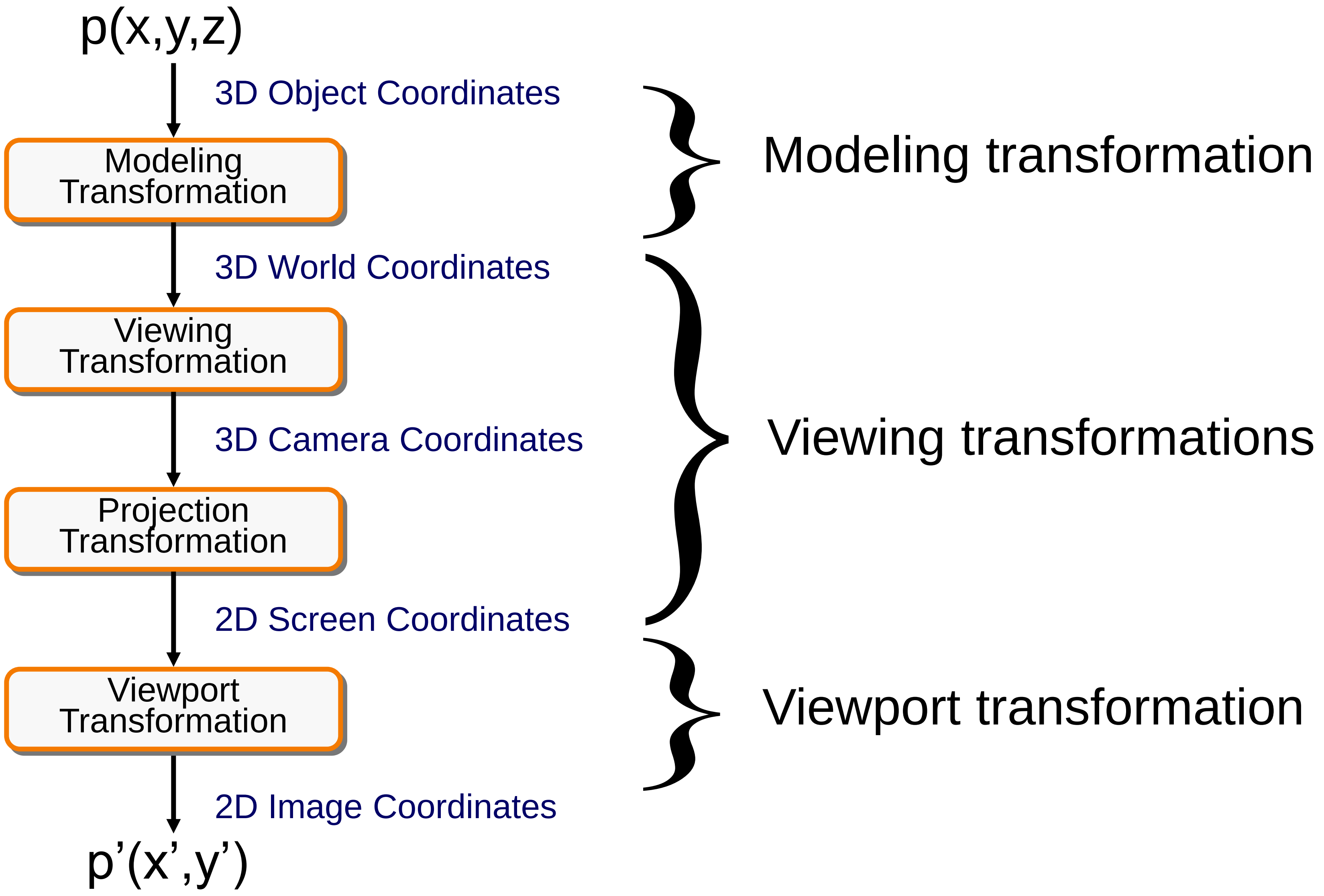
Viewport Transformation

- Window-to-viewport mapping



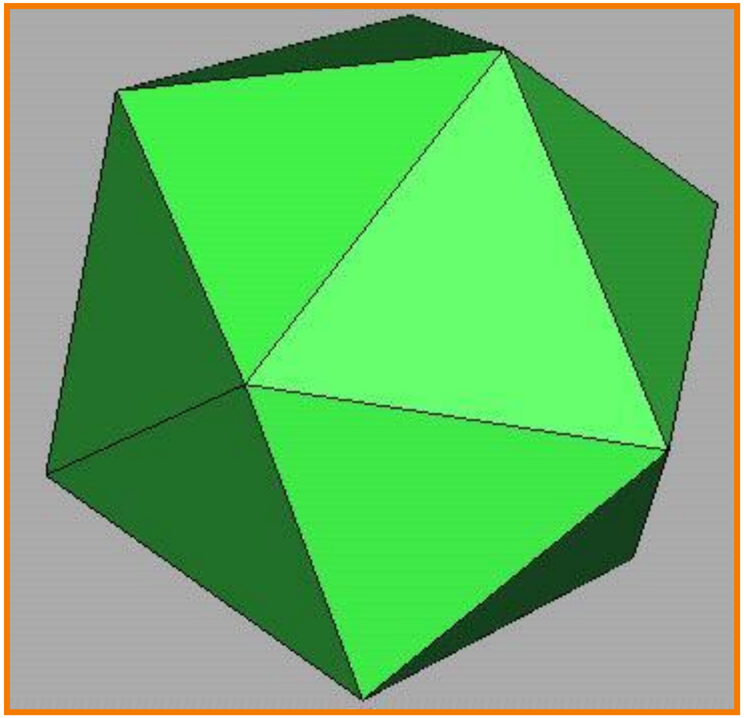

```
vx = vx1 + (wx - wx1) * (vx2 - vx1) / (wx2 - wx1);
vy = vy1 + (wy - wy1) * (vy2 - vy1) / (wy2 - wy1);
```

Summary of Transformations



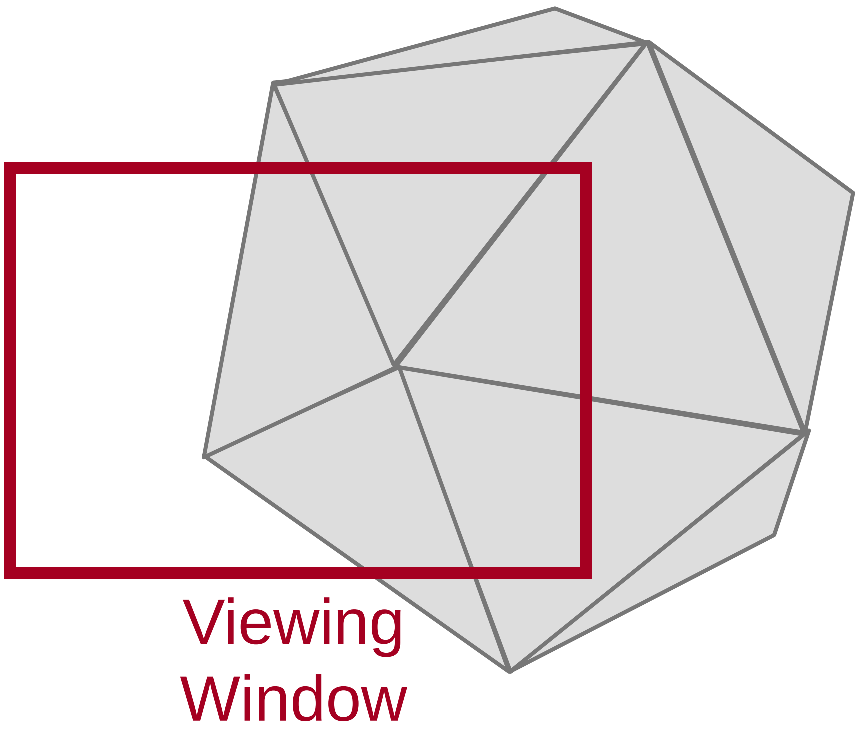
3D Rendering Pipeline (for direct illumination)

3D Primitives
↓ 3D Modeling Coordinates
Modeling Transformation
↓ 3D World Coordinates
Lighting
↓ 3D World Coordinates
Viewing Transformation
↓ 3D Camera Coordinates
Projection Transformation
↓ 2D Screen Coordinates
Clipping
↓ 2D Screen Coordinates
Viewport Transformation
↓ 2D Image Coordinates
Scan Conversion
↓ 2D Image Coordinates
Image



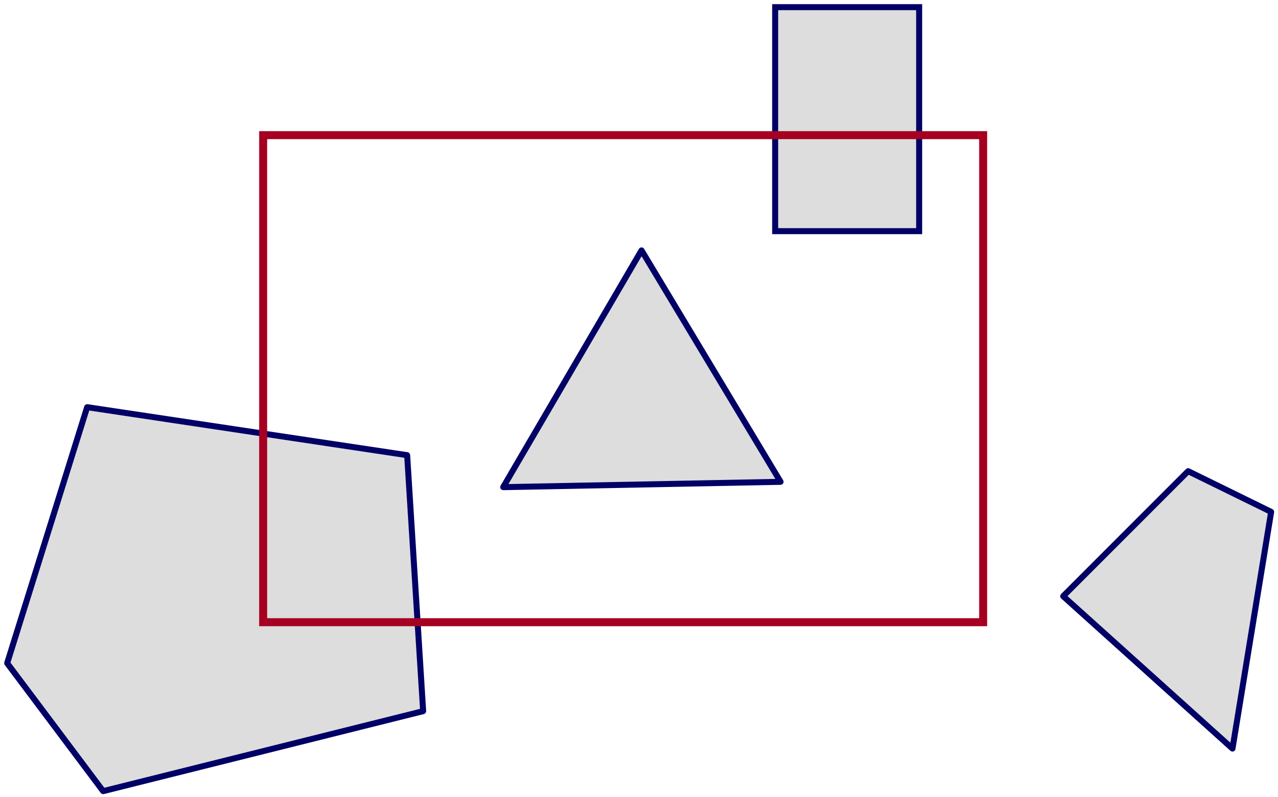
Clipping

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



Polygon Clipping

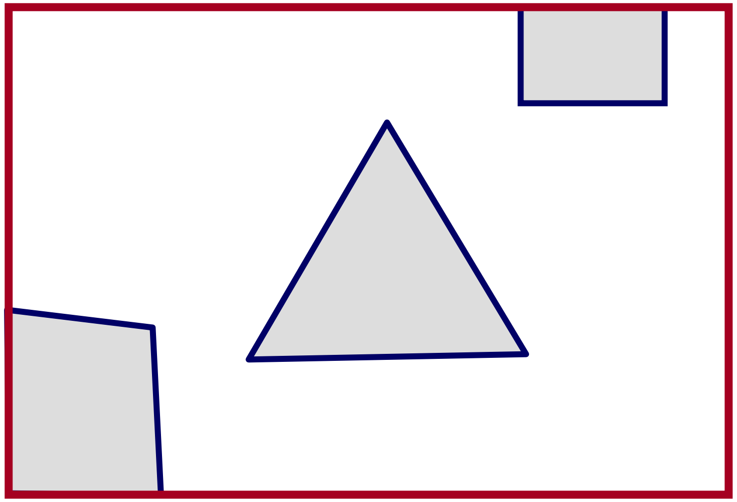
- Find the part of a polygon inside the clip window?



Before Clipping

Polygon Clipping

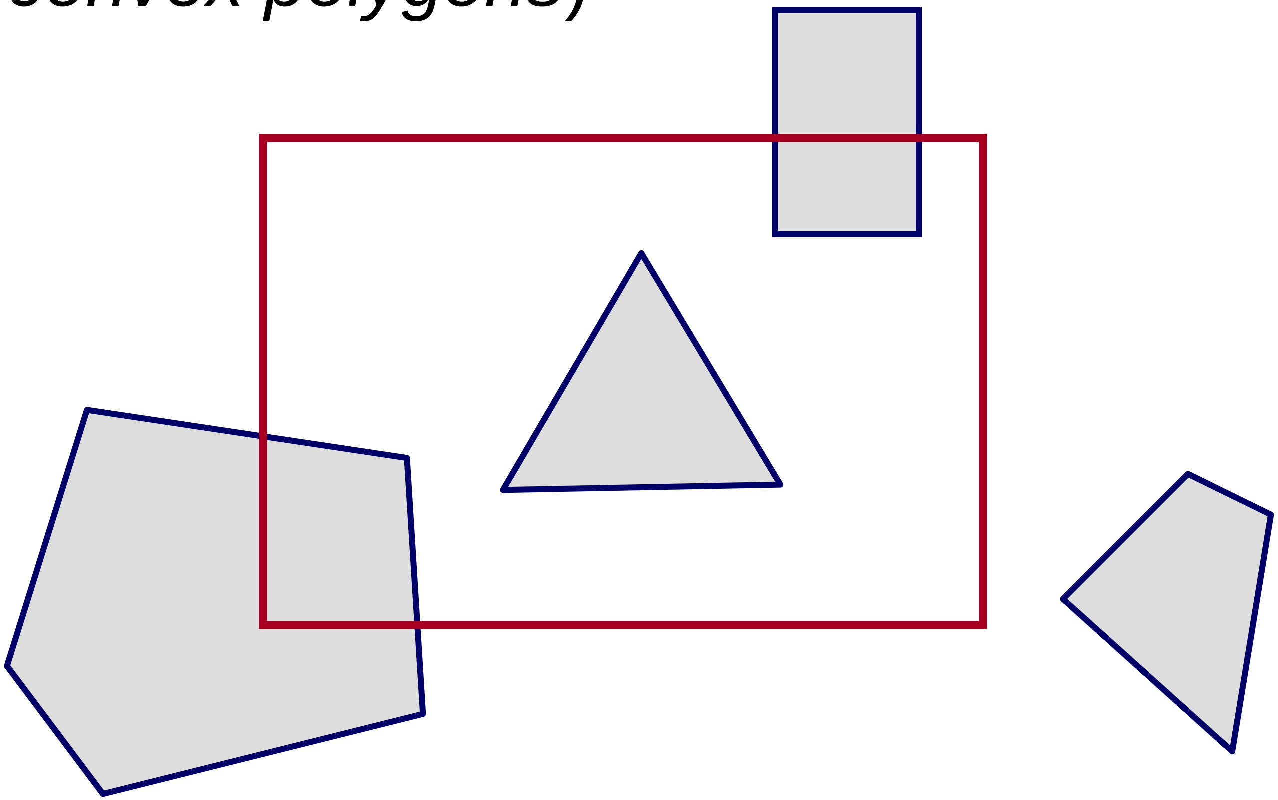
- Find the part of a polygon inside the clip window?



After Clipping

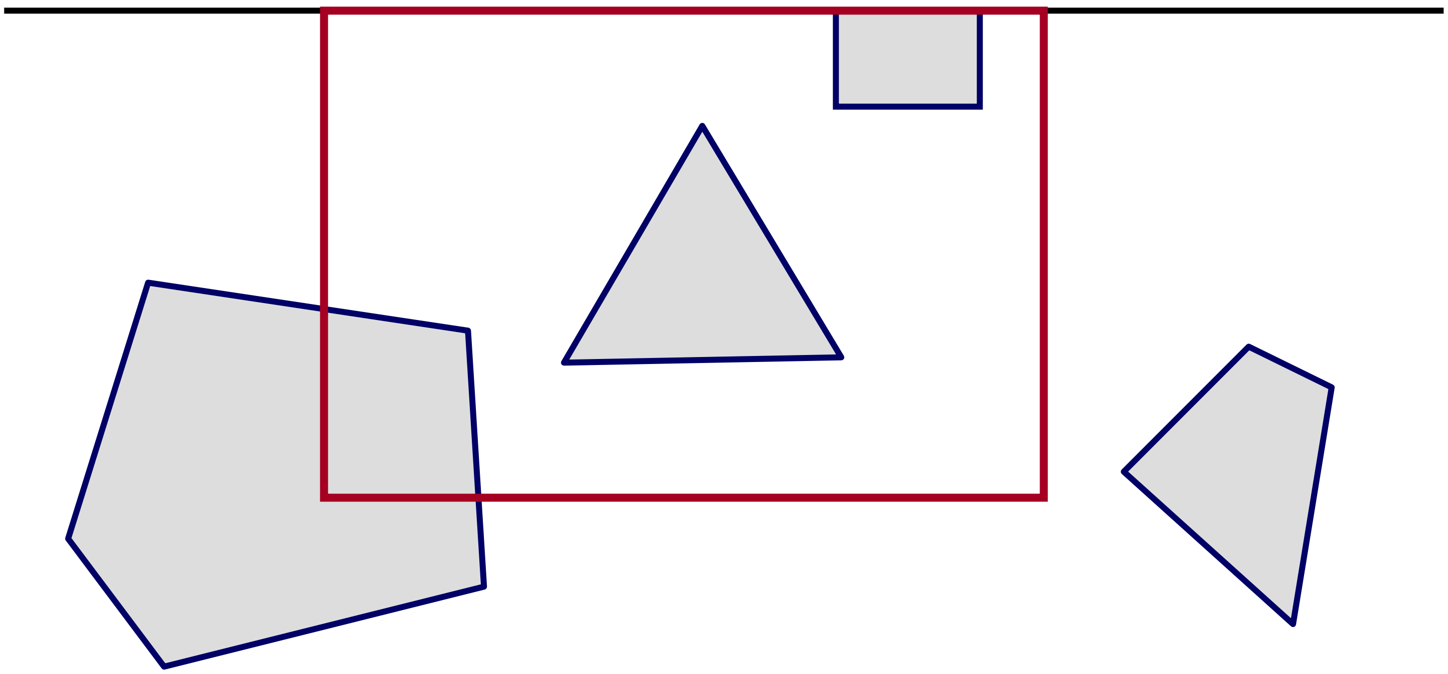
Sutherland Hodgeman Clipping

- Clip to each window boundary one at a time (*for convex polygons*)



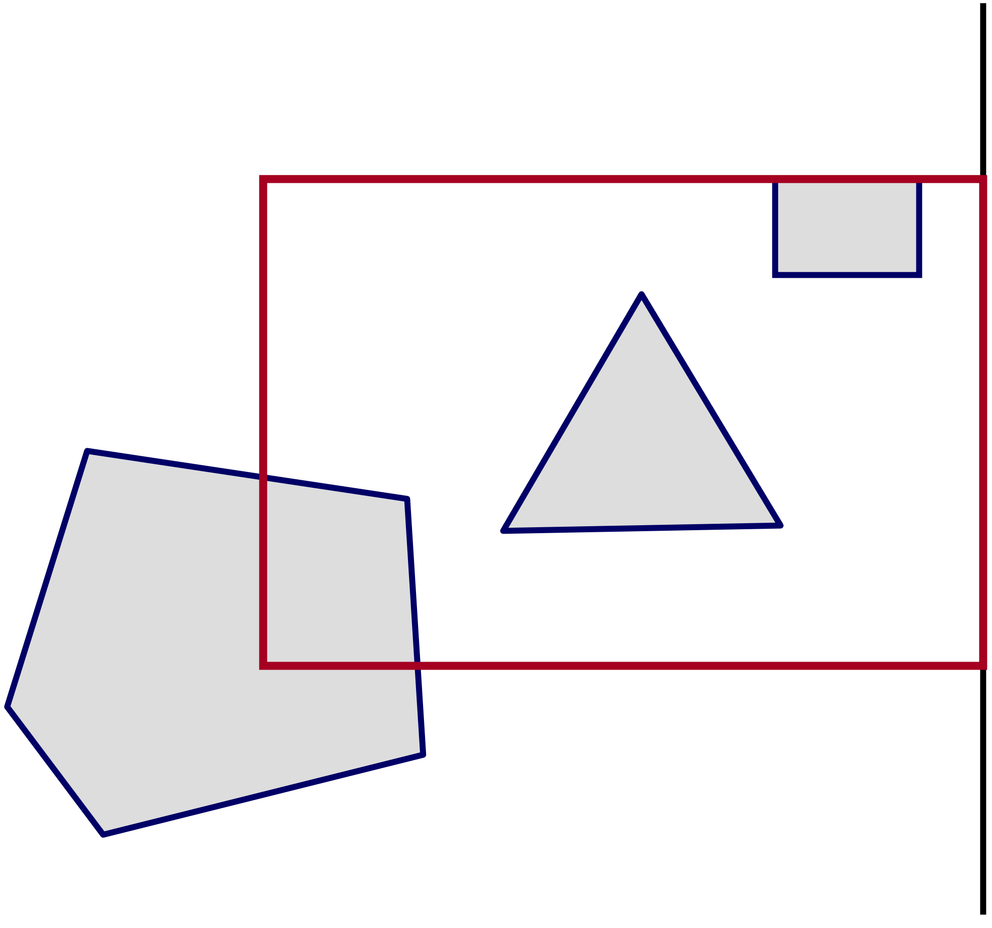
Sutherland Hodgeman Clipping

- Clip to each window boundary one at a time



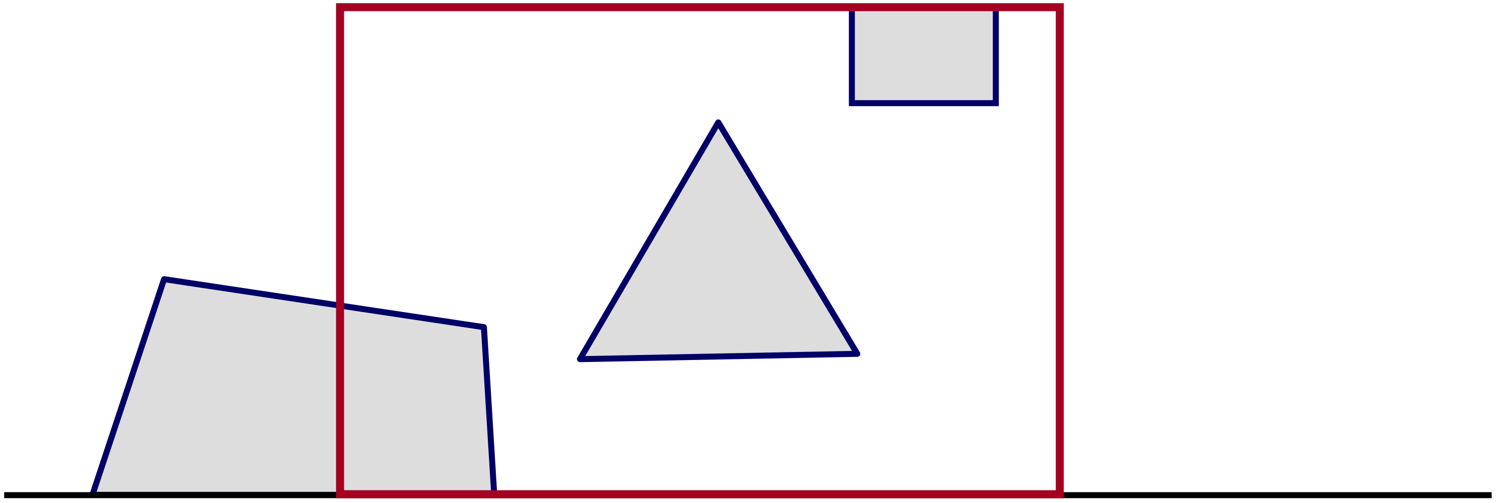
Sutherland Hodgeman Clipping

- Clip to each window boundary one at a time



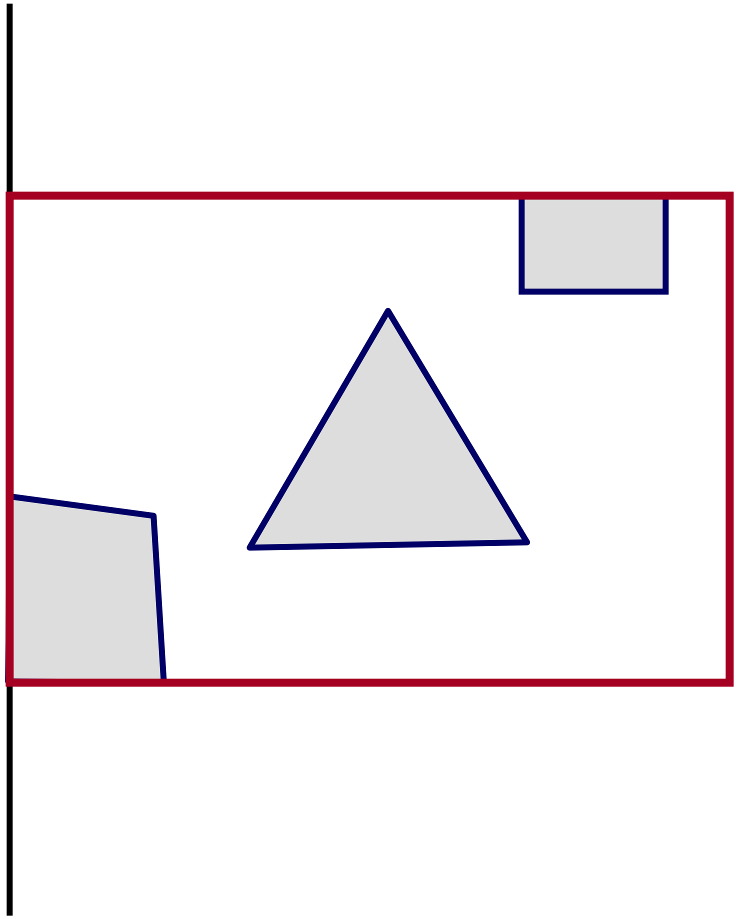
Sutherland Hodgeman Clipping

- Clip to each window boundary one at a time



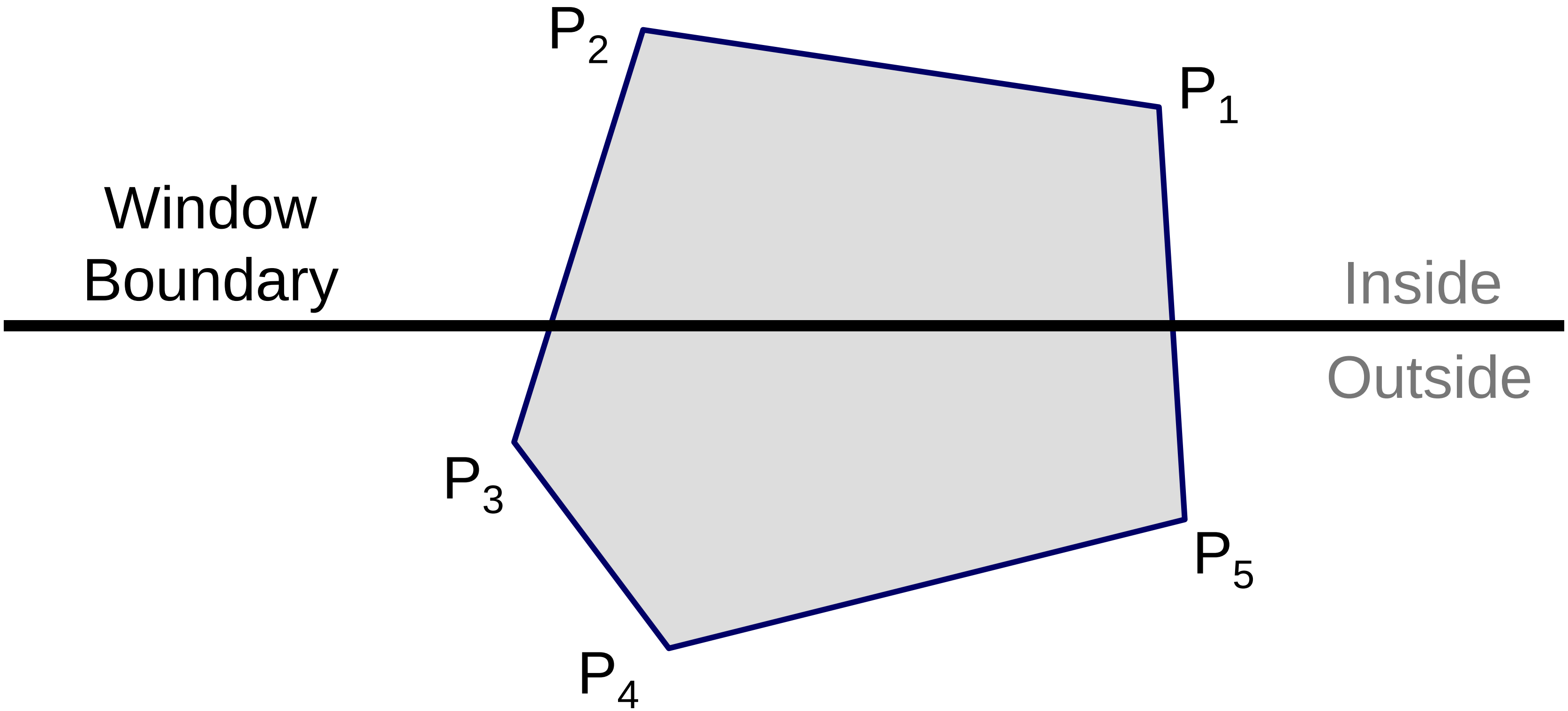
Sutherland Hodgeman Clipping

- Clip to each window boundary one at a time



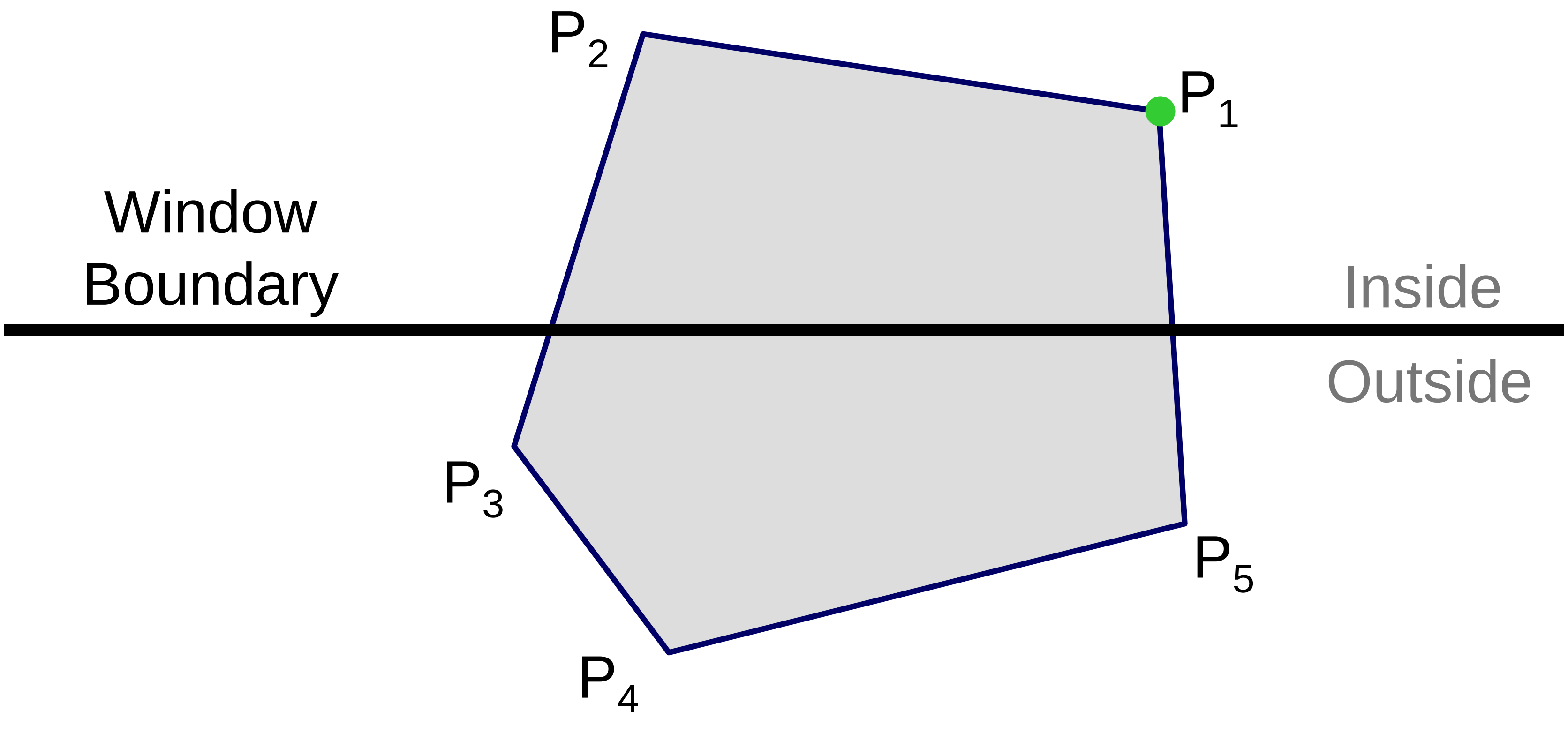
Clipping to a Boundary

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



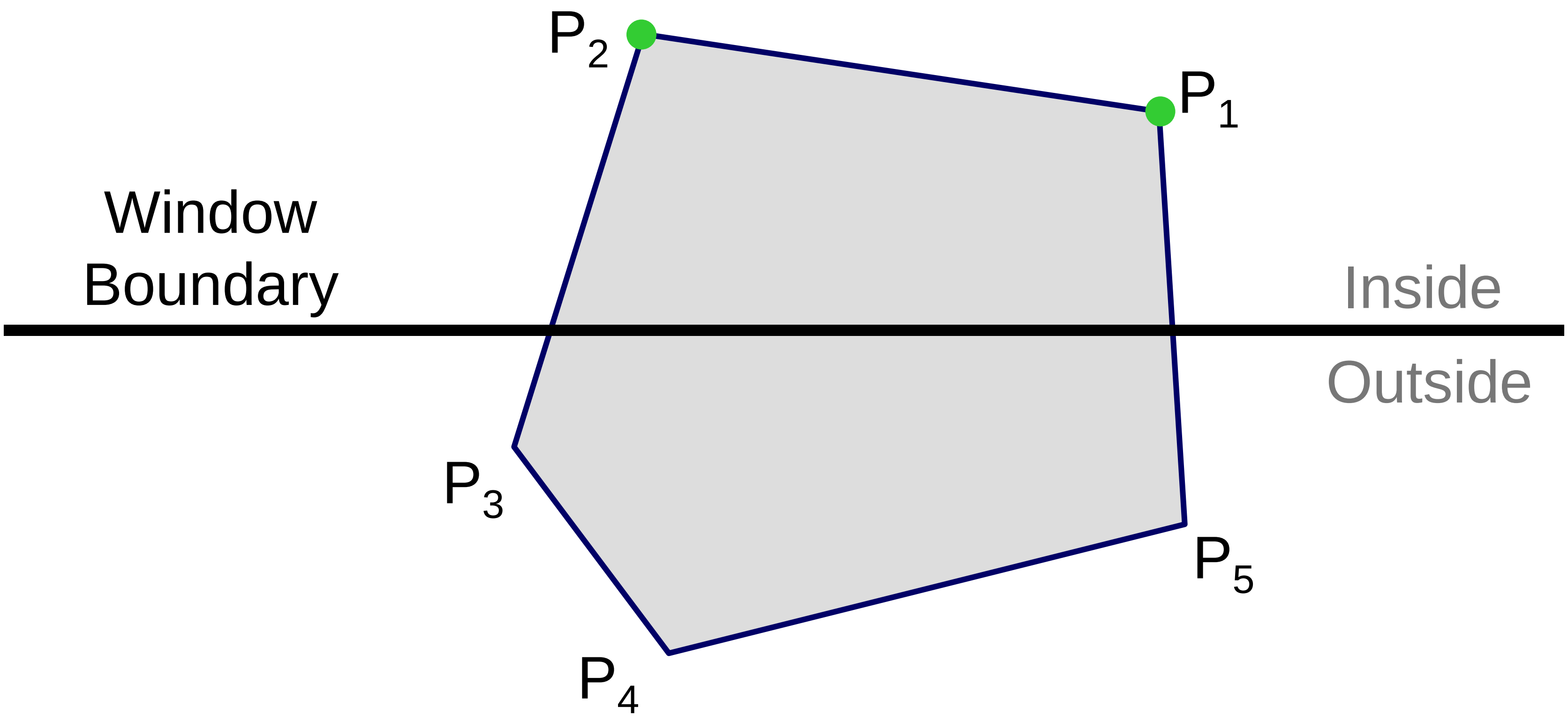
Clipping to a Boundary

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



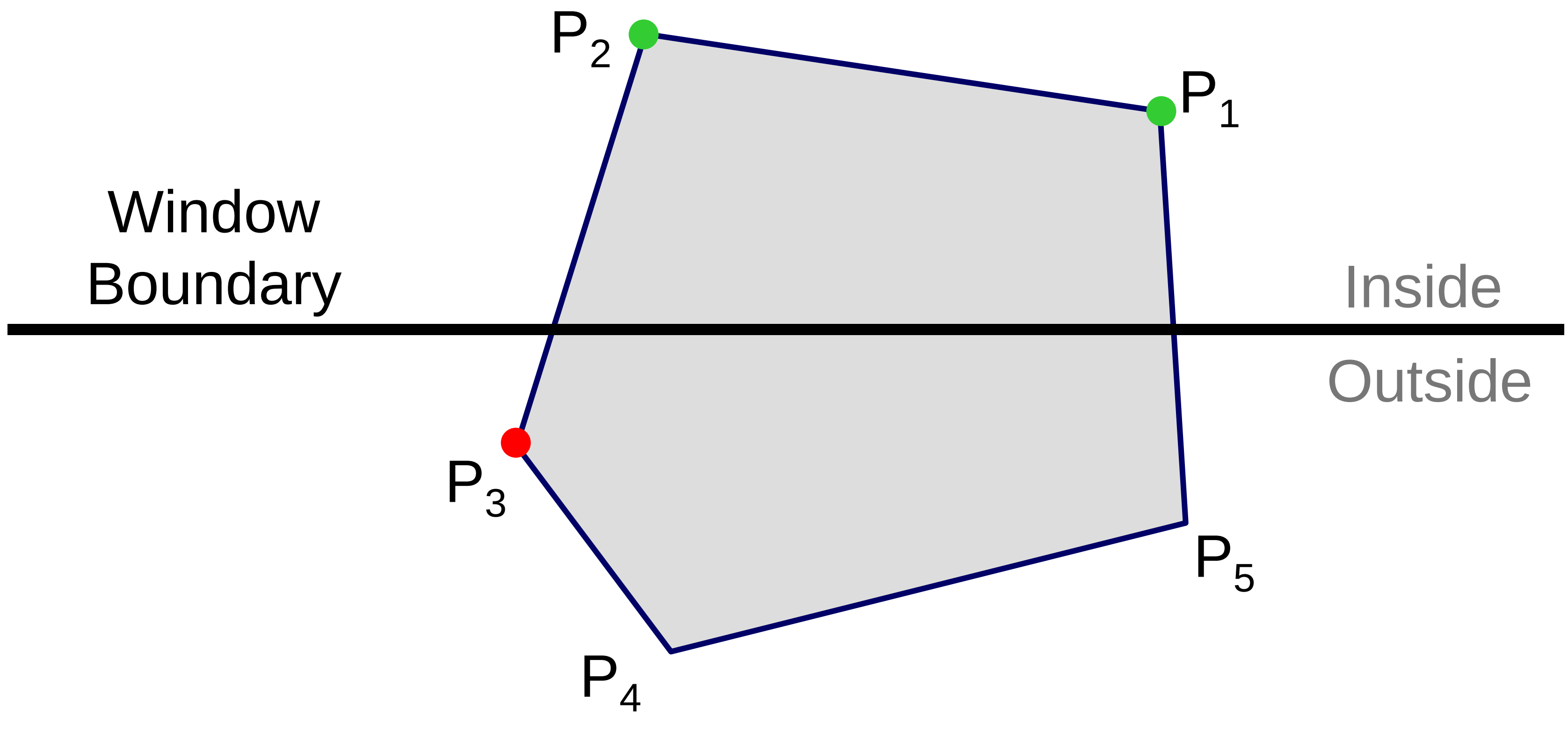
Clipping to a Boundary

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



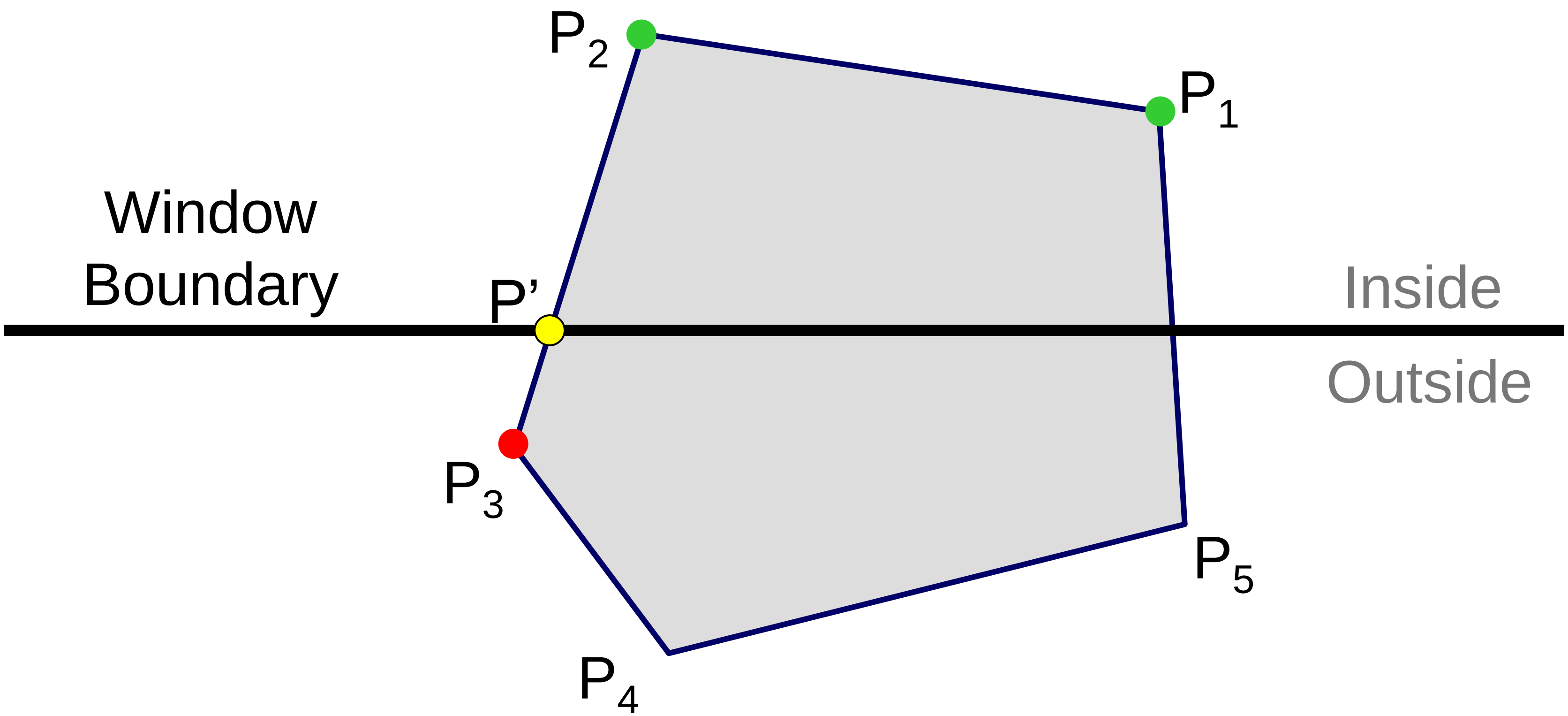
Clipping to a Boundary

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



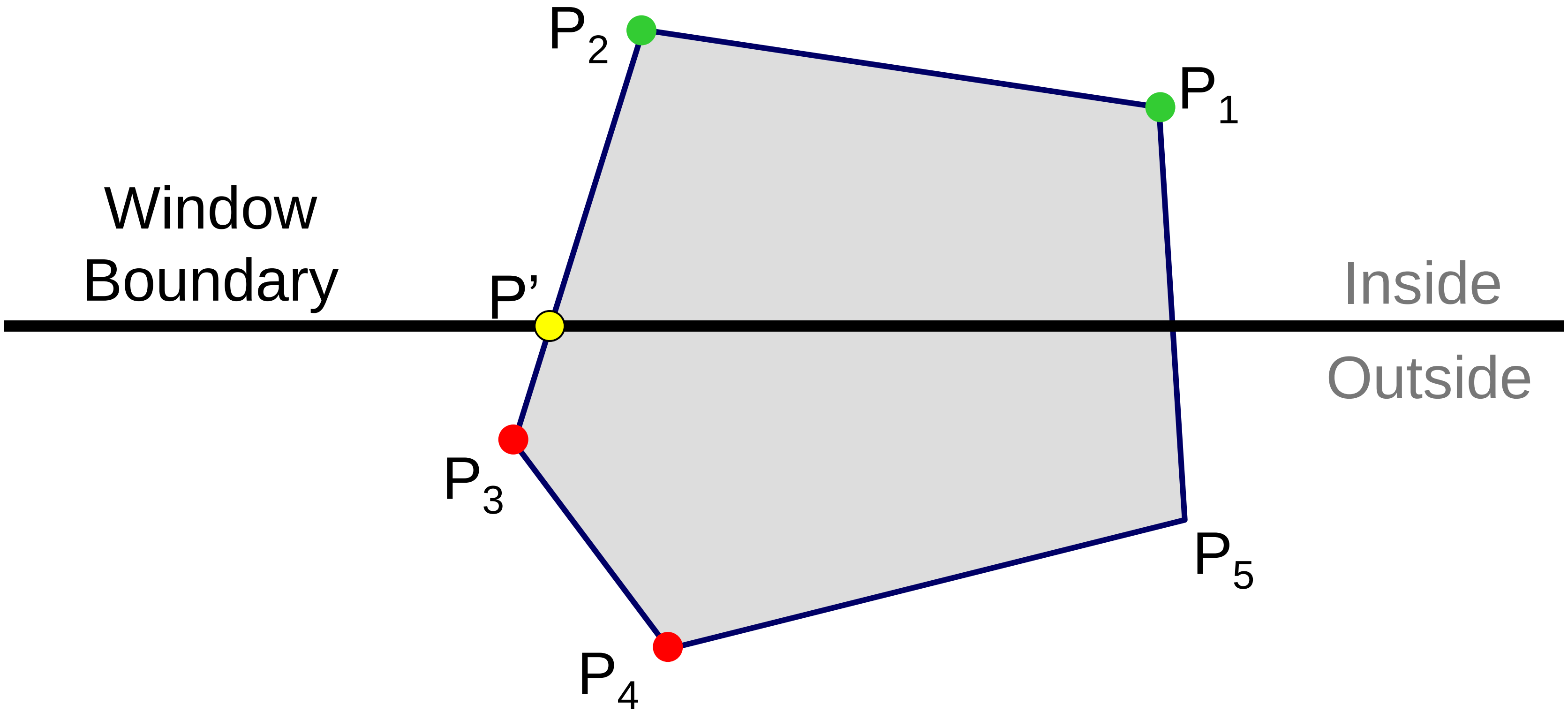
Clipping to a Boundary

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



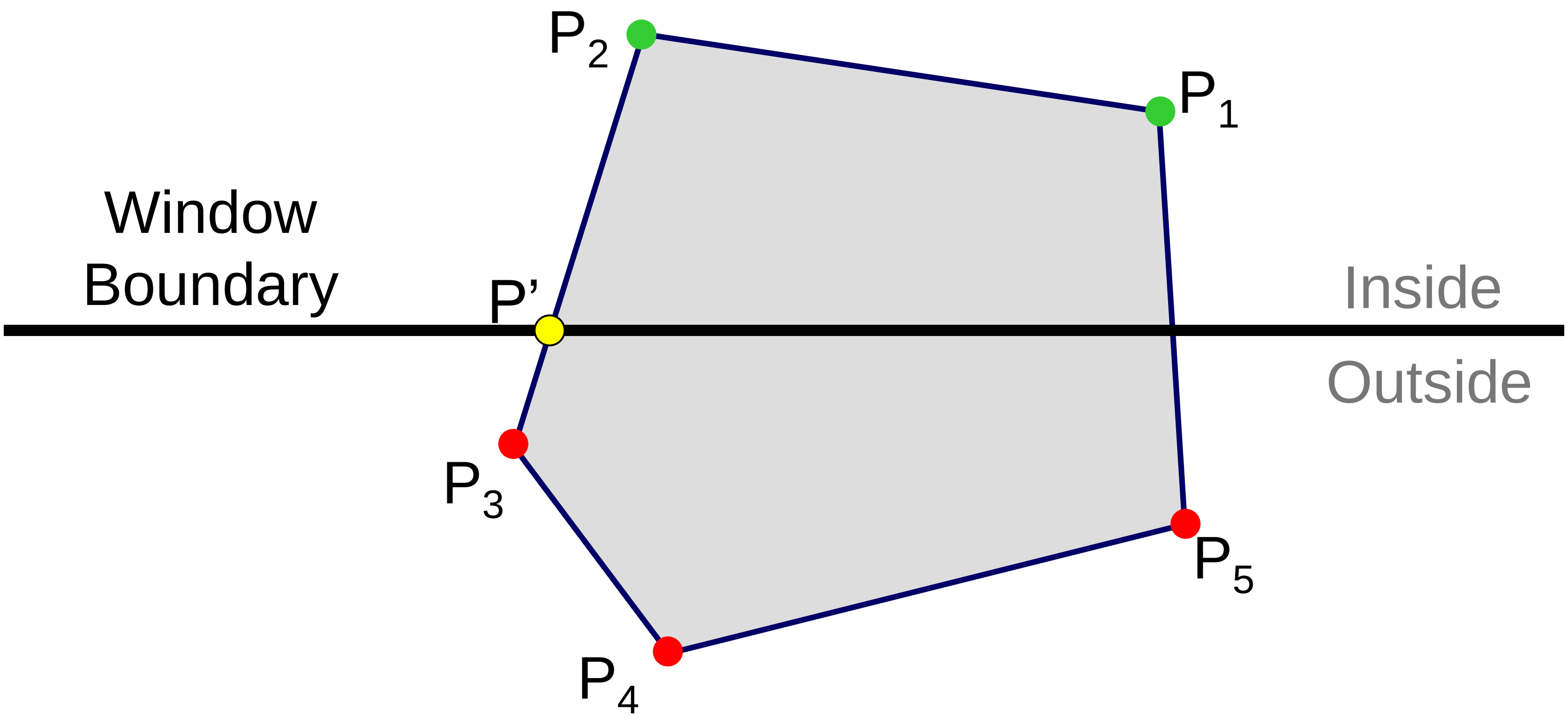
Clipping to a Boundary

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



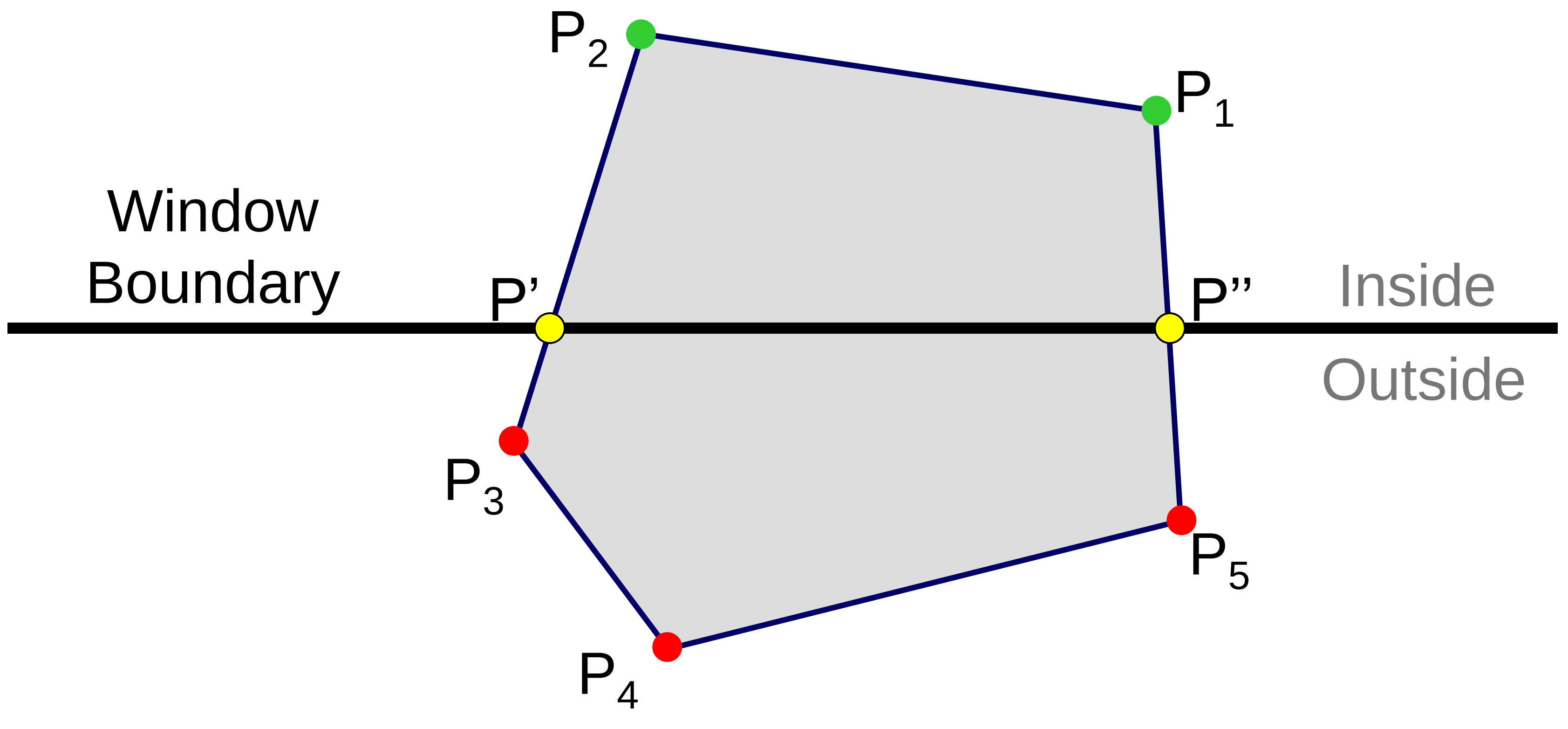
Clipping to a Boundary

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



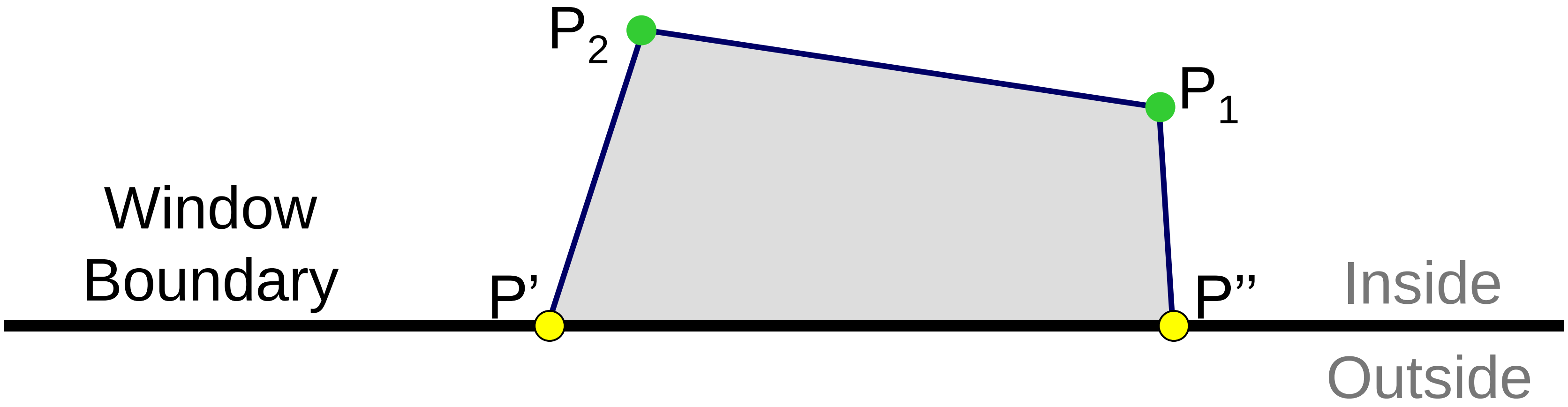
Clipping to a Boundary

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



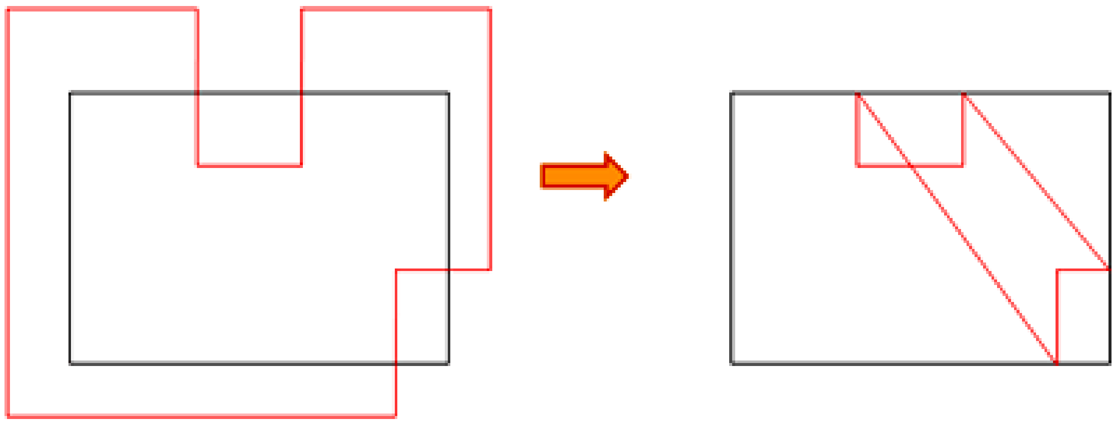
Clipping to a Boundary

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

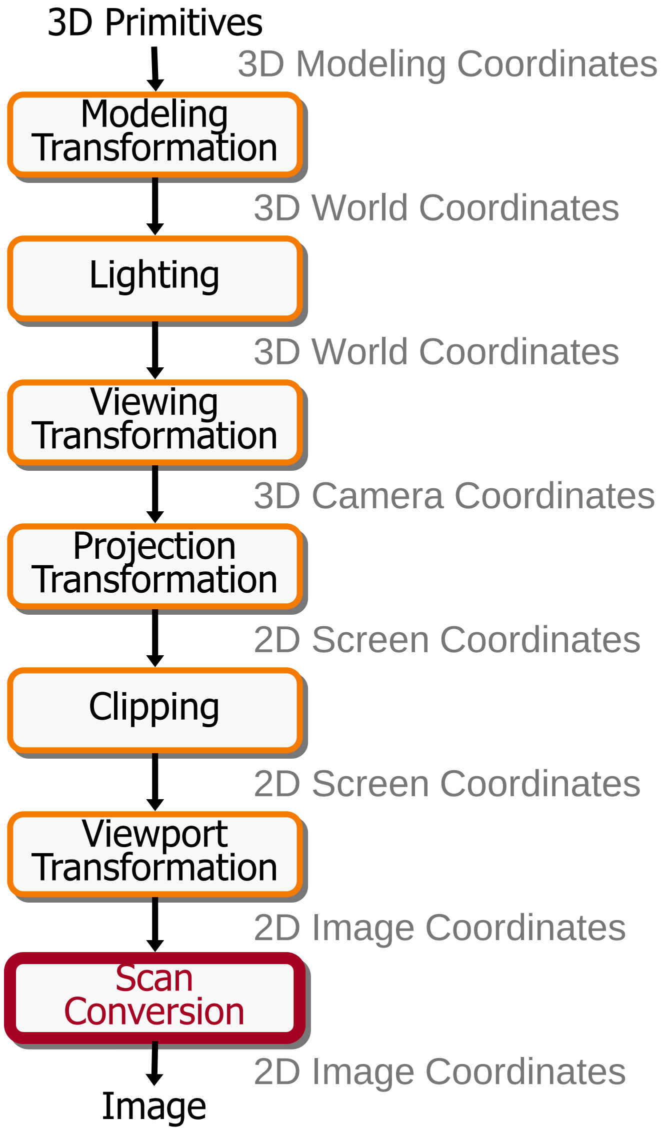


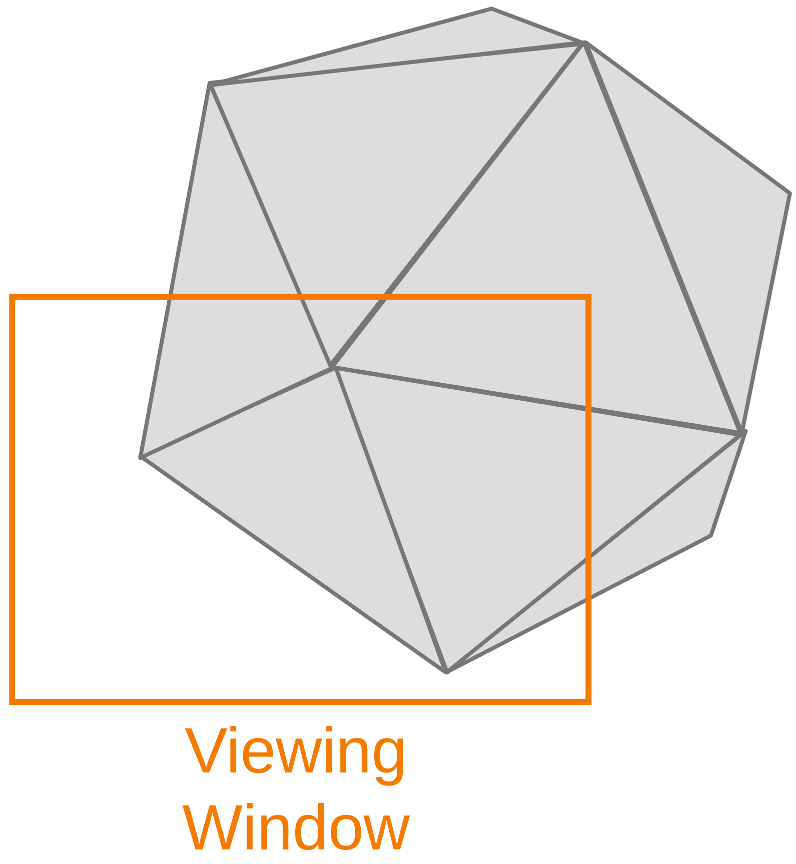
Sutherland Hodgeman Failure

- Concave Polygons

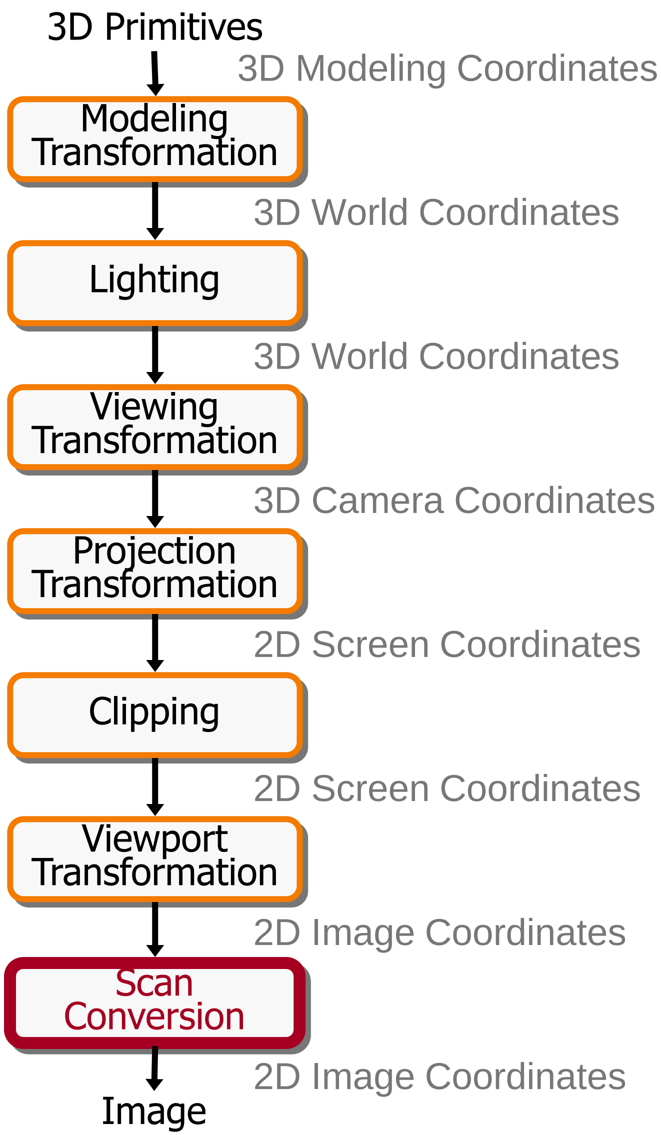


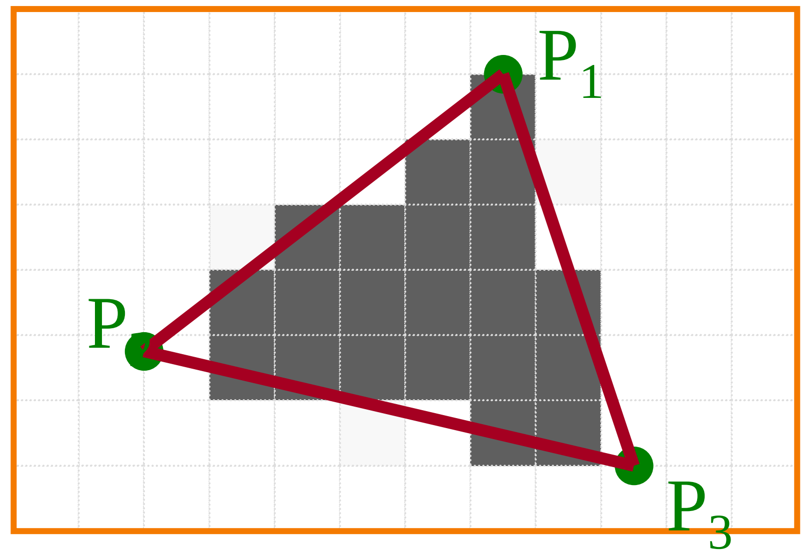
3D Rendering Pipeline (for direct illumination)





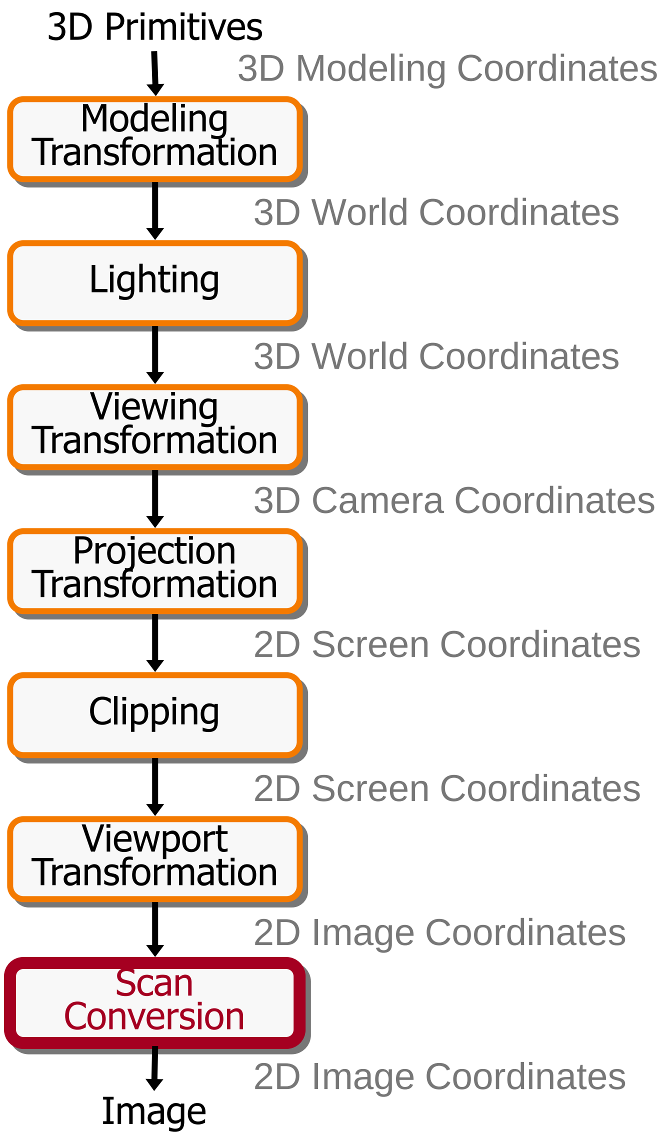
3D Rendering Pipeline (for direct illumination)

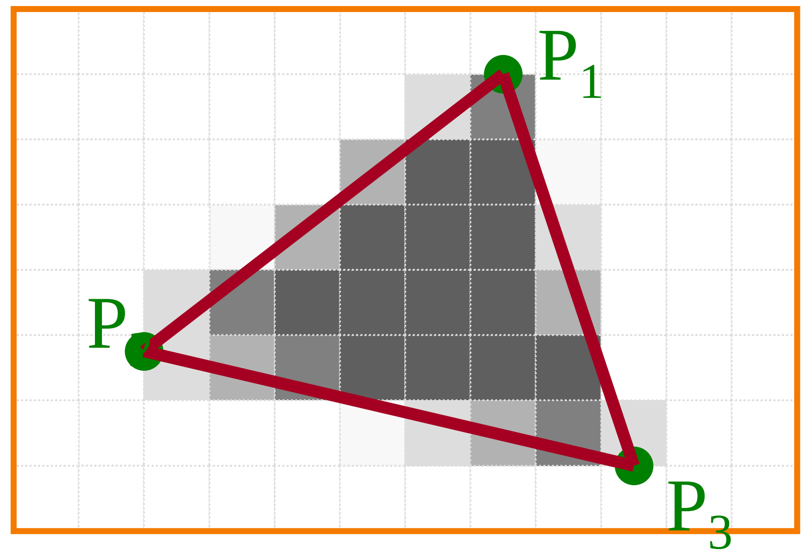




Standard (aliased) Scan Conversion

3D Rendering Pipeline (for direct illumination)





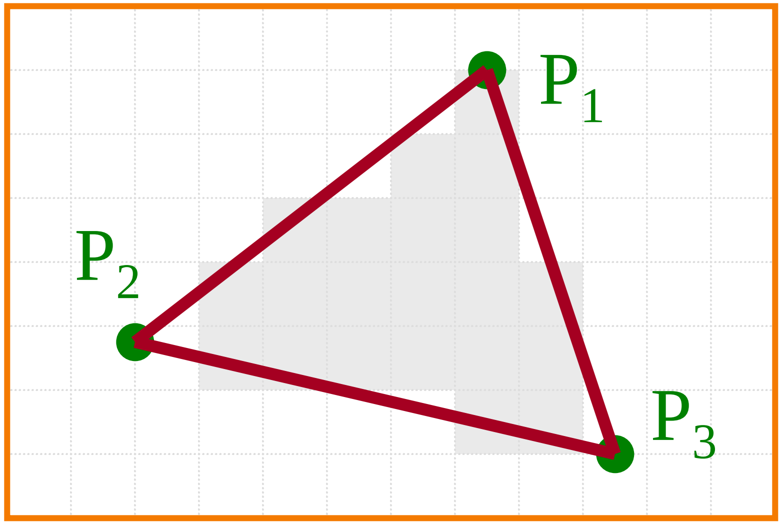
Antialiased Scan Conversion

Scan Conversion

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

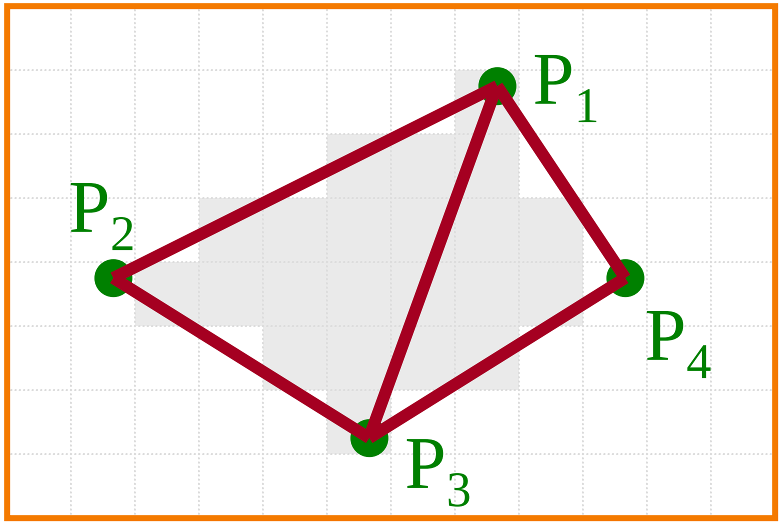
```
void SetPixel(int x, int y, Color rgba)
```

- Example: Filling the inside of a triangle



Triangle Scan Conversion

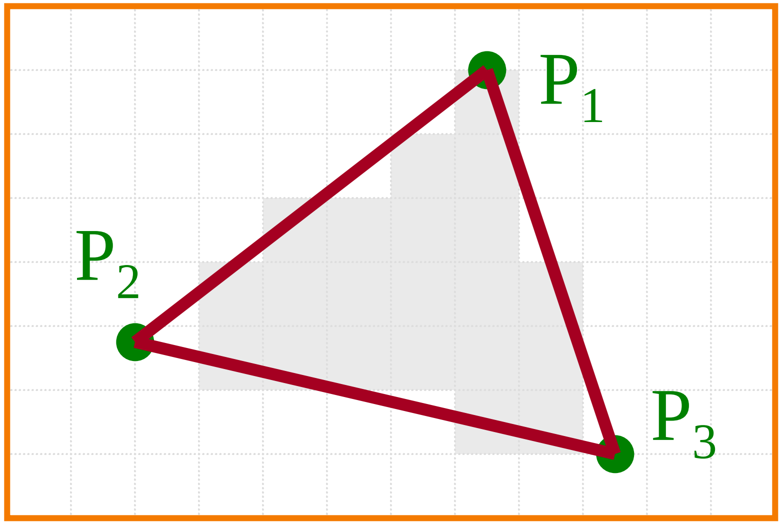
- Properties of a good algorithm
  - Symmetric
  - Straight edges
  - No cracks between adjacent primitives
  - (Antialiased edges)
  - FAST!



Simple Algorithm

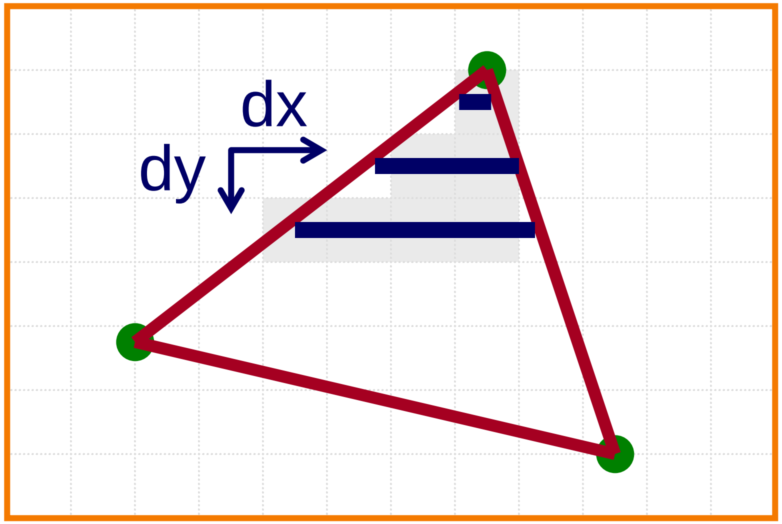
- Color all pixels inside triangle

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



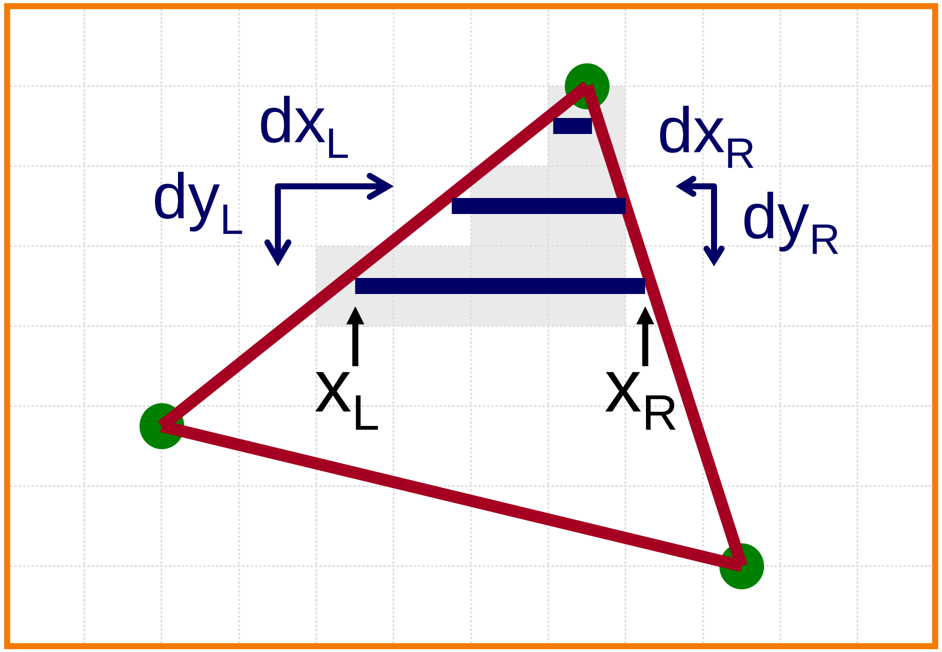
Triangle Sweep-Line Algorithm

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



Triangle Sweep-Line Algorithm

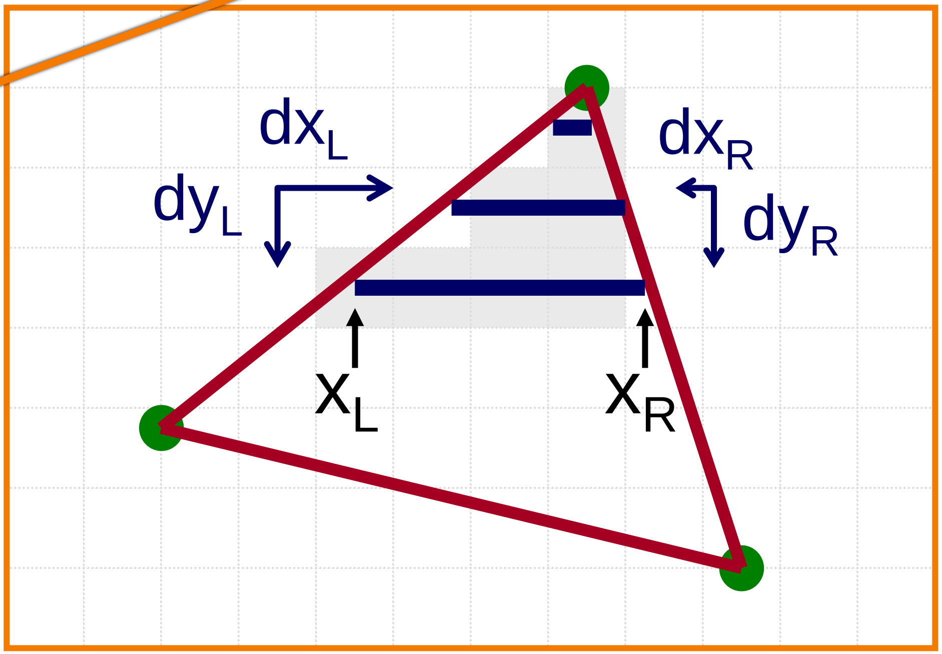
```
void ScanTriangle(Triangle T, Color rgba){
   for each edge pair {
      initialize xL, xR;
      compute dxL/dyL and dxR/dyR;
      for each scanline at y
         for (int x = xL; x <= xR; x++)
            SetPixel(x, y, rgba);
      xL += dxL/dyL;
      xR += dxR/dyR;
   }
}
```



Triangle Sweep-Line Algorithm

```
void ScanTriangle(Triangle T, Color rgba){
   for each edge pair {
      initialize x_L, x_R;
      compute dx_L/dy_L and dx_R/dy_R;
      for each scanline at y
         for (int x = x_L; x <= x_R; x++)
            SetPixel(x, y, rgba);
      x_L += dx_L/dy_L;
      x_R += dx_R/dy_R;
   }
}
```

Minimize computation in inner loops

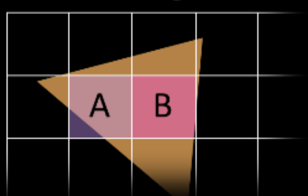


GPU Architecture

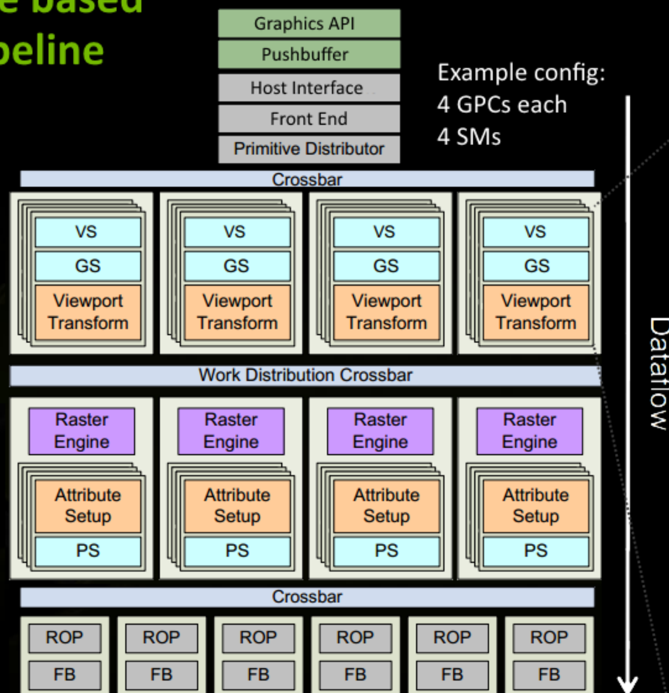
NVIDIA architecture based on Fermi logical pipeline

When tessellation is not used, two principle phases are sufficient. Work is redistributed across entire GPU after each phase.

Work Distribution Crossbar sends triangle to raster engine(s) based on screen rectangle



Multiple GPCs with their SMs can be shading the pixels of one triangle.



Example config:
4 GPCs each
4 SMs

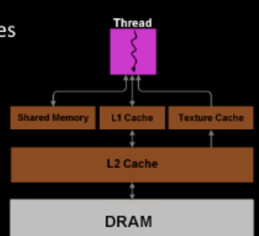
GF 100 Memory Hierarchy

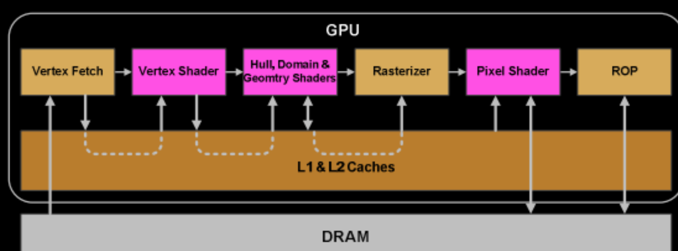
Uniform cache not shown, can cause warp-serialized access on divergent loads

~ latencies

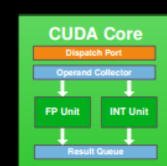
tens of cycles

several hundred cycles

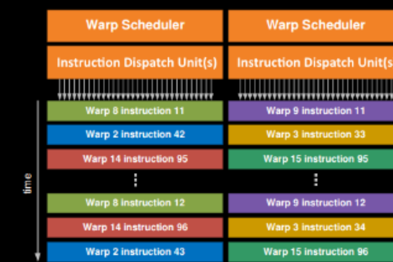




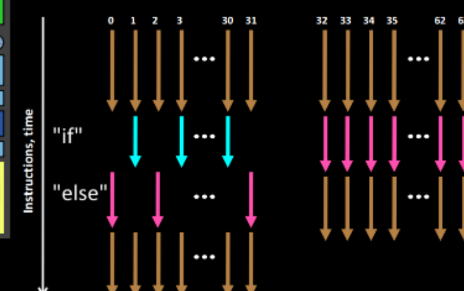
SM organizes threads in groups of 32 called warp. The threads within are processed in lock-step.



Each warp gets subset of register file. If a shader needs many registers -> less warps resident, less latency hiding



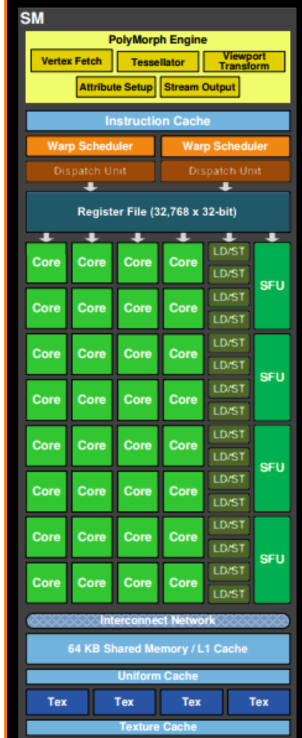
A given warp is processed in-order and it may take several executions until an instruction is advanced (depends on hw-generation and type of instruction). The scheduler switches between warps to avoid waiting for instructions that take longer (memory fetches...).

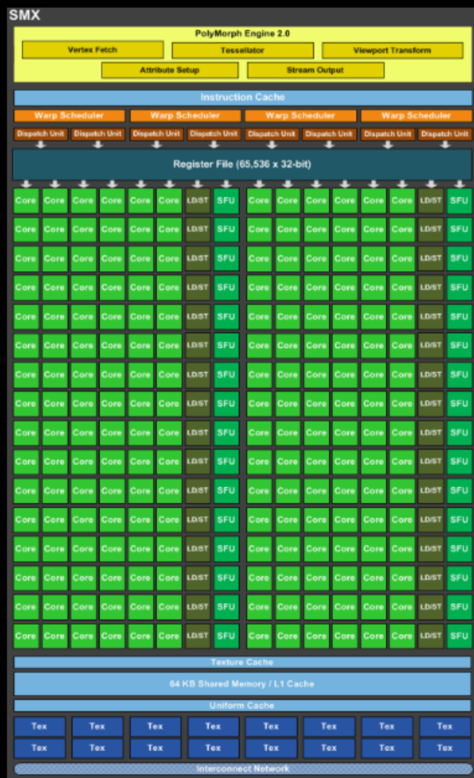


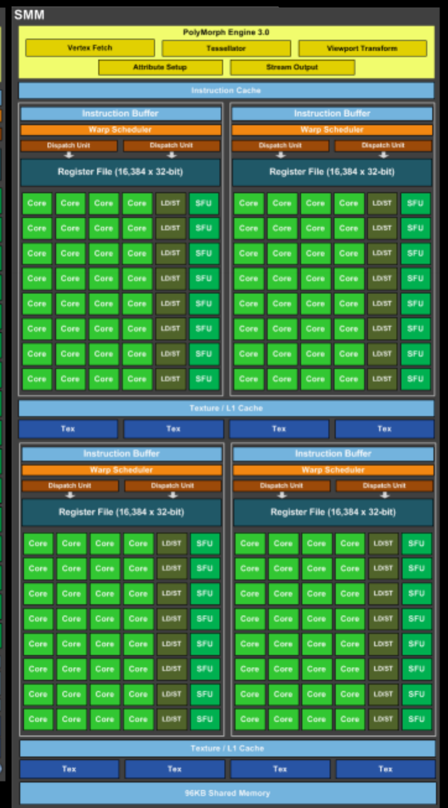
Divergent behavior between threads within warp (if/else block, loops with varying iterations..) can increase computation time for all because of lock-step processing and may risk under utilizing cores.

GPU Architecture

Fermi, Kepler, Maxwell Evolution







GM204 Architecture

image copy-pasted and annotated by @pixeljetstream

Kepler and Maxwell work in principle similar to Fermi.
The most obvious changes are typically in the SM design or number of ROPs. The overall design can be scaled from high-end desktop to mobile by varying the number of modules.

http://www.hardwarebg.com/b4k/files/nvidia_gf100_whitepaper.pdf
http://www.geforce.com/Active/en_US/en_US/pdf/GeForce-GTX-680-Whitepaper-FINAL.pdf
http://international.download.nvidia.com/geforce-com/international/pdfs/GeForce_GTX_980_Whitepaper_FINAL.PDF
on-demand.gputechconf.com/gtc/2013/presentations/S3466-Programming-Guidelines-GPU-Architecture.pdf
www.highperformancegraphics.org/previous/www_2010/media/Hot3D/HPG2010_Hot3D_NVIDIA.pdf

GPU Architecture