



3D Polygon Rendering Pipeline

Tom Funkhouser
Princeton University
COS 426, Spring 2007



3D Rendering Scenarios

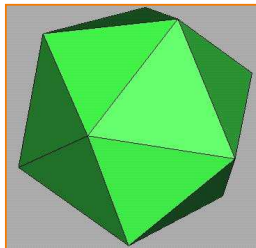
- Batch
 - One image generated with as much quality as possible for a particular set of rendering parameters
 - § Take as much time as is needed (minutes)
 - § Useful for photorealism, movies, etc.
- Interactive
 - Images generated in fraction of a second ($<1/10$) as user controls rendering parameters (e.g., camera)
 - § Achieve highest quality possible in given time
 - § Useful for visualization, games, etc.

rayview



3D Polygon Rendering

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



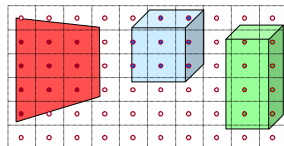
3D Polygon Rendering

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



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 surface radiance

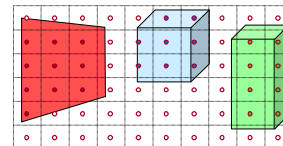


More efficient algorithms
utilize spatial coherence!

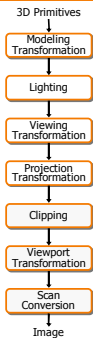


3D Polygon Rendering

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

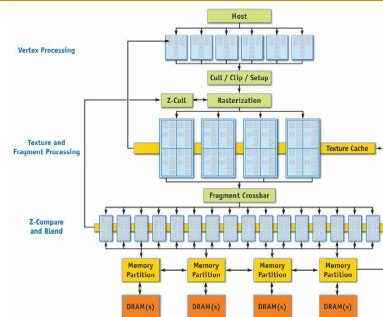


3D Rendering Pipeline (for direct illumination)



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

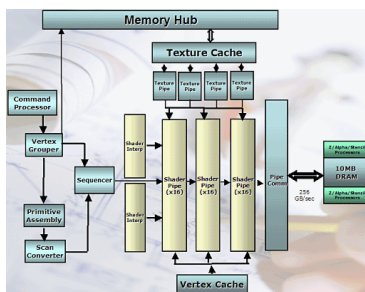
GPU Architecture



GeForce 6 Series Architecture

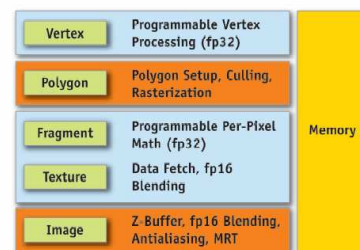
GPU Gems 2, NVIDIA

GPU Architecture



Xbox360, ATI

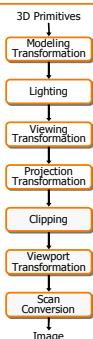
GPU Architecture



GeForce 6 Series Architecture

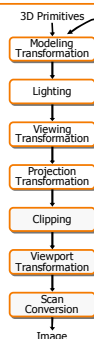
GPU Gems 2, NVIDIA

3D Rendering Pipeline (for direct illumination)



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

3D Rendering Pipeline (for direct illumination)

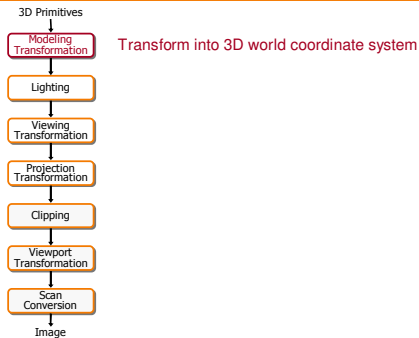


```

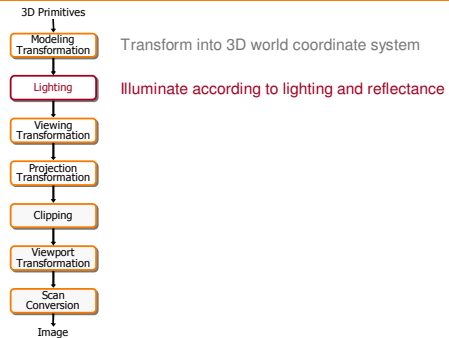
glBegin(GL_POLYGON);
glVertex3f(0.0, 0.0, 0.0);
glVertex3f(1.0, 0.0, 0.0);
glVertex3f(1.0, 1.0, 1.0);
glVertex3f(0.0, 1.0, 1.0);
glEnd();
    
```

OpenGL executes steps of 3D rendering pipeline for each polygon

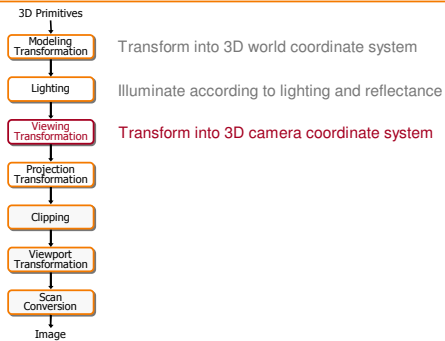
3D Rendering Pipeline (for direct illumination)



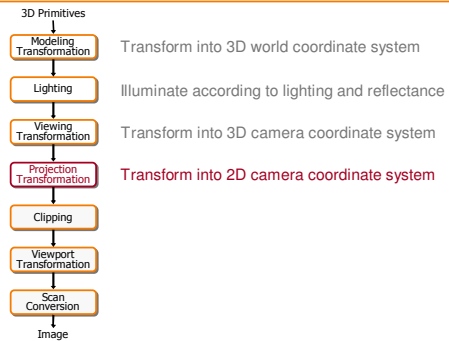
3D Rendering Pipeline (for direct illumination)



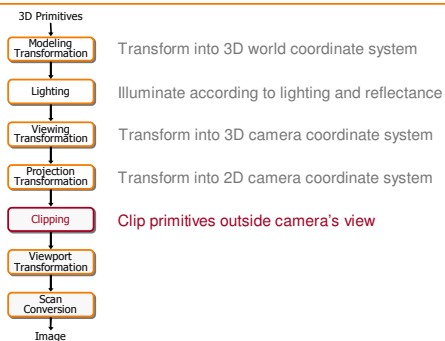
3D Rendering Pipeline (for direct illumination)



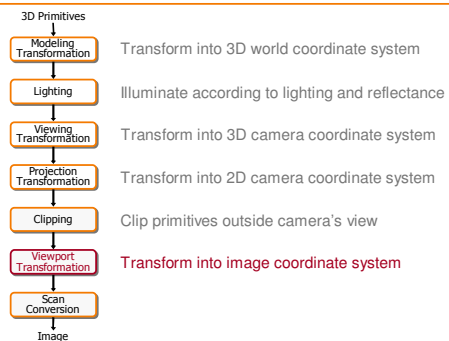
3D Rendering Pipeline (for direct illumination)



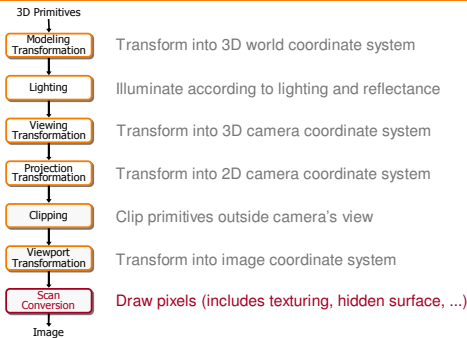
3D Rendering Pipeline (for direct illumination)



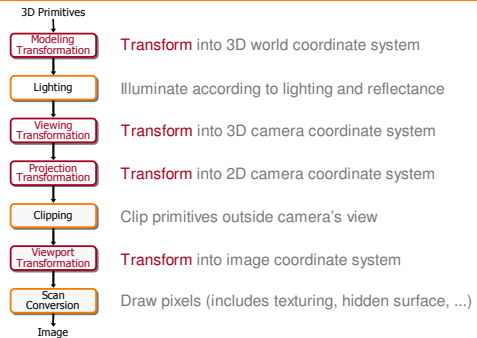
3D Rendering Pipeline (for direct illumination)



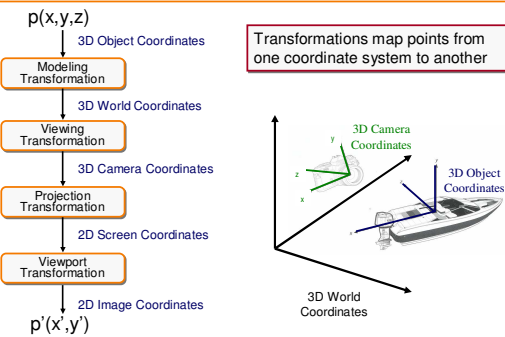
3D Rendering Pipeline (for direct illumination)



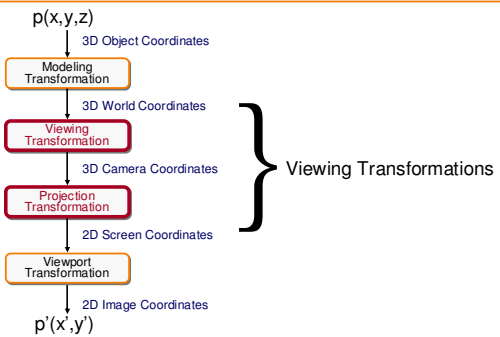
3D Rendering Pipeline (for direct illumination)



Transformations



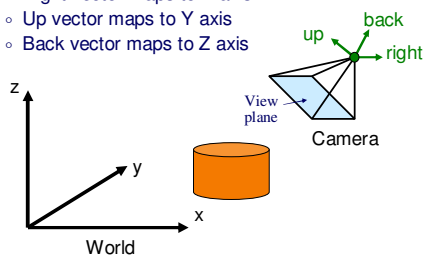
Viewing Transformations



Viewing Transformation



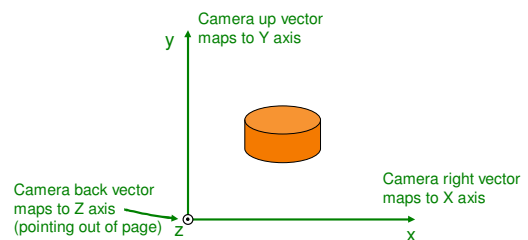
- Mapping from world to camera coordinates
 - Eye position maps to origin
 - Right vector maps to X axis
 - Up vector maps to Y axis
 - Back vector maps to Z axis



Camera Coordinates



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



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

$$p^w = T^{-1} p^c$$

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

↑
?

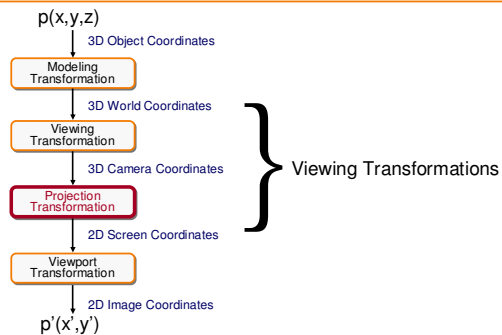
Finding the Viewing Transformation

- 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

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

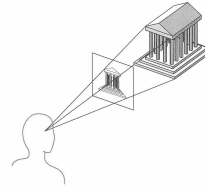
- This matrix is T^{-1} so we invert it to get T ... easy!

Viewing Transformations

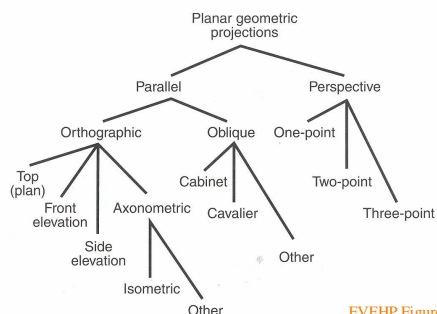


Projection

- General definition:
 - Transform points in n -space to m -space ($m < n$)
- In computer graphics:
 - Map 3D camera coordinates to 2D screen coordinates

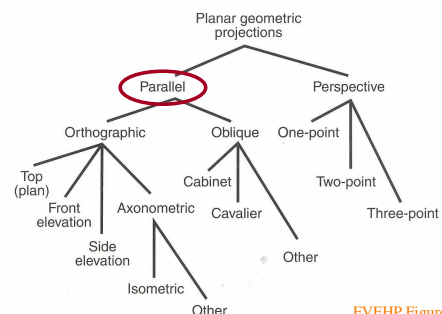


Taxonomy of Projections



FVFHP Figure 6.10

Taxonomy of Projections

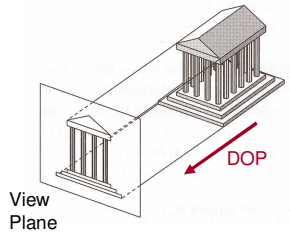


FVFHP Figure 6.10

Parallel Projection



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

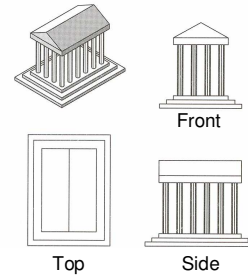


Angel Figure 5.4

Orthographic Projections



- DOP perpendicular to view plane

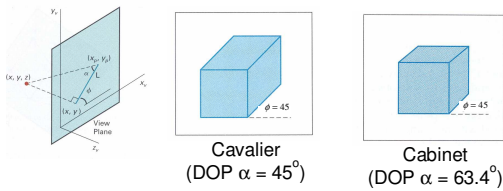


Angel Figure 5.5

Oblique Projections

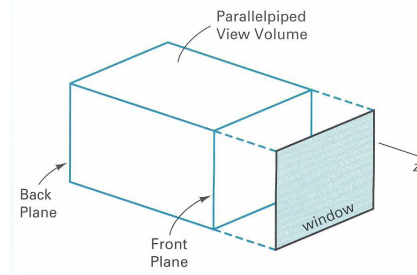


- DOP **not** perpendicular to view plane



H&B Figure 12.24

Parallel Projection View Volume

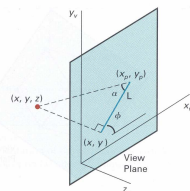


H&B Figure 12.30

Parallel Projection Matrix

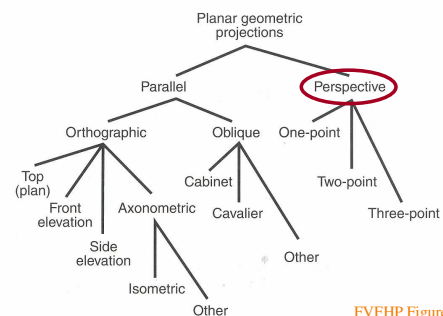


- General parallel projection transformation:



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

Taxonomy of Projections

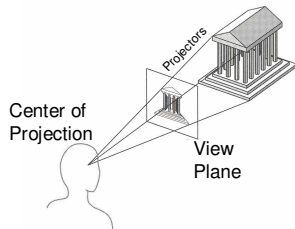


FVHP Figure 6.10

Perspective Projection



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

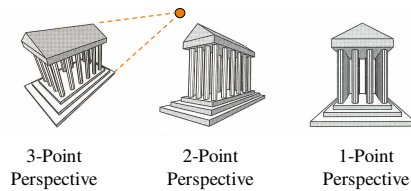


Angel Figure 5.9

Perspective Projection

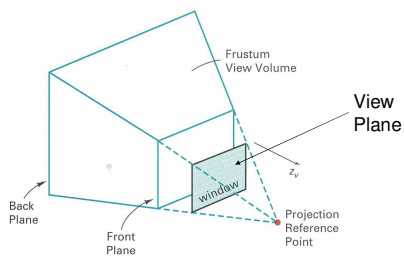


- How many vanishing points?



Angel Figure 5.10

Perspective Projection View Volume

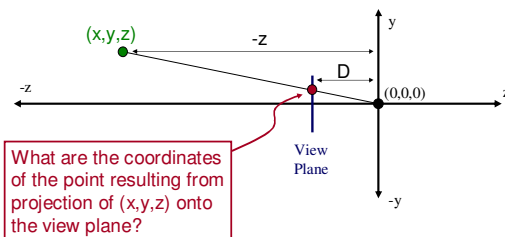


H&B Figure 12.30

Perspective Projection



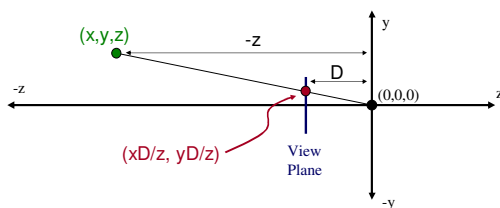
- Compute 2D coordinates from 3D coordinates with similar triangles



Perspective Projection



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



Perspective Projection Matrix



- 4x4 matrix representation?

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

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

Perspective Projection Matrix



- 4x4 matrix representation?

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

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

Perspective Projection Matrix

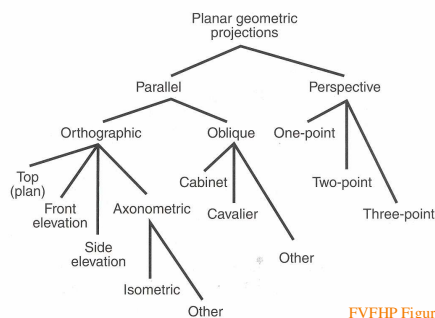


- 4x4 matrix representation?

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

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

Taxonomy of Projections

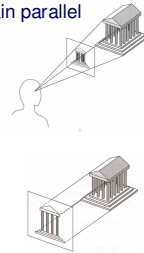


FVFHP Figure 6.10

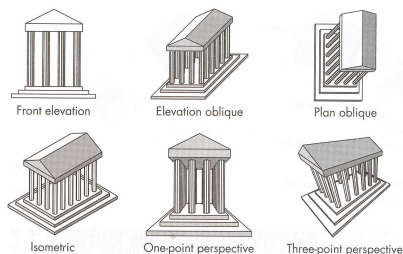
Perspective vs. Parallel



- 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



Classical Projections



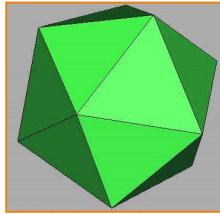
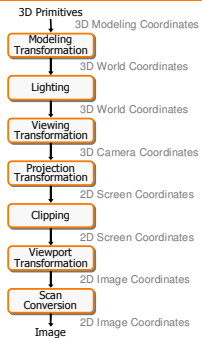
Angel Figure 5.3

Viewing Transformations Summary

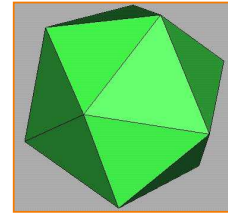
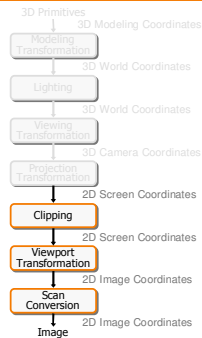


- Camera transformation
 - Map 3D world coordinates to 3D camera coordinates
 - Matrix has camera vectors as rows
- Projection transformation
 - Map 3D camera coordinates to 2D screen coordinates
 - Two types of projections:
 - Parallel
 - Perspective

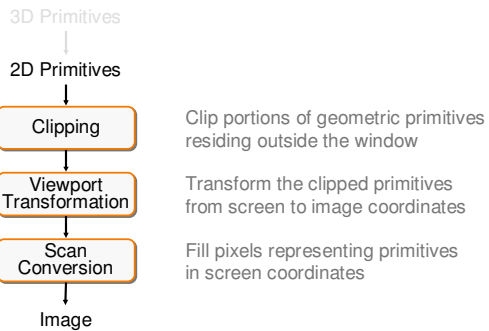
3D Rendering Pipeline (for direct illumination)



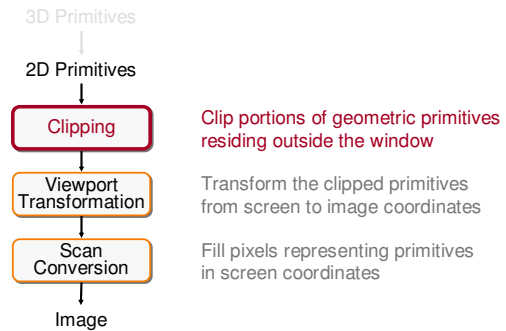
3D Rendering Pipeline (for direct illumination)



2D Rendering Pipeline



2D Rendering Pipeline



Clipping



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

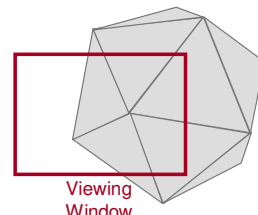


Screen Coordinates

Clipping



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

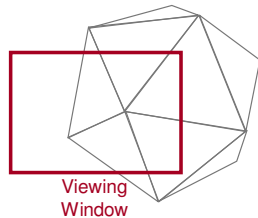


Viewing Window

Clipping



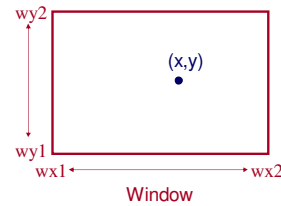
- Avoid drawing parts of primitives outside window
 - Points
 - Lines
 - Polygons
 - Circles
 - etc.



Point Clipping



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

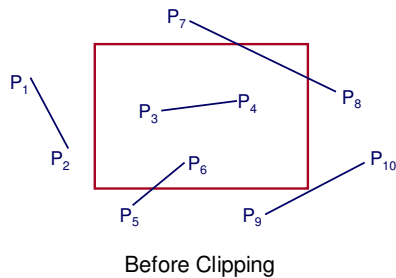


```
inside =
(x >= wx1) &&
(x <= wx2) &&
(y >= wy1) &&
(y <= wy2);
```

Line Clipping



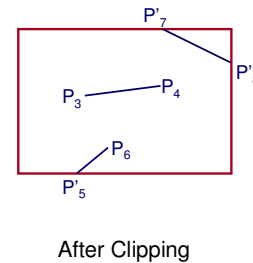
- Find the part of a line inside the clip window



Line Clipping



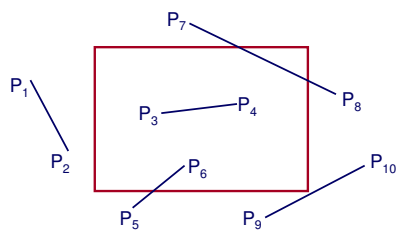
- Find the part of a line inside the clip window



Cohen Sutherland Line Clipping



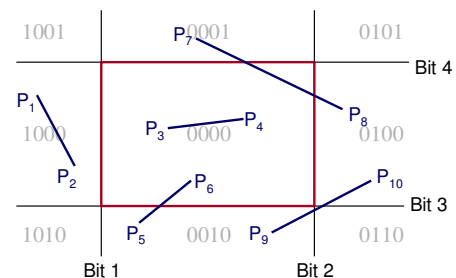
- Use simple tests to classify easy cases first



Cohen Sutherland Line Clipping



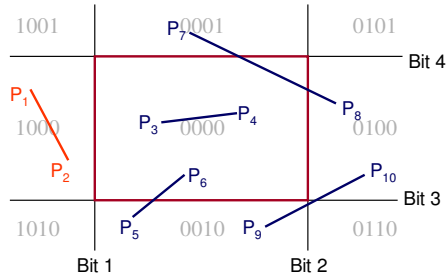
- Classify some lines quickly by AND of bit codes representing regions of two endpoints (must be 0)



Cohen Sutherland Line Clipping



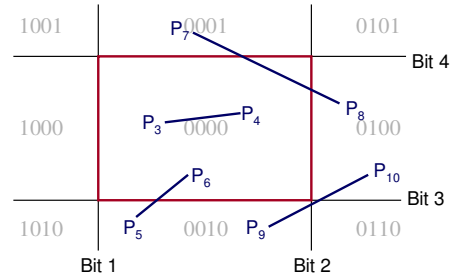
- Classify some lines quickly by AND of bit codes representing regions of two endpoints (must be 0)



Cohen Sutherland Line Clipping



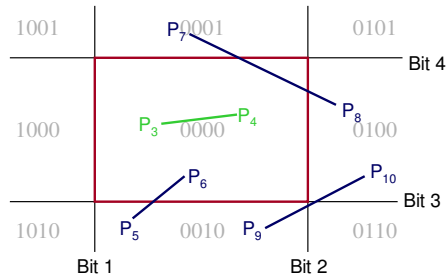
- Classify some lines quickly by AND of bit codes representing regions of two endpoints (must be 0)



Cohen Sutherland Line Clipping



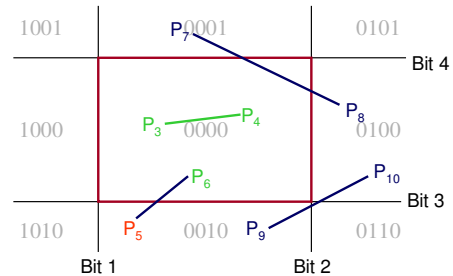
- Classify some lines quickly by AND of bit codes representing regions of two endpoints (must be 0)



Cohen-Sutherland Line Clipping



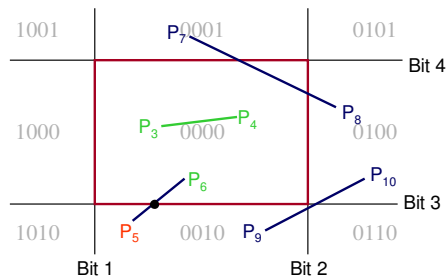
- Compute interesections with window boundary for lines that can't be classified quickly



Cohen-Sutherland Line Clipping



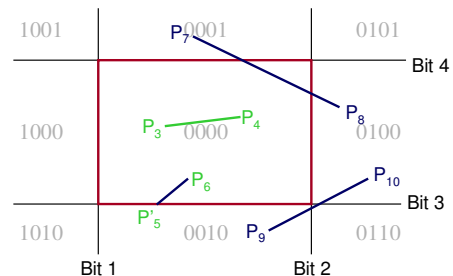
- Compute interesections with window boundary for lines that can't be classified quickly



Cohen-Sutherland Line Clipping



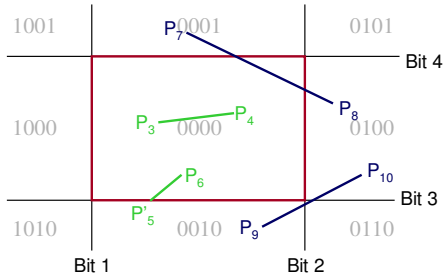
- Compute interesections with window boundary for lines that can't be classified quickly



Cohen-Sutherland Line Clipping



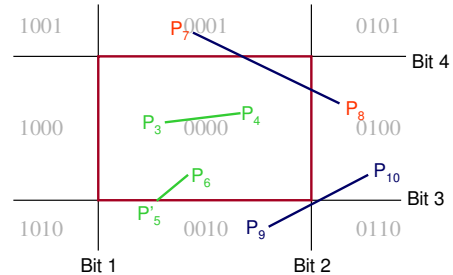
- Compute interesections with window boundary for lines that can't be classified quickly



Cohen-Sutherland Line Clipping



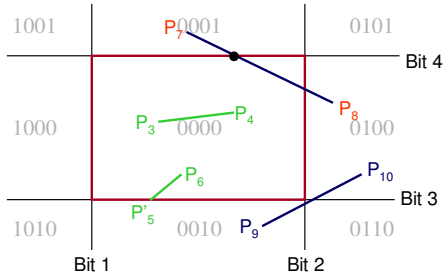
- Compute interesections with window boundary for lines that can't be classified quickly



Cohen-Sutherland Line Clipping



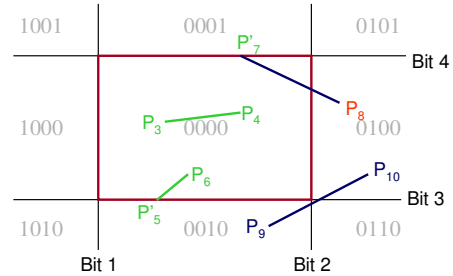
- Compute interesections with window boundary for lines that can't be classified quickly



Cohen-Sutherland Line Clipping



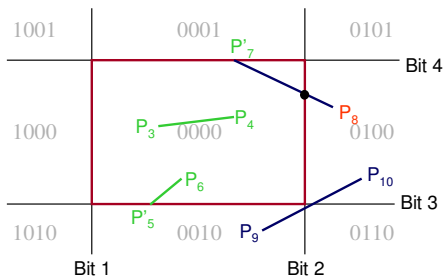
- Compute interesections with window boundary for lines that can't be classified quickly



Cohen-Sutherland Line Clipping



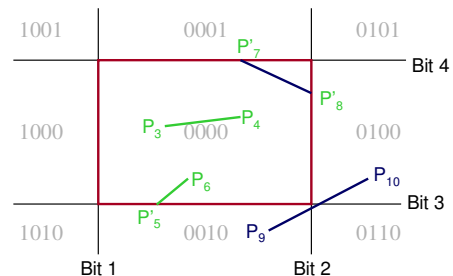
- Compute interesections with window boundary for lines that can't be classified quickly



Cohen-Sutherland Line Clipping



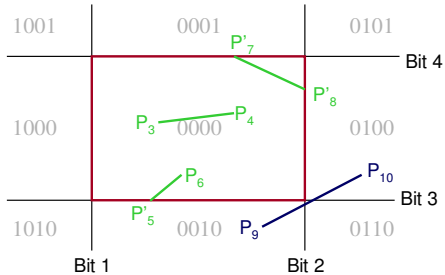
- Compute interesections with window boundary for lines that can't be classified quickly



Cohen-Sutherland Line Clipping



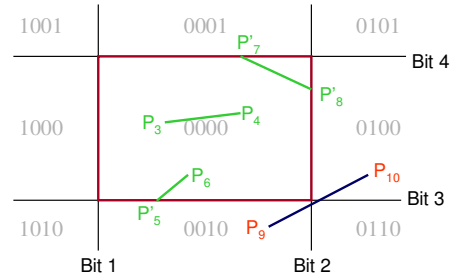
- Compute interesections with window boundary for lines that can't be classified quickly



Cohen-Sutherland Line Clipping



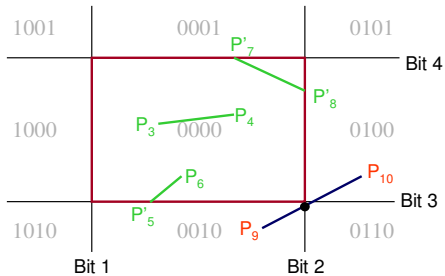
- Compute interesections with window boundary for lines that can't be classified quickly



Cohen-Sutherland Line Clipping



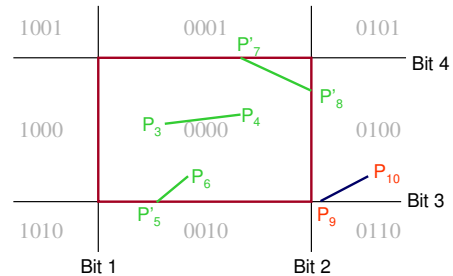
- Compute interesections with window boundary for lines that can't be classified quickly



Cohen-Sutherland Line Clipping



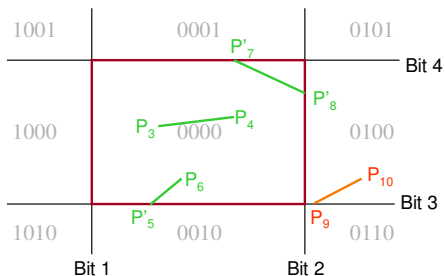
- Compute interesections with window boundary for lines that can't be classified quickly



Cohen-Sutherland Line Clipping



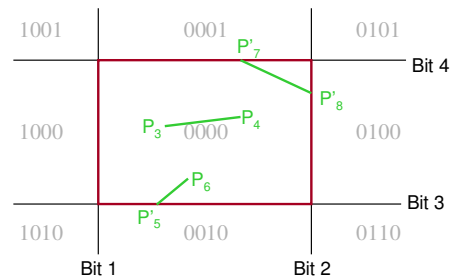
- Compute interesections with window boundary for lines that can't be classified quickly



Cohen-Sutherland Line Clipping



- Compute interesections with window boundary for lines that can't be classified quickly

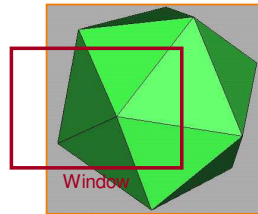


Clipping



- Avoid drawing parts of primitives outside window

- Points
- Lines
- Polygons
- Circles
- etc.

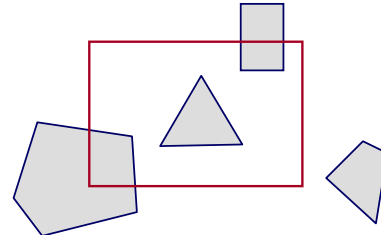


2D Screen Coordinates

Polygon Clipping



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

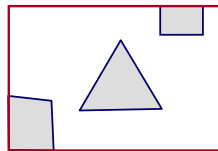


Before Clipping

Polygon Clipping



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

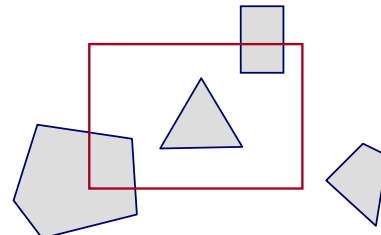


After Clipping

Sutherland Hodgeman Clipping



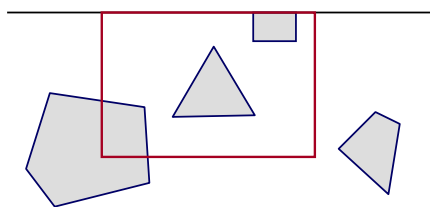
- Clip to each window boundary one at a time



Sutherland Hodgeman Clipping



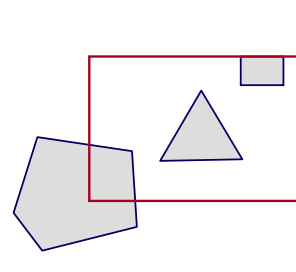
- Clip to each window boundary one at a time



Sutherland Hodgeman Clipping



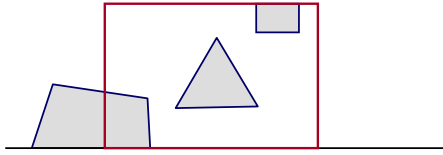
- Clip to each window boundary one at a time



Sutherland Hodgeman Clipping



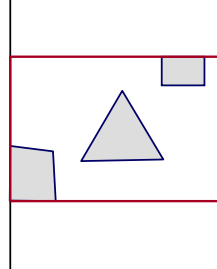
- Clip to each window boundary one at a time



Sutherland Hodgeman Clipping



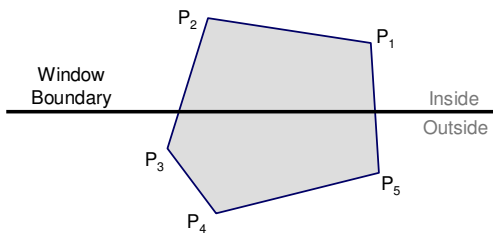
- Clip to each window boundary one at a time



Clipping to a Boundary



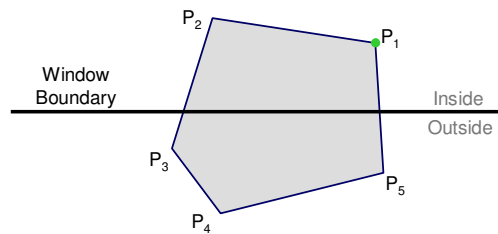
- Do inside test for each point in sequence, Insert new points when cross window boundary, Remove points outside window boundary



Clipping to a Boundary



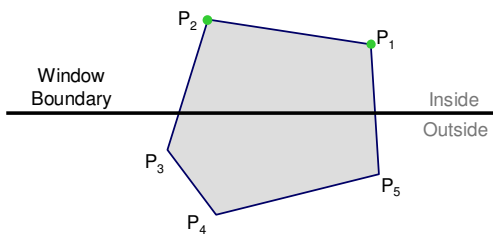
- Do inside test for each point in sequence, Insert new points when cross window boundary, Remove points outside window boundary



Clipping to a Boundary



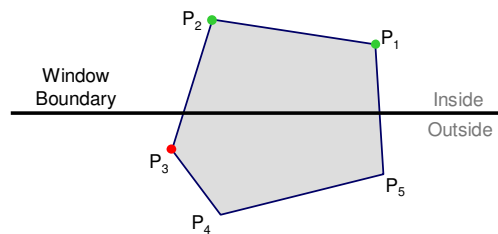
- Do inside test for each point in sequence, Insert new points when cross window boundary, Remove points outside window boundary



Clipping to a Boundary



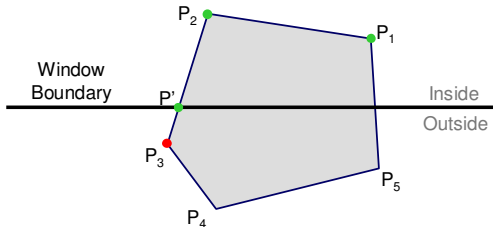
- Do inside test for each point in sequence, Insert new points when cross window boundary, Remove points outside window boundary



Clipping to a Boundary



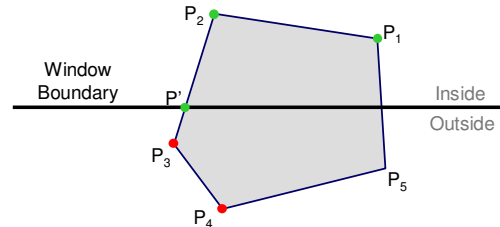
- Do inside test for each point in sequence, Insert new points when cross window boundary, Remove points outside window boundary



Clipping to a Boundary



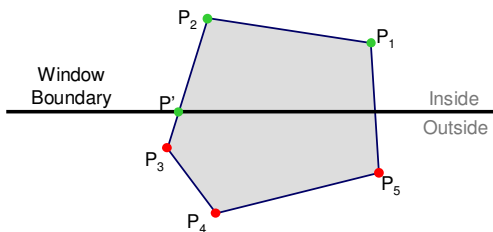
- Do inside test for each point in sequence, Insert new points when cross window boundary, Remove points outside window boundary



Clipping to a Boundary



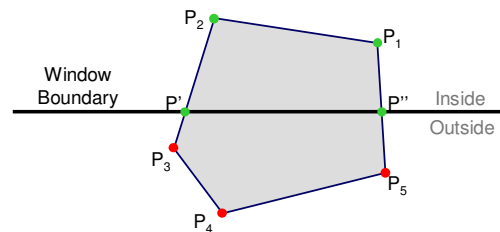
- Do inside test for each point in sequence, Insert new points when cross window boundary, Remove points outside window boundary



Clipping to a Boundary



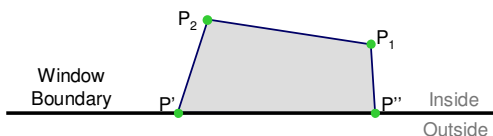
- Do inside test for each point in sequence, Insert new points when cross window boundary, Remove points outside window boundary



Clipping to a Boundary



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



2D Rendering Pipeline



3D Primitives

2D Primitives

Clipping

Clip portions of geometric primitives residing outside the window

Viewport Transformation

Transform the clipped primitives from screen to image coordinates

Scan Conversion

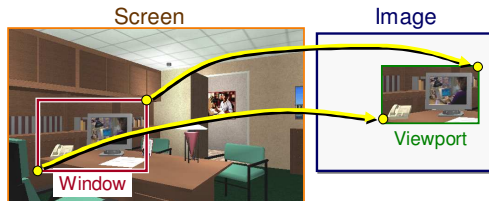
Fill pixels representing primitives in screen coordinates

Image

Viewport Transformation



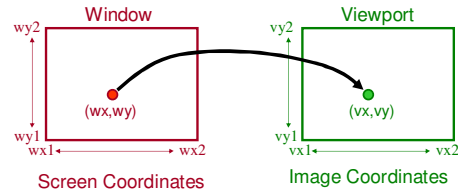
- Transform 2D geometric primitives from screen coordinate system (normalized device coordinates) to image coordinate system (pixels)



Viewport Transformation



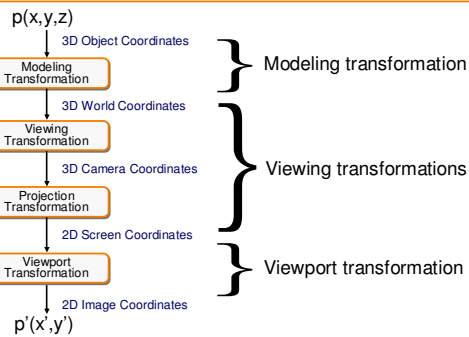
- Window-to-viewport mapping



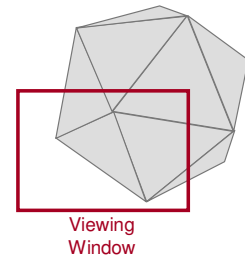
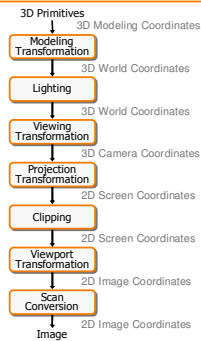
$$vx = vx1 + (wx - wx1) * (vx2 - vx1) / (wx2 - wx1);$$

$$vy = vy1 + (wy - wy1) * (vy2 - vy1) / (wy2 - wy1);$$

Summary of Transformations



Summary



Next Time

