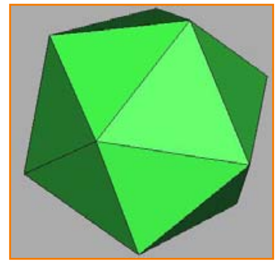
3D Polygon Rendering Pipeline

Adam Finkelstein
Princeton University
COS 426, Spring 2003

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

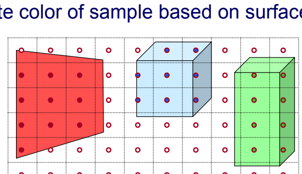
Quake II
(Id Software)

3D Polygon Rendering

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

Ray Casting Revisited

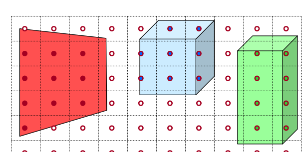
- For each sample …
  - Construct ray from eye position through view plane
  - Find first surface intersected by ray through pixel
  - Compute color of sample based on surface radiance



More efficient algorithms utilize spatial coherence!

3D Polygon Rendering

- What steps are necessary to utilize spatial coherence while drawing these polygons into a 2D image?



3D Rendering Pipeline (direct illumination)

3D Geometric Primitives
↓
Modeling Transformation
↓
Lighting
↓
Viewing Transformation
↓
Projection Transformation
↓
Clipping
↓
Scan Conversion
↓
Image

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

Example: OpenGL

Modeling Transformation
↓
Lighting
↓
Viewing Transformation
↓
Projection Transformation
↓
Clipping
↓
Scan Conversion
↓
Image

```
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 Geometric Primitives
↓
Modeling Transformation — Transform into 3D world coordinate system
↓
Lighting
↓
Viewing Transformation
↓
Projection Transformation
↓
Clipping
↓
Scan Conversion
↓
Image

3D Rendering Pipeline (for direct illumination)

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

3D Rendering Pipeline (for direct illumination)

3D Geometric 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
↓
Scan Conversion
↓
Image

3D Rendering Pipeline (for direct illumination)

3D Geometric 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
↓
Scan Conversion
↓
Image

3D Rendering Pipeline (for direct illumination)

3D Geometric 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
↓
Scan Conversion
↓
Image

3D Rendering Pipeline (for direct illumination)

3D Geometric 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
↓
Scan Conversion — Draw pixels (includes texturing, hidden surface, ...)
↓
Image

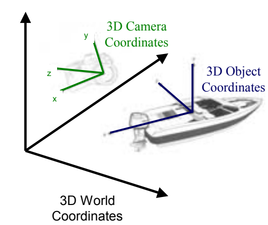
Transformations

3D Geometric 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
↓
Scan Conversion — Draw pixels (includes texturing, hidden surface, etc.)
↓
Image

Transformations

p(x,y,z)
↓ 3D Object Coordinates
Modeling Transformation
↓ 3D World Coordinates
Viewing Transformation
↓ 3D Camera Coordinates
Projection Transformation
↓ 2D Screen Coordinates
Window-to-Viewport Transformation
↓ 2D Image Coordinates
p'(x',y')

Transformations map points from one coordinate system to another



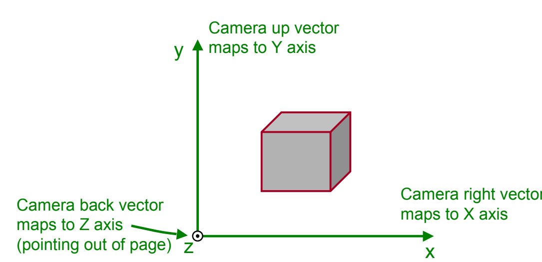
Viewing Transformations

p(x,y,z)
↓ 3D Object Coordinates
Modeling Transformation
↓ 3D World Coordinates
Viewing Transformation
↓ 3D Camera Coordinates
Projection Transformation
↓ 2D Screen Coordinates
Window-to-Viewport Transformation
↓ 2D Image Coordinates
p'(x',y')

} Viewing Transformations

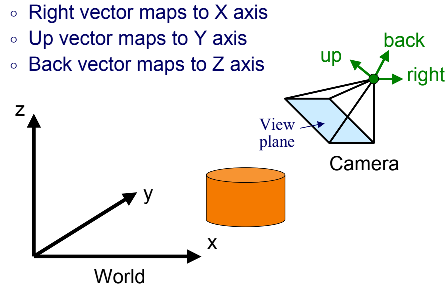
Camera Coordinates

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



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



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

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

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

?

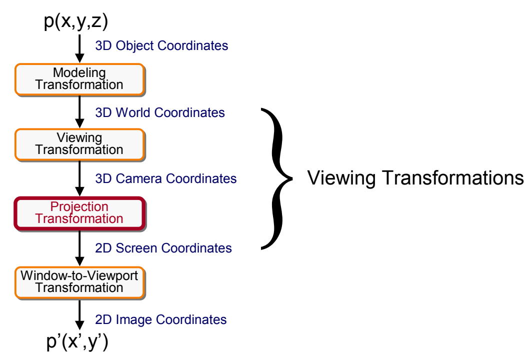
Finding the Viewing Transformation

- 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

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

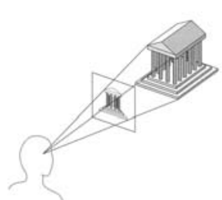
- This matrix is T^{-1} so we invert it to get T … easy!

Viewing Transformations

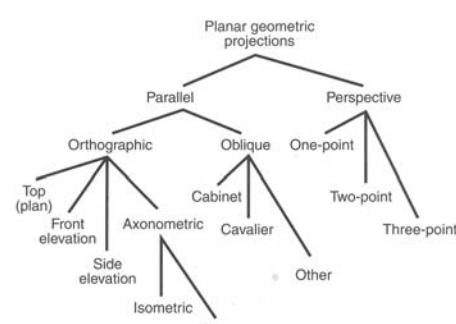


Projection

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

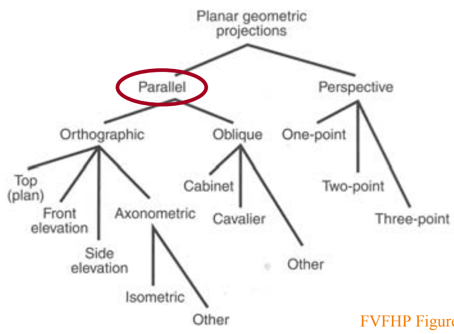


Taxonomy of Projections



FVFHP Figure 6.10

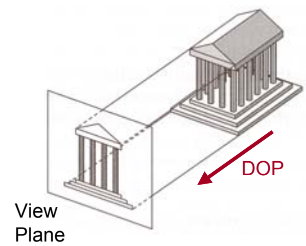
Taxonomy of Projections



FVFHP Figure 6.10

Parallel Projection

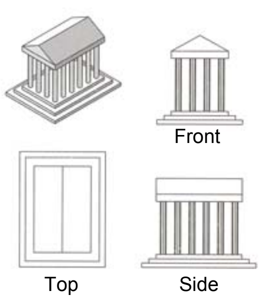
- Center of projection is at infinity
  - Direction of projection (DOP) same for all points



Angel Figure 5.4

Orthographic Projections

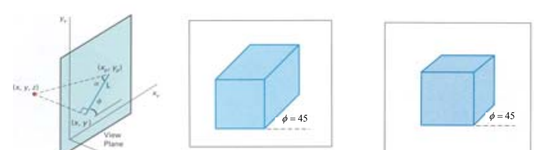
- DOP perpendicular to view plane



Angel Figure 5.5

Oblique Projections

- DOP not perpendicular to view plane

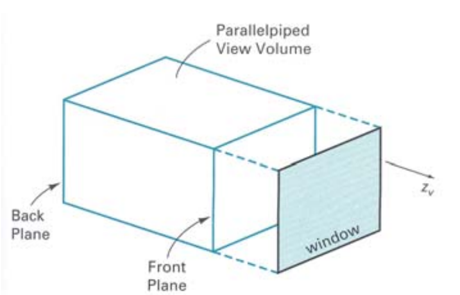


Cavalier (DOP α = 45°)

Cabinet (DOP α = 63.4°)

H&B Figure 12.24

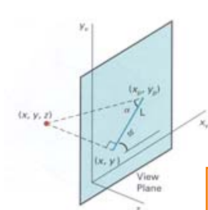
Parallel Projection View Volume



H&B Figure 12.30

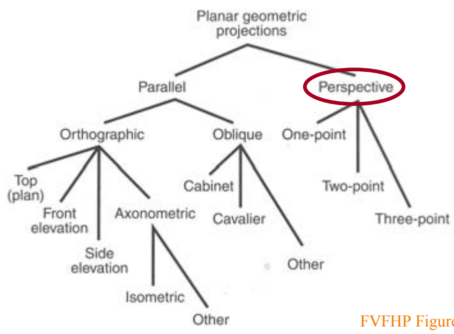
Parallel Projection Matrix

- General parallel projection transformation:



$$\begin{bmatrix} x_s \\ y_s \\ z_s \\ w_s \end{bmatrix} = \begin{bmatrix} 1 & 0 & L_1 \cos\phi & 0 \\ 0 & 1 & L_1 \sin\phi & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix}$$

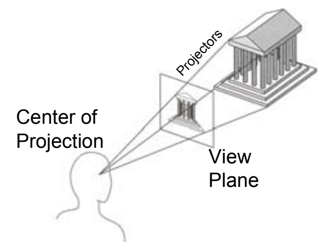
Taxonomy of Projections



FVFHP Figure 6.10

Perspective Projection

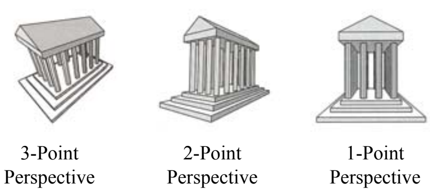
- Map points onto “view plane” along “projectors” emanating from “center of projection” (COP)



Angel Figure 5.9

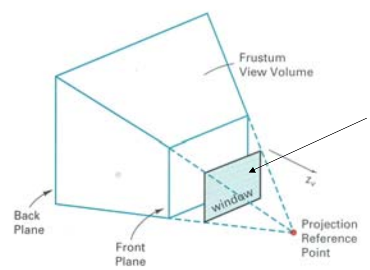
Perspective Projection

- How many vanishing points?



Angel Figure 5.10

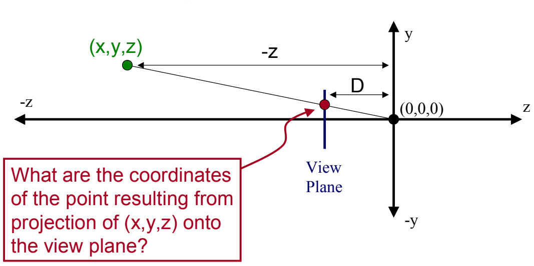
Perspective Projection View Volume



H&B Figure 12.30

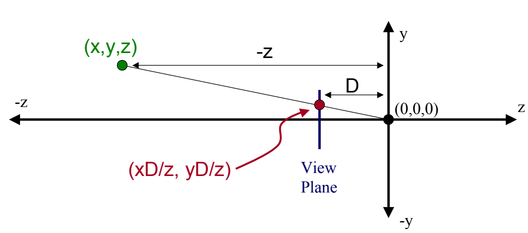
Perspective Projection

- 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?

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

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

Perspective Projection Matrix

- 4x4 matrix representation?

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

$$x' = x_c$$
$$y' = y_c$$
$$z' = z_c$$
$$w' = z_c / D$$

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

Perspective Projection Matrix

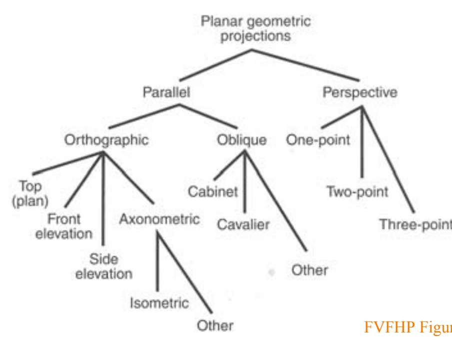
- 4x4 matrix representation?

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

$$x' = x_c$$
$$y' = y_c$$
$$z' = z_c$$
$$w' = z_c / D$$

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

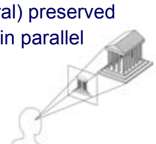
Taxonomy of Projections



FVFHP Figure 6.10

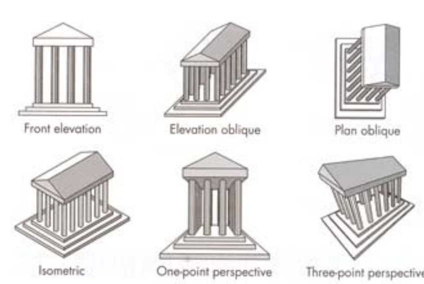
Perspective vs. Parallel

- 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

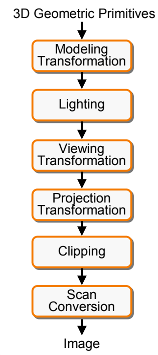


Angel Figure 5.3

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

What's next?



Transform into 3D world coordinate system

Illuminate according to lighting and reflectance

Transform into 3D camera coordinate system

Transform into 2D camera coordinate system

Clip primitives outside camera's view

Draw pixels (includes texturing, hidden surface, etc.)