Projections and Cameras

Intro to Projection

• Computer graphics treats all projections the same and implements them with a single pipeline
• Classical viewing developed different techniques for drawing different types of projections
• Fundamental distinction is between parallel and perspective viewing (even though mathematically parallel viewing is the limit of perspective viewing)
Perspective Projection

Parallel Projection
Orthographic Projection

Projectors are orthogonal to projection surface

Orthogonal Projection

• Set $z = 0$
• Equivalent homogeneous coordinate transformation:

$$
\begin{bmatrix}
px' \\
py' \\
0 \\
1
\end{bmatrix} =
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
px \\
py \\
pz \\
1
\end{bmatrix}
$$
Multi-view Orthographic Projection

- Projection plane parallel to principal face
- Usually form front, top, side views

Isometric (not orthographic)

In CAD and architecture, we often display three multiviews plus isometric.

Oblique Projections
Oblique Projections

Direction of projection \( \text{dop} = (\text{dop}_x, \text{dop}_y, \text{dop}_z) \)

Let's say we project onto \( z = 0 \) plane.

Oblique Projections

Direction of projection \( \text{dop} = (\text{dop}_x, \text{dop}_y, \text{dop}_z) \)

\[
\begin{bmatrix}
px' \\
py' \\
0 \\
1
\end{bmatrix} = \begin{bmatrix}
1 & 0 & -\text{dop}_x/\text{dop}_z & 0 \\
0 & 1 & -\text{dop}_y/\text{dop}_z & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}\begin{bmatrix}
px \\
py \\
pz
\end{bmatrix}
\]

Note: \( \tan \theta = \text{dop}_z/\text{dop}_x \)
**Perspective Projection**

- Assume COP (center of projection) is at the origin and
- Projection plane $z = d$, $d < 0$,

**Simple Perspective**

- Assume COP (center of projection) is at the origin and
- Projection plane $z = d$, $d < 0$, then:
**Perspective Equations**

Consider top and side views

\[
\begin{align*}
\frac{x_p}{z/d} &= \frac{x}{z/d} & y_p &= \frac{y}{z/d} & z_p &= d
\end{align*}
\]

**Homogeneous Coordinate Form**

consider \( \mathbf{q} = \mathbf{Mp} \) where

\[
\mathbf{M} = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 1 / d & 0
\end{bmatrix}
\]

\[
\mathbf{q} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad \Rightarrow \quad \mathbf{p} = \begin{bmatrix} x \\ y \\ z \\ z / d \end{bmatrix}
\]
**Perspective Division**

- But recall when $w \neq 1$, we divide by $w$ to return from homogeneous coordinates
- This *perspective division* yields

$$x_p = \frac{x}{z/d} \quad y_p = \frac{y}{z/d} \quad z_p = d$$

the desired perspective equations

**Simple Perspective**

Thus for a simple perspective with the COP at $(0,0,0)$, the image plane at $z = d$

$$\begin{bmatrix}
px' \\
p'y' \\
d \\
1
\end{bmatrix} = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 1/d & 0
\end{bmatrix}\begin{bmatrix}
px \\
py \\
pz \\
1
\end{bmatrix}$$
Viewing Volume

Orthographic View Volume

Perspective View Volume

near and far measured from camera
Clipping

- Removing the unseen geometry
- Direct (brute-force) solution - solve simultaneous equations for intersections of lines/edges at window edges

A point or vertex is visible if
\[ x_{\text{left}} < x < x_{\text{right}} \]
and
\[ y_{\text{bot}} < y < y_{\text{top}} \]

Clipping lines

Pipeline, clip each edge of the window separately:
Clip the vertices that are outside of the window and create new vertices at window border.

Result is still a single polygon but may have more vertices and an odd shape.

**Bounding box** - surrounds each polygon.
Clipping polygons

Trivially reject or accept if the bounding box falls completely inside or outside.

Normalization

- Rather than derive a different projection matrix for each type of projection, we can convert all projections to basic projections with a default *view volume*
- This strategy allows us to use standard transformations in the pipeline and makes for *efficient clipping*
- Volume known as *Canonical view volume*
Orthogonal Normalization

normalization \Rightarrow \text{find transformation to convert specified clipping volume to default}

\begin{align*}
\text{(left,bottom,near)} & \quad \rightarrow \quad \text{Canonical Volume} \\
(1,1,1) & \quad \quad \quad (-1,-1,-1)
\end{align*}

- Two steps
  - Move center to origin
    \[ T \left(-\frac{(left+right)}{2}, -\frac{(bottom+top)}{2}, -\frac{(near+far)}{2}\right) \]
  - Scale to have sides of length 2
    \[ S \left(\frac{2}{(right-left)}, \frac{2}{(top-bottom)}, \frac{2}{(near-far)}\right) \]
Orthogonal Normalization

\[ ST = \begin{bmatrix} \frac{2}{\text{right} - \text{left}} & 0 & 0 & -\frac{\text{right} + \text{left}}{\text{right} - \text{left}} \\ 0 & \frac{2}{\text{top} - \text{bottom}} & 0 & -\frac{\text{top} + \text{bottom}}{\text{top} - \text{bottom}} \\ 0 & 0 & \frac{2}{\text{near} - \text{far}} & -\frac{\text{far} + \text{near}}{\text{far} - \text{near}} \\ 0 & 0 & 0 & 1 \end{bmatrix} \]

Combined with orthogonal projection yields:

\[ P = M_{\text{orth}}ST \]

Perspective Projection

• How to normalize the frustrum of the perspective view

• Want to make a canonical volume to clip against, just like in the parallel case
**Canonical Perspective**

Consider a simple perspective with the COP at the origin, the far plane at $z = 1$, and a 90 degree field of view determined by the planes $x = \pm z$,

$y = \pm z$

**Effect on Clipping**

- Extension to Cohen-Sutherland Clipping

Bit 1 – above volume $y > z$
Bit 2 – below volume $y < -z$
Bit 3 – right of volume $x > z$
Bit 4 – left of volume $x < -z$
Bit 5 – behind volume $z > 1$
Bit 6 – in front of volume $z < z_{\text{min}}$
Pipeline View

- Can also add shear for asymmetric views for perspective

Camera coordinates

To set camera, we need one more step...
Defining and moving the camera

default camera          wanted camera

Moving the Camera

• We can move the camera to any desired position by a sequence of rotations and translations

• Example: side view
  - Rotate after moving camera away from origin

  - Camera View matrix:
  \[ V_{\text{cam}} = R_{\text{cam}} T_{\text{cam}} \]
Whole camera transform

• We can put these together as:

\[ T_{total} = M_{projection} \cdot V_{cam} \]

• But where does camera transform come from?
  - Prescribed
  - Generated automatically – for games

Camera movement

• Should mimic a real camera
  - Maintain up-vector
  - Slow turns
  - Avoid accelerations
Camera movement

• Visibility Planning for Camera Control

Oskan et al 2009
Camera control – Oskan et al

• Precomputation 1) Free-space

Space Sphere → Overlapping “portals” → Roadmap

Camera control – Oskan et al

• Precomputation 2) Visibility
Camera control – Oskan et al

- Run-time computation

Initial Conditions → Construct Overlap Path → Refine & Subsample

Camera control – Oskan et al

Initial Conditions → Construct Overlap Path → Refine & Subsample