Projection Matrices for Viewing and Clipping

Objectives

• Derive projection matrix for perspective projections
• Introduce camera frame
• Clipping
• Introduce projection normalization
• Camera's in GL
Computer Viewing

- Need to build transformation that defines the projection plane based on the chosen projection

Orthogonal Projection
Orthogonal Projection

• Set $z = 0$

• Equivalent homogeneous coordinate transformation:

\[
\begin{bmatrix}
px' \\
py' \\
0 \\
1
\end{bmatrix} =
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
px \\
py \\
pz \\
1
\end{bmatrix}
\]
Oblique Projections

Direction of projection dop = (dop_x, dop_y, dop_z)

Let’s say we project onto z = 0 plane
Oblique Projections

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

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

note: \( \tan \theta = \text{dop}_z/\text{dop}_x \)

Shear Matrix

• Equivalent to pulling faces in opposite directions
Shear Matrix

Consider simple shear along \( x \) axis

\[
x' = x + y \, sh_y \\
y' = y \\
z' = z
\]

\[
H = \begin{bmatrix}
1 & sh_y & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

Oblique Projection

xy shear (z value stays unchanged)

\[
H = \begin{bmatrix}
1 & 0 & -dop_x/dop_z & 0 \\
0 & 1 & -dop_y/dop_z & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

Projection matrix:

\[
M_{\text{projection}} = M_{\text{orth}} \, H
\]
Consider a simple perspective with the COP at 
(0,0,0), the image plane at \( z = d \)

Projection matrix:
\[
M_{\text{persp}} = \begin{bmatrix}
1 & 0 & 0 & 0 & px \\
0 & 1 & 0 & 0 & py \\
0 & 0 & 1 & 0 & pz \\
0 & 0 & 1/d & 0 & 1
\end{bmatrix}
\]

Projection matrix:
\[
M_{\text{projection}} = M_{\text{persp}}
\]
Computer Viewing

There are three aspects of the viewing process:

- Selecting a lens
  Setting the projection matrix

- Positioning the camera
  Setting the view-orientation matrix

- Clipping
  Setting the view volume

Defining and moving the camera

default camera
(from transform)  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, $V_{cam}$
      $$V_{cam} = R_{cam} T_{cam}$$

Whole camera transform

• We can put these together as:

  $$T_{total} = M_{projection} \ V_{cam}$$

• Note, while $M_{projection}$ is different for persp and parallel, they are both projections...
Camera coordinates

Camera coordinate frame relative to global from user input

OpenGL Orthogonal Viewing

```c
glOrtho(xmin, xmax, ymin, ymax, near, far)
glOrtho(left, right, bottom, top, near, far)
```

Near and far measured from camera
OpenGL Perspective

\[ \text{glFrustum}(x_{\text{min}}, x_{\text{max}}, y_{\text{min}}, y_{\text{max}}, \text{near}, \text{far}) \]

Using Field of View

- With \text{glFrustum} it is often difficult to get the desired view
- \text{gluPerspective}(\text{fov}, \text{aspect}, \text{near}, \text{far}) often provides a better interface
Clipping

Clipping remove unseen geometry

Direct solution:

Solve for intersections (simultaneous equations) between lines/edges at window sides

A point or vertex is visible if

\[
\text{left} < x < \text{right}
\]
and

\[
\text{bottom} < y < \text{top}
\]

Clipping lines

Pipeline, clip each edge of the window separately:
Clipping polygons

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.
Clipping polygons

Accelerations:

Bounding box - surrounds each polygon

Trivially reject or accept if the bounding box falls completely inside or outside.
Cohen-Sutherland Algorithm

- Region Checks: Trivially reject or accept for clipping

- Each vertex is assigned an 4-bit outcode
  - bit 1 - sign of (top - y), point is above window
  - bit 2 - sign of (y-bottom), point is below window
  - bit 3 - sign of (right-x), point is right of window
  - bit 4 - sign of (x - left), point is left of window

A line can be trivially accepted if both endpoints have an outcode of 0000.

A line can be trivially rejected if any of the same two bits in the outcodes are both equal to 1 (both endpoints are left,right, above, below the window)
Clipping 3D

Adds far and near clipping planes for 3D viewing volume

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 ⇒ find transformation to convert specified clipping volume to default

\[ \begin{align*}
\text{(left, bottom, near)} & \quad \text{to} \quad \text{(right, top, far)} \\
(1,1,1) & \quad \text{canonical volume}
\end{align*} \]

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

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

Oblique with Normalization

To use the same projection, we shear but don't project

\[ \begin{bmatrix} px' \\ py' \\ pz' \end{bmatrix} = \begin{bmatrix} 1 & 0 & -\text{dopx}/\text{dopz} & 0 \\ 0 & 1 & -\text{dopy}/\text{dopz} & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} px \\ py \\ pz \end{bmatrix} \]
Oblique with Normalization

$xy$ shear ($z$ values unchanged)

$$H = \begin{bmatrix} 1 & 0 & -\text{dopx/dopz} & 0 \\ 0 & 1 & -\text{dopy/dopz} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Projection matrix

$$P = M_{\text{orth}} H$$

Normalize as:

$$P = M_{\text{orth}} STH$$

---

Equivalency
Effect on Clipping

- The projection matrix $P = STH$ transforms the original clipping volume to the default clipping volume.

Perspective Projection

- How do we 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 $z < z_{min}$
**Pipeline View**

- Camera view transformation
- Transform projection to canonical view
- Clipping against canonical
- Projection to 3D → 2D

- Must add shear for asymmetric views, slightly different for persp vs. parallel

---

**Why do we do it this way?**

- Normalization allows for a single pipeline to be used with all (perspective and parallel) desired views
- Keep in four dimensional homogeneous coordinates as long as possible to retain three-dimensional information needed for hidden-surface removal and shading
- Standardize clipping