CSE 167:
Introduction to Computer Graphics
Lecture #6: Projection

Jürgen P. Schulze, Ph.D.
University of California, San Diego
Fall Quarter 2020
Projection
Projection

- **Goal:**
  Given 3D points (vertices) in camera coordinates, determine corresponding image coordinates

- **Transforming 3D points into 2D is called Projection**

- **Typically one of two types of projection is used:**
  - Orthographic Projection (=Parallel Projection)
  - **Perspective Projection**: most commonly used
Perspective Projection

- Most common for computer graphics
- Simplified model of human eye, or camera lens (*pinhole camera*)

- Things farther away appear to be smaller
- Discovery attributed to Filippo Brunelleschi (Italian architect) in the early 1400’s
Perspective Projection

- Project along rays that converge in center of projection

3D scene

2D image plane

Center of projection
Perspective Projection

Parallel lines are no longer parallel, converge in one point

Earliest example: La Trinità (1427) by Masaccio
Perspective Projection

From law of ratios in similar triangles follows:

\[
\frac{y'}{d} = \frac{y}{D} \rightarrow y' = \frac{yd}{D}
\]

Similarly:

\[
x' = \frac{xd}{D}
\]

By definition: \(z' = -d\)

- We can express this using homogeneous coordinates and 4x4 matrices as follows
Perspective Projection

\[ x' = \frac{xd}{D} \]
\[ y' = \frac{yd}{D} \]
\[ z' = -d \]

Projection matrix

\[
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 1/d & 0 & 0 \\
0 & 0 & 0 & 1 & 0
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z \\
x/d \\
y/d \\
z/d
\end{bmatrix}
= \begin{bmatrix}
x \\
y \\
z \\
-xd/z \\
-yd/z \\
-z/d
\end{bmatrix}
\begin{bmatrix}
1 \\
1 \\
1 \\
1 \\
1 \\
1
\end{bmatrix}
\]

Homogeneous division
Perspective Projection

Using projection matrix, homogeneous division seems more complicated than just multiplying all coordinates by \(-d/z\), so why do it?

It will allow us to:
- Handle different types of projections in a unified way
- Define arbitrary view volumes

Projection matrix \(P\):

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & -1/d & 0
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z \\
1
\end{bmatrix}
= 
\begin{bmatrix}
x \\
y \\
z \\
1
\end{bmatrix}
= 
\begin{bmatrix}
-xd/z \\
-yd/z \\
-z/d \\
1
\end{bmatrix}
\]
Topics

- View Volumes
- Vertex Transformation
- Rendering Pipeline
- Culling
View Volume

- View volume = 3D volume seen by camera

*Camera coordinates*

*World coordinates*
Projection Matrix

Camera coordinates

Projection matrix

Canonical view volume

Viewport transformation

Image space (pixel coordinates)
Perspective View Volume

**General view volume**

- Defined by 6 parameters, in camera coordinates
  - Left, right, top, bottom boundaries
  - Near, far clipping planes
- Clipping planes to avoid numerical problems
  - Divide by zero (multiplying all coordinates by \(-\frac{d}{z}\))
  - Low precision for distant objects
- Usually symmetric, i.e., left=-right, top=-bottom
Perspective View Volume

Symmetrical view volume

- Only 4 parameters
  - Vertical field of view (FOV)
  - Image aspect ratio (width/height)
  - Near, far clipping planes

- **Demo link**

\[
\tan\left(\frac{FOV}{2}\right) = \frac{\text{top}}{\text{near}}
\]

\[
\text{aspect ratio} = \frac{\text{right} - \text{left}}{\text{top} - \text{bottom}} = \frac{\text{right}}{\text{top}}
\]
Perspective View Volume

Rule of thumb to calculate projection matrix:
1. Convert the view-frustum to the simple symmetric projection frustum
2. Transform the simple frustum to the canonical view frustum

Ref: http://info.ee.surrey.ac.uk/Teaching/Courses/eem.cgi/lectures_pdf/lecture3.pdf
Perspective Projection Matrix

- General view frustum with 6 parameters

\[
P_{\text{persp}}(\text{left}, \text{right}, \text{top}, \text{bottom}, \text{near}, \text{far}) =
\]
\[
\begin{bmatrix}
\frac{2\text{near}}{\text{right-\text{left}}} & 0 & \frac{\text{right+\text{left}}}{\text{top-b\text{ottom}}} & 0 \\
0 & \frac{2\text{near}}{\text{top-bottom}} & \frac{\text{top+\text{bottom}}}{\text{top-bottom}} & 0 \\
0 & 0 & \frac{-((\text{far+\text{near}})}}{\text{far-\text{near}}} & -1 \\
0 & 0 & -\frac{2\text{far-\text{near}}}{\text{far-\text{near}}} & 0
\end{bmatrix}
\]
Perspective Projection Matrix

- Symmetrical view frustum with field of view, aspect ratio, near and far clip planes

\[
P_{\text{persp}}(\text{FOV}, \text{aspect}, \text{near}, \text{far}) = \begin{bmatrix}
\frac{1}{\text{aspect} \cdot \tan(\text{FOV} / 2)} & 0 & 0 & 0 \\
0 & \frac{1}{\tan(\text{FOV} / 2)} & 0 & 0 \\
0 & 0 & \frac{\text{near} + \text{far}}{\text{near} - \text{far}} & \frac{2 \cdot \text{near} \cdot \text{far}}{\text{near} - \text{far}} \\
0 & 0 & 0 & -1 \\
0 & 0 & 0 & 0
\end{bmatrix}
\]

Ref: https://www.youtube.com/watch?v=ohksz3A00fk&t=79s
Projection Matrix

- How to determine if a matrix is projection matrix?
Canonical View Volume

- Goal: create projection matrix so that
  - User defined view volume is transformed into canonical view volume: cube \([-1,1] \times [-1,1] \times [-1,1]\)
  - Multiplying corner vertices of view volume by projection matrix and performing homogeneous divide yields corners of canonical view volume

- Perspective and orthographic projection are treated the same way

- Canonical view volume is last stage in which coordinates are in 3D
  - Next step is projection to 2D frame buffer
Canonical View Volume

- Summary so far in a demo
Viewport Transformation

- After applying projection matrix, scene points are in *normalized viewing coordinates*
  - Per definition within range $[-1..1] \times [-1..1] \times [-1..1]$
- Next is projection from 3D to 2D (not reversible)
- Normalized viewing coordinates can be mapped to image (=pixel=frame buffer) coordinates
  - Range depends on window (view port) size: $[x_0...x_1] \times [y_0...y_1]$
- Scale and translation required:

\[
D(x_0, x_1, y_0, y_1) = \begin{bmatrix}
\frac{(x_1 - x_0)}{2} & 0 & 0 & \frac{(x_0 + x_1)}{2} \\
0 & \frac{(y_1 - y_0)}{2} & 0 & \frac{(y_0 + y_1)}{2} \\
0 & 0 & 1/2 & 1/2 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]
Lecture Overview

- View Volumes
- Vertex Transformation
- Rendering Pipeline
- Culling
Complete Vertex Transformation

- Mapping a 3D point in object coordinates to pixel coordinates:

\[
p' = DPC^{-1}Mp
\]

- **M**: Object-to-world matrix
- **C**: camera matrix
- **P**: projection matrix
- **D**: viewport matrix
Complete Vertex Transformation

- Mapping a 3D point in object coordinates to pixel coordinates:

\[ p' = DPC^{-1}Mp \]

- **M**: Object-to-world matrix
- **C**: camera matrix
- **P**: projection matrix
- **D**: viewport matrix
Complete Vertex Transformation

- Mapping a 3D point in object coordinates to pixel coordinates:

\[
p' = DPC^{-1}Mp
\]

- **M**: Object-to-world matrix
- **C**: camera matrix
- **P**: projection matrix
- **D**: viewport matrix
Complete Vertex Transformation

- Mapping a 3D point in object coordinates to pixel coordinates:

\[ p' = DPC^{-1}Mp \]

- **M**: Object-to-world matrix
- **C**: camera matrix
- **P**: projection matrix
- **D**: viewport matrix
Complete Vertex Transformation

- Mapping a 3D point in object coordinates to pixel coordinates:
  \[ p' = DPC^{-1}Mp \]
  - **M**: Object-to-world matrix
  - **C**: camera matrix
  - **P**: projection matrix
  - **D**: viewport matrix

Spaces:
- Object space
- World space
- Camera space
- Canonical view volume
- Image space
Complete Vertex Transformation

- Mapping a 3D point in object coordinates to pixel coordinates:
  \[ p' = DPC^{-1}Mp \]

  \[ p' = \begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} \quad \text{Pixel coordinates:} \quad \begin{bmatrix} x'/w' \\ y'/w' \end{bmatrix} \]

- **M**: Object-to-world matrix
- **C**: camera matrix
- **P**: projection matrix
- **D**: viewport matrix
Complete Vertex Transformation in OpenGL

- Mapping a 3D point in object coordinates to pixel coordinates:

\[
p' = DPC^{-1}Mp
\]

- **M**: Object-to-world matrix
- **C**: camera matrix
- **P**: projection matrix
- **D**: viewport matrix
Complete Vertex Transformation in OpenGL

- ModelView matrix: $C^{-1}M$
  - Defined by the programmer.
  - Think of the ModelView matrix as where you stand with the camera and the direction you point it.

- Projection matrix: $P$
  - Think of the projection matrix as describing the attributes of your camera, such as field of view, focal length, etc.

- Viewport, $D$
  - Specify via `glViewport(x, y, width, height)`
Vertex Shader Code

```glsl
layout (location = 0) in vec3 position;
// ...

uniform mat4 projection;
uniform mat4 view;
uniform mat4 model;

void main() {
    gl_Position = projection * view * model * vec4(position, 1.0);
    // ...
}
```
The Complete Vertex Transformation

- **Model Matrix**
- **Camera Matrix**
- **Projection Matrix**
- **Viewport Matrix**

**Object Coordinates**

**World Coordinates**

**Camera Coordinates**

**Canonical View Volume Coordinates**

**Window Coordinates**

- `glm::lookAt` e.g. `glm::perspective`
- `glViewport`