Viewing and Projections

Don Fussell
Computer Science Department
The University of Texas at Austin
A Simplified Graphics Pipeline

Application

Vertex batching & assembly

Triangle assembly

Triangle clipping

NDC to window space

Triangle rasterization

Fragment shading

Depth testing

Color update

Depth buffer

Framebuffer

University of Texas at Austin    CS354 - Computer Graphics    Don Fussell
A few more steps expanded

1. Application
2. Vertex batching & assembly
3. Vertex transformation
4. Lighting
5. Texture coordinate generation
6. View frustum clipping
7. Perspective divide
8. NDC to window space
9. Triangle assembly
10. Back face culling
11. Triangle rasterization
12. Fragment shading
13. Depth testing
14. Depth buffer
15. Color update
16. Framebuffer
Conceptual Vertex Transformation

glVertex* API commands

object-space coordinates

(Modelview matrix)

eye-space coordinates

User-defined clip planes

Projection matrix

clip-space coordinates

View-frustum clip planes

cropped eye-space coordinates

Perspective division

Viewport + Depth Range transformation

window-space coordinates

to primitive rasterization

University of Texas at Austin    CS354 - Computer Graphics    Don Fussell
Eye Coordinates (not NDC)

-Z direction
“looking into the screen”

+Z direction
“poking out of the screen”
Planar Geometric Projections

- Standard projections project onto a plane
- Projectors are lines that either
  - converge at a center of projection
  - are parallel
- Such projections preserve lines
  - but not necessarily angles
- Nonplanar projections are needed for applications such as map construction
Classical Projections

Front elevation

Elevation oblique

Plan oblique

Isometric

One-point perspective

Three-point perspective
Perspective vs Parallel

- Computer graphics treats all projections the same and implements them with a single pipeline.
- Classical viewing developed different techniques for drawing each type of projection.
- Fundamental distinction is between parallel and perspective viewing even though mathematically parallel viewing is the limit of perspective viewing.
Taxonomy of Projections

- Planar geometric projections
  - Parallel
    - Multiview
      - Orthographic
    - Axonometric
    - Isometric
  - Perspective
    - 1 point
    - 2 point
    - 3 point
    - Dimetric
    - Trimetric
Parallel Projection

Object

Projector

Projection plane

DOP

University of Texas at Austin  CS354 - Computer Graphics  Don Fussell
Perspective Projection

Projector

Object

Projection plane

COP
Orthographic Projection

Projectors are orthogonal to projection surface
Multiview Orthographic Projection

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

isometric (not multiview orthographic view)

in CAD and architecture, we often display three multiviews plus isometric
Advantages and Disadvantages

- Preserves both distances and angles
  - Shapes preserved
  - Can be used for measurements
    - Building plans
    - Manuals
- Cannot see what object really looks like because many surfaces hidden from view
  - Often we add the isometric
The default projection in the eye (camera) frame is orthogonal.

For points within the default view volume:

\[ x_p = x \]
\[ y_p = y \]
\[ z_p = 0 \]

Most graphics systems use view normalization.

All other views are converted to the default view by transformations that determine the projection matrix.

Allows use of the same pipeline for all views.
Default projection is orthographic
Orthogonal Normalization

$$\text{glOrtho}(\text{left}, \text{right}, \text{bottom}, \text{top}, \text{near}, \text{far})$$

Normalization $\Rightarrow$ find transformation to convert specified clipping volume to default

Diagram: Orthographic projection from $(\text{left}, \text{bottom}, \text{near})$ to $(1, 1, 1)$.
OpenGL Orthogonal Viewing

`glOrtho(left, right, bottom, top, near, far)`

near and far measured from camera
Homogeneous Representation

default orthographic projection

\[ x_p = x \]
\[ y_p = y \]
\[ z_p = 0 \]
\[ w_p = 1 \]

\[ p_p = Mp \]

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

In practice, we can let \( M = I \) and set the \( z \) term to zero later
Orthographic Eye to NDC

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

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

**Prototype**

- `glOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far)`

**Post-concatenates an orthographic matrix**

\[
\begin{bmatrix}
\frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\
0 & \frac{2}{t-b} & 0 & -\frac{r-l}{t+b} \\
0 & 0 & -\frac{2}{f-n} & -\frac{f+n}{f-n} \\
0 & 0 & 0 & 1
\end{bmatrix}
\]
glOrtho Example

Consider

- `glLoadIdentity();`
- `glOrtho(-20, 30, 10, 60, 15, -25)`
  - `left=-20, right=30, bottom=10, top=50, near=15, far=-25`

Matrix

\[
\begin{bmatrix}
\frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\
0 & 2 & 0 & -\frac{r+l}{t+b} \\
0 & \frac{2}{t-b} & 0 & -\frac{r+l}{t-b} \\
0 & 0 & \frac{-2}{f-n} & -\frac{f+n}{f-n} \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

\[
\begin{bmatrix}
1 & 0 & 0 & -\frac{1}{5} \\
\frac{1}{25} & 0 & 0 & \frac{3}{2} \\
0 & \frac{1}{20} & 0 & \frac{1}{4} \\
0 & 0 & \frac{1}{20} & 1
\end{bmatrix}
\]
Axonometric Projections

Allow projection plane to move relative to object

classify by how many angles of a corner of a projected cube are the same

none: trimetric
two: dimetric
three: isometric
Types of Axonometric Projections

- Dimetric
- Trimetric
- Isometric
Advantages and Disadvantages

- Lines are scaled (*foreshortened*) but can find scaling factors
- Lines preserved but angles are not
  - Projection of a circle in a plane not parallel to the projection plane is an ellipse
- Can see three principal faces of a box-like object
- Some optical illusions possible
  - Parallel lines appear to diverge
- Does not look real because far objects are scaled the same as near objects
- Used in CAD applications
Oblique Projection

Arbitrary relationship between projectors and projection plane
Advantages and Disadvantages

- Can pick the angles to emphasize a particular face
  - Architecture: plan oblique, elevation oblique
- Angles in faces parallel to projection plane are preserved while we can still see “around” side

- In physical world, cannot create with simple camera; possible with bellows camera or special lens (architectural)
Perspective Projection

Projectors converge at center of projection
Vanishing Points

- Parallel lines (not parallel to the projection plan) on the object converge at a single point in the projection (the vanishing point)
- Drawing simple perspectives by hand uses these vanishing point(s)
Three-Point Perspective

- No principal face parallel to projection plane
- Three vanishing points for cube
Two-Point Perspective

- On principal direction parallel to projection plane
- Two vanishing points for cube
One-Point Perspective

- One principal face parallel to projection plane
- One vanishing point for cube
Perspective in Art History

Pietro Perugino, 1482
Perspective in Art History

Pietro Perugino, 1482

Vanishing point
Humanist Analysis of Perspective

[Albrecht Dürer, 1471]
Advantages and Disadvantages

- Objects further from viewer are projected smaller than the same sized objects closer to the viewer (*diminution*)
  - Looks realistic
- Equal distances along a line are not projected into equal distances (*nonuniform foreshortening*)
- Angles preserved only in planes parallel to the projection plane
- More difficult to construct by hand than parallel projections (but not more difficult by computer)
1-, 2-, and 3-point Perspective

- A 4x4 matrix can represent 1, 2, or 3 vanishing points
- As well as zero for orthographic views

1-point perspective  2-point perspective  3-point perspective
Simple Perspective

- Center of projection at the origin
- Projection plane $z = d$, $d < 0$
Consider top and side views

Perspective Equations

\[ x_p = \frac{x}{z/d} \quad y_p = \frac{y}{z/d} \quad z_p = d \]
Homogeneous Form

consider $q = Mp$ where

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix}$$

$$q = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \Rightarrow p = \begin{bmatrix} x \\ y \\ z \\ z/d \end{bmatrix}$$
OpenGL Perspective

\texttt{glFrustum(left, right, bottom, top, near, far)}
Consider a simple perspective with the COP at the origin, the near clipping plane at $z = -1$, and a 90 degree field of view determined by the planes $x = \pm z, y = \pm z$. 

\[
\begin{align*}
(1, 1, -1) \\
(-1, -1, -1)
\end{align*}
\]
Simple Eye to NDC

\[
N = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & \alpha & \beta \\
0 & 0 & -1 & 0
\end{bmatrix}
\]

after perspective division, the point \((x, y, z, 1)\) goes to

\[
x' = \frac{x}{z}
\]

\[
y' = \frac{y}{z}
\]

\[
z' = -(\alpha + \frac{\beta}{z})
\]

which projects orthogonally to the desired point regardless of \(\alpha\) and \(\beta\)
Picking $\alpha$ and $\beta$

If we pick

$$\alpha = \frac{\text{near} + \text{far}}{\text{far} - \text{near}}$$
$$\beta = \frac{2\text{near} \times \text{far}}{\text{near} - \text{far}}$$

the near plane is mapped to $z = -1$
the far plane is mapped to $z = 1$
and the sides are mapped to $x = \pm 1$, $y = \pm 1$

If we start from the simple eye frustum, we end up with the NDC clipping cube
Normalization Transformation

original clipping volume

z = -x

original object

z = -far

COP

z = -near

distorted object

projects correctly

new clipping volume

z = 1

x = -1

z = 1

x = 1

University of Texas at Austin    CS354 - Computer Graphics    Don Fussell
Frustum Transform

Prototype

\[ \text{glFrustum} \left( \text{GLdouble } \left[ \begin{array} {cccc} 2n & 0 & r + l & 0 \\ r - l & 0 & t + b & 0 \\ 0 & t - b & 0 & 0 \\ 0 & 0 & - (f + n) & -2fn \\ 0 & 0 & f - n & 0 \\ 0 & 0 & -1 & 0 \end{array} \right] \right) \]

Post-concatenates a frustum matrix
**glFrustum** Matrix

- **Projection specification**
  - `glLoadIdentity();`
  - `glFrustum(-4, +4, -3, +3, 5, 80)`
  - `left=-4, right=4, bottom=-3, top=3, near=5, far=80`

- **Matrix**

\[
\begin{bmatrix}
  \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\
 0 & \frac{2n}{t-b} & \frac{r-l}{t-b} & 0 \\
 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\
 0 & 0 & -1 & 0
\end{bmatrix}
\]

\[
= \begin{bmatrix}
  \frac{5}{4} & 0 & 0 & 0 \\
 0 & \frac{5}{3} & 0 & 0 \\
 0 & 0 & -\frac{85}{75} & -\frac{800}{75} \\
 0 & 0 & -1 & 0
\end{bmatrix}
\]

-symmetric left/right & top/bottom so zero
glFrustum Example

Consider

- `glLoadIdentity();`
- `glFrustum(-30, 30, -20, 20, 1, 1000)`
  - `left=-30, right=30, bottom=-20, top=20, near=1, far=1000`

Matrix

\[
\begin{bmatrix}
\frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\
0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\
0 & 0 & -(f+n) & -2fn \\
0 & 0 & f-n & f-n \\
0 & 0 & -1 & 0 \\
\end{bmatrix}
\]

\[
= \begin{bmatrix}
1 & 0 & 0 & 0 \\
\frac{30}{1001} & 0 & 0 & 0 \\
0 & \frac{20}{1001} & 0 & 0 \\
0 & 0 & \frac{999}{1001} & \frac{2000}{1001} \\
0 & 0 & -1 & 0 \\
\end{bmatrix}
\]

-Z axis

*symmetric left/right & top/bottom so zero*
glOrtho and glFrustum

- These OpenGL commands provide a parameterized transform mapping eye space into the “clip cube”
- Each command
  - glOrtho is orthographic
  - glFrustum is single-point perspective
Next Lecture

- More viewing
- Transform from object to eye space