Viewing and perspectives
Overview

• Classical views
• Computer viewing
• Perspective normalization
Classical Viewing

• Viewing requires three basic elements
  – One or more objects
  – A viewer with a projection surface
  – Projectors that go from the object(s) to the
    projection surface
• Classical views are based on the relationship
  among these elements
  – The viewer picks up the object and orients it how
    she would like to see it
• Each object is assumed to constructed from flat
  principal faces
  – Buildings, polyhedra, manufactured objects
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 Planar Geometric Projections

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

Object

Projector

Projection plane

DOP
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
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, games
Example, isometric/dimetric
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)
One-Point Perspective

- One principal face parallel to projection plane
- One vanishing point for cube
Two-Point Perspective

- One principal direction parallel to projection plane
- Two vanishing points
Three-Point Perspective

- No principal face parallel to projection plane
- Three vanishing points for cube
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)
Computer Viewing

• There are three aspects of the viewing process, all of which are implemented in the pipeline,
  – Positioning the camera
    • Setting the model-view matrix
  – Selecting a lens
    • Setting the projection matrix
  – Clipping
    • Setting the view volume
The OpenGL Camera

• In OpenGL, initially the object and camera frames are the same
  – Default model-view matrix is an identity
• The camera is located at origin and points in the negative z direction
• OpenGL also specifies a default view volume that is a cube with sides of length 2 centered at the origin
  – Default projection matrix is an identity
Default Projection

Default projection is orthogonal

Projection plane $z=0$
Moving the Camera Frame

- If we want to visualize object with both positive and negative z values we can either
  - Move the camera in the positive z direction
    - Translate the camera frame
    - Move the objects in the negative z direction
    - Translate the world frame
  - Both of these views are equivalent and are determined by the model-view matrix
    - Want a translation (`glTranslatef(0.0,0.0,-d);`)
Moving Camera back from Origin

frames after translation by \(-d\)

\[ d > 0 \]

default frames

(a) (b)
Moving the Camera

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

• Example: side view
  – Rotate the camera
  – Move it away from origin
  – Model-view matrix $C = TR$
OpenGL code

- Remember that last transformation specified is first to be applied

```c
glMatrixMode(GL_MODELVIEW)
glLoadIdentity();
glTranslatef(0.0, 0.0, -d);
glRotatef(90.0, 0.0, 1.0, 0.0);
```
The LookAt Function

- The GLU library contains the function `gluLookAt` to form the required modelview matrix through a simple interface
- Note the need for setting an up direction
- Still need to initialize
  - Can concatenate with modeling transformations
- Example: isometric view of cube aligned with axes

```cpp
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0., 1.0, 0.0);
```
gluLookAt

```
glLookAt(eyex, eyey, eyez, atx, aty, atz, upx, upy, upz)
```
Other Viewing APIs

- The LookAt function is only one possible API for positioning the camera
- Others include
  - View reference point, view plane normal, view up (PHIGS, GKS-3D)
  - Yaw, pitch, roll
  - Elevation, azimuth, twist
  - Direction angles
Projections and Normalization

• 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
Homogeneous Coordinate Representation

default orthographic projection

\[ \begin{align*}
x_p &= x \\
y_p &= y \\
z_p &= 0 \\
w_p &= 1
\end{align*} \]

\[ \mathbf{p}_p = \mathbf{M} \mathbf{p} \]

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

In practice, we can let \( \mathbf{M} = \mathbf{I} \) and set the \( z \) term to zero later.
Simple Perspective

- Center of projection at the origin
- Projection plane \( z = d, \ d < 0 \)
Perspective Equations

Consider top and side views

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

consider $q = Mp$ where $M =$

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

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

• However \( w \neq 1 \), so we must 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

• We will consider the corresponding clipping volume with the OpenGL functions
OpenGL Orthogonal Viewing

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

The `glOrtho` function specifies a rectangular viewing frustum with

- `left`, `right`, `bottom`, `top` are the clipping planes aligned with the axes
- `near`, `far` are distance planes

Near and far measured from the camera.
OpenGL Perspective

`glFrustum(left, right, bottom, top, near, far)`
Using Field of View

- With `glFrustum` it is often difficult to get the desired view
- `gluPerspective(fovy, aspect, near, far)` often provides a better interface

![Diagram of perspective projection]

\[ \text{aspect} = \frac{w}{h} \]
Normalization

• Rather than derive a different projection matrix for each type of projection,
  – convert all projections to orthogonal projections with the default view volume

• Use standard transformations in the pipeline

• Makes for efficient clipping
Pipeline View

modelview transformation → projection transformation → perspective division

nonsingular

clipping → projection

against default cube

4D → 3D

3D → 2D
Notes

• Stay in four-dimensional homogeneous coordinates through both the modelview and projection transformations
  – Both these transformations are nonsingular
  – Default to identity matrices (orthogonal view)
• Normalization lets us clip against simple cube regardless of type of projection
• Delay final projection until end
  – Important for hidden-surface removal to retain depth information as long as possible
Orthogonal Normalization

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

normalization ⇒ find transformation to convert specified clipping volume to default

(left, bottom, -near) → (1, -1, 1)

(right, top, -far) → (1, 1, -1)
Orthogonal Matrix

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

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

- Set $z = 0$
- Equivalent to the homogeneous coordinate transformation

\[
M_{\text{orth}} = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

- Hence, general orthogonal projection in 4D is

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

• The OpenGL projection functions cannot produce general parallel projections such as

• However if we look at the example of the cube it appears that the cube has been sheared

• Oblique Projection = Shear + Orthogonal Projection
General Shear

Top view

Side view
Shear Matrix

xy shear (z values unchanged)

\[ H(\theta, \phi) = \begin{bmatrix}
1 & 0 & -\cot \theta & 0 \\
0 & 1 & -\cot \phi & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
\end{bmatrix} \]

Projection matrix

\[ P = M_{\text{orth}} \ H(\theta, \phi) \]

General case:

\[ P = M_{\text{orth}} \ \text{STH}(\theta, \phi) \]
Effect on Clipping

• The projection matrix $\mathbf{P} = \mathbf{STH}$ transforms the original clipping volume to the default clipping volume.

![Diagram showing the effect of clipping]

- **DOP (Depth of Field Plane)**
  - Near plane
  - Far plane
  - Clipping volume
  - Object

- **Top view**
  - $x = -1$
  - $z = 1$

- **Distorted object** (projects correctly)
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

$$x = \pm z, \ y = \pm z$$
Perspective Matrices

Simple projection matrix in homogeneous coordinates

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

Note that this matrix is independent of the far clipping plane
Generalization

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

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

\[
x'' = x/z \\
y'' = y/z \\
Z'' = -(\alpha + \beta/z)
\]

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 $x = \pm 1, y = \pm 1$

Hence the new clipping volume is the default clipping volume
Normalization Transformation

original clipping volume

original object

new clipping volume

distorted object projects correctly

\[ z = -x \]

\[ z = x \]

\[ z = \text{far} \]

\[ z = \text{near} \]

\[ x = -1 \]

\[ x = 1 \]

\[ z = 1 \]

\[ z = -1 \]
Normalization and Hidden-Surface Removal

- Although our selection of the form of the perspective matrices may appear somewhat arbitrary, it was chosen so that if $z_1 > z_2$ in the original clipping volume then the for the transformed points $z_1' > z_2'$

- Thus hidden surface removal works if we first apply the normalization transformation

- However, the formula $z'' = -(\alpha + \beta/z)$ implies that the distances are distorted by the normalization which can cause numerical problems especially if the near distance is small
OpenGL Perspective

- \texttt{glFrustum} allows for an unsymmetric viewing frustum (although \texttt{gluPerspective} does not)

\begin{align*}
z &= z_{\text{min}} \\
(x_{\text{min}}, y_{\text{min}}, z_{\text{max}}) &\quad (x_{\text{max}}, y_{\text{max}}, z_{\text{max}})
\end{align*}

COP
OpenGL Perspective Matrix

• The normalization in `glFrustum` requires an initial shear to form a right viewing pyramid, followed by a scaling to get the normalized perspective volume. Finally, the perspective matrix results in needing only a final orthogonal transformation

\[
P = \text{NSH}
\]

our previously defined perspective matrix \hspace{6cm} \text{shear and scale}
Why do we do it this way?

• Single pipeline for both perspective and orthogonal viewing
• Stay in four dimensional homogeneous coordinates as long as possible
  – retains three-dimensional information needed for hidden-surface removal and shading
• Simplify clipping
• Next time:
  – Illumination and shading
  – Light sources
  – Reflection
  – Shading in OpenGL