3D Viewing (2)

Introduction to Computer Graphics
Torsten Möller / Manfred Klaffenböck
Reading

• Chapter 5 of Angel
• Chapter 13 of Hughes, van Dam, ...
• Chapter 7 of Shirley+Marschner
Objectives

• What kind of camera we use? (pinhole)
• What projections make sense
  – Orthographic
  – Perspective
• The viewing pipeline
• Viewing in WebGL
• Shadows
General viewing and projections

- With scene specified in world coordinate system
- Position and orientation of camera
- Projection: perspective or parallel
  - Clip objects against a view volume – which one?
  - Normalize the view volume (easier to do this way)
  - The rest is orthographic projection
Perspective Projections

- Need to describe the viewing
  - VRP - view reference point
Perspective Projections

- Need to describe the viewing
  - VRP - view reference point
  - VN - view normal
Perspective Projections

- Need to describe the viewing
  - VRP - view reference point
  - VN - view normal
  - VUP - up vector
Perspective Projections

- Need to describe the viewing
  - VRP - view reference point
  - VN - view normal
  - VUP - up vector
  - COP - centre of projection
Clipping

- Projection defines a view volume
- Add front and back clipping plane
- Make view volume finite
- Why is this useful for a perspective projection?
Clipping (2)

• How should we clip a scene against the view volume?
• We could clip against the actual coordinates, but is there an easier way?
Clipping (3)

• Canonical view volume
  – Parallel:
    \[x = -1, x = 1, y = -1, y = 1, z = 0, z = -1\]
  – Perspective:
    \[x = z, x = -z, y = z, y = -z, z = -z_{\text{min}}, z = -1\]
Viewing Process

- Apply normalizing transformation
- Clip against canonical view volume
- Project onto projection plane
- Transform into viewport in 2D device coordinates for display

3D world-coordinate output primitives

2D device coordinates
Normalization - Parallel

- translate the VRP to the origin
Normalization - Parallel (2)

- rotate the view reference coordinates such that the VPN becomes the z axis, u becomes the x axis and v becomes the y axis
Normalization - Parallel (3)

- shear so that the direction of projection is parallel to the z axis
Normalization - Parallel (4)

- translate and scale into the canonical view volume
Normalization - Parallel (5)

- translates the VRP to the origin
- rotate the view reference coordinates such that the VPN becomes the z axis, u becomes the x axis and v becomes the y axis
- shear so that the direction of projection is parallel to the z axis
- translate and scale into the canonical view volume
Normalization - Parallel (6)

- Projection matrix:

\[ M = S \times T \times SH \times R \times T \]
Normalization - Perspective

- translates the VRP to the origin
Normalization - Perspective (2)

- rotate the view reference coordinates such that the VPN becomes the z axis, u becomes the x axis and v becomes the y axis.
Normalization - Perspective (3)

• Translate so that the center of projection is at the origin
Normalization - Perspective (4)

- shear so that the centre line of the view volume becomes the z axis
Normalization - Perspective (5)

- scale into the canonical view volume
Normalization - Perspective (6)

- translates the VRP to the origin
- rotate the view reference coordinates such that the VPN becomes the z axis, u becomes the x axis and v becomes the y axis.
- translate so that the center of projection is at the origin
- shear so that the centre line of the view volume becomes the z axis
- scale into the canonical view volume
Normalization - Perspective (7)

• Projection matrix (perspective):

\[ M = P_{pers} \times S \times SH \times T \times R \times T \]

• Projection matrix (parallel):

\[ M = P_{parall} \times S \times SH \times R \times T \]
Objectives

• What kind of camera we use? (pinhole)
• What projections make sense
  – Orthographic
  – Perspective
• The viewing pipeline
• Viewing in WebGL
• Shadows
Computer Viewing

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

• In WebGL, 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
• WebGL 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
Moving the Camera Frame

• If we want to visualize an 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 $\text{Translate}(0.0, 0.0, -d)$;
  – $d > 0$
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 $C = TR$
The LookAt Function

- The GLU library contained the function `gluLookAt` to form the required modelview matrix through a simple interface
- Note the need for setting an up direction
- Replaced by `LookAt()` in `mat.h`
  - Can concatenate with modeling transformations
- Example: isometric view of cube aligned with axes

```cpp
mat4 mv = LookAt(vec4 eye, vec4 at, vec4 up);
```
gluLookAt

- **LookAt(eye, at, up)**

\[
\begin{align*}
\text{eye}_x, & \quad \text{eye}_y, & \quad \text{eye}_z \\
\text{up}_x, & \quad \text{up}_y, & \quad \text{up}_z \\
\text{at}_x, & \quad \text{at}_y, & \quad \text{at}_z
\end{align*}
\]
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
WebGL Orthogonal Viewing

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

near and far measured from camera
WebGL Perspective

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

- With Frustum it is often difficult to get the desired view

**Perspective** (*fov, aspect, near, far*)

often provides a better interface

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

![Diagram showing perspective with field of view (FOV) and aspect ratio](image.png)
Objectives

- What kind of camera we use? (pinhole)
- What projections make sense
  - Orthographic
  - Perspective
- The viewing pipeline
- Viewing in WebGL
- Shadows
Application of projection: shadows

- Essential component for realistic rendering
- Naturally use projections
  - Can produce **hard** shadows
  - Only handles shadows on a plane
- More advanced shadow algorithms exist, e.g., soft shadows and penumbra (not easy to do)
- Blinn 74
Shadow via projection

• A **shadow polygon** is obtained through projection where the center of projection is a light source.
Shadow polygon: parallel projection

- Project shadow on $z = 0$
- Light at $\alpha$ (directional light)
- Derive the projection matrix

\[
\begin{bmatrix}
x_s \\
y_s \\
0 \\
1
\end{bmatrix}
= \begin{bmatrix}
x_p \\
y_p \\
0 \\
1
\end{bmatrix}
\begin{bmatrix}
x_p \\
y_p \\
z_p \\
1
\end{bmatrix}
\]
Shadows: parallel projection

• Light at infinity: \[ S = P - \alpha L \]
  \[ 0 = z_P - \alpha z_L \]

• hence projection:

\[
\begin{bmatrix}
    x_S \\
    y_S \\
    0 \\
    1
\end{bmatrix} = \begin{bmatrix}
    1 & 0 & -x_L/z_L & 0 \\
    0 & 1 & -y_L/z_L & 0 \\
    0 & 0 & 0 & 0 \\
    0 & 0 & 0 & 1
\end{bmatrix}\begin{bmatrix}
    x_P \\
    y_P \\
    z_P \\
    1
\end{bmatrix}
\]

Now draw the object on the ground plane
Shadows: perspective projection

- Project on plane $z = 0$
- **Point light source**
- Derive the matrix

\[
\begin{bmatrix}
    x_S \\
    y_S \\
    0 \\
    1
\end{bmatrix}
= 
\begin{bmatrix}
    x_P \\
    y_P \\
    z_P \\
    1
\end{bmatrix}
\]
Shadows: perspective projection

• Local Light: \( S = P - \alpha (P - L) \)

\[ \alpha = \frac{z_P}{z_P - z_L} \]

• projection matrix:

\[
\begin{bmatrix}
    x_S \\
    y_S \\
    0 \\
    1
\end{bmatrix} =
\begin{bmatrix}
    -z_L & 0 & x_L & 0 \\
    0 & -z_L & y_L & 0 \\
    0 & 0 & 0 & 0 \\
    0 & 0 & 1 & -z_L
\end{bmatrix}
\begin{bmatrix}
    x_P \\
    y_P \\
    z_P \\
    1
\end{bmatrix}
\]

\( x_P \cdot (-z_L) + z_P \cdot x_L \)

\( z_P - z_L \)
Shadow of a teapot
Shadows

• requires no extra memory
• easily handles any number of light sources
• only shadows onto ground plane
• cannot handle objects which shadow other complex objects
• every polygon is rendered $N+1$ times, where $N$ is number of light sources