Perspective Projection

Wolfgang Heldrich

Course News

Assignment 1
- Due February 2

Homework 1
- Discussed in labs this week

Homework 2
- Exercise problems for perspective
- Discussed in labs next week

Reading
- Chapter 6

Recap: Transformation Hierarchies

Hierarchical Modeling

Advantages
- Define object once, instantiate multiple copies
- Transformation parameters often good control knobs
- Maintain structural constraints if well-designed

Limitations
- Expressivity: not always the best controls
- Can’t do closed kinematic chains
  - *Keep hand on hip*

Display Lists

Concept:
- If multiple copies of an object are required, it can be compiled into a display list:
  - `glNewList( listId, GL_COMPILE );`
  - `glBegin( ... );`
  - `... // geometry goes here`
  - `glEndList();`
  - // render two copies of geometry offset by 1 in z-direction:
  - `glCallList( listId );`
  - `glTranslatef( 0.0, 0.0, 1.0 );`
  - `glCallList( listId );`

Advantages:
- More efficient than individual function calls for every vertex/attribute
- Can be cached on the graphics board (bandwidth)
- Display lists exist across multiple frames
  - *Represent static objects in an interactive application*
**Shared Vertices**

**Triangle Meshes**
- Multiple triangles share vertices
- If individual triangles are sent to graphics board, every vertex is sent and transformed multiple times
  - **Computational expense**
  - **Bandwidth**

**Triangle Strips**

**Idea:**
- Encode neighboring triangles that share vertices
- Use an encoding that requires only a constant-sized part of the whole geometry to determine a single triangle
- N triangles need n+2 vertices

**Triangle Strips**

**Orientation:**
- Strip starts with a counter-clockwise triangle
- Then alternates between clockwise and counter-clockwise

**Triangle Fans**

**Similar concept:**
- All triangles share on center vertex
- All other vertices are specified in CCW order

**Triangle Strips and Fans**

**Transformations:**
- n+2 for n triangles
- Only requires 3 vertices to be stored according to simple access scheme
- Ideal for pipeline (local knowledge)

**Generation**
- E.g. from directed edge data structure
- Optimize for longest strips/fans

**Vertex Arrays**

**Concept:**
- Store array of vertex data for meshes with arbitrary connectivity (topology)
- `GLfloat *points[3*nvertices];`
- `GLfloat *colors[3*nvertices];`
- `GLuint *tris[nvertices] =` `{0, 1, 3, 2, 4, ...};`
- `glVertexPointer(..., points);`
- `glColorPointer(..., colors);`
- `glDrawElements(GL_TRIANGLES,...,tris);`
Vertex Arrays

**Benefits:**
- Ideally, vertex array fits into memory on GPU
- Then all vertices are transformed exactly once

**In practice:**
- Graphics memory may not be sufficient to hold model
- Then either:
  - Cache only parts of the vertex array on board (may lead to cache thrashing!)
  - Transform everything in software and just send results for individual triangles (bandwidth problem: multiple transfers of same vertex)

---

**Projective Rendering Pipeline**

Object world viewing device

**Scene graph**

Object geometry Modelling Transforms

Viewing Transform

Projection Transform

Result: all vertices of scene in shared 3D world coordinate system

---

**Rendering Pipeline**

Object world viewing device

Scene graph Object geometry

Modelling Transform

Viewing Transform

Projection Transform

Result: scene vertices in 3D view (camera) coordinate system
Rendering Pipeline

- Scene graph
  - Object geometry
    - Modelling
      - Transforms
    - Viewing
      - Transform

- Projection Transform

Perspective Transformation

- **Pinhole Camera**:
  - Light shining through a tiny hole into a dark room yields upside-down image on wall

Real Cameras

- Pinhole camera has small aperture (lens opening)
  - hard to get enough light to expose the film

- real pinhole camera

- Lens permits larger apertures
- Lens permits changing distance to film plane without actually moving the film plane

- camera

- price to pay: limited depth of field

Real Cameras - Depth of Field

- **Limited depth of field**
  - Can be used to direct attention
  - Artistic purposes

Perspective Transformation

- In computer graphics:
  - Image plane is conceptually in front of the center of projection

- Perspective transformations belong to a class of operations that are called projective transformations
- Linear and affine transformations also belong to this class
- All projective transformations can be expressed as 4x4 matrix operations
**Perspective Projection**

- **Synopsis:**
  - Project all geometry through a common center of projection (eye point) onto an image plane.

```
            x
            |
            |
            |
            y
            |
            z
```

**Perspective Projection**

- **Analysis:**
  - This is a special case of a general family of transformations called projective transformations.
  - These can be expressed as 4x4 homogeneous matrices!
    - E.g. in the example:
      \[
      T = \begin{bmatrix}
      1 & 0 & 0 & 0 \\
      0 & 1 & 0 & 0 \\
      0 & 0 & 1 & 0 \\
      0 & 0 & 0 & 1 \\
      \end{bmatrix} T \begin{bmatrix}
      x \\
      y \\
      z \\
      1 \\
      \end{bmatrix} = \begin{bmatrix}
      x/z \\
      y/z \\
      -1 \\
      1 \\
      \end{bmatrix}
      \]

**Projective Transformations**

- **Convention:**
  - Viewing frustum is mapped to a specific parallelepiped.
    - Normalized Device Coordinates (NDC)
      - Only objects inside the parallelepiped get rendered.
      - Which parallelepiped is used depends on the rendering system.
  - OpenGL:
    - Left and right image boundary are mapped to \(x=\pm 1\) and \(x=-1\).
    - Top and bottom are mapped to \(y=\pm 1\) and \(y=1\).
    - Near and far plane are mapped to \(z=\pm 1\) and \(z=1\).
Projective Transformations

- **Why near and far plane?**
  - Near plane:
    - Avoid singularity (division by zero, or very small numbers)
  - Far plane:
    - Store depth in fixed-point representation (integer), thus have finite range of values (0...1)
    - Avoid/reduce numerical precision artifacts for distant objects

- **Alternative specification of symmetric frusta**
  - Field-of-view (fov) $\alpha$
  - Frv/f
  - Field-of-view in y-direction ($\alpha_y$) + aspect ratio

- **Properties:**
  - All transformations that can be expressed as homogeneous 4x4 matrices (in 3D)
  - 16 matrix entries, but multiples of the same matrix all describe the same transformation
    - 15 degrees of freedom
    - The mapping of 5 points uniquely determines the transformation

- **Determining the matrix representation**
  - Need to observe 5 points in general position, e.g.
    - $[\text{left},0,0,1]^T \rightarrow [1,0,0,1]^T$
    - $[0,\text{top},0,1]^T \rightarrow [0,1,0,1]^T$
    - $[0,0,-\text{left},1]^T \rightarrow [0,0,1,1]^T$
    - $[0,0,-\text{top},1]^T \rightarrow [0,0,0,1]^T$
    - $[\text{left}^*\text{top}^*,0,-\text{left},1]^T \rightarrow [1,1,1,1]^T$
  - Solve resulting equation system to obtain matrix

Demos

- Tuebingen applets from Frank Hanisch
  - [http://www.gits.uni-tuebingen.de/project/sgrid/doc/heitke/sketch/](http://www.gits.uni-tuebingen.de/project/sgrid/doc/heitke/sketch/)
  - [Applet/index.html](http://www.gits.uni-tuebingen.de/project/sgrid/doc/heitke/sketch/Applet/index.html)
**Perspective Derivation**

\[
\begin{bmatrix}
\begin{array}{c}
x' \\
y' \\
z'
\end{array}
\end{bmatrix} =
\begin{bmatrix}
E & 0 & 0 & x \\
0 & F & 0 & y \\
0 & 0 & C & z \\
0 & 0 & -1 & 1
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z \\
w
\end{bmatrix}
\]

- \( x' = E \cdot x \) left \( \rightarrow x' / w' = 1 \)
- \( y' = F \cdot y \) right \( \rightarrow y' / w' = -1 \)
- \( z' = C \cdot z \) top \( \rightarrow z' / w' = 1 \)
- \( w' = -z \) bottom \( \rightarrow y' / w' = -1 \)
- \( z = \text{near} \rightarrow z' / w' = 1 \)
- \( z = \text{far} \rightarrow z' / w' = -1 \)

\[
y' = F_y + B_z, \quad \frac{y'}{w'} = \frac{F_y + B_z}{w'}, \quad 1 = F_y + B_z
\]

\[
1 = F \cdot \frac{z}{-z}, \quad 1 = F \cdot \frac{-z}{-z}, \quad 1 = F \cdot \frac{z}{-z}, \quad 1 = F \cdot \frac{-z}{-z}
\]

**Perspective Example**

- **view volume**
  - \( \text{left} = -1 \), \( \text{right} = 1 \)
  - \( \text{bot} = -1 \), \( \text{top} = 1 \)
  - \( \text{near} = 1 \), \( \text{far} = 4 \)

\[
\begin{bmatrix}
2n & 0 & r+l & 0 \\
r-l & 2n & r+l & 0 \\
0 & t-b & 0 & 0 \\
0 & 0 & -f+n & -2fn
\end{bmatrix}
\begin{align}
\begin{bmatrix}
x' \\
y' \\
z' \\
w'
\end{bmatrix}
\end{align}
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & -5/3 & -8/3 \\
0 & 0 & -1 & 0
\end{bmatrix}
\]

**Orthographic Camera Projection**

- Camera’s back plane parallel to lens
- Infinite focal length
- No perspective convergence
- Just throw away \( z \) values

\[
\begin{bmatrix}
y' \\
z'
\end{bmatrix} =
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z \\
w
\end{bmatrix}
\]

**Projective Transformations**

- **Properties**
  - Lines are mapped to lines and triangles to triangles
  - Parallel lines do NOT remain parallel
    - E.g., rails vanishing at infinity
  - Affine combinations are NOT preserved
    - E.g., center of a line does not map to center of projected line (perspective foreshortening)

**Projection Taxonomy**
Perspective Projections
- classified by vanishing points

Axonometric Projections
- projectors perpendicular to image plane

View Volumes
- specifies field-of-view, used for clipping
- restricts domain of z stored for visibility test

View Volume
- Convention
  - Viewing frustum mapped to specific parallelepiped
    - Normalized Device Coordinates (NDC)
    - Same as clipping coords
    - Only objects inside the parallelepiped get rendered
    - Which parallelepiped?
      - Depends on rendering system

Perspective Matrices in OpenGL
- Perspective Matrices:
  - glFrustum(left, right, bottom, top, near, far)
    - Specifies perspective xform (near, far are always positive)
  - glOrtho(left, right, bottom, top, near, far)

Convenience Functions:
- gluPerspective(fovy, aspect, near, far)
  - Another way to do perspective
- gluLookAt(eyeX, eyeY, eyeZ, centerX, centerY, centerZ, upX, upY, upZ)
  - Useful for viewing transform

Projective Rendering Pipeline
- object space
- world space
- viewing space
- clipped space
**Window-To-Viewport Transformation**

- **Generate pixel coordinates**
  - Map $x, y$ from range $-1...1$ (normalized device coordinates) to pixel coordinates on the screen.
  - Map $z$ from $-1...1$ to $0...1$ (used later for visibility).
  - Involves 2D scaling and translation.

**Coming Up:**

- **Wednesday:**
  - More on perspective projection

- **Friday/Next Week**
  - Lighting/shading