Review

- What we know by now:
  - How to define objects mathematically (explicit, implicit, parametric)
  - How to represent objects for rendering (vertices, lines, triangles)
  - How to transform objects (transformation matrices)
  - How 2D objects are rasterized, colored, textured

- What we will learn next:
  - How (3D) objects are projected onto screen (camera settings)
  - How objects are clipped against camera frustum
  - Techniques for 3D rendering (lighting, occlusion, etc.)
Chapter 5

1. Introduction
2. The Computer Graphics Pipeline
3. Object Representation
4. Object Transformation
5. 3D – Projections, Camera, and Lighting
6. Scene Representation and Interaction
7. Advanced Texturing and Shading
From 2D to 3D

- 3D introduces
  - Perspective (camera models)
  - Occlusion (depth)
  - Perception challenges
  - Interaction challenges
The Camera

- Defines viewing orientation and projection
- Defines what region or volume is visible on screen
- Has its own coordinate frame
- Defines “how objects appear on screen” (mapping to 2D)
Orthogonal Projection

- Orthogonal (orthographic) projection
  - 2D view is special case of 3D projection
  - World maps orthogonally onto screen (projectors are parallel)
  - Useful to measure distances (cf. Blender)
  - COP at infinity creates **viewing volume** defined by four (six) parallel/orthogonal planes
Parallel Projection

- Parallel projections
  - Orthogonal projection is a parallel projection
  - More general projections (oblique projections) do not require image plane and projections to be orthogonal
Perspective Projection

- Perspective projection
  - Objects further away project to smaller areas on screen
  - Perspective causes objects to appear “distorted”
  - **Viewing volume** (viewing frustum) has pyramid shape
Camera Positioning

• Coordinate system of camera not necessarily aligned with world frame
• Camera positioning is handled by ModelView transform
• Conventions:
  • Camera looks along negative z axis (near/far distances are along z)
  • Y axis points up

• OpenGL:
  • ModelView Matrix defines mapping from local to world, to camera frame
  • Projection Matrix defines mapping from camera/eye coordinates to clip-coordinates
Projection Parameters

• Parameters that govern projection properties:
  • **Field of View** (y: Angle between top and bottom plane)
  • **Near and Far** (distances of clipping planes)
  • **Aspect Ratio** (width/height of image plane)
  • In general: Properties of planes that delineate viewing frustum
Projection and Mapping Parameters

• Implementation in Computer Graphics APIs:
  • Specify projection parameters (OpenGL):

```c
glMatrixMode(GL_PROJECTION)
glFrustum(l, r, bottom, top, near, far)       //sets perspective projection
gluPerspective(fovy, aspect, near, far)     //sets perspective projection
glOrtho(left, right, bottom, top, near, far) //orthogonal projection
```

OpenGL 3.0+: Set up projection matrix explicitly/manually

• How does projection work conceptually?
Projection

- Projection maps viewing frustum to canonical volume $[-1,1]^3$
Projection

- In canonical volume x,y coordinates of points represent 2D location (**important**: depth values are still available)
- How does projection work mathematically?
Orthogonal Projection

- Viewing frustum defined by left, right, near, and far planes
- Easy mapping to \([-1,1]^3\]
- Translate and scale to map to canonical volume

\[
T(-(right + left)/2, -(top + bottom)/2, (far + near)/2) \\
S(2/(right - left), 2/(top - bottom), -2/(far - near))
\]

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

• Points are projected along ray through camera origin
• Points along a ray are projected to the same location
Perspective Projection

\[ \frac{x}{z} = \frac{x_p}{z_p} \]

\[ x_p = \frac{x}{z/z_p} \]

\[ z_p = d \]

\[ x_p = \frac{x}{z/d} \]

\[ y_p = \frac{y}{z/d} \]