Review of Chapter 5 – Camera and Perspective

• Camera defines a **center of projection** and viewing volume (**frustum**)

• **Projection Transformation** maps viewing volume to canonical volume

• **Orthogonal projection** useful for measuring of lengths and 2D scenes

• **Perspective projection** shrinks objects that are farther away from the camera
Review of Chapter 5 – Depth

- Vertices still keep depth information after perspective transformation
- Depth is interpolated across triangles for per-fragment depth values

- Depth has to be resolved during drawing
  - Depth ordering/Depth sort (object based ordering)
  - Z-buffering (fragment based ordering)
Review of Chapter 5 – Transparency

• Transparent geometry presents new challenges
• Order-dependent alpha-blending is common

• Order-dependent transparency requires sorting of objects
  • Drawing objects in back-to-front order
  • Advanced techniques (depth-peeling) possible, but slow
Review of Chapter 5 - Lighting

• Lighting greatly improves 3D perception
• Basic lighting can be computed with local information (local lighting) and is suitable for implementation in the standard graphics pipeline

• Shading colors determined by Phong Lighting Model
  • Ambient reflection (independent of light direction)
  • Diffuse reflection (dependent on normal and light direction)
  • Specular reflection (dependent on viewing, light, and normal direction)
Review of Chapter 5 - Lighting

• Three types of polygonal shading
  • Flat shading (one normal per triangle)
  • Smooth/Gouraud Shading (one normal per vertex, shading color is interpolated)
  • Phong/Per-Fragment Shading (one normal per fragment as obtained by interpolating vertex normals)
Chapter 6

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
Scenes

- Rendering a scene requires (efficient) drawing of several objects

- Additional challenges:
  - Objects consisting of multiple dependent components
  - Complex object relationships
  - Scenes can be dynamic and change
  - Scene too big to be rendered at once
  - (Interactive/efficient) scene exploration required
Scene Definition

• A scene is defined as a collection of objects in the same frame of reference
• The ModelView Matrix places objects into a common space (world space and view space)
Object Construction

- We know how to model an object as a collection of vertices and attributes.

```plaintext
float data[] = {
    // vertices of triangles
    // Bottom face coord. // Top face coord. // vertex normals
    0,0,0, 0,1,0, 0,-1,0,
    1,0,0, 0,1,1, 0,-1,0,
    1,0,1, 1,1,1, 0,-1,0,
    ...  ...  ...  ...
    0,0,0, 0,1,0, 0,-1,0,
    1,0,1, 1,1,1, 0,-1,0,
    0,0,1, 1,1,0, 0,-1,0,
};
```

triangle vertices
Object Modeling

- Scenes are rarely collections of independent, static objects
- Especially mechanical and/or dynamic objects require a more sophisticated **object modeling** approach
Object Modeling

- Exploit object properties
  - Repeated components
  - Connected components (relationships)
  - Hierarchical behavior
Object Modeling - Relationships

- Relationships between components can be expressed as a graph.
**Object Modeling - Relationships**

- **Trees** are directed graphs without cycles and with one top node.
Object Modeling - Relationships

- **Trees** are *hierarchical* representations (c.f., object oriented programming)
- Child nodes are expressed relative to their parent nodes
- An example in graphics:
Hierarchical Object Modeling

- Hierarchical Modeling
Hierarchical Object Modeling

• Incremental transformations

\[ T(0, \text{dy}_1) \, R(\alpha_1) \]

transformation relative to torso

\[ \text{dy}_1 \]

\[ T(0, \text{dy}_2) \, R(\alpha_2) \]

transformation relative to first arm segment

\[ \text{dy}_2 \]
Hierarchical Object Modeling

\[
p = T(dx_0, dy_0)T(0, dy_1)R(\alpha_1)T(0, dy_2)R(\alpha_2) \cdot \begin{pmatrix} 0 \\ dy_3 \end{pmatrix}
\]
Hierarchical Object Modeling

- Tree traversal
- Gather relative information

Root: Torso
- Left Upper Arm
  - Left Lower Arm
- Head
- Left Upper Leg
  - Left Lower Leg
How do we render the complete object? Tree traversal strategy needed.
Tree Traversal Techniques

- **Pre-order traversal** (left to right, depth first)
Tree Traversal Techniques

- Pre-order traversal can be implemented with matrix stack

```plaintext
drawTorso();
pushMatrix();
multMatrix(M_{lu});
drawLUA();
multMatrix(M_{la});
drawLLA();
popMatrix();

pushMatrix();
multMatrix(M_{nl});
drawLUL();
multMatrix(M_{ll});
drawLLL();
popMatrix();
```
Tree Traversal Techniques

- Transformations are not the only characteristics that can be applied per node

- Color, texture, material properties etc. might have to be changed

- Some of these properties are not inherently hierarchical
Kinematics in Animation

• Object representation ‘attached’ to a **kinematic** skeleton

See also: ‘Rigging’ and ‘skinning’