ECS 175 - Summary

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
The Graphics Card

- Graphics Card
  - A highly parallel computer (stream processing, SIMD)
The Graphics Pipeline

Vertices → Vertex Processor → Clipper and Assembler → Rasterizer → Fragment Processor → Pixels

3D Triangles → Clipped Triangles → 2D Triangles → Fragments
Rasterization

- Rasterization identifies fragment locations and interpolates attributes from vertices to fragments.

Sampling leads to aliasing.
Interpolation

- **Interpolation** constructs “new” data within the range of known data points

- Construct a function with continuous domain from **discrete data** (e.g., colors given at vertices)

![Diagram showing linear and quadratic interpolations]

- Linear interpolation
- Quadratic interpolation
Attribute Interpolation

- Colors specified per vertex

- 2D **texture coordinates** \((s,t)\) specified per vertex

- Texels mapped to pixels – aliasing possible
General Object Representations

- Three ways to represent curves and surfaces

  - **Explicit**
    \[ y = f(x) \]
    ‘Graphs’, terrain, etc.

  - **Implicit**
    \[ f(x, y) = 0 \]
    Defines membership function

  - **Parametric**
    \[ f(u) = (x(u), y(u), z(u)) \]
    Bezier curves, etc.

For efficient rendering and rasterization functions are **tessellated** into triangles (Marching Quads, etc.)
When/Where are transformations performed?

Object Coordinates → Vertex Processor → Clipper and Assembler → Normalized Device Coordinates → Rasterizer → Window Coordinates → Fragment Processor
Homogeneous Coordinates

• How to interpret a transformation matrix

\[ v_{world} = \begin{pmatrix} 0.5 & 0 & 0 & p_1 \\ 0 & 0.5 & 0 & p_2 \\ 0 & 0 & 0.5 & p_3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot v_{local} \]

Local coordinate frame (source) expressed in global coordinates (target).

Inverse matrix reverses transformation.

Homogeneous coordinates distinguish between points and vectors.
Projection

- **Projection** is performed after **ModelView transformation**
- Projection maps viewing frustum to canonical volume $[-1,1]^3$
Clipping

• Before **perspective division**, we **clip** primitives by testing in clip coordinates (this is how it is done in OpenGL):

\[
\begin{align*}
-w_p & \leq x_p \leq w_p \\
-w_p & \leq y_p \leq w_p \\
-w_p & \leq z_p \leq w_p
\end{align*}
\]
Viewport Transform

- From normalized device coordinates to window/screen coordinates
Adding Depth

- How do we render images that contain objects at different depths?

- Objects in the back have to appear behind objects in the front

- Solve the **hidden-surface removal** problem
The Z-Buffer

- Several fragments created by rasterizing primitives can map to the same target pixel
- Compare and update value in the **z-buffer** (resolve depth-ordering problem per fragment)
Transparency and Depth

- Transparency is typically modeled using **alpha-blending**
- Final color contains contributions of multiple fragments

```
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```

\[
RGB_{final} = (1 - A_{in}) \cdot RGB_{existing} + A_{in} \cdot RGB_{in}
\]
Adding Light: The Phong Lighting Model

- **Phong lighting model** uses 4 vectors and three types of reflection (ambient, diffuse, specular)
- Efficient approximation of local lighting

\[ r \text{ = direction of ideal reflection} \]
The Phong Model

- Intensity is sum of individual reflected intensities

\[ I = \frac{1}{a + bd + cd^2} (k_d L_d \max(l \cdot n, 0) + k_s L_s \max((r \cdot v)^\alpha, 0)) + k_a L_a \]

- Distance attenuation
- Specular reflection
- Diffuse reflection
- Ambient reflection

Intensity is sum of individual reflected intensities
Complex Objects: Hierarchical Object Modeling

- Hierarchical Modeling
Forward and Inverse Kinematics

- **Forward Kinematics** computes the end-position of the kinematic chain dependent on parameter values

\[ p = f(\theta) \]

“Where is the hand of my robot if I choose these angles at the joints?”

- **Inverse Kinematics** finds parameters for a desired state/position

\[ \theta = g(p) \quad (\theta = f^{-1}(p) \quad (?)) \]

“What angles should I use at the joints to place my hand over there?”
• Combining this hierarchy with spatial subdivision or spatial partitioning can greatly improve rendering performance
• This is because our camera can be located inside the scene. Visibility function can be stored in tree.
Interaction

- **Interaction**
  - Allows user to change parameters of graphical representation
  - Defined through **User Interface**
  - APIs define **logical devices** for input
  - **Graphical User Interfaces** present options to the user
  - **Input mode** governs how and when input is processed

- **Typical tasks** in computer-graphics
  - Navigation (virtual sphere, …)
  - Picking (color-based picking, …)
  - Changing scalar/vector attributes
Advanced Techniques

- **Level-Of-Detail Methods**
  - Reduce or increase geometric complexity when needed
  - Exploit properties of perspective projection

- **Procedural Methods**
  - Details on-demand
  - Language-Based Systems
  - Trees, Plants

- **Perlin Noise**
  - Nature is “random”
  - White noise is not optimal; Perlin noise better
Advanced Techniques

• **Normal Mapping**
  - Textures can be used to store normal data in tangent space
  - Increases lighting details, but preserves geometry/silhouette

• **Environment Mapping**
  - Sphere and Cube-Mapping are established techniques
  - Environment is pre-rendered into texture(s)
  - Texture coordinates are created dynamically based on reflection vector
  - Can also be used for refraction, skyboxes, etc.

• **Shadow Mapping**
  - Shadow mapping and shadow volumes
Advanced Techniques

• **Ambient Occlusion**
  - Approximates global lighting
  - Hemisphere sampling approximates partial occlusion
  - Screen-space techniques operate on depth buffer

• **Ray-Tracing**
  - Casting rays through pixels in the image plane
  - Ray splitting and reflection
  - Recursive ray-tracing
  - Well-suited for highly specular scenes
  - Highly parallelizable
  - Rays are samples – anti-aliasing techniques are required
Advanced Techniques

- **Volume Rendering** displays volumetric (scalar) data
- Combines contributions of voxels by splatting, slicing, or ray casting
- **Transfer function** defines visual mapping