1. Course Overview Revisited
   • Modeling: how to represent objects
   • Animation: how to control and represent motion
   • Rendering: how to create images
   • OpenGL graphics library

2. Basic Graphics Programming
   • The graphics pipeline
   • Pipelines and parallelism
   • Latency vs throughput
   • Efficiently implementable in hardware
   • Not so efficiently implementable in software
   • Course approach: walk the pipeline left-to-right

Graphics Functions
   • Primitive functions (points, lines, polygons)
   • Attribute functions (color, lighting, material)
   • Transformation functions (homogeneous coord)
   • Viewing functions (projections)
   • Input functions (callbacks)
   • Control functions (GLUT library calls)

3. Interaction
   • Client/Server Model
   • Callbacks
   • Double Buffering
   • Hidden Surface Removal

Announcements
   • Assignment 4 due Thursday before lecture
   • Lecture by John Ketchpaw
   • Midterm next Tuesday
     – In class
     – Closed book
     – One double-sided sheet of notes permitted
     – Everything covered in lecture so far
   • Assignment 3 movies
     – Some flaws may be problems in production software
     – Enjoy!
Client/Server Model

- Graphics hardware and caching
- Important for efficiency
- Need to be aware where data are stored
- Examples: vertex arrays, display lists

Hidden Surface Removal

- Classic problem of computer graphics
- What is visible after clipping and projection?
- Object-space vs image-space approaches
- Object space: depth sort (Painter’s algorithm)
- Image space: ray cast (z-buffer algorithm)
- Related: back-face culling

4. Transformations

- Vector Spaces
- Affine and Euclidean Spaces
- Frames
- Homogeneous Coordinates
- Transformation Matrices
- OpenGL Transformation Matrices

Geometric Interpretations

- Lines and line segments
- Convexity
- Dot product and projections
- Cross product and normal vectors
- Planes

Lines and Line Segments

- Parametric form of line: \( P(\alpha) = P_0 + \alpha \vec{d} \)
- Line segment between \( Q \) and \( R \):
  \[ P(\alpha) = (1-\alpha) \vec{Q} + \alpha \vec{R} \text{ for } 0 \leq \alpha \leq 1 \]

Convex Hull

- Convex hull defined by
  \[ P = \alpha_1 P_1 + \cdots + \alpha_n P_n \]
  for \( \alpha_1 + \cdots + \alpha_n = 1 \)
  and \( 0 \leq \alpha_i \leq 1, i = 1, \ldots, n \)
Projection

- Dot product projects one vector onto other
  \[ u \cdot v = |u| \|v\| \cos(\theta) \]

Normal Vector

- Cross product defines normal vector
  \[ u \times v = n \]
  \[ |u \times v| = |u| |v| |\sin(\theta)| \]
  - Right-hand rule

Plane

- Plane defined by point \( P_0 \) and vectors \( u \) and \( v \)
- \( u \) and \( v \) cannot be parallel
- Parametric form: \( T(\alpha, \beta) = P_0 + \alpha u + \beta v \)
- Let \( n = u \times v \) be the normal
- Then \( n \cdot (P - P_0) = 0 \) iff \( P \) lies in plane

Homogeneous Coordinates

- In affine space, \( P = \alpha_1 v_1 + \alpha_2 v_2 + \alpha_3 v_3 + P_0 \)
- Define \( 0 \cdot P = 0 \), \( 1 \cdot P = P \)
- Points \([\alpha_1, \alpha_2, \alpha_3, 1]^T\)
- Vectors \([\delta_1, \delta_2, \delta_3, 0]^T\)
- Change of frame

Affine Transformations

- Compose
  - Rotations, translations, scalings
  - Express in homogeneous coords (4 \times 4 matrices)
- Apply from right to left!
  - \( R p = (R_z R_y R_x) p = R_z (R_y (R_x p)) \)
  - Postmultiplication in OpenGL
- Think in terms of composition
  - Translation to and from origin
  - Remember geometric intuition

5. Viewing and Projection

- Camera Positioning
- Parallel Projections
- Perspective Projections
Camera in Modeling Coordinates

- Camera position is identified with a frame
- Either move and rotate the objects
- Or move and rotate the camera
- Those views are inverses!
  - Each transformation
  - Order of transformation
  - gluLookAt utility

Orthographic Projections

- Projectors perpendicular to projection plane
- Simple, but not realistic

Perspective Viewing

- Characterized by foreshortening
- More distant objects appear smaller

```
y/z = y_p/d  so  y_p = y/(z/d)
Note this is non-linear!
Need homogeneous coordinates
```

Perspective Projection Matrix

- Represent multiple of point
  \[
  \begin{bmatrix}
  z/d \\
  x/d \\
  y/d \\
  1
  \end{bmatrix}
  =
  \begin{bmatrix}
  x \\
  y \\
  z \\
  z/d
  \end{bmatrix}
  \]
- Solve
  \[
  M \begin{bmatrix}
  x \\
  y \\
  z \\
  1
  \end{bmatrix}
  =
  \begin{bmatrix}
  1 & 0 & 0 & 0 \\
  0 & 1 & 0 & 0 \\
  0 & 0 & 1 & 0 \\
  0 & 0 & 1/d & 0
  \end{bmatrix}
  \]

6. Hierarchical Models

- Matrix and attribute stacks
- Save and restore state
- Exploit natural hierarchical structure for
  - Efficient rendering
  - Example: bounding boxes (later in course)
  - Concise specification of model parameters
  - Example: joint angles
  - Physical realism

Hierarchical Objects and Animation

- Drawing functions are time-invariant
- Can be easily stored in display list
- Change parameters of model with time
- Redraw when idle callback is invoked
Complex Objects
• Tree rather than linear structure
• Interleave along each branch
• Use push and pop to save state

Unified View of Computer Animation
• Models with parameters
  – Polygon positions, control points, joint angles, ...
  – \( n \) parameters define \( n \)-dimensional state space
• Animation defined by path through state space
  – Define initial state, repeat:
    – Render the image
    – Move to next point (following motion curves)
• Animation = specifying state space trajectory

Animation vs Modeling
• Modeling: what are the parameters?
• Animation: how do we vary the parameters?
• Sometimes boundary not clear
• Build models that are easy to control
• Hierarchical models often easy to control

Basic Animation Techniques
• Traditional (frame by frame)
• Keyframing
• Procedural techniques
• Behavioral techniques
• Performance-based (motion capture)
• Physically-based (dynamics)

7. Lighting and Shading
• Approximate physical reality
• Ray tracing:
  – Follow light rays through a scene
  – Accurate, but expensive (off-line)
• Radiosity:
  – Calculate surface inter-reflection approximately
  – Accurate, especially interiors, but expensive (off-line)
• Phong Illumination model:
  – Approximate only interaction light, surface, viewer
  – Relatively fast (on-line), supported in OpenGL

Light Sources and Material Properties
• Appearance depends on
  – Light sources, their locations and properties
  – Material (surface) properties
  – Viewer position
• Ray tracing: from viewer into scene
• Radiosity: between surface patches
• Phong Model: at material, from light to viewer
Types of Light Sources

- Ambient light: no identifiable source or direction
- Point source: given only by point
- Distant light: given only by direction
- Spotlight: from source in direction
  - Cut-off angle defines a cone of light
  - Attenuation function (brighter in center)
- Light source described by a luminance
  - Each color is described separately
  - $I = [I_r, I_g, I_b]^T$ (I for intensity)
  - Sometimes calculate generically (applies to r, g, b)

Phong Illumination Model

- Calculate color for arbitrary point on surface
- Compromise between realism and efficiency
- Local computation (no visibility calculations)
- Basic inputs are material properties and $l, n, v$:
  $I = vector\ to\ light\ source$
  $n = surface\ normal$
  $v = vector\ to\ viewer$
  $r = reflection\ of\ I\ at\ p$
  (determined by $l$ and $n$)

Summary of Phong Model

- Light components for each color:
  - Ambient ($L_a$), diffuse ($L_d$), specular ($L_s$)
- Material coefficients for each color:
  - Ambient ($k_a$), diffuse ($k_d$), specular ($k_s$)
- Distance $q$ for surface point from light source
  $I = \frac{1}{a + bq + cq^2}(k_dL_d(I \cdot n) + k_sL_s(r \cdot v) + k_aL_a)$
  $l = vector\ from\ light$
  $n = surface\ normal$
  $v = vector\ to\ viewer$
  $r = l\ reflected\ about\ n$

Normal Vectors

- Critical for Phong model (diffuse and specular)
- Must calculate accurately
  - From geometry (e.g., differential calculus)
  - From approximating surface (e.g., Bezier patch)
- Pitfalls
  - Unit length (some OpenGL support)
  - Surface boundary

8. Shading in OpenGL

- Polygonal shading
- Material properties
- Approximating a sphere [example]

Polygonal Shading

- Curved surfaces are approximated by polygons
- How do we shade?
  - Flat shading
  - Interpolative shading
  - Gouraud shading
  - Phong shading (different from Phong illumination)
- Two questions:
  - How do we determine normals at vertices?
  - How do we calculate shading at interior points?
Gouraud Shading

- Special case of interpolative shading
- How do we calculate vertex normals?
- Gouraud: average all adjacent face normals
  \[ n = \frac{n_1 + n_2 + n_3 + n_4}{|n_1 + n_2 + n_3 + n_4|} \]
- Requires knowledge about which faces share a vertex

Data Structures for Gouraud Shading

- Sometimes vertex normals can be computed directly (e.g. height field with uniform mesh)
- More generally, need data structure for mesh
- Key: which polygons meet at each vertex

Drawing a Sphere

- Recursive subdivision technique quite general
- Interpolation vs flat shading effect

Recursive Subdivision

- General method for building approximations
- Research topic: construct a good mesh
  - Low curvature, fewer mesh points
  - High curvature, more mesh points
  - Stop subdivision based on resolution
  - Some advanced data structures for animation
  - Interaction with textures
- Here: simplest case
- Approximate sphere by subdividing icosahedron

Subdivision Example

- Icosahedron after 3 subdivisions (fast converg.)

9. Curves and Surfaces

- Parametric Representations
  - Also used: implicit representations
- Cubic Polynomial Forms
- Hermite Curves
- Bezier Curves and Surfaces
Parametric Forms

- Parameters often have natural meaning
- Easy to define and calculate
  - Tangent and normal
  - Curves segments (for example, \(0 \leq u \leq 1\))
  - Surface patches (for example, \(0 \leq u, v \leq 1\))

Approximating Surfaces

- Use parametric polynomial surfaces
- Important concepts:
  - Join points for segments and patches
  - Control points to interpolate
  - Tangents and smoothness
  - Blending functions to describe interpolation
- First curves, then surfaces

Cubic Polynomial Form

- Degree 3 appears to be a useful compromise
- Curves:
  - \(p(u) = c_0 + c_1 u + c_2 u^2 + c_3 u^3 = \sum_{k=0}^{3} c_k u^k\)
  - Each \(c_k\) is a column vector \([c_{kx}, c_{ky}, c_kz]^T\)
  - From control information (points, tangents) derive 12 values \(c_{kx}, c_{ky}, c_kz\) for \(0 \leq k \leq 3\)
  - These determine cubic polynomial form

Geometry Matrix

- Calculate approximating polynomial from control point with geometry matrix \(M\)
  - \(p(u) = c_0 + c_1 u + c_2 u^2 + c_3 u^3\)
  - \(\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} = M \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix}\)
  - Each form of interpolation has its own geometry matrix

Standard Methods

- Hermite curves
  - Given by 2 points, 2 tangents
  - \(C^1\) continuity, intersect control points
- Bezier curves
  - Given by 4 control points
  - Intersects 2, others approximate tangent
- Bezier surface patches
  - Given by 16 control points
  - Intersects 4 corners, other approximate tangents

Hermite Curves

- Another cubic polynomial curve
- Specify two endpoints and their tangents

\([\text{diagram correction } p(t) = p']\)
Bezier Curves

- Widely used in computer graphics
- Approximate tangents by using control points

\[ p'(0) = 3(p_1 - p_0) \]
\[ p'(1) = 3(p_3 - p_2) \]

10. Splines

- Approximating more than 4 control points
- Piecing together a longer curve or surface

B-Splines

- Use 4 points, but approximate only middle two
- Draw curve with overlapping segments
  0-1-2-3, 1-2-3-4, 2-3-4-5, 3-4-5-6, etc.
- Curve may miss all control points
- Smoother at joint points

Cubic B-Splines

- Need \( m+2 \) control points for \( m \) cubic segments
- Computationally 3 times more expensive
- \( C^2 \) continuous at each interior point
- Derive as follows:
  - Consider two overlapping segments
  - Enforce \( C^0 \) and \( C^1 \) continuity
  - Employ symmetry
  - \( C^2 \) continuity follows

Rendering by Subdivision

- Divide the curve into smaller subpieces
- Stop when "flat" or at fixed depth
- How do we calculate the sub-curves?
  - Bezier curves and surfaces: easy (next)
  - Other curves: convert to Bezier!

Subdividing Bezier Curves

- Given Bezier curve by \( p_0, p_1, p_2, p_3 \)
- Find \( l_0, l_1, l_2, l_3 \) and \( r_0, r_1, r_2, r_3 \)
- Subcurves should stay the same!
Preview I

- Physically based models
  - Particle systems
  - Spring forces (cloth)
  - Collisions and constraints
- Rendering
  - Clipping, bounding boxes
  - Line drawing
  - Scan conversion
  - Anti-aliasing

Preview II

- Textures and pixels
  - Texture mapping
  - Bump maps
  - Environment maps
  - Opacity and blending
  - Filtering
  - Image transformation
- Ray tracing
  - Spatial data structures
  - Bounding volumes

Preview III

- Radiosity
  - Inter-surface reflections
  - Ray casting
- Scientific visualization
  - Height fields and contours
  - Iso-surfaces
  - Marching cubes
  - Volume rendering
  - Volume textures

Announcements

- Assignment 4 due Thursday before lecture
- Lecture by John Ketchpaw
- Midterm next Tuesday
  - In class
  - Closed book
  - One double-sided sheet of notes permitted
  - Everything covered in lecture so far