Midterm
• topics covered: through rasterization (H2)
  • rendering pipeline
  • transforms
  • viewing/projection
  • rasterization
• topics NOT covered
  • color, lighting/shading (from 2/15 onwards)
  • H2 handed back, with solutions, on Wed

Red Book Reading For Midterm
• Ch Introduction to OpenGL
• Ch State Management and Drawing Geometric Objects
• App Basics of GLUT (Aux in v 1.1)
• Ch Viewing
• App Homogeneous Coordinates and Transformation Matrices
• Ch Display Lists

Review: Reflection Equations
• Phong specular model
  \[ I_{\text{specular}} = k_s I_{\text{light}} ( \mathbf{h} \cdot \mathbf{n} ) \]
  \[ \mathbf{h} = (\mathbf{l} + \mathbf{v}) / 2 \]
• or Blinn-Phong specular model
  \[ I_{\text{specular}} = k_s I_{\text{light}} ( \mathbf{h} \cdot \mathbf{n} ) \]
• full Phong lighting model
  \[ I_{\text{total}} = k_a I_{\text{ambient}} + \sum_{i=1}^{#\text{lights}} \left( k_d (\mathbf{n} \cdot \mathbf{l}_i) + k_s (\mathbf{v} \cdot \mathbf{r}_i) \right) \]
  \[ \mathbf{h} = (\mathbf{l}_i + \mathbf{v}) / 2 \]

Review: Lighting
• lighting models
  • ambient
    • normals don't matter
  • Lambert/diffuse
    • angle between surface normal and light
  • Phong/specular
    • surface normal, light, and viewpoint

Review: Shading Models
• flat shading
  • compute Phong lighting once for entire polygon
• Gouraud shading
  • compute Phong lighting at the vertices and interpolate lighting values across polygon

Phong Shading Artifacts
• perspective transformations
  • affine combinations only invariant under affine, not under perspective transformations
  • thus, perspective projection alters the linear interpolation!

FCG Reading For Midterm
• Ch 1
• Ch 2 Misc Math (except for 2.5.1, 2.5.3, 2.7.1, 2.7.3, 2.8, 2.9)
• Ch 5 Linear Algebra (only 5.1-5.2.2, 5.2.5)
• Ch 6 Transformation Matrices (except 6.1.6)
• Sect 13.3 Scene Graphs
• Ch 7 Viewing
• Ch 3 Raster Algorithms (except 3.2-3.4, 3.8)

Phong Shading
• linearly interpolating surface normal across the facet, applying Phong lighting model at every pixel
  • same input as Gouraud shading
  • pro: much smoother results
  • con: considerably more expensive
• not the same as Phong lighting
  • common confusion
  • Phong lighting: empirical model to calculate illumination at a point on a surface

Phong Shading Difficulties
• computationally expensive
  • per-pixel vector normalization and lighting computation!
  • floating point operations required
  • lighting after perspective projection
    • messes up the angles between vectors
    • have to keep eye-space vectors around
    • no direct support in pipeline hardware
    • but can be simulated with texture mapping

Shading Artifacts: Silhouettes
• polygonal silhouettes remain

Gouraud Shading Artifacts
• perspective transformation problem
  • colors slightly "swim" on the surface as objects move relative to the camera
  • usually ignored since often only small difference
    • usually smaller than changes from lighting variations
  • to do it right
    • either shading in object space
    • or correction for perspective foreshortening
  • expensive – thus hardly ever done for colors

Lighting/Shading IV
• linearly interpolating surface normal across the facet, applying Phong lighting model at every pixel
  • good to use fractions/trig functions as intermediate values to show work
  • but final answer should be decimal number
  • allowed during midterm
  • calculator
  • one notes page, 8.5"x11" , one side of page
  • your name at top, hand in with midterm, will be handed back
  • must be handwritten

Shading Artifacts
• perspective transformations
  • affine combinations only invariant under affine, not under perspective transformations
  • thus, perspective projection alters the linear interpolation!

Sect 13.3 Scene Graphs

Gouraud Shading Artifacts
• perspective transformation problem
  • colors slightly "swim" on the surface as objects move relative to the camera
  • usually ignored since often only small difference
    • usually smaller than changes from lighting variations
  • to do it right
    • either shading in object space
    • or correction for perspective foreshortening
  • expensive – thus hardly ever done for colors

Phong Shading
• linearly interpolate the vertex normals
  • compute lighting equations at each pixel
  • can use specular component
  \[ I_{\text{shiny}} = \sum_{i=1}^{#\text{lights}} k_s (\mathbf{n} \cdot \mathbf{r}_i) \]
  \[ \mathbf{r}_i = \mathbf{l}_i \times \mathbf{v}_i \]
  \[ \mathbf{n} = \text{normalize}(\mathbf{n}) \]

Shading Artifacts
• perspective transformation problem
  • colors slightly "swim" on the surface as objects move relative to the camera
  • usually ignored since often only small difference
    • usually smaller than changes from lighting variations
  • to do it right
    • either shading in object space
    • or correction for perspective foreshortening
  • expensive – thus hardly ever done for colors

Midterm
• for all homeworks+exams
  • good to use fractions/trig functions as intermediate values to show work
  • but final answer should be decimal number
  • allowed during midterm
  • calculator
  • one notes page, 8.5"x11" , one side of page
  • your name at top, hand in with midterm, will be handed back
  • must be handwritten

Gouraud Shading Artifacts
• perspective transformation problem
  • affine combinations only invariant under affine, not under perspective transformations
  • thus, perspective projection alters the linear interpolation!

Sect 13.3 Scene Graphs

Gouraud Shading Artifacts
• perspective transformation problem
  • colors slightly "swim" on the surface as objects move relative to the camera
  • usually ignored since often only small difference
    • usually smaller than changes from lighting variations
  • to do it right
    • either shading in object space
    • or correction for perspective foreshortening
  • expensive – thus hardly ever done for colors

Phong Shading
• linearly interpolate the vertex normals
  • compute lighting equations at each pixel
  • can use specular component
  \[ I_{\text{shiny}} = \sum_{i=1}^{#\text{lights}} k_s (\mathbf{n} \cdot \mathbf{r}_i) \]
  \[ \mathbf{r}_i = \mathbf{l}_i \times \mathbf{v}_i \]
  \[ \mathbf{n} = \text{normalize}(\mathbf{n}) \]
Shading Artifacts: Orientation
- interpolation dependent on polygon orientation
- view dependence!

Shading Artifacts: Shared Vertices
- vertex B shared by two rectangles on the right, but not by the one on the left
- first portion of the scanline is interpolated between DE and AC
- second portion of the scanline is interpolated between BC and GH
- a large discontinuity could arise

Shading Models Summary
- flat shading
  - compute Phong lighting once for entire polygon
- Gouraud shading
  - compute Phong lighting at the vertices and interpolate lighting values across polygon
- Phong shading
  - compute averaged vertex normals
  - interpolate normals across polygon and perform Phong lighting across polygon

Computing Normals
- per-vertex normals by interpolating per-facet normals
- OpenGL supports both
- computing normal for a polygon
  - three points form two vectors
  - computing intersection of vectors
  - normalize to unit length
- which side is up?
  - convention: points in counterclockwise order

Global Illumination Models
- simple lighting/shading methods simulate local illumination models
- no object-object interaction
- global illumination models
  - more realism, more computation
  - leaving the pipeline for these two lectures!
- approaches
  - ray tracing
  - radiosity
  - photon mapping
  - subsurface scattering

Ray Tracing
- simple basic algorithm
- well-suited for software rendering
- flexible, easy to incorporate new effects
  - Turner Whitted, 1990

Simple Ray Tracing
- view dependent method
  - cast a ray from viewer's eye through each pixel
  - compute intersection of ray with first object in scene
  - cast ray from intersection point on object to light sources

Reflection
- mirror effects
  - perfect specular reflection

Refraction
- happens at interface between transparent object and surrounding medium
  - e.g. glass/air boundary
  - Snell's Law
    - $c_1 \sin \theta_1 = c_2 \sin \theta_2$
    - light ray bends based on refractive indices $c_1$, $c_2$
Ray Tracing

Recursive Ray Tracing
• ray tracing can handle
  • reflection (chrome/mirror)
  • refraction (glass)
  • shadows
  • spawn secondary rays
    • reflection, refraction
    • if another object is hit, recurse to find its color
  • shadow
    • cast ray from intersection point to light source, check if intersects another object

Basic Algorithm
for every pixel \( p_i \) { 
  generate ray \( r \) from camera position through pixel \( p_i \) for every object \( o \) in scene { 
    if \( r \) intersects \( o \) 
      compute lighting at intersection point, using local normal and material properties; store result in \( p_i \)
    else 
      \( p_i \)= background color
  }
}

Ray Tracing Algorithm
RayTrace(\( r \), scene) 
  obj := FirstIntersection(\( r \), scene) 
  if (no obj) return BackgroundColor; else begin 
    if (Reflect(obj)) then 
      \( \text{reflect \_color} \) = RayTrace(ReflectRay(\( r \), obj)); 
    else 
      \( \text{reflect \_color} \) = Black; 
    \( \text{return Shade} \) (\( \text{reflect \_color, reflect \_color, obj} \)); 
  end;

Algorithm Termination Criteria
• termination criteria
  • no intersection
  • reach maximal depth
  • number of bounces
  • contribution of secondary ray attenuated below threshold
  • each reflection/refraction attenuates ray
  • need to limit maximum depth of ray tree to ensure termination of ray-tracing process!

Ray Generation
• camera coordinate system
  • origin: \( C \) (camera position)
  • viewing direction: \( v \)
  • up vector: \( u \)
  • x direction: \( x= v \times u \)
• note:
  • corresponds to viewing transformation in rendering pipeline
  • like gluLookAt

Algorithm Generation
• other parameters:
  • distance of camera from image plane: \( d \)
  • image resolution (in pixels): \( w, h \)
  • left, right, top, bottom boundaries in image plane: \( i, r, t, b \)
  • then:
  • lower left corner of image: \( O = C + d \cdot v + i \cdot x + b \cdot u \)
  • pixel at position \( i, j \) \((i=0, w-1, j=0, h-1)\):
    \( P_{ij} = O + i \cdot x + j \cdot y \)

Ray Trees
• all rays directly or indirectly spawned off by a single primary ray

Ray Trees Terminology
• terminology:
  • primary ray: ray starting at camera
  • shadow ray
  • reflected/refracted ray
• ray tree: ... by a single primary ray

Ray Intersections: Spheres
• to determine intersection:
  • insert ray \( R_s(t) \) into \( S(x,y,z) \):
    \( (c_x + t \cdot v_x)^2 + (c_y + t \cdot v_y)^2 + (c_z + t \cdot v_z)^2 = r^2 \)
  • solve for \( t \) (find roots)
    • simple quadratic equation

Ray Intersections: Other Primitives
• implicit functions
  • spheres at arbitrary positions
  • conic sections (hyperboloids, ellipsoids, paraboloids, cones, cylinders)
• polytopes
  • convex polygons
  • implicit functions
  • solve for \( t \) (find roots)
  • simple quadratic equation

Ray Intersections
• to determine intersection:
  • insert ray \( R_s(t) \) into \( S(x,y,z) \):
  • \( (c_x + t \cdot v_x)^2 + (c_y + t \cdot v_y)^2 + (c_z + t \cdot v_z)^2 = r^2 \)
  • solve for \( t \) (find roots)
  • simple quadratic equation

Ray Tracing
• issues:
  • generation of rays
  • intersection of rays with geometric primitives
  • geometric transformations
  • lighting and shading
  • efficient data structures so we don’t have to test intersection with every object

Ray Tracing
• issues:
  • generation of rays
  • intersection of rays with geometric primitives
  • geometric transformations
  • lighting and shading
  • efficient data structures so we don’t have to test intersection with every object

Ray - Object Intersections
• inner loop of ray-tracing
• task: given an object \( o \), find ray parameter \( t \), such that \( R_s(t) \) is a point on the object
• such a value for \( t \) may not exist
• solve a set of equations
• intersection test depends on geometric primitive
  • ray-sphere
  • ray-triangle
  • ray-polygon