Lighting, Illumination, and Shading

CPSC 314

The Rendering Pipeline

Geometry Processing

Geometry Database → Model/View Transform. → Lighting → Perspective Transform. → Clipping

Rasterization

Scan Conversion → Texturing → Depth Test → Blending → Frame-buffer

Fragment Processing
Illumination

Goal
Model interaction of light with matter in a way that appears realistic and is fast

• Phenomenological reflection models
  – Ignore real physics, approximate the look
  – Simple, non-physical
  – Phong, Blinn-Phong
• Physically based reflection models
  – Simulate physics
  – BRDFs: Bidirectional Reflection Distribution Functions

Components of Illumination

Two components
• Light sources and surface properties

Light sources (or emitters)
• Spectrum of emittance (i.e., color of the light)
• Geometric attributes
  – Position
  – Direction
  – Shape
• Directional attenuation
• Polarization
Components of Illumination

**Surface properties**
- Reflectance spectrum (i.e., color of the surface)
- Subsurface reflectance
- Geometric attributes
  - Position
  - Orientation
  - Micro-structure

Light Transport Assumptions

**Geometrical optics:**
- Light is photons not waves)
- No diffraction
- No polarization (some sunglasses)
  - Light of all orientations gets through
- No interference (packets don’t interact)
  - Which visual effects does this preclude?
Light Transport Assumptions II

**Color approximated by discrete wavelengths**
- Quantized approx of dispersion (rainbows)
- Quantized approx of fluorescence (cycling vests)

**No propagation media (surfaces in vacuum)**
- No
  - Atmospheric scattering (fog, clouds)
  - Refraction (mirages)
  - Gravity lenses
- But methods exist for all these effects

**Superposition (lights can be added)**
- No nonlinear reflection models
  - Pretty good assumption (only few non-linear materials)

Light Sources and Materials

**Appearance depends on**
- Light sources, locations, properties
- Material (surface) properties
- Viewer position

**Local illumination**
- Compute at material, from light to viewer

**Global illumination (later in course)**
- Ray tracing: from viewer into scene
- Radiosity: between surface patches
Illumination in the Rendering Pipeline

**Local illumination**
- Only models light arriving directly from light source
- No interreflections and shadows
  - Can be added through tricks, multiple rendering passes

**Light sources**
- Simple shapes

**Materials**
- Simple, non-physical reflection models

Light Sources

**Types of light sources**
- Directional/parallel lights
  - E.g. sun
  - Homogeneous vector
- (Homogeneous) point lights
  - Same intensity in all directions
  - Homogeneous point
- Spot lights
  - Limited set of directions
  - Point + direction + cutoff angle
Light Sources

**Area lights:**
- Light sources with a finite area
- Can be considered a continuum of point lights
- Not available in many rendering systems

Light Sources

**Ambient lights**
- No identifiable source or direction
- Hack for replacing true global illumination
  - (light bouncing off from other objects)
Physics of Diffuse Reflection

**Ideal diffuse reflection**

- Very rough surface at the microscopic level
  - *Real-world example: chalk*
- Microscopic variations mean incoming ray of light equally likely to be reflected in any direction over the hemisphere
- Reflected intensity only depends on light direction!

Lambert’s “Law”

Intuitively: cross-sectional area of the “beam” intersecting an element of surface area is smaller for greater angles with the normal.
**Phong Lighting**

*Most common lighting model in computer graphics*

- (Phong Bui-Tuong, 1975)

\[ I_{\text{specular}} = k_s I_{\text{light}} \cos^n \phi \]

- \( n_{\text{shiny}} \): purely empirical constant, varies rate of falloff
- \( k_s \): specular coefficient, highlight color
- no physical basis, works ok in practice

---

**Phong Examples**

- varying \( I \)
- varying \( n_{\text{shiny}} \)
Calculating Phong Lighting

compute cosine term of Phong lighting with vectors

\[ I_{\text{specular}} = k_s I_{\text{light}} (v \cdot r)^n_{\text{shiny}} \]

- \( v \): unit vector towards viewer/eye
- \( r \): ideal reflectance direction (unit vector)
- \( k_s \): specular component
  - highlight color
- \( I_{\text{light}} \): incoming light intensity

**how to efficiently calculate \( r \)?**

Shading

**Lighting:**

- We now have an illumination model for a point on a surface

**If surface defined as mesh of polygonal facets, which points should we use?**

- Fairly expensive calculation
- Several possible answers, each with different implications for visual quality of result
Flat Shading

- Simplest approach calculates illumination at a single point for each polygon
- Obviously inaccurate for smooth surfaces

Flat Shading Approximations

If an object really is faceted, is this accurate?

no!

- For point sources, the direction to light varies across the facet
- For specular reflectance, direction to eye varies across the facet
Improving Flat Shading

What if evaluate Phong lighting model at each pixel of the polygon?

- Better, but result still clearly faceted

For smoother-looking surfaces we introduce vertex normals at each vertex

- Usually different from facet normal
- Used only for shading
- Think of as a better approximation of the real surface that the polygons approximate

Vertex Normals

Vertex normals may be

- Provided with the model
- Computed from first principles
- Approximated by averaging the normals of the facets that share the vertex
**Gouraud Shading**

*Most common approach, and what OpenGL does*

- Perform Phong lighting at the vertices
- Linearly interpolate the resulting colors over faces
  - Along edges
  - Along scanlines

Does this eliminate the facets?

interior: mix of $c_1$, $c_2$, $c_3$

edge: mix of $c_1$, $c_2$

does this eliminate the facets?

edge: mix of $c_1$, $c_3$

**Gouraud Shading Artifacts**

*Often appears dull, chalky*

*lacks accurate specular component*

- if included, will be averaged over entire polygon

this interior shading missed!

this vertex shading spread over too much area

© Wolfgang Heidrich
Gouraud Shading Artifacts

*Mach bands*

- Eye enhances discontinuity in first derivative
- Very disturbing, especially for highlights

Discontinuity in rate of color change occurs here
Gouraud Shading Artifacts

Perspective transformations

- Affine combinations only invariant under affine, not under perspective transformations
- Thus, perspective projection alters the linear interpolation!

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

© Wolfgang Heidrich
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

Linearily interpolate the vertex normals

• Compute lighting equations at each pixel
• Can use specular component

\[
I_{\text{total}} = k_a I_{\text{ambient}} + \sum_{i=1}^{\#\text{lights}} I_i \left( k_d (\mathbf{n} \cdot \mathbf{l}_i) + k_s (\mathbf{v} \cdot \mathbf{r}_i)^n_{\text{shiny}} \right)
\]

remember: normals used in diffuse and specular terms

discontinuity in normal’s rate of change harder to detect
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 hardware**
- But can be simulated with texture mapping

---

Shading Artifacts: Silhouettes

**Polygonal silhouettes remain**

![Gouraud vs. Phong Silhouettes](Image)
Shading Artifacts: Orientation

Interpolation dependent on polygon orientation

- View dependence!

Interpolate between AB and AD

Rotate -90° and color same point

Interpolate between CD and AD

Shading Artifacts

View dependence:

- This problem only applies to polygons with more than 3 vertices (quadrilaterals and more)
- For triangles, this interpolation procedure is identical to computing Barycentric coordinates!
- This is why OpenGL subdivides quadrilaterals into triangles
Barycentric Coordinates

• Convex combination of 3 points

\[ \mathbf{x} = \alpha \cdot \mathbf{x}_1 + \beta \cdot \mathbf{x}_2 + \gamma \cdot \mathbf{x}_3 \]

with \( \alpha + \beta + \gamma = 1, \ 0 \leq \alpha, \beta, \gamma \leq 1 \)

• \( \alpha, \beta, \) and \( \gamma \) are called barycentric coordinates

Another way to compute them:

\[ \mathbf{x} = \alpha \mathbf{x}_1 + \beta \mathbf{x}_2 + \gamma \mathbf{x}_3 \]

with

\[ \alpha = A_1 / A \]
\[ \beta = A_2 / A \]
\[ \gamma = A_3 / A \]
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
The Rendering Pipeline

Geometry Database → Model/View Transform → Lighting → Perspective Transform → Clipping

Geometry Processing

Scan Conversion → Texturing → Depth Test → Blending → Framebuffer

Rasterization → Fragment Processing

Coming Up

Monday:
- Victoria Day

Wednesday:
- Polygon Clipping / Visibility