Shading

- Shading is the means by which we transform colored polygons and other graphical elements into gradations of colors that represent meaning such as shape, material properties and more.
- Rendering is the process of producing an image from a modeled scene using shading and other techniques.
Why we need shading

• Suppose we build a model of a sphere using many polygons and color each the same color. We get something like

• But we want

Shading

• Why does the image of a real sphere look like

• Light-material interactions cause each point to have a different color or shade
• Need to consider
  - Light sources
  - Material properties
  - Location of viewer
  - Surface orientation (normal)
Scattering

• Light strikes A
  - Some scattered
  - Some absorbed
• Some of scattered light strikes B
  - Some scattered
  - Some absorbed
• Some scattered light strikes A
  and so on

Scattering

• Could define the infinite scattering and absorption of light, and is described by the so-called rendering equation
• Rendering equation is global and includes
  - Shadows
  - Multiple scattering from object to object
• Cannot (yet?) be solved in general
• Ray-tracing is one local simplification appropriate for reflective surfaces
Global Effects

Local vs Global Rendering

• Correct shading requires a global calculation involving all objects and light sources
  - Incompatible with pipeline model which shades each polygon independently (local rendering)
• However, in computer graphics, especially real time graphics, we are happy if things “look right”
  - Use any of many techniques for approximating global effects
Light-Material Interaction

- Light that strikes an object is partially absorbed and partially scattered (reflected)
- The light reflected determines the color and brightness of the object
  - A surface appears red under white light because the red component of the light is reflected and the rest is absorbed
- The reflected light is scattered in a manner that depends on the smoothness and orientation of the surface

Ambient Light

- Ambient light is the result of multiple interactions between (large) light sources and the objects in the environment
- Amount and color depend on both the color of the light(s) and the material properties of the object

\[ L_v = k_a L_a \]

Coefficients, \( k_{r,b,g} \) for each color component

ambient reflection coef intensity of ambient light
Surface Types

- The smoother a surface, the more reflected light is concentrated in the direction a perfect mirror would reflect the light.
- A very rough surface scatters light in all directions.

- smooth surface
- rough surface

Lambertian Surface

- Perfectly diffuse reflector
- Light scattered equally in all directions
- Amount of light reflected is proportional to the vertical component of incoming light.

- reflected light is proportional to the angle of incidence or $\sim \cos \theta_i$
- So, $L_v = k_d \cos \theta_i L_d$

- Note, doesn’t matter where the viewer is (same true for ambient)
Ideal Reflector (Mirror)

- Normal is determined by local orientation
- Angle of incidence = angle of reflection

Note, vectors $l, n, r$ are coplanar

- Here, viewer position matters. If viewer is not in the direction of the reflection, it is not seen by the viewer

Specular Surfaces

- Most surfaces are neither ideal diffusers nor perfectly reflective (ideal mirrors)
- But many surfaces show specular highlights where reflected light is concentrated in the direction close to that of the perfect reflection

- Here again, viewer position matters
Phong Model

- A simple model that can be computed rapidly
- Includes three components
  - Diffuse
  - Specular
  - Ambient
- Uses four vectors
  - To source, I
  - To viewer, v
  - Normal, n
  - Perfect reflector, r

Modeling Specular Reflections

- Phong proposed using a term that dropped off as the angle between the viewer and the ideal reflection increased

\[ L_s = k_s L_i \cos^a \phi \]

- Specular intensity
- Shininess coefficient (shininess coef)
- Incoming intensity
- Specular reflect coeff (a.k.a. absorption coeff)
The Shininess Coefficient

- Values of $\alpha$ between 100 and 200 correspond to metals
- Values between 5 and 10 give surface that look like plastic

Adding up the Components

The Phong model can be written as

$$L = k_d L_d \cos \theta + k_s L_s \cos^\alpha \phi + k_a L_a$$

Replacing angles with dot product of vectors:

$$L = k_d L_d \mathbf{l} \cdot \mathbf{n} + k_s L_s (\mathbf{v} \cdot \mathbf{r})^\alpha + k_a L_a$$

Note, $I$ is sometimes used in place of $L$ for illumination.
Material Properties

- Material properties match light source properties
  - Nine absorption coefficients
    - $k_{dr}, k_{dg}, k_{db}, k_{sr}, k_{sg}, k_{sb}, k_{ar}, k_{ag}, k_{ab}$
  - Shininess coefficient $\alpha$

Light Sources

- In the Phong Model, we add the results from each light source
- Each light source has separate diffuse, specular, and ambient terms to allow for maximum flexibility even though this form does not have a physical justification
- Separate red, green and blue components
- Hence, 9 coefficients for each light
  - $L_{vd, r}g^t b^t$, $L_{vs, r} g^t b^t$, $L_{va, r} g^t b^t$
Simple Light Sources

- **Point source**
  - Model with position and color
- **Spotlight**
  - Restrict light from ideal point source (+more)
- **Directional source**
  - Point at infinite distance away (parallel)
- **Ambient light**
  - Same amount of light everywhere in scene
  - Can model contribution of many sources and reflecting surfaces

Light Sources

Modeling real-world light sources is difficult because we must consider light coming from all points on the source:
Example

Only differences in these teapots are the parameters in the Phong model.

Alternative to Phong Shading

The Phong model can be written as

\[ L = k_d L_d \mathbf{l} \cdot \mathbf{n} + k_s L_s (\mathbf{v} \cdot \mathbf{r})^\alpha + k_a L_a \]
**Alternative to Phong Shading**

Starting with the Phong model

\[ L = k_d L_d \cdot n + k_s L_s (v \cdot r) + k_a L_a \]

Replace

\[ (v \cdot r) \text{ with } (h \cdot n) \]

where \( h \) is "halfway" vector:

\[ h = \frac{l + v}{|l + v|} \]

---

**Alternative to Phong Shading**

The Phong model can be re-written as

\[ L = k_d L_d \cdot n + k_s L_s (h \cdot n) + k_a L_a \]

Assumes the viewer is at an infinite distance and the lights are as well (which is true for directional lights) which are the defaults for OpenGL.
Efficiency with "halfway" vector:

- $h$ is independent of position and surface curvature so calculate once for each light, use for entire frame (or longer)
- but $r$ depends on the normal and must be recalculated for each pixel of the image

$\text{OpenGL computes shading in this fashion}$
Shading Polygons

- Polygons often approximate curve surfaces but are inherently flat
- Consider polygonal ‘sphere’
- Want to smooth the rough face of each surface facet
- How do we fix this?

Smooth Shading

- We can simply find a new normal at each vertex for a sphere
- Results in smoother shading
- Note silhouette edge
Vertex Normals

- For polygonal meshes, Gouraud in 70's proposed we use the average of normals around a mesh vertex

\[ n = \frac{n_1 + n_2 + n_3 + n_4}{|n_1| + |n_2| + |n_3| + |n_4|} \]

Gouraud Smoothing

- AKA: *intensity interpolation* shading
- Find vertex normals
- Apply Phong light model at each vertex
- Interpolate vertex shades across each polygon
- Used in OpenGL
Phong Smoothing
- Not the Phong lighting model!
- AKA: normal-vector interpolation shading
- Find vertex normals
- Interpolate vertex normals across edges and then across polygon
- Find shades using normals across polygons

Interpolating Over Polygons
- Given values at vertices of polygon, how do we interpolate data over interior?
  - values could be either normal or color
Comparison

• If the polygon mesh approximates surfaces with high curvatures, Phong smoothing may look smooth when Gouraud shows edges.

• Phong smoothing requires much more work than Gouraud smoothing.

• Both leave the silhouette jagged.
Surface Mapping aka Surface Rendering

• Introduce Mapping Methods
  - Texture Mapping
  - Environmental Mapping
  - Bump Mapping

• Go over strategies for
  - Forward vs backward mapping

The Limits of Geometric Modeling

• Although graphics cards can render over 10 million polygons per second, that number is insufficient for many phenomena
  - Clouds
  - Grass
  - Terrain
  - Hair
Three Types of Mapping

- **Texture Mapping**
  - Uses images to fill inside of polygons
- **Environmental (reflection mapping)**
  - Uses a picture of the environment for texture maps
  - Allows simulation of highly specular surfaces
- **Bump mapping**
  - Emulates altering normal vectors during the rendering process

Texture Mapping

generic model  texture mapped
Environment Mapping

Bump Mapping
Texture Mapping

• Geometry and lighting alone do not provide sufficient visible detail
• “Paste” 2D image onto 3D surface
• Surface appears much more complex than reality
Texture Mapping
Mapping textures to models

• Although the idea is simple---map an image to a surface---there are several tricky steps involved.

![Diagram of mapping textures to models]

- 2D image
- 3D surface

**Parametric coordinates**

**Texture coordinates**

**World coordinates**

**Screen coordinates**
Mapping textures to models

• How do we find mapping?
• Consider going from texture coordinates to a point a surface
• Must define mapping

\[ x = x(s,t) \]
\[ y = y(s,t) \]
\[ z = z(s,t) \]

Texture "Painting"

• 3D Paint packages establishes an arbitrary mapping (model to map)
• Then records the texture as the user "paints" on the model
Rendering Textures

- In rendering, we perform **backward mapping**
  - Given a pixel, we want to know which point on an object it corresponds to and
  - Given a point on an object, we want to know which point in the texture it corresponds to:
    - Map of the form
      \[ s = s(x,y,z) \]
      \[ t = t(x,y,z) \]
    - These functions are difficult to find in general

Simple Backward mapping

- Example: map to cylinder
Simple Backward mapping

Parametric cylinder

\[ x = r \cos 2\pi u \]
\[ y = r \sin 2\pi u \]
\[ z = v/h \]

Maps rectangle in \( u, v \) space to cylinder of radius \( r \) and height \( h \) in world coordinates

Thus,

\[ s = u \]
\[ t = v \]

maps from the texture space

Simple Backward mapping

- A direct solution to the simple mapping problem can be found analytically
- Example: mapping a cube
Two-part (Backward) mapping

- One solution to the general mapping problem is a two-part mapping: first map the texture to a simple intermediate surface (like the cylinder).
- Then map to the object of interest.

Two-Part Mapping

- Second, map from the intermediate object to the actual object by:
  - Normals from intermediate to actual
  - Normals from actual to intermediate
Two-Part Mapping

• In General, no direct method exists

• Best choice of secondary mapping depends on the model being mapped

Sampling
Sampling

• During drawing, look up color from texture using continuous texture coordinates
Sampling

• Linear blending
- Smooth appearance

\[
\begin{align*}
\alpha &- 1 - \beta \\
\beta &- 1 - \alpha \\
\alpha \beta &
\end{align*}
\]
Other Uses of Texture Mapping

- Environment Mapping
- Bump/Normal Mapping
- Displacement Mapping
- ...

Any attribute of the surface position, normal, color, etc... can be placed in a texture

Environment Mapping

- Cheap attempt at modeling reflections
- Makes surfaces look metallic

- Use six textures to model faces of a cube
- Assume cube faces infinitely far away
- The normal (or reflected eye vector) is used to find which of the textures to use and what texture coordinate
Environment Mapping

Environment Mapping
Environment Mapping

Reflected ray: \( r = 2(n \cdot v)n - v \)

viewer

\( n \)

reflective surface

\( r \)

environment texture image
Environment Mapping

- Easy to use with simple projection for a box but can cause distortions
Bump/Normal Mapping

- Replace colors R,G,B with coordinates X,Y,Z
- Interpret pixels as normal vectors
- Makes the shading look more complicated than geometry really is

After bump mapping

Bump Mapping (2D Example)

- Surface with normals
- Bump texture
- "New" surface with texture
Bump Mapping

No bump mapping  With bump mapping

Bump/Normal Mapping Example
Bump/Normal Mapping Example
Bump Mapping combined with Texture

Bump Mapping combined with Texture
Displacement Mapping

- Offset geometry in direction of normal
- Encode offset inside texture
- Used to actually change the geometry and provide more detail (especially silhouette)

Bump/Normal Mapping Example
Displacement Mapping Example
More Examples

More Examples
Problems with Displacement Maps

Actually move geometry based on texture map – how is still open question in general

Expensive and difficult to implement in many rendering pipelines