5. Shading

Reading

Recommended:

- Hearn & Baker, Sections 14.1–2, 14.8–9
- Foley, et al., 16.1–5
- Neider, Davis & Woo, Chapter 9
- James F. Blinn and Martin E. Newell. Texture and reflection in computer generated images. *Communications of the ACM* 19(10): 542–547, October 1976.
- Paul S. Heckbert. Survey of texture mapping. *IEEE Computer Graphics and Applications* 6(11): 56–67, November 1986.
- William T. Reeves, David H. Salesin, and Robert L. Cook. Rendering antialiased shadows with depth maps. *Computer Graphics* (SIGGRAPH '87) 21(4): 283–291, July 1987.
- Jody Duncan Shay. Young Sherlock Holmes: Anything but elementary. *Cinefex* 26: 36–67, May 1986.

Introduction

In these notes, we'll build up to the "Phong illumination model":

- Not physically-based
- Gives a first-order approximation to physical light reflection
- Very fast
- Widely used

Later we'll look at other "fakes" you can add for:

- Textures
- Shadows
- Transparency
- Reflection and refraction

The problem

Given:

- A point P on a surface
- Visible through pixel p
- A unit vector $\mathbf{V}$ from P to the viewer
- A unit vector $\mathbf{L}$ from P to a point light source

Find: The intensity and color of light radiating from P to the viewer.

Solved by developing a model of light interaction with materials.

This model is known as:

- A "shading model"
- A "lighting model"
- A "light reflection model"
- A "local illumination model"
- A "reflectance model"

Ambient reflection

The simplest shading rule is to just give each polygon a single color:

- A common solution in 2D drawing programs
- Can also use in 3D

Gives the shading equation:

$$I = k_i$$

where:

- I is the resulting intensity
- k_i is the intrinsic shade associated with the object

Alternate characterization

The same shading rule has an alternate characterization:

Suppose that the only light is *diffuse* and *nondirectional* – for example:

- Object suspended in a room coated with latex paint
- Light source is not directly visible to the object

Alternate characterization, cont.

In this case, light reflects from the surface according to its "reflectivity" k_a.

Gives the new ambient shading equation (14-1):

$$I = k_a I_a$$

where:

- k_a is the "ambient-reflection coefficient"
  - Really, the "reflectance of ambient light"
  - This reflectance is assumed to be "diffuse" — equal in all directions
- I_a is the "ambient intensity"
  - Really, the "intensity of the ambient light"

Wavelength dependance

Really, k_a and I_a are functions over all wavelengths λ.

Ideally, we should do the calculation on these functions:

$$I(\lambda) = k_a(\lambda)\, I_a(\lambda)$$

and then find good RGB values to respresent $I(\lambda)$.

"Traditionally," though, k_a and I_a are represented in RGB, and the computation is performed on each color component separately:

$$I_{\text{red}} = k_{a,\text{red}}\, I_{a,\text{red}}$$

$$I_{\text{green}} = k_{a,\text{green}}\, I_{a,\text{green}}$$

$$I_{\text{blue}} = k_{a,\text{blue}}\, I_{a,\text{blue}}$$

Diffuse reflection

Review of ambient shading model:

- Objects have different colors
- We can control the overall light intensity
  - What happens when we turn off the lights?
  - What happens as the light intensity increases?
  - What happens if we change the color of the lights?

Problem: So far, objects are all uniformly lit.

- Not the way things usually appear
- Due to the existence of "directional light sources"

"Diffuse" or "Lambertian" reflection will allow reflected intensity to vary with the direction of the light.

Diffuse reflectors

Diffuse reflection occurs from dull, matte surfaces, like latex paint or chalk.

These "diffuse" or "Lambertian reflectors" reradiate light equally in all directions.

Picture a rough surface with lots of tiny "microfacets":

The microfacets have the effect of distributing light rays in all directions.

Note:

- Light may actually penetrate the surface, bounce around, and then reflect back out.
- Accounts for colorization of diffusely reflected light by plastics

Diffuse reflectors, cont.

While the reflected intensity from a diffuse surface does not depend on the direction of the viewer, the incoming energy does depend on the direction of the light source:

An illuminated area (H&B, fig 14-8)

Diffuse reflectors, cont.

The incoming energy is proportional to $\cos\theta$, giving the diffuse reflection equations (14-2), (14-3):

$$\begin{aligned} I &= k_d I_\ell \cos\theta \\ &= k_d I_\ell (\mathbf{N} \cdot \mathbf{L})_+ \end{aligned}$$

where:

- k_d is the "diffuse reflection coefficient"
- I_ℓ is the intensity of the light source
- $\mathbf{N}$ is the normal to the surface
- $\mathbf{L}$ is the direction of the light source
- $(x)_+$ means $\max\{0, x\}$

Intensity drop-off with distance

The laws of physics state that the intensity of a point light source must drop off with its distance squared.

We can incorporate this effect by multiplying I_ℓ by $1/d^2$.

Sometimes, this distance-squared drop-off is considered "too harsh." Hearn & Baker suggest using

$$f(d) \;=\; \min\left\{1,\; \frac{1}{a_0 + a_1 d + a_2 d^2}\right\}$$

for user-supplied constants a_0, a_1, a_2.

Specular reflection

"Specular reflection" accounts for the highlight that you see on objects.

- Particularly important for smooth, shiny surfaces, such as:
  - · Metals
  - · Polished rocks
  - · Plastics
  - · Safeway apples

Properties:

- Specular reflection depends on the viewing direction $\mathbf{V}$
- Often determined soley by the color of the light
  - · Corresponds to absence of internal reflections

"Derivation"

(H&B, fig 14-12)

For a perfect mirror reflector, light is reflected about $\mathbf{N}$. So

$$I \;=\; \begin{cases} I_\ell & \text{if } \mathbf{V} = \mathbf{R} \\ 0 & \text{otherwise} \end{cases}$$

For a near-perfect reflector, you might expect the highlight to fall off quickly with increasing angle ϕ.

Known as:

- "Specular" reflection (Phong)
- "Rough specular" reflection
- "Directional diffuse" reflection
- "Glossy" reflection

"Derivation," cont.

(H&B, fig 14-13)

One way to get this effect is to take $(\mathbf{R} \cdot \mathbf{V})$, raised to a power n_s.

As n_s gets larger,

- The drop-off becomes {more, less} gradual
- Gives a {larger, smaller} highlight
- Simulates a {more, less} glossy surface

Thus, the specular component is given by (14-6):

$$I \;=\; k_s\, I_\ell\, (\mathbf{V} \cdot \mathbf{R})_+^{n_s}$$

The Phong shading model

Since light is additive, we can handle multiple lights by taking the sum over every light.

The Phong shading model can be summarized by putting all the terms we have seen together (14-13):

$$I \;=\; k_a\, I_a \;+\; \sum_i f(d_i)\, I_{\ell i}\, [k_d\, (\mathbf{N} \cdot \mathbf{L}_i)_+ \;+\; k_s\, (\mathbf{V} \cdot \mathbf{R}_i)_+^{n_s}]$$

Note: $k_a, k_d, k_s, I_a, I_{\ell i}, I$ are all RGB vectors.

Choosing the parameters

Experiment with different parameters settings. To get you started, here are a few suggestions:

- Try n_s in the range [0, 100]
- Try $k_a + k_d + k_s \leq 1$
- Use a small k_a (~ 0.1)

	n_s	k_d	k_s
Metal	large	small, color of metal	large, color of metal
Plastic	medium	medium, color of plastic	medium, white
Lunar dust	0	large, white	0

Texture mapping

Texture mapping (Neider et al., fig 9-1)

Texture mapping allows you to take a simple polygon and give it the appearance of something much more complex.

- Due to Ed Catmull, PhD thesis, 1974
- Refined by Blinn & Newell, 1976

Texture mapping ensures that "all the right things" happen as a textured polygon is transformed and rendered.

Applications

The applications of texture mapping are endless:

- Image warping
- Transparency mapping
- Painting
- Surface trimming
- Solid textures for marble, wood, etc.
- Computing shadows
- Volume rendering

Non-parametric texture mapping

With "non-parametric texture mapping":

- Texture size and orientation are fixed
- They are unrelated to size and orientation of polygon
- Gives cookie-cutter effect

Parametric texture mapping

Parametric mapping (Neider et al., fig 9-2)

With "parametric texture mapping," texture size and orientation are tied to the polygon.

Idea:

- Separate "texture space" and "screen space"
- Texture the polygon as before, but in texture space
- Deform the textured polygon into screen space

Implementing texture mapping

Method:

- Scan conversion is done in screen space, as usual
- Each pixel is colored according to the texture
- Texture coordinates are found by Gouraud-style interpolation

Note: Mapping is more complicated if you want to do perspective right!

Antialiasing

If you point-sample the texture map, you get aliasing:

Heckbert, fig 5(a)

Proper antialiasing requires area averaging in the texture:

Heckbert, fig 5(e)

Computing the average color

The computationally difficult part is summing over the covered pixels.

Several methods have been used:

1. Brute force:
   - Just sum
   - (Original method)

Computing the average color

Mip maps (Neider et al., fig 9-3)

2. "Mip maps":
   - Lance Williams, 1983
   - Stands for "multum in parvo" — many things in a small place
   - Keep textures prefiltered at multiple resolutions:
   - Figure out closest two levels
   - Linear interpolate between the two
   - Gives artifacts in animations

Computing the average color

3. "Summed area tables":
   - Frank Crow, 1984
   - Keep sum of everything below and to the left
   - Use four table lookups:
   - Requires more memory
   - Works well for animations
   - Gives excessively blurry textures
   - Extensions exist for trimming the covered region

Computing the average color

4. Stochastic sampling:
   - Gives noisy images, unless enough samples are taken
   - An extremely simple and robust technique

Solid textures

Q: What kinds of artifacts might you see from using a wood-grained veneer instead of real wood?

One solution is to use "solid textures":

- Use model-space coordinates to index into a 3D texture
- Like "carving" the object from the material

One difficulty of solid texturing is coming up with the textures. . . .

Bump mapping

Textures can be used for more than just color.

In "bump mapping," a texture is used to perturb the normal:

- The normal is perturbed in each parametric direction according to the partial derivates of the texture:
- These bumps "animate" with the surface

Q: What artifacts in the images would reveal that bump mapping is a fake?

Displacement mapping

In "displacement mapping," a texture is used to perturb the surface geometry itself:

- Gives rounded edges, even on silhouette
- Requires doing hidden surface after shading

Environment mapping

(H&B, fig 14-92)

In "environment mapping" (also known as "reflection mapping"), a texture is used to model an object's environment:

- Rays are bounced off objects into environment
- Color of the environment used to determine color of the illumination
- Really, a simplified form of "ray tracing"
- Environment mapping works well when there is just a single object — or in conjunction with ray tracing

Transparency

- Rays can also be transmitted through a surface, and then refracted into a texture map:

- Textures can also be used to control the transparency of a surface, itself.
  - Gardner has made nice clouds this way.

Combining texture maps

Using texture maps in combination gives even better effects, as *Young Sherlock Holmes* demonstrated. . . .

Shadow algorithms

Computing shadows is analogous to computing hidden surfaces:

- Surfaces in shadow are exactly those surfaces hidden from the light!
- Thus, for every hidden surface algorithm there is a corresponding shadow algorithm

I'll just describe one here. . . .

Depth buffer shadow algorithm

Based on the Z-buffer hidden surface algorithm.

Rendering is done in two passes:

First pass:

- Compute scene from point of view of each light
- For each light, save only closest z in a "depth map":

Depth buffer shadow algorithm, cont.

Second pass:

- Render scene from camera
- For each point on surface:
  - Transform point to coordinate system of each light
  - Compare transformed z against depth map z
  - Surface is in shadow if transformed z is further away
  - Light source contribution is 0 for surfaces in shadow
  - Need to use a slight bias ϵ so that surface does not shadow itself

Depth buffer shadow algorithm, cont.

Antialiasing:

- Vanilla algorithm has considerable aliasing
- Problem can be alleviated by taking several samples over each pixel
- Transform each sample and compare against depth map
- Use percentage closer for attenuation:

- Called "percentage-closer filtering"
- Can get softer-looking shadows by artificially spreading out the samples
- Idea originally tested in Luxo Jr.
- Recently implemented in hardware on SGI machines

Summary

What to take home from this lecture:

1. All about Phong shading:
   - The physical meaning and "derivation" of each term
   - How the terms are computed
   - What effects each term contributes to the image
   - What varying the parameters will do
2. What texture mapping is, and what it's good for.
3. The various approaches to antialiased texture mapping:
   - Brute force
   - Mip maps
   - Summed area tables
   - Stochastic sampling
4. The relationship between shadow algorithms and the hidden surface problem.
5. High-level understanding of the depth buffer shadow algorithm.