Game Programming

Bing-Yu Chen
National Taiwan University
Lighting & Texturing

- Rendering Pipeline
- Illumination Model
- Shading Models
- Texture Mapping
- Environment Mapping
- Bump Mapping
- Shadow Maps
Rendering Pipeline

- Polynomial Evaluator
- Per Vertex Operations & Primitive Assembly
- Rasterization
- Texture Memory
- Per Fragment Operations
- Frame buffer
- Pixel Operations
- Display List
- CPU
Illumination (Shading) Models

- Interaction between light sources and objects in scene that results in perception of intensity and color at eye

- **Local vs. global** models
  - Local: perception of a particular primitive only depends on light sources *directly* affecting that one primitive
    - Geometry
    - Material properties
    - Shadows cast (global?)
  - Global: also take into account *indirect* effects on light of other objects in the scene
    - Light reflected/refracted
    - Indirect lighting
Local vs. Global Models

direct lighting
indirect lighting
The Phong Illumination Model

- A simple model that can be computed rapidly
- Has three components
  - Diffuse
  - Specular
  - Ambient
- Uses four vectors
  - To source
  - To viewer
  - Normal
  - Perfect reflector
Basics of Local Shading

- **Diffuse reflection**
  - light goes everywhere; colored by object color

- **Specular reflection**
  - happens only near mirror configuration; usually white

- **Ambient reflection**
  - constant accounted for other source of illumination

![Image of a cube with different shading components: ambient, diffuse, specular]
Ambient Shading

- add constant color to account for disregarded illumination and fill in black shadows; a cheap hack.
Diffuse Shading

- Assume light reflects equally in all directions
- Therefore surface looks same color from all views; “view independent”
Illumination Models

- **Ambient Light**: \( I = I_a k_a \)
  - \( I_a \): intensity of the ambient light
  - \( k_a \): ambient-reflection coefficient: 0 ~ 1

- **Diffuse Reflection**: \( I = I_p k_d \cos \theta \)
  - \( I_p \): point light source’s intensity
  - \( k_d \): diffuse-reflection coefficient: 0 ~ 1
  - \( \theta \): angle: 0° ~ 90°
Diffuse Reflection

\[ I = I_p k_d (\vec{N} \cdot \vec{L}) \]

- \( \vec{L} \): direction to the light source
- \( \vec{N} \): surface normal
- \( \theta \): angle between \( \vec{N} \) and \( \vec{L} \)
Examples

diffuse-reflection model with different $k_d$

ambient and diffuse-reflection model with different $k_a$

and $I_a = I_p = 1.0, k_d = 0.4$
Light-Source Attenuation

\[ I = I_a k_a + f_{\text{att}} I_p k_d (\vec{N} \cdot \vec{L}) \]

- \( f_{\text{att}} \): light-source attenuation factor
- if the light is a point source

\[ f_{\text{att}} = \frac{1}{d_L^2} \]

where \( d_L \) is the distance the light travels from the point source to the surface

\[ f_{\text{att}} = \min\left(\frac{1}{c_1 + c_2 d_L + c_3 d_L^2}, 1\right) \]
Examples

\[ \frac{1}{d_L^2} \]

\[ \frac{1}{d_L} \]
Colored Lights and Surfaces

- If an object’s **diffuse color** is
  \[ O_d = (O_{dR}, O_{dG}, O_{dB}) \]  then \[ I = (I_R, I_G, I_B) \]

where for the red component

\[ I_R = I_{aR} k_a O_{dR} + f_{att} I_{pR} k_d O_{dR} (\vec{N} \cdot \vec{L}) \]

however, it should be

\[ I_\lambda = I_{a\lambda} k_{a} O_{d\lambda} + f_{att} I_{p\lambda} k_d O_{d\lambda} (\vec{N} \cdot \vec{L}) \]

where \( \lambda \) is the **wavelength**
Diffuse Shading

- For color objects, apply the formula for each color channel separately.
Specular Shading

- Some surfaces have highlights, mirror like reflection; view direction dependent; especially for smooth shiny surfaces
Specular Surfaces

- Most surfaces are neither ideal diffusers nor perfectly specular (ideal refectors)
- Smooth surfaces show specular highlights due to incoming light being reflected in directions concentrated close to the direction of a perfect reflection
Specular Reflection

- $\vec{L}$: direction to the viewpoint
- $\vec{N}$: normal vector
- $\vec{R}$: direction of reflection
- $\vec{V}$: direction to the viewpoint
- $\alpha$: angle
- $\theta$: angle
The Phong Illumination Model

\[ I_\lambda = I_{a\lambda} k_a O_{d\lambda} + f_{att} I_{p\lambda} [k_d O_{d\lambda} \cos \theta + W(\theta) \cos^n \alpha] \]

- \( W(\theta) = k_s \): specular-reflection coefficient: 0~1

- so, the Eq. can be rewritten as

\[ I_\lambda = I_{a\lambda} k_a O_{d\lambda} + f_{att} I_{p\lambda} [k_d O_{d\lambda} (\vec{N} \cdot \vec{L}) + k_s (\vec{R} \cdot \vec{V})^n] \]

- consider the object’s **specular color**

\[ I_\lambda = I_{a\lambda} k_a O_{d\lambda} + f_{att} I_{p\lambda} [k_d O_{d\lambda} (\vec{N} \cdot \vec{L}) + k_s O_{s\lambda} (\vec{R} \cdot \vec{V})^n] \]

- \( O_{s\lambda} \): specular color
The Phong Illumination Model

\[
\begin{align*}
\cos \alpha & \quad 0^\circ \quad 90^\circ \\
\cos^2 \alpha & \quad 0^\circ \quad 90^\circ \\
\cos^4 \alpha & \quad 0^\circ \quad 90^\circ \\
\cos^6 \alpha & \quad 0^\circ \quad 90^\circ 
\end{align*}
\]
Specular Shading

diffuse

diffuse + specular
Calculating the Reflection Vector

- Fall off gradually from the perfect reflection direction

\[
\vec{R} = \vec{N} \cos \theta + \vec{S}
\]

\[
= \vec{N} \cos \theta + \vec{N} \cos \theta - \vec{L}
\]

\[
= 2\vec{N} \cos \theta - \vec{L}
\]

\[
= 2\vec{N}(\vec{N} \cdot \vec{L}) - \vec{L}
\]
The Halfway Vector (Blinn-Phong)

Rather than computing reflection directly; just compare to normal bisection property.

\[ \vec{H} = \frac{\vec{L} + \vec{V}}{|\vec{L} + \vec{V}|} \]

\[ \Rightarrow \cos \alpha \approx \vec{N} \cdot \vec{H} \]
Multiple Light Sources

If there are $m$ light sources, then

$$I_\lambda = I_{a\lambda} O_{d\lambda} + \sum_{1 \leq i \leq m} f_{att, i} I_{p\lambda_i} [k_d O_{d\lambda} (\vec{N} \cdot \vec{L}_i) + k_s O_{s\lambda} \cos^n \alpha_i]$$

$$\approx I_{a\lambda} O_{d\lambda} + \sum_{1 \leq i \leq m} f_{att, i} I_{p\lambda_i} [k_d O_{d\lambda} (\vec{N} \cdot \vec{L}_i) + k_s O_{s\lambda} (\vec{R}_i \cdot \vec{V})^n]$$

$$\approx I_{a\lambda} O_{d\lambda} + \sum_{1 \leq i \leq m} f_{att, i} I_{p\lambda_i} [k_d O_{d\lambda} (\vec{N} \cdot \vec{L}_i) + k_s O_{s\lambda} (\vec{N} \cdot \vec{H}_i)^n]$$
Computing Lighting at Each Pixel

- Most accurate approach: Compute component illumination at each pixel with individual positions, light directions, and viewing directions
- But this could be expensive...

\[ I_1 \quad I_2 \quad I_3 \]

\[ y \]

\[ y_1 \quad y_s \quad y_2 \quad y_3 \]

Scan line
Shading Models for Polygons

- Flat Shading
  - Faceted Shading
  - Constant Shading
- Gouraud Shading
  - Intensity Interpolation Shading
  - Color Interpolation Shading
- Phong Shading
  - Normal-Vector Interpolation Shading
Flat Shading

- Assumptions
  - The light source is at infinity
  - The viewer is at infinity
  - The polygon represents the actual surface being modeled and is not an approximation to a curved surface
Flat Shading

- Compute constant shading function, over each polygon
- Same normal and light vector across whole polygon
- Constant shading for polygon

\[ I_p = I \]
Intensity Interpolation (Gouraud)

\[ I_a = I_1 \frac{y_s - y_2}{y_1 - y_2} + I_2 \frac{y_1 - y_s}{y_1 - y_2} \]

\[ I_b = I_1 \frac{y_s - y_3}{y_1 - y_3} + I_3 \frac{y_1 - y_s}{y_1 - y_3} \]

\[ I_p = I_a \frac{x_b - x_p}{x_b - x_a} + I_b \frac{x_p - x_a}{x_b - x_a} \]
Normal Interpolation (Phong)

\[ N_a = N_1 \frac{y_s - y_2}{y_1 - y_2} + N_2 \frac{y_1 - y_s}{y_1 - y_2} \]

\[ N_b = N_1 \frac{y_s - y_3}{y_1 - y_3} + N_3 \frac{y_1 - y_s}{y_1 - y_3} \]
Gouraud v.s. Phong Shading

Gouraud

Phong

Gouraud

Phong
Shadows

\[ I_\lambda = I_{a\lambda} k_a O_{d\lambda} + \sum_{1 \leq i \leq m} S_i f_{att_i} I_{p\lambda_i} [k_d O_{d\lambda_i} (\vec{N} \cdot \vec{L}_i) + k_s O_{s\lambda_i} (\vec{R}_i \cdot \vec{V})''] \]

\[ S_i = \begin{cases} 
0, & \text{if light } i \text{ is blocked at this point} \\
1, & \text{if light } i \text{ is not blocked at this point} 
\end{cases} \]
The Quest for Visual Realism

Model

Model with Shading

Model with Shading and Textures

Copyright©1997, Jeremy Birn
Previously, we assume that reflection properties such as are constant within each triangle.

However, some objects have complex appearance which arises from variation in reflection properties.

The common technique to handle this kind of variation is to store it as a function or a pixel-based image and “map” it onto a surface.

The function is called *texture map* and the process is called *texture mapping*. 
Texture Maps

Tom Porter’s Bowling Pin
Texture Mapping

geometric model
texture mapped
Texture Maps

- How is texture mapped to the surface?
  - Dimensionality: 1D, 2D (image), 3D (solid)
  - Procedural v.s. table look-up
  - Texture coordinates \((s,t)\)
    - Surface parameters \((u,v)\)
    - Projection: spherical, cylindrical, planar
    - Reparameterization

- What does texture control?
  - Surface color and transparency
  - Illumination: environment maps, shadow maps
  - Reflection function: reflectance maps
  - Geometry: displacement and bump maps
Texture Mapping

2D mapping 3D mapping
Where does mapping take place?

- Mapping techniques are implemented at the end of the rendering pipeline.
  - Very efficient because few polygons pass down the geometric pipeline.

![Diagram of rendering pipeline]

- Vertices → Geometric processing → Rasterization → Display
- Pixels → Pixel operations
Simple Texture Mapping
Texture Mapping = Pattern Mapping

texture map → u
surface of object
four corners of pixel on screen
Parameterization

- A parameterization of a surface is a one-to-one mapping from a suitable domain to the surface.

**parametric surface:**  
\[ Q(s,t) = (x, y, z) \]

**surface parameterization:**  
\[ P(x, y, z) = (s, t) = Q^{-1}(x, y, z) \]
Parameterization

- How to get $P$ from $S$?
  - for each vertex of $S$, find its $(u,v)$
  - from $(u,v)$ of $P$, map image to $S$

- A parameterization of a surface is a mapping $\rho: (x,y,z)\rightarrow(u,v)$ from 3D space to 2D space
Antialiasing
Aliasing

- Point sampling of the texture can lead to aliasing errors

Point samples in u,v (or x,y,z) space
go to miss blue stripes

Point samples in texture space
Magnification and Minification

Example:

Texture Polygon Magnification

Texture Polygon Minification
Changing Resolution

![Image showing the concept of changing resolution with red and white circles, indicating the red circles might be lost or replaced in the larger grid.](image-url)
Nearest Neighbor

- a.k.a. zero order interpolation
- use 1 nearest neighbor
Bilinear

☐ a.k.a.
  first order interpolation

☐ use 4 nearest neighbors
Bicubic

- a.k.a. second order interpolation
- use 16 nearest neighbors
Example

nearest neighbor  bicubic  bilinear  ground truth
Example

nearest neighbor
bilinear
bicubic
MIP Mapping

- MIP Mapping is one popular technique for precomputing and performing this prefiltering.

- Computing this series of filtered images requires only a small fraction of additional storage over the original texture.
Storing MIP Maps
Example

point sampling

linear filtering

mipmapped point sampling

mipmapped linear filtering
Environment Mapping
Sphere Mapping
Box Maps
Spherical Mapping
Box Mapping

- Easy to use with simple orthographic projection
- Also used in environmental maps
Second Mapping

- Map from intermediate object to actual object
  - Normals from intermediate to actual
  - Normals from actual to intermediate
  - Vectors from center of intermediate
Environment Maps

ray traced  environment map
Bump Mapping

- Textures can be used for more than just color
  \[ I = k_a I_a + \sum_i f_{att} I_p \left[ k_d (\vec{N} \cdot \vec{L}_i) + k_s (\vec{R}_i \cdot \vec{V})^n \right] \]

- In bump mapping, a texture is used to perturb the normal:
  - The normal is perturbed in each parametric direction according to the partial derivatives of the texture.
Bump Mapping
Bump Mapping
Bump Mapping
Illumination Maps
Texture Mapping in Quake

Texture Only

Texture & Light Maps

Light Map
Shadow Maps
Basic Steps of Shadow Maps

- Render the scene from the light’s point of view,
- Use the light’s depth buffer as a texture (shadow map),
- Projectively texture the shadow map onto the scene,
- Use “texture color” (comparison result) in fragment shading.
Shadow Buffer

\[ I = k_e + k_a I_a + \sum_i S_i \, f_{\text{att}} \, I_{p_i} \, [k_d (\vec{N} \cdot \vec{L}_i) + k_s (\vec{R}_i \cdot \vec{V})^n] \]