CS180: Introduction to Computer Graphics
Spring 2019, Lingqi Yan, UC Santa Barbara

Lecture 8:
Shading 1 (Illumination, Shading and Graphics Pipeline)

Last Lectures

• Rasterization
  - Rasterizing triangles
  - Z-Buffering
  - Sampling and antialiasing
Today

- Shading
  - Illumination & Shading
  - Graphics Pipeline
What We’ve Covered So Far

- Position objects and the camera in the world
- Compute position of objects relative to the camera
- Project objects onto the screen
- Sample triangle coverage

Slide courtesy of Prof. Ren Ng, UC Berkeley
Rotating Cubes in Perspective
Rotating Cubes in Perspective
What Else Are We Missing?

Credit: Bertrand Benoit. “Sweet Feast,” 2009. [Blender /VRay]
Shading
Shading: Definition

- In Merriam-Webster Dictionary
  shad·ing, [ˈʃeɪdɪŋ], noun

  The darkening or coloring of an illustration or diagram with parallel lines or a block of color.

- In this course

  The process of applying a material to an object.
Simple Shading
(Blinn-Phong Reflectance Model)
Perceptual Observations

Specular highlights

Diffuse reflection

Ambient lighting

Photo credit: Jessica Andrews, flickr
Shading is Local

Compute light reflected toward camera at a specific shading point

Inputs:

- Viewer direction, $v$
- Surface normal, $n$
- Light direction, $l$ (for each of many lights)
- Surface parameters (color, shininess, ...)

Slide courtesy of Prof. Ren Ng, UC Berkeley
Shading is Local

No shadows will be generated! (shading ≠ shadow)
Diffuse Reflection

Light is scattered uniformly in all directions
  • Surface color is the same for all viewing directions

Lambert’s cosine law

Top face of cube receives a certain amount of light
Top face of 60° rotated cube intercepts half the light
In general, light per unit area is proportional to $\cos \theta = l \cdot n$
Light Falloff

Intensity here: $I$ / $r^2$

Slide courtesy of Prof. Ren Ng, UC Berkeley
Lambertian (Diffuse) Shading

Shading independent of view direction

\[
L_d = k_d \left( \frac{I}{r^2} \right) \max(0, \mathbf{n} \cdot \mathbf{l})
\]

- illumination from source
- diffuse coefficient
- diffusely reflected light
- \( \mathbf{n} \) is the normal vector
- \( \mathbf{l} \) is the direction of the light source
- \( \mathbf{v} \) is the viewing direction
- \( \theta \) is the angle between the normal and the light source
Lambertian (Diffuse) Shading

Produces diffuse appearance

\[ k_d \]
Specular Shading (Blinn-Phong)

Intensity depends on view direction

• Bright near mirror reflection direction
Specular Shading (Blinn-Phong)

Close to mirror direction ⇔ half vector near normal

- Measure “near” by dot product of unit vectors

\[
\mathbf{h} = \text{bisector}(\mathbf{v}, \mathbf{l})
\]

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

\[
L_s = k_s \left( \frac{I}{r^2} \right) \max(0, \cos \alpha)^p
\]

\[
= k_s \left( \frac{I}{r^2} \right) \max(0, \mathbf{n} \cdot \mathbf{h})^p
\]
Cosine Power Plots

Increasing $p$ narrows the reflection lobe

![Graphs showing the effect of increasing $p$ on the cosine power plots.](image-url)
Specular Shading (Blinn-Phong)

Blinn-Phong

\[ L_s = k_s \left( \frac{I}{r^2} \right) \max(0, n \cdot h)^p \]
Ambient Shading

Shading that does not depend on anything

- Add constant color to account for disregarded illumination and fill in black shadows

\[ L_a = k_a I_a \]

ambient coefficient

reflected ambient light
Blinn-Phong Reflection Model

\[ L = L_a + L_d + L_s \]
\[ = k_a I_a + k_d \left( \frac{I}{r^2} \right) \max(0, \mathbf{n} \cdot \mathbf{l}) + k_s \left( \frac{I}{r^2} \right) \max(0, \mathbf{n} \cdot \mathbf{h})^p \]
Shading Methods / Frequencies
Shading methods

What caused the shading difference?
Shade each triangle (flat shading)

**flat** shading

- Triangle face is flat — one normal vector
- Not good for smooth surfaces
Shade each vertex (Gouraud shading)

**Gouraud shading**

- Interpolate colors from vertices across triangle
- Each vertex has a normal vector
Shade each pixel (Phong shading)

**Phong shading**

- Interpolate normal vectors across each triangle
- Compute full shading model at each pixel
- Not the Blinn-Phong reflectance model
Shading Frequency: Face, Vertex or Pixel

Shading freq.: Face, Vertex, Pixel
Shading type: Flat, Gouraud, Phong

Defining Per-Vertex Normal Vectors

Best to get vertex normals from the underlying geometry

- e.g. consider a sphere

Otherwise have to infer vertex normals from triangle faces

- Simple scheme: average surrounding face normals

\[ N_v = \frac{\sum_i N_i}{\| \sum_i N_i \|} \]
Defining Per-Pixel Normal Vectors

Barycentric interpolation (next lecture) of vertex normals

Don’t forget to normalize the interpolated directions
Graphics Pipeline
Graphics Pipeline

1. **Input:** vertices in 3D space
2. **Vertices positioned in screen space**
3. **Triangles positioned in screen space**
4. **Fragments (one per covered sample)**
5. **Shaded fragments**
6. **Output:** image (pixels)
Graphics Pipeline

Model, View, Projection transforms

Application

Vertex Stream

Vertex Processing

Triangle Processing

Triangle Stream

Rasterization

Fragment Stream

Fragment Processing

Shaded Fragments

Framebuffer Operations

Display

Slide courtesy of Prof. Ren Ng, UC Berkeley
Graphics Pipeline

1. Application
   - Vertex Processing
     - Vertex Stream
   - Triangle Processing
     - Triangle Stream
   - Rasterization
   - Fragment Processing
     - Fragment Stream
   - Framebuffer Operations
     - Shaded Fragments
2. Display

Sampling triangle coverage

Slide courtesy of Prof. Ren Ng, UC Berkeley
Rasterization Pipeline

1. Application
   - Vertex Processing
     - Vertex Stream
   - Triangle Processing
     - Triangle Stream
   - Rasterization
     - Fragment Stream
   - Fragment Processing
     - Shaded Fragments
   - Framebuffer Operations
     - Display

Z-Buffer Visibility Tests

Slide courtesy of Prof. Ren Ng, UC Berkeley
Graphics Pipeline

Application

Verteck Processing

Triangle Processing

Rasterization

Fragment Processing

Framebuffer Operations

Display

Vertex Stream

Triangle Stream

Fragment Stream

Shaded Fragments

Shading

Ambient + Diffuse

+ Specular = Blinn-Phong Reflectance Model

Slide courtesy of Prof. Ren Ng, UC Berkeley
Graphics Pipeline

- **Application**
  - **Vertex Processing**
    - **Vertex Stream**
  - **Triangle Processing**
    - **Triangle Stream**
  - **Rasterization**
    - **Fragment Stream**
  - **Fragment Processing**
    - **Shaded Fragments**
  - **Framebuffer Operations**

- **Display**

Texture mapping (next lecture)

Slide courtesy of Prof. Ren Ng, UC Berkeley
Shader Programs

- Program vertex and fragment processing stages
- Describe operation on a single vertex (or fragment)

Example GLSL fragment shader program

```glsl
uniform sampler2D myTexture;
uniform vec3 lightDir;
varying vec2 uv;
varying vec3 norm;

void diffuseShader()
{
    vec3 kd;
    kd = texture2d(myTexture, uv);
    kd *= clamp(dot(-lightDir, norm), 0.0, 1.0);
    gl_FragColor = vec4(kd, 1.0);
}
```

- Shader function executes once per fragment.
- Outputs color of surface at the current fragment’s screen sample position.
- This shader performs a texture lookup to obtain the surface’s material color at this point, then performs a diffuse lighting calculation.
Shader Programs

• Program vertex and fragment processing stages
• Describe operation on a single vertex (or fragment)

Example GLSL fragment shader program

```glsl
uniform sampler2D myTexture; // program parameter
uniform vec3 lightDir; // program parameter
varying vec2 uv; // per fragment value (interp. by rasterizer)
varying vec3 norm; // per fragment value (interp. by rasterizer)

void diffuseShader()
{
    vec3 kd;
    kd = texture2d(myTexture, uv); // material color from texture
    kd *= clamp(dot(-lightDir, norm), 0.0, 1.0); // Lambertian shading model
    gl_FragColor = vec4(kd, 1.0); // output fragment color
}
```
Snail Shader Program

Inigo Quilez
Snail Shader Program

Goal: Highly Complex 3D Scenes in Realtime

- 100’s of thousands to millions of triangles in a scene
- Complex vertex and fragment shader computations
- High resolution (2-4 megapixel + supersampling)
- 30-60 frames per second (even higher for VR)
Graphics Pipeline Implementation: GPUs

Specialized processors for executing graphics pipeline computations

Discrete GPU Card
(NVIDIA GeForce Titan X)

Integrated GPU:
(Part of Intel CPU die)

Slide courtesy of Prof. Ren Ng, UC Berkeley
GPU: Heterogeneous, Multi-Core Processor

Modern GPUs offer ~2-4 Tera-FLOPs of performance for executing vertex and fragment shader programs. Tera-Op's of fixed-function compute capability over here.