Introduction to Computer Graphics
GAMES101, Lingqi Yan, UC Santa Barbara

Lecture 8:
Shading 2
(Shading, Pipeline and Texture Mapping)

Announcements

• Homework 2
  - 45 submissions so far
  - Upside down? No problem
  - Active discussions in the BBS, pretty good

• Next homework is for shading

• Today’s topics
  - Easy, but a lot
Last Lecture

- Shading 1
  - Blinn-Phong reflectance model
    - Diffuse
    - Specular
    - Ambient
  - At a **specific shading point**
Today

• Shading 2
  - Blinn-Phong reflectance model
    - Specular and ambient terms
  - Shading frequencies
  - Graphics pipeline
    - Texture mapping
  - Barycentric coordinates
Recap: Lambertian (Diffuse) Term

Shading independent of view direction

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

Energy arrived at the shading point

Energy received by the shading point

Diffusely reflected light

Diffuse coefficient (color)
Recap: Lambertian (Diffuse) Term

Produces diffuse appearance

$k_d$ →
Specular Term (Blinn-Phong)

Intensity depends on view direction
- Bright near mirror reflection direction
Specular Term (Blinn-Phong)

V close to mirror direction ⇔ **half vector** near normal

- Measure “near” by dot product of unit vectors

\[
h = \text{bisector}(v, l) = \frac{v + l}{\|v + l\|}
\]

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

Increasing $p$ narrows the reflection lobe

\[ \text{cos} \alpha, \quad \text{cos}^2 \alpha, \quad \text{cos}^8 \alpha, \quad \text{cos}^{64} \alpha \]
Specular Term (Blinn-Phong)

Blinn-Phong

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

Note: showing \( L_d + L_s \) together
Ambient Term

Shading that does not depend on anything

- Add constant color to account for disregarded illumination and fill in black shadows
- This is approximate / fake!

\[ L_a = k_a I_a \]
Blinn-Phong Reflection Model

\[ L = L_a + L_d + L_s \]

\[ = k_a I_a + k_d \left( \frac{I}{r^2} \right) \max(0, \mathbf{n} \cdot \mathbf{l}) + k_s \left( \frac{I}{r^2} \right) \max(0, \mathbf{n} \cdot \mathbf{h})^p \]
Questions?
Shading Frequencies
Shading Frequencies

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 (how?)
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

\[ \bar{N}_{v} = \frac{\sum_{i} N_{i}}{\parallel \sum_{i} N_{i} \parallel} \]
Defining Per-Pixel Normal Vectors

Barycentric interpolation (introducing soon) of vertex normals

Don’t forget to normalize the interpolated directions
Graphics (Real-time Rendering) Pipeline
Graphics Pipeline

Application → Vertex Processing → Triangle Processing → Rasterization → Fragment Processing → Framebuffer Operations → Display

- **Input:** vertices in 3D space
- **Vertices positioned in screen space**
- **Triangles positioned in screen space**
- **Fragments (one per covered sample)**
- **Shaded fragments**
- **Output:** image (pixels)
Graphics Pipeline

Application

Vertex Processing

Triangle Processing

Rasterization

Fragment Processing

Framebuffer Operations

Vertex Stream

Triangle Stream

Fragment Stream

Shaded Fragments

Model, View, Projection transforms

Display
Graphics Pipeline

Application

Vertex Processing
  Vertex Stream

Triangle Processing
  Triangle Stream

Rasterization
  Fragment Stream

Fragment Processing
  Shaded Fragments

Framebuffer Operations
  Display

Sampling triangle coverage
Rasterization Pipeline

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

Z-Buffer Visibility Tests
Graphics Pipeline

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

Shading

Ambient + Diffuse + Specular = Blinn-Phong Reflectance Model
Graphics Pipeline

Application

Vertex Processing

Vertex Stream

Triangle Processing

Triangle Stream

Rasterization

Fragment Stream

Fragment Processing

Shaded Fragments

Framebuffer Operations

Display

Texture mapping (introducing soon)
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

Procedurally modeled, 800 line shader.
http://shadertoy.com/view/ld3Gz2
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)
Modern GPUs offer ~2-4 Tera-FLOPs of performance for executing vertex and fragment shader programs.
Texture Mapping
Different Colors at Different Places?

\[ L_d = k_d \times \left( \frac{1}{r^2} \right) \times (n \cdot l) \]
Surfaces are 2D

Surface lives in 3D world space

Every 3D surface point also has a place where it goes in the 2D image (texture).
Texture Applied to Surface

Rendering without texture

Rendering with texture

Texture

Each triangle “copies” a piece of the texture image to the surface.
Visualization of Texture Coordinates

Each triangle vertex is assigned a texture coordinate (u,v)
Texture Applied to Surface

Rendered result

Triangle vertices in texture space
Textures applied to surfaces
Visualization of texture coordinates
Textures can be used multiple times!

example textures
used / **tiled**
Thank you!

(And thank Prof. Ravi Ramamoorthi and Prof. Ren Ng for many of the slides!)