Lecture 2:
Recap: Blinn-Phong Reflectance Model, Graphics Pipeline, Shading Languages, Rendering Equation
Announcement

• Some adjustments in Project 0
  - How about extending its deadline by a week (9 days)?
  - But I’m asking for Blinn-Phong in exchange!

• Some adjustments in #projects
  - How about we remove project 3?
  - Project 1: Variance Soft Shadow Mapping
  - Project 2: Precomputed Radiance Transfer, or Real-time Ray Tracing
  - Final project: Up to you!
Recap: Last Lecture

• Motivation: why real-time rendering

• Evolution of real-time rendering

• Technological and algorithmic milestones
  - Programmable graphics hardware
  - Precomputation-based methods
  - Interactive Ray Tracing
Today’s Lecture
Outline

- A simple material: Blinn-Phong reflectance model
- Basic GPU hardware pipeline
- Shading languages
- The rendering equation
What is material?

- Material is the cause of appearance
What is material?

- Material is **how the light interacts objects**
What is material?

- Material is **how the light interacts objects**
What is this material?

- Material is *how the light interacts objects*
What is material?

- Material is **how the light interacts objects**

Glass
What is this material?

- Material is how the light interacts objects
What is material?

- Material is **how the light interacts objects**
Let’s make a material!

- Specular highlights
- Diffuse reflection
- Ambient lighting
Configuration

- Viewing direction: $v$
- Light direction: $l$
- Surface normal: $n$
- Surface Parameters: (color, shininess)
Diffuse Term

- Light is **emitted** uniformly in all directions
  - But is not **received** uniformly

- 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 $\cos \theta = l \cdot n$
Diffuse Term

intensity here: $I / r^2$

intensity here: $I$
Diffuse Term

• independent of view direction

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

- Produces matte appearance
Specular Term

- Intensity depends on view direction
  - Bright near mirror reflection direction
Specular Term

- Close to mirror $\Leftrightarrow$ half vector near normal
  - Measure “near” by dot product of unit vectors

\[
\begin{align*}
\mathbf{h} &= \text{bisector}(\mathbf{v}, \mathbf{l}) \\
&= \frac{\mathbf{v} + \mathbf{l}}{\|\mathbf{v} + \mathbf{l}\|} \\
L_s &= k_s \left( \frac{I}{r^2} \right) \max(0, \cos \alpha)^p \\
&= k_s \left( \frac{I}{r^2} \right) \max(0, \mathbf{n} \cdot \mathbf{h})^p
\end{align*}
\]
Specular Term

- Increasing $p$ narrows the reflection lobe
Specular Term

- Increasing p narrows the reflection lobe

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

- Does not depend on anything
  - Add constant color to account for disregarded illumination and fill in black shadows

\[ L_a = k_a I_a \]

[Diagram of ambient term with equation and diagram of light source and surface.]
Blinn-Phong Reflectance Model

- Adding everything together

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

- A texture is an image that specifies spatially-varying colors (Kd).
Applying the material

Shading

- In Merriam-Webster Dictionary
  shading, ˈʃeɪdɪŋ, noun
  The darkening or coloring of an illustration or diagram with parallel lines or a block of color.

- By Lingqi
  (1) The process of applying a material to an object.
  (2) The process of local rendering.

CS291A, Winter 2019

Lingqi Yan, UC Santa Barbara
Shading == local rendering

- No shadows, no indirect illumination
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 Phong material model
Defining Per-Pixel Normals

- Barycentric interpolation of vertex normals
In fact...

- As long as there are enough faces / vertices, any shading method works fine
Textures can affect shading!

- Textures doesn’t have to only represent colors
  - What if it stores the height / normal?
  - Bump / normal mapping
  - Fake the detailed geometry

Relative height to the underlying surface
Textures can affect shading!

- Displacement mapping — a more advanced approach
  - Actually subdivides the mesh and modify the geometry
Questions?
Outline

- Blinn-Phong reflectance model
- **Basic GPU hardware pipeline**
- Shading languages
- The rendering equation
Create geometry, lights, materials, textures, cubemaps, ... as inputs

Transform and lighting calcs. Apply per-vertex operations

Textures, Cubemaps

Per-pixel (per-fragment) operations
These fixed function stages can be replaced by a general per-vertex calculation using vertex shaders in modern programmable hardware.
Pixel or Fragment Pipeline

Rasterization (scan conversion) → Texture Mapping → Z-buffering → Framebuffer

These fixed function stages can be replaced by a general per-fragment calculation using fragment shaders in modern programmable hardware.
GPU Programmable Shaders

Programmable in Modern GPUs (Vertex Shader)

Geometry Primitive Operations

Scan Conversion (Rasterize)

Texture Memory

Fragment Operations

Programmable in Modern GPUs (Fragment Shader)

Vertices

Images

Pixel Operations

Framebuffer

Traditional Approach: Fixed function pipeline (state machine)
New Development (2003-): Programmable pipeline
Painter’s Algorithm

- Inspired by how painters paint
  - Paint from back to front, overwrite the previous
Painter’s Algorithm

- Requires sorting in depth
  - $O(n \log n)$ for $n$ triangles

- Can have unresolvable depth order

[Foley et al.]
Z-Buffer

• Store current min. z-value for each sample position

• Additional buffer for depth values
  - framebuffer stores color values (RGB, RGBA, …)
  - depth buffer (z-buffer) stores depth (16 to 32 bits)
Z-Buffer Example

Rendering

Depth buffer

Image credit: Dominic Alves, flickr.
Z-Buffer Algorithm

- Initialize depth buffer to $\infty$

- During rasterization:

  for (each triangle T)
    for (each sample (x,y,z) in T)
      if (z < zbuffer[x,y]) // closest so far
        framebuffer[x,y] = rgb; // update color
        zbuffer[x,y] = z; // update z
Z-Buffer Algorithm
Z-Buffer Complexity

• Complexity
  - $O(n)$ for $n$ triangles (assuming triangles are small)

• Most important visibility algorithm
  - Implemented in hardware for all GPUs
  - Used by OpenGL
Simplified OpenGL Pipeline

- User specifies vertices (one by one, or using vertex buffer object (VBO))

- For each vertex in parallel
  - OpenGL calls user-specified vertex shader: Transform vertex (ModelView, Projection), other ops

- For each primitive, OpenGL rasterizes
  - Generates a fragment for each pixel the fragment covers

- For each fragment in parallel
  - OpenGL calls user-specified fragment shader:
    - Shading and lighting calculations
  - OpenGL handles z-buffer depth test unless overwritten

- Modern OpenGL is “lite” basically just a rasterizer
  - “Real” action in user-defined vertex, fragment shaders
Questions?
Paper Presentations

- Sign up for paper presentations in class
  - One week before each presentation class
  - Three high quality papers to choose from
  - Claim your favorite topic ASAP
  - Email me to confirm, I’ll update website

- This time?