Lecture 2:
Recap: The Basics of Computer Graphics
Last Lecture

• What is CS291A about?

• Motivation: why real-time rendering

• Evolution of real-time rendering

• Technological and algorithmic milestones
Today

• Technological and algorithmic milestones
  - Programmable graphics hardware
  - Precomputation-based methods
  - Interactive Ray Tracing

• Recap

• Project 0
Technological and Algorithmic Milestones

- Programmable graphics hardware (shaders) (20 years ago)

A New Dawn demo, NVIDIA
https://www.geforce.com/games-applications/pc-applications/a-new-dawn/videos
Technological and Algorithmic Milestones

- Programmable graphics hardware (shaders)
Technological and Algorithmic Milestones

• Precomputation-based methods (15 years ago)
  - Complex visual effects are (partially) pre-computed
  - Minimum rendering cost at run time
Technological and Algorithmic Milestones

• Precomputation-based methods

All-Frequency Rendering of Dynamic, Spatially-Varying Reflectance
Wang et al.
Technological and Algorithmic Milestones

- Precomputation-based methods: Relighting
  - Fix geometry
  - Fix viewpoint
  - Dynamically change lighting

[Ng, Ramamoorthi, Hanrahan 04]
Technological and Algorithmic Milestones

• Interactive Ray Tracing (8-10 years ago: CUDA + OptiX)
  - Hardware development allows ray tracing on GPUs at low sampling rates (~1 samples per pixel (SPP))
  - Followed by post processing to denoise

Car interactively rendered using NVIDIA OptiX

Pixar’s real-time previewer
Technological and Algorithmic Milestones

• What do you think [is / will be] the milestone of CG today and in the future?
  - [Winter 2019] Deep learning?
  - [Winter 2019] Real-Time Ray Tracing?
  - [Winter 2019] VR with realistic graphics?
  - [Spring 2020] Quantum Computing?
  - [Spring 2020] AR

• Let’s re-look at this slide several years later!
Questions?
Today

• Technological and algorithmic milestones

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

- Material is the reason of various appearance
What is material?

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

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

• Material is how the light interacts with objects
What is material?

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

Glass
What is this material?

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

- Material is **how the light interacts with 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

![Diagram showing light emission and reception](image)

- 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

\[ I / r^2 \]

intensity here: \( I / r^2 \)

intensity here: \( I \)
Diffuse Term

- independent of view direction

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

• Produces matte appearance
Texture Mapping

• A texture is an image that specifies spatially-varying colors (Kd).
Specular Term

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

- Close to mirror $\iff$ 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

$$L_s = k_s (I/r^2) \max(0, \mathbf{n} \cdot \mathbf{h})^p$$
Ambient Term

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

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

- Adding everything together

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

- No shadows, no indirect illumination
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
Questions?
Today

• Technological and algorithmic milestones

• Recap
  - A simple material: Blinn-Phong reflectance model
  - Basic GPU hardware pipeline
  - Shading languages
  - The rendering equation
Basic Hardware Pipeline

**Application**

Create geometry, lights, materials, textures, cubemaps, ... as inputs

**CPU**

Create geometry, lights, materials, textures, cubemaps, ... as inputs

**Geometry**

Transform and lighting calcs.

Apply per-vertex operations

**GPU**

Textures, Cubemaps

Per-pixel (per-fragment) operations
Geometry or Vertex Pipeline

Model, View Transform → Projection → Clipping → Screen

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

Programmable in Modern GPUs (Fragment Shader)
- Fragment Operations

Vertices → Scan Conversion (Rasterize) → Fragment Operations

Images → Pixel Operations → Texture Memory

Framebuffer

Traditional Approach: Fixed function pipeline (state machine)
New Development (2003-): Programmable pipeline
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
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
Simplified OpenGL Pipeline

- 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
  - Three high quality papers to choose from
  - Claim your favorite topic ASAP
    (or you have to coordinate with other students later)

• This time?