Introduction
• Course Introduction

• Logistics

• History Of Graphics
Staff

(these are people that will help you in the course)

Nancy Pollard
[ npollard ]

Lucas Hurley
[ lmhurley ]

Divya Kartik
[ dkartik ]

David Krajewski
[ dkrajews ]

Ria Manathkar
[ rmanathk ]

Daniel Zeng
[ dlzeng ]
• Course Introduction

• Logistics

• History Of Graphics
Important Links

• Course Web Site:    http://15462.courses.cs.cmu.edu/spring2024
• Course Piazza:      https://piazza.com/class/lqxw6show0c6aj
• Course Slack:       link will be posted on piazza
• Course Gradescope:  link will be posted on piazza
• Course OH Queue:    link will be posted on piazza

• If you are having trouble accessing any of the links, please speak to a TA
Grading

- 5% A0: Math/Code Review
- 15%: A1: Rasterization
- 15%: A2: MeshEdit
- 15%: A3: PathTracing
- 15%: A4: Animation
- 10% Writtens
- 20% Exams
- 5% Participation
Why does this course exist?
4 Components Of Graphics

A1: Rasterization
A2: MeshEdit
A3: PathTracing
A4: Animation
4 Components Of Graphics

Batman (1956) DC Comics

God of War: Ragnarok (2022) Santa Monica Studio

Toy Story 3 (2010) Pixar

Floor Planning (2020) IKEA
Graphics In Video Games
Graphics In Technology
that’s a lot of graphics...
and we’re here to learn how to draw them all
math is hard, but you don't have to do it alone!
math is hard.
math is art.
Why Math?

- Lot of graphics concepts use math:
  - Coordinate systems
  - Transforms
  - Ray-casting
  - Color conversions
  - Intersection tests
  - Geometric queries
  - Physical simulations
    - And much more!

- Graphics is about converting data into simulations & experiences
  - Math helps us get there

- It is okay if you are not good at math!
  - But by the end of this course you will be : )
The Math Behind Graphics

\[
\begin{bmatrix}
1 & 2 & \cdots & n \\
1 & a_{11} & a_{12} & \cdots & a_{1n} \\
2 & a_{21} & a_{22} & \cdots & a_{2n} \\
3 & a_{31} & a_{32} & \cdots & a_{3n} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
m & a_{m1} & a_{m2} & \cdots & a_{mn}
\end{bmatrix}
\]

[ Linear ... Algebra ]

< Vector, Calculus >
Assignments

• **65% Assignments**
  • [05%] A0: Math Review
  • [15%] A1: Rasterization
  • [15%] A2: MeshEdit
  • [15%] A3: PathTracing
  • [15%] A4: Animation

• Solutions must be your own (you may not collaborate)

• A1 – A4 will have checkpoints! (Ex: A1.0, A1.5) Please submit on time

• Total of 5 late days for all assignments. **Cannot use late days on A4.5!**
  • After late days, 10% deduction in grade per day
Assignment 0.0: Math Review

- [2.5%] A0.0:
  - Linear Algebra
    - Linear Maps
    - Span
    - Orthonormal Bases
    - Matrices
  - Vector Calculus
    - Functions as Vectors
    - Inner/Cross Product
    - Determinant
    - Gradient

- Everyone has a unique assignment
  - Numbers (and solutions) are different for each student

- Submissions autograded
  - Unlimited submissions
  - You do not need to answer all problems
    - Extra credit for anything extra answered
Assignment 0.5: Code Review

- [2.5%] A0.5:
  - Setting Up Scotty3D
    - Cloning Repo
    - Setting Up Environment
    - Building Code
  - C++ Tests
    - Running Test Cases
    - Learning C++ Syntax

- Goal is to get you familiar with coding practices and syntax needed to complete coding assignment

- What is Scotty3D?
Assignments 1-4: Scotty3D

• We will give you a fully-working 3D graphics application with a working GUI that can rasterize, edit geometry, render scenes, and create animations
  • **The catch:** we removed all the core graphics operations from the application

• **Goal:** take what you’ve learned during lectures to build back the application
  • **Note:** there is not one correct solution! There are many ways to solve these graphics problems. We call them “algorithms” :)

• You will use the same codebase for all 4 assignments
  • Assignments are designed to be independent: bugs in A2 should not impact your A4 submission
Assignments 1-4: Scotty3D

[ A1: Rasterization ]

[ A2: MeshEdit ]

[ A3: PathTracer ]

[ A4: Animation ]
Assignment 1: Rasterization

- **A1.0: Rasterization Checkpoint**
  - Transformations
  - Lines
  - Triangles
  - Depth + Blending

- **A1.5: Rasterization Final**
  - Interpolation
  - Mip-Maps
  - Supersampling

- **Goal:** write a rasterizer that converts geometry into rasterized images
  - If you do not know the difference between a raster and render, you will learn : )
Assignment 2: MeshEdit

• **A2.0: MeshEdit Checkpoint**
  • Local Geometry Ops
    • Flip Edge
    • Split Edge
    • Collapse Edge
    • Extrude Face

• **A2.5: MeshEdit Final**
  • Global Geometry Ops
    • Triangulation
    • Linear Subdivision
    • Catmull-Clark Subdivision

• **Goal:** be able to create and manipulate geometry to model new 3D characters and scenes
Assignment 3: PathTracer

- **A3.0: PathTracer Checkpoint**
  - Camera Rays
  - Intersection Tests
  - BVH

- **A3.5: PathTracer Final**
  - Path Tracing
  - Materials
  - Direct Lighting
  - Environment Lighting

- **Goal**: create a render engine that can take any scene and create a photorealistic rendering out of it
  - We will learn ‘non-photorealistic’ styles in this class too
Assignment 4: Animation

- A4.0: Animation Checkpoint
  - Spline Interpolation
  - Skeleton Kinematics

- A4.5: Animation Final
  - Linear Blend Skinning
  - Particle Simulation

- **Goal**: make a platform for users to create animations out of geometry and scene files
Get creative!

• At the end of each assignment, you will use your working Scotty3D implementation to create a:
  • **A1:** Rasterized Artwork
  • **A2:** Character/Object model
  • **A3:** Rendered Environment
  • **A4:** Animation

• The best work is showcased at the end of the semester
A1 Past Creations
A3 Past Creations
Is this entire class programming?

Hint: no
• **10% Writtens**
  • Each class has an associated written assignment worth 100pts
    • Posted on the course website
    • Due the week after
  • Can work in groups of up to 3
  • No late days, but you may skip up to 2 writtens
  • Submit to Gradescope
Exams

• **20% Exams**
  • [10%] Midterm
  • [10%] Final

• Exam content will come from lectures, not just assignments.
  • Please attend class : )

• Final is cumulative.

• Standard 3” x 3” handwritten sticky note is allowed (front and back)

• We will provide practice exams closer to the exam date
Participation

- 5% Participations
  - Asking/Answering questions on piazza
  - Asking/Answering question on course slides
  - Attending lecture
What We Really Want From You

• **We want you to** be good programmers + have programming maturity
  • At the level of 15213/513 is the bare minimum.

• **We want you to** not be afraid of large codebases
  • The essence of Computer Graphics is large codebases and how to work with them.

• **We want you to** be able to read docs and language specs
  • There are large ReadMe docs for every assignment. Make sure you understand them before coding.

• **We do NOT want you to** have the relevant skills from day one.
  • We instead ask that you take the time to develop these skills while in this course, as they are common in industry and research.

• **We want you to** have fun
  • This is a creative class, make sure to learn, and you’ll be proud of what you learn to make.
• Course Introduction

• Logistics

• History Of Graphics
Before that,
What is Computer Graphics

**computer graphics** /kəmˈpyʊdər ˈɡrafɪks/ n.
The use of computers to synthesize visual information.
What is Computer Graphics

Drawing an image requires doing millions of the same operations across millions of triangles, lights, pixels, etc.
The CPU

- **Generic hardware**
  - Can do many things
    - Schedule/synchronize threads
    - Run dynamic loops
    - Compile code
    - Execute web scripts
    - Order a package off Amazon

- **A few cores**
  - Tens of cores, each with several threads
  - Can do parallel processing, but not much
  - Heterogeneous cores, not every core has the same performance
    - High performance cores
    - Energy-efficient cores

- **Small data**
  - Few proprietary registers
  - Small (if any) caches
  - Needs to spill into larger shared caches/DRAM
The CPU

- **Generic hardware**
  - Can do many things
    - Schedule/synchronize threads
  - Run dynamic loops
  - Compile code
  - Execute web scripts
  - Order a package off Amazon

- **A few cores**
  - Tens of cores, each with several threads
  - Can do parallel processing, but not much
  - Heterogeneous cores, not every core has the same performance
    - High performance cores
    - Energy-efficient cores

- **Small data**
  - Few proprietary registers
  - Small (if any) caches
  - Needs to spill into larger shared caches/DRAM

We don't need all this functionality!
We just want to draw some triangles!
The GPU

- **Specialized hardware**
  - Really good at doing a few operations
  - Catalogue of operations kept small
    - Easy to fetch smaller list of ops

- **Thousands of cores**
  - Can run the same operation on hundreds of thousands of data points at once
    - Good when the same code runs on data
    - Bad when divergence occurs

- **Large data**
  - Many registers for each core
  - Large GPU memory
  - Modern systems have shared memory with CPU
    - Easy for scheduling/data transfer

- “why buy a fireplace when you can buy a gpu” – nvidia ceo, probably
The GPGPU

- ‘General Purpose’ Graphics Processing Unit
  - Also known as the ‘modern GPU’
  - Sacrifices specialized hardware components for more general operations

- GPUs originally used for rendering
  - Data scientists ‘hacked’ GPUs by using the vertex shader to perform compute on large data systems
    - Led to the creation of compute shaders
  - GPUs now contain many more programmable stages and can be used in data science and machine learning

- Paradigm shift: sacrifice fixed function for more programmability
The GPU
The Graphics Pipeline

- Sometimes called the:
  - 3D Graphics Pipeline
  - Rasterization Pipeline
  - GPU Pipeline

- GPU was designed specifically to run this pipeline fast

- Entire pipeline was fixed-function.
  - You provide the data, a vertex shader, and a fragment shader, and the GPU does the rest.
  - Fixed-function == fast!
    - By limiting what an architecture can do, that makes the architecture really good at what it can do.
      - In graphics, we need to run the same operations over millions of datapoints.

Graphics Pipeline Tutorial (2019) Vulkan
Change Of Space

- Half the pipeline is in 3D, half is in 2D
  - Remember: we start with a 3D scene descriptor and end with a 2D image

- Moving from 3D to 2D scene provides many benefits:
  - Higher precision operations
  - Faster computations
  - Easier parallelism
  - Less data to manage
  - Less operations overall
Side Note: What Is A Shader?

• Shaders are any string of code run on the GPU
  • Not specific to graphics, any GPU code is shader code
    • Ex: Compute shaders

• Most shader code looks like it was written in C
  • Perfect for C++ graphics developers

• The term was originally created to refer to the user-defined portion of the Graphics Pipeline

• Every system’s GPU is different, therefore the CPU needs to compile (translate) the code into the GPU’s spec
  • For large graphics systems (think video games) with a common architecture (PS5, Xbox, etc.), shaders will be compiled before being shipped
    • Known as pre-compiled shaders
  • PCs on the other hand need to compile shaders when game first start since GPUs vary per PC
3D Graphics Systems Stack

scene.glb → vertices → primitives → fragments → image.png

Converting data into ... well, more data
But this data is pretty!
There is no GPU without U!

Basic Steps of Ray Tracing

1. Shoot Rays
We send each ray from the camera through a pixel on the virtual screen into the scene.

2. Find Hits
We find the closest scene hit by the ray using an intersection test.

3. Compute Color
We use information such as known light in the scene and the object’s material to calculate the pixel’s final color.

4. Compute Transparency
We check if the ray is intersected by any other object.

5. Compute Reflection
We repeat the Ray Tracing procedure for each reflection.

6. Compute Reflection
We find the nearest object hit by the ray.

7. Compute Transparency
We check if the ray is intersected by any other object.

8. Compute Transparency
We check if the ray is intersected by any other object.

Credit: Mia Tang