Introduction

- Course Introduction
- Logistics
- History Of Graphics

Staff

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

Nancy Pollard
[npollard]

David Krajewski
[dkrajews]

Lucas Hurley
[lmhurley]

Carson Piehl
[cpiehl]

Anoushka Naidu
[anaidu]

- ~~Course Introduction~~
- Logistics
- History Of Graphics

Important Links

Course Web Site:	http://15362.courses.cs.cmu.edu/spring2025
Course Piazza:	https://piazza.com/class/m5ie6hhy9fy7k
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

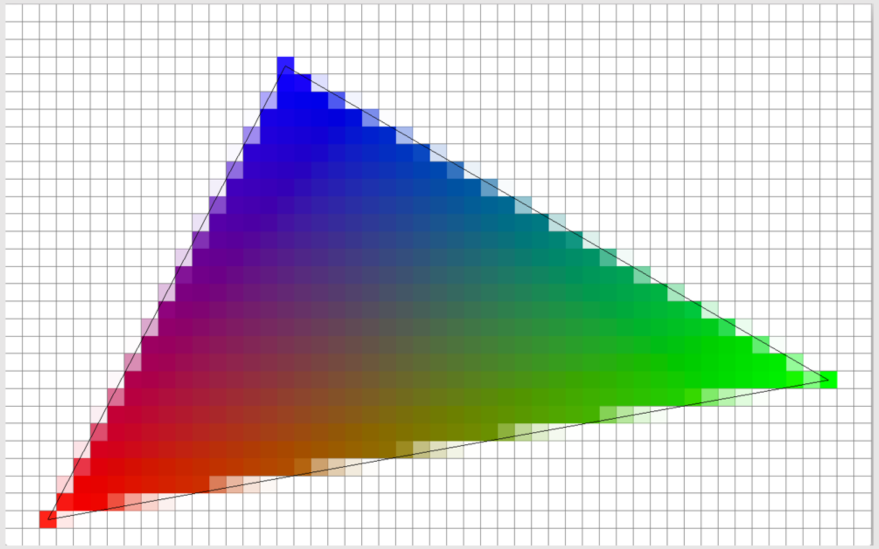
- If you are having trouble accessing any of the links, please speak to a TA

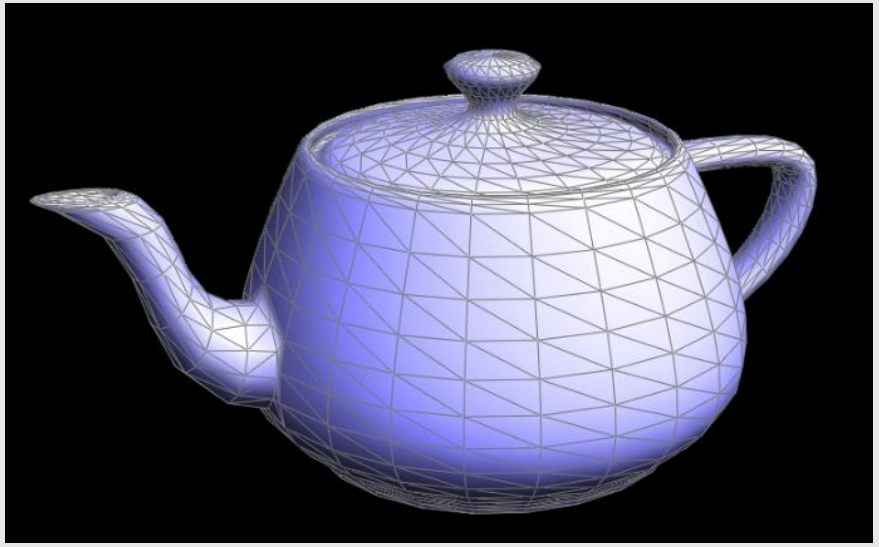
Grading

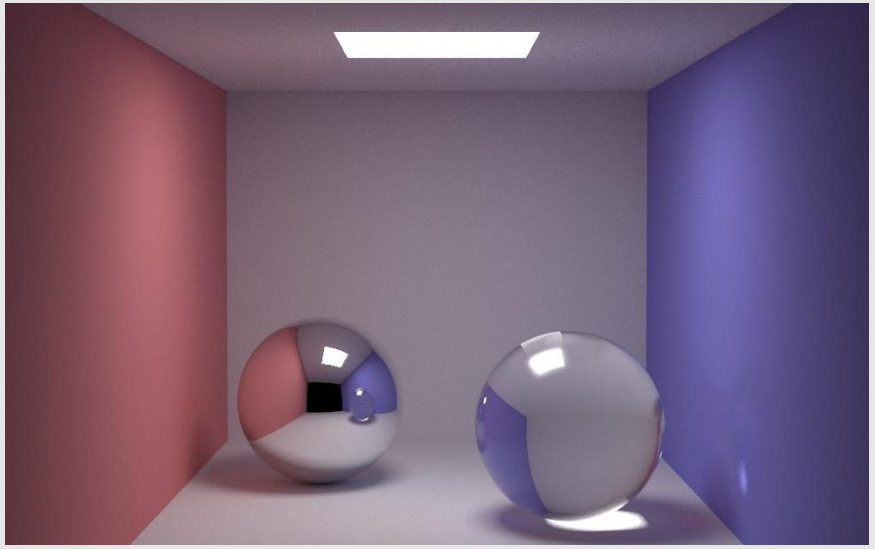
- **5% A0: Math/Code Review**
- **15%: A1: Rasterization**
- **15%: A2: MeshEdit**
- **15%: A3: PathTracing**
- **15%: A4: Animation**
- **10% Writtens**
- **20% Exams**
- **5% Participation**

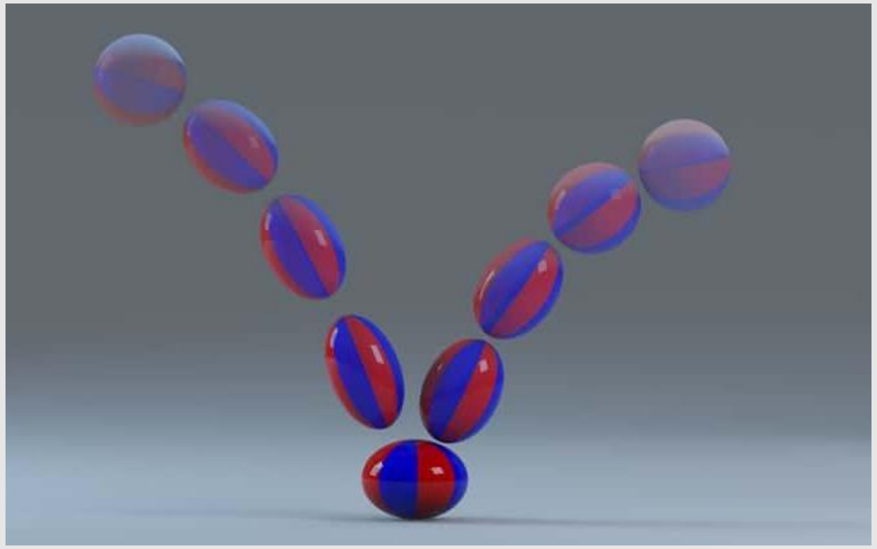
Why does this course exist?

4 Components Of Graphics


A1: Rasterization


A2: MeshEdit


A3: PathTracing


A4: Animation

4 Components Of Graphics

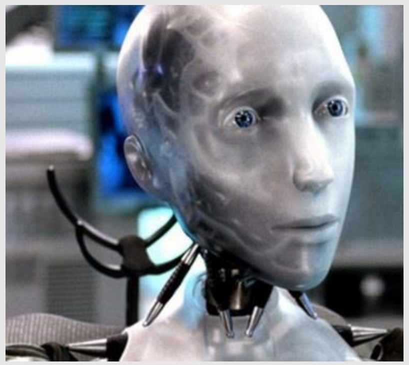
Batman (1956) DC Comics

"Lotso Poses" by Daniel Arriaga
Toy Story 3, 2010
Pencil on paper
Toy Story 3 (2010) Pixar

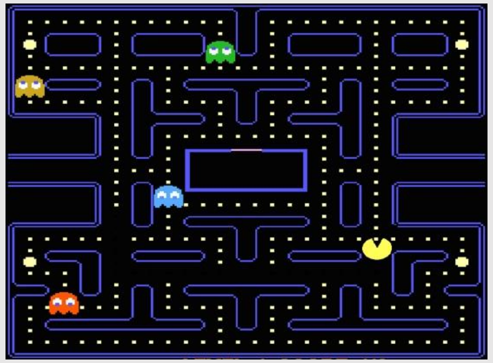
Floor Planning (2020) IKEA

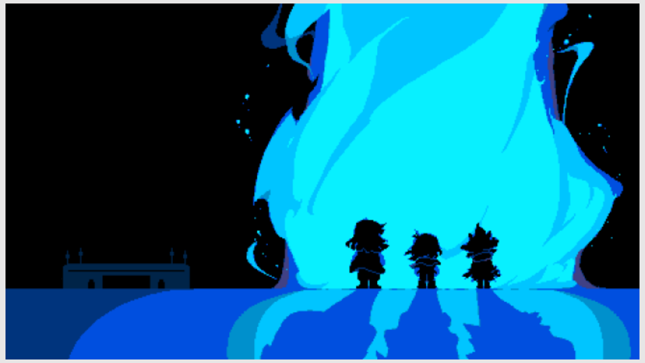
God of War: Ragnarok (2022) Santa Monica Studio

Graphics In Movies

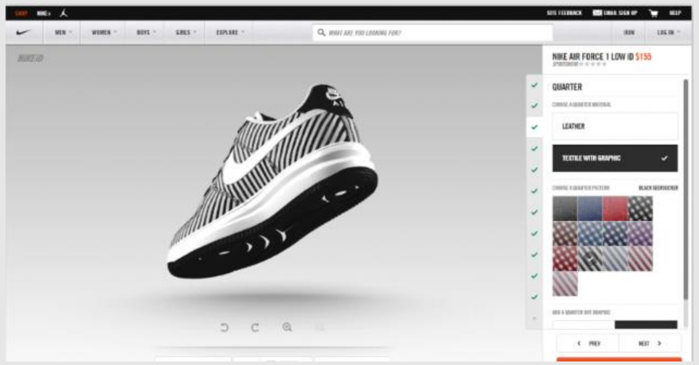


Graphics In Video Games



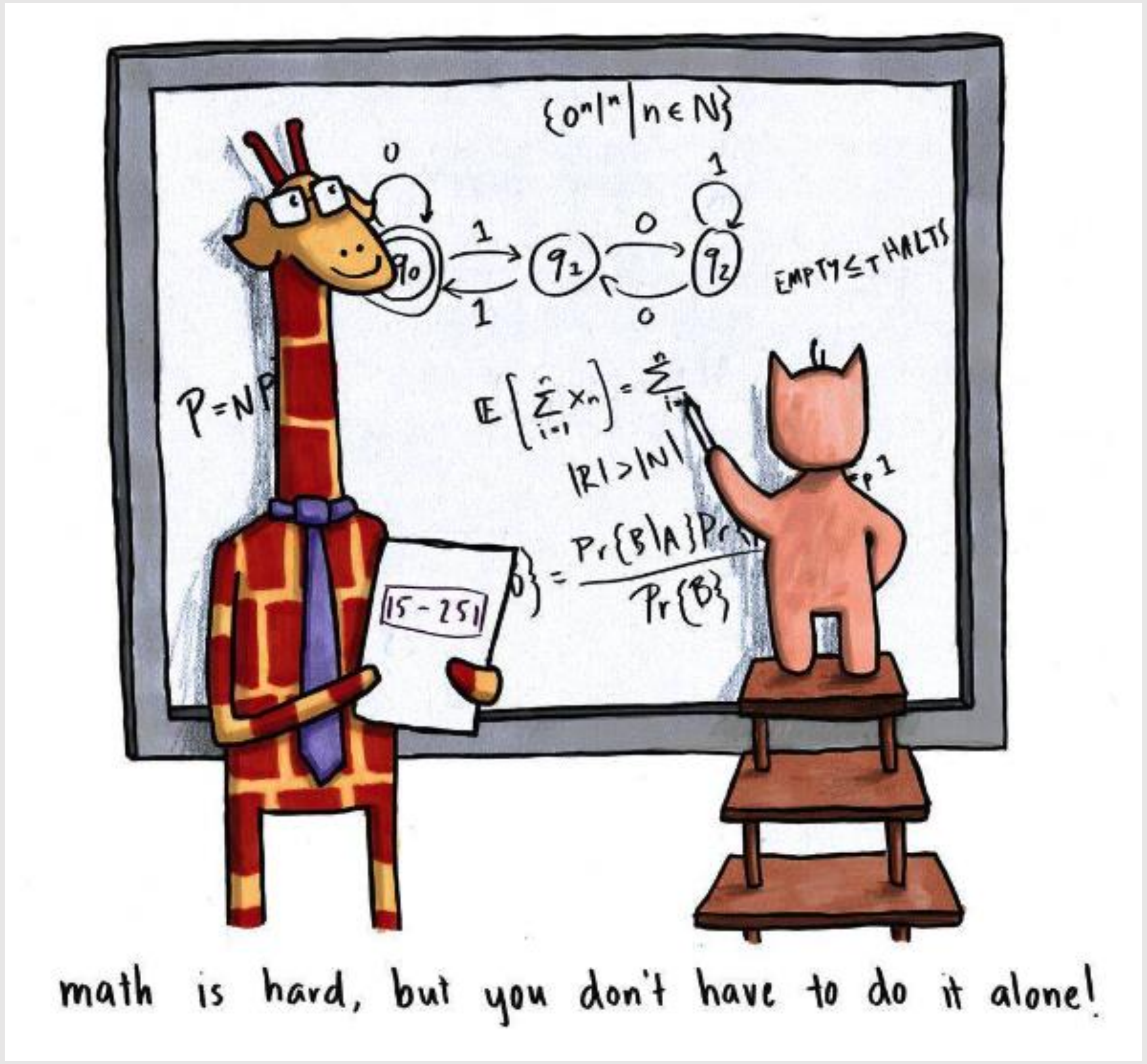


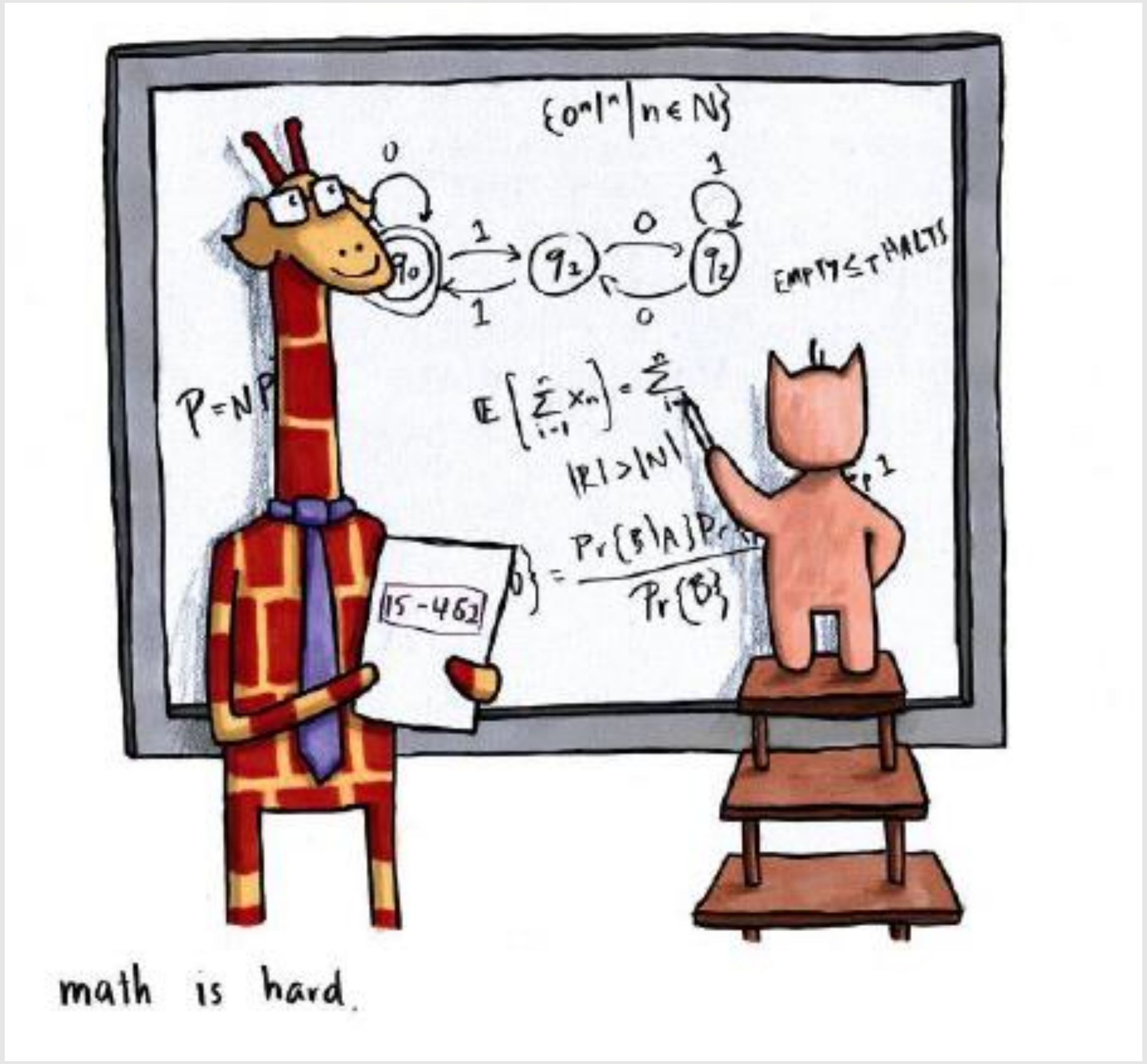
Graphics In Technology

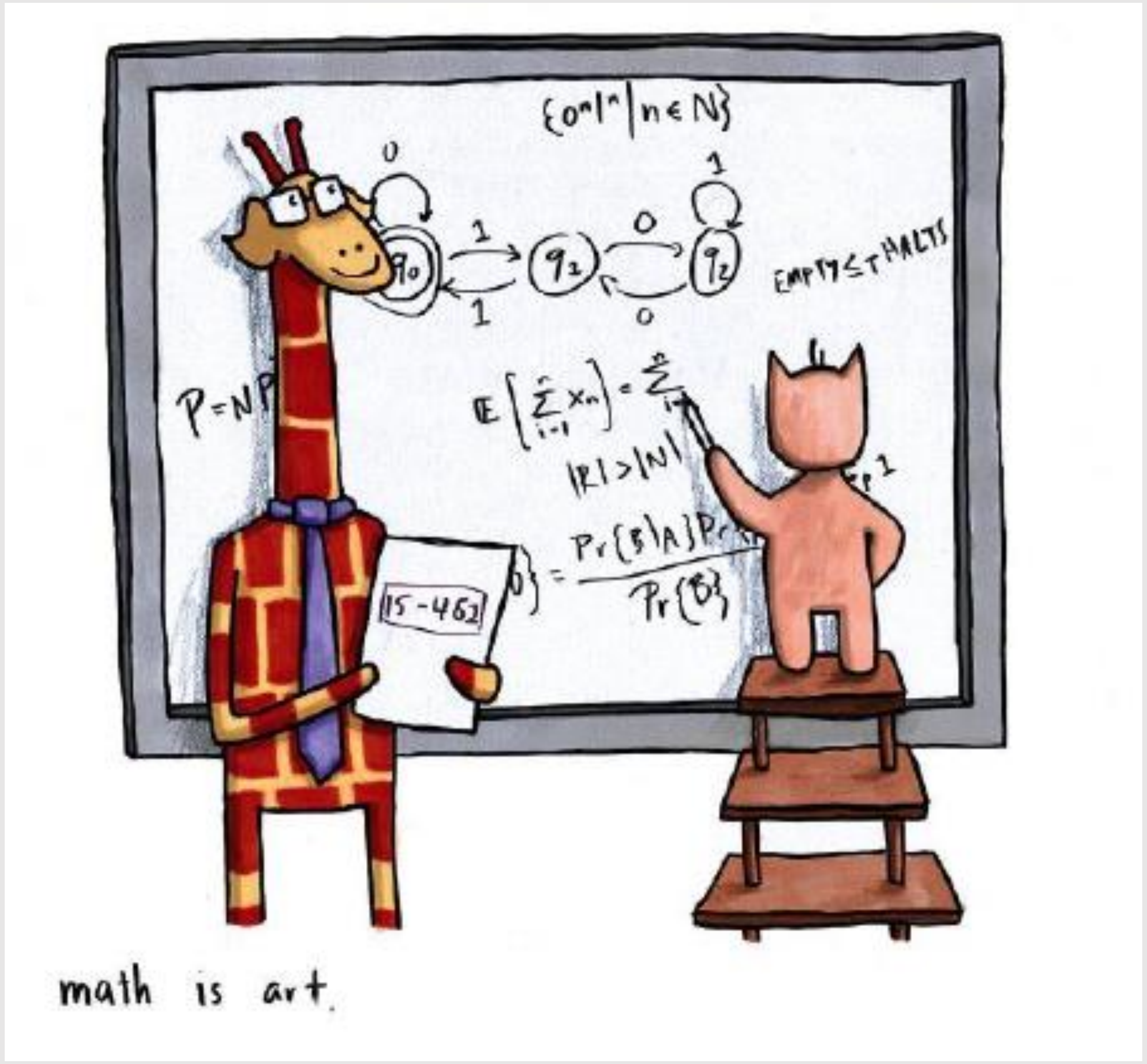


that’s a lot of graphics…

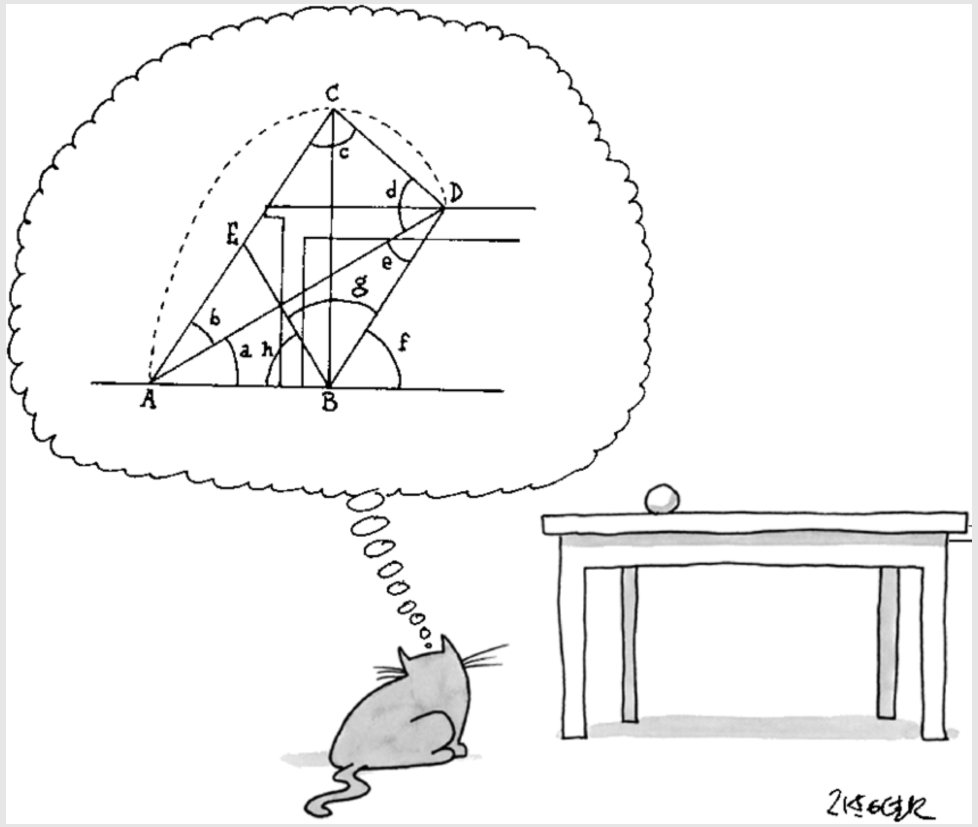
and we’re here to learn how to draw them all


$\{0^n1^n \mid n \in \mathbb{N}\}$
q_0
q_1
q_2
EMPTY $\le_T$ HALTS
P=NP
$\mathbb{E}\left[\sum_{i=1}^{n} X_n\right] = \sum_{i=1}^{n}$
$|\mathbb{R}| > |\mathbb{N}|$
$= \frac{\Pr\{B|A\}\Pr\{A\}}{\Pr\{B\}}$
15-251
math is hard, but you don't have to do it alone!


$\{0^n1^n \mid n \in N\}$
P=NP
EMPTY ≤T HALTS
15-462
|R|>|N|
math is hard.


{0^n1^n | n ∈ N}
P=NP
EMPTY ≤_T HALTS
|R| > |N|
15-462
math is art.

Why Math?



The New Yorker Collection (2001) Jack Ziegler

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

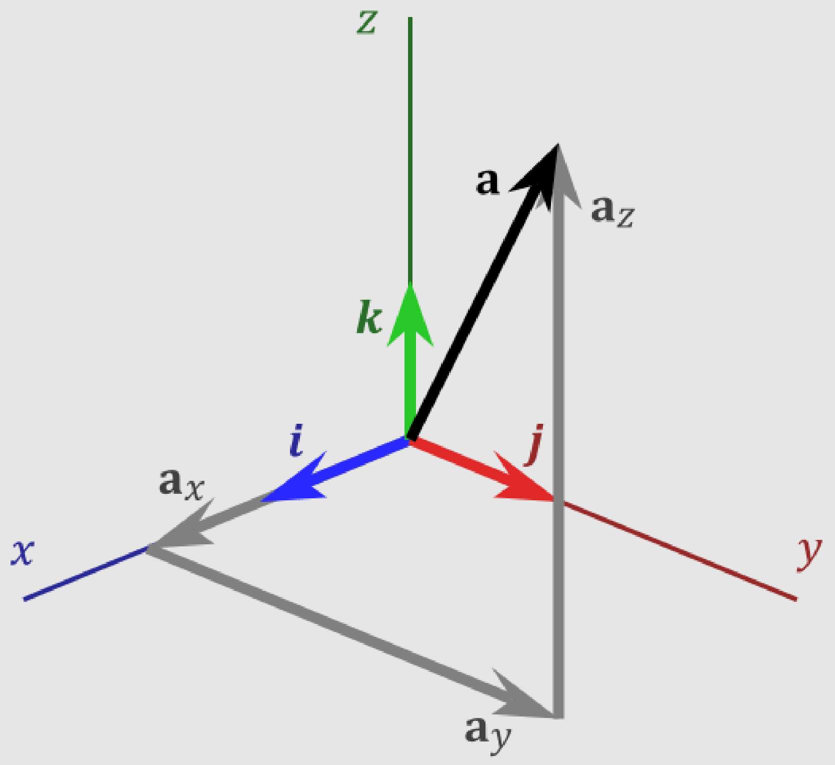
- Graphics is about converting data into simulations & experiences
  - Math helps us get there

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

The Math Behind Graphics

$$\begin{array}{cc} & \begin{array}{cccc} 1 & 2 & \cdots & n \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ \vdots \\ m \end{array} & \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ a_{31} & a_{32} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \end{array}$$

[Linear … Algebra]



< Vector, Calculus >

Assignments

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

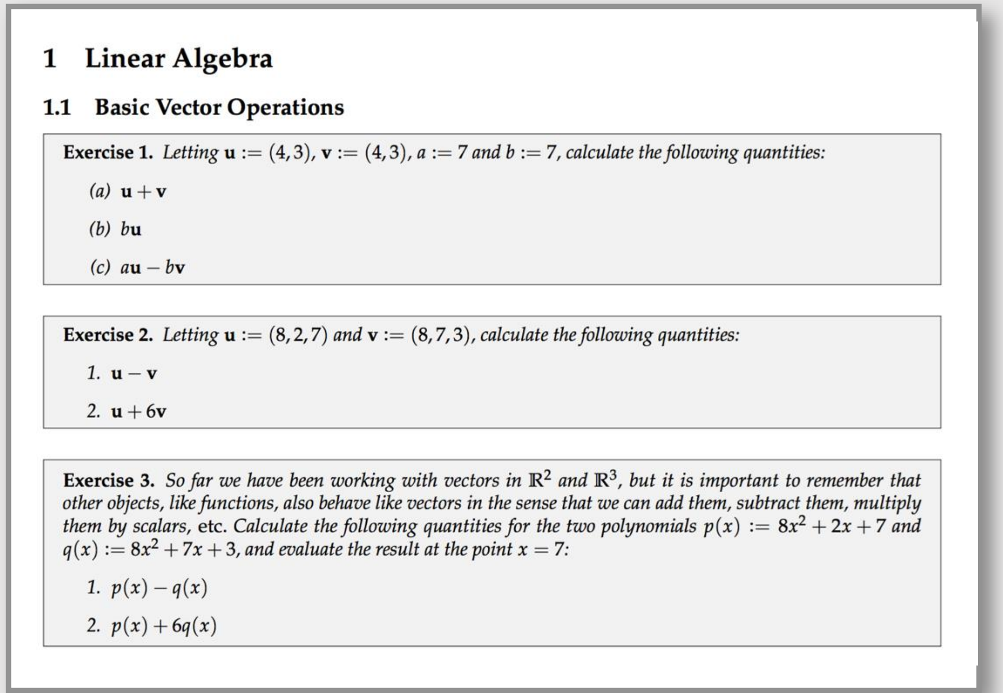
  - Solutions must be your own (you may not collaborate)

  - 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



1 Linear Algebra

1.1 Basic Vector Operations

Exercise 1. *Letting* $\mathbf{u} := (4,3)$, $\mathbf{v} := (4,3)$, $a := 7$ *and* $b := 7$, *calculate the following quantities:*

(a) $\mathbf{u}+\mathbf{v}$

(b) $b\mathbf{u}$

(c) $a\mathbf{u} - b\mathbf{v}$

Exercise 2. *Letting* $\mathbf{u} := (8,2,7)$ *and* $\mathbf{v} := (8,7,3)$, *calculate the following quantities:*

1. $\mathbf{u}-\mathbf{v}$
2. $\mathbf{u}+6\mathbf{v}$

Exercise 3. *So far we have been working with vectors in* $\mathbb{R}^2$ *and* $\mathbb{R}^3$, *but it is important to remember that other objects, like functions, also behave like vectors in the sense that we can add them, subtract them, multiply them by scalars, etc. Calculate the following quantities for the two polynomials* $p(x) := 8x^2+2x+7$ *and* $q(x) := 8x^2+7x+3$, *and evaluate the result at the point* $x=7$:

1. $p(x) - q(x)$
2. $p(x) + 6q(x)$

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

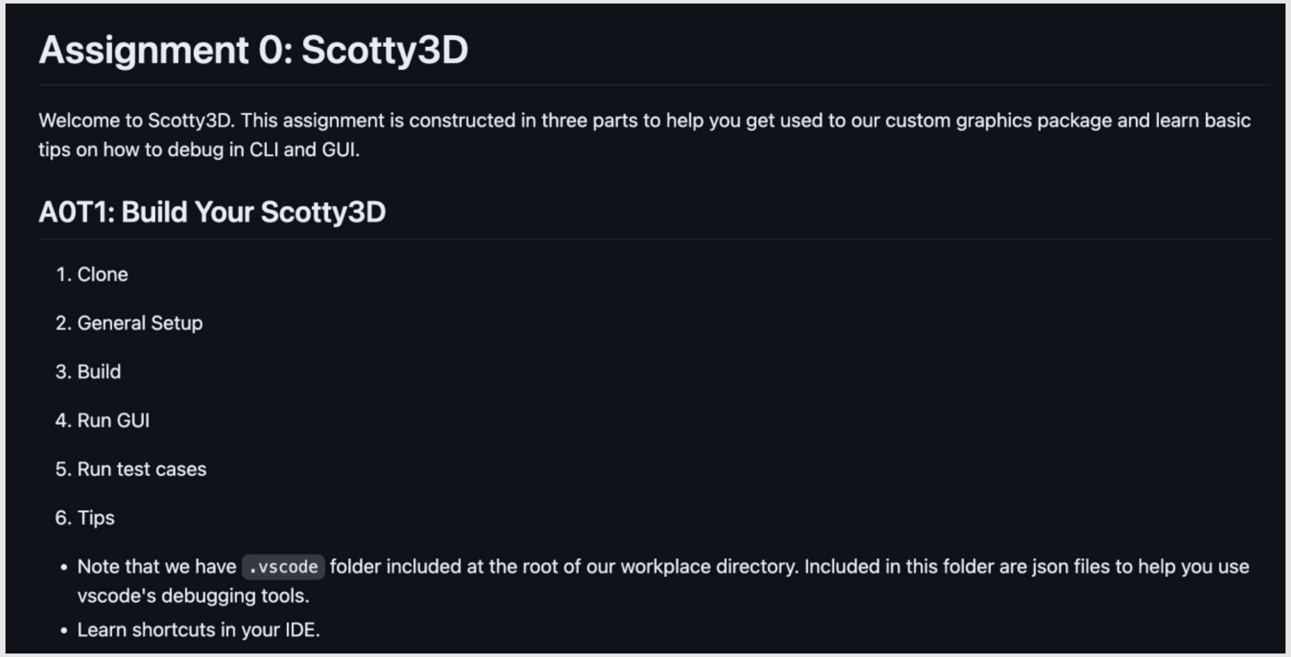
- Submissions autograded
  - Unlimited submissions
  - You do not need to answer all problems
    - Extra credit for anything extra answered

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?



Assignment 0: Scotty3D

Welcome to Scotty3D. This assignment is constructed in three parts to help you get used to our custom graphics package and learn basic tips on how to debug in CLI and GUI.

A0T1: Build Your Scotty3D

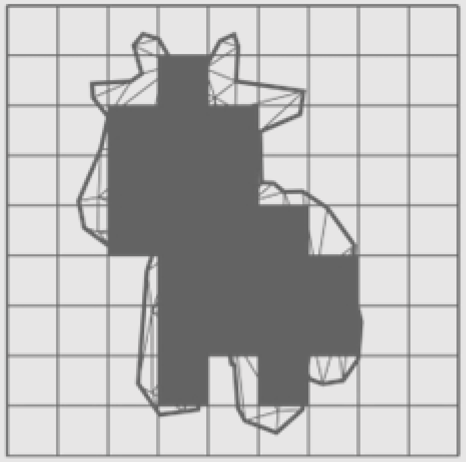
1. Clone
2. General Setup
3. Build
4. Run GUI
5. Run test cases
6. Tips

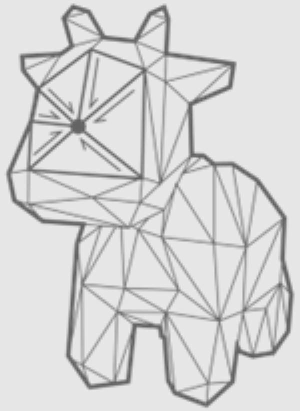
- Note that we have `.vscode` folder included at the root of our workplace directory. Included in this folder are json files to help you use vscode's debugging tools.
- Learn shortcuts in your IDE.

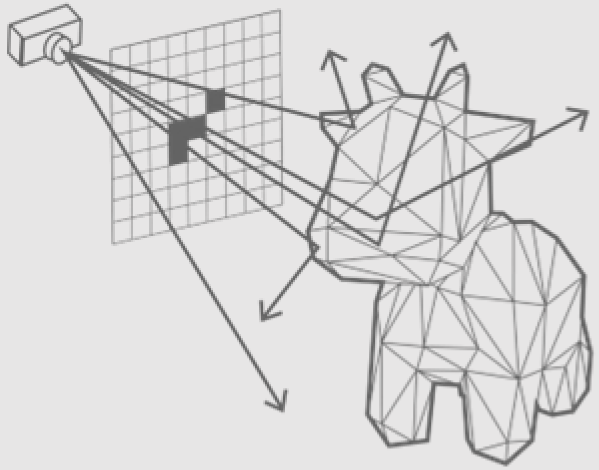
Assignments 1-4: Scotty3D

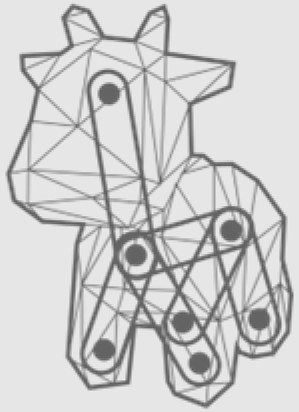
- We will give you a fully-working 3D graphics application with a working GUI that can rasterize, edit geometry, render scenes, and create animations
  - **The catch:** we removed all the core graphics operations from the application
- **Goal:** take what you've learned during lectures to build back 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]

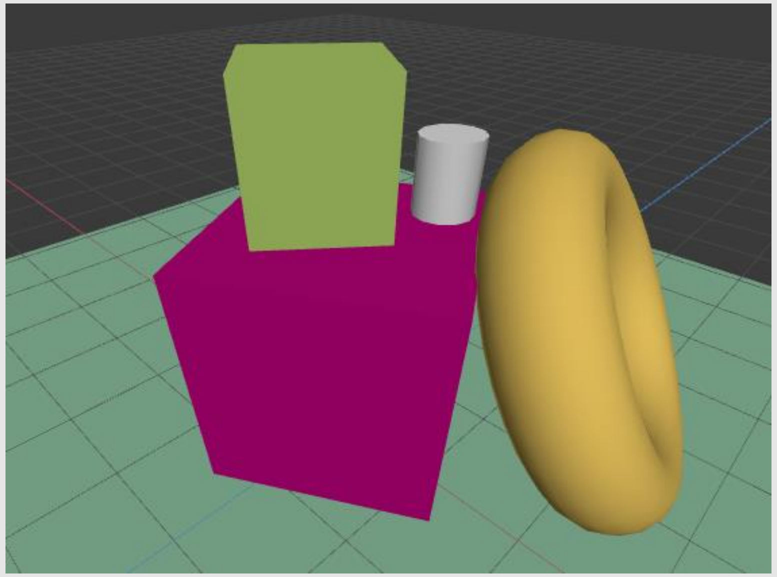

[A4: Animation]

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

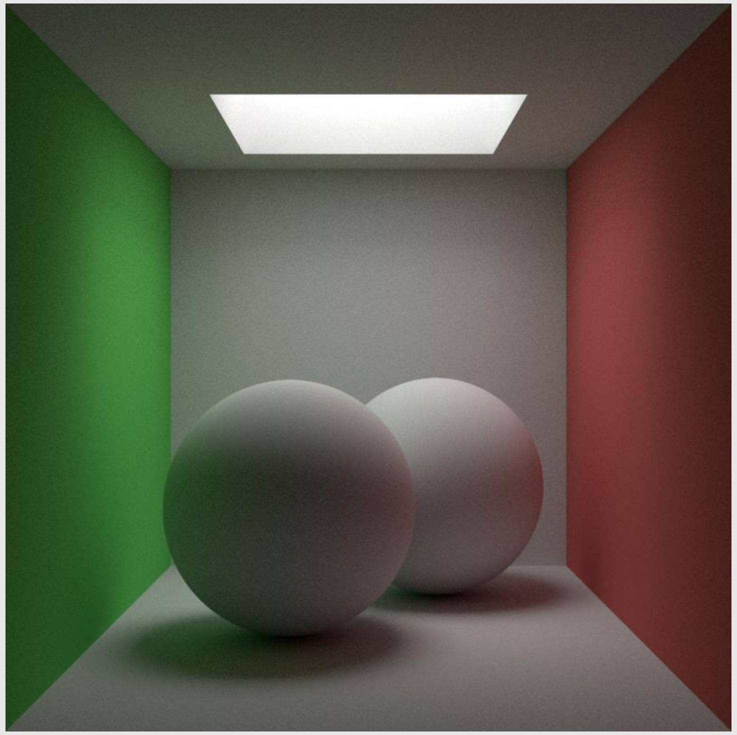
- **Goal:** be able to create and manipulate geometry to model new 3D characters and scenes

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

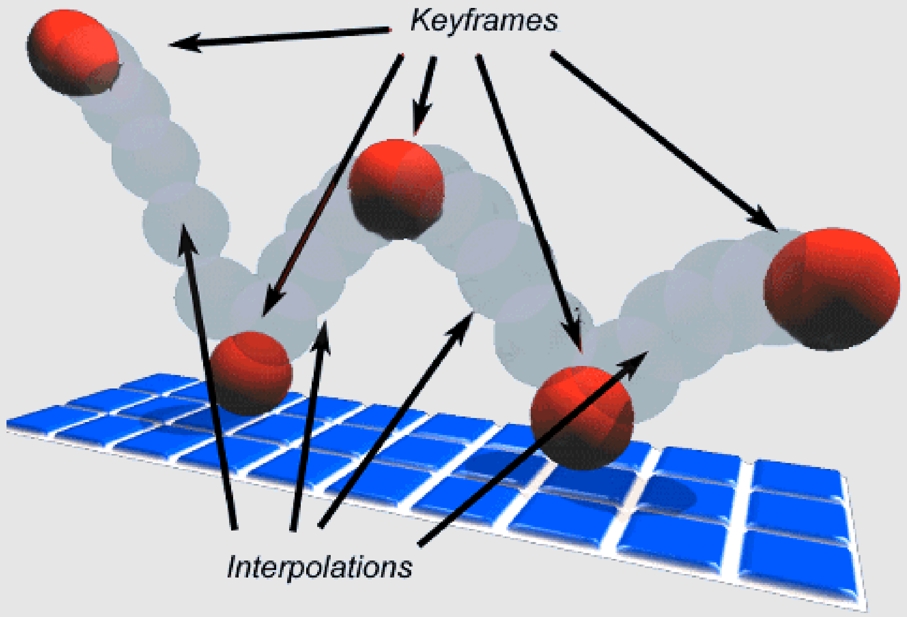


Assignment 4: Animation

- **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

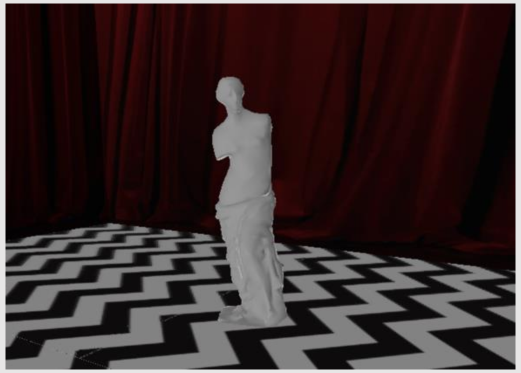
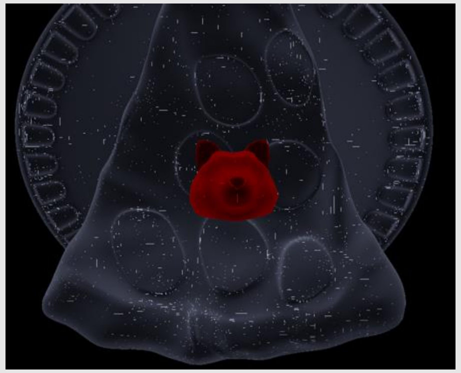
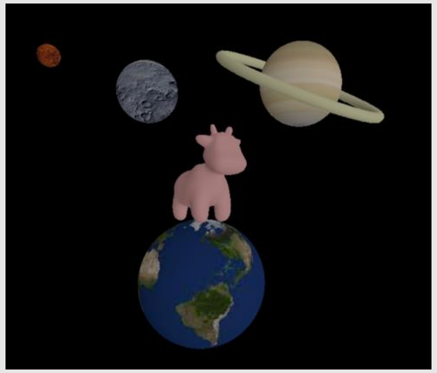
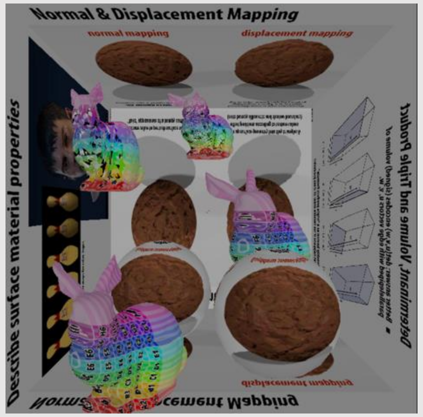
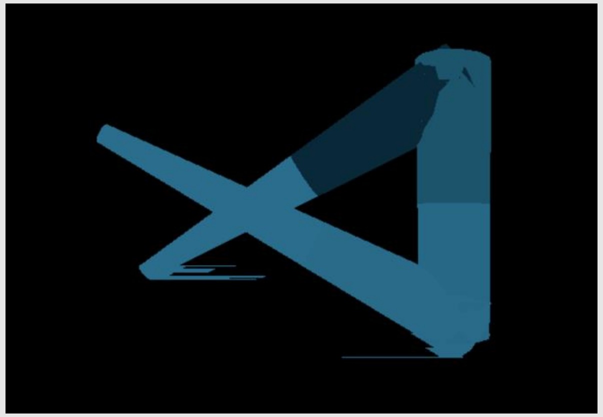
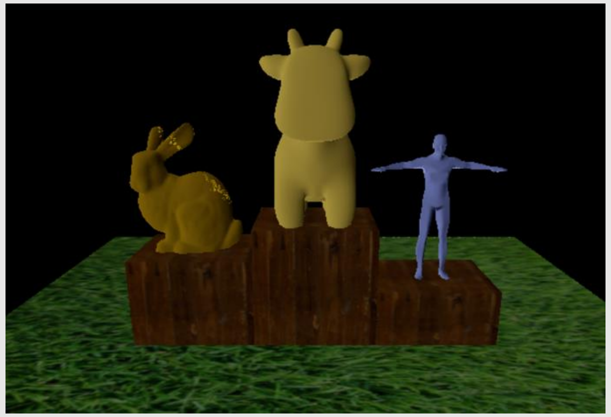
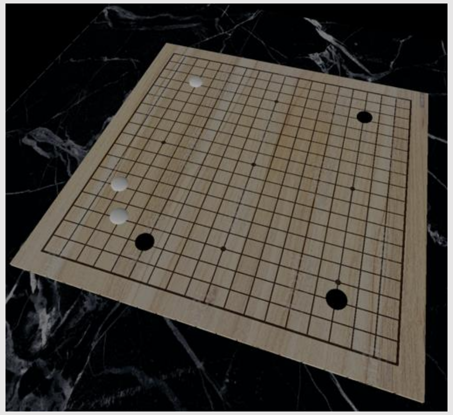


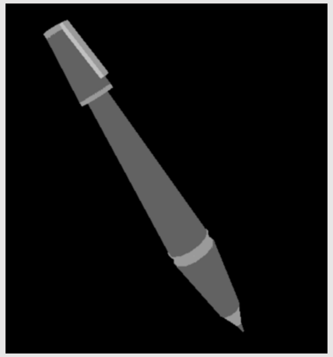
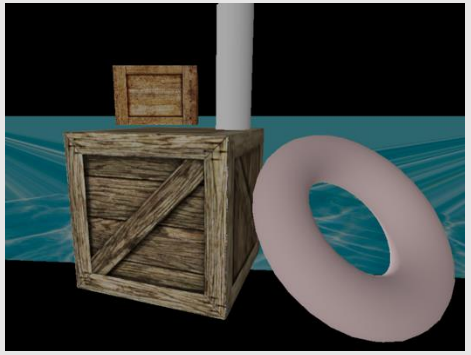
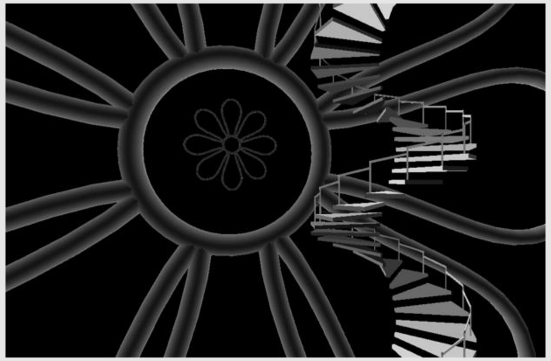
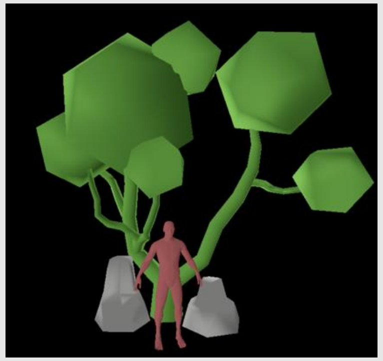
Get creative!

- 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

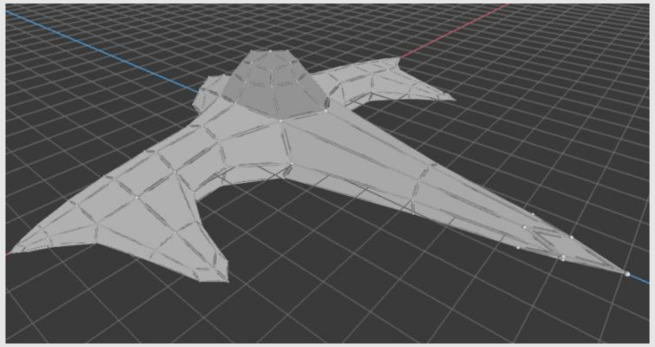
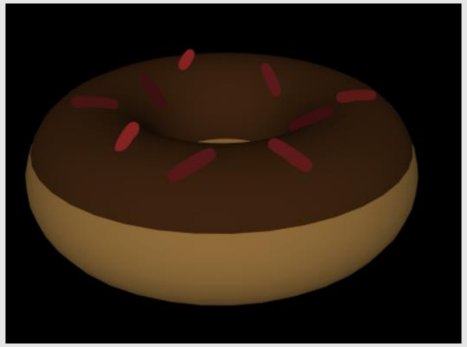
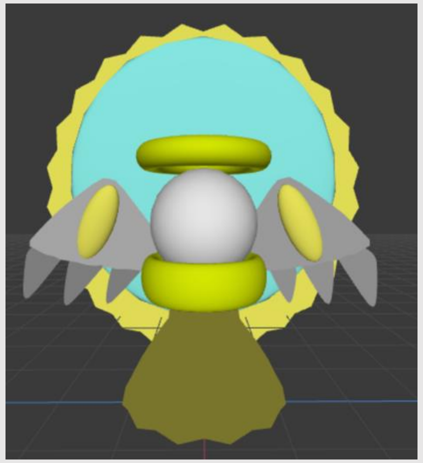
- The best work is showcased at the end of the semester

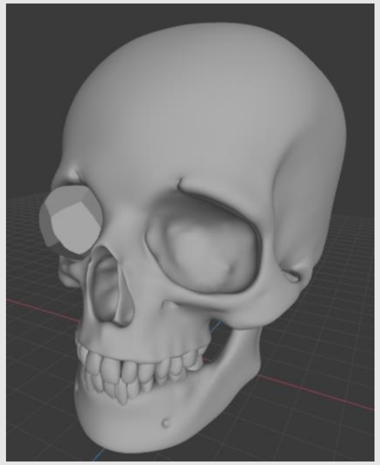
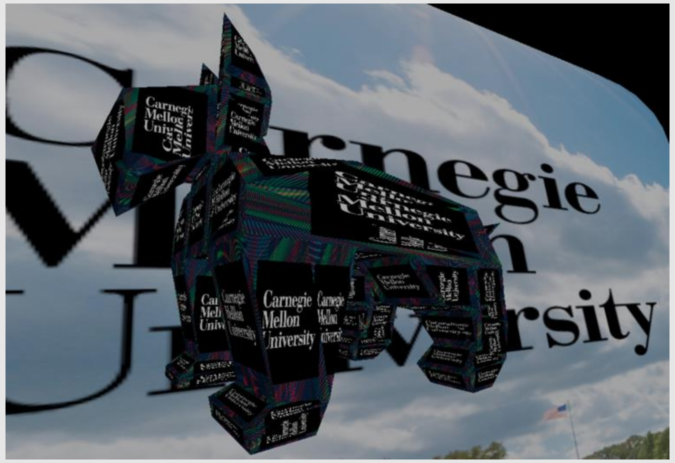
A1 Past Creations

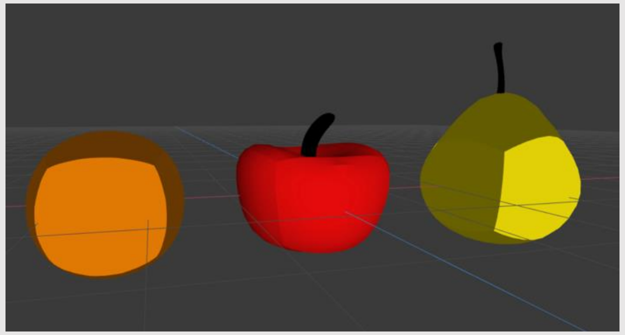




Normal & Displacement Mapping
normal mapping
displacement mapping
Describe surface material properties




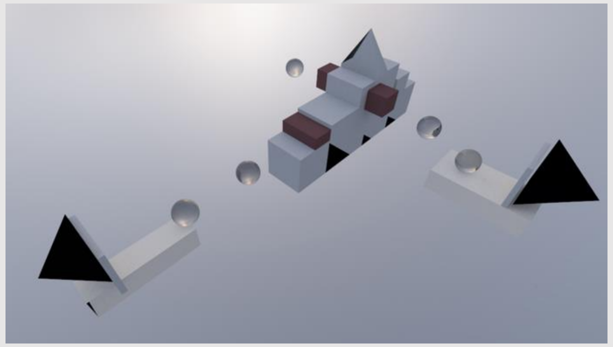
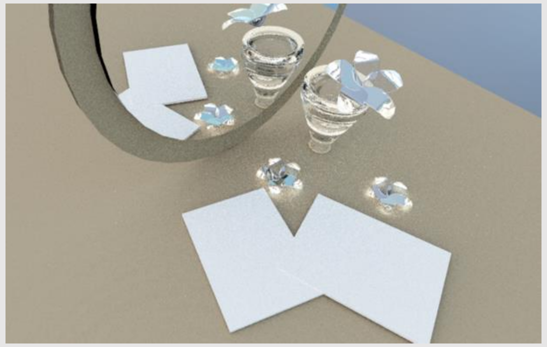
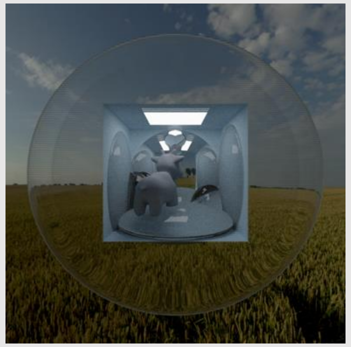
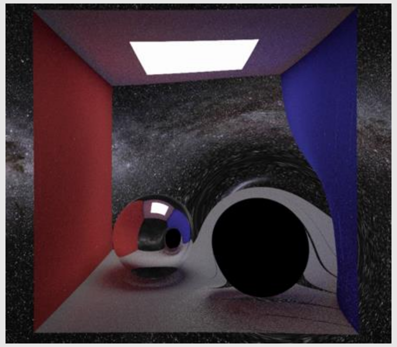
A2 Past Creations

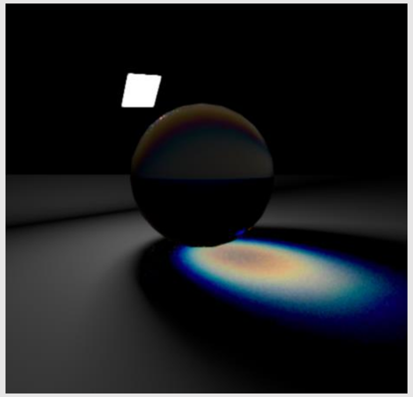




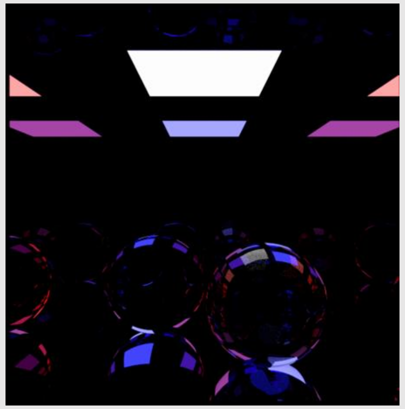





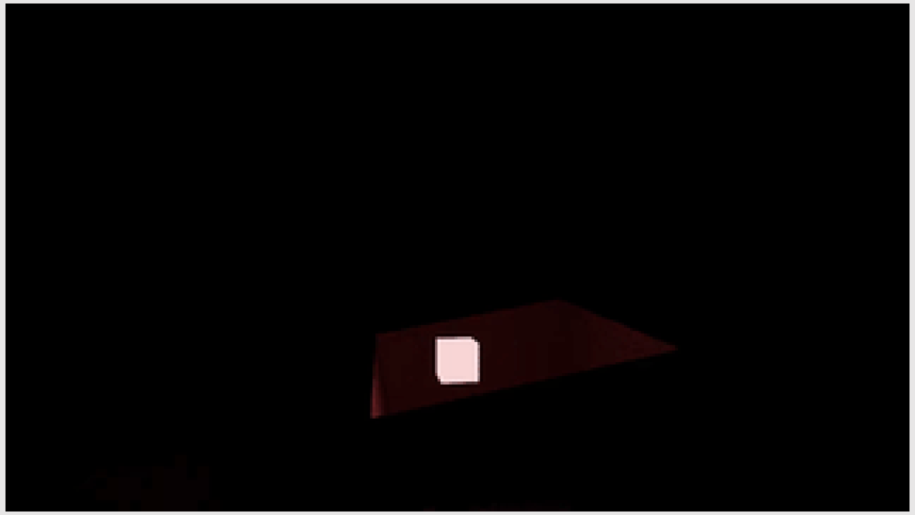
A3 Past Creations

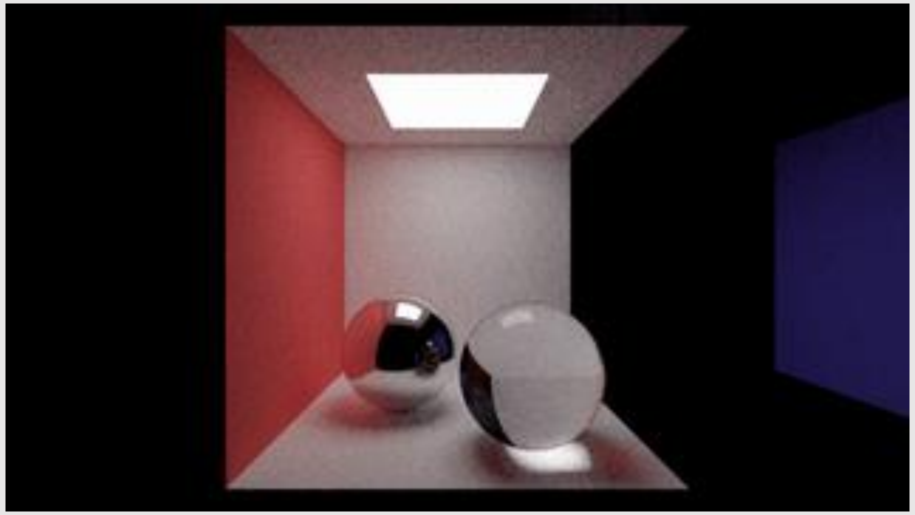









A4 Past Creations



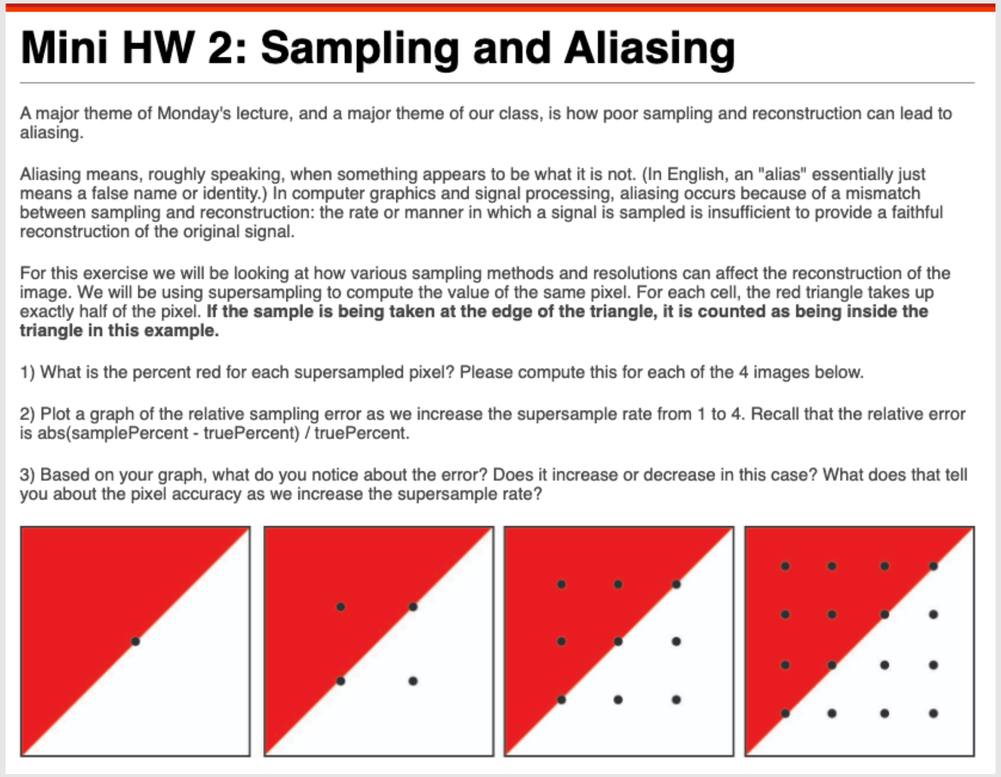


Is this entire class programming?

Hint: no

Writtens

- **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



Mini HW 2: Sampling and Aliasing

A major theme of Monday's lecture, and a major theme of our class, is how poor sampling and reconstruction can lead to aliasing.

Aliasing means, roughly speaking, when something appears to be what it is not. (In English, an "alias" essentially just means a false name or identity.) In computer graphics and signal processing, aliasing occurs because of a mismatch between sampling and reconstruction: the rate or manner in which a signal is sampled is insufficient to provide a faithful reconstruction of the original signal.

For this exercise we will be looking at how various sampling methods and resolutions can affect the reconstruction of the image. We will be using supersampling to compute the value of the same pixel. For each cell, the red triangle takes up exactly half of the pixel. **If the sample is being taken at the edge of the triangle, it is counted as being inside the triangle in this example.**

1) What is the percent red for each supersampled pixel? Please compute this for each of the 4 images below.

2) Plot a graph of the relative sampling error as we increase the supersample rate from 1 to 4. Recall that the relative error is abs(samplePercent - truePercent) / truePercent.

3) Based on your graph, what do you notice about the error? Does it increase or decrease in this case? What does that tell you about the pixel accuracy as we increase the supersample rate?

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

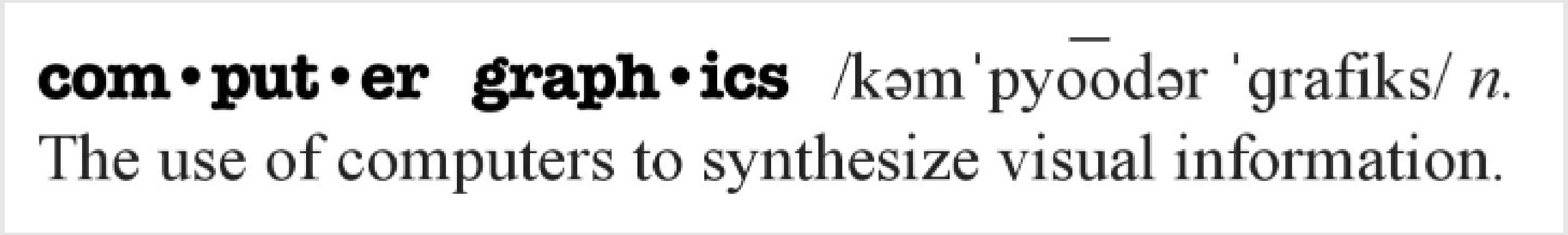
What We Really Want From You

- 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

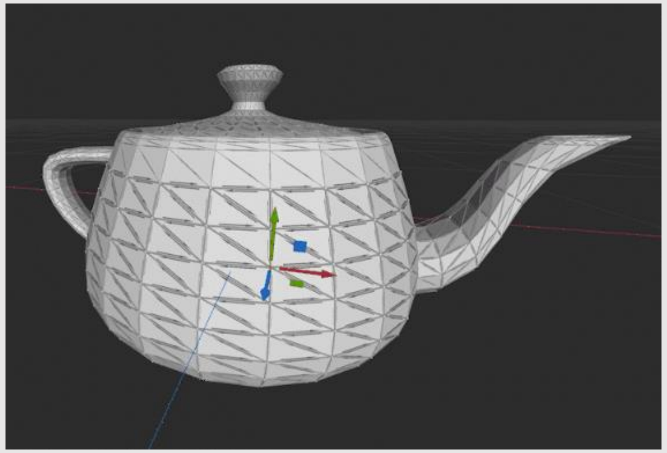
Before that,

What is Computer Graphics


com•put•er graph•ics /kəmˈpyo͞odər ˈgrafiks/ *n.*
The use of computers to synthesize visual information.

computer vision $\longrightarrow$

computer graphics $\longrightarrow$



What is Computer Graphics

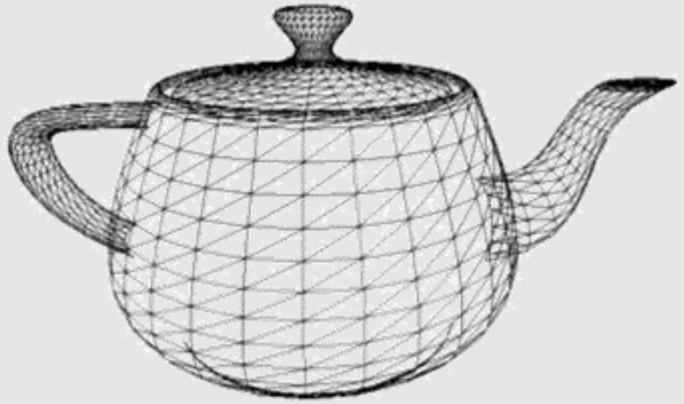


Image credit: Henrik Wann Jensen

Input: description of a scene

3D surface geometry (e.g., triangle meshes)
surface materials
lights
camera

Output: image

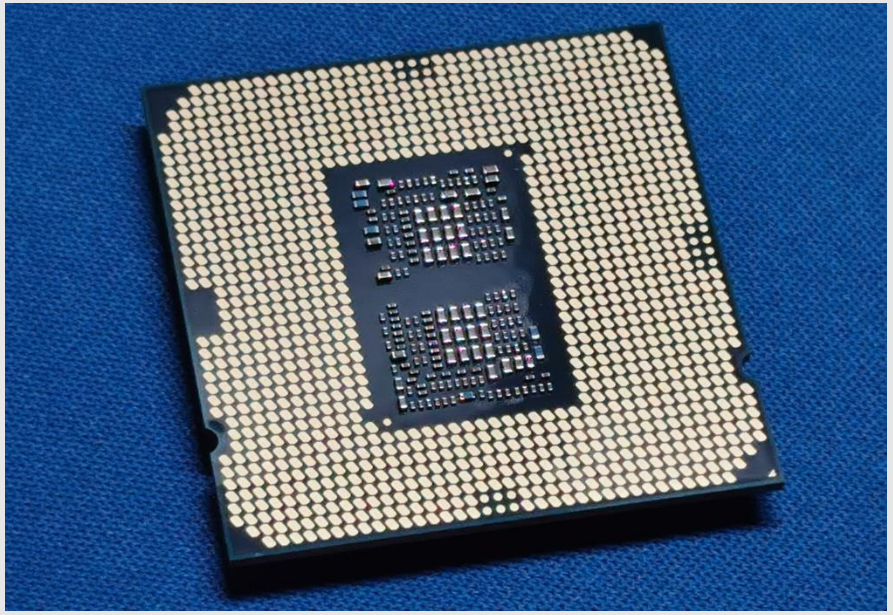
Drawing an image requires doing millions of the same operations across millions of triangles, lights, pixels, etc.

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


Core i7 (2008) Intel

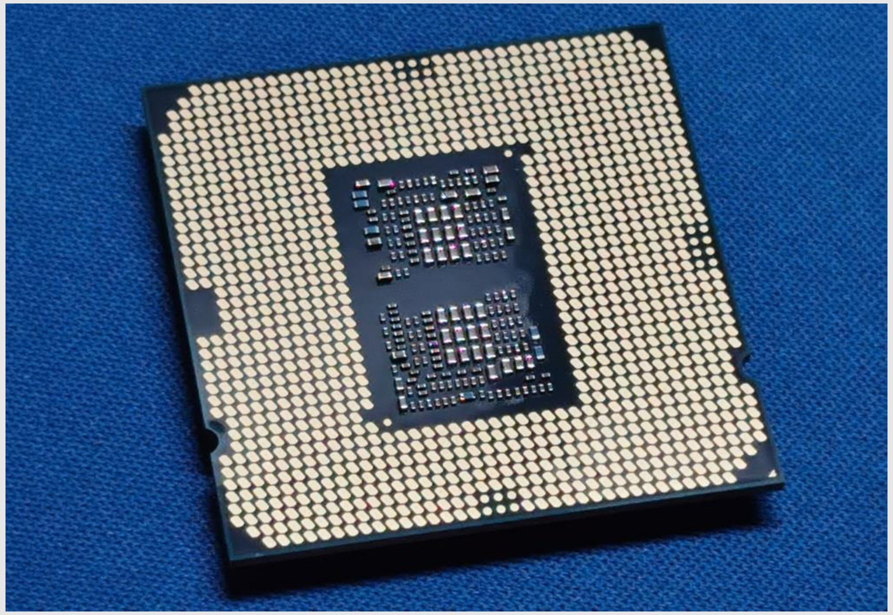
The CPU

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

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

- **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


Core i7 (2008) Intel

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

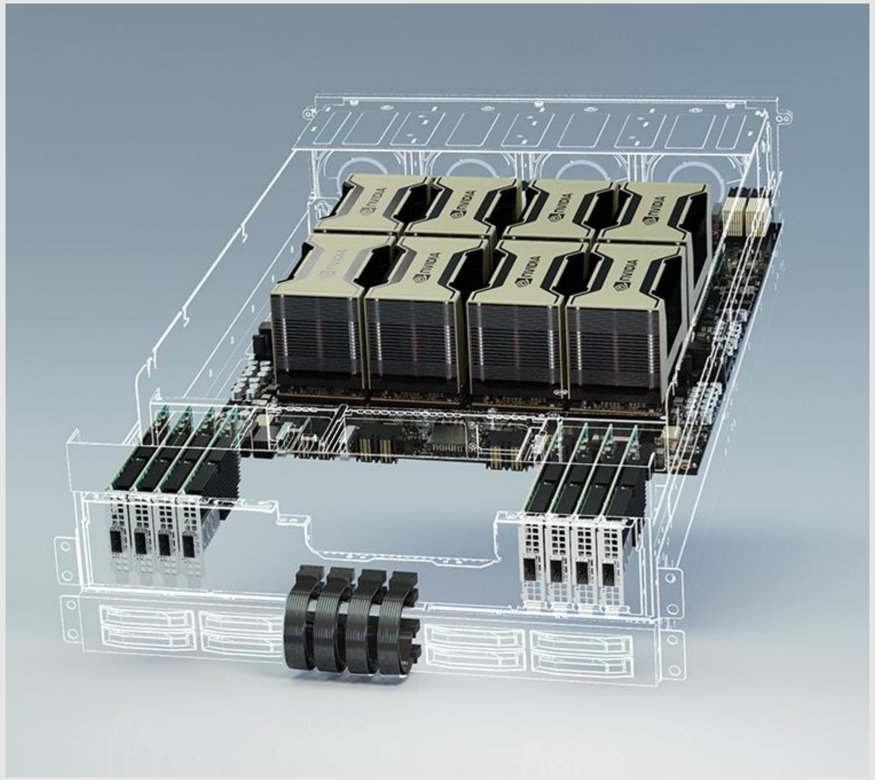
Geforce 256 (1999) Nvidia

The GPGPU

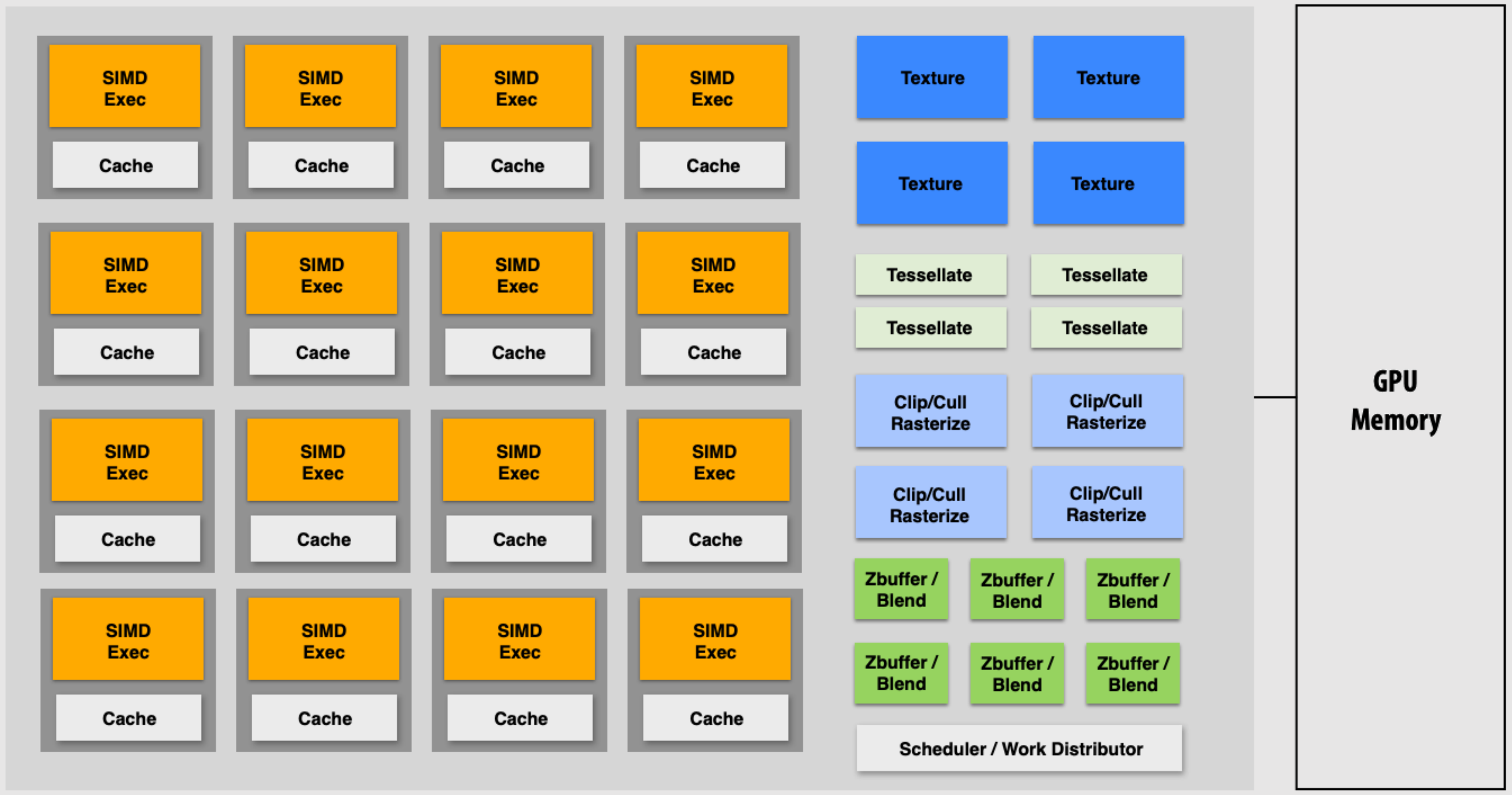
- **‘General Purpose’ Graphics Processing Unit**
    - Also known as the ‘modern GPU’
    - Sacrifices specialized hardware components for more general operations

- 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

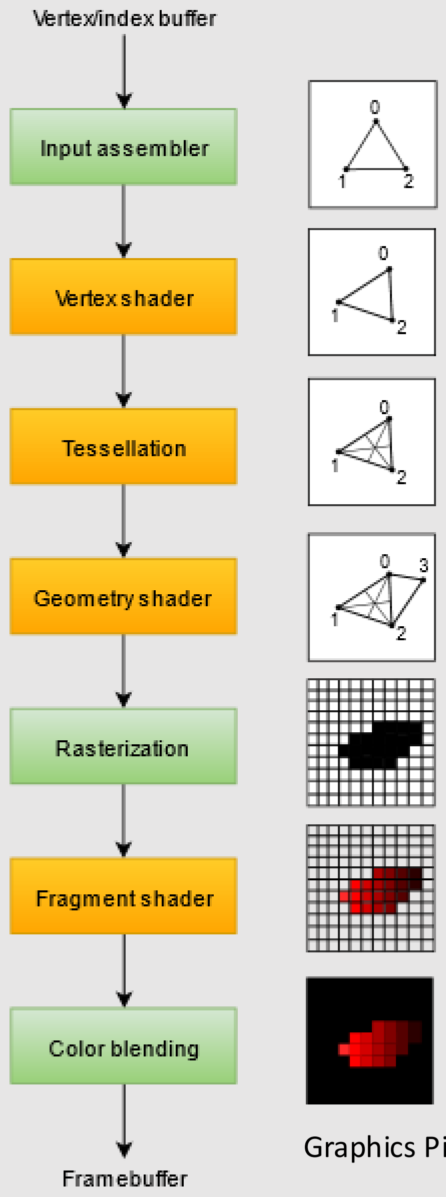
- **Paradigm shift:** sacrifice fixed function for more programmability


Data Centers (2020) Nvidia

The GPU


SIMD Exec
Cache
SIMD Exec
Cache
SIMD Exec
Cache
SIMD Exec
Cache
SIMD Exec
Cache
SIMD Exec
Cache
SIMD Exec
Cache
SIMD Exec
Cache
SIMD Exec
Cache
SIMD Exec
Cache
SIMD Exec
Cache
SIMD Exec
Cache
SIMD Exec
Cache
SIMD Exec
Cache
SIMD Exec
Cache
SIMD Exec
Cache
Texture
Texture
Texture
Texture
Tessellate
Tessellate
Tessellate
Tessellate
Clip/Cull Rasterize
Clip/Cull Rasterize
Clip/Cull Rasterize
Clip/Cull Rasterize
Zbuffer / Blend
Zbuffer / Blend
Zbuffer / Blend
Zbuffer / Blend
Zbuffer / Blend
Zbuffer / Blend
Scheduler / Work Distributor
GPU Memory

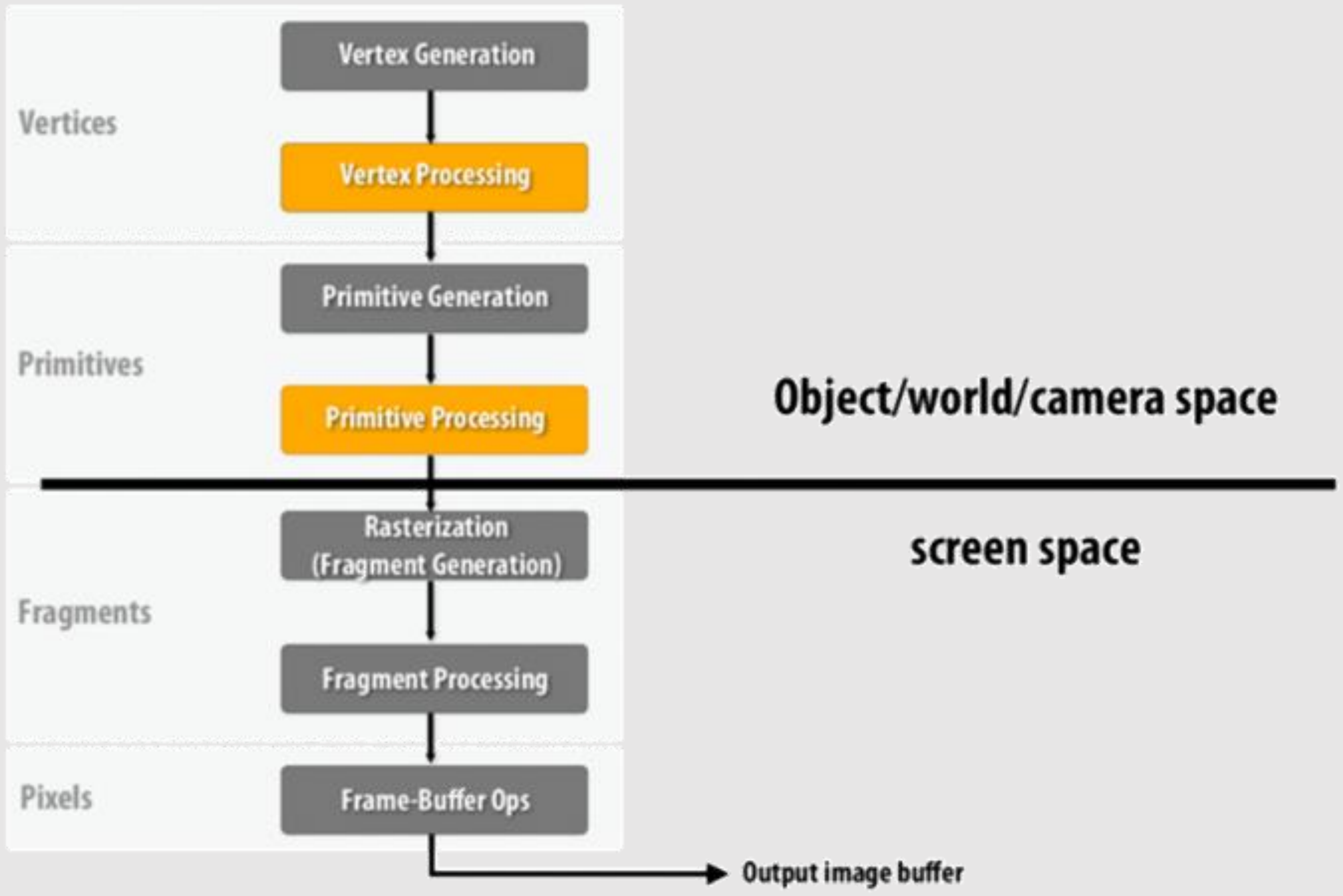
The Graphics Pipeline



Graphics Pipeline Tutorial (2019) Vulkan

- 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.

Change Of Space

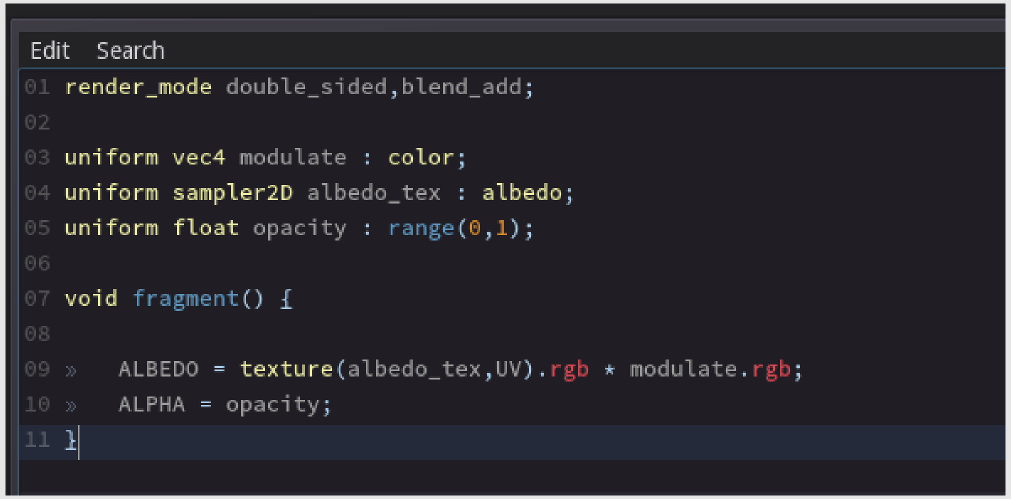


- Half the pipeline is in 3D, half is in 2D
  - Remember: we start with a 3D scene descriptor and end with a 2D image

- Moving from 3D to 2D scene provides many benefits:
  - Higher precision operations
  - Faster computations
  - Easier parallelism
  - Less data to manage
  - Less operations overall

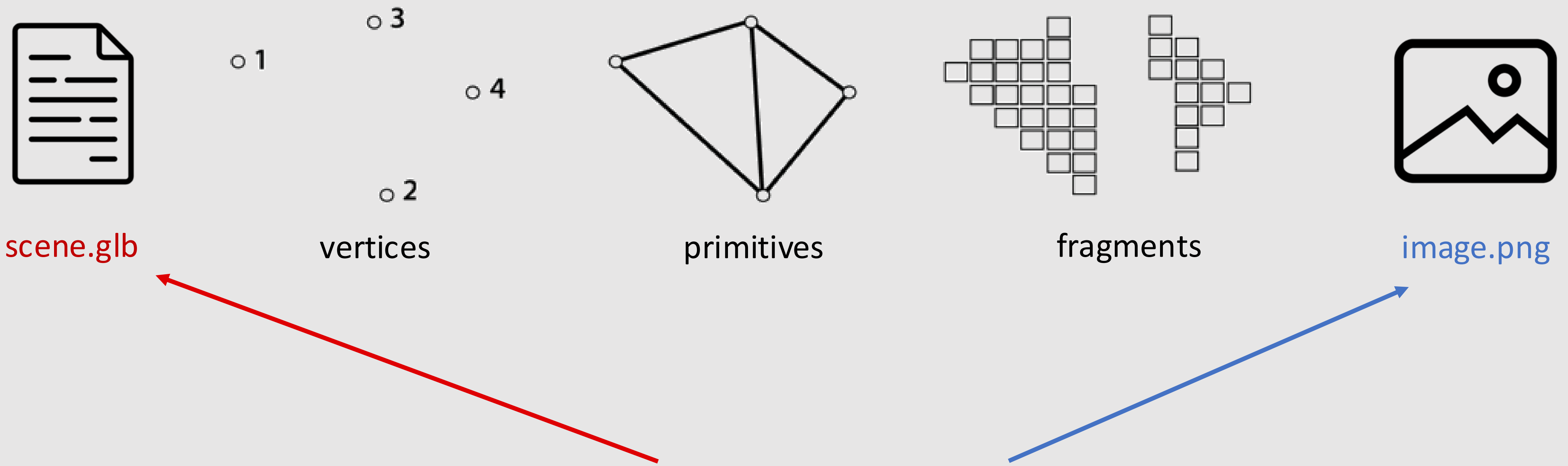
Side Note: What Is A Shader?

- 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



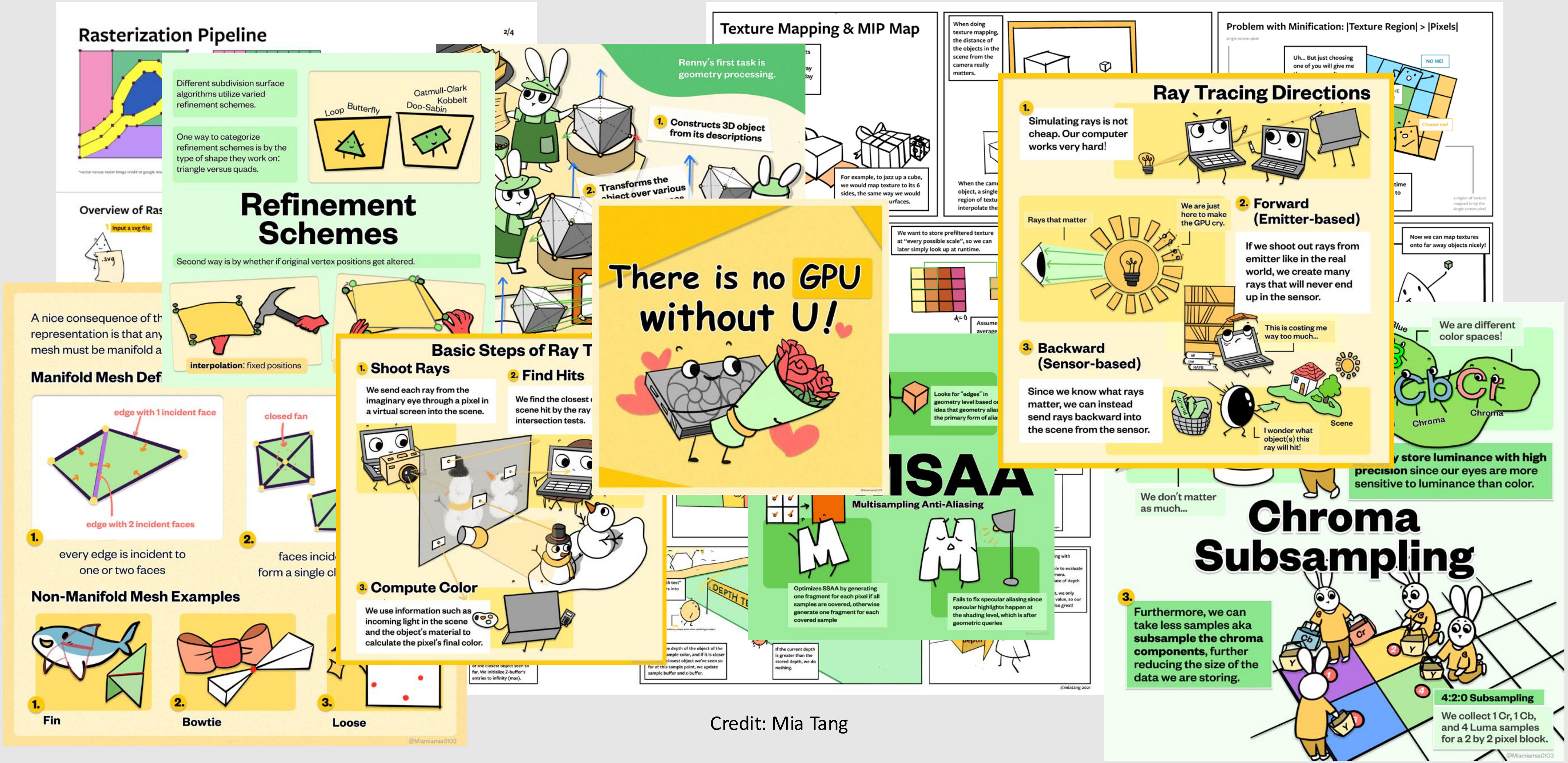
```
render_mode double_sided,blend_add;

uniform vec4 modulate : color;
uniform sampler2D albedo_tex : albedo;
uniform float opacity : range(0,1);

void fragment() {

	ALBEDO = texture(albedo_tex,UV).rgb * modulate.rgb;
	ALPHA = opacity;
}
```

3D Graphics Systems Stack



Converting data into … well, more data
But this data is pretty!

Much More Computer Graphics To Learn!


Rasterization Pipeline
Overview of Ras
Refinement Schemes
Different subdivision surface algorithms utilize varied refinement schemes.
One way to categorize refinement schemes is by the type of shape they work on: triangle versus quads.
Loop
Butterfly
Catmull-Clark
Kobbelt
Doo-Sabin
Second way is by whether if original vertex positions get altered.
interpolation: fixed positions
Renny's first task is geometry processing.
1. Constructs 3D object from its descriptions
2. Transforms the object over various
Manifold Mesh Def
edge with 1 incident face
edge with 2 incident faces
closed fan
1. every edge is incident to one or two faces
2. faces incid form a single cl
Non-Manifold Mesh Examples
1. Fin
2. Bowtie
3. Loose
Basic Steps of Ray T
1. Shoot Rays
We send each ray from the imaginary eye through a pixel in a virtual screen into the scene.
2. Find Hits
We find the closest scene hit by the ray intersection tests.
3. Compute Color
We use information such as incoming light in the scene and the object's material to calculate the pixel's final color.
There is no GPU without U!
Texture Mapping & MIP Map
Problem with Minification: |Texture Region| > |Pixels|
Ray Tracing Directions
1. Simulating rays is not cheap. Our computer works very hard!
2. Forward (Emitter-based)
If we shoot out rays from emitter like in the real world, we create many rays that will never end up in the sensor.
Rays that matter
We are just here to make the GPU cry.
This is costing me way too much...
3. Backward (Sensor-based)
Since we know what rays matter, we can instead send rays backward into the scene from the sensor.
I wonder what object(s) this ray will hit!
Scene
MSAA
Multisampling Anti-Aliasing
Optimizes SSAA by generating one fragment for each pixel if all samples are covered, otherwise generate one fragment for each covered sample
Fails to fix specular aliasing since specular highlights happen at the shading level, which is after geometric queries
We are different color spaces!
Cb Cr
Chroma
y store luminance with high precision since our eyes are more sensitive to luminance than color.
We don't matter as much...
Chroma Subsampling
3. Furthermore, we can take less samples aka subsample the chroma components, further reducing the size of the data we are storing.
4:2:0 Subsampling
We collect 1 Cr, 1 Cb, and 4 Luma samples for a 2 by 2 pixel block.

Credit: Mia Tang