

Final Exam Review

Part 2

15-462 / 15-662 Computer Graphics

Numerical Integration

- In rendering (global illumination), what are we integrating, i.e., what integral do we want to evaluate?
- How do we use the trapezoid rule to integrate a function?
- How does work increase with dimensionality of our function?
 - This is why we typically use Monte Carlo integration in graphics!
- Give a high level overview of the process of Monte Carlo integration
- What is a probability density function (PDF)?
- What is a Cumulative Distribution Function (CDF)?
- The Inversion Method can be used to correctly draw a sample from a PDF.
 - Sketch the overall step by step process for using the Inversion Method
- What is rejection sampling? Show how to use rejection sampling to sample area of a circle, volume of a sphere, directions on a sphere, and solid angles from a hemisphere.

Monte Carlo Rendering

- **What is Expected Value? .. Variance? .. Bias?**
- **What is Importance Sampling? How is it used in cosine-weighted sampling of the hemisphere?**
- **What is the Monte Carlo method (in general)?**
- **Explain how a Monte Carlo method can be used to solve the Rendering Equation.**
- **What is Russian Roulette and why do we need it?**
- **How can we use Russian Roulette and still have an unbiased estimator?**

Variance Reduction

- Be familiar with the following expression for Monte Carlo integration. What is the role of each term?

$$I = \lim_{n \rightarrow \infty} V(\Omega) \frac{1}{n} \sum_{i=1}^n f(X_i)$$

- Give an example of how we can reduce variance in our rendered results in a path tracing algorithm without increasing the number of samples.
- What does it mean for an estimator to be consistent?
- What does it mean for an estimator to be unbiased?
- Give a concrete example of how a renderer could give a biased estimate of an image. Is the renderer in your example consistent? Explain your answer.
- Give five examples of how you can reweight samples in a pathtracing algorithm in order to do importance sampling.
- What are the main ideas behind bidirectional path tracing?
- How would you enumerate all possible paths in a scene?
- How does Metropolis-Hastings sampling work?
- Assume you have code to generate random paths and code to mutate existing paths. Write pseudocode for Metropolis-Hastings path tracing.

More Variance Reduction

- **What is Stratified Sampling?**
- **Why is it preferred to random sampling?**
- **Hammersley and Halton points are pseudo random sampling techniques to generate points with low discrepancy. What is discrepancy? Why do we want to generate low discrepancy samples?**
- **Give a concise one sentence description of each of the following rendering algorithms that makes it clear the differences between them. Use a diagram to illustrate your description:**
 - **Rasterization**
 - **Ray casting**
 - **Ray tracing**
 - **Path tracing**
 - **Bidirectional path tracing**
 - **Metropolis Light Transport**
 - **Photon Mapping**
 - **Radiosity**
- **Which of these algorithms are best for capturing reflective surfaces? caustics? color bleeding? subsurface scattering? refraction? ...**

Intro to Animation

- **How were the first animations created? How were the first films created? Give some examples.**
- **Describe some of the first computer generated animations, giving the developer / artist and timeframe.**
- **Animations are created from keyframes. How do we interpolate between those keyframes?**
- **Why do we avoid splines of degree higher than three in computer graphics?**
- **Write a cubic polynomial $P(t)$ in parameter t , which may describe a cubic spline.**
- **What are the constraints for $P(t)$ to interpolate endpoints p_1 at time $t=0$ and p_2 at time $t=1$?**
- **What are the constraints for $P(t)$ to have tangent vector r_1 at time $t=0$ and r_2 at time $t=1$?**

Intro to Animation

- This pair of constraints describes a Hermite spline. Derive the polynomial coefficients for the Hermite spline and write the cubic polynomial in terms of p_1 , p_2 , r_1 , and r_2 .
- Put your result in matrix form.
- Give properties of the Hermite spline in terms of continuity (C1, C2, etc.), interpolation, and local control.
- What type of spline has C2 continuity and interpolation, but not local control?
- What type of spline has C2 continuity and local control, but does not interpolate its key points?
- What are Catmull-Rom splines? How are tangents computed for Catmull-Rom splines?
- What are blend shapes and where are they used? What exactly is interpolated when using blend shapes for animation?

Intro to Animation

- **Bezier, Hermite, and Catmull-Rom splines are really all the same thing. Any one representation can be converted to any of the others. Explain the differences between them.**
- **Be able to express a Hermite spline in different ways — as a cubic polynomial, in matrix form, or derive it from its control parameters.**
- **How do we ensure continuity between cubic spline segments? C0 continuity? .. C1 continuity? .. is C2 continuity possible in general?**

Dynamics and Time Integration

- What is the “animation equation”?
- What is the difference between an ODE and a PDE? Give some examples of systems we can simulate by integrating an ODE.
- Sketch an overall system for simulating an ODE using a block diagram. Be clear about what is the state, how you advance the state forward in time, and what integrator you are choosing.
- When is the Euler-Lagrange equation useful?
- Be able to work through a simple example of obtaining dynamic equations of motion using Lagrangian mechanics.
- Describe how to put together a mass-spring system to simulate cloth.
- What are the forces on each cloth “particle”?
- What is Forward Euler integration and what is its disadvantage? Can you show a simple example where it fails?
- What is Backward Euler integration and what are its pros and cons?
- What is Symplectic Euler integration?

Kinematics

- **Be able to describe an optimization problem in standard form (parameters, objective, and constraints) and give a couple of simple examples.**
- **How do you know you have a minimum of an objective function in an optimization problem without constraints? What must be true?**
- **What is the difference between forward and inverse kinematics?**
- **Is forward kinematics an optimization problem? Is inverse kinematics an optimization problem?**
- **Write an expression for the forward kinematics of a simple character or robot.**
- **What is the Jacobian? Compute the Jacobian for a simple character or robot.**
- **What is the Jacobian Transpose technique for inverse kinematics? Is it guaranteed to converge? What does that mean in practice? Does it give a locally optimal solution or a globally optimal one? Why?**

Physically Based Animation and PDEs

- What is the difference between a PDE and an ODE? Give examples of when you might use each one and why.
- Interpret this sentence using an equation and a diagram: Solving a PDE looks like *“use neighbor information to get velocity (...and then add a little velocity each time)”*
- Burger’s equation is first order in time and second order in space. What does that mean? What are the orders of the Laplace equation? The heat equation? The wave equation? Be able to figure out the order of an equation from an expression of the equation itself.
- What are examples of questions we can answer using the Laplace equation, the heat equation, and the wave equation respectively?
- Outline the basic strategy for solving a PDE.
- What is the Laplace operator? Write it out as a sum of partial derivatives.
- Copy down the heat equation from the slides. Write out the process of solving this equation using Forward Euler integration.
- Copy down the wave equation from the slides. Write out the process of solving this equation using Forward Euler integration.

Physically Based Animation and PDEs

- Be able to use the grid version of the Laplacian to do smoothing on a grid.

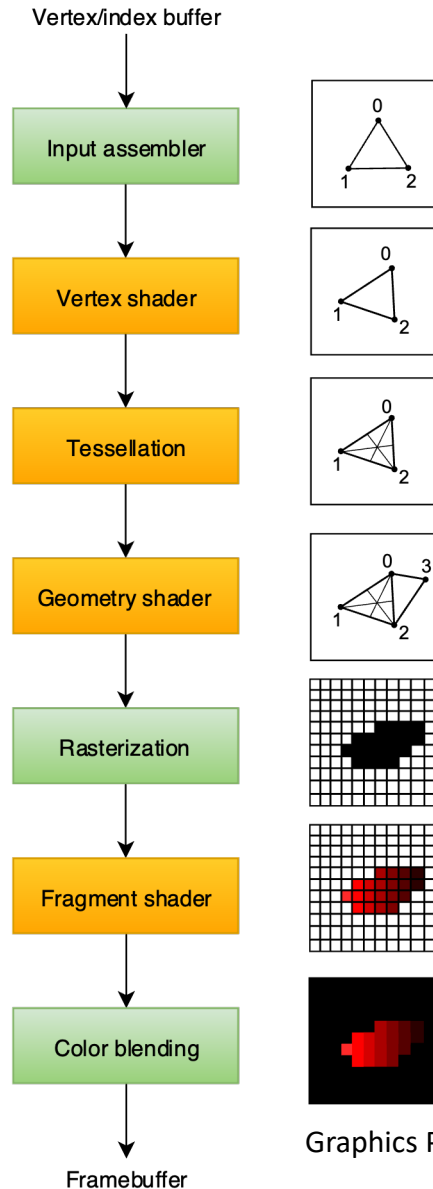
You may find these slides from last semester's review helpful

- **A1: Rasterization**
- A2: Geometry
- A3: Rendering
- A4: Animation

Pixel Pushing

- Shaders
 - Vertex Shader
 - Fragment Shader
- Texturing
 - Nearest Neighbor
 - Bilinear Filtering
 - Trilinear Filtering
- Perspective Transform
- Scene Graphs

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

Nearest Neighbor Sampling

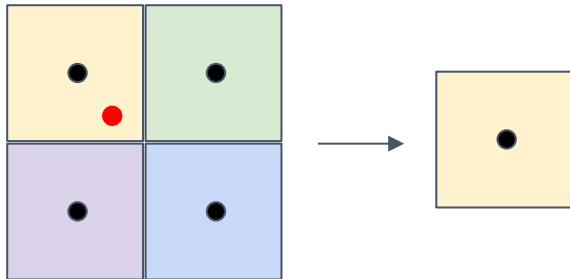
- **Idea:** Grab texel nearest to requested location in texture

$$x' \leftarrow \text{round}(x), \quad y' \leftarrow \text{round}(y)$$

- **Requires:**

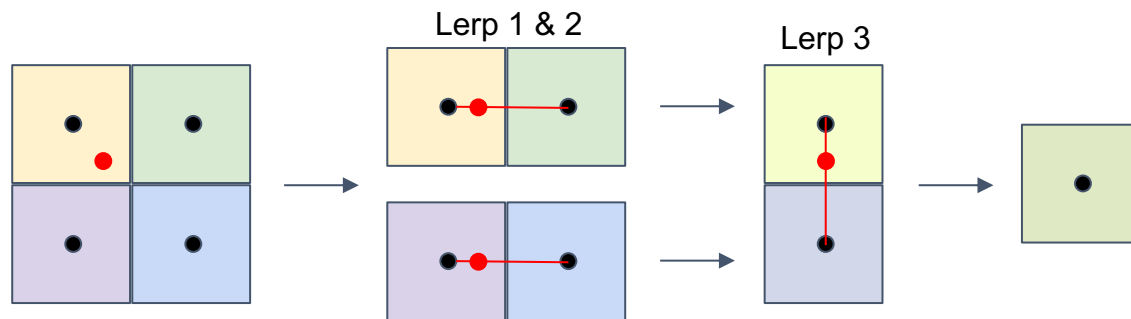
- 1 memory lookup
- 0 linear interpolations

$$t \leftarrow \text{tex.lookup}(x', y')$$



Bilinear Interpolation Sampling

- **Idea:** Grab nearest 4 texels and blend them together based on their inverse distance from the requested location
 - Blend two sets of pixels along one axis, then blend the remaining pixels
- **Requires:**
 - 4 memory lookup
 - 3 linear interpolations



$$x' \leftarrow \text{floor}(x), \quad y' \leftarrow \text{floor}(y)$$

$$\Delta x \leftarrow x - x'$$
$$\Delta y \leftarrow y - y'$$

$$t_{(x,y)} \leftarrow \text{tex.lookup}(x', y')$$

$$t_{(x+1,y)} \leftarrow \text{tex.lookup}(x' + 1, y')$$

$$t_{(x,y+1)} \leftarrow \text{tex.lookup}(x', y' + 1)$$

$$t_{(x+1,y+1)} \leftarrow \text{tex.lookup}(x', +1 y' + 1)$$

$$t_x \leftarrow (1 - \Delta x) * t_{(x,y)} + \Delta x * t_{(x+1,y)}$$

$$t_y \leftarrow (1 - \Delta x) * t_{(x,y+1)} + \Delta x * t_{(x+1,y+1)}$$

$$t \leftarrow (1 - \Delta y) * t_x + \Delta y * t_y$$

Trilinear Interpolation Sampling

- **Idea:** Perform bilinear interpolation on two layers of the mip-map that represents proper minification/magnification, blending the results together
- **Requires:**
 - 8 memory lookup
 - 7 linear interpolations

$$L_x^2 \leftarrow \frac{du^2}{dx} + \frac{dv^2}{dx}$$

$$L_y^2 \leftarrow \frac{du^2}{dy} + \frac{dv^2}{dy}$$

$$L \leftarrow \sqrt{\max(L_x^2, L_y^2)}$$

$$d \leftarrow \log_2 L$$

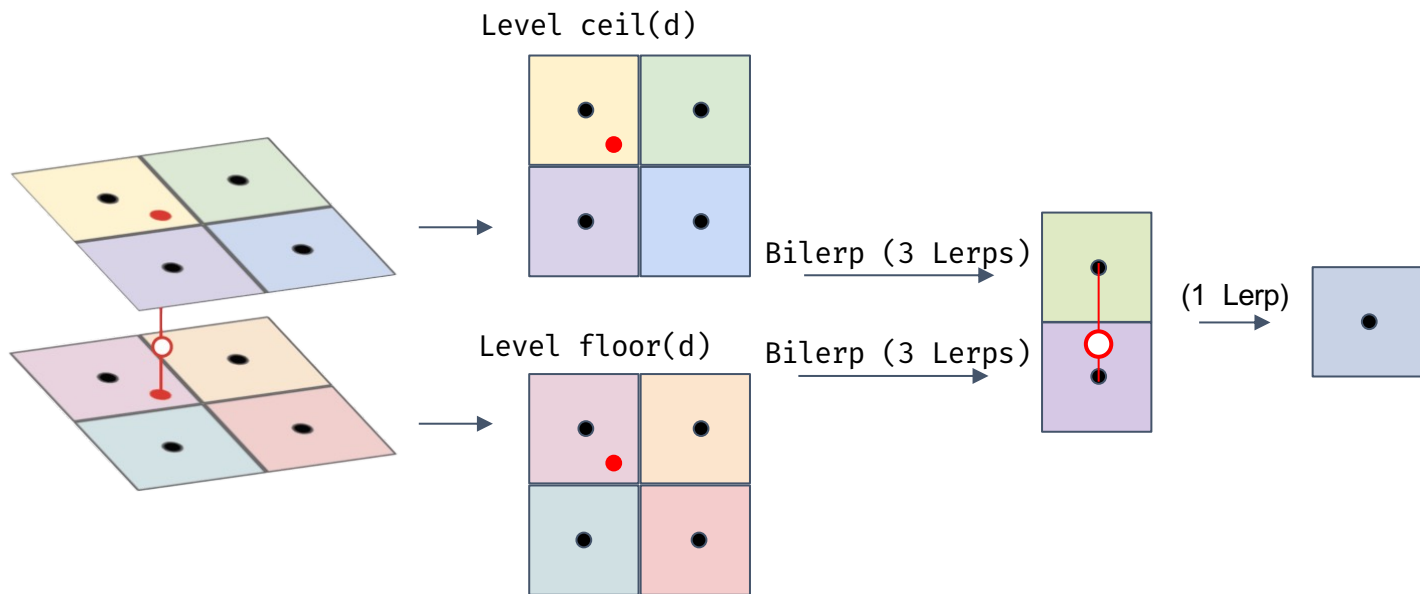
$$d' \leftarrow \text{floor}(d)$$

$$\Delta d \leftarrow d - d'$$

$$t_d \leftarrow \text{tex}[d']. \text{bilinear}(x, y)$$

$$t_{d+1} \leftarrow \text{tex}[d' + 1]. \text{bilinear}(x, y)$$

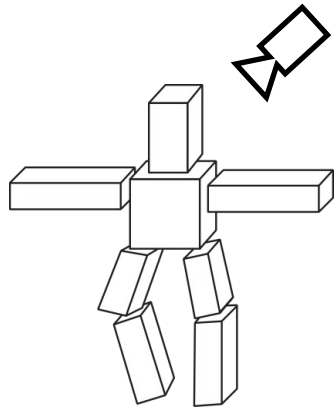
$$t \leftarrow (1 - \Delta d) * t_d + \Delta d * t_{d+1}$$



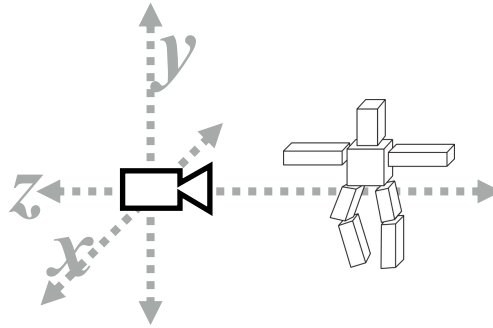
Perspective Projection



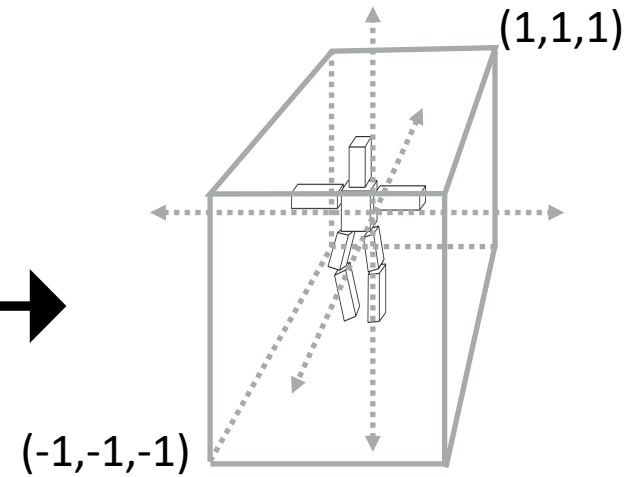
Perspective Projection



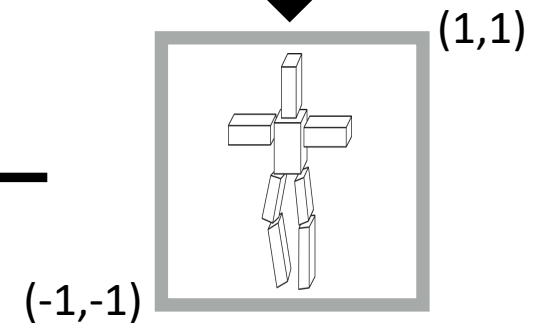
Original description of object.



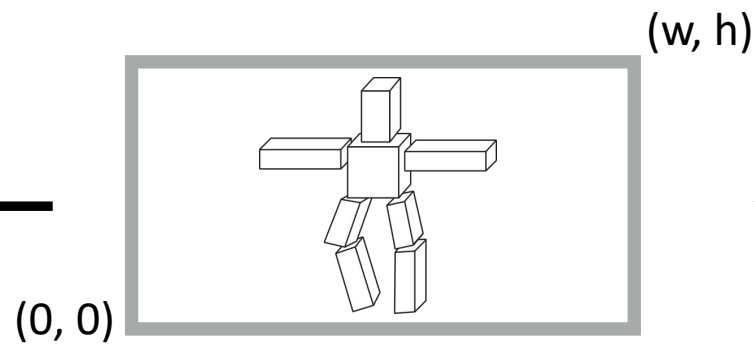
Object relative to camera.
Camera at origin looking down $-z$ axis.



Everything visible to camera mapped to a cube.



Everything visible to camera mapped to a cube.



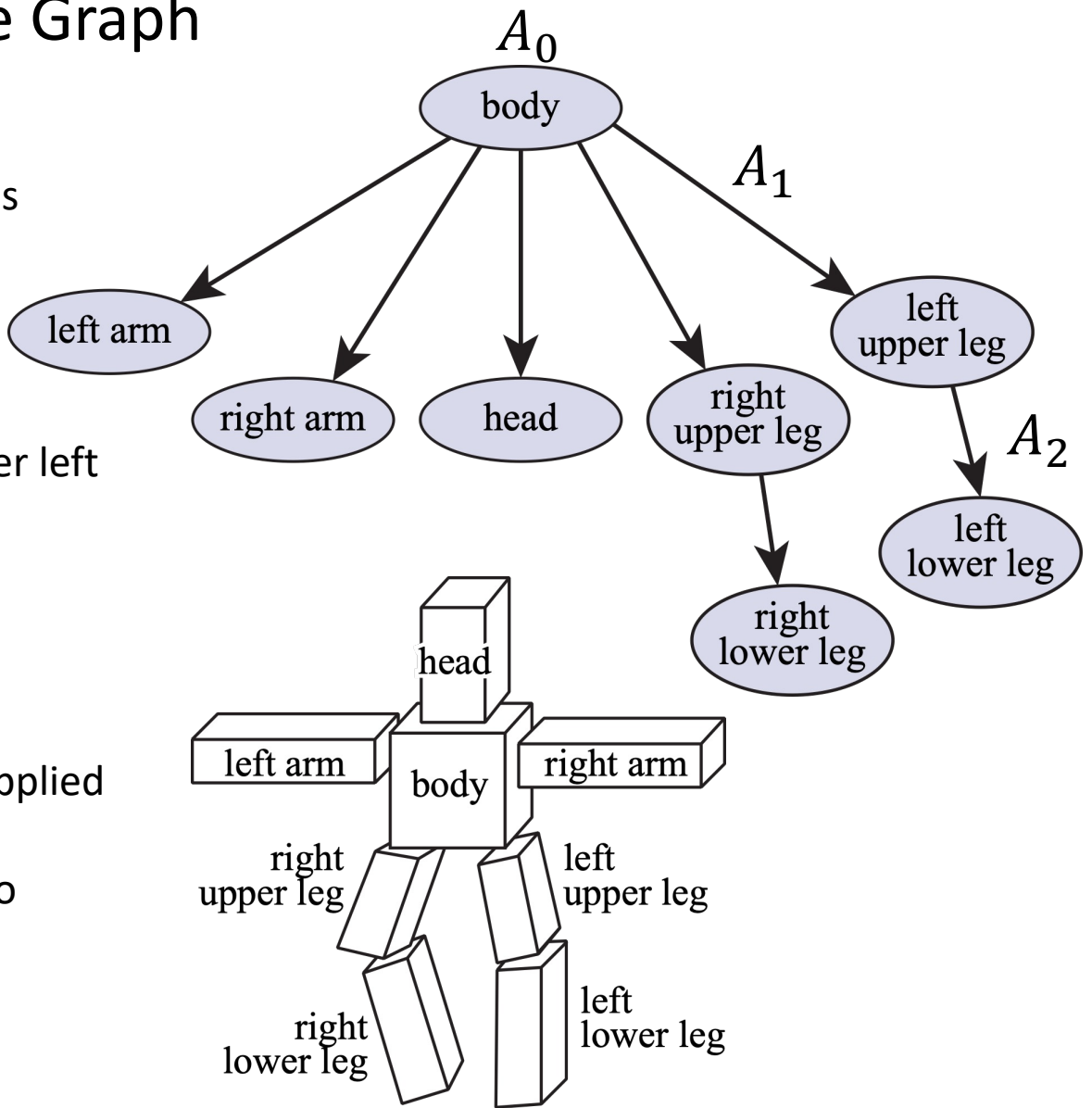
Coordinates stretched to image dims.
Image flipped upside down.



[Rasterization Stage]

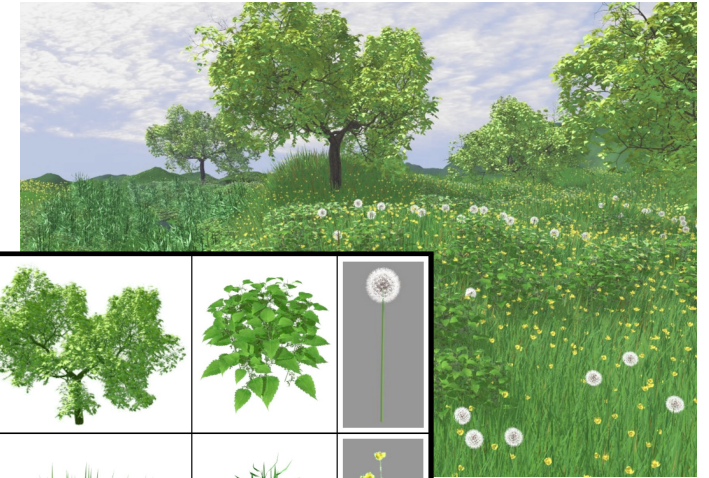
Scene Graph

- Suppose we want to build a skeleton out of cubes
- **Idea:** transform cubes in world space
 - Store transform of each cube
- **Problem:** If we rotate the left upper leg, the lower left leg won't track with it
 - **Better Idea:** store a hierarchy of transforms
 - Known as a **scene graph**
 - Each edge (+root) stores a linear transformation
 - Composition of transformations gets applied to nodes
 - Keep transformations on a stack to reduce redundant multiplication
- **Lower left leg transform:** $A_2A_1A_0$

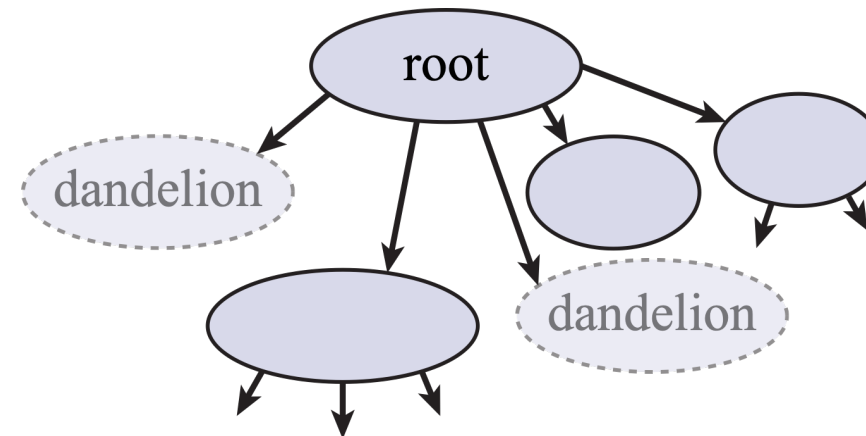
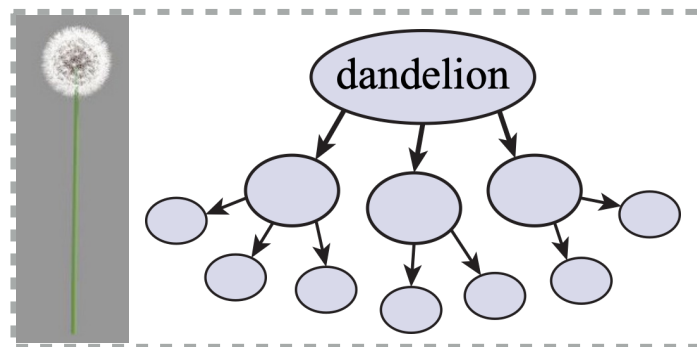


Instancing

- What if we want many copies of the same object in a scene?
 - Rather than have many copies of the geometry, scene graph, we can just put a “pointer” node in our scene graph
 - Saves a reference to a shared geometry
 - Specify a transform for each reference
 - **Careful:** Modifying the geometry will modify all references to it



Realistic modeling and rendering of plant ecosystems (1998) Deussen et al



- ~~A1: Rasterization~~

- **A2: Geometry**

- A3: Rendering

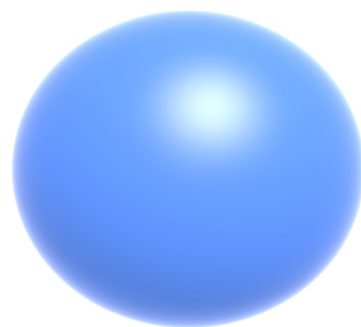
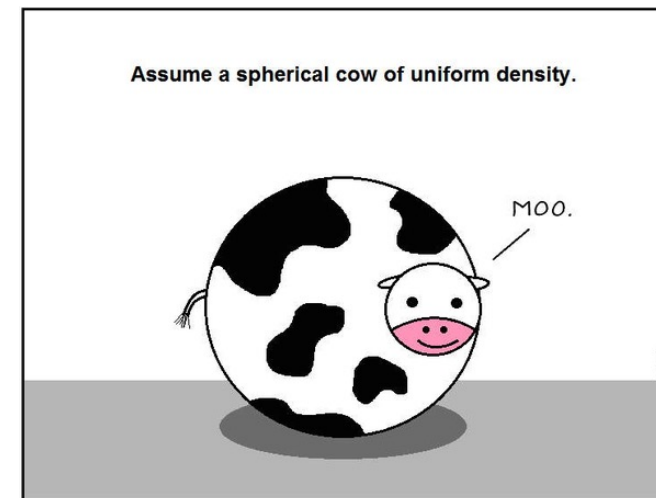
- A4: Animation

Meshes

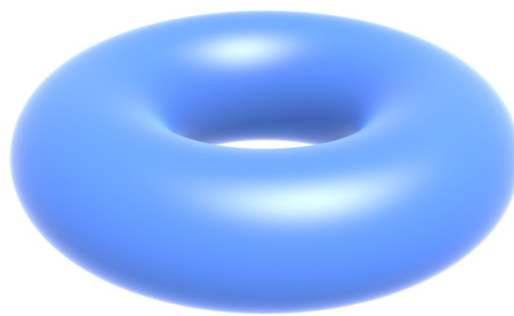
- Types of Geometric Representations
 - Algebraic Surfaces
 - CSG
 - Blobby
 - Level Set
 - Fractals
 - Point Cloud
 - Meshes
- Global Mesh Operations
 - Subdivision
 - Isotropic Remeshing
- Spatial Data Structures
 - BVH
 - KD-Tree
 - Uniform Grid
 - Quadtree/Octree

Algebraic Surfaces [Implicit]

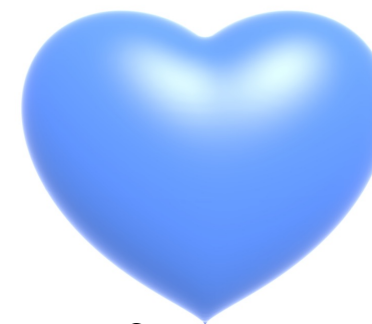
- Simple way to think of it: a surface built with algebra [math]
 - Generally thought of as a surface where points are some radius r away from another point/line/surface
- Easy to generate smooth/symmetric surfaces
 - Difficult to generate impurities/deformations



$$x^2 + y^2 + z^2 = 1$$



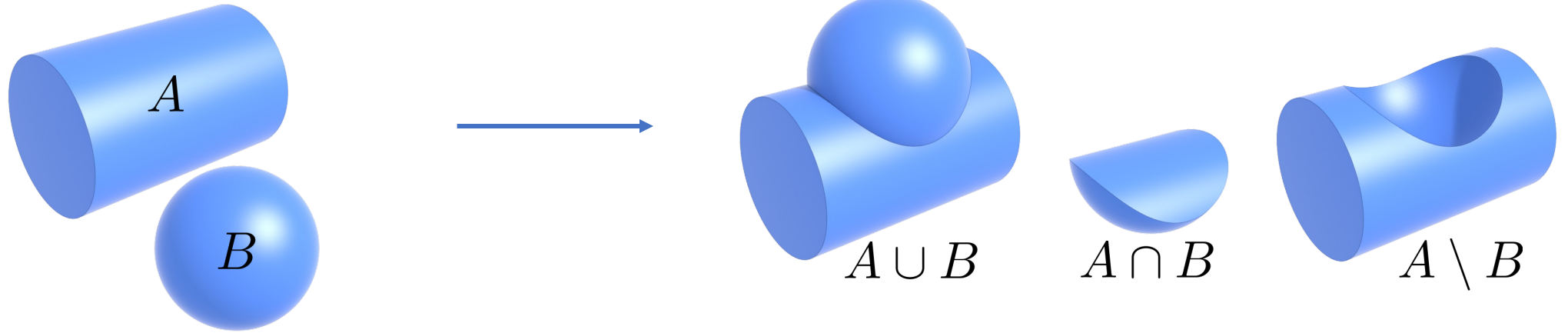
$$(R - \sqrt{x^2 + y^2})^2 + z^2 = r^2$$



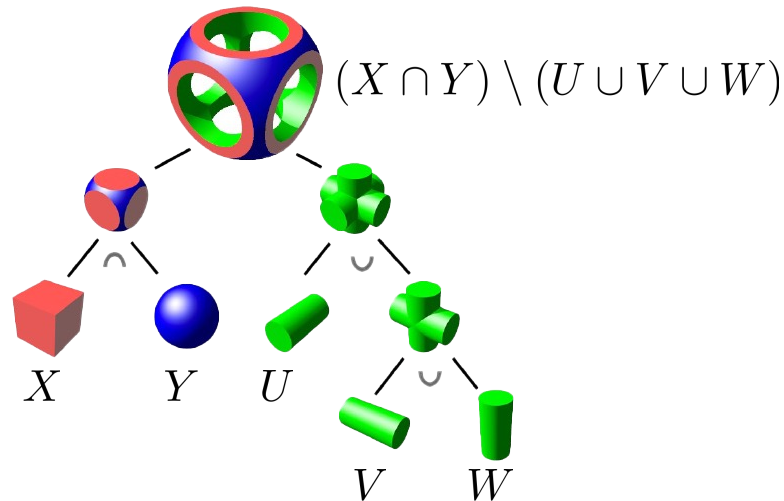
$$\left(x^2 + \frac{9y^2}{4} + z^2 - 1\right)^3 = x^2 z^3 + \frac{9y^2 z^3}{80}$$

Constructive Solid Geometry [Implicit]

- Build more complicated shapes via Boolean operations
 - Basic operations:

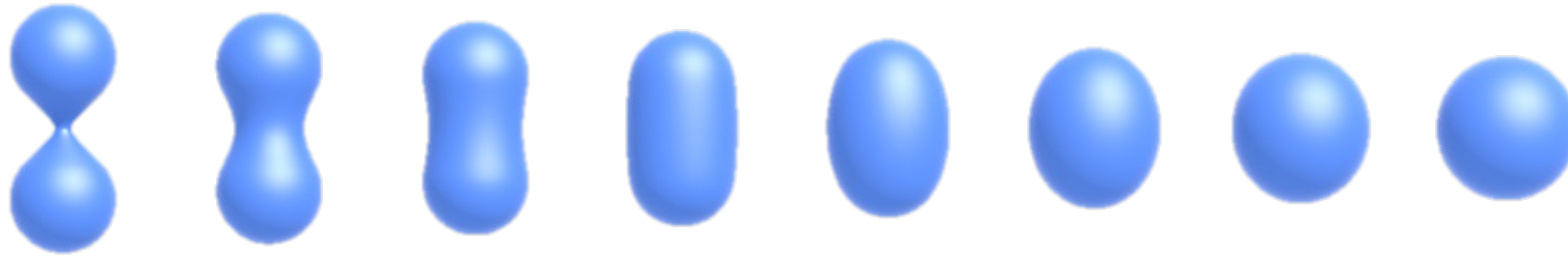


- Can be used to form complex shapes!

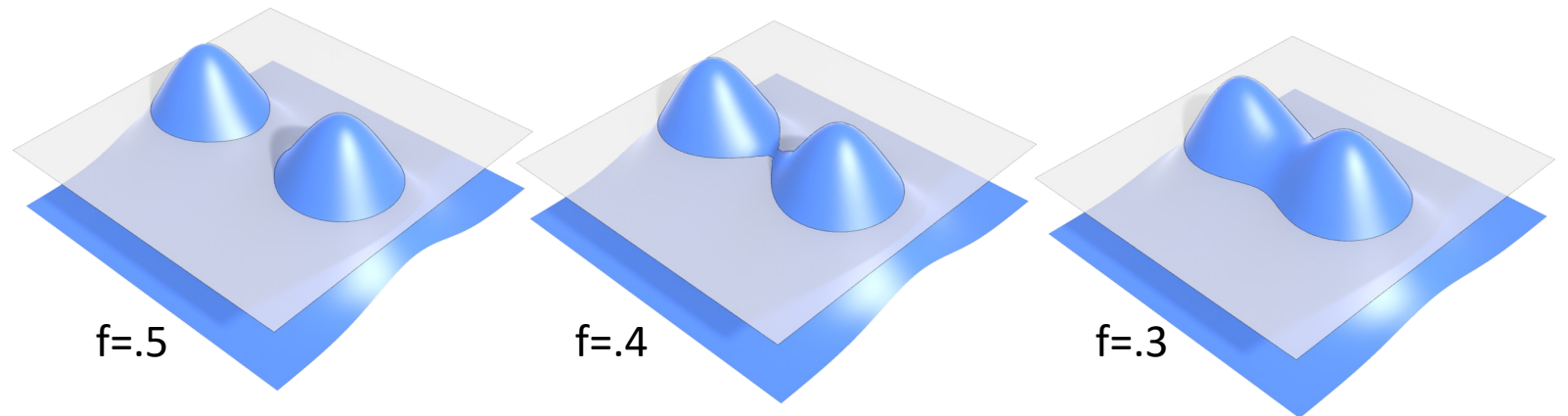


Bloppy Surfaces [Implicit]

- Instead of Booleans, gradually blend surfaces together:

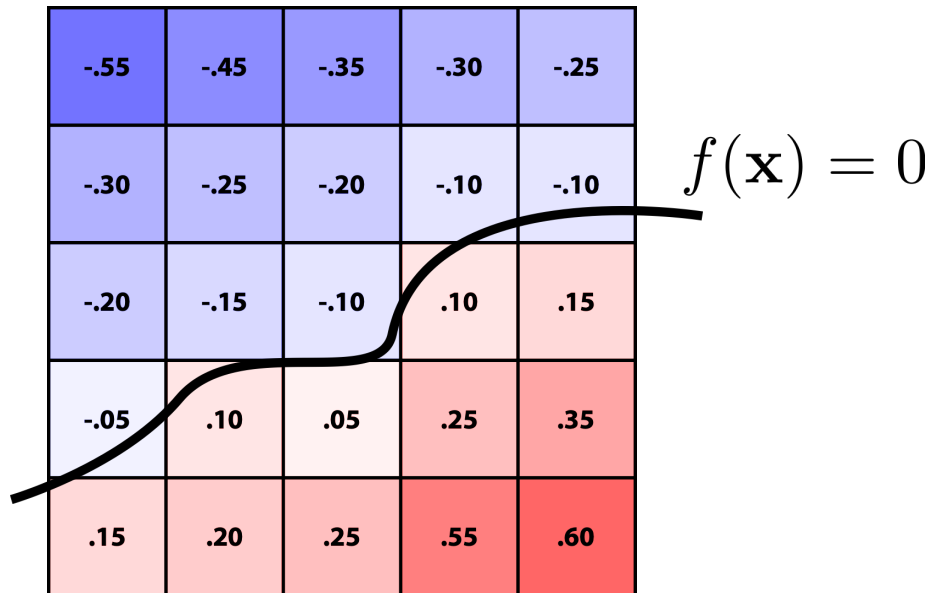


- Easier to understand in 2D: $\phi_p(x) := e^{-|x-p|^2}$ (Gaussian centered at p)
 $f := \phi_p + \phi_q$ (Sum of Gaussians centered at different points)



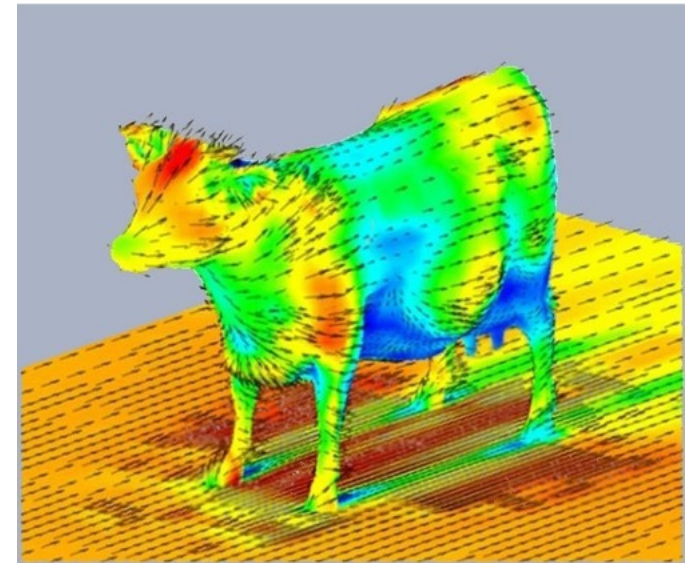
Level Set Methods [Implicit]

- Implicit surfaces have some nice features (e.g., merging/splitting)
 - But, hard to describe complex shapes in closed form
 - **Alternative:** store a grid of values approximating function



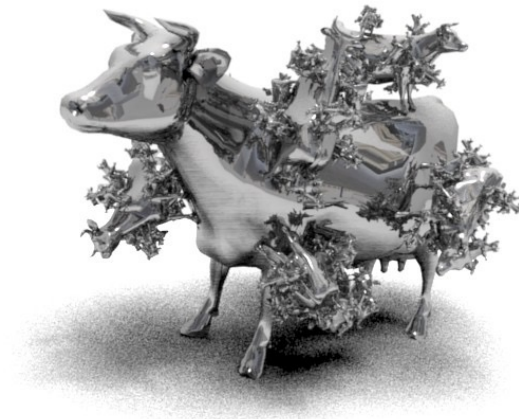
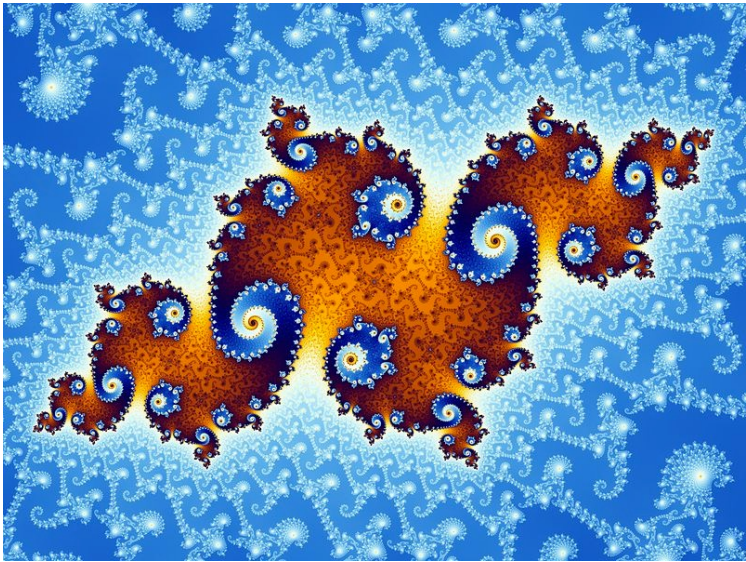
- Surface is found where interpolated values equal zero
- Provides much more explicit control over shape (like a texture)
- Unlike closed-form expressions, runs into problems of aliasing!

The aerodynamics of a cow:



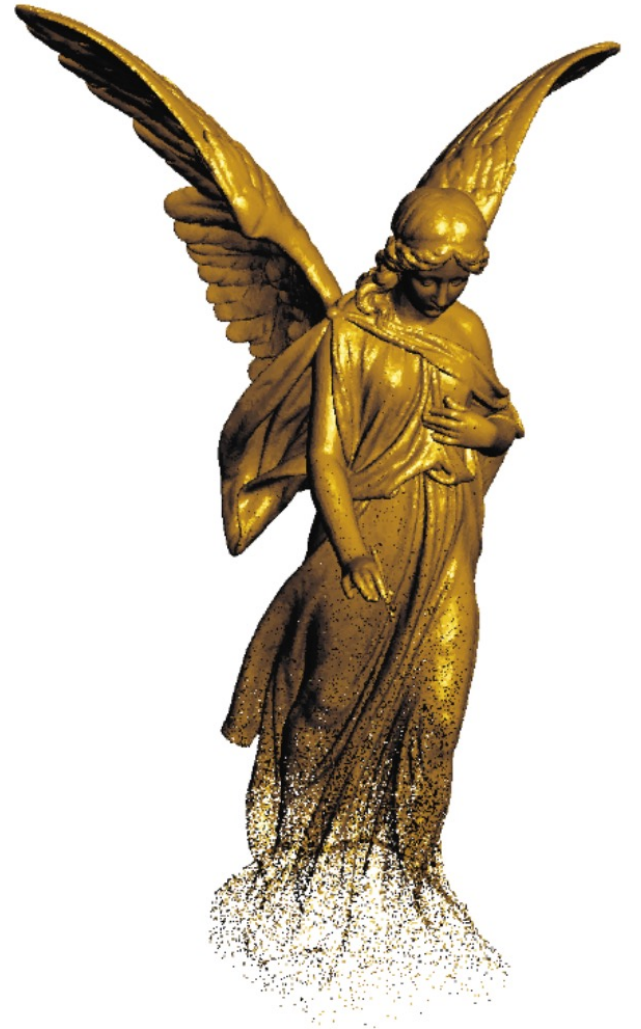
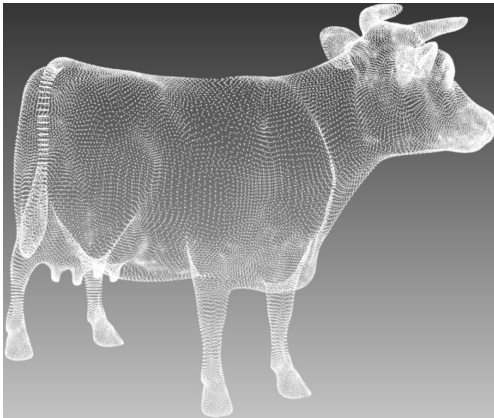
Fractals [Implicit]

- No precise definition; exhibit self-similarity, detail at all scales
- New “language” for describing natural phenomena
- Hard to control shape!



Point Cloud [Explicit]

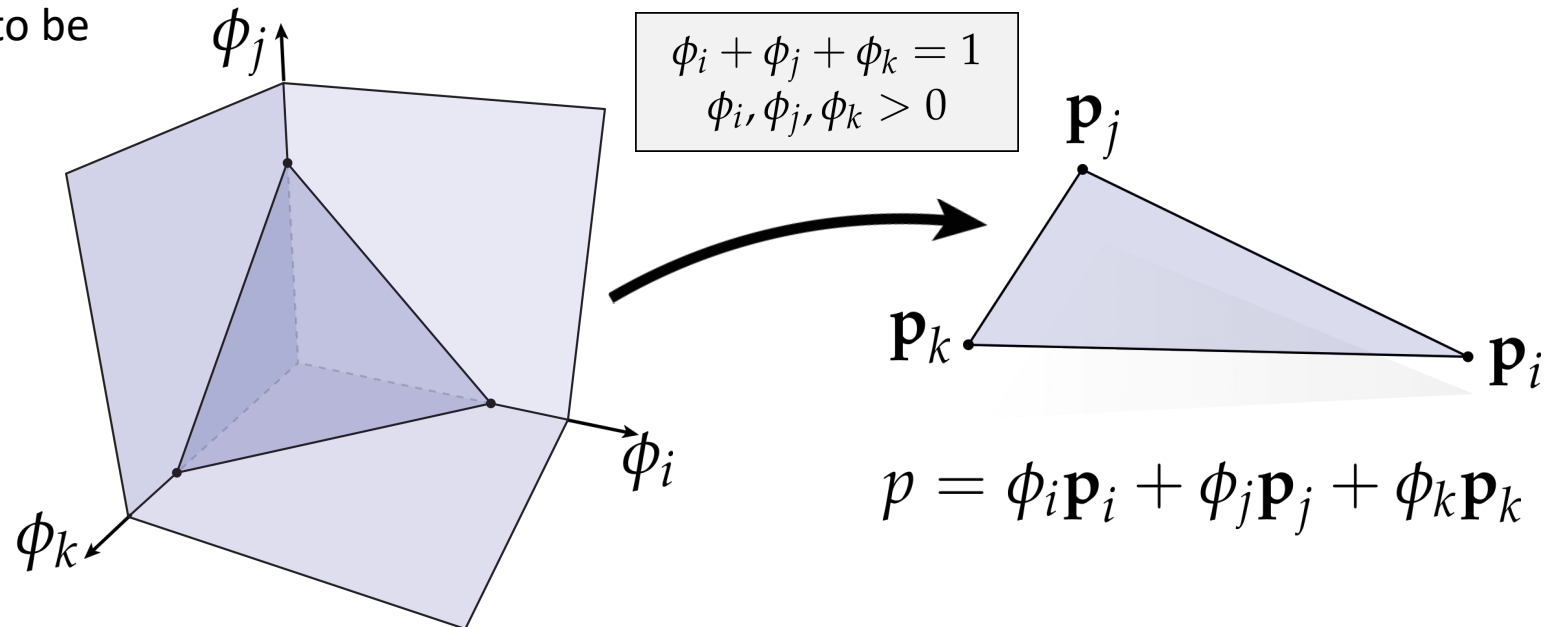
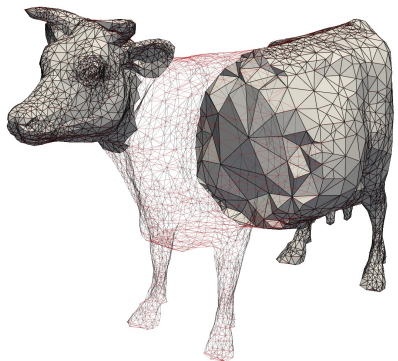
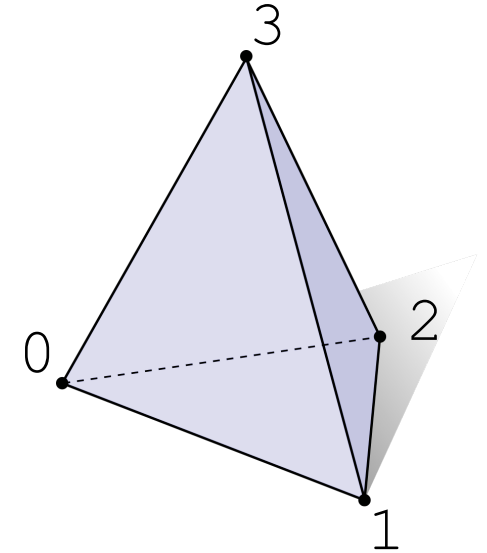
- Easiest representation: list of points (x, y, z)
 - Often augmented with normal
- Easily represent any kind of geometry
- Easy to draw dense cloud ($\gg 1$ point/pixel)
- Easy for simulation
- Large lookup time
- Large memory overhead
 - Hard to interpolate undersampled regions
 - Hard to do processing / simulation /
 - Result is just as good as the scan



Triangle Mesh [Explicit]

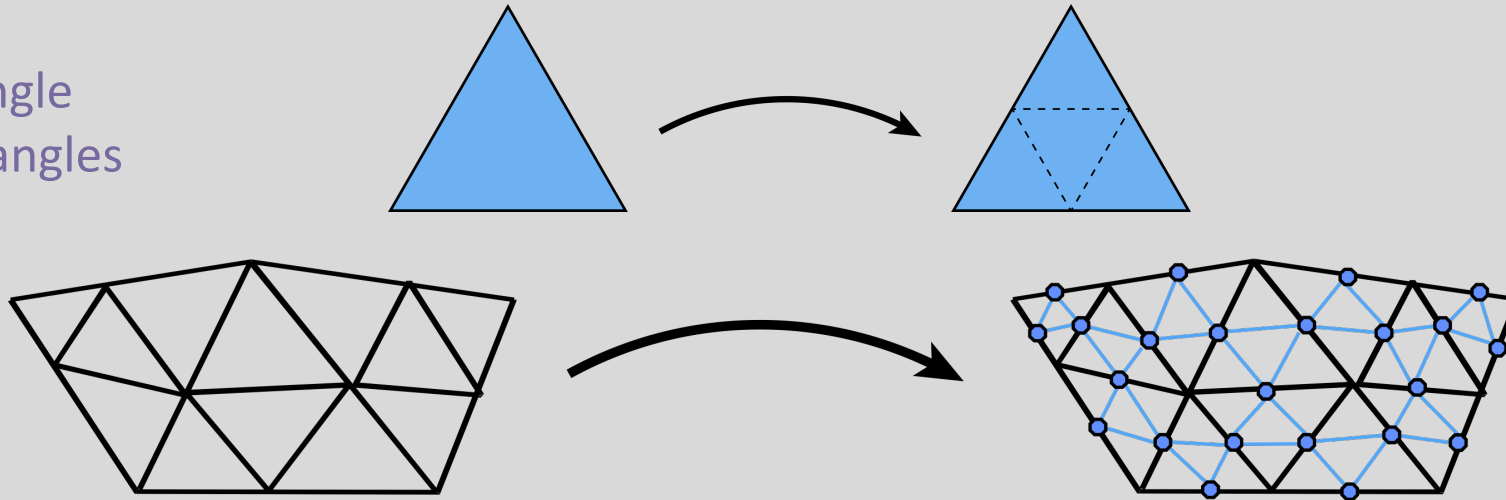
- Large memory overhead
 - Store vertices as triples of coordinates (x,y,z)
 - Store triangles as triples of indices (i,j,k)
- Easy interpolation with good approximation
 - Use barycentric interpolation to define points inside triangles
- Polygonal Mesh: shapes do not need to be triangles
 - Ex: quads

	[VERTICES]			[TRIANGLES]		
	x	y	z	i	j	k
0:	-1	-1	-1	0	2	1
1:	1	-1	1	0	3	2
2:	1	1	-1	3	0	1
3:	-1	1	1	3	1	2

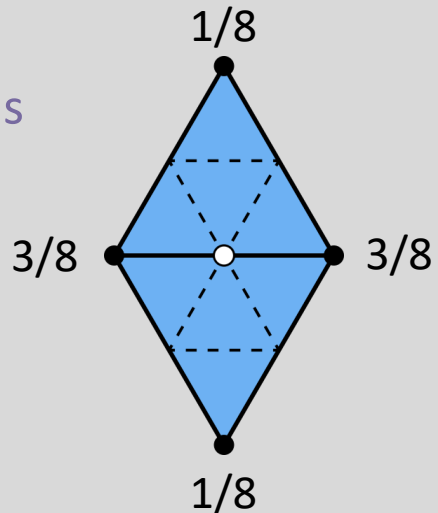


Loop Subdivision

Step 1:
Split triangle
into 4 triangles

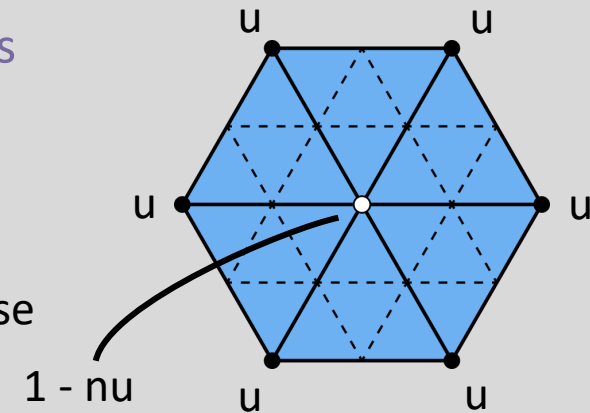


Step 2:
Assign new coords



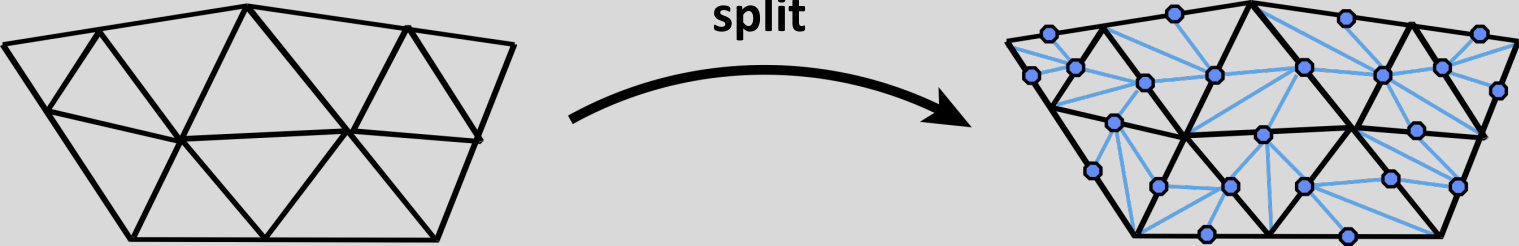
Step 3:
Assign old coords

n - vertex degree
 u - $3/16$ if $n=3$
 $3/(8n)$ otherwise

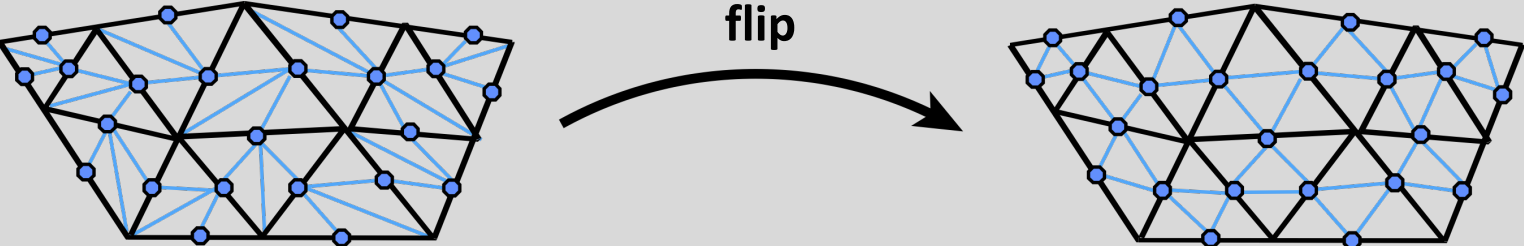


Loop Subdivision Using Local Ops

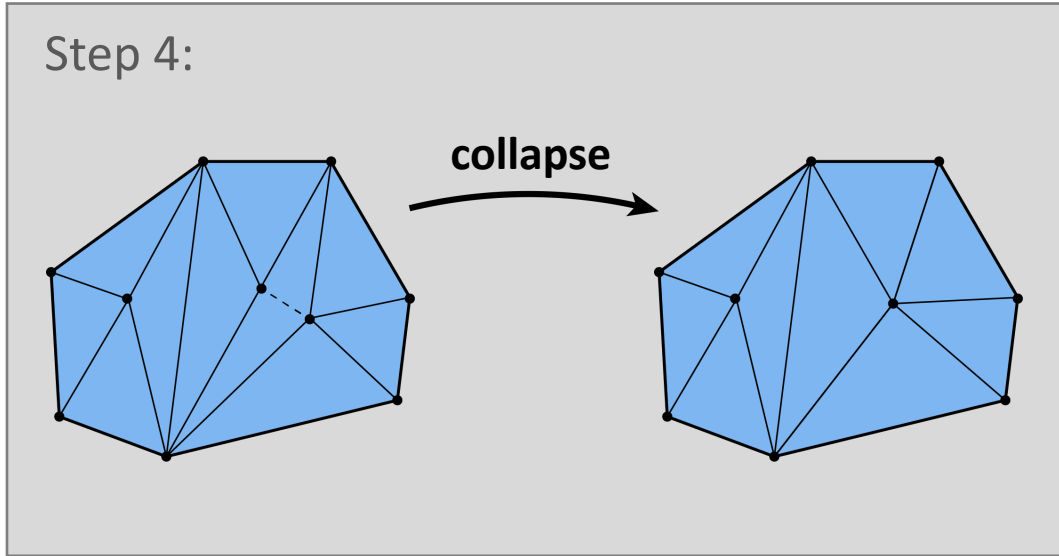
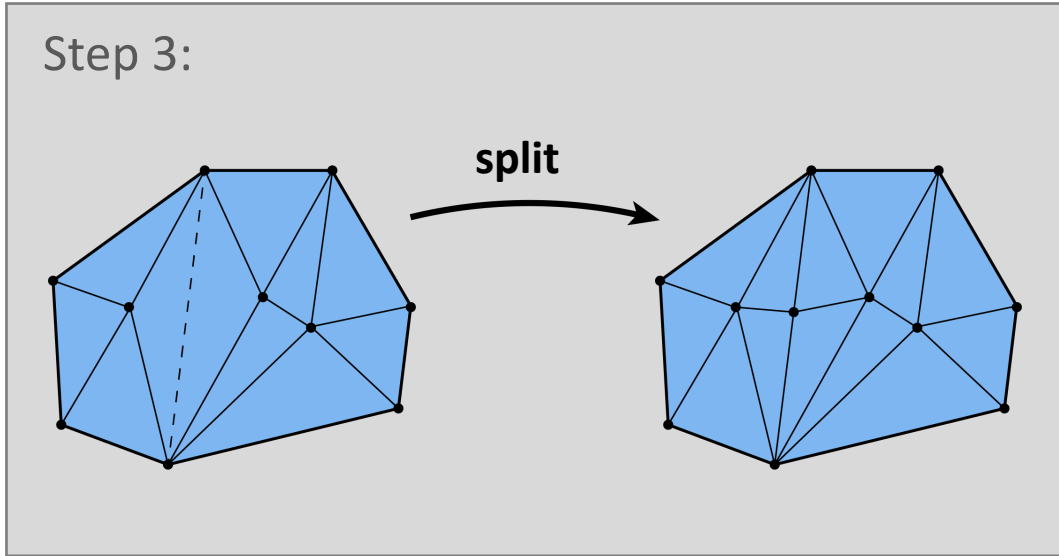
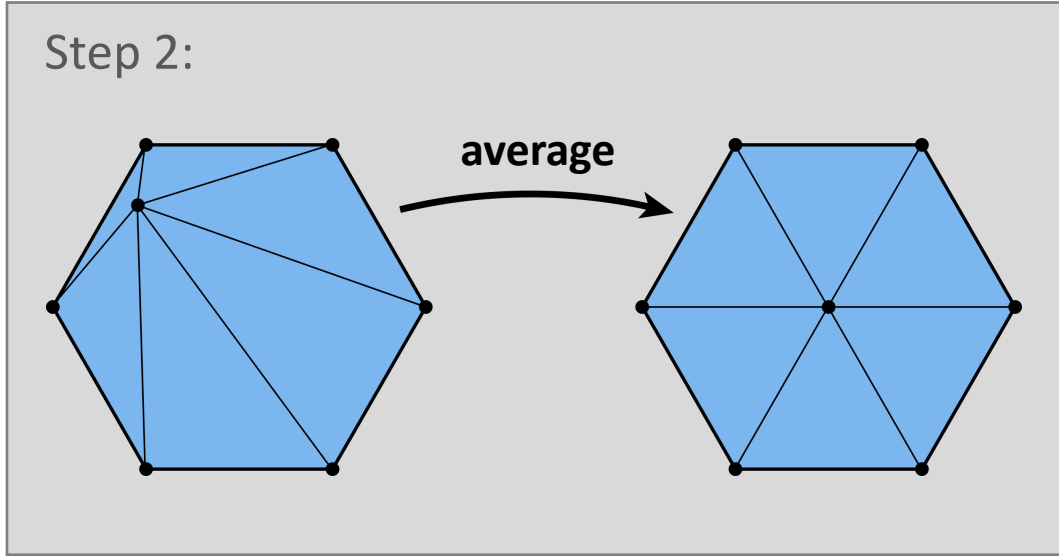
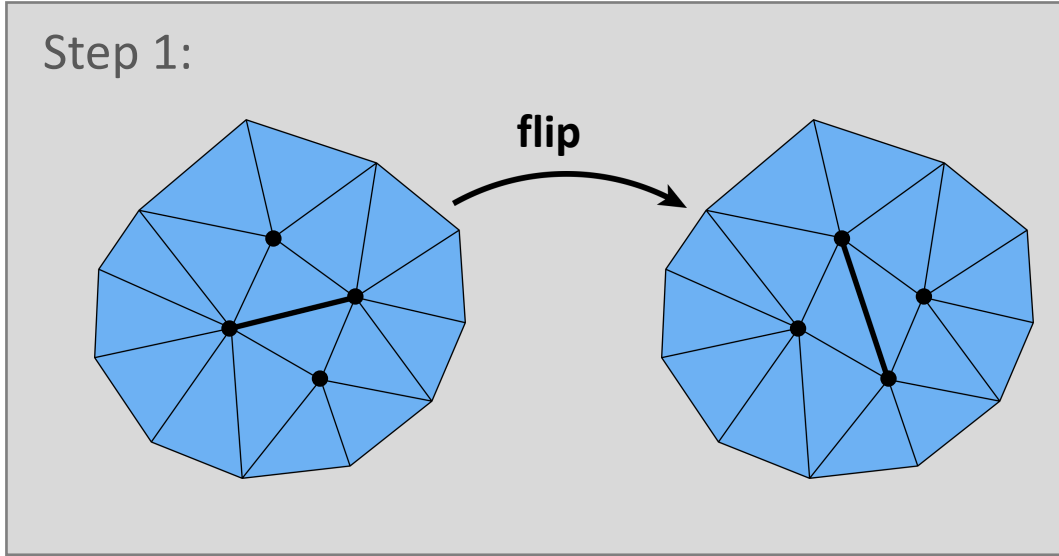
Step 1:
Split all edges in any order



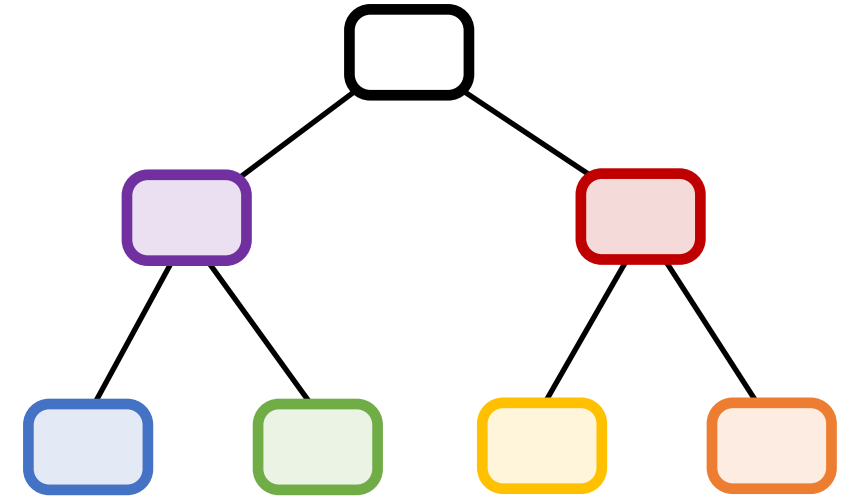
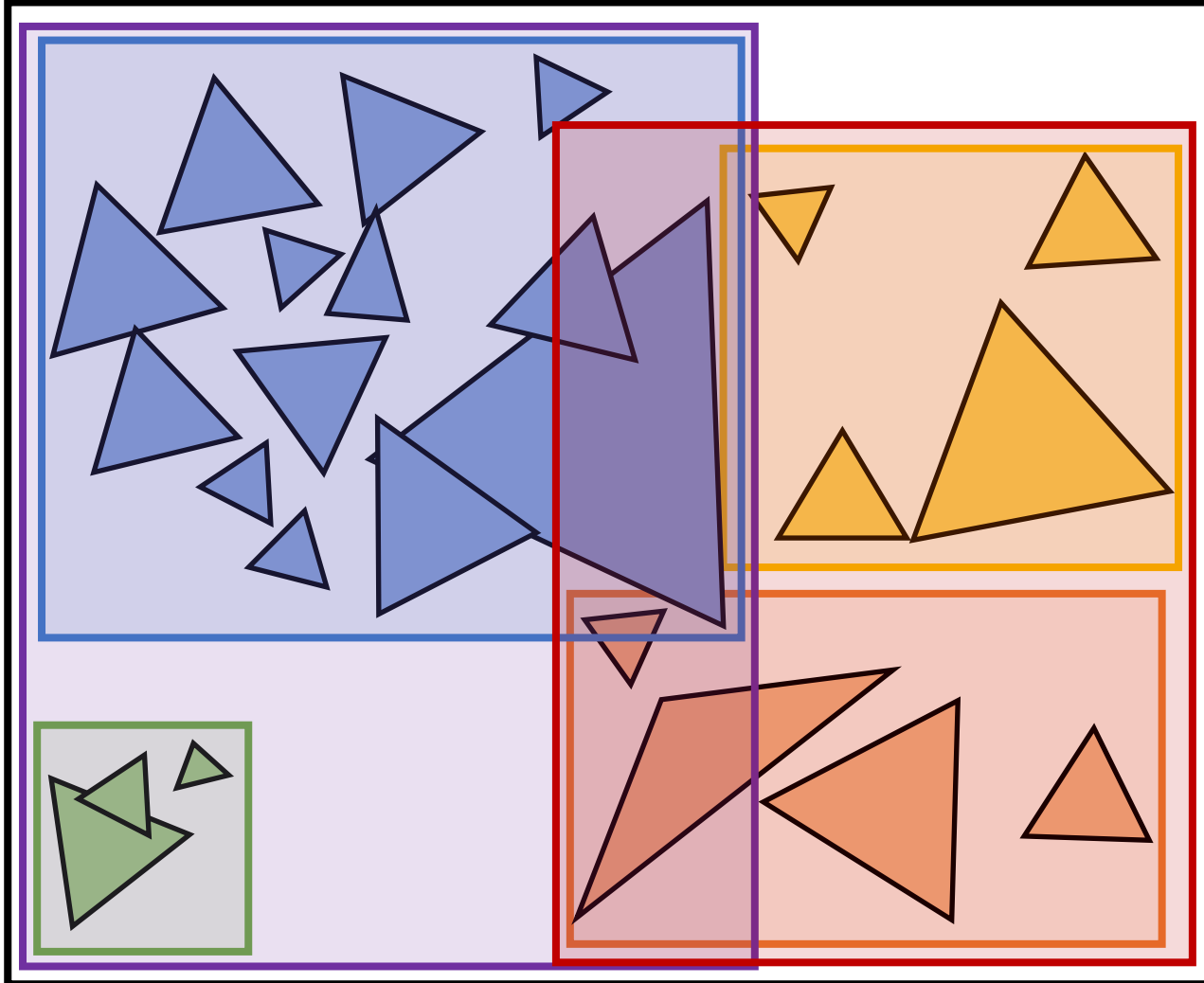
Step 2:
Flip new edges until they touch two new vertices



Isotropic Remeshing

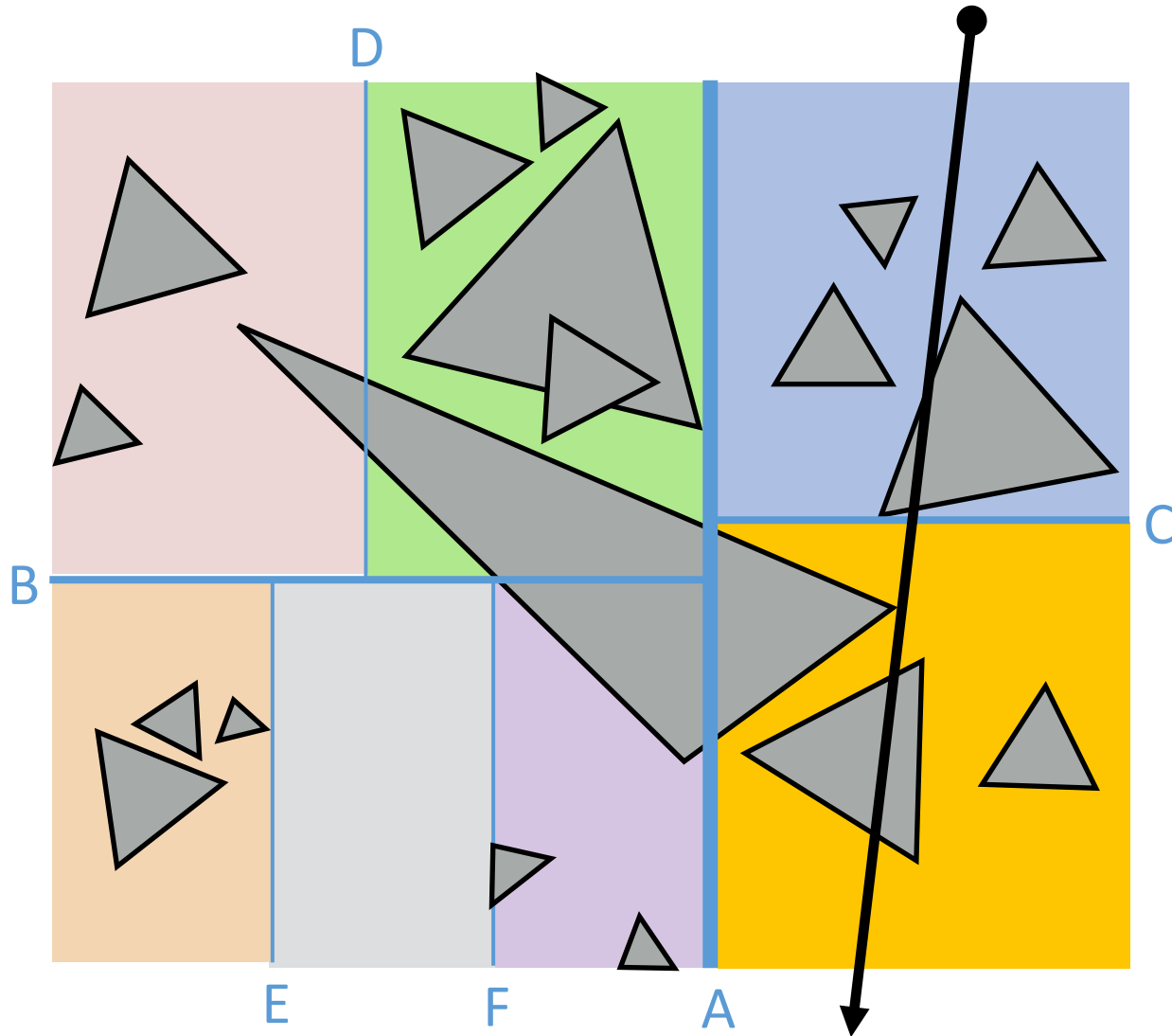


BVH Example

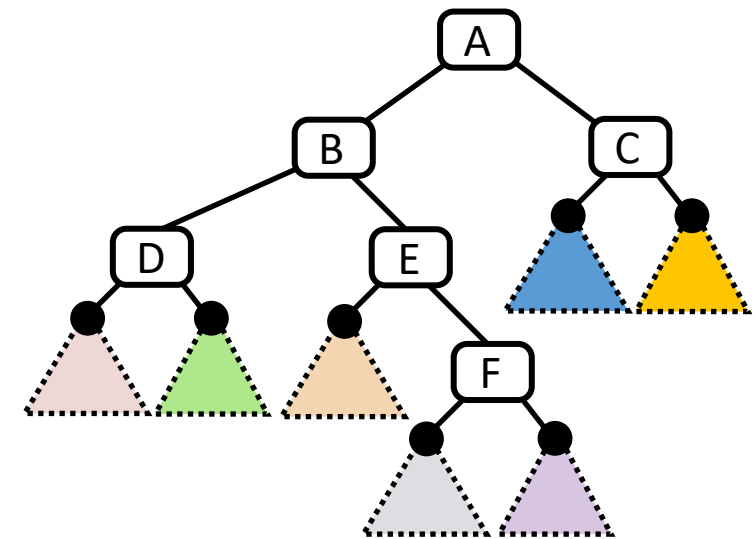


Bounding boxes will sometimes intersect!

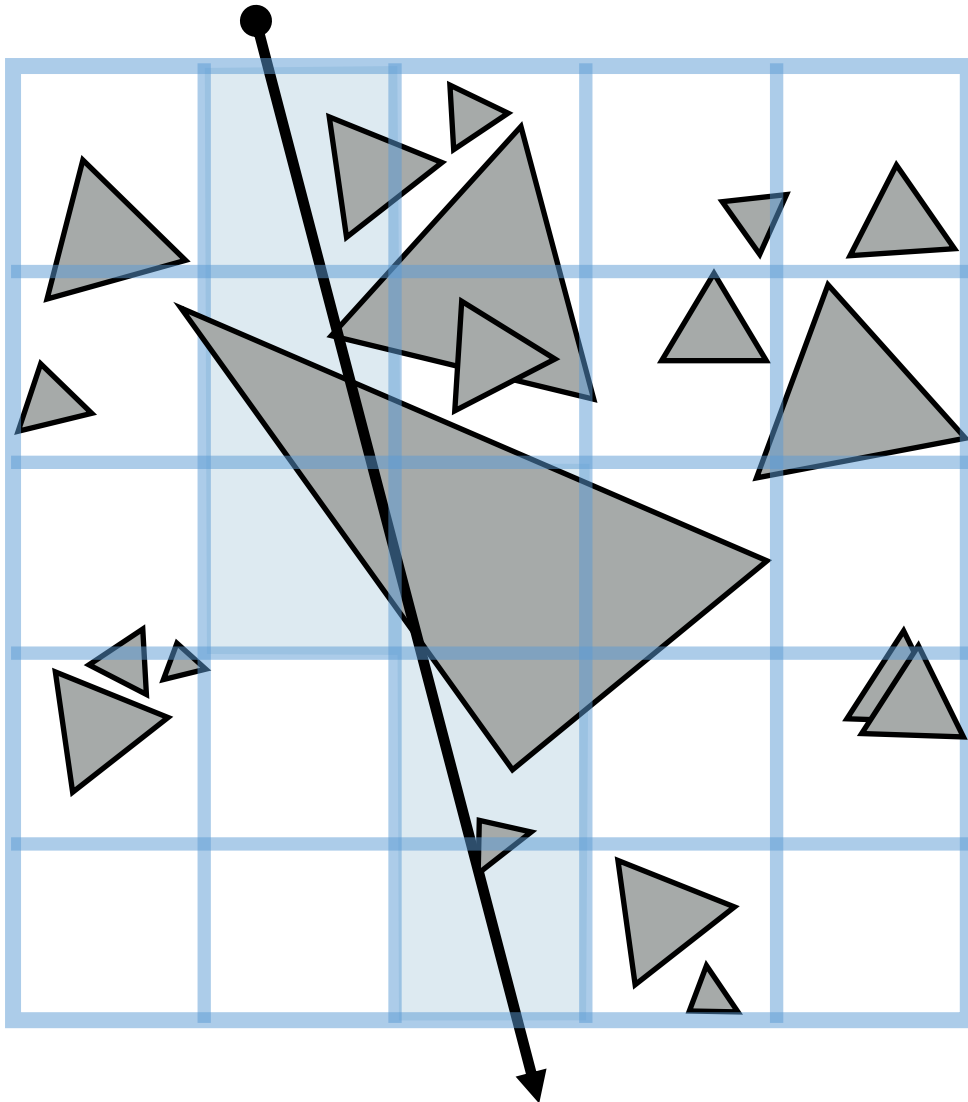
K-D Trees



- Recursively partition space via axis-aligned partitioning planes
 - Interior nodes correspond to spatial splits
 - Node traversal proceeds in front-to-back order
 - Unlike BVH, can terminate search after first hit is found
 - Still $O(\log(N))$ performance

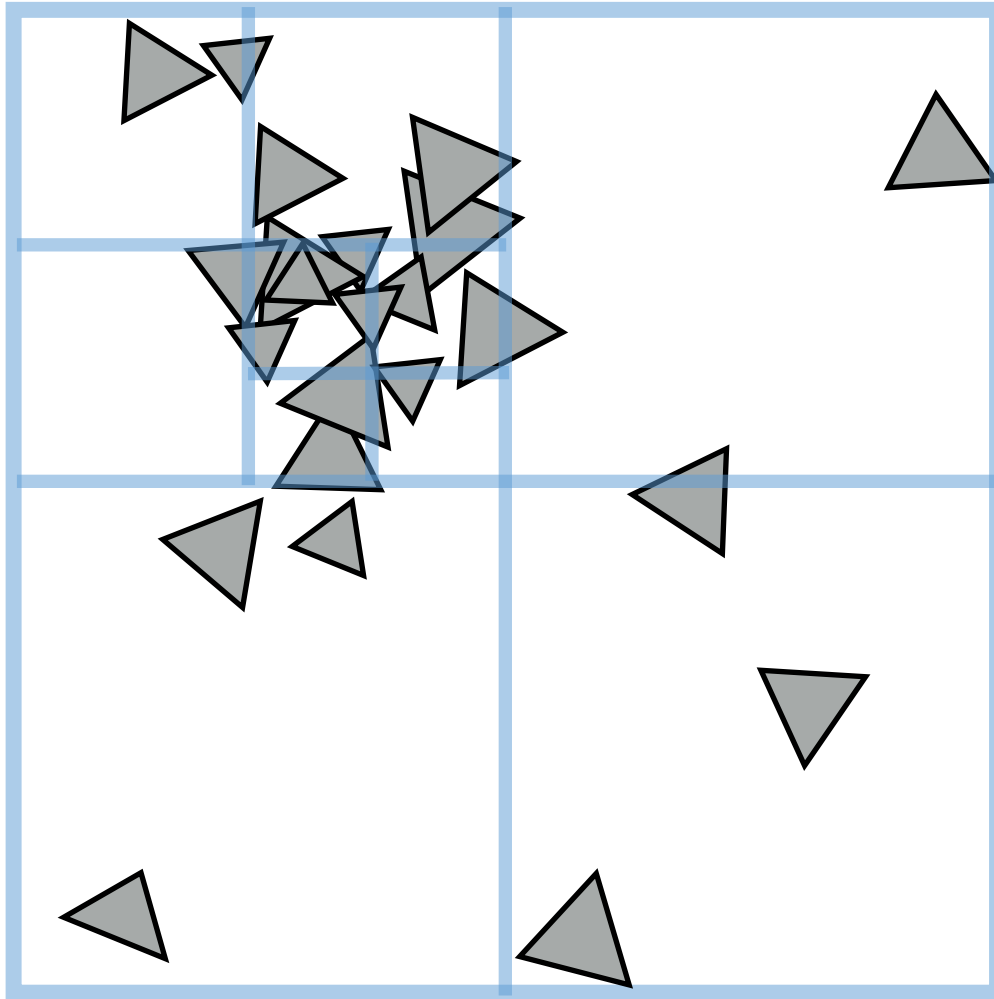


Uniform Grid



- Partition space into equal sized volumes (volume-elements or “voxels”)
- Each grid cell contains primitives that overlap voxel. (very cheap to construct acceleration structure)
- Walk ray through volume in order
 - Very efficient implementation possible (think: 3D line rasterization)
 - Only consider intersection with primitives in voxels the ray intersects
- What is a good number of voxels?
 - Should be proportional to total number of primitives N
 - Number of cells traversed is proportional to $O(\sqrt[3]{N})$
 - A line going through a cube is a cubed root
 - Still not as good as $O(\log(N))$

Quad-Tree/Octree



- Like uniform grid, easy to build
- Has greater ability to adapt to location of scene geometry than uniform grid
 - Still not as good adaptability as K-D tree
- **Quad-tree:** nodes have 4 children
 - Partitions 2D space
- **Octree:** nodes have 8 children
 - Partitions 3D space

- ~~A1: Rasterization~~

- ~~A2: Geometry~~

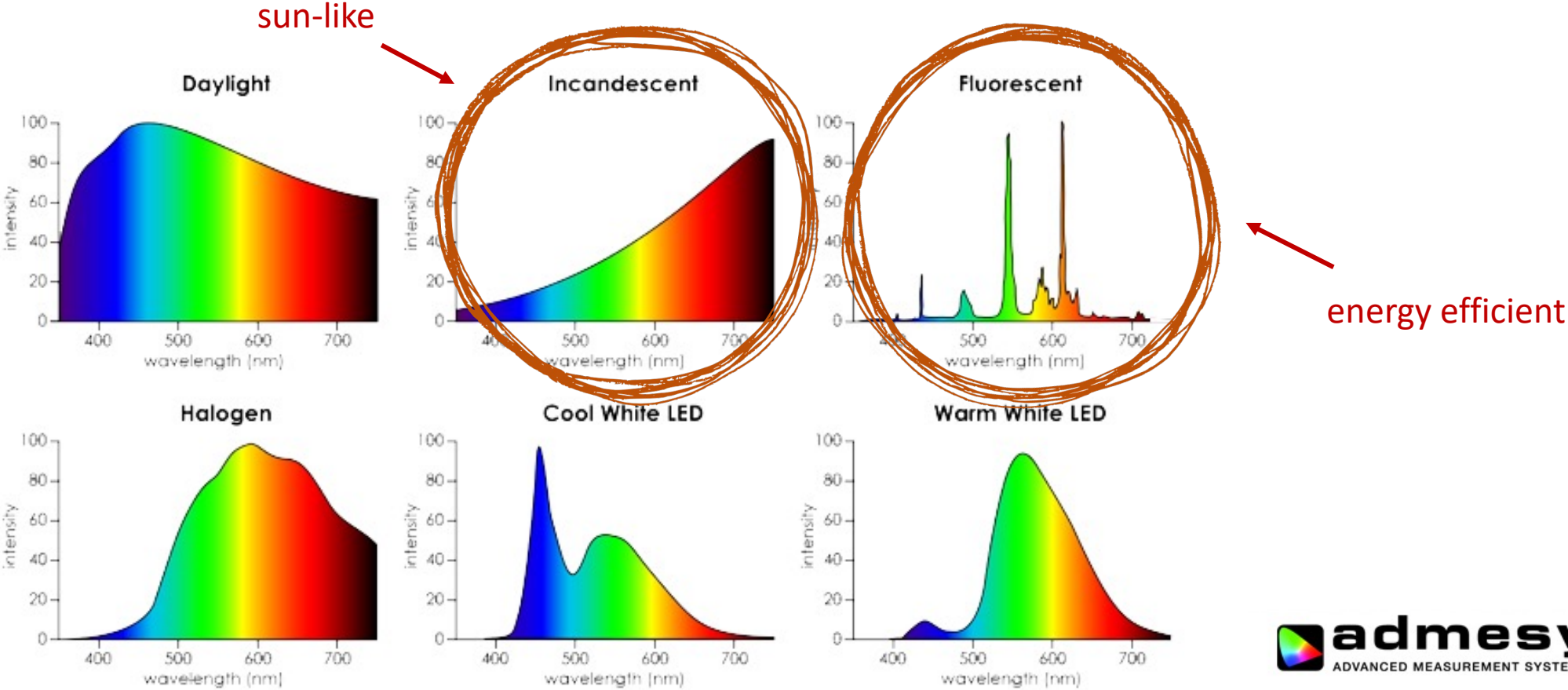
- A3: Rendering

- A4: Animation

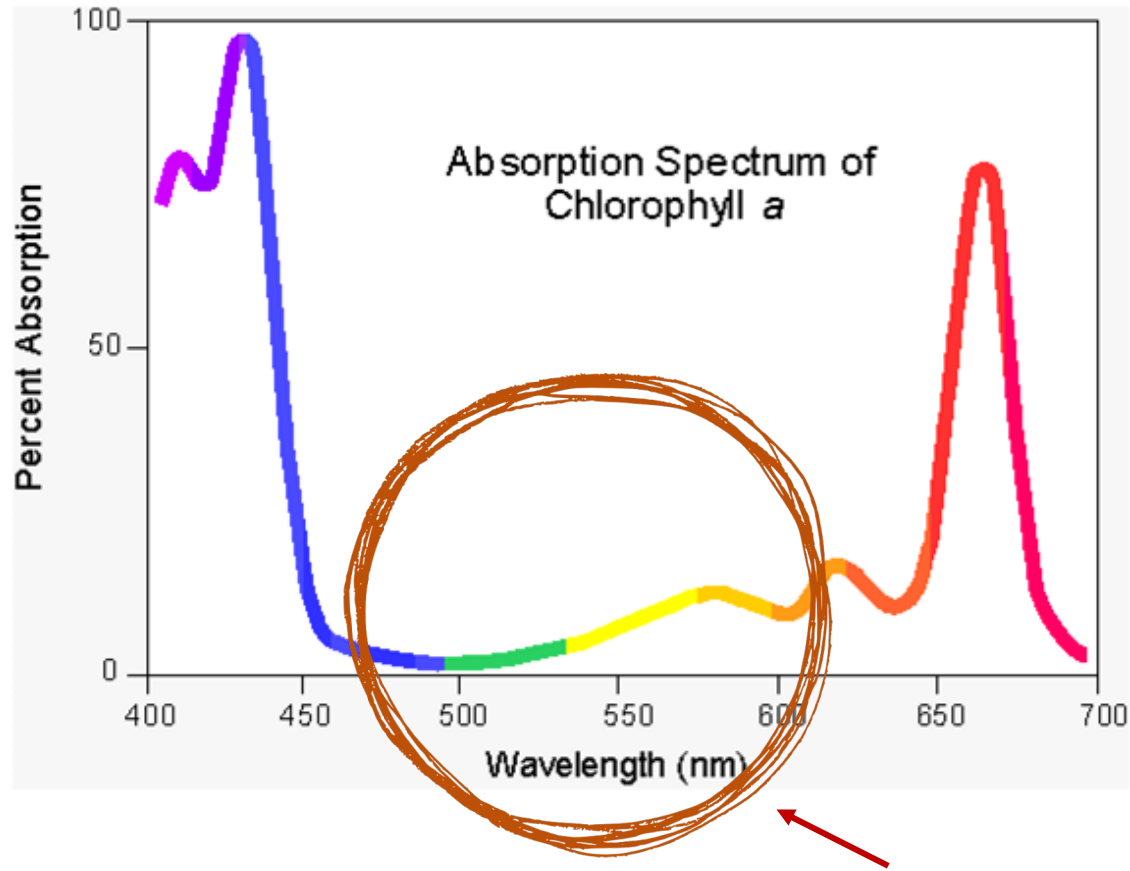
Color & Radiometry

- Absorption vs Emission
- Eyes vs Cameras
 - Pupil
 - Lens
 - Rods
 - Cones
- Radiance
 - Radiant Energy
 - Radiant Energy Density
 - Radiant Flux
 - Irradiance
- Lambert's Law

Emission Spectrum Examples



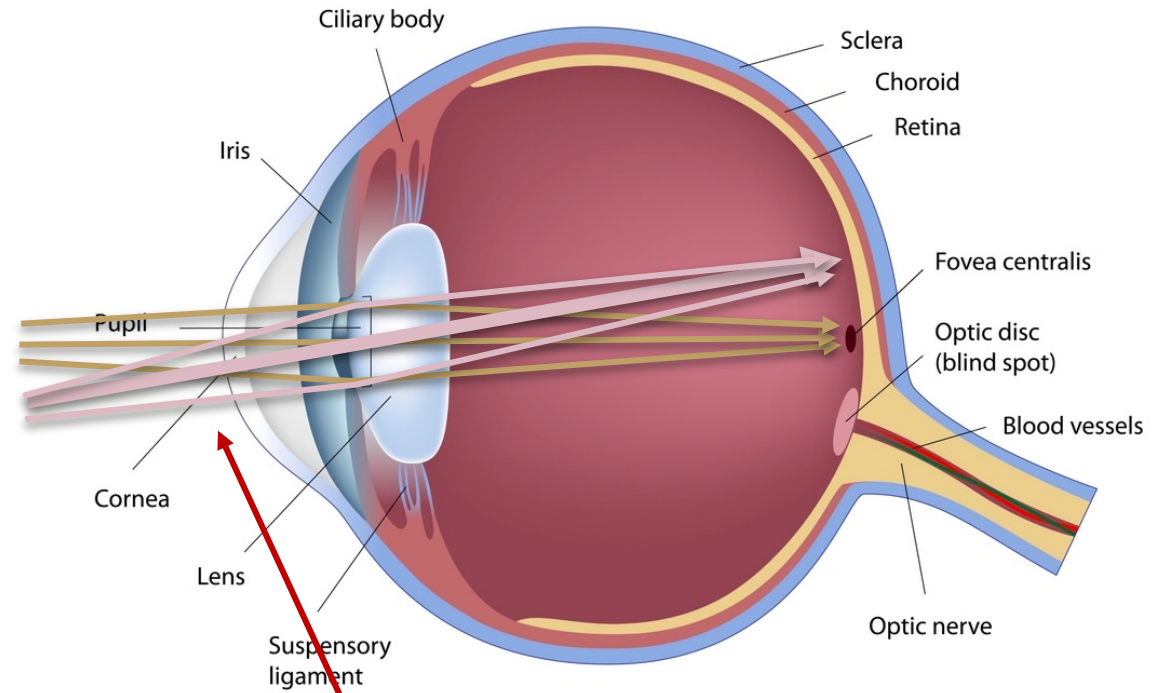
Absorption Spectrum Examples



plants are green because they do not absorb green light

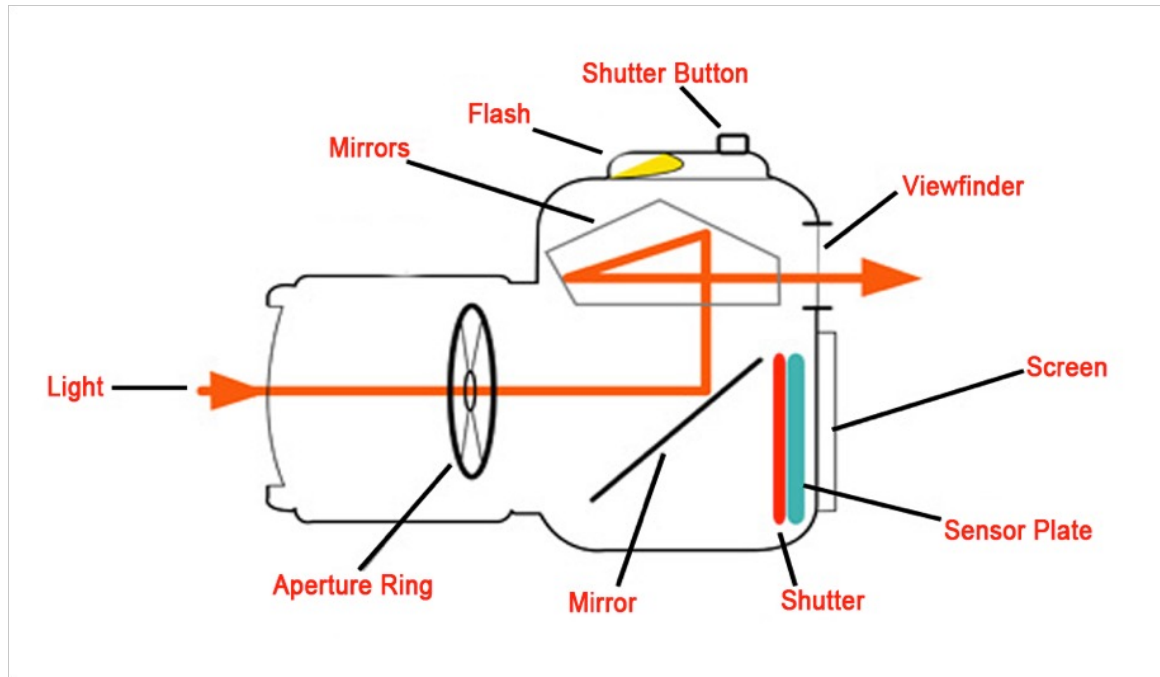
'Eye' See What You Mean

- Eyes are biological cameras
 - Light passes through the pupil [black dot in the eye]
 - Iris controls how much light enters eye [colored ring around pupil]
 - Eyes are sensitive to too much light
 - Iris protects the eyes
 - Lens behind the eye converges light rays to back of the eye
 - Ciliary muscles around the lens allow the lens to be bent to change focus on nearby/far objects
- 130+ million retina cells at the back of the eye
 - Cells pick up light and convert it to electrical signal
 - Electric signal passes through optic nerve to reach the brain



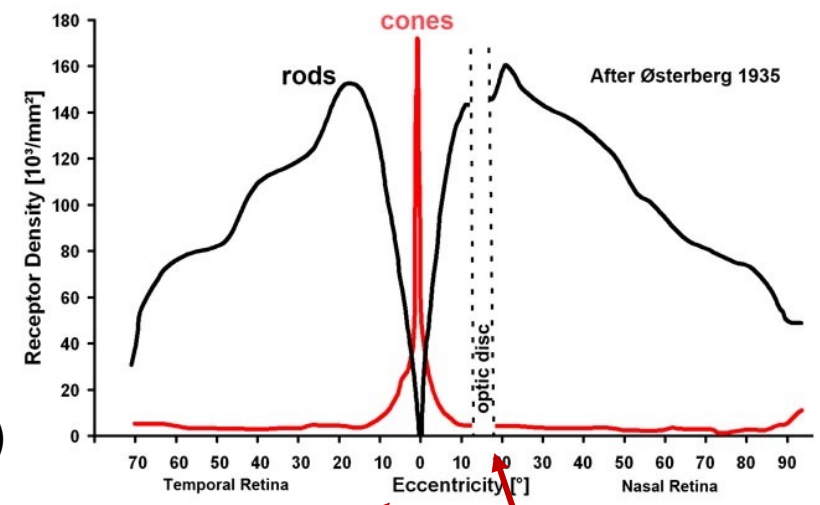
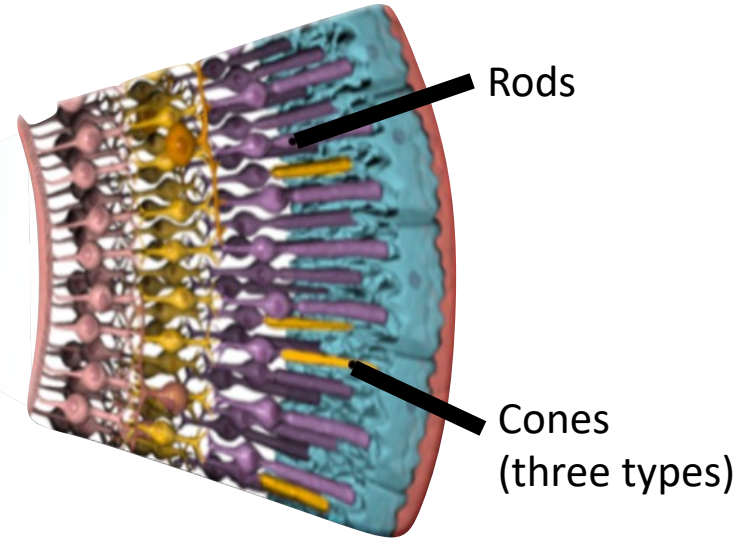
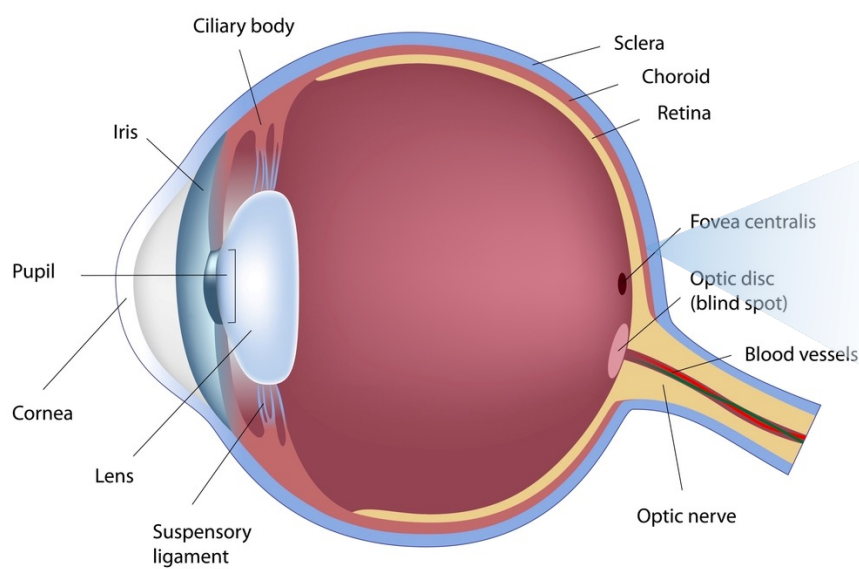
*Image appears backwards!
Don't worry, brain flips it right-side up*

The Biological Camera



- **Pupil** is the **camera opening**
 - Allows light through
- **Iris** is the **aperture ring**
 - Controls aperture
- **Lens** is the...well, **lens**
 - Can change focus
- **Retina** is the **sensor**
 - Converts light into electrical signal
- **Brain** is the **CPU**
 - Performs additional compute to correct raw image signal

Rods & Cones



- Cones are primary receptors near fovea used under high-light viewing conditions
 - Approx. 6-7 million cones in the human eye
 - Capture color
- Rods are primary receptors far from fovea used under low-light viewing conditions
 - Approx. 120 million rods in human eye
 - Capture intensity

Best vision at center of cones!

Human blind spot

Spectral Response of Cones

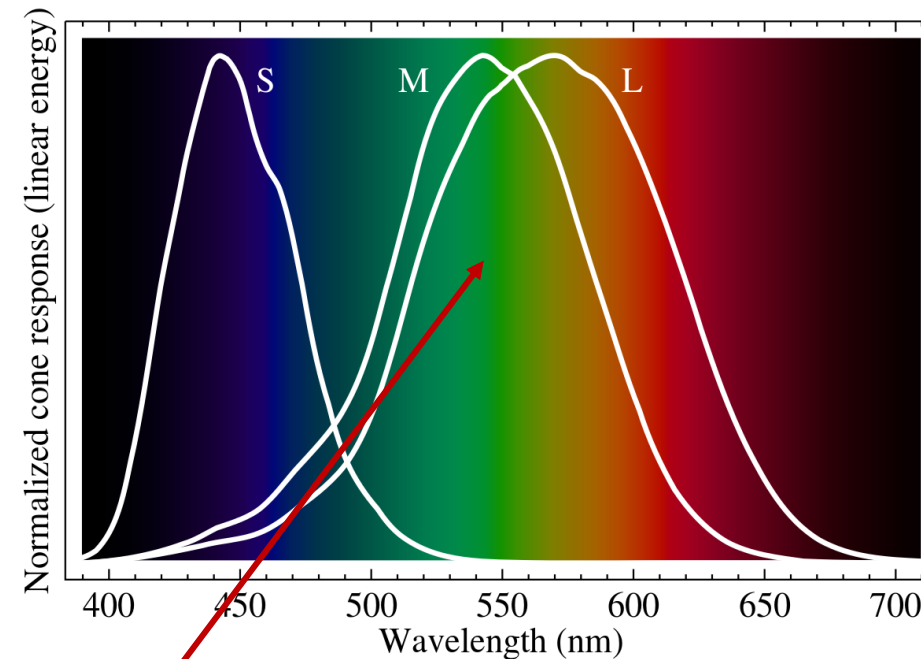
- Long, Medium, and Small cones pick up Long, Medium, and Small wavelengths respectively
- Each cone picks up a range of colors given by their response functions
 - Not much different than absorption spectrum
- Each cone integrates the emission & response to produce a single signal to transmit to the brain

$$S = \int_{\lambda} \Phi(\lambda) S(\lambda) d\lambda$$

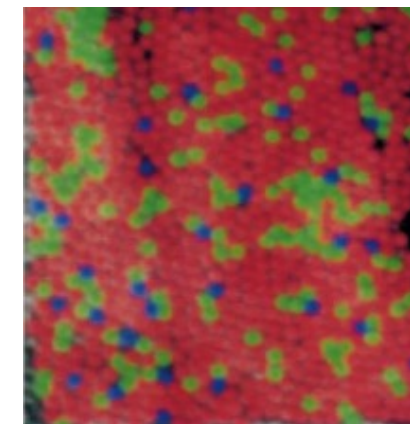
$$M = \int_{\lambda} \Phi(\lambda) M(\lambda) d\lambda$$

$$L = \int_{\lambda} \Phi(\lambda) L(\lambda) d\lambda$$

- Uneven distribution of cone types in eye
 - ~64% L cones, ~ 32% M cones ~4% S cones



A lot of green picked up!



Radiant Recap

Radiant Energy
(total number of hits)
Joules (J)

Radiant Energy Density
(hits per unit area)
Joules per sq meter (J/m^2)

Radiant Flux
(total hits per second)
Watts (W)

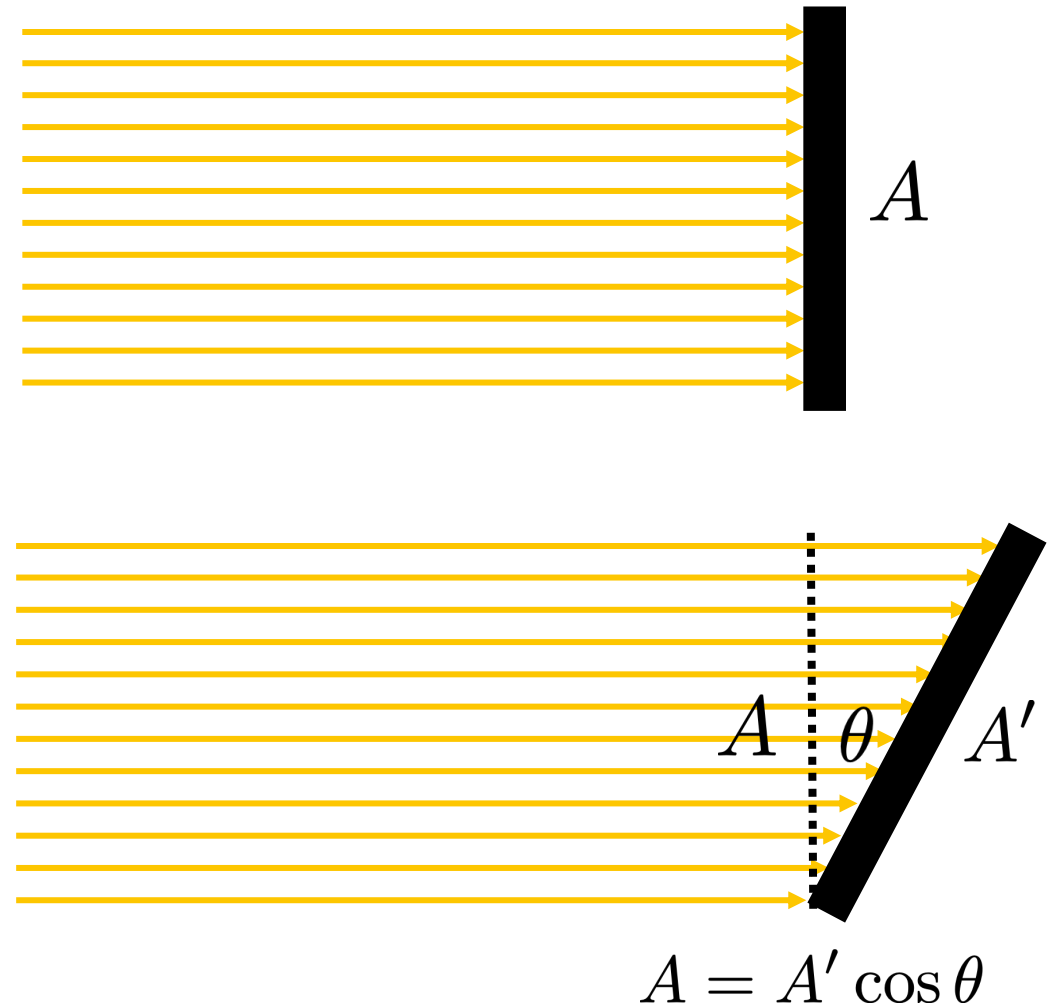
Radiant Flux Density
a.k.a. Irradiance
(hits per second per unit area)
Watts per sq meter (W/m^2)

Lambert's Law

- Irradiance (E) at surface is proportional to the flux (Φ) and the cosine of angle (θ) between light direction and surface normal:

$$E = \frac{\Phi}{A'} = \frac{\Phi \cos \theta}{A}$$

- Consider rotating a plane away from light rays
 - Plane will darken until it is perpendicular to light rays, then it will be completely black



The Rendering Equation

- The Rendering Equation
- Rendering Methods
 - Forwards Path-Tracing
 - Backwards Path-Tracing
 - Bi-Directional Path-Tracing
 - Metropolis Light Transport
- Variance Reduction
 - Sampling Rate
 - Ray Depth
- BRDFs
 - Lambertian
 - Mirror
 - Glass

The Rendering Equation

$$L_o(\mathbf{p}, \omega_o) = L_e(\mathbf{p}, \omega_o) + \int_{\mathcal{H}^2} f_r(\mathbf{p}, \omega_i \rightarrow \omega_o) L_i(\mathbf{p}, \omega_i) \cos \theta \, d\omega_i$$

$L_o(\mathbf{p}, \omega_o)$ outgoing radiance at point \mathbf{p} in outgoing direction ω_o

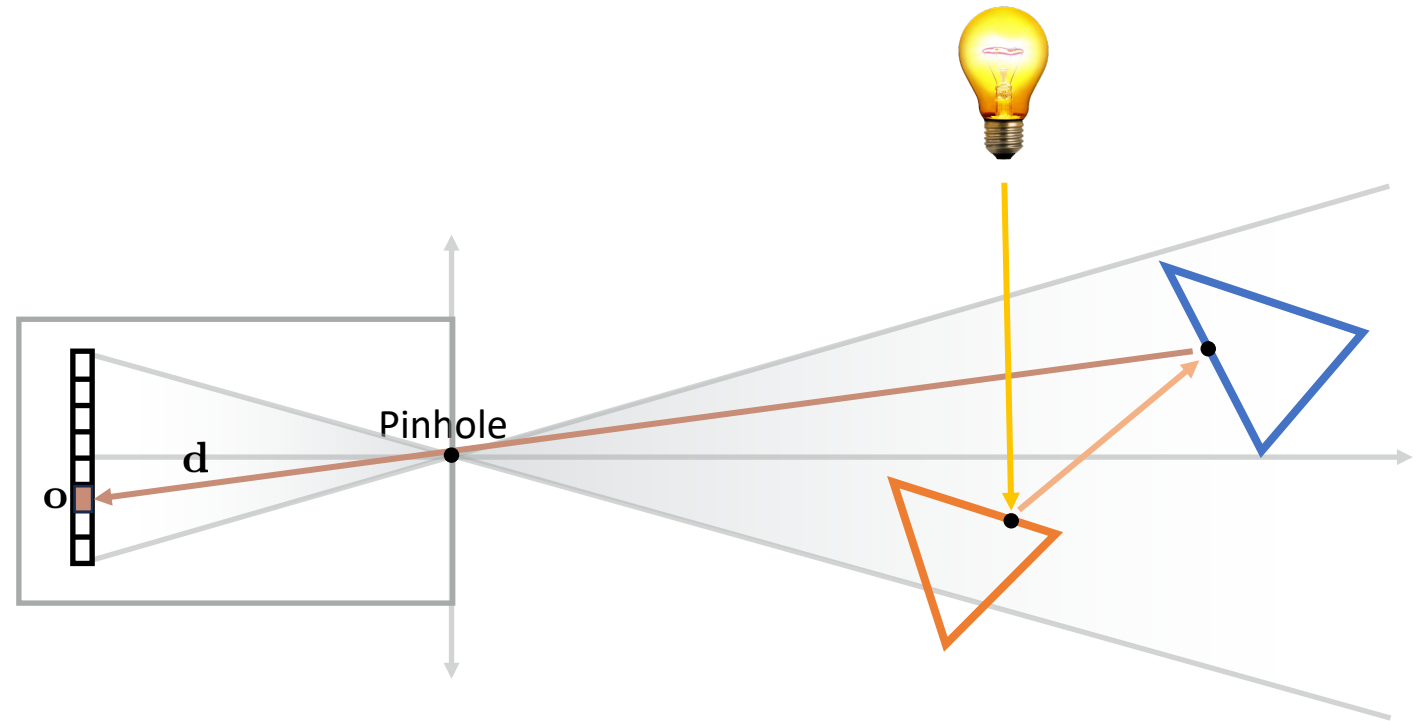
$L_e(\mathbf{p}, \omega_o)$ emitted radiance at point \mathbf{p} in outgoing direction ω_o

$f_r(\mathbf{p}, \omega_i \rightarrow \omega_o)$ scattering function at point \mathbf{p} from incoming direction ω_i to outgoing direction ω_o

$L_i(\mathbf{p}, \omega_i)$ incoming radiance to point \mathbf{p} from direction ω_i

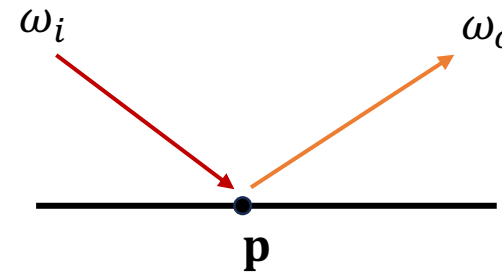
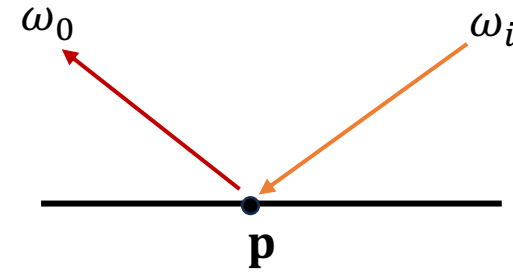
Example Of A Simple Renderer

- Yellow light ray generated from light source
- Ray hits orange specular surface
 - Emits a ray in reflected direction
 - Mixes yellow and orange color
- Ray hits blue specular surface
 - Emits a ray in reflected direction
 - Mixes blue and yellow and orange
- Ray passes through pinhole camera
 - Light recorded on photoelectric cell
 - Incident pixel will be brown in final image



Hemholtz Reciprocity




- Reversing the order of incoming and outgoing light does not affect the BRDF evaluation
 - $f_r(\mathbf{p}, \omega_i \rightarrow \omega_o) = f_r(\mathbf{p}, \omega_o \rightarrow \omega_i)$
- Critical to reverse pathtracing algorithms
 - Allows us to trace rays backwards and still get the same BRDF affect

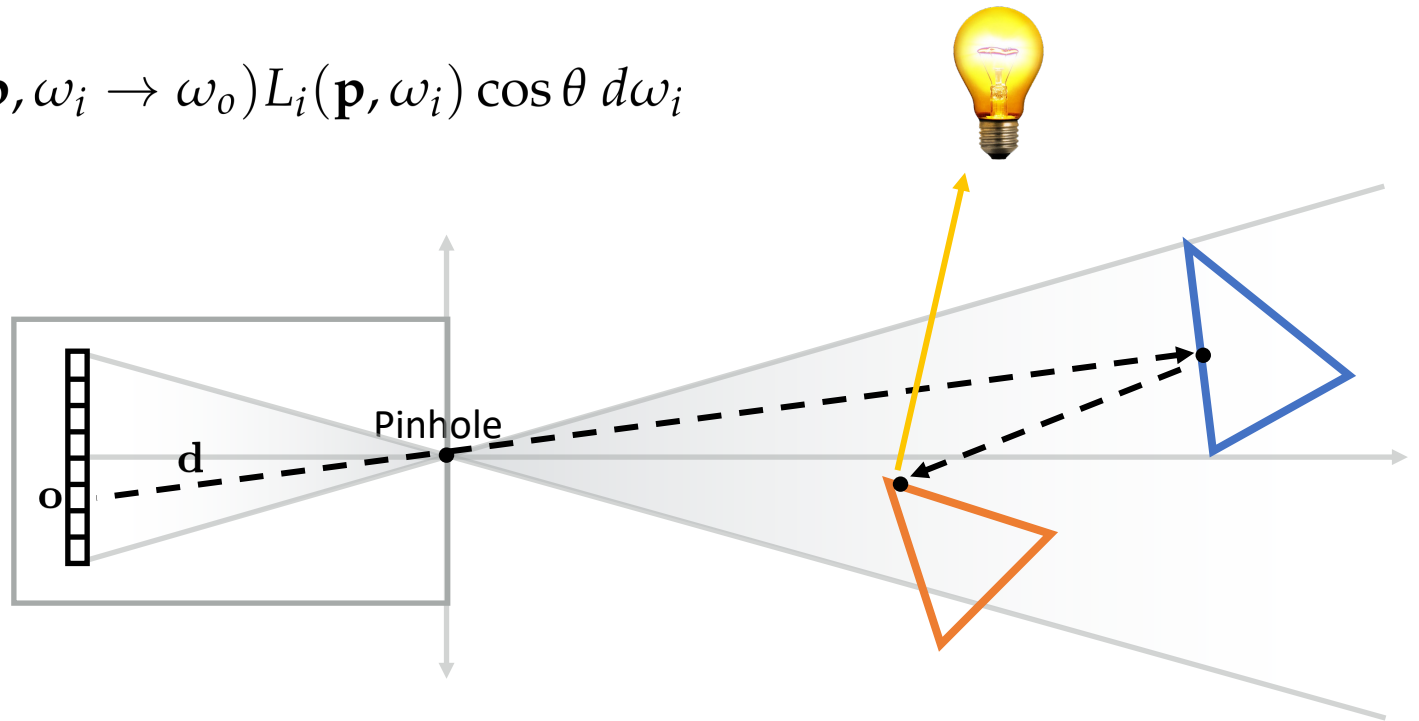


Example Of A Simple Backwards Renderer

[ray depth 2]

$$L_o(\mathbf{p}, \omega_o) = L_e(\mathbf{p}, \omega_o) + \int_{\mathcal{H}^2} f_r(\mathbf{p}, \omega_i \rightarrow \omega_o) L_i(\mathbf{p}, \omega_i) \cos \theta d\omega_i$$

- Intersect  , no emission
- Intersect  , no emission
- Ray terminate, emission 

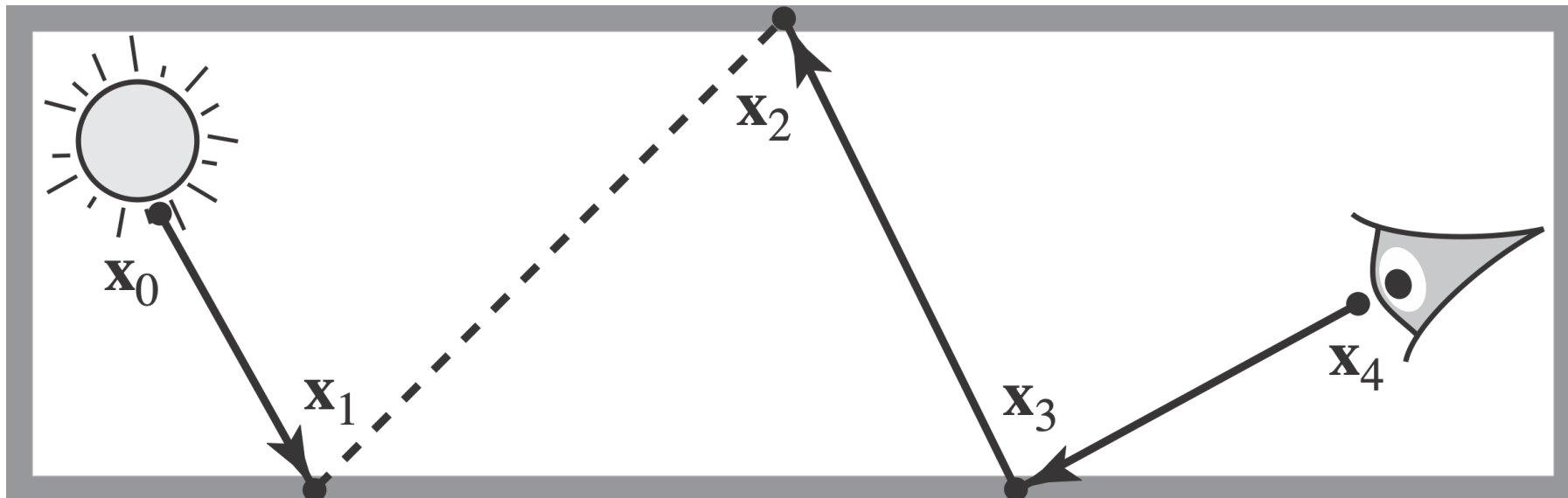


$$L(\text{pixel}) = L_e(\text{ray}_1) + f_r(\text{obj}_1)[L_e(\text{ray}_2) + f_r(\text{obj}_2)[L_e(\text{ray}_3)]]$$

$$L(\text{pixel}) = \square + f_r(\triangle)[\square + f_r(\triangle)[\text{yellow square}]]$$

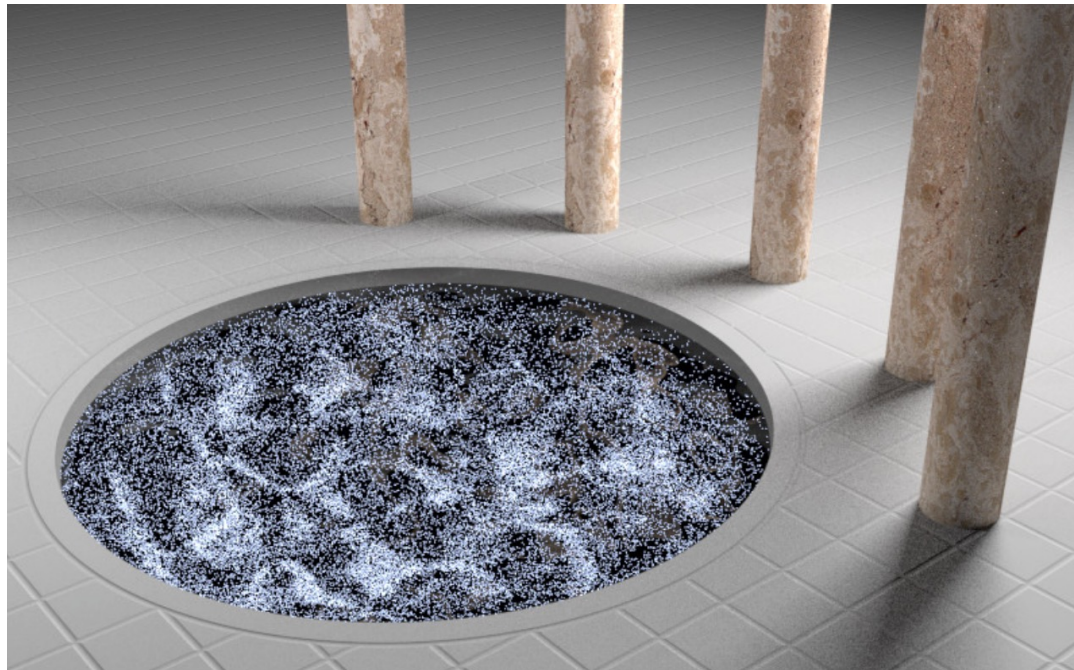
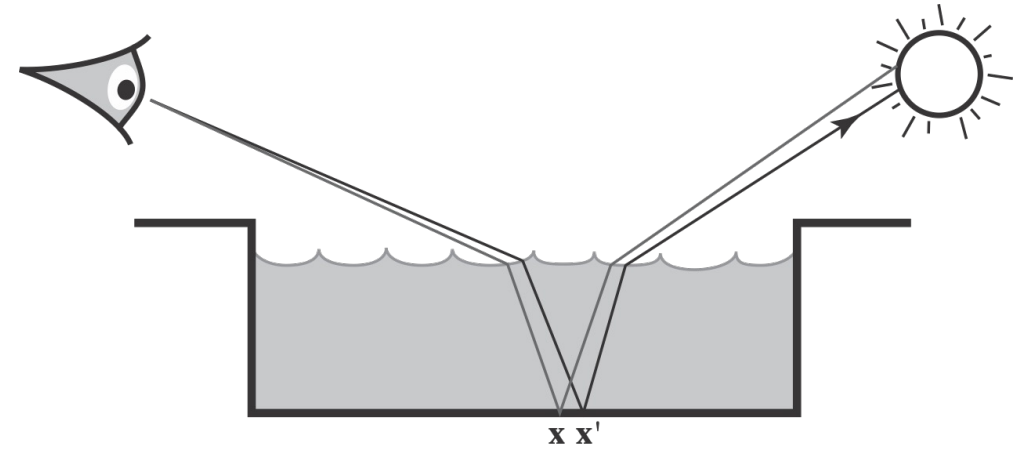
Bidirectional Path Tracing

- If path tracing is so great, why not do it **twice**?
 - Main idea of bidirectional!
- Trace a ray from the camera into the scene
- Trace a ray from the light into the scene
 - Connect the rays at the end
- Unbiased algorithm
 - No longer trying to connect rays through non-volume sources
- Can set different lengths per ray
 - Example: Forward $m = 2$, Backward $m = 1$

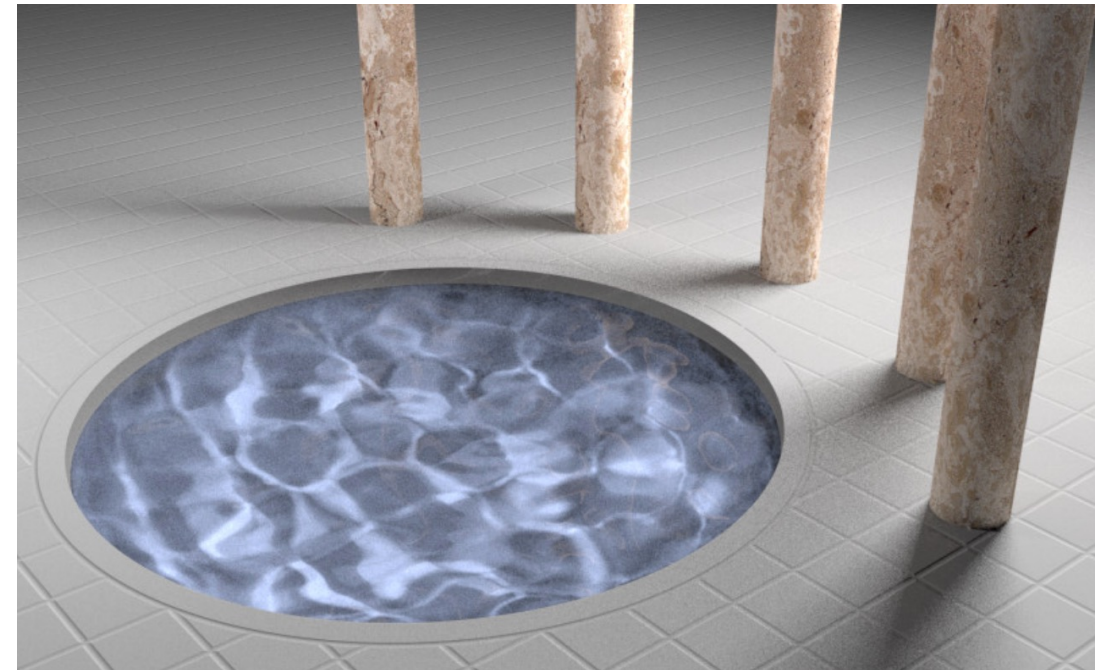


Metropolis Light Transport

- **Similar idea:** mutate good paths
- Water causes paths to refract a lot
 - Small mutations allows renderer to find contributions faster
- Path Tracing and MLT rendered in the same time



[Path Tracing]



[Metropolis Light Transport]

Number Of Ray Samples

- **Number of Rays**
 - How many rays we trace into the scene
 - Measured as samples (rays) per pixel [spp]
- Increasing the number of rays increases the quality of the image
 - Anti-aliasing
 - Reduces black spots from terminating emission occlusion



[1 spp]



[16 spp]

Number Of Ray Bounces

- **Number of Ray Bounces**
 - How many times a ray bounces before it terminates
 - Measured as ray bounce or depth
- Increasing the number of ray bounces increases the quality of the image
 - Better color blending around images
 - More details reflected in specular images



[2 depth]



[8 depth]

Lambertian Material

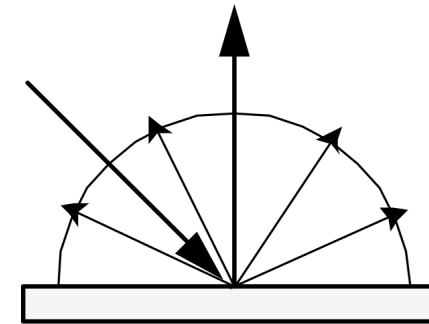
- Also known as diffuse
- Light is equally likely to be reflected in each output direction
 - BRDF is a constant, relying on albedo (ρ)

$$f_r = \frac{\rho}{\pi}$$

- BRDF can be pulled out of the integral

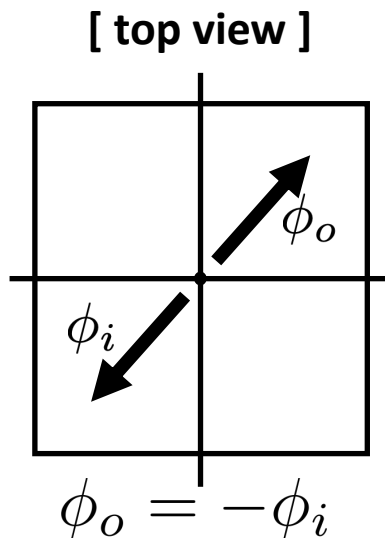
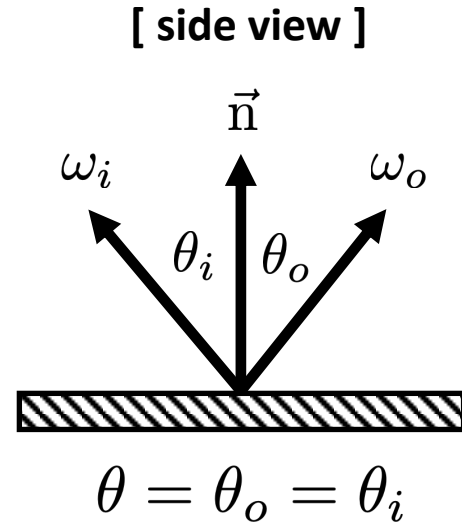
$$\begin{aligned} L_o(\omega_o) &= \int_{H^2} f_r L_i(\omega_i) \cos \theta_i d\omega_i \\ &= f_r \int_{H^2} L_i(\omega_i) \cos \theta_i d\omega_i \\ &= f_r E \end{aligned}$$

- Easy! Pick any outgoing ray w_o



Minions (2015) Illumination Entertainment

Reflective Material



- Reflectance equation described as:

$$\omega_o = -\omega_i + 2(\omega_i \cdot \vec{n})\vec{n}$$

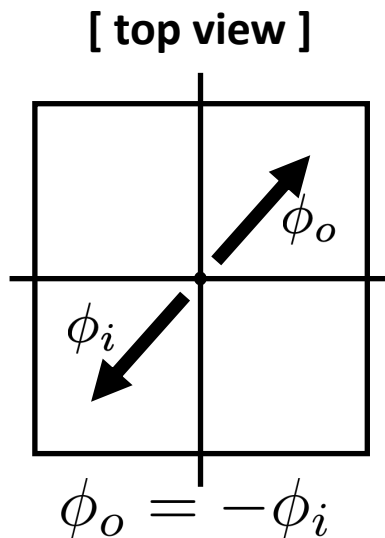
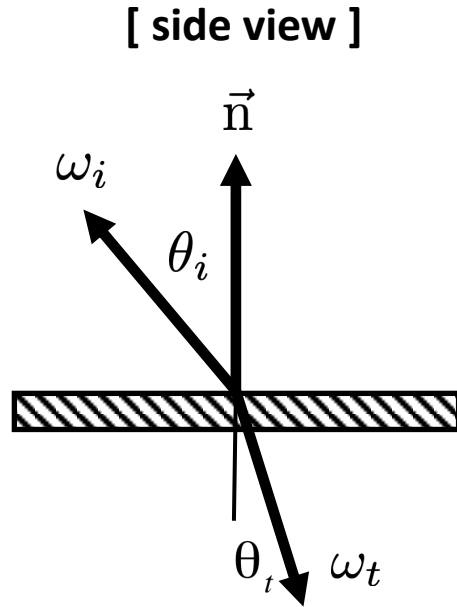
- Why is the ray ω_i pointing away from the surface?
 - Just syntax. Incoming and outgoing rays share same origin point \mathbf{p}

- BRDF represented by dirac delta (δ) function:

$$f_r(\theta_i, \phi_i; \theta_o, \phi_o) = \frac{\delta(\cos \theta_i - \cos \theta_o)}{\cos \theta_i} \delta(\phi_i - \phi_o \pm \pi)$$

- 1 when ray is perfect reflection, 0 everywhere else
- All radiance gets reflected, nothing absorbed
- In practice, no hope of finding reflected direction via random sampling
 - Simply pick the reflected direction!

Refractive Material



- Refractive equation described as:

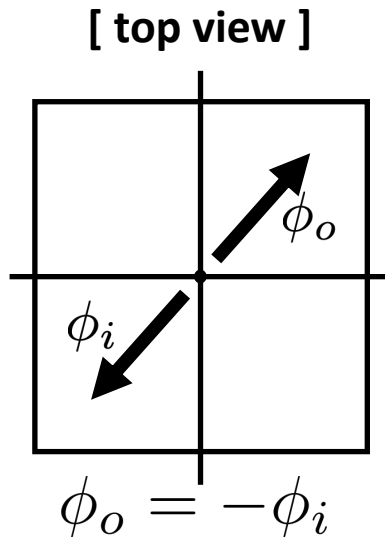
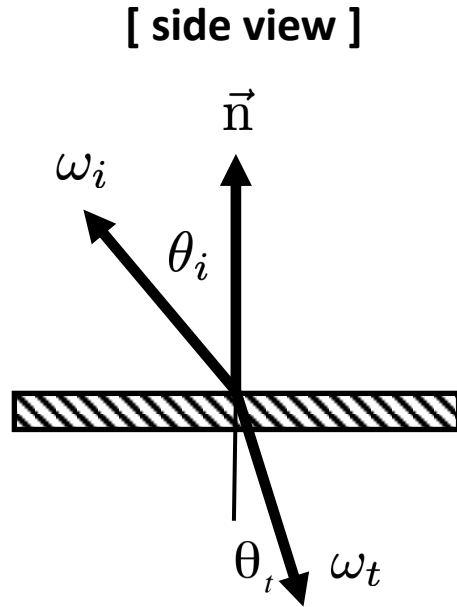
$$\eta_i \sin \theta_i = \eta_t \sin \theta_t$$

- Also known as Snell's Law
- η_i and η_t describe the index of refraction of the incoming and outgoing mediums
 - Example: η_i is air, η_t is water

Medium	η
Vacuum	1.0
Air (sea level)	1.00029
Water (20°C)	1.333
Glass	1.5-1.6
Diamond	2.42

- η is the ratio of the speed of light in a vacuum to that in a second medium of greater density
 - The larger the η , the denser the material

Refractive Material



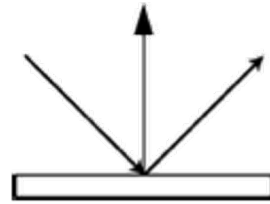
- Refractive equation described as:

$$\eta_i \sin \theta_i = \eta_t \sin \theta_t$$

- Also known as Snell's Law
- Can rewrite the equation as:

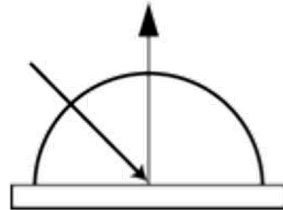
$$\begin{aligned} \cos \theta_t &= \sqrt{1 - \sin^2 \theta_t} \\ &= \sqrt{1 - \left(\frac{\eta_i}{\eta_t}\right)^2 \sin^2 \theta_i} \\ &= \sqrt{1 - \left(\frac{\eta_i}{\eta_t}\right)^2 (1 - \cos^2 \theta_i)} \end{aligned}$$

Types of Reflectance Functions



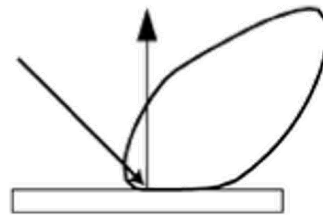
Ideal Specular

- Perfect mirror



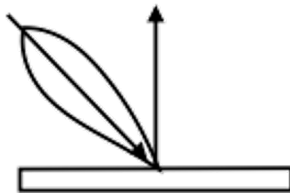
Ideal Diffuse

- Uniform in all directions



Glossy Specular

- Majority of light in reflected direction



Retroreflective

- Reflects light back towards source

- ~~A1: Rasterization~~

- ~~A2: Geometry~~

- ~~A3: Rendering~~

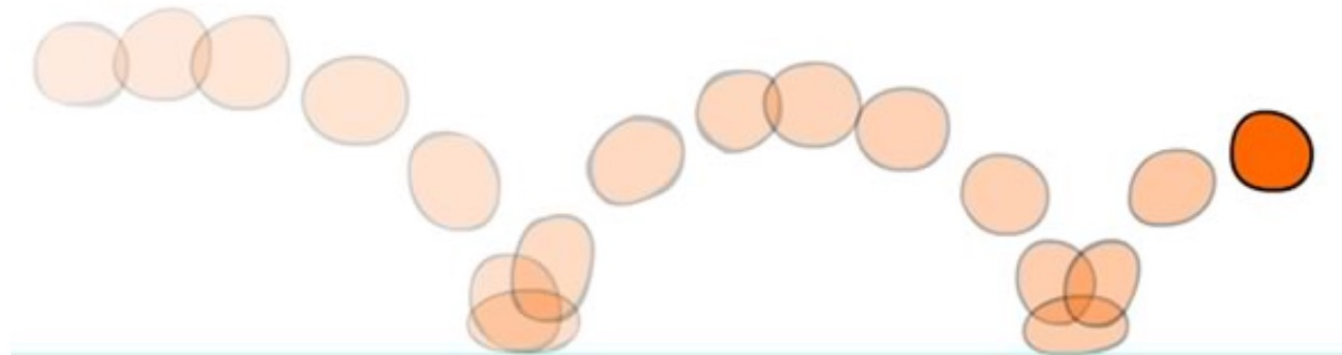
- A4: Animation

Principles Of Animation

- 12 Principles
 - Easing
 - Arcs
 - Timing
- Motion Graphs
 - Displacement
 - Velocity
 - Acceleration
- Splines
 - Natural Splines
 - Hermite/Bezier Curves
 - B-Splines

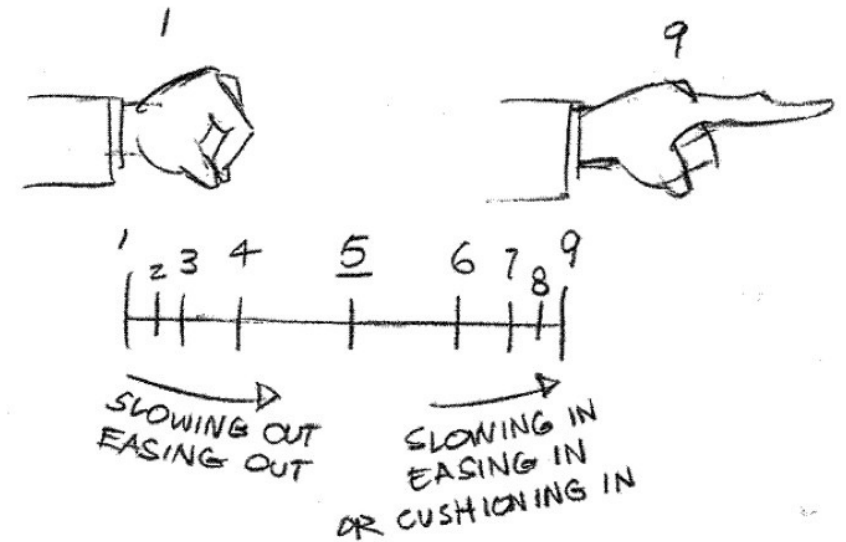
Onion Skinning

- **Onion-Skinning** is a tool that lets you see **previous** and **future frames** at a lower opacity
 - Helps when you have two keyframes and want to add an in-between frame
 - Based off translucency of cel paper
- Can also help visualize the **spatial trajectory** and **motion** of objects
 - Good debugging tool to make sure trajectories are arc like and maintain proportions



Easing

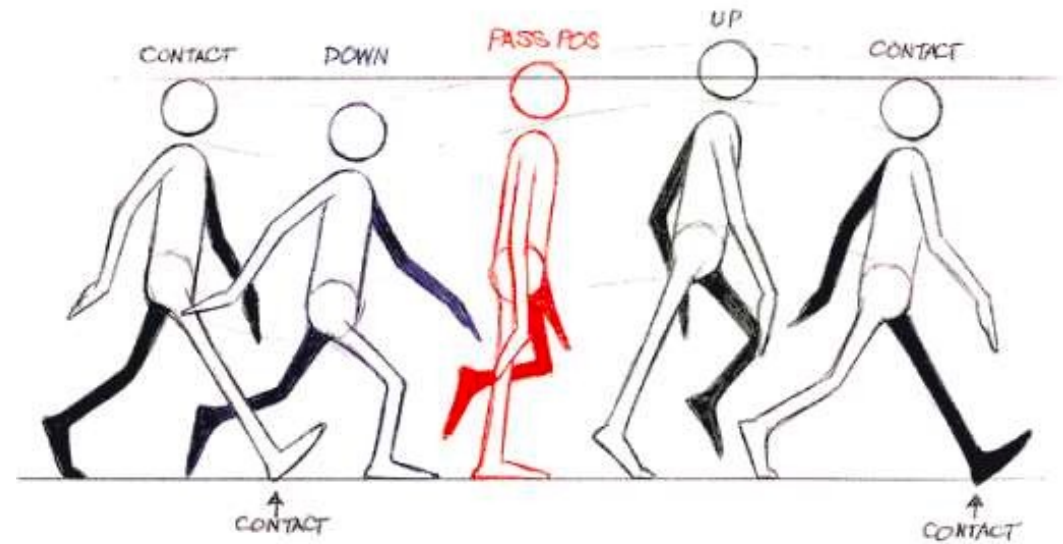
- **Easing** is a strategy where objects **accelerate** into and out of their motion
 - Derived from physics
 - Objects with **inertia** have to feel a force in order to ease their way into a new momentum
- Visualized in a 1D chart with tick marks with equal time separation but varying spatial separation
 - The closer the tick marks, the smaller the spatial separation, and the slower the motion.
- Draw a frame in the middle of frames 1 and 9 (call it 5), then a frame between 1 and 5 (call it 4), then 1 and 4 (call it 3), and then 1 and 3 (call it 2)
 - Referred to as **subdivision**
 - Easy strategy to guarantee appropriate easing



Illusion of Life, 1999

Straight Ahead vs. Pose To Pose

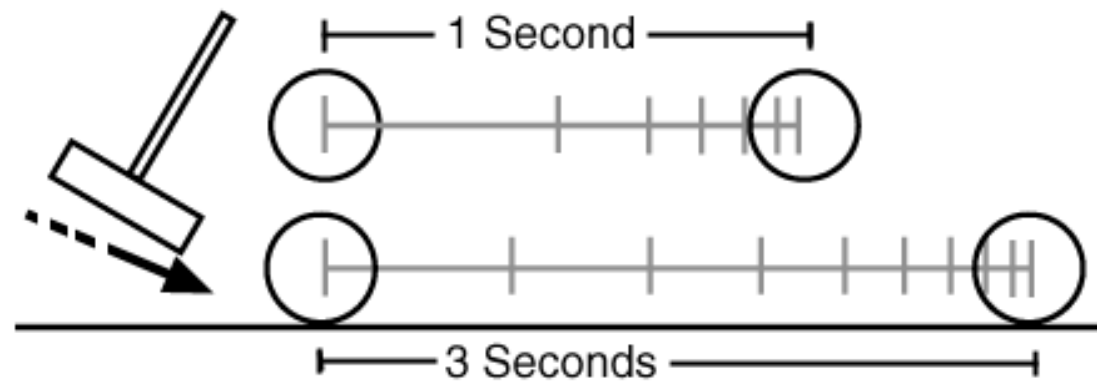
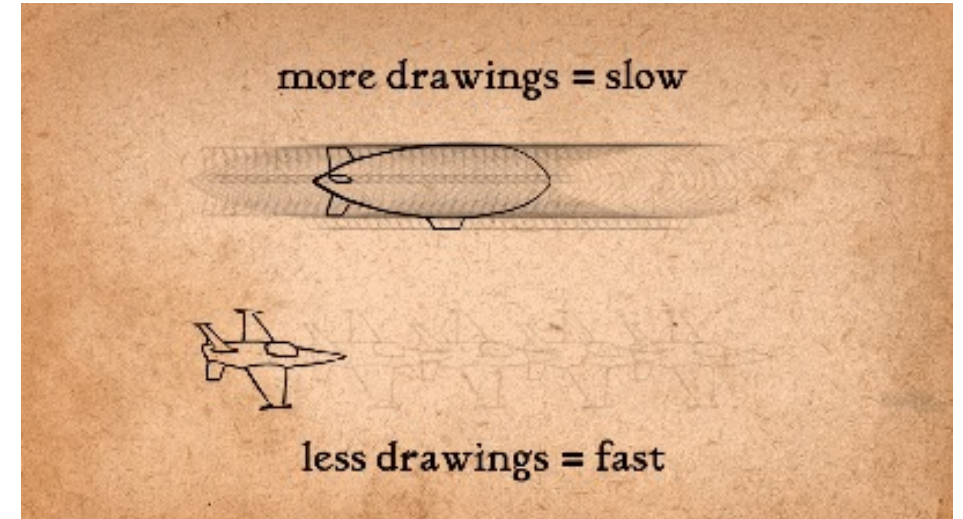
- Straight Ahead is the process of drawing in every frame **sequentially**
 - Easier to create more realistic movements this way, but harder to keep proportions constant
 - Characters end up being less dynamic and less exaggerated
- Pose to Pose is the process of drawing in **key frames first**, and then going back to draw in-betweens
 - Allows for more controlled and dynamic posing
 - Adopted more in animation settings where computers are able to help out with the in-between stages
- With Pose to Pose, senior artists draw keyframes, junior artists draw in-betweens



The Animator's Survival Kit (2001) Richard Williams

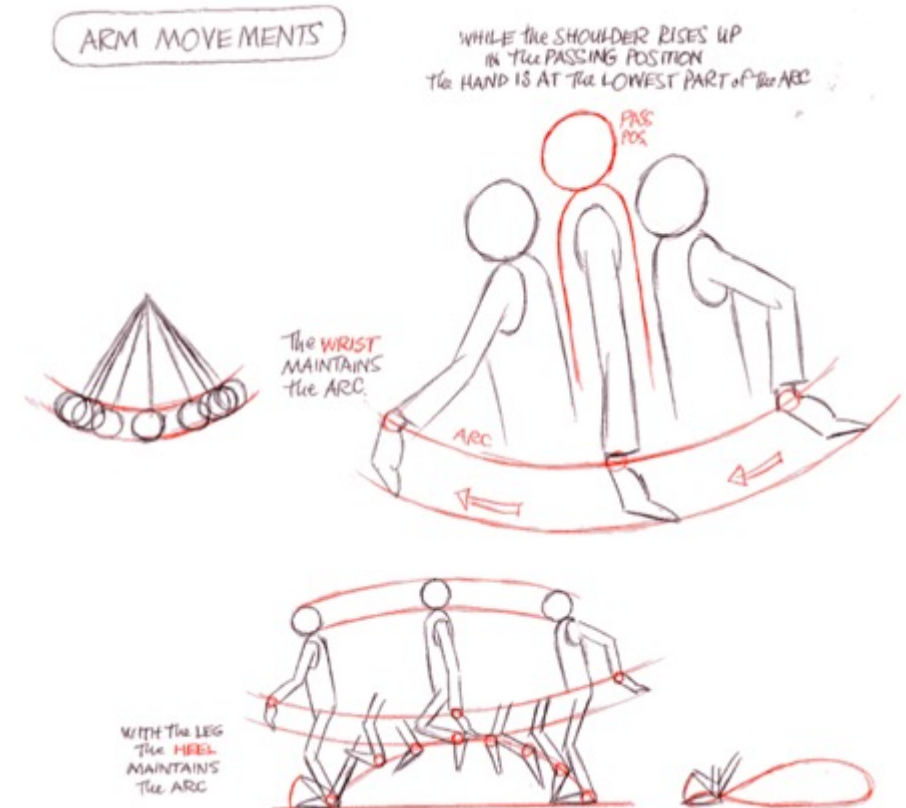
Timing

- Timing is how the motions play out, and at what **time intervals**
 - Used to determine how fast an object should be moving
 - How many frames should be used for the motion?
 - The more frames, the slower
- **Temporal linear interpolation:** velocity never changes
- **Temporal non-linear interpolation:** velocity changes



Arc Motions

- Arc Motions guarantee that **spatial trajectories are arc-like**
 - Helps to build fluidity in the motion
- Joints rotate instead of translating
 - Allows for arc-like movements
- Walk cycles are a combination of many arc movements



Natural Splines

- Can build a spline out of piecewise cubic polynomials p_i
 - Each polynomial extends from range $t = [0,1]$
 - Polynomials should connect on boundary

- Keyframes agree at endpoints [C0 continuity]:

$$p_i(t_i) = f_i, \quad p_i(t_{i+1}) = f_{i+1}, \quad \forall i = 0, \dots, n-1$$

- Tangents agree at endpoints [C1 continuity]:

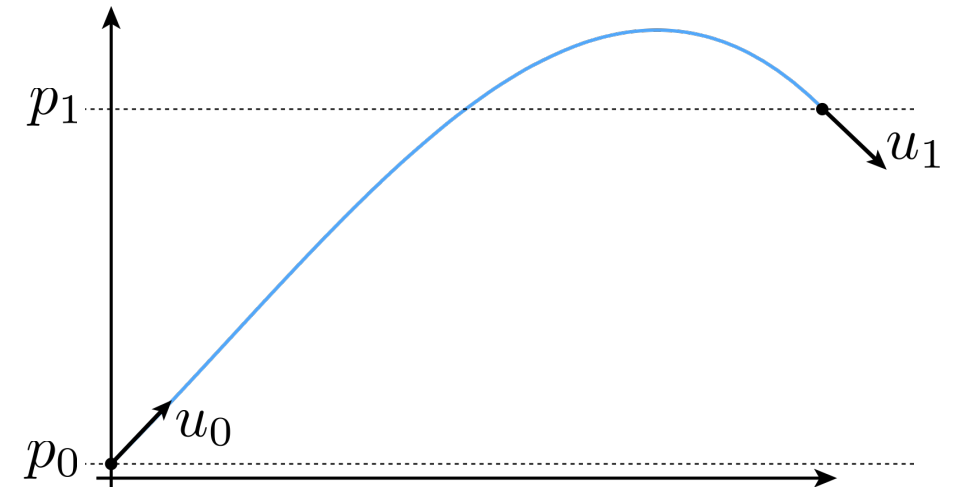
$$p'_i(t_{i+1}) = p'_{i+1}(t_{i+1}), \quad \forall i = 0, \dots, n-2$$

- Curvature agrees at endpoints [C2 continuity]:

$$p''_i(t_{i+1}) = p''_{i+1}(t_{i+1}), \quad \forall i = 0, \dots, n-2$$

- Total equations:
 - $2n + (n-1) + (n-1) = 4n - 2$
- Total DOFs:
 - $2n + n + n = 4n$
- Set curvature at endpoints to 0 and solve

$$p'_0(t_0) = 0, \quad p''_0(t_{i+1}) = 0$$



Hermite/Bézier Splines

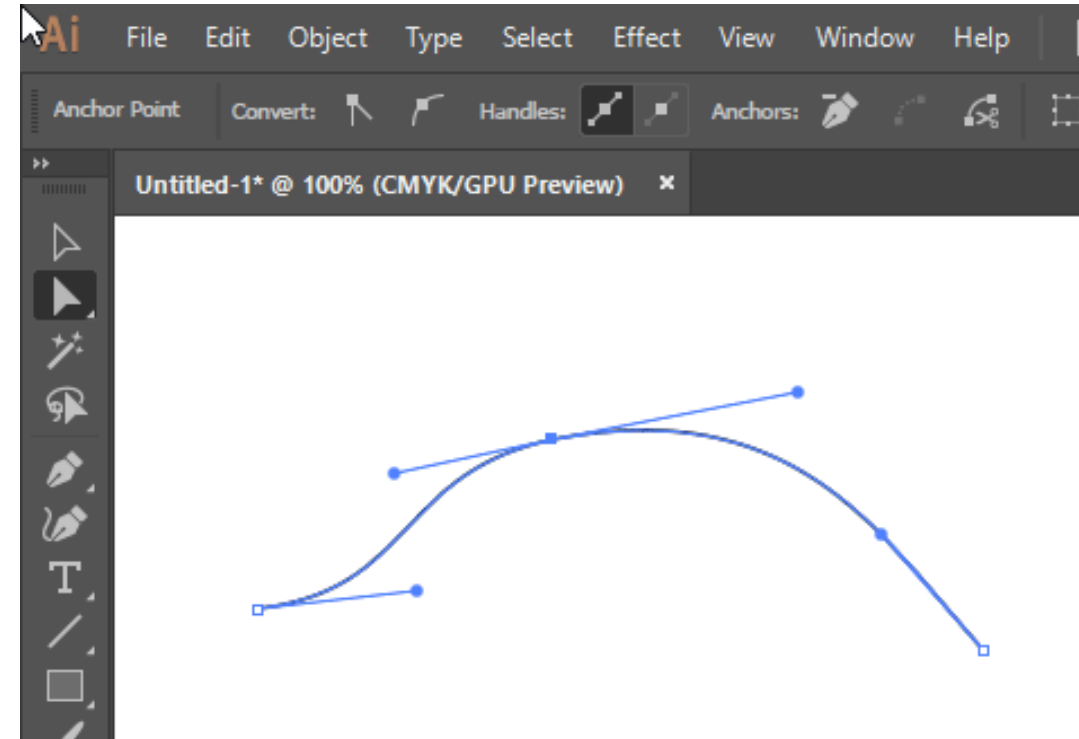
- Each cubic “piece” specified by endpoints and tangents
 - Keyframes set at endpoints:

$$p_i(t_i) = f_i, \quad p_i(t_{i+1}) = f_{i+1}, \quad \forall i = 0, \dots, n - 1$$

- Tangents set at endpoint:

$$p'_i(t_i) = u_i, \quad p'_i(t_{i+1}) = u_{i+1}, \quad \forall i = 0, \dots, n - 1$$

- Natural splines specify just keyframes
 - Bézier splines specify keyframes and tangents
 - Can get continuity if tangents are set equal
- Total equations:
 - $2n + 2n = 4n$
- Commonly used in vector art programs
 - Illustrator
 - Inkscape
 - SVGs



B-Splines

- Compute a weighted average of nearby keyframes when interpolating

- B-spline basis defined recursively, with base condition:


$$B_{i,1}(t) := \begin{cases} 1, & \text{if } t_i \leq t < t_{i+1} \\ 0, & \text{otherwise} \end{cases}$$

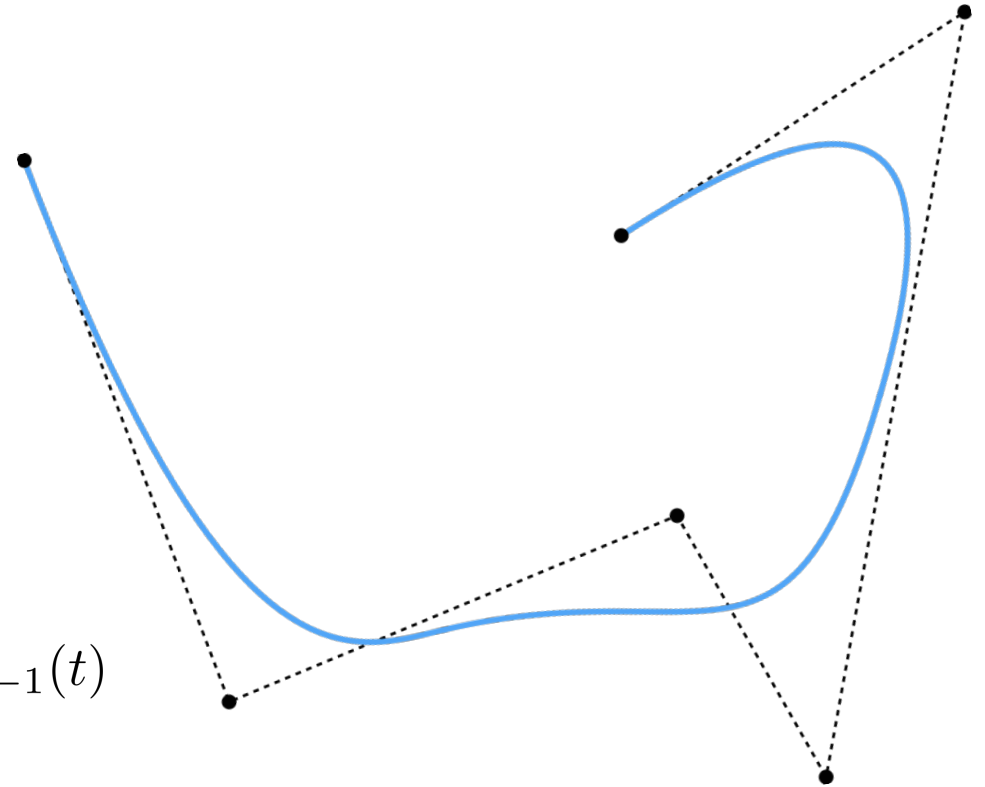
- And inductive condition:

$$B_{i,k}(t) := \frac{t-t_i}{t_{i+k-1}-t_i} B_{i,k-1}(t) + \frac{t_{i+k}-t}{t_{i+k}-t_{i+1}} B_{i+1,k-1}(t)$$

- B-spline is a linear combination of bases:

$$f(t) := \sum_i a_i B_{i,d}$$

degree 



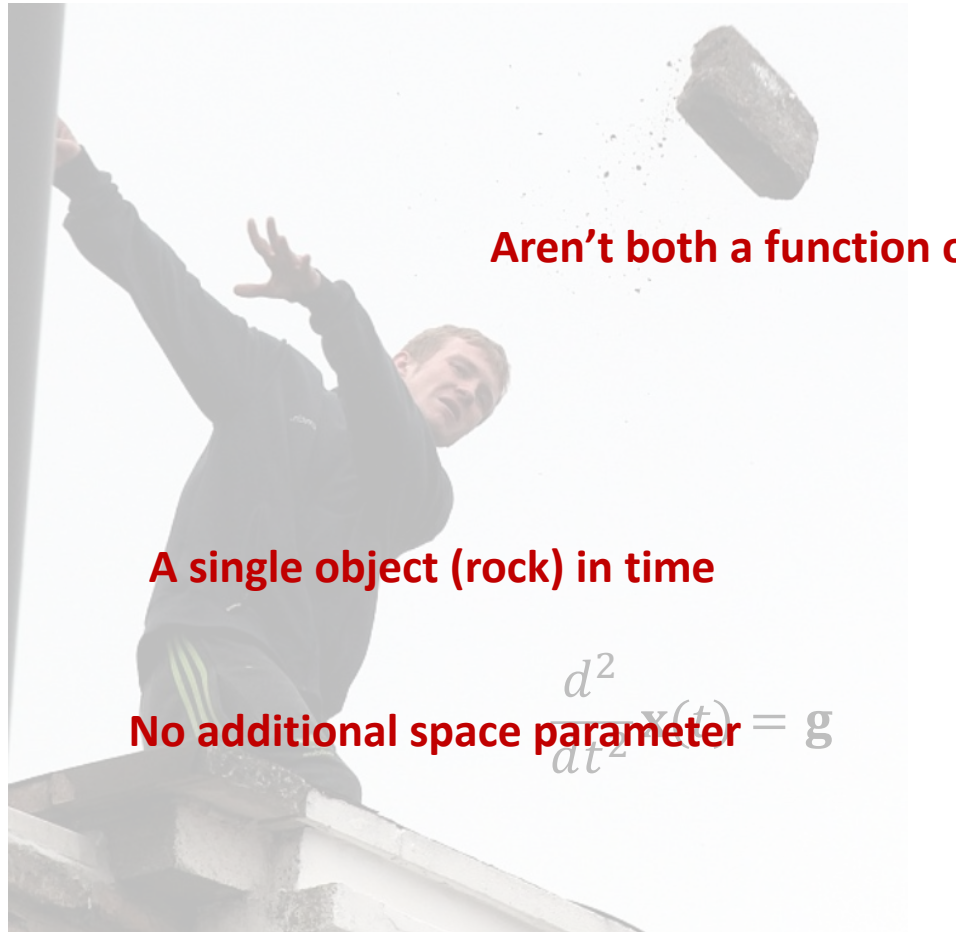
Splines Review

	[Interpolation]	[Continuity]	[Locality]
Linear	✓	✗	✓
Natural	✓	✓	✗
Hermite	✓	✗	✓
Bezier	✓	✗	✓
Catmull-Rom	✓	✗	✓
B-Spline	✗	✓	✓

Simulations

- ODE vs PDE
- Time Integration
 - Forwards Euler
 - Symplectic Euler
- Lagrangian
 - 2nd Derivative
- Boundary Conditions
 - Dirichlet
 - Neumann

ODEs vs. PDEs



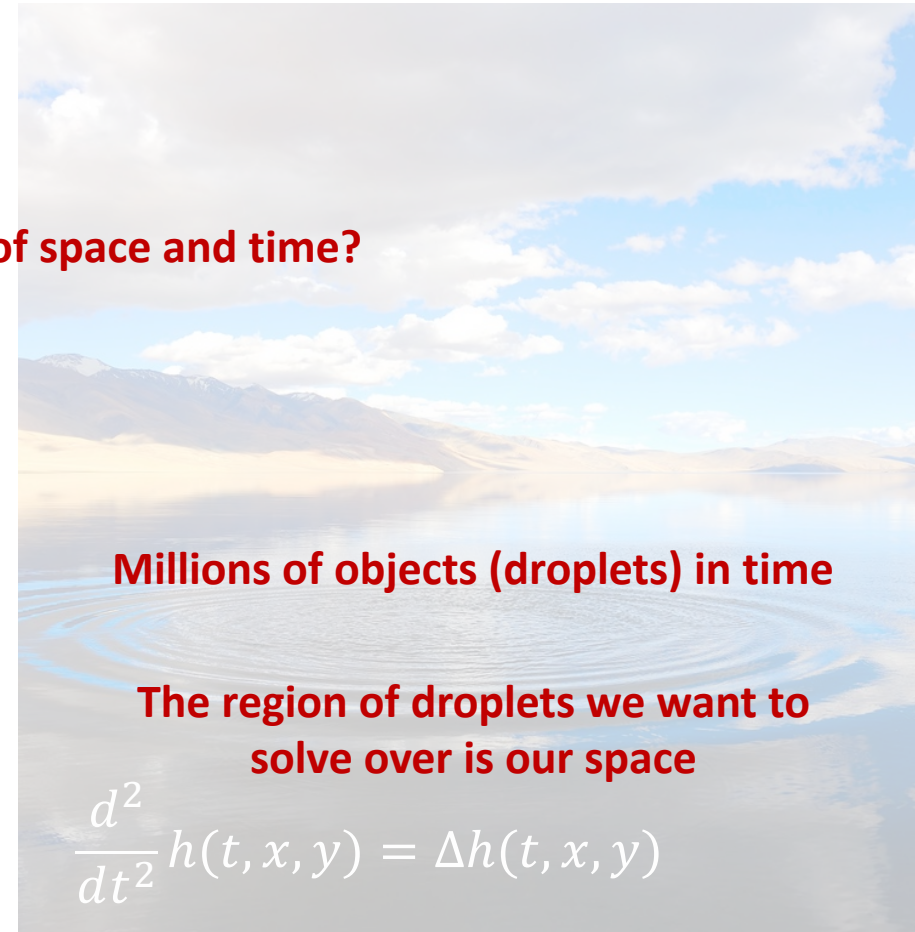
Aren't both a function of space and time?

A single object (rock) in time

No additional space parameter

$$\frac{d^2}{dt^2} \mathbf{x}(t) = \mathbf{g}$$

[ODE] yeeting a rock



Millions of objects (droplets) in time

The region of droplets we want to solve over is our space

$$\frac{d^2}{dt^2} h(t, x, y) = \Delta h(t, x, y)$$

[PDE] yeeted rock lands in pond

Explicit Euler Methods

[Forward]

$$v_{k+1} = v_k + \tau * a(q_k)$$

$$q_{k+1} = q_k + \tau * v_k$$

[Symplectic]

$$v_{k+1} = v_k + \tau * a(q_k)$$

$$q_{k+1} = q_k + \tau * v_{k+1}$$

[Verlet]

$$v_{k+1} = v_{k+0.5} + \frac{\tau}{2} * a(q_k)$$

$$q_{k+1} = q_k + \tau * v_{k+1}$$

$$v_{k+1.5} = v_{k+1} + \frac{\tau}{2} * a(q_k)$$

[RK2]

$$v'_{k+1} = \tau * a(q_k)$$

$$v''_{k+1} = \tau * a\left(q_k + \frac{v'_{k+1}}{2}\right)$$

$$v_{k+1} = v_k + v''_{k+1}$$

$$q_{k+1} = q_k + \tau * v_{k+1}$$

[RK4]

$$v'_{k+1} = \tau * a(q_k)$$

$$v''_{k+1} = \tau * a\left(q_k + \frac{v'_{k+1}}{2}\right)$$

$$v'''_{k+1} = \tau * a\left(q_k + \frac{v''_{k+1}}{2}\right)$$

$$v''''_{k+1} = \tau * a\left(q_k + v'''_{k+1}\right)$$

$$q_{k+1} = q_k + \frac{1}{6} (v'_{k+1} + 2v''_{k+1} + 2v'''_{k+1} + v''''_{k+1})$$

The Laplacian Operator

- All of our model equations used the Laplace operator
 - Laplace Equation $\Delta u = 0$
 - Heat Equation $\dot{u} = \Delta u$
 - Wave Equation $\ddot{u} = \Delta u$
- Unbelievably important object showing up everywhere across physics, geometry, signal processing, and more
- What does the Laplacian mean?
 - **Differential operator:** eats a function, spits out its 2nd derivative
 - What does that mean for a function: $u: \mathbb{R}^n \rightarrow \mathbb{R}$?
 - Divergence of gradient

$$\Delta u = \nabla \cdot \nabla u$$

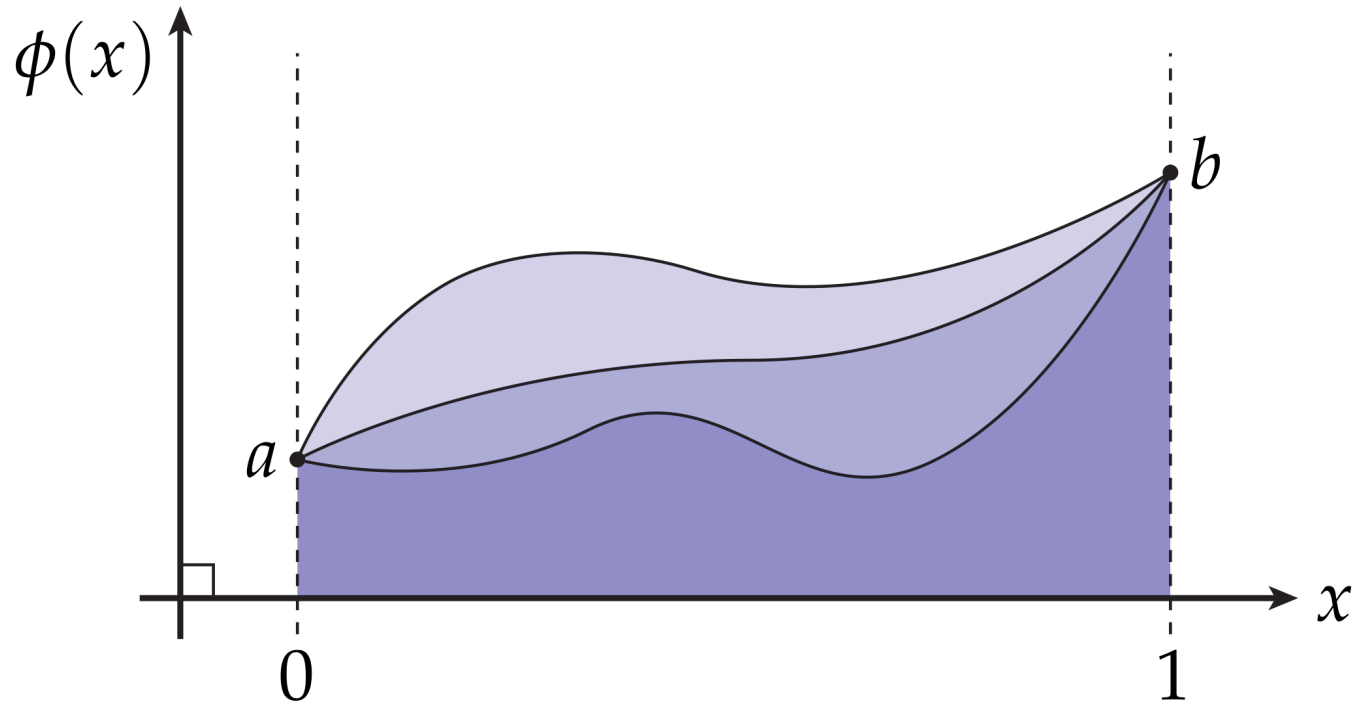
- Sum of second derivatives

$$\Delta u = \frac{\partial u^2}{\partial x_1^2} + \dots + \frac{\partial u^2}{\partial x_n^2}$$

- Deviation from local average
- ...

Dirichlet Boundary Conditions

Dirichlet: boundary data always set to fixed values

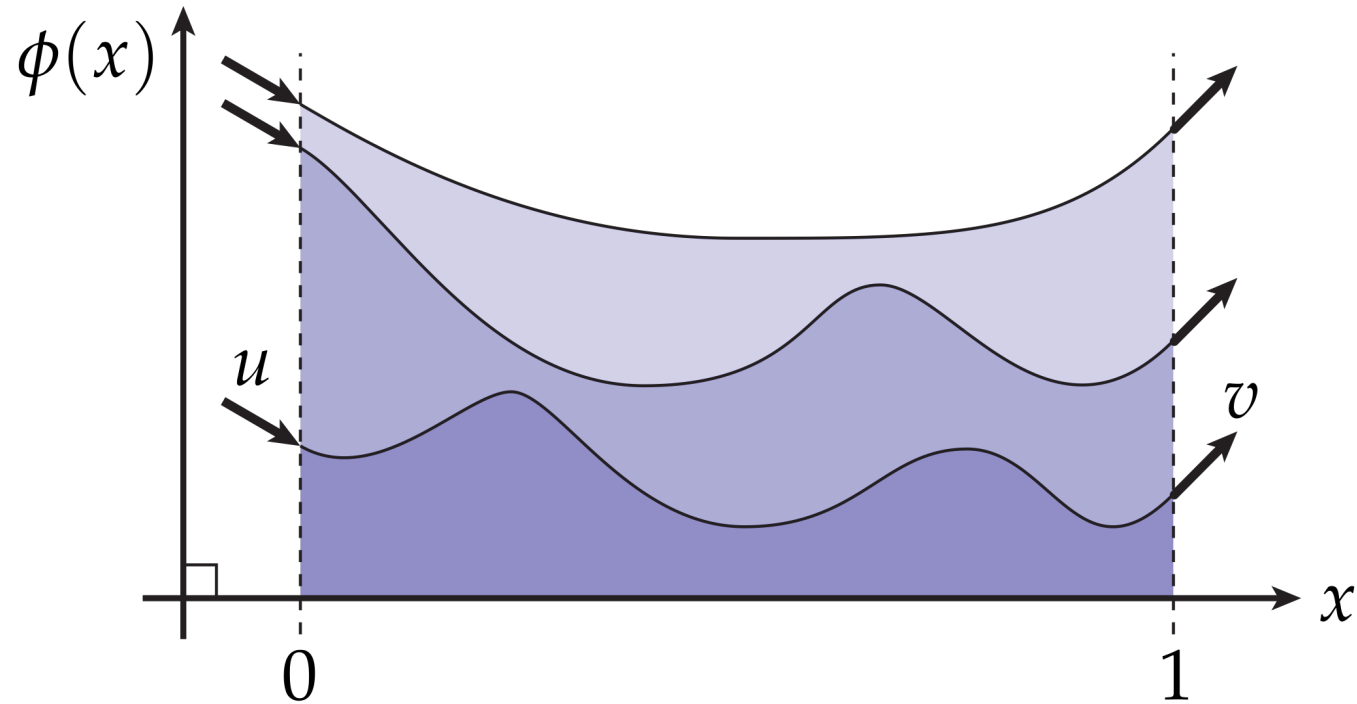


Example: $\phi(0) = a, \phi(1) = b$

Many possible functions interpolate values in between

Neumann Boundary Conditions

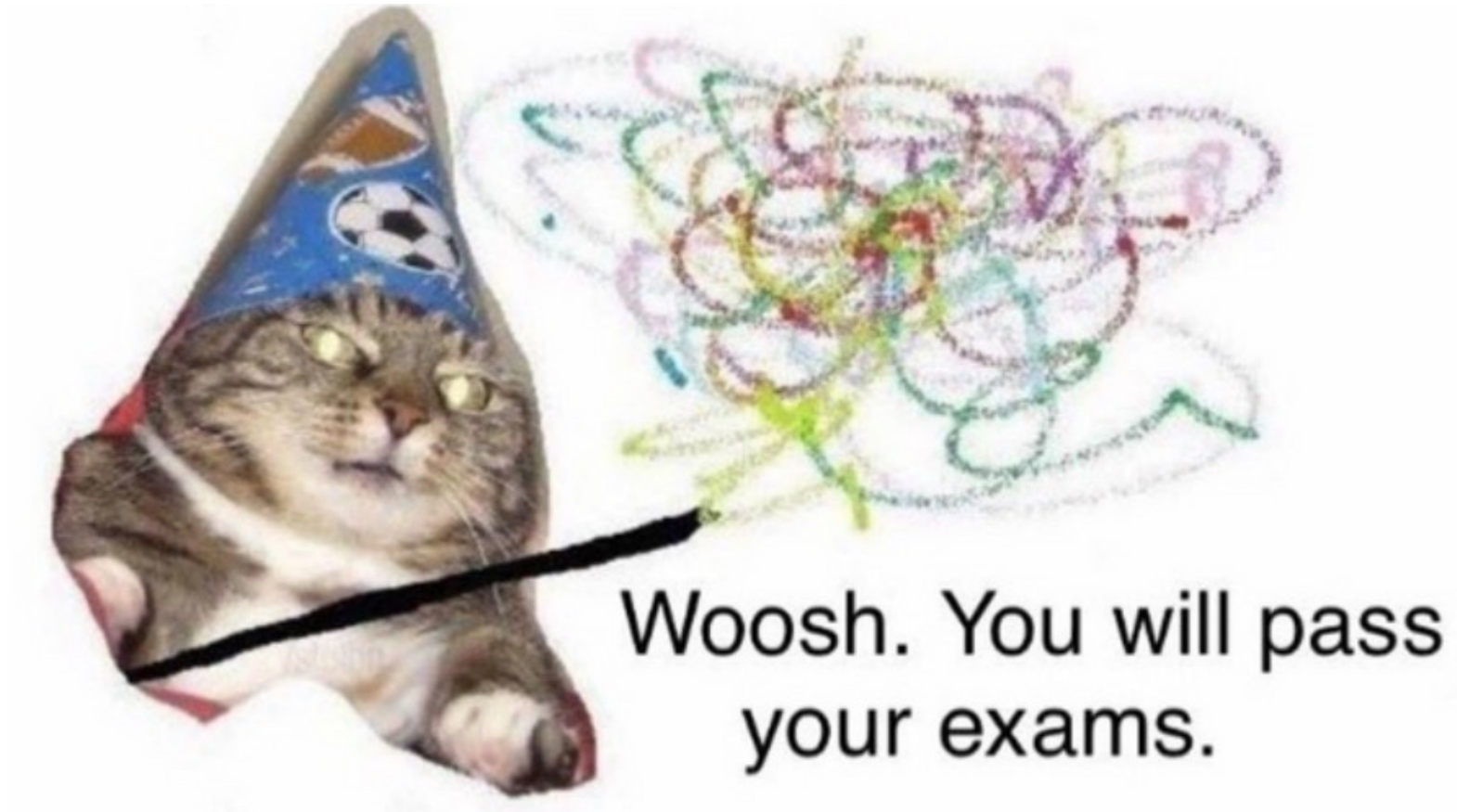
Neumann: specify derivatives across boundary



Example: $\phi'(0) = u$, $\phi'(1) = v$

Again, many possible functions

Good Luck!



Course Wrapup

15-462 / 15-662 Computer Graphics

Upcoming Courses

Fall 2024

15-327/15-627	Monte Carlo Methods and Applications	Keenan Crane, Gautam Iyer
15-362/15-662	Computer Graphics	Oscar Dadfar, Minchen Li
15-463/15-663/15-862	Computational Photography	Ioannis Gkioulekas
15-466/15-666	Computer Game Programming	Jim McCann
15-473/15-673	Visual Computing Systems	Oscar Dadfar
15-472/15-672/15-772	Real-Time Graphics	Jim McCann

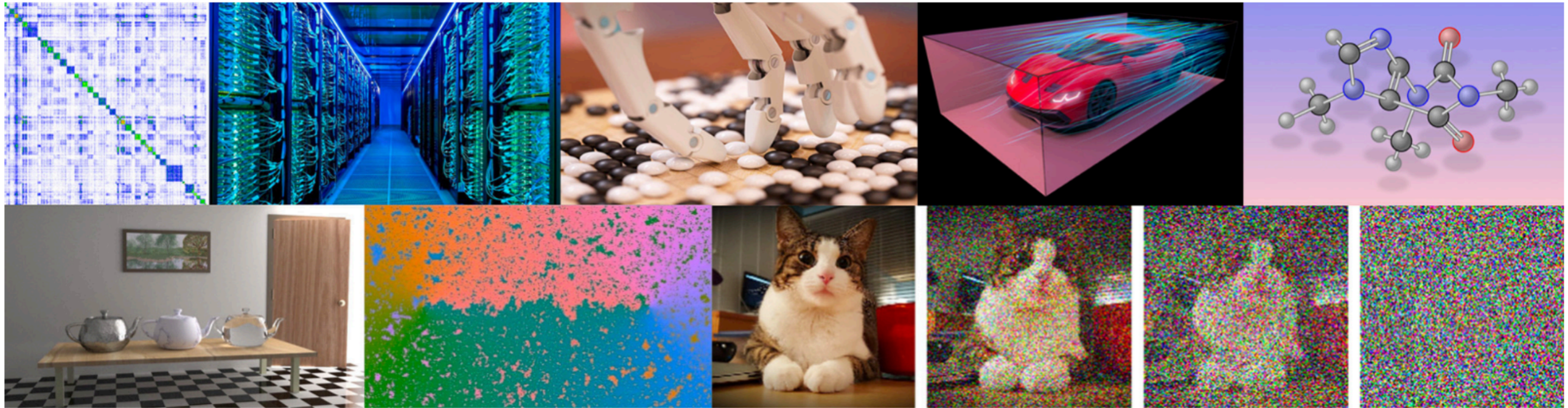
Spring 2025

15-367/15-867	Algorithmic Textiles Design	Jim McCann
15-458/15-858	Discrete Differential Geometry	Keenan Crane
15-362/15-662	Computer Graphics	Nancy Pollard
15-464/15-664	Technical Animation	Nancy Pollard
15-468/15-668/15-868	Physics-based Rendering	Ioannis Gkioulekas
15-769	Physics-based Animation of Solids and Fluids	Minchen Li
16-726	Learning-based Image Synthesis	Jun-Yan Zhu

Monte Carlo Methods and Applications

CMU 21-387 | 15-327 | 15-627 | 15-860 FALL 2023

geometry.cs.cmu.edu/montecarlo



[Home](#) — [Course Info](#) — [Schedule](#) — [Assignments](#) — [Resources](#) — [Course Policies](#)

Instructors: Keenan Crane (CSD/RI) and Gautam Iyer (MSC)

Course Info

The Monte Carlo method uses random sampling to solve computational problems that would otherwise be intractable, and enables computers to model complex systems in nature that are otherwise too difficult to simulate. This course provides a first introduction to Monte Carlo methods from complementary theoretical and applied points of view, and will include implementation of practical algorithms. Topics include random number generation, sampling, Markov chains, Monte Carlo integration, stochastic processes, and applications in computational science. Students need a basic background in probability, multivariable calculus, and some coding experience in any language. Coding assignments will be done in [Python](#) 🐍.

15-463, 15-663, 15-862 Computational photography

Fall 2023



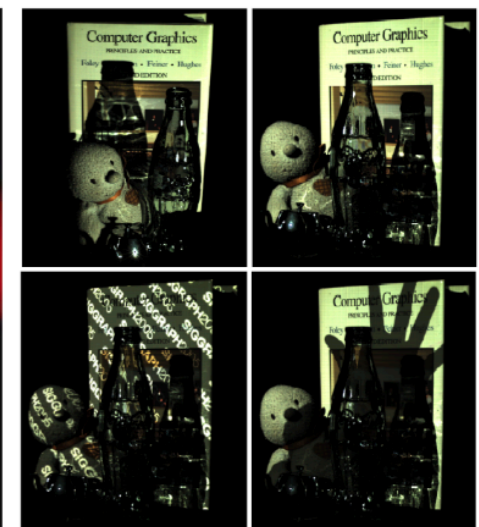
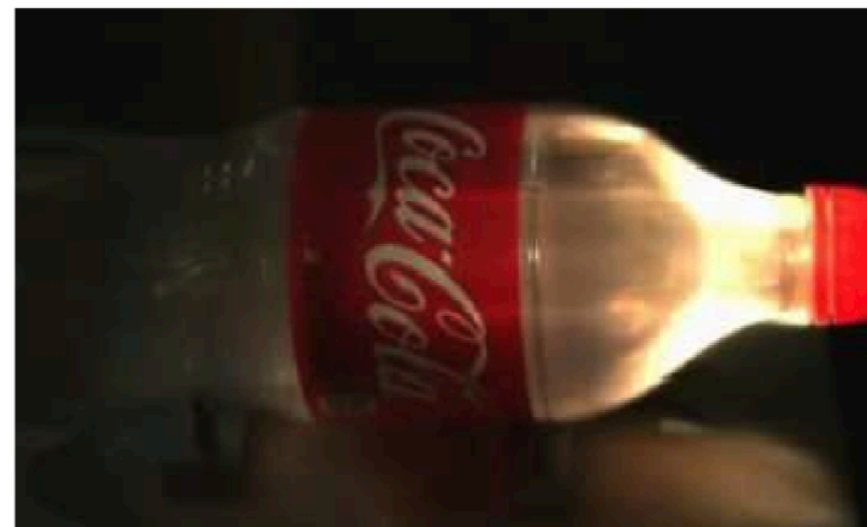
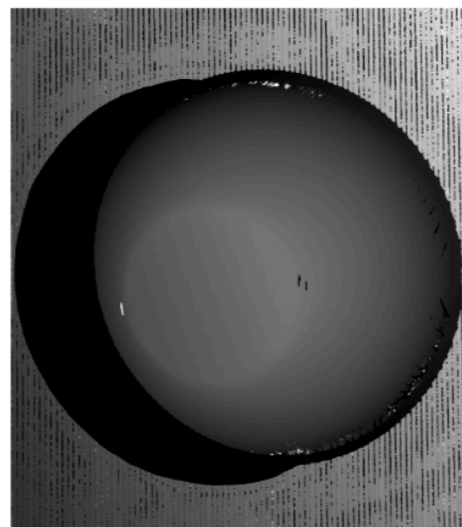
📅 Time: Mondays, Wednesdays 11:00 am - 12:20 pm ET

📍 Location: GHC 4303

👤 Instructor: [Ioannis \(Yannis\) Gkioulekas](#)

👤 Teaching assistants: [Tanli Su](#)

☰ Platforms: [Canvas](#), [Slack](#)



Computational photography brings together graphics, vision, optics, and imaging. In this course you will learn all about: (top, left to right) photographic optics and image processing pipeline; exposure, high-dynamic-range imaging, and noise; color and image editing; focus, lightfields, and coded photography; (bottom, left to right) radiometry and photometric stereo; projective geometry, stereo, and structured light; time-of-flight imaging and computational light transport.

15-463, 15-663, 15-862 Computational photography

Fall 2023



📅 Time: Mondays, Wednesdays 11:00 am - 12:20 pm ET

📍 Location: GHC 4303

👤 Instructor: [Ioannis \(Yannis\) Gkioulekas](#)

👤 Teaching assistants: [Tanli Su](#)

☰ Platforms: [Canvas](#), [Slack](#)

Course description

Computational photography is the convergence of computer graphics, computer vision, optics, and imaging. Its role is to overcome the limitations of traditional cameras, by combining imaging and computation to enable new and enhanced ways of capturing, representing, and interacting with the physical world.

This course provides an overview of the state of the art in computational photography. At the start of the course, we will study modern image processing pipelines, including those encountered on mobile phone and DSLR cameras, and advanced image and video editing algorithms. Then we will continue to learn about the physical and computational aspects of tasks such as 3D scanning, coded photography, lightfield imaging, time-of-flight imaging, VR/AR displays, and computational light transport. Near the end of the course, we will discuss active research topics, such as creating cameras that capture video at the speed of light, cameras that look around walls, or cameras that can see below skin.

The course has a strong hands-on component, in the form of seven homework assignments and a final project. In the homework assignments, students will have the opportunity to implement many of the techniques covered in the class, by both acquiring their own images of indoor and outdoor scenes and developing the computational tools needed to extract information from them. Example homework includes building end-to-end HDR imaging pipelines and structured light scanners. For their final projects, students will have the choice to use modern sensors and other optical instrumentation provided by the instructor (lightfield cameras, time-of-flight sensors, projectors, laser sources, and so on).

Cross-listing: This is both an advanced undergraduate and introductory graduate course, and it is cross-listed as 15-463 (for undergraduate students), 15-663 (for Master's students), and 15-862 (for PhD students). Please make sure to register for the section of the class that matches your current enrollment status.

15-466/15-666 Computer Game Programming

<http://graphics.cs.cmu.edu/courses/15-466-f21/>

Computer Game Programming (Fall, 2023)

TR 15:00–16:50 in Posner Hall 151.

Taught by **Jim McCann** (Office Hours after class in EDSH 229 or **by appointment**.)

With TA help from Alan Lee (Office Hours Monday 17:00–19:00 in the Smith Hall second floor comm on area; e-mail **here**.)



Games

In this class, we made a bunch of small games, as well as five final games. These games were launch[ed] *live and in person* on Friday, December 8th at 4-7pm in Tepper Simmons B.

- Ultimate Pillow Fight
- Assemble Monsters
- Tech Wiz
- Swordmaster
- Help, I'm Trapped in the Psychedelic Time Dimension Again (Second Time This Week)

15-469/669: Visual Computing Systems

[reset](#) |



Fall 2023

Instructor: [Oscar Dadfar](#) | OH Fri 3pm - 5pm Graphics Lounge

TA: Daniel Zheng | OH Mon. + Wed. 1pm - 2pm Graphics Lounge

Tues/Thurs 3:30pm - 4:50pm | WEH 8427

12 Units

Visual computing tasks such as computational imaging, image/video understanding, and real-time graphics are key responsibilities of modern computer systems ranging from sensor-rich smart phones to large datacenters. These workloads demand exceptional system efficiency and this course examines the key ideas, techniques, and challenges associated with the design of parallel, heterogeneous systems that accelerate visual computing applications. This course is intended for graduate and advanced undergraduate-level students interested in architecting efficient graphics, image processing, and computer vision platforms.

Description | Visual computing tasks such as computational imaging, image/video understanding, and real-time graphics are key responsibilities of modern computer systems ranging from sensor-rich smart phones to large datacenters. These workloads demand exceptional system efficiency, and this course examines the key ideas, techniques, and challenges associated with the design of parallel, heterogeneous systems that accelerate visual computing applications. This course is intended for graduate and advanced undergraduate-level students interested in architecting efficient graphics, image processing, and computer vision platforms.

Real-Time Computer Graphics

Also known as 15-472/672/772. Tuesdays and Thursdays from 14:00-15:20 in Posner Hall 151 during Spring 2024. Taught by Jim McCann. Office hours after class in Smith 229 or by appointment.

Real-time computer graphics is about building systems that leverage modern CPUs and GPUs to produce detailed, interactive, immersive, and high-frame-rate imagery. Students will build a state-of-the-art renderer using C++ and the Vulkan API. Topics explored will include efficient data handling strategies; culling and scene traversal; multi-threaded rendering; post-processing, depth of field, screen-space reflections; volumetric rendering; sample distribution, spatial and temporal sharing, and anti-aliasing; stereo view synthesis; physical simulation and collision detection; dynamic lights and shadows; global illumination, accelerated raytracing; dynamic resolution, "AI" upsampling; compute shaders; parallax occlusion mapping; tessellation, displacement; skinning, transform feedback; and debugging, profiling, and accelerating graphics algorithms.

Principles

This course is based on four principles:

- ✦ **Do Things for Reasons** -- if you don't understand why you are doing something, look deeper. (Corr: if you don't understand why someone else is doing something, ask.)
- ✦ **No Magic** -- (as much as possible) avoid black boxes and needless helper libraries.
- ✦ **Test and Improve** -- make things efficient through real-world testing. (Corr: asymptotic complexity is not the only factor; constants matter in the real world.)
- ✦ **Go Big** -- push systems to their limits.

Real-Time Graphics

What it means to be real-time and general techniques for getting there.

T Jan 16 Course Overview; Core Principles; Interactive, Real-Time, and Beyond; Big Ideas; CPU Benchmarking and Fast Code Intuition; **»A0a** Timing, Math, and Intrinsic;



Tuesday, January 16th

The Graphics Pipeline



How we get pixels onto the screen quickly. From attributes to framebuffers.

R Jan 18 Vulkan Intro; A Picture of Vulkan; Uniforms, Attributes, Vertices, Primitives, Framebuffers; Opening a Window; **»A0b** Vulkan Tutorial; **T Jan 23** Framework Style; Vulkan Debugging Session; **»A0a** Scoreboard Check-In; **R Jan 25** Javascript; Build systems;

Data

Getti
TJ
sub
nor
con
(an
Exp

Culling and Occlusion



What you don't render can't slow you down.

R Feb 1 Frustum culling; Portal rendering; Level-of-detail; Occlusion queries; Geometry images; **T Feb 6** Bottlenecks; **R Feb 8: No Class / HI Work Time**

Materials

Matching the real world, or doing better.

T Feb 13 detail with textures; **R Feb 15** physical materials; procedural materials; approximations and ground truth; material capture; antialiasing; glossy surfaces; precomputation; **»A2** Materials out; **T Feb 20** Interesting Mesh Ideas; **»A1** Scene Viewer due (before class); **R Feb 22** HI debrief + Code Share;



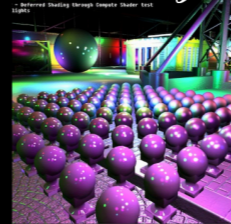
Color



Primaries, white points, and encodings oh my.

T Feb 27 color spaces; demo: P3 vs sRGB primaries; demo: HDR; vulkan formats and color spaces;

Direct Lighting

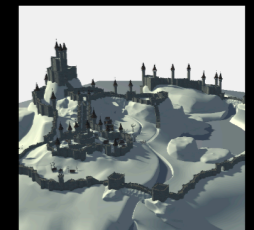


The first bounce of lighting is always the brightest.
T Mar 12 sphere, spot, and sun lights; lambertian contribution from a spherical area light; the "representative point" approximation; light loops;

Shadows

Solving the visibility problem.

R Mar 14 shadow maps; cubic shadow maps; percentage-closer filtering; soft shadows with PCSS;
F Mar 15; **»A3** Lights out;



Real-time Raytracing

Just brute-forcing the lighting.

T Mar 19 raytracing extensions, sample sharing;



Post-processing

Fixing the pixels later.

T Mar 26 bokeh; lens flares; motion blur; SSO; screen-space reflections;



Volume Rendering

Clouds, medical data, and fire.

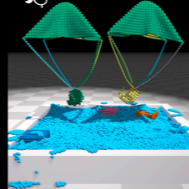
R Mar 28 raymarching smoke and clouds; screen-space surfacing for liquids; Nubus3;



Simulation

Making things move without keyframes.

T Apr 2 Eulerian vs Lagrangian; shallow-wave equations; grid-based smoke; **»F** Final Project topic selection; **R Apr 4** particle-based smoke; particle-based fluid; particle-based solids; **F Apr 5**; **»A3** Lights due (end of day); **»F** Final Project out;



29th

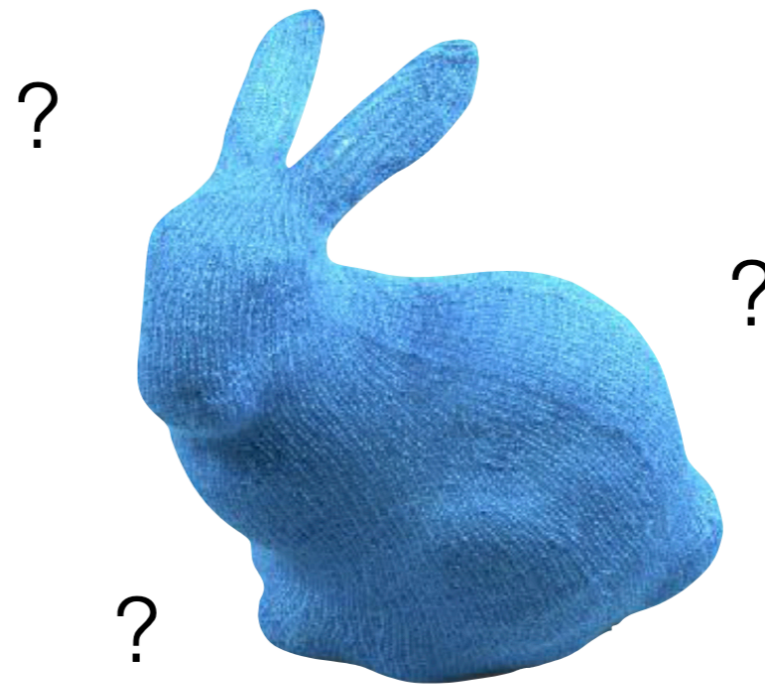
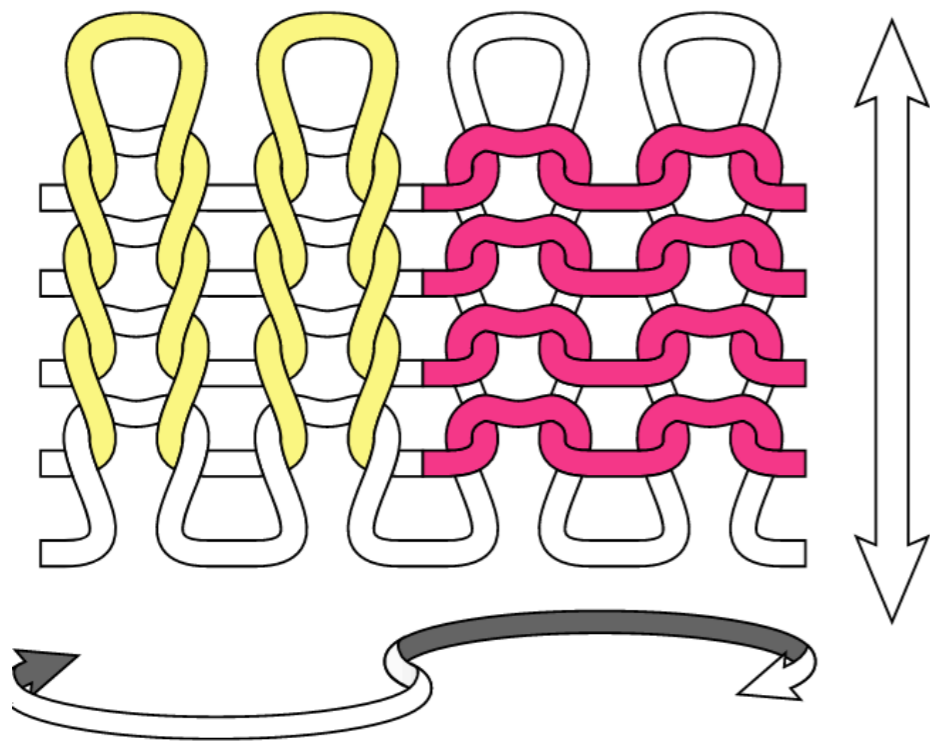
Thursday, March 14th
-Friday, March 15th

Thursday, March 28th

15-367/15-867 Algorithmic Textiles Design

Description

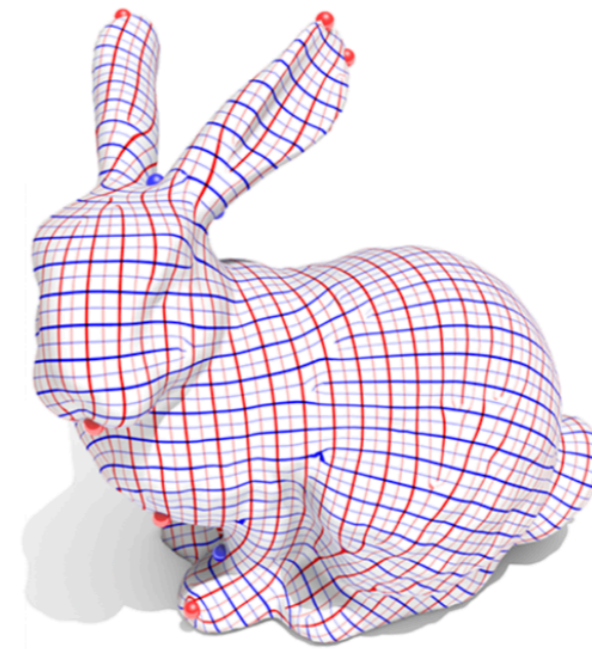
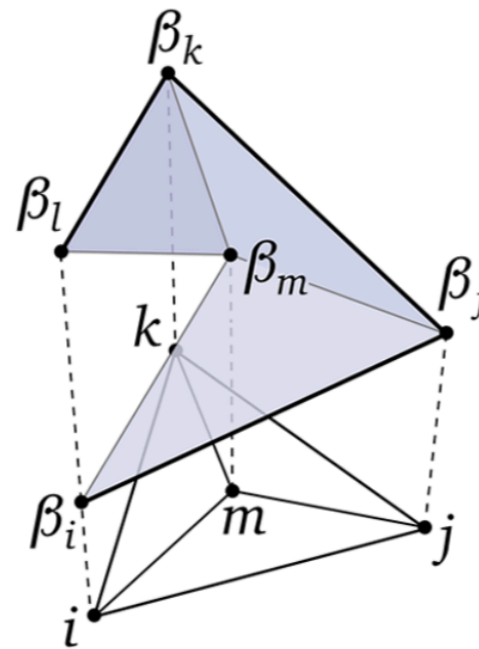
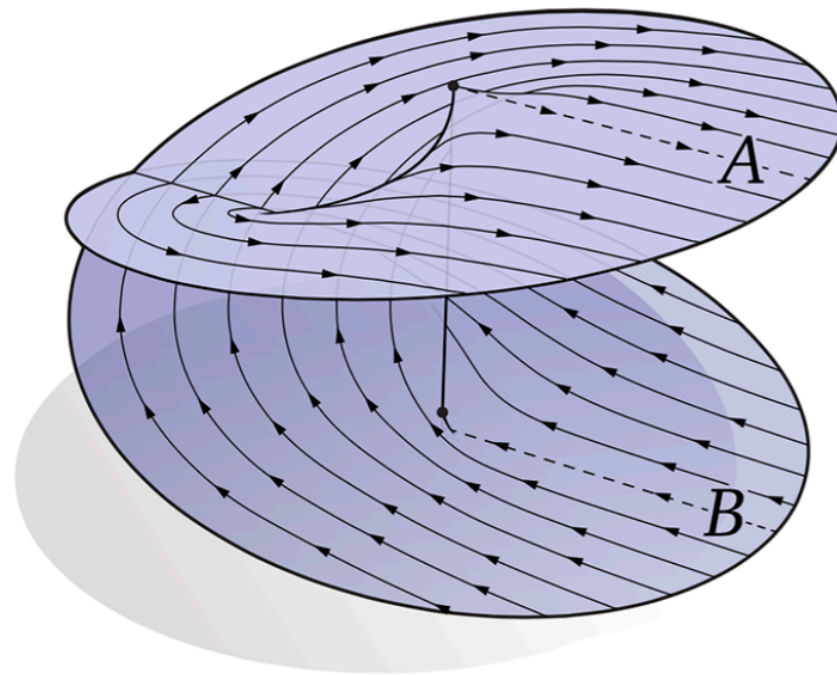
Textile artifacts are -- quite literally -- all around us; from clothing to carpets to car seats. These items are often produced by sophisticated, computer-controlled fabrication machinery. In this course we will discuss everywhere code touches textiles fabrication, including design tools, simulators, and machine control languages. Students will work on a series of multi-week, open-ended projects, where they use code to create patterns for modern sewing/embroidery, weaving, and knitting machines; and then send these patterns to be fabricated in the textiles lab. Students in the 800-level version of the course will be required to create a final project which develops a new algorithm, device, or technique in the realm of textiles fabrication.



CS 15-458/858: Discrete Differential Geometry

CARNEGIE MELLON UNIVERSITY | SPRING 2022 | TUE/THU 11:50-1:10 | GHC 4215

[ASSIGNMENTS](#) [CALENDAR](#) [COURSE DESCRIPTION](#) [COURSE NOTES](#) [GRADING POLICY](#) [SLIDES](#)



Reading 9—Choose Your Own Adventure (due 4/26)

April 19,
2022

Uncateg
orized

There are *way* more topics and ideas in Discrete Differential Geometry than we could ever hope to cover in this course. For this final reading assignment, you can choose from one of several options that we'll cover in the remainder of our course:

<https://brickisland.net/DDGSpring2022/>

CS 15-458/858: Discrete Differential Geometry

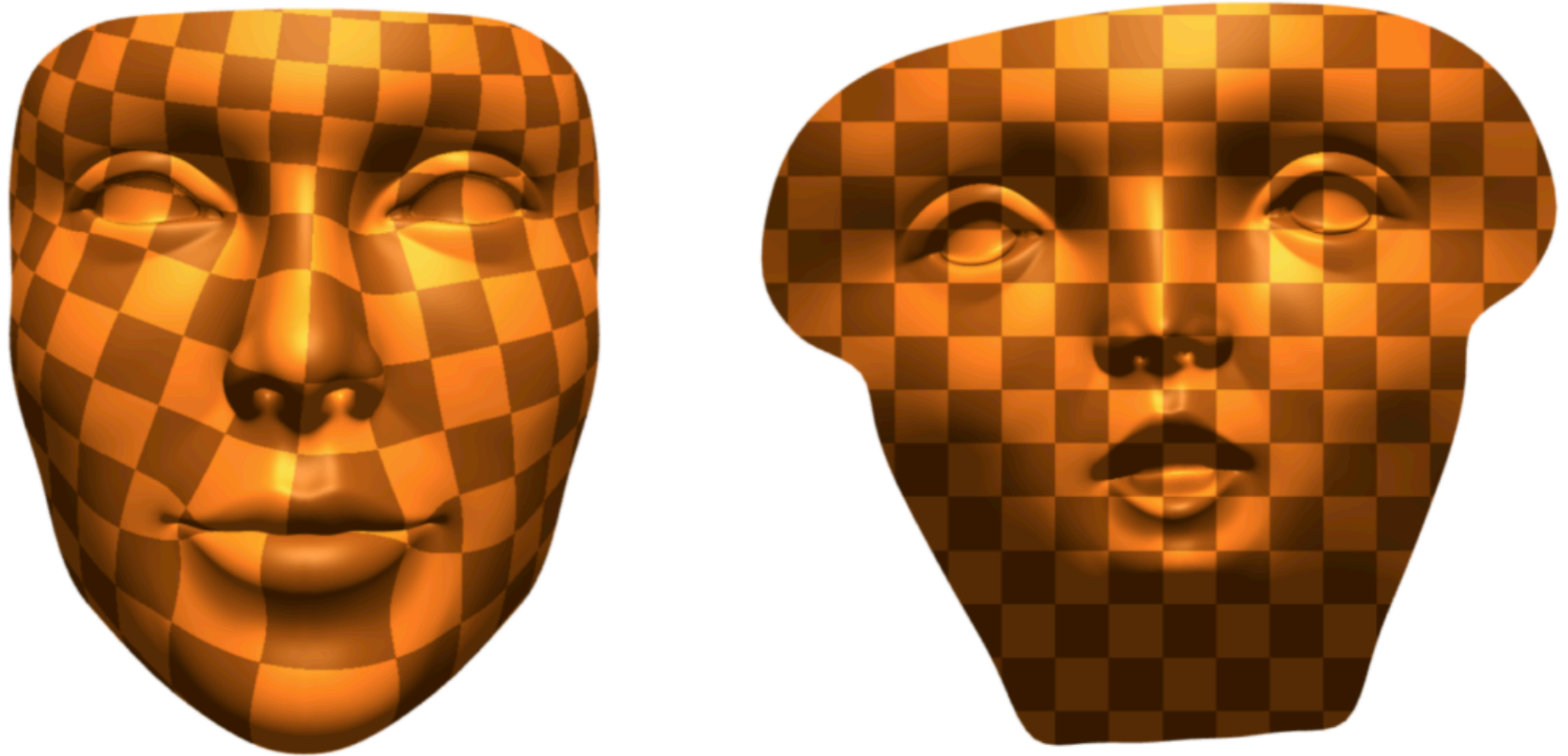
CARNEGIE MELLON UNIVERSITY | SPRING 2022 | TUE/THU 11:50-1:10 | GHC 4215

Assignment 4 [Coding]: Conformal Parameterization (due 4/20)

April 6,
2022

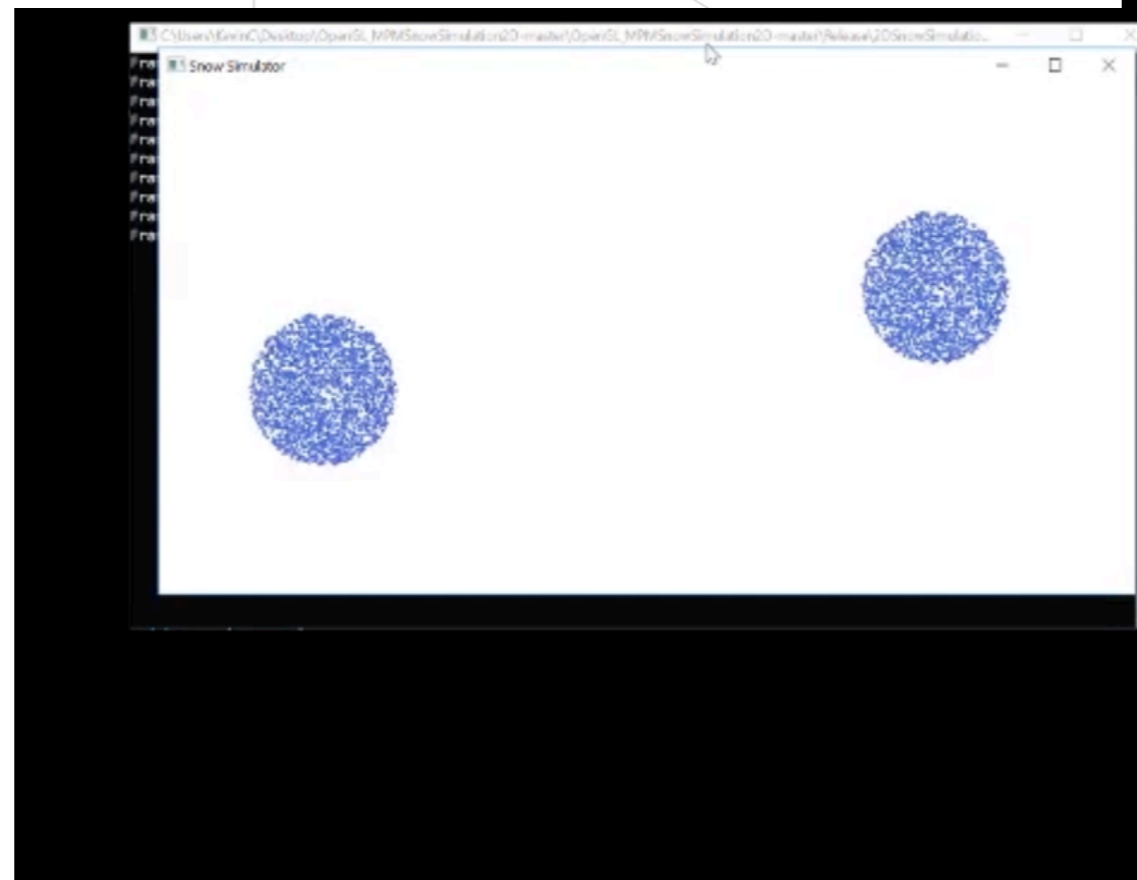
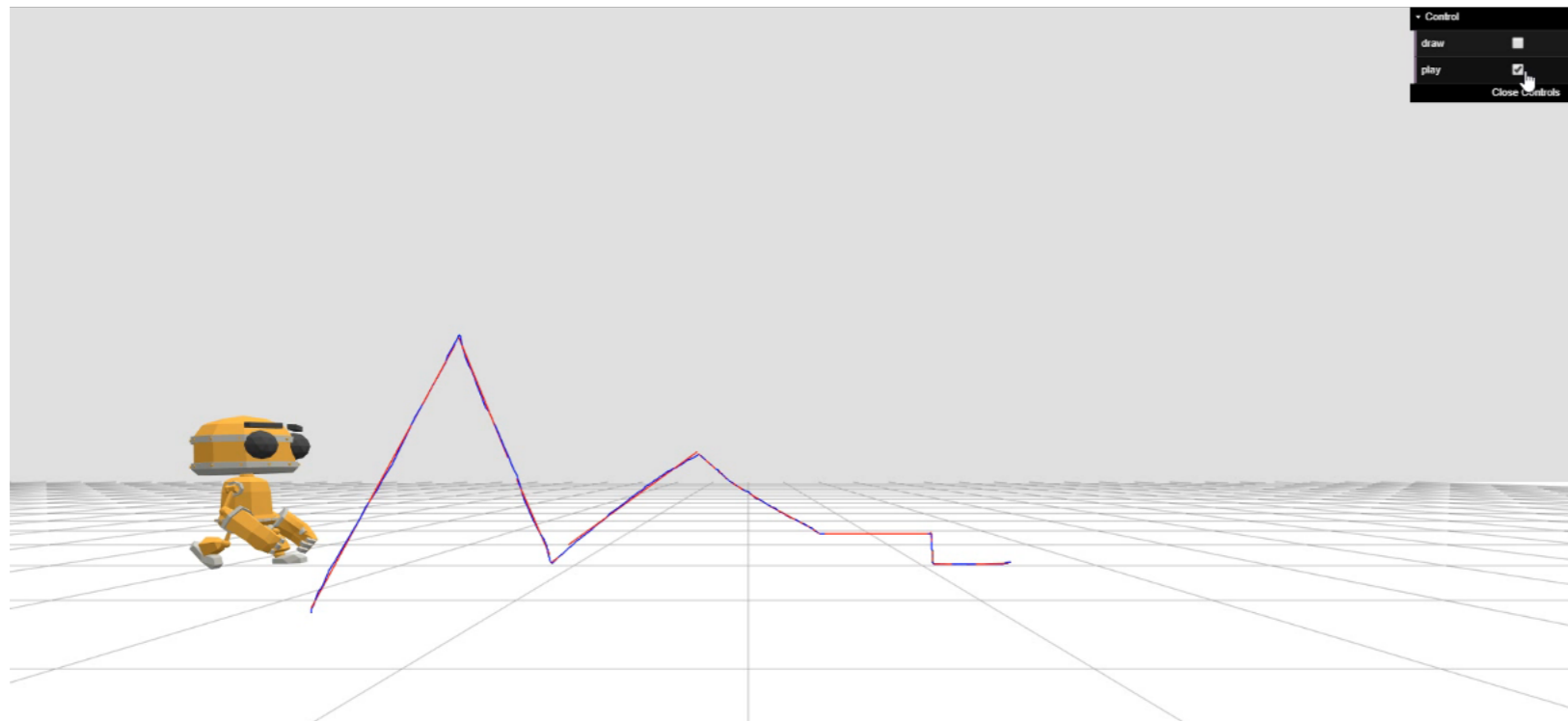
Assignm
ents

Leave a
commen
t



For the coding portion of your assignment on conformal parameterization, you will implement the [Spectral Conformal Parameterization](#) (SCP) algorithm as described in the course notes. Please implement the following routines in

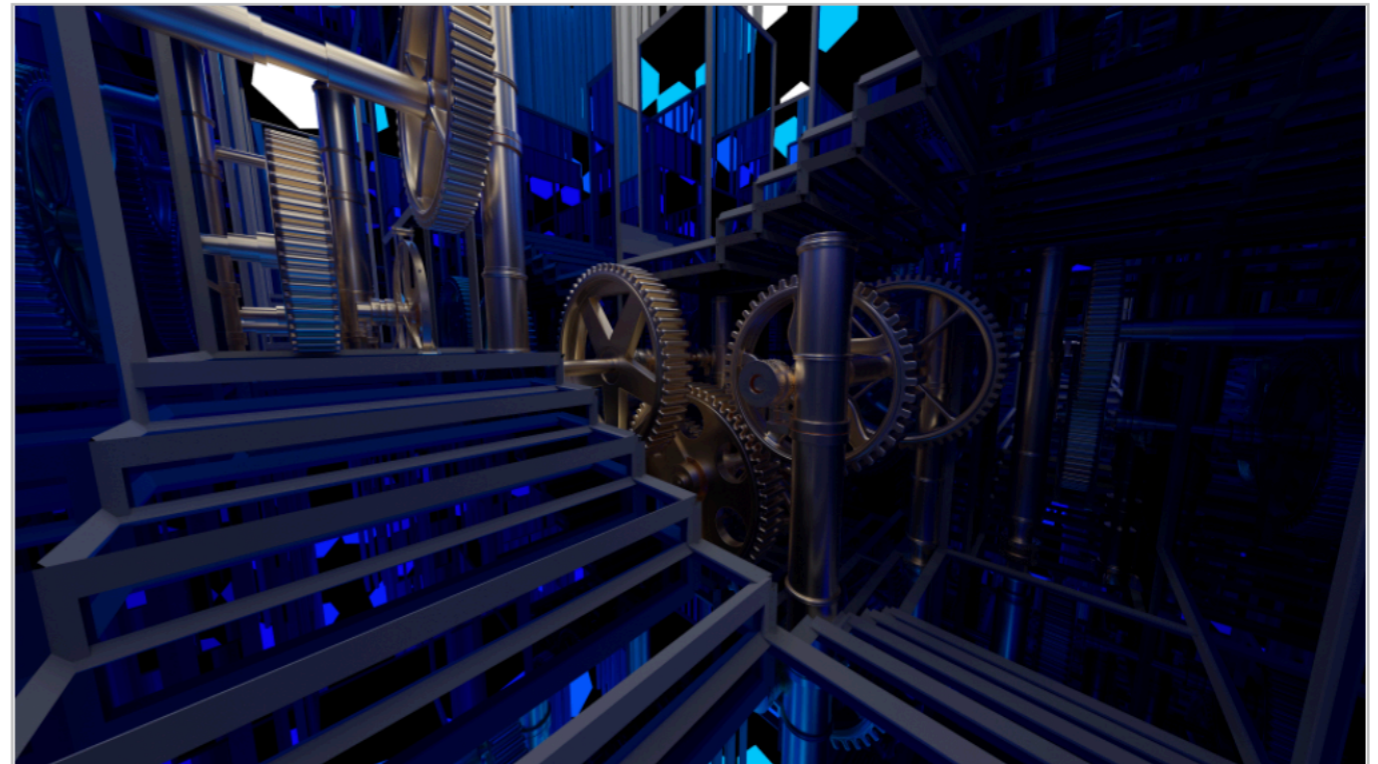
15-464/15-664 Technical Animation



Particle_Diam = 0.005
Compress = 1-19e-2
Stretch = 1+7.5e-3
Hardening = 5.0
Density = 400

🏆 **Technical award winner**

**15-468, 15-668, 15-868
Physics-based rendering,
Rendering competition**



Max Slater

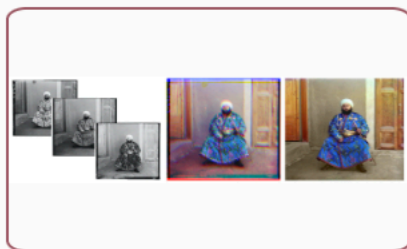
🏆 **Art award winner**



Arpit Agarwal

http://graphics.cs.cmu.edu/courses/15-468/2021_spring/

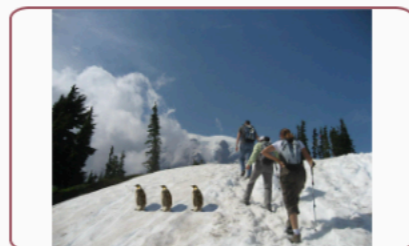
16-726 Learning-Based Image Synthesis



Assignment #1 - Colorizing the Prokudin-Gorskii Photo Collection

Winner: [Riyaz Panjwani]

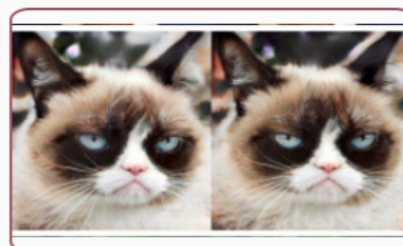
Honorable Mentions: [Harry Freeman]



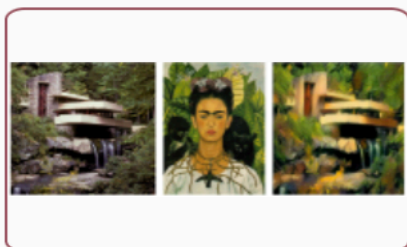
Assignment #2 - Gradient Domain Fusion

Winner: [Riyaz Panjwani]

Honorable Mentions: [Tomas Cabezon Pedroso] [Harry Freeman]



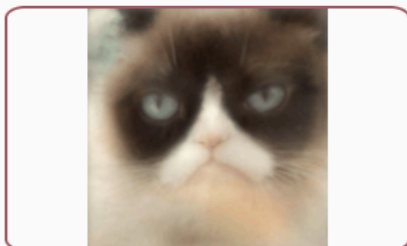
Assignment #3 - When Cats meet GANs



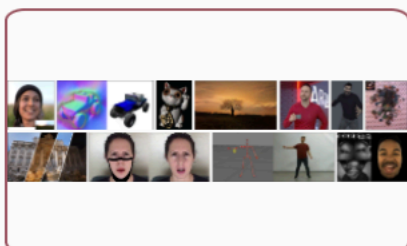
Assignment #4 - Neural Style Transfer

Winner: [Lena Du]

Honorable Mentions: [Yutian Lei] [Riyaz Panjwani] [Sean Chen]



Assignment #5 - GAN Photo Editing



Final Project

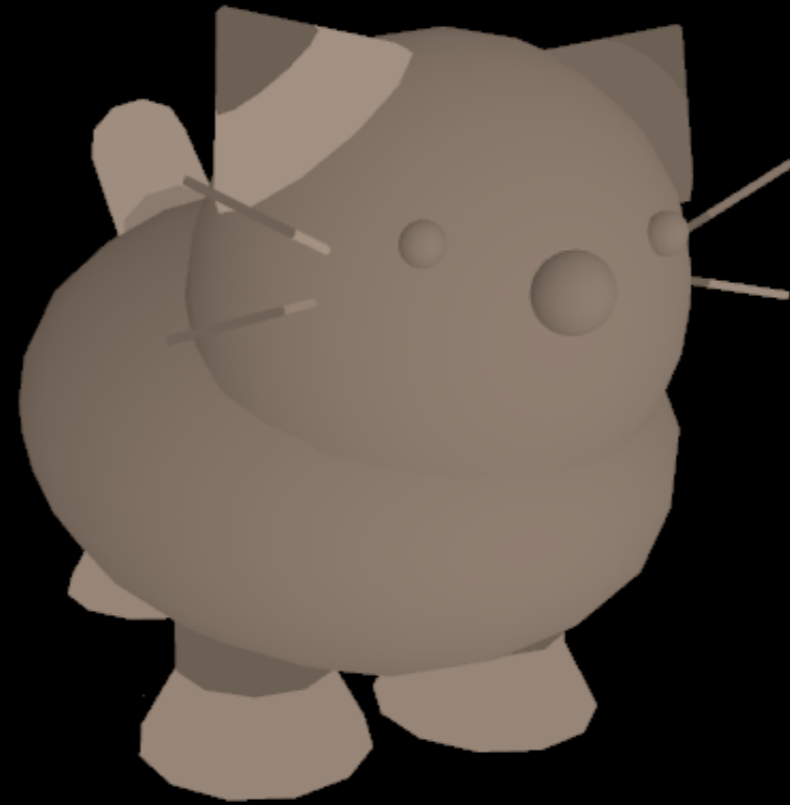
<https://learning-image-synthesis.github.io/sp22/>

Graphics Concentration

Your projects!

A1

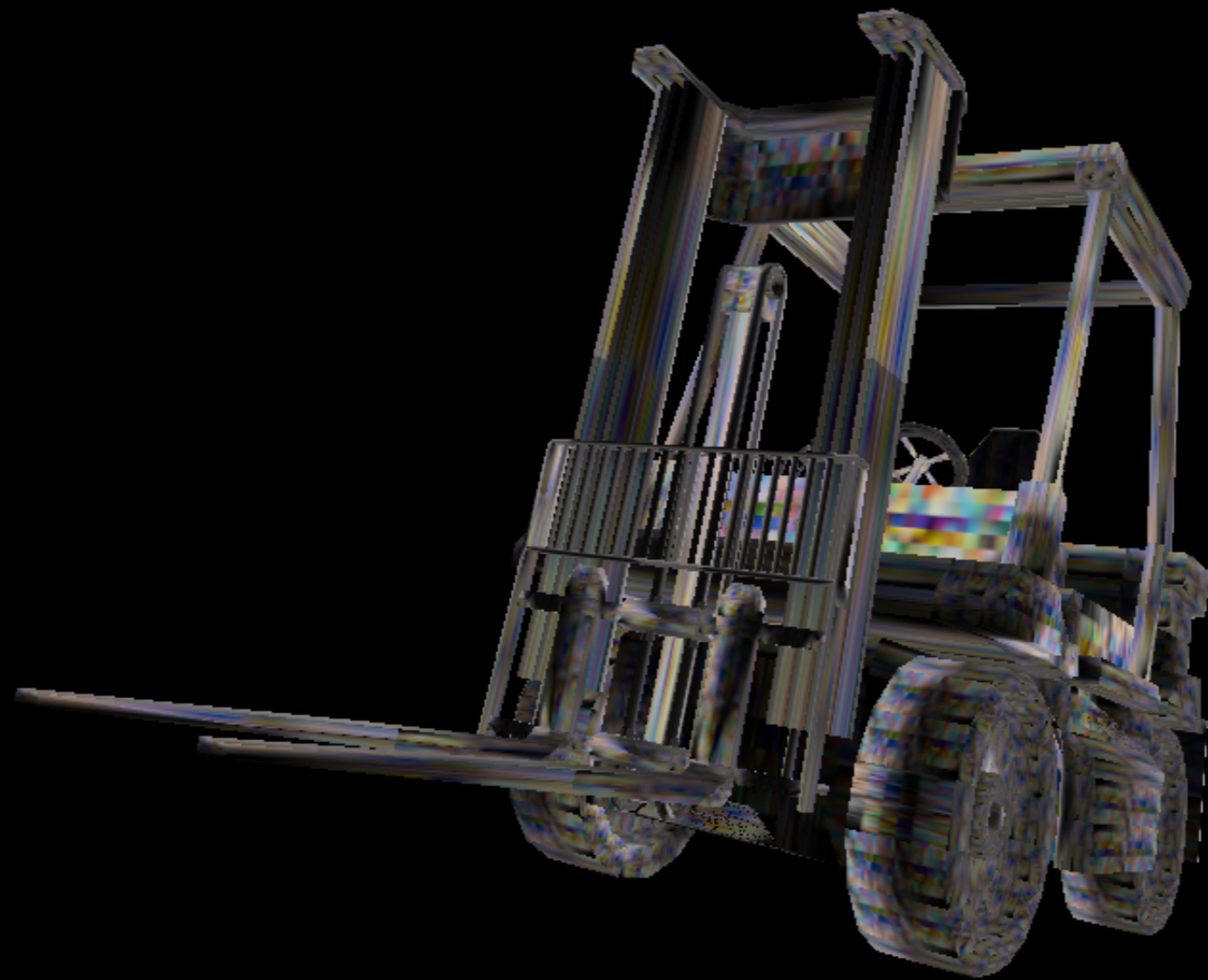
Livia Lai



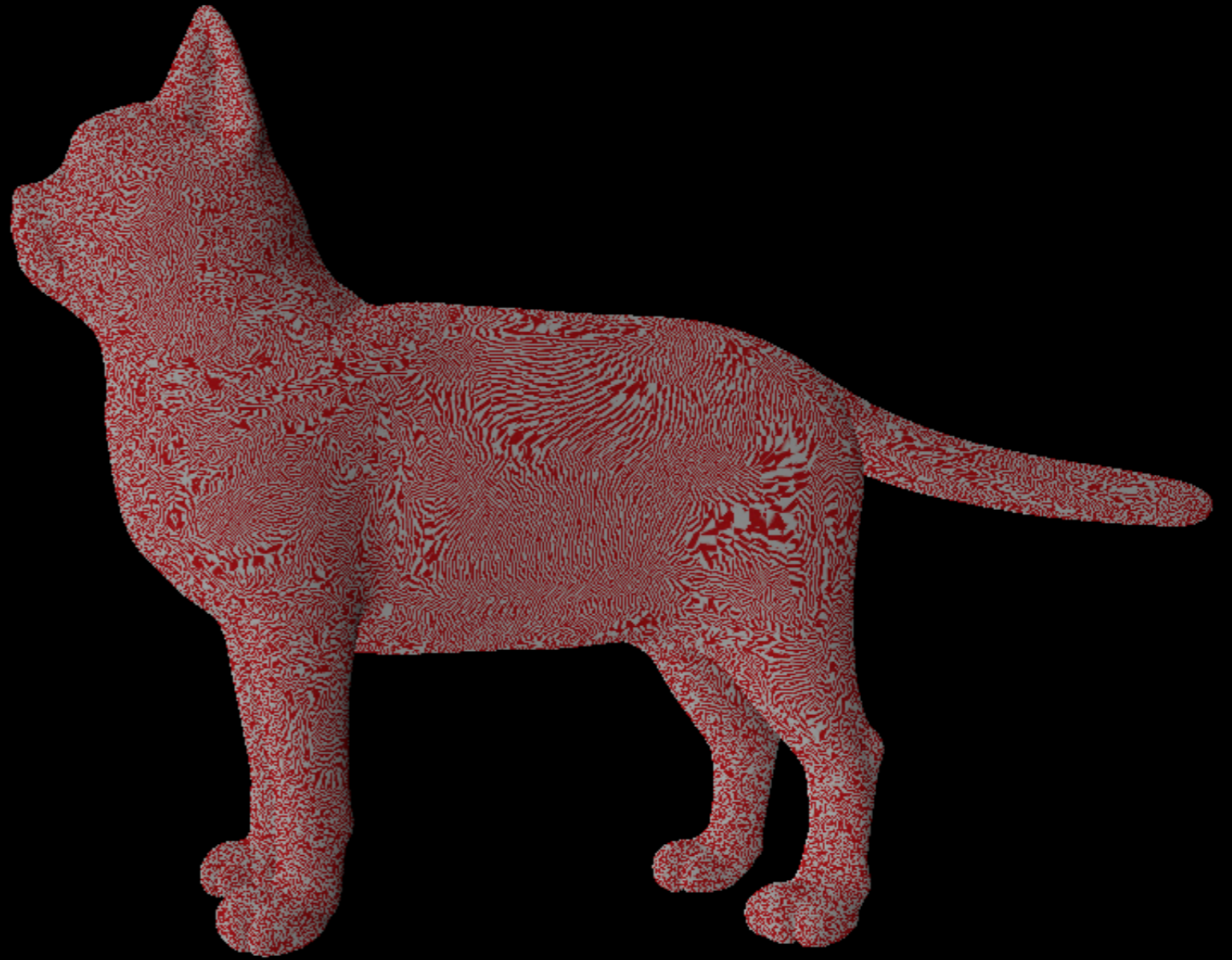
Livia Lai



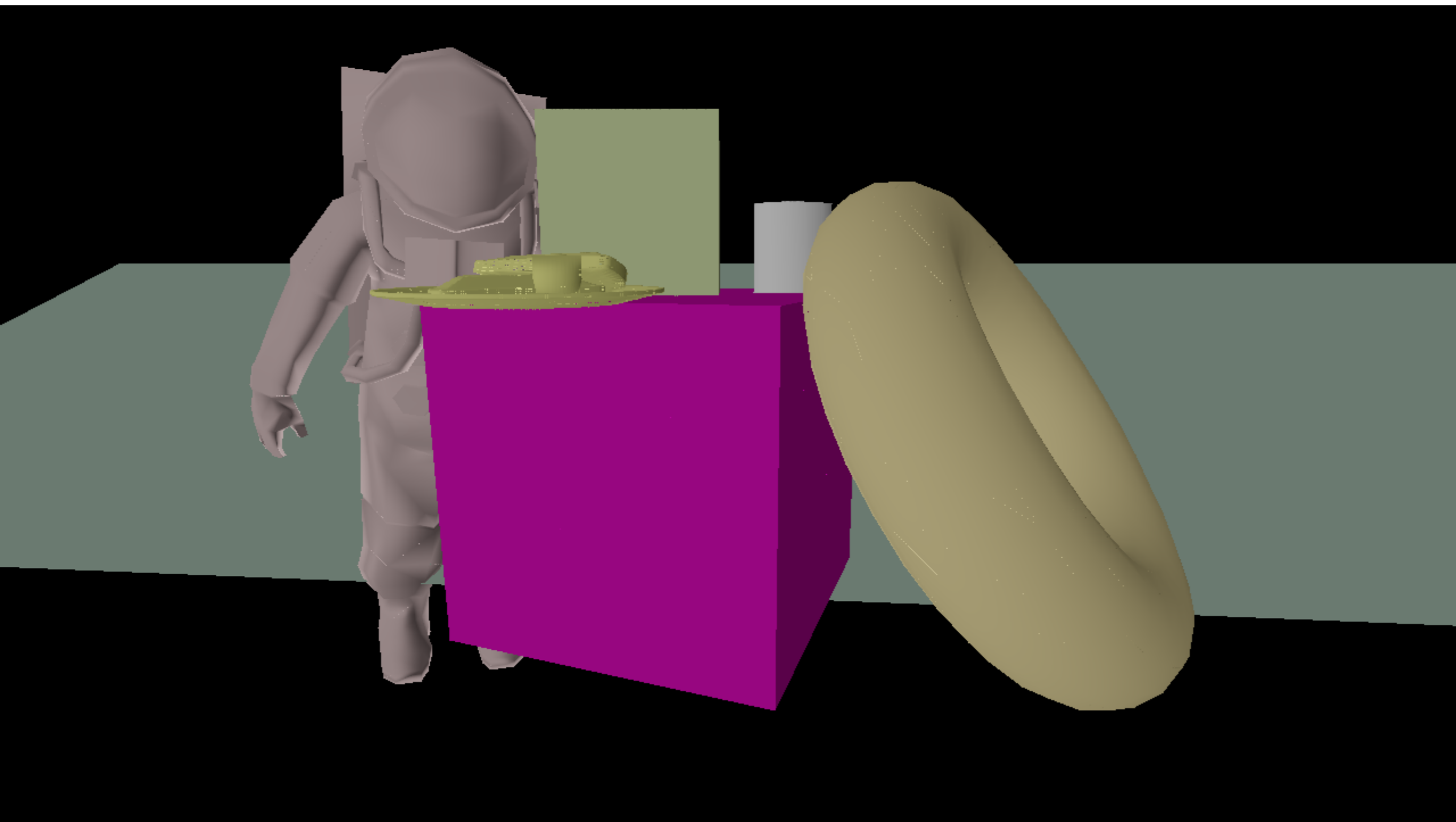
Andy Jiang



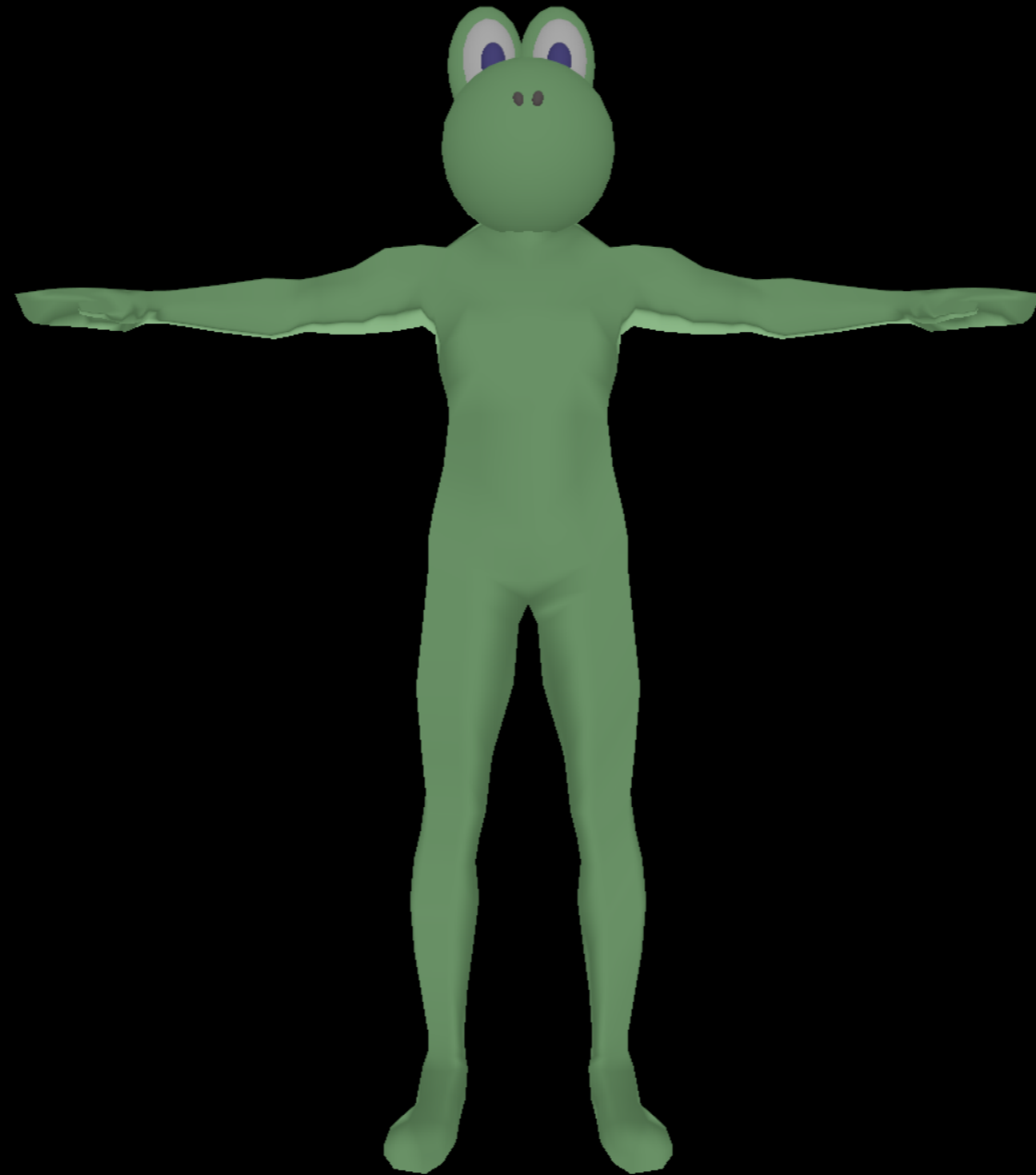
Ryan Lau



Jack Liu



Angus Koon Yan Yiu

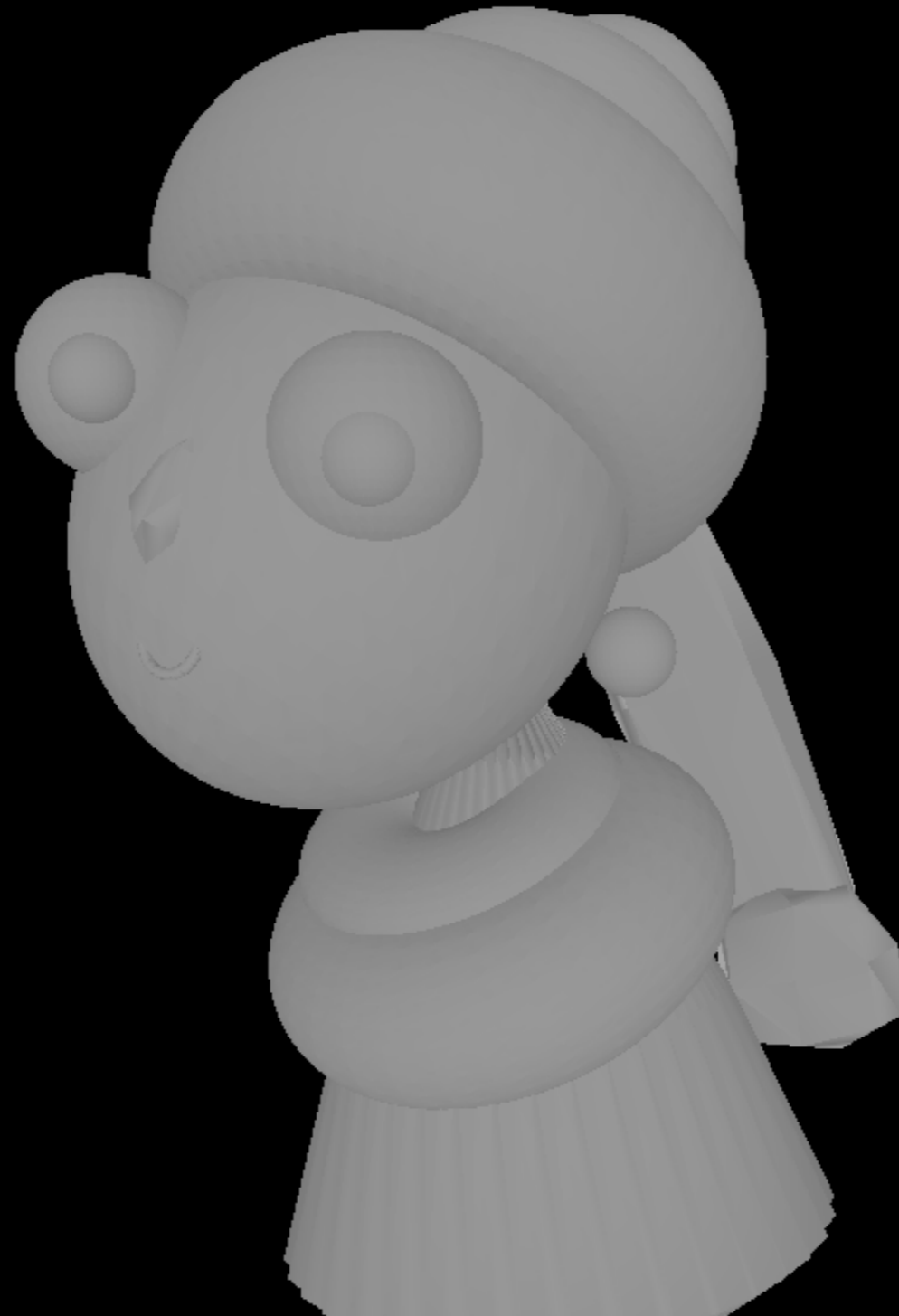


Dazhou Hou

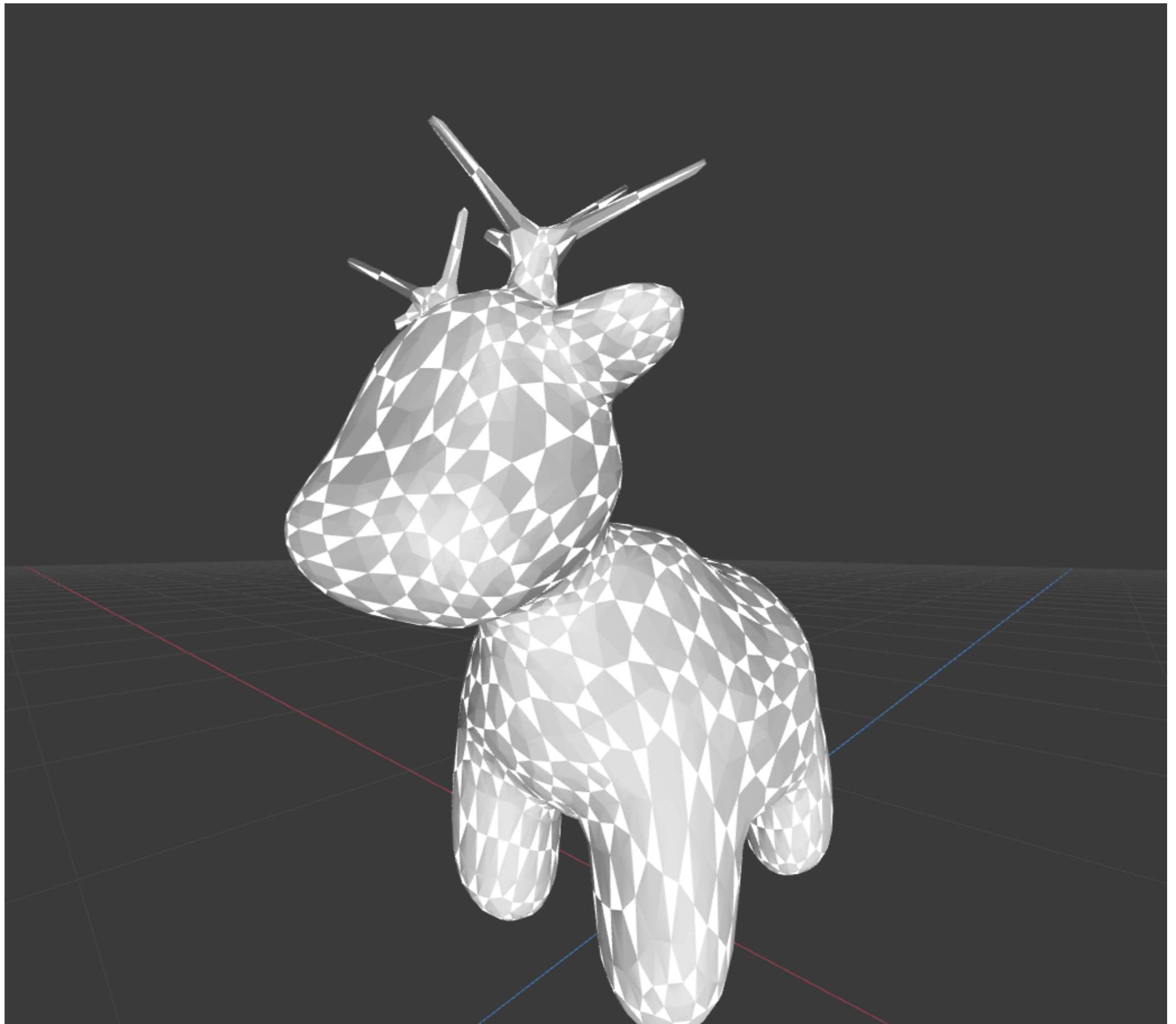


A2

??



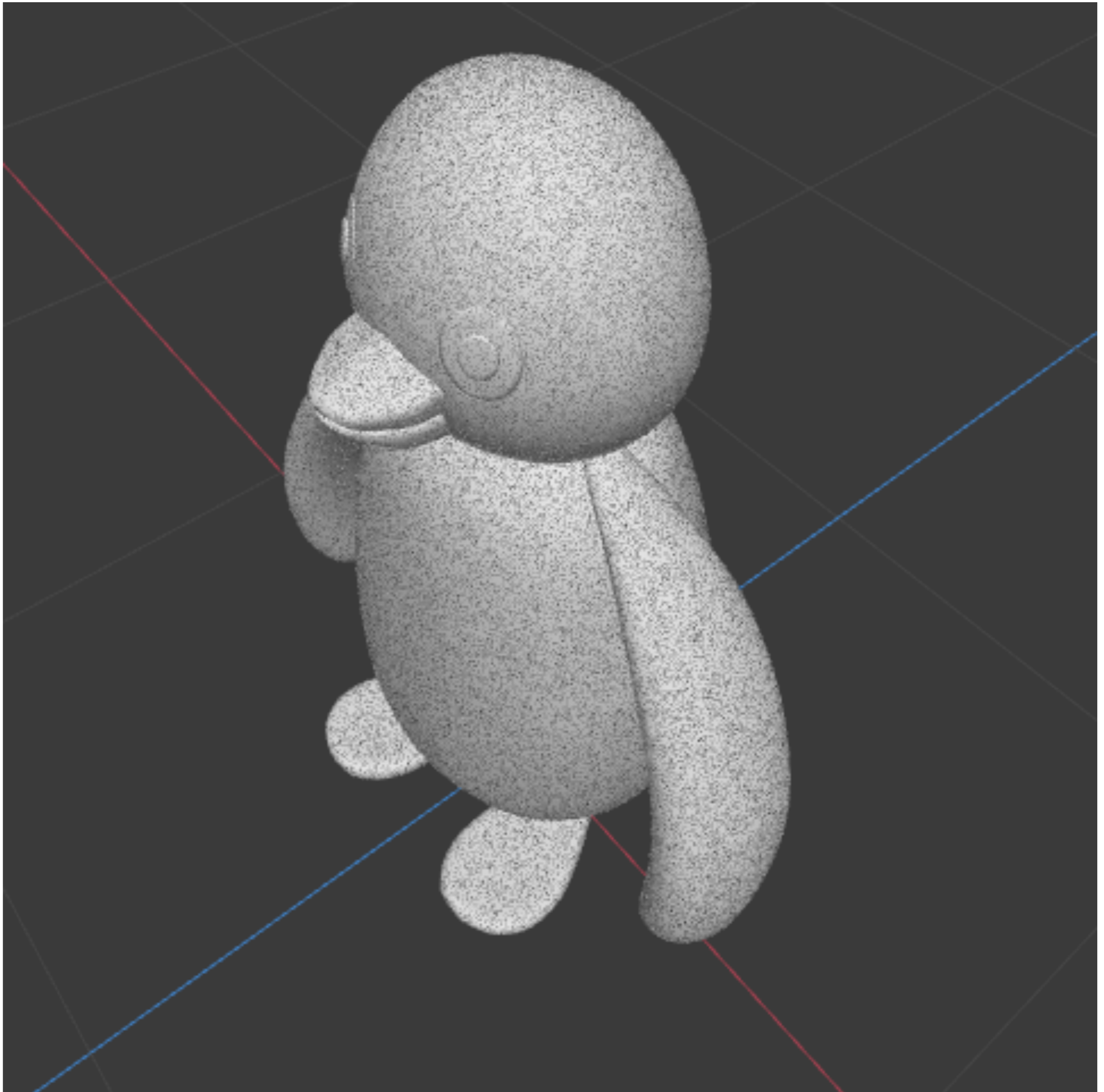
??



??

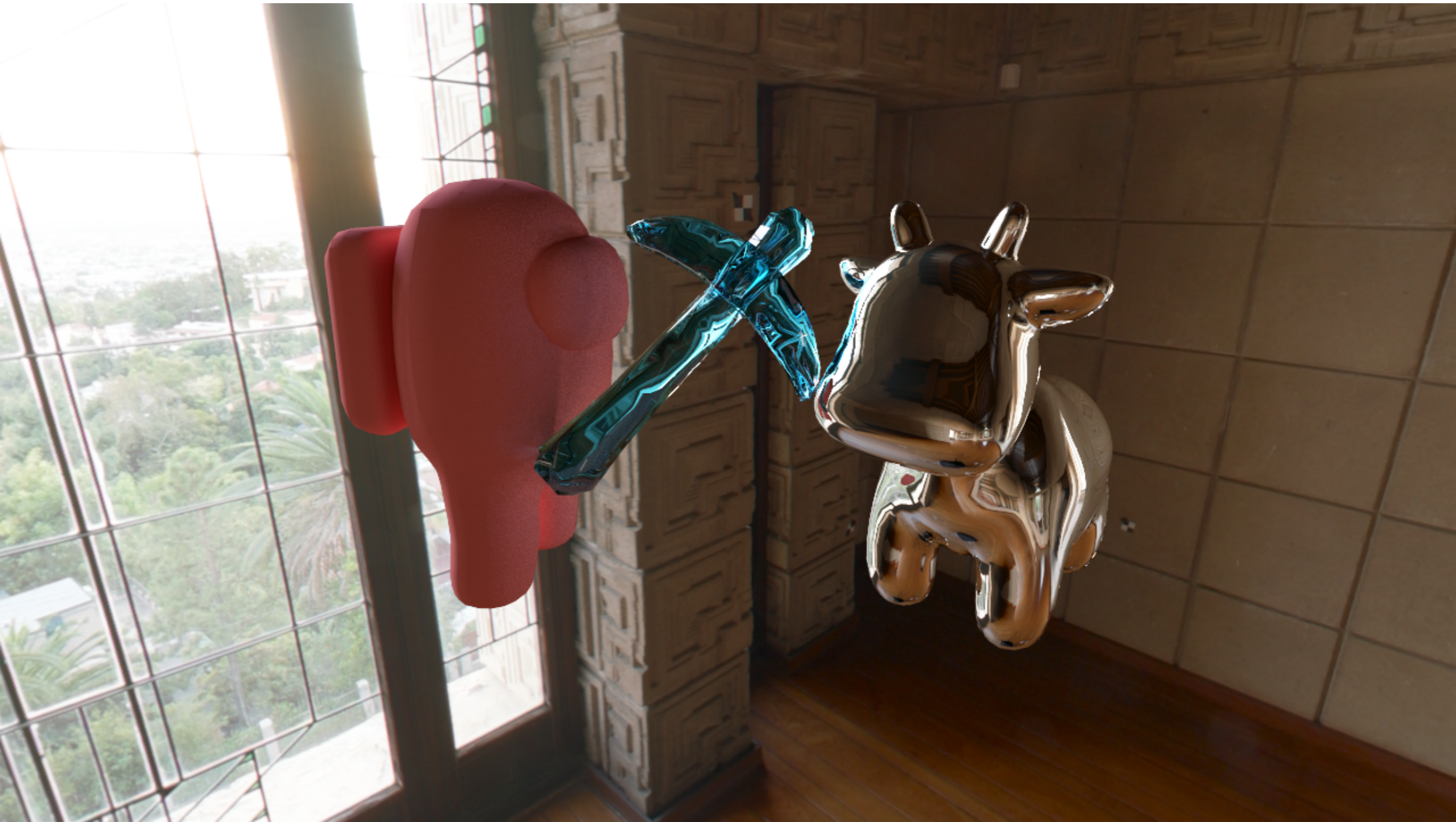


??

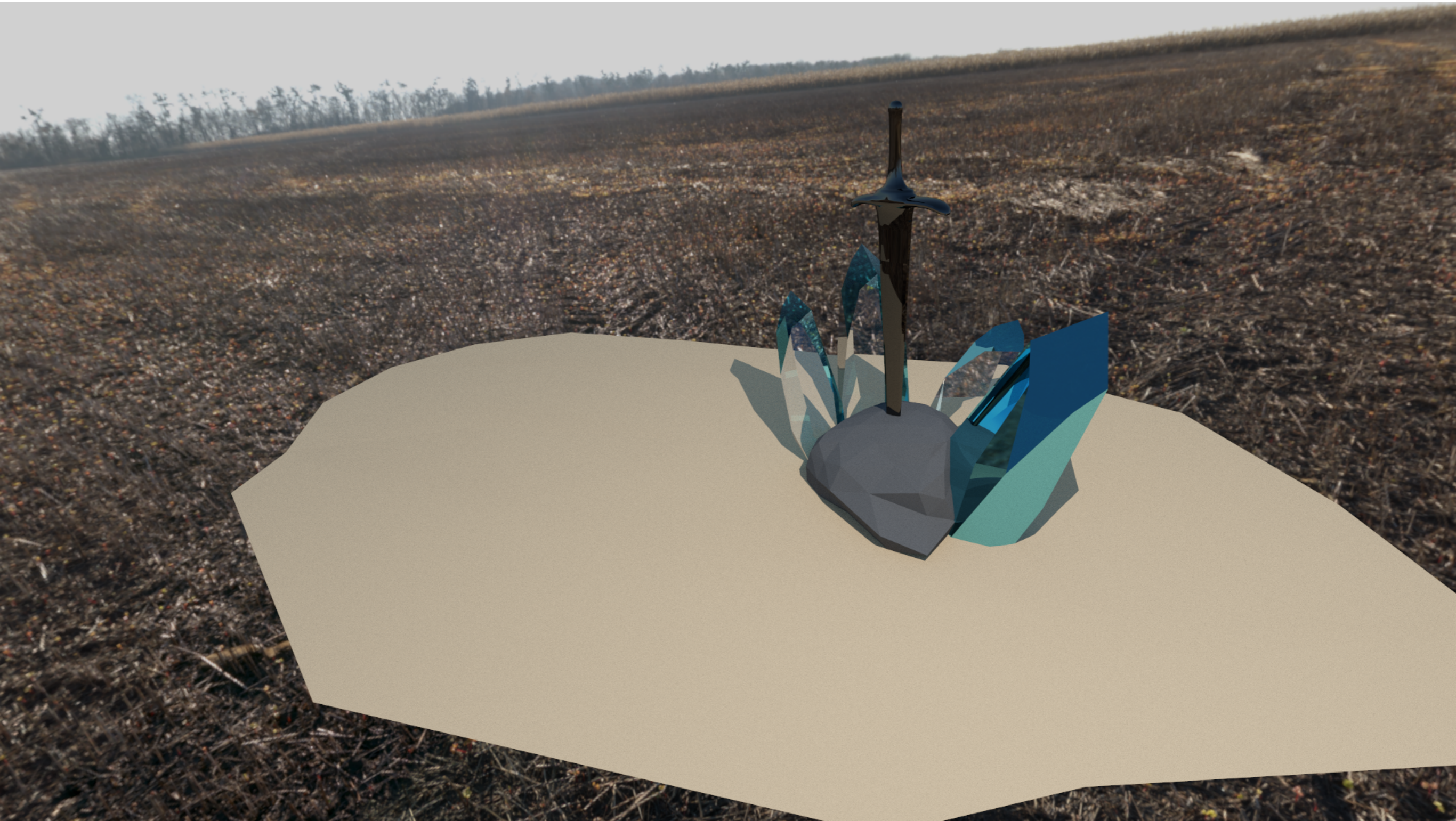


A3

Steven Lee



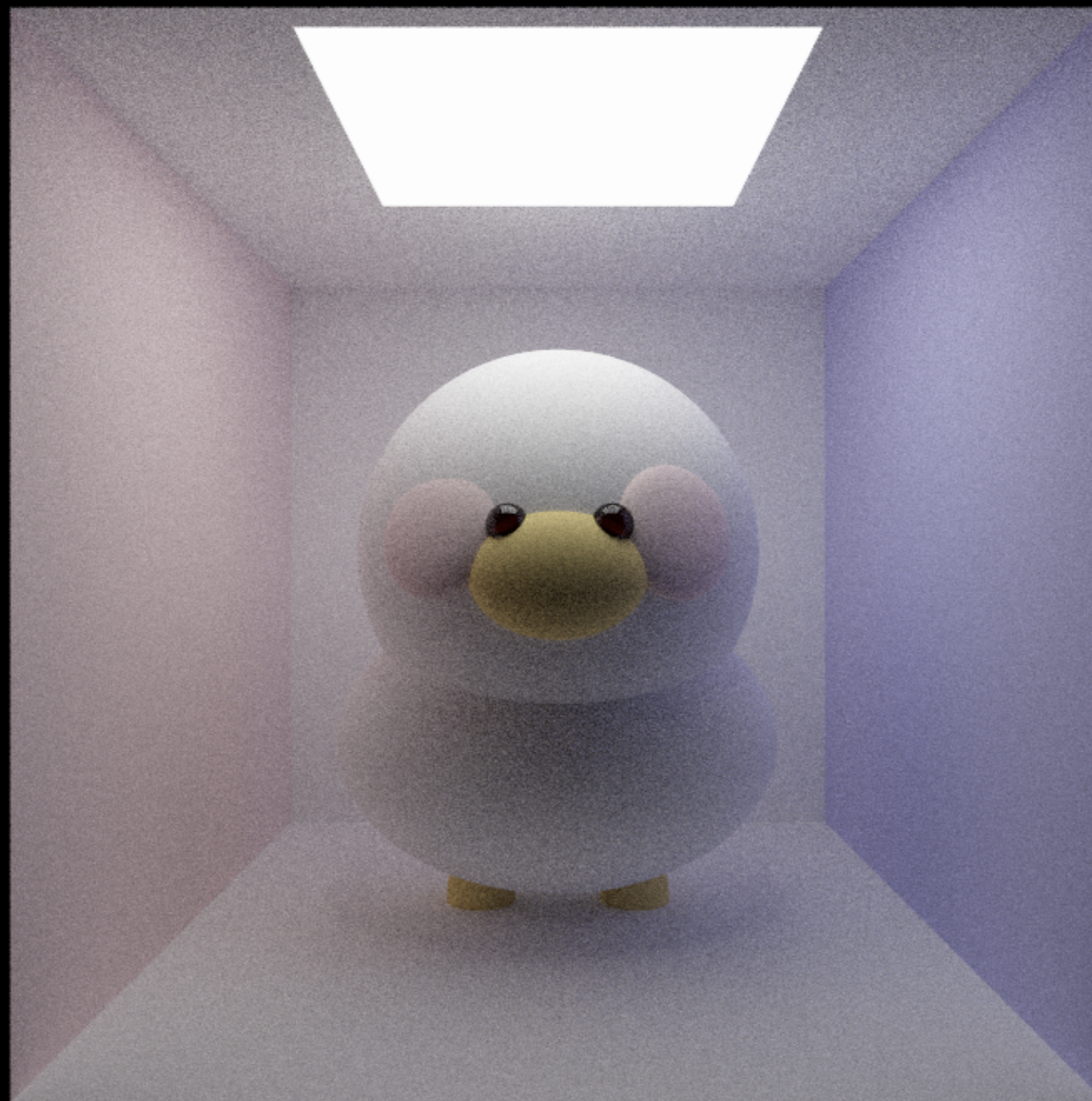
Xun Zhang



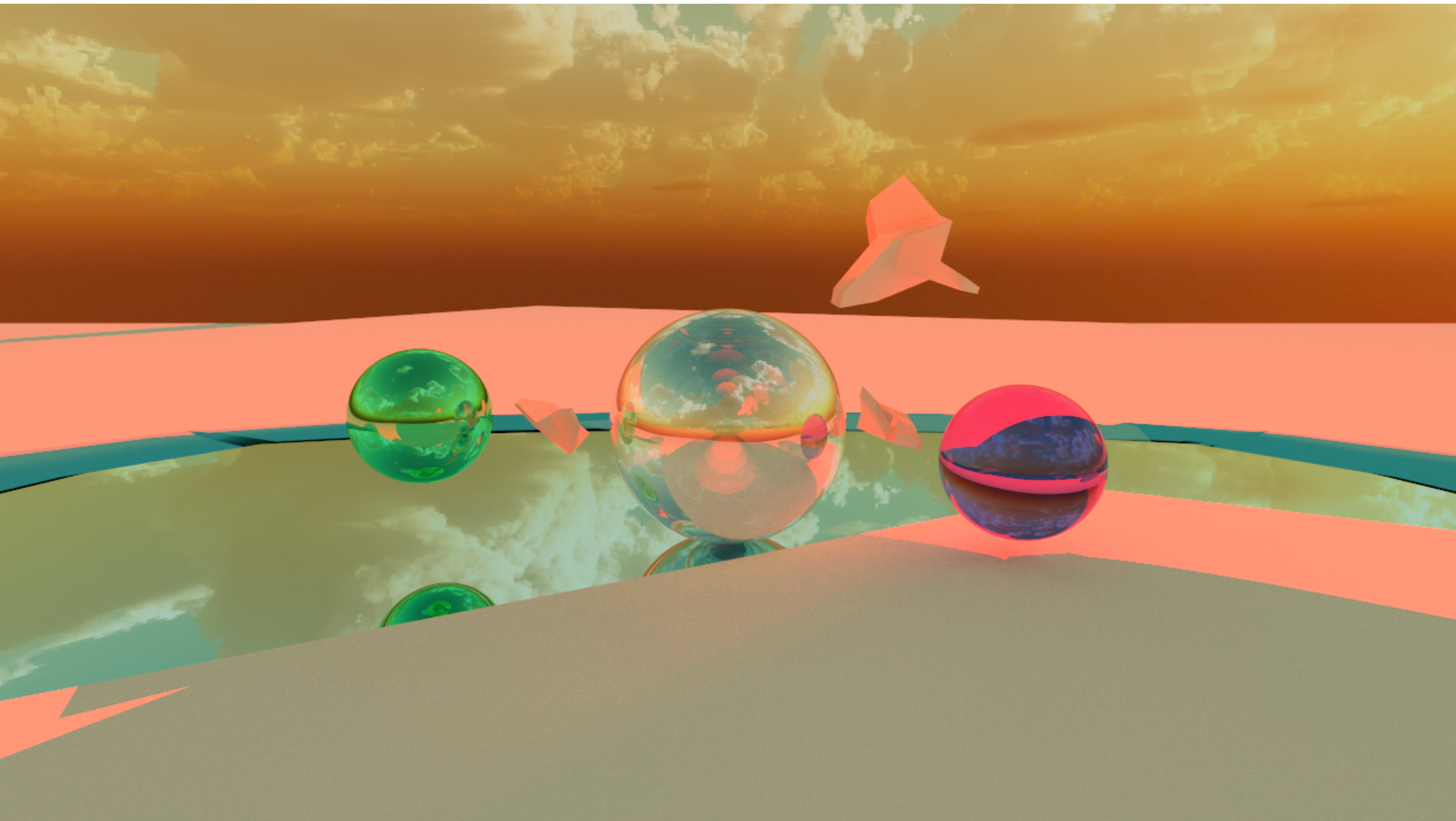
Raymond Luo



Taylor Kynard



Emily Amspoker



Annie Li



A4

Thanks for being a great class!
See you at the final! (study hard, but don't stress too much)

