CS3451:

Awesome Journey

in Computer Graphics

Final Project Brainstorm

Bo Zhu

School of Interactive Computing

Georgia Institute of Technology

Motivation

- All the images/animations I show in this lecture are from final projects done by students in the previous years
- I use them to introduce visual rendering techniques and artistic designs that might motivate your final project
- Let these images kick-start your **brainstorming** and serve as a source of **inspiration**. They're not meant to be requirements or constraints, so don't let them limit your imagination and creativity
- There are always more than one way to implement graphics effects
- Think about your own story, pick the right techniques, and implement them with your own customization
- Be creative; be brave to try new things ☺

Some words about failure exploration:

Go ahead and try out new things, and don’t worry if it doesn't work out the first time.

This picture was done by a student exploring smoke rendering for the final project. It might not look the best and the algorithm doesn’t work perfectly, but we gave it full marks to acknowledge his achievements, because this project is aiming high and exploring something cool and new.

Skybox

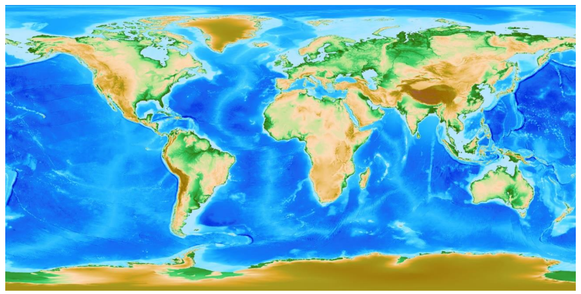
Technical Contribution: Sky Bounding Volume

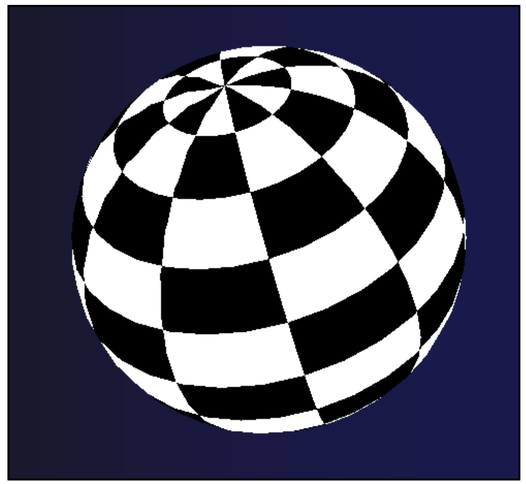
- Sky bounding volume is the most popular technique contribution that almost every team employs in their final project
- It is GPU low-cost and easy to implement, which can significant enhance the aethetic aspect of your rendering
- Two ideas to implement a sky bounding volume
  - Create a huge **sphere** that covers the entire scene and apply texture mapping onto its surface (recommended to implement)
    - Pros: Easy to implement, and easy to enable dynamic effects (by rotating the sphere)
    - Cons: May have artifacts in the poles
  - Create a huge **box** that covers the entire scene and apply texture mapping onto its surface
    - Pros: a lot of cubemap textures available online
    - Cons: requires extra implementation of OpenGL cubemap texture

Difficulty Level: ⭐

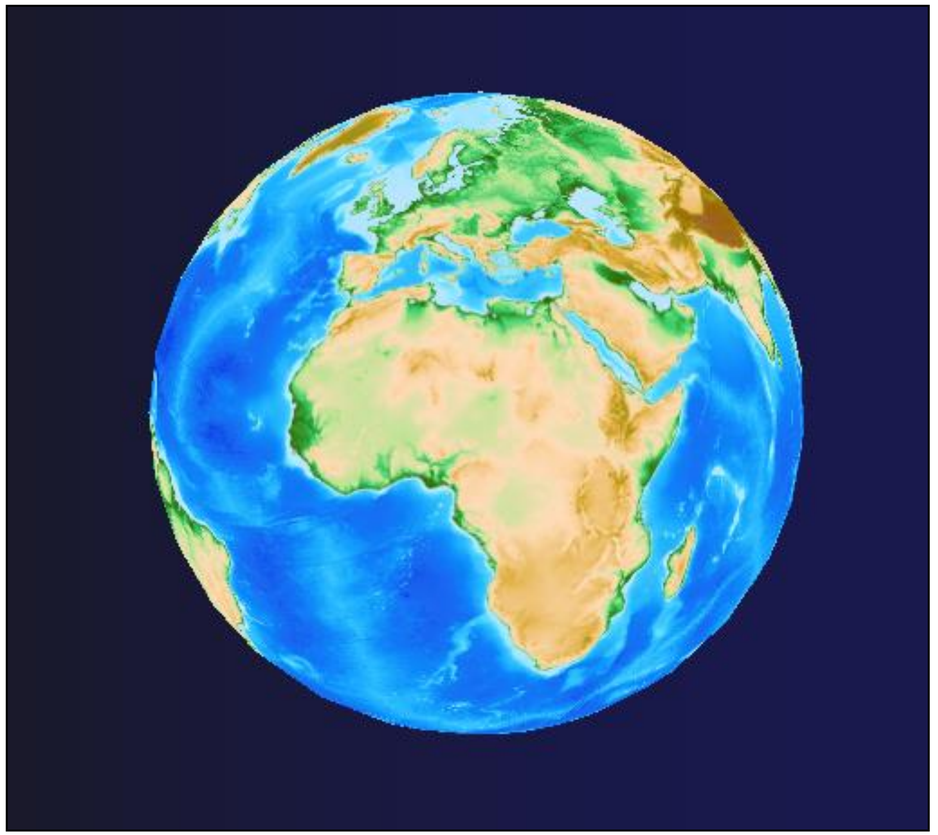
Sky Sphere: Implementing texture mapping on a sphere

- Map an image onto a sphere's surface with proper uv
- We will discuss this in details in future lectures

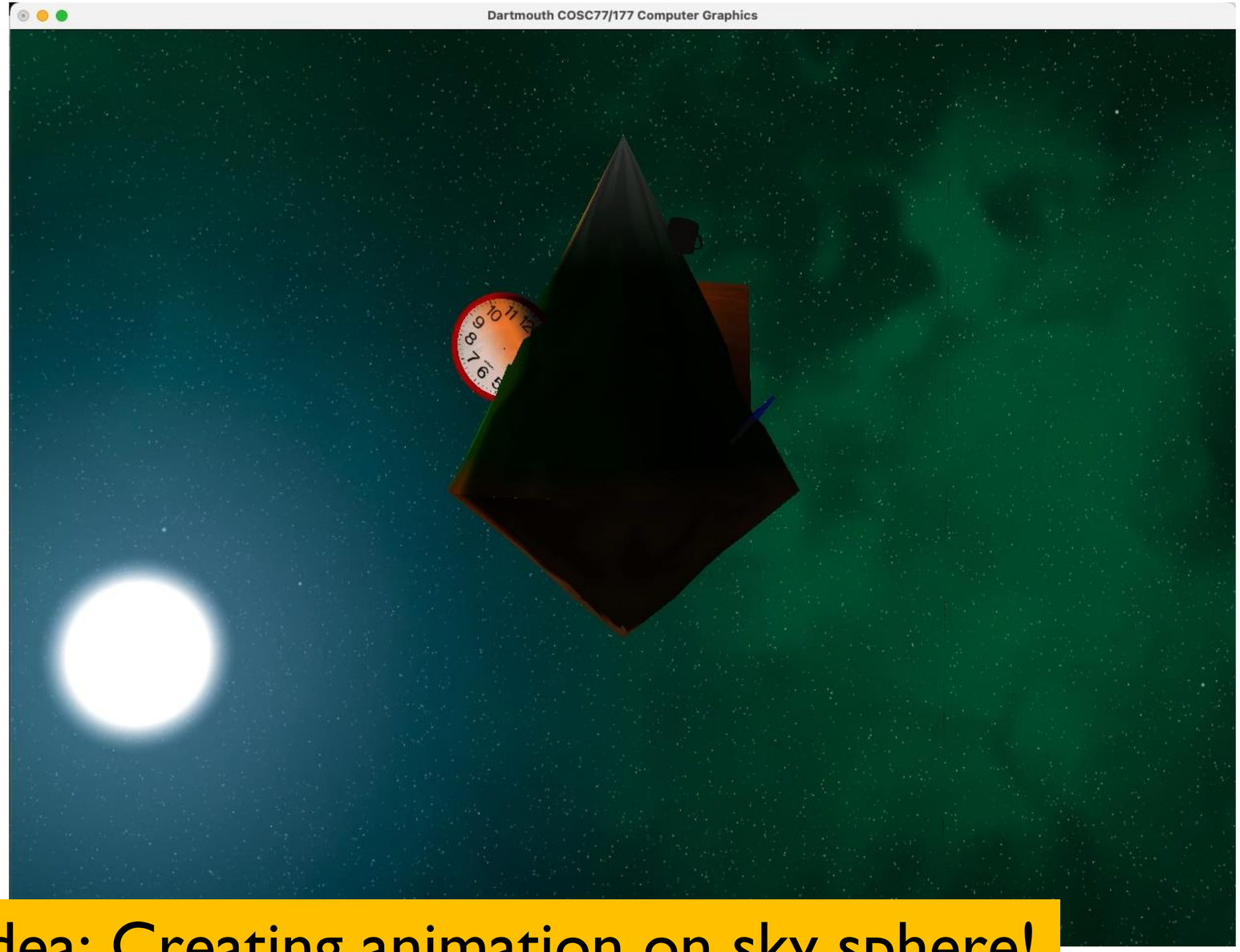




spherical uv mapping



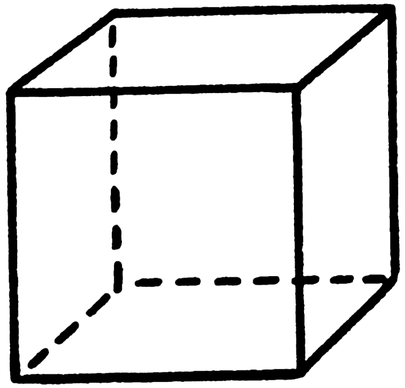
The observer looks from the sphere's interior

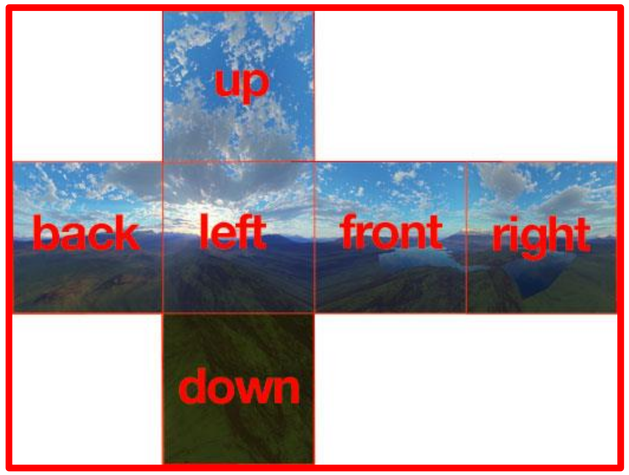


Awesome Idea: Creating animation on sky sphere!

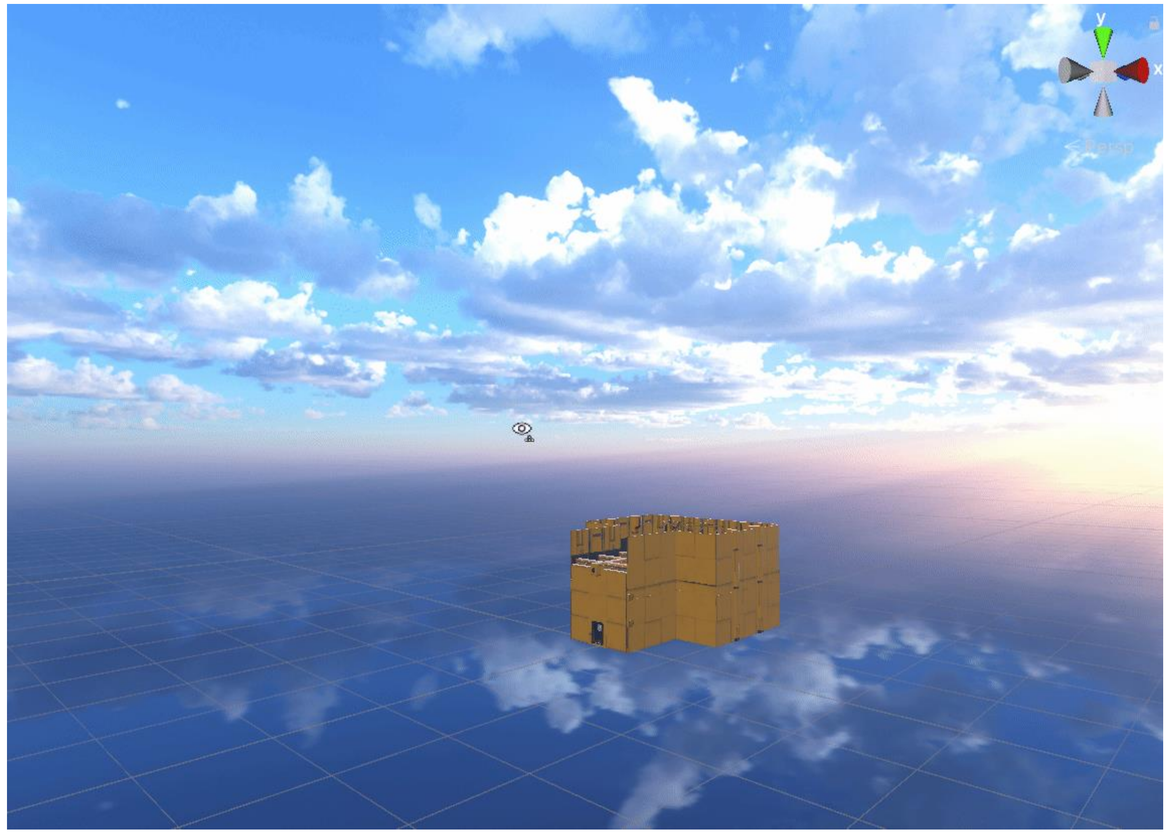
Skybox: Implementing texture mapping on a cube

- Map each facet of the box with an image
- We will discuss this in details in future lectures





cubic uv mapping



The observer looks from the cube's interior

Reading: https://learnopengl.com/Advanced-OpenGL/Cubemaps

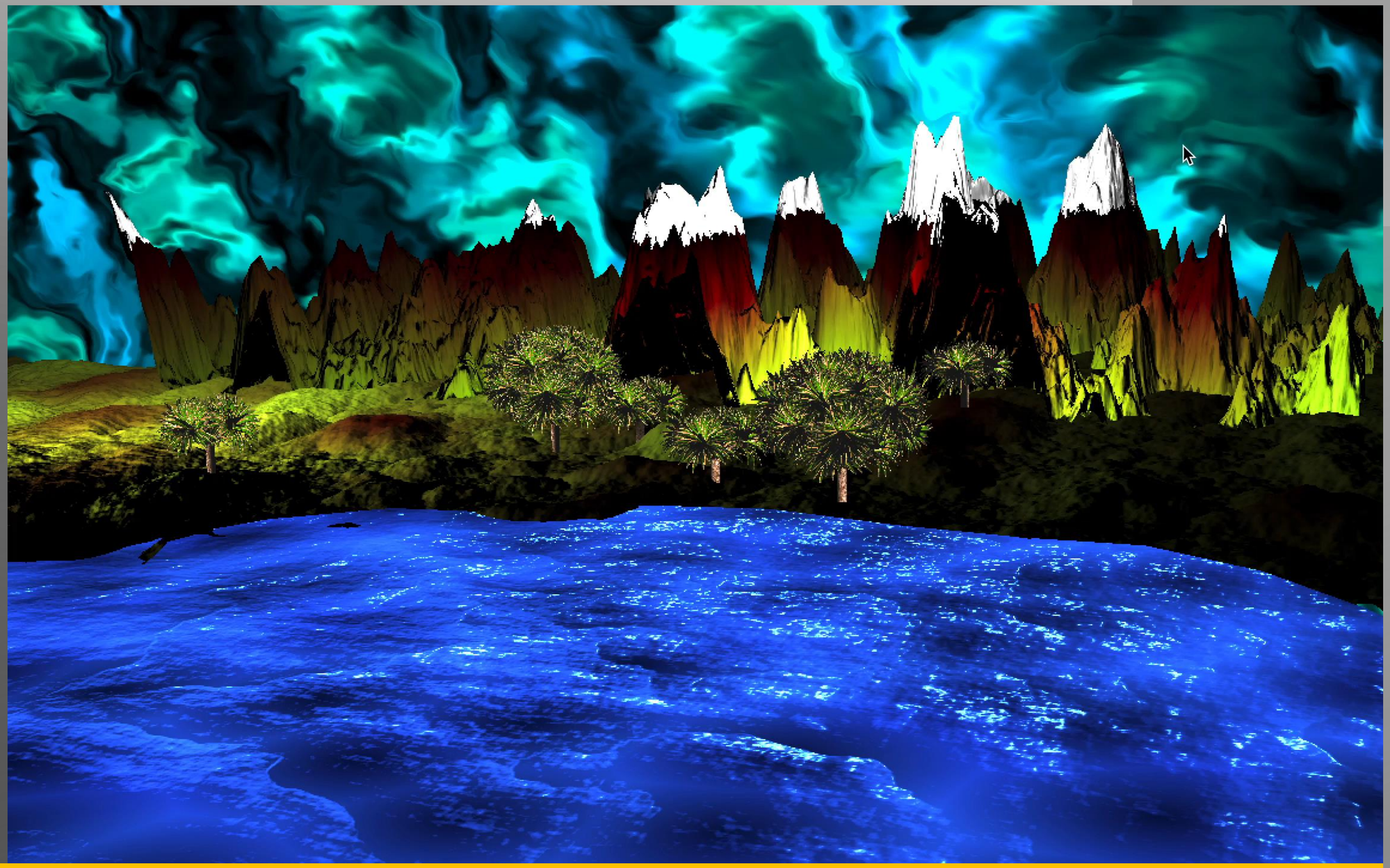
Skybox Texture

- A lot of free skybox texture models available on line
- Each texture contains six subimages for each face of the cube
- You need to be careful to ensure the consistency among different texture faces

Background Rendering

Background Rendering

- Another option to implement the background rendering is to render a giant plane on the background and put your animations/colors by implementing the plane's fragment shader
- This is the idea of many of our programming assignments, e.g., A1, A7 and A8
- This idea is particularly suited for cosmic scenes, e.g., by rendering the galaxy, starry sky, etc.



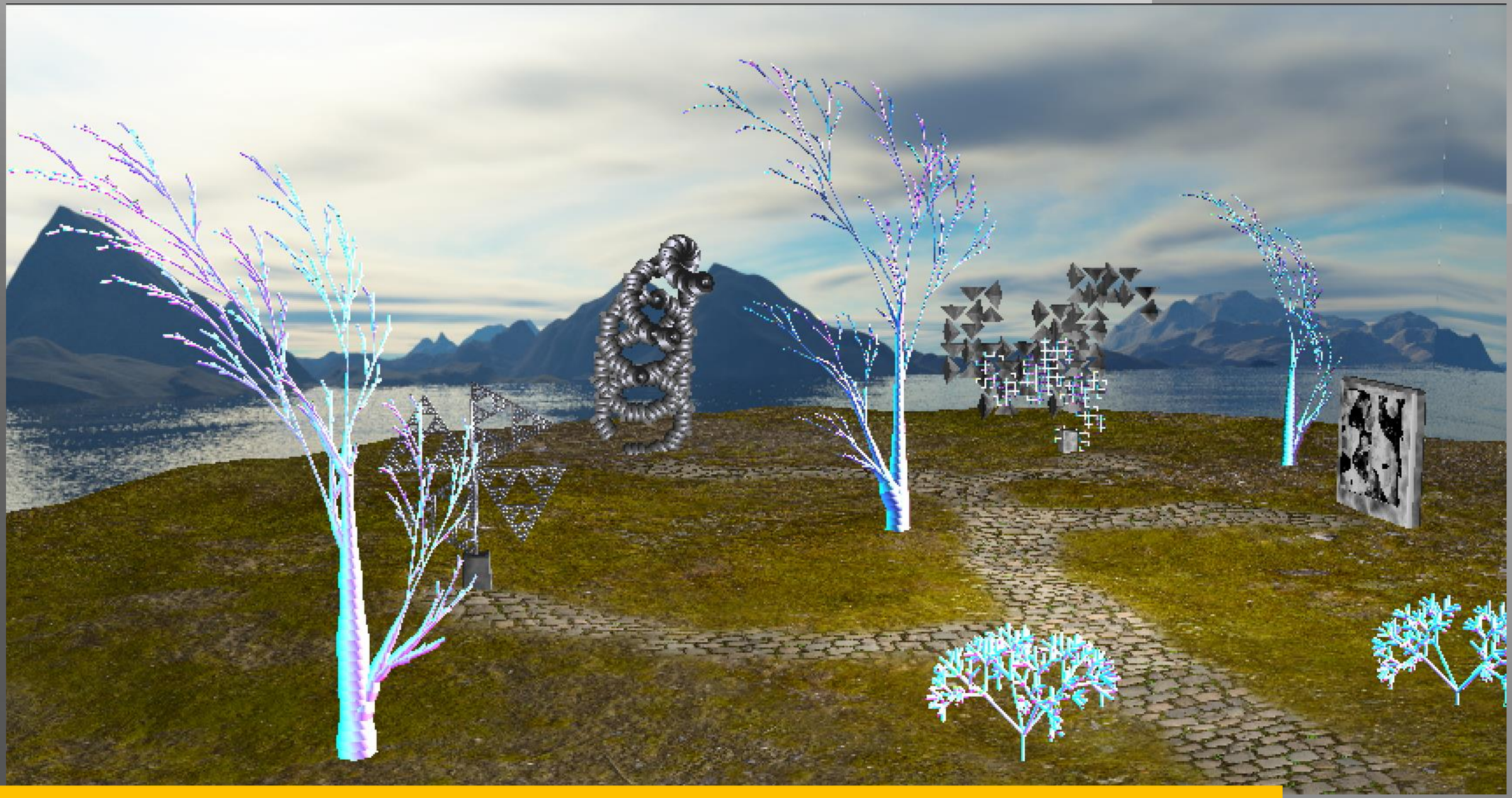
Awesome Idea: Implementing a noise function on the background!

L-system

Technical Contribution: L-system

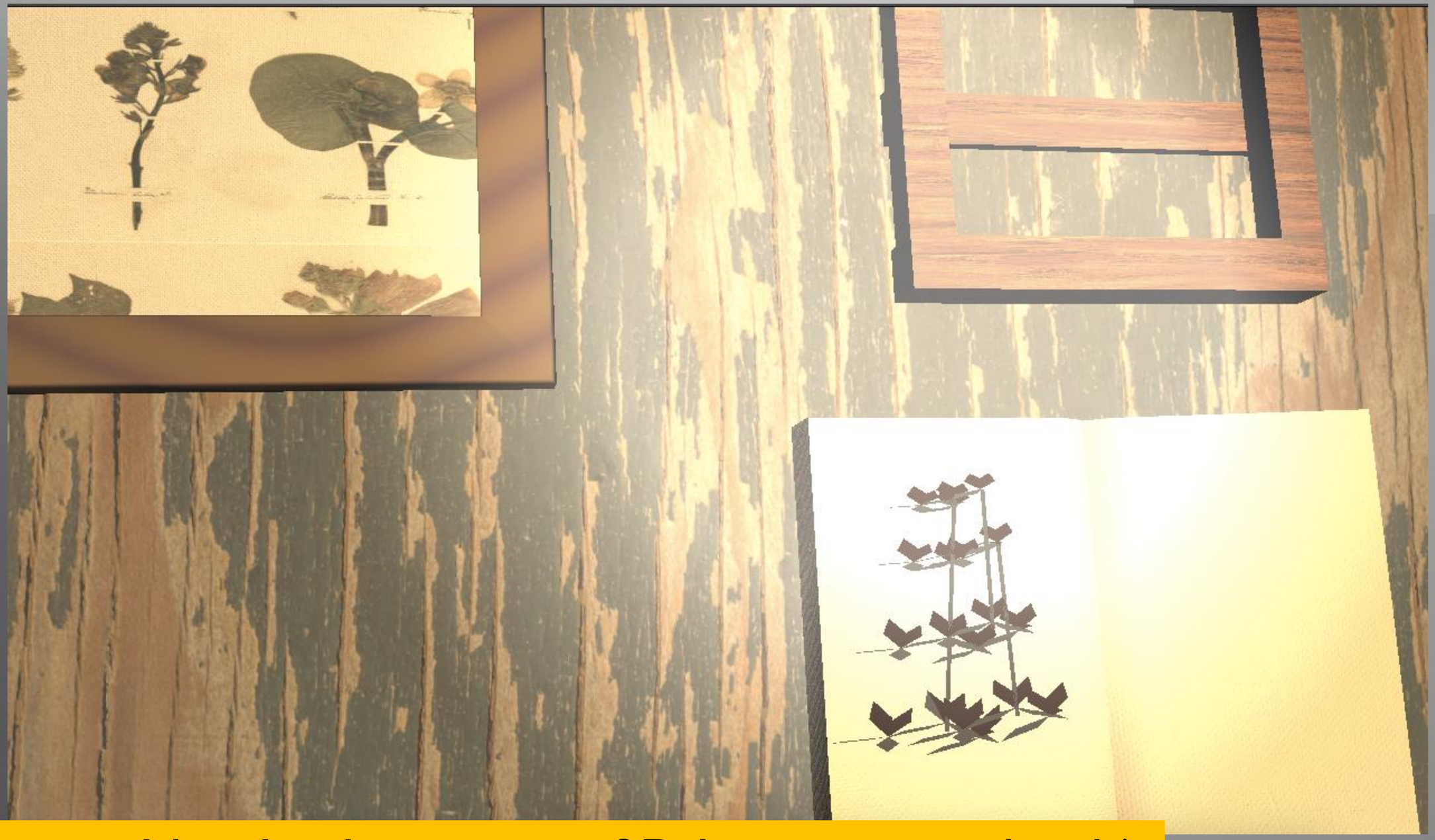
- L-system is another popular topic: teams chose to implement procedural algorithms to generate meshes for plants
- Two key problems to address:
    - An L-system grammar parser that takes rules as input and generates **strings** as output
    - A string-to-mesh procedure that translate the string representation to a **triangle mesh** that can be rendered in GLSL
    - If you choose to implement a 3D L-system, you need to figure out how to create a cylindrical mesh based on the two endpoints

Difficulty Level: ⭐⭐



Awesome Idea: Render a 2D L-system in a sculpture garden!

Awesome Idea: Render a 2D L-system as a background!



Awesome Idea: Implementing a 2D L-system in a book!

Some teams bravely take the challenge of rendering a 3D L-system.

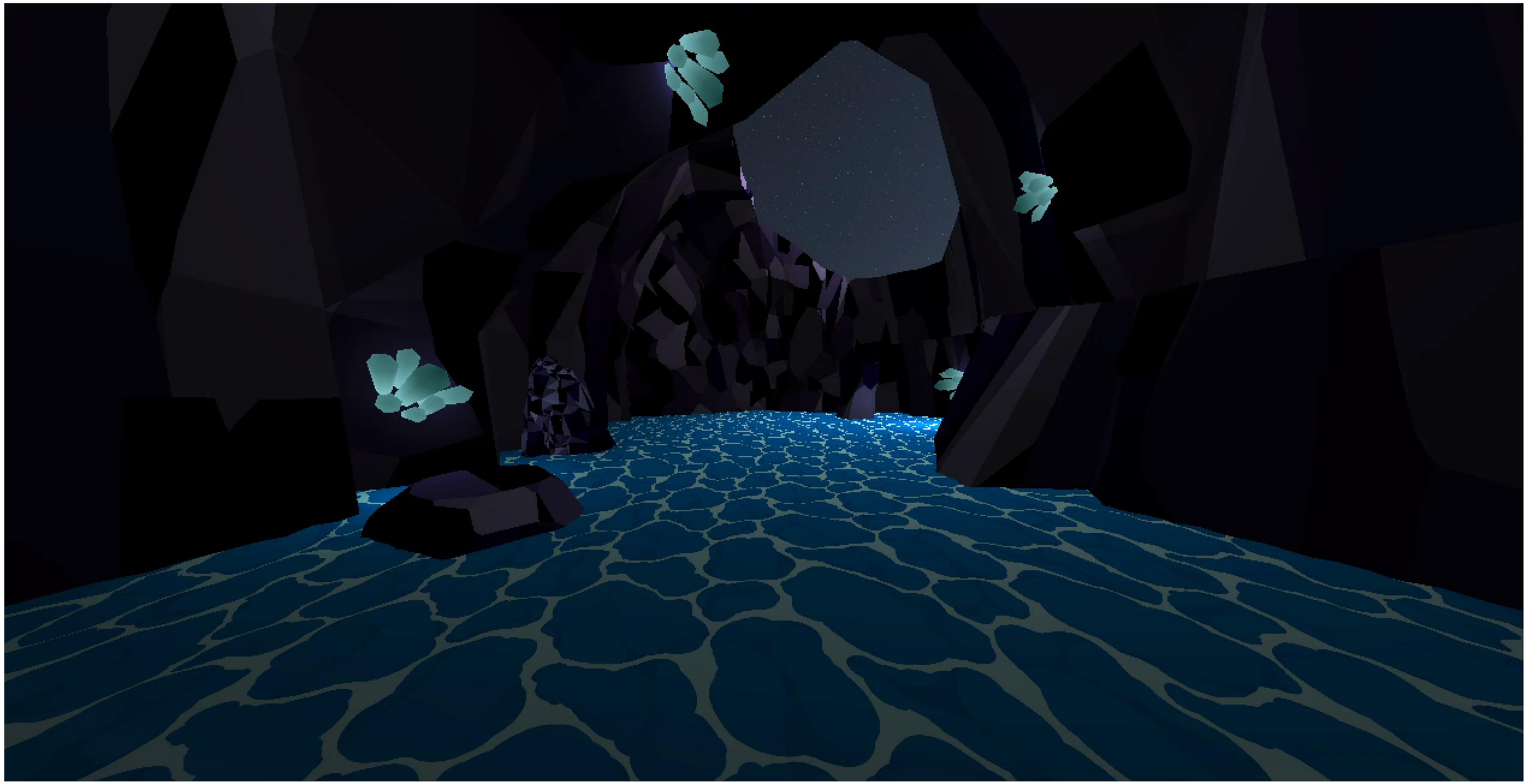
Some teams focused on creating a robust L-system tree generator.

Noise

Technical Contribution: Noise Texture

- Many teams chose to implement new noise functions other than Perlin noise to broaden the scope of procedural modeling from mountain terrains to other natural phenomena
- Worley noise (Voronoi) and fBM are some popular ones
- These noise functions are used to generate water wave, terrain on a planet, desert, etc.
- Noise-based animation is also a popular effect that teams like to incorporate to enhance the dynamic effects
- Some advanced noise features, such as caustics, are also explored by some pioneering teams in animating underwater scenes

Difficulty Level: ⭐⭐



Awesome Idea: Worley Noise for water surface

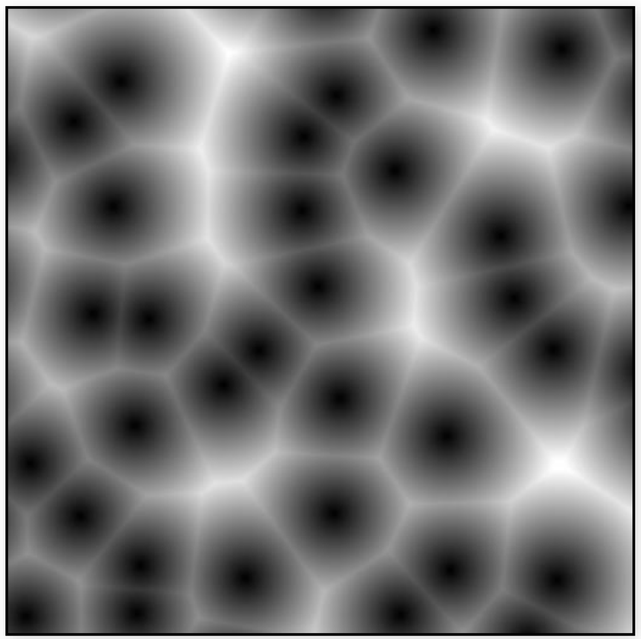
Worley Noise

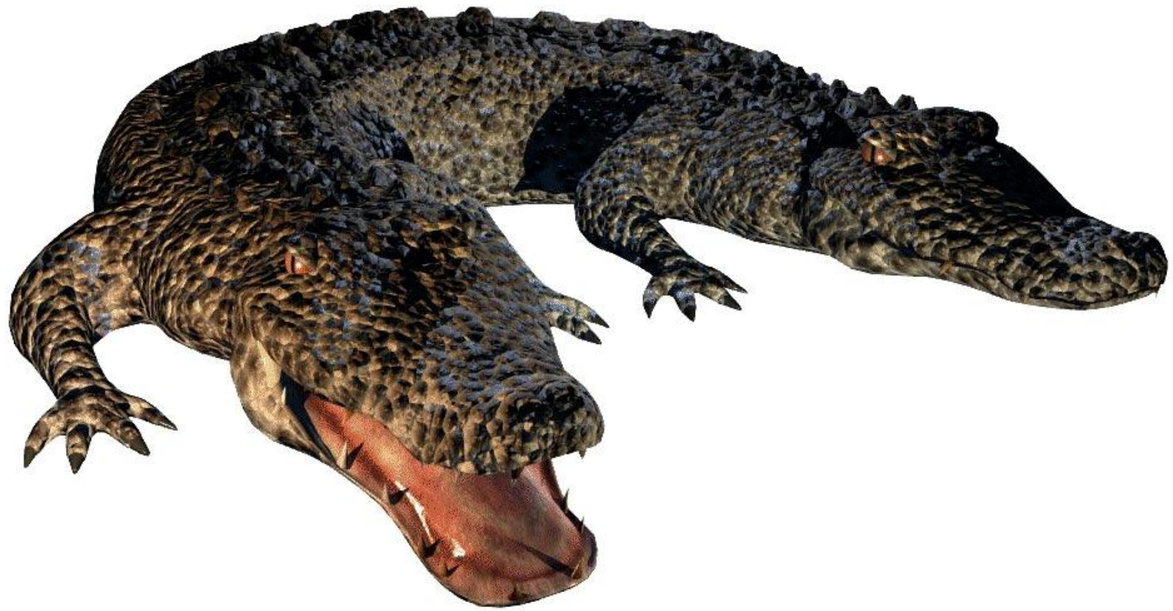
- "Cellular texture" function

  Introduced in 1996 by Steve Worley

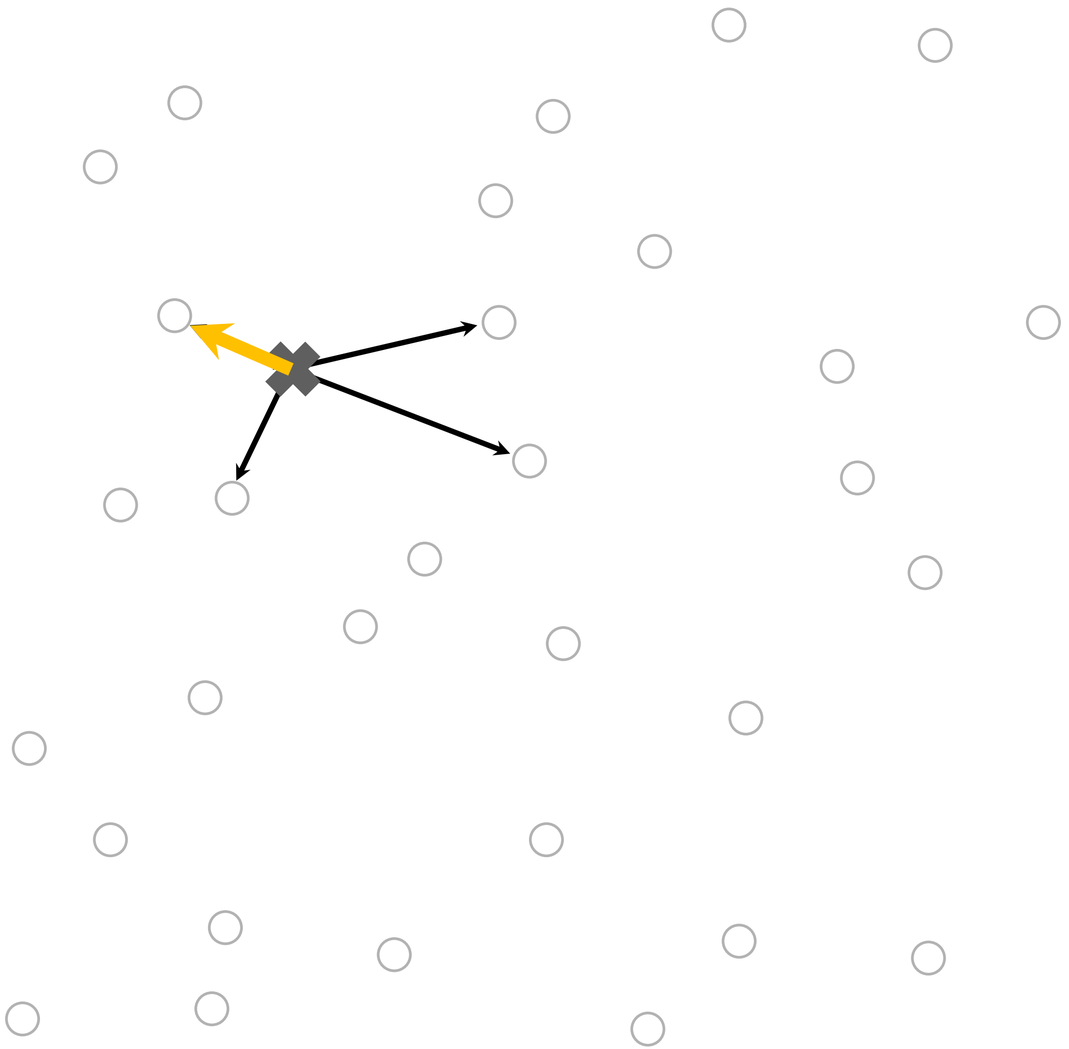
- Randomly distribute "feature points" in space

  $f_n(x)$ = distance to n^{th} closest point to x





2D Worley noise: f1



2D Worley Noise: f_1

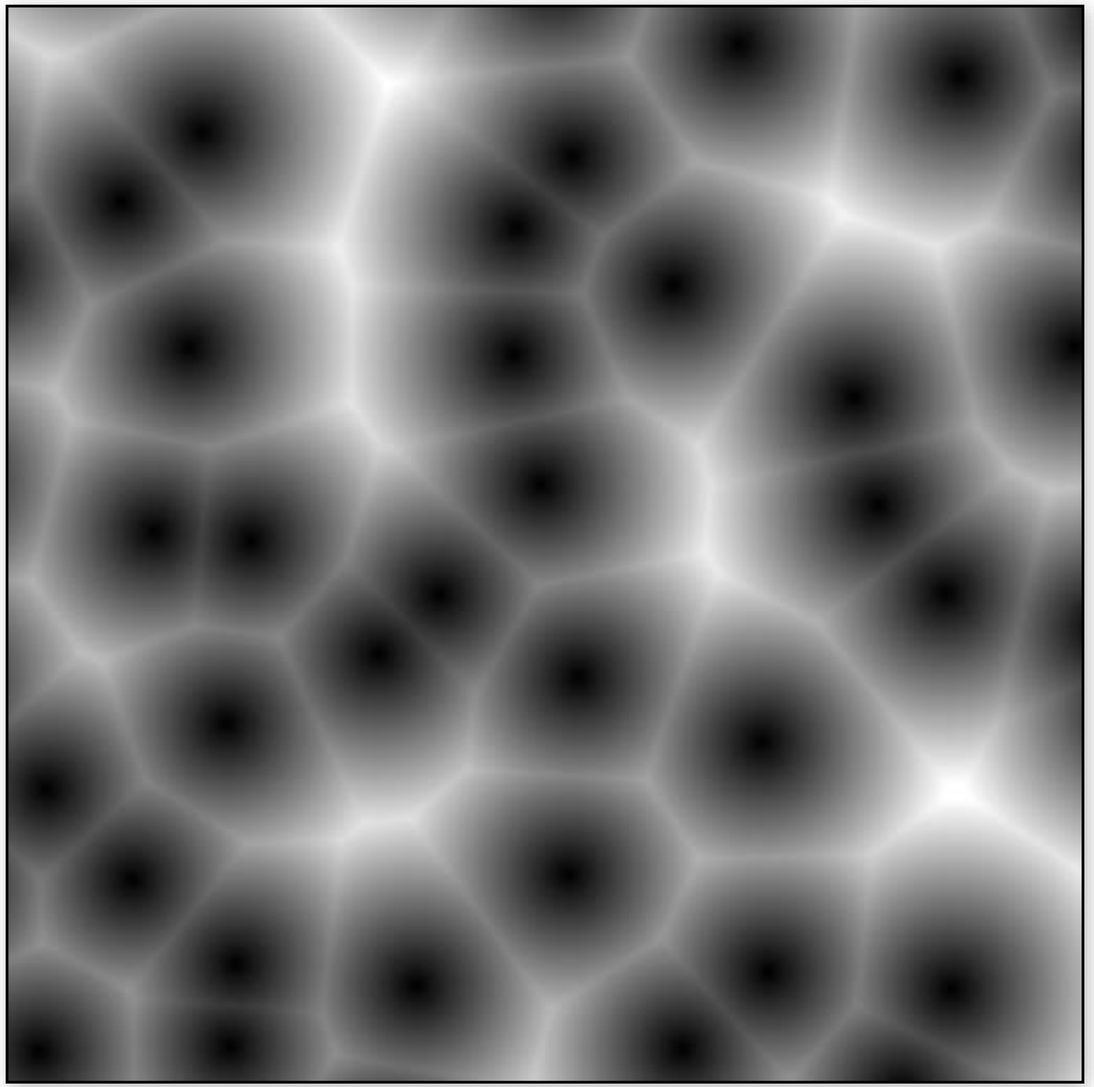


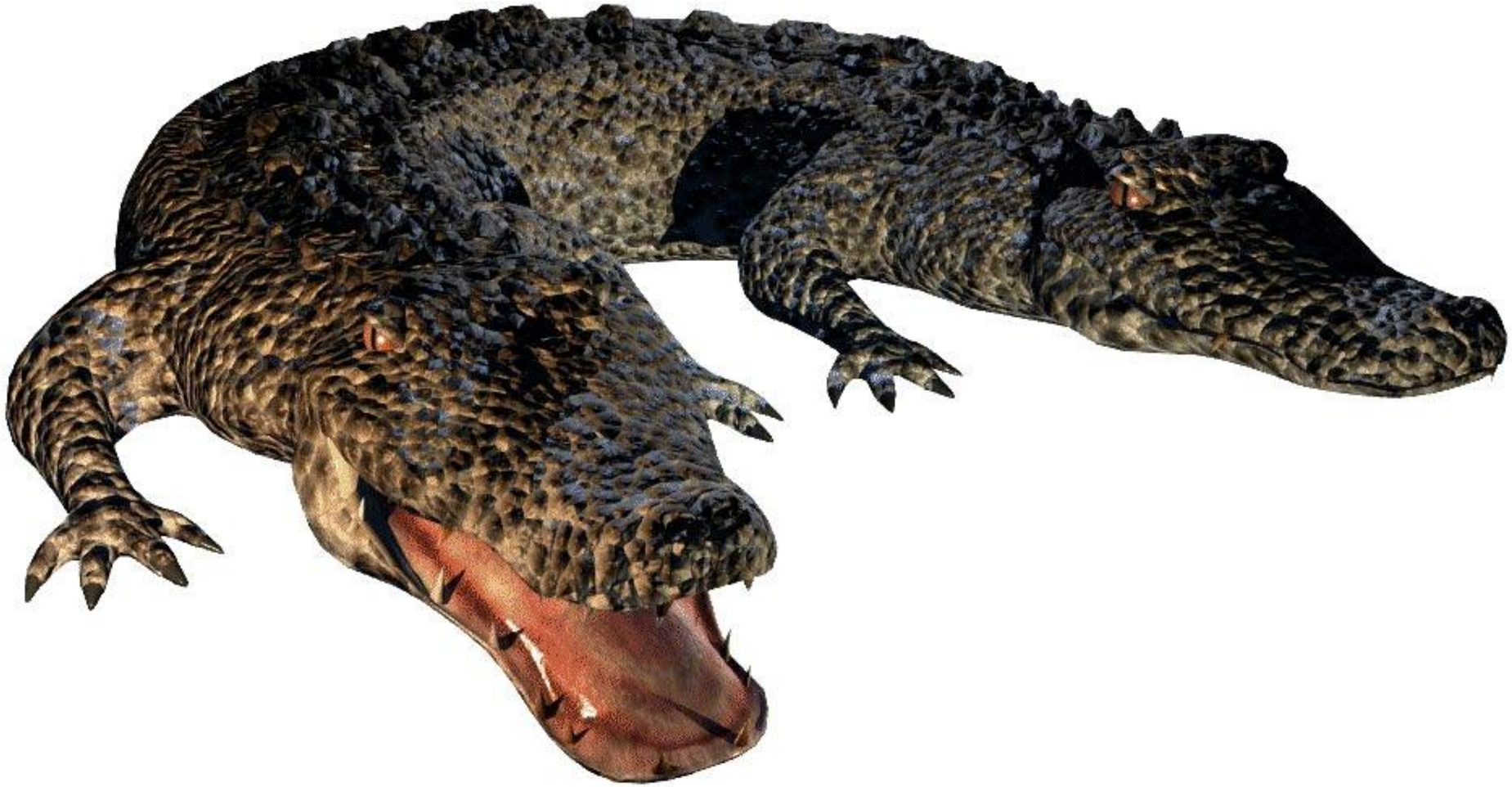
Image Credit: Wojciech Jarosz

Worley Noise in Graphics Rendering

fractal F1, bump map

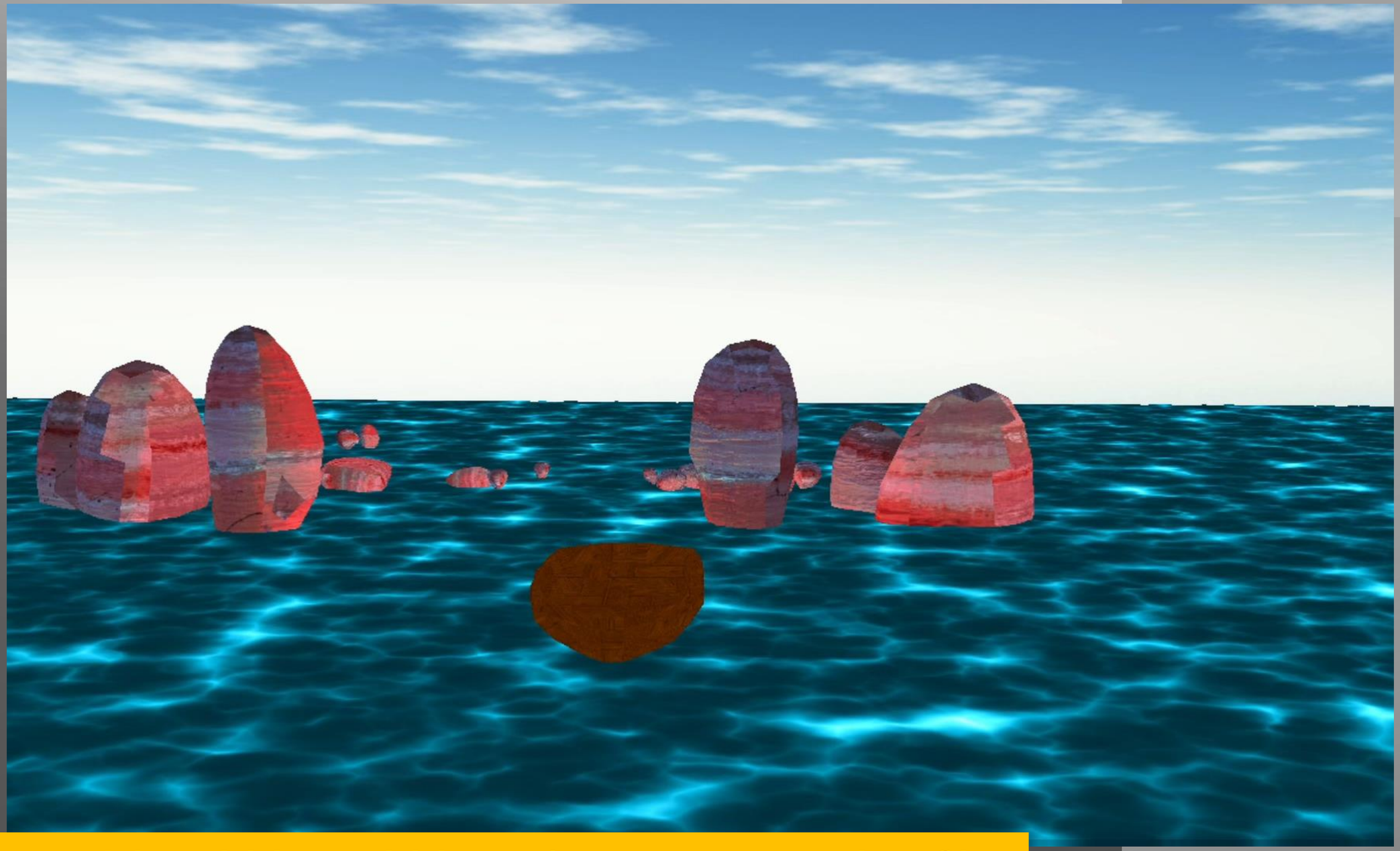
Image Credit: Steve Worley

Worley Noise

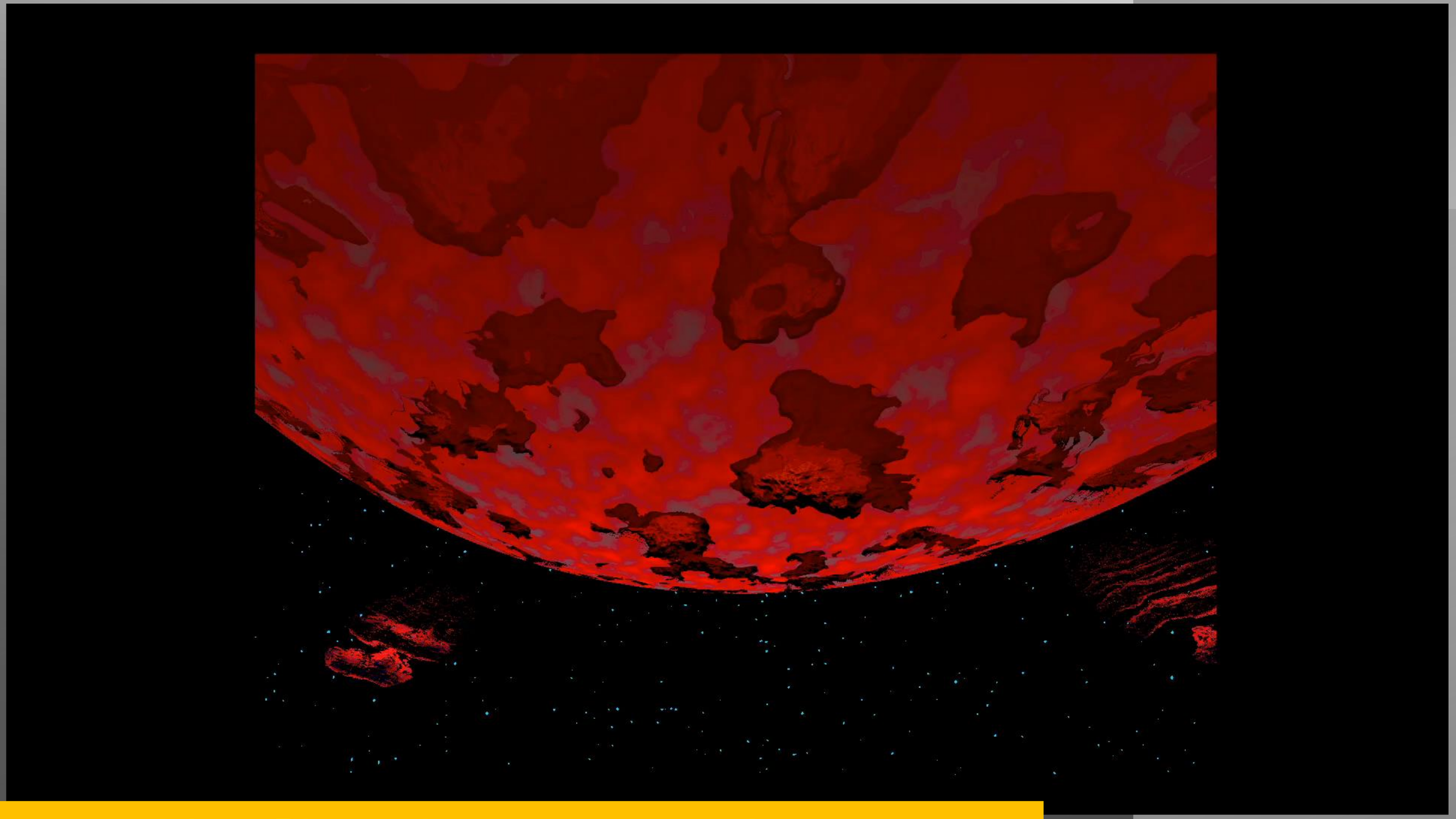


fractal f_1, color and bump map

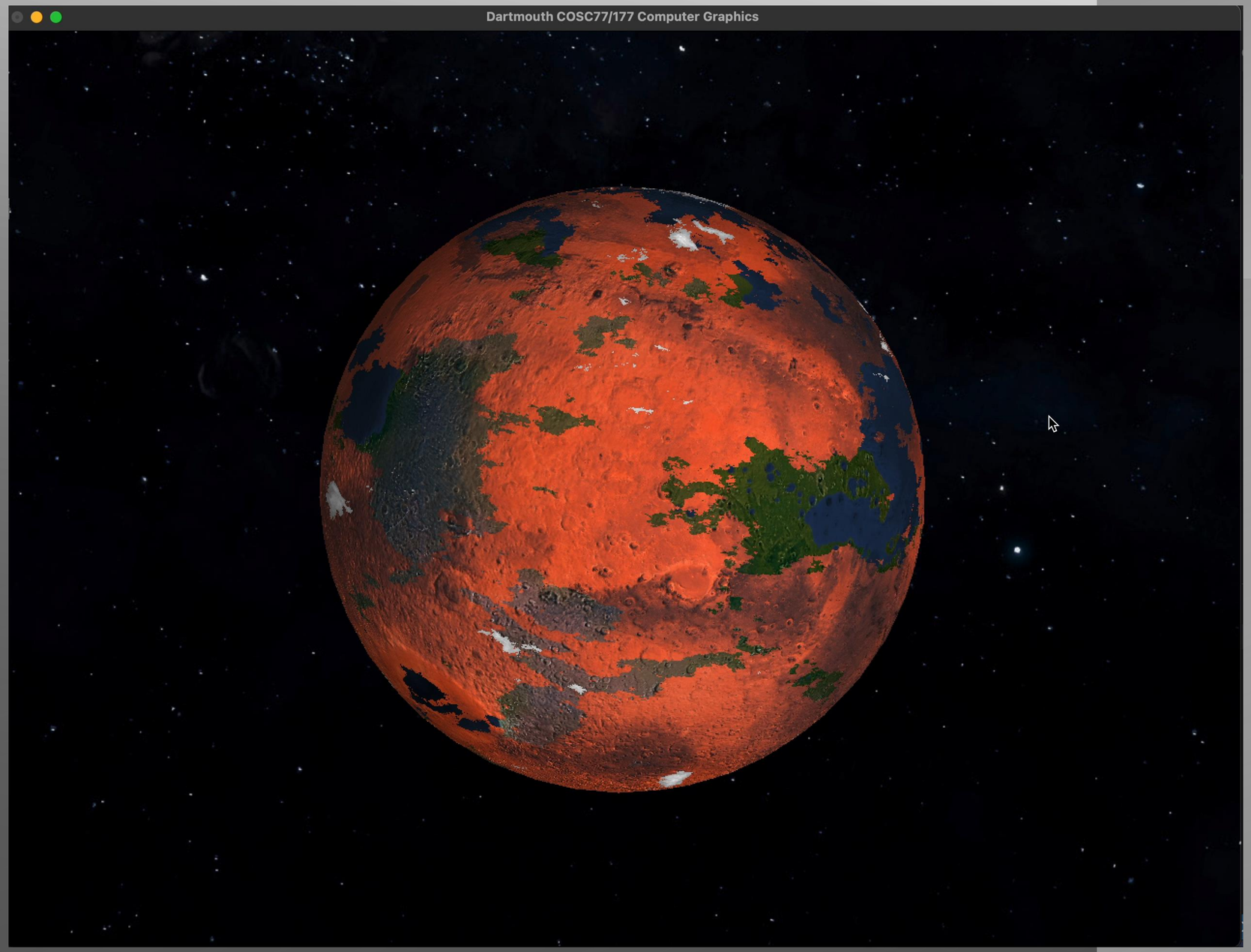
Image Credit: Steve Worley



Awesome Idea: Water Wave Animation under Sky



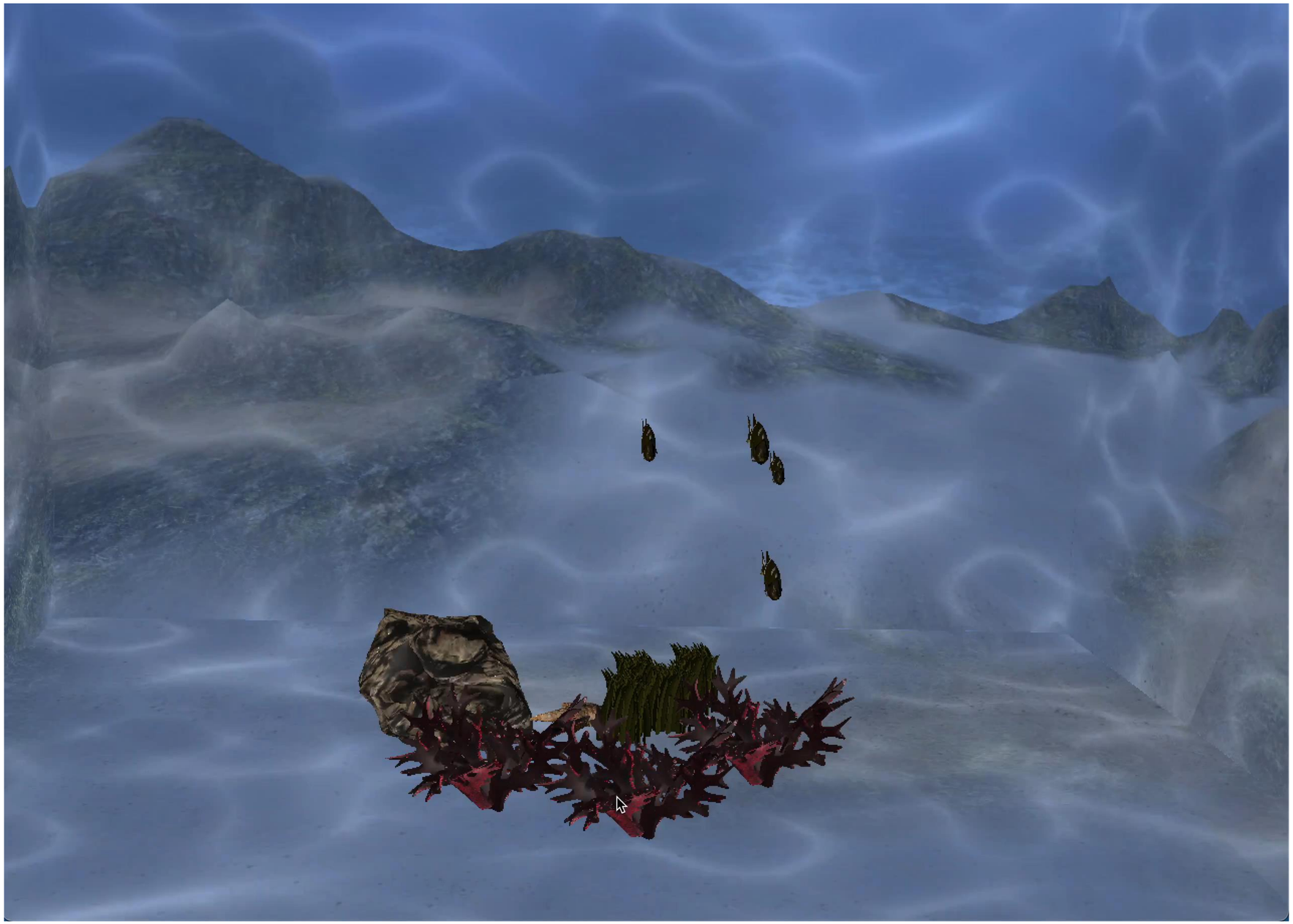
Awesome Idea: Planet lava terrain with ray tracing



Awesome Idea: Cloud Animation on a Planet

Awesome Idea: Underwater Caustics

Difficulty Level: ⭐⭐⭐



Awesome Idea: Animating time elapse on a terrain

Awesome Idea: Animating water waves along a canal

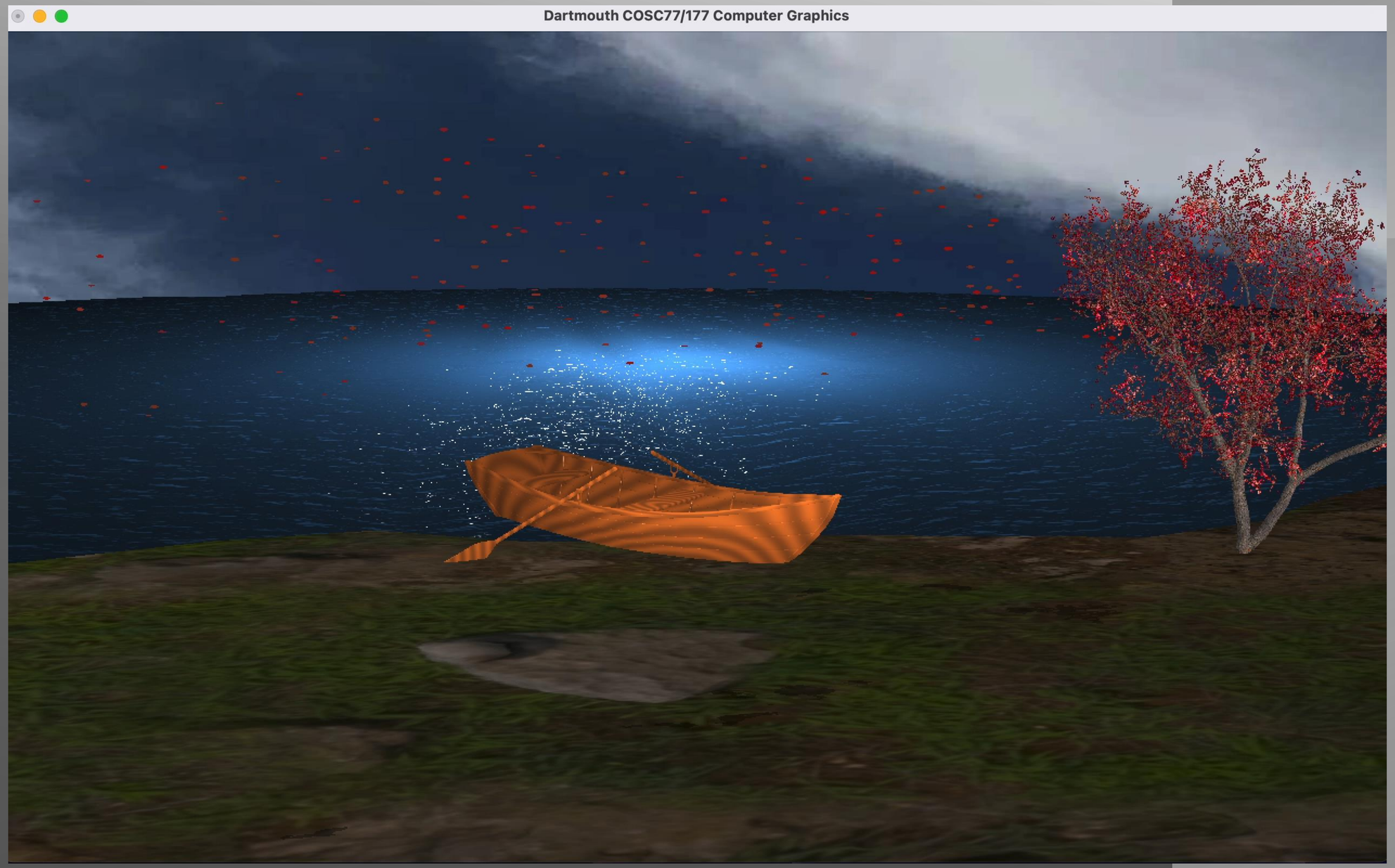
Particles

Technical Contribution: Particles

- Particle system is a popular technique being chosen by many teams to produce visually appealing animation effects
- Any particle system that is not fireworks (A8) will be counted as a new technical contribution (e.g., fireflies, stars, smoke, fire, etc.)
- Both CPU-based and GPU-based particle systems have been explored
- Some teams chose to implement CPU-based particle systems with control forces for physically-based animation effects, e.g., wind forces, snow falling
- Some teams chose to implement GPU-based particle systems such as stars, fireflys
- Particle systems are usually combined with other techniques, such as procedural modeling or random noise functions, to model large-scale natural phenomena

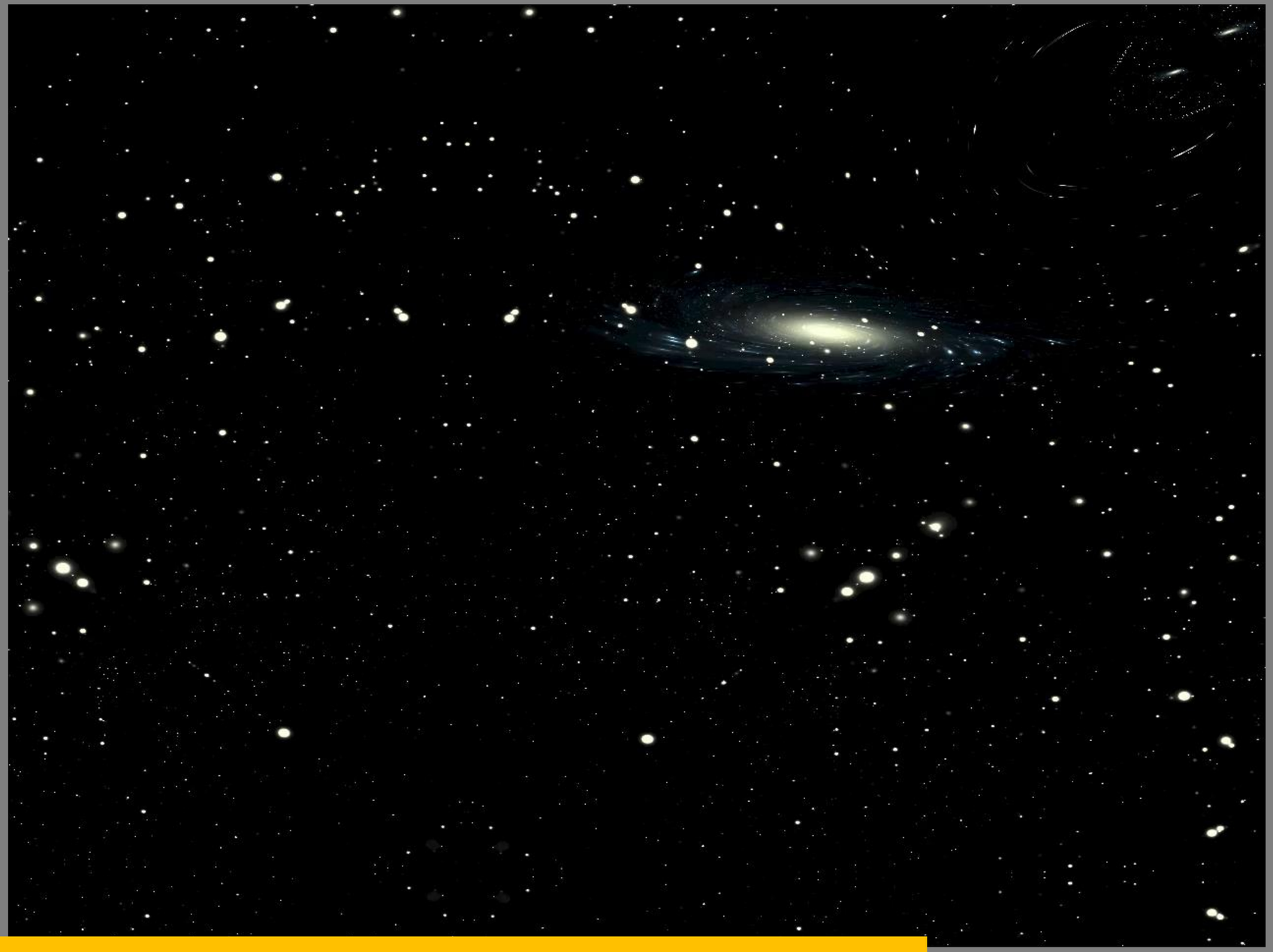
Difficulty Level: ⭐⭐

Awesome Idea: Animating fireflies with particles!



Awesome Idea: Animating leaves with particles

Awesome Idea: Animating snow with particles



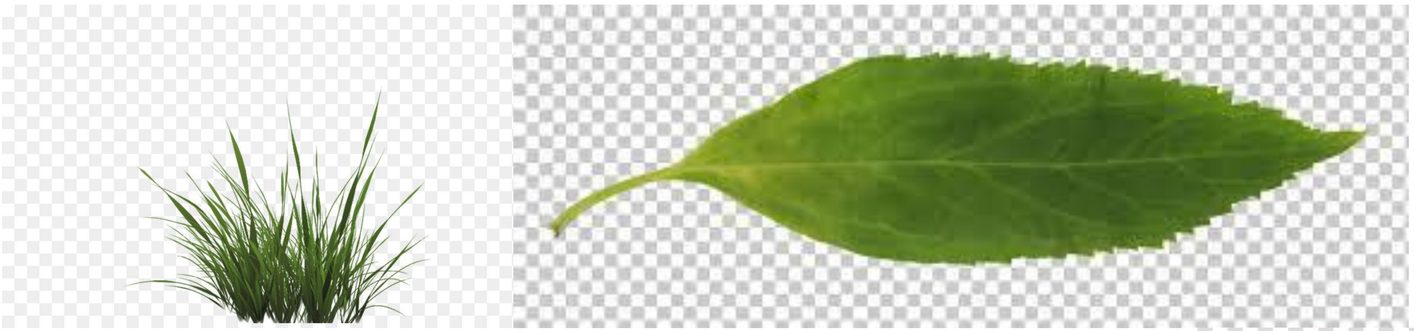
Awesome Idea: Animating galaxy with particles!

Transparency

Technical Contribution: Alpha Blending

- The key idea of employing transparency is to load textures with the alpha channel (the 'a' channel in rgba) and then turn on **alpha blending** during OpenGL rendering
- Pseudocode example:

```
glEnable(GL_BLEND);
…
glDisable(GL_BLEND);
```

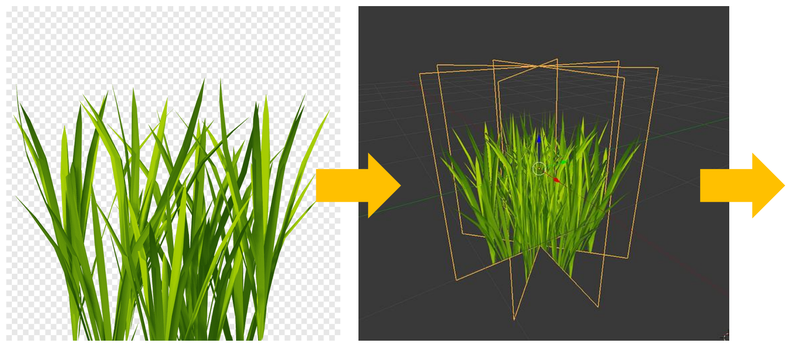


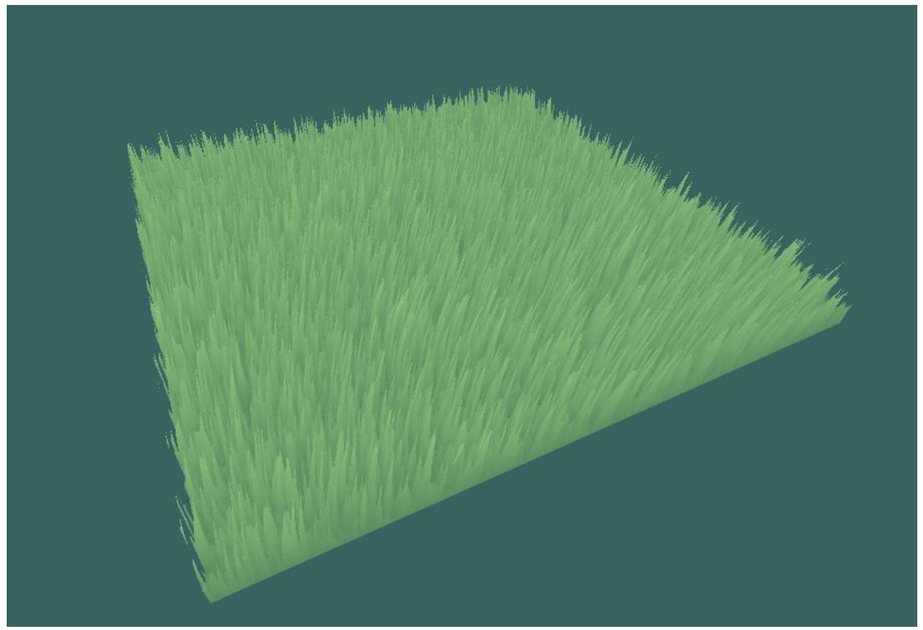
- Alpha blending is particularly useful for rendering objects with complicated silhouettes, such as grass, leaf, petal, etc.

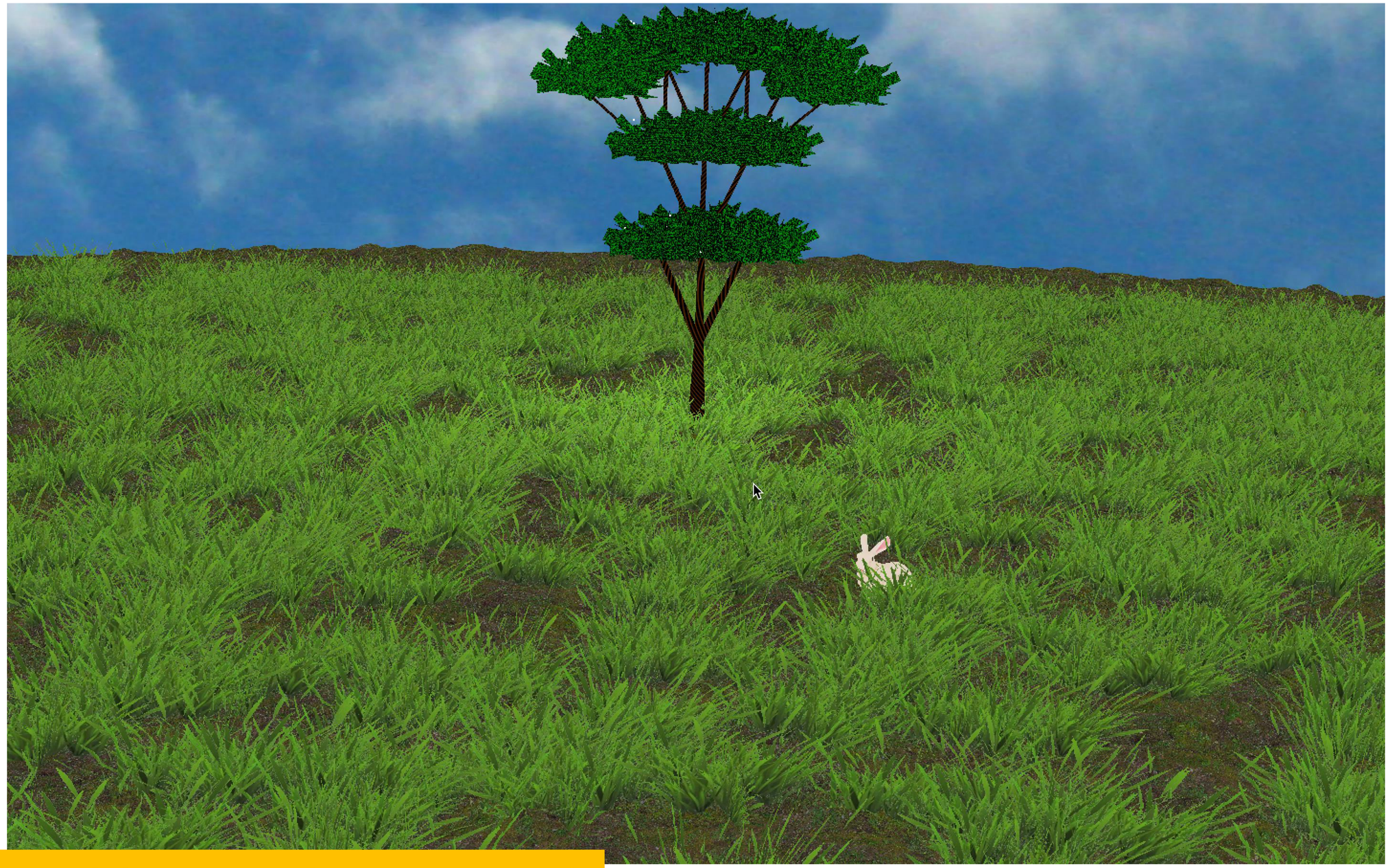
Difficulty Level: ⭐⭐⭐

Alpha Blending for **Grass** Rendering

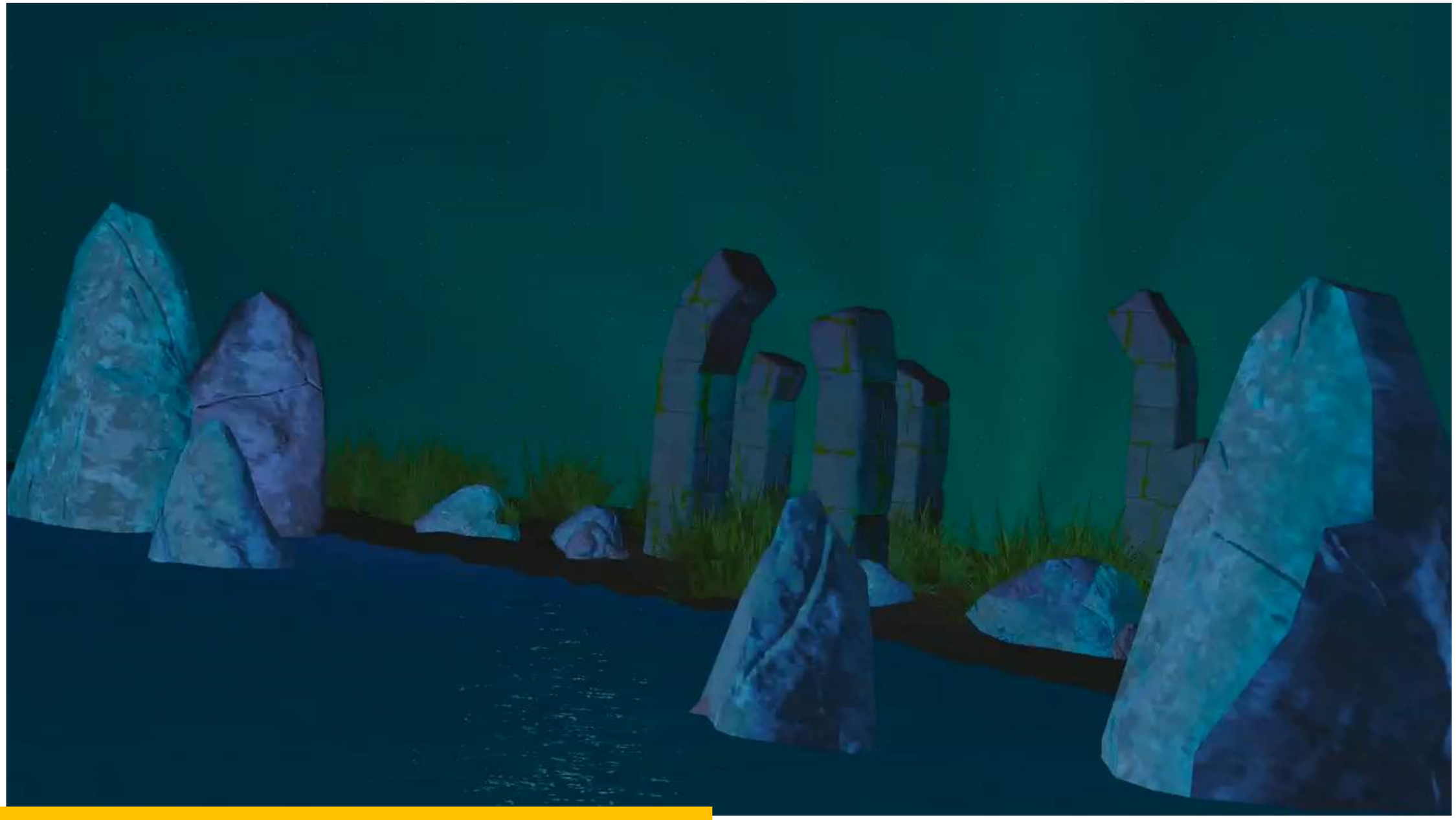
- Render each piece of grass as a billboard with textures
- Put multiple billboards with rotated angles to mimic 3D
- Animate a large number of billboards with wind







Awesome Idea: rendering grass!



Another example of grass rendering

Playing with Lights

Technical Contribution: Spotlight and Dynamic Light

- Playing with lighting models in GLSL to enhance lighting effects is an interesting topic to explore
- Some teams wrote some very interesting shader code to simulate the effect of sunlight streaming into room
- These light effects can interact with textures to produce more intriguing visual effects
- Other advanced lighting effects in GLSL can be explored, such as spotlight and area light

Difficulty Level: ⭐⭐

Dynamic Lighting

Idea: simulate a moving light source streaming into a dark room through window texture

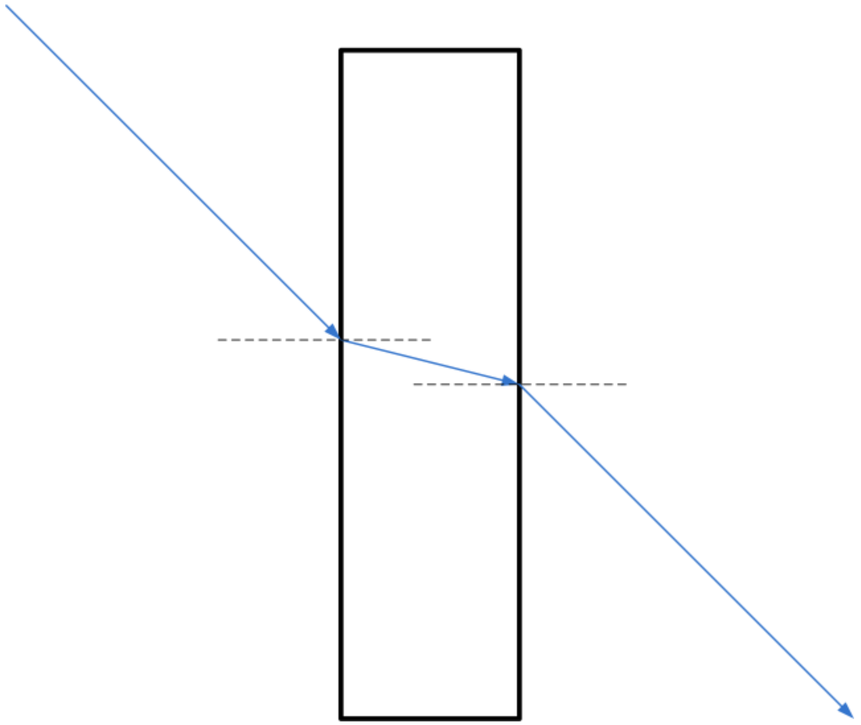
Awesome Idea: light streaming through window!

Technical Contribution: Advanced Features for Ray Tracing

- You may consider implementing features beyond a basic ray tracer as the core technical contribution for your final project
- We don't have previous works to show you, as the previous versions of our ray tracer is not as powerful as our A7 :D
- But I will introduce a few topics that you may consider to challenge yourself if you are a big fan of ray tracing
- Most of these techniques rely on casting more than one rays for each pixel, and then use their combination to achieve advanced lighting effects such as soft shadow, motion blur, participating media, etc.

Difficulty Level: 

Refraction



- Trace a refracted (transmitted) ray if the object is transparent
  - Ray passes through object
  - Shade transmitted ray and return color
  - Add color to shading at the original point
    - Multiply by the refraction coefficient k_t

Beer's Law

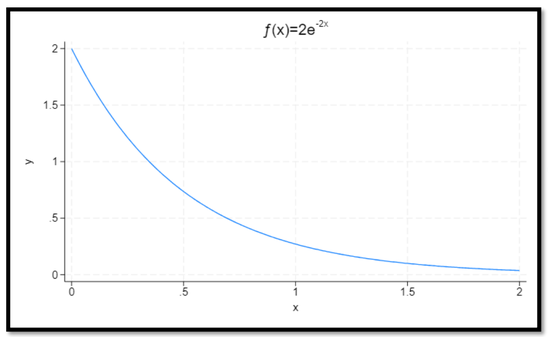
- If the media is homogeneous, the attenuation along the ray can be described using Beer's Law:

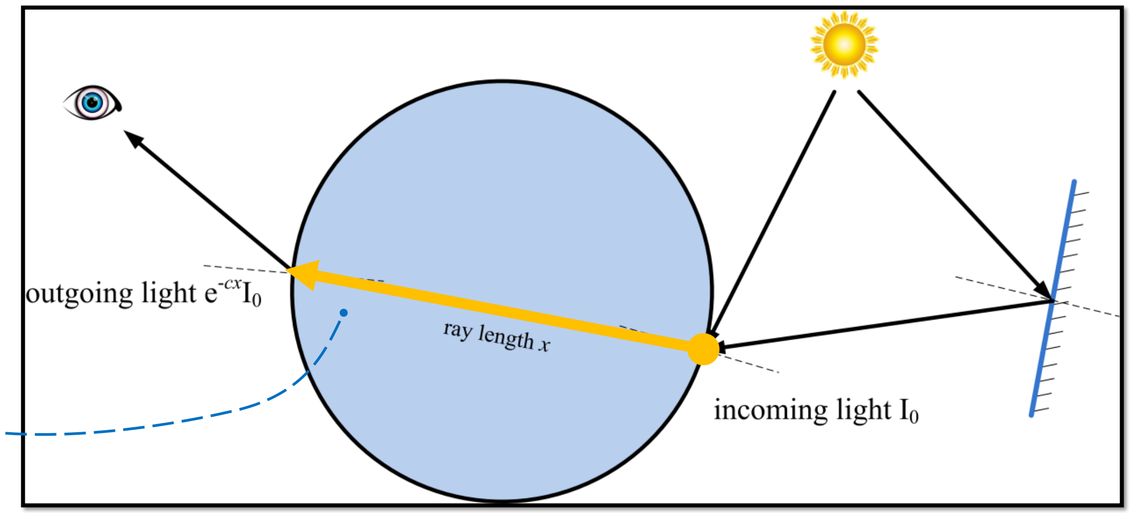
$$\frac{dI}{dx} = -cI$$

in which *I* is the light intensity, *x* is the distance along the ray, and *c* is the attenuation constant.

- Solve this ODE with the initial value $I(0) = I_0$, we have:

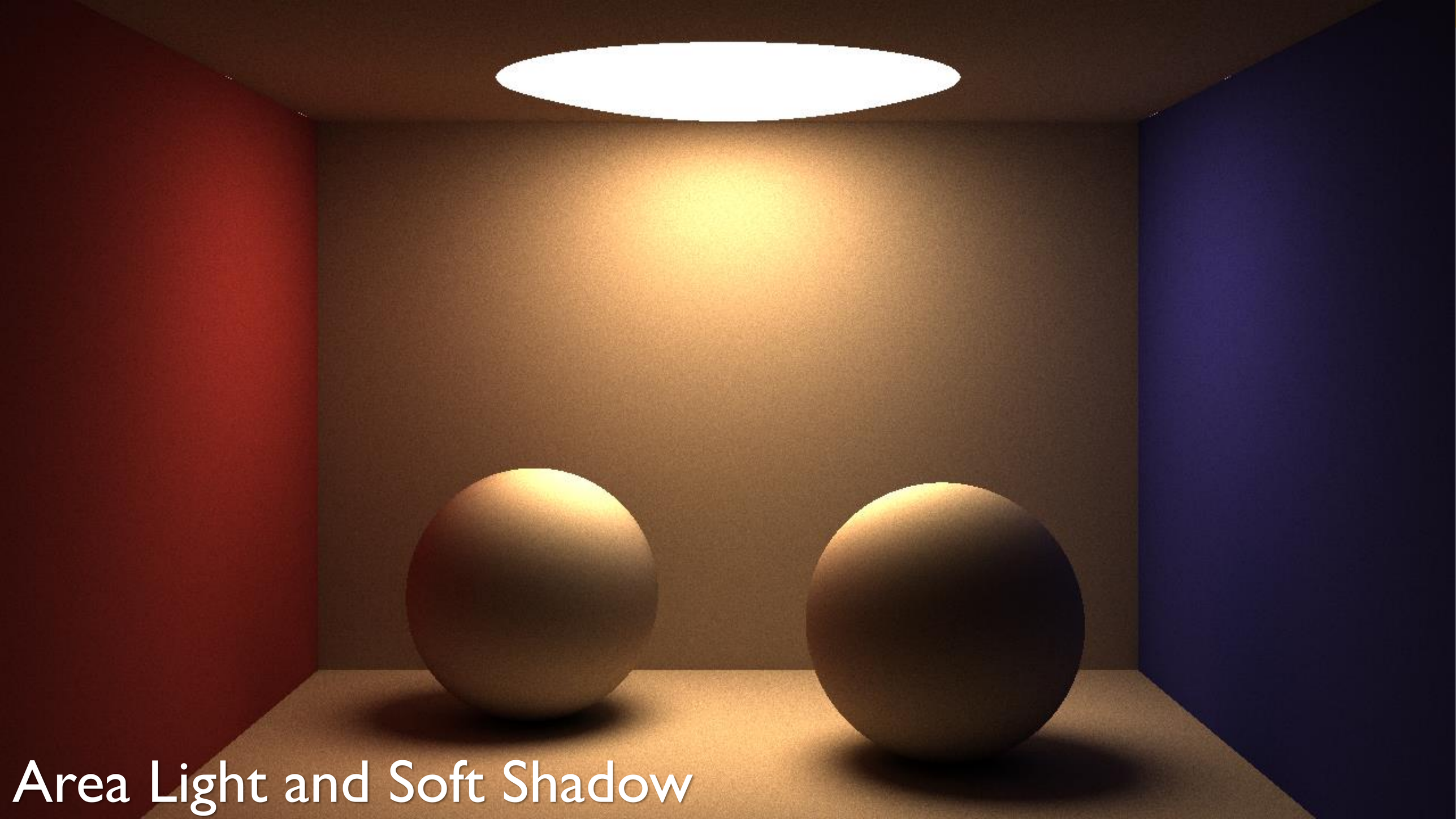
$$I(x) = I_0 e^{-cx}$$



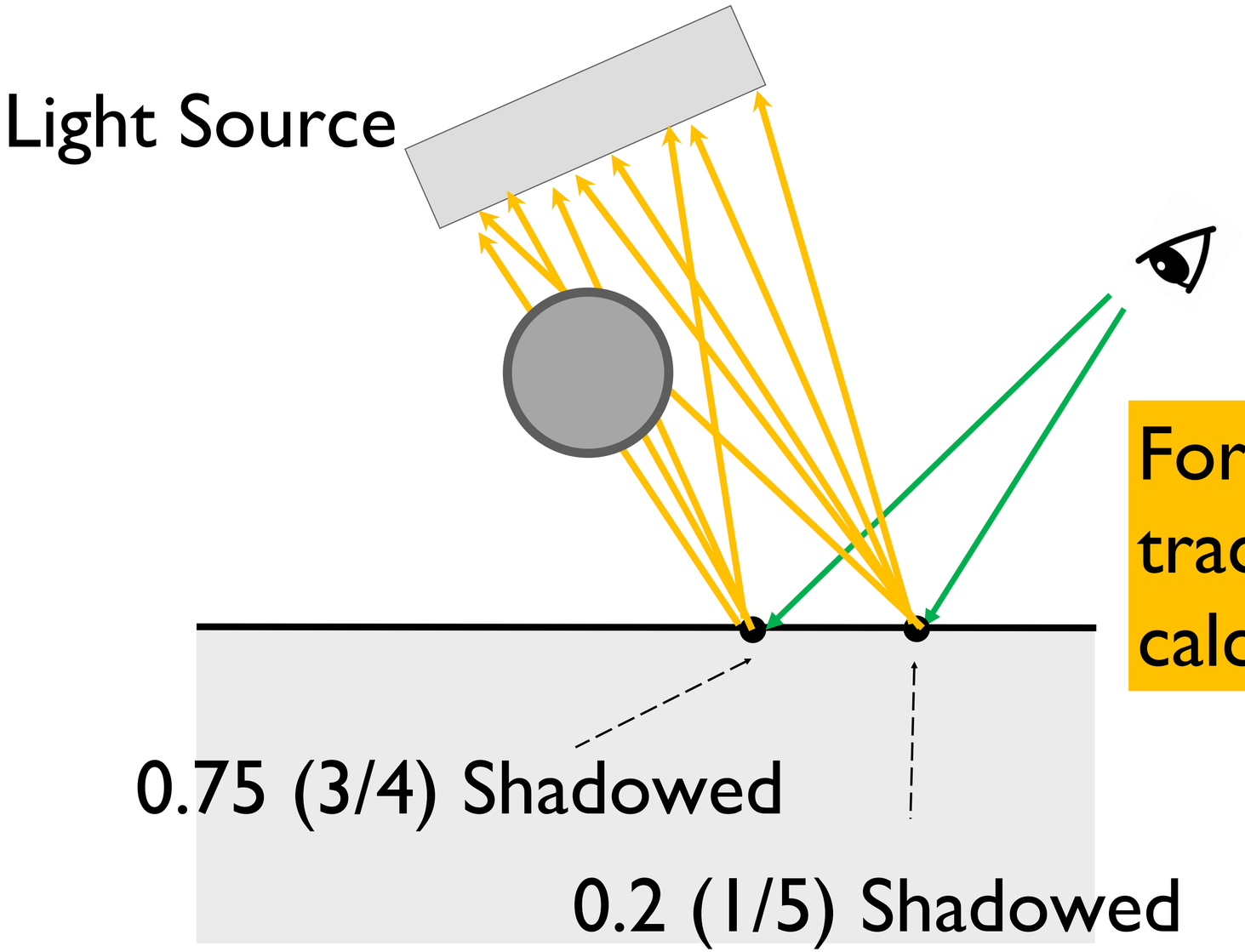


Advanced ray tracing You might want to consider

- Soft shadow and area light
- Depth of field
- Motion blur
- Participating media


Area Light and Soft Shadow

Key Idea: Trace Multiple Shadow Rays



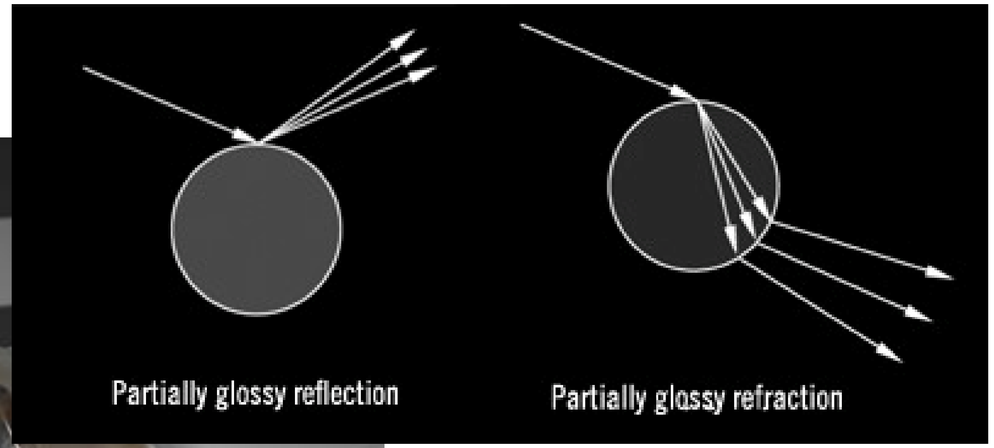
For each intersection, we trace multiple shadow rays to calculate its soft shadow value

Soft Shadow Pseudocode

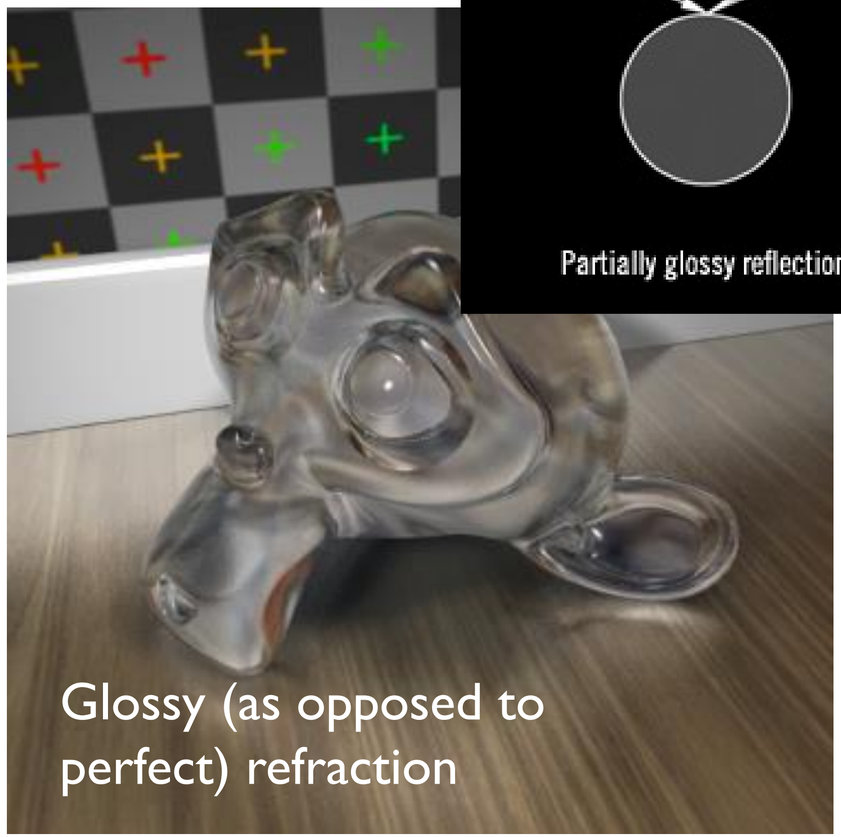
```
trace(Ray ray):
        hit = surfaces.intersect(ray)
        if hit
                emit = hit->mat->emit(ray)
                sRay = hit->mat->scatter(ray)
                return emit + trace(sRay)
        else
                return backgroundColor
```

Non-Ideal Reflection/Refraction

- **Key Idea:** Trace multiple rays for each reflection/refraction to consider the non-ideal surface



Glossy (as opposed to mirror) reflection

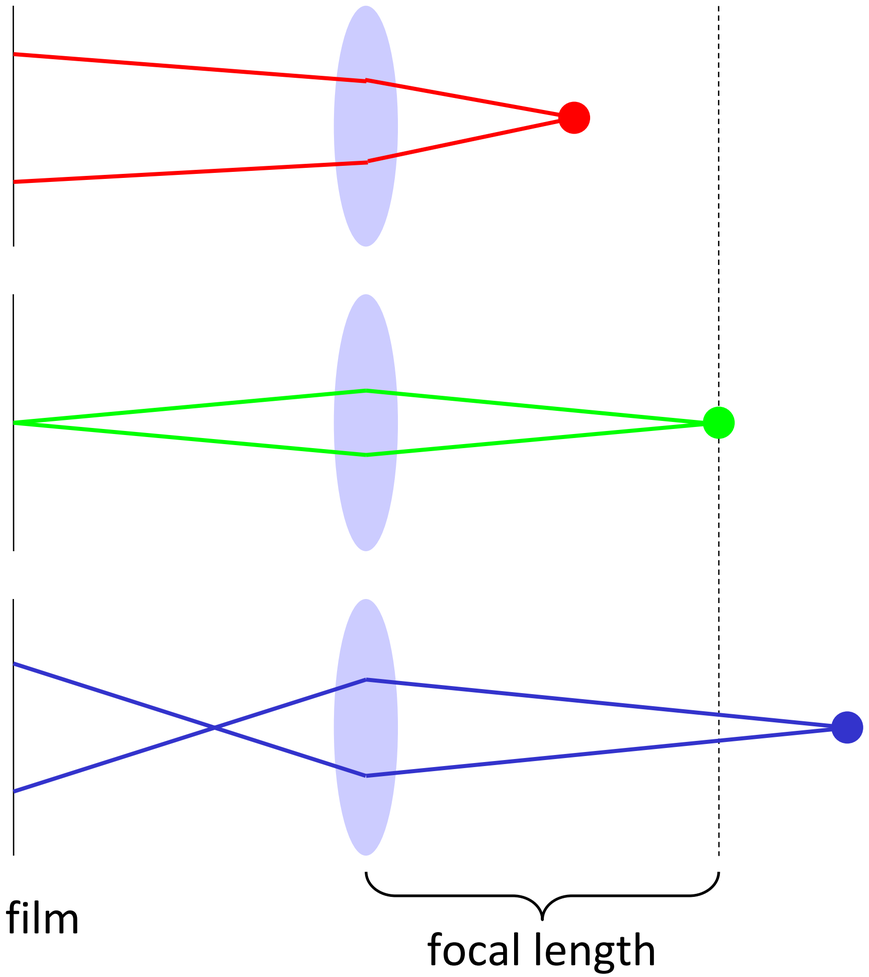


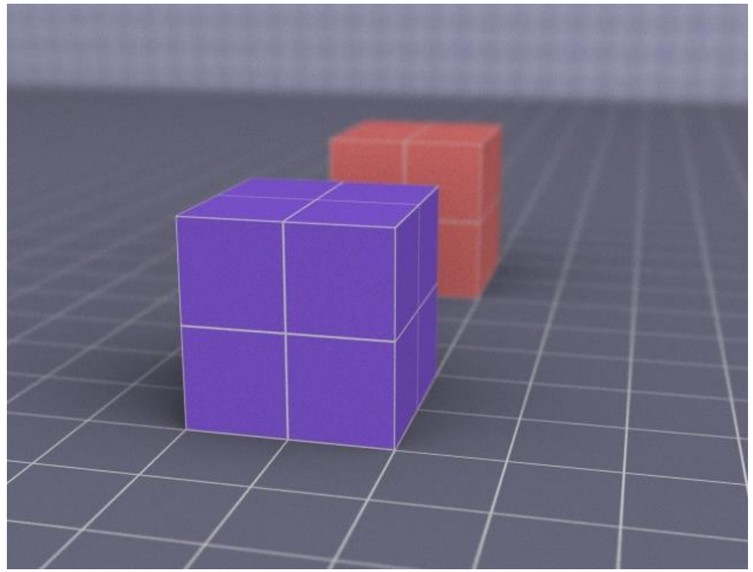
Glossy (as opposed to perfect) refraction

Depth of Field

Depth of Field

- **Key Idea:** trace multiple rays per pixel to sample the lens aperture





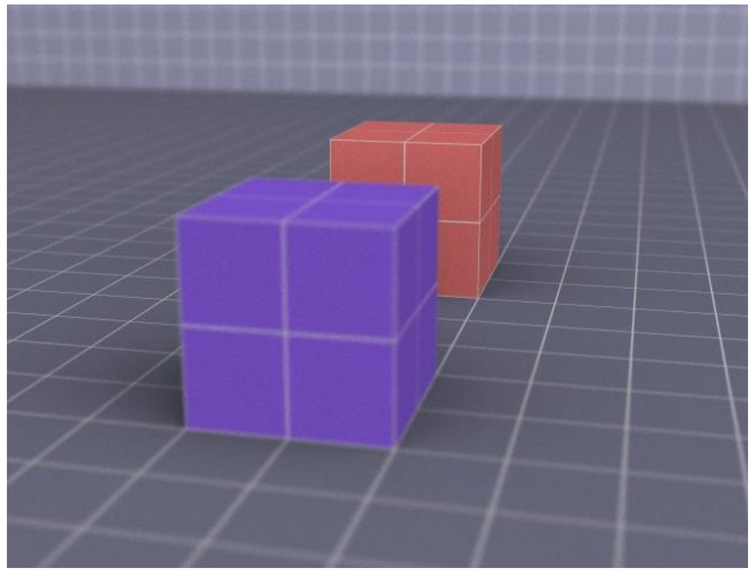
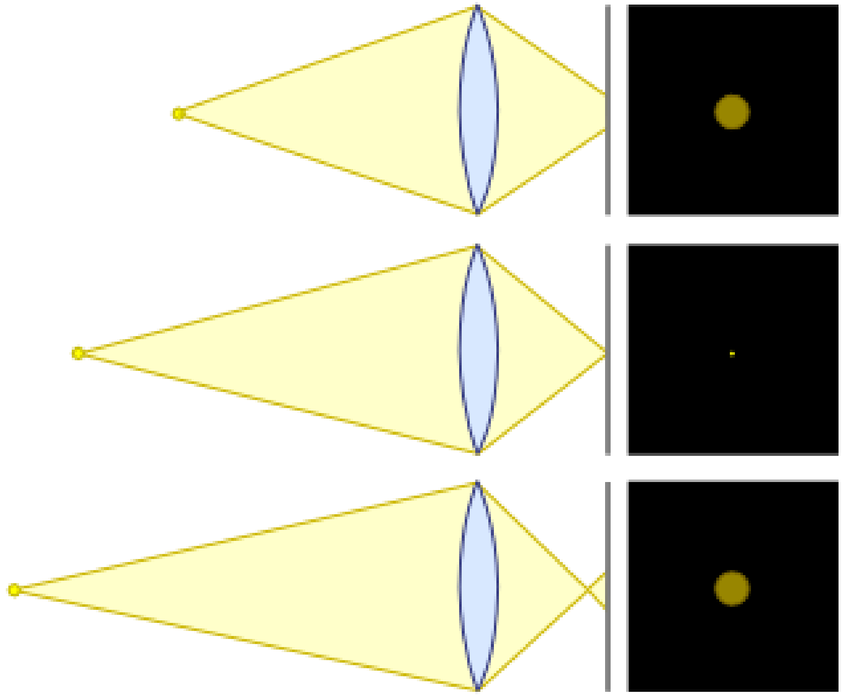


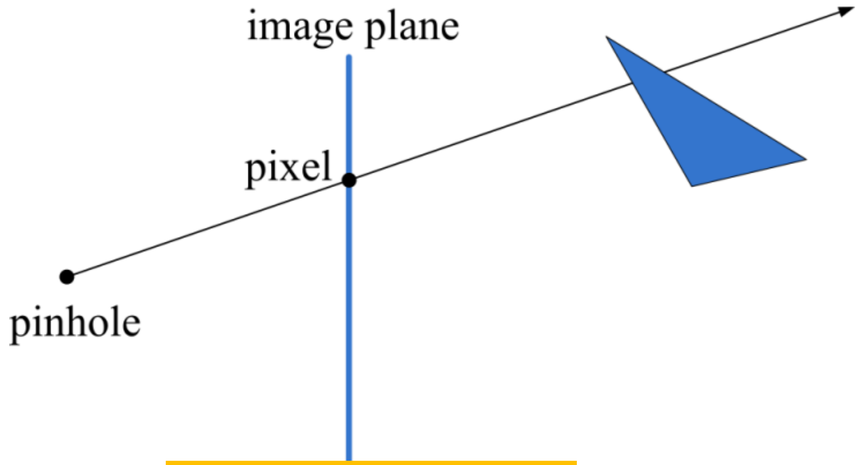
Image by Justin Legakis

Circle of confusion

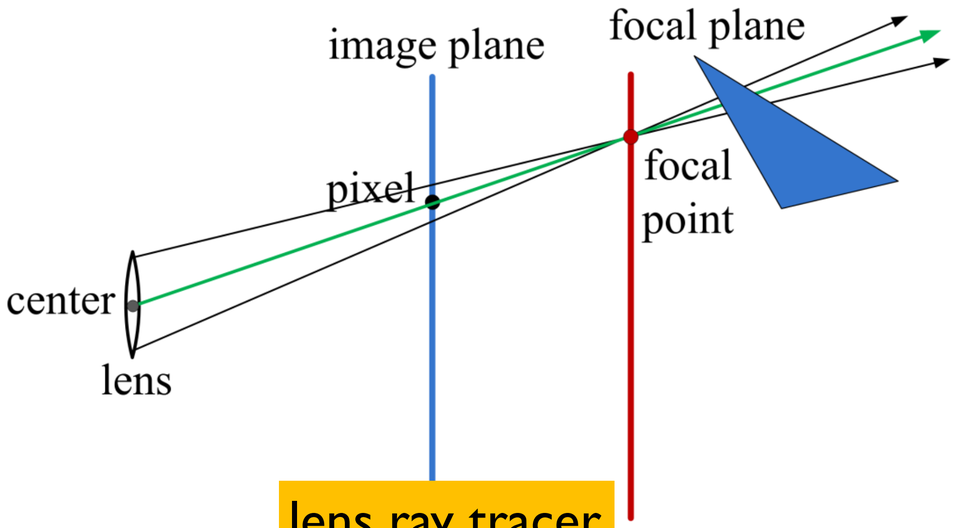


- An optical spot caused by a cone of light rays from a lens not coming into perfect focus when imaging a point source
- When the spot is approximately equal to the size of a pixel on the sensor, the object seems to be "in focus"
- Objects at varying distances from the camera require the sensor to be placed at different distances from the lens in order for the object to be "in focus"
- **Depth of Field** - the distance between the nearest and farthest objects in a scene that appear roughly "in focus" (the circle of confusion is not too big)

Depth of Field Implementation



pinhole ray tracer



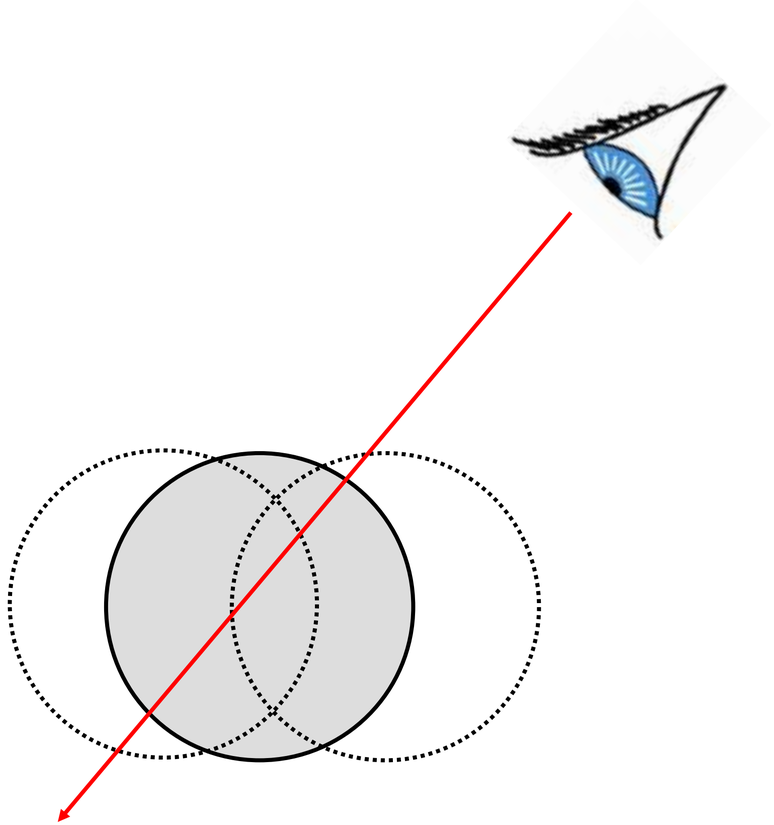
lens ray tracer

Algorithm

- Specify a focal plane (red) where objects will be in focus
- The "focal point" is calculated as the intersection of the ray (green) and the focal plane
- For each pixel, replace the pinhole "eye" with a circular region
- Shoot multiple rays from sampled points in the circular region through the focal point
- Average the color: objects further away from the focal plane will have more blurring

Motion Blur

- **Key Idea:** Sample objects temporally over time interval

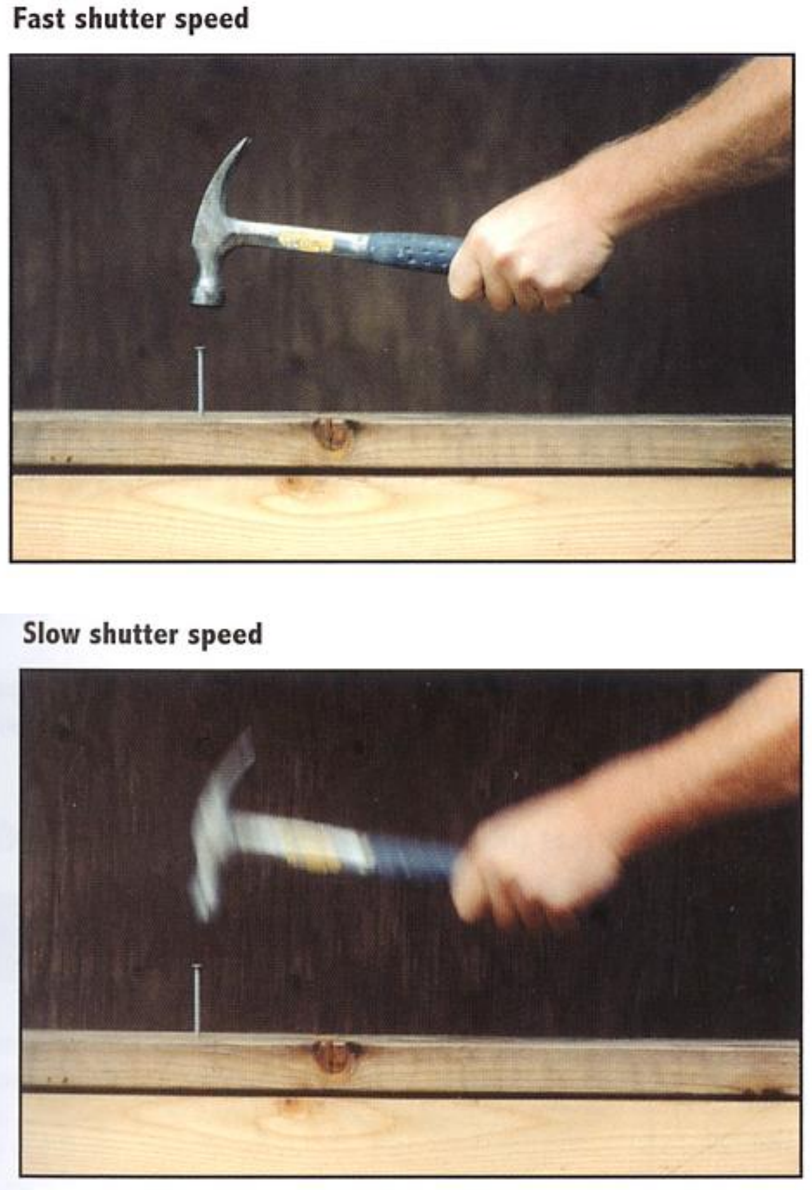


Shutter Speed

- The shutter allows light to hit the sensor for a finite duration of time
- Objects which move while the shutter is open create multiple images on the sensor, resulting in motion blur
- A faster shutter speed prevents motion blur, but can severely limit the amount of light available to the sensor making the resulting image too dark (especially when the aperture size is small)

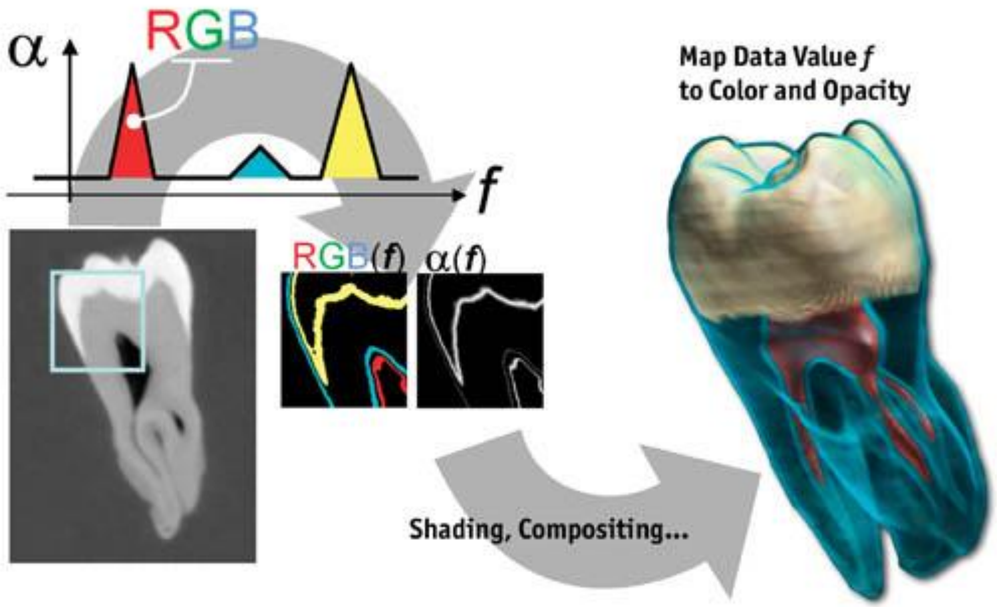
Motion Blur Ray Tracing

- Set up animations for moving objects during the time interval in which the shutter is open $[T_0, T_1]$
  - E.g. describe the transform of the object by a function F(t) for $t \in [T_0, T_1]$
- For each ray:
  - Assign a random time $t_{ray} = (1-\alpha)T_0 + \alpha T_1$, all objects in the scene are placed in their time t_{ray} locations
  - Trace the ray with the time t_{ray} scene and get a color for that ray



Volumetric Ray Tracing

- Generate images of a 3D volumetric data set without explicitly extracting geometric surfaces

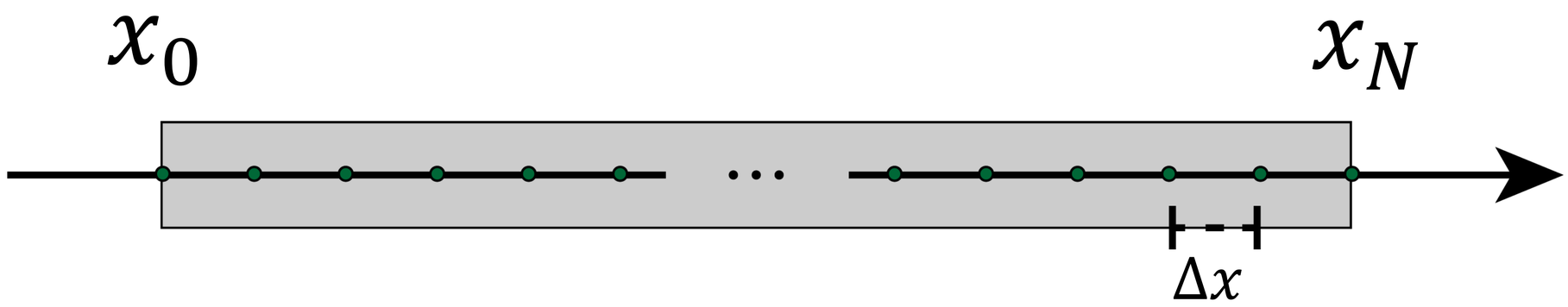


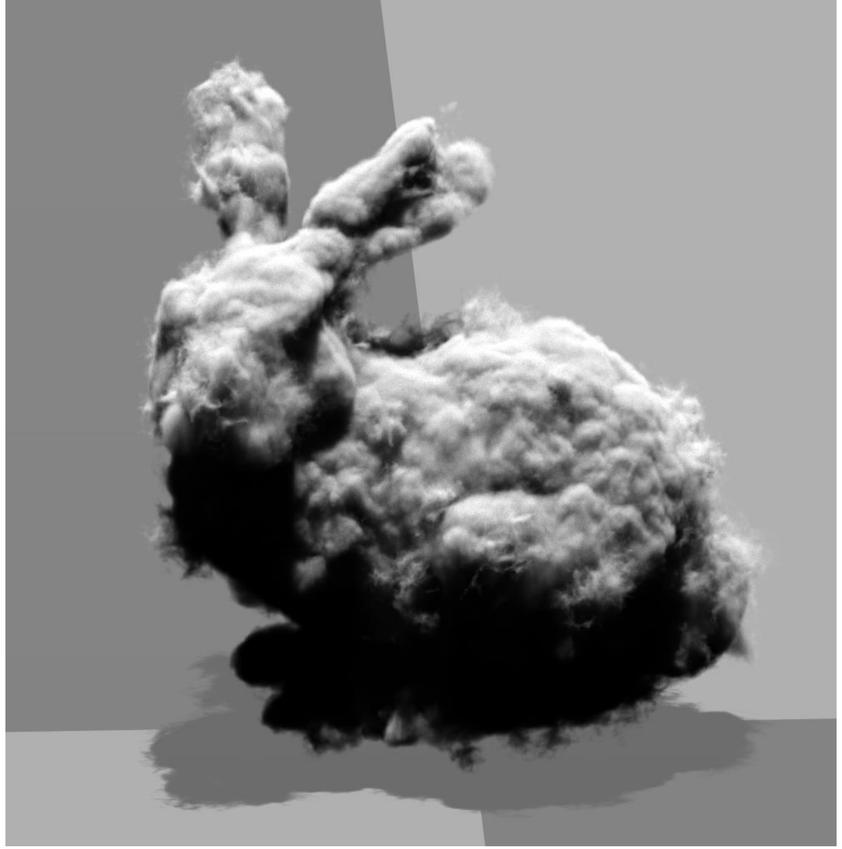
https://developer.download.nvidia.com/books/HTML/gpugems/gpugems_ch39.html

Key Idea: Discretize a ray into many small segments

- **Key Idea: Discretize the ray into N segments**, and accumulate the attenuated color from each segment along the ray
- The attenuation along the i-th segment is set to be $e^{-c(.5(x_{i-1}+x_i))\Delta x}$
  - $\Delta x = (x_N - x_0)/N$ is the segment length, $c(.5(x_{i-1}+x_i))$ is the attenuation constant of each segment, and $x_i = x_0 + \mathrm{i}\Delta x$
- The total attenuation along the ray is computed via multiplication:

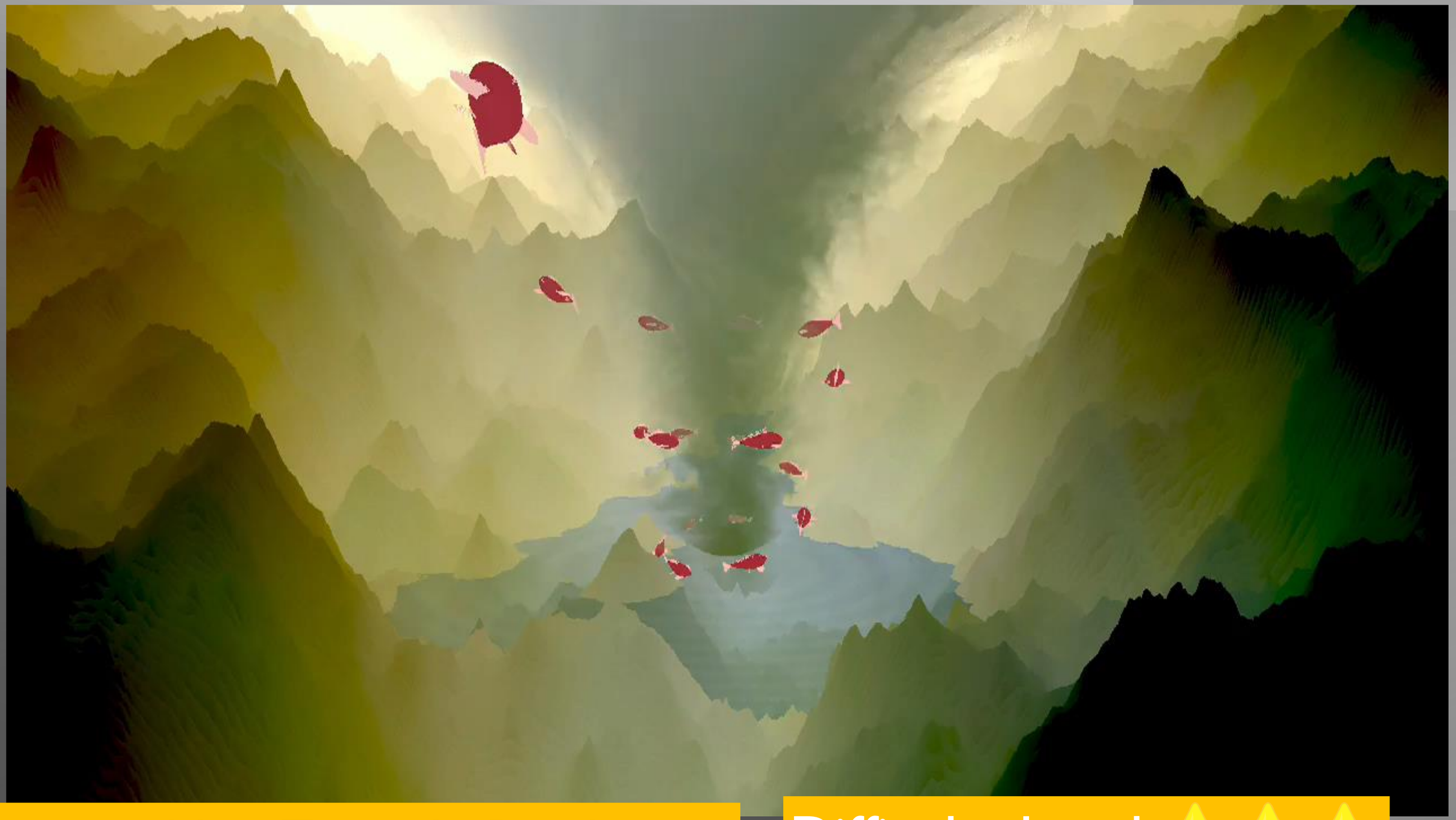
$$e^{-c(.5(x_0+x_1))\Delta x} e^{-c(.5(x_1+x_2))\Delta x} \dots e^{-c\left(.5(x_{N-1}+x_N)\right)\Delta x}$$





Ray Tracing of Participating Media

- Trace rays into a volume of cloud/smoke/fire
- Sample each ray with many small segments
- Calculate the attenuation and emission of each segment
- Calculate the final color by accumulating colors along each ray



Awesome Idea: Tornado Rendering

Difficulty Level: ⭐⭐⭐


Good luck with your graphics journey!