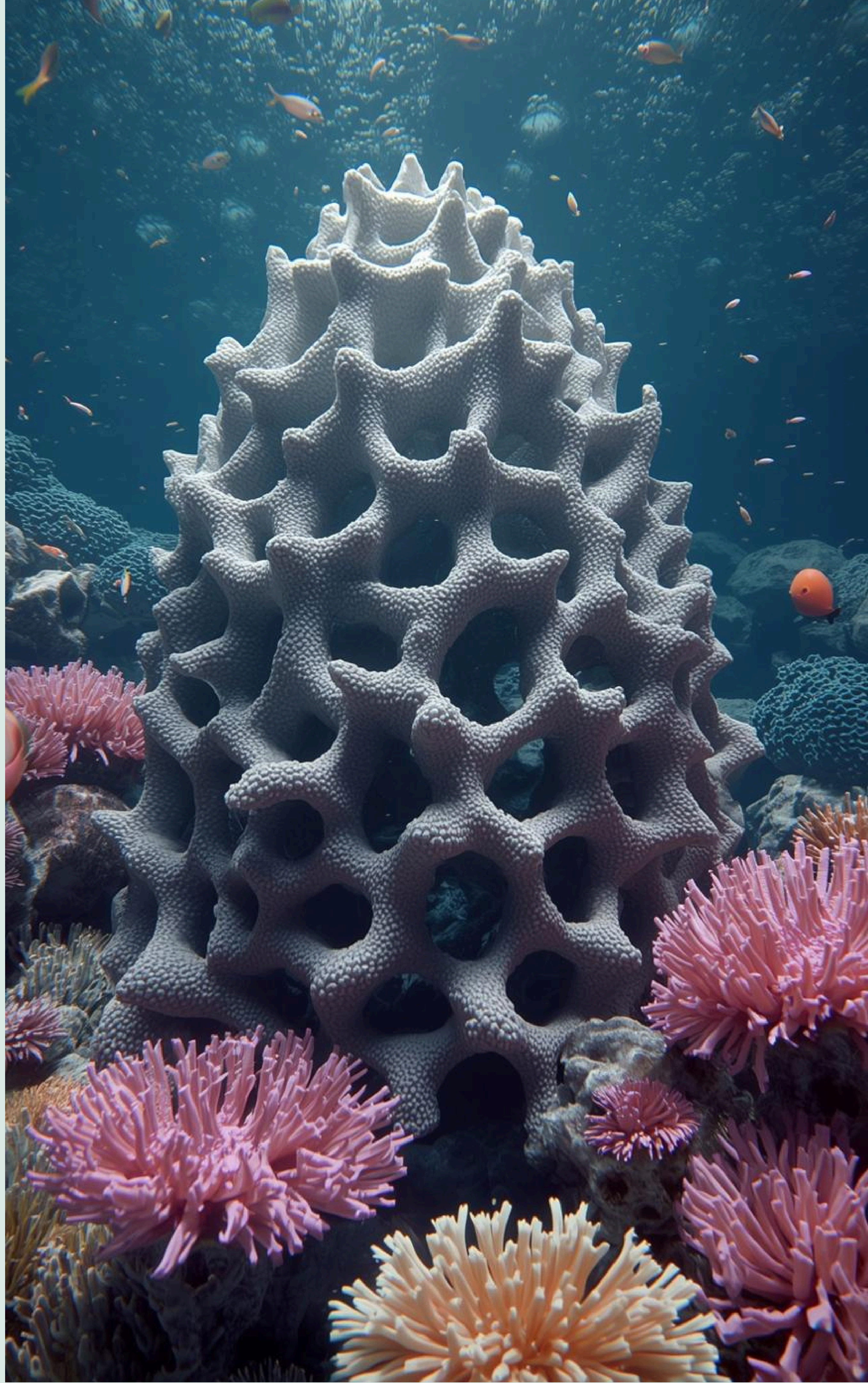
CSCI-510, ROCHESTER INSTITUTE OF TECHNOLOGY

3D Stochastic Coral Structures using L-Systems



Presented by Anna Leung, Bennett Chang

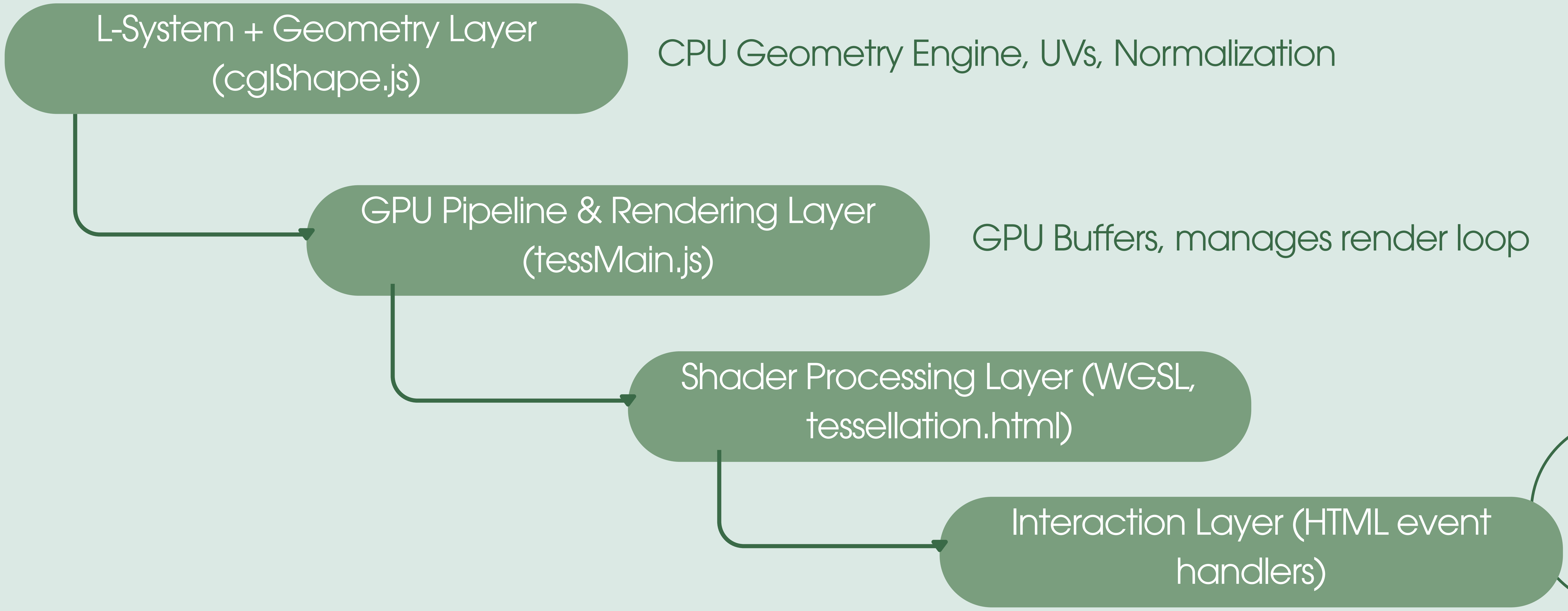
Project Requirements and Satisfaction

Meeting Goals with L-System Implementation

We developed a 3D procedural coral reef generator through a combination of texturing (sand & coral), camera integration, and procedural generation. We used stochastic L-Systems to generate unique organic structures every time the scene is refreshed.

High-Level Code Architecture

Structure and components overview; very similar to class assignments


L-System + Geometry Layer (cglShape.js)
CPU Geometry Engine, UVs, Normalization
GPU Pipeline & Rendering Layer (tessMain.js)
GPU Buffers, manages render loop
Shader Processing Layer (WGSL, tessellation.html)
Interaction Layer (HTML event handlers)

Key Components Overview

L-System Grammar and Modules

This project utilizes **L-System grammar** to define coral structures, integrates stochastic rule applications for variability, and establishes a robust rendering pipeline to visualize underwater coral growth effectively.

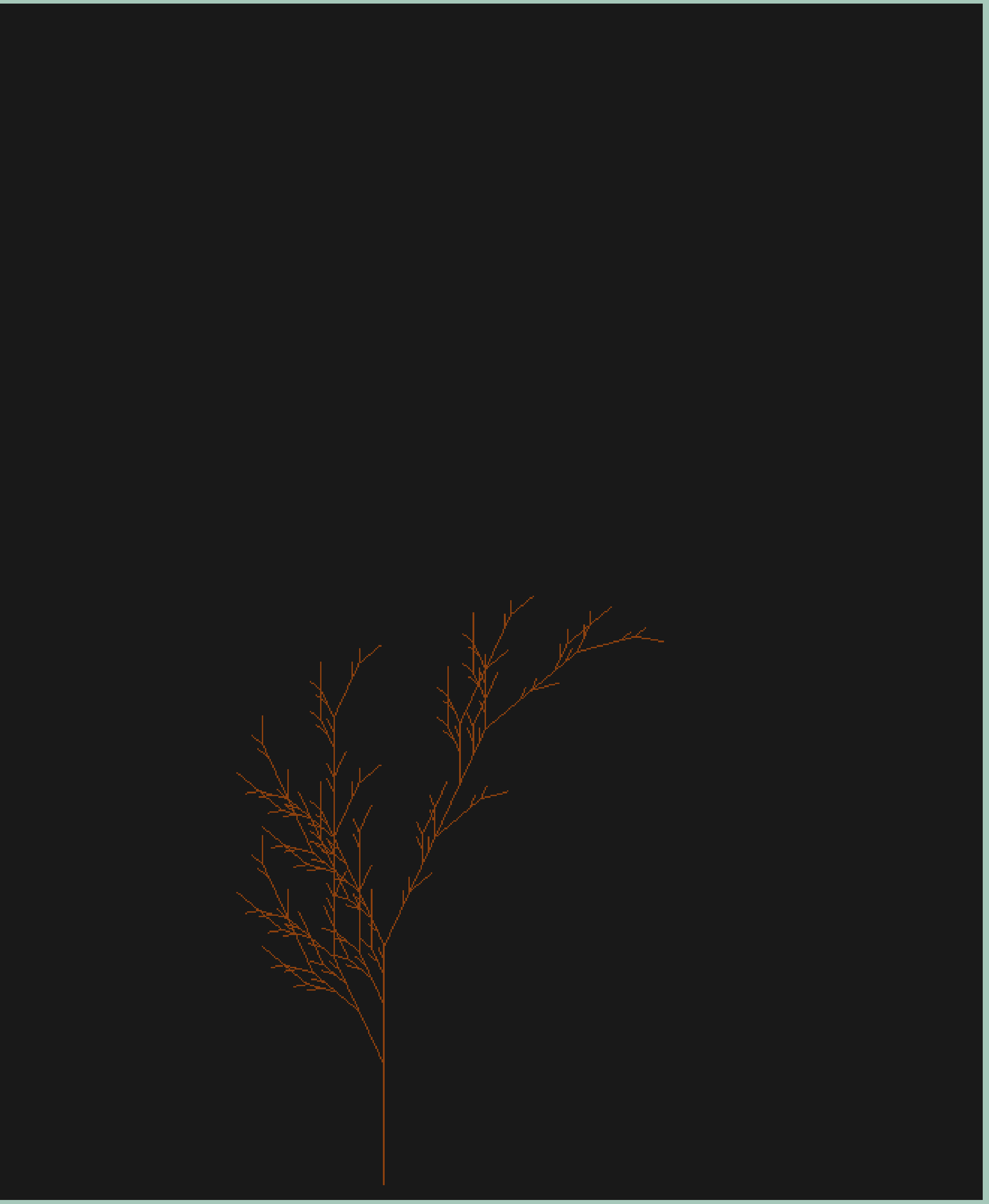
- A stochastic 3D L-System that expands using probabilistic rules.
- A full mesh-generation pipeline producing tessellated cylindrical branches.
- A WebGPU renderer with custom WGSL shaders (vertex, fragment).
- 3D ‘sand box’ as seabed/ground.
- User interaction: camera panning, rotation, tessellation control, and color modes.

Coral Structures: Before and After

Exploring Variations Through Parameter Adjustments

Initially, we had a different approach to the project. We originally proposed an L-system generation of different trees grouped together in a small forest. There was the potential to add leaves and bark, as well as integrate 3 types of L-System grammar: Fractal, Sympodial, & Stochastic.

Initial Plan

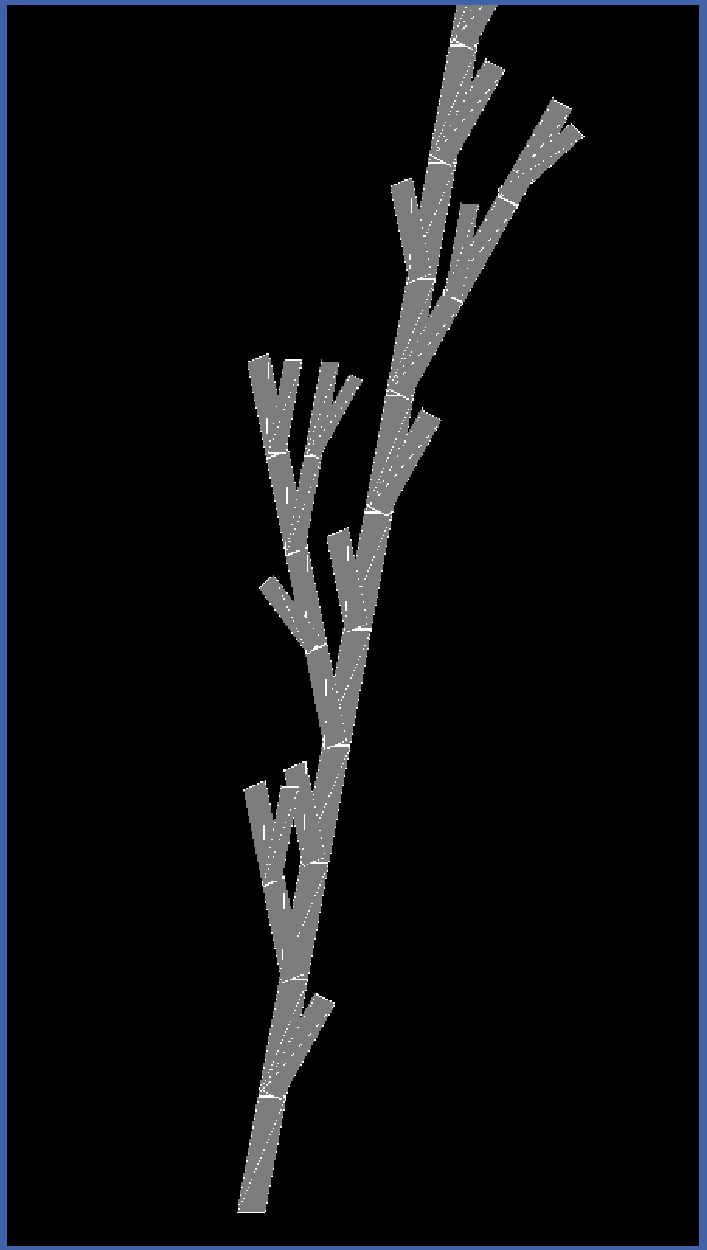
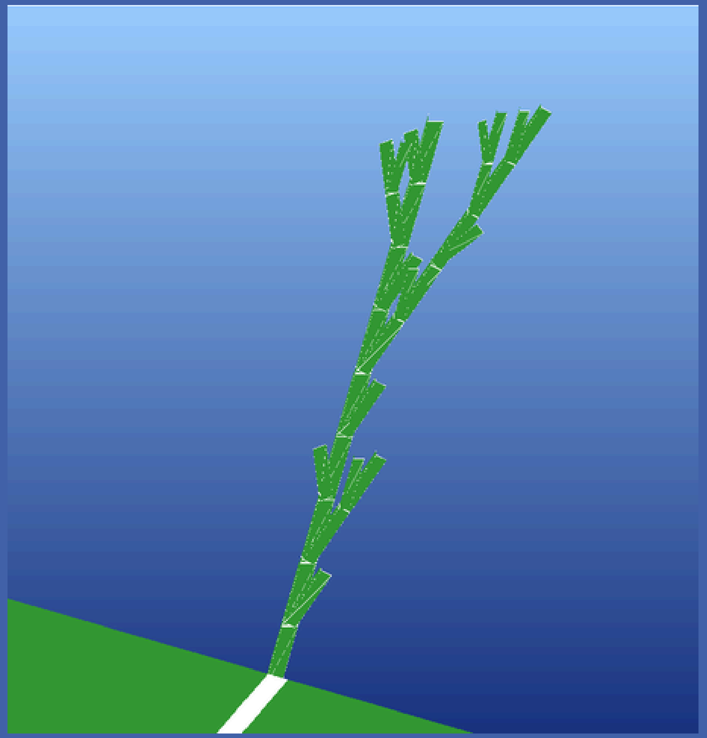

Fractal

Sympodial

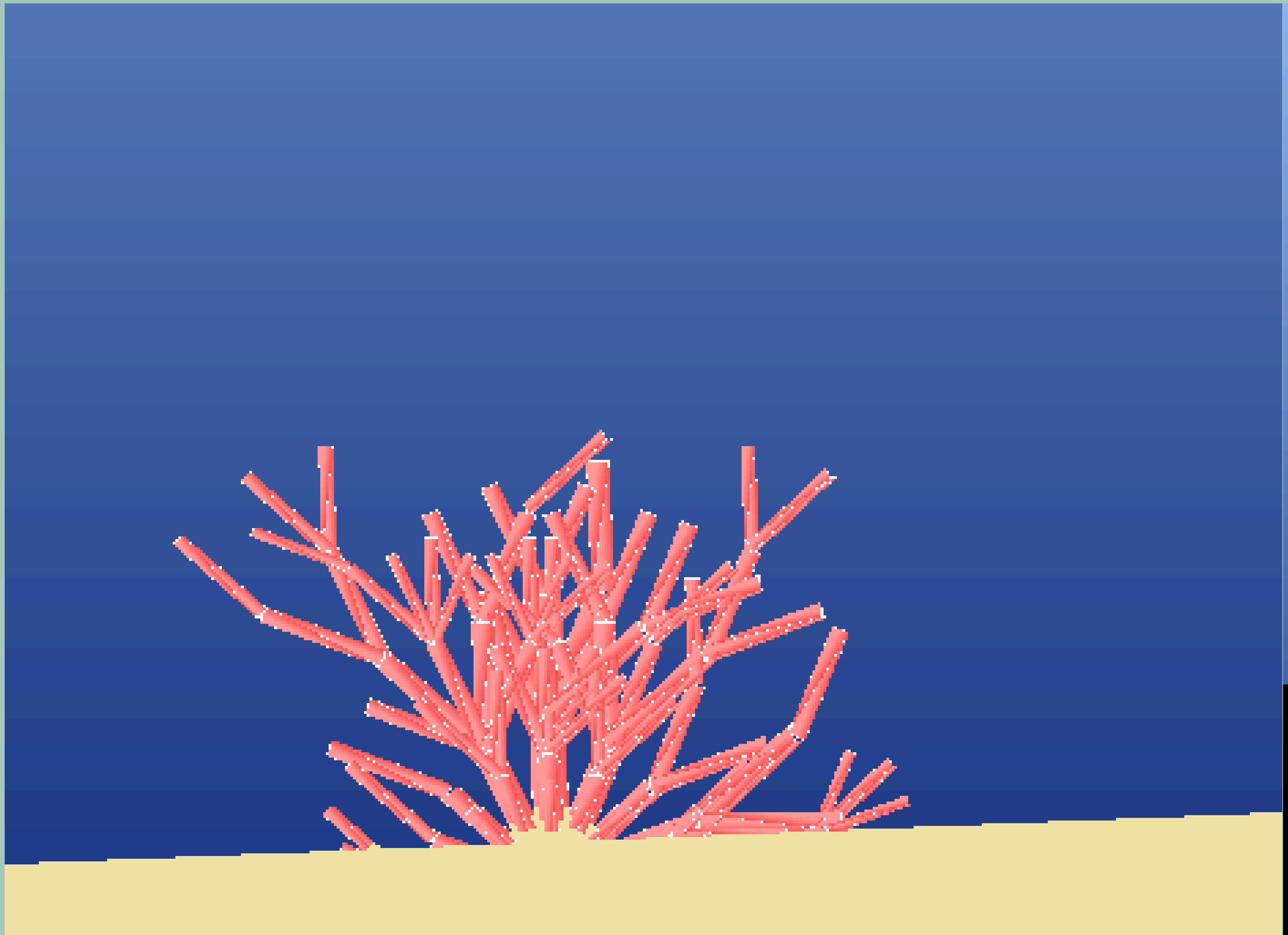
Stochastic

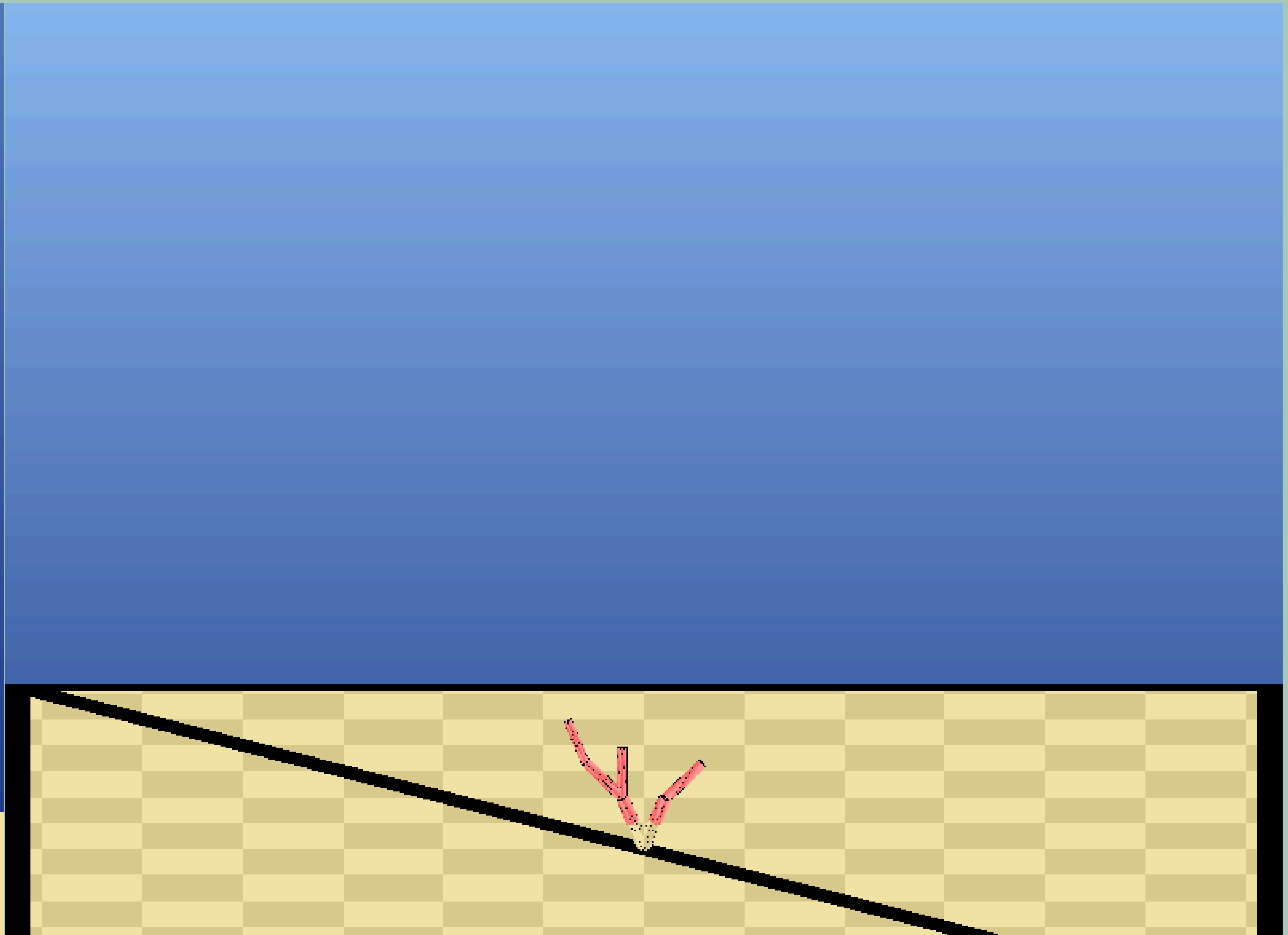
Progress

BEFORE

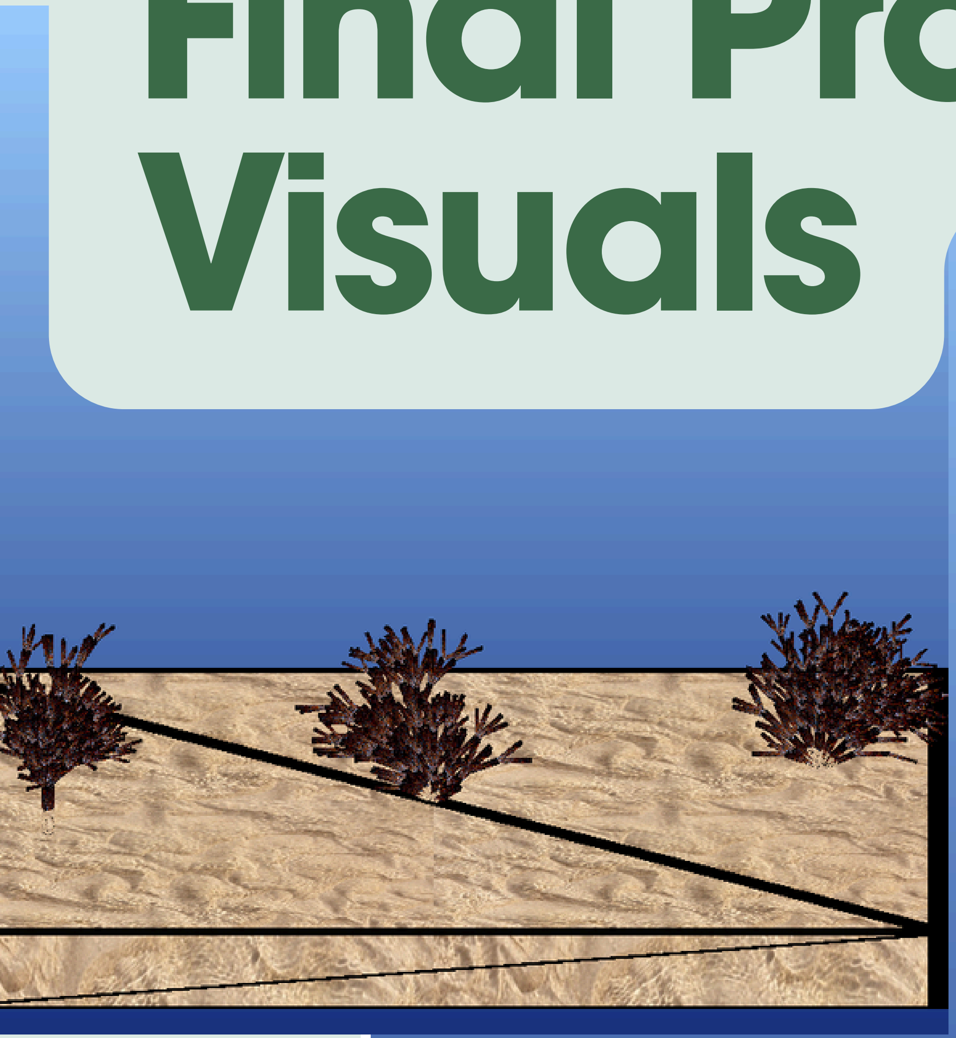



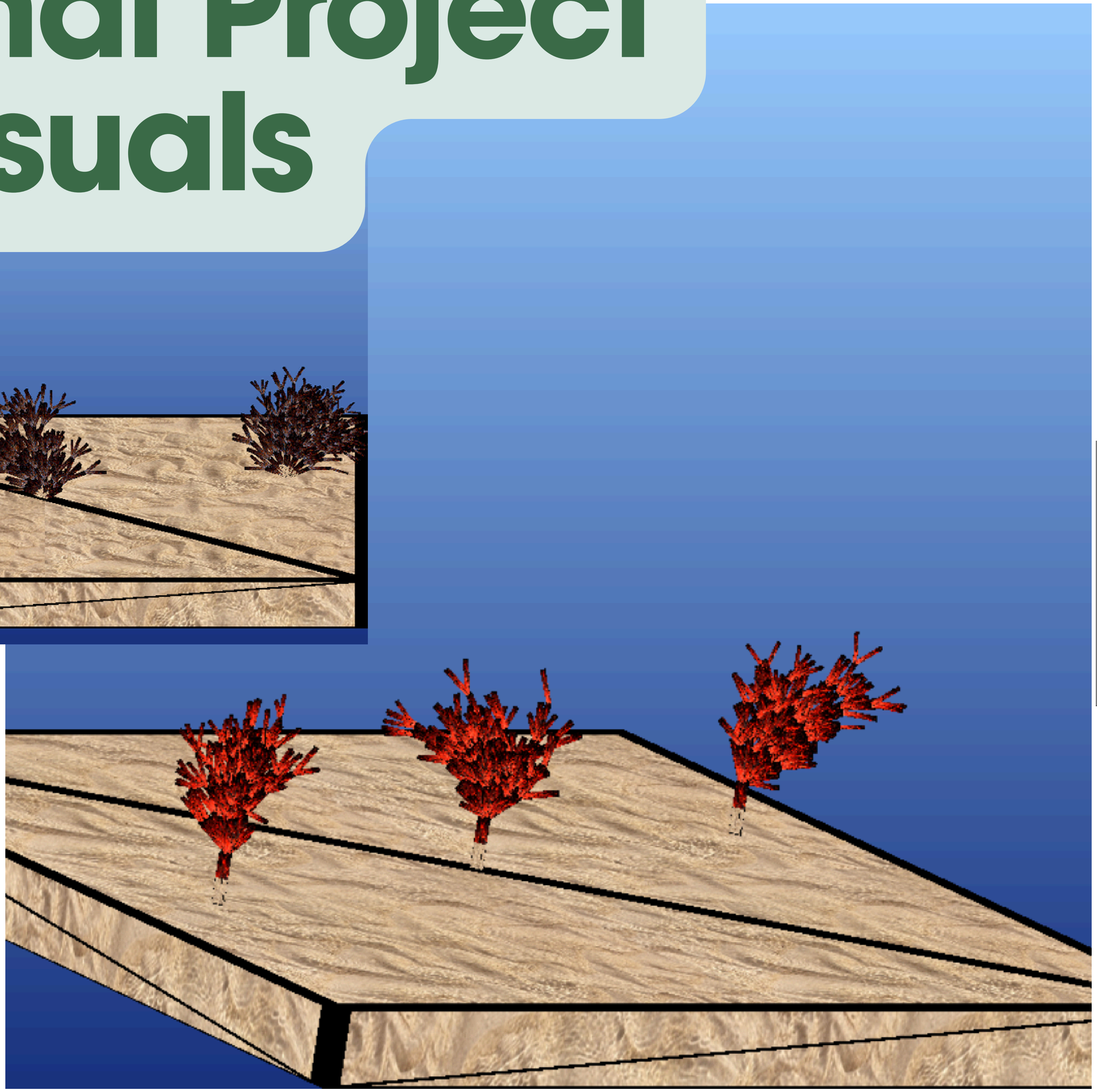
Middle Stages



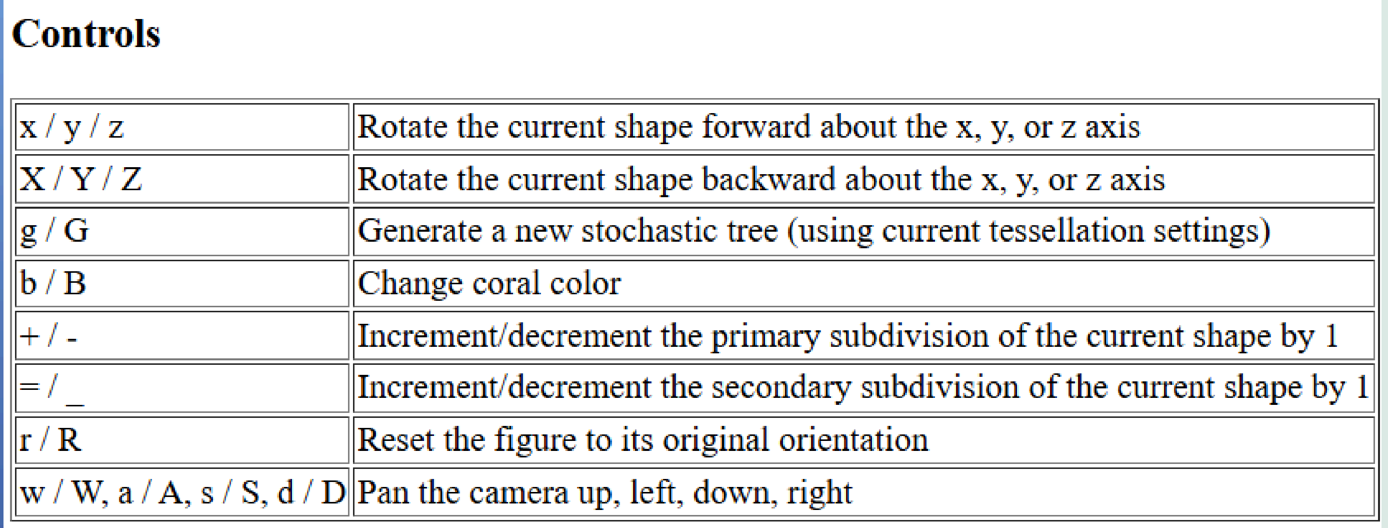


Final Project Visuals





3D, Random, & Textured



Controls

x / y / z	Rotate the current shape forward about the x, y, or z axis
X / Y / Z	Rotate the current shape backward about the x, y, or z axis
g / G	Generate a new stochastic tree (using current tessellation settings)
b / B	Change coral color
+ / -	Increment/decrement the primary subdivision of the current shape by 1
= / _	Increment/decrement the secondary subdivision of the current shape by 1
r / R	Reset the figure to its original orientation
w / W, a / A, s / S, d / D	Pan the camera up, left, down, right

The final coral structures showcase the **intricate growth patterns** generated by L-Systems. Compared to the beginning, the corals grow wider rather than longer and preserve their growth history between iterations. When the texture is added, they become vibrant are no longer dull.

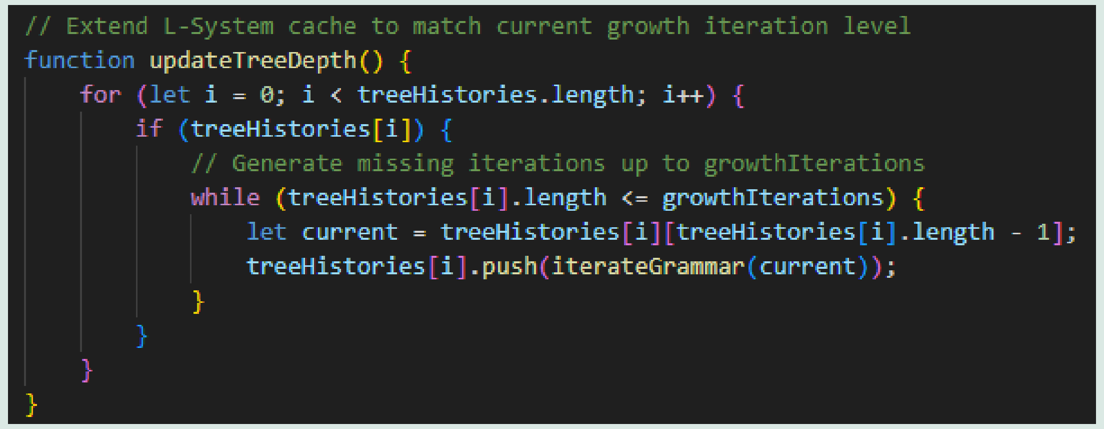
Best Implementations Explained

Highlighting Innovative Code Techniques

- Maintaining Tree Growth History
    - Implemented a grammar history array storing each iteration state.
- 3D Rotation of Branch Segments
    - Implemented incremental axis rotations (rotate X/Y/Z) on a local direction vector each step.
- Texture on 2 Different Models
    - Able to map JPG files to the models we created, including 3 always-random corals

Key Implementations

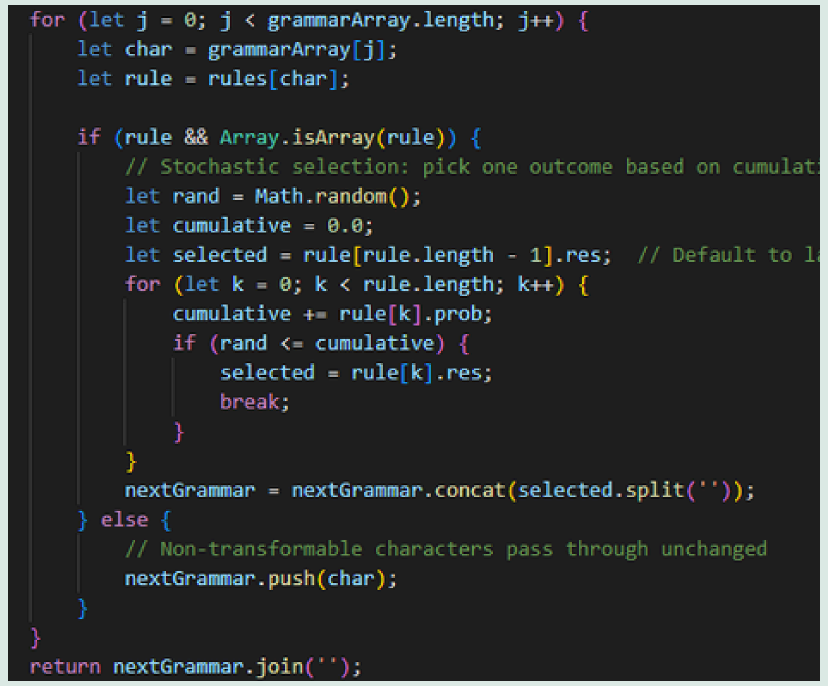
Innovative Code Snippets



```
// Extend L-System cache to match current growth iteration level
function updateTreeDepth() {
    for (let i = 0; i < treeHistories.length; i++) {
        if (treeHistories[i]) {
            // Generate missing iterations up to growthIterations
            while (treeHistories[i].length <= growthIterations) {
                let current = treeHistories[i][treeHistories[i].length - 1];
                treeHistories[i].push(iterateGrammar(current));
            }
        }
    }
}
```

Efficient Recursion

This method minimizes computation time by effectively breaking down the coral growth into simpler recursive calls, allowing for rapid rendering of complex structures with minimal performance impact.

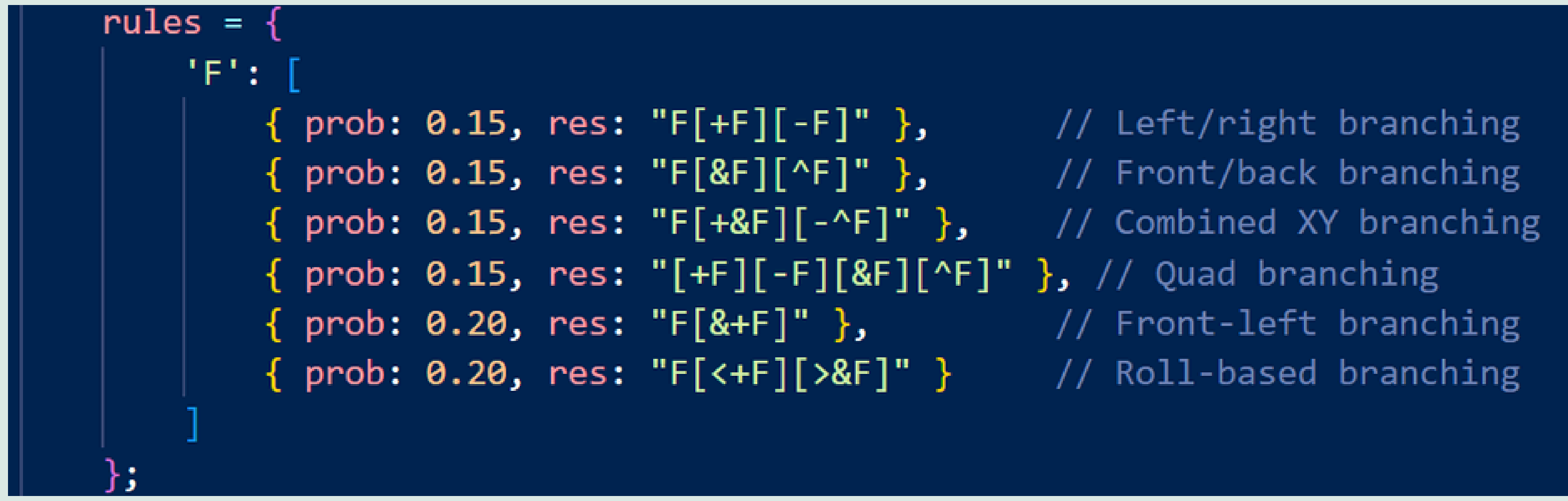


```
for (let j = 0; j < grammarArray.length; j++) {
    let char = grammarArray[j];
    let rule = rules[char];

    if (rule && Array.isArray(rule)) {
        // Stochastic selection: pick one outcome based on cumulat
        let rand = Math.random();
        let cumulative = 0.0;
        let selected = rule[rule.length - 1].res;  // Default to l
        for (let k = 0; k < rule.length; k++) {
            cumulative += rule[k].prob;
            if (rand <= cumulative) {
                selected = rule[k].res;
                break;
            }
        }
        nextGrammar = nextGrammar.concat(selected.split(''));
    } else {
        // Non-transformable characters pass through unchanged
        nextGrammar.push(char);
    }
}
return nextGrammar.join('');
```

Random Branching

Utilizing a stochastic approach, this logic introduces variability in coral shapes, enhancing realism while maintaining control over complexity, fostering the generation of diverse underwater ecosystems efficiently.



```
rules = {
    'F': [
        { prob: 0.15, res: "F[+F][-F]" },      // Left/right branching
        { prob: 0.15, res: "F[&F][^F]" },      // Front/back branching
        { prob: 0.15, res: "F[+&F][-^F]" },    // Combined XY branching
        { prob: 0.15, res: "[+F][-F][&F][^F]" }, // Quad branching
        { prob: 0.20, res: "F[&+F]" },         // Front-left branching
        { prob: 0.20, res: "F[<+F][>&F]" }     // Roll-based branching
    ]
};
```

Optimized Rule Selection

By prioritizing the application of certain rules based on environmental parameters, this technique streamlines the L-system processing, ensuring that the most visually appealing coral structures are generated efficiently. It was fun playing around with the probability and different branchings.

Integrating & Managing Randomness in Coral Generation

How Randomness Is Used

Stochastic Rule Application
Probabilistic L-system rules introduce natural variation in branch angles and growth patterns, use of Math.random().

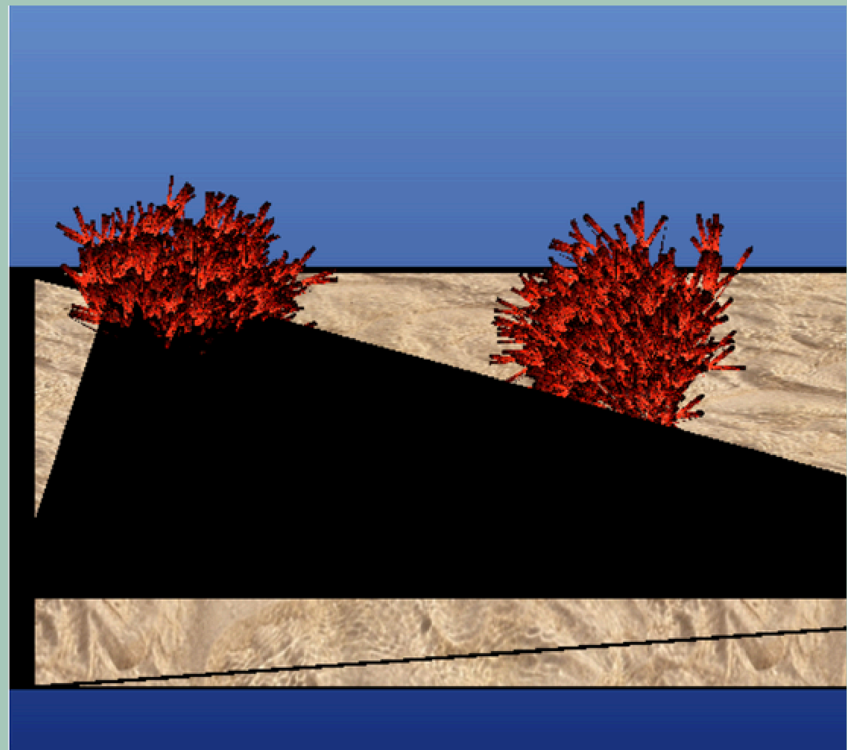
Recursive Growth Logic
Stochastic branching topology and orientation produce complex coral formations, starting at 3 growth iterations but also able to backtrack to 1.

Addressing Some Key Challenges

Consistency
Maintained controlled randomness through seed-based methods for reliable experimentation.

Performance
Managed performance limitations by tuning tessellation subdivision levels (radial/vertical) to balance visual fidelity with triangle count.

Recursion Limits
Prevented unnecessary re-computation by caching L-System history strings. Hardware still limited to around 5 recursion depths.



Future Enhancements

Refactoring and Advanced Models

Enhance the branch geometry by incorporating more surface details, transforming the cylindrical segments into more coral-like forms with natural curvature, texture variation, and surface irregularities.

Code Refactoring

To improve maintainability and readability, refactoring the existing code would be helpful. We currently just follow the format from previous assignments, but if we wanted to expand more on the project, this process would ensure that components are modularized appropriately.

Stochastic Model Advancements

Exploring more advanced stochastic models could lead to greater realism in coral structure generation. Integrating new algorithms will allow for more intricate patterns and variations, significantly enriching visual outcomes.

Shading and Gradients

Implementing dynamic lighting models to accurately simulate surface reflections and depth based on light direction, as well as using Phong shading to simulate realistic 3D illumination. Building upon the fragment shader for 'wet' spots.

Perlin Noise

Implementing vertex displacement mapping using Perlin noise to transform the flat seabed into an organic, undulating terrain. This would require subdividing the ground mesh into a high-density grid to support the geometric detail instead of just 2 triangles.

Thank you!