Abstract:

This artwork creates a shifting 3D landscape that grows and changes in real time. Using custom shaders in p5.js, it builds a field of glowing pillars that constantly transform in response to the viewer's presence. Each pixel's color and texture arise from simple rules, but together they form a surprisingly rich and alive visual world. As you move and interact, patterns rearrange, colors blend in unexpected ways, and shapes take on new forms. The piece becomes a living, evolving environment, inviting you to experience the meeting point of controlled structure and free-flowing creativity.

From Start to Finish: Design, Implementation, and Challenges

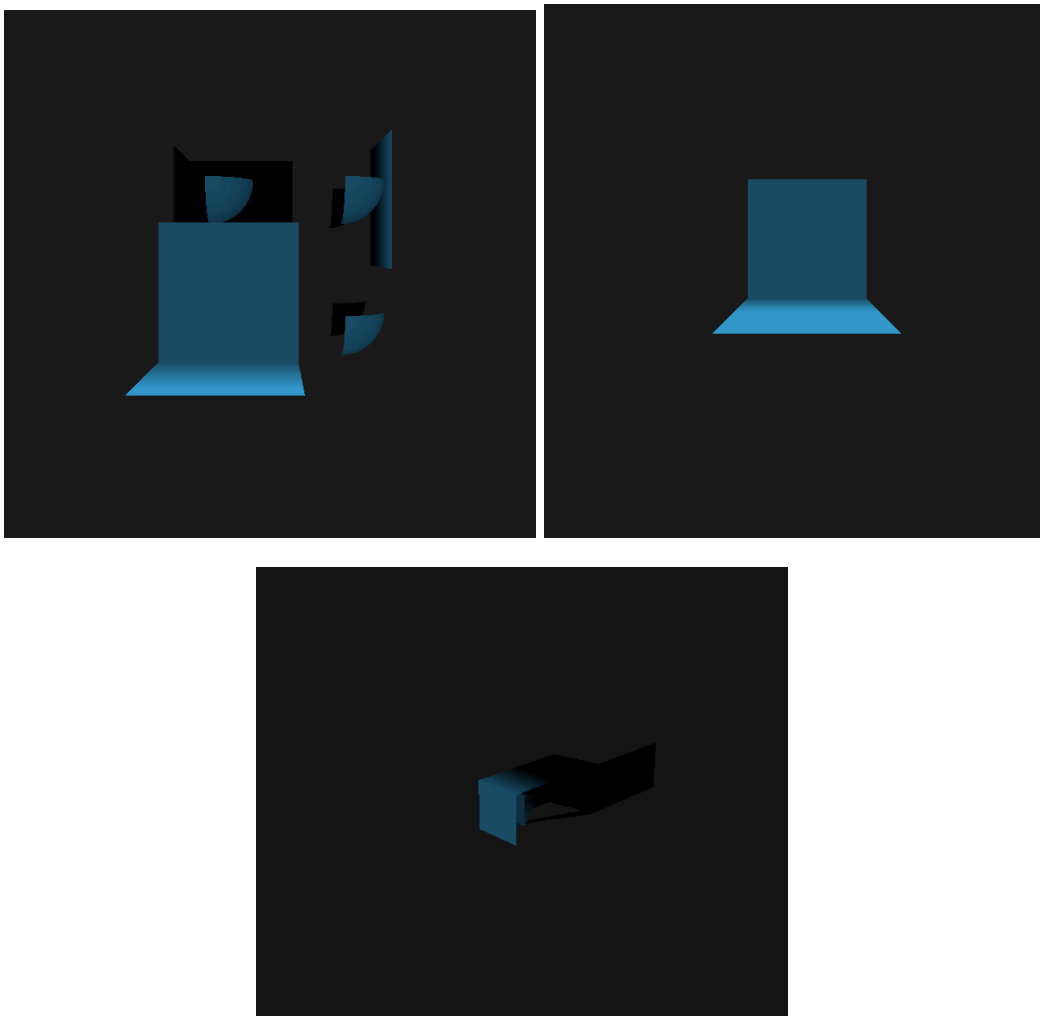
Initial Concept and Goals:

I began this project as a simple p5.js WebGL sketch with the idea of creating a 3D field of pillars. Initially, I wanted an array of rectangular "pillars" rising from a plane, each with some variation in height. My goals included producing visually interesting scenes, integrating dynamic animation, using custom shaders for a holographic or iridescent effect, and eventually exploring interactivity and more realistic lighting techniques.

Basic Setup and Pillar Generation:

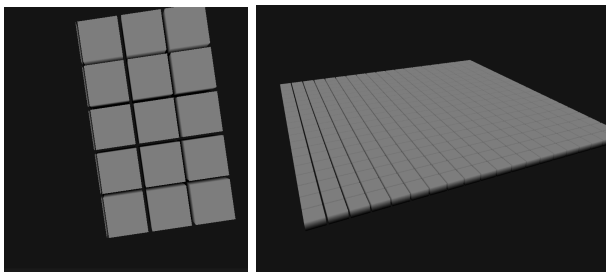
1. **Grid Arrangement:**

   To start, I generated a grid of pillars. Each pillar was positioned in rows and columns, calculated so that all pillars would be centered around the origin. At first, each pillar's height was random within a defined range.



2. **Vertex Data and Normals:**

   I manually defined the geometry for each pillar (essentially stretched cubes). For proper lighting, I computed face normals. This ensured correct shading once I introduced custom shaders and lighting effects.



Introducing Shaders and Holographic Effects:

3. **Custom Vertex and Fragment Shaders:**

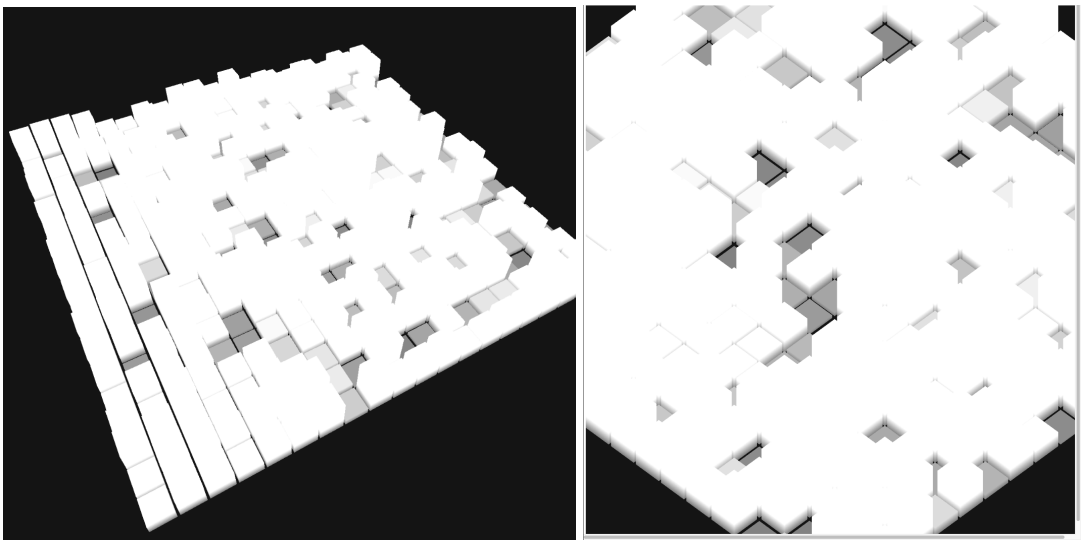
After establishing the geometry, I wrote custom vertex and fragment shaders. The sketch passed position and normal data to the fragment shader, which handled color and special visual effects. I aimed for a holographic, iridescent look by incorporating Fresnel effects, time-based color shifts, and blending multiple colors dynamically.

4. **Color Palettes and Fresnel:**

   By providing a palette of colors as a uniform array, I could mix these hues based on viewing angle and time. Initially, I struggled with ensuring uniform names and types matched between JavaScript and GLSL, but once I got them aligned, I achieved a shimmering, ever-changing color scheme.

Enhancing Complexity and Movement: 5. **Dynamic Heights with Animation:**

Rather than static heights or random changes, I introduced animations. Each pillar smoothly transitioned between start and target heights over a certain duration, using easing functions like `easeInOutQuad`. This gave the scene a sense of organic motion.

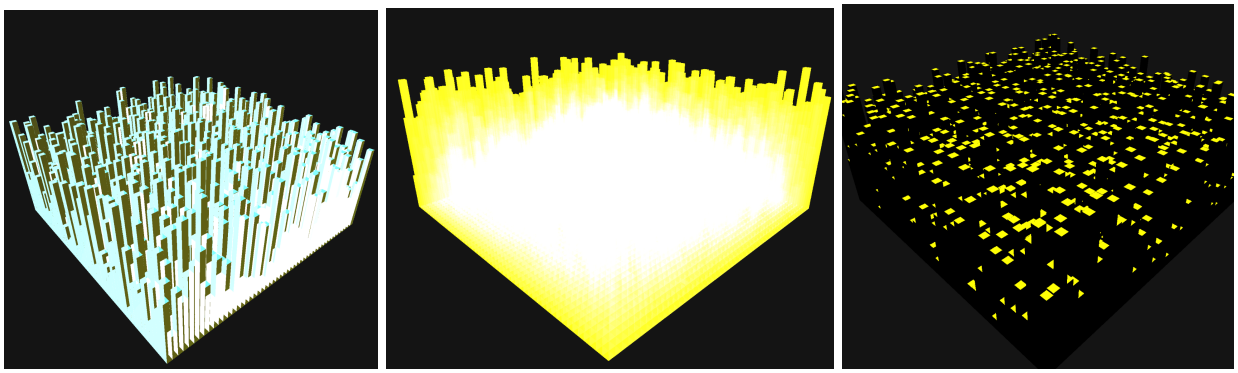


6. **Layered Waves and Patterns:**

   To create more interesting dynamics, I replaced simple random targets with procedural waves. By layering multiple sine and cosine functions with different frequencies and

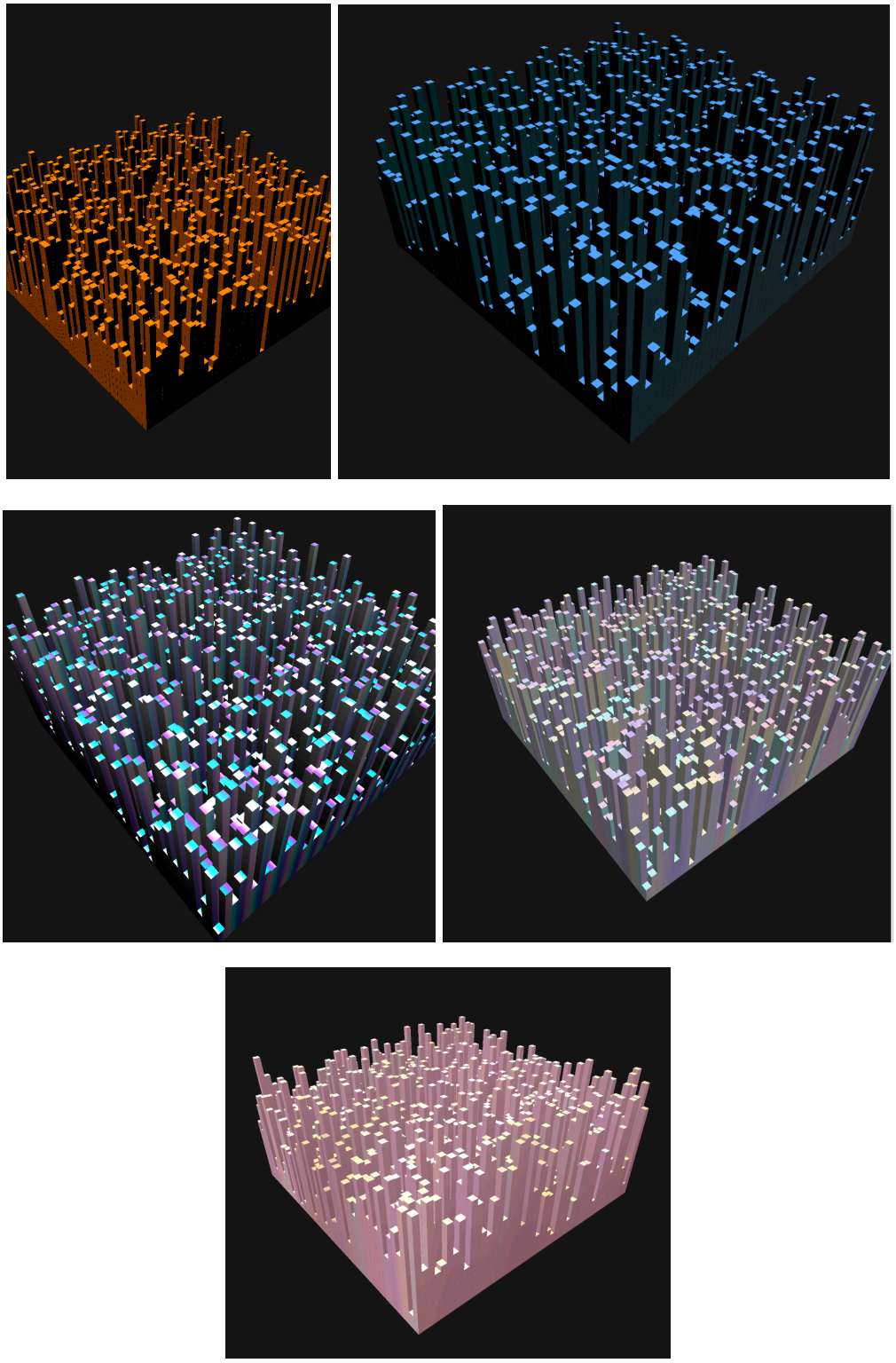
phases, I achieved complex, evolving landscapes. Tweaking parameters took time—I had to experiment until I found compelling visual rhythms.

Realistic Lighting and Environment Mapping: 7. **Directional Lighting and Metalness:** I added directional "sunlight" and experimented with metallic surfaces, controlled by `uMetalness` and `uRoughness`. This gave pillars a reflective quality. Without shadows or environment reflections, the lighting felt incomplete, so I moved on to environment mapping.



8. **Environment Maps:**

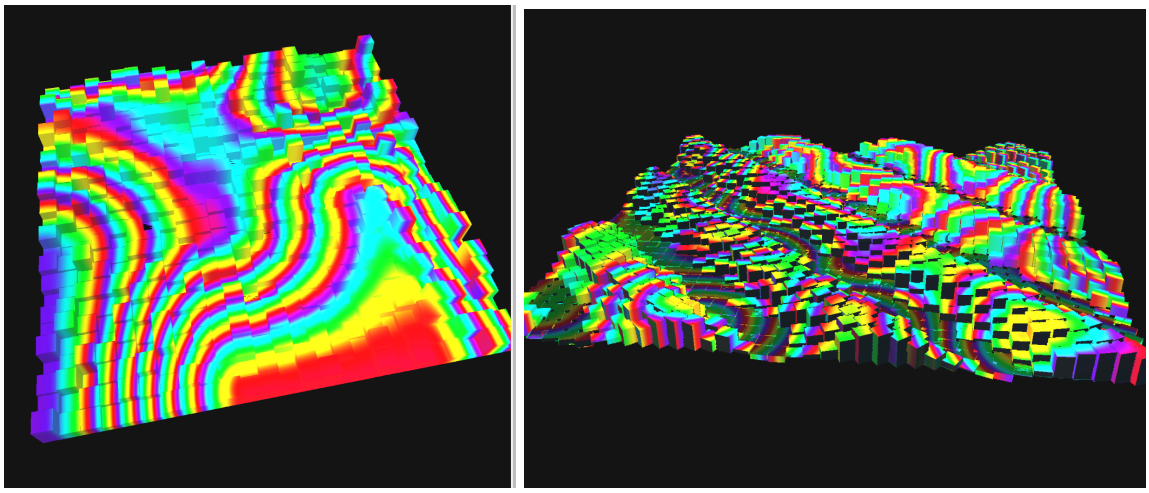
   By loading a panoramic environment map and sampling it in the fragment shader using a reflection vector, I approximated realistic reflections. This step significantly improved the metallic appearance of the pillars. However, success depended on the quality and nature of the environment image and balancing reflectivity with base colors and roughness.



Interactivity and User Input: 9. **Mouse Interaction:**

I incorporated user input to make the scene more engaging. For example, `mouseX` and `mouseY`

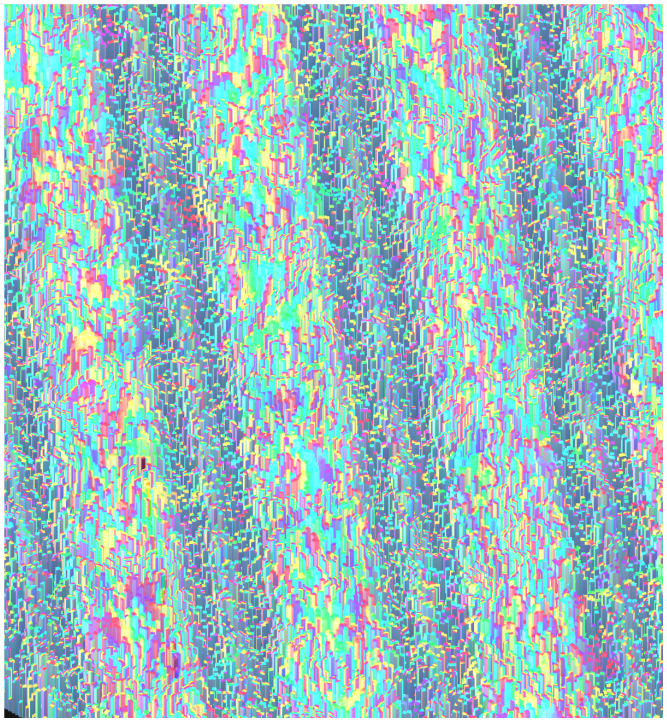
influenced wave frequencies and amplitudes, making the landscape respond to mouse movements. I also implemented a basic picking mechanism: when the mouse hovered over a pillar, that pillar would rise. This required approximating a picking ray from screen coordinates to world space—challenging but achievable with some vector math and assumptions about the camera orientation.



Performance Considerations: 10. **Performance and Rendering Approaches:**

As the scene became more complex—with numerous pillars, ongoing animations, and advanced shading—performance became a concern. Rendering a large, animated grid in real-time could strain the CPU/GPU.

I discovered that by calling `noLoop()` after generating the scene, I could move away from continuous animation and free up resources. With `noLoop()`, the sketch doesn't re-render every frame. This allowed me to significantly increase the number of pillars, resulting in a highly detailed, visually rich static image. This trade-off meant losing the real-time motion but gaining the ability to handle more geometry and complexity, turning the scene into an intricate, static background that would be too costly to animate continuously.



Refinement and Cleanup: 11. **Code Organization and Readability:**
Over time, the codebase grew complicated. I improved clarity by encapsulating logic into functions (`updatePillars()`, `createCube()`, etc.), using descriptive names, and maintaining consistent formatting. This made it easier to understand the flow of data and the sequence of transformations, and it simplified future tweaks and enhancements.

Issues Encountered and Solutions:

- **Uniform and Attribute Mismatches:**
  At first, I struggled with uniforms and attribute handling. Minor errors in naming or data types caused confusing bugs. Checking logs, carefully reviewing code, and ensuring consistency solved these problems.

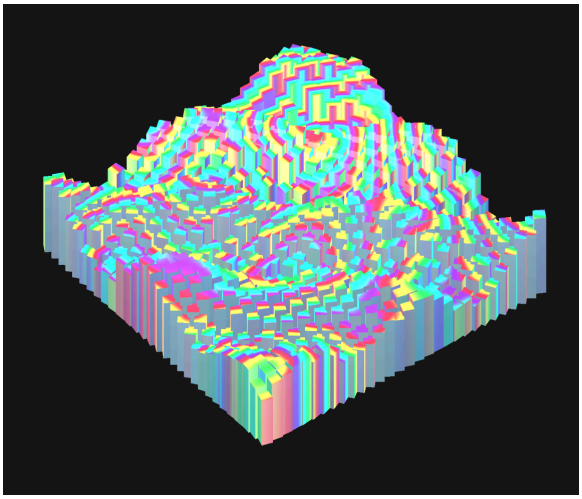
- **Lighting and Realism Limits:**
  Without advanced techniques like real-time shadows or HDR environment maps, achieving photorealism was difficult. I accepted a stylized look and focused on interesting patterns and color effects instead.

- **Raycasting Complexity:**
  Converting mouse coordinates to a 3D pick ray was nontrivial, especially with transformations and camera positioning. Although my solution wasn't perfectly accurate, it worked well enough to let users interact with the scene meaningfully.

- **Balancing Performance and Complexity:**
  Real-time updates and large grids challenged performance. By switching to `noLoop()` and focusing on a single static render, I could achieve a much denser field of pillars, making for an impressive background scene that was not possible with continuous animation.

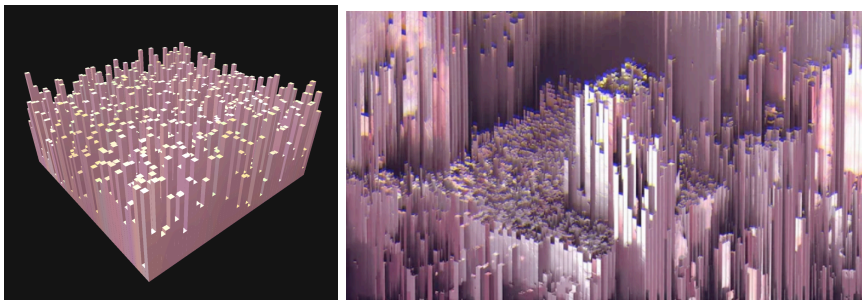
Final Outcome: From my initial concept of simple cubes to a vibrant, interactive, and intricately rendered environment, the project demonstrated the depth and flexibility of p5.js and WebGL. I integrated custom shaders, layered wave patterns, environment mapping, and user interaction to create a dynamic and visually fascinating space. Although I didn't reach full photorealism, I explored a variety of techniques and learned to balance aesthetic goals, performance constraints, and technical complexity.

In the end, I produced a scene that could be both animated (with fewer pillars) or rendered statically (with `noLoop()` to achieve greater complexity). This versatility, along with the wealth of lessons learned, made the journey from start to finish deeply rewarding.



Discussion and Assessment:

In terms of the original goals, which were to create a visually interesting, holographic-like field of pillars with dynamic motion and some degree of interactivity, the project ultimately achieved what I set out to do. I managed to generate a large set of pillars, each animated and shaded with iridescent, time-based color effects. At one point, I had reached a state where the shimmering waves, subtle reflectivity, and interactive controls provided the aesthetic and complexity I had envisioned. I had the holographic feel, the evolving landscape, and even some user-driven variation through mouse movement. In that sense, I had created a dynamic, attention-grabbing piece that demonstrated both technical skill and aesthetic flair.



Still, even after meeting my initial criteria, I felt that it needed more. Rather than settling for the initial version, I continued editing, improving the code, and exploring environment maps, layered waves, and metallic properties. I experimented with environment reflection quality and the distribution of colors and frequencies. This iterative process is common in generative art. Although I was already satisfied at one stage, I found I could continually push the piece further, exploring new parameters, richer waves, or heightened complexity. This speaks to the nature of working on generative art: there is always potential for another layer of detail or complexity that can lead to unexpected and delightful outcomes.

When I look at the project as a complex system, it behaves as one. Numerous parameters—such as wave frequencies, amplitudes, color palettes, lighting directions, metalness, and roughness—interact in subtle and sometimes unpredictable ways. Small changes in a single parameter can create disproportionately large effects on the final appearance. This

interconnectedness means that the scene feels “alive” and emergent, more than just the sum of its parts. Working with these layers taught me that generative systems often require careful tuning and a willingness to embrace unpredictability.

From the perspective of generative art, the piece succeeded in expressing a constantly changing, algorithmically driven aesthetic. Rather than a static sculpture, it behaved like an evolving tableau shaped by code, time, and user input. The interplay of color, movement, and reflection created a hypnotic visual, and the ability for users to influence patterns with their mouse imparted a sense of personal connection. In this way, the project entered the realm of generative art, where I defined rules, and the computer executed them, often revealing surprising outcomes.

On a personal note, I am satisfied with what I accomplished. The work demonstrates a range of techniques, shading, environment mapping, procedural animation, and user interaction, brought together into a cohesive system. While not photorealistic or fully physically accurate, it stands as an example of creative coding that balances technical skill with aesthetic exploration. It showed me that there is always room to expand further, possibly with more refined lighting models, additional forms of input, or integrating advanced noise functions and new geometric forms. Each iteration opened up new possibilities, which is both the joy and the challenge of generative art.

I achieved my initial goals and then pushed beyond them, discovering that a generative system can continually evolve in intriguing ways. I am proud of the progression and the depth reached, and I view this result as a stepping stone to even richer experiments in the future.

References:

Halladay, K. (n.d.). The basics of Fresnel shading. Kyle Halladay - The Basics of Fresnel Shading. https://kylehalladay.com/blog/tutorial/2014/02/18/Fresnel-Shaders-From-The-Ground-Up.html

The book of shaders. The Book of Shaders. (n.d.-b). https://thebookofshaders.com/