Lightning Simulation in Blender

Tong Dai
Team Advisor: Hongyu Wen

1. Abstract

In this project, I am adding a new procedural generator to Infinigen. Infinigen is an open-source project that uses classic graphics techniques to generate 3D meshes, materials, and scenes using Python. Drawing from this, my project aims to contribute to the procedural generators of Infinigen, specifically my project implements a small-scale simulation of lightning paths. The project focuses on replicating the unpredictable nature of lightning paths, along with creating visual effects that capture the intensity and dynamic light emissions characteristic of lightning.

2. Introduction

2.1. Goals

The main objective of this project is to visually render and simulate lightning paths, aiming to contribute to the development of Infinigen and the broader field of computer vision. As the field of computer vision continues to advance rapidly, the demand for 3D data is escalating. Infinigen serves to address this challenge, and this project aims to enhance the capabilities of Infinigen by providing realistic lightning simulations. In addition, with the unpredictable nature of lightning, capturing real-world data for this natural phenomenon is exceptionally challenging. Therefore, by providing a tool for the realistic rendering of lightning paths, this project aims to address the gap in current data collection and contribute to the pool of 3D datasets of lightning.

2.2. Previous Works

Previous research in simulating and rendering lightning paths has primarily focused on finding the balance between physical accuracy and visual acceptability. Unlike most physical objects, lightning paths require a complex approach due to its unique and random formation process and material properties. To render a physically accurate simulation, previous research has utilized particle systems to simulate the stepped leader progression towards ground zero for lightning paths. [1] However, there have been other methods that simplify the process by using pseudorandom generators to create lightning segments. [2] Although not physically accurate, these methods have reproduced visually accurate representations of lightning. There are limitations to these methods as the lightning path and branch structure become highly sensitive to the seed chosen for the number generator. These methods can result in excessive branching of the lightning paths.

2.3. Approach

This project focuses on rendering visually accurate lightning paths by using pseudorandom generators to generate lightning segments and branches. [2] Pseudorandom generators are used for properties of the lightning path, including the branch length, branch angle, and probability of branch forming. As mentioned in Section 2.2, there are limitations to this approach in that the seed ends up playing a crucial role in affecting the structure of the lightning. Therefore, under different seeds, the simulation could work well or poorly.

Specifically, my approach is adapted from the paper "Visual Simulation and Rendering of Lightning" by Samuel Atkinson and Ian Chamberlain. [2] The approach defines lightning branches as a series of straight line segments with varying directions to mimic the jagged appearance of real lightning. Parameters, such as branch segment length, angle, and branching probability are controlled by me, though oftentimes generated by pseudorandom generators, to construct the lightning path. This approach does take into account the stepped leader idea, where there is a main branch where smaller branches are generated from. The algorithm to generate branching patterns is called recursively in order to allow for smaller branches to emerge from the main branch, and by adjusting the parameters of the branch itself, this approach achieves a balance between randomness and controlled branch geometry.

3. Methods

To render a visually compelling and accurate lightning path, there are several key components to my program. Besides the lightning geometry, the material of the object rendered in Blender contains visual effects that I created for my own liking. Hence, there are other ways to implement the visual effects of the lightning path with the shader in Blender as well as the background.

3.1. There is a class Branch that stores the properties of the lightning branches. It contains properties of the lightning branch, and it also includes multipliers to modify these properties dynamically, which allows for varied and realistic branching patterns in the simulation of lightning. The properties of the branch itself are listed below:

- **3.1.1.** **start**: the starting position of the branch
- **3.1.2.** **direction**: the direction of the branch
- **3.1.3.** **distance**: The total length the branch can reach.
- **3.1.4.** **starting_radius**: The initial thickness of the branch.
- **3.1.5.** **branch_probability**: The likelihood of creating a new branch.
- **3.1.6.** **mean_branch_length**: The average length of each branch segment.
- **3.1.7.** **max_segment_angle**: The maximum angle between segments.
- **3.1.8.** **mean_segment_length**: The average length of segments.
- **3.1.9.** **max_branch_angle**: The maximum branching angle.
- **3.1.10.** **rotation_normal**: The normal vector for rotation calculationsHi

3.2. Besides the class Branch, there are two functions that handle the generation of lightning branches. The handle_branching function determines whether a new branch should be generated based on randomness and branch properties. It calculates new branch characteristics, like angle and length, and creates a new Branch instance with these attributes, calling branch_geometry for further processing. The branch_geometry function iteratively constructs the geometry of a branch. It uses a while loop to create new segments of the branch, calculate the positions and angles of the branch, and adds these segments to the Blender mesh.

3.3. In addition, there's a custom shader function named my_shader. It generates two random colors and assigns them to a color ramp node, which blends these colors based on the geometry of the object. The function also includes an emission node to make the object emit light colored by the color ramp. In addition, it uses a sky texture for environmental lighting and combines these effects using a mix shader.

The final shader is connected to the material output, which affects the object's appearance in the rendered scene. This custom shader is implemented according to my perception of the visual effects that a lightning path should have, but it also includes effects that I find interesting and visually appealing.

3.4. Additionally, there is a set_eevee_bloom function that allows for the glowing effect to be rendered on the lightning. This is done by configuring the Blender EEVEE rendering engine and enabling the bloom effect.

3.5. The last component of my program is the set_background function. It creates a new world in Blender and sets up its background. It uses nodes to define the background's appearance, including a shader node for the background and an environment texture node, which is linked to an HDR image for realistic lighting. This setup is then assigned as the current world's background in the scene.

3.6. In addition to the functions and classes of this program, there are parameters that are randomly generated to control the geometry and material of the lightning path. They include:

3.6.1. 'distance': controls the length of the lightning branch

3.6.2. 'starting_radius': controls the thickness of the lightning branches

3.6.3. 'branch_angle': control the maximum angle that the branches can have

3.6.4. 'emission_strength': controls the light emission strength of the material

3.6.5. 'sun_elevation': controls the angle at the light hits the lightning branch

3.6.6. 'air_density': controls the environmental air density of the scene in which the lightning branch is rendered in

3.6.7. 'geometry_mode_idx': controls the geometry of the lightning branch

4. Results

I generated lightning paths using seed = 181913. The program is ran with the command line: python -m infinigen.launch_blender -m infinigen_examples.asset_dev -- --seed 181913 --output_folder outputs/demo --mode param_demo. The parameters that are passed into the program include:

4.1.1. 'distance': np.linspace(3.0, 7.0, num=10),

4.1.2. 'starting_radius': np.linspace(0.01, 0.05, num=10),

4.1.3. 'emission_strength': np.linspace(0, 20, num=10),

4.1.4. 'sun_elevation': np.linspace(0, np.pi * 2, num=10),

4.1.5. 'branch_angle': np.linspace(30, 55, num=10),

4.1.6. 'geometry_mode': np.linspace(0, 3, num=3, dtype = int)

The program does render visually appealing and somewhat accurate lighting paths as shown at the end of this paper from Figures 1 to 4. I did not attach figures that demonstrate the effect of varying branch_angle and geometry_mode as my computer does not have enough memory to generate the outputs. Drawing from the results, I am able to see noticeable differences in the visual effects of the lightning path as the parameters are varied. However, as I ran the program across 10 different seeds, the results all indicate that there is excessive branching towards the bottom of the main branch for

the lightning path. In addition, the smaller branches of the lightning paths are very short, which means that the branches are not spread out across the horizontal direction.

5. Discussion

Overall, the approach we took is promising as it does generate visually accurate simulations of lightning paths. Under the conditions that the parameters are fine-tuned to a certain range, this approach is also computationally fast to generate output images. However, there is follow-up work to be done next. For example, despite the limitations and variability caused by using pseudorandom generators, the parameters passed into the implementation can be and should be fine-tuned to a certain range so that excessive branching can be limited. Additionally, future work could explore integrating more advanced algorithms that may offer better control over the randomness and distribution of the lightning paths so that the branches of the lightning paths can be more spread out in the horizontal direction. In the future, I would like to explore the approach that prioritizes the physical-accuracy of the lightning branch model instead of the current approach that prioritizes the visual-accuracy. Despite the limitations of this approach, I did learn a lot from this project, which includes utilizing Infinigen to build new procedural assets. In addition, this project was my first time using Blender and Blender's Python API, and I found it to be a very rewarding and interesting learning experience.

6. Conclusion

In conclusion, we did effectively attain our goal as visually realistic lightning paths are generated. The next steps would be to explore the different approaches and models that I have not touched upon as well as to revisit the issues that were prominent in the current approach for this project. This includes tuning the parameters to an appropriate range that works for most seeds and adjusting the branching geometry of the lighting path.

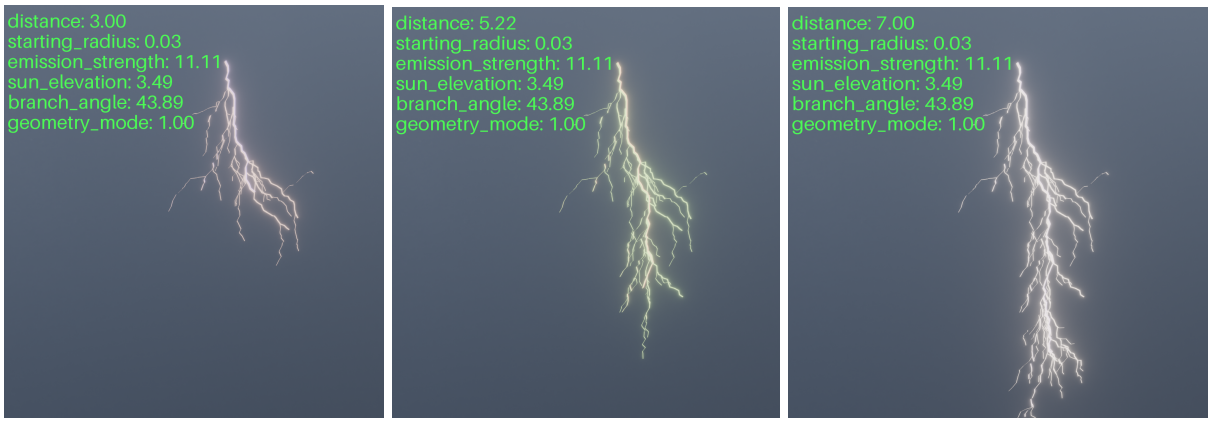

Figure 1, Varying Distances for the Lightning Path

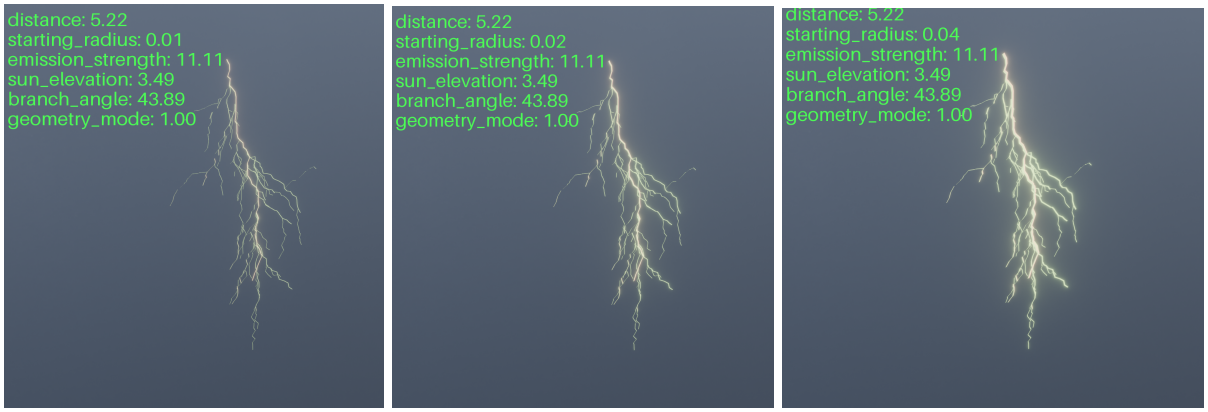

Figure 2, Varying Starting Radius for the Lightning Path

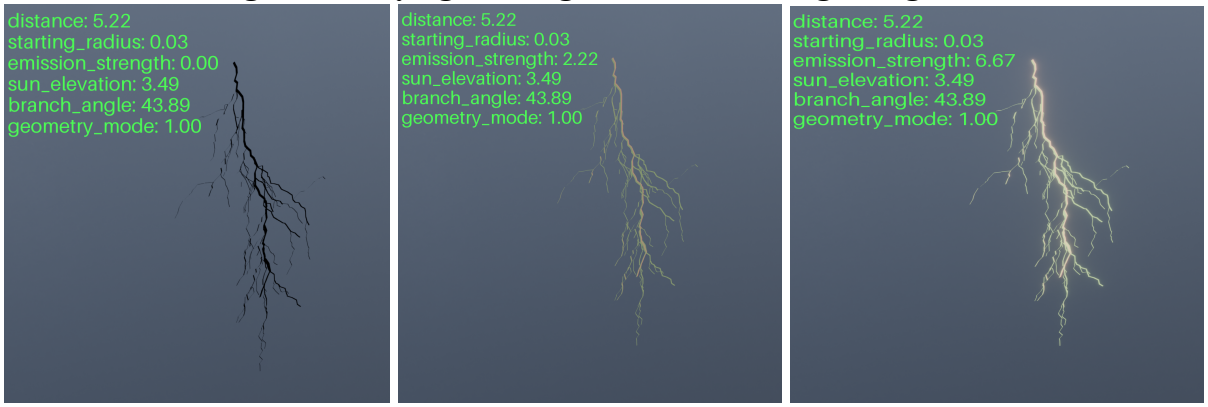

Figure 3, Varying Emission for the Lightning Path

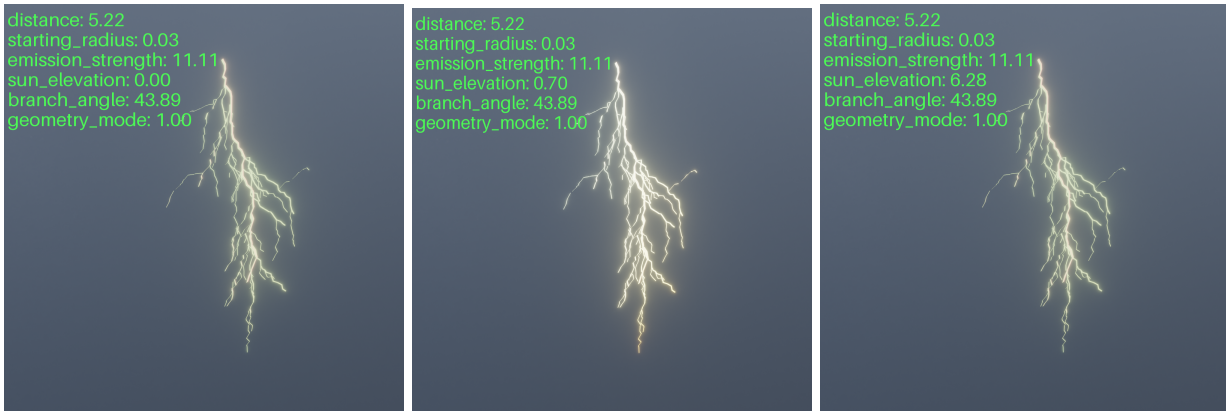

Figure 4, Varying Sun Elevation for the Lightning Path

Works Cited

[1] Reed, Todd, and Brian Wyvill. “Visual Simulation of Lightning: Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques.” *ACM Conferences*, 1 July 1994, dl.acm.org/doi/10.1145/192161.192256

[2] Atkinson, Samuel, and Ian Chamberlain. “Visual Simulation and Rendering of Lightning.” *Rensselaer Polytechnic Institute*, www.cs.rpi.edu/~cutler/classes/advancedgraphics/S17/final_projects/sam_ian.pdf. Accessed 16 Dec. 2023.