Creating a Procedurally Generated Lightning Storm

Arsh Banerjee

December 15, 2023

Project Codebase:
`https://github.com/arsh-banerjee/COS-426-Final-Project/`

Abstract

This project aims to procedurally generate a photorealistic render of a lightning strike on a landscape. The Blender API is used to create a random series of cylindrical branches with varying brightness. For each lightning strike, the lateral spread, levels of brightness, color gradient, and intensity gradient are all customizable parameters. The A.N.T. Landscape is used to create a barren landscape, while Infinigen assets such as raindrops, clouds, and stone textures are used to fully compose the scene.

Asset Renders: `https://github.com/arsh-banerjee/COS-426-Final-Project/tree/main/Asset%20Renders`

1. Introduction

1.1. Goal

The overarching goal of the project was to add a new procedural asset to Infinigen, a procedural generator of realistic 3D scenes. Accordingly, this project aimed to implement a lightning-strike asset to the generator that is developed from scratch using Blender's python API. With the implementation, scenes would be created procedurally, utilizing Infinigen's many features to highlight the asset. In the ideal case, this project would benefit users of Infinigen in their use cases by diversifying the range of assets available. Even without integration, this project would still demonstrate a procedural method to generate long branching structures that could be used in other assets such as spider webs, leaf-webbing, or even cell growth on medium.

1.2. Related Work

There are several methodologies discussing how to create lightning as an asset in the Blender GUI, however, none address it from the context of Blender's Python API. Moreover, many of the methodologies that are discussed involve some human-element of defining the shape of the strike or even physically drawing via pencils what direction the strike should take. These approaches successfully create incredibly realistic lightning shapes as they just physically trace real images of strikes, however, this comes at the cost of human involvement versus procedural generation. There also existed some guides to creating lightning procedurally, however, none were implemented in Blender's python API and were not used for the creation of the asset.

2. Approach & Methodology

Abstractly, the approach for such an asset would include a starting point from which a shape would randomly move towards the ground in staggered steps. Within that process, the shape would, recursively, randomly branch out, once again moving toward the ground. Generally, this approach would work well where a limited number of strikes are required over a large surface. One potential issue is that since the growth and recursion is random, specific strikes could be overly computationally expensive. This would also work best where the scene does not have a center of focus. Because of the randomness, the path of the strike is unpredictable and if a particular subject was being recorded, the scene could become obstructed inadvertently.

The implementation of the scenes displayed below contained three seperate phases. The first involved creating a helper function to create a singular branch from a start point. The second phase was defining the logic in the `create_asset()` function to utilize the helper which was followed by code additions needed to compose the final scene.

The helper function `create_branch()` contains the bulk of the implementation. The methodology used was to create a primative vertex in Blender at the designated start location for the branch. The point was then extruded in a direction defined by a random x and y component and a random, negative z component. The maximum magnitude of the x and y component is passed thru as a parameter. This extrusion process was repeated until the branch reached a "floor" which was passed a parameter of the helper function. This had the effect of creating a line defined by a set of verticies. Once again, using the Blender API, this line was converted to a curve where upon altering the `bevel_depth` attribute of the curve a 3D cylinder was created with a corresponding mesh. It should also be noted that at each iteration of the extrusion, there exists a chance that the corresponding node would be the site of a new branch. If that node were chosen, the coordinates were stored in a list that was passed back to the main function upon completion. This helper function also set the material of the mesh which was an Emission material with a strength and color passed as a parameter.

For this particular piece, there were certainly several possible implementations, of which the most obvious one would entail defining the series of vertices, edges, and faces in numpy and using bmesh to convert these to a Blender mesh. This method would provide more flexibility over the exact geometry of the mesh in terms of the cross section. For instance, the cross section could be hexagon instead of a "circle" reducing the number of faces needed. This method, however, would create larger overhead in the logic needed to define the points, edges, and faces for each segment of the lightning strike. The blender extrusion method simplifies the growth logic at the cost of control over the mesh shape.

The next phase of the implementation included defining the logic in `create_asset()` which involved creating the 4 controllable input parameters and calling `create_branch()` appropriately. The two geometry related parameters were `level` and `lateral` which defined shape related aspects of the strike. Both were input paramters of `create_branch()` where level defined for many levels deep the strike would continue to split. For `level=1`, there would simply be one branch from the top to bottom with no branching whereas `level=3` would have the main branch plus two further levels of branching. The `lateral` dictates how much movement the strike makes in the x and y components, thus affecting the "spread".

The two paramters that controlled material properties were `decay` and `temperature`. The `decay` parameter ranged from $(0, 1]$ and was used to make the appearance of the stike more realistic. Just as real lightning strikes reduce their brightness that further out it branches, `decay` dictates the emission strength as the level of the branch increases. Specifically, the strength S is defined by $S^{decay \times level}$. Thus, the rate of decay determines how quickly the brightness falls off. While `decay` alters the emission strength of the material, the `temperature` parameter alters the color of the material as the level increases. It should be noted that neither parameter simply changes the whole's objects brightness or color but does it accordingly to the level of the branch. From a visual analysis, it appeared that the main branch of a strike was white while ones furthest from the main appeared to have purple or blue tints. Accordingly, the `temperature` parameter controls the rate at which the color transition from white to blue as the branch continues further out.

Lastly, the scene had to be composed to portray the strikes as realistically as possible. The scene was modeled after someone looking out at the horizon on a dark night. As such, there was no sky lighting and the Infinigen asset `kole_clouds` was used to impply a stormy night. The Infinigen asset `RaindropFactory` was also used to create rain droplets to further emphasize the effect. The landscape utilized the A.N.T. landscape plugin to create a large, noisy mesh for the terrain. Afterwards the Infinigen asset `stone` was utilized to apply a texture to the terrain. The landscape could have been created using a variety of methods, ranging from creating meshes for the terrain to utilizing Infinigen assets to create the entire terrain mesh. The A.N.T. landscape plugin was chosen because it remained computationally inexpensive while still utilizing Infinigen assets and adding to the horizon.

3. Results

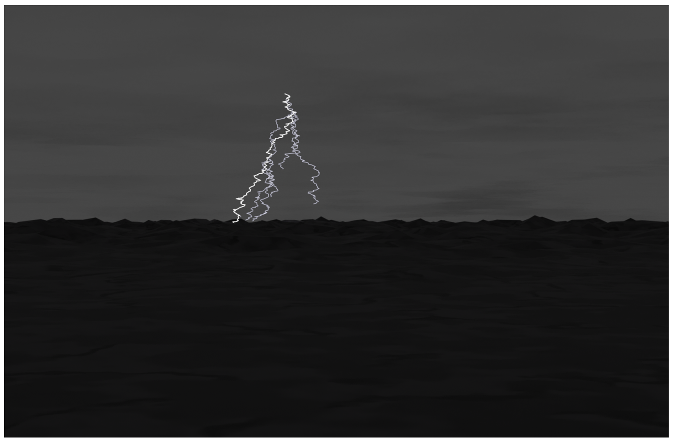
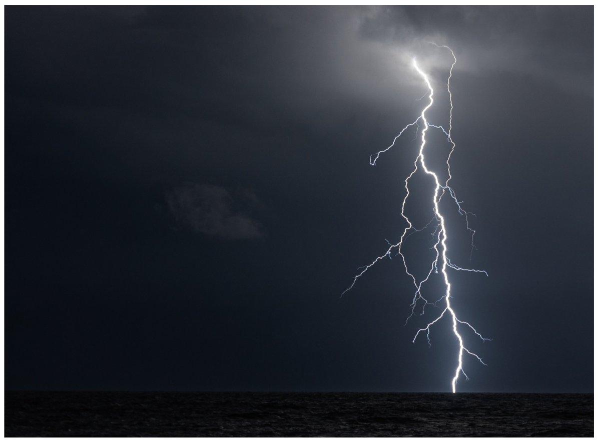
The most straightforward method to determine the level of success is a comparison to a real photograph of the phenomena. Another important metric is also the added computational requirements from this singular asset. In rendering large scenes, the computational requirements are critical to rendering quality and time. Accordingly, the implementation of this asset should attempt to optimize in this aspect. To begin, let us analyze the increase in rendering time due to the asset. In order to determine this metric, the scene was rendered with the asset, without the asset, and without the asset including an added light source. The time needed to render each scene is described below:

Scene	Time [s]
With Asset	59
Without Asset	29
Without Asset with added lighting	37

Becuase all the lighting in the scenes below come from the lightning strike itself, removing is drastically reduced the time needed to render. Once the sky lighting was added, the render time increased, however, the lighting strike still adds at a minimum, 22 seconds per asset usage. The renders related to the four parameters are included here:

`https://github.com/arsh-banerjee/COS-426-Final-Project/tree/main/Asset%20Renders`

The following aims to compare the asset to a real photo of lightning taken from unsplash to assess the success in recreating the natural phenomena.

4. Conclusion

The shape of the lightning stike does not exactly match the real phenomena, however, with some specific parameter settings it does come relatively close. Tweaking the randomized movement of the strike may help the asset become more photorealistic. The next steps would be to explore further logic of the random movement to more closely match the real phenomena. Another interesting aspect of this project to explore would be to see whether different cross section shapes would reduce the total time needed to render each strike asset. Currently, the cross section is a circle which increases the total number of faces, whereas other shapes may help optimize this metric. Broadly speaking, the next step for the asset would be to create a lightning factory that could create multiple strikes associated with stormy clouds. It would be unrealistic for users to render and place each strike individually, however, creating a factory similar to that used for raindrops could make the asset more usuable.

5. Honor Code

I pledge my honor that this paper represents my own work in accordance with University regulations
/s/ Arsh Banerjee