Volume Lightning Rendering and Generation Using L-Systems

Chris LaPointe Devin Stiert

Advanced Computer Graphics 2009

Rensselaer Polytechnic Institute

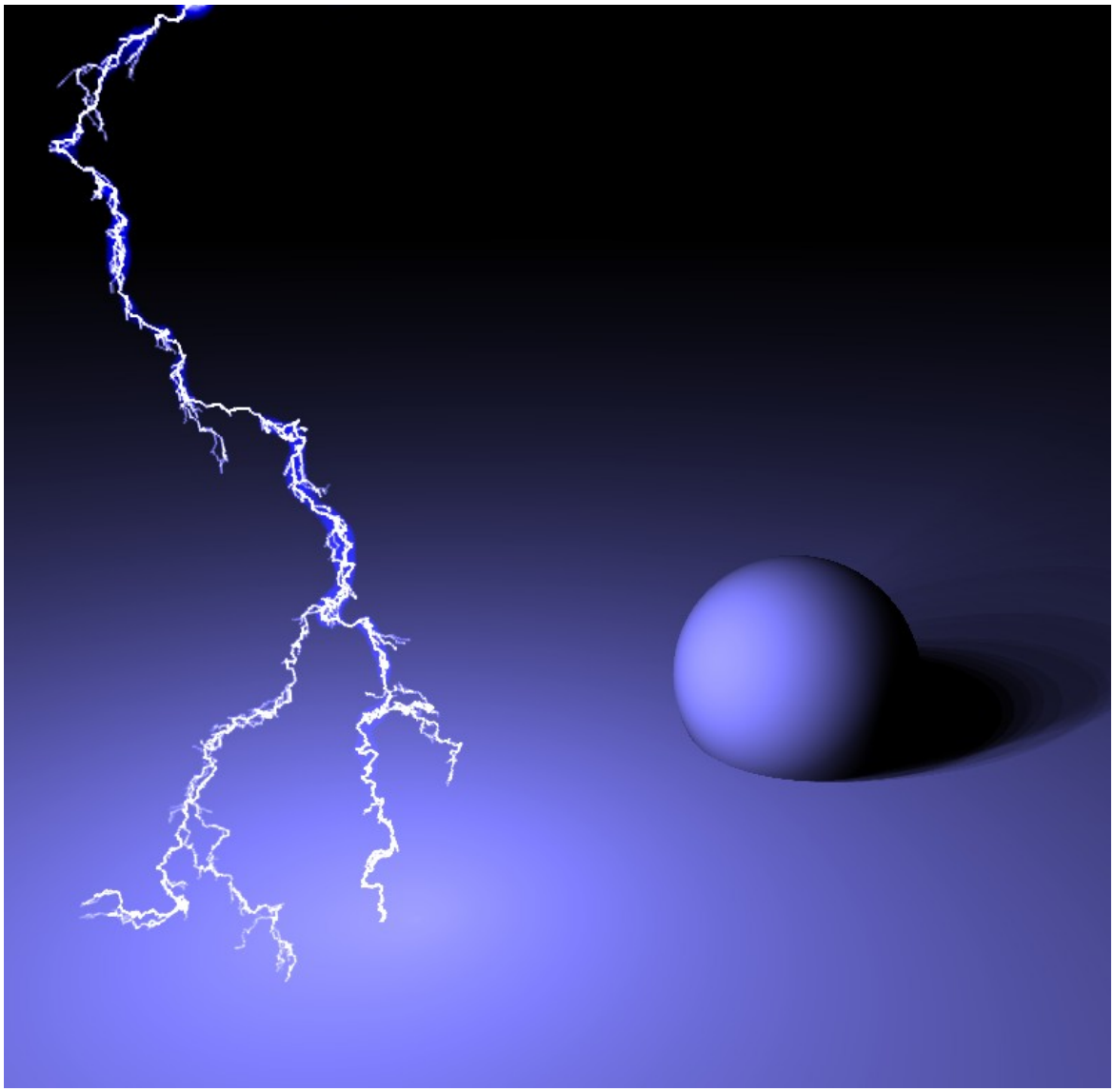

Figure 1: Rendering of volume lightning

Abstract

Our paper takes advantage of a volumetric data structures and L-Systems to dynamically and randomly generate bolts of lightning. After volume generation has been completed, a ray tracer is used to step through the voxels within the volume to render the texture in an environment with appropriate lighting that affects the scene.

Keywords: lightning, volumes, ray tracing, L-Systems

1 Introduction and Motivation

Lightning is something that is not significantly studied within the graphics field of computer science. Recreation of natural phenomenon has always seems to catch the interest of researchers in the field, so why not lightning? We provide a realistic method to generate and render lightning within a scene.

Our method of generates three dimensional lightning using a L-System to generate the fractal looking pattern randomly, following a given path, and then using a ray tracer to render the result. In order to hold on to detail from one step to the next, the result of the L-System is stored within a volume texture. Even though a volume texture can be quite large, it has its benefits, which will be discussed later in the paper.

In addition, a lighting technique is provided in the paper to demonstrate how a volume can affect the diffuse (and potentially specular) surfaces within a scene.

2 Background Readings

Niemeyer et al. [4] developed the dielectric breakdown model, or DBM. This model can describe a number of different natural phenomena including lightning. It uses the Laplace equation to assign probability of the path of the modeled phenomena to pass through each position on a grid.

Theodore and Ming modify this model for the specific use of generating lighting[2]. They later improve on their technique using an improved rendering method[3]. While physically accurate, their technique looses realism in rendering due to its obvious artifacts and flat appearance.

3 Rendering and Generation

For our software, rendering is processed in two completely separate steps (and completely different programs for that matter). This acts as an ease in the implementation, a render-time saving strategy, and a scalability model to the application. The first program(3.1) renders a lightning "fractal" using a simple L-System technique. The second program(3.2) takes the volume generated by 3.1 and renders it as a volume with appropriate lighting techniques.

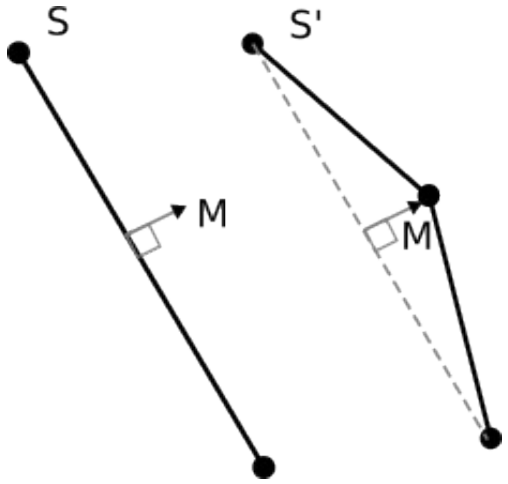

Figure 2: L-System applied to a segment

3.1 Generation Algorithm

Several algorithms were attempted to generate realistic looking lightning. On the third attempt an L-system technique was used, similar to that described in [1]. This technique is settled on for its ease of implementation, speed of calculation, and realistic results.

The method described in [1] is meant for a two dimensional game-like scenario where lightning needs to be generated on the fly. Our algorithm took ideas from their algorithm and expanded it to the third dimension, as well as improving upon its results, since speed was not a main concern.

The first step our algorithm is to take a line segment, *S*, that starts at the source of the bolt, and has a destination of the end of the bolt, either as specified by the user or automatically generated. Then, using rules provided by the L-System, the line segment is subdivided at the midpoint and displaced at an angle perpendicular to itself, *M*, which has a random magnitude, to form *S'* (*Figure 2*). The algorithm then repeats for both line segments in *S'* until the number of levels of detail, *n*, is reached. On each iteration, the maximum random magnitude that *M* can be is halved. The result will ultimately be very similar to Perlin noise (*Figure 3*).

This method works well for the main shaft of the lightning

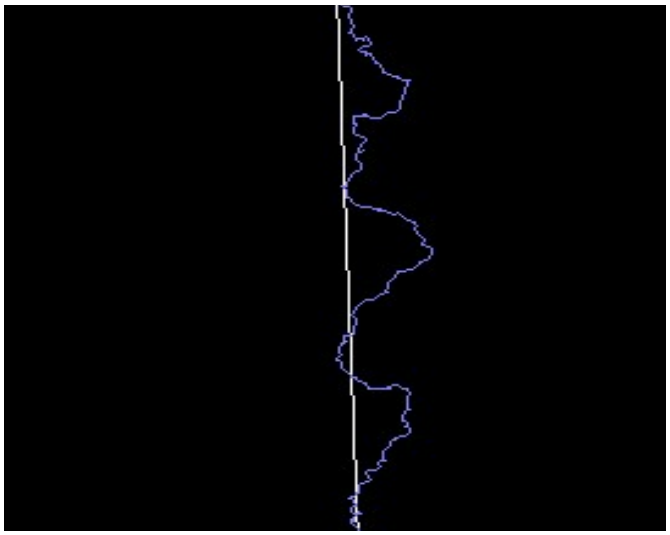

Figure 3: Line Segment after L-System Computation

strike, but does not provide the splits and bolts that are commonly seen coming off large charges of electricity. In order to provide this, there is a probability $P_{B'}$ which determines whether or not a segment will have the additional *B'* offspring (*Figure 4*). A good value for value for P_B turned out to be around 0.5. Once a bolt *B* has been created, it is treated just like any other line segment in the algorithm, and has its own probability to subdivide and be offset. In addition, the intensity *I*, of *B* is half that of *S'*. The intensity will be discussed later, and primarily affects the color and alpha of the rendering.

The series of line segments generated from the L-System is simply drawn directly to a volume buffer with size WxHxD. Once the volume buffer has all the segments from the L-System, further operations can be applied to it.

Once the line segments are fully created to a sufficient visually appealing level, we add color and glow to them. In general, the main beam of the light has the color C=(0.5,0.5,1.0,1.0) in RGBA, and the branch has the color C*I. This puts the branches slightly in the background compared to the main bolt.

After applying color, we run a blur on a separate copy of the image using a Canny-Deriche filter. Then the image is brightened several ordered of intensity until it is visible. This "glow" image, G, is then simply just added to the bolt image. This provides a nice "glow" or "halo" effect around the lightning and brings out highlights in large clusters of electricity.

The last step is for efficient and space requirements only. The image is then flipped for compatibility with the volume ray tracer, and also cropped so that only

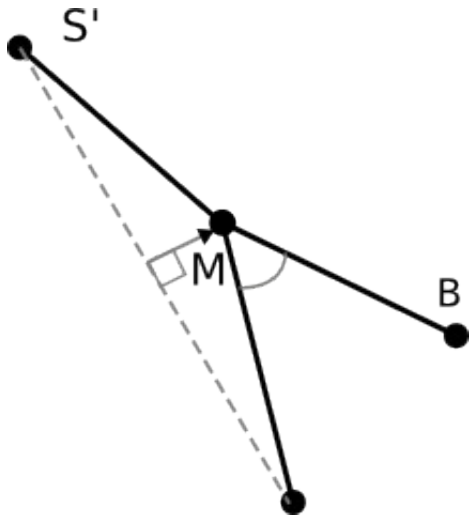

Figure 4: Branch, B, splitting off after a segment has been split.

brightened pixels are saved in the image file, both saving memory and reducing render time.

3.2 Rendering Volumes

The volume tracer is an extension to a basic ray tracer. After loading the volume file, the ray tracer first generates an approximation of the light sources in the volume. Ideally each voxel in the volume would be treated as a light source, but this would cause a massive slowdown in rendering for a very small increase in accuracy. To approximate the lights the volume is scaled down to a more manageable size and the colors are linearly interpolated. We found that around a 64^3 light cube map was the largest our systems could handle due to memory constraints but gave us acceptably realistic lighting. The tracer scans through the scaled down volume and adds any points with a brightness above a specified threshold into the scene (*Figure 5*). By only adding the brighter lights to the scene we save a substantial amount time that would be spent tracing to lights with little to no contribution to the final color.

During the tracing stages the volume is treated as a AABB (Axis Aligned Bounding Box). We choose to implement a very simple AABB-ray intersection algorithm. Faster, more complicated algorithms exist, but we felt the speed increase would be insignificant compared to the amount of time spent tracing through the volume and the amount of time it would take to implement. In addition to keeping track of the t value, or distance from ray origin to an intersection with an object, the distance to the exit of the object is stored as well, called *tfar*. To determine the color of a pixel whose ray intersects with the AABB the ray

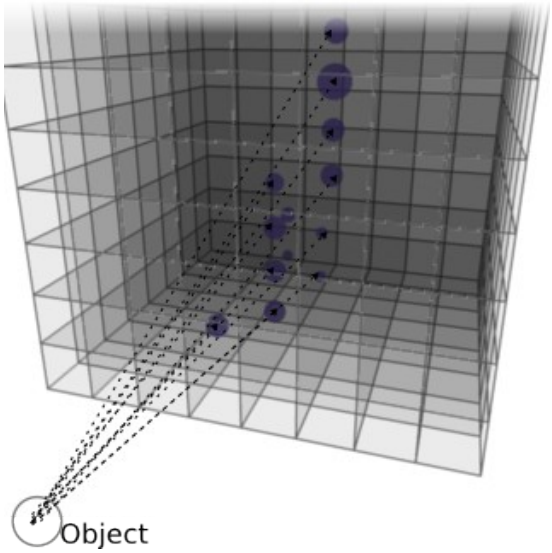

Figure 5: Calculating light from an object looking at simplified voxels in the lightning

tracer starts at the point marked by *t*, and steps through the volume by a delta (usually .25 units) of a unit at a time until it reaches *tfar*. At each step the values of the closest points are interpolated and summed as the tracer progresses.

We choose an additive color model, using the equation *diffuse += light_color*Δ* where the delta is used to step the strength of a light. We chose this method because our volumes are purely emissive with each particle only emitting light and having no diffuse or specular components. After the volume is traced through, a secondary ray is shot out of the back of the AABB and the resulting color is added in. This accounts for the transparency of the volume.

4 Results

To render the volume image with the size of 256^3 with one lightning bolt (one original segment, eight levels of detail), on a 2.6 GHz Pentium IV machine took 40.4 seconds. Simply increasing the number of bolts does not double the time, simply taking 40.9 seconds when two bolts were computed. The solution is very scalable when any number of lightning bolts. The primary time consumption, over 60%, comes from the Canny-Deriche blur filter to calculate the glow map.

Memory is often a major issue with this implementation. If the image has the size of WxHxD with RGBA, then the size of memory must be 8*W*H*D. The memory overhead comes from the fact the image is RGBA, and the glow filter needs to store a second copy of the image.

Ray tracing a volume takes significantly more time. The scene seen in (*Figure 1)* takes 523 seconds to render on a Core 2 Duo 2 GHz system with 2 GB of RAM. The majority of memory used by the program is taken up by storing the volume texture within memory, which is *4*W*H*D*.

5 Future Work

This solution comes out with decent results, however, is not perfect. Some of its main issues lie in its memory consumption and render time. Reduction of these resources would be possible with more time and research into the project.

5.1 Difficulties in Generation

As mentioned in 3.1, the L-System technique to generate bolt "fractals" was used only after trying two other techniques. The first technique that was attempted was the physically based method as described in [2] and [3]. Time, knowledge of physics, and resources went into our inability to implement this method. As realistic as the equations were, the method in the paper did have its flaws. Their implementation limited the result to a two dimensional image, and often had artifacts where the image was placed in the rendering. This method was abandoned after realizing its mathematical and implementation complexity.

The next attempt was a common method using Perlin noise differences and gamma correction. The idea was to make two passes at Perlin noise, subtract the differences, invert the image, and apply a high value gamma filter to it. This produced electricity-like results, but had the problem that it could not isolate a single bolt. In addition, when the algorithm was extended to encompass the third dimension, it no longer looked like any electrical field, but rather, it appeared to be caverns. After applying more filters and trying to isolate a certain region to turn into a single bolt within the 3D field, the method was again abandoned for its inability to produce convincing results.

The method we chose to go with using L-Systems was relatively straight forward to implement in two dimensions, and was not hard to extend to three. One of the main difficulties was picking probabilities and intensity ratios to provide a believable result. Regardless, this method had overall better performance and results than the previously attempted methods.

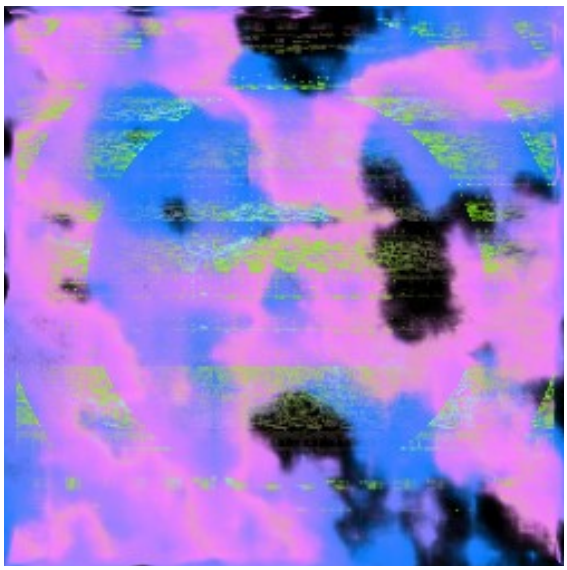

Figure 6: Rendering of a Perlin-generated volume showing artifacts with a low delta

5.2 Difficulties in Volumes

We ran into several difficulties in creating the volume tracer. First we saw strange noise in circular groupings (*Figure 6*). We determined this to be caused by too large of a Δ step while tracing through the volume. If the Δ was too large it would skip over multiple voxels while tracing through at large angles. Reducing the step to much smaller than the resolution of the grid fixed this.

There were also some issues determining what type of color mixing to use. We experimented with both additive and multiplicative colors. We decided on an additive technique because we felt that would best simulate the real behavior of a lightning bolt. To be extended to render non-emissive volumes other coloring techniques would need to be implemented.

The tracer also does not deal with the case of other objects intersecting the volume. Currently in this case the objects intersecting the volume are 'clipped' by the AABB containing the volume. Similarly any objects with a surface flush to one of the faces of the AABB will suffer from a noisy incorrect appearance due to floating point errors. We simply offset other objects off of the AABB slightly.

5.3 Improvements

There are many possible improvements to the method this paper describes. Memory is one of the biggest issues. A 512^3 image takes 8 GB of memory just to generate! One of the major improvements that could be made to the bolt generator is to simply take less memory by intelligently choosing a volume size, or completely abandoning the volumetric method of storing lightning all together. This would also increase the running time of the program because the Canny-Deriche filter only needs to go through a fraction of the pixels to generate the glow map.

Speed increases for the generator could also be found by switching to a more native library rather than relying on CImg library's image abstraction classes, which are heavily functional, but bloated [6].

6 Conclusion

Our method renders realistic lightning successfully within another scene. Lighting affects are provided, and can relatively quickly (compared to rendering) generate new lightning bolts and save them to a volume texture.

The application could be improved in several ways to both speed up production using better data structures, and to reduce memory consumption by storing lightning data in something other than a volume texture.

References

[1] *DRILIAN'S HOUSE OF GAMES*. Webpage - <http://drilian.com/2009/02/25/lightning-bolts/>

[2] KIM THEODORE, LIN MING C. 2004. Physically Based Animation and Rendering of Lightning. *University of North Carolina at Chapel Hill*.

[3] KIM THEODORE AND LIN MING C. 2007. Fast Animation of Lightning Using an Adaptive Mesh. *University of North Carolina*.

[4] NIEMEYER L., PIETRONERO, AND WIESMANN H. J. 1983. Fractal Dimension of Dialectric Breakdown. *Brown Boveri Research Center*.

[5] PERLIN KEN. 1985. An Image Synthesizer. *ACM SIGGRAPH Computer Graphics*.

[6] TSCHUMPERLE DAVID. 2004. The Cimg Library. Webpage - <http://cimg.sourceforge.net>

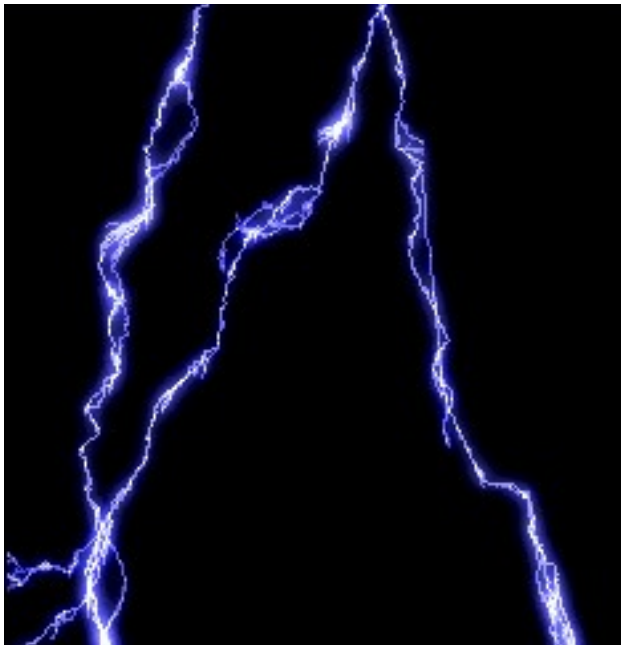

Figure 7: Initial test of 2D L-System generator

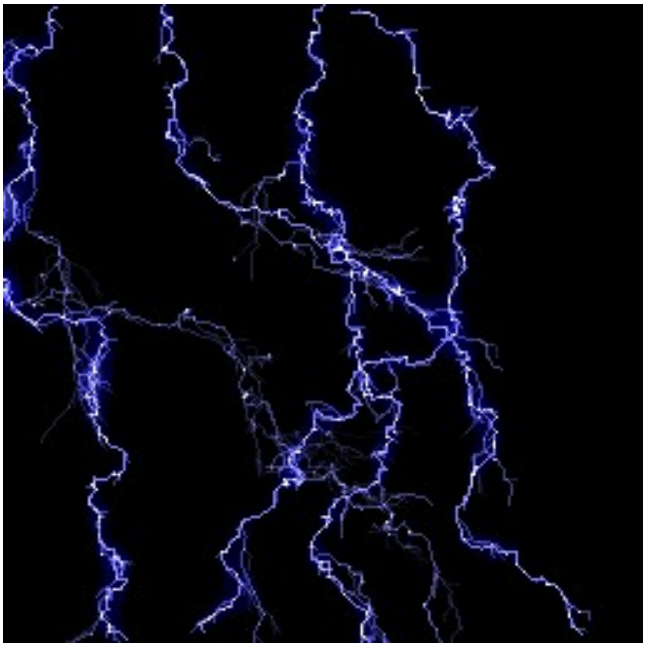

Figure 8: Multiple beams of lightning in a single volume.

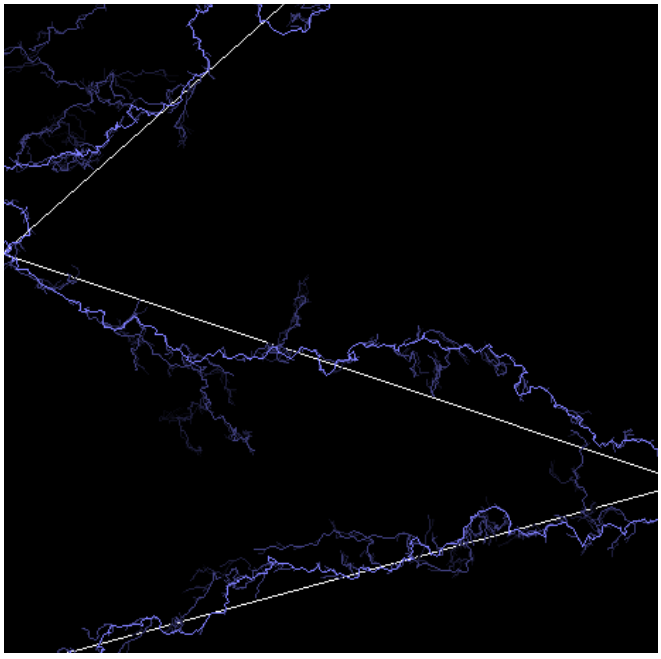

Figure 10: The L-System has the ability to follow a path (This image is simply a preview, and not a rendering).

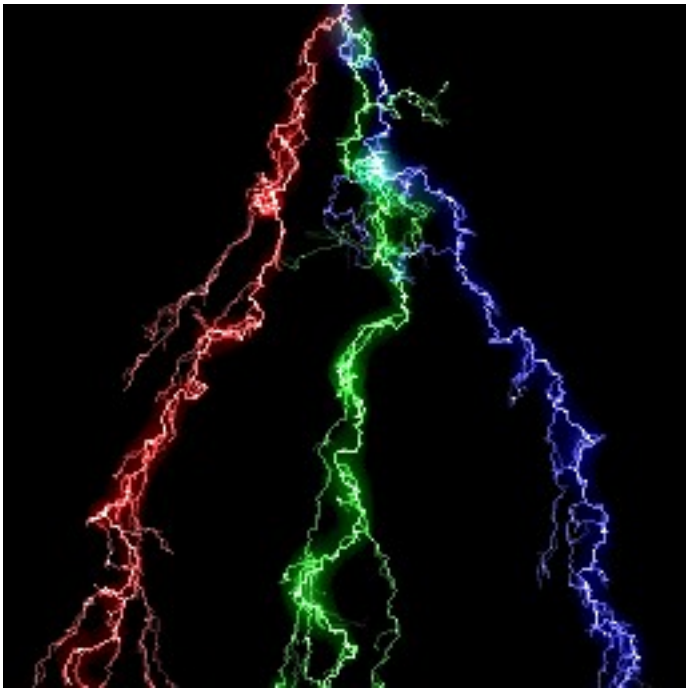

Figure 9: Lightning color can easily be changed to represent a variety of effects

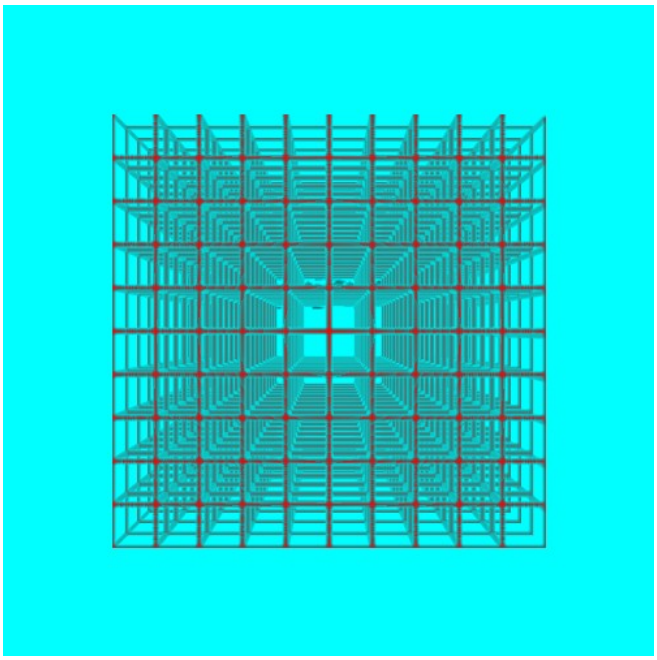

Figure 12: Volume rendering of a grid

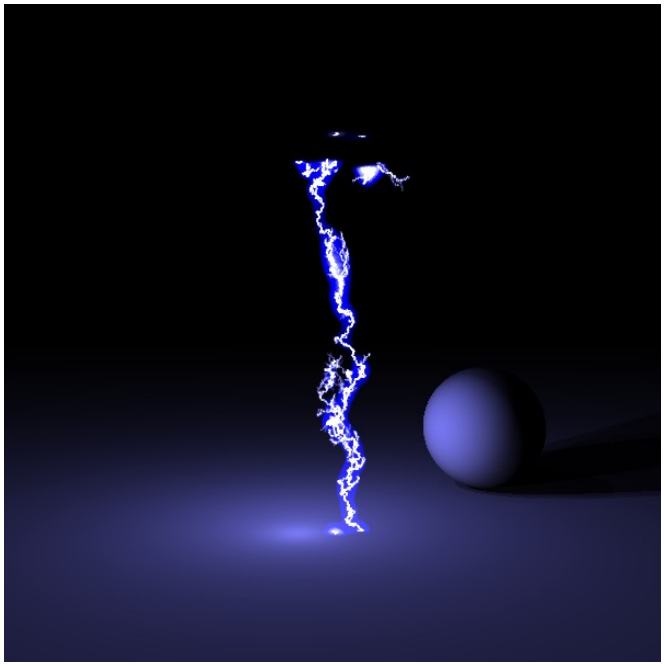

Figure 11: Rendering of a volume lightning strike in a simple scene (Lightning is upside down in this image)

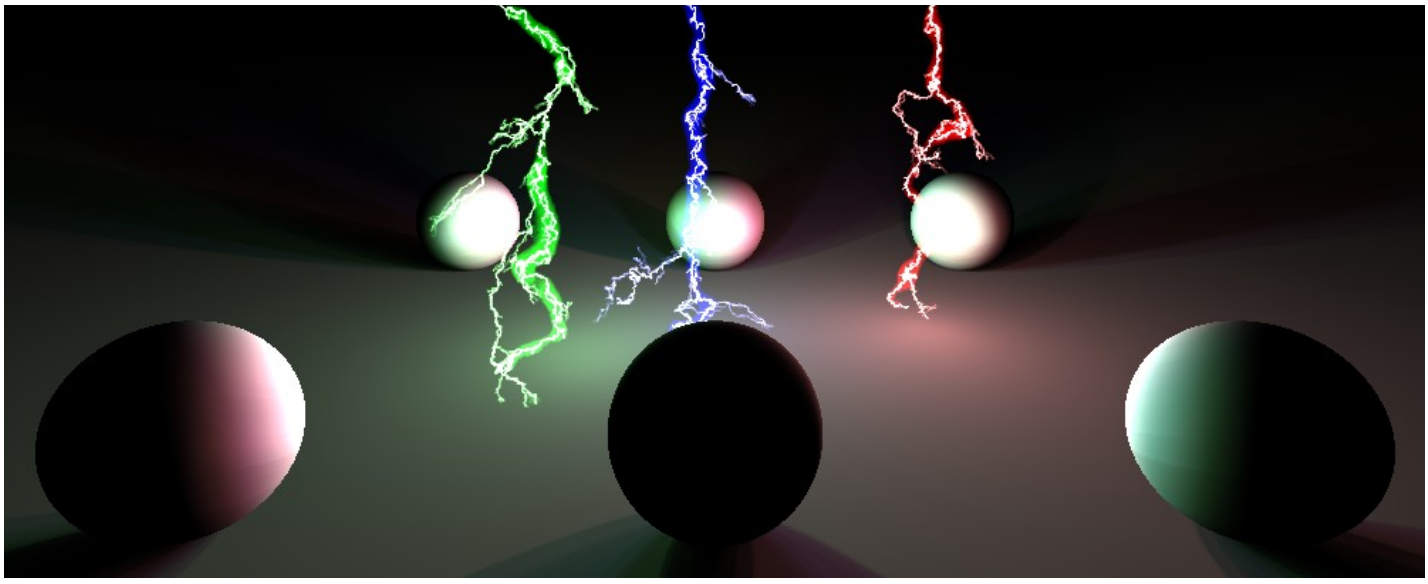

Figure 13: Rendering of scene with three different color lightning bolts