# Fast Simulation of Lightning for 3D Games

Jeremy Bryan and SK Semwal
Department of computer Science, University of Colorado, Colorado Springs
semwal@eas.uccs.edu

## Abstract

Simulating a lightning stroke supporting real time interaction involves developing a model for the main stroke, and then recursively generating similar models for any branches that may occur. A number of methods have been developed in this field, but most of the research has concentrated on rendering algorithms. This thesis generates volumetric data for a three dimensional lightning stroke through the usage of cellular automata. The goal was to develop a method where realistic lightning strokes could be generated and displayed in real-time. An algorithm, the complex lightning generation model has been designed and implemented in C++. The algorithm uses an automaton with simple rules based on random numbers and probability. Results are presented that compare our results to those created by other researchers in this area.

**Keywords**: Simulation of lightning, Cellular Automata.

## 1. Introduction

Creating realistic models of physical phenomenon has been the topic of many research papers in recent years. Generating a lightning model in three-dimensional space could affect a number of applications. Real-time lightning generation could be expanded into the computer gaming industry to enhance future games and effects. In addition, the ability to direct the lightning stroke to any given point could prove beneficial to the visual arts industry. In real-life, a lightning stoke cannot happen without the presence of a strong electrical field. This electrical field begins to accelerate free ions in the atmosphere to a very high velocity. These ions create a "stepped leader" which propagates from the cloud towards ground-zero in discrete steps. As the stepped leader approaches the ground, it attracts an "upward positive leader." As these two electrical occurrences draw closer to one another, the "attachment process" begins. The meeting of the upward and downward leaders triggers the "return stroke" which initiates the lightning flash we see and the thunder we hear. The entire flash lasts about .5 seconds and carries approximately 10,000 amps of current. Once the initial return stroke has been established, there can be as many as four or five subsequent strokes along the main channel. This is where lightning gets its flickering effect. [Reed1994]. Much of the research done in lightning graphics has concentrated on ray-tracing a three-dimensional model in order to produce a two dimensional image. The benefit to creating our three-dimensional model in real-time is being able to move the camera around interactively. Our approach uses the concept of cellular automata in order to propagate the lightning strokes through the atmosphere. Cellular automata are dynamic systems where volumes are created using cells that contain values that change based on pre-determined rules and the values of neighboring cells. Using this simple concept, complex global patterns can emerge and accurate representations are possible. We implement the cellular automata concept by creating a three dimensional lattice to represent the atmospheric space that we want the lightning to propagate through. New algorithms were developed in order to determine the path that the lightning will take. The rules that have to be written for each cell in the automata to perform this path selection are relatively simple and hence easy to update or replace. [Kaushal2004] No control points are needed to generate the stroke and branches.

## 2. Motivation

Reed et al [Reed1994] discussed a method for rendering lightning using conventional ray-tracing. "Lightning is represented as a collection of connected, finite length rays in 3D space." [Reed1994] used a simple method for creating their 3D lightning model. Starting with a seed segment in the clouds, new segments were spawned and randomly rotated about the seed segment. One particular website [UCCS] has done a fabulous job of describing a lightning strike by using a computer simulation. Unfortunately, this method is only presented for 2D representations. Dobashi et al [Dobashi2001] (Figure 2) used the identical modeling method described above.and include scattering effects due to clouds and atmospheric particles. The addition of these elements significantly enhances the overall scene (Figure 2).

Figure 1 and 2

## 3. Cellular Automata

Neuman as formal models of self-reproducing organisms [Sarkar2000]. Recently [Kaushal] have used cellular automata in an algorithm for three dimensional morphing. By looking at the atmosphere as a collection of voxels and allowing the lightning leaders to propagate by inspecting the local neighborhood, we could avoid the use of global controls. The cellular automaton which was used in our design has the following characteristics: (a) It is a three-dimensional lattice. (b) The dimension of the lattice is that of the volume. (c) Each cell can either be empty or contain a segment of lightning stroke. (d) The position of the cell in the lattice has no effect on its behavior. Hence, all cells are equal. The cellular automaton is non-circular.

We wanted to control the path of the lightning, or at least the point where the lightning strikes the ground. This would be an attractive feature in 3D games simulation of lightning. For example, a game may require that the car be hit, so the lightning needs to strike where the car is. We give users a targeting cursor which they are free to move about the environment using the arrow keys on the keyboard. The location of this cursor determines the starting point for our lightning structure. The cell that contains this point is added to a linked list data structure so that we may reference the data later. At this point, the algorithm must decide which cell to propagate the lightning into next. As discussed in the introduction, every cell in the cellular automaton has an associated r value that represents miscellaneous atmospheric conditions. Using this value, the electric field for each neighboring cell can be calculated using Maxwell's Equations. Upon finding the neighbor with the largest electrical field, we add that cell's coordinates into the linked list.

In real life, lightning progresses through the atmosphere in discrete steps. To model this behavior, the algorithm creates a random number to represent the length of a given segment. The segment continues to travel in its current direction through the given number of cells. The electrical field calculation does not have to take place until the current segment is terminated, allowing the lightning to have a wide range of deviations from it starting point. Otherwise, it would just appear to be a straight, wiggly line.

This process continues until a lightning segment reaches the ceiling, or the maximum altitude for our simulation. At this point, we have a linked list that contains objects we named CVoxels. A CVoxel represents a voxel in the cellular automata. It contains x, y, and z coordinates, an r value, and a null header for another linked list. The additional linked list is needed because any given cell can be the parent or root for a branch that is to be generated. This data represents our main stroke to be displayed.

Once the main stroke calculations have been completed, we need to generate branches. To accomplish this, we iterate through the main stroke data and use simple probability to decide whether or not the current cell we are looking at is going to be the parent of a new branch. In reality, the likelihood of a branch being spawned increases as the distance to the ground decreases. In other terms, the probability of a branch being generated is inversely proportional to the distance we currently are from the ground. This behavior was implemented and the results were not good. There were not enough branches at higher elevations, and the multitude of branches near the ground resembled the root network of a tree. Therefore, we reverted back to using a uniformly distributed probability function so that the chances of branching were equal at any point on the main stroke.

***PSEUDOCODE***

```
Proc GenerateComplexLightning

// Load the starting coordinates
list.add target.coords

while height <= MAX_ALTITUDE
Assign r to 26 neighbor cells if  they do not have one
Calculate E for 26 neighbor cells if not done previously
Find maximum E
list.add maxE.coords
end while

for each voxel in list
      temp = rand()
      if temp < BRANCH_PROBABILITY then
                              GenerateComplexBranch()
      end if end for
end proc
```

## 4. Complex Branch Generation

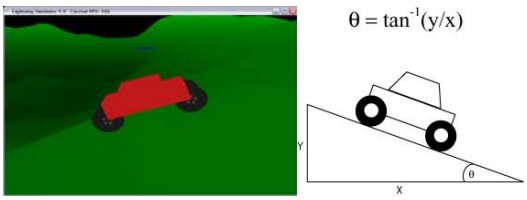
Branches can occur off the main stroke as well as off other branches. We implemented some arbitrary maximum branch depth in the interest of keep the algorithm running fast. In order to generate branches, we must traverse the data structure that contains the data for the main stroke. During that traversal, we use simple random numbers and probability to determine if a stroke segment needs to spawn a branch. If it is determined that a branch needs to be created, then the appropriate function is called and branch generation begins. The actual creation of the branch is done identically to that of the main stroke. However, there is a major addition to the branch method that needs to be noted. Branches do not necessarily need to reach ground-zero. They can terminate in the atmosphere. To achieve this feature, a random number is created at the beginning to represent this branch's maximum length. Branch creation is terminated if it reaches its maximum length or when it encounters ground-zero.

***PSEUDOCODE***

```
Proc GenerateComplexBranch

BranchLength = rand()

while (BranchHeight >= 0 And BranchLength >=0)

Assign r to neighbor cells if they do not have one
Calculate E for cells if not done previously
Find maximum E
BranchList.add maxE.coords
end while

for each coord in BranchList.coords

   if (rand() >= BRANCH_PROBABILITY) then

         GenerateComplexBranch()

   end if

end for
```

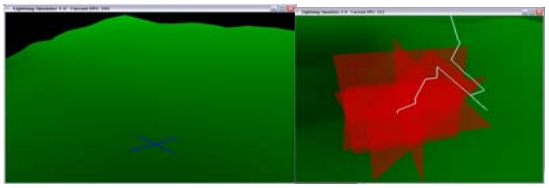
## 5. The Lightning

The most dynamic part of the program is the ability to create simulated lightning strikes at will. Initially, there was some concern about the time lag between pressing the space bar and displaying the stroke. This proved to be an unnecessary concern. For any game (Figures 4-6) to be interesting, you have to have some objective for the user to accomplish. Our simple objective is to show explosion.




Figures 3 and 4

To render the car the program made use of OpenGL's display list capabilities. The car's body, cab, wheels, and bolts are all rendered at the axis origin. The body and cab are just two rectangular boxes with the smaller one, representing the cab, laid atop the other. Four triangle fans are drawn and translated accordingly to appear as wheels. Five bolts are drawn on each wheel by creating a circle which only has five segments so that it looks like a hexagon.In order for the car to look like it was actually driving along the terrain, it is important for it to rotate realistically. It must rotate about the y-axis to ensure the nose of the car is pointed in the direction of travel (Figure 4). Once the car has been drawn and undergone rotational transforms, it needs to be translated to its current position on the terrain. The car just follows a simple circular path so, based on its velocity, the next location on the circle is calculated and then the car is translated. In order for the user to be able to determine where they want the lightning to strike, they must have some mechanism available to them to designate the desired strike point. This is accomplished through the use of a crosshairs target that can be moved about the terrain with the use of the keyboard arrows.



Figures 5 and 6

Maintaining the position of the crosshairs is simple. The program keeps track of the current x and z coordinates and updates them accordingly upon user input. The target varies in the y-direction based on the height of the terrain it is currently above. This allows the target to "hug" the terrain as it moves about. Success must be rewarded. When the user successfully targets the car and a collision between the car and lightning is detected, some sort of stimulus must be provided to inform the gamer of that status. We spent a considerable amount of time experimenting with different explosion effects. Particle engines, light sources, and bitmaps were all looked at and discarded for various reasons. In the interest of simplicity, the explosion is modeled with expanding rectangular planes that change color (Figure 6). Three rectangles, parallel to the coordinate axes, are used to create the effect.

## 5. Results

From the standpoint of attempting to create an entertaining game environment, we believe our results turned out very favorably. When the actual lightning stroke is evaluated, we encounter mixed results. The main stroke tends to look good, although not extremely dynamic from one strike to the next. On the other hand, the branches seem to be a little less believable. The algorithms were run on a standard desktop PC with a single Pentium IV processor running at 3.39 GHz and 1.0 GB of RAM. We created a simple survey that contained four black and white pictures of different lightning strokes created by different researchers (Figure 7). The first choice was a 3D model created by Andrew Glassner. [Glassner1999] The second image was from our work. The third picture was from the Reed publication. [Reed1994] Finally, the fourth choice was taken from Dobashi's results. [Dobashi2001] The survey instructed the participants to rank the pictures from 1 to 4 based on the realism of the shape, with 1 being the best and 4 being the worst. The following table illustrates the survey results:

| | Glassner | BR_S | Reed | Dobashi |
|---|---|---|---|---|
| | | | | |
| Mean | 3.171875 | 2.25 | 2.28125 | 2.296875 |
| Median | 4 | 2 | 2.5 | 2 |
| Mode | 4 | 1 | 3 | 2 |
| StdDev | 0.96863 | 1.140871 | 1.075982 | 1.03402 |

Table 1

As you can see from the data in Table 1, the results were close. Our model had an average score of 2.25

while the Ross' scored 2.28 and Dobashi's came in at 2.29. These results are extremely similar. While this data does not suggest that our model is any better than the other two, it does suggest that ours is at least comparable. Our sample size was 64. In order to get a more accurate representation the sample size should probably be increased significantly.

## 6. Conclusions and Future Research

We have shown that real-time support of a realistic lightning strike is achievable using our algorithm. The lightning strokes can be created through the use of cellular automata and work well by creating acceptable stroke sequences during game play. There are several ways in which the lightning generation quality could be improved. For example, more efforts are needed for lightning up the sky. The relationship of branches with the main branch could also be improved as the branches don't seem to propagate away from their parent enough to emulate true lightning. We would also like to add the capability in our implementation where the uses can select certain voxels in space that they want the lightning to pass through. This feature would allow users to shape the lightning as they see fit.

We studied current research in computer lightning models and rendering and concluded that a one major problem is the need for real-time model generation as opposed to ray-traced images. We discuss a new algorithm, making use of a simple cellular automaton. The concept of the algorithm is relatively simple, but that does not preclude it from producing acceptable results. Comparing our results to those from other researchers is difficult. They had an unlimited amount of time to produce their model where we were attempting to display our results in real-time. Given this difference, we feel like our results are extremely satisfying. We have proven that the use of cellular automata is a viable approach to modeling lightning in three dimensions. The application of this research could empower 3D games and visual arts industries. The project implementation was done in C++ and several different models were compared to ours. Analysis of the results shows that our algorithm is ideal for game applications because of the support of interactivity and produces good results.

## 7. References


[Reed 1994] Todd Reed and Brian Wyvill, "Visual Simulation of Lightning" International Conference on Computer Graphics and Interactive Techniques, Proceedings of the 21st annual conference on Computer Graphics and Interactive Techniques Pages: 359 – 364, 1994

[Dobashi2001] Yoshinori Dobashi, Tsuyoshi Yamamoto, and Tomoyuki Nishita "Efficient Rendering of Lightning Taking into Account Scattering Effects due to Cloud and Atmospheric Particles" Proceedings of the 9th Pacific Conference on Computer Graphics and Applications, Pages: 390, 2001

[Droun2003] Druon S, Crosnier A, Brigandat L "Efficient Cellular Automata for 2D / 3D Free-Form Modeling" Journal of WSCG (Winter School of Computer Graphics) Volume: 11, Number 1, Pages: 102-108, 2003

[Sarkar2000] Palash Sarkar "A brief history of cellular automata" ACM Computing Surveys (CSUR) Volume 32 , Issue 1 ,Pages: 80 – 107, 2000

[UCCS] "Simulating Lightning" http://www.uccs.edu/~physics/simulations/lightning.html

[Glassner1999] Andrew Glassner "The Digital Ceraunoscope: Synthetic Thunder and Lightning" Microsoft Resarch Technical Report MSR-TR-99-17;Year of Publication: 1999.

[Kaushal2005] SK Semwal and K Chandrashekhar, Cellular Automata for 3D Morphing of Volume Data, 13th International Conference in Central Europe on Computer Graphics, Visuzliation, and Computer Vision, 2005, pp. 1-8, 20

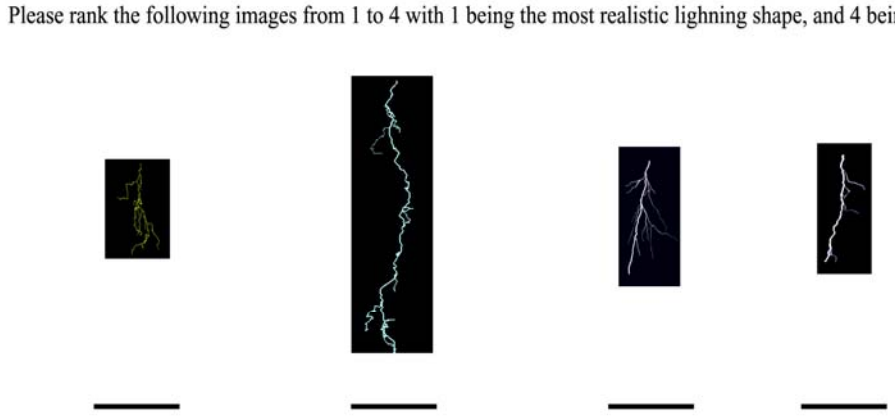




Figure 7: Four images – Glassner, BR-S, Reed, Dobashi