Physically-Inspired, Interactive Lightning Generation

Jeongsu Yun
KAIST
jeongsu.yun@gmail.com

Myungbae Son
KAIST
nedsociety@gmail.com

Byungyoon Choi
KAIST
byungyoonc@gmail.com

Theodore Kim
University of California, Santa Barbara
tedkim.cs@gmail.com

Sung-eui Yoon*
KAIST
sungeui@kaist.edu

Abstract

We present an interactive technique for generating realistic lightning. Our method captures the main characteristics of the *dielectric breakdown model*, a physical model for lightning formation. Our algorithm uses a distance-based approximation to quickly compute the electric potentials of different charge types. In particular, we use a rational function in lieu of summed potentials to better produce interesting lightning patterns. We also propose to use the waypoints commonly available in many game scenes to guide lightning shapes in complex scenes. We found that our algorithm is two times faster than the state-of-the art method with better controls on lighting shapes, and can generate realistic lightning shapes interactively.

Keywords: lightning, interactivity, rendering

Figure 1: A night scene with lightning generated by our method. It takes 38 ms on a 128 x 128 grid by using a single core and is about 20 times faster than using the conjugate gradient method from [Kim and Lin, 2004].

1 Introduction

Realistic simulation and rendering of natural phenomena such as snow, rain, and lightning can improve immersion in movies and games. In movies, lightning is often used to set the mood and emphasize fear. In games, it is frequently used to portray realistic weather and represent the effects of magic.

Unfortunately, generating realistic lightning using physically-based techniques can be very time consuming [1]. While this running time may be feasible for movie production, they are too slow for interactive applications like games. As a result, current games utilize pre-rendered images or approaches based on randomized trees. Immersion in the game can be hampered by the repetitive display of the same patterns, and the low quality of the results.

In this paper, we propose an approximate, physically-inspired method to interactively generate realistic lightning. Our algorithm aims to reproduce the main characteristics of a physically based technique, the dielectric breakdown model (DBM) [2, 1], but remove the dependence on numerical solvers such as the conjugate gradient method. We first propose to represent the potential as a rational function that

*Corresponding author.

combines different types of electric charges. We then introduce additional controls for the lightning shape by combining of two parameters, η and ρ, in our approximation method based on a rational function (Sec. 4.1). Finally, we suggest the use of waypoints to generate lightning shapes for complex scenes (Sec. 4.2).

We have compared our method against the DBM computed using conjugate gradient method [1], and observe speedups of over two orders of magnitude. We also compared our algorithm against the state-of-the-art method, the DBM approximation with a summation model of potentials [3]. Our method shows roughly twice as fast as this approach with better shape controls, and we found that ours show better approximation to the DBM approach and thus a wider applicability to different configurations.

2 Previous work

We review prior methods that are directly related to our method.

2.1 Lightning Shapes

Reed and Wyvill [4] propose an empirical method to generate lightning shapes based on the observation that lightning branches are randomly distributed at roughly 16 degrees. New segments are generated by rotating about the parent segment at an angle of about 16 degrees. In the similar approach, Glassner proposes a two-pass algorithm following the statistics of the lightning [5]. A large-scale structure of the lightning stroke is generated in the first step, and then zig-zag patterns are added as details to a long, straight stroke.

Niemeyer et al. presented a dielectric breakdown model (DBM) that physically explains dielectric breakdown phenomena (e.g. lightning and surface discharge) [2]. Thanks to its physical origins, DBM has been used widely for the lightning simulation. We explain the algorithm in detail in Sec. 3.

Sosorbaram et al. use DBM to generate lightning shapes [6], but utilize a local approximation to the electric potential field instead of solving the exact Laplace equation from the DBM. Kim and Lin [1] propose a robust method to generate the lightning shape by solving the Laplace equation with the conjugate gradient method. However, creating the lightning shape can be time-consuming because the conjugate gradient method is an iterative algorithm. To improve the computation time, methods that utilize adaptive meshes like quadtrees or octrees were proposed [7, 8]. [3] propose a fast method that simulates lightning using a distance-based approach. It is similar to [6], which uses an electric potential equation, but uses spherical coordinates.

There are some approaches to generate lightning in real-time. Matsuyama et al. use the GPU to solve the Laplace equation [9]. By limiting the conjugate gradient method to two to four iterations, real-time rendering is achieved. Nvidia provides a real-time DirectX 10 example that uses geometry shader. Unfortunately, this approach is not physics-based.

In a different direction, Xu and Mould generate similar patterns using a path-planning based approach that finds the least-cost paths on a weighted graph within a randomly weighted regular lattice [10]. In this paper, we propose a physically-inspired method that generates lightning by better approximating the characteristic value distributions of an electric potential field.

2.2 Lightning Rendering

How to render the atmospheric scattering of lightning is also a significant problem. Traditional rendering techniques that use polygon models show rather low quality for rendering lightning.

Reed and Wyvill extended ray tracing to render the lightning stroke and glow [4]. Sosorbaram et al. propose a volume rendering technique using a 3D texture [6]. Dobashi et al. use hierarchically structured metaballs and precomputed lookup tables that store the integrated intensity of light scattering [11]. Kim and Lin [1] and Bickel et al. [7] utilize an atmospheric point spread function, which describes the scattering of light in participating media [12], as a convolution kernel to render the final lightning.

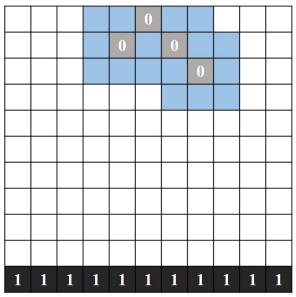


Figure 2: The grid used for DBM. Black and grey cells respectively represent the positive charges on the ground and the negative charges at the bottom of cloud. Blue cells are the next candidates that could be added to the lightning.

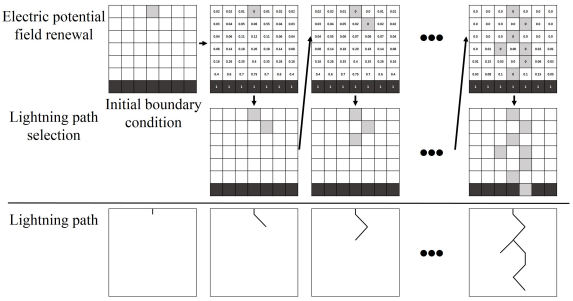


Figure 3: Generating a lightning shape using DBM.

3 Background

We first introduce the basic physics of the lightning and explain the dielectric breakdown model [2] for lightning simulation. We then present the details of our method for quickly animating and rendering lightning.

3.1 The Physics of Lightning

Lightning occurs when a large charge difference exists between two areas such as a cloud and the ground. 90% of all cloud-to-ground lightning is *downward negative lightning* that occurs when negative charges from a cloud, and spreads out towards positive charges on the ground. Negative charges move through the lightning stream from the air to the earth until an equilibrium state is reached.

Experimental observations have shown that lightning branches maintain an angle of about 16 degrees with their parent branch, and that the lightning has a fractal dimension of approximately 1.7 [2]

3.2 Dielectric Breakdown Model (DBM)

DBM [2] uses a regular grid representation and calculates an electric potential, ϕ, for each grid cell. Figure 2 shows a grid representation for the lightning simulation. Negative charges are placed at the top and their electric potentials are set to $\phi = 0$. Positive charges are placed at the bottom and are set to $\phi = 1$. The boundaries of the grid are also set to $\phi = 0$. These three type of electric potentials are fixed and treated as boundary conditions. For the remaining grid cells, the electric potentials are calculated by solving the Laplace equation:

$$\nabla^2 \phi = 0. \tag{1}$$

After computing ϕ over the grid, the lightning is grown by randomly selecting a "candidate" cell neighboring the existing lightning. The probability of selection is weighted according to the electric potential. In Figure 2, blue cells are the candidate cells for the current lightning. The probability of selection for each candidate cell i is computed using the following normalization equation:

$$P_i = \frac{(\phi_i)^\eta}{\sum_{j=1}^{n} (\phi_j)^\eta}, \tag{2}$$

where *j* is an index of each candidate cell and *n* is the total number of the candidate cells.

The electric potential for the chosen cell is then set to $\phi = 0$. The chosen cell becomes part of the lightning and is added into the boundary condition. This process is repeated until the lightning reaches the cells that have the positive charge on the ground. Figure 3 shows the overall process of generating lightning using DBM. η can control the number of branches. As η increases, the lightning shape has fewer subbranches. Therefore, DBM is often called the η model. A value of two or three is used for a realistic lightning.

Kim and Lin [1, 8] proposed practical techniques for using DBM to generate lightning. These techniques are based on DBM and solve the Laplace equation by using the conjugate gradient method with an incomplete Cholesky preconditioner. While this technique can generate realistic shapes, it can be computationally expensive, because the conjugate gradient method

0.002	0.003	0.001	0	0.001	0.003	0.002
0.006	0.008	0.002	0	0.002	0.008	0.006
0.015	0.021	0	0.027	0	0.021	0.015
0.032	0.062	0.086	0.106	0.086	0.062	0.032
0.051	0.109	0.176	0.226	0.176	0.109	0.051
0.062	0.148	0.281	0.447	0.281	0.148	0.062
0.05	0.138	0.355	1	0.355	0.138	0.05

Figure 4: This shows values of the potential field after solving the Laplace equation using the conjugate gradient method on a 7 x 7 grid.

has a time complexity of $O(G^{1.5})$ [13], where G is the number of the grid cells. We found that it takes about 1.5 seconds to create a lightning shape using a 128 x 128 grid.

4 Our Method

In this section, we explain various components of our method for interactive lightning generation.

4.1 Rational Approximation of Electric Potentials

We propose a method that quickly approximates the dominant properties of electric potentials, and allows them to be computed at interactive rates. We do not solve the Laplace equation using the conjugate gradient method. We found that there are two simple properties of a potential field that can be described according to its cell type:

- The potential increases towards the positive cells.
- The potential decreases towards the negative cells on the lightning path and boundary cells.

Figure 4 shows an example of the electric potential on a 7 x 7 grid. The potential depends on the distance to each type of cells, similar to other physical equations for electrostatics such as Coulomb's law. We are not the first one to observe such phenomenon. In physics, it is well-known that a potential at a point x that satisfies the Laplace equation is equal to the average potential computed on a virtual sphere located at the point x, each of which is governed by the electric potential equation [14]:

$$V = \frac{1}{4\pi\epsilon_0}\left(\frac{q}{r}\right), \tag{3}$$

where q and r are an electric point charge and the distance from the charge to the point x, respectively. This idea is also adopted in prior works [6, 3].

We have found that these prior techniques can generate less interesting branching patterns in the lightning shape depending on configurations. According to our tests, we conjecture that this phenomenon can occur because the potential can have very similar values between the candidate cells and other charges along the boundary condition. As a result, those candidate cells are likely to have similar probabilities of being chosen. Therefore, the computed final lightning shape tends to spread out to all directions from the starting position.

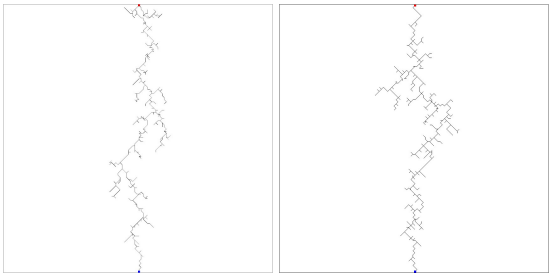
Instead, we propose to approximate the $1/r$ relation of the electric potential that is controlled by ρ, as follows:

$$V_{approx} = \sum_{i=1}^{n}\left(\frac{1}{r_i}\right)^{\rho}, \tag{4}$$

where $\rho > 1$ and i indicates an index of other charged cells; details of considering other charged cells are explained later. Once we have computed the electric potential between the candidate cells and other charged cells, we use the normalization equation, Eqn. 2.

Before we compute the potentials, we divide the charges into three types: positive charges, negative charges along the lightning path, and boundary charges. We then calculate the electric potentials based on those types separately as *P*, *N*, and *B*,

Most prior approaches sum these three terms (*P*, *N*, *B*), to compute the final potential. However, we found that a simple sum does not create interesting branching patterns. For example, consider the 'tip effect', which indicates that a region surrounded by negative charges has a high probability of having negative charge [2]. The summation of the terms fails to produce this effect. The issue arises mainly because we compute potentials by assuming each charged point is in the center of a cell. As a result, the $1/r$



(a) CGM (b) Our method

Figure 5: Comparison of lightning shapes. Red and blue rectangles represent the start and goal position of the lightning. Our method shows a similar result to DBM, but is about 20 times faster.

term cannot create extremely small values between even neighboring cells.

To address this issue, and to better express the properties of a potential field, we propose to use the following rational function:

$$\phi = \frac{P}{N \times B}. \tag{5}$$

Since we divide positive potentials with those of negative ones, we can generate stronger negative potentials among nearby negative charges. In other words, this rational function is adopted to re-produce the two key properties of the potential field.

Figure 5 compares lightning that was generated by the conjugate gradient method [1] to our method on a 128 x 128 grid. Figure 6 shows a potential starting from the top and increasing towards the bottom of a 128 x 128 grid. The summation model that sums *P*, *N*, and *B* does not match well to the result of DBM, while our rational model shows a similar value distribution to the reference.

Arguably, our parameter ρ has a similar effect as the existing parameter η used in the normalization equation. However, we have found that it has a subtle yet meaningful effect compared to that of η. To show the different behaviors of η and ρ, Figure 7 shows lightning with different values. As the value of ρ increases, the lightning has fewer branches and shows stronger directionality to the target position. The η parameter of DBM also controls the number of branches and shows a similar effect to ρ. Unlike ρ, increasing η trims small branches, while maintaining the main stream. As a result, applying a proper ρ value before computing the probability with η can make more appealing branching patterns than those results only from the η term.

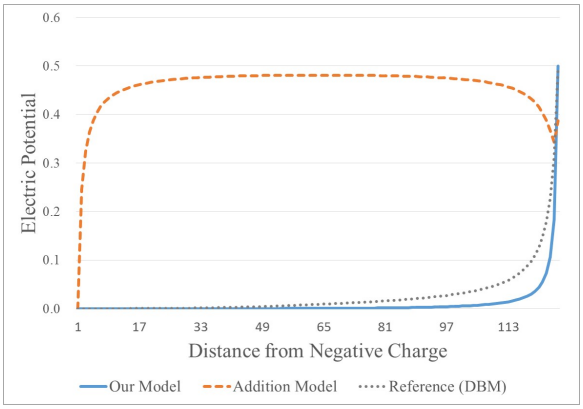


Figure 6: Distribution of the potential on the 128^2 grid scene shown in Fig. 5. The x and y axes represent distance from the initial negative charge and the computed electric potential, respectively. The summation model does not show a proper value distribution, while our rational model shows a similar pattern to the one computed by DBM.

We also provide different types of the lightning that are not the cloud-to-ground lightning. Figure 8 shows the lightning for multiple targets. Figure 9 shows a chain lightning, where the lightning goes through the targets in a row.

4.2 Avoiding Local Minima

There can be objects or obstacles that should not be hit by the lightning, such as wall in a game scene. For such obstacles, we assign negative charges and treat them as a part of the boundary condition.

When we have a complex scene, our method encounters local minima, similar to other potential field methods [15]. Since our algorithm computes potentials based on the distance, when a scene has obstacles that block the target position from the starting position, the lightning branches can spread out excessively (Figure 10).

To handle this problem, we utilize waypoints that are generated by path planning methods such as the A* algorithm. Many games already use fast path planning algorithms to compute such waypoints for various purposes (e.g., computing navigable paths [16]). Waypoints consist

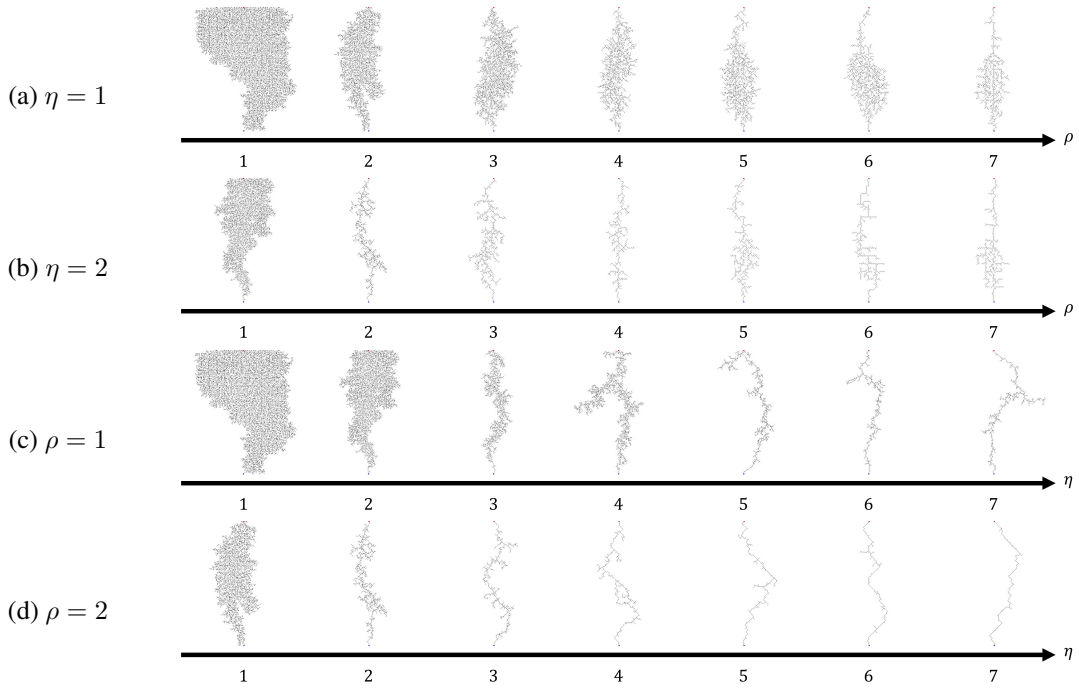


Figure 7: Lightning shapes as a function of η and ρ. By increasing ρ values, we can trim down branches with a stronger directionality to the target position. On the other hand, as we use bigger η values, we trim down small branches, while maintaining the main branches; see the pdf file for zoomed-in views.

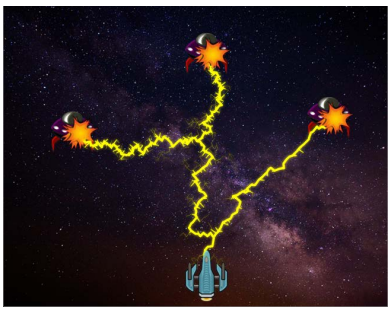


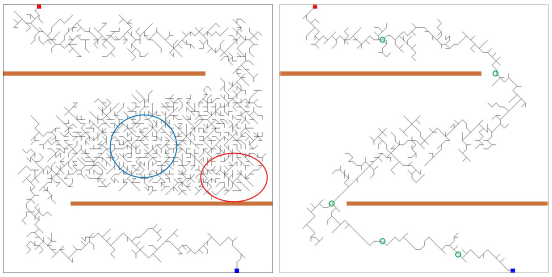
Figure 8: An example of a space scene with a lightning weapon for multiple targets generated by our method.

Figure 9: An example of the chain lightning. A sorcerer casts a magic of chain lightning that goes through the enemies.

of a sequence of points that define a path. To guide a lightning shape in a complex scene, we access the first waypoint cell, W, in the waypoint list and use it as a positive charged cell instead of considering the target positive charges. Once the lightning pattern reaches the cell W, we iterate the process by accessing the next waypoint, and continue until we reach the final waypoint in the list, which is set to the target positive cell. Figure 10 shows the lightning path guided by waypoints.

4.3 Rendering

We utilize physical characteristics such as the thickness and brightness of the lightning stream to render the lightning [4, 1]. The lightning stream is commonly decomposed into the main and secondary channels. The main channel is a path that connects the starting and target points, while the secondary channels are sub-branches of the main channel. The main channel receives a bright and constant intensity, while the secondary channel receives a reduced intensity in proportion to the distance from the main channel. Also, the secondary channel has half of



(a) Result w/o waypoints (b) Result w/ waypoints

Figure 10: Lightning shapes computed with and without waypoints. The brown objects represent obstacles. (a) shows the local minima problem. Cells in the red circle are closer to the goal than cells in the blue circle. The lightning tries to grow in the red circle, even though it cannot reach the goal directly. (b) shows the result with waypoints, represented by green circles.

thickness of the main channel.

Our rendering algorithm uses deferred rendering and is implemented as in the OpenGL Shading Language (GLSL). We render a scene and its lightning to separate framebuffer textures. For rendering the lightning shape, we apply the thickness and brightness to each branch while considering the depth buffer. Lines of the lightning are rendered by using billboard techniques as rectangles. We use cylinders and spheres to represent branches of the lightning for three dimensional scenes (Figure 12).

To represent the glow effect of the lightning, we use a fast two-pass Gaussian blur filter, which is a widely used technique in games [17]. First, we apply the Gaussian blur horizontally to the framebuffer texture of lightning and, then, use the filter vertically on the previous result. At the last stage, we combine two textures to get the final image. We also utilize jittering [8] to reduce an artifact of grid regularities that appear due to the lack of a fine grid resolution.

5 Implementation and Results

We implemented our algorithm in C++ using data structures in STL. Figure 1 shows a night scene with the lightning that is generated by our method. All the experiments are run by using a single core on a 2.6 GHz Core i7 PC. Details on implementation and acceleration techniques are available at the supp. report.

Table 1: 2D timing comparisons: The table shows the average time (ms) to generate lightning. The number in parenthesis is the average number of lightning branches.

Grid size (2D)	DBM (Kim and Lin)	Our method
32 x 32	26 (52)	2 (56)
64 x 64	199 (136)	13 (175)
128 x 128	1429 (395)	60 (371)
256 x 256	18555 (1262)	713 (1347)

Table 2: 3D timing comparisons: The table shows an average time (ms) to generate lightning. The number enclosed in parenthesis is the average number of lightning branches.

Grid size (3D)	DBM (η=3)	Our method (η=2, ρ=3)
32 x 32 x 32	3720 (76)	27 (75)
64 x 64 x 64	104918 (280)	204 (313)

5.1 Comparisons

We ran experiments to compare the performance of our method with DBM [1] and its approximate method, the fast Laplacian growth (FLG) method [3].

First, we compared the computation time of our method against DBM on a simple scene that has a negative charge at the top and a positive charge at the bottom. We used $\eta = 2$ and $\rho = 3$ in most cases, except for the 128 x 128 and 256 x 256 cases. We used $\eta = 3$ and $\rho = 3$ for those cases to produce similar branching to the result of DBM [1]. We performed ten trials of both algorithms and reported average values. Tables 1 and 2 show the average time to generate the lightning shape and the number of branches with different grid sizes for both two-and three-dimensional scenes. Our method is about 20 times faster for 2D scenes and 320 times faster for 3D scenes.

Overall, DBM is not suitable for interactive applications, while our algorithm can provide interactive frame rates at 64^2 for 2D and 32^3 for 3D scenes. Furthermore, when we can allow multiple frames, e.g., two or three frames, to asynchronously generate a lightning shape, our

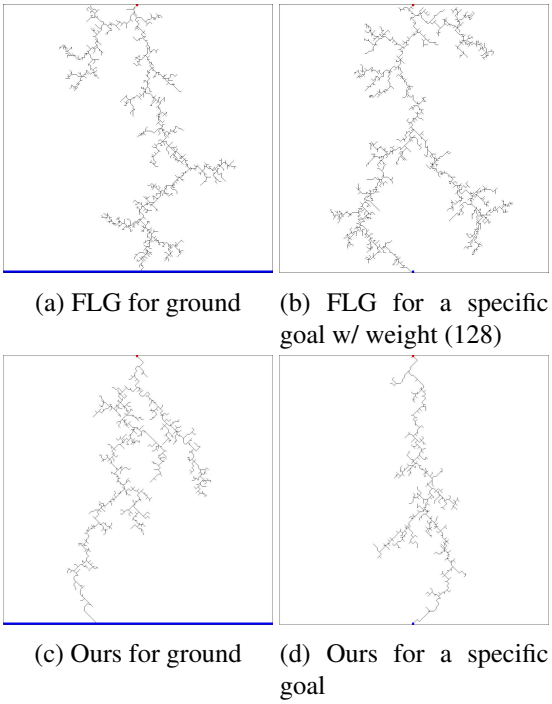
Table 3: Time (ms) comparison per branches on a 512 x 512 grid with the same $\eta = 2$ value used.

branches	DBM	Fast Laplacian growth	Our method
100	87157	10	8
200	91705	22	12
300	96864	38	19
400	102639	55	26
500	109009	72	35
1000	149833	181	94
2000	255792	496	288
3000	343199	921	565
4000	409383	1432	902
5000	458477	2022	1319

method becomes practical with 128^2 and 64^3 grids.

We also compared the computation times of different methods as a function of the number of branches for DBM, FLG [3], and our method. We used a 512 x 512 grid and same $\eta = 2$. Table 3 shows a computation time for each number of branches. In this case, our method also shows significantly faster performance than DBM. In addition, our method is slightly faster, about two times, than the fast Laplacian growth. While our performance over FLG is minor, we found that our method can generate a wide variety of lightning shapes (Fig. 8) thanks to our rational function and two different parameters (Fig. 7). In addition, we have shown that our method can handle more complex scenes by utilizing waypoints (Fig. 10).

We have additionally tested how our method and FLG behave in different scene configurations, especially, scenes with the ground positive changes or with a specific target. We found that while FLG supports the ground charges easily, it requires a high weight for the positive charge for a specific single target scene to attract the lightning to the particular destination. This is mainly because the electric potential for the positive charge is too small than other negative charges without using such a high weight. On the other hand, our method can generate the lightning shape that proceeds to the target for both ground and specific single target scenes without adjusting any weights for positive charges. This property is achieved by our rational approximation



(a) FLG for ground (b) FLG for a specific goal w/ weight (128)

(c) Ours for ground (d) Ours for a specific goal

Figure 11: Behavior difference with fast Laplacian growth (FLG). Red and blue rectangles represent the start and goal position respectively. FLG requires a high weight to attract the lightning to a specific goal, while our method can generate the lightning shape for that case, without adjusting any weights.

that effectively attracts the lightning to the target positive charges. Figure 11 shows the lightning shapes for ground and single target scenes.

6 Conclusion

We have presented a physically-inspired, interactive algorithm for generating realistic lightning. Our algorithm shows visually similar results to previous physically based methods at interactive speeds. Our algorithm can be used in games that require realistic lightning or magic effects.

While our algorithm can generate the lightning quickly, higher resolution of the grids consistently results in more realistic lightning. An obvious direction of accelerating our method to support these resolutions is to implement our algorithm on the GPU. Lightning emits light and generates a sound at the time of discharge. For representing a realistic lightning, the lightning should be considered as a light source in the

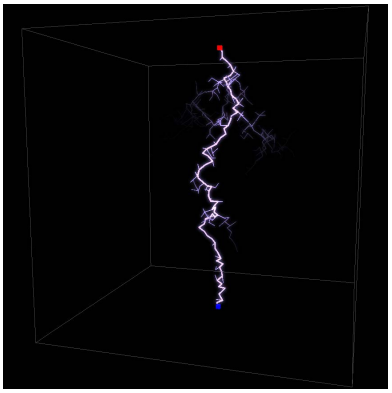


Figure 12: The lightning in a 3D scene with a 64 x 64 x 64 grid. The lightning stream is rendered by cylinders and spheres that are generated by geometry shader.

scene. Furthermore, sounds generated by considering the distance between the user and the lightning could have a substantial impact on user immersion in games.

Acknowledgements

This work was supported in part by MSIP/IITP [R0126-17-1108] and MSIP/NRF (No. 2013-067321).

References

[1] Theodore Kim and Ming C. Lin. Physically based animation and rendering of lightning. In *Pacific Graphics*, pages 267–275, 2004.

[2] L. Niemeyer, L. Pietronero, and H. J. Wiesmann. Fractal dimension of dielectric breakdown. *Physical Review Letters*, 52(12):1033–1036, 1984.

[3] Theodore Kim, Jason Sewall, Avneesh Sud, and Ming C Lin. Fast simulation of laplacian growth. *IEEE computer graphics and applications*, 27(2):68–76, 2007.

[4] Todd Reed and Brian Wyvill. Visual simulation of lightning. In *SIGGRAPH*, pages 359–364. ACM, 1994.

[5] Andrew Glassner. The digital ceraunoscope: Synthetic thunder and lightning, part 1. *IEEE CG & A*, 20(2):89–93, 2000.

[6] B. Sosorbaram, T. Fujimoto, K. Muraoka, and N. Chiba. Visual simulation of lightning taking into account cloud growth. In *Computer Graphics International*, pages 89–95. IEEE, 2001.

[7] B. Bickel, M. Wicke, and M. Gross. Adaptive simulation of electrical discharges. In *Vision, Modeling, and Visualization*, 2006.

[8] Theodore Kim and Ming C. Lin. Fast animation of lightning using an adaptive mesh. *IEEE TVCG*, 13(2):390–402, 2007.

[9] K. Matsuyama, T. Fujimoto, and N. Chiba. Real-time animation of spark discharge. *The Visual Computer*, 2006.

[10] Ling Xu and David Mould. Constructive path planning for natural phenomena modeling. In *Artificial Intelligence Tech. for Computer Graphics*, pages 83–102. 2009.

[11] Y. Dobashi, T. Tamamoto, and T. Nishita. Efficient rendering of lightning taking into account scattering effects due to clouds and atmospheric particles. In *Pacific Graphics*, pages 390–399, 2001.

[12] S. Narasimhan and S. Nayar. Shedding light on the weather. In *CVPR*. IEEE, 2003.

[13] J. R. Shewchuk. An introduction to the conjugate gradient method without the agonizing pain, 1994.

[14] David Jeffrey Griffiths and Reed College. *Introduction to electrodynamics*, volume 3. prentice Hall Upper Saddle River, 1999.

[15] Oussama Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *The international journal of robotics research*, 5(1):90–98, 1986.

[16] Xiao Cui and Hao Shi. A*-based pathfinding in modern computer games. *International Journal of Computer Science and Network Security*, 11(1):125–130, 2011.

[17] J. Mitchell, M. Ansari, and E. Hart. Advanced image processing with directx® 9 pixel shaders. *ShaderX2*, 2004.