Simulating Lightning: Adaptations of the Dielectric Breakdown Model and Diffusion Limited Aggregation

Sean Sullivan, Nick Pagel, Taras Holovko
AM 205 Final Project

December 13, 2018

1 Introduction

Numerous natural phenomena are occurrences of fractal growth. From the growth of dendritic cells to the distribution of blood vessels to lightning, fractal structures can be seen across various disciplines. Despite its prevalence, fractal growth is inherently difficult to simulate—current simulations suffer from both model and computational constraints.

One example of this is that of lightning. Even with understanding of the drivers behind lightning formation, many models suffer from inaccuracy and those do not are too slow or computationally expensive to be used liberally. As with many simulations, there has been a lack of domain expertise imputed in computer-generated models. In the case of lightning, the underlying physics behind its occurrence is well known, but this knowledge has not necessarily translated itself into code.

There are a vast number of methods that that make simulation of lightning more efficient. In this paper we examine three methods for this simulation, two based on the Dielectric Breakdown Model (DBM) and one based on Diffusion Limited Aggregation (DLA). Each method possesses its own merits, with DLA being faster but not physically based, and DBM being physically based but more computationally expensive. Several techniques to speed up solving the discretized Poisson equation that arises in DBM include the conjugate gradient method, regular and multi-level spectral deferred correction, and fast Poisson solvers, preconditioners for iterative solvers (incomplete Cholesky, incomplete Poisson, etc.), and various adaptive mesh refinement (AMR) techniques (octree and quadtree AMR, Berger-Oliger AMR, etc.).

Of course, the extent to which our paper incorporates these methods is limited by its scope, and we are therefore forced to be selective in the methods that we take on. Therefore, to optimize the simulation of lightning—the core problem presented in our paper—we focus on making adjustments to DBM and DLA with a select few of the common optimization techniques listed above and our own modifications.

1.1 Physical Basis of Lightning Simulation

There are two criteria for simulating lightning that we consider when examining previous work. We consider the physical basis of the methods, that is, how realistically they model the physics of the world, and we consider the qualitative assessment of their accuracy of lightning generation.

Lightning, a form of electric discharge, occurs with significant charge difference between objects. In the case of our simulations, this is the difference in charge between electrons with a large negative charge in a cloud and the ground, with a relatively positive charge. Simulating this phenomenon is the basis of our paper, though all of our methods are extensible to arbitrary initial charge configurations. Lightning starts in a cloud with a column of negative charge called an *initial breakdown*. From this initial column of charged air, another breakdown then occurs adjacent to the column, creating an additional negatively charged point. This process continues, a single stroke to the ground, called the *stepped leader* is formed as electrons jump from the bolt of lightning to adjacent points, preferring those with the largest charge difference. This bolt is formed nearly instantaneously lasting about 35 ms [1]. Each strike of lightning that occurs, however, is made up of several successive bolts, with this being the first. The stepped leader deposits negative charges along the way, which makes way for subsequent *dart leaders* which are influenced by the deposited charges to follow a similar path to the first bolt. Each of these leaders, because of this bias, occur much more quickly, taking about 1-2 ms, and this principle holds in the simulation as well: simulating subsequent leaders is much faster.

Another way to consider the physical basis of lightning is looking at the growth as a fractal, considering the fractal dimension and generating similar dimension fractals. Lightning, as well as many other natural phenomena, tends to have a fractal dimension of 1.7 as first investigated in [2]. These phenomenon are known as Laplacian growth, and are based physically in a Poisson PDE. In simulating DLA, then, which possesses less of the physical basis outlined above, we can aim to create simulations with a similar geometric properties through fractal dimension.

1.2 Background on Various Growth Models

Before possible improvement techniques can be presented and their validity assessed, it is critical to understand previous work in the area. Research has been conducted exploring the physical properties of lightning propagation, and its interaction with electric fields. Our paper will focus on the incorporation of this research into simulations.

[3] discusses three different kinds of fractal growth algorithms: DBM, DLA, and Hastings-Levitov iterative conformal mapping (HLCM). HLCM is the most mathematically involved of the three algorithms and does not clearly derive from physical principles and as such this paper is focused on DBM and DLA.

The physically based motivation of DBM is outlined in [2], and involves a two step process. First a potential field is solved over a grid with a discretiza-

tion of the Poisson equation. Then, the potential difference of points adjacent to lightning is calculated and a point is randomly selected with probability proportional to this potential difference. This is the point to which the lightning "jumps" in this time step. This adds to the boundary condition and the Poisson equation is solved again and the process repeated until the lightning has grown to the desired size. In [3], an alternative formulation for DBM is proposed that avoids the computationally intensive step of solving the Poisson equation by considering each point of lightning as a point charge that influences every other grid point. Each time step then amounts to adding the influence of the new point across the grid as opposed to solving a large sparse linear system.

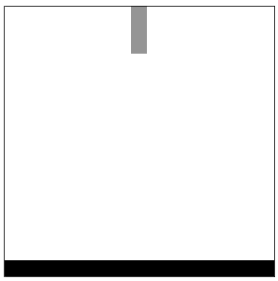
Finally, we consider DLA. DLA has been used to simulate lichen, frost, and other Laplacian growth phenomena but is limited in the number of structures that it can create as pointed out in [3]. In DLA a random walk starts from a random location in the grid and continues until adjacent to the lightning. Once adjacent, this point is considered as growth point for the lightning with some stickiness factor. If the point sticks, the lightning grows to this point and a new random walk is started. The process repeats until the desired size of lightning is achieved.

These methods generate fractals of various dimensions and have hyperparameters to control the fractal dimension. For example, in DBM, instead of weighting the random selection proportional to the potential difference, it can be weighted proportional to some arbitrary power η of the potential difference for $\eta = 0$ every point adjacent to the lightning is equally likely to be the growth site while for $\eta \geq 4$ the lightning is very nearly a straight line.

1.3 Problem with Growth Models

Although the traditional DBM algorithm presents a means of generating sophisticated and beautiful visualizations of lightning, the algorithm is at once computationally slow and memory intensive [3]. In fact, when run, DBM iterates over each point on the grid and calculates the charge's effect on that space via a discretized Laplace equation—and then repeats this process for as many time steps as desired, updating each element of the grid at every time step. Obviously, this is not an efficient method of generating each time step; this is all the more so given that massive grid sizes, up to the size of 256^3, are necessary to have a high level of precision and detail in visualizations. Such grid sizes occupy up to 469MB of memory, assuming single precision [4].

Recent literature proposes a few different ways to resolve this problem. [4], for one, suggests the use of an adaptive mesh, a means of refining numerical simulations in order to more accurately calculate the value of a point on a grid. In other words, typical numerical simulations use a predetermined time step to calculate a value across an evenly-spaced grid. An adaptive mesh, on the other hand, spaces the grid differently in order to allow more frequent or more accurate updates at highly variable regions of the grid. For example, in the case of lightning, it is unnecessary to simulate the regions far away from any darts in high detail, but regions closer to the dart should be refined to maximize



(a)

	1	
1	-4	1
	1	

(b)

Figure 1: (a) Boundary condition on a 17×17 grid with initial breakdown. Grey cells with charge $\phi = 0$ represent lightning and black cells with $\phi = 1$ represent the ground. (b) 2-D Laplacian stencil for cell on interior

accuracy and resolution in the model.

However, while the methodology used in [4], including the adaptive mesh, is effective, it can be improved. For example, the authors note that the Incomplete Cholesky factorization is inefficient—suggesting that there are other algorithms, or preconditioners to algorithms, that may be worthwhile examining.

2 Methods

We now discuss the explicit mathematical formulation of our methods, following from previous work and the physical principles outlined above, first DBM and then DLA. For both of the methods we start with an initial charge configuration. We choose to use a boundary condition based on the initial breakdown of lightning with a potential distribution as in Figure 1.

2.1 Traditional Dielectric Breakdown Model

The algorithm for traditional DBM to generate a single leader can be summarized as follows:

1. Initialize a charge distribution based on initial breakdown
2. Calculate the potential distribution over the entire grid based on boundary conditions as governed by the Poisson PDE shown in equation (1) below.
3. Randomly select one of the adjacent cells for the lightning to jump to and update the boundary conditions to reflect this
4. Repeat steps 2 and 3 until the process until the lightning hits the ground

We now outline each of these steps explicitly, developing the mathematical basis for our methods.

Step 1: We start with some initial configuration such as that given in Figure 1a.[1] This provides us with initial boundary conditions with which to solve for the potential ϕ at every cell.

Step 2: From our boundary conditions, we can spread the charges of the lightning and ground through by solving the Laplacian equation, derived directly from the first and second Maxwell equations for an electric field.

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

With two dimensional finite differences we have

$$\begin{aligned} \nabla^2 \phi_{i,j} &= \left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \right) \phi_{i,j} \\ &= \frac{\phi_{i+1,j} - 2\phi_{i,j} + \phi_{i-1,j}}{h^2} + \frac{\phi_{i,j+1} - 2\phi_{i,j} + \phi_{i,j-1}}{h^2} \\ &= \frac{\phi_{i,j+1} + \phi_{i+1,j} - 4\phi_{i,j} + \phi_{i-1,j} + \phi_{i,j-1}}{h^2} \end{aligned} \tag{2}$$

For interior grid cells this looks like the stencil in Figure 1b. We get an equation as above for each point on the grid, resulting in $N \cdot M$ linear equations for an $N \times M$ grid. This linear system can be represented as the matrix product

$$L_{N \times M} \cdot \phi = 0$$

where the construction of $L_{N \times M}$, including boundary conditions, is detailed in Appendix 1 and where ϕ is a vector of length $N \times M$ created by stacking the N length-M row vectors of ϕ values on the grid. This is a sparse linear system that we can now solve using a variety of techniques for the grid potentials ϕ.

Step 3: From the solved ϕ distribution we move onto the third step. Here we randomly select a cell adjacent to the lightning to jump to. Let I be the index set of adjacent cells at the frontier of the lightning. Then, because $\phi_j = 0$ at a point j in the lightning, the potential difference between the lightning and a point $i \in I$ is just $\phi_i - 0 = \phi_i$. Our physical intuition, as detailed above, suggests that we should weight larger charge differences (which are larger ϕ_i) as more likely growth sites. This idea is echoed in previous work by [2] and [3]. An intuitive equation, then, for the probability of growth at an adjacent site i to the lightning is:

$$p_j = \frac{(\phi_j)^\eta}{\sum_{i \in I} (\phi_i)^\eta} \tag{3}$$

In the above equation, η is a hyperparameter that allows for control of the branching frequency. If $\eta = 0$ each growth site is equally likely, while if $\eta \geq 4$

[1] It is important to note that while it is useful to think about this grid as a simple 17×17 grid, [3] later goes on to note that a grid space of 256^2 is necessary to produce acceptable resolutions.

then the branching is heavily influenced to the largest potential, which is that closest to the ground. For large η, then, the bolt is nearly straight. Adjusting η within the 0 to 4 range produces a variety of one and two-dimensional structures.

Step 4: At this point we have added a point to the lightning and have a new boundary condition, which requires solving the Laplace equation from step 2 again before adding a new point. Repeating this process until the lightning reaches the ground we generate a full leader.

Note that this can easily be extended to more than two dimensions, say for instance if it were of interest to generate a three-dimensional visualization.

To generate simulations on a grid that does not have boundary artifacts from edges requires grids of at least 128 cells in each dimension. With a grid that is 128^2 cells in 2D we have $L_{128\times128}$ is a $128^2 \times 128^2$ matrix. Extending to three dimensions we could have a sparse linear system with dimensions $128^3 \times 128^3$. This matrix can barely be stored in memory, let alone easily manipulated and as such our implementation uses almost exclusively sparse matrices and methods. Solving this system is the most computationally expensive part of DBM and our efforts are largely focused on speeding, or bypassing, this step.

2.2 Dart Leaders

After the initial leader hits the ground, a physically based DBM model generates several subsequent dart leaders, each time depositing residue charge in the wake of the previous leader. This means after the process above is carried out, the process must be repeated for a number of further leaders.

The residue charge can be incorporated by acknowledging the Laplace equation as a special case of the Helmholtz equation, also derived from Maxwell's equations:

$$\left(\nabla^2 + \left(\frac{\omega}{c}\right)^2\right)\phi = -4\pi\rho \tag{4}$$

Here $-4\phi\rho$ is the charge density on the grid. The term $\frac{\omega}{c}$ is near 0 in this case as lightning's angular velocity ω is negligible relative to its linear velocity, which is very near the speed of light c. Eliminating this term, the equation is similar to the Laplace equation already seen with an additional electromagnetic term representing the existing charge on the grid.

$$\nabla^2\phi = -4\pi\rho \tag{5}$$

For our stepped leader, the right hand side is 0 as no residue charge deposits are present in the air. For subsequent dart leaders, we choose a small value for ρ to have a minor bias towards the previous path the most recent leader. ρ is recalculated after each leader is generated, depositing positive charges along the previous leaders path and growing lightning in generally the same path for each leader. This does not change our discretized Laplacian system beyond the right

hand side of the equation and so the algorithm from above applies still for the subsequent leaders.

2.3 Efficient Poisson Solvers

The most computationally expensive step in our DBM algorithm is solving the Poisson equation for each time step as we add points to a leader, and is therefore our target to increase the efficiency of simulation. We examine the following methods, including Incomplete Cholesky Conjugate Gradient method as used in [4] and novel improvements:

2.3.1 Sparse Inversion

A brute force, initial baseline for solving the system $Ax = b$ is to invert the matrix A to solve as $x = A^{-1}b$. In our implementation the matrix A is large sparse Poisson system, which we solve with `scipy.sparse.linalg.inv`. This approach, while reasonable on small grid sizes, has a time complexity that scales poorly with grid size. For a grid with $n = 128^2$ cells, inverting A requires $\mathcal{O}(n^3)$ operations. This becomes substantially worse on larger grid sizes. Furthermore, the inverse of a sparse matrix is nearly always dense, requiring a space complexity proportional to the size of A, or $\mathcal{O}(n^2)$.

Moreover, given that A is updated as points are added after each iteration, sparse inversion, when used for each iteration, solves an entirely new problem, and as such is more computationally intensive. This is not a good method, but serves as a useful baseline for comparison

2.3.2 Cholesky Decomposition

We can improve upon sparse inversion by noting that our matrix A is symmetric, positive-definite, meaning that it has a unique decomposition as LL^T, called the Cholesky decomposition. Here L is lower triangular. After calculating this factorization of A, we have the system $LL^Tx = b$. We can solve $Ly = b$ for y using forward substitution and then $L^Tx = y$ for x using backward substitution. Furthermore, our implementation of boundary conditions, and the fact that only one boundary condition is added at each time step means the matrix A is altered in only one row and column per time step. This allows us to use rank-1 updating to update the Cholesky factor L without recalculating the full factorization [5].

Using an implementation of Cholesky factorization specific to sparse matrices `sksparse.cholmod.cholesky`, the time complexity of solving the system is decreased to $\mathcal{O}(n^{3/2})$ [6] and space complexity is also improved as a full, dense matrix the size of A is not stored.

2.3.3 Incomplete Cholesky Conjugate Gradient (ICCG)

The Incomplete Cholesky Conjugate Gradient method (ICCG) uses an Incomplete Cholesky preconditioner with the conjugate gradient (CG) method to ensure efficient convergence of the CG algorithm.

Given some constant preconditioner M that approximates A and some guess for x, the algorithm for the preconditioned conjugate gradient (PCG) method can be described by Algorithm 1.

Algorithm 1 Preconditioned Conjugate Gradient

1: $\mathbf{r} = \mathbf{b} - A\mathbf{x}$
2: Solve $M\mathbf{z} = \mathbf{r}$ for $\mathbf{z}$
3: $\mathbf{p} = \mathbf{z}$
4: $q = \mathbf{r}^T\mathbf{z}$
5: **while** True **do**
6: $\quad \alpha = q/(\mathbf{p}^T A\mathbf{p})$
7: $\quad \mathbf{x} = \mathbf{x} + \alpha\mathbf{p}$
8: $\quad \mathbf{r} = \mathbf{r} - \alpha A\mathbf{p}$
9: $\quad$ **if** q sufficiently small **then** exit loop
10: $\quad$ **end if**
11: $\quad$ Solve $M\mathbf{z} = \mathbf{r}$ for $\mathbf{z}$
12: $\quad \beta = \frac{\mathbf{z}^T\mathbf{r}}{q}$
13: $\quad \mathbf{p} = \mathbf{z} + \beta\mathbf{p}$
14: $\quad q = \beta q$
15: **end while**
16: **return x**

This algorithm is equivalent to applying the regular Conjugate Gradient method to the system $M^{-1}Ax = M^{-1}b$. Iterations of the conjugate gradient method when unconditioned is related to the condition number of A. By choosing M such that $M^{-1}A$ has a minimal condition number, we can substantially speed up the convergence of the algorithm. One such M that works well is the Incomplete Cholesky preconditioner. We modify this factorization, which takes $M = LL^T$ to be $M = LDL^T$ to avoid taking square roots in the factorization for a minor speed up. To maintain sparsity, as well, we ensure L has the same sparsity pattern as A, zeroing all elements that are not. In line 2 of our algorithm, then, we have

$$M\mathbf{z} = \mathbf{r} \implies LDL^T\mathbf{z} = \mathbf{r} \tag{6}$$

This system can be solved very efficiently using backward and forward substitution and so steps 2 and 11 are not too computationally expensive to weigh out the benefits of the lower condition number.

2.3.4 Incomplete Poisson Conjugate Gradient (IPCG)

The Incomplete Poisson Conjugate Gradient (IPCG) method applies the same preconditioned conjugate gradient algorithm that is outlined for the Incomplete Cholesky Conjugate Gradient (Algorithm 1), but instead preconditions specifically Poisson matrices, like those described in the first appendix.

We can decompose A additively such that $A = L + D + L^T$, where L is a lower triangular matrix and D is diagonal. These preconditioners are positive-definite and symmetric, so we can moreover easily decompose M such that $M^{-1} = KK^T$, where $K = I - LD^{-1}$. It then follows in line 2 of our algorithm that we have

$$M\mathbf{z} = \mathbf{r} \Longrightarrow \mathbf{z} = KK^T\mathbf{r} \tag{7}$$

This preconditioner, tailored specifically to the problem at hand with the Poisson equation, functions even better than the incomplete Cholesky preconditioner, and requires only a single matrix multiplication (instead of forward and backward substitution) at each conjugate gradient iteration. The method by which it decreases the condition number more effectively for the Poisson problem is described in [7].

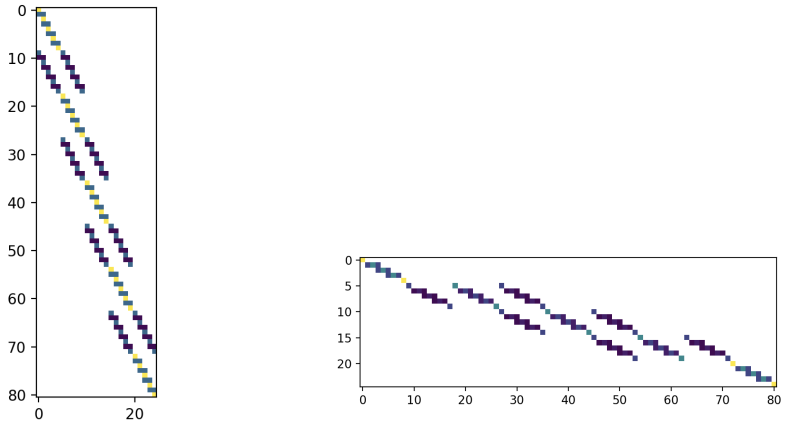
2.3.5 Multigrid Method

In each of the methods discussed and applied above, updates to an individual cell take particularly long to propagate across the matrix, or grid, due to the level of granularity associated with large matrices—the methods apply locally. As such, per the lecture notes for the multigrid method covered in class, such methods will take at least $\mathcal{O}(N)$ operations per point to converge; this is because the lowest order eigenmodes take a long period of time to be "captured."

The multigrid method is a means of avoiding this issue by introducing increasingly coarser grids, upon which there are considerably fewer points, to refine the solution. A multigrid with 4 levels that is applied to a 32×32 matrix, for instance, might have matrices of dimensions 32×32, 16×16, 8×8, and 4×4. On the coarsest grids with the fewest gridpoints, eigenmodes that appear at a much lower frequency can be computed much more quickly.

Per our code, we can generalize this to a series of descending grids, where the original problem is on grid 0 and there are g increasingly coarser grids, labeled $1, 2, ..., g$ (specified by the user through the parameter `levels` when using the function). We apply our function for the multigrid method, `multigrid`, recursively, such that our original matrix starts at the highest level of refinement 0. It is then made increasingly coarse (through a restriction operator, to be discussed below) as it descends to our coarsest level g (which we specify through the parameter `levels`). At the descending levels, a solution operator—in this case a Gauss-Seidel sweep, `gauss-seidel`, is applied. Once at the level g, we apply our method of interest (whether sparse inversion, Cholesky Decomposition, ICCG, or IPCG) to this most coarse grid to solve for a solution, and then "send" the matrix back up to the most refined level (through an interpolation operator, to be discussed below), with which we started. As the matrix ascends, we once again apply the Gauss-Seidel sweep.

It is important to note that we define the number of times the Gauss-Seidel solution operator is applied on the way up and down through the parameters `v_down` and `v_up`. Per the recommendation of the lecture notes, we set the defaults `v_up`= 2 and `v_down`= 0, which yields a similar result to `v_down`= 2 but is less computationally intensive.



(a) (b)

Figure 2: 2(a) is the interpolation matrix used to shift from a $(5, 5)$ to a $(9, 9)$ matrix, and 2(b) is the restriction matrix used to shift from a $(9, 9)$ to a $(5, 5)$ matrix

The algorithm uses restriction and interpolation operators, or `restrict` and `interp` in our code, to shift between these levels. The interpolation operators, which are effectively large rectangular matrices, calculate values at common gridpoints between different levels of grids through linear interpolation to shift from a level i to a level $i - 1$, i.e. up to a more refined level. The restriction operators do the same, but shift from a level i to a level $i + 1$, i.e. down to a coarser level. Matrices for each of these two operators are visualized in Figure 2, where different shades of colors represent different values to interpolate between levels (e.g. $\frac{1}{2}, \frac{1}{4}$, etc.).

We can use the multigrid method in conjunction with any of the methods established above (sparse inversion, Cholesky Decomposition, ICCG, or IPCG). It is important to note that the efficiency of the function in our case is, however, constrained by the implementation of the Gauss-Seidel solver in Python rather than C++.

2.4 DBM-Inspired Algorithm

[3] proposes an algorithm based off traditional DBM, but is altered because the addition of single point charges at each time step does not drastically alter the charge in the whole grid as in the Laplace equation, the most computationally expensive part of DBM.

In this method, the boundary conditions taken on are comparable to those of traditional DBM.

Although it is not explicitly the case in which we simulate lightning, let us consider a negatively charged circle at the center of the grid, with radius R_1,

and a larger positively-charged ring, with radius R_2. The PDE is reduced to an Euler equation in 3 dimensions, whereby the solution is as follows as pointed out in [3]:

$$\phi = c_1 + \frac{c_2}{r} \tag{8}$$

One can then solve for the constants c_1 and c_2 with the boundary conditions:

$$c_1 = -\left(\frac{R_1}{R_2} - 1\right)^{-1} \tag{9}$$

$$c_2 = \left(\frac{1}{R_2} - \frac{1}{R_1}\right)^{-1} \tag{10}$$

Then, as the radius of the positively-charged ring, R_2, goes to infinity, ϕ goes to the following:

$$\phi = 1 + \frac{R_1}{r} \tag{11}$$

This represents a single point charge; it then follows that the potential for a cell can be calculated per the following:

$$\phi_i = \sum_{j=0}^{n} \left(1 - \frac{R_1}{r_{i,j}}\right) \tag{12}$$

where $r_{i,j}$ represents the distance between our cell at i and a point charge at j, and where n is the total number of point charges.

Having established this, it is now possible—using this new method—to update the potential at each viable site on the grid per the following:

$$\phi_i^{t+1} = \phi_i^t + \left(1 - \frac{R_1}{r_{i,j}}\right) \tag{13}$$

where $\left(1 - \frac{R_1}{r_{i,j}}\right)$ represents the contribution of the new point charge to the cell's potential, and ϕ_i^{t+1} and ϕ_i^t are the potentials at a point at i in the next and prior time steps, respectively.

While [3] solves this equation from the polar Laplacian in 3 dimensions, we find different results in two dimensions. In 2 dimensions we have the polar Laplacian equation, for potential function in polar coordinates:

$$\nabla^2\phi(r,\theta) = \frac{\partial^2\phi}{\partial r^2} + \frac{1}{r^2}\frac{\partial^2\phi}{\partial\theta^2} + \frac{1}{r}\frac{\partial\theta}{\partial r} = 0 \tag{14}$$

For our simple circle example, then we have $\phi(R_1,\theta) = 1$ and $\phi(R_2,\theta) = 0$, which do not vary with angle so the second to last term above vanishes and we have the ODE

$$\frac{\partial^2\phi}{\partial r^2} + \frac{1}{r}\frac{\partial\theta}{\partial r} = 0 \tag{15}$$

which again is a Euler equation with solution $\phi = c_1 + c_2 \log r$
Solving for c_1, c_2 yields

$$c_1 = \frac{-\log R_1}{\log R - \log R_1} \tag{16}$$

and

$$c_2 = \frac{1}{\log R - \log R_1} \tag{17}$$

So in 2D we have

$$\phi = c_1 + c_2 \log r = \frac{\log r - \log R_1}{\log R_2 - \log R_1} \tag{18}$$

which diverges as R_2 approaches infinity. We can still use an appropriate value for R_2 such as $Nh/2$ when simulating on a finite grid, however, and this is how we produce 2D simulations.

For either 2 or 3 dimensions, one of the changes that results from this adapted method is that the potentials for each cell are no longer limited to $[0, 1]$, and thus it becomes necessary to normalize all of the charges in the grid. Let ϕ_{min} and ϕ_{max} represent the minimum and maximum potentials on the grid; then, the probability with which a growth site is selected is represented by:

$$p_i = \frac{(\phi_i)^n}{\sum_{j=1}^{n} (\phi_j)^n} \tag{19}$$

where we have, as a method of normalization, that

$$\phi_i = \frac{\phi_i - \phi_{min}}{\phi_{max} - \phi_{min}} \tag{20}$$

Then, following the initial step of the algorithm in which a point charge is inserted and the potential is calculated for the relevant sites to which the charge can expand, the algorithm iterates through the following sequence:

1. Randomly select the cell into which the lightning will expand based upon the p_i for each cell and add a point charge at that cell.

2. Update the potential of all of the sites into which the lightning can expand using the equation above for ϕ_i^{t+1}.

3. Repeat this process starting from the first step.

2.5 Diffusion Limited Aggregation

Diffusion limited aggregation (DLA) is an alternative to the Dielectric Breakdown Model that can be used to simulate Laplacian growth. One critical difference between the two methods is their derivations:

DLA starts similarly to DBM, with a point in the center of an $n \times n$ grid (or, in the case of lightning, at the top of such a grid). However, instead of selecting a growth side based upon a weighted probability that takes into account the electric charge, another random point on the grid is selected. The first point is the "lightning" and the second is the origin of an n dimensional random walk. The random walk is allowed to continue until it occupies a grid space within one grid space (adjacent to) the lightning. The new space then is added to the "lightning." This method is highly intuitive, but does not yield the desired visualization for lightning since the random walks tend to add to the structure horizontally as much as they do vertically.

In other words, DLA, as applied normally, is more apt to simulate particle collisions and particle aggregation than Laplacian growth; the reason for this is the lack of directionality. The method allows for growth from any site, each equally approximately as likely as the next (in the early timesteps). While in DBM, the ρ term "forces" the equation to grow in one direction, it does not exist in the DLA model. Instead, a particle can walk until it reaches a cell occupied by the "lightning." In the early stages, there are no branches to "block" the random walk and thus the probability of growth from any given site is almost equal—this ultimately produces a structure that branches outward in a branch-like fashion more than a lightning-inspired bolt. Compared to DBM, this is drastically different. It is clear from Figure 3 that the model does not have the best branching patterns to support lightning.

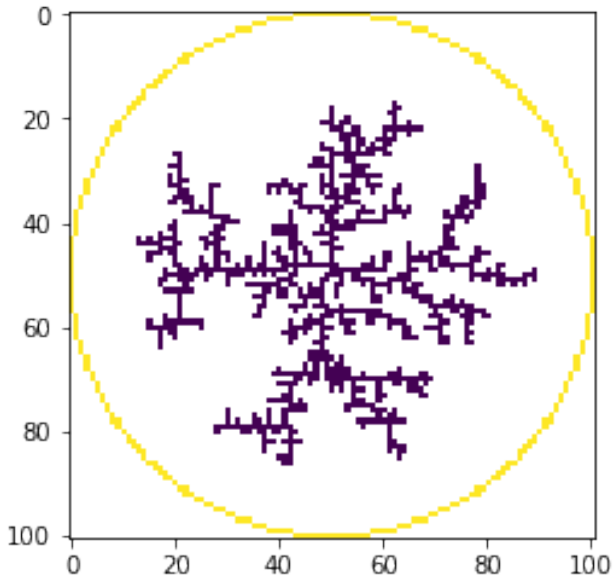


Figure 3: DLA on a 100 x 100 grid. Note the random, semi-symmetric branching pattern.

In order to tune the model and introduce some semblance of non-randomness in an otherwise random model, a stickiness parameter is introduced. This stickiness parameter dictates the probability that any given "encounter" between the randomly walking point and the lightning results in it being added to the lightning. A greater stickiness parameter means a greater probability of sticking. In the event that a particle does not stick, it is simply allowed to continue walking. If this results in another "encounter" immediately after the first, this is permitted. The grid space is not excluded. The stickiness parameter can thus be considered analogous to the η in DBM and helps to control the thickness of the model. As with η, a higher value corresponds to a thinner model.

As mentioned, the main limitations of DLA are the lack of a first principles derivation and a lack of directionality. While the first of these issues is impossible to remedy without fundamentally changing the model and its approach, the latter deficiency is not so inherently critical to the model that it cannot be fixed. One possible way of injecting directionality into the model is simply weighting the probability of selecting any given grid space based on the current direction of growth. The probability that a random walk travels a given distance d is inversely correlated with d (equation 21 shows the root mean squared error in 1 dimension for N steps). This means that biasing the initiation sites of the random walk can also lead to a directionally biased model.

$$RMSD = \sqrt{N} \tag{21}$$

The algorithm is outlined below:

Algorithm 2 Diffusion Limited Aggregation

```
procedure GRID(n, n)                          ▷ initialize an n x n grid
    Grid ←lightning                           ▷ sets initial lightning positions
    steps = 50
    for all steps do
        start = random(n,n)        ▷ Select a random point to begin the walk
        while No lightning has been added do
            if adjacent to lightning then
                if point sticks then
                    update Grid
                else
                    resume walk
                end if
            end if
        end while
    end for
    return Grid
end procedure
```

2.5.1 Tip-Biased Random Walk Start Points

As mentioned above, a random walker can take a potentially infinite number of steps before impacting the lightning; the average number of steps taken before an "encounter" only increases as the number of dimensions increases. This issue becomes especially apparent in a larger grid, where the majority of points are a great distance away from the nearest lightning, especially at early timesteps. In order to counteract this, start points for the random walk were biased. Ultimately, the updated model would have the greatest probability of selecting a point k units away from the lightning, with the probability decreasing for points both closer and farther away than distance k. Grid spaces greater than distance k were not considered to be viable start points. Some sample probabilities are outlined below, with a k value of 3.

Distance	0	1	2	3(k)	4	5	6
Probability	0	.1	.2	.4	.2	.1	0

Table 1: This table shows probabilities of selecting a point for a random walk starting point for given distances and a k-value of 3.

The biased random walk start points can also be used to introduce directionality in the model. By biasing the random walker's starting position to be within distance k of the lightning's tip, it is more likely to end up extending the dart rather than branching off from it.

To accomplish this, instead of calculating the distance from all the points of lightning on the grid, only the distance from the tip must be calculated. The Euclidean distance of all given points was calculated from the tip of the dart leader.

This distance matrix was then normalized such that the points at distance k had the greatest probability of being selected as a start point for the random walker. After this, the rows were normalized to be between 0 and 1 and the entire matrix divided by the sum of the normalized matrix in order to ensure the probabilities summed to 1. All values less than zero, which corresponded to points greater than distance $2k$ from the lightning, were set to 0. In order to do this, the following equation was applied to all elements of the matrix:

$$\frac{\text{normalize } (k - |k - G_{a,b}|)}{\sum(\text{normalize } (k - |k - G_{a,b}|))} \tag{22}$$

This yielded a probability matrix from which a new point could then be selected, taking into the account the weights at each grid space.

Pseudocode for the algorithm is shown below:

Algorithm 3 Tip-Biased Diffusion Limited Aggregation

```
1: procedure GRID(n, n)
2:     Grid ←lightning
3:     steps = 50
4:     for all steps do
5:         bottom = lowest-point(Grid == lightning)        ▷ Lowest point of lighting
6:         distances = calc-distances(bottom)              ▷ Find the distances
7:         probs = generate-prob(distances)                ▷ Convert to probabilities
8:         while No lightning has been added do
9:             if adjacent to lightning then
10:                if point sticks then
11:                    update Grid
12:                else
13:                    resume walk
14:                end if
15:            end if
16:        end while
17:    end for
18:    return Grid
19: end procedure
```

3 Results

3.1 Dielectric Breakdown Model

Our simulations of lightning using DBM (with the IPCG method) can be seen in Figure 4, in which we take snapshots of the simulation for a 256×256 grid and $\eta = 3$ at various timepoints. In Figure 4, we stylized the appearance of the lightning by adjusting certain graphing settings—giving it the appearance of real lightning. Furthermore, we have included a video of the full simulation in Figure 4 along with this PDF and our code.

We varied lightning

There were a number of variables to consider in comparing the outcomes of the DBM simulations: grid size, selection of η, selection of the four methods (sparse inversion, Cholesky Decomposition, ICCG, IPCG), and use of the multigrid method. Given that each of our methods ultimately produces similar plots, we we were most interested in examining the relationship between grid size and simulation time by each of our methods.

As expected, computation time increases drastically as grid size increases. Per Figure , of the four methods used, IPCG was the quickest—and handled increases in grid size best—and was then followed by Cholesky Decomposition, then ICCG, and finally sparse inversion, which required more than thirty minutes for a 95×95 matrix, and was therefore not included in our plot.

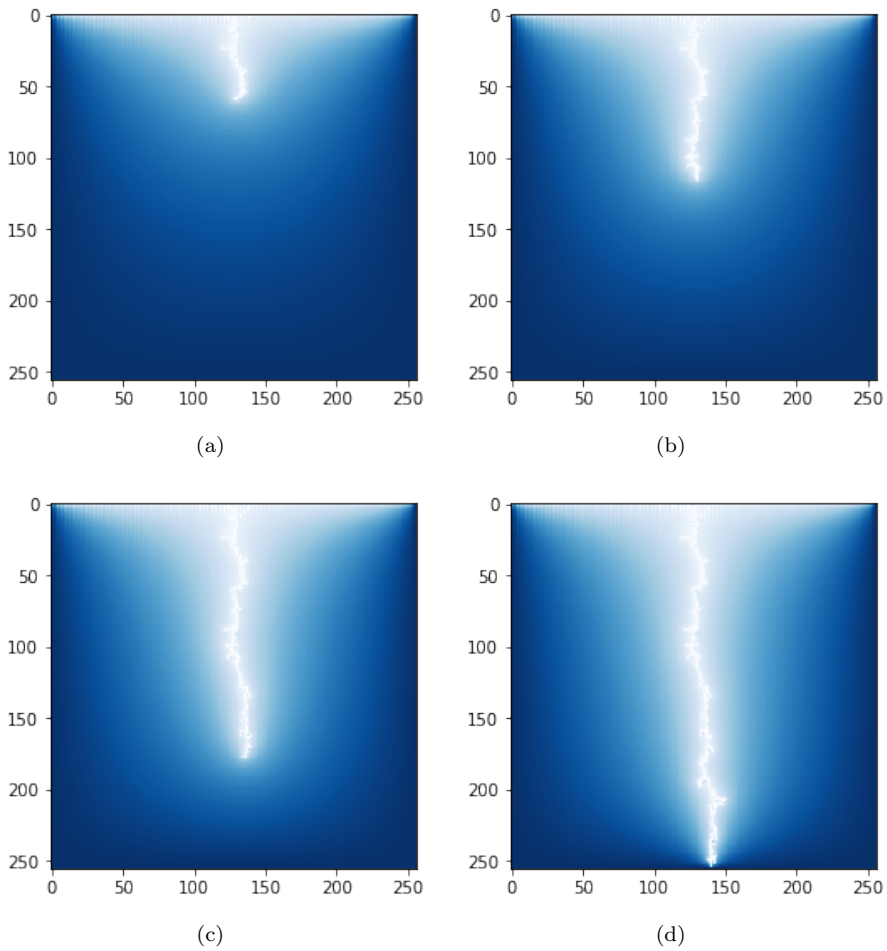


Figure 4: These figures highlight a 256×256 lightning generation using the DBM algorithm with the IPCG method for $\eta = 3$ at various time steps as it is generated, where we have stylized the appearance of the lightning by changing certain graphing settings.

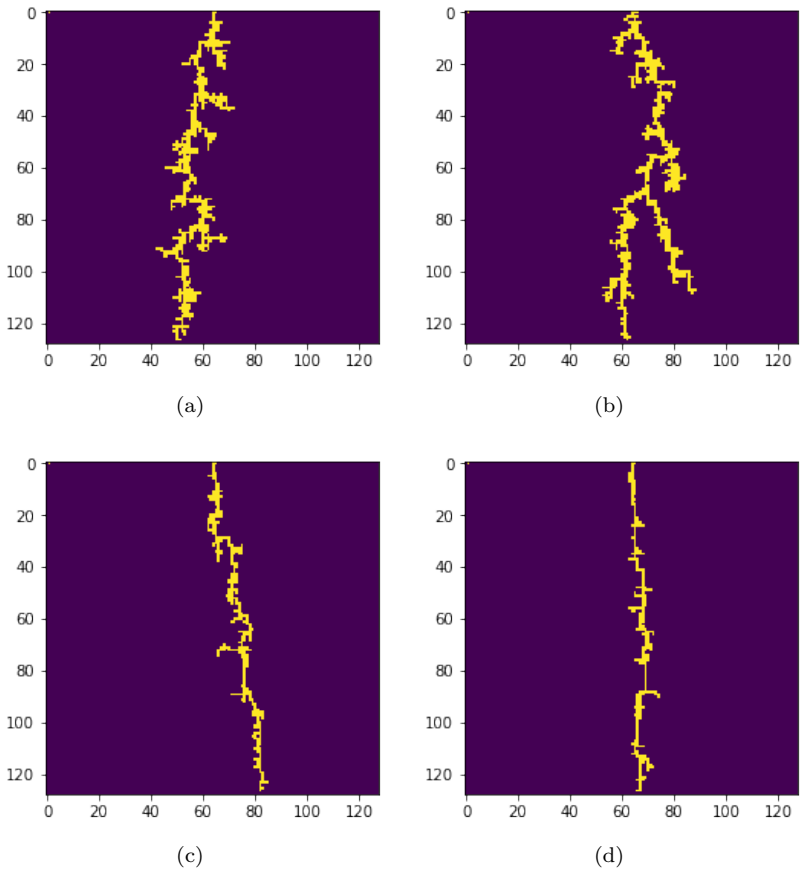


Figure 5: These figures compare a 128×128 lightning generation using the DBM algorithm with the IPCG method for various η, where 5(a) corresponds to $\eta = 1$, 5(b) to $\eta = 2$, 5(c) to $\eta = 3$, and 5(d) to $\eta = 4$. We can clearly see that that the lightning becomes increasingly thinner and less branched for higher values of η.

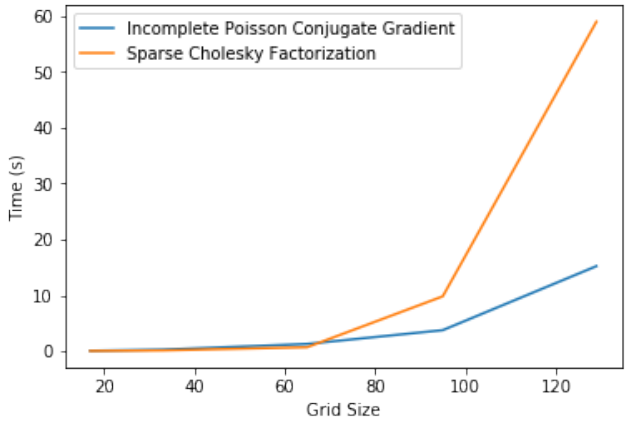


Figure 6: The above plot shows the time in seconds for the DBM model at various grid sizes for both Incomplete Poisson Conjugate Gradient and Sparse Cholesky Factorization

3.2 DBM-Inspired Algorithm

In this section, we consider two main sets of results:

1. In figure 7, we compare lightning generation with the DBM-inspired algorithm on a 128×128 grid for varying η.

2. In figure 8, we examine the time that the DBM-inspired algorithm demands for lightning generation for various grid sizes.

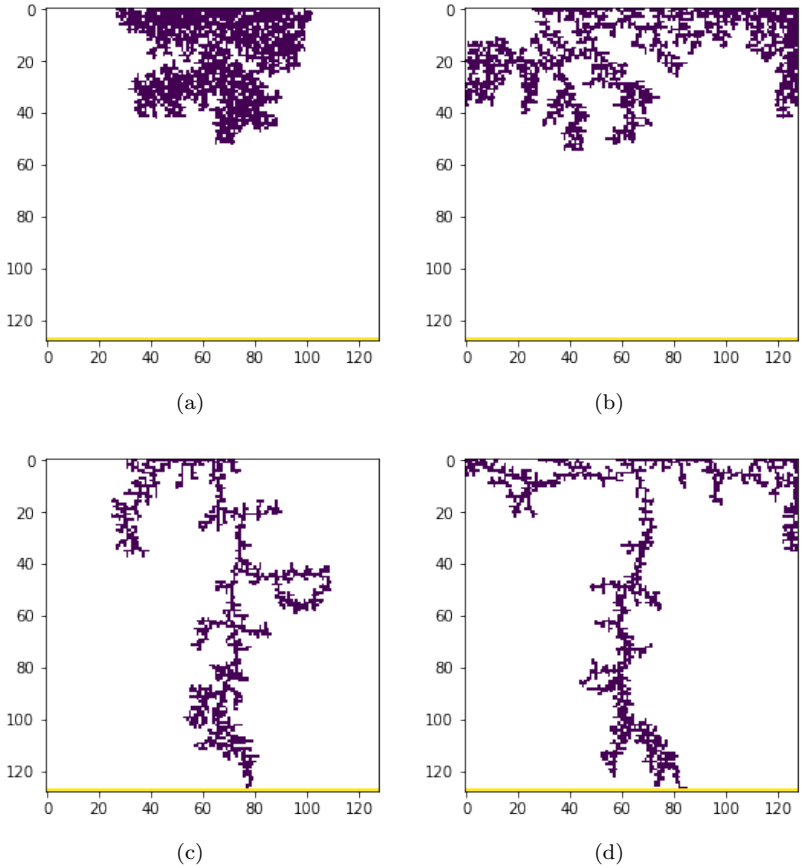


Figure 7: These figures highlight a 128×128 lightning generation using the DBM-inspired algorithm for varying η: 7(a) with $\eta = 2$; 7(b) with $\eta = 4$; 7(c) with $\eta = 6$; and, 7(d) with $\eta = 8$. Here, we notice that the branches becomes thinner for higher η, as expected. We do not reach the floor of the grid for all η because the number of iterations is cut off at 2000.

3.3 Diffusion Limited Aggregation

The two primary objectives of improving DLA were increased efficiency and more lightning-like behavior. As the results below show, both of these objectives have been accomplished. Tip-biased DLA is about 3 times as fast as normal DLA and much more directed.

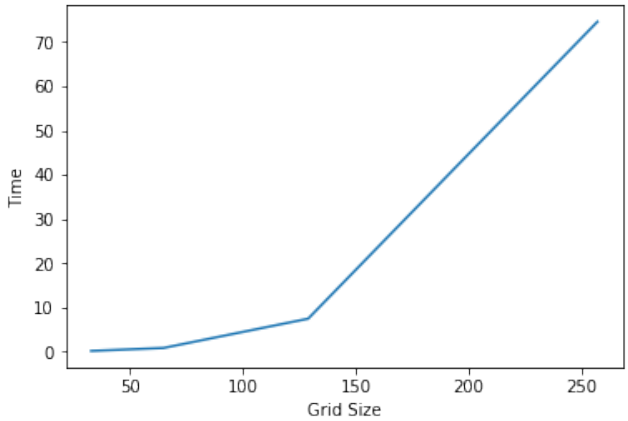


Figure 8: The above plot shows the time in seconds for the DBM-inspired at various grid sizes. As grid size increases, the time to generate lightning using the DBM-inspired model increases significantly.

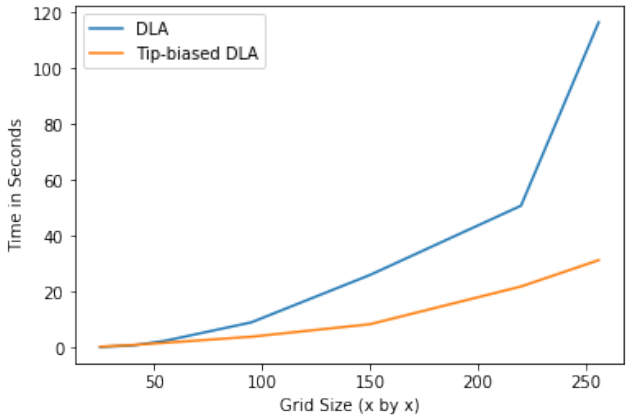


Figure 9: The above plot shows the time in seconds for DLA and tip-biased DLA.

3.3.1 DLA vs Tip-Biased DLA Speed

Grid Size (n x n)	DLA Time	Tip-Biased DLA Time	Ratio $\left(\frac{DLA}{Tip-biased}\right)$
25	0.1999	0.3002	.67
40	0.7559	0.9290	.81
55	2.4048	1.7061	1.41
95	9.0232	3.8864	2.32
150	26.0105	8.3660	3.11
220	50.8343	21.8839	2.32
256	116.5688	31.3275	3.72

Table 2: Table containing the times and ratio of times for each grid size.

In figure 9, runtime is plotted against grid size. It is easy to see that as grid size increases, the difference in time grows. Table **??** shows that it takes almost half as long to generate lightning on a given grid size with the tip-biased method as opposed to the original implementation. All results are using a distance of 3 for the maximum probability of spawning a random walk.

3.3.2 DLA Biasing and Directionality

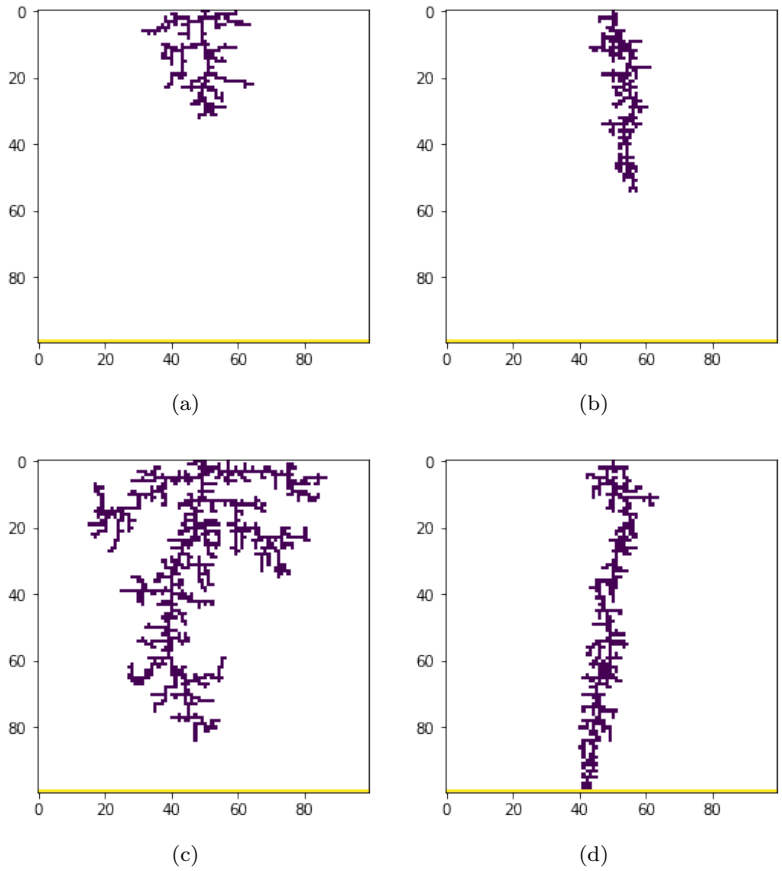


Figure 10: 10(a) is the undirected method at 250 iterations; 10(b) is the directed method at 250 iterations; 10(c) is the complete undirected method at 250 timesteps; and, 10(c) is the complete directed method at 250 timesteps.

In figure 10, the lightning approximation from DLA is shown in two dimensions. In figures 10(b) and 10(d), there is a clear directionality while in figures 10(a) and 10(c), the lightning seems to randomly branch off early and often. All results are using a distance of 3 for the maximum probability of spawning a random walk.

3.3.3 Tip-biased DLA and Distance

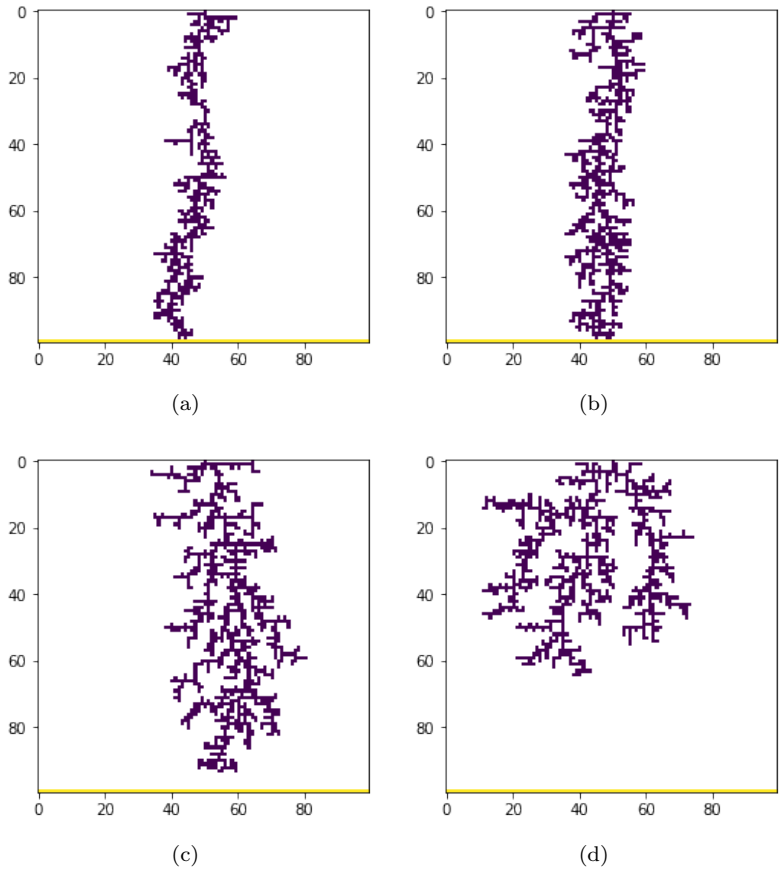


Figure 11: 11(a) is the undirected method at 250 iterations; 11(b) is the directed method at 250 iterations; 11(c) is the complete undirected method at 250 timesteps; and, 11(d) is the complete directed method at 250 timesteps.

Figure 11 demonstrates the directionality imposed by the tip bias. As the distance with the maximum probability of spawning a random walk increases, the lightning becomes less linear.

Figure 12 clearly show the increased efficiency from biasing random-walk start points towards the tip. As the distance with the maximum probability of spawning a random walk increases, so too does the time.

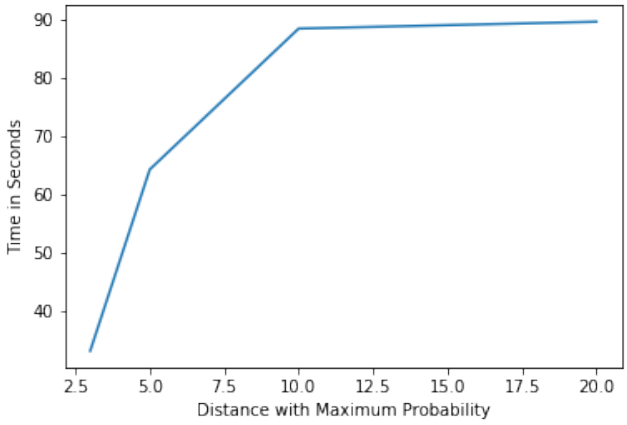


Figure 12: The above plot shows the time to generate lightning for various distances with a maximum probability of spawning a random walker.

4 Discussion

4.1 Dielectric Breakdown Model

The Dielectric Breakdown Model is the slowest of our models, but with our introduction of the Incomplete Poisson Conjugate Gradient we achieve speeds much faster than the speeds of the Incomplete Cholesky Conjugate Gradient as proposed by [3]. The speeds for IPCG approaches the speed of DLA which is quite an accomplishment for a physically based system. These results could be made even faster when combined with multi grid. Unfortunately, limitations of the language of implementation, Python, inhibited our ability to lower constants of proportionality in the implementations of some of our algorithms. For example, our Gauss-Seidel algorithm, as implemented in Python, is much slower than some of the library functions for sparse matrices. Using library functions focused on sparse matrices for methods like multi grid would be one direction for future improvement for the DBM model we created. Despite the fact that not every algorithm was implemented in a low-level language, we still achieved remarkable results with the varied preconditioner. Generating, in under 20 seconds the full lightning bolt on a grid of 129×129 is much better than either the sparse Cholesky factorization, or preconditioning with Incomplete Cholesky.

The lightning generated is also very realistic. For various η values we see results comparable in accuracy to those of [3] but area able to generate them orders of magnitude faster, even for large grid sizes. When augmented by a C++ implementation of their OctTree refinement or the multigrid method, this difference could be even more profound. Our method demonstrates that even without these enhancements, it is possible to generate lightning quickly and robustly with a pre-conditioner that, to our knowledge, has not been applied to the problem of Laplacian growth.

A future investigation could look at an adaptive mesh refinement using the Incomplete Poisson pre-conditioner we implemented. This would benefit from the fact the we would not need to solve over the entire grid and could solve a refined grid only in areas around the lightning and could provide results even faster than a multigrid combination with the pre-conditioner.

4.2 DBM-Inspired Algorithm

Our DBM-inspired model in 2 dimensions is much faster than DBM but is still slower than DLA when using tip-biased DLA with a small tip bias. Our DBM inspired model is not optimized in the sense that it computes the potential ϕ at every point on the grid for every time step (granted in a more efficient way than solving the Poisson equation). If instead we had computed the potential at values adjacent to the lightning only, our DBM inspired model might have times more comparable with that of DLA. Because the lightning is nearly a column for $\eta \geq 5$, the potential would be solved at only a number of grid points proportional to $1/N$ of the grid points that we currently solve at. This would substantially speed up computations across all grid sizes. Even despite the fact that this method is not fully optimized, it outperforms even the fastest DBM model that solves the Poisson equation and does so by considering boundary conditions as point charges. This decrease in overhead per time step is clearly worth it in the efficiency of algorithm but makes η harder to interpret.

As shown in the visuals above, much higher η values are required for this model than for the traditional DBM model. This can generally be attributed to the fact that as each charge is added, the magnitude of the charge before normalizing increases. Because of this, the distribution of charges of the entire grid after normalizing is concentrated much closer to the negative side $\phi = 0$, than the positive side $\phi = 1$. This makes low η values closer to random as the sum of lightning point charges near the lightning creates a relatively uniform charge distribution in any direction around the lightning. Higher values of η magnify any differences in charge to allow the slight differences that do exist from the ground charges to show more fully.

Another contribution of our paper is the examination of this model in 2 dimensions. The influence of charges in the polar domain for 2 dimensions leads to a different result than that in 3 dimensions proposed by [3]. This 2 dimensional representation is limiting in that we cannot approach the infinite limit of point charges as we can in 3d, which causes boundary artifacts in the results. Solving the 2D Laplacian in polar coordinates, however, does make this method physically based much like the traditional DBM model. This is an advantage this model has over DLA. In DLA we do not solve for a full charge distribution as the aggregation of lightning is based on random walks and not charges. There is a clear tradeoff across the three models between physical basis and speed. This DBM inspired model sacrifices some of the physical founding of the traditional DBM model as it only approximates a solution to the Laplace equation and has some dependence on the boundary, working better with radial configurations like the point and sphere configuration shown above.

Despite these limitations, however, it is based on the Laplacian equation and the physics that dictate development of electrostatic charges in lightning and offers a significant speed up over the other model.

4.3 Diffusion Limited Aggregation

4.4 Improvements

It is clear from the above solve results that tip-based DLA is a significant improvement over normal DLA in terms of speed and visual accuracy.

The speed of the algorithm was improved by a factor of nearly 4 at larger grid sizes. While the algorithm was slower at first, for grid sizes under 55, it quickly began to outperform normal DLA. Given that it has been noted a grid of 256^3 is necessary to properly render the lightning, tip-based DLA is undoubtedly superior to normal DLA for rendering lightning on larger grids. This increase in speed is due to the closer starting distance. As mentioned previously, a random walk in DLA could be nearly infinite if it starts far enough away from the lightning. However, by starting it closer, the time of the random walk, and thus the lightning generation, is significantly decreased.

Moreover, the lightning produced by tip-based DLA was more visibly accurate when compared with normal DLA. By introducing a distance-based probability relative to the tip, random walks are more likely to start near the tip and thus more likely to impact and stick to the tip of the dart. Thus, the bolt is likely to continue to grow a given direction, similar to actual lightning.

Varying the distance of the maximum probability of spawning a random walk clearly speeds up the time and directionality. We found a distance of 3 to be ideal for simulating lighting similar to the DBM models above with an *eta* value of greater than 6. While this means the random walker must only take 2 steps to "stick" to the lightning, it allows for accurate and fast simualtions of lightning on large grid sizes.

4.4.1 Limitations

While there were improvements to DLA in tip-based DLA, there were also shortcomings. First of all, the directionality is still lacking. 10 only shows one possible starting condition and can grow only downard. This is due to the algorithm's implementation. The algorithm searches for the deepest part of the dart and calculates distances based upon that. Instead, in the future, it should seek out the direction the dart is travelling dynamically. This could be implemented possibly by allowing a few random iterations and identifying the predominant direction.

However, the ideal solution to the directionality issue would be to bias the bolt towards different potentials, just as real lightning behaves. Instead of biasing it in a random direction, a potential gradient could be calculated as it is in DBM, thus introducing some true physical properties of lightning into the model.

Accordingly, the final issue with tip-based DLA is its lack of first principles. While the bolt may appear similar to real lightning, its behavior in an electric field would not be similar to actual lightning, meaning the model is only accurate as a simulation of appearance and not one of behavior. Any true simulation should seek to emulate all aspects of a phenomena, not only the appearance.

Appendices

1 Discretized Poisson Implentation

Here we offer an explanation of the implementation of our methods for solving the Poisson equation in DBM, starting with the construction of the Poisson matrix and then the imposition of boundary conditions on this matrix

1.A Construction of the Discretized Poisson Operator

In this section we offer a efficient method for constructing the discretized Poisson operator in arbitrary dimensions using building functions from the `scipy.sparse` library. Given the 2D discretization of the Laplacian operator in equation (2) that corresponds to the stencil from Figure 1b, we can consider a simple 3×3 grid of potentials. The corresponding matrix for the linear system arising from equation (2) at each point would be represented in the 9×9 matrix:

$$
\begin{aligned}
L_{3\times 3} &= \left[\begin{array}{ccc|ccc|ccc}
-4 & 1 & & 1 & & & & & \\
1 & -4 & 1 & & 1 & & & & \\
& 1 & -4 & & & 1 & & & \\
\hline
1 & & & -4 & 1 & & 1 & & \\
& 1 & & 1 & -4 & 1 & & 1 & \\
& & 1 & & 1 & -4 & & & 1 \\
\hline
& & & 1 & & & -4 & 1 & \\
& & & & 1 & & 1 & -4 & 1 \\
& & & & & 1 & & 1 & -4
\end{array}\right] \\
&= \begin{bmatrix}
L_3 - 2I_3 & I_3 & \\
I_3 & L_3 - 2I_3 & I_3 \\
& I_3 & L_3 - 2I_3
\end{bmatrix}
\end{aligned}
\tag{23}
$$

Where L_3 is the 1-dimensional discretized Laplacian:

$$
L_3 = \begin{bmatrix}
-2 & 1 & \\
1 & -2 & 1 \\
& 1 & -2
\end{bmatrix}
\tag{24}
$$

We can further separate

$$
\begin{aligned}
L_{3\times 3} &= \begin{bmatrix} L_3 - 2I_3 & I_3 & \\ I_3 & L_3 - 2I_3 & I_3 \\ & I_3 & L_3 - 2I_3 \end{bmatrix} \\
&= \begin{bmatrix} L_3 & & \\ & L_3 & \\ & & L_3 \end{bmatrix} + \begin{bmatrix} -2I_3 & I_3 & \\ I_3 & -2I_3 & I_3 \\ & I_3 & -2I_3 \end{bmatrix} \\
&= I_3 \otimes L_3 + L_3 \otimes I_3
\end{aligned} \tag{25}
$$

Where $A \otimes B$ is the kronecker (tensor) product of A and B given by the block matrix

$$
A \otimes B = \begin{bmatrix} A_{1,1} \cdot B & \dots & A_{1,n} \cdot B \\ \vdots & \ddots & \vdots \\ A_{m,1} \cdot B & \dots & A_{m,n} \cdot B \end{bmatrix} \tag{26}
$$

Intuition would suggest, and [6] confirms, that in general, we have

$$
L_{N\times M} = I_N \otimes L_M + L_N \otimes I_M
$$

Which is implemented in the `scipy.sparse` library as `scipy.sparse.kronsum` such that $L_{N\times M} =$ `scipy.sparse.kronsum`(L_N, L_M). This generalizes to three, or arbitrarily many dimensions as $L_{N\times M\times P} =$ `scipy.sparse.kronsum`$(L_{N\times M}, L_P)$

Given an $N \times M$ grid then, we can construct equation (1) efficiently as $L_{N\times M} \cdot \phi = 0$.

1.B Arbitrary Boundary Condition

If we have a boundary condition, such as the initial condition given in Figure 1a that defines $\phi_i = b_i$ for $i \in I$ with I the set of indices of boundary points that are defined. With these boundary conditions many of the equations in the linear system $L \cdot \phi = 0$ must be reformulated. We solve this problem by modifying the matrix L and the matrix of zeros on the right hand side of the equation to end up with a new system $\hat{L}\phi = \hat{b}$.

We would like the ith row of this system to reflect the boundary condition at i for each $i \in I$. A simple way to do this is to set $\hat{L}_{i,j} = \delta_{ij}$ for each $i \in I$ and all grid indices j with δ_{ij} is the kronecker delta ($\delta_{ij} = 1$ if and only if $i = j$ and is zero otherwise). This zeros the ith row of L except a one on the diagonal. Setting $\hat{b}_i = b_i$ the ith row of the system is now our ith boundary condition.

Unfortunately, zeroing only rows means that $\hat{L}$ is no longer symmetric, which makes some of our preconditioning or factorization methods inefficient or unusable. Denote this asymmetric L by $\hat{L}_a$. We can correct the symmetry of $\hat{L}_a$ by zeroing the ith column for $i \in I$ to get $\hat{L}$. Zeroing the ith column, however, removes the influence of the ith point on adjacent points in the grid, and we must account for this by modifying $\hat{b}$.

Let $\phi^{(0)}$ be a vector with $\phi_i^{(0)} = b_i$ for $i \in I$ and $\phi_j^{(0)} = 0$ for $j \notin I$. Then $\hat{L}_a \cdot \phi^{(0)}$ is exactly the influence of the boundary conditions on their neighbors and the system so we can set $\hat{b} = -\hat{L}_a \cdot \phi^{(0)}$. This results in the final system $\hat{L}\phi = -\hat{L}_a \cdot \phi^{(0)}$ with symmetric $\hat{L}$. Furthermore, $\hat{L}$ has a sparsity pattern that is a subset of Ls and accurately incorporates the arbitrary boundary conditions. This sparsity pattern upholds the integrity of the Incomplete Cholesky and Incomplete Poisson preconditioners.

References

[1] V.A. Rakov. *Fundamentals of Lightning.* Cambridge University Press, 2016.

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

[3] T. Kim, J. Sewall, A. Sud, and M. Lin. Fast simulation of laplacian growth. *IEEE Computer Graphics and Applications*, 27(2):68–76, mar 2007.

[4] T. Kim and M. Lin. Fast animation of lightning using an adaptive mesh. *IEEE Transactions on Visualization and Computer Graphics*, 13(2):390–402, mar 2007.

[5] Timothy A. Davis and William W. Hager. Row modifications of a sparse cholesky factorization. *SIAM Journal on Matrix Analysis and Applications*, 26(3):621–639, jan 2005.

[6] J.W. Demmel. *Applied Numerical Linear Algebra.* Other Titles in Applied Mathematics. Society for Industrial and Applied Mathematics (SIAM, 3600 Market Street, Floor 6, Philadelphia, PA 19104), 1997.

[7] Costas Sideris, Mubbasir Kapadia, and Petros Faloutsos. Parallelized incomplete poisson preconditioner in cloth simulation. In *Motion in Games*, pages 389–399. Springer Berlin Heidelberg, 2011.