# Fast Animation of Lightning Using An Adaptive Mesh

Theodore Kim and Ming C. Lin
University of North Carolina at Chapel Hill

**Abstract**—We present a fast method for simulating, animating, and rendering lightning using adaptive grids. The "dielectric breakdown model" is an elegant algorithm for electrical pattern formation that we extend to enable animation of lightning. The simulation can be slow, particularly in 3D, because it involves solving a large Poisson problem. Losasso, et al recently proposed an octree data structure for simulating water and smoke, and we show that this discretization can be applied to the problem of lightning simulation as well. However, implementing the incomplete Cholesky conjugate gradient (ICCG) solver for this problem can be daunting, so we provide an extensive discussion of implementation issues. ICCG solvers can usually be accelerated using "Eisenstat's trick", but the trick cannot be directly applied to the adaptive case. Fortunately, we show that an 'almost incomplete Cholesky' factorization can be computed so that Eisenstat's trick can still be used. We then present a fast rendering method based on convolution that is competitive with Monte Carlo ray tracing, but orders of magnitude faster, and also show how to further improve the visual results using jittering.

**Index Terms**—I.3.5.i Physically based modeling

## 1 Introduction

Lightning effects are an ubiquitous visual effect in science fiction and fantasy films. From the birth of the monster in 1931's *Frankenstein* to the lightning from the Emperor's fingers in *Return of the Jedi*, electrical discharge has a long history as a dramatic tool in the visual effects industry.

Despite the popularity of this effect, there has been relatively little research into physically-based modeling of lightning. The existing research is largely empirical, essentially generating a random tree-like structure that qualitatively resembles lightning. The previous work is also limited to brief flashes of lightning, and provides no method for animating a dancing, sustained stream of electricity. Modeling the fractal geometry of electrical discharge and similar patterns has attracted much attention in physics. In this article, we adapt the *dielectric breakdown model* (DBM) for modeling lightning formation.

The DBM can be very expensive to compute, so we introduce a fast version of DBM using an adaptive mesh. Our approach is built upon a symmetric discretization of the Poisson problem, and can produce high quality results in a fraction of the running time. We show that this approach is accurate enough for graphical simulation. Its applications can extend beyond visual effects to more physically demanding applications, such as commercial flight simulation.

**Main Results:** In this article, we present the following[1]

- A physically-based approach based on the *dielectric breakdown model* for modeling electrical discharge;
- A novel technique of animating sustained electrical streams by solving a Poisson equation;

Fig. 1. **Lightning In A Bottle:** Lightning generated using our method that has been constrained to the inside of a bottle using our user parameters.

- A fast, accurate rendering method using a convolution kernel for describing light scattering in participating media;
- A parameterization that enables simple artistic control of the simulation.
- A fast adaptive mesh simulation of the DBM that drastically improves the running time by up to two orders of magnitude;
- An 'almost incomplete Cholesky' factorization that al-

1. A preliminary version of this manuscript appeared as one of the cover-image articles in [1].

lows Eisenstat's trick [2] to be used in the solver;
- A jittering method that improves the rendering results.

We also provide extensive implementation details of the adaptive mesh algorithm.

**Organization:** The rest of the paper is organized as follows. A brief survey of related work is presented in Sec. 2. In Sec. 3, we briefly summarize the physics of lightning formation. We present the original dielectric breakdown model as well as our proposed extension in Sec. 4. The adaptive mesh algorithm is presented in Sec. 5. An efficient rendering method is present in Sec. 6. User parameters are presented in Sec. 8, followed by implementation details and discussion in Sec. 9. Finally, conclusions and possible directions for future work are given in Sec. 10.

## 2 Previous Work

Reed and Wyvill present a lightning model based on the empirical observation that most lightning branches deviate by an average of 16 degrees from parent branches [3]. A set of randomly rotated line segments are generated with their angles normally distributed around 16 degrees. In subsequent work, modifications are made to this random line segment model. Glassner [4] performs a second pass on the segments to add "tortuosity", and Krus [5] replaces the normal distribution with a more easily controlled randomized binary tree.

Notably, Sosorbaram, Fujimoto, Muraoka and Chiba [6] use the dielectric breakdown model (DBM) to guide the growth of a random line segment tree with a local approximation of the potential field. But, their approach does not appear to implement full DBM, as it does not solve the Laplace equation.

Electric discharges are neither solid, liquid, or gas, but instead are the fourth phase of matter, *plasma*. It is a light source with no resolvable surface, so traditional rendering techniques are not directly applicable. To address this problem, Reed and Wyvill [3] describe a ray tracing extension for both a lightning bolt and its surrounding glow. Alternatively, [6] proposes rendering 3D textures. Dobashi, Yamamoto and Nishita [7] provide the most rigorous treatment of the problem by first presenting the associated volume rendering integral, and then presenting an efficient, approximate solution.

In electrical engineering, there are three popular models of electric discharge: gas dynamics [8], electromagnetics [9], and distributed circuits [10]. However, none of these are directly applicable to visual simulation, as they respectively approximate the electricity as a cylinder of plasma, a thin antenna, and two plates in a circuit.

## 3 The Physics of Electric Discharge

We classify the physics literature into two categories. The first deals with the physical, experimentally observed properties of lightning and related electrical patterns. A good survey of this approach is given by Rakov and Uman [11]. The second is a more qualitative approach that characterizes the geometric, fractal properties of electric discharge. A good survey of this approach is given by Vicsek [12].

### 3.1 Physical Properties

Electrical discharge occurs when a large charge difference exists between two objects. In the case of lightning, the case is usually that the bottom of a cloud has a strong negative charge and the ground possesses a relatively positive charge. Electrons possess negative charge, so the charge difference is equalized when electrons are transferred from the cloud to the ground in the form of lightning. This case is referred to as 'downward negative lightning'. While other types can exist, downward negative lightning accounts for 90 percent of all cloud-to-ground lightning. For illustrative purposes, we will show here how to simulate this most common type of lightning. But, it should be noted that we can handle the other types of lightning by trivially manipulating the charge configuration.

Lightning is actually composed of several bolts, or 'strokes' in rapid succession. The first stroke is referred to as the *stepped leader*. The subsequent strokes, called *dart leaders*, tend to follow the general path of the previous leaders, and generally do not exhibit as much branching as the stepped leader. We note that while previous work could in principle simulate this effect using attractor particles, the method we present in this article provides a natural, physically consistent method of accounting for this phenomena.

Lightning is initiated in clouds by an event known as the *initial breakdown*. During the initial breakdown, the conductivity in a small column of air jumps several orders of magnitude, effectively transforming the column from an insulator to a conductor. Charge then flows into the newly conductive air. Another breakdown then occurs somewhere along the perimeter of the newly charged air. This chain of events repeats, forming a thin, tortuous path through the air, until the charge reaches the ground.

### 3.2 Geometric Properties

The physical processes that give rise to the breakdown are still not well understood. However, a great deal of progress has been made in characterizing the geometric shape that the breakdown ultimately produces. Electric discharge has been observed to have a fractal dimension of approximately 1.7 [13]. Many disparate natural phenomena share this same fractal dimension, including ice crystals, lichen, and fracture patterns. Collectively, all the patterns that share these fractal properties are known as *Laplacian growth* phenomena.

There are three commonly known techniques for simulating Laplacian growth: Diffusion Limited Aggregation [14], the Dielectric Breakdown Model [13], and Hastings-Levitov conformal mapping [15]. All three produce qualitatively similar results. We elect to use the Dielectric Breakdown Model here because it gives the closest correspondence to the physical system being simulated and allows the addition of natural, physically intuitive user controls.

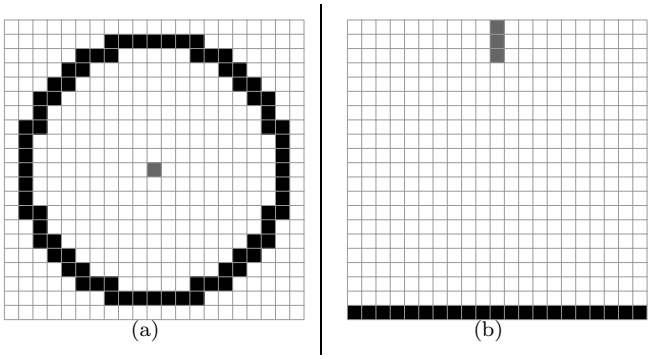

Fig. 2. Different charge configurations for simulation. Grey: $\phi = 0$; Black: $\phi = 1$ On the left is the original initial configuration, on the right is our lightning initial configuration.

## 4 The Dielectric Breakdown Model

The Dielectric Breakdown Model, or DBM, was first described by Niemeyer, Pietronero, and Wiesmann [13], and is also sometimes referred to as the $\eta$ model. We first present the model described in the original paper, and then propose a modification to simulate dart leaders and sustained electric arcs.

### 4.1 The Laplacian Growth Model

The original charge configuration from [13] is shown in Figure 2(a). Over a 2D grid, the quantity $\phi$, the electrical potential at each point, is tracked. First, a negative charge is placed at the center by setting $\phi = 0$ at the center grid cell. Then, a circle of positive charge is constructed around the center charge by setting a surrounding circle to $\phi = 1$. The potential at the remaining grid cells are then set by solving the Laplace equation (Eqn. 1) over the grid, with the center charge and the surrounding circle treated as boundary conditions.

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

The Laplace equation produces a linear system that must then be solved. There are standard ways of solving the Laplace equation and the related Poisson equation, the reader is referred to [16]. In our implementation, we solved the system using conjugate gradient with an Incomplete Cholesky preconditioner [17]. However, for high resolution simulations, the solver can become prohibitively slow, so we will present an efficient adaptive grid method in Section 5.

Once the Laplace equation has been solved, we construct a list of all the grid cells that are adjacent to a negative charge ($\phi = 0$). One of these grid cells is then randomly chosen as a growth site (i.e. the site of the next breakdown). The chosen cell is set to $\phi = 0$ and is treated as part of the boundary condition in subsequent iterations. The probability of a grid cell being chosen is weighted according to its potential. The weight function is given in Eqn 2.

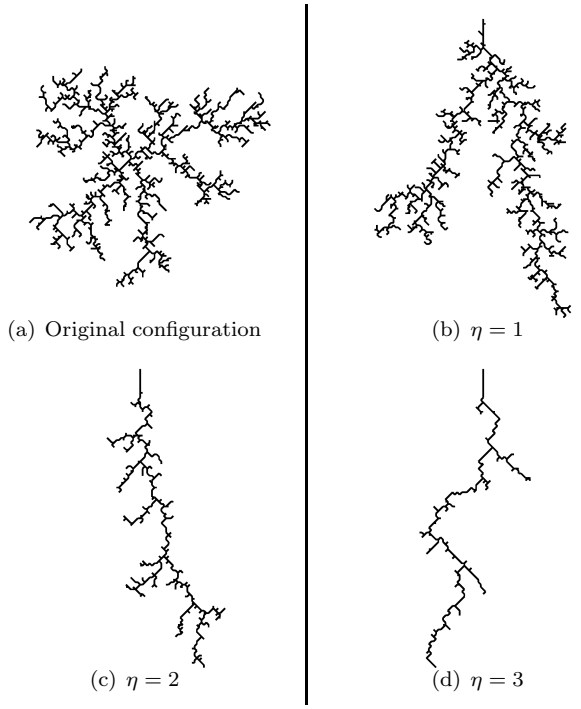

Fig. 3. Simulation results from different charge configurations. 3(a) is result of configuration from Figure 2(a). 3(b) - 3(d) are configurations from 2(b) with various $\eta$.

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

where $i$ is a cell in the list of adjacent cells, and $n$ is the total number of cells in the list. The $\eta$ term is a powerful user parameter that enables high-level control of the simulation via a single parameter value. Several different simulations with different $\eta$ values are given in Figs. 3(b)-3(d). The parameter will be discussed more in section 8.

Subsequent iterations proceed by solving the Laplace equation again over the 2D domain, and again selecting a growth site according to Eqn 2. The iterations are repeated until the user obtains the desired results. The technique generalizes trivially to three dimensions by simply solving the 3D Laplace equation.

The classic configuration produces a radial discharge, as shown in Figure 3(a). In order to produce lightning-like patterns, we instead use the initial configuration shown in Figure 2(b). We start with a small amount of negative charge at the top of the 3D domain, representing an initial branch of lightning. The bottom edge of the domain represents the ground, and is thus set to positive charge. The results of running the simulation on this initial configuration with different $\eta$ are shown in Figures 3(b) - 3(d).

### 4.2 A Poisson Growth Model

Once we have formed an initial stepped leader, we would like to have a method for generating subsequent dart leaders that follow the same general path. Since the path changes slightly with each successive dart leader, a large

number of dart leaders will produce the 'dancing' effect present in a sustained electric arc.

We hypothesize that the reason that a dart leader follows the same general path as a stepped leader is because there exists residual positive charge along the old leader channel that attracts the new dart leader. In order to simulate this behavior, we need a method of introducing residual charge into the simulation.

While DBM can simulate many different kinds of natural phenomena, we observe that for the case of electricity, the Laplace equation can be viewed as a special case of the Helmholtz equation (Eqn 3).

$$\left(\nabla^2 + \left(\frac{\omega}{c}\right)^2\right)\phi = -4\pi\rho \tag{3}$$

where $\omega$ is angular velocity, $c$ is the speed of light, and $\rho$ is charge density. The Helmholtz equation is derived directly from the Maxwell equations for electricity and magnetism, so it provides a clean connection between fractal growth and classical physics. The Laplace equation can be viewed as the case where the charge density is equal to zero and the relativistic $\left(\frac{\omega}{c}\right)^2$ term is ignored. As lightning bolts already have a linear velocity that already approaches the speed of light, the angular component will be negligible. So, if we continue to ignore the relativistic term but re-introduce the charge density term, the electromagnetic Poisson equation is obtained:

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

If we now solve this equation in place of the Laplace equation, we can produce the desired dart leader behavior. We define a second grid of values in space, $\rho$, that is initially set to zero. This essentially reduces Eqn. 4 to the Laplace equation for the initial iteration. After we generate our first bolt, we deposit charge along the leader channel by setting $\rho$ along the channel to a positive value. When generating subsequent bolts, the new $\rho$ values will automatically attract the new bolt to the old path. After each new bolt is generated, we clear the previous $\rho$ field and repopulate it with charges along the new leader channel.

Fortunately, because the Poisson and Laplace equations are very similar, the only implementation overhead required for our modified model is a minor change to the residual calculation in the conjugate gradient solver. A similar model has also been proposed in the physics literature [18] for inhomogeneous dielectric insulators independently. For computational efficiency of visual rendering, we choose to ignore inhomogeneity and treat air as a homogeneous media.

# 5 Adaptive Mesh Simulation

While the algorithm from the previous section produces visually acceptable results, the running time and memory footprint can be prohibitive, especially in 3D. In practice, we have found that in order to obtain results where grid artifacts are not distracting, at least a $256^3$ grid must be used. In order to run preconditioned conjugate gradient (PCG), 7 arrays of this size are necessary. Assuming single precision, this consumes 469 MB of memory. Increasing the resolution to $512^3$ bloats the footprint to an unacceptable 3.7 GB. Even with an efficient conjugate gradient implementation, the matrix becomes so large that simulations can take many days.

However, the lightning itself certainly does not grow to populate all $512^3$ cells, so if we can use a hierarchical data structure to place coarse grids in regions far from the lightning, it should significantly reduce the memory footprint. As the number of overall grid cells decreases, the matrix size decrases as well, and the running time of the simulation improves considerably.

Until recently, solving the Poisson problem over an adaptive grid had the caveat that it produced a non-symmetric matrix, prohibiting the use of PCG. While similar Krylov subspace methods such as GMRES and BCG could be applied to the non-symmetric case, these methods have robustness issues and can fail to converge. Recently, [19] overcame this limitation by deriving an adaptive grid discretization that is symmetric, allowing the use of the more robust PCG. We will review the discretization and provide extensive implementation details.

## 5.1 Octree Construction

We elected to use a *balanced* octree in our simulations, because it greatly simplifies the implementation of the solver. A balanced octree is an octree where, given any leaf node, the size of each neighbor differs by at most a factor of two. We enforce this property because it constrains the maximum number of neighbors of a leaf nodes to 24. As a leaf node and its neighbors define a row in the Poisson matrix, this bounds the complexity of performing an incomplete factorization on a leaf node.

The balancing algorithm is available in [20], but is simple enough to be described informally. We scan all the leaf nodes in the octree, and if a leaf is found to violate the balancing property, we split it, and add its new children to the list of leaves left to scan.

## 5.2 A Symmetric Discretization

In this subsection, we review the symmetric discretization of [19] and [21]. For simplicity, we will demonstrate the discretization in 2D.

On a regular grid, the weights given to a grid cell and its neighbors are shown in Figure 4(a). The symbol $\Delta x$ is the length of one side of the center cell. This is the traditional Laplace stencil, where the weights are obtained by Taylor expansion. In contrast, on an irregular adaptive grid (Figure 4(b)), it is unclear what the weights of the center and two smaller right cells should be.

Surprisingly, we can simply set the weights of the neighbors to one over the length of the *larger* cell. For example, in Figure 4(b), the center cell is larger than the right cells, so we weight the two right neighbors to $\frac{-1}{\Delta x}$ (Figure 4(c)). The center weight is then set to the negated sum of all its neighbor weights. Next, consider the case where one of the neighbors is larger than the center cell (Figure 4(d)).

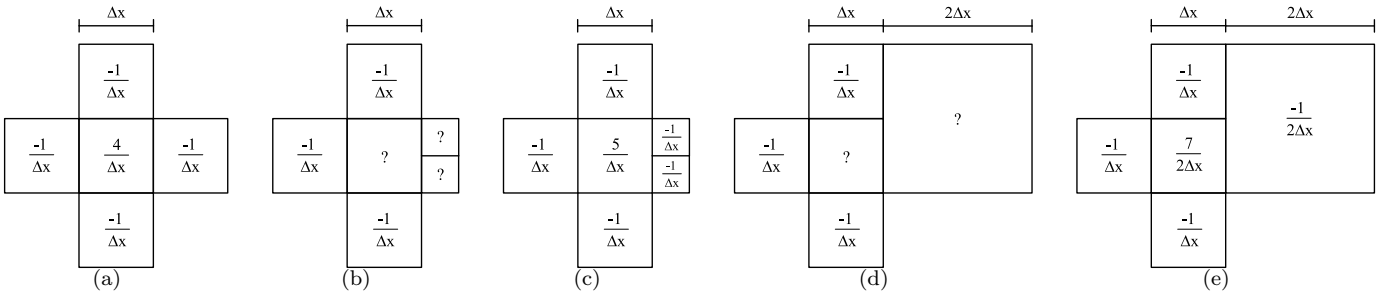


Fig. 4. Stencil weights for an adaptive mesh. (a) Regular grid stencil (b) A cell with more refined neighbors. (c) The cell with symmetric weights filled in. (d) A cell with a less refined neighbor. (e) The cell with symmetric weights filled in.

```
1  for k = 1:n
2    A(k,k) = sqrt(A(k,k))
3    for i = k + 1:n
4      if A(i,k) != 0
5        A(i,k) /= A(k,k)
6    for j = k + 1:n
7      for i = j:n
8        if A(i,j) != 0
9          A(i,j) -= A(i,k) * A(j,k)
```

Fig. 5. Generic Incomplete Cholesky for an $n \times n$ matrix $A$. (From [22])

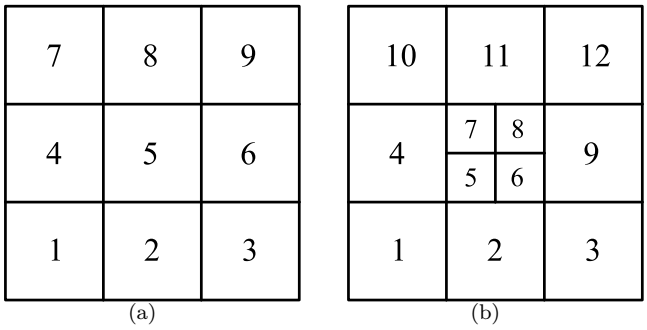


Fig. 6. (a) Natural ordering for a regular mesh. (b) Traversal ordering for adaptive mesh. Note that cell 9 is not traversed until its two neighbors in the negative $x$ direction (cells 6 and 8) are traversed.

In this case, since the right neighbor is larger, we set the weight correspondingly to $\frac{-1}{2\Delta x}$. Again, the center weight is then set to the negated sum of all its neighbor weights (Figure 4(e)). The 3D case is a straightforward extension; we compute weights in the positive and negative $z$ directions in the same way, and again set the center cell to the negated neighbor sum.

According to [19], other choices for the weights exist in addition to the maximum length. The minimum length, or a linear combination of the two lengths could be used as well. All choices lead to symmetric matrices however, and no choice appears to offer drastically better performance. So, for simplicity, we used the maximum length.

## 5.3 Incomplete Factorization

In order for conjugate gradient to converge quickly, a preconditioner must be used. One of the most popular preconditioners is the zero-fill incomplete Cholesky factorization [22]. Incomplete Cholesky is usually stated in general matrix terms (Figure 5). However, it is difficult to translate this generic algorithm to cell weight relationships over the adaptive grid. A brute force implementation of this algorithm using a generic sparse matrix library would be inefficient, as it would be dominated by unnecessary pointer dereferences. An efficient implementation would instead exploit the structure of the matrix, but this requires a more intuitive understanding of when and where zeros appear during the factorization. In order to save the readers the tedious and time-consuming process of solving many brute force examples to gain this intuition, we will explicitly describe the relationship here.

### 5.3.1 *Traversal Order*

In Line 1 of Figure 5, we iterate over the columns of the matrix. Each grid cell and its weights define a column of the matrix. For example, the stencils shown in Figs. 4(a), 4(c), and 4(e) each define a column. For the case of regular mesh, there is an obvious order in which we could traverse the mesh points (Figure 6(a)). This is known as the 'natural ordering'. Alternately we could traverse it in the reverse order, or perhaps in column-major order. The point is, there is an obvious 'scanline' order in which we can traverse the cells. For the case of an adaptive mesh, there is no equally obvious order.

There are definitely incorrect orders in which to traverse the cells. In Line 9 of Incomplete Cholesky, while factoring column $k$, entries in columns $k+1$ to $n$ of $A$ are altered as well. Conversely, columns 1 to $k-1$ remain unaltered. Therefore, before we factor column $k$, we must be sure that all the columns in $A$ that are going to change values in column $k$ have already completed their changes. Otherwise, we will obtain an invalid factorization.

There are many ways to achieve a valid order, of which we will describe one. In 2D, we only factor a grid cell if all its neighbors in the negative $x$ and $y$ direction have already been factored. In 3D, all the neighbors in the negative $z$ direction must have been factored as well. A 2D example is shown in Figure 6(b). It is easily verified that this ordering meets the criteria of the previous paragraph.

```
1  For k = 1:n
2    Weight(k,k) = sqrt(Weight(k,k))
3
4    For i = each neighbor of k
5      If Order(k) < Order(i)
6        Weight(k,i) /= Weight(k,k)
7
8    For i = each neighbor of k
9      If Order(k) < Order(i))
10       Weight(i,i) -= Weight(k,i) * Weight(k,i)
12
13   For i = each neighbor of k
14     If Order(k) < Order(i)
15       For j = each neighbor of k
16         If (Order(k) < Order(j))
17           If (i and j are neighbors)
18             Weight(i,j) -= Weight(k,i) * Weight(k,j)
```

Fig. 7. Incomplete Cholesky in terms of an adaptive mesh.

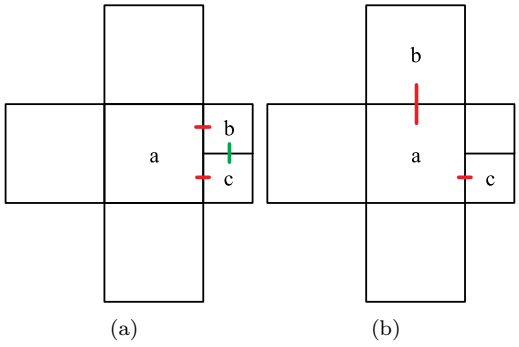


Fig. 8. (a) Example of lines 13-18 of Incomplete Cholesky. Cells $b$ and $c$ are neighbors, so the product of the weights with the red lines must be subtracted from the weight denoted in green. (b) Example of lines 19-21 of Modified Incomplete Cholesky. Cells $b$ and $c$ are *not* neighbors, so the product of the two red weights is subtracted from the center weights of $b$ and $c$.

### 5.3.2 Incomplete Cholesky

For the remainder of the section, we will use the following notation. $Weight(a, b)$ is the weight between cells $a$ and $b$. $Weight(a, a)$ represents the center weight for cell $a$. Note that since the matrix is symmetric, $Weight(a, b) = Weight(b, a)$. $Order(a)$ is the order in which the cell $a$ is traversed. If $Order(a) < Order(b)$, then $b$ is traversed after $a$.

At each cell $a$, we only store the weights for neighbors $n$ where $Order(a) < Order(n)$. (Effectively, the weights in the positive $x$, $y$ and $z$ directions.) We do this because the matrix is symmetric, so storing all weights at all the cells would be redundant. Incomplete Cholesky in terms of this adaptive mesh notation is given in Figure 7.

Lines 1-6 of Figure 7 correspond fairly directly with Lines 1-5 of Figure 5. The correspondance between Lines 8-17 of Figure 7 and Lines 6-9 of Figures 5 is less clear. In Lines 4-6, we assigned new weights to the neighbors in the positive $x$, $y$ and $z$ directions. In Lines 8-10, we subtract the square of these new weights from the center weights of same neighbor cells.

Lines 13-17 are perhaps the most difficult to interpret,

```
1  for k = 1:n
2    A(k,k) = sqrt(A(k,k) + delta * A(k,k))
3    for i = k + 1:n
4      if A(i,k) != 0
5        A(i,k) /= A(k,k)
6    for j = k + 1:n
7      for i = j:n
8        if A(i,j) != 0
9          A(i,j) -= A(i,k) * A(j,k)
10       else
11         A(i,i) -= alpha * A(i,k) A(j,k)
```

Fig. 9. Generic Modified Incomplete Cholesky for an $n \times n$ matrix $A$. The symbols $alpha$ and $delta$ are relaxation parameters that the user sets.

```
1  For k = 1:n
2    Weight(k,k) = sqrt(Weight(k,k) + delta * Weight(k,k))
3
4    For i = each neighbor of k
5      If Order(k) < Order(i)
6        Weight(k,i) /= Weight(k,k)
7
8    For i = each neighbor of k
9      If Order(k) < Order(i))
10       Weight(i,i) -= Weight(k,i) * Weight(k,i)
12
13   For i = each neighbor of k
14     If Order(k) < Order(i)
15       For j = each neighbor of k
16         If (Order(k) < Order(j))
17           If (i and j are neighbors)
18             Weight(i,j) -= Weight(k,i) * Weight(k,j)
19           Else If (i != j)
20             Weight(i,i) -= Weight(k,i) * Weight(k,j)
21             Weight(j,j) -= Weight(k,i) * Weight(k,j)
```

Fig. 10. Modified Incomplete Cholesky in terms of an adaptive mesh.

so we will provide an example here. In Figure 8(a), we have two cells, $b$ and $c$, that are adjacent to cell $a$, and also adjacent to each other. In this case, if $Order(a) < Order(b)$ and $Order(a) < Order(c)$, then we subtract the product of the two weights shown in red from the weight shown in green. This must be done for all cases where $a$ has two neighbors that are traversed after $a$, and the neighbors are neighbors of each other as well. On a balanced octree, there are 16 cases where this occurs. Determining these 16 cases is left as an exercise to the reader.

### 5.3.3 Modified Incomplete Cholesky

In [19], Losasso et al. report using a preconditioner "based on an incomplete LU Cholesky factorization that we modify to ensure that the sum of LU is equal to the row sum of original matrix." This corresponds to the Modified Incomplete Cholesky preconditioner [23]. However, the version in [23] is often combined with the version in [24]. In practice, we have found this combination of techniques gives faster results than Incomplete Cholesky alone, so we will describe the factorization here. The generic Modified Incomplete Cholesky factorization, using both the techniques from [23] and [24] is shown in Figure 9.

Modified Incomplete Cholesky in terms of an adaptive grid is shown in Figure 10. Two significant modifications have been made from the Incomplete Cholesky case. First,

a *delta* parameter has been added to Line 2. Second, Lines 19-21 are new. We illustrate the significance of these new lines in Figure 8(b). The factorization is the same as before, except an *else* has been added to the innermost loop. If $a$ has two neighbors $b$ and $c$, and both are traversed after $a$, but $b$ and $c$ are *not* neighbors, the product of the two weights in red must be subtracted from the *center* weights of $b$ and $c$. Again, determining the explicit cases is left as an exercise to the reader.

### 5.4 Eisenstat's Trick

Applying Incomplete Cholesky (IC) adds an additional forward and backward substitution step to each conjugate gradient iteration. For large matrices, this additional overhead is negligible compared to the computation time saved by the reduced number of iterations. However, for modest sized matrices such as those produced by a quadtree lightning simulation, it can have a noticeable impact on the running time.

This performance impact becomes particularly apparent if we try to implement a SIMD version of ICCG. Most of the steps of CG map trivially to SSE intrinics, with the exception of a matrix-vector produce and the already mentioned forward and backward solves. These two operations therefore become the main performance bottlenecks.

Eisenstat [2] described a method of eliminating the matrix-vector multiply, thus eliminating one of the bottlenecks. Unfortunately, "Eisenstat's trick" as it is described in the original paper only applies to preconditioners with a very specific structure. The IC preconditioner for a regular grid fits this structure, but the adaptive version described in the previous section does not. Fortunately, we have found that it is possible to construct an approximation to the IC preconditioner such that Eisenstat's trick can still be applied.

Assume we are solving the matrix $A$, which can be decomposed to $A = L+D+L$, where $L$ is strictly lower triangular, and $D$ is a diagonal matrix. Eisenstat's trick can be applied if the preconditioner $M$ can be stated in terms of a diagonal matrix $\hat{D}$ such that $M = (\hat{D}+L)\hat{D}^{-1}(\hat{D}+L)^T$. While the IC factorization over a regular grid can in fact be decomposed into this form, the factorization over the adaptive grid cannot. Instead, we obtain a factorization $M = (\hat{D}+\hat{L})\hat{D}^{-1}(\hat{D}+\hat{L})^T$, where $\hat{L} \neq L$.

The origin of this modified $\hat{L}$ can be traced back to lines 13-18 of Fig. 7. In the regular grid case, lines 6-9 of the generic IC factorization translate to modifications along the diagonal. However, as shown by the green line in Figure 8(a), the adaptive case modifies entries that are off-diagonal as well. These off-diagonal modifications 'ruin' the possibility of the $\hat{L}$ in the final factorization matching the original $L$.

However, nothing prevents us from going ahead and computing the $\hat{D}$ matrix for the adaptive case, even if the final result produces $\hat{L} \neq L$. The algorithm to compute this $\hat{D}$ is given in Fig 11. If we discard the computed $\hat{L}$ after $\hat{D}$ has been constructed and replace it with the original $L$, we can still apply Eisenstat's trick. The preconditioner

```
1  For k = 1:n
2    For i = each neighbor of k
3      If Order(k) < Order(i))
4        Weight(i,i) -= Weight(k,i) * Weight(k,i) /
5                       Weight(k,k)
6
7    For i = each neighbor of k
8      If Order(k) < Order(i)
9        For j = each neighbor of k
10         If (Order(k) < Order(j))
11           If (i and j are neighbors)
12             Weight(i,j) -= Weight(k,i) * Weight(k,j) /
13                            Weight(k,k)
```

Fig. 11. Computing the $\hat{D}$ matrix for Eisenstat's trick over an adaptive mesh.

that is being applied will not be the exact IC preconditioner, but it will instead be an 'almost' IC preconditioner that lacks the off-diagonal modifications to $L$ depicted in Fig. 8(a).

We have found that this 'almost' IC preconditioner works well in practice. In modest resolution (512 x 512) quadtree simulations, the number of iterations required to converge to working precision remains identical to those required by the original IC preconditioner. In larger octree simulations, the preconditioner only requires a handful of additional iterations. We have found that even the extra computational cost of these additional iterations does not outweigh the benefit of having eliminated the matrix-vector product from the inner CG loop. While it is possible that for a large enough matrix, the 'almost' IC preconditioner will require enough extra iterations that its performance benefit is invalidated, we did not encounter such a case during the course of our simulations.

### 5.5 Discussion

Using this discretization, we have experienced a speedups of over two orders of magnitude. We ran two comparisons for 2000 timesteps using the 3D equivalent of Fig. 2(b), one on a $128^3$ mesh and the other on a $256^3$ mesh. The adaptive $128^3$ simulation ran 22 times faster, while the adaptive $256^3$ simulation ran 150 times faster. Log plots of the performance are shown in Figure 12.

Despite these obvious gains, the adaptive mesh is still not a simulation panacea. If large objects are inserted as boundaries in the grid, the memory footprint of the simulation can grow quickly. This is because unlike in the regular grid case, neighbor information per grid cell is not implicitly known from the grid dimensions, and must be explictly constructed. A cell can have up to 24 neighbors (4 neighbors at each of the 6 cube faces), increasing the memory footprint of a single grid cell by a considerable 96 bytes (A 4 byte pointer for each of the 24 neighbors). Instead of explicitly storing the neighbors, neighbor queries could be performed on the octree as necessary, but this is a tradeoff, as it significantly increases the overall running time. Explicit storage of the stencil and the incomplete factorizations raise the same tradeoff issues.

Finally, for the parameters *alpha* and *delta* in Modified

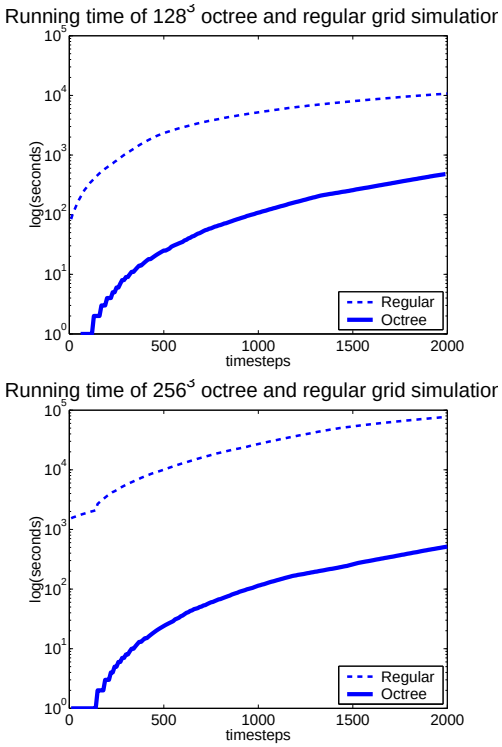


Fig. 12. Logarithmic timing plots of regular grid solver and octree solver. Both simulations are shown for 2000 timesteps. Over a $128^3$ grid, the octree solver ran 22 times faster, and over a $256^3$ grid, the octree solver ran 150 times faster.

Incomplete Cholesky, we ran a wide range of tests to determine good parameter values. We found values of *alpha* approaching one ($alpha \approx 0.95$) and *delta* values near zero ($delta \approx 0.05$) yielded good results. These findings are consistent with other studies of these parameters, such as those of [25].

## 6 Rendering

For the rendering of electricity, we borrow the method proposed by Narasimhan and Nayar [26] that produces spectacular results. In the paper, analytical models are obtained that reduce the rendering of certain types of participating media to a 2D convolution. The results are competitive with expensive Monte Carlo techniques such as photon mapping, but run in seconds instead of hours. We will first summarize the pertinent formulae from [26], then describe how we use it to generate a convolution kernel, and finally show how we render electricity.

### 6.1 Atmospheric Point Spread Function

The convolution kernel produced by the method of [26] is called an Atmospheric Point Spread Function, hereon referred to as an APSF. The APSF is a series expansion of the Henyey-Greenstein phase function, a popular function for describing the scattering of light in participating media. The basis functions used are Legendre polynomials, whose series form are shown in Eqn. 5.

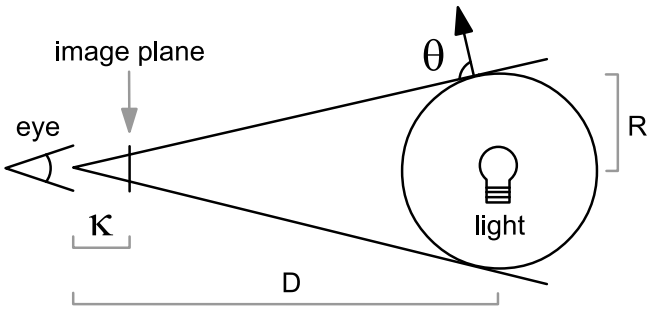


Fig. 13. Pinhole camera geometry for generating APSF kernel

$$L_i(x) = \frac{((2i-1)\cdot x \cdot L_{i-1}(x) - ((i-1)\cdot L_{i-2}(x))}{i} \quad (5)$$

In order the evaluate the series, the following base cases are also necessary: $L_0(x) = 1$, $L_1(x) = x$. The full APSF, $I(T, \mu)$, is then given in Eqn. 6.

$$I(T,\mu) = \sum_{m=0}^{\infty} (g_m(T) + g_{m+1}(T))\, L_m(\mu) \quad (6)$$

where

$$g_m(T) = e^{-\beta_m T - \alpha_m \log T} \quad (7)$$

$$\alpha_m = m + 1 \quad (8)$$

$$\beta_m = \frac{2m+1}{m}(1 - q^{m-1}). \quad (9)$$

A base case is necessary: $g_0(T) = 0$. The variable $q$ is the scattering parameter from the Henyey-Greenstein phase function. Increasing $q$ from 0 to 1 increases the density of the medium, and can be thought of as transitioning the weather from clear skies to rain. The *optical thickness*, $T$, is equal to $R\sigma$, where $R$ is the radial distance from the viewer, and $\sigma$ is the extinction coefficient of air. Finally $\mu$ is the cosine of the radial direction $\theta$ from the source.

### 6.2 Generating a Convolution Kernel

The APSF is a three dimensional function that describes how much light is reaching any point in space around a point light source. If we can determine how a single point light spreads out on the image plane, we can then use this point spread function as a convolution kernel to render a light source of arbitrary shape.

Assume we want to generate an $n \times n$ resolution convolution kernel of physical size $M \times M$. We sample the APSF according to the geometry in Figure 13. In this figure, we assume a pinhole camera model where $\kappa = 0.025$ meters, about the width of an eyeball. We also assume the lightning stroke is two kilometer away: $D = 2000$. We treat $R$ as a user parameter that allows control over the width of the 'glow' around the stroke.

In order to compute the value $I(T, \mu)$ at each point on the kernel, we need to determine a value $\mu$ at each sample. If we assume the point light source projects onto the center

of the kernel, the $\mu$ value at kernel sample $(x, y)$ follows by trigonometry (Eqns. 10 - 12).

$$\nu_{x,y} = \sqrt{\frac{(x-\frac{n}{2})\times M}{n} + \frac{(y-\frac{n}{2})\times M}{n}} \quad (10)$$

$$\omega_{x,y} = \frac{\kappa^2 D - \kappa\sqrt{-\nu_{x,y}^2 D^2 + \kappa^2 R^2 + \nu_{x,y}^2 R^2}}{\kappa^2 + \nu_{x,y}^2} \quad (11)$$

$$\mu_{x,y} = \pi - \tan^{-1}\frac{\kappa}{\nu_{x,y}} - \sin^{-1}\frac{D-\omega_{x,y}}{R} \quad (12)$$

If $\frac{\nu_{x,y}}{\kappa} > \frac{R}{D}$, then we are outside the desired width of the glow, and the kernel value should be set to zero. The APSF only drops off to zero at infinity, so in practice, the smallest non-zero value of the kernel must be subtracted from the other non-zero values of the kernel to prevent the silhouette of the kernel from appearing in the final image.

We are making a simplifying assumption here that all parts of the lightning bolt are exactly two kilometers away. While this is not strictly true, unless the bolt spans a very large physical domain, we believe it is a reasonable approximation. In the extreme case of a lightning bolt originating from far away and hitting the camera, this assumption breaks down, and several different depth-dependent kernels must be computed. However, even in this extreme case, the time required to generate several different kernels is still orders of magnitude less than using a Monte Carlo renderer.

### 6.3 Rendering Electricity

Even for large electric discharges like lightning, the plasma channel is only several centimeters in diameter [11]. We hypothesize that humans perceive that the stroke is thicker because the the brighter portions of glow exceed the range perceptible by the human visual system, so they bleach together into what looks like a thicker bolt.

With this hypothesis in mind, we model the plasma channel as a series of thin line segments. We then apply the APSF kernel to a 2D rendering of these line segments to simulate the glow. If the brightness of the plasma channel is set correctly, the APSF should produce luminance values that exceed the range of the display device, creating the expected thick bolt. In this way, we can remain physically consistent while avoiding the need for a complex geometric representation of plasma.

We proceed in three stages. First, we construct a graph from the simulation. We then assign different luminance values to each graph edge, as some parts of a lightning stroke are brighter than others. Finally, we render the graph edges as line segments and apply the APSF.

#### *6.3.1 Constructing the Graph*

We observe that the construction of the lightning stroke can also be seen as the construction of a directed tree. The root of the tree is represented by the initial discharge from the beginning of the simulation. When a grid cell is added to the lightning stroke, we create a corresponding graph node, and then search the cell neighbors for one that is already on the stroke. Such a neighbor must exist, as it is a necessary condition for the grid cell to have been selected as a growth site. This neighbor is then set as the parent node, and the newly added grid cell is recorded as the child. When a grid cell adjacent to the ground is added, we halt the simulation. In nature, growth would end at this point because the charge now has a direct conduit to the ground.

#### *6.3.2 Assigning Wattage*

With our tree, we can now assign a separate luminance value to each line segment. We divide the line segments into three classes: the main channel, secondary channels, and side channels. The majority of the charge flows through the main channel, so it should be brightest. By inspection of photographs, it is clear that there are dimmer but distinct secondary channels in most strokes, and branching off from the secondary channels, barely perceptible side channels. Locating the main channel is straightforward. The node corresponding to the grid cell that hit the ground, along with all its ancestors, constitute the main channel.

Locating the secondary and side channels is more involved. Every node adjacent to the main channel that is not on the main channel forms the root to a new tree. Within each such tree, the charge selects a single preferred path that becomes the bright secondary channel. It is unclear how this path is selected; perhaps the path that had the largest potential differences during the breakdown process is selected. For aesthetic effect, we set the path with the greatest number of nodes as the secondary channel. Off of this longest secondary channel, we also add other 'long' paths according to a user-defined cutoff. This maximizes the length of the dramatic, snaking tendrils that surround the central channel. All the remaining edges are now considered to be side channels.

We must now assign a wattage to each edge. While there exists some data on the wattage of the main channel (Between $1.3 \times 10^6$ Watts / m and $3.9 \times 10^5$ Watts / m according to [11]), we have been unable to find data on the wattage of secondary or side channels. We have attempted to estimate the wattages by deconvolving photographs of lightning, but this method requires a high dynamic range image of lightning that can resolve the bleached portion of the stroke, as well as the APSF values corresponding to the scene. We used heuristic values that brought us into close qualitative agreement with photographs.

We rendered the line segments and convolved them with the APSF settings given in Table I. We do not set the main channel to the wattage given by [11], because in the absence of tone mapping, this would bleach the entire scene. The application of tone mapping for rendering lightning will be investigated in our future work.

## 7 Jittering

The most noticable rendering artifact is grid regularities that appear due to insufficient resolution in the simulations. Ideally, a grid cell in the simulation would correspond to one pixel in the final image. In practice, this sort of resolution is impractical. However, the appearance

of these artifacts can be greatly reduced by jittering the simulation results.

When constructing the graph, we usually assume that the lightning stroke passed directly through the center of the grid cell. However, as the grid cell is homogeneous, it is fair to say that the lightning may have passed through any portion of the cell. Given a grid cell to add to the graph, instead of selecting the center of the cell every time, we instead select a random point on the interior of the cell.

Using this simple jittering technique, the appearance of grid regularities can be reduced for a very small computational cost. A comparison of lightning strokes with and without jittering are shown in the right half of Figure 16. The grid artifacts are especially visible in the secondary channels of the lower right image, but jittering greatly reduces the prevalence of these artifacts.

## 8 User Controls

Our modified DBM permits user control through four parameters: an $\eta$ variable to control the 'branchiness' of the stream, a charge density field $\rho$ to control the path of the stream, a boundary condition to repel the stream, and an overall charge configuration to control where the stroke begins and ends.

The effect of the $\eta$ variable in Eqn. 2 can be seen in Figure 3(b) - 3(d). At $\eta = 1$, dense branching is observed. As $\eta$ increases, the density of the branching decreases. Hastings observes that at $\eta = 4$, the stream transitions into a non-fractal, one-dimensional curve [27]. So, the domain of the $\eta$ parameter is effectively in the range of $(1, 4)$. A physical interpretation of the $\eta$ is not entirely clear, it can perhaps be viewed as the ambient resistivity of the air.

As $\rho$ is a 2D field representing the image plane, the user can 'paint' into it any desired charge distribution. The lightning stroke will then be attracted to this painted path as described in Section 4.2.

In addition to attracting the electric arc, the user may want to repel the arc from certain regions. For instance, there may be an obstacle in the scene that the user does not want the arc to intersect. This effect can be achieved by setting the interior of the obstacle to $\phi = 0$. This sets the charge of the object to the same charge as the arc, causing the obstacle to repel the arc. However, we must then be careful in our implementation not to add grid cells adjacent to the obstacle to the list of candidate growth sites in Eqn. 2. We demonstrate an extreme example of this control in Fig. 14. The repulsor texture is shown in the lower right, with green denoting repulsors and black denoting valid growth sites.

| Figure | n | M | q | m | T | R |
|---|---|---|---|---|---|---|
| 17, 16 | 256 | 1.0 | 0.99 | 200 | 1.001 | 200 |
| 15 | 64 | 1.0 | 0.9 | 200 | 1.1 | 100 |

TABLE I

*APSF settings used:* $m$ CORRESPONDS TO THE NUMBER OF TERMS USED IN THE LEGENDRE SERIES.

Finally, we have only shown two charge configurations: the circle in Figure 2(a), and the lightning configuration in Figure 2(b). However, arbitrary charge configurations also produce electric arcs. The arc can begin from any arbitrarily shaped negative region, and terminate at a positive object. In this way, it is possible to construct an arc between any two objects in an arbitrary scene.

## 9 Implementation and Results

We have implemented our algorithms in C++ and OpenGL. We ran simulations for several scenes on a 2.66 Ghz Xeon processor. Unless otherwise noted, all simulations were performed on a $512 \times 512$ grid with $\eta = 1$ and $\rho = \frac{1000}{4\pi}$ along the main channel. The renderings were performed in POV-Ray, and then convolved and composited using ImageMagick.

Note that when implementing Eqn. 5, recursively evaluating the series is an exponential time operation. However, evaluating from the bottom up, (ie in the order $L_0(x), L_1(x), L_2(x)$ ...) is a dynamic programming solution that can be done in linear time. Using this method is more efficient. Also, as the convolution kernel in subsection 6.2 is isotropic, it can be performed quickly with two $n \times 1$ filters instead of one $n \times n$ filter.

In Figure 17, we demonstrate how the user can repel the bolt from arbitrary objects. The lightning must start from the top of the Cornell Box and find a path to the floor, while avoiding the two beams in the center. In Figure 17, we demonstrate how the user can attract the bolt to an arbitrary object. The magenta electrode in the center is set to a negative charge, and blue ball is set to a positive charge. As the blue ball moves, the electric arc follows.

### 9.1 Comparison to Maya Lightning

We compared our results to those of the most widely available alternative, the lightning tools in Maya. We compare to both the Maya particle and paint lightning effects.

In Figure 15, we both validate our results by comparing our renderings with a photograph, and compare our results to the Maya paint effect. The scene was simulated on a $256^3$ grid. Most likely, the paint effect uses something similar to [3]. First, our example displays a degree of branching that is much closer to that of the photograph. We tried adjusting the Maya simulation to create more branches, but the current example is the best we were able to achieve. Second, in our result, the brightness of the glow from the side branches varies, whereas the glow in the Maya example is incorrectly dim and uniform. They do not appear as tortuous as those in our results and the photo.

In Figure 16, we animate a dancing electric arc between two electrodes. We constructed a similar scene in Maya and compared our results to the particle effect. While the renderers in Maya and POV-Ray differ, comparing the high-level features is still useful. The Maya particle element generates a base mesh that is blurred by a gaussian kernel. However, due to the coarseness of the base mesh, it is difficult to attain the correct blur. Too little, and

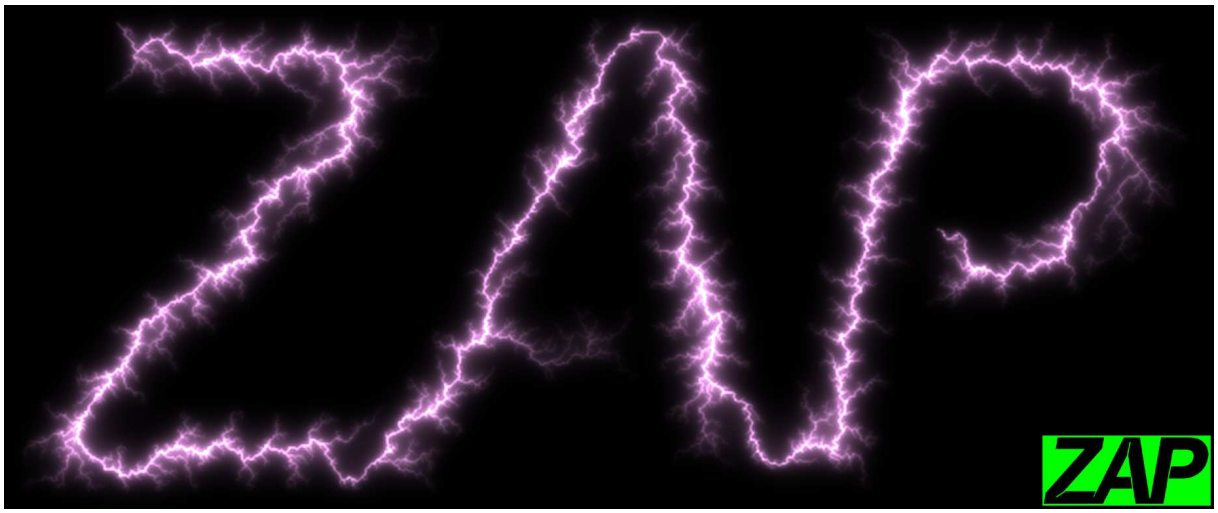


Fig. 14. **User controls example:** We generated lightning in the shape of the word ZAP. Aside from a simple input texture (inset, lower right) the simulation proceeded with no user intervention. In the input texture, green denotes a repulsor. To our knowledge, no previous method can generate such a detailed example with similar ease.

Fig. 15. Validation **Left:** Results using our algorithm **Center:** Photograph **Right:** Results from Maya Lightning. Note that the Maya result has smaller side branches that lack the glow and tapering brightness of the photo.

the base mesh is visible. Too much, and all the details are blurred away. Side branches are not handled at all. By contrast, our results handles side branching, and the APSF effectively blurs the base mesh away while preserving the visual tortuosity.

## 10 Conclusion and Future Work

We have presented a physically based algorithm for the simulation, animation, and rendering of sustained electric arcs. We believe that our approach is the most rigorous, physically consistent method available up to date. However, there are several areas for refinement.

While the adaptive grid solver can greatly decrease the memory footprint and running time of the simulation, we still encounter problems in the presence of complex boundaries. In this case, the adaptive solver can perform on par with or slightly worse than the regular grid case. Alternate particle-based methods might be able to alleviate some of these problems.

While our rendering method is physically consistent, it would be more realistic to use some sort of tone mapping operator to bring the luminance values back into the range of the display device. No operator was used here because we were unsure which would be appropriate. In the tone mapping literature, a 'bright' object is usually daylight or a lightbulb, so it is unclear if some of these methods would break down in the presence of luminance values many orders of magnitude brighter.

While the use of the convolution kernel generates impressive results, there are still some unresolved issues. It assumes the scattering medium is homogeneous, so it does not explicitly handle the effects of either internal obstacles

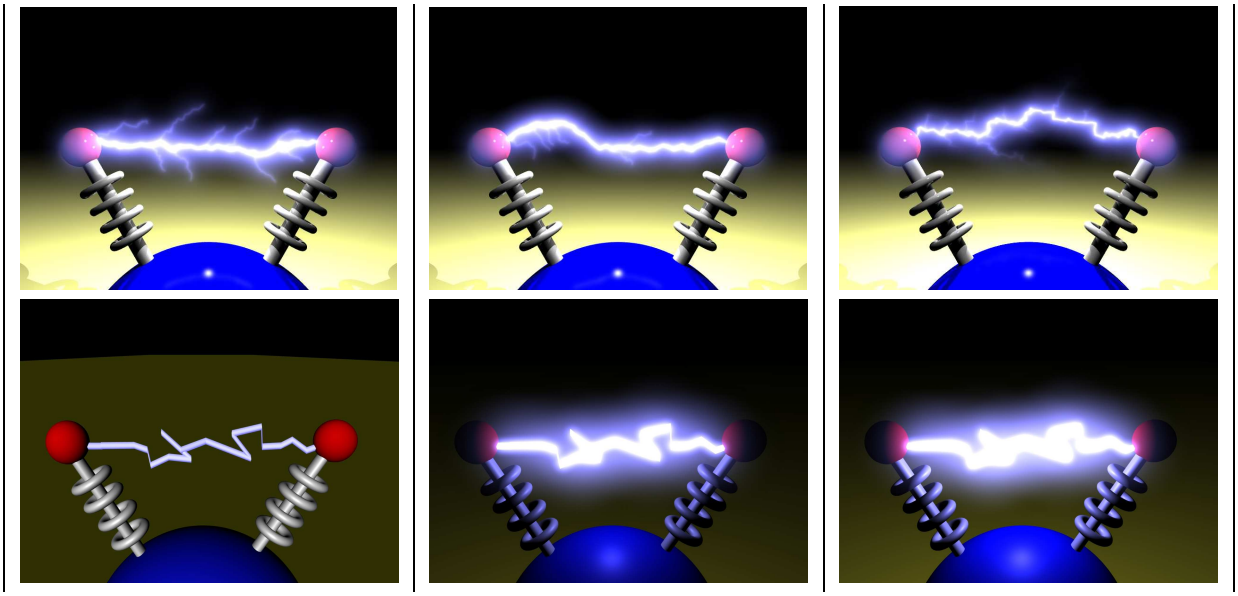

Fig. 16. **Electric arc leaping between two electrodes:** The images on the top row use our algorithm, and the images on the bottom are from Maya. **Top row:** The rightmost was generated with an octree. Note the qualitative similarity to the regular grid results. **Bottom row left to right:** The base mesh from Maya particle lightning. The mesh with low glow added; note that jaggies from the base mesh are visible. The mesh with high glow added; the base mesh has blurred away. Maya particle lightning does not account for side branches.

or clouds. A scene requiring a volume caustic still needs a Monte Carlo renderer. The approach described in [7] appears to be the best solution for a scene containing clouds. While an analytical solution may also be possible for these cases, one has not yet been found.

Finally, we have only presented one type of Laplacian growth: electric arcs. The Laplacian growth encompasses many disparate phenomena, including ice formation, material fracture, lichen growth, tree growth, liquid surface tension, vasculature patterns, river formation, and even urban sprawl. Modeling of Laplacian growth is well worth exploring for visual simulation of natural phenomena.

## References


[1] T. Kim and M. Lin, "Physically based modeling and rendering of lightning," *Proc. of Pacific Graphics 2004*, pp. 267–275, 2004.

[2] S. Eisenstat, "Efficient implementation of a class of preconditioned conjugate gradient methods," *SIAM Journal on Scientific Computing*, vol. 2, pp. 1–4, 1980.

[3] T. Reed and B. Wyvill, "Visual simulation of lightning," *Proc. of SIGGRAPH*, pp. 359–364, 1994.

[4] A. Glassner, "The digital ceraunoscope: Synthetic thunder and lightning," Microsoft Research, Tech. Rep. MSR-TR-99-17, 1999.

[5] P. Kruszewski, "A probabilistic technique for the synthetic imagery of lightning," *Computers and Graphics*, pp. 287–293, 1999.

[6] B. Sosorbaram, T. Fujimoto, K. Muraoka, and N. Chiba, "Visual simulation of lightning taking into account cloud growth," *Computer Graphics International*, pp. 89–98, 2001.

[7] Y. Dobashi, T. Yamamoto, and T. Nishita, "Efficient rendering of lightning tacking into account scattering effects due to clouds and atmospheric particles," *Proc. of Pacific Graphics*, pp. 390–400, 2001.

[8] E. Dubovoy, M. Mikhailov, A. Ogonkov, and V. Pryazhinsky, "Measurement and numerical modeling of radio sounding reflection from a lightning channel," *J. Geophys. Res.*, vol. 100, pp. 1497–1502, 1995.

[9] Y. Baba and M. Ishii, "Numerical electromagnetic field analysis of lightning current in tall structures," *IEEE Trans. Pow. Rel.*, vol. 16, pp. 324–328, 2001.

[10] C. Baum and L. Baker, "Analytic return-stroke transmission-line model," *Electromagnetics*, vol. 7, pp. 205–228, 1987.

[11] V. Rakov and M. Uman, *Lightning: Physics and Effects*. Cambridge University Press, 2003.

[12] T. Vicsek, *Fractal Growth Phenomena*. World Scientific, 1992.

[13] L. Niemeyer, L. Pietronero, and H. J. Wiesmann, "Fractal dimension of dielectric breakdown," *Physical Review Letters*, vol. 52, pp. 1033–1036, 1984.

[14] T. Witten and L. Sander, "Diffusion-limited aggregation, a kinetic critical phenomenon," *Physical Review Letters*, vol. 47, no. 19, pp. pp. 1400–1403, 1981.

[15] M. Hastings and L. Levitov, "Laplacian growth as one-dimensional turbulence," *Physica D*, vol. 116, pp. 244–252, 1998.

[16] J. Demmel, *Applied Numerical Linear Algebra*. SIAM, 1997.

[17] J. R. Shewchuk, "An introduction to the conjugate gradient method without the agonizing pain," Carnegie Mellon University, Tech. Rep., 1994.

[18] M. Noskov, V. Kukhta, and V. Lopatin, "Simulation of the electrical discharge development in inhomogeneous insulators," *Journal of Physics D*, vol. 28, pp. 1187–1194, 1995.

[19] F. Losasso, F. Gibou, and R. Fedkiw, "Simulating water and smoke with an octree data structure," *Proc. of SIGGRAPH*, pp. 457–462, 2004.

[20] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, *Computational Geometry: Algorithms and Applications*. Springer, 2000.

[21] F. Losasso, F. Gibou, and R. Fedkiw, "Spatially adaptive techniques for level set methods and incompressible flow," *Computers and Fluids*, (in press).

[22] G. Golub and C. V. Loan, *Matrix Computations*. Johns Hopkins University Press, 1996.

[23] I. Gustafsson, "A class of first order factorization methods," *BIT*, vol. 18, pp. 142–156, 1978.

[24] O. Axelsson and G. Lindskog, "On the eigenvalue distribution of a class of preconditioning methods," *Numerische Mathematik*, vol. 48, pp. 479–498, 1986.

[25] C. Ashcraft and R. Grimes, "On vectorizing incomplete factorization and ssor preconditioners," *SIAM Journal on Scientific Computing*, vol. 9, pp. 122–151, 1988.

[26] S. Narasimhan and S. Nayar, "Shedding light on the weather," *Proceedings of IEEE CVPR*, pp. 665–672, 2003.

[27] M. Hastings, "Fractal to nonfractal phase transition in the dielectric breakdown model," *Physical Review Letters*, vol. 87, no. 17, 2001.

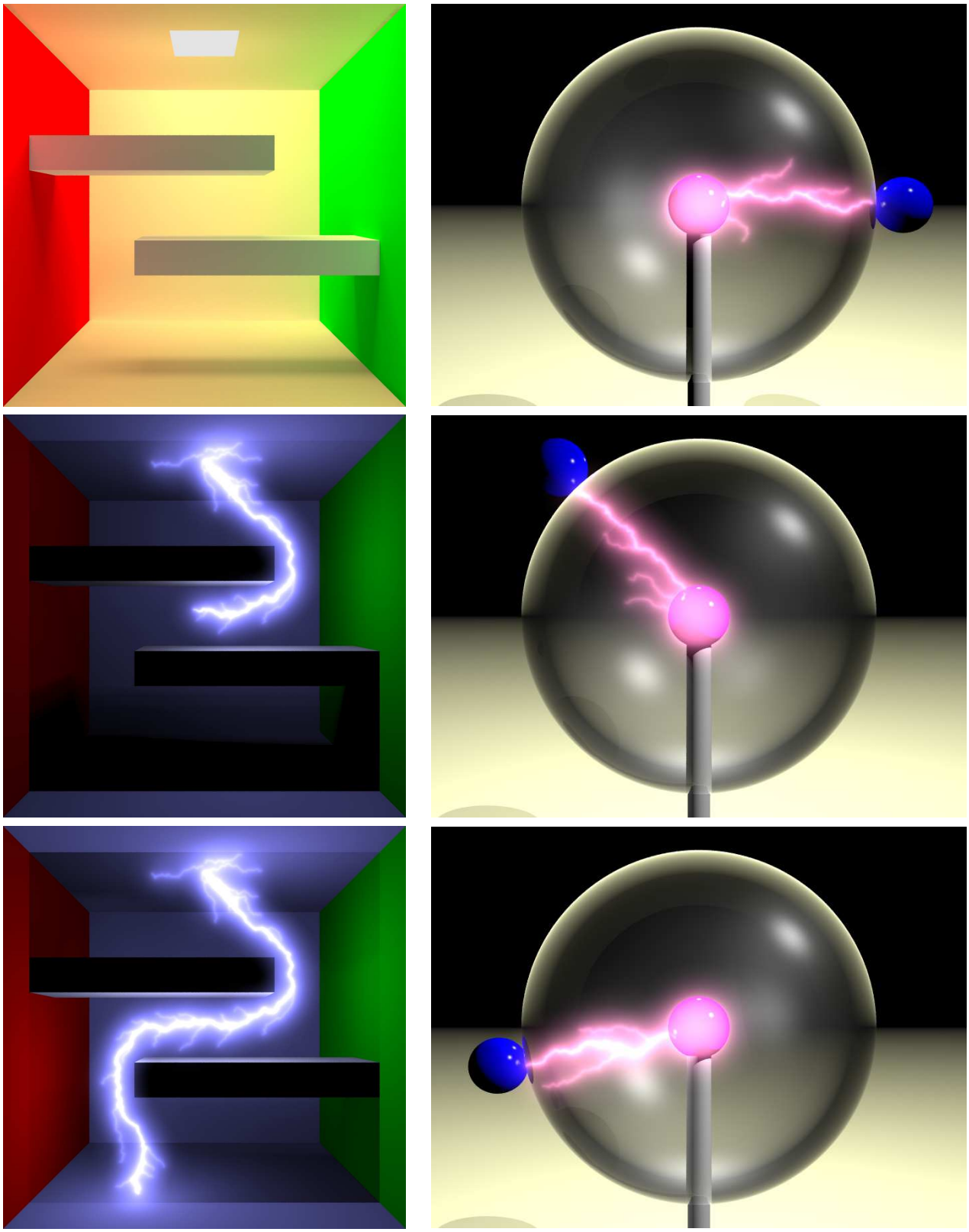

Fig. 17. **Left:** Lightning dodging obstacles in a Cornell Box. *Top to bottom:* The Cornell Box setup; Lightning dodging the first obstacle; Lightning dodging the second obstacle **Right:** Lightning following a blue ball. The magenta electrode is set to negative charge, and the blue ball to positive charge. As the blue ball moves, the arc follows.