Fast Simulation of Laplacian Growth

Theodore Kim, Jason Sewall, Avneesh Sud and Ming C. Lin
University of North Carolina at Chapel Hill, USA
{kim,sewall,sud,lin}@cs.unc.edu

***Abstract*— Laplacian instability is the physical mechanism that drives pattern formation in many disparate natural phenomena. However, current algorithms for simulating this instability are impractically slow and memory intensive. We present a new algorithm that is over three orders of magnitude faster than previous methods and decreases memory use by two orders of magnitude. Our algorithm is based on the dielectric breakdown model from physics, but is faster, more intuitive, easier to implement, and simpler to control. We demonstrate the ability of our algorithm to simulate various natural phenomena and compare its performance with previous techniques.**

***Index Terms*— procedural texturing, natural phenomena, fractals, diffusion limited aggregation, dielectric breakdown model**

I. Introduction

Laplacian instability occurs when a smooth interface evolving under a Laplacian field develops rapidly growing spikes and branches. Many fields are Laplacian, including the steady-state heat equation, electric potential, and an incompressible fluid pressure field. The instability has been connected to many disparate phenomena, such as dendrites on snowflakes, forks on lightning, quasi-steady-state fracture, lobes on lichen, coral, riverbeds, vasculature, and urban sprawl patterns. Since the formation of non-smooth features from smooth initial conditions is counter-intuitive, this topic has attracted attention in physics, chemistry, and material science.

Simulating Laplacian instability, or *Laplacian growth*, has not been a widely used technique in computer graphics, perhaps because existing algorithms, specifically the *dielectric breakdown model* (DBM) [1], have prohibitive space and time requirements, and are daunting to implement efficiently. However, many natural structures arise through this instability, thus fast simulations of Laplacian instability can provide a powerful, general, and physically-based method for describing a wide variety of natural phenomena.

In this paper, we present a fast Laplacian growth algorithm that is structurally similar to DBM but simulates a different physical case, *fuse breakdown* [2]. Our algorithm admits a spherical harmonic solution, which allows it to take into account arbitrary boundary data, such as an environment map. We have not found a similar algorithm elsewhere in the literature, and our method appears to be novel from a physics standpoint as well, as it suggests that Laplacian growth does not require the evolving interface to be an equipotential surface. The main contributions of this paper are:

- A fast, exact, memory-optimal fractal growth algorithm;
- A spherical harmonic formulation that takes into account arbitrary boundary information, such as environment maps;
- A user parameter that allows high-level modulation of dimension, along with intuitive controls for low-level detail;
- Simple implementation that requires neither the linear system solvers nor point location data structures of previous methods;
- Proof-of-concept demonstration of our algorithm on modeling of several natural phenomena and pattern formation.

We apply our algorithm to generation of the tree and lightning as shown in Figures 4 and 5(a), and to the Tesla coil discharge in Figure 7. We measured a performance gain of over three orders of magnitude, and two orders of magnitude in memory savings over previous techniques. We also demonstrate the use of our algorithm to augment the existing framework of L-Systems [3], as shown in Figure 6.

II. Previous Work

Laplacian growth algorithms are closely related to fractals, as they can often produce structures with a fractal dimension. The term ‘fractal’ was coined by Mandelbrot [4] to characterize non-integral, or ‘fractional’ values for physical dimension. In addition to the usual 1D, 2D and 3D, Mandelbrot defined in-between dimensions such as 1.71D or 2.55D. The Koch snowflake and the Mandelbrot set have since become instantly recognizable computer-generated structures, and fractal terrains, fractal textures [5], and L-Systems [3] are now standard visual effects tools.

There are three classes of Laplacian growth algorithms. Listed in order of increasing generality they are: diffusion limited aggregation [6], the dielectric breakdown model [1], and Hastings-Levitov iterative conformal mapping [7]. We will abbreviate them respectively as DLA, DBM, and HLCM. While HLCM is the most general algorithm, it is also the most mathematically involved, and can be difficult to interpret physically. Alternately, DLA has a simple physical interpretation, but can only produce a narrow set of fractal structures. By comparison, DBM offers clean physical intuition while still producing a rich variety of growth structures.

In computer graphics, DLA has been used to produce ice [8], lichen [9], and DBM has been used to create lightning [10]. Many other natural phenomena can be generated by these algorithms [11], but they require supercomputer-scale resources for the simulation. However, the efficiency of our algorithm should make the generation of these phenomena practical on commodity desktop PCs.

A. The Dielectric Breakdown Model

As our algorithm is structurally similar to DBM, we will briefly describe the DBM algorithm here. DBM was first

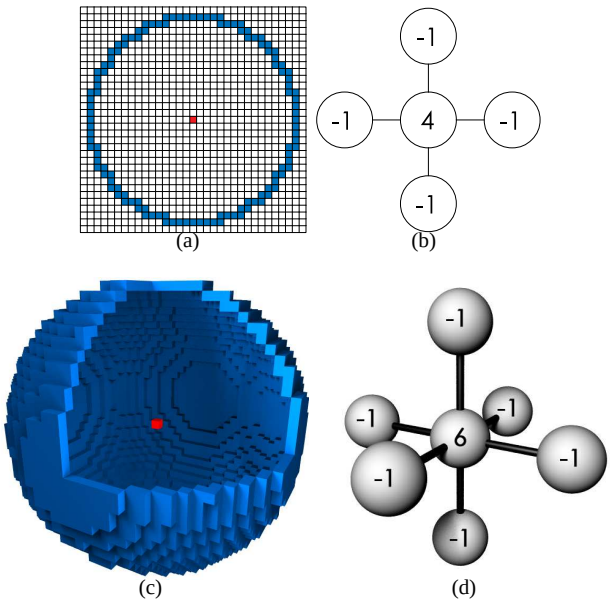


Fig. 1. (a) Initial conditions for 2D DBM. Red: $\phi = 0$, Blue: $\phi = 1$ (b) 2D Laplace stencil (c) Initial conditions for 3D DBM (Octant cut away for clarity). (d) 3D Laplace stencil

described by Niemeyer et al. [1] to simulate the branching patterns that occur in electric discharge. While the model generalizes to many natural phenomena, we will describe it intuitively in terms of electrical discharge. The simulation proceeds in three steps:

1) Calculate the electric potential ϕ on a regular grid according to some boundary condition.
2) Select a grid cell as a 'growth site' according to ϕ.
3) Add the growth site to the boundary condition.

One application of these three steps is considered a single iteration of DBM. The algorithm is iterated until the desired growth structure, or *aggregate*, is obtained.

The 2D initial boundary conditions described in the original paper [1] are shown in Figure 1(a). The red cells represent a boundary condition of $\phi = 0$, and the blue cells are $\phi = 1$. Intuitively, $\phi = 0$ corresponds to a region of negative charge, and $\phi = 1$ a region of positive charge. The potentials ϕ in the neutral white cells are obtained by solving the Laplace equation,

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

according to these boundary conditions. In 2D, the Laplace equation can be solved by constraining the values of the grid cells according to the 5 point Laplacian stencil (Figure 1(b)). These constraints produce a linear system that can then be solved with an efficient solver such as conjugate gradient.

Once the potential ϕ is known, a growth site must be selected. All grid cells that are adjacent to negative charge are considered candidate growth sites. The growth site is then randomly selected from a distribution weighted according to the local potential at each candidate site. The weighted probability function is given in Eqn. 2,

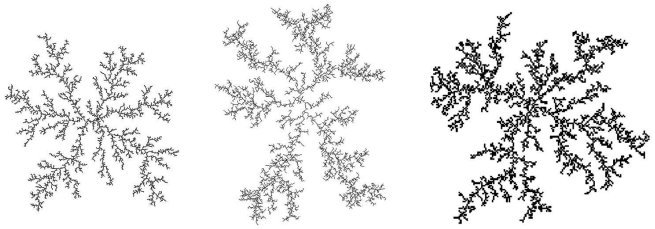


Fig. 2. **From left to right:** Results of DLA, our algorithm, and DBM. Despite the fact that we are solving a different physical case, the characteristic branching patterns of Laplacian growth are still observed.

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

where i is the index of some candidate growth site, n is the total number of candidate growth sites, ϕ_i is the potential at site i, and p_i is the probability of selection for site i. Once the site has been selected, it is set to $\phi = 0$, and treated as a boundary condition in subsequent iterations. The algorithm proceeds until the desired growth structure is obtained. Three-dimensional growth can be obtained by instead solving the 7 point Laplacian stencil (Figure 1(d)) over a 3D grid, with an initial enclosing sphere instead of a circle (Figure 1(c)). The initial boundary condition in Fig. 1 is arbitrary, and could be set to other configurations to produce different discharge patterns.

The η term in Eqn. 2 is a user parameter that controls the dimension of the growth structure. At $\eta = 0$, a fully 2D growth structure known as an *Eden cluster* is produced [12], and at $\eta = 4$, a 1D line is obtained [13]. Therefore, by tuning η between 0 and 4, the entire spectrum of structures between 1 and 2 dimensions can be obtained. Similarly, in three dimensions, the spectrum between 1D and 3D can be obtained by tuning η.

III. Fast Simulation of Laplacian Instability

In this section, we will propose a faster, more memory efficient Laplacian growth algorithm. In DBM, a large amount of computation time is spent on the first step. Computing the potential is expensive because we are numerically treating the interior of the aggregate as a perfect conductor, and the charge redistributes drastically even if only a small perturbation to the boundary is introduced. Therefore, computing the potential field for this new distribution is still expensive. In order to circumvent this problem, we constrain the interior of the aggregate to be a perfect insulator instead. Even under these different physical conditions, we have found that the branching patterns characteristic of Laplacian instability still occur (Figure 2). Physically, this suggests that the conductivity of the aggregate is not an essential component of Laplacian instability.

DBM simulates *dielectric* breakdown; when breakdown occurs an insulator (ie dielectric) is converted into a conductor. Our algorithm instead simulates the opposite case, *fuse* breakdown [2], where a conductor converts into an insulator.

A. Solving the Laplace Equation

In the first step of DBM, the solution of Laplace equation can be determined by solving a large linear system corresponding to a uniform grid. However, as the grid size increases, the linear system quickly becomes intractable. Each step of the DBM algorithm increases the size of the aggregate by one, so growing an aggregate of size n takes n timesteps. Given a grid containing G cells, growing an aggregate of size n takes at best $O(n \times G^{1.5})$ time. As G is typically very large, particularly in 3D, this running time scales poorly.

It appears that a conjugate gradient solver should be able to solve the linear system very efficiently since the shape of the aggregate changes relatively slowly between timesteps. However, as stated before, the charge distribution on the interior of the aggregate changes rapidly in order to enforce the $\phi = 0$ boundary condition along the aggregate surface, causing non-trivial changes in the potential field. There exist variants of DBM that try to solve for this charge distribution directly, such as [14], but their formulations involve solving a dense $G \times G$ linear system.

In contrast, we instead treat the interior as an insulator and no longer require a large regular grid. Therefore, we can eliminate the need to solve a large linear system. Conceptually, we replace the $\phi = 1$ and $\phi = 0$ boundary conditions with positive and negative point charges. By summing the fields induced by these point charges, we can compute the potential field produced by a collection of insulated point charges. Formally, this is still a valid solution to the Laplace equation because the Laplace equation is linear, and admits superposed solutions.

We must first determine the field induced by one such insulated point charge. In order to facilitate later performance comparisons to DBM, we utilize boundary conditions that are analogous to those in DBM. Assume we have an $N \times N \times N$ grid, where h is the physical length of a grid cell. The initial case of DBM corresponds to the case where a negatively charged circle of radius $R_1 = \frac{h}{2}$ is located at the center of the grid, and is surrounded by a larger, positively charged ring of radius $R_2 = \frac{Nh}{2}$. The potential in the space between the circle and ring is then defined by the Laplace equation (Eqn. 1). We can solve for ϕ analytically in this initial case by using the spherical form of the Laplace equation:

$$\nabla^2\phi(r,\theta,\beta) = \frac{\partial^2\phi}{\partial r^2} + \frac{1}{r^2}\frac{\partial^2\phi}{\partial\theta^2} + \frac{1}{r^2\sin^2\theta}\frac{\partial^2\phi}{\partial\beta^2} + \frac{2}{r}\frac{\partial\phi}{\partial r} = 0. \tag{3}$$

The positive hollow sphere and negative sphere can then be stated as Dirichlet boundary conditions:

$$\phi(R_1,\theta,\beta) = 0 \tag{4}$$

$$\phi(R_2,\theta,\beta) = 1. \tag{5}$$

In this case, the boundary conditions are independent of θ and β, so we can drop the middle two terms, reducing the PDE to an Euler equation whose solution is the 3D Green's function:

$$\phi = c_1 + \frac{c_2}{r} \tag{6}$$

The constants can then be solved for using the boundary conditions:

$$c_1 = -(\frac{R_1}{R_2} - 1)^{-1} \tag{7}$$

$$c_2 = (\frac{1}{R_2} - \frac{1}{R_1})^{-1}. \tag{8}$$

As R_2 approaches infinity, this function reduces to:

$$\phi = 1 - \frac{R_1}{r} \tag{9}$$

The $R_2 \to \infty$ case is arguably what DBM is trying to simulate in the first place. In nature, we do not often encounter negative charges surrounded by uniform rings of positive charge. Instead, we assume that the universe is charge conserving, and for every negative charge, the universe contains sufficient positive charge to balance it out. However, these positive charges are only well approximated by a homogeneous ring when they are very far away from the negative point charge.

Now that we know the potential induced by a single point charge as shown in Eqn. 9, we can solve for the potential field induced by many insulated point charges by simply summing their respective fields. For some grid cell of index i, the potential can then be calculated by:

$$\phi_i = \sum_{j=0}^{n}\left(1 - \frac{R_1}{r_{i,j}}\right) \tag{10}$$

where j is the index of a point charge, $r_{i,j}$ is the distance between grid cell i and point charge j, and n is the total number of point charges. By treating the aggregate as an insulator instead of a conductor, we have now circumvented the most computationally expensive step of DBM.

B. Algorithm Description

We will now use Eqn. 10 to design a new fast algorithm for simulating Laplacian instability. DBM solves the Laplace equation over a large regular grid because the linear system requires a numerical medium through which far away boundary conditions can be propagated to the candidate sites. Our formulation requires no such propagators. While Eqn. 10 may appear to be an expensive series to compute, we only need to compute it at the candidate sites. Additionally, we observe that there would be a good deal of repeated work between successive iterations. Unlike in DBM, the potential field we construct changes very slowly, as the addition of a single point charge does not force a charge redistribution in the rest of the aggregate. In fact, the values of ϕ_i from the previous iteration are already correct, save for one new charge. In order to exploit this coherence, the potential at each candidate site can instead be computed as:

$$\phi_i^{t+1} = \phi_i^t + (1 - \frac{R_1}{r_{i,t+1}}) \tag{11}$$

where ϕ_i^{t+1} corresponds to the potential at position i at timestep $t+1$, ϕ_i^t is the potential at the same point at the previous timestep, and $(1 - \frac{R_1}{r_{i,t+1}})$ is the potential contributed by the new $(t+1)$th point charge.

Finally, we observe that in DBM, the value of the potential field is constrained to the $[0, 1]$ range. In our formulation, this constraint no longer holds. However, in order for the η exponent in Eqn 2 to be effective, this constraint must be enforced. Therefore, prior to growth site selection, we normalize the potential values according to the current minimum and maximum potential values, ϕ_{min} and ϕ_{max}:

$$p_i = \frac{(\Phi_i)^\eta}{\sum_{j=1}^{n}(\Phi_j)^\eta} \tag{12}$$

where:

$$\Phi_i = \frac{\phi_i - \phi_{min}}{\phi_{max} - \phi_{min}}. \tag{13}$$

We have now described all the components of the fast growth algorithm. To sum up, the algorithm is initialized as follows:

1) Insert a point charge at the origin,
2) Locate the candidate sites around the charge. On a square 2D (3D) grid, these would be the eight (twenty-six) neighbors,
3) Calculate the potential at each candidate site according to Eqn. 10.

An iteration of the algorithm is as follows:

1) Randomly select a growth site according to Eqn. 12.
2) Add a new point charge at the growth site.
3) Update the potential at all the candidate sites according to Eqn. 11.
4) Add the new candidate sites surrounding the growth site.
5) Calculate the potential at new candidate sites using Eqn. 10.

We note that while Eqn. 10 is the Green's function for the 3D case, we also use it when growing 2D fractals. The 2D solution to the Laplace equation requires an impractical charge conservation constraint to be enforced due to the presence of a logarithm. Details are in Appendix A of the supplemental materials.

C. Spherical Harmonic Solution

The algorithm we have described generalizes to arbitrary Dirichlet boundary conditions, allowing the incorporation of spherical functions such as an environment map into the growth conditions. This is accomplished by replacing the outer sphere of uniform positive charge with a function. The new boundary conditions can be stated as:

$$\begin{aligned} \phi(R_1, \theta, \beta) &= 0 \\ \phi(R_2, \theta, \beta) &= f(\theta, \beta). \end{aligned}$$

The function introduces angular dependency, so we must now solve the full spherical Laplace equation (Eqn. 3). Methods of solving this equation are well-known, and are available in any book on partial differential equations. First we replace $f(r, \theta, \beta)$ with its spherical harmonic series

$$f(\theta, \beta) = \sum_{n=0}^{\infty} \sum_{k=-n}^{n} A_{kn} Y_n^k(\theta, \beta), \tag{14}$$

where $Y_n^k(\theta, \beta)$ is a spherical harmonic basis function and A_{kn} is its corresponding coefficient. The spherical Laplace equation is usually solved for as the product of a radial, polar, and azimuthal function. Eqn. 14 already satisfies the polar and azimuthal components, but we must choose an appropriate radial function. Only two radial harmonic functions, r^n and r^{-n-1}, are capable of satisfying our boundary conditions. Our potential function thus takes the form:

$$\phi(r, \theta, \beta) = \sum_{n=0}^{\infty} (a_n r^n + b_n r^{-n-1}) \sum_{k=-n}^{n} A_{kn} Y_n^k(\theta, \beta). \tag{15}$$

The coefficients a_n and b_n solve to:

$$\begin{aligned} a_n &= \frac{R_2^{n+1}}{R_2^{2n+1} - R_1^{2n+1}} \\ b_n &= \left(R_2^{-(n+1)} - \frac{R_2^n}{R_1^{2n+1}} \right)^{-1}. \end{aligned}$$

In order to incorporate the spherical harmonic solution into the overall algorithm, we then use Eqn. 15 in place of Eqn. 9.

However, we cannot set $R_2 = \infty$ as we did in the homogeneous case. Consider the limit of b_n:

$$\lim_{R_2 \to \infty} \left(R_2^{-(n+1)} - \frac{R_2^n}{R_1^{2n+1}} \right)^{-1} = 0.$$

All b_n are forced to zero with the exception of the $n = 0$ case:

$$\lim_{R_2 \to \infty} \left(\frac{1}{R_2} - \frac{1}{R_1} \right)^{-1} = -R_1.$$

The same holds true for a_n. Thus, if we try to set $R_2 = \infty$, we obtain the following result:

$$\phi(r, \theta, \beta) = \left(1 - \frac{R_1}{r} \right) A_{00}.$$

Note that this equation is exactly the same as Eqn. 9, but scaled by the global average of the spherical function, A_{00}. Intuitively, this occurs because the outer boundary condition has been pushed out so far that any angular variation is completely suppressed, and only the overall average remains. Therefore, in order for angular effects to appear in the simulation, R_2 must be set to some finite value. In our simulations, we set R_2 to twice the expected radius of the final aggregate.

D. Algorithm Analysis and Comparison

Over n iterations, the running time of our algorithm is $O(n^2)$. For a single iteration, steps 1, 3, and 5 of our algorithm require $O(n)$ time, and steps 2 and 4 require $O(1)$ time. Therefore, over n iterations, the running time is $O(n^2)$. This is optimal for any exact potential field approach because the contribution of a new point charge must be computed every iteration. As electric potentials have infinite support, this requires updating $O(n)$ candidate sites. By comparison, the running time of DBM is approximately $O(n * G^{1.5})$, since conjugate gradient runs in roughly $O(G^{1.5})$. Since $n << G$, this running time is considerably larger than our algorithm. An exact bound on the running time of DLA is difficult to obtain, as the algorithm contains a Monte Carlo step. However,

experimental results [15] suggest that in 2D, DLA runs in approximately $O(n^{1.71})$, and in 3D, $O(n^{2.55})$.

Our algorithm requires $O(n)$ memory, where n is the number of growth sites in the final aggregate. We only need to track the locations of the point charges and candidate sites, so a large uniform grid is unnecessary. By contrast, DBM requires $O(G)$ space, where G is the number of cells in a uniform grid. In general, $n << G$, so the memory savings are significant, especially in 3D. DLA requires a point location data structure, for which we chose Delaunay triangulation, which has a worst case memory bound of $O(n^2)$. For our DLA implementation, we used the off-lattice DLA version described in [11]. Because our basic algorithm structure differs significantly from that of DLA, any quantitative comparison will necessarily be somewhat tenuous. However, the running time of the off-lattice version does not involve a grid factor G, making comparison to our algorithm more natural.

	Total Runtime	Memory Use
DBM	$O(n * G^{1.5})$	$O(G)$
2D DLA	$O(n^{1.71})$	$O(n^2)$
3D DLA	$O(n^{2.55})$	$O(n^2)$
Our Algorithm	$O(n^2)$	$O(n)$

TABLE I

Asymptotic Analysis: THE FIRST COLUMN IS THE TOTAL TIME TO GROW AN AGGREGATE OF SIZE n, AND THE SECOND COLUMN IS THE MEMORY REQUIREMENTS OF EACH ALGORITHM. IN ALL COLUMNS, G IS THE SIZE OF A UNIFORM GRID. IN GENERAL, G IS MUCH LARGER THAN n, SO ELIMINATING THE G TERM RESULTS IN A SIGNIFICANT SPEEDUP.

E. User Parameters

As our fast algorithm still utilizes the η variable, the ability to generate a final structure of arbitrary dimension is retained. However, the 2D to 1D transition range of $0 \leq \eta \leq 4$ shifts to approximately $0 \leq \eta \leq 10$. It is unclear if the switch from a conductor to an insulator alone is responsible for this shift. The exact mapping of η to the dimensionality of the aggregate is still an area of active research in physics, so further study is necessary to determine the significance of this shift.

The algorithm also permits the use of repulsors and attractors. The growth can be repulsed from user specified regions by inserting extra negative charges into the simulation, and neglecting to add candidate sites around these charges. The growth will then be repulsed from that region of extra charge. Conversely, positive charge can be inserted into the simulation, and growth will be attracted to this positive charge. These two parameters can be used to 'paint' a desired path for the aggregate. The spherical harmonic version of the algorithm provides an efficient method of representing and simulating a dense, complex array of distant repulsors and attractors.

The attractors and repulsors are not restricted to point charges, but can be any geometry that has a closed form solution to the Laplace equation. Therefore, geometries such as infinite lines, planes, and cylinders can be used to manipulate the growth.

steps	DLA (sec)	DBM (sec)	our algorithm (sec)	speedup over DLA	speedup over DBM
2000	434	21821	6	72x	3636x
4000	1691	43119	20	84x	2156x
6000	3571	64650	43	83x	1503x
8000	6099	89199	73	83x	1221x
10000	9093	115466	110	82x	1049x
12000	12881	143165	156	82x	917x
14000	16987	171825	209	81x	822x
16000	21548	202364	269	80x	752x
18000	26592	234085	336	79x	696x
20000	31907	267016	410	77x	651x

(a)

steps	our algorithm (KB)	DBM (KB)	memory savings
2000	1064	687865	646x
4000	1916	687865	359x
6000	2844	687865	241x
8000	3552	687865	193x
10000	4336	687865	158x
12000	5360	687865	128x
14000	4668	687865	147x
16000	5552	687865	123x
18000	6528	687865	105x
20000	7252	687865	94x

(b)

Fig. 3. *Performance comparison.* The top table is compares the running time of our algorithm to that of DBM in the lightning scene in Figure 5(a). The bottom table shows the memory consumption for the same scene. Overall, our algorithm is **651** times faster than DBM, **77** times faster than DLA, and consumes **94** times less memory than DBM.

IV. IMPLEMENTATION AND RESULTS

We implemented the our algorithm in C++ using the linked list and multimap templates available in STL. We implemented DBM with Incomplete Cholesky Conjugate Gradient as its solver. The solver exploits the Intel SSE instruction set, has the Laplace equation hard-coded into the calculations, and was compiled under ICL 8.0. Compared to the commonly available IML++ [16] implementation, our solver performs an average of 7 times faster. The DBM solver was allowed to terminate at a generous 4 digits of precision; adding more digits increased running time by roughly a factor of 2 per digit. The infinity norm was used instead of the usual 2-norm, as it more accurately characterizes the precision of the solution at the candidate sites. DLA requires a point location data structure that allows incremental construction and nearest neighbor queries. We used the optimized Delaunay triangulation module of CGAL [17] for this purpose. All the timings were collected on a 3.2 GHz Xeon PC.

A. Tree Benchmark

The tree in Figure 4 was created by running our algorithm to 1 million particles with $\eta = 3$. In order to simulate the effects of the ground and sun, we placed an infinite plane of negative charge under the initial charge. The potential of this plane was calculated as $\frac{1}{r}$, where r is the perpendicular distance of a point in space to the plane.

We mirrored these environmental conditions in DBM by also placing a negatively charged plate under the initial charge. A plate of positive charge was also placed along the top edge of the simulation grid because DBM requires the presence of some positive charge. The grid resolution was set to 256^3 because visual artifacts become unacceptable at lower resolutions, and η was set to 1 to match the visual results of the tree generated using our algorithm. We mirrored our algorithm's conditions in DLA by emitting particles from a far away plate in the positive y direction. The ground could not be simulated because DLA does not admit any notion of a 'repulsor'.

Our algorithm generated the 1-million-particle tree in Figure 4. (Please see the supplementary document for a photograph of a real maple tree with similar characteristics as the tree shown here.) Due to the massive running time of DBM, we had to limit our comparisons to the first 20,000 particles of the simulation. Our algorithm generated 20,000 particles **1,112** times faster than DBM and reduced memory consumption by a factor of **95**. Compared to DLA, our algorithm generated 20,000 particles **132** times faster. The supplementary appendices also contain a side-by-side visual comparison between an image of our tree and a photograph of a real maple.

The pioneering work L-Systems work of Prusinkiewicz, Lindenmayer, and collaborators [3] has been widely used in computer graphics to generate plants. While our algorithm does not approach the performance of L-Systems, we have found that the two can work in tandem to efficiently simulate plant growth under complex lighting conditions. The spherical harmonic version of our algorithm can be implemented as an L-System environmental module [18]. In that work, the lighting conditions are taken into account by solving the volume rendering equation on a regular grid. By using our spherical harmonic solution in place of this more expensive solver, our algorithm provides a highly efficient, approximate alternative. The result can be seen in Fig. 6, where biased tree growth adds only a few seconds to the overall running time, yet well captures the influence of directional lighting effects on plant growth.

B. Lightning Benchmark

The lightning in Figure 5(a) was created by running our algorithm to 250,000 particles and setting $\eta = 6.3$. We again simulated the ground as a plane of infinite charge, but switched the polarity to positive, and placed the initial charge very far from the plane. The initial conditions for DBM and DLA were set identically to the tree scene because although the specific phenomena being simulated was different, the basic notion of a branching object growing towards a faraway plane remained the same.

Again, due to the massive running time of DBM, we limited our comparison to the first 20,000 particles. Our algorithm generated the first 20,000 particles **651** times faster than DBM, and **77** times faster than DLA (Figure 3(a)). While the performance gain over DBM may appear inferior to that witnessed in the tree scene, it should be noted that we ran DBM at half the resolution of our algorithm.

Our algorithm's lightning simulation corresponds to a 512^3 DBM simulation, but due to resource limitations, we were only able to obtain timing data for a 256^3 grid DBM simulation. Running a 512^3 grid DBM simulation to 20,000 particles would require 5.5 GB of memory and several months of computation. However, we project that our algorithm would give a factor of nearly $30,000$ speedup and a memory savings factor of nearly 400 using the same grid size of 512^3.

Figure 7 shows a Tesla coil discharge simulated by our algorithm. Tesla coils are known for discharging the type of electrical streamers that are common in science fiction and fantasy films. Previous lightning methods [19] can produce visually distracting self-intersections when simulating such large scale patterns. In contrast, our physically-based algorithm causes the streamers to naturally repulse each other, producing a self-avoiding pattern without any user intervention. In order to suppress grid aliasing artifacts, we jittered each particle inside its grid cell.

C. Terrain Generation

Our algorithm can be used to generate heightfields for terrain representation. The lower left image in Figure 5(b) shows a heightfield representation of the SIGGRAPH logo. The fractal used in this scene was generated in less than 2 minutes with an η of 7 and an initial boundary charge configuration designed to constrain the fractal growth to a region resembling the SIGGRAPH logo. We sampled the resulting potential field on a regular grid and used these values to generate the heightfield, which was then rendered in Blender. Please see the supplementary video for a fly-through of this terrain. (This animation also appeared in the Electronic Theatre at SIGGRAPH 2005.)

D. 2D Benchmarks

All the 2D forms in Figure 5(b) were computed in under 2 minutes. Explicit timing comparisons are difficult because they were generated using user parameters that have no clear analogs in DBM and DLA. Instead, we performed timing comparisons with the canonical configuration (Figure 1(a)) and found that our algorithm is up to **5,748** faster than DBM in 2D. Additional 2D timing data is available in Appendix B of the supplemental materials.

E. Rendering

For the tree rendering (Fig. 4), the leaves were placed at every fifth particle, and the results were rendered as swept sphere cubic splines in POV-Ray. The hybrid tree images were rendered in Blender using L-Systems derived from the environmentally-sensitive systems [18]. The lightning scene was modeled in POV-Ray, and the lightning was rendered using the method described by Kim and Lin [10]. The Tesla coil scene was composed and rendered in Blender, with postprocessing applied using the same method. The 2D fractal forms in Figure 5(b) were rendered by calculating Eqn. 10 at every pixel, and then colored according to the method described by Mandelbrot and Evertsz [20]. The colormaps were taken from the popular Fractint program.

F. Discussion and Limitations

The memory requirements of our algorithm at runtime are less stringent than those of DBM. Memory consumption for DBM is constant because it must allocate all the memory it will use over the lifetime of the simulation at the very beginning. This requirement quickly becomes intractable, as a 1024^3 DBM simulation would require over 44 GB of RAM, a size that is beyond most commodity hardware. By comparison, our algorithm allocates memory incrementally during the simulation and uses orders of magnitude less overall.

From Table I, it may appear that 2D DLA is faster than the 2D version of our algorithm; but in practice, we have found that 2D DLA *only* excels at generating structures of precisely dimension 1.71D using the canonical configuration (Figure 1). Our algorithm appears to outperforms DLA in all other cases. See Appendix B for more details.

V. Conclusions and Future Work

We have presented a fast, simple algorithm for simulating Laplacian instability. Our algorithm is mathematically similar to previous methods, but is near optimal in both space and time. We hope that this algorithm will make the simulation of many large-scale Laplacian growth phenomena practical. This method is suitable for simulating several natural phenomena, such as snowflakes, lightning, fracture, lichen, coral, riverbeds, vasculature, and urban sprawl patterns. Therefore, the ripest avenues for future work are rigorous investigations of the physical and mathematical connections between our algorithm and these seemingly disparate phenomena.

Although we artificially constrained our simulation to a virtual square grid, nothing prevents us from generating candidate sites at arbitrary neighbor locations. As such, we could introduce anisotropies into the simulation that were previously impossible due to non-physical grid tiling constraints.

Finally, the Green's function for the Laplace equation is known to exist in arbitrarily high dimensions. For any dimension $D > 2$, the function is:

$$\phi = a + \frac{b}{r^{D-2}}$$

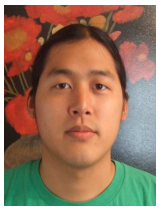
where r is the 2-norm of the difference between two vectors for length D, and a and b are constants. Using our algorithm, we can use these formulæ to construct 'hyperfractals' of arbitrarily high dimension. While it is unclear how to interpret these higher-dimensional fractals, they present an interesting avenue for further investigation.

Acknowledgements

The authors would like to thank Gilles Tran (http://www.oyonale.com/) for making his POV-Ray scenes available online. Figures 5(a) and 4 are derived from his work and used under Creative Commons license 1.0. The lightprobe in Fig. 6 was created using HDRI data courtesy of Industry Graphics (www.realtexture.com) from the LightWorks HDRI Starter Collection (www.lightworkdesign.com). This work was supported in part by Army Research Office, Defense Advanced Research Projects Agency, Intel Corporation, National Science Foundation, Office of Naval Research, and RDECOM.

References

[1] L. Niemeyer, L. Pietronero, and H. J. Wiesmann, "Fractal dimension of dielectric breakdown," *Physical Review Letters*, vol. 52, pp. 1033–1036, 1984.
[2] B. K. Chakrabarti and L. G. Benguigui, *Statistical physics of fracture and breakdown in disordered systems*. Oxford University Press, 1997.
[3] P. Prusinkiewicz and A. Lindenmayer, *The Algorithmic Beauty of Plants*. Springer Verlag, 1990.
[4] B. Mandelbrot, *The Fractal Geometry of Nature*. W H Freeman, 1982.
[5] D. Ebert, F. Musgrave, D. Peachey, K. Perlin, and S. Worley, *Texturing and Modeling: A Procedural Approach*. AP Professional, 1998.
[6] T. Witten and L. Sander, "Diffusion-limited aggregation, a kinetic critical phenomenon," *Physical Review Letters*, vol. 47, no. 19, pp. pp. 1400–1403, 1981.
[7] M. Hastings and L. Levitov, "Laplacian growth as one-dimensional turbulence," *Physica D*, vol. 116, pp. 244–252, 1998.
[8] T. Kim, M. Henson, and M. Lin, "A hybrid algorithm for modeling ice formation," *Proc. of ACM SIGGRAPH / Eurographics Symposium on Computer Animcation*, 2004.
[9] B. Desbenoit, E. Galin, and S. Akkouche, "Simulating and modeling lichen growth," *Proc. of Eurographics 2004*, 2004.
[10] T. Kim and M. Lin, "Physically based modeling and rendering of lightning," *Proc. of Pacific Graphics 2004*, 2004.
[11] T. Vicsek, *Fractal Growth Phenomena*. World Scientific, 1992.
[12] M. Eden, "A two dimensional growth process," in *Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability*, 1961.
[13] M. Hastings, "Fractal to nonfractal phase transition in the dielectric breakdown model," *Physical Review Letters*, vol. 87, no. 17, 2001.
[14] C. Amitrano, "Fractal dimensionality of the η model," *Physical Review A*, vol. 39, pp. 6618–6620, 1989.
[15] K. Moriarty, J. Machta, and R. Greenlaw, "Parallel algorithm and dynamic exponent for diffusion-limited aggregation," *Physical Review E*, vol. 55, no. 5, pp. 6211–6218, 1997.
[16] J. Dongarra, A. Lumsdaine, R. Pozo, and K. Remington, "A sparse matrix library in c++ for high performance architectures," in *Proceedings of the Second Object Oriented Numerics Conference*, 1992, pp. 214–218.
[17] A. Fabri, G.-J. Giezeman, L. Kettner, S. Schirra, and S. Schönherr, "The cgal kernel: A basis for geometric computation," in *Proc. 1st ACM Workshop on Appl. Comput. Geom.*, vol. 1148, 1996, pp. 191–202.
[18] R. Mech and P. Prusinkiewicz, "Visual models of plants interacting with their environment," in *SIGGRAPH '96: Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, 1996, pp. 397–410.
[19] T. Reed and B. Wyvill, "Visual simulation of lightning," *Proc. of SIGGRAPH*, 1993.
[20] B. Mandelbrot and C. Evertsz, "The potential distribution around growing fractal clusters," *Nature*, vol. 348, pp. L143–L145, 1990.



Theodore Kim Theodore Kim received his M.S. (2003) and Ph.D. (2006) in Computer Science from the University of North Carolina at Chapel Hill under Ming Lin, and a B.S. (2001) in Computer Science from Cornell University. His thesis dealt with physically-based simulation and rendering of solidification. He is currently a post-doctoral researcher at UNC, and is investigating efficient, physics-based simulation of various natural phenomena.

Jason Sewall Jason is currently a graduate student of Professor Ming Lin at the University of North Carolina at Chapel Hill researching topics from physically based modelling and graphics. He received a Batchelor of Arts in Mathematics and a Batchelor of Science in Computer Science from the University of Maine in 2004.

Avneesh Sud Avneesh Sud is a final year PhD student at Dept of Computer Science at University of North Carolina at Chapel Hill, under the advisory of Prof. Dinesh Manocha. He received his B.Tech (2000) in Computer Science and Engineering from Indian Institute of Technology, Delhi, and his MS (2003) in Computer Science from the University of North Carolina at Chapel Hill. His research interests include geometric and solid modeling, collision detection and proximity queries, effective utilization of commodity graphics hardware to solve geometric problems and interactive rendering of large models.

Ming Lin Ming C. Lin received her B.S., M.S., Ph.D. degrees in Electrical Engineering and Computer Science in 1988, 1991, 1993 respectively from the University of California, Berkeley. She is currently a full professor in the Department of Computer Science at the University of North Carolina (UNC), Chapel Hill. Prior to joining UNC, she was an assistant professor of the Computer Science Department at both Naval Postgraduate School and North Carolina A&T State University, and a Program Manager at the U.S. Army Research Office.

Fig. 4. A tree generated with one million particles using only our algorithm, **1112** times faster than DBM and **132** times faster than DLA.

Fig. 5. **a)** Lightning generated with a 250,000 particle aggregate. Our algorithm is at least **3 orders of magnitude faster** than existing methods and consumes at least **2 orders of magnitude less** memory. **b)** 2D Fractal forms generated with our algorithm. The SIGGRAPH logo is a frame from a short animation featured in the SIGGRAPH 2005 Electronic Theater.

Fig. 6. A tree demonstrating bias toward a light source (in the upper right), automatically generated with our method incorporated into "environmentally-sensitive" L-Systems. Inset: a tree grown with the same L-System, without environmental bias.

Fig. 7. A Tesla coil discharging, generated with 500,000 particles with our algorithm. The physical foundation of our algorithm makes the simulated lightning streams repulse each other naturally, producing a self-avoiding pattern without any user intervention.