# Frost Formation Via Monte-Carlo Simulations

Ricky Palaguachi*, Noah Roselli†, Divjyot Singh‡, Joseph Torsiello§
New Jersey Institute of Technology
Newark, New Jersey
*rsp84@njit.edu, †ndr23@njit.edu, ‡ds877@njit.edu, §jnt24@njit.edu

***Abstract*—Throughout this paper, the underlying physics for frost formation on a microstructured surface is explained. Then two Monte-Carlo simulations of frost spreading on micropillar surfaces were developed to recover experimental results of frost formation in different regimes of temperature and humidity. The first model was inspired by Diffusion Limited Aggregation (DLA) which involves sending out random walkers one at a time; the second model vectorized this algorithm and sped up the computational times for the pattern formation. Ultimately, the results were able to replicate the experimental results in different regimes by tweaking parameters that described the probability of a random walker freezing or drying.**

## I. Introduction

Frost nucleation and spreading on solid surfaces is a common process which is of interest and concern in everyday life; thus, it is important to understand the mechanisms involved in frost spreading. While this process is difficult to quantify on unstructured surfaces, growth of frost on microstructured surfaces is easier to quantify and model. Recent papers ([1], [2]) have considered frosting on such surfaces experimentally. One novel finding that was reported in these papers is that there are different regimes of frost growth that are characterized by the surface temperature and the relative humidity of the environment. For intermediate humidity, it appears that the frost growth is fractal.

For example, as shown in Figure 1, different regimes cause different patterns to form. In Section II, the underlying physical mechanism will be described for why patterns such as these occur, however to gain intuition into these shapes, it is relatively simple to understand. In each of these runs, the surface temperature is the same; the only thing that is changing is the relative humidity of the air above the microstructured surface. As the humidity increases, the amount of surface area that the frost particles cover grows; furthermore, the shape becomes rounder and rounder and ultimately overtakes most of the domain.

Understanding the mechanisms for frost formation is of utmost importance to the global infrastructure because of how much impact frosting has on crops, plumbing, refrigerators, and HVAC systems. Understanding why and how frost forms can help politicians develop strategies to minimize the impact in cost that frost formation has on real world systems. For example, ice impacts aviation, telecommunications, electrical transmission, hydropower, wind power, oil rigs and the transportation industries. According to [1], frost accounts for 40% of road accidents in winter, accumulation of frost on refrigerators or heat exchangers decreases their efficiency by

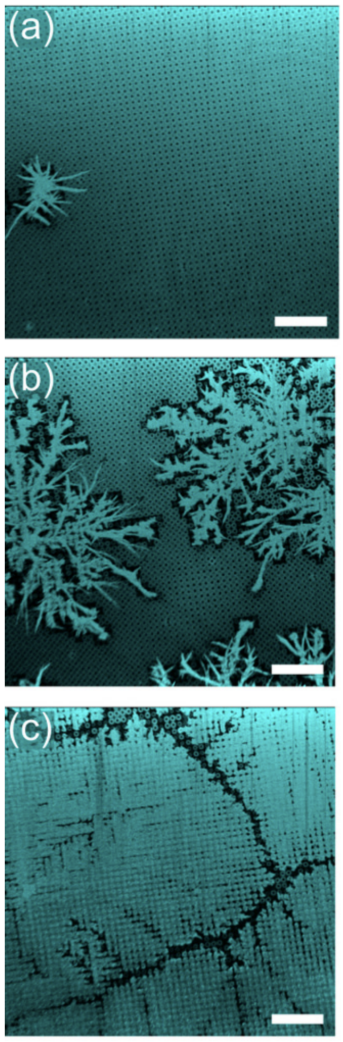


Figure 1. Frost for different relative humidity (RH) result in different shapes. All three runs are kept at $T_{\text{set}} = 30°C$. In (a) RH = 14%, in (b) RH = 24%, and in (c) RH = 34%. Figure courtesy of [2]. The frost particles are the light turquoise and the dry regions are black.

Figure 2. Schematic showing the supercooled condensation stage. You can see the individual water droplets growing.

Figure 3. Schematic showing the onset of freezing inside a water droplet. The four drops shown are in the condensed liquid (blue), recalescence (teal), ice front propagation, and frozen droplet (cyan) stages respectively.

Figure 4. Schematic showing the formation of frost halos. The arrows represent the water vapor leaving the drop in the recalescence stage. The two figures show condensation and desublimation halos respectively.

as high as 50-75%, and wind turbines experience power losses at about 50% with ice accretion.

## II. A Brief Introduction to the Physics

To be able to model frost formation, it is important to first understand the physics behind how these frost particles form in nature. In a recent review [1], this physics is presented and the condensation frosting process is broken into 5 stages: (i) supercooled condensation, (ii) onset of freezing, (iii) frost halos, (iv) interdroplet ice bridging and dry zones, and (v) percolation clusters and frost densification.

Note that while these stages are being described separately, a lot of these processes happen in parallel rather than sequentially.

### A. Supercooled Condensation

For frost formation to begin, there must be a surface with water vapors surrounding it. The temperature of the surface must be below the dew point, which is the temperature at which water condenses. However, this is not a sufficient condition; condensation involves the formation of new interfaces, which needs a certain degree of supersaturation in the surrounding air. This necessary condition can be mathematically calculated using the principles of thermodynamics and estimating the Gibbs free energy. This can be interpreted as subcooling, or simply a temperature lower than the dew point. There is also a possibility of desublimation, where the water vapor directly freezes to ice and skips the liquid condensation stage.

Once water droplets start to form and grow, the distance between neighboring water droplets decreases and their vapor profiles start to overlap as shown in Figure 2. This results in a temporary decrease in the rate of water droplet growth. However, as the droplets keep growing they start to merge or coalesce. A large number of coalescence events results in the formation of thin water layer and accelerated growth.

### B. Onset of Freezing

Now, we zoom into a single water droplet to understand how the freezing process begins. Just like the condensation process, the temperature needs to be below the freezing point, but that is not a sufficient condition by itself. A certain degree of subcooling is necessary for the formation of new interfaces, which can be calculated using the Gibbs free energy.

The freezing inside the droplets begins as a probabilistic nucleation event. When a large number of these embryo form, it results in a large release of latent heat, and this stage is called recalescence. The droplets is now a mixture of frozen nuclei and liquid water, and is analogous to an *ice slushy*. This stage is followed by a freeze front propagation, which starts from the solid surface where the water droplet is attached and moves away towards the air. This typically ends in a pointy tip. This process is illustrated in Figure 3.

### C. Frost Halos

An interesting phenomenon observed during the formation of frost is halos around the droplets during the recalescence stage. As we know, in this stage, the temperature of the droplet increases to the freezing temperature, and this results in some water vapor escaping the droplets. Given the right thermodynamic conditions, these micro droplets can condense on the surface around the *ice slushy*, forming a ring or halo of liquid water. It is also possible that the water vapor desublimes to form an ice halo. Although this is mathematically possible, these desublimation halos are yet to be verified experimentally. Figure 4 shows this process of formation of frost halos.

### D. Interdroplet Ice Bridging and Dry Zones

As water droplets, the vapor pressure in the neighboring region decreases. Due to this, they pull water vapor from neighboring droplets which are still in the recalescence stage, which are releasing water vapors for the frost halos. When this flow of water vapor from the recalescence to ice droplets touches the ice, it results in immediate freezing of the the water vapor and the recalescence droplet. This phenomenon is referred to as ice bridging, and it causes an accelerated growth of ice formation.

As the water vapors move towards the ice particles, it is also possible that the recalescence droplets evaporates completely before the water vapors reach the ice. This causes the formation of dry zones at the location of the recalescence droplet, which are devoid of water and water vapor.

Ice bridging and dry zone formation is shown in Figure 5

### E. Percolation Clusters and Frost Densification

The ice bridging phenomena discussed before results in a chain reaction as more and more ice droplets are formed. The phenomenon percolates through the entire cluster forming ice bridges or dry regions depending on whether these bridging attempts succeed or not. Once an interconnected ice network is formed throughout the cluster, water vapor can now condense outside the plane on top of the droplets. This increases the mass and, hence, density of the frost particles. These two phenomena are shown in Figure 6.

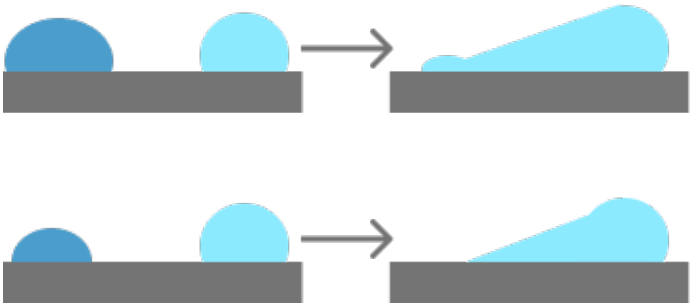

Figure 5. Schematics showing interdroplet ice bridging (above) and dry zone formation (below). Note how the release of water vapor from the recalescence water droplets results in a decrease of size or complete evaporation.

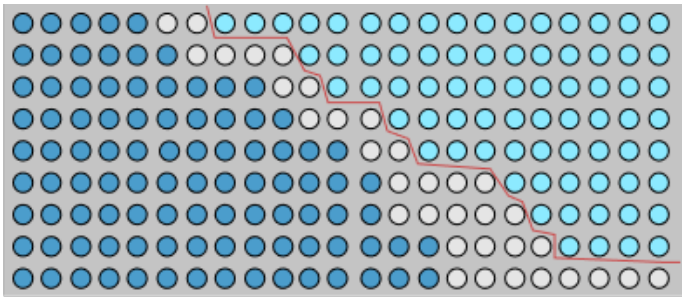

Figure 6. Schematics showing the formation of percolation clusters (above) and frost densification (below). Notice the formation of dry regions (white) as the percolation clusters form.

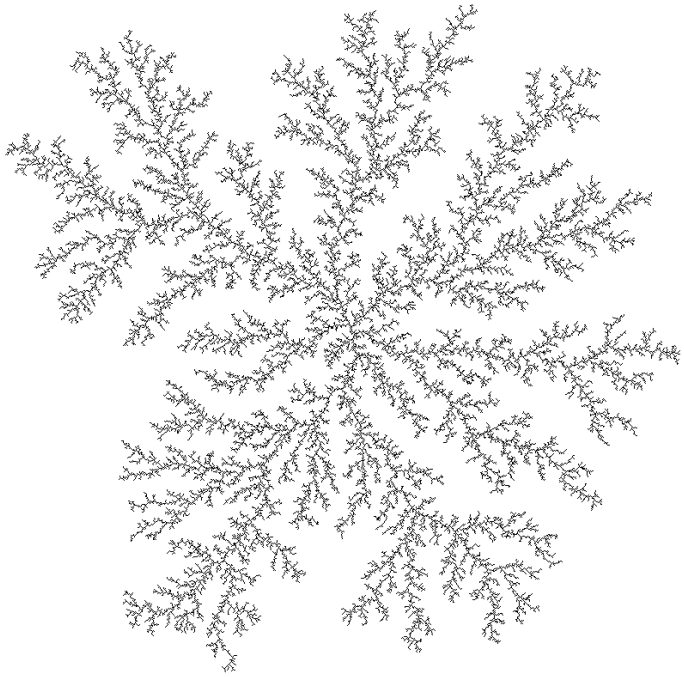

Figure 7. An example of a cluster that is formed through Diffusion Limited Aggregation (DLA), courtesy of [5]. Random walkers congregate against an aggregate and grow outwards in a fractal-like pattern.

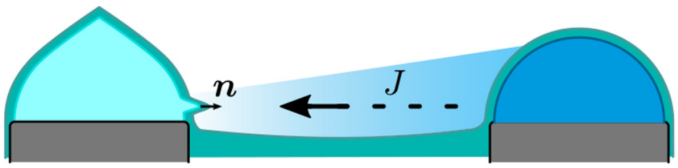


Figure 8. The flux felt from an ice particle to the water droplet, as shown in [2]. These refer to the Equations derived in III-A

## III. Building Up Algorithm

To create the algorithm, we begin with diffusion limited aggregation (DLA). A statistical mechanics growth model initially introduced by Witten and Sander [3], random walkers are sent out from infinity to a central cluster; upon being adjacent to a particle in the cluster, the random walker sticks and becomes part of the cluster, thus causing the nucleus to grow. The name comes from the concept that the particles' movement is defined by diffusion; it is diffusion-limited because the particles that are growing the aggregate are assumed to be in a low enough concentration that they never come in contact with each other; and the central site where the growth is is considered to be the aggregate. As the number of random walkers grows, the aggregate begins to take on a fractal-like shape, as shown in Figure 7. DLA is applicable to a plethora of situations: Hele-Shaw flow, dendritic growth, and dielectric breakdown, to name a few [4].

### A. Connection of DLA to Physics

The governing physics begins with the continuity equation, which tells us that water is conserved throughout the process. Explicitly, this states that:

$$\frac{\mathrm{d}c}{\mathrm{d}t} = -\nabla \cdot J, \tag{III.1}$$

where $c$ is the concentration of water at a node and $J$ is the flux, which can be expressed as:

$$J = D\nabla c \cdot \hat{n}, \tag{III.2}$$

where $D$ is some diffusion constant, $\nabla c$ is the gradient of water on the nucleus's surface, and $\hat{n} = \nabla S/\|\nabla S\|$ is the surface normal vector [2]. We can put these two relations together to end up with the diffusion equation:

$$\frac{\mathrm{d}c}{\mathrm{d}t} = -D\nabla^2 c. \tag{III.3}$$

These variables can be shown in Figure 8. Following the method outlined in [6], we can connect DLA to the diffusion equation. Specifically, let there be a two-dimensional array with lattice spacing $\Delta x$. Furthermore, define $P_{ij}(n)$ as the probability of finding the walker at the position $ij$ after $n$ steps. Then, at the previous step there is an equal probability of

finding the walker at any of the neighboring sites, granted we only consider the "cardinal" directions, and not any movement along diagonals. This can be written as:

$$P_{ij}(n) = \frac{1}{4}\left[P_{i-1,j}(n-1) + P_{i+1,j}(n-1) + \right. \\ \left. + P_{i,j-1}(n-1) + P_{i,j+1}(n-1)\right], \quad \text{(III.4)}$$

which can be rewritten as:

$$P_{ij}(n) - P_{ij}(n-1) = \quad \text{(III.5)}$$
$$= \frac{1}{4}\left[P_{i-1,j}(n-1) - 2P_{i,j}(n-1) + P_{i+1,j}(n-1)\right.$$
$$\left. + P_{i,j-1}(n-1) - 2P_{i,j}(n-1) + P_{i,j+1}(n-1)\right].$$

Then, we let $t = n\Delta t$ and can express:

$$P(n) - P(n-1) = P\left(\frac{t}{\Delta t}\right) - P\left(\frac{t-\Delta t}{\Delta t}\right) \quad \text{(III.6)}$$
$$\approx \Delta t \frac{\mathrm{d}P(t)}{\mathrm{d}t}. \quad \text{(III.7)}$$

We also let $x = i\Delta x$, and concern ourselves, without loss of generality to a one dimensional case. Then:

$$P_{i-1} - 2P_i + P_{i+1} \approx (\Delta x)^2 \frac{\mathrm{d}^2 P(x)}{\mathrm{d}x^2}. \quad \text{(III.8)}$$

Combining the results up to the dimension of our problem, our equation becomes:

$$\frac{\mathrm{d}P(t)}{\mathrm{d}t} = \frac{(\Delta x)^2}{2N\Delta t}\nabla^2 P(t), \quad \text{(III.9)}$$

where $N$ is the dimension of the lattice. Thus, random walkers can be used to solve a diffusion equation, and the model that we are working with is governed by a diffusion equation.

### *B. Creating the Algorithm*

The underlying algorithm is very similar to the DLA algorithm, however we introduce a probability of sticking such that if a random walker is adjacent to the aggregate it has a probability $p$ of sticking instead of immediately ensuring that it will stick.

*1) Playing Field:* The algorithm begins with creating the grid on which the random walkers exist, which we called the "playing field." This was done by defining a length and a width, which we labeled as `rx` and `ry`. Then, a domain that is `2 rx + 1` by `2 ry + 1` is created, such that the origin is at the center and the four corners are `(rx, ry)`, `(rx, -ry)`, `(-rx, ry)`, and `(-rx, -ry)`. To keep track of all of the data that we are storing throughout the process, we use `MATLAB` cell arrays. We initialize the cell array to store five pieces of data: the ordered pair of each lattice point, the current state of the lattice point, the neighbors (in any of the four major directions), the neighbors that are ice, and the neighbors that are dry. All of these concepts will become clearer as this progresses, however, it feels important to mention this is the origin of all of the data.

Each lattice point is initialized such that their current state is liquid. However, to begin the frosting process, we manually change the state of the origin to be ice.

*2) Sending out Random Walkers:* Then, we begin to send out random walkers. Random walkers are spawned on a circle of radius `r` such that angle has equal probability of being chosen; then we define, as typical $x = r\cos\theta, y = r\sin theta$. Then, however, we define a piecewise function that maps the chosen point to a grid point, such:

$$(x,y) \to \begin{cases} (\lceil x \rceil, \lceil y \rceil) & \text{if } 0 \le arg(x,y) < \pi/2 \\ (\lfloor x \rfloor, \lceil y \rceil) & \text{if } \pi/2 \le arg(x,y) < \pi \\ (\lfloor x \rfloor, \lfloor y \rfloor) & \text{if } \pi \le arg(x,y) < 3\pi/2 \\ (\lceil x \rceil, \lfloor y \rfloor) & \text{if } 3\pi/2 \le arg(x,y) < 2\pi \end{cases} \quad \text{(III.10)}$$

which ensures that the walkers are not on the interior of the circle to begin their random walk, as is shown in Figure 9. This is done such that all walkers are given enough time to do a random walk such that their behavior is still governed by diffusion.

We also ensure this by providing a "buffer region" between the aggregate and the radius at which the random walkers are being sent out. Let `buffer` be equal to an arbitrary value; as the aggregate grows, we calculate the further radial point to the highest whole number - that is:

$$\texttt{furthest} = \max\{\left\lceil \sqrt{x_i^2 + y_i^2} \right\rceil\}, \quad \text{(III.11)}$$

for all $i$ in the number of particles. We can succinctly do this by tracking, over each iteration, the radius of the newly introduced particle against the previous iteration's maximum value. Then we can update the value of `furthest` according to any changes that occur.

Then, we send walkers out at r = `buffer + furthest`. This ensures that on each newly sent out random walker, the radius at which they are being released relative to the aggregate is constant. For the runs that we show in the upcoming sections, `buffer` = 30. This can also be easily shown in an example; take the beginning of the algorithm, for example. Since the origin is the only ice particle, the value of `furthest` = 0, and thus random walkers will be sent out from r = 30. Then, after one iteration, let an ice particle stick at $(0, 1)$. Hence, the value of `furthest` = 1 and the random walkers will now be sent from r = `buffer` + `furthest` = 30 + 1 = 31 units. This process continues for the rest of the runs.

On each iteration, we also check whether this new radius is equal to the size of the grid that we are working on; if it is, we stop the algorithm. Thus, we are not growing the domain at all and only have it such that we limit the number of particles that are sent out or we abort once the radius is equal to the size of the grid.

*3) Probability of Freezing:* As explored before, the difference between the algorithm developed and DLA is that there is a probability that the random walker freezes instead of, as in DLA, the particle immediately freezing if it is adjacent to the aggregate. This allows different types of aggregates to form, as is explored in Section VI. We let the $p_{\text{freeze}} \in [0, 1]$.

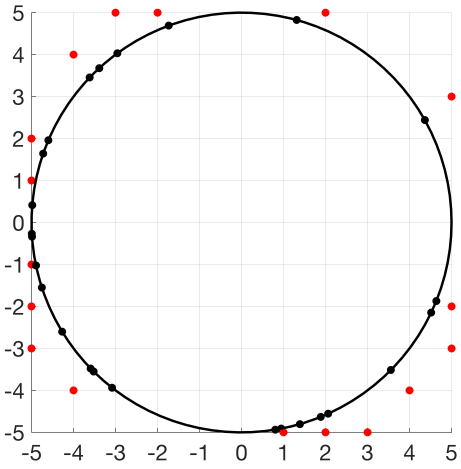


Figure 9. Random walkers are spawned on a circle of an arbitrary radius (plotted in the figure is r = 5). All angles are equally likely, as is shown in the spawned walkers, which are the black dots. To ensure that these walkers end up on grid points, we choose, according to Equation III.10, where to send the walkers. These are shown in red. All walkers are pushed outside of the circle. For more information as to why this should be refer to Section 2.

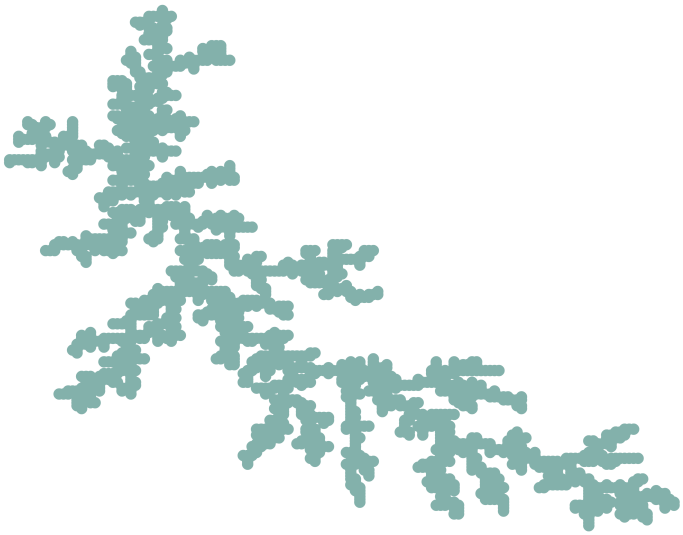

Figure 10. Algorithm run with $p_{\text{freeze}} = 1$. This exhibits the same properties as DLA, which is as expected.

*4) Pseudocode:* The pseudocode for the algorithm, then, is quite simple.

```
create_playing_field();
seed = freeze(0,0);
while (length(stuck) ≤ particles)
    random_walker();
    while (¬stick)
        walk();
        if(on_aggregate)
            if (rand < p_freeze)
                stick();
                stuck += [walker.x, walker.y];
                update_radius();
            else
                walk();
            end
        end
    end
end
```

*5) Further Details and Fine Lines:* Some fine points are outlined in this section for the explicit details of the algorithm. For further clarity, refer to the MATLAB code included at the end of this report.

The walker will only move in the four cardinal directions - up, down, left, or right.

If the walker is against the aggregate and does not freeze, it does not immediately walk to its next point. Instead, the algorithm checks to see if the walker's next walk will cause it to walk into ice. If it does, the walker does not walk; it will continue to choose a new location to walk to until it finds one that is available. Then, it will walk.

We limit the number of iterations that a random walker can have during its lifetime. Each time the random walker walks, this number is increased by one. Once the number of iterations exceeds the limit, the particle is discarded and a new random walker is sent out. This ensures that the algorithm does not run for infinite time if the walker somehow avoids the aggregate for long enough.

If the walker exits the domain, it is sent to the opposite side. For example, if the current $x$ position exceeds `nx`, it will be sent to $x = -\texttt{nx}$.

### C. Algorithm Results

In the following section, the results of the algorithm with different values of $p_{\text{freeze}}$ are investigated. In all of the runs, the number of particles is limited to $N = 3000$, each walker is limited to $n = 5000$ iterations, the domain size if `nx = ny = 200`, and the probability of sticking will be explicitly mentioned.

All of the results are shown in Figures 10, 11, 12, 13, and 14. For high probabilities, the patterns begin to resemble DLA. For low probabilities, the aggregate is much more circular. This is because high probabilities of sticking encourage the aggregate to grow outwards; once a branch begins to form, it is much more likely to continue to grow. On the other hand, if the probability of sticking is low, then the particles will remain around the aggregate for longer. For example, if the particle is adjacent to the aggregate at some point, and does not stick, it will continue to walk; since the probability of sticking is low, it will most likely continue to walk for a long period of time and ultimately end up causing the aggregate to remain more circular.

### D. Including Drying Regions

Recall from Section II that the fourth stage of frosting can result in either ice bridging or a dry region forming. If the distance between two neighboring sites is too large, or if the

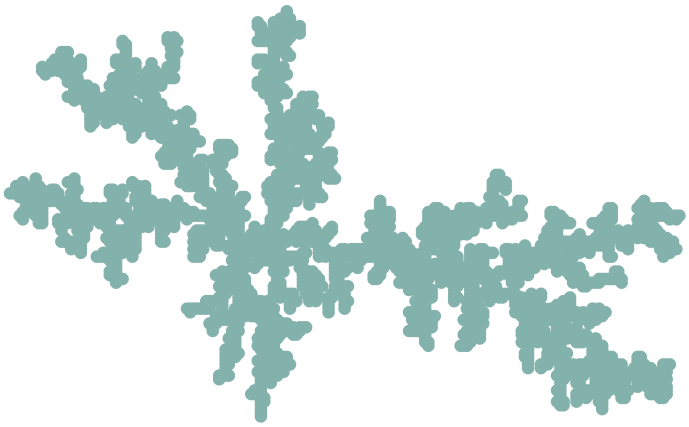

Figure 11. Algorithm run with $p_{\text{freeze}} = 0.5$. As the probability begins to decrease, the branches begin to get thicker. It took the computer 27.72 minutes to perform this algorithm.

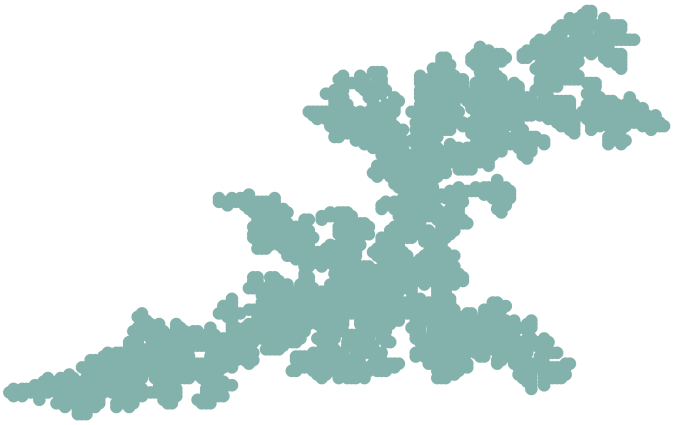

Figure 12. Algorithm run with $p_{\text{freeze}} = 0.1$. Decreasing the probability even further makes the arms even larger. It took the computer 20.06 minutes to complete this algorithm.

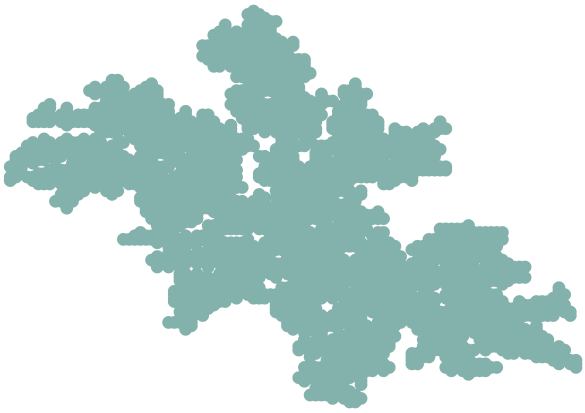

Figure 13. Algorithm run with $p_{\text{freeze}} = 0.05$. The lower probability really begins to encourage circular growth. It took the computer 63.38 minutes to complete this algorithm.

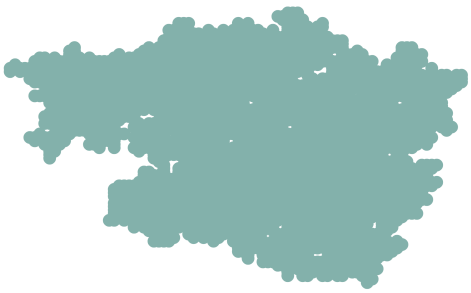

Figure 14. Algorithm run with $p_{\text{freeze}} = 0.01$. For this low of a probability, the output is much more circular. Although the scale is not shown, this image was much more compact than the previous images. It took the computer 32.71 minutes to complete this algorithm.

liquid droplet is too small, the liquid will form a bridge, but the particle itself will never freeze. Thus, it will become a dry region. In the algorithm that we have, the domain has liquid particles every unit apart; therefore, this distance will never be an issue. Furthermore, there is no size restriction that we have on the droplets, thus the size will also never be an issue. Therefore, we have to find a way to add some method to the algorithm that we already have in order to include dry regions on the model. Recall that the purpose of this project is to simulate results from previous physical studies. Since dry regions are included inherently in the physics, we need to implement it into our algorithm.

The simplest way of doing this was by simply adding a probability of drying on the `if` statement that was already implemented in the algorithm from Section 4. However, we must be strategic with the way that we define the probability of freezing and drying. We consider a situation where a particle is against the aggregate; if the particle does not freeze, then the particle has a probability of drying. These two situations are independent and will never overlap; therefore, we consider two values between 0 and 1, inclusive, and let them be the way that we would like the particle to behave - we call one `freezing` and one `drying`. Identically, `freezing` is the probability of freezing, $p_{\text{freeze}}$. However, we must choose $p_{\text{dry}}$ such that it behaves according to `drying`, but is mapped between `freezing` and 1. Therefore we let:

$$p_{\text{freeze}} = \texttt{freezing} \tag{III.12}$$

$$p_{\text{dry}} = \texttt{freezing} + (1 - \texttt{freezing})\texttt{drying}. \tag{III.13}$$

Take the extreme values of `freezing` and `drying`, for example. If we let `freezing` = 1, then $p_{\text{freeze}} = 1$ and $p_{\text{dry}} = 1$ regardless of the value for `drying`. Thus, we get back DLA. On the other hand, take `freezing` = 0 and `drying` = 1. Then $p_{\text{freeze}} = 0$ and $p_{\text{dry}} = 1$.

Thus, we can rewrite our algorithm just slightly to behave according to these new probabilities. Take the `if` statement aforementioned. It will be trasnformed such that:

```
r = rand();
if (r < p_freeze)
    stick();
    stuck += [walker.x, walker.y];
    update_radius();
elseif (r < p_dry)
    dry();
    dried += [walker.x, walker.y];
    update_radius();
else
    walk();
end
```

*1) Results and Discussion:* In subsequent sections, specifically Section VI, we will compare some of the results from this algorithm to the vectorized algorithm explored in Section IV. However, for completeness and discussion, an example algorithm output with frozen and dry regions is shown in Figure 15.

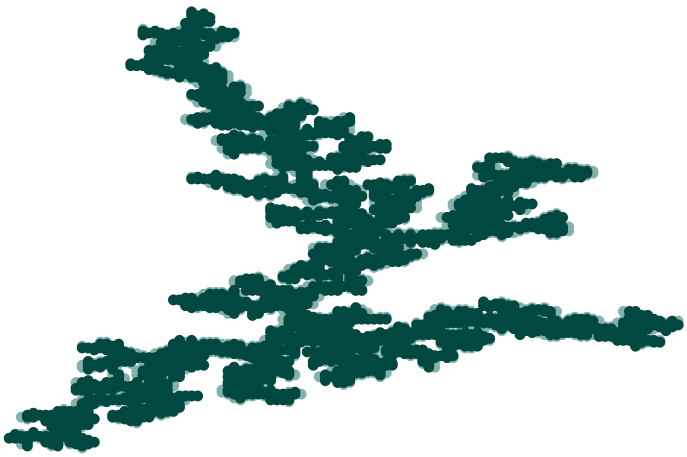

Figure 15. The algorithm with `freezing` = 0.25 and `drying` - 0.125 on a domain `nx = ny = 200`, a total size of $N = 5000$ particles, with $n = 3000$ iterations set as the limit. The dark green is the frost and the light blue is the dry region.

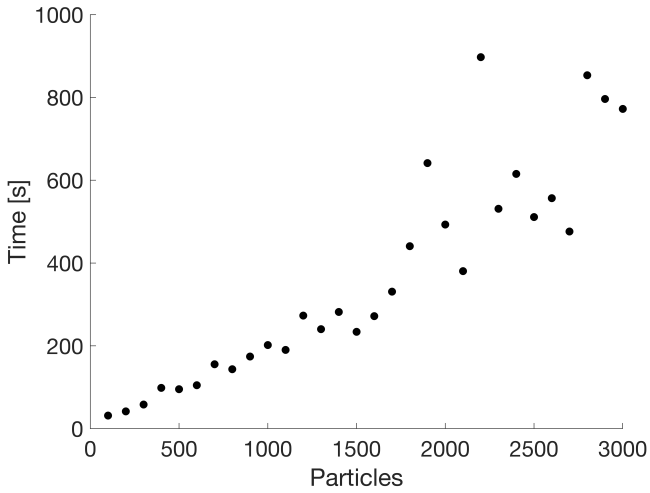


Figure 16. The time that it takes to run the algorithm for increasing values of $N$, the total number of particles in the model. This was produced on a domain of with `nx = ny` = 100, `freezing` = 0.25, and `drying` = 0.125.

From the performance of the algorithm, there are some clear pros and cons that arise. The algorithm works - it provides image that can be related directly to experimental results and replicate the way we expect it to behave. On the other hand, it is costly and can take a long time for the algorithm to run. For example, as shown in Figure 16, for early runs, it seems to be linearly increasing as the number of particles increases, but for high values of $N$, there are greater and greater fluctuations in the time that it takes for the algorithm to run. Therefore, we seek to find methods to decrease the computational time that it takes for the algorithm to run without losing the geometry of the resulting fractals.

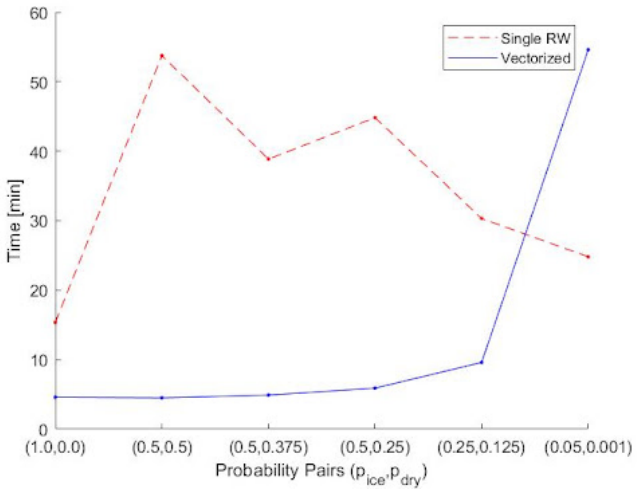


Figure 17. Here is a plot of the time (minutes) each algorithm (Single RW and Vectorized) takes to complete for 6 different pairs of freezing probabilities $p_{\text{ice}}$ and drying probabilities $p_{\text{dry}}$.

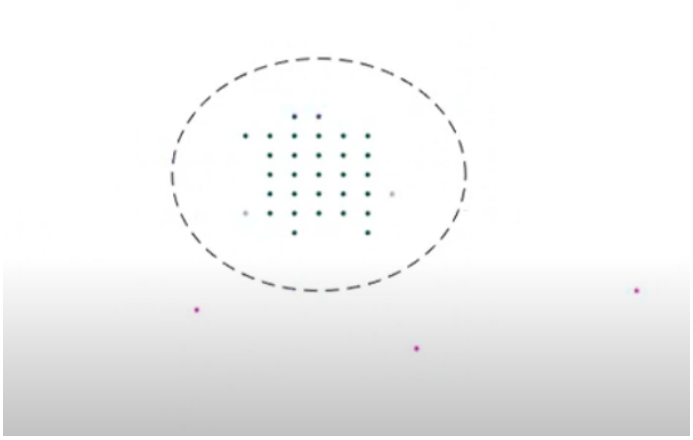

Figure 18. This image illustrates the movement of the vectorized walkers (magenta), the ice particles making up the frost crystal (dark cyan), and the dry region (light cyan). The walkers all started on the black dashed circle.

## IV. Vectorizing the Algorithm

One important initiative that was taken to improve the efficiency of the algorithm was vectorization. Vectorization involves making use of arrays (and array operations) to have multiple operations happen at once programmatically. This is different from parallelization, which involves having multiple operations run at the same time via multiprocessing (distributing the processes on CPUs) and/or multithreading (distributing the processes on several threads). While parallelization can be more efficient since it separates long-running processes into several short-running subprocesses, vectorization was ultimately chosen to make the algorithm more efficient. This is because parallelization is best implemented when attempting to run several 'independent' subprocesses so that no overlaps or 'data races' occur, where two or more threads/CPUs are running in the same process. In order to parallelize the frosting model, multiple processes would have to be tracked to ensure no overlapping, including the state of the frost crystal and dry regions, the positions of all walkers, and the radius of the circle at which walkers are released. Vectorization allows for more efficiency and a way to programmatically store the current states of the domain, walkers, and radius, for every iteration.

Overall, some benefits of vectorization over the simple algorithm, include more control and flexibility, (generally) better efficiency, and (for certain parameters) reproducibilty of the results from the updated DLA algorithm. In fact, Figure 17 compares the amount of time it takes to complete each algorithm (vectorized and not-vectorized) for different pairs of fixed freezing and drying probabilities, and for most probabilities (except those near zero) the vectorized algorithm outperforms the non-vectorized algorithm in efficiency. However, some drawbacks include increased complexity, an exponential decrease in efficiency for smaller freezing and drying probabilities (as the figure also illustrates), and that this vectorized algorithm is a slightly different process from the updated DLA algorithm (which in fact must be illustrated and shown to give similar results).

### A. Vectorizing the Walkers

Figure 18 gives an illustration of how the vectorized walkers (magenta) move throughout a fixed-size domain, and how they can create the frost regions (dark cyan) and dry regions (light cyan) similar to the previous algorithm that implements sticking probabilities with DLA. The walkers all start on a circle of specified radius to simulate coming from infinity (large relative to the size of the frost domain), with all initial cell array positions saved in a column array. In contrast to the previous algorithm, the walkers are released at once, with each walker still having a uniform chance of going into any one of four coordinate directions (up/down/left/right) after each iteration.

The vectorized walkers, walk, by updating the 'walk' function to take in the array of walker positions as input, creating arrays for the x and y coordinates of these positions, and adding either -1, 0, or 1 to each coordinate such that each walker has a uniform probability of moving in one of the four directions. Updating these coordinates for every iteration is done with the following lines of MATLAB code in the walk function.

```
1       rx = randi([-1 0],1,particles) + ...
            randi([0 1],1,particles);
2
3       ry = zeros(1,particles);
4       ry(rx == 0) = randi([2 3],1,length( ...
            ry(rx == 0) ));
5       ry(ry == 2) = -1;
6       ry(ry == 3) = 1;
7
8       xNew = x + rx;
9       yNew = y + ry;
```

This code creates a vector `rx` with elements that have a 1/4 chance to be -1, 1/4 chance to be +1, and a 1/2 chance to be 0. This also creates a vector `ry` with elements that are 0 in the `rx` vector, having a 1/2 chance of becoming either 1 or -1 (respectively), and the rest of the elements in `ry` remaining zero since they are -1 or 1 in `rx` (i.e. the walker moved in an x-direction already). By then adding `rx` to the x coordinates and `ry` to the y coordinates, all the walkers walk to their next position. Most importantly, for each walker, the probability of moving left (right) is 1/4 since each element in `rx` has a 1/4 chance of being -1 (+1), and the probability of moving down (up) is 1/4 since each element in `ry` has a (1/2 times 1/2) chance of being -1 (+1). Each walker essentially moves with uniform probability in one of the four coordinate directions.

Overall, while a few changes have been made to the nature of the previous process of sending a single walker out one at a time, a few features of these vectorized walkers remain unchanged. This includes that any walkers which move beyond the boundaries of the domain are made to go to the other opposite boundary. Also, any walkers which step into a forbidden region (ice or dry) move back/stay at their previous locations. Stepping into a forbidden region only happens when one or more walkers are adjacent to this region, do not become ice or dry, and then step into the region.

### *B. Updating Sticking Probability*

Whenever two or more walkers meet adjacent to the same ice particle, the sticking probability is updated to include the union probability of each walker sticking (i.e. if one sticks they all stick). This is simply the probability that at least one of $N$ walkers (that were adjacent) stick.

The probability that none of $N$ walkers stick is modeled by a binomial distribution with zero 'successes' ($k = 0$) and success probability $\mathbb{P}_f \equiv$ Freezing Probability, i.e. $\mathbb{P}_0 = \binom{N}{0}{\mathbb{P}_f}^0(1-\mathbb{P}_f)^{N-0} = (1-\mathbb{P}_f)^N$. Therefore, the probability that at least one walker sticks is simply $1-\mathbb{P}_0$, which gives the new sticking probability $\mathbb{P}$ of taking all $N$ walkers together as one:

$$\mathbb{P} = 1 - (1-\mathbb{P}_f)^N \tag{IV.1}$$

If this probability is satisfied, all the walkers stick. If the probability is not satisfied, all the walkers also have a probability to dry using the same probability model (with $\mathbb{P}_f$ replaced by the fixed drying probability). Otherwise, all walkers simply continue to walk together.

The process of having walkers adjacent to the same ice particle, either walk together as one (i.e. removing all duplicate positions in the cell array) or stick with this new union probability, was chosen over simply having any walkers that stick, stick, and any walkers that do not stick, continue to walk (of course also having the probability to dry). This is because any of those walkers that continue to walk, may walk further into the forbidden region and get trapped, since any walker which moves into the forbidden region must move back to its previous location (but the previous location may now be part of the frost domain). While the event that two or more walkers meet and are adjacent to the same ice particle is rare, even when thousands of walkers are released at once, this event is still considered in order to programmatically avoid any walkers being trapped within the forbidden regions. This is also an example of dealing with overlapping processes programmatically.

### *C. Verifying the Diffusive-Nature of the Vectorized Walkers*

Consider a single random walker which moves on a 2D lattice. Since moving up or right increases the Manhattan distance by 1 (and moving down or left by -1 respectively), a random walk on a 2D lattice can be modeled (proportionally) by the already well-established 1D random walk (which either moves left or right with equal probability $\frac{1}{2}$). Let's denote the position of this 1D walker after an $n$ number of iterations as $x_n$. Also, let $s_i = 1$ if the step at iteration $i$ was to the right, and let $s_i = -1$ if the step at iteration $i$ was to the left.

The average distance this 1D walker travels $\langle x_n \rangle$ is shown here,

$$\langle x_n \rangle = \langle \sum_{i=1}^{n} s_i \rangle = \sum_{i=1}^{n} \langle s_i \rangle = \sum_{i=1}^{n} \frac{1}{2}(-1) + \frac{1}{2}(1) = 0 \tag{IV.2}$$

which is expected for a random walk which is equally likely to move in any of two coordinate directions.

The average squared distance of the 1D walker, on the other hand, is not zero and is shown here.

$$\begin{aligned}\langle {x_n}^2 \rangle &= \langle \sum_{i=1}^{n}\sum_{j=1}^{n} s_i s_j \rangle = \sum_{i=1}^{n} \langle {s_i}^2 \rangle + \sum_{i=1}^{n}\sum_{j>i}^{n} \langle s_i s_j \rangle \\ &= \sum_{i=1}^{n} \left( \frac{1}{2}(-1)^2 + \frac{1}{2}(1)^2 \right) + \\ &\sum_{i=1}^{n}\sum_{j>i}^{n} \left( \frac{1}{4}(1)(1) + \frac{1}{4}(-1)(1) + \frac{1}{4}(1)(-1) + \frac{1}{4}(-1)(-1) \right) \\ &= n + 0 = n \end{aligned} \tag{IV.3}$$

Therefore, the root mean square (rms) of the distance for the 1D walker (as well as proportionally to the vectorized walkers on the 2D lattice) is $d = \sqrt{\langle x_N \rangle} = \sqrt{N}$.

To ensure that the code which creates and runs the vectorized walkers still exhibit a diffusive nature, the average distance a walker travels $d$ was compared with the square root of the number of iterations $\sqrt{N}$, as the number of iterations increases. These two variables should tend to one another as $N$ approaches infinity, as developed previously.

Figure 19 and Figure 20 show the relative difference between $d$ and $\sqrt{N}$ (Relative Difference $= \frac{|d-\sqrt{N}|}{\sqrt{N}}$) for 5,000 vectorized walkers which start on a circle of radius 30 and 170, respectively. The standard domain of 401 by 401 is chosen (200 from the origin on all sides), and there is no seed for either radius because only the nature of the walkers is being tested. As the plots show, while the relative difference between $d$ and $\sqrt{N}$ does appear to be increasing, it is relatively small with the average distance never going beyond double $\sqrt{N}$ (i.e. the relative difference remains below 1). More importantly, the

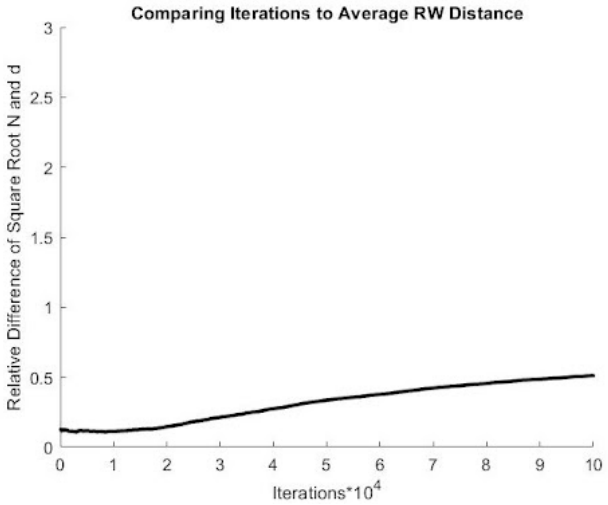


Figure 19. For walkers starting on a circle of radius 30, the relative difference between the average distance a walker travels $d$ and the square root of the number of iterations $\sqrt{N}$, is plotted versus the number of iterations that have occurred. 5,000 walkers are used, and the domain is 401 by 401.

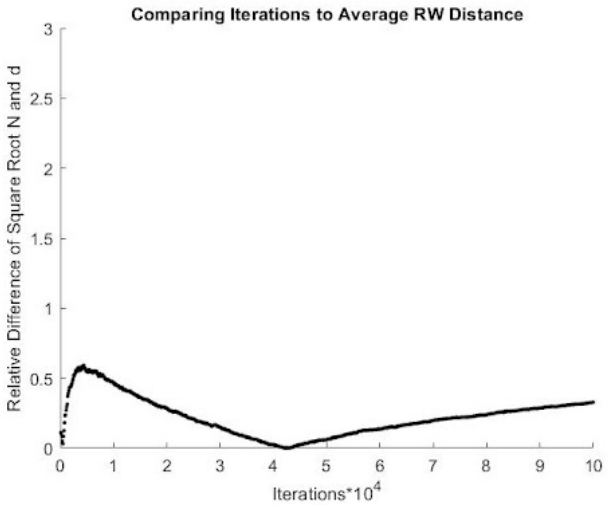


Figure 20. For walkers starting on a circle of radius 170, the relative difference between the average distance a walker travels $d$ and the square root of the number of iterations $\sqrt{N}$, is plotted versus the number of iterations that have occurred. 5,000 walkers are used, and the domain is 401 by 401.

max number of iterations for the walkers is usually fixed at 10,000, and as Figure 19 shows, the relative difference remains quite small (below 0.25) for the first 10,000 iterations. This is not the case for when the radius is 170, which suggests that a smaller radius of around 30 should initially be used (unless more iterations are performed which could cost some extra time).

### *D. Implementing the Vectorized Walker Batches*

Figure 21 demonstrates what happens when too many walkers (5,000) are released within an initial radius of 30, which eventually grows over time as the crystal increases. Even though a smaller radius of around 30 should be used to

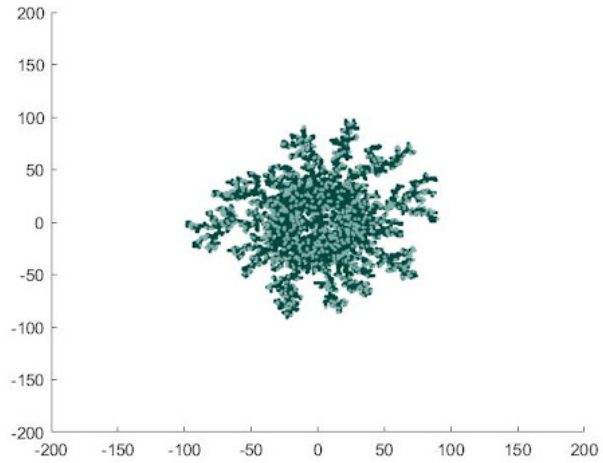

Figure 21. The vectorized algorithm with `freezing` = 0.75, `drying` = 0.25, `nx = ny` = 200, $N$ = 5000 particles, `max iterations per batch` = 10000, and `walkers per batch` = 5000.

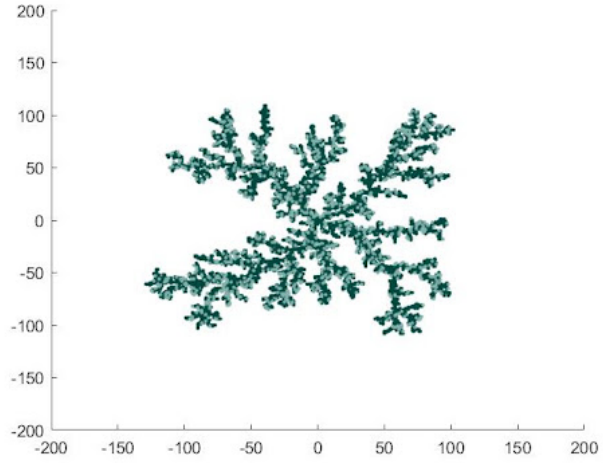

Figure 22. The vectorized algorithm with `freezing` = 0.75, `drying` = 0.25, `nx = ny` = 200, $N$ = 5000 particles, `max iterations per batch` = 10000, and `walkers per batch` = 500.

effectively simulate walkers coming from infinity, if a lot of walkers are spawned with a relatively small region, they are biased to all stick to the seed at once. This bias can be fixed by releasing a smaller number of walkers in groups or 'batches', while still starting with an initially small radius of 30 that grows relative to the size of the crystal, and such that the walker spawning circle remains a distance of 30 from the crystal.

Figure 22 shows the crystal formed when a more reasonable number of walkers are released per batch (500) with the same initial radius of 30. As shown, the crystal grows similar to the updated DLA-like growth expected. A comparison of the results from both algorithms (vectorized and not-vectorized) for different fixed freezing and drying probability pairs, can

be found in the Results section.

Overall, maintaining a small radius of 30 and releasing a small number of walkers relative to the initial area within the initial walker spawning circle, allows for the vectorized walkers to effectively be used to create the crystal in the most efficient and timely manner. Both algorithms were worked on in parallel, and as discussed previously, the results are compared in the Results section.

## V. Fractal Analysis Algorithms

Two different algorithms are used to analyze and extract the fractal dimension $d$ from the different frost crystals created: 1. The Box-Counting Fractal Dimension Algorithm and 2. The Correlation Fractal Dimension Algorithm [7]. Even though each of these two algorithms may appear similar, the fact that they are two different algorithms at the core, leads to potentially different results. Different fractal dimension algorithms are well-known to give different but relatively close results, which still makes them useful metrics for comparing many different fractal patterns, including frost growth.

### A. The Box-Counting Fractal Dimension Algorithm

The Box-Counting Fractal Dimension Algorithm is a very well-known algorithm for computing fractal dimensions (also known as the Minkowski-Bouligand dimension). Here, the procedure of the algorithm is simply laid out:

1. Load data file: A matrix the size of the specified domain, whereby each element is either a 0 if it represents water, a 1 if it represents an ice particle/region, and a 2 if it represents dry particle/region. Note: Any element which contains a 2 is removed, because only the fractal dimension of the frost crystal is analyzed.
2. Define several data resolutions $s$
   - The data resolution, $s$, is a number between 0 and 1, which is proportional to the number of boxes (or pixels) that are used to 'visualize' all of the data ($s = 1$ is defined as the 'max resolution', where a box simply covers an element in the matrix from the data file). It is also inversely proportional to the side length of each box ($1/s$), with the unit box length taken to equal 1 to coincide with the max resolution $s = 1$.
3. Count the number of boxes $N(s)$ programmatically for each data resolution $s$
   - Starting with $s = 1$, the number of boxes $N(1)$ is simply the number of ice particles in the data file (i.e. summing all nonzero elements). For all other $s$, a new data matrix is created which is scaled by $s$ (the resolution). This new data matrix is created by breaking up the main data file matrix into several $1/s$ by $1/s$ matrices (which is done by using the 'conv2' function on MATLAB, and removing any extra indices created from the function). Any of these matrices which simply contain one or more ice particles (element = 1), is considered a box and counted. For each $s$, $N(s)$ is recorded. Note: Any boundaries that stick out are simply omitted.
4. Plot $\ln N$ vs. $\ln s$ and fit to a linear function: The slope of this linear function gives the computed fractal dimension $d$.

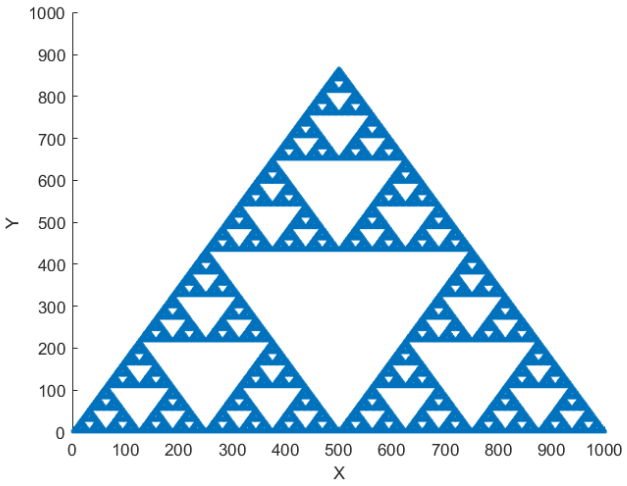


Figure 23. Example of a Sierpinski Triangle

### B. The Correlation Fractal Dimension Algorithm

The Correlation Fractal Dimension Algorithm is an algorithm used to describe the dimensionality for fractals produced in several physical models, including in Shear-Thinning Hele-Shaw Flow in fluid dynamics [7]. Here, the procedure of the algorithm is simply laid out:

1. Load data file: A matrix the size of the specified domain, whereby each element is either a 0 if it represents water, a 1 if it represents an ice particle/region, and a 2 if it represents dry particle/region. Note: Any element which contains a 2 is removed, because only the fractal dimension of the frost crystal is analyzed.
2. Define several $\epsilon$-neighborhoods
   - In contrast to the box-counting method, the number of particle pairs within a radius of $\epsilon$, $g(\epsilon)$, are counted for different $\epsilon$. $\epsilon = 1$ is defined simply as the unit lattice distance.
3. Calculate the Correlation Integral $C(\epsilon)$ for each $\epsilon$

$$C(\epsilon) = \frac{g(\epsilon)}{N^2} \tag{V.1}$$

($N$ is the number of ice particles in the main data file matrix)

4. Plot $\ln C(\epsilon)$ vs. $\ln \epsilon$ and fit to a linear function: The slope of this linear function gives the computed fractal dimension $d$.

### C. Test Case: The Sierpinski Triangle created on MATLAB

Figure 23 shows the Sierpinski Triangle fractal, which has a fractal dimension of $d = \ln 3 / \ln 2 \approx 1.585$ between 1D

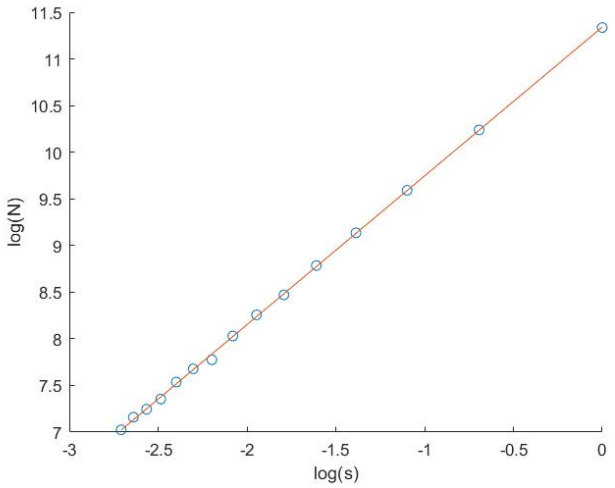


Figure 24. This shows a graph of the ratio of the natural logarithms of the number of boxes $N$ and the different data resolutions $s$ (blue dots). This data is fit to a linear function (red line).

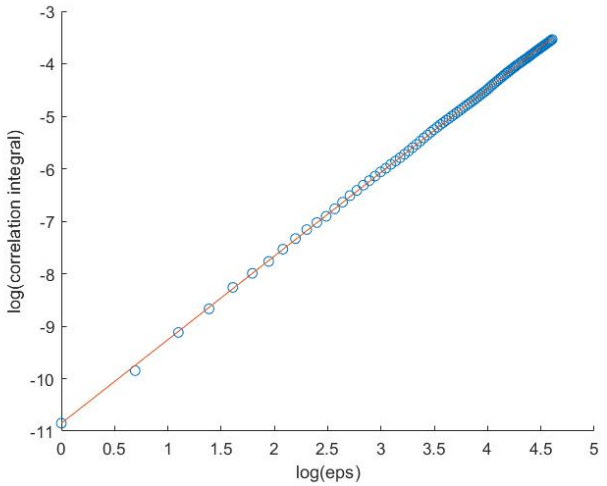


Figure 25. This shows a graph of the ratio of the natural logarithms of the correlation integral $C$ and the different epsilons $\epsilon$ (blue dots). This data is fit to a linear function (red line).

and 2D. This well-known fractal is used to test both fractal dimension algorithms.

Figure 24 shows $\ln N$ vs. $\ln s$ for 15 different $s$ that goes like the inverse power of 2, fit to a linear function. The best-fit slope is approximately $1.595$, which has a percentage error of approximately $0.985\%$ relative to the true value for the Sierpinski Triangle.

Figure 25 shows $\ln C(\epsilon)$ vs. $\ln \epsilon$ for $\epsilon$ from 1 to 100, fit to a linear function. The best-fit slope is approximately $1.594$, which has a percentage error of approximately $0.896\%$ relative to the true value for the Sierpinski Triangle.

Both algorithms are shown to give fractal dimensions relatively close to the true fractal dimension of the Sierpinski Triangle. This confirms that these two algorithms can be used to measure the fractal dimension of several frost crystals. The relative percentage difference between the fractal dimensions of both algorithms is $0.056\%$, which is also relatively small.

## VI. Results and Discussion

We run simulations for the original algorithm (which we will call the simple algorithm) and the vectorized algorithm and compare the two in this section. This is done first visually and then by doing fractal analysis to calculate the fractal dimension of the output from the two algorithms. In all of the following graphics, `nx = ny = 200`, `drying` and `freezing` are noted in the captions of the figures, $N = 5000$ particles are released, and $n = 3000$ is the lifespan of the random walkers.

### A. Comparing Algorithm Outputs

In Figures 26 to 37, different results are shown for both of the algorithms. In each of the captions, the specific values for `freezing` and `drying` are provided. There is a sense of randomness in both of the algorithms, as the random walkers are spawned from random locations and move in random directions, however, more or less, it visually appears that the results are similar. In Section VI-B, we go into more detail on the fractal dimension for each of the algorithms using the two previously defined fractal dimension algorithms (Section V).

### B. Fractal Analysis

For simplicity, we have split up the two fractal dimension algorithms to better comprehend the results.

| Freezing | Drying | Box (Simple) | Box (Vector) |
|---|---|---|---|
| 1 | 0 | 1.3708 | 1.3769 |
| 0.5 | 0.5 | 1.6141 | 1.5825 |
| 0.5 | 0.375 | 1.5545 | 1.5689 |
| 0.5 | 0.25 | 1.5398 | 1.5429 |
| 0.25 | 0.125 | 1.5819 | 1.6021 |
| 0.05 | 0.001 | 1.6633 | 1.6397 |

| Freezing | Drying | Corr. (Simple) | Corr. (Vector) |
|---|---|---|---|
| 1 | 0 | 1.5636 | 1.6118 |
| 0.5 | 0.5 | 1.6030 | 1.6253 |
| 0.5 | 0.375 | 1.5914 | 1.6112 |
| 0.5 | 0.25 | 1.6564 | 1.6730 |
| 0.25 | 0.125 | 1.5557 | 1.6415 |
| 0.05 | 0.001 | 1.7571 | 1.7499 |

Although the results are not consistent between the two methods that were used to calculate the fractal dimension, they provide consistent results between two of the same runs of the patterns that were developed in the simple and the vectorized algorithms. The discrepancies between fractal algorithms is not of too much concern since we were attempting to show that the results between frosting algorithms were consistent. This appears to be relatively true. However, the slight changes between the numbers can be due to a handful of reasons: for example, in some of the simple algorithm runs, not all 5000 particles stuck, therefore we were dealing with shapes of different relative sizes. Furthermore, there is randomness

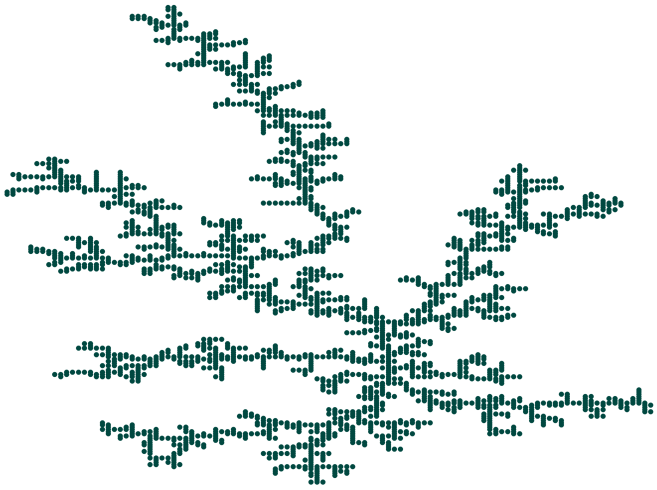

Figure 26. Running the simple algorithm for `freezing` = 1, `drying` = 0. This is DLA. It took 15.37 minutes to compute. Only 2508 particles stuck.

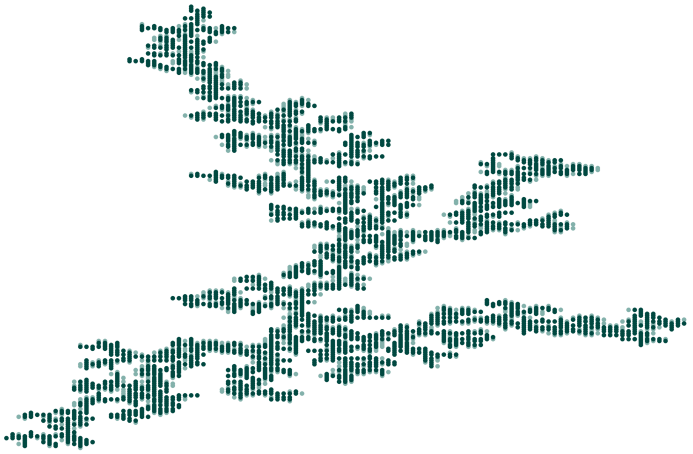

Figure 27. Running the simple algorithm for `freezing` = 0.25, `drying` = 0.125. It took 30.31 minutes for this to generate. Only 3545 particles stuck.

inherent to both of the algorithms and they will never be the exact same.

## VII. Conclusion

As explored in Section II, the underlying physics for frost formation was developed and understood. In Sections III and IV, the two algorithms that were developed were explained and build up from the ground. In Section VI, many examples of the results for different values of `freezing` and `drying` were provided; ultimately, it appears that, in tweaking these two parameters, we can replicate the physical result that was shown in Figure 1. For example, the formation that is shown in Figure 1 (a) resembles the results from Figures 26 and 32. Similarly, the results in Figure 1 (b) resembles the results from Figures 31 and 37. Finally, the results in Figure 1 (c) resembles the results from Figures 35 and 34. Finally, the fractal analysis that was performed in Section VI-B shows consistent results between the simple and vectorized algorithms, though not necessarily the same results between the two fractal analysis algorithms.

For potential next steps for this project, smaller probabilities could be studied closer in order to understand the solid regime a bit better. Furthermore, mathematical models could be developed that relate the physical characteristics of the system, such as the humidity and temperature, to the sticking probabilities. Finally, improvements could be made to the algorithms such that if the entire aggregate is covered by a dry region, the algorithm stops. Ultimately, however, the goal of the project was met: we were able to develop two algorithms that are able to replicate experimental results from previous studies.

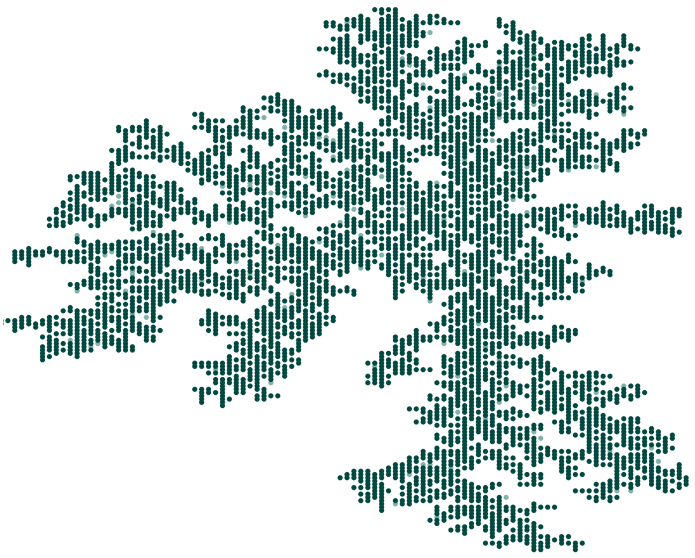

Figure 28. Running the simple algorithm for `freezing` = 0.05, `drying` = 0.01. It took 24.81 minutes for this to generate. All 5000 particles stuck.

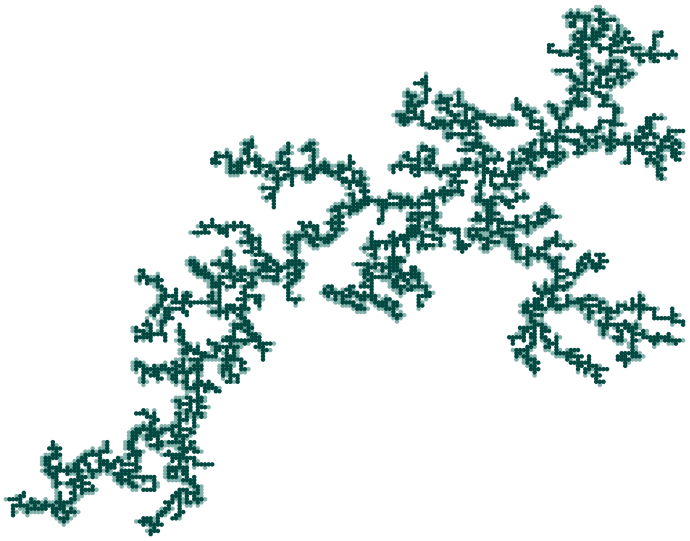

Figure 29. Running the simple algorithm for `freezing` = 0.5, `drying` = 0.5. It took 53.76 minutes for this to generate. Almost all of the particles ended up sticking - only 4832, however, did. This was due to the aggregate growing to the point where it was too large for the algorithm to continue.

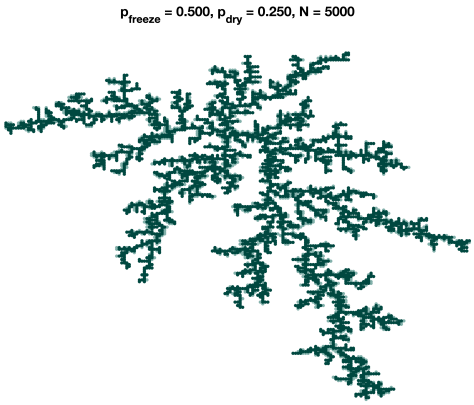


Figure 31. Running the simple algorithm for `freezing` = 0.5, `drying` = 0.25. It took 44.83 minutes for this to generate. All 5000 particles stuck.

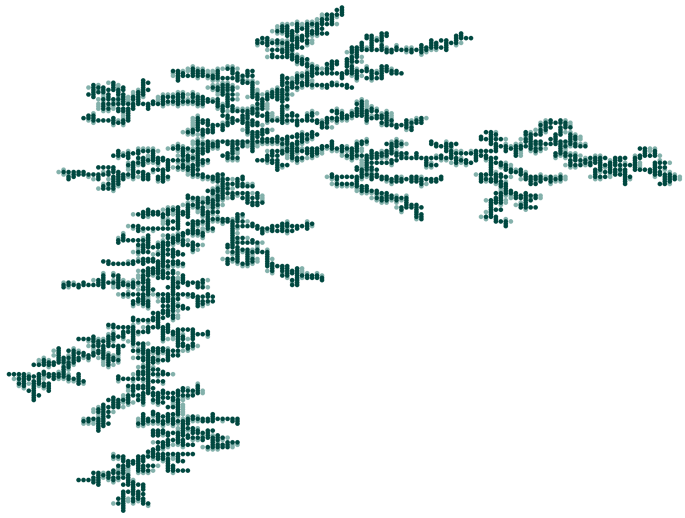

Figure 30. Running the simple algorithm for `freezing` = 0.5, `drying` = 0.375. It took 38.87 minutes for this to generate. Only 3703 of the 5000 particles ended up sticking. This was due to the size of the aggregate at the bottom; it reached the outer edge of the domain and triggered the abort statement.

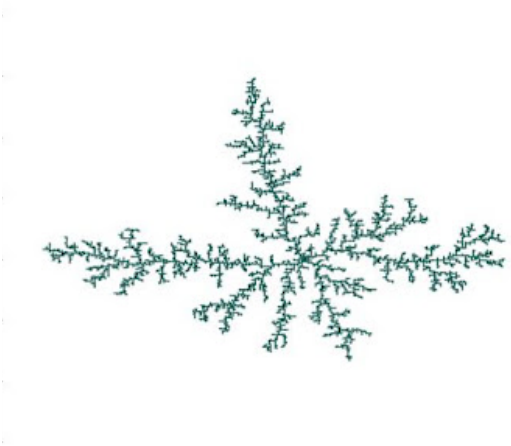

Figure 32. Running the vectorized algorithm for `freezing` = 1, `drying` = 0. It took 4.60 minutes for this to generate. All 5000 particles stuck.

Figure 33. Running the vectorized algorithm for `freezing` = 0.25, `drying` = 0.125. It took 9.60 minutes for this to generate. All 5000 particles stuck.

Figure 34. Running the vectorized algorithm for `freezing` = 0.05, `drying` = 0.001. It took 54.60 minutes for this to generate. All 5000 particles stuck.

Figure 35. Running the vectorized algorithm for `freezing` = 0.5, `drying` = 0.5. It took 4.50 minutes for this to generate. All 5000 particles stuck.

Figure 36. Running the vectorized algorithm for `freezing` = 0.5, `drying` = 0.375. It took 4.90 minutes for this to generate. All 5000 particles stuck.

Figure 37. Running the vectorized algorithm for `freezing` = 0.5, `drying` = 0.25. It took 5.90 minutes for this to generate. All 5000 particles stuck.

## REFERENCES


[1] S. Nath, S. F. Ahmadi, and J. B. Boreyko, "A review of condensation frosting," *Nanoscale and Microscale Thermophysical Engineering*, vol. 21, no. 2, pp. 81–101, 2017. [Online]. Available: https://doi.org/10.1080/15567265.2016.1256007

[2] L. Hauer, W. Wong, A. Sharifi-Aghili, L. Kondic, and D. Vollmer, "Frost spreading and pattern formation on microstructured surfaces," *Physical Review*, vol. 104, pp. 1–7, 2021. [Online]. Available: https://journals.aps.org/pre/pdf/10.1103/PhysRevE.104.044901

[3] T. J. Witten and L. Sander, "Diffusion-limited aggregation, a kinetic critical phenomenon," *Physical Review Letters*, vol. 47, no. 9, pp. 1400–1403, 1981. [Online]. Available: https://journals.aps.org/prl/pdf/10.1103/PhysRevLett.47.1400

[4] S. Martineau, "Directed diffusion-limited aggregation," *Latin American Journal of Probability and Statistics*, vol. 14, pp. 249–270, 2017. [Online]. Available: https://alea.impa.br/articles/v14/14-15.pdf

[5] P. Borke, "Dla - diffusion limited aggregation," online, 2014. [Online]. Available: http://paulbourke.net/fractals/dla/

[6] M. Richmond, "Random walks," online. [Online]. Available: http://star-www.dur.ac.uk/~tt/MSc/

[7] J. Adriazola, B. Gu, L. Cummings, and L. Kondic, "Simulating shear-thinning hele-shaw flow," 2022.