AEP 4380 Final Project: Diffusion-Limited Aggregation

Gregory Kaiser

December 16th, 2019

1 Introduction

Diffusion limited aggregation (DLA) is an algorithm which simulates the collection of free moving particles into a solid structure. It is relevant to the study of a variety of physical phenomena including lichen growth, Lichtenberg figures, and snowflake growth.[2][10] Essentially, particles randomly decide their movements as time passes, and then either attach to an existing aggregate in the case of ice crystals, or carve a path forward towards a lower potential in the case of a Lichtenberg figure.[14]

The inspiration for this project comes mainly from an interest in visually interesting physical simulations. Due to the weather in Ithaca, NY, we are surrounded by examples of water ice growth in many forms. Most obviously, snow crystals collect in the atmosphere to create large aggregates that can resemble a DLA-style structure[4][10].

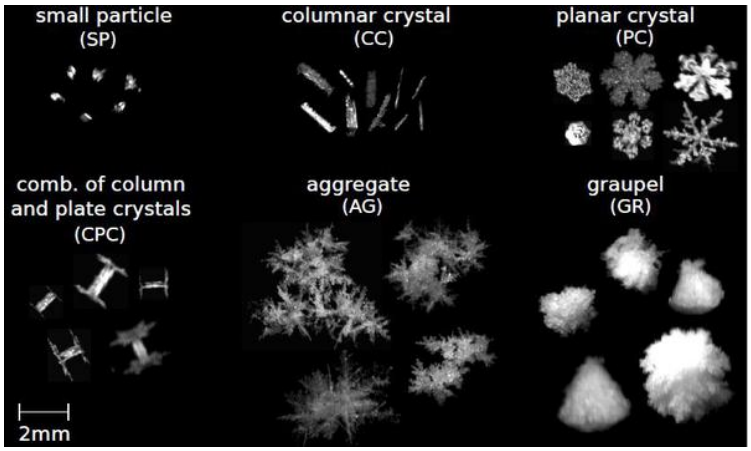

Figure 1: Images taken by Praz et. al. classifying classes of snowflake "hydrometeors."[10]

While each unit snowflake has a more complicated and geometrically restricted formation, the higher order aggregate of these snowflakes can resemble a DLA-type shape.

Dielectric breakdown is also relevant to this DLA algorithm, since the pattern formed during discharge through insulators have fractal properties.[9]

In this project, a form of DLA is implemented on a square grid to simulate the growth of a crystal structure. Free particles undergo a random walk along this grid until they are adjacent to a

Figure 2: Left: Lighting is a dielectric breakdown of the air.[13] Right: Lichtenberg figures generated using a microwave transformer and a piece of wood. This is a popular subject of many project based YouTube videos.[14]

crystalline particle. The likelihood that a randomly walking particle sticks to an existing crystal structure is determined probabilistically. The likelihood of sticking is also increased if more crystal particles surround a free particle.

The structure of the aggregate is altered by changing these probabilities, and the fractal dimension of this aggregate is calculated and compared to other literature.

2 Methods

2.1 Random Walk

A random walk is a simple algorithm that approximates the diffusion of a single particle along a given dimension. In two dimensions, two random numbers are generated to determine where a particle will "step". Since any given particle must move from where it is and is constrained in this algorithm to a square grid in two dimensions, there are 8 possible movements for a single particle.

If the particle begins at the center of a grid, it will slowly walk outwards to the edge. After n steps, a single particle will move an average distance of $a\sqrt{n}$ from its original location in each dimension, where a is the length of a lattice constant.[8]

This algorithm resembles Brownian motion, the random movement of small particles due to collisions with surrounding gas or liquid molecules.[12]

2.2 The Cloud

A cloud of N particles is initialized with random positions between a range of values in the x and y directions. The particles were all initialized at once so that when simulating the random walk of any one given particle, it is constrained to move in a way that doesn't conflict with its neighbors. This is very similar to simulating the random walk of one particle at a time, since each random movement is generated one at a time, however each step is constrained to a certain set of allowed

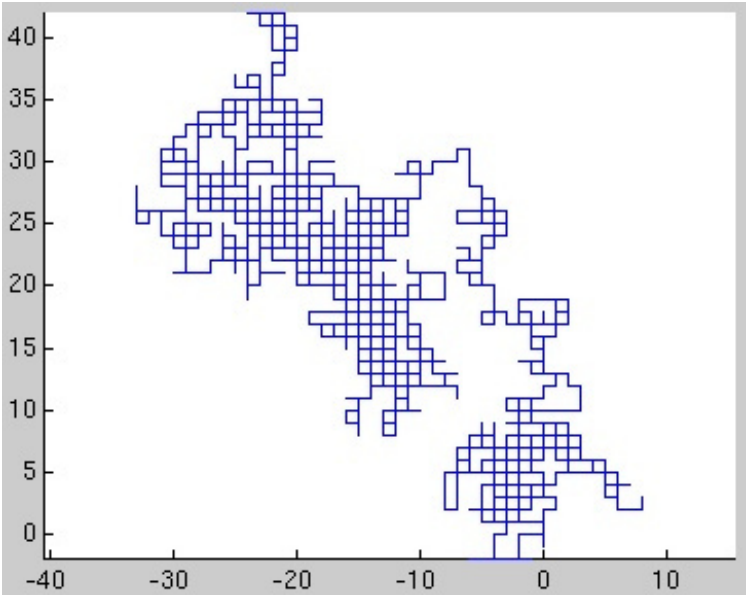

Figure 3: Example of a 2d random walk on a square lattice.[11]

moves.

This initialization used a struct with three fields for keeping track of each particle. Two integers store the position of the particle, and a third is a 0 or 1, noting that the particle is freely moving, or a crystal, respectively. The arrayt class is also used heavily for its convenience and bounds checking for debugging.[17]

The decision to simulate all of the particles walking at once is motivated by the prospect of animation, where the simulation of all particle positions is necessary and interesting. A video by Bruce Land from the Electrical and Computer Engineering Department at Cornell University shows the beauty of this idea by using a microcontroller and connected display.[3] Most DLA algorithms simulate a random walk one particle at a time, but in a real aggregate there is some density of particles which are slowly building onto the crystal structure.

A seed particle is initialized to allow the crystal to grow. For most runs, the seed was a single particle at the center of the field, but other seeds generated very interesting crystal structures as well.

At each time step, every particle has the option to move. To make sure that particle movements do not overlap, a simple method called pos_overlap checks if the position desired already exists as one of the particle positions. If there is an overlap, it returns the index of that particle.

To check if the particle is stuck, a method called is_stuck checks if there is overlap on all possible movement spots. If it is not stuck, random movements are generated until one of those valid movements is chosen. This is inefficient, since the random movements being generated are not always allowed, and another random movement must be found.

To check if a particle is adjacent to a crystal, the above position overlap algorithm is used in is_crystal_adjacent to find the index of each overlapping particle, and then that index is used to check whether or not the particle is a crystal. Each possible direction of overlap is noted as north, northeast, east, etc. This way the position being investigated is clear. The number of adjacent

crystals is also returned by this method, in order to weight crystallization of surrounded particles more.

Also, the random walkers are constrained to the x and y range originally specified, to keep them from wandering off to infinity.

After N_{steps} time steps, an aggregate of some form emerges as particles adhere to one another. The initial density of particles is determined by the limited range of x and y values, and the initial number of particles.

$$\rho_{approx} = \frac{N}{\Delta x \Delta y} \tag{1}$$

This density determines the initial rate of crystallization, and the depletion of particles local to the crystal causes branching and therefore occlusion of the inner parts of the structure.[2]

2.3 Sticking

When a particle randomly moves to a position adjacent to an existing crystal particle, another random number between 0 and 1, $rand_{stick}$, is generated which represents the random thermal energy of the particle. If this number exceeds a certain threshold, E_{thresh} the particle is allowed to continue moving, as opposed to becoming a crystal itself. The condition for crystallization in the vicinity of one crystal is:

$$rand_{stick} < E_{thresh} \tag{2}$$

If a particle is surrounded by n_{adj} adjacent crystal particles, the threshold value increases, making it easier for a particle to stick if surrounded by more than one crystal particle. In this way, aggregates with different densities can be generated. The condition for crystallization in the vicinity of a number of crystals is:

$$rand_{stick} < E_{thresh} n_{adj} \tag{3}$$

Because there is a higher probability of sticking when surrounded by more particles, the structures created using a low threshold value will look more densely populated. Essentially, the structure favors adding new particles to positions that are nestled nearby to many other particles. Parts of the crystal that are usually occluded by branches farther away are therefore made more available to particles which can get away with coming into contact with the crystal a few times while getting closer to the center.

Because particles are allowed to stick to one another along the diagonal directions, as well as horizontally/vertically, structures generated by this algorithm differ slightly from other more restricted models. This also affects the fractal dimension calculation.[15]

3 Results and Discussion

Running this DLA algorithm takes quite a while, since each particle is kept track of at each time step. However, this does not effect the overall structure of the aggregate, since most of the time is spent keeping track of the random movements of each freely moving particle.

Since the aggregate generated by a DLA algorithm resembles a fractal shape[6], a correlation between particles and aggregate size should be able to be drawn. The length of the branches, and the number of particles that make up that branch, should be related by the following expression:

$$N(r) \sim r^D \tag{4}$$

where $N(r)$ is the number of particles in the crystal structure within a circle of radius r, and D is some non-integer value representing the power law dimension of the aggregate.[9] Therefore, a plot of $log(N(r))$ versus $log(r^D)$ should produce a graph with a slope equivalent to the fractal dimension of the aggregate.

The distance is calculated by finding the integer-rounded distance from the seed particle at the center to a given particle's position upon crystallization. This then increments a value stored in a separate distance array. This keeps track of the number of particles that fall between one integer number of lattice constants from the center, and another.

In the end, each of the buckets representing a certain integer radius is added to the next, so that the distance array stores the number of particles that exist *at or below* a certain radius.

For all plots, $N = 5000$ particles simulated over a range of $\Delta x = \Delta y = 200$ lattice constants for $N_{steps} = 4000$ time steps. Keeping these values constant, the threshold value for sticking is lowered, producing an increasingly dense structure.

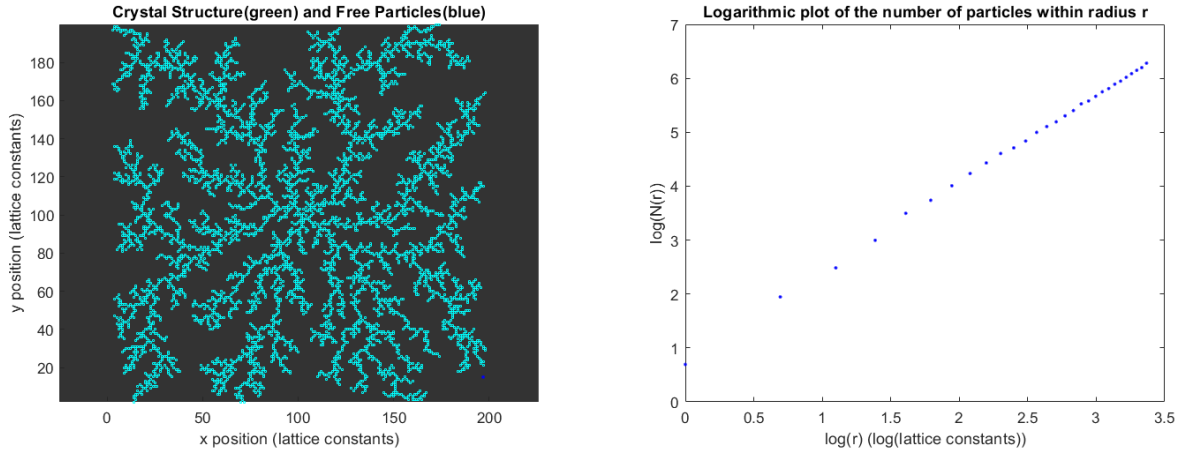

Figure 4: A thermal threshold value of 1.0 means that being adjacent to a crystal has a 100% probability of causing a particle to become a crystal as well. A power law dimension of 1.66 matches the literature closely.[9]

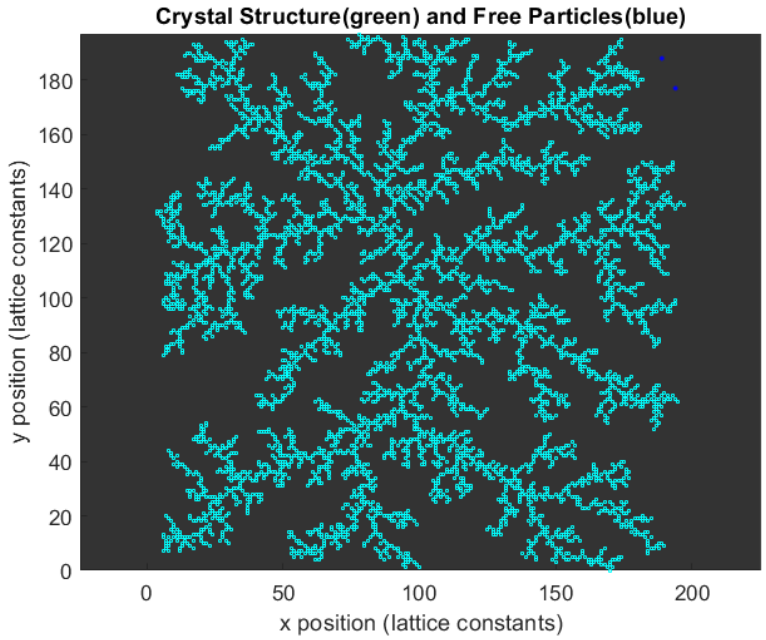

Figure 5: A thermal threshold value of 0.8 closely resembles the 100% sticking case. A fractal dimension of 1.67 is found.

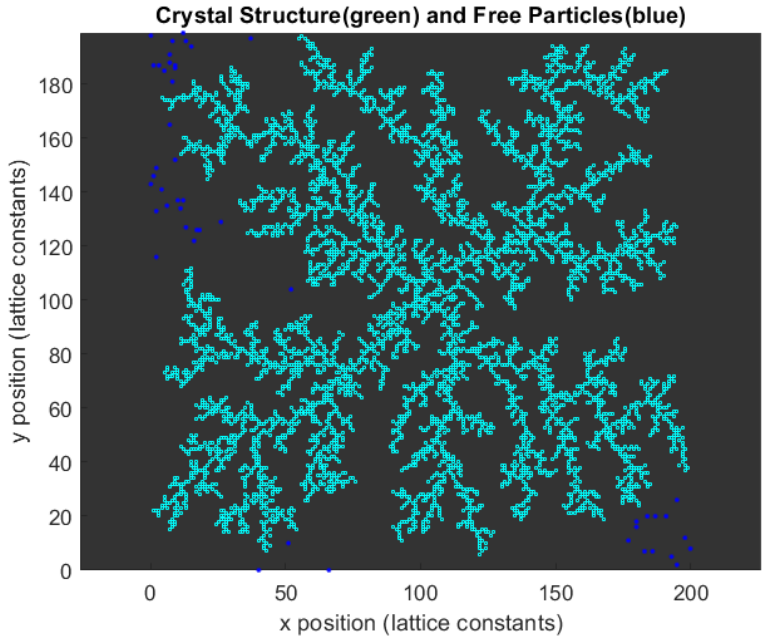

Figure 6: A thermal threshold value of 0.6 shows some slightly higher density, as expected. This has a fractal dimension of approximately 1.52.

The trend seems to be that the fractal dimension goes down as a function of sticking crystal density, against intuition. Unfortunately, the method used for calculating the slope of the log-log graphs was quite crude, and Niemeyer et. al. suggest that the fractal dimension calculation is severely

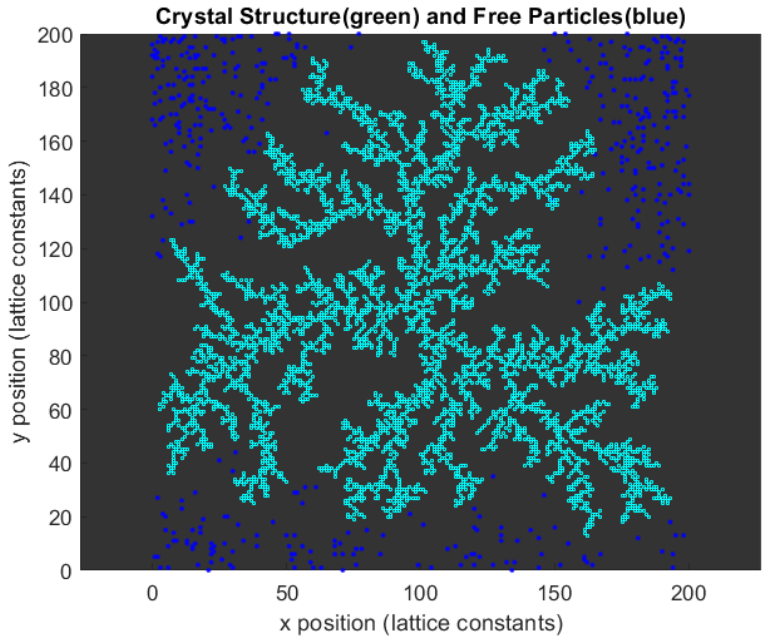

Figure 7: A thermal threshold value of 0.4 shows much higher density, and also has more free particles at the same number of simulation steps due to a lower sticking probability coefficient. This has a fractal dimension of approximately 1.67.

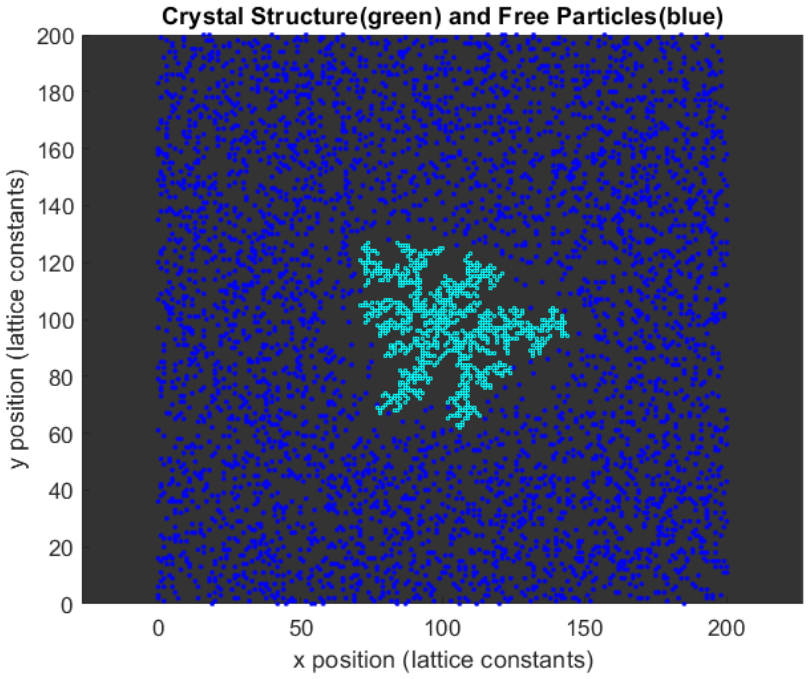

Figure 8: After only 1500 steps, a threshold value of 0.2 already shows signs of very high density. The fractal dimension is 1.36 at this point in the simulation, but comparison to other calculations without taking into account number of particles aggregated might not be meaningful.

limited by the square grid and finite number of particles. This is noted for further investigation at a later time.

As a function of time, the aggregate grows and depletes the surrounding cloud as the crystal takes up more space on screen.

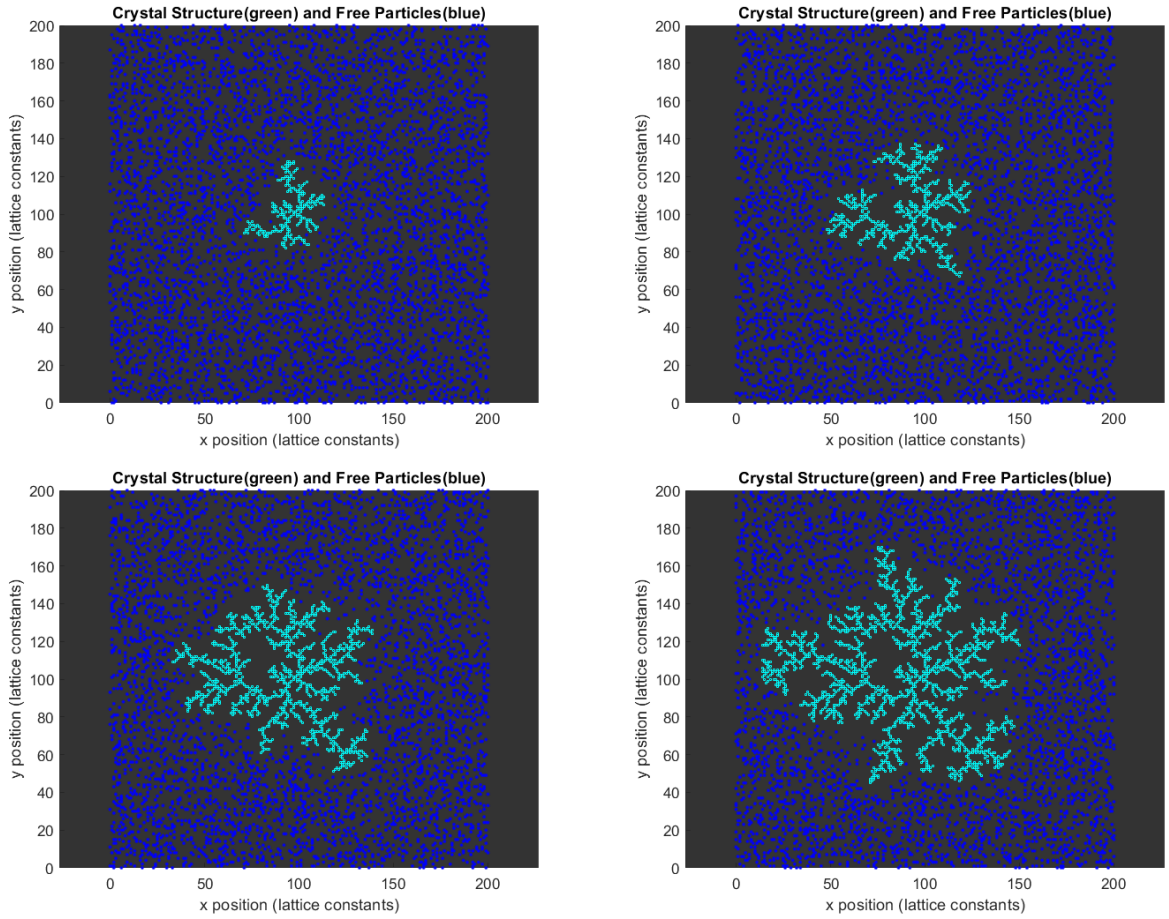

Figure 9: The aggregate at 200, 400, 600 and 800 simulation steps, with 100% sticking probability and 5000 initial particles.

This set of images, with a few more, was made into an animated gif using an online tool.[18]

While a central seed resembles many phenomena, such as a point source, or a single random event, crystals sometimes form in sheets.

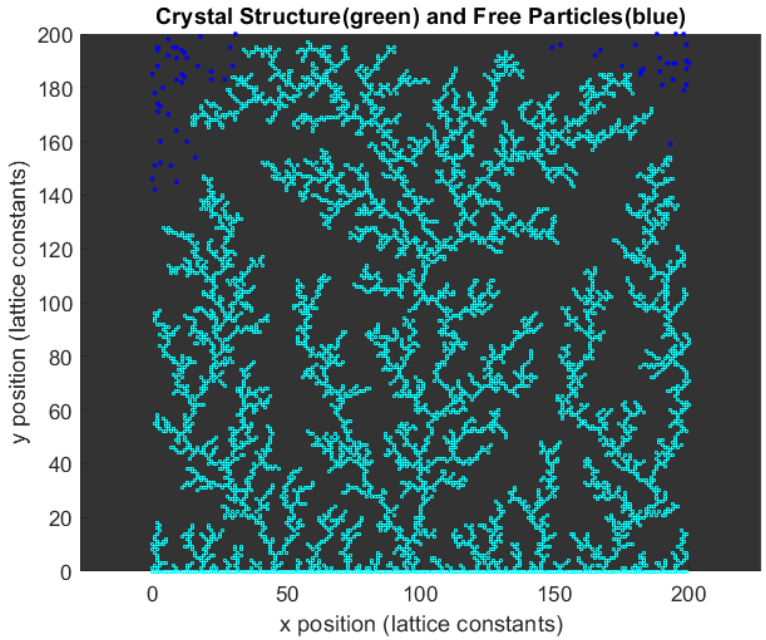

Figure 10: An aggregate formed using a starting crystal sheet and 100% sticking probability.

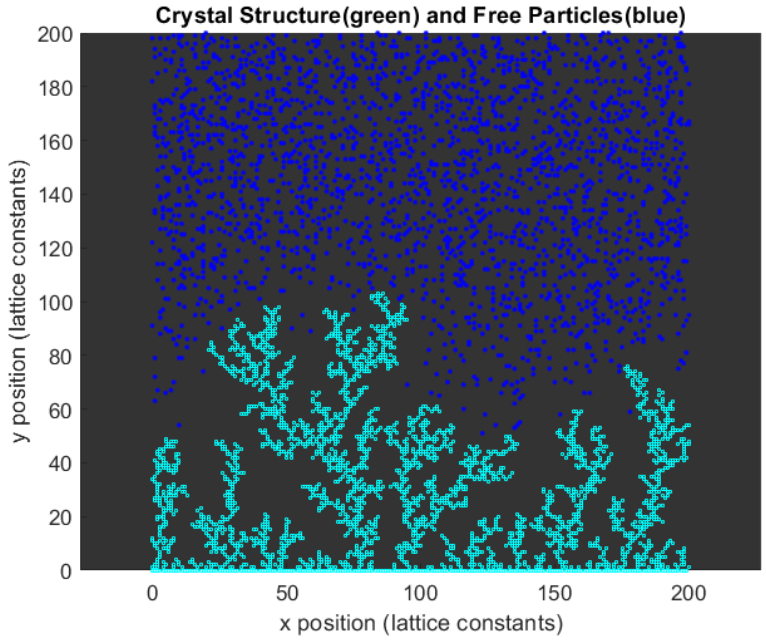

Figure 11: A denser crystal formed with a starting crystal of a sheet and sticking threshold 0.4.

For the aggregation of particles to a crystal, this closely resembles the physical picture, as particles

randomly wander towards the crystal after some time. In the case of dielectric breakdown, Niemeyer et. al. use a different model, which places a "seed" of zero potential, $\phi = 0$ at the center, and $\phi = 1$ in a circle at some distance away. Then the pattern grows to the next time step based on a local approximation of the electric field.[9]

While both methods involve a probabilistically determined parameter, and while each generate a similar fractal pattern, each model is quite physically distinct.

The model used in this report will also show symptoms of being performed on a square grid, at large enough distance from the origin, as shown by André Offringa[15] and Kim et. al[16]. This indicates that the simulation could be performed off-grid for a more physically accurate picture.[15]

One possible way to make the geometry of this fractal resemble more symmetrically shaped ice crystals would be to simulate not only the Brownian motion of a water molecule or small ice particle, but also to simulate its orientation in space.[4] This would require close analysis of the bond angle of H_2O and also random changes in its orientation.

If simulated off-grid, however, one could simply accept/reject the correct orientation probabilistically, and allow a molecule to click into the correct orientation.

As mentioned previously, the fact that particles can attach themselves diagonally is not precisely physical, since particles can become a crystal at different distances to existing crystal structures. This has an effect on the overall shape, and could be used as a way to bias specific geometries in the future.

4 Conclusions and Extensions

Implementing this algorithm with a cloud of particles, while it will be very useful for future animation, should be scrapped for the simulation of a large number of particles with low sticking coefficient. Because each particle is constrained by the particles surrounding it, free or crystal, this implementation scales poorly with N particles. Simulating each particle one at a time would decrease this time complexity immensely.

Clearly, DLA is a powerful algorithm due to its simplicity, but also due to its physical significance. At its root is a random number generator, itself an important tool in physical simulations due to the ubiquity of randomness in many natural phenomena. With some simple additions to this algorithm, energy minimization could simulate thermal equilibrium. Biasing the random walk could also allow modeling of bacteria behavior[20] or food-searching in slime molds.[19]

Because water has a more complicated set of restrictions on how it forms ice crystals, the formation of the classic-looking, hexagonal-looking ice crystals is harder to implement, and was not investigated fully in this project.

Geometric constraints on where particles are allowed to attach (for example, along a hexagonal lattice as opposed to a square one), can yield shapes that resemble water ice crystals. Additional details, like a water/ice equilibrium at the boundary of the crystal, can yield more interesting images.[7]

Taking into account crystal surface curvature also simulates water ice aggregation better, because a smoother surface is less likely to allow another particle to attach. Since higher curvature surfaces

are adhered to more often, the crystal tends to smooth out, as demonstrated by Tamás Vicsek.[5]

More care could also be taken to analyze the details of the phase transition/equilibrium between crystal and vapor since some energy must be released when a particle attaches itself to the surface.

Gravitational effects were neglected in this report, where some algorithms take into account a gravitational bias for the "cloud" of freely moving particles.[2] Also, the likelihood that some freely moving particles form a crystal on their without an adjacent seed could cause sticking of freely moving particles to happen faster. In combination with an associated mass for each particle, snow aggregates could spontaneously form and fall.

However, this plain DLA setup produced quite satisfying visuals, and opened the door to a number of possible paths for projects which take this idea to the next level. In fact, the connection between this type of DLA algorithm and statistical mechanics is hard to ignore. By allowing the particles to collide with one another, a variety of thermodynamic laws could be demonstrated.

References

[1] W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, *Numerical Recipes, The Art of Scientific Computing*. (3rd edit.), Cambridge Univ. Press, 2007, (ISBN 978-0-521-88068- 8, QA297 .N866 2007)

[2] Land, Bruce. *Adafruit TFT LCD Display Model 1480* section: "Diffusion-limited Aggregation." October 10th 2019. http://people.ece.cornell.edu/land/courses/ece4760/PIC32/index_TFT_display.html

[3] Land, Bruce. "Diffusion-limited Aggregation." YouTube video: Published July 9th, 2019, 4:26. https://www.youtube.com/watch?v=6DH-bSIQmHM

[4] Maruyama, Ken-ichi and Yasushi Fujiyoshi. "Monte Carlo Simulation of the Formation of Snowflakes." American Meteorological Society Journal of Atmospheric Science. Published online May 1 2005. https://journals.ametsoc.org/doi/pdf/10.1175/JAS3416.1

[5] Vicsek, Tamás. "Pattern Formation in Diffusion-Limited Aggregation." Physical Review Letters, December 1984. https://journals.aps.org/prl/pdf/10.1103/PhysRevLett.53.2281

[6] Witten and Sander, Physical Review Letters, Vol 47, 1981, p. 1400-1403. https://journals.aps.org/prl/pdf/10.1103/PhysRevLett.47.1400

[7] Gravner, Janko and David Griffeath. "Modeling snow crystal growth: a three-dimensional mesoscopic approach." October 30, 2008. http://psoup.math.wisc.edu/papers/h3l.pdf

[8] Weisstein, Eric W. "Random Walk–2-Dimensional." From MathWorld–A Wolfram Web Resource. http://mathworld.wolfram.com/RandomWalk2-Dimensional.html

[9] L. Niemeyer, L. Pietronero, and H. J. Wiesmann. *Fractal Dimension of Dielectric Breakdown* Physical Review Letters, Volume 52 Number 12, 19 March 1984. https://journals.aps.org/prl/pdf/10.1103/PhysRevLett.52.1033

[10] Praz, Christophe, Yves-Alain Roulet, Alexis Berne. "Solid hydrometeor classification and riming degree estimation from pictures collected with a Multi-Angle Snowflake

Camera." Atmospheric Measurement Techniques, March 2017. https://www.atmos-meas-tech.net/10/1335/2017/amt-10-1335-2017.pdf

[11] M2-TUM, Fabian Hansch, Stefanie Schetter. "Symmetric 2D Random Walk." Scientific/Educational MATLAB Database. http://m2matlabdb.ma.tum.de/example.jpg?MP_ID=309

[12] web.mit.edu "Brownian motion and random walks." An educational page on Brownian motion and random walks. http://web.mit.edu/8.334/www/grades/projects/projects17/OscarMickelin/brownian.html

[13] Ryan Fowler Photography (ABC Open contributor). Article by Carol Rääbus. "How to stay safe from lightning strikes during a storm." ABC News Posted 29 Nov 2017. https://www.abc.net.au/news/image/7143660-3x2-940x627.jpg

[14] Lejf Diecks. "HowTo: Make Lichtenberg figures using microwave transformer and safety precautions." YouTube video 15:31. Published Feb 7, 2017. Original Video: https://www.youtube.com/watch?v=jHOJSxYEh5s Image source: https://i.ytimg.com/vi/jHOJSxYEh5s/maxresdefault.jpg

[15] Offringa, André. "Diffusion Limited Aggregation." https://www.astro.rug.nl/ offringa/Diffusion%20Limited%20Aggregation.pdf

[16] Kim, Theodore, Michael Henson, and Ming C. Lin "A Hybrid Algorithm for Modeling Ice Formation." ACM SIGGRAPH Symposium on Computer Animation (2004). http://gamma.cs.unc.edu/HYBICE/hybrid_ice.pdf

[17] Kirkland, Earl. *Array Class Objects in C/C++ for Vectors and Matrices* AEP 4380 Fall 2019. https://courses.cit.cornell.edu/aep4380/secure/arrayt.hpp

[18] EZgif.com. Online GIF maker and image editor. https://ezgif.com/maker/ezgif-1-c3b23867-gif

[19] Will Stevens. "Physarum polycephalum 1" YouTube Video, 8:54. https://www.youtube.com/watch?v=mvBSkt6LhJE&feature=emb_logo

[20] Ribosome Studio "Chemotaxis: Bacteria attracted by a sugar crystal" YouTube Video, 0:17. https://www.youtube.com/watch?v=F6QMU3KD7zw

Source Code

```
/*  AEP 4380 Final Project
    Diffusion Limited Aggregation
    (and possible snowflake formation)

    Run on Windows core i7 with gcc version 8.2.0 (MinGW.org GCC-8.2.0-3)

    Gregory Kaiser December 16 2019

*/
#include <cstdlib>
#include <cmath>
#include <iostream>
#include <fstream>
#include <iomanip>
//Numerical Recipes type definitions for use of rendom number generator
#include "nr3.h"
//Numerical Recipes RNG
#include "ran.h"
//#define ARRAYT_BOUNDS_CHECK
//from class website -
//Kirkland, E: https://courses.cit.cornell.edu/aep4380/secure/arrayt.hpp
#include "arrayt.hpp"


Ullong seed = 20938475;//time(NULL);

using namespace std; //makes writing code easier

//struct for particle position and crystal state
struct particle{
    int xpos;
    int ypos;
    int is_crystal;
};

int pos_overlap(arrayt<particle>&, int, int);
int is_stuck(arrayt<particle> &, particle &);
int is_crystal_adjacent(arrayt<particle> &, particle &);

int main(){

    ofstream fp; //output file for crystal and free particle data
    fp.precision(9);

    fp.open("fp19_1.dat");
    if(fp.fail()){
        cout<<"cannotopenfile"<<endl;
        return(EXIT_SUCCESS); //defined by standard library
    }

    ofstream fp2; //output file for table
    fp2.precision(9);

    fp2.open("fp19_rdata.dat");
    if(fp2.fail()){
        cout<<"cannotopenfile"<<endl;
```

```
    return(EXIT_SUCCESS); //defined by standard library
}
//Assignment
Ranq1 myrand = Ranq1(seed);
int num_part = 5000, randx, randy, i, j;
int sim_steps = 4000, num_steps = 0;
int xrange = 200, yrange = 200;

arrayt<particle> particles(num_part);
arrayt<int> distances(xrange);
for(i=0;i<distances.n1();i++){
    distances(i)=0;
}
//seed a crystal
//point seed
i=0;
particles(i).xpos = xrange/2;
particles(i).ypos = yrange/2;
particles(i).is_crystal = 1;
i++;
// //sheet seed
// for(i=0;i<200;i++){
//     particles(i).xpos = i;
//     particles(i).ypos = 0;
//     particles(i).is_crystal = 1;
// }
//initialize all the rest of the particles
int chosen=0,bad;
while(i<num_part){
    //generate random positions x and y
    randx = floor(myrand.doub()*xrange);
    randy = floor(myrand.doub()*yrange);
    j=i; bad=0;
    while(j>=0&&!bad){//check that they don't overlap with previous positions
        if(randx==particles(j).xpos&&randy==particles(j).ypos){
            bad=1;//reject this initialization
        }
        j--;
    }
    if(!bad){//this is a good set of random positions
        particles(i).xpos = randx;
        particles(i).ypos = randy;
        particles(i).is_crystal = 0;
        i++;//move onto next particle
    }
}

int maxx = 0, maxy = 0;
int overlap_i, in_range, adj_crys, curx, cury, dist, randmove;
double rand_therm, therm_thresh=.8;
//move each particle to a random position that doesn't overlap
while(num_steps<sim_steps){
    i=0;
    while(i<num_part){//generates a valid movement for particle i if it can move
        if(particles(i).is_crystal==0&&!is_stuck(particles, particles(i))){//moveable particle
            //generate random movement
            // randx = floor(myrand.doub()*3)-1;
            // randy = floor(myrand.doub()*3)-1;
            randmove = floor(myrand.doub()*9);
```

```
switch(randmove){
    case 0://north
        randx = 0;
        randy = 1;
        break;
    case 1://northeast
        randx = 1;
        randy = 1;
        break;
    case 2://east
        randx = 1;
        randy = 0;
        break;
    case 3://southeast
        randx = 1;
        randy = -1;
        break;
    case 4://south
        randx = 0;
        randy = -1;
        break;
    case 5://southwest
        randx = -1;
        randy = -1;
        break;
    case 6://west
        randx = -1;
        randy = 0;
        break;
    case 7://northwest
        randx = -1;
        randy = 1;
        break;
}
curx = particles(i).xpos;
cury = particles(i).ypos;
//check overlap
overlap_i = pos_overlap(particles, curx+randx, cury+randy);
//check if the new position is in range
if(((curx+randx)>xrange)||((curx+randx)<0)||((cury+randy)>yrange)||((cury+randy)<0)){
    in_range = 0;
}
else{
    in_range = 1;
}
//allow or reject
if(overlap_i<0&&in_range){//this random movement doesn't overlap and is in range, allow it
    particles(i).xpos += randx;
    particles(i).ypos += randy;
    //check if it is now adjacent to a crystal
    adj_crys = is_crystal_adjacent(particles, particles(i));
    if(adj_crys>0){
        //if adjacent to a crystal, probabilisitcally allow it to become a crystal
        rand_therm = myrand.doub();
        if(rand_therm<therm_thresh*adj_crys){
            particles(i).is_crystal = 1;
            //update max x and y
            if(particles(i).xpos-particles(0).xpos>maxx){
                maxx = particles(i).xpos-particles(0).xpos;
```

```
                            }
                            if(particles(i).ypos-particles(0).ypos>maxy){
                                maxy = particles(i).ypos-particles(0).ypos;
                            }
                            //add to the plot of particles at a certain distance
                            dist =
                            sqrt((particles(0).xpos-particles(i).xpos)*(particles(0).xpos-particles(i).xpos)
                            +(particles(0).ypos-particles(i).ypos)*(particles(0).ypos-particles(i).ypos));
                            distances(dist) = distances(dist)+1;
                            //cout<<"stick"<<endl;
                        }
                    }
                    i++;//move along to the next particle
                }
                else{//there is overlap with an existing particle
                    //don't increment i (this particle needs a different random move)
                }
            }
            else{//this is a crystal, or this particle is stuck, just move on
                i++;
            }
        }
        num_steps++;
        if(num_steps%20==0){
            cout<<num_steps<<endl;
        }
    }
    for(i=0;i<num_part;i++){
        fp<<particles(i).xpos<<setw(15)<<particles(i).ypos<<setw(15)<<
            particles(i).is_crystal<<setw(15)<<i<<setw(15)<<sim_steps<<endl;
    }
    fp2<<distances(0)<<setw(15)<<0<<endl;
    for(i=1;i<distances.n1();i++){
        distances(i)+=distances(i-1);//add the number of particles in lower radii to this one
        fp2<<distances(i)<<setw(15)<<i<<endl;
    }
    cout<<maxx<<setw(10)<<maxy<<endl;
    fp.close();
    fp2.close();
    return(EXIT_SUCCESS);
} //end main

//checks if the position entered already exists
//returns the integer index of the existing overlapping particle
//or returns -1 if no such particle exists
int pos_overlap(arrayt<particle> &existing, int x, int y){
    int num = existing.n1(),j=0,exists=0;
    while(j<num&&!exists){//check that they don't overlap with previous positions
        if(x==existing(j).xpos&&y==existing(j).ypos){
            exists=1;//rejected!
            return j;
        }
        j++;
    }
    //if exists==0, the position doesn't overlap
    return -1;
}

//checks if a particle is adjacent to a crystal structure
```

```
//returns number of adjacent crystals
int is_crystal_adjacent(arrayt<particle> &existing, particle &check){
    int posx = check.xpos, posy = check.ypos, adjacent=0;
    //find the particles which are adjacent to this one (grid)
    int north = pos_overlap(existing, posx, posy+1);
    int northeast = pos_overlap(existing, posx+1, posy+1);
    int east = pos_overlap(existing, posx+1, posy);
    int southeast = pos_overlap(existing, posx+1, posy-1);
    int south = pos_overlap(existing, posx, posy-1);
    int southwest = pos_overlap(existing, posx-1, posy-1);
    int west = pos_overlap(existing, posx-1, posy);
    int northwest = pos_overlap(existing, posx-1, posy+1);

    //if adjacent exists and is a crystal, true
    if(!(north<0)&&existing(north).is_crystal==1){
        adjacent++;
    }
    else if(!(northeast<0)&&existing(northeast).is_crystal==1){
        adjacent++;
    }
    else if(!(east<0)&&existing(east).is_crystal==1){
        adjacent++;
    }
    else if(!(southeast<0)&&existing(southeast).is_crystal==1){
        adjacent++;
    }
    else if(!(south<0)&&existing(south).is_crystal==1){
        adjacent++;
    }
    else if(!(southwest<0)&&existing(southwest).is_crystal==1){
        adjacent++;
    }
    else if(!(west<0)&&existing(west).is_crystal==1){
        adjacent++;
    }
    else if(!(northwest<0)&&existing(northwest).is_crystal==1){
        adjacent++;
    }

    return adjacent;

}

//tells if the particle fed is stuck or not
//returns 1 if stuck (surrounded by other particles)
//returns 0 else
int is_stuck(arrayt<particle> &existing, particle &check){
    int posx = check.xpos, posy = check.ypos;
    //find the particles which are adjacent to this one (grid)
    int north = pos_overlap(existing, posx, posy+1);
    int northeast = pos_overlap(existing, posx+1, posy+1);
    int east = pos_overlap(existing, posx+1, posy);
    int southeast = pos_overlap(existing, posx+1, posy-1);
    int south = pos_overlap(existing, posx, posy-1);
    int southwest = pos_overlap(existing, posx-1, posy-1);
    int west = pos_overlap(existing, posx-1, posy);
    int northwest = pos_overlap(existing, posx-1, posy+1);

    //if all exist, this particle is stuck
```

```
    if(north>=0&&northeast>=0&&east>=0&&southeast>=0&&south>=0&&southwest>=0&&west>=0&&northwest>=0){
        return 1;
    }
    else{
        return 0;
    }
}

%MATLAB SCRIPT FOR PARSING DATA
load 'fp19_1.dat'
load 'fp19_rdata.dat'
xpos = fp19_1(:,1);
ypos = fp19_1(:,2);
crystal = fp19_1(:,3);
m = length(xpos);
figure
hold on
axis equal
set(gca,'Color',[.2 .2 .2])
for(i=1:m)
    if crystal(i)==0
        plot(xpos(i),ypos(i),'Color', [0 0 1], 'Marker', '.');
    else
        plot(xpos(i),ypos(i),'Color', [0 1 1], 'Marker', 's', 'MarkerSize',2.5);
    end
end
%title('Crystal Structure')
title('Crystal Structure(green) and Free Particles(blue)')
xlabel('x position (lattice constants)')
ylabel('y position (lattice constants)')
%% fractal dim
figure
size=50;
r = fp19_rdata(1:size,2);
nums = fp19_rdata(1:size,1);
slope = (log(nums(size,1))-log(nums(2,1)))/(log(r(size,1))-log(r(2,1)))
plot(log(r),log(nums),'b.')
title('Logarithmic plot of the number of particles within radius r')
xlabel('log(r) (log(lattice constants))')
ylabel('log(N(r))')
```