https://github.gatech.edu/ntian30/

6730-Project-Snowflake-Generation-using-Cellular-Automata

MODELING OF SNOWFLAKE FORMATION WITH CELLULAR AUTOMATA

April 25, 2023

Chao-Wen Chang, Bowen Mu, Nianhan Tian, and Xiaohan Yu

Georgia Institute of Technology

ABSTRACT

Snowflake, also known as snow crystals, can display hexagonal symmetry while exhibiting a diverse range of forms [12]. The shape of snow crystals is influenced by the saturation and temperature of the growth environment [9]. The various shapes of snow crystals include dendrites, plates, column, needles and many others. These diverse range of shapes can also combine to create endless varieties of snowflakes. In this project, our goal is to build upon an existing cellular automate model that simulates hexagonal snowflake formation while varying parameters that are temperature- and saturation-dependent parameters. In this project, we aim to use CA to model the process of snowflake formation from a pure water system under different temperatures. By investigating the effects of temperature, we hope to reproduce the morphologies of snowflakes and gain a better understanding of this natural phenomenon.

1 Introduction and Project Description

Cellular automata (CA) are algorithms that describe the spatial and temporal evolution of complex systems by applying local switching rules to the discrete cells of a regular lattice. As chemical engineering students, we would like to connect CA with some applications in related fields. For example, it can be applied for modeling the structure within a whole casting or within its part. In the first case, the texture across the wall and changes in grain size are modeled [4], while in the second one – the morphology of individual crystals is predicted [13, 6, 16].

Crystallization is a complex phenomenon that has intrigued scientists for centuries, and the formation of snowflakes is one of the most beautiful natural phenomena. Understanding the process of snowflake formation has important implications in various fields. Most of the research for crystallization is conducted by experiments, which could be very time-consuming. Therefore, CA is a suitable tool for modeling the crystallization of snowflakes in a relatively quick manner.

The majority of cellular automata simulations focused on snowflake formation have been conducted under isothermal conditions or only for dendritic crystals. However, real-world snowflakes display a wide range of morphologies at different temperatures, as depicted in Figure 1 [11]. Thus, the aim of this project is to use cellular automata to simulate the process of snowflake formation in a pure water system while accounting for the impact of temperature on the formation process. By examining the influence of temperature, we aim to replicate the morphologies depicted in Figure 1 and advance our understanding of this natural phenomenon.

In Project Phase I, we established a basic cellular automata model for snowflake formation and analyzed how various update rules can affect the resulting shapes. In Project Phase II, we identified parameters that directly correlate changes in temperature to changes in shape and evaluated how these parameters can result in different shapes during the snowflake

formation process.

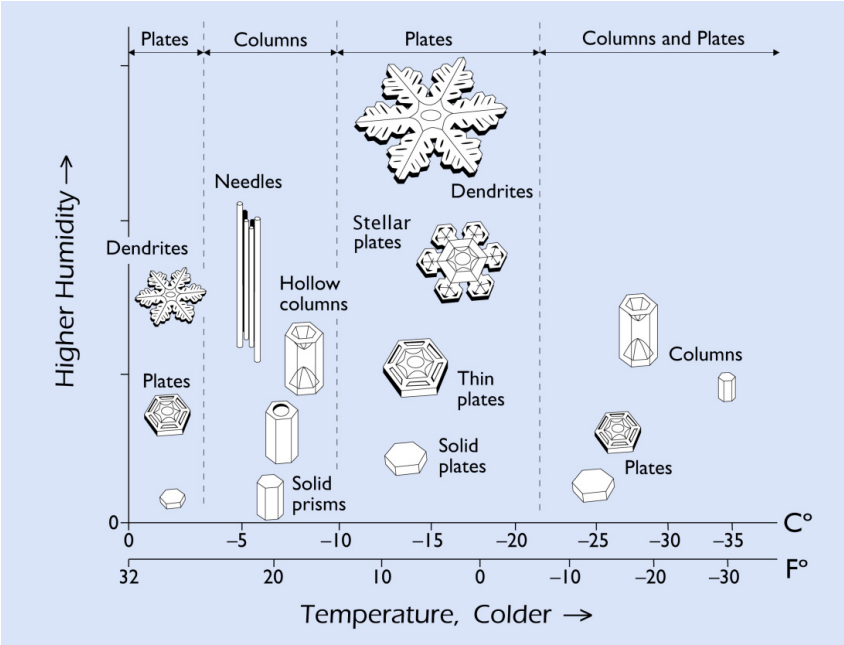

Figure 1: Snow Crystal Morphology Diagram [11]

2 LITERATURE SURVEY

2.1 Modeling Snowflake Formation

In [5], the growth of snowflakes is modeled by CA considering a lattice of hexagons where each cell has six neighborhoods. There are only two states for each cell, solid and liquid. The rules for the snowflake evolution are: If a cell is solid, it remains in this form in the next time step; if a cell is liquid, it becomes solid in the next time step if and only if one of its neighbors is solid currently. It is also possible to convert this hexagonal lattice into a typical rectangular lattice.

Zhao et. al. introduced a more complicated model where three states including solid, liquid, and near-solid are considered [15]. In addition, each cell is associated with a value between zero and one. The value indicates the state of the cell: A cell with value of one is solid; A cell with value less than one and has a solid neighbor is near-solid; Otherwise, a cell is liquid. The evolution rules for the values consider several factors including the state of the cell, the diffusion rate, and the states are changing due to the change of the values.

Liu et al. proposed a basic CA model for simulating snowflake formation. [8] Their model is based on the diffusion-limited aggregation (DLA) algorithm, which is commonly used for simulating the growth of dendritic structures. The parameters of the models were varied to simulate different crystallization rates and branch growth rates. The model is able to reproduce the dendritic structure of snowflakes observed in experiments.

2.2 Advanced Methods

In order to gain a deeper understanding of snow crystal growth, Xu and Wang showed a CA model for two-dimensional non-isothermal snowflake formation [14]. The new model combines vapor diffusion and heat diffusion Their model agrees with previous experimental

observations and satisfies the crystal growth dynamic theory. It is found that latent heat release has an impact on the small-scale structure of snow crystals, and this impact is more significant when vapor diffusion is slow and heat diffusion dominates.

While existing research on crystallization has used two-dimensional models to simulate snowflake formation, a study by Ning et al [10] examined the extension of those models to three dimensions. The three-dimensional study enabled the modeling of needle growth or capped columns which naturally occur in snowflake formation. This study used a three-dimensional crystalline structure of ice as the cellular arrangement. The cell-based framework is then updated through local rules, which entail detecting receptive cells and averaging data from non-receptive locations.

3 Description of the Snowflake Formation System

The system of snowflake formation is a complex process influenced by various environmental factors such as temperature, humidity, and pressure [9]. When water vapor in the air goes straight from a gas to a solid form, without passing through a liquid state, it results in formations that are referred to as snow crystals [7]. The crystallization of a snowflake involves the freezing of water vapor into ice crystals. The process begins when water vapor in the air comes into contact with tiny particles, such as dust or other impurities, that act as nucleation sites for ice crystal formation. Once the water vapor condenses onto the nucleation site, the individual ice crystals begin to grow as more water molecules attach to them. As the ice crystal grows, it develops six branches or arms, which are all connected at the center point. Each arm can branch further into smaller branches, creating a complex and symmetrical structure.

The final shape and size of the snowflake depend on various factors, such as temperature, humidity, and atmospheric pressure, which can affect the rate and manner in which water vapor freezes onto the crystal. The conditions under which the snowflake forms can also influence its texture, transparency, and overall appearance. At low temperatures, water vapor molecules tend to stick to the surface of ice crystals more readily, leading to slower growth rates and the formation of simpler crystal shapes such as plates or columns. At higher temperatures, the growth rate is faster, and the crystal shapes tend to be more complex, such as dendrites, needles, and stars [9]. High humidity levels can lead to the formation of larger, more intricate snowflakes, while low humidity levels can result in the formation of smaller, simpler crystals.

4 Conceptual Model of Project

Our study utilize two models, which were detailed in sections §5.1 and §5.2 to simulate the complex process of snowflake formation. The shape of snowflakes is influenced by several hyperparameters, including the diffusivity coefficient (α), which is directly impacted by temperature, and β and γ, which are indirectly linked to temperature and associated with cell humidity in both models. Accordingly, our final goal is to establish equations that establish a correlation between temperature changes and these hyperparameters, which ultimately lead to diverse shapes during snowflake formation.

5 Simulation and Results

We successfully reproduced two models using cellular automata from two papers.

5.1 3-Equation CA

This model is from Liu et al [8]. They considered three scenarios of a cell: 1) crystallized, 2) not crystallized but surrounded by or near crystallized cells, 3) not crystallized and no near crystallized cells. Three different equations are used respectively:

$$S_{V0}^{t+1} = S_{V0}^{t} + \gamma \tag{1}$$

$$S_{V0}^{t+1} = S_{V0}^{t} + \gamma + (S_{V1}^{t} + S_{V2}^{t} + S_{V3}^{t} + S_{V4}^{t} + S_{V5}^{t} + S_{V6}^{t})/12 \tag{2}$$

$$S_{V0}^{t+1} = S_{V0}^{t} * 0.5 + (S_{V1}^{t} + S_{V2}^{t} + S_{V3}^{t} + S_{V4}^{t} + S_{V5}^{t} + S_{V6}^{t})/12 \tag{3}$$

S_{V0} is the state of the cell of interest, and S_{V1} to S_{V6} are the states of the nearest neighbors of this cell. The initial density distribution is controlled by β, and γ controls the cellular capture of water.

We successfully generated different snowflakes using this model, as shown in Fig 2-4. By tuning the parameters, the shapes and sizes of the snowflakes vary. When β is larger, the initial density is higher. This leads to thicker branches in the snowflakes. When γ is larger, the speed of cells capturing water is higher. Denser snowflakes are generated, and branches are less likely to form.

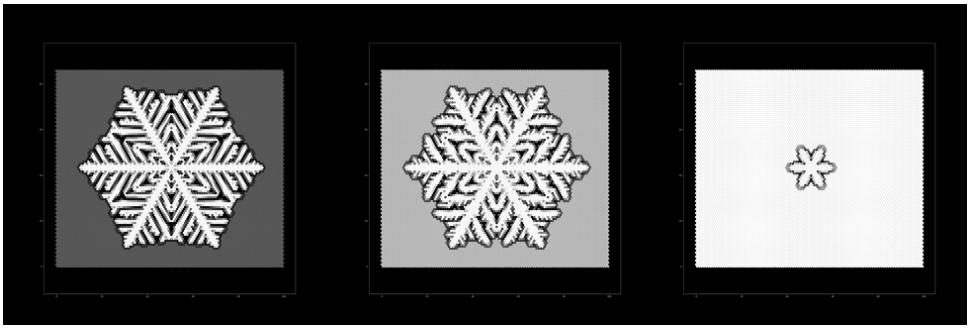

Figure 2: Snowflakes generated with $\beta = 0.3$. From left to right, γ takes $0.0001,\ 0.001,\ 0.01$.

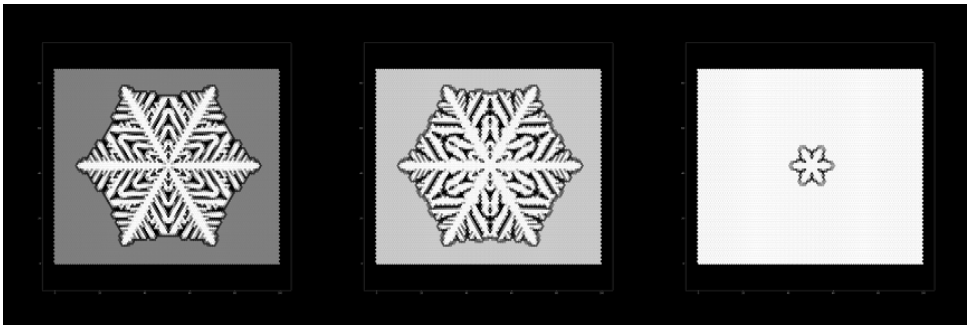

Figure 3: Snowflakes generated with $\beta = 0.5$. From left to right, γ takes $0.0001,\ 0.001,\ 0.01$.

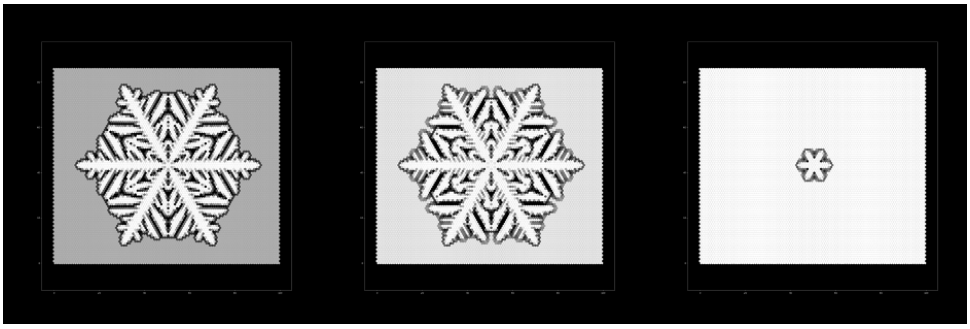

Figure 4: Snowflakes generated with $\beta = 0.7$. From left to right, γ takes $0.0001,\ 0.001,\ 0.01$.

The code for generating snowflakes using the 3-equation model can be found in $3-equation_CA.py$ in our Github repo. The following is the breakdown explanation of this algorithm.

```python
class three_equation_snowflake_generation:
    def __init__(self, data, N_X, N_Y, beta, gamma, timestep, T):
        self.data=data
        self.T = T
        self.N_X=N_X
        self.N_Y=N_Y
        self.beta=beta
        self.gamma=gamma
        self.break_while_flag=0
        self.timestep=timestep
        self.center_index=int(N_Y/2)*N_X+int(N_X/2)
        self.boundary=int(N_Y*0.4)

        self.current_state=0
        self.updated_state=0
        self.crystalized_flag_ring1=0
        self.crystalized_flag_ring2=0
        for i in self.data.keys():
            self.data[i]['state']=self.beta
        self.data[self.center_index]['state']=1
```

Figure 5: 3-equation Model Class Initialization in python

We first initialized the grid with the input size, beta, gamma, timestep and temperature, as shown in Fig 5. We also set the state of the central point to 1. In Fig 6, there are two helper functions. `if_crystalized_in_circle` takes in the indices of the nearest neighbors of a given cell, and then decides whether there are crystallized cells in the nearest neighbors. `decide_receptive` is a function to loop through all the cells to decide whether they are receptive cells (1) or non-receptive cells (0). If a cell is receptive, it means that the cell itself is crystallized or there are crystallized cells in its nearest neighbors.

```python
def if_crystalized_in_circle(self,circle_pos_list):
    crystalized_flag=0
    for i in circle_pos_list:
        if self.data[i]['state']>=1:
            crystalized_flag=1
            break
    return crystalized_flag

def decide_receptive(self):
    #1: non-receptive < 1
    #0: receptive >=1

    for i in self.data.keys():
        crystalized_flag=self.if_crystalized_in_circle(self.data[i]['ring1'])
        if self.data[i]['state']>=1:
            self.data[i]['receptive']=0
        elif crystalized_flag==1:
            self.data[i]['receptive']=0
        else:
            self.data[i]['receptive']=1
```

Figure 6: 3-equation Model helper functions in python

In Fig 7, we wrote three functions for three equations respectively. The output of the three functions are the updated states of the cell. We then updated the state of a given cell using its index as the input. We made decision of which update equation to use by looking at the cell itself, its nearest neighbors and also its second surrounding circle of cells.

```python
def update_equ_1(self):
    return self.current_state+self.gamma

def update_equ_2(self,ring1_index):
    self.updated_state=0
    for i in ring1_index:
        self.updated_state+=self.data[i]['state']*self.data[i]['receptive']/12
    self.updated_state=self.current_state+self.updated_state+self.gamma
    return self.updated_state

def update_equ_3(self,ring1_index):
    self.updated_state=0
    for i in ring1_index:
        self.updated_state+=self.data[i]['state']*self.data[i]['receptive']/12
    self.updated_state=self.current_state*0.3+self.updated_state
    return self.updated_state
```

Figure 7: Functions of 3 equations in python

```python
def update_center_3_equ(self,index):
    self.current_state=self.data[index]['state']

    #if center larger or equal to 1, remain crystalized
    if self.current_state>=1:
        self.updated_state=self.update_equ_1()
    #if center smaller than 1, not fully crystalized
    else:
        self.crystalized_flag_ring1=self.if_crystalized_in_circle(self.data[index]['ring1'])
        self.crystalized_flag_ring2=self.if_crystalized_in_circle(self.data[index]['ring2'])

        #crystalized cell in first circle
        if self.crystalized_flag_ring1==1:

            self.updated_state=self.update_equ_2(self.data[index]['ring1'])

        #elif crystal in second circle
        elif self.crystalized_flag_ring1==0 and self.crystalized_flag_ring2==1:
            self.updated_state=self.update_equ_3(self.data[index]['ring1'])
        #elif no crystal in 2 circles:
        else:
            self.updated_state=self.update_equ_1()
    return self.updated_state
```

Figure 8: 3-equation Model one time-step one cell update function in python

```python
def boundary_condition(self,center_pos,this_pos,threshold):
    #if the crystallized cell reaches the threshold, stop the generation
    #0: update it
    dist_to_center=sum((np.array(center_pos)-np.array(this_pos))**2)
    if dist_to_center>threshold**2:
        self.break_while_flag=1
    return self.break_while_flag


def generation_and_plot(self):
    time_count=0
    while time_count<self.timestep:
        #update the receptive property
        self.decide_receptive()

        updated_state_list=[]
        for j in self.data.keys():
            updated_state_list.append(self.update_center_3_equ(j))
        for j in self.data.keys():
            self.data[j]['state']=updated_state_list[j]
            if updated_state_list[j]>=1:
                self.break_while_flag=self.boundary_condition(self.data[self.center_index]['pos'],self.data[j]['pos'],self.boundary)
        time_count+=1


        if self.break_while_flag==1:
            break

    plt.style.use('dark_background')
    fig_width = 20
    fig, ax = plt.subplots(figsize=(fig_width, fig_width))

    for i in range(self.N_X*self.N_Y):
        hex = RegularPolygon((self.data[i]['pos'][0], self.data[i]['pos'][1]), numVertices=6, radius=1/np.sqrt(3),
                             orientation=np.radians(0),  facecolor = (1,1,1),
                             alpha=min(1,self.data[i]['state']))#, edgecolor='k')
        ax.add_patch(hex)
    ax.axis('equal')
    # plt.savefig('../image/3equation_T_{}.png'.format(self.T))
    # plt.savefig('../image/3equation_b_{}_g_{}.png'.format(self.beta, self.gamma))
```

Figure 9: 3-equation Model boundary condition and visualization in python

In Fig 9, we first set the boundary condition of the snowflake growth, which is the distance of the farthest crystallized cell to the center should be less or equal to 0.4 times the size of the grid. The in the function `generation_and_plot` we looped through the `update_center_3_equ` for all cells in the grids, and we did this the given timestep times. Then the final generated snowflake was visualized using the greyscale.

5.2 Reiter's model CA

This model is from [12]. The cells in the model are divided into receptive and nonreceptive cells, where the former are icy cells or cells with an icy neighbor, and the latter are the rest cells. Each cell is associated with a value. There are three parameters in the model including the background level β, the addition constant γ, and the diffusivity α. The values of cells are updated as follows:

For all the receptive cells, the values of the next time step will be the current value plus a diffusion term plus a constant, which is:

$$u(t+1,P) = u(t,P) + \frac{\alpha}{12}\left(-6u(t) + \sum_{N\in nn} u(t,N)\right) + C \tag{4}$$

Here, u is for the value of cell, P for the position, nn means nearest neighbours. The whole second term on the right hand side of the equation is the diffusion term. The non-receptive cell values are updated by

$$u(t+1,P) = u(t,P) + \frac{\alpha}{12}\left(-6u(t) + \sum_{N\in nn} u(t,N)\right) \tag{5}$$

By tuning the three parameters, we can get different shapes of snowflakes (see Figure 10-12). When γ takes larger values, each branch of the snowflake becomes wider. When β takes larger values, the snowflake looks more delicate.

The relevant codes for generating results by Reiter's model are as follows. Figure **??** shows the initial set up of the Reiter's model; Figure 14 shows the update of Reiter's model for one step; Figure 15 shows the automatic termination condition of Reiter's model; Figure 16 shows the whole update of Reiter's model combining one step's update and the automatic termination condition.

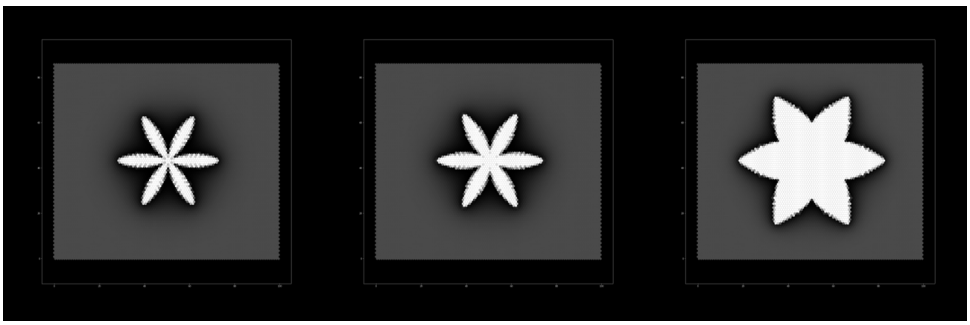

Figure 10: Snowflakes generated with $\alpha = 0.8$ and $\beta = 0.3$. From left to right, γ takes $0.0001,\ 0.001,\ 0.01$.

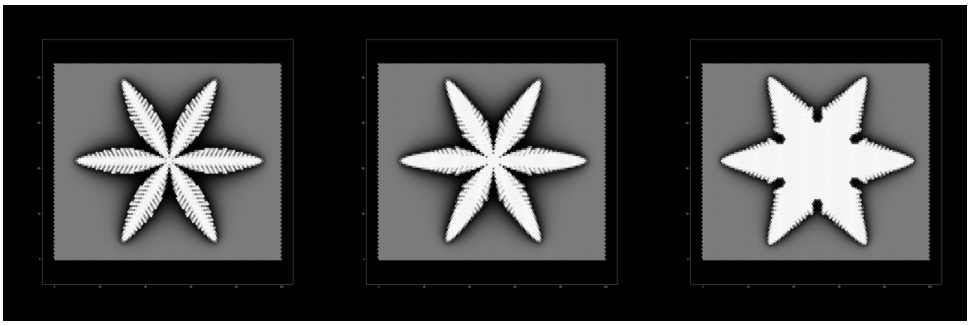

Figure 11: Snowflakes generated with $\alpha = 0.8$ and $\beta = 0.5$. From left to right, γ takes $0.0001,\ 0.001,\ 0.01$.

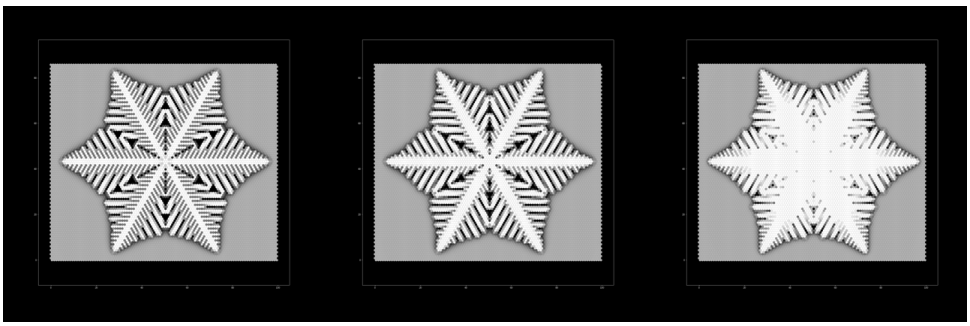

Figure 12: Snowflakes generated with $\alpha = 0.8$ and $\beta = 0.7$. From left to right, γ takes $0.0001,\ 0.001,\ 0.01$.

```python
class ReinerModel:
    def __init__(self, data, N_X, N_Y, alpha, beta, gamma):
        self.data=data
        self.N_X=N_X
        self.N_Y=N_Y
        self.alpha=alpha
        self.beta = beta
        self.gamma=gamma
        self.break_while_flag=0
        self.center_index=(N_Y+1)*N_X/2
        self.boundary=int(N_Y*0.4)-1

        for i in self.data.keys():
            self.data[i]['state']=beta
            self.data[i]['add_con_state'] = 0
            self.data[i]['nonreceptive_state'] = 0
            self.data[i]['avg_state'] = 0
        self.data[self.center_index]['state']=1
```

Figure 13: Initial set up of Reiter's model in python.

```python
    def update_one_step(self):
        for i in range(N_X * N_Y):
            # update the add constant state
            flag = 0
            if self.data[i]['state'] >= 1:
                flag = 1
            else:
                for j in self.data[i]['ring1']:
                    if self.data[j]['state'] >= 1:
                        flag = 1
                        break
            self.data[i]['add_con_state'] = flag * (self.data[i]['state']+self.gamma)

            # update the nonreceptive state
            if flag == 1:
                self.data[i]['nonreceptive_state'] = 0
            else:
                self.data[i]['nonreceptive_state'] = self.data[i]['state']

        for i in range(N_X * N_Y):
            # update the average state
            neighbor_add_up = sum(self.data[j]['nonreceptive_state'] for j in self.data[i]['ring1'])
            self.data[i]['avg_state'] = (1-0.5*self.alpha)*self.data[i]['nonreceptive_state'] + \
                self.alpha/(2*len(self.data[i]['ring1']))*neighbor_add_up

            # update the state
            self.data[i]['state'] = self.data[i]['add_con_state'] + self.data[i]['avg_state']
```

Figure 14: One step update of Reiter's model in python.

```python
def automatic_stop(self):
    flag = False
    boundary_li = []
    boundary_li.extend(range(self.N_Y))
    boundary_li.extend(range((self.N_X-1)*self.N_Y, self.N_X*self.N_Y))
    boundary_li.extend(range(0, self.N_X*self.N_Y, self.N_Y))
    boundary_li.extend(range(self.N_X-1, self.N_X*self.N_Y, self.N_Y))
    flag = True
    for i in boundary_li:
        if abs(self.data[i]['state']-self.beta)>0.01*self.beta:
            flag = False
            print(i)
            break
    return flag
```

Figure 15: Automatic termination of code in python.

```python
def update_n_step(self, n_iter=100):
    # for i in range(n_iter):
    #     self.update_one_step()

    while self.automatic_stop():
        self.update_one_step()

    # Some values are greater than 1, turn them to 1
    for i in range(N_X * N_Y):
        if self.data[i]['state'] > 1:
            self.data[i]['state'] = 1

    # make the plot
    plt.style.use('dark_background')
    fig_width = 20
    fig, ax = plt.subplots(figsize=(fig_width, fig_width))

    for i in range(self.N_X*self.N_Y):
        hex = RegularPolygon((self.data[i]['pos'][0], self.data[i]['pos'][1]),
                             numVertices=6, radius=1/np.sqrt(3),
                             orientation=np.radians(0),  facecolor = (1,1,1),
                             alpha=min(1,self.data[i]['state']))#, edgecolor='k')
        ax.add_patch(hex)
    ax.axis('equal')
    # plt.savefig('a_{}_b_{}_g{}.png'.format(self.alpha, self.beta, self.gamma))
```

Figure 16: The whole generation of Reiter's model results in python.

5.3 CA Grid Choice

When comparing our results with the papers, some snowflakes under the same parameters are not completely identical.

We developed an algorithm to generate and visualize hexagonal grids of any given size. The surrounding nearest neighbors (first- and second-nearest) and the states of the cells are stored in the dictionary. This is shown in Fig 17. The crystallization state of a grid is presented by the transparency of the grid.

```python
# make grid
def generateGrid(N_X, N_Y):
    # input: number of grids on x (horizontal) and y (vertical) direction
    # generate grids with spacings = 1
    X, Y = np.meshgrid(np.arange(N_X), np.arange(N_Y), sparse=False)
    X = X.astype(np.float64)
    Y = Y.astype(np.float64)
    Y = Y*np.sqrt(3)/2
    X[::2, :] += 1/2

    return X, Y




def genDict(X, Y):
    # return dictionary of each grid point containing positions, first- and second-nearest neighbor lists and crystallization states

    pos = [[x,y] for x, y in zip(X.flatten(), Y.flatten())] # list of grids
    tree = sk.KDTree(pos)
    idx_dist1 = tree.query_radius(pos, r=1.5)
    idx_dist2 = tree.query_radius(pos, r=2.5)


    for k in range(len(pos)):
        idx_dist1[k] = list(set(idx_dist1[k]) - set([k]))  # first ring
        idx_dist2[k] = list(set(idx_dist2[k]) - set(idx_dist1[k]) - set([k])) # second ring


    states = [0]*len(pos)
    d = {'pos': pos, 'ring1': idx_dist1, 'ring2': idx_dist2, 'state': states}
    df = pd.DataFrame(data=d)
    data = df.to_dict('index')

    return data
```

Figure 17: Hexagonal grid generation in Python.

We found a Github repo reproducing the snowflake formation CA models [3], and they have successfully reproduced the paper results. Under the same parameters (α, β, γ, number of iterations, the comparison of the 3-equation model and the GitHub model is shown in Fig 18.

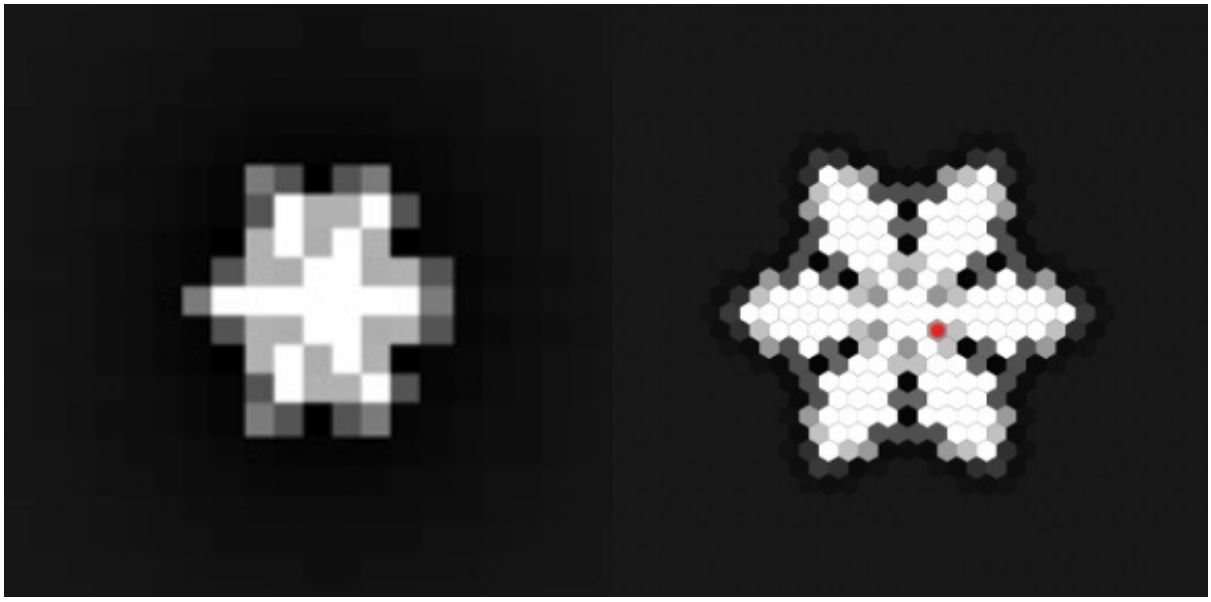

Figure 18: Snowflake generated using (left) square, (right) hexagonal grid

Our model resulted in a perfect symmetric snowflake, while the model from the Github repo generated a distorted snowflake. This is because they are using an ordinary square grid. The nearest neighbors of a given cell are not 6 cells but 8 cells. They need to consider odd/even rows when they are selecting the nearest neighbors of a given cell. This leads to asymmetry in snowflake generation. We observe that the problem went away when using very large timesteps.

Both the papers didn't specify the grid they used in the code, but the model from GitHub did reproduce the paper results. This indicates that the papers might use the square grid as well. We decided to go on with our hexagonal grid since this approach is more reasonable and a better representation of the real snowflake formation.

5.4 Experimental Results and Validation: The Influence of Temperature and Humidity

Under real-world situations, both the temperature and humidity of the environment hugely affect the formation of snowflakes. In this section, we derive the relationships between the temperature and humidity of the environment and model parameters to simulate the formation of snowflakes under realistic conditions, using existing formulas as well as our physics intuition. Then we validate the simulated output to shapes in Fig. 1 under various temperatures and humidity.

In Reiter's model, α is the water mass diffusivity, which is determined by temperature; β is the background level of the crystallinity state; γ captures the idea that some water may be available from outside the plane of growth. We think β and γ can be captured by using temperature and humidity values.

In the 3-equation model, there are only two parameters, β and γ, which have the same definition as in the Reiter's model.

We have to find a way to connect these parameters to temperature and humidity. We interpret α as the crystallization rate of ice. However, based on our literature scan, there is barely any paper addressing this topic. So instead, α is denoted as the water vapor self-diffusivity in the atmospheric environment. The relationship between the temperature and the self-diffusivity for gas particles in a binary mixture can be described as the following semi-empirical equation based on Chapman–Enskog theory [2]:

$$D(T,P) = \frac{AT^{1.5}}{P\sigma_{12}^2\Omega}\sqrt{\frac{1}{M_1}+\frac{1}{M_2}} \tag{6}$$

Where A is an empirical constant, M_1 and M_2 are the molecular weights of water vapor and air, and σ_{12} is the average collision diameter between two gas species. Ω is a temperature-dependent non-linear collision integral, but since it has an order of magnitude of 1, we treat

it as a constant. As a result, we simplify the above equation into:

$$D(T,P) \approx 5.7x10^{-5}\frac{T^{1.5}}{P} \tag{7}$$

where D (cm^2/s), T (K), P (atm).

Then we normalize the above equation in the range of 0-1 to represent α. Assuming temperature and pressure are 0 $^\circ C$ and 1.013 bar at sea level, and the temperature and pressure drop rate due to the elevation of altitude is 0.65 $^\circ C$/10 meter and 0.001 bar/10 meter respectively since snowflakes usually form between -30 to 0 $^\circ C$, it forms at an altitude around 4620 m by rough estimation. α can be normalized as the equation below, considering the temperature and pressure range in the atmosphere:

$$\alpha(T,P) \approx \frac{\frac{(T+273.15)^{1.5}}{P} - 4456.48}{2423.37} \tag{8}$$

where T ($^\circ C$), P (bar).

The relationship between the temperature and the saturated water vapor pressure can be described as the following empirical equation (Arden Buck equation)[1]:

$$P^{sat}(T) = 6.1115\exp((23.036 - \frac{T}{333.7})(\frac{T}{279.82+T})) \tag{9}$$

where D (cm^2/s), T ($^\circ C$), P (hPa).

Therefore, the parameter γ, which denotes the availability of water from outside the plane of growth, can be expressed as the relative humidity:

$$\gamma(T, p_{H_2O}) = \frac{p_{H_2O}}{p^{sat}} \tag{10}$$

where p_{H_2O} denotes the partial pressure of water vapor, and p^{sat} denotes the equilibrium/saturated vapor pressure of water.

β can be interpreted as the ability of a grid to form ice, which is the degree of crystalliza-

tion, a.k.a. crystallinity. Unfortunately, based on our literature search, this term can only be found in polymers, not in ice formation. Therefore, we have to come up with a method to quantify β. Since temperature affects the degree of crystallization directly, it is reasonable to assume that β is the product of γ and a temperature term, where the lower temperature has a higher capability to form ice:

$$\beta(T, p_{H_2O}) = \gamma(T, p_{H_2O}) \frac{0 - T}{0 - (-30)} \tag{11}$$

Hence, by knowing T (in $^\circ C$, ranged from -30 to 0$^\circ C$), atmospheric P (in bar, ranged from 0.551 to 1.013 bar), and water vapor partial pressure (in hPa), α, β, and γ can be obtained. By adjusting the temperature T, we can render different configurations for snowflakes, as seen in Fig. 19 and 20.

By comparing the shapes of snowflakes generated by Reiter's model and the 3-Equation model and those in Fig. 1, we observe that the snowflakes generated by the 3-Equation model resemble the shapes of real snowflakes better. Although the artificially generated snowflakes under -5$^\circ C$ seem to be very off, the ones generated under -15$^\circ C$ do look similar to real snowflakes. Therefore, we can indicate that both simulation models with updated temperature-to-parameter relations could generate plate-shaped snowflakes relatively well, especially between -15$^\circ C$ and -10$^\circ C$.

Humidity also plays a significant role in snowflake formation. Water vapor pressure p_{H_2O}, ambient pressure P, and temperature T are the factors of relative humidity. For Reiter's model (21), higher humidity causes plate-shaped snowflakes to form, whereas lower humidity facilitates delicate topologies. The 3-equation model possesses the same tendency of delicacy (**??**), but the speed of formation is lower, which is a bit counter-intuitive.

Our model has room for improvement. Firstly, only the crystallization of ice is explained. However, at higher temperatures ($above - 5^\circ C$), the formation of snowflakes also affect by melting which results in plate-like topologies rather than delicate structures. Maybe adding

additional rules describing such an effect would be helpful. Secondly, for temperatures higher than -10°C, we might need to sort out a new model and temperature-to-parameter relations to simulate column-shaped snowflakes. For future work, one can develop time-dependent p_{H_2O}, P, and T functions, and apply them to Reiter's and 3-component model. By doing so, if we know the thermodynamic pathway of snowflake formation (though it may be very complicated), we can better predict the shape of snowflakes.

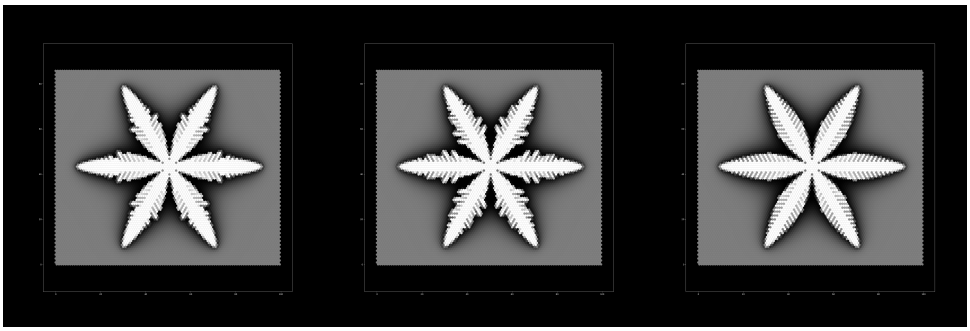

Figure 19: Snowflakes generated with different temperature values using Reiter's model. From left to right, T takes $-5,\ -10,\ -15°C$.

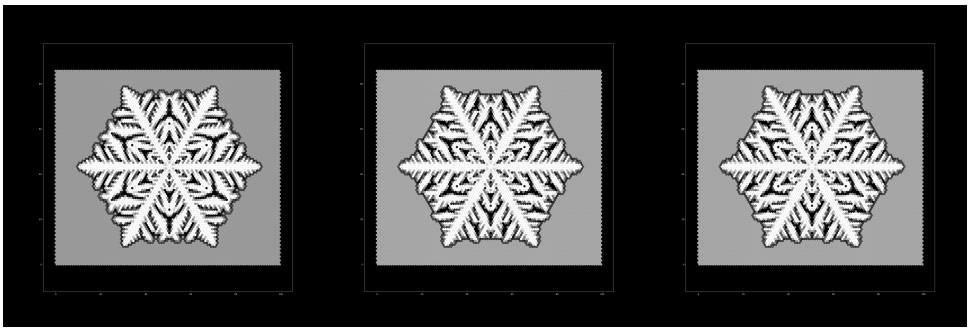

Figure 20: Snowflakes generated with different temperature values using the 3-equation model. From left to right, T takes $-5,\ -10,\ -15°C$.

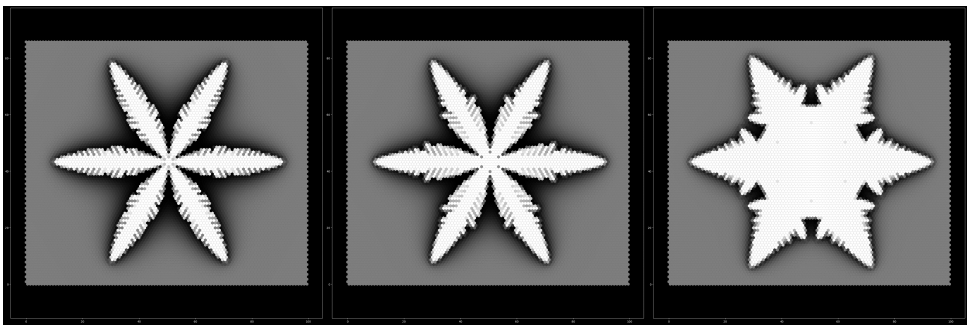

Figure 21: Snowflakes generated with different humidity values using Reiter's model. From left to right, P_{H_2O} takes $0.001,\ 0.01,\ 0.1hPa$. Temperature is set to $-5°C$

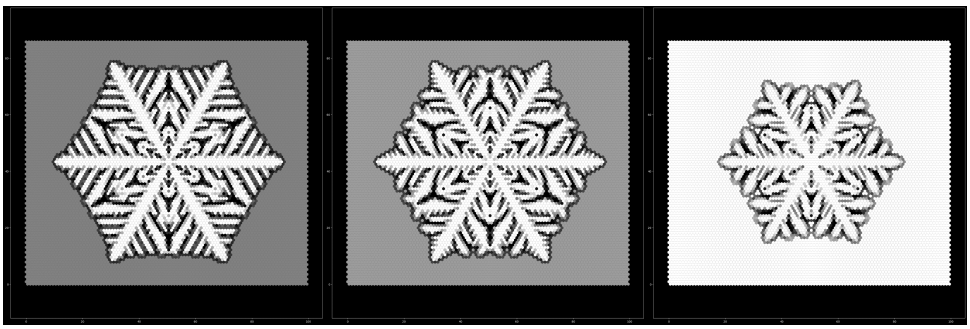

Figure 22: Snowflakes generated with different humidity values using the 3-equation model. From left to right, P_{H_2O} takes $0.001,\ 0.01,\ 0.1hPa$. Temperature is set to $-5°C$

6 Conclusions

The objective of this project is to enhance already existing cellular automata models that simulate the formation of snowflakes. The models we have studied are Reiter's model and the 3-Equation model. During the first phase of the project, the aim is to vary parameters that depend on temperature and saturation while utilizing CA to model the formation of snowflakes in a pure water system under different temperature conditions. During the second phase, through the exploration of temperature effects, we aspire correlate artificial model parameters with temperatures of the realistic environment and to replicate snowflake structures, and improve our comprehension of this natural occurrence. After simulating the process of snowflake formation under realistic conditions, we observed that both models produce shapes that closely resemble actual snowflakes when temperatures range between -15°C and -10°C. Delicate snowflakes are generated under low temperature and low humidity conditions. However, for temperatures above -10°C, we recommend developing a new model and establishing new temperature-to-parameter relationships to accurately simulate non-plate-shaped snowflakes as well as the melting effect.

REFERENCES

[1] Arden L. Buck. New equations for computing vapor pressure and enhancement factor. *Journal of Applied Meteorology*, 20:1527–1532, 1981.

[2] E. L. Cussler. *Diffusion: mass transfer in fluid systems.* 1984.

[3] dcxSt. Snowflake automaton. `https://github.com/dcxSt/snowflake_automaton`, 2021. Accessed on April 24, 2023.

[4] Ch.-A Gandin, J.-L Desbiolles, M Rappaz, Ph Thé, and Thé Voz. A three-dimensional cellular automaton-finite element model for the prediction of solidification grain structures. *Metallurgical and Materials Transactions A.*, 30:3153–3165, 1999.

[5] Richard J Gaylord and Kazume Nishidate. *Modeling nature: Cellular automata simulations with Mathematica®.* Springer, 2013.

[6] L. Wang H. Yin, S.D. Felicelli. Simulation of a dendritic microstructure with the lattice boltzmann and cellular automaton methods. *Acta Materialia*, 59:3124–3136, 5 2011.

[7] Kenneth G. Libbrecht. The formation of snow crystals. *American Scientist*, 93(1):52–59, 2005.

[8] Jianmin Liu. Simulation of the crystallization process based on cellular automata - snowflake formation from pure water system. *Computer Aided Chemical Engineering*, 48:187–192, 2020.

[9] Ukichiro Nakaya. *Snow crystals: natural and artificial.* Harvard university press, 1954.

[10] Chen Ning and Clifford A. Reiter. A cellular model for three-dimensional snow crystallization. *Computers Graphics*, 31(4):668–677, 2007.

[11] OpenSnow. Snowflakes - explained.

[12] Clifford A. Reiter. A local cellular model for snow crystal growth. *Chaos, Solitons Fractals*, 23(4):1111–1119, 2005.

[13] Andrew C. Warden, Bryce A. Little, and Victoria S. Haritos. Fa cellular automaton model of crystalline cellulose hydrolysis by cellulases. *Biotechnology for Biofuels*, 4, 2011.

[14] X. Y. Xu and W. Wang. Nonisothermal cellular automata simulation of two-dimensional snow crystal growth. *Physical Review E*, 106(055309), 2022.

[15] Yifan Zhao, Stephen A Billings, and Daniel Coca. Cellular automata modelling of dendritic crystal growth based on moore and von neumann neighbourhoods. *International Journal of Modelling, Identification and Control*, 6(2):119–125, 2009.

[16] A. Zyska, Z. Konopka, M. Łągiewka, and M. Nadolski. Modelling of the dendritic crystallization by the cellular automaton method. *Archives of Foundry Engineering*, 16:99–106, 3 2016.

APPENDIX: DIVISION OF LABOR

Chao-Wen Chang - Coding of setting up of initial hexagonal grid properties and visualizations. Write-up for (§4, and modeling and coding of §5.4).

Bowen Mu - Modeling, coding, and writing of Reiter's model (§5.2). Selecting and formating the Latex template.

Nianhan Tian - Setting up Github and write up of Abstract, Introduction and Description of System (§1). Modeling, coding and writing of Influence of Temperature and Humidity (§5.4).

Xiaohan Yu - Modeling, coding, and writing of 3-equation model (§5.1).