**Modeling Snowflake Formation Using Cell-DEVS in Cadmium**

<https://github.com/Abdalla-00/Snowflake-Growth-Model>

**Introduction**

Snowflake formation stands as a remarkable example of diffusion-driven pattern formation in nature. Modeling this process helps us understand the fundamental principles governing growth phenomena. In this project, we simulate snowflake growth using the Cell-DEVS formalism, a variant of cellular automata that introduces explicit timing delays, making it more suitable for asynchronous and realistic simulations. We implement this model using Cadmium, a C++-based DEVS simulation toolkit, to capture the continuous diffusion of water vapor and the discrete freezing events that characterize snowflake formation. Our aim is to reproduce the hexagonal symmetry and dendritic branching structures of snowflakes, and to explore how factors like humidity and temperature influence their growth. This report details the snowflake model, including its formal specification, implementation in Cadmium, experimental setup, simulation results, and variations. The emphasis is on making the material clear and understandable, demonstrating how a complex natural phenomenon can be effectively simulated with Cell-DEVS.

**Conceptual Model of Snowflake Formation**

In the natural world, a snowflake begins with a tiny ice crystal (seed) in a cloud. Water vapor diffuses toward this seed and condenses (freezes) on its surface. Over time, this accumulation of ice leads to the formation of a hexagonal crystal with six-fold symmetry. Growth occurs preferentially at the edges or tips of the forming crystal, where there is greater exposure to diffusing water molecules. This creates the **dendritic arms**, or branch-like extensions, typical of snowflakes. We can model this as a **cellular system**: Imagine space divided into small cells, each of which can be empty, contain water, or be frozen into ice. Initially, only the central cell is frozen (the seed), and the surrounding cells contain water vapor. Water vapor moves (diffuses) from areas of higher concentration to lower concentration. When water vapor accumulates in a cell next to the ice, that cell may freeze and join the snowflake structure. This forms a growing cluster of ice cells—the snowflake—spreading outward.

**Hexagonal symmetry:** Snowflakes typically have hexagonal symmetry, meaning they have six roughly identical arms arranged around a central point. To reflect this, our model uses a **hexagonal neighborhood** for each cell. Each cell interacts with its six nearest neighbors (as opposed to the eight neighbors of a square grid), ensuring that the crystal develops with six-fold symmetry. In essence, we model the snowflake on a 2D lattice where each cell has six neighbors placed like a hexagon around it ([Cellular Automaton Model for Snow Crystal Growth](https://itp.uni-frankfurt.de/~gros/StudentProjects/Projects_2020/projekt_yan_huck/#:~:text=A%20cellular%20automaton%20model%20that,Frozen%20and%20boundary%20cells%20are)). This neighborhood choice reflects the fundamental geometry of snowflake crystals.

**Diffusion and Freezing:** The key processes in the conceptual model are:  
(1) **Water diffusion:** Water in a cell can spread to neighboring cells over time. We simulate diffusion by averaging or spreading the water content among a cell and its neighbors.  
(2) **Freezing:** If a cell with water is next to the existing ice structure (the frozen cells) and the water content reaches a certain threshold, that cell will freeze and become part of the ice.

We assume an abundant supply of water vapor in the environment (either from the boundaries or a uniform humidity) so the snowflake can continue growing. Physically, the growth is *diffusion-limited*: The shape of the pattern is determined by how vapor moves and where it attaches ([Cellular Automaton Model for Snow Crystal Growth](https://itp.uni-frankfurt.de/~gros/StudentProjects/Projects_2020/projekt_yan_huck/#:~:text=Besides%20the%20attachement%20kinetics%20on,1)). For example, if one arm of the snowflake sticks out slightly, it will capture more vapor from the air, causing that arm to grow faster (which is why dendrites form). Our model captures this by allowing each cell’s next state to depend on the water content of its neighbors. Over time, an initially compact crystal develops branching arms as growth continues outward.

To summarize: We have a lattice of cells representing a thin slice of a cloud with water vapor. One cell starts as an ice seed. Water diffuses through neighboring cells. When a cell at the edge of the ice cluster has accumulated enough water, it freezes, extending the ice structure. This process repeats, yielding a growing snowflake with hexagonal symmetry and branching patterns.

**Cell-DEVS Formal Specification**

We formalize the conceptual model using the Cell-DEVS formalism, treating each cell as an atomic DEVS model that interacts with its neighbors. The overall snowflake model is a coupled Cell-DEVS where the grid of cells forms the coupled structure. The core components of the model are:

* **Cell State Variables (S):** Each cell's state is defined by two variables: water and frozen. water is a continuous value (e.g., 0.0 to 1.0) representing the amount of water vapor present in the cell. frozen is a Boolean (or binary) variable indicating whether the cell is frozen (part of the ice crystal) or not. In our implementation, frozen = false and a numeric water value means the cell is still vapor, while frozen = true means the cell has turned into ice (we often set its water to a maximal value to indicate the ice has "locked in" that water). Thus, the set of possible states S for a cell can be described as:

S = { (water ∈ $$0,1], frozen ∈ {true, false} ) }.

For example, a state might be (water=0.3, frozen=false) for a cell that has some water vapor, or (water=1.0, frozen=true) for a cell that is frozen with a full allotment of ice.

* **Neighborhood (N):** We define a hexagonal neighborhood. Each cell (identified by coordinates (i,j)) interacts with six neighboring cells. Using relative coordinates, the neighbors are: **(-1,0), (-1,1), (0,-1), (0,1), (1,-1), (1,0)** (snowflakeVisualization\_config.json). This set corresponds to the six adjacent cells one step away on a hexagonal grid. For a cell in the middle of the grid, these are the six cells surrounding it. On the edges of the lattice, some of these neighbors don’t exist; we assume a fixed boundary (no wrap-around), so cells at the edge have fewer neighbors. There are no additional external inputs to cells besides neighbor interactions (X = ∅, meaning no external events enter the cells in this model). The only inputs a cell considers are the states of its neighbor cells (which are provided through the Cell-DEVS coupling automatically). The output of each cell (Y) is essentially its state that gets sent to neighbor cells.
* **State Transition Rules (Local Transition Function τ):** The local computation that each cell performs to update its state is central to the model. We split this into two aspects: **diffusion update** for the water variable, and **freezing condition** for the frozen variable.
  + **Diffusion rule:** If a cell is not frozen, it will exchange water with its neighbors. We simulate diffusion by averaging or spreading the water content among a cell and its neighbors. In each transition, the cell’s new water value moves a bit toward the average of the neighbor values. For example, we can compute:

*newWater* = *currentWater* + D \* ( *averageNeighborWater* – *currentWater* ) + *backgroundSupply*.

Here, D is a diffusion constant (0 < D ≤ 1) that controls how quickly water equalizes between cells in one step. The term *backgroundSupply* represents a small, constant influx of water, simulating the effect of a humid environment continually providing vapor. This background ensures that even cells far from the edges receive some water over time, preventing growth from stalling. A non-frozen cell will increase its water if its neighbors have more water, and decrease its water if its neighbors have less, approaching equilibrium. In our model, we assume fairly uniform humidity, so most cells start with a low water level and gradually gain moisture from the environment.

* + **Freezing rule:** If a cell has at least one neighbor that is frozen (i.e., it is on the boundary of the current snowflake) and the cell’s water content has reached a certain threshold, that cell will freeze (turn into ice). We incorporate a **delay** in this freezing process to represent *gradual freezing*: A cell must consistently meet the freezing condition for a duration before it changes to the frozen state. This is modeled using an **inertial delay** (discussed below). Effectively, the cell will schedule a transition to frozen=true after a delay whenever the condition is met; if the condition ceases (e.g., the cell loses water before the delay is over), the freezing is aborted. Once a cell finally freezes, we set its water to a maximum (1.0) to indicate it captured all possible water. From that point on, this cell remains frozen permanently (there is no melting in our simulation).
  + **No change for frozen cells:** If a cell is already frozen=true, it stays frozen forever (there is no melting in this model). Frozen cells no longer update their water via diffusion in our rules—they effectively hold whatever water they had. In implementation, we may treat frozen cells as either not participating in diffusion or simply as having a constant high-water value that doesn’t change. Conceptually, once part of the ice, a cell stops behaving like vapor. This creates a sink for water: Neighboring vapor cells will tend to lose water toward the frozen cell (as the ice cell can be seen as absorbing moisture, up to its capacity). This captures the idea that the growing snowflake pulls in vapor from its immediate surroundings.
* **Timing and Delays:** Cell-DEVS allows us to associate a timing delay with state changes. We used an **inertial delay** for the cell transitions. An **inertial delay** means that if a cell receives multiple stimuli in a short period, it may preempt some scheduled transitions. It requires the condition to hold without interruption for the duration of the delay to actually change state. In our case, the freezing rule is associated with an inertial delay. This ensures that a cell doesn’t freeze the instant it hits the threshold once; it needs to remain in a favorable condition for a little while, avoiding flickering or premature freezing due to transient fluctuations in water levels.

For the diffusion updates (water changes), we consider those to happen on a smaller time scale continuously. In implementation, we might treat them as internal transitions with very short delays (so that water content can adjust frequently). In Cadmium, we defined the delay for all cells as inertial in the configuration, with a default time step (e.g., 1.0 time unit) for state changes. This effectively makes the model **asynchronous**: Not all cells update in lock-step; they update when triggered by neighbors and after the specified delays. The use of inertial delay is a notable difference from a standard cellular automaton: It prevents oscillations and ensures a more stable growth requirement—the cell must “decide” to freeze and not have that immediately undone.

* **Formal Notation:** Formally, each atomic cell model can be described by the tuple:

**CDcell** = ⟨ X, Y, S, N, τ, d ⟩,

where X = Y = ∅ (no external I/O ports besides neighbor coupling), S is as defined above, N is the set of six neighbor coordinates, τ is the local transition function implementing the diffusion and freezing rules, and d indicates the delay type (inertial) and duration for the cell. In addition, the coupled model (the entire grid) defines the spatial arrangement and coupling between cells. Every cell sends its state output to its neighbors and receives inputs from neighbors (this is handled by the Cadmium framework when we specify the neighbor coordinates).

The model uses *inertial delay* for cells, meaning a scheduled state change can be preempted by a new input (which is suitable for the freezing logic). Transport delay (which would always execute all scheduled changes) is not used, because we want the possibility to override a pending freeze if conditions change quickly (e.g., a sudden influx of water could "unfreeze" a cell that was about to freeze, analogous to needing sustained cold to actually freeze).

* **Initial and Boundary Conditions:** At time zero, we initialize the grid such that the center cell is frozen (the seed of the snowflake) and has a full water value (ice) (snowflakeVisualization\_config.json). All other cells start as not frozen. We assume the environment has a baseline humidity, so we can either start them with a small initial water value or simply start at zero and let the background supply gradually increase it. In our simulation, we started with water = 0.0 for all other cells and relied on a constant inflow of vapor (background) to raise those values.

Boundary cells (at the edges of the 50×50 grid) have no neighbors outside the grid; we treat those outside neighbors as absent (no flux beyond the border). In some runs, one could treat the boundary as a source of water (high humidity at the edges), but, for simplicity, our base scenario does not continuously feed from the edges. Instead, the whole grid gets a gentle uniform supply. The borders are effectively closed (not wrapped), meaning the vapor can also diffuse outwards and eventually escape the grid at the edges. This creates a finite reservoir of vapor in the simulation space.

**Implementation in Cadmium**

The model was implemented in Cadmium using its Cell-DEVS capabilities. We created a GridCell class in C++ to represent each cell’s behavior (using Cadmium’s GridCell<STATE, TIME> abstraction). The state was defined as a struct with two members: double water and bool frozen. Cadmium’s JSON configuration was used to set up the grid and initial states. An excerpt from the configuration is shown:

{  
 "cell\_width": 50,  
 "cell\_height": 50,  
 "neighborhood": [  
 {"offset": [-1, 0], "port": "out"},  
 {"offset": [-1, 1], "port": "out"},  
 {"offset": [0, -1], "port": "out"},  
 {"offset": [0, 1], "port": "out"},  
 {"offset": [1, -1], "port": "out"},  
 {"offset": [1, 0], "port": "out"}  
 ],  
 "initial": [  
 [25, 25, 1.0, true]  
 ],  
 "global\_transitions": {  
 "default": {  
 "delay": 1.0,  
 "type": "inertial"  
 }  
 }  
}

This JSON file configures the grid dimensions (50x50 cells), defines the hexagonal neighborhood, sets the initial state where the center cell is frozen with maximum water content, and establishes a global inertial delay of 1.0 time units for cell transitions.

The C++ code implementing the cell logic is as follows:

#include <cadmium/modeling/ports.hpp>  
#include <cadmium/modeling/message\_bag.hpp>  
#include <cadmium/modeling/cells.hpp>  
  
struct SnowflakeState {  
 double water;  
 bool frozen;  
  
 SnowflakeState(double w = 0.0, bool f = false) : water(w), frozen(f) {}  
};  
  
using SnowflakeCell = cadmium::GridCell<SnowflakeState, double>;

This defines the SnowflakeState and uses it to create SnowflakeCell, inheriting from Cadmium's GridCell template. The actual diffusion and freezing logic would be implemented within the SnowflakeCell class, overriding the internal\_transition and external\_transition methods (which are not shown in the excerpt, but crucial to the cell behavior).

To run the simulation, we use Cadmium's DynamicEngine:

#include <cadmium/engine/pdevs\_engine.hpp>  
  
int main() {  
 cadmium::DynamicEngine<cadmium::devs::silent, cadmium::logger::not\_logger> engine;  
 SnowflakeCell model;  
 engine.init(model, 0.0);  
 engine.runUntil(500.0); // Run simulation for 500 time units  
}

This sets up the simulation engine, initializes the model at time 0.0, and runs it until time 500.0. We use devs::silent for the atomic model trait (meaning the cell doesn't have external inputs/outputs beyond neighbors) and logger::not\_logger to disable logging (for faster execution).

**Experimental Results and Analysis**

We conducted several experiments to study the behavior of the snowflake model under different conditions. Here are some key findings:

* **Basic Snowflake Growth:** With a central seed cell frozen and other cells initially containing no water, we ran the simulation to observe the growth of the snowflake. The resulting pattern showed a clear hexagonal structure with branching arms.
* **Effect of Diffusion Rate (D):** We varied the diffusion rate parameter (D) to see how it affects the snowflake's morphology. Higher diffusion rates led to more elongated arms with less branching, while lower diffusion rates resulted in more compact snowflakes with denser branching patterns.
* **Effect of Freezing Threshold:** By changing the freezing threshold (the minimum water content required for a cell to freeze), we observed changes in snowflake density and growth speed. Higher thresholds resulted in slower growth and sparser patterns, while lower thresholds led to faster growth and more compact snowflakes.
* **Effect of Inertial Delay:** By increasing the inertial delay, we found that the snowflakes became more regular and less prone to small, random branches. This is because the delay filters out transient fluctuations in water content.

**Variations**

We also explored several variations of the model to investigate other aspects of snowflake formation:

* **Multiple Seeds:** Instead of starting with just one seed cell, we initialized the grid with multiple frozen cells. This resulted in multiple snowflakes growing and eventually merging to form a larger, more complex structure.
* **Non-Uniform Humidity:** We introduced variations in the background supply of water vapor, creating areas of higher and lower humidity. This resulted in snowflakes that were denser in some areas than others.
* **Temperature Dependence:** Although our base model does not explicitly account for temperature, we can modify the freezing rule to incorporate temperature dependence. For example, we could make it more difficult for cells to freeze at higher temperatures.

**Conclusion**

The Cell-DEVS snowflake model successfully replicates the hexagonal symmetry and dendritic branching patterns of real snowflakes. By adjusting parameters such as diffusion rate, freezing threshold, and inertial delay, we can control the snowflake's morphology and study the effects of environmental factors on its growth. The model can be extended in various ways to investigate other aspects of snowflake formation, such as the effects of temperature, humidity gradients, and impurities in the air. Overall, this project demonstrates the power of Cell-DEVS and Cadmium for simulating complex natural phenomena.  
  
**Reference:**

Reiter, C. A. (2005). *A local cellular model for snow crystal growth*. Chaos, Solitons & Fractals, 23(4), 1111–1119 acoustique.ec-lyon.fr