HAW HAMBURG

BACHELORTHESIS
Kamila Shirinova

# Snowpack simulation on the GPU in Unity

FACULTY OF COMPUTER SCIENCE AND ENGINEERING
Department of Information and Electrical Engineering

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

HAMBURG UNIVERSITY
OF APPLIED SCIENCES

Hochschule für Angewandte
Wissenschaften Hamburg

Kamila Shirinova

# Snowpack simulation on the GPU in Unity

Bachelor Thesis based on the examination and study regulations
for the Bachelor of Engineering degree programme
*Bachelor of Science Information Engineering*
at the Department of Information and Electrical Engineering
of the Faculty of Engineering and Computer Science
of the University of Applied Sciences Hamburg
Supervising examiner: Prof. Dr. Klaus Jünemann
Second examiner: Prof. Dr. Leutelt Lutz

Day of delivery: 16. September 2023

**Kamila Shirinova**

**Title of Thesis**

Snowpack simulation on the GPU in Unity

**Keywords**

Snow, Simulation, GPU, Compute, Unity

**Abstract**

Snow is a fascinating yet very complex phenomenon, its behavior is defined by a lot of interconnected physical processes. Snowpacks and snow covers are common subjects in computer modeling, especially in video gaming development, where snowy environments are featured quite often. While there exists a variety of snow modeling solutions, with some being quite advanced, not a lot of them dive into the specifics of these internal processes. This thesis tackles the relation between snow density, temperature, stiffness, and hardness, and simulates its vertical compression as a function of those parameters and the applied pressure over time. Moreover, the simulation uses a 3-dimensional grid as a means to discretize the snowpack volume and the performance advantage of a compute shader, run on the GPU. As the intended application of this simulation is game development, it is implemented as a Unity project.

**Kamila Shirinova**

**Thema der Arbeit**

Schneedecken-Simulation auf der GPU in Unity

**Stichworte**

Schnee, Simulation, GPU, Compute, Unity

**Kurzzusammenfassung**

Schnee ist ein faszinierendes und doch sehr komplexes Phänomen, dessen Verhalten durch eine Vielzahl miteinander verbundener physikalischer Prozesse bestimmt wird.

Schneemassen und Schneedecken sind häufige Themen in der Computermodellierung, insbesondere bei der Entwicklung von Videospielen, in denen verschneite Umgebungen recht häufig vorkommen. Es gibt zwar eine Vielzahl von Lösungen zur Schneemodellierung, von denen einige recht fortschrittlich sind, aber nur wenige gehen auf die Besonderheiten dieser internen Prozesse ein. Diese Arbeit befasst sich mit der Beziehung zwischen Schneedichte, Temperatur, Steifigkeit und Härte und simuliert die vertikale Kompression des Schnees in Abhängigkeit von diesen Parametern und von externem Druck über Zeit. Die Simulation nutzt ein dreidimensionales Gitter als Mittel zur Diskretisierung des Schneevolumens und einen auf der GPU laufenden Compute-Shader, um bessere Leistung zu erzielen. Da die beabsichtigte Anwendung dieser Simulation die Entwicklung von Spielen ist, wird sie als Unity-Projekt implementiert.

# Contents

# Glossary

**CFD** CFD, or Computational Fluid Dynamics, is a branch of fluid mechanics that involves the use of numerical methods and computational techniques to simulate and analyze the behavior of fluids (liquids and gases) and their interactions with surfaces and boundaries, as well as other related phenomena. Simulation methods such as MPM, SPH and FLIP all fall under the broader umbrella of CFD.

**Compute shader** A compute shader is a type of shader in computer graphics programming that is used to perform general-purpose computations on a graphics processing unit (GPU).

**Eulerian grid** An Eulerian grid, also known as a fixed grid or spatial grid, is a mathematical framework used in scientific simulations. In an Eulerian grid the computational domain is divided into a regular, fixed arrangement of grid cells or voxels, and the grid points or cell centers remain stationary throughout the simulation.

**Extended Column Test** The Extended Column Test (ECT) is a snow stability test used in snow science and avalanche forecasting to assess the potential for snowpack instability and the likelihood of triggering an avalanche. This test involves creating an extended column of snow by cutting a rectangular block of snow from the snowpack and applying pressure to the snow surface. The goal is to observe if and how cracks propagate across the column.

**Firn** Firn refers to a transitional stage in the transformation of snow into glacial ice. It is a granular, compacted snow layer that lies between freshly fallen snow and glacial ice.

**FLIP** FLIP, or Fluid-Implicit Particle, is a computational method used in computer graphics and fluid simulation to accurately simulate the behavior of fluids, such as liquids and gases, in animations and visual effects. This is a hybrid method,

that combines two popular techniques: the grid-based Eulerian approach and the particle-based Lagrangian approach.

**HLSL** HLSL, or High-Level Shader Language, is a programming language developed by Microsoft for use in graphics and compute shaders.

**Hooke's Law** Hooke's Law is a fundamental principle in physics that describes the relationship between the force applied to an elastic material and the resulting deformation or change in its shape.

**Lagrangian particles** Lagrangian particles are commonly used in computational simulations, to represent a specific material or fluid particle within a flow or simulation. The movement and behavior of these particles, together with other relevant properties, are tracked as they move through space and time.

**MPM** MPM, or Material Point Method, is a computational framework used for simulating the behavior of materials undergoing complex deformation and dynamics. Much like FLIP, this one combines aspects of both the Lagrangian (particle-based) and Eulerian (grid-based) approaches, however, its focus on simulating material behavior sets it apart from traditional fluid simulations..

**Polynomial regression** A type of regression analysis used to model the relationship between a dependent variable and one or more independent variables. In polynomial regression, the relationship between the variables is expressed as a polynomial equation of a certain degree.

**SPH** SPH, or Smoothed Particle Hydrodynamics, is a computational method used for simulating fluid flows, soft materials, and other phenomena involving complex interactions. SPH is a specific method within CFD that focuses on simulating fluids using particles rather than a grid-based approach.

**Vertex-fragment shader** A vertex-fragment shader is a pair of shaders used in modern graphics programming pipelines to control the visual appearance of objects in a 3D scene. These shaders work in conjunction to determine how vertices (points) of 3D models are transformed and how pixels (fragments) are shaded to produce the final rendered image.

# 1 Introduction

Snow is an extremely versatile phenomenon which is to this day not yet fully studied. A characteristic property of snow that makes it stand out among other particle-representable volumes like sand or fluids is its innate compressibility. This makes it so much more interesting to explore its behavior, trying to recreate it in the context of what is loved and enjoyed by many - including myself - the video games. Though snowy levels are utilized extensively and modern game titles often do a great job at displaying them, a physically accurate representation of the processes that run inside the snow-pack is typically ignored. Not without a reason, since snow in video games is usually only used for visual "dressing" and a higher player immersion, in which case the features of snow simulation mostly limit themselves to leaving footprints/tyre tracks on the snow cover, snowfall and accumulation, collision response and sometimes phase change. If a snow simulation system is present in a game, it is usually responsible for modeling snow dynamics as an immediate response to the interaction with the in-game environment or entities, not considering the long term effects on the snow-cover state. The point of this thesis, however, is to explore the internal processes of the snowpack that is represented as a 3D volume in discretized space, where the state of each space unit is defined by a set of spatial properties. When applied in game development, this kind of exposed spatial information at any point in space could open a lot of doors for new gameplay options and mechanics. The process simulated here is the vertical compression/compaction of the snowpack under its own weight or external pressure, with the amount of compression being dependent on the snowpack's internal properties.

## 1.1 Outline

This thesis is divided into 7 chapters. The current chapter with general introduction is followed by the Background chapter, that contains the research made for this thesis, both on the topic of real-life snow physics and the existing implementations of snow simulation.

The goals and limitations of this thesis are listed in the Requirements chapter, followed by the Design chapter that explains how the snow theory from the Background chapter is adapted for the practical part (sometimes referred to as the project). The Implementation chapter is fully dedicated to the structure and inner workings of the project. Both the visual and the numerical results of the implemented simulation are presented in the Evaluation and Tests chapter, where they are checked against the requirements stated earlier. Finally, the Summary chapter draws a conclusion for this thesis together with ideas for further improvements.

All figures, plots and screenshots featured in this document were created for this thesis by the author of this thesis, unless stated otherwise.

# 2 Background

This chapter covers the research made on the general topic of snowpack physics as well as the existing methods of related simulations in applied computer graphics, in particular, in game development. These prerequisites helped form a general understanding of snow behavior during compression and this knowledge was later used when constructing the simulation algorithm for the practical implementation. This chapter is divided into two parts: the first one is devoted to the theory behind snow, explaining how it compresses and what factors influence its behavior during compression. The second part covers a number of selected works that are relevant to this topic. Since the focus of this thesis is shifted more towards the correct interpretation and implementation of some of the aspects of snow physics, rather than exploring the most efficient implementation strategy, the respective theory sections are covered in corresponding proportion.

## 2.1 Snow Theory

The following physical properties and their connections were studied in the scope of this thesis:

- Snow Density
- Snow Temperature
- Snow Hardness
- Snow Stiffness

### 2.1.1 Snow Density

Snow compression, or, compaction is defined by the density changes within the snowpack - here referred to as snow density profile. Freshly fallen snow usually forms a loose, porous layer. As it lies, it gradually compacts under the influence of its own weight and various meteorological factors. At a *uniform temperature*, after enough time has passed for the snow not to be considered "fresh" anymore the density will gradually increase with snow depth, according to Rikhter [9]. This state of the snow, during which only slight changes in its density occur will be referred to as "stabilized snow" in this thesis.

This correlation between the snow depth and its density is depicted in figure 2.1.

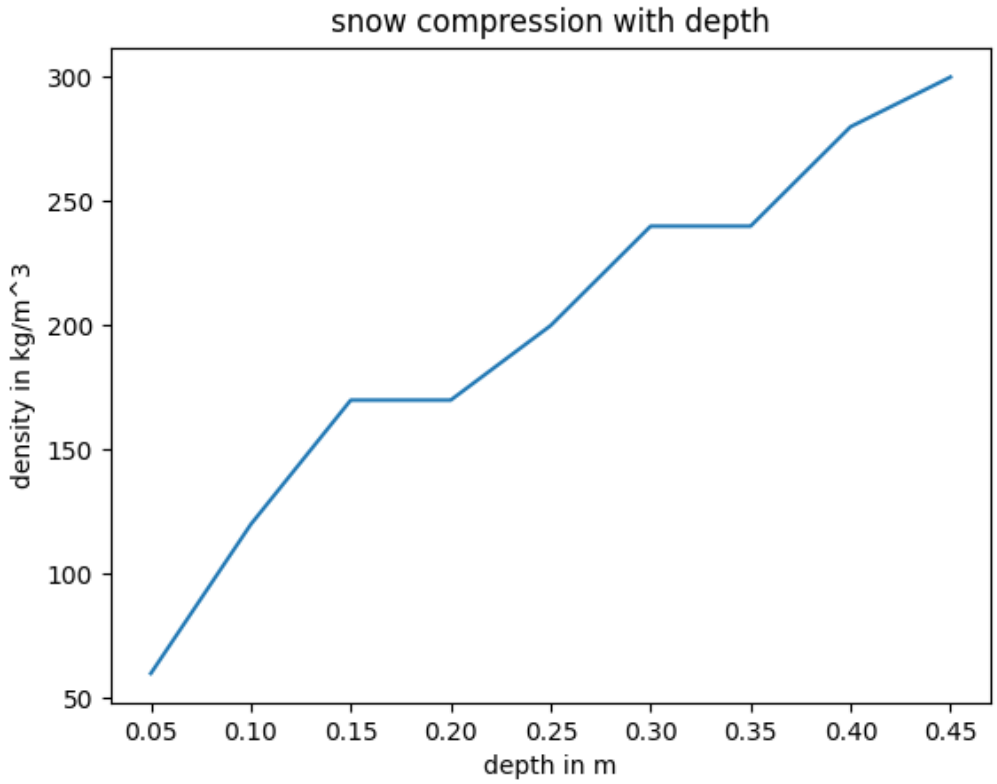



Figure 2.1: Snow density profile for a given depth, based on empirical data taken from [9]. The units were converted to kilograms and meters.

When thinking about the density profile of stabilized snow it is natural to assume that the snow density will always increase with depth, however, this may not always be the

case. In reality, whether the density increases or decreases with depth depends on the snowpack temperature gradient, which is created by the heat convection and micro-sublimation processes that constantly run inside the snowpack. The nature of those processes is determined by the temperature of the underlying surface on which the snow is accumulating.

In areas where snow accumulates on an icy surface that is usually colder than the snow itself, the pressure of its own weight and the heat convection work in the same direction, making the water fumes travel down, where they then crystallize. This way, the snow becomes denser towards the bottom and eventually turns into firn. This process would be different if the snow were forming on soil instead of ice, where due to the soil being warmer, the heat distribution would make the water fumes rise up, and the water then crystallizing closer to the snow surface would make the upper layers denser. The examples of scenario 1 (arctic snow forming on ice) and scenario 2 (snow forming on soil) are shown in fig. 2.2, left and right respectively.

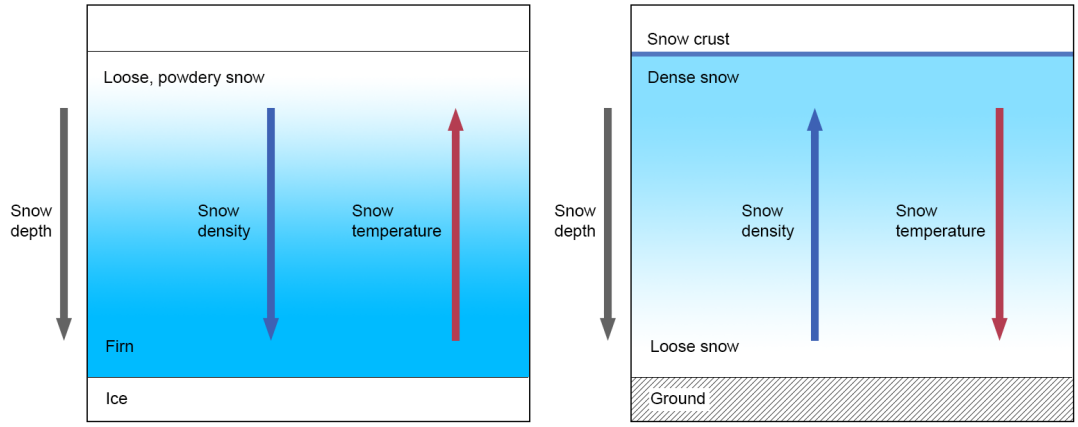



Figure 2.2: Snow density profile for different underlying surfaces. The arrows indicate the direction in which the given quantity increases.

Now that the general picture of stabilized snow density is covered, a more granular inspection of the factors that contribute to the density changes can be done. In general, the intensity with which the snow compacts under pressure depends on the following properties:

- hardness
- stiffness

- viscosity
- cohesion
- friction properties

These properties are referred to as the independent properties of the material, and can be measured directly. Such properties have been used to determine how well a surface made of this material is suited for transport [2]. For this simulation, only the snow hardness and, to a lesser extent, the snow stiffness are considered, and will be covered in the next sub-sections in more detail.

### 2.1.2 Snow Hardness

The main characteristic of snow strength can be described by its hardness. The snow hardness is defined as the minimum force or pressure required for the initial snow deformation to happen, or the maximum force/pressure the snow can withstand without deforming [2]. Thus, the following equations are true for the snow hardness H:

$$H = F_{max}/A \tag{2.1}$$

where H is the snow hardness [Pa], $F_{max}$ is the maximum withstandable force [N], A is the pressure area [m].

In the experiment described in "The Strength of Snow in Compression" by Gold [6], the stress required to continuously push a circular plate into a flat snow surface with the rate of penetration of 0.3 meters per second was measured. The recorded values can be seen in figure 2.3

A notable observation of the dependencies in figure 2.3 is that after surpassing the hardness at around 50 $g/cm^2$, the stress (pressure) curves fit well into the linear relation of $\delta x = \frac{\delta P * A}{k}$, where x is the penetration [cm], P is the change in active pressure [$g/cm^2$], A is the plate area of contact [$cm^2$] and k is the spring coefficient [$g/cm$], and conform to the Hooke's Law. This assumption is heavily relied on during the construction of the simulation algorithm. Additionally, Gold states that the snow hardness depends on its internal properties such as density, temperature, and crystal size, and not the area of the plate. The correlation between these parameters is expressed by equation 2.2:

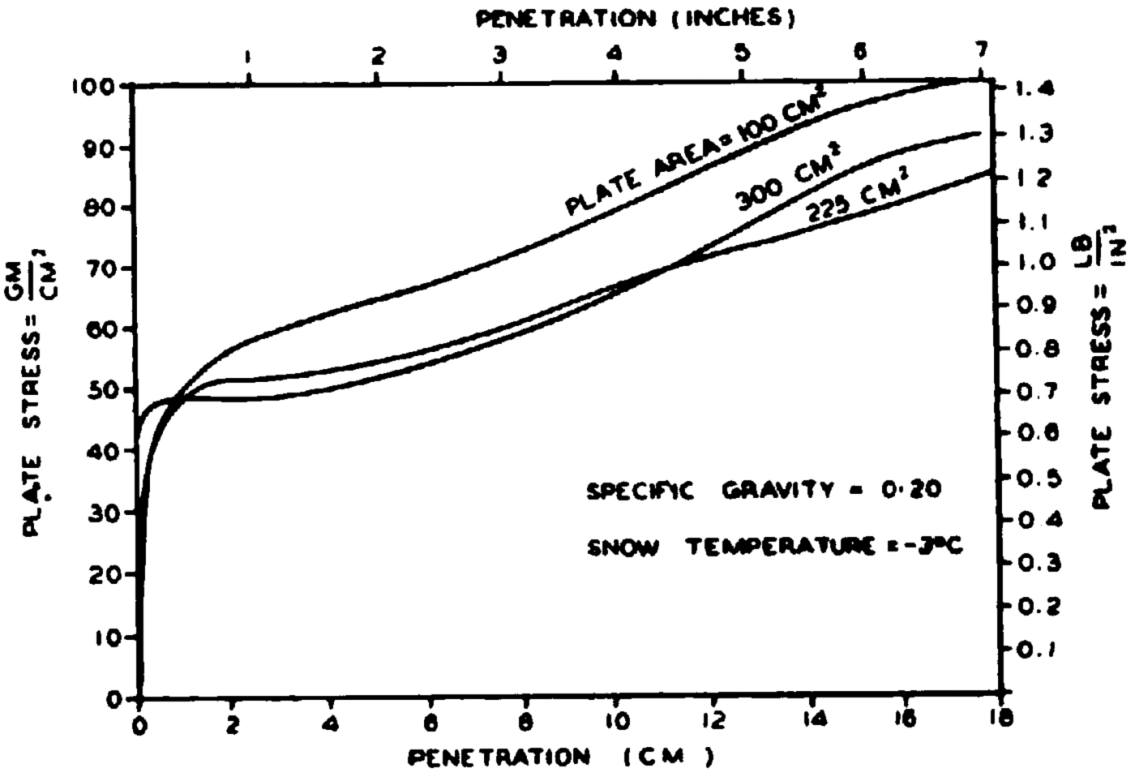



Figure 2.3: Plate stress (Pressure) as a function of penetration, recorded for plates of different areas, taken from [6]

$$H \propto \rho^{3.92} \cdot e^{-0.063 \cdot T} \cdot e^{-0.340 \cdot d} \tag{2.2}$$

where

$$0.2 < \rho < 0.4$$
$$-40 < T < 0$$
$$0 < d < 3$$

In the equation 2.2, H denotes the snow hardness [Pa], $\rho$ is the snow density $[gm/cm^3]$, T is the snow temperature $[°C]$, and d is the average grain size $[mm]$.

In a more recent study on the mechanical properties of snow by Barakhtanov et al.[2], the dependence of snow hardness on its density and temperature is estimated by polynomial regression, and takes the form of:

$$H(\rho) = \sum_{j=0}^{n} C_j \cdot \rho^j \tag{2.3}$$

on density alone, and

$$H(\rho, T) = \sum_{j=0}^{n} C_{(n-j),j} \cdot \rho^{(n-j)} \cdot T^j \tag{2.4}$$

on both density and temperature. Here, H is the snow hardness [Pa], $\rho$ is the snow density $[g/cm^3]$, T is the snow temperature $[°C]$, n is the maximum power of polynomial dependence, and $C_j$ and $C_{(n-j),j}$ are the regression coefficients $[0.1^{3 \cdot j} \cdot \frac{m^{3 \cdot j - 2} \cdot N}{kg^j}]$.

This way, an equation for the snow hardness for the winter period could look like as follows, according to the study [2]:

$$\begin{aligned} H(\rho, T) = &-0.16599 + 0.542153 \cdot \rho - 0.08925 \cdot T + 0.447667 \cdot \rho^2 + 0,159256 \cdot \rho \cdot T \\ &- 0.01336 \cdot T^2 + 0.261043 \cdot \rho^3 - 0.15863 \cdot \rho^2 \cdot T + 0,022601 \cdot \rho \cdot T^2 - 0.00032 \cdot T^3 \end{aligned} \tag{2.5}$$

with H, $\rho$ and T denoting snow hardness [Pa], density $[g/cm^3]$ and temperature $[°C]$, respectively.

### 2.1.3 Snow Stiffness

The stiffness is the property of a material that defines its deformation in response to applied force, and can be expressed by:

$$x = P/k \tag{2.6}$$

where x is the (vertical) indent in the snow cover [m], P is the pressure normal to the snow surface [Pa], and k is the spring coefficient [Pa/m].

The spring coefficient k is also dependent on the snow temperature - the lower the temperature, the more resistant the snow becomes to any deformation. However, for the sake of simplicity, only the influence of density on the the spring coefficient will be considered here, for which the same regression formula has been proposed by Barakhtanov et al.[2]:

$$k(\rho) = \sum_{j=0}^{n} C_j \cdot \rho^j \tag{2.7}$$

Where k is the spring coefficient [Pa/m], $\rho$ is the snow density $[g/cm^3]$, n is the maximum power of polynomial dependence, and $C_j$ are the regression coefficients $[0.1^{3 \cdot j} \cdot \frac{m^{3 \cdot j - 2} \cdot N}{kg^j}]$.

## 2.2 Related Work

When it comes to representation of deformable snow covers in computer graphics, the current state of technology offers quite a number of solutions, ranging from the most simple and undemanding ones (like using heightmaps to offset terrain) to complex system solvers that are capable of extremely realistic visualisation but are by no means real-time.

While non real-time solutions make little sense in the context of game development, it is still interesting to mention Disney's Material Point Method (MPM) for Snow Simulation by Stomakhin et al. [11], which shows impressive realism in simulating snow dynamics for different types of snow. Their method is capable of handling snow deformation, fracture and stickiness in a variety of scenarios. Though the featured technique treats the snow volume as a continuous material and does not render each snow flake individually, it utilizes both a static Eulerian grid and a set of moving Lagrangian particles for its calculations. The particle domain is responsible for tracking conserved properties such as mass, momentum and deformation, while force interpolation happens on the grid. The mass and the momentum are "checked in" by particles onto the grid; the grid then computes forces, updates velocities, checks for collisions and finally transfers the updated velocities back to particles. The particles then use the updated velocities to calculate the deformation gradient, check for collisions once more and update their positions; the cycle then repeats. The collisions are handled in both domains the same way, i.e. twice per simulation cycle. The different behavior of snow types (chunky/powdery snow, fragile/hardened snow, etc.) is governed by a set of manually tweakable snow parameters. This MPM was later augmented to cover phase changes as well [12].

In contrast to this, Gissler et al. [5] introduce a real-time Lagrangian approach based on a Smoothed Particle Hydrodynamics (SPH) method that is also capable of simulating the deformation, breaking, compression and hardening of snow and phase change from fluid to snow. Additionally, due to SPH discretization, where single snow flakes *are* modeled individually, their solution natively covers snow fall and accumulation, which was not covered by the previous example. Gissler et al. inherit the physical basis for their snow simulation from the model proposed by Stomakhin et al. [11], but argue than an SPH realization can be an equal alternative to MPM.
Another real-time particle based solver for snow dynamics is introduced by Goswami et al. [7], where the simulation simplicity and efficiency is achieved by avoiding the use of complex CFD solvers altogether and being implemented in parallel on the GPU. The same paper gives a great overview of the industry achievements at the time of its publication. The algorithm proposed by Goswami et al. [7] models the phase change to water and ice, non-recoverable particle compression and particle cohesion through establishing bonds particles that "stick together".
The most undemanding approach to snow cover representation in computer graphics is pure surface based, and is also the most common one. Here the entire snow surface is represented by a textcolorredheight or displacement map, that project collisions with objects into changed heights/indents in the snow surface. An example of such implemented technique is given by J. Svensson [13]. Another example of an in-game implementation was presented on the Game Developers Conference during the talk on Deformable Snow Rendering in Batman: Arkham Origins [3].
The method used in this thesis is grid-only coupled with surface representation for snow cover rendering. All the physics related calculations are performed on an Eulerian grid (later referred to as grid), including the mass transfer. A single height buffer is used to represent the indented snow surface.

# 3 Requirements

This section is dedicated to outlining the scope of the task, the simulation goals, and the constraints. At the end of this chapter, a requirement map is introduced, which will serve as the assessment criteria for the final project implementation.

## 3.1 Project Scope

The goal of this thesis is to model the changing density and height of a snowpack over time as a function of its own weight, vertically applied external pressure, snow hardness, temperature, and stiffness.

In order to visualize and test the algorithm a demo project is setup, that implements and runs the simulation. The entire simulation is performed iteratively in discretized timesteps and visualized on a 3d grid consisting of $n \cdot n \cdot n$ cubic cells. Each of those grid cells represents a fixed volume in space that can only contain either snow or air, visualized accordingly, and stores information on physical properties at that location (average temperature, density, etc). Implementing this kind of partitioning, rather than calculating the average or total values for the entire snowpack, enables the cell grid to eventually have non-uniform densities, leading to the desired gradient. Another base for the grid-like representation of the space is that the information computed and stored during the simulation could easily be sampled or contributed to by any entities within the grid at any time. This notion will be used when detecting and registering applied external pressure from objects in contact with the snow surface.

Since this thesis only focuses on vertical snow compression, it is helpful to think of vertical stacks of such snow cells as snow columns. Snow columns serve as a convenient way of representing properties that can be aggregated along the snow depth, e.g. the total column mass or height.

During the simulation, the cells are updated iteratively and the entire process is tracked by the changed cell densities and total heights of the snowpack columns. These calculations are carried out in a compute shader in order to maximize the performance.

## 3.2 Project Use Cases

Within the scope of the demo project, 3 scenarios will be tested, i.e. 3 experiments will be conducted to simulate snowpack compression under different weather conditions. These conditions are defined by varying input parameters, mainly the (snowpack) temperature or temperature gradient. The setting in which the snowpack is modeled, and that defines the expected compression behavior, is considered the "Polar setting" from figure 2.2, left. Moreover, the effects from heat sublimation in the same figure 2.2, right will be ignored, as not to further extend the scope of the project. Other than that, any change in initial snow-pack parameters should reflect on the snowpack's behavior and its stabilized state - for example, a lower temperature should lead to higher snow hardness, and therefore smaller compression amount and change in snow cover height.

The effects of the external pressure will be tracked by the indents left by a number of static objects of a given mass and area of contact with the snow surface. The expected behavior in this case is universal for all test cases; objects with a bigger contact area should fall deeper into the snowpack, and vice versa. In theory, this feature could become a base for modelling a range of difficulties for trespassing snowy territories under different weather conditions. As an example, during cold weather, due to a lower compressability of the snow, one would have an easier time treading the snowy surface, without sinking too deep into the snow. Alternatively, travelling with skis or by sleigh could also be considered a safer option, since a bigger contact area would lower the pressure on the snow surface.

Below are listed the scenarios (use cases) in which the simulation experiments in the demo project will be conducted.

### 3.2.1 Uniform initial conditions

The first scenario will test the amount of compression of the entire snow volume with uniform initial temperature, both under its own weight and under external pressure. The

resulting behavior should match the theory described in section 4.1.4, and the overall density profile picture in stable state should resemble that in figure 2.1.

### 3.2.2 Non-uniform initial conditions

The second use case aims to recreate the scenario 2 discussed in chapter 2, shown in figure 2.2, right. Although modelling heat convection is outside of the scope of this thesis, it is still interesting to investigate how the non-uniform initial temperature affects the simulation, as compared to it's uniform configuration. The snow is still expected to become denser at higher temperatures during this simulation. The snow will be initialized with a bottom-up temperature gradient (figure 2.2, right), since the theoretical result of a top-down temperature gradient (figure 2.2, left) is too close to and already covered by the previous use case with the uniform initial conditions.

### 3.2.3 Run-time change of conditions

So far the simulation conditions were said to be pre-initialized for every expeniment and to remain constant through-out the test runs. This time, the temperature will change while the simulation still runs, in order to replicate snowpack layer formation due to accumulation of snow at different points in time under different weather conditions. Visualization of those layers should be indicative of the snowpack's recent history, like how many recent snowfalls have happened and under what weather conditions. Drawing a parallel to real-life situations, this use case could represent a scenario where a weak snow layer has formed that has made the snowy territory unsafe for passage. In the world of backcountry skiing, a close inspection of the snowpack for identifying any weak layers is key to predicting and avoiding avalanches. A weak layer in a snowpack refers to a specific layer within the accumulated snow that has reduced structural integrity and strength compared to the surrounding snow layers. An example of a snow-pit wall, allegedly carved in order to perform the Extended Column Test, containing such layers is shown in figure 3.1.

A more granular coverage of the requirements/constraints that encompass all three use cases are listed in the next section.

## 3.3 Project Constraints and Requirements

The density is calculated as the result of the change in the snow volume, driven either by gravitational force or by externally applied vertical pressure. The changed volume, in turn, is derived from the deformation/indent length x via formula 4.5 (The detailed algorithm is covered in the next chapter), and the factors that influence the deformation amount are limited by the scope of this project to:

- Snow hardness [Pa]
- Snow stiffness [Pa/m]

with the snow hardness being dependent on the following parameters:

- Snow density $[kg/m^3]$
- Snow temperature $[°C]$

and snow stiffness only depending on the density, forming a feedback loop.

In order to help form a behavior curve of the snow density for a given snow depth, the following constraints were derived from the theory described in the previous section, based on scenario 1 illustrated in figure 2.2 (Left, Snow density increases with depth):

1. The snow density starts at some uniform minimum value $\rho_{fresh}$
2. The snow density stays at that value until a pressure threshold that is defined by the snow hardness is passed by acting pressure
3. The snow density then keeps increasing until it reaches some maximum value
4. In between the minimum and maximum points the density is proportional to the pressure acting on the snow at the given point in space.

Moreover, in order to establish the overall relations between the variables in question (hardness, temperature, stiffness, and density), regardless of the specific formulas chosen, the following conditions must be satisfied:

5. The snow hardness is proportional to its density (increasing the snow density increases its hardness)

6. The snow hardness is in reverse proportion to its temperature (increasing the snow temperature decreases its hardness)

7. The snow stiffness, defined by the spring coefficient, is proportional to its density (increasing the snow density increases the spring coefficient)

With the above constraints in mind, the first crude predictions of what the density curve could look like were made (see Figure 3.2). This graph shows the theoretical response of uniform snow density to pressure, for some initial density $\rho_{fresh,1}$, temperature $T_{fresh,1}$ and hardness $H_{fresh,1}$, plotted against the snow depth. The graph can be interpreted in the following way: for an imaginary slab of snow, with uniform initial density and temperature, the snow starts to compress under its own weight at point $depth_1$ below the surface, and around point $depth_2$ it reaches its maximum density $\rho_{max}$. The point $depth_1$ is a crucial turning point associated with the depth at which the pressure from the overlying snow layers is enough to exceed the snow hardness and make the underlying snow compress and deform. This point shifts further to the right from the origin, the higher the initial density value is, according to the prerequisites 2.2 and 2.5, as well as for a lower snow temperature.

In addition to the compaction of snow under its own weight, the correct response to the (vertical) external pressure is modeled. The external pressure is provided by the weight pressure of in-scene objects (snow colliders). The assessment criteria is the indent made in snow surface by multiple objects of same mass, but different collision areas, that must satisfy:

8. With constant mass, a larger collision area results in a lower pressure, and, therefore, lower indent.

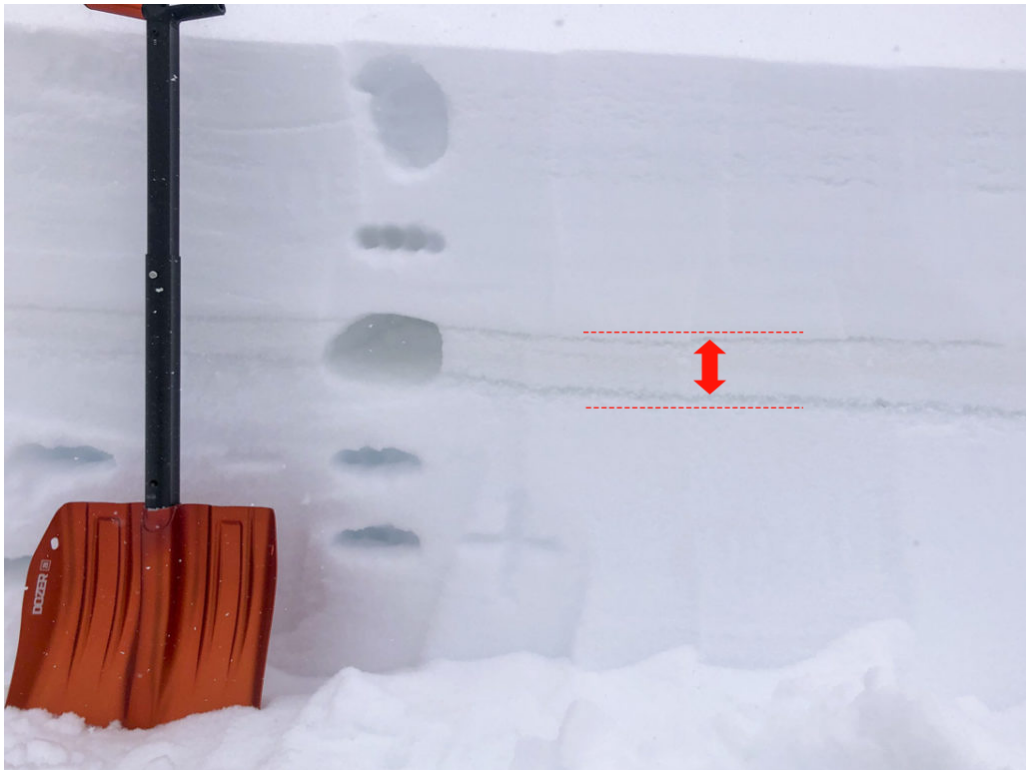
Finally, since only the vertical compression is considered here, an assumption can be made that the snow mass does not transfer left or right, only down, due to vertical compression itself. Therefore, the mass conservation constraint is added to the list:

9. The total mass of a column stays constant, whereas the mass of a cell may change.

All of the constraints listed above are summarized into a set of requirements and mapped to the use cases by which they are covered in table 3.1. In general, the listed requirements apply to *all* use cases and should be fulfilled always, but the table only lists the ones whose discussion in the Evaluation chapter will be focused on those specific requirements.

Table 3.1: Project requirement map

| № | **Requirement** | **Constraint(s)** | **Use case(s)** |
|---|---|---|---|
| 1 | Snow can compress under pressure | 1,2,3,4 | 1,2,3 |
| 2 | Snow only starts compressing after a certain pressure threshold is passed | 2 | 1 |
| 3 | Snow should not compress indefinitely | 3 | 1,2 |
| 4 | Higher snow density leads to less compression | 5,7 | 1,2,3 |
| 5 | Higher snow temperature leads to more compression | 6 | 1,2,3 |
| 6 | Objects that lie on the snow cover sink deeper into snow the smaller the areas of contact between them and the snow cover, and the bigger their masses are | 8 | 1 |
| 7 | There should be no artificial increase in snow mass i.e. snow mass should stay constant unless fresh snow is added explicitly | 9 | 1,2,3 |



(a)

(b)



Figure 3.1: Snowpack containing a visible weak layer (a) and a weak layer forming on the surface prior to its burial (b). The weak layers are identified by their relatively lower strength and density, and thus higher fracture potential than their surrounding layers. The photos are taken from avalanche.org.

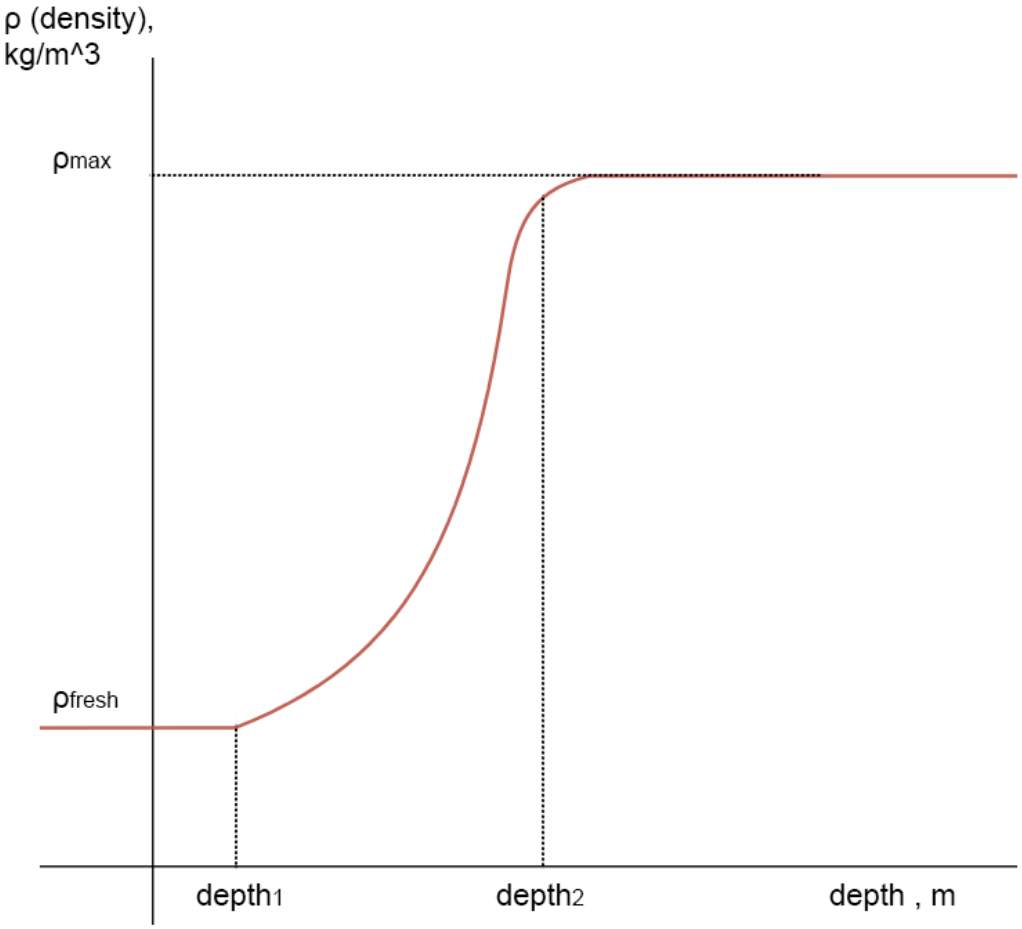



Figure 3.2: Theoretical visualization of the snow density at each depth level as a function of its initial parameters and the pressure of its own weight, based on constraints 1 - 8. The x-axis represents the depth from the snow surface, while the y-axis is the resulting pressure gradient.

# 4 Design

This chapter explains in detail the algorithm that was constructed for the simulation based on the theory and the requirements. Before being implemented in shader code, the simulation algorithm was prototyped in Python. The next section gives a detailed breakdown of the relations between the pressure, hardness, and density that is used by the simulation, followed by preliminary Python tests.

## 4.1 Simulation Algorithm

As already mentioned, the simulation iterates over all grid cells, and each iteration updates some of the cell's properties. A complete cycle ends with the updated densities and snow column heights, then the next cycle is run with the newly computed parameters and so on. Every cycle that a snow cell goes through, consists of the following steps:

1. Calculate the total pressure acting on the cell
2. Calculate the cell hardness based on its density and temperature
3. Calculate the cell stiffness (spring coefficient) based on its density
4. Calculate the compression indent based on the total pressure, hardness and stiffness
5. Calculate the new temporary cell density candidate based on the cell indent amount
6. Calculate final cell densities by resampling the columns, as well as total column heights and masses

The next subsections explain every step in detail. The snow cells are denoted $cell_i$, where $i$ is the index (number) of the cell counted from the top, e.g $cell_0$ is the top most snow cell that is in contact with air, $cell_1$ is the one below it and so on. All cells have equal and constant dimensions, their side size, or height is denoted by $h_{cell}$.

### 4.1.1 Pressure

For an $i^{th}$ snow cell, with n = the total number of cells per column, the total pressure acting on that cell is calculated as:

$$P_i = F_{i,mg}/A_{cell} + P_{i,ext} \tag{4.1}$$

with

$$F_{i,mg} = \sum_{k=0}^{n} m_k \cdot g \tag{4.2}$$

and

$$m_k = \rho_k \cdot V_{cell} \tag{4.3}$$

where $P_i$ is the total cell pressure [Pa], $F_{i,mg}$ is the total gravitational force acting on the cell [N], A is the cell base area $[m^2]$ which is equal to $h_{cell}^2$ and $P_{i,ext}$ is the externally applied pressure [Pa]. $F_{i,mg}$ is calculated from the sum of masses of all the overlying cells, including the current one, multiplied with the gravitational constant g = 9.81 $m/s^2$. A cell's mass is calculated as its average density multiplied by the constant cell volume $V_{cell}$ = $h_{cell}^3$ $[m^3]$.

### 4.1.2 Hardness and the spring coefficient

The snow hardness is calculated using the direct formula 2.5 proposed by Barakhtanov et al.

$$\begin{aligned} H(\rho, T) = &-0.16599 + 0.542153 \cdot \rho - 0.08925 \cdot T + 0.447667 \cdot \rho^2 + 0,159256 \cdot \rho \cdot T \\ &- 0.01336 \cdot T^2 + 0.261043 \cdot \rho^3 - 0.15863 \cdot \rho^2 \cdot T + 0,022601 \cdot \rho \cdot T^2 - 0.00032 \cdot T^3 \end{aligned} \tag{4.4}$$

with H, $\rho$ and T denoting the snow hardness [Pa], density $[kg/m^3]$ and temperature $[°C]$, respectively.

The spring coefficient k uses the modified version of formula 2.7:

$$k = \rho^{n_1} \tag{4.5}$$

Where k is the spring coefficient [Pa/m], $\rho$ is the (average) density $[kg/m^3]$, $n_1 = 3.56$ was estimated during prototyping, by making sure the density curve does not compress indefinitely while still showing appropriate range.

### 4.1.3 Compression indent

The new cell density can be calculated as the result of change in the snow volume after compression, under assumption that the snow mass is constant per column (requirement 8). The change in the snow volume is dictated by the indent/compression amount x, resulting from the external vertically applied force and gravitation. Based on the formulas 2.1 and 4.5, and the observations from fig 2.3 the vertical indent of snow under vertically applied force/pressure can be described with the equation:

$$x = min(P - H, 0)/k \tag{4.6}$$

where x is the indent amount [m], P is the pressure acting on the snow cell [Pa], H is the snow hardness [Pa] and k is the spring coefficient [Pa/m].

### 4.1.4 Compressed density

The compressed density of a cell $cell_i$ is calculated as a function of the cell indent x, and assigned back as the temporary partial density of that cell. Knowing that for a constant mass, the volume and the density are in reverse proportion, and ignoring adjacent cells, the new density can be calculated as:

$$\frac{V_i}{V_i'} = \frac{h_{cell}^3}{h_{cell}^2 \cdot (h_{cell} - x)} = \frac{h_{cell}}{h_{cell} - x}$$

Therefore,

$$\rho_i' = \rho_i \cdot \frac{h_{cell}}{h_{cell} - x} \tag{4.7}$$

Where $V_i$ is the cell volume before compression $[m^3]$, $h_{cell}$ is the cell side size [m], $V_i'$ is the partial volume after compression $[m^3]$, $\rho_i$ and $\rho_i'$ are the old and the new cell

densities $[kg/m^3]$, before and after compression respectively. Figure 4.1 gives a visual representation of this calculation.

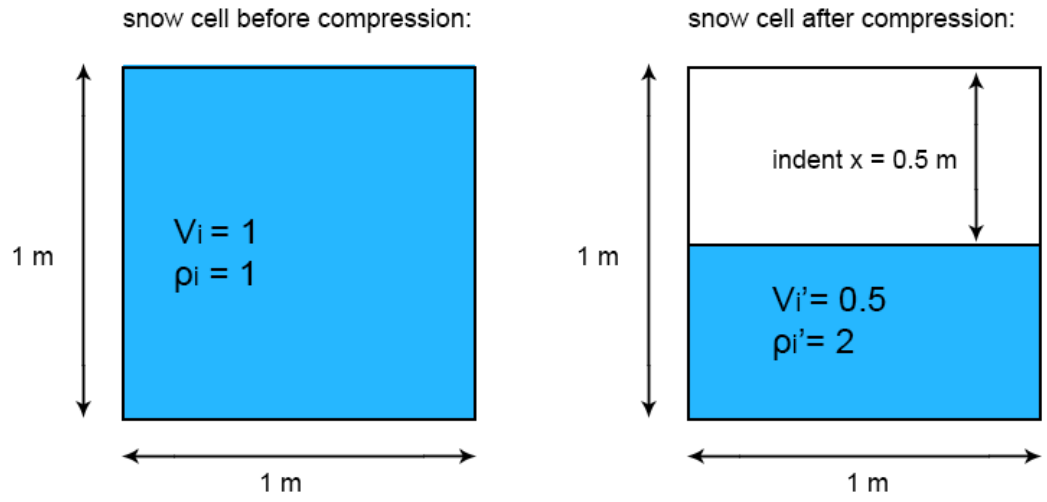



Figure 4.1: Visualization of the compressed density calculation for an example cell of $h_{cell} = 1m$ and a compression indent $x = 0.5m$, based on equation 4.7. $V_i$,$\rho_i$,$V_i'$ and $\rho_i'$ are the variables defined in equation 4.7.

The new density value $\rho_i'$ is a temporary value, and is only true for the partial cell volume $V_i'$. The average call densities are calculated by resampling all cells from bottom to top, which is described in the last step.

### 4.1.5 Column resampling

After storing the calculated temporary densities for each cell together with their indents, each column is iterated from bottom to top to calculate the final cell masses and densities, update the total heights and masses of each column. The average density of the current cell is calculated as $\frac{m_{cell}}{V_{cell}}$, where $m_{cell}$ is the total cell mass, and $V_{cell}$ is the cell volume. The total cell mass is calculated by combining the product of the partial density and partial volume of the current cell with products of densities sampled from the upper cells and their partial volumes until the volume of the current cell is filled up, or until the current $\frac{cellmass}{cellvolume}$ ratio reaches the maximum density that a cell is able to hold. Figure 4.2 demonstrates this process. The total column masses and heights are calculated as

shown in equations 4.8 and 4.9.

$$m_{column} = \sum_{k=0}^{n} m_{cell,n-1-k} \tag{4.8}$$

and updates its height by subtracting all of the cell indents from the initial height value

$$h'_{column} = h_{column} - \sum_{k=0}^{n} x_{n-1-k} \tag{4.9}$$

Here, $m_{column}$ is the total mass of the snow column [kg], $m_{cell}$ are individual cell masses that the column consists of. $h'_{column}$ and $h'_{column}$ are the updated and the old total column heights [m] respectively, and x are the individual cell indents [m].

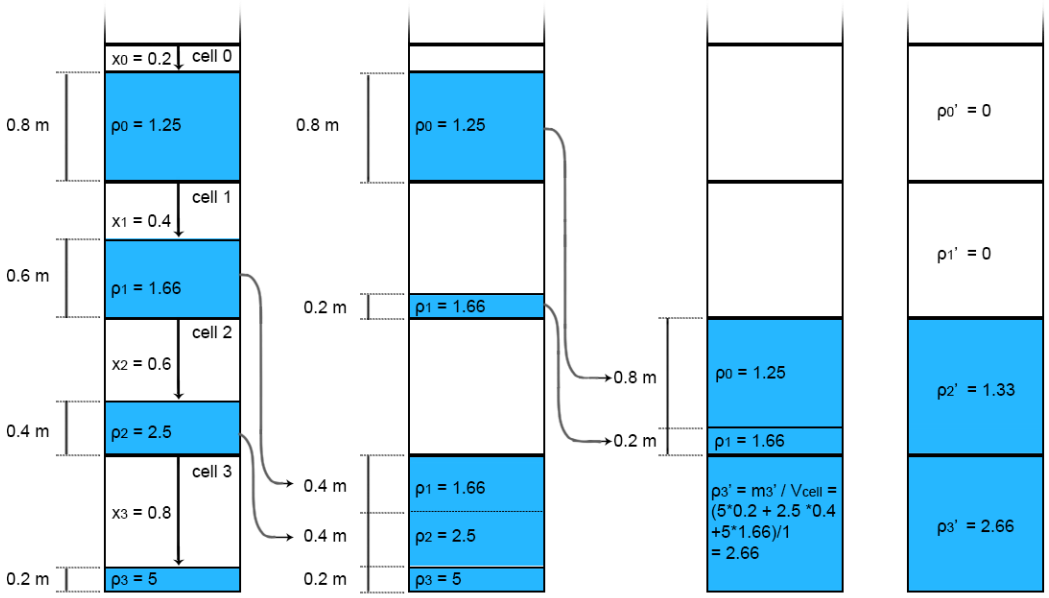



Figure 4.2: Visualization of the cell density resampling process. The densities $\rho_{0,1,2,3}$ are the partial densities $[kg/m^3]$ calculated during the previous step 4.1.4 for cells 0-3, $x_{0,1,2,3}$ are example indents [m], $\rho'_{0,1,2,3}$ are the final average densities $[kg/m^3]$, $m'_3$ is the average mass [kg] of cell 3, $V_{cell}$ is the constant cell volume $[m^3]$ (in this example, $Vcell = 1$ for a cubic cell of height 1 m).

This calls the end of one simulation cycle. The simulation continues to iterate until all cells become stable. A cell can reach stability either by arriving at the maximum snow density or having its hardness outgrow the active pressure.

## 4.2 Prototyping in Python

The above algorithm was tested on a singe column with a total height of 8 m. The column was partitioned into 50 cells, each of height 0.04 m. The simulation was run for multiple cycles using:
start density of fresh snow = 20 $kg/m^3$
snow temperature = - 3 $°C$
and $n_1$ = 3.56 for equation 4.5. The x in the equation 4.6 is capped at 99% of the cell height to avoid division by zero and overflows. Due to the iterative nature of the simulation, the error produced by the clamp operation can be neglected.
Figure 4.3 demonstrates the effects of the 1st cycle on the initial fresh density. With this configuration, it takes about 2 meters for the weight pressure to exceed the snow hardness and start compressing the underlying snow cells, which can be seen in the first (left) subplot in figure 4.5. Most of the compression happens during the first cycle (figure 4.3, right) and with each consecutive cycle the changes in densities become less and less significant. This decay is shown in figure 4.4 that combines the density curves after cycles 1,2,5 and 10.

The pressure and hardness curves plotted against each other during cycles 1,2,5 and 10 can be seen in figures 4.5 and 4.6. These plots demonstrate how with each cycle an increase in density leads to increase in hardness (constraint 5). Due to the polynomial relation between density and hardness, the hardness curve completely outgrows the weight pressure after cycle 10, stopping the density curve from any further changes. This ensures that even when being far from reaching the maximum cell density limit, the average cell density will not increase indefinitely, regardless of how many cycles are run.

Another test run was conducted by lowering the temperature and keeping the start density as it was: start density of fresh snow = 20 $kg/m^3$
snow temperature = - 10 $°C$
This configuration resulted in a far lower impact of the weight pressure due to increased snow hardness, fulfilling constraint 6. Figure 4.7 compares the density behavior for different temperatures, showing that it now takes more snow volume for its weight pressure to exceed the snow hardness. The density behavior of cycles 1,2,5 and 10 can be seen in figure 4.8.

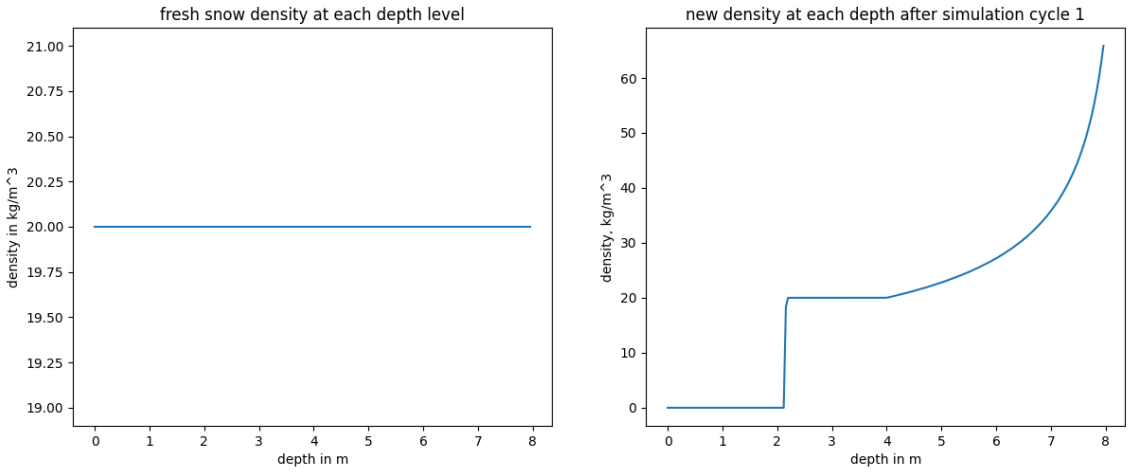



Figure 4.3: The fresh density before the simulation start (left) and the density gradient after the 1st cycle (right). The column total height is compressed by 2 meters.

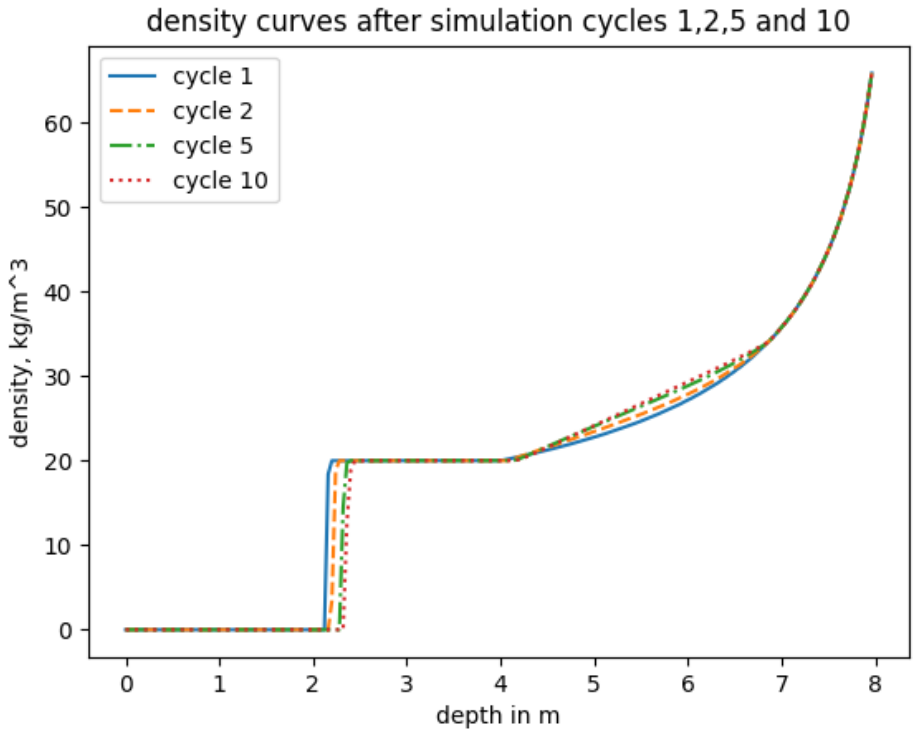



Figure 4.4: Density curves after cycles 1,2,5 and 10, for start density = 20 $km/m^3$ and temperature = -3 $°C$.

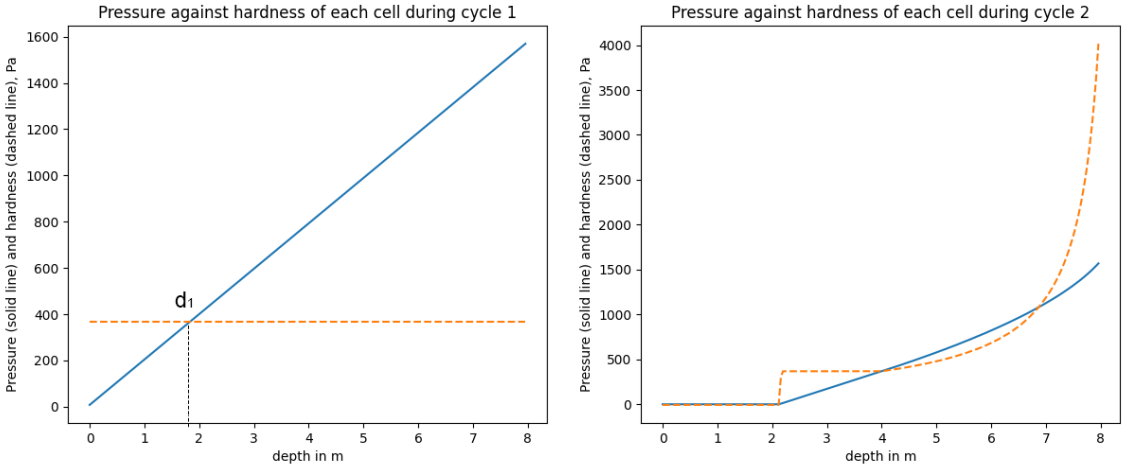



Figure 4.5: The pressure and hardness curves plotted against each other during cycles 1 (left) and 2 (right). Density changes only happen at those levels of depth where the pressure value (solid line) is greater than the hardness value (dashed line). For cycle 1 it is the entire length after point $d_1$, where the weight pressure exceeds the snow hardness.

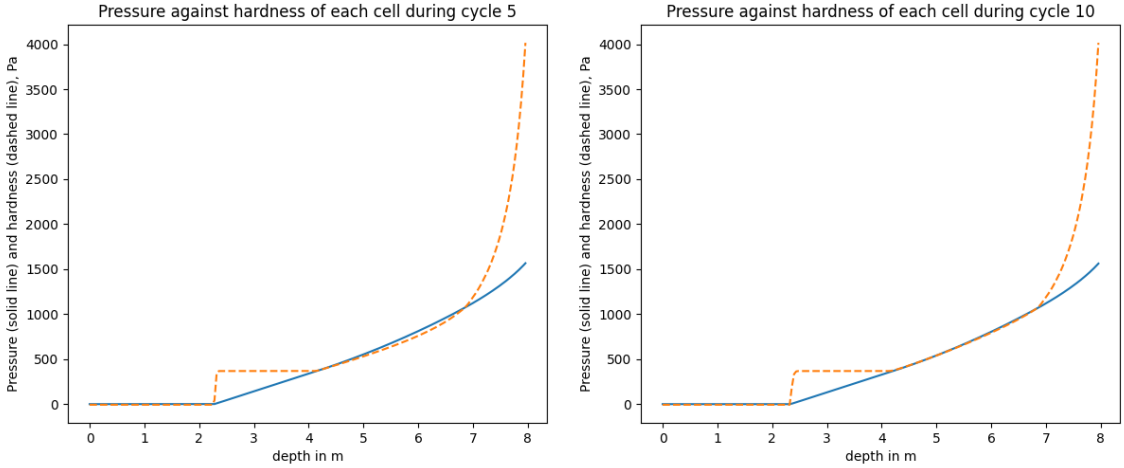



Figure 4.6: The pressure and hardness curves plotted against each other during cycles 5 (left) and 10 (right). As the hardness curve catches up with the weight pressure, the density stabilizes.

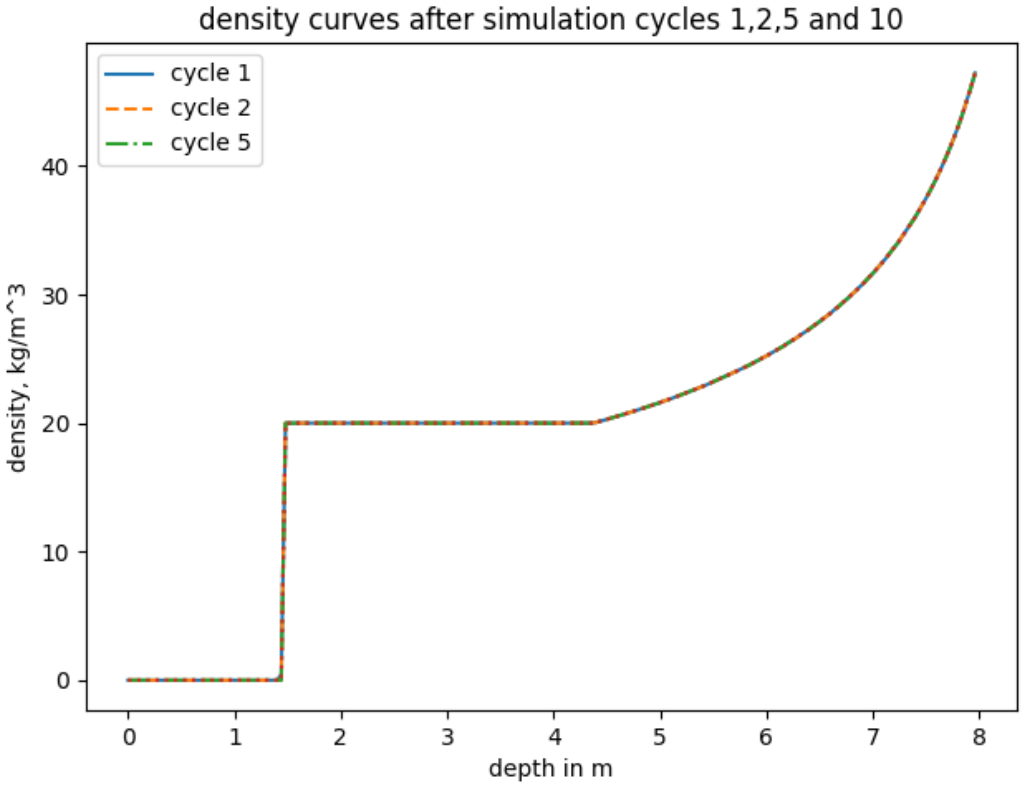



Figure 4.7: Density curves after cycles 1,2,5 and 10, for start density = 20 $km/m^3$ and temperature = -10 $°C$. The total column height is compressed by 1.5 meters.

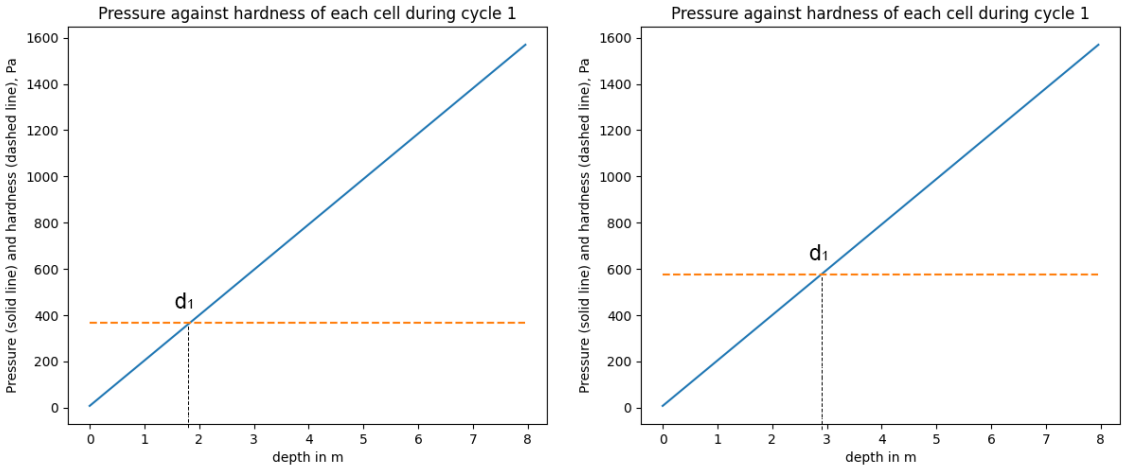



Figure 4.8: The pressure and hardness curves plotted against each other during cycle 1, for temperatures -3 $°C$ (left) and -10 $°C$(right). The point $d_1$ shifts further to the right for lower temperatures, indicating increase in snow hardness.

# 5 Implementation

This chapter covers the implementation of the practical part of the thesis (project). The Project Overview section is focused on listing the tools of choice and describing the overall project structure, followed by the Simulation and Visualization sections that cover the corresponding project features.

## 5.1 Project Overview

The practical part of the thesis is developed as a Unity project. As mentioned in the previous chapter, the simulation is implemented on a grid of cells, that represents a snowpack volume. The immediate visualization is done by rendering a grid of cubes, with colors indicating the different parameters, such as cell density, cell pressure, cell position, and so on.

### 5.1.1 Tools of choice

Below are listed the software tools that were used during the implementation together with their short descriptions:

**Unity 3D**
Software for independent game development, that provides a flexible and easy-to-use infrastructure for building games, including support for writing compute and custom pixel shaders. Moreover, the extensive documentation and manuals provided both by the official Unity website and third-party sources makes it very appealing to work with. All programming in Unity is done through C# scripting.

**Visual Studio**
Unity 3D does not provide its own code editing tool, however, supports external code editors and debuggers. Visual Studio is the default script editing software for Unity and can be attached to the Unity client instance for debugging during runtime.

**NSight**
NSight is a GPU debugging software that enables a frame-by-frame inspection of draw calls and retrieving data such as structured buffers from the GPU

**Python 3**
In addition to algorithm prototyping, Python scripts are used in this project for evaluating the data retrieved from the GPU.

### 5.1.2 Project Structure

The project implementation is split between the CPU and the GPU. The CPU-side code is responsible for the creation and routing of data, and user input control and is written in C#. The GPU part in turn comprises of a compute shader and a vertex-fragment shader, and is written in the shader-specific language HLSL. The object-class diagram in Figure 5.1 shows the relations between different project components, without going into very specific implementation details.

The code for all physics-related calculations resides in a single compute shader with multiple kernels, where each includes one or several simulation steps covered in the previous chapter. Since most of the calculations associated with the cell updates are self-contained and involve the same amount of steps, the parallelism offered by the compute shader threads is used for performance gains. This applies to cases where the grid iteration order does not matter, meaning that all the cells can be updated at the same time. In cases where the grid cells must be iterated in a specific order and rely on data from adjacent cells, the computation time can still benefit from parallelism by handling all the grid columns concurrently.

The code that is responsible for visual debugging and rendering of the 3d grid is in the vertex-fragment shader, associated with the grid material.

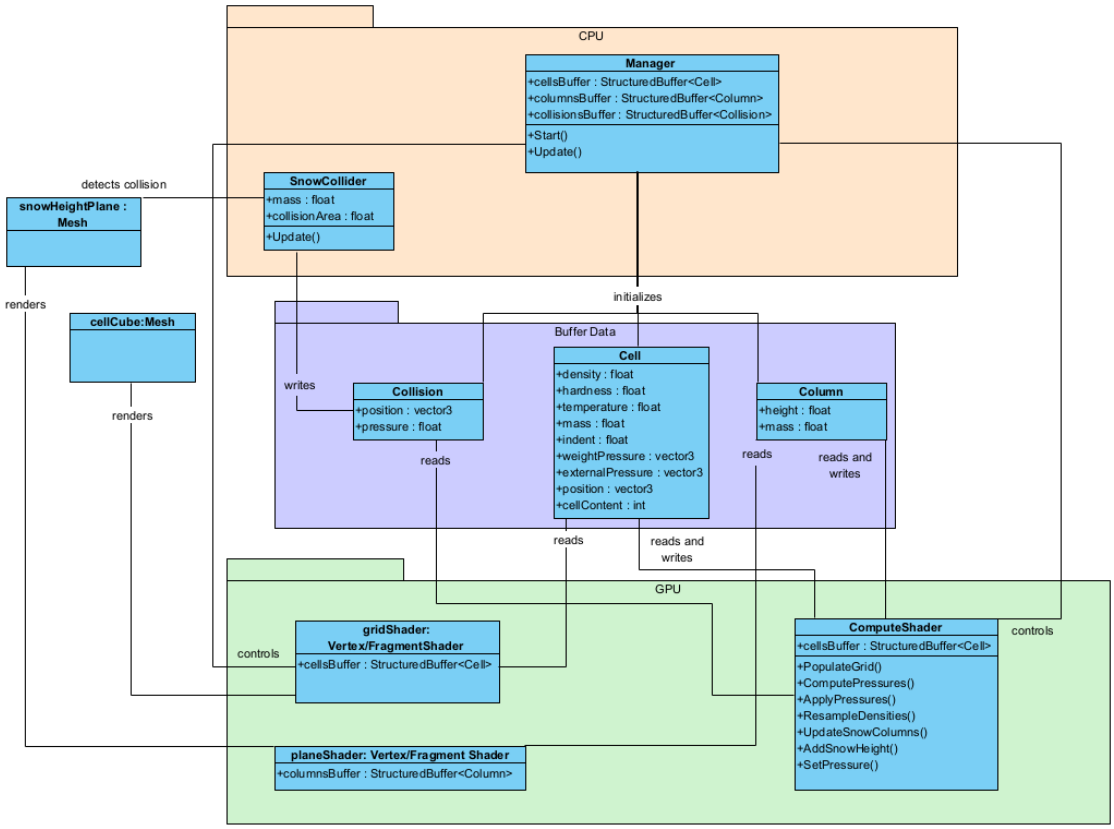



Figure 5.1: Object-class diagram of the project

### Data Structures

The data for holding input and output values covered in the previous chapter is encapsulated into structs, namely the cell data struct and the column data struct.

A *cell* struct contains the following data:

- Snow density
- Snow hardness
- Snow temperature
- Snow mass
- Snow indent

- Weight pressure
- External pressure

as well as some meta-data, containing information about the grid:

- Cell world position
- Cell content - snow, air or ground

The cell grid only lives as a concept and does not have its own dedicated struct. Instead, it is represented as an arrangement of n · n · n cells instanced with corresponding world coordinates, with n denoting the number of cells per grid dimension. This initial creation of cells and their arrangement is handled by the CPU-side code as well. The grid is defined by its initialization parameters which are listed below:

- cell side size
- cell number along the x-axis (grid width)
- cell number along the z-axis (grid depth)
- cell number along the y-axis (grid height)

A grid contains 2 types of cells: air cells and snow cells. If a cell is marked as ground or air, it will be skipped during most of the simulation-related computations.

A *column* struct looks as follows:

- Snow column height
- Snow column mass

It is important to differentiate between a column buffer - which is a collection of column data structs, and a grid column - which is a conceptual representation of a vertical stack of cell structs.

Additionally, there is a special data struct for holding *collision* information:

- World position (of affected cell)
- Pressure

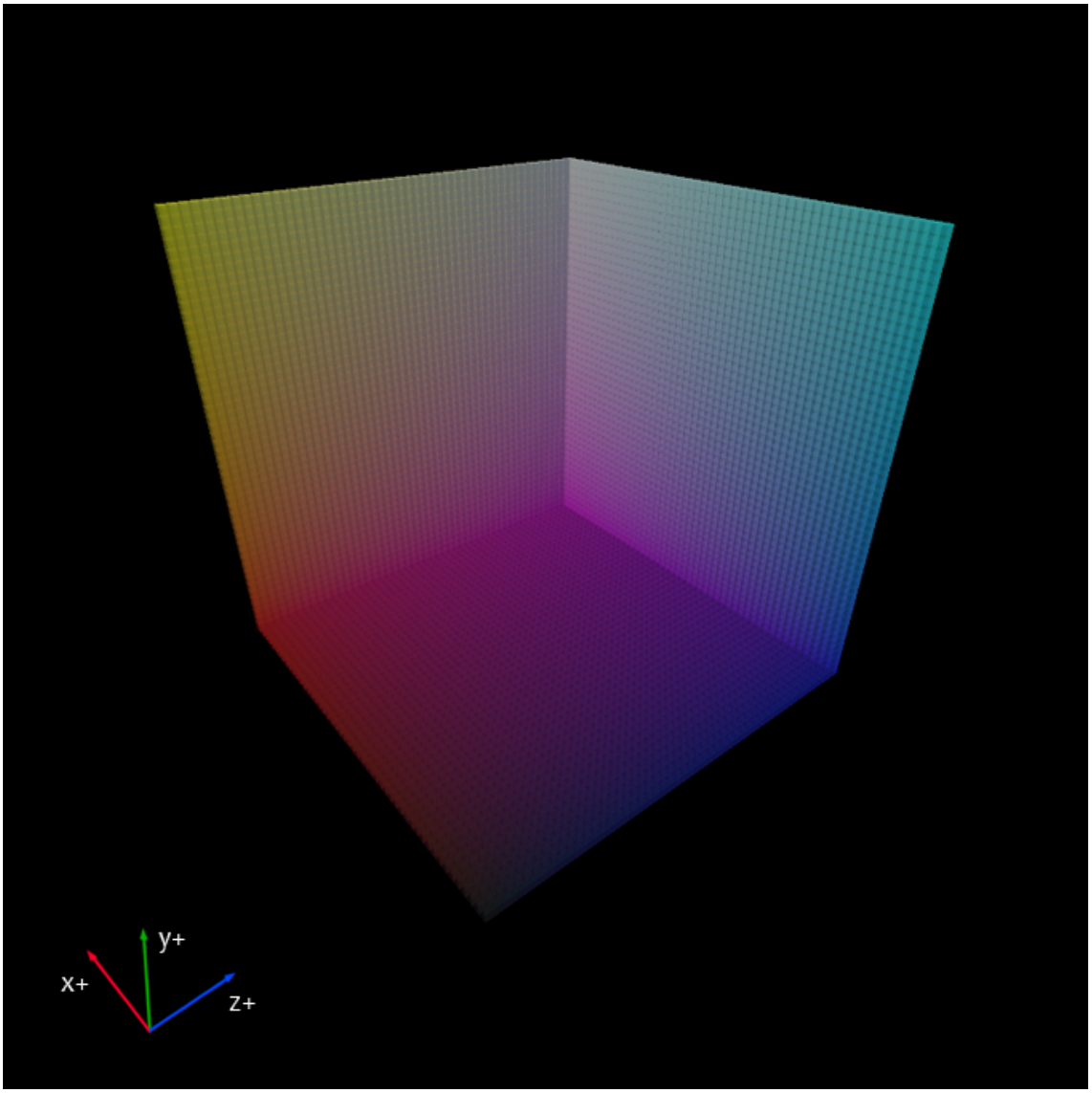



Figure 5.2: The 3d grid consisting of $50 \cdot 50 \cdot 50$ cells, with their world positions represented by color. The gizmo in the bottom left corner indicates the world coordinate system.

### CPU

The CPU-side code lives in a C# script with the Manager class that contains all the initialization logic, is responsible for data routing between the CPU and the GPU, and the command execution order during every frame update. By default, all newly created C# scripts inherit from Unity's MonoBehavior class, which provides the following overrid-

able functions: Update() and Start(). The Update() function executes every frame. The Start() function is executed only once in a script's lifetime before any of the Update() calls. The Manager class, as well as the SnowCollider, have their initialization logic nested in the Start() function. At initialization time, the cell grid is created, with each cell's parameters set to 0, except for the density, which is set to freshSnowDensity from table 5.1. All data structs are routed between the CPU and the GPU using structured buffers. The Manager script is in charge of authoring updates on those buffers every frame, i.e. 1 frame = 1 simulation cycle. It also controls the simulation and visualization by passing over some changeable parameters to the GPU (like simulation speed, grid output toggles, and so on). Simulation dispatch calls, as well as the control parameters updates happen in the Update() function. Table 5.1 refers to such simulation control parameters and their default values.

Table 5.1: Global parameters

| **parameter** | **controls** | **initial value** | **applies at** |
|---|---|---|---|
| | *grid related* | | |
| cellSize | cell height/side size | 0.2 $m$ | init time |
| gridWidth | cells along x-axis | 50 | init time |
| gridDepth | cells along z-axis | 50 | init time |
| gridHeight | cells along y-axis | 50 | init time |
| | *simulation related* | | |
| cellV | cell volume, derived from cellSize | 0.008 $m^3$ | init time |
| kN | density exponent for formula 4.5 | 3.56 $kg/m^3$ | run-time |
| freshSnowDensity | fresh snow density | 20 $kg/m^3$ | init time |
| maxSnowDensity | maximum snow density | 100 $kg/m^3$ | init time |
| airTemperature | air cells' temperature | -3 $°C$ | run-time |
| addedSnow | height of freshly added snow | 5.8 $m$ | run-time |
| timeScale | simulation speed | 1 | run-time |
| | *visualization related* | | |
| showGrid | grid visualization toggle | true | run-time |
| showDensity | density visualization toggle | true | run-time |
| showTemperature | temperature visualization toggle | false | run-time |
| showPressure | pressure visualization toggle | false | run-time |

### Collision and external pressure

The CPU-side code also detects the collision of objects with the snow surface, and logs which grid cells are affected and the external pressure per cell. In order to detect and correctly pass the collision information a SnowCollider class is implemented. That class holds a reference to the collision buffer, as well as contains parameters that define the collider itself, like its mass and the collision area. When the collision happens, the snow collider fills the buffer with all generated collisions. Those values will be used during the simulation for calculating the total vertical pressure. There are 3 snow collider objects present in the scene, represented by 3 differently sized cuboids of same mass (figure 5.3). The cuboids with their defining parameters are listed in table 5.2.

Table 5.2: In-scene snow colliders

| | **mass** | **collision area** |
|---|---|---|
| **Collider 1** | 100 kg | 16 $m^2$ |
| **Collider 2** | 100 kg | 1 $m^2$ |
| **Collider 3** | 100 kg | 0.1 $m^2$ |

The external pressure that is received by each colliding cell of the snow surface is calculated as the collider's gravitational force divided by its total contact area with the snow, here referred to as collision area.

#### GPU

After the cell and the column buffers are created on the CPU, their references are passed down to the GPU. The compute shader executes the simulation cycles, triggered by the Manager script every frame, and updates the buffers with newly calculated values. The vertex/fragment shaders are assigned to the rendered geometry and then use those values for visual output.

## 5.2 Simulation

The compute shader handles all the simulation-related work. The simulation algorithm described in section 4.1 is implemented by 5 compute kernels. A compute kernel refers to a specific function or program that is executed on a GPU's compute units to perform parallel computations. Most kernels involved in this simulation cover one or more steps

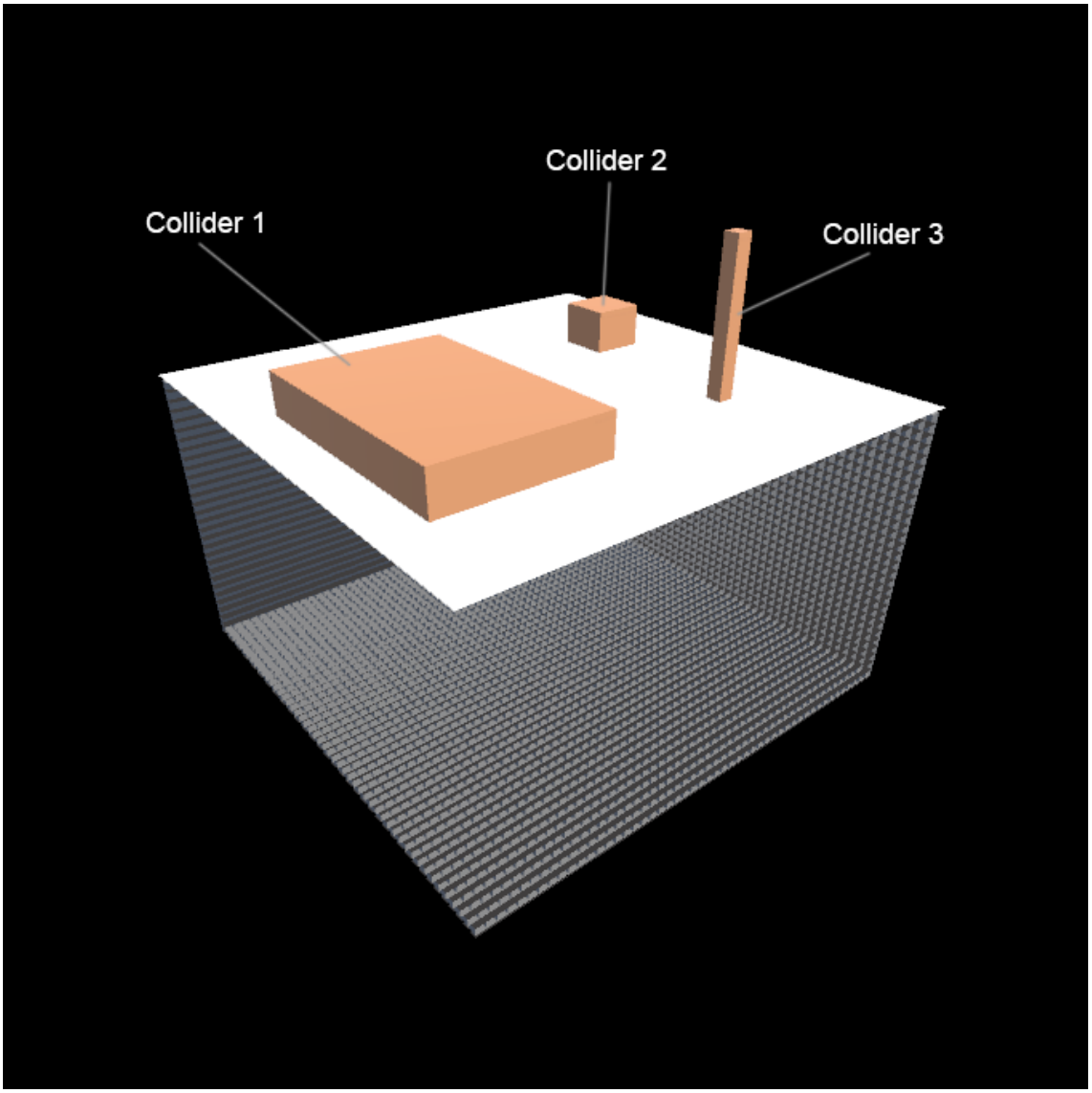



Figure 5.3: Snow colliders 1,2 and 3 from table 5.2, in their initial positions before the simulation start - their volumes almost entirely above the snow surface.

of the algorithm. Sometimes, due to different mapping between the buffer indexes and kernel threads, a step is split between kernels. Additionally, there are 2 helper kernels that are dispatched on demand by the Manager script. For referencing convenience, the helper kernels will be covered here first:

**AddHeight**

This kernel simply adds the value addedSnowHeight, supplied by the CPU side to all the

column heights in the columns buffer. The updated structured buffer will then be used for adding fresh snow cells.

**SetPressure**
This kernel is dispatched every time a collision with the snow surface is detected. It reads from the collisions buffer that had been updated by the SnowCollider and sets the external pressure of all the affected cells in the cells buffer.

The simulation kernels are dispatched every frame. Each kernel thread is run per grid cell unless stated otherwise. Below is a detailed breakdown of every kernel in the same order as they are dispatched by the Manager script's update call:

**PopulateGrid**
The first kernel fills out all the cell parameters and prepares them for simulation, based on their content (label). The total snow height is sampled from the columns buffer. The cells are labeled "air" if they are above the snow height and "snow" otherwise. The following cell properties are calculated and stored:

**Cell temperature** Air cells always sample their temperature from the controllable airTemperature value. If a cell is freshly labeled as "snow", it samples its temperature from airTemperature and keeps it for as long as it remains labeled "snow", meaning that for older "snow" cells the temperature remains constant.

The following properties are calculated only for "snow" cells, and for "air" cells are set to 0:
**Cell density** For fresh "snow" cells density is sampled from freshSnowDensity, for old "snow" cells remains intact.
**Cell hardness** Hardness is calculated and stored based on formula 4.4.
**Cell mass**: Mass is calculated by a simple relation $\frac{celldensity}{vCell}$.

In the next kernels, the calculations are only carried out for "snow" cells.

**ComputePressures**
This kernel is dispatched per grid column, instead of a grid cell, and fully covers the step 4.1.1 of the simulation algorithm. Within the column, it iterates through all cells from top to bottom to calculate and store the total acting pressure (formula 4.1). The

weight pressure is calculated during the for loop iteration, via formula 4.2. The external pressure is sampled from the corresponding cell's external pressure property, where it had been added by the SetPressure() kernel.

**ApplyPressures**
This kernel combines the algorithm steps 4.1.3 and 4.1.4. The spring coefficient is calculated on the fly via formula 4.5. The indent from compression is calculated via formula 4.6. In order to make use of the discretized time steps while still retaining the control over the simulation speed, the result x is multiplied with the scaled timeStep value. The full modification then looks like the following:

$$x^{'} = min(x * timeStep * timeScale, cellSize * 0.99) \tag{5.1}$$

where x is the indent per cell [m], $cellSize$ is the cell side size/height, timeStep is the time passed since the last simulation cycle/frame [s], supplied by Unity, and timeScale is the simulation speed controller. The partial density is calculated via formula 4.7 and stored as the temporary density value of that cell. It will be updated for the correct, final one by the next kernel.

The step 4.1.5 from the algorithm is split between the last two kernels:

**ResampleDensities**
This kernel covers the density resampling part, by, again, iterating grid columns, this time from bottom to top. The correct densities are calculated as illustrated in figure 4.2.

**UpdateSnowColumns**
The last kernel simply calculates the corresponding total column height and mass, based on formulas 4.9 and 4.8 respectively for each column struct in the columns buffer. Based on these height values, the cells that are no longer within the snowpack volume will be labeled as "air" during the next PopulateGrid() kernel execution.

## 5.3 Visualisation

The cell grid rendering is done with a help of a vertex-fragment shader that reads data from the cells buffer and renders each cell according to its properties. Additionally, there

are toggles controlled from the CPU side, listed under the visualization tag of table 5.1. This is useful for visual debugging and quick result assessment.

Additionally, there is another vertex-fragment shader applied to a horizontal plane, that offsets the plane's vertices based on the sampled heights from the snow column buffer. This is done to visually output the column heights when the grid visualization is turned off. Different parameters with the initial configuration in table 5.1 are visualized in figures 5.4, 5.5 and 5.6.

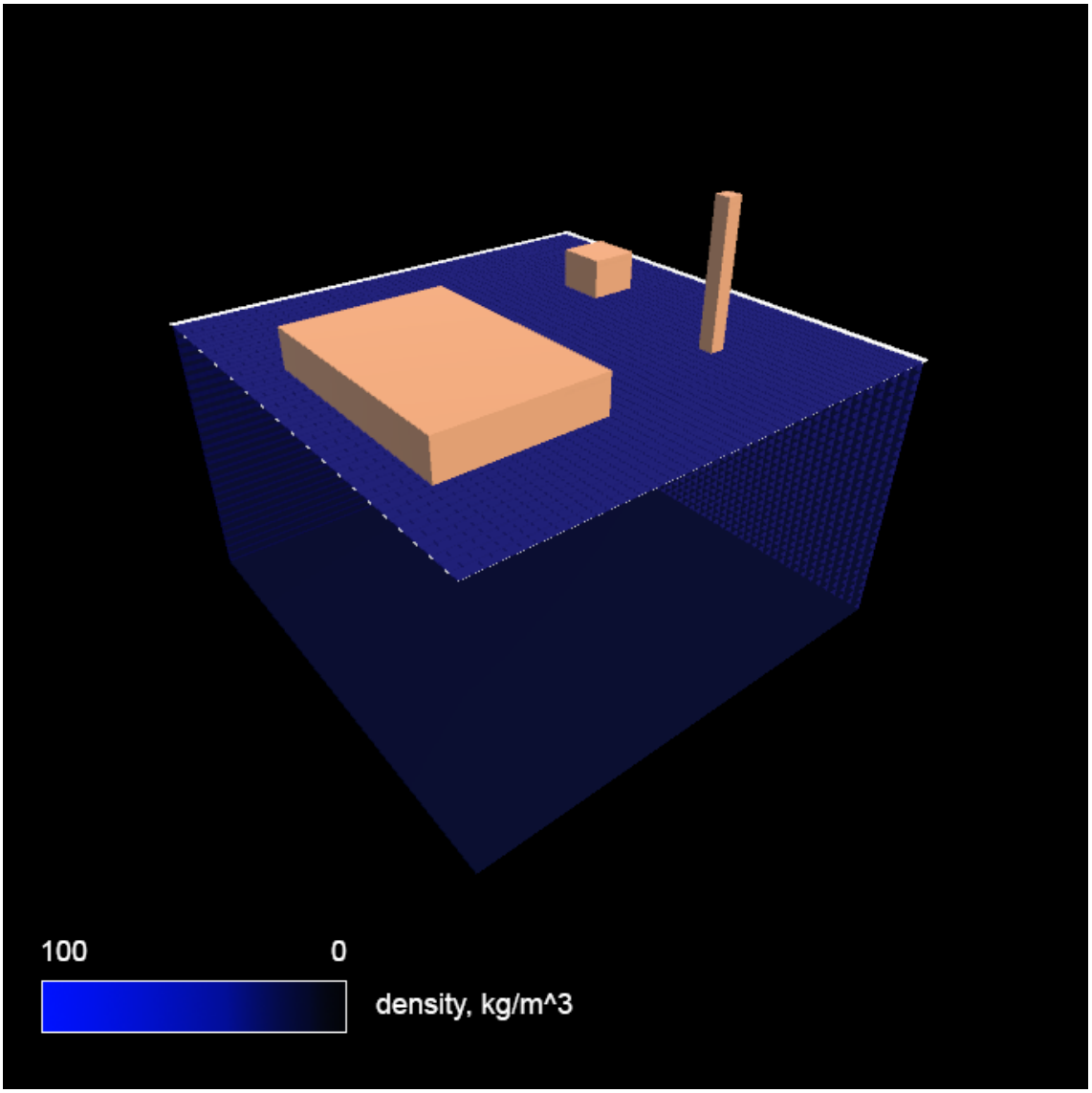



Figure 5.4: Visualization of grid density, for cells where density is greater than 0. The color intensity represents the percentage of the maximum density. In this example, the uniform fresh density before the 1st cycle is visualised as a ratio $\frac{20kg/m^3}{100kg/m^3}$. The screenshot is taken from the implemented demo project.

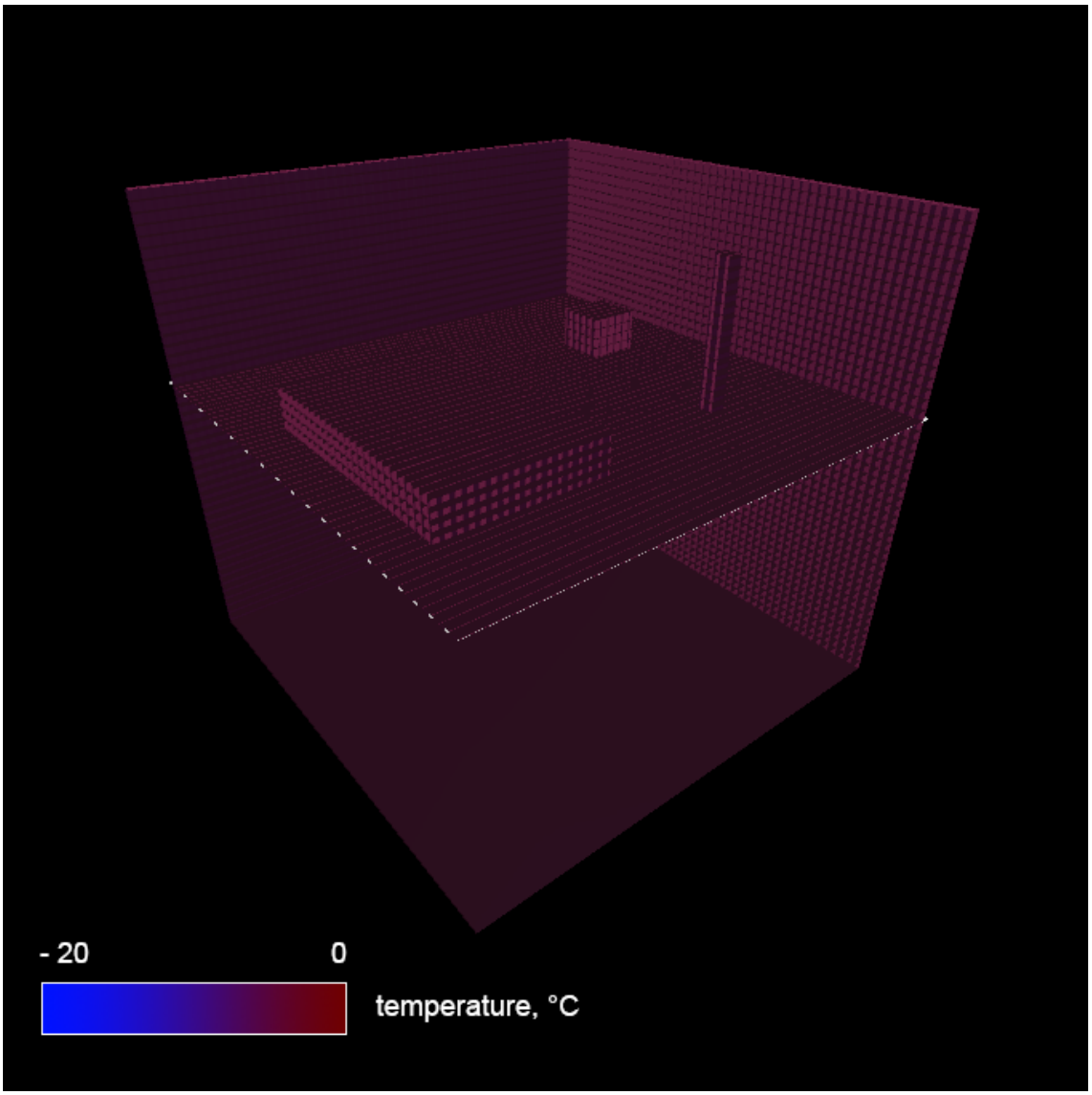



Figure 5.5: Visualization of grid temperature. The color hue represents temperature range from 0 to -20; red to blue respectively. This example visualizes the start temperature of -3 $°C$ before the 1st cycle. The screenshot is taken from the implemented demo project.

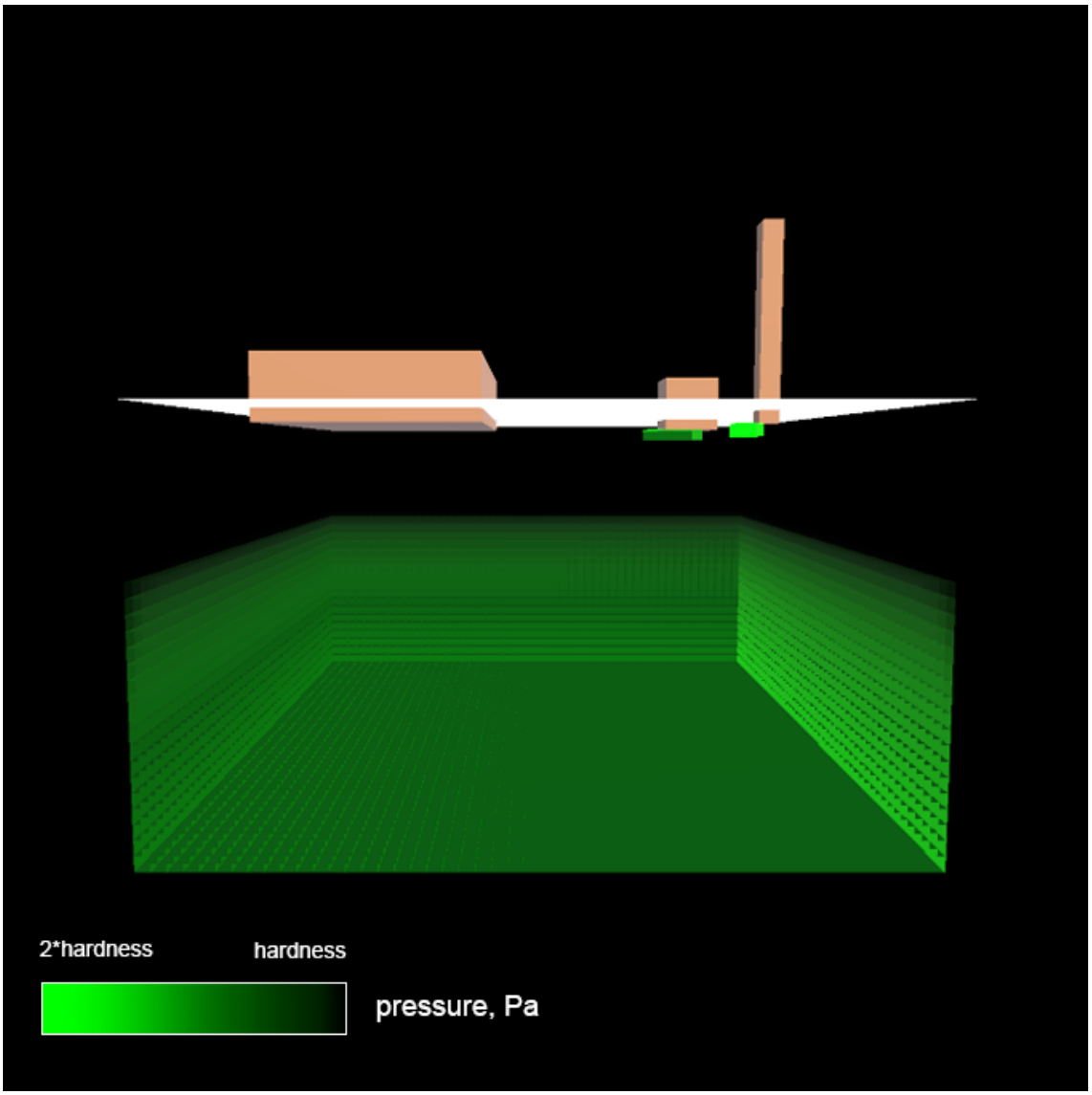



Figure 5.6: Visualization of total active grid pressure where it exceeds the hardness, for the initial values in table 5.1. The screenshot is taken from the implemented demo project.

# 6 Evaluation and Tests

In this chapter the results of the simulation are evaluated. Each section of this chapter covers a separate test with its initial conditions and results, that correspond to the use cases described in the Requirements chapter (chapter 3). In total, there are 3 tests, with the first one being a general assessment of the expected behavior with the default initial configuration that is shown in table 5.1, and split into 2 sub-tests. The other 2 tests dive into the specifics of different temperature configurations and their effects on the resulting density profile. Once the simulation reaches visual stability, the cell buffer data is retrieved for evaluation via the GPU debugging software NSight and is plotted in Python, the plots are then presented alongside the visual results from the project build window.

## 6.1 Uniform temperature

The first test aims at evaluating the compression of snow under conditions of uniform initial density and temperature. This test includes 2 simulation runs (test 1.a and test 1.b), one for a higher and another one for a lower value of the initial temperature, with the rest of the configuration copied from table 5.1. The defining input parameters of each sub-test are reiterated in table 6.1. The same table lists the tracked output variables that characterize the overall compression response and give some assessment criteria. Figures 6.1 and 6.2 demonstrate the start, half-point and stable states of simulation for sub-tests a and b, respectively. In both cases the snow colliders show similar expected behavior: collider 1 not moving due to its weight pressure not being enough to exceed the snow hardness, collider 2 indenting the snow surface underneath it just a little bit and collider 3 being almost entirely submerged into the snowpack by the end of the simulation. Figures 6.1.d and 6.2.d both show a slight density gradient that increases towards the bottom, however the exact numerical values given in figure 6.3 indicate a clear difference in the behaviors of snow hardness and density in both cases. Figure

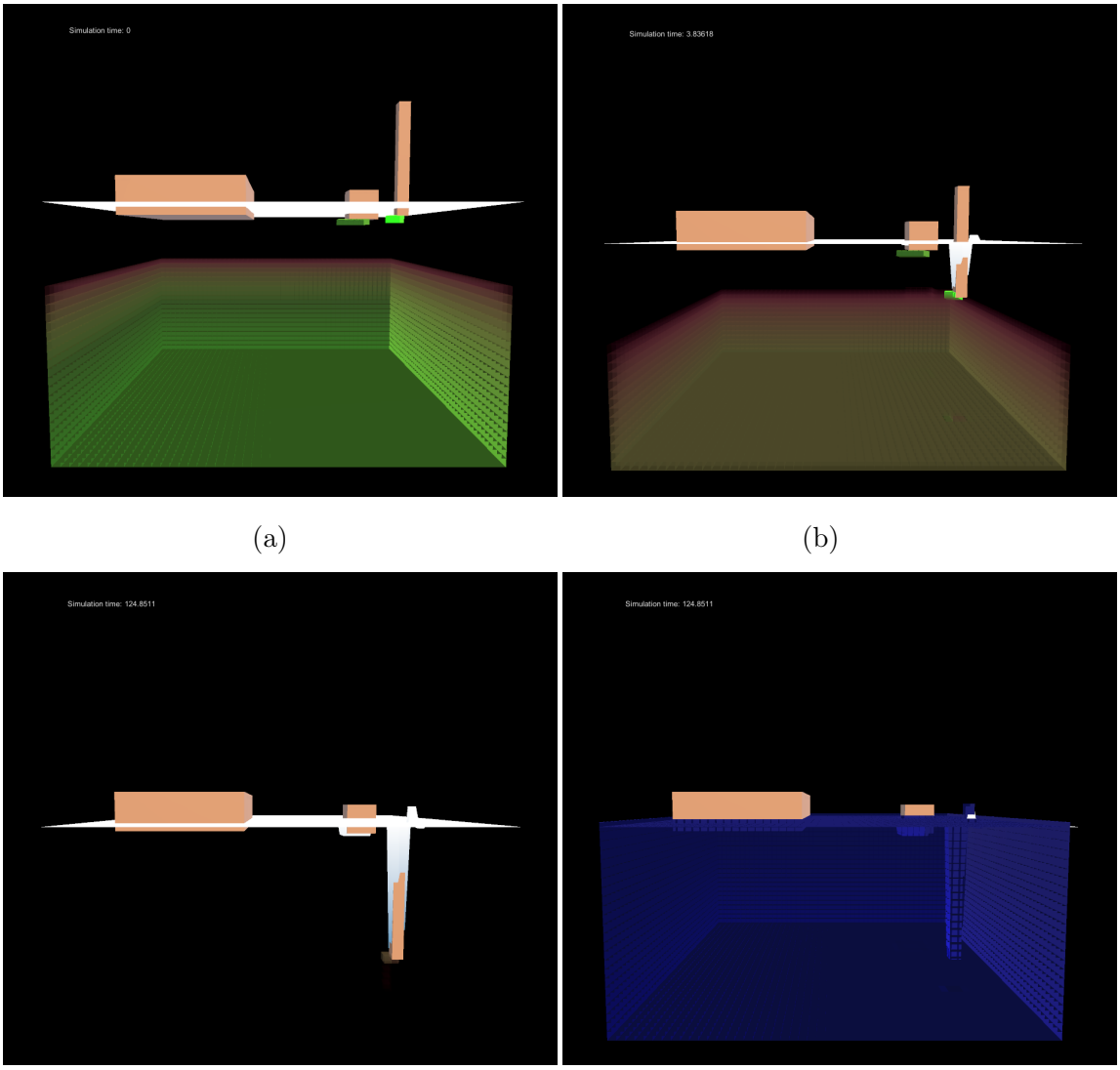
6.3.b shows lower values for density and higher values for hardness across the entire snow column, as well as the compression starting as a deeper level for test 1.b than the results of test 1.a shown presented in figure . Thus, the results of 2 simulation runs, authored under the same conditions except for the temperature, show less compression for a lower temperature value. The tracked output variables in table 6.1 show that despite the column resampling some artificial mass increase is still present, leading to failed mass conservation requirement. The incorrect mass accumulation persists through-out all tests. Though neglectable, this discrepancy is evident of one major drawback of a grid-based implementation in 3D and that being the non-trivial mass transferring problem. Moreover, there is an artifact of an incorrectly added height in vicinity of the indent from the thinnest collider (collider 3), that is likely tied to a mis-match in coordinates mapping between the affected cell and the height map.

Table 6.1: Test 1 parameters

| *Input parameters* | | **Test 1.a** | **Test 1.b** |
|---|---|---|---|
| **Fresh snow density** | | 20 $kg/m^3$ | 20 $kg/m^3$ |
| **Temperature** | | -3 $°C$ | -10 $°C$ |
| *Output parameters* | | **Test 1.a** | **Test 1.b** |
| | Before sim. | After sim. | After sim. |
| **Column mass** | 4.64 kg | 4.92 kg | 4.66 kg |
| **Column height** | 5.8 m | 4.8 m | 5.4 m |
| **Column avg. density** | 20 $kg/m^3$ | 24.39 $kg/m^3$ | 21.57 $kg/m^3$ |

## 6.2 Temperature gradient - bottom-up

The bottom-up temperature gradient was achieved by initializing the cell grid with their the temperatures decremented by 2 $°C$ every 0.8 meters, starting with -3 $°C$ at the bottom. Table 6.2 lists the input parameters as well as the tracked output parameters for this test. The resulting start temperature profile is shown in figure 6.4.a. Figures 6.4.b and 6.4.c show a relative fast compression rate in comparison to the previous tests. Though the amount of total height decrease is considerably lower than for test 1.a and exceeds that of test 1.b only by 0.2 meters, the compression at the bottom of the grid, where cells have the lowest hardness and are exposed to highest weight pressure, is visibly higher than of test 1, as shown by the stable density profile in figure 6.4.d. This is backed

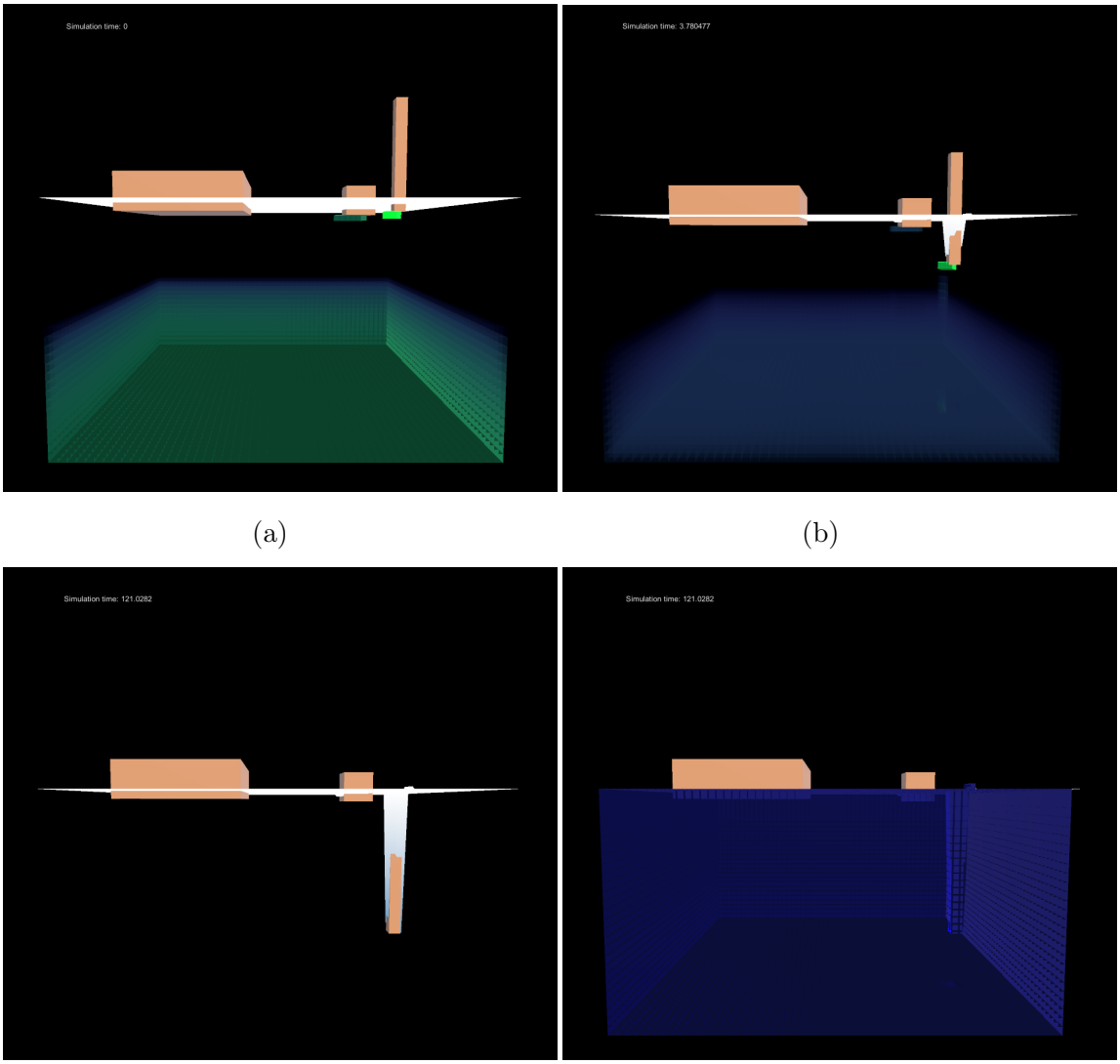



(a) (b)

(c) (d)

Figure 6.1: Visualization of simulation run for test 1.a, taken from the implemented demo project. Screenshots show the pressure profile mixed with the current temperature of -3 $°C$ at simulation time = 0 s (a), 30 s (b) and approx. 2 minutes (c). Sub-figure (d) shows the stable-state density gradient

by the plots in figure 6.7, where at deeper levels the snow density reaches its highest values quite fast.



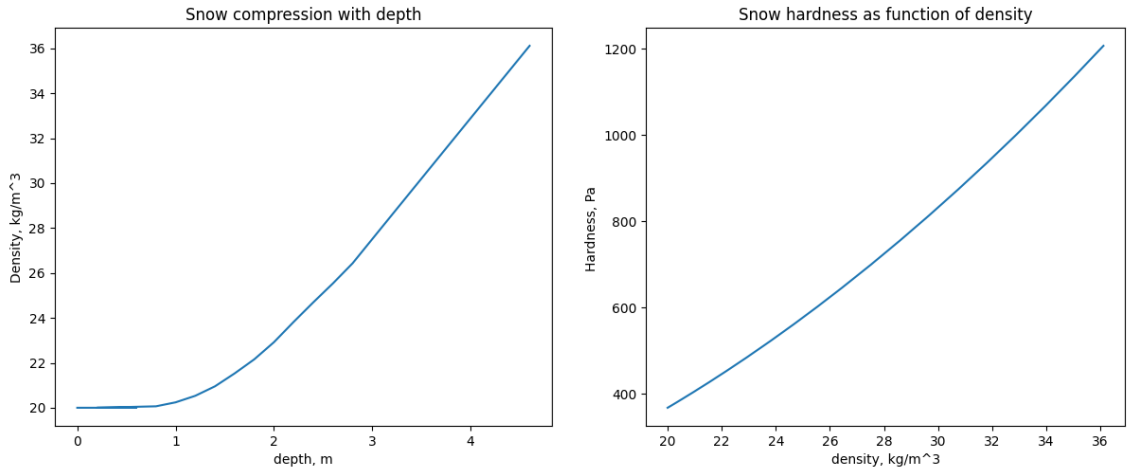

(a) (b)

(c) (d)

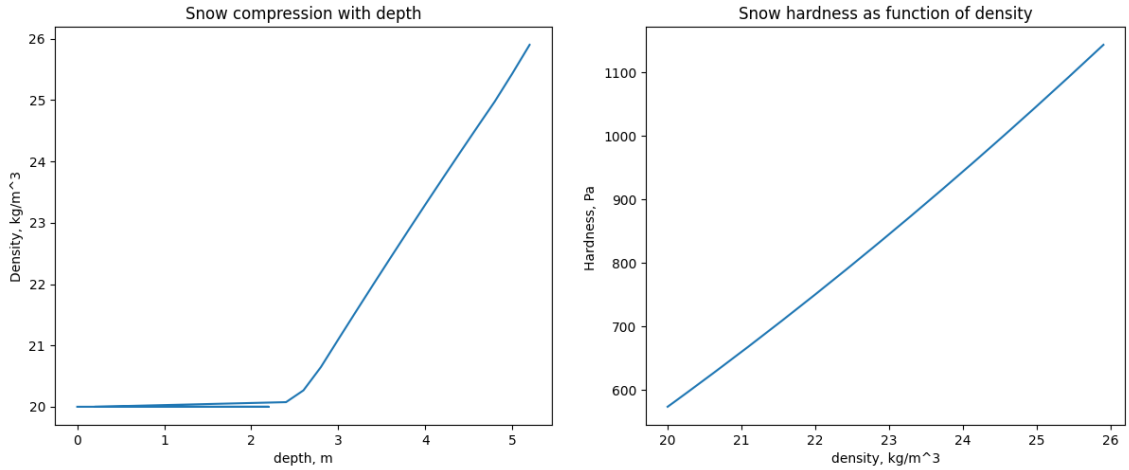
Figure 6.2: Visualization of simulation run for test 1.b, taken from the implemented demo project. Screenshots show the pressure profile mixed with the current temperature of -10 $°C$ at simulation time = 0 s (a), 30 s (b) and approx. 2 minutes (c). Sub-figure (d) shows the stable-state density gradient

## 6.3 Run-time snow layering

The last test aims to mimic multiple snowfalls under different conditions, defined by the air temperature at the time of new snow formation. Instead of initializing the entire




(a)




(b)

Figure 6.3: Simulation results of test 1.a (a) and test 1.b (b), taken from cell data of the central column. The curves show the snow density as a function of depth (left) and snow hardness as a function of density (right).

snowpack height before the simulation start, this time the snow height is added consecutively, with enough time in between new additions for the existing snow to settle down and stabilize. The aim of this test is to force the visible snowpack layer formation due to uneven compression, as described in section 3.2.3.

The simulation parameters listed in table 6.3 were hand-tuned in order to produce the largest visible difference in snow hardness and density. The stabilized density profile of the snowpack is depicted in figure 6.6.a, along with its temperature profile in figure

Table 6.2: Test 2 parameters

| *Input parameters* | | |
|---|---|---|
| **Fresh snow density** | 20 $kg/m^3$ | |
| **Temperature** | range {-3 ; -15} | |
| *Output parameters* | | |
| | Before sim. | After sim. |
| **Column mass** | 4.48 kg | 4.92 kg |
| **Column height** | 5.6 m | 5 m |
| **Column avg. density** | 20 $kg/m^3$ | 24.6 $kg/m^3$ |

6.6.b. The plots in figure 6.6 show which cells are responsible for the layer with the lowest density.

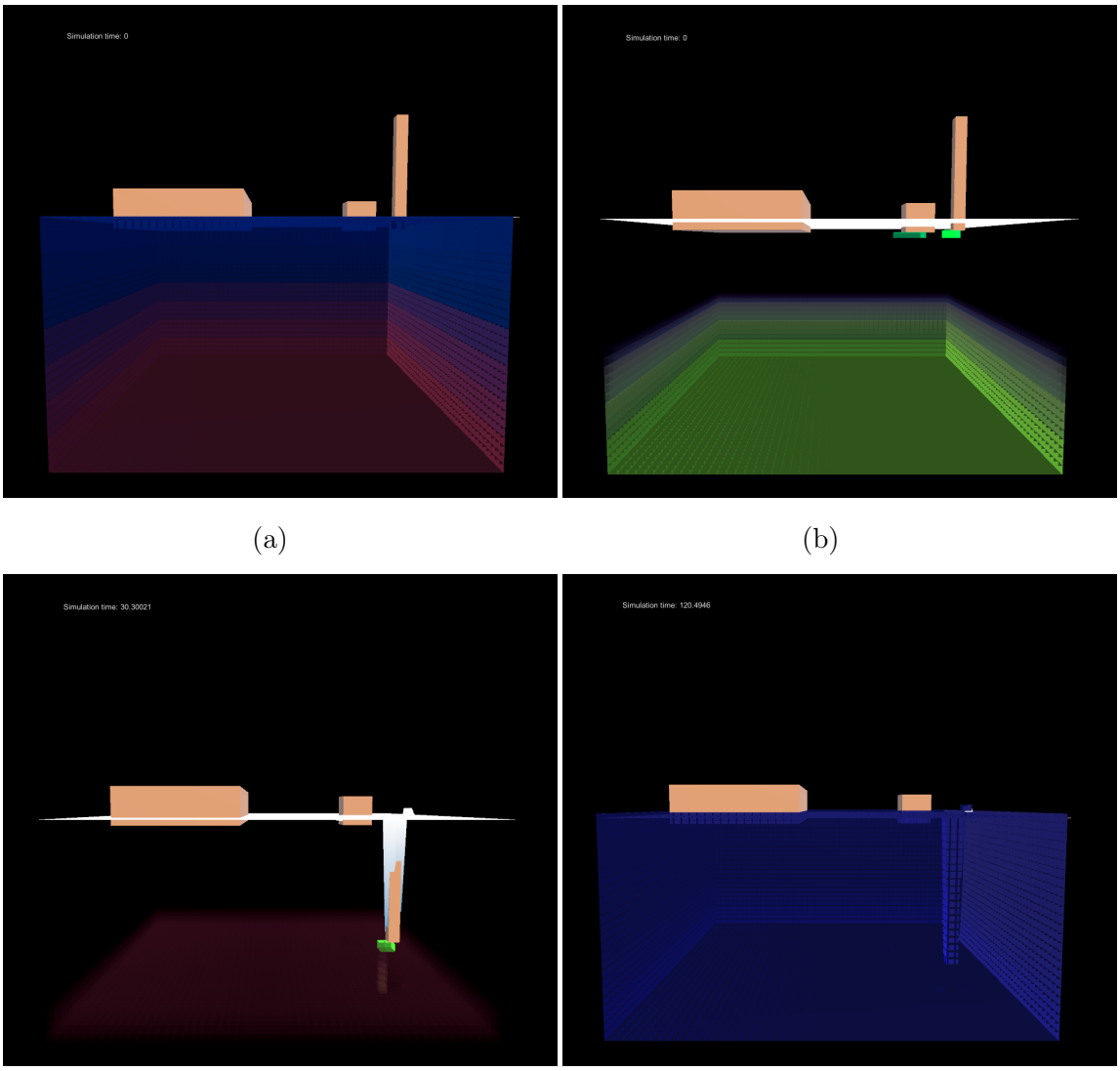
Table 6.3: Test 3 parameters

| | **Snowfall 1** | **Snowfall 2** | **Snowfall 3** | **Snowfall 4** |
|---|---|---|---|---|
| **Fresh snow density** | 20 $kg/m^3$ | 20 $kg/m^3$ | 20 $kg/m^3$ | 20 $kg/m^3$ |
| **Added height** | 3.6 m | 1.2 m | 3.6 m | 3.2 m |
| **Temperature** | - 3 $°C$ | -15 $°C$ | - 3 $°C$ | - 3 $°C$ |

This concludes the simulation experiments. The fulfilled and failed requirements are reiterated in table 6.4.

Table 6.4: Project evaluation map

| **Req №** | **Constraint(s)** | **Use case(s)** | **Experiment(s)** | **Successful** |
|---|---|---|---|---|
| 1 | 1,2,3,4 | 1,2,3 | 1.a, 1.b, 2, 3 | yes |
| 2 | 2 | 1 | 1.a, 1.b | yes |
| 3 | 3 | 1,2 | 1.a, 1.b , 2 | yes |
| 4 | 5,7 | 1,2,3 | 1.a, 1.b, 2, 3 | yes |
| 5 | 6 | 1,2,3 | 1.a, 1.b, 2, 3 | yes |
| 6 | 8 | 1 | 1.a, 1.b | yes |
| 7 | 9 | 1,2,3 | 1.a, 1.b, 2, 3 | no |




(a) (b)

(c) (d)

Figure 6.4: Visualization of the simulation run for test 2, taken from the implemented demo project.. Screenshots show the start temperature gradient alone (a), the pressure profile mixed with temperature gradient at simulation time = 0 s (b) and 30 s (c). Sub-figure (d) shows the stable-state density gradient

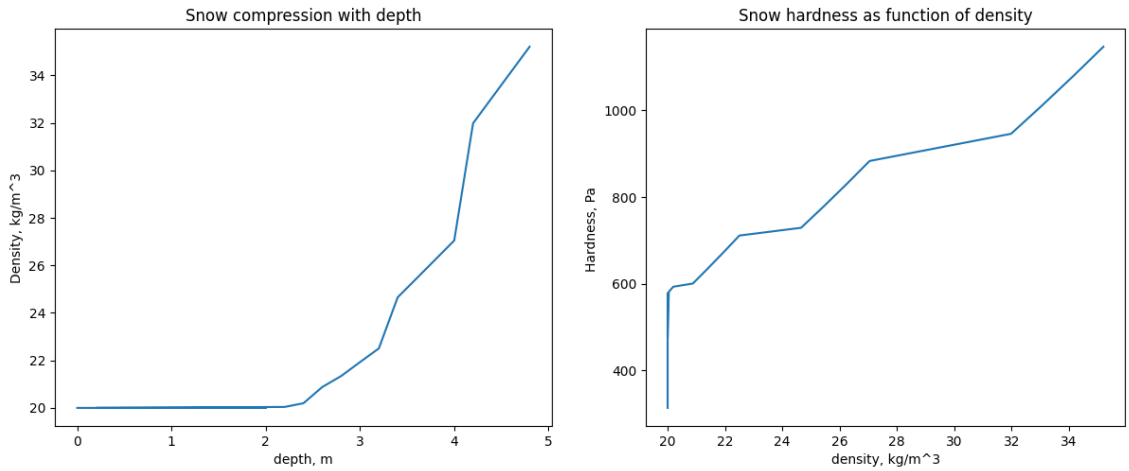
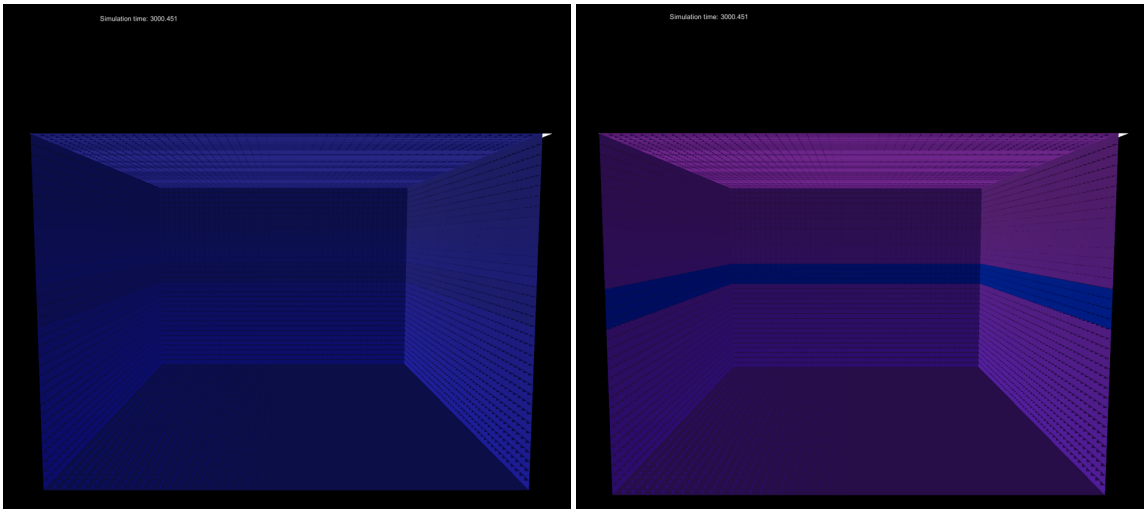



Figure 6.5: Simulation results of test 2, taken from cell data of the central column. The curves show the snow density as a function of depth (left) and snow hardness as a function of density (right).



(a) (b)

Figure 6.6: Visualization of the simulation run for test 3, taken from the implemented demo project.. Screenshots show the stable state density profile (a), and the temperature (b) of the snowpack.

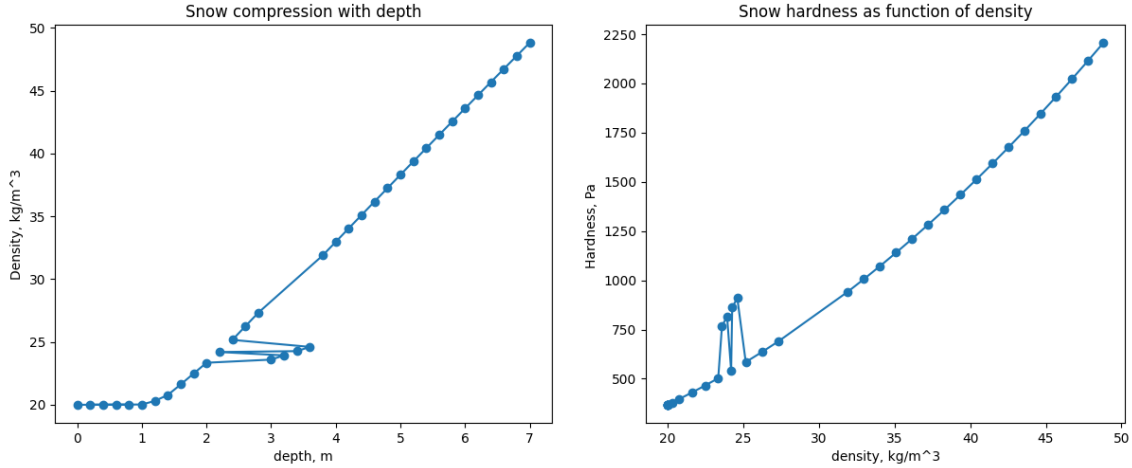



Figure 6.7: Simulation results of test 3, taken from cell data of the central column. The discretized graphs show the snow density as a function of depth (left) and snow hardness as a function of density (right), the individual data points are marked with circles.

# 7 Summary and Outlook

As mentioned in the previous chapter, the most obvious limitation of a fixed grid-based implementation in 3D space is that the correct mass transfer proves to be a challenging task, an issue that could be easily solved by moving particles. In all the other aspects, grid calculations showed quite stable results in replicating the relation between the snow density, temperature and hardness, and, subsequently the deformation amount. This dependency could be further extended by taking into account such parameters as grain size, humidity, porosity etc. The vertical compression and mass movement could also be extended to cover all directions. It would be interesting to see the results produced by coupling this grid-based implementation with a particle system, that could potentially fix the limitations stated earlier. Moreover, the same particle system could be used to simulate snow fracture and stickiness (for example, by modeling bonds between particles as described in [7]).

Additionally, the problem of where and how much the snow should accumulate during a snow fall could be explored further. Figure 7.1 shows the different configurations of accumulated snow based on the underlying terrain and the wind direction, that could serve as a good basis for templating procedural snow cover generation. Simulating the snow fall itself would become a trivial task once particles are utilized. So, a more sophisticated simulation coupled with Lagrangian particles could make up for a versatile self-contained snowy environment that "lives its own life", waiting for the player to interact with it.

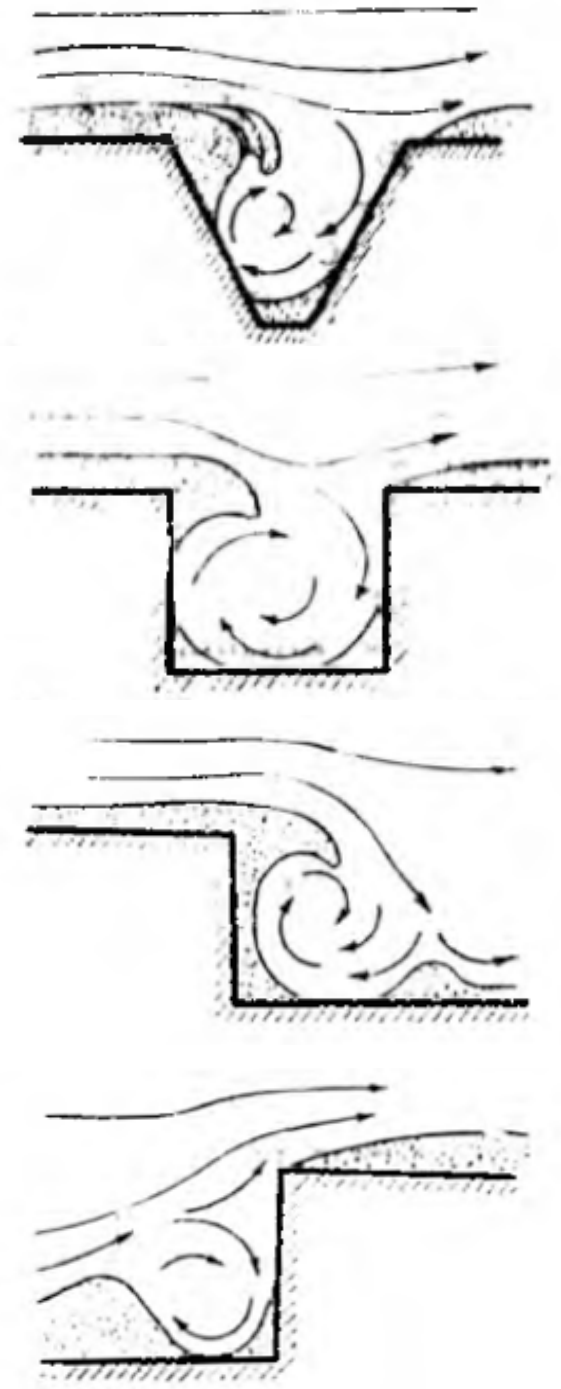


Figure 7.1: Snow accumulation on different shapes of underlying terrain. The wind flow is indicated with arrows. The original figure is taken from [9].

# Bibliography


[1] A., Anikin A. ; V., Barakhtanov L. ; O, Donato I.: Fiziko-mekhanicheskie svoistva snega kak polotna puti dl'a dvizhenia mashin [Physical and mechanical properties of snow as a roadbed for the movement of vehicles]. In: *Mashinostroenie i komputernie tekhnologii [Mechanical engineering and computer technology]* (2010), 10, Nr. 10, S. 5. – URL http://technomag.edu.ru/doc/160649.html

[2] Barakhtanov, Lev ; Belyakov, Vladimir ; Blokhin, Aleksandr ; Denisenko, Elena: Mathematical dependence of physical-mechanical properties of snow cover as support base for movement of vehicles. In: *Science and Education of the Bauman MSTU* 12 (2012), 08

[3] Barre-Brisebois, Colin: *Deformable Snow Rendering in Batman: Arkham Origins*. 2014. – URL https://www.gdcvault.com/play/1020177/Deformable-Snow-Rendering-in-Batman. – Game Developers Conference

[4] DeWalle, David R. ; Rango, Albert: *Principles of Snow Hydrology*. Cambridge University Press, 2008

[5] Gissler, Christoph ; Henne, Andreas ; Band, Stefan ; Peer, Andreas ; Teschner, Matthias: An implicit compressible SPH solver for snow simulation. In: *ACM Transactions on Graphics* 39 (2020), 07

[6] Gold, Lorne W.: The Strength of Snow in Compression. In: *Journal of Glaciology* 2 (1956), Nr. 20, S. 719–725

[7] Goswami, Prashant ; Markowicz, Christian ; Hassan, Ali: Real-time particle-based snow simulation on the GPU. In: *Eurographics Symposium on Parallel Graphics and Visualization* :, 2019. – open access

[8] Purho, Petri: *Exploring the Tech and Design of 'Noita'*. 2020. – URL https://www.gdcvault.com/play/1025695/Exploring-the-Tech-and-Design. – Game Developers Conference

[9] Rikhter, Gavriil D.: *Snezhnyi pokrov, ego formirovanie i svoistva [Snow cover, its formation and properties]*. AN SSSR Publ., 1945

[10] Sofer, M: Sneg [Snow]. In: *Nauka i Zhizn' [Science and Life]* (1982), Nr. 1, S. 33–39

[11] Stomakhin, Alexey ; Schroeder, Craig ; Chai, Lawrence ; Teran, Joseph ; Selle, Andrew: A material point method for snow simulation. In: *ACM Transactions on Graphics (TOG)* 32 (2013), 07

[12] Stomakhin, Alexey ; Schroeder, Craig ; Jiang, Chenfanfu ; Chai, Lawrence ; Teran, Joseph ; Selle, Andrew: Augmented MPM for phase-change and varied materials. In: *ACM Transactions on Graphics* 33 (2014), 07, S. 1–11

[13] Svensson, Josen: REAL-TIME RENDERING OF DEFORMABLE SNOW COVERS, URL https://api.semanticscholar.org/CorpusID:208195618, 2019

[14] Tarasov, Lev V.: *Fizika v Prirode [Physics in Nature]*. "Prosveshcheniye" Publ., 1988

# A Appendix

The appendix to this thesis is in electronic form and can be obtained from the supervisors. It contains the project build application, the Unity editor project with the source code and a digital copy of this thesis. Those can be accessed from the root folder in the following way:

The project application can be launched via Build/Snow Simulation.exe.
The source code can be accessed from the Snow-Simulation/Assets/Scripts folder.
The digital copy is accessible under the file name thesis.pdf.

## Declaration

I declare that this Bachelor Thesis has been completed by myself independently without outside help and only the defined sources and study aids were used.

| City | Date | Signature |
| --- | --- | --- |