DEVELOPMENT AND IMPLEMENTATION OF SNOWMELT SIMULATIONS IN AN AUGMENTED REALITY SANDBOX

by

Tyler Ainsworth

A thesis submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Master of Science in
Applied Energy and Electromechanical Systems

Charlotte

2021

Approved by:

_____
Dr. Elizabeth Smith

_____
Dr. Wesley Williams

_____
Dr. Jacelyn Rice-Boayue

ABSTRACT

TYLER AINSWORTH. Development and implementation of snowmelt simulations in an augmented reality sandbox. (Under the direction of DR. ELIZABETH SMITH)

In cold climates snowmelt runoff prediction is an important factor in the prediction of seasonal flooding and springtime water availability. Engineers study snowmelt and its effects on the environment, such as changes in stream flow, for this purpose. Typically, snowmelt is taught at the college level. This research creates a snowpack and snowmelt simulation that can be used to introduce the concept to younger audience.

The Augmented Reality (AR) sandbox was created by University of California (UC) Davis' W.M. Keck Center for Active Visualization in the Earth Sciences (KeckCAVES). An AR sandbox includes an Xbox Kinect depth sensor, a short throw projector, a computer with a graphics card, and a sand surface. The sandbox allows users to move the sand that is then augmented in real time with a colored topographic map and simulated water flow. AR sandboxes are a hands-on tool that can be used to teach students concepts such as runoff, watersheds, and the reading topographic maps. UC Davis has made the code and designs for their AR sandbox open source and available online. Similar sandboxes can now be found around the world as both a museum exhibit and as learning or research tools in schools.

A mathematical model snowpack and snowmelt simulation was developed for the AR sandbox. The model implements a height-based freeze mechanism to generate a snowpack based on a user created topography and a simulate time based melting behavior. This set of equations used a percentage based system to remove an amount from the existing snowpack and add that to the liquid water quantity. The level of detail and complexity captured by this snowpack and snowmelt model is consistent with the realtime nature of the AR sandbox and the overall level of detail in the existing calculations. The new model was then implemented into the AR sandbox

code. The software design supports easy implementation into other code bases in the open source community.

To support the use of the snowmelt simulation as an educational tool, a set of educational materials were created. These materials were made with the intent of being used alongside the AR sandbox with the snowmelt simulation to create a semi-guided hands-on display.

This research developed a numerically accurate snowmelt simulation with intuitive visual effects for use as a research and education tool. The code developed for this simulation can be extended further for other weather-based simulations. Possible future additions could include frozen lakes, avalanches, and weather fronts.

ACKNOWLEDGEMENTS

I would like to thank everyone who supported me through my Master's degree. First, without Dr. Elizabeth Smith mentorship and guidance I would not have been able to succeed as I have with my degree. Further, I thank Dr. Wesley Williams whose support as a member of my thesis committee and who encouraged me as an undergraduate student to pursue a graduate degree. I would also like to thank Dr. Jacelyn Rice-Boayue for being a part of my committee and lending her knowledge and expertise to the project.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

1D One Dimensional

2D Two Dimensional

3D Three Dimensional

API Application Programming Interface

AR Augmented Reality

ARES Augmented REality Sandtable

GIS Geographic Information System

GPU Graphics Processing Unit

ISO International Organization for Standards

KeckCAVES W.M. Keck Center for Active Visualization in the Earth Sciences

NEH National Engineering Handbook

NSF National Science Foundation

OpenGL Open Graphics Library

RGBA Red, Green, Blue, Alpha

RK Runge-Kutta

SWE Shallow-Water Equations

UC Davis University of California, Davis

UNCC University of North Carolina at Charlotte

USDA United States Department of Agriculture

USGS United States Geological Survey

Vec3 Vector3 Data Type

Vec4 Vector4 Data Type

CHAPTER 1: INTRODUCTION

1.1 PROBLEM STATEMENT

In some parts of the United States, snowmelt runoff accounts for up to 75% of the water supplies [1]. Snowmelt runoff is runoff that occurs when a buildup of snow, a snowpack, melts. Engineers work to simulate snowmelt runoff in order to be able to predict its effect on the environment. Some of these effects include impacts on the environment and ecosystem through flooding and erosion.

Typically, snowmelt is taught within the hydrology curriculum in higher education. However, the basic principles are taught earlier. In kindergarten through high school the curriculum generally focuses on why ice melts and the water cycle through to the effects of runoff and erosion as students progress.

In recent years schools have been acquiring new tools and equipment for use in the classroom. One of these new tools is the Augmented Reality (AR) Sandbox. Originally designed in 2014 by University of California, Davis (UC Davis) under National Science Foundation (NSF) funding, the goal of the project was to "raise public awareness and increase understanding and stewardship of freshwater lake ecosystems and earth science processes..."[2]. In the design by UC Davis users can manipulate sand within the sandbox to create a desired landscape that is sensed by an XBox Kinect sensor and processed to create a colored topographic map of the sandscape. This map is then projected back onto the sand and updated as users further modify the sand topography. Users are also able to trigger simulated rainfall in the created landscape.

While AR sandboxes are becoming more widely used for education, in museums as well as schools, the base use case of simulating runoff remains largely unchanged. The creation of a simulated snowpack and simulated snowmelt runoff would expand the use of an AR sandbox as a hands on method of teaching these concepts. This

research details the process through which a snow simulation was developed including the mathematical expressions and their implementation into the existing AR sandbox code.

1.2 LITERATURE REVIEW

1.2.1 Augmented Reality Sandbox

The AR sandbox utilizes a playsand surface, an Xbox 360 Kinect camera, a short throw projector, and a computer to sense a user created topography and project a colored height map onto the sand surface. UC Davis has made the code for the project opensource and available on their website and provided general guidelines for users to build the physical system [2]. In UC Davis's design the AR sandbox code uses the C++ coding language along with OpenGL, a graphics rendering Application Programming Interface (API), to create the topographic map. Section 3 goes into further detail on the method in which C++ and OpenGL are used.

The AR sandbox constructed at the University of North Carolina at Charlotte consists of the basic required components along with an adjustable aluminum frame and a thermoplastic tub instead of a more common cantilever electronics support and solid wood construction. The overall Figure 1.1 shows a 3D render of the sandbox with components labeled. Table 1.1 describes the numbered components.

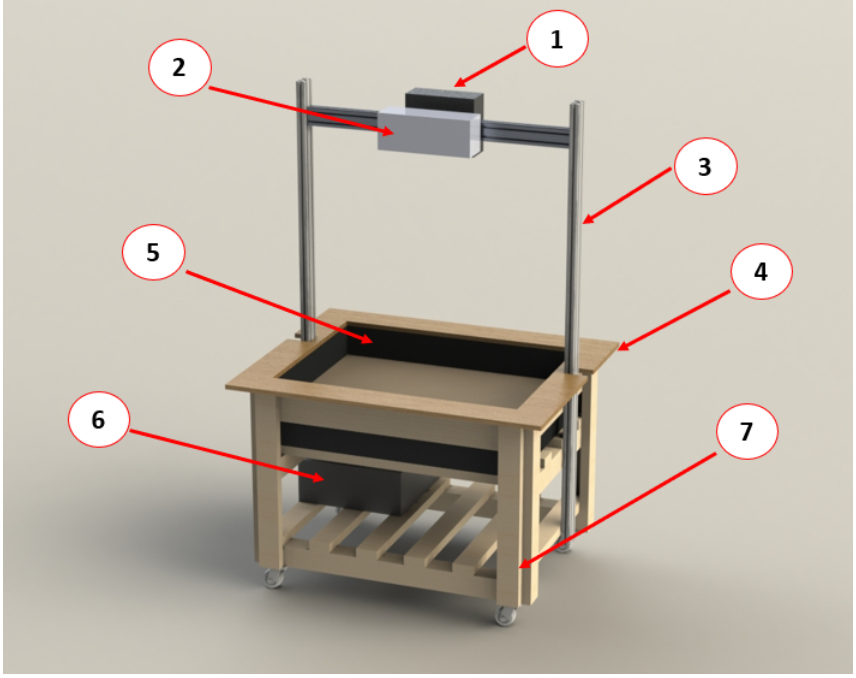


Figure 1.1: 3D model of the AR sandbox at UNCC with component labels.

Table 1.1: A list of the labeled components in UNCC's AR sandbox.

Label Number	Component Name
1	Short throw projector
2	Xbox Kinect depth camera
3	Extruded aluminum frame
4	Removable wooden ledge for display
5	Thermoplastic tub for playsand
6	Computer system
7	Wooden main sandbox body

Figure 1.2 displays and image of the sand while the AR sandbox is running.

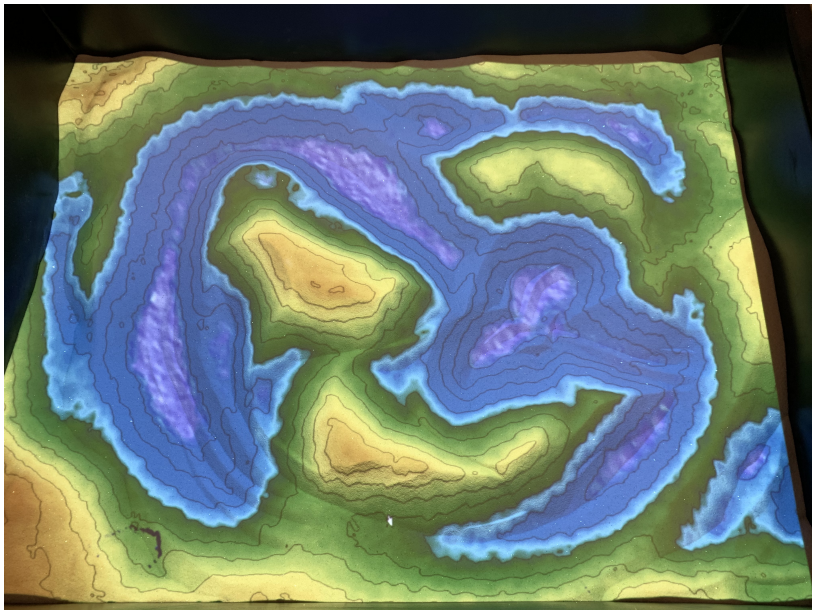


Figure 1.2: An image of the topographic lines and colors projected onto the sand by the AR sandbox.

As can be seen in Figure 1.2, the colored height map of the AR sandbox starts at red-orange at the highest point and blends into green and then shades of blue at lower elevations. The abrupt shift from green to blue is where the water table begins. Functionally this is a height of zero within the table and the topography is either above (positive) or below (negative) relative to this value.

Another main feature of the sandbox is the ability to add simulated water that uses the gathered topography data and runs shallow-water flow simulations. The sandbox makes use of the Saint-Venant equations in order to run this simulation.

1.2.1.1 Saint-Venant Equations for Shallow Water Flow

The Saint-Venant equations are a set of unidirectional equations derived from the Navier-Stokes equations. Also called the Shallow-Water Equations (SWE), these derived equations are used in the modeling of shallow-water flow. The main requirement of this being that the vertical scale is much smaller than the horizontal scale of the problem. Developed over a century ago [3], this system of equations continues to be in use today in a variety of applications. Equation 1.1 shows the set of equations for

one dimensional (1D) cases. Table 1.2 lists the included variables and their definition. This system does not include the effects of friction between the water or fluid and the topography.

$$\begin{cases} h_t + (hu)_x = 0, \\ (hu)_t + (hu^2 + \frac{1}{2}gh^2)_x = -ghB' \end{cases} \tag{1.1}$$

Table 1.2: List of variables and definitions used in Eqn: 1.1.

Variable	Definition
B	The bottom height or Bathymetry
h	The height of the water column above B
hu	The discharge in the x direction
g	The gravitational constant.

In the approximately 150 years since their development, there have been further derivations of these equations for various specific use cases. One such investigation looks at the inclusion of a coefficient of friction and viscosity into a Saint-Venant system [4]. In the standard derivation of the Saint-Venant equations there lacks method in which to determine the main factor, either velocity or momentum, in jumps in some one dimensional (1D) simulations. A dam break simulation using the equations described were compared to a simulation performed to validate the equations[4].

Another implementation, called the "positivity preserving central-upwind scheme" [5] decreases numerical errors that can occur in 'dry' conditions. Dry conditions are described as when the fluid height, above the bottom topography, is equal to zero or is calculated as a negative number. This specific derivation is highly useful for applications were a variable bathymetry height can result in sections of the bottom topography being above the fluid height during the simulation. In an AR sandbox a large amount of the simulation area is considered dry. In Figure 1.2, the topographic lines can be seen in areas where the water height is equal to zero. As such the AR

sandbox makes use of this scheme.

Current Sandbox Algorithm

UC Davis' AR sandbox utilizes the Saint-Venant equations for shallow-water flow. The sandbox uses the the positivity-preserving scheme detailed in [5] in a two dimensional (2D) form. The equations used in the table are further detailed in Section 2.1.

When run, the AR sandbox only solves the SWE equations if simulated water is present in the table. This means that before water is added or after the water is 'drained' or 'evaporated' from the table the calculations are skipped to improve improve performance.

1.2.1.2 Uses in Research and Industry

AR sandboxes can also be used in various research projects. One novel project modified the UC Davis software to create an "empirically realistic ant colony foraging simulation"[6]. Their software was created in Unity3D and C#[6] as opposed to OpenGL and C++ as used in the original. A simulation was created in which ants search for food and bring it back to the nest leaving pheromone trails. These trails can be disrupted by the user and a set water level can be used to force the ants to find different paths.

The technology and framework behind the AR sandbox hasn't been limited to use in schools and museums. A technical report by the US Army Research Lab utilizes an Augmented REality Sandtable (ARES) and custom software to test the benefits of an AR system on the users ability to "construct a sand table terrain model that accurately depicts the terrain and tactical plan as outlined..." [7]. Participants used either ARES or a traditional sandtable with no AR capabilities and were scored on six accuracy based criteria. The study found that the use of ARES resulted in higher scores in the accuracy of element placement and the perceived workload also resulted

in ARES' favor.

A version of the AR Sandbox technology called the Simtable has been developed to simulate natural disasters such as wildfires and floods [8]. The Simtable is designed to use Geographic Information Systems (GIS) with the sandtable and allow for natural disaster response planning. Users can determine an area and, in the case of wildfire simulation, select an ignition point and

Another use of the AR sandbox technology is its use in landscape design and geodesign applications. One study investigates the possible benefits of using an AR sandbox for outdoor trail design [9]. It was found that the sandbox provided both a learning and design environment use for visualization and communication between users.

1.2.1.3 Uses in Education

The AR sandbox framework has found multiple uses as a hands-on educational and research tool since its creation. AR sandboxes can be found across the world in multiple schools and museums with the UC Davis website listing locations such as Ithaca High School, Somersworth High School, Fairmount Water Works Interpretive Center and Catavento Cultural e Educacional science center [2].

Multiple studies have been conducted to determine the benefits of utilizing this technology in the classroom. One such study looks into the effect that using a version of the sandbox has on the spatial thinking and visualization of three dimensional landscapes. This study was done at in the Department of Geological Sciences at East Carolina University with an undergraduate class geology class. Based on the self-reported survey taken by the students, the sandbox was highly effective in enhancing student learning of the fluvial and coastal features as part of a lesson [10].

1.2.2 Snowmelt Runoff

Snowmelt runoff is the runoff that occurs due to the melting of a snowpack or accumulation of snow. This snow accumulation persists through the colder seasons before melting in the spring. The rate at which the melt occurs can be dependent on a variety of factors, including temperature and the amount of rainfall.

1.2.2.1 Importance of Modeling Snow and Snowmelt

In the northern parts of the US and colder climates world wide, the modeling and prediction of snowmelt is a key factor in topics such as flood prevention. During weather events where warm rain increases the melt rate of snow cover can create sudden surges of runoff. A case study completed in 1998 models a flood event in the Pacific Northwest in which the sensitivity of snowmelt to climate and land cover is shown[11].

Modeling the current snowmelt is also a method in which scientists can look at even larger scale phenomenon such as climate change. Due to snowmelt being dependant on the temperature of a given region the period during the year in which the melting occurs can give further insight into the the climate of the area. An investigation into the timing of snowmelt and the resulting effects seen in streamflow in Colorado shows how the snow cover in these high altitudes has been melting earlier in the year [12]. Similarly, changing snowmelt patterns has the potential to have effect on food productions in areas where snowmelt plays a large part in the irrigation system [13].

By creating a snowmelt simulation that is accessible for younger or less specialized audiences it allows for an introduction into fairly complex topics. Being able to interact in a hands-on manner can help garner interest in these topics that they might not otherwise.

1.2.2.2 Equations

There are multiple methods in which to calculate the amount of snowmelt generated. The US Department of Agriculture (USDA) lists both a degree-day method, utilizing a temperature index, and a regression analysis method in the National Engineering Handbook (NEH). The degree-day method for estimation utilizes a coefficient C_m that varies depending on the amount of degree-days, days in which it has been lower than the base temperature during the period, and the location [14]. Equation 1.2 shows this method where:

M = snowmelt (in/day)

T_a = mean daily air temperature(°F)

T_b = base temperature (65 °F)

$$M = C_m(T_a - T_b) \tag{1.2}$$

The regression analysis method detailed in the NEH is more complex. The equation is created using regression analysis to calculate the coefficients of the most relevant parameters of a energy balance approach. This results in equations that can depend on factors such as rainy or non-rainy periods, tree cover, and temperature. An example of an equation created to calculate snowmelt during rainy periods in a partially forested area [14] is shown in Equation 1.3,

$$M = C[0.09 + (0.029 + 0.00504v + 0.007P)(T_a - T_F)] \tag{1.3}$$

where:

M = snowmelt (in/day)

v = wind speed 50 feet above the snow surface (mph)

T_a = air temperature (°F)

T_F = freezing temperature (°F)

P = rainfall (in/day)

C = Coefficient to account for variations

Both Equations 1.2 and 1.3 are two of many possible equations that can be used to calculate the amount of snowmelt in an area. Many equations can be created based on the possible applications and goals of a study.

CHAPTER 2: METHODOLOGY

The first step of this project is the development of the necessary equations. These equations were initially created outside of the sandbox framework and were later tailored to the base code of the sandbox. This was done to prove the equations and negate any issues that could arise from coding errors as opposed to math related errors. The equations created were placed into two categories based on their subtractive (melting the snow) or additive (creating snow from rain) uses. These equations were developed while keeping in mind the current equations being used in the AR sandbox for the shallow water simulation.

2.1 AR Sandbox Saint-Venant Equations

The existing sandbox code uses a two dimensional (2D) implementation of the SWE shown in Eqn 1.1. No new variables are introduced but the discharge now has both u and v components, and the derivative of the bottom height must have a specified direction. Equation 2.1 shows the 2D shallow water governing equations.

$$\begin{cases} h_t + (hu)_x + (hv)_y = 0, \\ (hu)_t + (hu^2 + \frac{1}{2}gh^2)_x + (huv)_y = -ghB_x, \\ (hu)_t + (huv)_x + (hv^2 + \frac{1}{2}gh^2)_y = -ghB_y \end{cases} \tag{2.1}$$

Dry conditions are handled by setting negative water heights to zero. The impact of dry conditions on velocity are handled through desingularization using the approach in Kurganov and Petrov [5]. This approach updates the velocity components using a correction factor based on the water height (h). As h approaches zero, the denominator approaches zero as well. The epsilon term prevents the divide by zero that would occur.

$$\begin{cases} u = \dfrac{1.41421356237309h(hu)}{\sqrt{h^4 + max(h^4, \epsilon)}}, \\ v = \dfrac{1.41421356237309h(hv)}{\sqrt{h^4 + max(h^4, \epsilon)}} \end{cases} \tag{2.2}$$

The 15 digit constant is the mathematical value of the square root two. The programmers used the constant as an efficiency measure since the square root operation is one of the slower mathematical operations to perform computationally.

Epsilon (ϵ), Equation 2.3 is a function of the cell size. Since the table uses uniform cell sizes in each direction the dimension of any cell can be used for this calculation.

$$\epsilon = 0.1max(dx, dy, 1.0) \tag{2.3}$$

The stability of the algorithm is maintained by limiting local speeds and using an adaptable timestep size. The simulation uses the second-order strong stability preserving Runge-Kutta (RK) temporal integration scheme with a second order positivity preserving central-upwind scheme for spatial derivatives.

2.2 One Dimensional SWE Solver

Solving the Saint-Venant equations in one dimension allows for the simulation of these equations in a less computationally intensive manner. This cuts down on the time needed to develop and iterate the simulation while still generating a clear and understandable simulation with necessary initial and boundary conditions. For this SWE equation simulation two boundary conditions are considered: reflective or zero gradient. A reflective or solid boundary condition occurs when the fluid that would flow out of the simulation boundary is conserved. This results in the fluid 'bouncing' off the edge of the simulation as if it was reflecting off of a wall. A zero gradient boundary condition means that water that flows outside the boundary of the simulation is not conserved or removed. The fluid in the simulation would appear to 'flow off the table'. The will generally result in the total amount of fluid in a

simulation to decrease over time depending on the bottom topography.

Another factor in SWE simulation solvers is what is known as 'dry' conditions. As the name suggests, occurs when there is no fluid in an area of the simulation when h is equal to, or near to, zero. Depending on the derivation of the SWE equations this can cause errors while attempting to run the simulation. There have been multiple methods to negate these errors developed including those mentioned in [5].

2.2.1 Dam Break Simulation

A common means of using the 1D SWE equations is in a one dimensional dam break simulation. These simulations are representations of a large amount of water being released through a wall and the resulting water flow. For this research an existing dam break simulation was found: "1D SWE dam break simulation" [15]. Made available by James Adams through MATLAB's Central File Exchange in 2014, this program allows for the solving of a dam break problem with four solving methods: Lax-Friedrichs, Lax-Wendroff, MacCormack, or Adams Average scheme.

Along with the option of four methods to solve the SWE equations the dam break simulation program also includes the options to solve with zero gradient or reflective boundary conditions. The default values for the simulation include an overall water depth of 0.1 meters, a water depth behind the dam (h) of 0.2 meters. Figure 2.1 shows the initial conditions when using the Lax-Wendroff Scheme.

This existing program was modified to better match the conditions found in the AR sandbox. The first modification was to create a function to allow for a variable bottom topography B. Another modification was made to allow the equations to be solved with the 'dry' condition. This was done by adjusting the SWE equations and the dependant functions to implement the positivity-preserving scheme described in [5].

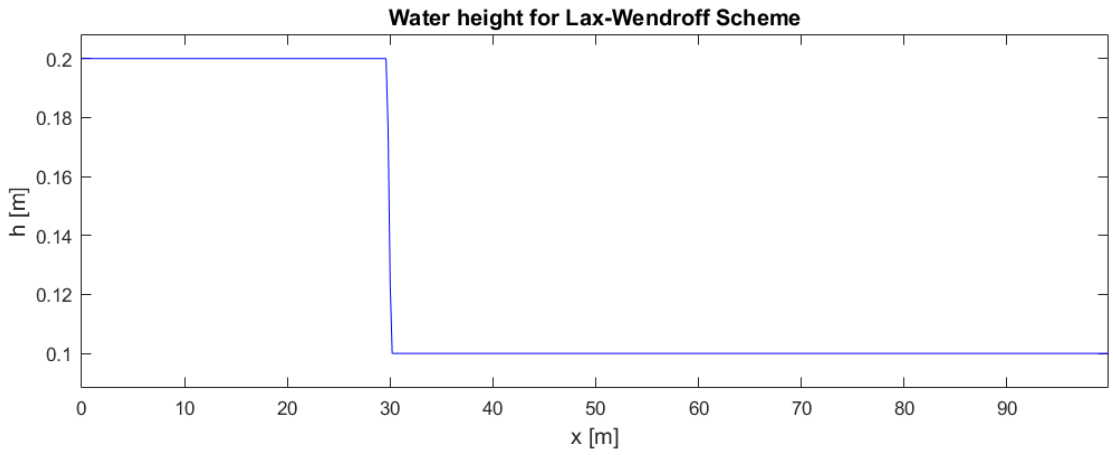


Figure 2.1: Default initial values of the one dimensional SWE program.

2.3 Snow Simulation

Based on other snowmelt runoff equations and with consideration to the limitations of the sandbox, it was decided to create a simplified snowmelt equation for ease of initial implementation and lower possible impacts on run-time. Despite being simplified, there are some factors that occur in nature that were necessary to properly simulate and visualize snowmelt and a snowpack. These factors that were included are the presence of a snow line and the snow melting over time.

A snow line is the point at which permanent snow cover begins on a mountain. This concept is visually distinct and evokes the desired look of snow capped mountains even at the small scale of the AR sandbox. For most users it is fairly intuitive for snow to develop on the top of a mountains. By including this it allows users to generate snow in a predictable area without needing to include other more complicated factors such as ambient temperature with corresponding visual indicators. As such, using this concept to determine where snow in the sandbox will generate is a highly efficient method.

The manner in which the snow line is represented in the new equations is through the use of a variable called 'critical height' or `critHeight`. This value represents the

height above which any precipitation added to the simulation is generated as snow rather than water. The inclusion of this variable would allow for the areas snow is able to generate to be adjustable while still being a consistant height across the sandbox. A pseudocode example implementation of the critical height value is shown below.

```
IF bathymetry Is Greater Than critical height
    Precipitation is snow
ELSE IF bathymetry Is Less Than critical height
    Precipitation is water
END IF
```

The other main effect included in the new equations and simulation is a melting over time behavior. Ensuring that the snow in the table melted over time would result in a more realistic simulation. If the melting step was not done as a function of time, a timestep based loop, the total amount of snow would melt as one. This would be the equivalent of a large block of water being released, similar to a dam break but not similar to a snowpack melting. Additionally, the choice between a set amount of snow being removed versus a percentage was considered. Both options provide a method for melt over time but due to the variability of the amount of snow in a given area it was decided to use a percentage based system. This would create a smooth transition from snow to no snow at small amounts.

There are two main pieces taken into account to create the melting behavior: removing volume from the snow variable and adding equal volume to the water variable. Two recursive equations were developed to remove or add a percentage of existing snow. Equations 2.4 and 2.5 show the subtractive and additive equations respectively where s is the snow height and h is the height of the water.

$$s_t = s_{t-1} - (0.15 s_{t-1}) \tag{2.4}$$

$$h_t = h_{t-1} + (0.15 s_{t-1}) \tag{2.5}$$

A 15% change was chosen based on the current snow depth and the length of time

to reach approximately zero remaining snow. Figure 2.2 shows a snow height over time graph of three melt rates.

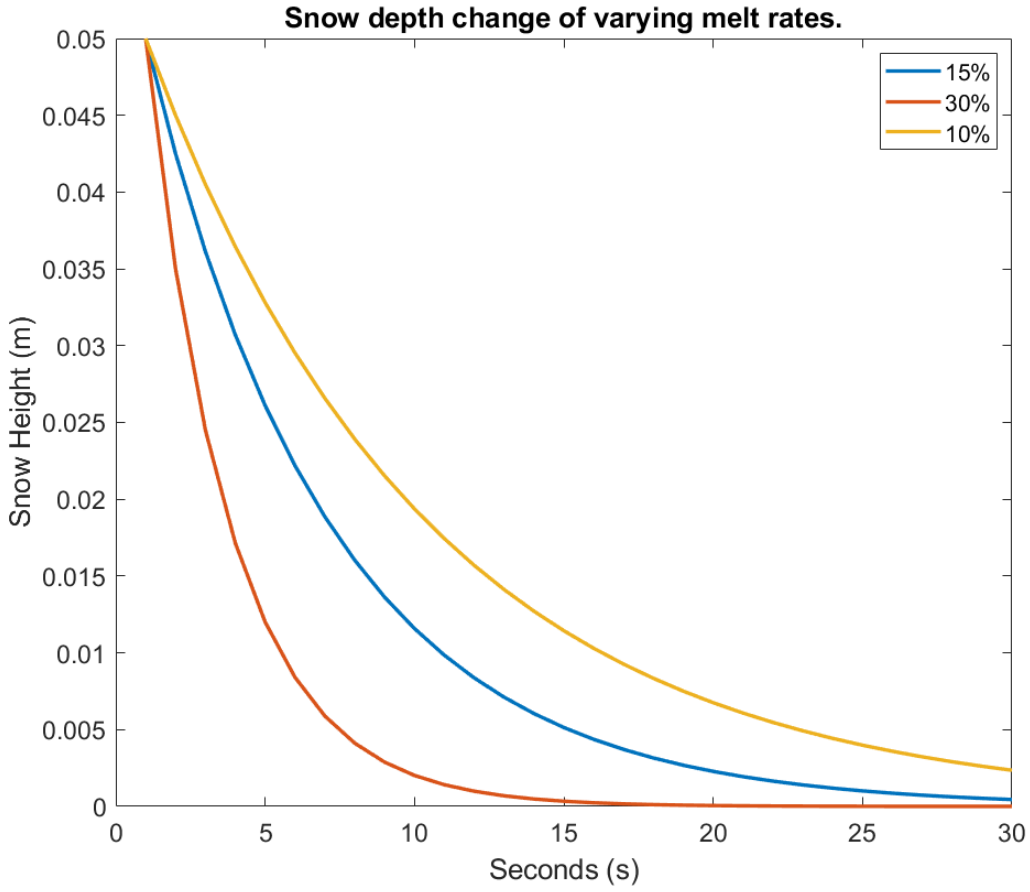


Figure 2.2: Snow height over time at 10%, 15%, and 30% of snow removed per second.

Due to the snow amount being its own variable separate from the water amount, the water added to that area persists. It was decided to remove the water in areas where snow is created in order to keep the total amount of precipitation in the AR sandbox consistent. This was done by once again using the critical height variable. When the sandbox checks the bathymetry height to determine if snow should be created, that same check can be used to remove the water in that area. In the AR sandbox code the water amount, velocity, and acceleration are contained in a single variable. When removing the water all of these values are set to zero. This stops

the code from running the SWE equations when not needed and cutting down on the processing required. An example of this can be seen below

```
IF bathymetry Is Greater Than critical height
    Set water height to zero
    Set water veolocit and acceleration to zero
END IF
```

2.3.1 Initial conditions for Snowmelt and Snowpack

In order to test the developed equations the 1D SWE Dam Break solver using the Lax-Wendroff scheme was used. The following base initial conditions were used: an initial depth (h) of 0.05 meters, a width (x) of 50 meters, and a sine wave as the bathymetry height (B). The equation for B is shown in Equation 2.6. Figure 2.3 shows the initial conditions of the SWE model before the snowmelt and snowpack equations were implemented.

$$B = (0.2sin(0.1(2x))) + 0.25 \tag{2.6}$$

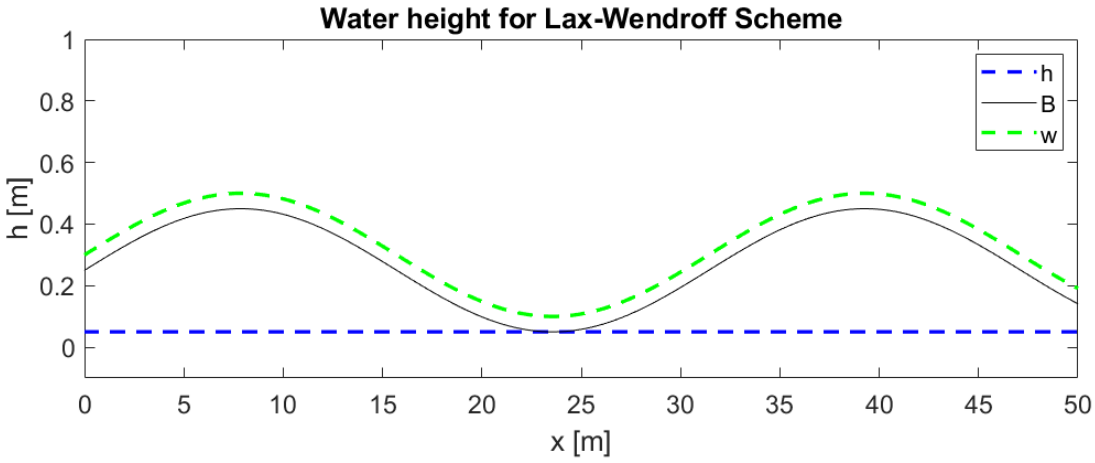


Figure 2.3: MATLAB plot showing the initial conditions with which the snowmelt simulation is designed.

2.3.2 Snowmelt

The first piece of the snowmelt simulation to be tested was the melt behavior. The initial conditions were modified to include a preexisting snowpack above a set height.

The function that generates the initial water height was modified to instead generate an amount of snow. Equation 2.7 shows the set of equations used and Figure 2.4 shows the new initial condition plot.

$$[h, s] = \begin{cases} s = 0.05, h = 0, & B \geq 0.35 \\ s = 0, h = 0.05, & B < 0.35 \end{cases} \tag{2.7}$$

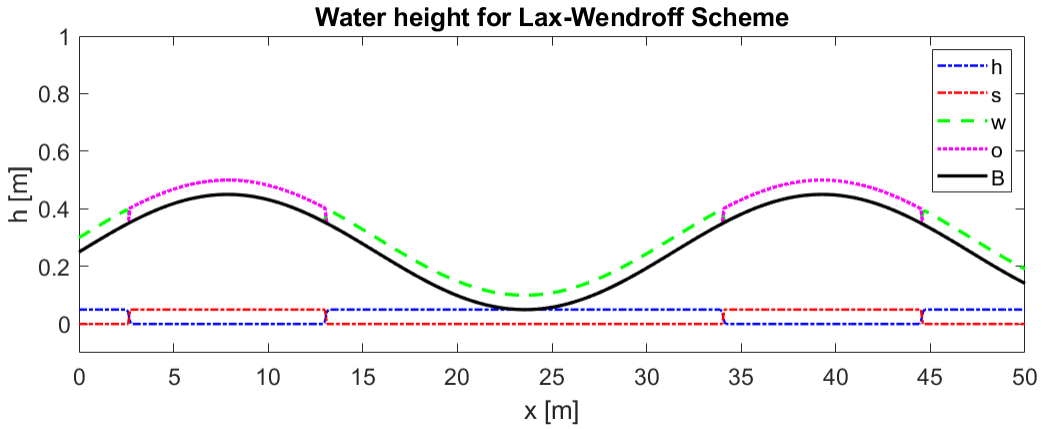


Figure 2.4: Modified initial conditions to include an existing snowpack.

For all simulations done with the new equations the variables B, h, and w, are consistant with the Saint-Venant equations listed in Eqn **??**. New variables were added to implement the snow, these being o and s. The two variables correlate to their respective water counterparts where o is the height of the snow above the bathymetry and s is the height of the snow including the bathymetry.

Beginning with the initial conditions shown in Figure 2.4, at $t = 15$ seconds the snow begins to melt. Figure 2.5 shows MATLAB plots at t =14.5, 15, 20, & 25. At 14.5 seconds the water has moved down the slopes in the topography and $w = B$ at the peaks of the topogrpahy as seen by the overlap of the lines. When the snow begins to melt there is a visible, if slight, difference between B and h as the initial 0.0075 meters of snow is melted and added to the water variable. After five seconds approximately half of the initial snow value has melted and by $t = 25$ approximately

two thirds has melted. After $t = 25$ seconds, 10 seconds after the melt behavior has started, the amount of snow visible in the plots is very low. This is caused by the low starting amount and as a result a cut-off volume is set for use in the sandbox to prevent infinitely small, non-visible amounts of snow in the sandbox.

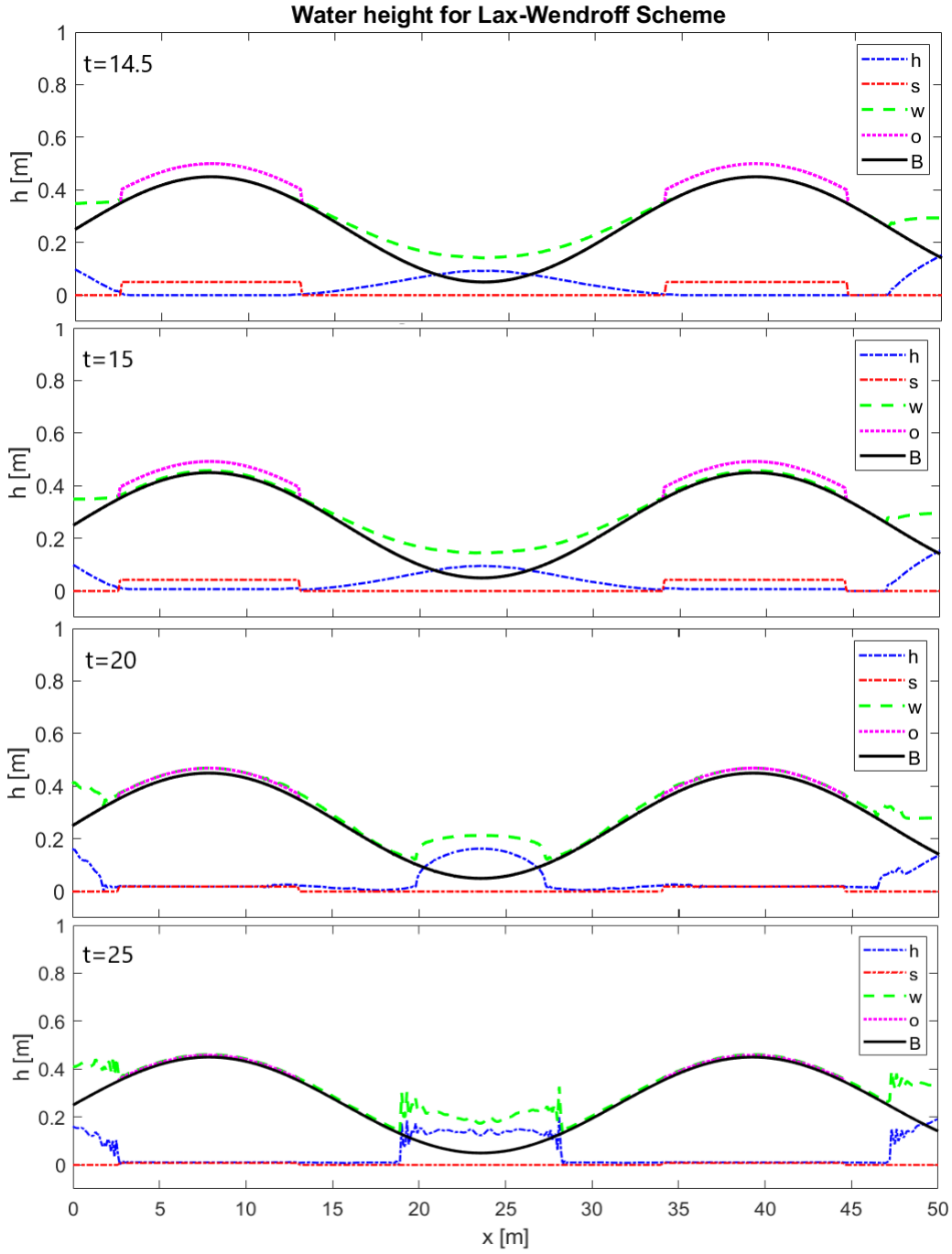


Figure 2.5: MATLAB plots displaying melt progress at t=14.5, 15, 20, and 25 seconds.

In the model at $t = 25$ there is a good amount of noise present like that at approximately $x = 20$. This likely due to the pool of water in the center, approximately $x = 25$, rising to a peak from the original volume of water before moving back outwards. This water that is moving outwards collides with the continued snowmelt runoff and the opposing motion causes relatively large amounts of noise.

2.3.3 Adding Snow

In order to add snow to the simulation, a set of equations similar to Eqn 2.5 needed to be implemented. While an initial snowpack can be added to the table by modifying the initial conditions as in Eqn 2.7, this does not allow for snow to be added while calculations are being done and the simulation is running. A 'rain' function was created to house the set of equations used to add water or snow mid-calculation.

The initial conditions used in Section 2.3.2 were modified to not include a snowpack. This results in a simulation that starts with portions of the bathymetry being dry. Figure 2.6 shows the graph of these conditions with the initial water only being placed below the assigned $B = 0.35$ critical height.

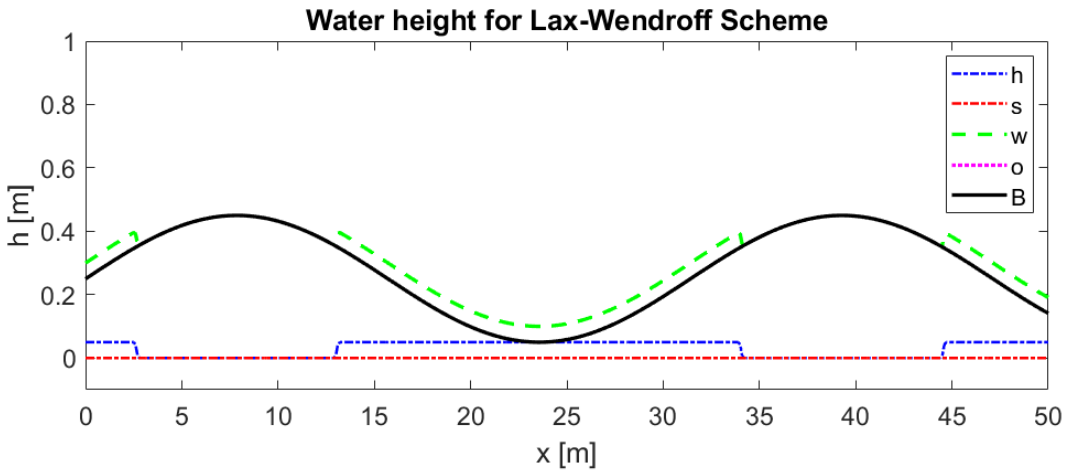


Figure 2.6: SWE simulation initial conditions not including an initial snowpack.

Equation 2.8 shows the equations used to add snow or water. Not utilizing the melting behavior, snow and water were added every two seconds for ten seconds.

Figure 2.7 shows plots at $t = 2$, 6, and 10 seconds displaying the increasing volume.

$$[h, s] = \begin{cases} h + 0.01 & B < 0.35 \\ s + 0.01 & B \geq 0.35 \end{cases} \tag{2.8}$$

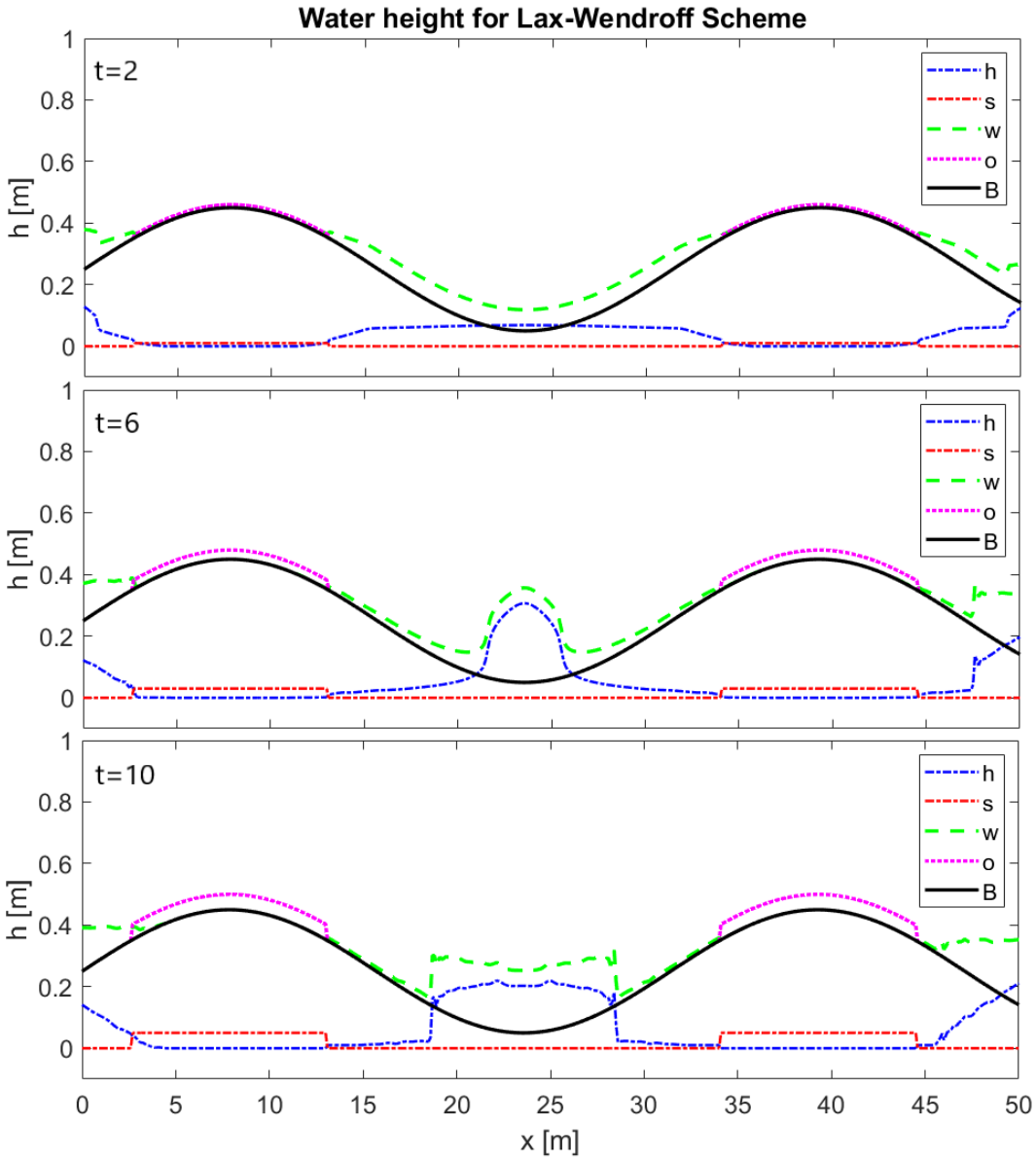


Figure 2.7: MATLAB plots showing increased snow and water at $t = 2$, 6, and 10 seconds.

2.4 Simulation Accuracy

There are multiple methods with which to calculate the volume of snowmelt and snowmelt runoff. Common methods, including degree day methods and regressive methods, take into account factors that are not available within the sandbox framework. Some of these factors include the air temperature, the albedo or reflectivity of the snowpack, and wind [14]. While the inclusion of these factors can result in a more accurate simulation of snowmelt runoff, they also increase the amount of processing required for both the snowmelt calculations specifically and keeping track of these variables within the sandbox.

These considerations were a large part of the decision to create a set of fairly simplified snowmelt equations. The ability to integrate these variables, such as albedo or ambient temperature, into the existing sandbox, without a large change in the base functionality, limited the available information with which to base the calculations on. Regardless of the limited available variables, an emphasis was put on creating a simulation that was as functionally and visually accurate as possible. The developed approach is time accurate and conserves water, along with being a consistent with the current level of accuracy with the SWE simulation in the sandbox.

CHAPTER 3: IMPLEMENTATION

The AR sandbox code distributed by UC Davis utilizes both C++ and OpenGL. C++ being the coding language the bulk of the sandbox is written in and OpenGL is the rendering Application Programming Interface (API). C++ is an International Organization for Standards (ISO) standardized coding language that is one of the most frequently used languages [16]. OpenGL is an open source API used for rendering graphics with specifications maintained by the Khronos Group [17]. OpenGL utilizes the processing chip present on many high end graphics card. This Graphical Processing Unit (GPU) is capable of doing many simultaneous calculations very quickly in order to support graphical tasks such as animation in video games and rendering of solid models in design programs. This API is not proprietary to a specific graphics card brand, unlike Cuda which is specific to NVidia hardware. Unlike other some other rendering APIs, OpenGL is strictly a graphics rendering system [17].

C++ and OpenGL are typically used together to create and run programs that utilize the GPU on a graphics card to provide additional processing power to solve mathematical equations and render images and textures. This is done by passing data between the computer hard drive (C++) and the graphics card (OpenGL). Figure 3.1 shows this initial divide between the retrieved camera data in the computer and the GPU. In order to move the data from one side to the other the programs use data structures including framebuffer objects, texture objects, and samplers to transfer, store, and calculate data as if they were graphical elements.

Initially the depth data from the camera (bathymetry) is stored in an array on the hard drive in C++. In order to perform the required mathematical functions, the data needs to be passed into a shader on the graphics card. This is done by passing the data to a texture object. Texture objects can be ‘cell-centered’ or ‘vertex-centered’. A cell-centered texture is when the point used for calculations is in the center of a cell and vertex-centered is when points on the corners of the cell are used. When

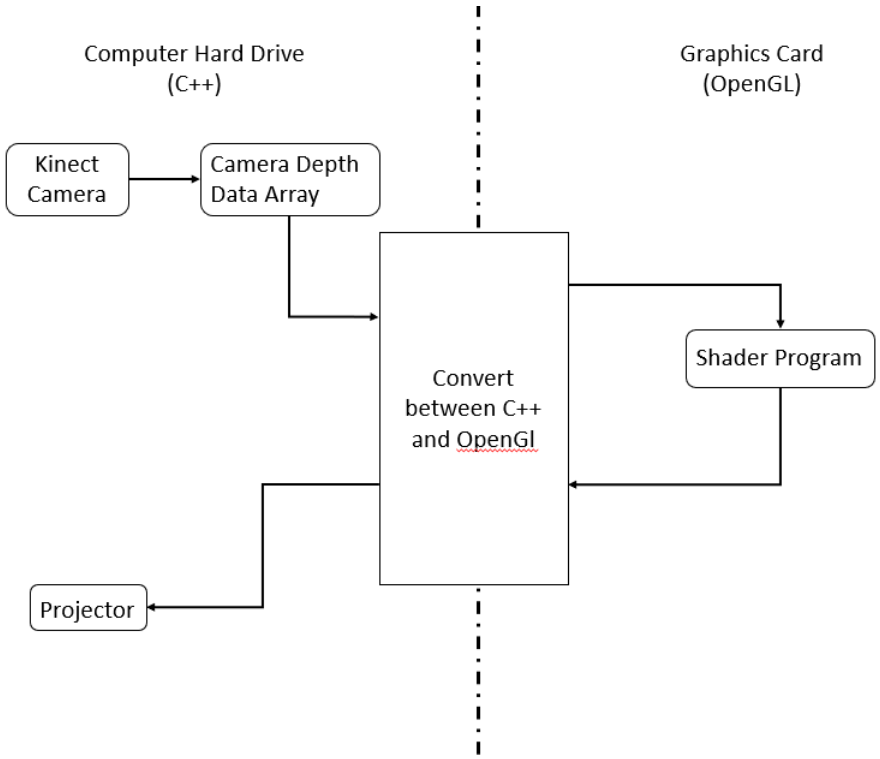


Figure 3.1: Flow chart displaying the separation of data between the hard drive and the GPU.

the shader is created the name, data type, and order of input variables is defined. Later, in the code when the shader is called the texture object is attached to an input variable. These texture object variables are called samplers within the AR sandbox code but other data types can also be used such as a float or other numerical data type.

Shaders are small programs that are run on a computer's graphics card and used for computation in the AR sandbox. When initially created in C++, the shader is given a footprint of the inputs it should receive. These inputs are given a name, such as being given the name 'sampler', and these variables are used within the shader file. A shader's output uses the Vector4 (vec4) RGBA data type called `gl_FragColor`. This output value is passed to the assigned framebuffer that is then passed to another data structure, such as a texture object.

A framebuffer is created similarly to a shader in that it is created, data structures

such as texture objects are attached, and it is later referenced within a function. When used in the AR sandbox program the framebuffer is called prior the shader. This is to ensure the proper data location is written to by the shader so the C++ program can access the data. Figure 3.2 shows an updated graphic of the computer to graphics card transfer where `gl_FragColor` is the shader output being sent to C++.

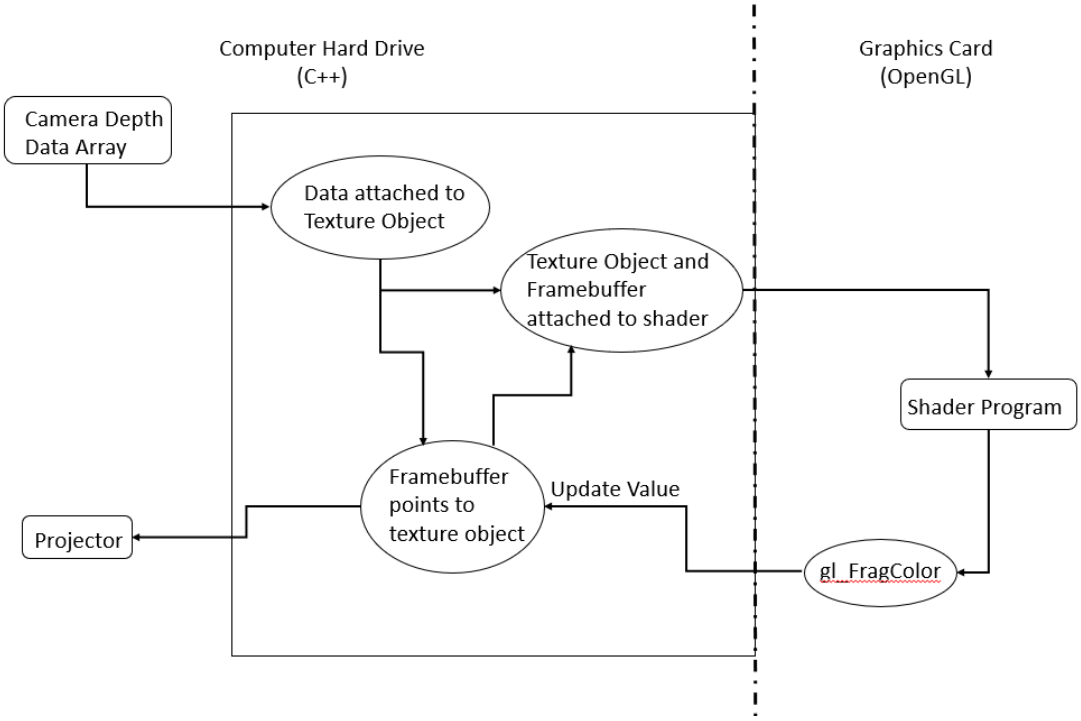


Figure 3.2: Flow chart displaying how data is converted between OpenGL and C++.

This back and forth transfer of data is used for most of the calculations necessary to run the AR sandbox as well as adding the color. The colors and contour lines of the topographic map are generated in this manner. Figure 3.3 shows a simplified example of how the depth data is passed between the computer hard drive and the graphics card to achieve this effect. The final step shown in the figure is the texture being sent to the projector to be viewed by the user.

3.1 Data Management

For the snowmelt simulation two sets of data structures were created: the 'snow' set and the 'freeze' set. The snow set includes the shader, texture object, and framebuffer

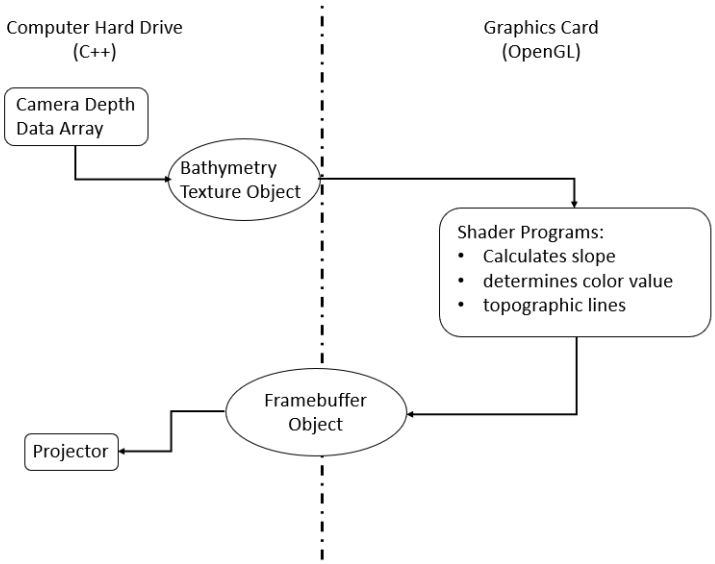


Figure 3.3: A flow chart displaying how the depth data is processed from C++ to OpenGL.

involved with the creation of the snow, and the freeze set includes the shader and framebuffer used to remove water once snow has been created. These two sets of structures, along with both the quantity texture object and the Runge-Kutta (RK) step shader, are used to simulate and visualize the snowmelt and snowpack in the sandbox. The quantity texture object is used for the shallow-water simulation and contains the three variables w, hu, and hv that are necessary for the SWE calculations. Unlike the snow shader, the freeze shader does not have a corresponding texture object. This is because the freeze shader modifies the quantity texture object as part of the water to snow conversion process.

3.1.1 Data Flow

When simulated rainfall is added to the table, either through the rain gesture or the rain function key, the program begins to run the shallow water simulation. The majority of the calculations for this simulation occur within a function named `runSimulationStep`. This function calls the shaders responsible for the math and texture updates involved for each step in the process. There are eight total steps in

the function. The shaders and their purpose is shown in Table 3.1.

Table 3.1: Shallow-water simulation steps from the AR Sandbox.

Number	**Step Description**	**Shader**
1	Calculate temporal derivative of most recent quantities.	N/A
2	Perform the initial Euler integration step.	`eulerStepShader`
3	Calculate temporal derivative of intermediate quantities.	N/A
4	Perform the final Runge-Kutta integration step.	`rungeKuttaStepShader`
5	Converts water to snow or removes snow from areas below critical height.	`snowShader`
6	Removes water from areas above critical height	`freezeShader`
7	Render all water sources and sinks additively into the water texture.	`waterAddShader`
8	Update the conserved quantities based on the water texture.	`waterShader`

The snowmelt simulation steps, running the snow and freeze shaders, takes place between steps four and five. This was done so that the changes made to the quantity values and the water in the table were rendered and updated properly without making too many changes to the base code. Due to the new steps being included in the function to run the shallow water simulation they aren't active until water is added to the table. This improves the program's efficiency by preventing unneeded calculations from being executed each timestep. Further discussion on the runtime impacts of the added functionality is included in Section 4.1.

Figure 3.4 shows how the sandbox algorithm with the new snow and freeze calculations. The snowmelt simulation and the two main shaders involved create a second branch in the shallow flow simulation. The main effects the snowmelt simulation has on the shallow flow simulation are the changes to the quantity value when removing or adding water in the freeze shader and Runge-Kutta shader respectively. With the use of the critical height checks, the sandbox can run the shallow flow simulations in

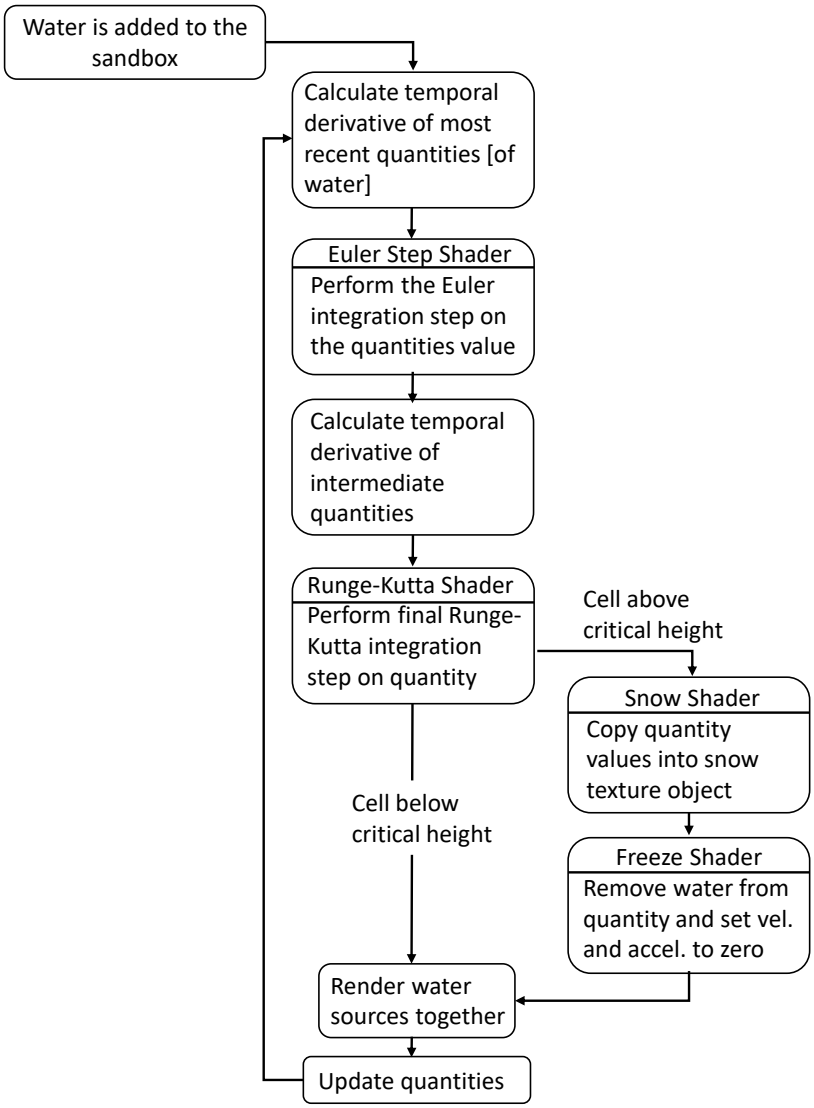


Figure 3.4: A flowchart demonstrating the flow of information for the shallow-water simulation.

areas below the critical height without needing to run through the snow and freeze shaders unnecessarily.

Additionally, the inclusion of the snowmelt simulation does not impact the standard functionality of the table. By setting the critical height above the max height of the table, an educator could remove the snow simulation without having to modify the code in any way.

3.1.2 Snow Shader

The snow shader converts the water above the critical height to snow and stores that value in the snow texture object. The snow texture object then holds the amount of snow in each cell to use in the next timestep. The second step in this shader is to remove the snow that exists below the critical height so that it can be treated as water in a different shader. The main algorithm of the snow shader is shown below as pseudocode.

```
IF bathymetry height Is Greater Than critical height
    Add water quantity to snow texture object
ELSE
    IF snow amount Is Greater Than minimum
        Multiply snow amount by melt rate
        Subtract from total snow amount
    ELSE
        Set snow amount to zero
    END IF
END IF
```

The above code illustrates the main two calculations of the snow shader: accumulating snow into a snowpack and removing snow from the snowpack. The outer condition detects the bathymetry height whereas the nested statement determines if the current snow amount is lower than a minimum amount. The use of these nested statements allows one shader to perform two actions on the snow texture object as opposed to two shaders, one accumulating snow and one melting it. The 'Set snow amount to zero' result of the nested statements ensures that there is not a continuously smaller amount of snow that is no longer visible within the sandbox.

3.1.3 Freeze Shader

This shader and framebuffer pair are used to update the quantity vector once snow has been added to the snow texture object. The main function of the shader is to remove the water added above the critical height. The bathymetry height is compared to the critical height value like in the snow shader and the amount of water in that piece is set to zero.

```
IF bathymetry height Is Greater Than critical height
    Set quantity equal to bathymetry.
END IF
```

In the AR sandbox code the quantity texture object holds the water amount as the combined value, w, of the bathymetry height and the water height. As such, in order to set the water height to zero the variable is set to the current bathymetry height. The water velocity and acceleration are also set to zero. The freeze shader works to negate possible issues that could arise if the water wasn't removed and only its velocity was set to zero. This helps to maintain an accurate total precipitation amount in the table and prevent errors such as Z-Fighting, where two overlapping textures can begin to glitch through each other when rendered.

3.1.4 Runge-Kutta Step Shader

This original purpose of this shader is to run the calculations for the Runge-Kutta integration step of the shallow water simulation. In order to run the second portion of the melt step, this shader was modified to include the addition of water to the quantity value. Two nesting if statements were added between the multiplication step and the attenuation of the new calculated value. These statements check the current bathymetry height against the critical height value similar to the snow and freeze shaders. The amount of snow is then multiplied by the melt rate and added to the value new quantity value. The full shader is shown in Appendix A.5.

3.2 Visualization

One of the objectives of this project was to develop the graphical texture for snowpack and melting snow. The snow texture needed to be distinct from the water and topography textures while keeping within the ability of the graphics card and projector as well as being consistent with the aesthetics of the sandbox. The final snow texture was generated utilizing a similar method as the water texture and the melting behavior is displayed through the use of transparency in combination with the snow

texture. An image of the final water and snow color textures are shown in Figure 3.5.

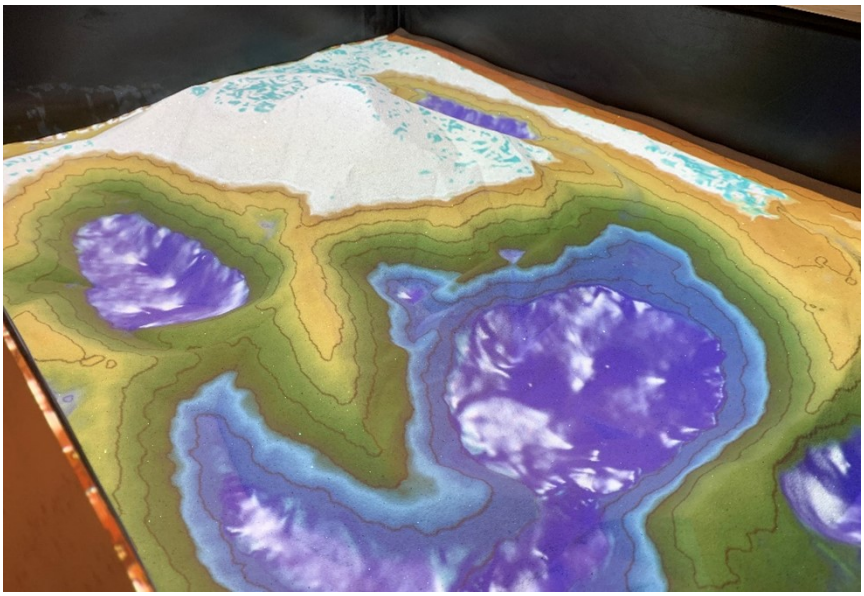


Figure 3.5: Picture of the snow and water colors as seen in the sandbox.

3.2.1 Color Mechanics

The AR sandbox uses multiple shaders to render the colors of the countour map and the water. The main shader used to create the water color texture is the '`SurfaceAddWaterColor`' shader. This shader checks if there is water or snow at a cell and then colors that cell according to the amount of water or snow present.

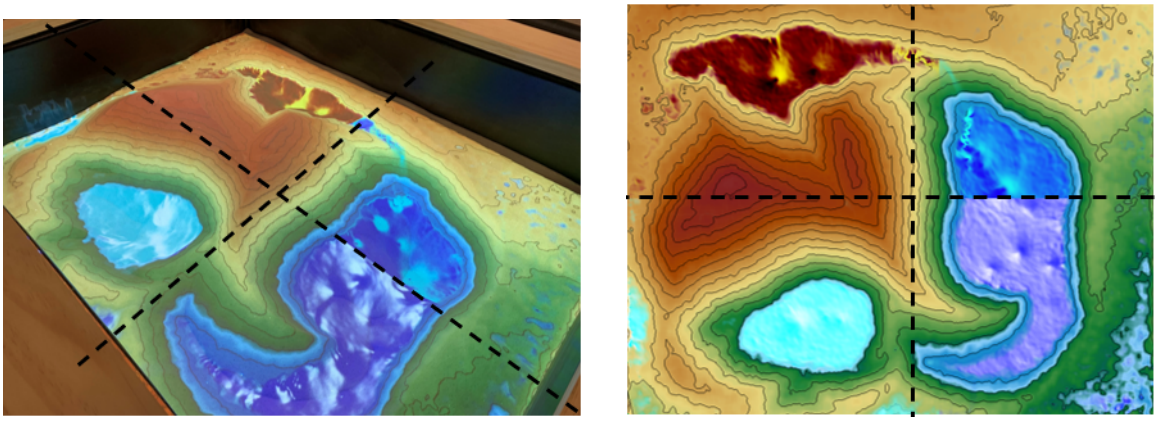
The main method used to determine if the snow or water is rendered in an area within the AR sandbox is an If statement that checks if there is a volume greater than 0.00001 of water or snow. This is the primary check to determine which color is being rendered dependent on the volume of the specific type of precipitation. Within these statements the color in a physical location can be further changed along with the general precipitation color.

In the sand surface is discretized into cells based on the resolution of the camera. The origin of the coordinate system is located at the bottom left of the table when viewed from the projector side. Within the shader the coordinate of a cell can be accessed in the `gl_FragCoord` variable that contains both x and y parts.

When the color of the precipitation is changed dependent on the `gl_FragCoord`, different sections of the sandbox can display different water colors. A pseudocode example of this cell-based logic is shown below.

```
IF gl_FragCoord X Is Greater Than X Given
    AND gl_FragCoord Y Is Greater Than Y Given
    Display water color one
ELSEIF gl_FragCoord X Is Greater Than X Given
    AND gl_FragCoord Y Is Less Than Y Given
    Display water color two
END IF
```

Figure 3.6 shows the results of a similar logic structure. The four quadrants of the table displaying water as four different colors. Using location to influence water color is a useful tool in the development of future capabilities such as visualy depicting ground cover or pollution.

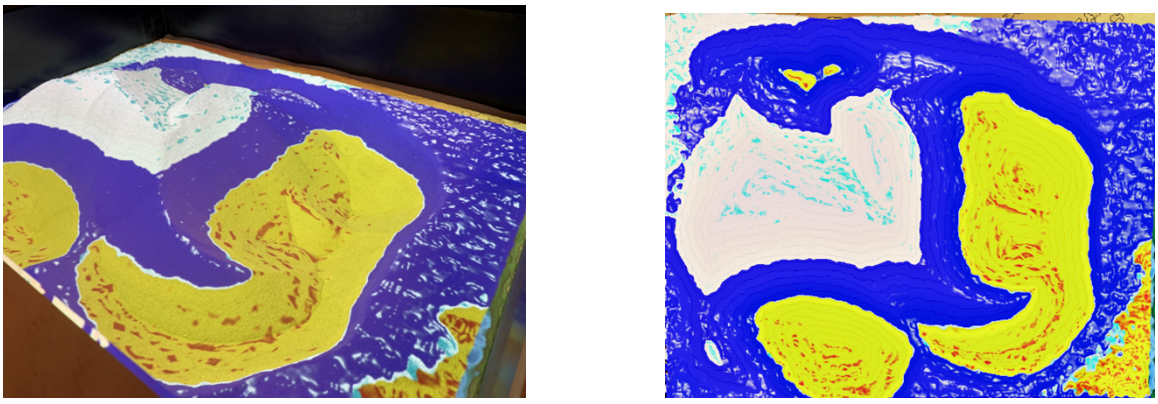


(a) (b)

Figure 3.6: A picture (a) and screenshot (b) displaying four different water colors.

The color of the precipitation can also be changed based on the z direction. Using a similar If-ElseIf method shown above using the `gl_FragCoord` variable, instead the bathymetry height is used. This allows for the color of the water or snow to be

changed as the height of the topography increases. Figure 3.7 shows this being used to change the color of snow in the sandbox.



(a) (b)

Figure 3.7: A picture (a) and screenshot (b) displaying different colored stripes in snow.

While the two methods to change the precipitation color based on location within the table are not currently used, they are nonetheless useful and could be used to highlight specific areas of the table during a demonstration.

3.2.2 Color Appearance

The sandbox uses the Vector RGBA datatype, specified as vec4 or vec3 in OpenGL, format to specify the colors of the snow and water. This format is used as the output for all shaders. In the water color shader the red, green, and blue values needed to project on the sand surface are returned. The other shaders use the three or four components to hold data such as snow quantity or discharge per cell. For the standard water color the sandbox uses a blue base color (`vec4(0.0, 0.0, 1.0, 1.0)` or hex #0000ff) and replaces the two zero value components with a variable that creates white noise. This variable '`colorW`' uses a custom method to generate white noise based on the volume of water in the table. This creates a shifting pattern that gives the impression of ripples due to the variable volume. The code creates the variable '`wn`' using a normalized value of the cells surrounding and the current one. Equation

3.1 shows the vec3 value that is then normalized to create the `wn` variable where Q indicates a value in the quantity sampler and L being the x and y dimensions of the cell.

$$[(Q_{x-1,y} - Q_{x+1,y})L_y], [(Q_{x,y-1} - Q_{x,y+1})L_x], [2L_xL_y] \tag{3.1}$$

The `wn` variable is then used in the equation for the variable `colorW`. The dot product of `wn` and the normalized vec3 color code for light blue is found and then raised to the power of 100.

The same method was used to color the snow texture with three main differences. The first being that the base color used is white (hex #ffffff) with the first two variables being replaced with `1-colorW`. Second, since there is no water in the quantity texture object in most areas with snow, the quantity variable in the equation for `wn` is replaced with the bathymetry height value. This results in a texture that has static patches of color as seen in the snow color instead of the moving ripple effect. This is due to the fact that unlike the quantity texture, the bathymetry texture does not vary as much. The third change is the color value used for the dot product step. In the water texture this value is a light blue (hex code #bfbfff) whereas in the snow texture this value is replaced with a dark blue (hex #0d11bf). This value is one factor that affects the color of the patches created with the white noise.

Both the water and snow color utilize the `colorW`, or `1-colorW`, in at least one position. Using the `colorW` variable in the color code is what allows for the generated white noise to appear. Since `colorW` changes based on the cell where the color is being calculated, where and how it is used in the `waterColor` variable changes the resulting color. Figure 3.8 shows the difference in the snow texture when changing the use of the `waterColor` variable.

The opacity of the water or snow is created by mixing the generated color with the color of the topography at that cell. The amount that the colors are mixed is

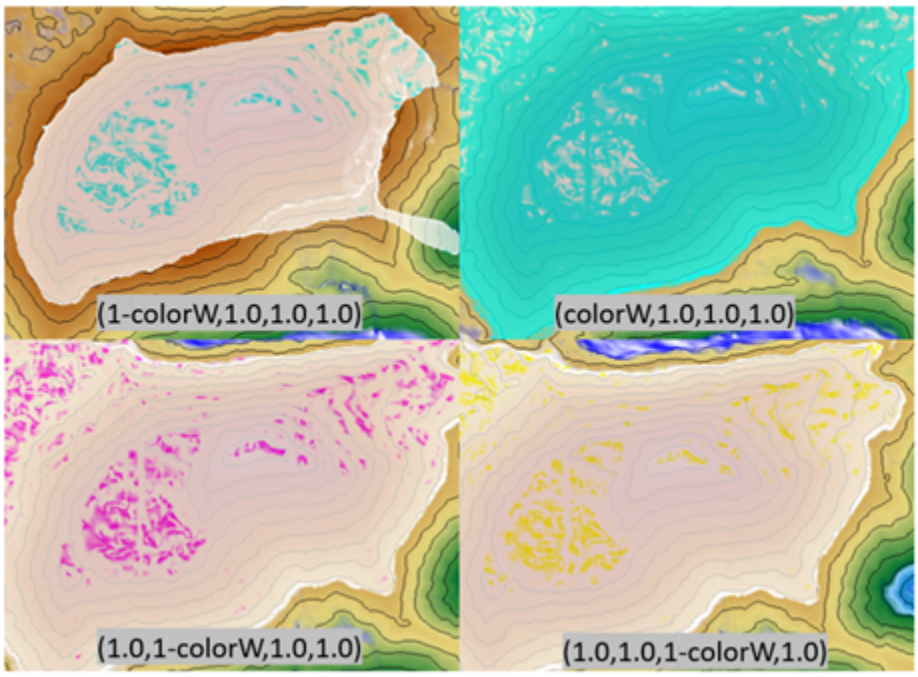


Figure 3.8: Snow color with the *colorW* variable used in four ways.

dependent on the amount of snow or water in that cell. This creates the effect of the precipitation fading away as the volume decreases or slowly becoming opaque as it builds up in different areas.

As seen in Figure 3.5 the final snow appearance is a solid white base with stationary light blue patches. This is in contrast to the water that has a solid dark blue base with varying light blue ripples. The final colors used were chosen for their similarity to colors seen in snow and ice in nature while being within the ability of the sandbox. The choice to use patches of color to highlight snow covered areas was made so that users can visually differentiate between a snowy area and other effects such as very high mountains or white sand with nothing projected onto it.

3.3 User Interface

In order to make the snowmelt simulation easier to use, a file was created to allow the user to change the critical height and melt rate variables. Utilizing this file means that a user would not need to adjust the two variables in all the locations they occur

or need to restart the sandbox. Due to the main screen being the sand projection, a second screen would be needed to display the terminal window and a text window with the snow configuration file. The '`snowConfig.cfg`' file includes comments that describes the two variables included and outputs the initial value to the terminal window on sandbox start-up and when the configuration file is saved. An example of the snow configuration file is included in Appendix A.3.

The snow configuration file works by using a file monitor system. While the AR sandbox is running it checks to see if the configuration file is modified. When the file is updated and saved the values of the critical height and melt rate are updated with the configuration file values. If the variables are not required to change during the use of the AR sandbox they can be set in the config file prior to startup using the sand as a screen.

3.4 Model Limitations

The snowpack and snowmelt model makes use of the existing sandbox algorithm and inputs to create the simulation described in this work. The shallow water simulation in the AR sandbox is a qualitative simulation. This is largely due to the need for the simulation to be real-time. The lower fidelity of the simulation allows for the GPU to run the necessary calculations quickly and project the results onto the sand without a large delay. The snowmelt simulation was created at a similar level of detail to help ensure the sandbox continues to run real time.

Additionally, the shallow flow and snowmelt calculation are completed cell by cell. Due to this, calculating the total volume of precipitation in the table can't be calculated reliably as the simulation runs. Along with the cell based calculations, while the snow volume is stored as only snow, the water volume is stored as the combined water and bathymetry height values. The bathymetry height can be subtracted but the result from quantity would not account for the changing discharge value and would increase calculation time. Further development of the AR sandbox software

outside the shallow water flow and snowmelt simulations would be necessary in order to accommodate quantitative analysis.

3.5 Code Distribution

The bulk of the coding on this project is contained to six existing files and the three new files. Table 3.2 shows the names of these files.

Table 3.2: Table showing the names of files that were created or modified.

Modified Files	New Files
rungeKuttaStepShader.fs	snowShader.fs
SurfaceAddWaterColor.fs	freezeShader.fs
WaterTable2.cpp	snowConfig.cfg
WaterTable2.h	
SurfaceRenderer.cpp	
SurfaceRenderer.h	

A main factor of this project was to make sure that the snowmelt simulation was easy and straight forward to implement into the existing sandbox framework. By limiting the amount of files that were modified it allows owners of existing sandboxes to easily include the snowmelt code. This could be done by simply replacing existing files with ones that include the edited code or by allowing users with existing modifications to easily incorporate the new code as well.

All modifications to existing code are tagged with a comment listing them as 'Snow Supporting' or simply 'Snow'. Searching for that keyword will allow users to find any changes from the base sandbox code and implement them as they see fit. All the code modified and created for this project is posted on GitHub at https://github.com/esmit248/ARSandbox_Snow.

CHAPTER 4: RESULTS

This project resulted in the creation of a working snowpack and snowmelt simulation for use in augmented reality sandboxes. The simulation runs in the current AR sandbox base code without needed to make any major modifications to allow for ease of implementation and minimal impact on the usability. Resources were created to show possible options for educational displays that can be used to alongside the new simulation.

4.1 Run-time Impact

One of the main concerns while developing the snowmelt simulation was the effects of the added code on the runtime of the sandbox. The goal was to have as little an effect on the runtime of the sandbox as possible with the goal being no impact at all. To quantify the runtime changes, both the original UC Davis sandbox code and the new modified code were run for five minutes with simulated water in the table. The new code was run with snow and without snow to asses the effects of just adding extra code and adding more running code. The code base includes logic to notify developers when the solution is processed too slowly to achieve realtime performance. In these cases the current timestep is terminated and the solution moves to the next timestep. These slow-downs can occur when there is a lot of fluid motion to resolve. In this case the added logic structures where a concern for runtime. A script was used to count the number of incomplete timesteps in the five minute duration of the test. For each run, the the global add water tool was used to add water for two seconds at ten seconds into the run. Each of the three variations was run ten times and the average incomplete timesteps were found. Table 4.1 shows these averaged values.

It was found that the inclusion of the snowmelt simulation did not significantly impact the runtime of the AR sandbox. The average number of incomplete timesteps did not change greatly between the original code and the modified code. There was

Table 4.1: Average number of incomplete timesteps in a five minute duration.

Code Iteration	**Average Incomplete Timesteps**
Original	3450.1
Snow, Inactive	3471.3
Snow, Active	4401.4

an increase seen when the snowpack and snowmelt simulation was active. Since there was no significant difference between the inactive modified code and original code, this is likely due to the additional steps and logic added for the simulation. The main impact to runtime that was found was the percent increase of water in the sandbox. When large amounts of water were added, such as the amount added when using the global tool, there is visible slow down in the shallow water simulation. This was found to be true for all three variations of the code. Removal of water through the built-in drain function lowers the visual lag caused by this.

4.2 Walk-through

The following section describes the usage and results of the added functions included in the snowmelt code. Figures are included to further display the visual aspects of these functions.

4.2.1 Adding snow to the sandbox

The simulated snowpack can be generated concurrently with adding water to the table. This can be done through built-in functions, the global 'rain' function or the rain gesture. Figure 4.1 shows the sandbox with no precipitation and after snow and water have been added to the table. The critical height used is 3.5. The blue contours begin at zero and represent the water table. Critical height is measure with respect to the water table line with positive values being above the water table and negative values below it.

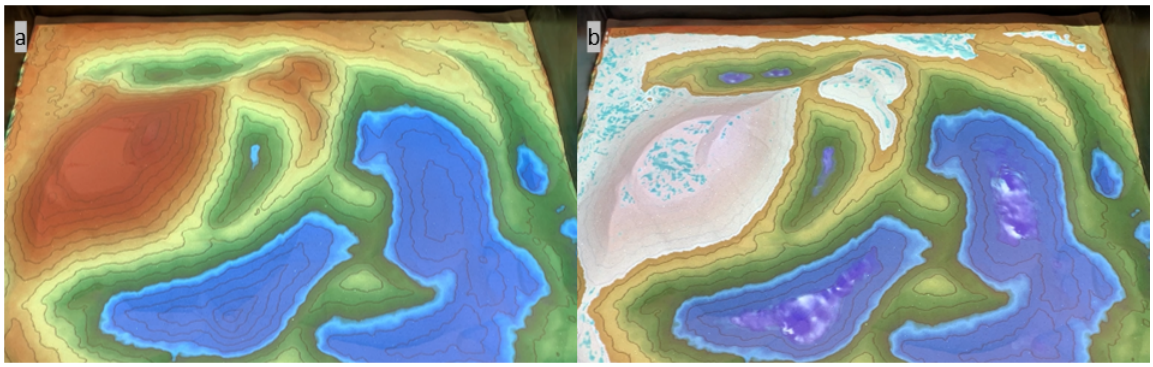


Figure 4.1: Sandbox display with no precipitation (a) and precipitation (b).

4.2.2 Melting the snow

While there is snow in the sandbox, the critical height can be changed in the config file to melt the snow. Figure 4.2 displays the melting behavior in four stages. The critical height starts at 3.5 and is adjusted to 7.5 in order to melt the snow that exists in the region between 3.5 and 7.5. As the snow melts the transparency of the snow increases with time. In the final picture (4) the water level of the "lake" in the lower right of the image is higher as a result of the snowmelt runoff.

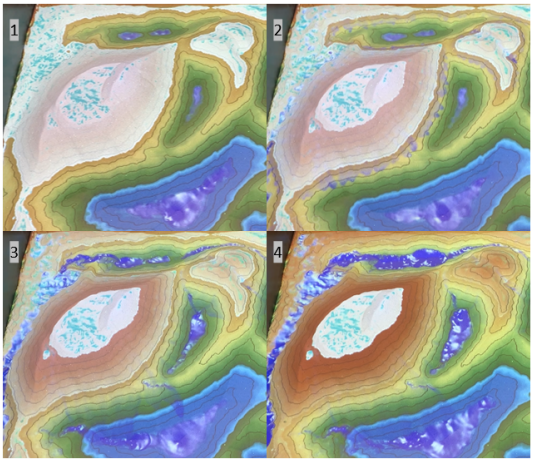


Figure 4.2: Snowmelt displayed in four stages. 1: Starting height, 2 and 3: two stages of melting in progress, 4: Fully melted at new height.

4.2.3 Changing bathymetry in snow covered area.

When the sand topography is changed, the water and snow update based on the new bathymetry value. In the case where the sand is moved in an area with snow, the snow will melt. This is shown in Figure 4.3.

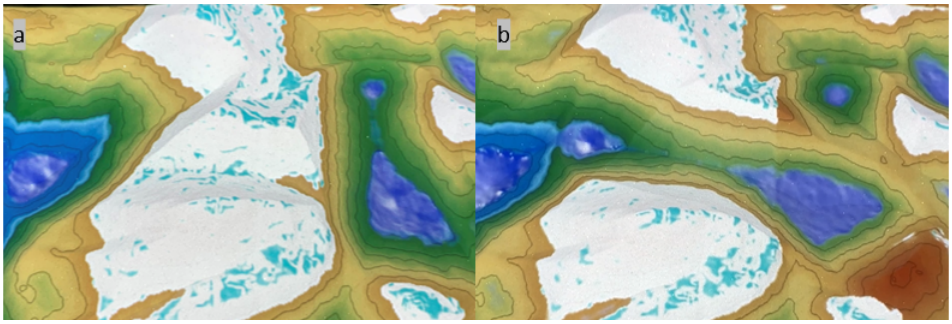


Figure 4.3: Snow melting due to a change in bathymetry: a) Before sand is moved and b) After snow has melted.

In areas with water that are then built up to be above the critical height the water will freeze into snow. This process can be seen in Figure 4.4 that shows the sandbox before and after a pool of water is frozen by changing the bathymetry.

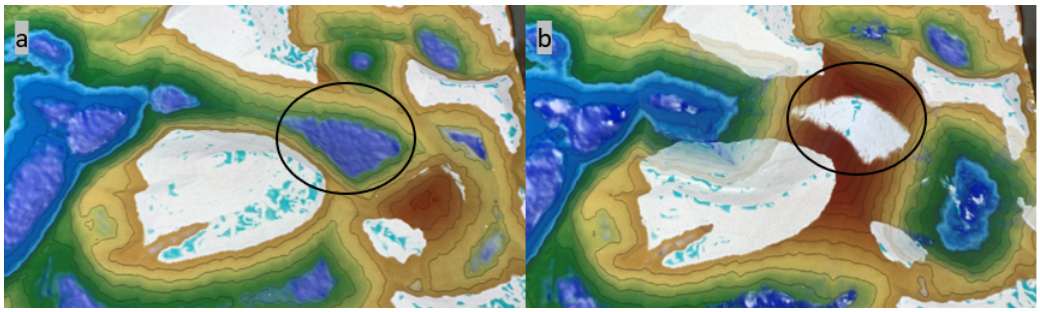


Figure 4.4: Water freezing to snow when sand topography is changed: a) before sand is moved and b) after sand is moved.

4.2.4 Completely Freezing the Sandbox

The critical height value can be set much lower than the water table height. This results in all existing water in the table to freeze. This is shown in Figure 4.5.

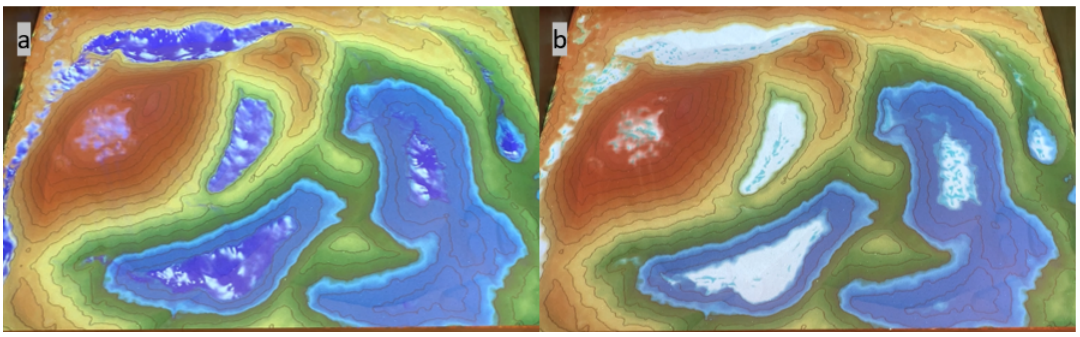


Figure 4.5: Converting all water in the sandbox to snow: a) Sandbox with no snow and b) Sandbox will all water turned to snow.

4.3 Educational Resources

A set of educational resources were created for the snowmelt simulation. These include a poster with guiding questions for the snowmelt simulation that can be displayed alongside of the sandbox during use as an interactive display. The background information can be used by parents or facilitators to encourage discussion on topics including snowmelt flooding and how precipitation changes due to altitude. A facilitator could lead a discussion on water resources based on the amount of precipitation that remains water or liquid with and without a snowpack. This displays how a snowpack acts as a battery for water and the availability of water at different times as well as the effects of the snowpack lowering initial available resources or causing flooding when that reservoir is melted. A similar method of generating a snowpack, discussing available water sources, and generating a smaller snowpack at a higher critical height can be used to display the impact of climate change on water resources and when they're available.

Full images of educational resources can be found in Appendix B.

CHAPTER 5: CONCLUSIONS

This research produced a snowpack and snowmelt simulation that was implemented within an AR sandbox. The real-time model is based on a critical or freeze height and utilizes a time based melting model. The simulation can be used to teach the concept of a snow line, a snowpack, and snowmelt runoff and its effects on stream flow and flooding.

5.1 Summary

AR Sandboxes are useful hands-on tools that can be utilized in both research and education. The base code developed by UC Davis was designed to be able to teach geographical concepts such as watersheds and runoff. The expansion of the AR sandbox code to include a simulated snowpack and snowmelt runoff allows for both concepts to be introduced to a younger audience. The modifications to the base code were contained to the minimum amount of files necessary in order to be easily implemented into existing AR sandboxes in the open source community.

The snow simulation consists of two main pieces: the accumulation of snow and the melting behavior. A new variable, critical height, was used to determine where in the AR sandbox the snow would be generated. This variable is dependant on the height of the sand topography (bathymetry) and allows for user-made changes in the topography to be reflected in the snowpack. Depending on the changes to the sand topography, existing snow may melt or be moved to a new location and existing water may be convert to snow. The overall water quantity is conserved in the sandbox as the freezing water is converted to snow and the melting snow is converted to water.

The melting behavior was created with a pair of recursive equations. One equation removes a percentage of the existing snowpack and another to adds that amount to the water variable. Utilizing a percentage value for the melting behavior creates a smooth transition between there being a snowpack and the snow being completely

melted. This is very noticeable with small amounts of snow where if a set amount melted per timestep there would be a more abrupt transition.

Educational resources were made as supplemental material for the snowpack and snowmelt simulation. These can be used as part of a hands-on display to demonstrate guiding questions for users to consider. The resources along with of the code created for the snowpack and snowmelt simulation is available online on GitHub at https://github.com/esmit248/ARSandbox_Snow. This continues the open source precedent of the AR sandbox and allows others in the community to use, and potentially build upon, the model.

It was found that the inclusion of the snowmelt did not have a noticeable impact on sandbox performance. When the original UC Davis code was compared to the modified code, with no snow being generated, the average amount of incomplete timesteps taken was similar. Yet, when snow was being generated, the average amount of incomplete timesteps was higher. It was found that this is The runtime test shows that the main factor that causes lag in the sandbox program is an increase in simulated water in the table and this consistent across both code versions.

5.2 Future Work

Further educational resources can be created for the simulation. These could include informative posters that display the use of the snowmelt simulation as a discussion tool for topics such as flooding, cyclical water resources, and climate change.

The creation of a basic snowmelt simulation for the AR Sandbox allows for more simulation to be created within the same scope. Further developing the freezing mechanic could lead to the creation of frozen lakes rather than the existing water being turned to snow and the simulation of avalanches. In a similar vein, the combination of the rain and snow mechanics could be combined with wind and temperature effects. This could create weather fronts or other weather phenomenon that are dependent on topography within the sandbox such as rain shadows, areas where mountains block

precipitation on one side.

REFERENCES

[1] USGS, "Snowmelt runoff and the water cycle." https://www.usgs.gov/special-topic/water-science-school/science/snowmelt-runoff-and-water-cycle?qt-science$_c enter_o bjects = 0qt - science_c enter_o bjects$.

[2] S. E. Reed, O. Kreylos, S. Hsi, *et al.*, "Shaping watersheds exhibit: An interactive, augmented reality sandbox for advancing earth science education," in *AGU Fall Meeting Abstracts*, vol. 2014, December 2014.

[3] M. D. Saint-Venant, "Theory of the non-permanent movement of water, with application to the floods of rivers and the introduction of tides into their beds," July 1871.

[4] J. F. Gerbeau and B. Perthame, "Derivation of viscous saint-venant system for laminar shallow water: Numerical validation," 2001.

[5] A. Kurganov and G. Petrova, "A second-order well-balanced positivity preserving central-upwind scheme for the saint-venant systems," in *Communications in Mathematical Sciences*, vol. 5, March 2007.

[6] L. Smith, J. McCormack, and Z. Xiong, "Augmented reality sandpit simulating ant colonies," in *2018 IEEE International Conference on Multimedia Expo Workshops (ICMEW)*, July 2018.

[7] K. S. Hale, J. M. Riley, C. Amburn, and N. Vey, "Evaluation of Augmented REality Sandtable (ARES) during Sand Table Construction," technical report, US Army Research Laboratory, January 2018.

[8] S. Guerin, "About the simtable." http://www.simtable.com/about/. Accessed 10/24/21.

[9] A. Afrooz, H. Ballal, and C. Pettit, "Implementing Augmented Reality Sandbox in Geodesign: A Future for Geodesign," in *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 4, 2018.

[10] T. L. Woods, S. Reed, S. Hsi, J. A. Woods, and M. R. Woods, "Pilot Study Using the Augmented Reality Sandbox to Teach Topographic Maps and Surficial Processes in Introductory Geology Labs," in *Journal of Geoscience Education*, vol. 64, June 2018.

[11] D. Marks, J. Kimball, D. Tingey, and T. Link, "The Sensitivity of Snowmelt Processes to Climate Conditions and Forest Cover During Rain-on-Snow: A Case Study of the 1996 Pacific Northwest Flood.," in *Hydrological Processes*, vol. 12, December 1998.

[12] D. Clow, "Changes in the Timing of Snowmelt and Streamflow in Colorado: A Response to Recent Warming," in *Journal of Climate*, vol. 23, May 2010.

[13] Y. Qin, J. T. Abatzoglu, S. Siebert, *et al.*, "Agricultural Risks From Changing Snowmelt.," *Nature Climate Change*, vol. 10, 2020.

[14] US Department of Agriculture, "National engineering handbook," in *Hydrology*, ch. 11, 2004.

[15] J. Adams, "1D Shallow Water Equations Dam Break." (https://www.mathworks.com/matlabcentral/fileexchange/46475-1d-shallow-water-equations-dam-break), MATLAB Central File Exchange, May 2014. Retrieved April 29, 2021.

[16] "Information." https://www.cplusplus.com/info/. Accessed 10/24/21.

[17] Khronos Group, "What is OpenGL." https://www.khronos.org/opengl/wiki/FAQ#What_is_OpenGL.3F. Accessed 10/24/21.

APPENDIX A: C++ Code

A.1 Snow Shader code

```
#extension GL_ARB_texture_rectangle : enable
uniform float criticalheight;
uniform float meltRate;
uniform sampler2DRect snowSampler;
uniform sampler2DRect quantitySampler;
uniform sampler2DRect bathymetrySampler;
void main()
    {\*Calculate if surface height is higher than the critical height*\
    vec3 snowAmt = (0.0,0.0,0.0);
    float B=(texture2DRect(bathymetrySampler,
        vec2(gl_FragCoord.x-1.0,gl_FragCoord.y-1.0)).r+
        texture2DRect(bathymetrySampler,
        vec2(gl_FragCoord.x,gl_FragCoord.y-1.0)).r+
        texture2DRect(bathymetrySampler,
        vec2(gl_FragCoord.x-1.0,gl_FragCoord.y)).r+
        texture2DRect(bathymetrySampler,
        vec2(gl_FragCoord.xy)).r)*0.25;
    vec3 q=texture2DRect(quantitySampler,gl_FragCoord.xy).rgb;
    float w = q.r;
    vec3 snow = texture2DRect(snowSampler, gl_FragCoord.xy).rgb;
    float snowMelt = meltRate/100000;
    if(B>=criticalHeight)
        {snowAmt.r = snow,r + (w-B);
    }else
        {snowAmt.r = snow.r;
        if(snowAmt.r>0.0001)
            {
            snowAmt.b = snow,r * snowMelt;
            snowAmt.g = snow.g + snowAmt.b;
            snowAmt.r = snowAmt.r - snowAmt.g;
        }else
            {snowAmt.r = 0.0;
            }
        }
    gl_FragColor=vec4(snowAmt,0.0);
    }
```

A.2 Freeze Shader Code

```
#extension GL_ARB_texture_rectangle : enable
uniform float criticalHeight;
uniform sampler2DRect quantitySampler;
uniform sampler2DRect bathymetrySampler;
void main()
    {float B=(texture2DRect(bathymetrySampler,
        vec2(gl_FragCoord.x-1.0,gl_FragCoord.y-1.0)).r+
        texture2DRect(bathymetrySampler,
        vec2(gl_FragCoord.x,gl_FragCoord.y-1.0)).r+
        texture2DRect(bathymetrySampler,
        vec2(gl_FragCoord.x-1.0,gl_FragCoord.y)).r+
        texture2DRect(bathymetrySampler,
        vec2(gl_FragCoord.xy)).r)*0.25;
```

```
    vec3 q = texture2DRect(quantitySampler,gl_Fragcoord.xy).rgb;
    vec3 noHu;
        gl_FragColor = vec4(q,0.0);
        if(B>=criticalHeight)
            {noHu = vec3(B,0.0,0.0);
            gl_FragColor=vec4(noHu,0.0);
            }
    }
```

A.3 Snow Configuration File Code

```
/*The value below is the critical height or freeze line. The water
table is at 0. Positive values are above the water table.*/
3.5
/*The value below is the Melt Rate. The default value is 5.
Larger numbers will make the snow melt faster.*/
5.0
```

A.4 Precipitation Coloring Section of SurfaceAddWaterColor.fs

```
void addWaterColor(in vec2 fragCoord, inout vec4 baseColor)
    {
    /*Calculate the water column height above this fragment*/
    float b=(texture2DRect(bathymetrySampler,
    vec2(gl_FragCoord.x-1.0,gl_FragCoord.y-1.0)).r+
    texture2DRect(bathymetrySampler,
    vec2(gl_FragCoord.x,gl_FragCoord.y-1.0)).r+
    texture2DRect(bathymetrySampler,
    vec2(gl_FragCoord.x-1.0,gl_FragCoord.y)).r+
    texture2DRect(bathymetrySampler,
    vec2(gl_FragCoord.xy)).r)*0.25;
    float waterLevel=texture2DRect(quantitySampler,waterTexCoord).r-b;
    vec2 snow = texture2DRect(snowSampler,waterTexCoord).rg;
    /*Check if surface is under water:*/
    if(waterLevel>0.000001)
        {vec3 wn=normalize(vec3((texture2DRect(quantitySampler,
            vec2(waterTexCoord.x-1.0,waterTexCoord.y)).r-
            texture2DRect(quantitySampler,
            vec2(waterTexCoord.x+1.0,waterTexCoord.y)).r)
            *waterCellSize.y,(texture2DRect(quantitySampler,
            vec2(waterTexCoord.x,waterTexCoord.y-1.0)).r-
            texture2DRect(quantitySampler,
            vec2(waterTexCoord.x,waterTexCoord.y+1.0)).r)
            *waterCellSize.x,2.0*waterCellSize.x*waterCellSize.y));
        float colorW=pow(dot(wn,
            normalize(vec3(0.075,0.075,1.0))),100.0)*1.0-0.0;
        vec4 waterColor=vec4(colorW,colorW,1.0,1.0);
        baseColor=mix(baseColor,waterColor,
            min(waterLevel*waterOpacity,1.0));
        }
    /*Check if surface has snow:*/
    if(snow.r>0.000001)
        {vec3 wn=normalize(vec3((texture2DRect(bathymetrySampler,
            vec2(waterTexCoord.x-1.0,waterTexCoord.y)).r-
            texture2DRect(bathymetrySampler,
```

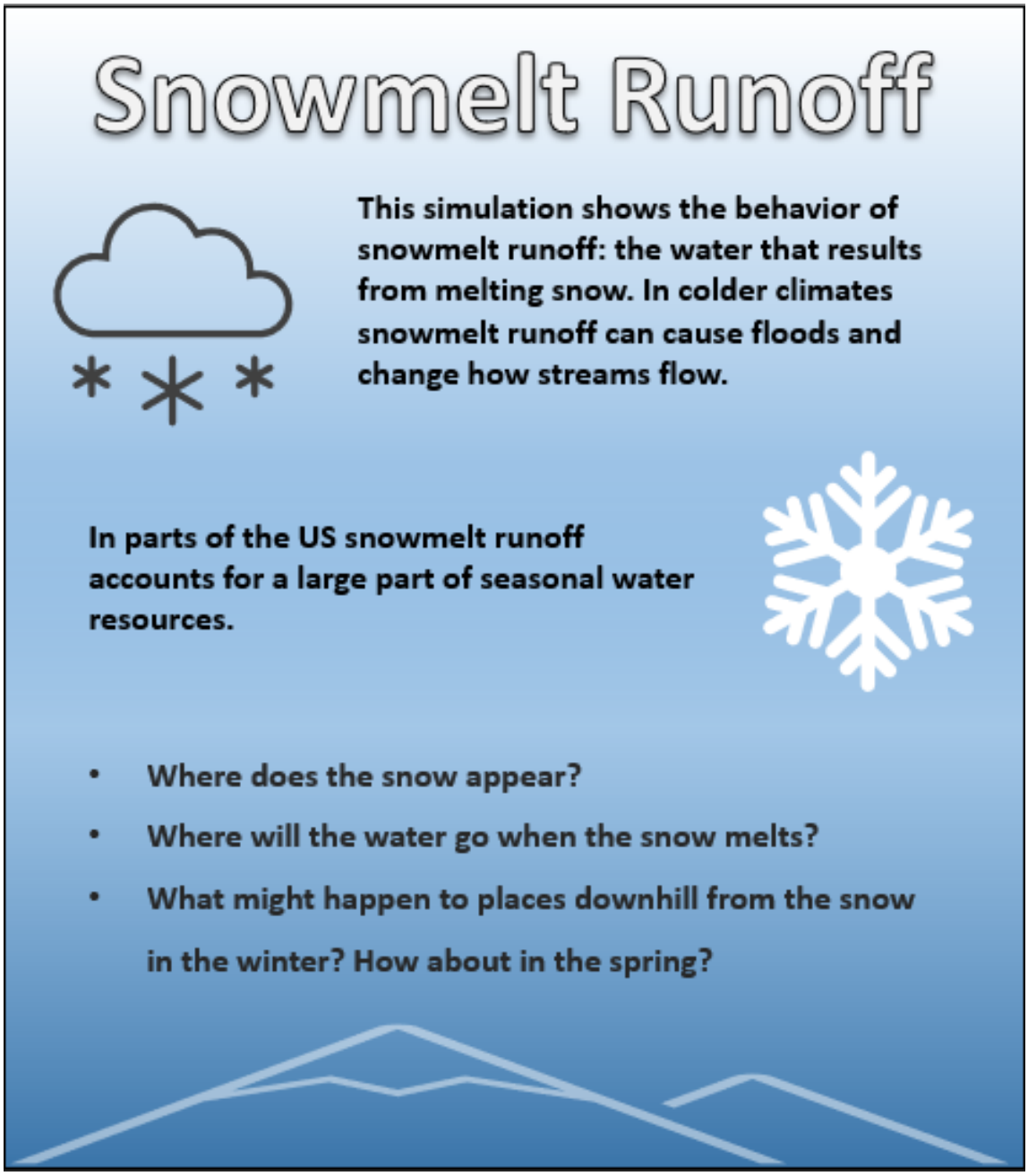
```
            vec2(waterTexCoord.x+1.0,waterTexCoord.y)).r)
            *waterCellSize.y,(texture2DRect(bathymetrySampler,
            vec2(waterTexCoord.x,waterTexCoord.y-1.0)).r-
            texture2DRect(bathymetrySampler,
            vec2(waterTexCoord.x,waterTexCoord.y+1.0)).r)
            *waterCellSize.x,2.0*waterCellSize.x*waterCellSize.y));
        float colorW=pow(dot(wn,
            normalize(vec3(0.0529,0.0702,0.75))),100.0)*1.0-0.0;
        vec4 waterColor=vec4(1-colorW,1.0,1.0,1.0);
        baseColor=mix(baseColor,waterColor,
            min(snow.r*waterOpacity,1.0));
        }
    }
```

A.5 Runge-Kutta Step Shader

```
#extension GL_ARB_texture_rectangle : enable
uniform float stepSize;
uniform float attenuation;
uniform float criticalHeight;
uniform float meltRate;
uniform sampler2DRect quantitySampler;
uniform sampler2DRect quantityStarSampler;
uniform sampler2DRect derivativeSampler;
uniform sampler2DRect snowSampler;
uniform sampler2DRect bathymetrySampler;
void main()
    {/* Calculate the Runge-Kutta step: */
    vec3 q=texture2DRect(quantitySampler,gl_FragCoord.xy).rgb;
    vec3 qStar=texture2DRect(quantityStarSampler,gl_FragCoord.xy).rgb;
    vec3 qt=texture2DRect(derivativeSampler,gl_FragCoord.xy).rgb;
    vec3 newQ=(q+qStar+qt*stepSize)*0.5;
    /* Adds melted snow to quantity: */
    vec3 snow = texture2DRect(snowSampler,gl_FragCoord.xy).rgb;
    vec3 holdMelt = vec3(0.0,0.0,0.0);
    float B=(texture2DRect(bathymetrySampler,
    vec2(gl_FragCoord.x-1.0,gl_FragCoord.y-1.0)).r+
    texture2DRect(bathymetrySampler,
    vec2(gl_FragCoord.x,gl_FragCoord.y-1.0)).r+
    texture2DRect(bathymetrySampler,
    vec2(gl_FragCoord.x-1.0,gl_FragCoord.y)).r+
    texture2DRect(bathymetrySampler,
    vec2(gl_FragCoord.xy)).r)*0.25;
    float snowMelt = meltRate/100000;
    if(B<criticalHeight)
        {holdMelt.r = snow.r;
        if(holdMelt.r>0.0001)
            {holdMelt.g = holdMelt.g + holdMelt.r * snowMelt;
            newQ.x = newQ.x + holdMelt.g;
            holdMelt.r = holdMelt.r - holdMelt.g;
            }
        }
    newQ.yz*=attentuation;
    gl_FragColor=vec4(newQ,0.0);
    }
```

APPENDIX B: Educational Resources

B.1 Snowmelt Runoff Guiding Question Poster



B.2 General Snow Simulation Poster