CSCI 370 Final Report

Walker Water - Watershed Analytics

Nathan Howland
Luke Hart
Cole Rincon
Chase McDonald

Revised June 15, 2023

CSCI 370 Summer 2023

Prof. Amelia Read

Table 1: Revision history

Revision	Date	Comments
1.0	May 15, 2023	Completed Sections: I. Introduction
1.1	May 16, 2023	Completed Sections: II. Functional Requirements III. Non-Functional Requirements IV. Risks
1.2	May 18, 2023	Completed Sections: V. Definition of Done
1.3	May 19, 2023	Completed Sections: XIV. Team Profile Modified Sections ● References ● Appendix A
2.0	May 26, 2023	Completed Sections: VI. System Architecture
3.0	May 31, 2023	Completed Sections: VIII. Software Test and Quality IX. Project Ethical Considerations
4.0	June 11, 2023	Completed Sections X. Results XI. Future Work XII. Lessons Learned
4.1	June 12, 2023	Updated All Sections
4.2	June 13, 2023	Completed Section: VII. Technical Design
4.3	June 15, 2023	Implement suggestions from report review

Table of Contents

I. Introduction **4**
II. Functional Requirements **4**
III. Non-Functional Requirements **4**
IV. Risks **5**
V. Definition of Done **6**
VI. System Architecture **6**
Input/Output Specifications 6
System UML 7
VII. Technical Design **9**
Grid Cell Proportioning 9
VIII. Software Test and Quality **11**
Unit Tests 11
Functional Tests 12
Code Reviews 12
IX. Project Ethical Considerations **13**
X. Results **14**
Features Implemented 15
XI. Future Work **16**
Performance Enhancements 16
Improved Accuracy in Groundwater Analysis 16
Flow Analysis 16
Higher Resolution Grid Data 16
XII. Lessons Learned **17**
XIII. Acknowledgements **18**
XIV. Team Profile **18**
References **19**
Appendix A – Key Terms **19**

I. Introduction

Walker Water is a software development company focused on delivering high quality irrigation water management tools. Recent developments emphasize snow water supply forecasting methods. This involves analysis and measurement of snow water equivalent (SWE) content within the snowpack, resulting in basin-wide SWE estimates, and snowmelt runoff modeling. The benefit to water managers and end users is to reliably predict snow water availability to reservoir fill and streams during runoff. This knowledge allows irrigators to determine if the upcoming season will be in short or long supply of water. A previous field session team began this process by creating an algorithm which would trace the path of a single “marble”, representing a drop of water, as it followed the path of least resistance down the mountain. We will add on to their software by modeling the snowmelt and capturing the amount of water at every point on the land at any given time in the simulation. We will receive elevation data and a map of the SWE directly from Walker Water, in addition to any technical information necessary to build an accurate simulation.

II. Functional Requirements

1. The simulation must accept a CSV file representing elevation data for a mountain at points on a grid. This input file includes UTM coordinates, latitude, longitude, elevation data, snow water equivalent data, and runoff coefficient data, for every data point.
2. The simulation must also read reservoir data stored in CSV files to get information on reservoir names, positions, initial water volumes, and capacities.
3. The program must output a binary file representing the amount of water at each point in the grid at the end time, as well as the volume of water contained in each reservoir.
4. The program must be able to produce results for any generic given period of time.
5. This iteration of the simulation will assume surface friction is negligible but will include an absorption coefficient to begin the complex modeling of how much water is absorbed into the ground as it makes its way down the mountain.

III. Non-Functional Requirements

1. The program is written in C++, since that is what previous iterations of the program were developed with.
2. Development must be done on Windows machines in the Visual Studio IDE to ensure new developments are compatible with previous software
3. The program’s output will be a binary file with water supply in each grid cell at each time step. The specifications of the file format will be provided to Walker Water for input to their 2D GIS basemap and 3D visualization system.
4. The program is delivered to the client as source code which can be modified and compiled as the client sees fit.

IV. Risks

Table 2: Technology Risks

Risk	Likelihood	Impact	Mitigation Plan
Errors in Existing Simulation: Mr. Walker informed the team that in the simulation developed last Fall, water traveled uphill, which is not correct.	Very Likely	Minor	We spent time correcting errors in the old code and carefully familiarized ourselves with the previous field session group's software so that we understood their work and could correct any other errors.
Inaccurate Simulation Results: If the final product developed by Walker Water has errors, drought predictions could be compromised and agriculture in the region would suffer.	Unlikely	Major	We consulted with water resource experts through Walker Water to ensure that our simulation functions as accurately as possible.
Performance Issues: Since the simulation may need to process millions of water molecules, it may run slowly.	Likely	Minor	We invested time into optimizing our simulation to ensure it runs as quickly as possible.

Table 3: Skill Risks

Risk	Likelihood	Impact	Mitigation Plan
C++: The simulation needs to be programmed in C++, so limited experience in this language could be a risk	Unlikely	Major	All team members are proficient in C++ so we do not anticipate our skill here to be an issue.
OpenGL: Walker Water's animation software is written in OpenGL, so limited knowledge of OpenGL could be a risk	Unlikely	Minor	The team will work through OpenGL tutorials if deemed necessary, however since our simulation will mostly work with text-based inputs and outputs, detailed knowledge of OpenGL may not be necessary.

V. Definition of Done

The product is done when the simulation can produce a binary file mapping the path of many water molecules traveling through the terrain. The simulation should be able to produce this result for any given length of simulation time. The client will test the software by displaying the contents of the binary file with both a 2D GID based map and 3D OpenGL visualization renderer. The client will verify that the water follows a path along natural drainages in a downhill direction. Additionally, the client will ensure that water stops and pools in natural catchments. The product will be delivered as a source code repository that the client can download, run, and modify in the future.

VI. System Architecture

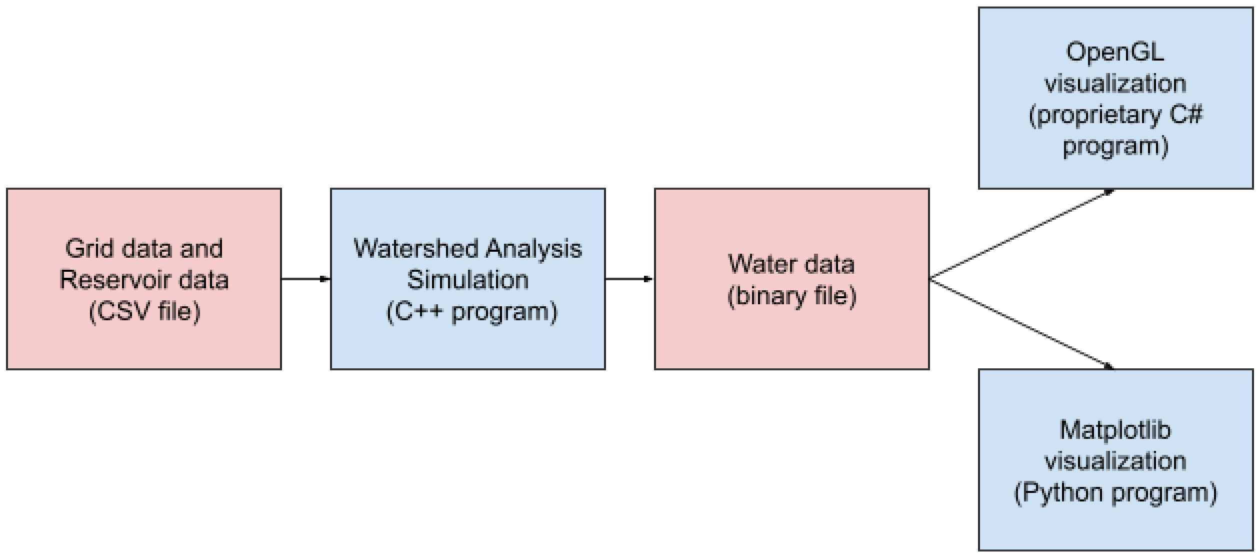
Input/Output Specifications

Walker Water provided simulation data through a set of CSV files. This data includes all of the following:

- An assigned integer row and column for each cell in the rectangular grid that we are simulating.
- An elevation for each cell in the grid.
- The latitude/longitude and UTM coordinate of each grid cell so we can understand each grid cell's position.
- An initial SWE for each cell in the grid, which lets us assign an initial depth of water to each cell in the simulation.
- A runoff coefficient for each cell that approximates how water will interact with the material on the terrain of that cell.
- A list of reservoirs with names, capacities, initial volumes, outline polygons, and spill location coordinates.

Our watershed analysis program reads this data and runs the simulation. We then output the simulation results to a binary file in a format agreed upon between Walker Water and the team. This binary file contains information of the depth of water in every grid cell at every point in the simulation so that the simulation can be completely recreated and animated from this file. Also included is the volume of every reservoir at every iteration of the simulation. Walker Water is then able to display the contents of this binary file in their OpenGL 3D graphics program. For testing purposes, we created a simple and lightweight display program in Python using matplotlib to visualize our results. This process is summarized in figure 1.

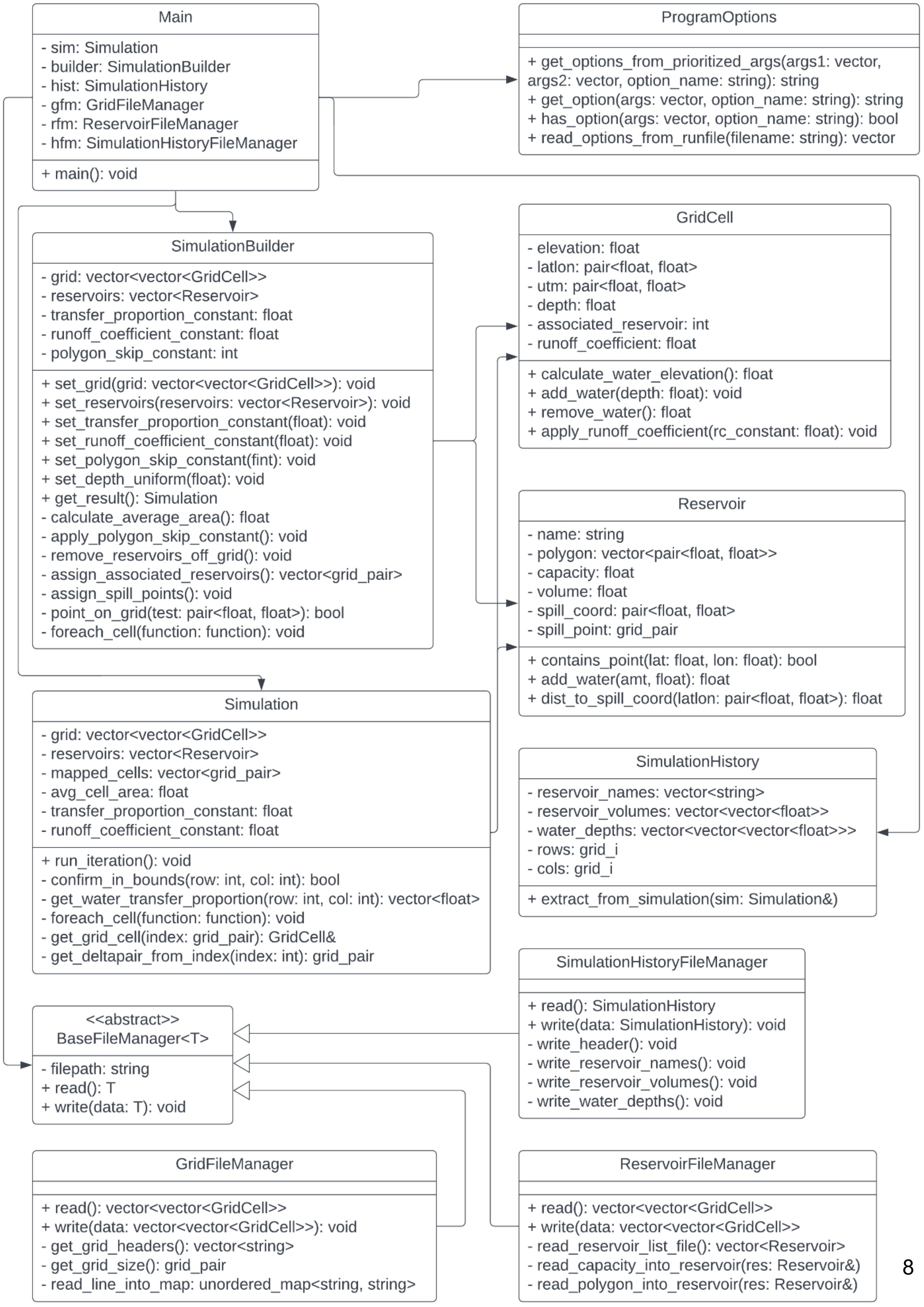
Figure 1: Input-Output Specification Diagram


Grid data and Reservoir data (CSV file)
Watershed Analysis Simulation (C++ program)
Water data (binary file)
OpenGL visualization (proprietary C# program)
Matplotlib visualization (Python program)

System UML

The simulation architecture is summarized in figure 2. Program execution begins in the main method. Here, a ReservoirFileManager and a GridFileManager is used to read file data for the specified file paths. This file data is then stored in a 2D vector of GridCell objects to represent every cell in a 2 dimensional section of land, and a 1D vector of Reservoir objects to enumerate all the reservoirs on that section of land. Each grid cell has a latitude/longitude coordinate, a UTM coordinate, an elevation, a depth of water, and a runoff coefficient based on the type of surface material in the area of that grid cell. This data is passed into the SimulationBuilder object. The SimulationBuilder collects simulation data and user-defined parameters, then handles some initial setup for the data before it is ready to be simulated, such as assigning grid cells to their correct associated reservoirs. Once the caller has supplied all data, they can call SimulationBuilder.get_result() to obtain a Simulation object. The Simulation object handles calculating how much water should be moved to each cell and how water should interact with reservoirs in each iteration based on current water depths and reservoir volumes. The main method then instructs the Simulation to complete a given number of iterations and queries the state of the simulation object at each iteration to generate output data, which is stored in a SimulationHistory object. Main will then use the SimulationHistoryFileManager to write the required output.

Figure 2: System UML

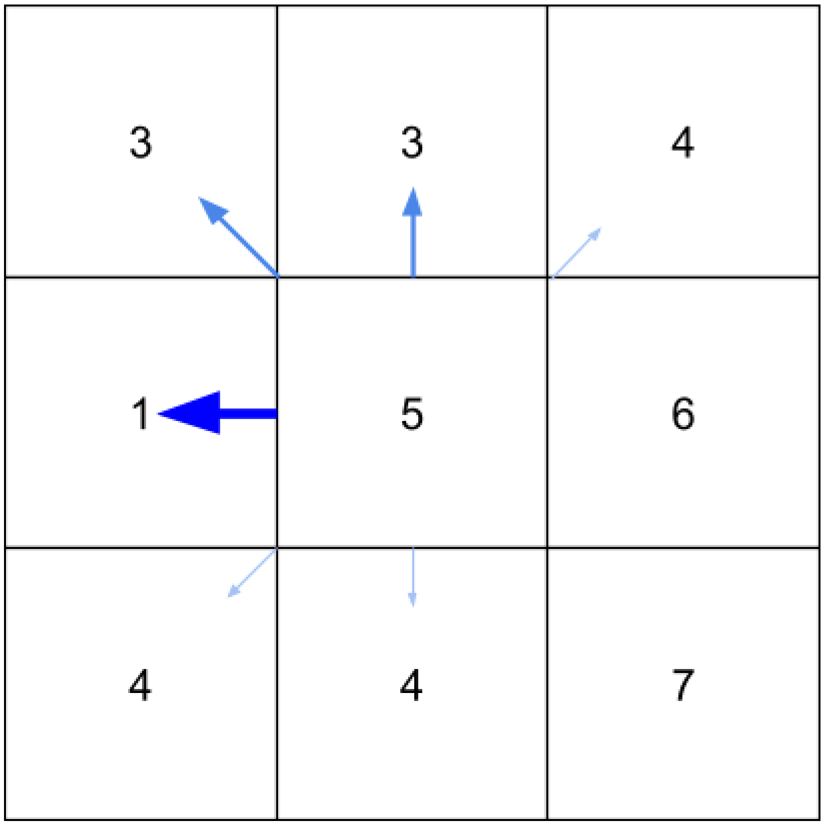

Main
- sim: Simulation
- builder: SimulationBuilder
- hist: SimulationHistory
- gfm: GridFileManager
- rfm: ReservoirFileManager
- hfm: SimulationHistoryFileManager
+ main(): void
ProgramOptions
+ get_options_from_prioritized_args(args1: vector, args2: vector, option_name: string): string
+ get_option(args: vector, option_name: string): string
+ has_option(args: vector, option_name: string): bool
+ read_options_from_runfile(filename: string): vector
SimulationBuilder
- grid: vector<vector<GridCell>>
- reservoirs: vector<Reservoir>
- transfer_proportion_constant: float
- runoff_coefficient_constant: float
- polygon_skip_constant: int
+ set_grid(grid: vector<vector<GridCell>>): void
+ set_reservoirs(reservoirs: vector<Reservoir>): void
+ set_transfer_proportion_constant(float): void
+ set_runoff_coefficient_constant(float): void
+ set_polygon_skip_constant(fint): void
+ set_depth_uniform(float): void
+ get_result(): Simulation
- calculate_average_area(): float
- apply_polygon_skip_constant(): void
- remove_reservoirs_off_grid(): void
- assign_associated_reservoirs(): vector<grid_pair>
- assign_spill_points(): void
- point_on_grid(test: pair<float, float>): bool
- foreach_cell(function: function): void
GridCell
- elevation: float
- latlon: pair<float, float>
- utm: pair<float, float>
- depth: float
- associated_reservoir: int
- runoff_coefficient: float
+ calculate_water_elevation(): float
+ add_water(depth: float): void
+ remove_water(): float
+ apply_runoff_coefficient(rc_constant: float): void
Reservoir
- name: string
- polygon: vector<pair<float, float>>
- capacity: float
- volume: float
- spill_coord: pair<float, float>
- spill_point: grid_pair
+ contains_point(lat: float, lon: float): bool
+ add_water(amt, float): float
+ dist_to_spill_coord(latlon: pair<float, float>): float
Simulation
- grid: vector<vector<GridCell>>
- reservoirs: vector<Reservoir>
- mapped_cells: vector<grid_pair>
- avg_cell_area: float
- transfer_proportion_constant: float
- runoff_coefficient_constant: float
+ run_iteration(): void
- confirm_in_bounds(row: int, col: int): bool
- get_water_transfer_proportion(row: int, col: int): vector<float>
- foreach_cell(function: function): void
- get_grid_cell(index: grid_pair): GridCell&
- get_deltapair_from_index(index: int): grid_pair
SimulationHistory
- reservoir_names: vector<string>
- reservoir_volumes: vector<vector<float>>
- water_depths: vector<vector<vector<float>>>
- rows: grid_i
- cols: grid_i
+ extract_from_simulation(sim: Simulation&)
<<abstract>>
BaseFileManager<T>
- filepath: string
+ read(): T
+ write(data: T): void
SimulationHistoryFileManager
+ read(): SimulationHistory
+ write(data: SimulationHistory): void
- write_header(): void
- write_reservoir_names(): void
- write_reservoir_volumes(): void
- write_water_depths(): void
GridFileManager
+ read(): vector<vector<GridCell>>
+ write(data: vector<vector<GridCell>>): void
- get_grid_headers(): vector<string>
- get_grid_size(): grid_pair
- read_line_into_map: unordered_map<string, string>
ReservoirFileManager
+ read(): vector<vector<GridCell>>
+ write(data: vector<vector<GridCell>>
- read_reservoir_list_file(): vector<Reservoir>
- read_capacity_into_reservoir(res: Reservoir&)
- read_polygon_into_reservoir(res: Reservoir&)

VII. Technical Design

Grid Cell Proportioning

The main method through which our simulation handles the distribution of water across land over time is through a water proportioning algorithm. In this algorithm, we consider every cell in the simulation grid and determine what proportion of water in each cell will move to each of its 8 neighbors. This process is visualized in figure 3, which shows how water moves from a center cell to these 8 neighbors, with each cell's elevation labeled in a view from above. Note that when the elevation difference between a center cell and its neighbor is large relative to the others, the arrow from the center cell to that neighbor is larger and darker. This serves to depict a larger proportion of water being diverted to this cell, since there is a steeper downhill in this direction. Likewise, when the elevation difference is relatively smaller, the arrows are smaller. For neighbors whose elevations are higher than the center cell, there is no arrow, since no proportion of water in the center cell will be sent in the uphill direction. In short, to calculate these proportions, we simply calculate the difference between a center cell and each of its neighbors (ignoring those with greater elevation than the center cell), then scale each of these differences so that each value will become a decimal proportion, representing the proportion of water in the center cell that will be moved to each neighbor.

Figure 3: Visualization of relative proportions



This method can lead to issues in situations where there is either a large quantity of water in a center cell or a small difference between the elevations of the center and neighboring cells. Consider the situation in figure 4a, which shows a side view of the situation shown in figure 3. When the water in the center cell is distributed in equal proportions to the cells on the left and right, the center cell is left with a lower total height than the combination of the elevation and depth of water in its neighbors. This situation is unrealistic, since water is essentially traveling uphill. To solve this problem, we calculate our proportions in the same way as described previously, then we solve the following equation to calculate the exact amount of water that can be dispersed by the center cell without allowing the center's water level to drop below that of its neighbors:

$$d = (e_c - e_h) / (1 + p_h)$$

Where

d = The total depth of water to disperse

e_c = The combined height of the elevation and water depth of the center cell

e_h = The combined height of the elevation and water depth of the highest neighboring cell such that $e_h < e_c$

p_h = The previously calculated proportion of water that we initially wanted to send to the cell corresponding to e_h

Once this value is calculated, we can divide it by the total amount of water in the center cell to get the proportion of water we can disperse to each neighbor combined. Then, we multiply the initial neighbor proportions by this proportion to obtain the proportion of water we can send to each neighbor. The results of this solution are summarized in figure 4b.

Figure 4a: Overproportioning demonstration

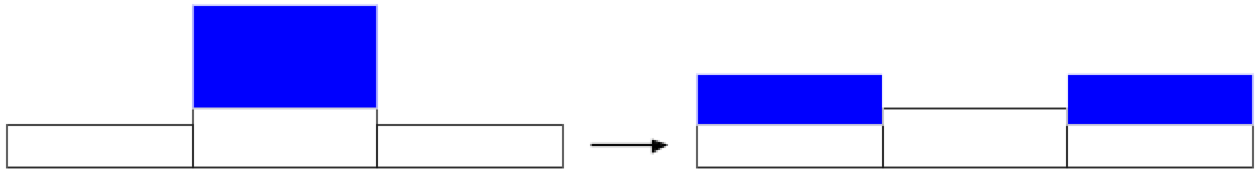
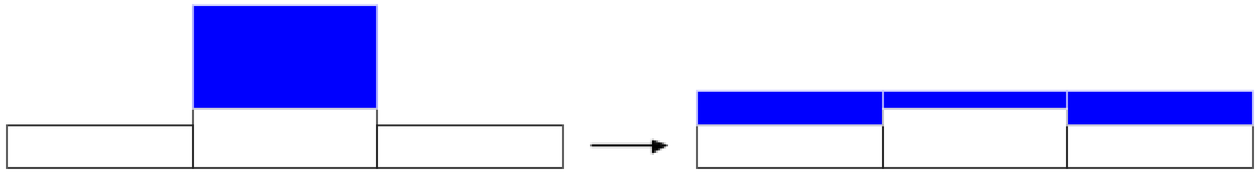


Figure 4b: Overproportioning solution



VIII. Software Test and Quality

Unit Tests

Unit tests serve two purposes: first, they ensure that individual algorithmic components of our simulation are completely free of bugs at the time of their creation. Second, they ensure that refactors and other adjustments to our program's functionality do not cause bugs to reappear. We are using Visual Studio's CppUnitTest Framework to write tests and run them in the Visual Studio IDE. All unit tests must pass for our product to be acceptable. We have not set specific code coverage metrics to avoid unnecessary tests for simple units, but we enforce a requirement for tests on any unit more complex than a getter or setter. Unit tests contribute to the quality of our final product by ensuring that the underlying algorithms our simulation relies upon are completely free of bugs and no strange behavior will be seen in our output files. Our unit tests include the following:

- **Program Options Tests:** These tests confirm that the program is correctly able to read data supplied over command line arguments or through an input file. For instance, the user needs to be able to supply the simulation with file names for the input grid data, the input reservoir data, and the output binary data. These tests confirm that our helpers functions for reading this input are working correctly.
- **File I/O Tests:** We have created sample data files to test reading and writing for all reservoir data, all water data, and all elevation data that is used in our software. These tests check that writing data to a file, then reading that data using the corresponding read operation, returns the same data as what was initially written.
- **Simulation Builder Tests:** These tests confirm that the simulation builder is correctly processing the data the user supplies to it. This is a large set of tests since much of the logic used to set up simulation data is very complex. One test confirms that the builder removes reservoirs from the list if they do not exist within the coordinate data provided as input. Another checks that GridCells are assigned to the correct associated reservoir. There are many additional tests in this category, but all relate to the initialization of the simulation.
- **Reservoir Tests:** These tests confirm all the behaviors of reservoirs. We test for the distance to a reservoir's spill coordinate from a test point, as well as the polygon containment logic to see if a latitude-longitude point is contained within a reservoir's outline polygon.
- **Simulation Tests:** These tests are more complex, since the exact behavior of the simulation object is subject to simulation parameters and may be adjusted depending on what creates the most accurate results. However, we can still test for a few key principles that should stay constant. First, we can test the relative values of water transfer proportions; we know that steep descents should result in higher proportion assignments than shallow descents, and that flat transitions or ascents should not receive any water. Additionally, we can test for water conservation, that the amount of water that starts in our simulation's system should stay constant between iterations.

Test Results: All Unit Tests are passing as of the date of our project's completion.

Functional Tests

The team has created a simple Python script to display simulation results using matplotlib to use for functional testing. Ultimately, simulation results are displayed in Walker Water's proprietary C# software to be visualized. However, this software needed adjustments in order to accommodate the output files produced by our simulation, so in the meantime, we used another way to visualize our results. Functional tests contribute to the overall quality of our product by making sure the water data our program outputs "looks right," and that our simulation results conform to the following criteria, among others:

- Water must move downhill.
- Water must generally follow the paths of drainages in the real world.
- Simulated particles must flow "like water." For example, using our visualization, we were able to identify a bug in our simulation, where water on a flat surface that should have come to a resting equilibrium state actually continued to move and create an oscillating depth pattern. The visualization tool helped us to identify and fix this unrealistic behavior.
- Reservoirs must spill their contents when full.

Test Results: Functional Tests were able to identify many bugs during the duration of our project. Most notably, we had a bug where small "ripples" in a simulated surface of water could grow into large waves, rather than settling into a flat surface. Our functional tests allowed us to visualize this behavior and find a solution, so that water in our simulation now reaches a state of equilibrium if it rests on a flat surface.

Code Reviews

Code reviews serve two purposes: first, they ensure that all team members are familiar with the codebase as changes are made. Second, they ensure that all code we write follows SOLID programming principles and meets the standards of quality outlined in the CSCI306 Clean Code Document. Every time a team member wishes to merge the changes made in a feature branch back into the main branch, at least one other team member needs to look over the changes to ensure that they are up to par. If not, the other team member should suggest changes to make the code more readable, maintainable, and clear. Code reviews contribute to the overall quality of our program by ensuring that our simulation can be easily understood by outsiders. Our code will serve as a starting point for future Walker Water employees and field session groups developing this simulation, so the code we write must be readable and it must be able to scale up to a larger program. Code reviews helped us achieve this.

Test Results: Code reviews have allowed us to ensure that all of our program is written efficiently and is easy to understand.

IX. Project Ethical Considerations

Since the software we developed has the potential to impact the decisions of thousands of farmers and other water-users in the Surface Creek Valley, we need to carefully consider the ethics of the decisions we make for this project.

The following ACM/IEEE Principles are particularly pertinent to the development of our project:

- **3.11. Ensure adequate documentation, including significant problems discovered and solutions adopted, for any project on which they work.**
  Our project was not finished by the end of this field session, and future developers will need to continue our work. We must carefully document our problem solving process so that we can provide future developers with enough context to understand our code. Failure to properly document our project may result in being forced to redo the work we have done so far, at significant cost to the client.
- **7.05. Give a fair hearing to the opinions, concerns, or complaints of a colleague.**
  Working as a team, we will need to maintain a continuous dialog between one another in order to avoid conflict. Animosity between teammates could slow down the development process, so it is preferable to resolve disagreements as soon as they arise.
- **8.02. Improve their ability to create safe, reliable, and useful quality software at reasonable cost and within a reasonable time.**
  We are all students with lots to learn, so it's important to recognize this principle because learning should be our primary focus in this project.

The following ACM/IEEE Principles are most at risk of being violated in our project:

- **1.04. Disclose to appropriate persons or authorities any actual or potential danger to the user, the public, or the environment, that they reasonably believe to be associated with software or related documents.**
  This project aims to predict how much water farmers and residents in the Surface Creek Valley will have in any given summer. Predicting less water availability than what is actually available could cause unnecessary conservation efforts and economic harm to farmers. Predicting more water availability than what is actually available would be even more disastrous, in the worst case causing severe water shortages in the region. It is our responsibility as developers to communicate these risks to users of our software, in hopes of avoiding overreliance on our software that could cause these issues to occur.
- **3.02. Ensure proper and achievable goals and objectives for any project on which they work or propose.**
  Since we have relatively little experience working on professional software projects, it will be difficult for the team to estimate the scope of work we will be able to complete during the time available, as we are not yet certain in our abilities. We will need to be overly cautious when estimating timelines for this project to not overestimate our abilities and promise more to the client than we can deliver.
- **3.04. Ensure that they are qualified for any project on which they work or propose to work by an appropriate combination of education and training, and experience.**

Given the lack of experience of our team, we can say with confidence that we are not qualified to develop a complex fluid simulation that will be used to make real-world decisions on water policy. Because of this, we must emphasize a focus on education while developing our product, so that the team can learn as much as possible in order to make the simulation as accurate as possible. Furthermore, we must communicate to the client that because of our lack of qualifications, any software we produce must be verified by experts or tested thoroughly against real world data before it is trusted.

The following Michael Davis Tests can be used to justify the ethical implications of our project:

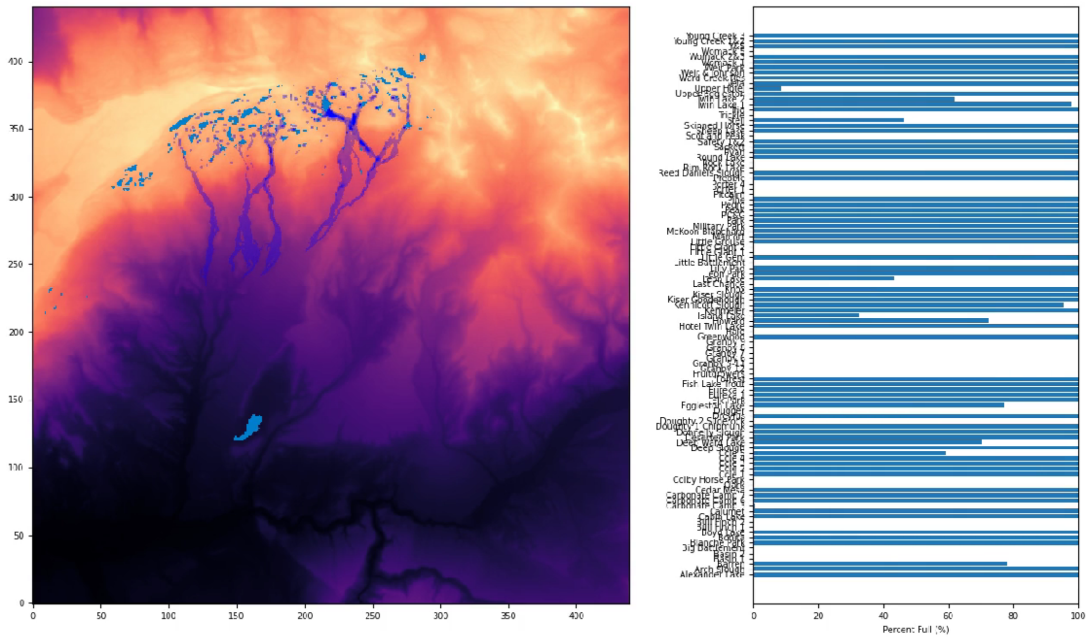
1. **The Harm Test:** We are confident that our simulation will pass the harm test, since the alternative to our simulation is for water managers to simply use rough snowfall estimates and past experience to predict droughts. Our simulation will at least be much more standardized than such rough estimates, making it useful even if it lacks some accuracy due to the complexity of the variables that need to be modeled. In short, the benefits of developing this simulation outweigh the harms.
2. **The Publicity Test:** The choice to develop this simulation would look very positive on the front page of a newspaper. It stands a chance to greatly improve the accuracy of reservoir volume predictions, which would be a positive aspect of this project in the eyes of the public.

If the ethical considerations for our project are not comprehensive enough, we could end up developing a product that promises more than it can deliver. It is vital that the team is honest with the client about our technical abilities and our lack of qualifications to develop a complex fluid simulation. While we will still be able to develop a comprehensive and potentially very accurate product despite these qualifications, it is still important to have our efforts verified by experts to ensure that our code will be as accurate to the real world as possible. Doing so will ensure that the public and environment in and around the Surface Creek Valley can stay safe.

X. Results

Figure 5 depicts the result of our simulation, a single frame in an animation of water flowing down the Grand Mesa. This is a bird's eye view of the terrain, where lighter background colors indicate higher elevations. Reservoirs are indicated by a light blue color, and water at various points in the terrain is colored darker blue, with darker colors representing higher depths of water. On the right side is a bar graph indicating the percentage of capacity of each reservoir at this point in the simulation.

Figure 5: Simulation results screenshot



Features Implemented

- Grid file and reservoir file input - Our program can read the client's data files that contain information on a land grid with elevation, SWE depths, coordinates, and RC values, and separate files that include information on reservoir names, positions, capacities, and sizes.
- Simulation file output - Our program can efficiently output a binary file containing all relevant data produced by our simulation, including reservoir volumes and grid cell depths at all simulated times. Because we are using a binary file format, this feature is very efficient, and writes data much faster than what could be written to a plain text file.
- Realistic water movement - Our program can accurately simulate the motion of water over terrain features. The paths water takes in our simulation matches the paths of drainages in the real world. We have adjustable parameters that can be used with real-world data to calibrate the timing of our simulation to be accurate to the real world.
- Reservoir volume tracking - We can keep precise measurements of how much water is in each reservoir at any simulated point in time.
- Reservoir spilling - When reservoirs reach their capacity, water spills over the reservoir's retaining structure and exits at the same point it would in the real world.
- Simple groundwater analysis - Our simulation includes basic interactions with groundwater, where we remove some water from our simulation grid in every iteration in an effort to simulate the effect of water being absorbed into the ground. This feature is implemented using terrain surface qualities estimated by a hydrologist so that the simulation will correctly reflect which land areas absorb more water than others.
- Simple visualization script - Our simulation includes a Python script which uses matplotlib to visualize the results of our simulation. The performance in this script is not very good, but it does a good job for quick tests to confirm that our simulation is working.

XI. Future Work

Performance Enhancements

Currently our simulation is single-threaded. The grid cell polygon assignment operation and the proportion calculation and assignment step that takes place in every simulation iteration could both be executed asynchronously to improve the speed of the program. Ideally, the team that implements this feature would be knowledgeable on multithreaded programs in c++, but anyone with an understanding of c++ and an ability to learn multithreading could complete this functionality. This shouldn't take a well-prepared group of programmers more than a few workdays to implement.

Improved Accuracy in Groundwater Analysis

The current groundwater simulation uses very rough approximations to remove water from the simulation grid as if it were being absorbed into the ground. Future work on this simulation could be directed towards making this aspect of the simulation more accurate. For instance, the simulation could keep track of how much water had previously been absorbed by cells and reject some amounts of future absorption if a cell reaches a water capacity limit. Programmers will need to work with hydrologists on this to ensure the simulation is accurate, but other than this requirement, programmers would need little other experience other than a solid understanding of c++. This feature could take upwards of a week to implement depending on how accurate the simulation needs to be in this respect.

Flow Analysis

A potential application of this simulation would be to measure the flow rate of streams or rivers in volume per second, since this data is useful to water managers. This would require difficult advanced calculations on the data we are already simulating to collect these numbers. Additionally, we would first need higher resolution elevation data to correctly visualize precise paths of rivers. This feature would most likely take a competent c++ programmer over a week to implement.

Higher Resolution Grid Data

Our elevation data only had points every 100 meters. Ultimately, Walker Water would like simulations to be as accurate as one point every meter. Simulating this resolution would require significant performance improvements (see above). Other than this, our simulation should be able to handle this resolution of data without much extra modification, so provided the data, a programmer with minimal experience should be able to apply this in a few days at most.

XII. Lessons Learned

Fully understand the project requirements before writing any code
Our project was difficult to fully plan because the details of hydrology and fluid dynamics are incredibly complex, and even at the end of our projects, we still fail to fully understand the details. However, we would have been better off if we had taken more time at the beginning of the project to learn these topics better and understand exactly how we would implement them in code. Specifically, we only really thought about how groundwater would play into the simulation near the end of our project, and as a result, we needed to execute a major refactor in order to fit this feature in. Had we put more thought into this feature in the beginning, we would have been able to save more time in the long run.

Documentation means more than just commenting your code
The previous field session group that produced the starting point for this project had adequate comments in their code, but it was still easy to get lost in the details of their program. It would have been much easier to get oriented in their code if we had been provided a detailed general overview of the purpose of each part of their code. For this reason, we will provide a readme with our program which outlines the algorithmic structure of our code. This way, future developers can easily orient themselves in our program before diving into the details of the code.

The simplest solution might not be the best, but it's a good place to start
There are many complexities in fluid dynamics, and starting our simulation was overwhelming as we tried to consider every variable that might impact the flow of water. However, we ultimately found that many of these complexities could be simplified to produce almost the same result. For instance, at one point in our project, we tried to dive deep into how we could accurately model the velocity of water as it moved down the hill. We were quickly overwhelmed by the scientific details of this problem, but we found that we could still control the speed of our water by simply adjusting the proportion of water moving from one cell to another during each iteration. This approach was much simpler, and it still looks very accurate.

XIII. Acknowledgements

We would like to thank the following people for helping us learn about the development cycle and advance our software engineering careers in this project:

- John Walker - Our primary point of contact at Walker Water.
- Leron Wells - A contracted developer at Walker Water, who supplied test data for us to use in our simulation and helped inform decisions on the technical side of our project.
- Trevor Hirsche - The hydrologist we communicated with to develop a preliminary iteration of groundwater analysis.
- Amelia Read - Our faculty advisor for the project who helped to facilitate group retrospective meetings and guided our project's development.

XIV. Team Profile

Luke Hart
Focus Area: Data Science
Hometown: Longmont, CO
Experience: IBM Chatbot Student Worker, ITS Consultant, Controller Operations
Hobbies: Guitar, Paddleboarding, Bowling

Nathan Howland
Focus Area: Data Science
Hometown: Durango, CO
Experience: Head of Project Management/Part Developer - Startup Company
Hobbies: Piano, Weightlifting, Golf

Chase McDonald
Focus Area: Computer Engineering
Hometown: Bend, OR
Experience: Software Engineering Intern at Tyler Technologies
Hobbies: Rock Climbing, Guitar

Cole Rincon
Focus Area: Computer Engineering
Hometown: Houston, TX
Hobbies: Hiking, Video Games

References

[1] “What is a watershed?,” NOAA’s National Ocean Service, https://oceanservice.noaa.gov/facts/watershed.html (accessed May 19, 2023).
[2] “National Weather Service Glossary,” NOAA’s National Ocean Service, https://forecast.weather.gov/glossary.php?word=catchment+area (accessed May 19, 2023).
[3] “How are UTM coordinates measured on USGS topographic maps?,” How are UTM coordinates measured on USGS topographic maps? | U.S. Geological Survey, https://www.usgs.gov/faqs/how-are-utm-coordinates-measured-usgs-topographic-maps. (accessed May 19, 2023).
[4] “Snow Water Equivalent (SWE) – Its Importance in the Northwest,” U.S. Department of Agriculture, https://www.climatehubs.usda.gov/hubs/northwest/topic/snow-water-equivalent-swe-its-importance-northwest. (accessed June 12, 2023).

Appendix A – Key Terms

Term	Definition
Catchment Area	In hydrologic terms, an area having a common outlet for its surface runoff [2]
Watershed	“A land area that channels rainfall and snowmelt to creeks, streams, and rivers, and eventually to outflow points such as reservoirs, bays, and the ocean” [1]
UTM Coordinates	Universal Transverse Mercator Coordinate System - a plane coordinate grid system named for the map projection on which it is based, which consists of 60 zones, each 6-degrees of longitude in width [3]
Runoff coefficient (RC)	A coefficient assigned to an area of land which approximates the proportion of water that will ultimately reach streams and reservoirs from that area of land.
Snow Water Equivalent (SWE)	A measurement of the depth of water contained within snow [4].