

Web-based Animated Avalanche Simulation

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Media Informatics and Visual Computing

by

Leon Stempfer

Registration Number 12222127

to the Faculty of Informatics

at the TU Wien

Advisor: Associate Prof. Dr.in techn. Manuela Waldner, MSc

Assistance: Univ.Prof. Dipl.-Ing. Dr.techn. Eduard Gröller

Vienna, September 30, 2026

Leon Stempfer

Manuela Waldner

Declaration of Authorship

Leon Stempfer

I hereby declare that I have written this thesis independently, that I have completely specified the utilized sources and resources and that I have definitely marked all parts of the work – including tables, maps and figures – which belong to other works or to the internet, literally or extracted, by referencing the source as borrowed.

I further declare that I have used generative AI tools only as an aid, and that my own intellectual and creative efforts predominate in this work. In the appendix “Overview of Generative AI Tools Used” I have listed all generative AI tools that were used in the creation of this work, and indicated where in the work they were used. If whole passages of text were used without substantial changes, I have indicated the input (prompts) I formulated and the IT application used with its product name and version number/date.

Vienna, September 30, 2026

Leon Stempfer

Acknowledgements

First and foremost I want to thank my parents for giving me the support to make my studies possible. Secondly I want to thank my friends, Karo and Livia, and my sister, Laura, for proof reading this work. Lastly I want to thank Manuela Waldner and Eduard Gröller, my supervisors, for the guidance and understanding with this project.

Abstract

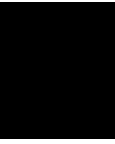
There are many tools for avalanche simulation or animation, but most are focused on academic study or used by professional government agencies and thus not easily accessible for everybody. These expert tools are complex to understand and use correctly, and need computational resources that not everybody has access to. This thesis uses weBIGeo a web-based geographical visualisation platform to make an easily accessible avalanche animation that can be used in a standard web browser.

This animation utilises Smoothed Particle Hydrodynamics and Simplified Friction-Limited Model to calculate the movement of particles to emulate the simulation path of an avalanche in real-time. The evaluation is compared against the “AvaFrame: com1DFA model”, an avalanche simulation tool used by the Austrian government, and the output closely resembles the path of the Madlein avalanche. This animation performs at 60 FPS with a GPU pass taking around 0.5ms, simulating around 15,000 active particles.

Contents

Abstract	vii
Contents	ix
1 Introduction	1
2 Related Work	3
2.1 weBIGeo – Geospatial Visualization Tool	3
2.2 Smoothed Particle Hydrodynamics	3
2.3 Real Time Simulation and Animation	3
3 Concept and Algorithms	5
3.1 Smoothed Particle Hydrodynamics (SPH)	5
3.2 Runout Model	7
3.3 Rendering	8
4 Implementation	11
4.1 Initialisation	11
4.2 Particle Movement	12
4.3 Runout Model	18
4.4 Rendering	19
5 Results	21
5.1 Particle Density Scaling	21
5.2 Comparison with State of the Art Models	23
6 Conclusion and Future Work	25
6.1 Conclusion	25
6.2 Future Work	26
Overview of Generative AI Tools Used	29
List of Figures	31
List of Tables	33

List of Algorithms	35
Bibliography	37



Introduction

Due to the chaotic nature of avalanches and the multitude of factors that need to be considered when trying to predict where and how avalanches will go off, most existing avalanche animation or simulation systems are complex to configure correctly and time intensive to run. The extent and power of these avalanches are often underestimated by Alpinists, as they cannot visualise the speed and destructive force of an avalanche. The abstract simulation models take time to configure and calculate and are not viable for most people and the easily understandable animations take a lot of computational power, which is not available to everybody.

This project tries to close the gap between easily understandable and fast, and accurate routes. It does so with a real-time avalanche animation that runs in a web browser, even on a laptop. The animation estimates the path of an avalanche without computationally expensive calculations, while still providing reasonably accurate results.

This endeavour is undertaken in the weBIGeo framework [KKCW25], a research project at TU Wien. This project aims to make spatial information, the whole of Austria, accessible through a website. It holds multiple features, like snow covering, avalanche simulation, and cloud rendering. Avalanche animation is an extension to avalanche simulation that builds upon the same groundwork.

By utilising the real-time capability of the Smoothed Particle Hydrodynamics (SPH) [MCG03] algorithm paired with the runout model Simplified Friction-Limited Model [HJRZ13], a fast and accurate animation of an avalanche is possible.

This thesis encompasses two research questions:

- How many particles are needed to get realistic avalanche trajectories in a real-time animation?

- Can Smoothed Particle Hydrodynamics be used for real-time snow avalanche animation?

The remainder of this thesis is structured as follows: Chapter 2 gives an overview of the relevant prior work in Smoothed Particle Hydrodynamics, real-time avalanche/particle animation and simulation and the weBIGeo and AlpineMaps frameworks this system builds upon. Chapter 3 will focus on the concept and algorithmic background of SPH, SFLM as well as the rendering pipeline. Chapter 4 describes the implementation of the key algorithms and rendering techniques within the weBIGeo framework. Chapter 5 presents results in the form of a case study and a comparison to the AvaFrame [TWO23] avalanche simulation tool. Finally, Chapter 6 will draw conclusions and outline future work for this project.

Related Work

This chapter focuses on weBIGeo [KKCW25], the simulation tool AvaFrame [TWOF23], the animation software “Matter” [BG25], and algorithm used in this project, Smoothed Particle Hydrodynamics [MCG03].

2.1 weBIGeo – Geospatial Visualization Tool

WeBIGeo is a research project at TU Wien, which visualises Austria in a 2.5D grid-based environment on the web [KKCW25]. WeBIGeo is based on the open source framework AlpineMaps. AlpineMaps is written in C++ and Qt to render maps in a 3D tile based environment. My thesis is an expansion of weBIGeo and uses the tile based environment, the established pipeline and features used in the weBIGeo Avalanche Simulation [KKR⁺25].

2.2 Smoothed Particle Hydrodynamics

Smoothed Particle Hydrodynamics, from now on referred to as SPH in this paper, as described by Müller et al. [MCG03] is the main algorithm used for particle movement. It is a tile based method for particle movement that is used for water animation in real-time systems and has already been applied to snow simulations [MDDM24].

2.3 Real Time Simulation and Animation

Physically accurate avalanche simulation like rapid calculations for avalanche maps [TN24], that can take hours, or the AvaFrame com1DFA [TWOF23], which takes about 20 minutes to complete, are powerful tools for the simulation of a path an avalanche takes. These simulation systems are perfect for researchers that focus on avalanche prevention

and are used by governments [TWO23] to determine forced avalanche launch sites. “Matter” is a software from Blatny, L. and Gaume, J. [BG25] that takes up to an hour to produce a realistic avalanche animation. It calculates granular movements for different scenarios and aims to deepen understanding and further research of granular particle flow. On the other side there are avalanche simulations like “weBIGeo: Interaktive Lawinensimulation im Web” [KKR⁺25], which approximates the path an avalanche will take in seconds, and works on most Laptops.

Concept and Algorithms

3.1 Smoothed Particle Hydrodynamics (SPH)

To achieve real-time execution rates, the fluid is simulated using a Lagrangian SPH model following Müller et al. (2003) [MCG03]. The fluid volume is discretized into particles carrying constant mass m_i , position $\mathbf{r}_i$, velocity $\mathbf{v}_i$, and density ρ_i . By tracking particles rather than evaluating a stationary grid, mass conservation is inherently guaranteed, allowing the continuity equation to be omitted. Furthermore, because particles move alongside the fluid, the non-linear convective term $(\mathbf{v} \cdot \nabla)\mathbf{v}$ vanishes, simplifying the governing Navier-Stokes momentum equation to Newton's second law:

$$\mathbf{a}_i = \frac{d\mathbf{v}_i}{dt} = \frac{\mathbf{f}_i}{\rho_i}$$

where $\mathbf{f}_i$ represents the total force density acting on particle i .

3.1.1 SPH Force Density Fields

The total force density field $\mathbf{f}_i$ is decomposed into pressure, viscosity, surface tension, and external forces:

$$\mathbf{f}_i = \mathbf{f}_i^{\text{pressure}} + \mathbf{f}_i^{\text{viscosity}} + \mathbf{f}_i^{\text{surface}} + \mathbf{f}_i^{\text{external}}$$

Particle density ρ_i is calculated using a radial kernel function $W(\mathbf{r}, h)$ across neighbouring particles within support radius h :

$$\rho_i = \sum_j m_j W(\mathbf{r}_i - \mathbf{r}_j, h)$$

The pressure p_i is derived using an ideal gas state equation with rest density ρ_0 and stiffness parameter k as described by Desbrun et al. [DG96]:

$$p = k(\rho - \rho_0)$$

To satisfy Newton's third law and ensure symmetric force exchange between particles, the pressure gradient force is computed using the arithmetic mean of particle pressures:

$$\mathbf{f}_i^{\text{pressure}} = - \sum_j m_j \frac{p_i + p_j}{2\rho_j} \nabla W(\mathbf{r}_i - \mathbf{r}_j, h)$$

Viscosity forces is computed to model shear friction and are symmetrized by evaluating relative velocity differences:

$$\mathbf{f}_i^{\text{viscosity}} = \mu \sum_j m_j \frac{\mathbf{v}_j - \mathbf{v}_i}{\rho_j} \nabla^2 W(\mathbf{r}_i - \mathbf{r}_j, h)$$

Surface tension is incorporated by tracking a continuous colour field $c_s(\mathbf{r}) = \sum_j \frac{m_j}{\rho_j} W(\mathbf{r} - \mathbf{r}_j, h)$. The gradient yields the surface normal $\mathbf{n} = \nabla c_s$, and the divergence of the normalized measures surface curvature $\kappa = -\frac{\nabla^2 c_s}{|\mathbf{n}|}$. The resulting surface force density is given by:

$$\mathbf{f}_i^{\text{surface}} = -\sigma \nabla^2 c_s \frac{\mathbf{n}}{|\mathbf{n}|}$$

which is only calculated when $|\mathbf{n}|$ exceeds a predefined numerical threshold.

3.1.2 Specialized Smoothing Kernels

Standard SPH kernels introduce numerical instabilities in real-time setups. To prevent particle clustering and artificial energy gain, three specialized kernels with support radius h are required:

- General Kernel (W_{poly6}): Applied for density and colour field computations. The equation uses r^2 , avoiding expensive square root operations:

$$W_{\text{poly6}}(r, h) = \frac{315}{64\pi h^9} \begin{cases} (h^2 - r^2)^3 & 0 \leq r \leq h \\ 0 & \text{otherwise} \end{cases}$$

- Pressure Kernel (W_{spiky}): Applied to pressure gradient calculations. Its non-vanishing gradient at $r \rightarrow 0$ generates strong repulsive forces, preventing particle clustering under high pressure:

$$W_{\text{spiky}}(r, h) = \frac{15}{\pi h^6} \begin{cases} (h - r)^3 & 0 \leq r \leq h \\ 0 & \text{otherwise} \end{cases}$$

- Viscosity Kernel ($W_{\text{viscosity}}$): Applied to viscosity calculations. Its Laplacian is strictly positive ($\nabla^2 W \geq 0$), preventing negative damping and stabilizing the velocity field:

$$W_{\text{viscosity}}(r, h) = \frac{15}{2\pi h^3} \begin{cases} -\frac{r^3}{2h^3} + \frac{r^2}{h^2} + \frac{h}{2r} - 1 & 0 \leq r \leq h \\ 0 & \text{otherwise} \end{cases}$$

3.1.3 Algorithmic Execution Loop

Spatial search complexity is reduced from $O(n^2)$ to $O(nm)$, where n is the number of particles and m the number of particles in the neighbouring cells, using a uniform spatial hashing grid with cell size h . CPU memory cache efficiency is optimized by storing particle objects contiguously inside grid cell structures.

- Step 1 (Spatial Hashing): Clear the uniform grid, assign particles to cells based on position $\mathbf{r}_i$, and sort particle data in memory.
- Step 2 (Density and Pressure): Query adjacent $3 \times 3 \times 3$ grid cells to compute particle density ρ_i via W_{poly6} and pressure $p_i = k(\rho_i - \rho_0)$.
- Step 3 (Force Evaluation): Evaluate $\mathbf{f}_i^{\text{pressure}}$ with ∇W_{spiky} , $\mathbf{f}_i^{\text{viscosity}}$ by using $\nabla^2 W_{\text{viscosity}}$, and $\mathbf{f}_i^{\text{surface}}$ with the W_{poly6} as long as $|\mathbf{n}_i| > \text{threshold}$ and add external body forces.
- Step 4 (Integration and Boundary Constraints): Calculate acceleration $\mathbf{a}_i = \frac{\mathbf{f}_i}{\rho_i}$, update velocity and position, and enforce the non-penetration boundary conditions by reflecting normal velocity components against solid surfaces.

3.2 Runout Model

To estimate the maximum potential runout distance of the flow process, the Simplified Friction-Limited Model (SFLM)[HJRZ13] is implemented. The SFLM is an empirical approach that applies a constant travel angle φ , frequently referred to in the literature as the angle of reach or “Fahrböschung” angle, to track a simplified energy balance along the flow trajectory. A known limitation of simple gravity-driven energy models on steep terrain is the accumulation of excessive kinetic energy, which causes unrealistic flow acceleration. To mitigate this issue, the model incorporates a maximum velocity threshold (V_{max}) to constrain simulated speeds within realistic bounds.

3.2.1 Energy Balance and Frictional Dissipation

The model calculates energy conditions iteratively on a cell-by-cell basis across the digital elevation model. A candidate cell i receives flow only if the kinetic energy remaining after the last step remains strictly positive ($E_{\text{kin}}^i > 0$). The frictional energy loss (E_f^i) incurred during motion from the active central cell to adjacent target cell j is calculated relative to the horizontal distance:

$$E_f^i = 2g \cdot \Delta x \cdot \tan \varphi$$

where g is the acceleration due to gravity, Δx is the incremental horizontal distance between the source cell and target cell j , and $\tan \varphi$ represents the energy line slope derived from the empirical travel angle.

3.2.2 Velocity Updating and Model Termination

The flow velocity V_i at target cell j is updated by converting potential energy gained through a drop in elevation (Δh) into kinetic energy, subtracting frictional losses, and enforcing the upper velocity bound:

$$V_i = \min \left(\sqrt{V_0^2 + 2g\Delta h - 2g\Delta x \tan \varphi}, V_{max} \right)$$

In this equation, V_0 represents the incoming velocity from the upstream cell, and Δh denotes the vertical drop between cell centres. Based on empirical observations of typical debris flow events, V_{max} is set to 15m/s [HJRZ13] to prevent unrealistic kinetic energy build-up. Flow propagation along a specific path ceases when total frictional dissipation balances or exceeds the sum of potential and kinetic energy ($V_i \leq 0$). When this condition is met, kinetic energy is reduced to zero, defining the terminal runout boundary along that trajectory.

3.3 Rendering

Alpha blending is the standard technique used in computer graphics to composite translucent objects over a background. The fundamental mathematics of this process rely on the Porter-Duff over operator, which determines how a foreground source colour ($\mathbf{C}_{src}$) with a given opacity (α_{src}) is combined with an existing background destination colour ($\mathbf{C}_{dst}^{old}$) [KGZ⁺12].

3.3.1 Pre-multiplied Alpha Compositing

To resolve interpolation artifacts and maintain mathematical consistency, modern rendering pipelines utilize premultiplied alpha. In this approach, the source RGB colour vector is multiplied by its own alpha value prior to the blending stage: [PD98]

$$\mathbf{C}_{pre} = \mathbf{C}_{src} \cdot \alpha_{src}$$

By associating the colour energy directly with its opacity beforehand, the compositing equation simplifies. The source colour no longer needs to be scaled by alpha during the blend operation, as its energy has already been attenuated:

$$\mathbf{C}_{dst}^{new} = \mathbf{C}_{pre} \cdot 1 + \mathbf{C}_{dst}^{old} \cdot (1 - \alpha_{src})$$

3.3.2 Hardware Blend State Formulation

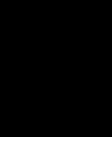
In the context of graphics hardware, the compositing equation is executed using configurable source (F_{src}) and destination (F_{dst}) blend factors:

$$\mathbf{C}_{dst}^{new} = \mathbf{C}_{source_input} \cdot F_{src} + \mathbf{C}_{dst}^{old} \cdot F_{dst}$$

Source Factor (F_{src}): 1

Destination Factor (F_{dst}): $1 - \alpha_{\text{src}}$

This mathematical equation guarantees associative compositing, meaning multiple transparent layers will add light up correctly regardless of the order in which they are linearly filtered or interpolated, eliminating dark edge artifacts.



Implementation

This system is divided into four phases: particle initialisation, two runtime phases (SPH calculations for particle movement and the runout model evaluation), and a final rendering phase.

- **Initialisation:** A modified version of the weBIGeo avalanche simulation pipeline is used to determine the initial spawning positions of particles. Rather than spawning particles in all loaded terrain tiles, this stage applies a spatial mask over a defined area, which encompasses the exact bounding region where particles are generated.
- **Runtime - SPH:** The SPH pipeline consists of multiple compute shader passes: preparation, density evaluation, and force computation. These shaders calculate the physical forces acting on the particles, thereby moving the mass of snow.
- **Runtime - Runout Model:** Executed immediately after the SPH pipeline, a dedicated compute shader evaluates terrain-dependent kinetic dissipation using the SFLM [HJRZ13]. This pass determines whether the avalanche possesses sufficient remaining energy to continue propagating across the terrain.
- **Rendering Phase:** Alpha blending is applied across the particles during rasterization, transforming discrete point elements into a visual continuous snow mass [SV14].

4.1 Initialisation

To ensure an efficient and accurate avalanche simulation, the existing particle initialisation system required modification. The baseline start-point system of the weBIGeo framework [KKCW25] selects affected terrain tiles along a linear coordinate path, loading those

tiles and initializing particles across all areas where terrain parameters match the release criteria.

While functional for basic animation, this architecture natively generates particles on surrounding slopes, valley floors, and regions outside the actual avalanche path. This drastically inflates the total particle count, introducing severe performance bottlenecks and visual artifacts from non-relevant snow particles.

To address these limitations, the line-based tile loader was replaced with a coordinate polygon mask. This polygon can be configured to cover either the influence area of the avalanche or strictly its initial release area. Both approaches have advantages and drawbacks:

- **Full Path Initialisation** spawns particles across the entire prospective influence area of the avalanche, pre-populating resting snow mass along the slope. While this could theoretically model dynamic snow accumulation, all active particles in the current implementation accelerate simultaneously under global gravity upon launch without a stationary rest state. Furthermore, this approach increases the total particle count, depending on the size of the avalanche up to a factor of ten, leading to performance degradation.
- **Release Area Initialisation (Selected Approach)** spawns particles exclusively within the upper release area of the avalanche. This method yields superior performance by significantly reducing the particle count, while avoiding structural disadvantages in flow representation due to the uniform acceleration model.

A hybrid solution, where particles are pre-allocated across the entire slope but remain stationary until impacted by the advancing avalanche front is the optimal approach (see Chapter Conclusion and Future work).

4.2 Particle Movement

The SPH algorithm is split into three shaders for this project, one prepares the data, one for the density and pressure calculations, and one for outside forces and the enforcement of boundaries.

4.2.1 Preparation Pass

The preparation compute shader constructs the spatial data structures necessary for efficient $O(N)$ neighbour searching, bypassing brute-force $O(N^2)$ pairwise distance checks across all particles by partitioning the 2D domain into a uniform grid. This pass operates via two mechanisms: initializing the spatial grid buffer and inserting particles into per-cell lock-free singly linked lists.

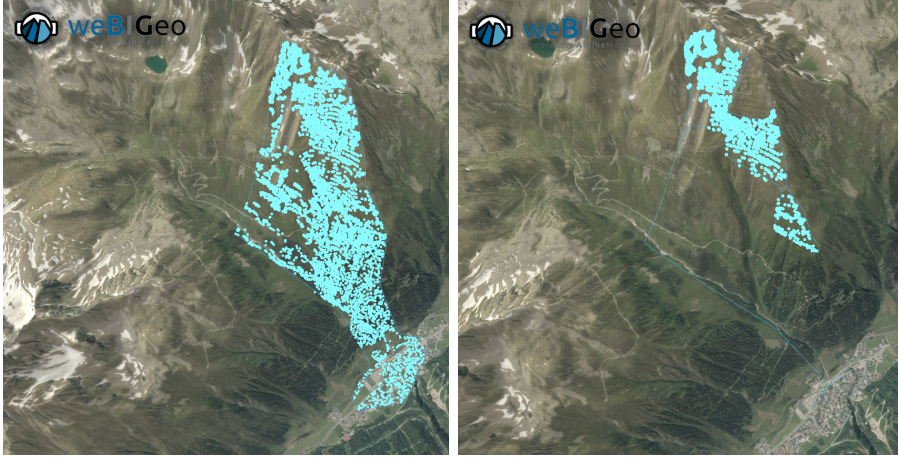


Figure 4.1: Comparison of the initialisation methods for the Madlein avalanche (Full Path Initialisation - left; Release Area Initialisation - right)

The helper function `position_to_cell_index` converts a particle’s continuous world-space position into a discrete 1D spatial cell index. Following index calculation, the thread verifies whether the particle remains within valid simulation bounds. Out-of-bounds particles are flagged for removal during the subsequent memory compaction phase.

Algorithm 4.1: Position-to-cell index.

Input: Position $\mathbf{p}_{xy}$; region minimum $\mathbf{r}_{\min}$; region size $\mathbf{r}_{\text{size}}$; output resolution (R_x, R_y)

Output: Cell index i

- 1 $\mathbf{uv} \leftarrow (\mathbf{p}_{xy} - \mathbf{r}_{\min}) / \mathbf{r}_{\text{size}};$
 - 2 $x \leftarrow \text{clamp}(\text{u32}(uv_x R_x), 0, R_x - 1);$
 - 3 $y \leftarrow \text{clamp}(\text{u32}((1 - uv_y) R_y), 0, R_y - 1);$
 - 4 $i \leftarrow y R_x + x;$
 - 5 **return** $i;$
-

To handle simultaneous GPU writing operations without lock contention or data corruption, the list insertion utilizes atomic operations on the array of cell heads. Performing an `atomicStore(&cell_heads[idx], INVALID_INDEX)` on the cell head pointer allows multiple threads to safely insert particles into the same grid cell in parallel, maintaining a thread-safe singly linked list structure across the grid.

4.2.2 Density and Pressure

The computation of fluid density and pressure in the SPH pipeline relies on local particle interactions within a finite smoothing radius h . To achieve the optimal computational complexity of $O(N)$, the SPH density shader utilizes a 2D spatial acceleration grid. By leveraging pre-constructed single-linked lists (`cell_heads` and `particle_next`),

each GPU thread restricts its density summation to particles residing within the active particle's local grid cell and its eight immediate neighbours. The shader maps continuous world-space positions to spatial grid coordinates using two helper functions `position_to_cell_coords(positionxy)` 4.1 and `cell_coords_to_index(cell)` 1.

This algorithm 1 maps continuous 2D coordinates within the simulation domain $[\mathbf{p}_{\min}, \mathbf{p}_{\min} + \mathbf{S}]$ into integer cell coordinates $(c_x, c_y) \in [0, R_x - 1] \times [0, R_y - 1]$.

Algorithm 4.2: Cell coordinates to index.

```

1
  Input: Cell coordinates  $\mathbf{c} = (c_x, c_y)$ ; output resolution  $(R_x, R_y)$ 
  Output: Cell index  $i$ 
2  $i \leftarrow \text{u32}(c_y)R_x + \text{u32}(c_x)$ ;
3 return  $i$ ;

```

The equation

$$\text{Index}(c_x, c_y) = c_y \cdot R_x + c_x$$

flattens 2D grid coordinates (c_x, c_y) into a unique 1D linear memory index.

During compute shader execution, thread i determines its home cell coordinate $(c_{x,i}, c_{y,i})$ and executes a dual nested loop over relative offsets $\Delta x, \Delta y \in \{-1, 0, 1\}$. Out-of-bounds neighbouring cells are checked and skipped prior to memory lookup:

$$c_{\text{neighbour}} = (c_{x,i} + \Delta x, c_{y,i} + \Delta y), \quad \text{where } 0 \leq c_{\text{neighbour},x} < R_x \text{ and } 0 \leq c_{\text{neighbour},y} < R_y$$

During each iteration of the list traversal, the shader evaluates the geometric relationship between the target particle i and neighbour particle j . First, the relative displacement vector $\mathbf{r} = \mathbf{x}_i - \mathbf{x}_j$ and its squared Euclidean distance are computed. They are then passed into the `sph_poly6_kernel` helper function. If the squared distance falls within the squared smoothing radius h^2 , the kernel returns the corresponding spatial weight $W_{\text{poly6}}(r^2, h)$. This value is multiplied by the uniform particle mass m (`sph_particle_mass`) and adds up directly into the local density variable ρ_i .

Once the density of particle j has been added to the density total, the thread advances to the next particle in the linked list by updating its index pointer to $j \leftarrow \text{particle_next}[j]$. The thread continues this cycle of distance evaluation, kernel weighting, and pointer advancement until the tail of the list is reached. Once the current cell's linked list is exhausted, the outer loop advances to the next cell in the 3×3 neighbourhood, ensuring that all potential interaction partners within the smoothing radius h are evaluated. Standard SPH equations can suffer from severe instability in sparsely populated regions or at the fluid's surface, where density drops drastically. The shader implements two crucial clamping mechanisms to mitigate this:

- **Density Lower Bound:** After the accumulation loop, the computed density is clamped to a minimum of 50% of the fluid's predefined rest density. This pre-

vents catastrophic underflow and avoids division-by-zero errors in subsequent force integration steps.

- **Repulsive Pressure Enforcement:** The fluid pressure is derived from the calculated density using a linearized equation of state, scaled by a stiffness parameter. Crucially, the resulting pressure is mathematically clamped using a $\max(0.0, \text{sph_pressure_stiffness} * (\text{rho} - \text{sph_rest_density}))$ function. This ensures that the fluid strictly exerts outward, repulsive forces.

Both the stabilized density and pressure values are then written to global storage buffers, making them available for the subsequent force calculation phase.

Algorithm 4.3: Density and pressure computation.

Input: Particle positions $\mathbf{p}_i$; particle count N ; cell-head array; particle linked-list array; particle mass m ; smoothing length h ; rest density ρ_0 ; pressure stiffness k

Output: Density ρ_i and pressure p_i for particle i

```

1  $i \leftarrow$  global invocation index;
2  $N \leftarrow \text{atomicLoad}(\text{output\_count})$ ;
3  $\mathbf{p}_i \leftarrow \text{positions}[i].xyz$ ;
4  $\mathbf{c}_i \leftarrow \text{positionToCellCoords}(\mathbf{p}_i.xy)$ ;
5  $W_x \leftarrow R_x, W_y \leftarrow R_y$ ;
6 for  $\Delta y \leftarrow -1$  to 1 do
7   for  $\Delta x \leftarrow -1$  to 1 do
8      $\mathbf{c}_n \leftarrow \mathbf{c}_i + (\Delta x, \Delta y)$ ;
9     if  $\mathbf{c}_n.x < 0$  or  $\mathbf{c}_n.y < 0$  or  $\mathbf{c}_n.x \geq W_x$  or  $\mathbf{c}_n.y \geq W_y$  then
10      continue;
11   end
12    $j \leftarrow \text{atomicLoad}(\text{cellHeads}[\text{cellCoordsToIndex}(\mathbf{c}_n)])$ ;
13   while  $j \neq \text{INVALID\_INDEX}$  and  $j < N$  do
14      $\mathbf{r} \leftarrow \mathbf{p}_i - \mathbf{p}_j$ ;
15      $\rho_i \leftarrow \rho_i + m W_{\text{poly6}}(\mathbf{r} \cdot \mathbf{r}, h)$ ;
16      $j \leftarrow \text{particleNext}[j]$ ;
17   end
18 end
19 end
20  $\rho_i \leftarrow \max(\rho_i, \frac{1}{2}\rho_0)$ ;
21  $\text{densities}[i] \leftarrow \rho_i$ ;
22  $\text{pressures}[i] \leftarrow \max(0, k(\rho_i - \rho_0))$ ;
```

4.2.3 Force Evaluation, Kinematic Integration, and Terrain Boundary Dynamics

The final compute pass of the SPH pipeline resolves physical interactions, evaluates governing forces, integrates particle motion over discrete time steps, and enforces terrain boundary constraints. Executed in parallel across active particles on the GPU, this pass completes the time step by translating computed density and pressure fields into updated spatial positions and velocities. Internal fluid forces consist of repulsive pressure gradient forces and momentum damping viscous forces.

- **Symmetrical Pressure Force:** The pressure gradient force $\mathbf{F}_i^{\text{pressure}}$ drives particles apart in high-density regions. To preserve linear momentum the shader uses a symmetrized pressure equations weighted by the gradient of the spiky smoothing kernel ∇W_{spiky} . The spiky kernel gradient maintains strong repulsive forces even as inter particle distances decrease, preventing particle clustering and numerical instability.

$$\mathbf{F}_i^{\text{pressure}} = - \sum_{j \neq i} m \cdot \left(\frac{P_i + P_j}{2\rho_j} \right) \nabla W_{\text{spiky}}(\mathbf{x}_i - \mathbf{x}_j, h)$$

- **Viscous Force:** Internal fluid friction and shear resistance are modelled via the viscosity force $\mathbf{F}_i^{\text{viscosity}}$. This force smooths velocity variations among neighbouring particles where μ represents the fluid viscosity coefficient (`sph_viscosity`) and $(\mathbf{v}_j - \mathbf{v}_i)$ is the relative velocity vector.

$$\mathbf{F}_i^{\text{viscosity}} = \sum_{j \neq i} \mu \cdot m \cdot \left(\frac{\mathbf{v}_j - \mathbf{v}_i}{\rho_j} \right) \nabla^2 W_{\text{visc}}(\|\mathbf{x}_i - \mathbf{x}_j\|, h)$$

- **Singularity Safeguards:** To guarantee numerical stability, candidate neighbour particles j undergo strict threshold checks prior to force evaluation. Interactions where $j = i$ are skipped. Additionally, the inter-particle distance $r_{\text{len}} = \|\mathbf{x}_i - \mathbf{x}_j\|$ must satisfy $\epsilon < r_{\text{len}} < h$, where ϵ (`sph_epsilon`) acts as a lower bound to prevent division-by-zero singularities and explosive acceleration vectors when particles overlap. Densities ρ_i and ρ_j are similarly clamped above ϵ .

Algorithm 4.4: SPH force evaluation.

Input: Particle index i ; particle count N ; particle position $\mathbf{p}_i$; particle velocity $\mathbf{v}_i$; pressure p_i ; base cell $\mathbf{c}_i$; grid dimensions (W_x, W_y) ; cell-head array; particle linked-list array; particle mass m ; smoothing length h ; viscosity coefficient μ ; epsilon ε

Output: Pressure force $\mathbf{f}_{\text{pressure}}$ and viscosity force $\mathbf{f}_{\text{viscosity}}$

```

1  for  $\Delta y \leftarrow -1$  to 1 do
2    for  $\Delta x \leftarrow -1$  to 1 do
3       $\mathbf{c}_n \leftarrow \mathbf{c}_i + (\Delta x, \Delta y)$ ;
4       $j \leftarrow \text{atomicLoad}(\text{cellHeads}[\text{cellCoordsToIndex}(\mathbf{c}_n)])$ ;
5      while  $j \neq \text{INVALID\_INDEX}$  and  $j < N$  do
6        if  $j \neq i$  then
7           $\mathbf{p}_j \leftarrow \text{positions}[j].xyz$ ;
8           $\mathbf{r} \leftarrow \mathbf{p}_i - \mathbf{p}_j$ ;
9           $r_{\text{len}} \leftarrow \text{length}(\mathbf{r})$ ;
10         if  $r_{\text{len}} < h$  and  $r_{\text{len}} > \varepsilon$  then
11            $\rho_j \leftarrow \max(\text{densities}[j], \varepsilon)$ ;
12            $p_j \leftarrow \text{pressures}[j]$ ;
13            $\mathbf{v}_j \leftarrow \text{velocities}[j].xyz$ ;
14            $\mathbf{f}_{\text{pressure}} \leftarrow \mathbf{f}_{\text{pressure}} - m \frac{p_i + p_j}{2\rho_j} W_{\text{spiky}}(\mathbf{r}, h)$ ;
15            $W_{\text{visc}} \leftarrow \nabla^2 W_{\text{visc}}(r_{\text{len}}, h)$ ;
16            $\mathbf{f}_{\text{viscosity}} \leftarrow \mathbf{f}_{\text{viscosity}} + \mu m \frac{\mathbf{v}_j - \mathbf{v}_i}{\rho_j} W_{\text{visc}}$ ;
17         end
18       end
19        $j \leftarrow \text{particleNext}[j]$ ;
20     end
21   end
22 end
23 return  $\mathbf{f}_{\text{pressure}}, \mathbf{f}_{\text{viscosity}}$ ;

```

External Forces and Explicit Kinematic Integration: Following the accumulation of internal SPH forces, external environmental forces are added to compute net acceleration. Gravity acts along the negative z -axis, scaled by particle density to yield volumetric force.

The total acceleration $\mathbf{a}_i$ is derived by dividing total accumulated forces by the particle's local density ρ_i :

$$\mathbf{a}_i = \frac{\mathbf{F}_i^{\text{pressure}} + \mathbf{F}_i^{\text{viscosity}} + \mathbf{F}_i^{\text{gravity}}}{\rho_i}$$

The particle's velocity and position are updated using a first-order Forward Euler integration scheme over the discrete time increment Δt (dt):

$$\mathbf{v}_i^{(t+\Delta t)} = \mathbf{v}_i^{(t)} + \mathbf{a}_i \cdot \Delta t$$

$$\mathbf{x}_i^{(t+\Delta t)} = \mathbf{x}_i^{(t)} + \mathbf{v}_i^{(t+\Delta t)} \cdot \Delta t$$

Algorithm 4.5: Particle velocity and position update.

Input: Pressure force $\mathbf{f}_{\text{pressure}}$; viscosity force $\mathbf{f}_{\text{viscosity}}$; particle density ρ_i ; gravity g ; particle velocity $\mathbf{v}_i$; particle position $\mathbf{p}_i$; time step Δt

Output: Updated particle velocity $\mathbf{v}_i$ and position $\mathbf{p}_i$; particle speed s_i

```

1  $\mathbf{f}_{\text{gravity}} \leftarrow (0, 0, -g\rho_i)$ ;
2  $\mathbf{a}_i \leftarrow (\mathbf{f}_{\text{pressure}} + \mathbf{f}_{\text{viscosity}} + \mathbf{f}_{\text{gravity}}) / \rho_i$ ;
3  $\mathbf{v}_i \leftarrow \mathbf{v}_i + \mathbf{a}_i \Delta t$ ;
4  $s_i \leftarrow \text{length}(\mathbf{v}_i)$ ;
5  $\mathbf{p}_i \leftarrow \mathbf{p}_i + \mathbf{v}_i \Delta t$ ;
6 return  $\mathbf{v}_i, \mathbf{p}_i, s_i$ ;

```

Domain Boundary Check and Despawning: If a particle moves outside the horizontal simulation region $[\mathbf{p}_{\min}, \mathbf{p}_{\min} + \mathbf{S}]$ or drops below z_{despawn} , it is flagged as inactive. Its position is reset to $(\mathbf{p}_{\min, x}, \mathbf{p}_{\min, y}, z_{\text{despawn}})$ and its velocity is zeroed, removing it from active computation in subsequent frames.

Terrain Tangent Velocity Projection: To prevent particles from penetrating the terrain, the surface normal $\mathbf{n}$ is sampled from `normal_texture` at normalized UV coordinates. The raw RGB texture values are unpacked from $[0, 1]$ to $[-1, 1]$ and normalized. The normal velocity component is subtracted from $\mathbf{v}_i$, constraining flow vector motion tangentially along the terrain surface:

$$\mathbf{v}_i \leftarrow \mathbf{v}_i - (\mathbf{v}_i \cdot \mathbf{n})\mathbf{n}$$

Latitude-Corrected Height Map Snapping: The particle's vertical position z_i is anchored to the digital elevation model sampled from `height_texture`. An altitude correction factor is applied based on geographic latitude $\phi = \text{y_to_lat}(y_i)$ to account for map projection distortion:

$$z_i = h_{\text{raw}} \cdot \frac{1}{\cos(\phi)}$$

Finally, the updated spatial position $(x_i, y_i, z_i, 1.0)$ and velocity $(v_{x,i}, v_{y,i}, v_{z,i}, 0.0)$ are written back to global GPU storage buffers, completing the simulation time step.

4.3 Runout Model

The compute pass executes per-particle in parallel, modifying particle velocities based on local terrain height differences and recording particle deposition into a 2D spatial grid.

Direction Fallback and Forward Step Sampling: To evaluate energy changes along the path of travel, each GPU thread determines its motion direction and samples the terrain elevation one step ahead.

- **Direction Fallback (safe_dir2):** The primary motion direction is derived from the particle's horizontal velocity `vel.xy`. To prevent division-by-zero or undefined behaviour when a particle is nearly stationary ($\|\mathbf{v}_{xy}\| \leq \epsilon$), the helper function `safe_dir2` falls back to the local downhill gradient derived from the terrain normal. If the terrain is flat, it defaults to a unit direction vector $(1.0, 0.0)$.
- **Forward Step:** The horizontal direction `dir2` is scaled by the inverse texture dimensions to compute a target coordinate `uv_next` exactly one step ahead in UV space.
- **Step Distance (dX):** Derived from `uv_step` scaled by the physical region size `settings.region_size`.
- **Elevation Drop (dH):** Evaluated by taking the difference between the current height map sample (`height_here`) and the forward sample (`height_next`).

Energy Balance and Velocity Adjustment: The core SFLM execution updates particle kinetic energy through conditional branching based on local terrain slope and frictional dissipation.

- **Energy Term Evaluation:** The target kinetic energy state `energy_term` is computed directly in WGSL, where `v0` is the scalar entry speed and `phi` is the internal friction angle.
- **Slope-Based Damping:** The thread calculates the local travel angle using `atan2( max(dH, 0.0), max(dX, eps) )`. If this angle falls below `sflm_min_travel_angle`, the velocity vector is multiplied by `sflm_damping` to simulate momentum loss on flat terrain.
- **Target Speed Rescaling:** If `energy_term ≤ 0.0`, frictional resistance has exhausted the particle's kinetic energy. Velocity is multiplied by `sflm_damping`, written to `velocities[idx]`, and the thread terminates early. If `energy_term > 0.0`, the target speed is calculated as $\min(v_{\max}, \sqrt{\text{energy_term}})$. The velocity vector `vel` is rescaled by the factor `target_speed/current_speed`. If `target_speed` falls below `sflm_stop_velocity`, velocity is set to zero to bring the particle to a complete halt.

4.4 Rendering

To visualise a continuous mass the particles need to overlap and become one. The fragment shader implements a view-dependent alpha modulation pass that outputs premultiplied alpha values directly for GPU blending.

1. Surface Facing Ratio and Center Boosting

The shader calculates how the sphere fragment faces the camera by evaluating the dot product between the normalized surface normal (`in.normal`) and the view direction `view_dir`:

- a) Facing Factor (`facing`): Clamped to $[0.0, 1.0]$, yielding 1.0 at the centre of the particle facing the camera and 0.0 along silhouette edges.
- b) Edge Softening (`center_boost`): The WGSL `smoothstep(0.4, 1.0, facing)` function applies a Hermite interpolation. Fragments with a facing ratio below 0.4 smoothly fade to fully transparent, while fragments above 1.0 reach full intensity. This eliminates hard mesh silhouettes and produces soft particle boundaries.

2. Opacity Scaling and Premultiplied Alpha Output

The final opacity alpha combines the particle's uniform alpha with the computed facing boost:

- a) Colour Tinting: The base colour is set to a light blue for the distinction to the background snow.
- b) Premultiplied Formatting: The RGB colour channels are multiplied directly by alpha prior to returning. Outputting a premultiplied `vec4f(colour * alpha, alpha)` allows the GPU pipeline's hardware blend state to correctly accumulate overlapping particle volumes without artifacts.

CHAPTER 5

Results

All measurements reported in this chapter were performed on a HP ZBook Power 15.6 inch G10.

Component	Specification
CPU	13th Gen Intel(R) Core(TM) i7-13700H 2.40 GHz
RAM	32,0 GB
GPU	NVIDIA RTX A500 Laptop GPU
OS	Windows 11
WebGPU	backend Dawn (native)

Table 5.1: Hardware and software environment specifications used for benchmark evaluations.

This chapter evaluates this avalanche animation in two perspectives. The first section consists of a case study focusing on the performance measuring the particle density and its corresponding framerate. Section 2 compares the output of this implementation to the Open Avalanche Framework [TWO23] on the trajectory of the avalanche.

5.1 Particle Density Scaling

To evaluate the efficiency and real-time viability of the avalanche pipeline within the weBIGeo framework, a performance case study was conducted across multiple particle tiers ranging from 1,569 to 297,906 active particles. The first evaluation separates isolated GPU simulation pass performances including SPH neighbour search, density/pressure evaluation, force integration, and SFLM terrain dynamics 5.1. The second one compares the particle count to the framerate of the whole application to the particle count 5.2.

Isolated Simulation Compute Performance

The execution duration of the compute pipeline demonstrates high computational efficiency.

- **Median Compute Scaling:** Across lower to mid-range workloads (1,569 to 46,248 particles), median GPU compute time remains below 0.50 ms per frame. Even at the highest load tier of 297,906 particles, the median compute dispatch consumes only 2.62 ms.
- **Tail Latency and Density Spikes:** While median execution times scale smoothly, the 75th percentile and maximum frame times show variance at higher particle counts. This skew is caused by localized particle clustering during high-density flow states, which increases thread divergence and list traversal lengths within specific spatial grid cells.
- **Theoretical Compute Throughput:** Based purely on isolated GPU simulation dispatches, the compute pipeline yields an equivalent throughput exceeding 360 FPS for up to 77,080 particles, and maintains an equivalent 76.60 FPS at 297,906 particles.

Table 5.2: GPU timing and frame-rate measurements for different particle counts.

Particles	Min.	P_{25}	Median	P_{75}	P_{95}	Max.	GPU FPS	FPS
1,569	0.13	0.13	0.17	0.19	0.98	1.50	4279.02	60.01
15,416	0.13	0.23	0.32	0.33	5.17	10.42	1206.17	60.01
28,372	0.19	0.26	0.30	0.31	6.55	18.54	1012.15	59.95
46,248	0.45	0.45	0.45	0.52	9.43	29.75	633.57	59.91
77,080	0.72	0.78	0.78	0.85	15.72	47.97	364.55	58.90
154,160	1.44	1.76	1.90	2.09	32.11	86.24	164.24	44.90
297,906	2.22	2.49	2.62	5.83	76.48	173.53	76.59	23.35

Application-Level Frame Rate (weBIGeo Integration)

When evaluating end-to-end performance, including 3D sphere instancing, depth map sampling, view-dependent alpha blending, and browser event loop synchronization, a distinct performance threshold emerges.

- **Real-Time Baseline (1,569 to 46,248 particles):** The application maintains a constant 60.0 FPS cap, demonstrating that full interactive performance is achieved without frame dropping for standard simulation sizes.
- **Transition Phase (77,080 particles):** Frame rate begins to decouple slightly from the display refresh cap, averaging 58.90 FPS.

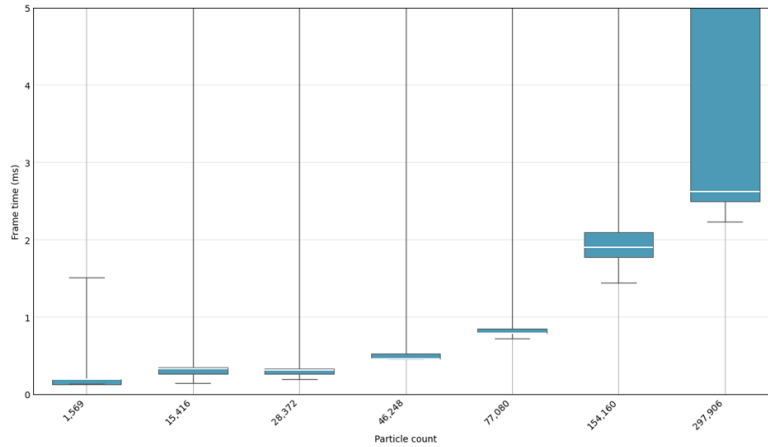


Figure 5.1: Graph showing the time it takes to calculate SPH and SFLM for an average frame compared to the amount of particles loaded.

- High-Density Performance (154,160 to 297,906 particles): Total frame rate drops to 44.90 FPS at 154.1k particles and 23.35 FPS at 297.9k particles.

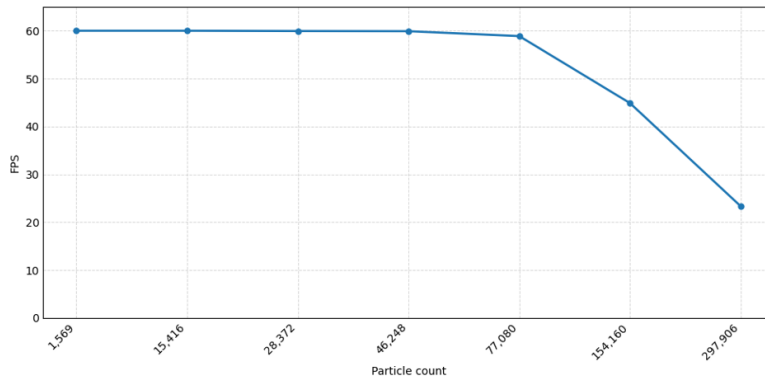


Figure 5.2: Graph showing the frames per second for different amounts of particles.

The disparity between the median GPU compute time (2.62 ms , equivalent to about 380 FPS) and the final application frame rate (23.35 FPS) at 297,906 particles indicates that the primary performance bottleneck at extreme particle counts shifts from SPH/SFLM physics computation to rasterization overhead, specifically depth-buffer sampling, instanced draw call overhead, and overdraw during volumetric alpha blending.

5.2 Comparison with State of the Art Models

To validate the physical plausibility and spatial accuracy of the particle pipeline, the simulation output was benchmarked against a reference avalanche model generated with

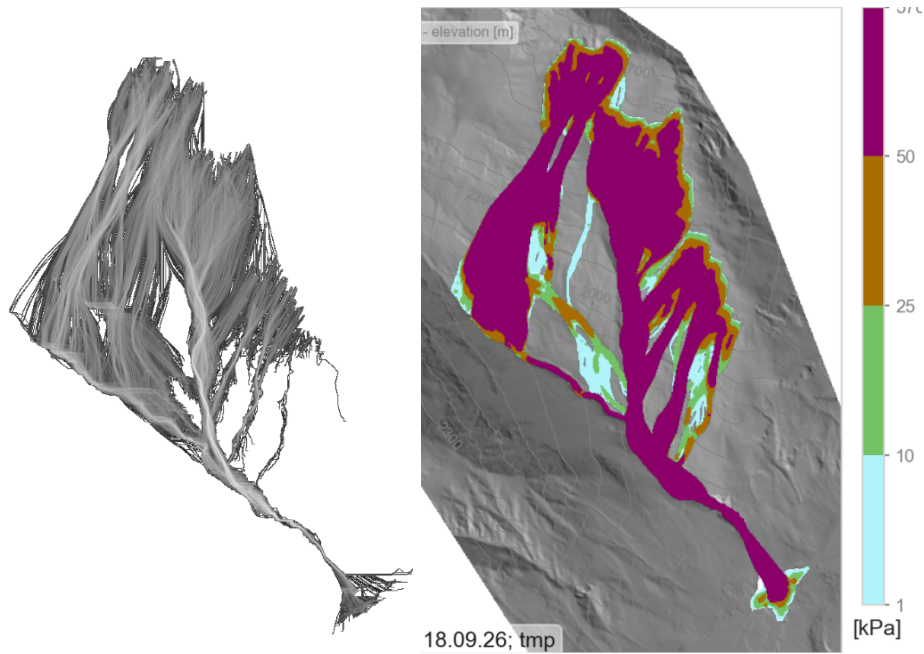


Figure 5.3: Cell count of the path of particles in the weBIGeo avalanche animation (left) compared to the path of the avalanche in the AvaFrame’s com1DFA model (right)

AvaFrame, an open-source, state-of-the-art framework for avalanche simulation developed by the Austrian Research Centre for Forests (BFW) [TWO23].

The reference dataset “com1DFA: DFA-Kernel” A qualitative spatial comparison between the particle trajectory accumulation and the AvaFrame peak pressure distribution demonstrates strong similarities across all major flow phases:

- **Release Area:** Both models initiate from identical upper release zones on the steep terrain slope. As the flow accelerates downward, the topographically enforced split into two dominant left and right flow channels around elevation ridge obstacles are present in both models.
- **Topographic Channelization:** For the most part the weBIGeo animation follows the same paths as the AvaFrame model. The major difference in the models is the size of the gap between the left and right avalanche trajectories. This could be an problem with the size of exact region of the release area or an too high amount of particles that are spawned in.
- **Runout Zone and Deposition Fan:** At the foot of the slope the flow exits the narrow chute and decelerates on flatter terrain. The weBIGeo particle system forms a distinct triangular deposition fan that matches the terminal runout boundary and spatial spread of the AvaFrame reference simulation.

Conclusion and Future Work

6.1 Conclusion

This thesis represents a pipeline of real-time avalanche animation in the weBIGeo visualisation platform. This system calculates the SPH algorithm with an SFLM runout model to calculate the movement of the particles and create a realistic avalanche movement.

An easier way to visualise an potential avalanche was the motivation behind this system and was partly successful. Using and understanding the system works without prior knowledge, but the restrictions with the predetermined release areas (see chapter Future Work) diminish the flexibility of the system.

The animation of this thesis works well and performs at 60FPS up until 50,000 particles, which should be enough for most possible avalanches, while the GPU passes described in this thesis only take about 0.5 ms per frame.

The two research questions asked in the introduction are answered as followed. Firstly the best result, within this avalanche animation, were produced with 10,000-15,000 particles, this is the amount that is spawned in the release areas, and as of the limitations of this system without the snow particles covering the whole slope.

With less particles the forces that hold together the avalanche channels are too weak and they split into many smaller streams and drift off into random directions. More particles can have the effect of “jumping over” ledges and not following the flow of the terrain and overflow up other slopes 6.1.

Secondly yes and no, SPH is a viable algorithm for real-time snow avalanche animation, but not without a runout model. The SPH parameters can be changed so it acts like snow instead of water for the acceleration and continuous flow of the avalanche, but not to stop as soon as an avalanche would, at least not for the original SPH algorithm that was designed for water [MCG03]. With the help of a runout model the problem of stopping can be solved.

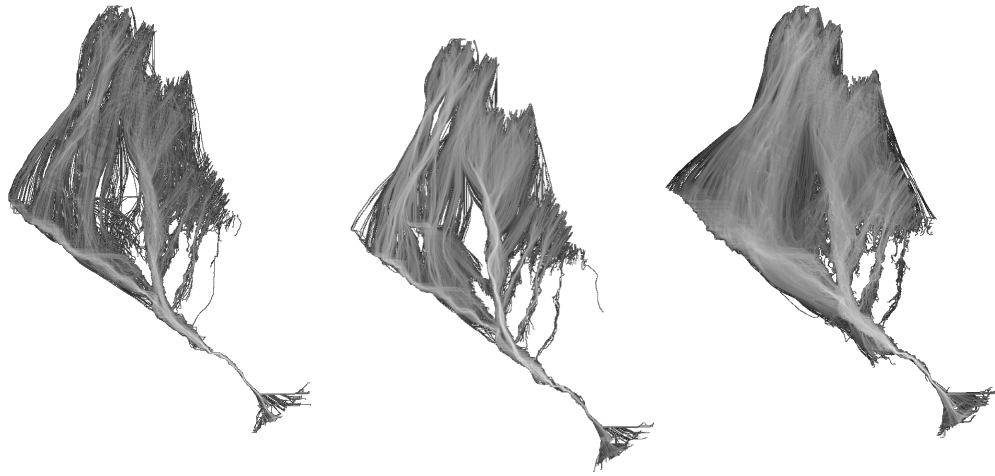


Figure 6.1: Particle cell count to low (left) sees many small channels that sometimes travel in random directions; 10,000-15,000 particles (middle) follows the terrain; Particle cell count to high (right) overflows terrain and flows up the wrong slope

All in all the real-time system delivers good results and creates a decent foundation for future expansions in real-time particle based animations.

6.2 Future Work

Even though this project worked as intended, several improvements within the development of this project came to light:

1. Release Area:

- a) Splitting the Area where particles spawn into a separate release area, where the particles would become active at the start, and a snow cover of the slope, where particles would be in a dormant state until the avalanche front would reach them, would simulate the movement and mass of a real avalanche better due to the accumulation of additional mass.
- b) Release anywhere would be a mechanism, where an area of the map could be selected or a separate interface would open for the user to draw a polygon, where an avalanche should start. This would give the project a way higher flexibility than the pre-loaded avalanche areas that exist currently.

2. **Integration with Snow Cover:** Another project in the weBIGeo framework is a realistic snow covering. The data of the amount of snow could be translated into an amount of particles that would represent this amount of snow to then start an avalanche with.

3. **Avalanche barricades:** A future expansion of this work could be placeable avalanche barricades. These could then be used for testing possible positions for the safety of critical infrastructure.
4. **Output:** The system can already produce cell count information after an avalanche animation, but this system could be expanded to include different important parameters like speed or power.
5. **Customisable Presets:** Currently, the system is customisable with a multitude of parameters, both for SPH and SFLM. Presets for differing snow levels or conditions of snow would make this system more accessible.

Overview of Generative AI Tools Used

Multiple AI tools were used in the creation of this thesis. Github CoPilot and ChatGPT 5.6 Terra, Claude Haiku 4.5 and Sonnet 4.6 were used for debugging, and implementation.

In the implementation process AI was used to shorten repeating tasks, like shader, bind group layout, and pipeline creation. In the creation of shader files, the corresponding bind group layout, and pipeline settings, a query like “Create an new shader `avalanche_particle_SPH_density`, copy the binding groups and structure of `avalanche_particle_SPH_prepare`, and create the corresponding code in `PipelineManager`” was given. This query would create a new shader file and write base structure similar to the source file. The binding groups and the pipeline functions would be created, like the matching functions from the given source.

For debugging, AI was used for understanding the C++ and WebGPU error messages and subsequently giving possible sources of the error. For this, the query was the error message.

For writing, ChatGPT 5.1 and Gemini 3.1 Pro was used for pseudo code generation and rewording and improvement of sentences and structure.

List of Figures

4.1	Comparison of the initialisation methods for the Madlein avalanche (Full Path Initialisation - left; Release Area Initialisation - right)	13
5.1	Graph showing the time it takes to calculate SPH and SFLM for an average frame compared to the amount of particles loaded.	23
5.2	Graph showing the frames per second for different amounts of particles. .	23
5.3	Cell count of the path of particles in the weBIGeo avalanche animation (left) compared to the path of the avalanche in the AvaFrame's com1DFA model (right)	24
6.1	Particle cell count to low (left) sees many small channels that sometimes travel in random directions; 10,000-15,000 particles (middle) follows the terrain; Particle cell count to high (right) overflows terrain and flows up the wrong slope	26

List of Tables

5.1	Hardware and software environment specifications used for benchmark evaluations.	21
5.2	GPU timing and frame-rate measurements for different particle counts.	22

List of Algorithms

4.1	Position-to-cell index.	13
4.2	Cell coordinates to index.	14
4.3	Density and pressure computation.	15
4.4	SPH force evaluation.	17
4.5	Particle velocity and position update.	18

Bibliography

- [BG25] L. Blatny and J. Gaume. Matter (v1): an open-source mpm solver for granular matter. *Geoscientific Model Development*, 18(22):9149–9166, 2025.
- [DG96] Mathieu Desbrun and Marie-Paule Gascuel. Smoothed particles: A new paradigm for animating highly deformable bodies. In Ronan Boulic and Gerard Hégon, editors, *Computer Animation and Simulation '96*, pages 61–76, Vienna, 1996. Springer Vienna.
- [HJRZ13] P Horton, M Jaboyedoff, al Rudaz, B et, and Markus Zimmermann. Flow-r, a model for susceptibility mapping of debris flows and other gravitational hazards at a regional scale. *Natural hazards and earth system sciences*, 13(4):869–885, 2013.
- [KGZ⁺12] Lars Kühne, Joachim Giesen, Zhiyuan Zhang, Sungsoo Ha, and Klaus Mueller. A data-driven approach to hue-preserving color-blending. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2122–2129, 2012.
- [KKCW25] Patrick Komon, Gerald Kimmersdorfer, Adam Celarek, and Manuela Waldner. Data-driven compute overlays for interactive geographic simulation and visualization. In *2025 IEEE Visualization and Visual Analytics (VIS)*, pages 186–190. IEEE, 2025.
- [KKR⁺25] Patrick Komon, Gerald Kimmersdorfer, Markus Rampp, Paula Spannring, Felix Oesterle, Jan-Thomas Fischer, Adam Celarek, and Manuela Waldner. webigeo: Interaktive lawinensimulation im web. In *Tagungsband des 6. internationalen Lawinensymposiums*, pages 150–153, 2025.
- [MCG03] Matthias Müller, David Charypar, and Markus Gross. Particle-based fluid simulation for interactive applications. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '03, page 154–159, Goslar, DEU, 2003. Eurographics Association.
- [MDDM24] Bolisetty NVS Raghavendra Mahesh, Abhiram R Devesh, Chethan Kumar DR, and Stefan Mauler. Simulation of snow-structure interaction using

- smoothed particle hydrodynamics (sph). *17th International LS-DYNA Conference 2024*, 2024.
- [PD98] Thomas Porter and Tom Duff. *Compositing digital images*, page 355–361. Association for Computing Machinery, New York, NY, USA, 1998.
- [SV14] Marco Salvi and Karthik Vaidyanathan. Multi-layer alpha blending. In *Proceedings of the 18th Meeting of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 151–158, 2014.
- [TN24] I-Chen Tsai and Takashi Nakamura. Rapid calculation for avalanche maps by GPGPU-based snow avalanche model. *Cold Regions Science and Technology*, 223:104220, July 2024.
- [TWOF23] Matthias Tonnel, Anna Wirbel, Felix Oesterle, and Jan-Thomas Fischer. AvaFrame *com1DFA* (v1.3): a thickness-integrated computational avalanche module – theory, numerics, and testing. *Geoscientific Model Development*, 16(23):7013–7035, November 2023.