Assignment of bachelor's thesis

Title:	Hydraulic erosion
Student:	Victor Kataev
Supervisor:	Dr. Ing. Bedřich Beneš, Ph.D.
Study program:	Informatics
Branch / specialization:	Web and Software Engineering, specialization Computer Graphics
Department:	Department of Software Engineering
Validity:	until the end of summer semester 2022/2023

Instructions

Hydraulic erosion is a frequently occurring, natural phenomenon. Erosion simulation is essential for realistic terrain modification in computer graphics. The aim of this paper is to create a simulation and visualization of hydraulic erosion based on the presented articles.

1) Perform a search of articles on the topic of hydraulic erosion.
2) Focus on the representation of material using a voxel grid and fluid as Smoothed particle hydrodynamics, analyze the possibilities of this combination for your visualization.
3) Design and implement a prototype simulation and visualization of hydraulic erosion.
4) Use the C++ language and parallelism.
5) Test the prototype appropriately.

**FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE**

Bachelor's thesis

Hydraulic erosion

Victor Kataev

Department of Software Engineering
Supervisor: Dr. Ing. Bedřich Beneš, Ph.D.

May 6, 2022

Acknowledgements

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on May 6, 2022   . . . .

Citation of this thesis

Abstrakt

Tato práce se zaměřuje na studium a implementaci simulace hydraulické eroze v počítačové grafice. Výsledkem této práce je plně 3D aplikace, která simuluje a vizualizuje dopady způsobené kapalinou na terén. Pro simulaci tekutiny je použit Lagrangeův přístup nazvaný Smoothed Particle Hydrodynamics(SPH). Implementace využívá vícevláknové možnosti procesoru, aby se zvýšil výkon při výpočtech simulace tekutin.

Klíčová slova   eroze, simulace tekutin, SPH, částice, řídký objem, Navier-Stokesovy rovnice, grafická aplikace, procedurální generování

Abstract

The current thesis focuses on the study and implementation of simulation of hydraulic erosion in computer graphics. The result of this work is a fully 3D application that simulates and visualizes the impact caused by fluid over terrain. For fluid simulation a Lagrangian approach named Smoothed Particle Hydrodynamics(SPH) is used. The implementation leverages the CPU's multithreading capabilities in order to boost the performance in fluid simulation calculations.

Contents

List of Figures

List of Tables

Introduction

Simulation of hydraulic erosion along with fluid simulation has been one of the hottest and actively researched topics in computer graphics over the course of the past decade. Hydraulic erosion is one of the phenomena that can be seen very often in nature. This phenomenon produces the most influential results when it comes to changes and deformations applied over terrain, which makes it a topic of particular interest in procedural generation.

The demand for such simulation may exist in the domains like engineering, scientific visualization, video game development, visual effects in film and television. While the first two i.e. engineering and scientific fields usually require a considerable degree of fidelity in order to produce practical results video games and films, on the other hand, are primarily focused on visually plausible outputs and thus tolerate the simulation to be approximated to look sufficiently realistic.

My motivation for choosing this topic is to develop and deepen my knowledge in the field of procedural generation as well as particle systems with their further application and integration in such sophisticated software systems as game engines and systems alike.

The theoretical part of this work describes the theory behind the Lagrangian approach of fluid simulation named Smoothed Particle Hydrodynamics(SPH) and the theory behind erosion and deposition of material.

As the assignment of this work states that the material is supposed to be represented as a grid of voxels. However, subsequently, in agreement with the supervisor, it was decided to give preference to constructing the terrain from a heightmap and represent the material as boundary particles on its surface. The major reason for that is coupling SPH fluid with the voxel grid ended up being inefficient. SPH is a solver for the Lagrangian approach in fluid simulation. The Lagrangian approach represents a fluid as a set of particles freely moving around in space, whereas a voxel grid is space-fixed. The result of coupling these techniques together is a significant decrease in the simulation efficiency. When a fluid particle is close to the surface it has to detect all of

the material voxels around it to wash them out. When one voxel is washed out it exposes 5 voxels underneath it. These have to be tested by the fluid particle as well in the same iteration. When this is done for a lot of fluid particles in each iteration it quickly becomes a performance bottleneck. Rather than washing the material out voxel by voxel, it would be nice to have fluid particles interact with the boundary particles only and then based on the amount of the material washed out from them adjust the depth of the terrain.

The practical part is dedicated to the description of the implementation of fluid and erosion models from the theoretical part as well as the generation of terrain from a heightmap and how this is all visualized in a 3D scene using OpenGL.

Goals

The primary goal of the current thesis is to study, how hydraulic erosion can be simulated in computer graphics field, and implement a fully 3D working prototype that will run this simulation and visualize its results.

In order to simulate the erosion itself, the simulation of fluid must be done in the first place. The fluid is then poured onto the 3D terrain, where its particles interact with the boundary particles of the terrain located on its triangles. Pouring a small number of fluid particles will not give the most tangible results, therefore the preference is given to big volumes of water. To increase the efficiency of fluid simulation, its calculation will be parallelized.

It is important to note that the goal of this thesis is not to produce a fidelitous simulation suitable for engineering or scientific employment, but rather to achieve visually satisfactory results by using the given approximated models.

Overall the goals of this work can be broken down as follows:

1. Study SPH fluid simulation model and erosion simulation model

2. Implement simulation of the fluid in parallel fashion

3. Make collisions between fluid and a terrain generated from a heightmap

4. Implement the erosion model

5. Visualize the results by using OpenGL

Analysis and Design

In this chapter, we will study first, how fluids can be simulated using Lagrangian approach and then we will study, how the erosion is simulated leveraging this approach. In Section 1.1 we will familiarize with the famous Navier-Stokes equations. In Section 1.2 we describe the Smoothed particle hydrodynamics solver and in Section 1.4 we introduce how these concepts can be used to simulate sparse volumes. In section 1.5 we will examine the erosion computational model and how this model is coupled with SPH particles to achieve the material advection and deposition.

1.1 Navier-Stokes Equations

The state of a fluid in a given time can be determined by its physical quantities. The Navier-Stokes equations (NSE) are the equations that describe viscous fluids by taking into account those quantities. Mathematically they express conservation of momentum and conservation of mass for Newtonian fluids. The classical formulation for incompressible fluids is as follows:

$$\rho(\frac{\partial}{\partial t} + \mathbf{u} \cdot \nabla)\mathbf{u} = -\nabla p + \mu\nabla \cdot (\nabla\mathbf{u}) + \mathbf{f}, \tag{1.1}$$

$$\nabla \cdot \mathbf{u} = 0, \tag{1.2}$$

where p is the pressure field, μ is the viscosity field, ρ is the density field, $\mathbf{u}$ is the velocity field and $\mathbf{f}$ is the sum of external forces acting on the fluid e.g. gravity.

Basically, this equation is the application of the Newton's second law for fluid motion. The right hand side describes all the forces that are applied on the fluid. The left part represents the product of mass and acceleration.

Equation (1.2) describes the conservation of mass for incompressible fluids.

1.2 Smoothed Particle Hydrodynamics

Smoothed particle hydrodynamics (SPH) is a computational method that originates from astrophysics. It is used in multiple research fields for simulating the mechanics of continuous systems. SPH is widely popular in simulation of meshfree Lagrangian fluid flows. We will be going through the Lagrangian approach in greater details in section 1.4. At its core SPH is an interpolation method to approximate values and derivatives of continuous field quantities by using discrete sample points. The sample points are identified as smoothed particles that carry concrete entities, e.g. mass, position, velocity, etc., but particles can also carry estimated physical field quantities dependent of the problem, e.g. mass-density, temperature, pressure, etc [1]. The SPH quantities are macroscopic and obtained as weighted averages from the adjacent particles [3].

Since it is an interpolation method any continuous field that we want to calculate can be represented as a quantity function A(r) and interpolated using integral. The integral interpolant is then defined over the space Ω by

$$A_I(\mathbf{r}) = \int_\Omega A(\mathbf{r}')W(\mathbf{r} - \mathbf{r}', h)\, d\mathbf{r}'. \tag{1.3}$$

In the above formula $\mathbf{r}$ is a point from Ω for which the quantity function A is being interpolated, W is smoothing kernel and h is the smoothing length, which controls the smoothness or roughness of the kernel. The numerical equivalent of the above integral is done by approximating it with summation:

$$A_S(\mathbf{r}) = \sum_j A_j \frac{m_j}{\rho_j} W(\mathbf{r} - \mathbf{r}_j, h), \tag{1.4}$$

where j iterates over all of the neighbouring particles, m_j is mass of particle j, ρ_j is its density and A_j is any quantity field of A at r_j that is being approximated. Equation (1.4) is the basis of the SPH method for interpolati any continuous quantities.

In NSE some fields contain first and second derivatives. In order to obtain a differentiable interpolant of (1.4) we only need to differentiate its kernel function. Thus, the first and second derivatives of (1.4) will be:

Gradient:

$$\nabla A_S(\mathbf{r}) = \sum_j A_j \frac{m_j}{\rho_j} \nabla W(\mathbf{r} - \mathbf{r}_j, h), \tag{1.5}$$

Laplacian:

$$\nabla^2 A_S(\mathbf{r}) = \sum_j A_j \frac{m_j}{\rho_j} \nabla^2 W(\mathbf{r} - \mathbf{r}_j, h), \tag{1.6}$$

1.3 Smoothing Kernels

The kernels of SPH interpolants have pivotal impact on how different quantities in NSE are calculated, which will influence the overall behavior of the fluid. Therefore, for each individual quantity field a suitable kernel function must be chosen. Because the interpolation occurs by iterating over the values of adjacent particles our kernel functions must meet a bunch of limitations.

As suggested in [4] suitable kernels require to have two properties:

$$\int_{\Omega} W(\mathbf{r}, h) \, dr = 1 \tag{1.7}$$

and

$$\lim_{h \to 0} W(\mathbf{r}, h) = \delta(\mathbf{r}), \tag{1.8}$$

where δ is Derac's delta function

$$\delta(\mathbf{r}) = \begin{cases} \infty, & \|\mathbf{r}\| = 0 \\ 0, & \text{otherwise} \end{cases} \tag{1.9}$$

The above conditions will not be enough for the SPH method additional requirements should also be fulfilled to ensure stable behavior:

the kernel must be positive

$$W(\mathbf{r}, h) \geq 0, \tag{1.10}$$

the kernel must be symmetric

$$W(\mathbf{r}, h) = W(-\mathbf{r}, h), \tag{1.11}$$

the kernel must ensure that any interactions outside of the smoothing radius h are omitted and not calculated

$$W(\mathbf{r}, h) = 0, \|r\| > h. \tag{1.12}$$
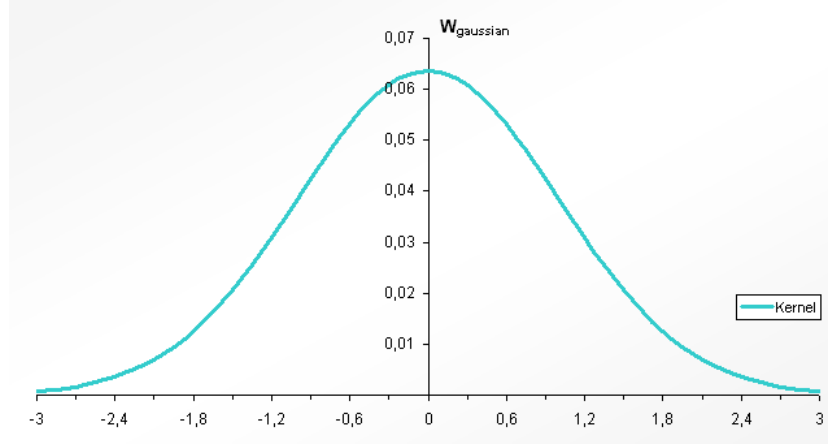
Figure 1.1: the isotropic Gaussian kernel. Taken from [1].

When there's a new interpolation of an SPH equation to be found then it is always best to assume the kernel as Gaussian [1]. The isotropic Gaussian kernel Figure 1.1 in n is given by

$$W_{gaussian}(\mathbf{r}, h) = \frac{1}{(2\pi h^2)^{\frac{3}{2}}} e^{-(\|\mathbf{r}\|^2/2h^2)}, h > 0, \tag{1.13}$$

Despite the fact that this kernel has good mathematical properties it is not constrained by (1.13), i.e. it calculates contribution of all particles rather than only the ones within the smooth radius, which makes it not the best option in practical implementation, this is why different kernels will be used. The examples of those kernels are listed below.

Standard Kernel

The standard kernel (Figure 1.2) is used for all the calculations by default unless specified differently. It has similar bell curve as the Gaussian kernel and it fulfills the constrain of (1.12). Its computation is faster, because it doesn't compute neither exponential function nor square root. The formulation of the standard kernel is

$$W_{default}(\mathbf{r}, h) = \frac{315}{64\pi h^9} \begin{cases} (h^2 - \|\mathbf{r}\|^2)^3 & 0 \le \|\mathbf{r}\| \le h \\ 0 & \|\mathbf{r}\| > h, \end{cases} \tag{1.14}$$

gradient

$$\nabla W_{default}(\mathbf{r}, h) = -\frac{945}{32\pi h^9} \mathbf{r} (h^2 - \|\mathbf{r}\|^2)^2, \tag{1.15}$$
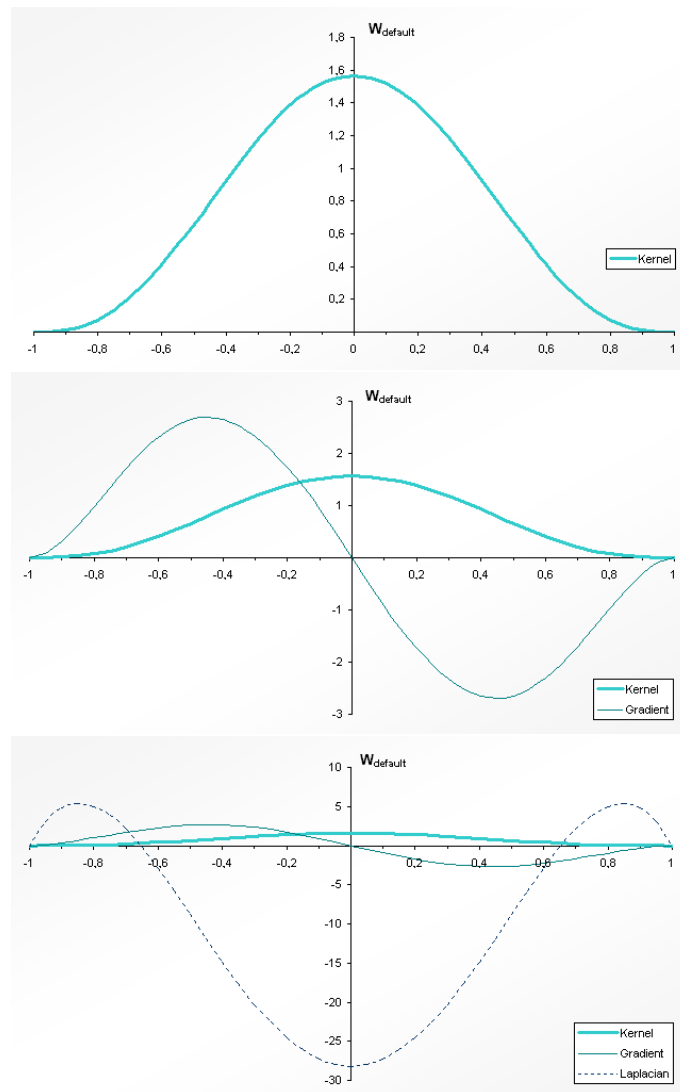
Figure 1.2: the standard kernel with gradient and Laplacian. Taken from [1].

Laplacian

$$\nabla^2 W_{default}(\mathbf{r}, h) = -\frac{945}{32\pi h^9}(h^2 - \|\mathbf{r}\|^2)(3h^2 - 7\|\mathbf{r}\|^2). \qquad (1.16)$$

Figure 1.2 depicts the standard kernel, its gradient and Laplacian for smoothing radius h = 1.

Pressure Kernel

The pressure term in NSE requires the computation of derivation. We can see in Figure 1.2 that derivation for $r \to 0$ is 0. Due to that fact particles will build clusters. This makes the standard kernel unsuitable in the evaluation of the pressure force. According to [5] a good option for the pressure term will the spiky kernel

$$W_{pressure}(\mathbf{r}, h) = \frac{15}{\pi h^6} \begin{cases} (h - \|\mathbf{r}\|)^3 & 0 \leq \mathbf{r} \leq h \\ 0 & \|\mathbf{r}\| > h, \end{cases} \tag{1.17}$$

gradient

$$\nabla W_{pressure}(\mathbf{r}, h) = -\frac{45}{\pi h^6} \frac{\mathbf{r}}{\|\mathbf{r}\|} (h - \|\mathbf{r}\|)^2,$$

$$\lim_{r \to 0^-} \nabla W_{pressure}(r, h) = \frac{45}{\pi h^6}, \lim_{r \to 0^+} \nabla W_{pressure}(r, h) = -\frac{45}{\pi h^6}, \tag{1.18}$$

Laplacian

$$\nabla^2 W_{pressure}(\mathbf{r}, h) = -\frac{90}{\pi h^6} \frac{1}{\|\mathbf{r}\|} (h - \|\mathbf{r}\|)(h - 2\|\mathbf{r}\|),$$

$$\lim_{r \to 0} \nabla^2 W_{pressure}(r, h) = -\infty \tag{1.19}$$

Figure 1.3 depicts the pressure kernel for smoothing radius h = 1.
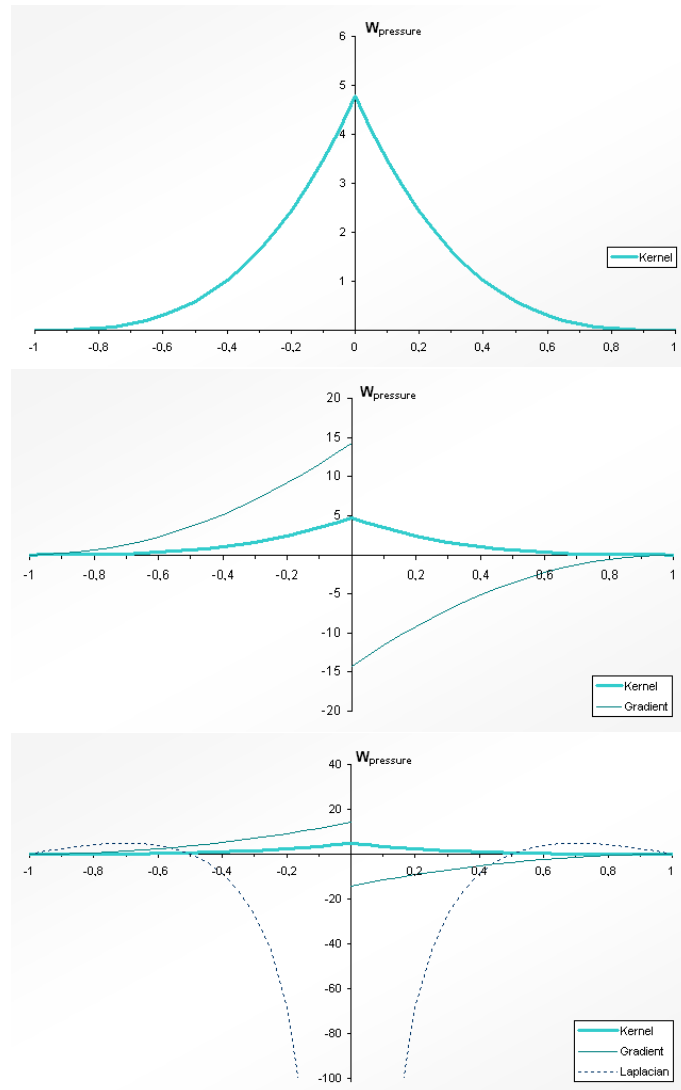
Figure 1.3: the pressure kernel with gradient and Laplacian. Taken from [1].

Viscosity Kernel

To calculate the viscosity term we need to evaluate its Laplacian, which implies that we will have to evaluate the Laplacian of its kernel. In order to maintain stability of the system the kernel has to be positive at any given time. As can be seen from the diagrams of two previous kernels they do not secure the Laplacian to be positive and thus cannot be used for the viscosity term. This is why we will need a different kernel. As proposed by [5] the kernel for viscosity will be

$$W_{viscosity}(\mathbf{r}, h) = \frac{15}{2\pi h^3} \begin{cases} -\frac{\|\mathbf{r}\|^3}{2h^3} + \frac{\|\mathbf{r}\|^2}{h^2} + \frac{h}{2\|\mathbf{r}\|} - 1 & 0 < \|\mathbf{r}\| \leq h \\ 0 & \|\mathbf{r}\| > h, \end{cases} \tag{1.20}$$
$$\lim_{r \to 0} W_{viscosity}(r, h) = \infty,$$

gradient

$$\nabla W_{viscosity}(\mathbf{r}, h) = \frac{15}{2\pi h^3} \mathbf{r}(-\frac{3\|\mathbf{r}\|}{2h^3} + \frac{2}{h^2} - \frac{h}{2\|\mathbf{r}\|^3}), \tag{1.21}$$
$$\lim_{r \to 0^-} \nabla W_{viscosity}(r, h) = +\infty, \lim_{r \to 0^+} \nabla W_{viscosity}(r, h) = -\infty,$$

Laplacian

$$\nabla^2 W_{viscosity}(\mathbf{r}, h) = \frac{45}{\pi h^6}(h - \|\mathbf{r}\|), \tag{1.22}$$

Figure 1.4 depicts the viscosity kernel for smoothing radius h = 1.
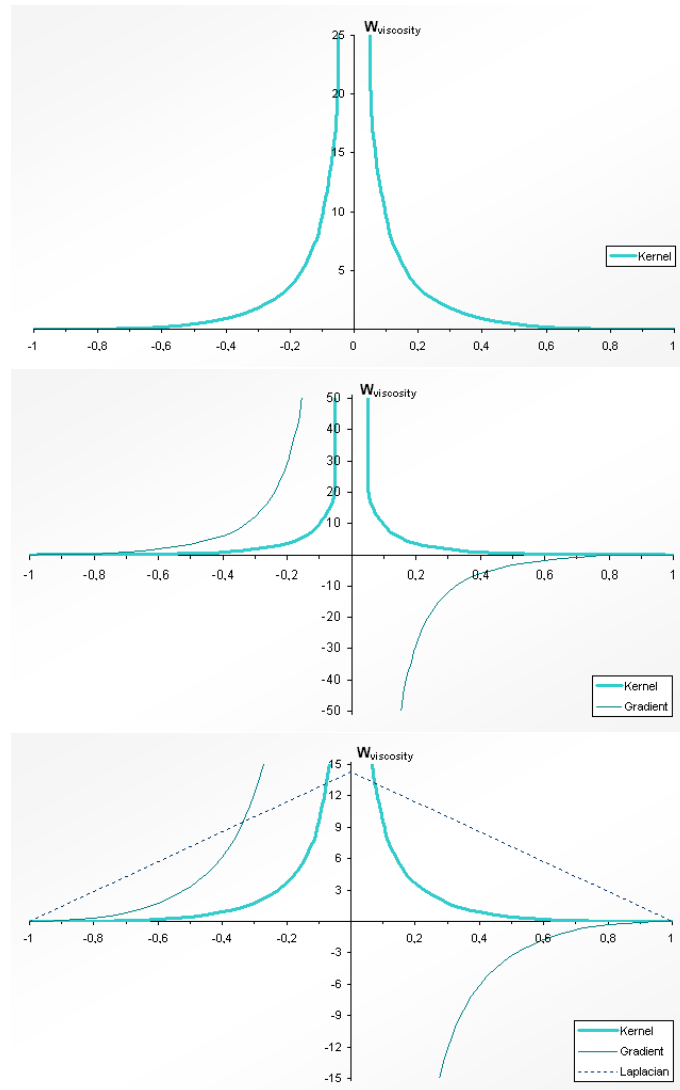
Figure 1.4: the viscosity kernel with gradient and Laplacian. Taken from [1].
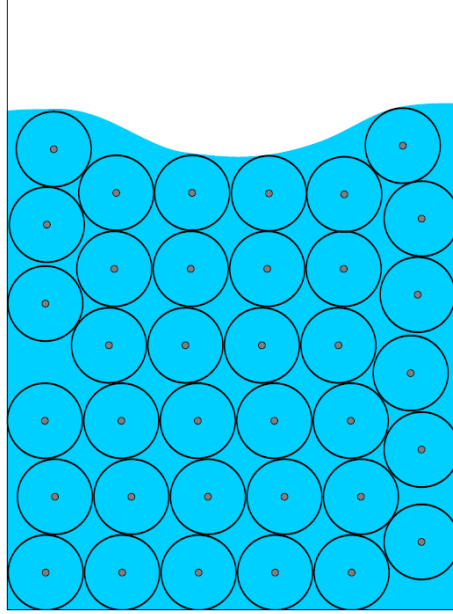
Figure 1.5: Lagrangian fluid particles. 2D projection. Taken from [2].

1.4   Lagrangian Fluid Dynamics

Compared to the classical (Eulerian) method of simulating fluids that uses 3D grid in its simulation the Lagrangian method doesn't use any grid, but uses particles instead (Figure 1.5). These particles carry physical values that are needed to represent the fluid. Each individual particle carry velocity, position, mass and smoothed quantities obtained from SPH approximation.

The NSE in the Lagrangian specification are significantly simplified. Equation (1.2) describes the mass preservation. If we assume that the number of particles is fixed and the mass of each particle is constant we can omit equation (1.2). In the Lagrangian method all particles freely move around the scene and thus it implies that any field quantity now depends on time only (t) rather than time and position (t and r) as it is in the Eulerian specification. The acceleration of a particle is then obtained by the ordinary time derivative $\frac{d}{dt}$ of its velocity $\mathbf{u}(t)$, which means that the equation (1.1) describing preservation of momentum will be simplified.

The formulation of the NSE for the Lagrangian specification has then the following formulation:

$$\rho \frac{d\mathbf{u}}{dt} = -\nabla p + \mu \nabla^2 \mathbf{u} + \mathbf{f}. \tag{1.23}$$

The right hand side describes the sum of forces acting on a particle. Where $-\nabla p$ term represents pressure, $\mu \nabla^2 \mathbf{u}$ the viscosity and $\mathbf{f}$ is the sum of all the

external forces. The sum of these three fields $\mathbf{F} = -\nabla p + \mu \nabla^2 \mathbf{u} + \mathbf{f}$ defines the acceleration of a Lagrangian particle

$$a_i = \frac{d\mathbf{u}_i}{dt} = \frac{\mathbf{F}_i}{\rho_i}, \tag{1.24}$$

where a_i is acceleration, $\mathbf{u}_i$ is velocity, ρ_i is density and $\mathbf{F}_i$ is the sum all forces of particle i.

1.4.1 Internal Forces

Density

In order to apply SPH to compute quantity fields of a particle we must know its mass and density. While mass is a user input constant density is a continuous field and has to be calculated. To compute its value for particle i we need to plug it into (1.4).

$$
\begin{aligned}
\rho_i = \rho(\mathbf{r}_i) \\
= \sum_j \rho_j \frac{m_j}{\rho_j} W(\mathbf{r}_i - \mathbf{r}_j, h) \\
= \sum_j m_j W(\mathbf{r}_i - \mathbf{r}_j, h),
\end{aligned} \tag{1.25}
$$

where W is $W_{default}$ from (1.14).

Pressure

The first term on the right hand side of (1.23) is the negative gradient of pressure. It describes the tendency of particles to move to the areas with minimum pressure

$$\mathbf{f}_i^{press} = -\nabla p. \tag{1.26}$$

By applying SPH approximation (1.5) we get

$$\mathbf{f}_i^{press} = -\sum_{i \neq j} p_j \frac{m_j}{\rho_j} \nabla W(\mathbf{r}, h), \tag{1.27}$$

Equation (1.27) requires that the pressure of each particle participating in calculation is to be known. We can derive its value from the ideal gas formula

$$pV = nRT, \tag{1.28}$$

where p is pressure, $V = \frac{1}{\rho}$ is volume, n is number of particles per one mole, R is ideal gas constant, T is temperature. For the constant mass and

15

constant temperature we can substitute the right hand side with gas stiffness constant k

$$pV = k \tag{1.29}$$

$$p\frac{1}{\rho} = k \tag{1.30}$$

$$p = k\rho \tag{1.31}$$

The above equation works well for ideal gas, however, for fluid it won't work as well. The particles will always have repulsive forces. That's the tendency of ideal gas to expand and occupy the space. In contrast, fluid must exercise cohesion and have constant density at rest. According to [6] we can use modified version of the ideal gas state equation, the final formula then will be

$$p = k(\rho - \rho_0), \tag{1.32}$$

where ρ_0 is the rest density constant.

The final thing is that the pressure force (1.27) is not symmetrical. This means that, when two particles with different pressure interact with each other the action-reaction law will not be conserved, because the pressure force will be asymmetric. We can symmetrize the pressure forces by modifying (1.27) as follows [1]

$$\mathbf{f}_i^{press} = -\rho_i \sum_{j \neq i} (\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2}) m_j \nabla W(\mathbf{r}_i, \mathbf{r}_j, h). \tag{1.33}$$

Because we want to avoid clustering of particles we cannot afford using the gradient of standard kernel here, since its value goes to zero with $\mathbf{r} \to 0$. Therefore in the above equation the special kernel for pressure $W_{pressure}$ (1.18) must be used.

Viscosity

When fluid flows its particles undergo friction. When friction happens kinetic energy transforms into heat. This friction between particles in terminology of fluid dynamics is called viscosity and is scaled by μ constant. By applying (1.6) to the second term in (1.23) we get the following SPH formulation

$$\mathbf{f}_i^{visc} = \mu \nabla^2 \mathbf{u}(\mathbf{r}_i)$$
$$= \mu \sum_{j \neq i} \mathbf{u}_j \frac{m_j}{\rho_j} \nabla^2 W(r_i - r_j, h). \tag{1.34}$$

As in case with pressure the viscosity force is also asymmetric. As suggested by [5] we can symmetrize it as follows

$$\mathbf{f}_i^{visc} = \mu \sum_{j \neq i} (\mathbf{u}_j - \mathbf{u}_i) \frac{m_j}{\rho_j} \nabla^2 W(\mathbf{r}_i - \mathbf{r}_j, h). \tag{1.35}$$

Here the Laplacian is being calculated. The Laplacian of standard and pressure kernel produce negative outputs, which is not going to be suitable for viscosity term, because it will cause instabilities. Considering that fact a special kernel for viscosity W_{visc} (1.22) must be used.

1.4.2 External Forces

The third term on the right hand side of (1.23) represents the sum of all the external forces acting of a fluid particle

$$\mathbf{f}^{ext} = \sum_n \mathbf{f}^n,$$

where n represents each individual force. Some forces may be used directly on each particle, while others will require application of SPH on adjacent particles. Below are described two examples: the gravity, that acts directly on each particle and surface tension, which value is calculated from adjacent particles.

Gravity

The gravity force is applied for each particle regardless of their location or surroundings. It does not require SPH approximation to be used. For every particle gravity force will be

$$\mathbf{f}_i^{grav} = \rho_i \mathbf{g}, \tag{1.36}$$

where $\mathbf{g}$ is the gravitational acceleration constant.

Surface Tension

The surface tension force is not included in the NSE, therefore we need to consider it as an external force. All the attractive forces inside the fluid between adjacent particles are in perfect balance with each other, however, on the surface this balance is not preserved which causes surface tensions. The traction force of surface particles, oriented inside the fluid, is called the inward surface normal. For inner particles this normal is zero it only applies for the particles that are on or near the surface. The surface tension strength depends on the curvature of the surface, the greater the curvature is the greater the

tension will be. As proposed by [5] we will use the following formulation for surface tension force

$$\mathbf{f}_i^{surf} = -\sigma \nabla^2 c_i \frac{\mathbf{n}_i}{\|\mathbf{n}_i\|}, \tag{1.37}$$

where σ is the tension coefficient that depends on the material of the fluid (e.g. water, air), c_i is the smoothed value of color field for particle i, $\mathbf{n}_i$ is the inward surface normal of particle i.

The color field term c is used to determine the surface of the fluid. Its value is $c = 1$ at particle location and $c = 0$ everywhere else [5]. In SPH formulation this term is calculated as follows

$$\begin{aligned} c_i = c(\mathbf{r}_i) \\ = \sum_j c_j \frac{m_j}{\rho_j} W(\mathbf{r}_i - \mathbf{r}_j, h) \\ = \sum_j \frac{m_j}{\rho_j} W(\mathbf{r}_i - \mathbf{r}_j, h). \end{aligned} \tag{1.38}$$

The gradient of this color field is the inward surface normal

$$\begin{aligned} \mathbf{n}_i = \nabla c(\mathbf{r}_i) \\ = \sum_j \frac{m_j}{\rho_j} \nabla W(\mathbf{r}_i - \mathbf{r}_j, h). \end{aligned} \tag{1.39}$$

This normal is not zero only near the surface. The curvature of the surface is described by

$$\kappa = \frac{-\nabla^2 c}{\|\mathbf{n}\|}. \tag{1.40}$$

The final formula for surface tension is

$$\mathbf{f}^{surf} = \sigma \kappa \mathbf{n}, \tag{1.41}$$

which, applied to each particle, will give us (1.37). When $\|\mathbf{n}\| \to 0$ numerical instabilities may arise, thus we have to compute $\mathbf{f}_i^{surf}$ only when

$$\|\mathbf{n}_i\| \geq l,$$

where $l > 0$ is some threshold relating to the particle concentration.

1.4.3 Collision

Now that we've described the physics behind fluid's behaviour we need to describe how it is supposed to behave when it interacts with a solid. The collision algorithm is basically composed from two parts: collision detection and collision response. Let's describe them individually.

Collision Detection

In order for a particle to interact with surrounding objects (in our case that will be a heightmap) we need to know particle's position **p** and its velocity **v**. The particle's position prior to collision can be obtained by subtracting from the current position its velocity vector multiplied by Δt time step. The collision model with a heightmap can be observed on Figure 1.6.
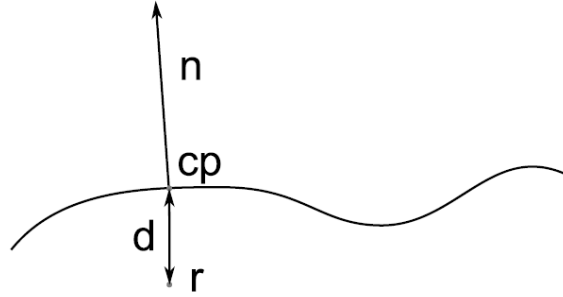
Figure 1.6: Collision with heightmap. Taken from [2].

To calculate collision response we will need to know the following data:

1. **cp** - contact point

2. d - penetration depth

3. **n** - normal vector at the contact point

As [1] put it the contact point **cp** does not necessarily have to be the intersection point on the line segment between the current position **p**(t) and the previous position **p**(t - Δt) we can use the closest point in the normal direction if it doesn't influence the result of collision too much. Before we talk about how we can use these data for calculating collision response we need to mention, how we detect collision with a heightmap fist.

A heightmap is represented as a matrix of heights. For each point its height z can be obtained from this matrix by providing its $[x, y]$ coordinates. The implementation of terrain via heightmap will be discussed in more details in subsection 2.2. For any point **r** we can thus easily find height z and normal **n** of the surface. The penetration depth d and contact point **cp** are then easy to calculate

$$d = z(\mathbf{r}) - r_z,$$

$$\mathbf{cp} = \mathbf{r} + [0, 0, d].$$

Collision Response

To simulate collision response we need to project a particle onto the surface i.e. we must assign the contact point $\mathbf{cp}$ to particle's position $\mathbf{r}$

$$\mathbf{r}_i = \mathbf{cp}$$

and adjust particle's velocity along the normal $\mathbf{n}$ direction [1]

$$\mathbf{u}_i = \mathbf{u}_i - 2(\mathbf{u}_i \cdot \mathbf{n})\mathbf{n} \tag{1.42}$$

Equation (1.42), describes a perfect elastic collision, which means that the kinetic energy will be conserved. That implies that a fluid particle will bounce off the surface, which fluids don't usually do. For that we need to control how much the kinetic energy will be conserved to achieve more realistic results. We will do this by introduction of the restitution constant [1]

$$\mathbf{u}_i = \mathbf{u}_i - (1 + c_R)(\mathbf{u}_i \cdot \mathbf{n})\mathbf{n}, \tag{1.43}$$

where $0 \leq c_R \leq 1$ is the coefficient of restitution.

However, this will not be enough, because the magnitude of c_R may be 1 and equation (1.43) will increase the kinetic energy of the particle, it will virtually behave no different from the equation (1.42). For that we need to constrain the outgoing energy to never exceed the incoming energy. We'll do this by introducing the ration of penetration depth to the distance between the last particle position and the penetrating position [1]

$$\mathbf{u}_i = \mathbf{u}_i - (1 + c_R \frac{d}{\Delta t \|\mathbf{u}_i\|})(\mathbf{u}_i \cdot \mathbf{n})\mathbf{n}, \tag{1.44}$$

where we implicitly assume that $\|\mathbf{u}_i\| > 0$.

1.4.4   Time Integration

In order to actually move a fluid particle we need to advance its position through the time constant Δt that is set globally. Equation (1.24) is evaluated to obtain the particle's acceleration and then this acceleration is employed to advance the position.

In this work the Leap-Frog integrator will be used. The major peculiarity of the Leap-Frog integrator is that the position and velocity of a particle are not updated in parallel. It uses a half time step to compute the new value of particle's velocity and then uses that value to obtain the updated position of the particle. Figure 1.7 illustrates the concept. The integration scheme yields

$$\mathbf{u}_{t+\frac{1}{2}\Delta t} = \mathbf{u}_{t-\frac{1}{2}\Delta t} + \Delta t \mathbf{a}_t, \tag{1.45}$$

$$\mathbf{r}_{t+\Delta t} = \mathbf{r}_t + \Delta t \mathbf{u}_{t+\frac{1}{2}\Delta t}. \tag{1.46}$$
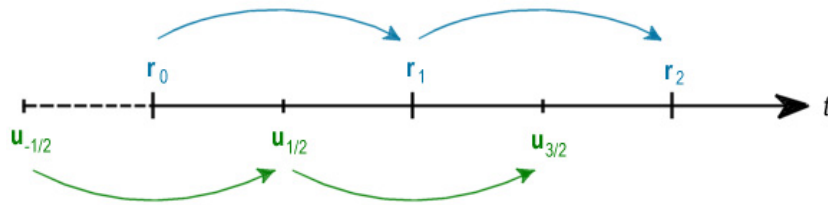
Figure 1.7: Leap-Frog scheme. Taken from [1].

with the initial velocity offset

$$\mathbf{u}_{-1/2\Delta t} = \mathbf{u}_0 - \frac{1}{2}\Delta t \mathbf{a}_0.$$
(1.47)

The velocity in time t can be obtained by following approximation

$$\mathbf{v}_t \approx \frac{\mathbf{v}_{t-\frac{1}{2}\Delta t} + \mathbf{v}_{t+\frac{1}{2}\Delta t}}{2}.$$
(1.48)

1.5 Hydraulic Erosion Model

In this section we will describe the hydraulic erosion model proposed by [3] that couples the SPH fluid simulation model, discussed in section 1.4 and the Eulerian physically based erosion model.

Simulation of hydraulic erosion has been a tough challenge in Computer Graphics. Any hydraulic erosion simulation is based on the simulation of fluid flows. The classical way is to use the Eulerian method, but its major disadvantage is the usage of 3D grid that is not easy to scale dynamically, that makes it unsuitable to simulate sparse volume. It will rapidly become memory costly, when applied for large domains. This is why in their work [3] proposed to employ SPH fluid model to simulate erosion phenomenon. Its memory efficiency enables to simulate large scale terrains, since it focuses its computations only in regions, where fluid particles physically present.

SPH particles will be used to carry material sediment and thus no additional particles are needed to represent it. The sediment will be contained directly within a SPH particle and represented as percentage of the volume it occupies. The movement of sediment is driven by both implicit and explicit advection. The implicit advection follows the flow of SPH particles, whereas the explicit is the donor-acceptor advection scheme, where sediment only advects from donor to acceptor. The donor-acceptor scheme will be discussed more in details in subsection 1.5.4.

1.5.1 Boundary Particles and External Forces
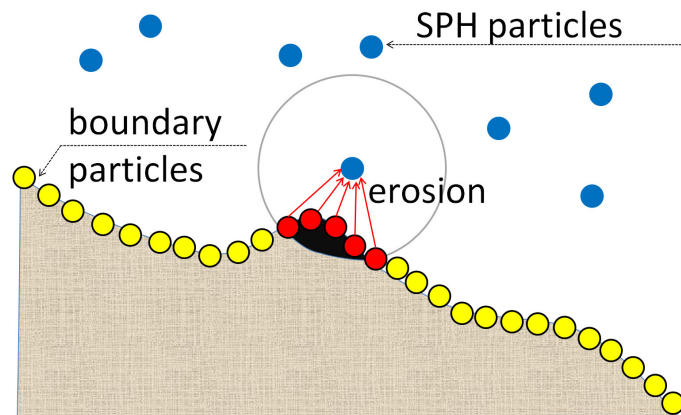
Figure 1.8: Boundary particles is the means for interacting SPH particles with the terrain. Taken from [3].

The fluid-terrain interaction will take place via boundary particles that will be seeded, with the in-between distance Δs, over the triangles of the

terrain that is represented as a heightmap Figure 1.8. This interaction will involve three things: friction, erosion and deposition.

In subsection 1.4.2 we've broken down and described the external forces used in the equation (1.23) that act upon fluid. We concluded that we would be taking into account only gravity $\mathbf{f}^{grav}$ and surface tension $\mathbf{f}^{surf}$. Now, when boundary particles come into play, we will extend the external forces term $\mathbf{f}^{ext}$ by appending the force due to solid boundary $\mathbf{f}^{bound}$. The resulting external force is going to be the sum of the following three components [3]

$$\mathbf{f}^{ext} = \mathbf{f}^{grav} + \mathbf{f}^{surf} + \mathbf{f}^{bound}. \tag{1.49}$$

In [3] the boundary force $\mathbf{f}^{bound}$ is represented as the sum of no-slip $\mathbf{f}^{ns}$ and no-penetration conditions $\mathbf{f}^{np}$ i.e.

$$\mathbf{f}^{bound} = \mathbf{f}^{ns} + \mathbf{f}^{np}. \tag{1.50}$$

The no-slip condition states that the fluid cannot penetrate the surface and the no-slip condition sets the fluid's relative speed on the boundary to zero, which represents friction. However, we already described, how fluid particles will collide with the terrain in subsection 1.4.3. That collision method will substitute the no-penetration condition. Therefore, we will drop the $\mathbf{f}^{np}$ term and the equation (1.50) will take the following form

$$\mathbf{f}^{bound} = \mathbf{f}^{ns}, \tag{1.51}$$

where we only calculate friction in a fluid-boundary interaction. The no-slip $\mathbf{f}^{ns}$ is given as [3]

$$\mathbf{f}^{ns}(\mathbf{r}) = \sum_b L_b^2 \tau^{visc}(|\mathbf{r} - \mathbf{r}_b|), \tag{1.52}$$

where b is the boundary particles, $L_b = \Delta s$ is the distance between the boundary particles and τ^{visc} is traction. The traction is expressed as [3]

$$\tau^{visc}(r) = -\mu_{bf}\mathbf{v}\nabla^2 W_{visc}(r, h), \tag{1.53}$$

where μ_{bf} is the boundary friction constant and W_{visc} is the viscosity kernel (1.22).

1.5.2 Erosion

One important thing here is that fluid flows produce shear stress on boundary particles. This is a force caused by parallel fluid forces. To apply it on boundary particles [7] proposed to give them non-Newtonian fluid characteristics via a power-law model

$$\tau = K\theta^n, \tag{1.54}$$

where τ is the shear stress, K is the shear stress constant, θ is the shear rate (a measure of shear deformation). The shear rate can be approximated as follows [3]

$$\theta = \frac{v_{rel}}{l}, \tag{1.55}$$

where v_{rel} is the velocity of the fluid relative to solid surface and l is the distance over which the shear rate is applied.

Next we need to the erosion rate. The erosion rate is related with shear stress. It was formulated by [8] as follows

$$\varepsilon = K_\varepsilon(\tau - \tau_c), \tag{1.56}$$

where K_ε is the erosion strength and τ_c is the critical shear stress (material erosion resistance). The change of mass of particle b by the SPH particle j within the distance of smooth radius h is [3]

$$\frac{dM_b}{dt} = -\sum_j L_b^2 \varepsilon(j). \tag{1.57}$$

1.5.3   Sediment Transportation

The process of sediment transportation is described by the general equation [3]

$$C(\mathbf{x}, t) = P(C) + J(C, \mathbf{x}, t), \tag{1.58}$$

where C is percentage of local volume of SPH particle occupied by sediment particles, P represents the physical redistribution processes and J is the sources and sinks that reflect erosion and deposition. In order to apply equation (1.58) in SPH we need to use the diffusion equation for SPH proposed by [4]

$$\frac{dC}{dt} = \frac{1}{\rho}\nabla(D\nabla C) + J, \tag{1.59}$$

where $\frac{dC}{dt}$ is the total derivative, denoting the time rate of change following particles at velocity $\mathbf{u}$ and D is the molecular diffusivity. In our simulation we use $\mathbf{u}$ to describe the total velocity of a sediment particle, which is composed from the sum of fluid velocity $\mathbf{v}$ and settling velocity $\mathbf{v}_s$ [3].

$$\mathbf{u} = \mathbf{v} + \mathbf{v}_s. \tag{1.60}$$

In section 8.3 we addressed that the NSE for the Lagrangian fluid dynamics are simplified, due to the fact that this approach describes fluid flows as a set of particles freely moving around in the space and that any quantity field depends on time only rather than time and position. As a result on the left-hand side we changed the acceleration term by substituting $\frac{\partial}{\partial t} + \mathbf{u}\cdot\nabla$ with total

time derivative $\frac{d}{dt}$. In our simulation we use the implicit sediment particles i.e. sediment particles follow SPH particles and contained within them. In order to switch from the framework of explicit particles into implicit we first switch into a space-fixed Eulerian frame of reference. We do so by substituting the total derivative in (1.59) with $\frac{\partial}{\partial t} + \mathbf{u} \cdot \nabla$ and plugin (1.60) into the velocity term, which yields [3]

$$\frac{\partial C}{\partial t} + (\mathbf{v} + \mathbf{v}_s) \cdot \nabla C = \frac{1}{\rho}\nabla(D\nabla C) + J. \qquad (1.61)$$

Opening the brackets yields

$$\frac{\partial C}{\partial t} + \mathbf{v} \cdot \nabla C + \mathbf{v}_s \cdot \nabla C = \frac{1}{\rho}\nabla(D\nabla C) + J. \qquad (1.62)$$

By replacing the term $\frac{\partial C}{\partial t} + \mathbf{v} \cdot \nabla C$ back by the total derivative $\frac{\partial C}{\partial t}$ we'll get the advection-diffusion equation for sediment tranport in SPH

$$\frac{dC}{dt} = -\mathbf{v}_s \cdot \nabla C + \frac{1}{\rho}\nabla(D\nabla C) + J. \qquad (1.63)$$

In the following section we describe the numerical implementation of the advection term $-\mathbf{v}_s \cdot \nabla C$ and the diffusion term $\frac{1}{\rho}\nabla(D\nabla C)$ in the SPH context.

1.5.4   Advection (donor-acceptor scheme)
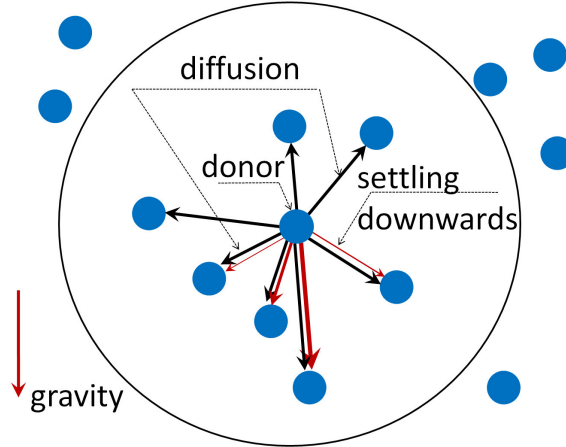
Figure 1.9: The donor-acceptor scheme. Donor particles pass the sediment to acceptor particles, whose relative position is with respect to direction of the settling velocity. Taken from [3].

The term $-\mathbf{v}_s \cdot \nabla C$, from equation (1.63), represents advection of sediment between SPH particles in the direction of the settling velocity. To implement

it [3] introduced the donor-acceptor scheme that describes interaction between particles. A SPH particles i thus is either a donor or an acceptor in every $i-j$ particle interaction. It is being an acceptor, when its relative position with respect to the donor particle corresponds the direction of the settling velocity vector Figure 1.9. For SPH the advection term is expressed as

$$-\mathbf{v}_s \cdot \nabla C = -\sum_j \begin{cases} m_j \frac{C_j}{\rho_j}(\mathbf{v_s} \cdot \hat{\mathbf{r}}_{ij})F(|\mathbf{r}_{ij}|, h), & \mathbf{v_s} \cdot \mathbf{r}_{ij} \geq 0 \\ m_i \frac{C_i}{\rho_i}(\mathbf{v_s} \cdot \hat{\mathbf{r}}_{ij})F(|\mathbf{r}_{ij}|, h), & \mathbf{v_s} \cdot \mathbf{r}_{ij} < 0, \end{cases} \quad (1.64)$$

where

$$\nabla W(\mathbf{r}_{ij}, h) = \hat{\mathbf{r}}_{ij}F(|\mathbf{r}_{ij}|, h),$$

$$\mathbf{r}_{ij} = \mathbf{r}_i - \mathbf{r}_j,$$

$$\hat{\mathbf{r}}_{ij} = \mathbf{r}_{ij}/|\mathbf{r}_{ij}|$$

and F is the derivative of the cubic spline kernel W taken with respect to $|\mathbf{r}_{ij}|$. Particle i acts as the acceptor, when $\mathbf{v}_s \cdot \mathbf{r}_{ij} \geq 0$ and as a donor otherwise. The hindered settling velocity $\mathbf{v}_s$ is formulated as

$$\mathbf{v}_s = \frac{2}{9}r_s^2 \frac{\rho_s - \rho_f}{\mu}\mathbf{g}f(C), \quad (1.65)$$

where ρ_s, ρ_f are the sediment and fluid densities, $\mathbf{g}$ is the gravity acceleration constant r_s is the radius of a sediment particle, μ is the viscosity of the fluid and C is the solid volume fraction at the acceptor particle. The function $f(C)$ is the hindering settling function approximating decreasing advection rate with higher sediment concentration. It is approximated using the Richardson-Zaki relation [9]

$$f(C) = 1 - (C/C_{max})^e, \quad (1.66)$$

where C_{max} is the maximum solid volume fraction in a fluid particle and e is an exponent $4 < e < 5.5$. For the case of $C > C_{max}$ we set $f(C) = 0$. When $C > C_{max}$ the saturated particle do not receive the sediment. Figure 1.10 depicts the transportation of the material along the settling velocity vector.

1.5.5 Diffusion

Diffusion is the tendency of the material to flow from a high concentration area to low concentration area Figure 1.11. As suggested by [4] the diffusion can be expressed as

$$\frac{dC}{dt} = \frac{1}{\rho}(D\nabla C), \quad (1.67)$$
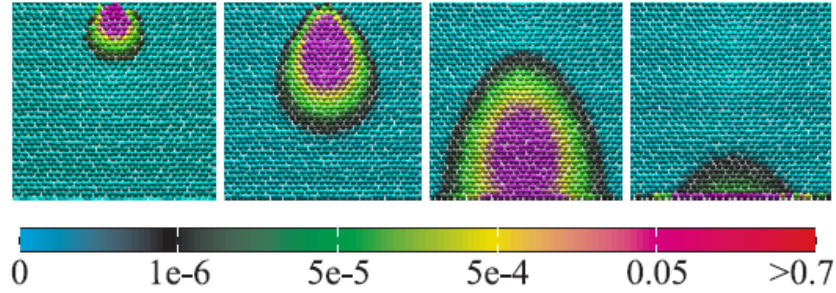
Figure 1.10: Material advection. The sediment advects with respect to direction of the settling velocity that corresponds with gravity. Taken from [3].
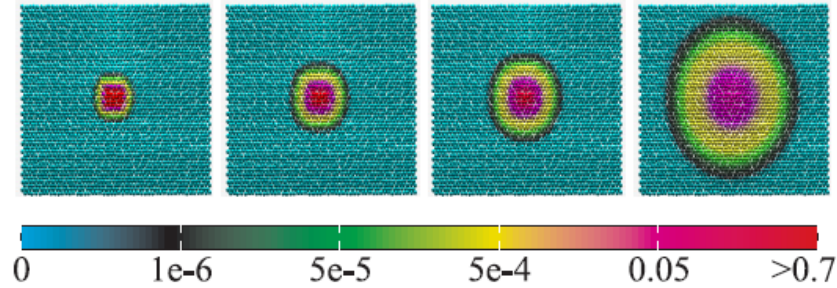
Figure 1.11: Material diffusion. Blue represents low concentrated areas while red is highly concentrated. Taken from [3].

where D is the coefficient of diffusion. The diffusion equation for SPH will be [3]

$$\frac{dC_i}{dt} = \sum_j \frac{m_j}{\rho_i \rho_j} D(C_i - C_j) F(|\mathbf{r}_i - \mathbf{r}_j|, h), \tag{1.68}$$

where F is the gradient function of a cubic spline kernel. This equation shows a positive rate of change for values of $C_i < C_j$.

1.5.6 Deposition

Deposition is the process of moving the material contained within a fluid particle onto a boundary particle once the fluid particle gets close to the ground. This process is calculated in two steps: the SPH particles communicate with boundary particles and boundary particles communicate with the terrain. Figure 1.12 depicts this process. The change of mass of a boundary particle b due

27
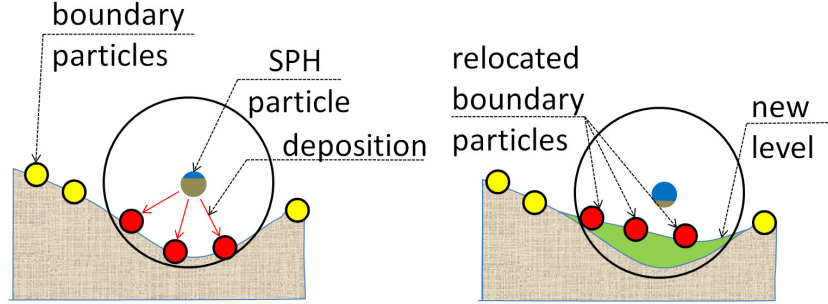
Figure 1.12: Material deposition. The sediments flows from SPH particle to boundary particles (left). The terrain height is changed and new boundary particles generated (right). Taken from [3].

to exchange of C from fluid particle j is [3]

$$\frac{dM_b}{dt} = \sum_j \rho_s \frac{m_j}{\rho_j} \frac{dC(j)}{dt} \tag{1.69}$$

where $dC(j)/dt$ is the advection term from (1.64) with fluid particles strictly set as donors and boundary particles as acceptors. For the case of $(\mathbf{v}_s \cdot \mathbf{r}_{ij}) < 0$, there is no material deposition.

1.5.7 Terrain Modification

SPH fluid particles interact with the terrain via boundary particles that are located on the triangles of the terrain. The terrain is represented as a regularly sampled heightmap. The change of mass of a triangle p is related with the change of mass of all its boundary particles and is expressed as follows [3]

$$\frac{dM_p}{dt} = \sum_b \frac{dM_b}{dt}. \tag{1.70}$$

Because we use a regular heightmap the deformation of the terrain corresponds to changing the height of the triangle's vertices. The equation for computing the total height change H of a triangle is [3]

$$H = \frac{3}{6} \frac{m}{\rho_s} \frac{1}{A_b}, \tag{1.71}$$

where m is the triangle's mass, the term $(1/6)$ reflects the fact that changing one vertex results in a change of six attached triangles to an inner vertex on the heightmap, the term A_b is the area of the vertically projected triangle. For the deposition case i.e. $H > 0$ we first distribute H to the lowest triangle vertices, when their height are even we then distribute H uniformly. The same

principle applies for the erosion case ($H < 0$). We subtract the height from the heightest vertices first and once they're all even we subtract uniformly.

Implementation

The previous section 1 describes the theoretical part of fluid and erosion model. In this section we talk about how we can put this theory in a practical use. The entire simulation is run on CPU using OpenMP API to boost the calculation by leveraging multithreading. The logic of the simulation is illustrated on Figure 2.1. We will talk in details about each step down below in this chapter.
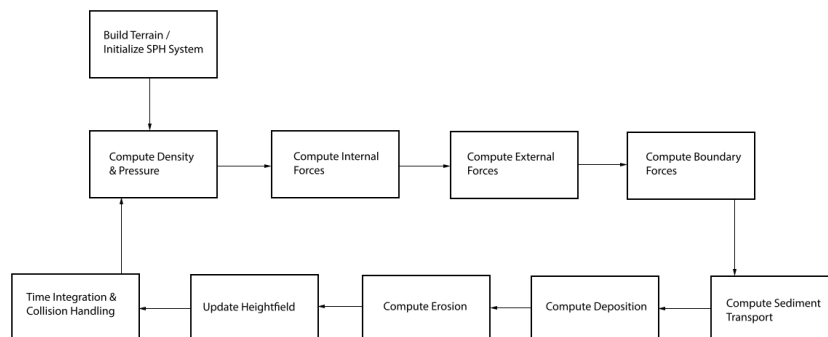
Figure 2.1: The simulation algorithm scheme.

Our simulation uses a lot of particles (fluid + boundary). We need to know the adjacent particles of each SPH particle to calculate its physical quantities and to pass the carrying sediment between them. And also, we need to know which boundary particles are in range of each fluid particle in order to erode/deposit material from/onto them. That implies the involvement of lots of iterations until we find the particles we need. This part of simulation can very quickly lead to performance issues and significantly increase the simulation time. Luckily, it can be improved algorithmically.
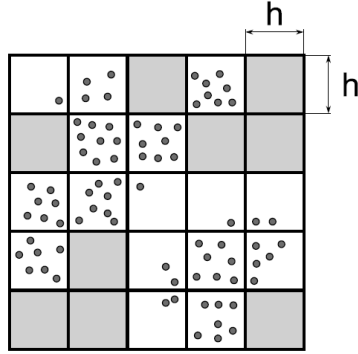
Figure 2.2: Spatial Hashing data structure to accelerate particles' neighbours search. The cell size h is the smooth radius of SPH particles. Taken from [2].

2.1 Fast Nearest Neighbor Search

Iterating over all particles in the scene and checking if a given particle is adjacent to the current particle will have $O(n^2)$ time complexity. This complexity can be reduced down to $O(mn)$ by employing the *spatial hashing* method.

2.1.1 Spatial Hashing

The spatial hashing method is an algorithm that enables us to efficiently find nearest neighbors Figure 2.2. Basically, it is a way of using a hash table to subdivide 3D space. Each particle position goes through a hash function, which maps a 3D position to 1D by generating a hash key for each grid cell. The hash function proposed by [10] is as follows

$$hash(\hat{\mathbf{r}}) = (\hat{\mathbf{r}}_x p_1 \ xor \ \hat{\mathbf{r}}_y p_2 \ xor \ \hat{\mathbf{r}}_z p_3) \ mod \ n_H, \tag{2.1}$$

where n_H is the size of the table and

$$\hat{\mathbf{r}}(\mathbf{r}) = (\lfloor \mathbf{r}_x/l \rfloor, \lfloor \mathbf{r}_y/l \rfloor, \lfloor \mathbf{r}_z/l \rfloor)^T \tag{2.2}$$

is the position of a point in 3D space, with cell size l. The three unknowns in (2.1) are

$$p_1 = 73\ 856\ 094$$

$$p_2 = 19\ 349\ 663$$

$$p_3 = 83\ 492\ 791$$

According to analyses from [10] the most efficient table size n_H is

$$n_H = prime(2n), \qquad (2.3)$$

where $prime(x)$ is the function that returns a prime number which is $\geq x$ and n is the amount of particles in the simulation. For the cell size we use

$$l = h, \qquad (2.4)$$

which is the most optimal, since we want to only include particles that are within the smoothing radius i.e. $\|\mathbf{r}\| > h$.

2.1.2 Spatial Queries

Before being able to quickly access adjacent particles we need to build up the hash table itself. We can do this by iterating twice through the whole set of particles.

The first pass creates a hash key for each particle and puts it in appropriate position in the hash table

$$hash\_table[hash(\hat{\mathbf{r}}(\mathbf{r}_i))] = Particle_i, \qquad (2.5)$$

where $\mathbf{r}_i$ is the position of particle i. Each hash_table value has to be a dynamic list, where we will store the adjacent particles with particle i.

The second pass performs the actual particle queries. For a given particle position $\mathbf{r}_Q$ we calculate the bounding box around it as follows

$$BB_{min} = \hat{\mathbf{r}}(\mathbf{r}_Q - (h, h, h)^T), \quad BB_{max} = \hat{\mathbf{r}}(\mathbf{r}_Q + (h, h, h)^T). \qquad (2.6)$$

Then we iterate from BB_{min} to BB_{max} in each three axis, creating unique discrete positions pos_D and retrieve a dynamic list L for each such position

$$L = hash\_table[hash(pos_D)]. \qquad (2.7)$$

Finally we iterate over L and check for each particle j if it is within the smoothing radius of the query particle i.e. $\|\mathbf{r}_Q - \mathbf{r}_j\| \leq h$ and add it to a resulting particle container.

2.2 Terrain

We implement the terrain as a heightmap. Because the water will only interact with the surface there's no need to import a 3D mesh with a complex geometry that might cause additional performance issues. It is sufficient to have only the surface of the terrain. That's what a heightmap (or heightfield) is designed for. We import a grayscale bitmap and build the terrain from the values of its
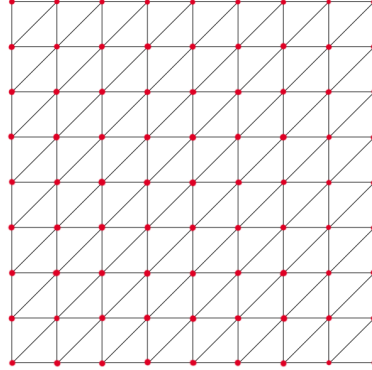
Figure 2.3: Top-view projection on grid accelerated terrain. Red dots represent geometry points in 3D space.

pixels. Each pixel in the bitmap has a value in range from 0 to 255. This value will serve as a height for a 3D point of our terrain. The bitmap is essentially a 2D matrix of pixels. Thus, we can easily build the entire terrain geometry by iterating over the entire bitmap extracting the height on each position. For a terrain point (x, y, z) we obtain its coordinates as follows

$$(x, y, z) = (i, j, h(i, j)), \tag{2.8}$$

where

$$h(i, j) - height\ value\ at\ i, j\ position, \tag{2.9}$$

i is the index that iterates from 0 to Dim_x of the bitmap, j is the index that iterates from 0 to Dim_y.

Looking from top-view projection at our terrain we can see that all points are evenly distanced between one another giving us a uniformly distributed grid (Figure 2.3). We are going to leverage this property to efficiently access any triangle. The distance between points in both directions will be represented by a 2D vector called *cell_size* and is going to be a user input value. Each cell is built up from 4 points and 2 triangles. We can map a 3D point (x, y, z) to a grid cell with index (x_i, y_i) as follows

$$(x_i, y_i) = (\lfloor \frac{x}{cell\_size} \rfloor, \lfloor \frac{y}{cell\_size} \rfloor). \tag{2.10}$$

In the equation (2.10) we don't need the height coordinate z, because we want to figure out where the point is on XY plane.

We can reverse the equation (2.10) as follows

$$(x_w, y_w) = (\lfloor x_i * cell\_size \rfloor, \lfloor y_i * cell\_size \rfloor). \tag{2.11}$$

Equation (2.11) converts a cell index into world space coordinates. The coordinates of vector (x_w, y_w) will always correspond to coordinates of "top-left" point of the (x_i, y_i) cell in XY projection. By extracting the height of this position we obtain the coordinates of the top-left point in 3D space

$$(x_w, y_w, h(x_w, y_w)). \tag{2.12}$$

By leveraging these properties we can efficiently determine where exactly a SPH particle is located on the grid and access all the appropriate vertices. We initialize the terrain by iterating over the bitmap once and for each (i, j) pair we compose its 3D version by using (2.8) and storing these geometry data in a dedicated buffer.

2.3 Collision

In section 1.4.3 we talked about how a SPH particle will detect and react on a collision with a heightfield. Further, in section 1.4.4 we described how we can employ the Leap-Frog time integration scheme to advance particles based on delta time step. In this section we are going to use them in collision handling algorithm.

Our goal is to register an intersection with a triangle or multiple triangles in 3D space. When a particle intersects the terrain it means that it de facto goes through the surface and ends up underneath it. Because all the triangles are located arbitrarily we have to implement a robust algorithm that will successfully detect any collision with any triangle at any point on the terrain. We can detect whether a particle i intersected a triangle by computing its next position in Δt time step

To detect whether the particle intersected with a given triangle we employ a ray casting technique. The idea is to cast a ray from the particle position along the directional vector. We can easily calculate the directional vector of a particle by subtracting its next position from the current

$$\mathbf{dir} = \mathbf{r}_{t+\Delta t} - \mathbf{r}_t, \tag{2.13}$$

where $\mathbf{r}_{t+\Delta t}$ is obtained by (1.46)

In order to detect a ray-triangle intersection we need to know three things: position, directional vector and the vertices of the triangle. We can efficiently obtain them by employing equations (2.10) and (2.11) to convert particle's position (x, y, z) into a cell index (x_i, y_i) and then obtain the top-left (in XY plane) point coordinates (x_w, y_w). To form a triangle we need to get the adjacent points within the cell. We get them by adding *cell_size* to each coordinate

$$(x_w, y_w)$$

$$(x_w + cell\_size, y_w)$$

$$(x_w, y_w + cell\_size)$$

$$(x_w + cell\_size, y_w + cell\_size)$$

By extracting the height value from the heightfield and assigning it to z coordinate we get 3D positions of these points

$$A = (x_w, y_w, h(x_w, y_w))$$

$$B = (x_w + cell\_size, y_w, h(x_w, y_w))$$

$$C = (x_w, y_w + cell\_size, h(x_w, y_w))$$

$$AA = (x_w + cell\_size, y_w + cell\_size, h(x_w, y_w))$$

We use Moller-Trumbore algorithm to detect a ray-triangle intersection. As an input we have to provide a position, a direction and three triangle's vertices. The output is a boolean value whether the collision has happened or not and the parameter t. This parameter determines the intersection point on the ray in the ray equation
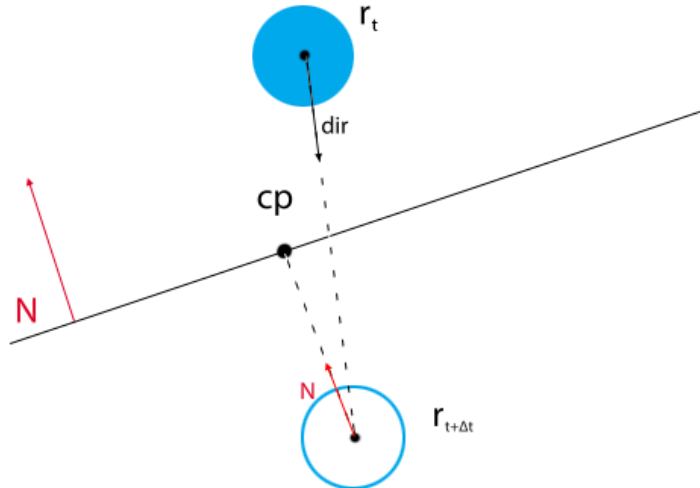
Figure 2.4: Particle is projected back onto the triangle it intersects with.

$$\mathbf{R} = \mathbf{O} + t * \mathbf{d}, \tag{2.14}$$

where $\mathbf{O}$ is the ray's origin and $\mathbf{d}$ is the direction vector

In our case, we want to determine the collision once the particle is underneath the surface, so we need to provide the next position $\mathbf{r}_{t+\Delta t}$, reversed direction -$\mathbf{dir}$ and three vertices of a triangle

$$\mathbf{bool}\ intersection = MollerTrumbore(\mathbf{r}_{t+\Delta t}, -\mathbf{dir}, \mathbf{A}, \mathbf{B}, \mathbf{C}, \&t) \tag{2.15}$$

If particle intersects a triangle it means that a collision happened and the particle must be projected back onto the surface (triangle). We calculate the contact point $\mathbf{cp}$ by applying the Moller-Trumbore one more time, but this time as a directional vector we specify the triangle's normal vector

$$\mathbf{bool}\ intersection = MollerTrumbore(\mathbf{r}_{t+\Delta t}, \mathbf{n}_{ABC}, \mathbf{A}, \mathbf{B}, \mathbf{C}, \&t). \tag{2.16}$$

The contact point is then obtained as follows

$$\mathbf{cp} = \mathbf{r}_{t+\Delta t} + t * \mathbf{n}_{ABC}. \tag{2.17}$$

The penetration depth d can be easily calculated

$$d = \|\mathbf{cp} - \mathbf{r}_{t+\Delta t}\| \tag{2.18}$$

Figure 2.4 depicts how SPH particle is projected back onto triangle.

(a) Particles intersects with a triangle, but failed to be projected on it along the triangle's normal $\mathbf{N}$.

(b) Particles is projected on the triangle by calculating the interpolation of normal vectors of the triangle it intersects with and the adjacent triangle.
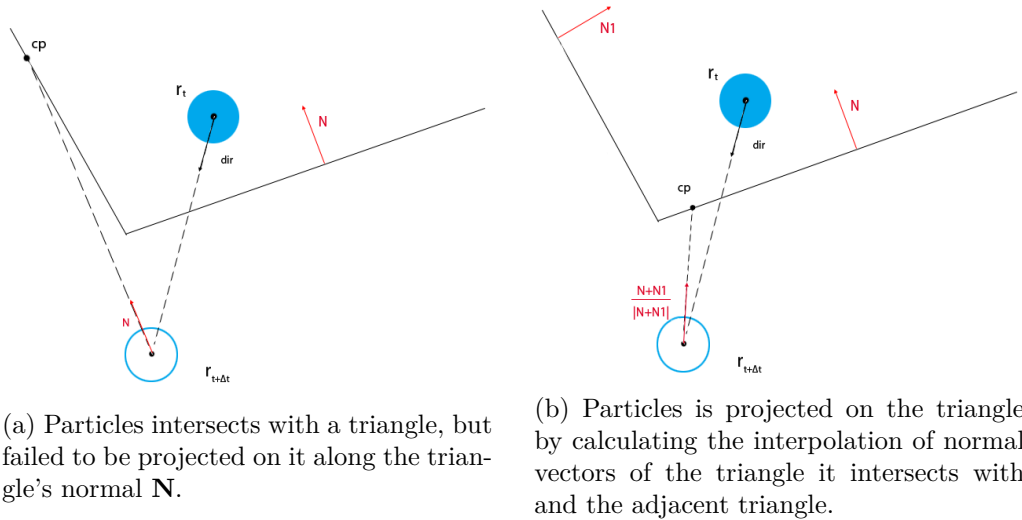
Figure 2.5: Edge case, when SPH particle is projected by means of interpolation of normal vectors.

The last thing to solve is edge cases. When the particle intersects the triangle somewhere on the edge we can have hard time projecting it back as shown on Figure 2.5a). We solve this problem by taking the both triangle's normal, calculating their interpolation and then trying to project the particle on either of those triangles Figure 2.5b).

The second edge case is when the particle intersects the triangle on its corner. Here we adhere to the same principle only this time we want to interpolate the normal vectors of six triangles and try to project the particle on each of them.

Let's wrap up the section with pseudo code of the entire algorithm

1: **function** Collision(*posCur*, *posNext*, &*cp*, &*norm*) -> bool
2:   $dir \leftarrow posNext - posCur$
3:   $cellIdx \leftarrow toCellIdx(posCur)$
4:   $cellIdxNext \leftarrow toCellIdx(posNext)$
5:   **if** $posCur = posNext$ **then**   ▷ if in the same cell
6:    $cellTriangles[] \leftarrow getCellTrinagles(cellIdx)$
7:    **if** $MollerTrumbore(posNext, -dir, cellTriangles[0], \&t)$ **then**
8:     $norm \leftarrow cellTriangles[0].norm$
9:     **if** $MollerTrumbore(posNext, norm, cellTriangles[0], \&t)$ **then**
10:      $cp \leftarrow posNext + t * norm$
11:      **return** true
12:     **else if** edge case within the same cell **then**
13:      $mapBetweenTrianglesOfTheSameCell(\&t, \&norm)$
14:      $cp \leftarrow posNext + t * norm$
15:      **return** true
16:     **else if** edge case between cells **then**
17:      $mapBetweenTrianglesWithAdjacentCell(\&t, \&norm)$
18:      $cp = posNext + t * norm$
19:      **return** true
20:     **else if** corner case **then**
21:      $mapBetweenSixTrianglesAtTheCorner(\&t, \&norm)$
22:      $cp \leftarrow posNext + t * norm$
23:      **return** true
24:     **end if**
25:    **end if**
26:    **if** $MollerTrumbore(posNext, -dir, cellTriangles[1], \&t)$ **then**
27:     $norm \leftarrow cellTriangles[1].norm$
28:     **if** $MollerTrumbore(posNext, norm, cellTriangles[1], \&t)$ **then**
29:      $cp \leftarrow posNext + t * norm$
30:      **return** true
31:     **else if** edge case within the same cell **then**
32:      $mapBetweenTrianglesOfTheSameCell(\&t, \&norm)$
33:      $cp \leftarrow posNext + t * norm$

```
34:            return true
35:        else if edge case between cells then
36:            mapBetweenTrianglesWithAdjacentCell(&t, &norm)
37:            cp = posNext + t * norm
38:            return true
39:        else if corner case then
40:            mapBetweenSixTrianglesAtTheCorner(&t, &norm)
41:            cp ← posNext + t * norm
42:            return true
43:        end if
44:      end if
45:    else                                                  ▷ If in different cells
46:      if corner case then
47:          mapBetweenSixTrianglesAtTheCorner(&t, &norm)
48:          cp ← posNext + t * norm
49:          return true
50:      else
51:          cellTriangles[] ← getCellTriangles(cellIdxNext)
52:          if MollerTrumbore(posNext, −dir, cellTrinagles[0], &t) then
53:              norm ← cellTriangles[0]
54:              if MollerTrumbore(posNext, norm, cellTrinagles[0], &t) then
55:                  cp ← posNext + t * norm
56:                  return true
57:              else
58:                  mapBetweenTrianglesWithAdjacentCell(&t, &norm)
59:                  cp ← posNext + t * norm
60:                  return true
61:              end if
62:          end if
63:          if MollerTrumbore(posNext, −dir, cellTrinagles[1], &t) then
64:              norm ← cellTriangles[0]
65:              if MollerTrumbore(posNext, norm, cellTrinagles[1], &t) then
66:                  cp ← posNext + t * norm
67:                  return true
68:              else
69:                  mapBetweenTrianglesWithAdjacentCell(&t, &norm)
70:                  cp ← posNext + t * norm
71:                  return true
72:              end if
73:          end if
74:      end if
75:      return false
76:    end if
77: end function
```

2.4 Hydraulic Erosion Model

In this section we will take a look on a high level, how an individual step of erosion is implemented. Here we will give definitions of different containers and functions that will be later used in Subsection 2.8 The algorithm of hydraulic erosion simulation can be broken down, according to [3] into the following steps:

1. Calculate fluid and boundary forces.

2. Calculate sediment transfer among SPH particles.

3. Calculate erosion and deposition exchange between SPH and boundary particles.

4. Update terrain height according to the change of sediment in boundary particles.

Let's elaborate on each step individually.

2.4.1 Boundary Forces

In this step the following things must be carried out:

1. Seed the cell in which a SPH particle i is located.

2. Detect the nearest boundary particles to the SPH particle i.

3. Calculate friction for particle i.

4. Add particle i to the neighbour list of all the boundary particles that it detected.

5. Append the detected boundary particles by SPH particle i to the container with all the detected boundary particles in this iteration.

6. Allocate buffer for each sph-boundary advection terms.

As we discussed in section 2.2 the position of a SPH particle can be translated into cell index to figure out where exactly on the terrain it is located. Once we figure out which cell it is we want to seed boundary particles over the cell's triangles (in subsection 2.6.3 we will talk about how we do this). The set of seeded particles are stored in a kd-tree (subsection 2.6.2). We use a hash table named $m\_SeededCells$ to store the boundary particles of each cell. The key of this table is a cell index and the value is kd-tree with boundary particles.

After we detected nearest boundary particles of a given SPH particle i we calculate friction(no-slip), as described in subsection 1.5.1.

Next we want to remember all the nearest boundary particles of the current particle i. We store all the detected boundary particles in current iteration in an unordered set $m\_NearestBoundaryParticles$. So, we easily merge the boundaries of particle i into $m\_NearestBoundaryParticles$.

Also we want to remember, what SPH particle is associated with what boundary particle, so we create a hash table, where we store a list of fluid particles for each boundary particles $m\_FluidsOfBoundary$. The key represents the id (while seeding each boundary particles is assigned one) and the value is the list of fluid particles.

Knowing how many boundary particles were detected and with what fluid particles each of them is associated we can allocate a container for each fluid-boundary interaction pairs dC_BP. This container will store the advection terms of each such pair from equation (1.69).

2.4.2 Sediment Transfer

SPH particles pass material to each other before it is settled back onto the surface. This process is described by the advection and diffusion equations (1.64) and (1.68). Both of them require to know the gradient of cubic spline $F(r, h)$, which according to [4] is

$$F(q, h) = \begin{cases} \frac{1}{4\pi h^3} * (12 * (1-q)^2 - 3 * (2-q)^2) & q < 1, \\ -\frac{1}{4\pi h^3} * 3 * (2-q)^2 & q \geq 1, \end{cases} \tag{2.19}$$

where

$$q = \|\mathbf{r}_{ij}\| * \frac{2}{h}. \tag{2.20}$$

Similarly as in Subsection 2.4.1 we store the advection term of each $i - j$ interaction pairs between all fluid particles. We name this collecion dC.

2.4.3 Deposition

Computing the deposition we store the advection term for each fluid-boundary interaction in $dC\_BP[ij]$.

The gradient of the cubic spline is the same as for sediment transfer (2.19). The part prior to the advection term in the equation (1.69) is responsible for converting the percentage of occupied volume C into the actual mass. This enables us to define the C to mass conversion as follows

$$C\_TO\_MASS(C) = \rho_s \frac{m_j}{\rho_j}. \tag{2.21}$$

We've already familiarized with these variables in Subsection 1.5.4

2.4.4 Erosion

The equation of erosion requires to know the relative velocity of current fluid particle to the surface, as it is stated in equation (1.55). We can obtain this velocity via dot product of the particle's velocity and the normal vector of the triangle

$$v_{rel} = \|\mathbf{v}\| - abs(\mathbf{v} \cdot \mathbf{n}), \qquad (2.22)$$

where $\mathbf{v}$ is the SPH particles velocity and $\mathbf{n}$ is the normal vector of the surface triangle.

Because the equation (1.57) only describes the change of mass on boundary particle b we must update the sediment ration of SPH particle j manually. We do so by converting the eroded mass produced by (1.57) for each SPH particle j and convert it to C as follows

$$MASS\_TO\_C(m) = \frac{1}{\rho_s * m_j/\rho_j} * m, \qquad (2.23)$$

where m is the output of equation (1.57).

2.4.5 Terrain Modification

At this point all the detected boundary particles have their delta mass dM calculated. Each such particle is located on some triangle, this means that we can calculate the total mass for each triangle. We store the total masses of both triangles of a cell in a hash table named $m\_TriangleMass$, where key is cell index and value is pair of two floats representing the masses of each triangle individually. Also we keep track only of those cells whose boundary particles were detected and stored in $m\_NearestBoundaryParticles$ container. We store those cells in a vector named $cellsToUpdate$. After all the boundary particles have been iterated over and masses accumulated we update the geometry of cells from the $cellsToUpdate$ container.

2.5 Fluid Particles

In this subsection we'll talk about what fields a fluid particle structure must have. Unlike a regular SPH particle we need to extend ours due to the fact that we implicitly represent sediment particles. We append $float\ Sedim$ that represents how much volume is occupied by sediment in per cents and $float\ Sedim\_delta$ that represents the change of occupied volume in the current iteration. The structure below defines a SPH particle

```
struct FluidParticle
{
    Vec3f Position;
    Vec3f Velocity;
    Vec3f Acceleration;
    float Density;
    float Pressure;
    Vec3f PressureForce;
    Vec3f ViscocityForce;
    Vec3f GravityForce;
    Vec3f SurfaceForce;
    Vec3f SurfaceNormal;
    Vec3f BoundaryForce;
    float Sedim;
    float Sedim_delta;
}
```

2.6   Boundary particles

In Subsection 1.5.1 we discussed that our erosion model uses boundary particles as a mediator between the fluid and the terrain. SPH particles first cause impact over boundary particles and then this impact is transformed into the terrain's changes. With that in mind we need to have them seeded throughout the whole terrain, because SPH particles must access them a any give position on the terrain. Having all of them everywhere at once will be inefficient that's why we will seed them only at places where SPH particles are.

2.6.1   Definition

Similarly as with fluid particles let's define the structure for boundary particles.

```
struct BoundaryParticle
{
    Vec3f Position;
    Vec3f TriangleNormal;
    char Triangle;
    float dM;
}
```

The *char Triangle* field denotes on which of the two triangles of a cell a given boundary particle located. In this implementation we use either $'A'$ or $'B'$. The *float dM* denotes the change of mass of the current particle described by (1.69).

2.6.2 Kd-tree

In section 2.2 we defined our terrain as a heightfield. We made it grid-accelerated and established that each cell consists of two triangles. When a SPH particle approaches the surface it detects all the boundary particles within its smooth radius as depicted on Figure 1.8. In order to efficiently find the closest boundaries we store them in a kd-tree. For each cell we build a new kd-tree and store its particles in there. Each kd-tree is then stored in a hash table, where key is a cell index and value is a pointer to allocated kd-tree. When a fluid particle enters the cell we apply nearest neighbours search algorithm and detect, which particles are within the smooth radius.

2.6.3 Seeding

Before we store any particles in a kd-tree we need to seed them. The principle of this seeding algorithm was derived from famous *scan line algorithm.* We seed the particles in scan line fashion for a triangle in 3D space. Consider the Figure 2.6. We have a cell composed of two triangles **ABC** and **DBC**. Let's say we want to seed particles over the triangle **ABC**. We do so by moving from **C** to **A** along the **AC** vector with Δs step and at the same time we move along the hypotenuse **BC**. Thus, we obtain two points to compose a line segment that is parallel with the cathetus **AB**. Finally, we move along this line segment with the same Δs step from left to right i.e. from cathetus **AC** to hypotenuse **BC** and place particles. Because all the points of both triangles are 3D points we virtually fill the triangle in 3D space. The same principle applies for the second triangle only now we move along the **DC** vector.
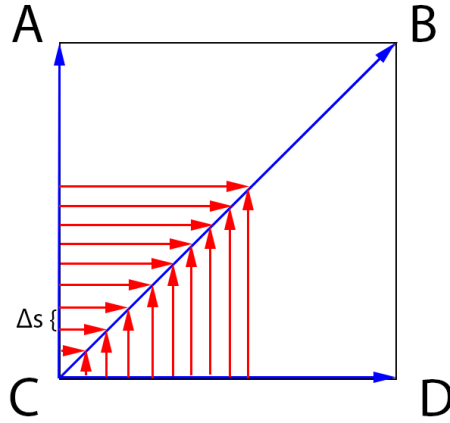
Figure 2.6: Seeding triangles of a cell in 3D space.

2.7 Physical Parameters

In this section we describe what actual quantities we use in the simulation.

2.7.1 Fluid

To achieve the most stable and reliable behaviour of fluid its physical parameters must be carefully chosen. The search of such values that enable to achieve a high level of realism and at the same time keep the simulation stable might be a tough challenge. Below is the Table 2.1 for water material taken from [1] with physical parameters of fluid quantities in the standard System International (SI) units, that provide good simulation results in terms of both realism and stability. In this thesis we focus only on water material.

2.7.2 Erosion

Throughout the Section 1.5 a number of constants are used that weren't defined so far. Most of them were taken from [3], however the boundary friction constant μ_{bf} was determined experimentally. The value that gives the most realistic behaviour varies between 0.1 and 0.25. In this work 0.11 was chosen. Table 2.2 presents the constants used in erosion model.

Description	Symbol	Value	Unit
Density (rest)	ρ_0	998.29	$\frac{kg}{m^3}$
Mass (particle)	m	0.02	kg
Viscosity	μ	3.5	$Pa \cdot s$
Surface tension	σ	0.0728	$\frac{N}{m}$
Gas stiffness	k	3	J
Restitution	c_R	0	n/a
Smooth radius	h	0.0457	m

Table 2.1: Physical parameter for water simulation

2.8 The Simulation Algorithm

2.8.1 Build the Terrain

1. Read the bitmap with the dimensions Dim_x, Dim_y and store it in a 1D array $m\_Heightfield$.

2. For z from 0 to Dim_x do:

   a) For x from 0 to Dim_y do:

      i. Get height at (x, z) using (2.9): $y = h(x, z)$.
      ii. Add the coordinates of the 3D point into $vertexData$ buffer considering the cell size:
      iii. $vertexData.append(x * cell\_size)$.
      iv. $vertexData.append(y)$.
      v. $vertexData.append(z * cell\_size)$.

The $vetexData$ buffer is regenerated each time the terrain gets updated and is only used for passing the geometry data to OpenGL pipelines to render the visual output. The origin of the terrain is at $(0, 0)$, which theoretically can be changed, but is not necessary in this simulation. When a particle interacts

Description	Symbol	Value	Unit
Distance between boundary particles	Δs	0.02285	m
Boundary friction constant	μ_{bf}	0.11	n/a
Erosion strength	K_ε	0.1	n/a
Critical shear stress	τ_c	3	Pa
Sediment density	ρ_s	3	$\frac{kg}{m^3}$
Maximum solid volume fraction	C_{max}	0.7	n/a
Exponent (Richardson-Zaki)	e	4.5	n/a
Coefficient of diffusion	D	0.1	n/a
Sediment particle radius	r_s	0.00001	m

Table 2.2: Physical parameter for erosion simulation

with the terrain it translates its 3D position into a cell index, in which it is currently located by using (2.10). Knowing the cell index coordinates we access the heightfield buffer and extract a corresponding height of each point of the cell. Thus, we don't need to access *vertexData* buffer to obtain point's coordinates.

2.8.2 Initialize SPH system

1. Initialize fluid constants from the Table 2.1.

2. Allocate n particles and set them positions, velocities, $acceleration = 0$, $sediment = 0$, $sediment\_delta = 0$.

3. Create the spatial hashing data structure using (2.3) and (2.4).

4. Create an empty **unordered_set** $m\_NearestBoundaryParticles$, where we will store all the detected boundary particles in a given iteration that are within the smooth radius of fluid particles.

5. Create an empty **unordered_map** $m\_FluidsOfBoundary$ that will store a list of fluid particles that detected a given boundary particle.

6. Create an empty vector dC defined in subsection 2.4.2, where we will store all the interaction pairs between fluid particles.

7. Create an empty vector dC_BP defined in section 2.4.3, where we will store all the interaction pairs between a fluid and a boundary particle.

We can omit the initialization of the leap-frog integrator for each particle here. It will be done automatically while calculating the acceleration term in further subsection (Subsection 2.8.11) in the very first iteration of simulation.

2.8.3 Compute Density and Pressure

Before we begin iterating through the particles we insert them into the spatial hashing data structure and build a list of neighbours for each of them using the spacial query from Subsection 2.1.2. Let's call this procedure $RebuildTable()$. It clears the table from the previous iteration and builds it back for the current one.

$RebuildTable()$
(The following for-loop is parallelised with OpenMP)
For each particle i do:

1. Iterate through each of its neighbour j from the neighbourhood N_i and compute its density ρ_i using (1.25)

2. Compute pressure using (1.32).

2.8.4 Compute Internal Forces

(The following for-loop is parallelised with OpenMP)
For each particle i do:

1. Iterate through each of its neighbour j from the neighbourhood N_i.

2. Compute the pressure force using (1.33).

3. Compute the viscosity force using (1.35).

4. Compute the surface normal using (1.39).

5. $f^{internal} = f^{pressure} + f^{viscosity}$

2.8.5 Compute External Forces

(The following for-loop is parallelised with OpenMP)
For each particle i do:

1. Compute the gravity force using (1.36).

2. Compute the color field using (1.38).

3. Compute the surface curvature using (1.40).

4. Compute the surface tension force using (1.41).

5. $f^{external} = f^{gravity} + f^{surface}$

2.8.6 Compute Boundary Forces

(The following for-loop is parallelised with OpenMP)
For each particle i do:

1. Seed the cells that are in range of smooth radius of the particle i, which are not seeded, using the technique described in the Subsection 2.6.3.

2. $f^{boundary} = 0$.

3. Find nearest boundary particles to the particle i using the $nearest neighbour search$ algorithm of kd-tree for each detected cell around particle i and put them to a temporary collection **unordered_set** $nearest\_boudary$.

4. If $nearest\_boundary$ is not empty:

   a) For each boundary particle bp in $nearest\_boundary$:
      i. Compute the no-slip using (1.52) and add it to $f^{boundary}$

5. For each boundary particle bp in $nearest\_boundary$ push back the current fluid particle i into the $m\_FluidsOfBoundary[bp]$.

6. Merge $nearest\_boundary$ into $m\_NearestBoundaryParticles$.

2.8.7 Compute Sediment Transfer

For each fluid particle i do:

1. For each neighbour particle j from neighbourhood N_i:

   a) Compute the advection term of $i - j$ interaction using (1.64) and set the result to dC.

   b) Compute the diffusion term of $i - j$ interatction using (1.68) and add the result to dC.

   c) If dC of current $i - j$ pair is ≤ 0 then: $sedim\_delta_i\ +=\ dC[ij]$ else: $sedim\_delta_j$ -= $dC[ij]$.

   The size of dC is the number of interaction pairs between fluid particles in a given iteration that can be computed from the neighbours list of each particle from the spatial hashing data structure.

2.8.8 Compute Deposition

For each boundary particle bp in $m\_NearestBoundaryParticles$:

1. For each fluid particle fp in $m\_FluidsOfBoundary[bp]$:

   a) If $fp.sedim \leq 0$ then $continue$.

   b) Compute the term $\frac{dC(j)}{dt} from(1.69)$ and put the result to dC_BP.

   c) If $dC\_BP < 0$ for the current $bp - fp$ pair:

      i. Add dC_BP to sediment delta of fp.

      ii. Compute the deposition of material using (1.69) and subtract the result from the mass of the current boundary particle bp.

2.8.9 Compute Erosion

For each boundary particle bp from $m\_NearestBoundaryParticles$:

1. For each fluid particle fp from $m\_FluidsOfBoundary[bp]$:

   a) Compute the erosion of material using (1.57) and subtract the result from mass of current boundary particle bp.

   b) Convert the result of erosion from the previous step into the sediment percentage C and add it to sediment delta of current fluid particle fp.

2.8.10   Update Heightfield

For each boundary particle bp in $m\_NearestBoundaryParticles$:

1. Convert $bp.Position$ to $cell\_index$.

2. If $cell\_index$ was not discovered before then create $triangleMass[cell\_index] = pair(0.0, 0.0)$ and $cellsToUpdate.append(cell\_index)$.

3. Depending to which triangle the boundary particle bp belongs add $bp.dM$ to either $triangleMass[cell\_index].first$ or $triangleMass[cell\_index].second$.

4. For each $cell\_index$ in $cellsToUpdate$:

   a) Get geometry of cell triangles $getCellTriangles(cell\_index)$.

   b) Update vertices of both triangles from the previous step in the heightfield using (1.70) and (1.71).

   c) Erase all boundary particles in the cell $cell\_index$ (they will be seeded again in Step 1 of 2.8.6).

2.8.11   Time Integration and Collision Handling

(The following for-loop is parallelised using OpenMP)
For each fluid particle i:

1. $f^{external} \mathrel{+}= f^{boundary}$.

2. $F_i = f^{internal} + f^{external}$.

3. Compute particle's acceleration a_i using (1.24).

4. Compute $\mathbf{u}_{t+1/2\Delta t}$ and $\mathbf{r}_{t+\Delta t}$ using (1.45) and (1.46).

5. Compute collision using the algorithm from Subsection 2.3.

6. If a collision occurred

   a) Compute penetration depth d using (2.18).

   b) Update the velocity using (1.44).

7. If sediment of current particle i is $sedim_i < 0$ then $sedim_i = 0$

8. $sedim_i \mathrel{+}= \Delta t * sedim\_delta_i$.

9. If $sedim_i < 0$ then $sedim_i = 0$.

10. Zero out delta sediment of the current particle: $sedim\_delta_i = 0$.

2.8.12   Render

1. Render the terrain using *vertexData* buffer.

2. For each fluid particle i do:

    a) Render a sphere with its center at $\mathbf{r}_i$ and radius r = 0.001.

  We do not render the boundary particles, because we need them only to perform the computations.

Results

In this chapter we will examine what outcomes we obtain when we put to use our implementation from Chapter 2. We will test the fluid individually in order to demonstrate how it behaves in the first place and then we add a terrain into the scene and pour some volume on it and simulate erosion.

3.1 Fluid

In this section we will test, how the Lagrangian solver works. First we will examine how it behaves with different parameter values and then we execute performance tests.

3.1.1 Fluid Parameters

Below is the illustration of a water volume with 1000 particles dropping in the invisible box Figure 3.1. The properties of the fluid are the exact much of those described in Table 2.1.

The water material has very low viscosity as a result its particles are very mobile and agile. When a water volume hits a solid the particles splash all over. If we risen viscosity i.e. risen inner friction between particles the volume will bear a more resemblance with honey. Below is the same volume with higher viscosity Figure 3.2.

Compared to the previous animation the particles don't move around so freely and are less splashy. Their motion is slowed down by friction with each other.

If we try to increase the particle's mass the density field will increase, hence the pressure will increase too. This implies that the particles will repel more from each other and have more space in between them Figure 3.3. They are splashy enough, but too heavy to bounce off too high.

The above tests demonstrate that our Lagrangian solver produces the expected behaviours and has a pretty high degree of realism.
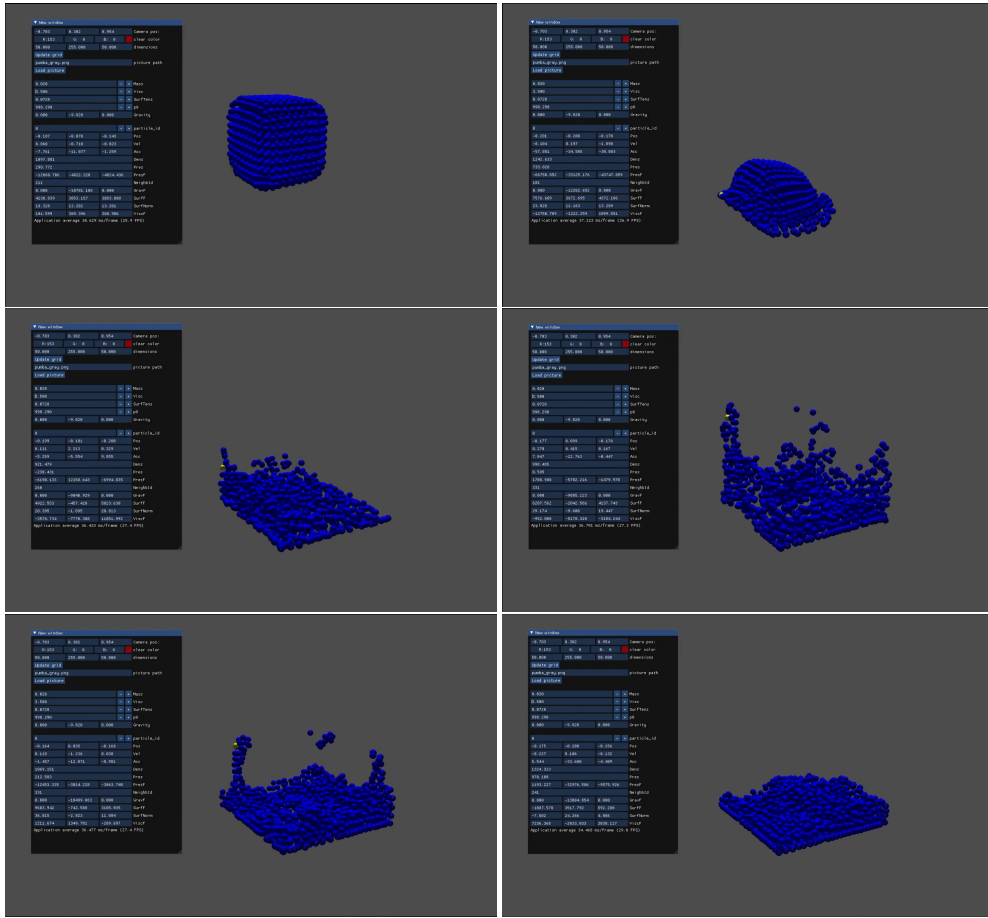
Figure 3.1: 1000 SPH particles with the parameters of water.

3.1.2 Performance Test

In this subsection we present the results of performance test of SPH system Table 3.1. The frequencies are measured in frames per second. The simulation was run on Intel(R) Xeon(R) W-2245 CPU @ 3.90GHz 3.91GHz with 128GB of RAM on Windows 10.

3.2 Erosion

Now that we have our water tested let's pour it onto an actual terrain and have the erosion algorithm to do its job.

3.2.1 Ditch

Figure 3.4 features a ditch with a ridge in the middle. The volume of 200 000 SPH particles is poured onto it. The radius of each particle is $r = 0.001m$.
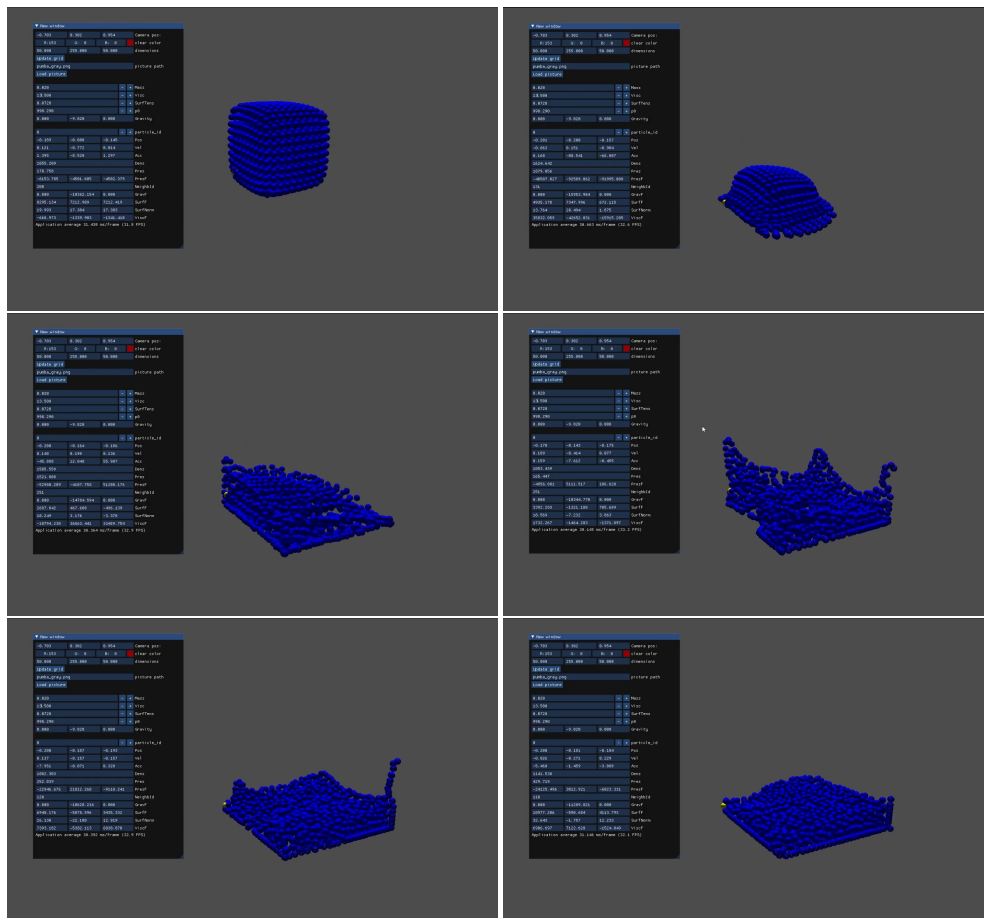
Figure 3.2: 1000 SPH particles water particles with increased viscosity (visc=17.5).

The cell size is $cell\_size = 0.1m$. Particles life time is 3000 frames.

The ridge in the middle seeks to being washed out and leveled off with the surface. The material from the ridge is taken away by water particles and deposited towards the area on left from it. The longer the particles exist the more the ridge gets leveled off with the surface.

The ridge in the middle is the spot that gives the most pronounced visual results in fact the entire terrain gets eroded and the surface level throughout the scene lowers a little. The comparison before/after is illustrated on Figure 3.5.

3.2.2 Meander

The next scene is a meander. Similarly as in the previous test a volume of water with 200 000 particles is poured on. The particles lifetime is 2000 frames
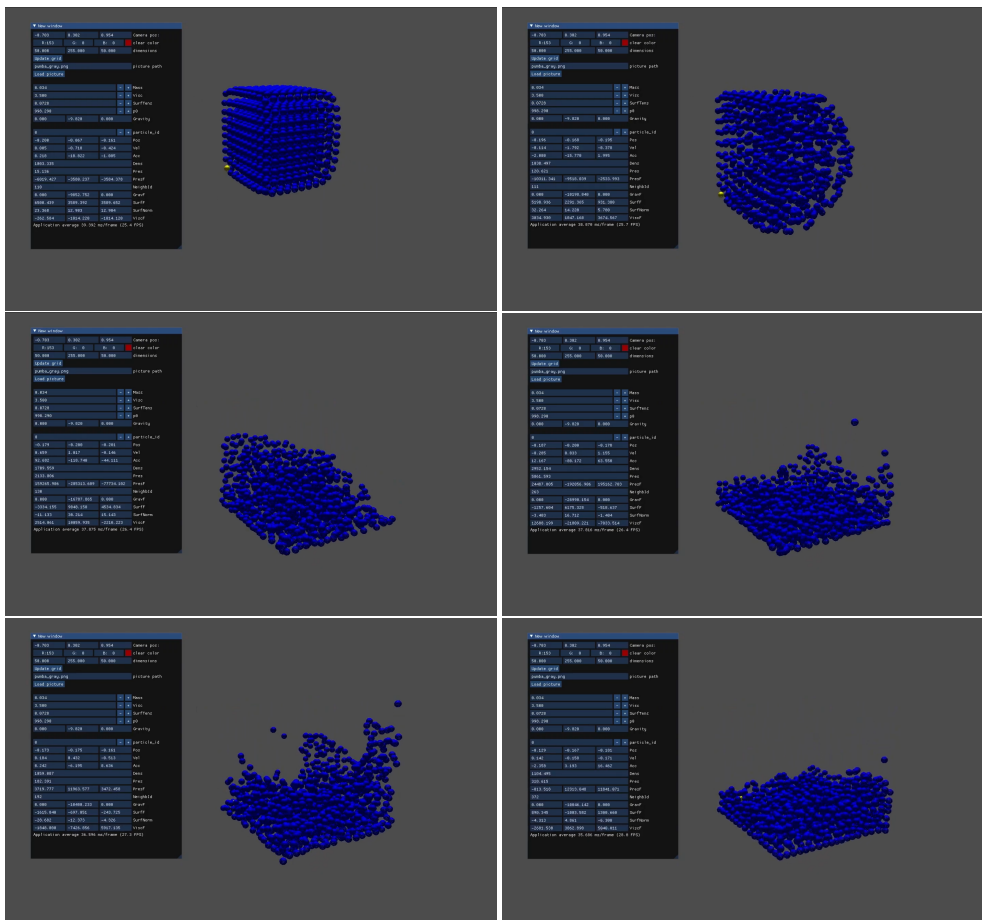
Figure 3.3: 1000 SPH water particles with increased mass and default viscosity (mass=0.034)

here. As can be seen of Figure 3.6 the material reached by fluid gets eroded while the part of the meander where fluid didn't reach remains untouched.

3.2.3 Performance Test

While simulating erosion SPH particles will get to interact with boundary particles that they detect. Unlike in the case with calculations of fluid forces interaction between boundary and fluid particles cannot be parallelized and is calculated sequentially. Table 3.2 illustrates the results of erosion simulation with different ammount of particles. The choice of map doesn't influence the performance very much, since SPH particles detect boundary particles within their smooth radius. On any map a SPH particle encompasses on average the same amount of boundary particles.

n	fps
8000	45.7
10000	36.2
20000	17.9
50000	6.1
100000	3.5

Table 3.1: SPH system performance test.

n	fps
5000	13.6
10000	6.1
25000	2.3
50000	1.3
100000	0.7
200000	0.4

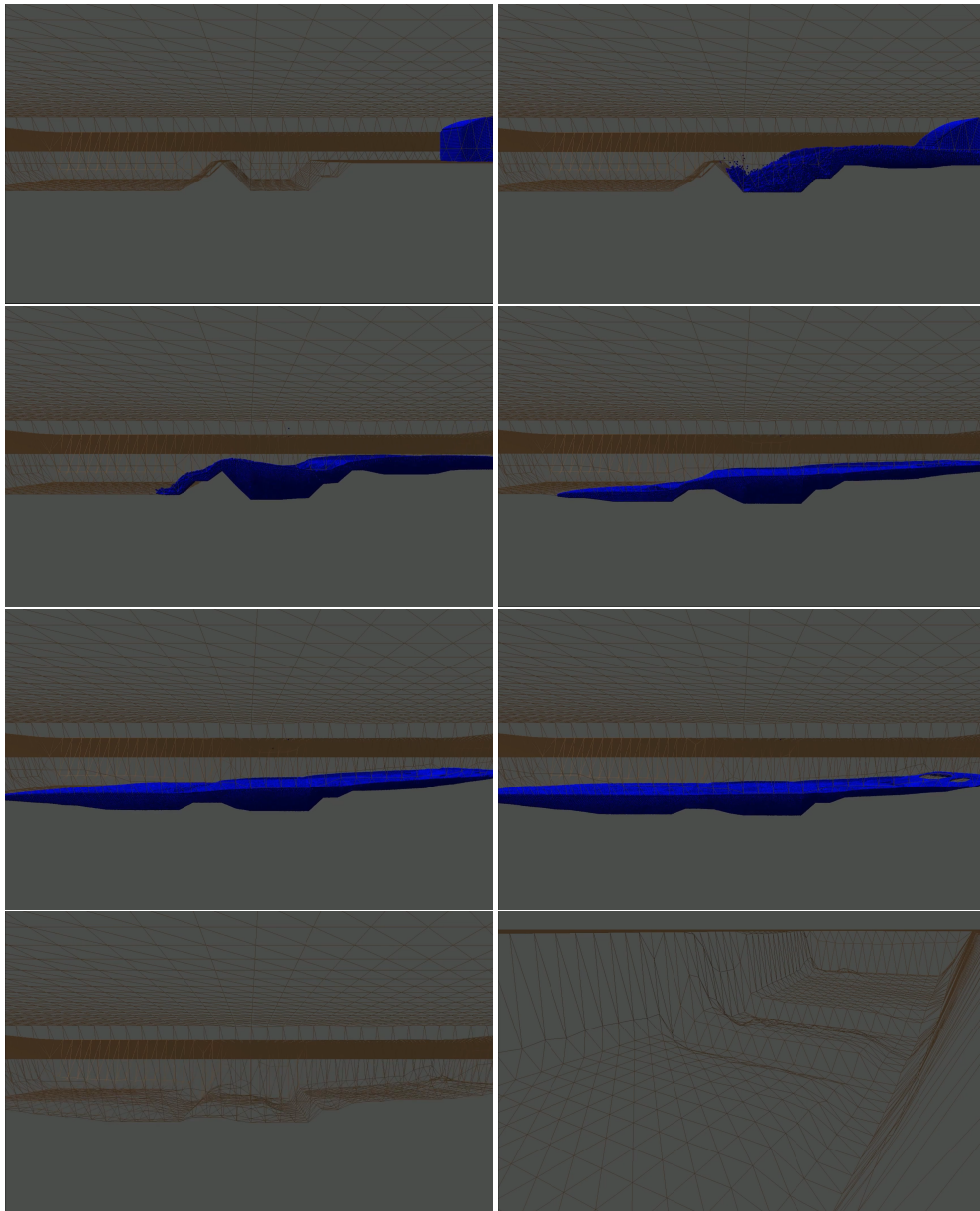Table 3.2: Results from performance test of SPH system + erosion.

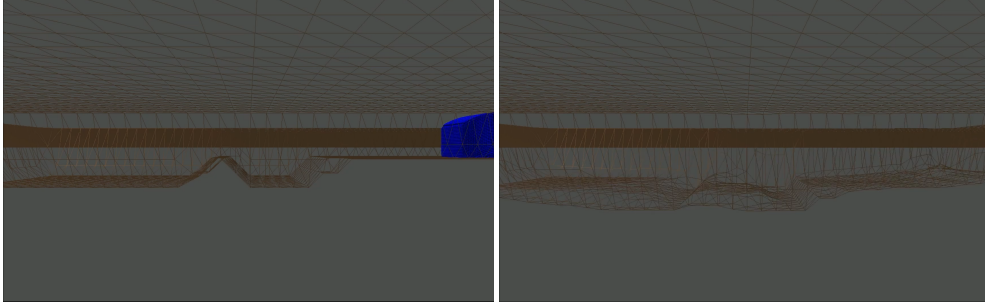Figure 3.4: A ditch with a ridge. 200 000 SPH particles.

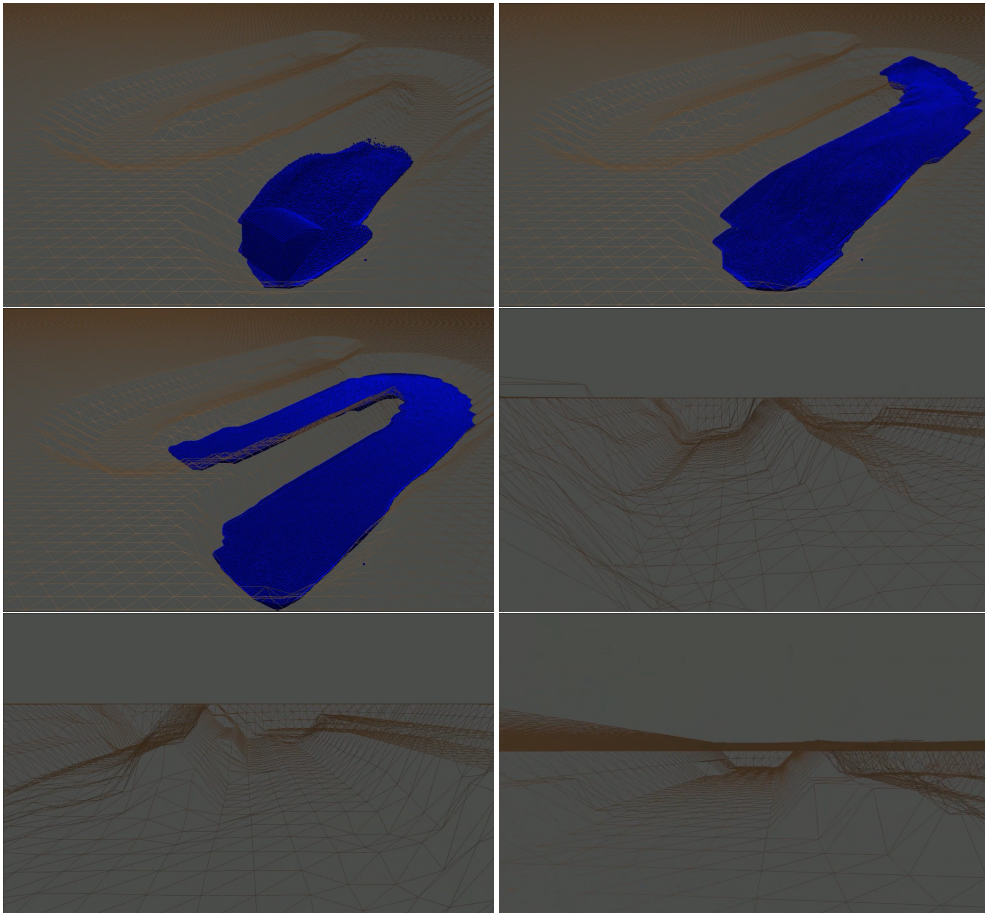Figure 3.5: Before/after. The ditch with the ridge. 200 000 SPH particles.

Figure 3.6: A meander. 200 000 SPH particles.

CHAPTER 4

Future Work

This work can be extended to leverage GPU power to parallelize fluid simulation. Although the paralleliztion via CPU was used here it doesn't compare with the GPU parallelization. This will enable to create greater scale simulations with more particles. Such boost in performance will allow to achieve the same results as in this work in much shorter time. A very popular tool for general-purpose computing on GPU is CUDA platform from NVIDIA.

Next thing that might be done is improving the visual outputs. As of right now the water is rendered as a set of spheres. This is good, because we can easily keep track of every particle and immediately spot if somethings goes wrong. It serves a good purpose in developing the system, however once we are confident that the system is stable enough and will want to deliver a graphically pleasant result then improving the visual part sounds like a good idea. Particularly the technique named *Screen Space rendering*[11] might be a good option to render translucent surface.

Conclusion

In this work the simulation as well as visualization of hydraulic erosion was implemented. It first went through the theory behind fluid simulation and then hydraulic erosion model. The implementation section of this work put in practice theoretical concepts of both fluid system and erosion model and demonstrated how these concepts can be incorporated. To improve the performance of the entire simulation the fluid system was designed to leverage the parallelism of CPU. In the end the preference to heightmap over voxel grid was given due to the improvement in efficiency.

Bibliography

[1] Kelager, M. Lagrangian fluid dynamics using smoothed particle hydro-dynamics. *University of Copenhagen: Department of Computer Science*, volume 2, 2006.

[2] Novák, O. Simulace viskózních kapalin. *Prague, 2007. Master thesis. CTU in Prague, Faculty of Electrical Engineering, Department of computing.*

[3] Krištof, P.; Beneš, B.; et al. Hydraulic erosion using smoothed particle hydrodynamics. In *Computer Graphics Forum*, volume 28, Wiley Online Library, 2009, pp. 219–228.

[4] Monaghan, J. J. Smoothed particle hydrodynamics. *Annual review of astronomy and astrophysics*, volume 30, no. 1, 1992: pp. 543–574.

[5] Müller, M.; Charypar, D.; et al. Particle-based fluid simulation for interactive applications. In *Symposium on Computer animation*, volume 2, 2003.

[6] Desbrun, M.; Gascuel, M.-P. Smoothed particles: A new paradigm for animating highly deformable bodies. In *Computer Animation and Simulation'96*, Springer, 1996, pp. 61–76.

[7] Wojtan, C.; Carlson, M.; et al. Animating Corrosion and Erosion. In *NPH*, Citeseer, 2007, pp. 15–22.

[8] Partheniades, E. Erosion and deposition of cohesive soils. *Journal of the Hydraulics Division*, volume 91, no. 1, 1965: pp. 105–139.

[9] Richardson, J.; Zaki, W. Sedimentation and fluidisation: Part I. *Chemical Engineering Research and Design*, volume 75, 1997: pp. S82–S100.

[10] Teschner, M.; Heidelberger, B.; et al. Optimized spatial hashing for collision detection of deformable objects. In *Vmv*, volume 3, 2003, pp. 47–54.

[11] van der Laan, W. J.; Green, S.; et al. Screen space fluid rendering with curvature flow. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games*, 2009, pp. 91–98.

Acronyms

SPH Smoothed Particle Hydrodynamics

NSE Navier-Stokes equations

Contents of enclosed CD

readme.txt . . . . the file with CD contents description
exe . . . . the directory with executables
src . . . . the directory of source codes
    wbdcm . . . . implementation sources
    thesis . . . . the directory of LaTeX source codes of the thesis
text . . . . the thesis text directory
    thesis.pdf . . . . the thesis text in PDF format