

Real-time Simulation of Granular Matter using Smoothed Particle Hydrodynamics

Masterarbeit

zur Erlangung des Grades Master of Science (M.Sc.)
im Studiengang Computervisualistik

vorgelegt von
André Neder

Erstgutachter: Prof. Dr.-Ing. Stefan Müller
(Institut für Computervisualistik, AG Computergraphik)
Zweitgutachter: Alexander Maximilian Nilles, M.Sc.
(Institut für Computervisualistik, AG Computergraphik)

Koblenz, im Januar 2024

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Ja Nein

Mit der Einstellung der Arbeit in die Bibliothek bin ich einverstanden. ☐ ☐

(Ort, Datum) (Unterschrift)

Inhaltsverzeichnis

1	Introduction	1
2	Related Work	2
3	Basics	3
3.1	Navier-Stokes equation	3
3.2	Signed Distance Functions	3
3.3	Bitonic Sort	4
4	Method	5
4.1	Smoothed Particle Hydrodynamics	5
4.1.1	Weakly Compressible SPH	5
4.1.2	Predictive-Corrective Incompressible SPH	5
4.1.3	Implicit Incompressible SPH	6
4.2	Libraries & Technologies	6
5	Implementation	8
5.1	Neighborhood Search	8
5.2	Density	9
5.3	Unilateral incompressibility	9
5.4	External forces	10
5.5	Gravity	10
5.6	Drag	10
5.7	Strain & Stress	11
5.8	Pressure	12
5.9	Internal Forces	13
5.10	Integration	13
5.11	Boundary Handling	13
5.11.1	Particle-Based Approaches	13
5.11.2	Volume Maps	14
5.12	Upsampling	15
5.13	Visualization	16
6	Evaluation	17
6.1	Performance	17
7	Conclusion & Future Work	18

1 Introduction

Granular matter consists of discrete, macroscopic particles that interact through contact forces, exhibiting unique behaviors such as flowing, layering or piling. Simulating such behaviors in realistic scenarios is especially challenging, due to the necessity of employing a very high number of discrete elements to achieve visually plausible simulations. Granular materials find diverse applications across geophysics, civil engineering, pharmaceuticals, and other industries, playing crucial roles in phenomena such as landslides, construction, powder mixing, storage and transportation. A powerful technique for simulating such materials is Smoothed Particle Hydrodynamics (SPH). SPH is a meshless Lagrangian method that has been employed a lot in fluid dynamics and can also be used to simulate granular behavior. By combining well-established SPH techniques for simulating granular materials with the latest developments in SPH research, real-time simulations might become achievable. In the pages ahead, I will assess the doability of real-time simulations for granular matter, by combining established methods and recent advancements to achieve a dynamic and interactive implementation.

2 Related Work

...

3 Basics

3.1 Navier-Stokes equation

The basic concept behind most fluid simulations lies in the Navier-Stokes equations, which describe the motion of fluids and deformable solids. As granular flow can be interpreted as a kind of fluid they will be employed here aswell. One of the most important parts is the continuity of mass equation, which „describes the evolution of an object’s mass density ρ over time, i.e.,“ [11]

$$\frac{D\rho}{Dt} = -\rho(\nabla v) \quad (1)$$

with v beeing the velocity and ∇ the gradient operator. It states that the mass of fluid entering a given region must be equal to the mass leaving that region, accounting for any change in density within the fluid. The second part is the incompressible Navier-Stokes equation

$$\rho \frac{Dv}{Dt} = -\nabla p + \mu \nabla^2 v + f_{ext} \quad (2)$$

which describes the conservation of momentum. Here „the pressure p can be interpreted as a Lagrange multiplier that has to be chosen such that the“[11] conservation of mass is fulfilled. μ denotes the dynamic viscosity of the fluid and f_{ext} the influence of external forces.

3.2 Signed Distance Functions

Signed distance functions (SDF) are a versatile tool in computer graphics. They determine the shorted distance of a point in space to the surface of geometric object, while preserving information about if the point is inside or outside the object. This allows for precise implicit reconstruction of surfaces. A greate source for understanding and utilizing SDFs has been [21].

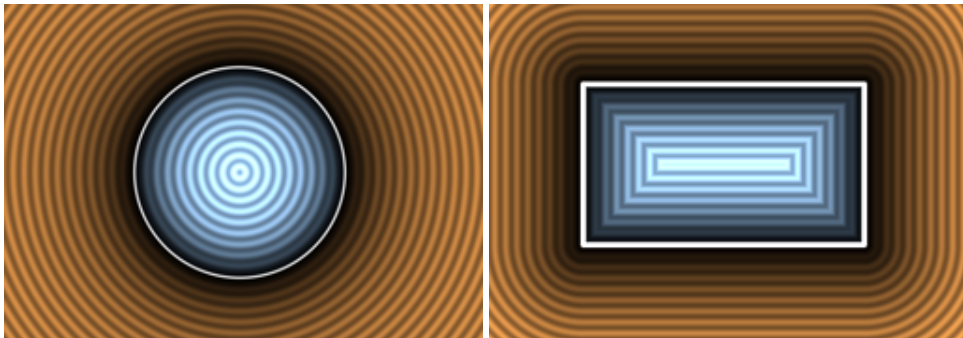


Abbildung 1: SDF of a circle and a box in 2D [22]

3.3 Bitonic Sort

Bitonic merge sort is a sorting algorithm with a performance complexity of $O(\log^2(n))$. The algorithm can be implemented on the GPU to achieve a massive performance increase by parallelization. In this work, a modified version of the implementation by @tgfrerer has been used [23]. The algorithm uses a pattern of comparison to sort a list of elements in parallel. The algorithm has been modified to use Vulkan's Push Constants to select the type of comparison inside of the shader provided by [23]. Another change is the use of Specialization Constants to set the local workgroup size of the compute shader invocation.

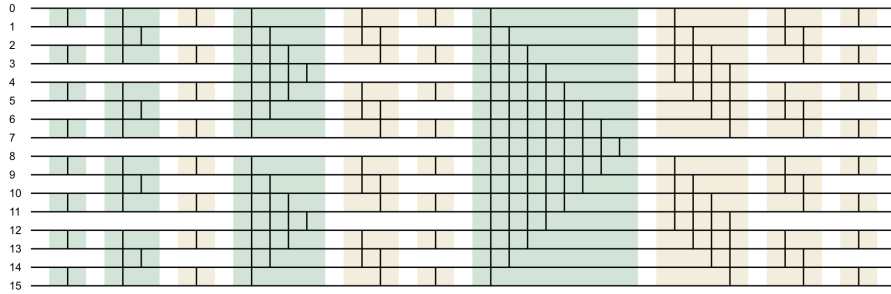


Abbildung 2: Sorting diagram for 16 sortable elements. Horizontal lines represent array indices into our array of sortable elements, each vertical line represents one worker thread performing a compare-and-swap operation. [23]

4 Method

4.1 Smoothed Particle Hydrodynamics

Smoothed Particle Hydrodynamics (SPH) is a computational technique widely employed in fluid dynamics and other fields for simulating complex physical phenomena. SPH represents the fluid or material as a collection of discrete particles. Each particle has physical properties such as density, pressure and velocity.

These interact through a kernel function W with radius h , which smoothes these properties over neighboring particles, allowing for a continuous representation of the material and facilitating the simulation of intricate behaviors, including fluid flow, collisions, and deformations. An arbitrary field A can be approximated as shown, with an error in the order of h^2 .

$$A(r) \approx \int A(r')W(|r - r'|, h)d(r') \quad (3)$$

The integral is then approximated using a Riemann summation over the neighboring particles.

$$A(r) \approx \sum_j A_j W(|r - r_j|, h) \quad (4)$$

This concept is then applied to calculate the properties of a given particle. SPH's versatility extends beyond fluids, making it applicable to problems involving granular materials, astrophysics, and even solid mechanics.

//Todo: more about the kernel ?

4.1.1 Weakly Compressible SPH

Weakly Compressible Smoothed Particle Hydrodynamics (WCSPH) is a variant of the traditional SPH method where, the incompressibility constraint is relaxed, allowing for a more computationally efficient simulation while still capturing the essential fluid behaviors. This approach is particularly useful in scenarios where the fully incompressible SPH formulation might be computationally expensive and it finds a sweet spot between precision and computational efficiency. This is especially useful in the context of large-scale simulations or real-time applications.

4.1.2 Predictive-Corrective Incompressible SPH

Predictive-Corrective Incompressible Smoothed Particle Hydrodynamics (PCISPH) is an advancement in the SPH method designed to address issues related to incompressibility in fluid simulations. It employs a two-step process where a predictive step estimates particle velocities and a corrective step which iteratively adjusts these velocities to enforce incompressibility. This technique

is particularly useful in scenarios where traditional SPH methods such as WCSPH struggle with maintaining fluid volume conservation, providing more accurate and stable simulations, especially for scenarios involving complex fluid interactions, like fluid splashes or turbulent flows.

4.1.3 Implicit Incompressible SPH

Implicit Incompressible Smoothed Particle Hydrodynamics (IISPH) is a variant of the SPH method that also addresses the challenge of maintaining incompressibility in fluid simulations. Unlike traditional SPH methods, IISPH employs an implicit formulation for the pressure term in the Navier-Stokes equations, allowing for larger time steps and improved stability.

4.2 Libraries & Technologies

C++ & Vulkan

When choosing a framework for a fluid simulation using SPH most papers use C++ in combination with OpenMP to achieve parallelization, other implementations involve Cuda to use the massive parallel compute performance of GPUs.(SRC) For rendering these simulations, dedicated renderers are used. In this implementation Vulkan is used to simulate and visualize the granular material on the GPU using a combination of compute shaders and the rasterization pipeline. This allows for massive parallel computation of the simulation together with the visualization in real time, giving maximum flexibility and extensibility by for example a raytraced visualisation. Vulkan uses SPIRV Shaders, which allows the use of any shading language, that can be compiled to it. I opted for GLSL for its well known C like structure. <https://www.vulkan.org/>

CMake

CMake is a popular choice in software development for its role as a cross-platform build system and project configuration tool, streamlining the build process, enhancing code portability, and offering flexibility in managing diverse project structures and dependencies. <https://cmake.org/>

Vulkan Memory Allocator

In difference to other Graphics APIs Vulkan requires the programmer to do the memory management on the GPU. Vulkan Memory Allocator (VMA) is employed to achieve efficient management and optimization of GPU memory, simplifying memory allocation and deallocation processes and improving overall performance. <https://gpuopen.com/vulkan-memory-allocator/>

shaderc

Shaderc is utilized for the compilation and linking of shaders due to its user-friendly interface and simplicity in handling these processes. This made is straight foward to compile the used GLSL shaders to SPIRV for Vulkan to use. <https://github.com/google/shaderc>

TinyGLTF & STB Image

In computer graphics, TinyGLTF is often used for seamless glTF asset loading, and STB Image is favored for its straightforward approach to handling and processing images, both known for their efficiency and ease of use across different applications. <https://github.com/syoyo/tinygltf>

Dear ImGui

Dear ImGui is a popular immediate mode graphical user interface (GUI) library. It simplifies the process of creating and integrating user interfaces into applications by providing an intuitive and efficient framework and has implementations for nearly every common graphics API and windowing library. It also supports features like multiple viewports and docking elements to them. <https://github.com/ocornut/imgui>

GLFW

GLFW is a widely-used windowing library that simplifies the creation and management of windows, handling user input and facilitating cross-platform development for OpenGL- and Vulkan-based applications. <https://www.glfw.org/>

glm

GLM is a versatile C++ mathematics library designed for graphics programming, providing a comprehensive set of functions and classes for vector and matrix operations commonly used in computer graphics. <https://github.com/g-truc/glm>

TriangleMeshDistance

TriangleMeshDistance offers a user-friendly programming interface for effortlessly creating signed distance fields of adjustable resolution from 3D meshes. <https://github.com/InteractiveComputerGraphics/TriangleMeshDistance>

5 Implementation

...

Algorithm 1 Full Simulation Frame

```

1: find neighbors
2: compute  $\rho$  and  $F^{drag}$ 
3: compute  $s$ 
4: compute  $v$  advection
5: compute  $\rho$  advection
6:  $l = 0$ 
7: while  $l < 2 \parallel \rho_{avg} - \rho_0 < \eta$  do
8:   compute  $d_{ij}p_j$ 
9:   compute  $p^{l+1}$ 
10:   $p(t) = p^{l+1}$ 
11:  apply yield criterion
12:   $l = l + 1$ 
13: end while
14: compute  $F^p$  and  $F^f$ 
15: integrate
16: advect HR particles

```

5.1 Neighborhood Search

Using the SPH approximation involves finding all neighboring particles within the smoothing length h . In a naive implementation this means iterating all other particles and check their distance to each other. With a runtime of $O(n^2)$ this is not an option for a realtime scenario. To improve the neighborhood search a number of algorithms could be employed. In this implementation a grid based approach has been used, where every particle is first segmented into a spatial grid with cell size of h . This provides the possibility to find every neighbor of a given particle, by finding its cell and iterating over its members and all adjacent cells. This dramatically decreases the amount of particles that have to be checked every frame. To get a list of particles for each cell, all grid entries, consisting of a cell key and a particle ID, are first sorted by the cell key. As this sorting has to be done at the beginning of every frame, this has to be done as fast as possible. To achieve this, a parallel sorting algorithm, such as radix sort or in this case bitonic merge sort is used. After the list of grid entries has been sorted, the start of every cells entries has to be found, by checking if the cell key of an entry and its predecessor are the same. The starting index of the cell is then stored in an array of length of n , being the particle count, at the position of the cellkey. This implies that all cell keys have to be within the range 0 to $(n - 1)$. When

iterating over a given cells entries, the entries particle ID is used to check if the particle is within the smoothing length. A cell is found by calculating a particles cell key an the keys of its neighboring cells, and looking up the starting index of that cells entries. The entries can then be iterated until the cell key changes. In the below examples a list of 8 grid entries is shown, first in their unsorted state, then after they have been sorted and lastly the list of starting indices of each cell, where -1 denotes entries where no cell starts. In the actual implementation unsigned integers are used to represent the starting indices and the maximum unsigned integer is used for empty cells.

Tabelle 1: Example of unsorted list of grid entries

Cell key	2	7	5	3	7	7	5	2
Particle ID	0	1	2	3	4	5	6	7

Tabelle 2: Example of sorted list of grid entries

Cell key	2	2	3	5	5	7	7	7
Particle ID	0	7	3	2	6	1	4	5

Tabelle 3: Example of list of starting indices

Starting indices	-1	-1	0	2	-1	3	-1	4
------------------	----	----	---	---	----	---	----	---

This approach allows the grid to have an undefined size as only the amount of particles matters for the creation of it.

5.2 Density

When implementing a SPH simulation the first thing to do is calculate the density ρ of a particle. It is determined by a sum over the neighboring particles. The volume V of each particle within the smoothing length h is multiplied with the density of it and the weighting kernel $W_{ij} \equiv W(x_i - x_j, h)$. As the volume can be expressed as $V = \frac{m}{\rho}$ we can simplify the equation to only the sum of weighted masses. (SRC)

$$\rho_i = \sum_j V_j \rho_j W_{ij} = \sum_j m_j W_{ij} \quad (5)$$

5.3 Unilateral incompressibility

The next step of the simulation is to compute a pressure that compensates the difference in density between particles. The density and pressure underlie

the principle of unilateral incompressibility, which can be expressed in the form:

$$\rho_i \leq \rho_0 \perp p \geq 0 \quad (6)$$

This says that the density of a given particle i can not be greater than the rest density of the material and the pressure can not become negative.

5.4 External forces

A change in density can be achieved by applying external forces to the particles.

5.5 Gravity

Gravity is given as a fixed acceleration in a given direction in 3D space. Multiplied with the weight of a given particle we get an actual force.

5.6 Drag

As an additional force that approximated the interaction with air a drag force F^{drag} was applied. Which uses a simplified implementation of the drag by (SRC). For this the relative velocity difference of a particle to the surrounding air is computed and its length squared. (SRC)

$$v_{i,rel}^2 = |v_a - v_i|^2 \frac{v_a - v_i}{|v_a - v_i|} \quad (7)$$

The velocity of the air v_a can be set to 0 or could be taken from a velocity field of some kind. Computing the drag force, to apply to each particle, involves determining the crosssectional area A_i of a particle. This can be represented by the area of a circle. Then the actual area exposed to oncoming air needs to be determined. For this a cone which is defined by the direction of the relative velocity difference and a set angle is used to check if there are any other particles that shield the current particle from the air. This is done by a dot product of the relative velocity difference and the vector to the neighboring particles, which returns the angle between them. To not only get a boolean value for the shielding, an occlusion value is introduced, which is set to the smallest angle between neighboring particles in the cone. This value is then subtracted from 1 and added to the drag formulation as weight ω . Further parameters for the equation are ρ_a the density of the air or the medium that surrounds the particles and $C_{D,i}$ the drag coefficient for shape of the particle. The whole formula is then composed as:

$$F_i^{drag} = \frac{1}{2} \rho_a v_{i,rel}^2 C_{D,i} \omega A_i \quad (8)$$

The deformation of a particle by the surrounding air is left out in this implementation.

5.7 Strain & Stress

To simulate friction between the particles of the granular material we first need to calculate the strain ε , which measures the deformation of a material under external forces and is represented as a tensor of rank 3. This involves determining the velocity gradient ∇u_i by an outer product with the kernel gradient ∇W_{ij} .

$$\nabla u_i = \sum_j V_j \nabla W_{ij} u_j^T \quad (9)$$

The strain is then computed as:

$$\varepsilon = \frac{1}{2}(\nabla u_i + \nabla u_i^T) \quad (10)$$

What is stress... For the computation of the frictional stress a parameter D , which relates the frictional stress s to the dissipation of strain, is calculated as:

$$D_i = \frac{2m_i^2 \Delta t}{\rho_i^2} \sum_j \frac{1}{\rho_j} \nabla W_{ij} \nabla W_{ij}^T \quad (11)$$

The parameter can be precomputed for a prototype particle with a filled neighborhood, to make the computation more efficient. The stress is obtained by multiplying the strain tensor with the inverse of D .

$$s_i = D^{-1} \varepsilon \quad (12)$$

For most dry granular materials there is no or neglectable cohesion (The tendency of particles to stick to each other). This means that we can subtract the mean hydrostatic stress $s_{i,hydrostatic}$ from the stress computed above, enforcing a traceless deviatoric stress $s_{i,deviatoric}$, which will further be used as s_i .

$$s_{i,hydrostatic} = \frac{1}{2} \text{Tr } s_i \quad (13)$$

$$s_{i,deviatoric} = s_i - s_{i,hydrostatic} \quad (14)$$

Cohesion can be handled similarly to friction. (...)

The amount of friction that can be applied is limited by the pressure. This is expressed by the Drucker-Prager yield criterion:

$$\|s_i\| \leq p_i \alpha \quad (15)$$

$$\alpha = \sqrt{2} \sin \Theta \quad (16)$$

where α is the frictional coefficient for an angle of repose θ (the maximum angle a material can pile before slipping) and $\|s_i\|$ the frobenius norm of s_i . The yield constraint can be approximated in a piecewise linear manner by applying it to each component of s

5.8 Pressure

In traditional SPH methods the pressure p is calculated by an equation of state like the ideal gas equation

$$p_i = k * (\rho_i - \rho_0) \quad (17)$$

or Tait's equation

$$p_i = \frac{\rho_0 * B}{n} n \frac{\rho_i}{\rho_0} \quad (18)$$

where k , B and n are parameters controlling the stiffness of the pressure calculation, which means the scaling of a density error and the resulting pressure. PCISPH then tries to minimize the mean density error by doing multiple iterations, correcting the pressure to apply. As WCSPH does not provide a good enough incompressibility for stable friction calculations and PCISPH can take a lot of iterations to converge, I opted for IISPH as it provides pressures that result in a very small mean density error, after only few iterations and supports larger timesteps as other variants of SPH.

The IISPH implemetation consist of two parts. The first being the advection of the velocities and densities and the second, the iterations to solve for the pressure. For the advection all external forces are summed up and the particle velocities are advanced in time to get an intermediate velocity v_i^{adv} . [20]

$$F_i^{adv} = F^g + F_i^{drag} \quad (19)$$

$$v_i^{adv} = v_i + \Delta t \frac{F_i^{adv}}{m_i} \quad (20)$$

The intermediate velocity is then used to calculate predicted densities ρ_i^{adv} for all particles. [20]

$$\rho_i^{adv} = \rho_i + \Delta t \sum_j m_j v_{ij}^{adv} \nabla W_{ij} \quad (21)$$

// TODO: Pressure calculation pressure

$$p_i^{l+1} = (1 - \omega) p_i^l + \omega \frac{1}{a_{ii} * \Delta t^2} (\rho_0 - \rho_i^{adv} - \Delta t^2 \psi) \quad (22)$$

$\omega = 0.5$

$$\psi = \sum_j m_j (\sum_j d_{ij} p_j^l - d_{jj} p_j^l - \sum_{k \neq i} d_{jk} p_k^l) \nabla W_{ij} \quad (23)$$

$$\sum_{k \neq i} d_{jk} p_k^l = \sum_k d_{jk} p_k^l - d_{ji} p_i^l \quad (24)$$

$$\rho_i^{l+1} = |p * a_{ii} * \Delta t^2 - (\rho_0 - \rho_i^{adv} - \Delta t^2 \psi)| + \rho_0 \quad (25)$$

$$a_{ii} = \sum_j m_j (d_{ii} - d_{ji}) \nabla W_{ij} \quad (26)$$

5.9 Internal Forces

For the calculation of the pressure force of a given particle the SPH concept is applied.

$$F_i^p = -m_i \sum_{j \neq i} m_j \left(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2} \right) \nabla W_{ij} \quad (27)$$

A similar formulation can be used to determine the frictional force of that particle.

$$F_i^f = -m_i \sum_{j \neq i} m_j \left(\frac{s_i}{\rho_i^2} + \frac{s_j}{\rho_j^2} \right) \nabla W_{ij} \quad (28)$$

5.10 Integration

For the integration of the simulation a simple Euler integration has been used. It could be exchanged for a more advanced integrations scheme such as Leap-Frog or Verlet.

$$v_i(t + \Delta t) = v_i^{adv} + \Delta t \frac{F_i^p}{m_i} \quad (29)$$

$$x_i(t + \Delta t) = x_i + \Delta t v_i(t + \Delta t) \quad (30)$$

5.11 Boundary Handling

5.11.1 Particle-Based Approaches

Most implementations of SPH use the concept of AK et al. when implementing boundary handling and rigidbody interactions. The idea is to represent boundaries and rigidbodies by sampling their surface or volume with a dense mesh of particles, which can then be incorporated into the normal simulation. For static objects the boundary particles are not integrated in time. For dynamic rigidbodies the particles representing them, collect the counteracting forces, they exert onto the fluid. The collected forces can then be integrated by a rigidbody physics solver. This method presents some drawbacks. One of them being the amount of particles required to sample the 3D models in the scene, which then have to be checked for each neighboring fluid particle.

5.11.2 Volume Maps

In this implementation the concept of Volume Maps is employed. They are an implicit boundary representation, inspired by the density maps concept. They use signed distance functions to determine the volume of the intersection of the kernel function around a particle and the boundary. For this, the boundary or bounding box of any arbitrary mesh is extended by the kernel radius and then sampled on a grid with a set resolution. Here the Triangle-MeshDistance library is used to create a signed distance representation of any triangle based mesh. For each sample a vector to the nearest point on the surface of the object is stored aswell as the intersection volume V_B , which is calculated by the signed distance Φ , treated by a cubic extension function γ^* , to get a continuous differentiable function, which results in a smooth transition at $x = h$. [10]

$$V_b(x) = \int_{N(x)} \gamma^*(\Phi(x')) dx' \quad (31)$$

$$\gamma^*(x) = \begin{cases} \frac{C(x)}{C(0)}, & \text{if } 0 < x < h \\ 1, & \text{if } x \leq 0 \\ 0, & \text{otherwise} \end{cases} \quad (32)$$

For this cubic extension function the cubic spline kernel $C(x)$ [19] is used. The use of this function does not enforce the use of the same function for the SPH approximation, where any kind of kernel function can be used. [10] The grid is then stored as a 32-bit 4-channel floating point 3-dimensional image and uploaded to the GPU. As Vulkan does not support arrays of 3D images, as it does for 2D, the descriptor indexing extension has to be used to allow for multiple 3D textures being bound dynamically. When sampling the texture in the shader, the nearest position on the surface and the intersection volume can be determined using only one texture read access instead of iterating all neighboring boundary particles as done in the particle-based approach. After transforming the volume map from its texture space into world space inside of the shader, a smooth value for both required properties is can be determined by the sampler. These values can the be used in any of the above mentioned equations, by substituting the mass for the determined volume V_b and the reference density of the simulated material ρ_0 .

$$\rho_i = \sum_b V_b \rho_0 W_{ib} \quad (33)$$

$$F_i^f = -m_i \sum_b V_b \rho_0 \left(\frac{s_i}{\rho_i^2}\right) \nabla W_{ib} \quad (34)$$

$$F_i^p = -m_i \sum_b V_b \rho_0 \left(\frac{p_i}{\rho_i^2}\right) \nabla W_{ib} \quad (35)$$

For the calculation of the resulting boundary forces only the pressure and stress of the particle is used, other than in the calculation of the forces between particles, where both particles contribute to the force. At this point the boundary forces can be applied to the actual rigidbody the volume map represents, to allow for two way coupling of the simulation and the boundaries.

5.12 Upsampling

As granular matter consists of vast number of grains, simulating them all as individual particles, would not only make the computational cost and memory consumption incredibly high, but would also result in a very restricted time step due to the CFL condition. To achieve real time results, the simulation is done for a set of low-resolution (LR) particles which approximate the granular flow, which are then upsampled to another set of high-resolution (HR) particles. [1]

// TODO: Sampling

After the above described algorithm is executed on the low-resolution particles, the high resolution particles are advected based on the velocity field of the low-resolution simulation. This is again done by the SPH concept, but with a different smoothing length h_{HR} which is set to $h_{HR} = 3h_{LR}$. To find the velocity of a HR particle a distance based weight w is introduced.

$$w(d_{ij}) = \max(0, (1 - \frac{d_{ij}^2}{h_{HR}^2})^3) \quad (36)$$

In this equation i refers to a high-resolution particle and j to a low-resolution particle or a boundary and d_{ij} to the distance between them. The velocity is computed as the weighted average velocity over the neighboring LR particles.[1]

$$v_i^*(t + \Delta t) = \frac{1}{\sum_j w(d_{ij})} \sum_j w(d_{ij}) v_j \quad (37)$$

As the LR particles should be able to not only follow the flow of the granular material, but react to gravity when there are no LR particles in its range, a blending weight α is introduced, to define the contributions of the simulation and external forces. The final velocity of each HR particle is then computed as:

$$v_i(t + \Delta t) = (1 - \alpha) v_i^*(t + \Delta t) + \alpha (v(t) + \Delta t \frac{F^g}{m}) \quad (38)$$

where α is:

$$\alpha = \begin{cases} 1 - \arg \max_j w(d_{ij}), & \text{if } \frac{\arg \max_j w(d_{ij})}{\sum_j w(d_{ij})} \geq 0.6 \\ 1 - \arg \max_j w(d_{ij}), & \text{if } \arg \max_j w(d_{ij}) \leq w(r_{LR}) \\ 0, & \text{otherwise} \end{cases} \quad (39)$$

The constant 0.6 is an empirically tested value for the above defined high-resolution smoothing length. [1]

5.13 Visualization

...

6 Evaluation

...

6.1 Performance

...

7 Conclusion & Future Work

...

Literatur

- [1] M. Ihmsen, A. Wahl und M. Teschner, „High-Resolution Simulation of Granular Material with SPH,“ in *Workshop on Virtual Reality Interaction and Physical Simulation*, J. Bender, A. Kuijper, D. W. Fellner und E. Guerin, Hrsg., The Eurographics Association, 2012, ISBN: 978-3-905673-96-8. DOI: 10.2312/PE/vriphys/vriphys12/053-060.
- [2] I. Alduan, Á. Tena und M. A. Otaduy, „Simulation of High-Resolution Granular Media,“ in *CEIG 09 - Congreso Espanol de Informatica Grafica*, C. Andujar und J. Lluch, Hrsg., The Eurographics Association, 2009, ISBN: 978-3-905673-72-2. DOI: 10.2312/LocalChapterEvents/CEIG/CEIG09/011-018.
- [3] M. Ihmsen, N. Akinci, M. Gissler und M. Teschner, „Boundary Handling and Adaptive Time-stepping for PCISPH,“ Jan. 2010, S. 79–88. DOI: 10.2312/PE/vriphys/vriphys10/079-088.
- [4] N. Bell, Y. Yu und P. J. Mucha, „Particle-Based Simulation of Granular Materials,“ in *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, Ser. SCA '05, Los Angeles, California: Association for Computing Machinery, 2005, S. 77–86, ISBN: 1595931988. DOI: 10.1145/1073368.1073379. Adresse: <https://doi.org/10.1145/1073368.1073379>.
- [5] I. Alduán und M. A. Otaduy, „SPH Granular Flow with Friction and Cohesion,“ in *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, Ser. SCA '11, Vancouver, British Columbia, Canada: Association for Computing Machinery, 2011, S. 25–32, ISBN: 9781450309233. DOI: 10.1145/2019406.2019410. Adresse: <https://doi.org/10.1145/2019406.2019410>.
- [6] B. Solenthaler und R. Pajarola, „Predictive-Corrective Incompressible SPH,“ in *ACM SIGGRAPH 2009 Papers*, Ser. SIGGRAPH '09, New Orleans, Louisiana: Association for Computing Machinery, 2009, ISBN: 9781605587264. DOI: 10.1145/1576246.1531346. Adresse: <https://doi.org/10.1145/1576246.1531346>.
- [7] R. Narain, A. Golas und M. C. Lin, „Free-Flowing Granular Materials with Two-Way Solid Coupling,“ in *ACM SIGGRAPH Asia 2010 Papers*, Ser. SIGGRAPH ASIA '10, Seoul, South Korea: Association for Computing Machinery, 2010, ISBN: 9781450304399. DOI: 10.1145/1866158.1866195. Adresse: <https://doi.org/10.1145/1866158.1866195>.
- [8] C. Gissler, S. Band, A. Peer, M. Ihmsen und M. Teschner, „Generalized Drag Force for Particle-Based Simulations,“ *Comput. Graph.*, Jg. 69, Nr. C, S. 1–11, Dez. 2017, ISSN: 0097-8493. DOI: 10.1016/j.cag.2017.09.002. Adresse: <https://doi.org/10.1016/j.cag.2017.09.002>.

- [9] T. Probst und M. Teschner, „Monolithic Friction and Contact Handling for Rigid Bodies and Fluids Using SPH,“ *Computer Graphics Forum*, Jg. 42, Jan. 2023. DOI: 10.1111/cgf.14727.
- [10] J. Bender, T. Kugelstadt, M. Weiler und D. Koschier, „Volume Maps: An Implicit Boundary Representation for SPH,“ in *Proceedings of the 12th ACM SIGGRAPH Conference on Motion, Interaction and Games*, Ser. MIG '19, Newcastle upon Tyne, United Kingdom: Association for Computing Machinery, 2019, ISBN: 9781450369947. DOI: 10.1145/3359566.3360077. Adresse: <https://doi.org/10.1145/3359566.3360077>.
- [11] D. Koschier, J. Bender, B. Solenthaler und M. Teschner, „A Survey on SPH Methods in Computer Graphics,“ *Computer Graphics Forum*, Jg. 41, S. 737–760, Mai 2022. DOI: 10.1111/cgf.14508.
- [12] Z.-B. Wang, R. Chen, H. Wang, Q. Liao, X. Zhu und S.-Z. Li, „An overview of smoothed particle hydrodynamics for simulating multiphase flow,“ *Applied Mathematical Modelling*, Jg. 40, Nr. 23, S. 9625–9655, 2016, ISSN: 0307-904X. DOI: <https://doi.org/10.1016/j.apm.2016.06.030>. Adresse: <https://www.sciencedirect.com/science/article/pii/S0307904X16303419>.
- [13] S. Band, C. Gissler, M. Ihmsen, J. Cornelis, A. Peer und M. Teschner, „Pressure Boundaries for Implicit Incompressible SPH,“ *ACM Trans. Graph.*, Jg. 37, Nr. 2, Feb. 2018, ISSN: 0730-0301. DOI: 10.1145/3180486. Adresse: <https://doi.org/10.1145/3180486>.
- [14] M. Fujisawa und K. T. Miura, „An Efficient Boundary Handling with a Modified Density Calculation for SPH,“ *Computer Graphics Forum*, Jg. 34, Nr. 7, S. 155–162, 2015. DOI: <https://doi.org/10.1111/cgf.12754>. Adresse: <https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.12754>.
- [15] T. Takahashi, Y. Dobashi, T. Nishita und M. C. Lin, „An Efficient Hybrid Incompressible SPH Solver with Interface Handling for Boundary Conditions,“ *Computer Graphics Forum*, 2018, ISSN: 1467-8659. DOI: 10.1111/cgf.13292.
- [16] M. Macklin, M. Müller, N. Chentanez und T.-Y. Kim, „Unified Particle Physics for Real-Time Applications,“ *ACM Trans. Graph.*, Jg. 33, Nr. 4, Juli 2014, ISSN: 0730-0301. DOI: 10.1145/2601097.2601152. Adresse: <https://doi.org/10.1145/2601097.2601152>.
- [17] M. Weiler, D. Koschier, M. Brand und J. Bender, „A Physically Consistent Implicit Viscosity Solver for SPH Fluids,“ *Computer Graphics Forum*, Jg. 37, Nr. 2, S. 145–155, 2018. DOI: <https://doi.org/10.1111/cgf.13349>. Adresse: <https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.13349>.

- [18] A. Sommer, U. Schwanke und E. Schoemer, „Interactive High-Resolution Simulation of Granular Material,“ *Journal of WSCG*, Jg. 30, Nr. 1-2, S. 9–15, 2022, ISSN: 1213-6964. DOI: 10.24132/jwscg.2022.2. Adresse: <http://dx.doi.org/10.24132/JWSCG.2022.2>.
- [19] J. J. Monaghan, „Smoothed Particle Hydrodynamics,“ *Annual Review of Astronomy and Astrophysics*, Jg. 30, Nr. 1, S. 543–574, 1992. DOI: 10.1146/annurev.aa.30.090192.002551. Adresse: <https://doi.org/10.1146/annurev.aa.30.090192.002551>.
- [20] M. Ihmsen, J. Cornelis, B. Solenthaler, C. Horvath und M. Teschner, „Implicit Incompressible SPH,“ *IEEE Transactions on Visualization and Computer Graphics*, Jg. 20, Nr. 3, S. 426–435, 2014. DOI: 10.1109/TVCG.2013.105.
- [21] I. Quilez, „Inigo Quilez :: computer graphics, mathematics, shaders, fractals, demoscene and more,“ Adresse: <https://iquilezles.org/articles/distfunctions/>.
- [22] I. Quilez, „Inigo Quilez :: computer graphics, mathematics, shaders, fractals, demoscene and more,“ Adresse: <https://iquilezles.org/articles/distfunctions2d/>.
- [23] @tgfrerer, „Implementing Bitonic Merge Sort in Vulkan Compute,“ Adresse: https://poniesandlight.co.uk/reflect/bitonic_merge_sort/.