Fast GPU-based Fluid Simulations Using SPH

Øystein E. Krog and Anne C. Elster

Dept. of Computer and Information Science
Norwegian University of Science and Technology (NTNU)
oystein.krog@gmail.com, elster@idi.ntnu.no

Abstract. Graphical Processing Units (GPUs) are massive floating-point stream processors, and through the recent development of tools such as CUDA and OpenCL it has become possible to fully utilize them for scientific computing. We have developed an open-source CUDA-based acceleration framework for 3D Computational Fluid Dynamics (CFD) using Smoothed Particle Hydrodynamics (SPH). This paper describes the methods used in our framework and compares the performance of the implementation to previous SPH implementations. We implement two different SPH models, a simplified model for Newtonian fluids, and a complex model for Non-Newtonian fluids, which we use for simulation of snow avalanches. Having implemented two different models, we investigate the performance characteristics of SPH simulations on the GPU and find that despite the larger bandwidth-requirements of the complex model the GPU scales well. Our simulations are rendered interactively and in real-time. Using an NVIDIA GeForce GTX 470 Fermi-based card we achieve 215.4, 122.2 and 64.9 FPS for the simple model and 69.6, 37.4 and 19.1 FPS for 64K, 128K and 256K particles respectively.

Keywords: GPU, CFD, SPH, GPGPU, CUDA, Fluid, Newtonian, Non-Newtonian

1 Introduction

Simulating fluids is a computationally intensive problem, especially in 3D. Due to large computational demands most fluid simulations are not done in real-time. We have developed a new open-source[1] framework for 3D SPH calculations on the GPU, where we provide computational primitives that accelerate the building blocks of the SPH algorithm. By using our framework the number of particles modeled in the simulation can be increased considerably, and the overall simulation speed is greatly increased compared to CPU-based simulations. Our work differs from previous works in several ways. We provide a modularized framework that can be used for implementing different SPH models. We use an acceleration algorithm that is well-suited for the GPU and finally we compare two different SPH models implemented using the same framework, thus giving a measure of how well the GPU scales with more complex models.

[1] http://code.google.com/p/gpusphsim/

2 Previous Work

Some of first implementations of SPH on the GPU were by Harada *et al.*[4] and Zhang *et al.*[15]. This was before the introduction of CUDA and it was thus done using OpenGL and Cg which imposed severe limits on the implementations. Since then there has been growing interest in the implementation of SPH on the GPU resulting in several implementations that take full or partial advantage of the GPU and CUDA [14][2][5].

3 Computational Fluid Dynamics

Fluid dynamics is described using the the Navier-Stokes equations, and in their Lagrangian form consist of mass and momentum conservation:

$$\frac{d\rho}{dt} = -\rho\nabla \cdot \mathbf{v} \tag{1}$$

$$\frac{d\mathbf{v}}{dt} = -\frac{1}{\rho}\nabla p + \frac{1}{\rho}\nabla \cdot \mathbf{S} + \mathbf{f} \tag{2}$$

Where $\mathbf{v}$ is the velocity field, ρ the density field, ∇p the pressure gradient field resulting from isotropic stress, $\nabla \cdot \mathbf{S}$ the stress tensor resulting from deviatoric stress and $\mathbf{f}$ an external force field such as gravity. For incompressible Newtonian fluids the momentum conservation reduces to:

$$\frac{d\mathbf{v}}{dt} = -\frac{1}{\rho}\nabla p + \frac{\mu}{\rho}\nabla^2\mathbf{v} + \mathbf{f} \tag{3}$$

Where the term μ is the dynamic viscosity of the fluid.

3.1 Smoothed Particle Hydrodynamics

In SPH the different effects of Navier-Stokes are simulated by a set of forces that act on each particle. These forces are given by scalar quantities that are interpolated at a position $\mathbf{r}$ by a weighted sum of contributions from all surrounding particles within a cutoff distance h in the space Ω. In integral form this can be expressed as follows [9]:

$$A_i(\mathbf{r}) = \int_\Omega A(\mathbf{r}')W(\mathbf{r} - \mathbf{r}', h)d\mathbf{r}' \tag{4}$$

The numerical equivalent is obtained by approximating the integral interpolant by a summation interpolant [9]:

$$A_i(\mathbf{r}_i) = \sum_j A_j \frac{m_j}{\rho_j} W(\mathbf{r}_{ij}, h) \tag{5}$$

where j iterates over all particles, m_j is the mass of particle j, $\mathbf{r}_{ij} = \mathbf{r}_i - \mathbf{r}_j$ where $\mathbf{r}$ is the position, ρ_j the density and A_j the scalar quantity at position $\mathbf{r}_j$.

For a more comprehensive introduction to SPH, please refer to [9].

3.2 Snow Avalanche SPH

Snow avalanches wary greatly in behavior, from powder-snow avalanches to so called dense-flow, or flowing snow avalanches. Snow avalanches can often appear as a viscous flow down a slope, and it is this obvious property which has prompted the use of fluid dynamics in avalanche simulation [1]. Several viscosity models exist for modeling Non-Newtonian fluids, and rheological parameters have been collected for flowing snow [7]. Many SPH models exist for viscoelastic fluids, from melting objects [11] to lava flows [12] and generalized rheological models [6]. We implement an SPH simulation of a Non-Newtonian fluid with configurable support for multiple rheological models to approximate the behavior of a snow avalanche.

4 Methods and Implementation

The simulation framework uses CUDA, a parallel computing architecture developed by NVIDIA. We parallelize the calculation of SPH by assigning a thread to each particle in the simulation. Each thread is then responsible for calculating the SPH sums over the surrounding particles.

When accessing memory on the GPU *coalesced* (correctly structured) access is very important. Due to the nature of the acceleration algorithm we use, perfectly coalesced access is unfortunately not possible. By utilizing the texture cache on the GPU this problem is greatly alleviated.

4.1 Nearest-Neighbor Search

The summation term in the SPH-formulation is computationally heavy, it requires looking at many nearby particles and computing interactions between them. To avoid a naive brute-force $O(N^2)$ search for neighbors, a nearest-neighbor search algorithm is commonly used, such as a linked list or a uniform grid. Our framework uses the acceleration algorithm found in the NVIDIA CUDA "Particles" demo [3], which is better suited to the GPU than many previously used algorithms. It can be summarized as follows:

1. Divide the simulation domain into a uniform grid.
2. Use the spatial position of each particle to find the cell it belongs to.
3. Use the particle cell position as input to a hash function (a spatial hash)
4. Sort the particles according to their spatial hash.
5. Reorder the particles in a linear buffer according to their hash value.

Particles in the same cell can then appear ordered in the linear buffer, the specifics depending on the spatial hash function. Finding neighbors is thus just a matter of iterating over the correct indices in the buffer. To sort the particles on the GPU we used a radix sort [13].

4.2 Non-Newtonian Fluids

Non-Newtonian fluids differ from Newtonian fluids in that their viscosity is not constant. In a Newtonian fluid the relation between shear stress and the strain rate is linear, with the constant of proportionality being the viscosity. For a Non-Newtonian fluid the relation is nonlinear and can even be time-dependent. There exist many classes of Non-Newtonian fluids, and many types of models, of which we implement several. The complex SPH model differs primarily from the simple SPH model in that it includes the much more complex stress calculation presented in [6] and in that the viscosity parameter is not constant but modeled using a rheological viscosity model.

4.3 SPH Models

We have implemented two different SPH models, a simple and a complex model. The simple SPH model is a partial implementation (discarding surface tension) of a well-known SPH model designed for interactivity [10]. In the complex SPH model we combine some of the techniques used in [10] with models from [6] and [12]. Compared to the Simple model we use a more accurate smoothing kernel, we use a more accurate calculation of shear forces and we support a range of rheological models which enable us to simulate different Non-Newtonian fluids.

4.4 SPH Formulation

By using the SPH formulation the Navier-Stokes equations can be approximated:

$$\rho_i = \sum_j m_j W(\mathbf{r}_{ij}, h) \tag{6}$$

$$\mathbf{f}_i^{pressure} = -\frac{1}{\rho}\nabla p(\mathbf{r}_i) = \sum_{j \neq i} m_j (\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2}) \nabla W(\mathbf{r}_{ij}, h) \tag{7}$$

The incompressible fluid is simulated as a weakly compressible fluid where the incompressibility constraint is applied to the pressure p by using an equation of state given by the ideal gas law with an added rest density[4]: $p = k(\rho - \rho_0)$

Finally the stress force is calculated:

$$\mathbf{f}_i^{stress} = \frac{1}{\rho}\nabla \cdot \mathbf{S}_i = \sum_{j \neq i} \frac{m_j}{\rho_i \rho_j} (\mathbf{S}_i + \mathbf{S}_j) \cdot \nabla W(\mathbf{r}_{ij}, h) \tag{8}$$

Where the Non-Newtonian fluid stress tensor $\mathbf{S}$ is calculated as [6]. Thus we have that the acceleration for a particle is given by:

$$\mathbf{a}_i = f_i^{pressure} + f_i^{stress} + f_i^{external} \tag{9}$$

4.5 SPH Algorithm

Due to the data dependencies between the various forces, some of them must be calculated separately. Each calculation step is essentially a summation over neighboring particles, and we combine the force calculations using loop fusion as far as it is possible. For the complex SPH model we end up with the following steps:

1. Update the hashed, radix sorted, uniform grid.
2. Calculate the SPH density
3. Calculate the SPH velocity tensor
4. Calculate the SPH pressure and SPH stress tensor
5. Apply external forces and integrate in time using Leap-Frog.

For the simplified SPH model, the stress tensor is replaced with a simplified viscosity approximation. Thus viscosity force can be computed together with the pressure in step 4 and as a result the simple SPH model can drop an entire SPH summation loop.

5 Results

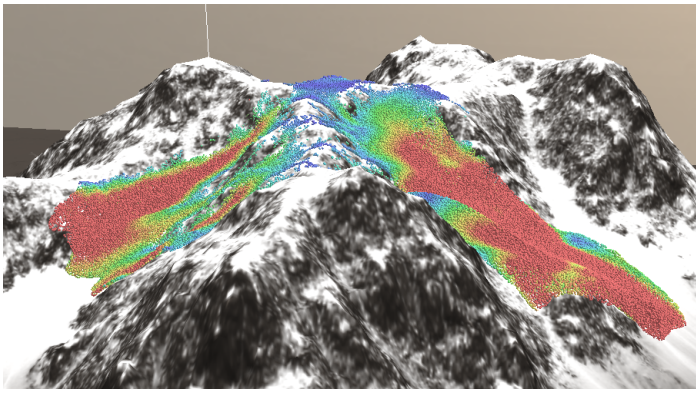


Fig. 1. A screenshot of the simple SPH model with 256K particles interacting with a terrain model. Hue-based gradient shading for the velocity of the particles.

Comparing and evaluating the performance of the simulations is difficult due to the large amount of parameters and their effect on performance. In addition it is hard to compare to other SPH implementations due to different SPH models and parameters. For the simple SPH model we compare against Mller and Harada which use a very similar SPH model, using rest density selected to simulate water, with the dynamic viscosity set to 1. For the complex SPH model we have not

found comparable implementations so must compare against our implementation of the simple SPH model. To obtain absolute performance numbers we use a fairly simple simulation setup; a square simulation domain with simple repulsive forces as walls where a cubic volume of fluid is dropped into a shallow pool of water Figure 2. The performance numbers were measured when the fluid had reached a stable equilibrium to avoid errors introduced by fluctuations in simulation performance.

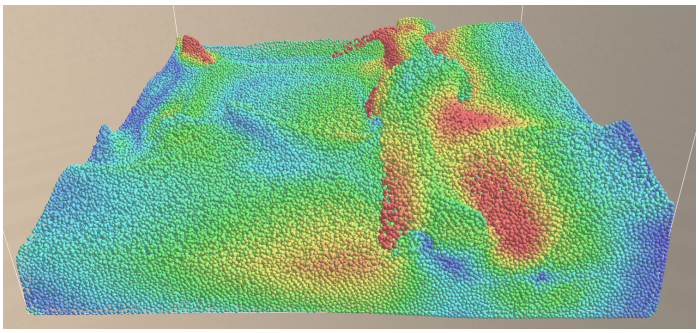


Fig. 2. A screenshot of the simple SPH model with 256K particles in a box of repulsive forces. Hue-based gradient shading for the velocity of the particles.

Hardware For all our performance results we used a fairly high-end computer equipped with an Intel Core2 Quad Q9550 processor. We compare three different graphics cards, an NVIDIA GeForce GTX 260, a GeForce GTX 470 and a Tesla C2050, thus covering both the previous and the current generation of GPU processors.

Simulation Parameters The simulation parameters were chosen for their stability and how realistic the resultant behavior fluid appeared. To improve the consistency and validity of the results we chose to use a simple Newtonian fluid rheology for performance measurements of the complex SPH model as well, which eliminates the complexity of measuring the performance of a model with a variable viscosity, while maintaining the computational complexity of the model.

We select a time step (dt) of 0.0005, and a rest density (ρ_0) of 1000. In addition we employ a simple aspring-like external boundary force with a stiffness and dampening coefficients of 20000 and 256 respectively. Finally we set a viscosity (μ) of 1.0. Our simulation is scaled with a factor of 0.0005.

Memory Usage Since our implementations do not use constant or shared memory to any significant degree, the only memory usage that is of importance is the usage of the global memory on the device. Due to the hashed uniform grid

structure the memory usage is highly efficient, since the particle data is stored in continuous memory buffers that are fully utilized. For the Simple SPH model the memory usage is $176N$ bytes where N is the number of particles. The memory usage of the complex SPH model is $240N$ bytes. This means that it is possible to simulate very large systems even on commodity hardware. Using the NVIDIA Tesla C2050, we have simulated up to 12 million particles with the Simple SPH model, thus using roughly 2GB of memory.

5.1 Performance Results

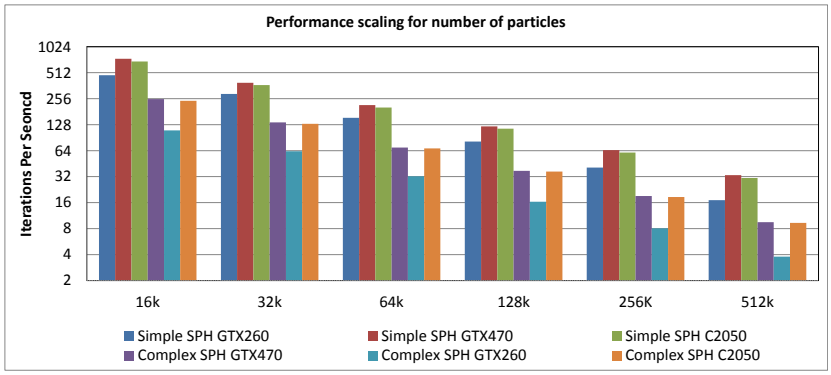


Fig. 3. Performance scaling for the simple SPH model.

By manually optimizing register usage, reordering memory accesses and optimizing the block sizes for the CUDA code, performance gains as large as 40% over our earlier implementation were realized [8].

The performance of the Tesla C2050 is nearly identical to that of the GeForce GTX 470. The small difference in performance can most likely be attributed to the difference in memory speed (1500 MHz vs. 1674 MHz) and clock speed (1150 MHz vs. 1215 MHz). Our implementation is single precision only, so we can not utilize the greatest feature of the Tesla; greatly increased double precision performance.

It is interesting to note that though the Tesla has greater memory bandwidth due to the larger memory bus (384 bit vs. 320 bit), this does not seem to increase performance. This could be due to the slightly lower clocks of the Tesla or it may mean that the raw memory bandwidth is less important than the performance of the texture cache. Since the memory access pattern of the hashed uniform grid algorithm cannot be coalesced perfectly there will be uncoalesced memory access. In the context of this knowledge, the missing performance increase may mean that the performance is severely bottlenecked by these uncoalesced memory accesses, much in the same manner as a pipeline stall.

Access to the global memory on the GPU is not perfectly coalesced due to limitations of the GPU acceleration algorithm, however this limitation is greatly mitigated by use of the texture cache. The texture cache was found to increase performance between 55% and 200% [8].

5.2 NVIDIA Fermi architecture

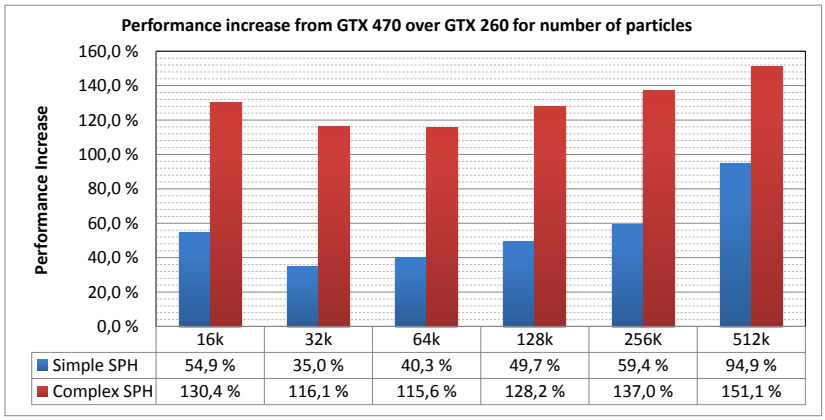


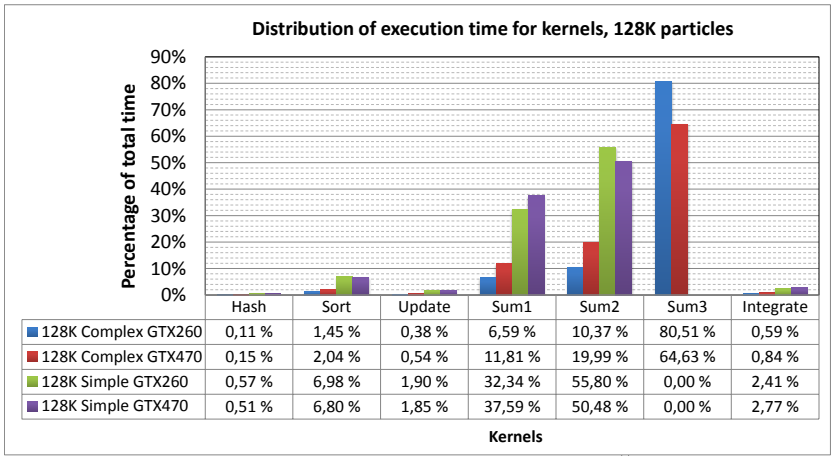
Fig. 4. Performance comparison between the GeForce GTX 470 and the GeForce GTX 260 for different amount of particles.

The new GF100/Fermi-architecture provides a large increase in performance compared to the GT200-architecture.

In Figure 4 we show the performance increase from using a GeForce GTX 470 over a GeForce GTX 260. These two GPUs have memory bandwidths of 111.9 GiB/s and 133.9 GiB/s, an increase of 20%, but the increase in measured performance is much larger, clearly indicating that the implementations are not completely bandwidth bound. The complex SPH model benefits the most, with improvements up to 150%, while the Simple SPH model sees improvements up to 95%. This large increase in performance can primarily be attributed to the new features of the Fermi architecture, which allows for greater utilization of the GPU resources (occupancy) and which minimizes the performance penalty of imperfectly coalesced memory access.

5.3 Kernels

We have measured the relative performance of the different kernels in our two SPH implementations. Our findings show that the most performance intensive parts are in the calculation of the SPH summation over neighboring particles. This is due to the large amount of computation and memory transfer (from global memory on the GPU) that occurs in these steps.



	Hash	Sort	Update	Sum1	Sum2	Sum3	Integrate
128K Complex GTX260	0,11 %	1,45 %	0,38 %	6,59 %	10,37 %	80,51 %	0,59 %
128K Complex GTX470	0,15 %	2,04 %	0,54 %	11,81 %	19,99 %	64,63 %	0,84 %
128K Simple GTX260	0,57 %	6,98 %	1,90 %	32,34 %	55,80 %	0,00 %	2,41 %
128K Simple GTX470	0,51 %	6,80 %	1,85 %	37,59 %	50,48 %	0,00 %	2,77 %

Fig. 5. Distribution of calculation time for the algorithm steps.

5.4 Rendering Overhead

We use direct rendering of the particle data using shaders on the GPU. This means that the rendering overhead is fairly small, though relatively large for small amounts of particles. With 16K particles the overhead is 95% and 60% for the simple model and 30% and 40% for the complex model, on the GT260 and the GTX470 respectively. For 256K particles this is reduced to 20% and 5% for the simple model and 8% and 3% for the complex model, on the GT260 and the GTX470 respectively.

5.5 Performance Review

We have compared our implementation performance for the Simple SPH model with that of other implementations. This algorithm has been widely implemented since it is very well suited for interactive or real-time simulation and as such it is possible to find comparable implementations. Unfortunately we have found that it is nearly impossible to do a review of earlier implementations that is both comprehensive and accurate since most authors do not specify all the parameters they use. In addition there are slight differences in the SPH models and finally also because of the different hardware used. Nonetheless we have attempted a comparison, if only to give a rough picture of the performance landscape.

We find that our GPU implementation is significantly faster than earlier GPU implementations, even for implementations using faster graphics cards. One such implementation[14] use a NVIDIA GTX 280 and get 66 iterations per second at 16K particles. Comparing their implementation against our implementation running on a GTX260 (without rendering) we see a 6x speedup. It is also interesting to note that our implementation seems to scale better, though the available data is not enough to draw any conclusions.

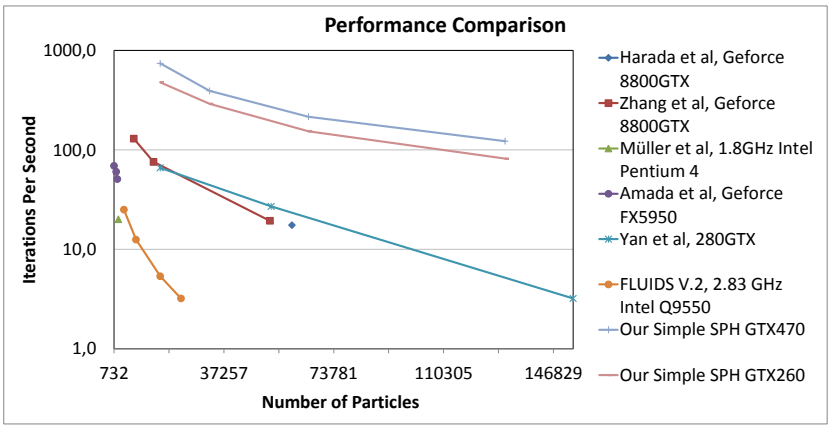


Fig. 6. Performance of our implementations compared to others.

Harada *et al.*[4] achieves real-time performance at 17 FPS with 60000 particles on an NVIDIA GeForce 8800GTX and Zhang *et al.*[15] achieves 56 FPS with 60000 particles using the same GPU. Both these implementations use OpenGL and Cg and are thus very constrained compared to more recent implementations using CUDA. A very interesting comparison is with the FLUIDS V.2 software, which is a highly optimized SPH implementation for the CPU. Unfortunately FLUIDS can only use one of the cores in this CPU so it should be assumed that the performance could be almost quadrupled using all 4 cores. Comparing the FLUIDS software with our GPU implementation (with rendering), we see speedups of 91x for the GeForce GTX 470 and 49x for the GeForce GTX 260 with 16K particles.

5.6 Real-Time Appearance

By scaling the simulation domain, and relaxing the accuracy requirements by selecting large time step, the fluid simulations produce beliveable real-time fluid animations. Our complex SPH model is not as well suited to real-time simulation due to the necessity of a somewhat lower timestep in order to support higher viscosities, but by using 64K particles it is still possible to simulate avalanche-like animations in real-time. We found that using a Cross rheological model best captured the behavior of a flowing snow avalanche (Figure 7).

6 Conclusions

In this paper, we presented an implementation of Smoothing Particle Hydrodynamics (SPH) on the GPU using. Our implementation achieves very good

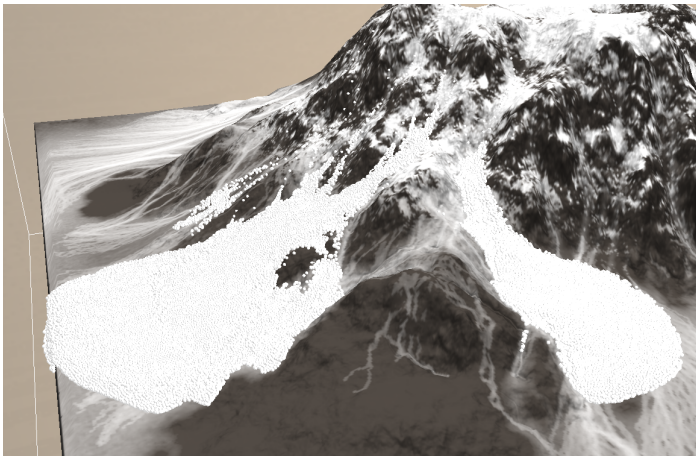


Fig. 7. A screenshot of the complex SPH model with 64K particles. We use a cross rheological model to approximate the behavior of a flowing snow avalanche.

performance since we take advantage of the massive amount of parallelism available on modern GPUs, as well as use specialized acceleration data structures. As a result of the computational acceleration afforded by the use of GPUs our simulations can maintain very high performance with large problem sizes. This produces real-time simulations whose animation appears more correct and realistic than previously seen efforts.

6.1 Current and Future Work

Simulations of snow can be used for everything from gaming to avalanche prediction. For games snow simulation can help create complex environments, which can lead to numerous possibilities for game-play mechanics. Predicting the behavior of snow avalanches can help prevent loss of both life and property. Our simulation framework can be used for more complex SPH models that can be used to produce qualitatively correct simulations.

The resource usage of our model has been investigated and it was found that it does not consume much memory but is very memory bandwidth intensive and suffers from imperfectly coalesced memory access, it would be interesting to research ways to improve the memory access pattern.

Finally, the visualization of the fluid model can be improved, at the moment a very simple but efficient and low-cost method of direct particle rendering is used. By using a surface reconstruction model such as Marching-Cubes a real surface can be rendered. It is also possible to use screen-space surface rendering techniques to approximate the fluid surface without the large computational cost associated with true surface reconstruction.

References

1. Bovet, E., Chiaia, B., Preziosi, L.: A new model for snow avalanche dynamics based on non-newtonian fluids. Meccanica (2008), http://www.springerlink.com/content/g82xk01766833788
2. Crespo, A.J.C.: Application of the Smoothed Particle Hydrodynamics model SPHysics to free-surface hydrodynamics. Ph.D. thesis, University of Vigo (2008)
3. Green, S., NVIDIA: CUDA Particles, Presentation slides. Tech. rep., NVIDIA (2008)
4. Harada, T., Koshizuka, S., Kawaguchi, Y.: Smoothed Particle Hydrodynamics on GPUs (2007), http://www.inf.ufrgs.br/cgi2007/cd_cgi/papers/harada.pdf
5. Herault, A., Bilotta, G., Dalrymple, R.A.: SPH on GPU with CUDA. Journal of Hydraulic Research 48(Extra Issue), 74–79 (2010)
6. Hosseini, S.M., Manzari, M.T., Hannani, S.K.: A fully explicit three-step SPH algorithm for simulation of non-Newtonian fluid flow. International Journal of Numerical Methods for Heat & Fluid Flow 17(7), 715–735 (2007), http://dx.doi.org/10.1108/09615530710777976
7. Kern, M.A., Tiefenbacher, F., McElwaine, J.N.: The rheology of snow in large chute flows. Cold Regions Science and Technology 39(2-3), 181 – 192 (2004), http://www.sciencedirect.com/science/article/B6V86-4CS4G7W-1/2/664993b41275bfb273c4b9b1d40cfd52
8. Krog, Ø.E.: GPU-based Real-Time Snow Avalanche Simulations. Master's thesis, NTNU (June 2010)
9. Liu, G.R., Liu, M.B.: Smoothed Particle Hydrodynamics: A Meshfree Particle Method. World Scientific Publishing Company (12 2003), http://amazon.com/o/ASIN/9812384561/
10. Mller, M., Charypar, D., Gross, M.: Particle-based fluid simulation for interactive applications. In: SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation. pp. 154–159. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland (2003)
11. Paiva, A., Petronetto, F., Lewiner, T., Tavares, G.: Particle-based non-newtonian fluid animation for melting objects. Computer Graphics and Image Processing, Brazilian Symposium on 0, 78–85 (2006)
12. Paiva, A., Petronetto, F., Lewiner, T., Tavares, G.: Particle-based viscoplastic fluid/solid simulation. Computer-Aided Design 41(4), 306 – 314 (2009), http://www.sciencedirect.com/science/article/B6TYR-4TTMNFW-1/2/3e798fdc322f7e878f386d435f80b01b, point-based Computational Techniques
13. Satish, N., Harris, M., Garland, M.: Designing efficient sorting algorithms for manycore GPUs. Tech. rep., NVIDIA Corporation, Los Alamitos, CA, USA (2009), http://dx.doi.org/10.1109/IPDPS.2009.5161005
14. Yan, H., Wang, Z., He, J., Chen, X., Wang, C., Peng, Q.: Real-time fluid simulation with adaptive SPH. Comput. Animat. Virtual Worlds 20, 417–426 (June 2009), http://portal.acm.org/citation.cfm?id=1568678.1568695
15. Zhang, Y., Solenthaler, B., Pajarola, R.: GPU accelerated SPH particle simulation and rendering. In: SIGGRAPH '07: ACM SIGGRAPH 2007 posters. p. 9. ACM, New York, NY, USA (2007)