Real-time particle simulation of fluids

Zsolt Horváth
Supervised by: Adam Herout

Faculty of Information Technology
Brno University of Technology
Brno / Czech Republic

Abstract

Physically plausible simulation of fluids in real-time is mostly achieved using approximations of the Navier-Stokes equations. Recent methods simulate fluids by exploiting the capabilities of modern graphics processing units. This article describes a method called Smoothed Particle Hydrodynamics (SPH), which is a numerical approximation of the Navier-Stokes equations. The real-time simulation allows for interactivity which is a great advantage against the offline methods. Offline methods are not running in real-time. The main goal of this project was to experiment with the Smoothed Particle Hydrodynamics running in realtime on the GPU.

Keywords: particle systems, fluid simulations, Navier-Stokes equations, smoothed particle hydrodynamics, CUDA, GPGPU, marching cubes

1 Introduction

Real-time simulation of fluids is a hot topic and major challenge in computer graphics. Fluids like liquids and gases are an important part of our life and environment. In real-time grapics we tradicionally try to reproduce a part of our world as visually realistic as possible, but unfortunately it is hard to simulate. Researchers concentrate on developing new, better and faster methods to simulate and visualize fluids. It is commonly said to be one of the hardest phenomenon to simulate realistically and even harder to simulate detailed fluids interactively. The offline methods which run not in real-time, can generate physically and visually better results, but with no user interaction, which is a disadvantage.

The mostly used method in computer graphics to simulate fluids is the Smoothed Particles Hydrodynamics (SPH) [6]. SPH is based on the Navier–Stokes equations. Because of the complexity of these equations, they can be solved only in simple cases. Generally, they are solved by a numerial method. The SPH method has good approximation which computes the most important properties of fluids like density, pressure and viscosity.

The aim of this work was to experiment with the SPH method on the CUDA platform. The experiments were mostly focused on achieving a simulation running in real time with different types of visualization methods and on speeding up the slowest parts of the implemented algorithms.

This work is structured as follows. In Section 2 we describe the present state of methods used in this area. Then the Section 3 and 4 present theoretical aspects. Section 5 contains important implementation details. The results are summarized in Section 6.

2 Related work

In 1822 Claude Navier and in 1845 George Stokes formulated the famous Navier-Stokes equations that describe the motion of the fluid substances [3]. With these equations which describe the conservation of momentum together with the two additional equations for the mass and energy conservation, it is possible to simulate the fluid flow.

Simulations apply numerical methods to solve the nonlinear partial differential equations. One common way to do this is to treat the fluid as a continuum, discretize the spacial domain into a grid, and use the finite differences or the finite volume method [16]. In the literature about the fluid simulations, the grid-based fluid models are called Eulerian models. The fluid is thought of as being composed of fluid cells that form a uniform grid. Each cell contains a number of fluid molecules, or particles. The grid-based methods, as a matter of principle, have the drawback of a bounded simulation space which is caused by the finite memory of the computation devices. The fluid can not flow freely in the virtual environment because it can not exist outside the grid; it is locked in the grid.

The grid provides a solution to estimate the derivatives using a finite difference method (FDM). For theoretical details on Eulerian fluids see [5, 8, 19]. Although the Eulerian method provides better description of some the properties of fluids (mass-density, pressure field) compared to the Lagrangian method, but the major disadvantage is the grid itself.

The particle-based methods in the literature are called the Lagrangian models. These methods represent fluids using a discrete set of particles. These particles simulate the flow of the fluid by solving the particle hydrodynamics. For the real-time applications this has some advantages

against the grid-based methods. The biggest advantage is that the fluid can spread freely in the simulation space. For these reasons, we focus on the Lagrangian method based on the Smoothed Particle Hydrodynamics [6], which is the most popular solution for this kind of applications simulating the fluids. Each particle distributes its fluid properties to the surronding particles in the near neighbourhood using a radial kernel function (the smoothing kernel). The evaluated new property of a particle is the sum of property quantities of neighbour particles weighted by the smoothing kernel.

The Smoothed Particle Hydrodynamics method was introduced in 1977 by Monaghan and Gingold [15] and independently by Lucy [9]. First, it was used in astrophysics to simulate large scale gas dynamics [18]. Later, it was applied to incompressible flow problems. In the real-time applications, at first the Eulerian method was favoured. Müller, Charypar and Gross [12] showed that the SPH is well suitable for interactive real-time applications. Vizualization of the results plays as important role as the main simulation process in these applications. The most often used methods for rendering are these: point splatting [12], marching cubes or marching tetrahedra [21, 20], and raycasting [7].

3 Smoothed Particle Hydrodynamics

This section is focused on the theoretical explanation of the Lagrangian equations and how they are discretized. The most important parts are symmetrizations of the forces, like pressure and viscosity. The last part about the smoothing kernels describes the force computations.

3.1 Lagrangian equations

This subsection describes the Lagrangian method of the fluid simulation. The conservation of mass / continuity is given by:

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

where ρ is density, $\mathbf{v}$ velocity and t is time. Using the substantive derivative [4], which specifies: $\frac{d\mathbf{v}}{dt} = \frac{\partial \mathbf{v}}{\partial t} + v \cdot \nabla v$, it defines the strength of how viscous the fluid is. We get the Lagrangian formulation of the conservation of momentum:

$$\frac{d\mathbf{v}}{dt} = -\frac{1}{\rho}\nabla P + \frac{1}{\rho}\nabla \cdot \tau + \mathbf{f}, \tag{2}$$

where v is the velocity field, ρ the density, P the pressure, $\mathbf{f}$ are external forces and τ the viscosity coefficient. We can ignore the mass conservation if we assume that the number of particles in constant. Finally end with this expression:

$$a_i = -\frac{1}{\rho_i}\nabla P_i + \frac{1}{\rho_i}\nabla \cdot \tau_i + \mathbf{f}_i = f_i^{pressure} + f_i^{stress} + f_i^{external}, \tag{3}$$

where a_i is the acceleration of a particle, $f_i^{pressure}$ is the pressure force, f_i^{stress} is the deviatoric stress (viscosity) and $f_i^{external}$ is the sum of external forces (e.g. gravity, boundaries). The remaining equations are derivable from previous equations or they can be found in [13, 11].

3.2 Discretization

The SPH can be used for any kind of fluid simulation. This is an interpolation method for the particle systems. In this method, the field quantities are only defined at discrete particle locations and can be evaluated anywhere in the space. For this purpose, the SPH distributes the property quantities in the neighbourhood of any particle using the smoothing kernel. In the SPH, a scalar quantity is interpolated at a specific location by a weighted sum of the contributions from all particles:

$$A_i(\mathbf{r_i}) = \sum_j m_j \frac{A_j}{\rho_j} W(\mathbf{r_i} - \mathbf{r_j}, h), \tag{4}$$

where j iterates over all particles, m_j is the mass of the particle, A_j is the scalar property, r_j is the position and h is the radius of the smoothing kernel.

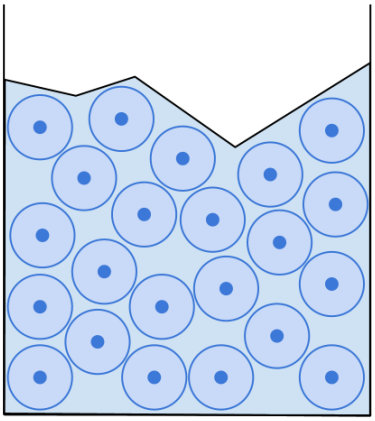

Figure 1: Lagrange particle-based fluid structure in 2D. The particles are represented by the dots. The circles represent the volume of each particle.

The function $W(\mathbf{r},h)$ is called the smoothing kernel with the core radius h. The W must be even ($W(\mathbf{r},h) = W(-\mathbf{r},h)$) and have finite support. If the W is even and normalized, the interpolation is of second order accuracy. The kernel is normalized if the following is true:

$$\int W(\mathbf{r})\, d\mathbf{r} = 1. \tag{5}$$

Each particle i represents a certain volume $V_i = \frac{m_i}{\rho_i}$; this means they have mass and density. The mass m_i of each particle i is constant throughout the full simulation process. The density ρ_i needs to be evaluated at every timestep. With the substitution we get the following equa-

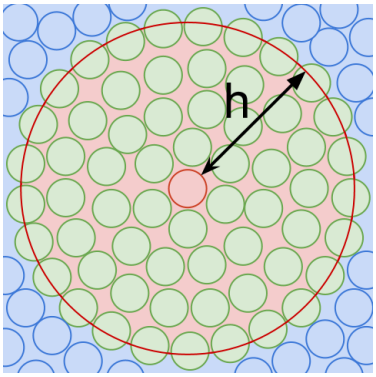

Figure 2: The smoothing distance h and the surrounding particles within it.

tion for the density at a given position $\mathbf{r}$:

$$\rho_i(\mathbf{r}) = \sum_j m_j W(\mathbf{r_i} - \mathbf{r_j}, h) \tag{6}$$

In the SPH, the derivatives of the field need not to be evaluated. These derivatives only affect the smoothing kernels. The gradient of A is:

$$\nabla A_i(\mathbf{r_i}) = \sum_j m_j \frac{A_j}{\rho_j} \nabla W(\mathbf{r_i} - \mathbf{r_j}, h), \tag{7}$$

while the laplacian of A is given by:

$$\nabla^2 A_i(\mathbf{r_i}) = \sum_j m_j \frac{A_j}{\rho_j} \nabla^2 W(\mathbf{r_i} - \mathbf{r_j}, h). \tag{8}$$

The main problem of the SPH is that to derive the fluid equation it is not guaranteed to satisfy all the physical principles such as symmetry of the forces and conservation of the momentum. The problems are solved via different types of smoothing kernels [13], which are discussed in the next chapters, see Fig. 3, 4, and 5.

3.3 Pressure

The applications of the SPH rule described in Eq. (1) yields:

$$\mathbf{f}_i^{pressure} = -\nabla p(\mathbf{r}_i) = -\sum_j \frac{p_j}{\rho_j} \nabla W(\mathbf{r_i} - \mathbf{r_j}, h) \tag{9}$$

As we mentioned in the previous subsection the force would not be symmetric. It can be seen simply when only two particles interact. The first particle i only uses the pressure at particle j to compute its pressure force. Since the pressures at the location of the two particles are not equal in general, the pressure forces will not be symmetric between them. Newton's 2nd law states that, to every action there is always an equal and opposite reaction: or the forces of two bodies on each other are always equal and are directed in opposite directions. There are different types of solutions for this symmetrization problem in the literature. Gross, Müller and Charypar [12] described a simple and fast solution of this problem:

$$\mathbf{f}_i^{pressure} = -\sum_j \frac{p_i + p_j}{2\rho_j} \nabla W(\mathbf{r_i} - \mathbf{r_j}, h). \tag{10}$$

Now the pressure force is symmetric because this equation uses the aritmethic mean of the pressures of the interating particles.

At first, the pressure need to be evaluated, which is done in two steps. The first is density computation from Eq. (6). Then the pressure can be computed via the ideal gas state equation:

$$p = k\rho, \tag{11}$$

where k is the ideal gas constant that depends on the temperature. Desbrun and Gascuel [10] suggest a modified version of the pressure computation:

$$p = k(\rho - \rho_0), \tag{12}$$

where ρ_0 is the rest density. This modification does not affect the pressure forces matematically. However, this makes the simulation numerically more stable, because it has influence only on the gradient field smoothed by the SPH.

3.4 Viscosity

In the SPH, the viscosity yields:

$$\mathbf{f}_i^{viscosity} = \mu \sum_j m_j \frac{\mathbf{v}_j}{\rho_j} \nabla^2 W(\mathbf{r_i} - \mathbf{r_j}, h), \tag{13}$$

where μ is the specific viscosity constant for the fluid. The viscosity force is asymmetric, too. Since the viscosity forces are only dependent on the velocity differences and not on the absolute velocities, the solution of this problem is simple. It can be easily symmetrized by using the velocity differences:

$$\mathbf{f}_i^{viscosity} = \mu \sum_j m_j \frac{\mathbf{v}_j - \mathbf{v}_i}{\rho_j} \nabla^2 W(\mathbf{r_i} - \mathbf{r_j}, h) \tag{14}$$

The particle is accelerated in the direction of the relative speed of its environment.

3.5 Smoothing Kernels

The smoothing kernels used in the interpolations have a great influence on speed, stability and physical plausibility of the simulation and should be chosen wisely. Choosing a good smoothing kernel can be important for several aspects of the simulation. The numerical accuracy is highly dependent on the smoothing kernel and the research has shown that certain kernels offer better results than others [1]. The SPH uses different kernels for each calculation. Even though the Gaussian kernel has very nice mathematical properties, it is not always the best kernel to use.

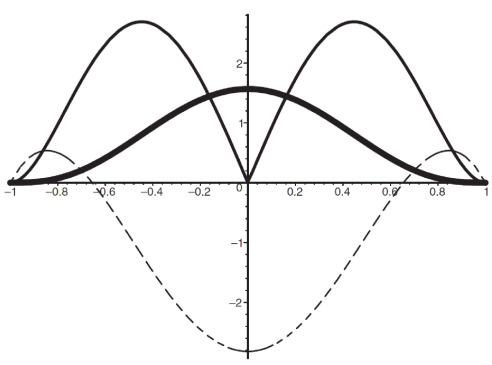

Figure 3: Smoothing kernel W_{poly6} from [12]. The thick lines show the kernels, the thin lines their gradients in the direction towards the center and the dashed lines the Laplacian.

It does not have compact support, and requires the evaluation of computally expensive exponential function. Instead of that, the 6^{th} degree polynomial kernel can be used, which was suggested in [12] for density computation:

$$W_{poly6}(\mathbf{r},h) = \frac{315}{64\pi h^9}\left(h^2 - |\mathbf{r}|^2\right)^3 \qquad (15)$$

An important feature of this kernel is that $\mathbf{r}$ appears only in the squared form and can be evaluated without computing square root in the distance calculations.

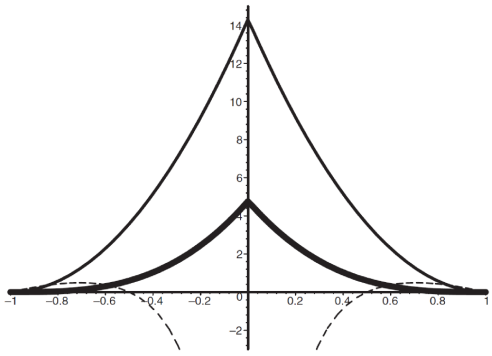

Figure 4: Smoothing kernel W_{spiky} from [12]. The thick lines show the kernels, the thin lines their gradients in the direction towards the center and the dashed lines the Laplacian.

The gradient from the pressure field is used in the calculation of the pressure force. For our pressure force evaluations between the particles we employ the spiky kernel [12] as our pressure kernel, which yields:

$$W_{spiky}(\mathbf{r},h) = \frac{15}{\pi h^6}\left(h - |\mathbf{r}|\right)^3, \qquad (16)$$

which generates the necessary repulsion forces.

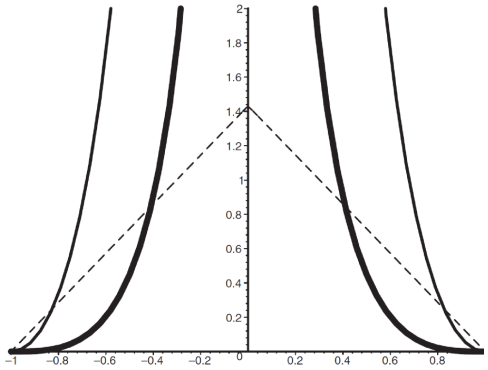

Figure 5: Smoothing kernel W_{vis} from [12]. The thick lines show the kernels, the thin lines their gradients in the direction towards the center and the dashed lines the Laplacian.

The viscosity of a fluid is a phenomenon which is caused by the internal friction force between the particles. It decreases the kinetic energy by converting it into heat. This means that this force gives stability and smoothing effect on the velocity field in fluids. The SPH variant at the viscosity force term is:

$$W_{vis}(\mathbf{r},h) = \frac{15}{2\pi h^3}\left(-\frac{|\mathbf{r}|^3}{2h^3} + \frac{|\mathbf{r}|^2}{h^2} + \frac{h}{2|\mathbf{r}|} - 1\right) \qquad (17)$$

The Laplacian of the smoothing kernel in Eq. (17) is constrained to be positive. This is required because the forces due to viscosity to can increase the relative velocity, and thereby introduce energy and instability into the system.

How to derive the remaining equation from the previous like the gradient and the laplacian of the smoothing kernels, are discussed in [11, 16].

4 Surface Tracking and Visualization

An important part of the simulation process is rendering and visualization its results. The choice of the rendering method depends on many aspects. Two main types exist: online (real-time) and offline rendering. Offline methods provide more plausible results, but the computations take longer time. In this case, a video is created from the rendered frames. The biggest disadvantage is that the user loses the interaction with the simulation system.

4.1 Point Sprites

Rendering particles with the point sprites method is an easy, simple and fast solution comparing to the other rendering methods, like the marching cubes or raycasting method. The implementation is done in the GPU's fragment shader. Each particle is rendered as a square shaped formation. The size of this square is dependent on the distance between the particle and the camera. In the fragment shader, the pixels outside the computed radius are

discarded. Pixels that are not discarded are shaded to create the fake 3D ball effect, see Fig. 12.

4.2 Marching Cubes

Marching cubes is a method for extraction of isosurfaces in volumetric data. The method is based on the triangulation of the isosurface. The volume is sampled by a marching cube, which is traversing the volume. At each position the corners of the cube are tested, whether they are above or below the isosurface. By the configuration of the corners, triagles are generated to cover the surface.

This method highly depends on the resolution of the sampling grid, see Fig. 6. To eliminate the sharp edges, tessellation can be used. The algorithm is easily paralellizable; this means that, to speed up the rendering of the isosurface the GPU is used. The disadvantage of this method is the need of the additional normal interpolation for better shaded results.

5 Implementation

The aim of this work was to experiment with the methods described in the previous chapter and implement them running in realtime on the GPU. Next, the implementation details of simulation and rendering are described.

5.1 Uniform Grid

The uniform grid is used to track and search for the adjacent particles in the grid cells while evaluating the field quantities with the smoothing kernels. The implementation of this structure is inspired by the work of Green [17]. The space is divided into uniformly sized cells. A particle is assigned to a cell by its position. The minimal possible size of a cell can not be smaller than the size of the particles. A particle can potentially overlap several grid cells, which means that when procesing the quantities, the particles in the neighbouring cells must also be examined. The

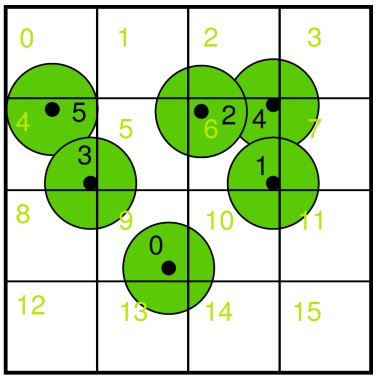

Figure 7: Particle mapping in uniform grid from [17].

assigned cell of a particle is known by its actual position, so the neighbour cells can be found, too. Each cell has its unique hash code and each particle has its unique index. Two arrays are needed: the first one contains indices of the particles and the second one contains the hash codes of the containing cells of the particles. In the next step, the two arrays are sorted using the array which contains the hash codes, as the key for sorting. This process is illustrated by figure 8. As seen in the figure, the cell with the hash code 1 contains two particles with indices 1 and 3. Finally an iteration is needed over the hash array to find the start and end positions of the cells and store them in an extra array.

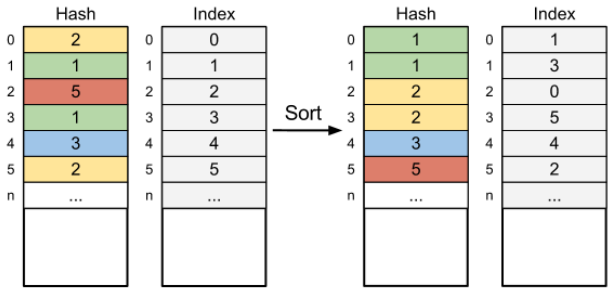

Figure 8: Sorting of particle indexes and hash codes.

5.2 Simulation

The simulation part has three main steps. The first step is density and pressure evaluation. This step is sped up by the precalculad parts of the smoothing kernels. Each kernel has a constant part without the variable vector $\mathbf{r}$. The size of the vector represents the distance between the two examined particles. The precalculated values are stored in the constant memory of the GPU. This solution results in an extremly fast access to the values and saves computational time.

In the second step, the internal forces are evaluated, like the pressure force and the viscosity force. The details of the computations are described in the previous sections.

The last step is the integration of the velocity and further position update. At each integration step, the new velocities are computed. The velocity depends on the internal (presurre, viscosity) and external forces (computed from the particle–boundary interactions). The new position is computed from the previous position and the actual velocity. To intergrate the velocity, the Leapfrog intergration [2] is used. The name of this integrator is a result of the above formulation of it; the velocities ”leap over” the positions. The Leapfrog integration is a simple method to numerically integrate the differential equations. This method is a fairly good compromise between the naive Euler method and more advanced methods that require more than a single evaluation for each force. The default scheme can also be formulated in a form where all quantities are defined at discrete times only:

$$x_{i+1} = x_i + v_i \mathrm{d}t + \frac{a_i}{2}\mathrm{d}t \qquad (18)$$

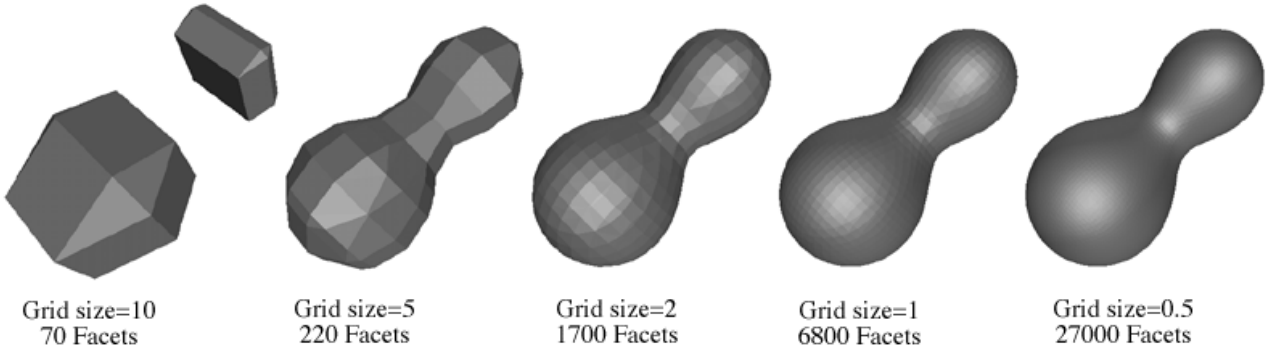

Figure 6: This figure shows the dependence of the maching cubes method on the resolution of the sampling grid from [14]. The higher grid resolution allows courser or finer approximation of the isosurface.

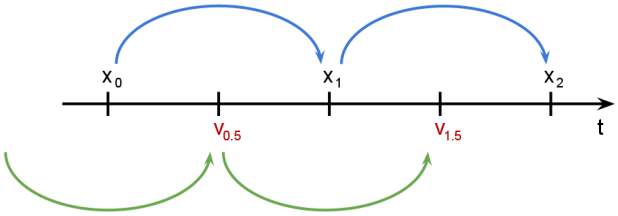

Figure 9: The leapfrog integrator.

$$v_{i+1} = v_i + \frac{a_i + a_{i+1}}{2} \mathrm{d}t. \tag{19}$$

5.3 Rendering

As we mentioned in section 4, two rendering methods were used in this work. A simpler of them is the point sprites method. This solution is computed in the GPU's fragment shader.

The second is the marching cubes method [14]. Comparing to the point sprites method, it can not be rendered direcly using the particle positions. The marching cubes method is based on the triangulation of the fluid surface. The efficient implementation of the algorithm is done by the lookup tables. Edges and corners of the cube are numbered. An 8-bit vector is used to store the configuration of the cube. Each corner has a unique number which is used as an index into the bit vector. If the corner is below the iso–value, than the nth bit (where n is the unique number of the corner) is set to 0, or to 1 when it is above the iso–value.

The lookup tables are in size of 256, because of the possible configurations (2^8). Three types of lookup tables are used. The first contains 12-bit vectors. These vectors are similar to the configuration vector of the cube, but this vector stores the intersected edges for the specified configuration. The next table contains the number of required triangles for the configurations. The third and last table contains the unique numbers of intersected edges for the required triagles.

The first step is to evaluate the occupied voxels and compute the number of the required triangles for each voxel. The following step is to count the number of the required triangles to cover the whole surface. It follows the allocation of the vertex buffer for the vertices of the triagles. In the final step the normals are interpolated. It is very important for the additional tessellation. The tessellation is used to get smoother surface. The details could be also improved by using a higher resolution grid.

6 Results

By using the GPU, the SPH method, and the marching cubes method, the goal of this work was achieved. The SPH method is running in real-time with 60000 particles when no rendering method is used. When using lower grid resolutions, there is a performace drop compared to higher resolutions, see tables 1, 2, 3, and 4. This effect is caused by the high density of particles in the cells. Per cell computations take longer time, because of the higher particle count. The test results of the implemented simulation and the rendering methods are summarized in tables 1, 2, 3, and 4.

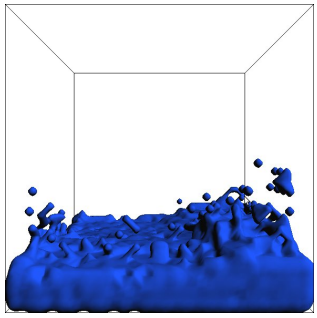

Figure 10: Fluid rendered with the marching cubes method.

When using the marching cubes and/or the tessellation there is a performace drop at higher grid resolutions, see the test results.

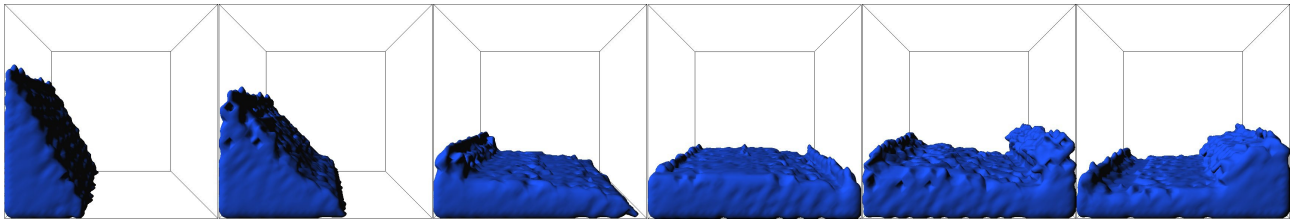

Figure 11: The fluid motion is simulated by 15,000 fluid particles with marching cubes rendering.

	Number of particles: 15000		
Grid res.	30x30x30	50x50x50	80x80x80
SIM	6	5	4
MC	2	5	12
NI	8	11	15
Sum	16	21	31

Table 1: The benchmark shows the results measured in milliseconds at different grid resolutions. *SIM* is the simulation, *MC* is the Marching cubes rendering, and *NI* stands for the normal interpolation. In each table, the upper row contains the number of particles used in the benchmark.

	Number of particles: 30000		
Grid res.	30x30x30	50x50x50	80x80x80
SIM	19	13	9
MC	2	5	12
NI	9	12	18
Sum	30	30	39

Table 2: Benchmark results for 30,000 particles.

The algorithms are implemented in the C++ programming language. The sourcecodes for the GPU were implemented for the CUDA runtime API 4.0. The API allows high abstraction level and provides fine support for programming the GPU. The final program was tested on the NVidia GeForce 540M graphics card, with 96 cores.

7 Conclusions and Future Work

The SPH method is a very realistic and fast appoximation of the real world fluids, but it has some limitations. One of these limitations is the need of large number of particles for the simulation to get realistic results.

Many experiments were made in this work to speed up the SPH and its rendering methods. The uniform grid helps the access to adjacent particles. This structure speeds up the simulation, but it needs additional computations, which can be better optimized, like the radix sorting of its arrays.

In the future more experiments can be done to speed up the searching and sorting the particles. The additional surface tension computation can improve the surface details of the fluid and make it more realistic. Other experiments can be done to improve the existing algorithms of rendering, e.g. extract a better approximation of the surface with marching cubes or implement the raycasting method, which provides much more plausible results.

	Number of particles: 45000		
Grid res.	30x30x30	50x50x50	80x80x80
SIM	31	22	18
MC	3	5	13
NI	9	12	20
Sum	43	39	51

Table 3: Benchmark results for 45,000 particles.

	Number of particles: 60000		
Grid res.	30x30x30	50x50x50	80x80x80
SIM	45	32	24
MC	3	5	13
NI	10	13	30
Sum	58	60	67

Table 4: Benchmark results for 60,000 particles.

References

[1] A. J. C. Crespo. *Application of the Smoothed Particle Hydrodynamics model SPHysics to free-surface hydrodynamics*. PhD thesis, University of Vigo, 2008.

[2] D. H. Eberly. *Game Physics*. Elsevier Science Inc., 2003.

[3] G. K. Batchelor. *An Introduction to Fluid Dynamics*. Cambridge University Press, 1967.

[4] R.A. Granger. *Fluid Mechanics*. Courier Dover Publications, 1995. ISBN 0486683567.

[5] M. J. Harris. Fast fluid dynamics simulations on the gpu. In *In GPU Gems, Programming Techniques, Tips, and Tricks for Real-Time Graphics*. Addision-Wesley, 2004.

[6] J. J. Monaghan. Smoothed Particle Hydrodynamics. *Reports on Progress of Physics*, 68:8, 2005.

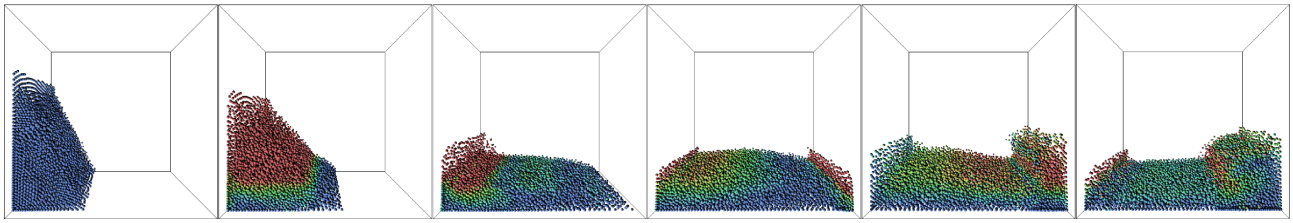

Figure 12: The fluid motion is simulated by 15,000 fluid particles with point sprites rendering. Colors indicate the velocity field of particles, red is high and blue is the low.

[7] R. Westermann J. Krüger. Acceleration techniques for gpu-based volume rendering. In *Proceedings of the 14th IEEE Visualization*, 2003.

[8] K. Erleben, J. Sporring, K. Henriksen, H. Dohlmann. *Physics Based Animation*. Charles River Media, 2005.

[9] L. B. Lucy. A numerical approach to the testing of the fission hypothesis. *Astronomical Journal*, 82:1013–1024, 1977.

[10] M. Desbrun, M. Gascuel. Smoothed particles: A new paradigm for animating highly deformable bodies. *Eurographics Workshop on Computer Animation and Simulation (EGCAS)*, pages 61–76, 1996.

[11] M. Kelager. Lagrangian Fluid Dynamics Using Smoothed Particle Hydrodynamics, 2006.

[12] M. Gross M. Müller, D. Charypar. Particle-based fluid simulation for interactive applications. In *Proceedings of 2003 ACM SIGGRAPH Symposium on Computer Animation*, 2003.

[13] O. E. Krog. *GPU-based Real-Time Snow Avalanche Simulations*. PhD thesis, Norwegian University of Science and Technology, 2010.

[14] P. Bourke. Polygonising a scalar field [online]. `http://paulbourke.net/geometry/polygonise/`, 1994.

[15] J. J. Monaghan R. A. Gingold. Smoothed particle hydrodynamics: theory and application to non-spherical stars. *Royal Astronomical Society, Monthly Notices*, 181:375–389, 1977.

[16] S. Auer. *Realtime particle-based fluid simulation*. PhD thesis, Technische Universität München, 2009.

[17] S. Green. Particle Simulation using CUDA [online]. NVidia Corporation, 2010.

[18] V. Springel. Smoothed particle hydrodynamics in astrophysics. *Annual Review of Astronomy and Astrophysics*, 48:391–430, 2010.

[19] J. Stam. Stable fluids. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, 1999.

[20] Y. Uralsky. Practical metaballs and implicit surfaces. In *Game Developers Conference 2006 Presentation*, 2006.

[21] H. E. Cline W. E. Lorensen. Marching cubes: A high resolution 3d surface construction algorithm. In *SIGGRAPH '87 Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, 1987.