Smoothed Particle Hydrodynamics in Zero Gravity

Group 7
Oscar Westberg, Teodor Vik, Mikael Zackrisson, Anton Österblad

March 14, 2014

Summary

This project was part of the course TNM085, Modeling project at Linköping University. Groups were able to choose their own research topics and simulate a physical model. This report describes a simulation of a system using Smoothed Particle Hydrodynamics (SPH).

The result is a program able to simulate a 2D SPH particle system with water-like attributes. Up to 900 particles can be simulated simultaneously. The program uses C++ and OpenGL to simulate the system in real time.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Fluid simulation is a popular tool in computer graphics [1] and it is often used to simulate liquid, smoke and explosions. Depending on the quality requirements fluid simulations can be extremely time-consuming animations or simple real-time particle systems. Since particle system is a good way to simulate a complex effect it is very popular amongst developers and thus there are a lot of information to find about the subject. The report is based on Smoothed Particle Hydrodynamics, which is one of the most popular techniques when creating a fluid.

1.2 Purpose

The purpose of this project is to, based on mathematical models and physical formulas, perform a simulation of water in both earth like conditions and in zero gravity. The simulation shall be displayed graphically in real-time.

Chapter 2

Physical systems

2.1 Research

There are several techniques for simulating a fluid. The most common techniques are based on the Navier-Stokes equations which describes the flow of fluids. To practically use the Navier-Stokes equations it is common to make some simplifications in which some properties are assumed to be constant. There are two fundamental approaches on how to implement the Navier-Stokes equations; the Eulerian method and the Lagrangian method. Both are widely used in computer graphics.

The Eulerian method uses a grid-based system, in which each grid point has fluid properties, like velocity, density, pressure, etc., but the points never move. The Lagrangian method is based on a collection of particles that, in addition to the fluid properties, has a position and can move.

2.2 Navier-Stokes equations

Navier-Stokes [2] equations describe the motion of fluid substances as a result of forces.

$$\frac{\partial \overrightarrow{v}}{\partial t} + (\overrightarrow{v} \cdot \overrightarrow{\nabla})\overrightarrow{v} = \frac{-\overrightarrow{\nabla} p}{\rho_m} + \frac{\mu}{\rho_m}\nabla^2 \overrightarrow{v} + \frac{\overrightarrow{f}_{ext}}{\rho_m} \tag{2.1}$$

$$\nabla v = 0 \tag{2.2}$$

The forces are based on the physical quantities of the fluid, velocity v, mass density ρ and pressure p, and also viscosity μ and f_{ext} external forces. Equation 2.1 describes the conservation of momentum, and basically it is Newton's second law for a fluid.

$\overrightarrow{\nabla} p$ the pressure gradient, is due to the particles' will to even out the pressure based on its surroundings. Without it the particles would not know which way to go.

μ the viscosity term, will basically determine how much the velocity of neighbouring particles will affect each other. With a high viscosity constant it is more likely that nearby particles will move together.

f_{ext} is external forces such as gravity.

2.3 Smoothed Particle Hydrodynamics

Smoothed Particle Hydrodynamics (SPH) [2] is a Lagrangian method that divides the fluid into a set of particles. Mass-density, pressure, etc. are estimated using SPH approximations. The properties of a SPH-particle is affected by other particles that lie within a certain “smoothing” distance, thus affected only by its neighbours. The "smoothing" distance is illustrated in Figure 2.1 b [3]. Any desired scalar quantity can be determined for each particle by weighting that quantity from neighboring particles with a “kernel function”, which is illustrated in Figure 2.1 a.

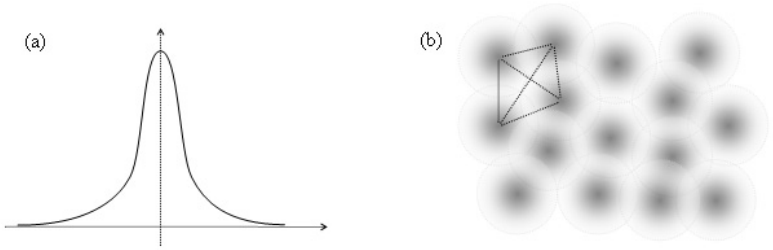

Figure 2.1: Smoothed Particle Hydrodynamics

An SPH simulation involves repeating the following steps as time advances:

1. Compute density at each particle.

   The density will be calculated using the smoothing function (smoothing kernals).

   $$\rho_s(\overrightarrow{r}_j) \approx \sum_i w(\overrightarrow{r}_{ij}) \tag{2.3}$$

   r_j and r_i are two different points in the particle system. r_{ij} is the difference between them.

2. Compute pressure from density.

   The pressure can be approximately calculated using the density. k is a constant. ρ_0 is the ideal density of the fluid.

   $$p_i = k(\rho_i - \rho_o) \tag{2.4}$$

3. Compute forces from pressure gradients.

   Forces come from differences in pressure between two points (pressure gradients).

   $$\nabla p(\overrightarrow{r}_j) = \rho_j \sum_j (\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2}) \nabla w(\overrightarrow{r}_{ij}) \tag{2.5}$$

   ∇w is a gradient calculation using the smoothing function.

4. Compute viscosity.

   $$\frac{\mu}{\rho_i} \nabla^2 v_i \approx \frac{\mu}{\rho_i} \sum_j (\frac{v_j - v_i}{p_j}) \nabla^2 w(r_{ij}, h) \tag{2.6}$$

5. Compute external forces.
   Laws of gravity $F_{gravity} = \rho_i \cdot g$

6. Apply those forces to move particles.
   In order to advance the particles the equations below were calculated for each particle..

$$a_i = (-\nabla p(\overrightarrow{r}_j) + \frac{\mu}{\rho_i}\nabla^2 v_i + g) \tag{2.7}$$

$$v_i = v_i + \Delta t \cdot a_i \tag{2.8}$$

$$r_i = r_i + \Delta t \cdot v_i \tag{2.9}$$

Chapter 3

Implementation

3.1 Matlab

A simple version of the model was simulated in MATLAB in order to test the theory behind the system. This eliminated the chances of any potential errors being caused by the rendering or implemented functions. The calculations were performed in the order that was previously described in section 2.3.

3.2 OpenGL

The particle system's functionality was tested in MATLAB, however the final product was created in C++. The program uses the libraries glm, glew, glfw, OpenGL and standard C++ functionality to render and simulate the particle system. From the main loop the program first checks for input, then updates the particle's properties and then render the scene mainly by using a fragment shader.

Since only the closest neighbors of each particle will have an effect on its behavior it's unnecessary to calculate the effect on every particle for each particle. This gives N^2 calculations. The performance was improved by using spatial subdivision. A uniform grid subdivides the simulation space into a grid of uniformly sized cells. This means that the the force for a given particle can be calculated by only comparing it with all its neighbors within a certain radius.

To ensure the rendered scene looks like water instead of a collections of dots the technique metaballs [4] was implemented. A metaball emits an electric field according to the formula: $[f(x, y) = 1.0/(x^2 + y^2)]$

If the accumulated contribution of several metaball fields reaches a certain threshold, the shader renders the color blue. The metaball technique makes the entire particle system look like a fluid. Figure 3.1 illustrates how the metaballs emits electric fields.

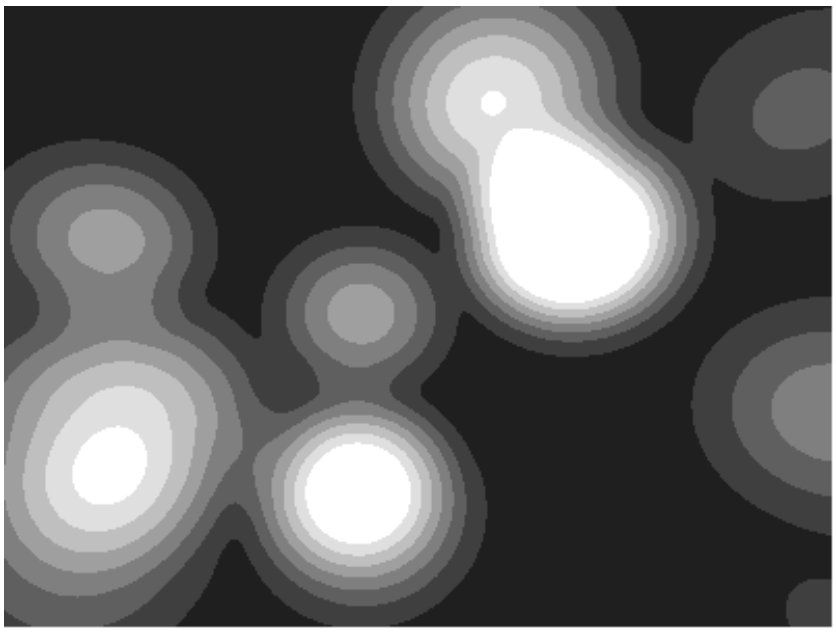

Figure 3.1: Metaball fields

Chapter 4

Result

The SPH-parameters for water and yoghurt were obtained by trial and error during the simulations in matlab. The determined constans are given in Table 4.1.

Table 4.1: SPH parameters from Matlab simulation

SPH parameters	**Water**	**Yoghurt**	**Unit**
Particle mass	0.013	0.014	kg
Ideal density	1000	1050	kg/m^3
Viscosity constant	3.5	20.0	$Pa \cdot s$
Stiffness	3.0	5.0	J
Smoothing distance	0.032	0.04	m
Kernel particles	32	50	n/a

The end product is an interactive program simulating a customizable amount of particles in either zero gravity or with gravity. The particle system is rendered in 2D in a window. Up to 900 particles run at full speed, it is also possible to change shader by pressing the “s" button on the keyboard. The shader with black water and red transparent squares displays the particles’ pressure.

Figure 4.1 displays a picture of the running simulation without metaballs activated. Figure 4.2 displays a picture of the running simulation with metaballs activated. When the metaballs are activated the simulation appear to graphically look like a fluid.

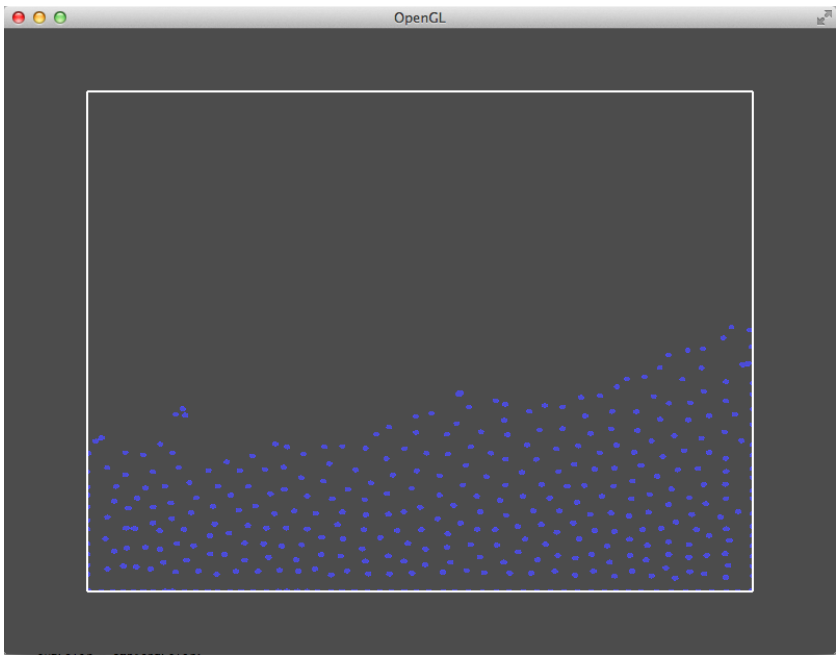

Figure 4.1: Running the program without metaballs activated

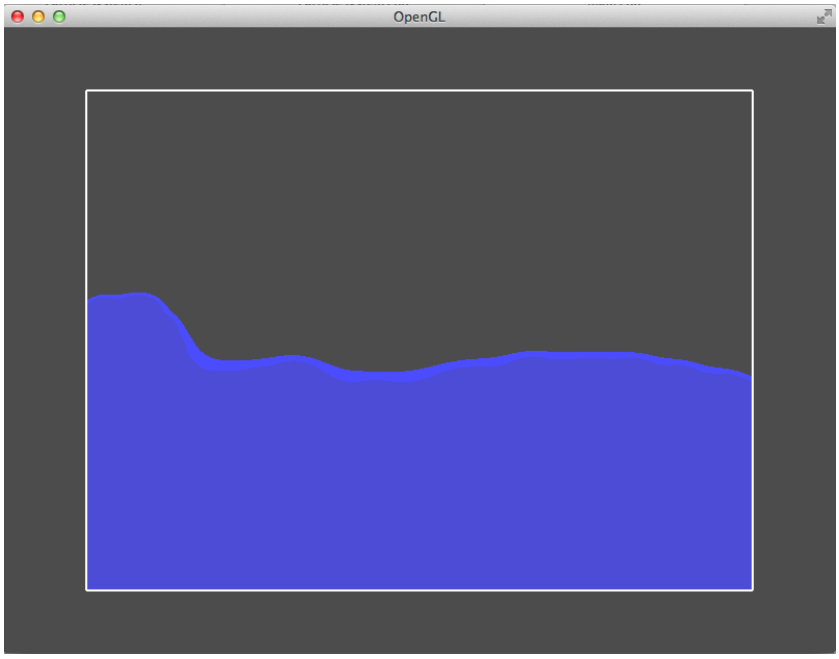

Figure 4.2: Running the program with metaballs activated

Chapter 5

Conclusion and Discussion

Matlab was a great tool to test if the implementation of the physics for the system was correct. Since Matlab shows the correct output there were never any doubts that if something went wrong it was because of miscalculations and not how the system was drawn. However when all code for the system was implemented and the search for correct variables began OpenGL was better to use because it was much faster. The fast transition between Matlab and OpenGL was possible because we had separated the group into working with Matlab and OpenGL simultaneously.

There are a lot of articles about which variables to use when simulating water. However these have been guidelines and not the final values in our simulation since they did not provide us with a satisfying result. After tweaking the system we now think it behaves like water.

The initial idea of the program was to render water in 3D and to use advanced shaders to create a realistic look. The final result however is in only two dimensions. By adding a third dimensions it will drastically increase the shear number of particles and it was decided to stick with two dimension in the final version because of the performance restrictions of the system. The calculations are basically the same if a third dimension is added and will not be difficult to add in terms of coding.

C++ is well known for being a fast programming language and was an obvious choice to start, in retrospect it was still a good choice to use C++. While the final product does not have 3D functionality, any other choice of programming language would most likely not reach the same level of speed with the team's current programming proficiency.

There are several improvements to be made, for instance by introducing multi-core calculations on the CPU or making calculations on the GPU instead. OpenGL is a vital part of drawing in 3D and was a must in the project's initial phase. Since the final product only uses 2D rendering other drawing libraries such as SFML and SDL would have been viable.

Bibliography

[1] Auer, S. *Realtime Particle-Based Fluid Simulation*. Computer Graphics and Visualization, October 4, 2008.

[2] Kelager, M. *Lagrangian Fluid Dynamics Using Smoothed Particle Hydrodynamics*. Department of Computer Science, University of Copenhagen, January 9, 2006. [www] http://image.diku.dk/projects/media/kelager.06.pdf , 24 January 2014.

[3] Dr. Michael J. Gourlay. *Fluid Simulation for Video Games (part 2)*. Intel Developer Zone, May 30, 2012. [www] http://software.intel.com/en-us/articles/fluid-simulation-for-video-games-part-2 , 28 January 2014.

[4] Geiss, R. *Lagrangian Fluid Dynamics Using Smoothed Particle Hydrodynamics*. Department of Computer Science, University of Copenhagen, March 3, 2000. [www] http://www.geisswerks.com/ryan/BLOBS/blobs.html , 13 February 2014.

Appendix A

Manual

How to run the program and hardware requirements

The easiest way to run the program is to execute the runnable files (water.exe or yoghurt.exe) found in the folder "Release". Make sure all the shader files and the glew32.dll-file are included in the same folder as the runnable file. The computer running the program must have hardware supporting OpenGL 3.3 or above.