University of Derby

School of Computing & Mathematics

A project completed as part of the requirements for the

BSc (Hons) Computer Games Programming

entitled

Real-time interaction of fluids with deformable surfaces

By

Alexandru – Cristian Lup

a.lup1@unimail.derby.ac.uk

2014 - 2015

ABSTRACT

This paper presents a fully unified particle based framework for simulating real-time interactions between fluids and soft-bodies. The implementation is based on position based dynamics and incorporates two constraints for simulating the phenomena mentioned above: density constraint to satisfy the incompressibility for the fluid simulation and shape matching constraint to simulate large deformations. The two simulations present a full two-way coupling method which is based on the recent research in the area of physically based animation. The focus of the implementation is on speed and stability, the intended use of the framework being real-time simulations.

Table of Contents

1. INTRODUCTION

The area of physically based animation in real-time applications is constantly gaining more attention from the developers as more and more computational power becomes available. Rigid body physics has been used in video games and other interactive simulations for a very long time, but in most instances, only very basic physical phenomena like gravity, collisions and forces were simulated due the limited computational power available (Gamasutra 2007).

1.1 Hardware evolution

With the development of parallel computing platforms like CUDA and OpenCL, the introduction of compute shader programmable stage in the graphics APIs, the level of granularity and accuracy of the physics simulations has increased allowing them to become more than an aesthetic feature of the game and actually driving the gameplay like in "World of Goo", "Where's my water" or the "Portal" series.

The CPU power and the number of cores and threads per CPU has also increased considerably in the last decade allowing physics simulations to gradually become no longer limited to rigid bodies and simple simulations.
Given this boost of hardware performance, a whole new range of computationally expensive applications like fluid dynamics, deformable objects or fractures can be simulated (Gamasutra 2007).

1.2 Industry

Physics simulations and physically based animations are commonly used in other areas than games. They have a very wide range of applications in various areas like computer aided design, medical software, film industry or engineering.

One of the most important area where physics simulations and physically based animations are used is engineering. Here, the main purpose of the simulation is usually to complement an existing real experiment. The main focus in this type of simulation is the accuracy, as the ultimate purpose is to confirm the results of an experiment (Muller 2014).

On the other side, there is the area of physics simulations in graphics. The idea here is to imitate physical phenomena and produce a good looking results. This approach focuses on controllability while sacrificing accuracy. The trade-off is usually made in order to lower the duration of the simulation and increase its stability and speed (Muller

2014). This kind of simulation can be split into two main categories: real-time simulations and offline simulations.

1.3 Real-time vs Offline simulation

Depending on the level of detail and quality required, a simulation can be made in real-time, allowing a certain degree of interactivity or offline where data generated is loaded by a renderer which can produce the desired effect.

Offline methods are usually used to simulate effects and physical phenomena for movies. These use high resolution grids for accurate fluid simulations and finite element methods for deformable meshes (Muller 2014). Unified solvers, based on particles are very popular in offline rendering tools like Nucleus in Maya or Lagoa made by SoftImage.

Computer aided design, medical software and game physics represent the interactive physics area. To be considered interactive, a simulation needs to be able to update the state of the world at certain rate which depends on the type of application. Usually these rates need to be above 15, 30, or 60 frames per second.

1.4 Two-way coupling

The problem of simulating multiple physics phenomena interacting can occur in all of the above mentioned areas. Simulating both a fluid and deformable objects in the same application is computationally challenging because of the complicated physics of the fluid and the complexity of the deformable objects. On top of this, the interaction between the two has to be handled (Benra 2011).
Two-way coupling, as opposed to one-way coupling which simulates the effect of one phenomenon on the other, simulates the action produced by each simulation on the other one. Some examples which can illustrate this, are the interaction of a balloon and the water inside it or pouring water on a piece of cloth. There are loads of real-life examples to illustrate this situation which makes this problem more and more tempting as more computational power becomes available (Yin 2013).

1.5 Aims and objectives

The focus of this dissertation is to create a flexible simulation which can take advantage of the recent research in the field of physically based animation, be easily ported from a desktop environment to mobiles and does not require specialized hardware to run on. The main challenges to overcome, are achieving real-time performance while offering a believable outcome and coupling the two elements (fluid and soft body) so that they benefit from full two-way interaction.

The topic of this dissertation, real-time interaction of fluids with deformable surfaces implies the study of three different sub-topics which together form the objective of the study. The first one is the position based dynamics approach used to simulate these phenomena in a unified way in a particle based environment. Using a position based dynamics framework and implementing a series of constraints like distance, bending, volume or shape matching, we can build the components of the simulation. The fluid part of the simulation can be simulated within the Position Based Dynamics framework by using a density constraint which enforces constant density (Macklin 2013). The deformable objects on the other hand can be simulated by implementing a shape-matching constraint (Muller 2005). Ultimately both components need to implement two-way interaction capabilities. Fluid-solid coupling can be implemented by treating fluid particles as solid particles when they come in contact with rigid bodies or soft bodies and by using the fluid's rest distance as contact distance (Macklin 2013).

2. LITERATURE REVIEW

2.1 Strategy

The literature review will present the three major concepts which are involved in this dissertation: position based dynamics for simulating fluids, meshless deformation for simulating deformable objects and a unified method of coupling these to achieve two-way interaction between them.

2.2 Position based dynamics

2.2.1 Introduction

The traditional approach for simulating dynamics is by using forces (PhysXInfo. 2013). These are accumulated at each time step and then, using Newton's second law of motion, the acceleration is calculated. Then, by using a two-step integration the velocity and the position are calculated.

Some other methods use impulse to control the animation. This skips one level of integration because impulse operates directly on velocity.
The next natural step forward is to work directly with positions. This offers total control over the particles and solves the instability problems caused by energy gain when doing explicit integration. This approach allows the integration of various constraints for simulating different phenomena like gases, water, soft bodies, rigid bodies or cloth with two-way interaction between them.

2.2.2 Previous work

One of the first mentions of simulating dynamics using constraints, is the paper Gascuel et al. (1994). The approach creates certain transformations like translations and rotations based on position displacement and uses these to simulate the rigid body dynamics (Gascuel 1994). This method is based on iterative corrections of displacement and offers a very limited range of motion.

The same idea of displacement based constraint is revived by Faure et al. (1999), but now with strong emphasis on interactivity rather than accuracy. The method uses linearized displacement constraints combined with an iterative equation solver and the Stoermer integration scheme to handle the instability problems (Faure 1999).

Desbrun et al. (1999) used the idea of constraint projection to limit the stretching of the springs in his mass-spring simulation.

Faure's idea was further generalized by Muller et al. in (Muller 2007) , when compared to the above mentioned papers, this method operates directly on non-linearized constraint functions which is a full position-based approach compared to (Desbrun 1999), where he used constraint projection just as a tweaking mechanism for his simulation.

Position based dynamics was developed initially for simulating rigid body dynamics but was extended later to handle fluids by enforcing a density constraint in a position based dynamics framework.

2.2.3 Examples of position based dynamics in practice

This position based dynamics approach was first used in practice as a solver for the physics simulation in the Hitman Codename 47's engine. At that moment, idea was not officially formulated and the engine used Verlet integration rather than Euler integration (Jakobsen 1999), but the functionality of the solver is very similar. Another example is the OpenCloth framework, which simulates cloth using different methods ranging from position based dynamics, classic Newtonian dynamics or finite element methods (Opencloth 2012). Position based dynamics methods are also used in the PhysX SDK (PhysXInfo 2013) as well as Bullet physics and Havok engines (Muller 2014).

2.3 Deformable bodies

2.3.1 Review of existing methods

During the years, many approaches to simulate deformable surfaces have been developed. Terzopoulos et al. (1987) published the first paper about the representation of deformable bodies in computer graphics (Muller 2005). He used the theory of elasticity to model differential equations which can simulate the behavior of soft bodies with respect to time.

Most approaches to simulate deformable bodies focus on accuracy of the material representation and on stability, sacrificing the performance (University of Pennsylvania 2014). There are very few implementations which are suitable for interactive applications such as games because most approaches require a specific object representation, do not handle properly large deformations and does not provide unconditional stability required for games (Muller 2005).

One algorithm which implements a simulation of large deformations while remaining stable even when parts of the mesh are inverted, is presented in the paper (Irving 2004). The algorithm uses a finite element representation of both

volumetric object and thin shells. Although the implementation is stable and accurate, it is not fast enough for interactive applications.

Deformable objects have seen many implementations over the years, like mass-spring systems in Baraff & Witkin (1998), finite differences approach in (Irving 2004) or finite element methods used by Debunne et al. (2001) (Muller 2005).

To further improve these, various methods which can trade accuracy for performance have started to be developed. Different acceleration strategies have been used by Baraf & Witkin in the (Baraff and Witkin 1998) which used a fast implicit integration method or multi-resolution models used by Debunne in (Debunne 2001).

2.3.2 Shape matching

The approach for simulating deformable object used in this dissertation is based on the paper Muller et al. (2005). This a geometric particle based implementation which is based on shape matching instead of deriving the deformation from the object properties which would require a different representation of the objects (e.g. tetrahedral). The choice of this method is motivated by the fact that it is based on a particle representation and it can be easily coupled with other particle based simulations like particle based fluids.

The algorithm works on a set of particles with known positions and masses which have no connectivity information and don't use a specific object representation. The particles within the particle cloud don't have inter-particle interactions but can interact with the environment and external forces (Muller 2005).

The biggest problem when using shape matching is to set the correspondences between the initial state of the particle cloud and the current state of the representation. The solution proposed in this paper handles this by setting the initial and the current particle cloud representation based on the particle's initial and current position which are known in advance. These are used by the shape matching algorithm to find the goal position of the particle. Then, during each update step, the particle is then pulled towards its goal position (Muller 2005).

2.4 Fluids

2.4.1 Review of existing methods

When simulating physics based animation in computer graphics, there are two major approaches to represent fluids and deformable solids. They can be modelled as a continuum (Eulerian approach) or discrete (Lagrangian approach). The choice of the approach influences how the motion of the fluid is tracked and handled. If treated from the Lagrangian perspective, each point in the fluid is considered as a separate particle and has its own position and velocity. The Eulerian approach handles things differently. Instead of tracking each particle individually, it separates the volume occupied by the fluid into cells and tracks the properties of the fluid in those cells. The flow of the fluid will contribute to changes of the fluid properties in the cells like density or velocity (Bridson 2008).

Because the purpose of this project is to simulate both fluids and deformable solids in a unified manner which favours the two way coupling between the two, the approach for the fluid part of the simulation is particle based.

Particle based methods have been around for a couple of decades, the first paper written on the theory and applications of smoothed particle hydrodynamics (SPH) dating from 1977 (Smoothed particle hydrodynamics - Seminar 2005). The theory of SPH was adapted for simulating fluid particles by Monaghan in 1994, the initial paper formulated SPH in the context of astrophysics (Monaghan 1992).

SPH method for simulating fluid flow has a couple of properties which make it attractive from an implementation point of view: mass conservation, pressure term which results from a weighted contribution of the neighbouring particles rather than through solving equations or because of the Lagrangian discretization of the fluid which makes the domain not required to be known in advance (Macklin 2013).

SPH methods usually have problems when trying to enforce incompressibility requiring stiff equations which limit the time step (Becker 2007).

Muller et al. (2003) presented a SPH implementation which uses viscosity and implements surface tension by using a low stiffness equation of state.

There have been attempts to create particle based simulations by using hybrid methods as well. Brackbill and Ruppel (1988) is an example where a grid type representation is being used for solving the pressure term and the velocity calculated is then passed to act directly on individual particles.

2.4.2 Position based fluids

The implementation presented in this project is based on the paper (Macklin 2013). This method was chosen over a SPH based implementation because of the stability of the algorithm when using a large time-step. The PBD solver of the fluid is based on the principles used in SPH methods. The density estimation and the gradient calculations are obtained using kernel functions. Enforcing incompressibility is expensive in traditional SPH implementations but the position based fluid approach solves this by using an iterative density solver which solves a system of nonlinear constraints to enforce constant density (Macklin 2013)

Compared to SPH, PBF offers the same convergence but it does not suffer from energy loss. This is solved by applying vorticity confinement as a velocity post process (Macklin 2013 - Slides). SPH also suffers from fluctuations of the neighbours and when this happens, enforcing incompressibility becomes slow and the simulation can become unstable because of the errors produced by the density estimator kernels. This problem can be addressed in SPH based methods only by reducing the time step, which is not desirable in real-time applications or by increasing the number of neighbour particles (Macklin 2013).

2.5 Unified coupling for fluids and deformable bodies

2.5.1 Introduction

Implementing both fluid and deformable particles based on a common particle structure allows to easily connect the particles using constraints and handle collisions in a unified way (Macklin 2013).

2.5.2 Offline vs. Real-time unified solvers

The idea of a unified solver which can simulate multiple types of phenomena is not new. Unified solvers which use particles as a building blocks have been used to model offline visual effects in Maya – Nucleus solver and Softimage – Lagoa in the recent years (Macklin 2013).

The idea of a unified solver has been implemented by NVIDIA, with NVIDIA FleX. NVIDIA FleX is a particle based simulation technique for real-time visual effects which will be introduced as a feature in the PhysX SDK version 3.4 (PhysXInfo 2013). The Flex pipeline is based on a highly parallel constraint solver that uses the GPU compute capability. The solver provides a variety of

constraint types that allow particles to interact with each other (NVIDIA GameWorks 2013). NVIDIA FleX is going to be included in Unreal Engine 4, a preview version being already available as a special source branch on GitHub (PhysXInfo 2015).

2.5.3 Previous work

During time, there have been numerous attempts to model and simulate real-time fluids and deformable bodies based on particles but most of them as two different frameworks which made it very difficult to fully couple them and simulate two ways interactions.

Point based methods have been used by Muller et al. (2004) to accurately simulate deformations of volumetric objects based on material properties (Muller 2004).

Smoothed particle hydrodynamics (SPH) methods have been used by Solenthaler et al. (2007) to simulate interactions between fluids and solids, where both phenomena are implemented using SPH by only tweaking parameters and using a special kernel function (Solenthaler 2007).

One of the first attempts to simulate fluid-deformable interactions was made by Carlson (2004) where he presented a unified way to simulate a series of phenomena like rigid, melting and flowing materials fully coupled with fluids. Carlson used an Eulerian approach to simulate the fluids which makes his method hybrid.

Muller et al. (2004) presented a method of simulating particle based fluids using SPH coupled with mesh based deformable bodies designed to be used in real-time applications. He used momentum transfer between the fluid particles and the polygons in the mesh in both directions using virtual boundary particles (Carlson 2004).

A similar, but more accurate method was implemented by Akinci et al. (2012) where he accurately calculated boundary forces to act upon both the fluid and the rigid body object (Akinci 2012).

2.5.4 Current implementation

The current implementation is based on the ideas presented in the paper (Macklin 2013) and treats both the fluid and soft body particles as solids when they come in contact and uses the kernel smoothing distance as the collision check distance. To resolve the deformable-fluid interaction, the solid particles are included in the fluid density estimation calculation (Macklin 2013).

2.6 Conclusion

Given the existing research done in the area of physically based animation, the current implementation proposes to implement a Lagrangian unified solver based on position based dynamics which can integrate two constraints for simulating fluids and deformable bodies. For the fluid simulation, a density constraint will be integrated into the framework while the deformable bodies will be based on the shape matching algorithm implemented as a positional constraint as well. The position based dynamics algorithm is based on the paper (Muller 2007) while the idea of density constraint is presented in the paper (Macklin 2013). The current implementation, will implement a modified version of the algorithm from the paper (Macklin 2013) which focuses on creating a parallel solver for the position based dynamics framework.

The fluid – soft-body interaction, is partly based on the ideas presented in the paper Muller et al. (2014) (chapters 5. Rigid bodies - for the collision response implementation using signed distance fields which is going to be adapted to handle soft-bodies, and chapter 7.1 Fluid-solid coupling - for incorporating the solid particles of the soft bodies into the fluid density estimation).

3. METHODOLOGY

3.1 Position based dynamics

3.1.1 Introduction

The aim of this project is to create a framework which can simulate two-way interactions between fluids and deformable bodies at interactive frame rates.

The implementation can be divided into four big parts which constitute the objectives of the dissertation: the implementation of the fluid simulation, the deformable body simulation, the fluid – soft-body two way coupling and gaining the performance data required for analysing the performance of the simulation.

The simulation is built upon the idea of a unified system in which each subsystem is built upon the same building block – the particle. This makes this approach a lot more powerful than others discussed in the literature review chapter because the code can be re-used, collision detection and response is performed easily without having to deal with complex shapes and provides a straight forward intuition towards parallel or concurrent computation (Macklin 2013).

3.1.2 Traditional approach

As an alternative to the classic Newtonian mechanics for simulating dynamics, Muller et al. in (Muller 2007) presented a way of simulating dynamics by calculating position displacements and operating directly on the position of the objects. This gives total control over the particles and solves the instability problems caused by energy gain when using explicit integration (Muller 2007).

3.1.3 Advantages of using Position based dynamics

A framework based on position based dynamics offers certain advantages compared to traditional approaches (Muller 2007):

- Controllability, as the positions of the objects can be manipulated directly;
- Does not suffer from instability problems caused by explicit integration methods and remains stable even with larger time steps;
- The position based solver can handle various constraints which makes it very flexible in terms of the phenomenon simulated;

Almost as simple as explicit Euler integration, position based dynamics represents a method to solve the problems caused by unstable explicit integration and in the same time without the cost of implicit methods. PBD offers almost the same stability as the implicit integration methods, but without sacrificing the performance of the simulation (University of Pennsylvania 2015).

The main idea of PBD is to replace internal forces with constraints and integration with constraint projection. PBD is not a rigorous way of replacing Newton's laws of motion but is good enough for applications which don't require accuracy like video games (University of Pennsylvania 2015).

3.1.4 Constraints

The position based dynamics approach allows the integration of various constraints for simulating different phenomena like gases, liquids, soft bodies, rigid bodies or cloth, making the framework easily expandable based on the type of simulation which needs to be simulated (Macklin 2013).

The framework is simulated using position based dynamics and uses two types of particles to simulate the fluid and the deformable bodies. Both types are based on a common base type to allow the framework to treat them in a unified way.

The spatial discretization is done using point masses represented by particles which have the mass m_i, position p_i and velocity v_i as the main characteristics (Muller 2007). These masses or particles are connected by constraints.

A constraint consists of a function, a set of indices of the particles which are affected by the constraint, a stiffness parameter and a type which can be equality or inequality. The constraint represents the building block of position based dynamics. The stiffness parameter is a measure of how strictly the constraint is satisfied.

C_j: $\mathbb{R}^{3n_j} \rightarrow \mathbb{R}, C_j\left(x_{i1}, x_{i2,} \ldots x_{in_j}\right) = 0$ j - represents the constraint index

Indices $\{i_1, \dots\ i_{n_j}\}, i_k\ \in\ [1, 2, \dots\ N]$

$k_j\ \in\ [0\ ..\ 1]$ Stiffness parameter

3.1.5 Implementation

Pseudo – code for the position based dynamics solver (Muller 2007):

1. *forall particles i do* $v_i \leftarrow\ v_i * \Delta t * w_i * f_{ext}(x_i)$
2. *damp velocities* $(v_1 \dots\ v_N)$
3. *forall particles i do* $p_i \leftarrow\ x_i + \Delta t * v_i$
4. *forall particles i do generate collision constraints* $(x_i \rightarrow\ p_i)$
5. *loop solver iterations*
6. *forall constraints j do project constraint j*
7. *forall particles i do*
8. $v_i \leftarrow\ (p_i - x_i)\ /\ \Delta t$
9. $x_i \leftarrow\ p_i$

3.1.6 Implementation description

1. Integrate acceleration due to external forces
   $w_i * f_{ext}(x_i)$ – Acceleration

   $f_{ext}(x_i)$ – External force represented by the gravity
2. Damp velocities due to friction – can be used a special damping technique or use a basic damping like $v_i \leftarrow\ v_i * 0.999$
3. Apply Newton's first law. Calculates how the particles would move with no constraints
   p_i represents the predicted position before any position correction due to internal force is applied
4. Initialize an array of constraint structures which are going to be used in the projection step
6. Calculate a position offset so that after applied to the current position the constraint is satisfied
8. Recalculate the velocity based on the corrected position
9. Update the current position using the corrected position

   Because the collision is based on the predicted position and the predicted position changes every iteration, the collision constraints need to be generated on the fly every iteration (University of Pennsylvania 2015). Otherwise collisions can be missed and this can cause more instances of tunnelling.

3.1.7 Method description

The whole magic of position based dynamics happens in the constraint projection step. What this step does, is to update the predicted position p_i so that the constraints are approximately satisfied (University of Pennsylvania 2015).

The solver goes through the list of constraints one by one and projects them. This results in a system of non-linear equations and inequalities which are solved in a Gauss-Seidel type iteration (Muller 2007).

For an equality constraint, the general form of the position correction can be calculated as follows:

Taylor expansion:

$C(p + \Delta p) \approx C(p) + \nabla_p C^T(p) * \Delta p;$

Δp – position correction to be found (*)

Choose Δp so that $C(p + \Delta p) \approx 0$ (**)

(*) => $\Delta p = \lambda * \nabla_p * C(p)$ where λ represents a step in the direction of the gradient

(**)

=> $C(p) + \lambda * \left\|\nabla_p C(p)\right\|^2 = 0 \Rightarrow \lambda = \frac{-C(p)}{\left\|\nabla_p C(p)\right\|^2}$ - scaling factor

=> $\Delta p = \frac{-C(p)}{\left\|\nabla_p C(p)\right\|^2} * \nabla_p C(p)$ (***) the expression of the position correction

One restriction to the steps presented above is how the constraint function behaves when the correction is applied. The constraint function needs to be translation and rotation invariant so that it can conserve linear and angular momentum (University of Pennsylvania 2015).

If this requirements is not satisfied, then the projection will step will create some ghost forces which will drag and rotate the object (Muller 2007). If we consider all particles to have the same mass m then the linear momentum is conserved when $\sum_i m_i * \Delta p_i = 0$. In the same way the angular momentum is conserved when $\sum_i r_i \times (m_i * \Delta p_i) = 0$.

In the case of conservation of linear momentum $\sum_i m_i * \Delta p_i = 0$ => $\sum_i \Delta p_i = 0$ as the mass has to have a value different than 0.

$C(p_i + t, p_j + t) \overset{\text{Translation invariance}}{=} C(p_i, p_j) \quad => \quad \sum_i \nabla_{p_i} C(p) = 0 \quad (*)$
(University of Pennsylvania 2015)

$\Delta p \overset{(***)}{=} \frac{-C(p)}{\|\nabla_p C(p)\|^2} * \nabla_p C(p) => \sum_i \Delta p_i = -ct * \sum_i \nabla_p C(p)$ (****)

(*) => $\sum_i \Delta p_i = 0$ => linear momentum is conserved
(University of Pennsylvania 2015)

(****)

The stiffness parameter of the constraint can be incorporated in different ways. The method chosen for this implementation just multiplies the position correction by the stiffness parameter. One problem that needs to be considered is that the effect of the stiffness parameter over multiple iterations is not linear. To solve this problem, for a given stiffness parameter k we calculate $k' = 1 - (1 - k)^{\frac{1}{number\ of\ iterations}}$ and use this in the solver because the effect of this is linear dependent on k and independent of the number of iterations (Muller 2007).

3.2 Position based fluids

3.2.1 Introduction

There are various methods and techniques for simulating fluids, but particle based methods are more attractive than the grid based methods because of their simplicity and flexibility when implemented in a real-time environment (Monaghan 1992).

Particle based approaches simplify the original Navier Stokes equations by making the mass conservation equation and the convection term dispensable (Muller 2003).

Smoothed particle hydrodynamics (SPH) methods and general particle based methods for fluid simulation, because of their Lagrangian discretization, they are not restricted to a certain domain. This can be useful in applications where the domain is not known in advance (Monaghan 1992).

3.2.2 Functionality

With the position based dynamics framework implemented, different constraint can now be integrated. To simulate fluid dynamics Muller et al. in (Macklin

2013) proposed a method to simulate fluids using an iterative density solver integrated into a position based dynamics framework.

Position based fluids (PBF) is a Lagrangian method for solving the Navier Stokes equations which is fast, stable but not as accurate as grid based (Eulerian) methods (Macklin 2013 - Slides).

The solver works by solving a system of non-linear constraints to enforce constant density. The main advantage of using position based fluids is that it offers great stability while using larger time steps which is a problem in other SPH based methods (Macklin 2013 - Slides).

The solver is based on the SPH method and uses density estimation and smoothing kernel functions (Macklin 2013).

3.2.3 Incompressibility

Enforcing incompressibility can be computationally expensive in traditional SPH implementations and usually requires a time step which is impractical for real-time applications (Macklin 2013). The position based fluid method works by setting a set of positional constraints to enforce constant density.

3.2.4 Advantages of using position based fluids over other SPH based methods

Using position based fluids over other SPH based methods presents a series of advantages which makes this method tempting:

- Allows larger time steps
- Smaller smoothing radius which helps improving the performance
- Relatively easy to enforce incompressibility. Offers the same incompressibility and convergence as modern SPH methods but has the stability of position based dynamics (Macklin 2013)

3.2.5 Issues to be addressed when using position based fluids

The method suffers from the same problems as traditional SPH methods. The method is sensible to fluctuations in neighbour particles count. When the number of neighbour particles decreases, enforcing incompressibility becomes more costly and can also create stability issues (Akinci 2012).

In the PBF method, these problems can be solved by using an artificial pressure term. This improves the particle distribution, creates surface tension and lowers the neighbour requirement from traditional SPH.

Another problem which needs to be addressed is the energy loss. Similar to other position based approaches, the position based fluids implementation suffers from accumulation of damping which creates undesirable visual effects.

This is sorted out by applying a velocity post-process in the form of vorticity confinement (Macklin 2013).

3.2.6 Method description

The main idea of position based fluids is the same as in the general position based dynamics implementation.

For the current implementation we generate a series of density constraints to enforce the incompressibility and then transform them into position constraints which can be applied directly per particle basis. This implementation fits very well in the already implemented position based dynamics framework (Macklin 2013) (Macklin 2013 - Slides).

To enforce incompressibility, the solver needs to solve a series of constraints defined as one per particle and based on the particle's position and the position of its neighbour particles.

The expression of density constraint:

$C_i(p_1, \dots p_n) = \frac{\rho_i}{\rho_0}$ where $p_1, \dots p_n$ represent the positions of the particles

ρ_i density at the particle i calculated using the standard SPH density estimator$\rho_i = \sum_j m_j * W(p_i - p_j, h)$. The standard SPH density estimator represents a weighted sum over the neighbours of particle i. $p_i - p_j$ represents the position delta between the current particle and the j^{th} neighbor

ρ_0 water rest density 1000 $\frac{kg}{m^3}$

h the value of the smoothing distance

This formula represents the core idea of SPH and uses samples from a discretized field at particle level to estimate a certain quantity (Monaghan 2005). Depending on the quantity calculated, the kernel function needs to be changed so it ensures stability and accuracy (Muller 2003).

For density estimation the Poly6 kernel is used:

$$W_{Poly_6}(r, h) = \frac{315.0f}{64.0f * \pi * h^9} * \begin{cases} (h^2 - r^2)^3, 0 \le r \le h \\ 0, \ otherwise \end{cases}$$

For pressure computation the Poly6 kernel generates some weird behaviour as the particles tend to build clusters when the pressure is too high (Muller 2003). To solve this, another kernel which can generate the necessary repulsion force is used:

$$W_{Spiky}(r,h) = \frac{15.0f}{\pi * h^6} * \begin{cases} (h-r)^3, 0 \le r \le h \\ 0, otherwise \end{cases}$$

The position based dynamics framework need to know the position correction to satisfy the density constraint.

To satisfy the density constraint we have $C(p + \Delta p) \approx 0, here\ C(p + \Delta p) \approx C(p) + \nabla_p C^T(p) * \Delta p$ $(*)$;

Δp is a translation along the direction of the constraint gradient so it has the following form $\Delta p = \lambda * \nabla_p * C(p)$ (**)

(*) => $C(p) + \nabla_p C^T(p) * \lambda * \nabla_p * C(p) = 0$ => $\lambda_i = \frac{-C_i(p)}{\sum_k \left|\nabla_{p_k} C_i\right|^2}$

(**)

The value of λ needs to be calculated individually for each particle. To enhance the stability of the algorithm, a relaxation parameter ϵ needs to be added as an extra term to the denominator of the previous expression $\lambda_i = \frac{-C_i(p)}{\sum_k \left|\nabla_{p_k} C_i\right|^2 + \epsilon}$ (***)

(Macklin 2013)

To be able to calculate λ_i we need the expression of the gradient of a function defined on particles which has the following formula:

$\nabla_{p_k} C_i = \frac{1}{\rho_0} * \sum_j \nabla_{p_k} W(p_i - p_j, h)$. This can have two variations depending on whether k is a neighbour of the current particle or not. If k is a neighbour of the current particle the gradient of the constraint is the same as the general formula for the gradient of a function defined on particles. When particle k is not a neighbour of the current particle the formula becomes $\nabla_{p_k} C_i = \frac{-1}{\rho_0} * \nabla_{p_k} W(p_i - p_j, h)$ (Monaghan 1992).

(**) => $\Delta p_i = \frac{1}{\rho_0} * \sum_j (\lambda_i + \lambda_j) * \nabla W(p_i - p_j, h)$ which represents the position correction used in the position based dynamics framework

(***)

The artificial pressure term mentioned in the Problems section is specified in terms of the smoothing kernel itself.

$S_{correction} = -k * \left(\frac{W(p_i - p_j, h)}{W(\Delta q, h)}\right)^n$ where Δq is a point inside the smoothing kernel radius, usually between 0.1f * h and 0.3f * h, k and n are small positive constants (Macklin 2013 - Slides).

This term represents the position correction and it will be included in the position update step:

$$\Delta p_i = \frac{1}{\rho_0} * \sum_j (\lambda_i + \lambda_j + S_{correction}) * \nabla W(p_i - p_j, h)$$

For the current implementation viscosity is implemented as a velocity post-process and it is important for coherent motion.

$$v_i \leftarrow v_i + c * \sum_j v_{ij} * W(p_i - p_j, h) \; with \; v_{ij} = v_i - v_j \; and \; c \; typically \; 0.01$$ (Macklin 2013).

3.3 Deformable bodies – Shape matching

3.3.1 Introduction

Classic position based dynamics methods are well suited for simulating thin shells like cloth but less suitable for simulating volumetric objects using distance constraints (University of Pennsylvania 2014). The accurate way of simulating deformable bodies is using finite element methods. Unfortunately these are complicated to implement and slow for real-time applications.

3.3.2 Advantages of using shape matching

Shape matching offers several advantages over other methods of simulating deformations. These advantages can be split into three categories:

1. Efficiency – unlike other methods which use slow implicit integration, shape matching is fast and simple to compute and it does not require implicit integration. Also the method is not based on a specific object representation like finite element method which uses tetrahedrons and requires a lot of preprocessing (Muller 2005). Also the method does not require any connectivity information between the particles.
2. Stability – Shape matching offers unconditional stability which is a must for interactive application such as video games.
3. Controllability – This method allows the developers to easily control the magnitude of the deformations by altering the stiffness parameter α.

3.3.3 Disadvantages

The method has also a couple of disadvantages. As mentioned above, it works by enforcing a positional constraint, but it does not fit exactly into the position based dynamics framework. Also, because the particles have no connectivity information, using this method can create some difficulties when coupling the deformable body with other deformable bodies or fluids triggering phenomena like tunnelling, interpenetration and locking.

3.3.4 Method description

Shape matching is a meshless method which has no connectivity between the particles (Muller 2005). This implies that the internal forces need to be handled to mimic the behaviour of a soft body.

The main idea of shape matching is to find an optimal translation and rotation of the rest pose shape which after being applied to the rest pose configuration will move the points to the current state as close as possible (University of Pennsylvania 2014).

Rest pose Current pose

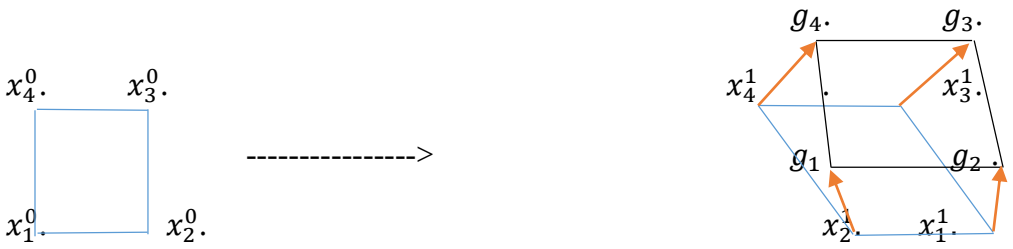
1. 2. 1'. 2'.

---------->

3. 4. Simulate using external force 3'. 4'.

E.g. gravity

Rest pose Current pose and transformed pose



$x_1^0, x_2^0, x_3^0, x_4^0$ – rest pose of point cloud configuration

$x_1^1, x_2^1, x_3^1, x_4^1$ – current pose – the state of the particles after applying the external forces

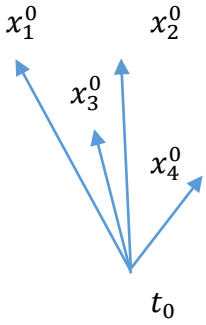
g_1, g_2, g_3, g_4 – goal positions - the closest transformation of the current pose to the rest pose obtained by applying a translation t and a rotation matrix R which needs to be orthonormal and to have the determinant equal to 1.

Once completed the translation and rotation, the goal positions can be calculated. Once these are calculated, the internal restorative forces can be simulated by making the particles in the current state move towards their corresponding goal position g (Muller 2005).

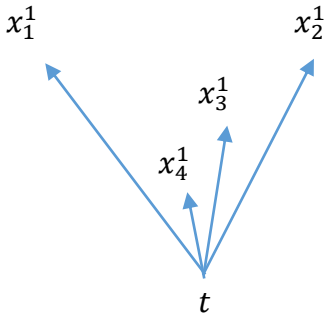
The rotation and the translation are calculated by minimizing the following sum:

$\sum_i W_i(R * (x_i^0 - t_0) + t - x_i^1)^2$ where t and t_0 represent translation points and R the rotation matrix.

Arbitrary point cloud



Target point cloud



Let $(x_i^0 - t_0) = q_i$ and $(x_i^1 - t) = p_i$ => we need to find the optimal rotation and translation that best aligns the above vectors from the arbitrary cloud configuration to the vectors in the target cloud configuration (University of Pennsylvania 2014).

Use these notations the sum becomes $\sum_i W_i(R * q_i - p_i)^2$

The minimization problem can be solved by calculating the derivative of this sum and then setting it to be equal to zero (University of Pennsylvania 2014). From there, one intuitive solution for the translation points is:

$t_0 = \frac{\sum_i m_i * x_i^0}{\sum_i m_i}$ which is the center of mass of the initial point cloud

$t = \frac{\sum_i m_i * x_i^1}{\sum_i m_i}$ which is the center of mass of target point cloud

There are multiple solutions for t and t_0 but the most straight forward solution is represented by the centers of mass (Muller 2005).

The next step is to calculate the optimal rotation matrix R. Finding the optimal rotation starts with solving a more relaxed problem, finding a 3x3 matrix A with no restrictions such that $\sum_i m_i(A * q_i - p_i)^2$ is minimized (University of Pennsylvania 2014). By using the same idea, to differentiate and then set it to be equal to zero, the matrix A can be expressed as:

A = $(\sum_i m_i * p_i * q_i^T) * (\sum_i m_i * q_i * q_i^T)^{-1} = A_{pq} * A_{qq}^{-1}$ (Muller 2005).

Having calculated the rotation matrix A, matrix R can be calculated by minimizing the Frobenius norm $\|A - R\|^2$. This can be achieved by using singular value decomposition applied on matrix A which represent the best linear transformation (University of Pennsylvania 2014).

A $\overset{\text{polar decomposition}}{=}$ R * S, where R is orthonormal with determinant 1 and S is a symmetric matrix.

=> R = $A * S^{-1}$

Once the optimal rotation is calculated, the goal positions can be written as:

$g_i = R * (x_i^0 - x_{cm}^0) + x_{cm}$ (Muller 2005)

Based on the goal positions, the time step integration is done to calculate the velocity and to update the positons of the particles:

$$v_i(t + \Delta t) = v_i(t) + \alpha * \frac{g_i(t) - x_i(t)}{\Delta t} + \Delta t * \frac{f_{ext}(t)}{m_i}$$

$$x_i(t + \Delta t) = x_i(t) + \Delta t * v_i(t + \Delta t),$$

where α represents the stiffness parameter with $\alpha = 1$ being as rigid as possible and $\alpha = 0$ simulating a body with no internal forces.

The term $\frac{g_i(t) - x_i(t)}{\Delta t}$ can be seen as a special force which pushes the particles to the position where they would go if the object was rigid (University of Pennsylvania 2014).

The method described here is similar to position based dynamics in the sense that it integrates the acceleration by using explicit Euler, but then instead of finding a position correction which is simply applied to the current position, it simply readjusts the new position based on the translation and rotation found to match the goal position as good as possible.

3.4 Soft-body – fluid coupling

3.4.1 Introduction

Having both the fluid simulation and the deformable body simulation implemented using particles which extend from a common base particle type, makes the collision detection and collision response easier to handle. The main reasons for choosing particles as building blocks for both simulations is that this approach reduces the number of collision types that need to be handled to one (Muller 2014) and also particles are flexible enough to model both phenomena.

3.4.2 Collision detection and response

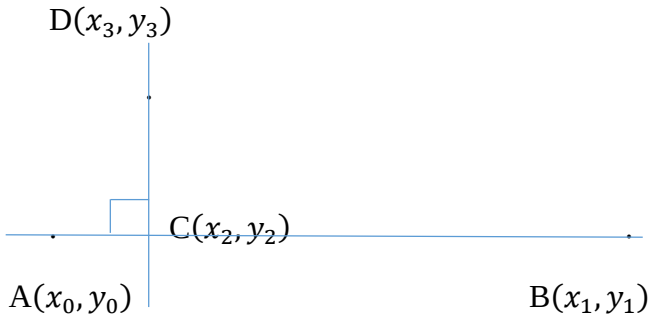
As the particles are represented by circles which are characterized by position, velocity and radius, the only type of collision that needs to be handled is circle-circle collision.

As opposed to static collision detection, continuous or dynamic collision detection provides more accuracy and solves the problem of circles moving through other circles when the velocity update between two consecutive integration steps is larger than the diameter of the circle (Muller 2014).

To address this issue, continuous collision detection is used in the current implementation. Continuous collision detection works by detecting a collision before it happens and correcting it straight away (Muller 2014). To achieve this, the algorithm needs to find the closest point on a segment line to another point which is not on the segment line. So, given a segment line determined by two points A(x_0, y_0) and B(x_1, y_1) and an exterior point D(x_3, y_3), we need to find C$(x_2, y_2) \in$ AB, where C is the intersection of the segment line AB with the perpendicular on AB which goes through point D.

The equation of the line determined by the two points A and B is:

$$x * (y_1 - y_0) + y * (x_0 - x_1) = x_0 * y_1 - x_1 * y_0$$



Let $y_1 - y_0 = a$, $x_0 - x_1 = b$ and $x_0 * y_1 - x_1 * y_0 = c_1$ which make the above equation

a * x + b * y = c_1 (*) =>

The equation of the perpendicular on the segment line AB is –b * x + a * y = c_1. This perpendicular needs to pass through the exterior point D => -b * x_3 + a * y_3 = c_2 (**).

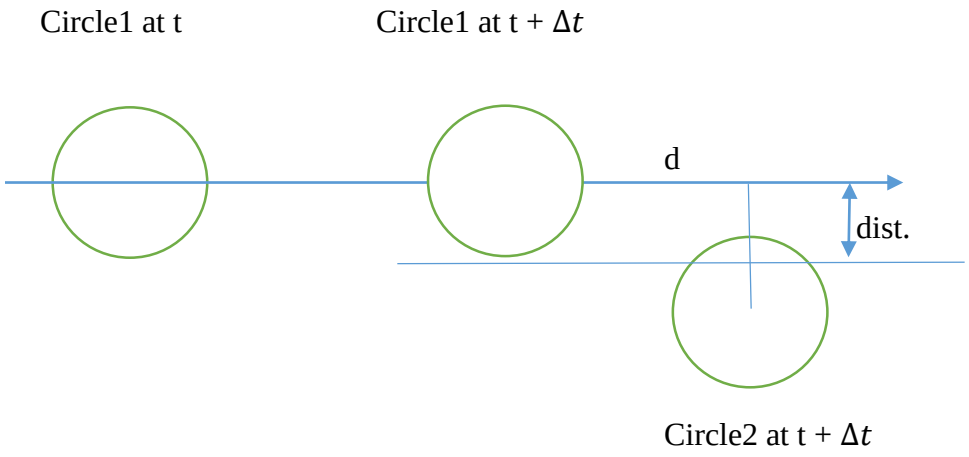
Now we have to find the intersection of the two lines, which represents the closest point on the line AB to the point D by solving the following system:

$$\begin{cases} a * x + b * y = c_1\ (*) \\ -b * x + a * y = c_2\ (**) \end{cases}$$

The determinant of the system is d = $\begin{vmatrix} a & b \\ -b & a \end{vmatrix} = a^2 + b^2$. If d = 0 then the point D ∈ AB and the point we are looking for is exactly D. If the determinant is different than zero the coordinates of the point we are looking for are (Ericleong.me 2015):

$$\begin{cases} c_x = \dfrac{a * c_1 - b * c_2}{d} \\ c_y = \dfrac{a * c_2 - b * c_1}{d} \end{cases}$$

Using the above algorithm, continuous circle-circle collision detection can be easily implemented.



The first step is to find the closest point on the segments line determined by the trajectory of the circle1 to the centre of the circle2 – d. If the distance (dist.) between point d and the centre of circle2 is less than the diameter of the circle, then a collision will occur. The trajectory of circle1 is determined by the current position and its next position based on its velocity (Ericleong.me 2015).

Once the collision is detected, the position based dynamics framework is used to correct the current position of the colliding particles. To handle collisions, the following distance constraint is used C(p_1, p_2) = $|p_1 - p_2|$ – d, where p_1, p_2 are

the centers of the two circles and d represents the diameter of the circle (Macklin 2013).

From the general expression of position correction in position based dynamics we have

$\Delta p_i = -ct * w_i * \nabla_{p_i} C(p_1, \dots, p_n)$ (*)

The derivatives of the constraint function with respect to the points p_1 and p_2 are:

$\nabla_{p_1} C(p_1, p_2) = n$ (**)

$\nabla_{p_2} C(p_1, p_2) = -n$, where n is the collision normal n = $\frac{p_1 - p_2}{|p_1 - p_2|}$.

The scaling factor for position based dynamics $\lambda = \frac{-C(p)}{\|\nabla_p C(p)\|^2}$ can be written as

$\lambda = \frac{-C(p)}{\sum_i w_i * \|\nabla_p C(p)\|^2} = -\frac{|p_1 - p_2| - d}{w_1 + w_2}$ (***), because the length of the gradient of the constraints is one.

The final position correction for the two colliding circles in position based dynamics is:

(*)

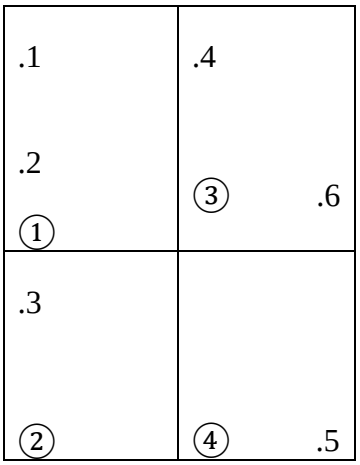
(**) => $\begin{cases} \Delta p_1 = -\frac{w_1}{w_1 + w_2} * (|p_1 - p_2| - d).n \\ \Delta p_2 = \frac{w_1}{w_1 + w_2} * (|p_1 - p_2| - d).n \end{cases}$ (Jakobsen 1999)

(***)

3.4.3 Spatial hashing

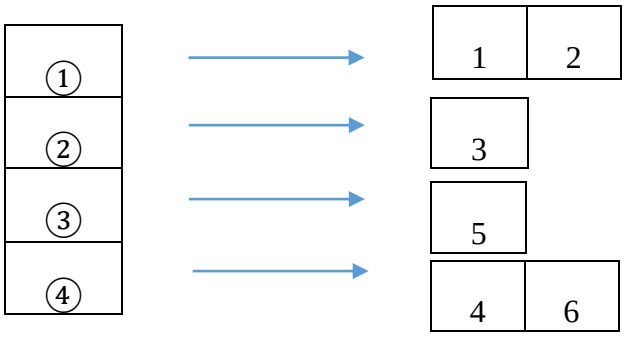
Because the number of collision checks per update step has a complexity of O (N^2), where N is the number of particle in the simulation, this can easily become a performance bottleneck as the number of particles increases.

To reduce the numbers of collision checks, a spatial hashing algorithm was implemented. Spatial hashing works by projecting a 2D area into a 1D hash table. The entire area of the simulation is divided into rectangular sub-areas called cells. Each cell has a unique hash id and it behaves like a container which can store references to particles (The mind of Conkerjo 2009).



2D area divided into cells

Each cell contains a certain number of particles



The spatial hashing algorithm uses a dictionary as main container. The keys are represented by the hash value which is calculated as the index of the cell in the 2D array and the value is the list of particles in that cell.

This means that in each update step, the number of collision checks per particle is reduced to the number of particles in the current bucket.

There are loads of debates over the performance of the spatial hashing algorithm used for collision detection. One possible alternative would be using quadtrees. Depending on the nature of the objects checked for collision, the performance of the two algorithms can vary considerably. If the dimension of the objects varies, the quadtree algorithm gives better performance (Christer 2014).

In the implemented simulation, the particles from both simulations have the same radius which motivates the decision of using spatial hashing over quadtrees. The disadvantage of spatial hashing is represented by the fact that it relies heavily on the performance of the data structure used to store the keys and the buckets (Tristam 2009). Also the cell dimension in the spatial hashing

algorithm can influence the overall performance of the algorithm (Tristam 2009).

3.4.4 Soft-body – fluid two-way coupling

To handle the interaction between fluid and deformable particles, the framework implements a combination of continuous collision detection, a position based dynamics solver for collision response and fluid density influenced by the presence of deformable particles inside the fluid smoothing radius. The interaction between the two is handled differently for the soft-body and the fluid.

Soft-body -> Fluid

This section discusses the influence of the soft-body particles on fluid particles. As the framework does not rely on any forces to resolve collisions, the fluid simulation uses the soft-body particles in the local fluid density estimation (Macklin 2013). This makes the fluid density estimation add a weighted sum of kernel function values calculated only based on soft-body particles.

The initial expression for fluid density estimation $\rho_i = \sum_j W(p_i - p_j, h)$ becomes

$\rho_i = \sum_{fluid\ particles} W(p_i - p_j, h) + s * \sum_{soft-body\ particles} W(p_i - p_j, h)$.

The new term represents the local density of the soft-body particles based on all deformable particles which exist in the fluid smoothing radius h and is calculated using the same kernel function as the standard fluid density estimator, Poly6 (Macklin 2013).

The parameter s is used to compensate for any difference in particle density between the two simulations when particles are at rest. Ideally, if at rest, the distance between particles is the same in both fluid and the soft-body, then a value of 1 for the parameter s should be used (Macklin 2013). If the deformable particle density is higher than the fluid particle density than a value less than 1 should be used for s.

The resulted effect is characterized by a dissipative behaviour of the fluid particles when they get closer to soft bodies as the local fluid density increases locally. This acts like a small force which pushes the fluid particles further from the soft-body particles.

Fluid -> Soft-body

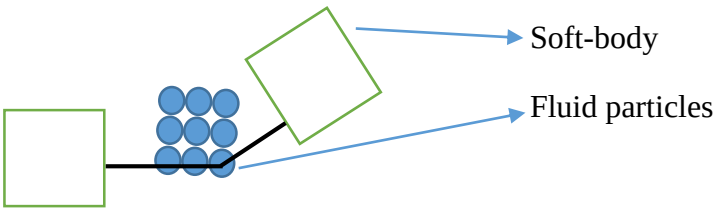
When the fluid particles collide with the soft-body particles, the framework treats them both as solid particles and handles the collision response using the position based dynamics solver and uses the smoothing distance as contact distance (Macklin 2013). However, this implementation suffers from problems like interpenetration, tunnelling and locking.

Because the soft-body is meshless and the particles which form it are not connected, the fluid particles can push apart the soft-body particles and get stuck between these. If the velocity of the fluid particle is too high, then these can completely go through the soft-body creating a tunnelling effect.

To address this, the idea of a signed distance field which can be sampled per particle basis is used. A distance field is a scalar field that measures distances from a given point to an object (Michael 2008). The distance field is useful as it can incorporate multiple pieces of information like the closest object, the penetration depth and the collision normal. The penetration depth is represented by the minimum translation distance (MTD) while the direction of the minimum translation distance represents the collision normal (Michael 2008).

For each particle in the fluid simulation we calculate and store the minimum translation distance, which represents the smallest distance to any of the edges of the deformable objects existing in the simulation (Michael 2008).

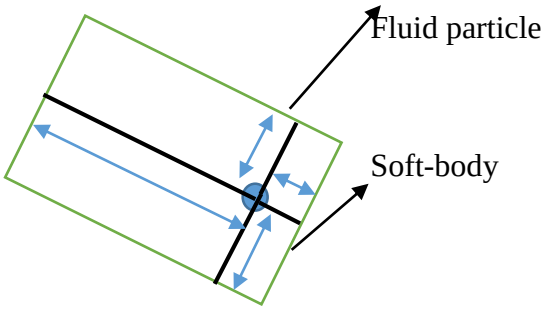
If the fluid particle has not penetrated any soft-body (the centre of the particle falls outside the convex hull of the soft body) then the minimum translation distance can be seen like this:



This situation is only relevant if the maximum penetration depth of the fluid particles is less than the particle radius. Also we are only interested in distances which fall within the segment line determined by the soft-body edge and not the extension of it.

If the fluid particle is inside the soft-body, the minimum translation distance can be calculated only by comparing distances to the edges of the colliding soft body.

For a rectangular shape soft-body the minimum translation distance is defined as MTD = min (D1, D2, D3, D4)



D1, D2, D3 … Dn represent signed distances which are negative for particles which are inside the soft body and positive otherwise.

Once the minimum translation distance was found, we calculate the vector from the centre of the fluid particle to the closest edge and store its gradient. This value will be used later as the collision normal between the fluid particle and the soft-body (the fluid particle will be pushed outside the soft-body towards the closest edge). So, given the current setup we have the flexibility of choosing a maximum penetration depth. Every particle with a signed distance less than that will be projected outside of the soft-body using an iterative scheme similar to the position based dynamics solver.

Given the collision normal $n_{ij} = \nabla(signed\ distance\ field)$, the position correction at each iteration step for particle i will be $\Delta p_i = -s * (MTD * n_{ij})$ (Macklin 2013). This pushes the particle outside the soft-body, until the position satisfies the collision depth check. The constant s acts as a stiffness parameter $\in [0..1]$, where a value of 1 will project the position to match the last valid one.

3.5 Optimizations

3.5.1 Introduction

To take advantage of the entire CPU power available, a parallel implementation of the fluid solver using multithreading was used. The decision is motivated by the nature of the fluid simulation which is highly granular and allows a straight forward task splitting process.

The implementation is designed to work using a custom number of threads based on the hardware the application is supposed to run on and uses CPU threads to achieve parallelism.

3.5.2 Implementation

During the development of the framework, three different approaches have been attempted to make the fluid solver parallel. For the first two, only the C++ standard library was used while the last one used a combination of Boost library and standard C++.

The first approach was using mutexes and conditional variables to prevent race conditions. This method proved to be very slow offering lower performance than the single threaded version due to the bottleneck created by the mechanism of locking and unlocking mutexes.

The second approach was to restrict the use of concurrency in such way so it does not need to handle race conditions and consequently executes tasks in parallel. In terms of performance, this got very close to the single thread approach which was still not satisfactory. After profiling the application, it was found out that the thread creation became the bottleneck, the process being extremely costly.

After these unsuccessful attempts, it was obvious that using locking mechanisms and thread creation was not suitable for real-time application loops. To avoid using these, a thread pool was used to reuse the existing threads while still using the approach presented in the second method to avoid the need of locking mechanisms.

In order to avoid the need of locking mechanisms, only the following computations have been done in parallel: calculating the lambda value, calculating the constraint value, calculating the position displacement and the minimum translation distance. For the thread pool, the threadpool type in the Boost library was used.

To be able to reuse threads, four auxiliary methods have been created (one for each of the above mentioned calculation) which take an interval of indices as an input and handle a subset of the initial set of particles based on the number of threads available. This methods represent the base on which four lists of tasks are created.

Setup thread pool

Foreach thread

Calculate step = particleCount / threadCount

*startIndex = step * currentThreadIndex*

*endIndex = step * (currentThreadIndex + 1)*

LambdaTaskList.add (bind (AuxComputeLambda (startIndex, endIndex)))

PositionCorrectionTaskList.add (bind (AuxPositionCorrection (startIndex, endIndex)))

ConstraintTaskList.add (bind (AuxConstraint (startIndex, endIndex)))

MTDTaskList.add (bind (AuxComputeMTD (startIndex, endIndex)))

In the main loop, the thread pool schedules these tasks from each list based on the threads available to be used in the thread pool:

Foreach currentTask in LambdaTaskList

threadPool schedule currentTask

threadPool waits for tasks to finish

4. RESULTS AND ANALYSIS

4.1 Data gathering

4.1.1 Test system

All the tests were conducted on the same system with no additional workload on the CPU. The application runs entirely on the CPU doing both the physics update and rendering the particles. The rendering, represents approximately 20% CPU workload difference which resulted when testing the simulation with the rendering on or off.

The test system has the following specifications:

AMD FX-6300 six core CPU 3.0 GHz
NVIDIA GeForce 650TI Boost GPU
8GB Dual Channel DDR3 1600 MHz
Windows 8.1 64bit

The demo application was developed and built using Visual Studio 2013 and uses the SFML library for rendering, glm for vector/matrix manipulation and the Boost library for multithreading.

4.1.2 Simulation parameters values

For the testing process, a series of predefined parameter values were set:

For the spatial partitioning algorithm, the size of each cell was chosen to be four times the particle radius. This value was chosen by conducting experiments with different values and noticing how the neighboring requirement is satisfied based on the cell size. As properties for the base particle, a radius of 3.0 and a particle mass of 2.0 was chosen. For the position based dynamics framework, the number of iteration for the solver was 3 while the stiffness parameter 0.1.
The fluid is simulated using viscosity an artificial pressure term and collision with the container handled by the position based dynamics framework. The kernel smoothing distance was chosen to be equal to the cell size value used for the spatial partitioning. The viscosity of the fluid set to 1.0 but this is very flexible and allows a wide range of values to be set based on how the fluid needs

to behave. For the velocity damping, a very simple model was chosen. During each update step, the velocity of the fluid particle is dampened using a value of 0.999. For the particle position relaxation parameter a value of 0.00001 was chosen. This play a vital role in the stability of the simulation and needs to be chosen based on other parameters like particle mass, particle radius, smoothing distance and cell size.

For testing purposes, the simulation uses a fixed delta of 1/60s, but allows a wide range of values to be used, the simulation remaining stable even at higher time steps. The time step of the simulation and the parameters of each simulation are mutually dependent and they might need additional tweaking to handle changes on either side.

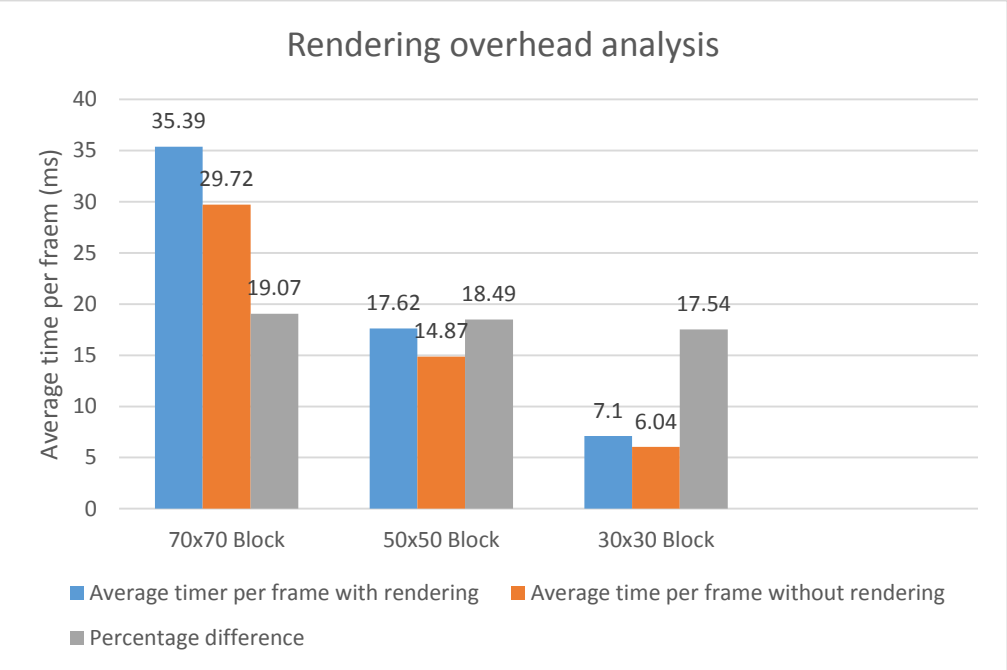
4.1.3 Performance analysis tool

The performance data is generated from the program itself. It uses a high precision clock class available in the SFML library to measure the time elapsed between two consecutive render calls. Using this information, it calculates the number of frames per second (FPS), minimum, maximum and average FPS as well as the minimum, maximum and average time per frame.
The duration of the benchmark can be customized and the average time per frame is calculated as the duration of the benchmark divided by the total number of frames rendered in the set time interval.
The benchmark can be set to start automatically when the application is run, in which case will run for the specified time interval, or can be started and ended manually while running the application.
When the benchmark has ended either by manually stopping it or when the time interval has ended, the information gathered is printed in the console attached to the application or written in the "BenchmarkResults.txt" file in the root of the project.

4.1.4 Consistency and additional considerations

To achieve consistency between tests and make the results as relevant as possible, the same set of parameters (particle, spatial partitioning, soft-body and fluid) will be used during all the tests.
The first benchmark, simulates only the fluid with no interaction, this test's purpose being to find an optimal base configuration so that when tested in conjunction with the soft-bodies, the CPU workload created by the fluid simulation is at minimum.
The fluid is created as a square dam with particles equally distributed and with no initial velocity. The spawn position is consistent between each application

session which ensures the same simulation is obtained every time the application runs (assuming the same number of particle is used).
Each soft-body is represented as a square block of 6x6 particles equally distributed. The soft-bodies are spawned in a row at the top of the container uniformly distributed with no initial velocity, linear or angular momentum. This, similar to the fluid simulation ensures that the same simulation is obtained giving the fact that the soft-bodies are the same size.



The above graph presents the CPU overhead created by the rendering part of the application. The data is extracted only based on fluid rendering, as the fluid has the major contribution to the amount of time per frame spent rendering.

The percentage difference between the version which does the rendering and the one which doesn't, stays relatively constant when using various fluid dam sizes which helps achieving the consistency and accuracy of the testing.

4.2 Interaction behaviour

Dynamic collision detection is not yielding the desired behaviour. Tunnelling of the soft-body particles through other soft-body particles still occurs. Also soft-bodies can occasionally lock into each other due to the missed collision detection.

When the pressure created by the fluid particles becomes too high (due to their rapid movement and high local density) the soft-body particles oscillation amplitude increases which creates gaps on their boundary. The bigger the amplitude of the oscillation is, the higher the change of a fluid particle getting inside the soft-body or the soft-body interpenetrating another soft-body is.

4.3 Testing – benchmarks

4.3.1 Fluid simulation – benchmark results

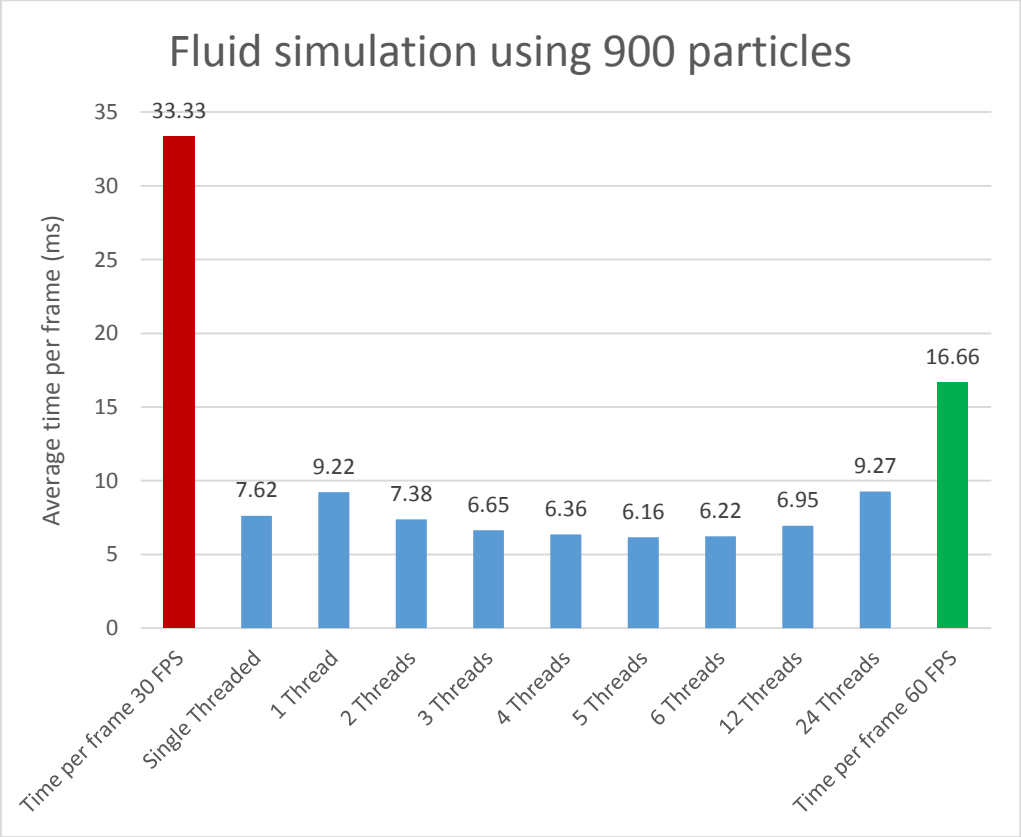


Fig. 1

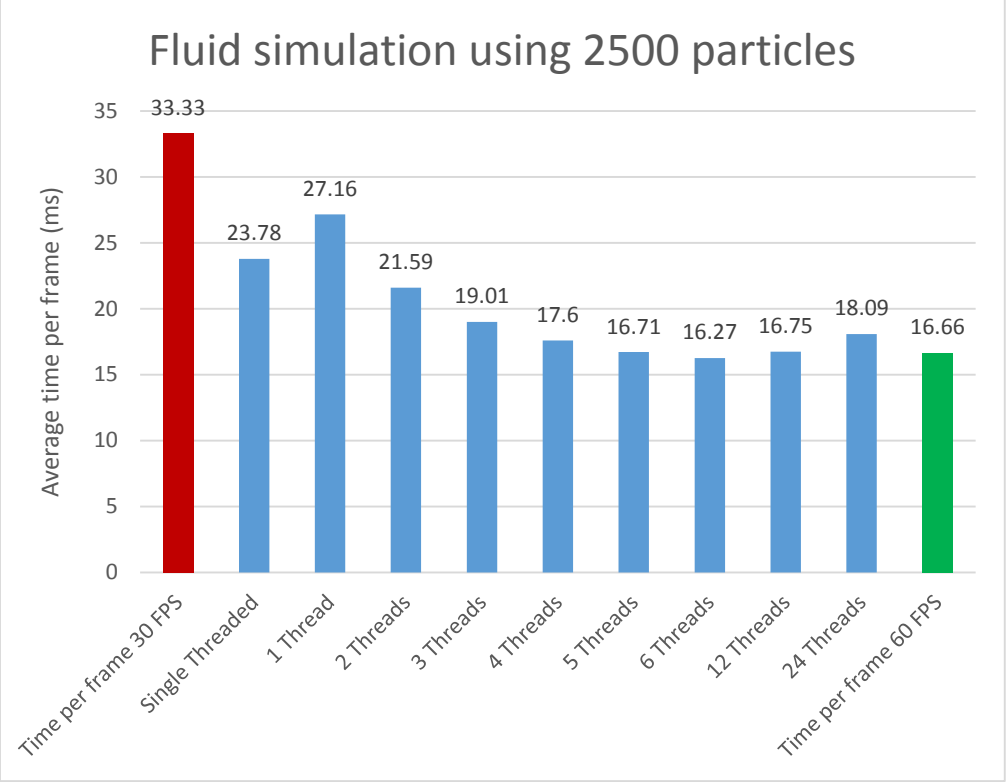

Fluid simulation using 2500 particles
Average time per frame (ms)
35
30
25
20
15
10
5
0
33.33
23.78
27.16
21.59
19.01
17.6
16.71
16.27
16.75
18.09
16.66
Time per frame 30 FPS
Single Threaded
1 Thread
2 Threads
3 Threads
4 Threads
5 Threads
6 Threads
12 Threads
24 Threads
Time per frame 60 FPS

Fig. 2

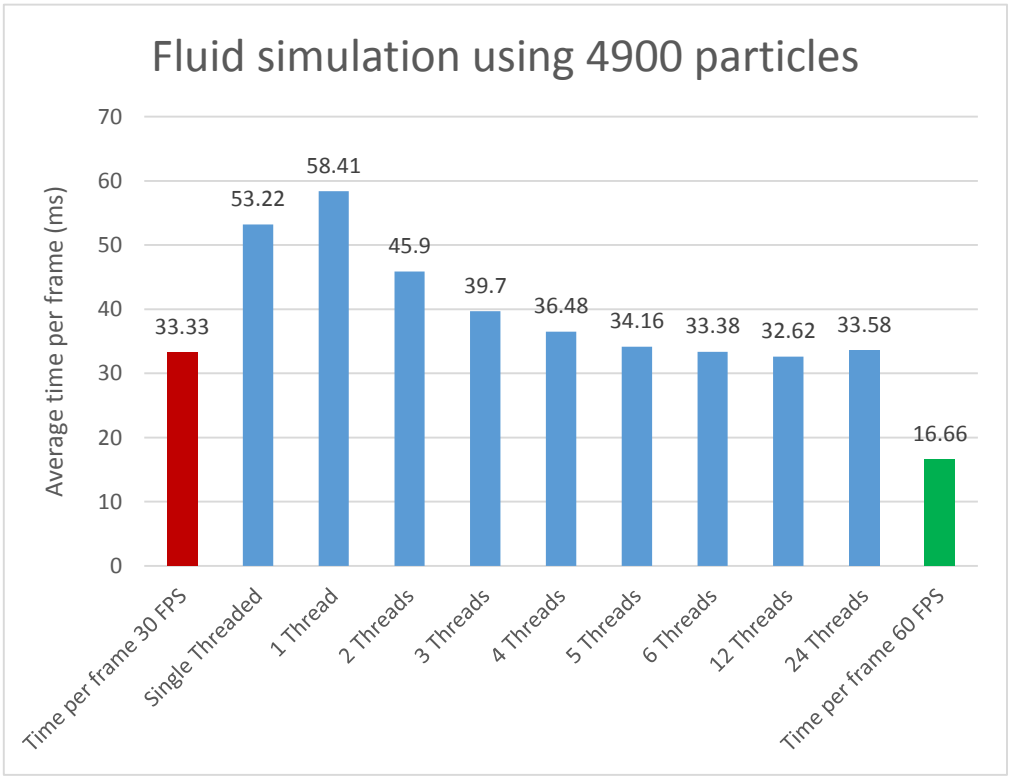

Fluid simulation using 4900 particles
Average time per frame (ms)
70
60
50
40
30
20
10
0
33.33
53.22
58.41
45.9
39.7
36.48
34.16
33.38
32.62
33.58
16.66
Time per frame 30 FPS
Single Threaded
1 Thread
2 Threads
3 Threads
4 Threads
5 Threads
6 Threads
12 Threads
24 Threads
Time per frame 60 FPS

Fig. 3

4.3.2 Fluid – soft-body interaction – benchmark results

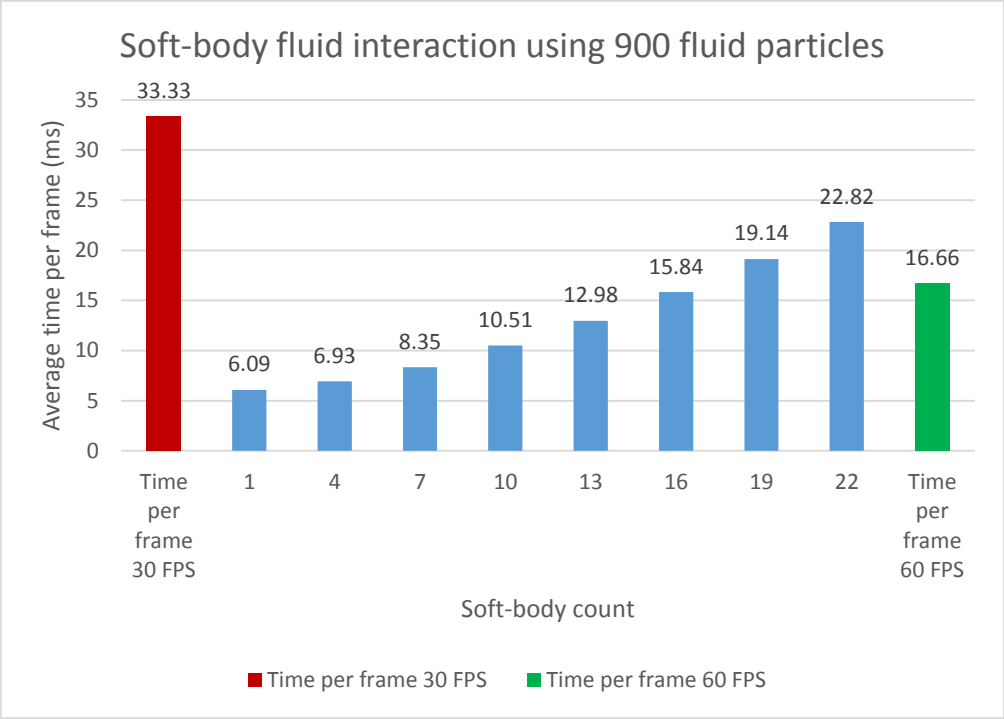


Fig. 4

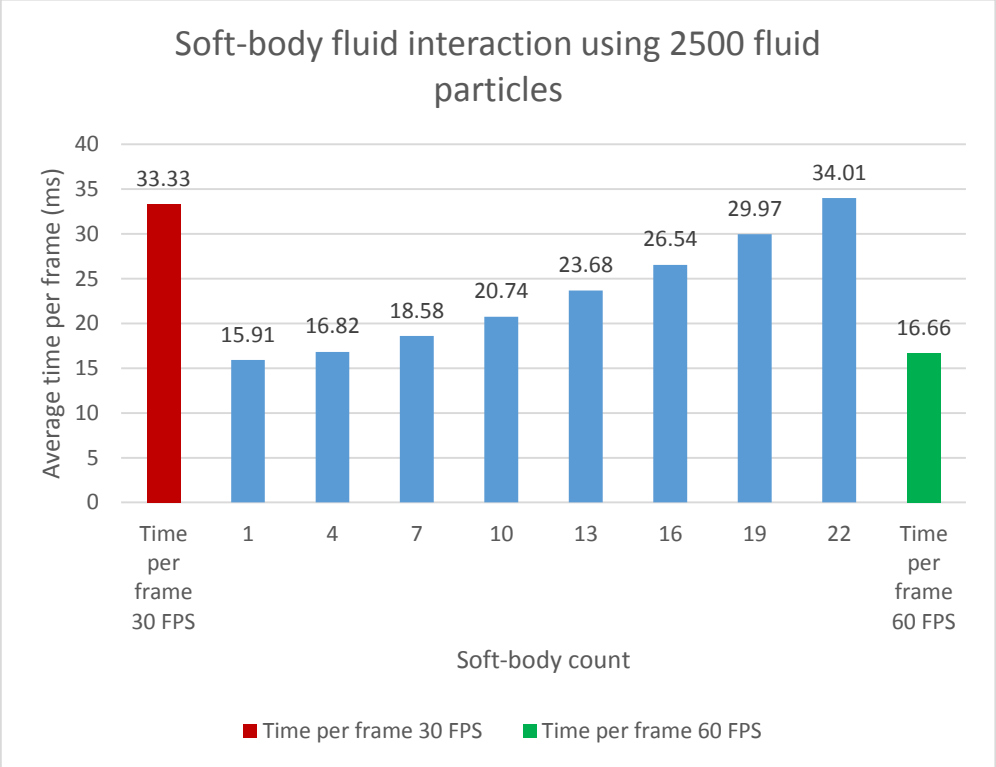


Fig. 5

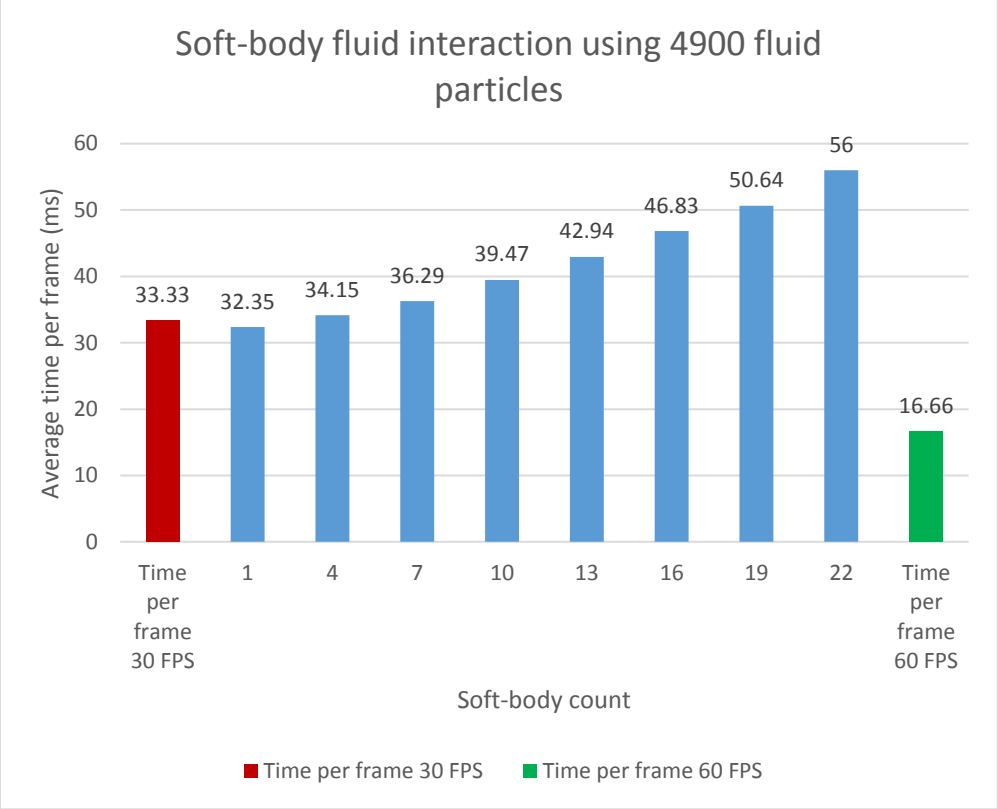


Fig. 6

4.4 Data analysis

4.4.1 General

The performance analysis is done based on the average time per frame calculated in a fixed time interval of 30 seconds from the start of the application. The graphs showed in Fig. 1-3 display a dependency of the fluid simulation average frame time on the number of threads the application runs on.

Each graphs display two reference times per frame which are usually of interest in the game development industry – 33.33ms which is required for a frame rate of 30 and 16.66ms for a frame rate of 60. These are meant to help the reader to easily understand where the performance of the algorithm is.

The parameters used for testing were chosen so that the application is pushed to the limits imposed by the real-time requirement but also to get a good perspective on how the performance scales when varying different parameters.

4.4.2 Fluid performance analysis

The graphs presented in Fig. 1-3 illustrate the performance of the fluid simulation single threaded while running on the main thread and multithreaded

using 1, 2, 3, 4, 5, 6, 12 and 24 threads. These values were chosen as the testing platform utilizes a six core CPU.

As expected, running the simulation in the main thread yields good performance compared to running on a custom created thread. This is due to the fact that the thread creation and starting as well as the context switching are very expensive processes in terms of CPU power.

The performance of the single threaded version, is very close to the performance of the application running on two cores. This, again demonstrates the high cost of thread scheduling. The difference in performance starts to increase as the number of particles increases, the multithreaded version becomes more efficient than the single threaded one. The performance boost of the 2 thread version over the single threaded one increases from 3.2% for 900 fluid particles, to 10.1% for 2500 fluid particles and finally to 15.9% for 4900 fluid particles.

According to the tests, the 6 core multithreaded version obtains the best overall results. Again, the performance gain compared to the single threaded version increases as the number of particles in the simulation increases. The performance boost of the 6 thread version over the single threaded one increases from 22.5% for 900 fluid particles, to 46.15% for 2500 fluid particles and finally to 59.43% for 4900 fluid particles.

The performance gain is still smaller than expected for a six thread implementation over a single threaded one. This can be justified considering the fact that only four calculations were made in parallel as explained in the chapter 4.5.

Increasing the number of threads available above the number of core in the CPU lowers the overall performance of the application due to the cost of thread starting and context switching.

4.4.3 Fluid – soft-body interaction performance analysis

The graphs presented in Fig. 4-6 benchmark the performance of the interactions between the fluid and soft-bodies. There are three different configurations for the fluid with 900, 2500 and 4900 particle, these being simulated using the six thread version of the application.

The dependency illustrated shows how the average frame time varies based on the number of deformable bodies the fluid has to interact with.

To evaluate the performance of the fluid – soft-body interaction, the 2500 fluid particle configurations with no soft-bodies was chosen for a base value of the average time per frame. Then using the results from 2500 fluid particle interacting with soft-bodies, the amount of time required for each extra soft-body added to the simulation was calculated. For 4 soft bodies the frame time increased by 0.55ms which gives an average of 0.13ms per extra soft-body.

Going further, and calculating the time required to simulate more soft-bodies gives the following results: 0.44ms per soft-body for 10 soft-bodies, 0.64ms per soft-body for 16 soft-bodies and finally 0.8ms per soft-body when simulating 22 soft-bodies. Ideally, this should have yielded a similar overhead per soft-body, but the interaction between the two simulations does not scale as expected.

The justification for this is represented by the fact that the soft-body – soft-body collision detection has a quadratic complexity O (N^2) and does not allow the application to scale linearly.

4.4.4 Conclusion

The performance of the fluid simulation alone can be improved by transforming more code to execute in parallel. As it currently is the fluid can almost handle 4900 particle at 30FPS which is a good achievement for running the simulation on the CPU.

A high difference between the maximum and the minimum update time per frame was noticed during testing. This can be considered a disadvantage when using the framework implementation in a real-time application such as a video game as it can create stuttering. The effect is not noticeable during the test so it does not represent a big problem.

The soft-body part of the simulation can be easily improved and once the collision issue is resolved, the performance should scale linearly with the number of soft-bodies.

According to the results obtained, the aim of this project was achieved, the application being able to simulate the interaction of the fluid simulation and the soft-bodies at real-time frame-rates and with a reasonable amount of detail.

5. EVALUATION

5.1 Limitations

Currently the interaction between the deformable particles does not use the spatial partitioning algorithm. The check for collision is made in a brute-force manner. This adds a lot of CPU overhead and penalizes the performance considerably as the number of particles increases. This can be solved by moving the spatial partitioning update step in the particle manager rather than inside the fluid simulation as it currently is. This would allow soft-bodies to enquire for the set of neighbours easily.

The parameter tweaking process is slow and difficult as there is very little control from inside a running instance of the application. Each parameter change, apart from the fluid viscosity and fluid damping value require the entire application to be rebuilt. In order to solve this, a larger set of parameters should be accessible from the user interface. An implementation for this already exists in the framework, but due to the limited time only the viscosity and damping parameters were added.

The interaction with the fluid is limited to the collision against the container's walls and the interaction with the soft-bodies. The fluid does not react to any forces induced by the user, this limiting its controllability. One solution to this problem can be to add virtual particles locally and include them in the particle density calculation as a neighbour of the current particle. This would create a repulsion force which can push the particles away from the application point.

The parameters for both simulations as well as the spatial partitioning and particle properties strongly rely on each other. This represents a well-known problem in Lagrangian fluid simulations and requires a lot of attention when modifying any parameters.

5.2 Future work

One way to further improve the performance of the application considerably is to implement a GPU parallel fluid solver. This can be implemented either by using a specialized API like CUDA or OpenCL or by using shader's compute capability. This feature would be used only by the hardware which is able to run it and fall back to the multithreaded implementation otherwise.

The current work can be extended by finalising the implementation of the marching squares algorithm for rendering the fluid and using Bezier curves to represent the contour of the soft-bodies. The implementation currently has these features implemented but they need to be refined and optimized so the performance penalty is minimized.

Another feature which would increase the productivity when using the framework is the inclusion of a fully featured user interface which would allow to tweak the parameters on the fly without having to recompile the project. This is also motivated by the well-known fact that the particle based simulations are difficult to tweak due to their high number of parameters.

Implementing a more customizable framework in terms what constraints are being used and taking advantage of the position based framework to incorporate them, would allow the creation of a wider range of effects in a unified, particle based way. This would benefit from reusing the existing code as well as treating the collisions in a unified way.

5.3 Conclusion

After conducting all the testing and analysing the results produced, we can say that the current implementation improves on the most common approaches used for simulating Lagrangian fluids and deformations like smoothed particle hydrodynamics (Macklin 2013 - Slides) and finite element method (Muller 2005) both in terms of performance and stability . The trade-off was made in terms of accuracy of the simulations, but still the framework provides a lot of flexibility and careful parameter tweaking can offer believable results.

The performance of the simulation is good, allowing a certain level of real-time interactivity, based on the amount of detail and the complexity of the simulation. The current framework offers a solid base from where the points suggested in the Future work section can be considered to further expand and optimize the framework.

This approach of simulating multiple phenomena, in the same environment, in a particle based unified way, has already proven to be a very popular and viable choice in offline rendering tools like Nucleus in Maya or Lagoa made by SoftImage. With the integration of the FleX framework based on position based dynamics into Unreal 4, Nvidia makes the first step towards making the real-time physically based animation a common occurrence in the games industry.

6. APPENDIX

6.1 Application

The current submission contains two release builds. A 32bit version of the application and a 64bit of the application.

The location of the executable which should be runnable without any changes are \Build\32bit\2DFluidSoftBodiesInteraction for the 32bit version and \Build\64bit\2DFluidSoftBodiesInteraction for the 64bit version.

The current build simulates 1600 fluid particle and 5 soft-bodies each containing 36 deformable particles using two treads. The stiffness of the soft-body is currently set to a relatively low value to illustrate the effect of the fluid particles colliding with the soft-body. This however will make the soft-bodies penetrate each other easily. For a higher stiffness value of the soft-bodies, the effect of the fluid particles on the soft-bodies becomes less evident.

6.2 Controls

While running the application, additional fluid particles can be added by pressing the "F" key and then using the mouse left click to position the fluid dam.

To add a soft-body, the key "S" needs to be pressed and the left click to position it. Once the soft-body is positioned, it does not start to update the physics immediately. To start the physics update the right mouse click needs to be pressed.

While being updated, the soft-body is intractable using the mouse left click. Click on a particle or close enough to it and holding the mouse button pressed, will allow to the user to drag the soft-body around.

Also to change the velocity damping and the viscosity of the fluid, the up and arrow keys needs to be first used to select an option. Once selected the option will be displayed in upper-case and the user can change the current value in real-time by using the mouse scroll.

7. REFERENCES

Gamasutra. 2007. *Physics in Games: A New Gameplay Frontier*. [ONLINE] Available at:http://www.gamasutra.com/view/feature/131207/physics_in_games_a_new_gameplay_.php. [Accessed 27 March 15].

Macklin, M, 2013. Unified Particle Physics for Real-Time Applications. *ACM*, [Online]. ACM TOG 33(4), 4-5 7-8. Available at: http://mmacklin.com/uppfrta_preprint.pdf [Accessed 21 April 2015].

Muller, M, 2007. Position Based Dynamics. [Online]. 3rd Workshop in Virtual Reality Interactions and Physical Simulation, 1-5. Available at: http://matthias-mueller-fischer.ch/publications/posBasedDyn.pdf[Accessed 22 April 2015].

University of Pennsylvania. 2015. *Physical Based Animation*. [ONLINE] Available at:https://www.youtube.com/watch?v=fH3VW9SaQ_c. [Accessed 15 April 15].

Stam, J, 2009. Nucleus: Towards a Unified Dynamics Solver for Computer Graphics., [Online]. IEEE International Conference on Computer-Aided Design and Computer Graphics, 1-3. Available at:http://www.autodeskresearch.com/pdf/nucleus.pdf [Accessed 10 April 2015].

NVidia GameWorks. 2013. *PhysX FleX*. [ONLINE] Available at: https://developer.nvidia.com/physx-flex. [Accessed 18 April 15].

PhysXInfo. 2013. *Introducing NVIDIA FLEX: unified GPU PhysX solver*. [ONLINE] Available at:http://physxinfo.com/news/11860/introducing-nvidia-flex-unified-gpu-physx-solver/. [Accessed 13 April 15].

Macklin, M, 2013. Position Based Fluids. . [Online]. ACM TOG 32(4), 1-5. Available at:http://mmacklin.com/pbf_sig_preprint.pdf [Accessed 22 April 2015].

M Macklin. 2013 - Slides. *Position Based Fluids*. [ONLINE] Available at: http://mmacklin.com/pbf_slides.pdf. [Accessed 14 April 15].

Akinci, N, 2012. Versatile Rigid-Fluid Coupling for Incompressible SPH. . [Online]. Siggraph 2012, 1-4. Available at: http://cg.informatik.uni-freiburg.de/publications/2012_SIGGRAPH_rigidFluidCoupling.pdf[Accessed 02 April 2015].

Monaghan, J. J., 1992. Smoothed Particle Hydrodynamics. . [Online]. Annual Review of Astronomy and Astrophysics, 1-2. Available at:http://www.annualreviews.org/doi/pdf/10.1146/annurev.aa.30.090192.002551 [Accessed 09 March 2015].

Muller, M., 2003. Particle-Based Fluid Simulation for Interactive Applications. . [Online]. Eurographics/SIGGRAPH Symposium on Computer Animation (2003), 1-7. Available at:http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.121.844&rep=rep1&type=pdf [Accessed 12 April 2015].

Monaghan, J.J., 2005. Smoothed particle hydrodynamics. . [Online]. 1-3. Available at:http://cg.informatik.uni-freiburg.de/intern/seminar/particleFluids_Monaghan%20-%20SPH%20-%202005.pdf [Accessed 21 March 2015].

Muller, M., 2005. Meshless Deformations Based on Shape Matching. . [Online]. 1-8. Available at:https://www.cs.drexel.edu/~david/Classes/Papers/MeshlessDeformations_SIG05.pdf [Accessed 22 April 2015].

University of Pennsylvania. 2014. *Physical Based Animation*. [ONLINE] Available at: https://www.youtube.com/watch?v=6Zsv2PyeQ5c. [Accessed 17 April 15].

Muller, M., 2014. Unified Particle Physics for Real-Time Applications. . [Online]. ACM TOG 33(4), 1-12. Available at: http://mmacklin.com/uppfrta_preprint.pdf [Accessed 23 April 2015].

Ericleong.me. 2015. *Circle-Circle Collision Tutorial*. [ONLINE] Available at:http://ericleong.me/research/circle-circle/. [Accessed 17 April 15].

The mind of Conkerjo. 2009. *Spatial hashing implementation for fast 2D collisions*. [ONLINE] Available at:https://conkerjo.wordpress.com/2009/06/13/spatial-hashing-implementation-for-fast-2d-collisions/. [Accessed 05 March 15].

Thomas Jakobsen. 1999. *Advanced Character Physics*. [ONLINE] Available at:http://www.gamasutra.com/view/feature/131313/advanced_character_physics.php?print=1. [Accessed 20 March 15].

Tristam MacDonald. 2009. *Spatial Hashing*. [ONLINE] Available at:http://www.gamedev.net/page/resources/_/technical/game-programming/spatial-hashing-r2697. [Accessed 20 March 15].

Christer Byström. 2014. *Quadtree vs Spatial Hashing - a Visualization*. [ONLINE] Available at:http://zufallsgenerator.github.io/2014/01/26/visually-comparing-algorithms/. [Accessed 05 April 15].

Michael. 2008. *Collision detection for particle systems*. [ONLINE] Available at: http://lab.polygonal.de/?p=239. [Accessed 06 April 15].

Solenthaler, B., 2007. A Unified Particle Model for Fluid-Solid Interactions. , *Computer Animation and Virtual Worlds*, [Online]. 1-2. Available at: https://graphics.ethz.ch/~sobarbar/papers/Sol07b/Sol07b.pdf[Accessed 31 March 2015].

Muller, M., 2004. Point based animation of elastic, plastic and melting objects.,[Online]. Proceedings of the 2004 ACM SIGGRAPH, 1-3. Available at: http://dl.acm.org/citation.cfm?id=1028542 [Accessed 04 April 2015].

Carlson, M. T., 2004. *RIGID, MELTING, AND FLOWING FLUID*. Doctor of Philosophy. Georgia: College of Computing Georgia Institute of Technology.

Muller, M., 2004. Interaction of Fluids with Deformable Solids. , [Online]. , 1-4. Available at:http://matthias-mueller-fischer.ch/publications/casa2004.pdf [Accessed 04 April 2015].

physxinfo. 2015. *PhysX FleX*. [ONLINE] Available at: http://physxinfo.com/wiki/PhysX_FleX. [Accessed 15 April 15].

Matthias Müller. 2014. *Physics in Games*. [ONLINE] Available at: http://matthias-mueller-fischer.ch/talks/GI2014.pdf. [Accessed 21 April 15].

Benra, F.-K., 2011. A Comparison of One-Way and Two-Way Coupling Methods for Numerical Analysis of Fluid-Structure Interactions. *Journal of Applied Mathematics*, [Online]. , 1. Available at:http://www.hindawi.com/journals/jam/2011/853560/ [Accessed 22 April 2015].

Yin, X., 2013. Particle-Based Simulation of Fluid-Solid Coupling. , [Online]. , 1-2. Available at:http://link.springer.com/chapter/10.1007%2F978-3-642-45037-2_37#page-1 [Accessed 21 April 2015].

code.google.com/p/opencloth/. 2012. *opencloth*. [ONLINE] Available at:https://code.google.com/p/opencloth/. [Accessed 20 April 15].

physxinfo. 2013. *PhysX Research: Position-based Methods for the Simulation of Solid Objects*. [ONLINE] Available at: http://physxinfo.com/news/10572/physx-research-position-based-methods-for-the-simulation-of-solid-objects/. [Accessed 22 April 15].

Gascuel, J-D, 1994. Displacement constraints for interactive modelling and animation of articulated structures. , [Online]. , 1-2. Available at: http://link.springer.com/article/10.1007%2FBF01901286#page-1[Accessed 13 March 2015].

Faure, F., 1999. Interactive Solid Animation Using Linearized Displacement Constraints. , [Online]. , 1-2. Available at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.4.602&rep=rep1&type=pdf[Accessed 15 March 2015].

Desbrun, M., 1999. Interactive animation of structured deformable objects. , [Online]. , 1-2. Available at:http://www.multires.caltech.edu/pubs/GI99.pdf [Accessed 05 March 2015].

Terzopoulos, D., 1987. Elastically Deformable Models. , [Online]. Computer Graphics, Volume 21, Number 4, 1. Available at: http://design.osu.edu/carlson/history/PDFs/ani-papers/terzopoulos-deformable.pdf [Accessed 12 March 2015].

Irving, G., 2004. Invertible Finite Elements For Robust Simulation of Large Deformation. , [Online]. ACM SIGGRAPH Symposium on Computer Animation (2004), 1-2. Available at:https://www.math.ucla.edu/~jteran/papers/ITF04.pdf [Accessed 19 March 2015].

Baraff and Witkin, D. A., 1998. Large Steps in Cloth Simulation. , [Online]. COMPUTER GRAPHICS Proceedings, Annual Conference Series, 1998, 1-2. Available at:http://www.cs.cmu.edu/~baraff/papers/sig98.pdf [Accessed 02 March 2015].

Barbara Cutler. 2010. *Advanced Computer Graphics*. [ONLINE] Available at:http://www.cs.rpi.edu/~cutler/classes/advancedgraphics/S10/lectures/10_rigid_body_deformation_fracture.pdf. [Accessed 17 March 15].

Debunne, G., 2001. Dynamic Real-Time Deformations using Space & Time Adaptive Sampling. , [Online]. , 1-2. Available at: http://www-ljk.imag.fr/Publications/Basilic/com.lmc.publi.PUBLI_Inproceedings@1172c0fd434_5d0eaf/DDCB01.pdf[Accessed 23 March 2015].

Bridson, R., 2008. *Fluid Simulation for Computer Graphics*. 1st Ed. .: CRC Press Taylor & Francis Group.

Pozorski, J., 2002. SPH computation of incompressible viscous flows. *Journal of theoretical and applied mechanics*, [Online]. , 1-2. Available at: http://www.ptmts.org.pl/2002-4-pozorski-w.pdf [Accessed 03 April 2015].

Michael J. Gourlay. 2012. *Fluid Simulation for Video Games*. [ONLINE] Available at:https://software.intel.com/en-us/articles/fluid-simulation-for-video-games-part-1. [Accessed 20 April 15].

Becker, M., 2007. Weakly compressible SPH for free surface flows. , [Online]. ACM SIGGRAPH Symposium on Computer Animation (2007), pp. 1–8, 7-8. Available at: http://cg.informatik.uni-freiburg.de/publications/2007_SCA_SPH.pdf [Accessed 07 April 2015].

Brackbill, J.U., 1988. Fluid-Implicit-Particle.[Online].1. Available at:http://www.researchgate.net/publication/222452290_Flip_A_low-dissipation_particle-in-cell_method_for_fluid_flow [Accessed 20 April 2015].