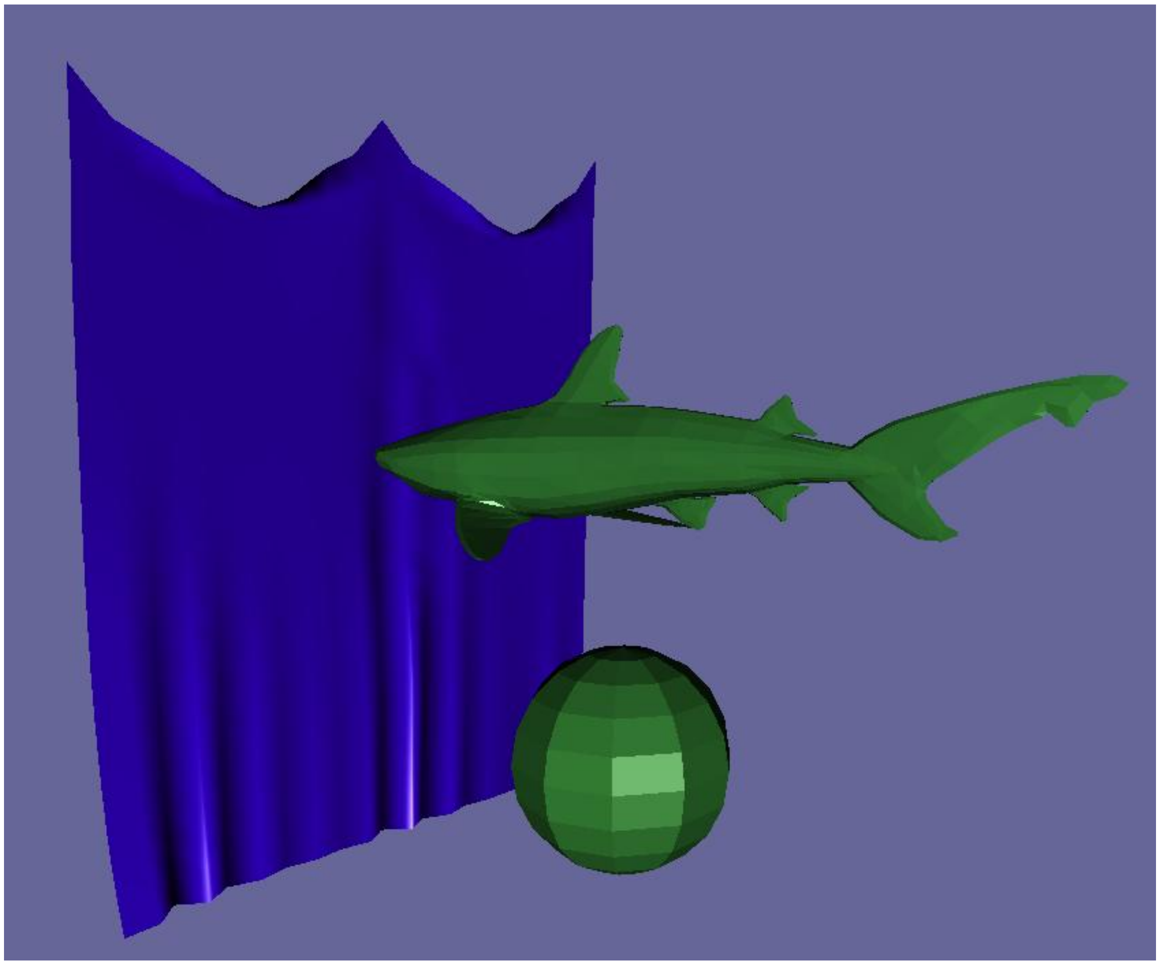
BOURNEMOUTH UNIVERSITY

INTERACTIVE TOOL FOR CLOTH SIMULATION

by

VOLHA KOLCHYNA

MSc Computer Animation and Visual Effects



NCCA

August, 2011

Abstract

Animation of cloth has experienced a huge development in the recent years. However, real-time realistic simulation still remains a challenge. This project investigates different methods and techniques used in cloth simulation, including methods for the numerical integration, collision detection, collision handling. Big research has been done on wrinkles enhancement approaches.

The tool developed for this project allows to create the visual scene by loading static and animated objects from the external applications in the OBJ file format. Any static object can be dynamically converted into the cloth and unique parameters for each cloth object may be specified. The cloth objects handle collisions with static objects, animated objects, as well as self collisions. Cloth simulation created by the tool can be recorded as a sequence of OBJ files and later used in the external application, such as Maya.

Table of Contents

List of Figures and Illustrations

Chapter One: INTRODUCTION

The production of visually astonishing movies, such as Avatar, Lord of the Rings, Kung-Fu Panda would not be possible without flexible and efficient technology. Big part of this technology is physical simulation tools for creating smoke, hair, rigid bodies, cloth movements. A physical simulation is usually a numerical integration performed on computer: a differential equation, which describes the motion of the object, is integrated in time to generate the action of the simulated object.

This thesis concentrates on cloth simulation for computer animation. It is extremely important for the cloth simulation to be highly realistic and physically plausible, otherwise, the virtual nature of the animated character will be revealed. Extensive research has been done in this area for the last few decades, which allowed creation of memorable characters such as Shrek, *Shrek*; Boo from *Monsters, Inc*; Elf from *Lord of the Rings*. In this projects different approaches to achieve effective cloth simulation results has been investigated and analysed. The most appropriate methods, in terms of Time-To-Implement versus Performance, have been implemented.

1.1 OBJECTIVES

The objectives of the project included:

- **Conversion of the uploaded mesh into cloth:** the user should have a possibility to dynamically load the mesh in OBJ format, created by the external tool and convert it into the cloth

- **Interaction of cloth with static/dynamic objects:** cloth should be able to handle collisions with the static objects on the scene, as well as with animated objects
- **Self-collisions handling:** cloth should handle collisions with itself
- **Multiple cloth possibility:** tool should allow to create multiple clothes in one scene
- **Cloth constraints:** user defined points on the cloth which get fixed
- **Possibility to load animation into the tool:** allow to load an animated object, created in the external tool
- **Export of the cloth simulation from the tool:** the possibility to record the simulation created by the tool and open it in the external tool

1.2 RESOURCES USED FOR IMPLEMENTATION

- **NCCA Graphics Library:** the Open-source library developed by Jon Macey, containing classes to work with vectors, matrices, textures and others
- **BOOST Library**: the Open-source library used to work with loops effectively
- **QtCreator:** the IDE for implementation
- **Doxygen**: tool to generate the documentation
- **Autodesk Maya**: modelling software to create meshes and animations for loading into the tool

Chapter Two: PREVIOUS WORK

The results of the research in clothe simulation and animation can be tracked to the last 25 years of research in CGI. The pioneer works of Terzopoulos et al. (1987), Carignan (1992), Provot (1995) had as a focus the simulation of flexible objects. A detailed recount of the early history of cloth simulation can be found in the compilation of NG and Grimsdale (1996), which was later followed by House and Breen's extensive list (2000) and Volino et al. In (2005).

A detailed revision of the work produced in this field is far beyond the scope of this project, due to its broad range. The following is a brief chronological inventory of the key theoretical approaches to the problem of cloth simulation here developed:

Terzopoulos et al., (1987), Elastically deformable models, Computer Graphics, SIGGRAPH 87. Uses a rectangular mesh and a semi-implicit integration to determine the new positions. The practical use of the theory of elasticity is used for the animation.

Breen et al., (1994), Predicting the drape of woven cloth using interacting particles, SIGGRAPH '94. Brought into perspective a simulation of real life properties in a particle-based approach and incorporated the measuring system of Kawabata.

Provot, (1995): Deformation constraints in a mass-spring model to describe rigid cloth behavior, Graphics Interface '95. Applied the explicit Euler integration (simple ordinary differential equation) to update a particle system that is a massless-spring system.

Provot, (1997): Collision and self-collision handling in cloth model dedicated to design garments, Proceedings of the Eurographics Workshop on Computer Animation and Simulation (1997). In his later work Provot explored and incorporated collisions and self-collisions of the cloth.

Baraff and Witkin, (1998): Large steps in cloth simulation, SIGGRAPH'98. In search of calculation efficiency and improvement (while not losing relevant information and maintaining the clothe stability), introduced the implicit integration with large time steps . As a result they created a very fast algorithm.

Desbrun et al., (1999): Interactive animation of structured deformable objects, Proceedings of Graphics Interface (GI 1999). Extended Baraff and Witkin’s research to an even faster algorithm.

Volino and Magnetat-Thalmann, (2000): Implementing fast cloth simulation with collision response, Proceedings of Computer Graphics International (CGI 2000). The use of an implicit integration method propelled the cloth self collisions (2000).

Volino and Magnetat-Thalmann, (2000): Comparing efficiency of integration methods for cloth simulation, Computer Graphics International Proceedings. In the later work 2001, they compared different integration methods.

Bridson et al., (2002): Robust treatment of collisions, contact and friction for cloth animation, SIGGRAPH '02. Explored the self collisions of the cloth. The solution is based on an AABB tree optimizing the neighbour links and therefore gaining at a micro-level.

Bridson et al., (2005): Simulation of clothing with folds and wrinkles, SIGGRAPH '05. In this later work, the interest concentrated in how to maintain the folds and wrinkles of the cloth while in contact with rigid objects.

Villard and Borouchaki, (2005): Adaptive meshing for cloth animation. Adaptive-mess method allows to increase definition as required, because it allows the use of low definition meshes in simulation, by making use of a quad tree to subdivide the mesh.

Selleet al., (2009): Robust high-resolution cloth using parallelism, history-based collisions, and accurate friction, IEEE. Transactions on Visualization and Computer Graphics. By keeping collision history it is possible to handle high resolution cloth objects (up to 2 million particles).

Chapter Three: TECHNICAL BACGROUND

3.1 Mechanical Simulation of the Cloth

Computer cloth simulation intends to reproduce virtual cloth with given parameters, such as thickness, stiffness, weight, damping. Manipulation with these parameters allows creation of variety of different materials and simulation of their behaviour.

The cloth also has to react to the environment. These interactions are collisions with the environment objects, which could be static and dynamic, as well as self-collisions between various garment parts. The parameter of such interaction is friction, which accounts for reaction of the object to the colliding surface. The gravity should also be taken in account while simulation. This force is proportional to the mass of the object and creates an acceleration that pulls objects towards the ground. Advanced models might consider aerodynamic forces and external forces, which may come from the surrounding objects or the user.

To represent the behaviour of the cloth, additional equations are required to reproduce the fundamental laws of mechanics and create physically-plausible simulations. Among them, *Newton's Second Law, Hooke's Law, energy conservation laws.* This laws may be combined in different variation forms to produce the simulation, which suits particular problem.

3.1.1 Mechanical Models

After combining the equations for cloth materials with mechanical laws, complex systems of mathematical equations will be created, usually partial differential equations or other types of differential systems (Magnenat-Thalmann, 2004). The numerical solution of a system of differential equations requires discretization. Depending on where in the simulation process the discretization takes place, two major groups of schemes for producing simulation can be described:

- *Continuum mechanics,* which calculates the material properties through quantities varying continuously in time.
- *Particle systems,* which represent the cloth as a set of points (masses). These masses interact between themselves and with the environment via ‘forces’, which approximately models the behaviour of a garment.

3.1.1.1 Continuum Mechanics

As mentioned by Thalmann, a continuum mechanics describes the state of the object using continuous expressions; usually the surface deformation energy related the local surface deformation (elongation, shearing, curvature).

- *Advantages:* accurate models, which are physically correct due to using mechanical laws and models for the material properties
- *Disadvantages:* heavy computational requirements; rendering drawbacks

3.1.1.2 Particle system models

The most common method to represent the cloth presented as a polygonal mesh. The geometrical discretization corresponds to the descretization of the geometrical model.

Because of relative simplicity of the model and realistic results it produces, this approach has been chosen as a method for cloth simulation in this thesis. A good way to design a cloth model based on particle system is a Mass-Spring system.

3.1.2 Mass-Spring System

Accurately described by (Provot, 1995) this is model became very popular for cloth simulation. Each vertex of the cloth mesh gets represented by a particle (mass). The particles interact with the neighbouring particles in different ways, depending on the properties of the cloth model chosen. Particles are linked to each other by a ‘spring’ representing the elastic behaviour of the material. Different types of springs are used to simulate various materials properties:

- *Metric elasticity*: elongation springs along the lattice edges
- *Sharing elasticity*: lattice angle springs or diagonal elongation springs
- *Curvature elasticity*: flexion springs between opposing edges or elongation springs between opposite vertices.

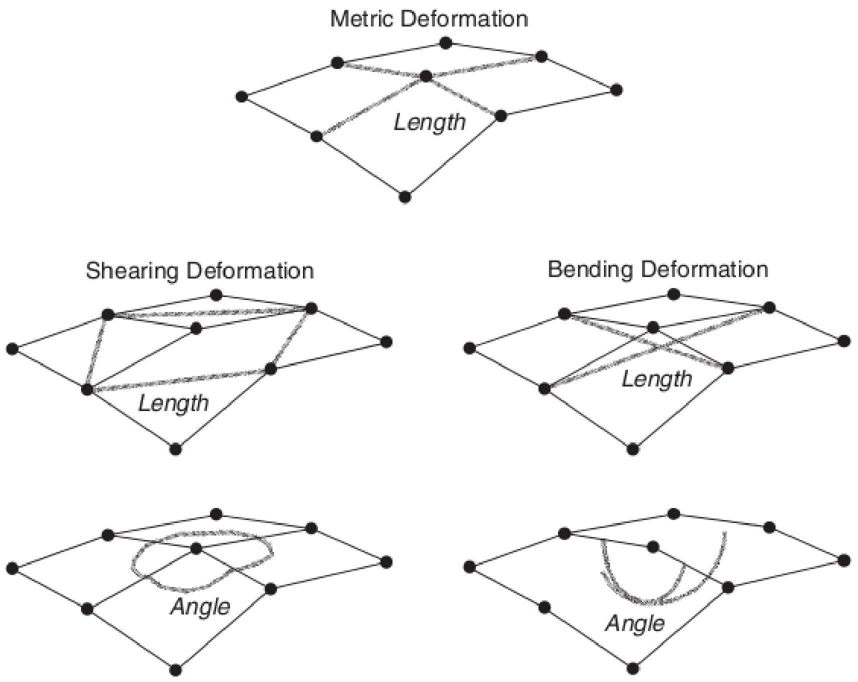


Figure 3.1 Different types of Deformations, Thalmann et al., 2004

3.1.2.1 Spring models

Depending on the type of the mesh different spring models might be used to simulate the properties described above.

For quadrilateral mesh it is appropriate to use Structural, Shear and Flexion springs.

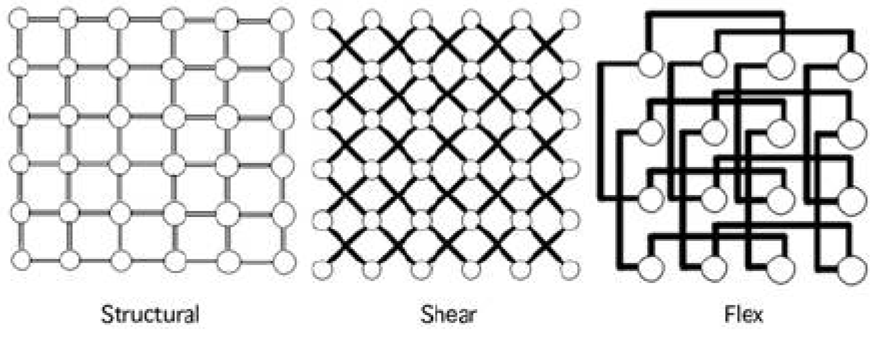


Figure 3.2 Structural, Shear and Flex springs (Liberatore n.d)

In the case of triangulated mesh Edge and Bending springs might be used to handle extension and compression.

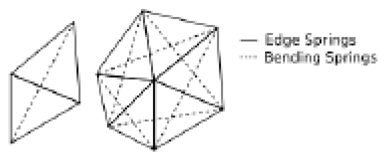


Figure 3.3 Edge and Bending Springs Model (Selle et.al, 2009)

3.1.2.2 Forces

Particles interact with a set of forces. The forces can be applied directly to the particles or to the springs and then distributed to the masses. According to the Hook Law in order to maintain the spring *stiffness* the following force should be applied:

$$F_{stiffness} = -k_s x \tag{3.1.1}$$

where k_s is the stiffness coefficient, x is the exceeding spring length.

To prevent the particles from oscillating the *damping* force should be applied:

$$F_{damping} = -k_d v \tag{3.1.2}$$

where k_d is a damping coefficient and v is velocity difference between the particles building the spring.

A combined equation for stiffness-damping spring may look in the following way:

$$F = -\, k_s(|L| - L_0)\left(\frac{L}{|L|}\right) - k_d(v_1 - v_2) \tag{3.1.3}$$

where $|L|$ is the length between two particles, L is the vector from one particle to another, L_0 is the rest length of the spring, $v_1 - v_2$ difference between particles velocities respectively.

The forces acting on each particle depend on the state of the system at each moment, which is represented by the positions and speeds of all the particles. The forces represent all the mechanical impacts on the system, such as internal forces (elasticity, viscosity, gravity, aerodynamics) and external forces (collisions with other objects). Formulating the equation of the motion for a particle leads to a large second-order ODE system that has to be integrated using one of the standard integration methods.

3.2 Integration Methods

The complex model equations cannot be solved analytically. The simulation has to be calculated using numerical process. In a mass-spring system the problem is minimised to solving secondary-order differential equation, where variables are the mass positions along the evolving time.

3.2.1 Explicit numerical Integration Methods

Explicit integration methods are the dimpliest methods available to solve ODE. They predict the future state of the system based on derivatives.

3.2.1.1 Euler Integration

The basic explicit integration method. It is based on the Newton's second law of motion and uses the information at the beginning of the time step to find the velocity and the position of the particle at the end of the step.

$$\bar{a} = \frac{F}{m} \tag{3.2.1}$$

$$\bar{v} = \bar{v_0} + \Delta t * \bar{a} \tag{3.2.2}$$

$$\bar{x} = \bar{x}_0 + \Delta t * \bar{v} \tag{3.2.3}$$

Explicit Euler method has stability problems. The method is not symmetrical, so if large time steps are used, the estimated path will deviate from the actual path greatly.

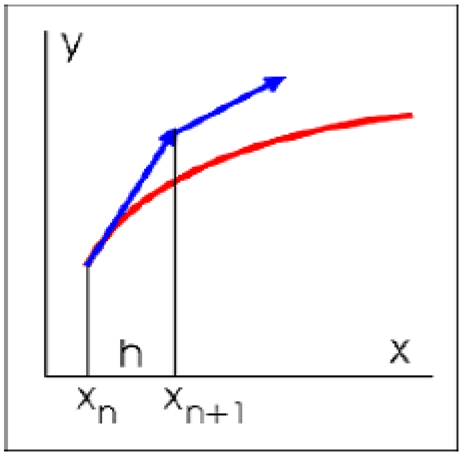


Figure 3.4 Explicit Euler Integration (Kieran et al., 2005)

As shown in the Figure 3.3 the tangent from the beginning is used to approximate the behaviour over the entire step. This fact demonstrates that Euler method is not accurate.

3.2.1.2 Verlet Integration Method

Verlet Integration is a fast method for numerically integrating the equations of motion. It has the benefit of being quite stable, especially in enforced boundary conditions. It is also very fast to compute and under right condition it is 4th order accurate and requires two steps to start working. Disadvantages of Verlet are that it handles changing time steps badly.

$$v(t+\Delta t) = v(t) + \frac{1}{2}(a(t) + a(t+\Delta t))\Delta t \qquad (3.2.4)$$

3.2.1.3 Runge-Kutta 4th Order

The family of Runge-Kutta methods is based on taking intervals during the time step to approximate a more accurate answer. This allows to improve the accuracy in comparison with Euler and Mid-Point integration methods as well as take larger time steps. As described by Conte and de Boor(), the Euler method is used to compute the intervals during the time step.

Initial position and velocity of the particle:

$$k_1 = \Delta t f(x_n, y_n) \tag{3.2.4}$$

Position and velocity of the particle at time step $\frac{h}{2}$

$$k_2 = \Delta t f(x_n + \frac{1}{2}\Delta t, y_n + \frac{1}{2}k_1) \tag{3.2.5}$$

$$k_3 = \Delta t f(x_n + \frac{1}{2}\Delta t, y_n + \frac{1}{2}k_2) \tag{3.2.6}$$

Position and velocity of the particle at time step h

$$k_4 = \Delta t f(x_n + \Delta t, y_n + k_1) \tag{3.2.7}$$

Final position and velocity are calculated using Taylor Series

$$y_{n+1} = y_n + \frac{1}{6}k_1 + \frac{1}{3}k_2 + \frac{1}{3}k_3 + \frac{1}{6}k_4 + O(h^5) \qquad (3.2.8)$$

3.2.2 Implicit Numerical Integration Methods

The implicit Euler Integration method, also called Backward Euler, is proven to be stable and accurate. The implicit method finds a new position whose derivative can update the current value to the new ZZZZ

$$y_{n+1} = y_n + hf'(x_{n+1}, y_{n+1}) \qquad (3.2.8)$$

The major difficulty in using implicit integration methods is that they involve resolution of a large linear equation system for each iteration. This being a 'stopping' factor for implementing implicit methods.

Various approaches have been proposed to resolve this issue. Kang and Cho (2000), for example, suggested to linearize the problem. Desbrun et al. (1999) used inverse matrices for these purpose. A robust solution was proposed by Baraff and Witkin (1998) by using the Conjugate Gradient method. Volino and Magnenat_thalmann (2000b) improved this approach by evaluating the matrix of the Conjugate Gradient algorithm 'on the fly' for each iteration.

3.3 Choosing a Suitable Integration Method

Obviously, implicit method have advantage over explicit methods in most application for computer graphics. While explicit methods need to have time steps adapted to prevent numerical instability, implicit method can afford to use large time steps. Unfortunately, implicit methods are harder to implement and they are much more expensive in the computational time required.

Implicit methods are not the perfect solution for any kind of a problem. Providing a great stability they do not always provide the same level of accuracy due to the large time steps. Having this in mind the choice of the integration method should depend on the purpose of the simulation. In case where accuracy is required, the explicit methods would perform better. In order to gain stability, one should implement the implicit method.

In (Volino and Magnenat-Thalmann 2001) is performed a comparison of explicit and implicit models in terms of computational performance and simulation accuracy.

The purpose of this thesis is to implement an accurate cloth simulation, that is why Runge Kuta method of 4th order method became a natural choice. Explicit Euler and Verlet methods were also implemented to compare their performances.

In typical cloth simulations, the fourth Runge_Kutta method (used by Eberhadt, 1996 for example) has proven to be far superior to the Euler method and second-order Midpoint method. Even though this method computationally is more expensive, it allows to use much bigger time steps. The accuracy and stability have also proved to be much higher.

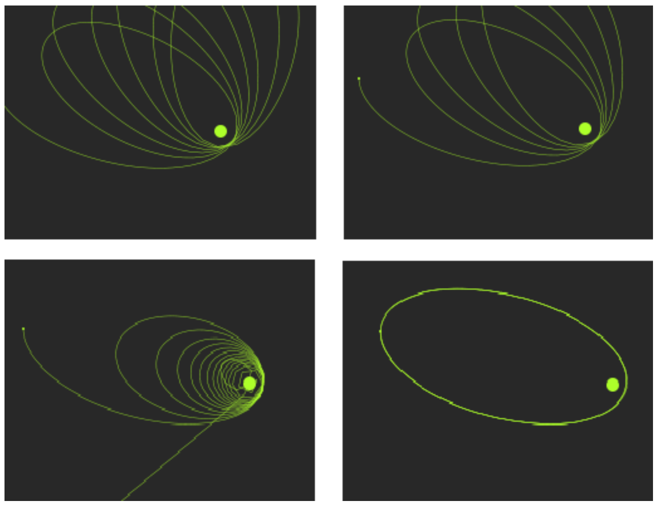


Figure 3.5 Different integration methods: Left-top corner: Explicit Euler; Right-top corner: Verlet; Left-down corner: Runge-Kutta and M.W.Kutta; Right-down corner:Runge-Kutta of 4th order.

The Figure 3.5 is the graphical representation of four integration methods (Explicit Euler, Verlet, Runge-Kutta and M.W.Kutta, Runge Kutta of 4th order), demonstrating the accuracy of these methods (Boesch, 2010). The goal was to draw an ellipse. It is clearly evident that Runge-Kutta of 4th order method provides the most accurate results.

3.4 Collisions

Accurate resolving of collisions plays a major role in realistic Cloth Simulation. From the point of view of mechanical simulation, dealing with collisions involves two types of problems:

- *Collision detection:* allows to find geometrical contacts between the objects
- *Collision response:* integrating the resulting reaction and friction effects in the mechanical simulation

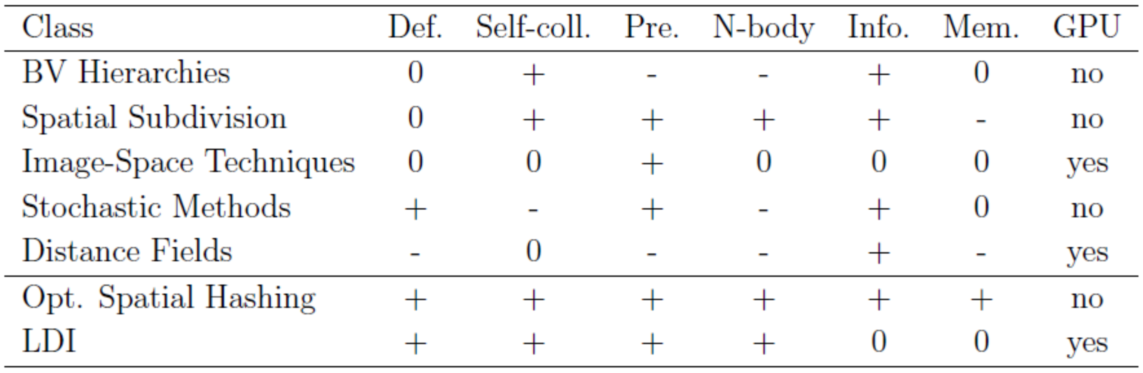
The process of finding collisions is usually broken into two steps . During the first step objects' locations are analysed and objects are broken into the groups of potential collision. During the second step collision tests are performed within the same collision group. This two steps allow to speed up the process of collision detection greatly, however the father optimization should take place in order to achieve real-time or close to real-time performance.

Methods for optimization of ccollision detection can be classified into five groups: Bounding-volume hierarchies, spatial subdivision, image-space techniques, stochastic methods and distance fields. The most popular of them are bounding-volume hierarchies and spatial subdivision.

The idea of BVHs is to recursively divide the object primitives and build a tree structure, where internal nodes will contain the links to their child nodes. The leafs of the tree should contain the references to the associated object primitives. Apart from that, each node in the tree will contain a bounding volume (BV) that encloses the associated primitives or child nodes shapes.

In order to detect the collision between two objects using BVH, the structure should be traversed top-down and pairs of tree nodes are recursively tested for overlap. If the overlapping nodes are leaves of the BVH, then the enclosed primitives are tested for exact intersection. If only one node is a leaf while the other one is an internal node, the leaf node is tested against each of the children of the internal node.

In comparison with the BVH, which works in the object space, Spatial Subdivision technique operates in a world space. Spatial subdivision allows accelerating the collision detection process by subdividing the space into cells. Only objects which share the same cell will be tested for a collision.



Class	Def.	Self-coll.	Pre.	N-body	Info.	Mem.	GPU
BV Hierarchies	0	+	-	-	+	0	no
Spatial Subdivision	0	+	+	+	+	-	no
Image-Space Techniques	0	0	+	0	0	0	yes
Stochastic Methods	+	-	+	-	+	0	no
Distance Fields	-	0	-	-	+	-	yes
Opt. Spatial Hashing	+	+	+	+	+	+	no
LDI	+	+	+	+	0	0	yes

Figure 3.6 Evaluation of different collision detection methods, Heidelberger (2007)

The Figure 4.1 implemented by Heidelberger (2007) illustrates the ratings of different collision detection methods based on seven criteria: Applicability to deformable objects (Def.), self-collision support (Self-coll.), minimal pre-processing (Pre.), n-body support (N-body), collision information quality (Info.), memory usage (Mem.) and graphics hardware acceleration (GPU). The rating is either good (+), neutral (0) or bad (-).

As it might be seen from the table the Optimized Spatial Hashing (Portioning) method allows to produce the best results. Based on this analysis the decision was made to implement this method in the thesis.

3.5 Optimized Spatial Partioning

The method is based on spatial subdivision, but addresses the high memory consumption with the use of a hashing scheme. Hash function is used to map cells to a finite number of hash table entries. This leads to a highly efficient collision detection solution that is very well suited for the interactive cloth simulation.

Each cell of this grid maintains the list of primitives that are fully or partially contained within the cell. This type of mapping will lead to collisions which will need to be resolved during the next step (Figure 4.2).

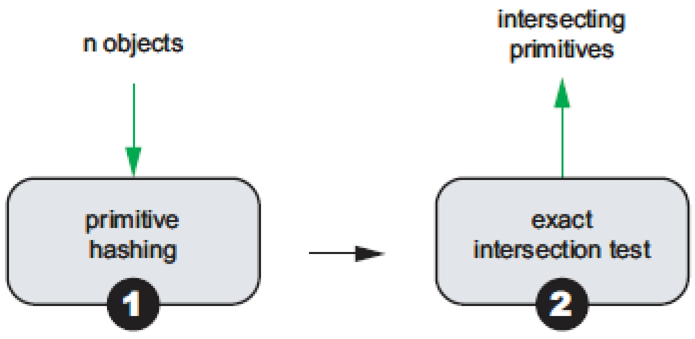


Figure 3.7 Two stages of the Spartial Partioning Algorithm, Heidelberger (2007)

3.5.1 Point Hashing

In order to define collisions between the objects of the scene each vertex of the mesh will be tested for the intersection with each triangle from a potential collision group. For this purpose each point of the object should be discretized into spatial grid.

If coordinates of a point p are(x, y, z), the indices of the corresponding cell will be (i, j, k), where s_{grid} is the grid cell size.

$$(i, j, k) = \left(\left\lfloor \frac{x}{s_{grid}} \right\rfloor, \left\lfloor \frac{y}{s_{grid}} \right\rfloor, \left\lfloor \frac{z}{s_{grid}} \right\rfloor\right) \tag{4.1.1}$$

The index in the hash table can be computed using formula:

$$h = \mathcal{H}(i, j, k) \tag{4.1.2}$$

Where $\mathcal{H}$ is an appropriate hash function.

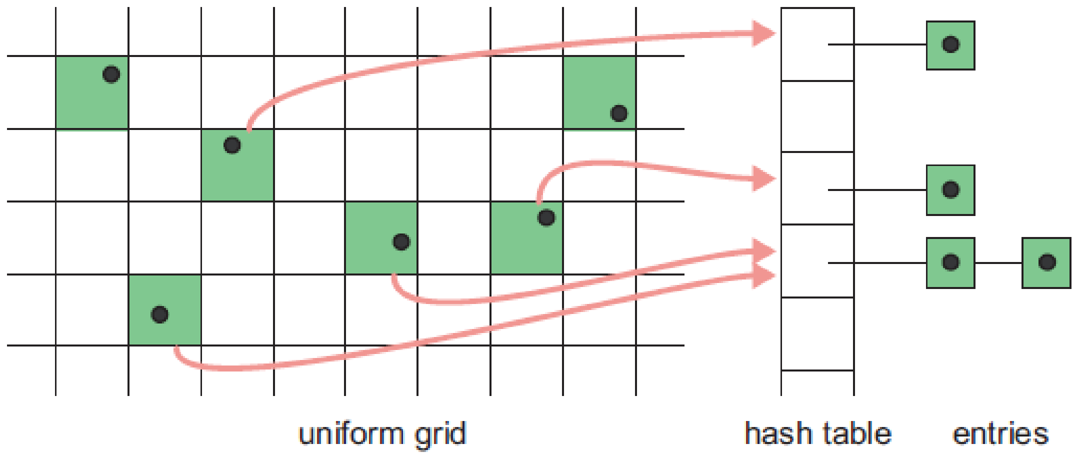


Figure 3.8 Points hashed into a hash table, Heidelberger (2007)

Results of spatial subdivision implemented in the tool are illustrated in the Figure 4.4.

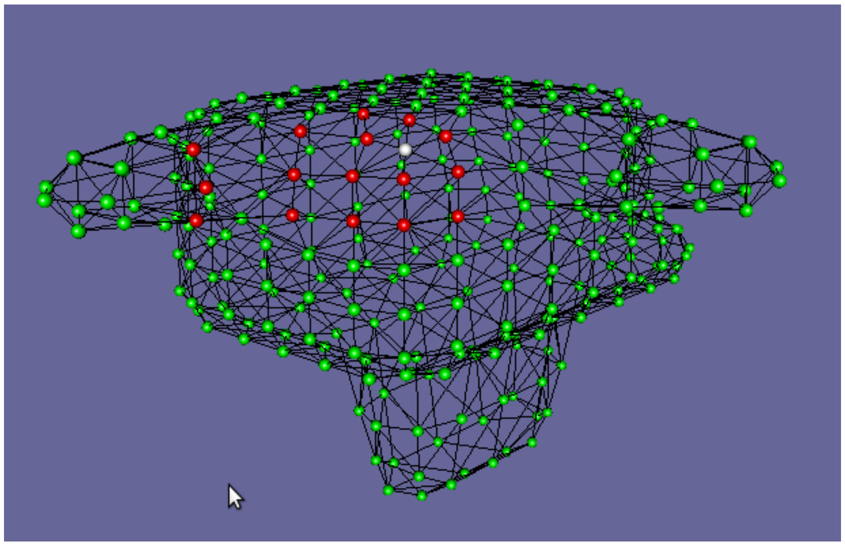


Figure 3.9 Spatial Partioning. In red are the neighbours of the particle coloured in white

3.5.2 Point-triangle Intersection

During the first stage of the algorithm all object primitives were discretized into cells and hashed into hash table. During the second step the collision test should be performed between all primitives within hash table. Because the mesh is composed of the series of triangles, collision test will be executed between a particle and a triangle (Moller and Trumbore).

A point $T(u, v)$, on a triangle is given by

$$T(u, v) = (1 - u - v)V_0 + uV_1 + uV_2 \qquad (4.1.1)$$

Where (u, v) are the barycentric coordinates, which must fulfil $u \geq 0, v \geq 0$ and $u + v \leq 1$. Computing the intersection between the ray, $R(t)$, and the triangle, $T(u, v)$ is equivalent to $R(t) = T(u, v)$

$$O + tD = (1 - u - v)V_0 + uV_1 + vV_2 \qquad (4.1.2)$$

Rearranging the terms gives:

$$[-D, V_1 - V_0, V_2 - V_0] \begin{bmatrix} t \\ u \\ v \end{bmatrix} = O - V_0 \qquad (4.1.3)$$

The above can be thought as translating the triangle to the origin, and transforming it to a unit triangle in $y\&z$ with the ray direction aligned with x, as illustrated in Figure 4.1. ($M = [-D, V_1 - V_0, V_2 - V_0]$)

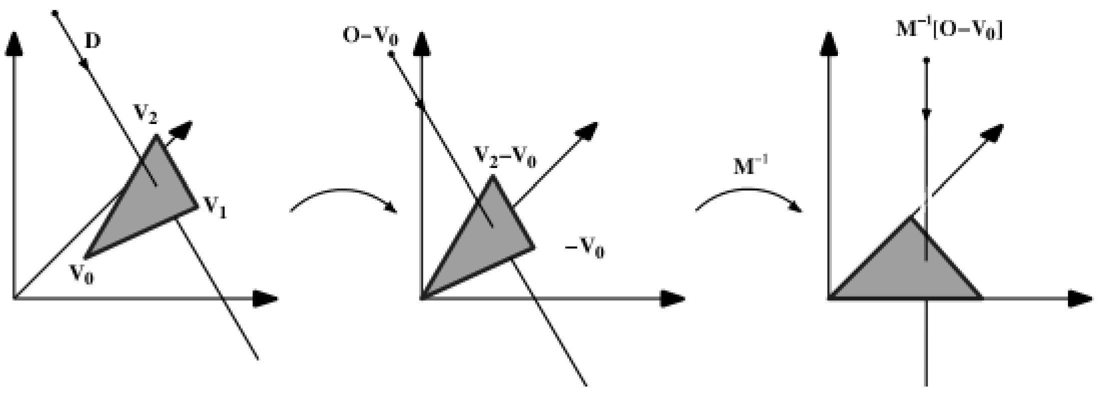


Figure 3.10 Point Triangle Intersection, Moller 1997

3.6 Collision Response

Correct collision response is essential for a realistic cloth simulation. Volino and Magnenat-Thalmann (2000a) three type of responces, which can be executed:

- *The position correction*: alters the position of the mass so the collision distance is maintained

- *The speed correction*: alters the speed of the colliding vertices so the collision distance is obtained at the next frame
- *The acceleration correction*: alters the acceleration of the colliding vertices so the collision distance is obtained two frames thereafter.

In this thesis first and the second approaches will be used for calculating the collision response. As demonstrated by Volino and Magnenat_Thalmann the combination of this three methods may provide the most robust solution for the collision response

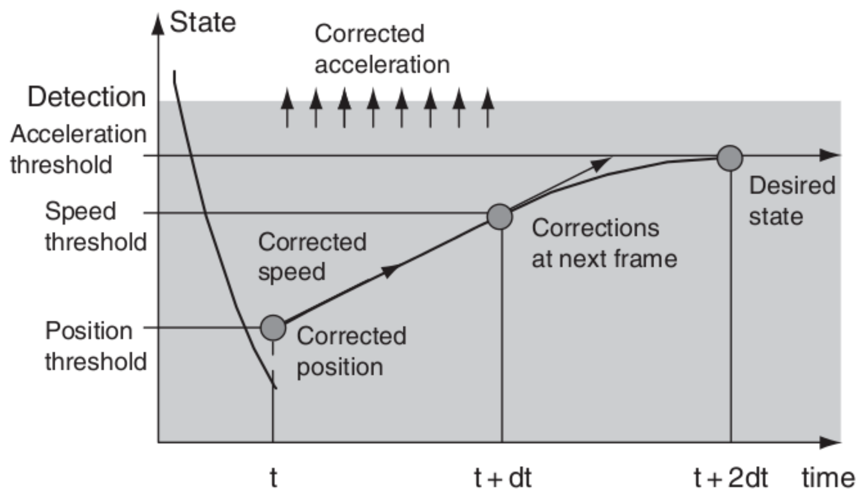


Figure 3.11 Combined Corrections of Position, Speed, acceleration, Thalmann, 2004

Depending on which object the cloth is colliding with, different collision response methods should be executed. Extensive work in this area has been done by Bridson (2002) and Bridson (2005). The collisions response method implemented in this thesis will be based on the approach suggested by Bridson.

3.6.1 Collision With Static Objects

In the case of the collision with the static objects the particle of the cloth will be the only object to change its position and velocity. The new position of the particle will be obtained using the following equation:

$$\bar{P}x = P\overline{C_x} - (\bar{n} * thickness) \tag{4.2.1}$$

Where $P\overline{C_x}$ is the collision point on triangle, $\bar{n}$ is a collision normal and $thickness$ is a thickness of the cloth, which defines the distance at which the collision will be detected.

In addition to modifying particle position it is necessary to adjust the particle velocity. Since the collision happened between the cloth and the object, the friction between them shall also be taken in account.

The laws of friction describe the forces that are applied to each of the objects when they are in contact. The friction forces could be derived from the Coulombian law. If the collision is partially "inelastic" there is some of dissipation of the energy on collision. In case of completely inelastic collision the energy will be completely dissipated. Both forces are integrated to generate a correct response:

$$\begin{cases} if\ \|\overrightarrow{v_T}\| \geq k_f\|\overrightarrow{v_N}\|, \overrightarrow{v_T} = \overrightarrow{v_T} - k_f\|\overrightarrow{v_N}\|\dfrac{\overrightarrow{v_T}}{\|\overrightarrow{v_T}\|} - k_d\overrightarrow{v_N} \\ if\ \|\overrightarrow{v_T}\| \geq k_f\|\overrightarrow{v_N}\|, \overrightarrow{v_T} \end{cases} \tag{4.2.2}$$

where $\overrightarrow{v} = \overrightarrow{v_T} + \overrightarrow{v_N}$ is the resultant velocity of the particle before the collision, and $\overrightarrow{v_T}$ and $\overrightarrow{v_N}$ are the tangential and the normal velocity components of the particle, respectively. k_f is the friction coefficient and k_d is the dissipation coefficient.

3.6.2 Self-Collisions

In case of self collisions for each colliding particles pair the repulsion force between two particles can be added. The impulse can be calculated based on the relative velocity of two particles according to the following equation:

$$I = \frac{mv_n}{2} \tag{4.2.3}$$

where I is the magnitude of the impulse, m is the mass of the particle and v_n is a relative velocity of the particle in a normal direction. The repulsion forces can be obtained using equation:

$$\tilde{I} = \frac{2I}{1 + w_1^2 + w_2^2 + w_{32}^2}$$

$$\overrightarrow{v_i^{new}} = \overrightarrow{v_i} + w_i\left(\frac{\tilde{I}}{m}\right)\hat{n},\ \ i = 1,2,3 \tag{4.2.4}$$

$$\overrightarrow{v_4^{new}} = \overrightarrow{v_4} + \left(\frac{\tilde{I}}{m}\right)\hat{n}$$

where $\tilde{I}$ is the weighted impulse, $\overrightarrow{v_i^{new}}$ indicates the new velocities of three particles of the triangle and $\overrightarrow{v_4^{new}}$ is a new velocity of a particle.

3.6.3 Collisions With Moving Objects

The collision response in this case will be the same as in case of self collision with the exception that the velocities of three particles of the triangle will not get updated. Only one particle will change its velocity in order to avoid the collision.

$$\overrightarrow{v_4^{new}} = \overrightarrow{v_4} + \left(\frac{\tilde{I}}{m}\right)\hat{n}$$

Chapter Four: IMPLEMENTATION

4.1 Architecture Design

The step of architectural design of any software is very important. The smart and flexible solution will allow extending the program in the future or easily modifying its behaviour. This was the main criteria while developing the design for Cloth Simulation tool. The class diagram can be found in Appendix A.

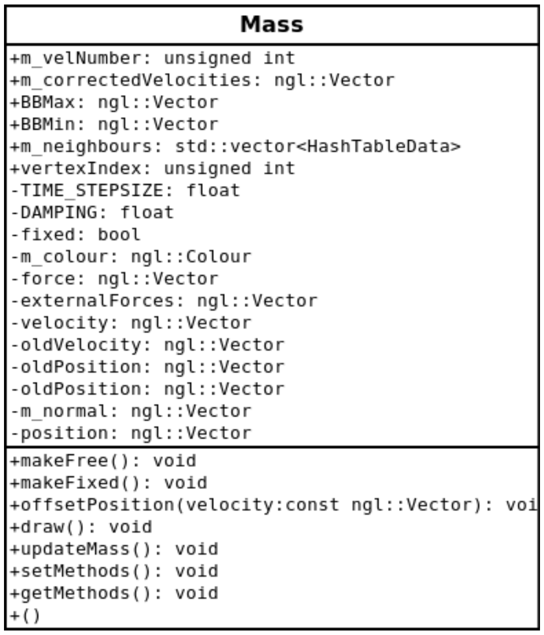
4.1.1 Main Classes

The main classes which have been developed include:

4.1.1.1 Mass:

In order to represent the particle of the cloth Mass class has been designed. It is named Mass, but not Particle in order to highlight its physical property of having weight. The class describes all the properties of the particle, such as mass, velocity, position, force acting on a particle. It also contains list of neighbouring particles, which is forms while spatial portioning and use in collision detection.

In order to be able to render the cloth as mass-system draw method is included, which allows to render the particle as a sphere.



```
Mass
+m_velNumber: unsigned int
+m_correctedVelocities: ngl::Vector
+BBMax: ngl::Vector
+BBMin: ngl::Vector
+m_neighbours: std::vector<HashTableData>
+vertexIndex: unsigned int
-TIME_STEPSIZE: float
-DAMPING: float
-fixed: bool
-m_colour: ngl::Colour
-force: ngl::Vector
-externalForces: ngl::Vector
-velocity: ngl::Vector
-oldVelocity: ngl::Vector
-oldPosition: ngl::Vector
-oldPosition: ngl::Vector
-m_normal: ngl::Vector
-position: ngl::Vector
+makeFree(): void
+makeFixed(): void
+offsetPosition(velocity:const ngl::Vector): voi
+draw(): void
+updateMass(): void
+setMethods(): void
+getMethods(): void
+()
```

Figure 4.1 Mass class Diagram

4.1.1.2 Spring:

This class represents a spring of a mass-spring system. It has references to two particles, which are located on the ends of the springs. As a physical entity the spring has stiffness, damping properties and rest length. There are methods (updateSpring(), correctSpringLength()), which allow to modify spring parameters and transfer this changes to the masses of the spring. These methods are usually called during the integration step.

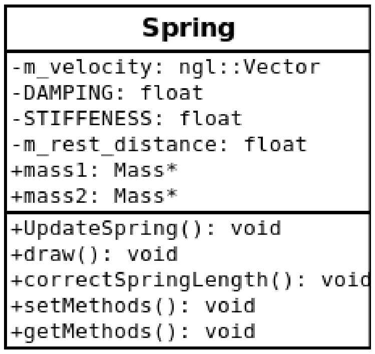
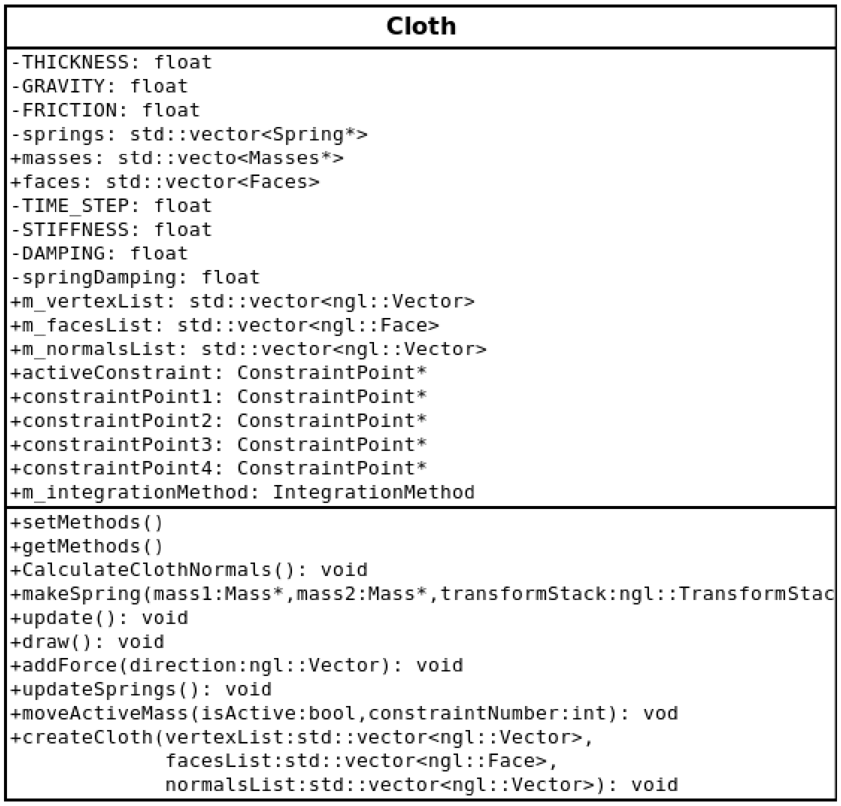


Figure 4.2 Spring Class Diagram

4.1.1.3 Cloth:

Cloth is the main class, which represents the cloth itself. It contains all the properties which physical cloth has: thickness, stiffness, damping, gravity, friction. Because it is build from masses and springs it has lists of masses and springs, which builds particular cloth. The class also contains references to the lists of vertices, faces and normals in order to be able to render it.

In order to update the state of the cloth the timeStep parameter is specified and integration method. The cloth also has references to some of its particles which might be specified by a user as a constraint points. In case it happens so, these constraint points will be fixed during the simulate ion.



```
Cloth
-THICKNESS: float
-GRAVITY: float
-FRICTION: float
-springs: std::vector<Spring*>
+masses: std::vecto<Masses*>
+faces: std::vector<Faces>
-TIME_STEP: float
-STIFFNESS: float
-DAMPING: float
-springDamping: float
+m_vertexList: std::vector<ngl::Vector>
+m_facesList: std::vector<ngl::Face>
+m_normalsList: std::vector<ngl::Vector>
+activeConstraint: ConstraintPoint*
+constraintPoint1: ConstraintPoint*
+constraintPoint2: ConstraintPoint*
+constraintPoint3: ConstraintPoint*
+constraintPoint4: ConstraintPoint*
+m_integrationMethod: IntegrationMethod
+setMethods()
+getMethods()
+CalculateClothNormals(): void
+makeSpring(mass1:Mass*,mass2:Mass*,transformStack:ngl::TransformStac
+update(): void
+draw(): void
+addForce(direction:ngl::Vector): void
+updateSprings(): void
+moveActiveMass(isActive:bool,constraintNumber:int): vod
+createCloth(vertexList:std::vector<ngl::Vector>,
             facesList:std::vector<ngl::Face>,
             normalsList:std::vector<ngl::Vector>): void
```

Figure 4.3 Closs Class Diagram

4.1.1.4 ObjObject:

Because the scene created while simulation may not only contain cloth, but also static objects as well as animation, ObjObject class was created in order to represent any environment object in the scene. Because it is easy to manipulate one list of objects to represent events in the environment rather than to work with two or three lists, it was decided to make an ObjObject class a common class for cloth, animation and static object. In case of animation and static object, there is no difference in manipulating them. In case of cloth, an extra field m_cloth, which refers to the cloth object, will be initialised.

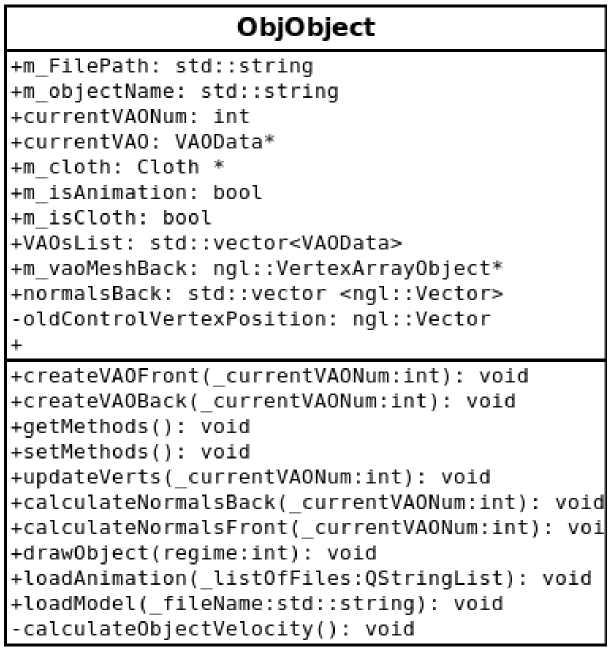
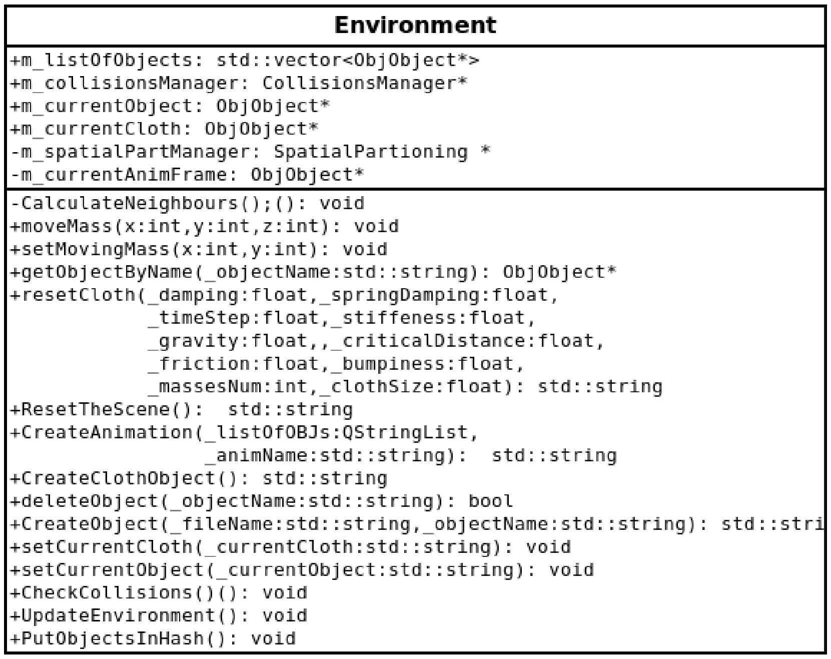

ObjObject
+m_FilePath: std::string
+m_objectName: std::string
+currentVAONum: int
+currentVAO: VAOData*
+m_cloth: Cloth *
+m_isAnimation: bool
+m_isCloth: bool
+VAOsList: std::vector<VAOData>
+m_vaoMeshBack: ngl::VertexArrayObject*
+normalsBack: std::vector <ngl::Vector>
-oldControlVertexPosition: ngl::Vector
+
+createVAOFront(_currentVAONum:int): void
+createVAOBack(_currentVAONum:int): void
+getMethods(): void
+setMethods(): void
+updateVerts(_currentVAONum:int): void
+calculateNormalsBack(_currentVAONum:int): void
+calculateNormalsFront(_currentVAONum:int): voi
+drawObject(regime:int): void
+loadAnimation(_listOfFiles:QStringList): void
+loadModel(_fileName:std::string): void
-calculateObjectVelocity(): void

Figure 4.4 ObjObject Class Diagram

4.1.1.5 Environment:

As have been mentioned in the previous description the Environment class manipulates the ObjObject, which might be cloth, static objects or animation. Environment is responsible for performing all the tasks which come from the user after they have been ‘interpreted’ by m_gl window. These tasks may include: loading of new objects/animation into the scene, conversion of the static object into the cloth, deleting the objects in which the user is not interested any longer, maintain the collisions between the objects of the scene, set-up parameters and integration methods to each particular cloth, descretize all the objects of the scene into hash cells for fast collision detection.

To be able to perform these tasks it uses FilesManager, SpatialPartioning, CollisionsManager, which are described below.



```
Environment
+m_listOfObjects: std::vector<ObjObject*>
+m_collisionsManager: CollisionsManager*
+m_currentObject: ObjObject*
+m_currentCloth: ObjObject*
-m_spatialPartManager: SpatialPartioning *
-m_currentAnimFrame: ObjObject*
-CalculateNeighbours();(): void
+moveMass(x:int,y:int,z:int): void
+setMovingMass(x:int,y:int): void
+getObjectByName(_objectName:std::string): ObjObject*
+resetCloth(_damping:float,_springDamping:float,
            _timeStep:float,_stiffeness:float,
            _gravity:float,,_criticalDistance:float,
            _friction:float,_bumpiness:float,
            _massesNum:int,_clothSize:float): std::string
+ResetTheScene():  std::string
+CreateAnimation(_listOfOBJs:QStringList,
                 _animName:std::string):  std::string
+CreateClothObject(): std::string
+deleteObject(_objectName:std::string): bool
+CreateObject(_fileName:std::string,_objectName:std::string): std::stri
+setCurrentCloth(_currentCloth:std::string): void
+setCurrentObject(_currentObject:std::string): void
+CheckCollisions()(): void
+UpdateEnvironment(): void
+PutObjectsInHash(): void
```

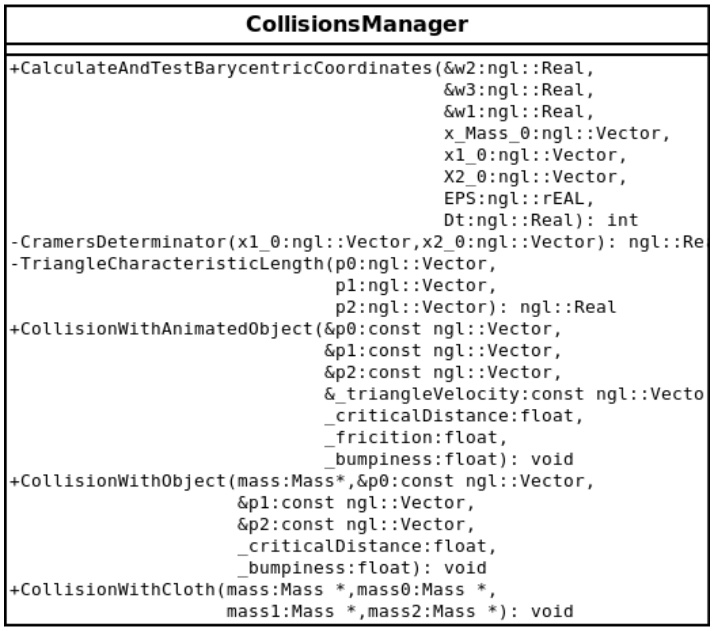
Figure 4.5 Environment Class Diagram

4.1.1.6 Collisions Manager:

CollisionsManager is an independent class, which contains the method for collision detection and collision response. It can be easily expanded to perform new methods of collisions detection/response.

4.1.1.7 Spatial Partioning:

SpatialPartioning class implements all methods for hashing the points of the object in the corresponding cells of the space. It allows to calculate the neighbours of each particles, which will be used to run the collision tests with by a CollisionsManager.



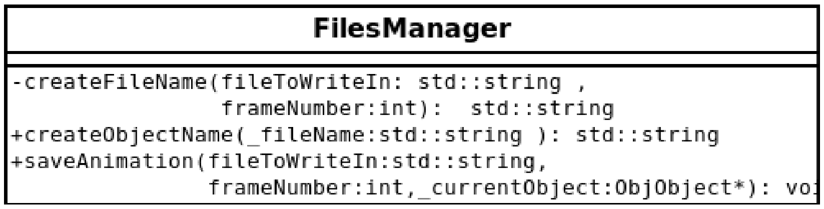
```
CollisionsManager

+CalculateAndTestBarycentricCoordinates(&w2:ngl::Real,
                                        &w3:ngl::Real,
                                        &w1:ngl::Real,
                                        x_Mass_0:ngl::Vector,
                                        x1_0:ngl::Vector,
                                        X2_0:ngl::Vector,
                                        EPS:ngl::rEAL,
                                        Dt:ngl::Real): int
-CramersDeterminator(x1_0:ngl::Vector,x2_0:ngl::Vector): ngl::Re
-TriangleCharacteristicLength(p0:ngl::Vector,
                              p1:ngl::Vector,
                              p2:ngl::Vector): ngl::Real
+CollisionWithAnimatedObject(&p0:const ngl::Vector,
                             &p1:const ngl::Vector,
                             &p2:const ngl::Vector,
                             &_triangleVelocity:const ngl::Vecto
                             _criticalDistance:float,
                             _fricition:float,
                             _bumpiness:float): void
+CollisionWithObject(mass:Mass*,&p0:const ngl::Vector,
                     &p1:const ngl::Vector,
                     &p2:const ngl::Vector,
                     _criticalDistance:float,
                     _bumpiness:float): void
+CollisionWithCloth(mass:Mass *,mass0:Mass *,
                    mass1:Mass *,mass2:Mass *): void
```

Figure 4.6 CollisionsManager Class Diagram

4.1.1.8 Files Manager:

FilesManger is responsible for tasks related to writing/reading the files as well as generating new files names (for the sequences of file, for example).



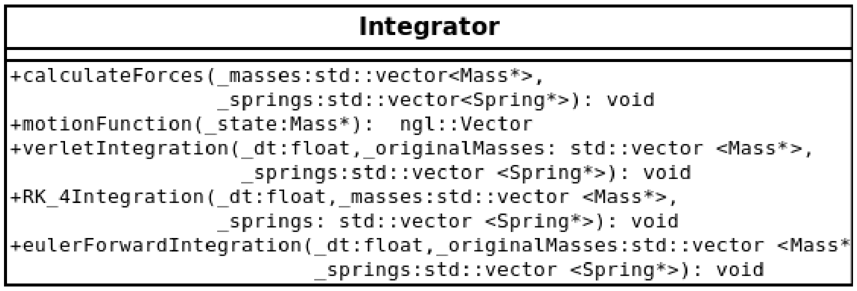
```
FilesManager

-createFileName(fileToWriteIn: std::string ,
                frameNumber:int):  std::string
+createObjectName(_fileName:std::string ): std::string
+saveAnimation(fileToWriteIn:std::string,
               frameNumber:int,_currentObject:ObjObject*): vo
```

Figure 4.7 FilesManager Class Diagram

4.1.1.9 Integrator:

This class is used by a cloth class to update its behaviour. There are three integration methods implemented at the moment Forward Euler, Verlet and Runge-Kutta of 4th order, but it can easily be expanded to any number of integration methods.



```
Integrator
+calculateForces(_masses:std::vector<Mass*>,
                 _springs:std::vector<Spring*>): void
+motionFunction(_state:Mass*):  ngl::Vector
+verletIntegration(_dt:float,_originalMasses: std::vector <Mass*>,
                   _springs:std::vector <Spring*>): void
+RK_4Integration(_dt:float,_masses:std::vector <Mass*>,
                 _springs: std::vector <Spring*>): void
+eulerForwardIntegration(_dt:float,_originalMasses:std::vector <Mass*
                         _springs:std::vector <Spring*>): void
```

Figure 4.8 Integrator Class Diagram

4.2 Export/Import of Files

One of the main requirements of the system developed is a possibility load the objects into the scene from external applications, such as Maya, Houdini, for example.

The tool successfully satisfies this requirement. The .OBJ file format to load the meshes into the scene was chosen as it is a popular format supported by many external modelling tools and is very easy to parse.

In order to load the animation into the tool, the Mel script from Highend3D.com was used to export the sequence of .obj files from Maya.

The developed tool also allows to record the simulation as a sequence of OBJ files. The user need to specify the name of the file in the format: *fileName.$F00...00.obj* and the name for each file of the sequence will automatically be generated by the Files Manger of the tool.

4.3 Creating Cloth Model

For reading an .obj file containing the mesh ngl::Obj class is used. There are some restrictions which are applied to the meshes in this case. The mesh should be triangulated and contain normals and texture information. After the mesh is obtained it is being traversed to create a mass-spring system.

Generation of masses is straight-forward: each vertex of the mesh will correspond to the particle. Creation of springs is more complicated process, because the ,obj file does not provide the information about the order of triangles. In this situation every triangle should be compared with every other triangle to check if they share the edges, and create a spring in case if they do. The algorithm has been described by Luis Pereira (2010):

```
for every face $f_i$ do
  for every vertex $v_i$ in $f_i$ do
    for every other face $f_j$ do
     for every vertex $v_j$ in $f_j$ do
       if $f_i$ and $f_j$ share two edges then
         $v_i$ IsolatedV ertex($f_i$)
         $v_j$ IsolatedV ertex($f_j$)
         CreateBendSpring($v_i$; $v_j$)
      end if
```

```
            end for
        end for
    end for
end for
```

The Figures 5.9, 5.10 illustrates the mass-spring systems which has been created for a rectangular shape:

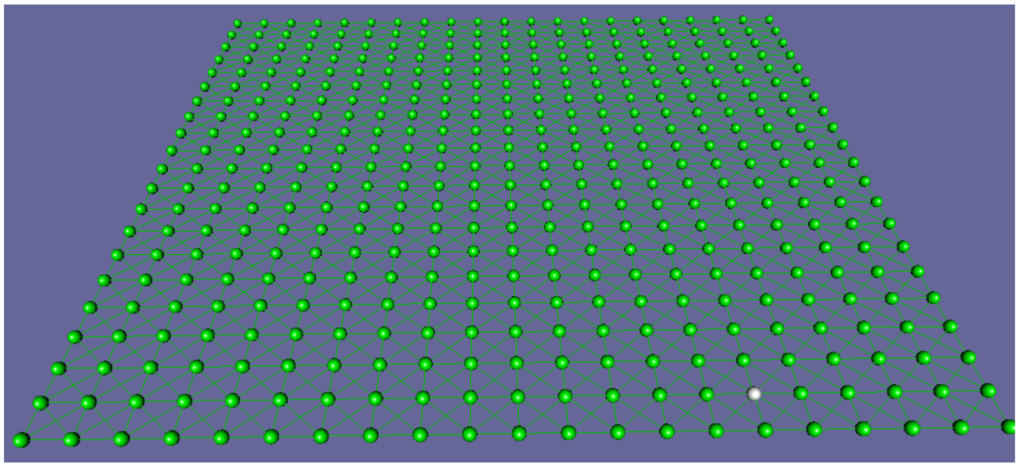


Figure 4.9 Springs created for a quadratic mesh

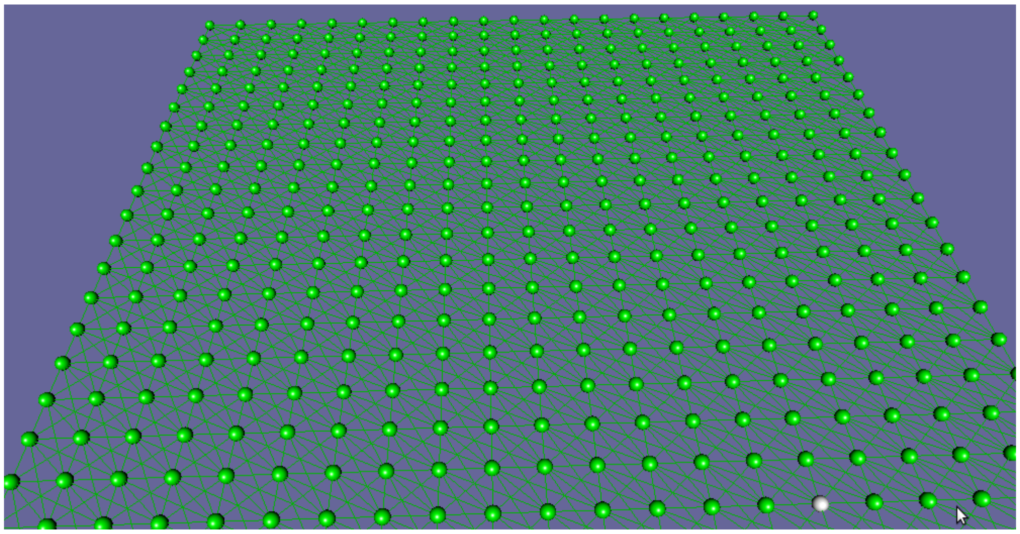


Figure 4.10 Springs created for a triangular mesh

4.4 Rendering the Objects

Apart from mass-spring system visualisation the object of the scene can also be rendered as polygons. Ngl::VertexArrayObject is used for this purpose. For a cloth object the VAO is generated ‘on the fly’, because the vertices and the normals needs to be apdated at each timeStep. When loading the animation using this approach, it caused serious performance issues.

Because the animation data is loaded all at once and we do not need to update it at each frame, it was decided to create the list of VAO objects for each frame of animation at the moment of loading the animation for the first time. This approach requires some time at the moment of first load to create all the VAO objects, but it allows to run the simulation later withought slow-down.

It was also noticed that the two-sided cloth such as rectangular piece, gets rendered only on one side. The other side is displayed black, because the normals of the mesh are oriented in the opposite direction.

To fix this problem, it was possible to use OpenGL option glDisable(GL_CULL_FACE), but the idea was born to display the two sides of the cloth in different colour. This is not possible with the standard OpenGL options. In order to achieve the desirable result it was decided to render the cloth twice. One time with original normals and the second time with the inverse normals and changed colour. The results of such approach are illustrated in Figure5.11.

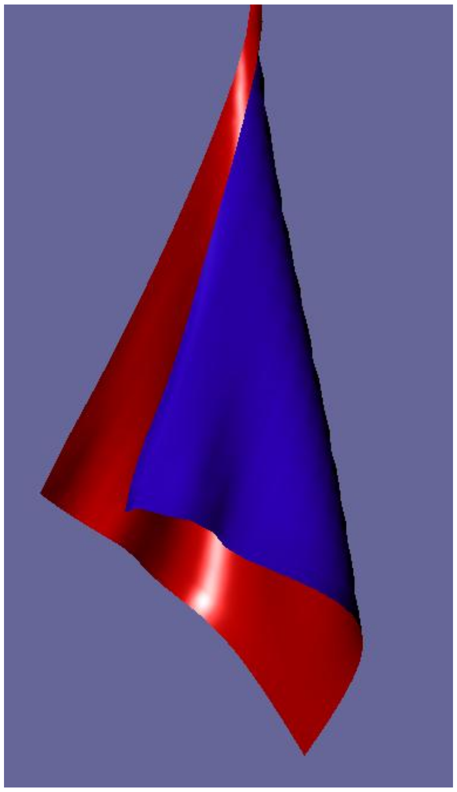
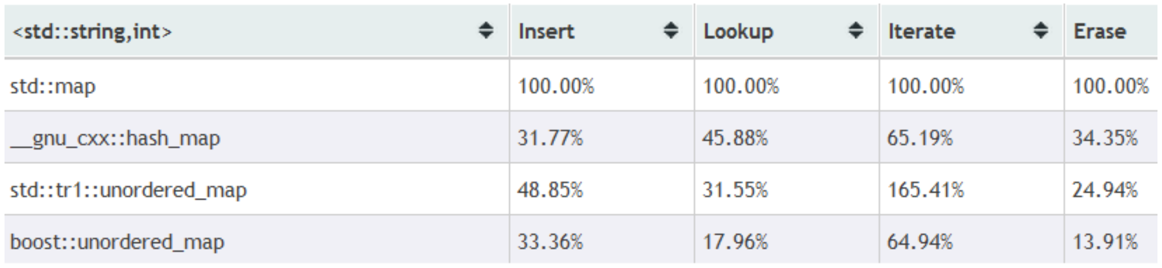


Figure 4.11 Peice of cloth rendered in different colour from different sides

4.5 Spatial Partioning

Spatial Partioning is a technique used to facilitate collision detection by dividing the space into cells and allocating the points of all objects into those cells based on hash algorithm. For implementing this algorithm different structures has been analysed which could be used as a table for storing the data. The candidates were: std::multimap, boost::multiMap. The goal was to select the fastest one in order to maintain the benefits of spatial portioning method. After the research on this topic a comprehensive comparison has been found, implemented by Didriksen, 2009.

The results are demonstrated in the Figure 5.12



<std::string,int>	Insert	Lookup	Iterate	Erase
std::map	100.00%	100.00%	100.00%	100.00%
__gnu_cxx::hash_map	31.77%	45.88%	65.19%	34.35%
std::tr1::unordered_map	48.85%	31.55%	165.41%	24.94%
boost::unordered_map	33.36%	17.96%	64.94%	13.91%

Figure 4.12 Performance test for lookup

The performance of std::map is taken as 100%. Lower the percentage is – the better the performance. We can see from the table above that boost::multimap ‘wins’ in its performance. This became a choice of data container for the spatial table.

4.6 Collisions Algorithm

During the spatial portioning all the verteces of all the objects get allocated in a multimap and the neighbours of each particle gets calculated. The collisions test is run as the next step:

```
For every object obj in the scene do
    If (obj.isCloth == true)  then
        For every mass mass in the obj do
            If (obj.isFixed == false) then
                For every neighbour neib from object neighbours do
                    If (neib == obj)  then
                        STOP;
                End if
                If (neib is outside of BBox of a mass) then
                    STOP;
                End if
                For every face face of the neib do
                  If (neib.isCloth == true) then
                      COLLISION_WITH_CLOTH_TEST();
                  End if
                  If (neib.isAnimation == true) then
                      COLLISION_WITH_ANIMATED_OBJECT();
                  End if
                  If ((neib.isAnimation == false) && (neib.isCloth == false)) then
                      COLLISION_WITH_OBJECT();
                  End If
                End For
            End If
        End For
    End if
End For
For every object obj in the scene do
    If (obj.isCloth == true)  then
        For every mass mass in the obj do
            SET_VELOCITY();
        End For
    End if
End For
```

4.7 Wrinkles Simulation

While implementing this thesis extensive research has been done on enhancing animated garments with folds and wrinkles. This subject is of the great interest today. One of the authors who focused his attention on maintaining realistic wrinkles on cloth is Bridson et al., 2005.

On the last SIGGRAPH conference few different approaches have been suggested to produce fine wrinkles. One of them is to attach a higher resolution mesh to a coarse cloth simulation allowing the wrinkle vertices to diverge from their original position within a limited range, suggested by Muller et al. 2010. Another very fast method was proposed by Wang and colleges, 2010: The method uses a precomputed database of fine wrinkles details and combines it with a coarse cloth dynamics.

One of the most promising approaches today, developed by Popa et al. 2010, is to calculate believable wrinkle geometry using specialized curve-based implicit deformers and combine it with coarse simulation at a next stage.

A big effort has been done in this thesis to implement the method suggested by Popa and his collegues. The main idea of algorithm is to enhance existing coarse simulation with fine wrinkles generated as a separate mesh.

In order to define the position where wrinkles appear, an innovative solution was used. Authors of the paper noticed that the wrinkles appear in the areas of compression. Having this in mind the location of the wrinkles can be defined by calculating the compression field of the cloth. Using this field the wrinkle curves get generated along the field. The actual mesh deformation with wrinkles is performed using convolution surfaces with wrinkles curves as skeletons (Figure 5.13):

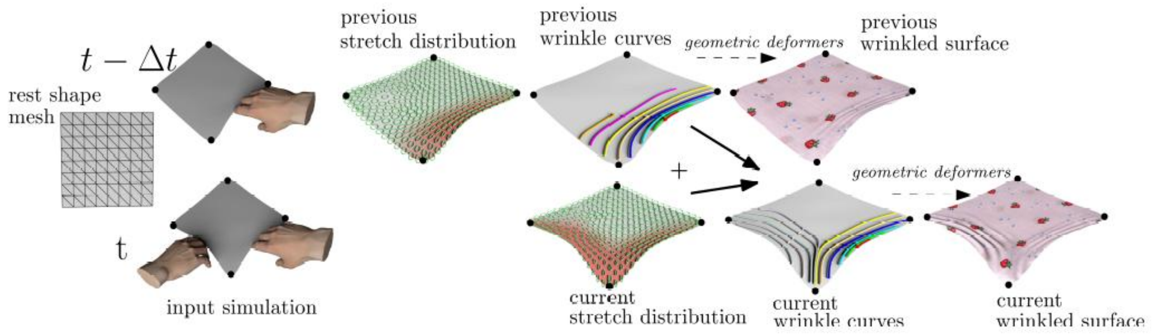


Figure 4.13 Algorithm Overview of Wrinkles Generation, Popa 2010

The main steps to implement this approach include:

1. Calculate the stretch tensor U of the coarse simulation compared to the user-given rest shape
2. Calculate a wrinkle vector field in the cloth surface which magnitude reflects the level of compression
3. Extract the wrinkles curves from the vector field using integration method
4. Define seeds for smooth wrinkle propagation while animation
5. Deform the wrinkle mesh using the convolution model and wrinkle curve as a skeleton
6. Deform the coarse mesh using projection method

Unfortunately, while implementing the method many barriers have appeared. The biggest challenge, for example, was to understand the math behind the method, as equations presented in the paper are very compressed.

Another challenge, which stopped from complete implementation of the method, was appearance of ‘hidden’ steps. For example, in order to be able to project the wrinkles on the coarse mesh it first has to be triangulated. Robust triangulations, which will not slow-down the system is on its own a complex tasks.

Taking these barriers in consideration, the decision was made to complete the wrinkles generating method as part of another project and concentrate on cloth simulation.

However, some interesting results have been achieved:

The areas of wrinkles appearance have been defined using the stretch tensor (Figures 5.14, 5.15).

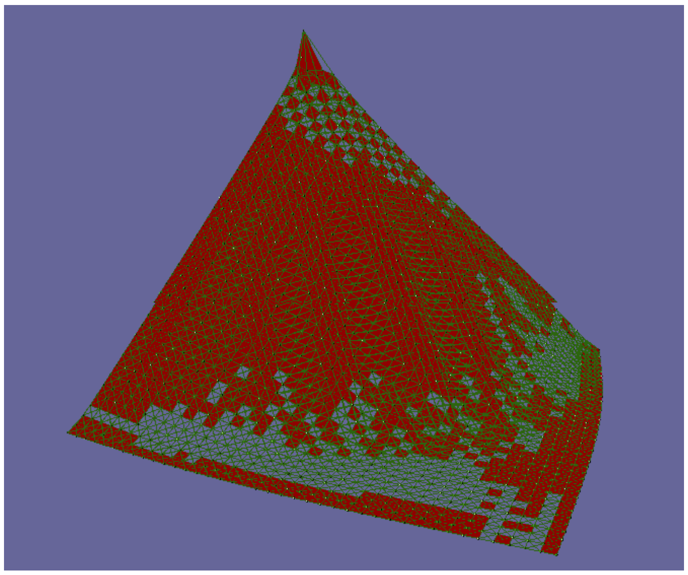


Figure 4.14 Areas of compression (in green)

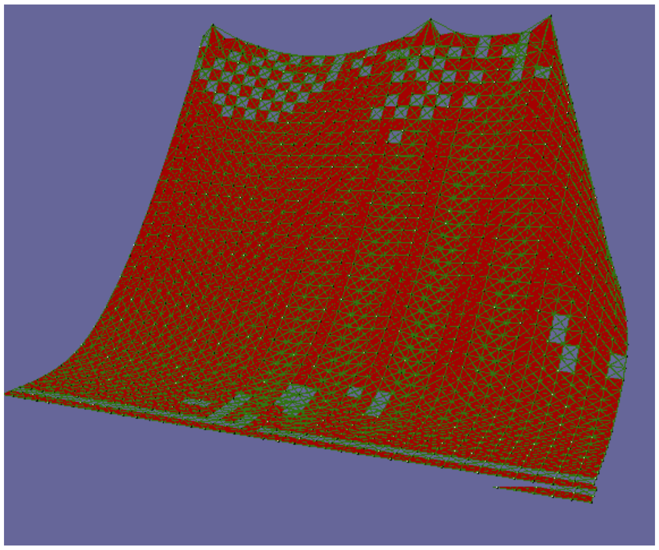


Figure 4.15 Areas of compression (in green)

The deformation of the wrinkle mesh using the convolution surfaces and the wrinkle curve as a skeleton, and projection of the wrinkle into the mesh has been achieved. The wrinkle curve used as a skeleton was drawn by hand. (Figure5.16)

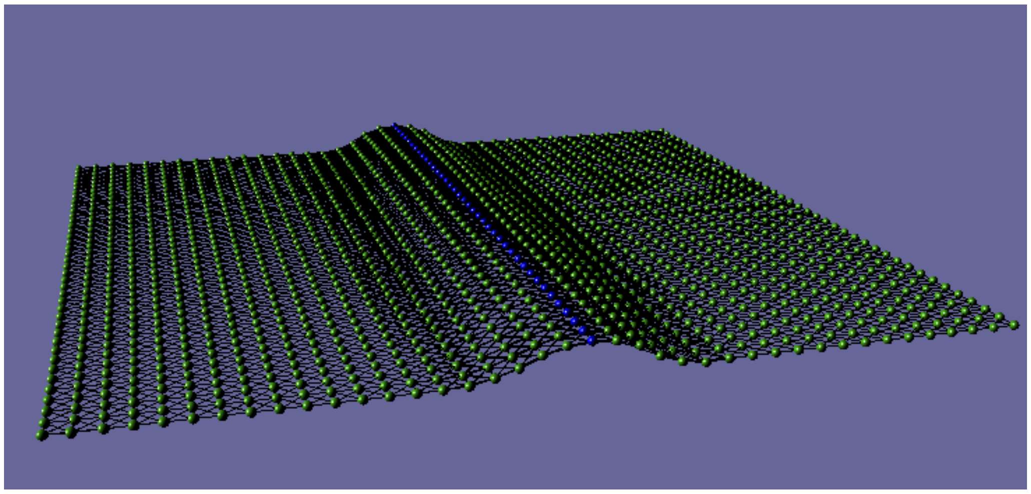


Figure 4.16 a wirunkle generated using convolution surface based on skeleton defined by a line in blue

The missing step between the two accomplishments is automatic generation of wrinkles curves based on stretch tensor. This step was postponed due to the time limit and will be completed in the nearest future.

Chapter Five: RESULTS

The result of the project is implementation of interactive tool for Cloth Simulation, which provides the variety of options to create different types of scenes and behaviour:

- Export of multiple objects into the scene
- Export of animation into the scene
- Possibility to import the simulation created by the tool
- Simulation of different cloth materials

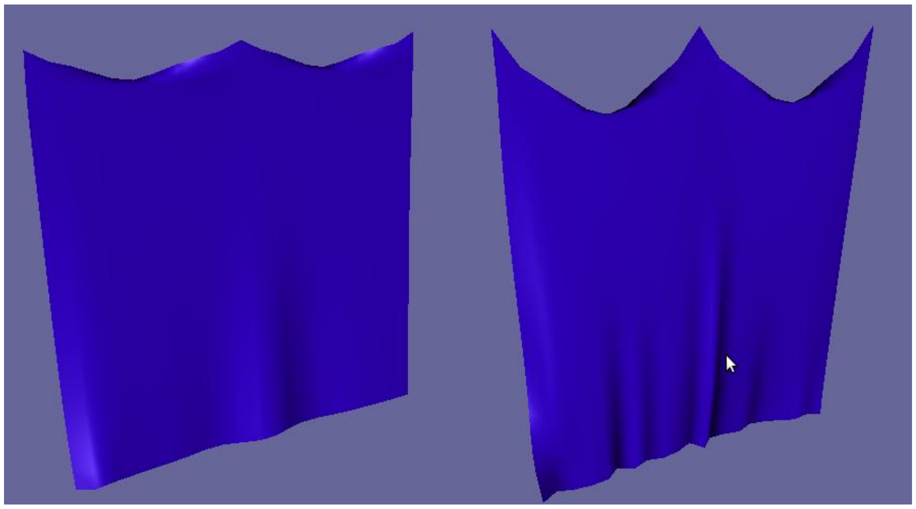


Figure 5.1 Different cloth materials

- Simulation of multiple cloth objects

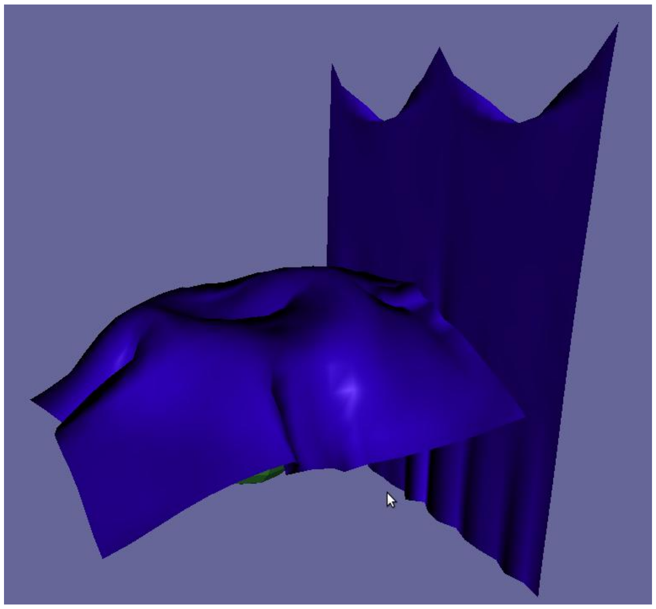


Figure 5.2 Multiple cloth objects in one scene

- Possibility to specify different settings for different objects within one scene
- Collisions handling

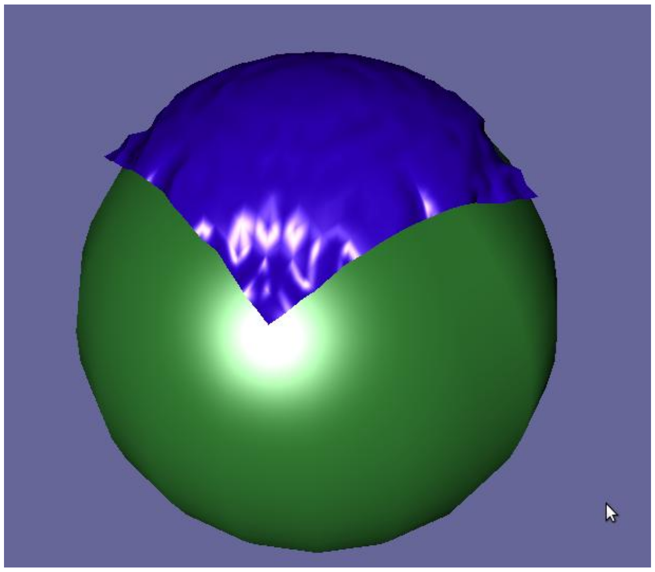


Figure 5.3 Collision with the static object

- Interactive manipulation of the cloth using the constraint points

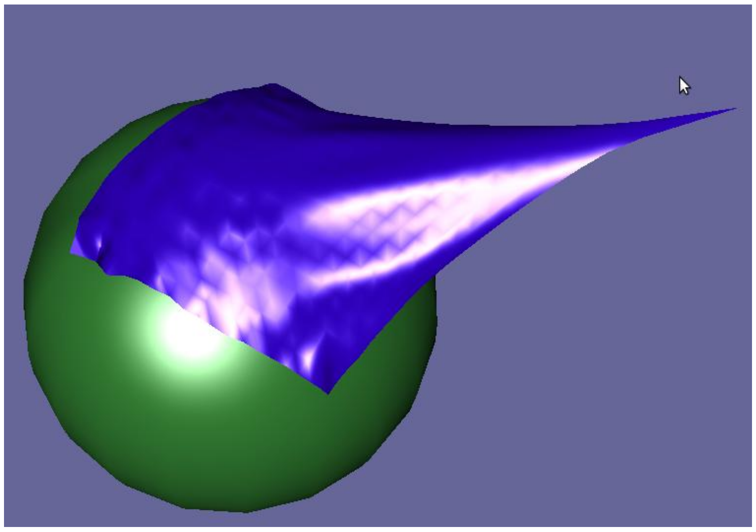


Figure 5.4 Manipulation with the object using constraints. The cloth is being pulled by a user to the right

Limitations:

- With some combination of the parameters the simulation might ‘explode’, due to the integration methods instability. This often happens when changing from Runge-Kuta of 4th Order integration type to the Forward Euler method. Or specifying large step and large stiffness for the cloth. To establish the control over the system, reset button should be pressed or there might be a need to restart the program.
- The collision detection is not performing perfectly. The bumpiness and breakage of the material may appear.

  To avoid the bumpiness many authors suggested to modify while collision handling only the velocity of the particle and do not modify the position. This approach completely removed the ‘bumpiness’, but collision handling appeared a bit late. This is because the velocity change has effect on the neighbouring particles only during the next simulation step.

The breakage of the material may appear while colliding with the animated object. This happens if the speed of the animated object is high enough and the cloth does not have time to modify its state accordingly. Handling these types of situations is the scope of the bigger project and will require implementation of robust techniques on collisions handling with fast moving objects.

- The cloth may penetrate itself while self-collisions. This type of situation only happens when multiple cloth wrinkles appear very close to each other and the cloth particles push each other in all directions. If the wrinkles have some distance between, the method of self-collisions performs perfectly well.

Chapter Six: CONCLUSIONS

The project has been completed successfully and accomplished all the gaols, which has been set. The final product is an interactive Cloth simulation tool, which is easy and entertaining to manage.

There is a scope for the further development of the project in the future:

- Implicit Integration methods may be implemented. This will allow to achieve greater stability.
- Robust collisions handling techniques may be implemented to avoid the limitations which may appear in the current project
- Completion of the wrinkles generation mechanism would add a great realism to the tool
- The tool can be integrated into pipeline
- GPU calculations will allow to run the simulation at the real-time rate even with multiple objects added to the scene

References

Baraff, D. and Witkin, A. (1998). Large steps in cloth simulation, SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques, ACM, New York, NY, USA, pp. 43{54. Available from :
http: // doi. acm. org/ 10. 1145/ 280814. 280821 [Accessed 22 June 2011]

Boesch, F. (2010). Integration by Example - Euler vs Verlet vs Runge-Kutta. Codeflow. Available from : http://codeflow.org/entries/2010/aug/28/integration-by-example-euler-vs-verlet-vs-runge-kutta/ [Accessed 02 August 2011]

Breen, D. E., House, D. H. and Wozny, M. J. (1994). Predicting the drape of woven cloth using interacting particles, SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques, ACM, New York, NY, USA, pp. 365-372. Available from : URL: http: // doi. acm. org/ 10. 1145/ 192161. 192259 [Accessed 17 July 2011]

Bridson, R. Et al. (2002). Robust treatment of collisions, contact and friction for cloth animation, SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques, ACM, New York, NY, USA, pp. 594-603. Available from : URL: http: // doi. acm. org/ 10. 1145/ 566570. 566623 192259 [Accessed 15 July 2011]

Bridson, R., Marino, S. and Fedkiw, R. (2005). Simulation of clothing with folds and wrinkles, SIGGRAPH '05: ACM SIGGRAPH 2005 Courses, ACM, New York, NY, USA, p. 3. Available from : http: // doi. acm. org/ 10. 1145/ 1198555. 1198573 [Accessed 10 June 2011]

Carignan, M., Yang, Y., Magnenat-Thalmann, N. and Thalmann, D. (1992). Dressing animated synthetic actors with complex deformable clothes, Computer Graphics (Proceedings of ACM SIGGRAPH 92), ACM Press, pp. 99-104.

Choi, K. And Ko, H. (2000). Stable but Responsive Cloth. Available from URL: http://graphics.snu.ac.kr/~kjchoi/publication/cloth.pdf [Accessed 27 June 2011]

Desbrun, M., Schr□oder, P. and Barr, A. (1999). Interactive animation of structured deformable objects, Proceedings of Graphics Interface (GI 1999), Canadian Computer-Human Communications Society, pp. 1-8. Available from URL: http: // www-grail. usc. edu/ pubs/ DSB_ GI99. Pdf [Accessed 04 August 2011]

Eberhard, P. And Bischof, C. (1996), *Automatic differentiation of numerical integration algorithms*, Math.Comp. **68** , no. 226, 717–731.

Heidelberger, B. Et al. (2010). Optimized Spatial Hashing for Collision Detection of Deformable Objects. Available from URL:

http://www.beosil.com/download/CollisionDetectionHashing_VMV03.pdf [Accessed 14 June 2011]

House, D. H. and Breen, D. E. (eds) (2000). Cloth modeling and animation, A. K. Peters, Ltd., Natick, MA, USA.

Kieran, E et al., L. (2005). Cloth simulation, Master's thesis, Bournemouth University, Poole, UK. Available from URL: http: // nccastaff. bournemouth. ac. uk/ jmacey/MastersProjects/ Msc05/ cloth_ simulation. Pdf [Accessed 15 August 2011]

Liberatore, M. (n.d.). Comparing numerical integration methods in a simulator for the draping behavior of cloth. Available from URL: http://src.acm.org/liberatore/ liberatore.html. [Accessed 12 June 2011]

Magnenat-Thalmann, N and Thalmann, D., first, 2004 *Handbook of Virtual Humans*.Chichester: John Wiley & Sons Ltd.

Moller, T. (1997). Fast, Minimum Storage Ray/Triangle Intersection. Available from URL: http://jgt.akpeters.com/papers/MollerTrumbore97/ [Accessed 15 August 2011]

Ng, H. N. and Grimsdale, R. L. (1996). Computer graphics techniques for modeling cloth, IEEE Comput. Graph. Appl. 16(5): 28-41. Available from URL: http: // dx. doi. org/ 10. 1109/ 38. 536273 [Accessed 08 August 2011]

Parent R, 2002.Computer Animation Algorithms and Techniques, Morgan Kaufmann Publishers, San Francisco, USA

Pereira, L. 2010. C++ Cloth Simulation. Master's thesis, Bournemouth University. Available from URL: http://nccastaff.bournemouth.ac.uk/jmacey/MastersProjects/MSc2010/07LuisPereira/doc/html/index.html [Accessed 02 June 2011]

Rohmer, D., Popa, T., Cani, M., Hahmann, S. and Sheffer, A. 2010. Animation wrinkling: Augmenting Coarse Cloth Simulations with Realistic-Looking Wrinkles. In: ACM SIGGRAPH, December 2010, Seoul, South Korea. ACM Transaction on Graphics (TOG), Available from URL: http://www-ljk.imag.fr/membres/Damien.Rohmer/documents/publications/10_sigasia_wrinkle/10_sigasia_wrinkle.html [Accessed 12 Feb 2011].

Provot, X. (1995). Deformation constraints in a mass-spring model to describe rigid cloth behavior, Graphics Interface '95, pp. 147-154.

Provot, X. (1997). Collision and self-collision handling in cloth model dedicated to design garments, Proceedings of the Eurographics Workshop on Computer Animation and Simulation (CAS 1997), Springer-Verlag, pp. 177-189. Available from URL: http: // www-rocq. inria. fr/ mirages/ SYNTIM_ OLD/ textes/Collisions_ vetements. ps. Gz

[Accessed 07 July 2011]

Selle, A., Su, J., Irving, G. and Fedkiw, R. (2009). Robust high-resolution cloth using parallelism, history-based collisions, and accurate friction, IEEE. Transactions on Visualization and Computer Graphics 15(2): pp. 339-350.
Available from URL: http: // dx. doi. org/ 10. 1109/ TVCG. 2008. 79
[Accessed 14 July 2011]

Terzopoulos, D., Platt, J., Barr, A. and Fleischer, K. (1987). Elastically deformable models, Computer Graphics (Proceedings of ACM SIGGRAPH 87), ACM Press, pp. 205-214.

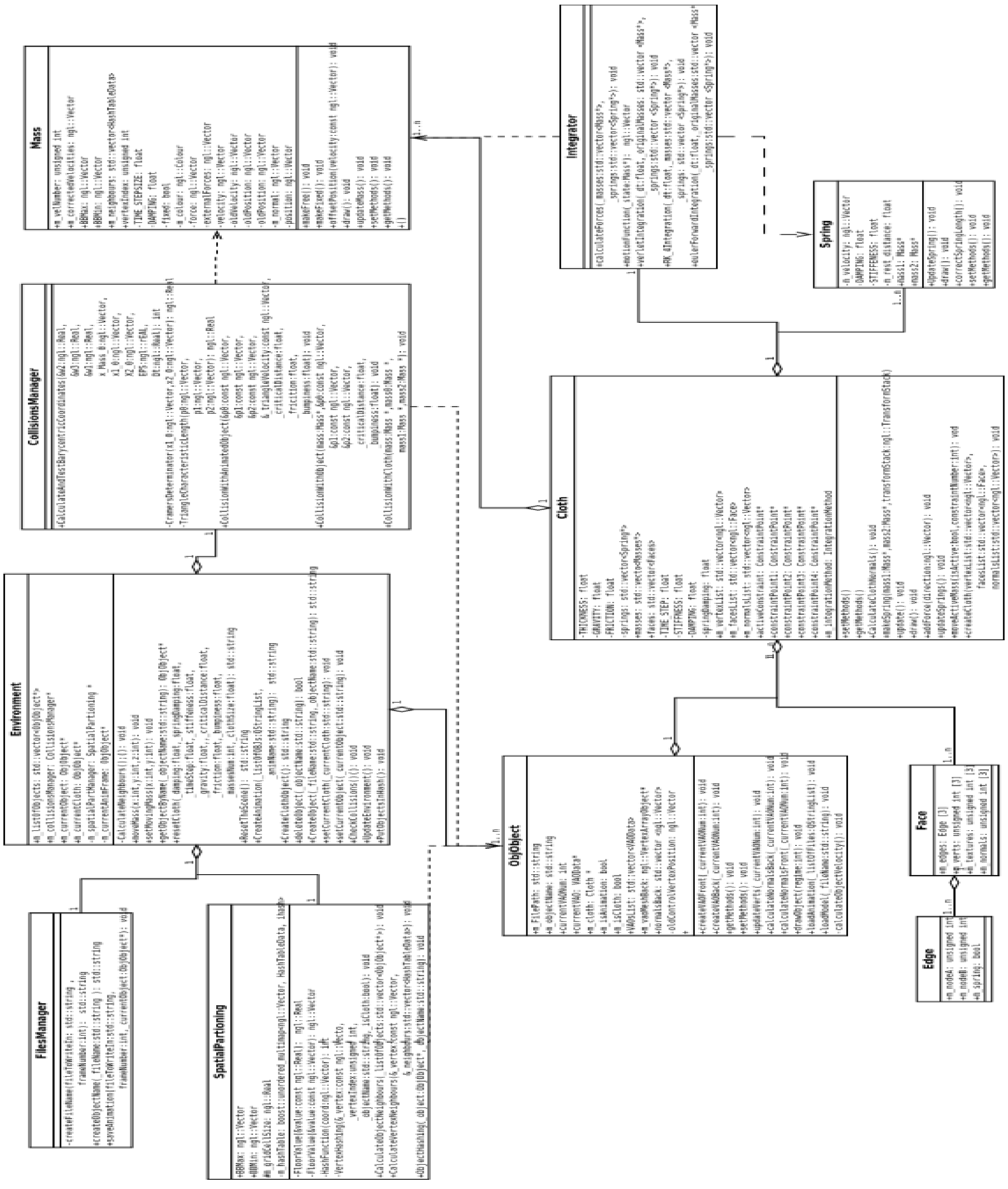
Villard, J. and Borouchaki, H. (2005). Adaptive meshing for cloth animation, Eng. with Comput. 20(4): pp.333-341. Available from URL: http: // dx. doi. org/ 10. 1007/ s00366-005-0302-1 [Accessed 23 June 2011]

Volino, P. and Magnenat-Thalmann, N. (2000). Implementing fast cloth simulation with collision response, Proceedings of Computer Graphics International (CGI 2000), IEEE Computer Society, pp. 257-268. Available from URL:http: // www. miralab. unige. ch/ papers/ 47. Pdf [Accessed 04 August 2011]
Volino, P. and Magnenat-Thalmann, N. (2001). Comparing efficiency of integration methods for cloth simulation, Computer Graphics International Proceedings, IEEE

Computer Society, pp. 265-274.

Volino, P, Cordier, F. andMagnenat-Thalmann, N (2005). From early virtual garment simulation to interactive fashion design. Computer-Aided Design Journal, Elsevier, 37:593–608

Appendix A.


Mass
CollisionsManager
Environment
FilesManager
SpatialPartioning
Integrator
Spring
Cloth
ObjObject
Face
Edge
1..n
1