CLOTH MODELLING: FINAL REPORT

LIAM CHRISTOPHER LEES

SUBMITTED IN PARTIAL FULFILMENT

OF THE PREREQUISITE FOR THE DEGREE OF

Bachelor of Science

in

THE DEPARTMENT OF COMPUTER SCIENCE

(Faculty of Undergraduate Studies)

The University of Hull

May 2016

WORD COUNT: ~14,900

Abstract

Cloth simulation is one of the hardest aspects of computer graphics, it is a deceptively simple real-world item that is taken for granted, yet actually has very complex internal and environmental interactions. This project seeks to explore and implement the techniques employed by leaders in both industry and research.

keywords cloth, interactive, mass-spring, euler, verlet, runge-kutta, collisions, implicit, IMEX, real-time, physically based simulation

Table of Contents

"Imagination will often carry us to worlds that never were.
But without it we go nowhere."
Carl E. Sagan

1 Introduction

"The drape of woven cloth has intrigued humans for centuries. This is evident in the flowing of robes contained in the sculptures of ancient Rome, the intricate folds of fabric depicted in the paintings of the Renaissance, and the elaborate billowing clothing of the 18th century. Even in the modern times artists such as Christo realised that the image of draping cloth over a structure like the Reichstag in Berlin is fascinating and provocative. It has always been clear that woven materials have unique properties that allow them to deform in ways significantly different than other sheet, e.g. paper, vinyl and metal foils. Cloth's special deformation capabilities have been noted and recognised through the ages but never fully understood from scientific or engineering perspective."

David E. Breen, 1994

This statement by Breen is a good way to introduce cloth modelling. Cloth isn't like most materials; it does not have simple static forms that can be geometrically modelled. The change in shape of cloth is dependent on the state of the cloth, and because of this the complex states the cloths can conform to, requires unique methods to be formulated in order to describe it effectively.

Physically based modelling has become an active and significant area of research in computer graphics. Emerging in the late eighties to the early nineties out of the need for *true* simulations that could relieve animators from explicitly specifying complex passive motion. Although physically based models are strongly grounded in mathematics, the maths involved needn't be any more difficult than the maths that underlines many other areas of computer science. The problem of animating cloth has received wide examination since the early days of computer graphics research.

The motion simulation of cloth is a fundamental cornerstone of, among other things, virtual character animations. Convincing animations are now required in both the games and motion picture industry. In the garments industry, textile engineers and textile designers have also become attentive to progressions in this area, as their interests lie in predicting specific fabric behaviour such as it's motion and drape. By applying these understandings garment manufactures hope the CAD/CAM techniques used in traditional engineering can be used to reduce prototype yields of garments.

Generally an engineers focus lie in the physical accuracy of their models whereas computer graphics researchers place prominence on the computing performance of such models. Fast, accurate and generalised techniques form the pillars of quality research in this field.

This project seeks to examine a number of methods employed by the leaders in this field. In doing so we investigate the process of modelling an effective cloth simulation.

2 Background

The cloth simulation problem has attracted much research over the past thirty years from the computer graphics community. A collection of unique techniques have been proposed over this time, each one producing differing results. This section seeks to briefly cover the relevant research in the field of cloth modelling. However, more in-depth investigations and backgrounds are presented, where applicable, throughout this report.

This problem has of course been previously covered in engineering literature, typically implementing finite element models supported by rigorous quantitative analysis. Some principal papers in this area are, in no particular order, (Collier *et al.,* 1991), (Zhao *et al.,* 1997), (Ascough, Bez, & Bricis, 1996), (Tan *et al.,* 1999) and these are overviewed in later sections of this report.

Thorough examinations of early research in this field can be found in the works of (Ng & Grimsdale, 1996), (House & Breen, 2000), (Volino & Magnenat-Thalmann, 2000). A more up-to-date comprehensive study of the field and associative cross-over computer graphics fields can be found in the excellent 2008 SIGGRAPH course notes (Müller *et al.*, 2008).

The first cloth simulation was produced in 1986 in (Weil, 1986). Presented was a geometrically based approach involving catenary functions manipulated to resemble the drape of cloth when suspended at points of constraint. In the same year, in his Masters thesis, (Feynman, 1986) proposed a physically based model grounded in continuum mechanics which employed a "multi-grid" solver that minimised the energy of the system to make prediction of the deformations of the fabric when draped over simple solids.

(Terzopoulos *et al.,* 1987, 1988), in their work provided a more extensive solution. Employing a continuum mechanics approach, their elastic model made predictions of the dynamics of a cloth in three spatial dimensions. A finite discretisation of Lagrangian equations of motion were formulated in their model, they also employed a symplectic time integration scheme. The model demonstrated was capable of handling contact with static objects and simple aerodynamical forces. Their work gave the ground work to future research that followed.

Beginning in the early nineties, a research group based out of the University of Geneva's MIRAlab laboratory began pioneering contributions to various computer graphics paradigms of the time. Of the goal's they tackled, one was to dress virtual characters. Their work built upon the work of (Terzopoulos *et al.,* 1987, 1988), modifying the damping mechanics and collision detection. Their collision model also handled cloth self-collisions and response. The group's work to date has continued to improve models, focusing more on the specific areas of

self-collision consistency and the efficiencies of various numerical methods and their behaviours.

In 1994, (House *et al.*, 1994), presented a new approach to cloth modelling utilising particle-systems and elastic springs. They proposed that cloths internal mechanical properties were of warp and weft networks, not the continuum approach of past research. Therefore, they made a claim that their model was a more appropriate solution to the problem. Using supporting data collected from the *Kawabata Evaluation System* (Kawabata, 1980) and a minimising energy approach similar to that presented in (Feynman, 1986) only using a gradient descent algorithm (background to this algorithm is given in *Appendices* [**B**]), their model produced predictions of real materials draping quite accurately. In 1996, (House, DeVaul, & Breen, 1996) formulated a Newtonian particles-system model to simulate the dynamic motion of cloth. Although, it was found that structural complications would arise when attempting to maintain the cloths structure, this was found to be directly related to the systems stiffness sum. To overcome this they elected to change the structure, substituting their current constraints for fixed-length constraints. Specifically devised for the problem, they used a ranked Lagrange multiplier technique. The next year, (DeVaul, 1997) in his work advocated for a constraints based model that resolved convergence iteratively, outlining the flaws of the mass-spring models that were quickly gaining traction in the field. However, constraint modelling saw little adoption, it would be years until researchers switched to the model proposed by DeVaul.

In 1995 a more simplistic and complete model for cloth was presented in the work of (Provot, 1995). This was a particle based mass-spring model which was able to simulate cloths motion efficiently. The work dealt with the issue associated with mass-spring systems inherent stiffness issues, identified as the 'super-elastic' effect, solved by use of inverse dynamics concepts to constraint enforcement. Springs that were identified to have stretched 10% of their original rest length were relaxed, having the effect of stretching neighbouring springs which in turn would relax the system until convergence was satisfied. In practice the model would converge with pleasing results. Subsequent research sought to tackle the problem of parameter estimations and collision detection and response in research found in (Louchet *et al.*, 1995).

In their seminal paper, (Baraff & Witkin, 1998), introduced an altogether new approach formulating a complete implicit numerical integration model allowing for relatively large step sizes without falling victim to the instabilities plagued by previous research. This proved to be a robust solution to the stiffness problem. In the same paper they also proposed a modified pre-conditioned conjugate gradient solver which could enforce constraints within the implicit phase. Their model, based on some influences from continuum finite element

methods (FEM) could handle generalised mesh topologies. Following this work, (Baraff, Witkin, & Kass, 2003) working with Pixar inc., introducing post intersection analysis techniques to handle degenerate situations occurring in cloth self-collision situations.

In view of (Baraff & Witkin, 1998), others e.g. (Desbrun, Schröder, & Barr, 1999), (Kang *et al.,* 2000), have strived to the enhance the computational performance of this approach. Further specifics as well as the original research is examined in later sections.

In 2002, (Choi & Ko, 2002), building on the foundations of the numerical techniques employed in (Baraff & Witkin, 1998), they present a particles-based model that could overcome pre *blow-up* instabilities, a problem area among researchers prior to this. Their model used a more realistic, non-linear bending formulation. In later work, (Choi & Ko, 2003) protracted their model to support generic triangular mesh topologies.

(Bridson, Fedkiw, & Anderson, 2002), and (Bridson *et al.*, 2003), focused on maximising the visual verisimilitude of cloth simulation. Among their contributions are the robust handling of collisions, friction and contact, and a unique *implicit-explicit* scheme. This and other innovations are discussed further in later Sections.

3 Models

Comprised of woven threads, it is the arrangement of the weave and fibre type making up a cloth that arbitrates its visual properties, its motion, and the way it feels, etc. For an induction into cloth from a modeller's viewpoint, excellent insight is provided in the first chapter of (House & Breen, 2000).

A real world element of cloth can be defined by several geometrical and physical characteristics. Further, it is abstractly thought of as a surface of two dimensions propagating a space of three spatial dimensions. It experiences deformational stretches tangential to the plane, and stretches perpendicular or normal to the surface (out of plane). For most fabrics cloth has an inherent intransigence to deformations along its plane, generally we observe much less resilience to out-of-plane displacements. In addition, the standardised system for the measurement of fabric deformations, the Kawabata Evaluation System (Kawabata, 1980), tells us that most fabrics tend not to resist orthotropic stretching and that cloth more easily stretches along shear planes that are inline with the fabric's fibre directions.

As with many physical phenomena, it is an unattainable proposition to make exact models of cloth, or to simulate its motion. The requirements to do so would involve formulations that would accommodate for quantum mechanical physics not even properly understood by leading physicists of the respective field. If we were to suppose that we do indeed understand the discreet physics at play, the fundamental problem would still be an insurmountable computational task. Thus the aims and scope of this project must therefore be much more modest in it's expectations. The properties we seek in the physical model we implement are:

- Efficiency, from a computational standpoint
- Fidelity, with respect to the visual accuracy of the statics and dynamics
- Reactivity, ability to behave appropriately to internal and external stimuli

This section gives concise analysis of the two dominant models, examined in the relation of continuum and particle-system models.

3.1 Particle-Systems & Finite Element Methods

3.1.1 FEM Models

Models grounded in continuum mechanics have origins founded in the study of the deformation of elastic continua with regards to the theory of elasticity (Hutchinson, 1989), owing a lot of its advancements to a long line of historical notable contributions such as Euler, Poisson, Laplace, Bernoulli, Green, and many others. Continuum models consider a body to have consistent structures, this allows the system to be defined by a *finite* system of first-order Jacobian differentials.

Regularly deployed by engineers in their models for various deformable structures. They are typically numerically solved using the principals of finite methods. FEM is a method that seeks to make predictions of functions which satisfy the equilibrium states of deformations between a predetermined set of elements that form the the structure. Conditions are also put in place to ensure continuity of the function is also satisfied. An assortment of formulations of element types have been used in this way, including beams, plates, and shells.

Continuum solutions, in a variety of forms have been applied across computer graphics disciplines. (Terzopoulos *et al.*, 1987, 1988) solution involved the use of a simplified set of elasticity formulations with their derivations lending to finite differencing techniques. Whereas in (Baraff & Witkin, 1998), the solution proposed entailed the use of a segmented mesh representation of cloth. From this they adopted a discrete continuum formulation that was applied to the mesh on a segment-by-segment basis, this formed the groundwork for calculations of stretching both in-plane and out-of-plane.

3.1.2 Particle-Systems

Considered to be the most elementary approach to simulate deformable structures, is the representation in terms of the self-styled ‘particle-system’ (occasionally attributed as a ‘mass-spring’ system). A plain section of the archetype is presented in Fig. 3.1. The model represents a discretised arrangement of mass-points. These points are then connected to one another by a set of springs conditioned to oppose the anamorphosis of the construction.

It was (Breen, House, & Wozny, 1994) that originally popularised particle-systems for the application of simulating cloth. The research group contended that fabric is not of homogeneous consistency, but a system of threads woven into an interlocked arrangement; cloth is said to be controlled by a state governed by friction as opposed to molecular bondings. The group ran tested their model in various settings, contrasting their observations with the data collected from the Kawabata Evaluation System (Kawabata, 1980), they found their model had acceptable coherence.

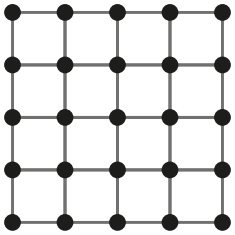


Fig. 3.1: Segment of a simple mass-spring mesh

At a conceptual level, modelling a cloth is rather visceral, hence its reputation. Indeed, the particle models present in research are typically much more intrinsic than the one depicted above.

3.1.3 The Shared Relation of Continuum (FEM) & Particle-Systems

One might ask how these models relate to one another. Are they similar? Indeed, various researchers have investigated such questions. (Kass, 1995), proves that there was an identifiable correspondence between these two approaches when formulated for a single dimension. (Etzmuß, Groß, & Staßer, 2002), demonstrated that their particle based model was defined as a contiguous semi-discretisation of a continuum formulation. However, it would seem more apparent such disparity found in the differing particle-systems and FEM models are as meaningful as the separation of the two categorisations. That is to say the question as to which approach is best to tackle cloth may be immaterial.

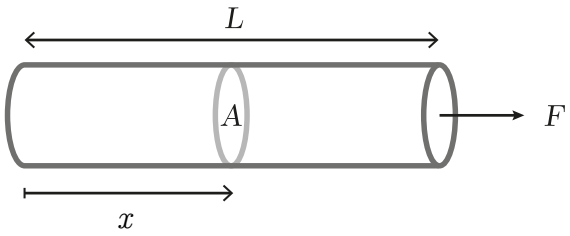


Fig. 3.2: Element experiencing axial in-plane deformation

Further examination is required to define the relationship between these two approaches. Here we examine formulations in one dimension. In this case they are identical. A physical construction is represented in Fig. 3.2. Depicted is a beam like cloth thread with a length L, cross-section A, and a mass per length unit ρ the beam is constrained to deform according to a Young's modulus of E and a damping coefficient μ along it's axial plane. Continuum mechanics models the behaviour of this continua to conform to the following partial differential equation (PDE) shown below (*Eqn. 3.1*)

$$\rho\ddot{\varepsilon} = EA\frac{\partial^2\varepsilon}{\partial x^2} + \mu\frac{\partial^2\dot{\varepsilon}}{\partial x^2} \qquad \textit{Eqn. 3.1}$$

where ε here is the the material's direct strain and \ddot{ε} follows as the strain rate

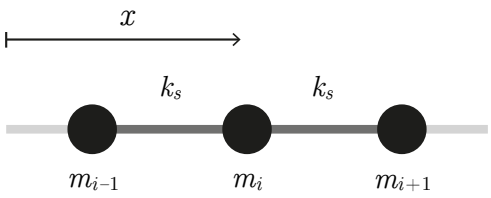


Fig. 3.3 section of a 1D particle-system

Conversely, Fig. 3.2 is modelled as a particle system and is depicted in Fig 3.3. Instead, here the fibre is abstractly thought of as a physical assortment of mass-points m_1, ..., m_n, individually displaced by an elastic spring constraint with dimension h, h here is comparable to the formulation $L/(n-1)$, damping coefficient k_d, and a springs stiffness constant of k_s. Every point is said to have a mass ρh. The force f on a particle i of the system is given by the following formulation (*Eqn. 3.2*)

$$f_i = k_s(x_{i+1} - 2x_i + x_{i-1}) + k_d(\dot{x}_{i+1} - 2\dot{x}_i + \dot{x}_{i-1}) \qquad \textit{Eqn. 3.2}$$

in the above, x_i and $\ddot{x}_i$ are the positions and velocities, respectively, of particle i. The dynamics of this system and indeed all derivative systems using this model adhere to the Newtonian law, $f = m\ddot{x}$, applying this knowledge yields the following (*Eqn. 3.3*)

$$\rho h\ddot{x}_i = k_s(x_{i+1} - 2x_i + x_{i-1}) + k_d(\dot{x}_{i+1} - 2\dot{x}_i + \dot{x}_{i-1}) \qquad \textit{Eqn. 3.3}$$

From applying variable modifications for this single dimensional abstraction, the strain ε is identified as being the divergence rate bounded by mass members' last and current positions. If we propose that the beam's left bound as the origin, the relationship between a point's displacement ε, and its position x along with the first and second derivatives compound to form (*Eqn. 3.4*)

$$\rho h\ddot{\varepsilon}_i = k_s(\varepsilon_{i+1} - 2\varepsilon_i + \varepsilon_{i-1}) + k_d(\dot{\varepsilon}_{i+1} - 2\dot{\varepsilon}_i + \dot{\varepsilon}_{i-1}) \qquad \textit{Eqn. 3.4}$$

From the above we can separate the central finite difference when considering the higher-order derivatives, the formulation is given below (*Eqn. 3.5*)

$$\frac{\partial^2 \varepsilon_i}{\partial x^2} = \frac{1}{h^2}(\varepsilon_{i+1} - 2\varepsilon_i + \varepsilon_{i-1}) \qquad \textit{Eqn. 3.5}$$

thus a rearrangement of the formulation obtains (*Eqn. 3.6*)

$$\rho\ddot{\varepsilon}_i = k_s h\frac{\partial^2\varepsilon_i}{\partial x^2} + k_d h\frac{\partial^2\dot{\varepsilon}_i}{\partial x^2} \qquad \textit{Eqn. 3.6}$$

We now can deduce that the two formulations for a continuous model and particle-system model, (*Eqn. 3.1* & *Eqn. 3.6*) share relationships between E and k_s, and between μ and k_d.

3.2 Positions Based Approach

The approaches covered previously give solutions for the simulation of dynamic systems by means of calculations bounded by the force properties of the system. Forces (internally and externally) are collected from which accelerations can be drawn from based on Newtonian mechanics. A time integration method is applied, to then update velocity quantities from which a new position is obtainable. A few methods also use a velocity impulse based model from which velocities are directly manipulated, although typically these methods have a tendency to be quite unstable.

However, more recently new solutions to simulation with implications applicable to the soft body field have been proposed. Revolving around the direct manipulation of positional data this area of active interest is known as *Position Based Dynamics*, we discuss the various models that encompass this approach.

3.2.1 Jakobsen's Method

(Jakobsen, 2001) can be thought of as the first work to employ and explain elements of what Position Based Dynamics is today. Proposes a totally constraints based model for simulating soft-body and rigid-body dynamics. The method is straightforward in retrospection and relatively simple, requiring just one pass over every constraint in the system per interval. The researcher makes claims that the model *probably* cannot be done any faster than what were presented

A system similar to that of the particle-system illustrated in Fig. 3.1 is employed. Like the particle-system, the mesh used forms the basic framework, again a similar constraints configuration connects nodes to one another. Purely a positions based approach revolving around a derivation of a verlet formulation (discussed further in section 4.3), forces here are only ever evaluated if they are external to the system.

3.2.2 Müller *et al.* Method

(Müller *et al.*, 2007) presents a method that builds on the concepts of constraints management in (Jakobsen, 2001), generalising the approach and applying projections based on the rate of change of the constraint function. The method uses an Eulerian scheme to

calculate new positions at every timestep. They also present a constraints solution that can effectively adjust the out-of-plane bending function without affecting the stiffness function of constraints.

By working on positional units directly, one solves for the equilibrium and *projects* the position. This gives rise to a system of second order equations, solved by iteratively calculating the constraints functions projected positions.

3.2.3 Position Based Verlet & Position Based Dynamics®

This section seeks to clarify the technical differences of the often confused (Jakobsen, 2001) approach and Position Based Dynamics (PBD), (Müller *et al.*, 2007).

For any method used in cloth simulation, three things co-exist:

- Calculation of external forces
- Application of constraint enforcement (i.e. calculation of internal forces)
- Integration of positions and velocities

Simulations attempt to solve 2nd order stiff ODEs by means of integrating 2 first order differential equations (positions and velocities). Verlet as proposed in (Jakobsen, 2001) is a method of integration that approximates the ODE by using a single formulation for calculating the new position by an implicit representation of the velocity (a technical examination is found in section 4). Alternatively PBD is a method of solving the 2nd order equation by first calculating an initial prediction of new positions using an explicit Euler scheme. Since explicit computation is not accurate enough, a constraint condition is applied on the predicted positions to validate the values. Once predicted values pass this stage, they are integrated to the new position.

We can use verlet or any other integration scheme used in the final phase of PBD as well to attain a new position. So the difference is clear, Verlet is an integration scheme, PBD is an algorithm that may use Verlet integration in the final step to get new positions.

3.3 Cloth Modelling

In our work, we present an implementation of the mass-spring model borrowing aspects from the approach proposed in (Provot, 1995). See Fig. 3.4, each mass-point in the mesh are joined by three classes of spring;

- *Structural* springs (stiff), where a mass-point is joined to its four adjacent neighbours (in-plane)
- *Shear* springs (less stiff), where a mass-point is joined to its four diagonal neighbours (in-plane)
- *Bend* springs (weak), where a mass-point is joined to the next eight immediate neighbours (out-of-plane)

Managing how these springs constrain fundamentally dictates the behaviour of the cloth. (Breen, House, & Wozny, 1994), in their work uniquely managed shear deformation by controlling the rotational function of torque, specifically ignoring the central energies involved. Employed was a non-linear function of energy aligned parallel to the fibre orientations of fabric to accurately control out-of-plane deformations. Expanding on the Breen model, (Eberhardt, Weber, & Straßer, 1996) included in their model non-continuous phenomena such as the lagging hysteresis behaviours observed in cloth. (Bridson, Marino, & Fedkiw, 2003) properly isolated the out-of-plane function of a mesh by measuring the specific divergence of adjoining points. (Etmuß, Groß, & Straßer, 2003) handled shear and bend conditions employing continuous concepts to approximate the finite difference.

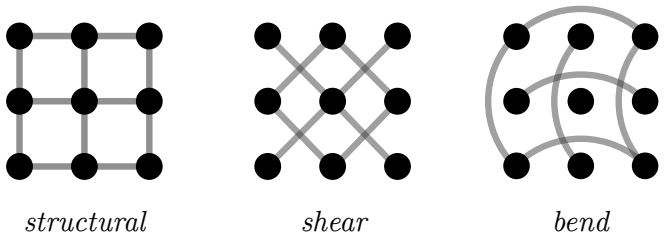


Fig. 3.4 Provot [1995] connectivity structure for mass-spring structure

3.3.1 Mass-Spring Physical Formulation

Following the definition in section 3.1.3, an expansion of the composition is necessary to accommodate for the two spatial dimensions of a mesh. Mass-points are said to have a position $\mathbf{x}_i$ and a velocity $\ddot{\mathbf{x}_i}$. An interconnecting network of springs S forms the structural links between masses. The spring forces acting on the adjacent particles of a spring are as follows (*Eqn. 3.7* & *Eqn. 3.8*)

$$\mathbf{f}_i = \mathbf{f}^s(\mathbf{x}_i, \mathbf{x}_j) = k_s \frac{\mathbf{x}_j - \mathbf{x}_i}{|\mathbf{x}_j - \mathbf{x}_i|}(|\mathbf{x}_j - \mathbf{x}_i| - l_0) \qquad \textit{Eqn. 3.7}$$

$$\mathbf{f}_j = \mathbf{f}^s(\mathbf{x}_i, \mathbf{x}_j) = -\mathbf{f}(\mathbf{x}_i, \mathbf{x}_j) = -\mathbf{f}_i \qquad \textit{Eqn. 3.8}$$

Where i and j are simply the index identifier for of the adjoining mass-points and l_0 is the rest length. The forces in the above formulation conserve momenta across the system ($\mathbf{f}_i + \mathbf{f}_j$

$= 0$) and reciprocate the elongation of spring constraints ($|\mathbf{x}_j - \mathbf{x}_i| - l_0$), proportionality may also be met by substituting for the continual k_s/l_0 in place of the constant k_s (*Eqn. 3.9* & *Eqn. 3.10*)

$$\mathbf{f}_i = \mathbf{f}^d(\mathbf{x}_i, \mathbf{x}_j, \dot{\mathbf{x}}_i, \dot{\mathbf{x}}_j) = k_d(\dot{\mathbf{x}}_j - \dot{\mathbf{x}}_i) \cdot \frac{\mathbf{x}_j - \mathbf{x}_i}{|\dot{\mathbf{x}}_j - \dot{\mathbf{x}}_i|} \qquad \textit{Eqn. 3.9}$$

$$\mathbf{f}_j = \mathbf{f}^d(\mathbf{x}_i, \mathbf{x}_j, \dot{\mathbf{x}}_i, \dot{\mathbf{x}}_j) = -\mathbf{f}_i \qquad \textit{Eqn. 3.10}$$

The damping forces and the projected velocity difference onto the spring are both proportional, they also conserve momentum. Unifying the left and right force terms gives the final formulation (*Eqn. 3.11*)

$$\mathbf{f}(\mathbf{x}_i, \mathbf{x}_j, \dot{\mathbf{x}}_i, \dot{\mathbf{x}}_j) = \mathbf{f}^s(\mathbf{x}_i, \mathbf{x}_j) + \mathbf{f}^d(\mathbf{x}_i, \mathbf{x}_j, \dot{\mathbf{x}}_i, \dot{\mathbf{x}}_j) \qquad \textit{Eqn. 3.11}$$

3.3.2 Constraints Enforcement

From having a valid formulation of the system's force dynamics, applying the formulation to an applicable particle network is a relatively straightforward task. In Fig. 3.5 presented is a crude illustration of how the system is intended to enforce a constraint once its new position has been calculated.

In certain conditions this current formulation is not sufficient in constraining the lengths of elongation between constraint pairs appropriately, indeed simply increasing the stiffness might seem sufficient for certain cases but does not solve the underlying problem. Specifically this problem refers to instances of cloth movement that aren't caused by analytic processes e.g points of collision/contact or points that are excluded from movements. For this presents the problem of calculating the force exerted to the system at the fixed points. Directly the function here is uncomputable using just the positions or velocities. However, a solution can be indirectly found by eliminating some confounding factors.

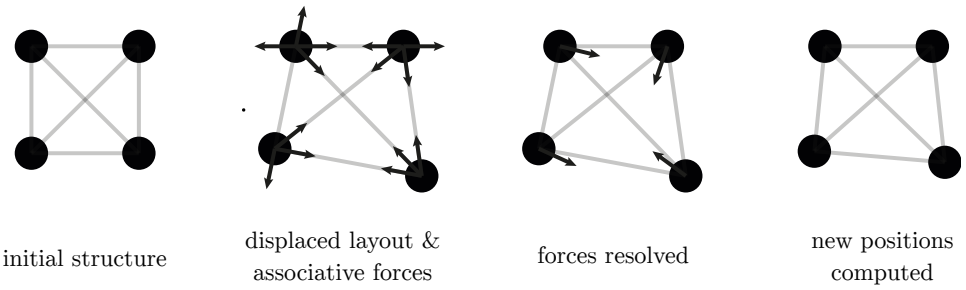


Fig. 3.5 Segment of a mass-spring system before and after constraint satisfaction

In the case outlined, (Provot, 1995) proposes that the solution is computable when properties of the fixed positions are analysed. We know the point of constraints displacement

is equal to \overrightarrow{0}, hence calculating the actual velocity and its resulting force (also equal to \overrightarrow{0}). The integration step of the simulation is for the most part ignored for the fixed point. For the result found, the displacement of the points are reset to the *a priori* known values.

In cloth simulation inverse dynamics procedures are also employed to deal with collisions of the cloth object with other objects, and to handle self-collisions. Such methods are discussed in depth in (Carignan *et al.*, 1991), in regards to this project, it was deemed outside of the scope of this project to implement the latter solution.

3.3.3 Collisions

When referring to the soft-body mechanics of cloth, the problem of collision handling is a decidedly involved task, indeed doing so whilst having any bearing with the physical accuracy that one might expect is a problem area in this field, to our knowledge, we were unable to uncover any papers that claimed to achieve this feet. For a given cloth model, all points are on the surface, and all these points have the potential of internally colliding with one another or the greater external environment for a given time step. This problem is furthered when factoring the amount of particles a cloth must be made of before a degree of realism is achieved. Generally in most applications that number is in excess of 10^4 particles. Interpenetrations of a thin cloth from the *wrong side* is a noticeable, visually aesthetically unpleasing occurrence that is difficult to correct once it has happened. A rigid-body simulation often has relatively few collisions that vary in strength from resting contact to high velocity impulse forces. Brute force methods of detection can usually grind the simulation. There are various papers that propose complicated models that claim to take on the different properties of a cloth, not least (Baraff, Witkin, & Kass, 2003), and (Bridson, Fedkiw, & Anderson, 2002), offering different solution to a self-collision tangling problem that plagued previous cloth systems done by others before it.

3.3.3.1 Cloth-Object

Ordinarily, this is relatively easy to deal with when compared to the latter collision problem. Predominantly there are two relatively primitive approaches; here objects are assumed to be static and have a known position.

3.3.3.2 Physically Accurate Model

This algorithm considers a cloth at time t and position $\mathbf{x}^t$, and a next computed position, $\mathbf{x}^{t+\Delta t}$. For every face that the cloth is comprised, consider its current and next position, if between these two positions ($\mathbf{x}^t$, $\mathbf{x}^{t+\Delta t}$) an intersection occurs with an external point/object, iterate times between the positions until an order of pre-defined accuracy is achieved to determine the time of contact between the two systems. In the case of faces previously

contacting one another but in the current state are no longer, disconnect these systems. When a face is identified as being in contact with a surface, it is marked, calculations must then be done to find the frictional forces from the normals of the contact surfaces. The simulation is then free to continue with the new step, and repeat the algorithm.

This method is hard to implement and also very slow due to the nature of having to make iterative time steps within the natural time step of the system. The system will at times be subject to instances where there will be numerous points of contact and thus a smaller step will be required to continue a simulation making the animation jittery at times. Some papers make claims to deal with friction forces correctly, (Bridson, Fedkiw, & Anderson, 2002), but most recognise the inefficiencies of this back tracking method and instead make a guess as to where the collision is between the initial and new positions.

3.3.3.3 Practical Model

The alternative to the above method is by no means without its flaws, however it is much simpler to implement and the impact on speed is reduced. Whereas the method described previously would account for face/edge collisions to an intermediated step accuracy, this approach is ignored in favour of a much more naïve technique; as such instances illustrated in the figure below will evade detection when using this algorithm.

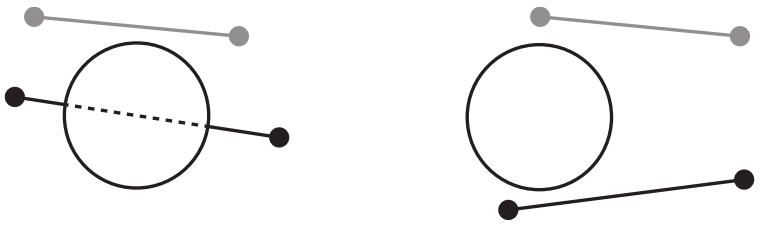


Fig. 3.6 Vertex positions before (grey), and their current (black)

Collisions using this model are dictated by individual vertex positioning information of the system. If a vertex is identified to be on the inside/wrong-side of an objects mesh an action is performed, either the vertex is returned to it's previous position or alternatively a repulsion force is acted on it.

3.3.3.4 Cloth-Cloth

Self intersections are more difficult to calculate than simpler external geometries. This is due to $O(n^2)$ pairs of structures that are able to collide with one another in a cloth with n nodes. The size of n, the number of particles the system is made of, is as discussed earlier, very

large and gets ever larger as the techniques evolve. This increases the demand for a fast and robust method for dealing with cloth-cloth collision handling.

Various researchers have approached this problem with varying degrees of success. (Baraff, Witkin, & Kass, 2003), explains that cloth-cloth collisions are detected by checking the particles and the edges of cloth sub-primitives. Their solution involved the use of strongly dampened springs to separate the cloth

Geometric self-collisions which occur in folding and contact situations are computationally expensive. To resolve this shortcoming, researchers recommend the use of active self-repulsions throughout intervals

3.3.3.4.1 Ray-Triangle Intersection

Cloth self-collisions can be abstractly resolved by a set of line/ray segments tested for intersection with triangles. Each vertex of cloth has a direction and velocity that can be tested with all of the triangles that make the structure.

If a triangle is defined as having three vertices ABC and the line segment is defined as the line between points P and Q. The ray PQ intersects ABC if the line penetrates the triangle ABC. Fig. 3.7 illustrates this mechanism.

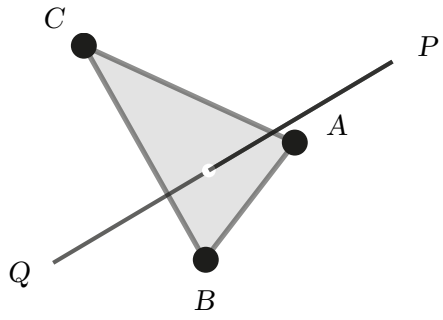


Fig. 3.7 3D Ray-Triangle intersection

Calculating whether ray-triangle penetration has occurred, we make use of the *scalar triple products.* In vector calculus the triple product is the product of three 3-dimensional Euclidean vectors, expressed by $(\mathbf{a} \times \mathbf{b}) \cdot \mathbf{c}$. Consider the following (*Eqn. 3.12*, *Eqn. 3.13*, & *Eqn. 3.14*)

$$u = [PQ, PC, PB] \qquad \textit{Eqn. 3.12}$$

$$v = [PQ, PA, PC] \qquad \textit{Eqn. 3.13}$$

$$w = [PQ, PB, PA] \qquad \textit{Eqn. 3.14}$$

If the line PQ has entered the triangle ABC (defined anti-clockwise) the line must pass to the left of the edges BC, CA and AB, and so expressions u \geq 0, v \geq 0 and w \geq 0 respectively must all be true, (Ericson, 2005).

4 Numerical Time Integration

Many researchers have applied various integration techniques, from the early work of (Terzopoulos *et al.*, 1987, 1988) employing a semi-implicit solver. Subsequent research focused on explicit methods, predominantly the Euler method and the higher order classical Runge-Kutta method. (Baraff & Witkin, 1998) proposed a modified implicit backward Euler scheme. This method had favourable stability properties over previous primitive solutions, and although it is only first order accurate and has a tendency in rare cases to diverge, it has provided significant improvements in cases where a large step size is advantageous (e.g. Real-Time graphics). In essence, implicit schemes have therefore become a recent area of interest in soft-body modelling. More recently, researchers have employed *implicit-explicit* (IMEX) techniques.

Several considerations must be accounted for when selecting an appropriate integration method, with the typical characteristics we seek, being stability and physical accuracy. The mathematical underpinnings governing the behaviour of a solution must be examined; a question such as whether the formulation conserves quantities or simply dampens to achieve convergence has very real consequences. Over the following section we give analysis of the explicit, symplectic, and implicit schemes in regards to the frameworks of cloth modelling.

4.1 Euler Scheme

Given a set of initial configurations of the cloth, along with external force inputs, a solution is required predicting the motion over time. In the case of a particle system, a semi-discretisation is defined for a space, and the solution is required to solve an initial value problem (IVP) by integrating a set of ordinary differential equations (ODEs). To simplify the problem, the set of ODEs are unified into a single super system, and the behaviour of the system is described by the ODE of the form (*Eqn. 4.1*)

$$M\ddot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \dot{\mathbf{x}}) \qquad \textit{Eqn. 4.1}$$

M here can be thought of as both a simple mass and a $3n \times 3n$ mass matrix (designated across its diagonal quadrant) defined for a cloth structure composed of n mass-points. Further, the force $\mathbf{f}$ and acceleration $\backslash\text{ddot}\{\mathbf{x}\}$ are vectors of size $3n$. Rearranging for acceleration to become the subject of the gives the Newtonian expression $\backslash\text{ddot}\{\mathbf{x}\} = \mathbf{f}/M$, where $\mathbf{x}$ here is a second derivative of the position with respect to time. From this, it is possible to make calculations relating to the force of the structure (Müller *et al.*, 2008). As our first advancement towards simulation, separating the second differential into two first order partial differentials is essential for a force based model like mass-spring system, they follows as (*Eqn. 4.2* & *Eqn. 4.3*)

$$\dot{\mathbf{v}} = \frac{\mathbf{f}(\mathbf{x}, \mathbf{v})}{M} \qquad \textit{Eqn. 4.2}$$

$$\dot{\mathbf{x}} = \mathbf{v} \qquad \textit{Eqn. 4.3}$$

An analytic interpretation of this follows as (*Eqn. 4.4* & *Eqn. 4.5*)

$$\mathbf{v}(t) = \mathbf{v}_0 + \int_{t_0}^{t} \frac{\mathbf{f}(t)}{M} dt \qquad \textit{Eqn. 4.4}$$

$$\mathbf{x}(t) = \mathbf{x}_0 + \int_{t_0}^{t} \mathbf{v}(t) dt \qquad \textit{Eqn. 4.5}$$

Here we see the sum of the integral changes up to time t, based on the initial values $\mathbf{v}(t_0)$ and $\mathbf{x}(t_0)$. Thus the simulation corresponds to simply calculating *Eqn. 4.4* & *Eqn. 4.5* from the initial condition t_0. Hence, the expressions *time integration* and *simulation* are frequently recycled synonymously. Using finite differencing we can numerically solve the equations to give approximations of the derivatives. (*Eqn. 4.6* & *Eqn. 4.7*)

$$\dot{\mathbf{v}}(t) = \frac{\mathbf{v}^{t+1} - \mathbf{v}^t}{\Delta t} + O(\Delta t^2) \qquad \textit{Eqn. 4.6}$$

$$\dot{\mathbf{x}}(t) = \frac{\mathbf{x}^{t+1} - \mathbf{x}^t}{\Delta t} + O(\Delta t^2) \qquad \textit{Eqn. 4.7}$$

Where Δt is the individual time step and t is the current time unit, the addendum of $O(\Delta t^2)$ simply notes the Big-O complexity of the formulation. Exchanging these approximates with the first order differentials (*Eqn. 4.6* & *Eqn. 4.7*) give the update rule below (*Eqn. 4.8* & *Eqn. 4.9*)

$$\mathbf{v}^{t+1} = \mathbf{v}^t + \frac{\mathbf{f}(\mathbf{x}^t, \mathbf{v}^t)\Delta t}{M} \qquad \textit{Eqn. 4.8}$$

$$\mathbf{x}^{t+1} = \mathbf{x}^t + \Delta t \mathbf{v}^t \qquad \textit{Eqn. 4.9}$$

This is known as the Forward Euler method. Quantities here from the current step are directly compounded into the values obtained in the next step, this is referred to as explicit integration. A simple modification to the scheme in order to make it more stable across intervals is done by evaluating $\mathbf{v}^{t+1}$ instead of $\mathbf{v}^t$ on the RHS of *Eqn. 4.9* by virtue of the new velocities availability at that point in time, this scheme is known as Forward-Backward Euler and is a semi-implicit (symplectic) solution. The implementation scheme of this is fairly straight forward; below is the algorithm that would be performed inside of the simulation loop.

1. **loop**:
2. **for all** particles i
3. $\mathbf{f}_i \longleftarrow \mathbf{f}^{gr} + (\mathbf{f}^{ex})_i + \sum \mathbf{f}(\mathbf{x}_i, \mathbf{x}_j, \mathbf{v}_i, \mathbf{v}_j)$

```
4.          endfor
5.          for all particles i
6.              v_i ⟵ v_i + Δt f_i/M_i
7.              x_i ⟵ x_i + Δt v_i
8.          endfor
9.      endloop
```

Here, $\mathbf{f}^{gr}$ is the gravitational force and $\mathbf{f}^{ex}$ is a composition of all other external forces due to, collisions, wind, user interfacing etc. Considered one of the simplest of numerical integration schemes, though it is not without its pitfalls. Stability for instance is dictated by the size of the time step and its relationship with the stiffness of the system, usually restricting the steps to very small values. More detailed stability analysis will follow in later sections. In a typical scenario a time step must be in the approximate range of 1×10^{-4} to 1×10^{-3}, meaning a considerable amount of steps are essential between two visualised intervals. The prime explanation for the instability in steps is that they obtain values *blindly*. The scheme makes the assumption that the function of force is linearly continuous over an interval. Let us examine a case whereby a spring is elongated slightly, the consequence of this action causes adjacent particles to move towards one another. In a situation where the time step is large, particles pass the equilibria of the initial configuration, the particle thus changes direction concurrently with the step. This becomes problematic as the initial force prior to the step is used through the entirety of the interval. The specific situation in-practice causes the particle to overshoot which invariably would lead to an eventual unrecoverable system explosion.

4.2 Runge-Kutta Scheme

To reduce the possibility of instabilities, some researches have opted to use precise integration schemes. Favourites among researcher's are the higher-order integrators, offering superior stability over more primitive schemes. These classes of integrators evaluate forces multiple times within a step to curtail the complications cited. A family of schemes that offer these higher-order accuracies are the Runge-Kutta solutions. Second order Runge-Kutta (RK2) replaces (*Eqn. 4.8* & *Eqn. 4.9*) with (*Eqn. 4.12* & *Eqn. 4.13*)

Eqn. 4.10
$$\mathbf{a}_1 = \mathbf{v}^t + \frac{\Delta t \mathbf{f}(\mathbf{x}^t, \mathbf{v}^t)}{2M}$$

$$\mathbf{a}_2 = \frac{\mathbf{f}(\mathbf{x}^t + \frac{\Delta t \mathbf{v}^t}{2}, \mathbf{v}^t + \frac{\Delta t \mathbf{f}(\mathbf{x}^t, \mathbf{v}^t)}{2M})}{M}$$
Eqn. 4.11

Eqn. 4.12
$$\mathbf{x}^{t+1} = \mathbf{x}^t + \Delta t \mathbf{a}_1$$

$$\mathbf{v}^{t+1} = \mathbf{v}^t + \Delta t \mathbf{a}_2$$
Eqn. 4.13

This shows how to get from the current intervals $\mathbf{x}_t$ and $\mathbf{v}_t$ to the next. A modification to the algorithm used in the Forward Euler formulation to fit this scheme is written below; again this would be implemented in the simulation loop when all the necessary initialisations have been considered.

```
1.      loop:
2.          for all particles i
```

3. $(\mathbf{v}^t)_i \longleftarrow \mathbf{v}_i$
4. $\mathbf{f}(\mathbf{x}^t, \mathbf{v}^t)_i / M_i \longleftarrow \left[\mathbf{f}^{gr} + (\mathbf{f}^{ex})_i + \sum \mathbf{f}(\mathbf{x}_i, \mathbf{x}_j, \mathbf{v}_i, \mathbf{v}_j)\right]/M_i$
5. **endfor**
6. **for all** particles i
7. $(\mathbf{a}_1)_i \longleftarrow \mathbf{v}_i + \Delta t \mathbf{f}_i / M_i$
8. $(\mathbf{a}_2)_i \longleftarrow \left[\mathbf{f}^{gr} + (\mathbf{f}^{ex})_i + \sum \mathbf{f}(\mathbf{x}_i + \Delta t \mathbf{v}_i/2, \mathbf{x}_j + \Delta t \mathbf{v}_j/2, \mathbf{v}_i + \Delta t \mathbf{f}_i/2M_i, \mathbf{v}_j + \Delta t \mathbf{f}_j/2M_j)\right]/M_i$
9. $\mathbf{x}_i \longleftarrow \mathbf{x}_i + \Delta t (\mathbf{a}_1)_i$
10. $\mathbf{v}_i \longleftarrow \mathbf{v}_i + \Delta t (\mathbf{a}_2)_i$
11. **endfor**
12. **endloop**

From the above, we can see that forces are evaluated twice for every time step, thus one step of RK2 takes as much time as two Forward Euler intervals. Nonetheless, in contrast to the standard Euler method, RK2 of course maintains second order precision. This in effect means that we now attain a fourth of the error present in the latter scheme. Accuracy can be further improved, there is an implementation of the Runge-Kutta scheme which is fourth order accurate (RK4) that is well suited for cloth simulation, and it is formulated below (*Eqn. 4.20* & *Eqn. 4.21*)

Eqn. 4.14
$$\mathbf{a}_1 = \mathbf{v}^t + \frac{\Delta t \mathbf{f}(\mathbf{x}^t, \mathbf{v}^t)}{2M}$$

Eqn. 4.15
$$\mathbf{a}_2 = \frac{\mathbf{f}(\mathbf{x}^t + \frac{\Delta t \mathbf{v}^t}{2}, \mathbf{v}^t + \frac{\Delta t \mathbf{f}(\mathbf{x}^t, \mathbf{v}^t)}{2M})}{M}$$

Eqn. 4.16
$$\mathbf{b}_1 = \mathbf{v}^t + \frac{\Delta t \mathbf{a}^2}{2}$$

Eqn. 4.17
$$\mathbf{b}_2 = \frac{\mathbf{f}(\mathbf{x}^t + \frac{\Delta t \mathbf{a}_1}{2}, \mathbf{v}^t + \frac{\Delta t \mathbf{a}_2}{2})}{M}$$

Eqn. 4.18
$$\mathbf{c}_1 = \mathbf{v}^t + \Delta t \mathbf{b}_2$$

Eqn. 4.19
$$\mathbf{c}_2 = \frac{\mathbf{f}(\mathbf{x}^t + \Delta t \mathbf{b}_1, \mathbf{v}^t + \Delta t \mathbf{b}_2)}{M}$$

$$\mathbf{x}^{t+1} = \mathbf{x}^t + \frac{\Delta t (\mathbf{v}^t + 2\mathbf{a}_1 + 2\mathbf{b}_1 + \mathbf{c}_1)}{6} \quad \textit{Eqn. 4.20}$$

$$\mathbf{v}^{t+1} = \mathbf{v}^t + \frac{\Delta t (\frac{\mathbf{f}(\mathbf{x}^t, \mathbf{v}^t)}{M} + 2\mathbf{a}_1 + 2\mathbf{b}_1 + \mathbf{c}_1)}{6} \quad \textit{Eqn. 4.21}$$

Among the most ubiquitous integration technique, the Runge-Kutta methods today is typically used in the numerical computational sciences. Looking closer at the formulation we see that the function of force is sampled four times. This halves the error, resulting in an overall reduction of 1/16. An algorithm implementation of RK4 simply be derived from the RK2 algorithm outlined perviously

4.3 Verlet Scheme

Higher order solutions attempt to eliminate instabilities by sampling forces multiple times across an interval. A different approach involves using quantities evaluated from the last step when approximating the derivatives. A collection of schemes are established on top of this concept. Made widely popular for cloth by (Jakobsen, 2001), Verlet integration typically was

used in fluid dynamics, today it is at the forefront of all things "real-time" in computer graphics. What makes it so attractive in respect to this work is its invariance under-time reversal and its ability to accurately conserve the energy of the system. Both are important properties of *true* Hamiltonian systems and it is thus important for the dynamics of our simulation to respect them in order to attain a reasonable level of realism.

In Verlet integration positions of the last time step, $\mathbf{x}^{t\,-\,\Delta t}$, are held in an additional state variable to produce a more accurate estimation. Two Taylor series representations of positions in opposing time directions yields the following (*Eqn. 4.22* & *Eqn. 4.23*)

$$\mathbf{x}(t-\Delta t) = \mathbf{x}(t) - \dot{\mathbf{x}}(t)\Delta t + \frac{\ddot{\mathbf{x}}(t)\Delta t^2}{2} - \frac{\dddot{\mathbf{x}}(t)\Delta t^3}{6} + O(\Delta t^4) \qquad \textit{Eqn. 4.22}$$

$$\mathbf{x}(t+\Delta t) = \mathbf{x}(t) + \dot{\mathbf{x}}(t)\Delta t + \frac{\ddot{\mathbf{x}}(t)\Delta t^2}{2} + \frac{\dddot{\mathbf{x}}(t)\Delta t^3}{6} + O(\Delta t^4) \qquad \textit{Eqn. 4.23}$$

Rearranging and appending appropriate terms of these two equations produces (*Eqn. 4.24*)

$$\mathbf{x}(t+\Delta t) = \mathbf{x}(t) + [\mathbf{x}(t) - \mathbf{x}(t-\Delta t)] + \frac{\mathbf{f}(t)\Delta t^2}{M} + O(\Delta t^4) \qquad \textit{Eqn. 4.24}$$

Now linear and cubic conditions can cancel each other forming a fourth order accurate scheme. In this method velocity is not strictly specified explicitly. Rather, the previous position of the last step remains in a buffer. However, we can expose the velocity, known as Verlet velocity and is defined as, $\mathbf{v}(t) = [\mathbf{x}(t) - \mathbf{x}(t-\Delta t)]/\Delta t$. Consolidating this knowledge into a frame interval notation gives the following (*Eqn. 4.25* & *Eqn. 4.26*)

$$\mathbf{x}^{t+1} = \mathbf{x}^t + \mathbf{v}^t\Delta t + \frac{\mathbf{f}(\mathbf{x}^t)\Delta t^2}{M} \qquad \textit{Eqn. 4.25}$$

$$\mathbf{v}^{t+1} = \frac{(\mathbf{x}^{t+1} - \mathbf{x}^t)}{\Delta t} \qquad \textit{Eqn. 4.26}$$

Interestingly this method of integration functions on just positional values, velocities here are only first order accurate and indeed not used. However, it is an alternative representation of the last position of a particle.

4.4 Implicit Scheme

Integration schemes examined thus far have only been *conditionally stable*, that is to say there exists a range for step values Δt, that the system is maintains numerically stability. Typically this region of stability is dictated by the accumulated stiffness of the system. Real-time applications such as in a computer game, call for integration schemes that are *unconditionally stable* across all conceivable situations. This is difficulty when in this instance a time step size is linked with the frame rate. Researchers have addressed this by

implementing the so-called *implicit* integration schemes. A scheme used mostly in numerical computations, though suited for computer graphics, is the implicit variant of Euler integration (Backward Euler). A slight modification to the Forward Euler (*Eqn. 4.8* & *Eqn. 4.9*) method is all that is needed to compose an implicit solution (*Eqn. 4.27* & *Eqn. 4.28*)

$$\mathbf{v}^{t+1} = \mathbf{v}^t + \frac{\Delta t \mathbf{f}(\mathbf{x}^{t+1})}{M} \qquad \textit{Eqn. 4.27}$$

$$\mathbf{x}^{t+1} = \mathbf{x}^t + \Delta t \mathbf{v}^{t+1} \qquad \textit{Eqn. 4.28}$$

The total amount of forces in the structure is singularly dependent on the positions. Frictional forces have the effect of stabilising the structure, so we instead ignore these quantities, electing to explicitly define them after the implicit solve step. An implicit solution induces significant amounts of numerical damping; in a common scenario there is no need to artificially add any.

The core change we see here are made at the point of evaluation of the the quantities projected to the next interval. In this case a direct manipulation is not possible instead we require an alternative approach. We derive two implicit formulations for a non-linearised system of equation. The solution to the system is such that they solve for the quantities at the next interval. An explicit scheme, as discussed previously, will arrive at new quantities 'blindly', whereas the implicit scheme here arrives at quantities that directly point back to the previous state, this also has the added benefit of conserving physical properties of the simulation.

By unifying the two single vectors (position and velocity) with the multi-dimensional force we obtain the first step towards solving for the system (*Eqn. 4.29, Eqn. 4.30,* & *Eqn. 4.31*)

$$\mathbf{x} = [\mathbf{x}_1^T, \ldots, \mathbf{x}_n^T]^T \qquad \textit{Eqn. 4.29}$$

$$\mathbf{v} = [\mathbf{v}_1^T, \ldots, \mathbf{v}_n^T]^T \qquad \textit{Eqn. 4.30}$$

$$\mathbf{f}(\mathbf{x}) = [\mathbf{f}_1(\mathbf{x}_1, \ldots, \mathbf{x}_n)^T, \ldots, \mathbf{f}_n(\mathbf{x}_1, \ldots, \mathbf{x}_n)^T]^T \qquad \textit{Eqn. 4.31}$$

A sparse $3n \times 3n$ mass matrix $\mathbf{M}$ is populated across its main diagonal composed of the masses symmetric to the particle-system, $[m_1, m_1, m_1, \ldots, m_n, m_n, m_n]$. this leads to the formulations (*Eqn. 4.32* & *Eqn. 4.33*)

$$\mathbf{M}\mathbf{v}^{t+1} = \mathbf{M}\mathbf{v}^t + \Delta t \mathbf{f}(\mathbf{x}^{t+1}) \qquad \textit{Eqn. 4.32}$$

$$\mathbf{x}^{t+1} = \mathbf{x} + \Delta \mathbf{v}^{t+1} \qquad \textit{Eqn. 4.33}$$

Substituting *Eqn. 4.33* into *Eqn. 4.32* gains the single system necessary for solving the next time steps velocity $\mathbf{v}^{t+1}$ (*Eqn. 4.34*)

$$\mathbf{M}\mathbf{v}^{t+1} = \mathbf{M}\mathbf{v}^{t} + \Delta t\mathbf{f}(\mathbf{x}^{t} + \Delta t\mathbf{v}^{t+1}) \qquad \textit{Eqn. 4.34}$$

The system here is essentially non-linear in the positions that we derive from it. An algorithm implementation that is used to efficiently solve this system is known as a Gradient Descent method, background is given in *Appendices* [**A**].

4.4.1 Implicit-Explicit (IMEX) Scheme

The two basic methods of numerical integration previously covered are of the classes implicit and explicit. The former are more stable though require the calculation of a large sparse matrix composed of both linear and nonlinear systems to be solved using a *preconditioned conjugate gradient* (PCG) method or similar. The latter are simple at the expense of reduced stability. The so called *implicit-explicit* (IMEX) scheme is a class of integration schemes that provide a way to solve stiff systems efficiently, providing the system meets certain requirements. Sometimes mistakenly confused with the similar symplectic or semi-implicit approaches, an IMEX integrator for all intents and purposes ignores axial rotational momenta of the Jacobian matrix. A more in depth definition is found in (Eberhardt, Etzmuß, & Hauth, 2000). We implement the scheme described in the works of (Desbrun, Shröder, & Barr, 1999). Their algorithm is provided in *Appendices* [**B**].

5 Design & Development

In this section we examine the development implementation of our model. We begin with looking at the initial experimentation done at the preliminary stage. We continue with detail of the actual implementation.

5.1 Development Environment

C++ has been the dominant programming language for computer graphics in both industry and research. It has for many reasons remained at the forefront of usage for the most leading developments in this area of computer graphics.

It's offers powerful general purpose programming features with performance in mind. Combining low-level features like direct memory manipulation with high-level features such as object-orientation, generics, and exceptions. These make C++ highly adaptable thus it was deemed it was the most logical choice for this project.

It was decided that this project would make use of the open standard for 3D computer graphics which of course is the OpenGL graphics framework. Other potential graphics APIs were considered, such as the exciting new industry low-level graphics frameworks, the cross industry standard Vulkan and the propriety offerings from Apple (Metal) and Microsoft (Direct3D 12). These new frameworks offer, in much the same vain of C/C++, an unmanaged framework for communicating with a GPU, this is done in part to address industry frustrations with the regulated nature of past frameworks. However these frameworks are highly specialised scarce little documentation thus it was determined that the more traditional API were more than capable to delivery the requirements of this project.

Third-party non-proprietary libraries used for this project:

- GLEW, v1.13.0, OpenGL Extension bindings library
- GLFW, v3.1.2, peripherals and OpenGL window context handling library
- GLM, v0.9.7.1, OpenGL C++ Mathematics library
- AntTweakBar, v1.12, light OpenGL GUI library

5.1.1 Hardware

Development and testing was done on a Apple MacBook Pro with the following specifications:

- Intel® Dual Core i5, 2.7 GHz, Broadwell™ Architecture
- Intel® Iris Graphics 6100, 845 GFLOPS @ 1.1GHz, Broadwell™ Architecture

5.2 System Design

As discussed in section 3.3, we decided that we would implement a forced based particle-mass-spring cloth derived from the model described in Provot [1995]. This model gives the necessary scope to implement various tried and tested research from time integration schemes to collision algorithms.

The initial prototype consisted of several classes, the most important of which are shown in the UML class diagram in Fig. 5.1. Each instance of the *Particle* class represents each individual particle in the simulation. As such, it holds all relevant data such as position, velocity, and mass, which are all needed to perform all necessary simulation calculations in the *Cloth* class.

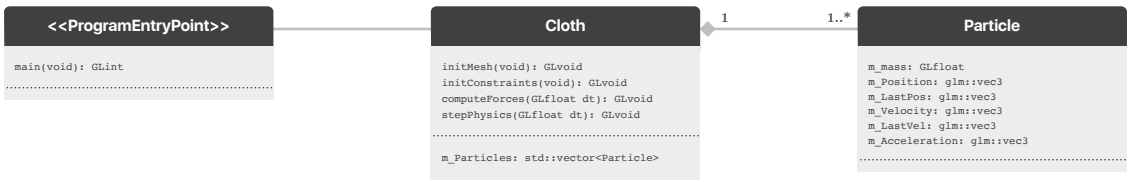


Fig. 5.1 Initial UML Class diagram

A basic schematic of the compute cycle is provided in the activity diagram shown in Fig. 5.2. After the initialisation of the window context and all necessary instantiations, the program steps into a user conditioned simulation loop capturing inputs from peripheral devices, which are used to interface with the cloths' properties. After all inputs are captured for the interval, the simulation is stepped forward by a user defined time step. As discussed in section 4, a time step must be small to ease instabilities and inaccuracies of the integration.

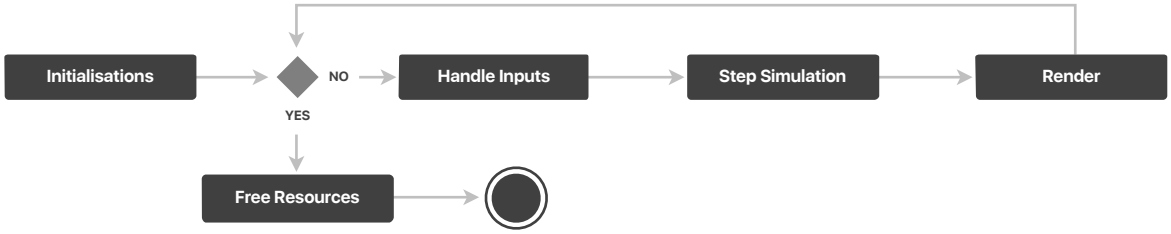


Fig. 5.2 Initial UML Activity diagram

From the above, the principal function of the activity of the program can be found in the *Step Simulation* process, composed of three sub-actions. Forces between all constraint pairs are calculated, section 3.3.1, position and velocities are then integrated by any one of the schemes previously examined in section 4. This is proceeded by a dynamic inverse constraints enforcement algorithm to realign particle positions according to the mass-spring method described in (Provot, 1995) and discussed in 3.3.2.

5.3 System Implementation

Before development could effectively begin, modern OpenGL (3.3+) foundational initialisation code must be written. A class was written to handle GLSL shader compilation with effective compilation exception and error handling. In addition, a render function that can pass in and manage Vertex Array Objects (VAOs) and Vertex Buffer Objects (VBOs) was implemented.

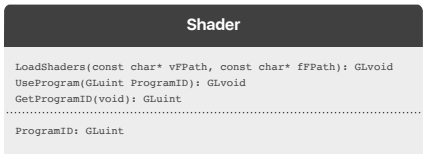


Fig. 5.3 Shader Class

Initial development began with investigations into the mass-spring system. Specifically, examinations were conducted on single segments of the model. A single spring connecting two mass-points and manipulating it along a one dimensional plane, according to the particle-system examined in section 3.1.3. For the integration portion of this simulation, we elected to use the Forward Euler scheme, section 4.1. From Fig. 5.1, a *Particle* class contains the properties of a mass-point, which gives us the ability to create a `std::vector` of mass-points in the *Cloth* class where an associative container of spring creates constraints between respective particles. After testing the system with various parameters for spring constants and rest lengths, satisfying results were produced, as can be seen in Fig. 5.4 below.

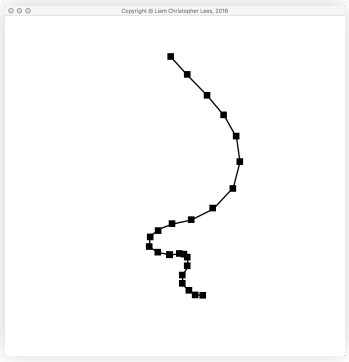
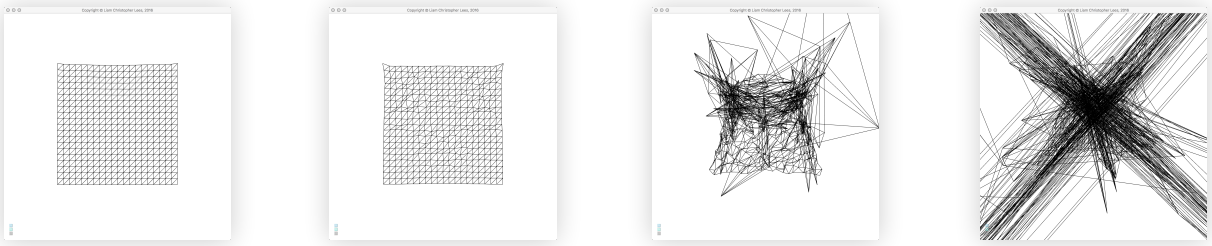


Fig. 5.4 Frame capture of initial string/1D particle system

With relatively respectable results observed, expanding the current implementation to a $m \times n$ two dimensional model required little work to the initial design; adding shear and bend spring handling were all that were needed. However, the system at this point was no longer producing the smooth stable simulations seen in the initial 1D design. Instead, this produced the instabilities described by some researchers as *blow up*, caused by the timestep size coupled by a relationship with the stiffness of the system. Mass-points in the system, after a period, began to oscillate erratically until the system was no longer indiscernible from the initial structure, Fig. 5.5 shows this phenomena occurring in our simulation.



50^{th} frame	100^{th} frame	150^{th} frame	200^{th} frame
Stable	Oscillating	Unstable	"Blow up"

Fig. 5.5 Frame captures of a Forward Euler scheme exploding

In later implementation we employ the various time integration methods that are described in section 4. For the time being reducing, the timestep size will suffice until the later stages of this development.

Further observations revealed that when the amount of point-masses were increased the implementation gave incorrect visuals. The system would become overstretched when draped at fixed points. Indeed this aligns with what (Provot, 1995) described as the *super-elastic* effect.

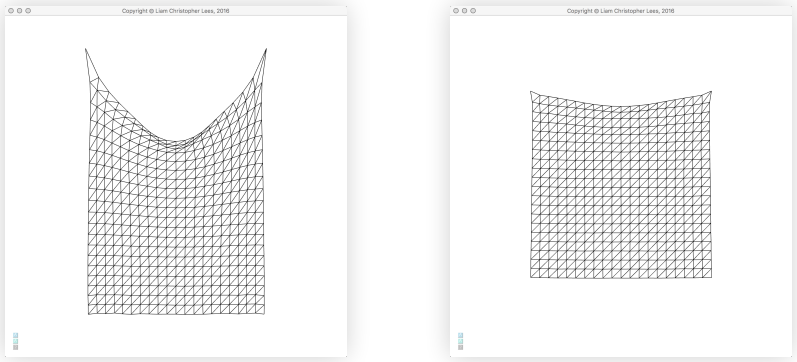


Fig. 5.6 Frame captures of the 'Super-Elastic' effect and the Inverse Dynamics solution

As described in section 3.3.2, a solution grounded in inverse dynamics was proposed. Testing

indicated that this somewhat solved the problem to a degree as can be seen in the comparison Fig. 5.6.

5.3.1 Integration

As is evident, the integration scheme used by the system ultimately governs the realism of the simulation. We aimed to implement into our model both the simplistic and complex schemes used today in research and industry for soft-body simulations.

In this sub-section we apply the technical analysis of time integration methods discussed in section 4 to a code implementation compatible with our model. The model thus far has used the basic Forward Euler scheme with limited success.

```
GLvoid Cloth::EXP_ForwardEuler(GLfloat dt) {

    for (auto p = particles.begin(); p != particles.end(); p++) {

        glm::vec3 V = p->getVelocity();
        glm::vec3 Vlast = p->getLastVelocity();
        glm::vec3 X = p->getPosition();
        glm::vec3 A = p->getAcceleration();

        V = V + (A * dt);
        X = X + (Vlast * dt);

        p->setNewVelocity(V);
        p->setNewPosition(X);
    }
}
```

Fig. 5.7 Forward Euler integration scheme (C++)

An elementary alteration to the current scheme can be made to improve its performance. It was contended in section 4.1 that by simply evaluating the new velocity when calculating the new position, the value is implicitly attained. This is the Forward-Backward integration method. Below is the implementation we used

```
GLvoid Cloth::SYM_ForwardBackwardEuler(GLfloat dt) {

    for (auto p = particles.begin(); p != particles.end(); p++) {

        glm::vec3 V = p->getVelocity();
        glm::vec3 X = p->getPosition();
        glm::vec3 A = p->getAcceleration();

        V = V + (A * dt);
        X = X + (V * dt);

        p->setNewVelocity(V);
        p->setNewPosition(X);
    }
}
```

Fig. 5.8 Symplectic Forward-Backward Euler integration scheme (C++)

We observed significant improvements over the previous scheme with these modifications. In some cases we achieved timestep sizes five times in excess of the derivative method.

5.3.1.1 Implicit Implementation

Dealing with the turbulent nature inherent to explicit methods and their variants is arguably the biggest frustration when seeking to implement an effective simulation system. Post

dampening of the system is a solution that has the effect of reducing energies across the timestep thus reducing feedback. However, this is an indirect solution. Consequences of this give the cloth different material properties, and generally we observe viscous like motion resembling a heavy rubber like material. Section 4.4 discusses the mathematics of an Implicit Euler scheme known as Backward Euler that solves derivatives at the next timestep using current state variables.

Fig. 5.9 UML Class diagram of relationship of *Cloth* with the *LargeVM* C++ template

To implement this scheme, modifications to the core code were required. The current program at present would not be able to handle the requirements of this scheme. It was concluded that a C++ template class would be needed (*LargeVM*) to handle mathematic computation of the large system of vectors and matrices of types `glm::vec3` and `glm::mat3x3` respectively for the various properties of the cloth, specifically; the system vectors position `xVec`, velocities `vVec`, force `fVec` and the system matrices, the mass Jacobian `mMat`, and the springs stiffness Jacobian `kMat`. In Fig. 5.10 presented is our implementation

```
GLvoid Cloth::IMP_BackwardEuler(GLfloat dt) {

    for (auto p = particles.begin(); p != particles.end(); p++) {

        xVec[p->getIndex()] = p->getPosition();
        fVec[p->getIndex()] = p->getForce();
        vVec[p->getIndex()] = p->getVelocity();
    }

    LargeVM<glm::mat3x3> A = mMat - (glm::pow(dt, 2) * kMat);
    LargeVM<glm::vec3> b = (mMat * V) + (dt * f);

    SolveCG(A, vVecNew, b);

    for (auto p = particles.begin(); p != particles.end(); p++) {

        GLsizei ind = p->getIndex();

        xVec[ind] += (dt * vVecNew[ind]);
        vVec[ind] = vVecNew[ind];

        p->setNewPosition(xVec[ind]);
        p->setNewVelocity(vVec[ind]);
    }
}
```

Fig. 5.10 Implicit Backward Euler integration scheme (C++)

As can be seen from the code above, the implementation is partially complete. The scheme contains a system of linear and non-linear problems that need to be solved and we solve this by using a conjugate gradient method to converge on to the solution, in this case the new velocity. (Baraff & Witkin, 1998) used a Pre-conditioned Conjugate Gradient (PCG) solver to ensure fast convergence in the model they proposed for their implicit problem. We elected to use the simpler solver without the prior conditioning.

The Conjugate Gradient (CG) method is the most prominent iterative method for solving

sparse systems of linear equations. CG is an effective solution for solving our partial differential of the form (*Eqn. 5.2*)

$$Ax = b \qquad \textit{Eqn. 5.1}$$

Where x here is the unknown vector, b is a known vector, and A is a known, square ($3n \times 3n$), symmetric, positive-definite matrix. Our implementation is presented below but a more in-depth discussion of the CG method is found in *Appendices* [**A**].

```
GLvoid Cloth::SolveCG(LargeVM<glm::mat3x3> A, LargeVM<glm::vec3>& x, LargeVM<glm::vec3> b) {
    // NOTE, operations on LargeVM objects are overloaded by LargeVM
    LargeVM<glm::vec3> r = b - (A * x);
    LargeVM<glm::vec3> d = r;
    LargeVM<glm::vec3> q;

    GLfloat alpha, beta;

    GLfloat delOld;
    GLfloat delNew = dot(r, r); // 'dot' here is not the same as glm::dot, but a function of LargeVM
    GLfloat del0 = dNew;

    GLfloat eTol = std::pow(1*10, -10); // Error tolerance
    GLsizei iterMax = 5; // Maximum iterations
    GLsizei i = 0;

    while ((delNew > (std::pow(eTol, 2) * del0) && (i < iterMax)) { // iterate until (Ax = b is satisfied) or (iterMax is met)

        q = A * d;

        alpha = delNew / dot(d, q);

        x = x + alpha * d;
        r = r - alpha * q;

        delOld = delNew;
        delNew = dot(r, r);
        beta = delNew / delOld;

        d = r + (beta * d);

        i++;
    }
}
```

Fig. 5.11 CG Solver for the Backward Euler scheme (C++)

As expected, the Backward Euler scheme gave significant performance improvements over previous scheme. Under certain conditioning we were able to achieve relatively large time steps until the system showed signs of instability (system oscillation). The instabilities here are a product of the system transforming into a non-linear system as $\Delta t^2 k_s$ approaches a system critical value, of course pre-conditioning of the solver would allow for even larger step-sizes, (Baraff & Witkin, 1998).

5.3.1.1 IMEX Implementation

Some researchers have, in modern times, opted to use implicit-explicit integration schemes, that offer the ease we associate with explicit schemes and the refined properties of implicit time integration. In our implementation we chose to use one of the first schemes proposed for cloth simulation. (Desbrun, Schröder, & Barr, 1999) in their work, they eloquently detail a algorithm which is both fast and efficient for animating mass-spring particle-systems. We examined this method in *Section* 4.4 and the algorithm of the work is provided in *Appendices* [**B**].

We begin by calculating a force filtering constant matrix for the cloths structure, we can later utilise when computing the force approximation.

```cpp
const glm::mat3x3 I = mat3(1); // Identity/Unit Matrix
const glm::mat3x3 H = mat3(-1, 1, 0,
                           1,-2, 1,
                           0, 1,-1); // Hessian Matrix

GLfloat mT = 0.0f; // Total Mass
glm::vec3 Xg = glm::vec3(0); // Centre of Gravity

for (auto p = particles.begin(); p != particles.end(); p++) {
    mT = mT + p->getMass();
    Xg = Xg + p->getPosition();
}

const glm::mat3x3 W = glm::inverse(I - H * (powf(dt, 2) / mT)); // Force Field Filter
Xg /= NumP_XY;
```

Fig. 5.12 Initialisation of the force filtering matrix (C++)

The matrix here $(I - \Delta t^2 H/M)$ has some rather desirable properties. First, H, a Hessian matrix abstraction of the cloths structure, is symmetric in nature and more importantly has a zero eigenvalue for the eigenvectors global translation $(1, ..., 1)^T$ of the system and thus will not have an impact on internal force dynamics. As a consequence, the inverse matrix W has a eigenvalue of one for the same translation. Multiplication of the W matrix with the cloths internal forces obtains a Diracian *filtering,* as directed by (Kass, 1995).

In the extreme case defined as $\Delta t^2 k_s < 1$ the filtering will no longer become Diracian, meaning for small values of stiffness and step size the implicit solution is equivalent to explicit integration.

The next phase in this scheme involves calculating the projected force.

```cpp
glm::vec3 dTor = glm::vec3(0); // Global Torque

// Prediction
for (auto p = particles.begin(); p != particles.end(); p++) {

    GLfloat m = p->getMass();

    glm::vec3 Fp = glm::vec3(0);
    glm::vec3 F = p->getForce();
    glm::vec3 X = p->getPosition();
    glm::vec3 V = p->getVelocity();

    Fp = Fp + F * W; // Predicted Force (Filtered)
    dTor = dTor + cross(Fp, X); // Global Torque Value

    V = V + ((((F + Fp) * dt) / m));
    X = X + (V * dt);

    p->setNewVelocity(V);
    p->setNewPosition(X);
}
```

Fig. 5.13 Force prediction (C++)

This prediction does not consider rotational effects and thus regrettably does not conserve the components momentum. These rotations are deliberately avoided for good reason, forces do not change in magnitude making accurate prediction of the angle unfeasible.

A simple post-correction on positions is necessary to align linear momenta and attempt to angular momenta to conserve. As stated, this algorithm does not preserve the former modes

quantity, we can however make this value more balanced. below in Fig. 5.13 shows this post-fix.

```
vec3 Fc; // Corrected Force

// Angular Momento & Linear Momento Post-Fix Correction
for (auto p = particles.begin(); p != particles.end(); p++) {

    GLfloat m = p->getMass();
    vec3 X = p->getPosition();

    Fc = cross(dTor, (Xg - X)) * dt; // Corrected Force
    X = X + Fc * (powf(dt, 2) / m);

    p->setNewPosition(X);
}
```

Fig. 5.14 Momentum correction (C++)

As can be seen, we accomplish this by taking the global torque (Fig. 5.14) and then crossing it with the difference between the centre of gravity and the evaluated position. This according to (Desbrun, Schröder, & Barr, 1999) *linearly* approximates the momentum with satisfactory results.

In testing we were quite disappointed at the results we garnered. We saw little if any performance improvements over the simpler symplectic scheme, making us question the authors claims to the viability of such a schemes, in our view, has far overreaching unnecessary complexities. However, in view of further testing, we feel it is prudent to draw such conclusions as the validity of the implementation is questionable and probably wrong.

5.3.1.1 Position-Based Implementation

Our *Position-Based* implementation required certain aspects of our forces based model to be modified to accommodate for this alternative approach to modelling. As explained previously in section 3.2.1, (Jakobsen, 2001) described an approach to cloth modelling omitting the need for a velocity evaluation layer, instead we are tasked with solving a single 2nd order differential equation.

Although strictly not used to the same extent of a force based model, we still do have to calculate a velocity component when we compute an elements force. Known as *Verlet Velocity* this step calculates the rate of change of the position over the current and previous step, Fig. 5.15 shows the function that calculates this

```
inline vec3& Cloth::verletVelocity(Particle& p, GLfloat dt) const {
    return((p.getPosition() - p.getLPosition()) / dt);
}
```

Fig. 5.15 Verlet Velocity calculator (C++)

We can then simply iteratively assign the new values explicitly across the system. seen in

Fig. 5.16, we initialise the velocity and the component dampening.

```
for (auto p = particles.begin(); p != particles.end(); p++) {

    if (m == VERLET)
        p->setVelocity(verletVelocity(*p, dt));

    //  Force Due To Dampening
    if (m == VERLET)
        p->addForce(GlobalDamping * verletVelocity(*p, dt)); // Position Based
    else
        p->addForce(GlobalDamping * p->getVelocity()); // Force Based
}
```

Fig. 5.16 Velocity initialisation step (C++)

After appropriate force calculations are completed. We are now able to proceed to the concluding integration of positions, as specified and formulated by (Jakobsen, 2001), we use the Verlet scheme, technical analysis is given in section 4.3.

```
Glvoid Cloth::EXP_Verlet(Glfloat dt) {

    for (auto p = particles.begin(); p != particles.end(); p++) {

        vec3 X = p->getPosition();
        vec3 Xlast = p->getLPosition();
        vec3 A = p->getAcceleration();

        X = X + (X - Xlast) + (A * glm::pow(dt, 2));

        p->setNewPosition(X);
    }
}
```

Fig. 5.17 Verlet integration scheme (C++)

Considering the simplicity of this scheme when compared to one of the more complex integration methods, one would justifiably have understandable prior dispositions to the performance of such a scheme. Nonetheless upon testing the solution we observed outstanding versatility from the solution, with significant performance improvements over the IMEX, symplectic and Backward-Euler schemes. We were capable of achieving time steps in excess of 0.3000 time units, however to a considerable degree a comparison may be immaterial.

Upon examining these two approaches we noted that the output simulations with variables controlled across models, the visual translations did not remain consistent for a force based model compared with a position based. In our observations, a force based simulation would more often than not take on the properties of a *light* ductile fabric, whereas (with the same parameters) a position based model would take on the appearance of a visually differential *heavier* rubber-like fabric. This doesn't necessarily prove one is superior to the other but rather cements the fact that the models are fundamental distinct.

5.3.2 Collision Detection & Response

Techniques for solving the collision dynamics of a particle model cloth have previously been tackled briefly in section 3.3.3. In this section we seek to apply the theory to our cloth

model. Constraints on time have limited the scope of what we could have potentially achieved, having said that whilst the implementation is not efficient, it is effective. We cover collisions with the topological surfaces, planar, spherical and self.

5.3.2.1 Planar Topologies

Collision detection with a plane is a relatively simple task. The plane in this instance is treated as infinite along the perpendicular to the y axis. When a cloth is detected to have a position equal to the y value of the plane a collision is detected. A frictional resistive force is then applied to the appropriate colliding particles.

5.3.2.2 Spherical Topologies

In a similar vain to planar collisions, the handling of spherical primitives are based on a radius length value and its relation to the positional Euclidean distances of mesh points, as opposed to a model expandable to more complicated topologies.

Fig. 5.18 Frame capture of a cloth draped over a sphere

5.3.2.3 Cloth-Cloth

As previously discussed, a self-collision algorithm is very difficult to implement effectively. It very quickly became apparent from looking at the data structures associated to the task that the problem would fall into the categorisation NP hard, that is to say the computational complexity exponentially increases as the number of nodes searches increase. This is a problem. In order to have more realistic simulations, we require high resolution dense meshes, however, as is evident this would only increase complexity thus having significant impact on the simulations performance.

In our implementation we use a simple method of position response on collision detection. The solution is not without its problems, it generally works correctly with low resolution

meshes with stiff bend constraints.

5.3.3 Normals

For various features to be implemented (Lighting, Wind, etc.) we require, for each vertex, an associative normal vector. Essentially at every interval, normal values need to be calculated due to the dynamic nature of cloths ever changing state. Below presents the formal formulation of a vertex normal calculating method (*Eqn. 5.1*)

$$N = (V_1 - V_2) \times (V_3 - V_1) \qquad \textit{Eqn. 5.1}$$

The N, for a specific point V_1 can be calculated from the positions of a triangle comprised of points V_1, V_2 and V_3. Fig. 5.19 illustrates the solution

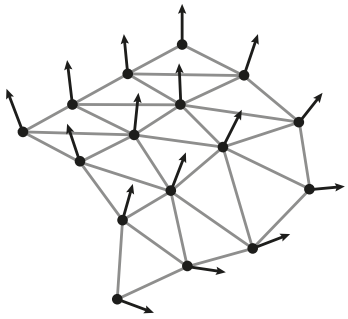


Fig. 5.19 Mesh vertex normals

The direct implementation of this was comparably as straightforward as the theory discussed prior. A method to calculate the new vertex normals of the system after force and integration computation had finished was contrived. First we implement a polygon normal calculator for the three component particles of the *Particle* class.

```
inline vec3 Cloth::getPolyNormal(Particle* pA, Particle* pB, Particle* pC) {
    return(cross((pB->getPosition() - pA->getPosition()), (pC->getPosition() - pA->getPosition())));
}
```

Fig. 5.20 Polygon normal calculation method (C++)

From this, we can calculate per-vertex normal information

```
for (auto f = faces.begin(); f != faces.end(); f++) {

    vec3 newNormal = getPolyNormal(f->particleA, f->particleB, f->particleC);

    f->particleA->addToNormal(newNormal);
    f->particleB->addToNormal(newNormal);
    f->particleC->addToNormal(newNormal);
}
```

Fig. 5.21 Vertex normal calculation method (C++)

We use a GLSL geometry shader to then visualise the normals across the cloths mesh,

producing what is seen in Fig. 5.22.

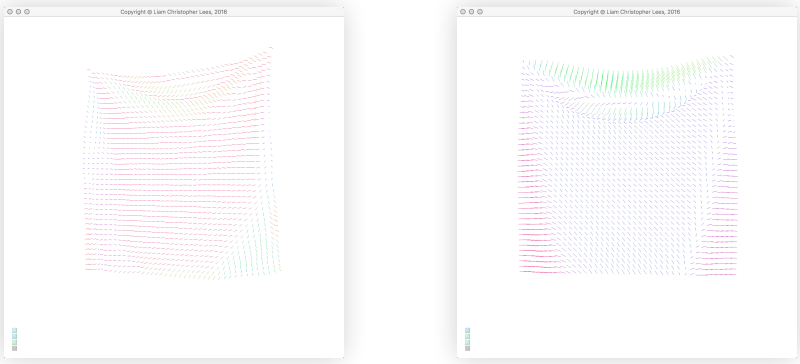


Fig. 5.22 Frame captures of our model with its vertex normals are dynamically mapped

5.3.4 Wind Aerodynamics

Adding wind to cloth expands the simulation to include the fluttering effects we see when a cloth is subject to an external force vector. The model we employ to implement this feature is not strictly accurate in the physical sense. The wind affects the cloth, but the cloth does not affect the wind; to do so would require a vast amount of fluid dynamic calculations, outside the scope of this project. However, we produce reasonable effects, resembling the oscillatory dynamics of winds affect on cloth.

We individually calculate a defined wind force across all of the triangle primitives that form the structure of the cloth.

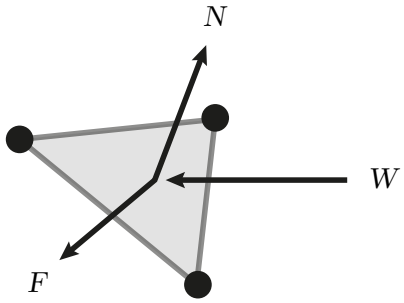


Fig. 5.23 Wind reaction with a mesh polygon segment

The force acting on the segment due to air molecules bouncing of it can always be assumed to be in the direction of the normal vector of the polygon. The force here is proportional to the surface area of the triangle, the angle at which the wind interacts with the triangle, and the velocity of the wind. Below outlines the formulation of the resultant force of wind on a segment. (*Eqn. 5.2*)

$$F = \hat{N}(N \times W) \qquad \textit{Eqn. 5.2}$$

Where F is the resultant force, N is the surface normal and W is the wind force. In the above when we calculate the normal vector, the length of the vector is proportional to the area of the triangle, which simplifies the implementation to a certain extent.

```
GLvoid Cloth::setWind(const glm::vec3 direction) {

    for (auto f = faces.begin(); f != faces.end(); f++) {

        glm::vec3 normal = getPolyNormal(f->particleA, f->particleB, f->particleC);
        glm::vec3 force = normal * glm::dot(glm::normalize(normal), direction);

        f->particleA->addForce(force);
        f->particleB->addForce(force);
        f->particleC->addForce(force);
    }
}
```

Fig. 5.24 Wind force calculation method (C++)

Fig. 5.X gives our C++ implementation for setting wind forces for each particle upon wind being enabled. In Fig. 5.25 we can see the visual results.

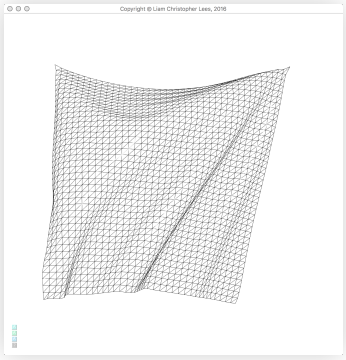


Fig. 5.26 Frame capture of a cross-wind across a cloth

The wind here behaved as expected, fluctuating and billowing with comparable characteristic that might be seen in a real-world scenario of the same conditions.

5.3.5 Shading/Lighting

To give the cloth more realistic visuals, appropriate lighting must be implemented. This is achieved using GPU shader programming. Using GLSL shaders we implement the classical Phong light shading scheme. There are three light components that make up a lighting model; ambient, diffuse and specular.

5.3.5.1 Ambient Lighting

Ambient lighting refers to light which comes from all directions and is controlled by global

light and material ambient terms. It's mathematical formulations follows as (*Eqn. 5.3*)

$$I_a = G_a M_a + L_a M_a \qquad \textit{Eqn. 5.3}$$

Where I_a is the ambient light reflected to the camera, G_a is the global ambient term, M_a is the ambient coefficient of the material and L_a is the colour of the ambient light.

5.3.5.2 Diffuse Lighting

Diffuse Lighting assumes the light is perceived by the viewer with the same intensity regardless of where the viewer is in relation to the object. In more technical terms, the surface is said to be isotropic. This is known as *Lambertian reflectance,* (Angel, 2006). The amount of light that is diffused is relational to the coefficient and angle between the normal of the surface and the incidence ray.

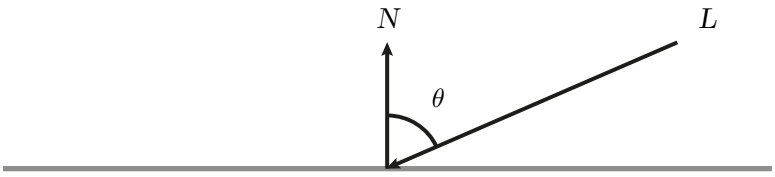


Fig. 5.27 Diffuse light reflection

The diffuse intensity formulation follows as (*Eqn. 5.4*)

$$I_0 = L_d M_d \cos(\theta) \qquad \textit{Eqn. 5.4}$$

Where I_0 is the reflected intensity, L_d is the colour of the diffuse light, M_d is the diffuse coefficient of the material and \theta is the angle between the ray and the normal of the surface.

5.3.5.3 Specular Lighting

Specular lighting is the mirror-like reflectance that light inhibits when illuminating some materials, such behaviour is described by the law of reflection. The Phong model for lighting states that the specular component of light that enters the eye should be proportional to the coplanar cosine between the reflectance ray and the vector to the eye. If the eye vector coincide with the reflection vector then the maximum specular intensity is received, (Phong, 1972).

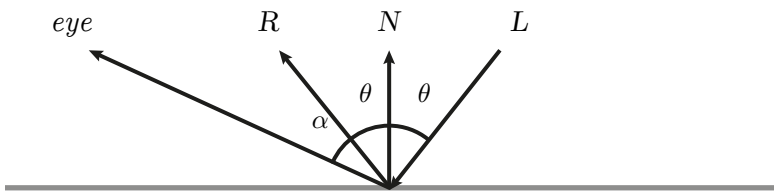


Fig. 5.28 Specular reflection

The reflection formulation follows as (*Eqn. 5.5*)

$$R = -2N(L \cdot N) + L \qquad \textit{Eqn. 5.5}$$

From the above, L is the component of the light source vector to the point, R follows as the direction of the reflected light, and the *eye* is simply the vector from the point to the eye. From this the specular component of the light is calculated as (*Eqn. 5.6*)

$$spec = (R \cdot eye)^s L_s M_s \qquad \textit{Eqn. 5.6}$$

Where s is the specular shininess value, L_s is the specular intensity, and M_s is the materials specular coefficient. As the lights source and eye diverges the specular intensity reduces proportionally. This reduction rate of the specular function is also controlled by a shininess factor. A higher shininess factor has the effect of a smaller bright spot as less light is entering the virtual eye from the surrounding vertices, or fragments.

```
vec3 phongModel(vec3 position, vec3 norm) {

    vec3 S = normalize(light.position - position);
    vec3 V = normalize(-position.xyz);
    vec3 R = reflect(-S, norm);

    vec3 ambient = light.ambientColour * material.ambientColour;

    float SdotN = max(dot(S, norm), 0.0);

    vec3 diffuse = light.diffuseColour * material.diffuseColour * SdotN;

    vec3 specular = vec3(0.0);

    if (SdotN > 0.0)
        specular = light.specularColour * material.specularColour * pow(max(dot(R, V), 0.0), material.specularShininess);

    return(ambient + diffuse + specular);
}
```

Fig. 5.29 Phong shading model (GLSL)

5.3.5.4 Per-Vertex Shading Vs Per-Pixel Shading

When light shading is calculated and applied, it can be implemented either per-vertex or per-pixel.

Per-Vertex lighting only computes the colour at each vertex. The colour of each poly is then filled by the standard graphics pipeline. This is done by interpolating the colours at the vertices known as the Gouraud smooth shading model. Per-Vertex lighting in most cases leads to noticeable visual artefacts, such as the distinct edges of polygons in a low poly mesh.

Per-Pixel lighting, also known as per-fragment lighting, is an illumination model applied to each pixel fragment. Normals at each fragments are calculated by interpolating the normals from the surrounding vertices (in GLSL this is achieved by use of the `smooth` qualifier). This results in much smoother looking reflections of specular and diffuse elements, which greatly increases the visual quality of a given object.

Fig. 5.30 compares our implementations of both a per-vertex Gouraud shader and per-pixel Phong shader scheme applied to a 50×50 node cloth structure. As can be seen, we also applied a cross-wind so we could more properly observe the occlusions.

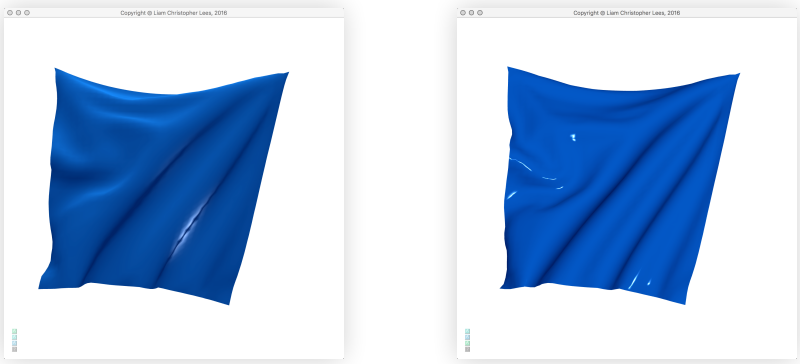


Fig. 5.30 Frame captures of Per-Vertex shading (left) and Per-Pixel shading (right)

From the above, we can observe that visible polygon artefacts are present in the Per-Vertex implementation, particularly at points of specular reflectivity. While not entirely perfect, we notice that the visual imperfection present in the latter are obscured to a greater degree.

5.4 User Interface Design

It was decided, due to practicality and the constraints on time, that a third party library would be used to provide the majority of the heavy lifting in respect to user intractable controlling. There are various libraries for OpenGL that meet such needs however the most popular among researchers seemed to be the open source C/C++ library *AntTweakBar* (ATB) created by Philippe Decaudin. We found, with a little work, that it could be easily integrated into our simulation and offered extensive feature set that were, in our opinion, the right balance between simplicity and complexity. ATB has currently ceased development as of 2009.

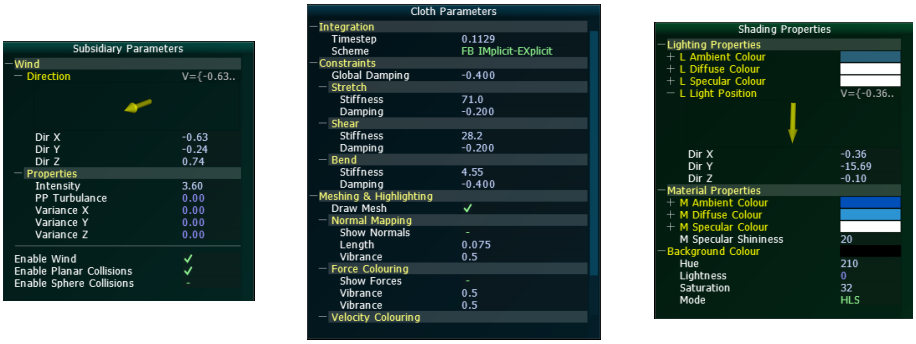


Fig. 5.31 Captures of AntTweakBar contexts

Linking it with the latest version of GLFW on our development system proved problematic and we experienced rendering errors that hindered development of the UI for a significant period until the issue was eventually resolved.

6 Evaluation & Review

We began this project with limited prior understandings of the mechanics of physically based simulations. A large portion of this undertaking was dedicated to the familiarising of the discrete mathematics that underpin the literature of research from the field. Various methods and approaches to the cloth problem were examined during the period and most of the modern techniques are discussed in the relevant previous sections. As is evident, we chose to use the mass-spring particle-system model proposed by Xavier Provot, which at the time of initial development seemed like the most solid grounding for a simulation with our requirements in mind.

Building on top of this model we have implemented seven integration schemes, some of which are used in leading industry software. Initial development prototypes employing the simple Forward Euler scheme with one dimensional particle systems yielded stable smooth results, however this did not scale to the two dimensional mesh complexities of cloth. Based on these shortcomings we elected to expand the model to include more complex schemes that could provide desirable features, such as larger step sizes and stiffer systems. The results were as expected, though it did throw into question the mass-spring systems shortcoming and thus underlying instability issues that such sophisticated integrators are required to simulate cloth *correctly*.

Also implemented were collision detection and response handling for basic static objects such as a planar surface and a spherical surface along with cloth self-collisions which employed a ray triangle intersection algorithm. Collision handling of basic shape primitives proved robust, however our self-collisions solution behaved contrary to the literature describing it, suggesting our implementation was defective in some way. Unfortunately time did not permit us to fully examine and diagnose this issue.

6.1 Results

In terms of the actual physical correctness of the simulation, little priority had been allocated. Thus results are all based on the visual and computational performance, as stated by others, if it is visually convincing it is more than likely to be close to a physically accurate simulation (House & Breen, 2000). The computer graphics industry normally use visual results as a metric to ensure quality, so visual and computational performance are the measurements that have been made to ensure a benchmark for quality.

6.1.1 Simulation Stability & Performance Discussion

The stability of our model from the onset of development was an important property that we

aimed to preserve as we increased the timestep or/and the stiffness of the system. An animation with real-time applications requires that the timestep variable is able to dynamically vary with the change of the frame rate. We tested the time-step performance of the seven integration schemes implemented. We drape a cloth by two fixed corner points whilst controlling various variables; mesh dimensions (50×50), stiffness (Stretch: 15.0, Shear: 10.0, Bend: 2.0), global velocity damping (-0.400), we increase the step size until signs of instability were observed. The recorded results are presented in Fig. 6.1 below

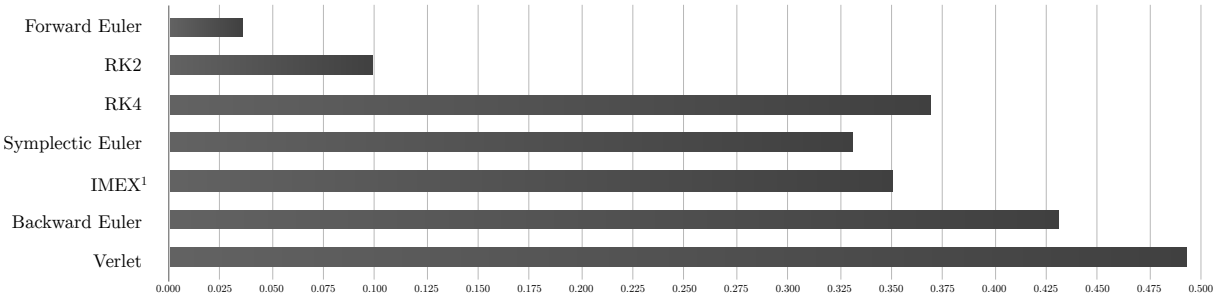


Fig. 6.1 Integration schemes stability performances (higher Δt unit is regarded as better)

From the results we can see that the (Jakobsen, 2001) positions based Verlet approach appears to perform best compared to its counter part schemes. Interestingly this scheme unlike Backward Euler, IMEX, RK4, and RK2 has a very low computational complexity comparable to that of the simplistic Forward and Symplectic Euler schemes, as is evident from Fig. 5.17 we see that it's time complexity is $O(n)$. This of course is a very attractive property in terms of the schemes viability for integration into a real time environment. The implicit schemes Backward Euler and IMEX trail behind.

Backward Euler performs best when the metric of ranking is confined to models solving for problems force based only. Its important to note that whilst computationally more complex than Verlet, in most cases this scheme will converge in just two iterations when the CG method (typically it took four iterations when we tested with the Steepest Descent variant) is employed. However, on approach to the critical point of instability, we saw the convergence time of the solution increase exponentially until the linear continuity of the system broke down thus leading to our primitive CG solver failing and system *blow up* occurring.

The Runge-Kutta results are interesting in that RK2 can be seen to be wholly unstable in most situations whereas RK4 appears to be exponent in nature to RK2 with significant

[1]Our IMEX implementation on further inspection is most likely flawed, at the point of writing this we were unable to identify the exact root of the problem. We have however decided to include the scheme into our results and analysis though we feel it is appropriate to declare our concerns of the accuracy of said results.

stability improvements. However, the RK4 algorithm is very slow to the point of unviable for any simulation thats purpose is not offline.

The semi-implicit schemes, Symplectic Euler and IMEX, whilst two very different approaches to integration both yielded similar stability results. The ratio of time complexity to stability when compared, is likely to be higher in the IMEX scheme even though it performed slightly better in the step size results. Thus determination of the superiority of either scheme is undeterminable at present.

6.1.2 Visuals

The visual results are good. A cloth-like behaviour that meets the objectives for the initial specifications have, in our opinion, been met if not surpassed. The simulation looks convincing under most conditioning as is evident from the capture we see below

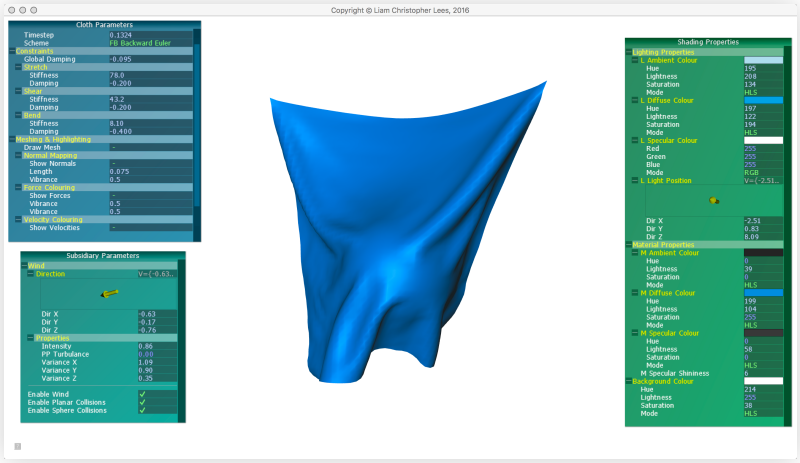


Fig. 6.2 Frame capture of the final deliverable (note, the cloth is draped over a sphere with a north westerly wind applied)

6 Conclusion

Despite the achievements made in this project, there are many avenues for exploration and examination that could be done to extend the functionality of the software. Not least the fixes of the current functionality or lack of. Frustratingly we failed to have a properly working implementation of the following:

- Self-Collisions
- Baraff & Witkin implicit integration model
- Position Based Dynamics

The self-collisions ray-intersection history based approach we employed had a tendency in most cases to become victim to *wrong-side* entanglement, a problem that plagued early works of the field. A more modern approach proposed in (Baraff, Witkin, & Kass, 2003) is said to resolve such instances of tangling. Described is a model free of history based interpretation, an algorithm performs a so called post Global Intersection Analysis (GIA) of the cloths mesh. This algorithm has proven its robustness and is used extensively in offline production animations.

The implicit method for resolving large step size as presented in (Baraff & Witkin, 1998) was attempted though we were unable to accurately calculate the Jacobian force derivative that was required in the preconditioning of the CG solver. This of course was regrettable, though would definitely be an optional route for further development. However, during the course of this development project the shortcoming experienced with the mass-spring force based model were evident by the complex mathematical formulations required to accommodate for the fundamental energy instabilities inherent to the model. PBD shows much promise, we integrated a semi discretisation of PBD, implementing position based Verlet into our code base which saw much improvements as is discussed previously. Of course this is just an element of the of the PBD approach outlined in (Müller *et al.* 2007). Thus we see a complete implementation as the most promising direction for future development.

Appendix [A]

Conjugate Gradient (CG) Method

The method of conjugate gradients was developed independently in 1952 by Magnus Hestenes & Eduard Stiefel. It is an iterative algorithm composed of a composite set of simple ideas that are used in the numerical solutions of sparse systems of linear equations, namely those whose matrix is real, symmetric ($A^T = A$) and said to be positive-definite ($x^T Ax > 0$ for all non-zero vectors x). We use this method to solve the large sparse system present in our Backward-Euler implicit integrator that otherwise would be too large to be handled by a more direct implementation (i.e Cholesky Decomposition).

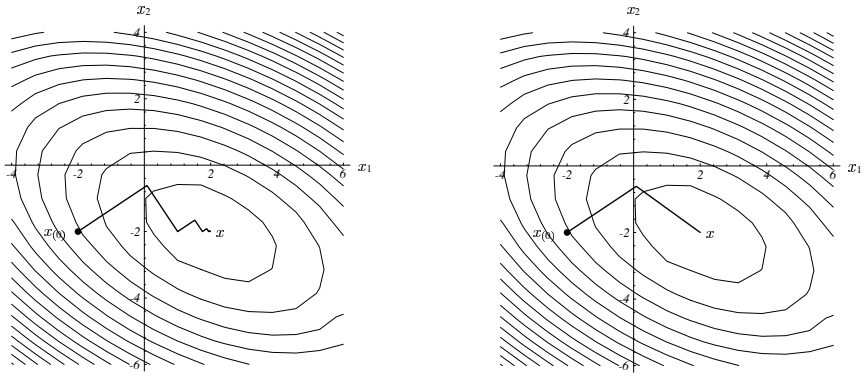


Fig. B.1 Method of Steepest Descent (left) and the Conjugate Gradient method (right) (Shewchuk, 1994)

From the definition *Eqn. 5.1*, we attempt to find the minimum of the equivalent quadratic of the form which is expressed below (*Eqn. B.1*)

$$f(x) = \frac{1}{2}x^T Ax - b^T x \qquad \textit{Eqn. B.1}$$

Where the derivative follows as (*Eqn. B.2*)

$$\frac{\mathrm{d}f}{\mathrm{d}x} = Ax - b \qquad \textit{Eqn. B.2}$$

Thus finding the minimum of $f(x)$ using the derivative obtains the solution to $\mathrm{A}x = b$. Specifically, the solution here could be done by use of the method of Steepest Descent (SD), which takes steps in the negative gradient direction. We begin by taking an initial guess of the value of x, then test for the gradient at that point, iterating the process until x is met. For each successive line search iteration the step size is minimised, below outlines the schematic of such iterative scheme (*Eqn. B.3, Eqn. B.4, & Eqn. B.5*)

$$r_i = b - Ax_i \qquad \textit{Eqn. B.3}$$

$$\alpha_i = \frac{r_i^T r_i}{r_i^T A r_i} \qquad \textit{Eqn. B.4}$$

$$x_{i+1} = x + \alpha_i r_i \qquad \textit{Eqn. B.5}$$

However this can be quite computationally taxing, the CG method improves the SD method by avoiding repetitious steps, Fig. D.1 illustrates this. Steps now are made in directions which are A-orthogonal or conjugate (when $d_i{}^T A d_i = 0$). The result of this method is that all new residuals are orthogonal for all previous residuals, we outline the basic conjugate method algorithm applicable to our needs and the SD method (right) for reference, it is expressed below

1. $i \longleftarrow 0$
2. $r \longleftarrow b - Ax$
3. $d \longleftarrow r$
4. $\delta^{new} \longleftarrow r^T r$
5. $\delta_0 \longleftarrow \delta^{new}$
6. **while** $i < i^{max} \wedge \delta^{new} > \varepsilon^2 \delta_0$ **do**
7. $\quad q \longleftarrow Ad$
8. $\quad \alpha \longleftarrow \delta^{new} / d^T q$
9. $\quad x \longleftarrow x + \alpha d$
10. $\quad r \longleftarrow r - \alpha d$
11. $\quad \delta^{old} \longleftarrow \delta^{new}$
12. $\quad \delta^{new} \longleftarrow r^T r$
13. $\quad \beta \longleftarrow \delta^{new} / \delta^{old}$
14. $\quad d \longleftarrow r + \beta d$
15. $\quad i \longleftarrow i + 1$

1. $i \longleftarrow 0$
2. $r \longleftarrow b - Ax$
3. $\delta \longleftarrow r^T r$
4. $\delta_0 \longleftarrow \delta$
5. **while** $i < i^{max} \wedge \delta^{new} > \varepsilon^2 \delta_0$ **do**
6. $\quad q \longleftarrow Ar$
7. $\quad \alpha \longleftarrow \delta / r^T q$
8. $\quad x \longleftarrow x + \alpha r$
9. $\quad r \longleftarrow r - \alpha q$
10. $\quad \delta \longleftarrow r^T r$
11. $\quad i \longleftarrow i + 1$

For the inputs A, b, a starting value of x, a maximum number of iterations i^{max}, and an error tolerance $\varepsilon < 1$.

Appendix [B]

Desbrun *et al.* Implicit-Explicit Integration Scheme

Taken from Figure 6 in the paper *Interactive animation of structured deformable objects*, contained below is their algorithm implementation (modified to conform to the notations and conventions we use), these calculations of course would take place in the simulation phase of the animation.

1. // Initialisation
2. **precompute** $\mathrm{W} = (\mathrm{I_n} - (\Delta t^2/M)H)^{-1}$
3. // Simulation loop
4. **loop**:
5. $\quad \mathbf{x}_G \longleftarrow 0$
6. $\quad$ // Compute internal forces $\mathbf{f}(\mathbf{x}^t, \mathbf{v}^t)_i$
7. $\quad$ **for all** particles i
8. $\quad\quad \mathbf{f}(\mathbf{x}^t, \mathbf{v}^t)_i \longleftarrow 0$
9. $\quad\quad \mathbf{x}_G \longleftarrow \mathbf{x}_G + \mathbf{x}_i$
10. $\quad\quad$ **for all** particles j linked by spring (i, j)
11. $\quad\quad\quad \mathbf{f}(\mathbf{x}^t, \mathbf{v}^t)_i \longleftarrow \mathbf{f}(\mathbf{x}^t, \mathbf{v}^t)_i + k_{ij}(|\mathbf{x}_i - \mathbf{x}_j| - ((l_0)^{ij})((\mathbf{x}_i - \mathbf{x}_j)/|\mathbf{x}_i - \mathbf{x}_j|)$
12. $\quad\quad\quad \mathbf{f}(\mathbf{x}^t, \mathbf{v}^t)_i \longleftarrow \mathbf{f}(\mathbf{x}^t, \mathbf{v}^t)_i + \Delta t k_{ij}(\mathbf{v}_j - \mathbf{v}_i)$
13. $\quad\quad$ **endfor**
14. $\quad$ **endfor**
15. $\quad \mathbf{x}_G \longleftarrow \mathbf{x}_G \,/\, N$
16. $\quad \delta T \longleftarrow 0$
17. $\quad$ // Integrate the approximation
18. $\quad$ **for all** particles i
19. $\quad\quad (\mathbf{f}^{filtered})_i \longleftarrow \left[\, \sum \mathbf{f}(\mathbf{x}_j, \mathbf{v}_j) W_{ij} \right]_j$
20. $\quad\quad \delta T \longleftarrow \delta T + \left[(\mathbf{f}^{filtered})_i \wedge \mathbf{x}_i \right]$
21. $\quad\quad (\mathbf{v}_i)^{t+\Delta t} \longleftarrow (\mathbf{v}_i)^t + \Delta t \left[\mathbf{f}^{gr} + (\mathbf{f}^{ex})_i + (\mathbf{f}^{filtered})_i \right] / M_i$
22. $\quad\quad (\mathbf{x}^{new})_i \longleftarrow \mathbf{x}_i + \Delta t (\mathbf{v}_i)^{t+\Delta t}$
23. $\quad$ **endfor**
24. $\quad$ // Post correction of angular momentum
25. $\quad$ **for all** particles i
26. $\quad\quad (\mathbf{f}^{corrected})_i \longleftarrow (\mathbf{x}_G - \mathbf{x}_i) \wedge \delta T \Delta t$
27. $\quad\quad (\mathbf{x}^{new})_i \longleftarrow (\mathbf{x}^{new})_i + \Delta t^2 (\mathbf{f}^{filtered})_i / M_i$
28. $\quad$ **endfor**
29. $\quad$ // Update positions and velocities
30. $\quad$ **for all** particles i
31. $\quad\quad (\mathbf{v}_i)^{t+\Delta t} \longleftarrow ((\mathbf{x}_i)^{t+\Delta t} - (\mathbf{x}_i)^t) / \Delta t$
32. $\quad\quad (\mathbf{x}_i) \longleftarrow (\mathbf{x}^{new})_i$
33. $\quad$ **endfor**
34. **endloop**

References

Zhao, Y., Wong, T., Tanks, S. and Chen, W. (1997). A model for simulating flexible surfaces of cloth objects. Computers & Structures, 63(1), pp.133-147.

Ascough, J., Bez, H. and Bricis, A. (1996). A simple finite element model for cloth drape simulation. International Journal of Clothing Science and Technology, 8(3), pp.59-74.

Tan, S., Wong, T., Zhao, Y. and Chen, W. (1999). A constrained finite element method for modelling cloth deformation. The Visual Computer, 15(2), pp.90-99.

Ng, H. and Grimsdale, R. (1996). Computer graphics techniques for modelling cloth. IEEE Comput. Grap. Appl., 16(5), pp.28-41.

House, D. and Breen, D. (2000). Cloth modelling and animation. Natick, Mass.: A K Peters.

Volino, P. and Magnenat-Thalmann, N. (2000). Virtual clothing. Berlin: Springer.

Weil, J. (1986). The synthesis of cloth objects. ACM SIGGRAPH Computer Graphics, 20(4), pp.49-54.

Terzopoulos, D., Platt, J., Barr, A. and Fleischer, K. (1987). Elastically deformable models. ACM SIGGRAPH Computer Graphics, 21(4), pp.205-214.

Terzopoulos, D. and Fleischer, K. (1988). Deformable models. The Visual Computer, 4(6), pp.306-331.

Breen, D., House, D. and Wozny, M. (1994). A Particle-Based Model for Simulating the Draping Behaviour of Woven Cloth. Textile Research Journal, 64(11), pp.663-685.

Kawabata, S. (1980). The standardisation and analysis of hand evaluation. The Textile Machinery Society of Japan.

Kass, M., (1997). An introduction to physically based modelling: an introduction to continuum dynamics for computer graphics. Computer Graphics Tutorial, ACM SIGGRAPH, pp.60-66.

House, D., DeVaul, R. and Breen, D. (1996). Towards simulating cloth dynamics using interacting particles. International Journal of Clothing Science and Technology, 8(3), pp.75-94.

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

Provot, X., (1997). Collision and self-collision handling in cloth model dedicated to design garments (pp. 177-189). Springer Vienna.

Baraff, D. and Witkin, A. (1998). Large steps in cloth simulation. Proceedings of the 25th annual conference on Computer graphics and interactive techniques - SIGGRAPH '98.

Baraff, D., Witkin, A. and Kass, M. (2003). Untangling cloth. TOG, 22(3), p.862.

Bridson, R., Fedkiw, R. and Anderson, J. (2002). Robust treatment of collisions, contact and friction for cloth animation. TOG, 21(3).

Hutchinson, J. (1989). Applied Elasticity: Matrix and Tensor Analysis of Elastic Continua (J. D. Renton). SIAM Rev., 31(1), pp.141-142.

Etzmuß, O., Groß, J. and Straßer, W. (2003). Deriving a particle system from continuum mechanics for the animation of deformable objects. IEEE Trans. Visual. Comput. Graphics, 9(4), pp.538-550.

Müller, M., Heidelberger, B., Hennix, M. and Ratcliff, J. (2007). Position based dynamics. Journal of Visual Communication and Image Representation, 18(2), pp.109-118.

Müller, M., Stam, J., James, D. and Thürey, N., (2008), August. Real time physics: class notes. In ACM SIGGRAPH 2008 classes (p. 88). ACM.

Eberhardt, B., Weber, A. and Straßer, W. (1996). A fast, flexible, particle-system model for cloth draping. IEEE Comput. Grap. Appl., 16(5), pp.52-59.

Carignan, M., Yang, Y., Thalmann, N. and Thalmann, D. (1992). Dressing animated synthetic actors with complex deformable clothes. ACM SIGGRAPH Computer Graphics, 26(2), pp.99-104.

Eberhardt, B., Etzmuß, O. and Hauth, M. (2000). Implicit-Explicit Schemes for Fast Animation with Particle Systems. Eurographics, pp.137-151.

Shewchuk, J. (1994). An Introduction to the Conjugate Gradient Method Without the Agonising Pain. Technical Report. Carnegie Mellon Univ.

Jakobsen, T., (2001). Advanced character physics. In Game Developers Conference (pp. 383-401).
[online] Available at: cs.cmu.edu/afs/cs/academic/class/15462-s13/www/lec_slides/Jakobsen.pdf [accessed 1 Feb. 2016]

Desbrun, M., Schröder, P., and Barr, A. (1999) Interactive animation of structured deformable objects. In Proceedings of the 1999 conference on Graphics interface '99, pp.1-8.

Ericson, C. (2005). Real-time collision detection. Amsterdam: Elsevier.

Phong, B.T., (1975). Illumination for computer generated pictures. Communications of the ACM, 18(6), pp. 311-317.

Feynman, C.R., (1986). Modeling the appearance of cloth (Doctoral dissertation, Massachusetts Institute of Technology).

Angel, E. (2006). Interactive computer graphics. Boston: Pearson/Addison-Wesley.