Real-Time Simulation and Visualisation of Cloth using Edge-based Adaptive Meshes.

Timothy John Richard Simnett

A thesis submitted for the degree of
Doctor of Philosophy
at the University of East Anglia
School of Computing Sciences
September 2012

Real-Time Simulation and Visualisation of Cloth using Edge-based Adaptive Meshes.

Timothy John Richard Simnett

Abstract

Real-time rendering and the animation of realistic virtual environments and characters has progressed at a great pace, following advances in computer graphics hardware in the last decade. The role of cloth simulation is becoming ever more important in the quest to improve the realism of virtual environments.

The real-time simulation of cloth and clothing is important for many applications such as virtual reality, crowd simulation, games and software for online clothes shopping. A large number of polygons are necessary to depict the highly flexible nature of cloth with wrinkling and frequent changes in its curvature. In combination with the physical calculations which model the deformations, the effort required to simulate cloth in detail is very computationally expensive resulting in much difficulty for its realistic simulation at interactive frame rates. Real-time cloth simulations can lack quality and realism compared to their offline counterparts, since coarse meshes must often be employed for performance reasons.

The focus of this thesis is to develop techniques to allow the real-time simulation of realistic cloth and clothing. Adaptive meshes have previously been developed to act as a bridge between low and high polygon meshes, aiming to adaptively exploit variations in the shape of the cloth. The mesh complexity is dynamically increased or refined to balance quality against computational cost during a simulation. A limitation of many approaches is they do not often consider the decimation or coarsening of previously refined areas, or otherwise are not fast enough for real-time applications.

A novel edge-based adaptive mesh is developed for the fast incremental refinement and coarsening of a triangular mesh. A mass-spring network is integrated into the mesh permitting the real-time adaptive simulation of cloth, and techniques are developed for the simulation of clothing on an animated character.

Acknowledgements

I would like to express my gratitude to my supervisors Prof. Andy Day and Dr. Stephen Laycock for their help and support during my studies. I wish also to thank Dr. Robert Laycock for his support and especially for discussions and assistance while working on my second publication.

Finally, I would like to thank my family for their continuing support and encouragement during my time at university.

Table of Contents

List of Tables

List of Figures

Listings

Chapter 1

Introduction

Cloth simulation has had an increasing role in computer graphics in the last decade, realistic characters require realistic clothing after-all. There are many methods and techniques that have been developed to simulate deformable cloth, each with their own strengths and weaknesses. Impressive realism of simulated cloth has already been achieved, but at understandably high computational cost associated with the use of dense meshes. The highly flexible nature of cloth can be challenging; complex collisions with objects in the virtual environment must be detected and resolved efficiently otherwise they become a significant bottleneck to performance. Offline simulation may be performed where the final result is much more important than the time spent computing it, such as the use in animated films where offline rendering is also performed. However, real-time applications are just as important as their offline counterparts including real-time virtual environments, crowd simulation, games and online clothes shopping. The cloth unfortunately can suffer from unrealistic behaviour and visual quality due to the simplifications made in order to achieve interactive frame rates. Also it is not possible to repeat or manually tweak problematic frames as can be done with offline work, in-order to be most useful, real-time simulations must run adequately without constant user intervention. In this research we consider an adaptive approach for the real-time simulation of cloth and virtual clothing on

characters. At the heart of the approach, the mesh density is dynamically changed to balance detail against computational cost.

In this thesis we present a novel edge-based approach for the fast incremental adaptive refinement and coarsening of a mesh with sufficient performance for its real-time use. Our adaptive mesh is designed with flexible support for both refinement and coarsening criteria that trigger the splitting and re-joining of edges, such as the commonly used curvature-based criteria but we also demonstrate the use of collision, edge length, and back-face coarsening for clothing. We integrate a mass-spring network improved with material coordinates and seaming to allow adaptive clothing to be created and worn on virtual animated characters.

1.1 Background

Much research in computer graphics is focussed on the modelling and rendering of real world phenomena and continuing advancements enable ever increasingly detailed virtual environments. Human aspects are an important part of realism in these environments, not only including modelling and animation of virtual characters but also the simulation and rendering of their clothing. In the past virtual clothing for real-time applications was part of character animation, garments were commonly baked into the character's mesh and texture. As hardware has improved, clothes could be then constructed separately with their own mesh and textures such that a character may change its outfit by swapping between garment meshes in real-time. This is exploited in many games, where skeletally animated characters may be dressed with clothes or fitted with various armour. However, this couples the clothing tightly with the underlying character's mesh, so it is only generally suitable for tightly fitting garments which deform directly with the skin. Free moving deformable clothing is much more challenging and must be physically simulated or otherwise evolved over time.

Cloth simulation is a large field, relying on the combined use of many areas including applied mathematics, numerical analysis, computational geometry, physics, collision detection and response, 3D modelling, rendering and user interaction. It is not surprising that many different cloth simulation methods exist given the range of perspectives that researchers in these fields will have. In the real world, cloth behaviour is the result of complex interactions between billions of atoms, and between thousands of woven threads. It is not feasible to model cloth at such a level; remarkably offline simulations are able to capture great realism by only considering the macro properties of cloth that we see in our everyday lives. Often regarding it as a very thin, flexible and nearly inextensible surface, many different models have been constructed to approximate this behaviour.

1.2 Motivations and Research Objectives

The real-time simulation of cloth is difficult; real-time applications suffer from limited processing resources with a requirement for a high update rate in order to achieve a smooth and stable cloth simulation. It is hard to imagine a time when computer hardware becomes so advanced that off-line simulations are made redundant; as hardware improves, our expectations of quality also increases. Cloth simulation must keep pace with advances in other areas of virtual environments and virtual reality otherwise it will become a weak link in achieving overall realism of a scene. The computational cost of simulated clothing is the main reason why it is not in widespread use throughout real-time applications in computer graphics. One can sacrifice detail to achieve an interactive cloth simulation with a coarse mesh relatively easily, however, the detail of coarse meshes are not sufficient compared to what is achievable with tightly fitting garments using character animation and skinning approaches. The task is even more difficult considering that the cloth simulation will likely need to run with, and share

the processing and memory resources on the host system with other tasks such as character animation, artificial intelligence and motion planning.

In this work, we aim to present techniques that improve the ability to model cloth and clothing on virtual characters in real-time. We particularly focus on the development of an adaptive mesh that can change its detail in response to the current shape of the cloth. Adaptive approaches have been shown to be effective for accelerating cloth simulation but have been previously only used extensively for cloth draping. A limitation of many approaches was that refinement in the mesh could not be reversed (coarsened) although not strictly required for draping as the cloth has a final resting position. There has only been one instance where an adaptive mesh has been developed and used for clothing on an animated character albeit offline, the approach took 87 milliseconds to maintain and update the adaptive mesh and 1.2 seconds to compute the cloth simulation each frame [LV05]. Refinement was controlled by curvature of the mesh and importantly featured coarsening. Coarsening is a requirement for clothing on animated characters since without coarsening the character's movements could eventually cause the cloth to become completely refined negating any benefit to using an adaptive mesh. It is important that the relative cost of any adaptive mesh remain small compared to the simulation and collision costs in order to gain overall performance improvements through their use. Therefore, the main objective of this work is to achieve the real-time physical simulation of clothing using an adaptive mesh which is to be developed with low overheads. The limitation of many previous methods must be avoided and therefore it feature the ability to vary the coarseness of the mesh in order to allow the dynamic simulation of clothing.

1.3 Thesis Outline

In Chapter 2, we survey the state of the art of cloth simulation covering a wide range of topics divided into the following areas: cloth simulation, modelling and visualisation; collision detection; levels of detail techniques including adaptive meshes; and virtual clothing. Chapter 3 describes our novel contribution of an edge-based method for fast incremental refinement of triangular meshes. Refinement and coarsening is controlled by splitting and rejoining edges in the mesh for which we investigate a number of edge-based criteria. Simulation data is integrated in the edge-based adaptive mesh and we apply it to the task of simulating cloth throughout this thesis using a mass-spring network. Chapter 4 describes our initial work on clothing; we incorporate seaming to allow garments to be created and draped on a static character in real-time using pre-computed collision structures. Additional considerations are required to extend the work for animated characters; Chapter 5 describes our efforts in this regard, an adaptive simulation of cloth is performed on an animated character. In particular, the cost of collision detection must be minimised and we partition the character using two types of bounding structures including a simple new type of bounding cylinder. The adaptive mesh is extended to allow simple visibility based criteria to be used including for coarsening the back facing regions of garments. Finally, in Chapter 6 we present an extension to a curved surface which is modified to generate a more plausible cloth-like surface at a minimal overhead. The adaptive mesh is employed only for rendering the curved surface and we employ a fast coarse cloth simulation on an animated character. We conclude in Chapter 7 and discuss possible future work that could lead on from the research presented in this thesis.

Chapter 2

Literature Survey

Research into the physical characteristics and geometry of cloth has been studied for more than 70 years in the textile industry, beginning with "The Geometry of Cloth Structure" published in 1937 [PS37]. It was adopted in the 1980's by the computer graphics community, with more focus on visual realism and plausibility rather than the strict mechanical properties required by the textile industry. Early work with cloth predominantly featured draping of fabrics, such as e.g. [Wei86, BHW94, HC98, WAY03], often focusing on the mechanical properties of draping. A key system to measure the mechanical properties of real cloth is the Kawabata Evaluation System (KES) [KG80] . It is important to be able to predict aesthetic qualities perceived by human touch and performance in clothing manufacture relating to comfort and best fit to the human body [KN89]. KES data are non-linear and collected using four machines: tensile and shearing, bending, compression, surface friction and roughness testing machines. For example tensile data can be collected for a sample of real cloth describing the non-linear relationship between stretched length and the tension force produced. A robust cheaper alternative is FAST (Fabric Assurance by Simple Testing) but it only captures linear data [Min95]. Both Fast and KES data can be used in cloth simulations, for example Luible and Magnenat-Thalmann compare and use both for accurate cloth simulation [LMT08]. Also there has been research into

estimating cloth properties from videos of real cloth [BTH+03]. Interest has moved on to more dynamic situations from simple flags to whole garments. As such, cloth simulation involves a very wide range of research from textiles and mathematics to rendering and collision detection.

In this chapter we discuss and divide the previous related literature into four parts. First we survey existing literature concerning cloth simulation, modelling and visualisation, and then we explore the previous use of adaptive meshes as a technique to accelerate cloth simulations. We provide an overview of virtual clothing and garment simulation techniques, and finally we cover collision detection. A relatively old, but good general survey of deformable surfaces focusing on topology, geometry and deformation can be found in [MDA01].

2.1 Cloth Simulation, Modelling and Visualisation

Cloth simulations generally either use a physically based or geometric method; physically based methods broadly fit into one of two categories, Discrete and Continuous models. The simulation of cloth is a large field and there have been many methods, most being variations on similar base concepts. There is also great variation in the computational costs associated with these methods which is often a trade-off between that and realism. The requirements of the cloth simulation depend on the application, but generally we want the most realistic result that the hardware can achieve in a finite amount of time. This is especially the case in interactive applications; although sometimes fine-tuned control over the behaviour and the look and feel of cloth can be a very important feature for users such as artists in the case of animations for films (e.g. [CGW+07]). We also cover data driven techniques in relation to cloth simulation.

2.1.1 Physically-Based Simulation

Physically-based cloth simulation is formulated as a time-varying partial differential equation (PDE), which after discretisation is solved as an ordinary differential equation (ODE) [BW98]. The state of the cloth is not predictive; the instantaneous forces and accelerations of a point within the cloth depend on the surrounding positions and velocities. In-order to evolve the cloth system with time; the system is stepped forward in discrete time steps, numerically integrating Newton's equations of motion (ODE). The numerical solution convergences to the real solution as the time step approaches zero, but generally we want to use a time step that is as large as possible for best performance while still producing a smooth animation. Haulth [Hau05] describes numerical techniques for physical cloth simulation including particle systems, finite difference methods and finite element methods while covering explicit and implicit integration and stability. For a more general review, Nealen *et al.* [NMK$^+$05] review physically based deformable models and applications of them such as entertainment, surgical simulations, fluid/smoke animation, hair and also cloth.

There are many numerical integration methods and techniques; their accuracy and ease of use varies but we will discuss a few popular ones here for the reader's convenience. One of the simplest methods is Euler integration or the Forward Euler method, which works on the premise that you can find the approximation of a nearby point on a curve by advancing in the direction of the tangent to the curve (i.e. $y_{n+1} = y_n + h.f(t_n, y_n)$ where h is the step size and $f(t, y)$ is the gradient). It is a 1st order explicit method and as such relies on small time steps in order to remain accurate and stable. An example of a 2nd order explicit method is Verlet integration, which was made popular for molecular dynamics by Verlet [Ver67] but since has been borrowed by the computer graphics field including for cloth simulation. Velocity Verlet is a popular form where velocity is not stored separately but is incorporated

as the difference between the current and previous positions [SABW82], which tends to achieve a greater stability over that of Euler's with little increase in computation cost. Higher order methods also exist; the most well-known family being Runge-Kutta which includes a 4th order method commonly referred to as RK4 that is popular. Runge-Kutta is an example of a one-step method, although using intermediate steps to compute the time step; all information is discarded and there is no memory of the previous steps. On the other hand, multi-step methods save the information from previous steps and use it to gain accuracy particularly for smooth problems; linear examples of multi-step methods include the Adams-Bashforth and the Adams-Moulton methods [AHS09]. Higher order methods can perform worse for numerically stiff problems such as found in cloth simulation, where there can be large variations from step to step which can be compounded by unpredictable collisions or external forces. Explicit methods are solved one particle at a time independently as if particles were not coupled and using only the current state of the system; conversely implicit methods solve an equation for a coupled system based on both its current and next (future) state. Implicit methods are not easy to use, requiring the use of iterative solvers operating on large (but often sparse) matrices. Implicit methods can, however, support a larger time step and as such their increased cost can be offset somewhat by using a fewer number of larger steps. The Backward Euler method is one of the simplest examples of an implicit method; it uses the approximation that the value of the next step can be found by advancing in the direction of the gradient of the next step, i.e. similar to Forward Euler but it is as if you are working backwards towards the original state. The value of the next step appears on both sides of the equation (see Equation 2.1.1); hence an iterative method is required to converge on the solution.

$$y_{n+1} = y_n + h.f(t_{n+1}, y_{n+1}) \tag{2.1.1}$$

The Conjugate Gradient (CG) method is often used to solve systems of linear equations, normally applied to very sparse systems that are too large to be solved by direct methods (for example by the Cholesky decomposition). An important aspect of the CG method and other iterative solvers is the speed of convergence, and therefore they are often used with Pre-conditioners to improve the problems suitability for numerical integration, hence the Preconditioned Conjugate Gradient (PCG) method is used for cloth simulation. The work of Baraff and Witkin's [BW98] is highly cited, and was instrumental in the increased use of implicit integration for cloth simulation. They used a maximum time step of 0.02 seconds (with adaptive time which automatically reduces the time step when required for stability), this 20ms seconds corresponds to 50Hz and although the author's do not specify their reasoning for using this maximum rate it may coincide with the character's animation rate. Also selecting a higher time step will likely result in the adaptive time stepping automatically reducing it much of the time anyway to maintain stability, in their examples the smallest adaptive time step needed was 0.625ms. Their approach was successfull in employing a large range of bending stiffness's while only effecting computation time by 5%, taking around 10.3 seconds per frame for a 2602 particle system running in $\mathcal{O}(n^{1.5})$ time. Baraff and Witkin were always able to maintain constraints exactly (including those on individual particles) independently from the number of CG iterations needed using their Modified Preconditioned Conjugate Gradient (MPCG) method.

Implicit-Explicit (IMEX) integrations schemes have been developed and used for cloth simulation. IMEX schemes allow partially stiff problems to be solved more efficiently, by performing implicit integration with the linear numerically stiff parts and

explicit with the remaining non-stiff parts [EEH00]. Bridson *et al.* [BMF03] presented a mixed implicit-explicit scheme where implicit integration was only performed with the velocity-dependant forces (e.g. damping forces) that could be efficiently solved by the CG method due to them being linear while not requiring any preconditioning. Explicit updates of positions allowed them to modify velocities to maintain constraints such as strain limiting; this allows weaker springs to be used. They state their method was well behaved even if bending forces are strong and dominate all other elastic forces.

Boxerman and Ascher [BA04] presented a technique to improve the sparsity of a cloth system and decompose it into parts that may be solved efficiently and in parallel by including an adaptive IMEX scheme. They made further improvements to sparsity by exploiting the physical model of Choi and Ko [CK02] and by also considering static constraints while improving the performance of Baraff and Witkin's MPCG technique [BW98]. The performance was highly dependent on the situation, but in a cloth covered arm bending simulation their method required 35-50% less row-vector multiples which the authors say they have found to correspond well to CG computation times but it remains to be seen how it would scale to a complete garment.

Inextensibility requires large in-plane forces, creating a numerically stiff problem that requires very small time steps to solve. Stability problems come from the fact that the error differences in explicit methods overshoot; larger time steps and or larger forces can cause the system to oscillate and explode as the overshoot gets exponentially worse. Some form of damping is almost always required to remove energy from the system; stability is improved but too much often reduces realism. Baraff and Witkin's [BW98] employed an adaptive time stepping approach to prevent divergence; they discard a step and repeat it with a smaller time step when simply

a large change in the stretch of a triangle is detected. Approaches like this must be carefully balanced to give overall performance gains, returning to larger time steps as fast as possible while minimizing discarded steps. Baraff and Witkin's approach is to retry a larger step size after two steps, if this fails the step size is reduced again but a longer wait is imposed before trying a limit of up to 40 steps. The midpoint method can be thought of as a combination of the Forward and Backward Euler methods where the gradient is evaluated half way through the step and it is a one-step method belonging to the Runge-Kutter family of methods and achieves 2nd order accuracy. Implicit methods undershoot and therefore do not feature instability like explicit methods, though errors still impose limitations on the size of the time step tending to over damp the cloth. Volino *et al.* [VMT05] used the Implicit Midpoint Method for cloth simulation; although the Midpoint Method never becomes unstable they state damping is required for stability. So to improve its efficiency and stability, they employ adaptive viscous damping leading to animations with similar motions compared to the Backward Euler method but at a reduced cost since a larger time step could be used.

The cost of implicit methods can be prohibitive to detailed cloth simulations, many authors work to combat this. Kang *et al.* [KCCP00] proposed an efficient approximated implicit method that ran in $\mathcal{O}(n)$ time with n springs. They incorporated a stable damping approach with air interactions. Although their method was stable, they required the use of a technique to correct super-elongated springs (ones that are unrealistically stretched) similarly to Provot's method [Pro95].

Discrete Particle-based Simulations

Discrete simulations are where the cloth is made from by discrete parts, often it is modelled by particles distributed over its surface; and interactions between the particles give rise to the cloth's behaviour. These interactions can be specified by

energy equations, with distance and curvature constraints. A widely used method is a mass-spring network or system. This approach uses an interconnected particle system of point masses and springs. Often Hooke's law type springs and equations are applied with Newton's Second law together with a numerical integrator. It has been used with both Triangles and Quad meshes defining the mass-spring layout, often with additional springs to simulate bending and shearing forces such as [Pro95, DSB99, ZY01, RC02, FGL03] and others. Figure 2.1. shows typical spring layouts for regular quad and triangle meshes. The popularity of the mass-spring network method lies in its ease of implementation particularly with an explicit integrator, but it can struggle with stability problems if care is not taken. In-extensibility requires large in-plane forces, creating a numerically stiff problem that requires very small time steps to solve. Bending forces are not always simulated by springs; variations based on the angle between adjoining faces across an edge are popular. Volino and Magnenat-Thalmann [VMT06] presented another alternative using the weighted sum of vertex positions to derive linear bending forces suitable for implicit integration. Accurate bending approaches from the field of engineering pose problems for standard cloth simulation methods, Thomaszewski [TW06] explore some different bending models for thin-flexible objects and describe their transition to a discrete sitting. Other bending models exist such as the quadratic bending model of Bergou et al. [BWH$^+$06]

Provot [Pro95] introduced an inverse dynamic procedure to correct the length of super-elongated springs which improved the usability of mass-spring systems as lower (more stable) spring constants could be used while still being able to limit stretching. Corrections are applied to positions of vertices to restore the length of the springs assuming that the direction is correct. The order of applying the corrections affects the convergence of the method; Provot did not address this and used an ad-hoc order. Kang *et al.* [KCCP00] proposed a dynamic ordering used with Provot's method based

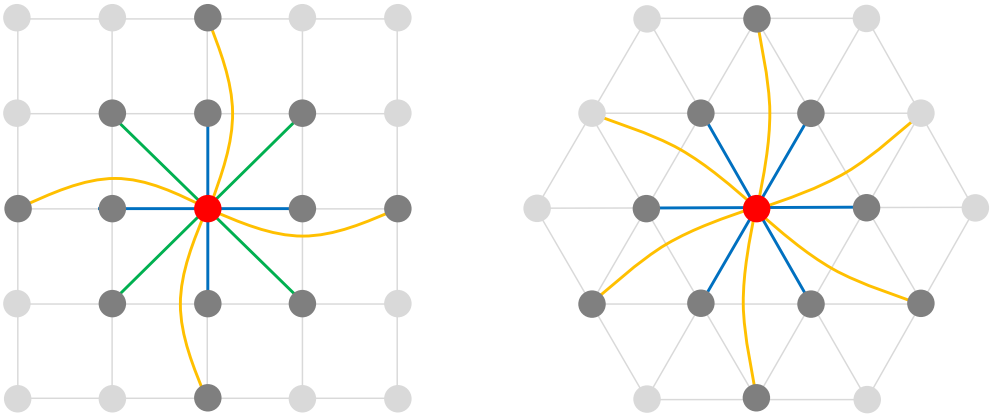

Figure 2.1: Mass spring networks constructed from a quad mesh (left) and a triangular mesh (right). Connected springs are highlighted in each around for a single vertex (red). Left) Each vertex is connected to two horizontal and two vertical stretch springs (blue), four diagonal shear springs (green) and four bend spring (yellow). Right) In the case of a regular triangular mesh with valence size, each vertex is connected to six stretch springs (blue) and six bend springs (yellow). There are no shear springs but for each triangle, its three stretch springs while working together will also counteract shearing.

on elongation length, using a bucket based sorting approach with buckets distributed uniformly over the range of the elongation lengths for n springs. This worked efficiently in $\mathcal{O}(n)$ time, they did not consider the order within buckets so the number of buckets and distribution between them is important to the result. More recently this has been researched by Ozgen and Kallman [OK11]. They considered the main direction of stretch to be in the direction of gravity, they traverse cloth edges starting simultaneously at fixed vertices and propagating along the direction of gravity building a correction map. The correction map describes the order in which to visit each spring to correct the length. Only the second (lower) vertex of each spring needs to be corrected assuming the first was either fixed or previously corrected. It is important to synchronise the expansion from fixed vertices to avoid artefacts and each vertex

has a visited count, and may only be visited a limited number of times. Springs were classified into three categories: Horizontal, Vertical and Shear springs; they claimed that this priority ordering produced correction maps with the "best-looking" deformations out of six possible combinations but they only provided illustrations for four out of the six combinations. Fixed correction maps are pre-computed for real-time performance; however, multiple maps can be pre-computed and switched between. This was demonstrated with a dress, vertices make contact with the knees at certain places during the animation and can be considered fixed, so three correction maps were needed: a) no knee contact, b) right knee contact, c) left knee contact. The authors therefore conclude that their method is suitable for cyclic animations, where multiple correction maps can be pre-computed. The approach is limited in situations where dynamic correction maps are needed or there are variable external forces (in addition to constant gravity) which would affect the main stretch directions. The author's say they achieved an improvement of the computation time by 80%, but unfortunately they did not give any timings or further information.

A related area is the simulation of thin shells which are flexible and extremely thin like cloth, but include a range of plastic to elastic materials such as light bulbs, paper and rubber. Thin shells normally require continuum mechanics to solve. However, Grinspun [GHDS03, Gri08] presented a simple discrete shell model for triangle meshes, and described how it could be implemented quickly by modifying a standard cloth simulator. Wardetzky *et al.* [WBG+08] proposed a hinge-based bending model for the discrete simulation of plates and shells derived from a purely geometric view (and also a derivation based on finite elements). They demonstrated their method on a number of examples, more interesting including cloth, such as a flag and draping on a sphere.

Kang and Cho [KC02] presented a bi-layered method to generate realistic wrinkles

with a large number of vertices by using two mass-spring networks, one rough mesh for global deformations (experiencing internal and external forces) and a fine mesh for realistic wrinkles (experiencing only internal forces). The bi-layered approach avoids over damping so relatively large time steps can be used (1/30 seconds).

Andrews [And06] modified a rigid body physics engine to simulate cloth, where the engine itself provides no mechanism to model deformable objects by connecting a grid of rigid body components by damped springs. The springs were created using custom joints, to allow cloth-like behaviour at interactive frame rates. This presents a very easy implementation, where many details such as numerical integration are handled by the physics engine. Often simulations focus on one domain, i.e. just the cloth simulation but its integration with other materials and objects are interesting but two-way interactions between different domains are difficult. Baraff and Witkin's [BW97] present an interleaved method for accurate geometric behaviour with minimal overhead compared to the simulations running independently. They were able to simulate particles streams, rigid bodies and cloth that were interacting with each other or otherwise connected together (cloth fixed to rigid objects).

Continuum-based Simulation

On the other hand, compared to discrete particle-based simulations, continuum-based methods aim to model the cloth as a whole by treating it as a continuous surface, the most widely used method to solve this is the Finite Elements Method (FEM). FEM is a numerical method to find the solution of partial differential equations (PDEs) by approximation, ensuring numerical stability applied to problems such as fluid flow, heat and elasticity. In FEM complicated problems are discretised into a finite number of small elements which are then solved in relation to each other. Cordero [Cor05] presented a model based on this for realistic behaviour of cloth while also providing a solution to the poor convergence of elastic properties usually associated with FEM.

The example given was of a figure dressed with a poncho consisting of 3325 elements; each frame took 14.67s with a time step of 10 μ seconds using a RK4 integrator producing detailed folds and wrinkles.

Thomaszewski *et al.* [TWS06] presented a method to model consistent bending using co-rotational subdivision finite elements in order to correctly simulate the folding and buckling behaviour of cloth. They compared the compression of a real fabric cylinder with simulated ones. Their method produced visually similar folding patterns to the real one, however, the cost of their method makes it only suitable for offline simulation. In comparison using a standard simple bending model, the fold patterns were different to the real sample; also the patterns completely changed using a finer mesh with the simple model.

Rodriguez-Navarro and Susin [RNS06] also presented a FEM cloth simulation, this time using the GPU. They improved a GPU-Gradient Conjugate method (required for solving the linear equation systems) together with an image based collision and self-collision detection algorithm. Achieving a frame rate of 27 fps on the GPU compared to only 3 fps on the CPU for a 128 x 128 isolated square mesh, which is compared to a Constraints based methods running at 2.6 and 0.3 fps, and a Mass-Spring System running at 60 and 8 fps on the GPU and CPU respectively. Though they provided no visual comparisons for the three methods, it would be unlikely all three would look the same. They demonstrated the high quality produced by their method with a detailed animated character wearing a dress, but unfortunately they gave no indication of the performance in this complex case.

Goldenthal *et al.* [GHF$^+$07] made use of Constrained Lagrangian Mechanics to limit stretching in the warp and weft directions by enforcing constraint equations on a quadrilateral mesh. Their Fast Projection method acts as a velocity filter, and can be integrated into existing simulation code although in this claim they were referring

to the use of implicit integration only. The method achieved a 25 times speed up with 1% allowed strain compared to that of only using implicit springs. In the case of draping a square 2D piece of cloth over a sphere; their method was faster than all of the four previous methods they tested, for different values of allowed strain and different numbers of vertices. Their work was also demonstrated on a virtual dancer, with a 10600 vertices mesh taking an average of 9 seconds per frame using fast projection and an implicit treatment of bending and shearing. Interestingly, the low strain achieved by their method ensured that the character's tight fitting trousers did not fall below its waist line.

English and Bridson [EB08] presented a finite element method using non-conforming elements to simulate in-extensible cloth with a new second order integration scheme. The use of non-conforming elements allows zero in-plane deformation (stretching) while freely allowing bending, but it produces discontinuous meshes. They therefore required the use of a second mesh for rendering which is coupled to the mesh of non-conforming elements. Improved convergence of their new integration scheme reduced the cost of a simulation of a cape from 5.37 to 3.96 seconds per frame as well as reducing numerical damping.

Non-linearity of cloth simulations is important to realism, but cloth systems are often linearised for simplicity and performance. Volino and Magnenat-Thalmann [VMTF09] presented an approach based on continuum mechanics for simulating non-linear tensile stiffness in an efficient way, by using an adapted strain-stress laws that precisely describes the cloth's non-linear behaviour. Forces are calculated on triangles elements based on their 2D parametric coordinates and their deformed 3D world coordinates. Their method was only 15% more expensive than an equivalent mass-spring network but was much more accurate. They were able to iterate 17,500 elements per second, the CG method took more than 50% of the time (10-15 iterations) for each

timestep. Thomaszewski *et al.* [TPS09] presented continuum-based strain limiting, allowing accurate control over all strain components (such as shearing and stretching) without the problem of discretisation dependant behaviour.

2.1.2 Geometric Techniques

In this section we cover geometric techniques; these are methods that do not make use of physics equations or integration as such; including geometric simulation and geometric wrinkles.

Geometric Simulation

Geometric techniques have been developed, the first being the use of skeletal animation or skinning; this is not suited to free-flowing garments but can produce good results for tight clothes. For instance cloth vertices remain fixed at a distance away from a point on the underlying character's mesh and the mesh is animated in the same way as the character's mesh. There has been work on a hybrid approach that uses this method for some parts of garments and a physically-based method for the parts that are free-flowing; Cordier and Magnenat-Thalmann [CMT02] achieved good results with this, much faster than simulating the whole garment purely by a physically-based method.

Müller [MHTG05] presented a new approach to the simulation of 3D deformable objects without the need of connectivity information and also being unconditionally stable unlike physically based methods. They replaced energies by geometric constraints, and replaced forces by distances between current and goal positions. The approach is mesh-less and uses a point cloud; goal positions are calculated by shape-matching of the undeformed state to the current state of the point cloud. Stumpp *et al.* [SSBT08] proposed an adapted shape-matching approach for the efficient and robust simulation of clothing that is based on Müller *et al.*'s deformation model. The

problem is partitioned into a set of overlapping clusters of three points that enables efficient extraction of the optimal rotation for the shape-matching as well as allowing out of plane deformations. The method supports a consistent treatment of shearing, stretching and bending moments and features additional 1D clusters to further reduce stretching. Computing the elastic response (excluding collisions) for a square piece of cloth was linear in time complexity taking approximately 30ms for 10k vertices. The example given for user interaction consisted of 109 vertices running at only 38 frames per second; evidently it is difficult to consider the methods real-time feasibility for clothes without a breakdown of the timing for this example.

Decuadin *et al.* [DJW+06] presented a fully geometric approach to clothing design, starting with 2D sketches of the contours and seam lines of the garment as drawn by the user directly on to a virtual mannequin. A 3D surface is automatically created and a natural rest state of the garment is calculated including folds due to collisions and gravity. A 2D sewing pattern can then be output from the final garment and used to create an actual real garment for a comparison to the 3D one. The method is good for garment prototyping; however, if a dynamic simulation is required the 2D outputted patterns could be used for a traditional physically-based simulation or another dynamic simulation method. Chen and Tang [CT10] also presented a fully geometric cloth simulation but it was general purpose. Inextensible cloth is simulated subject to gravity and collision-free constraint and their algorithm interpolates a smooth developable surface through a set of anchor points (and dynamic anchor points for collision) generating high quality cloth surfaces. The disadvantage is the approach is not dynamic and a final steady shape is eventually found, (in one example, a 961 vertex mesh takes 28.33 seconds to complete 56 iterations to reach the final state). However, it is robust and numerically stable and is able to achieve inextensibility and triangles remain isometric to their initial shape within very small tolerances.

Geometric Wrinkles

Wrinkles are very important to cloth animation; they give users visual cues that suggest that the cloth is deformed. Even slightly crumpled clothes look very different from that of perfectly flat cloth. There has been a lot of work recently in the area of generating cloth with wrinkles. Many methods work on coarse meshes (that are faster to simulate and allow real time rendering and interaction). Wrinkles can be added as details through texture and bump mapping shading methods. Methods designed explicitly with wrinkling in mind have become somewhat popular and several methods have been developed such as these for skin: [BKN02, RDMB08], cloth: [HBVMT99, KWH04, CGW+07, DJW+06, Lov06, RPC+10, WHRO10] and both: [MC10].

Larboulette and Cani [LC04] presented a method for designing wrinkles on pre-simulated cloth or skin using control curves and length conservation. Users define regions of influence on the mesh by placing 'wrinkling tools', the wrinkling tools controls real-time local deformation and subdivision is performed on the fly as needed. In a clothing example, it runs at 17 frames per second with 4 wrinkle regions. Loviscach [Lov06] presented an easy to control method to generate and render wrinkles without requiring a highly tessellated surface, wrinkles can be introduced in the rest pose through a 3D painting interface. A shirt consisting of 455 vertices, covering a screen area of 330,000 pixels was rendered with wrinkles at 328fps. It should be noted that at this point the collision handling can destroy the wrinkles; Bridson *et al.* [BMF03] noticed and addressed this by resolving collisions to a band above the surface such that wrinkles were preserved.

Other methods work on entire meshes, requiring less work from users but often with less user control. Rohmer *et al.* [RPC+10] combines a coarse cloth animation with a post-processing step for efficient generation of realistic-looking fine dynamic

wrinkles. It uses the stretch tensor of the coarse animation as a guide for wrinkle placement, ensuring temporal coherence by using a space-time approach for the placement mechanism. The method is fully automatic, with a single user control parameter to mimic different fabrics. Coarse simulations take 25ms to 2s per frame, with wrinkle generation taking 1s to 2s per frame compared to 10-25 seconds per frame for the high resolution simulations. However, high resolution simulations run at 10 to 25 seconds per frame, achieving a one order of magnitude speed up.

Wang *et al.*'s [WHRO10] method uses a pre-computed data set to perform example based wrinkle synthesis for clothing animation. The wrinkles are layered onto a coarse base mesh capturing similar fine scale wrinkles to the high-resolution simulation. The high resolution mesh of 25k vertices took 2-3 minutes per frame; the coarse simulation of 638 vertices took 72ms and 84ms with the added wrinkles.

Müller and Chentanez [MC10] presented a fast yet simple method to add wrinkles to a dynamic mesh such as cloth or skin. A higher resolution wrinkle mesh is attached to the coarse base mesh, wrinkle vertices are permitted to deviate from their attachment position within a limited range (specified by the user by painting on the mesh. A static solver is used to calculate the shape of the wrinkle mesh and it runs in parallel to the motion of the base mesh. Real-time performance is achieved; for a shirt on a character using a 7k triangle base mesh with a 28k triangle wrinkle mesh, the solver takes 4ms for five iterations per time step.

It is not the case that all coarse simulations employ a fine mesh; Hadap *et al.* [HBVMT99] makes use of bump mapping, by using a user defined wrinkle pattern modulated by a triangle deformation. At the time, on a MIPS R10000 200 MHz processor, wrinkle coefficients took 5 minutes per one thousand triangles. Later, Kimmerle *et al.* [KWH04] extended Hadap *et al.*'s algorithm with automatic procedural texture generation for the wrinkle pattern. They generated a multi-layer texture

from the strain deformation tensor of the simulation with loop subdivision to resolve collisions, taking a combined 3.48 minutes per simulation second compared to 12.2 minutes for the high resolution simulation.

2.1.3 Data-Driven Techniques

There are many techniques that are applied to pre-existing cloth simulations based on machine learning and data-driven techniques. Data-driven method in this context are methods which rely on the use of pre-recorded simulation data to create new simulations with increased performance and or realism by capturing qualities found in the pre-existing data. For instance, Kavan *et al.* [KSO10] presented a method to construct a skinned mesh from arbitrary vertex animations suitable for standard linear skinning. Their algorithm is based on iterative coordinate decent optimisation which achieves greater accuracy than previous methods, and yet requires one to two orders of magnitude less pre-processing time by employing sparse vertex weights. A 360 frame animation of a cloth skirt with 5095 vertices takes 22.8 seconds with 22 bones compared to a previous, less accurate method taking 25.2 minutes.

Cordier *et al.* [CMT04] presented a data-driven approach for real-time processing of clothes, starting with an analysis of a pre-simulated cloth sequence assuming positions at fixed time intervals are known. By comparing the cloth's behaviour to the character's skeleton an optimal combination of physical simulation and geometric approximation can be found. At run-time using this combination good performance and decent quality is achieved as long as the character's motion is close to the original sequence used. An Evening Dress model is reduced from 2992 triangle faces to a coarse mesh of 110 vertices and rendered using surface patches, running at 26 fps.

Online clothes shopping software (also known as virtual try-on software) aims to give the customer an idea of how the garment will look on them; it is an important

sales tactic preferable to using human models as it is not feasible to hire models of every shape and build. Cloth simulation is generally too slow for use with this software, it is important for the user to get immediate results as they will no-doubt want to flick through many garments quickly to see at a glance if they look good. Alternatively to cloth simulation, a set of models are used with pre-fitted clothes, allowing the user to choose one with the greatest likeness to them. However, this requires a great deal of different models all pre-fitted with different garments. Gillies *et al.* [GBC04] presented a compromise with a data-driven technique, by performing Principal Component Analysis on a set of avatars that had clothes pre-fitted to them offline. Users are presented with a weighted sum of these avatars to represent themselves, created from the user's tailored measurement or alternatively from a 3D scan. Then using those same weightings, the pre-fitted clothes could be dressed onto the user's avatar.

On a related noted, Ruiz and Buxton [RB01] presented an approach to reconstruct the surface of 3D scans of clothed people. They cite applications for creating, manipulating and animating life like human models such as for use in entertainment and films, games and the fashion industry. They use a model-based procedure designed to cope with unorganised 3D point clouds produced by humans wearing clothes with creases and folds. Exploiting the model of properties of fabric from [VSC01a], Ruiz and Buxton's procedure gives accurate results even with high levels of noise, irregular sampling and missing data. The surface fitting works by energy minimisation with the cloth model constrained by 'data-fitting forces', iteratively, the cloth model converges to the scanned points.

Feng *et al.* [FYK10] developed a hybrid method to capture the relationship between two resolutions of a cloth simulation. The data driven approach is trained using rotation invariant quantities extracted from the cloth models, and is independent of

the simulation technique for the lower resolution model. Combined with fast collision detection and handling, using dynamically transformed bounding volumes, real-time performance is achieved on the GPU. The disadvantage of this method is that it does not generalise well, where the fine mesh training data is very different from the coarse simulation. This is an important consideration in many data driven approaches, not just for cloth.

de Aguiar *et al.* [dASTH10] presented a method for learning clothing models that treats training data as a black box. It approximately resolves cloth-body collisions, and is a method to bridge skinning and physical simulation with the speed of the 1st and dynamic effects from the 2nd. The advantage of the method is the performance allowing 1000 or more characters in real-time (0.00049s for 1,454 triangle dress or 0.01497s for 40k triangle dress per 1/30th frame). However, the two-step training method took 1.5 hours in pre-processing time, the limitation is that it may not also generalise well to things not seen in the training data set and it also only approximately resolves cloth-body collisions.

Kavan *et al.* [KGBS11] presented a method for learning linear up-sampling operators for physically-based cloth simulation allowing coarse meshes to be enriched with mid-scale details in minimal time and memory budgets. They start by pre-computing a pair of coarse and fine training simulations aligned with tracking constraints using harmonic test functions. Then the upsampling operators are trained with a new regularization method, and reintroduce high frequency details into the coarse simulation using oscillatory modes. A Skirt with 196 coarse and 7016 fine points took 0.8ms and 0.1ms to upsample on the CPU and GPU respectively; the coarse simulation took 0.5ms to simulate. The limitations include that there is no collision processing on the fine mesh (bounding volumes were expanded instead). A simple coarse model is used, with linear upsampling operators and in its current form it is not suitable for

much higher resolutions.

Kim and Vendrovsky [KV08] presented a technique for a tool for animators, it allows users to drive the deformations of one object (garment) using the shape of another (character) for tight fitting clothes or where the deformation is closely linked to the pose (of a character). By analysing reference poses weightings are then calculated for fast linear blending, but it requires the use of collision detection. Secondary motions are missing, though if they are needed the technique can be used to guide the cloth simulation using spring constraints. It can be used in a hybrid fashion applied to more rigid areas and the dynamic areas are simulated.

2.1.4 Rendering, Texturing and Shading

Many authors do not specifically mention the rendering of the cloth, choosing to focus on the simulation and collision. However, the visual appearance of cloth can be as important as the deformation, often the look and feel of the cloth is due to the deformation combined with lighting allowing us to see the shape and wrinkles by way of shadowing and highlights. Subtle effects can make the difference between a realistic looking leather garment and plastic looking one. Programmable graphics shaders allows the use of many lighting models such as Phong [Pho75], Cook-Torrance [CT81] and Oren-Nayar [ON94]. It is beyond the scope of this thesis to discuss them here; however, one should try to select the lighting model to most accurately reproduce the look and feel for the type of cloth you desire to model. Often the choice of lighting parameters are more important than the actual choice of lighting model; the perception of simulated materials has been widely studied, for instance [Rus08].

Many clothes have colour patterns and stripes, made from woven coloured thread, dyed or printed; naturally rendering with textures can provide this. Daubert *et al.*

[DLHS01] presented an approach that takes into account view-dependent effects including occlusions, shadows and illumination for rendering of knitted patterns, in particular a woollen sweater with knit and purl loops. They model the micro geometry of knits and weaves using an implicit surface of Bézier curves, and they employ a spatially varying BRDF (bidirectional reflectance distribution function) representation. Later Sattler [SSK03] uses a BTF (bidirectional texture function) but instead of using BRDF for each texel, they generate view-dependant texture maps using principal component analysis. Their approach is able to be rendered and lit by point lights or environmental maps and also including self-shadowing producing high quality results using graphics hardware. Adabala *et al.* [AMT03] presented a technique for procedurally creating textures for twists of threads supporting variation in appearance due to the tightness of twist, thickness and roughness. These thread textures were composed into a texture which can be repeated seamlessly using a coloured weave pattern. Adabala *et al.* [AMTF03] developed this further to create a multi-texturing cloth rendered, combining procedural thread texture with generated BRDF textures and horizon maps (for self-shadowing). The final result was demonstrated on a dress allowing, in real-time, the illusion of woven thread with transparency resulting from an adjustable gap size between threads.

Variation is also an important consideration for realism, this is largely considered by the field of crowd rendering and often not considered in most cloth research. Furthermore, most crowds do not employ cloth simulations and rely on skinning approaches. A survey of real-time crowd rendering can be found here [RD05] and another on real-time crowd simulation including behaviour and rendering can be found here [ASDB08]. However, there is overlap in some research such as [DMK$^+$06] where Dobbyn *et al.* presented a real-time crowd rendering system for clothed characters using imposters with pre-simulated cyclical cloth animations. UV mapped imposters

are rendered and variation is added by selecting between diffuse textures.

Morimoto *et al.* [MTTT07] describes a method to simulate and visualise dyeing cloth based on weave patterns using a liquid diffusion model. Later Morimot *et al.* [MO09] were able to generate simulations of traditional tie-dyed cloth. It is possible that this could be applied to colour complete garments such as Japanese tie-dyed dresses. There has been work in simulating cloth while interacting with fluid [KSK04], the cloth is modelled with thickness and fluid particles exert focus on a garment and collisions can be treated uniformly. In reality, we expect the cloth to become wet and behaviour differently than when dry, while also visually appearing 'wet'. For instances, Huber *et al.*'s [HPS11] wet cloth simulation allowed the two-way coupling cloth and fluid simulations; the cloth was able to soak up fluids by diffusion which influenced the behaviour of the cloth via the additional weight.

2.1.5 Knitted and Yarn Level Simulation

Knitted fabrics appear very different to tightly woven ones, for example tending to have visible holes and different knits or stitches change the properties of the resulting cloth. A lower level approach can be employed to simulate or visualise cloth at the yarn level. Physically correct micro structures of cloth are key to its appearance, Meißeer and Eberhardt [ME98] presented an approach to correctly visualise them. Their system is able to use 'machine-code' used by knitting machines directly, and can be used for teaching or designing of knitted items by simulating a complete knitting machine with global parameters. Stiches are approximated on a grid by a mesh based structure, and then undergo physically based refinement simulating stretching, repelling and bonding of yarn using a mass-spring system.

Nocent *et al.* [NNR01] presented a mechanical level of detail method for knitted fabric, accelerated by reducing the degrees of freedom with parameter reduction using

parametric bounding volumes. Ngoc and Boivin [NB04] describes a non-linear cloth system which simulates the cloth as a complex yarn interlaced structure with friction but avoids a detailed 3D geometric model of actual yarn. They incorporate the using of KES data producing a realistic cloth simulation at fairly high cost, taking 0.115, 1.321 and 6.302 seconds per frame for 400, 2500 and 4900 particles respectively on a 2 GHz processor with a 0.001 second time step.

Kaldor *et al.* [KJM08] presented a detailed knitted cloth simulation at the yarn level, modelling individual yarns by multiple B-splines with length constraints. Stretch, bend and inter-yarn contact forces are simulated with damping and friction. One example was a simulation of a legwarmer with 35,200 spline segments was achieved offline in 10.8 minutes per frame. The method achieves similar static shapes and properties as real knitted; characteristics emerge starting from a flat input configuration and only differ by their interlocking pattern of stitches. For example, garter stitch becomes shorted and width, rib stitch is the opposite and stockinette stitch causes the fabric to start to roll up.

2.2 Collision Detection

The field of collision detection (CD) covers a wide variety of techniques ranging on input formats and complexity; therefore in this section we focus on reviewing collision detection methods that have been specifically developed or used for cloth and clothing or deformable surfaces while explaining general concepts that are important to the understanding of such work. There have been a number of surveys of collision detection techniques, [JTT01, Yus02, KHI+07], including those suitable for deformable objects [TKH+05], and a recent walk through of continuous collision detection techniques in virtual environments [SB11].

Collision detection is an important part of cloth simulations; cloth must be prevented from penetrating objects and surfaces, and garments must be prevented from penetrating character bodies. It is indeed rare that cloth is considered by itself in actual applications except perhaps flags or hanging clothes where a few simple particle positional constraints are enough. Depending on the situation, the cloth may self-intersect and self-collision detection (SCD) can be important due to the often very flexible nature of cloth. However, SCD is time consuming to detect and resolve, so it is often left out in some time dependant situations such as real-time clothing on characters. In the case of physically-based simulations; penalty forces can be applied as the cloth comes close to the surface to repel it but it is hard to achieve the perfect response to stop the collision, without under or over-shooting. Baraff and Witkins [BW98] used strong damped springs to push the cloth apart between cloth-cloth collisions with tangential damping forces to counteract sliding and emulating dynamic friction. However, for cloth-object collisions they enforced positional constraints on cloth particles. A simple approach is to 'push' cloth vertices onto the surface it has penetrated. Bridson *et al.* [BMF03] pushed cloth vertices instead onto a band above the surface so wrinkles were not overly destroyed, as they said they would be if pushed

flat with the surface. A more complicated method is to resolve collisions in a global way, Volino and Magnenat-Thalmann [VT00] presented a general geometrical correction method using the conjugate gradient method to find displacements to satisfy all constraints simultaneously.

2.2.1 Two-Phase Collision Detection

Two phase collision detection is almost always used, beginning with a broad phase followed by a narrow phase. The broad phase is to quickly prune away large areas of non-intersecting objects or regions, outputting a potentially colliding set which is then processed in the narrow phase. Most often bounding volume hierarchies are employed for the broad phase, but alternatives exist such as spatial hashing [THM+03]. Bounding volumes allow complicated objects to be represented by simpler volumes such as boxes, spheres and convex hulls that encompass the object; generally the quality of fit is traded-off against the cost of intersection testing. The narrow phase is where actual contacts are computed and resolved, such as between cloth particles or triangles and object triangles. It can be very costly, so an efficient broad phase is needed for all but simple applications.

Zhang and Yuen [ZY01] use a bounding hierarchy for collision detection with their multi-level cloth simulation; they add new nodes into leaves of the tree as the mesh is refined. They also use their voxel-based self-collision method from [ZY00]. They employ a uniform spatial subdivision technique, where the voxel size is determined by the longest edge in the cloth guaranteeing no edge will penetrate more than four voxels. Therefore each voxel will only need to be checked against four others reducing the number of potential collisions. Self-collision performance is improved by considering the curvature, by the fact that low curvature areas cannot self-intersect which is also exploited by [VT94, VCMT95, Pro97] previously.

Other bounding volumes have been used such as k-DOPs (discrete oriented polytope), constructed from k number of planes at infinity moved together until touching the object. The simplest is an axis-aligned bounding box, made from 6 planes. Mezger *et al.* [MKE02, MKE03] presented improvements for the efficiency of bounding volume hierarchies for cloth animation, employing 18-DOP volumes. They perform orientated inflation of the volumes with lazy updates using a tolerance distance. They also found it was worth considering other trees such as a quad tree instead of binary if overlap tests are cheap, reducing recursion depth although increasing the number of overlap tests in the worst case. In an example of clothes (10757 particles) on a walking avatar (28784 polygons), the hierarchy update took 114 milliseconds and collision detection took 49 milliseconds per frame. Subsequently, K-DOPs have been used by several other authors subsequently [WKK+05, BWH+06, TCYM08, HF07].

2.2.2 Discrete versus Continuous Collision Detection

There are two main collision detection approaches, Discrete Collision Detection (DCD) and Continuous Collision Detection (CCD). Discrete collision detection is where objects are considered stationary or static at discrete time steps. Objects are firstly advanced and then interpenetrations are detected and resolved at discrete steps. In order to find the point of collision, the simulation may be backtracked and then re-advanced with smaller and smaller time steps, homing in on the moment of collision. This is rarely satisfactory and very expensive, analytic methods are often used instead to find contact points between objects, and use them to resolve the collision using constraints or with penalty forces. There is no concept of what happens to the objects in-between time steps and as such DCD relies on small relative movements compared to the size and velocity of the objects otherwise tunnelling may result. Tunnelling is where a collision is completely missed due to the interpenetration only occurring

in-between the discrete time steps. For example a small ball with a high velocity can move completely through a wall in a single time step and no collision will be detected by DCD (see Figure 2.2).

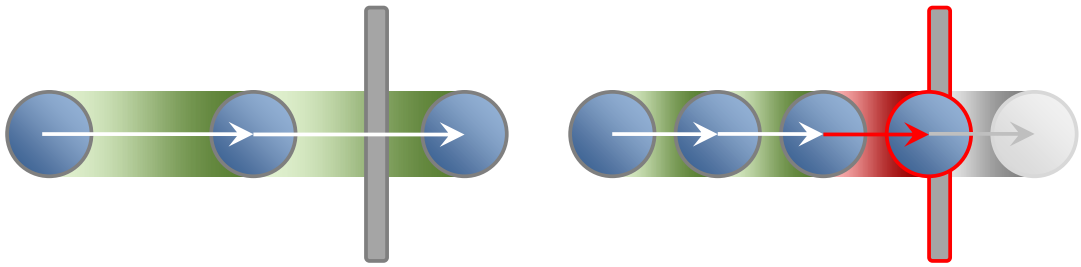

Figure 2.2: Discrete collision detection (DCD) can miss collisions due to tunnelling. Left) The ball tunnels through the wall without a collision being detected. Right) The time step is halved; resulting in smaller position updates such that a collision is detected. In this simple case, it is trivial to see that the ball cannot be allowed to move in a single step more than the sum of the ball's width (diameter) in addition to the wall's thickness to prevent any chance of tunnelling with DCD.

Continuous collision detection solves tunnelling by considering the continuous movement of objects in-between time steps, typically calculating a time of impact for each collision. The path of an object over a time step defines a volume, called its swept volume. The swept volumes of two objects do not intersect if there is no collision. If the volumes do intersect then further investigation is needed as they may or may not actually collide. It depends on the velocity and relative sizes, so the time of impact must be verified (see Figure 2.3). CCD is therefore more complicated and expensive than DCD; therefore often approximations are required for real-time applications. Swept volumes can be approximated by boxes, providing a less accurate fit but much cheaper intersection tests. Also, often linear movement over the time step is assumed, allowing relative velocities to be used easily, greatly simplifying the problem to that of a moving object against a static one. Similarly, objects can be

updated sequentially instead of simultaneously such that only the current object being updated is considered dynamic and all others are considered static using relative velocities. This allows objects to be simulated using different time steps and makes handling CCD for many objects in contact easier (also it is easy to reverse an update of a single object).

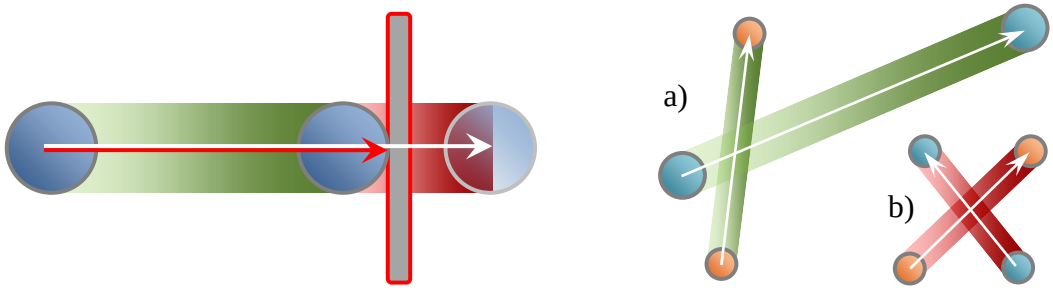

Figure 2.3: Continuous collision detection (CCD) can be performed between objects by examining the intersections of swept volumes. Left: CCD prevents the ball from tunnelling through the wall even though it moves large distances compared to the wall's thickness. Right: Intersection of paths does not guarantee a collision, it depends where the intersection occurs and the speed of the objects.

2.2.3 History versus History-Free Approaches

There is another consideration for collision detection, whether to use a history-based approach or a history-free approach. History-based approaches gain performance by using information from collisions at previous simulation steps (with an associated memory cost) often exploiting temporal coherence, and can make certain problem easier to solve such as which direction to resolve interpenetration of two surfaces (see Figure 2.4). One way to determine correct orientations is through voting, for instance Volino and Magnenat-Thalmann [VCMT95] forced all collision in the same region to have the same orientation according to the majority. Bridson *et al.* [BFA02] history-free approach applied repulsive forces between surfaces in close proximity to

reduce subsequent interpenetrations, but it had to be performed in a collision-free state to avoid interpenetrated areas being made worse (attractive forces can be used in interpenetrated areas [VCMT95, BWK03].) Selle at al. [SSIF09] noted that many opportunities for repulsions are therefore missed, since the cloth can move enough in a single step to generate interpenetrations without any causing any repulsive forces to be applied. Their history-based approach improved this and was able to incorporate history from the last collision-free state to determine pairs that they could then apply repulsions to at a finer granularity than the outer collision loop (i.e. also during time integration steps). They used the data to determine correct relative orientations and therefore correctly directed could be applied.

Continuous collision detection (CCD) does not necessarily imply a history-based approach, it can use information from the current step together with only predicted positions rather than detecting the collision after it happens. History-based approaches can fail seriously when errors are introduced from numerical precision issues, overlooked special cases or code bugs. Just as the history must be correct, assumptions must also be correct, often it is assumed there were no pre-existing intersections and that the system corrects all new intersections before advancing. However, one must consider what will result if the initial system configuration contains intersections and what if the resolution of a collision introduces additional intersections. The approach must be very robust or new collisions may not be correctly resolved or they may be missed entirely as the system progresses, leading to catastrophic failures such as a garment completely falling off of the character wearing it. For instance even with history data, Volino and Magnenat-Thalmann [VCMT95] approach required consistency checking and correction, since inaccurate response or complicated cases lead to incorrect orientations and therefore incorrect collision handling. The pinching of cloth is also such a case where history can be invalidated; this is discussed in Section

2.2.7. History-free approaches immediately have the benefit of being resilient to errors by being designed to handle nearly any previous state (depending on assumptions, e.g. intermediate collision-free state etc. but typically having no knowledge of the previous states); therefore they have a greater chance of recovery when things go wrong.

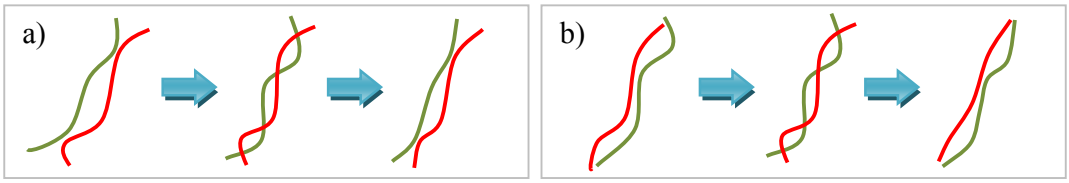

Figure 2.4: It is not always obvious how a collision should be resolved correctly; this figure shows such an example between two colliding surfaces (green and red). The intermediate state is identical for both a) and b), but knowledge of the previous state leads to two very different intersection-free states. The relative positions of the surfaces are maintained after collision resolution, a) green on the left and red on the right, b) red on the left and green on the right.

2.2.4 Collision Detection for High-Resolution Cloth

High fidelity offline cloth simulations are of little use and are severely limited without suitable collision detection and response. Bridson *et al.*'s [BFA02] work generates high quality wrinkles with complex collisions. By robustly processing collisions, contacts and friction, with a collision aware post processing step a surface is subdivided and iteratively smoothed for rendering. Their approach took approximately two minutes per frame for a square piece of cloth with 150 x 150 particles. Much work is focussed in the broad phase, reducing the number of elementary tests between adjacent primitives such as edge-edge, edge-triangle and triangle-triangle tests; in fact efficient collision detection is only achievable by reducing false positives as the performance of elementary tests themselves cannot be significantly improved further. Govindaraju *et*

al. [GKJ+05] presented a method that accurately detected all self-collisions for a 23K triangle cloth dress in 400-550 milliseconds. Their chromatic decomposition partitions a fixed topology mesh into independent sets, and uses a linear-time culling algorithm performing 1D overlap tests on the CPU and a 2.5D on the GPU, hence greatly reducing the number of elementary comparisons required. Curtis *et al.* [CTM08] achieved a 15 times reduction in the number of elementary tests and a 4.9 times speed increase. Their example of a dress worn by a flamenco dancer (26k vertices, 75k edges and 50k triangles) took 200ms to calculate all self-collisions and collisions with the character. They employ what they call "representative triangles", standard triangles that have been augmented with mesh features information (a feature is a vertex, edge or face used in elementary tests). Feature-based hierarchies which use three separate bounding volumes (one for each feature type), eliminate duplicate elementary tests and improve culling performance at the increased cost of memory (4-7 times more) and increased maintenance and traversal costs. Their representative triangles, allow the use of a single bounding hierarchy while retaining the benefits of feature-based hierarchies, an improvement compared to Hutter and Fuhrmann's [HF07] work which used multiple trees.

Tang *et al.* [TCYM08] introduced the concept of "orphaned sets" to remove nearly all redundant elementary tests between triangles achieving one order of magnitude speed up used with a bounding volume hierarchy, by employing connectivity based culling. Previous self-collision methods based on normal cones were limited to discrete collision detection; they extended them for continuous collision detection. They were able to reduce the number of false positives by 38 times for a high resolution piece of cloth (46k vertices and 92k triangles) draped over a rotating sphere with many self-collisions. It took on average 290ms to perform the CCD. Their approach is general and also handles breaking of objects or n-body simulations, but performance

is reduced with the loss or lack of connectivity information.

Cloth is not often simulated using high resolution meshes even off-line, typically less than 100,000 elements and far less for real-time simulations. Often we strive to use the fewest number of polygons that will generate an acceptable quality of simulations to save processing time. However, interestingly Selle *et al.* [SSIF09] used much higher resolutions producing impressive high fidelity simulations. Using meshes of up to an incredible 2 million triangles importantly with a robust history based collision framework that accurately handles friction, objects and self-collisions. Simulation times were on average between 6 and 45 minutes per frames for different simulation set-ups. It would be interesting to see how these off-line methods scale for much lower resolution cloth; it may be there is too much overhead where there will be far less false positive to prune away.

2.2.5 Impact Zones and Fast Moving Cloth

Fast moving cloth is a particular challenge, not only necessitating the use of continuous collision detection as tunnelling becomes severe but collision response becomes problematic. Penalty based approaches may not provide sufficient force to prevent collision with fast moving cloth. So often cloth velocities are averaged in regions of collision and the collision is resolved as if the cloth were rigid in so called zones of impact or impact zones (IZ). Impact zones were originally proposed by Provot [Pro97] for handling self-collisions but its definition was extended by Huh *et al.* [HMB01] to include collision against other bodies. Over time, clumps of rigid regions form and are merged together; it can be difficult for them to separate. Bridson *et al.*'s rigid impact zones (RIZ) [BFA02] improved upon [Pro97], preserving linear and angular momentum while including cloth thickness using proximity constraints. A few iterations of local impulses were used first before switching to the RIZs. They state that

their repulsion forces and initial collision impulses keep the zones small, isolated and infrequent; while tending to separate zones after they are formed helped by the cloth's thickness. Huh and Metaxas [HM06] presented an algorithm for collision resolution of clump-free fast moving cloth with self-collisions, taking 1-4 minutes per frame. The ordering is important; they determined that self-collisions must be resolved before rigid-cloth collisions, and that penetrations must be resolved before proximity constraints to minimise clumping. Collisions are grouped into impact zones, and edge-contacted cloth nodes, called collision clusters are resolved simultaneously with fewer worries of clumps forming. Later, Harmon *et al.* [HVTG08] achieved free-flowing motion even when cloth was forced through a narrow funnel, pushed through by a ball, generating complicated collisions. They presented an alternative fail-safe that instead of making regions rigid, they only cancelled impact motion but not sliding motion so that there is less artificial dissipation, replacing the "rigid impact zone" step of Bridson *et al.*'s algorithm [BFA02]. The cost per frame was 9.61 seconds per frame (compared to 18.81 seconds for the rigid impact zone) on a complicated funnel example with 16569 degrees of freedom.

2.2.6 Sharp Features

Objects with sharp features present many challenges for collision detection, since sharp features 'poke' through the cloth and causing it to get stuck. Fuhrmann *et al.* [FSG03] noticed this problem when using distance fields and only evaluated particles against the field. To combat this they introduced collision tests on the centre of edges and applied a correction to its two end particles if needed, for efficiency only edges where both particles were in close proximity to an object need be tested. Wong and Baciu [WB05] presented a dynamic collision detection approach for deformable surfaces against non-smooth objects, the deformable surface is partitioned into a finite

set of surfaces effectively pruning a large number of non-colliding areas. Collisions are resolved by constructing a penetration free motion space for particles and keeping the velocity within the motion space such that currently colliding pairs will not collide in subsequent motion. They present a number of examples of cloth colliding against very rough rigid surfaces with many spikes without penetrations.

2.2.7 Pinching of Cloth

There has been research into handling pinching of cloth between surfaces, such as with garments pinched under the arms or between the legs. The main problem is caused by object meshes self-intersecting, and therefore there is no correct way to resolve a collision for cloth trapped in these self-intersecting areas. This typically defeats history based approaches, and invalidates assumptions of the previous state being collision free. Baraff *et al.* [BWK03] presented an offline approach to untangle clothing with many self-collisions using a history-free approach and clever handling of these pinched regions. Intersection curves were found between the cloth mesh and another mesh (e.g. the character, or with the cloth itself) and then the meshes were coloured on both sides of the curves using a flood fill algorithm. The smallest coloured areas are taken to be intersection areas; this works well for small updates where collisions are not left to become so large that the wrong side is chosen. Attractive forces are applied between intersection areas and repulsive forces to non-intersecting areas to resolve a collision. They handled pinching by attaching cloth particles to the surfaces with weights when in close proximity of more than one surface allowing relative motion of the pinching surfaces, while tending to position the cloth half way between two stationary surfaces. Their approach with a garment with 18K vertices increased costs by less than 0.5 seconds per frame on a 2 GHz CPU, which they say is negligible for their offline simulation but obviously far too expensive for real-time

work.

2.2.8 Real-Time and Interactive Approaches

Prez-Urbiola and Rudomin [PUR99] presented a way to model multi-layer clothing on articulated implicit characters using a scalar distance field produced from ellipsoids. Isosurfaces were used to place garments into one or more independent layers. Cloth vertices are moved towards their assigned isosurface using the scalar field, together with a mass-spring network to give the garments shape. Interestingly as long as the garments remain locked to their different isosurfaces, no inter-garment collision detection is needed since their respective isosurfaces do not intersect one another. Rudomin and Meln [RM00] extended this approach to work with standard skinned characters; however the calculation of the scalar fields remained a serious bottleneck. Subsequently Rudomin and Castillo [RC02] achieved real-time performance but by no longer supporting multi-layer cloth. They still used a hierarchy of ellipsoids to approximate a virtual character, but made use of distance calculations instead of implicit isosurfaces for collision detection. Mass-spring particles move with the ellipsoids as the character is animated, and then they are allowed to move freely using physics simulation step while correcting any penetrations with the ellipsoids. Meng at al. [MMJ10] also used ellipsoids but with only a static character used for virtual try-on software for clothing.

Armless mannequins have been represented by contours for Garment CAD [MK05], a cloth particle that is between two contours is checked for collision by creating an interpolated contour at the height of the particle. They resolve collisions by reversing the velocity of a particle and damping it with a frictional force but the position is not corrected but instead rolled-back. Rapid collision detection was achieved with distance fields for cloth against rigid objects [FSG03], they use a fixed distance envelope

around objects for which they compute the field. Trilinear interpolation is used to reconstruct the distance any point, but to reconstruct the normals they say it is faster to analyse the gradient of the field. Collisions are resolved inelastically with added friction but exact intersections are not computed for performance reasons. Decaudin *et al.* [DJW+06] also used a precompiled distance field around a mannequin for collision detection for virtual garments, and moved particles outside in the direction of the distance field. Distance fields are problematic for animated or deformable objects; this is because pre-computation of the fields is costly in both CPU time and memory. Geometry images capture geometry (meshes) in a 2D space as an image of (x, y, z) coordinates, additional information such as surface normals or colour can be stored as additional images. Zink and Hardy's [ZH07] novel approach employs geometry images for the simulation and collision of cloth. Static objects are contained in a unit cube surrounded by a bounding sphere, in pre-processing their coordinates are converted into polar coordinates centred on the sphere stored in a square array indexed by the two angular components. It can contain more than one entry for each array position sorted by distance to the centre; additionally triangle interiors are rasterised so that there are no gaps in the array. The collision detection is performed by firstly checking cloth particles against the bounding sphere, and then each particle is tested against the array in polar coordinates. In this way, the array can be used as if it were a distance field.

Cordier and Magnenat-Thalmann's [CMT02] real-time clothing approach was only to perform collision detection with only floating parts of the clothing, skirts were tested against the character's leg which were approximated simply by cylinders. They pre-computed a potentially colliding list based on distance to the legs and normal orientations of vertices on the skirt, then during the simulated only these vertices were tested and the response was to kinematical correct their position and velocity.

They later [CMT04] employ pre-computed collision hulls created from offline pre-simulated cloth, relying on sufficient variation and range of motion that the hulls encompass all positions required during the runtime simulation. Again, only floating or loose areas of cloth are tested for collisions, the collision hulls greatly reduce the area of intersection needed to be tested against allowing real-time performance of 31-74 fps with different coarse simulations although separate collision timings were not given. Their pre-processing step took around 8 minutes for each simulation, excluding the offline pre-simulated cloth.

Achieving real-time performance puts tight limits on the complexity of the cloth and collision testing and approximation is a recurring theme for much real-time work. For instance, Fuhrmann *et al.* [FGL03] achieved interactive animation of cloth with self-collision by approximation only; it considered pairs of particles and held them apart using a bounding hierarchy of particles. A 1000 particle square mesh ran at 30 Hz allowing a user to drag it around and over a table top. Andrews [And06] used a rigid-body physics engine to simulate a small square piece of cloth at 40 fps. The cloth was approximated by a bounding sphere at each vertex. Such approaches are hard to work with, if the spheres are too large, the cloth will appear to hover above surfaces or if they are too small other objects could pass through the cloth.

Vassilev and Spanlang [VS00] used image-space interference tests for collision detection for the dressing of virtual characters using depth and normal maps. Subsequently, Vassilev *et al.* [VSC01b] used their image-space collision detection for clothes on an animated virtual character, at the time allowing animations to be produced at 3-4 fps. Front and back views are rendered, and the GPU is used to create two depth maps, two normals maps and two velocity maps for the character. The maps are transferred to the CPU's main memory and the cloth is tested against them by converting cloth vertex positions into the map coordinate system and then the

information from the maps is used to resolve each collision.

There has been other work using the GPU but also based on traditional techniques, such as Greßat al. [GGK06] presented real-time collision detection (including self-collision) for deformable surfaces composed of NURBS (Non-uniform rational B-spline) patches using a bounding volume hierarchy. Importantly there was no read-back needed to the CPU from the GPU, they demonstrated it on a simple deformable flag being hit by small balls running at 25 frames per second.

2.2.9 Multi-Core Collision Detection

Multi-core collision detection is proposed to accelerate the computation but it is not easy to achieve good scaling performance. It is difficult to divide tasks efficiently between cores; loads are not predictable and depend on the simulation and current collisions. Therefore, exploiting temporal coherence is important. Tang *et al.* [TMT09] presented a parallel algorithm for continuous collision detection between deformable models that does exploit temporal coherence. They maintain a "front" that records information from the last simulation step and updates it to check for new collisions in the current step. However, their incremental approach requires a large amount of memory and did not achieve a linear speedup. Running time was reduced to 18.94% with 16 cores compared to a single core, 37.34% and 24.06% with just 4 and 8 cores respectively on 2.93 GHz CPU. Calculating non-adjacent pair collisions was a particular bottleneck, not scaling past 8 cores (only reduced to about 40%); nonetheless interactive collision detection is achieved for a cloth-ball benchmark, taking 72ms per frame. Subsequently, their improved approach [TMT10] scaled much better and was able to achieve 7 and 13 times speed up for 8 and 16 cores, taking between 5.3ms to 32.5ms in benchmarks with 4k to 92k triangles. They use a 2-3 (balanced) tree

instead of a binary tree, they present adaptive strategies to decompose the computation in to sub-tasks while minimising synchronisation overheads and using a cache friendly layout.

2.3 Levels of Detail for Cloth Simulation

The use of Levels of Detail (LOD) techniques are common place in Computer Graphics. They are particularly used for rendering that exploits view-dependant criteria such as progressive meshes [Hop97] with criteria based on the view frustum, surface orientation and screen-space-geometric error. Techniques such as these allow great increases in performance for rendering, enabling large detailed scenes to be rendered at real time frame rates. LOD approaches has been extended to simulations, so called Simulation Levels of Detail (SLOD) [CH97] used subsequently, for example in multi-agent control [BH02] and plant motion [BK04]. Also the adaptive simulation of 3D deformable bodies have been researched [DDCB00, CHCK02, CHCK05]. LOD has been introduced for cloth simulations by way of multilevel and adaptive meshes. The geometric detail needs to be high in areas where the cloth is folded or wrinkled to ensure a reasonable approximation to the ideal shape of the cloth; this increases the computation time. In planar pieces, the shape can be represented faithfully by less polygons and per-pixel lighting provides an illusion of a smooth surface. Hence curvature is commonly used as a criteria to trigger increases in level of detail in cloth, for example in [HPH96, LV05, VB05].

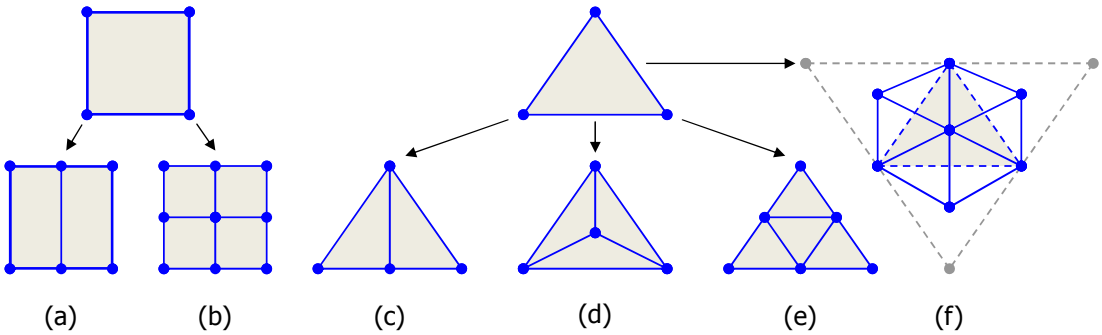

Figure 2.5: This figure shows various subdivision schemes for quads (a,b) and triangles (c-f).

Subdivision of quads and triangles are common place for rendering, multilevel and adaptive meshes can use similar subdivision or refinement schemes. In this work we will refer to the act of applying subdivision to a mesh as refinement and hence the reversal of it as coarsening. We express the number of times a polygon is subdivided by its refinement level. A polygon (or mesh) has a refinement level of zero before any subdivision, each time it is refined (often recursively), its refinement level is increased by one. A uniform refinement is when all polygons in a mesh are refined to the same level; conversely a non-uniform refinement therefore contains polygons from many different levels. Figure 2.5 shows various subdivision schemes for both quads and triangles. The regularity of a mesh is often a concern for accurate cloth simulation; the mesh's geometry determines the cloth's degrees of freedom and the ways in which it can bend. For instance, if using the simple bisection schemes (Figure 2.5 (a) and (c)) we must subdivide along the longest direction each time otherwise long thin quads and triangles would result which only allow increased bending in one direction. Subdividing triangles by inserting a new central point leads to two schemes; (d) is simple but produces an irregular pattern, the valence of existing vertices in the mesh are continually increased each level. Valence is important for performance, many costs such as vertex lighting normals and vertex mass calculations are proportional to the number of adjacent faces. The other scheme, $\sqrt{3}$-subdivision [Kob00] (f) is preferred over (d) and does not suffer from this problem (note: it could be obtained by flipping the original edges of (d) after subdivision). Regular subdivision of quads (b) and triangles (e) into four is also popular, it also increases resolution faster than $\sqrt{3}$-subdivision at each level (four times compared to three). Faster refinement may or may not be desired and it can be likened to tree structures where depth is often an important consideration, slower refinement means that more levels of refinement are required for a similar increase in resolution.

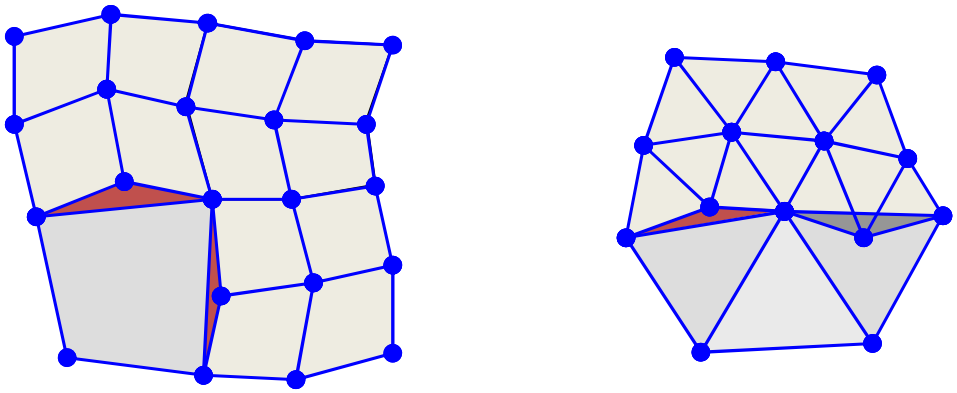

Figure 2.6: T-junctions result from refining deformable polygons in isolation, leaving holes between adjacent subdivided and non-subdivided polygons. Left: Three out of four quads are subdivided into four. Right: Three out of six triangles are subdivided into four.

A challenge with using subdivision for cloth modelling is dealing with adjacent geometry, leading to so called T-junctions. T-junctions if left can manifest as holes in the cloth, for example if the new vertices are free to move in Figure 2.6. This problem is discussed further in the following text as each author has their own way of generating a conforming mesh suitable for cloth simulation.

2.3.1 Multilevel and Multigrid Methods

Subdivision methods and simulations that perform or use uniform mesh refinements are commonly referred to as multilevel or multigrid methods. However, the terms multilevel and multigrid can lead to confusion because they have been used by different authors to mean different things. At their heart, one can imagine that a coarse mesh can be recursively refined to give a range of levels or grids, i.e. multilevel or multigrid.

Firstly, we cover the less commonly used meaning of multi-level methods, which have been previously used to accelerate draping simulations rather than full dynamic cloth simulations, suitable for when the cloth has a final resting position. It is an

interesting LOD technique that is based on elapsed time rather than traditional criteria. Zhang and Yuen's [ZY01] fast cloth draping simulation used this meaning of multilevel meshes, by dividing the process into several phases. The simulation starts with a coarse mesh, which is fastest to simulate. It is then subdivided after each phase, thereby increasing the detail and cost of the whole mesh where the mesh is closer to its final resting position. The time chosen to enter the next phase is crucial; once the coarser mesh has reached its balanced position, the final shape could not be noticeably improved by further subdivision. On the other hand, advancing phases too quickly will cause the performance gains of this method to be lost, so a balance must be achieved.

The more accepted meaning is from the field of numerical analysis where multigrid methods have been developed to solve differential equations using a hierarchy of discretisations, i.e. discrete levels of detail. They are designed to accelerate the convergence of iterative methods by solving a coarser problem and using the coarser levels to propagate global information to the finer levels. Such methods are commonly applied to engineering problems such as electromagnetic field modelling and fluid dynamics but have also been used for cloth simulation with a hierarchy of meshes. Oh *et al.* [ONW08] presented an efficient multigrid algorithm for cloth simulation with implicit integration; it was about four times faster than the preconditioned Conjugate Gradient method. Starting from a coarse mesh, multi-resolution meshes were generated and they were optimised for improved performance by making the triangles equilateral and uniform, all having the same area. Their algorithm was demonstrated on clothed characters, achieving from 3 to 10 frames per second and so allowing interactive garment prototyping of garments with up to about 10k particles.

Multigrid methods have also been used to accelerate very high density meshes; Lee *et al.* [LYO+10] presented a novel offline multi-resolution method to efficiently

perform large-scale cloth simulation. Starting from a detailed mesh, they dynamically simplify smooth regions and then reconstruct the full solution to the linear system by interpolation. Their simplification metrics consider smoothness of stretching, shear and bending forces, velocity, as well as collisions. The advantage of this method is that it achieves large performance gains for little loss of simulation quality. In their example, the cost for simulation of a character wearing a dress was reduced from 23,055ms with it's the 240k triangle mesh to 2745ms using their simplification and reconstruction technique. It would be interesting to know if this approach could scale to much lower resolution meshes and still remain efficient.

Hardware has moved towards the increased use of multi-core processors where data sharing becomes complicated, since simultaneous data writes must not be allowed. Lario *et al.* [LGPT01] presented a multilevel cloth simulation but used OpenMP to accelerate it and also compared its efficiency to MPI. OpenMP is an industry-standard API (application programmer interface) for shared-memory parallel programming [DM98]. MPI stands for Message Passing Interface, and it is a language independent communications protocol that is used to program parallel computers [GLDS96]. In Lario *et al.*'s [LGPT01] work, over 70% of the time was spent computing the finest level scaling almost identically for different square grid sizes of 128, 256, 512 and 1024. The parallel efficiency remained above 90% for 4, above 80% for 8 cores and above 40% for 16 cores for both OpenMP and MPI. In some cases OpenMP performed better than MPI and always better on one particular IBM processor, however, OpenMP's main attraction is described by the author as an "effortless way to get a parallel version of the code". MPI on the other hand requires much more effort to use but achieves better efficiency for large problem sizes using SGI and SUN processors.

2.3.2 Adaptive Meshes

Adaptive meshes are used for cloth simulation, in a similar spirit to progressive meshes or tessellation for rendering but they have wider uses for rendering, simulation and collision purposes. The main difference between them and multilevel or multigrid methods is that the local refinement is performed during simulation on a single adaptive mesh. The mesh complexity is dynamically increased (refined) and often can be decreased (coarsened) to balance quality against computational cost during runtime. Adaptive approaches typically aim to exploit the variations in shape (mainly curvature), to allow faster computation without sacrificing the detail to the same extent as using a coarse mesh all of the time. Ideally, the simulation will appear to the user to be the same if a detailed mesh or the adaptive one is used, resolution independence is therefore an important consideration.

Adaptive Meshes for Collision Detection

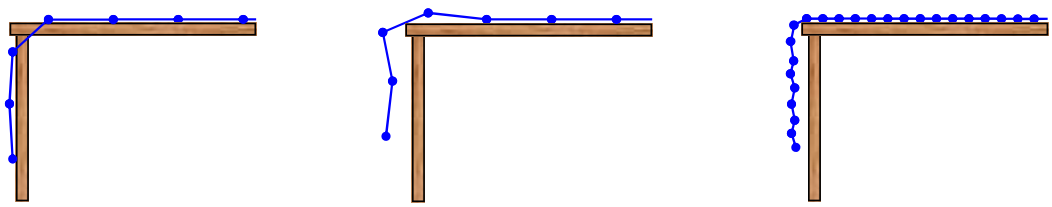

Figure 2.7: Coarse meshes have difficulties with resolving collisions; they may not have enough resolution to faithfully approximate the underlying surface. The figure shows a cross section of two coarse meshes (Left, Middle) and finer mesh (Right) draping over the side of a table. Left: Vertex only collision detection causes edges to intersect the table. Middle: Collision detection is performed with the edges also, but the mesh exhibits pivoting on the edge on the corner of the table, causing the cloth to not sit flat on the top of the table near the corners. Right) a finer mesh with full collision detection can greatly increase the ability to faithfully represent the underlying surface.

Coarse meshes can experience problems in conjunction with collision detection;

they may not be enough resolution to resolve intersections in a pleasant manner. For example, a coarse table cloth may not be able to bend at just the right point over the edge of a table, leading to a number of problems depending on how the collision was resolved, see Figure 2.7. However, the increased cost of refinement on simulation costs may be prohibitive and lessen the problem rather than removing it altogether. Previously, non-active points had been used for correct collision handling by Howlett and Hewitt [HH98]; savings were realised by not having to simulate those points. The non-active points are placed half way along edges in a quad mesh and also in the centre of each quad. They claimed that the constraint of a uniform mesh only being able to bend along predefined lines was effectively removed and the cloth could bend as required for collisions. The mesh was able to drape over edges of objects accurately although there is noticeable stretching as a result of the non-active points being moved to resolve collisions. They concluded that the edges' rest lengths must be modified or the non-active points must influence the simulation in-order to overcome the stretching; however they did not resolve this. Etzmuß *et al.* [EEHS00] presented a cloth particle system that adaptively generates new particles only when they are necessary specifically for the correct handling of collisions. This allowed them to perform fast and accurate simulations with coarse meshes. Their approach was better, the virtual points applied forces to the mesh via springs but they were not completely integrated into the simulation, their position is only valid for a single time step as it is calculated from the edge collision itself so they can be positioned exactly where needed. More recently, Sifakis *et al.* [SSIF07] presented a hybrid simulation of deformable solids. Their framework allowed embedding arbitrary sample points into a mesh for handling collisions, plasticity and fracture without the need for complex re-meshing although it was not accomplished in real-time. Hard bindings are introduced and constrained by barycentric coordinates of their location on a triangle in the mesh,

having no degrees of freedom they redistribute forces applied to their parents and are used to deal with T-junctions. Soft bindings connect additional simulated particles to hard bindings or even original particles of the mesh enabling two-way interaction between the particle-based system and the mesh-based framework.

Simulation with Adaptive Meshes

Hutchinson *et al.* [HPH96] presented the first adaptive method for mass-spring simulations; it was designed to resolve the guess work require by animators to select a suitable resolution for cloth animations. The method uses a quad mesh that is uniformly refined recursively to level 2, but for performance it uses a flattened data structure where the coarse mesh is refined to the finest level before simulation begins at the cost of memory. At the start, particles corresponding to coarsest level are active, and all others are non-active. Refinement is triggered for active particles if the angle across them between two connected springs (in either direction) exceeds a user defined tolerance; the simulation is stepped backwards to be repeated after refinement. Eight non-active particles are activated and then the simulation step is repeated and then continues until another refinement is triggered. T-junctions caused by the non-active points are handled by interpolation, i.e. fixing them to the centre of edges. The method produces a good speed up of cloth draping particularly as collision costs were large, but did not feature coarsening. An example of a piece of cloth (73x73 at the finest level) draped over a cup took at the time around eighteen hours for the full simulation but only one hour for the adaptive one.

The use of adaptive methods to simulate cloth introduces problems which are particularly prevalent when the mesh deforms. Care is needed to preserve the cloth's physical properties to avoid visually distracting artefacts around the subdivision seams. Furthermore the physical properties may be different from a uniform mesh.

Volkov and Li [VL03] found that fine wrinkles observed in the non-adaptive simulation were missing from their adaptive one, these fine wrinkles were attributed to buckling behaviour which cannot be detected by their curvature based criterion. It is easy to only focus on in-extensibility and in-plane forces, but bending can be an important part of cloth simulation for realism. The potential for sharp creases to become even more severe has also been noticed [LV05]. Villard and Borouchaki [VB05] presented an adaptive method to allow the mechanical system to behave without any constraint related to a uniform mesh e.g. resolution independence. New points are added to the mesh based on local curvature at that point. Four elements (quads) are subdivided into sixteen around the point creating eight new active nodes and eight virtual nodes. This becomes more complicated at successive refinements, some virtual nodes become active and the number of new nodes varies in order to preserve the warp and weft structure. Their adaptive and uniform simulation of cloth draped over a sphere produced very similar results as shown by a superimposed image of the two final meshes together, nonetheless the simulation time is improved by as much as six times by the adaptive mesh. However, a major limitation of the approach is that the refinement cannot be reversed; and as such this method cannot be used for clothing on animated characters. After much movement the mesh will eventually be fully refined resulting in no computational savings at all.

Mujahid *et al.* [MKM+04] effectively used OpenMP to implement an adaptive cloth simulation with both refinement and importantly coarsening using a quad mesh. So called 'ghost' particles needed for handling T-junctions were not simulated. In the work, load balancing was an important feature and improved simulation times by up to 32% when running on 8 cores even without considering cache memory coherency. Timings for 8 cores were 6.3 times faster than a single core, taking approximately 11.1 seconds for 20 iterations. The simulation is slower for early iterations, where a high

density mesh is used to resolve collisions and is then coarsened in flat areas as the simulation progresses. It would be advantageous to simulation times if a coarse mesh could be used from the onset, possibly using collisions as an additional refinement criterion.

Quad meshes are relatively easy to subdivide and to handle adaptation of the simulation parameters to achieve resolution independence, but are normally only demonstrated with square pieces of cloth. It is less simple for semi-regular triangle meshes, but they make it easier to create complicated shapes needed for clothing without becoming too irregular hence triangle meshes are commonly used for clothing. For instance, Li and Volkov [LV05] applied the $\sqrt{3}$-subdivision rule to clothes on an animated character. A conforming mesh was extracted from an adaptive hierarchy, with adaptation taking 87ms and the simulation taking 1.2 seconds on average per step for 12k triangle adaptive meshes. Their adaptive simulation used more slightly more triangles than the non-adaptive one but was of higher quality. It is rare to see adaptive meshes applied to clothing, most are demonstrated on square pieces of cloth such as Wang's [Wan02] earlier adaptive method which employed a triangular mesh. The method featured local refinement based on the curvature between adjacent triangles but a triangle is only refined using a 1-to-4 split if the curvature across all three of its edges is sufficient. There is no retriangulation and T-junction vertices are fixed with a force propagation model. It shows good resolution independence using implicit integration, however, unfortunately no performance statistics were provided.

An interesting and more generic method for incremental mesh adaptation which supports any triangle refinement rule such as 1-to-4 split or $\sqrt{3}$-subdivision was presented by Volkov and Li [VL03]. Incremental methods perform only small changes in a single update, such that the local level can only increase by one in the single step. It leads to a smooth transition between highly refined areas and coarse area.

Volkov and Li's method is based on a hierarchy of semi-regular meshes; it requires the construction of a conforming mesh to bridge different layers built from the triangles in the highest available resolution (this is more complicated than simply dealing with T-junctions, depending on the refinement rule different levels may not share any common vertices and vertices may be placed outside the original boundary of the mesh and need to be moved). However, they state that they were able to perform direct updates to the conforming mesh alleviating the need to rebuild it each frame. Their mesh was used in three applications, a physically based simple cloth simulation, real-time terrain visualisation and procedural modelling.

Large Adaptive Meshes and Out-of-core Storage

Large adaptive meshes can present additional challenges to remain efficient. The memory architecture of the computer must be taken into account where out-of-core storage is needed. Large multiscale terrain models in [VL03] exploited the subdivision connectivity of the adaptive mesh by storing related data in each data block, their hierarchical storage layout out performed linear storage by more than an order of magnitude. Typically only each 100th data request resulted in disk access. Cignoni *et al.* [CGG+03] faced similar problems in their approach for the interactive rendering of planet-sized textured terrain surfaces using an adaptive mesh, they used prefetching to hide disk latency when using out-of-core storage where less than 2% of their 4.7 GB data set is loaded into memory at any time. The surfaces are made from adaptive triangle patches suitable for batched rendering and they use right-angled triangles such that a patch connects correctly to patches on the next coarser level, the same level and the next finer level without any T-junctions. RAM is plentiful in modern computers and CPU's have large caches; for real-time or interactive cloth simulations the burden on computational resources is the limiting factor and out-of-core storage or memory usage is not normally a problem. However, it is important for the film

industry, where they require the processing power of large servers and server farms for modern block buster movies featuring many digitally created assets including offline simulated cloth.

Adaptive Meshes for Rendering

An interesting alternative to adaptive simulations, but highly related, are subdivision surfaces and tessellation methods where the surface of a mesh is subdivided, typically to increase the detail for rendering and lighting without imposing significant costs on storage and animation for large meshes. This can therefore be applied to coarse cloth simulations to improve the visual quality without increasing the cost of the simulation. Curved PN-triangles [VPBM01] are a good example of a generic method that can be performed in hardware and is applied to triangles. Three sided cubic Bézier patches form each curved PN-triangle, coefficients are calculated such that neighbouring patches match together perfectly without the need for adjacency information. Each patch is specified by only the triangle's three vertices and their normals. Lorenz and Dllner [LD08] presented a dynamic refinement technique using geometry shaders on the GPU with an incremental multi-pass scheme. Boubekeur and Schlick [BS05] presented a generic method for uniform mesh refinement on the GPU using a tessellation pattern and barycentric interpolation but could not support adaptive refinement. Procedural normal and displacement maps transform vertices' positions and normals to apply a variety of different refinement techniques including PN-triangles. Later Boubekeur and Schlick [BS08] presented a flexible adaptive refinement method also on the GPU that used stored refinement patterns which were looked up by coarse vertex 'depth-tags' (which specified refinement depth). GPUs are now starting to feature built in programmable tessellation units available through DirectX 11 and Opengl 4.0, often used with displacement mapping to render detailed objects from relatively simple geometry.

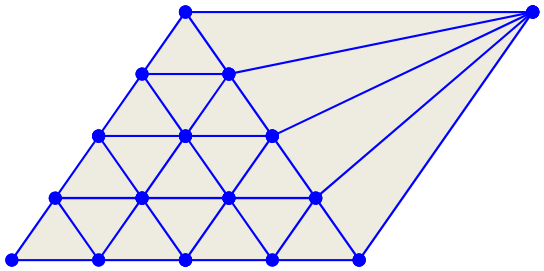

Figure 2.8: A simple way of dealing with T-junctions or non-conformity between adjacent polygons of different refinement levels is to split the coarser triangle into potentially very long thin triangles. This is acceptable for rendering (while the triangles are not near being degenerate), but will cause artefacts with deformations and simulations with mass-spring networks.

Cloth is not often considered by itself; typically the simulation involves complex interactions with other objects which occur as a result of collisions. Therefore, one possible difficulty is that the rendered surface may intersect the object even if the coarse cloth mesh does not. However, many fast GPU methods for rendering cannot be easily integrated with collision detection, since no actual mesh is refined but only passed out on the fly. If the actual mesh is topology updated, it could be used for simulation but rendering methods do not always deal with T-junctions in the best way for simulation. Often T-junctions are resolved simply by long thin triangles, see Figure 2.8. These long thin triangles if simulated will favour bending along their edges' directions; as well introducing irregularity.

Xia *et al.* [XESV97] has previously incorporated lighting into their adaptive real-time polygonal rendering approach, preserving the detail of specular highlights based on view direction and visibility. The case for additional adaption of the cloth purely for rendering is small; there is a good overlap between the criteria suitable for rendering and simulation. Furthermore, modern graphic cards capabilities for anti-aliasing

and per-pixel lighting improve visual quality for coarser meshes. There has been research into human perception for LOD methods. McDonnell *et al.* [MDCO06] evaluated different LOD representations of characters wearing cloth that had been physically simulated off-line. They showed that impostors work very well for these as well as they do for characters clothed by standard skinning methods. Hoppe's [Hop98] work on progressive meshes identified difficulties with view-independent LOD meshes, with many faces lying outside of the view frustum or being back facing incurring a cost for rendering even when culled early in the pipeline. This is not such a problem for cloth as non-visible regions are still required and interact with the visible sections; however, it would still be beneficial to reduce their cost.

2.4 Virtual Clothing and Garment CAD

This section explores the application of previous work for the design and simulation of virtual clothing and garments related computer aided design (CAD) in the area. This work can be valuable to the textile industry for prototyping garments and even for bioengineering such as for the thermal performance of clothing [MLW+08], but we focus on only the computer graphics aspects in relation to CAD. The simulation of clothing is much more involved than the simulation of square pieces of cloth; techniques from many areas must be brought together. Firstly garment patterns or meshes must be designed, created and dressed on the character before simulation can begin. Virtual characters must be created, rigged and animated together with efficient collision processing techniques to allow the characters to wear the garments. A recent survey on CAD methods in 3D garment design can be found here [LZY10], and another focusing on 3D CAD systems specifically for the clothing industry [SKC10]. Dedicated CAD software can be found in the textile industry for the designing of textiles, but custom software or 3D modelling packages such as Maya or 3DStudioMax are typically used in the field of computer graphics.

2.4.1 Garment Creation, Seaming and Dressing

In the real world creation of items of clothing, flat panels of fabric are typically cut into shapes and stitched together. For example a very simple T-shirt may be formed from two pieces of fabric that are sewn inside out, then it is turned right-side out with the seams hidden on the inside. Creating garments from 2D flat panels can make it easier to simulate, because it can provide undeformed lengths between points that can be used for spring rest lengths. It is possible to instead create (3D) all in one garment meshes, but these lack the easily comparable point of reference of 2D panels. The initial 3D state can be used for example to provide the rest lengths of springs

for a mass-spring system. Since we are also typically representing a curved surface with planar polygons in 3D models but shading them as if they were smooth then the rest lengths may not be accurate. Real cloth is a developable surface, so clothes can be unsewn, taken apart and flattened back into panels without distortion. Directly created 3D models are unlikely to be developable, it is noted in [DJW+06] that they therefore depict garments that are not physically plausible and are unlikely to be able to depict fold patterns specific to real garments in any rest position. We would also like to note that sometimes the important task of garment creation, seaming and dressing are not even mentioned by some authors, for example no details were given for the garment featured in [LV05].

Seams are visible on real garments but of course even if virtual garments are created using flat 2D panels, it is not necessary to have visible seams. Meshes can be joined together as if it was a single mesh. For instance Durapinar and Gudukbay's [DG07] virtual garment design and simulation system featured automatic pattern generation from a convex hull of boundary points. A mass-spring system was used to bring the seams close together by combining vertices into one by adding each vertex's total spring forces to the others, seaming took 4.313 seconds for a skirt with 1400 vertices.

Although not necessary to model visible seams, seams change the behaviour of real cloth and the visual appearance can be important for realism. Ma *et al.*'s [MHB06] method works using 3D irregular meshes with no seams, but then automatically constructs them along an arbitrary path on the surface of the mesh. They created seams using multiple layers of cloth (5, 3 and 4 layers for their three example seam types); the seam line is cut flush and then the additional cloth needed to model the seam is created rather than folding (and therefore shrinking) the existing cloth. They were able to simulate very detailed seams and realistic results with pucker

(the distortion of fabric along seams causing a rippled appearance) using a mass-spring network which was demonstrated on a pair of trousers with around 18.5k vertices and 35k triangles. The approach is only suitable for offline work; it requires fine meshes to model the seams making the collision costs particularly expensive. Pabst *et al.* [PKST08] looked at the influence of seams on the bending of fabric, together with an accurate bending model. If using a relatively coarse mesh, they found that it must be locally refined around the seams to smooth out the abrupt changes that would otherwise be present in the bending stiffness between adjacent elements, thus increasing the costs. However, while comparing a real garment with that of a simulated one, they described it to match "many aspects of the real garment very well" which was backed up by a visual comparison. In their simulation the bending forces alone took 65ms to calculate which was more expensive than two previous models by other authors.

It is not easy to animate a character such that it can get dressed as a human would, but we must be able to dress a garment onto a character somehow (i.e. pre-position it prior to simulation). A simple method is to simply place the garment at the best place over the character, with the character in a reference pose such as a T-pose which is common for rigging. Collision detection would then need to fix any interpenetration, but it is difficult to produce good results since different poses of the character and different positions of the clothes might be incompatible. Often dressing is performed at the same step as sewing [VT97, Vas00, PLAMT02, DG07], the panels are placed around the garment and are simulated with constraints or springs to bring them close enough to be joined. Users of the systems are responsible for rotating and positioning the panels to be in the best place to fit around the character. In response to this time consuming burden on the user, automatic positioning has been proposed [GFL03, FGLW03, LTG05]. Fuhrmann *et al.*'s [FGLW03] method employs bounding

surfaces fitted around body segments to which cloth pieces can then be automatically mapped to. Some additional data is needed including which side if visible (outward facing) after dressing, the assigned body segment, a direction vector relating to the bounding surface and a feature point. Their method takes less than 0.3 seconds to preposition various simple garments onto 3D body scans. Their approach was extended by Groß*et al.* [GFL03] to enable a body to be dressed with several garments simultaneously such that underwear can be positioned under a pair of trousers. The approach took 1.5 and 3.5 minutes to position 2 and 4 cloth garments simultaneously with 10k and 15k total vertices respectively. It also featured auxiliary particles to help guide the sewing to ensure cloth correctly wraps around the body, as well as the sewing of pockets onto garments.

Thanh and Gagalowicz [LTG05] developed automatic pre-positioning of virtual garments by using 2D generic figurines placed over a generic silhouette of a character. Two generic figurines are used for each garment, one for the front and back that match the design type (e.g. shirt, skirt, trousers etc.). They create mappings from generic silhouettes onto real ones (obtained from 3D characters), and the figurines are transformed using the mappings with 2D collision detection. Although their intent was for prepositioning of 3D garments, they left the extension from 2D to 3D for future work.

Igarashi and Hughes [IH02] presented techniques for putting clothes onto a 3D character and allowing interactive manipulation of them. Users are required to paint freeform lines onto the 2D cloth and corresponding lines onto the character and the system is able to automatically wrap clothes around the character in a best effort attempt to match the lines. Rewrapping can be done to adjust the position which is especially useful for topographical changes, with helpful pins to limit the area of effect or hold the cloth in place. Additionally the user is able to manipulate the cloth by

dragging it across the surface using a mouse, at the same time exploiting connectively information to prevent cloth vertices becoming stuck in local minima.

Fuhrmann [FGW05] devised semantics and ontologies for garment patterns that can be incorporated into the simulation of virtual clothing. Firstly they give an abstract description of a human body using feature points (e.g. neck-back, neck-left, left upper-arm etc.) and their associated body segment hulls (e.g. neck, left arm etc.). A garment data structure holds information about a piece of cloth including garment type, pattern information and seams. Garment properties are built from binary relations, for example some trousers composed of two parts X and Y might feature the relations: 'isLyingOn(X, leftLeg)' and 'isLyingOn(Y, rightLeg)'. The data is used for pre-positioning the cloth prior to sewing, editing the data can for example cause the trousers to be worn backwards. Additionally garments can be dressed in different orders using 'isDressedAfter(trousers, shirt)' which says that the trousers are dressed after the shirt, i.e. the shirt will be tucked into the trousers. The properties are integrated into their collision detection by assigning garments to a layer and the dressing order may even be changed robustly during simulation if not realistically (i.e. they will intersect while crossing over to change order). Its limitation is that it does not support garments to be both under and over each other, so a loosely tucked in shirt that overhang the trousers is not possible. Layered clothing approaches not only can allow more realistic dressing of characters but have also been used to increase performance. Cordier and Magnenat-Thalmann's [CMT02] three layered approach gained real-time performance for simple garments by trading realism by only simulating the third layer. The first and the simplest layer is for tight fitting regions, fixed to the character's skin using barycentric weights of the closest vertex on the closest triangle. The second layer is looser, confined on discs so that they can only move outwards from the surface, which is used for the sleeve and trouser regions

of the character. In this way it achieved a frame rate of 29fps for a shirt and jacket with 542, 1505 and 618 polygons on each layer respectively.

A combined approach for the creation and the dressing of clothing on virtual characters is the sketch based approach of Turquin *et al.* [TCH04]. Users are required to sketch 2D lines to form the outline of garments from front facing view, drawn over the 3D model of a character. The 3D positions are inferred using depth values and a mesh is created via sampling on a regular grid to fill in the outlines. Surface tension is taken into account so clothes such as skirts do not fit tightly to the inside of the legs. Only tightly fitting garments can be created but it is quick and easy for users to create them in just a few minutes. It also lacks control over the draping characteristics and is unaffected by gravity, so the realism of the approach could be improved. This approach was used subsequently at a starting point by Decuadin *et al.* [DJW+06] in their fully geometric approach for clothing design but they enhanced it by allowing the users to draw seam lines and dart lines (cuts in a cloth panel to help it fit better). The seam and dart lines are cut before they create a piecewise developable approximation from the meshes, which they then unfold to provide 2D garment patterns. The final 3D garment is realised using their specially designed control patches for procedural twisting and buckling patterns, the patches are fitted onto the 2D patterns and mapped back onto the 3D developable surface. In this way the deformation of the garments can be computed in different poses geometrically, however, the patterns are too regular which gives an unnaturally perfect feel.

2.4.2 Virtual Try-On

Online shopping for clothing typically features one or a few pictures of a garment but usually only modelled by a single human model. The model's shape and size most likely does not represent the majority of the customers' and therefore often items

must be returned because they are unwanted. A garment might not fit properly or maybe it just looked much nicer on the model but not on the customer after they bought it and tried it on. Customers are either entirely put off from buying clothing online or have to deal with the hassle of returning items. Virtual try-on approaches are a solution to this, by allowing customers to visit a virtual fitting room from the comfort of their own home but this is not easy. There have been several approaches specifically aimed for virtual try-on such as [PLAMT02, CC03, WKK+05, MMJ10], but one could argue that many of the approaches discussed previously in this chapter could be applied to this too. One additional consideration is the accuracy of virtual characters and their tailored measurements and often 3D body scanning is employed for this. D'Apuzzo [DA09] presented recent advances in 3D body scanning, many methods exist to capture body measurements and surfaces such as using lasers, white light scanning, passive and active systems. The captured data can be used for tailored garment design or for virtual garments on realistic bodies and virtual try-on software.

Protopsaltou *et al.* [PLAMT02] developed an internet based virtual fitting room using real 2D CAD garment patterns with standard sizes found in shops. A series of different sized generic body 3D models are generated using resized interpolated contours from a 3D reference model. The garments are sewn together around the 3D models and then are animated. Users could then select which model to use and browse a catalogue of garments to display on the model in a 3D viewer within their internet browser. While this goes some way to improving customers online shopping experiences it does not allow user supplied measurement so the choice of model size and appearance is improved but still limited. Chittaro and Corvagila [CC03] presented a method using Java and VRML (Virtual Reality Modeling Language) to preview 3D garments created from 2D CAD designs for evaluation or publishing on the web. They also created a cross-application data exchange format to combine CAD data

with additional data not found in the CAD data for the simulation including material properties and texturing. It takes ten minutes to stitch and reach a stable state for a shirt and skirt each with 2.5k vertices on a body of 5k vertices.

Realism of the garments is important for customers to be able to have a good representation of what the real item will be like in person before they actually buy it. Wacker *et al.* [WKK+05] presented an approach designed for virtual try-on purposes with photo realistic visualisation of clothes using BTF (bidirectional texture function) and environmental (reflection) mapping. They showed a comparison of standard texturing compared to their BTF rendered clothes, where standard texturing seriously compromised the realism of garments. We have already touched upon about rendering, texturing and shading for cloth in Section 2.1.4, much of this also applies here.

Virtual try-on software for clothing remains a difficult endeavour for any retailer to take up when high quality digital photographs provide cost effective and realistic images that customers are used to. In comparison, virtual try-on for eye wear and glasses is in widespread use by retailers, where simple approaches achieve realistic results. For example, a system where users upload a digital photograph of their face with images of glasses that can be overlaid is effective and easy to implement. Virtual try-on software for clothing cannot provide such a simple solution while also providing equivalent realism.

2.5 Summary

There has been much research in the area of cloth simulation in the field of computer graphics starting in the 1980s and continuing strongly through to the present day. We have a given a broad overview of the main methods and techniques for cloth simulation in the literature; these are often highly coupled systems encompassing many areas. Both implicit and explicit numerical integration used for physically based cloth simulation dominates the literature but there are alternatives such as geometric or data driven methods. Robust and stable collision detection is particularly difficult for deformable bodies and cloth has the additional challenges of self-collisions. The process of garment design which is at the heart of the textile industry is becoming increasingly important in computer graphics because of the need to generate high quality and realistic clothing. The fidelity of offline simulations is very good producing plausible behaviour and fine wrinkles resulting in believable and realistic animations and images. However, interactive and real time simulations face a severe lack of computational budget to achieve comparable results where coarse meshes must be employed that have low resolutions which cannot faithfully represent the shape of naturally deforming clothing. Improvements in hardware and parallel processing have enabled some of what was only achievable offline in the past to become available at interactive frame rates but of course these improvements have also advanced offline results as well making large scale cloth simulation possible. It is not sufficient to rely on hardware alone; the use level of detail methods such as adaptive methods has been shown to improve performance of cloth considerably but have rarely been used for virtual clothing.

Chapter 3

The Edge-Based Adaptive Mesh

3.1 Introduction

In this chapter we introduce our edge-based adaptive mesh that we have developed for use as a mass-spring network for cloth simulation. Many aspects of the adaptive mesh are covered in this chapter, including triangulation, memory management, cloth simulation and rendering. We will first motivate our approach and explain the idea and workings of the edge-based adaptive mesh.

3.1.1 Motivation and Inspiration

Adaptive meshes have been used to accelerate offline cloth simulations with good success, but they had not been fast enough for real-time work. Li and Volkov's [LV05] adaptive simulation spent 87ms on refinement and 1.2 seconds on average with 12k triangles for the cloth simulation. Similarly, Volkov and Li's [VL03] approach took 1.3 seconds in total with the adaption taking 9% of that (117ms) but simulation times were on average 2.6 times faster than the non-adaptive mesh. These approaches focus on the refinement of triangles and generating a conforming mesh between levels.

We were inspired to create an adaptive triangle mesh but where the triangles take a secondary role to refinement. We started by examining the possibility of edge-based refinement, since edges are simpler structures than triangles we thought it could lead

to a more efficient approach. Another important motivation is that the refinement must be able to be reversed, i.e. coarsened. Edges by themselves permit a very simple (and hopefully fast) refinement approach through bisection, splitting into two and coarsening by later rejoining those split edges. The natural way to represent this was recursively just like a binary tree structure. The hierarchy then implicitly keeps a history of edge splits, such that they can be reversed. If an edge was completely replaced by a split with two new ones, it would be difficult to select which ones to merge for coarsening. Triangles are important still, the next part was to come up with a way to generate a conforming refinement of triangles in an efficient way. We considered the simple cases where a triangle's three edges are either split or not and possible conforming triangulations that could be used in those cases. This lead us to come up with a state based triangulation approach based on the status of a triangles three edges, such that a refinement pattern is selected depending only on which edges are split. Following the same recursive approach as the edges, we can then form a hierarchy of triangles. New vertices are added on edge splits, and new edges are created as necessary for the refinement patterns. We explain the approach and our edge-based adaptive mesh in detail in the following sections of this chapter starting with the mesh representation next.

3.2 Mesh Representation

Polygon meshes have been widely studied and there exists many representations which differ in their memory use and efficiency for a particular application. Topological connectivity of the mesh is particularly important to us, as it is with many other subdivision methods. In this section we first provide some background information, discussing briefly some existing mesh representations and then will introduce the structure we use in our edge-based adaptive mesh.

3.2.1 Background

Vertex-vertex meshes are considered to have the simplest representation composed only of vertices, where each vertex contains a list of connected vertices. Face and edge information is completely implicit and can only be found by iterating over the vertex information so operations on edges or faces are not efficient. However, it can be efficient for shape morphing and it also has very small memory requirements. Face-Vertex meshes store more information, faces are composed from a number of vertices and each vertex stores its immediate neighbourhood of faces but again edges are implicitly defined. The increased memory usage has made the representation more efficient to use for tasks operating on faces and vertices, this format is commonly used in computer graphics for rendering. The Winged-Edge data structure [Bau72] goes further and represents all vertices, edges and faces explicitly; an edge is connected to its two vertices, to its two faces and its next and previous edge around each face. It can be made more compact by storing only the next edge, the previous edge can be found by iterating all the way around the face. The half-edge [Ket98] representation is similar to the winged-edge format but each edge is split into two oppositely directed half-edges, each contains half the connectivity information but with references to each other. The half-edges explicitly encode edge orientation whereas

the winged-edge does not, so direct traversal is possible whereas the winged-edge is case distinctive. The half-edge mesh has been shown to handle arbitrary polygonal meshes with irregular structure well and is the basis of Openmesh [BSBK02] which is an open source polygonal mesh framework.

Edge centric formats are able to handle n-sided polygons efficiently since the edge structures are always the same size. Faces only need to store a single reference to one of its edges; the other edges and the triangle's vertices can be found by traversing the edge's connectivity information. They are useful for editing meshes; however, they are less efficient for rendering than Face-Vertex meshes. Tobler and Maierhofer's [TM06] mesh format for subdivision combines the merits of the Winged-Edge and the Face-Vertex meshes, but requires slightly less memory than the Winged-Edge mesh uses by itself. The main disadvantage is they disallow topology changes of the mesh once it is created to enable efficient rendering on computer graphics hardware. The idea is the topological information can be used to generate a whole new subdivision mesh efficiently (rather than editing the existing one) for level of detail rendering of meshes. They provide the means to 'stitch' meshes together by setting a face from each mesh as equivalent, they show an example of a tree made from branches, twigs, leaves and trunk parts stitched together and smoothly subdivided.

3.2.2 Mesh Structure

Our adaptive mesh has a number of requirements, some of which reduce the complexity of the representation needed. For instance, the representation must be able to store a mesh composed of triangles but only triangles so we do not need the ability to handle arbitrary polygon meshes (such as those with n-sided polygons, or non-manifold surfaces) for modelling clothing. The mesh must support a two-dimensional manifold surface with boundary edges and holes (internal boundary edges) required

for clothing, this may seem unnecessary to state but it introduces subtle issues with connectivity. We cannot assume that an edge is adjacent to two triangles, but we can guarantee that each edge is adjacent to at least one triangle by disallowing edges without a face. There may be uses for an edge that has no adjacent triangles in cloth simulation such as for modelling a piece of string or lace. However, such cases would perhaps be better served by separate add on to the mesh where these special edges could be handled and rendered specially. This allows the mesh to be rendered efficiently as triangles in the same way normally realised in computer graphics.

In addition to connectivity information for the surface, the mesh must feature links between refinement levels to support our recursive refinement approach. In tree structures, a child node does not always require access to their parent node. This is the case in our mesh; two-way access is not required for our refinement approach and only one way access from parents to child is needed. However, parent references are simple to implement or can be passed down through the hierarchy as parameters to recursive functions if they are ever required for another application.

We formulate the mesh in an object orientated manner, where we store vertices, edges and triangles explicitly as three objects (classes) using the C++ programming language. The objects serve a dual purpose for adaptive refinement and cloth simulation. We will discuss each of them in more detail in the next parts. We consider what is needed for the adaptive mesh and what is also needed for the mass-spring system.

3.2.3 Vertex Structure

A vertex is both a point in the edge-based adaptive mesh and also a particle in the cloth simulation. Vertices are not responsible for the main operations during refinement; they are managed and controlled by their adjacent edges and triangles.

A vertex has the following variables stored in it:

Level The level on which this vertex resides in the adaptive mesh, a vertex in the base mesh has a level equal to zero.

Position The position of the vertex in 3D space, expressed as an xyz vector.

Previous Position This provides a limited history of the particle's position, when the simulation is advanced, the position is updated and the old value is copied here. The velocity of a particle can be found using this, and it is used by the numerical integration and can be useful for collision processing.

Surface normal The normalised vector that is perpendicular to the tangent plane on the surface at this vertex, we calculate it as an average of its adjacent triangle face normals weighted by their area. This will be used for lighting the mesh during rendering but also by refinement criteria.

Mass The mass of a particle, which is calculated as one third of the average mass of the particles surrounding adjacent triangles.

Force A vector where forces acting on a particle are accumulated before numerical integration.

The only connectivity information that a vertex requires is access to its adjacent triangles for the calculation of its surface normal and mass, like that provided by the Face-Vertex format. Typically, the lighting normals must be recalculated every time the cloth deforms before rendering and the mass must be recalculated when the adjacency of a vertex changes during refinement. We have chosen to store a list of adjacent triangles for each vertex enabling direct access for best performance at the cost of memory. In theory this list does not have a maximum size but in practice we will see later that some very good estimates can be made once so dynamic memory will

not be required even with the adaptive refinement of adjacent triangles (see Section 3.3.2). We illustrate the adjacent triangle connectivity required in Figure 3.1.

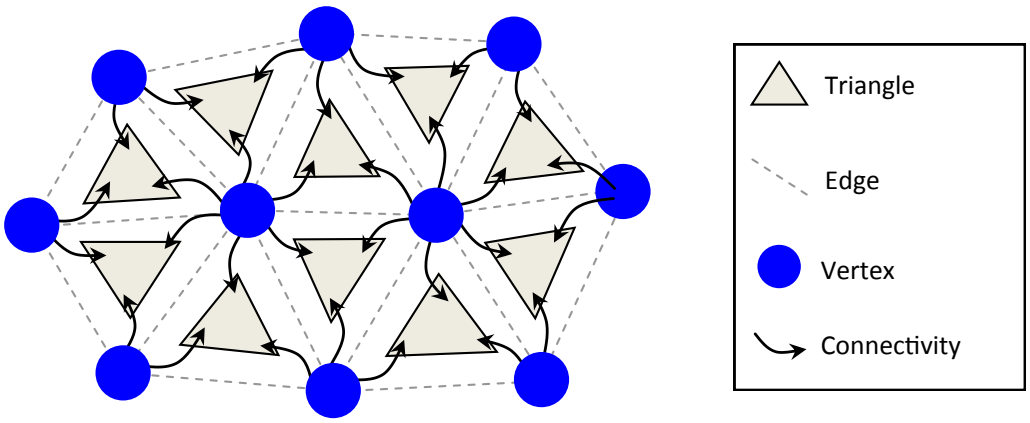

Figure 3.1: This figure shows the connectivity information for several vertices of a mesh, each vertex stores connectivity links to its adjacent triangles indicated by the arrows.

3.2.4 Edge Structure

Edges control refinement throughout the whole mesh, criteria decide when an edge should be split or later rejoined, and they also serve as stretch springs for the mass-spring simulation but also manage bending forces between their two adjacent triangles. We have two different versions for the edge's internal structure. This does not impact the refinement procedure from a conceptual level but it does change some of the details involved in the implementation of the adaptive mesh. We will firstly explain edges from a higher level and then explain both versions at a lower level.

An edge is created between two vertices (A and B) and it is adjacent to either one or two triangles only. On the boundary of the mesh there will be one adjacent triangle, otherwise two for all internal edges. We define left and right sides of an edge based on a local 2D view of the surface from above (the direction of surface normals)

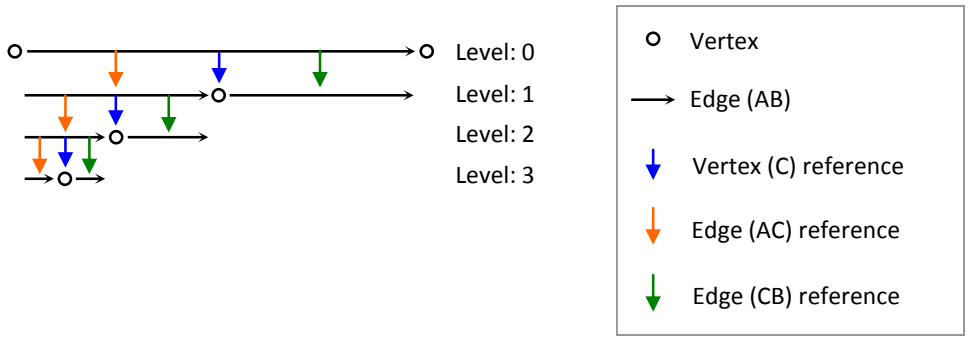

Figure 3.2: C are the central vertices created when an edge is split, AC and CB are child edges for the first and last half of the split edge, these labels are not unique but they are relative references from a parent to its immediate children. In the figure an edge has been split into two and then only AC referenced edges have been recursively split to level 3.

with the edge directed from A to B. An edge stores pointers to vertices and adjacent triangles. When refined, edges create a new vertex (C) at their centre and create two new child edges (AC and CB) connected from A to C and from C to B respectively. Coarsening of an edge is simple, the child edge central vertex are deleted. Edges therefore form a hierarchical tree of edges and vertices, see Figure 3.2 for an example tree. An edge has the following variables stored in it:

Level The level on which this edge resides in the adaptive mesh, an edge in the base mesh has a level equal to zero and its immediate children (two edges and centre vertex) will have a level of one.

Rest Length The undeformed length of this edge for the stretch spring.

Length The current length of this edge, optionally cached for reuse. Only needs to be recalculated when vertices move.

Bending Vertex references Bending force calculation requires access to each of the adjacent triangle's opposite vertex, found using connectivity information

but can be optionally cached.

Bending Data Depending on the bending model used, the edge may store some additional data for that purpose.

Edge-Pair Implementation

In our first implementation (published in [SLD09, SLD10], edges are made from two oppositely directed pieces that form a pair and are connected together with pointers to each other and hold a pointer to their adjacent triangle. This is inspired from the Half-Edge data structure but we do not store next and previous edge connectivity (we can find it easily since faces are always triangles) and we store additional data for refinement (vertex C and edges AC and BC).

In our approach, one edge is directed from A to B, and the other half is directed from B to A. One of the edges is designated as the control edge, this choice is arbitrary unless on the boundary. On the boundary of the mesh, only a single edge is used so it will be the control edge. It is only necessary to store a reference to the control edges in the mesh, the opposite edge in each pair can be found via the control edge. The control edge is responsible for mirroring any operations required (splitting and joining) so that both sides of the hierarchy are complete but oppositely directed. This approach can be specified with the class defined in Listing 3.1, we call them 'adaptive half edges' to avoid confusion with the Half-Edge data structure. The distinction between C++ references and pointers is considered; a reference as a class member variable can only be set in its constructors' initializer list and they cannot be reseated (changed to reference another object) or be null. Therefore in the majority of places we require pointers. A and B in Listing 3.1 could be references but that would prevent some topological changes (such as merging vertices) so for maximum flexibility we use pointers everywhere.

```
1  class AdaptiveHalfEdge
2  {
3  public:
4    AdaptiveHalfEdge *Opp; // oppositely directed edge, may be null
5    Triangle *T; // adjacent triangle, may be null if not control
       edge.
6    Vertex *A; // edge start
7    Vertex *B; // edge end
8    Vertex *C; // pointer to centre vertex, not split if null
9    AdaptiveHalfEdge *AC; // child first half, between A and C
10   AdaptiveHalfEdge *CB; // child second half, between C and B
11   /*   Rest of Edge Data   */
12 };
```

Listing 3.1: Edge-Pair Implementation: AdaptiveHalfEdge

The advantage of this approach is that there are no special considerations needed to exploit the connectivity, each edge in the pair works without needing to know which side you are accessing it from so it is simple to use. For example, triangles can refer to their three edges and rely on the fact that they are all defined in an anticlockwise direction. However, the cost to memory is high; every piece of data is duplicated for each edge in the pair including the hierarchy although each stores an opposite version of the other. Also some of the data is not directional, so care is needed to synchronise the duplicated data or only use data from the control edge. Storage is only efficient for edges on the boundary, since only a single side is stored.

Edge-Trio Implementation

The ability to be able to store a reference to a particular side is very useful, if not one must store a Boolean value with the reference which indicates the side pointed to. This has been used by [TM06], but they stored a tightly packed separate bit array (global to the whole mesh) for this purpose for optimal memory usage. However, their approach did not allow the mesh topology to change as the order in the array (and other arrays) had to be fixed. We do not want to have to store and maintain this extra Boolean variable, so we designed an approach where a single pointer can

reference an edge side as in our edge-pair implementation by using a trio of edge pieces including a main edge and two side pieces.

We create a main single edge directed from A to B, which stores all data about an edge and its hierarchy but does not directly store the references to adjacent triangles. We create two edge-sides that store a reference to the main edge and their adjacent triangle. The edge-sides are members of the main edge, so explicit references to them do not need to be stored and therefore the whole edge including the edge-sides can be allocated efficiently in a single block of memory. In this way, only a single pointer is duplicated and we keep many of the advantages of having half-edges without as much duplication of data, see Listing 3.2 and Figure 3.3.

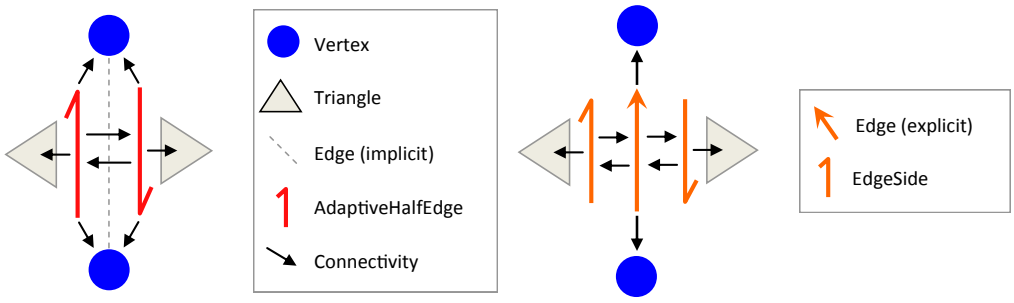

Figure 3.3: This figure shows the connectivity information for an edge, the Edge-Pair implementation is shown on the left and the Edge-Trio implementation is shown on the right.

```
1  class EdgeSide
2  {
3  public:
4    Edge *E; // pointer to edge this side belongs to.
5    Edge *T; // pointer to adjacent triangle, may be null if not
         control side and on a boundary of the mesh
6  };
7  class Edge
8  {
9  public:
10   EdgeSide leftside; // left edge side
11   EdgeSide rightside; // right edge side
12   Vertex *A; // edge start
13   Vertex *B; // edge end
14   Vertex *C; // pointer to centre vertex, not split if null
15   Edge *AC; // child first half, between A and C
16   Edge *CB; // child second half, between C and B
17   /*   Rest of Edge Data   */
18 }
```

Listing 3.2: Edge-Trio Structure

Although we have lost direct access to an edge side without considering the direction, member functions for the edge-sides can at least remove much of this new burden. For example, if we wish to access the first vertex in relation to an edge-side we can use the function in Listing 3.3. The theoretical cost of access is increased compared to the edge-pairs with an extra layer of indirection even if the direction is not important (e.g. to access the central vertex from a side, $side \rightarrow C$ vs. $side \rightarrow E \rightarrow C$). However, the performance of modern CPU's is highly dependent on access patterns and cache effects. In this approach, the edge is held in a single smaller block of memory so it should be more cache friendly, and the cost may be unnoticeable. Also the efficiency of operations on the edge itself is improved including importantly refinement, and coarsening since removing duplicate data so it is hard to ascertain overall performance. Therefore, we feel the main attraction to this approach is the memory requirements and the removal of data duplication. Each side of the Edge-Pair approach requires 7 pointers (28 Bytes) plus data so the pair costs 56 Bytes plus twice the size of the data; the Edge-Trio requires a total of just 9 pointers (36 Bytes) plus

data. Therefore the memory usage is reduced to 64.3% in the worst case (no data), if we assume edge data of 12 bytes (storing level, rest length and length) it becomes 60%, and it tends to 50% as data size increases and duplication of data becomes the overriding factor.

```
1  Vertex* EdgeSide::getVertexStart(void)
2  {
3    If(this == &E->leftside)
4      return E->A; // on the leftside, A is first
5    else
6      return E->B; // on the rightside, B is first.
7  }
```

Listing 3.3: EdgeSide::getVertexStart()

3.2.5 Triangle Structure

Triangles have the important responsibility for creating triangles on the next level and re-triangulating existing triangles on the next level. A single triangle is only responsible for its immediate children of which there can be up to 3 internal edges and up to 4 internal triangles. We cover the refinement approach in detail in Section 3.3. Triangles have no direct duties in the mass-spring simulation but provide data for the calculation of the particle's masses and surface normals, they have the following variables:

Level The level on which this triangle resides in the adaptive mesh, a triangle in the base mesh has a level equal to zero and its immediate children would have a level of one.

Material Coordinates The 2D coordinates of the triangle's vertices in an undeformed frame of reference (explained later in detail in Section 3.6.1).

Area The area of the underformed triangle.

Mass The mass of the triangle, found by its area multiplied by the cloth's density,

optionally cached for performance.

Face Normal The normalised vector that is perpendicular to the plane of this face, we calculate using the cross-product of two edge vectors.

Triangles require access to their three corner vertices and three (external) edges, providing the edges alone would give access to the vertices. However, we store direct references to the vertices for best performance for vertex access since they are used frequently for rendering. The basic outline of the structure can be seen in Listing 3.4, it does not make any difference which edge implementation is used and we illustrate the connectivity also in Figure 3.4. The first edge runs between the first and second vertices defined in an anti-clockwise order, therefore the second edge runs between the second and third vertices and the third edge is between the third and first vertices.

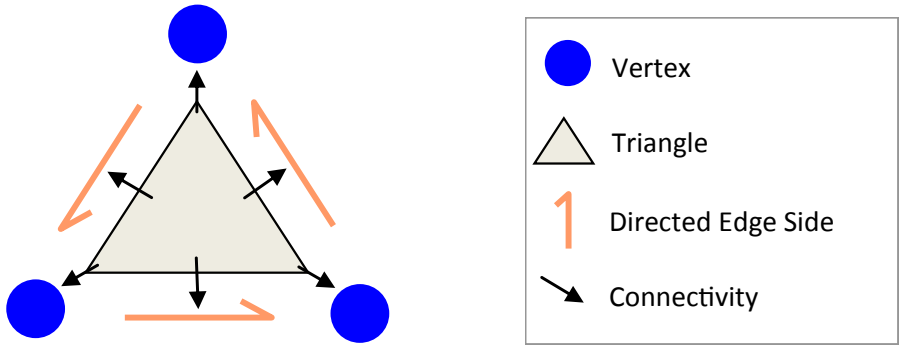

Figure 3.4: This figure shows the connectivity information for a triangle, each triangle is connected to three vertices and three edge sides which are defined in an anti-clockwise order.

```
class Triangle
{
public:
  Vertex* V[3]; // corner vertices.
  EdgeSide *exE[3]; // external edge sides.
  Edge *inE[3]; // internal edges on the next level.
  Triangle *inT[4]; // internal Triangles on the next level;
  /* rest of triangle data */
}
```

Listing 3.4: Triangle Structure

3.2.6 Managing properties and settings

There are many properties and settings for both the simulation and adaptive mesh, and a lot of them are required to be used in many places. The use of global variables can be considered bad practice but it serves a good purpose here to avoid passing many frequently and widely used variables. This brings performance savings by reducing function overheads and makes code easier to maintain. The properties that make up the cloth simulation and adaption are most conveniently stored in two classes, called ClothProperties and AdaptiveProperties. They provide convenient support for loading and saving properties to the hard disk in a human readable text format. The cloth properties also feature helper functions for converting S.I. units (lengths, accelerations, areas) to internal units using a scale factor of internal units per metre. All units are presented to the user in S.I. and are saved to disk in S.I., this makes it easier to use internal units other than metres but the user does not have to adjust the physical properties of the cloth themselves. Adaptive properties store the settings for refinement, such as the maximum refinement level allowed and criteria settings for refinement and coarsening.

We access each through a global pointer, which initially points to a static object with default values. Typically when a garment is updated, it has its own properties and they are bound to the global pointer, the update is performed and then unbound

by setting the pointer back to the default properties. This allows edges, vertices and triangles of the mesh access to the properties whenever required without the overhead or maintenance requirement of explicit passing. The cost of binding is very little, just a pointer is set manually or by the provided functions which can be inlined by the compiler. A possible usage of the cloth properties is shown below in Listing 3.5, the cloth should not be updated if the user has paused the simulation.

```
1  void Cloth::Update(void)
2  {
3    if(mClothProperties)  // bind the cloth's properties (if it has any)
4      mClothProperties->Bind();
5
6    if(!global_clothProperties->paused) // access through global pointer
7      Simulate();
8
9    ClothProperties::BindDefault(); // restore default properties
10 }
```

Listing 3.5: Example usage of the ClothProperty class to store and access global cloth settings

3.3 Adaptive Triangles

In this section we detail the refinement of triangles in the edge-based adaptive mesh; we focus on many implementation details important for run-time performance. Edges are responsible for controlling the adaptive mesh; however, the triangles form an important part of the refinement. We leave edges for now, since they are key details about the triangles to be understood which are important for edge refinement and coarsening. The representation of each triangle is straight forward; the vertices and edge sides are defined in an anti-clockwise order. As discussed earlier, this is accomplished by storing a reference to a directed edge side using either the edge-pair or edge-trio implementation.

3.3.1 State-based Refinement

We generate a refinement or subdivision pattern using a state-based approach, this ensures that the refinement is conforming between levels with no T-junctions. We refer to a triangle's original edges as its external edges; additional internal edges and internal triangles are created as required by the subdivision patterns. Internal edges and internal triangles are children of the triangle and it is their parent. A triangle is refined based upon the status of its external edges, that is whether they are split or not. This leads naturally to eight (2^3) states as there are three external edges of which each has two possible states. We label these eight states as S0 to S7, where S0 represents a triangle with no internal subdivision whilst S7 represents a triangle that has completed a full 1-to-4 subdivision. Subdivision patterns are generated as follows: States 1-3 uses the bisection rule, for states 4-6 the triangle is specially divided into three to match the edges. Notice that in this case of states 4-6, one of the internal edges could be placed in two ways (we address this later, see Section 3.3.5. The refinement patterns and their corresponding states can be seen in Figure

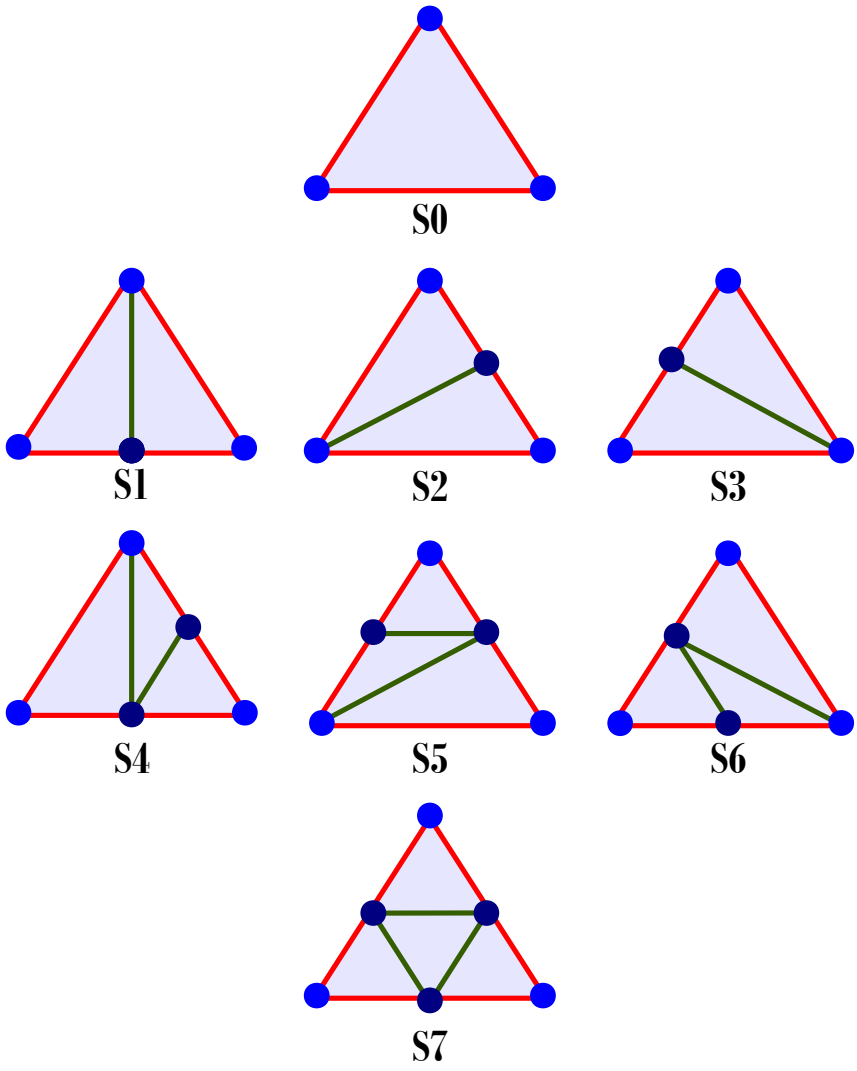

Figure 3.5: The state of a triangle is found from the status of its external edges, triangles are triangulated internal to conform to the edges. The eight states are labelled from S0 to S7.

3.5. Edges trigger refinement and retriangulation of their adjacent triangles when they are refined or coarsened.

In our incremental approach, it is important to prevent a difference of more than one refinement level between adjacent regions. This ensures a semi-regular refinement

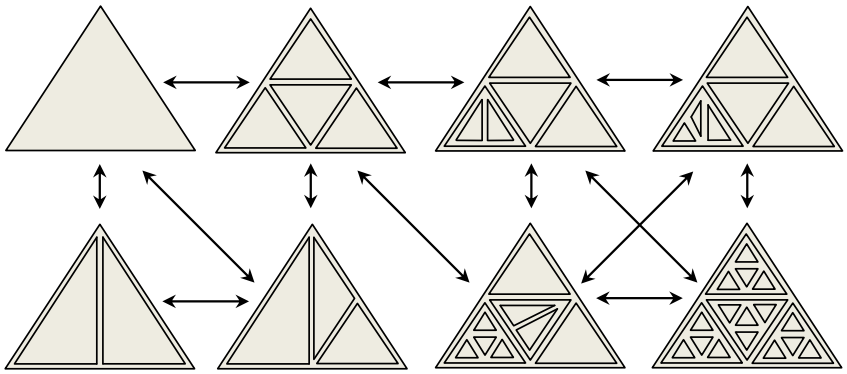

Figure 3.6: This figure shows some possible refinement patterns where internal triangles are drawn inside of their parent, in this way the triangle hierarchy can be visualised in 2D. The arrows indicate the possible incremental transitions between the refinement patterns.

on the borders between regions of different refinement levels and regular 1-to-4 refinement within regions of the same level. This is accomplished in two parts, first and simply, we only recurse to child triangles from a parent if the parent is in S7 (hence, we prohibit recursive refinement until S7). Secondly, since edges trigger the refinement, we have to prevent some edge splits. We provide a special state that indicates to edges that a triangle cannot be refined and this can be checked efficiently. We number this state -1 (minus one, labelled S-1); it is the same as S0 in all other respects. Child triangles are created with their S-1 as their state, when a triangle transitions to S7 it must change its children's states to S0 to enable further refinement. We show some possible refinement patterns and incremental transitions between them in Figure 3.6.

3.3.2 Vertex-Triangle Adjacency

Our refinement approach tends to keep the number of adjacent triangles around each vertex low (and the vertex valence low also); a base vertex originally surrounded by n refined triangles will have a worst case triangle adjacency of 2n (if all triangles

are bisected radially outwards from the vertex). A base vertex surrounded by n triangles that have been fully subdivided into four will still only be surrounded by n child triangles. Non-base vertices on (level one and up) will have a worst case triangle adjacency of 12 and a best case of 6 within the mesh. Non-base border vertices have a best case of 2 or 3 and worst case of 6 on the edge of the mesh. By knowing this we can easily allocate efficient fixed size arrays to store the adjacent triangle lists discussed earlier, that is the lists must be able to hold a minimum of 12 triangle references while also having a enough space for two times the largest number of adjacent triangles in the base mesh. This is specified at compile time; we lean on the safe side and allocate enough space for 20 triangle references allowing base meshes with up to 10 triangles adjacent surrounding any one vertex. This is very convenient for triangle adjacency since it is not affected by holes in the mesh; unlike vertex valence where the number of adjacent vertices and edges can exceed the number of adjacent triangles. If vertices require edge connectivity or vertex connectivity then any holes in the mesh would need to be taken into account.

When a triangle is retriangulated, it is responsible for maintaining the vertex triangle's lists so they are up to date with the current state. We do not require the list to be in any particular order so we do not spend any resources in maintaining the order. Vertices provide three relatively fast operations for the triangle to use to maintain the list: Insert, Remove and Replace. Insert adds a triangle to the list by placing at the end of the list and increments the size of the list. Remove searches the list for a specified triangle and removes it by swapping it with the triangle at the end of the list and decrements the size of the list. Replace searches for a triangle and swaps it with another specified triangle such that list does not change size (more efficient that using Remove followed by Insert).

3.3.3 Delayed Retriangulation

Retriangulation is a costly operation, it is not ideal to immediately perform it on the two adjacent triangles when an edge is refined or coarsened, this will lead to situations where triangles are retriangulated up to three times instead of once in a single refinement update. Although the problem seems straight forward, it is hard to come to a perfect solution to manage this. If there are few retriangulations needed in a single step, retriangulating a few triangles three times is unlikely to make noticeable difference. However, we must ensure good worst case performance for interactive applications, so we should only perform it once. We delay retriangulation until the final step, at a point where all edges have been processed for refinement or coarsening. We tag triangles where their triangulation is not up to date with their state; edges are responsible for setting this when they are split or rejoined. Then during retriangulation, triangles only retriangulate themselves if they are tagged, after which it is cleared to indicate that they are up to date.

3.3.4 State Transitioning

Triangulation is very simple to implement if transitioning starting from S0 to any other state (i.e. S0→SX). So to change from any state to another, we could first clear the triangulation completely back to S0 first and then triangulate to the desired state as if we were transitioning from S0. However, a triangle in S0 has vertices that contain references to it so we define a special cleared state where the triangle has no internal edges, no internal triangles and additionally there are no vertices that contain any references to it. We refer to this cleared state as SC, so a transition from S4 to S6 we follow this indirect route: S4→SC→S6. Clearing a state therefore involves the removal of all triangle references in the adjacent vertices, and deletion of all internal edges and internal triangles.

It can be seen that many transitions between two states share common features, so the indirect method is not ideal. For better performance it would be good to try to reuse as much as possible in the current state and therefore directly transition from one state to another. This requires much more code, since now we must implement every transition possible to take full advantage of this, i.e. every permutation of unique state pairs since the transition S1→S2 is different from S2→S1. There are 56 unique transitions required for the main eight states. Retriangulation of a triangle now involves calculating a state transition; we know that the external edges will not match the current state if a triangle has been marked to be updated. So given a triangle's current state and the status of its three edges, its new state can be determined quickly and the transition can be implemented using two layers of nested switch statements. The memory is reused for the internal edges and triangles, extra memory is only allocated or deleted when the number of edges or triangle increases or decreases respectively from one state to another. After the triangle is reconfigured into that state, it is untagged to indicate it is now up to date.

3.3.5 Triangle Configurations

Triangle states are uniquely defined by the status of their edges. However, as mentioned earlier the triangulation for states 4, 5 and 6 are not unique. We had arbitrarily chosen one of the possible two triangulations for each but this was not ideal. We have expanded this approach to include the other three triangulations missing from the previously defined eight states. We therefore require an additional identifier other than state for the triangulations; we call this a triangle's 'configuration'. There are a total of eleven configurations starting with an empty parent triangle to a completely sub-divided one (1-to-4 split) and all the possibilities in-between. Every state is mapped to a configuration, except in three cases where now each of those states

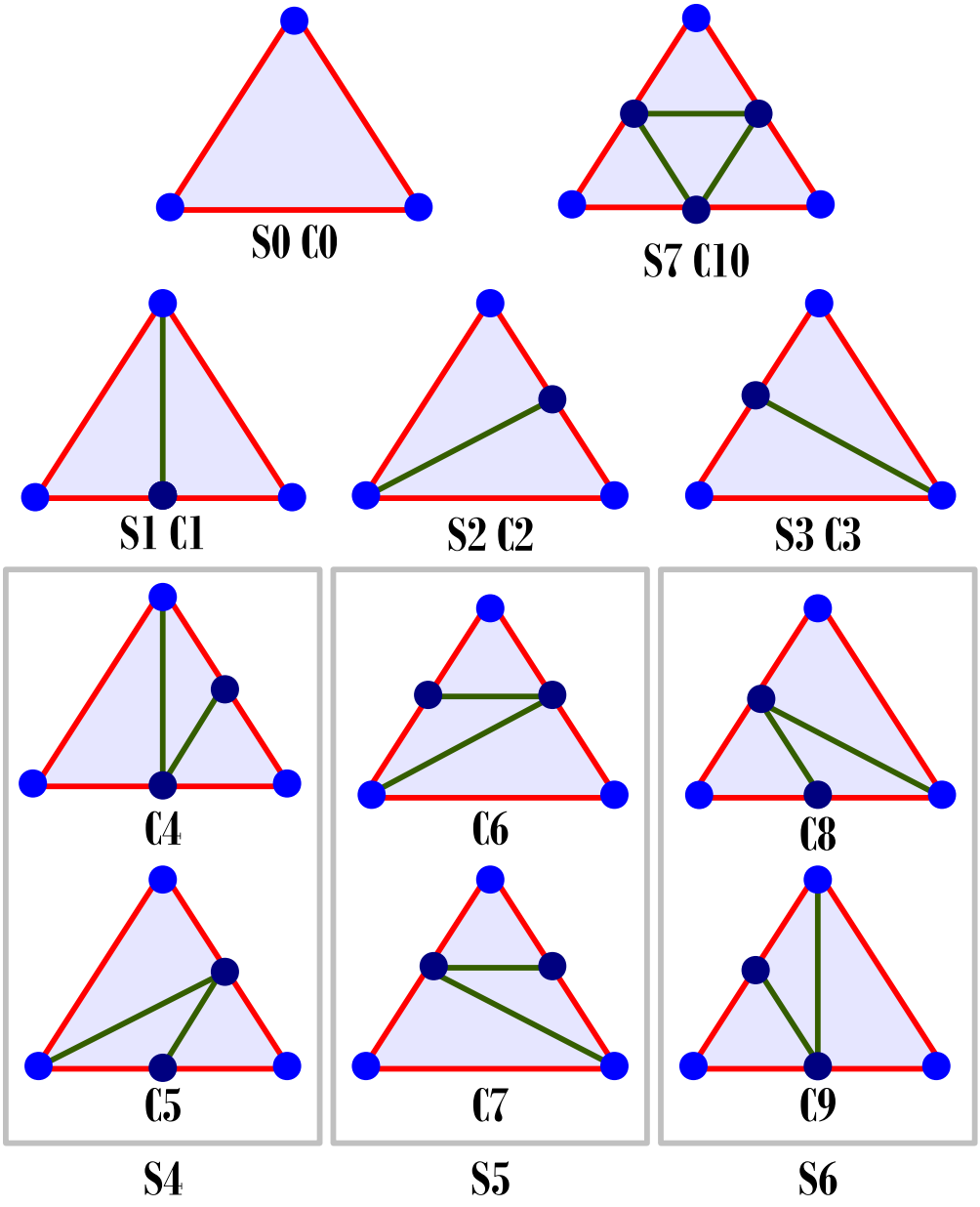

Figure 3.7: The state of a triangle maps to a configuration, in three cases two configurations share the same state.

maps to two possible configurations. Figure 3.7 shows all eight states and eleven configurations and their corresponding subdivision patterns.

The major motivation for employing configurations as well as states is to improve the performance of state transitioning; we can exploit the extra three configurations

for greater reuse. For example, consider that when transitioning from S1 (C1) to S4 (C4/C5), we can choose to transition to C4 as it is closer to the initial configuration than the other choice. If C5 were chosen instead the change may be visually detectable and manifest as popping in the rendered cloth. By choosing transitions of minimal change, not only is retriangulation faster and there may be less popping, it also keeps the forces more consistent between simulation steps as the cloth adapts.

The only unfortunate consequence is we now have even more transitions to implement, increasing from 56 with eight states, to 110 with eleven configurations. The triangulation is still state-based, i.e. there is no retriangulation unless the state changes so this removes six possibilities where the configuration changes without the state changing also. Also if we prohibit the non-minimal changes (where there is a choice), even less transitions are needed. However, we have implemented all of the transitions in order that we may provide complete performance statistics for all transitions.

We have looked at reducing the burden to implement so many direct transitions, such as by exploiting rotational symmetry. In the simple transitions such as C0→C1, C0→C2, C0→C3 it is easy to exploit the rotational symmetry of C1, C2 and C3. We could rotate the configuration just by altering the indices used to configure the triangle, for example if C1 places the new vertex on the first edge; then C2 places it on the second edge and C3 places it on the third edge. However, this can also create a lot of complexity and additional problems that can overshadow the benefits of code reuse. For instance, configurations can have more than one internal representation, in C1 the left or right child triangle may appear first in its array, and they could be swapped in C2 or C3. Also C10 could be rotated without any change in appearance, but the transition C10→C9 needs to know internal representation to reuse the lower left triangle. We feel it should be possible to come up with a conceptual model to describe

our configurations in a high level way. This could be used with automatic or generative programming to create source code for transitions between pairs of configurations by searching for the differences between them at compile time. However, this remains a large research problem in itself but it would be interesting to explore in the future since it would make refactoring the mesh for different purposes very easy.

The configurations can be seen in more detail in Figure 3.8, they tend to favour transitions where there are few internal edge changes. We feel this is good for performance, not only because they permit faster transitioning but we also expect them to be used more often. For example we expect transitions such as S1→S4 (a second edge splits) will occur more often than S1→S5 (where one edge rejoins and the other two split in a single step).

We have looked at this using the draping experiments that are employed later in this chapter (see Section 3.8.2), and we have recorded the total numbers of state transitions used for these. The results are summarised in Table 3.1, transitions such S1→S4 do occur more than S1→S5, and actually S1→S5 is not used at all. This is also the case with transitions S2→S5 versus S2→S6 and S3→S6 versus S3→S1, but overall these make up only a small percentage of the transitions. It can be seen that over 40% of transitions are those to State 7, including 29.5% from States 4,5,6 (two edge splits) and 9.1% from State 0 but very few from States 1,2,3 (one edge split). This means that the triangles are more likely to gradually refine where edges split one after another in contrast to all three of their edges splitting at once in a single update.

Configuration Selection

Whenever we wish to configure a triangle into states 4, 5 or 6 there is two possible configurations for each state, one of the edges may be placed in one of two positions. We have discussed that minimal changes are best, and we always use them. There

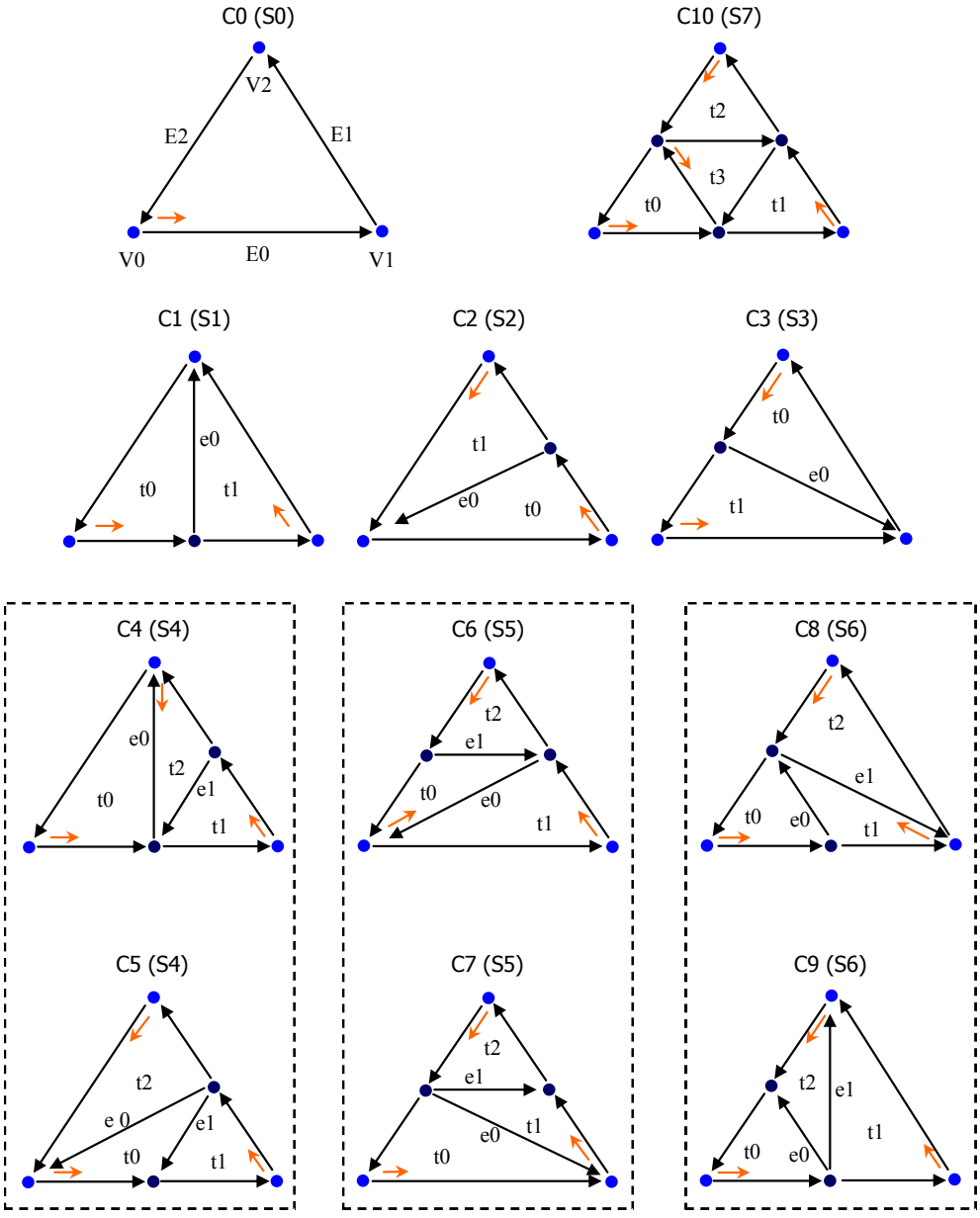

Figure 3.8: This figure shows the configurations in more detail, C0 has the external vertices and edges labelled V0,V1,V2 and E0,E1,E2. The internal triangles are labelled t0, t1, t2, t3 and the internal edges are labelled e0, e1, e2. The orange arrow indicates the order of vertices and edges in which the triangle is defined, for instance in C1, t0 starts from V0 with the first child of E0 in an anticlockwise direction.

are some case where the cost to transition between them is similar (e.g. C0→C4/C5), and we need to either use a default selection (choose C4 over C5, C6 over C7 and C8

Table 3.1: This table shows the frequency of state transitions as a percentage of a total of 30,471 recorded transitions for the draping experiments in Section 3.8.2. A state transition consists of a triangle changing from one state (row) to another state (collumn), for instance 9.1% of transitions were from State 0 (S0) to State 7 (S7) and 2.7% were from S7 to S0.

	0	1	2	3	4	5	6	7
0	-	1.6%	4.3%	1.6%	3.7%	3.7%	1.8%	9.1%
1	0.6%	-	0.0%	0.0%	1.5%	0.0%	1.1%	0.6%
2	0.8%	0.0%	-	0.0%	2.3%	2.3%	0.0%	1.2%
3	0.7%	0.0%	0.0%	-	0.0%	1.6%	0.9%	0.6%
4	0.5%	0.8%	1.0%	0.0%	-	0.1%	0.1%	10.8%
5	0.5%	0.0%	0.8%	0.9%	0.1%	-	0.1%	10.6%
6	0.4%	0.7%	0.0%	0.5%	0.1%	0.1%	-	8.1%
7	2.7%	0.8%	0.7%	0.8%	6.4%	6.0%	6.3%	-

over C9) or select it based on some criteria. We have investigated the use of three simple approaches based on edge curvature, edge length and edge rest length. The two length criteria choose the configuration that gives the shorter new edge length (to avoid long thin triangles), based on either the current length or its rest length. The curvature criterion looks to see which way the parent triangle is bending by evaluating the curvature along the two of the external edges. For the transition C0→C4/C5, if the bottom edges have more curvature than the top right edge, then C4 is chosen in preference to C5. Figure 3.9 shows very small differences between different selection criteria. The reason for this is that in most cases there is a faster transition available that will be used in precedence to the selection criteria, such that it will have little effect overall. Therefore we consider the use of the default criterion is perfectly valid, since it requires no additional processing while not negatively impacting on the refinement.

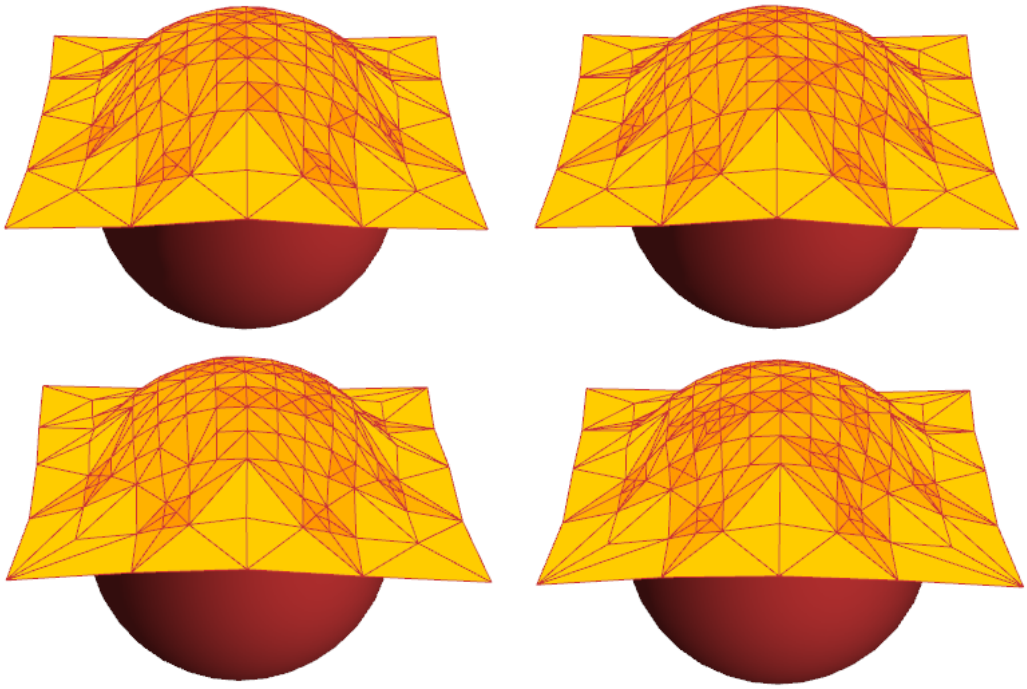

Figure 3.9: A comparison of resulting refinements generated from using different transition selection methods for transitions that are equal work when choosing between configuration 4/5,6,7 and 8/9 for states 4, 5 and 6 respectively. Top Left: Default choice, Top Right: edge length, Bottom Left: edge rest length, Bottom Right: edge curvature. The resulting refinements are very similar, so we consider the use of the default criterion (top left) is perfectly valid and beneficial since it does not require extra processing.

3.3.6 Transitioning Performance

We cannot analyse the performance using a standard mesh easily as we cannot set all triangles in the mesh to be the same configuration at once. Since we must split the external edges to match the configuration we wish to test; this is only possible if all triangles do not share edges with triangles adjacent to them. So for simplicity we have tested the performance on isolated triangles, each with their own unique three edges and three vertices. The vertices' adjacent triangle lists are maintained by the triangles, the cost of insertion, removal and replacing triangles in this list are important. So we additionally insert fake references into the lists for each vertex

to simulate other adjacent triangles so we do not bias the results by using isolated triangles. They only serve to take up space in the list, so the remove and replace operations will be slowed down as the list will have to be searched. We insert 4 null pointers followed by the actual triangle followed by 4 more null pointers, as if there were 9 adjacent triangles around each vertex.

We have repeated the performance tests on the 2000 triangles (this gives 8000 triangles after a 1-to-4 split which is a reasonable maximum amount of triangles expected in the mesh for real-time simulation) and averaged over 200 runs (the number of runs was chosen to give consistent averages where the operating system can introduce some variability); the results do not include the cost of splitting or rejoining edges but only the triangle configuration costs. We pre-allocated enough memory for the tests such that we achieve constant time allocation and deletion costs for triangles, edges and vertices (the memory pool approach we use is discussed later in Section 3.5). Table 3.2 shows the cost of configuring a triangle from starting from SC and the cost to clear the configuration back again in microseconds. The amount of work to configure a triangle is proportional to the number of internal triangles and internal edges; the performance shows three distinct groups i.e. C0-3 with two such triangles and one such edge, C4-C9 with three triangles and two edges and C10 with four triangles and three edges. Next we have performed tests to see the cost of a transition from one configuration indirectly to another via SC. Finally, Table 3.4 shows the performance achieved by implementing direct transitions and they can be compared to Table 3.3 to see the improvement. The improvement in transition costs varied significantly, in the worst case, the transition C0→C7 was only reduced to 99.6% (from 0.313 μs to 0.311 μs). However, the best case shows a large improvement reducing the cost of C9→C1 to 29.6% (from 0.292 μs to 0.087 μs). The average improvement was 62.1%, and the relative performance of all the transitions is summarised in Table 3.5.

Table 3.2: This table shows the average cost (in microseconds) to clear a triangle's configuration into the cleared state (SC) and the cost to configure it back the original configuration from SC.

State	Configuration	Clear (to SC) (μs)	Configure (from SC) (μs)
0	0	0.0467	0.0472
1	1	0.0650	0.1866
2	2	0.0646	0.1842
3	3	0.0648	0.1850
4	4	0.0738	0.2821
4	5	0.0748	0.2863
5	6	0.0751	0.2857
5	7	0.0744	0.2892
6	8	0.0740	0.2850
6	9	0.0739	0.2826
7	10	0.0781	0.3983

Table 3.3: This table shows the average performance (in microseconds) of transitioning from a starting configuration (row) to another configuration (column) indirectly via State 0, e.g C1→C2 takes 0.246 μs.

	0	1	2	3	4	5	6	7	8	9	10
0	-	0.207	0.209	0.208	0.310	0.311	0.320	0.313	0.314	0.314	0.430
1	0.067	-	0.246	0.252	0.394	0.398	0.382	0.384	0.399	0.403	0.507
2	0.068	0.246	-	0.251	0.398	0.406	0.406	0.398	0.379	0.389	0.512
3	0.067	0.252	0.258	-	0.384	0.389	0.409	0.402	0.405	0.408	0.511
4	0.076	0.276	0.273	0.262	-	0.401	0.409	0.389	0.406	0.406	0.527
5	0.077	0.277	0.277	0.264	0.407	-	0.407	0.393	0.394	0.406	0.535
6	0.078	0.260	0.273	0.272	0.391	0.390	-	0.396	0.389	0.405	0.530
7	0.077	0.258	0.271	0.274	0.389	0.397	0.400	-	0.396	0.413	0.538
8	0.078	0.307	0.256	0.273	0.403	0.403	0.417	0.461	-	0.429	0.538
9	0.077	0.292	0.279	0.293	0.448	0.401	0.405	0.397	0.407	-	0.529
10	0.078	0.287	0.290	0.289	0.413	0.408	0.409	0.409	0.408	0.413	-

Table 3.4: This table shows the average performance (in microseconds) of directly transitioning from a starting configuration (row) to another configuration (column), e.g C1→ C2 takes 0.145 μs.

	0	1	2	3	4	5	6	7	8	9	10
0	-	0.198	0.201	0.200	0.306	0.307	0.305	0.311	0.309	0.309	0.428
1	0.060	-	0.145	0.146	0.202	0.266	0.292	0.273	0.276	0.271	0.378
2	0.060	0.145	-	0.153	0.290	0.215	0.224	0.281	0.304	0.297	0.394
3	0.060	0.148	0.147	-	0.287	0.304	0.292	0.215	0.217	0.308	0.388
4	0.068	0.083	0.164	0.159	-	0.151	0.231	0.212	0.246	0.218	0.279
5	0.068	0.144	0.090	0.170	0.151	-	0.192	0.231	0.226	0.242	0.270
6	0.069	0.153	0.082	0.157	0.216	0.180	-	0.148	0.239	0.224	0.277
7	0.069	0.154	0.154	0.085	0.209	0.217	0.152	-	0.194	0.230	0.280
8	0.067	0.140	0.169	0.090	0.227	0.212	0.241	0.190	-	0.161	0.264
9	0.068	0.087	0.166	0.169	0.209	0.220	0.226	0.236	0.158	-	0.348
10	0.067	0.161	0.177	0.177	0.158	0.217	0.163	0.151	0.199	0.161	-

Table 3.5: This table shows the relative cost of directly transitioning (Table 3.4) compared with indirecty transitioning (Table 3.3), expressing the direct methods cost as a percentage of the indirect method. Therefore transitioning from a starting configuration C1 to C2, the cost has been reduced to 58.8% by using the direct method (e.g. reduced by 41.2%).

	0	1	2	3	4	5	6	7	8	9	10
0	-	96.0%	95.9%	96.2%	98.6%	98.7%	95.5%	99.6%	98.6%	98.2%	99.5%
1	89.0%	-	58.8%	58.1%	51.2%	66.9%	76.3%	70.9%	69.3%	67.2%	74.7%
2	88.1%	58.9%	-	60.8%	72.7%	53.1%	55.3%	70.7%	80.2%	76.3%	77.0%
3	90.2%	58.8%	56.9%	-	74.7%	78.1%	71.3%	53.4%	53.6%	75.4%	75.9%
4	88.5%	30.1%	60.0%	60.7%	-	37.5%	56.6%	54.6%	60.6%	53.8%	52.9%
5	89.1%	51.9%	32.4%	64.5%	37.2%	-	47.2%	58.6%	57.5%	59.6%	50.5%
6	88.8%	58.7%	30.1%	57.8%	55.2%	46.3%	-	37.2%	61.6%	55.4%	52.3%
7	90.3%	59.8%	57.0%	31.0%	53.7%	54.7%	38.1%	-	48.8%	55.7%	52.1%
8	85.6%	45.5%	66.2%	33.1%	56.2%	52.6%	57.8%	41.3%	-	37.4%	49.0%
9	89.3%	29.6%	59.5%	57.5%	46.8%	54.9%	55.8%	59.3%	38.8%	-	65.8%
10	86.0%	56.1%	61.1%	61.3%	38.2%	53.2%	39.8%	37.0%	48.6%	38.9%	-

3.4 Edge Refinement and Coarsening

This section describes the refinement and coarsening procedures for edges, and the criteria that govern when refinement or coarsening should be triggered. We always impose a maximum refinement level; this can be changed at run-time as desired and it acts to stop infinite refinement (or too deep) refinement even with no other criteria (or unsuitable ones). All other criteria are specified separately and there can be as many different criteria as needed for a particular application. There are also some simple rules which determine whether the refinement or coarsening is permitted; these are needed to ensure a conforming and error free state for the mesh.

3.4.1 Edge Adaption

Our incremental refinement procedure means an edge may be prevented from refinement by its two adjacent triangles; if the adjacent triangle's parents have not yet undergone 1-to-4 subdivision. Conversely an edge may refine once the triangles' parents are in state 7. We efficiently use the triangles state for the check required, as we previously defined the state of a child triangle whose parent is not in state 7 is set to '-1', i.e. S-1. In this way we do not need to check the parents' states and therefore we do not need to store a parent reference in each triangle. This allows maximum flexibility since edges may check for themselves to see if they are permitted to refine. We perform the maximum refinement level check first since it is the cheapest to compute followed by checking if a split is permitted. Once we know that an edge is allowed to split, we can then check the refinement criteria to see if we should split the edge by creating its immediate children (centre vertex and two edges). The recursive refinement procedure can be seen in Listing 3.6.

```
bool Edge::adapt_AllowSplit(void) const
{
  if(left.T->state == -1)
    return false; // may not split
  if(right.T!=0 && right.T->state == -1)
    return false; // may not split
  return true; // may split.
}
void Edge::adapt_SplitRecursive_criteria(void)
{
  if(isSplit()) // if this is already split, recurse to children.
  {
    AC->adapt_SplitRecursive_criteria(); // first half.
    CB->adapt_SplitRecursive_criteria(); // second half.
  }
  else if(level < adaptiveProperties->adapt_maxLevel)
  {
    if(adapt_Split_isAllowed())  // allowed to split?
    {
      if(adapt_SplitCriteria())   // check criteria.
      {
        /*  create centre vertex and two child edges */
      }
    }
  }
}
```

Listing 3.6: Recursive edge refinement procedure for the splitting of edges.

The coarsening procedure is a little more involved than refining; an edge that is split can be rejoined by simply deleting its immediate children. However, there are some checks needed so we do not destroy detail on subsequent levels and leave the mesh in an inconsistent state. It is logical that an edge cannot rejoin (coarsen) if its children are also split, so we must wait for them to rejoin. The allowable splits and rejoins of a single base (level 0) edge is shown in Figure 3.10, we have imposed a maximum refinement level of 3 in this case. Another way of looking at this is that the edge's adjacent triangles contain triangles on levels two higher than itself (may be as a result of the existence of the edge's childrens' children) and we do not allow a local change of more than one level in a single step. This exposes the fact that there are cases where this will be true, even though the edge's children are not split. Other edges of the adjacent triangle may have split children. The solution is we must check

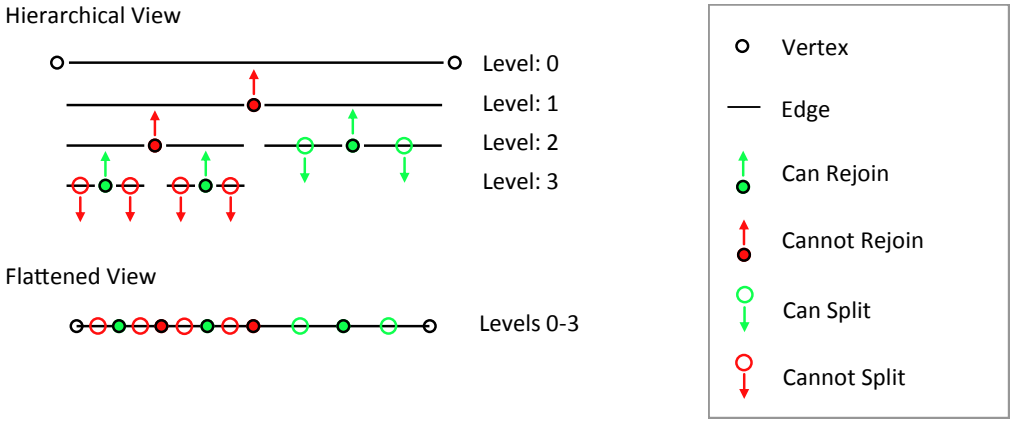

Figure 3.10: This figure shows how a hierarchical view of an edge can be flattened without loss of information for showing permitted edge splits or rejoins. The hierarchical view shows the parent-child tree of edge and vertices, for instance the level 0 edge is split and has 2 child edges and 1 child vertex on level 1. In this example, we have imposed a maximum refinement level of 3; the edges shown on level 3 would be able to be split if we had used a maximum level of 4.

the adjacent triangles' states and their children's state. If the adjacent triangles are in state 7 and if their children are further subdivided ($state > 0$) then the edge cannot be rejoined, see Listing 3.7. To help clarify this, we show some example refinements indicating allowable splits and rejoins in Figure 3.11.

```
1  bool Triangle::adapt_BlockExternalEdgeJoin(void) const
2  {
3    return (state==7 && (inT[0]->state > 0 || inT[1]->state > 0 || inT
       [2]->state > 0 || inT[3]->state > 0);
4  )
5  void Edge::adapt_JoinR_criteria(void)
6  {
7    if(isSplit()) // can't join if not split.
8    {
9      if( !(AC->isSplit() || CB->isSplit()) ) // If childs edges are not
         themselves split.
10     {
11       if(left->T->adapt_BlockExternalEdgeJoin())
12         return; // left Triangle preventing join.
13       if(right->T && right->T->adapt_BlockExternalEdgeJoin())
14         return; // right Triangle preventing join.
15       if(adapt_JoinCriteria()) // may rejoin, check criteria.
16       {
```

```
17          /* join edge, delete children */
18        }
19      }
20      else // either or both child edges are split, recurse.
21      {
22        AC->adapt_JoinR_criteria();
23        CB->adapt_JoinR_criteria();
24      }
25    }
26  }
```

Listing 3.7: Recursive edge coarsening procedure for the rejoining of edges.

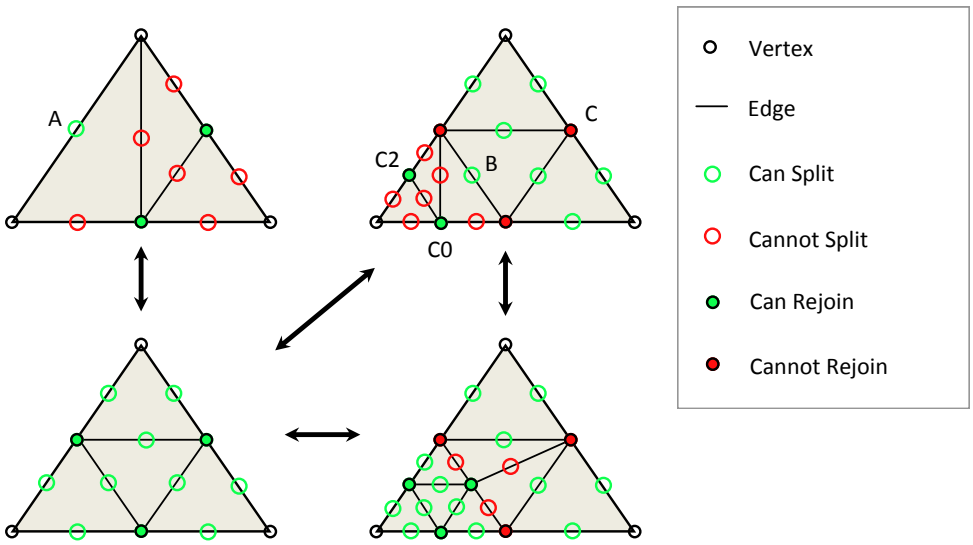

Figure 3.11: Refinement and coarsening of edge must take into account of adjacent triangles internal structure, the examples show the permitted splits and rejoins for different cases using the flattened view introduced for edges in Figure 3.10. Top Left: Most edges cannot be refined further until the last edge A is split and the main triangle has been subdivided into four (State 7) like the example shown Bottom Left. Top Right: Similarly, until edge B is split then edges cannot be refined further, leading to the example Bottom Right. Also edge C cannot be rejoined until edges C0 and C1 are rejoined, leading to the Botton Left example.

3.4.2 Refinement and Coarsening Criteria

The adaptive mesh supports any condition for refinement or coarsening that can be expressed as an edge based criteria. Most conveniently, criteria are implemented

through functions returning a Boolean value. We have three main criteria that we have used for cloth simulation with the adaptive mesh: Curvature, Length and Collision [SLD09].

Curvature

Curvature based criteria are very commonly used with adaptive meshes; there are different methods of measuring curvature. Previously on regular grids, the curvature has been measured simply by the angle at a vertex between two oppositely directed springs which are connected to the vertex [HPH96]. Li and Volkov [LV05] employed estimates of local approximation error using discrete mean curvature at the mesh vertices. Villard and Borouchaki [VB05] approximated curvature at a vertex by the deviation between the surface and the tangent plane by using the largest deviation found between the vertex normal and an adjacent face. Wang [Wan02] measured the angle between adjacent face normals. We must define curvature in a way suitable for our edge-based criteria, the most efficient way to do that is to measure curvature along an edge.

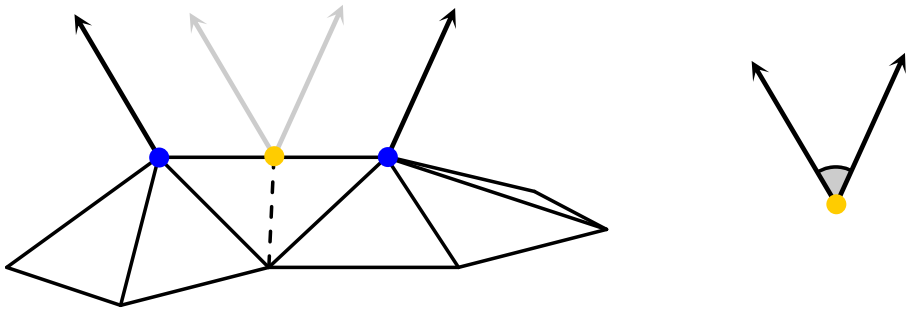

Figure 3.12: Left: Normals for two vertices (blue) shown for a highlighted edge on part of a mesh, the central vertex (yellow) is shown as if the edge were split. Right: Curvature is defined locally as the angle between the two normals, calculated by the dot product between the vectors.

We define curvature locally to each edge by the angle between the two surface

normals of the two vertices at the end of each; this is shown in Figure 3.12. When the angle is greater than an adjustable threshold, the edge is split; conversely when it is less than another threshold the edge can be rejoined. This permits an edge to have a curvature between the two values and remains unchanged, giving a tolerance to the threshold. It prevents the edge from flipping between split and joined states if the curvature oscillates back and forth. However, too large a difference between the two thresholds is not good, edges may remain coarse or refined for too long. The larger the difference, larger oscillations are ignored but edges may remain unchanged for too long. This can be implemented by the two functions shown in Listing 3.8.

```
1 bool Edge::adapt_SplitCriteria_Curature(void) const
2 {
3   return acos(dot_product(A->n,B->n)) > adaptiveProperties->
        criteria_split_curvature_angle;
4 }
5
6 bool Edge::adapt_JoinCriteria_Curature(void) const
7 {
8   return acos(dot_product(A->n,B->n)) < adaptiveProperties->
        criteria_join_curvature_angle);
9 }
```

Listing 3.8: Edge Curvature Criteria

Length and Edge Buckling

The almost inextensibility of cloth is often strived for in simulations, the type of fabric plays a big part in both extension and compression. For instance knitted wool's loose stitches and holes provide slack allowing limited extension of the cloth with small forces beyond which cause stretching and eventually breaking of the fibres occurs. This can be simulated by non-linear springs and forces using real measured data such as discussed in the literature review chapter (see Chapter 2). Compression is more interesting in some ways, because cloth buckles quickly under compression and it is the way in which wrinkles are created giving cloth its tell-tale look. Stretching cloth

in one direction also often causes compression in the other direction.

It is clear that coarse meshes cannot possibly produce fine wrinkles; the resolution of a mesh greatly changes its ability to model folding and buckling. Thomaszewski *et al.* [TWS06] noticed that if using a simple bending model, the fold patterns were different to real samples, but also the patterns changed completely with a finer mesh with the simple model. Their simulation method using finite elements was much better and produced a very good likeness compared to the samples but its computational cost limit it to only offline use. Volkov and Li [VL03] found that fine wrinkles observed in the non-adaptive simulation were missing from their adaptive one, and that these fine wrinkles were attributed to buckling behaviour which cannot be detected by the curvature based criterion but gave no solution. Mujahid et al. mentions the possible use of stretching as refinement criteria, though they did not link this to any particular use. In our own opinion we feel that refinement for stretched edges is not advantageous, greatly stretched edges apply large forces to the particles would be less numerically stable if refined. That is since more refined particles have lower masses typically causing them to experience larger accelerations (i.e. from $f = ma$), which without reducing the time step this will manifest as extra instability. However, reducing the time step has the implication that the cloth will need to be updated more frequently thus increasing the computational cost.

However, refinement on compression is a way to allow coarse meshes to buckle and simulate wrinkles and we applied this to draping of cloth in [SLD09]. Compression of an edge is measured by its current length compared to its rest length expressed as a percentage, e.g. 50% implies an edge has been compressed to half its undeformed size. Buckling behaviour is therefore enabled by the split edge's central vertex being created and then is free to move. This can be implemented as shown in Listing 3.9, like curvature we use two threshold values. It shows the case where caching edge

length can be beneficial to performance since it can be reused here.

```
bool Edge::adapt_SplitCriteria_Length(void) const
{
  float stretch = length(B->p - A->p) / restLength; // 100% = 1.0f
  return stretch < adaptiveProperties->criteria_split_length_percent
}

bool Edge::adapt_JoinCriteria_Length(void) const
{
  float stretch = length(B->p - A->p) / restLength; // 100% = 1.0f
  return stretch > adaptiveProperties->criteria_join_length_percent;
}
```

Listing 3.9: Edge Length Criteria

Collision

Collision can also be used as a refinement criteria, this is advocated when performing fast point-object collision checks. Point-object collision detection and response when combined with a coarse mesh and a relatively small object can result in the cloth falling through the object or the object sticking through the cloth. A solution to this is to perform full polygon-polygon collision detection and response but this is much more computationally expensive. If the fast point-object detection is to be used, the mesh must have enough vertices required to resolve the collision correctly. We therefore want to cause refinement in the adaptive mesh around the area of the collision, even if the curvature is not sufficient to cause edge splitting. We can refine edges that are deemed in collision and project its new centre vertex on to the surface of the object, see Figure 3.13. The criteria is very simple, the complexity is contained within the collision detection approach. In this work we deem that an edge is in collision when its centre (the location where the new vertex would be created) is inside of the object. At first glance this method appears to suffer from stretching like in [HH98]; the sum of two child edge lengths will be greater than their parent's length. However, in our case the new vertex is fully integrated into the cloth simulation successive iterations

will restore the edge lengths and the net effect is no different than when moving a vertex for any other collision. Additionally this criterion is combined with its own maximum level, so we can limit the refinement depth caused by collisions as needed while still allowing refinement to higher levels triggered by other criteria.

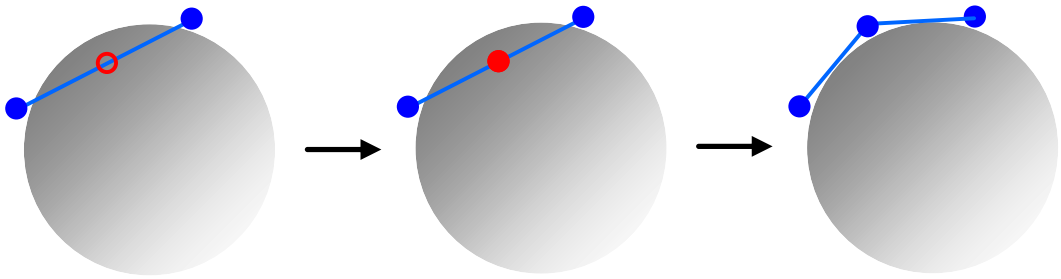

Figure 3.13: When a collision is detected with an edge, we can trigger refinement in the edge to allow the cloth to approximate the objects shape better. Left) collision is detected, Middle) edge is split and new vertex is created, Right) the vertex is projected out on to the surface of the object.

3.4.3 Conflicting Criteria

When using multiple criteria to control the refinement and coarsening of the mesh, problems can arise from these conflicting. For example, an edge may be in collision and therefore should be split, but the curvature criteria may determine that the edge should not be split and would therefore be rejoined in the next step. A simple way to deal with this is to impose a delay after an edge is split before it is allowed to rejoin [SLD09]. We found this was quite effective in most regards, but it did not remove the problem entirely. The edges could still rejoin and then immediately split every time the rejoin wait was finished. In [SLD10], the collision criterion was the main culprit causing the most visual flickering. So we checked that an edge was not in collision before rejoining, such that it would not immediately be split on the next step. The cost of additional collision checks were expensive so we kept the rejoin wait

count as a way to spread out the coarsening checks. Each time a rejoin is attempted, the rejoin wait count is reset and then another attempt at rejoining is not tried until the count exceeds a user settable number of steps. This setting can be determined experimentally, we have found a wait count of 10 adaptive steps works well when updating the mesh at 30Hz. In this case an edge (split and) rejoin may only occur a maximum of 3 times a second for a balance of greater performance and less flickering. Simulation performance can be hurt by using a too long setting; the mesh may be unnecessarily delayed from coarsening.

In general, it can be seen that to eliminate the problem fully, we must check all refinement criteria (not just collision) and verify that the edge should not remain split. If the refinement criteria indicates that the edge should be split (as if it was rejoined) then we do not need to check the coarsening criteria and therefore we should not rejoin the edge. This also permits a new rejoining criterion, which rejoins an edge whenever possible (i.e. as soon as not blocked by splitting criteria) for if specific rejoining criteria are not required.

3.4.4 Implementation

There is often more criteria defined than we wish to use in a simulation, e.g. we may wish to disable criteria based on collision for the simulation of a flag where there are no colliding objects nearby for optimal performance. This presents either a problem of performing an extra test to see if a criterion is enabled (once per edge per criteria); or a programming maintenance problem with replication of the edge splitting and edge joining functions with different combinations of criteria. We make use of C++ templates in combination with bit masks to provide us with a solution that combine criteria functions, and then we rely on the compiler to automatically generate different versions (such that disabled criteria are optimised away). We can order the calls to

the different criteria functions with the least costly first; typically collision tests last for best performance. We show how several edge refinement criteria can be combined using a template in Listing 3.10.

```
const unsigned Criteria_Split_Length    = 0x0001; // binary: 0001
const unsigned Criteria_Split_Curvature = 0x0002; // binary: 0010
const unsigned Criteria_Split_Collision = 0x0004; // binary: 0100

template <int Criteria>
bool Edge::adapt_SplitCriteriaT() const
{
  return ((Criteria & Criteria_Split_Length) &&
     adapt_SplitCriteria_Length()) ||
   ((Criteria & Criteria_Split_Curvature) &&
     adapt_SplitCriteria_Curvature()) ||
   ((Criteria & Criteria_Split_Collision) &&
     adapt_SplitCriteria_Collision());
}

typedef bool (Edge::*emptr)(void); // Edge member function pointer

emptr Edge::adapt_SplitCriteria_Curvature() const
{
  // curvature
  return &Edge::adapt_SplitCriteriaT<Criteria_Split_Curvature>;
}

emptr Edge::adapt_SplitCriteria_LengthCurvature() const
{
  // length and curvature
  return &Edge::adapt_SplitCriteriaT<Criteria_Split_Length |
     Criteria_Split_Curvature>;
}
```

Listing 3.10: Edge criteria template and example usage for combining criteria

Then we need only instantiate the template for used combinations of criteria and provide a facility to return a member function pointer at run-time based on enabled and disabled criteria. This is repeated for edge joining criteria in the same way.

3.5 Memory Management

Memory considerations are often overlooked in real-time simulations and often pre-computation is preferred wherever possible. However, adaptive meshes require a large amount of memory to store subsequent levels even for small base meshes because the hierarchy must be stored to enable efficient incremental refinement and coarsening. For instance, the number of triangles increases by four each level which adds onto the total of previous levels, one triangle is subdivided into 64 on level 4 but requires 85 triangles in total in the hierarchy (1+4+16+64). Real-time constraints could force the use of pre-allocation for all levels that may be needed; this is because the operating system's managed dynamic memory imposes a relatively large overhead for creation and deletion. In the case of C++, a heap structure is used to manage variable sized allocations of memory. The overhead is particularly troublesome if allocating many small objects one by one. Allocation performance is improved by allocating fewer objects, since we can use fixed sized triangle lists in the vertices and have combined directed edge sides into a single edge structure then we only have three objects to dynamically allocate (vertices, edges and triangles). The process of mesh refinement and coarsening is extremely likely to also cause memory fragmentation, just as if we were to allocate and free many different sized objects in a random order.

3.5.1 Memory Pools

The purpose of the adaptive mesh is to allow refinement only in areas that require more triangles through the use of user definable criteria. There is a limit on the overall number of triangles in a piece of cloth that can be simulated and rendered in real-time, and the refinement criteria should be chosen accordingly. We can make use of this to achieve large memory savings, by only pre-allocating the maximum memory needed by the mesh over the course of a real-time simulation. However, where that memory is

needed changes during the simulation so it needs to be dynamic in some regards so we employ memory pools. A memory pool is also known as fixed-size-blocks allocation, where a number of blocks with the same size are pre-allocated. If the blocks are the same size as the objects, efficient use of space is achieved so objects can be tightly packed with no wasted space. For this reason we use three memory pools, one for each of the core building blocks of the mesh: triangles, edges and vertices. We have overloaded C++'s operator for new and delete so that the pools can be used transparently (see Listing 3.11). As the mesh is refined and coarsened the pools are used dynamically, in the same way but with much less overhead compared to system managed memory. Several memory pool implementations can be defined and then selected between at compile time or disabled entirely so that comparisons can be made easily.

```
1
2  template <typename T>
3  class MemoryPool
4  {
5  public:
6    void * allocate(size_t bytes);
7    void free(void * p);
8    /* rest of implementation */
9  };
10
11 class Vertex
12 {
13 public:
14   static MemoryPool<Vertex> memoryPool; // Static global memory pool
         for vertices.
15
16   void * operator new(size_t bytes)
17   {
18     return memoryPool.allocate(bytes); // return a pointer from the
          memory pool.
19   }
20   void operator delete(void * p)
21   {
22     memoryPool.free(p); // pass the pointer to the memory pool to be
           freed.
23   }
24 };
```

Listing 3.11: Memory Pool and Vertex usage.

3.5.2 List based Memory Pool

We use a simple list approach that allows constant time access to the pools, a list of pointers to unused objects is maintained. When an allocation is requested, a pointer to a single object is taken from the list and returned. If the list is empty, we allocate the memory for many objects at once and add them all to the list. When an object is deleted, we place its pointer back into the list. The number of objects to allocate at once is configurable; we have found allocating enough memory for 250 objects at a time to work well. If much fewer than 250 objects at allocated at once the performance advantages of the memory pools are noticeably reduced. However, allocating many more objects at once gives little improvement in performance while increasing the amount of wasted memory space. In order for the pools to automatically shrink in size, additional overheads would be required to detect if a complete unused block is free. There is no guarantee that a block would ever become free especially with larger block sizes this become more unlikely. We have deemed this overhead for dynamically shrinking inacceptable, so the memory pools are not designed to decrease in memory usage. It is difficult to predict the exact requirements for a complex simulation especially with collisions; so the ability to grow dynamically with minimal overheads is favoured compared to manually selecting the size. Experimentation can achieve very good estimates, but requires the simulation to be run once with large pools to find maximum usage in a kind of pre-computation step which defeats the purpose of dynamic memory.

3.6 Cloth Simulation Integration

As previously mentioned the edge-based adaptive mesh was designed as a mass-spring network. The cloth simulation proceeds by firstly calculating spring forces and applying them to the vertices. The accumulated force vector of each vertex is used to calculate the acceleration of the particle and numerical integration is performed. Any collisions are then processed and finally the surface normals are recalculated.

3.6.1 Material Coordinates

During refinement we require access to un-deformed quantities of the mesh such as lengths and areas. Some of these values can be found easily by interpolation, such as the rest lengths of new edges when an edge is split into two equal pieces. However, we wanted something that we could fall back on in any situation with minimal overheads so we do not need rely on special cases. Most un-deformed cloth is flat and clothing is typically made from flat cloth panels so the natural solution is to store 2D undeformed coordinates in the mesh, i.e. material coordinates. The material coordinates are fixed during simulation, so no matter how the cloth is deformed we can calculate undeformed lengths and areas between any points.

One may think of material coordinates being like texture coordinates, texture coordinates map 2D textures on to 3D objects and material coordinates map 2D undeformed coordinates onto 3D deformed objects. However, texture coordinates are typically created during texture mapping by projecting vertices from a 3D object onto a simple developable surface such as planes, cylinders and spheres. Depending on the object and the suitability of the projection method this can create much distortion, lengths and areas on the texture will not map uniformly to lengths and areas on the object. Material coordinates need to be very precise for accurate simulation, so we have to create flat cloth meshes in 2D and directly use the 2D coordinates as the

material coordinates. The material coordinates are automatically calculated for new vertices during refinement, which map the newly created vertex to its correct relative undeformed position in 2D.

3.6.2 Spring Forces

Every edge in the adaptive mesh is a spring that applies directed forces to the two end masses. An edge has a rest length that is the undeformed length calculated from the material coordinates. At any instant, the force can be calculated by Hooke's law, $\mathbf{f} = -k\mathbf{x}$ where k is the spring constant and $\mathbf{x}$ is the extension. The extension is calculated from the difference between the current length and the rest length.

The spring constant must be recalculated as the mesh refines for each edge. Hooke's law type springs can be combined in series and parallel, the equivalent spring constants, k_p and k_s respectively are given by:

$$k_p = \sum_{i=1}^{n} k_i \tag{3.6.1}$$

$$\frac{1}{k_s} = \sum_{i=1}^{n} \frac{1}{k_i} \tag{3.6.2}$$

$$k_s = \frac{k}{n} \text{ (for } n \text{ identical springs)} \tag{3.6.3}$$

We define a global spring constant for a piece of cloth, k_g with units Nm^{-1} for a $1m$ length spring. From Equation 3.6.3., for an edge of length, l we can calculate k_e by dividing the global spring into n (now fractional) pieces:

$$k_e = k_g n \text{ (where } n = \frac{1}{l} \text{)} \tag{3.6.4}$$

This allows the calculation of spring constants for both the initial mesh and edges in the mesh resulting from splitting, next we must account for new edges introduced

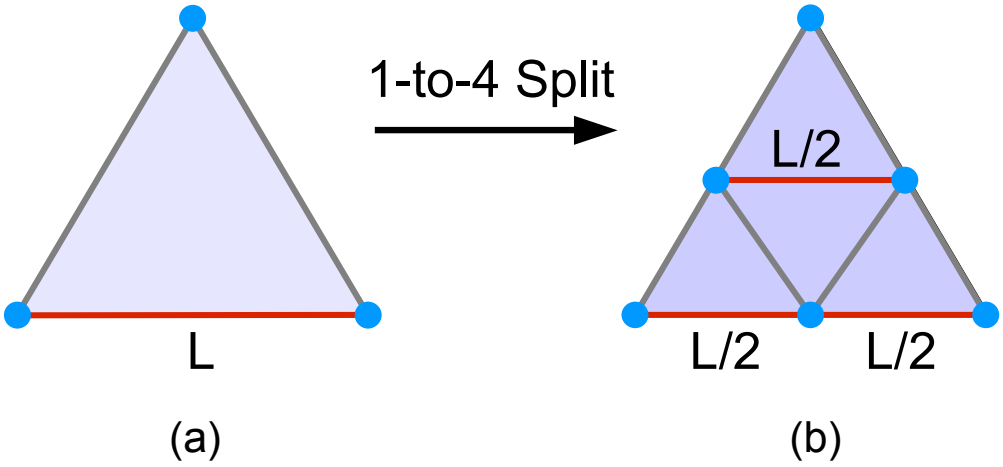

Figure 3.14: A triangle (a) is split into four (b), each edge is split into two equal lengths and for each edge direction (one such direction is highlighted in red), a new edge is introduced with the same length.

into the adaptive mesh. A triangle undergoes a 4-to-1 split each time it is fully refined (See Figure 3.14). Resolving along any of its original three edge directions, an edge with spring constant, k_e is bisected (two springs in series) and an additional spring of equal length is added in parallel. Following Equation 3.6.4. each spring will be assigned the constant $2k_e$ as the length has halved. This introduces undesirable extra stiffness, the new equivalent spring constant including the extra spring in parallel is equal to $\frac{3}{2}k_e$ when it should still be k_e. Simply, we must multiply Equation 3.6.4. by $\frac{2}{3}$ for each level the mesh is refined. Therefore, the corrected spring constant for any edge in the mesh at any level is given by:

$$k_e = \frac{k_g}{l}\left(\frac{2}{3}\right)^{level} \tag{3.6.5}$$

3.6.3 Bending Forces

Bending forces are important for realism, because all materials have some resistance to bending but these forces are much less than the in-plane forces preventing stretching. Bending forces in mass-spring networks are commonly implemented by springs but they do not have to be. We have implemented bending springs and another simple method based on curvature. The benefit of using a curvature based force is that no lengths are needed, whereas the springs require the undeformed lengths to be updated as the mesh refines.

Bending Springs

Bending springs stretch across an edge, connecting the adjacent triangles opposite vertices. During mesh refinement an edges adjacent triangle can change often, even if the edge itself is unchanged because of retriangulation. A triangle is responsible to keep the adjacent vertices of its edges updated if they are being cached by the edge, otherwise edges will find them by using connectivity information. A triangle is also responsible for invalidating the undeformed length of a bend spring, signalling to an edge that it must recalculate it from the material coordinates.

Bending Elements

Bending forces can be introduced through simple bending elements, made from an edge with vertices **A** and **B** shared by its two adjacent triangles with surface normals, $\mathbf{N}_{left}$ and $\mathbf{N}_{right}$, and from the triangle's two opposite vertices **L** and **R** (see Figure 3.15). Conceptually like bending springs these are separate elements, the edges serve a dual purpose as bending elements just as they also serve as springs. These bending elements simulate a simple bending force, producing a surface normal directed force and uses a global bending constant, k_b, for the whole garment which is expressed as in newtons per angle (radians). The force is zero when the triangles are perfectly flat,

smoothly rising to a maximum when the angle is 180°:

$$f_{mag} = k_b . \frac{cos^{-1}(\mathbf{N}_{left} \bullet \mathbf{N}_{right})}{\pi} \tag{3.6.6}$$

The forces for the four vertices are calculated such that the resultant force on each triangle is zero, but each triangle experiences a turning force that flattens the pair:

1: $\mathbf{A}_{force} = \mathbf{N}_{left} . \frac{f_{mag}}{2} + \mathbf{N}_{right} . \frac{f_{mag}}{2}$
2: $\mathbf{B}_{force} = \mathbf{N}_{left} . \frac{f_{mag}}{2} + \mathbf{N}_{right} . \frac{f_{mag}}{2}$
3: $\mathbf{L}_{force} = -\mathbf{N}_{left} . f_{mag}$
4: $\mathbf{R}_{force} = -\mathbf{N}_{right} . f_{mag}$

The forces must be applied in opposite directions when the bending element is folded the other way in order to flatten the pair, we detect this by checking if $(\mathbf{N}_{right} \bullet (L - R)) > 0$.

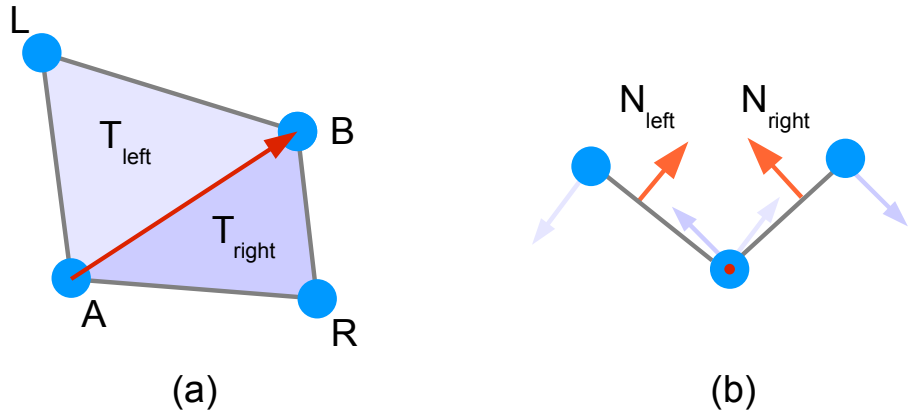

Figure 3.15: a) A bending element is made from two adjacent triangles and an edge (red), with shared edge vertices **A** and **B** and opposite vertices **L** and **R**. b) Forces are applied following the normal directions for each triangle ($\mathbf{N}_{left}$ and $\mathbf{N}_{right}$).

3.6.4 Verlet Integration

Numerical integration is an important part of physical simulations, allowing the simulation to be evolved over time integrating Newton's equations of motion. As discussed in the literature review (see Chapter 2.1.1), Verlet integration is used frequently in molecular dynamics for the trajectories of particles and has since become popular in computer graphics. There are many reasons for its popularity, not the least of which is that it offers greater stability than the Euler method but at little additional cost. Verlet is a 2nd order method compared to Euler which is only a 1st order method with regard to global error; but also since velocity can be implicit in Verlet, the velocity and position to cannot come out of sync so it is more stable. So we use the form where the velocity is incorporated as the difference between the previous and current positions, i.e. how far a particle has travelled in the last time step, Δt. The new position of a particle is calculated by:

$$\mathbf{x}_{t+\Delta t} = \mathbf{x}_t + (\mathbf{x}_t - \mathbf{x}_{t-\Delta t}) + \frac{1}{2}\mathbf{a}_t(\Delta t)^2 \tag{3.6.7}$$

The particle experiences a total force, f and has mass, m, from Newton's 2nd law, acceleration is given by:

$$\mathbf{a}_t = \frac{f}{m} \tag{3.6.8}$$

Simple damping can be performed to emulate air resistance by removing a fraction of the velocity each step; this is done by moving the previous position closer to the current position. Also it is useful to be able to fix a particle in one place; we used it to simulate hanging cloth. A simple way to prevent a vertex from moving is to store and use the inverse (reciprocal) mass for Equation 3.6.8 and multiplying with

the force instead of the force being divided by the mass. Therefore a particle may be fixed in place by giving it an artificial inverse mass of zero (in theory infinite mass); the equation will then calculate zero acceleration. However, if the particles are to be moved by other means than forces, then the particles should be checked to see if they are fixed before moving them.

3.6.5 Collision Constraints

Verlet integration allows a very simple collision response that is fast to calculate, by projecting the particles' positions outwards onto the surface of a collided object. The velocity is automatically affected by moving the current position, and can be further modified if needed by changing the particles' previous position. Although the response will not necessarily be realistic, collisions are handled in a plausible way and particles will appear to slide over surfaces. The adaptive mesh can generate enough vertex density that the cloth will not fall through objects. Vertex only collision detection and response has implications for newly created vertices (from edge splits). The collision detection method cannot always rely on the previous position of a new vertex being in a collision free state, as it may have been created within an object (although the collision is immediately resolved). Also, if the previous position is modified to change the velocity, that act may invalidate its collision free state.

3.6.6 Edge Length Constraints

The almost inextensibility of cloth is an important part of the realism when simulating it. The use of a very strong spring constant to combat this is problematic due to the numerical instabilities it can introduce by creating too large restorative forces that impart very large accelerations on particles which leads to progressive overshooting of particle positions and divergence of the solution. A smaller integration time step or more complex methods could be used but these are typically too costly for real-time

work. As previously disused, Provot [Pro95] presented an ad-hoc inverse dynamic method to apply to super-elongated springs, it works by assuming the spring direction is correct but not the distance between the two ends. When used with non-uniform meshes such as adaptive meshes, the approach must be modified to preserve the edge's centre of mass. For each edge in the mesh, a correction is applied to its end particles if it has stretched beyond a certain percentage of the length, *MaxStretch*, (See Figure 3.12 for the basic code). Additionally, we check if particles are fixed, if one is fixed then the correction is applied to the non-fixed particle ignoring relative masses.

```
1  void Edge::ConstrainLength(void)
2  {
3    vector3 vec( B->p - A->p ); // vector from A to B.
4    float extens = Length(vec);
5    float stretch = extens / restLength;
6
7    if(stretch > clothProperties->physics_MaxStretch)
8    {
9      float correction = (extens - restLength*clothProperties->
           physics_ MaxStretch)/extens;
10     float massA = A->getMass();
11     float massB = B->getMass();
12     float totalMass = massA + massB;
13     A->p += vec*(correction*massA/totalMass);
14     B->p -= vec*(correction*massB/totalMass);
15   }
16 }
```

Listing 3.12: Edge length constraining procedure, based on the work of [Pro95] modified for unequal masses

The order of which edges are processed does affect the convergence, we do as Provot did and thus perform it in no-particular order (i.e. the order that the edges happen to be stored in the mesh is followed). order. However, Ozgen and Kallman's [OK11] technique of following the direction of gravity may be better but it was limited to only static orderings for performance so it would not be suitable in an adaptive setting.

3.6.7 Decoupled Simulation

In our use of the edge based adaptive mesh, we have found often there are very few changes between updates such that the majority of refinement and coarsening costs are the criteria calculations. This is particularly the case with finer meshes which require small time steps for numerical stability where it is typical for the number of triangles in the mesh to only change by less than 20 in a single update. Also finer meshes have more edges for which the criteria must be checked. In order to not waste resources in this circumstances we proposed to completely decouple the simulation updates from the adaptive updates [SLD10]. In doing so, we can run the simulation at a higher rate, for example, the simulation can be run at 120Hz and the adaptive refinement can be updated at 30Hz. This is entirely up to the user to decide, since it is highly dependent on the situation. For instance, fast moving cloth will require a higher mesh update rate to keep the topology acceptably up to date compared to slowly deforming cloth. What is deemed acceptable may vary, and depend on distance to the cloth and the relative importance of the cloth compared to other objects in the in the scene.

3.6.8 Data Structure Traversal

Our adaptive mesh's hierarchy enables an efficient algorithm for refinement but we should consider if it imposes an overhead in other situations. The hierarchical data structure must be recursively traversed many times for many operations even when we only need access to the highest level in use (as used for simulation and rendering). This cost can be reduced by constructing a temporary list of references (pointers) to triangles, edges and vertices each time the adaptive mesh is updated. It is important to verify that a saving is achieved when factoring in the cost of the list creation, and also the list will use up some memory (pointers of four bytes for every vertex, edge

and triangle). The lists may be particularly effective with a decoupled simulation since they remain valid for a number of simulation steps. They can be used for face and vertex normal computation, edge spring forces, numerical vertex integration and edge length constraining. Table 3.6 shows our results of this approach from [SLD10].

Level	Triangles	Simulation	List Creation	Simulation with List
0	128	0.0498	0.0015	0.0339
1	512	0.2143	0.0066	0.1362
2	2048	0.8804	0.0332	0.5735
3	8192	2.9647	0.1362	2.4078

Table 3.6: Simulation times (in milliseconds) for a single simulation step using recursive traversal and temporary lists updated each adaptive step. A base mesh of 128 triangles is used and the cost of list creation can be seen.

3.7 Visualisation and Rendering

In this section we describe rendering techniques to visualise the adaptive mesh and render the cloth. The visualisation of the adaptive mesh can assist the users of the adaptive mesh greatly; choosing suitable edge criteria and their values is not easy. Statistics can be displayed to the user, just as showing the cost of adaption and the number of triangles in use but a visual representation can be more powerful. For example, the mesh may be too deeply refined in areas due to edge length criteria but not deeply enough in other areas due to curvature, the easiest way to check for this is visually with the ability to enable and disable criteria and watch their effect. We describe a number of ways to visualise the adaptive mesh, and finally we describe how we render the final mesh with lighting using OpenGL.

Mesh Levels

The most helpful rendering of the adaptive mesh is one showing the mesh's levels. We render each triangle of the mesh colour coded by its level; this gives a clear illustration of the depth of recursions. We use a simple orange to yellow colour scheme, where level 0 triangles are colour bright yellow and each successive level is rendered in a darker colour. We achieve this by the RGB triple $(1, 1 - (0.1 * level), 0)$, such that $level0 = (1, 1, 0)$, $level1 = (1, 0.9, 0)$ and $level2 = (1, 0.8, 0)$ etc. Edge levels may be drawn as simple lines, in a similar way; we must draw them if we wish to clearly distinguish between adjacent triangles. Z-fighting can occur between the rendered edges and the triangles; to combat this we must offset the triangles further away in the depth buffer so the edges are drawn clearly over the triangles. This can be achieved simply using OpenGl's polygon offset facility.

Unlike triangles and edges where they are effectively replaced by their children

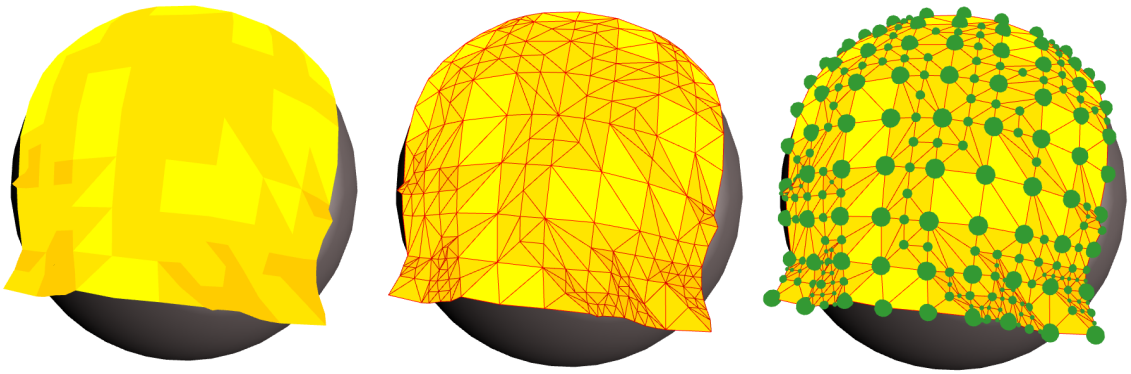

Figure 3.16: An adaptively refined mesh lying on a sphere, Left: mesh triangles are rendered and coloured according to their level, Middle: coloured edges are added, Right: Vertices are rendered as small spheres, level 0 at the largest and size is halved with each subsequent level.

during refinement, vertices of all levels are in use during the cloth simulation. Visualising the levels of vertices provides a way to see the corners of triangles on each level. Instead of colour coding vertices, we render them as scaled spheres, so as not to be overwhelmed by the different colours. A level 0 vertex is drawn with an appropriately sized radius (we use 1cm) and then we half the radius with each level. This is better than a linear scale because the level 0 vertices are clearly visible, but the higher level vertices are not drawn with such large spheres that occlude the edges and triangles. Figure 3.16, shows an example of an adaptive mesh lying over a sphere which is then rendered using combinations of these three techniques.

Visualising the Hierarchy

It is hard to relate 2D representations of the adaptive mesh to the 3D hierarchy; however, the hierarchy can be illustrated in 3D quite easily. We render the hierarchy using 2D material coordinates where each level is rendered on a different parallel plane which is offset vertically by a distance by the previous level. We connect the planes together with lines between common vertices shared between the levels. Higher levels are coloured darker, and also triangles which have children are coloured darker to

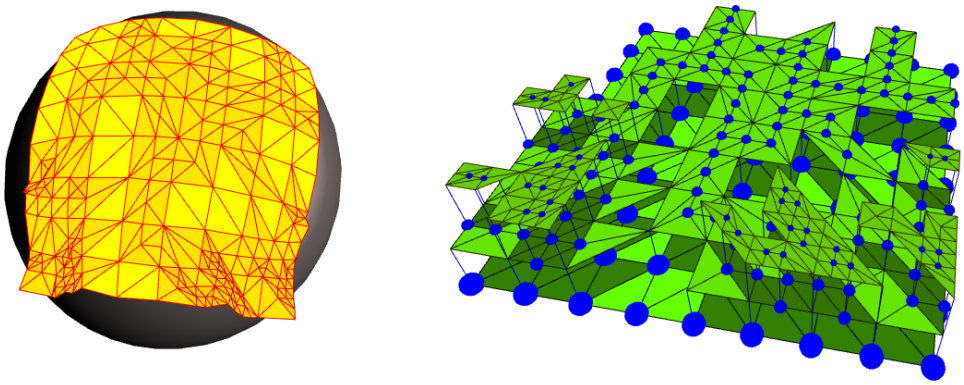

Figure 3.17: Left: A top down view of an adaptively refined mesh lying on a sphere, Right: its corresponding hierarchy is illustrated in 3D.

give a shadowing effect, see Figure 3.17.

3.7.1 Rendering and Lighting

In order to render the adaptive mesh on the GPU using OpenGL we must upload the data to the GPU every frame since the mesh is constantly changing. The simplest way to do this is to use OpenGL's immediate mode where vertex data is specified individually by individual commands. However, more efficient methods exist such as Vertex Arrays (VA) and Vertex Buffer Objects (VBO). VAs use client (CPU) memory data, and the display driver handles copying the data to the GPU when a draw command is issued. In the case of VBOs, the GPU maintains a buffer in its own memory, users must upload the data. VBO's achieve the best rendering performance when the data is not changed and can be reused for rendering, since the transferring of data from the GPU to the GPU is a significant bottleneck. One drawback to edge-based adaptive mesh is that it does not expose triangles in the most efficient way for rendering; on the other hand static geometry can be arranged very efficiently for rendering. So we cannot upload adaptive triangle data directly because the data is not contiguous in memory. We have achieved the best performance by copying triangle

vertex data from the mesh to an intermediate tightly packed array in order to get the best transfer rate to the GPU using Vertex Arrays. The array stores interleaved (positions and normals) vertex data for each triangle and it therefore stores three times as many vertices as there are triangles. The vertex data is duplicated for shared vertices, this is not ideal, an index array could be used. However, the cost of creating and maintaining a dynamic index array is prohibitively expensive for the adaptive mesh (triangles store pointers to their vertices, there are no inbuilt indices). Once the array is copied, we stream and render it on the GPU using the code shown in Listing 3.13.

```
1
2   struct Vertex
3   {
4     GLfloat normal[3];
5     GLfloat position[3];
6   };
7
8   void render(Vertex* data, unsigned int vertexCount)
9   {
10
11    // tell OpenGL where the data starts.
12    glEnableClientState(GL_NORMAL_ARRAY);
13    glNormalPointer(GL_FLOAT,sizeof(VERTEX), &data[0].normal[0]);
14    glEnableClientState(GL_VERTEX_ARRAY);
15    glVertexPointer(3, GL_FLOAT,sizeof(VERTEX), &data[0].position
          [0]);
16
17    // draw the array as triangles.
18    glDrawArrays(GL_TRIANGLES, 0, vertexCount);
19
20    glDisableClientState(GL_NORMAL_ARRAY);
21    glDisableClientState(GL_VERTEX_ARRAY);
22  }
```

Listing 3.13: Triangle vertex data streaming to GPU using OpenGL

We employ per-pixel lighting with the Phong lighting model using GLSL (OpenGL Shading Language) shaders, where the material is specified by standard ambient, diffuse and specular colour terms and combined with a shininess constant which alters the size of the specular highlights. Many different looks and feels of cloth-like material

can be achieved, see Figure 3.18 for some examples.

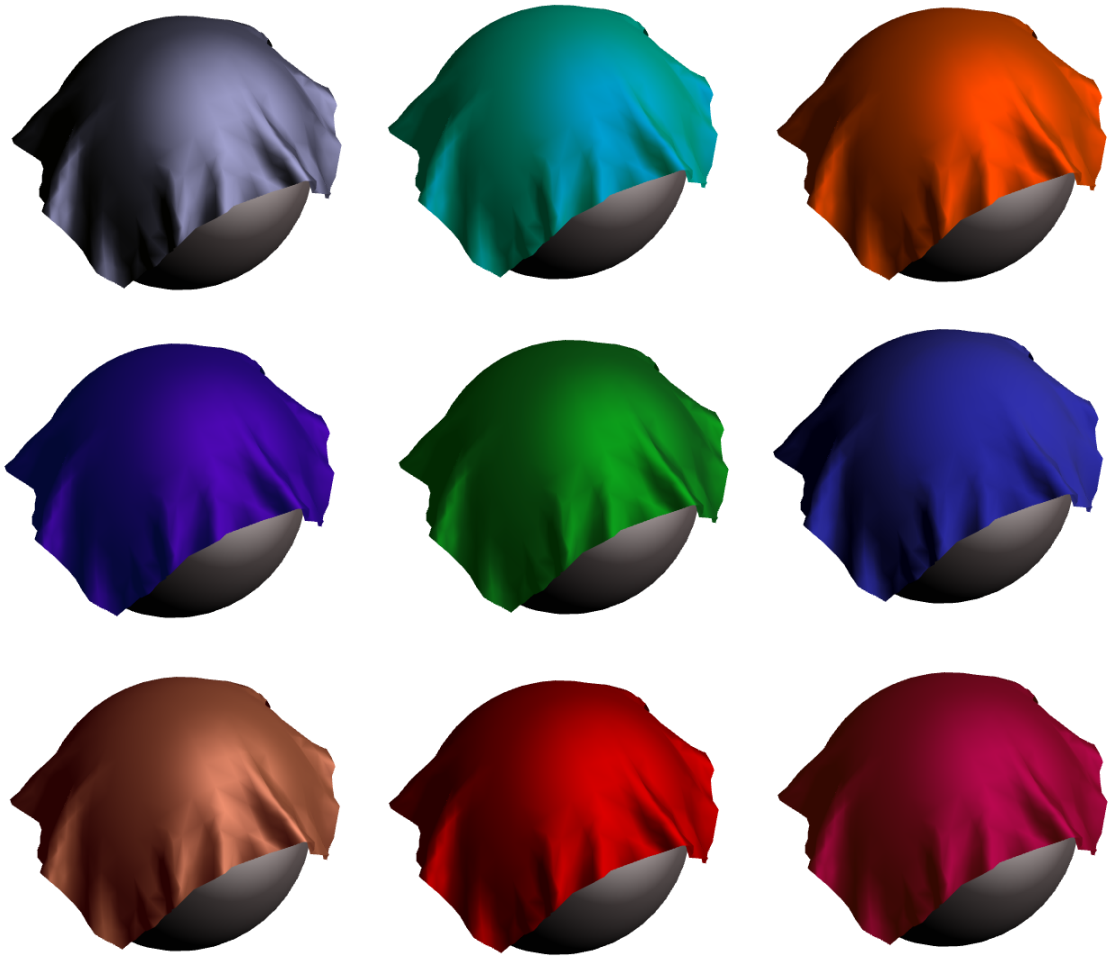

Figure 3.18: This figure shows a selection of different material colours applied to the render of a piece on cloth lying on a sphere, they are rendered using per-pixel lighting with a Phong lighting model shader.

3.7.2 Texturing

Plain rendered cloth, even with per-pixel lighting can look unrealistic and very much like plastic, real textiles are not so smooth. They can feature colourful patterns and pictures, ranging from modern printed t-shirts often sold as souvenirs with illustrations of holiday destinations to more traditional patterns such as check and tartan.

We can use texturing to add these extra details to cloth, but texture mapping requires texture coordinates. However, we have been able to use the 2D material coordinates we store in the adaptive mesh for simulation purposes also for texturing purposes. We can therefore perform simple texture mapping without adding any more overhead to the adaptive mesh. If independent texture coordinates were used instead, then these would need to be stored, generated and maintained during refinement and coarsening.

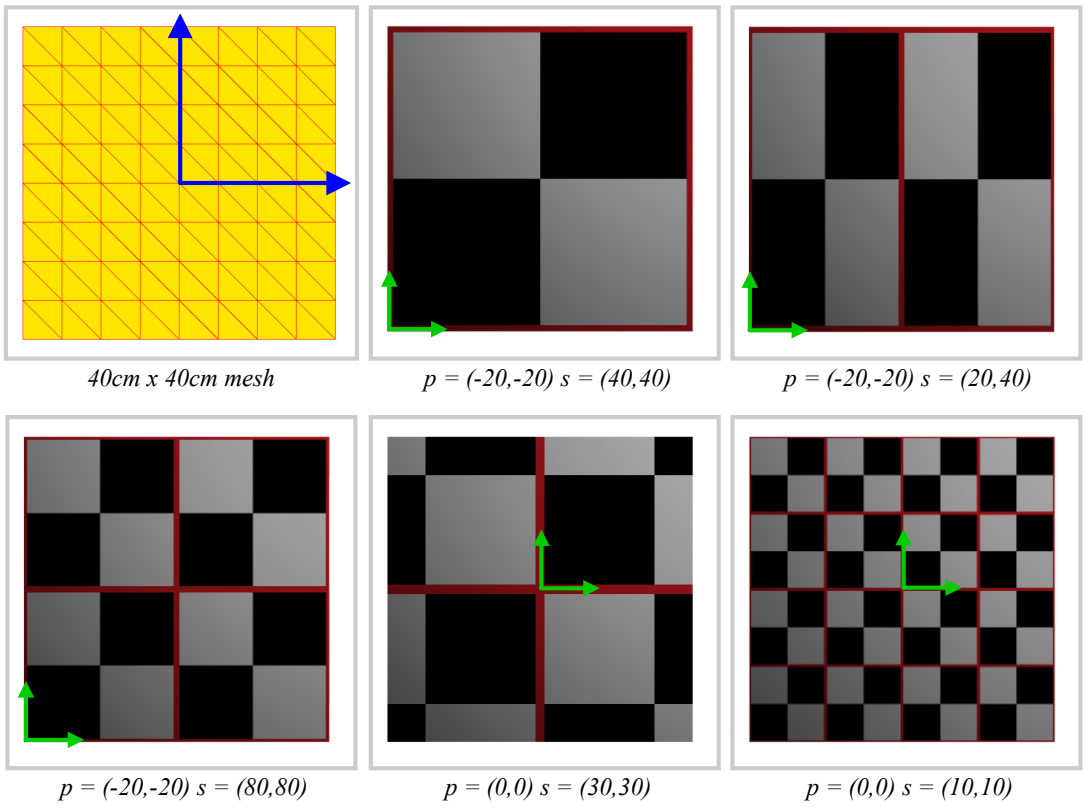

Figure 3.19: Texturing mapping from material coordinates to texture coordinates is demonstrated on a 40cm x 40cm square mesh (Top Left). The rest of the images show a simple test pattern textured onto the mesh. The texture is positioned onto the mesh in material coordinates, the green coordinate axis indicate the position (p), the size (s) of the texture effects its scale.

We define a simple mapping function to convert from a 2D material coordinate to a texture coordinate where the texture is defined in coordinates between 0 and

1. We accomplish this by placing the texture onto the cloth mesh in 2D, giving it a position, $p = (px, py)$ and size, $s = (sx, sy)$ defined in material coordinates. Then a texture coordinate, $t = (tx, ty)$ maps to a material coordinate $m = (mx, my)$ using the following formula:

$$mx = px + (tx * sx) \tag{3.7.1}$$

$$my = py + (ty * sy) \tag{3.7.2}$$

Trivially rearranging, the mapping from material coordinates to texture coordinates is therefore given by:

$$tx = \frac{mx - px}{sx} \tag{3.7.3}$$

$$ty = \frac{my - py}{sy} \tag{3.7.4}$$

We have tested this on a square mesh whose size in materials coordinates is 40 cm by 40 cm, centred at (0,0) extending from (-20,-20) to (20,20). The texture can be made to fill the whole mesh by simply setting p = (-20,-20) and s = (40,40), if instead s = (20,40) then the texture will repeat twice horizontally (see Figure 3.19 for these and other examples using a test pattern). The texture's colour is modulated with the diffuse component of the Phong lighting model, see Figure 3.20 for examples of rendering with tartan textures.

Figure 3.20: A hanging cloth mesh is rendered with three different tartan style textures.

3.8 Results

In this section we will now present the results and overall performance of the edge-based adaptive mesh as a whole and its use for cloth simulation as a mass-spring network. Firstly we will examine the performance of our adaptive mesh in a very controlled environment with uniform refinements. Secondly we will test it in a more dynamic and unpredictable setting, by performing cloth simulations with simple collisions.

3.8.1 Uniform Refinement

We have created a number of square meshes to test the performance of the adaptive mesh as a whole. They are created using quad meshes of sizes 4×4, 8×8, 16×16 and 32×32 where each quad is divided into two triangles such that the 4×4 mesh has $4 \times 4 \times 2 = 32$ triangles etc. The reason for choosing this range of sizes is that when they are uniformly refined they have the same triangle counts but on different respective levels. For instance, the 4×4 mesh will have 128 triangles on level 1 and the 8×8 mesh has 128 triangles on level 0. We can compare the performance for the same triangle counts but on different levels, therefore the overheads of the hierarchy can be seen where we expect that lower levels will be more efficient than higher levels with the same triangle counts. We perform uniform refinements of all the meshes to a maximum triangle count of 131,072, which includes the 4×4 mesh refined to level 6, 8×8 to level 5, 16×16 to level 4 and the 32×32 refined to level 3.

We have timed uniform refinement and coarsening of the meshes using both standard memory allocations and allocations using our memory pools, and we have repeated and averaged these timings over 100 runs. The timings together with the mesh sizes can be found in Tables 3.7, 3.8, 3.9 and 3.10. The most significant observation is that the memory pools are very effective; reducing the refinement times on average

to 47.8%. The coarsening times are also greatly improved; the memory pools reduce those costs to 43.4% on average. The recursions depth shows expected effects on performance in many cases, for instance refinement from 512 to 2048 triangles (using the memory pool) cost 0.270 ms for 4×4 mesh, 0.269 ms for 8×8 and 0.253 ms for 16×16. So the recursion depth does not affect the performance by any significant amount. However, with standard memory allocations there is much more variability from effects out of our control, so again recursion depth isn't a large performance concern.

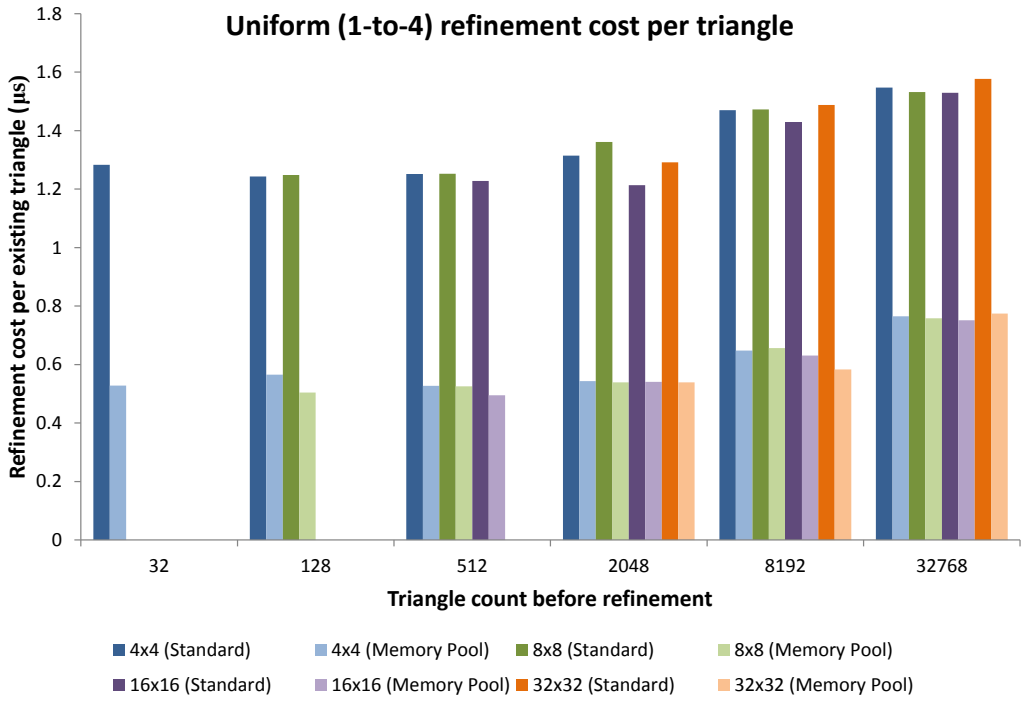

Figure 3.21: A comparison of the cost of refinement per existing triangle in the mesh using standard and memory pool allocations. Refinement increases the number of triangles by four times in the mesh and increases its level by one.

We can gain more insight into the relative performance of adaption by expressing the refinement and coarsening costs on a per triangle basis. For simplicity we divide the cost among the existing triangles, e.g. if a 32 triangle mesh is refined to 128

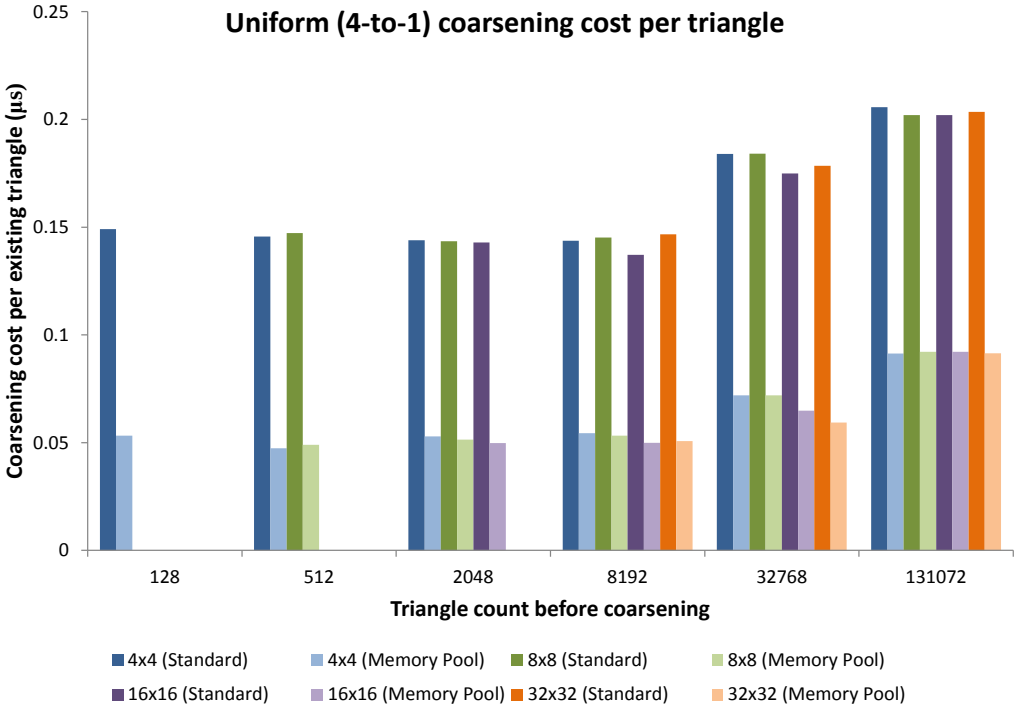

Figure 3.22: A comparison of the cost of coarsening per existing triangle in the mesh using standard and memory pool allocations. Coarsening reverses previous refinement of a mesh, decreasing the triangle count to a quarter and its level by one.

triangles then we divide the cost by 32, and, if a 128 triangle mesh is coarsened to 32 triangles, we divide the cost by 128. These costs are summarised on the graphs shown in Figures 3.21 and 3.22. The refinement cost per triangle is almost constant up to 2048 triangles (refining to 8192 triangles) at around 1.25 μs (microseconds) and 0.5 μs for standard and memory pool allocations respectively. After this the performance starts to drop off as the cost begins to increase with successive refinement, increasing to over 1.5 μs and 0.75 μs for standard and memory pool allocations respectively by the refinement of 32768 triangles (to 131072 triangles). This is most likely due to the CPU's cache size, we show the basic memory requirements for the meshes in Table 3.11. This is worked out from the size and number of vertices (144 bytes), edges (52 bytes) and triangles (120 bytes) that are in the hierarchy. The processor has a

level 3 cache (shared between all cores) of 8 MB and each physical core has a 256 KB level 2 cache. Therefore we can infer that the performance drop-off is caused by increased cache misses when the memory requirement of a single mesh exceed the CPU's level 3 cache of 8 MB by itself. Refinement from 8192 to 32768 increases memory consumption from around 2.65 MB to 10.5 MB, refining again to 131072 takes the memory costs up to around 42 MB. The memory usage for the different meshes refined to 131072 triangles is only marginally different, for example, the 4×4 mesh takes 42.12 MB and the 32×32 takes 41.99 MB. The number of vertices is the same for the matching triangle count (in the highest active level) and does not depend on the level. However, the edges and triangles in previous levels add to the total memory, the 4×4 mesh has an additional 1064 edges and 672 triangles in levels 0 to 2 and levels 3 to 6 correspond to the 32×32 mesh levels 0 to 3.

Table 3.7: Uniform refinement and coarsening performance for a square base mesh of 32 triangles (4×4 quad grid divided into two), the times show the refinement and coarsening performance (in milliseconds) with standard memory allocations (C++ new and delete) compared with overloaded operators which delegate the allocations to the memory pools.

				Standard Allocation		Memory Pool	
Level	Vertices	Edges	Triangles	Refine	Coarsen	Refine	Coarsen
0	25	56	32	0.041	-	0.017	-
1	81	208	128	0.159	0.019	0.072	0.007
2	289	800	512	0.641	0.075	0.270	0.024
3	1089	3136	2048	2.692	0.295	1.114	0.108
4	4225	12416	8192	12.041	1.177	5.305	0.445
5	16641	49408	32768	50.712	6.029	25.061	2.357
6	66049	197120	131072	-	26.952	-	11.970

Table 3.8: Uniform refinement and coarsening performance for a square base mesh of 128 triangles (8×8 quad grid divided into two), the times show the refinement and coarsening performance (in milliseconds) with standard memory allocations (C++ new and delete) compared with overloaded operators which delegate the allocations to the memory pools.

				Standard Allocation		Memory Pool	
Level	Vertices	Edges	Triangles	Refine	Coarsen	Refine	Coarsen
0	81	208	128	0.160	-	0.065	-
1	289	800	512	0.641	0.075	0.269	0.025
2	1089	3136	2048	2.787	0.294	1.105	0.105
3	4225	12416	8192	12.061	1.189	5.379	0.436
4	16641	49408	32768	50.203	6.033	24.856	2.359
5	66049	197120	131072	-	26.476	-	12.081

Table 3.9: Uniform refinement and coarsening performance for a square base mesh of 512 triangles (16×16 quad grid divided into two), the times show the refinement and coarsening performance (in milliseconds) with standard memory allocations (C++ new and delete) compared with overloaded operators which delegate the allocations to the memory pools.

				Standard Allocation		Memory Pool	
Level	Vertices	Edges	Triangles	Refine	Coarsen	Refine	Coarsen
0	289	800	512	0.629	-	0.253	-
1	1089	3136	2048	2.485	0.293	1.109	0.102
2	4225	12416	8192	11.697	1.124	5.171	0.409
3	16641	49408	32768	50.117	5.730	24.619	2.123
4	66049	197120	131072	-	26.481	-	12.072

3.8.2 Adaptive Refinement and Cloth Simulation

The results of uniform refinement tests in the previous section give a very good insight into its performance for refinement and coarsening. We are very interested in its performance with dynamic refinements and cloth simulation, where the costs of adaption criteria are also included. We drop cloth on to simple geometric objects (spheres and cylinders), the cloth drapes over and eventually slides off of the objects thereby generating constantly changing dynamic refinements. Collisions are performed analytically

Table 3.10: Uniform refinement and coarsening performance for a square base mesh of 2048 triangles (32×32 quad grid divided into two), the times show the refinement and coarsening performance (in milliseconds) with standard memory allocations (C++ new and delete) compared with overloaded operators which delegate the allocations to the memory pools.

				Standard Allocation		Memory Pool	
Level	Vertices	Edges	Triangles	Refine	Coarsen	Refine	Coarsen
0	1089	3136	2048	2.646	-	1.105	-
1	4225	12416	8192	12.186	1.201	4.777	0.415
2	16641	49408	32768	51.666	5.848	26.670	1.942
3	66049	197120	131072	-	26.670	-	11.995

with the objects; we project collided vertices out of the objects on to their surfaces. We render the objects at 95% of their collision size, such that very small intersections are removed. Even with this, the coarse meshes still experience intersections with the rendered objects so refinement is needed to remove these.

We use a base mesh of 32 triangles with a maximum refinement depth of level 4 (8192 triangles maximum). We have first performed a cloth simulation on a sphere and cylinder using uniform refinements of the mesh, the results of this are summarised in Table 3.12 and Table 3.13. The simulation and rendering costs are very similar to one another; this is expected because all costs are proportional to mesh density. The only variable cost is the edge length constraints, because the corrections are only applied to over stretched edges (over 105% of length). The collision costs are very fast in comparison with the totat costs with such simple objects since they permit very fast intersection tests for each vertex; still the sphere is much cheaper than the cylinder (0.0673 ms or 1.85% of the total compared to 0.2832 ms or 7.48% of the total for level 4 refined meshes).

Next we repeated the simulation with dynamic refinements using different combinations of criteria, A) collision, B) curvature and C) edge length (extension). We follow the decoupled simulation approach, and we perform one adaption update (30Hz)

to each four simulations steps (120Hz). We present our results showing average costs over four steps for simplicity; this means that the cost of the adaptive mesh update is shown at 25% of its actual cost each step instead of 100% every 4 steps. The results from all combinations of criteria are summarised in Table 3.14, showing the average costs per step. The most interesting result comes from the collision criteria when used alone, which shows our mesh is effective at coarsening. The cloth refines when it is in collision (so there are no visible intersections), and parts which are no longer colliding coarsen as the cloth falls off of the object bringing rapid savings (see Figure 3.23). The detailed step by step cost for the simulation with collision criteria falling onto a cylinder is shown in Figure 3.25, and for a sphere in Figure 3.26. The performance is proportional to the number of triangles, the trend holds with other criteria, for example Figure 3.27 and Figure 3.28 shows the results if curvature criteria are used (in this instance, split when greater than 7°, rejoin when less than 6.7°). Criteria can also be combined, Figure 3.29 shows the results if the collision and curvature criteria are used together. In this situation there is some coarsening as it falls off the sphere, but as the cloth becomes more wrinkled when it is no longer in contact with the smooth surface, so curvature criterion causes more refinement. The length criterion by itself causes no refinement on the cylinder (split when less than 98% and rejoin when greater than 99%) as the edges are stretched when draping over the object. However, when dropped onto a sphere, the corners of the cloth drape down around the sphere causes compression in the middle sections which in turn causes refinement, see Figure 3.30. Figure 3.24 shows a selection of screen captures from the simulations.

3.9 Summary

In this chapter we have presented an edge-based incremental approach to adaptively refine and coarsen a mesh, this has involved detailed explanations of its implementation focusing on efficiency and performance. We have also covered many important details of the mesh representation and the refinement procedure. The refinement and coarsening of the mesh is triggered by two main operations on edges, namely splitting and rejoining of edges. Any manner of edge-based criteria can be implemented and combined easily through functions returning Boolean values. We have demonstrated the use of three kinds of criteria based on curvature, edge length and collisions. We have seen that our state based triangulation approach allows very fast incremental retriangulation with a maximum transition cost of 0.529 microseconds and an average of 0.341 microseconds (Table 3.3). Especially with our reconfiguration approach that exploits similarities between configurations, the average transitions costs were reduced to 59.5% (Tables 3.4 and 3.5). We have seen that the standard methods of allocating memory dynamically using C++ impose large overheads, and we have alleviated them by employing three memory pools, one each for vertices, edges and triangles in the adaptive mesh.

We have demonstrated the adaptive mesh's suitability for real-time mass-spring network for cloth simulation using a simple piece of cloth colliding with geometric objects. The need for small time steps for stability can be problematic due to the many updates required each second, a rate much higher than required for the rendering of smooth animations. In these situations the cloth is moving very little between steps, so we have decoupled the simulation, so that the adaptive mesh may be updated independently when required. In our tests, the adaptive mesh updates cost an average of 7.51% of the total update costs, when we update it at 30Hz while running the simulation at 120Hz.

The coarsening ability of our adaptive mesh has been shown very effective with collision criteria; but the other criteria did not allow such coarsening since they do not coarsen wrinkled regions. However, the use of the collision criterion alone does not produce visually plausible cloth animations, even after a collision the resulting wrinkles should not be coarsened away. So the result is not unexpected, the selection of criteria is important and depends on the situation. It can be difficult to select good criteria thresholds for relatively unconstrained cloth, the cloth can stay too coarse or become totally refined. We anticipate the coarsening ability of our adaptive mesh will be utilised better with clothing, since the characters body shape will constrain the cloths shape and motion, such that the adaptive mesh can respond to the current pose of the character. Hence, the next stage is to move from single pieces of cloth to full garments and clothing for 3D virtual characters. This involves the creation of garments, character animation and more complicated collision detection.

Table 3.11: The memory requirements for the total number of vertices, edges and triangles in the adaptive hierarchy for uniform refinements of the square meshes. Vertices are 144 Bytes (including 80 Bytes for the adjacent triangle list that can accommodate up to 20 triangles), edges are 52 Bytes and triangles are 120 Bytes. The total memory is given in KB and MB.

Mesh	Level	Triangles	Total Vertices	Total Edges	Total Triangles	Total Memory
4×4	0	32	25	56	32	10.11 KB
	1	128	81	264	160	43.55 KB
	2	512	289	1064	672	173.42 KB
	3	2048	1089	4200	2720	685.17 KB
	4	8192	4225	16616	10912	2.65 MB
	5	32768	16641	66024	43680	10.56 MB
	6	131072	66049	263144	174752	42.12 MB
8×8	0	128	81	208	128	37.95 KB
	1	512	289	1008	640	166.82 KB
	2	2048	1089	4144	2688	678.58 KB
	3	8192	4225	16560	10880	2.65 MB
	4	32768	16641	65968	43648	10.55 MB
	5	131072	66049	263088	174720	42.11 MB
16×16	0	512	289	800	512	141.27 KB
	1	2048	1089	3936	2560	653.02 KB
	2	8192	4225	16352	10752	2.62 MB
	3	32768	16641	65760	43520	10.53 MB
	4	131072	66049	262880	174592	42.09 MB
32×32	0	2048	1089	3136	2048	552.39 KB
	1	8192	4225	15552	10240	2.52 MB
	2	32768	16641	64960	43008	10.42 MB
	3	131072	66049	262080	174080	41.99 MB

Table 3.12: Average computational cost in milliseconds for each step of the cloth simulation for a 32 triangle base mesh dropped onto a cylinder with uniform refinements.

Level	Triangles	Simulation	Collision	Render
0	32	0.0148	0.0028	0.0128
1	128	0.0516	0.0067	0.0191
2	512	0.1990	0.0207	0.0304
3	2048	0.7773	0.0746	0.1171
4	8192	3.0893	0.2832	0.4155

Table 3.13: Average computational cost in milliseconds for each step of the cloth simulation for a 32 triangle base mesh dropped onto a sphere with uniform refinements.

Level	Triangles	Simulation	Collision	Render
0	32	0.0180	0.0018	0.0117
1	128	0.0597	0.0031	0.0149
2	512	0.2205	0.0061	0.0302
3	2048	0.8040	0.0186	0.1207
4	8192	3.1552	0.0673	0.4230

Table 3.14: Average computational costs in milliseconds for adaptive simulation of cloth falling onto a cylinder and a sphere with different combinations of criteria: A) Collisions, B) Curvature and C) Edge Length. Corresponding screen captures of these simulations can be found in Figure 3.24.

Shape	A	B	C	Triangles	Adaption	Simulation	Collision	Render
Cylinder	x			1844.3	0.0810	0.9541	0.0901	0.1357
		x		2456.9	0.1039	1.1997	0.0932	0.1796
			x	32.0	0.0017	0.0159	0.0029	0.0130
	x	x		3840.9	0.1819	1.9710	0.1616	0.2709
	x		x	3483.0	0.1307	1.5719	0.1487	0.2162
		x	x	3909.6	0.1740	1.8972	0.1519	0.2735
	x	x	x	5535.7	0.2480	2.8129	0.2236	0.3769
Sphere	x			1307.1	0.0500	0.6626	0.0260	0.0985
		x		5529.4	0.2191	2.7015	0.0644	0.3629
			x	1627.5	0.0803	0.8089	0.0175	0.1223
	x	x		5800.1	0.2398	2.9184	0.0732	0.3878
	x		x	3313.6	0.1553	1.6504	0.0547	0.2355
		x	x	6012.8	0.2547	2.9422	0.0807	0.4093
	x	x	x	6378.2	0.2217	2.9484	0.0701	0.3808

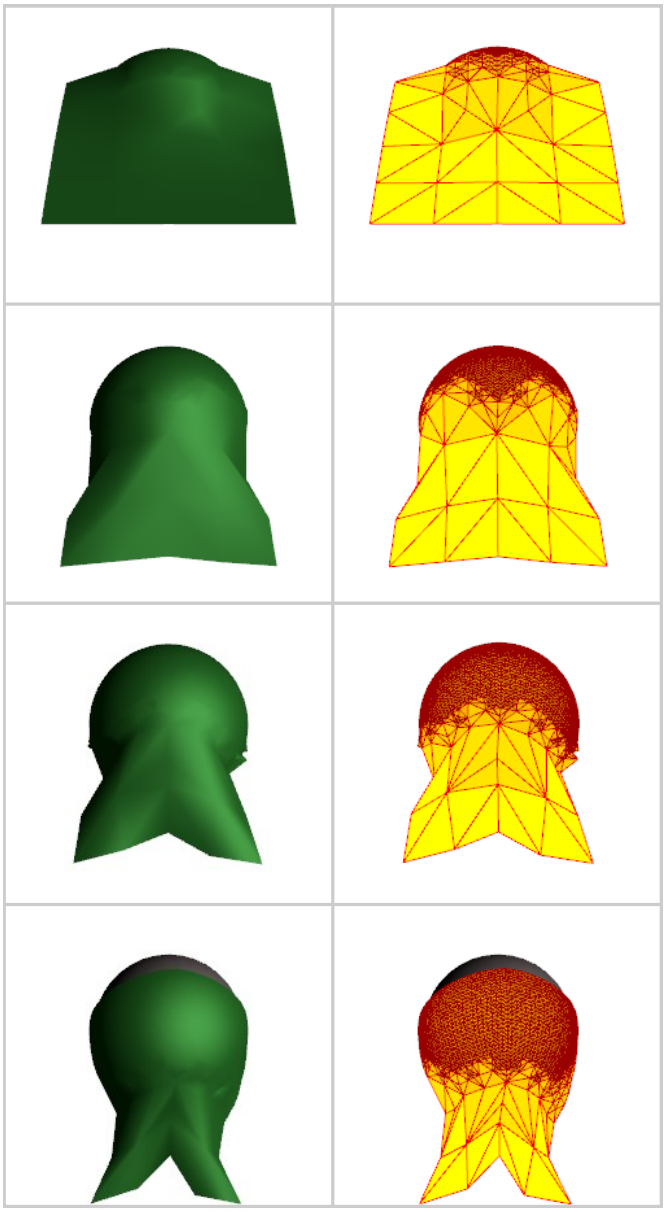

Figure 3.23: Sphere [A]: Screen captures from the sphere simulation with collision criteria, immediate coarsening is seen in parts of the cloth no longer in contact with the sphere.

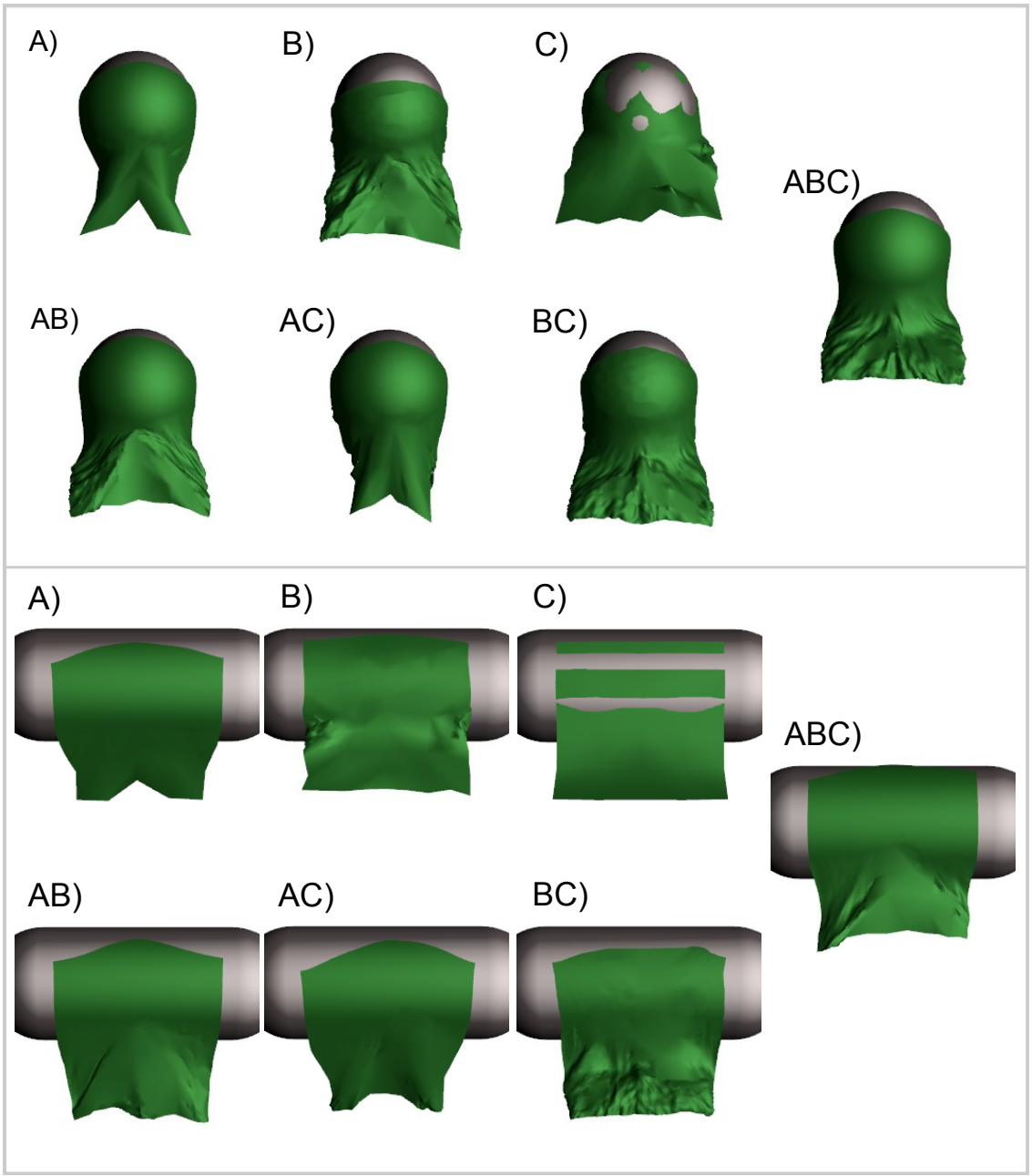

Figure 3.24: Screen captures from the simulations corresponding to those in Table 3.14 with the sphere (top) and Cylinders (bottom), they are shown for different combinations of criteria: A) Collision, B) Curvature and C) Length. Notice how C by itself is ineffective, but combined with other criteria it produces increased wrinkling.

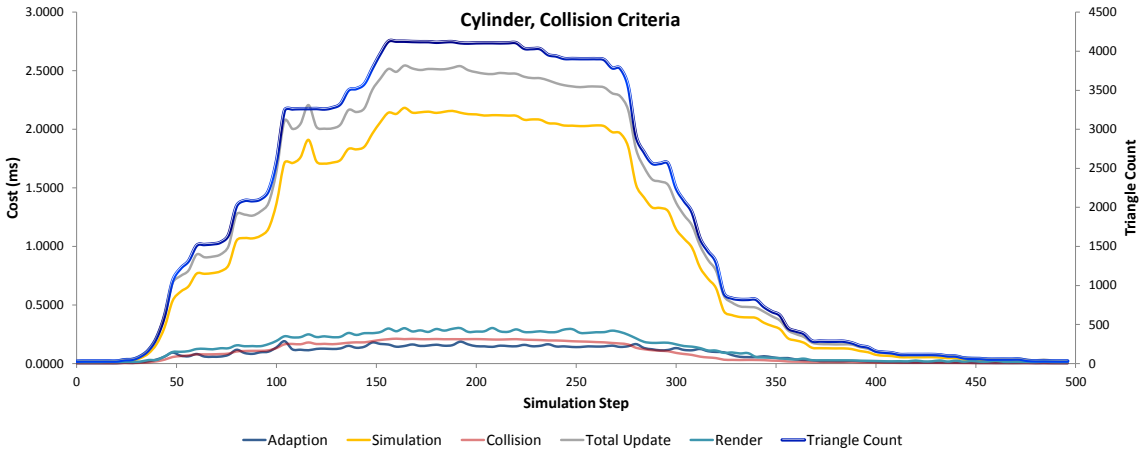

Figure 3.25: Cylinder [A]: Step by step timings for the cloth being dropped onto and falling off of a cylinder with collision criteria. All times are given in milliseconds (left axis), the total update includes the adaption, simulation and collision costs, the render cost is also given. The adaptive triangle count is shown (right axis).

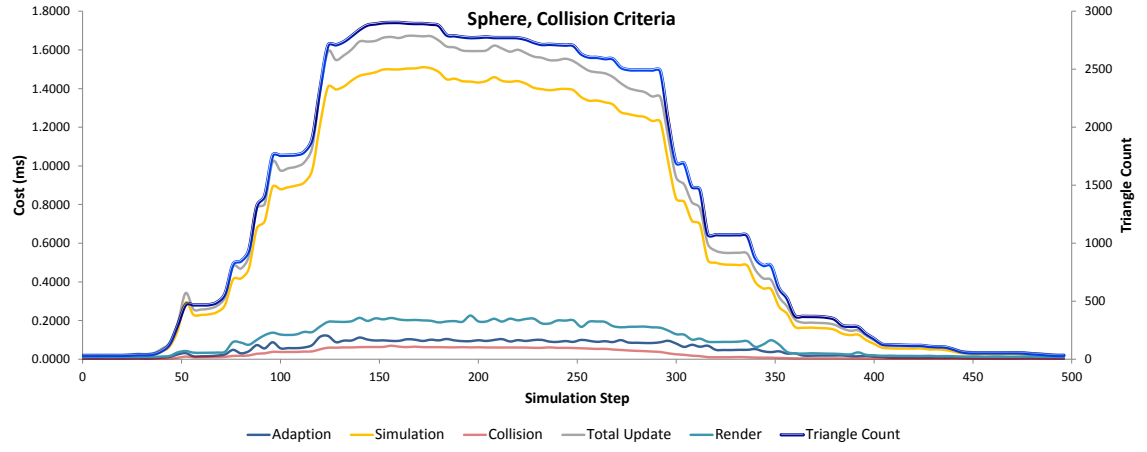

Figure 3.26: Sphere [A]: Step by step timings for the cloth being dropped onto and falling off of a cylinder with collision criteria. All times are given in milliseconds (left axis), the total update includes the adaption, simulation and collision costs, the render cost is also given. The adaptive triangle count is shown (right axis).

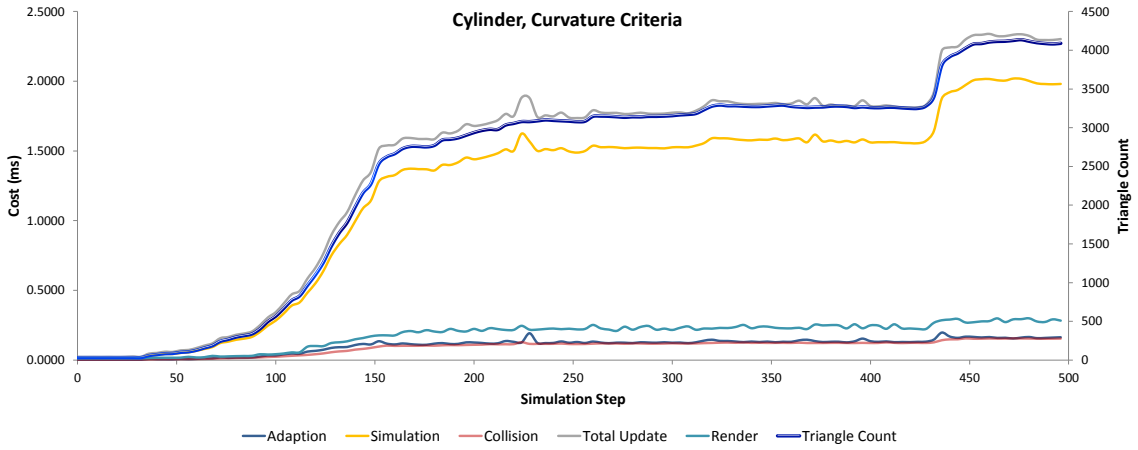

Figure 3.27: Cylinder [B]: Step by step timings for the cloth being dropped onto and falling off of a cylinder with curvature criteria. All times are given in milliseconds (left axis), the total update includes the adaption, simulation and collision costs, the render cost is also given. The adaptive triangle count is shown (right axis).

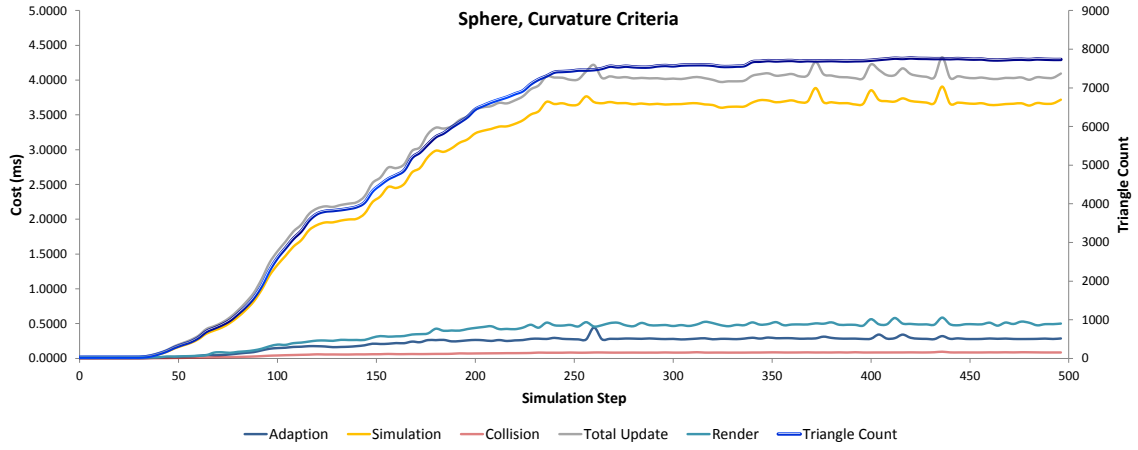

Figure 3.28: Sphere [B]: Step by step timings for the cloth being dropped onto and falling off of a cylinder with curvature criteria. All times are given in milliseconds (left axis), the total update includes the adaption, simulation and collision costs, the render cost is also given. The adaptive triangle count is shown (right axis).

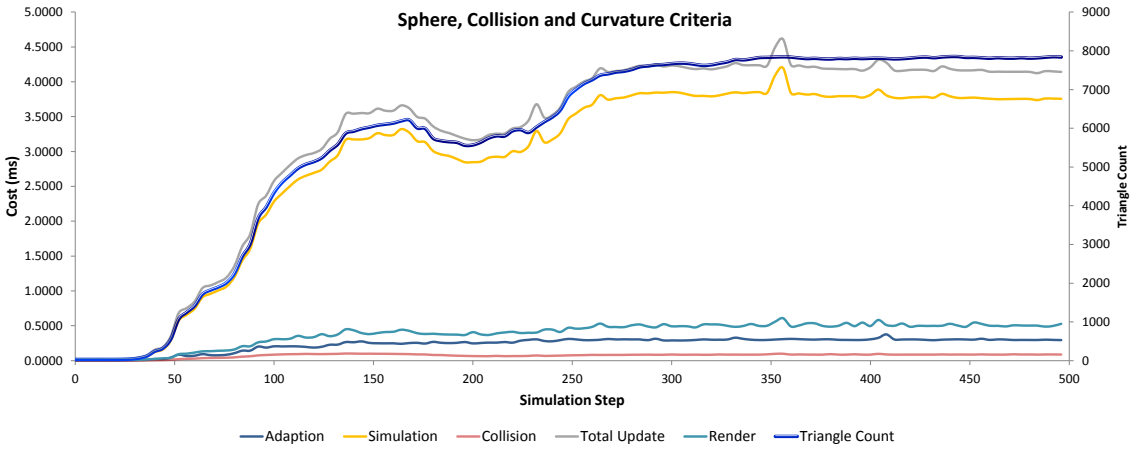

Figure 3.29: Sphere [AB]: Step by step timings for the cloth being dropped onto and falling off of a cylinder with collision and curvature criteria. All times are given in milliseconds (left axis), the total update includes the adaption, simulation and collision costs, the render cost is also given. The adaptive triangle count is shown (right axis).

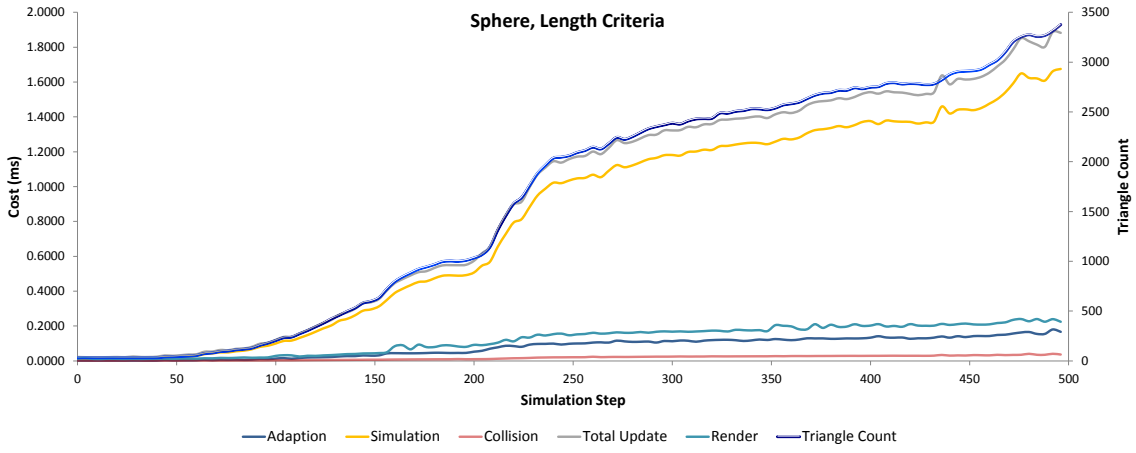

Figure 3.30: Sphere [C]: Step by step timings for the cloth being dropped onto and falling off of a cylinder with length criteria. All times are given in milliseconds (left axis), the total update includes the adaption, simulation and collision costs, the render cost is also given. The adaptive triangle count is shown (right axis).

Chapter 4

Real-Time Clothing using Edge-based Adaptive Meshes

4.1 Introduction

The simulation of clothing is much more involved than the simulation of simple small pieces of cloth; techniques from many areas must be brought together. Firstly garment patterns or meshes must be designed, created and dressed on the character before simulation can begin. Virtual characters must be created, rigged and animated together with efficient collision processing techniques to allow the characters to wear the garments. In this Chapter we focus on the initial stages of this problem, leaving out the complexity of animated characters for now. We refer the reader to Chapter 2.4 of our literature review for background information. This chapter begins with a description of the garment creation process that we follow to create clothing using our edge-base adaptive mesh. We employ efficient pre-computed collision structures to enable the real-time draping of clothing on a static character.

4.2 Garment Creation

In this section we will explain how we handle the creation of garments in our work; the approach we took was to create virtual garments in a similar way to real ones,

by seaming multiple cloth pieces together. A garment is specified by a cloth pattern, which is made from a group of 2D meshes including data that describes how the meshes are placed and joined together. Each 2D mesh is defined in its own coordinate space which we refer to as material coordinates. Meshes are combined following the cloth pattern to create an intermediate 3D cloth mesh which stores 3D positions and the 2D material coordinates. The 3D cloth mesh can be passed to constructors to create actual cloth simulation meshes such as a uniform mass-spring system or our edge-based adaptive mesh, see Figure 4.1.

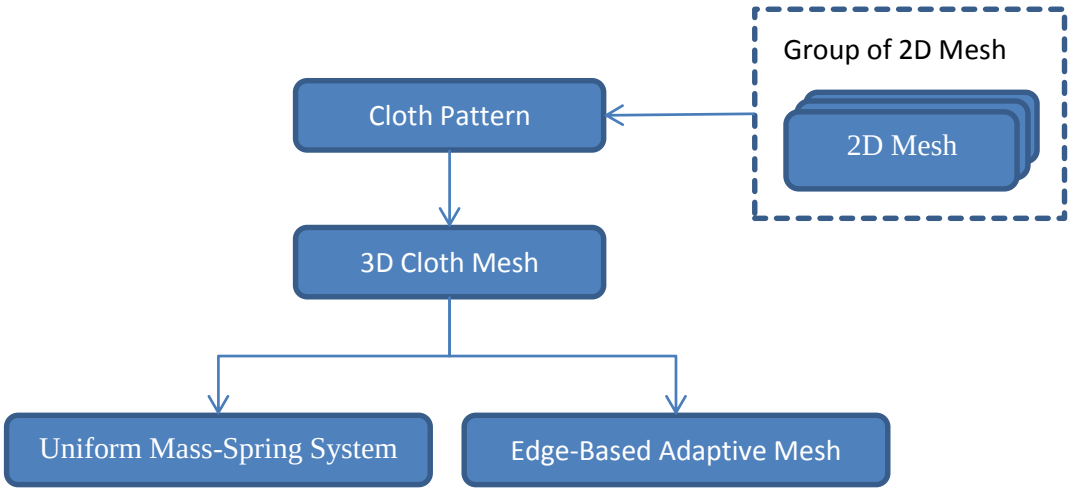

Figure 4.1: Data flow for the construction of cloth meshes, groups of 2D meshes are combined following a cloth pattern. The meshes are seamed together into an intermediate 3D mesh that combines 3D positions and 2D material coordinates, which is used as input to meshes for simulation.

4.2.1 Computer Aided Design of Cloth Patterns

In order to facilitate the creation of the 2D cloth meshes and most importantly complete patterns for garments, we implemented a simple user friendly design editing program with a graphical user interface. We have many operations for using the editor; we began with a small set of them required for basic functionality such as for the

creation, selection, moving, scaling and deletion of vertices and triangles. The advantage of custom software is the ability to add features as the need arises, and we have exploited this to enhance productivity with additional operations such as copy and paste or mirroring. Pictures may be imported and displayed as a background image, this can be helpful if you have a pattern to wish to trace (for example a scanned image or photo of a pattern). Commercial CAD packages support more features than our editor; ours was not designed to compete with them. For instance, we do not support curved outlines of patterns and their conversion to triangular meshes, however, we allow meshes to be imported and exported from our editor. The main focus is to create garments by seaming patterns together, and we have a 3D user interaction process by which this is achieved. The ability to easily edit the meshes was useful; it is very hard to envisage the final 3D shape of the garment on a body from its pattern, so often changes are needed to improve the fit of the garment.

2D Mesh Representation and File Format

We use Wavefront's Object (.obj) file format for the 2D meshes, it is a geometry definition file format that stores 3D geometry with normal and texture data in a simple human-readable text file. Although its specification includes other features such as curves and curved surfaces, these features are not commonly supported as people tend to implement their own simple functions to load and save them. We store the cloth vertices' 2D positions using XY coordinates, with a zero value for the Z coordinate, triangles are stored as three indices to their vertices. Triangles are defined in an anti-clockwise order, the surface normals are defined to be positive in the positive Z-direction using a right-handed coordinate system. Our program automatically reorders the triangle's vertices to ensure correct facing normals. We store additional data such as which meshes make up a pattern and their relative placements in simple text files.

Mesh Editing

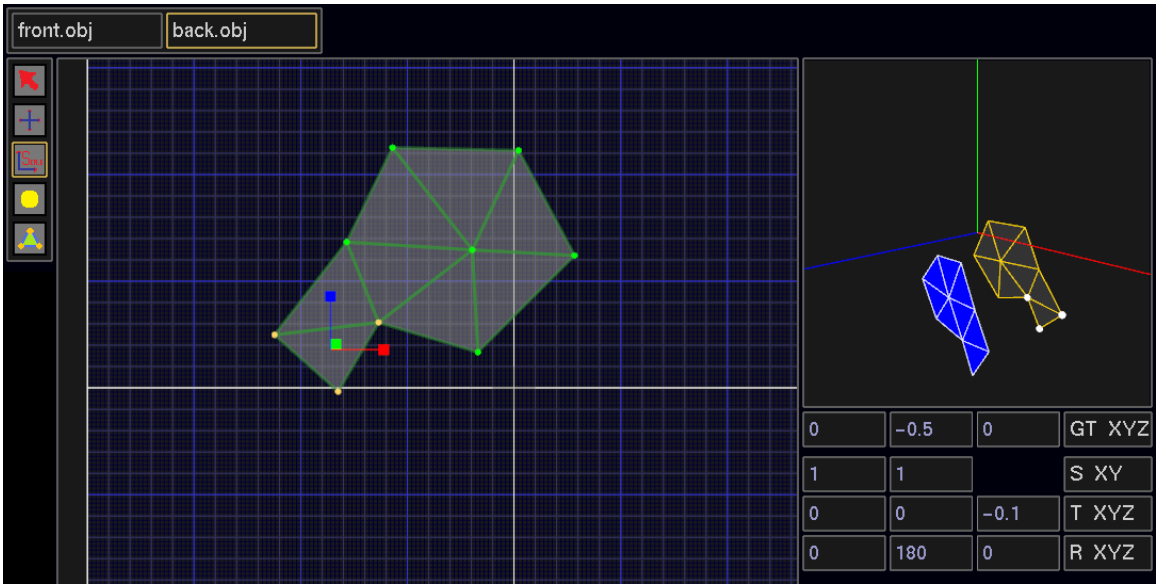

Figure 4.2: A screen capture from our pattern editor is shown. It features two meshes; the second mesh (back.obj) is currently being edited and is displayed on the material coordinate grid. Three vertices of a triangle has been selected with the scale tool active, the vertices are highlighted in both the 2D grid and 3D views (top right). The transformations are specified in the bottom right text boxes: global translation (GT XYZ), 2D mesh scale (S XY), 3D mesh translation and 3D mesh rotation (R XYZ).

Garment patterns are created from one or more 2D meshes that are loaded into the editor at once, meshes are selected between using tabs, see Figure 4.2. Meshes lay in the XY plane; they can be scaled (S) in 2D and then positioned in 3D using rotations about each axis (R) and translations (T) prior to seaming. Garments also feature a 3D global translation (GT) for convenience; this can be used for example to position a whole garment at the correct height on a character. A small viewing area shows the meshes in their transformed 3D positions in real-time during editing.

In order to define garment seams, links are added in a special mode that expands the small viewing area to cover the full screen. The links are specified by the user with the mouse by selecting groups of vertices that should be joined together. It is

permitted for a single mesh to be sewn between itself; this is useful to model cuts and darts (used in tailoring to improve the fit of the garment). It is not necessary for the user to specify which edges should be sewn together; they can be automatically worked out if two edges vertices are linked.

Seaming

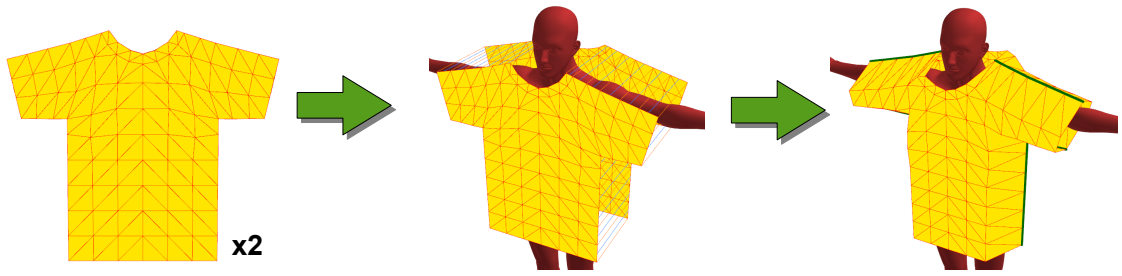

Figure 4.3: Left: initial 2D meshes, Middle: meshes are transformed and positioned in 3D with seaming links according to a cloth pattern, Right: after seaming, the original seams are highlighted in green.

The cloth pattern is followed to combine multiple 2D cloth meshes into a 3D cloth mesh, this involves joining the seams together along the meshes. Although the seams influence the behaviour of real garment, we do not believe that complicated models for seaming are currently feasible for real-time simulations. For instance, in Pabst *et al.*'s [PKST08] work, looking at the influence of seams on bending we found that coarse meshes had to be refined around seams. The cost of their bending force computation was 65 ms alone without considering the rest of the simulation or collision detection. Also Ma *et al.*'s [MHB06] method to construct detailed seams onto irregular meshes was also only suitable for offline work. Therefore we follow a simple approach of merging the 2D meshes along the seam lines into a single 3D cloth mesh. To combine the meshes, we insert the triangles into the 3D mesh one by one and perform vertex merging (sometime referred to as welding) by proximity. This relies on the cloth pattern returning identical positions for vertices that should be merged at the seams,

found from the user defined vertex links. The position returned for merged vertices is the average of all their positions, and we note that this causes stretching in edges connected to the merged vertices but the 3D cloth mesh contains the 2D material coordinates so the un-deformed size of the cloth is not lost. The 3D cloth mesh also constructs explicit edges including some simple connectivity data that can make the later construction of simulation meshes easier. Seaming for a T-Shirt is illustrated in Figure 4.3, using a mesh for the front and one for the back with seams above the shoulders, under the arms and down the sides.

4.2.2 Dressing

As previously mentioned, seaming causes stretching in the mesh from merging vertices but these does not cause major problems. Often dressing is performed at the same step as sewing [VT97, Vas00, PLAMT02, DG07], the panels are placed around the garment and are simulated with constraints or springs to bring them close enough to be joined. Although automatic positioning has been proposed [GFL03, FGLW03, LTG05] for dressing, we have not found any need to perform a complicated dressing procedure. We are able to dress the character successfully so the clothes are worn as would be expected in reality by employing heavy damping with a small time step to slowly and smoothly relax the cloth around a character's body. One could perform a simulation to bring the 2D cloth meshes together before seaming by enforcing a constraint that merged vertices remain at identical positions; however, the result is no different than performing the simulation after seaming, without the need for the constraints.

4.3 Discontinuous material coordinates

We have already explained that we create pieces of cloth, using a 2D mesh, which are defined in an un-deformed coordinate system, i.e. material coordinates. 2D material

co-ordinates allow the quick determination of un-deformed lengths needed for the physics calculations between any two points of the mesh and they are readily acquired from flat textile patterns. However, there is a challenge with managing material coordinates for the meshes. Ultimately there exists discontinuous co-ordinates along the seam lines between multiple meshes and this is also the case for a single mesh (e.g. a single mesh seamed to form a cylinder).

Commonly in computer graphics information such as normals and texture coordinates is stored per-vertex, and if a two adjacent faces need separate (non-continuous) ones then the vertices are duplicated. Therefore there are multiple vertices in the same place, this is fine for rendering and cracks do not appear as long as they remain in the exact same place. Material coordinates are like texture coordinates, but we cannot store the material coordinates as a vertex attribute as we cannot duplicate vertices along the seams for discontinuities like we can for rendering. The connectivity of the adaptive mesh must be maintained, so we need to handle this in a way that does not complicate refinement and coarsening too much or spoil the efficiency of the adaptive mesh. For this reason, as discussed earlier, we store three materials coordinates per triangle in both the intermediate 3D cloth mesh and the edge-based adaptive mesh. However, we would first like explain the initial way we handled discontinuous material coordinates and how this led to the current way.

We only need to support discontinuous material coordinates between base triangles of the adaptive mesh, along the edges and therefore the coordinates are continuous across a base triangle as it is subdivided. Since an edge between adjacent triangles consists of two sides, each side can have its own material co-ordinates. Rather than duplicating the vertices, we store the material coordinates separately such that each end of an edge holds pointers to both its vertex and material coordinate. Where the material coordinates are continuous, both sides of the edge hold pointers to the

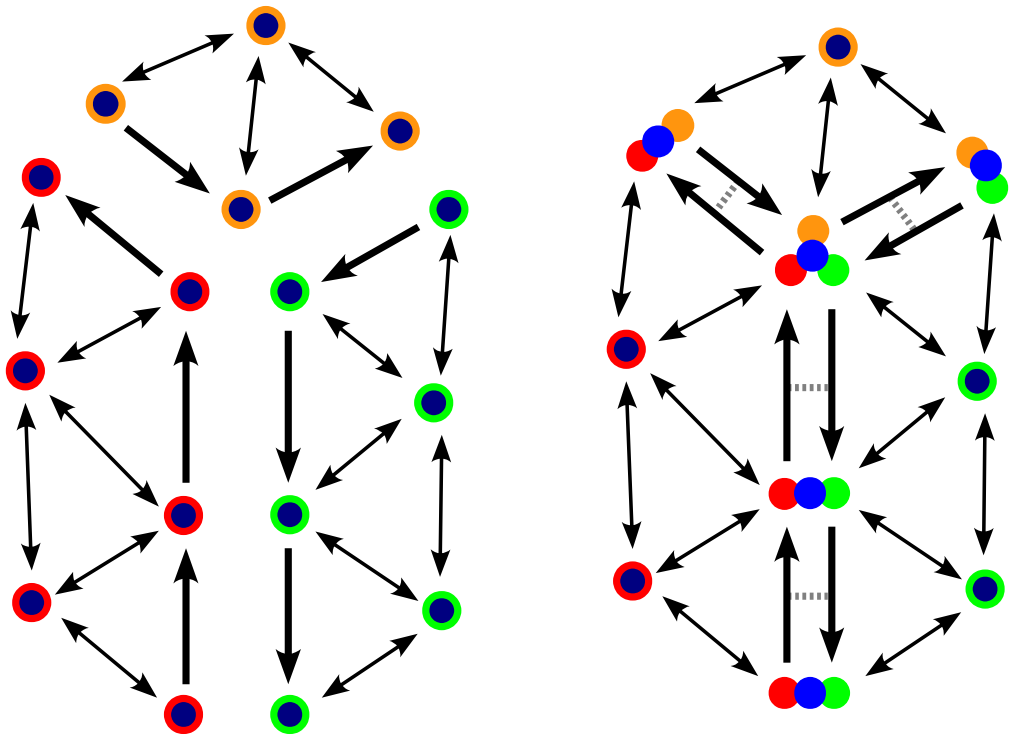

Figure 4.4: Left: Three meshes are shown each with their own set of material coordinates (red, orange, green) that are to be seamed together across their shared boundary edges. Right: Vertices are merged (blue) but material coordinates cannot be merged since they are discontinuous.

same pair of material coordinates. In the case of discontinuous coordinates, each side will hold reference to a different pair of co-ordinates. Figure 4.4 illustrates this; three meshes are to be seamed together with three sets of continuous material co-ordinates where vertices along the seams are merged but the material co-ordinates are not. The advantage of this is discontinuity can be detected by comparing pointers across two edge sides. However, there are disadvantages to this, the main one being that the memory for the material coordinates must be allocated and freed separately in addition to the vertices since it is more efficient to allocate fewer larger objects. Also the memory cost is not optimal; the size of a 32 bit pointer and floating point

number is the same (4 bytes). A material coordinate consists of two floats, so they are 8 bytes which is the size of two pointers. Each edge (two sides) stores 4 pointers which is enough memory to store two actual material coordinates. The total memory cost for the material coordinates in Figure 4.4 is 432 bytes (19x4x4 + 16x8, 19 edges and 16 material coordinates) which is enough to store 54 materials coordinates. In order to remove the separate allocation of material coordinates we can store them directly in the edge or triangles. If we store four material coordinates per edge, two within each side to support discontinuous coordinates then the memory cost will increase to 608 bytes (19x4x8). However, if we store three material coordinates per triangles it only costs 240 bytes (10x3x8) for the 10 triangles in the example. We feel that the loss of pointer comparisons to check for discontinuous coordinates is a good compromise. Since we are now dealing with floating point representation, instead of directly checking for equality, it is safer to check the length between them is less than a tolerance. Furthermore, the squared length can be checked which saves the cost of a square root.

4.3.1 The length of edges at discontinuous seams

Figure 4.5: Two triangles are to be seamed together, but their edge's lengths are not conforming, the seamed edge is given the average length. The only way for the other edges (red) to maintain their length is to distort the shapes of the triangles which will have a knock-on effect to other surrounding triangles (not shown).

The average length can be used for edges on the seam boundary, but for best results we recommend that the edge lengths conform to each other and are equal on both

sides of the boundary. If not, it will cause artefacts through adjoining edges as there will be conflicting rest states. Figure 4.5 highlights this, the adjoining edges will not be able to obtain their rest length without distorting the shape of the cloth triangles; this effect will propagate to neighbouring triangles. An equilibrium point may result where the difference between the deformed length and rest length is minimised but is likely to increase oscillation problem in the simulation as there will be no global rest state.

4.3.2 The calculation of lengths across discontinuous seams

After seaming has been performed, we wish to work out the length across the seam line for each pair of adjacent triangles on the seam for their bending springs. Although the edges will already be aligned in 3D, the triangles will be likely deformed and not lying flat compared with each other. Therefore, we cannot correctly use either the 3D distance or the discontinuous material coordinates.

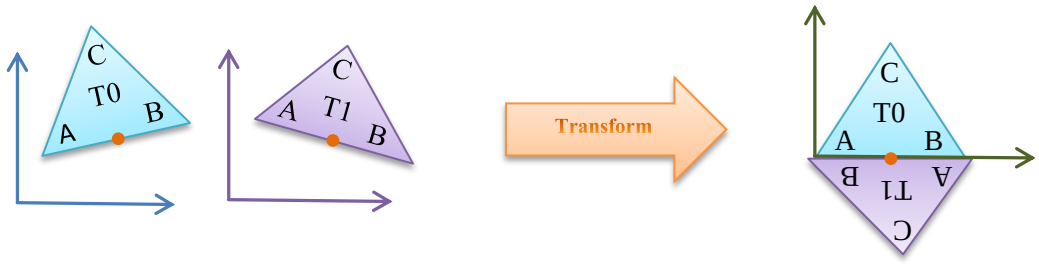

Figure 4.6: Two adjacent triangles (T0 and T1) material coordinates' are transformed to make them continuous across the seam along AB between them. The central point (orange) of the adjacent edges is aligned in case the edge lengths are different.

We transform the materials coordinates so that they are not discontinuous across the seam. We take the seamed triangles in pairs, labelling their vertices in anti-clockwise order A,B and C, such that the first triangle's (T0) vertices A and B matches

with the second's (T1) vertices B and A respectively across the seam edge, see Figure 4.6. The two opposite vertices are labelled C; the distance between those can be used for a bending spring. We arbitrarily choose T0's coordinates frame and for simplicity translate T0 such that A moves to the origin. Next we rotate T0, to make B coincide with the x-axis. Our method takes into account the possibility that boundary edge lengths are not conforming to increase flexibility for the user (even though it is preferable that they be the same length). We accomplish this by using an anchor point half way between A and B on both triangles; these will be aligned and we translate T1 to do so. Finally we must rotate T1 about the anchor point such that A and B are aligned with the x-axis and C is on the negative side of the x-axis. It is then trivial to find the length between T0-C and T1-C. We perform the transformation calculations directly for efficiency; however, a transformation matrix can be created if needed for other lengths or for reuse.

4.4 Collision Detection for Static Objects

Although geometric shapes provide straight forward collision checks by use of their parametric equations, building complex objects out of them is not easy. The decision was made to implement a general method that can be used with any 3D triangle model. The only requirement is that the triangles of the mesh must be outward facing (determined by an anticlockwise vertex winding) forming a surface where the cloth will not be allowed to penetrate through into the object. We have discussed that a problem with the highly flexible nature of cloth is that even if the vertices are not penetrating the object, edges and triangles may intersect the object's surface causing very noticeable visual artefacts. The use of the adaptive mesh greatly increases the cloths ability to approximate the underlying surface it is in contact with, but this alone is not sufficient. A full triangle-triangle collision approach provides a potential

solution although at a cost to processing compared to only vertex-triangle collision detection. We seek a compromise for use with real-time simulations; a suitable approach is to leave a region above the surface to hide these intersections. If too great a region is used, the gap will be very noticeable and unrealistic; relatively too small and intersections will still be visible. We saw this was effective for the simple spheres and cylinders, where we rendered them at 95% of their size. However, it is more difficult with arbitrary triangle meshes since we cannot simple scale their size for rendering. For example, a model of a character has its arms outstretched; if we scale character as a whole the central axis of arms change position and no longer be aligned. Furthermore with the simple spheres and cylinders, it is no different if their radii are scaled down for rendering or scaled up for collision. However, we do not want parts of a character's body to be scaled down for rendering that are not covered by the cloth, e.g. a character's head and facial features. Therefore scaling is not going to provide the answer, because scaling individual body parts will lead to discontinuities between them.

It can be seen we need to actually expand the surface outwards by a distance and test the expanded surface for collisions. To enable this, we use a separate mesh for collision and rendering. The original mesh is loaded, and then uploaded to the GPU and stored in a vertex buffer object (VBO) for rendering. The use of a VBO allows the efficient rendering of static geometry by removing the need to stream vertex data to the GPU every frame. The collision mesh is constructed using the smooth normals which are used for rendering, by using them to define the outwards direction of the surface. We expand the mesh by moving the vertices in the direction of the normals by an adjustable distance. The result is a shell around the original object, giving us the region to hide intersections within. Since the mesh used for rendering is stored in a VBO in GPU memory, the original mesh may be deleted from CPU memory so

the use of the collision mesh does not double the memory requirements.

Although the mesh is static, we allow for it to be arbitrarily rotated and translated in the scene, using a 3x3 rotation matrix and a 3D translation vector. The cloth is simulated in world co-ordinates, and vertices are first transformed into the object's local coordinate space to undergo the collision detection and response. Each cloth particle has a current position and one previous; used with Verlet numerical integration; since the object is static we can consider that the previous position is outside of the object. Hence the collision with a static object can be determined by checking the path from the previous position to the current position has not intersected the object. This process involves a broad phase to cull large areas of the object, and a narrow phase where we perform actual collision detection with the object's triangles.

4.4.1 Grid Construction

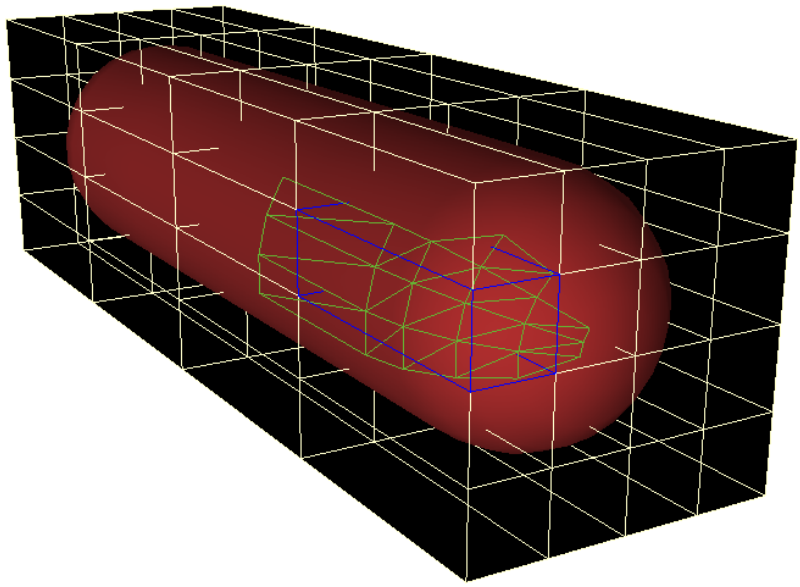

Figure 4.7: A cylinder object, showing the collision grid with triangles from the collision mesh for a highlighted cell.

The object is potentially a large polygon-soup (i.e. an arbitrary mesh of triangles

with no defined structure or relationships between the triangles.), and therefore we initially must partition it into a more efficient structure for collision detection in order to permit real-time computation. We employ a 3D grid of bounding structures or cells, where each cell is an axis aligned bounding box that contains a list of indices to all of the triangles of the object that are either inside or partially overlapping it. The first advantage is that it is possible to find directly the cell that a vertex is inside, so it can outperform tree based spatial subdivision techniques like octrees that have to be recursively searched. We have verified this by performing a simple experiment which builds and inserts a random distribution of points into both an octree and a grid based structure, and we have then timed the cost of searching the structures for the cell (or tree node) which contains each point. The sizes of the structures are modified to encompass different proportions of the total number of points (e.g. some point may be outside of the structure in which no cell or node will be found to contain the point) and the timings are repeated. The results of this is summarised in Table 4.1, it shows that a grid based structure is able to outperform an octree in all cases and takes between 87% to 22% of the time to search compared to the octree (with the octree becoming relatively worse as the structures fill up). Additionally the second advantage is that we can follow a path (or ray) through the grid of cells more easily, checking them in turn. We begin the construction by calculating the axis aligned bounding box of the object and then dividing it up into user defined divisions along each axis. The cells are not required to be cubes, and can be different sizes in each dimension. The memory and performance can be tuned by altering the size of the 3D cells. If large cells are used, more triangles are contained within each grid increasing the cost of the narrow phase. As the size of the cells are reduced, the memory cost increases because the number of cells increases and the total number of triangles indices in the cells increases since more and more triangles are contained

in multiple cells. The memory cost of an empty cell can be very small, just the size of a single pointer which is null for empty cells otherwise it points to the cell's list of indices. The cells are populated with indices of triangles that intersect them in pre-processing stage: We calculate a bounding box for each triangle and only test the triangle against cells which intersect the bounding box. Figure 4.7 shows the grid for a cylinder mesh, highlighting the triangles contained within one of the cells.

Table 4.1: This table shows the total time to search for a number of 3D points within an grid based structure and a octree in milliseconds. The grid structure is divided up into $32 \times 32 \times 32$ cells and the octree has a maximum level imposed such that it can only contain a corresponding maximum of $32 \times 32 \times 32$ leaf nodes (e.g 32768 cells and leaf nodes). We generate a random distribution of 32,768 3D points within a cube volume, then we vary the size of the structures (they are placed in the centre) to take up a certain amount of the total volume (% Vol.) and therefore enclose a variable number of points (Points Contained). Cells and leaf nodes may contain more than one point, in which case there is a list that must be linearly searched.

% Vol.	Points Contained	Grid Search Time (ms)	Octree Search Time (ms)
0%	0	0.431	0.490
10%	3337	0.826	1.698
20%	6582	0.976	2.838
30%	9869	1.183	4.038
40%	13152	1.381	5.303
50%	16440	1.584	6.560
60%	19721	1.828	7.880
70%	22900	2.078	9.145
80%	26227	2.342	10.515
90%	29599	2.637	12.037
100%	32768	2.917	13.395

4.4.2 Collision detection and response

The main collision algorithm between a cloth particle and an object proceeds as follows. The previous and current position are transformed into the object's coordinate space, the transformed points are hence referred to as A and B. We always check for intersections with the cell that A is contained within. If no intersections are found,

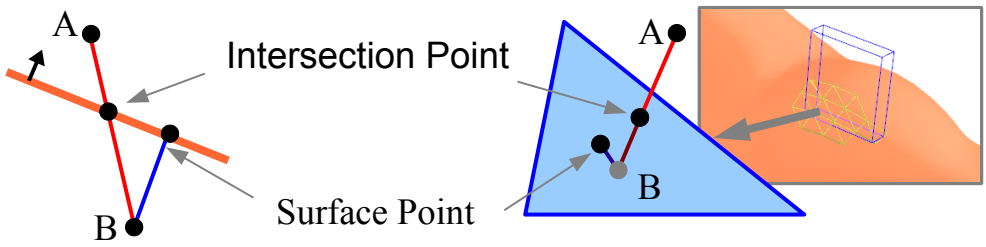

Figure 4.8: Vertex-triangle collision: The vertex's current position (B) is moved to the surface point, if the intersection point is within the triangle. Right: Shows a single grid cell with its overlapping triangles that are highlighted on the left shoulder of a character.

then we step along the AB advancing through all the cells along the way until a collision is found. If no collision is found upon reaching B's cell then the particle has not intersected the surface of the object.

Firstly for a particle to have intersected a triangle, B must be on negative side of the triangles plane (inside the object) and A must be on the positive side of the plane (outside of the object). We do not consider a particle moving from the inside to the outside to be a collision, so if they were inside for any reason they would never be trapped there by the collision detection. Figure 4.8 illustrates the vertex triangle collision test; firstly the line segment AB is intersected with the triangle's plane to find an intersection point (IP). If IP exists, the barycentric coordinates are calculated to determine if this lies within the triangle face; if this is so, the surface point (SP) is calculated by projecting B in the direction of the triangle's normal. The current position is moved to the SP after transforming it back into world coordinates.

4.4.3 Collision aware mesh refinement

We have previously discussed the problem of a vertex being created within a collision object where we described the collision criteria in Chapter 3.4.2. After an edge split, it is necessary to test the newly created vertex for collision because its initial position

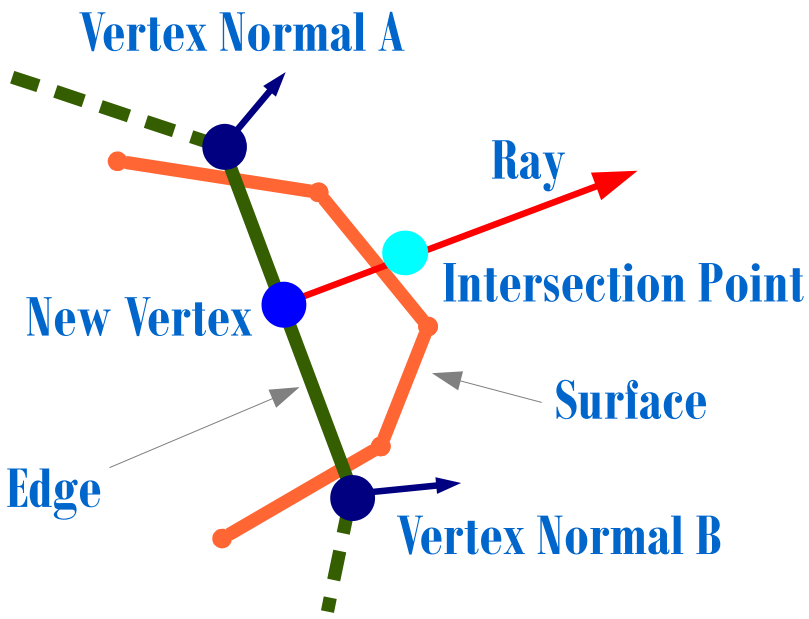

Figure 4.9: Detecting and correction of the position of a new vertex that has been generated inside an object: An ray is tested against the surface for an intersection; if an intersection is found then the vertex is moved out onto the surface.

is assigned the average of the edge's end vertices. We must be able to determine when this has occurred and find a suitable location to relocate the vertex to, but we cannot rely on its previous position being collision free as it was also interpolated. We test for collision by constructing a ray pointing outwards from the object and testing it for collision with the collision mesh. The direction we use is the average of the edges' two end normals. If there is a collision, then the vertex is moved to the intersection point adjusting the previous position to preserve the velocity, see Figure 4.9. This does rely on the cloth being orientated in the same direction as the surface it rests on. This may or not be acceptable depending on the application. In this work, clothes are never worn inside out so local cloth and surface directions always point in the same general direction. If the cloth were to be placed the other way up on a surface, the vertex will be moved onto the wrong part of the surface or if the surface is not closed then no intersection will be found. However, it is straight forward to extend

this approach so that cloth may drape on surfaces anyway up. Instead of testing the ray in one direction, the ray must be traced in opposite direction as well. Then the closest intersection out of the two will be on the correct part of the surface, see Figure 4.10.

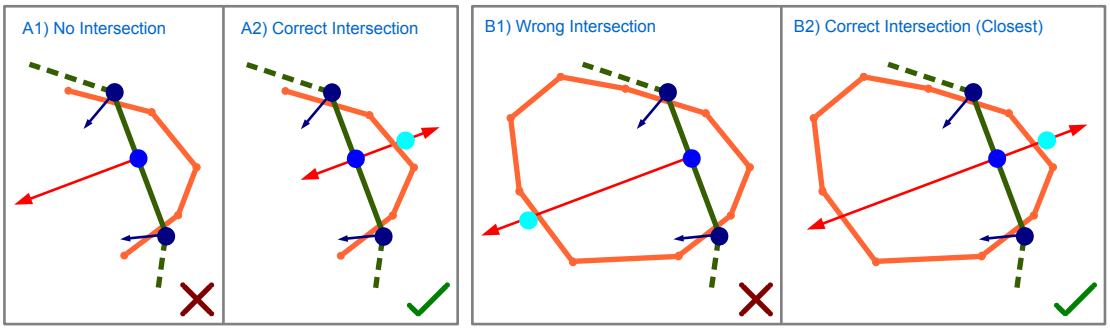

Figure 4.10: This figure shows the cloth's surface direction flipped compared to that of Figure 4.9, in this case the ray does not find the intersection (A1 and B1). However, if the ray is also followed in the opposite direction, correct intersections can be found (A2 and B2). A shows an open surface, whereas B shows a closed surface which encloses a volume such there is an inside and outside.

The importance of this is evident now with the need to dress a character; the coarse base mesh may be used to speed up the initial placement of the garment before enabling refinement with no fear of intersections when refinement is enabled. Figure 4.11 shows a coarse T-shirt dressed on the character before and after refinement is enabled.

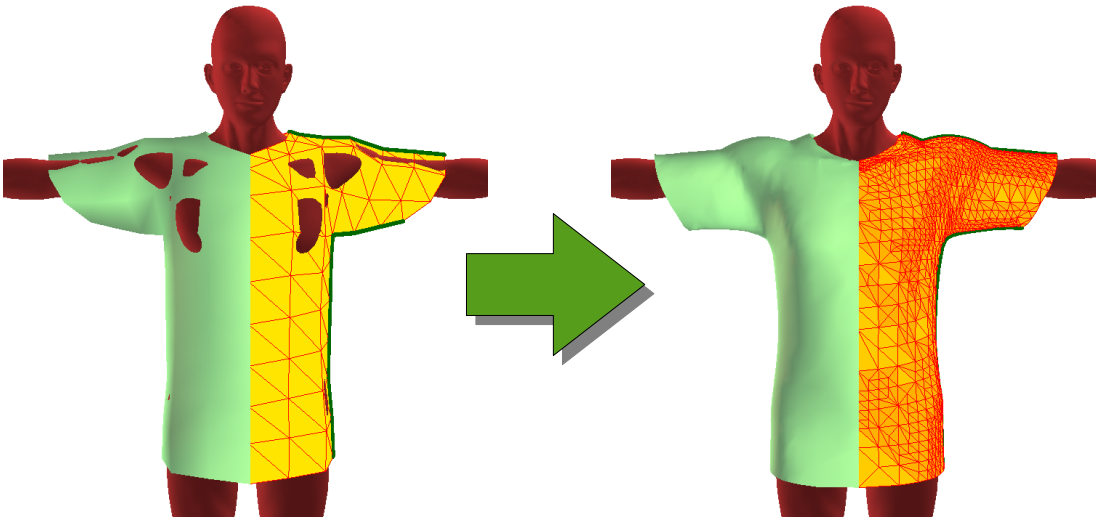

Figure 4.11: Left: A T-shirt is dressed and draped onto the static character; the base mesh is too coarse to prevent intersections with the character's surface. Right: refinement is enabled and the adaptive mesh generates sufficient vertices to resolve all intersections, newly created vertices are correctly moved onto the surface.

4.5 Results

We implemented our cloth simulation using C++ and used Opengl for rendering, the results presented in this paper were performed using a PC with a Intel 2.66 GHz Core i7 920 using a single thread with 6GB of RAM. The static character's mesh contained 67k triangles; we divided the collision grid into 32x32x32 cells with an average of 40 triangles for each non-empty cell. The limit of our approach with regard to achieving real-time performance on the hardware we employed is is approximately seven thousand triangles, with the simulation times being the limiting factor where mesh adaption takes less than 8% of the total time. The collision detection was slightly more expensive but was still efficient; it takes around 10-12% of the total time, therefore 80% of the overall costs were the mass-spring simulation and edge-length constraints. Figure 4.12 shows a mesh of this size, worn as a T-shirt by a static character that is simulated in real-time. Each four simulation updates and one

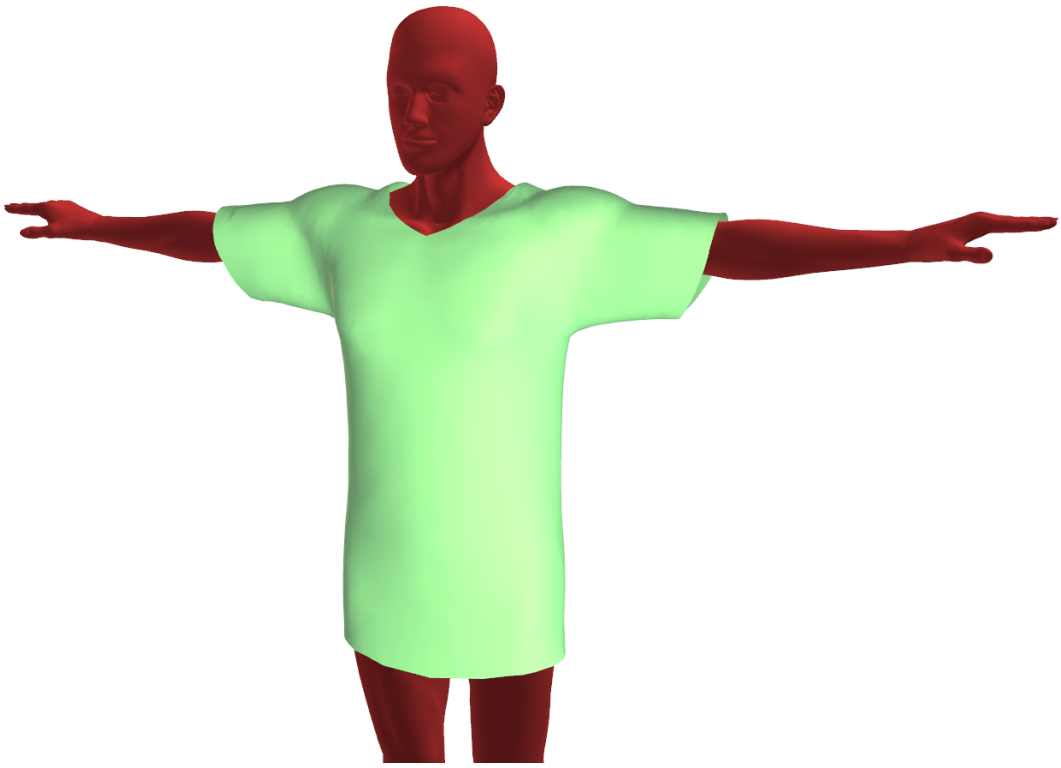

Figure 4.12: A T-shirt is draped on a static character with 67k triangles. The cloth is constructed from two base meshes seamed together, which totals 316 triangles, and are adaptively refined to 6199 out of a possible 20224 (Level 3) triangles. It takes approximately 22ms for each adaption and 4 simulation steps, running at 30Hz in real-time.

adaptive mesh update take a total of 22 ms running at 30 Hz, (e.g the simulation runs at 120 Hz). Figure 4.13 shows the corresponding adaptive hierarchy for the front mesh of the T-Shirt.

4.6 Summary

In this Chapter we have described the computer aided design process that we follow to create cloth patterns using 2D meshes which are seamed and combined together into an intermediate 3D mesh format. The process is demonstrated on a garment of a T-shirt to be dressed and draped on a 3D static character. To enable this, we implemented a robust collision detection scheme for the cloth against arbitrary

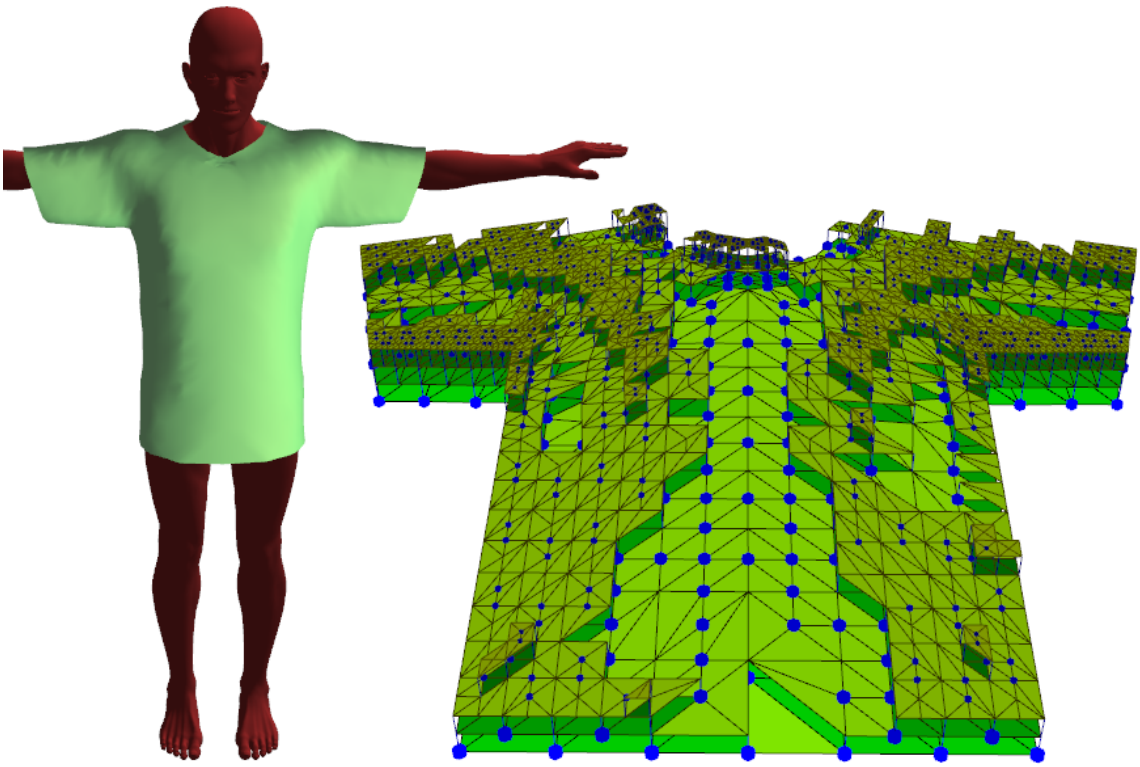

Figure 4.13: A T-Shirt draped on a static character, with the adaptive hierarchy of the front mesh illustrated.

triangle meshes and have integrated this into the adaption process, which together with seaming allows the real-time simulation of a detailed T-shirt on a virtual character. The next stage of our research is to move from a static character to moving animated character. Although the grid based approach is very efficient, it cannot support animated characters as it would need to be rebuilt each frame which is too costly. Therefore we have anticipated that much of the work and challenges will be in developing efficient collision handling for non-static objects.

The results show that the adaptive mesh would work well with additional level of detail approaches, for instance an off-screen character could be simulated using purely their base mesh (level 0). Figure 4.11 shows that our refinement method could cope well and be able to quickly return to detailed garments if the character were to

come back into view.

Chapter 5

Real-time Clothing using Adaptive Meshes with Animated Characters

5.1 Introduction

Collision detection costs can be a significant bottleneck for cloth especially with animated characters compared to static objects which can be partitioned into very efficient structures for collision detection. The mesh of animated characters is continuingly being deformed and so any bounding structures used for collision detection must be constantly updated every frame. In this chapter we advance our work on real-time clothing using our edge-based adaptive mesh, but instead of static objects, we perform the simulations on skeletally animated virtual characters. To this end we have developed a new method for fast collision detection using cylinders specially subdivided for regions of a character's mesh that we approximate with rigid sections for improved performance. Together with traditional bounding sphere hierarchies for deformable regions, we are able to perform real-time collision detection against a character's triangle mesh. Our adaptive mesh is augmented with additional refinement criteria based on visibility where large computational savings are achieved through back-face coarsening. We show how the adaptive mesh can be automatically controlled to achieve user defined targets for the number of triangles or time per

frame.

5.2 Character Animation

Computer animation is a very large field and animated characters bring additional complexities and costs to the simulation of clothing and there is a lot of work involved in their use. The simulation of clothing is especially difficult because it brings together many research areas together. In this section we will explain the main aspects of character animation for our work with clothing.

We employ skeletal animation which represents a character in two parts by a hierarchy of interconnected bones (also known as the skeleton or rig) and a surface mesh (or skin). The surface mesh is typically created in a T-pose or similar pose which aids the creation of the skeleton (rigging) and the process of skinning (attaching) the mesh to the skeleton. This pose is called the bind pose from which the mesh vertices are given a set of bone influences. If a vertex has a single influence, then it is said to be rigidly attached to the bone and will follow the relative translations and rotations of the bone exactly. It is the similar for multiple influences; a position is calculated for each bone as if they were rigidly attached but then all the positions according to each bone are combined using a weighted average.

A skeletal hierarchy of bones is required so that when a bone is rotated or translated its children are affected, for instance moving your upper arm causes relative movement of your lower arm and hand as well. Transformation matrices are composed for the skeletons bones; for each bone a world transformation matrix is calculated concatenating its parent's world matrix with its local one. The translation components of a bone's world matrix define its end position in world coordinates and the beginning of the bone can be found from its parent's position. In each skeleton there is a root bone with no parent that places the skeleton in relation to the origin

(0,0,0). In order to update the skin correctly, we must refer back to the bind pose to be able to relate each vertex to a bone. We can transform a vertex from its initial world space position into a bone's local space, and then transform it back to a new world space position which is relative to the bone's new position and orientation. To this end, we save the inverse matrix of the world transform for each bone from the bind pose; this matrix will transform a vertex into local bone space. Instead of performing the final calculation using two matrix multiplications, it is more efficient to concatenate a bone's current world matrix with the inverse bind matrix and use that instead. An animation for a character is made from a number of frames, where each frame describes a character's pose for that frame by bone positions and orientations for the whole skeleton. The process of updating the skin to match any pose of a character using the transformations is shown in Listing 5.1.

```
1  void updateSkin()
2  {
3    for(int b=0; b<boneCount; ++b)
4    {
5      // compute the matrix which transforms a vertex from the bind
           pose to the current pose.
6      skinMatrix[b] = boneWorldMatrix[b] * inverseBindMatrix[b];
7    }
8
9    for(int v=0; v<vertexCount; ++v)
10   {
11     Vector3 p(0,0,0);
12
13     for(int w=0; w<vertexWeight[v].count; ++w)
14     {
15       // transform the vertex from the bind pose to the current pose
             relative to an attached bone weighted it by its influence.
16       p += (skinMatrix[vertexWeight[v].boneIndex[w]] *
             vertexBindPosition[v]) * vertexWeight[v].boneWeight[w];
17     }
18     mesh->setVertexPosition(v, p);
19   }
20 }
```

Listing 5.1: Procedure for the update of the skin's surface vertex positions

5.2.1 Blending between Poses

We dress the character in the bind pose, we cannot just jump from one pose to a completely different pose in a single frame once the character is dressed; So for example, if we load an animation that doesn't start with the bind pose then we need a way to move the cloth to conform to the initial pose of the animation. We could try to create some temporary attachments between the cloth and the character's mesh or skeleton to use to move the cloth with the character in a single step. However, this is not easy and does not guarantee that the cloth will be positioned in a realistic or natural way in the new pose. Therefore we took the approach of generating a set of blended poses to move the character from one pose to another smoothly while simulating the cloth. The number of blended poses to generate and use are specified by the user since it is not worth the effort to automatically calculate how many is needed, it depends on the difference between the two poses. The character only needs to move slowly enough between the poses such that the cloth moves smoothly with the character without the discrete collision detection failing (it would fail if the character moved too far in a single step). The blended frames are calculated by interpolating bone matrices between the poses; for this we use spherical linear interpolation (Slerp) which is commonly used in computer graphics for animation.

5.3 Collision Detection with Skeletally Animated Characters

Collision detection with animated characters is expensive and could be a severe bottle neck that prevents the real-time simulation of clothing for all but low polygon characters and coarse cloth meshes. Efficient bounding structures can be constructed but it is the need to update or rebuild them each time the character moves that then becomes the bottleneck. We continue with performing only cloth-vertex collision checks

against the triangles of the character's mesh, but we no longer include the entirety of the mesh as we did with static meshes in the broad phase. Savings are realised by not considering collisions with the hands, feet or head, and therefore we do not have to maintain bounding volumes in these regions. The way the character is animated is important and must be considered to achieve good performance. The final intersection checks are made independent from the bounding structures such that we can employ two types of bounding structures efficiently. We perform discrete collision detection where the broad phase collects a list of potentially colliding triangles for each cloth vertex using the bounding structures, and in the narrow phase the vertex is tested against the list.

As we described earlier, we employ skeletal animation for the character and with this technique each vertex of the character's skin is connected to bones with weights. Bones connected with a higher weighting influence the position of that vertex more than those with low weightings; typically up to four bones are connected to any given vertex on the surface. In order to achieve real-time performance we exploit the fact that the greatest deformations of human's skin appear at joints and so for improved performance we consider that areas of small deformations can instead be approximated by rigid sections. That is, vertices that are influenced strongly by one bone can be skinned using that only that single bone and any small influences are ignored. For example, the middle sections of the lower and upper arms and legs can be considered rigid. We believe that this this simplification is justified due to the tight constraints imposed by real-time simulation and the tendency for collision detection to be a bottleneck for deformable objects like cloth. Furthermore, a viewer is unlikely to notice the difference unless inspecting the character in close detail and much of the time these areas will be covered by cloth. The character is therefore partitioned into two different types of regions, deformable and rigid, treating each differently for the

broad phase of the collision detection.

5.3.1 Bone Map

In order to create bounding structures, we need to be able to refer regions of the character to access vertices and triangles to place within the structures. It is useful to be able to load characters which are created in different 3D modelling and animation programs with different hierarchies or naming conventions for bones. We have created a bone map to indirectly map a bone name used for the collision structures to the name from the loaded in the file and find the index of the bone in the skeleton. Users can register an unlimited number of aliases for bones, for example the left upper arm bone may be registered as 'L Upper Arm', 'LUA', or however it appears in the file. The collision structures are created using the internal names, using a C++ enumeration type which specifies all common main bones for humans. This also allows a variable number of bones in the character, for example sometimes the number of bone segments for the spine varies. Bounding structures are specified for the maximum number of bones expected for any given character, and if there are bones missing from the loaded character then we simply do not create bounding structures for them.

5.3.2 Deformable Regions

Vertices that are skinned by more than one weight cause affected triangles to deform; we employ a standard bounding sphere hierarchy for collision detection in these regions. Many of these regions are around joints, vertices are concentrated and spheres can provide a good fit around them. Bounding sphere hierarchies are typically created by specifying two bone influences, such as the upper arm and the lower arm which would create a top level sphere around vertices around the elbow of the character. Some vertices and therefore triangles are influenced by more than two bones, so the

order of construction can affect the resulting bounding spheres. A triangle should only be placed in a single bounding structure, otherwise the bounding structures will be less efficient with more overlap and the collision detection could select the same triangle more than once as being potentially colliding with a cloth vertex. Therefore a triangle is checked that it has not been registered to another bounding structure before placing and registering it in a new one.

We construct the complete hierarchy in a top-down approach using a binary tree method. At each level the triangles within the bounding spheres are separated into two groups, split along the widest axis that the triangles span. Two bounding spheres are created at each level around the two groups and this is repeated until the bottom of the tree. At the bottom of the tree, the leaf nodes contain a bounding sphere which encompasses only a single triangle.

At runtime, each time the character is animated the bounding sphere hierarchy must be refitted. It would be too slow to reconstruct it completely, and as such we refit the hierarchy without changing its topology using a bottom-up approach. For every triangle, i.e. for every leaf node we calculate a bounding sphere that is centred on the average position of its three vertices. The radius is therefore the largest distance from the centre to one of the vertices. It is then straightforward to merge pairs of bounding spheres upwards through the tree to refit it without any further expensive operations on the triangles themselves, while still ensuring the spheres are fully encompassing. A cloth vertex can then be recursively tested against the hierarchy in world coordinates, returning a list of potentially colliding triangles.

5.3.3 Rigid Regions

One may pre-calculate simple bounding volumes to approximate the rigid regions as is typically done for static objects. The volume which naturally approximates the

rigid regions of our character is the cylinder. Our novel technique is to subdivide the cylinders, but importantly in order to preserve performance we do not use a deep recursive hierarchy (like with bounding spheres). We present a technique to directly access a list of nearest triangles within the cylinders. Each cylinder is sub-divided along its length into stacks, and then each stack is sub-divided into pie-shaped slices radially. Each slice contains a list of triangle indices that are either inside or overlapping it. A cylinder is constructed along a bone in the skeleton, vertices and triangles which are only influenced by the specified bone are inserted into it.

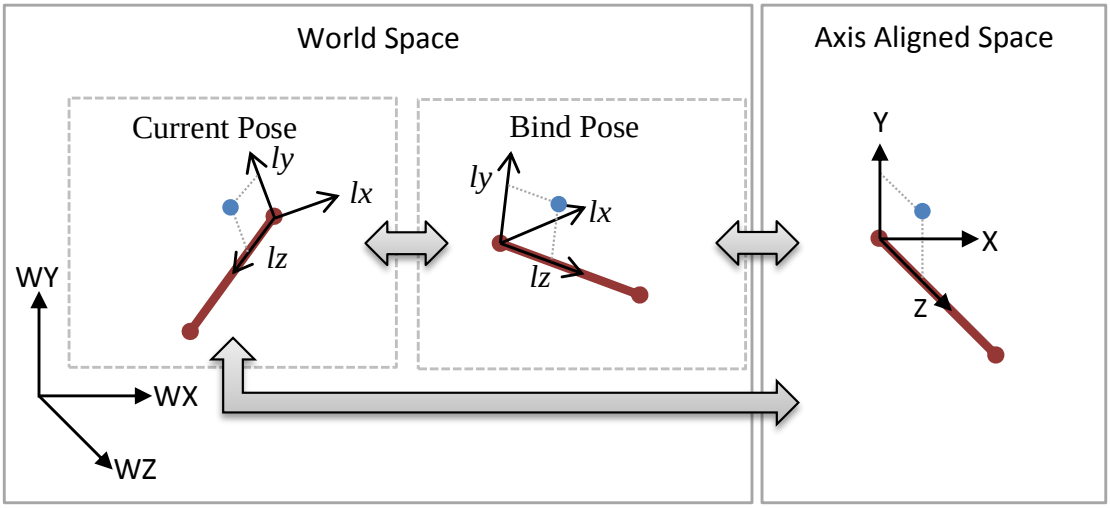

Figure 5.1: Left: A bone (red) is shown in world space with coordinate axis (WX,WY,WZ) in two poses: the current pose and the initial bind pose. An arbitrary rigid vertex (blue) from surface is selected which together with the direction of the bone defines local coordinate vectors (lx,ly,lz) for any pose. Right: A axis aligned coordinate system is shown, the start of the bone is placed at (0,0,0) and coordinate axis (X,Y,Z) correspond to local coordinate vectors (lx,ly,lz).

We perform collision detection with our cylinder structures in two coordinate spaces: world space to enable very fast early rejection tests for a vertex against the whole cylinder and a local axis aligned space for the final intersection tests. A bind pose is needed for skinning to be able to relate and transform a skin vertex from its initial position to its current world position depending on the current pose of

the character. We employ a similar approach for the cylinders using the bind pose, and so we initially define a cylinder in the bind pose along a bone. We select an arbitrary rigid vertex from surface in the bind pose to define its up direction, i.e. to define the Y-direction for the cylinder in its axis aligned space and the Z-direction is defined along the bone's length such that the selected vertex lies in the XZ plane (see Figure 5.1). We do not use the bind pose for collision detection at runtime, the direct transformation from the world space current pose to the axis aligned pose is used. However, the vertex positions from the bind pose are transformed to the axis aligned space for cylinder fitting process. We calculate the smallest radius (using the point-line distance to the main axis of the cylinder) possible that encloses all vertices. We bound the ends of the cylinder by the minimum (d_{min}) and maximum (d_{max}) distances along the main axis from the XY plane. The length of the cylinder is therefore the difference between the (d_{min}) and (d_{max}) and may be smaller or larger than the bone's actual length depending on the attached surface.

The next step of construction is to subdivide the cylinder into stacks (which are also cylinders) and then into slices radially from the centre each with their own radii. The cylinders are divided up into equally sized stacks along the Z-axis using a user specified length per stack rounded to up nearest integer number of stacks. Then each stack (which may be offset) is divided into a user specified number of slices radiating from the centre (see Figure 5.2).

Although the radii are always minimally fitting in each part of the cylinder; the best fit is achieved if we offset the stacks of the cylinder in a perpendicular direction from the main axis of the cylinder. We can calculate an minimally enclosing cylinder with an offset (see Figure 5.3) as follows: For every pair of vertices, (i, j) contained within the cylinder with positions $p(i)$ and $p(j)$, calculate a offset centre point, $(\frac{p(i)+p(j)}{2})$ and a corresponding offset radius. Then we find the pair with the

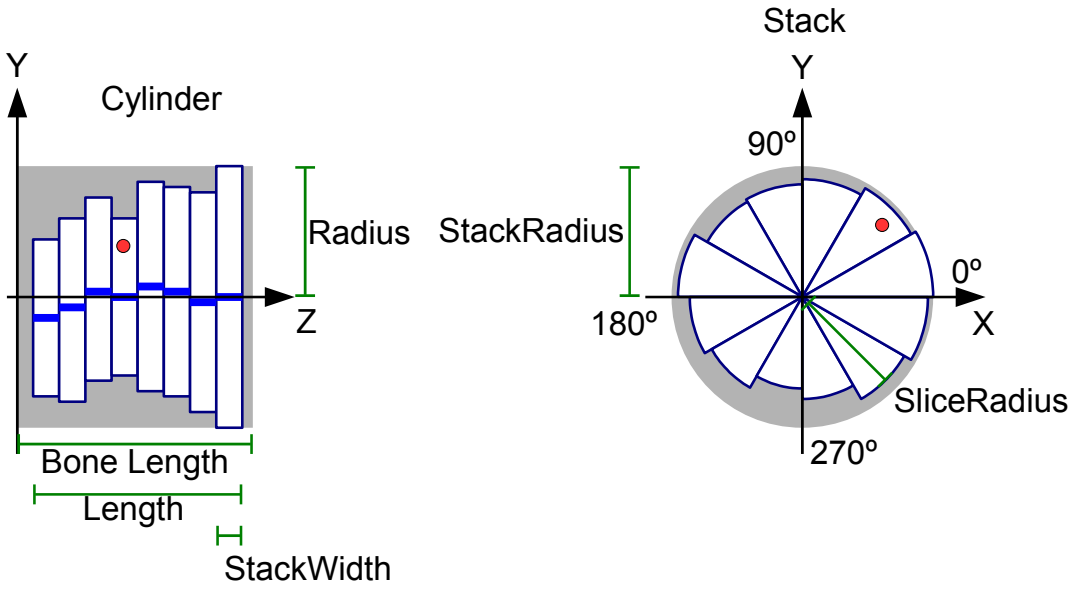

Figure 5.2: The figure shows two cross-sectional views of a cylinder bounding collision structure along a bone, the grey backgrounds show the maximum bounds of the cylinder. It is subdivided and its internal structure is shown, Left: The cylinder is divided into a number of stacks along the Z-axis and they also may be offset from this axis in the XY plane. Right: Each stack is divided into slices radially, one such slice is shown.

smallest offset radius which also satisfies the constraint that all other vertices are enclosed within it. The offset vector in axis aligned space is therefore given by the XY components of the offset centre vector and the Z component is discarded (it represents the distance along from the start of the bone).

The main advantage of this approach is that it allows very fast early out rejection tests in world coordinates. Given the cylinder's normalised local Z-axis ($axisZ$) and start position (A) in world coordinates then the distance along the cylinder (perpendicular to the cylinder's XY plane) for a point can be calculated: $distance_{\text{along}} = DotProduct(axisZ, point - A)$. So we can immediately reject this point if $distance_{\text{along}}$ lies outside of the cylinder's bounds, i.e. less than or greater than the cylinders minimum (d_{min}) or maximum (d_{max}) distances we computed earlier. It is also quite fast

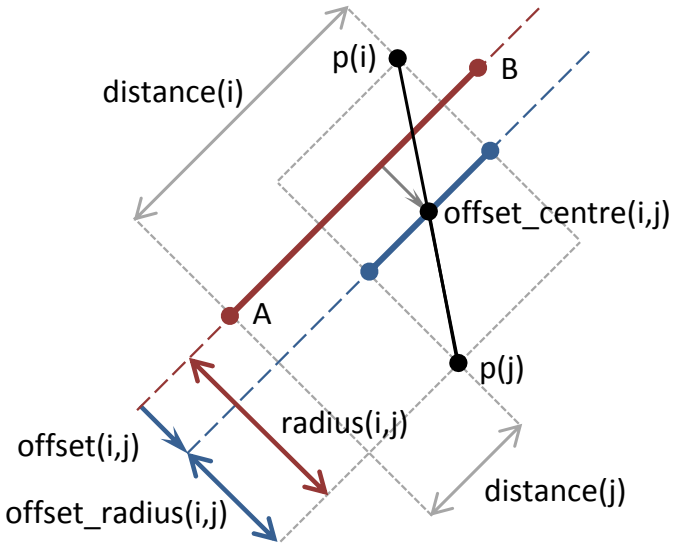

Figure 5.3: Construction of a cylinder for pairs of vertices (i, j) from the surface, the cylinder (A to B, red axis) can be offset (blue axis) to provide a tighter fit with a smaller radius. A minimally enclosing offset cylinder is found by searching all such pairs for the pair whose radius encloses all other vertices. The cylinders ends are bounded by a minimum and maximum distance along from A which bounds all vertices.

to check the radius against the point-line distance using: $closest_point = A + axisZ * distance_{\text{along}}$) with the radial distance (point-line distance) to the cylinder given by: $distance_{\text{radial}} = Length(point - closest_{\text{point}})$. It is possible to directly find which stack the vertex is in using the distance along the cylinder, and the stack's radius is compared. Then we transform the vertex into the axis aligned space to calculate the slice index using the angle around the Z-axis (in the XY plane) and perform a final radial check against the slice. If all checks pass then the slice's list of triangle indices is returned. Square roots can again be avoided by storing and using squared distances and radii in the calculations. The sub-division resolution used has no impact on the cost of a full vertex test (one in which there was no early rejection) against a cylinder, although memory usage and floating point accuracy could become an issue. At higher resolutions: each slice will provide a better fit and contain fewer triangles

which should reduce the effective cost of the collision detection against triangles since there will be less false positives.

5.3.4 Cloth-Triangle Collisions

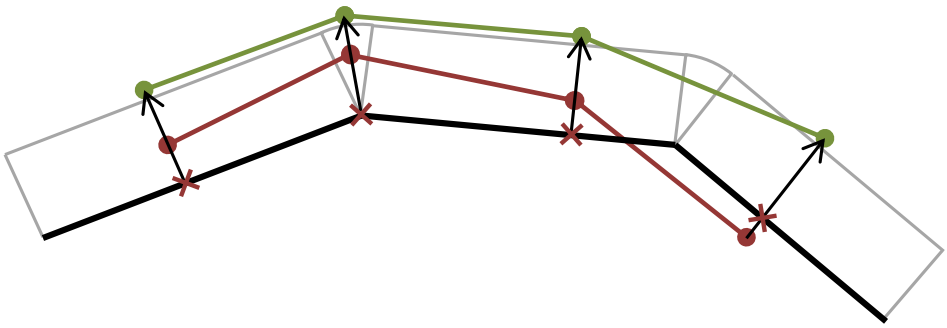

Figure 5.4: Collisions can be resolved by projecting cloth vertices outwards onto the offset surface (grey) by finding the closest points (red crosses) on the closest triangle of the surface (black). A cloth surface is initially shown in red and again after collision detection is performed in green.

We need to create an expanded surface for collision processing as we did for the static meshes and character in the previous chapter. However, it is not as easy as before where we could pre-calculate the surface and perform continuous collision detection between the previous and current position of a cloth particle. The effect of the character moving potentially invalidates the collision free state of the previous position of a cloth particle and also means the collision surface will need to be continually updated. The approach we take is to perform collision with an imaginary offset surface instead of the expanded surface and we must detect a collision using only current position of a cloth particle. Unlike the expanded surface where only the vertices were offset outwards their surface normals, all points on the offset surface are the same distance away from the original surface. We discussed in Chapter 2.2.2 that discrete collision detection (DCD) relies on small relative movements or tunnelling

can result. However, it is much faster to compute than the complex alternative of continuous collision detection that would need to consider movements of both the cloth and the character. So for performance we perform DCD with the offset surface of the character but since the surface is closed we can consider its volume. Therefore instead of looking for an intersection of a cloth particle moving through the surface, we just need to detect if a cloth particle is within the surface and then project it out onto the surface. We search for the closest triangle to a cloth particle; we can use it to check whether we are outside or inside of the offset surface. The particle is within the surface if it is below the triangle's plane. If the particle is above the triangle's plane then it is only inside of the offset surface if the distance from the particle to the triangle is less than the offset distance. So we need to find the closest point on the closest triangle, it lies either on a corner, along an edge or on the face. The direction to project the particle along extends from the closest point to the particle's position if it is above the surface or the opposite direction if it is below such that it points out of the surface (see Figure 5.4).

The approach is inherently self-correcting so we do not require a different approach for newly created vertices which may create within the surface as we did with static meshes. The only limitations are with respect to tunnelling effects, for example: if a particle or character moves so fast between two steps that the particle remains outside of the offset surface then no collision will be detected but their swept volumes between the steps may actually intersect. If a particle is within the offset surface then a collision will always be detected, however, the particle could be projected out of the surface in the wrong direction. For example, if a particle moves more than half-way into the surface than it will be projected out of the opposite side to where it entered. We have not seen these problems manifest since the cloth and character move relatively small distances each step where time steps are small (e.g. 8.33 ms for

120Hz updates) for stability.

Our collision structures (spheres and cylinders) are inflated to accommodate this collision surface offset, for an example of the cylinders see Figure 5.5. The list of potentially colliding triangles that is collected using the bounding structured is searched for the closest triangle for which the closest point is then computed then it will be projected out if there is a collision detected. The particle's velocity is set to the closest triangle's velocity if it is likely that there will be a collision next frame, i.e. the triangle is moving in the general direction of the cloth surface.

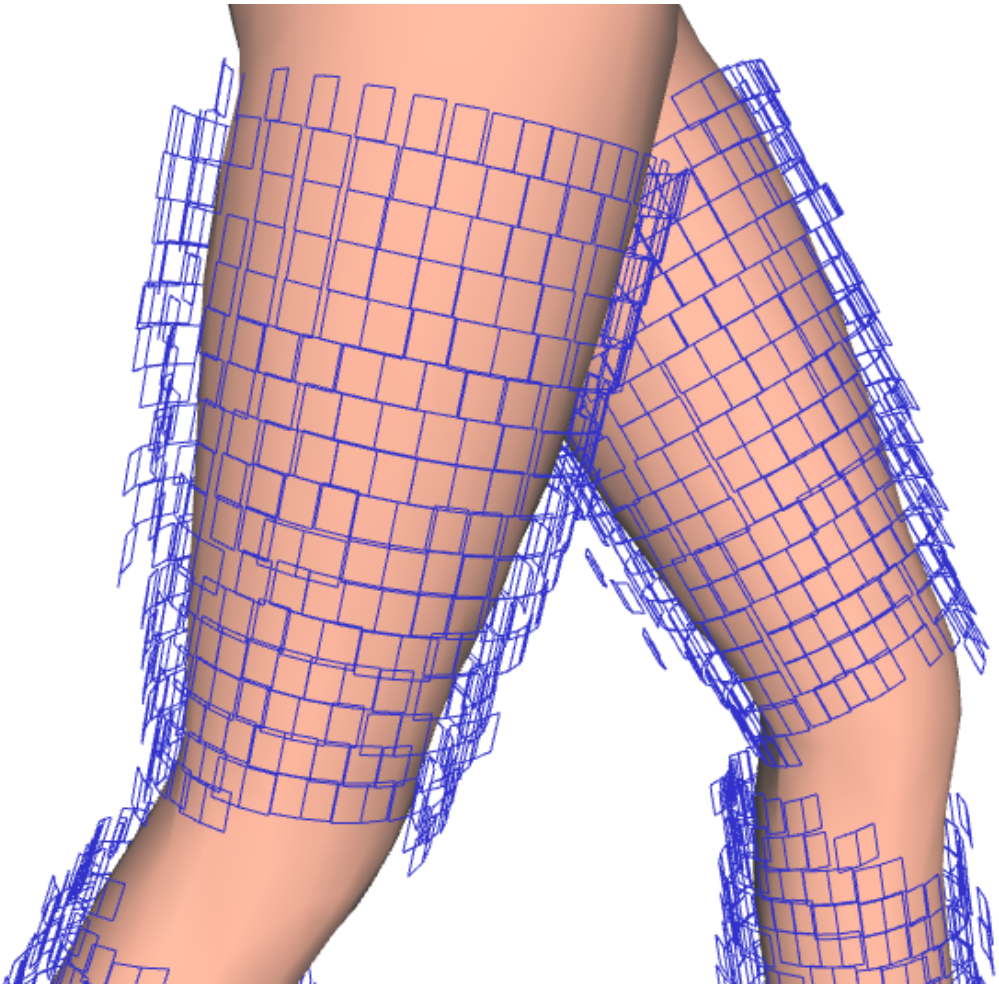

Figure 5.5: Screenshot of cylinder bounding structures on a character, subdivided into stacks of length 2 cm and 32 slices. Only the outer edges of the Slices are drawn for clarity.

5.3.5 Cloth-Character Synchronisation

When simulating garments being worn by an animated character, for optimal results we have found that their updates must be synchronised. If the character is being updated at a lower rate than the cloth, a garment will start to settle on the character in-between the character updates. So when the character is eventually updated and moves, the collision response can cause larger impulses on the cloth then if they were running at the same rate. This affects the smoothness of the cloth's motion compared to the character's animation. On the other hand, if the character is run at a higher rate than the cloth then collision there could still be smoothness issues on some frames if the rates are not multiples of each other. Also it is a waste of resources to update the character at a higher rate than the cloth.

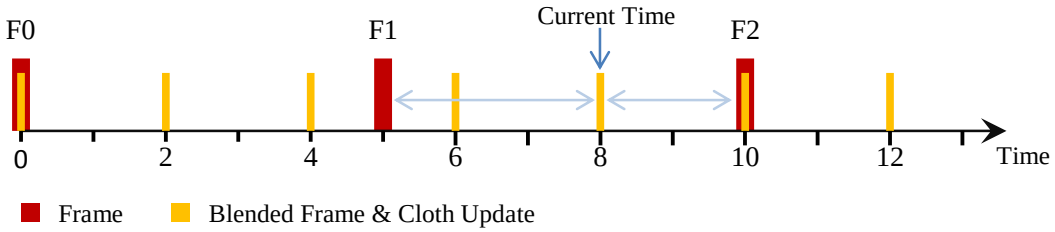

Figure 5.6: Animation frames (Red) and Synchronised Blended frames and Cloth Updates (Yellow) are positioned on a timeline. Blended frames are generated dynamically; the time between them is determined by the cloth time step (2 in this case) and use a blend factor of the proportion between them ($\frac{3}{5}.F2$ and $\frac{2}{5}.F1$ for the current time show).

We synchronise the updates using the garment's frame rate, the character is therefore also updated at the synchronised frame rate. This does not affect the playback speed of the animation but merely causes updates to occur at the same rate as the cloth. Blended frames are generated dynamically for the current time at each update by blending between the two closest frames, see Figure 5.6. To minimise the cost we only perform collision detection once and we do this as the final step of the cloth

update, so the character should be updated immediately before the cloth is updated. Therefore the cloth and character will only ever be rendered after all collisions have been resolved.

5.4 Cloth Simulation

In this work we employ a mass-spring network as described in Chapter 3.6 with an adaptive mesh. Springs are placed along the edges within the mesh, which influence both stretching and shearing, and bending forces are computed using the bending element approach. The simulation is advanced using Verlet numerical integration (see Section 3.6.4).

5.4.1 Unrealistic Stretching

Animated characters put much more demands on the simulation than we've seen up till now, and we've had problems being able to create good quality simulations using the mass-spring network. One of the main issues we have found with garments is unrealistic stretching under their own weight alone. This is particularly with long dresses; the stretching worsens as the mesh is refined but also is more pronounced as you move up the garment vertically (as the number of particles below increases, being pulled down by gravity). Although the edge length constraints procedure is quite effective (see Chapter 3.6.6), one problem with it is that as with all convergence problems, increasing the number of elements decreases the convergence rate. This is not something that Provot [Pro95] had to deal with since only uniform meshes were used. If we assume a single iteration is sufficient, as the mesh is refined we can increase the number of iterations in the same ratio as the increase in the number of edges as follows. This works well in practice and more similar results are achieved across uniform refinements, albeit at a (predictable) cost for heavily refined meshes,

strengthening the case for only refining where necessary. The numbers of iterations are global to the whole garment; this is not ideal for an adaptive refinement since coarse regions may undergo more iterations than needed and refined regions may undergo too few. There is not an easy solution to this, mesh segmentation could be considered but handling the borders between segments would be difficult and costly. Also a single iteration may not be sufficient for the coarse mesh.

We should also consider that garments draping on a static object are a different problem set to garments worn by an animated character. Stretching cannot be reduced by increasing spring forces without stability issues and the use of a smaller integration time steps or increased damping. High damping is not compatible with fast and fluid cloth motion on a moving character; it can make the cloth appear slow as if moving through a high viscous liquid whereas it's less noticeable on static objects. Also on static objects, edge lengths will eventually converge with time even if a single iteration is performed each step. Convergence is worsened by the animated character's constant movements, so on each frame the cloth's vertices need to converge to different locations than the previous frame to achieve the edge length constraints. So ultimately, even with the extra iterations to combat stretching; the weight of a dress still causes problems in the worst areas around the neck line and shoulders. The cost of combating stretching is already verging on 'too expensive' for a real-time simulation, and so we have added the ability to fix base (level 0) vertices of the mesh to the character's skin. After dressing, we automatically link the fixed vertices to the nearest character's triangle using barycentric coordinates with a surface offset (these can be edited and repositioned manually if needed). During the simulation the attached vertices will move and deform with the character's skin and give some savings since these vertices could be removed from the simulation and collision processing. The positions for vertices on higher levels are calculated using Bézier control points

based on PN-triangle patches [VPBM01]. After the simulation stage but before the normals are recalculated, we perform a smoothing of the mesh to produce a softer and less rigid look and feel for the cloth at a small cost. We use a weighted average of adjacent vertex positions and the weighting is defined by a Gaussian distribution calculated using adjacent edge lengths.

5.4.2 Dynamic Collision Contact Forces

In order to avoid unrealistic behaviour from simplifications made for real-time performance, such as the lack of a sophisticated friction model; we have looked to incorporate the collisions with the simulation using dynamic forces. When a collision occurs, we save the barycentric coordinate and a surface offset from the vertex to the collided triangle. We use it as a dynamic target position for a cloth vertex, based on the premise that clothes tend to move along with the character's skin due to friction and momentum. We use this dynamic target to calculate in-plane and out-of-plane weak spring forces to control sliding along and lifting from the surface, the cloth vertex's surface normal is used to define the out-of-plane direction with the in-plane being orthogonal to that.

These links automatically break after a set number of simulation steps or a predetermined distance is exceeded between the vertex and the surface dynamic position. Also the force is reduced proportionally with duration by multiplying by (1.0 - (durationSoFar/durationMax)), allowing the cloth to smoothly detach. A collision at any time will reset the link and its duration, whether or not the actual linked triangle changed.

5.5 View-Dependent LOD for Cloth

In this section we show how we have integrated view-dependent criteria for refinement and coarsening, with the Edge-based adaptive mesh for cloth simulation. Firstly introducing visibility determination for edges to use in the mesh adaption algorithm and then vertex based for use with back-face coarsening. Finally, we show measures assuring real-time performance.

5.5.1 Visibility Determination and Coarsening

The refinement is controlled by edges in the adaptive mesh, so we compute the visibility of the edges but it is often much easier to implement visibility determination for vertices. So we consider an edge as visible when either one of its two endpoints are visible (strictly it is partially visible). When an edge is not visible (both vertices are not visible), we either prevent it from splitting or otherwise we force it to rejoin if it has been previously split. An important consideration for performance is that this criterion overrides all others, such that when an edge is not visible the computation of calculating other potentially expensive criteria (such as curvature and collisions) is not performed. However, there is a problem with the Edge-based adaptive mesh due to the fact that it is incremental and requires a triangle to be fully refined before allowing further refinement on the next level. Non-visible regions will be completely coarse such that visible regions are not able to refine completely (after level 1) up to the border between them. Really, if an edge split is prevented and the edge is visible we should trigger refinement (edge splits) in the offending non-visible regions. Refinement needed in coarse regions may also be prevented; the iterative nature of the adaptive mesh will rectify this over a few steps. The end result of this is that the perceived transitional border of levels is moved onto the non-visible region of the cloth where it can no longer be seen. Furthermore, this approach to visibility based

coarsening allows any Boolean vertex visibility tests to be defined and they can automatically be used without needing them to be specifically designed for the edge-based adaptive mesh.

Back-face Coarsening

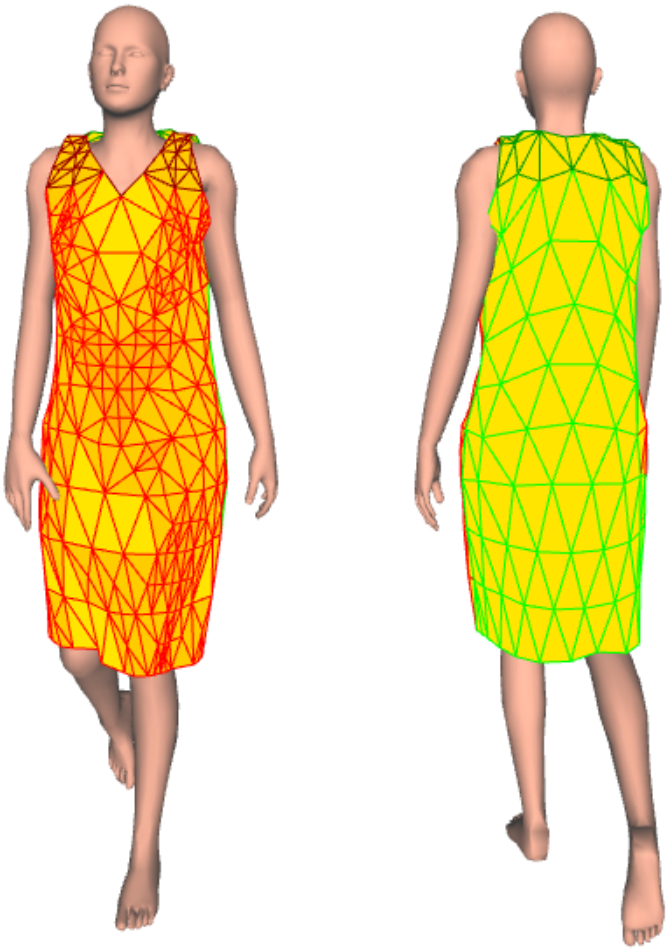

Figure 5.7: Screenshots of a character from the front and back camera views showing back-face coarsening.

We have experimented with the use of back-face coarsening for performance improvements by reducing the level of detail of the cloth in non-visible areas. Completely culling back-face regions from the cloth simulation would be very problematic, as they

are a major influence on the deformation of the front-facing regions. There are potential quality implications with back-face coarsening, however, we believe for real-time character clothing the trade off against performance is worthwhile. Typically, the back facing regions of garments are obscured by the front facing regions as well as the character's body. Furthermore, particularly in the case of virtual clothing, the rendering of the lining or inside of the garments is not often of interest. The vertex visibility test we use for back-face coarsening is based on surface orientation; the surface normal of each vertex is efficiently compared to the camera's view direction using a dot product. We have demonstrated this on a character, while simultaneously rendering the front and back views where the difference in quality and therefore performance savings are evident between the refined and coarse regions of the cloth, see Figure 5.7. The importance of moving the transitional between levels of refinement onto the non-visible areas is now evident, see Figure 5.8.

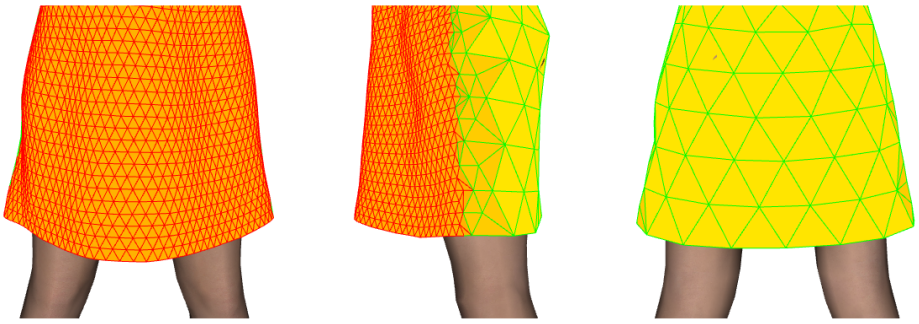

Figure 5.8: Front, Side and Back views of a character with back-face coarsening, edges which are not visible are coloured in green. The transitional zone between the fully refined and completely coarse regions is placed on the non-visible regions.

5.5.2 Assuring Real-time Performance

In real-time simulations, it is important that update times do not exceed an amount that disallows interactive frame rates. Adaptive meshes can be controlled in order to achieve this; one way is to impose a maximum triangle budget and stop further

refinement once the count is reached. The problem with this is that it relies on sufficient coarsening elsewhere such that areas that need to be refined are allowed too. Our approach is to try to balance the refinement and coarsening criteria in order to achieve a desired triangle count automatically. That is we can increase or decrease the curvature parameter to influence the number of triangles in the mesh. However, this is not completely predictable with the shape of the cloth dynamically changing in response to collisions with the character. It is more useful to assign the simulation a time budget instead of a triangle one, and if it exceeds this then steps should be taken to correct it. Conversely, when there is free time, the simulation detail can be increased. Distance can then be easily used to regulate the triangle or time budget, we use the distance from the camera to the character between user settable minimum and maximum distances and budgets. The adaptive mesh will therefore become more detailed when close to it and less detailed when far away.

Since it is not possible to achieve the exact desired budgets by modifying the criteria, smoothing was needed such that the cloth level of detail did oscillate too much from frame to frame with over and under shooting of the target. We use a 10-point weighted average for curvature criteria to control the adaption for this purpose.

After adaption, the current curvature angle used to control joining (coarsening) is added to the average which is weighted by the inverse of the target difference (target budget - actual achieved). The angle is then set to the updated weighted average, together with an additional correction to form the curvature criteria for the next step. The correction is a value based on the latest target difference, calculated as Angle * -(Difference / Budget). The angle used for splitting (refining) is more than that of joining and we find a four degrees difference works well. The initial conditions

are not very important, as long as the initial refinement conforms to the initial curvature criteria. Otherwise the first correction will be too extreme and cause a large overshoot next frame. Although this will eventually stabilise, it is undesirable so we perform three adaption steps (for a maximum refinement level set to three) before starting the simulation. For the time budgets; rather than following the above approach directly, we instead calculate an average cost per triangle and use that to set an appropriate triangle budget using: Triangle Budget = (TimeBudget / CostPerTriangle).

5.6 Results

The results presented are from tests performed on a PC with an Intel core i7 920 2.66GHz processor, 6GB RAM. We make use of OpenMP to parallelise loops, where there is no chance of simultaneous writes to the same memory location e.g. integration and collision processing. The character consists of 17043 triangles of which 4402 are used for collision with the cloth; 1978 rigid triangles in bounding cylinders and 2424 deformable triangles in bounding sphere hierarchies.

Table 5.1: Table of simulation parameters that we used for all of the experiments, these were chosen experimentally to give plausible cloth behaviour for all of the experiments.

Spring k k_g (N/m)	Bending k k_b (N/degree)	Density (kg/m^2)	Max. stretch (%)
80	0.002	0.1	1.05

We have performed a total of six experiments, four uniform (level 0 to 3) and two adaptive experiments with and without back-face coarsening. We use the same cloth properties for all the experiments, see Table 5.1. We set the splitting and joining angles to 16 and 12 degrees respectively for the adaptive experiments. The simulation is run at a rate of 100 Hz, using a time step, Δt, of 0.01 seconds for

the Verlet numerical integration. We automatically control the camera to test back-face coarsening fully; the camera starts facing the front of the character and slowly rotates 360 degrees around the character, pausing every 90 degrees such that the camera spends the same time stationary as it does rotating. Although the camera's position only directly effects back-face coarsening, it may cause differences in the rendering times so we use the same camera control sequence for all experiments.

Table 5.2: Average timings for updating the character each time it is animated. Total update includes the bounding structures cost (also shown separately), the triangle face normal and the origin-plane distance calculations needed for the collision detection. The average character skinning and rendering times are also shown.

Collision Data Structures			Character	
Cylinders (ms)	Spheres (ms)	Total (ms)	Skinning (ms)	Rendering (ms)
0.018	0.115	0.357	0.454	0.201

Table 5.3: Average triangle counts and the cloth's simulation, collision processing and rendering times for each of the experiments.

Exp.	Triangles	Simulation (ms)	Collision (ms)	Rendering (ms)
Level 0	238	0.358	0.310	0.032
Level 1	952	0.971	1.160	0.082
Level 2	3808	8.330	4.507	0.299
Level 3	15232	120.333	17.852	1.165
Adapt.	1631	2.978	2.047	0.148
Adapt. BFC.	914	1.430	1.156	0.099

A visual comparison of the six experiments can be found Figure 5.9. The mass-spring simulation was not able to capture significant extra detail with finer meshes; there are few wrinkles and most of the time the cloth is lying flat over the surface of the character. The adaptive mesh is therefore effective in comparison on a visual quality versus cost point of view; refinement improves the smoothness of the mesh compared to the coarse simulation. The level 3 simulation achieves little gains in

visual quality since it is largely similar in appearance to the level 2 simulation except for a slight increase in wrinkling in the lower parts of the dress. However, it has a much increased cost, where the edge length constraints to combat stretching dominates simulation the costs. Back-face coarsening introduces some differences to the shape of the visible cloth; however, it was very effective in reducing the total average costs by approximately 52% with a reduction of around 56% of the triangles. The simulation times particularly favour fewer triangles, see Table 5.3 for a breakdown of the timings for the experiments. The cost each frame for mesh adaption (included in simulation costs in the table) was on average 0.103 ms for the adaptive experiment and 0.067 ms for the adaptive experiment with back-face coarsening. More detailed collision statistics can be found in Table 5.4, showing the number and cost of collecting the lists of potentially colliding triangles together with the number of vertices in the cloth. On average, the collected list of potentially colliding triangles for each vertex contained 17.04 triangles, out of which the closest triangle was found and used to resolve the collision.

Table 5.4: Collision statistics showing the total number of potentially colliding triangles (PCT) and the time took to collect them for the bounding cylinders and spheres. The total number of cloth vertices in the mesh and the number of these that were found to be in collision are shown. All values shown are averaged over the entire simulations, the corresponding total time spent on collisions can be seen in Table 5.3.

Exp.	Cylinder PCT		Sphere PCT		Cloth	Cloth
	Num.	Time (ms)	Num	Time (ms)	Verts.	Vert. Colls.
Level 0	1364	0.058	658	0.081	125	60
Level 1	5318	0.239	2468	0.326	490	203
Level 2	21845	0.918	9775	1.225	1934	763
Level 3	92269	3.210	39420	4.291	7678	3165
Adapt.	11099	0.450	4393	0.605	846	338
Adapt. (BFC)	5990	0.316	2168	0.411	474	177

In order to evaluate the ability of the adaptive mesh to conform to triangle or

time budgets as discussed in Section 5.5.2., we have performed an additional two experiments where the budget is a predefined curve. Both methods show a similar ability at maintaining the budget, triangle budgets were on average 12.3% from the specified target and time budgets were slightly better at 8.2%. There is periodic over and under shoot due to the very unpredictable nature of the cloth's refinement while the character is moving. It may be possible to pre-calculate information that could help infer the correct angle to use at a given point in the animation. However, we feel this would negate the reason for using real-time cloth where you wish the cloth to respond to any character movement without running the simulation offline first. See Figure 5.10 for the achieved triangle budget and Figure 5.11 for the time budgets in relation to the allotted budgets.

5.7 Summary

In this Chapter, we have presented an approach for using adaptive meshes for clothing on animated characters. The deformable regions are a serious bottle neck for collision processing but the efficiency afforded by cylindrical bounding structures for the rigid regions has offered overall satisfactory performance. The cylindrical bounding structures have a very small update cost each frame (0.018 ms) compared to the bounding spheres (0.115 ms) that must be refitted in a bottom-up approach.

The adaptive mesh was extended to support visibility criteria that permit automatic coarsening in non-visible regions while leaving a necessary transitional border for increment refinement where it cannot be seen. This was demonstrated with back-face coarsening which achieves large computational savings while only minimally affecting the front-facing visible regions of the cloth. The difficult task of assuring real-time performance is partially solved by the adaptive mesh's ability to maintain time or triangle budgets through the automatic changing on curvature criteria.

The simulation method of using a mass-spring network has become cumbersome in our work with animated characters with complicated collisions; it particularly suffers from unrealistic stretching becoming worse with higher resolutions. Although we were able to combat these problems to some extent, the visual quality afforded by highest resolution mesh (level 3) is not much better than the lower resolution one as the increased resolution does not allow much more detailed wrinkling. However, the adaptive mesh is able produce higher quality simulations than the coarse mesh while adapting to any pose of the character. Important future work focuses on exploring other simulation methods that may perform better, even a method with a more expensive update costs could be used as long as it frequency of updates is lessened such that the total cost per second is not increased.

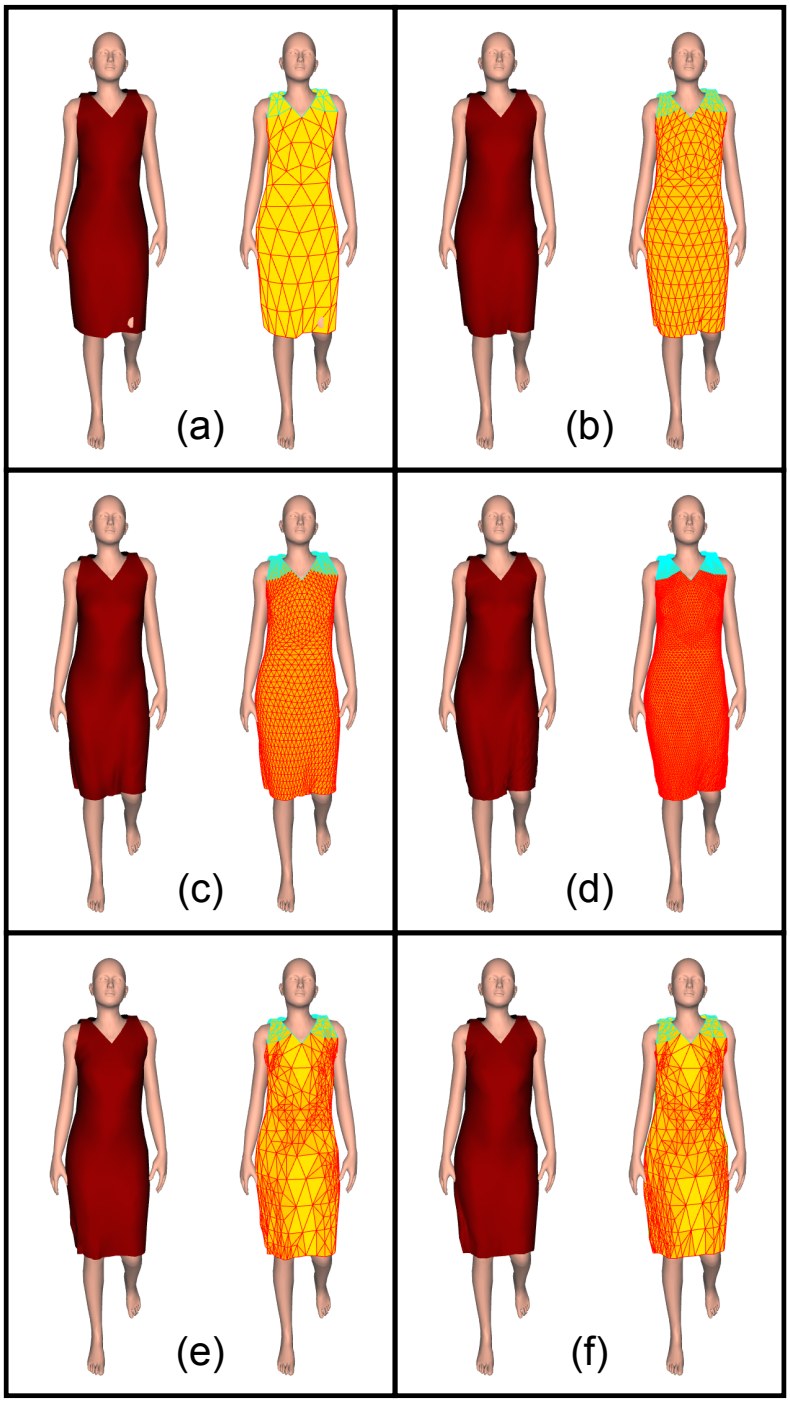

Figure 5.9: Screenshots of each experiment, taken at identical frame times shown rendered with lighting and adaptive mesh level-coloured triangles. Screenshots (a) to (d) correspond to Levels 0 to 3, (e) is the adaptive experiment and (f) is the adaptive experiment with back-face coarsening. (The fixed region of cloth on the shoulders are shown with coloured turquoise edges.). Note the visible interpenetrations on the level 0 simulation (a), there is insufficient vertex density.

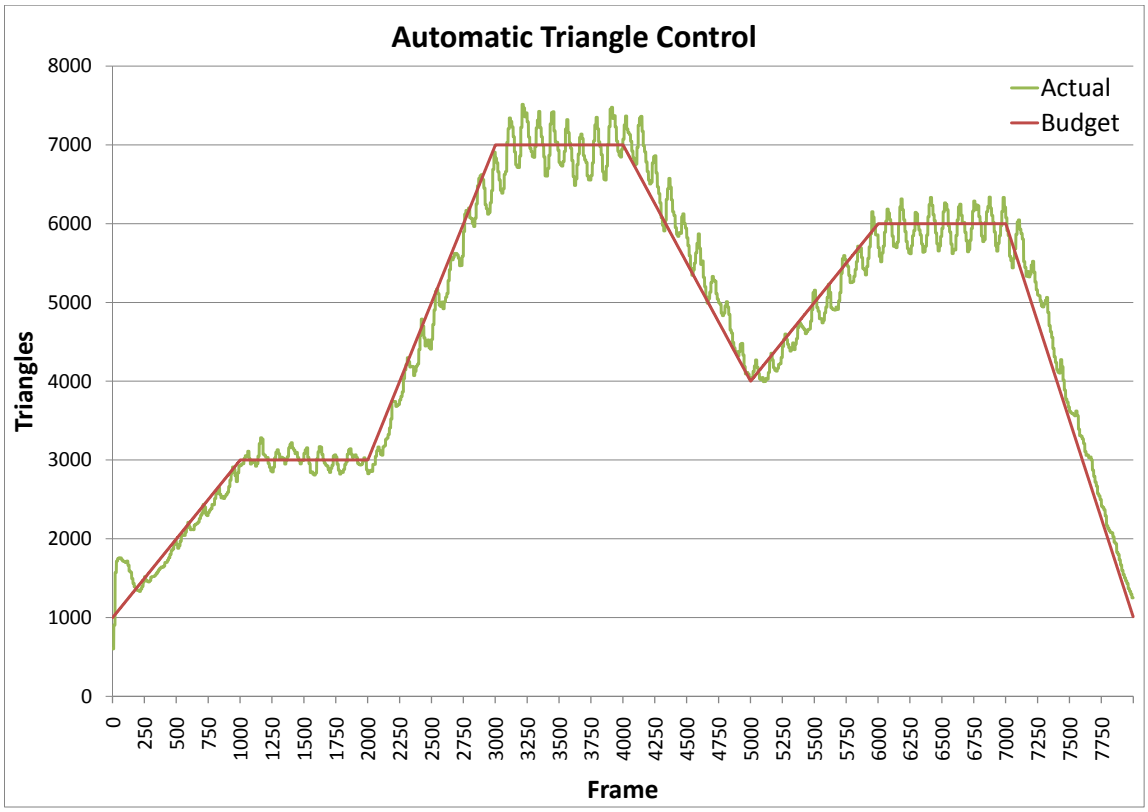

Figure 5.10: Graph showing the ability of the adaptive mesh to achieve a changing triangle budget.

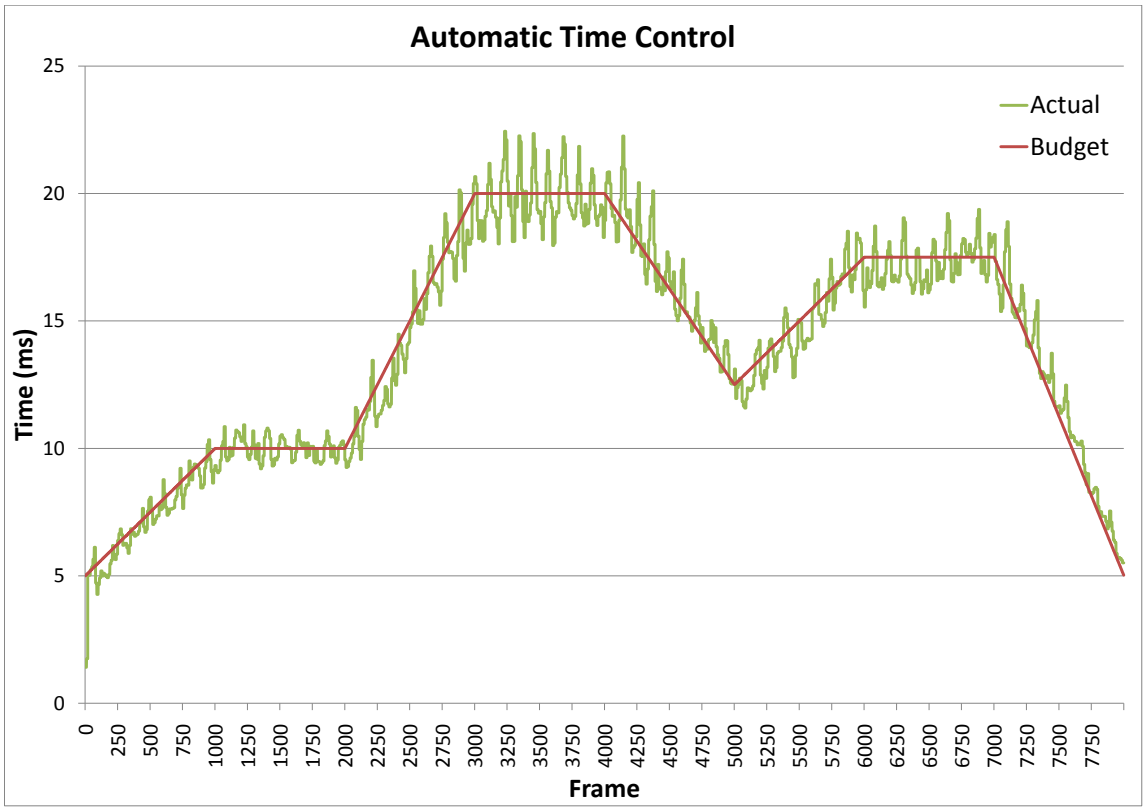

Figure 5.11: Graph showing the ability of the adaptive mesh to achieve a changing time budget, indirectly controlled by setting a triangle budget.

Chapter 6

Cloth Simulation with an Adaptive Curved Surface

6.1 Introduction

In this chapter we present a technique to simulate clothing using a coarse cloth simulation and an adaptive mesh for rendering. Focusing on real-time performance, we employ a fixed time step for the simulation and generate the surface of the adaptive mesh using a modified curved surface. The main contribution of this work is an extension of a curved surface to generate a more plausible cloth-like surface at a minimal overhead compared to the basic curved surface using an adaptive mesh. We also provide the user with parameters to control the surface.

6.1.1 Motivation

A popular technique for the simulation of cloth is to use a coarse mesh because of reduced costs; however, sufficient visual quality demands a more detailed fine mesh for rendering. These can be used in conjunction by the fine mesh typically sharing vertices with, or otherwise be coupled to the coarse mesh. Generally this is achieved by using curved or subdivision surfaces, or by other techniques specifically designed for wrinkling cloth. We have seen that adaptive meshes can provide increased performance compared to fine meshes, where a piece of cloth is adaptively subdivided

or refined to give greater local detail where it is needed most. The overheads of such approaches must remain small; we have presented a fast incremental edge-based adaptive mesh that can be used for real-time cloth simulation. Mass-spring networks are popular for cloth simulation, due to their simplicity and speed when working with coarse meshes. However, we found in the previous chapter that they can become cumbersome when working with animated characters, necessitating much expense by using many updates with small time steps for stability and measures to combat unrealistic stretching. We previously employed a simple curved surface for the fixed regions around the shoulders where the worst stretching occurs. Coarse mass-spring networks are also inherently more stable and easier to work with. So in this chapter, we propose to use a coarse simulation, but to achieve sufficient realism employ our edge-based adaptive mesh with a curved surface for rendering. The related work in the literature can be found in Chapter 2.1.2. We explain our adaptive curved surface approach in Section 6.2, the results are presented in Section 6.3 and finally we give a summary in Section 6.4.

6.2 Adaptive Curved Surface Approach

Curved surfaces are typically very smooth, whereas cloth will buckle and wrinkle. At the heart of our approach, we modify a curved surface in response to edge length to generate a better cloth like appearance. The adaptive mesh's base (least detailed) level vertices follow the positions of those of the coarse simulation exactly and vertices on finer levels are positioned according to the curved surface. This imposes only minimal restrictions on the simulation method, requiring only the vertices to match between the coarse simulation mesh and the base level of the adaptive mesh. The approach therefore involves copying vertex data from the coarse cloth simulation to the base level of the adaptive mesh each step. The complete algorithm proceeds as

follows:

1. Update simulation mesh (including collision processing).
2. Copy coarse vertex data (vertex positions and surface normals) from the simulation to the adaptive mesh and interpolate.
3. Update the adaptive mesh.
4. Update the surface.

6.2.1 Cloth Simulation

In this work we employ a mass-spring network just as we used in the previous chapters; however, in this instance using a separate coarse mesh for the simulation from the adaptive mesh. The coarse simulation uses a semi-regular triangular mesh for garments, they are created from seaming as described in Chapter 4.2.1. Springs are placed along the edges within the mesh, which influence both stretching and shearing. For every edge adjacent to two triangles, we have a cross-bending spring connected across it between the two triangles' opposite vertices.

As with the adaptive simulations in previous chapters, the simulation is advanced using Verlet numerical integration. As we are using a coarse mesh, the simulation is less expensive so we can afford to perform 5 iterations of the edge-length constraints (Chapter 3.6.6) per step to reduce unrealistic stretching in the mesh.

The adaptive mesh is not used for any part of the simulation, so some of the data we have previously stored in it are not needed (e.g. vertex force vector, mass and previous position). However, we still maintain the 2D material coordinates in the mesh together with undeformed rest lengths that they are used to calculate. One advantage of using the adaptive mesh is that non-uniform coarse meshes can be used more effectively, the coarse mesh can be designed for the simulation and the adaptive

mesh's criteria set for best visual results. For example, it is better to have a greater density of vertices where collisions are more likely and also where collision surfaces are less-planar e.g. more vertices near the thighs and chest. This is particularly favourable to this work, where we perform cloth-vertex against character-triangle collision detection. One could also perform collision detection with the adaptive mesh, but in this work we do not in order to get maximum performance, as collision processing is a major cost in cloth simulation. Instead, we leave a sufficient gap between objects and cloth such that no intersections occur.

6.2.2 Curved Surface

We base our curved surface on Curved PN-triangles [VPBM01], specifically the way in which Bézier control points are calculated for the edges to create a patch with an additional control point in the centre of the triangle. Curved PN-triangles are triangles that can be sub-divided in isolation, by using shared normals. This works by using the position and normal of each corner of a triangle to define a plane, and then points one third along each edge are projected onto their closest corner plane to calculate control points. As an alternative, Phong Tessellation [BS08] also uses similar projections but without control points, by barycentrically interpolating the projections of the flat points onto the plane defined by its normal at each corner. However they must then linearly blend between the flat surface and curved surfaces using a shape factor, they use a value of $\frac{3}{4}$ for convincing results. The tangents are not continuous across the patch borders, and it is designed to work in conjunction with Phong Shading [Pho75] for rendering of smooth meshes, it is especially suited for silhouettes with adaptive tessellation on characters and objects rather than cloth.

Instead of using triangle patches, we apply the PN-triangle approach just to edges (see Figure 6.1), with the two end points of edges fixed and use it to calculate the

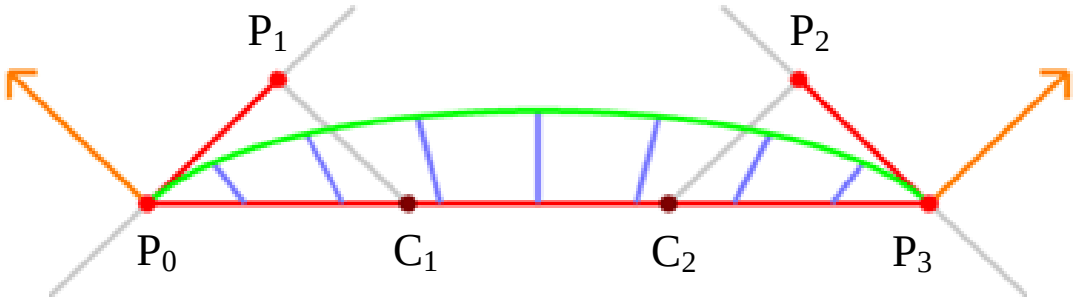

Figure 6.1: Surface normals at the ends of an edge (orange) are used to calculate two Bézier control points (P_1,P_2) just like in Curved PN-triangles [VPBM01], using the projection of the construction points, C_1 and C_2 (red) on to the normal planes (grey). (P_0, P_1, P_2, P_3) define a cubic Bézier curve (green). Blue lines show correspondence between vertices on the flat edge and the curve, for three recursive bisections of the edge.

positions for the vertices in-between. We start with the base edges of the adaptive mesh, positioning its central vertex and those in child edges. Then we recursively process the base triangles' internal edges, and finally the base triangles' internal sub-triangles. This works as if building scaffolding for the surface, calculating positions for vertices before processing the edges that have those as end points, see Figure 6.2.

6.2.3 Modifying the Curved Surface

Now that we have specified the basic curved surface using the PN-triangle inspired curved edges, we wish to modify the surface to generate a more wrinkled cloth like appearance. The Bézier control points of the curved edges are unfortunately not very convenient to modify in response to edge length. Also a curves' length cannot be calculated directly and many different curves can be found with the same length between any two points. Oshita *et al.* [OM01] tackle this with a large additional cost (60% more, increasing from 4ms to 6.4ms each frame); firstly their surface normals are weighted using the magnitudes of elastic forces (requiring data additional from

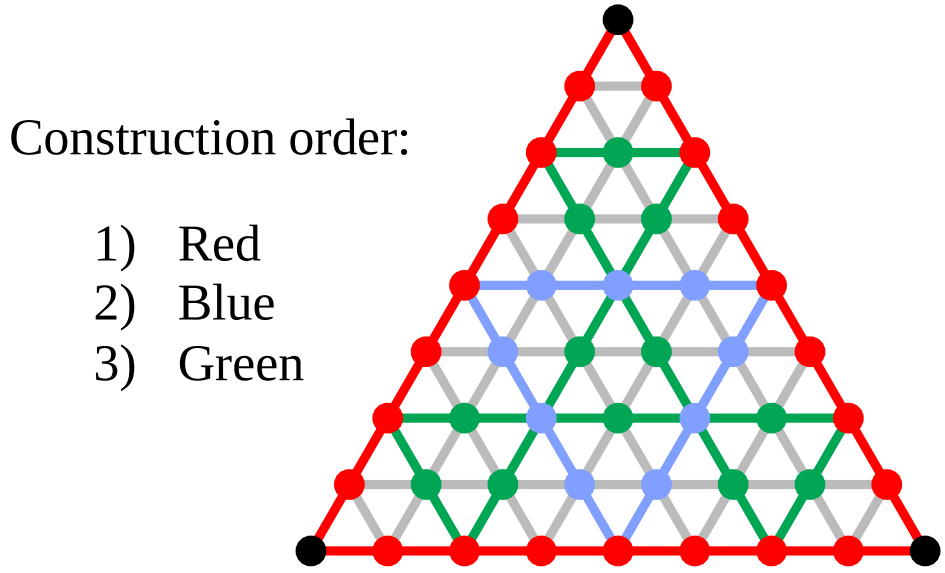

Figure 6.2: A triangle fully subdivided to level 3 into 64 triangles, i.e. 3 iterations of 1-to-4 splits. When constructing the surface, we must process edges such that edges and vertices are processed recursively with the following priority 1) Red, 2) Blue, 3) Green. The position of corner vertices (black), are copied from the coarse cloth simulation.

the cloth simulation); then to control curve length they split cubic Bézier (triangle) patches into 4 quadratic patches and iteratively move the centre point of each quadratic curve starting with an initial guess. Additionally they did not handle elongated edges and imposed constraints to prevent them in the simulation. Although cloth is often treated as inextensible, most real garments exhibit stretching as the body moves. Also it is possible to reason that when an edge is the correct length there should be no wrinkling; however, if we make such an edge flat visual quality will suffer due to smooth shading. Smooth shading breaks down when polygons are subdivided into co-planar regions. It is not acceptable to use less refinement for edges that are the correct length because it will conflict with any refinement criteria not

based on edge length. The curvature based criteria are especially important, so we make the argument that the surface must always be curved to some extent.

Therefore we follow a different approach focusing on performance and not imposing tight restrictions on inextensibility. We convert to use a Lagrangian polynomial to create a cubic curve whose property of intersecting all of its four control points is exploited for our needs. First we derive the equations for the interpolation curve. The curve is defined by (n+1) data (control) points: $(t_0, y_0), , (t_i, y_i), , (t_n, y_n)$, then the interpolation polynomial in the Lagrange form given is:

$$L(x) = \sum_{i=0}^{n} l_i(x).y_i \tag{6.2.1}$$

Where the Lagrange basis polynomials given by:

$$l_i(t) = \prod_{\substack{0 \le j \le n \\ i \ne j}} \frac{t - t_j}{t_i - t_j} \tag{6.2.2}$$

In the cubic case, there are 4 control points (n=3), this produces expected coefficients at the control points, intersecting all four control points perfectly:

t	$l_0(t)$	$l_1(t)$	$l_2(t)$	$l_3(t)$
0	1	0	0	0
$\frac{1}{3}$	0	1	0	0
$\frac{2}{3}$	0	0	1	0
1	0	0	0	1

Then for any value of t:

$$l_0(t) = \frac{(t - t_1)(t - t_2)(t - t_3)}{(t_0 - t_1)(t_0 - t_2)(t_0 - t_3)} = \frac{(t - \frac{1}{3})(t - \frac{2}{3})(t - 1)}{(0 - \frac{1}{3})(0 - \frac{2}{3})(0 - 1)} \tag{6.2.3}$$

$$= \frac{-9t^3 + 18t^2 - 11t + 2}{2} \tag{6.2.4}$$

$$l_1(t) = \frac{(t-t_0)(t-t_2)(t-t_3)}{(t_1-t_0)(t_1-t_2)(t_1-t_3)} = \frac{(t-0)(t-\frac{2}{3})(t-1)}{(\frac{1}{3}-0)(\frac{1}{3}-\frac{2}{3})(\frac{1}{3}-1)} \tag{6.2.5}$$

$$= \frac{27t^3 - 45t^2 + 18t}{2} \tag{6.2.6}$$

$$l_2(t) = \frac{(t-t_0)(t-t_1)(t-t_3)}{(t_2-t_0)(t_2-t_1)(t_2-t_3)} = \frac{(t-0)(t-\frac{1}{3})(t-1)}{(\frac{2}{3}-0)(\frac{2}{3}-\frac{1}{3})(\frac{2}{3}-1)} \tag{6.2.7}$$

$$= \frac{-27t^3 + 36t^2 - 9t}{2} \tag{6.2.8}$$

$$l_3(t) = \frac{(t-t_0)(t-t_1)(t-t_2)}{(t_3-t_0)(t_3-t_1)(t_3-t_2)} = \frac{(t-0)(t-\frac{1}{3})(t-\frac{2}{3})}{(1-0)(1-\frac{1}{3})(1-\frac{2}{3})} \tag{6.2.9}$$

$$= \frac{9t^3 - 9t^2 + 2t}{2} \tag{6.2.10}$$

Therefore, the final combined equation we use is as follows (taking four position control points, L_0, L_1, L_1, L_2 and a parameter, t):

$$\begin{aligned} L = &\frac{-9t^3 + 18t^2 - 11t + 2}{2}.L_0 + \frac{27t^3 - 45t^2 + 18t}{2}.L_1 + \\ &\frac{-27t^3 + 36t^2 - 9t}{2}.L_2 + \frac{9t^3 - 9t^2 + 2t}{2}.L_3 \end{aligned} \tag{6.2.11}$$

The four actual coefficients calculated for each value of t that are used at run-time are pre-calculated from the parts of the formula and stored in a lookup table for best performance. We only need a small lookup table because we perform edge bisections. Therefore we know in advance the set of t's required for edges at each level e.g. level one: $\{0.5\}$, level two: $\{0.25, 0.75\}$, level three: $\{0.125, 0.375, 0.625, 0.875\}$ and so on. Given a value for t, the lookup can resolve exactly to its four coefficients without approximation. We can use the index directly instead of t, so no floating

point conversion or calculations are needed. Given an edge, with two indices, A and B, corresponding to t at each end, e.g. $A = 0$ ($t = 0.0$) and $B = 2^{max\_level}$ ($t = 1.0$) for a base edge, the centre index, $C = (A + B)/2$. If C is an odd number, then the maximum level has been reached and no coefficients are stored for the next level. The table size required is therefore $2^{max\_level} + 1$ rows by 4 columns, for $t = 0.0$ to t=1.0 inclusive, we give an example in Table 6.1.

Table 6.1: Example of a look up table for the coefficients of the cubic Lagrangian polynomial supporting a maximum refinement level of 3, the value of t and its corresponding index and coefficients are shown.

t	Index	$L_0(t)$	$L_1(t)$	$L_2(t)$	$L_3(t)$
0.000	0	1	0	0	0
0.125	1	0.4443	0.7998	-0.3076	0.0635
0.250	2	0.1172	1.0547	-0.2109	0.0391
0.375	3	-0.0342	0.9229	0.1318	-0.0205
0.500	4	-0.0625	0.5625	0.5625	-0.0625
0.625	5	-0.0205	0.1318	0.9229	-0.0342
0.750	6	0.0391	-0.2109	1.0547	0.1172
0.875	7	0.0635	-0.3076	0.7998	0.4443
1.000	$2^3 = 8$	0	0	0	1

The Bézier curve that our surface is based on can be used to calculate control points for the Lagrangian curve simply as follows:

$$L_0 = P_0 \tag{6.2.12}$$

$$L_1 = CubicBézier(\frac{1}{3}, P_0, P_1, P_2, P_3) \tag{6.2.13}$$

$$L_2 = CubicBézier(\frac{2}{3}, P_0, P_1, P_2, P_3) \tag{6.2.14}$$

$$L_3 = P_3 \tag{6.2.15}$$

Also, for performance we hard code two cubic Bézier functions with the coefficients pre-calculated for $t = \frac{1}{3}$ and $t = \frac{2}{3}$ (Equations 6.2.13 and 6.2.14 respectively).

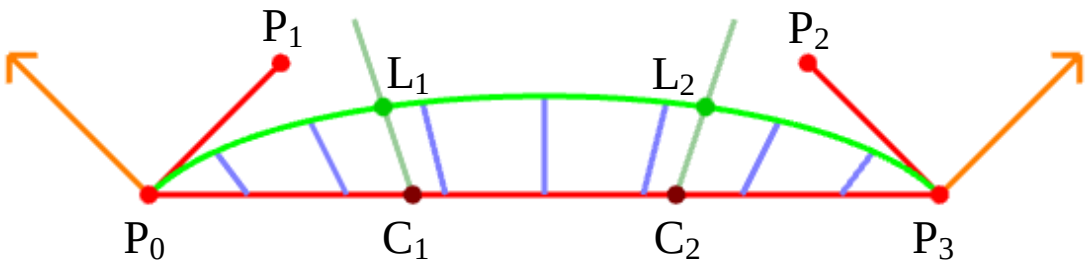

Figure 6.3: Bézier control points P_0, P_1, P_2, P_3 are used to calculate the two central control points (L_1 and L_2) for the cubic Lagrangian curve (green), at $t = \frac{1}{3}$ and $t = \frac{2}{3}$ on the Bézier curve. Blue lines show correspondence between vertices on the flat edge and the curve, for three recursive bisections of the edge.

We control wrinkling by moving the 2nd and 3rd control points (L_1 and L_2), along a line formed from their initial positions to the construction points (C_1 and C_2) respectively, see Figure 6.3. The aim of this is to change both the curved length and flatness in response to compression or stretching of the edge compared to its original length (RestLength). L_1 and L_2 are moved using the following formula:

$$L_n = L_n + (C_n - L_n) * M \text{ where } n = 0, 1 \tag{6.2.16}$$

We illustrate this with a curve modifier, $M = \frac{RestLength^2}{Length}$ on three curves with different edge lengths but all have the equal rest length, see Figure 6.4. The curved length is directly proportional to the edge length. If we take the edge length to be the desired curved length, it would be possible to iteratively calculate M to yield that as the curved length for any percentage of edge length, but there are two problems with this approach. Firstly, edges longer than 100% (stretched) will be represented by flat curves, which is a problem we discussed earlier. The desired curved length could be inflated so stretched edges are curved, but care would be needed as compressed edges would have very long curved lengths. Secondly, there will be a different set

of Ms for each unique relative configuration of the two edge normals. Looking up a set from the normal vectors is difficult; one must consider how many different cases to pre-calculate with a trade-off of accuracy and memory usage with some kind of interpolation between them. We should mention that hashing has previously been used for collision detection of dynamically deforming tetrahedrons [THM+03], using a spatial hash function of 3D vectors. It seems possible that a similar hash function for 3D vectors could be used to lookup the sets of pre-calculated M values, possibly by hashing on the difference between the two normals vectors.

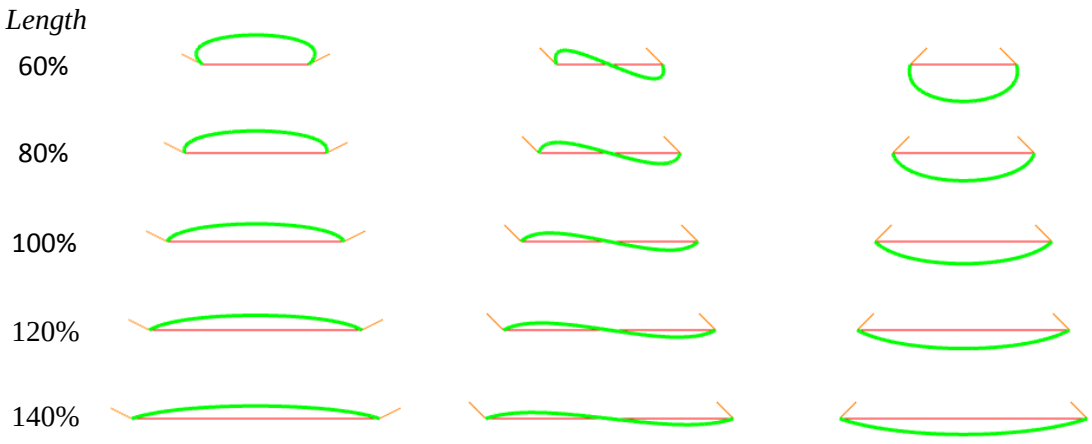

Figure 6.4: Diagrams to show the effect on three curve shapes at different percentages of rest length, for these we use a factor of: $M = (\frac{RestLength}{Length})^2$.

In our case, guaranteed efficiency and allowing stretched curved edges won over true length preserving curves. User control is often important particularly if the users are artists which wish to make adjustments the simulation to suit their needs, so we define several parameters to control the curves' response to edge length. The formulae are:

$$C = \frac{RestLength}{Length}.Compress \tag{6.2.17}$$

$$M = K_0 + K_1.C + K_2.C^2 + K_3.C^3 \tag{6.2.18}$$

Where *Compress* is a multiplier to allow the mesh to appear globally more or less compressed. K_0 is a constant factor, K_1, K_2 and K_3 are linear, quadratic and cubic respective factors to control the response to edge-length.

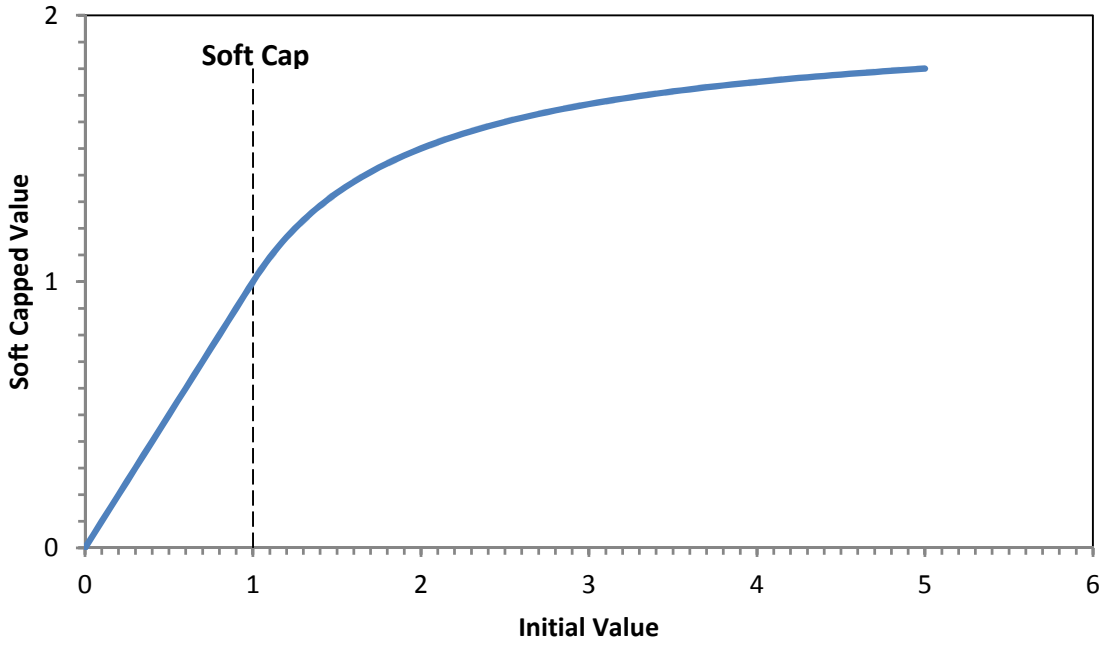

Figure 6.5: Graph illustrating the simple soft cap function we use ($SoftCap(M, SC) = SC + \frac{M-SC}{M}$). In this case a soft cap, SC=1.0 is shown, values less than this are not modified. We apply this to the curve modifier, M which is a dimensionless multiplier.

Modification of the original curve will affect the continuity of the surface, but significant discontinuities will only occur with significant compression; in areas where we expect greater wrinkling. However, we give the user a way to reduce this, by softly capping to the result of M. This prevents the surface looking excessively bumpy if there are large local compressions along edges while producing a smoother result than clamping the value to a maximum. The user can set this soft cap value as they

require, the following formula (illustrated in Figure 6.5) is used for when M exceeds the cap (SC):

$$SoftCap(M, SC) = SC + \frac{M - SC}{M} \tag{6.2.19}$$

6.2.4 Lighting Normals and Curvature Adaptive Criteria

Before we adapt the mesh, we interpolate the normals along the edges. This provides normals that are consistent with the coarse base mesh, no matter if the normal or positions of the adaptive surface have been modified by our curved surface. It helps avoid edges oscillating between splitting and joining, as the curvature criteria will give the same result whether the edge's child vertices are modified or not. Also these same interpolated normals are used to create the curved surface. Otherwise the surface would form a feedback loop with itself, using its current normals to calculate a new surface that has different normals, which in turn would create a different surface with again different normals and so on. The adaptive mesh's update (refinement and coarsening) is decoupled from the simulation and surface construction so they may be performed at different rates. The surface is always updated at the same rate as the cloth simulation. On any step where we do not update the adaptive mesh, we interpolate the normals at the same time as performing the surface construction. This provides slightly improved performance since we only iterate over the mesh once rather than twice in this case. Otherwise it is sufficient to only interpolate the normals once before we adapt the mesh because the adaptive mesh assigns any newly created vertices an interpolated normal as well, so the surface construction method will always have access to correct normals.

The interpolated surface normals can be thought of as construction normals, and unfortunately do not accurately represent the resulting curved surface (See Figure

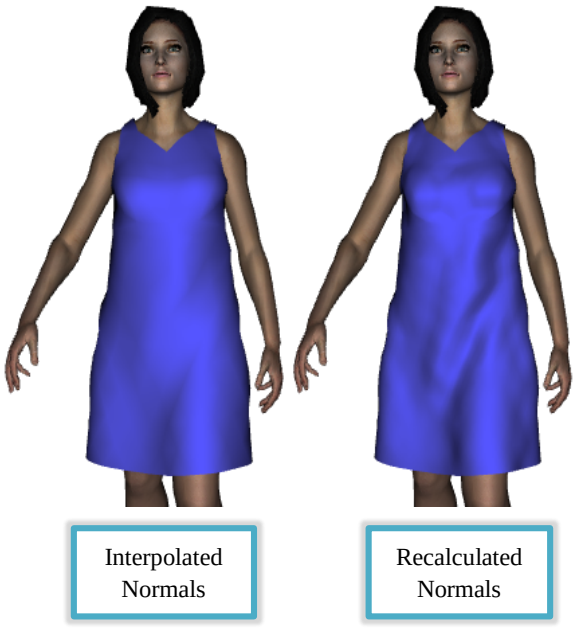

Figure 6.6: Interpolated surface normals are not sufficient to represent the curved surface, recalculated smooth surface normals reveal many hidden details.

6.6). Therefore we must finally calculate smooth normals for the entire surface at a cost; for each vertex we sum the adjacent triangles' normals weighted by their area and normalise.

6.2.5 Buckling and Edge-Length Adaptive Criteria

We have seen that an interesting alternative to curvature as criteria for adapting the mesh is edge length, this can simulate buckling behaviour. We implemented this simply, such that an edge can be split when it is when it is compressed past a certain percentage of edge length i.e. when $Length\% < Split\%$ and re-joined when $Length\% > Join\%$. However, this also presents a problem in that we do not wish the surface positions to effect the refinement as it is the coarse mesh that should drive it, we can use calculated bisected edge lengths instead. This means that the compression percentage is the same for an edge and its children. If we used this directly, as soon as an edge was compressed over the threshold, it would try to recursively refine infinitely

(limited by only the defined max level). We tackle this by imposing that successive levels must be more compressed before they are allowed to refine further; therefore we modify the criteria depending on the level of the mesh it is operating on. A user defined value which decreases $Split\%$ and $Join\%$ by a percentage each level, $Level\%Modifier$ (with a value less than one) is employed. For example, if a level 0 edge is set to be split at 80%, then a modifier of 90% means than a level 1 edge will split at 72% and a level 1 edge will split when compressed to 64.8%. The final criteria used for each level are therefore given by:

$$Split\%_{level} = Split\% * Level\%Modifier^{level} \tag{6.2.20}$$

$$Join\%_{level} = Join\% * Level\%Modifier^{level} \tag{6.2.21}$$

In order to improve performance we calculate the edge length as the final step in the simulation after collision instead of at the beginning. This does not affect the simulation, since it is used for the edge spring forces on the next frame and is still valid since the vertex positions are not modified in-between. However, if it is needed by the adaptive mesh this frame, it is copied from the simulation mesh to the adaptive mesh to be used by the edge length criteria (for joining and splitting), saving the need to recalculate it. The adaptive mesh must calculate the lengths for edges not part of the simulation, but only top-most edges (those without parent edges) are calculated with their children being cheaply assigned bisected (50%) lengths thus bringing further savings.

6.3 Results

In order to quantify the performance and appearance, we have tested our method with cloth simulations based on three experiments as follows: A) a of square horizontal

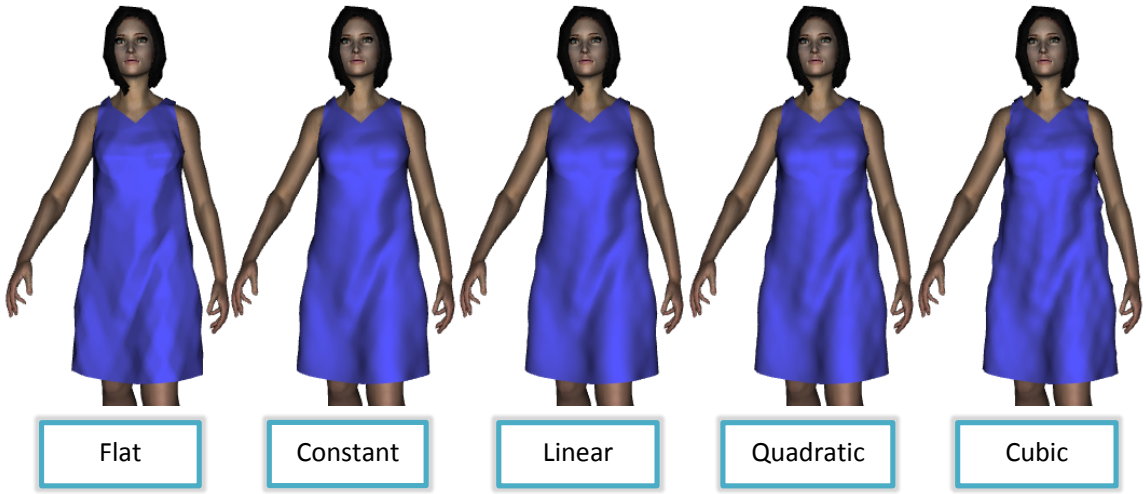

Figure 6.7: Screenshots of character wearing a dress (uniformly refined to level 2), with $\{K_0, K_1, K_2, K_3\}$ set to different values, Flat=$\{0,0,0,0\}$, $Constant = \{1,0,0,0\}$, $Linear = \{0,1,0,0\}$, $Quadratic = \{0,0,1,0\}$, $Cubic = \{0,0,0,1\}$, all with no soft cap.

piece of cloth pinned at two corners and dropped, B) a flag blowing in the wind and C) a dress worn on an animated virtual character. Experiments A and B run at a frame rate of 60Hz with time steps of 1/60 seconds, Experiment C runs at 80Hz with time steps of 1/80 seconds. We perform uniform and adaptive refinements to test the curved surface for each experiment. In all simulations with adaptive refinements, we perform it every five frames, i.e. so its cost is amortised over five frames. Additionally for Experiment C, we perform another adaptive simulation employing back face coarsening (discussed in Chapter 5.5.1). The results presented here were performed on a PC with an Intel core i7 920, 2.67 GHz processor using a single core; and an Nvidia GTX 285 graphics card using OpenGL for rendering with dynamic vertex buffers and Phong pixel shaders. The times for copying data from the coarse simulation to the adaptive mesh are included in the coarse simulations times, but in all cases were less than 0.005 ms. We use the same surface parameters ($Compress = 1.0$, $K0 = 0$, $K1 = 0.55$, $K2 = 0.55$, $K3 = 0.20$, $SC = 2$) for all

three simulation setups. However, screen captures of the dress (Experiment C) can be found in Figure 6.7 to illustrate the effect of different surface parameters.

The performance of the experiments A and B can be found in Tables 6.2, 6.3 and screen captures in Figures 6.8, 6.9. The first thing to notice is A has greater simulation costs than B even though they share the same sized mesh, this is because the edge length correction comes into effect more with experiment A. The effect of gravity on the hanging cloth causes more stretching than on the flag, the flag experiences a wind force that acts against gravity to some extent (the direction of the wind force was around 50 degrees upwards from the horizontal and to the right). The wind force we applied is not constant; the magnitude of force is attenuated with time and distance using a simple sine function where the wind has a cycle duration and velocity). This generates more natural movements of the cloth and causes a constantly changing refinement in the adaptive case. The curved surface construction is very fast and actually cost less than the surface normal recalculation, however, as discussed the surface normal calculation is a very necessary part.

Experiment C, the dress, involved collision detection and a virtual character where the other experiments did not. The character consists of 30834 triangles, it takes 2.34 ms to animate and skin each frame. There are 15 bounding sphere hierarchies (containing 3058 triangles) taking 0.56 ms to refit, whereas 9 bounding cylinders (containing 646 triangles) only take 0.013 ms to update. Rendering of the character takes 4.61 ms, the total cost for the character is therefore 7.52 ms each frame. We had still had problems with the weight of the dress causing too much stretching around the shoulder areas although much less than we experienced in the previous chapter. The problem was that the collision approach could be defeated with the very coarse simulation mesh where the dress could slip down the character. This happens when the separation distance between adjacent vertices (linked by springs) becomes

so much that for two connected vertices on the shoulders; one of them slides down the front and the other slides down the back of the character. Expensive solutions for this include increasing vertex density of the coarse mesh, performing more iterations of the edge length correction procedure or detecting and resolving edge collisions. However, mindful of performance we decided again to fix the offending shoulder vertices to the character. Overall the collision costs were a significant part of the total cloth simulation costs with the coarse mesh, they were between 1.21 to 1.23 ms (66.8% of the simulation costs). The dress also experienced some high frequency jittering, mainly around the character's chest while the character is in motion. It is hard to determine the exact cause, but it seems to be an artefact of moving cloth vertices multiple times (by the edge length correction procedure) for each single collision check combined with the local shape of the geometry of the character and coarse simulation mesh. When the cloth is stretched over a very rounded surface such as the character's chest, the procedure causes the cloth to shrink inwards under the surface. So when the collisions are finally resolved, coarse vertices can sometimes move large distances imparting large impulses on them. If we were to perform collision detection after each iteration of edge length correction the problem would be lessened, but the cost is prohibitively expensive to perform it multiple times each step. We were able to filter out much of the jittering by restricting the distance a cloth vertex may move in a single frame by imposing a speed limit to only problematic groups of vertices. It would be beneficial to explore the effect of Provot's procedure [Pro95] on collision detection in detail in the future, considering local curvature or comparing surface directions (cloth and collision surface) to isolate and constrain problematic vertices automatically. We would like to note that this problem is a limitation of only the cloth simulation and not with the curved surface approach we have presented in this chapter.

Table 6.2: Simulation A) a square horizontal piece of cloth pinned at two corners with a 250 triangle coarse simulation mesh. Rendered triangle counts are given, and times as average costs per frames are given in milliseconds (ms).

Level	Triangles	Coarse Simulation	Adapt	Surface Update	Surface Normals	Render	Total
0	250	0.419	-	0.004	0.029	0.023	0.483
1	1000	0.419	-	0.098	0.114	0.061	0.699
2	4000	0.418	-	0.360	0.464	0.270	1.519
3	16000	0.421	-	1.637	1.909	1.017	4.992
Adaptive	1286 (avg.)	0.426	0.411	0.099	0.125	0.079	0.815

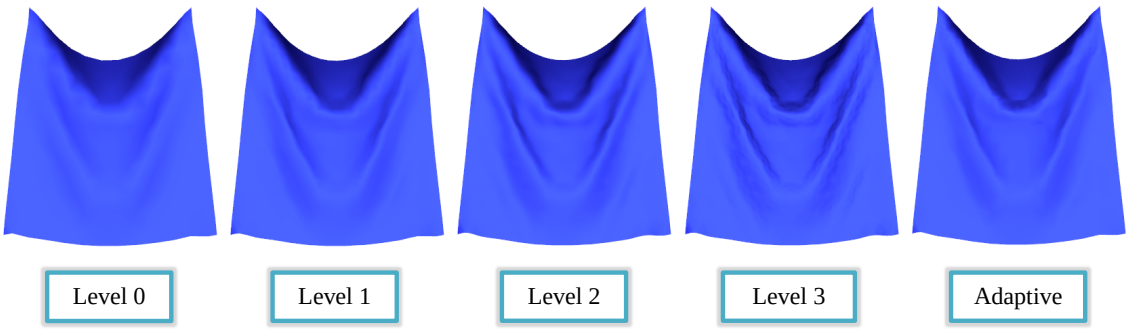

Figure 6.8: Simulation A), Screen captures from the simulation of the piece of cloth pinned at two corners showing it uniformly refined from level 0 (coarse) to level 3 and the adaptive refinement, these correspond to the results in Table 6.2.

6.4 Summary

We have presented a simple method to generate a curved surface for cloth in real-time with the integration of edge-length based parameters. The use of the adaptive mesh allows non-uniform coarse meshes to be used more effectively, with criteria set for best visual results. The smooth normal calculation is a significant cost but is important as interpolated normals are not sufficient, obscuring much of the surface detail. We have demonstrated the approach on a piece of hanging cloth, a flag and a character wearing an animated dress. In the case of the dress, back-face coarsening proved suitable and effectively reduced the combined cost of the surface construction, surface

Table 6.3: Simulation B) A cloth flag consisting of a 250 triangle coarse simulation mesh blowing in simulated wind. Rendered triangle counts are given, and times as average costs per frames are given in milliseconds (ms).

Level	Triangles	Coarse Simulation	Adapt	Surface Update	Surface Normals	Render	Total
0	250	0.365	-	0.004	0.022	0.022	0.419
1	1000	0.372	-	0.077	0.087	0.050	0.593
2	4000	0.368	-	0.272	0.348	0.210	1.204
3	16000	0.368	-	1.121	1.428	0.788	3.713
Adaptive	1206 (avg.)	0.365	0.485	0.097	0.118	0.080	0.748

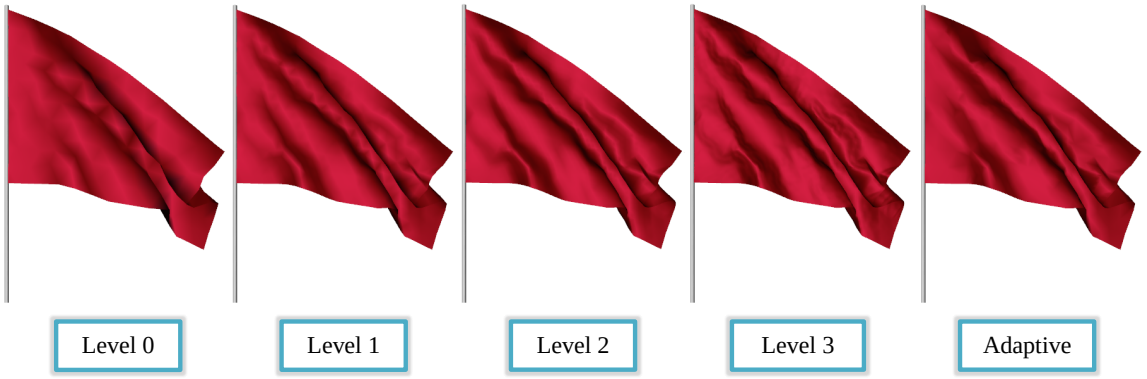

Figure 6.9: Simulation B), Screen captures from the simulation of the flag showing it uniformly refined from level 0 (coarse) to level 3 and the adaptive refinement, these correspond to the results in Table 6.3.

normals and adaption by 24%, including the coarse simulation costs and collision we made a more modest saving of 9.3% overall. Our surface mesh is suitable for further processing by additional techniques that require connectivity information, i.e. the normal calculation and rendering. It is able to capture medium scale wrinkles but unable to model very fine ones, such that refinement of the mesh past level 2 causes the surface to appear more bumpy than wrinkled. This leads to the possibility of future work for the development of a hybrid approach, using dynamic texturing based approach to introduce fine detailed wrinkles in conjunction with the work presented in this chapter.

Table 6.4: Simulation C) A dress on an animated character with a 488 triangle coarse simulation mesh. Rendered triangle counts are given, and times as average costs per frames are given in milliseconds (ms). The collision costs are included in the coarse simulation cost, and were between 1.21 to 1.23 ms (66.8% of the simulation costs).

Level	Triangles	Coarse Simulation	Adapt	Surface Update	Surface Normals	Render	Total
0	488	1.81	-	0.01	0.04	0.08	1.94
1	1952	1.81	-	0.16	0.17	0.16	2.30
2	7808	1.82	-	0.84	0.78	0.48	3.93
3	31232	1.83	-	3.33	2.98	1.67	9.81
Adaptive	3013 (avg.)	1.83	0.36	0.38	0.41	0.26	3.24
Adpt. BFC	2103 (avg.)	1.84	0.27	0.27	0.31	0.22	2.91

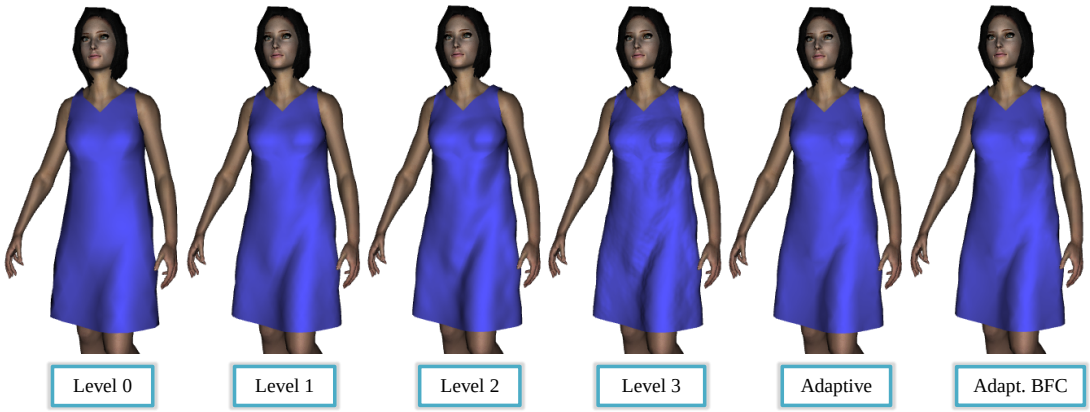

Figure 6.10: Screenshots of character wearing a dress showing levels 0 to 3 and the two adaptive surfaces (with and without back face coarsening, BFC), these correspond to the results in Table 6.4.

Chapter 7

Conclusions

7.1 Discussion and Conclusions

Since the 1980's cloth simulation has moved from being predominately based in textile research to becoming an important research interest in the field of computer graphics. The real-time simulation of cloth and in particular clothing remains a difficult task, in part due to the myriad of techniques from many areas that must be employed efficiently together. Before simulation can begin; garments must be designed and meshes created, which can have significant impacts on the final behaviour and the cost of the simulation. Coarse meshes can be used to achieve fast simulations, although they cannot accurately represent all possible shapes of very flexible materials like cloth. On the other hand, the use of fine meshes for large scale and detailed cloth simulation is unfortunately only achievable offline. Interactive and real-time applications cause severe computational constraints on the simulation of cloth, but there are many applications for its use in virtual reality, games and even virtual shopping software. Collision detection is almost always required for cloth simulation, but it represents a significant cost which can quickly grow out of control if additional measures are not taken. Level of detail techniques are commonly employed for real-time rendering in computer graphics for performance and it is for this same reason that adaptive

methods such as adaptive meshes have been developed for cloth simulation. Adaptive meshes aim to be a hybrid of coarse and fine meshes where local detail is created only as needed, exploiting that simulation and collision costs scale proportionally with mesh density. Adaptive meshes can improve the performance considerably but have been rarely used for virtual clothing and mainly feature for cloth draping in the literature.

We presented our main contributions in Chapter 3, namely the development of the edge-based adaptive mesh which is a novel approach for the adaptive refinement of a triangular mesh. The main difference from previous adaptive meshes is that we do not perform the direct subdivision of quads or triangles in the mesh. Such approaches often require special handling of the T-junctions that can result between two adjacent quads or triangles; this can happen if each are not subdivided to the same level or they are otherwise not subdivided in a conforming manner. Our approach is edge-based by the fact that refinement and coarsening of the mesh is controlled by two main operations on the edges; an edge split including the creation of central vertex for refinement and an edge rejoin for coarsening previously split edges. After which an efficient state-based retriangulation approach is followed for triangles that are adjacent to newly split or rejoined edges. The refinement pattern specified by the state of a triangle's three edges does not allow T-junctions to occur in the mesh. The approach is incremental and builds a hierarchy of levels; triangles must be fully subdivided into four before further refinement on higher levels is permitted to maintain the semi-regular refinement of the mesh. This strategy has permitted a very fast refinement approach; a 32 triangle base mesh can be refined by four levels to 8192 triangles for a total cost of 3.53 ms and subsequently coarsened back to 32 triangles in 1.57 ms. At all times connectivity of the mesh is maintained, both important for the refinement procedure but also important for the simulation and the surface normal

calculations. Refinement and coarsening criteria are specified by edge-based Boolean functions; we demonstrated the adaptive simulation of pieces of cloth on geometric objects with the use of three different kinds of criteria: curvature, edge length and collision. We found that the cost of evaluating criteria often exceed the refinement costs where very few changes occur between frames. It was for this reason we decoupled the adaptive update from the simulation, so that in our tests the adaptive mesh cost an average of 7.51% of total costs when updating it at 30Hz while the simulation ran at 120Hz.

The process of creating clothing, and the real-time adaptive simulation of the clothing draping on a static character was described in Chapter 4. We paid particular attention to the seaming of 2D meshes and their implications on supporting material coordinates within the mesh, where discontinuities occur across seams. A robust collision detection scheme was implemented using a pre-calculated grid based approach allowing the demonstration of collision aware refinement with clothing.

We advanced our work in order to perform the adaptive simulation of clothing on animated characters in Chapter 5. Collision detection was tightly integrated with the character animation and skinning methods employed. We made a small contribution in the form of a new type of bounding structure based on a cylinder subdivided both length ways and radially. The cylinder structures were employed for vertices that were approximated at being rigidly attached to a single bone for performance. Together with traditional bounding sphere hierarchies that were employed for deformable joints, overall real-time performance of the collision detection was achieved. However, in this work the simulation method using a mass-spring network became cumbersome when working with animated characters with complicated collisions; and it particularly suffered from unrealistic stretching that became worse at higher resolutions. This prevented a proper comparison of the adaptive simulation compared to fine meshes

(such as the level 3 uniform refinement), since the fine simulation cannot be regarded as a high quality ground truth. There were no particular fine wrinkles generated and the differences in many regions of the cloth were subtle except for the bottom free handing portion of the dress. However, the adaptive simulation can be considered higher quality than that of the coarse simulation because it was able to dynamically and more accurately approximate the curved areas better while preventing any visible intersections with the character's skin. However, the coarse simulation alone is not sufficient in this regard to quality; neither does it produce a plausible cloth animation since visible intersections are evident.

Also in Chapter 5, the extension of the adaptive mesh to support vertex-based visibility criteria was also described; using edge criteria to combine the Boolean visibility status of each edge's end vertices. This enabled large computation saving by permitting the coarsening of back-facing regions of clothing such that overall costs for a dress were reduced by approximately 52% with a reduction of around 56% of the triangles. The back-face regions do influence the front facing ones so back-face coarsening does introduce some differences; however, the savings afforded by this technique cannot be ignored.

Chapter 6 describes a novel approach for the real-time rendering of cloth using an adaptive curved surface which is simulated by a coarse mesh. The main benefits of employing coarse meshes are the reduction of the cost of the mass-spring network and collision detection while improving the stability and behaviour of the cloth. Standard curved surfaces are often designed for rendering smooth surfaces that are approximated with fewer polygons but have smooth lighting normals. Bézier control points are calculated following the approach inspired by PN-triangles patches [VPBM01], however, we converted to use a Lagrangian interpolation polynomial which permitted the simple integration of edge-length based parameters to generate a more plausible

cloth look and feel with small overheads. The approach is particularly able to model medium scale wrinkles, but is unable to model very fine one where higher refinement levels appear bumpy rather than wrinkled. Only so much can be achieved without moving the shared coarse vertices, however, there is a danger that moving them could introduce visible intersections with the character's surface which would need to be detected and resolved. Considering this, a particular attraction to the approach is its simplicity and speed where it has proved sufficient to only perform collision detection with the coarse mesh using a small offset from the characters surface. Back-face coarsening technique was again demonstrated and effectively reduced the combined cost of the surface construction, surface normals and adaptive mesh by 24%, a more modest saving of 9.3% was achieved overall if the coarse simulation and collision costs are included.

In conclusion we have advanced adaptive meshes into realms of real-time use for clothing where before it had only been achieved offline. The real-time physical simulation of clothing remains a difficult task due to the relativity limited computational budget that is available compared to offline simulation cloth. Both prior work and the techniques presented in this thesis will benefit from the trend of ever faster hardware in the future, however, there is still scope for improvements and new techniques for the real-time simulation of clothing.

7.2 Future Work

There are several areas of future research that could lead on from the work presented in this thesis. The first is the investigation of the use of more advance cloth simulation methods with the edge-based adaptive mesh. Although mass-spring networks seem simple and fast, the specific issues with stretching and stability can require

great expense to be resolved. For example the four times increase in the number of triangles for level 3 compared to the level 2 refinement resulted in an approximate 14.45 times increase in the simulation's computational cost (Table 5.3). This large discrepancy was due to the repeated edge-length constraints that were required to combat excess stretching, but compared to the collision costs which only increased by 3.95 times. The alternative to increase the spring constant which would increase both the simulation and collision costs since to remain stable a smaller time step with a higher update rate would have to be employed. The overall balance of computation time versus quality is difficult given real-time constraints and the effects of collision detection cannot be ignored. Even if real-time performance cannot be realised with a high quality simulation method, the edge-based adaptive mesh could still provide beneficial performance improvements. A related avenue of research is with regard to the evaluation and comparison of simulated cloth with reality, specifically in the context of this thesis the realism of adaptive cloth simulations needs to be studied. Then the adaptive simulation with a mass-spring network could be compared to both more advanced simulations but also to real cloth. It may then become possible to automatically set refinement criteria to achieve the best realism.

A specific large cost with the adaptive mesh is the constant calculation and monitoring of the edge criteria that control refinement and coarsening. We previously discussed that we decouple the adaptive update from the simulation for this reason as when the cloth is moving slowly very few changes are needed in the adaptive refinement each frame and the cost of evaluating criteria becomes dominant. However, it would be better to not to delay update of the refinement while finding a new way of improving the performance for the evaluation of criteria. To this end, future research could focus on the identification and partitioning of edges or small groups of them based on expected outcomes of criteria. For instance, curvature criteria evaluation

could be prioritised by the using the current rate of change of local curvature with a prediction of when the current curvature is likely to exceed the criteria's refinement thresholds. Also collision criteria evaluation could exploit additional broad phase collision detection techniques to prune away whole groups of edges.

An additional major area of future exploration lies in the use of parallel processing, particularly considering the use of GPUs. The edge-based adaptive mesh permits a very fast incremental refinement of triangular meshes on the CPU particularly benefitting from large cache sizes, we have seen performance decline for large meshes that exceed the CPU's cache size. The CPU's inherent friendliness to random access of data and recursion (recursive hierarchy of levels in the mesh) in processing is heavily relied upon. Whereas GPU's tend to have poor random access performance but have very high sequential throughput so locality of data is of the utmost importance. In the past GPUs did not support hardware recursion, however, for example NVidia's Fermi architecture [WKP11] supports a subset of C++ in CUDA including recursion while the hardware also provides fully cached data accesses to improve the performance of random access. Given the advancement of hardware and the continuing improvements to general purpose computing on graphics processing units (GPGPU) in recent years, this will be an important and worthwhile area of research. The main challenges with the edge-based adaptive mesh will be to reformulate the tasks of edge splits, edge rejoins, and retriangulation in an efficient way for parallelism. A major benefit to performance may be realised if all aspects i.e. simulation, collision detection and rendering can be performed on the GPU without the overheads of transferring data to and from the CPU each frame.

Bibliography

[AHS09] K. Atkinson, W. Han, and D. Stewart. *Numerical solution of ordinary differential equations*, volume 81. John Wiley & Sons Inc, 2009.

[AMT03] N. Adabala and N. Magnenat-Thalmann. A procedural thread texture model. *Journal of Graphics Tools*, 8(3):33–40, 2003.

[AMTF03] N. Adabala, N. Magnenat-Thalmann, and G. Fei. Real-time rendering of woven clothes. In *Proceedings of the ACM symposium on Virtual reality software and technology*, pages 41–47. ACM, 2003.

[And06] S. Andrews. Cloth simulation with a rigid body physics engine. Technical report, DISCOVER Lab, University of Ottawa Ontario, Canada, 2006.

[ASDB08] M. Azahar, M. Sunar, D. Daman, and A. Bade. Survey on real-time crowds simulation. *Technologies for E-Learning and Digital Entertainment*, pages 573–580, 2008.

[BA04] E. Boxerman and U. Ascher. Decomposing cloth. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 153–161. Eurographics Association, 2004.

[Bau72] B.G. Baumgart. Winged edge polyhedron representation. Technical report, DTIC Document, 1972.

[BFA02] R. Bridson, R. Fedkiw, and J. Anderson. Robust treatment of collisions, contact and friction for cloth animation. In *ACM Transactions on Graphics (TOG)*, volume 21, pages 594–603. ACM, 2002.

[BH02] D.C. Brogan and J.K. Hodgins. Simulation level of detail for multiagent control. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1*, page 206. ACM, 2002.

[BHW94] D.E. Breen, D.H. House, and M.J. Wozny. A particle-based model for simulating the draping behavior of woven cloth. *Textile Research Journal*, 64(11):663–685, 1994.

[BK04] J. Beaudoin and J. Keyser. Simulation levels of detail for plant motion. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 297–304. Eurographics Association, 2004.

[BKN02] Y. Bando, T. Kuratate, and T. Nishita. A simple method for modeling wrinkles on human skin. In *Computer Graphics and Applications, 2002. Proceedings. 10th Pacific Conference on*, pages 166–175. IEEE, 2002.

[BMF03] R. Bridson, S. Marino, and R. Fedkiw. Simulation of clothing with folds and wrinkles. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 28–36. Eurographics Association, 2003.

[BS05] T. Boubekeur and C. Schlick. Generic mesh refinement on GPU. In *Proceedings of the ACM SIGGRAPH/Eurographics conference on Graphics hardware*, pages 99–104. ACM New York, NY, USA, 2005.

[BS08] T. Boubekeur and C. Schlick. A flexible kernel for adaptive mesh refinement on GPU. In *Computer Graphics Forum*, volume 27, pages 102–113, 2008.

[BSBK02] M. Botsch, S. Steinberg, S. Bischoff, and L. Kobbelt. Open mesh - a generic and efficient polygon mesh data structure. In *OpenSG Symposium*, 2002.

[BTH+03] K.S. Bhat, C.D. Twigg, J.K. Hodgins, P.K. Khosla, Z. Popovic, and S.M. Seitz. Estimating cloth simulation parameters from video. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 37–51. Eurographics Association, 2003.

[BW97] D. Baraff and A.P. Witkin. Partitioned dynamics. Technical report, CMU-RI-TR-97-33, Robotics Institute, Carnegie Mellon University, 1997.

[BW98] D. Baraff and A. Witkin. Large steps in cloth simulation. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 43–54. ACM, 1998.

[BWH+06] M. Bergou, M. Wardetzky, D. Harmon, D. Zorin, and E. Grinspun. A quadratic bending model for inextensible surfaces. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, pages 227–230. Eurographics Association, 2006.

[BWK03] D. Baraff, A. Witkin, and M. Kass. Untangling cloth. *ACM Transactions on Graphics*, 22(3):862–870, 2003.

[CC03] L. Chittaro and D. Corvaglia. 3D virtual clothing: from garment design to web3d visualization and simulation. In *Proceedings of the eighth international conference on 3D Web technology*, pages 73–84. ACM, 2003.

[CGG+03] P. Cignoni, F. Ganovelli, E. Gobbetti, F. Marton, F. Ponchio, and R. Scopigno. Planet-sized batched dynamic adaptive meshes (P-BDAM). In *Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, page 20. IEEE Computer Society, 2003.

[CGW+07] L.D. Cutler, R. Gershbein, X.C. Wang, C. Curtis, E. Maigret, L. Prasso, and P. Farson. An art-directed wrinkle system for CG character clothing and skin. *Graphical models*, 69(5-6):219–230, 2007.

[CH97] D.A. Carlson and J.K. Hodgins. Simulation levels of detail for real-time animation. In *Graphics Interface*, pages 1–8. Citeseer, 1997.

[CHCK02] Y.J. Choi, M. Hong, M.H. Choi, and M.H. Kim. Adaptive mass-spring simulation using surface wavelet. In *Proceedings of International Conference on Virtual Systems and MultiMedia*, pages 25–33, 2002.

[CHCK05] Y.J. Choi, M. Hong, M.H. Choi, and M.H. Kim. Adaptive surface-deformable model with shape-preserving spring. *Computer Animation and Virtual Worlds*, 16(1):69–83, 2005.

[CK02] K.J. Choi and H.S. Ko. Stable but responsive cloth. *ACM Transactions on Graphics (TOG)*, 21(3):604–611, 2002.

[CMT02] F. Cordier and N. Magnenat-Thalmann. Real-time animation of dressed virtual humans. In *Computer Graphics Forum*, volume 21, pages 327–335. John Wiley & Sons, 2002.

[CMT04] F. Cordier and N. Magnenat-Thalmann. A data-driven approach for real-time clothes simulation. In *Proceedings of the Computer Graphics and Applications, 12th Pacific Conference*, PG '04, pages 257–266. IEEE Computer Society, 2004.

[Cor05] J.M. Cordero. Realistic Cloth Animation. In F. Ganovelli and C. Mendoza, editors, *Proceedings of the Workshop on Virtual Reality Interaction and Physical Simulation*, 2005.

[CT81] R.L. Cook and K.E. Torrance. A reflectance model for computer graphics. In *Proceedings of the 8th annual conference on Computer graphics and interactive techniques*, pages 307–316. ACM, 1981.

[CT10] M. Chen and K. Tang. A fully geometric approach for developable cloth deformation simulation. *The Visual Computer*, 26(6):853–863, 2010.

[CTM08] S. Curtis, R. Tamstorf, and D. Manocha. Fast collision detection for deformable models using representative-triangles. In *Proceedings of the 2008 symposium on Interactive 3D graphics and games*, pages 61–69. ACM, 2008.

[DA09] N. DApuzzo. Recent advances in 3d full body scanning with applications to fashion and apparel. *Optical 3-D Measurement Techniques IX, Vienna, Austria.*, 2009.

[dASTH10] Edilson de Aguiar, Leonid Sigal, Adrien Treuille, and Jessica K. Hodgins. Stable spaces for real-time clothing. *ACM Trans. Graph.*, 29:106:1–106:9, July 2010.

[DDCB00] G. Debunne, M. Desbrun, M.P. Cani, and A. Barr. Adaptive simulation of soft bodies in real-time. In *Computer Animation 2000. Proceedings*, pages 15–20. IEEE, 2000.

[DG07] F. Durupinar and U. Gudukbay. A virtual garment design and simulation system. In *Information Visualization 2007, 11th International Conference*, pages 862–870, 2007.

[DJW+06] P. Decaudin, D. Julius, J. Wither, L. Boissieux, A. Sheffer, and M.P. Cani. Virtual garments: A fully geometric approach for clothing design. In *Computer Graphics Forum*, volume 25, pages 625–634, 2006.

[DLHS01] K. Daubert, H.P.A. Lensch, W. Heidrich, and H.P. Seidel. Efficient cloth modeling and rendering. In *12th Eurographics Workshop on Rendering*, pages 63–70, 2001.

[DM98] L. Dagum and R. Menon. Open MP: An industry-standard API for shared-memory programming. *IEEE Computational Science and Engineering*, 5(1):46–55, 1998.

[DMK+06] S. Dobbyn, R. McDonnell, L. Kavan, S. Collins, and C. OSullivan. Clothing the masses: Real-time clothed crowds with variation. *Eurographics Short Papers*, pages 103–106, 2006.

[DSB99] M. Desbrun, P. Schröder, and A. Barr. Interactive animation of structured deformable objects. In *Proceedings of the 1999 conference on Graphics Interface*, pages 1–8. Morgan Kaufmann Publishers Inc., 1999.

[EB08] E. English and R. Bridson. Animating developable surfaces using nonconforming elements. In *ACM Transactions on Graphics (TOG)*, volume 27, page 66. ACM, 2008.

[EEH00] B. Eberhardt, O. Etzmuß, and M. Hauth. *Implicit Explicit Schemes for Fast Animation with Particle Systems*. WSI, 2000.

[EEHS00] O. Etzmuss, B. Eberhardt, M. Hauth, and W. Strasser. Collision adaptive particle systems. In *Proceedings Pacific Graphics 2000*, volume 4, 2000.

[FGL03] A. Fuhrmann, C. Gross, and V. Luckas. Interactive animation of cloth including self collision detection. *Journal of WSCG*, 11(1):141–148, 2003.

[FGLW03] A. Fuhrmann, C. Grofl, V. Luckas, and A. Weber. Interaction-free dressing of virtual humans. *Computers & Graphics*, 27(1):71–82, 2003.

[FGW05] A. Fuhrmann, C. Groß, and A. Weber. Ontologies for virtual garments. In *Workshop towards Semantic Virtual Environments (SVE 2005)*, pages 101–109, 2005.

[FSG03] A. Fuhrmann, G. Sobotka, and C. Groß. Distance fields for rapid collision detection in physically based modeling. In *Proceedings of GraphiCon 2003*, pages 58–65, 2003.

[FYK10] Wei-Wen Feng, Yizhou Yu, and Byung-Uck Kim. A deformation transformer for real-time cloth animation. *ACM Trans. Graph.*, 29:108:1–108:9, July 2010.

[GBC04] M. Gillies, D. Ballin, and B.C. Csáji. Efficient clothing fitting from data. *Journal of WSCG*, 12(1):129–136, 2004.

[GFL03] C. Groß, A. Fuhrmann, and V. Luckas. Automatic pre-positioning of virtual clothing. In *Proceedings of the 19th spring conference on Computer graphics*, pages 99–108. ACM, 2003.

[GGK06] A. Greß, M. Guthe, and R. Klein. GPU-based Collision Detection for Deformable Parameterized Surfaces. volume 25, pages 497–506. Eurographics Association, Blackwell Publishing, September 2006.

[GHDS03] E. Grinspun, A.N. Hirani, M. Desbrun, and P. Schröder. Discrete shells. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 62–67, 2003.

[GHF+07] R. Goldenthal, D. Harmon, R. Fattal, M. Bercovier, and E. Grinspun. Efficient simulation of inextensible cloth. In *ACM Transactions on Graphics (TOG)*, volume 26, page 49. ACM, 2007.

[GKJ+05] N.K. Govindaraju, D. Knott, N. Jain, I. Kabul, R. Tamstorf, R. Gayle, M.C. Lin, and D. Manocha. Interactive collision detection between deformable models using chromatic decomposition. *Proceedings of ACM SIGGRAPH 2005*, 24(3):991–999, 2005.

[GLDS96] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel computing*, 22(6):789–828, 1996.

[Gri08] E. Grinspun. A discrete model of thin shells. *Discrete Differential Geometry*, pages 325–337, 2008.

[Hau05] M. Hauth. Numerical techniques for cloth simulation. In *Clothing Simulation and Animation, SIGGRAPH 2003 Course #29*, 2005.

[HBVMT99] S. Hadap, E. Bongarter, P. Volino, and N. Magnenat-Thalmann. Animating wrinkles on clothes. In *Visualization'99. Proceedings*, pages 175–523. IEEE, 1999.

[HC98] J. Hu and Y.F. Chan. Effect of fabric mechanical properties on drape. *Textile research journal*, 68(1):57–64, 1998.

[HF07] M. Hutter and A. Fuhrmann. Optimized continuous collision detection for deformable triangle meshes. *Proceedings of WSCG 2007*, pages 25–32, 2007.

[HH98] P. Howlett and WT Hewitt. Mass-spring simulation using adaptive non-active points. In *Computer Graphics Forum*, volume 17, pages 345–353, 1998.

[HM06] S.B. Huh and D.N. Metaxas. A collision resolution algorithm for clump-free fast moving cloth. *The Visual Computer*, 22(6):434–444, 2006.

[HMB01] S. Huh, D.N. Metaxas, and N.I. Badler. Collision resolutions in cloth simulation. In *Computer Animation, 2001. The Fourteenth Conference on Computer Animation. Proceedings*, pages 122–127. IEEE, 2001.

[Hop97] H. Hoppe. View-dependent refinement of progressive meshes. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, volume 97 of *SIGGRAPH '97*, pages 189–198, 1997.

[Hop98] H. Hoppe. Smooth view-dependent level-of-detail control and its application to terrain rendering. In *Visualization'98. Proceedings*, pages 35–42. IEEE, 1998.

[HPH96] D. Hutchinson, M. Preston, and T. Hewitt. Adaptive refinement for mass/spring simulations. In *7th Eurographics Workshop on Animation and Simulation*, volume 45, 1996.

[HPS11] M. Huber, S. Pabst, and W. Straßer. Wet cloth simulation. In *ACM SIGGRAPH 2011 Posters*, page 10. ACM, 2011.

[HVTG08] D. Harmon, E. Vouga, R. Tamstorf, and E. Grinspun. Robust treatment of simultaneous collisions. In *ACM Transactions on Graphics (TOG)*, volume 27, page 23. ACM, 2008.

[IH02] T. Igarashi and J.F. Hughes. Clothing manipulation. In *Proceedings of the 15th annual ACM symposium on User interface software and technology*, pages 91–100. ACM, 2002.

[JTT01] P. Jiménez, F. Thomas, and C. Torras. 3D collision detection: a survey. *Computers & Graphics*, 25(2):269–285, 2001.

[KC02] Y.M. Kang and H.G. Cho. Bilayered approximate integration for rapid and plausible animation of virtual cloth with realistic wrinkles. In *Proceedings of Computer Animation, 2002.*, pages 203–211. IEEE, 2002.

[KCCP00] Y.M. Kang, J.H. Choi, H.G. Cho, and C.J. Park. Fast and stable animation of cloth with an approximated implicit method. In *Computer Graphics International, 2000. Proceedings*, pages 247–255. IEEE, 2000.

[Ket98] Lutz Kettner. Designing a data structure for polyhedral surfaces. In *Symposium on Computational Geometry*, pages 146–154, 1998.

[KG80] S. Kawabata and N.S.K. Gakki. The standardization and analysis of hand evaluation. 1980.

[KGBS11] Ladislav Kavan, Dan Gerszewski, Adam W. Bargteil, and Peter-Pike Sloan. Physics-inspired upsampling for cloth simulation in games. *ACM Transactions on Graphics (TOG)*, 30:93:1–93:10, August 2011.

[KHI+07] S. Kockara, T. Halic, K. Iqbal, C. Bayrak, and R. Rowe. Collision detection: A survey. In *Systems, Man and Cybernetics, 2007. ISIC. IEEE International Conference on*, pages 4046–4051. IEEE, 2007.

[KJM08] J.M. Kaldor, D.L. James, and S. Marschner. Simulating knitted cloth at the yarn level. In *ACM Transactions on Graphics (TOG)*, volume 27, page 65. ACM, 2008.

[KN89] S. Kawabata and M. Niwa. Fabric performance in clothing and clothing manufacture. *Journal of the Textile Institute*, 80(1):19–50, 1989.

[Kob00] L. Kobbelt. $\sqrt{3}$-subdivision. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 103–112. ACM Press/Addison-Wesley Publishing Co., 2000.

[KSK04] N. Kondoh, S. Sasagawa, and A. Kunimatsu. Creating animations of fluids and cloth with moving characters. In *ACM SIGGRAPH 2004 Sketches*, page 136. ACM, 2004.

[KSO10] L. Kavan, P.P. Sloan, and C. O'Sullivan. Fast and efficient skinning of animated meshes. In *Computer Graphics Forum*, volume 29, pages 327–336. Wiley Online Library, 2010.

[KV08] T.Y. Kim and E. Vendrovsky. DrivenShape: a data-driven approach for shape deformation. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 49–55. Eurographics Association, 2008.

[KWH04] S. Kimmerle, M. Wacker, and C. Holzer. Multilayered wrinkle textures from strain. In *Proceedings of the Vision, Modeling, and Visualization Conference 2004 (VMV 2004)*, pages 225–232. Aka GmbH, 2004.

[LC04] C. Larboulette and M.P. Cani. Real-time dynamic wrinkles. In *Computer Graphics International, 2004. Proceedings*, pages 522–525, 2004.

[LD08] H. Lorenz and J. Döllner. Dynamic mesh refinement on GPU using geometry shaders. In *Proceedings of the 16th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, pages 97–104, 2008.

[LGPT01] R. Lario, C. Garcia, M. Prieto, and F. Tirado. Rapid parallelization of a multilevel cloth simulator using OpenMP. In *Third European Workshop on OpenMP*, 2001.

[LMT08] C. Luible and N. Magnenat-Thalmann. The simulation of cloth using accurate physical parameters. In *Computer Graphics and Imaging*. ACTA Press, 2008.

[Lov06] J. Loviscach. Wrinkling coarse meshes on the GPU. In *Computer Graphics Forum*, volume 25, pages 467–476, 2006.

[LTG05] T. Le Thanh and A. Gagalowicz. Virtual garment pre-positioning. In *Computer Analysis of Images and Patterns*, pages 837–845. Springer, 2005.

[LV05] L. Li and V. Volkov. Cloth animation with adaptively refined meshes. In *Proceedings of the Twenty-eighth Australasian conference on Computer Science-Volume 38*, pages 107–113. Australian Computer Society, Inc., Darlinghurst, Australia, 2005.

[LYO+10] Y. Lee, S. Yoon, S. Oh, D. Kim, and S. Choi. Multi-resolution cloth simulation. In *Computer Graphics Forum*, volume 29, pages 2225–2232. Wiley Online Library, 2010.

[LZY10] Y.J. Liu, D.L. Zhang, and M.M.F. Yuen. A survey on CAD methods in 3D garment design. *Computers in Industry*, 61(6):576–593, 2010.

[MC10] Matthias Müller and Nuttapong Chentanez. Wrinkle meshes. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '10, pages 85–92, Aire-la-Ville, Switzerland, Switzerland, 2010. Eurographics Association.

[MDA01] J. Montagnat, H. Delingette, and N. Ayache. A review of deformable surfaces: topology, geometry and deformation. *Image and vision computing*, 19(14):1023–1040, 2001.

[MDCO06] R. McDonnell, S. Dobbyn, S. Collins, and C. O'Sullivan. Perceptual evaluation of LOD clothing for virtual humans. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 117–126. Eurographics Association, 2006.

[ME98] M. Meissner and B. Eberhardt. The art of knitted fabrics, realistic & physically based modelling of knitted patterns. In *Computer Graphics Forum*, volume 17, pages 355–362, 1998.

[MHB06] Liang Ma, Jinlian Hu, and George Baciu. Generating seams and wrinkles for virtual clothing. In *VRCIA '06: Proceedings of the 2006 ACM international conference on Virtual reality continuum and its applications*, pages 205–211, New York, NY, USA, 2006. ACM.

[MHTG05] M. Müller, B. Heidelberger, M. Teschner, and M. Gross. Meshless deformations based on shape matching. *ACM Transactions on Graphics (TOG)*, 24(3):478, 2005.

[Min95] P.G. Minazio. Fast-fabric assurance by simple testing. *International Journal of Clothing Science and Technology*, 7(23):43–48, 1995.

[MK05] N. Metaaphanon and P. Kanongchaiyos. Real-time cloth simulation for garment cad. In *Proceedings of the 3rd international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, pages 83–89, 2005.

[MKE02] J. Mezger, S. Kimmerle, and O. Etzmuß. Improved Collision Detection and Response Techniques for Cloth Animation. In *Proceedings of 10th Pacific Conference on Computer Graphics and Applications (PG 2002)*, 2002.

[MKE03] J. Mezger, S. Kimmerle, and O. Etzmuß. Hierarchical techniques in collision detection for cloth animation. *Journal of WSCG*, 11(1), 2003.

[MKM+04] A. Mujahid, K. Kakusho, M. Minoh, Y. Nakashima, SI Mori, and S. Tomita. Simulating Realistic Force and Shape of Virtual Cloth with Adaptive Meshes and Its Parallel Implementation in OpenMP. In *Proceedings of international conference on parallel and distributed computing and networks (PDCN2004)*, pages 386–91, 2004.

[MLW+08] A.H. Mao, Y. Li, R.M. Wang, X.N. Luo, and Y.P. Guo. A computational bioengineering system for thermal functional design of textile products. *Journal of Fiber Bioengineering and Informatics*, 1(2):107–116, 2008.

[MMJ10] Y. Meng, PY Mok, and X. Jin. Interactive virtual try-on clothing design systems. *Computer-Aided Design*, 42(4):310–321, 2010.

[MO09] Y. Morimoto and K. Ono. Computer-generated tie-dyeing pattern. In *ACM SIGGRAPH ASIA 2009 Posters*, page 36. ACM, 2009.

[MTTT07] Y. Morimoto, M. Tanaka, R. Tsuruno, and K. Tomimatsu. Dyeing theory based liquid diffusion model on woven cloth. In *International Conference on Computer Graphics and Interactive Techniques: ACM SIGGRAPH 2007 posters: San Diego, California*. Association for Computing Machinery, Inc, One Astor Plaza, 1515 Broadway, New York, NY, 10036-5701, USA, 2007.

[NB04] C.N. Ngoc and S. Boivin. Nonlinear cloth simulation. *INRIA Research Report #5099, National Institute for Research in Computer Science and Control*, 2004.

[NMK+05] A. Nealen, M. Müller, R. Keiser, E. Boxerman, and M. Carlson. Physically based deformable models in computer graphics. In *Computer Graphics Forum*, volume 25, pages 809–836, 2005.

[NNR01] O. Nocent, J.M. Nourrit, and Y. Remion. Towards mechanical level of detail for knitwear simulation. *WSCG 2001 Conference Proceedings*, 1(2):3, 2001.

[OK11] O. Ozgen and M. Kallmann. Directional constraint enforcement for fast cloth simulation. *Motion in Games*, pages 424–435, 2011.

[OM01] M. Oshita and A. Makinouchi. Real-time cloth simulation with sparse particles and curved faces. In *Computer Animation, 2001. The Fourteenth Conference on Computer Animation. Proceedings*, pages 220–227. IEEE, 2001.

[ON94] M. Oren and S.K. Nayar. Generalization of Lambert's reflectance model. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 239–246. ACM, 1994.

[ONW08] S.W. Oh, J. Noh, and K. Wohn. A physically faithful multigrid method for fast cloth simulation. *Computer Animation and Virtual Worlds*, 19(3-4):479–492, 2008.

[Pho75] B.T. Phong. Illumination for computer generated pictures. *Communications of the ACM*, 18(6):311–317, 1975.

[PKST08] Simon Pabst, Sybille Krzywinski, Andrea Schenk, and Bernhard Thomaszewski. Seams and bending in cloth simulation. *Workshop in Virtual Reality Interactions and Physical Simulation VRIPHYS*, 382(1):24–41, 2008.

[PLAMT02] D. Protopsaltou, C. Luible, M. Arevalo, and N. Magnenat-Thalmann. A body and garment creation method for an internet based virtual fitting room. *Advances in Modeling, Animation and Rendering*, pages 105–122, 2002.

[Pro95] X. Provot. Deformation constraints in a mass-spring model to describe rigid cloth behaviour. In *Graphics interface*, pages 147–147. Citeseer, 1995.

[Pro97] X. Provot. Collision and self-collision handling in cloth model dedicated to design garments. In *Graphics interface*, volume 97, pages 177–189, 1997.

[PS37] F.T. Peirce and D. Sc. The geometry of cloth structure. *Journal of the Textile Institute Transactions*, 28(3):45–96, 1937.

[PUR99] R.E. Pérez-Urbiola and I. Rudomín. Multi-layer implicit garment models. In *Shape Modeling and Applications, 1999. Proceedings. Shape Modeling International'99. International Conference on*, pages 66–71. IEEE, 1999.

[RB01] M. Ruiz and B. Buxton. A model-based procedure for fitting a surface to scans of clothed people. *Proceedings of Scanning*, 2001, 2001.

[RC02] I. Rudomin and J. Castillo. Real-time clothing: geometry and physics. *Journal of Winter School of Computer Graphics*, 2002.

[RD05] G. Ryder and AM Day. Survey of real-time rendering techniques for crowds. In *Computer Graphics Forum*, volume 24, pages 203–215, 2005.

[RDMB08] C.D.G. Reis, J.M. De Martino, and H.C. Batagelo. Real-time simulation of wrinkles. In *Proceedings of WSCG*, 2008.

[RM00] I. Rudomín and M. Melon. Multi-layer garments using hybrid models. *Visual 2000 Proceedings*, pages 118–128, 2000.

[RNS06] J. Rodriguez-Navarro and A. Susin. Non structured meshes for Cloth GPU simulation using FEM. *Workshop on Virtual Reality Interaction and Physical Simulation, VRIPHYS*, 2006.

[RPC+10] Damien Rohmer, Tiberiu Popa, Marie-Paule Cani, Stefanie Hahmann, and Alla Sheffer. Animation wrinkling: augmenting coarse cloth simulations with realistic-looking wrinkles. *ACM Transactions on Graphics (TOG)*, 29:157:1–157:8, December 2010.

[Rus08] Holly Rushmeier. The perception of simulated materials. In *ACM SIGGRAPH 2008*, pages 7:1–7:12. ACM, 2008.

[SABW82] W.C. Swope, H.C. Andersen, P.H. Berens, and K.R. Wilson. A computer simulation method for the calculation of equilibrium constants for the formation of physical clusters of molecules: Application to small water clusters. *J. Chem. Phys.*, 76(1):637–650, 1982.

[SB11] H.A. Sulaiman and A. Bade. Continuous collision detection for virtual environments: A walkthrough of techniques. *electronic Journal of Computer Science and Information Technology*, 3(1), 2011.

[SKC10] A.S.M. Sayem, R. Kennon, and N. Clarke. 3D CAD systems for the clothing industry. *International Journal of Fashion Design, Technology and Education*, 3(2):45–53, 2010.

[SLD09] T.J.R Simnett, S.D. Laycock, and A.M. Day. An edge-based approach to adaptively refining a mesh for cloth deformation. In *EG UK Theory and Practice of Computer Graphics 2009*, pages 77–84, 2009.

[SLD10] T.J.R Simnett, R.G. Laycock, and A.M. Day. Simulating Real-Time Cloth with Adaptive Edge-based Meshes. *Journal of WSCG*, 18(1–3):65–72, 2010.

[SSBT08] T. Stumpp, J. Spillmann, M. Becker, and M. Teschner. A Geometric Deformation Model for Stable Cloth Simulation. *Proceedings of VRIPHYS 2008*, pages 13–14, 2008.

[SSIF07] E. Sifakis, T. Shinar, G. Irving, and R. Fedkiw. Hybrid simulation of deformable solids. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 81–90. Eurographics Association Aire-la-Ville, Switzerland, Switzerland, 2007.

[SSIF09] Andrew Selle, Jonathan Su, Geoffrey Irving, and Ronald Fedkiw. Robust high-resolution cloth using parallelism, history-based collisions,

and accurate friction. *IEEE Transactions on Visualization and Computer Graphics*, 15:339–350, 2009.

[SSK03] M. Sattler, R. Sarlette, and R. Klein. Efficient and realistic visualization of cloth. In *Proceedings of the 14th Eurographics workshop on Rendering*, pages 167–177. Eurographics Association, 2003.

[TCH04] Emmanuel Turquin, Marie-Paule Cani, and John F. Hughes. Sketching garments for virtual characters. In *Eurographics Workshop on Sketch-Based Interfaces and Modeling*, pages 175–182, 2004.

[TCYM08] M. Tang, S. Curtis, S.E. Yoon, and D. Manocha. Interactive continuous collision detection between deformable models using connectivity-based culling. In *Proceedings of the 2008 ACM symposium on Solid and physical modeling*, pages 25–36. ACM, 2008.

[THM+03] M. Teschner, B. Heidelberger, M. Müller, D. Pomeranets, and M. Gross. Optimized spatial hashing for collision detection of deformable objects. In *Proceedings of Vision, Modeling, Visualization VMV'03*, pages 47–54, 2003.

[TKH+05] M. Teschner, S. Kimmerle, B. Heidelberger, G. Zachmann, L. Raghupathi, A. Fuhrmann, M.P. Cani, F. Faure, N. Magnenat-Thalmann, W. Strasser, et al. Collision detection for deformable objects. In *Computer Graphics Forum*, volume 24, pages 61–81, 2005.

[TM06] R.F. Tobler and S. Maierhofer. A mesh data structure for rendering and subdivision. In *Proceedings of WSCG (International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision)*, pages 157–162, 2006.

[TMT09] M. Tang, D. Manocha, and R. Tong. Multi-core collision detection between deformable models. In *2009 SIAM/ACM Joint Conference on Geometric and Physical Modeling*, pages 355–360. ACM, 2009.

[TMT10] M. Tang, D. Manocha, and R. Tong. MCCD: Multi-core collision detection between deformable models using front-based decomposition. *Graphical Models*, 72(2):7–23, 2010.

[TPS09] B. Thomaszewski, S. Pabst, and W. Straßer. Continuum-based Strain Limiting. In *Computer Graphics Forum*, volume 28, pages 569–576. John Wiley & Sons, 2009.

[TW06] B. Thomaszewski and M. Wacker. Bending models for thin flexible objects. In *WSCG short communication proceedings*, 2006.

[TWS06] B. Thomaszewski, M. Wacker, and W. Straßer. A consistent bending model for cloth simulation with corotational subdivision finite elements. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 107–116. Eurographics Association Aire-la-Ville, Switzerland, Switzerland, 2006.

[Vas00] T.I. Vassilev. Dressing virtual people. *Proceedings of Systemics, Cybernetics and Informatics*, 2000, 2000.

[VB05] J. Villard and H. Borouchaki. Adaptive meshing for cloth animation. *Engineering with Computers*, 20(4):333–341, 2005.

[VCMT95] P. Volino, M. Courchesne, and N. Magnenat Thalmann. Versatile and efficient techniques for simulating cloth and other deformable objects. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 137–144. ACM, 1995.

[Ver67] L. Verlet. Computer "experiments" on classical fluids. I. Thermodynamical properties of Lennard-Jones molecules. *Physical Review*, 159(1):98, 1967.

[VL03] V. Volkov and L. Li. Real-time refinement and simplification of adaptive triangular meshes. *Visualization, 2003. VIS 2003. IEEE*, pages 155–162, 2003.

[VMT05] P. Volino and N. Magnenat-Thalmann. Implicit midpoint integration and adaptive damping for efficient cloth simulation. *Computer Animation and Virtual Worlds*, 16(3-4):163–175, 2005.

[VMT06] P. Volino and N. Magnenat-Thalmann. Simple linear bending stiffness in particle systems. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 101–105. Eurographics Association, 2006.

[VMTF09] P. Volino, N. Magnenat-Thalmann, and F. Faure. A simple approach to nonlinear tensile stiffness for accurate cloth simulation. *ACM Transactions on Graphics (TOG)*, 28(4):105, 2009.

[VPBM01] A. Vlachos, J. Peters, C. Boyd, and J.L. Mitchell. Curved PN triangles. In *Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 159–166. ACM, 2001.

[VS00] T. Vassilev and B. Spanlang. Efficient cloth model for dressing animated virtual people. In *Proc of Learning to Behave Workshop. Enschede the Netherlands:[sn]*, pages 89–100, 2000.

[VSC01a] T. Vassilev, B. Spanlang, and Y. Chrysanthou. Efficient cloth model and collisions detection for dressing virtual people. In *ACM/EG Games Technology Conference*, pages 1–10, 2001.

[VSC01b] T. Vassilev, B. Spanlang, and Y. Chrysanthou. Fast cloth animation on walking avatars. In *Computer Graphics Forum*, volume 20, pages 260–267. Wiley Online Library, 2001.

[VT94] P. Volino and N.M. Thalmann. Efficient self-collision detection on smoothly discretized surface animations using geometrical shape regularity. In *Computer Graphics Forum*, volume 13, pages 155–166. Wiley Online Library, 1994.

[VT97] P. Volino and N.M. Thalmann. Developing simulation techniques for an interactive clothing system. In *Virtual Systems and MultiMedia, 1997. VSMM'97. Proceedings., International Conference on*, pages 109–118. IEEE, 1997.

[VT00] P. Volino and N.M. Thalmann. Accurate collision response on polygonal meshes. In *Computer Animation 2000. Proceedings*, pages 154–163. IEEE, 2000.

[Wan02] R. Wang. Adaptive cloth simulation. Technical report, School of Computer Science, Carnegie Mellon University, 2002.

[WAY03] Z. Wu, CK Au, and M. Yuen. Mechanical properties of fabric materials for draping simulation. *International Journal of Clothing Science and Technology*, 15(1):56–68, 2003.

[WB05] W.S.K. Wong and G. Baciu. Dynamic interaction between deformable surfaces and nonsmooth objects. *Visualization and Computer Graphics, IEEE Transactions on*, 11(3):329–340, 2005.

[WBG+08] M. Wardetzky, M. Bergou, A. Garg, D. Harmon, D. Zorin, and E. Grinspun. Simple and efficient implementation of discrete plates and shells. In *ACM SIGGRAPH ASIA 2008 courses*, page 13. ACM, 2008.

[Wei86] J. Weil. The synthesis of cloth objects. In *ACM Siggraph Computer Graphics*, volume 20, pages 49–54. ACM, 1986.

[WHRO10] H. Wang, F. Hecht, R. Ramamoorthi, and J. O'Brien. Example-based wrinkle synthesis for clothing animation. *ACM Transactions on Graphics (TOG)*, 29(4):107, 2010.

[WKK+05] M. Wacker, M. Keckeisen, S. Kimmerle, W. Straßer, V. Luckas, C. Groß, A. Fuhrmann, M. Sattler, R. Sarlette, and R. Klein. Simulation and visualisation of virtual textiles for virtual try-on. *Special Issue of Research*

Journal of Textile and Apparel: Virtual Clothing Technology and Applications, 9(1):37–47, 2005.

[WKP11] C.M. Wittenbrink, E. Kilgariff, and A. Prabhu. Fermi GF100 GPU architecture. *Micro, IEEE*, 31(2):50–59, 2011.

[XESV97] J.C. Xia, J. El-Sana, and A. Varshney. Adaptive real-time level-of-detail based rendering for polygonal models. *Visualization and Computer Graphics, IEEE Transactions on*, 3(2):171–183, 1997.

[Yus02] G.C.L. Yushu. A survey of collision detection. *Computer Engineering and Applications*, page 05, 2002.

[ZH07] N. Zink and A. Hardy. Cloth simulation and collision detection using geometry images. In *Proceedings of the 5th international conference on Computer graphics, virtual reality, visualisation and interaction in Africa*, pages 187–195. ACM, 2007.

[ZY00] D. Zhang and M.M.F. Yuen. Collision detection for clothed human animation. In *Computer Graphics and Applications, 2000. Proceedings. The Eighth Pacific Conference on*, pages 328–337. IEEE, 2000.

[ZY01] D. Zhang and M.M.F. Yuen. Cloth simulation using multilevel meshes. *Computers & Graphics*, 25(3):383–389, 2001.