TEARABLE CLOTH

A Thesis

by

KURT THOMPSON PHILLIPS

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

December 2008

Major Subject: Visualization Sciences

TEARABLE CLOTH

A Thesis

by

KURT THOMPSON PHILLIPS

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Approved by:

Chair of Committee,	Donald House
Committee Members,	Frederic Parke
	Gerald Vinson
Head of Department,	Tim McLaughlin

December 2008

Major Subject: Visualization Sciences

ABSTRACT

Tearable Cloth.

(December 2008)

Kurt Thompson Phillips, B.E.D., Texas A&M University

Chair of Advisory Committee: Dr. Donald House

This document proposes modifications to an established cloth simulation algorithm to allow for stretch deformation and tearing of simulated cloth in computer-generated imagery. Previous research is presented, followed by the development of a cloth simulation system with the addition of tearing behavior. Several results are given that show off individual features and behaviors that this thesis models.

DEDICATION

I dedicate this thesis to my family and friends, without whom I would not be achieving my life goals.

ACKNOWLEDGEMENTS

I would like to thank my committee chair, Dr. Donald House, and my committee members, Dr. Frederic Parke, and Dr. Gerald Vinson, for their guidance and support throughout the course of this research. Thanks to Dr. House for his guidance and patience throughout my graduate education. Thanks to Dr. Parke for his leadership. And, thanks to Dr. Vinson for his undivided excitement in my work.

I must give many thanks to other teachers and staff that greatly influenced my undergraduate and graduate education at Texas A&M. Rodney Hill encouraged me to explore my creativity, and my work since the early days of ENDS 101 has greatly benefited from his teachings. Many thanks to Ray Mullican for making me think outside the box and giving me the freedom to pursue my own ideas in his courses. Thanks to Mary Saslow, Karen Hillier, and Carol Lafayette for helping me develop my visual communication and story telling skills. Thanks to Bill Jenks, Glen Vigus, and Kevin Glueck for their technical support and encouragement throughout my graduate education. Last, but not least, I would like to thank Margaret Lomas for keeping order in the Vizlab and helping me with any situation I got myself into.

Thanks also go to my friends near and far for making my time at Texas A&M University a great experience. Thanks to Brooke Beane for her constantly energetic presence, always encouraging me to do great things. When it was time for a break, I could always count on Josh Rowe and Tammy Cuellar to loosen the stresses of the Vizlab and enjoy it. Thanks to Karthik Swaminathan for helping me troubleshoot almost

any problem, and for good company as we devoured Freebirds burritos. Thanks to Mayank “m-dawg” Singh, my other burrito buddy, for the GUI challenges and interesting tidbits on programming. Thanks to Chris Root and Alex Timchenko for their expertise and helping me with most of my programming issues. And thanks to Katie Van Maanen for her silliness, friendship, and outright craziness that helped make my time in the Vizlab memorable and enjoyable.

I would also like to thank all of the Aggies at Pixar Animation Studios for helping make my internship there extremely informative, fun, and memorable. Many thanks to my friends and colleagues at Dreamworks Animation for their encouragement in finishing my thesis.

Finally, thanks to my mother, Susie, father, Harvey, and brother, Dan, for their love and unending encouragement and excitement with everything I’ve ever done in my life. Thanks to Nana and Frank, and Mom and Papa, (my grandparents) for their love and encouragement as well, and allowing me to pursue an education without the financial burdens of the real world. And thanks to Ryan Bills and Justin Bauer, my brothers from other mothers, for their support over the years.

TABLE OF CONTENTS

CHAPTER Page

LIST OF FIGURES

FIGURE Page

FIGURE Page

CHAPTER I

INTRODUCTION

Cloth animation is considered by many to be one of the most difficult things to produce with computer-generated imagery. People interact with cloth on a day-to-day basis. It's on their bodies, covering tabletops, and part of any great magic show, so it is intuitively known whether or not its behavior is correct. Unlike a rigid object, cloth undergoes extreme deformations and is ever changing, making it difficult to model. At the same time, unlike soft elastic substances such as rubber, cloth has high tensile rigidity, making its dynamic equations highly stiff and difficult to simulate. Also, cloth is like any other object; it can be broken. Cloth is typically made by weaving strands of material, which are then held in place by friction alone. When put under enough stress, the weave can become extremely distorted, causing substantial permanent deformations or even tearing.

Tearing cloth is by no means a novel idea. Terzopolous and Fleischer [23] used principles of inelastic deformation to tear cloth-like nets and rubber sheets. Unfortunately, there is no recent published work specifically on tearing cloth. A wealth of work is present on fracturing solids, such as O'Brien and Hodgins' [14, 15] work on brittle and ductile fracture. Nonetheless, examples of tearing cloth have surfaced in recent years. Industrial Light and Magic ripped cloth in 2003's *Hulk* [8]. A year later,

---

This thesis follows the style of *IEEE Transactions on Visualization and Computer Graphics.*

ILM used similar principles in *Van Helsing* [24] to tear the skin off of a human character transforming into a werewolf. Rhythm & Hues Studios ripped cloth in a 2008 Hulk sequel, *The Incredible Hulk* [9]. A publicly available implementation of tearing cloth only recently surfaced in 2007 with Autodesk's nCloth system featured in Maya 8.5 [1].

This thesis focuses on the extension of an established cloth simulation technique that will mimic these cloth characteristics to create interesting and realistic tearing behavior. First, a history of cloth simulation research is presented. A complete history of cloth simulation is beyond the scope of this thesis, but the ideas inherent to the presented publications relate directly to my own cloth simulation system. Next, a brief history of fracture research is presented. Then, the basics of particle simulation are discussed in order to build a foundation for the next section, developing a cloth simulator. Several important concepts are introduced here to make the particle system behave like cloth. With a simple cloth simulator established, tearing behavior and its implementation in the simulator is explained. Special attention is given to the problems of measuring stress in a simulating mesh, geometrically separating the mesh, allowing for permanent deformation of the mesh, and how that behavior can be artistically controlled. Finally, rendering methods are presented which were used to produce the results of this research.

CHAPTER II

BACKGROUND

Tearing cloth requires knowledge of how cloth simulation surfaced and became practical in the computer graphics community. The investigation of fracturing solids provides some clues about how such principles could be used to tear cloth.

A. Cloth Simulation

Before computer simulation methods were considered, cloth objects were often modeled as rigid objects with textures mapped onto the surface. Convincing cloth movement was out of the question due to its incredibly complex nature. Modeling this behavior was of interest to the emerging computer graphics community.

1. The Synthesis of Cloth Objects

In *The Synthesis of Cloth Objects* [25], Jerry Weil modeled a special case of a rectangular piece of cloth hanging from to constrained points. The cloth was modeled as a two-dimensional grid of three-dimensional points. The resulting shape of the draping cloth was computed using a two-stage process.

The first stage used catenary curves (Fig. 1) to approximate the surface of the draped cloth. Catenary curves were used because they mimic the natural drape of a thread produced by gravity. These curves were computed between the corners of the cloth, producing a convex hull. The grid points of the cloth that lay along the curve were

easily positioned. This divided the grid points into distinct regions. These regions were subdivided by more catenary curves, and the grid points that lay on those curves were computed. This process was repeated until all grid points of the cloth were positioned.

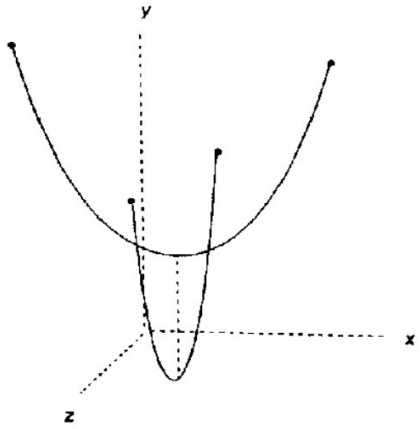
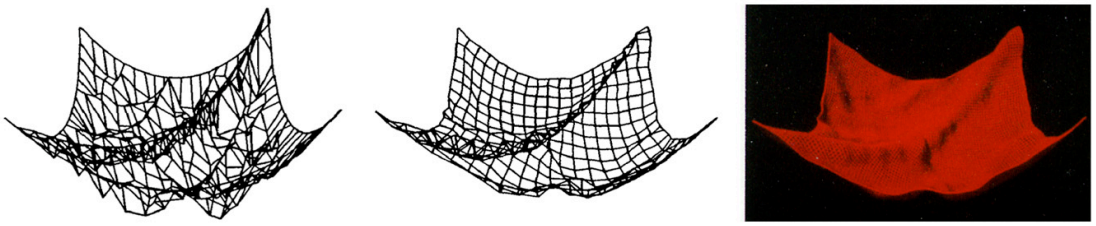


Fig. 1: Catenary curves from Weil [25].

The resulting shape was fairly jagged, requiring the next stage, relaxation (Fig. 2). In this stage, the distance between grid points was minimized, but kept within a given tolerance. Out-of-plane stiffness in the cloth could also be modeled at this stage as a constraint on the angle between grid points.



(a) (b) (c)

Fig. 2: Two stage process and final rendered image from Weil [25].

(a) surface approximation, (b) after 6 iterations of relaxation, (c) ray traced image

Weil's work was a step forward in cloth animation, but his methods provided only a static drape of cloth. Animation was not practical due to the unpredictable subdivision of the geometry generated by the catenary curves.

2. Elastically Deformable Models

By 1987, animating deformable objects was a possibility, but only if the objects were deformed by a predefined process. A user with exceptional skills at mimicking physical laws of motion was required to produce a convincing animation of a deformable object. But not even the most experienced animator would be able to animate a deformable model such as cloth by hand in an acceptable amount of time. The process would be tedious, problematic, and prone to human error.

Demetri Terzopoulos, John Platt, Alan Barr, and Kurt Fleischer introduced one of the first systems to dynamically simulate a deformable model in 1987. They presented a novel approach to use the laws of physics to govern the ever-changing shape of an object. In *Elastically Deformable Models* [22], Terzopolous et al. developed a deformable model for curves, surfaces, and solids based on the principles of elasticity theory [11]. They could simulate physical properties of an object, such as tension and rigidity, as well as the forces that cause the object to deform, such as gravity, internal forces, viscous forces, and forces resulting from collisions with other objects.

Terzopolous et al. used a continuum model for the deformable object. Continuum models are defined by continuous functions, providing the ability to represent smooth surfaces accurately. The authors derived equations for potential energies of deformation,

which were then used to define individual forces. For internal elastic forces, they proposed that the strain energy of an elastic body is the magnitude of the difference between the deformed body and its natural undeformed state (Fig. 3).

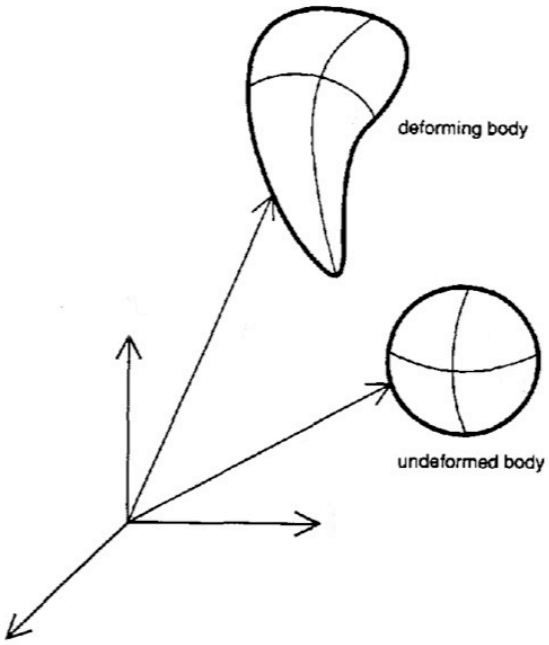


Fig. 3: Elastic formulation of deformable models from Terzopolous et al. [22].

Given a point on the surface, the resulting internal elastic force is similar to that of a standard Hookean spring. They defined other forces, including gravity and a force caused by the surface moving through a viscous fluid. Collisions were also possible by creating a force proportional to the amount of penetration detected, and in the normal direction of the penetrated surface. They also discussed the possibility of the deformable object colliding with itself by defining a self-repulsive collision force that surrounds the entire object.

All of the established forces were continuous in the material coordinates of the surface. To simulate the deformable model, the surface had to be discretized (Fig. 4) into a given number of evenly spaced nodes to support a finite difference approximation. The force equations could then be discretized at these nodes, resulting in a solvable system of linear differential equations.

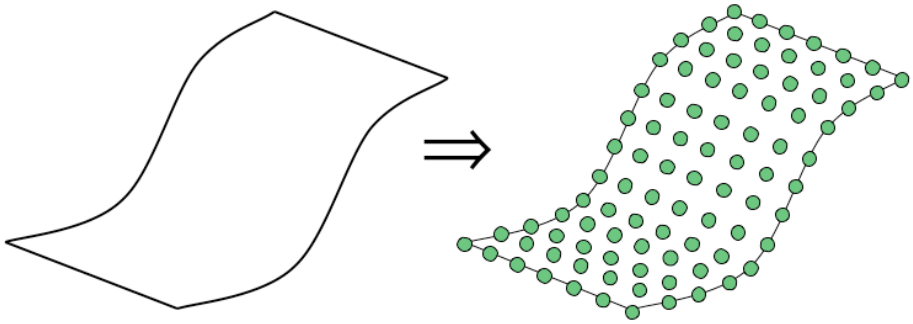


Fig. 4: Discretization.

Terzopoulos et al. integrated these force equations over time to simulate the dynamics of the deformable model. They divided a given time range into equal steps, and the integration method solved the system of linear differential equations for an individual time step. Once new positions were calculated from the integration, the continuous surface was regenerated using the discretized nodes as control points, and the process continued for each time step.

The results obtained by the authors demonstrated the ability of their deformable model to represent a wide range of materials. They simulated a thin plate, with high resistance to stretching and bending. The same model was simulated without resistance to bending and produced one of the first convincing simulations of a cloth-like object.

Expanding on this, they simulated a flag blowing in the wind, and a rug falling on rigid objects (Fig. 5).

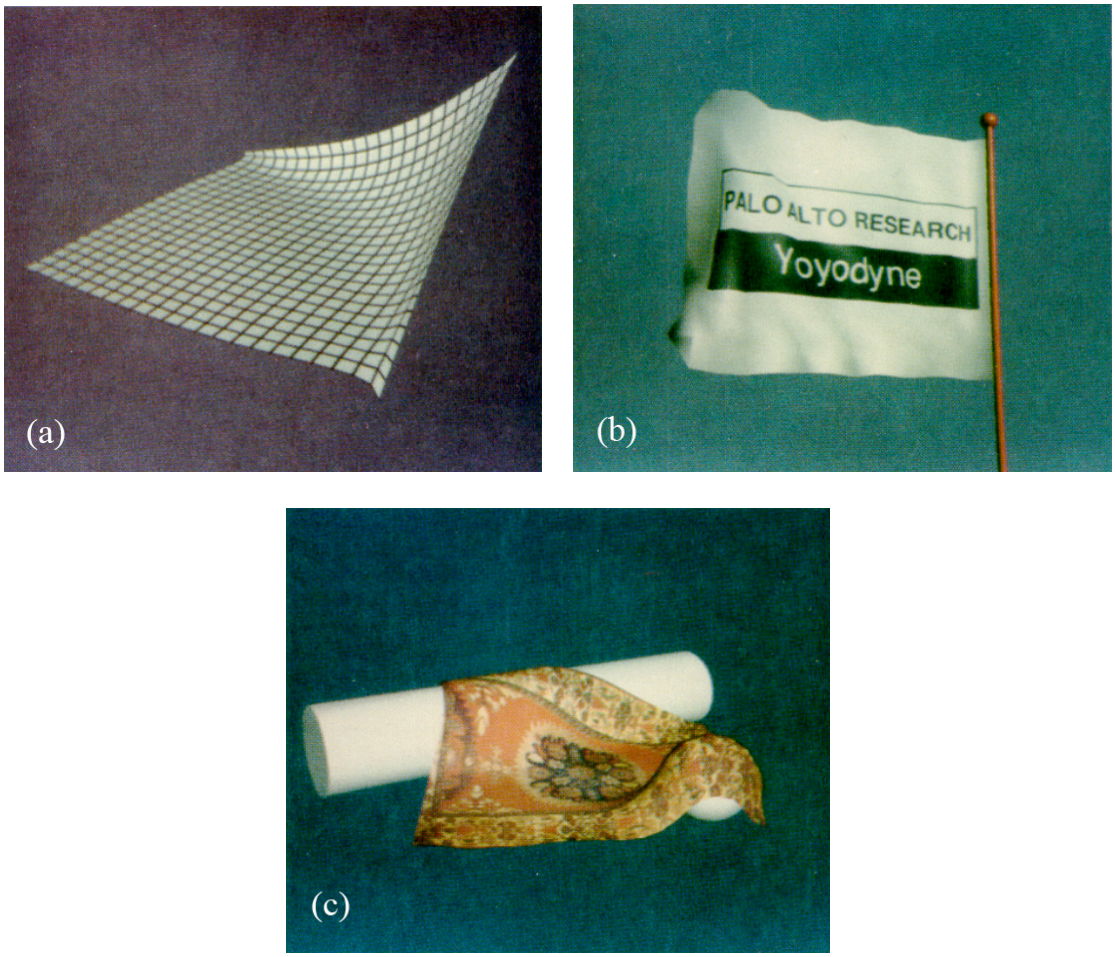


Fig. 5: Results from Terzopolous et al. [22].

(a) lifting an elastic surface, (b) flag waving in wind, (c) rug falling over obstacles.

3. Predicting the Drape of Woven Cloth Using Interactive Particles

In 1994, David Breen, Donald House, and Michael Wozny presented a novel idea in *Predicting the Drape of Woven Cloth Using Interactive Particles* [5]. In this paper, they used an interconnected network of particles to represent their cloth. This was an interesting approach because previous methods relied on a continuum approach similar to Terzopolous and Fleischer's work [22, 23]. The idea was that cloth is not a continuous material held together by molecular bonds; it is instead a complex mechanical system held together by friction where warp and weft threads cross. Collections of these crossings were represented as particles (Fig. 6). The assumption made was that by modeling this low-level structure, the correct large-scale behavior would emerge.

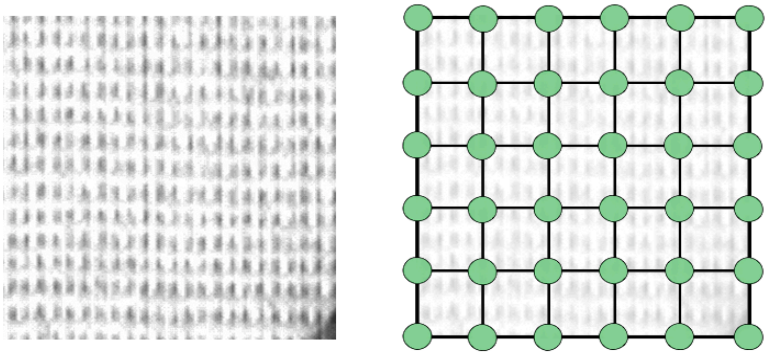


Fig. 6: Relationship of real cloth to interacting particles.

(a) warp and weft of fabric from Breen et al. [5], (b) basic particle layout.

The parameters for this physically based model were derived from fabric sample test results from the Kawabata Fabric Testing Device [10]. Real-world samples of

different types of cloth were tested with the device to measure cloth behavior when undergoing stress. Energy functions were then derived using approximations of the resulting experiential data. Four basic mechanical interactions were measured: stretching, compression, shearing, and out-of-plane bending.

The final drape of the cloth was computed using a three-phase process. In the first phase, the dynamics of each particle in the cloth model were calculated. These calculations took into account the influence of gravity in a viscous medium, as well as self and world collisions. The second phase used energy minimization techniques to enforce inter-particle constraints. The third phase adjusted particle velocities to account for particles that were moved in the second phase.

The authors were able to produce images of different types of draped cloth, including cotton, wool, and cotton/polyester hybrid fabrics. They compared these results to real-world photographs, and found that their method models cloth remarkably well (Fig. 7).

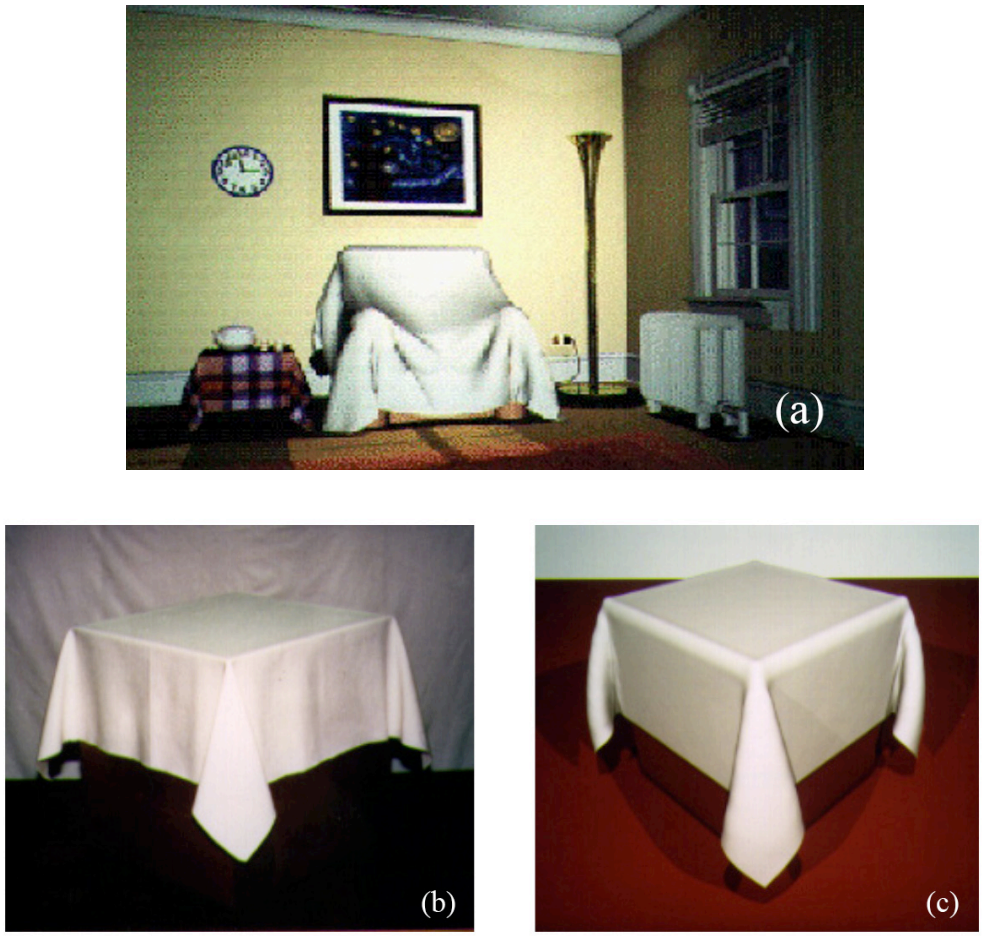


Fig. 7: Results from Breen et al. [5].

(a) draping cloth objects, (b) real vs. (c) virtual cloth drape.

4. Deformation Constraints in a Mass-Spring Model to Describe Rigid Cloth Behavior

As particle based systems became more prevalent, many mass-spring cloth animation techniques suffered from extreme spring elongation in areas of high stress, such as the corners of a hanging sheet. One way to alleviate this problem was to increase the stiffness of the cloth springs, but this often led to instability and required a smaller integration timestep, resulting in longer simulation times. Xavier Provot tackled this

problem of super-elastic cloth in 1995 with *Deformation Constraints in a Mass-Spring Model to Describe Rigid Cloth Behavior* [17]. He used the idea of inverse dynamics to provide a solution to this problem.

Inverse dynamics were used at the time to constrain cloth points. Many methods integrated the velocities of each cloth vertex, and then simply moved any constrained vertices to the desired position, essentially performing a dynamic inverse operation. In the case of the hanging sheet, these constrained points were fixed and had zero velocity. Provot extended this concept to handle super elongated springs. After each cloth particle was integrated, the deformation rates of the connecting springs were computed. If that rate exceeded a certain threshold, a dynamic inverse procedure was applied to any violating springs to shorten them. Assuming the spring's direction was valid, only the endpoints were moved. In the case of a free falling spring (no constrained endpoints), each endpoint was moved equally towards the center of the spring until the spring's length was within the threshold. If one of the endpoints was constrained, only the opposite endpoint was moved to achieve the threshold.

While it may seem problematic to randomly move vertices in a dynamic cloth simulation, the results were more than satisfactory. Provot's results showed that the problem of super-elongation could be avoided using this method, without decreasing the timestep of the simulation (Fig. 8).

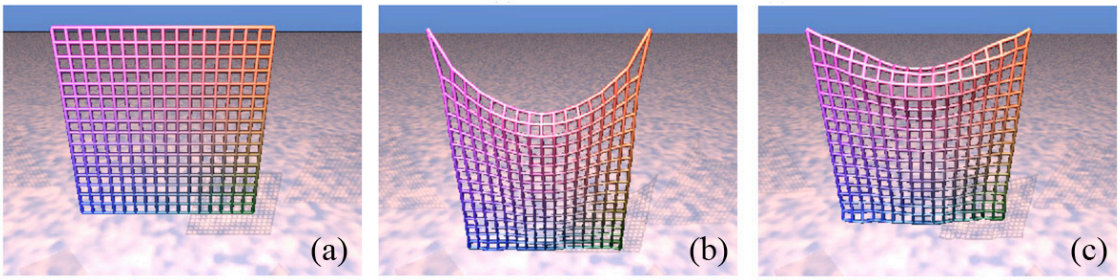


Fig. 8: Super-elongation post process from Provot [17].

(a) before simulation, (b) after simulation, (c) after super-elongation post process.

5. Large Steps in Cloth Simulation

In 1998, complex cloth simulation became much more practical with David Baraff and Andrew Witkin's paper, *Large Steps in Cloth Simulation* [2]. Their basic cloth model was similar to many, comprised of mass points connected by mass-less springs. Because cloth greatly resists stretching, the equations resulting from the forces caused by these stretch-resistant springs tended to be very stiff, which in turn required a small timestep to maintain numerical stability. Many cloth simulators at the time relied on an explicit integration method, which tended to be very slow for complex high-resolution cloth meshes. Baraff and Witkin solved this problem by using an implicit integration method.

Rather than calculating the next state of the cloth based on current conditions, the implicit method solved for the next state based on the conditions at the end of the timestep. A large linear system was constructed based on forces at each cloth vertex and how they were changing with respect to the vertex's position and velocity. The resulting system could be quite large (1200x1200 for a 400 node cloth model), which was expensive to solve due to the required matrix inversion. But luckily, it was fairly sparse,

with many elements set to zero. This characteristic allowed the system to be solved using the conjugate gradient method, which is a specialized procedure that uses an iterative process to solve sparse linear systems until a given tolerance of error is reached.

The equations that made up this linear system were all derived from a simple condition equation. The condition equation rose from the fact that all forces internal to the cloth tend towards a minimum. Equations for stretch forces were described as the desire for cloth particles to resist divergence from their parametric (u, v) positions in the cloth. A similar condition was created for shearing forces, and out-of-plane bending forces relied on the desire of the angle between two cloth triangles to be zero. External forces such as gravity and wind were derived in a more classical manner.

Baraff and Witkin also introduced novel techniques for constraining cloth particles. They proposed that the concept of mass can be expanded from a single scalar value to a 3x3 matrix. By modifying this matrix, cloth vertices could be constrained to any two-dimensional plane, a single axis of motion, or a single point in space. They also derived a way to arbitrarily move cloth vertices without introducing unwanted artifacts or oscillations in the cloth, which was useful for handling all kinds of collisions.

This work provided substantial gains in the speed and performance of cloth simulations. Baraff and Witkin provided animations and statistics for a wide variety of scenes, ranging from a simple tablecloth to a moving, clothed, multi-garmented character (Fig. 9).

Fig. 9: Results from Baraff and Witkin [2].

6. Other Recent Cloth Publications

Since Baraff and Witkin's groundbreaking paper, a wealth of cloth simulation publications have emerged, each dealing with a specific problem. From these papers, it seems that particle-based simulation has been solidly established as the most effective and efficient method for cloth simulation.

In 2002, Kwang-Jin Choi and Hyeong-Seok Ko published *Stable but Responsive Cloth* [7], a paper that gave special attention to the problem of unstable buckling. Buckling is the primary cause of wrinkles in cloth, and is caused by forces in the cloth plane acting against each other. Initially, cloth resists this compression, but the tiniest of forces in an out of plane direction can cause the cloth to buckle instantly (Fig. 10). This

rapid change in resistance can easily lead to numerical instability. Baraff and Witkin used viscous air drag to alleviate this problem, but Choi and Ko presented a method to keep buckling under control without using these fictitious forces. This allowed them to simulate a much wider range of fabrics (Fig. 11).

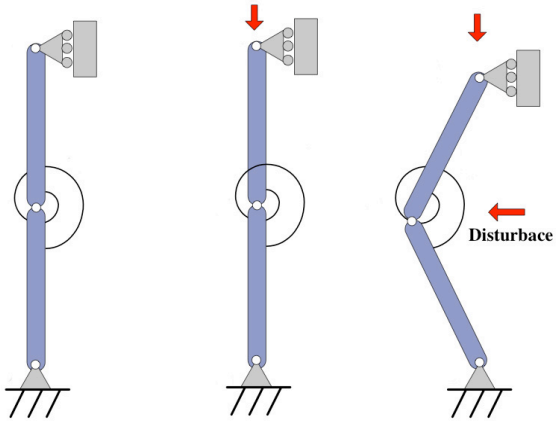


Fig. 10: Column buckling from Choi and Ko [7].

Fig. 11: Results from Choi and Ko [7].

Also in 2002, Robert Bridson, Ronald Fedkiw, and John Anderson published *Robust Treatment of Collisions, Contact and Friction for Cloth Animation* [6]. This paper tackled the complex issue of how to efficiently handle collisions in cloth simulation. To simulate believable clothing, a fairly high-resolution mesh was required, possibly consisting of several thousands of particles. Handling collisions between these particles and other objects (as well as the cloth itself) proved to be extremely computationally expensive. The main bottleneck of collision handling occurred during the collision detection stage. They proposed that many cloth-cloth collisions could be simply avoided by activating repulsive forces, such that the cloth never intersected itself; it mostly glided by itself. Collisions that did occur were detected by defining a bounding box hierarchy of the cloth's geometry. This allowed the collision detector to eliminate large areas of the cloth and put most of its effort into areas of concern. The resulting gains in speed allowed for highly detailed and realistic cloth to be simulated in a reasonable amount of time (Fig. 12).

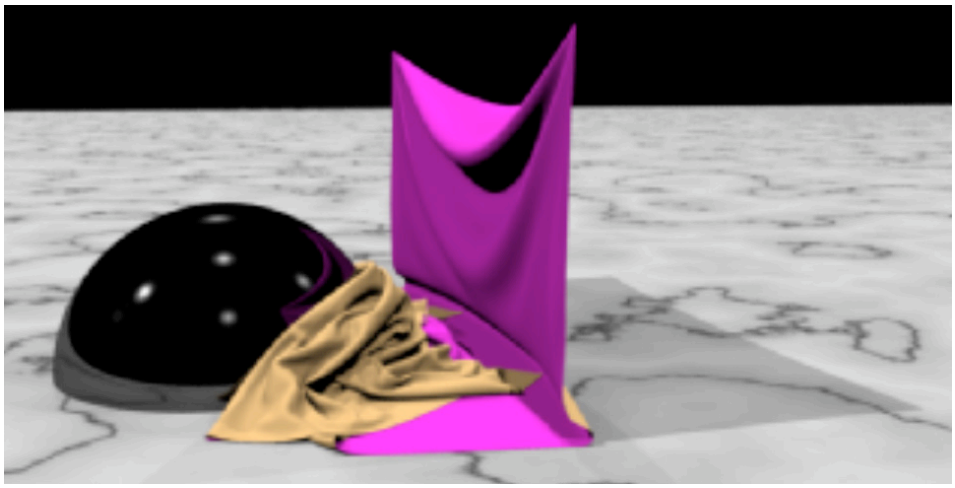


Fig. 12: Results from Bridson et al. [6].

B. Fracture

Tearing cloth is a specific problem that has not received much attention in distinguished publications. But, there is a wealth of information on fracturing solids, which provides a good foundation for the development of a cloth tearing algorithm.

1. Modeling Inelastic Deformation: Viscoelasticity, Plasticity

In 1988, Terzopoulos and Fleischer published *Modeling Inelastic Deformation: Viscoelasticity, Plasticity, Fracture* [23], a follow-up paper to their previous work on elastic deformation. In this paper, they explored inelastic behaviors of materials and how they could be implemented using their continuum approach. Inelastic behavior can be described as any deformation that does not obey pure Hookean laws. That is, the phenomena in materials that causes them to not return to their original state after being deformed. This occurs in real-life materials when they are put under enough stress to permanently change on a microscopic level.

Viscoelasticity describes the combined characteristics of elasticity and viscosity in a material. Elastic deformation occurs when a material returns to its original shape upon the removal of external forces. This happens because the material stores potential energy during deformation and completely releases it as the material returns to its original shape. Viscosity dampens this behavior, yet does not prevent it. Plasticity describes the phenomenon where a material is deformed to the point where the deformation is permanent; i.e. the elastic capabilities of the material are lost. Further deformation can cause a material to fracture.

Terzopoulos and Fleischer described these behaviors as assemblies of mechanical units, with the spring as the elastic unit, the dashpot as the viscous unit, and the slip joint as the plastic unit (Fig. 13).

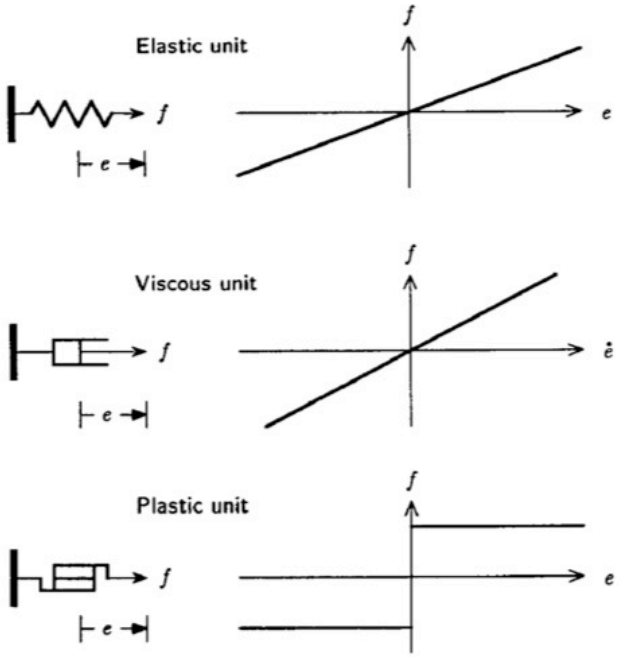


Fig. 13: Mechanical units with associated energy to force graphs from Terzopolous et al. [23].

The behaviors describing these units were built into their continuum model by deriving the necessary differential equations. But to handle the plastic behavior, a novel representation of deformable models was required.

Their primary elastic formulation described the total deformation of an object as geometric changes with respect to its rest shape. For elastic deformation, this resulted in restorative forces. For their plastic formulation, they provided a reference shape that

could deform as well. Their plastic units acted to deform the original reference shape (Fig. 14).

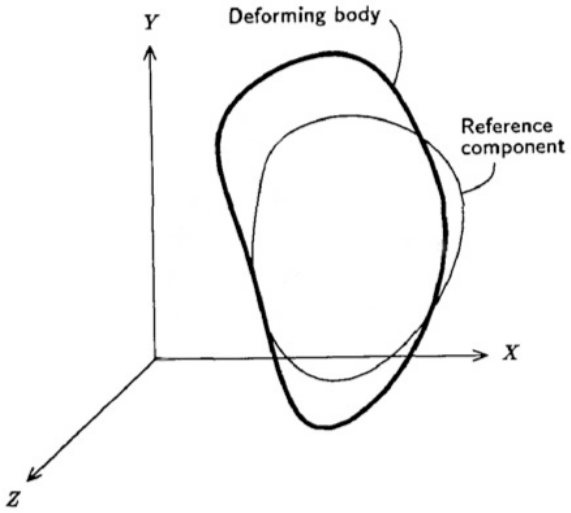


Fig. 14: Plastic formulation of deformable models from Terzopolous et al. [23].

Without a changing rest shape, the hybrid model acts the same as the primary model. Having the ability to modify the rest shape is the crux of modeling inelastic deformation. The model obtained this behavior by using applied forces to modify the rest shape and the material's properties.

The results obtained by Terzopoulos and Fleischer featured some of the most complex material behavior seen in computer graphics to date. They envisioned the ultimate virtual sculpting system, where a user could carve "computer plastecine" and haptically apply simulated forces to interactively shape the virtual material. To demonstrate fracture, they simulated a net falling on a sphere, as well as the tearing of a planar surface (Fig. 15).

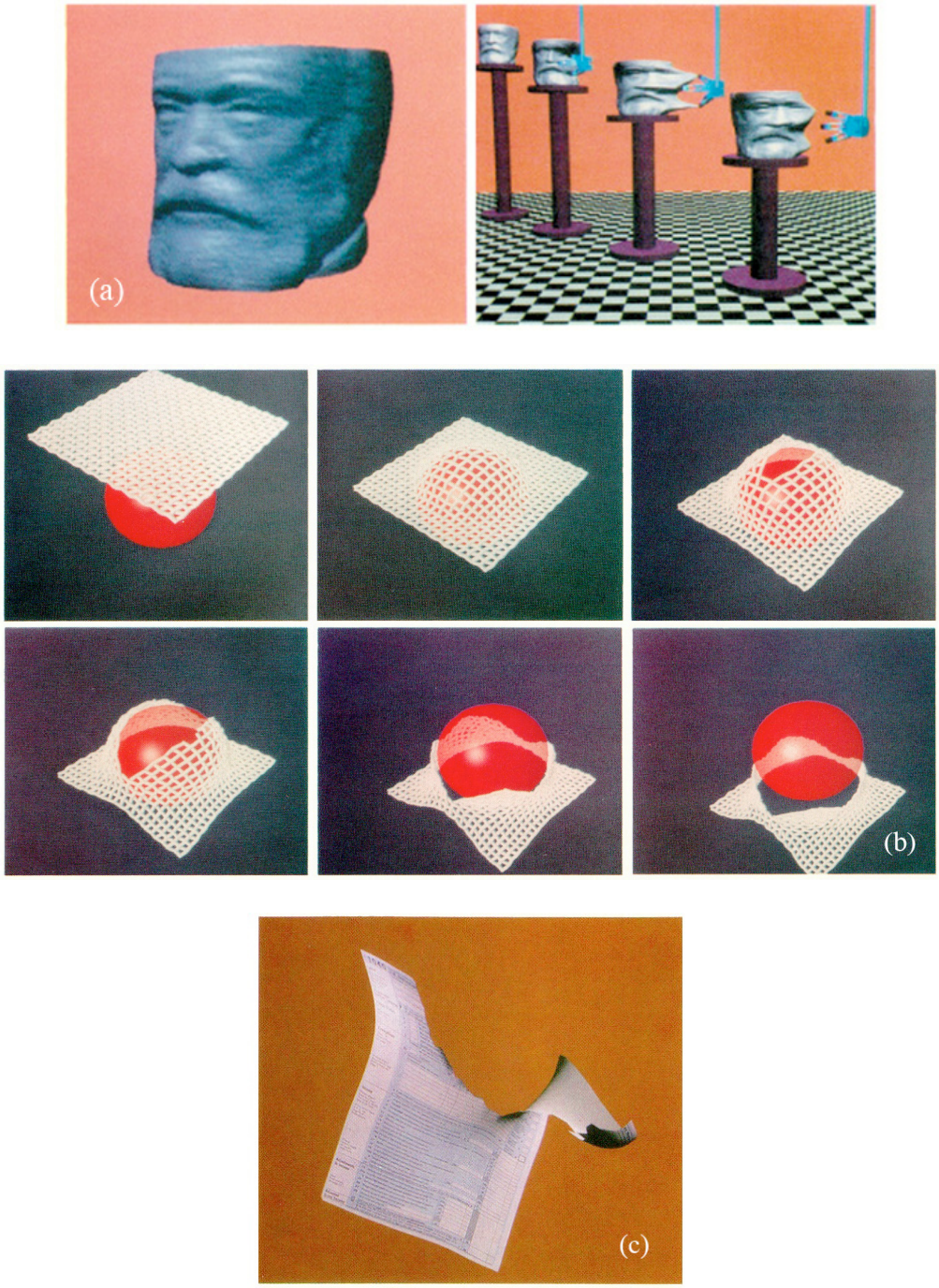


Fig. 15: Results from Terzopolous et al. [23].

(a) Modeling with computer plastecine, (b) a falling net, (c) tearing paper.

2. Animation of Fracture by Physical Modeling

In 1991, Alan Norton, Greg Turk, Bob Bacon, John Gerth, and Paula Sweeney published *Animation of Fracture by Physical Modeling* [13]. They combined several concepts to achieve a physically based fracturing model.

For their geometric model, Norton et al. developed a system of modules that could be combined to form larger structures. The individual module was a cube lattice, constructed of eight nodes and 12 edges that represented the bonds between the lattice's nodes (Fig. 16). Additional edges could also be added within the cube to provide extra structural support. Each node contained a mass quantity, a 3D vector for position, and another 3D vector for velocity. The bonds between each node had an associated force function that represented the attraction between the end nodes. What emerged from this setup was a particle based mass-spring system.

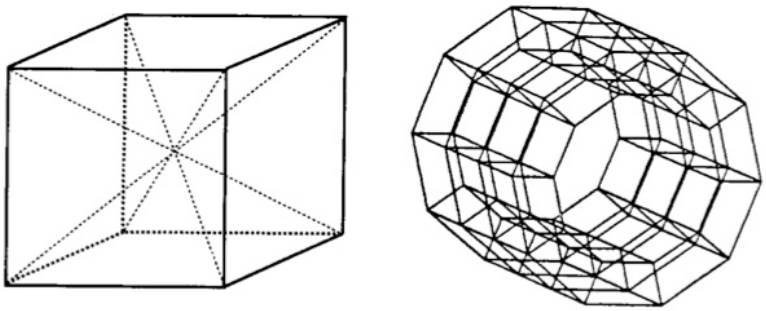


Fig. 16: Cube Lattice from Norton et al. [13].

The dynamic motion of this system was integrated with a simple forward Euler algorithm. Several other forces were taken into account. Gravity was treated as a simple constant downward force. Friction was modeled as a linear force proportional to the

amount of interpenetration, and in the direction opposing lateral motion. External collisions were treated with a repelling force based on the penetrating node's velocity and amount of penetration. Internal collisions (such as the object and a disconnected piece of itself) were handled with a penalty method, where temporary springs connecting the colliding node and the penetrated face repelled each other. Collision detection was expensive, but somewhat alleviated by a spatial subdivision of the model's nodes.

Norton et al. modeled fracture as the elastic limit of a material. When an object underwent a sufficient deformation, it would break at the point of greatest stress. As the object was simulated, these fractures grew as a result of stress accumulation (Fig. 17). The authors presumed that materials break more easily under tensile stress rather than compressive stress, so bonds were only broken by excessive stretching. But, there was a problem with simply disconnecting springs. Once a bond was broken, the cube unit typically lost its structural stability. To alleviate this issue, each bond was associated with a cell inside the cube. If one bond in a cell was broken, all bonds in that cell were broken. This resulted in the fragmentation of the object to be based on this cellular structure, adding structural stability to the modules.

Fig. 17: Falling teapot from Norton et al. [13].

3. Graphical Modeling and Animation of Brittle Fracture

In 1999, James O'Brien and Jessica Hodgins *published Graphical Modeling and Animation of Brittle Fracture* [14]. This paper developed a method for crack initiation and propagation in three-dimensional models. They simulated solid volumes using finite element methods, and by analyzing the stresses inside the model as it deforms, they could determine where cracks would form and in what direction they grew.

The crux of the paper was determining the origin and direction of cracks. Crack fractures are caused by internal stresses that emerge as an object deforms. The calculation of these stresses had to be made available by the deformation methods used to simulate the model. Of utmost importance were the stress magnitudes, directions, and whether they were compressive or tensile. For the simulation of the object, O'Brien and

Hodgins developed a model based on continuum mechanics. This method allowed the deformation of the material to be modeled in terms of strain, strain rates, and stress tensors. These quantities could then be used to calculate the elastic potential density of the volume, which was useful for calculating internal forces within the volume. To be able to measure these forces at specific points in the model, the continuous model could be discretized using finite element methods. The authors discretized their model as tetrahedra, based on the principle that tetrahedra can approximate any volume, just as triangles can approximate any surface.

During the simulation, the system calculated all internal forces acting on the tetrahedral nodes. These forces were used to construct a tensor that described the stress in the material and its tendency to separate. If this separation tensor was large enough, the material would split at the node. The separation tensor was represented as a matrix, whose largest eigenvalue could be directly compared to a material toughness to determine whether a fracture would occur. The eigenvector associated with this eigenvalue was used to determine the orientation of the separation plane. Once a fracture was determined, the object's mesh had to be reconstructed to account for the new discontinuity. All tetrahedral elements connected to the fracture node that were split by the separation plane were re-meshed to reveal the fracture (Fig. 18).

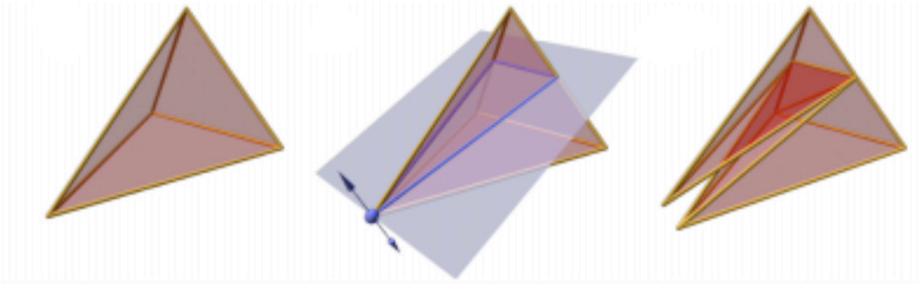


Fig. 18: A tetrahedron split by a separation plane from O'Brien and Hodgins [14].

The results of O'Brien and Hodgins work demonstrated that their fracture model was extremely effective. They provided images and animations of fractures such as a wrecking ball hitting an adobe wall. Also, a shattering bowl was compared to a real world example with strikingly accurate characteristics (Fig. 19). The authors stressed that this formulation was only appropriate for modeling brittle fracture. That is, a material that only goes through elastic deformation before fracture occurs. They tackled ductile fracture in their follow-up paper.

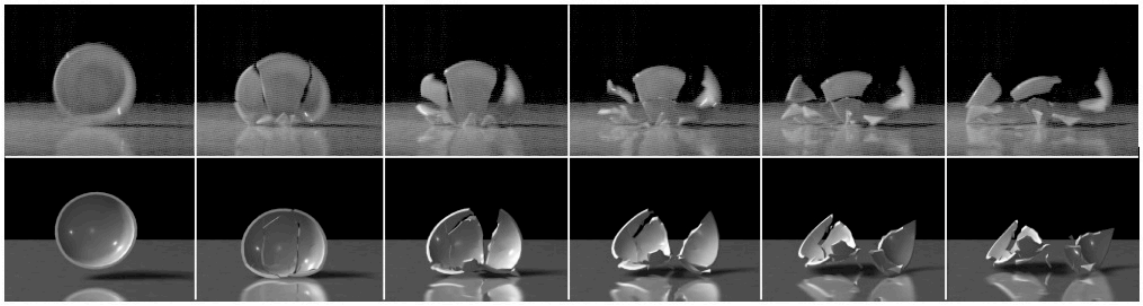


Fig. 19: Results from O'Brien and Hodgins [14].

Dropping a real bowl (top) versus a simulated bowl (bottom).

4. Graphical Modeling and Animation of Ductile Fracture

When deformed, ductile materials undergo a certain amount of elastic deformation before reaching a yield point, at which plastic deformation occurs; that is, deformation in which the material is permanently deformed and will not return to its original shape. Because of this, ductile materials tend to tear instead of shatter, such as with brittle materials. O'Brien, Hodgins, and Bargteil *published Graphical Modeling and Animation of Ductile Fracture* in 2002 [15], in which they introduced a simple modification to their last paper to compensate for the simulation and fracture of ductile materials.

The only major modification the authors made to their previous model is extending their continuum model to account for plastic deformation. They redefined strain as the sum of elastic strain and plastic strain, rather than just elastic strain. They proposed that the total strain could simply be measured from the object's geometric change in shape. Then, they calculated the strain due to plastic deformation using the von Mises yield criterion, which defined the strain at which plastic flow would begin. Since total strain was the summation of elastic and plastic strain, a simple subtraction gave the elastic strain. With the strains known, the fracture creation and propagation methods were the same, and the algorithm proceeded as in their previous paper.

The authors also redefined the fracture threshold. In their previous paper, the elastic limit was used to determine when to initiate a fracture. In this paper, this elastic limit was used to initiate plastic deformation, and a new plastic limit threshold was introduced to initiate fracture. These parameters could be changed to model a wide variety of ductile materials (Fig. 20).

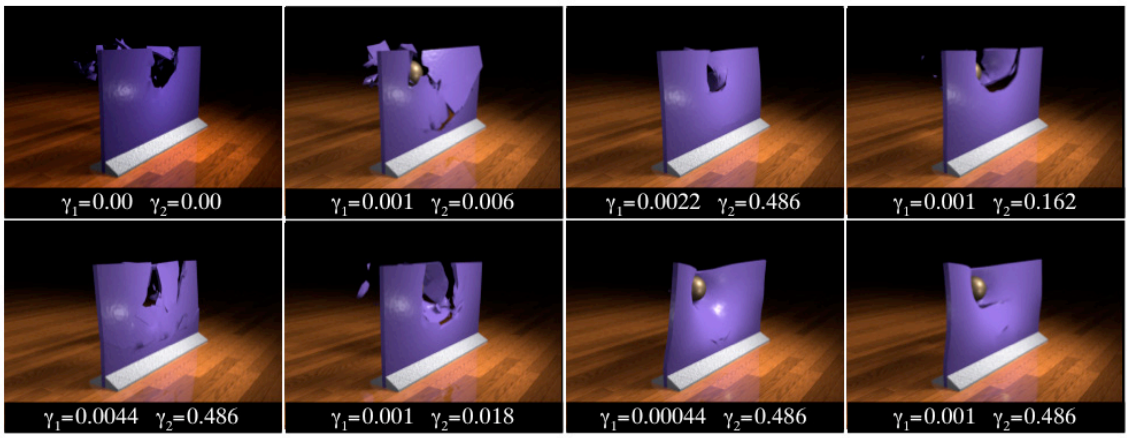


Fig. 20: Changing material elastic (γ_1) and plastic (γ_2) limits from O'Brien et al. [15].

The results obtained from this simple modification allowed the authors to animate a much wider gamut of materials. They provided very convincing results of ductile fracture, such as clay walls, thin sheets, and an unfortunate cartoon character. Again, they compared their results with real world examples, which are strikingly close in appearance (Fig. 21).

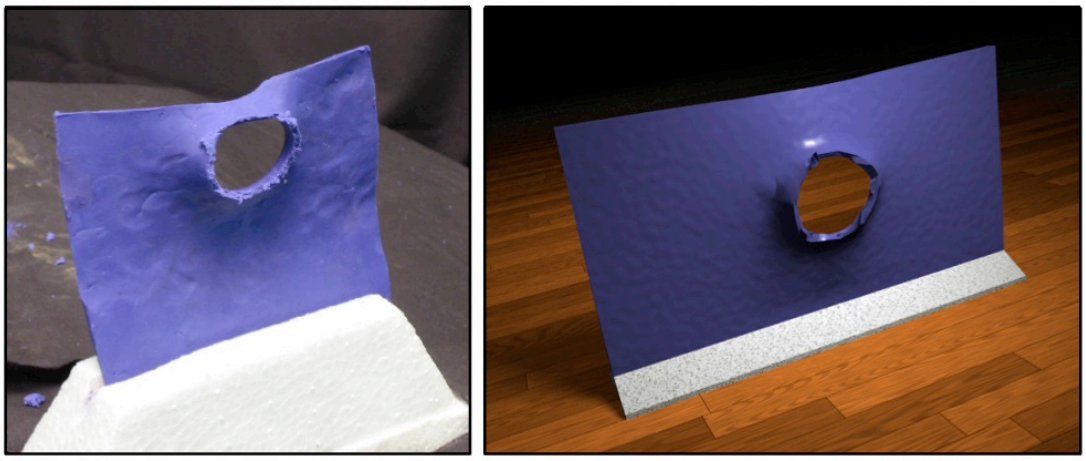


Fig. 21: Projectile shot through a real clay slab (left) versus a virtual simulation (right) from O'Brien et al. [15].

CHAPTER III

METHODOLOGY

To understand the methods developed in this thesis for permanent deformation and tearing of cloth, one must understand the basic underlying principles of particle simulation. Once this basic idea is understood, it can be expanded upon to build a simple cloth simulator. Then, the tearing algorithm can be applied.

A. Basic Particle Simulation

The overall goal of any forward dynamical simulation is to calculate the next configuration of an object, based on current knowledge of that object and any other interacting objects.

Consider a single particle. The particle has a current position and velocity, or *state*. Its state can be computed after a given amount of time as follows:

$$S_1 = S_0 + dS \tag{1}$$

where S_1 is the new state, S_0 is the current state, and dS is the change in state over time. Unfortunately, the particle's exact curves for position and velocity are unknown; therefore dS cannot be simply measured. But, dS can be approximated by assuming that the state's rate of change, its time derivative, is constant over time.

$$dS = h \cdot f(S_0) \tag{2}$$

where $f(S_0)$ is a function that yields the current time derivative of the particle's state, and h is the amount of time that has passed, commonly called the *timestep*. The timestep

chosen to simulate a particle should approximate its motion curves well. If the timestep is too big, the approximation may have a large error and the resulting change in state will be unreasonable and look visually incorrect. Small timesteps approximate the change in state more accurately, but this increases the amount of computation needed to simulate a specific amount of time. This method of calculating the new position of a particle is called Euler Integration (Fig. 22).

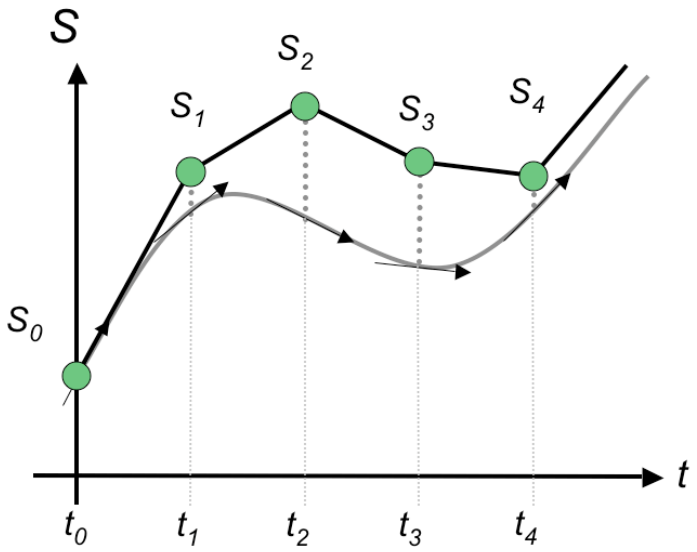


Fig. 22: Approximation of a curve using trajectories at timesteps

The ultimate goal of the derivative function is to calculate the current time derivatives of the particle's position and velocity. Luckily, the current time derivative of the particle's position is simply its current velocity. The current time derivative of the particle's velocity is its acceleration.

$$\frac{dx}{dt} = v \tag{3}$$

$$\frac{dv}{dt} = a \tag{4}$$

Objects accelerate and decelerate due to forces acting on them. Newton's Second Law states that the acceleration of a particle is directly proportional to the net force acting on the particle, and is inversely proportional to the mass of the particle.

$$F = ma \Rightarrow a = \frac{F}{m} \tag{5}$$

where F is the net force acting on the particle, m is the mass of the particle, and a is the acceleration of the particle. By defining forces that act on the particle, its acceleration can be calculated, and therefore its current time derivative can be calculated. That is then used to calculate the approximation of dS, and finally the next state of the particle can be computed.

B. Building a Simple Cloth Simulator

Cloth is not just a simple moving particle. It has an ever-changing internal shape. Because of this, cloth is typically modeled as many small particles connected by a network of edges. These edges can be modeled as dynamic springs that produce forces based on their change in length.

1. Defining the Cloth Model

The particles are laid out in an irregular pattern to avoid pattern artifacts in the simulation (Fig. 23). If many of the edges were to lie in a straight line, the cloth would tend to bend along that straight line. Using an irregular layout provides a certain amount of randomness in the cloth's bending behavior, and creates more realistic and interesting wrinkles.

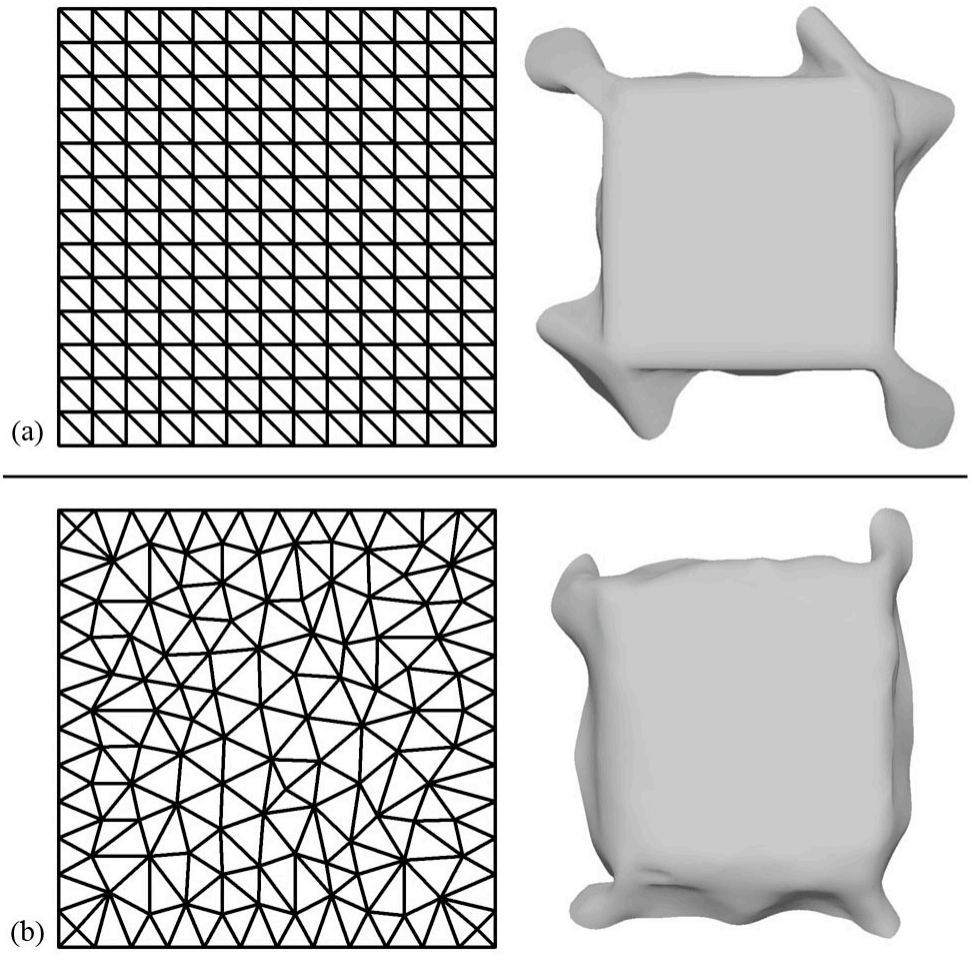


Fig. 23: Particle spacing: (a) regular versus (b) irregular.

(left) mesh layout, (right) draped over a cube. Created in Autodesk Maya [1].

In a particle-based cloth simulation, the mass of the cloth is modeled as being completely concentrated in its particles, i.e. the springs themselves don't have any mass, nor do the faces of the model. This type of model is commonly known as a mass-spring system. In this thesis, mass is distributed to particles based on the area of connected faces. For every (triangular) face on the mesh, one-third of the face's area is added to the particle's mass. Once every face has been processed, each cloth particle will have a mass representative of the amount of nearby material.

2. Forces

To figure out the net force acting on each particle, several forces must be considered, such as internal forces, gravity, wind, friction, and collisions. Internal forces comprised of specially designed spring forces model cloth's resistance to stretching, shearing, and bending. These internal forces are what contribute to most of the model's cloth-like behavior.

The most important internal property of cloth is its resistance to stretching. Stretch resisting forces are modeled using Hooke's Law, which states that a force due to a spring is directly proportional to the distance by which the spring is extended:

$$F_{spring} = -k(l - l_0) \tag{6}$$

where F_{spring} is the restoring force exerted by the spring, l is the spring's current length, l_0 is the spring's rest length, and k is the spring constant. A larger spring constant models a stiffer spring. Using this equation alone will result in a very oscillatory spring

system that never calms down. Therefore, it is important to introduce damping to the model, which will reduce the amplitude of the spring's oscillations over time. Damping is modeled as a force that is proportional to the magnitude of the node's velocity, but in the opposite direction:

$$F_{damp} = -d(v_j - v_i) \tag{7}$$

where F_{damp} is the damping force exerted by the spring damper or dashpot, v_i is the velocity of node *i*, and *d* is the damping constant. A larger damping constant will decrease the spring's oscillations more quickly. By combining the spring force and the damping force, the damped spring force equation is obtained:

$$F_i = -k(l - l_0) - d(v_j - v_i) \tag{8}$$

$$F_j = -F_i \tag{9}$$

where F_i is the damped spring force acting on node *i*. The force acting on node *j*, F_j, is simply the negative of F_i. These stretch forces should be accumulated for every spring, producing a net force acting on each node.

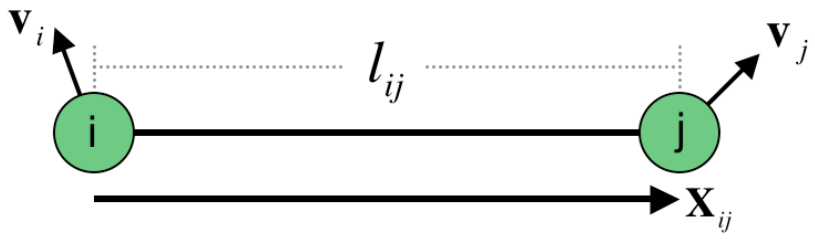


Fig. 24: Components of a stretch force

The three-dimensional model is easily extrapolated from this simple one-dimensional model. The $\mathbb{R}^3$ position of a node is denoted as **x**, and the $\mathbb{R}^3$ velocity of a node as **v**. The vector from node *i* to node *j* is denoted as $\mathbf{X}_{ij}$, with the unit vector $\hat{\mathbf{X}}_{ij}$. The length of $\mathbf{X}_{ij}$ is denoted as l_{ij} and the original rest length is denoted as l_{ij0}. The spring portion relies only on the change in length, so no projection is needed. For the damping portion, the velocity of node *j* relative to node *i* only in the direction of the spring is calculated by projecting it onto $\mathbf{X}_{ij}$. Utilizing these components of the spring force (Fig. 24), a final magnitude is obtained, and the force vector $\mathbf{F}_i$ in $\mathbb{R}^3$ is calculated as:

$$\mathbf{F}_i = \left(-k_{stretch}(l_{ij} - l_{ij0}) - d_{stretch}(\mathbf{v}_j - \mathbf{v}_i) \bullet \hat{\mathbf{X}}_{ij}\right) \cdot \hat{\mathbf{X}}_{ij} \tag{10}$$

$$\mathbf{F}_j = -\mathbf{F}_i \tag{11}$$

$\mathbf{F}_j$ is simply the negative of $\mathbf{F}_i$.

Another important internal property of cloth is its resistance to shearing. Shear resistance can be modeled as a function of the angle between two edges sharing a vertex (Fig. 25).

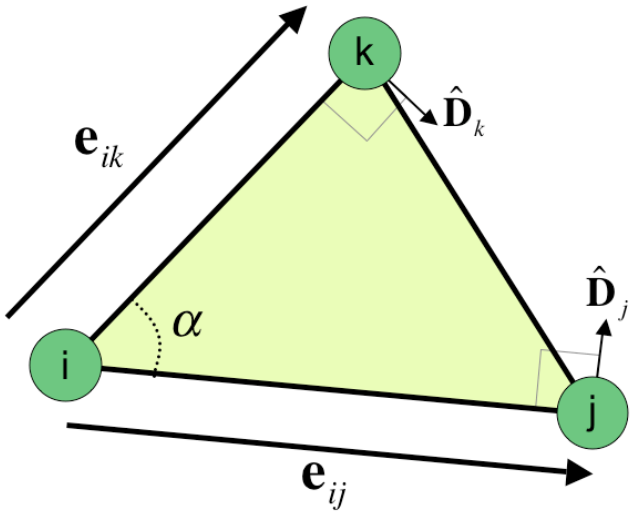


Fig. 25: Components of a shear force

Nodes *j* and *k* will receive torque forces in the interior perpendicular directions, $\hat{\mathbf{D}}_j$ and $\hat{\mathbf{D}}_k$ respectively. An equal and opposite reaction force is applied to node *i*. Let α be the angle between the edge vectors $\mathbf{e}_{ij}$ and $\mathbf{e}_{ik}$, with a rest angle α_0.

$$\mathbf{F}_j = \frac{k_{shear} \cdot (\alpha - \alpha_0)}{\left|\mathbf{e}_{ij}\right|} \bullet \hat{\mathbf{D}}_j \tag{12}$$

$$\mathbf{F}_k = \frac{k_{shear} \cdot (\alpha - \alpha_0)}{\left|\mathbf{e}_{ik}\right|} \bullet \hat{\mathbf{D}}_k \tag{13}$$

$$\mathbf{F}_i = -\left(\mathbf{F}_j + \mathbf{F}_k\right) \tag{14}$$

These forces are modulated by the shear spring constant k_{shear}, which is typically one or more orders of magnitude smaller than $k_{stretch}$. Damping is not applied here because most of the damping required is taken care of by the stretch resistance forces.

The last commonly considered internal property of cloth is its resistance to out-of-plane bending (Fig. 26).

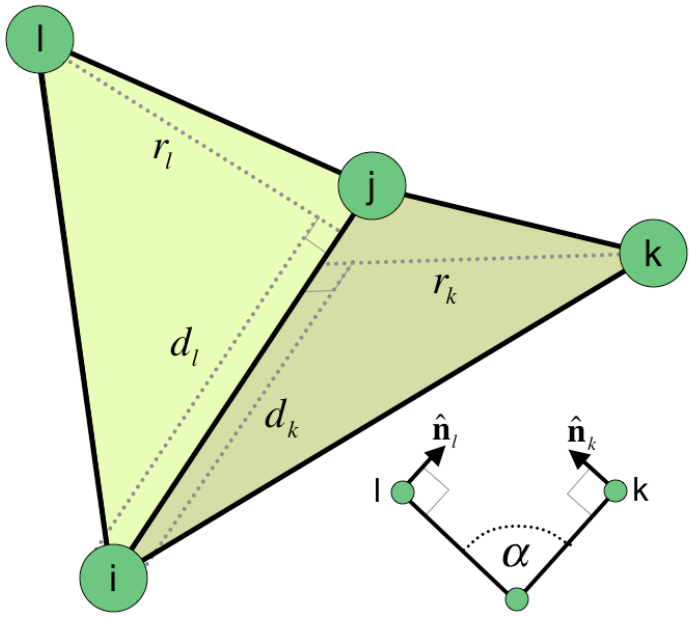


Fig. 26: Components of a bending force

Nodes *l* and *k* will receive forces due to torques in the direction of their face normals, $\hat{\mathbf{n}}_l$ and $\hat{\mathbf{n}}_k$ respectively. The forces given to nodes *i* and are a weighted sum of these forces due to torques in the opposite direction. Let α be the angle between the two faces, with a rest angle α_0. r_l is the radial distance of node *l* from $\mathbf{e}_{ij}$, i.e. the length of the vector perpendicular to line $\mathbf{e}_{ij}$ from a point on $\mathbf{e}_{ij}$ to the position of node *l*. d_l is the length of the projection of $\mathbf{e}_{il}$ onto $\mathbf{e}_{ij}$. r_k and d_k are similarly defined.

$$\mathbf{F}_l = \frac{k_{bend} \cdot (\alpha - \alpha_0)}{r_l} \cdot \hat{\mathbf{n}}_l \tag{15}$$

$$\mathbf{F}_k = \frac{k_{bend} \cdot (\alpha - \alpha_0)}{r_k} \cdot \hat{\mathbf{n}}_k \tag{16}$$

$$\mathbf{F}_i = -\left(\frac{d_l}{|\mathbf{e}_{ij}|} \cdot \mathbf{F}_l + \frac{d_k}{|\mathbf{e}_{ij}|} \cdot \mathbf{F}_k \right) \tag{17}$$

$$\mathbf{F}_j = -\left(\mathbf{F}_i + \mathbf{F}_k + \mathbf{F}_l \right) \tag{18}$$

The forces $\mathbf{F}_l$ and $\mathbf{F}_k$ are modulated by the bending spring constant k_{bend}, which is typically very small since cloth's resistance to out-of-plane deformation is very weak. No damping is required since this force is very small (but highly effective) compared to stretch and shear resistance forces.

External forces are important to consider since they govern how the cloth behaves in its environment. Arguably, the most important is gravity. Gravity is an interesting force because it causes a constant acceleration of an object, regardless of mass. With Newton's Second Law in mind, the force due to gravity is simply:

$$\mathbf{F} = m\mathbf{g} \Rightarrow \mathbf{a} = \mathbf{g} \tag{19}$$

where m is the mass of the node and $\mathbf{g}$ is the gravitational acceleration vector, typically $\langle 0, -9.8, 0 \rangle$ for Earth-like conditions. This force is accumulated on all nodes.

Air resistance and wind are modeled quite differently from other forces. Wind is a relative force, only effective when the motion of a surface opposes the wind direction (Fig. 27). For example, wind coming from a direction perpendicular to the surface $\mathbf{w}_h$ will have maximum effect. Wind coming from a parallel direction $\mathbf{w}_l$ will have minimal effect. To model this phenomenon, the surface to be the geometry of a face defined by nodes *i*, *j*, and *k* is considered.

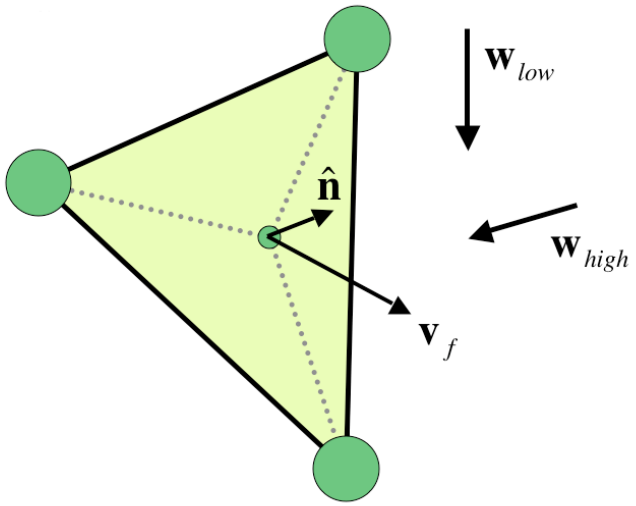


Fig. 27: Components of a wind force

The face has a normal $\hat{\mathbf{n}}$. The velocity of the face $\mathbf{v}_f$ is the averaged sum of the velocities of nodes *i*, *j*, and *k*. The wind vector **w** relative to the face's velocity is projected onto the face's normal, modeling the wind's contribution based on how directly it hits the surface. That value is modulated by the face's area A and the environment's air resistance μ. This gives a final magnitude, which is equally distributed to each node of the face in the opposite direction of the normal.

$$\mathbf{F}_{i,j,k} = A\mu \cdot \left[\left(\mathbf{w} - \mathbf{v}_f \right) \bullet \hat{\mathbf{n}} \right] \cdot \hat{\mathbf{n}} \tag{20}$$

Without wind (**w** set to **0**), this force models air resistance as viscous drag. These forces are accumulated for every face.

3. Integration Revisited

Euler integration is the simplest way to advance a particle, but proves to be very unstable when the force equations are very stiff, that is, the resulting force oscillates rapidly around a value of zero. To correctly approximate a stiff force, very small steps must be taken. This results in extremely slow cloth simulation, and if these forces change suddenly, not even small timesteps will keep the system stable (Fig. 28). A more precise integration method is needed.

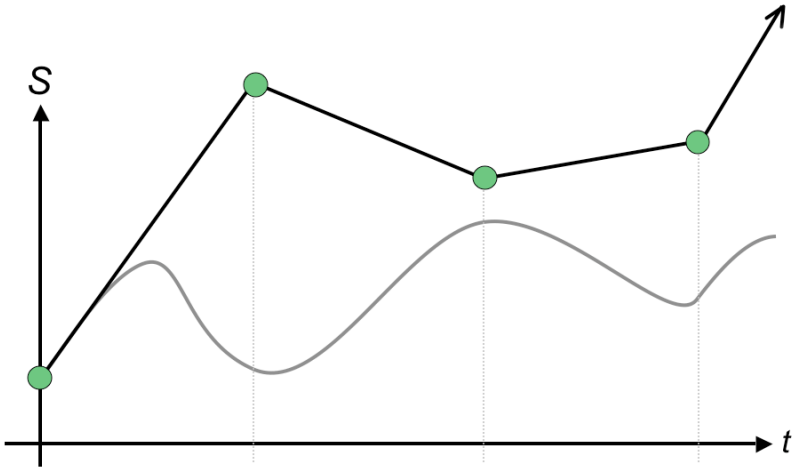


Fig. 28: Example of Euler integration

The approximation of *dS*, the change in state, is actually the first-order expansion of the Taylor series of the state's derivative. To alleviate the problems of Euler integration, this Taylor series can be further expanded. Expanding one more order yields the Midpoint Method of integration (Fig. 29 and Fig. 30). Essentially, an Euler step is computed as normal, storing the resulting change in state (1). The derivative is evaluated halfway between the current state and this change in state (2). This derivative is then

used (instead of the original one) to compute the final state of the particle for that timestep (3).

$$K = h \cdot f(S_{current}) \tag{21}$$

$$dS \approx h \cdot f\left(S_{current} + \frac{K}{2}\right) \tag{22}$$

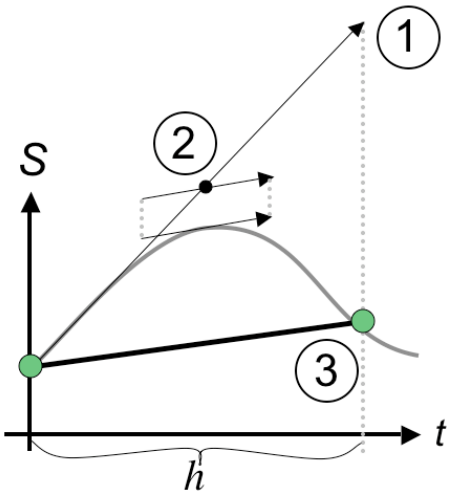


Fig. 29: The Midpoint Method of integration

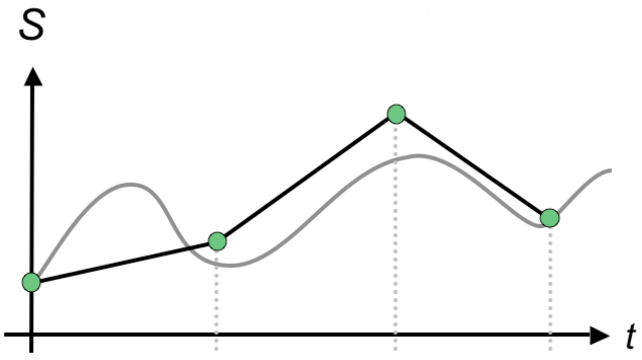


Fig. 30: Example of Midpoint integration

Taking this a few steps further, a fourth-order Taylor series expansion of the derivative can be achieved. This yields the Fourth-Order Runge-Kutta method of integration. The final expanded equation of motion depends on four evaluations of the trajectory function, and is typically calculated as follows:

$$K1 = h \cdot f(S_{current}) \tag{23}$$

$$K2 = h \cdot f(S_{current} + \frac{K1}{2}) \tag{24}$$

$$K3 = h \cdot f(S_{current} + \frac{K2}{2}) \tag{25}$$

$$K4 = h \cdot f(S_{current} + K3) \tag{26}$$

$$dS \approx \frac{1}{6}K1 + \frac{1}{3}K2 + \frac{1}{3}K3 + \frac{1}{6}K4 \tag{27}$$

See [3] for an in-depth discussion of the Midpoint Method and underlying principles of these methods. Even though this requires that the derivative function be calculated four times, the gain in stability allows a timestep more than four times greater than an Euler step. Fourth-Order Runge-Kutta is sufficient for basic cloth simulation. High-end production systems use a much implicit integration scheme, which is discussed in [2] and [12].

4. Constraints and Collisions

The system described so far can simulate cloth falling under gravity and interacting with wind. The most interesting behavior of cloth comes with its interaction with other

objects. Of most importance is the ability to fully constrain certain cloth particles in space.

There are many ways to constrain vertices in particle-based systems. One common way to constrain a vertex to a point in space is to set its mass to infinity. Therefore, due to Newton's Second Law, the resulting acceleration will be infinitesimally small. In systems like cloth, the mass of the particle is rarely stored. Instead, the particle's inverse mass, $\frac{1}{m}$, is stored. An advantage of this is to be able to simply set the particle's inverse mass to zero when its position needs to be constrained to a point in space.

$$\mathbf{a} = \frac{\mathbf{F}}{m} = \frac{\mathbf{F}}{\infty} \Rightarrow \mathbf{a} \approx \mathbf{0} \tag{28}$$

Whenever a vertex needs to be explicitly moved, it is temporarily point constrained. Consider the case of clamping the cloth on the left and right sides, uniformly moving the right side, and finally releasing it. The held vertices are constrained to their current position. To move the right side, the position of each of those vertices can be simply altered, still maintaining an inverse mass of zero. To "let go", the constraints are undone by resetting the vertices' inverse masses back to their initial values.

With a coarse enough mesh, the cloth model can be manipulated at interactive rates. A vertex can be picked in the 3D viewer by projecting the mouse cursor from the 2D camera space onto the cloth surface. If a vertex is sufficiently close to the projected point, that vertex is constrained to the projected mouse position. The constraint is

maintained until the mouse button is released. One downside to explicitly moving vertices is that it is very easy to unrealistically deform the cloth, which may lead to numerical instability.

This thesis does not implement collisions due to the amount of extra computation time that is required. The most basic collision detection algorithm requires that an intersection test is done for every vertex against every face. With a mesh containing 25 vertices, roughly 850 vertex-face intersection tests would be required. On top of that, it is also necessary to test for intersections between edges, resulting roughly in an additional 3,500 edge-edge intersection tests. Even without collisions, interesting results of tearing cloth can be created only with the ability to constrain vertices.

C. Tearing Cloth

Cloth is not a continuous medium. It is a highly complex network of threads that are woven together. This structure is maintained purely by friction. Discontinuities in cloth can occur when friction can no longer hold the structure together. If a thread breaks, the local amount of friction decreases, and the probability of a discontinuity at the macro level increases. On a macro level, weave separation and thread breaking can be modeled simultaneously and approximated as a discontinuity in the cloth mesh (Fig. 31).

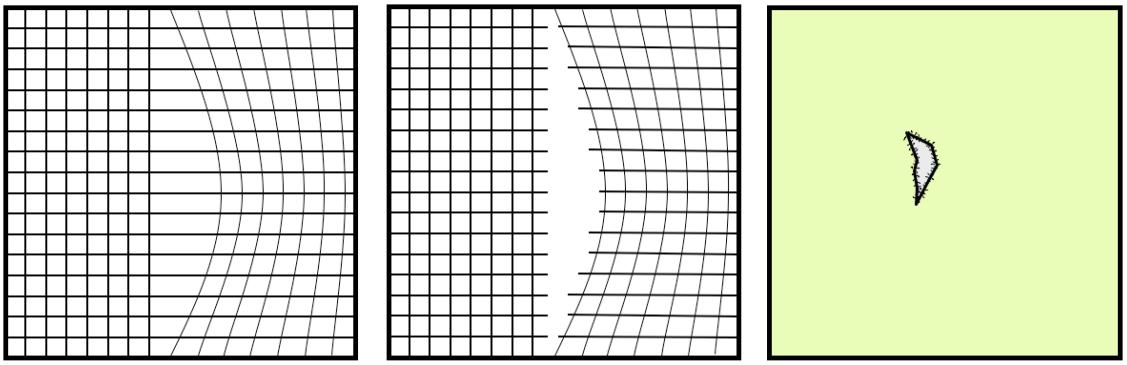


Fig. 31: Cloth tearing on the micro and macro level. (a) Threads splitting on the micro level, (b) threads breaking on the micro level, (c) cloth tearing on the macro level

Just as cloth can be simulated on a macro level, its tearing behavior can be simulated on a macro level. It is not necessary to calculate strain for each individual thread. The cloth mesh's already available discretization that is used to simulate its motion can be used advantageously. When fracturing solids, the mesh used to represent the object may not be fine enough to represent small-scale fractures, and the mesh structure will be evident in the result. One solution is to make the mesh denser, which makes these artifacts less apparent. Since cloth meshes are already dense, these artifacts will be minimized.

1. Calculating Strain

The method used to calculate strain and fracture the cloth mesh is based largely on the method of O'Brien and Hodgins [14]. Their work on brittle fracture within solids is simplified in this thesis. They calculate strain based on the internal forces of the object,

but their solid objects do not change shape until a fracture has already been established, thus making these force calculations more difficult. In cloth, the internal forces are almost free, due to the nature of cloth simulation (a network of springs). The internal forces of cloth are directly proportional to the length of its springs, and therefore the strain can be computed from these lengths.

In addition to the strain simplification, the mesh reconstruction algorithm is simplified. With three-dimensional solids, any object can be approximated by a collection of tetrahedra. Cloth can be thought of as a two-dimensional medium, where any shape can be approximated by a collection of triangles. The process of splitting such a mesh is a simplified form of the same problem dealt with by O'Brien and Hodgins [14], and is explained in detail below.

Tearing in cloth can be thought of as the loss of frictional ability to hold itself together. As the cloth weave is stretched, the friction in the weave can no longer hold the structure together, and the threads become separated. This can be modeled as geometric strain. In this cloth model, this entity can be measured as the change in length of internal springs (Fig. 32).

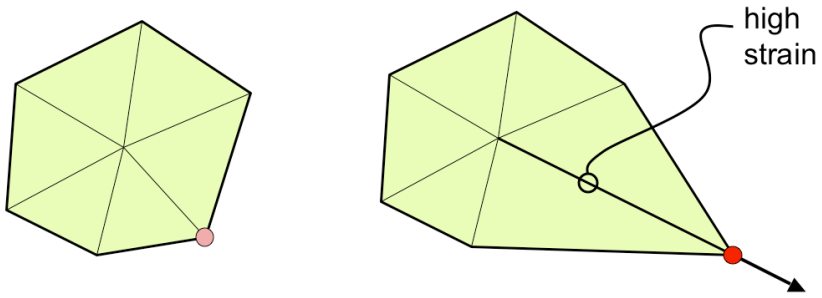


Fig. 32: Strain as spring elongation

Given a single vertex, the strain of connected edges can be measured and then be represented as the strain at the vertex (Fig. 33). This is useful for calculating where a tear will occur, but does not provide any information for what orientation the tear will have. By combining the amount of strain on an edge and the direction of that edge, a strain vector can be defined for that edge. These vectors can then me accumulated to fully describe the strain at a vertex.

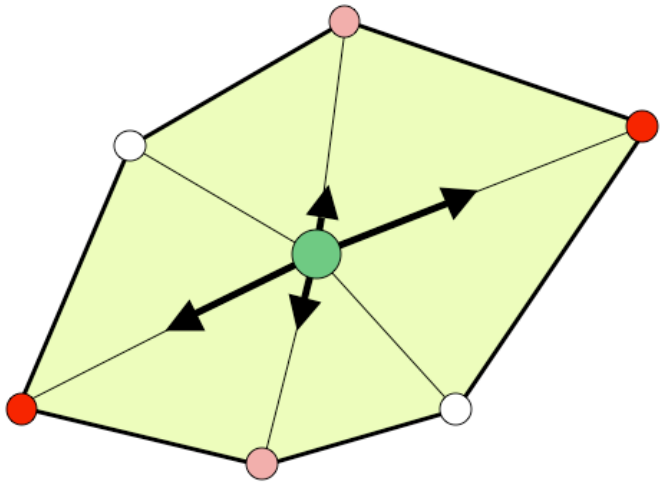


Fig. 33: Individual strain vectors at a vertex

These vectors cannot be simply added because two vectors might cancel each other out. Instead, they are combined into a separation tensor, very much in the same fashion as O'Brien and Hodgins [14] does with forces internal to an object.

A function $\mathrm{M}(\mathbf{v})$ is defined that, given a vector, builds a symmetric matrix that has an eigenvector **v** with the corresponding eigenvalue of *e*. The other two eigenvalues of the matrix are zero (with null eigenvectors). The $\mathrm{M}(\mathbf{v})$ function is defined as:

$$\mathrm{M}(\mathbf{v}) = \begin{cases} \mathbf{v}\mathbf{v}^T / e & : \quad \mathbf{v} \neq \mathbf{0} \\ \mathbf{0} & : \quad \mathbf{v} = \mathbf{0} \end{cases} \tag{29}$$

The separation tensor at a vertex is then defined as:

$$\varsigma = \left(\sum_{\varepsilon \in \{\varepsilon^+\}} \mathrm{M}(\varepsilon) \right) - \mathrm{M}(\varepsilon^+) \tag{30}$$

where $\sum_{\varepsilon \in \{\varepsilon^+\}} \mathrm{M}(\varepsilon)$ is the sum of all symmetric matrix representations of each of the strains at the vertex, and $\mathrm{M}(\varepsilon^+)$ is the symmetric matrix representation of the sum of all strains at the vertex.

The three eigenvectors of the separation tensor are then calculated. The largest of these three is the most significant and is defined as the strain vector. This vector describes the overall direction of the strain. If a tear occurs, the direction of this vector is perpendicular to the line that slices the cloth (Fig. 34). The magnitude of this vector is what is compared to material strength.

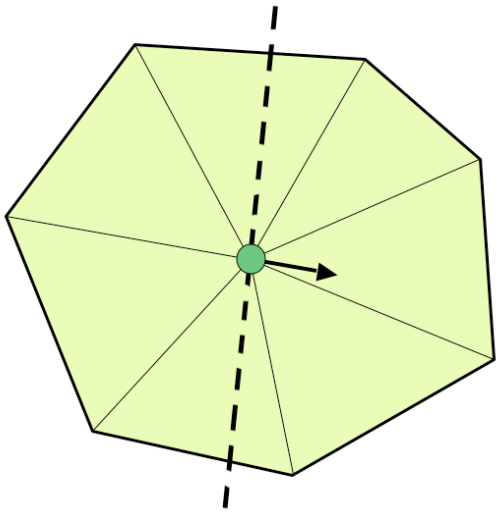


Fig. 34: Strain vector and slice line

During a given timestep, more than one tear may occur. Much like collisions are handled in a dynamic system, when multiple tears occur, the system is re-integrated slowly through the timestep, resolving tears one at a time. First, the tear vertex with the most amount of strain is identified, which represents the first vertex that would split over the course of the timestep. Assuming that strain changes linearly over the timestep, a ratio of the strength of the material with respect to the strain at the current vertex can be created.

$$\text{fraction of timestep} = \frac{\varepsilon_{strength}}{\varepsilon_{current}} \tag{31}$$

The system is integrated forward from the beginning of the timestep by this fractional amount to compute the state of the mesh just as the tear occurs. After the tear is resolved,

the remainder of the timestep is integrated. If another tear is found, this process is repeated, integrating forward from the previous tear's moment in time.

Due to cloth's weave structure, the direction a tear will propagate is not entirely isotropic. This is one example of micro level behavior that emerges on a macro level. Cloth typically tears in the warp or weft direction of its weave. This behavior can be enforced by modulating strain vectors according to their closeness to a warp or weft (parametric *uv*) direction (Fig. 35). The dot product of a strain vector and the warp or weft vector is used to do this. This mimics a stronger resistance to tearing as the strain vector diverges from warp and weft directions.

$$\begin{aligned} \varepsilon_{weft} &= \varepsilon \bullet \mathbf{u} \\ \varepsilon_{warp} &= \varepsilon \bullet \mathbf{v} \end{aligned} \tag{32}$$

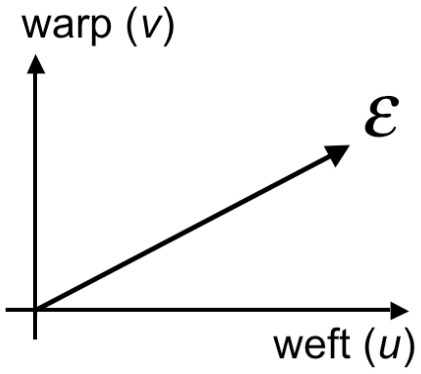


Fig. 35: Modulating strain based on warp/weft direction

2. Splitting the Mesh

Creating new geometry is a complicated process and can produce edges that are ill conditioned for simulation. An edge that is extremely short in comparison to other edges results in a spring that will require a lower timestep to maintain numerical stability. To alleviate this problem, the slice can be snapped to existing edges. Due to cloth's finely triangulated and irregular mesh, visual artifacts will be minimal.

Snapping is achieved by choosing the connecting edges that are closest to the slice line (Fig. 36). For each edge that contains the tear vertex, the dot product of the slice vector and the edge vector is used to find a combination that yields the smallest angle, therefore the closest. This is done for each direction of the slice line, originating at the tear vertex.

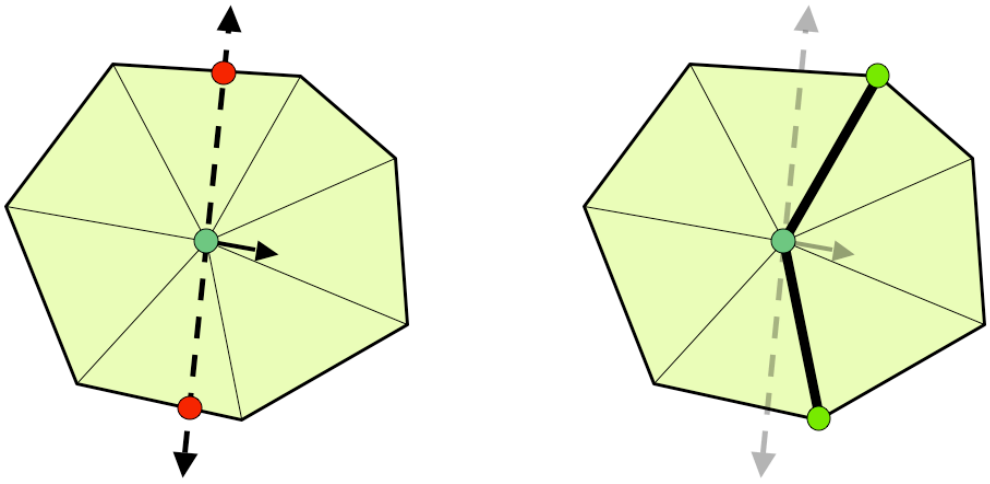


Fig. 36: Snapping slice line to nearest edges

The distance a tear progresses is limited by the triangulation of the mesh. A coarse mesh will produce large tears, whereas a fine mesh will produce small tears. If the

mesh's edge lengths are widely non-uniform, so will be the tear lengths. The visual results of this thesis rely on a uniform fine mesh.

Once a tear vertex and pair of edges are defined, the mesh separation process can proceed. This involves creating a new vertex, two new edges, and the redefinition of some existing faces. But before any new geometry is created, it is important to figure out which faces and edges will incorporate this new geometry.

Faces and edges associated with the new vertex are chosen based on their sided-ness of the sliced edges. A face's sidedness is determined by walking the mesh along faces that contain the tear vertex. Starting with any of the two slice edges, any one of the two faces that share that edge is picked. That face is added to a redefinition list. Next, the other edge in that face that contains the tear vertex is found. Given that edge, the opposite face is found and added it to the redefinition list. This process is repeated until the other sliced edge or a boundary edge is reached. All faces and edges that were added to the redefinition list, including the slice edges, are then redefined such that they use the newly created vertex. The other faces and edges continue to use their original configuration. Now that the faces and edges that will use the new geometry are known, the new geometry can be created.

First, a new vertex (v_{new}) is created by duplicating the tear vertex (v_{tear}). Then, two new edges (e_{new1} and e_{new2}) are created that connect the new vertex to the vertices on the slice edges that are opposite to the tear vertex ($v_{opposite1}$ and $v_{opposite2}$). With the new geometry defined, any faces and edges that use this new geometry must also be defined.

In this example, there are four faces (f_{up1}, f_{up2}, f_{up3}, f_{up4}) and three edges (e_{up1}, e_{up2}, e_{up3}) that need to be updated (Fig. 37).

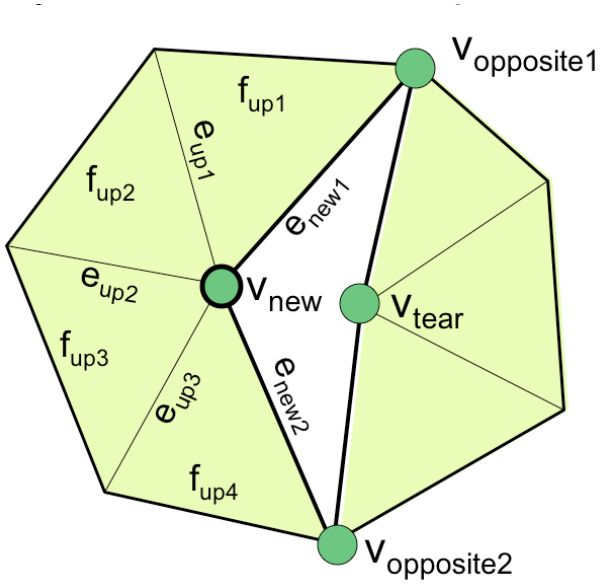


Fig. 37: Splitting the geometry

If a tear begins on the mesh boundary, the splitting algorithm is simplified. A new vertex (v_{new}) is still created, but in this example, there will only be one new edge (e_{new}). Only one face (f_{up}) and one edge (e_{up}) need to be updated (Fig. 38).

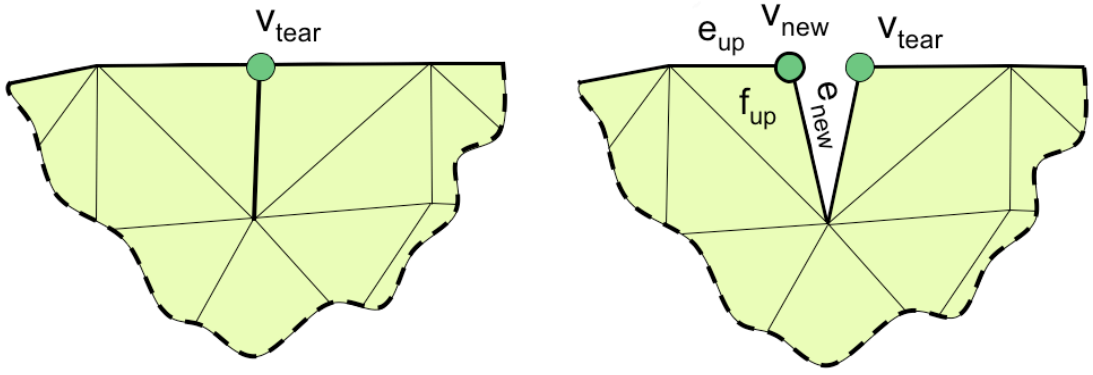


Fig. 38: A tear starting on the boundary

When a tear terminates on the mesh boundary, there is a single remaining vertex holding the mesh together. Since this vertex represents an infinitesimally small amount of material, this vertex as well is broken as well (Fig. 39). Other mesh updates are similar to the previous boundary case.

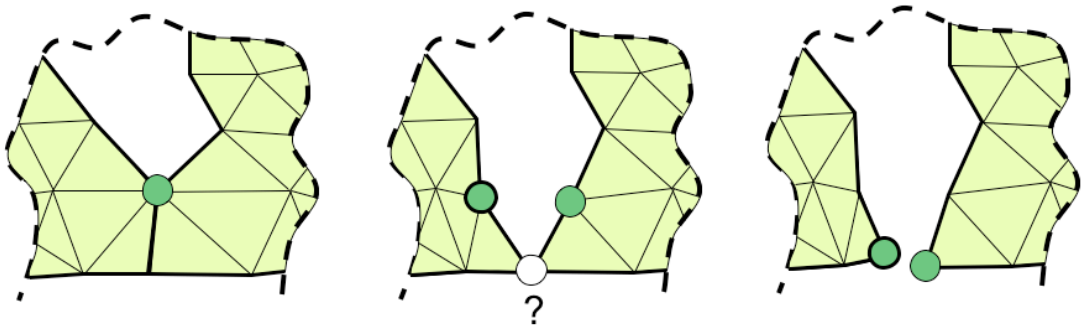


Fig. 39: A tear ending on the boundary

After the mesh has been split and new geometry has been constructed, it must be re-initialized for simulation. By default, new vertices have zero mass and zero velocity. To calculate the mass for the new vertex, the same method is used as when setting up the cloth. The same is done for the torn vertex. The velocity of the new vertex is simply copied from the torn vertex. New edges must also be initialized for simulation.

Even though purely physically based tearing can be interesting, it may not result in exactly what the user desires. The locations of tears can be influenced by changes in material strength. Rather than the whole cloth object having a uniform strength, the cloth's individual vertices can have their own strength parameter. These can be assigned manually, or by using a strength map, applied like a texture map (Fig. 40). During the

initialization of the cloth mesh, each vertex looks up its strength based on its parametric (*uv*) location in the strength map. This allows the user to define where tears will occur, but does not necessarily define when they will occur.

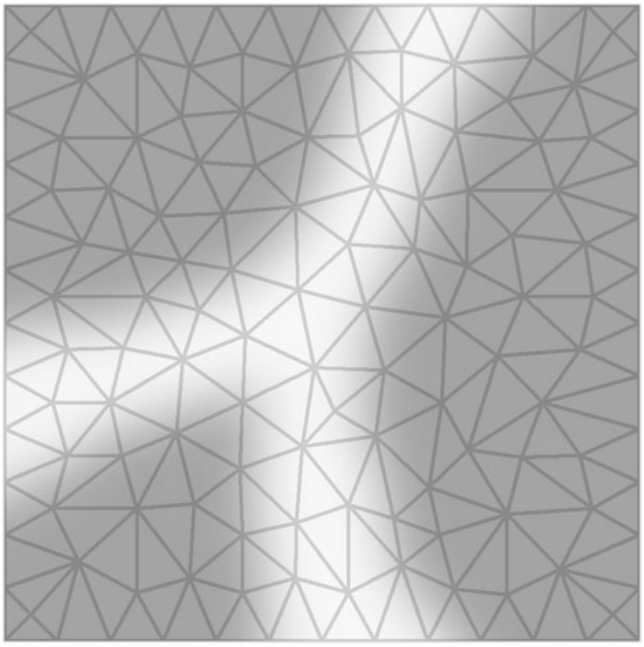


Fig. 40: Strength map

For more control over when tears occur, the entire physically based technique can be ignored. The strain mechanism and the tearing mechanism of this thesis run separately, independent from each other. Tears could easily be initialed manually, or by some other algorithm.

3. Plastic Deformation

Cloth is not a completely elastic material. Certain deformations can have permanent effects on its internal structure, a characteristic of plastic deformation. On the micro level, this is caused by weave slippage. This can be modeled on the macro level as an increase in surface area. To achieve such an increase, spring rest lengths simply need to be modified.

So far, the cloth-tearing model has been brittle. That is, it only undergoes elastic deformation. It is true that cloth does behave elastically most of the time. As cloth is deformed, it will display elastic behavior up to a certain point, the yield point, at which slippage (plastic deformation) will start to occur. As stress increases, slippage continues until the fracture point, which is when threads break.

Creating a rule to initiate plastic deformation, or plastic flow, at the yield point requires a more robust representation of strain. A spring's total strain can be composed into two components, elastic strain and plastic strain.

$$\varepsilon_{total} = \varepsilon_{elastic} + \varepsilon_{plastic} \tag{33}$$

In the perfectly elastic strain model, $\varepsilon_{plastic}$ is essentially always zero. For plastic deformation, the rest length of that spring is modified by a small amount if the elastic strain in a spring reaches the yield point. That increase in length represents an addition to the spring's plastic strain, and the amount of elastic strain will decrease by that amount. While elastic strain is ever changing, once plastic strain is added, it's permanent. As the spring continues to stretch, more elastic strain is introduced. If pulled beyond the yield point again, additional plastic deformation occurs. An example is given in Fig. 41.

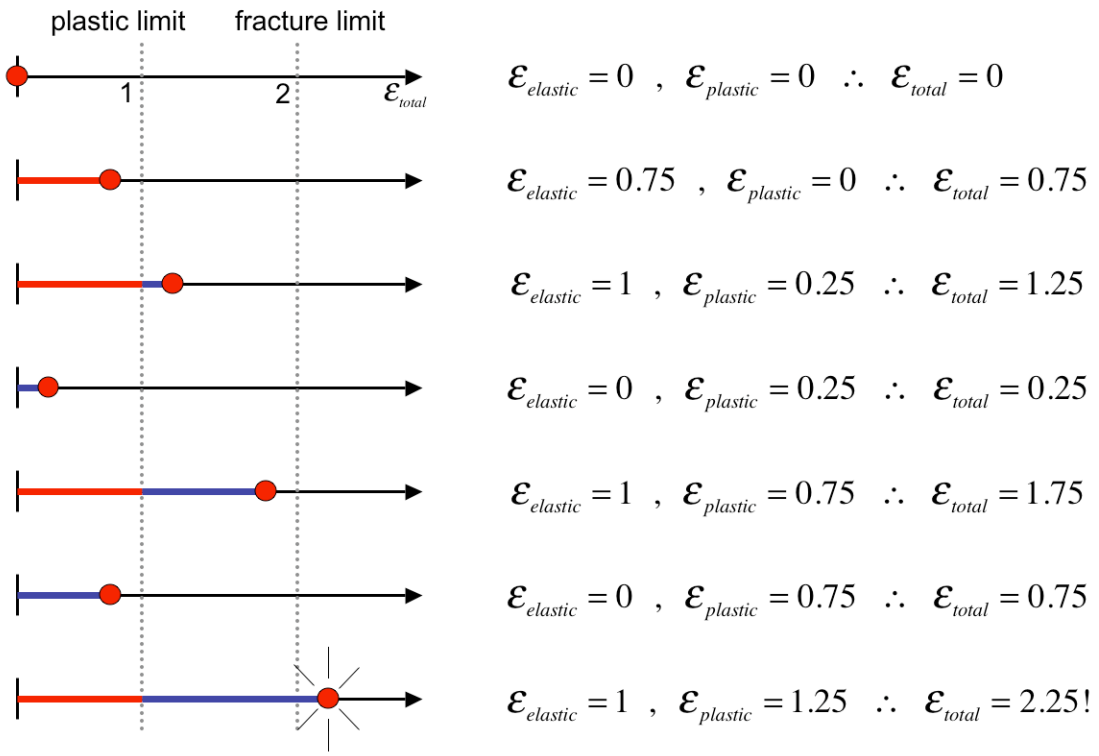


Fig. 41: Example of plastic deformation

Elastic strain is used to measure when a plastic update should happen, whereas the total strain is used to measure when a fracture will occur. To keep track of both strains, an additional rest length quantity is added to each edge to represent the spring's original undeformed rest length. Strain computations proceed as normal, using the spring's current rest length to compute elastic strain, and it's original rest length to compute total strain.

In this cloth model, strain is measured on edges and transferred into vertices. All strain computations are then done at the vertices, so a clear reverse-transferal of plastic

flow from vertices to edges must be established. Given a vertex, the edges that are attached to it which have a positive increase in length are modified. The spring's current length is stored as the current *rest* length. As the simulation continues, these springs will settle to this new length, therefore increasing the surface area of any adjacent triangles. The total strain (calculated from original rest lengths) is still used to compute fracture occurrence and orientation.

D. Rendering

The simulator only provides a rough OpenGL rendering of the cloth. An exporter was written to bring the simulation data into Maya for a higher quality rendering.

It's not necessary to export every timestep of the simulation. With timesteps as small as 0.001 seconds, 1000 frames would have to be stored to produce 1 second of animation. An acceptable frame rate for motion graphics is 30 frames per second. As the simulation progresses, the state of the cloth is cached in memory at intervals roughly equal to $1/30^{th}$ of a second. After a simulation is complete, the cache is written to disk. Each frame is written in the Alias Wavefront OBJ format [4], which holds vertex, face, and uv information.

To get this sequence of OBJ files into Maya, a script was written in the Maya Embedded Language (MEL). Each frame is imported as a mesh using Maya's own OBJ importer. Then a process is run that sets visibility keyframes on each mesh, causing it to be visible for only the time that it represents. The resulting triggering of visibility creates

the illusion of motion. The animation can be easily stretched or compressed by uniformly scaling the visibility keyframes.

To give the cloth its visual qualities, a cloth shader was built using Maya's shader tool, Hypershade. A simple Lambertian shader with very little reflectivity was used as a starting point. A texture map was used as the base color. Next, a procedural cloth weave was used as a subtle bump map. An additional node was added to catch light when we see the surface at a glancing angle. This mimics fuzz on the surface.

The cloth model was lit and rendered in Maya. Pixar's Renderman [16] software was used to render an additional ambient occlusion pass, which provides realistic and appealing shadowing where surfaces are close to each other. The result of this pass was multiplied with the Maya render to produce a final image (Fig. 42).

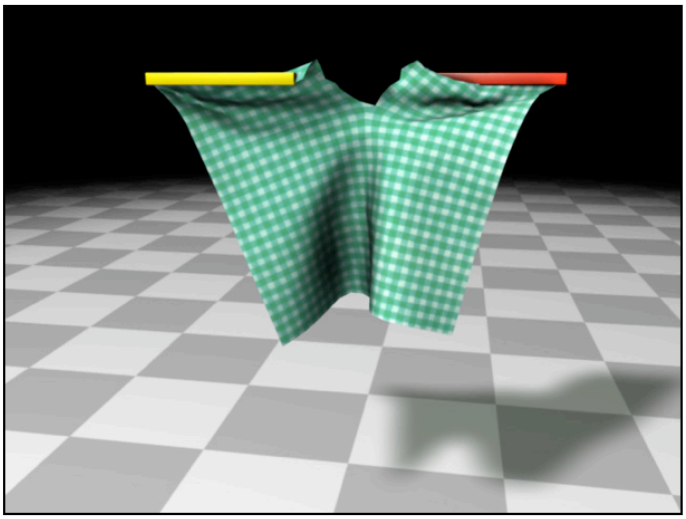


Fig. 42: Example rendered frame

CHAPTER IV

EVALUATION

The animations created from this thesis show a variety of cloth deformation and tearing behavior. The first three examples feature fully constrained vertices held by a static yellow object. A moving red object pulls other vertices to create the tears in the cloth. The fourth example shows the effect of harsh winds on cloth. The fifth example features plastic deformation, and the sixth example demonstrates the use of a strength map. The last animation shows an example of the interactive simulator and its visualization of strain.

Offline, non-interactive simulations were run to produce high-resolution cloth animations. On an Apple Power Mac with dual 2.5ghz G5 processors, cloth meshes comprised of 400 nodes took roughly one hour to produce one second of animation. Interactive simulations required a coarse mesh of about 25 nodes, with spring damping parameters set to a minimum. This resulted in the mesh having severe and visible oscillations, resembling a rubber sheet more than cloth.

Example 1 demonstrates a lateral tear (Fig. 43). One side of the cloth is held in place, while the opposite side is pulled at a constant rate. The cloth stretches and shortens in the middle, displaying the cloth's resistance to gains in surface area. A tear originates in the middle and rapidly progresses towards the cloth boundaries in a direction perpendicular to the motion of the side being pulled. Interestingly, many more tears originate close to the main tear, suggesting that a higher resolution mesh may result a high quality fringed edge.

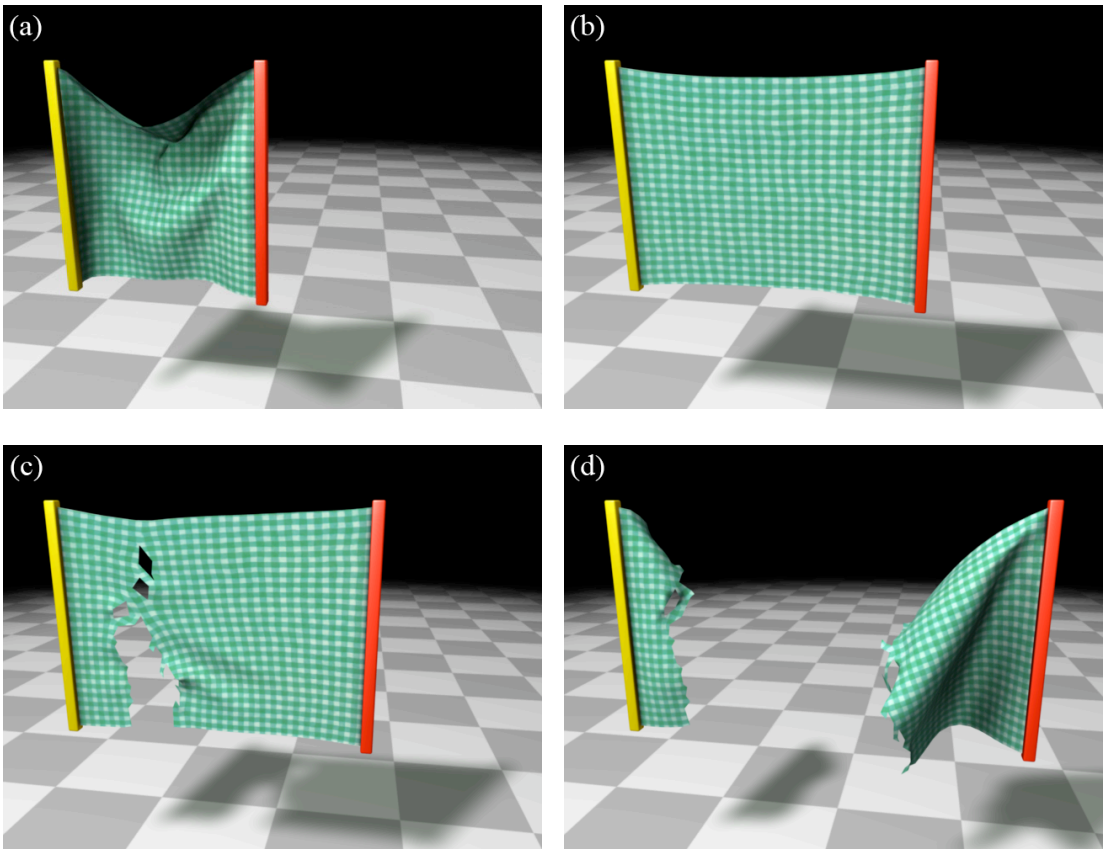


Fig. 43: Example 1: Lateral tear.

(a) at rest, (b) just before the tear, (c) mid-tear, (d) post-tear.

Example 2 is a shearing tear (Fig. 44). Half of the top boundary of the cloth is held in place, while the other half of the boundary is pulled away from the camera. The tear originates at the cloth's boundary between the two objects and progresses downward as the cloth is pulled. It can be seen here that the discretization of the mesh results in a *snagging* behavior. The amount of strain at each torn vertex grows slowly, and then suddenly releases as the vertex is torn.

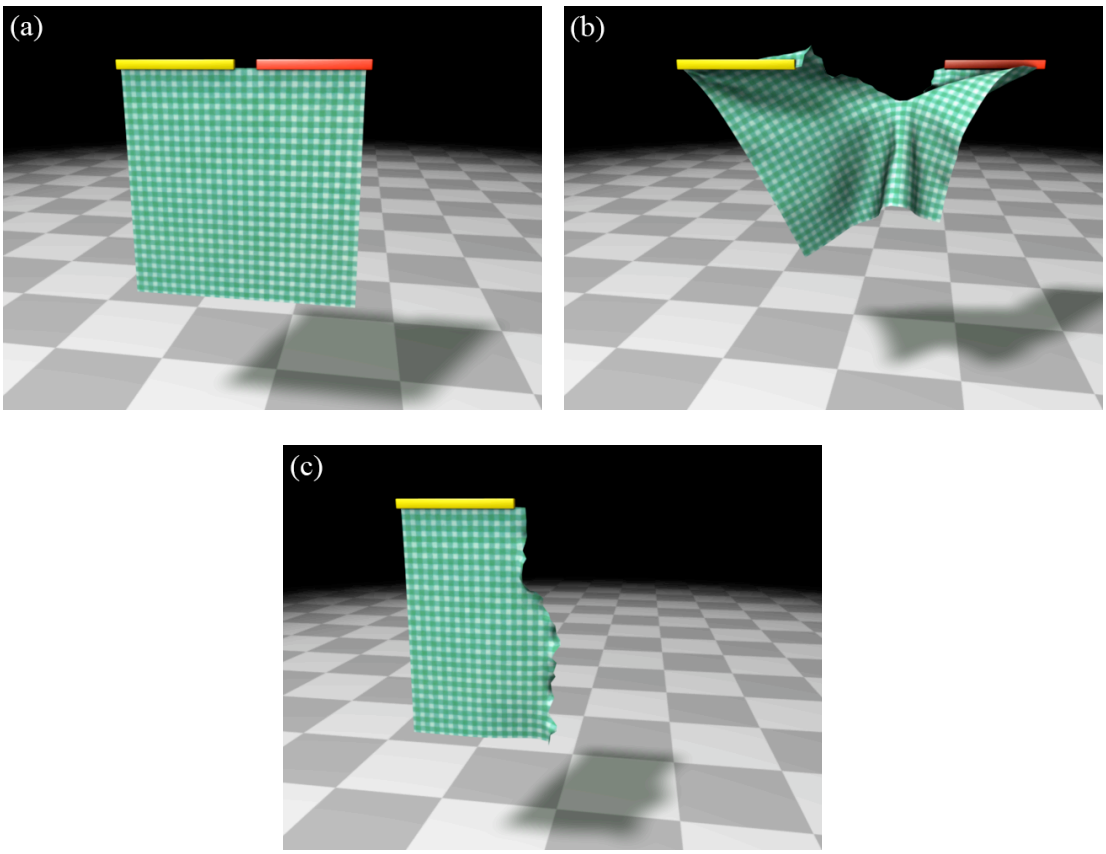


Fig. 44: Example 2: Shearing tear.

(a) at rest, (b) mid-tear, (c) post-tear.

Example 3 was created during a real-time session (Fig. 45). To achieve interactive rates, a very coarse mesh was used, and the damping of stretch spring was greatly reduced. The cloth's perimeter is fully constrained, and a red ball representing the user's mouse cursor grabs a vertex near the corner of the cloth and pulls diagonally. As the user changes the pulling direction, it is more evident here than in the first example that the direction of each tear is perpendicular to the motion of the pulling force.

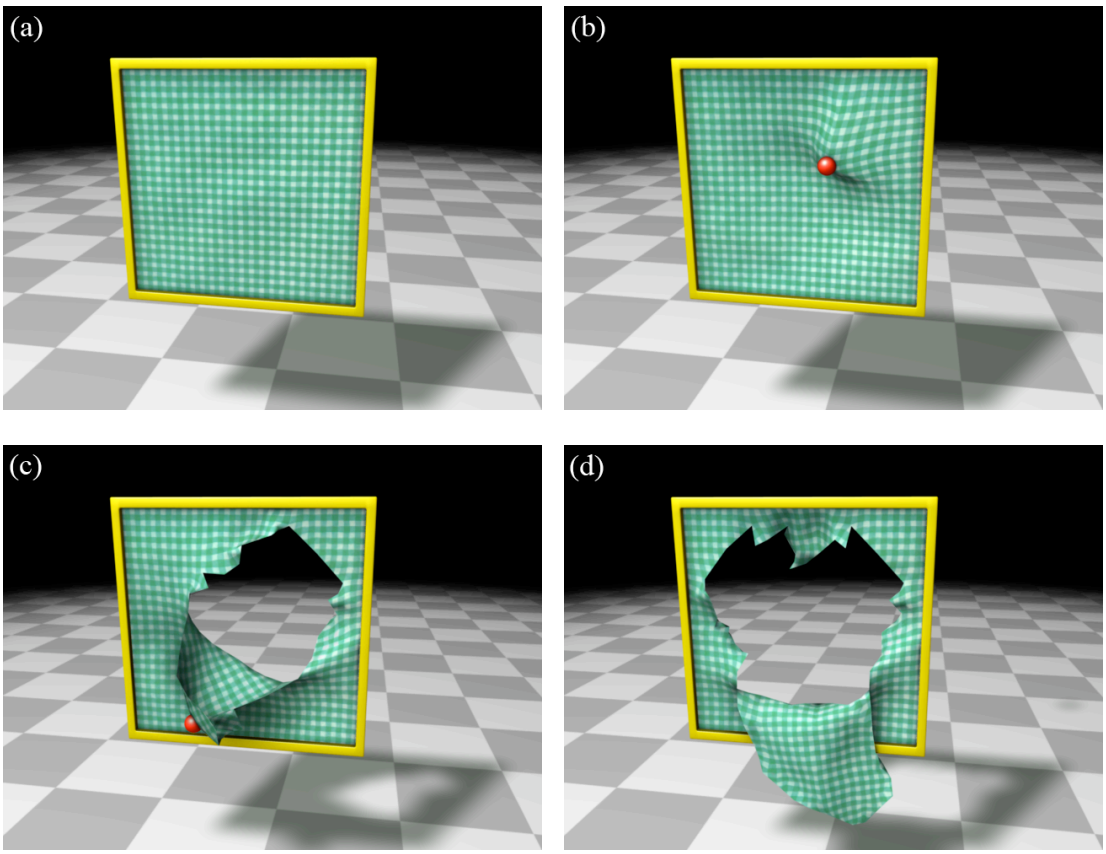


Fig. 45: Example 3: Interactive tear.

(a) at rest, (b) pulling on a vertex (c) mid-tear, (d) post-tear.

Example 4 features a hanging piece of cloth, with its sides constrained (Fig. 46). It fully relaxes and is then blown apart by a sudden blast of wind. This demonstrates the amount of strain and damage that wind alone can cause. The cloth is torn in multiple places before the center portion is completely disconnected and blown away.

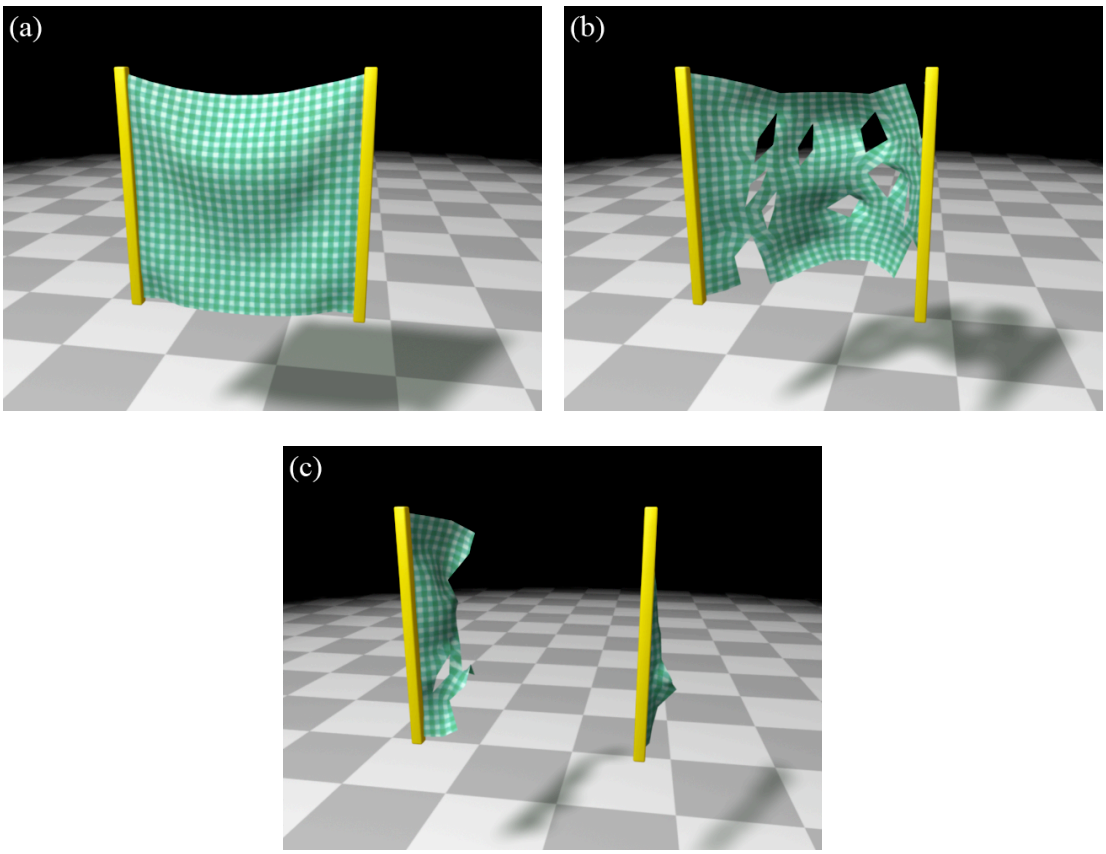


Fig. 46: Example 4: Wind induced tear.

(a) at rest, (b) several tears forming (c) post-tear.

Example 5 showcases the simulator's ability to handle plastic deformation (Fig. 47). The cloth is positioned parallel to the floor, constrained on two sides. One side is pushed inward to show the drape of the cloth before plastic deformation occurs (denoted by a red line). The cloth drapes in a uniform shape without any visible wrinkles. Then, the moving constraint is pulled outward until plastic deformation takes place. Returning to the original position, the cloth drapes much lower than before. There are visible stretch marks, suggesting that the plastic deformation was not uniformly distributed.

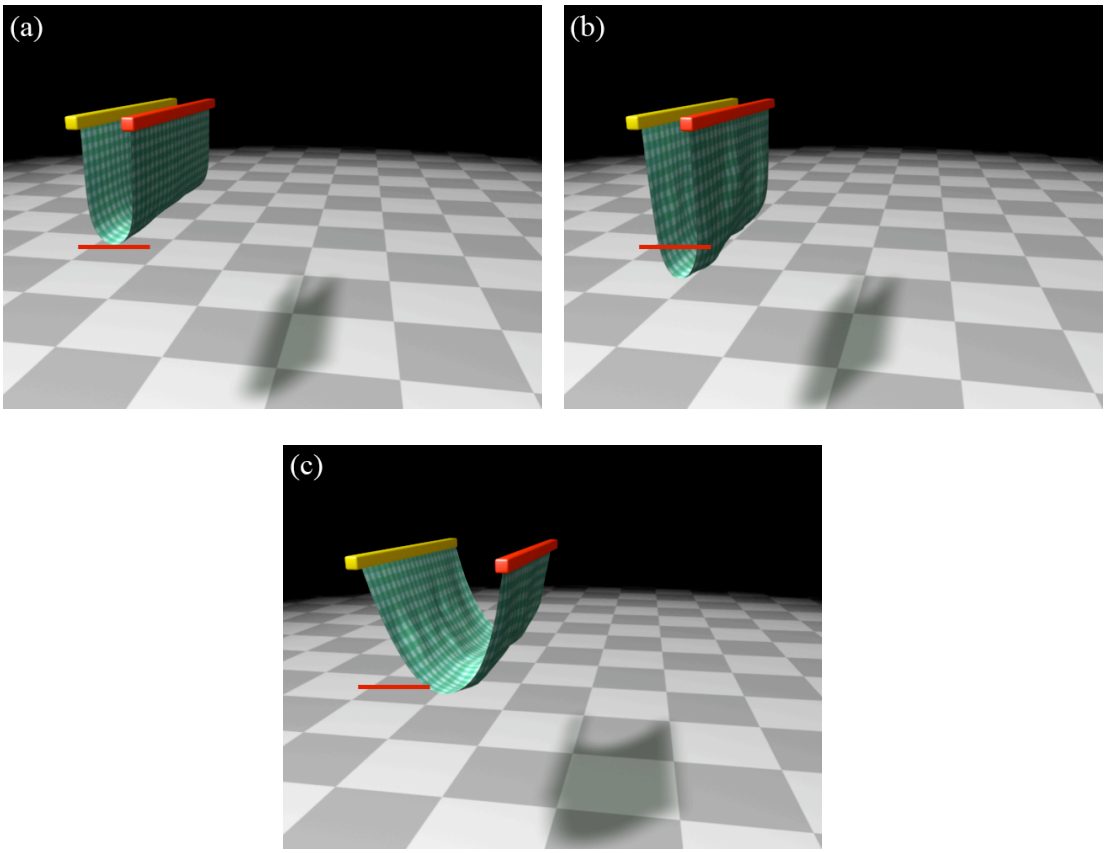


Fig. 47: Example 5: Plastic deformation.

(a) pre-deformed drape, (b) post-deformed drape (c) visible stretch marks.

The last rendered animation, Example 6, features the use of a strength map (Fig. 48). The configuration of the cloth and its constraints is similar to Example 1. Regardless of the strength map, the tear seems to originate at the same time and propagate at the same rate. Without the use of a strength map, the cloth tears near the center in a fairly straight vertical direction. When the strength map is used, it can be seen that the cloth tears in the predefined weak areas.

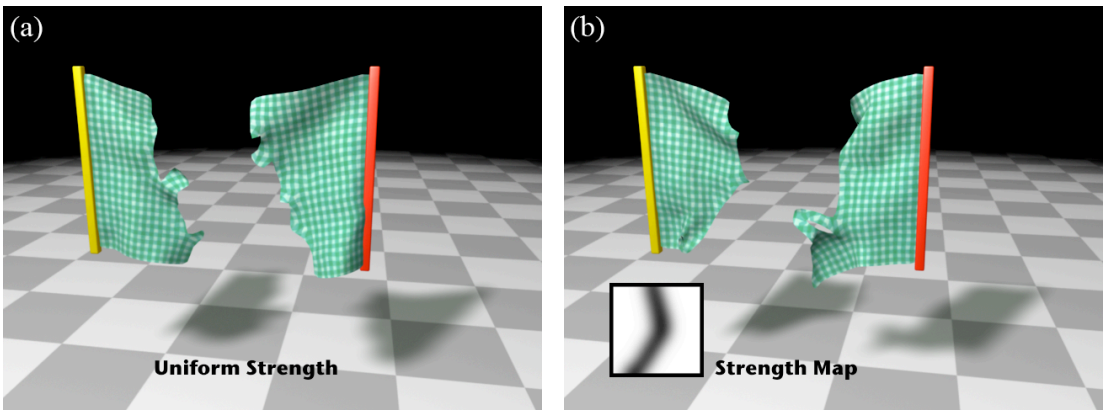


Fig. 48: Example 6: Effects of strength map.

(a) uniform strength, (b) strength modulated by strength map.

The final animation shows the interactive OpenGL simulator (Fig. 49). It features the data that was used to generate the animation in Example 3. The green vertices are fully constrained, and the red vertex is experiencing the most amount of strain. A short green line denotes the overall direction of the strain at that vertex. The

strain throughout the cloth is visualized by an increase in red color, making it easy to identify areas of high strain. It can be easily seen here that areas of high strain tear first.

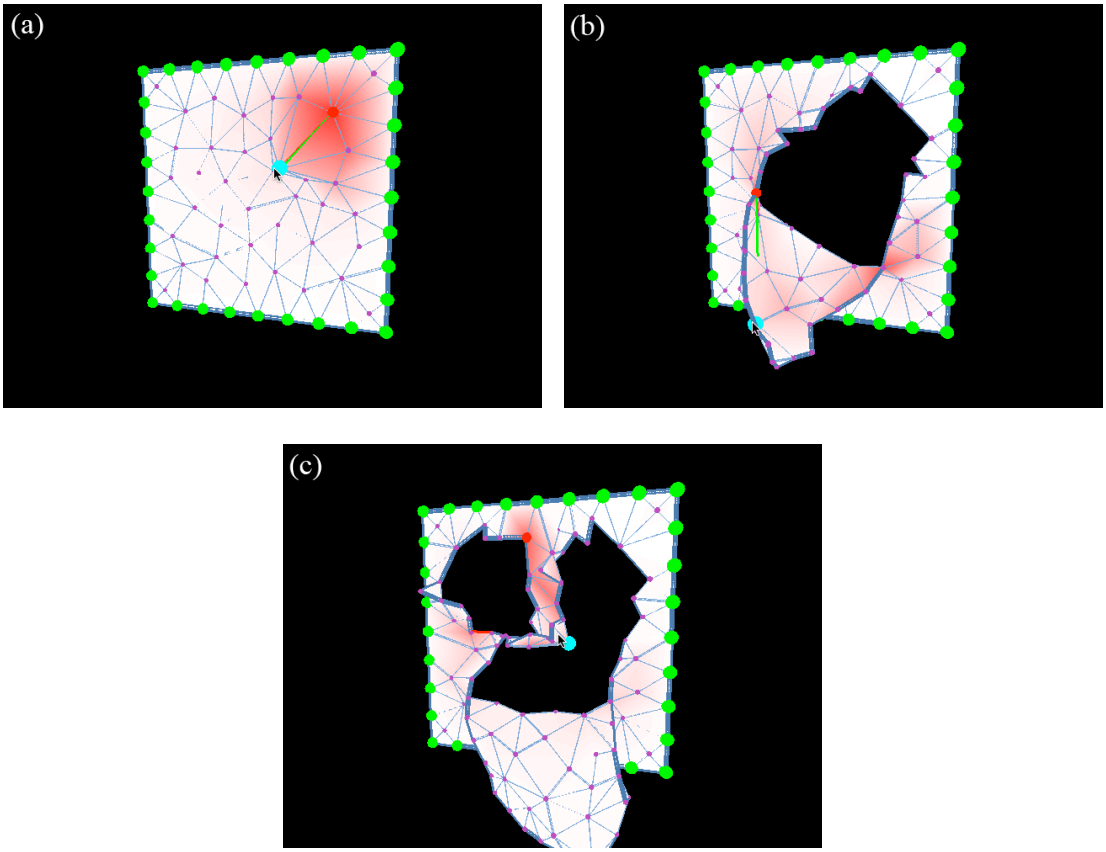


Fig. 49: Strain visualization.

(a) pulling on a vertex, (b) mid-tear, (c) tearing off an additional piece.

CHAPTER V

CONCLUSIONS AND FUTURE WORK

The speed of the cloth simulator can be greatly improved by using an implicit integration scheme to advance the cloth's state at a much faster rate. This would allow a user to interactively manipulate and tear a high-resolution cloth mesh in real time. In its most basic form, implicit integration requires solving a linear system:

$$\mathbf{M}\mathbf{x} = \mathbf{b} \tag{34}$$

where M is a matrix containing partial derivatives of forces, **x** is the changes in velocities that are being solved, and **b** is additional derivative data. Forming M and **b** requires a deep understanding of matrix calculus. A technical paper by Dean Macri helps to explain the required math [12]. Once M is constructed, it could simply be inverted to solve for **x**, but that is computationally expensive and defeats the purpose of using implicit integration for a speed gain. Luckily M is sparse, and there are methods available for solving such matrices, such as the Conjugate Gradient Method [20]. Unfortunately, these methods can be difficult to implement.

Another feature that could improve the simulator is the ability to handle collisions and friction. Brute force methods for collision detection are easy to implement, but are computationally expensive as described earlier. They were left out in this thesis because demonstration of the tearing algorithm can be effectively displayed with vertex constrains alone.

If implicit integration was used as well as collision handling, much more complex examples of tearing cloth would be possible. Clothed characters would be able to be practically simulated, resulting in animations featuring tearing off shirts, or a growing character that becomes too large for its clothing, such as The Incredible Hulk.

The implementation of these features could be avoided altogether if the tearing algorithm was implemented separately as a standalone package. This is possible because the algorithm primarily depends on measuring strain in the cloth mesh, and doesn't require much from the cloth simulator itself. Using the Maya API, a Maya plug-in could be developed for strain based tearing, whereas the cloth model (modeling, integration, collisions, etc) would be controlled by Maya's internal cloth simulator or a 3rd party simulator such as Syflex [21] or Qualoth [18].

The resulting animations supplied with this thesis lack motion blur. Motion blur is commonly into two types: 3D and 2D. 3D motion blur relies on interpolating the three-dimensional motion of an object. Several sub-frames are rendered and combined to produce a blur effect. My setup cannot use 3D motion blur due to a limitation in the importing of the cloth model sequence. Since each frame of the animation uses a different model, surface velocities are unknown and no interpolation can be made. One solution is to import one model and use the rest to build a library of blend-shapes, which creates a time continuous model that can be interpolated between frames. But, it is not possible to do this in Maya if the geometry of the mesh changes over time, which happens every time a tear occurs.

2D motion blur is a more viable solution. It depends on knowing the velocity of each pixel in the resulting two-dimensional image. A shader could be written which represents the three-dimensional velocity vector as a color, with the x component stored in the red channel, y in green, and z in blue. This image could then be used with a third party program, such as ReelSmart Motion Blur [19], to correctly blur each pixel.

Another improvement would be the addition of fringed edges where a tear occurs. Torn edge information in the cloth could be saved in a separate file while writing out the cloth model cache. This information could be used in Maya to create splines that are aligned with torn edges. Fur could be grown from these splines using Maya's Fur system. These frizzy edges would track with the animated model. They could even be dynamic, reacting to the cloth's motion.

REFERENCES

1. "Autodesk Maya," http://usa.autodesk.com. Accessed October 1, 2008.

2. Baraff, D. and Witkin, A. "Large Steps in Cloth Simulation," in *Proc. ACM SIGGRAPH '98*, pp. 43-54, 1998.

3. Baraff, D. and Witkin, A. "Physically Based Modeling: Online SIGGRAPH 2001 Course Notes," Pixar. http://www.pixar.com/companyinfo/research/pbm2001. Accessed October 1, 2008.

4. Bourke, P. "Object Files." Western Australian Supercomputer Program. http://local.wasp.uwa.edu.au/~pbourke/dataformats/obj. Accessed October 1, 2008.

5. Breen, D., House, D., and Wozny, M. "Predicting the Drape of Woven Cloth Using Interacting Particles," in *Proc. ACM SIGGRAPH '94*, pp. 365-372, 1994.

6. Bridson, R., Fedkiw, R., and Anderson, J. "Robust Treatment of Collisions, Contact and Friction for Cloth Animation," in *Proc. ACM SIGGRAPH '02*, pp. 594-603, 2002.

7. Choi, K. and Ko, H. "Stable but Responsive Cloth," in *Proc. ACM SIGGRAPH '02*, pp. 604-611, 2002.

8. *Hulk*. Dir. Ang Lee, Visual Effects by Industrial Light and Magic, 2003.

9. *The Incredible Hulk*. Dir. Louis Leterrier, Visual Effects by Rhythm and Hues Studios, 2008.

10. Kawabata, S. *The Standardization and Analysis of Hand Evaluation.* The Textile Machinery Society of Japan, Osaka, 1980.

11. Landau, L., and Lifshitz, E. *Theory of Elasticity.* Pergamon Press, London, UK, 1959.

12. Macri, D. "Real-Time Cloth," Intel Corporation, 2002.

13. Norton, A., Turk, G., Bacon, B., Gerth, J., and Sweeney, P. "Animation of Fracture by Physical Modeling," in *The Visual Computer*, Vol. 7, no. 4, pp. 210-219, July 1991.

14. O'Brien, J. and Hodgins, J. "Graphical Modeling and Animation of Brittle Fracture," in *Proc. ACM SIGGRAPH '99*, pp. 137-146, 1999.

15. O'Brien, J., Bargteil, A., and Hodgins, J. "Graphical Modeling and Animation of Ductile Fracture," in *Proc. ACM SIGGRAPH '02*, pp. 291-294, 2002.

16. "Pixar's Renderman," http://renderman.pixar.com. Accessed October 1, 2008.

17. Provot, X. "Deformation Constraints in a Mass-Spring Model to Describe Rigid Cloth Behavior," *Graphics Interface*, pp. 147-154, May 1995.

18. "Qualoth: The Quality Cloth Simulator," http://www.qualoth.com. Accessed October 1, 2008.

19. "ReelSmart Motion Blur," http://www.revisionfx.com/products/rsmb. Accessed October 1, 2008.

20. Shewchuk, J. "An Introduction to the Conjugate Gradient Method without the Agonizing Pain," Carnegie Mellon University, Tech. Rep. CMU-CS-TR-94-125, 1994.

21. "Syflex," http://www.syflex.biz. Accessed October 1, 2008.

22. Terzopolous, D., Platt, J., Barr, A., and Fleischer, K. "Elastically Deformable Models," in *Proc. ACM SIGGRAPH '87*, pp. 205-214, 1987.

23. Terzopolous, D. and Fleischer, K. "Modeling Inelastic Deformation: Viscoelasticity, Plasticity, Fracture," in *Proc. ACM SIGGRAPH '88*, pp. 269-278, 1988.

24. *Van Helsing*. Dir. Stephen Sommers, Visual Effects by Industrial Light and Magic, 2004.

25. Weil, J. "The Synthesis of Cloth Objects," in *Proc. ACM SIGGRAPH '86*, pp. 49-54, 1986.

VITA

Name:	Kurt Thompson Phillips
Address:	6911 Meadow Lake Ave., Dallas, TX 75214
Email Address:	kurtphillips@gmail.com
Website:	www.kurtphillips.com
Education:	B.E.D., Environmental Design with a Minor in Mathematics, Texas A&M University, 2003
	M.S., Visualization Sciences, Texas A&M University, 2008