Constraint Based Animation: A Cloth Simulation

Abstract

We use a constraint based approach to the simulation of a two dimensional particle system as a simulation of cloth. We test a minimum spanning tree method for the stabilization of the constraints.

1 Introduction

The main goal in this project is to simulate cloth, and along the ways look at the various elements that go into a physics based simulation. In particular, we wanted to simulate inextensible cloth as was done in [Goldenthal et al. 2007]. Hard constraints are a great way to make an inextensible object like cloth. We know that cloth does not stretch noticably, and thus the key to an accurate simulation comes from this. For this reason, constraint simulation is prefered, especially using hard constraints as opposed to spring penalty forces. A hard constraints simulation will give very tight restrictions on the deformation of the cloth.

Even with the choice of hard constraints, we still end up with deformations or perterbations from an accurate simulation that come from drift in the numerical integration. We will always have to deal with fixing this drift when using a constraint based approach. There are several ways to fix drift and the process is termed constraint stabilization. Familiar methods for constraint stabilization include Baumgarte and post step stabilization. In this work we present a new method for constraint stabilization, through the use of minimum spanning trees.

Since we our aim is a cloth simulation, the setup is quite simple. The cloth consists of an array of particles, each understood to have a mass, a velocity and forces that act upon it and these parameters are used to integrate the system. The structure of the cloth comes from constraints given between particles, which bind them together. Particles are constrained to be a fixed distance from their four neighbors (making a plus sign with the particle in the middle).

2 Related work

Cloth simulation is an active field of research. Integrating these kinds of systems in a way that is stable and efficient has been addressed in several different ways. Attempts at fast simulations as in [Baraff and Witkin 1998] highlight the complexity of the problem. Post step methods have been investigated in [Goldenthal et al. 2007] which are similar to our graph fix in that it is a step and project method. For an example of constraint based animation and a good introduction on integrating see [Witken and Baraff 2001]. For a nice description of minimum spanning trees, the algorithms of Prim and Kruskal as well as their computational complexity see [Cormen et al. 1999].

3 Background

4 Physics Based Animation

4.1 The Particle System

In the monolithic view, we lump all the particles together and form a large monolithic system. This state of the system is a phase space vector and is represented as an array of each particles' position in 3 dimensions long with each particles velocity in 3 dimensions. We then treat this larger vector as the mechanical system and integrate its motion.

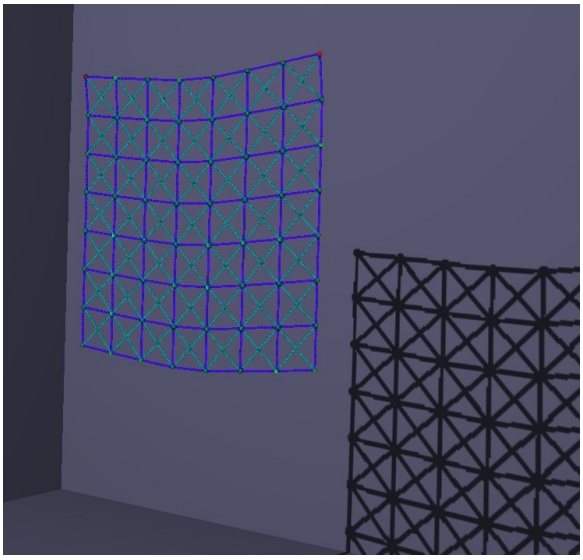

Figure 1: *The Cloth as a mesh of constraints and springs*

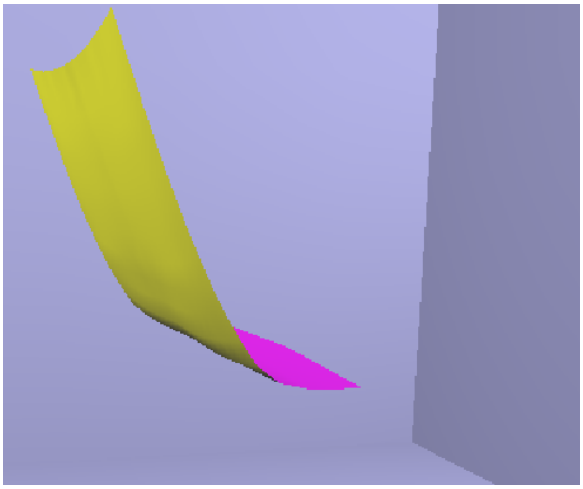

Figure 2: *A flowing cloth in simulation*

4.2 The Integrator

We used a symplectic Euler integration scheme for this simulation.

4.3 Constrained Mechanics

One can find the exact form of this system in the Pixar notes [Witken and Baraff 2001].

$$GM^{-1}G^T\lambda = -\dot{G}\dot{x} - GMF - k_{sc}g - k_{sd}\dot{g} \qquad (1)$$

4.3.1 Kinds of Constraints

Stretch A major ingredient in creating a general constrained mechanics system is just how one understands a constraint. Mathe-

matically, a constraint is an equation such as

$$\|\vec{x_1} - \vec{x_2}\| - l = 0 \tag{2}$$

which is a stretch constraint in Stam's [Stam 2009]. This states that the distance between two points $\vec{x_1}$ and $\vec{x_2}$ should be exactly of length l. The two "points" should refer to the positions of a particle.

Pinning One may have other constraints, such as pinning constraints. A pinning constraint will constrain a particle in a single dimension and would have the form

$$\vec{x_1}_x = 0 \tag{3}$$

This would fix the x coordinate of a particle to be exactly zero, and if forces acted on the particle, it would move, but only about the y-z plane.

4.4 Constraint Stabilization

When simulating a constrained system, we find that the state vector in position space will drift off of the constraint manifold. The prototypical example of this is a simple pendulum which oscillates but also slowly falls in the direction of gravity, eventually slowing down do to the lengthening of the pendulum. The constraint manifold for this system, a one particle system, is the circle along which we would normally expect the pendulum tip to move.

There are two methods commonly used to stabilize a constrained system, namely Baumgarte and post-step stabilization. The fast projection algorithm in [Goldenthal et al. 2007] is a form of post-step stabilization. It is an iterative algorithm that steps an unconstrained system back onto the constraint manifold.

Here we present a new method for constraint stabilization through the use of the minimum spanning tree. We can the constrained particle system can be seen as a graph, with particles as nodes and constraints as edges. Each edge can be weighted by the value by which the constraint is violated. Thus, we can find a spanning tree which maximizes the sum of these violations. Furthermore, onecan then fix only the constraints in the graph. The complexity of the most common algorithms used to find the minimum spanning trees is on the order of $O(V \log(V))$. After some refinement of the algorithms used in this project, we were able to bring the actual run times of the MST algorithm down to the theoretical limit.

Once computed the tree is then used to fix only those constraints in the tree. This is done simply by walking the graph from the root to each of the leaves. For each constraint we can move the child particle along the vector until the constraint is satisfied. We have two choices. Either we can just move each child particle to satisfy the current constraint as we walk the graph from root to leaves. Otherwise, we can store the vector displacement of the child particle, and cumulatively apply this change to each particle in the subtree. This graph fix technique would have the same complexity as tree traversal which is O(V) for a sheet.

4.4.1 Reconstruction

4.4.2 Error and Efficiency

We want to look at the errors due to drift. To do this we need a distance metric. We follow the method in [Kline and Pai 2003] which is to take the error as the largest absolute value over all the constraints in the constraint vector. We also look at an error metric that uses the constraint that has the largest as the norm of the error.

4.5 Velocities

4.6 Solving for Constraint forces

During constraint-based animation, we are computing forces which, if applied, exactly cancel those forces which might cause a particle to go where it is not supposed to go. For instance, imagine a rigid rod which is free only to pivot about one end. A particle at the free end of this rigid rod is constrained to be no farther from the pivot than the length of the rod. Thus, if the rod is rotating about the pivot, the particle at the end of the rod wants to fly off in the direction of its instantaneous velocity. A force, usually understood as a tension along the rod itself, must be applied to keep that particle moving along the expected trajectory.

It is precisely this force we want to compute during an animation if we wanted to simulate this kind of situation.

5 Implementation

5.1 Solving Linear Systems

To find the constraint forces, we need to solve the linear system 1. Thus we are trying to find λ such that

$$\lambda = (GM^{-1}G^T)^{-1}(-\dot{G}\dot{x} - GMF - k_{sc}g - k_{sd}\dot{g}) \tag{4}$$

which we recognize as being of the form $x = A^{-1}b$. To solve this linear system, we take advantage of the fact that the matrix $GM^{-1}G^T$ is sparse as well as symmetric and positive definite.

5.2 Conjugate Gradients

A system that is symmetric and positive definite can be solved using the method of conjugate gradients. An algorithm for conjugate gradients can be found in [Golub and Van Loan 1989]. Luckily, the matrix which we want to invert in constraint based animations is both symmetric and positive definite.

One of the goals of this project was to do a nice, fast linear solve. To achieve this, we avoid building the matrices and certainly avoid actually inverting any large dense matrix. This came back to the constraints class. This had to harmonize with the method we were going to use to solve the linear system and that method was conjugate gradients.

If we look closely at the algorithm, the matrix A in the linear system $Ax = b$, appears in two places.

$$\alpha_k = r_{k-1}^T r_{k-1} / p_k^T A p_k \tag{5}$$

and

$$r_k = r_{k-1} - \alpha_k A p_k \tag{6}$$

Each time it appears uninverted and as a part of a matrix vector multiplication. Recall that in our case $A = GM^{-1}G^T$. Thus, the conjugate gradients method could be programmed as long as we had some methods which could take a bunch of constraints, and a vector and compute the matrix vector computations. We use associativity so that we do the first matrix multiplication, then the second

$$(GM^{-1}G^T)x = (G(M^{-1}(G^T x))) \tag{7}$$

Thus, we only needed to create methods like *multiplyByG(aVector V)* and *multiplyByGTranspose(aVector V)* which spit out the resulting vector.

5.3 A Constraint Interface

Since the matrix G is sparse, we want to take advantage of this by avoiding the costly construction of the matrix itself. As in [Witken and Baraff 2001]

One of the tricks involved in the implementation of the constrained particle system was the definition of the constraint. The intent was to have objects which represented constraints. One might have chosen a constraint class and then subclasses could have been things like *Stretch* or *Bend* constraints. In our case, we defined a Java interface which outlined the kinds of methods a constraint might be responsible for. An object implementing the constraint then has local definitions for the methods.

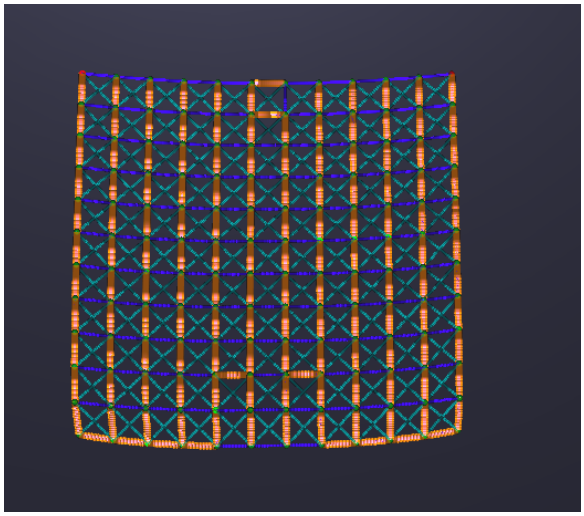

Figure 3: *At the lowest point of the arc, the vertical constraints are furthest from the set point*

6 Constraint Systems as Graphs

6.1 Constraint Systems as Graphs

We can see the sheet as a set of nodes and edges. The nodes are the particles and the edges are the constraints. Thus, the sheet can be seen as a planar graph.

6.2 Minimum Spanning Tree

Given some graph for a constrained particle system, one may find various sub graphs. One particular kind of subgraph is a tree. A subtree of a graph G, would be a graph with no cycles whose vertices are exactly the vertices of G and whose edges are some subset of the edges of G.

Suppose one could label the edges of G with real numbers. From this data, one could construct what is called the minimum spanning tree.

6.3 The Algorithms of Prim and Kruska

7 MST Constraint Stabilization

References

BARAFF, D., AND WITKIN, A. 1998. Large steps in cloth simulation. In *SIGGRAPH '98 Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pp 43 – 54.

CORMEN, T. H., LEISERSON, C. E., AND RIVEST, R. L. 1999. *Introduction to Algorithms*. The MIT Press, Cambridge, Massachusetts and London, England.

GOLDENTHAL, R., HARMON, D., FATTAL, R., BERCOVIER, M., AND GRINSPUN, E. 2007. Efficient simulation of inextenxible cloth. *ACM Transactions on Graphics 26*, 3 (July).

GOLUB, G. H., AND VAN LOAN, C. 1989. *Matrix Computations*, second ed. Johns Hopkins University Press, Baltimore and London.

KLINE, M. B., AND PAI, D. K. 2003. Post-stabilization for rigin body simulation with contact and constraints. In *Procedings of the IEEE International Conference on Robotics and Automation*, vol. 3, pp 3744–3751.

STAM, J. 2009. Nucleus: Towards a unified dynamics solver for computer graphics. In *2009 Conference Proceedings: IEEE International Conference on Computer-Aided Design and Computer Graphics*, 1–11.

WITKEN, A., AND BARAFF, D. 2001. Physically based modelling. Tech. rep., Pixar.

A Algorithms and Complexity

To compute a minimum spanning tree, there are two main algorithms Prim and Kruskal (cite the algorithms book). These algorithms have well known complexities which are functions of the number of nodes and edges in the graph. The complexity function for Kruskal is $\mathcal{O}(E \log V)$ and Prim is $\mathcal{O}(E + V \log V)$. We can compute their exact behaviour as a function of square sheet size since there is a function that takes the number of nodes in the sheet and produces the number of edges. One can compute this function, by looking at the Euler characteristic of a graph. This relates the vertices, V, edges, E, and faces,F, of a graph as follows

$$E = V + F - 2 \tag{8}$$

Since we can relate faces to vertices as

$$F = (\sqrt{V} - 1)^2 + 1 \tag{9}$$

We get a formula for edges as a function of vertices

$$\begin{aligned} E &= V + (\sqrt{V} - 1)^2 - 1 && (10)\\ &= 2(V - \sqrt{V}) && (11) \end{aligned}$$

We get the complexity functions

$$C_K = \mathcal{O}((2V - 2\sqrt{V})(\log V) \tag{12}$$

$$C_P = \mathcal{O}(2V - 2\sqrt{V} + V \log V) \tag{13}$$

At this point, we can compute which one increases faster as sheet size increases. To do this, we find the ratio of the two functions and then take the limit of this expression as the number of nodes goes to infinity.

$$\lim_{V \to \infty} = \lim_{V \to \infty} \log(V) = \tag{14}$$

This tells us that in this case, where we know that edges are a function of vertices, Kruskal performs worse than prim. In fact, if we look at the graph of this ratio, we can see that near zero, the rate of increase of the ratio is extremely steep, indicating that even near zero Kruskal is working much harder per node than Prim.

Of course, we are interested in only a range of values for the number of nodes which comprise our sheet. Specifically, we are interested in only that number of nodes that gives an animation which *looks* realistic and no more. That said, the graph of this ratio when looked at in the range of 0 to 40, which is the largest range we might consider will give hints as to which algorithm to use.

B Baumgarte Stabilization Experiments

The Baumgarte Stabilization method uses a spring damper force to attach the particle system to the constraint manifold. Thus, Baumgarte comes with two parameters which need to be tuned and these are the spring constant and the damping constant. Like ordinary spring parameters in a particle system, having very high stiffness can cause instability. In this section we look at an experiment performed which attempts to analyse the error performance of the system with varying Baumgarte stiffness. We look at a range of stiffnesses from small up to a point of clear integrator instability.