Draping Cloth on Meshes: Simple Yet Plausible

Lars Olsson & Samantha Lee
Rensselaer Polytechnic Institute

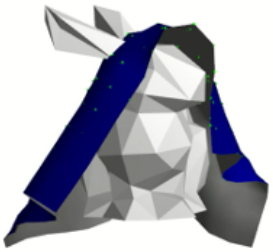

Figure 1: *Early result of cloth draped over bunny with too large a timestep, causing the ears to protrude.*

Abstract

Spring and mass cloth simulation has been explored in homework two of Professor Cutler's Advanced Computer Graphics Class. In this paper, we want to simulate draping cloth on meshes as simply as we can while still obtaining plausible results. To do this, we add collision checks, mainly for collisions of cloth with meshes and then secondarily for collisions of cloth with itself (self-collisions). Additionally, we apply spatial acceleration data structures to our cloths and meshes in order to speed up the collision checks. Using both provided scenes and some of our own, we have accumulated some information on how well these acceleration data structures work, as well as how various integration methods perform.

Keywords: spring mass cloth particles, collision detection, collision handling, octree, kdtree, integration methods.

1 Introduction

Collision detection is important for cloth simulations to look plausible. In this paper, we focus on collision detection of cloth onto a mesh, then also self collisions of the cloth. This builds on homework one, simplification and subdivision of meshes, and homework two, cloth and fluid simulation, and pulls the mesh and cloth portions into one framework. We chose this problem because we understood this part of the course best, have good cloth and mesh code (maybe as a result of the first reason), and then decided that this would be a feasible project in the time given. We are interested in implementing collision detection and handling. We are also interested in implementing spatial acceleration data structures to make it faster. In part this was motivated by our skepticism about how much speedup could occur when one must destroy and rebuild an acceleration data structure each time particles move, so we were naturally curious to test it out ourselves.

We will cover related previous work in section 2, then our core features in section 3. Discussion of results can be found in section 4, and then limitations and bugs, future work, and how we did our work together in sections 5, 6, & 7 respectively.

2 Prior Work

The topic of this paper is not new, at least in its component parts. Since we were basing this project off of the class' homework two code, we are using a spring-mass model for our cloth, where it is composed of particles in an even grid. This is fully fleshed out in the Provot 1995 paper [Provot 1995].

Acceleration structures have multiple uses, and collision detection is just one of them. One paper that follows this is Meagher's work, where octrees are introduced at length [Meagher 1980].

'Robust treatment of collisions, contact and friction for cloth animation' approaches the same problem as us - handling cloth draping on irregular meshes [Bridson et al. 2002]. However, it does this in a very complex and seemingly physically-accurate way, which we thought was infeasible to replicate given both our ability and the time provided. A particularly hard task would have been to modify our code to manipulate velocities/forces/impulses, where as we are usually adjusting positions directly (such as Provot correction).

'Collision and self-collision handling in cloth model dedicated to design garment', another paper by Provot, is similar to the previously mentioned paper in what it is attempting to solve [Provot 1997]. However it is older, and explains more of the basic components of the whole process of collisions, including both how to detect them and then resolve them. We found this paper to be the most helpful.

Finally, Matthew Loehr, a student in Professor Cutler's 2007 class, engaged in a similar project to this one [Loehr 2007]. The main difference between our project and his is that we are more interested in irregular mesh shapes, and his purely dealt with perfect spheres and similar constructs. Despite this, this prior work provided some useful clarification on details of collision detection.

3 Core Features

In this section, we will discuss our core features and some implementation details for plausibly draping cloth over set meshes, including cloth self intersections. We treat the meshes as immovable objects and simulate cloth falling onto the mesh then draping over the mesh. We also simulate cloths that have potential for self intersection.

3.1 Cloth and Mesh Models

Our cloth and mesh models were heavily based on Professor Cutler's provided homework code and our own work on the homeworks. The cloth model is a spring mass particle model with structural, shear, and flexion springs. The mesh is a triangular model

composed of vertices, edges, and triangles using a variant of the half-edge data structure. These models are different, yet we can abstract them to something similar to help with collision detection. The cloth can be thought of as a triangle mesh, where the vertices of the triangles are the mass particles and the edges are made up of two structural springs and one shear spring. In our method we systematically although arbitrarily define triangles in our cloth. Depending on how the cloth buckles, a particular patch of cloth might be divided into triangles in two different ways. We can imagine how this arbitrary definition (assuming only one of the two cases) may cause strange behaviors in corner cases, but our results are plausible and and show that this simplifying assumption we take is not too detrimental to the simulation.

3.2 Collision Detection

Done simply, naive collision detection will check every possible base structure against every other base structure for a running time of $O(n^2)$. Depending on how collision detection is done, this base structure could be vertices, edges, faces, or a mixture thereof. Following Provot's collision detection breaks down into two cases: edge-edge and vertex-face collision detection [Provot 1997]. Edge-edge collision requires finding the closest point between two edges, then determining if they are 'close' to each other. Vertex-face collision is a two step process. First calculate the plane that the face is on and check if the vertex is 'close' to the plane, then second, calculate if the vertex is inside the face or not. Our faces are always triangles, and we convert the vertex from its cartesian coordinates into its barycentric coordinates to determine if it is inside the triangle or not. This 'close' constant is variable and we chose ours using visual heuristics.

Our method only uses vertex-face collision detection and we get decently plausible results. There are some cases of corners where edge-edge collision detection would have helped. Additionally we do not do penetration detection, so if for some reason, for example too large of a timestep, the mesh has penetrated the cloth, it will stay that way. The assumption we make here is that timestep will be small enough that a point collision is detected before penetration. Our algorithm should then be able to handle (most) cases.

3.3 Collision Handling

While collision detection has a relatively canonical method, collision handling does not seem to. Provot's method differed from Bridson's method, and both of them were too complex or would just not integrate well into our current model of cloth and meshes and how movement is integrated. Instead of following any of these, in this paper we have created as simplistic a model as possible while retaining visual plausibility. In the previous section, we have shown how we can assume a similar triangulation structure to both the mesh and the cloth. Since we are only handling collisions between a vertex and a face, it is easy to tell where we should move the vertex: it should move away from the face. Usually, this direction can be determined by the normal of the face, which can be easily calculated. When the face is flipped around, moving along the normal will actually move the vertex closer; this usually results in penetration, which our model has little toleration for. This is particularly problematic because the cloth is flexible, and easily twisted around. Before fixing this, explosions in the simulating cloth self-intersection were common. The fix is simple though - just check the velocity of the face, and move away from that. This can be done by comparing angles; this is calculated by the dot product of the velocity with the normal.

Now that a direction has been established to move in, a decision has to be made about how far to move. A reasonable way to do this might be to check how close the vertex is to the face, and compute a ratio based on how far away the vertex needs to be in order to not be considered colliding. However, we opt for an even simpler approach. Instead, we just always move the colliding vertex away by a constant, which we calculate as a fraction (we use 1/16) of the minimum distance to be considered colliding. We then iterate the collision detection/response loop, until either all collisions have been resolved in that timestep or an arbitrary amount of iterations has gone by. We did not encounter much slowdown from implementing this versus just doing collision response once per timestep. With a sufficiently small fraction, the response will also move relatively smoothly.

You might notice that we have only mentioned how we move the vertex in any vertex-face collision. We have further simplified the model by making this the only action we ever take. Since our cloth models entirely relies on the vertices carrying information such as position, velocity, and acceleration, it is hard to adjust anything besides them. In addition, the faces of the cloth that we assume might not actually exist, so we thought it best not to mess with them when possible. This same reasoning is why we only handle the vertex-face case, and ignore the edge-edge case. Instead, we figure that the loop will eventually come around to the other side, and the points in the face will adjust if they also need to. This leads to an asymmetric response, but it doesn't appear to have greatly affected the simulation.

All of this has led to a very simple model for collision, where we loop through the vertices of the cloth, gather triangles of the surrounding geometry (both cloth and mesh possibly), and then perform constant adjustments in an appropriate direction determined by the triangle's normal if a collision has occurred. We are hard-pressed to imagine any simpler system for detecting and resolving collisions.

3.4 Spatial Acceleration Data Structures

To do collision detection in a smarter fashion, rather than checking every geometry element against every other geometry element, we will be selective about which base structures we check against. Using spatial data structures we can quickly select only the triangles that are within some range of the point we are checking. The Bridson paper uses oriented bounding box trees (OBB-trees), as do many other papers where collisions are being dealt with. However, the construction of an OBB-tree requires a relatively involved search through how the points should be split. Continuing in our search for simplicity, we have opted to only use octrees and kdtrees.

All of these acceleration data structures provide the same function: they allow quick queries of what geometry is in the vicinity of another piece of geometry. For our algorithm, we want to know what faces are in the neighbourhood of a vertex. Acceleration data structures work well when all the items that are stored can be compared to each other easily. In other words, we want to reduce faces to vertices. We do this by taking the centroid of the face, and using it for insertion into our trees. After having built these trees based on the centroids, it is easy to query for the neighbours of vertex, given one assumption. This assumption is that the faces being stored are small enough that the centroid is relatively close to all the corner vertices. If not, the vertex being checked might actually intersect at the corner, but not be close enough to the centroid for the acceleration data structure to detect them being in proximity. To make this assumption true, we use a simple subdivision method that does not move vertices to modify meshes that originally have large faces. An additional reason for subdividing large faces is face-face penetration patterns we get, even when all the vertices are correct in not penetrating.

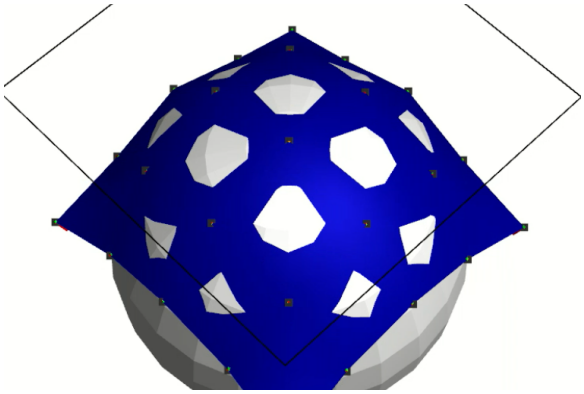

Figure 2: *Interference pattern produced by not subdividing large faces when draping a rough cloth over a rough sphere*

How do we actually get all the neighbours of a vertex? For octrees, we simply return all faces stored in the same cell as the vertex at an arbitrary depth, that we chose, in the tree. In our code, all models are scaled and translated into a cube from (-1,-1,-1) to (1,1,1). Therefore, we know the size of cells at any given depth in the tree. We chose a depth gives a cell size approximating our 'close' distance (what we consider the maximum colliding distance).

For kdtrees, we make a bounding box of radius of close distance around the vertex, and recurse through the tree accumulating all the faces in cells that intersect this bounding box. This method is more bullet-proof than the octree handling - in particular, the corner case of the query vertex being close to the edge of the cell is handled.

3.5 Multiple Integration Methods

A final feature of our code is that it handles more than just Euler integration. Specifically, it does Euler, Midpoint, Trapezoid, and Runge-Kutta integrations. Our code automatically reduces the timestep to help prevent explosions whenever any unusual (very large) movement occurs. We hope that these more accurate, though expensive, integration methods might allow the timestep to never be reduced. Overall, this might end up with the overall simulation being surprisingly cheaper.

4 Results

Here we test our method on various test cases: mostly draping cloth on meshes, and one to demonstrate cloth self collision handling. Each table is one test scene where the rows show different integration methods and the columns show differences from spatial data structures: none, octree, kdtree.

In general it is very clear that using a spatial acceleration data structure does exactly that - accelerates the process. As far as performance for octree compared to kdtree, there is some variation for which performs better. For integration methods, once again it varies as to what is better for what, without any apparent pattern. But it is apparent that in certain scenes, a more expensive integration calculation can be faster than having to run a cheaper integration method at a slower timestep.

Timing and images were all generated on a Lenovo Thinkpad T430s running Ubuntu 16.04 Linux, with an 3rd generation i7, and 16 Gb RAM. The unit of time is seconds.

Unless otherwise stated, the end result of each simulation was identical or very similar.

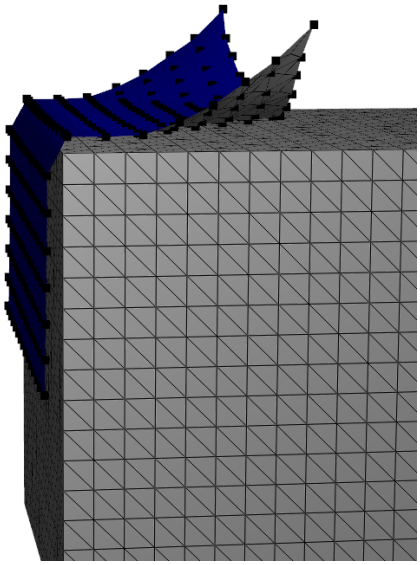

Figure 3: *End result of draping a cloth on a cube edge*

Table 1: *Cloth on cube edge. See Figure 3. [cube subdivided 4x] cube: 3072 triangles, cloth: 196 'triangles', *** went 30s w/o a single timestep advancement*

cloth on cube	no spatial ds	octree on mesh	kdtree on mesh
euler	***	64.30	48.89
midpoint	-	45.05	53.35
runge-kutta	-	69.52	82.92
trapezoid	-	37.93	44.69

5 Limitations & Bugs

We know we have memory problems - we made no effort to fix the destructors for our trees, though this does not seem to have had much impact on our code (typical examples run at around 150 megabytes of RAM used in a generous estimate). Our code is not the highest quality - calculations can definitely be cached to prevent recalculation, we have some duplicate and dead code, etc. All symptoms of a project with a deadline.

Besides these problems in our code, we don't really have any outstanding bugs or limitations that we haven't already mentioned in this paper. As with any cloth simulation, using too high of a timestep leads to explosion, and sometimes pretty patterns.

Care must be taken to make sure that the assumptions we have made in this paper are valid when setting up the scene. Are you checking the octree at the right level to match the 'close' distance? Do you have a plausible 'close' distance, and is the timestep small enough to detect collisions before anything penetrates anything else? These questions all have to be asked before simulating, which can be a bother.

6 Future Work

In the future, we would like to collect more data on integration methods and different acceleration data structures. We can use different acceleration methods for the cloth and the mesh, but have yet to come up with a good test scene for demonstrating that ability. Another dimension we have not captured data on is about the parameters for the kd- and octrees. Depth of the trees and minimum

Table 2: *Cloth on finer cube edge. [cube subdivided 6x] cube: 49152 triangles, cloth: 196 'triangles', *** went 30s w/o a single timestep advancement*

Cloth on finer cube edge	no spatial ds	octree on mesh	kdtree on mesh
euler	-	109.64	111.93

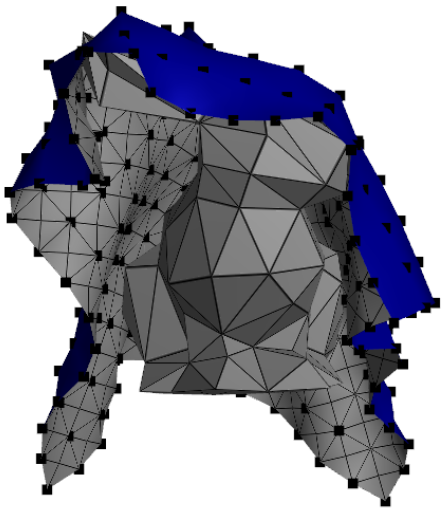

Figure 4: *Correct draping behaviour of cloth on a bunny. Note that the ears do not protrude (too much)*

number of nodes before a split are both constants.

Slightly more ambitious extensions would include implementing a variety of other acceleration data structures. In particular, axis-aligned bounding boxes (AABBs) and oriented bounding boxes (OBBs) seem to be in vogue among the papers we have read.

Finally, it is obvious that a model could be made that is more physically accurate. Extending this model to be anything like that would be very challenging, and would probably result in most of our code being thrown out. This might not be the best work to extend from ours though, as this was never our intention.

7 Work

We generally worked very closely together on everything. The vast majority of our work occurred when we were working side by side. A typical problem was approached by discussing it for a while, perhaps breaking off to read prior work, then coming back together when a solution seems apparent. After this, we would try to break down the problem into parts that both of us could implement simultaneously. If we instead were debugging a problem, we usually both set at it - fresh eyes find problems faster. That being said, we spent at least an hour debugging what turned out to be a missing set of braces.

Acknowledgements

Thanks Professor Cutler for the homework code.

Table 3: *cloth on bunny. See Figure 4. [no subdivision] bunny: 1000 triangles, cloth: 196 'triangles', ** quit at 5 min, halfway*

cloth on bunny	no spatial ds	octree on mesh	kdtree on mesh
euler	**	118.89	197.10

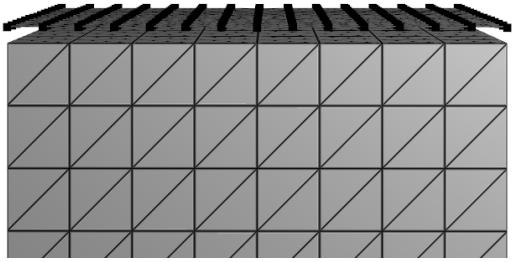

Figure 5: *Cloth at rest on a cube. Note the distance between - that is our 'close' distance*

References

BRIDSON, R., FEDKIW, R., AND ANDERSON, J. 2002. Robust treatment of collisions, contact and friction for cloth animation. *ACM Transactions on Graphics 21*, 3 (Oct.).

LOEHR, M. 2007. Cloth system extension: Collision volumes and self-intersection. *Advanced Computer Graphics Final Project*.

MEAGHER, D. J. 1980. Octree encoding: A new technique for the representation, manipulation and display of arbitrary 3-d objects by computer. *IPL-TR-80-111, Rensselaer Polytechnic Institute, Image Processing Lab.* (Oct.).

PROVOT, X. 1995. Deformation constraints in a mass-spring model to describe rigid cloth behavior. *Graphics Interface*, 147–154.

PROVOT, X. 1997. Collision and self-collision handling in cloth model dedicated to design garment. *Graphics Interface*, 17789.

Table 4: *cloth on cube. See Figure 5. [cube subdivided 3x] cube: 768 triangles, cloth: 196 'triangles'*

cloth on cube	no spatial ds	octree on mesh	kdtree on mesh
euler	53.92	1.53	0.93
midpoint	41.13	0.82	0.80
runge-kutta	41.06	1.29	1.62
trapezoid	40.96	0.63	0.70

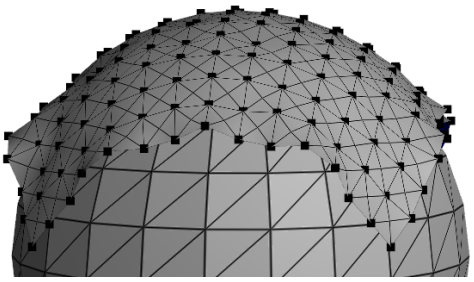

Figure 6: *Cloth draped on a sphere. Note the buckling behaviour of the cloth at the middle of its edges.*

Table 5: *cloth on top of sphere. See Figure 6. [no subdivision] sphere: 624 triangles, cloth: 196 'triangles'*

cloth on sphere	no spatial ds	octree on mesh	kdtree on mesh
euler	213.87	6.95	14.83
midpoint	206.75	6.89	16.85
runge-kutta	explodes	explodes	-
trapezoid	-	-	-

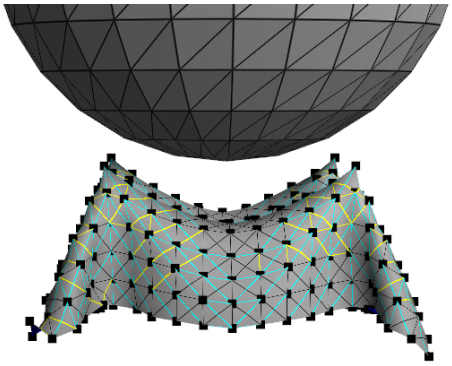

Figure 7: *Tablecloth without self-intersection*

Table 6: *tablecloth. See Figure 7. cloth: 400 'triangles', ***went 30s w/o a single timestep advancement*

tablecloth	no spatial ds	octree on cloth	kdtree on cloth
euler	***	73.92	70.57

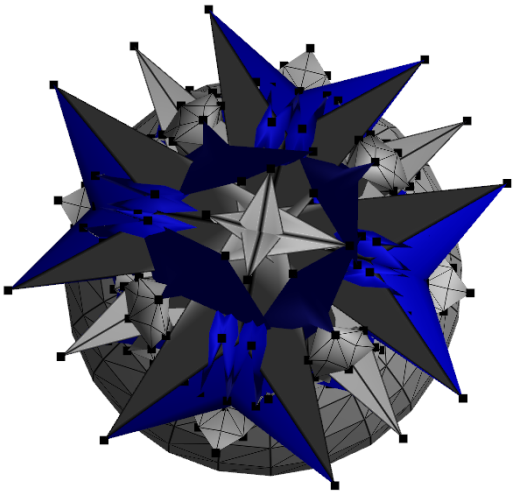

Figure 8: *An interesting explosion*

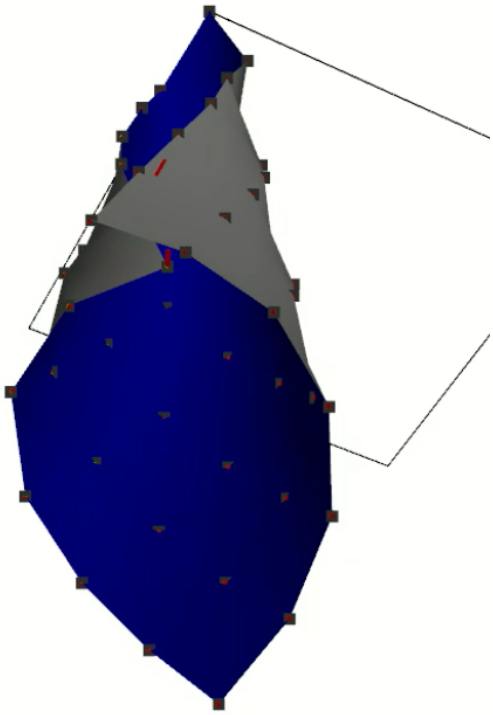

Figure 9: *Plausible result of curling for self intersection in a small cloth*