

CS 520 Final Project

Interactive Cloth Animation with Collision Detection

Tat Leung Chung
Chen Lan Yen

1.0 Introduction

In this project we've implemented a spring-mass system cloth using the simple blend, stretch and shear spring setup. The project is divided into 3 parts. The first part follows [1] to represent cloth as mass spring system. In the second parts, an integration method from [2] is employed to workaround problem in time step which must be inversely proportional to the stiffness. Although it is an approximated implicit integration scheme, the simulation is more suitable for real time. Finally implement our own collision detection scheme to perform real time cloth self-intersection and static model intersection.

2.0 Cloth Modeling

A cloth is represented as $n \times n$ particles in a rectangular grid. There are three different kinds of springs connecting between particles (see *Figure 1*). The structural spring connect each particle with it's immediately neighbors in the vertical and horizontal direction. The shear spring connect diagonal particles to preserve space between diagonal elements of the model. The flexion or blend spring connect every other cell alongside the structural springs. It applied a blend force to prevent folding too much.

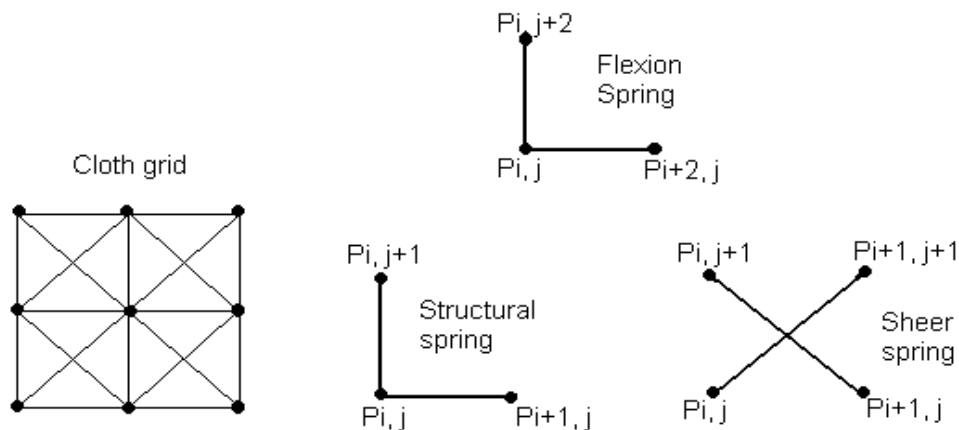


Figure 1: Mass spring model for cloth representation. Three kinds of spring are employed, structural, shear and blend/flexion spring.

The system uses the following standard force equations:

$$F_{\text{spring}} = (-k_{\text{stiffness}}(\text{natural_length} - |\mathbf{p}_i - \mathbf{p}_j|) - k_{\text{damp}}(v_i - v_j) \bullet (\mathbf{p}_i - \mathbf{p}_j) / |\mathbf{p}_i - \mathbf{p}_j|) \\ \times (\mathbf{p}_i - \mathbf{p}_j) / |\mathbf{p}_i - \mathbf{p}_j|$$

$$F_{\text{total}} = \text{mass} \times \text{gravity} + \sum F_{\text{structural}} + \sum F_{\text{shear}} + \sum F_{\text{blend}}$$

where \mathbf{p}_i , \mathbf{p}_j are particle position, v_i , v_j are particle velocities.

3.0 Integration scheme

Three different integration schemes are implemented: (i) Euler (ii) Midpoint and (iii) implicit predictor/corrector scheme (approximated implicit integration) in [2]. The implementation allows user is to switch between different schemes and observed the behavior at run time. We found that when using bigger time step, the system will explode (unstable) quickly with Euler scheme, it will also explode using midpoint scheme with slightly longer time. But there is no such problem with approximated implicit integration scheme.

3.1 Implicit predictor/corrector scheme

The simplest scheme for integrating a simulation system is the Euler equation:

$$\mathbf{v}_i^{n+1} = \mathbf{v}_i^n + \mathbf{F}_i^n \frac{dt}{m}$$

$$\mathbf{x}_i^{n+1} = \mathbf{x}_i^n + \mathbf{v}_i^{n+1} dt.$$

However, it has stability problem with larger time step. The idea of the implicit predictor/corrector scheme is used in our system. In this case the force at time t is replaced by the force at $n+1$, and the equation was:

$$\mathbf{v}_i^{n+1} = \mathbf{v}_i^n + \mathbf{F}_i^{n+1} \frac{dt}{m}$$

$$\mathbf{x}_i^{n+1} = \mathbf{x}_i^n + \mathbf{v}_i^{n+1} dt$$

We choose the implicit predictor [2] as a tool to integrate our cloth simulation because the function is efficient and stable. The difference between this and Euler/Midpoint scheme is that the force in implicit integration is split into two parts: a linear and a nonlinear term. Moreover, the Hessian matrix H is used to calculate a filtering matrix W and write H as an $n \times n$ matrix as follows: (n is the grid size)

$$\begin{cases} H_{ij} = k_{ij} & \text{if } i \neq j \\ H_{ii} = -\sum_{j \neq i} k_{ij} \end{cases}$$

After the Hessian matrix is computed, we use the following filter matrix W equation:

$$W = (I - (dt^2 / m) H)^{-1}$$

Note that the m is the average mass of each grid. We then have the approximate integration equation:

$$\Delta^{n+1} \mathbf{v} = \left(\mathbf{I} - \frac{dt^2}{m} H \right)^{-1} (\mathbf{F}^n + dt H \mathbf{v}^n) \frac{dt}{m}.$$

Since the integration scheme mentioned above doesn't preserve angular momentum - an important physical quantity, the resulting global torque is computed to compensate the factor:

$$\delta T = \sum_{i=1}^n (\mathbf{x}_G - \mathbf{x}_i) \wedge \mathbf{F}_i^{filtered}$$

$$\mathbf{F}_i^{correc} = (\mathbf{x}_G - \mathbf{x}_i) \wedge \delta T dt$$

where X_G is the gravity center. Accordingly, the new velocity and new position of each mass i in the system is updated.

3.2 Post-step modification of behavior

It is known that mass springs system is not a perfect model for cloth simulation. A post-step modification is employed to force the springs to keep away from being stretched too much. The approach used loops to pull every spring together along their axis, and the loop ended either it reached the index number or the error is limited under control. In our system, the default loop times were set to 30, and it worked better than the error limitation approach. Besides, positions of every spring are already computed in the step mentioned above. The resulting system looks more realistic (see *Figure 2*). *Figure 2b* is the result after the post-step modification, the red circle highlights where the differences are.

3.3 Wind Effect

External force effect from paper [12] is implemented to improve the reality of the system. The equation is:

$$\mathbf{F}_i^{drag} = K^{drag} \|\mathbf{v}_i^{wind}\| (\mathbf{n}_i \cdot \mathbf{v}_i^{wind}) \mathbf{n}_i.$$

$$\mathbf{n}_i = \frac{\sum_{j \text{ neighbors of } i} (\mathbf{x}_i - \mathbf{x}_j)}{\|\sum_{j \text{ neighbors of } i} (\mathbf{x}_i - \mathbf{x}_j)\|}$$

Which n_i is the normal for mass i and j is the neighbor next to i , K is a coefficient defined by user. In our case, the default K^{drag} is set to 0.8. V_i is the velocity of the external force which is the wind force in our system, this value is user adjustable.

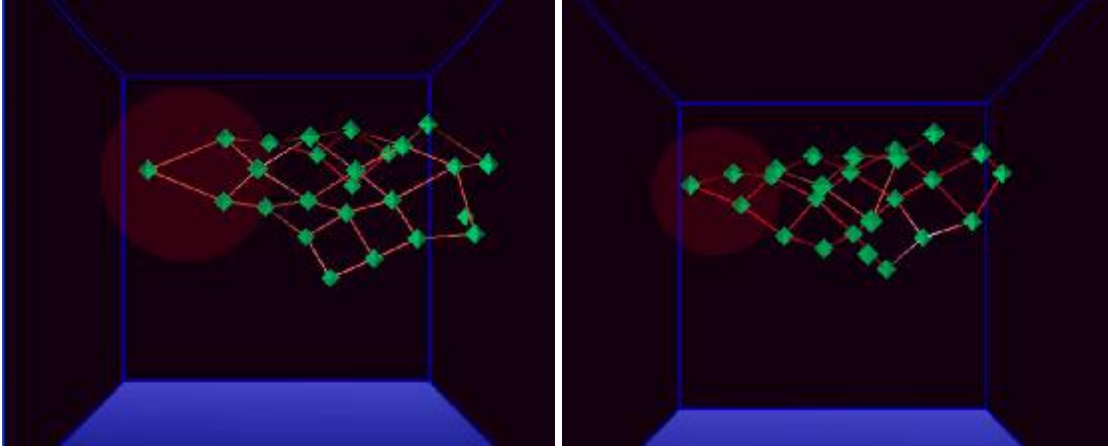


Figure 2: Post step modification in predictor/corrector scheme (a) before (b) after

4.0 Collision Detection

Given an $n \times n$ grid of mesh and static model consists of m triangles. It is not feasible to perform $O(n^4)$ cloth self-intersection test and $O(m \times n^2)$ cloth-model intersection test per frame in real time. A survey of collision algorithms suggests building a hierarchical data structure for the model. However, collision detection of deformable object like cloth is more challenge because the data structure needs to update for each moving triangle in real time. We prefer a data structure that can quickly update when triangle is moving.

It is observed that in cloth animation system the cloth triangles have the same natural size. Even when under stretch, the size may not differ too much from natural size due to resistant force of spring. We also make use of heuristic that neighboring triangles has blending and shearing spring between so it is safe to skip intersection check between them. It is also safe to assume that the space is bounded so we can devise a fast index scheme to map triangle into data structure. Finally there is abundant supply of memory nowadays so we can trade off memory for speed.

4.1 Our Algorithms

We use uniform space partitioning with cell size equal to the natural length of spring. Each vertex of triangle is index to the cell in constant time. Collision test is skipped for neighboring particles (i.e. particle with horizontal or vertical offset ≤ 1). For the case of mesh aligns with X/Y/Z axis initially this result in no two non-neighbor particles fall within the same cell. Hence expensive triangle-triangle intersection test is unnecessary in the beginning. To find which cells a triangle intersect, instead of using expensive cube-triangle intersection algorithm as done in voxels based approach [3], we can simply approximate by computing the extent of rectangular volume that vertices of triangle span and use all cells fall within. The 2D case is illustrated in *Figure 3*.

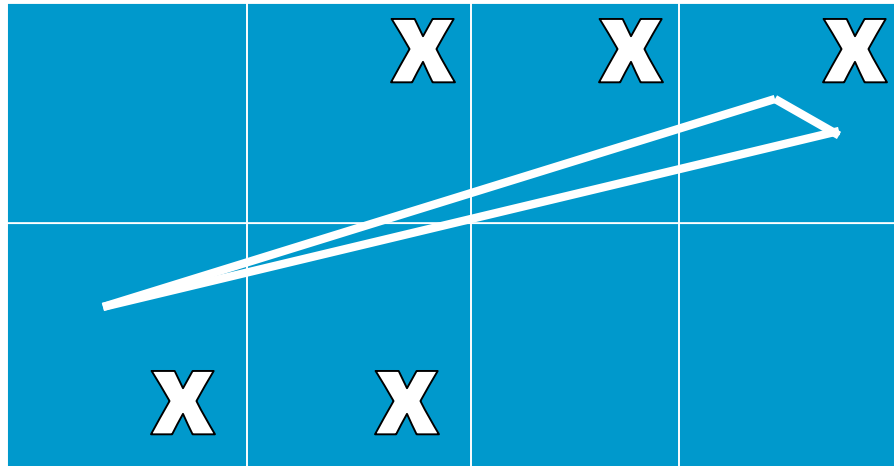


Figure 3: Finding cells that intersect a triangle. Triangle is inserted to all 8 cells as opposed to voxels based approach which only insert to cell mark with x.

Note that we skip cube-triangle intersection test in favor of adding triangle to cell that it may not intersect. This is actually not bad since at most 8 cells are involved for a triangle with natural spring length due to our choice of cell size. The chance of having 2 non-neighbor triangles in the same cell is not high if cloth is not in close proximity (within cell width) to each other. Besides, though we trade off cube-triangle intersection by triangle-triangle intersection we could eliminate some triangle-triangle test using a cache separating plane [4] (not currently implemented). The later is more efficient as temporal coherence is also exploited and result from [5] suggest with a 2 or 3 iteration most of non-collision triangle pair can be eliminated quickly. Note that both algorithms [4, 5] are applicable to the degenerate case of triangle. Thus most of the expensive triangle-triangle intersection test can be avoid.

Each cell keeps two arrays to store list of cloth triangles. For each frame, each triangle will insert to alternate array and the role is swap. This prevents linear time remove operation when triangle is deleted from cell. Triangle is inserted into cells in triangle index order and test with triangle exists in the same cell. A timestamp and last check index is keep for each triangle to prevent same intersect test perform multiple times when both triangles occur in more than one cells.

Any static model in the scene is preprocessed by inserting triangles that intersect cell into a static model array in cell. During runtime the cloth triangle will test with both cloth triangle array and static model array.

4.2 Collision Response

When there is a collision, binary search is used to find a time step with minimum number of triangle involves (typically less than 4). The instance just before collision is rendered. Triangles involve with collision is used to compute response. Since the approximate implicit method need to compute Hessian matrix if time step is different, we use Euler integration scheme in binary search to avoid computing the matrix inverse at run time. This is choosing instead of midpoint scheme due to its speed.

Ideally, we would love to compute a perfect response using the closest features pair line just before collision as normal. The new velocity is computed so that the angle between the normal is same as old velocity. If the particle in triangle involves more than one collision the average for each new velocity is computed. However, in practice the ideal response computed easily caused other particles to have collision in the next time step and the whole system will get stuck quite often. To avoid the problem, we simply set new velocity to friction times reverse old velocity for particles of triangle involved in a collision. This scheme seems works well mainly because there is no collision involved for those particles in previous time step. Nevertheless, in some case the whole system may still get stuck. This problem turns out to be the most challenging one we've encounter. Besides the system may take long time to reach stable state when collide with model with ten of thousands of triangles. Since binary search may suggest a tiny time step for the system to remain non-collision.

5.0 Implementation

Our cloth animation system (*Figure 4*) allows user to interactively pick a particle to manipulate cloth movement (*Figure 5*). It allows changing spring parameters (blend/shear/structural, stiffness/damping), friction, mass, time step, integration scheme, enabling/disabling gravity/collision, loading/clearing static model and switching to different texture while the system is running. For grid size and cloth initial orientation, the animation has to restart to take effect. Notice that we have to limit the acceleration (user adjustable), otherwise the system will be unstable. This is because we are using discrete integration scheme that may result in very large force. Information such as frame rate, time and number of triangle-triangle intersection test per frame are feedback to user.

We employed the polygon-cube code from [6] and triangle-triangle intersection code from [7]. The program has about 6800 line of code written in C++, OpenGL using GLUT and GLUI as interface, so it can port to any platform easily.

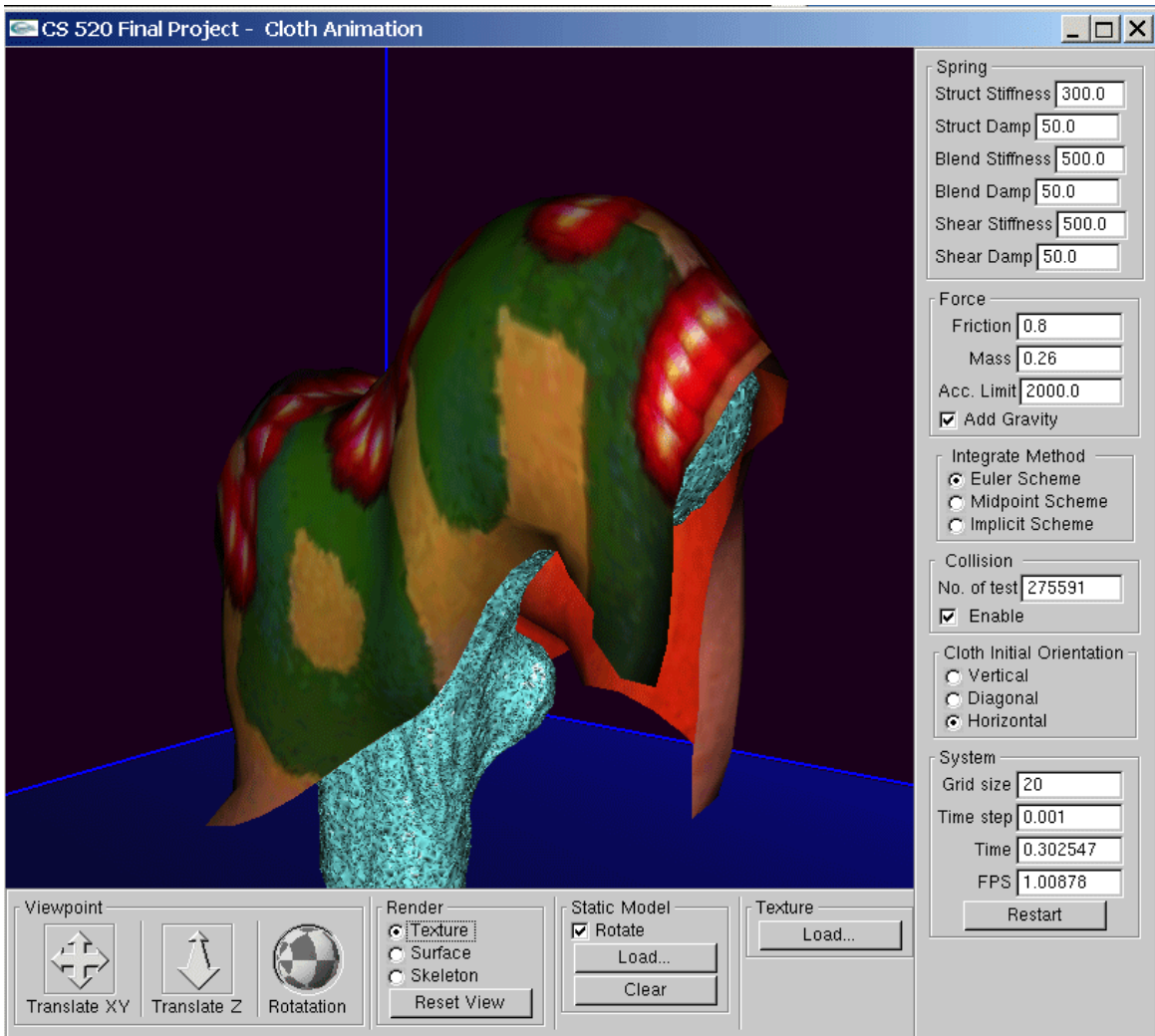


Figure 4: Our cloth animation system

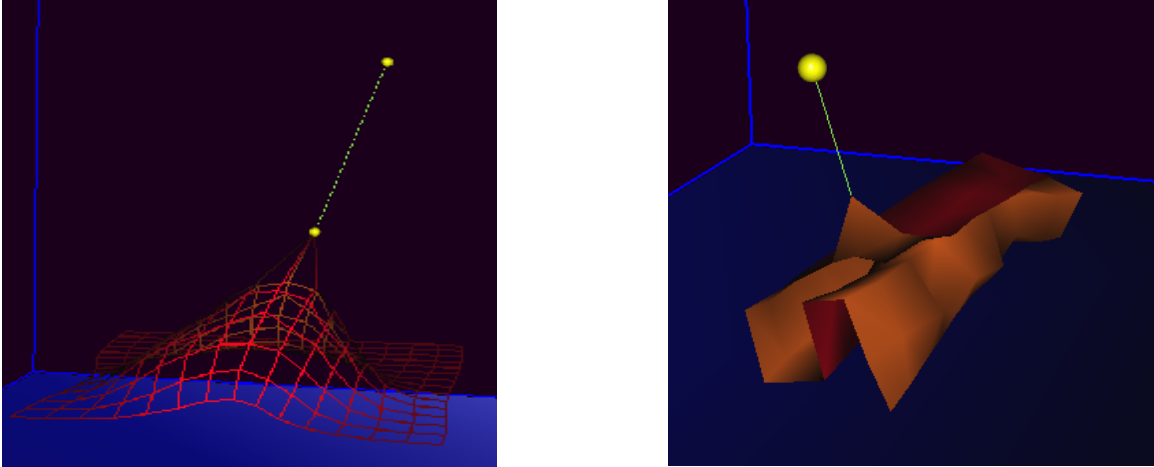


Figure 5: Interactive cloth manipulation

6.0 Results

6.1 Self-Intersection

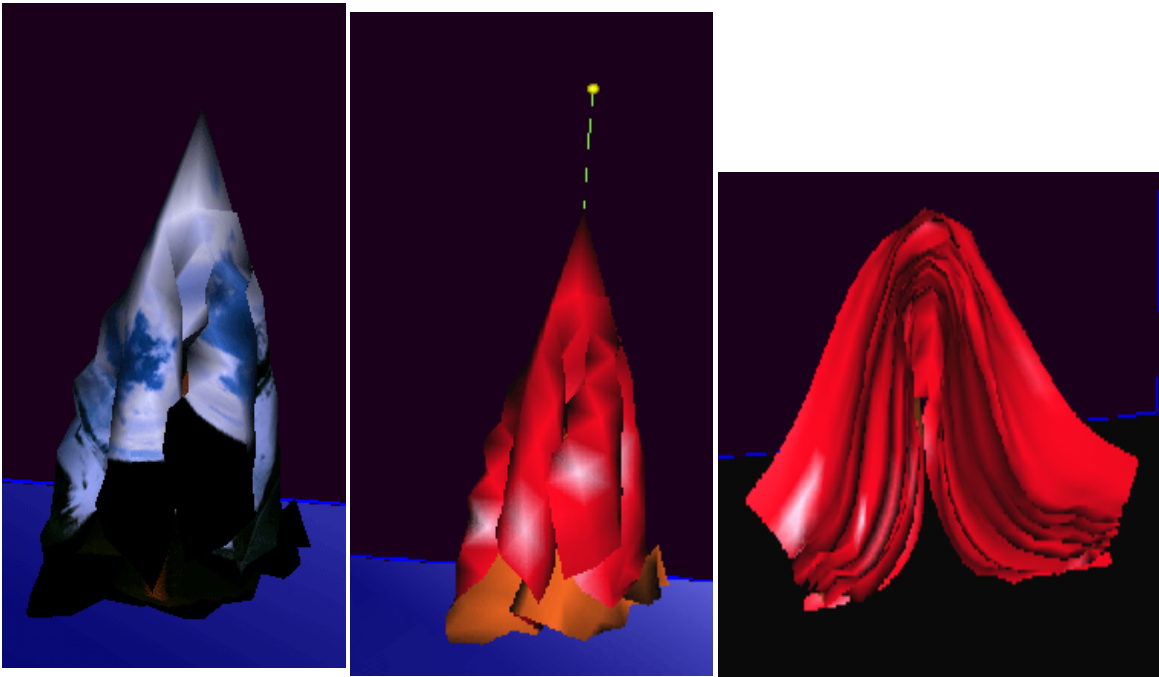


Figure 6: (a) Cloth hanging with texture (b) without texture (c) Cloth folding simulate with vertical cloth fall under gravity while user drag the middle point upwards.

6.2 Intersection with static model

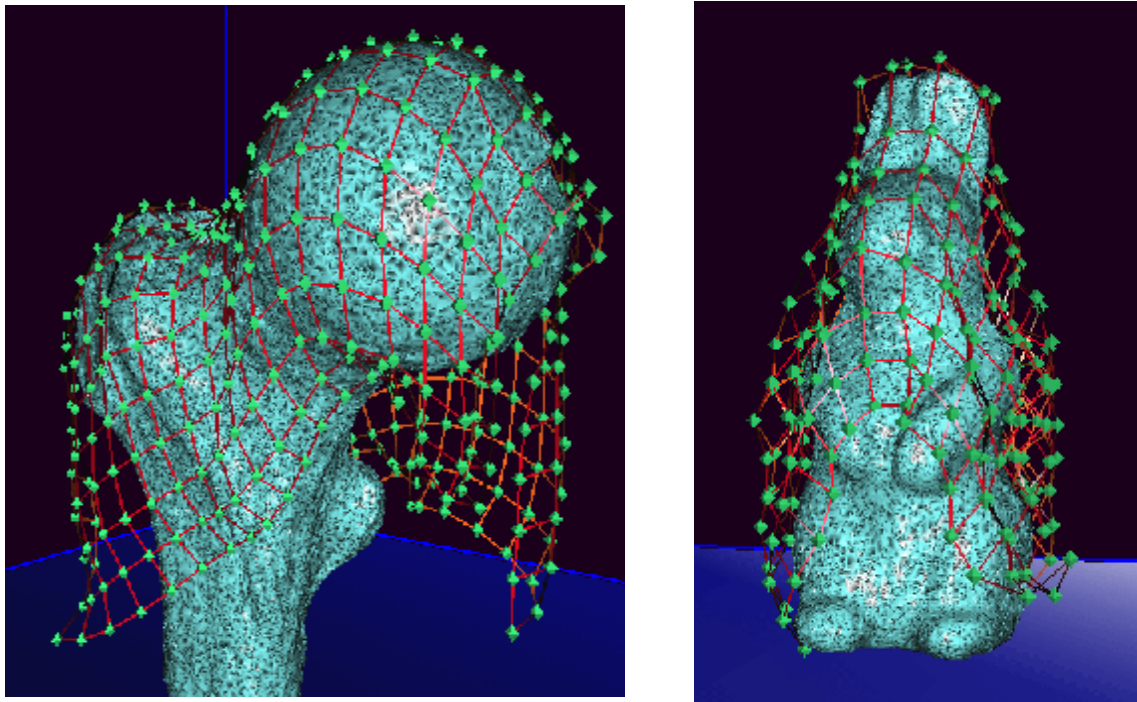


Figure 7 (a) Cloth covers a bone model (b) a rabbit model

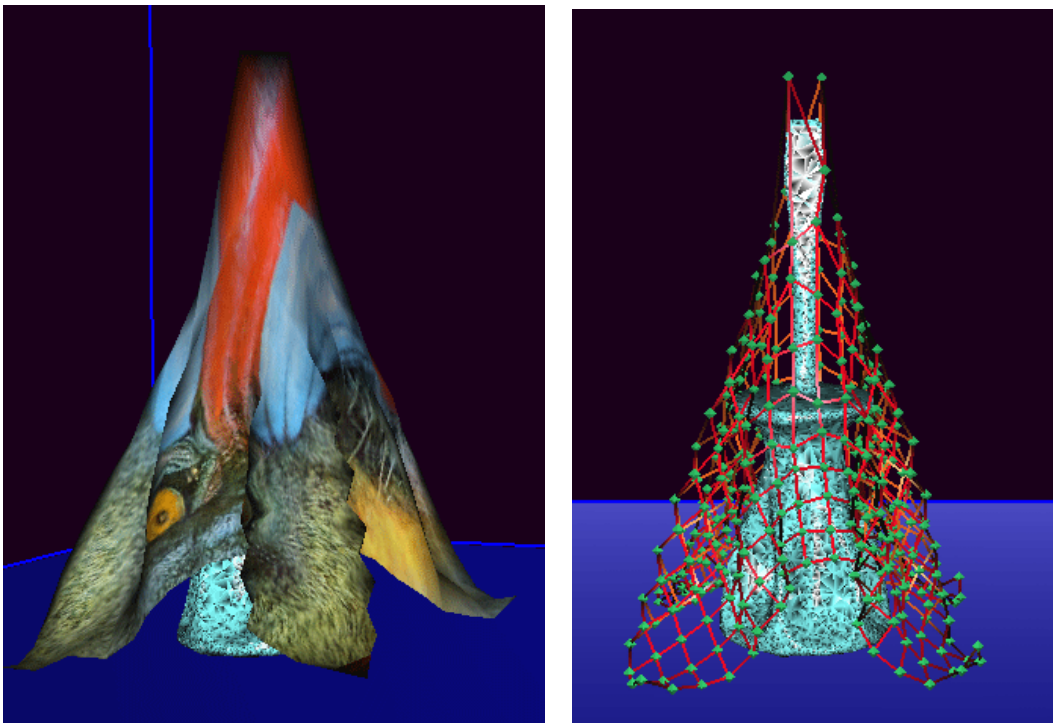


Figure 8 (a) Cloth covers a screw driver with texture (b) with skeleton

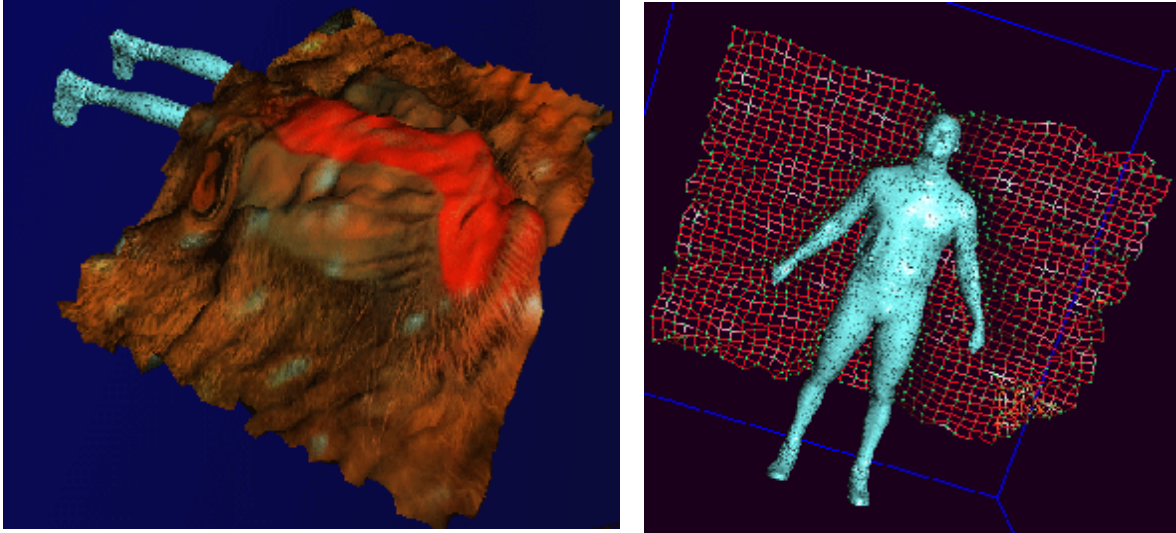


Figure 9 (a) Cloth covers a male with texture (b) with skeleton

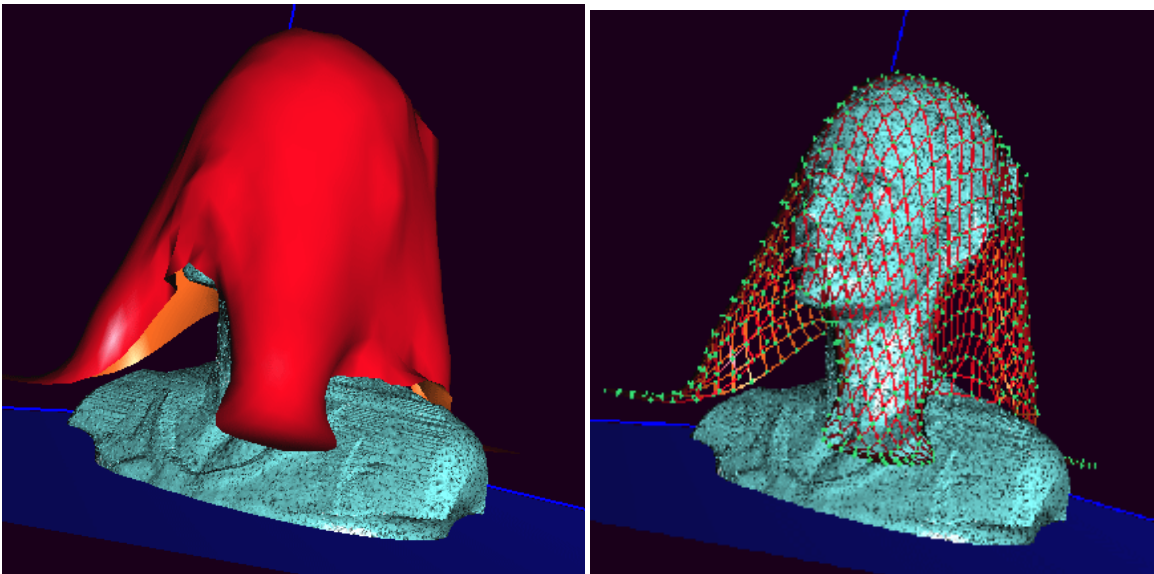


Figure 10 (a) Cloth covers a male with texture (b) with skeleton

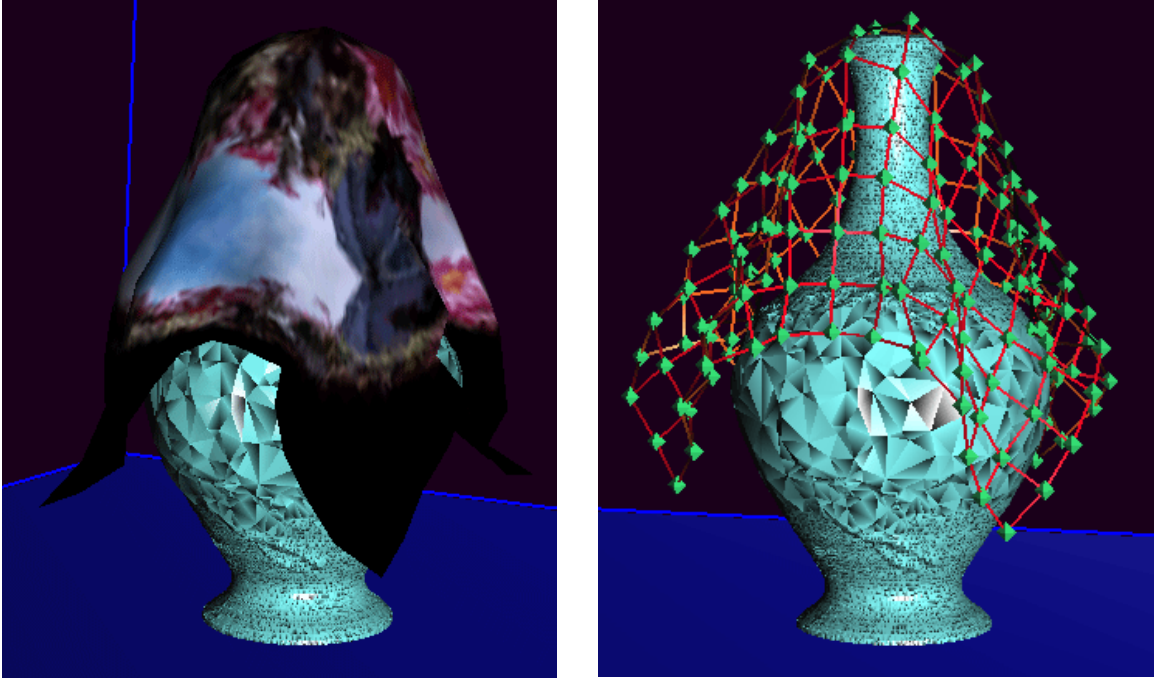


Figure 11 (a) Cloth covers a vase with texture (b) with skeleton

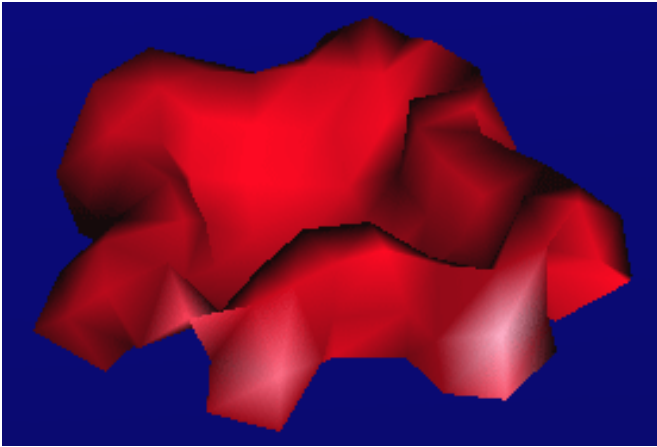


Figure 12: Cloth fall under gravity after the above vase disappear

7.0 Possible Improvement

One problem in current implementation is the time takes to achieve stable configuration when there is collision. When collide with static model with hundreds thousands of triangles it is very difficult to employ the user specified time step without getting into another collision. In most case binary search result in tiny time step to advance. The problem lies in the fact that we enforce all particles advance the same time step. If we allow particles far away from collision point to advance larger time step then particle close to collision point, we could have a more responsive system that stabilize faster.

Another possible improvement is to reduce triangle-triangle intersection test by implementation a cache separating plane algorithm as mention in Section 4.1. With both spatial and temporal coherence exploited this could dramatically increase the runtime performance. An added advantage of our proposed collision algorithm is that intersection test for each cell are performed independently. Thus we can partition cells into different group and do it in parallel on machine with multiple CPUs.

In our cloth model we compute vertex normal by first calculate plane normal using cross product, then averaging (weighted by area) the plane normal around the vertex. This per frame operation can be performed using current GPU to free up CPU time for other task. In fact, with the latest GeForce 6 graphics card, infinite long vertex/fragment program is now feasible. We can consider moving integration scheme or even collision code to faster GPU. Actually Nvidia already has cloth animation code using fragment program [8] available on web.

8.0 Conclusions

In this project we have implement an interactive cloth animation system. A naïve collision algorithm is proposed to perform collision for deformable cloth mesh. Experiment shown that it can detect self-intersection in interactive rate. We also proposed possible extension of our system for future research.

9.0 References

- [1] Xavier Provot. Deformation Constraints in a Mass-Spring Model to Describe Rigid Cloth Behavior, *Graphics Interface 1995*.
- [2] Mathieu Desbrun, Peter Schroder, Alan Barr. Interactive animation of structured deformable objects, *Graphics Interface 1999*.
- [3] Dongliang Zhang, Matthew M. F. Yuen. A Coherence-based Collision Detection Method for Dressed Human Simulation, *Computer Graphics forum, Volume 21 2002*.
- [4] Rich Rabbitz. Fast collision detection of moving convex polyhedra, *Graphics Gem IV, 1990*.
- [5] Kelvin Chung, Wenping Wang. Quick Collision Detection of Polytopes in Virtual Environments, *ACM Symposium on Virtual Reality Software and Technology, 1996*.
- [6] Green, Daniel, and Hatch, Don, Fast Polygon-Cube Intersection Testing, *Graphics Gems V, 1994*.
- [7] Philippe Guigue and Olivier Devillers. Fast and robust triangle-triangle overlap test using orientation predicates, *Graphics Tools 8(1):25-42, 2003*
- [8] http://developer.nvidia.com/object/demo_cloth_simulation.html
- [9] Matthias Wloka. Interactive cloth simulation,
http://developer.nvidia.com/object/gdc2001_clothsim.html
- [10] An Introduction to Physical Based Modeling from SIGGRAPH
<http://www-2.cs.cmu.edu/afs/cs/user/baraff/www/pbm/pbm.html>
- [11] Jeff Lander. Devil in the Blue Faceted Dress: Real-time Cloth Animation, *Game Developer Magazine, May 1999*.
- [12] Mark Meyer, Gilles Debunne, Mathieu Desbrun and Alan H. Barr. Interactive Animation of Cloth-like Object in Virtual Reality, *The journal of Visualization and Computer Animation, 2001*.