Dynamic Subdivision of Mass-Spring Cloth Simulations

Eleanor Olson
olsonk@rpi.edu
eleanorkolson@gmail.com
Rensselaer Polytechnic Institute
Troy, New York, USA

Kana Rudnick
rudnic2@rpi.edu
Rensselaer Polytechnic Institute
Troy, New York, USA

ABSTRACT

Representing cloth as a network of masses and springs is a common and effective method of simulating its behaviour. Traditionally, this network is uniformly dense across the surface; however, finding the ideal density for a given simulation can be challenging. This paper builds on previous work to automatically increase the density of mass-spring cloth simulations where necessary, simplifying the simulation process and increasing performance when compared with uniformly high-density simulations.

KEYWORDS

Cloth animation, mass spring simulation, computer rendering

1 INTRODUCTION

1.1 Background

The simulation of cloth is a deeply studied topic in computer graphics. With the frequency in which different cloth-like structures are seen in animation, a need to develop a system that allows for a believable visualization of the natural movement patterns of cloth is essential. Techniques like manual keyframe animation require too much effort on the part of the animator, and doing such manual editing for all cloths found in the decorations and backgrounds of scenes would take too much time. Therefore, attempts to simulate the physical behavior of cloth have been widely developed over time.

One of the most proficient methods for simulating cloth behavior has come from abstracting the cloth as a grid of suspended masses connected using springs. This model allows for the simulation of real forces acting on these masses, with the benefit of simplifying the representation to simple spring mass interactions. [Provot 2001]

This method of simulation functions well, but only if the given network of masses is sufficiently subdivided. This leads to an inevitable trade-off.

Too few masses will potentially lead to inaccurate simulation, especially around poles, corners, fixed points, and other sensitive or complex regions in the mesh. Too many masses, in contrast, will lead to a more accurate simulation at the cost of leading to a dramatic increase in the amount of computational effort needed to perform the simulation.

This paper proposes a method of simulation that finds a balance between computation time and accuracy. We propose a system of dynamically subdividing the mesh during simulation time around complex regions while not subdividing other regions. This allows for an increase in the precision of the simulation at these complex regions while keeping the overall granularity of the mesh, and therefore the computational time needed to simulate the movement of the mesh, low.

1.2 Preview

We begin in section 2 with an overview of related work and methods that inspired our work. We then describe the details of our representation of the mesh, and how we subsequently simulate that representation in section 3. In section 4, we describe the algorithm for subdividing a portion of the mesh as well as the criteria used to determine when subdivision is necessary. Finally, we discuss the results of our method as well as future work and improvements that can be made.

2 RELATED WORK

2.1 Mass Spring Cloth Simulation

One of the largest sources of inspiration for this work is that of Provot [Provot 2001]. He defines the system of mass-spring discretizations that we use as the basis for our representation of the mesh during simulation. This mass-spring representation allows for the simulation to only be dependent on simulating the motion of and forces acting on every particle independently, rather than having to accurately model the entire cloth as a whole. The paper also describes a simple yet effective system for managing inaccurate "super-elastic" deformations in the simulation, allowing for a successfully convincing simulation without requiring significant computational costs. Provot's methods work very well for most basic applications, and are rather simple to implement.

Due to this simplicity, there are aspects of the method that can be improved. For a grid of n particles along the x-axis and m particles along the y-axis, a subdivision that transforms every quad in the mesh into four quads results in a grid of particles $2n$ by $2m$. This means that using Provot's method, a single level of subdivision results in four times as many particles that must be simulated at every timestep: a dramatic increase in computational time.

2.2 Subdividing Cloth Simulation

Previous work has been done in running cloth simulations with dynamic subdivision. One such work that served as the primary inspiration for our work is that of Hutchinson, Preston, and Hewitt [Hutchinson et al. 1996]. The authors propose a method for simulating a cloth that has varying levels of subdivision throughout the mesh. This is accomplished through the use of a data structure that represents the full network, with each mass keeping track of what level of subdivision the mass is located on. They also propose a simple method of using the angle between neighboring masses to determine whether a point must undergo subdivision before simulation can be performed.

The technique, while simple, is effective in most cases. However, it lacks in a few areas, most notably in that due to the subdivision detection algorithm relying on neighboring masses, it fails to be effective at detecting if points that lack enough neighbors, including points on edges or points within a mesh that has not been subdivided at all in a given region. We look to improve upon this in our implementation, allowing for a thorough detection of when subdivision is necessary in all situations.

3 METHOD OF SIMULATION

3.1 Representing Cloth in Data

The underlying model of our simulation is directly built on the work described in [Hutchinson et al. 1996]. We construct a 2 dimensional grid of indices, each containing one of the following types of mass particles:

- Active: A particle that is actively simulated and may move freely
- Fixed: A particle that exists and contributes spring forces to adjacent particles, however may not itself move in space.
- Interpolated: A particle that is not actively simulated, and instead interpolates its position between neighboring particles. These particles also contribute spring forces to adjacent particles.
- None: No particle exists at this index. This occurs when one section of the cloth is subdivided, and therefore requires other sections to have "gaps" in the underlying representation.

All of these particles, excluding "None" particles, contain their current position and velocity, as well as their subdivision "layer". Particles created at the beginning of simulation begin on layer 0, and any subdivision increases a particle's layer by 1, as well as creating particles of that same higher layer around it.

3.2 Theory of Simulation

As in [Hutchinson et al. 1996], we begin with an input cloth of uniform, ideally low, mass density. To simulate this mesh without any subdivision, we follow a process identical to that laid out in [Provot 2001]. This entails iterating over each particle in the mesh and calculating the forces on it by adjacent springs, as well as any outside forces such as gravity. These forces are then integrated via explicit Euler integration, updating each particle's velocity, which is in turn explicitly integrated to update the particle's position.

To simulate a mesh with arbitrary subdivision, we follow a similar approach. We once again iterate over all possible indices in the graph, however not all indices will have existing particles. Therefore, we ignore all particles of type "None". Additionally, since the force on any given "Fixed" or "Interpolated" particle will not affect its position, these too may be skipped over.

To calculate the forces affecting a particle, we must find all springs connected to it. There are 3 different forms of springs in our simulation:

- Structural: Directly adjacent in 1 axis ($x \pm 1$ or $y \pm 1$)
- Shear: Adjacent along the diagonal ($x, y \pm 1$)
- Bend: 1 particle away in 1 axis($x \pm 2$ or $y \pm 2$)

These spring types can also be seen in Figure 1.

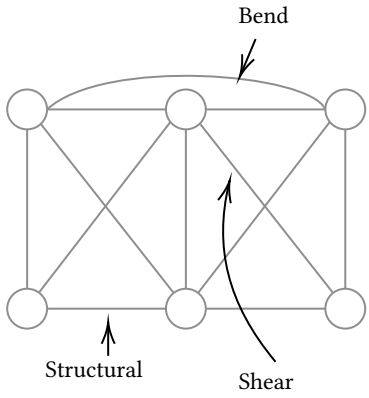

Figure 1: Example of different types of springs in the simulation.

```
x, y = dimension of cloth
m = maximum subdivision layer in cloth
for t in 0..2^m, i in 0..x, j in 0..y:
        p = particle(i,j)
        if p.type != Active:
            continue
        if p.layer < t:
                forces = forces(p)
                p.velocity += forces*timestep
                p.position += velocity*timestep
                p.position = correct(p.position)
```

Figure 2: Pseudocode of Cloth Simulation

The distance between springs, however, must be scaled depending on the layer l of the particle. Given a particle at layer 0, if the underlying mesh can represent 1 extra subdivision we know that we must skip over 1 index to get to the next particle of layer 0. This can be generalized for all layers as the following: if we take m as the maximum subdivision our underlying data structure can represent, we know that a particle on the same layer l will be 2^{m-l} indices offset. Therefore, we scale all spring distances by this factor.

In addition to properly finding adjacent springs in the mesh, the key insight from [Hutchinson et al. 1996] to simulate areas of differing subdivision smoothly is to simulate them multiple times per animation step. We recalculate forces, velocities, and positions for each particle 2^l times, where l is the particle's layer. Finally, we adjust particle positions to be maximally extended by an input parameter, avoiding hyperextended springs using the techniques from [Provot 2001]. Overall, this simulation takes the form shown in Figure 2.

3.3 Rendering

In traditional, uniform mass-spring simulation, it is trivial to construct a quad by connecting to the points directly adjacent. This step is required to pass our cloth into a rendering pipeline and

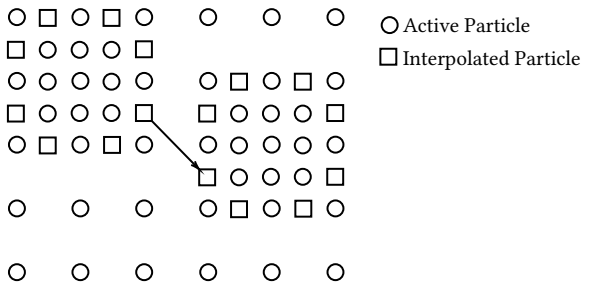

Figure 3: Example of interpolated particles finding a diagonal to create extraneous quads in the rendered mesh. This needs to be prevented to accurately create the rendered quad mesh from an underlying cloth mesh.

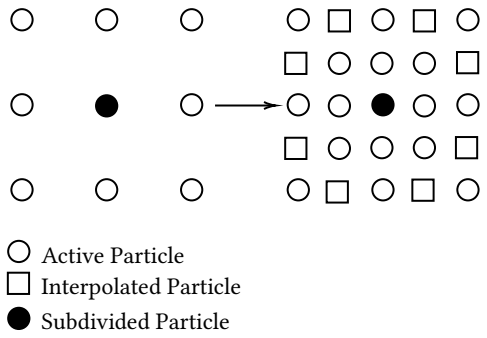

Figure 4: Subdividing around a point in the cloth mesh. In this case maximum subdivision $m = 1$, and the new subdivided particles have $l = 1$.

display it to the screen. In a subdivided cloth, however, this step becomes nontrivial.

To generate these quads, we begin by iterating over our mesh. When we find a particle that is not of type "None", we know that this will be the corner of one of our mesh's quads. Further, due to our subdivision scheme, discussed further in Section 4.1, we know that our cloth mesh will always decompose into square quads of the underlying mesh. That is to say, that each quad will have particles at indices $(i, j), (i+k, j), (i, j+k), (i+k, j+k)$, where (i, j) is our initial point, and k is some constant. To find this k, we check each index in the cloth mesh $(i + k, j + k), 0 \leq k$, until we find a particle not of type "None". We then pass all four of these particles' positions into the renderer. Finally, we skip our iteration to next check point $(i, j + k)$, the particle ending the quad on our iterated row.

While this method of quad generation works in most cases, it has the potential to create extraneous quads in the mesh. One situation where this occurs is shown in Figure 3, which finds a corner to construct but should not actually create a new quad. Due to the guarantee of square output quad generation, these artifacts can only appear when extending a diagonal from an "Interpolated" particle. Therefore, we assign each interpolated particle a layer l corresponding to the subdivision layer that generates it. We then bound the maximum value of $k \leq 2^{m-l}$, where m is the maximum subdivision represented in the cloth mesh. If no particle is found within $0 \leq k \leq 2^{m-l}$, the particle is skipped over. This technique allows us to robustly convert our subdivided cloth mesh into a series of quads that can be easily rendered.

4 METHODS OF SUBDIVISION

4.1 Adding Subdivision to Cloth

To dynamically subdivide our cloth, we must first ensure that the underlying representation is capable of representing the newly subdivided masses. Our underlying cloth mesh, therefore, stores the maximum representable subdivision, and increases the cloth's density if subdividing around a point that is already at the highest subdivision. To do this a new 2-dimensional representation is constructed, and each point (i, j) is stored at a new point $(2i, 2j)$.

To subdivide around a point (i, j), we first increase the layer of the point by 1. We then create 8 new points at each adjacent point offset by our new distance $d = 2^{m-l}$. Next, we add up to 8 interpolated points at positions $(i \pm 2d, j \pm d), (i \pm d, j \pm 2d)$. If a point at any of these indices already exists, we do not replace it with a new interpolated point. This new point generation is visualized in Figure 4. Between each of these points we also must create our 3 varieties of springs. These are offset with the same 2^{m-l} distance as before; however, each spring coefficient is also multiplied by 2^l. This ensures smooth simulation across layers. Each point can be subdivided at most once per animation time step.

4.2 Angle Based Subdivision Conditions

The simpler of our 2 methods we use to determine subdivision points relies on the angles between cloth masses, and uses the same techniques outlined in [Hutchinson et al. 1996]. With this technique, we iterate over all indices of the cloth mesh. We then check if that index has either a spring above and below $(i \pm d)$ or left and right $(j \pm d)$. This d is the same used when calculating particles connected via structural springs $(d = 2^{m-l})$. For each of these pairs that exist, we then construct 2 vectors, each originating at our central (i, j) point, and then extending to the adjacent 2 points. We then calculate the angle between these vectors, and if it is above a certain threshold, we subdivide around the central point.

4.3 Hanging Spring Subdivision Conditions

While angle based subdivision provides us with a method to subdivide the mesh, it does not account for all scenarios we may wish to do so. For instance, consider the trivial cloth shown in Figure 5. If the top particles remain fixed, intuition would suggest that the center should be subdivided to allow the cloth to hang. Because there is no central point to create an angle, however, this mesh will never subdivide.

To combat this, we have introduced a new branch of subdivision conditions called "hanging springs". Similar to angle based subdivision, we begin by iterating over the cloth mesh and finding particles connected via structural springs. Instead of looking at both sides, however, we consider each connected particle one at a time. Let us refer to our input particle as A and a structurally connected particle as B. We then imagine "hanging" our spring between A and B, and using the midpoint along this path to calculate a new angle. With this new point, we can perform our initial angle

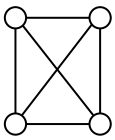

Figure 5: The simplest possible input cloth mesh. This mesh will never subdivide with angle based subdivision conditions, as it is not dense enough to measure any angles.

based subdivision rules to determine if subdividing would allow the simulation to produce a hanging effect.

To solve for this hanging midpoint M, we first project A and B's position from $\mathbb{R}^3$ to $\mathbb{R}^2$. Since we are concerned with the cloth's ability to hang with gravity, we make sure that our projection allows for free gravitational movement. This is equivalent to projecting onto a plane with a normal orthogonal to the gravity vector, which for our simulation is along the y axis. To further simplify, we center our $\mathbb{R}^2$ projection such that A is placed at $(0, 0)$. This gives us the projection from our $\mathbb{R}^3$ world space into projection space as:

$$p(x, y, z) = (x - A_x, z - A_z)$$

We then further reduce the problem by approximating the path of the hanging spring as a parabola. While hanging objects naturally form catenaries rather than parabolas, the approximation is significantly simpler to compute and provides a similar midpoint. With this, we have reduced our hanging spring generation into finding a parabola that goes through both point A and point B. When plugging these into a standard parabolic equation, we can see that any valid solution to this problem must take the form:

$$f(x) = \alpha x^2 + \left(\frac{B_y}{B_x} - \alpha B_x\right) x$$

This, however, represents a set of infinitely many parabolas, as determined by the α constant. To solve for this, we ask the user for a new input: the hanging coefficient, h. With this coefficient, we add a new requirement to our hanging parabola: the arc length of the parabola between A and B must be a certain l, where

$$l = \frac{h \cdot ||A_o - B_o||}{k}$$

In this equation, k is the spring coefficient between A and B, and A_o, B_o are the original positions of the particles in our input mesh. If the particles did not exist in the input mesh, their original positions are linearly interpolated between the particles that were. This definition of length means that as we increase the stiffness of our springs, either through input parameters or subdivision, A and B must be forced closer together to trigger a subdivision. This intuitively aligns with the understanding that stiffer cloths will bend less.

To use this length l to solve for α, we must evaluate the arc length on our input and equate it to l. This can be calculated with the equation:

$$l = \int_0^{B_x} \sqrt{1 + (2\alpha x + \frac{B_y}{B_x} - \alpha B_x)^2}\, dx$$

This equation can be expanded to remove the integral, however, it ultimately remains analytically unsolvable. Therefore, we solve for α numerically in our simulation. Once we have α calculated, we simply solve for $f\left(\frac{B_x}{2}\right)$, and then project $\left(\frac{B_x}{2}, f(\frac{B_x}{2})\right)$ into the $\mathbb{R}^3$ world space to perform our angle based decision.

5 RESULTS

We tested out dynamic subdivision on a number of different input cloth meshes, and for each input tested 3 different categories of simulation:

- Simulation without any subdivision
- Simulation with uniform subdivision across the mesh
- Simulation with dynamic subdivision

The full table of these can be seen in Figure 6. When looking at completely unsubdivided meshes, a number of artifacts appear. This is particularly true on the pole and tablecloth simulations, where the lack of subdivision creates jagged artifacts as the corners of the mesh collapse. This is counteracted by the dynamic subdivision system, which much more closely mimics the fully subdivided mesh.

This system, however, is not without its drawbacks, both in its results and its performance. We see that in these same situations, the dynamic subdivision remains unable to represent extra folds in the mesh that would appear with the pre-subdivided mesh. This is readily apparent in both our tablecloth and pole simulations, which create excess folds when simulated with higher fidelity. One way that this could be combated is to test for hanging spring subdivisions along the normal of a cloth quad in addition to along gravity, creating a sort of "popping spring" test. This, however, was out of the scope of testing for this paper.

While difficult to quantify, we do see a distinct speed disadvantage of the dynamic subdivision system whencompared to the complete lack of subdivision. To help combat this, we introduced an input parameter into the cloth input determining the frequency of subdivision checks. When the simulation checks for subdivision each frame, we see a slight but noticeable slowdown. This performance, however, remains orders of magnitude more performant in our tests than the pre-subdivided meshes.

6 CONCLUSION & FUTURE WORK

Our dynamic subdivision system has a number of advantages, but also comes with a number of caveats. One of the primary caveats is due to an expansion of our cloth inputs when compared with those allowed in [Hutchinson et al. 1996]. In that original paper, it is presumed that there do not exist any angles in the original cloth mesh that would trigger a subdivision. Given this, when their simulation detects an angle based subdivision it can “jump" back an animation step, perform subdivision, and continue on. Because our simulation allows for inputs that would trigger subdivisions in the first animation step, we cannot guarantee that there is a step to “jump" back to. This has the result that any angle based subdivision system will trigger subdivisions up to the maximum subdivision depth. While this is not true for hanging spring decisions, it causes significant over subdivision with high maximum depths. One solution to this we experimented with was smoothing

out subdivisions using the Loop subdivision scheme [Loop 1987], however we found that the smoothing on subdivision was jarring, and was not feasible for live use. One area of future work would be splitting subdivisions into 2 sections. If an angle based subdivision is triggered on the input mesh without simulation, then we would smooth the subdivision according to Loop's rules; however, if a subdivision occurred throughout the animation we would not perform any smoothing. This modifies the input mesh in such a way that we could guarantee future time steps could always "jump" back to to a state not requiring subdivision. As such, we could use the time step fixes as described by [Hutchinson et al. 1996] for unlimited subdivision depth.

Beyond this 2 step subdivision there remain a number of areas for future work in this field. The first and most useful would be the implementation of dynamic subdivision in concert with a more modern cloth simulation technique. While the classic method we used [Provot 2001] provides a functional cloth simulation, new techniques such as those discussed in [Baraff and Witkin 1998] and [Bridson et al. 2002] provide substantially more physically accurate and potentially more performant cloth simulations. Applying the subdivision conditions discussed in this paper would be relatively simple; however, allowing for the simulation of arbitrary subdivisions would require more thorough research.

Another area of experimentation exists in a 2 phase approach to this dynamic subdivision. In the first phase, we would actively check for subdivisions as necessary, creating a smooth mesh when hanging. We could then pass this subdivided mesh as input into phase 2, which would then not actively check for more subdivisions. This second phase would be substantially faster than phase 1, but still remain smoothly subdivided. This 2 phase approach could be particularly useful in interactive environments where one would want to pre-compute a minimal mesh when resting that could then interact with ambient factors or user interaction faster than our current systems.

7 DIVISION OF LABOUR

While some of the work for this project was inherently collaborative, the project can approximately be broken down by person as the following:

Eleanor Olson

- Implemented the layered simulation according to the specifications of [Hutchinson et al. 1996]
- Implemented the subdivision of the underlying cloth mesh.
- Designed and implemented the rendering of a subdivided cloth mesh to quads.
- Designed and implemented the ideas surrounding hanging spring subdivisions.
- Wrote the slideshow presentation accompanying the paper.
- Wrote the Methods of Simulation, Results, and Conclusion sections of this paper.

Kana Rudnick

- Implemented iteration of the cloth mesh for subdivision decisions.
- Implemented Angle Based spring decisions.
- Determined base ideas around potential subdivision conditions.
- Assisted with debugging of various sections of the simulation.
- Wrote the Introduction and Related Work sections of this paper.

8 ACKNOWLEDGMENTS

The groundwork for this simulation software was written by Professor Cutler for use in the homework of her Advanced Computer Graphics course, of which this paper is a product. This provided the basic parsing of object files, as well as rendering via both OpenGL and Metal. Additionally, this software provided us numerous tools for interactive with 3D objects and a robust vector library.

All of the code written for this project can be found on GitHub (https://github.com/eleanormally/dynamic-cloth). Our work is implemented in C++, and should run on all major operating systems, provided that OpenGL or Metal development frameworks are installed.

REFERENCES

[1] David Baraff and Andrew Witkin. 1998. Large steps in cloth simulation. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '98)*. Association for Computing Machinery, New York, NY, USA, 43âĂŞ54. https://doi.org/10.1145/280814.280821

[2] Robert Bridson, Ronald Fedkiw, and John Anderson. 2002. Robust treatment of collisions, contact and friction for cloth animation. *ACM Trans. Graph.* 21, 3 (July 2002), 594âĂŞ603. https://doi.org/10.1145/566654.566623

[3] Dave Hutchinson, Martin Preston, and Terry Hewitt. 1996. Adaptive Refinement for Mass/Spring Simulations. In *Computer Animation and Simulation '96*, Ronan Boulic and Gerard Hégron (Eds.). Springer Vienna, Vienna, 31–45.

[4] Charles Loop. 1987. *Smooth Subdivision Surfaces Based on Triangles.* Ph.D. Dissertation. University of Utah.

[5] Xavier Provot. 2001. Deformation Constraints in a Mass-Spring Model to Describe Rigid Cloth Behavior. *Graphics Interface* 23(19) (09 2001).

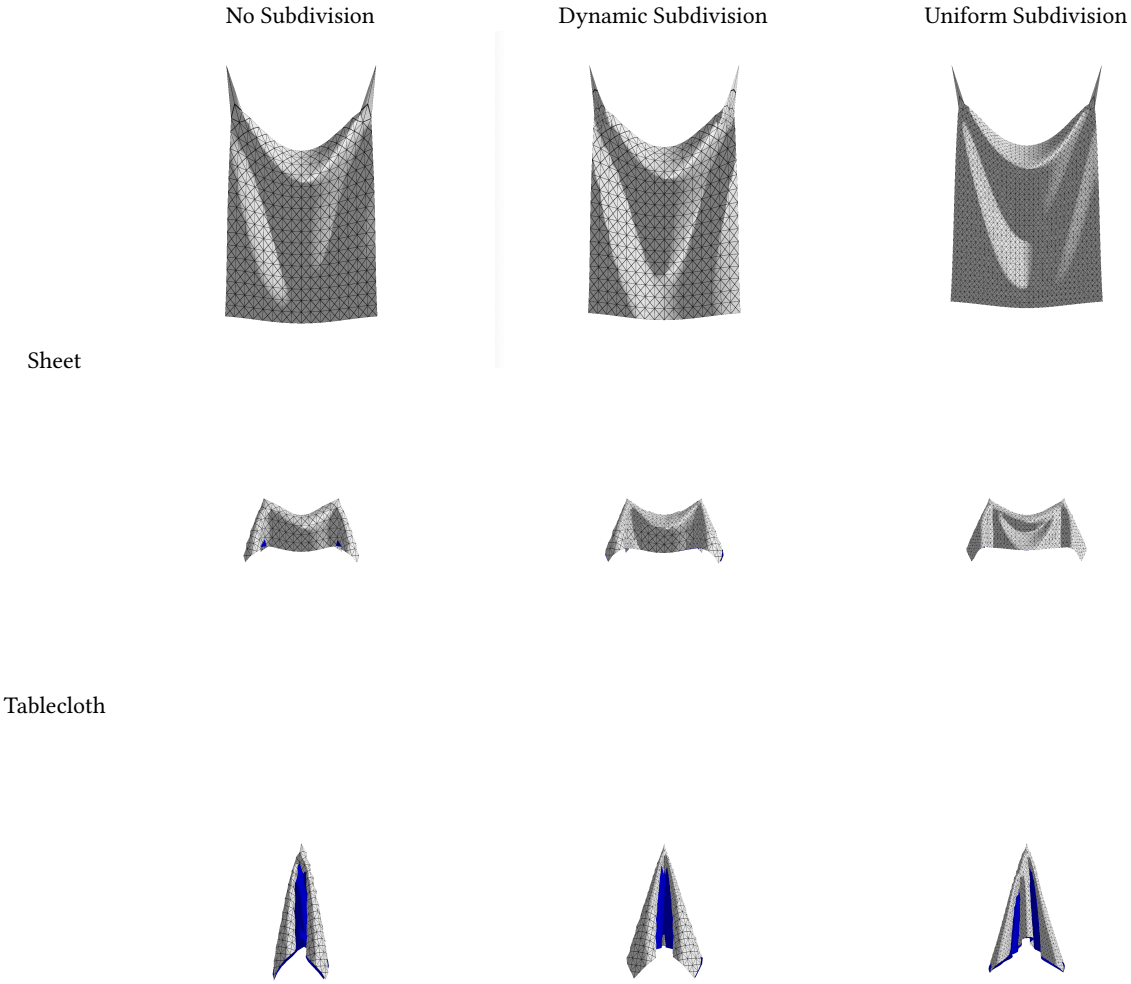

Figure 6: An example of different input cloths with various categories of subdivision. Each dynamic subdivision cloth was limited to 1 layer of maximum subdivision, and uniform subdivision similarly adhered to this single subdivision layer.