CSC 473: Fundamental Of Computer Animation 2023
Brandon Howarth
(Editors)

Volume 42 (2023), Number 8

Cloth Simulation with Tear Physics

Samuel Walker V00896718

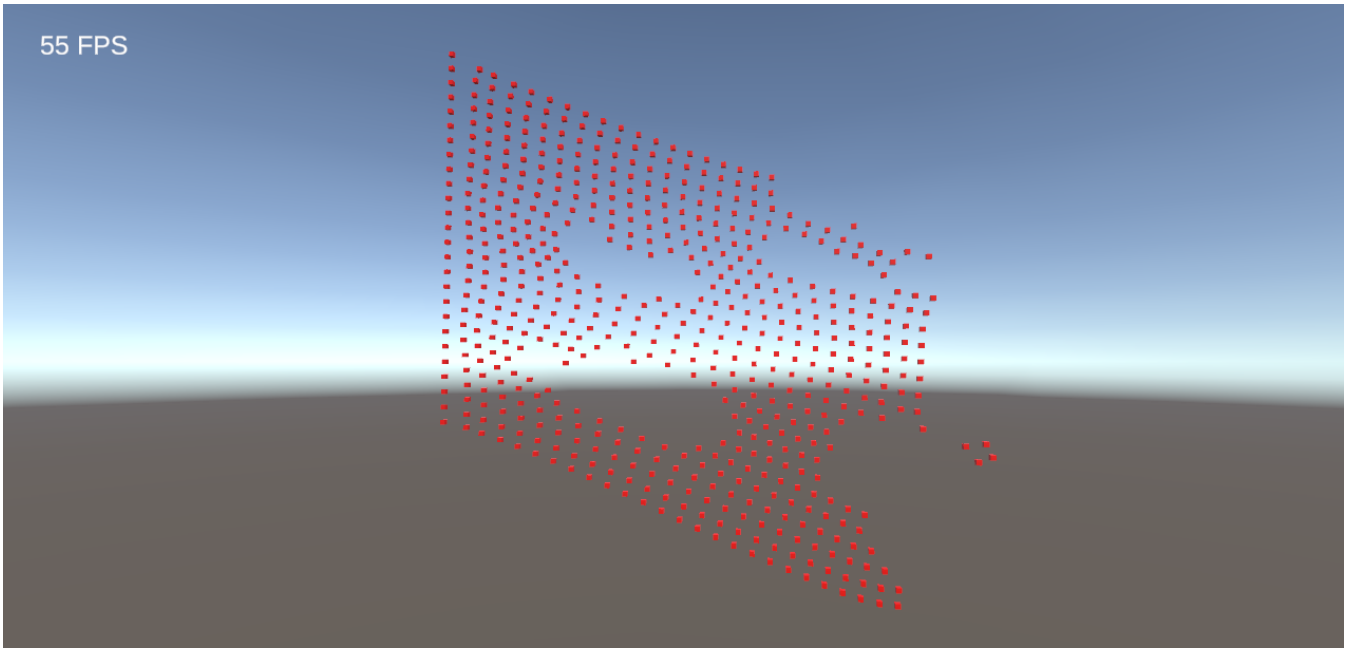

Figure 1: *A torn flag simulated with a mass-spring system*

Abstract

Cloth simulation has been an important field in computer graphics for many years and is still being improved upon to this day. With increased hardware capabilities, more efficient algorithms, and easier to access systems it has never been easier to create and simulate cloth-like materials both quickly and accurately. Cloth simulation is most commonly used in the film and games industries to make the virtual worlds they create seem more alive. However, its uses are expanding in many other industries such the fashion industry to simulate textiles and create new clothes.

Over the years many different methods of cloth simulation have been introduced to the market from static single frame cloth renderers to complex physically based cloth simulations. We have even learned how to create physically based simulators that run in real time. This paper describes a simple 3d cloth simulation using a mass-spring system with added ripping and tearing physics. This solution uses real time algorithms to accurately render multiple cloth styles that can accommodate many sizes of simulated cloth. This paper uses physical accuracy and frames per second to evaluate this method and to see how well it performs on consumer machines.

CCS Concepts

• ***Computing methodologies*** → ***Real-time simulation;***

1. Introduction

The most prominent industries for cloth simulation tend to fall under the entertainment category with many films and games using cloth simulation to build more realistic and immersive worlds. These complex simulations can be used to create flags, clothing, and many more small details that can bring life to an otherwise

lifeless world. Film studios can use more complex and accurate methods of cloth simulation that are not required to run in real time as they will often have large render farms that can take their time to render the most physically accurate final product possible. However, using a real time simulation technique can assist the CGI artist when first designing a scene as it allows the artist to develop the desired look on demand. Real time techniques improve workflow by not requiring the artist to wait for longer periods of time for their cloth simulation to be calculated at the render farm. In video games and other interactive media, developers require real time cloth simulation algorithms as interactive media is required to run at a decent speed that allows for user interaction. In these examples a simple and real time simulation like the one elaborated upon in this paper comes into play.

This paper outlines a solution that allows for real time cloth simulation using a particle and spring based system that physically and accurately simulates realistic looking cloth. This method also runs at reasonable speeds on consumer grade machines which is elaborated upon later in this paper. In this system we demonstrate Forward Euler, Symplectic Euler, and Verlet integration for different use cases and evaluate their performance in various ways. Including multiple integration methods allows the user to choose between a more realistic but slower simulation or a faster but less accurate one depending on hardware limitations or artistic needs. We also include a way to simulate the tearing or ripping of the simulated cloth. This allows for creating more interesting designs for a variety of purposes in the film and interactive media industries.

2. Related Work

There are many different methods for cloth simulation that I have found in my research. One of the first methods created for cloth simulation used what is called a geometric technique. This technique was pioneered by Andre Weil in 1986 by using hyperbolic cosine curves to create a simulated cloth-like material [Wei86]. This method was somewhat well suited for quickly creating single frame renders but was not very physically accurate or visually appealing when compared to more modern methods. However, this paper laid the groundwork for the methods we have today.

As more physically based models arose the geometric method became limited by its inability to create dynamic models and was overtaken by more modern methods. Most modern cloth simulations use a more physically accurate mass-spring system [BHW94] consisting of a grid of particles with constant mass connected by a net of multiple different varieties of springs that are covered in more detail in section 3 of this paper. These particle spring methods can be simulated in many different ways with some running in real time and some, more accurate methods, taking longer periods of time to render.

Many simple, real-time cloth simulations either use various Euler integration methods such as Symplectic Euler or Forward Euler integration for quick simulations. However, Verlet integration is more physically accurate but requires more calculations and is therefore more expensive to run on weaker machines. All three of these integration methods are explicit integration methods which are considered to be less physically accurate than implicit methods as they accept a larger percentage of error [BW98]. While explicit integration methods are less accurate they also run much faster which is important for creating a real time simulation. Implicit methods of integration are much more accurate but fail to run at real time [HBL04].

Forward Euler integration, while reportedly the simplest integration solution, can cause instability in the simulation especially when many forces are being calculated at the same time in complex simulations. This can cause the simulation to “blow up due to artificial energy gains” [Rot17]. A video example of this phenomenon on a cloth simulation is demonstrated in Erick Schimnowski’s video on Euler integration methods for cloth simulation [Sch]. Symplectic Euler reduces the "blowing up" problem by calculating the velocity in the next time step and using that information as well as the current velocity and current position to estimate the position at the next time step. This is elaborated upon in section 3.2.

Verlet integration is less likely to break down by more closely approximating the Taylor series. This solution tends to be more stable and more realistically detailed compared to Forward Euler or Symplectic Euler when used to simulate cloth. Because of this, Verlet integration will also last for longer periods of simulation time without “blowing up” [Smi11]. Thomas Jakobson, a researcher working for IO interactive, shows how this technique is used commonly in many popular video games [Jak01].

3. Overview

The method described in this paper uses many key aspects from the methods referenced in the related works section to create a successful mass-spring cloth simulation that runs in real time. Particles are generated in a grid pattern and a series of algorithms attach them together by using different types of springs with specific parameters. These methods together form a realistic cloth-like simulation.

Different types of springs achieve different goals to create a lifelike cloth simulation and are elaborated on in the following section. To simulate the physics of the springs we use and compare multiple different forms of integration. Verlet integration is the most accurate algorithm this paper uses for finding the approximation of the next position and velocity of the particle. Because Verlet integration uses more accurate calculations it is slightly slower and is more computationally expensive than the other methods tested in this paper.

Forward Euler integration is the fastest but the most prone to error of all of the methods used. Symplectic Euler integration uses slightly more complex information than Forward Euler while estimating integrals but it is also more prone to error than Verlet integration.

To create the ripping and tearing simulation we use a brush tool for the user to rip into the cloth to achieve the look they are striving for. We also use a simple method of simulating the cloth ripping under stress from the weight of the cloth, weaknesses created by user rips, or external forces such as gravity and wind creating too much excess force on the system.

The final step was to render a visible mesh to connect all of the

particles in a more realistic way. The method used for the generation of the mesh in this paper is quite simple but it is also incompatible with the dynamic ripping feature outlined in the previous paragraph.

3.1. Particles

As stated above, this paper outlines a method that uses a mass-spring system to simulate cloth-like materials. This system is defined by an array of particles that are connected by springs in different ways to create different styles of simulation. Each particle has a defined position, mass, velocity, and force. The position and velocity of each particle are influenced by the different integration methods and the force applied to each particle through the different and interconnected springs. The particles mass is constant throughout the simulation but it can be changed at the beginning to influence the properties of the simulated textiles. This can be used to create heavier or lighter cloth depending on the parameters.

This relationship can be seen in the following two images taken from the method described in this paper. One is set with the particles at their default mass 2 and the other has added mass to illustrate the changes that take place when the mass of the particles are changed 3.

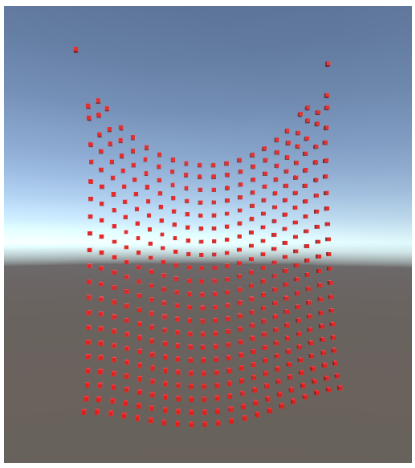

Figure 2: *Particles simulated with .2 mass*

3.2. Spring Forces

To calculate spring forces we use a couple different formulas. We must calculate both the spring force and the spring damper force that controls the springiness of the spring. Each spring object holds the springs strength (k^s_{ij}), a damper coefficient (k^d_{ij}), the rest length (l_{ij}), and two particles, i and j, that the spring connects. The position of these particles is denoted by x_i and x_j. To calculate the spring force we use the following formula:1

$$F^{sp}_{ij} = k^s_{ij}(l_{ij} - |x_i - x_j|)\frac{x_i - x_j}{|x_i - x_j|} \tag{1}$$

This equation makes it simple to calculate both the force of particle i onto particle j and the force from particle j onto particle i as each force is simply the negative of the other.

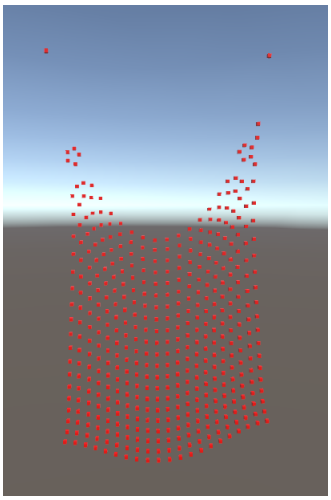

Figure 3: *Particles simulated with .5 mass*

To calculate the damper force on the spring we use the following formula: 2. This formula works similarly to the spring force 1 where the negative of the force from particle i to particle j equals the force from particle j to particle i. In this formula v_i and v_j are the velocities of the given particles.

$$F^d_{ij} = -k^d_{ij}((v_i - v_j) * \frac{x_i - x_j}{|x_i - x_j|})\frac{x_i - x_j}{|x_i - x_j|} \tag{2}$$

We then add these forces together for each particle with external forces and dampers such as wind (F^{ext}), gravity ($m_i g$), and environmental dampening (k_d).

$$F_i = -k_d v_i + m_i g + F^{ext} + \sum_j F^{sp}_{ij} + F^d_{ij} \tag{3}$$

3.3. Integration Methods

There are multiple different integration methods in the Euler family that can be used for simple cloth simulation. Forward Euler integration is one of the most simple forms of somewhat accurately calculating the next position 5 and velocity 4 of any given particle. This method only uses current information about the position, velocity, and force being applied to calculate the next position and the velocity of the particle at that next position. To calculate the velocity at the next position we multiply the change in time by the acceleration of the particle and add that to the current velocity 4. To calculate the position of a particle at the next time step. We take the current velocity of the particle, multiply it by the change in time and add that to the current position to get the next position. 5

$$v_p(t + \Delta t) = v_p(t) + \Delta t \frac{F(x_p(t), v_p(t), t)}{m_p} \tag{4}$$

$$x_p(t + \Delta t) = x_p(t) + v_p(t)\Delta t \tag{5}$$

This method of integration allows the simulation to run very quickly but also makes it more prone to error sometimes making

the simulation unstable when simulating many particles and springs over a longer period of time.

By using Symplectic Euler integration We can make the simulation more stable while only impacting performance by a small amount. To calculate the velocity of the particle with this method we use the same equation as forward Euler 4. The calculation for position is very similar as well except that it uses the estimated velocity at the next time step to calculate the next position. This method takes the velocity of the particle at the next time step, multiplies it by the change in time and adds it to the current position of the particle to generate the next position. 6

$$x_p(t+\Delta t) = x_p(t) + v_p(t+\Delta t)\Delta t \tag{6}$$

Verlet integration is still fast and runs in real time but does have a noticeable effect on frame rate when switching from Euler integration. However, this method also creates a more mathematically accurate simulation by using past, present, and future information to calculate what the next position and velocity will be for each particle. This allows the simulation to look more accurate and makes it less prone to error and less likely to break down. However, because it uses past information we must supplement information that does not exist at the beginning of the simulation. To solve this problem we simply use Symplectic Euler to generate the next position at time zero and continue to use Verlet integration for the rest of simulated time. This integration method calculates the next position by multiplying the acceleration (force/mass) by the time step squared then adding 2 times the current position of the particle and subtracting the position at the previous time step. 7

$$x_p(t+\Delta t) = 2 * x_p(t) - x_p(t-\Delta t) + \frac{f_p(t)}{mass} * \Delta t^2 \tag{7}$$

We then use the new position we just calculated to generate the velocity at the next position. 8. We do this by taking the next position and subtracting the previous position divided by 2 times the time step.

$$v_p(t+\Delta t) = \frac{x_p(t+\Delta t) - x_p(t-\Delta t)}{2\Delta t} \tag{8}$$

3.4. Structure Springs

After creating the net of particles, implementing the different integration methods, and calculating the spring forces we must find a way to parameterize these forces. This is where the different types of springs come in.

The first and most obvious type of springs are the structure springs. These springs are generated in between each particle at right angles creating a grid structure as shown in figure 4. This pattern of springs creates the general structural integrity of the cloth and forms the base for the other springs to build off of. To allow for adjustable sizes of cloth, this paper uses a specific algorithm to generate the springs. For this algorithm to work we must store the particles in a two dimensional array to represent their generated grid pattern.

Each particle [x,y] is connected to the particles [x,y+1], [x,y-1], [x+1,y], [x-1,y] by a spring 4. When only these springs are used the simulated cloth will fold in on itself and it becomes too soft and droopy 5.

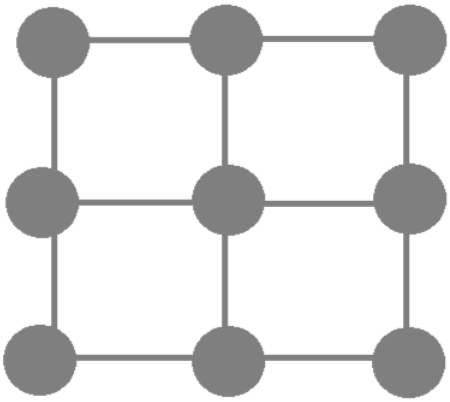

Figure 4: *Structural Springs*

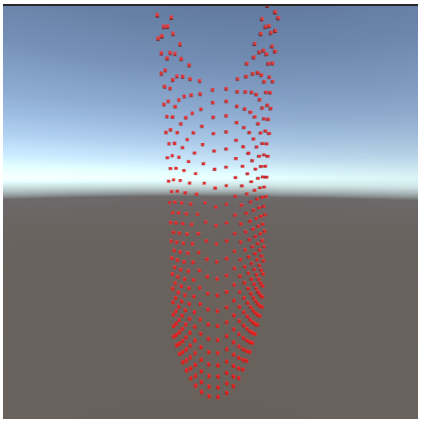

Figure 5: *Cloth Simulation with Only Structure Springs*

3.5. Shear Springs

Shear springs are implemented to stop the cloth from collapsing in on itself and to add structural rigidity to the material. These springs connect each particle to their diagonal neighbors to form a net like structure which takes the previously stretchy simulation and begins to create a more accurate cloth structure 7. By accessing each particle in the two dimensional array we can connect them to each other algorithmically.

Each particle [x,y] is connected to [x+1, y+1], [x-1,y+1], [x-1,y-1], [x+1,y-1] by a spring. Figure 6 shows the form that this algorithm creates

3.6. Flexion Springs

Flexion springs connect every other spring to make the cloth stiffer and to stop it from folding in unexpected ways. Without these springs the cloth tends to get tangled up in itself especially when external forces such as wind are applied. This phenomenon tends

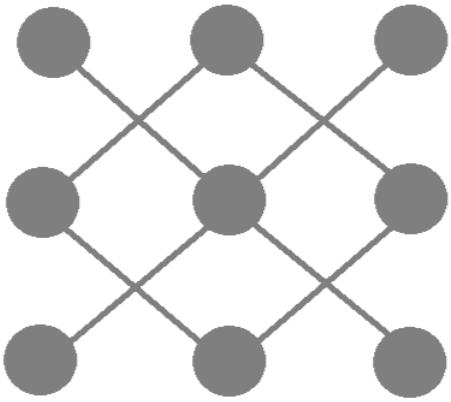

Figure 6: *Shear Springs*

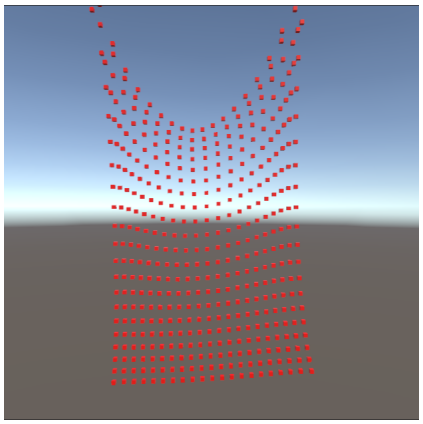

Figure 7: *Cloth Simulation with Structure and Shear Springs*

to take place more in the thinner and more wispy sections where tearing has taken place. The algorithm for creating these springs is as follows. Each particle [x,y] is connected to [x, y+2], [x+2, y], [x,y-2], [x-2,y] 8.

Using all of these types of springs together we can see a more cloth-like material forming 2. In implementing this method I made sure to check to make sure the spring doesn't already exist as to not double up on spring power for some particles.

3.7. Ripping and Tearing

A simple algorithm allows for the removal of particles and springs with a brush tool. This allows the user to create different types of non-uniform cloth simulation. When particles are subject to strong wind forces and the bond between the particles is weak the springs between them will tend to rip creating very interesting emergent behaviours with the simulation. To achieve this we simply measure

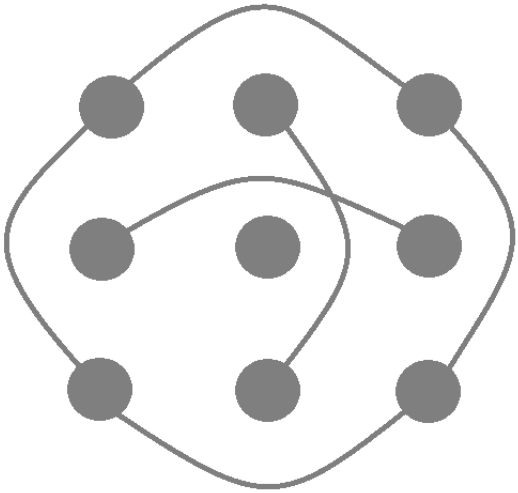

Figure 8: *Flexion Springs*

the distance between particles, if it becomes too great the spring in between the two particles will break allowing them to separate. With this simple rule a small tear will often cause a domino effect of ripping cloth along a seam 13.

3.8. Spring and Particle Settings

Changing different parameters within the cloth simulation can create drastically different types of simulated materials, some more accurate than others. This particular method described in this paper can be fairly finicky and will break down if the parameters are not set correctly. Changing the mass will change the weight of the overall cloth causing it to sag and pull giving the illusion of a heavy, stretchy material 3. Changing the strength of the springs can also affect this aspect. Making the springs stronger can help to resist against the heavy particles but it will also make the material more springy. The dampening coefficient will make the springs less springy and allow less give between each particle. Environmental drag can be changed to give the appearance of a material in a thicker substance like water.

Changing the gravity and wind strength is also possible in this method. The simulated wind consists of a simple force generated using a perlin noise function. This gives the appearance of inconsistent wind patterns and allows for some flapping of a simulated flag. This force is applied to all particles in the system evenly. While this is not the most realistic way to generate wind it works for the needs of this demonstration. These external forces are in the form of Vector3's that are added to the spring forces and applied to each particle.

3.9. Generating A Mesh

The final step was to generate a mesh in between the particles to make the actual cloth shape visible underneath the mass-spring sys-

tem 9. To generate a mesh in the shape of the cloth this paper used the particles as vertices and generated triangles by iterating through them and adding them to an array of points called the triangles array 1. We then render the mesh using the vertices in the triangles array by using GL.TRIANGLES or in our case the built in unity mesh system.

Algorithm 1 Mesh Generation

```
triIndex ← 0
vertIndex ← 0
for each particle along the height do
  for each particle along the width do
    triangles[triIndex + 0] ← vertIndex;
    triangles[triIndex + 1] ← vertIndex + 1;
    triangles[triIndex + 2] ← vertIndex + clothWidth + 1;
    triangles[triIndex + 3] ← vertIndex + 1;
    triangles[triIndex + 4] ← vertIndex + clothWidth + 2;
    triangles[triIndex + 5] ← vertIndex + clothWidth + 1;
    vertIndex ← vertIndex + 1;
    triIndex ← triIndex + 6;
  end for
  vertIndex ← vertIndex + 1;
end for
```

4. Quantitative Evaluation

To quantitatively evaluate the method outlined in this paper I recorded the frame rate while running the simulation. This is based on my own hardware and will fluctuate depending on the user. However, comparing these values will be very useful for evaluating performance of the method on user grade machines as well as creating important data on the performance of different features and parameters. The machine these figures are recorded on is a Lenovo Thinkpad T14 running Windows 11 pro. The processor is an AMD Ryzen 7 Pro 6850U with Radeon Graphics and the program is running in the unity editor on a solid state drive. The system has 32 gigabytes of Read Access Memory and the version of Unity the simulation is built in and runs on is 2021.3.17f1.

4.1. Integration Methods

Different integration methods have noticeable effects on performance. Verlet integration uses more computation power than the different Euler methods as it creates a closer approximation to the mathematically true position and velocity of any given particle. As we are running these integration methods every frame, having a faster integration method can greatly affect performance. For this test it is important to create an even test ground for each method to see how they perform under specific parameters.

4.1.1. Verlet Integration

For the first test I used a 20 by 20 particle cloth with no generated mesh, no wind, and no ripping or tearing under pressure. With these parameters and running the Verlet integration method I got fluctuating average frame rates between 50 and 70 frames per second on average. Generating a visible mesh between the particles did not seem to affect the frame rates very much retaining 50 to 70 frames per second 9. With the wind force applied the simulation would tend to run at about 40 to 50 frames per second on average. Adding the checks for rips while under stress caused very little change with frames per second occasionally dropping to below 40 but on average it stayed between 40-50 frames per second with the wind force applied.

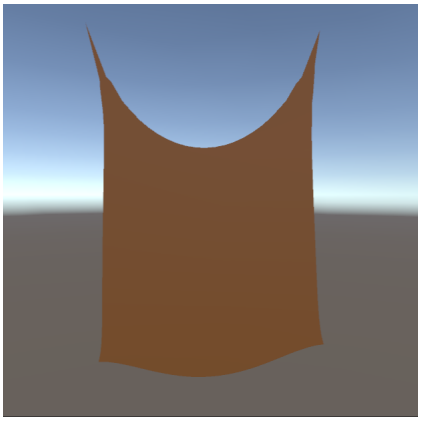

Figure 9: *Cloth Simulation with Invisible Particles and Mesh*

4.1.2. Symplectic Euler Integration

Symplectic Euler integration tends to run quite a bit faster with an average frame rate of 70-100 frames per second on my machine with a basic 20 by 20 grid of particles with no visible mesh, wind, or ripping under pressure. Again, generating the visible mesh makes very little change with the frame rate still bouncing between 70 to 100 frames per second on average. Adding the wind force affects the frame rate quite a lot making it drop to an average of 40 to 70 frames per second on average. These results surprised me as it seems that Symplectic Euler integration tends to run at about the same frames per second as Verlet integration when the wind force is applied. Again, adding the checks for breaking when the particles reach a certain distance does not affect the frame rate and it remains pretty similar.

4.1.3. Forward Euler Integration

For an added comparison I wanted to test the simulation with Forward Euler integration as well as it is quite easy to implement and it is one of the simplest methods of integration. With the basic 20 by 20 cloth, no additional wind force, no generated mesh, and no ripping under pressure I got around 70 to 90 frames per second on average with Forward Euler integration. The lack of improvement from Symplectic Euler could have come from the algorithm itself or because I used Unity's getComponent function which may cost more than using an established vector3 like I used for Symplectic Euler. Again, generating the visible mesh had little effect on the performance with the simulation still running at about 70 to 90 frames per second on average.

Adding wind force is where Euler integration became interesting. The method seemed to break down with the added force and

frame rates began to vary wildly going as low as 15 frames per second to as high as 149. The particles also began to twist and tangle in an interesting vortex pattern 10 as the simulation began to break down. The cloth then "exploded" and the particles left the the camera's viewport which caused the frame rate to drastically drop until it finally came to rest hovering at around 15 frames per second. "Not A Number" errors also started appearing as the particles flew off to infinity. Adding the tear on distance feature did not seem to change the frame rate with an average of 70 to 90 frames per second before it started breaking down again.

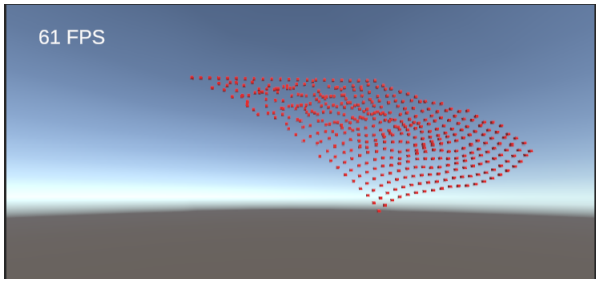

Figure 10: *Euler Behaviour Under Wind Force*

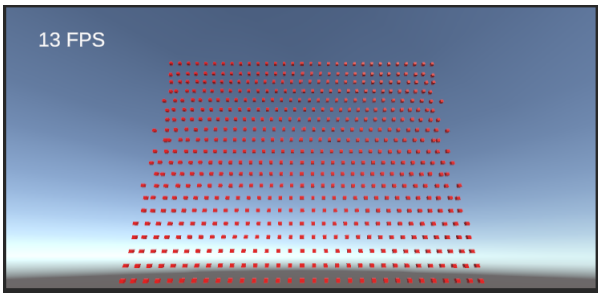

Figure 11: *Forward Euler Behaviour at 30x30*

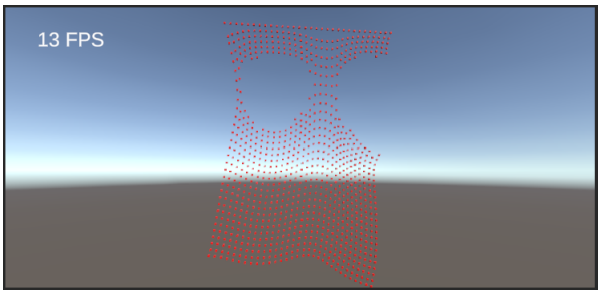

Figure 12: *Verlet Simulation at 30x30*

4.2. Maximum cloth size

Another method of testing that I wanted to employ was to see how many particles or how big of a cloth each integration method could handle with reasonable frame rates. With this method I also wanted to see if any emergent behavior arose when adding an increasing number of spring forces.

Throughout my testing with the number of particles I was able to generate up to a 30 by 30 grid of particles until the frame rates began falling under 15 frames per second with all of the integration methods. This was a rather surprising find as every integration method ran at an average of 12 to 14 frames per second with the 30 by 30 particle grid rendered 12. There was very little difference in performance between each integration method in my testing. Forward Euler was once again the one interesting outlier. When the simulation runs for 15 - 20 seconds the particles begin to shake back and forth. This is due to the increased level of error that comes with using this integration method 11. It is worth noting that these numbers are created on my own personal machine as well as in the Unity editor which will make the simulation run a bit slower than if it was running independently.

5. Qualitative Evaluation

Overall the method outlined in this paper works quite well for simulating relatively realistic cloth-like materials. Verlet and Symplectic Euler integration performed well and had very little artifacting from their approximation when compared to Forward Euler integration. With higher particle counts the cloth material looks more convincing however there are some performance issues that come with higher particle and spring counts with lower frame rates that make the tool less usable.

5.1. Parameterization

Throughout the building of this method I was constantly searching for the best parameters to make the most realistic looking cloth. The best solution I found was setting the spring strength to 20 which gives the cloth a good amount of spring and allows it to give under tension but not enough where the simulation looks too stretchy. For the spring dampening, setting it at 0.5 gave me the best results for adding a bit of stiffness to the springs. I set the rest length at 1 to keep the particles close enough together so that the cloth has a high enough resolution to look realistic.

While the parameters can be changed to achieve many different styles of cloth-like materials some combinations of parameters can cause the simulation to break down. This is particularly true when it comes to spring strength and particle mass.

5.2. Rip and Tear

The ripping and tearing feature is the main draw of this method that allows for interesting ripping simulations. Adding the rip under pressure feature adds a lot of realism and dimensionality to the simulation with the simulated cloth ripping at weak points and under wind pressure. This will often set off a chain reaction as the cloth will rip clean through where it is weakest 13.

6. Conclusion

Overall this project has been a success. This paper describes a method for real time cloth simulation with ripping and tearing mechanics which was successfully built. The simulation can be configured in many different ways and compares different integration

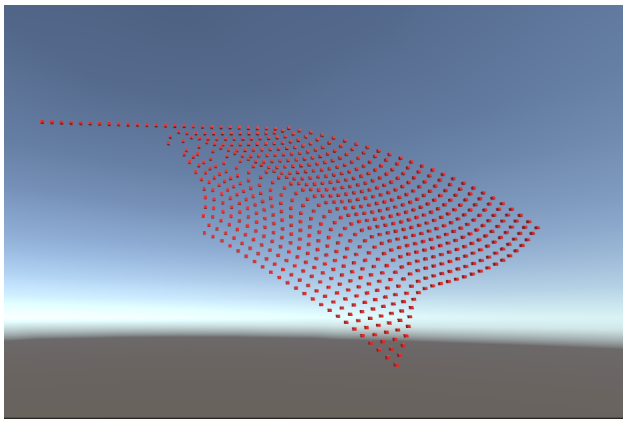

Figure 13: *Emergent ripping of cloth at weak point*

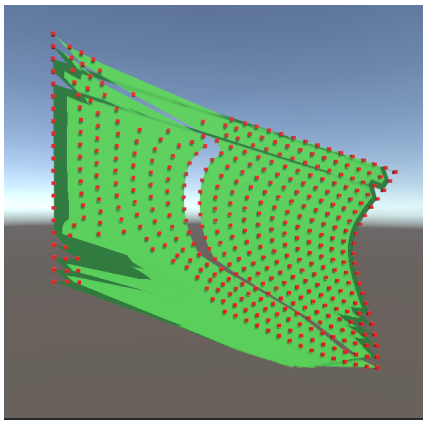

Figure 14: *Mesh Not Functioning*

methods for simulating cloth in a realistic and believable way. This paper succesfully displays the strengths of mass-spring systems for real time cloth simulation while also showing the strengths and weaknesses of different integration methods for this purpose. Overall Verlet integration seems to be a well formed approximation of integration that outclasses the Symplectic Euler method as well as the Forward Euler method. While Symplectic Euler is slightly more efficient, it is well worth the more physical accuracy achieved with Verlet integration. Forward Euler is almost unusable for complex cloth simulation as it begins to break down quickly as the simulation progresses through time.

There are some limitations that come with the method described in this paper. First and foremost is the resolution of the cloth. With this method we can run a 30 by 30 resolution if low frames per second will not be an issue for the use case. However, below 30 frames per second is mostly unusable for most interactive applications. This paper found that 25 by 25 was the largest possible cloth that ran above 30 frames per second. I am curious how well this method would run when optimized and rendered only using openGL and C++ with a more simplified system.

Another limitation is the lack of functionality for ripping cloth and generating a visible mesh at the same time. I was able to implement both of these features separately but ran out of time to integrate them together. When ripping the cloth with the visible mesh activated we get lots of strange artifacting and holes in the mesh where there shouldn't be. Even when a portion of the cloth separates completely from another piece the mesh the simulation continues to draw the mesh between them causing some unfortunate visuals 14.

The lack of real time interactivity also makes this method suffer. Being able to change parameters on run time would have been a great way to make this tool more functional and interesting. As the program currently works you must stop the simulation, change a parameter, then restart the simulation which takes a lot more time than should be necessary.

For future work I would like to make this method more performant so users can run more complex and large cloth simulations. Adding functionality to the mesh so it breaks in the correct places when simulating the ripping of the cloth. Increasing functionality and real time interactivity of the system will also be included in future builds to make the tool easier for users to change parameters on run time. Adding collisions and in world interaction would also be very interesting for simulating the creation of clothes and textiles.

References

[BHW94] BREEN D. E., HOUSE D. H., WOZNY M. J.: Predicting the drape of woven cloth using interacting particles. In *Proceedings of SIGGRAPH, 1994* (1994), IEEE.

[BW98] BARAFF D., WITKIN A.: Large steps in cloth simulation. In *Computer Graphics Proceedings, Annual Conference Series* (1998), IEEE.

[HBL04] HAYLER G., BANGAY S., LOBB A.: Large steps in cloth simulation. In *Implicit and Explicit Methods of Cloth Simulation* (2004), IEEE.

[Jak01] JAKOBSEN T.: Advanced character physics. In *2001 Proceedings, Game Developers Conference* (2001), IEEE.

[Rot17] ROTENBERG S.: Cloth simulation. In *CSE169: Computer Animation* (2017), IEEE, p. 37.

[Sch] SCHIMNOWSKI E.: Euler integration cloth. URL: https://www.youtube.com/watch?v=kEgy6zfgqD8.

[Smi11] SMITH P.: Flag in the wind: Using the fundamentals of cloth simulation to achieve a flag effect. IEEE.

[Wei86] WEIL J.: The synthesis of cloth objects. In *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1986), SIGGRAPH '86, Association for Computing Machinery, p. 49–54. URL: https://doi.org/10.1145/15922.15891, doi:10.1145/15922.15891.