Spring System for Cloth Simulation

A Report about the Outcome

Ibrahim Mushtaq

Ontario Tech University

Purpose

The purpose of this assignment is to explore how cloth can be simulated in a graphics environment. This is done through a Mass Spring System which allows each particle to be attached to one and other. The goal was simple, generate a cloth (Which is like a plane), and add forces to each particle within the cloth which can simulate cloth movement.

Theory

The Mass-Spring system model is a system of mass connected by a spring[1]. This system allows mesh deformation as it uses other masses to change. For the cloth simulation, there are three basics types of spring system that describes the overall structure of the cloth, as can be seen on *Figure 1*

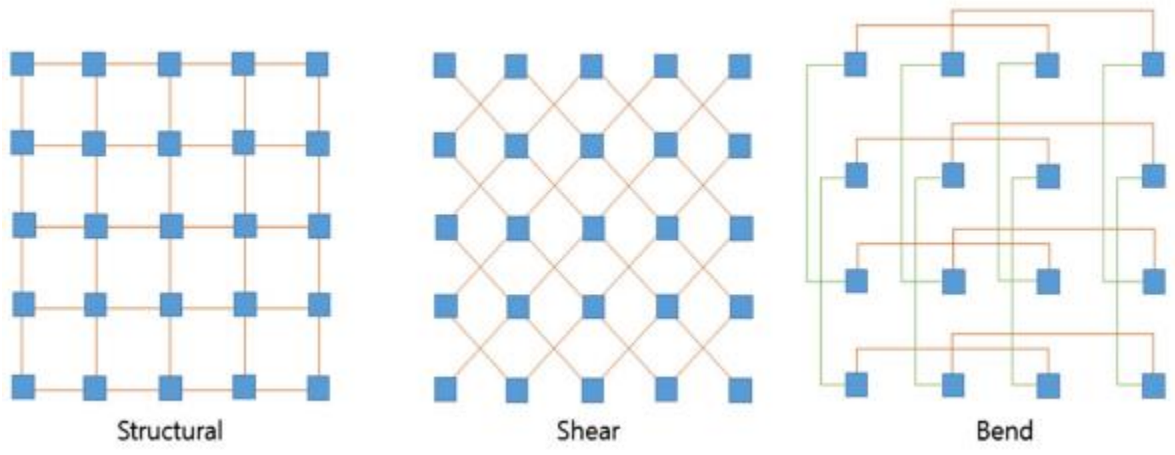

Figure 1 Three Spring System [2]

Each rectangle is represented as a node, or in this case a particle, which holds different information such as mass or velocity and position. The structural aspect of the system maintains the overall structure of the system[2]. The shear aspect maintains the cloth shape when an external force is applied in a diagonal direction[2]. The bend aspect maintains the cloth shape when it either folds or bend. To calculate the force on each spring system, the following *Figure 2* shows an equation where K_s is the spring stiffness coefficient, K_d is the spring dampness coefficient, L_0 is the distance of the two nodes and the position based of the nodes

$$f_1 = -\left[K_s(|p_1-p_2|-L_0) + K_d\left(\frac{(v_1-v_2) \times (p_1-p_2)}{|p_1-p_2|}\right)\right] \times \frac{(p_1-p_2)}{|p_1-p_2|} f_2 = -f_1$$

Figure 2 Equation for Spring Force [2]

Lastly with the force figure out, we can update the node through the 5 kinematic equation. However, there is a slight problem, finding a new position and the new velocity requires some sort of integration, luckily there are different technique of integrating explicitly such as Euler's method , Verlet Algorithm and Runge Kutta. They are all explicit, which doesn't guarantee full accuracy, but it does get the work done. In this case I choose Runge Kutta solely for benchmarking the CPU performance on a large-scale model.

Methods

With the theory figured out, implementing how one would make a cloth system gave a challenge. Creating the actual plot with particle and spring system wasn't too hard. For creating the particle and giving them a position would require two loops, and for the x value, just divide the inner loop iterator with the actual width and for the z value, divide the outer loop iterator with the height. Creating the mass spring system required more work. For the structure, I would have to check if inner loop iterator is less than the width, and the outer loop iterator is less than the height like so

```
//Set the structural springs
if (j < this->particleWidth - 1){
        …
}
if (i < this->particleHeight - 1){
        …
}
```

The shear and bending spring system can be made through the below code

```
//Set the shear springs
if (i > 0 && j < this->particleWidth - 1){
        …
}
if (i < this->particleHeight - 1 && j < this->particleWidth - 1){
        …
}


//Add Bending Springs
if (j < this->particleWidth - 2){
        …
}
if (i < this->particleHeight - 2){
        …
}
```

The next step is to solve the equation in *Figure 2*. The solution I have come up is solving the equation bit by bit as so

```
void ClothSpring::updateGeometry(float t){
        float length = glm::length(this->endParticles.first->getPosition() - this->endParticles.second-
>getPosition());

        glm::vec3 forceSpring;
        if (length == 0){
                forceSpring = glm::vec3(0.0f, 0.0f, 0.0f);
        }
        else{
                glm::vec3 coilAxis = glm::normalize(this->endParticles.first->getPosition() - this-
>endParticles.second->getPosition());
                float deltaLength;
                if (length > this->restLength){
        deltaLength = std::min<float>((length - this->restLength), ((1.1f) * this->restLength) - this-
>restLength);
                }
                else{
                        deltaLength = std::max<float>((length - this->restLength), ((0.9f) * this-
>restLength) - this->restLength);
                }

                forceSpring = this->springConstant * coilAxis * (deltaLength);
        }

        //Compute the dampening force
        glm::vec3 forceDampening1 = -this->dampConstant * this->endParticles.first->getVelocity();
        glm::vec3 forceDampening2 = -this->dampConstant * this->endParticles.second->getVelocity();

        this->endParticles.first->addForce(-forceSpring + forceDampening1);
        this->endParticles.second->addForce(forceSpring + forceDampening2);
}
```

Lastly, calculating the new velocity and position is done through the Runge Kutta algorithm at 4 degree[3] like so

```
void ClothParticle::updateGeometry(float deltaTime) {
	if (this->posFixed){
		this->velocity = glm::vec3(0.0f, 0.0f, 0.0f);
		this->acceleration = glm::vec3(0.0f, 0.0f, 0.0f);
		this->netForce = glm::vec3(0.0f, 0.0f, 0.0f);

		return;
	}

	glm::vec3 newPosition;
	glm::vec3 newVelocity;
	if (true){
		//based on https://www.intmath.com/differential-equations/12-runge-kutta-rk4-des.php
		glm::vec3 v1 = deltaTime * computeVelocity(0);
		glm::vec3 v2 = deltaTime * computeVelocity(0.5 * deltaTime);
		glm::vec3 v3 = deltaTime * computeVelocity(0.5 * deltaTime);
		glm::vec3 v4 = deltaTime * computeVelocity(deltaTime);
		newPosition = this->position + (1.0f / 6.0f) * (v1 + 2.0f * v2 + 2.0f * v3 + v4);

		glm::vec3 a1 = deltaTime * computeAcceleration(0, this->velocity);
		glm::vec3 a2 = deltaTime * computeAcceleration(0.5 * deltaTime, this->velocity + 0.5f *
a1);
		glm::vec3 a3 = deltaTime * computeAcceleration(0.5 * deltaTime, this->velocity + 0.5f *
a2);
		glm::vec3 a4 = deltaTime * computeAcceleration(deltaTime, this->velocity + a3);
		newVelocity = this->velocity + (1.0f / 6.0f) * (a1 + 2.0f * a2 + 2.0f * a3 + a4);

		this->acceleration = (newVelocity - this->velocity) / deltaTime;
	}
	else{
		//Compute new position and velocity by Euler integration
		glm::vec3 a_t = computeAcceleration(0, this->velocity);
		newPosition = this->position + this->velocity * deltaTime + 0.5f * a_t * deltaTime *
deltaTime;
		newVelocity = this->velocity + 0.5f * (a_t + computeAcceleration(deltaTime,
computeVelocity(deltaTime))) * deltaTime;
	}

	this->position = newPosition;
	this->velocity = newVelocity;

	this->netForce = glm::vec3(0.0f, 0.0f, 0.0f);
}
```

Results

When running the program, there were few issues that happened. Firstly, nothing was drawn on the screen. It turns out that I had an incorrect binding of vertex position. After fixing that, I had a problem were the cloth was rendered. But when running the simulation, it changed it form to a parallelogram and disappeared after a few moments. It turns out I wasn’t checking many cases for each position changes. On the third attempt on running the application, I have set two of the edge vertexes to be fixed in a sense like someone is holding it, but when the simulation started, the whole cloth beside the fixed vertex looked like it was falling. At this point in time, I couldn't figure out the problem. The similar problem can be seen here in *Figure 3*

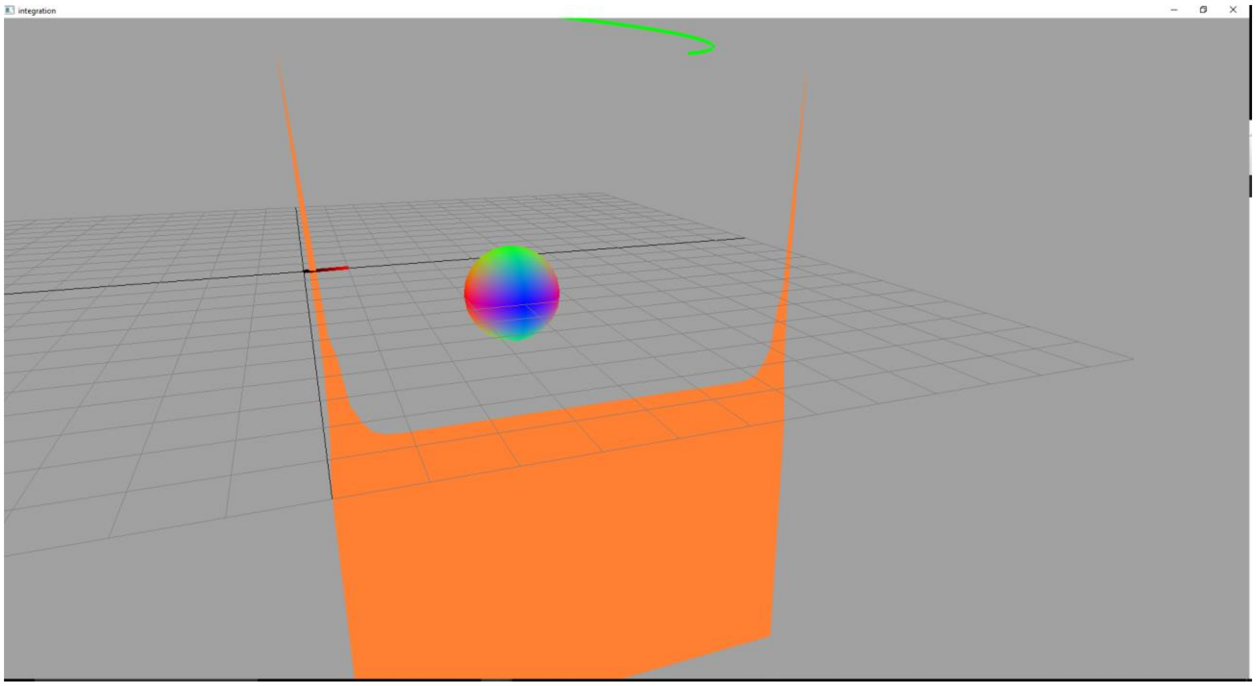

Figure 3 The whole Cloth system falling [4]

One person suggested that the k value is changing, however in my code, there was no k value that is being changed. I looked at pointer, and searched up better way of using them, and to my surprise, there are smart pointers in the GLM library. Somehow changing all the pointers to the smart pointer made it work.

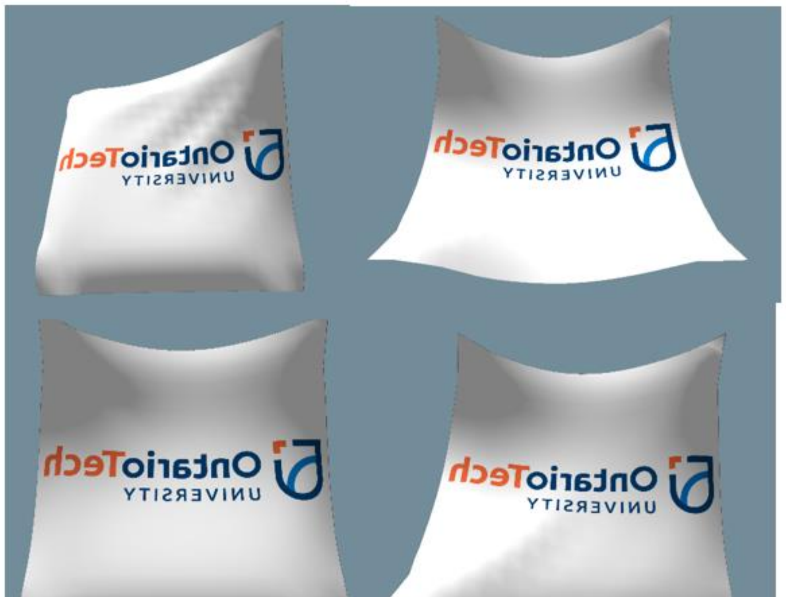

Figure 4 Top Left: 1 Corner Top Right: 4 Corner, Bottom Left 2 Corner, Bottom Right: 3 Corner

The resulting image *Figure 4* shows the cloth when its being hold on multiple corner. The result is as impressing, however when the cloth reaches the equilibrium state, it would start glitching, where it looks like it's being pulled, but it would jump up a bit. The reason being is that each frame, I am adding gravity, but I'm also resetting the netForce resulting in that jump.

The next result that should be discussed is how computationally expensive this is. Since all the physics is being done on the CPU by a single thread, and the GPU only rendering it, there are a huge bottleneck. When running on a machine with a I5 6300u with integrated Graphics, 8GB gave a 15fps performance. Running on a different machine that has i9 9900K and an RTX 2080ti FE and the files are on an NVMe Samsung drive resulted on a 30fps average performance.

Conclusion

Cloth simulation are a nice way on exploring the physics on a Spring mass system. As a result, the physics can be computationally expensive to compute. I believe to speed this up, requires the use of multithreading on the CPU or GPGPU programing through compute shaders. The glitching can be ironed out, and generally finding a faster integration method could benefit the computation time.

How to Use:

Everything is packaged in a way that it should run right out of the box. Just click on the solution file and run the program. I have also included 4 executables in a folder called Cloth Sims. The library that I have been using for the past assignment is used in this project. It called ImGui, which provides an immediate GUI system for the application. On this GUI, there is a start/pause simulation checkbox, checking it will start the simulation, while unchecking it will pause the simulation.

Bibliography

[1] Mass-Spring System. (n.d.). Retrieved from https://www.math24.net/mass-spring-system/.

[2] Choi, Y.-H., Hong, M., & Choi, Y.-J. (2018). Parallel cloth simulation with GPGPU. *Multimedia Tools and Applications*, *77*(22), 30105–30120. doi: 10.1007/s11042-018-6188-x

[3] Bourne, M. (n.d.). 12. Runge-Kutta (RK4) numerical solution for Differential Equations. Retrieved from https://www.intmath.com/differential-equations/12-runge-kutta-rk4-des.php.

[4] Nick, & NickNick. (2016, February 13). OpenGL Cloth simulation physics isn't working. Retrieved from https://stackoverflow.com/questions/35385167/opengl-cloth-simulation-physics-isnt-working.

https://graphics.stanford.edu/~mdfisher/cloth.html

[5] Matt's Webcorner. (n.d.). Retrieved from https://graphics.stanford.edu/~mdfisher/cloth.html.

[6] Baraff, D., & Witkin, A. (1998). Large steps in cloth simulation. *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques - SIGGRAPH 98*. doi: 10.1145/280814.280821