Super Smash Bros Shootout

CS184 Fall 2013 Final Project Report

Ting-Chun Wang (dw), Shuo Sun (ds), Gabriel Tan (ci), Meiqi Yang (ec)

Introduction

For our CS184 final project, we developed a 3D penalty shootout game that features realistic ball and cloth physics. Our primary aim was to replicate cloth behavior when a certain force is applied onto it. We did so by adopting the mass-spring model to simulate the behavior of the mesh/net. We then proceeded to add physics to simulate the motion of the soccer ball under gravity and different kicking forces. After that, we wrote collision algorithms to simulate the behavior of the ball when it collides with a goalkeeper or the soccer net. In addition, we planted two flags to demonstrate the behavior of our simulated cloth in wind.

Since this simulation is also a game, we rendered different characters for use as goalkeepers, which required the use of instancing. We designed a simple game-flow, which includes score-keeping and power-ups. Our game is a two-player game; one player controls the movement of the goalkeeper, and the other controls the direction and power of the ball. By making use of the ffmpeg library, we made it possible to render videos within the game. We also had textures mapped onto some of our objects (such as the flags) by using the FreeImage library. Finally, we sped up the real-time rendering of our scene by implementing vertex arrays.

Methodology

Our most important task was to first simulate cloth. After doing some preliminary literature review, we found a paper written by Xavier Provot that accurately

described the process of simulating cloth behavior realistically by applying deformation constraints. Having obtained a better understanding of how cloth is simulated, we then proceeded to split the project into individual tasks. First, we had to render a stadium, a soccer ball, as well as the goalkeepers in real-time. This was done with the help of instancing, and the implementation of vertex arrays to speed up the rendering process. Second, we had to create the physics of the ball and model its interaction with other objects (primarily collisions with a plane and a mesh). Lastly, we had to create and implement a fun game flow and graphic user interface that will attract users to play our game.

a. Cloth Simulation

We modeled the cloth using a mesh of $m \times n$ virtual masses, each mass being linked to its neighbors by massless springs. There are three kinds of linkages:

(1) Structural springs: springs linking masses [i, j] and [i+1, j], and masses [i, j] and [i, j+1].

(2) Shear springs: springs linking masses [i, j] and [i+1, j+1], and masses [i+1, j] and [i, j+1].

(3) Flexion springs: springs linking masses [i, j] and [i+2, j], and masses [i, j] and [i, j+2].

Assume each mass at time t is at position $P_{i,j}(t)$. Then at time $t + \Delta t$, we first calculate the forces applied to each point. There are two kinds of forces: internal force and external force. The internal force is caused by the tensions of the springs:

$$\boldsymbol{F}_{\text{int}}(P_{i,j}) = -\sum_{(k,l)\in R} K_{i,j,k,l}\left[\boldsymbol{l}_{i,j,k,l} - l^0_{i,j,k,l}\frac{\boldsymbol{l}_{i,j,k,l}}{\left\|\boldsymbol{l}_{i,j,k,l}\right\|}\right],$$

where $K_{i,j,k,l}$ is the stiffness of the spring, $l^0_{i,j,k,l}$ is the natural length of the spring, and $l_{i,j,k,l} = \overrightarrow{P_{i,j}P_{k,l}}$.

We identified three different external forces which would affect our simulation: (1) Gravity, which is calculated by the well-known formula, $\mathbf{F}_{gr}(P_{i,j}) = \mu \boldsymbol{g}$, where μ is the mass and $\boldsymbol{g}$ is the gravity acceleration; (2) Viscous damping, calculated by $\mathbf{F}_{dis}(P_{i,j}) = -C_{dis}\boldsymbol{v}_{i,j}$, where C_{dis} is the damping coefficient, and $\boldsymbol{v}_{i,j}$ is the velocity of the point; and (3) viscous interaction with wind, which is described by the equation $\mathbf{F}_{vi}(P_{i,j}) = C_{vi}\left[\boldsymbol{n}_{i,j} \cdot \left(\boldsymbol{u}_{fluid} - \boldsymbol{v}_{i,j}\right)\right]\boldsymbol{n}_{i,j}$, where $\boldsymbol{n}_{i,j}$ is the unit normal of the surface, and $\boldsymbol{u}_{fluid}$ is the velocity of the viscous fluid.

After summing all the forces, we compute the new position at time $t + \Delta t$ using the following kinematics equations:

$$\mathbf{a}_{i,j}(t + \Delta t) = \tfrac{1}{\mu}\mathbf{F}_{i,j}(t)$$

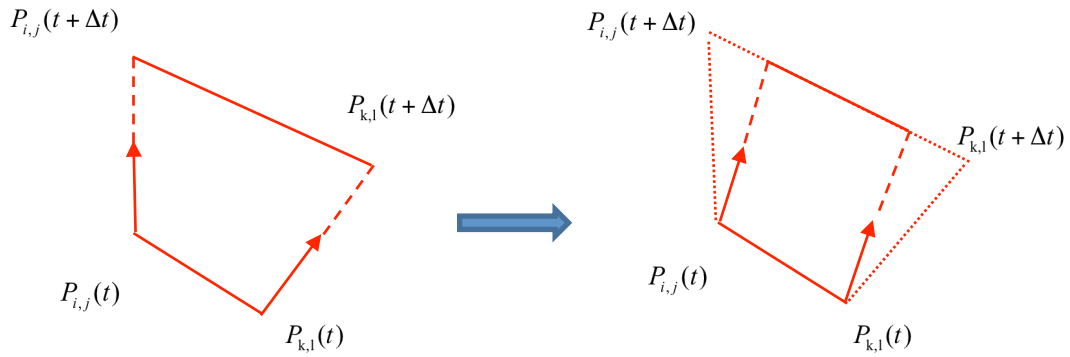
$$\mathbf{v}_{i,j}(t + \Delta t) = \mathbf{v}_{i,j}(t) + \Delta t\, \mathbf{a}_{i,j}(t + \Delta t)$$

$$P_{i,j}(t + \Delta t) = P_{i,j}(t) + \Delta t\, \mathbf{v}_{i,j}(t + \Delta t)$$

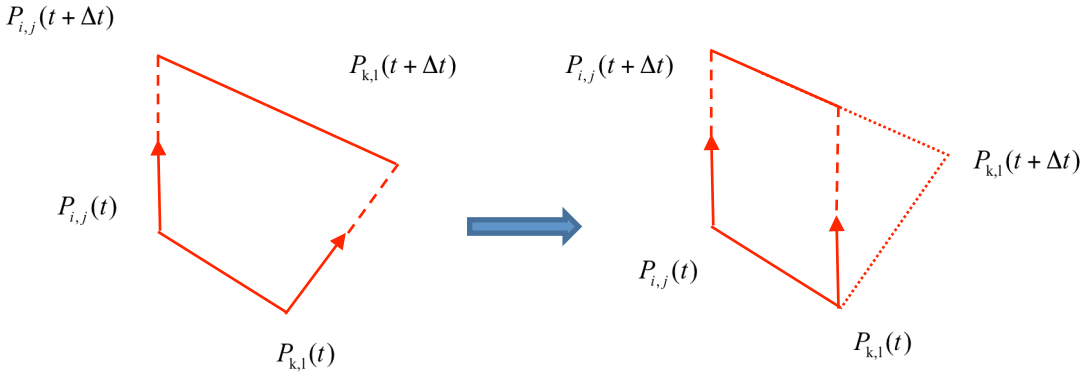
However, if we adopt the above procedure directly, the springs will have a “super elastic” behavior, leading to unrealistic elongation. For real woven fabrics, their elasticity is nonlinear, and their stiffness increases very rapidly when the deformation rate increases, which prevents them from having a high deformation rate. Thus, to more truthfully simulate the cloth in real world, we need to constrain the maximum deformation rate for our springs. If the deformation rate of a spring is greater than a critical deformation rate τ, then a dynamic inverse procedure is applied to the two ends of the spring so that its deformation rate exactly equals τ.

This can be classified into two cases: (1) both points are not fixed; or (2) only one of the points is fixed. If both points are fixed, we just leave them

unchanged. If both points are not fixed, they are evenly brought closer to their middle point until the deformation of the shrunk spring reaches τ,



If one of the ends is fixed, the other end is brought closer to the fixed end until the deformation of the shrunk spring reaches τ,

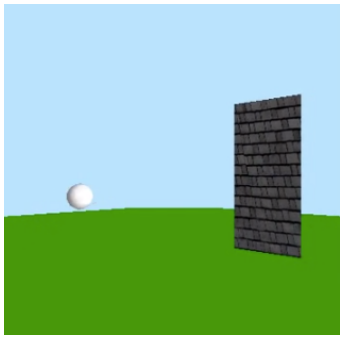


In practice, we chose τ = 10% for both structural and shear springs, and no constraint on flexion springs.

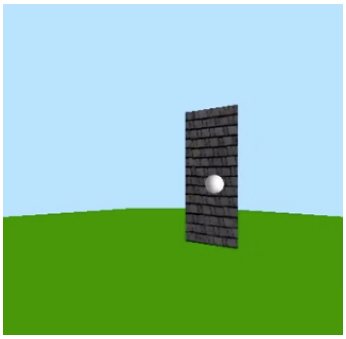
b. Basic Collision

Next, we worked on the basic physics of the ball and its interactions with its environment. Our first implementation had a ball moving towards a wall that is sliding from left to right. To determine whether a collision has occurred, we test for two conditions: (1) If the z-coordinate of the ball plus the ball radius is equal to the z-coordinate of the wall; and (2) if the x and y-coordinates of the ball are within the range of the x and y-positions of the

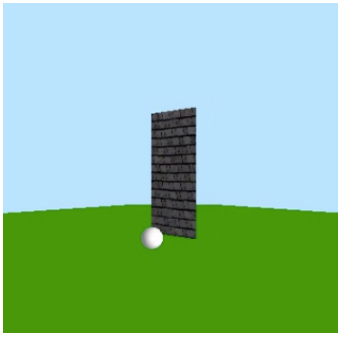
wall. Next, if the ball collides with the rectangular wall, it bounces off in a negative z-direction; otherwise, it goes through. The floor also acts as a wall, and the ball always bounces off it in a positive y-direction. The external forces acting on the ball are gravity and the initial push. Lastly, we accounted for losses in energy by making the ball lose a small amount of velocity after every bounce.



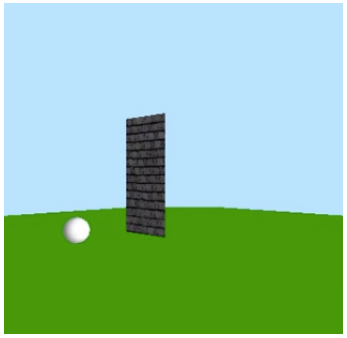
1. Ball moving towards wall



2. Ball colliding with the wall



3. Ball bouncing off floor



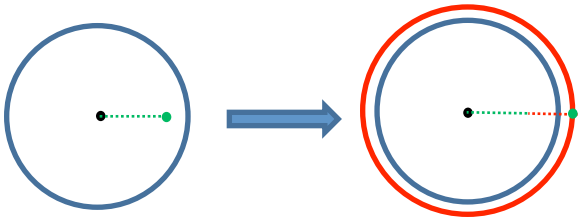
4. Ball after colliding with floor

c. Ball and Cloth Collision

For collision detection between the ball and the net, we test if the distance of each mass to the center of ball is smaller than the ball radius. If it is, the mass position is set to

$$P' = C + (\mathrm{r} + \delta)\frac{P - C}{\|P - C\|},$$

where C is the ball center position, r is the ball radius, P is the original position of the mass, and $\delta > 0$ is a parameter to prevent an infinite loop.



Also, a force is applied to the ball,

$$\mathbf{F} = k\left(r / \|C - P\|\right)\frac{C - P}{\|C - P\|},$$

where k is the bouncing force coefficient.

Since the position updating may fail the spring deformation rate constraint, we iteratively perform the deformation rate test and the collision detection until all springs satisfy both constraints.

d. Game Flow

Designing and implementing the flow of the game was rather straightforward (albeit time consuming), since there are many penalty shootout games out in the market for us to refer to. To make the gameplay smooth, we implemented interactive menus, keyboard controls, as well as in-game score counters. We also designed different special abilities for each of the goalkeepers.

e. Rendering

As we worked on the project, we anticipated the need to speed up our game due to the increasing number of simulations and object animations. Thus, we implemented instancing to allow a quicker rendering of our large object files (especially our stadium and goalkeepers). We also implemented a simple version of vertex arrays that greatly reduced the number of OpenGL calls needed to draw objects in the scene. To give an illustration, our final program ran at more than 40 frames per second with vertex arrays and less than 10

frames per second without. We even had to implement frame-control to make sure that the game was not running too quickly.

Since we had some time to spare, we decided to add in the ability to render videos within the game by using the ffmpeg library. The videos are read frame by frame, and each frame is mapped to an OpenGL texture. This gives us the flexibility to transform the video frames however we want. To further demonstrate our cloth simulation, we placed two flags beside the goalposts and had them interact with wind. The flags had textures mapped onto them by using the FreeImage library to read in .jpg files.

Limitations and Conclusions

As with most other projects, the greatest limitation we faced was time. We had a reasonable amount of time to work on the final project, but I believe that we would have produced something much more professional if given a couple more weeks.

One physical limitation of our project is the collision detection between the ball and the goalkeeper. We implemented this by creating a vertical plane that follows the movement of the rendered goalkeeper, and so the collisions were with that plane and not with the object itself. This has led to the possibility of fake collisions that might ruin the dynamics of the game. However, this implementation is still useful because it simplifies the collision algorithm and reduces the need for complicated calculations that might slow down the game. This illustrates the tradeoff between speed and the accuracy of the simulation.

In conclusion, we are thankful for being given the opportunity to work on such a challenging but fulfilling final project. Given the time constraint, it was a pity that we were not able to touch on other interesting topics such as fluid simulation or inverse kinematics. Nevertheless, working on the cloth simulation has taught us so much about the importance of math in the simulation of simple events that we take for

granted in our day-to-day lives. At the same time, we learnt first-hand that the process of developing a game is not an easy task to accomplish, especially when we have to include real-time simulations. Besides taking into account issues such as game mechanics, we had to consider aesthetics as well, so a decent amount of time was spent on fine-tuning the details of the application. Finally, we learnt the importance of encapsulation when working in a group of four people. Since we did our work individually, it became difficult and tedious when we had to put our code together into one coherent program. Thus, we attempted to create our own encapsulated classes, which were then used to complete the main program file.

Acknowledgements

We would like to acknowledge our GSI, Fu-Chung Huang, who gave us invaluable advice, as well as Professor James O'Brien for his lectures throughout this semester.

References

Provot, X. *Deformation Constraints in a Mass-Spring Model to Describe Rigid Cloth Behavior*

Lander, J. *Devil in the Blue Faceted Dress: Real-time Cloth Animation*

OpenGL Collision Detection. Retrieved from https://www.youtube.com/watch?v=Lg0kOoiCI80

Modeling Friction and Air Effects. Retrieved from https://www.youtube.com/watch?v=p5uhnSw8_Xw

Mosegaards Cloth Simulation Tutorial. Retrieved from http://cg.alexandra.dk/2009/06/02/mosegaards-cloth-simulation-coding-tutorial/

Appendix (Screenshots)

Image 1: Main Menu

Image 2: Character Selection Screen

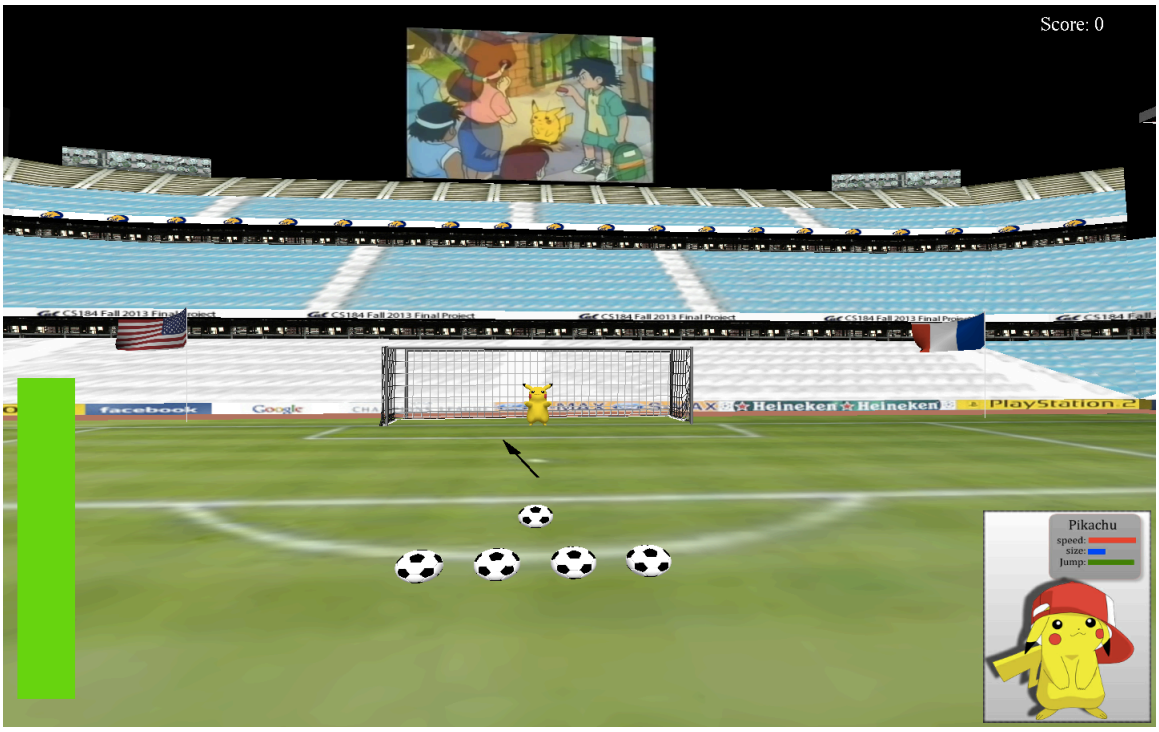


Image 3: Inside the game (with 2 flags)

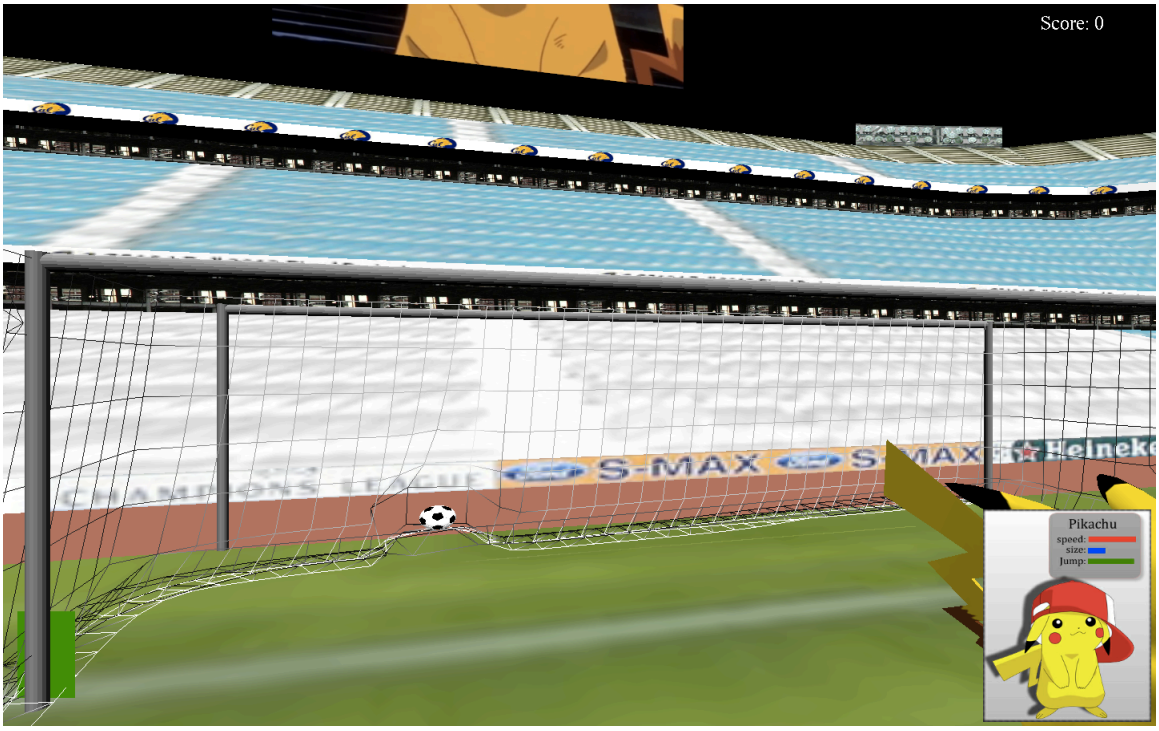


Image 4: Demonstration of net deformation after collision with the soccer ball