Final Project Report: Position Based Dynamics Football Net Simulator

Hassan Faour - Joe Abi Nassif - Mohammed Al Makdah

CMPS 299

Professor George Turkiyyah

Spring 2019/2020

**1- Introduction:**

For our final year project, our team decided to work in the field of computer graphics on Position Based Dynamics (PBD). PBD is an approach for simulating dynamics in computer graphics, it serves as an alternative for the usual force-based approach which accumulates external and internal forces, computes the object’s acceleration, and then computes and updates the new velocity and position of the object. PBD instead follows a different approach where each dynamic object is represented by vertices with mass, a position, and a velocity. PBD’s algorithm works by applying the forces on each vertex, projecting the desired constraints on the vertices, and finally computing and updating the position and velocity of these vertices.

For our project, we decided to implement this approach, and simulate these dynamics on cloth objects. We wanted to create a 3D football net simulator where a physics engine would utilize PBD on a ball, a net, and goal posts to mimic the collisions between them as they would appear in reality. We decided to use the Unity game engine which provides powerful tools to help game developers create cutting-edge, interactive games with ease.

**2- Designing the goal and net:**

Designing the goal post and the net were the first tasks we set on doing.

We used unity’s ProBuilder tool to model the goals post’s rigid body (figure 1).

As for the net, we were advised to use the Openmesh library to create a mesh out of vertices and edges. After installing Openmesh, we were able to create a simple mesh of a 3D Net resembling a goal. We then used Openflipper, which was created by the same institution as Openmesh, to model the net as an object which we exported for use in the Unity engine (figure 2).

We could have designed the mesh to be as detailed and goal-like as we would have liked, however our goal was to focus on the physics implementation. As a result, the simplistic net was later replaced by a prefabricated more eye-catching goal and net from the package “Football Goal Posts and Balls by Sim-3D” which we imported from Unity’s assets store.

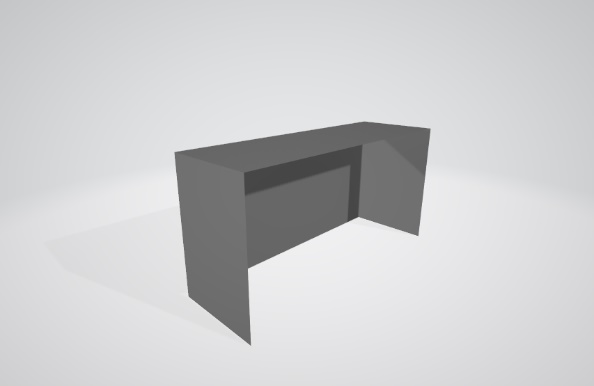
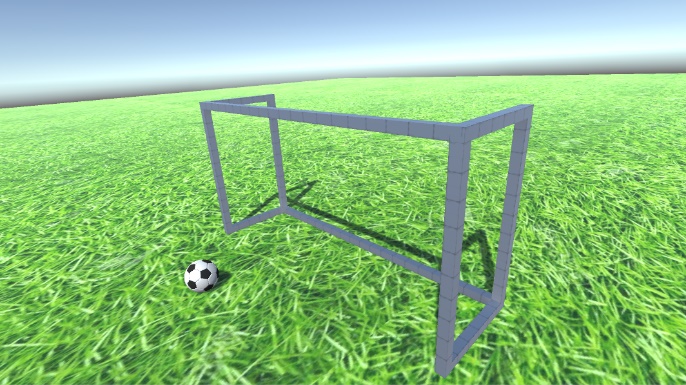
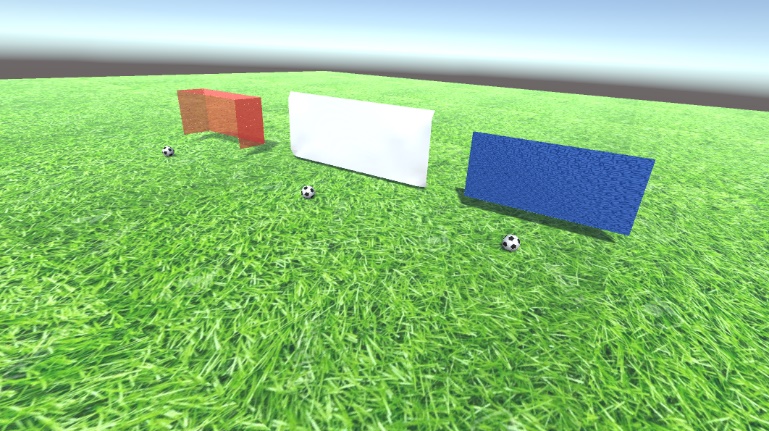


Figure : The goal post we designed using Unity’s ProBuilder

Figure : The net cloth we created using Openmesh and exported using Openflipper

**3- Implementing PBD into the mesh:**

For this step, we had several physics engine options to choose from, which were:

1. Obi Physics (Obi Cloth)
2. NVIDIA FleX
3. Unity’s built-in cloth component

To make sure we chose the best solution for our project, we tested all three of them.

We created a single scene where we put all 3 implementations together. We created a 3D plane for each engine, fixed these planes in their respective positions, and added footballs as rigid bodies, with a script that shot the balls at a net when the space bar was pressed. (figure 3)

Figure : Testing the 3 physics engines (from left to right: Nvidia, Unity's built-in cloth, Obi)

While all three of them acted almost in the same way and performed well in mimicking the behavior of a real-life cloth, we still had issues with the collision system of the nets. Specifically, that when a ball moved at high speeds or when we applied significant force to a ball, they kept passing through the cloth systems instead of being held by them. We tried to find workarounds online to try and solve this problem in each physics engine we were testing on. We did some research on collision detection to fix the problem, on topics such as interpenetration, AABB, and varied velocity settings.

With NVIDIA’s cloth, we eventually decided against that option as we could not find an acceptable solution for it. As for obi-physics, we were able to make the ball stop only if it was larger than the space between the vertices, and the collisions appeared more like rubber than net-like. And for unity’s built-in cloth component, we realized that its collision system was somewhat unreliable, as the ball would sometimes pass through and sometimes not, unpredictably, even without changing its settings or anything in the scene itself.

We were finally able to solve this by adding box colliders to the cloth, making it work at all times, realistically simulating how a cloth would react with collisions without any hiccups, giving us the effects we desired.

We therefore settled on unity’s built-in cloth component as our physics engine for the net to simulate PBD.

**4- Putting everything together:**

After successfully creating a working cloth with the desired behavior, it was finally time to put everything together and create our 3D football goal simulator.

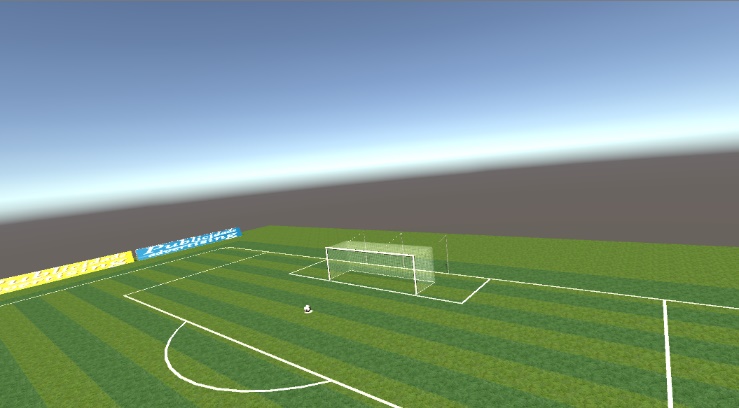
We started by setting up the scene, first with unity’s default lights and camera, and then by using the above mentioned package to import our stadium and the goal into our scene, and we also added the 3D rigid ball and positioned everything in its proper place (figure 4 and 5).

Figure : The setup of the scene



Figure : The initial camera view of our scene

Next, we added the cloth component to the net of this new goal and fixed the vertices of the edges to the goal posts (figure 6).

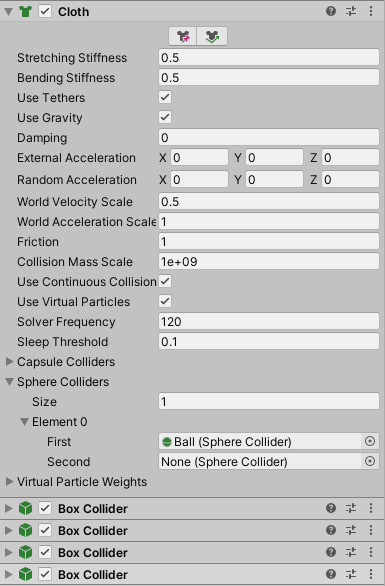
We then added a sphere collider reference of the ball to the net and made it interact with the ball on collision. And we set the variables of the net (stiffness, bending, damping, …) to give it the best effect possible (figure 7)

Figure 7: The initial settings and variables of our net cloth

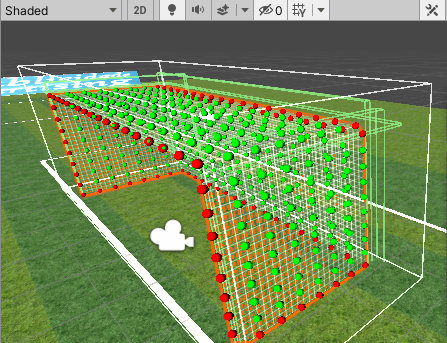
We then added colliders to this net, this was tricky since this net wasn’t just a regular one-sided plane like the one we tested on earlier, this one had 4 sides. So, what we did was creating four box colliders and positioning each collider at sweet spots over each side of the net. (figure 7)

Figure 6: The particles of the net, the red ones are fixed to their position

Now that the main part was over, we added more controls to the scene in order for us to monitor it better and control the variables without having to change it from inside the engine every time, in other words, we made the game ready to be exported into an executable file with controls available inside the game.

We set up a simple shooting system where the player can change the following:

* The initial position of the ball
* The direction of the shot
* The force of the shot
* The bending and stretching constraints of the net.
* The camera angle of view

That way, the player can witness the net interacting with the ball under different circumstances, such as differing cloth constraints as well as custom ball velocities. (figure 8 and figure 9)

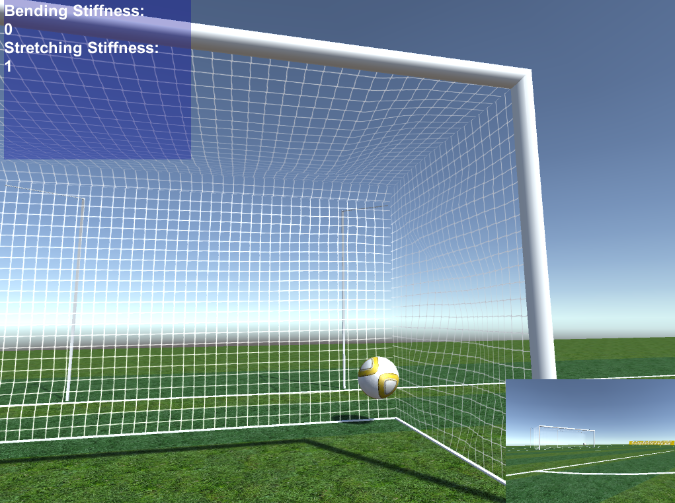


Figure 9: The ball after being shot shown in a different camera angle and with different net variables



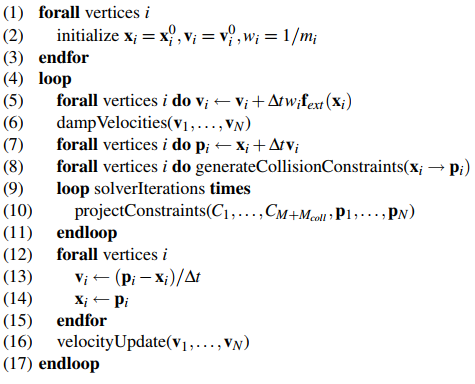
Figure 8: The shooting system of our game along with the controls and variables

**5- Limitations:**

The main limitation we faced while using unity for our simulation was the collision system. Our main problem was that the ball kept passing through the net, regardless of what physics engine we used. We were eventually able to solve this by adding box colliders to the net, but this wasn’t the primal solution we desired. Colliders have their downsides as they are simplistic, and intrinsically they have nothing to do with PBD, all they do is stop the ball from passing through a certain area. We were left with want of a better collision system; one we were not able to find for any of the physics engines we used in unity. A suggested solution is adding a collision script on the ball itself that allows it to interact better with the net. Another solution could be to increase the number and radius of the net’s particles as well as the size of the sphere collider and the number of substeps (iterations), which sacrifices performance and makes the ball look like it’s levitating on the ground since the collider is larger than the rendered mesh.

**6- The math behind PBD:**

The cloth being a 3D mesh consisting of vertices and edges, these vertices are the particles that undergo the physics simulation. Here is the main loop of the simulation (from line 4):



Lines 1-3 are to set the initial positions, velocities and inverse masses for each particle. (Making a particle fixed would be equivalent to setting its inverse mass to 0).

In line 5, external forces (usually gravity) are added to the velocities of each particle (notice that would not change for ). In line 6, the velocities are damped when necessary.

In line 7, each position is updated according to the corresponding velocity and stored in an estimated position (it’s estimated because constraints aren’t computed yet).

In line 8, collision constraints are generated.

In lines 9-11, the constraints are solved for the respective estimated positions.

In lines 12-15, the velocities are updated and the positions are set to the updated estimated positions.

In line 16, the velocities of colliding vertices are updated according to friction and restitution coefficients (how elastic the collision is).

Some of the constraints explained:

* Collision constraint:

For each vertex , let the ray coming from to .

If the ray enters an object, we check the following inequality constraint:

If the ray doesn’t penetrate another object, we check the following inequality constraint:

In general, to resolve a collision between a point and a triangle , we check the following inequality constraint:

To see on which side of the triangle is in. If , is above the triangle .

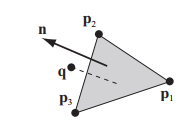


Figure 11: The point q is above the triangle (p1, p2, p3)

Additionally, we apply an impulse to a rigid body on collision with the cloth to get two-way interactions. That way, we can prevent interpenetration between the rigid body and the cloth.

* Stretching constraint:

For each pair of vertex positions forming an edge with length at rest:

* Bending constraint:

For each pair of triangles :

Let

.

.

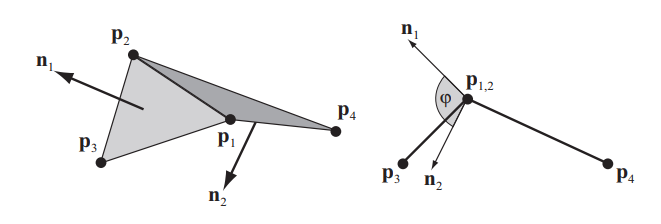


Figure 12: Two “bent” adjacent triangles

**7- Our Experience with unity:**

**Joe:**

I wanted to use Unity as it was the game engine I am most familiar with, as I made some small games and general experiments (including football-related) with it before. This, coupled with the abundance of tutorials, a large community, and a background in graphics from CMPS 285, made using the engine not a challenge.

The harder part was familiarizing myself with position-based dynamics as well as the details behind physics engines (notably PhysX which is the default for Unity). The biggest challenge was fixing the inter-penetration problem between the ball and the net.

**Mohammed:**

This wasn’t the first time I used unity, I previously tried using it to build some basic games on it, these games however barely involved using physics engines in them and I never thought of getting deep into how these engines work.

Building a real life simulation was more challenging than I thought, Starting with modeling the goal post, I created several rigid posts and had to position them perfectly in alignment with each others, and doing that in the 3D space was harder than I thought, this seemingly simple process took much time and effort and precision than I thought it would.

Testing to find the best physics engine was no easy task as well, I had to keep shooting the ball over and over and observe its behaviors on different positions and from different angles in order to understand how it behaves better.

**Hassan:**

My experience in Unity comes as an entirely new user to the engine. At first, I found it quite intimidating due to the huge amount of options there are to use. Even simple things like creating a scene looked tough at first, and there was difficulty inserting assets in the right place and navigating around the scene, it was a mess. Online videos were extremely helpful in explaining how to use the engine properly.

I found rolling updates of the engine to not be useful as we would have to re-import our project over and over. We should have stuck to one version of unity to begin with, which would have saved us a few compatibility issues.

Just from the engine itself, I found that individual features aren’t so well explained as is. Random testing of features on out project did not move me in the right direction, especially when it came to physics components. Playing with unknown features, while interesting in an of themselves, often broke things instead of helping. Tutorials online are a must.

Patience also important, as with all new things. After the steep beginner’s learning curve, it becomes easier and more intuitive to use the engine with time. Learning the correct terminology is also invaluable, and comes with time, allowing you to debug and narrow down issues online far easier.

Finally, an often overlooked goal is to enjoy development. Find a few interesting videos online and make a few test projects to try out the latest up and coming features. Fun and games are always a good method to stay stimulated and absorbed in the world of 3D animation.