Golf Without Friends

DD1354 Models and Simulation Project Report

Anders Steen
astee@kth.se
Michael Mathsson
mathsso@kth.se

March 15th 2019

Contents

1 Introduction **3**

2 Theoretical background **3**
2.1 Collision detection 3
2.1.1 Sphere and plane 3
2.1.2 Sphere and sphere 4
2.1.3 Sphere and mesh 4
2.2 Collision response 5
2.2.1 Friction 6
2.2.2 Implementation 6
2.2.3 Collision materials 7
2.3 Wind and air resistance 7

3 Method **7**
3.1 Tools and general method 7
3.2 Exclusion of concepts 8
3.3 Determining material property values 8

4 Results **9**

5 Conclusions **11**

6 Attachments **13**
6.1 Attachment 1: Initial Project Specification 13
6.2 Attachment 2: Final Project Specification 17

1 Introduction

This report concerns the project completed as part of the course *DD1354 Models and Simulation* at the Swedish Royal Institute of Technology, KTH. The report will explain the project, in particular the theoretical background, method, results and conclusions.

The project was done in a group of two people, the two authors of this report. Work was done together, with both members taking equal responsibility of all parts of the project.

See attachment 2 for the final project specification, containing problem definition and details about implemented features. The initial project specification can be seen in Attachment 1. The progress during the project can be tracked on the project blog [3].

2 Theoretical background

2.1 Collision detection

2.1.1 Sphere and plane

The method used for collision detection is the one introduced in one of the lectures. Translating the method into unity produces a function that takes a plane and the ball's position as parameters and returns the distance between them. To check if a collision has occurred you simply need to check if the distance between them is shorter than the ball's radius.

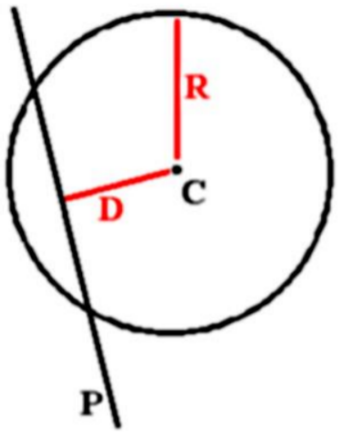

Figure 1: A diagram showing a collision between a sphere and a plane.

The distance D of Figure 1 can be calculated by comparing the result of the formula below with the radius of the sphere.

$$\frac{Ax + By + Cz + D}{\sqrt{A^2 + B^2 + C^2}}$$

The formula above is translated into code, which can look like the listing below.

```
float distanceFromPlane(Plane plane, Vector3 position) {
       float A = plane.transform.up.x;
       float Ax = A * position.x;
       float B = plane.transform.up.y;
       float By = B * position.y;
       float C = plane.transform.up.z;
       float Cz = C * position.z;
       float D = -Vector3.Dot(plane.transform.up, plane.transform
           ↪ .position);
       return (Ax + By + Cz + D) / (float)System.Math.Sqrt(A * A
           ↪ + B * B + C * C);
}
```

To keep the ball from actually going inside the plane one can use the difference between the ball's radius and the distance to the plane. If the distance is smaller than the radius the distance is simply increased be equal to the radius.

2.1.2 Sphere and sphere

The collision detection between two spheres is simple. One only needs to calculate the vector between the two ball's position and compare the magnitude of that vector to the sum of the radii of the two balls. If the distance is lesser than the sum of the radii it means that the spheres have collided with each other.

2.1.3 Sphere and mesh

The problem with using the sphere to plane function mentioned above is that it considers the plane to be of infinite size. This would obviously not work when used on any type of terrain. At first we considered using some kind of grid system where the ball would only check the distance to the planes in the same grid cube but ultimately we felt that this approach would be too limiting to our golf course design. What we did instead was that we designed a function to check whether the ball was within the boundaries of the plane as to ascertain whether or not the ball should be affected by the plane. This is not technically a mesh, but a collection of finite planes placed in a way that makes them resemble a mesh.

To check that the ball was within the bounds of a plane, first the points in the center of each edge was calculated. Then two lines were drawn straight across the plane between the pair of points. Vectors from each of the points to the ball were calculated and projected onto the line which the point in question was on. If all the projected vectors were shorter in length than the length of

the lines then the ball is considered within the bounds of the plane, meaning collision can be detected.

The method described above will allow planes to have arbitrary proportions and rotations. In the final simulation however, no rotation around the y-axis occurs, so the method is somewhat redundant in that aspect.

2.2 Collision response

The method used is an impulse based reaction model that simulates the collision between two objects.

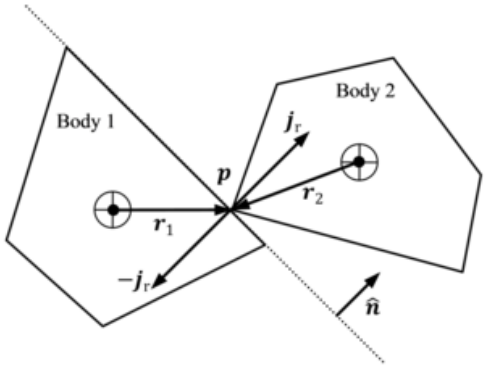

Figure 2: A diagram showing a collision between two objects in the impulse based collision model.

When using this model you start by calculating the magnitude of the impulse vector j_r. This is done using the formula below [4]:

$$j_r = \frac{-(1+e)\mathbf{v}_r \cdot \hat{\mathbf{n}}}{m_1^{-1} + m_2^{-1} + (\mathbf{I}_1^{-1}(\mathbf{r}_1 \times \hat{\mathbf{n}}) \times \mathbf{r}_1 + \mathbf{I}_2^{-1}(\mathbf{r}_2 \times \hat{\mathbf{n}}) \times \mathbf{r}_2) \cdot \hat{\mathbf{n}}}$$

In the formula, m_1 and m_2 are the masses of the two objects, $\mathbf{r}_1$ and $\mathbf{r}_2$ are the vectors between the contact point and the centers of mass, $\mathbf{I}_1$ and $\mathbf{I}_2$ are the inertia tensors of the objects, $\hat{\mathbf{n}}$ is the normal of the collision, e is the coefficient of restitution and $\mathbf{v}_r$ is the relative velocity [4].

The relative velocity is calculated using $\mathbf{v}_r = \mathbf{v}_{p2} - \mathbf{v}_{p1}$ where $\mathbf{v}_{pi}$ is calculated using $\mathbf{v}_{pi} = \mathbf{v}_i + \omega_i \times \mathbf{r}_i$ where ω_i is the angular velocity of the object [4].

The impulse vector is finally used to calculate the new velocities and angular velocities using the following equations [4]:

$$\mathbf{v}_1' = \mathbf{v}_1 - \frac{j_r}{m_1}\hat{\mathbf{n}}$$

$$\mathbf{v}_2' = \mathbf{v}_2 + \frac{j_r}{m_2}\hat{\mathbf{n}}$$

$$\omega_1' = \omega_1 - j_r \mathbf{I}_1^{-1}(\mathbf{r}_1 \times \hat{\mathbf{n}})$$

$$\omega_2' = \omega_2 - j_r \mathbf{I}_2^{-1}(\mathbf{r}_2 \times \hat{\mathbf{n}})$$

2.2.1 Friction

The friction is also calculated using an impulse based model based on the Coulomb friction model which uses coefficients of static friction (μ_s) and dynamic friction (μ_d). The object in question needs to overcome the static friction in order to move and if it does it gets affected by the dynamic friction. After translating these in terms of impulse you get the equations $j_s = \mu_s j_r$ and $j_d = \mu_d j_r$. Since the friction impulse vector is calculated differently depending on whether the force is strong enough to overcome the static friction we get the following equations [4]:

$$\mathbf{j}_f = \begin{cases} -(m\mathbf{v}_r \cdot \hat{\mathbf{t}})\hat{\mathbf{t}} & \mathbf{v}_r \cdot \hat{\mathbf{t}} = 0 \quad m\mathbf{v}_r \cdot \hat{\mathbf{t}} \leq j_s \\ -j_d \hat{\mathbf{t}} & \text{(otherwise)} \end{cases}$$

Where $\hat{\mathbf{t}}$ is the tangent vector calculated using [4]:

$$\hat{\mathbf{t}} = \begin{cases} \frac{\mathbf{v}_r - (\mathbf{v}_r \cdot \hat{\mathbf{n}})\hat{\mathbf{n}}}{|\mathbf{v}_r - (\mathbf{v}_r \cdot \hat{\mathbf{n}})\hat{\mathbf{n}}|} & \mathbf{v}_r \cdot \hat{\mathbf{n}} \neq 0 \\ \frac{\mathbf{f}_e - (\mathbf{f}_e \cdot \hat{\mathbf{n}})\hat{\mathbf{n}}}{|\mathbf{f}_e - (\mathbf{f}_e \cdot \hat{\mathbf{n}})\hat{\mathbf{n}}|} & \mathbf{v}_r \cdot \hat{\mathbf{n}} = 0 \quad \mathbf{f}_e \cdot \hat{\mathbf{n}} \neq 0 \\ \mathbf{0} & \mathbf{v}_r \cdot \hat{\mathbf{n}} = 0 \quad \mathbf{f}_e \cdot \hat{\mathbf{n}} = 0 \end{cases}$$

2.2.2 Implementation

The impulse based reaction model is used to calculate what happens when two arbitrary objects collide, but since our project only uses collisions between two balls and collisions between a ball and a plane we were able to simplify the equations. Since the planes are immovable and represents the earth we didn't have to change their velocity or angular velocity. And when the ball collides with a plane the mass of the plane and its inertia tensor can be assumed infinite which means that any product that uses the inverse of one of these as a factor always tends to zero. This means that the terms using these products do not have to be calculated. Another thing of note is that since the balls are perfectly round, the impact vector $\mathbf{r}$ is always going to run parallel with the normal, which means that $\mathbf{r} \times \hat{\mathbf{n}}$ is always going to be zero. This means that the terms using $\mathbf{r} \times \hat{\mathbf{n}}$ as a factor can also be removed.

Unfortunately this also causes a serious problem. Since the angular velocity only changes with a factor of $\mathbf{r} \times \hat{\mathbf{n}}$ it means that the angular velocity does not change at all. Since it would be unnatural for the ball to not have any changes in rotation we tried to calculate the new angular velocity by applying the friction vector as the torque in the torque equation $\tau = I\alpha$ where τ is the torque, I is the moment of intertia and α is the angular acceleration [5].

Although it sounded reasonable we could not get it to work properly. Due to time constraints we decided to settle for a less realistic solution and just apply a percentile loss of angular momentum based on how high the friction of the plane is. This unfortunately also means that the ball will not gain any angular velocity as it starts rolling down a hill.

2.2.3 Collision materials

Every plane in the simulation needed to have a material assigned to it, in order to allow for different collision response to different materials. Each material would need to have their coefficient of restitution, static and dynamic friction constants and angular friction constant defined. In order to render the materials different, there would need to be a rendering material associated with each collision material.

2.3 Wind and air resistance

There are multiple different levels of simulating wind. Advanced numerical methods can be used, for instance by solving the incompressible Navier-Stokes equations:

$$\frac{\partial u}{\partial t} + (u \cdot \nabla)u - \frac{\mu}{\rho_0}\nabla^2 u = -\nabla w + g$$

The equation is a differential equation that describes liquid flow. Solving the equation is a heavy computational task, and cannot easily be done in real-time simulations. The solution can be calculated in advance and the results used in a real-time simulation, but this will not allow for a dynamic, random behaviour [2].

In order to simulate wind in a real-time simulation, some approximations of wind behaviour have to be made to acquire a sufficiently simple method. The wind simulation method in this project was to simply add a horizontal force on the golf ball. The magnitude of the force depends on the ball's height above the ground plane. The direction and magnitude of the force would be randomised within reasonable ranges every ten seconds to simulate a changing wind.

Air resistance can be simulated as a force acting opposite to the velocity, and is described by this formula:

$$F_{air} = -cv_{abs}$$

where c is a constant and v_{abs} is the absolute velocity of the object [1].

3 Method

3.1 Tools and general method

The simulation was made using the Unity Game Engine. Unity contains many built-in assets such as collision detection and wind simulation that could be use

to create the simulation. Our problem definition required us to implement these concepts ourselves, and therefore the built-in assets were to be used minimally. Unity was only used for the purpose of rendering, visualising and making testing and tweaking easier.

Github was used in the development process to manage versions, keep track of development and sharing files between group members.

Development started with the implementation of the features listed in Attachment 1 in rough priority order. Sessions to work on the project were planned in an ad hoc fashion. The sessions consisted of sitting down together in the group, either in person or using an online conversation, and working on implementing features together. The general method was incremental, so the goal for each session was to have some completed work that could be presented in a blog post. Each blog post corresponds to a session [3].

3.2 Exclusion of concepts

Relevant concepts described in 2 were researched by searching for descriptions of the concepts online. As can be seen in Attachment 2, some of the concepts described in 2 were not implemented into the simulation. The decisions to not include certain concepts had multiple different reasons.

Implementing collision detection between spheres and meshes was deemed unnecessary. The other types of collision detection were implemented first, and although a mesh could have been used to create more realistic terrain, planes worked sufficiently well. Within the limited time frame of the project, mesh collision was not considered a high priority considering its complexity.

The exclusion of water as a material in the simulation was based in the fact that the other materials were implemented first, were sufficient, and that water was considerably more difficult to implement. The limited time frame also played a role in the exclusion.

From the start, we knew that it would be difficult and therefore time consuming to implement wind using the Navier-Stokes equations. This would have required us to create a separate simulation where the wind is calculated in advanced. The results from that simulation would then have to be integrated into the main simulation. This was something we considered being of low priority at first, and soon realised that it would be impossible to complete within the time frame.

3.3 Determining material property values

The static, dynamic and angular friction and energy absorption values for each type of material had to be set manually. There were attempts to find reliable values online to achieve true empirical realism in the simulation, but no reliable data for this type of application could be found. The decision was therefore made to opt for a simulation of perceived reality, rather than true reality. Values were set to make the materials behave in a way corresponding to how they would be expected to behave. For instance, the energy absorption of sand was set to

100%. This was based on our own expectation of how a ball would act upon landing in sand, stopping immediately in this case. This was done for all values until a satisfactory level of realism in the simulation could be perceived.

4 Results

The final simulation is the result of implementing the features listed in Attachment 2. Figure 3, Figure 4 and Figure 5 show screenshots from the running simulation. Figure 6 and Figure 7 show screenshots from the project inside the Unity editor. Videos of the complete simulation and earlier prototypes can be seen on the project blog [3].

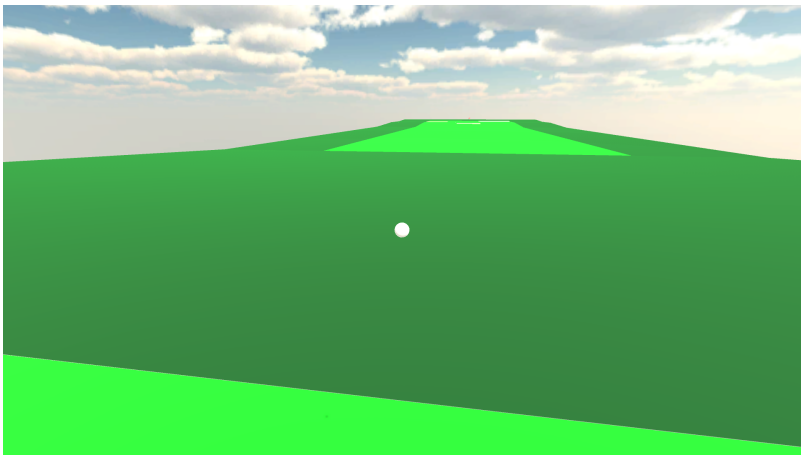

Figure 3: The golf course in the simulation seen from slightly beyond the starting position. The different colors of green colored materials represent rough, fairway and green in order of ascending brightness. The flag is marks the spot where the hole is.

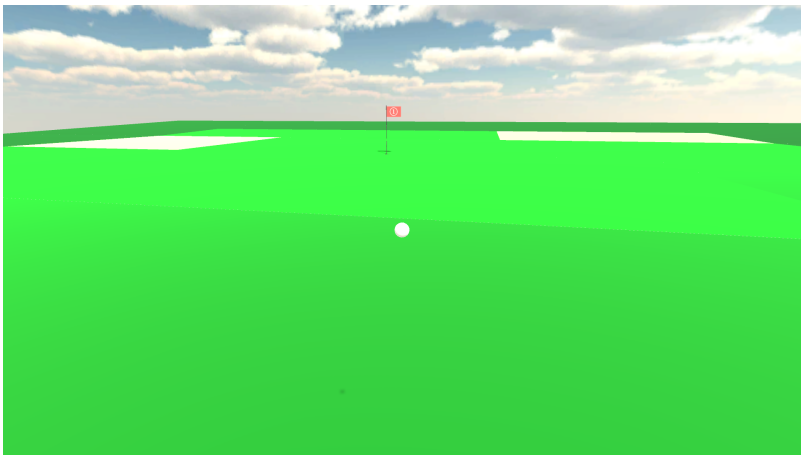

Figure 4: The area at the end of the golf course. The beige material is represents sand.

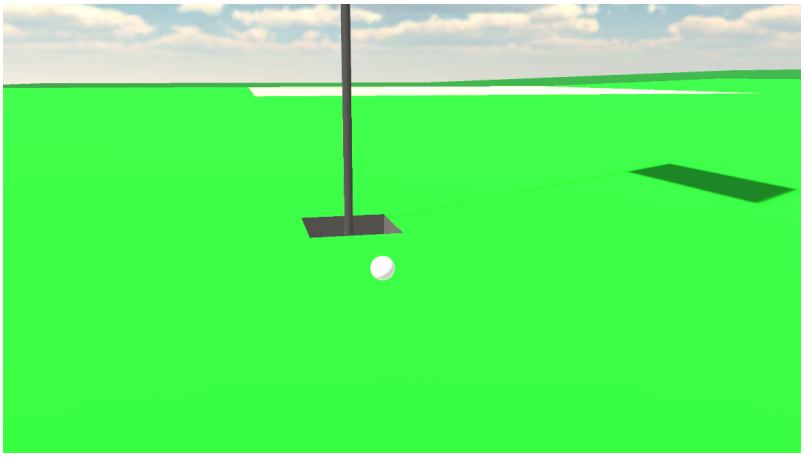

Figure 5: The ball in close proximity to the hole in the golf course.

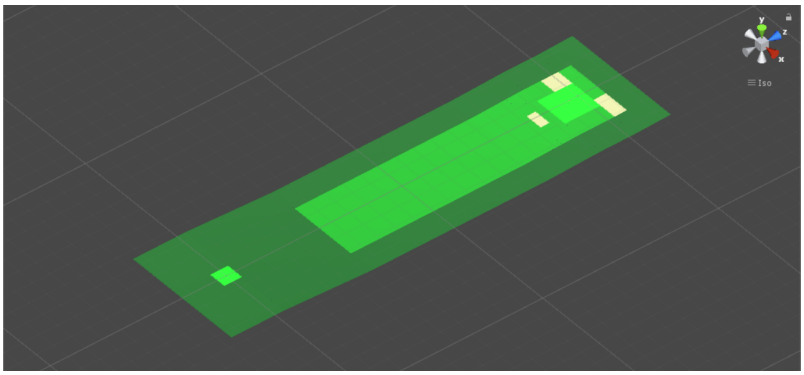

Figure 6: The golf course seen from the scene view in the Unity editor.

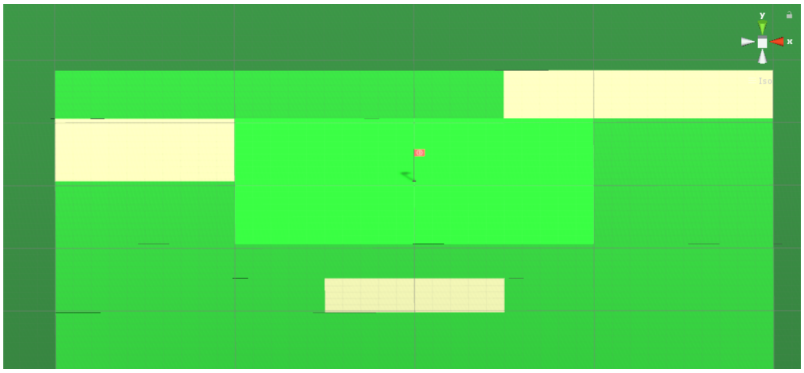

Figure 7: The hole area seen from the scene view in the Unity editor.

5 Conclusions

The conclusions of this project will concern mostly our thoughts about the results in regards to what was planned.

When planning the project features and setting the scope of the project, great care was taken to ensure that the scope, topic and level of ambition was appropriate for the project and especially for the target grade. This meant requesting feedback from teaching assistants multiple times during the planning of the project. The thought behind this was to correct our plans frequently, instead of planning far ahead, receiving feedback and having to make large corrections. The result of this planning process is the initial project specification seen in Attachment 1.

It was known to us that some of the features (such as water and Navier-Stokes wind simulation) would be unlikely to be implemented within the time frame. Therefore, not much action was taken to make sure that there would be enough

time to implement those features. In the final project specification, which can be seen in Attachment 2, those features have been removed. Evidently, not many of the features were removed, and those that were removed were seen as peripheral features from the start.

The conclusion to be made from this is that our planning and specification work was effective. This conclusion is only valid assuming the project receives the target grade, which according to received feedback, is likely.

One core feature that was not possible to fully implement in time was the advanced friction mentioned in section 2.2.1. The reason for this was that the time frame did not allow us to design a robust physical simulation system, leading us to create a system that was difficult to adapt to arbitrary physical concepts. This was disappointing, since the simulation would have a more realistic behaviour had we implemented it, and also since we believed it possible to implement at the start. Instead, the concept was simplified.

Although it is unfortunate that the concept could not be implemented, this was anticipated in the initial project specification. The last row of the Risks and challenges table specifies the mitigation plan of simplifying unexpectedly advanced concepts. The conclusion is that making mitigation plans is a good idea, but also to that care should be taken to construct robust systems early in development, so that concepts can be more easily implemented.

References

[1] *DD1354 Models and Simulation Lab 2 Specification*. From spring term of 2019.

[2] *DD1354 Models and Simulation Lab 3 Specification*. From spring term of 2019.

[3] *Golf Without Friends Project Blog*. `https://modsimspheres.blogspot.com/`. Accessed: 2019-03-14.

[4] *Impulse-based reaction and friction model*. `https://en.wikipedia.org/wiki/Collision_response#Impulse-based_reaction_model`. Accessed: 2019-03-15.

[5] *Torque Formula (Moment of Inertia and Angular Acceleration)*. `http://www.softschools.com/formulas/physics/torque_formula/59/`. Accessed: 2019-03-15.

6 Attachments

6.1 Attachment 1: Initial Project Specification

Golf Course Simulation using Unity

Specification of project in DD1354 Models and Simulation,
Royal Institute of Technology KTH

Michael Mathsson mathsso@kth.se

Anders Steen astee@kth.se

Grade ambition: C

1 Background

Collision detection is a physical phenomenon that is very naturally part of our world. Therefore, a physical simulation that does not implement collision detection will be extremely lacking. Multiple different methods exist for simulating collisions, with varying sophistication. The geometry of an object determines the best way of detecting collisions. Spheres, e.g golf balls, are defined by a point in space and a radius. Detecting collisions between spheres simply means checking whether the points of two spheres are closer to each other than the sum of their radii.

Simulating realistic collisions does not only involve correctly detecting the collisions, but the simulation responding to the collision in a realistic manner. Objects bouncing off surfaces should have their momentum changed according to certain rules. For instance, a golf ball landing on grass will bounce in a way distinct from a golf ball landing in sand.

The interest for this kind of simulation came from lectures in the course and a collision detection project done by other students. Another source of inspiration is games such as Wii Sports Golf, which implements collision detection and reaction in a real-time gameplay setting.

2 Problem

Implementing collision detection between a sphere and a plane or mesh, as well as simulating realistic collision reaction is the initial problem. This means changing an object's momentum according to attack angle, velocity and material of the colliding objects. Achieving further realism is the secondary problem. Implementing wind simulation and creating a realistic golf course environment are included in this secondary problem.

3 Implementation

The first implementation goal is to create working collision detection and reaction between a sphere and a plane and mesh. This will be done in Unity using a combination of Unity's built-in objects and our own collision detection methods. As few Unity assets as possible will be used, they are only meant to provide visuals for the simulation.

Figure 1

After the implementation of the core features is complete, more variation will be introduced. This will be in the form of different materials of colliding objects creating different collision reactions. Air resistance and wind changing the momentum of golf balls will also be implemented if time permits.

The final result will be a real-time simulation in Unity that handles collision detection and reaction in a golf course setting. The realism of the simulation will depend on how difficult the core features turn out to be to implement. See Figure 1 for a conceptual picture of the best-case end result.

3.1 Specification ideas

Our planned features, in descending order of relevance and priority.

- Collision detection
    - Sphere to plane
    - Sphere to sphere
    - Sphere to mesh
- Realistic bouncing
    - Correct angle
    - Realistic conservation of energy
- Collision with different materials
    - Bounciness varying based on terrain
    - Friction when rolling
    - Materials:
        - Grass
            - Rough
            - Green
            - Fairway
        - Sand
        - Water
- Wind simulation
    - Air resistance
        - Specific to the shape of a golf ball
    - Affecting golf balls
    - Calculated using rigorous methods such as Navier-Stokes equations
- Aesthetically pleasing environment

4 Risks and challenges

Below is a table describing potential risks and challenges specific to this project.

Description	Contingency plan
Finding accurate physics equations	If we can't figure out how it works in reality we can complement it with pseudo-physics.
Combining Unity assets with our code	Either use more of Unity's assets or make our own assets and settle for an ugly simulation and more work required.
Some physical phenomenon are difficult to implement	Those difficult features can be discarded from the project, be simplified or possibly included through unity assets.

5 References

1. Project blog: https://modsimspheres.blogspot.com/
2. Collision detection in course lecture slides: https://www.kth.se/social/files/5c5c028f56be5b3639e2695f/DD1354%20Particle%20Systems%20and%20Collision%20Detection%202019.pdf
3. Descriptions of collision reaction:
    a. https://en.wikipedia.org/wiki/Inelastic_collision
    b. https://en.wikipedia.org/wiki/Momentum#Conservation
4. Wii Sports Golf example: https://www.youtube.com/watch?v=-Mm7xoaTzdM

6.2 Attachment 2: Final Project Specification

Golf Course Simulation using Unity

Specification of project in DD1354 Models and Simulation,
Royal Institute of Technology KTH

Michael Mathsson mathsso@kth.se

Anders Steen astee@kth.se

Grade ambition: C

1 Background

Collision detection is a physical phenomenon that is very naturally part of our world. Therefore, a physical simulation that does not implement collision detection will be extremely lacking. Multiple different methods exist for simulating collisions, with varying sophistication. The geometry of an object determines the best way of detecting collisions. Spheres, e.g golf balls, are defined by a point in space and a radius. Detecting collisions between spheres simply means checking whether the points of two spheres are closer to each other than the sum of their radii.

Simulating realistic collisions does not only involve correctly detecting the collisions, but the simulation responding to the collision in a realistic manner. Objects bouncing off surfaces should have their momentum changed according to certain rules. For instance, a golf ball landing on grass will bounce in a way distinct from a golf ball landing in sand.

The interest for this kind of simulation came from lectures in the course and a collision detection project done by other students. Another source of inspiration is games such as Wii Sports Golf, which implements collision detection and response in a real-time gameplay setting.

2 Problem

Implementing collision detection between a sphere and a plane or mesh, as well as simulating realistic collision response is the initial problem. This means changing an object's momentum according to attack angle, velocity and material of the colliding objects. Achieving further realism is the secondary problem. Implementing wind simulation and creating a realistic golf course environment are included in this secondary problem.

3 Implementation

The first implementation goal is to create working collision detection and response between a sphere and a plane and mesh. This will be done in Unity using a combination of Unity's built-in objects and our own collision detection methods. As few Unity assets as possible will be used, they are only meant to provide visuals for the simulation.

After the implementation of the core features is complete, more variation will be introduced. This will be in the form of different materials of colliding objects creating different collision responses. Air resistance and wind changing the momentum of golf balls will also be implemented if time permits.

Figure 1

The final result will be a real-time simulation in Unity that handles collision detection and response in a golf course setting. The realism of the simulation will depend on how difficult the core features turn out to be to implement. See Figure 1 for a conceptual picture of the best-case end result.

3.1 Features

Our implemented features, in descending order of relevance and priority.

- Collision detection
    - Sphere to plane
    - Sphere to sphere
- Realistic bouncing
    - Correct angle
    - Realistic conservation of energy
- Collision with different materials
    - Bounciness varying based on terrain
    - Friction when rolling
    - Materials:
        - Grass
            - Rough
            - Green
            - Fairway
        - Sand
- Wind simulation
    - Air resistance
        - Generally for spherical objects
    - Affecting golf balls
- Simple golf course environment

4 Risks and challenges

Below is a table describing potential risks and challenges specific to this project.

Description	Contingency plan
Finding accurate physics equations	If we can't figure out how it works in reality we can complement it with pseudo-physics.
Combining Unity assets with our code	Either use more of Unity's assets or make our own assets and settle for an ugly simulation and more work required.
Some physical phenomenon are difficult to implement	Those difficult features can be discarded from the project, be simplified or possibly included through unity assets.

5 References

1. Project blog: https://modsimspheres.blogspot.com/
2. Collision detection in course lecture slides: https://www.kth.se/social/files/5c5c028f56be5b3639e2695f/DD1354%20Particle%20Systems%20and%20Collision%20Detection%202019.pdf
3. Descriptions of collision response:
    a. https://en.wikipedia.org/wiki/Inelastic_collision
    b. https://en.wikipedia.org/wiki/Momentum#Conservation
4. Wii Sports Golf example: https://www.youtube.com/watch?v=-Mm7xoaTzdM