Projectiles - Individual report

1. Introduction

The project provided an interesting challenge. Working on an already existing game engine, and trying to familiarize myself with the inns and outs of working with a rendering engine. As my part was focused on the physics I also had the opportunity of working with several different physics engines.

2. Assignment

2.1. Extension Outline

My extension was focused on projectiles. Trying to simplify the use of different projectiles like bullets, shells and grenades. The different behavior patterns of these elements provided a unique opportunity to explore quite a few of the possibilities of the OGRE combined with nVidias PhysX API. The idea behind the extension was at first to only make it possible to use projectiles. But later evolved into not only making the use of projectiles, but also simplifying it for further use in any possible extensions. I realized that functionality is not the only problem, but the extension must provide usability, so it can be easily implemented.

2.2. Extension Details

2.2.1 Goals

I've set a number of goals for myself. The idea basically evolved from implementing bullet drop and bullet piercing. I've extended that idea later on, to provide an interface for the use of several different types of projectiles. Including shells, bullets and grenades.

Over time, I've evolved my goals, to not only cover the functionality, but also make the interface simple to merge and use, this would become helpful when presenting the tech demo. As my knowledge of OGRE changed, so did my goals and what I hoped to accomplish with my extension.

2.2.2 Goals achieved

I have achieved quite a few of the goals that I've set for myself, while there is definitely room for improvements, overall I am satisfied with the result.

Usability

I am very satisfied with how my extension works, the architecture, in my opinion, makes it very clear on how to be used. There are quite a few things that however would need to be implemented. With the help of void pointers, I would also like to achieve a way to have a custom callback for certain actors. This would help when developing games and would give the user of the extension more functionality, besides just one that only affects the physics.

Shells:

I've managed to implement shells, without too much effort. However, explosions were a bit tricky due to the lack of nVidia documentation, while some tutorials and examples are provided, they're not something you could use at any given time, so adjustments had to be made.

Due to the nature of how projectiles work, and the fact that implementing CCD would have been quite slow, and is generally not preferred when seeking performance, I've decided to remove the CCD part and instead focus on implementing two type of shells. One is focused on really fast shells, in this case, the speed would be so great, that external physical factors wouldn't matter all that much, especially in close quarter games. This is the 'fast shell' type, the other slow shell, works by generating an object, and firing towards the target.

The first type would be appropriate for tanks, or maybe flak cannons, that have a high velocity of projectiles when they exit.
The slow shell, however does have a bit more variation, it can however mostly be used for projectiles that explode on contact. The force and the mass of the parameters can be set, which gives us extra functionality when we want to achieve different behavior on our projectiles.

Grenades:

Grenades work very well. They explode after a certain amount of time, they react to the environment and after a certain amount of time, create an explosion.

2.2.3. Goals not achieved

Bullets:

Yes, bullets are there, but they're in their most simplistic form. Just a straight line, while bullet drop can be implemented by doing a raycast, and depending on the distance, calculate a new raycast, which would have it's origin changed by the drop calculated (distance * speed = time; we use time in the drop equation, or alternatively use ballistic tables which would also help with drag) . If the new distance and the old distance are above a certain threshold (e.g. we shoot a floating ball, but the bullet drops, the next object could be far away), we calculate the drop again.

The reason however this wasn't implemented is, because it adds a lot of computation, for results, that in most video games are not even important. There are of course some niche video games that will take into account (e.g. Sniper Elite), but these are the exception more than the rule. For realistic physical simulations, I think I would need to rewrite the whole extension to suit the functionality, that would be required in a realistic physics simulation.

Piercing:

Piercing was one of my main goals, the main reason, I decided not to implement it, was not because it would be hard, but because it would make the model quite complex. Since the programmer receives the raycast hit from firing bullets. The reason behind this is quite simple, for piercing to work, we'd need to keep a structure of all objects that can be penetrated and check it against every raycast hit. It's easier for this to be done in the main code, the other problem would be, that we would have to return these hits as a data structure containing several hits, this might make the returning results quite annoying to use, since you would have to check the amount of elements in the structure, and then loop through them all for each time you fire a bullet. I decided, that it would be easier for someone to implement it, in a way that would be more familiar and specific to his own uses.

Structure:

While I am satisfied with most of the things that are concerned with the overall structure of the extension, there are quite a few things that I couldn't solve in a way that I would feel comfortable with. The reason for this is that I am still fairly unfamiliar with the C++ language, and learning it over this course has caused some issues now and then. The main thing I disliked was that there was a need for a separate class that took care of the collisions. However that was a PhysX limitation. This however did cause a few other problems. Accessing external data structures was quite hard. Especially those that were related to the ProjectileCannon class. This meant that I had to declare some structures that were placed in the ProjectileCannon.cpp file, but were not part of the class, the same was true for some other data structures. While I am sure there is a way around it, I haven't found it. The need to access the pList in the collision code led to most of these problems.

Overall:

2.2.4. Bugs:

ForceField despawning problem:

This bug occurs, when you shoot several projectiles that cause explosions, the problem shows, when a ForceField does not get removed properly. While the purge function, should remove any force fields, or more specifically change their size, after a certain amount of time this however does not seem to happen.

I have spent quite some time trying to remove this bug, even going so far, to destroy the force field completely. I assume, that this is because of how force fields are implemented in nVidias PhysX.

Objects can't be affected with a new forcefield:

I think this bug is closely related to the one before. However, the way this one works is that once an object is affected by a forcefield a newly created forcefield might not affect the same object again.

As an interesting fact, in the 'Obliterate' function, which is more of a joke than anything else. The objects get affected again, however the huge forcefield created does not seem to remove itself properly.

Fast Shell demo bug:

This bug occurs, when using the fast shell projectile, at certain angles. While the program does exactly what it's supposed to do, the raycast returns an odd location and distance. I am assuming this is not a bug with the extension itself, but rather a bug in the demo application, with the raycast colliding with the static actor, that we used to represent the player.

This would explain why it comes only from certain angles. This however wouldn't be a problem in a proper implementation. In our case, the object is mostly static, so the cube representing the player moves back and forth, in a realistic scenario, the hitbox would change it's orientation. Which means that the raycast would not collide with the hitbox of the player character.

2.3. Assignment - Group contributions

Besides doing my work on my personal extension, I have helped with the implementation of my extension, as well as helping our software engineer on integrating PhysX with the engine itself.

Towards the completion of the tech demo, I also helped in the graphics side. Trying to figure out how OGRE handles textures and making sure our objects had them. As my extension required some input, I helped with the coding of the part of the program that handles the input from the keyboard. This was needed so we would properly switch different types of projectiles.

During the early development phase of the demo, I also handled the shadows so that they appeared properly on the scene. This included working with shadows, making sure objects have the right properties. For this I used the shadow volume algorithm, that is already a part of OGRE.

3. Evaluation

3.1. Personal work

I am fairly satisfied with my personal work, however, I do feel there are quite a few decisions I have made in the implementation that I am not completely satisfied with. I would like to make the extension even easier to use.

One of the things, that I am not satisfied is, that I have not managed to implement a way, to simply pass functions to the code, for both using your own custom objects as projectiles, as well as using functions for custom collisions, or at least have the possibility to find a way to return the affected objects. While it was not crucial when presenting our tech demo, it is something that I think would be necessary when developing games any further.

The core of this assignment for me personally, was to familiarize myself with working on pre-made game engines. While OGRE has some very nice documentation you can work with, when working with nVidia's PhysX, it tends to get quite complicated. The wrapper, NxOgre, helps a lot. While nVidia did provide an extensive sample database, sometimes it was hard to wrap your head around the whole concept. The tutorials while useful to a degree, only show a very specific use. This is where most problems would start to manifest themselves, when you start modifying code for your own use, a lot of things can go wrong. The developer zone on nVidias home page, is not accessible to non-licensed developers, so I had to get an older version of PhysX as well. The forums are open, but I haven't found them very useful, the NxOgre forums were easier to read through, and the content felt more helpful. PhysX is quite often used, but there is no central knowledge database on which you can rely when developing your own applications. While the API is straightforward and for the most part, quite simple to use, it still gets complicated when you are fine tuning your application.

While PhysXs tutorials focused on the advanced part, NxOgre was a bit more beginner friendly, having a few tutorials that helped me get through the starting troubles, and familiarize myself with the inner workings of both, PhysX and OGRE.

3.2. Group work

While I enjoyed immensely working with my group. We've had a few glitches, most of the were organizational however. When we started on deciding our extensions, we all rushed into our own, instead of picking a common theme. A common theme would make it much easier to plan ahead, especially for the tech demo.

While early, we were mostly focused on our own extension, when we finally started putting it all together, we did work together much more. We've managed to cooperate without any problems. We were all willing to adjust any other plans we had, to meet up and work on the project together. As a group, I'd say we evolved at least from the start, and if we had another opportunity to do this project, we would have done many things differently and improved upon our past mistakes. But as a saying goes 'Anyone can be a general after the battle is done.' I could point out a lot of mistakes we've done, but from my personal perspective, they were not there because we're not capable, but mostly because of lacking experience as a part of a bigger project. However, this did change as our project evolved, and I believe most of the team members gained valuable experience which I'm sure will make a difference in our next endeavors.

As always, the final tech demo lacked some polish and I believe, that we would be able, given time, achieve even more that we did.

4. Personal reflection

I have to admit, that the project was a very positive experience for me. I've gained experience in many areas. While OGRE might be it's own engine, I believe the practices it uses, are quite common for many other engines as well, and I'm sure I would be able to apply the knowledge of OGRE to other engines as well. Towards the end, I've also used Blender to export some models into OGRE, while I am still not that familiar with it, I have basic knowledge on the subject, from where I can expand further on, if need be.

I believe PhysX provided me with the general knowledge of how physics in games work, as well as how they combine with rendering engines. While the documentation is a bit scarce, I am sure, that given time and access to a greater database of knowledge (nVidia Developer Zone), would help me expand my skills further. During my time, I've also looked at other physics engines as well. This would make it easy for me to adapt to other engines.

Working in a small tight group was definitely an interesting aspect for me. Surprisingly enough, even though I was the only non-swedish speaker in the group, there never seemed to be a language barrier between us. As an exchange student, I haven't known anyone in the group that well, but we've managed to pull it together and work really well, despite our differences. I am aware that in future projects, I won't have the luxury of picking my own team, as I did. However, the fact that we were all able to work together so well was definitely a plus.

Appendix - Development diary

28/10 - 1h - Group discussion
30/10 - 1h - Ogre3D familiarization
31/10 - 1.5h - Ogre3D familiarization, projects done in Ogre3D
2/11 - 1h - Group discussion
3/11 - 1h - Group discussion
5/11 - 4h - C++, Different physics engines
12/11 - 2h - Reading up on Bullet, ODE libraries
13/11 - 3h - Reading up on PhysX, Newton libraries
15/11 - 1h - Specified my demands for the Physics engine (callbacks, CCD etc.)
16/11 - 0.5h - Group meeting
17/11 - 2h - Getting PhysX working with Ogre3D, NxOgre???, basic example
18/11 - 2h - Switching to Bullet
20/11 - 3h - Reading up on Bullet, looking through samples
22/11 - 2h - Reading on OgreBullet
23/11 - 3h - Switching back to PhysX, lack of documentation
26/11 - 4h - Working on modifying a basic example.
28/11 - 2h - Implementing firing, dropping the idea of CCD
30/11 - 5h - Implementing core data structures, reading up on vectors.
1/12 - 3h - Implementing custom callbacks
3/12 - 6h - Implementing force fields
4/12 - 4h - Implementing force fields
5/12 - 5h - Implementing grenades
6/12 - 2.5 - Working on my presentation
7/12 - 1h - Presentation, group meeting
8/12 - 5h - Adding the concept of launchers, reworking the code to work with several launchers and one launcher controller
9/12 - 4h - Adding fastShells and bullets
10/12 - 3h - Implementing getters/setters, adding helper functions
11/12 - 5h - Commenting the code, trying to fix forcefields
13/12 - 8h - Group work, enabling extensions, working on the grenade skin, working on input
15/12 - 10h - Group work, implementing physics in demo, working on shadows and light sources in scene
16/12 - 10h - Group work, working with textures, trying to solve forcefield bug
17/12 - 8.5h - Group work, working on final adjustements, trying to solve forcefield bug and raycast bug