Proposal Draft V1 (References TBD)
Building a Physics Engine From Scratch

0.1 Introduction

Physics Engine, a term frequently used in video game industry and science research, is a tool that simulates physical phenomena using a computer. The first computer, ENIAC, at one point used a Physics Engine to help design military weapons[ref]. At the core of Physics Engine, the semi-realistic results are obtained through a combination of computation efficient numerical approximations, careful modelling of the objects, and sometimes clever hacks that enables the engine to just make the cut.

There are some common terminologies that in Physics Engines.

- *Rigid body* is an individual object which will be simulated in the engine. A rigid body is an object that does not distort or bend, as opposed to soft body and fluid, which gives rigid bodies the benefit of simplicity. A typical Physics Engine needs to keep track of its mass, position, orientation, (both linear and angular) velocity and impulse.

- *Collider* is a part of a rigid body. Complex rigid bodies tend to get separated into simple, convex colliders like spheres and boxes in the pipeline.

- *Collision Detection* and *Collision Resolution* are components of the engine which handle the interaction between colliders. Commonly Physics Engine continuously advances the time by a small fraction of a second and moving the objects according to their speed. After each position update, Collision Detection will kick in and detect if a pair of colliders would intersect. If so, Collision Resolution will decide whether and how they will get bounced and separate apart, updating their velocities accordingly.

However, the most common way Physics Engine is presented is as a big, mysterious library where all the computation is done under the hood. A wide selection of open-source physics engine are available online, including PhysX[ref], Box2D[ref] and Bullet[ref]. It is interesting for me to see how they exactly work by setting up a project of my own.

0.2 Description

My project aims to build a 3D real-time physics engine from scratch that implements basic modules, without making use of any currently available physics engine. My focus will be to realise the visual effects rather than being extermely accurate or efficient considering the limited time, my available hardware resources, and the easiness of experiments and showcase. This is also why I choose to build a real-time physics engine over a high-precision one. In addition, this project will also provide a basic framework that is easy to be extended upon in the future, if necessary. Overall, this project will be a great opportunity for myself to learn more about physics and programming.

Here is a list of the basic core modules I plan to implement:

- Object Modelling: Create data structures for rigid bodies, and design interfaces for applying forces and adding colliders.
- Collision Detection: Adopt many algorithms to decide if collisions happen.
- Bounce: Part of Collision Resolution where some maths are used to find the approximated results of a bounce.
- Friction: Part of Collision Resolution where vast assumptions are used to simulate real-life friction.
- Stability: Part of Collision Resolution where some hacky methods are used to prevent certain visual artefacts, for example if two objects are stacked together the upper one should not be bouncing up and down constantly.

The project could then be evaluated by comparisons using simple experiments[ref] against existing popular physics engines.

Of course, there are many other modules I plan to have a look at as extensions. One of them is fluid simulation. Fluid simulation involves approximations to fluid equations, and can have different levels of complexity depending on the topic. For example, simulation of buoyant force of hard objects submerged in water will be simpler than simulation of the flow of the fluid. I will try my best to cover as much in fluid simulation as I can.

There are more possible extension tasks to do if I have more time. For one, it is nice to have real time rendering to accompany my engine. For the

sake of showcase I might be using existing rendering libraries like Blender[ref], but it is certainly better to be able to not rely on them.

Another extension is to take a look at the performance of the engine. I will give different implementations for CPU and for GPU, which will then allow me to draw comparisons about their contributions in performance.

Finally, soft-body simulation is also a possible module to look into. Unlike rigid bodies, now the shape of the objects can change in a certain degree. I will need to research and choose what to implement.

0.3 Starting Point

Personally, I have some basic familarity of programming in C++, having completed a small project using it during my internship. My Graphics knowledge only comes from the two Part IA and IB graphics courses.

Additionally, I have done a bit of research online, and am therefore decently confident about the abundence of tutorial resources, despite not actually having spent time reading through them. For example, a full video series about physics engine is available on the Internet[ref]. I might also make use of 3D graphics libraries like CUDA[ref], but as of now I have no prior working experience with them.

The aforementioned open-source physics engine libraries are also available for me to look into some possible solutions or draw comparisons, but I will not be using their simulation code in my project.

0.4 Success criterion

For the core:

- Implement all basic modules: Object Modelling, Collision Detection, Bounce, Friction, Stability.
- Evaluate the engine by comparing it with popular existing engines.
- Demonstrate the engine works with screenshots of simple examples.

For extensions (ordered by priority):

- Implement fluid dynamics.

- Implement real time rendering, which should allow the project meet all four previous criteria without third-party rendering libraries.
- Provide performance comparisons for whether GPU is used, and for other interesting parameters like the number of cores used.
- Implement soft-body dynamics.

0.5 Work plan

Michaelmas term
* Now - 15 Oct
Writing project proposal
Research for libraries to use
Environment setup
Milestone: Complete full project proposal
* 16 Oct - 29 Oct
Setting up the project framework
Implement basic interfaces into rendering libraries
Milestone: Able to produce a blank video or an empty interactive demo
* 30 Oct - 19 Nov
Implement Object Modelling
* 20 Nov - 3 Dec
Implement Collision Detection
Christmas break
* 4 Dec - 17 Dec
Implement Bounce, Friction
* 18 Dec - 14 Jan
Implement Stability
Milestone: Complete the core
Lent term
* 15 Jan - 28 Jan
Buffer phase for core implementation
Core evaluation if core is completed
Write progress report
Milestone: Completed a draft of the progress report
* 29 Jan - 15 Feb
Buffer phase for core evaluation

Start extension implementations
Finish progress report
Milestone: Submit the Progress Report (Deadline 2 Feb)
* 16 Feb - 1 Mar
Implement extensions
Start dissertation write up
* 1 Mar - 15 Mar
Wrap up implementations
Continue writing the dissertation
Milestone: Implementations completed
Easter break
* 16 Mar - 1 Apr
Completed a draft of the dissertation, available for view and feedback
Milestone: First draft of dissertation completed
* 2 Apr - 22 Apr
Improve the dissertation based on the feedback
Milestone: Second draft of dissertation completed
Easter term
* 23 Apr - 1 May
Improve the dissertation based on the feedback
Milestone: Third and final draft of dissertation completed
* 2 May - 10 May
Finalise the dissertation
Milestone: Submit the final dissertation (Deadline 10 May)

0.6 Resource declaration

- I will be using my personal laptop (specs) as my main working device.
- For backup and workflow tracking, I will make use of GitHub, Google Drive, and Overleaf
- For development I will be using rendering libraries like Blender, as well as GPU interfaces such as CUDA.
- As a backup plan I have another laptop for working.