Project Proposal - Building a Physics Engine from Scratch

Oct 2023

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[1]. At the core of physics engine, semi-realistic results are obtained through a combination of computation-efficient numerical approximations, careful modelling of the objects, and sometimes clever hacks that enable the engine to just make the cut.

There are some common terminologies in physics engines.

- **Rigid body** is an individual object that will be simulated in the engine. A rigid body is an object that does not distort or bend, as opposed to soft body or fluid, which gives rigid bodies the benefit of simplicity. A typical physics engine needs to keep track of its mass, position, orientation, linear velocity, 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, moving the objects according to their speed. After each position update, Collision detection will kick in and detect if a pair of colliders will intersect. If so, Collision resolution will decide whether and how they will get bounced and separated apart, updating their velocities accordingly.

Many open-source physics engines are available online, including PhysX[2], Box2D[3] and Bullet[4]. However, the most common way a physics engine is presented is as a big, mysterious library where all the computation is done under dozens of dependencies and documentation. As a result, adding a simple feature could take a lot of effort of plowing through documentation and files. I want to build my own physics engine, which gives me the flexibility to add whatever I want because I would know exactly how it works.

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 extremely accurate or efficient, considering the limited time, my available hardware resources, and the ease of experiments and showcases. This is also why I chose 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 extend upon in the future, if necessary. Overall, this project will be a great opportunity for me 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 recording attributes of rigid bodies and design interfaces for applying forces and adding colliders. Typical physical attributes of rigid bodies include mass, position, orientation, linear velocity, angular velocity, and impulse.

- 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 heuristics 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 against existing popular physics engines. More on this in the Evaluation section.

Extension modules I plan to look at include:

- Fluid simulation:

  Fluid simulation involves approximations to fluid equations, and can have different levels of complexity depending on the topic. For example, the simulation of buoyant force of hard objects submerged in water will be simpler than the simulation of the flow of the fluid. I will try my best to cover as much in fluid simulation as I can.

- Real-time rendering:

  For the sake of showcasing, I might be using existing rendering libraries like Blender[5], but it is certainly better to not rely on them.

- Performance evaluation

  I will give different implementations for CPU and GPU, which will then allow me to draw comparisons about their contributions to performance.

- Soft-body simulation

  Unlike rigid bodies, the shape of objects can change to a certain degree. Therefore, it is much harder to model them and deal with collisions. I will need to do research and choose what to implement.

3 Evaluation

The evaluation of this project is split into three parts: Benchmark selection, Core functionality, and Performance evaluation.

3.1 Benchmark selection

Since the functionality evaluations will be largely comparison-based, I will be choosing at least three open-source physics engines that support similar features to compare against. The following engines have been found as possible candidates:

Physics engine	Website
Advanced Simulation Library	asl.org.il
Bullet	pybullet.org
Newton Game Dynamics	newtondynamics.com/forum/newton.php
Open Dynamics Engine	www.ode.org
PAL	www.adrianboeing.com/pal
PhysX	www.nvidia.com/en-gb/geforce/technologies/physx
Project Chrono	projectchrono.org
Siconos	nonsmooth.gricad-pages.univ-grenoble-alpes.fr/siconos
SOFA	www.sofa-framework.org
Tokamak physics engine	github.com/isegal/TokamakPhysics

I plan to further narrow them down by testing if they work in my environment, if they have sufficient documentation, and if they support the features I am going to compare on.

3.2 Core functionality

I will evaluate the accuracy of the physics engine by setting up experiments like pendulums and boxes sliding down inclined planes in my physics engine. I will then compare how the rigid bodies move in the engine with the well known physical results of how these systems behave in real life.

3.3 Performance evaluation

This part is considered as an extension. The performance will similarly be measured with an experiment.

There will be comparisons between my engine and other benchmark engines, as well as between different implementations of my engine.

I will define, measure and then plot the average computational time of a time step, while I gradually make the scene more complicated by adding in more objects.

4 Starting Point

Personally, I have some basic knowledge 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 abundance 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[6]. I might also make use of 3D graphics libraries like CUDA[7], 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.

5 Success criteria

For the core:

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

For extensions (ordered by priority):

- Implement fluid dynamics.
- Implement different versions of the engine for whether GPU is used, and for other interesting parameters like the number of cores used. Then evaluate the performance.
- Implement real-time rendering, which should allow the project to meet all previous criteria without third-party rendering libraries.
- Implement soft-body dynamics.

6 Work plan

Michaelmas term

Now - 15 Oct

Write project proposal
Research for libraries to use
Environment setup
Milestone: Complete full project proposal

16 Oct - 29 Oct

Set 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

Complete 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)

7 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.

References

[1] Raúl Rojas and Ulf Hashagen. *The first computers: History and architectures*. MIT press, 2002.

[2] Physx. `https://www.nvidia.com/en-gb/drivers/physx/physx-9-19-0218-driver/`. Accessed: 2023-10-12.

[3] Box2d. `https://box2d.org/`. Accessed: 2023-10-12.

[4] Bullet. `https://github.com/bulletphysics/bullet3`. Accessed: 2023-10-12.

[5] Blender. `https://www.blender.org/`. Accessed: 2023-10-12.

[6] Physics engine tutorial. `https://www.youtube.com/playlist?list=PLEETnX-uPtBXm1KEr_2zQ6K_0hoGH6JJ0`. Accessed: 2023-10-12.

[7] Cuda. `https://developer.nvidia.com/cuda-toolkit`. Accessed: 2023-10-12.