Custom Physics Documentation

PHYSIC SIM NAME

Benjamin\_Luke\_Scott

2022

Contents

[Visualised Game Using Your Custom Physics Simulation 2](#_Toc95731610)

[Custom Physics Simulation Interactions 3](#_Toc95731611)

[Custom Physics Simulation Class Diagram 4](#_Toc95731612)

[Custom Physics Simulation Potential Improvements 5](#_Toc95731613)

[Improvement #1 5](#_Toc95731614)

[Improvement #2 5](#_Toc95731615)

[Improvement #3 5](#_Toc95731616)

[Third Party Libraries 6](#_Toc95731617)

[References 6](#_Toc95731618)

# Visualised Game Using Your Custom Physics Simulation

The game I chose to visualise my physics simulation is Pachinko. The game works by having kinematic circles fall because gravitational force is being added each frame as well as a collision check. When the circle touches any Physic Object the collision check will apply contact forces causing the circle to bounce. You gain point depending on what coloured box trigger the circles enter at the bottom (see game imagine).

I Created the Custom physics engine using bootstrap and C++. It works by updating the simulation each fixed time step set by the player. (The higher the timestep the less accurate/slower the simulation). Meaning each frame, it will apply a gravitational force and do a collision check on every Physic Object inside the simulation.

**Menu Screen**

Graphical user interface

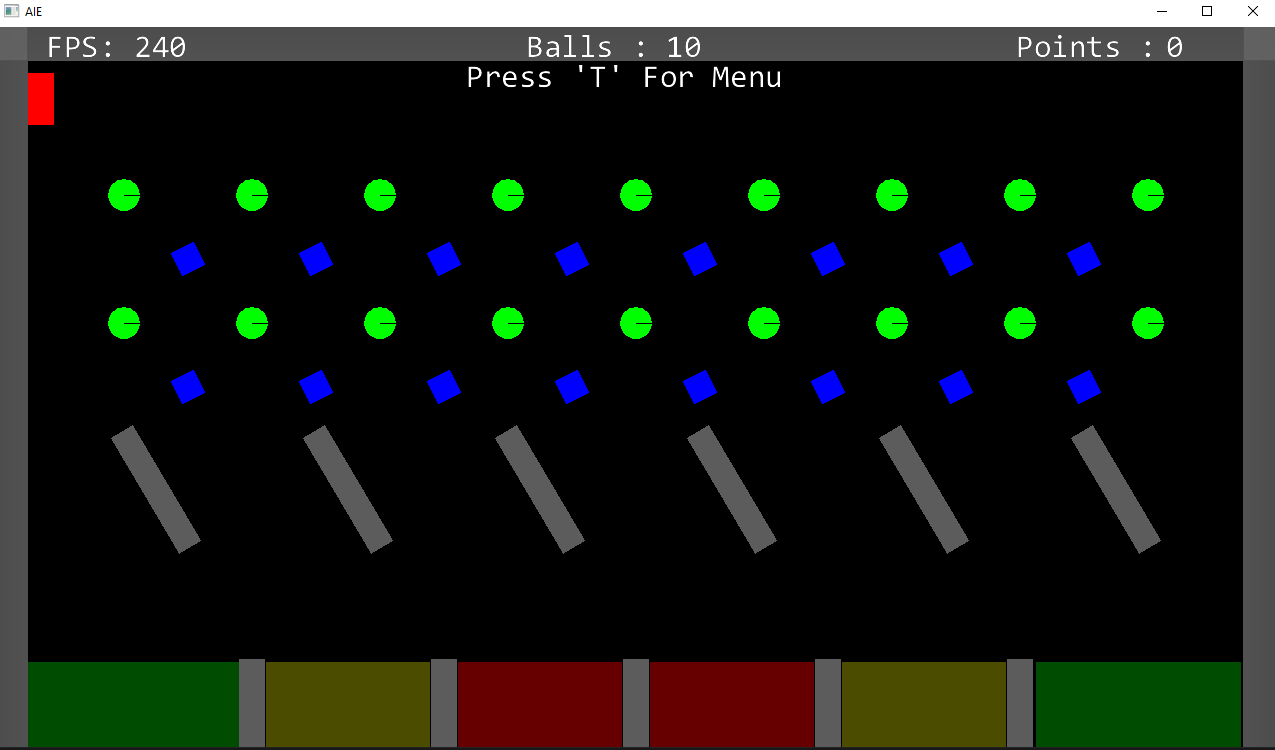
Description automatically generated

**Help Screen**

Text

Description automatically generated

**Game Screen**



**Game Screen being played.** Graphical user interface

Description automatically generated

# Custom Physics Simulation Interactions

The custom physics simulation is demonstrating how objects interact when they collide with each other like in the real world for example, if a big ball with lots of mass collides with a small ball with little mass, you would expect the small ball to be launched away.

Each Physic Objects have a Rigid Body attached to allow the physics objects to react to real time physics. This includes reactions to forces, mass, gravity, elasticity, drag and momentum.

If a physic object is a Trigger, it will be none kinematic meaning it doesn’t react to physics instead it will have a bool value toggled/function called on when a physics object collides with it and when it exits the collision.

There are 9 collision checks in this simulation each one taking in 2 Physic Objects and resolving the collision by applying contact forces. The simulation gets both objects mass, collision norm and penetration values to works out where to set its new position each frame.

Plane2plane, Plane2circle, Plane2box, Circle2plane, circle2circle, circle2box, Box2plane, box2circle, box2box.

# Custom Physics Simulation Class Diagram

See attached Html or draw.io files to view UML class diagram.

# Custom Physics Simulation Potential Improvements

## Improvement #1

**Octree Detection** –Currently the simulation is checking every frame if any object has collided based on the position of each object in the world space. If you keep adding objects to the simulation it will drop its framerate to keep up with the demand.

Octrees would be a potential improvement because it improves the way you check if an object has collided and reports the geometric details of all the points of contact. “It builds an axis-aligned hierarchical data structure that is generated by recursively subsiding the axis-aligned bouncing box (AABB) into eight equally sized cells”.

Using the octree method means not only can we detect more then just basic convex shapes, but the detection would be more accurate and the memory that is allocated is a lot less each frame.

The Octree uses memory pool, meaning there is a list of connected memory blocks in which each block stores a pointer to the next block. If the memory pool is exhausted more memory is allocated. Discarded nodes will release their memory block back into the memory pool for reuse making octrees more efficient.

## Improvement #2

**Collision Layers** – An improvement would be to add collision layers into my simulation like how unity does collision layers. A Layer collision matrix allows you to choose what objects can collide with any other object assigned on the same layer. All objects by default would be on the first layer unless assigned to another. This creates the ability to have collide like normal unless you assign it to another layer. My current simulation has objects colliding with all objects.

This would be a potential improvement if I desired the player or any object to not collide without making the object a trigger and remaining the same after having the player pass through the object.

## Improvement #3

**Concave Shapes** – Current my simulation only draws and detects convex shapes. These are shapes that have curves going outward and the middle is usually thicker than its edges and all parts point outward. For example, squares, circles, pentagons and triangles etc.

Adding concave shapes means my simulation can have shapes that curve inward like the inside part of a bowl or if you take a bite out of a cookie.

A Potential improvement would add the ability to use and detect collision with concave shapes. Using SAT or GJK detection.

Separating axis theorem (SAT) works with all shapes not just convex. “Sat essentially states if you are able to draw a line to separate two shapes, then they do not collide”

GJK is a popular algorithm that can determine a collision among complicated convex shapes. It finds the direction of the shape and origin, when shape a and b are added together, and they share the same origin (Minkowski difference) then they have collided.

Having GJK detection would be a potential improvement because I would mean I can have more convex shapes added in the simulation.

# Third Party Libraries

Bootstrap – Open-source toolkit / framework.

GLM – OpenGL mathematics library

# References

AIE Online Course. n.d. ***AIE’s Online Course*** *-* ***Creating a Physics Engine***. [online] Available at: <https://aie.instructure.com/courses/813/pages/physics-for-games-creating-a-physics-engine> [Accessed 18 April 2022].

Chong, K., 2022. ***Collision Detection Using the Separating Axis Theorem***. [online] Game Development EnvatoTuts+. Available at: <https://gamedevelopment.tutsplus.com/tutorials/collision-detection-using-the-separating-axis-theorem--gamedev-169> [Accessed 18 April 2022].

Pierce, R., 2022. ***Concave Definition*** *(Illustrated Mathematics Dictionary)*. [online] Mathsisfun.com. Available at: <https://www.mathsisfun.com/definitions/concave.html> [Accessed 18 April 2022].

Ridner, A., 2022. ***Concave: Definition, Shape & Function***. [online] Study.com - Concave. Available at: <https://study.com/academy/lesson/concave-definition-shape-function.html> [Accessed 18 April 2022].

Serrano, H., 2013. ***Visualizing the GJK Collision detection algorithm*** *— Harold Serrano - Game Engine Developer*. [online] Harold Serrano. Available at: <https://www.haroldserrano.com/blog/visualizing-the-gjk-collision-algorithm> [Accessed 18 April 2022].

Technologies, U., 2021. *Unity -* ***Manual: Layer-based collision detection***. [online] Docs.unity3d.com. Available at: <https://docs.unity3d.com/Manual/LayerBasedCollision.html> [Accessed 18 April 2022].

Truong, N., Arikatla, S. and Enquobahrie, A., 2022. ***Octree-based Collision Detection*** *in iMSTK*. [online] Kitware Inc. Available at: <https://www.kitware.com/octree-collision-imstk/> [Accessed 18 April 2022].

Kamburelis, M., 2016. ***Octrees***. [online] Castle-engine.io. Available at: < https://castle-engine.io/vrml\_engine\_doc/output/xsl/html/section.how\_octree\_works.html#section.octree\_checking\_collisions > [Accessed 18 April 2022].