Physics application report: Duncan Sykes

What is it Simulating?

- The physics simulation is demonstrating a simplified version of the game ‘English Billiards’. Where two players compete to “sink” their chosen set of balls (either striped or solid) with a ‘cue ball’ (white or red ball) using Cues (a stick to hit the cue ball with).
- The balls are displayed as 2D spheres on screen, the cue is controlled by clicking and dragging near the cue ball within a set radius of it. Clicking, holding and then releasing near the cue ball moves it away from the cue based on a distance calculation. When the cue ball strikes a billiard ball, the force of the collision transfers the velocity of the cue ball into the billiard ball and sets it In motion.
- If a billiard ball collides with a hole, it is safely deleted from the simulation and adds + 1 to the score.

Collision Resolution and Detection:

Collision Detection works by using the distance between two or more objects. If this distance becomes close to zero, we determine that the objects are getting closer together. Once ‘zero distance’ is reached, it can be said that two objects have collided.

Collison Resolution is the process of determining of what happens after a collision has been detected. In this simulation, we use the physical model of “Newtown’s Law of Restitution for instantaneous Collisions with No Friction”.

Calculating impulse to update the object velocities, all other non-collision forces (gravity etc) are ignored.

To resolve a collision between two objects we need a few things:

- The relative velocities of the two objects
- The Collision Normal
- The coefficient of velocity: (assuming perfect elasticity)

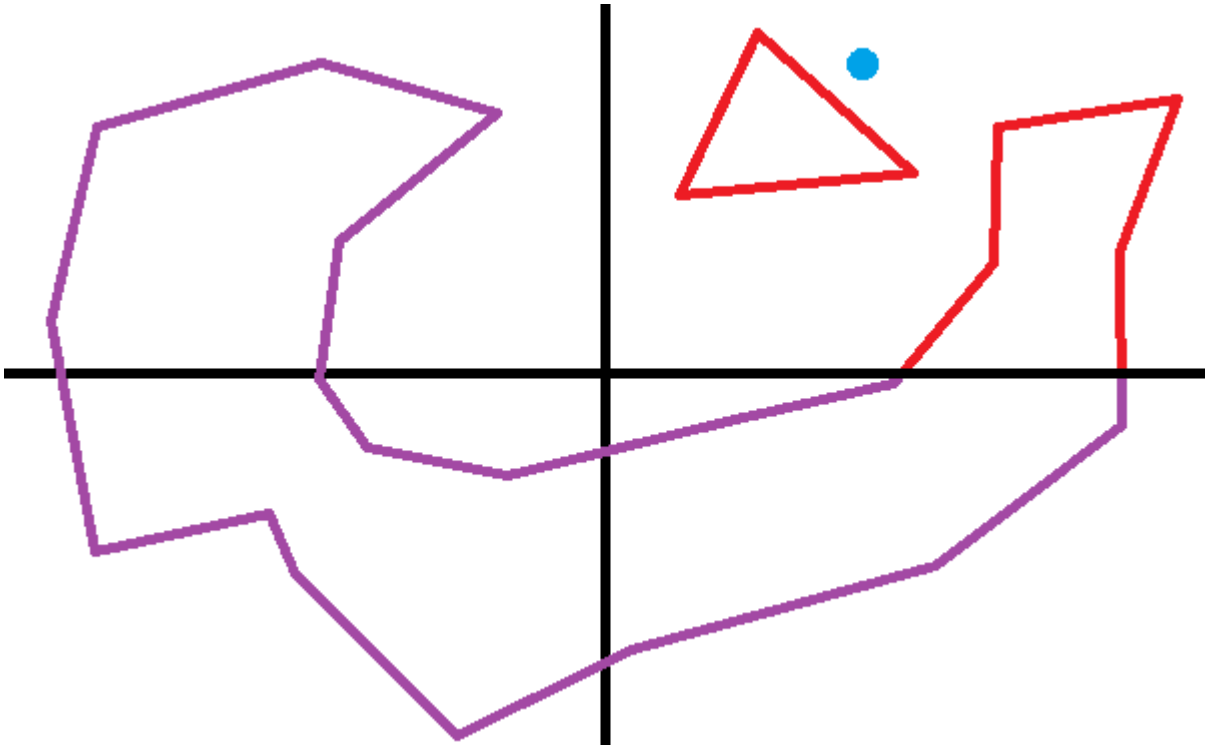
We then compute an impulse with the proper magnitude and apply it to both objects, but in opposite directions.

Possible Improvements

Quadtrees:

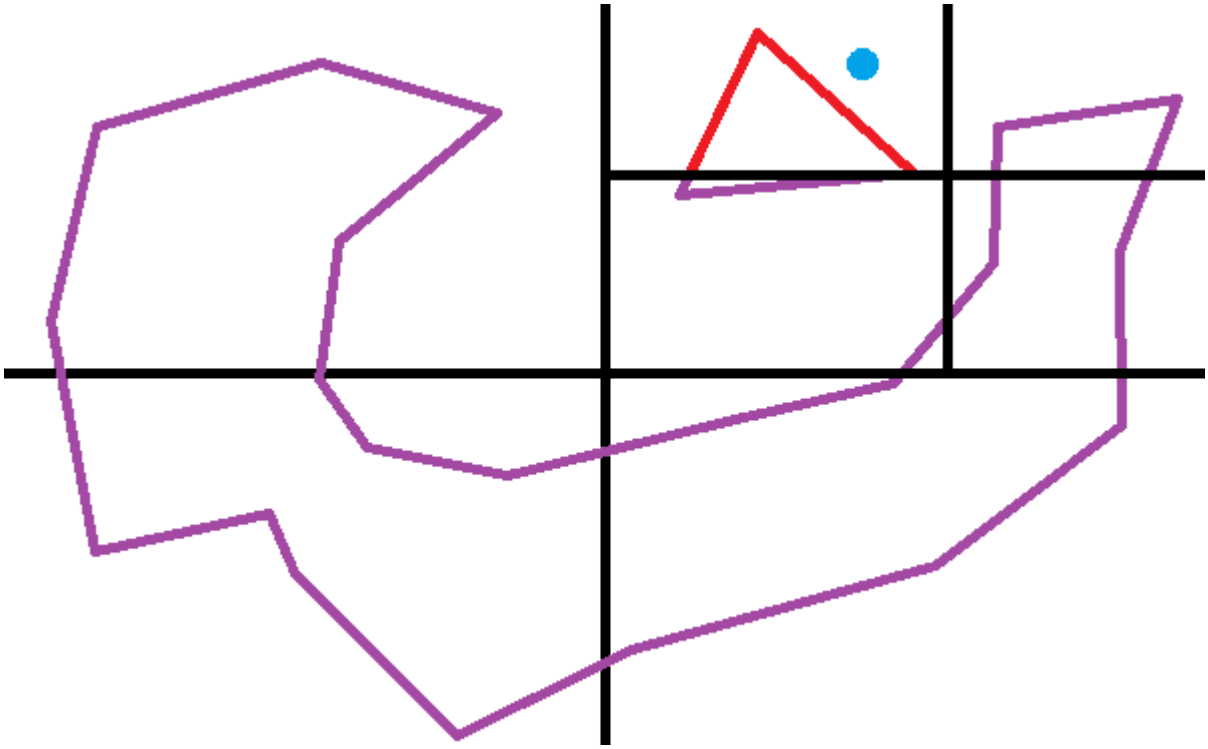
Quadtrees are a data structure that is used to divide a 2D region into more manageable space. It is an extension of a binary tree, but with four child nodes instead of two.

The problem with the current system of detection, is that it checks for every object for every other object. (Currently in this simple demo its not much of an issue, but if this system were scaled up to detect the collisions between 200 concurrent objects, performance would suffer). If a collection of objects were only in the upper part of the screen - we should not have to check every object across different parts of the simulation for collision.



Say the red lines are a wall, and the blue dot is a dynamically moving sphere, we should only really be checking for a collision in that top part of the screen.

We can go even further still though, if the particle is in the upper left of the top quadrant of the screen, we should only need to check that left part of the upper quadrant.



This continuing pattern can be continued even further.

Rather than testing all walls and objects, you can split areas into quadrants, and then those into sub quadrants and so forth. You can easily go from 500 checks for a particle to maybe 5 or 6.

Object Pool Pattern

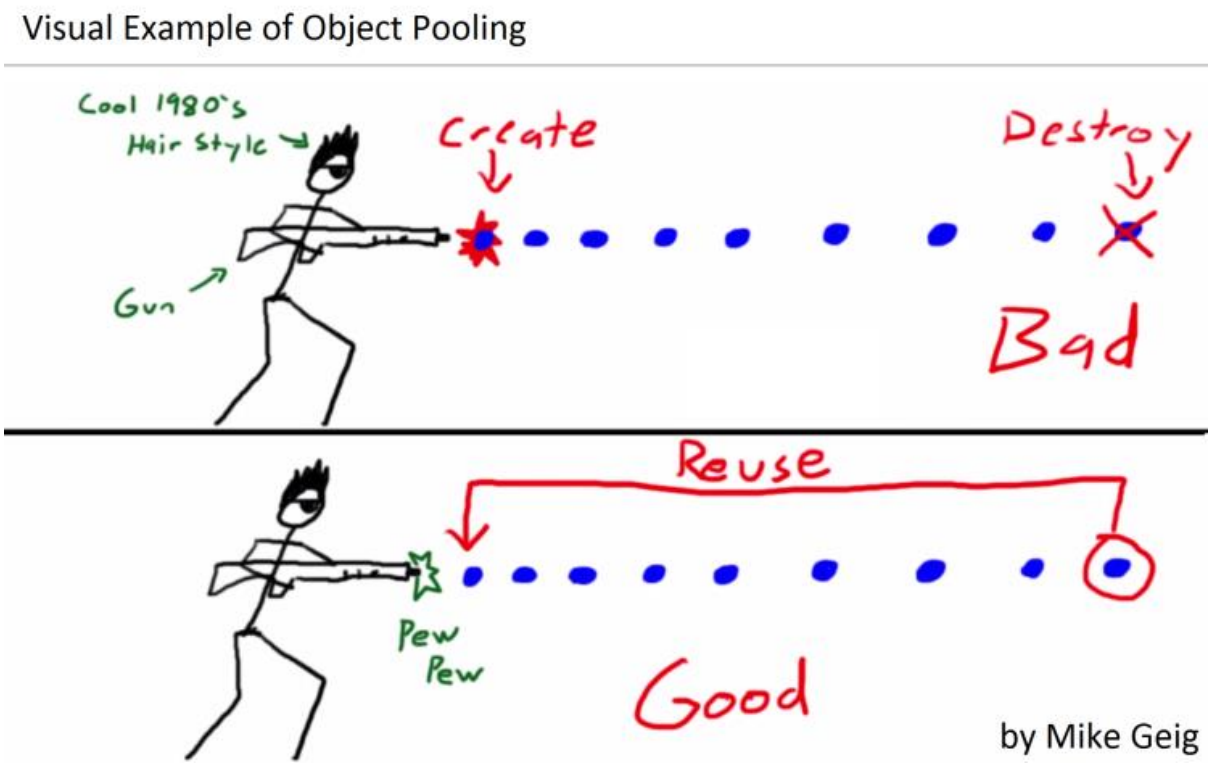
The object pool pattern is a set of initialised objects that are kept “ready to use” – a pool – rather than allocating and destroying on demand. The pool design creates a set of objects that can be reused. When a new object is needed, it is requested from the pool. If no objects are present in the pool, a new item is created and returned. When the object has been used and is no longer needed it is returned to the pool.

Object pooling could be implemented via smart pointers in C++. The constructor of a smart pointer can request an object from the pool, and the destructor can return it.

An example found online showed that object pooling is useful when processing facial recognition data. The developer created a thread safe pool to share facial detection objects between threads, only loading as many as need. The main thread owns the detection pool – which owned the detection objects.

Object pooling can also help prevent fragmentation. Fragmentation means the free space in our memory heap is broken into smaller pieces of memory rather than one large block. The total free memory available may be large, but the largest region of continuous free space may be very small. Even if fragmentation is infrequent, it will still gradually reduce the heap to an unusable state.

Due to fragmentation and memory allocation, it is important that physics engines and games are careful about memory management. The object pool allows us to allocate one big chunk of memory up front and not freeing it while the program is running. This allows us to freely allocate and deallocate objects without worrying about fragmenting memory.



Sleep / Awake Cycling

Most popular physics engines divide dynamic objects into two groups, “awake” and sleeping”. Objects sleep when they sit at rest, and wake when moved or accelerated by some outside influence. A sleeping object behaves like a static object – its movement is not integrated over time (due to being at rest). Engines typically ignore collisions between objects that are sleeping and/or static.

A sleeping object sitting on a static floor doesn’t clip or fall through it. Even though the object has a lack of collision detection, gravity is ignored since all movement integration is ignored.

A common approach is to monitor the objects velocity, and when the velocity is below a defined threshold for several frames, the object is put to sleep. It can awaken again by colliding with an awake object, or by a series of events.

Some known issues with this method are physical interactions where an object may have near zero velocity for a few frames but has not ended its current path. A simple visualisation is a ball is rolling up a hill with via its own momentum, it's near the top where it can roll down the other side, but it slows down for just a few frames at the peak, it still has a little bit of momentum, but the engine sees that the ball has gone past the threshold for entering a sleep state, and hence the ball stops rolling.

A workaround for this could be to implement dynamic sleep states for each object.

References and research material

References

deletion, C. and Wakely, J., 2021. *C++ object-pool that provides items as smart-pointers that are returned to pool upon deletion*. [online] Stack Overflow. Available at: <https://stackoverflow.com/questions/27827923/c-object-pool-that-provides-items-as-smart-pointers-that-are-returned-to-pool> [Accessed 26 February 2021].

En.wikipedia.org. 2021. *General-purpose computing on graphics processing units*. [online] Available at: <https://en.wikipedia.org/wiki/General-purpose_computing_on_graphics_processing_units> [Accessed 26 February 2021].

En.wikipedia.org. 2021. *Object pool pattern*. [online] Available at: <https://en.wikipedia.org/wiki/Object_pool_pattern> [Accessed 26 February 2021].

En.wikipedia.org. 2021. *Physics engine*. [online] Available at: <https://en.wikipedia.org/wiki/Physics_engine> [Accessed 26 February 2021].

Gamasutra.com. 2021. *Scaling a Physics Engine to Millions of Players*. [online] Available at: <https://www.gamasutra.com/view/feature/173977/scaling_a_physics_engine_to_.php?page=2> [Accessed 26 February 2021].

In a 2D physics engine, h., drone.vs.drones, d. and drone.vs.drones, d., 2021. *In a 2D physics engine, how do I avoid useless collision resolutions when objects come to a rest?*. [online] Game Development Stack Exchange. Available at: <https://gamedev.stackexchange.com/questions/114925/in-a-2d-physics-engine-how-do-i-avoid-useless-collision-resolutions-when-object> [Accessed 26 February 2021].

Roblox Blog. 2021. *Roblox Physics: Building a Better Sleep System - Roblox Blog*. [online] Available at: <https://blog.roblox.com/2020/07/roblox-physics-building-better-sleep-system/> [Accessed 26 February 2021].