Simple Collision Engine

Michael Gorbunov

A still image attempting to show collision response (motion).
View the demo to actually try out the result.

Demo

View a live demo of the result here: https://mrgorbunov.github.io/get-get-golf/

Overview

Winter break during 12th grade (Dec 2020) I thought it would be cool to make a golf game. However, it would've been even cooler if I also wrote the physics engine. So, I wrote the physics engine and then never finished the game. Nonetheless creating the engine was a very exciting week scribbling geometry problems in my notebook.

The big thing this taught me was the power of dot products, which as you will see are everywhere. Also, to keep things simple I stuck to 2D, and only considered collisions between circles and line segments. Compared to GJK or really any 3D geometry problem, everything here is trivial. However, I came into this project really only understanding that a dot product is |a||b|cos(theta), and its relevance with projections. This should be evidence of my ability to work through problems.

The following also may not seem like a lot, but it's all that was necessary for the engine.

Collision Detection

Every physics tick, the engine checks for collisions between the circle and all segments. This means it is important to be able to quickly say no, because most checks will not be collisions.

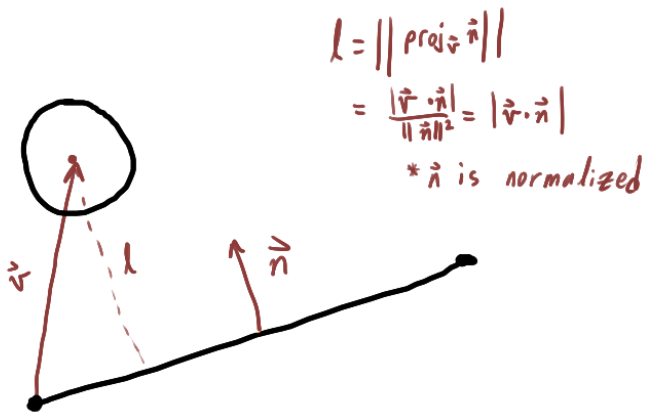
Collisions are thus checked in the following order, where each step assumes the previous one was inconclusive, but still narrows down possibilities.

1. Distance

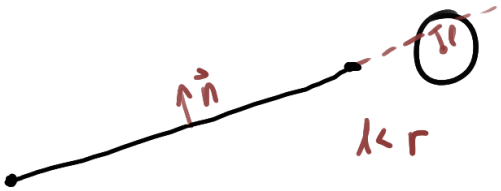
This can definitively say no.

If the circle is further from the line than its radius, then it is not touching the segment. However, this check is done against the line, not the line segment, so it cannot definitively say yes.

Calculating this is done by projecting the vector from an endpoint onto the normal vector, which in this case simplifies to a dot product.



This cannot definitively say a collision happens, see below:

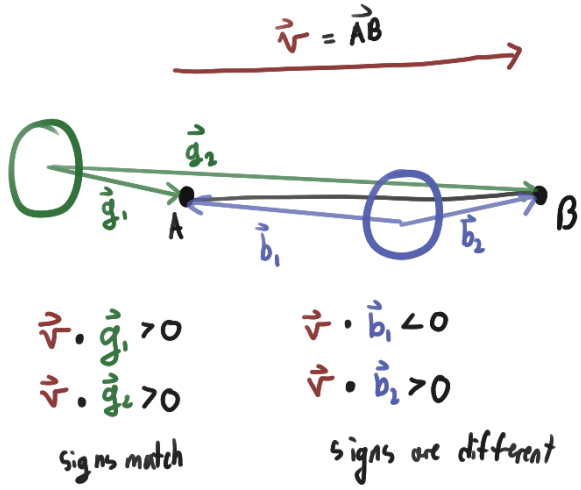


Note: Distance between a line and a point was covered in my Calc III class so this wasn't new.

2. Sign of Dot Products

At this point, we know the circle is insertecting the line, but it might not be intersecting the line segment.

An important property of dot products is their sign, which tells whether two vectors point in the same direction. For a circle, we can generate a vector pointing to each end point on a segment. Then take the dot product of the vector and a vector spanning the segment, and the signs will tell us whether the circle is within the segment.



This check is not exhaustive. It tells whether the circle center is along the line segment, but if the circle can still intersect the line even if the center is just off to the side.

Note: In writing this report, I realised I could simply have taken the dot product b_1 with b_2 and used the sign of that. This would've been simpler and avoided the need to calculate vector v.

3. Collision with Segment Endpoints

The final check necessary is to see if the circle encloses either endpoint of the line segment. This is done with a simple distance calculation, wherein if the either endpoint is less than the radius away from the center, the circle is colliding.

In implementation, square distance and square radius are used to avoid an expensive square root calculation.

Collision Response

Collision Location

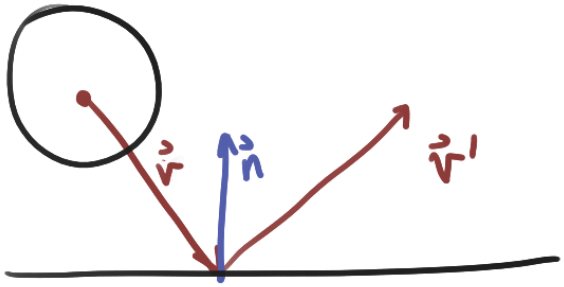
The simulation is done via Euler integration, where each physics tick the ball moves forward by some small vector along its velocity. When a collision is detected, the ball is already somewhat inside the line segment, however for a clean response it's important to push the ball back out of the segment.

While this could've been mathematically determined, I saw an easier solution. I just binary searched along the movement vector to determine at what point the collision first happened. Since this is a computer science item, I will not discuss it further.

Changing Velocity Vector

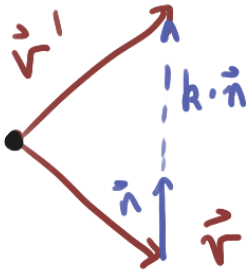
For these collisions, I assume elasticity. Because the walls do not move, there is 0 change to speed, and the ball's velocity simply reflects along the segment.

The exciting observation is that this reflected vector is also equal to the original vector plus some scalar quantity of the normal vector (shown left).



This gives 2 restrictions for the reflected vector:

1. $\|\vec{\mathbf{v}'}\| = \|\vec{\mathbf{v}}\|$
2. $\vec{\mathbf{v}'} = \vec{\mathbf{v}} + k \cdot \vec{\mathbf{n}}$



Because **n** is normalized, k in the second equation is equal to || **v'** - **v** ||. The picture shows, this quantity equals twice the height of **v** projected onto **n**. So, k is equals 2 · || $\text{proj}_{\mathbf{n}}(\mathbf{v})$ ||, which simplifies to 2 · (**n** * **v**).

Breaking this down component by component, the following very simple equations come out:

$$v'_x = v_x + (\vec{\mathbf{v}} \cdot \vec{\mathbf{n}}) \cdot n_x$$
$$v'_y = v_y + (\vec{\mathbf{v}} \cdot \vec{\mathbf{n}}) \cdot n_y$$