Final Project: Rigid Body Simulation

STUDENT NAME: YUHAO ZHANG & ZHELIN ZHU
STUDENT NO.: 2021533141 & 2021533077
EMAIL: ZHANGYH7@ & ZHUZHL@

1 INTRODUCTION

In the field of computational physics and computer graphics, the simulation of rigid body dynamics stands as a cornerstone, particularly in the context of gaming, animation, and virtual reality. This report introduces a Rigid Body Simulator, which can simulate collisions and friction between two of the most fundamental shapes in geometry - balls and boxes.

2 IMPLEMENTATION METHOD

In this part, we will focus on the following concept, which are the key points of this simulator.

- sphere sphere collision detection
- sphere sphere collision processing
- friction processing
- continuous collision detection
- collision detection acceleration
- GJK (Gilbert-Johnson-Keerthi)
- EPA (Expanding Polytope Algorithm)
- Interesting function as change the size and position of model,add the wind.

2.1 Model in opengl

To start our rigid body simulation ,we should model different shapes,we implement sphere and box as convex shape models. The sphere mesh is sampled according to the uv sampling.

$$x = r\cos(u)\cos(v), y = r\cos(u)\sin(v), z = r\sin(u)$$

the box is divided into six rectangle,each rectangle is divided into two triangles. Besides the shape ,we store the position,velocity,rotation ,mass information in body class,we can easily get model matrix according to them.Each frame,we just update the condition and apply the model matrix.

2.2 sphere sphere collision detection

In rigid body simulation,we update the object station according to the force it receives. The constant force such as gravity is easy to process. The toughest part is the collision part. Before we start to process collision we need to detect the collision, when it comes to the sphere-sphere, it is to check whether the center difference is smaller than the sum of the radius.

$$|c_1 - c_2| < r_1 + r_2$$

When collision happens,we store the information about the collision normal, and collision point in the contact class.

2.3 sphere-sphere collision processing

When collision happens,we need to process collision.

(1) To avoid interpenetration, we leverage the center of mass doesn't change and collision point 's distance equal to zero,we can update

$$x1' = x1 + \frac{\frac{d}{m_1}}{\frac{1}{m_1}+\frac{1}{m_2}}, x2' = x2 - \frac{\frac{d}{m_2}}{\frac{1}{m_1}+\frac{1}{m_2}}$$

(2) Using the conservation of momentum we can get the collision impulse as

$$J = \frac{(1+\epsilon)(v_2 - v_1)}{\frac{1}{m_1}+\frac{1}{m_2}}$$

$$J_1 = \frac{(1+\epsilon)m_1m_2(v_2 - v_1)}{m_1+m_2}, J_2 = \frac{(1+\epsilon)m_1m_2(v_1 - v_2)}{m_1+m_2}$$

(3) we also need to consider the rotation, which is related to torque τ and angular momentum L,the angular impulse is $J_{angular} = r \times J_{linear}$,and use the conservation of angular momentum, we can get the impulse as

$$J = \frac{(1+\epsilon)(v_2 - v_1)}{\frac{1}{m_1}+\frac{1}{m_2} + (I_1^{-1}(r_1 \times n) \times r_1 + I_2^{-1}(r_2 \times n) \times r_2) \cdot n}$$

where ϵ relates to the elasticity.
the angular velocity can be updated by $\omega_1^{'} = \omega_1 + I^{-1}(r_1 \times n) \cdot J$, $\omega_2^{'} = \omega_2 + I^{-1}(r_2 \times n) \cdot J$

(4) we also consider the friction impulse which can be tricky handled

$$J = \frac{\mu v_t}{\frac{1}{m_1}+\frac{1}{m_2} + (I_1^{-1}(r_1 \times v_t) \times r_1 + I_2^{-1}(r_2 \times v_t) \times r_2) \cdot v_t}$$

adding them all we can get the final impulse and the correct station.

2.4 Continuous Collision Detection

To handle the situation that fast moving will pass through the geometry in sphere-sphere collision we can calculate the accurate time collision happens by assuming the fixed linear velocity during this frame,generate ray-sphere collision function and store time of impact into contact class and update according to it.We also need to consider the order of the time of impact,we can sort them and then update sequentially. When it comes to convex collision,it is to calculate the time of impact according to the fastest linear velocity and the closest point.

2.5 Collision Detection Acceleration

There's a problem: even if two objects are far apart and there's no chance of a collision, our collision detectors will still spend a lot of time detecting if they collide. And when the number of objects

increases, this cost becomes very large, so is there a way to reduce the detection time in this circumstances?

The naive solution is not so complicate, we use bounding box. Instead of checking all objects, we first check if their bounding boxes intersect. But doing things brute force is pretty effective for getting started. But it's not very efficient when the number of objects starts increasing, we can easily get the complexity:

$$C_n^2 = \frac{n(n-1)}{2} = O(n)$$

It means that we would need to come up with a clever method to cull out potential collision pairs.

Then we will introduce the concept of a broadphase and a narrowphase. In the broad phase, the physics engine will determine which objects can possibly collide against each other, and then store the potential object collision in a list - this is often known as a collision pair. Then, in the narrow phase, the engine iterates over the list of potential collision pairs, and determines whether they really are colliding, and if so, resolves the collision.

There are many structures we can use, but we will be using the basic structure, sweep and prune, for simplicity. By sorting all the objects by the boundary projection on the axis (1, 1, 1), we can get all the possible collision pairs and this reduce the number of collision detection.

2.6 GJK algorithm

Gilbert-Johnson-Keerthi (GJK) algorithm is the method we use to determine whether two convex shape intersect. But before we introduce it, let's talk about how to transfer the intersection problem to a simpler problem.

Let's introduce Minkowski Sums. The Minkwoski sum is what happens when you have two sets of points A and B. And you add every point in A with every point in B. And in the fancy pants mathy math lingo:

$$A + B = \{a + b | a \in A, b \in B\}$$

And for two 2D convex shape, their Minkowski Sum is a bigger convex shape, which is important for this algorithm.

Next, we compute the Minkowski Sums between A and -B, if the new simplex C contains the origin, we say A and B contact or intersect. Obviously, it will cost a lot if we try to find the exact shape of C. So can we check this thing in a cheaper way? Here comes GJK algorithm.

The GJK algorithm works as follows:

- Initialization: The algorithm starts by selecting an initial point in the Minkowski difference.
- Simplex Construction and Evaluation: It then constructs the simplex around this point. Depending on the simplex's shape,we can apply simplex1D ,simplex2D,simplex3D which project origin into different simplex shape,then the algorithm determines if the origin can be enclosed within it.
- Iteration: If the simplex does not contain the origin, the algorithm moves the simplex towards the origin and adds new points to it. This process is iterated, continually adjusting the simplex and checking for the origin.
- Convergence: The algorithm concludes either when the simplex encloses the origin (indicating a collision) or when it can no longer advance towards the origin (indicating no collision).

Its iterative nature allows it to quickly converge to a solution, which will also accelerate the collision detection process.

2.7 EPA algorithm

Okay, we've got the collision information between two simplex, then we need the contact point on A and the point on B and the contact normal between them. Otherwise we won't be able to properly resolve the collision between the two bodies.

We also transfer the problem to Minkowski Sum: for the simplex C we mentioned above, we need to find the closest edge or surface from the origin.

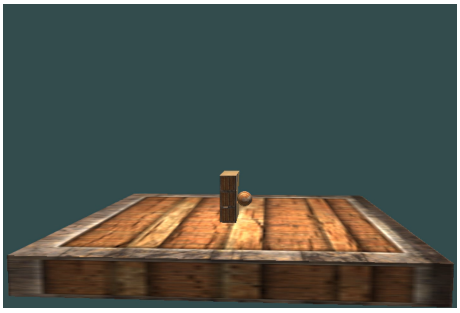
Expanding Polytope Algorithm (EPA), is an efficient way to find the closest edge, it use the simplex we get in the GJK algorithm, and expand the closest edge or surface. If it can not be expanded any more, it is the destination. And the perpendicular to the origin is the contact normal we need.

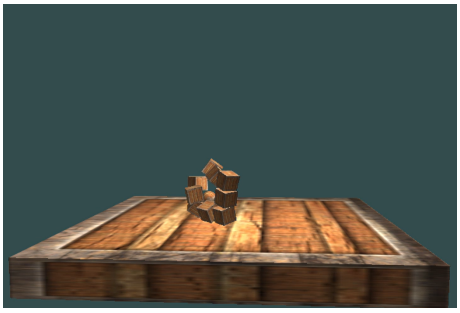
2.8 Function we add

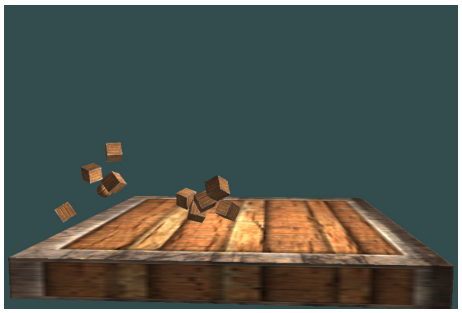
We add the interaction of mouse, from mouse vector (2D),since the mouse vector is generated by mvp(model view projection matrix),we can inverse the mvp matrix to get mouse vector in world space ,and calculate distance between the mass center.When distance <min distance ,then try to modify the size of sphere or the position.We also add the wind ,you can test it by pressing the key 1.

3 RESULT

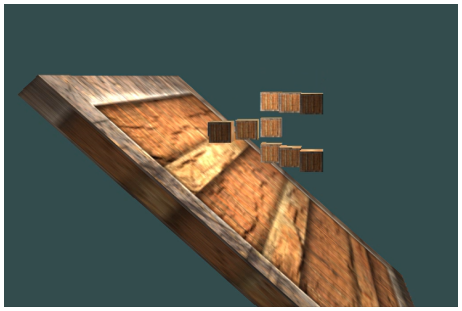
3.1 case1: a ball hitting the wall

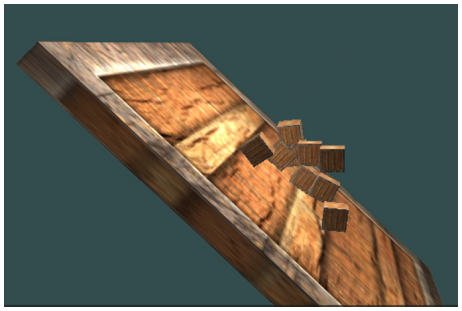


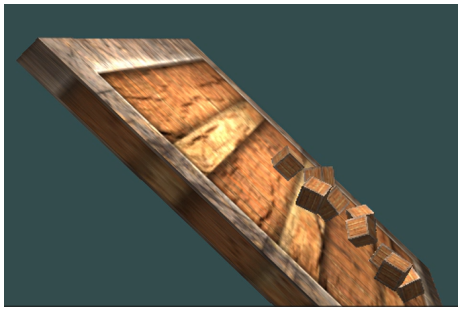




3.2 case2: boxes falling on the slope







If you want to learn more about this project, you can visit *report/result.html*.