Rigid Body Simulation

XinLong Yi, Zhuocheng Shang

1. Introduction

Rigid bodies are pervasive in daily life and simulations of their physical behaviors are useful for many fields such as industries, movie factories or game developing. In this project, we are going to develop a framework that will simulate rigid bodies. The goal of this project is aiming at implementing rigid body simulation. We intend to realize some movement of balls such as bouncing and collision. The report will include background information, goals and methods we considered and analysis of this project. Functions and algorithms are developed based on the paper *"Nonconvex Rigid Bodies with Stacking".*

2. background

2.1 Rigid body simulation paper

The paper "*Nonconvex Rigid Bodies with Stacking",* describes their novel contributions of plausible simulation. The paper firstly states their novel methods to present geometries with many friction activities. It combines two functions: triangulations of rigid body surface and signed distance. According to the paper, combining these two functions is useful. For example the signed distance function can be used to check "inside/outside" in a short time. Another advantage is this can provide more precise normals even for sharp objects.

Time integration is another novel contribution done in the paper. Although there are many precise algorithms designed for integrating time in free light, the result is not accurate if it has contact and collision. The new approach intends to separate collision and contact detection and this method eliminates velocity updating influenced by contact and collision.

Collison and contact are two main fields that need plausible algorithms. It is difficult to implement when there are many collisions, especially needs to consider the order. The algorithm in the paper says that collisions firstly temporarily put objects to the predicted locations and then check interference. Then processes pari objects if they intersect. Since collisions change velocity of objects, the algorithm would repeat several times and generate new velocity. The collision may lead to separation or non-seperation situation. For each intersecting pair, the team sets up vertices of the object and checks if they are inside another object or not. Note that all objects are stored in a list initially and consider their order based on appearance time. In the paper, separation vertex will be removed from a list, and continue processing rest points until all points are separated at once.

Contact modeling algorithm has a similar idea but without considering coefficient. The paper also contributes to the contact graph which suggests the order of contracts in order to generate a more efficient contact algorithm.

Further, computing collision means calculating kinetic and static friction as well. The paper describes in detail how the equation will be for computing these two frictions. While simulating rigid objects, rolling and spinning friction should also be taken into consideration. In the paper, authors treat these as kinetic and static friction.

Our team plans to implement these well-established algorithms in order to develop a framework that can simulate rigid bodies physical movement such as collision and friction.

2.2 OpenGL

In this project we use OpenGL to construct objects because it provides a convenient and complete environment for developing graphics. For implementing 3D objects, we applied OpenGL libraries: GLFW and GLAD. GLFW provides independent APIs to handle functions such as creating windows, contexts and surfaces, reading input, handling events. GLAD helps to generate our loader to satisfy versions of OpenGL drivers. It also supports loading debugging headers which helps for debugging locating. GLAD is used to manage function pointers as well, which helps to generate locations of functions during compile time.

3. Concrete goals and methods

In the proposal, we mentioned four concrete goals, they are Geometric Representation, Collision Model, Main Algorithm and Visualization. We will discuss the methods we used to achieve them.

At the beginning of the object, we defined a class termed *Object*, which will be used as base class for all of the objects in the *world*. This class includes some basic data members like vertices, indices, mass, velocity and etc. Then we define a *Sphere* class and *Plane* class. They inherit from the Object class. Some important methods are concrete in these classes like isIntersect(), updateVelocity() and updateVertices(). In order to save the memory, we initialize the vertices only once, then we apply indices to instruct visualize in OpenGL.

We do not save every vertex's absolute position and velocity directly, we save vertex position relative to the center of mass, this is the same as OpenGL. Then we apply translation matrix and rotation matrix to get vertex position in the world. Therefore, we only have to save the position of the center of mass, then vertex information can be calculated. So far, geometric representation and visualization parts are achieved.

As mentioned in the paper, there are four main steps in the algorithm: collision detection, advance velocity, contact resolution and position update.

Same as methods in the paper, collisions are detected by predicting where the objects will move to in the next time step, temporarily moving them there, and checking for interference. If there are no collisions, we want the object to be moved to the predict position. For example, Assume an object currently position and velocity are x and v, we test the interference using

the predicted position $x' = x + \Delta t\, v'$, such that $v' = v + \Delta tg$, and apply collision impulses to the current velocity v . Since collisions change the velocity of rigid bodies, new collisions might be occured after the update, we do the five iterations. Besides that, since the order of the objects list will also influence the result, we mix up this objects list regularly to reduce the influence.

For the interference detection, the paper proposes checking if some of the sample vertices of one object are in another one. This method is good for some irregular rigid bodies. However, this project just achieved plane and sphere, so we can use a pretty simple method to check the interference. And combine with the interference and the collisions, the collision detection part done.

When updating the velocity of the object, we assume the collision process is an elastic collision. So the energy and the momentum should stay the same before and after the collision. We assume the two objects have masses m_1 , m_2 and velocities v_1 , v_2 . After collision, the velocities becomes u_1 , u_2 . Then we have:

$$\tfrac{1}{2}m_1{v_1}^2 + \tfrac{1}{2}m_2{v_2}^2 = \tfrac{1}{2}m_1{u_1}^2 + \tfrac{1}{2}m_2{u_2}^2$$

$$m_1 v_1 + m_2 v_2 = m_1 u_1 + m_2 u_2$$

Solve this equation, we have:

$$u_1 = \frac{m_1 - m_2}{m_1 + m_2} v_1 + \frac{2m_2}{m_1 + m_2} v_2$$

$$u_2 = \frac{2m_1}{m_1 + m_2} v_1 + \frac{m_2 - m_1}{m_1 + m_2} v_2$$

This solves the magnitude of the velocity after the collision, and the direction is opposite with the original. Then, the velocity update part is finished.

Position update part is fairy simple, just update the center of mass's position according to the current position and velocity. However, the contact resolution part and rotation of the rigid body didn't achieved, this will be discussed in the next part.

4. Analysis and Conclusion

This project failed in contact resolution and the add rotation to the object. Mainly in solving the collision, I only consider the velocity update and forget to calculate the force that applied to the object. Which makes the contact part hard to achieve. Besides that, we didn't schedule well in the project, making us not have enough time to solve the rotation part.

We realize that we have to calculate the force when we place a sphere on a plane and with no velocity, this sphere will fall slowly and go through the plane. This bug from the missing of the contact part. We also meet another interesting bug: when we appling collision to a moving sphere and stay sphere, the new velocity to the moving sphere will be negative zero, -0.0. This will make it disappear in the scene. We solve this by adding a tiny number to the velocity, then, another object minus this tiny velocity. This can guarantee that negative zero will not show up.

If this project could be started over, we would think more carefully before code. We spend a lot of time rewriting the code. At first, we wrote the sphere and plane separately. Then, when we check the collision, we realize they cannot be stored in the same list and should code a lot in collision detection. This inspires us to use Object as base class, and we rewrite these. That time consuming and should not happen if we thought carefully beforehand.