Assignment 2 : Rigid body dynamics simulation

NAME: TIANYUAN WU
STUDENT NUMBER: 63305667
EMAIL: WUTY@SHANGHAITECH.EDU.CN

1 INTRODUCTION

In this assignment, I implemented an simple engine for rigid body dynamics simulation. The implementation is in C++14 with OpenGL. It's worth mentioning that this engine is **all implemented by me**, without any 3rd party libraries for calculation or rendering (except OpenGL). The workflow of this engine can be described as:

(1) Add objects (rigid bodies)
(2) Set initial configuration of rigid bodies
(3) Begin simulation (integration)
(4) Collision detection (using SAT Theorem)
(5) Collison handling
(6) Simulation by integration...

In my implementation, a general BasicShape class is provided, all shapes (Sphere, Cube, etc) can derive from this class. Also, a general `RigidBody` class is provided for the simulation. For every `RigidBody`, we can add velocity (v), angular velocity (ω), force(F) and torque(τ) to it, and the animation (result of simulation) can be automatically generated. What's fancy is, all implementation is done using templates, which means it's very easy to implement the simulation for other shapes once related API is provided.
For collision detection, I used a SAT Theorem based implementation, which can handle the collision between cubes well in practice.
For collision contact handling, my implementation will first calculate the normal of the face where the collision occurred. Then, the impluse J of this collision will be calculated. Finally, we will put apply this impluse for both objects, and re-calculate the velocity and angular velocity. Here is a screenshot of the my implementation.
All of these functions are tested under `macOS 10.15`, with `Apple clang 12.0.0`.

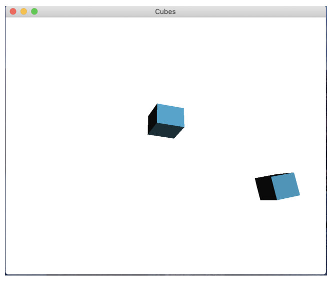

Fig. 1. Screenshot of my implementation

2 IMPLEMENTATION DETAILS

The implementation of this engine can be described in 3 parts: (1) Basic rigid body dynamics without collision handling; (2) Collision detection; (3) Collision contact handling.

2.1 Basic rigid body dynamics

In my implementation of rigid body dynamics, every rigid body hold its own velocity (v), angular velocity (ω), force(F) and torque(τ). Every rigid will update its position and orientation in `update()` funtion. The simulation process can be described by the following pseudo code: In `update()` function, we need to re-calculate the

Algorithm 1 Simulation process

```
1: while true do
2:     for object : rigidBodies do
3:         object.update();
4:         object.show();
5:     end forShowObjectsOnWindow();
6: end while
```

velocity by given force and angular velocity by given torque.
By Newton's law:

$$\mathbf{F} = m\mathbf{a}$$

we can get the change of velocity:

$$d\mathbf{v} = \frac{\mathbf{F}}{m}dt$$

The change of angular velocity is more complicated. In this case (all object are cubes), so the inertia tensor in its own coodinates is:

$$\mathbf{I}_{local} = \begin{bmatrix} \frac{1}{6}ml^2 & 0 & 0 \\ 0 & \frac{1}{6}ml^2 & 0 \\ 0 & 0 & \frac{1}{6}ml^2 \end{bmatrix}$$

In my implementation, the orientation is represented by a rotation matrix $\mathbf{R}$. So, in world coodinates, the inertia tensor can by calculated as:

$$\mathbf{I}_{global} = \mathbf{R}\mathbf{I}_{local}\mathbf{R}^T$$

Hence, the change of ω is:

$$d\omega = \mathbf{I}_{global}^{-1}\tau(t_i)dt$$

Then, the new position s can be calculated by:

$$d\mathbf{s} = \mathbf{v}dt$$

and the new orientation (rotation matrix) can be calculated by:

$$d\mathbf{R} = \omega^*\mathbf{R}dt$$

where ω^* is the following matrix:

$$\omega^* = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix}$$

Hence, we can update the orientation and position by steps shown above in `update()` function.

2.2 Collision Detection

In my engine, a SAT Theorem based collision detector is implemented. For all objects are cubes, it's very easy to get the normals and coodinates of vertices, so we can detect collisions between cube A and B by following pseudo-code:

Algorithm 2 SAT collision detection

```
1:  for norm  norms_A do // 6 norms for a cube
2:      BOOL allOutside = True;
3:      for v  vertices_B do // 8 vertices for a cube
4:          if $v \cdot norm < 0$ then
5:              allOutside = False;
6:          end if
7:      end for
8:      if allOutside then
9:          return False;
10:     end if
11: end for
12: return True;
```

2.3 Collision Contact Handling

When contact collision is detected, by the algorithm shown above, we'll also know the norm of collision and the collision point. So, we can calculate the new velocity and angular velocity by the conversion of momentum and angular momentum. The impluse of the collision is denote as J, thus the change of velocity and angular velocity can be calculated by:

$$\mathbf{F}dt = \mathbf{J}$$

$$\tau_{impluse} = (p - x(t)) \times \mathbf{J}$$

$$d\omega = I^{-1}(t_0)\tau_{impluse}$$

And J can be calculated by:

$$J = j\hat{n}(t_0)$$

where

$$j = \frac{-(1+\epsilon)v_{rel}^{-}}{1/M_a + 1/M_b + \hat{n}(t_0)(I_a^{-1}(r_a \times \hat{n}(t_0))) \times r_a + \hat{n}(t_0)(I_b^{-1}(r_b \times \hat{n}(t_0))) \times r_b}$$

Hence, we can calculate the velocity and angular velocity.

3 RESULTS

In this assignment, basic rigid body dynamics, collision detection and collision contact handling are all implemented. Also, I designed some test cases to test different conditions, the test results are provided below.

(1) Basic rigid body dynamics (Cube 1 is spinning without linear motion under a constant angular velocity; Cube 2 is also spining without linear motion, but with a constant Torque τ, so the ω of Cube2 will increase by time. Cube3 is doing a linear motion with a constant velocity, while Cube 4 is doing a constant acceleration. Cube 5 is doing a linear motion plus self spining).

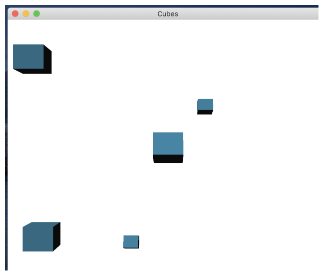

Fig. 2. Basic rigid body dynamics

(2) Collision detection (we can set the system "paused" when a collision is detected, so that we can get this screenshot)

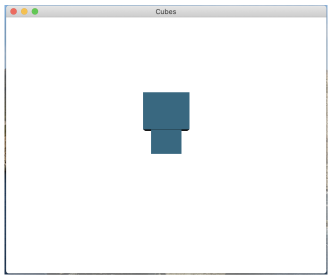

Fig. 3. Collision detection

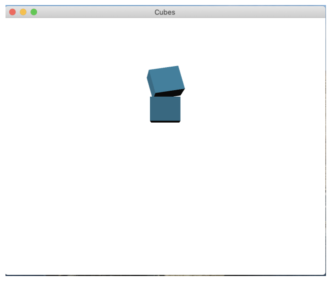

Fig. 4. Collision detection

(3) Collision contact handling. This figure shows the position and orientation of these 2 cubes after collision occured.

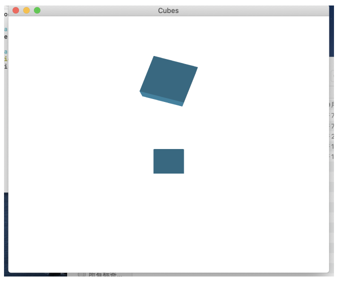

Fig. 5. Collision contact handling

For the performance, the FPS (frame per second) of my engine can reach ~60fps under ~10 cubes (test platform is Intel i5-9400f and macOS 10.15, the calculation is done single-threaded, and the complier is `Apple clang 12.0.0`.