

Assignment 4 : Rigid body simulation

NAME: NING BI

STUDENT NUMBER: 85745238

EMAIL: BINING@SHANGHAITECH.EDU.CN

1 INTRODUCTION

In this assignment, I created a scene with a rigid box and a floor. The rigid body dynamics for box with translate and rotation are shown in this scene. A box will fall on the ground and bounce few times.

The task I have completed are as follow:

- (1) basic rigid body dynamics.
- (2) Collision detection and contact treatment for a box object.

2 IMPLEMENTATION DETAILS

2.1 Rigid body representation

A rigid body is defined by a solid shape with mass distribution. The body is also associated with its state, such as the position of center of mass as well as the rotation about the local coordinate system. The RigidBody is defined as below:

```
class RigidBody
{
    Matrix3x3 Io; // constant inertia tensor
    Rigidstate rigidStateNew;
    Rigidstate rigidState;
    StateDot rigidStateDot; // delta state
    Vector3d center;
    Vector3d bodyForce;
    Vector3d vertexForce[8];
    Vector3d Vpn; // plane normal
    Vector3d vertexPos[number];
    Vector3d vertexPosNew[number];
}
```

A constant (in body-space) 3 x 3 intrinsic inertia tensor I_0 , which is pre-computed in RigidBody class.

```
Matrix3x3 Io = Matrix3x3(
    mass*1.0 / 12 * (size_*size_* 2), 0, 0,
    0, mass*1.0 / 12 * (size_*size_* 2), 0,
    0, 0, mass*1.0 / 12 * (size_*size_* 2)
);
```

The time-dependent state as class Rigidstate, which includes its position X , the quaternion Q , linear momentum P and angular momentum L .

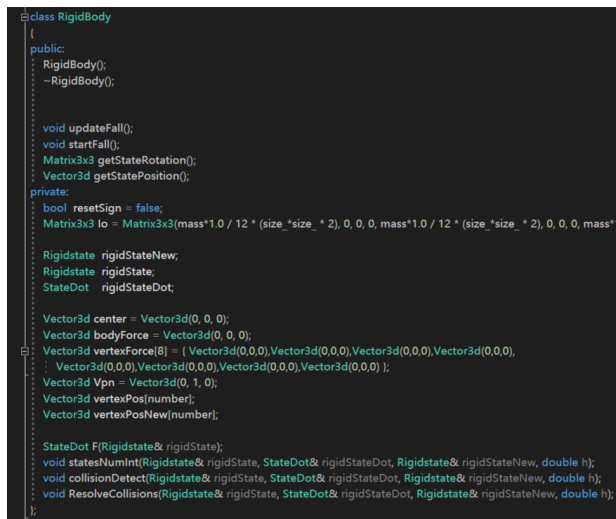
```
class Rigidstate {
```

```
public:
    Vector3d xposition;
    Quaternion quater;
    Vector3d pfmom;
    Vector3d lamom;
}
```

Quaternion is used to update the rotation of the rigid body
 $\text{Matrix3x3 } R = \text{rigidState.quater.rotation}();$

State class StateDot is defined to store some temporary quantities to assist the motion process, which can be directly used to update the state over time.

```
class StateDot {
public:
    Vector3d velocity; // linear velocity
    Quaternion quaterA; // differentiation of quaternion
    Vector3d force;
    Vector3d torque;
}
```



```
class RigidBody
{
public:
    RigidBody();
    ~RigidBody();

    void updateFall();
    void startFall();
    Matrix3x3 getStateRotation();
    Vector3d getStatePosition();
private:
    bool resetSign = false;
    Matrix3x3 Io = Matrix3x3(mass*1.0 / 12 * (size_*size_* 2), 0, 0, 0, mass*1.0 / 12 * (size_*size_* 2), 0, 0, 0, mass*1.0 / 12 * (size_*size_* 2));

    Rigidstate rigidStateNew;
    Rigidstate rigidState;
    StateDot rigidStateDot;

    Vector3d center = Vector3d(0, 0, 0);
    Vector3d bodyForce = Vector3d(0, 0, 0);
    Vector3d vertexForce[8] = { Vector3d(0,0,0), Vector3d(0,0,0), Vector3d(0,0,0), Vector3d(0,0,0),
        Vector3d(0,0,0), Vector3d(0,0,0), Vector3d(0,0,0), Vector3d(0,0,0) };
    Vector3d Vpn = Vector3d(0, 1, 0);
    Vector3d vertexPos[number];
    Vector3d vertexPosNew[number];

    StateDot F(Rigidstate& rigidState);
    void statesNumInt(Rigidstate& rigidState, StateDot& rigidStateDot, Rigidstate& rigidStateNew, double h);
    void collisionDetect(Rigidstate& rigidState, StateDot& rigidStateDot, Rigidstate& rigidStateNew, double h);
    void ResolveCollisions(Rigidstate& rigidState, StateDot& rigidStateDot, Rigidstate& rigidStateNew, double h);
}
```

Fig. 1. RigidBody Class

2.2 Rigid body dynamics

Euler's explicit method is implemented to update the state at each frame.

$$y(x_{n+1}) \approx y(x_n) + hf(x_n, y(x_n))$$

1:2 • Name: NING Bi
 student number: 85745238
 email: bining@shanghaitech.edu.cn
 In statesNumInt() the center point in new state is computed as:

$$x(t + \Delta t) = x(t) + v(t)\Delta t$$

$$v(t) = p(t)/mass$$

The quaternion of new rigid state is computed as:

$$q(t + \Delta t) = q(t) + \frac{1}{2}\omega(t)q(t)\Delta t$$

Linear momentum and angular momentum for the next state:

$$p(t + \Delta t) = p(t) + F(t)\Delta t$$

$$l(t + \Delta t) = l(t) + \tau(t)\Delta t$$

Then update next particle positions of the rigid body.

```
Vector3d centerNew = rigidStateNew.xposition;
Matrix3x3 R = rigidStateNew.quater.rotation();
vertexPosNew[i] = R * vertexInBoxSpace[i] + centerNew;
```

```
class Rigidstate {
public:
    Vector3d xposition;
    Quaternion quater;
    Vector3d pfmom;
    Vector3d lamom;

    Rigidstate() : xposition(Vector3d(0, 0, 0)),
                  quater(Quaternion()),
                  pfmom(Vector3d(0, 0, 0)),
                  lamom(Vector3d(0, 0, 0))
    {
    }
}
```

Fig. 2. RigidState

2.3 Rigid object dynamics with collision

In this assignment, only one box will collide with the ground. So only vertex/face collision detection is implemented. For eight corners of the box, its distance to the floor is calculated.

$\text{float axbyczd} = \text{DotProduct}(\text{Position}, \text{floor.Normal}) + \text{floor.d}$

where Position is the eight corners' position, floor.d is the floor equation's intercept.

2.4 Rigid object dynamics with resting contact

When the box is considered to stop motion, the velocity need to be sat to zero.

```
do
    ResolveCollisions(cornerIndex);
```

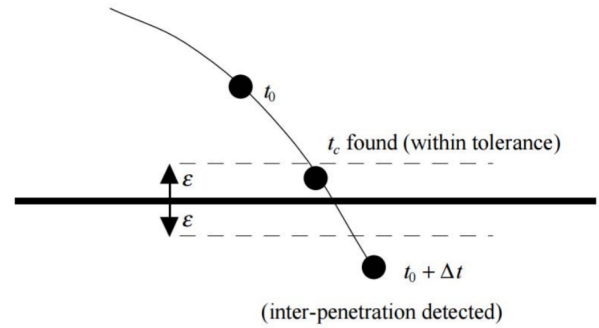


Fig. 3. Rigid object dynamics with collision

```
Counter++;
while (
    (CheckForCollisions() == Collision) &&
    (Counter < LIMIT));
```

3 RESULT

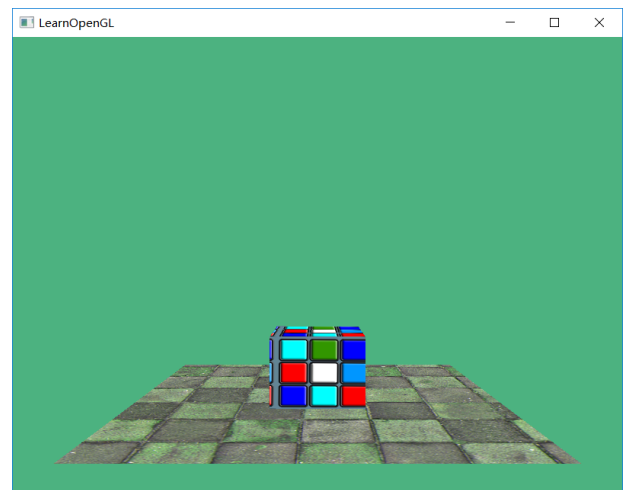


Fig. 4. The results



Fig. 5. The results