Assignment 4 : Rigid Body Simulation

NAME: ZHAO DONGHAO
STUDENT NUMBER: 20096438
EMAIL: ZHAODH@SHANGHAITECH.EDU.CN

1 INTRODUCTION

In this assignment, a simple rigid body simulation for 3D computer animation is completed. In rigid body simulation, the basic operation is to translate and rotate the geometries over time. The key question is how to determine these translations and rotations based on the mass distribution of the rigid body as well as the external forces. In addition, collision detection and contact treatment are also done in order to make the rigid body dynamics realistic.

2 IMPLEMENTATION DETAILS

Rigid Body Dynamics

The system of bodies are updated at each frame in function updateFall(). First, gravity and net torques are applied to each body. Next, the velocities of the rigid body are integrated. In case of constraint, the solver corrects the velocities at the collision in order to enforce all constraints. Here, collision detection are performed. Finally, the new state are evolved over time again.

Ordinary differential equations for the rigid body dynamics are solved by solution in function statesNumInt() and F().In every frame we call statesNumInt() to get new state of our box, and h is given as the step size between two states. In statesNumInt() center point in new state is computed.Next quaternion of new rigid state is computed and update the quaternion, and then the angular velocity is computed.

Next I obtain net force and torque. For net force, you need to sum up gravity at all vertices. And the computation of net torque is similar, vertex position is towards the center of mass, then linear momentum and angular momentum for the next state are got. Next particle positions of your rigid body are updated.

Collision Detection

In my assignment, I only consider vertex/face collision detection (implemented collision part in function collisionDetect()). For eight corners of the box, I calculate its distance to the floor.If the distance is less than a small threshold, the corner point is interpenetrating the floor. If the distance is less than the threshold, the corner point will collide with the floor. If the distance is larger than epsilon, the rigid body's property is uodated. For penetration case, I decrease the update time and repeat the evolution and collision detection process.

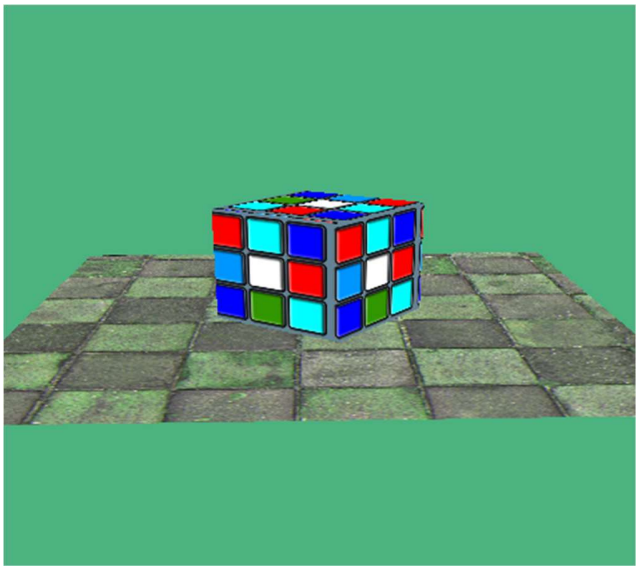
Colliding Contact

When colliding contact happens, the velocity of rigid bodies change instantly to prevent penetrating into another object. First, calculate the velocity of the vertex point. Next, calculate relative velocity. Base on the sign of the relative velocity, there are three cases depend on Vrel.

Resting Contact

Resting contact refers to the contact where there is no relative motion between two objects when they are in contact. In this case, a contact force is computed automatically to enforce a non-penetrate constraint.We consider the condition of resting contact to be applied when the relative velocity Vrel is smaller than thresholdWhen the box keeps contacting with the floor for many times, we believe the box should stop its motion.We keep computing the contact force and collision detection until the time limit is reached. When the box is considered to stop motion, the velocity is set to zero.

3 RESULTS



1:2 • Name: Zhao Donghao
student number: 20096438
email: zhaodh@shanghaitech.edu.cn

```
void RigidBody::updateFall()
{
    unsigned int counter = 0;
    for (int i = 0; i<number; i++) {
        if (vertexPosNew[i].y < 0) {
            counter++;
        }
    }
    if (counter > 1) {
        resetSign = true;
        return;
    }

    double hStep = 0.01;

    statesNumInt(rigidState, rigidStateDot,
        rigidStateNew, hStep);
    collisionDetect(rigidState, rigidStateDot,
        rigidStateNew, hStep);

    for (int i = 0; i<number; i++) {
        vertexForce[i] = Vector3d(0, 0, 0);
    }

    rigidState = rigidStateNew;
    center = rigidState.xposition;
    for (int i = 0; i<number; i++) {
        vertexPos[i] = vertexPosNew[i];
    }

}
```

```
void RigidBody::statesNumInt(Rigidstate& rigidState,
    StateDot& rigidStateDot, Rigidstate& rigidStateNew, double h)
{
    StateDot K1 = F(rigidState);
    rigidStateDot = K1;
    rigidStateNew = rigidState + rigidStateDot * (h);

    Vector3d centerNew = rigidStateNew.xposition;
    Matrix3x3 R = rigidStateNew.quater.rotation();

    vertexPosNew[0] = R * (Vector3d(-halfSize, -halfSize, halfSize)) + centerNew;
    vertexPosNew[1] = R * (Vector3d(halfSize, -halfSize, halfSize)) + centerNew;
    vertexPosNew[2] = R * (Vector3d(halfSize, -halfSize, -halfSize)) + centerNew;
    vertexPosNew[3] = R * (Vector3d(-halfSize, -halfSize, -halfSize)) + centerNew;
    vertexPosNew[4] = R * (Vector3d(-halfSize, halfSize, halfSize)) + centerNew;
    vertexPosNew[5] = R * (Vector3d(halfSize, halfSize, halfSize)) + centerNew;
    vertexPosNew[6] = R * (Vector3d(halfSize, halfSize, -halfSize)) + centerNew;
    vertexPosNew[7] = R * (Vector3d(-halfSize, halfSize, -halfSize)) + centerNew;
}
```

```
StateDot RigidBody::F(Rigidstate& rigidState)// compute force
{
    StateDot rigidStateDot;

    rigidStateDot.velocity = (1.0 / mass)*rigidState.pfmom;
    Matrix3x3 R = rigidState.quater.rotation();
    Matrix3x3 Iinverse = R * (Io.inv()) * (R.transpose());
    Vector3d w = Iinverse * rigidState.lamom;
    rigidStateDot.quaterA = 0.5 * w * rigidState.quater;

    Vector3d totalforce(0, 0, 0);
    for (int i = 0; i<number; i++) {
        totalforce = totalforce + vertexForce[i];
    }
    totalforce = totalforce + bodyForce;
    rigidStateDot.force = totalforce;

    Vector3d totalTorque(0, 0, 0);
    Vector3d torque(0, 0, 0);
    for (int i = 0; i<number; i++) {
        torque = (vertexPos[i] - center) % vertexForce[i];
        totalTorque = totalTorque + torque;
    }
    rigidStateDot.torque = totalTorque;

    return rigidStateDot;
}
```

```
void RigidBody::collisionDetect(Rigidstate& rigidState, StateDot& rigidStateDot, Rigidstate& rigidStateNew, double h)
{
    for (int i = 0; i<number; i++) {
        if ((vertexPosNew[i].y - wallY)*(vertexPos[i].y - wallY)<0) { //下一时刻即将穿透
            if ((vertexPosNew[i] - vertexPos[i]).norm() > 0.1) {
                while (1)
                {
                    statesNumInt(rigidState, rigidStateDot, rigidStateNew, h*0.5);
                    if (vertexPosNew[i].y <wallY) {
                        break;
                    }
                    else
                    {
                        rigidState = rigidStateNew;
                        center = rigidState.xposition;
                        for (int i = 0; i<number; i++) {
                            vertexPos[i] = vertexPosNew[i];
                        }
                    }
                }
                collisionDetect(rigidState, rigidStateDot, rigidStateNew, h*0.5);
            }
            else
            {
                //resting
                if (rigidStateDot.velocity.norm()< thr) {
                    unsigned int counter = 0;
                    for (int i = 0; i<number; i++) {
                        if (vertexPosNew[i].y < thr) {
                            counter++;
                            break;
                        }
                    }
                }

                Matrix3x3 R = rigidState.quater.rotation();
                Matrix3x3 Iinverse = R * (Io.inv()) * (R.transpose());
                Vector3d w = Iinverse * rigidState.lamom;
                Vector3d r = vertexPos[i] - center;

                double Vn = Vpn * (rigidStateDot.velocity + w % r);
                double j = -1 * (1 + Cr)*Vn / (1.0 / mass + Vpn * (Iinverse*(r%Vpn) % r));
                Vector3d J = j * Vpn;

                rigidState.pfmom = rigidState.pfmom + J;
                rigidState.lamom = rigidState.lamom + r % J;

                statesNumInt(rigidState, rigidStateDot, rigidStateNew, h);
            }
        }
    }
}
```