Assignment 5:
Rigid Body Simulation

NAME: YANG HONGDI
STUDENT NUMBER: 2019533234
EMAIL: YANGHD@SHANGHAITECH.EDU.CN

1 INTRODUCTION

In this assignment, simple rigid body simulation is performed without considering rotation. The intersection between spheres and walls and intersection between spheres is performed.

2 IMPLEMENTATION DETAILS

2.1 Make the Scene Move

To make the scene move, we update the spheres' P(linear momentum), speed and position every dt. As

$$P = F * t = M * v$$

we update sphere's state by function below.

```
void Sphere::Forward(float dt)
{
    for (Force const_force : constant_forces)
    {
        this->P += const_force.F * dt;
    }
    for (Force trans_force : transient_forces
        )
    {
        this->P += trans_force.F * dt;
    }
    this->v = this->P / this->M;
    this->x += this->v * dt;
    UpdateOpenGLObject();
}
```

2.2 Collision Detect

2.2.1 SphereParallelgoram Collision Detection. For collision detection between sphere and parallelgoram, we first check the distance between sphere and the plane the parallelgoram is on.

```
  vec3 normal = glm::normalize(glm::cross(
      parlgrm_s1, parlgrm_s2));
    vec3 temp_line = parlgrm_x - sphere_x;
    float dist = abs(glm::dot(normal,
        temp_line));

    if (dist - sphere_r > 0) // not intersect
    {
        collision_info.is_collided = false;
        return false;
    }
```

Then we will check if the intersect circle is in the parallelgoram. We first check if the collision point is in the parallelgoram, and then we will check if the circle intersect with each edge of the parallelgoram.

2.2.2 SphereSphere Collision Detection. For collision detection between spheres, we simply compute the distance between the center of spheres. If the distance is smaller than the sum of two spheres' radius, then the two spheres are collided.

2.3 Adjust Collision

As we are updating the spheres' state discontinuously, it is possible that the sphere would collide with wall at a deep point in the wall, so the sphere may not be able to get out next dt and then it will go into the wall. To prevent this, we need to adjust the collision.
To adjust the collision, we need to find a time between t and $t + \Delta t$ where the intersection degree is in the intersection tolerance.

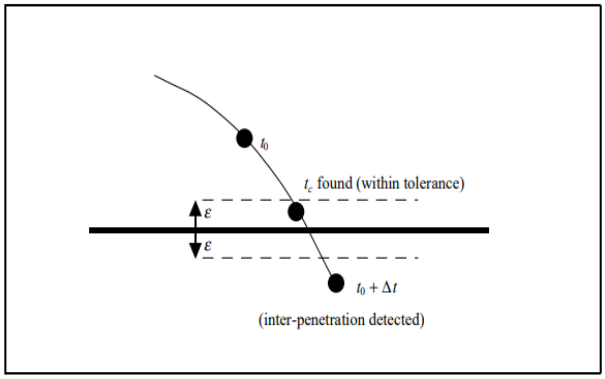


Fig. 1. find tc in tolerance

so we find the suitable time t_c recursively between t and $t + \Delta t$.

```
void Scene::FindRealCollision(float&
    intersect_degree, float delta_t)
{
    if (!IsCollided(intersect_degree))
    {
        this->Forward(delta_t / 2);
        return FindRealCollision(
            intersect_degree, delta_t / 2);
```

```
    }
    else if (intersect_degree -
        intersect_tolerance > 1e-4)
    {
        this->Backward(delta_t);
        this->Forward(delta_t / 2);
        return FindRealCollision(
            intersect_degree, delta_t / 2);
    }
}
```

2.4 Handle Colliding contact

2.4.1 Handle collision between sphere and wall. When handling collison between sphere and wall, as wall is static, we consider its Mass as infinity and zero speed.
Then we compute the sphere's speed along the normal direction of the wall.

```
vec3 sphere_speed = sphere.GetV();
vec3 normal = glm::normalize(sphere.GetX()
    - temp_collision_info.collision_point);
float vrel = glm::dot(sphere_speed, normal)
    ;// relative speed on normal direction
vec3 v_normal = vrel * normal;
```

If $v_{rel} > threshold$, we consider the ball is leaving the wall, thus not collided.
If $v_{rel} > -threshold$, we consider the ball has resting contact with the wall, and we will apply support force on the sphere.

```
Force sphere_press_force;
for (Force force : sphere.constant_forces)
    sphere_press_force.F += force.F;
for (Force force : sphere.transient_forces)
     sphere_press_force.F += force.F;
sphere_press_force.F = glm::dot(
    sphere_press_force.F, normal) * normal;
Force N;
N.F = -sphere_press_force.F; // support
    force
```

And if neither of above, we do the real collision handling.
We consider the collision make the sphere's relative speed v_{rel} become $-0.8 * v_{rel}$ in dt time, so we compute F by

$$F = \frac{M \cdot -(1+0.8) \cdot v_{rel}}{dt}$$

During collision, we also consider the speed will decrease 0.01 along the tangent direction, so we also add a Force $F = \frac{M \cdot -(1-0.99) \cdot v_{tan}}{dt}$ to the sphere.

2.4.2 Handle collision between spheres. For the collision between spheres, we also first compute the relative speed along the intersection direction. The relative speed is considered as $v_1 - v_2$

```
vec3 sphere1_speed = sphere1.GetV();
vec3 sphere2_speed = sphere2.GetV();
vec3 collision_normal = glm::normalize(
    sphere1.GetX() - sphere2.GetX());
float vrel = glm::dot(sphere1_speed -
    sphere2_speed, collision_normal);
vec3 v_normal = vrel * collision_normal;
```

Then, same as above, we check if the sphere is leaving each other or having resting contact.
If the spheres are having real collision, we consider the collision make the relative speed v_{rel} become $-0.8 * v_{rel}$, and we compute the total collision force F.

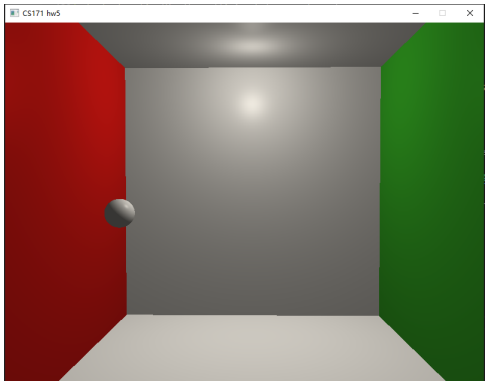
$$F = \frac{-(1+0.8) \cdot v_{rel}}{(\frac{1}{M_1} + \frac{1}{M_2}) \cdot dt}$$

And for each ball, we add $\frac{F}{2}$ on it.

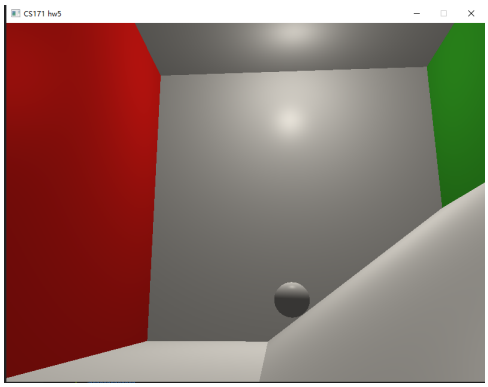
The transient forces will be cleared immediately after updating the sphere for next dt.

3 RESULTS

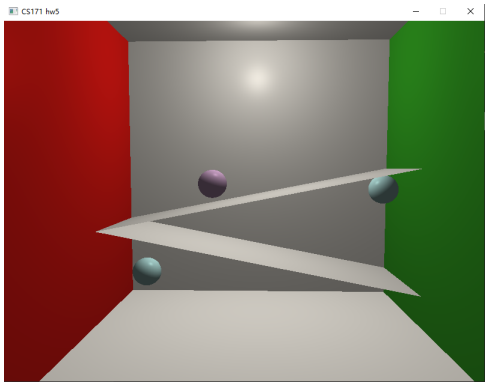
The results will be showed in real-time to TAs.



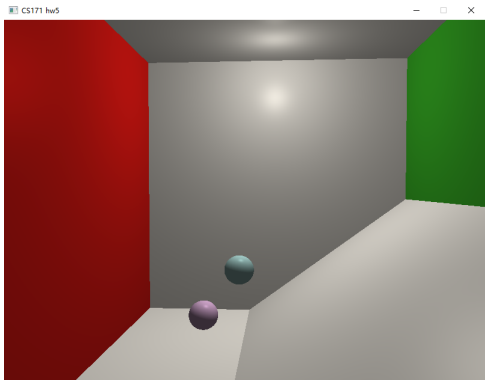
(a) Scene0



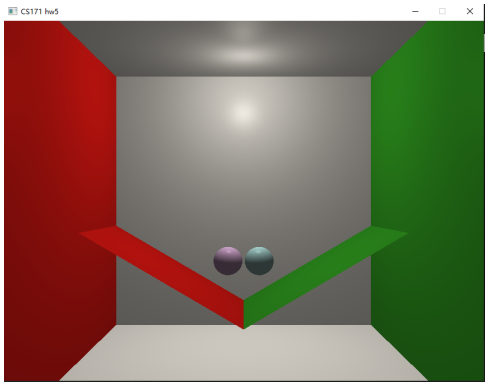
(b) Scene1



(c) Scene2



(d) Scene3



(e) Scene4