Rigid Body Simulation

Xinlong Yi, Zhuocheng Shang

Outline

- Introduction
- Concrete Goals
- Result and Analysis
- Conclusion

Introduction

Why we choose rigid body simulation?

- The importance and useful of rigid body

The basic goals of our project

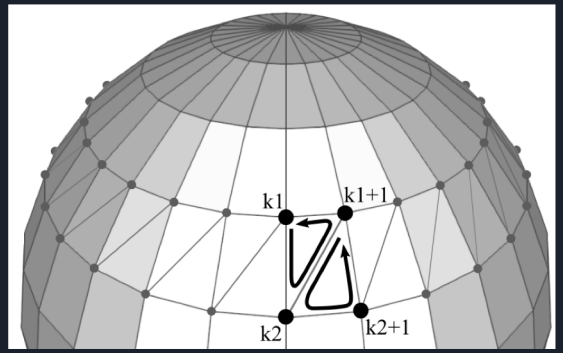
- Visualize basic rigid bodies in 3D scene.
- Simulate the rigid bodies' collision and rotation
- Implement the algorithm in "*Nonconvex Rigid Bodies with Stacking*"

Concrete Goals

- Geometric Representation
- Visualization
- Collision Model
- Main Algorithm

Geometric Representation

Sphere



- Generate vertices on the surface of the sphere.
- In horizontal direction, every two vertices and center generate same angle.
- Same as vertical direction
- Using indices of vertex to save the memory

Interference Detecting

Paper method: Check some sample vertices of one object in another one or not.

Pros: Can be generalize to irregular rigid bodies

Cons: Need more calculation

Our method: Applying geometric method to detecting intersect or not

Pros: Easy to compute

Cons: Cannot generalize to other object

Collision Model

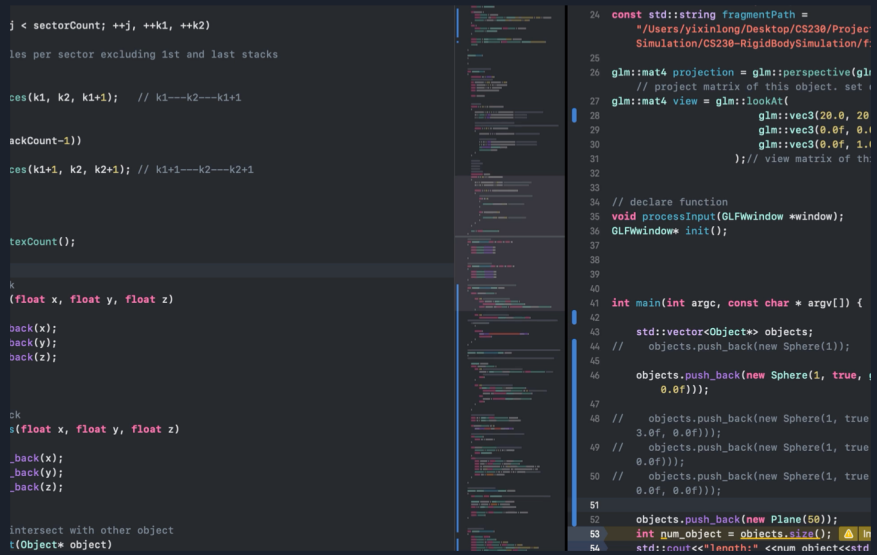
- Check every pair of objects in the scene.
- Using position of next time step x’ to check interference. Where

$$x' = x + \Delta t(v + g\Delta t)$$

- Applying collision impulse to current velocity v
- Update the velocity

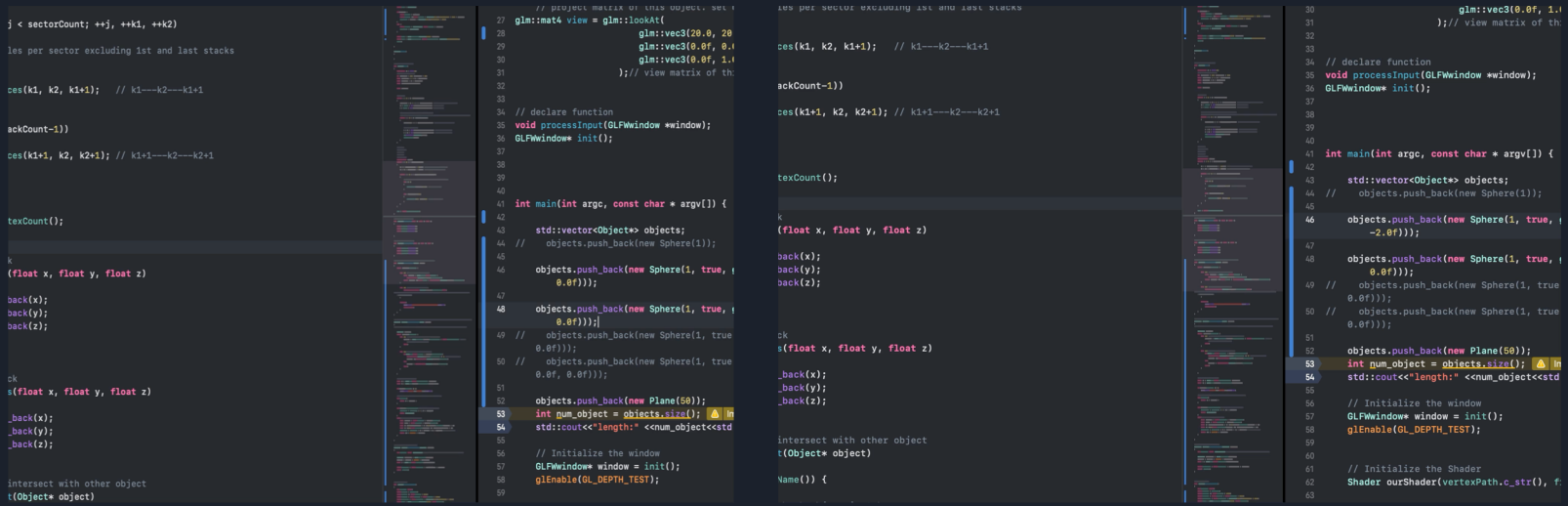
Result

Bouncing ball



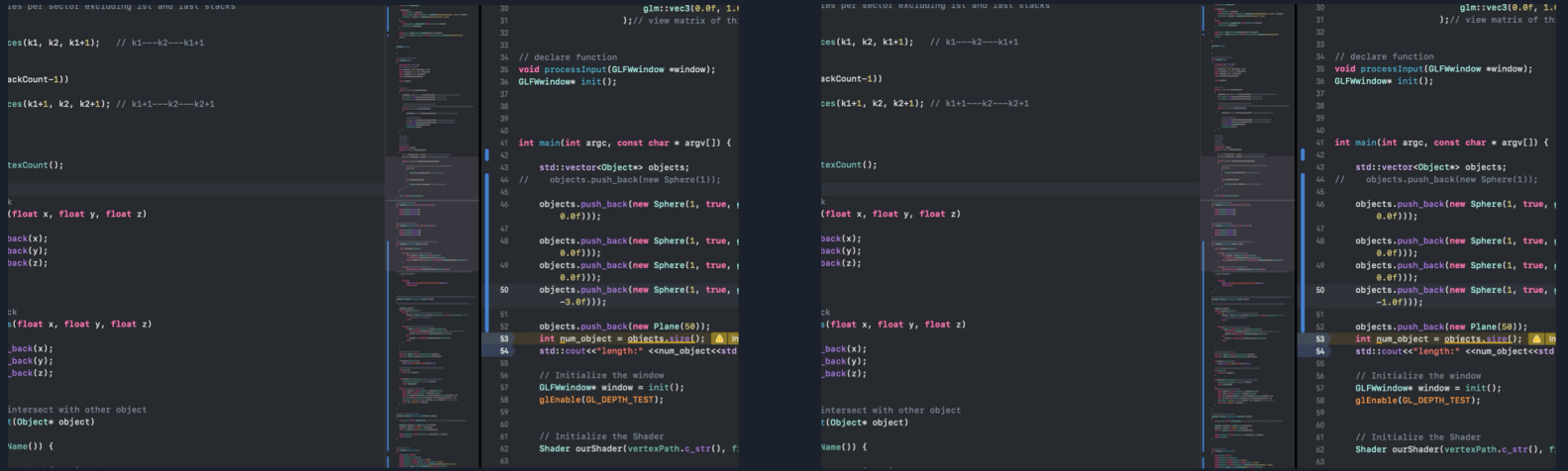
Result

Ball Collisions



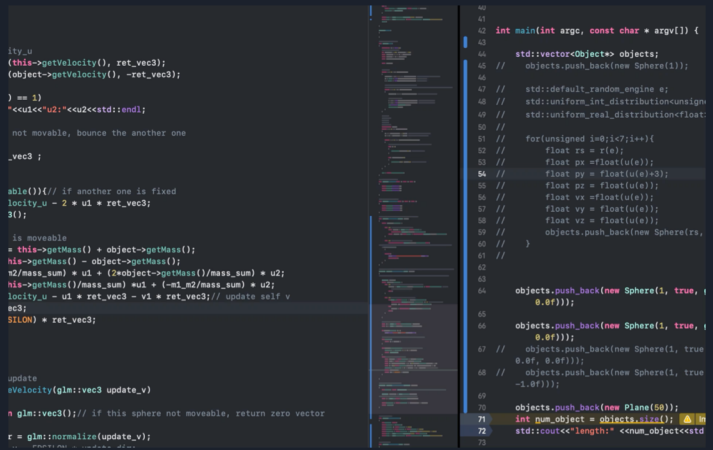
Result

Momentum Propagation

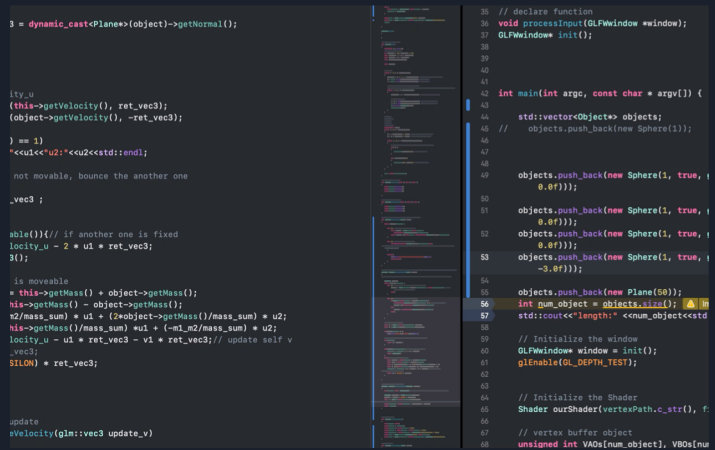


Analysis

Bugs



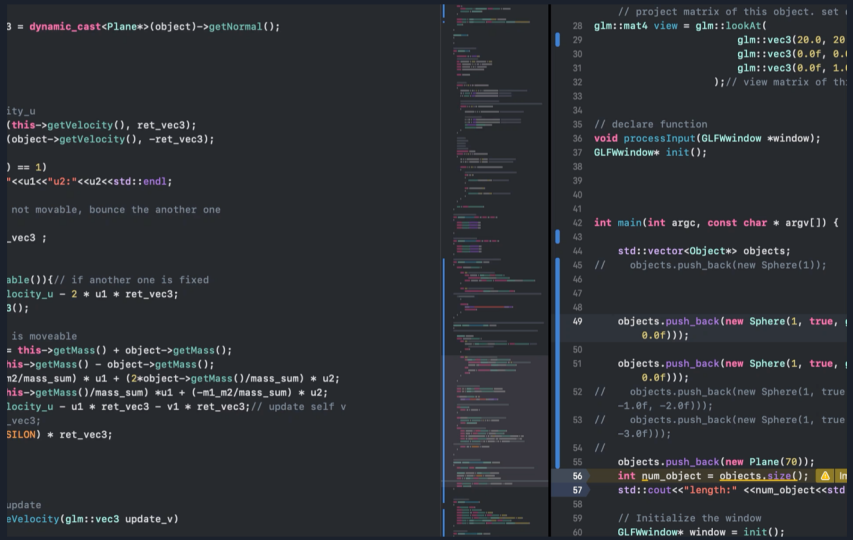
Negative Zero



Update Iteration

Analysis

Bugs



```
3 = dynamic_cast<Plane*>(object)->getNormal();
ity_u
(this->getVelocity(), ret_vec3);
(object->getVelocity(), -ret_vec3);
) == 1)
"<<u1<<"u2:"<<u2<<std::endl;
 not movable, bounce the another one
_vec3 ;
able()){// if another one is fixed
locity_u - 2 * u1 * ret_vec3;
3();
 is moveable
= this->getMass() + object->getMass();
his->getMass() - object->getMass();
m2/mass_sum) * u1 + (2*object->getMass()/mass_sum) * u2;
his->getMass()/mass_sum) *u1 + (-m1_m2/mass_sum) * u2;
locity_u - u1 * ret_vec3 - v1 * ret_vec3;// update self v
_vec3;
SILON) * ret_vec3;
update
eVelocity(glm::vec3 update_v)
```

```
    // project matrix of this object. set
glm::mat4 view = glm::lookAt(
                        glm::vec3(20.0, 20
                        glm::vec3(0.0f, 0.
                        glm::vec3(0.0f, 1.
                    );// view matrix of th

// declare function
void processInput(GLFWwindow *window);
GLFWwindow* init();

int main(int argc, const char * argv[]) {
    std::vector<Object*> objects;
//    objects.push_back(new Sphere(1));

    objects.push_back(new Sphere(1, true,
        0.0f)));
    objects.push_back(new Sphere(1, true,
        0.0f)));
//    objects.push_back(new Sphere(1, true
    -1.0f, -2.0f)));
//    objects.push_back(new Sphere(1, true
    -3.0f)));
//
    objects.push_back(new Plane(70));
    int num_object = objects.size();
    std::cout<<"length:" <<num_object<<std

    // Initialize the window
    GLFWwindow* window = init();
```

Object Stacking

Conclusion

- Failed in contact resolution part. We only considered elastic collision and update velocity in a naive way, fail to take force into consideration. This makes us have to rewrite most part of code to add contact part.

- Failed in rotation part. If project could start over, we would think more carefully and thoroughly before code.

Questions ?