# **Project: Massive Rigid-Body Simulation**

NAME: LIU YIFEI

EMAIL: LIUYF7@SHANGHAITECH.EDU.CN

YAN YIHENG NAME:

**EMAIL:** YANYH1@SHANGHAITECH.EDU.CN

#### **ACM Reference Format:**

Name: Liu Yifei, email: liuyf7@shanghaitech.edu.cn, Name: Yan Yiheng, email: yanyh1@shanghaitech.edu.cn . 2023. Project: Massive Rigid-Body Simulation. 1, 1 (January 2023), 8 pages. https://doi.org/10.1145/ nnnnnnn.nnnnnnn

#### Contents

Contents		Ī
1	Introduction	1
2	Implementation Details	1
2.1	Engine pipeline	1
2.2	Collision pair calculation	1
2.3	Constraints	(
3	Results	8
3.1	Division of labor	8

## 1 INTRODUCTION

In this project, we implemented a simple and complete convex polygon physics engine based on Erin Catto's "Iterative Dynamics with Temporal Coherence" and other modern physics engines. In short, we implemented the main modules of mechanics calculation, collision detection, and constraint solving on top of the basic components of the previous homework, and managed to simulate the collision mechanics between objects and object environments with good results.

## 2 IMPLEMENTATION DETAILS

# 2.1 Engine pipeline

In the mass spring homework, we have implemented two parts to analyze the forces and update the velocity and position. The overall flow proposed by Erin Catto is shown in Figure

Author's address: Name: Liu Yifei email: liuyf7@shanghaitech.edu.cn

Yan Yiheng

email: yanyh1@shanghaitech.edu.cn.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2023 Association for Computing Machinery.

XXXX-XXXX/2023/1-ART \$15.00

https://doi.org/10.1145/nnnnnnn.nnnnnnn

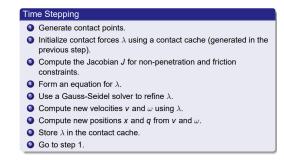


Fig. 1. Erin Catto's pipeline

# 2.2 Collision pair calculation

Imagine a scene with N objects, if we perform collision detection between every two of them, the computational complexity required is  $O(N^2)$ , which is obviously unacceptable for a computer. So we divide the collision detection into two phases, Broad-Phase and Narrow-Phase.

Broad-Phase uses some Bounding Volume to represent the collision information of rigid bodies, and then saves these Bounding Volumes in a spatial division, so that the rigid body pairs that may collide with each other can be selected in a short time.

## 2.2.1 Broad-Phase.

By using some kind of Bounding Volume to represent the collision information of rigid bodies, and then using spatial division to save these Bounding Volumes, it is possible to filter the rigid body pairs that may collide with each other in a short time. The usual methods are AABB (axis-aligned bounding boxes), OBB (oriented bounding boxes), Circle/Sphere.



Fig. 2. Broad-Phase VH type

```
std::vector<std::pair<Collider*, Collider*>>&
     BroadPhase::ComputePairs()
colliderPairs.clear();
#pragma omp parallel for schedule(dynamic,1)
for (int iA = 0; iA < aabbs.size(); iA++)</pre>
```

```
2 · Name: Liu Yifei
    email: liuyf7@shanghaitech.edu.cn
   Name: Yan Yiheng
        yanyh1@shanghaitech.edu.cn
                                                                     v[0] = glm::vec3(aabbL->min.x, aabbL
6
                                                        40
        for (int iB = iA + 1; iB < aabbs.size();</pre>
                                                                         ->min.y, aabbL->min.z);
7
                                                                     v[1] = glm::vec3(aabbL->max.x, aabbL
            iB++)
                                                        41
8
                                                                         ->min.y, aabbL->min.z);
        {
9
             if (aabbs[iA]->collider->GetBody()->
                                                        42
                                                                     v[2] = glm::vec3(aabbL->max.x, aabbL
                 GetGroup() != 0)
                                                                         ->max.y, aabbL->min.z);
10
             {
                                                        43
                                                                     v[3] = glm::vec3(aabbL->min.x, aabbL
                      if (aabbs[iA]->collider->
                                                                         ->max.y, aabbL->min.z);
11
                          GetBody()->GetGroup() ==
                                                                     v[4] = glm::vec3(aabbL->min.x, aabbL
                          aabbs[iB]->collider->
                                                                         ->max.y, aabbL->max.z);
                          GetBody()->GetGroup())
                                                                     v[5] = glm::vec3(aabbL->max.x, aabbL
                                                        45
12
                               continue;
                                                                         ->max.y, aabbL->max.z);
                                                                     v[6] = glm::vec3(aabbL->max.x, aabbL
13
             }
                                                        46
                                                                         ->min.y, aabbL->max.z);
14
             if (aabbs[iA]->collider->GetBody()->
15
                                                                     v[7] = glm::vec3(aabbL->min.x, aabbL
                 GetInvMass() == 0.0f && aabbs[iB
                                                                         ->min.y, aabbL->max.z);
                 ]->collider->GetBody()->
                                                        48
                 GetInvMass() == 0.0f)
                                                                     for (int i = 0; i < 8; i++)
                                                        49
16
                      continue;
                                                        50
                                                                     {
                                                                              v[i] = aabbL->collider->
17
                                                        51
18
             if (Overlap(aabbs[iA], aabbs[iB]))
                                                                                  GetBody()->
                      colliderPairs.push_back(std::
                                                                                  LocalToGlobalPoint(v[i]);
19
                          make_pair(aabbs[iA]->
                                                        52
                                                                     }
                          collider, aabbs[iB]->
                                                        53
                          collider));
                                                                     AABB* aabb = aabbs[i];
                                                        54
20
        }
                                                        55
                                                                     aabb->min = glm::vec3(FLT_MAX);
21
   }
                                                        56
                                                                     aabb->max = glm::vec3(-FLT_MAX);
22
                                                        57
                                                                     for (int i = 0; i < 8; i++)
23
        return colliderPairs;
                                                        58
                                                                     {
                                                                              aabb->min.x = glm::min(aabb->
24
   }
                                                        59
25
                                                                                  min.x, v[i].x);
26
   void BroadPhase::Update()
                                                        60
                                                                              aabb->min.y = glm::min(aabb->
27
    {
                                                                                  min.y, v[i].y);
28
        glm::vec3 v[8];
                                                                              aabb->min.z = glm::min(aabb->
                                                        61
        for (int i = 0; i < aabbsL.size(); i++)</pre>
29
                                                                                  min.z, v[i].z);
30
        {
                                                        62
                                                                              aabb->max.x = glm::max(aabb->
                                                                                  max.x, v[i].x);
31
             AABB* aabbL = aabbsL[i];
32
                                                        63
                                                                              aabb->max.y = glm::max(aabb->
             if (aabbL->collider->GetShape() ==
33
                                                                                  max.y, v[i].y);
                 Collider::Sphere)
                                                                              aabb->max.z = glm::max(aabb->
                                                        64
34
                                                                                  max.z, v[i].z);
                      aabbs[i]->min = aabbL->min +
35
                                                        65
                                                                     }
                          aabbs[i]->collider->
                                                                }
                          GetBody()->GetPosition();
                                                           }
                                                       67
36
                      aabbs[i]->max = aabbL->max +
                                                            2.2.2 Narrow-Phase.
                          aabbs[i]->collider->
                                                            A geometry is said to be convex if any two points of the line within
                          GetBody()->GetPosition();
                                                            the geometry must fall within the geometry. Almost all collision
37
                      continue;
                                                            algorithms are based on convex geometry by default. For a concave
38
             }
                                                            geometry, a convex geometry can be generated using the QuickHull
39
                                                            algorithm, or the concave geometry can be decomposed into several
```

convex geometries using similar algorithms such as V-HACD. In

incidentFace =

general, convex geometry can satisfy most of our needs. For oddly 32 shaped objects, it is possible to approximate them with some regular collision shapes.

In the physics engine will use a variety of collision body shape, some 33 shapes of the collision calculation is very intuitive and simple, such as between two spheres, determine whether the distance between the two centers of the circle is greater than the sum of the radius, you can directly calculate whether the collision. Separating Axis Theorem (SAT) principle: two convex polygons do not intersect when and only when there must be a straight line, the projection of  $^{36}$ two convex polygons on this line does not intersect. Alternatively, 37 it can be described as follows: if two convex polygons intersect, the projections on all lines are intersecting.

```
void DetectHullvsHull(std::vector<Manifold>&
       manifolds, HullCollider* A, HullCollider*
        B)
2
   {
3
        FaceQuery faceQueryA;
4
        if (!QueryFaceAxes(faceQueryA, A, B))
5
                 return;
        assert(faceQueryA.separation <= 0.0f);</pre>
6
7
        FaceOuery faceOueryB;
8
        if (!QueryFaceAxes(faceQueryB, B, A))
9
10
                 return:
        assert(faceQueryB.separation <= 0.0f);</pre>
11
12
13
        EdgeQuery edgeQuery;
14
        if (!QueryEdgeAxes(edgeQuery, A, B))
15
                 return;
16
        assert(edgeQuery.separation <= 0.0f);</pre>
17
        float maxFaceSep = faceQueryA.separation
18
            > faceQueryB.separation ? faceQueryA.
            separation : faceQueryB.separation;
19
        float epsilon = 0.05f;
20
        if (edgeQuery.separation > maxFaceSep +
21
            epsilon)
22
        {
23
            // edge contact
            CreateEdgeContact(manifolds, A, B,
24
                edgeQuery);
25
        }
        else
26
2.7
        {
            // face contact
28
29
            int incidentFace;
30
            if (faceQueryA.separation >
                faceQueryB.separation + epsilon)
31
```

```
FindIncidentFace(B, A,
                        faceQueryA.faceIndex);
                    CreateFaceContact(manifolds,
                        B, A, incidentFace,
                        faceQueryA.faceIndex);
            }
            else
            {
                    incidentFace =
                        FindIncidentFace(A, B,
                        faceQueryB.faceIndex);
38
                    CreateFaceContact(manifolds,
                        A, B, incidentFace,
                        faceQueryB.faceIndex);
39
            }
40
       }
   }
41
42
   void DetectSphereWithSphere(std::vector<</pre>
43
       Manifold>& manifolds, SphereCollider* A,
       SphereCollider* B)
   {
44
45
        glm::vec3 CA = A->GetBody()->
           LocalToGlobalPoint(A->GetCentroid());
        glm::vec3 CB = B->GetBody()->
           LocalToGlobalPoint(B->GetCentroid());
47
        glm::vec3 normal = CB - CA;
48
49
        float dist2 = glm::12Norm(normal);
        float ra = A->GetRadius();
50
51
        float rb = B->GetRadius();
        float rSum = ra + rb;
52
53
54
        if (dist2 > rSum)
55
                return;
56
57
        normal = glm::normalize(normal);
58
59
60
        glm::vec3 PA = CA + A->GetRadius() *
           normal;
61
        glm::vec3 PB = CB - B->GetRadius() *
           normal;
        glm::vec3 C = (PA + PB) * 0.5f;
62
        float penetration = rSum - dist2;
63
64
65
       Manifold m;
       m.contacts.push_back(std::move(Contact(A
           ->GetBody(), B->GetBody(), C, normal,
             penetration));
```

```
4 · Name: Liu Yifei
    email: liuyf7@shanghaitech.edu.cn
    Name: Yan Yiheng
    email: yanyh1@shanghaitech.edu.cn
67
         manifolds.push_back(m);
                                                      102
                                                                   m.contacts.push_back(std::move(
                                                                       Contact(A->GetBody(), B->GetBody
68
    }
                                                                       (), contact, -normal, -
    /// <summary>
69
70
    /// Check whether A sphere is collided with a
                                                                       maxSeparation)));
         Hull(OK)
                                                      103
                                                                   manifolds.push_back(m);
71
    /// </summarv>
                                                      104
                                                               }
72.
    /// <param name="manifolds"></param>
                                                      105
                                                               else
    /// <param name="A"></param>
                                                      106
73
                                                               {
   /// <param name="B"></param>
                                                      107
                                                                   glm::vec3 normal(0);
    void DetectSphereWithHull(std::vector<</pre>
                                                      108
                                                                   glm::vec3 point(0);
        Manifold>& manifolds, SphereCollider* A,
                                                      109
                                                                   HalfSpace plane;
        HullCollider* B)
                                                                   float distance = 0.0f;
                                                      110
76
                                                                   float minDistance = FLT_MAX;
   {
                                                      111
77
         glm::vec3 center = A->GetBody()->
                                                                   int bestFace = -1;
                                                      112
             GetPosition();
                                     // sphere
                                                      113
                                                                   for (int i = 0; i < B->GetFaceCount()
             center
                                                      114
                                                                       ; i++)
78
79
         if (QueryPoint(B, center))
                                                      115
                                                                   {
80
         {
                                                      116
                                                                        normal = B->GetBody()->
             float maxSeparation = -FLT_MAX;
                                                                            LocalToGlobalVec(B->GetFace(i
81
82
             float separation = 0.0;
                                                                            )->normal);
             int bestFitFace = -1;
                                                                        point = B->GetBody()->
83
                                                      117
                 should be max separation
                                                                            LocalToGlobalPoint(B->GetFace
84
                                                                            (i)->edge->tail->position);
             for (int i = 0; i < B->GetFaceCount() 118
                                                                        plane = HalfSpace(normal, point);
85
                 ; i++)
                                                      119
                                                                        distance = plane.Distance(center)
86
             {
                                                                            ;
87
                 glm::vec3 normal = B->GetBody()->120
                     LocalToGlobalVec(B->GetFace(i 121
                                                                        if (distance > 0.0f && distance <</pre>
                                                                             minDistance)
                      )->normal);
                 glm::vec3 origin = B->GetBody()->122
88
                                                                        {
                     LocalToGlobalPoint(B->GetFace 123
                                                                                 minDistance = distance;
                      (i)->edge->tail->position);
                                                                                 bestFace = i;
                                                      124
89
                                                      125
                                                                        }
                 separation = glm::dot(origin -
                                                                   }
90
                                                      126
                     center, normal);
                                                      127
91
                 if (separation > maxSeparation)
                                                      128
                                                                   if (minDistance > A->GetRadius())
92
                 {
                                                      129
                                                                            return;
93
                          maxSeparation =
                                                      130
                              separation:
                                                      131
                                                                   // intersect with closest face plane
94
                          bestFitFace = i;
                                                      132
                                                                   glm::vec3 contact;
                                                      133
                                                                   float penetration;
95
                 }
96
             }
                                                      134
                                                                   normal = B->GetBody()->
97
                                                                       LocalToGlobalVec(B->GetFace(
98
             glm::vec3 normal = B->GetBody()->
                                                                       bestFace) -> normal);
                 LocalToGlobalVec(B->GetFace(
                                                      135
                                                                   contact = center - minDistance *
                 bestFitFace)->normal);
                                                                       normal;
99
             glm::vec3 contact = center -
                                                      136
                                                                   std::vector<glm::vec3> verts;
                 maxSeparation * normal;
                                                      137
                                                                   HEdge* e = B->GetFace(bestFace)->edge
100
                                                      138
101
             Manifold m;
                                                                       ;
```

```
139
             do {
                                                      171
                                                                       }
                      verts.push_back(B->GetBody() 172
                                                                       e = e->next;
140
                          ->LocalToGlobalPoint(e-> 173
                                                                   } while (e != B->GetFace(bestFace)->
                          tail->position));
                                                                       edge);
141
                      e = e->next;
                                                      174
                                                              }
142
             } while (e != B->GetFace(bestFace)-> 175
                                                          }
                 edge);
                                                      176
                                                          void DetectCollision(std::vector<Manifold>&
                                                              manifolds, Collider* A, Collider* B)
143
             if (QueryPoint(contact, verts, normal177
144
                                                      178
                                                                   if (A->GetShape() == Collider::Sphere
                 ))
145
             {
                                                                        && B->GetShape() == Collider::
                      penetration = A->GetRadius()
                                                                       Sphere)
146
                          - minDistance;
                                                      179
                                                                   {
                      Manifold m;
                                                                            DetectSphereWithSphere(
147
                                                      180
                      Contact c(A->GetBody(), B->
                                                                                manifolds, static_cast<</pre>
148
                          GetBody(), contact, -
                                                                                SphereCollider*>(A),
                          normal, penetration);
                                                                                static_cast <
                      m.contacts.push_back(c);
                                                                                SphereCollider*>(B));
149
150
                      manifolds.push_back(m);
                                                      181
                                                                   }
                                                                   else if (A->GetShape() == Collider::
151
                      return;
                                                      182
                                                                       Hull && B->GetShape() == Collider
152
             }
                                                                       ::Hull)
153
             return;
154
             // intersect with edges of closest
                                                      183
                                                                   {
                 face
                                                      184
                                                                            DetectHullvsHull(manifolds,
             e = B->GetFace(bestFace)->edge;
                                                                                static_cast<HullCollider</pre>
155
                                                                                *>(A), static_cast<
156
             glm::vec3 pA, pB;
157
             do {
                                                                                HullCollider*>(B));
158
                 pA = B->GetBody()->
                                                      185
                                                                   }
                     LocalToGlobalPoint(e->tail->
                                                                   else if (A->GetShape() == Collider::
                                                     186
                     position);
                                                                       Hull && B->GetShape() == Collider
                 pB = B->GetBody()->
                                                                       ::Sphere)
159
                     LocalToGlobalPoint(e->twin-> 187
                                                                   {
                                                                            DetectSphereWithHull(
                     tail->position);
                                                      188
                                                                                manifolds, static_cast<
160
                 if (IntersectSegmentSphere(pA, pB
                      , A, contact))
                                                                                SphereCollider*>(B),
161
                 {
                                                                                static_cast<HullCollider</pre>
162
                          normal = contact - center
                                                                                *>(A));
                                                      189
                                                                   }
                          float 1 = glm::length(
                                                                   else if (A->GetShape() == Collider::
163
                                                      190
                              normal):
                                                                       Sphere && B->GetShape() ==
                          normal /= 1;
                                                                       Collider::Hull)
164
                          penetration = A->
                                                      191
165
                                                                   {
                              GetRadius() - 1;
                                                      192
                                                                            DetectSphereWithHull(
                          Manifold m;
                                                                                manifolds, static_cast<</pre>
166
167
                          Contact c(A->GetBody(), B
                                                                                SphereCollider*>(A),
                                                                                static_cast<HullCollider</pre>
                              ->GetBody(), contact,
                               normal, penetration)
                                                                                *>(B));
                                                      193
                                                                   }
                                                                   else {
168
                          m.contacts.push_back(c); 194
                          manifolds.push_back(m);
169
                                                      195
                                                                            assert(false);
170
                                                                   }
                          return;
                                                      196
```

```
6 • Name: Liu Yifei
email: liuyf7@shanghaitech.edu.cn
Name: Yan Yiheng
email: yanyh1@shanghaitech.edu.cn
```

## 2.3 Constraints

197

. Constraints are very important in physics engines, just like shader is the core of achieving various rendering effects, Constraints are the core concept in physics engines. In some real-world physics constraint solving, such as robot motion, is very complex and time 18 consuming. In games, due to frame rate requirements, we do not 19 have enough time to get absolutely correct results, and we aim for a reasonably accurate and stable solution.

For example, if a box is placed on a fixed plane, we solve for the P point in the lower left corner of the box. The constraint of the contact constraint is that the contact points of the two objects do not move along the normal direction of the contact. This gives us an easy representation of the position constraint C. It is not enough to have only the position constraint, we derive C to obtain a formula for the velocity constraint:

$$\dot{C} = (\dot{\boldsymbol{p}}_B - \dot{\boldsymbol{p}}_A) \cdot \boldsymbol{n} + (\boldsymbol{p}_B - \boldsymbol{p}_A) \cdot \dot{\boldsymbol{n}}$$

Since we assume that the ground is fixed, we have  $\dot{p}_A = 0$ , and  $p_A$  and  $p_B$  are actually equal, so we simplify the equation to get:  $\dot{C} = \dot{p}_B \cdot n$ 

```
\dot{p}_{B} = v_{B} + \omega_{B} \times \mathbf{r}

\dot{C} = (v_{B} + \omega_{B} \times \mathbf{r}) \cdot \mathbf{n}

\dot{C} = \vec{n}(v'_{p_{a}} - v'_{p_{b}})

= \vec{n}(v_{a} - v_{b}) + \lambda_{n} \left(\frac{1}{m_{a}} + \frac{1}{m_{b}}\right) + \vec{n}(w_{a} \times r_{a} - w_{b} \times r_{b}) + 30

\lambda_{n} \left(\frac{\vec{n}(r_{a} \times \vec{n}) \times r_{a}}{I_{a}} + \frac{\vec{n}(r_{b} \times \vec{n}) \times r_{b}}{I_{b}}\right)

= \vec{n}(v_{p_{a}} - v_{p_{b}}) + \lambda_{n} \left(\frac{1}{m_{a}} + \frac{1}{m_{b}} + \frac{\vec{n}(r_{a} \times \vec{n}) \times r_{a}}{I_{a}} + \frac{\vec{n}(r_{b} \times \vec{n}) \times r_{b}}{I_{b}}\right)^{2}

= 0

\lambda_{n} = \frac{-\vec{n}(v_{p_{a}} - v_{p_{b}})}{\frac{1}{m_{a}} + \frac{1}{m_{b}} + \frac{\vec{n}(r_{a} \times \vec{n}) \times r_{a}}{I_{a}} + \frac{\vec{n}(r_{b} \times \vec{n}) \times r_{b}}{I_{b}}}

34

35
```

```
#define BAUMGARTE 0.2f
   #define VELOCITYSLOP 0.5f
   #define POSITIONSLOP 0.001f
4
   Contact::Contact(Body* A, Body* B, const glm
5
       ::vec3& position, const glm::vec3& normal
        , const float penetration)
            : A(A), B(B), position(position),
6
                normal(normal), penetration(
                penetration), impulseSumN(0.0f)
7
   {
        ComputeBasis(normal, tangent[0], tangent
8
           [1]);
9
        rA = position - A->GetCentroid();
10
11
        rB = position - B->GetCentroid();
```

```
12
13
        CalculateJacobian(JN, normal);
        CalculateBias();
14
15
        for (int i = 0; i < 2; i++)
       {
                impulseSumT[i] = 0.0f;
                CalculateJacobian(JT[i], tangent[
                kt[i] = CalculateEffectiveMass(JT
                    [i], A, B);
       }
   }
   //Normal constraint(OK)
   float Contact::CalculateNormalConstraint()
       const
25
   {
            glm::vec3 vA = A->GetVelocity() + glm
26
                ::cross(A->GetAngularVelocity(),
            glm::vec3 vB = B->GetVelocity() + glm
                ::cross(B->GetAngularVelocity(),
28
            return glm::dot((vB - vA), normal);
30 //3-Compute the Jacobian J for non-
       penetration and friction constraints(OK)
31 void Contact::CalculateJacobian(Jacobian& J,
       const glm::vec3& axis)
            J = Jacobian(-axis, -glm::cross(rA,
                axis), axis, glm::cross(rB, axis)
               );
34
35
   //Slop bias(OK)
   void Contact::CalculateBias()
37
38
            // restitution
            float e = (A->GetRestitution() + B->
39
                GetRestitution()) * 0.5f;
40
            float vSep =
                CalculateNormalConstraint();
41
            //bias = e * vSep;
42
            bias = e * (glm::min(vSep - 0.5f, 0.0)
                f));
43
   }
45
   void Contact::SolveVelocities(Velocity& vA,
       Velocity& vB)
46
   {
47
            if (CalculateNormalConstraint() >= 0)
```

```
void Contact::SolvePositions(Position& pA,
48
            {
                                                            Position& pB)
49
                     return;
            }
                                                        {
50
                                                     84
                                                                 float K = CalculateEffectiveMass(JN,
51
                                                     85
52
            float lambda, oldImpulse;
                                                                     A, B);
                                                                 float C = -BAUMGARTE * (glm::max(
53
            float uf = (A->GetFriction() + B->
                                                     86
                GetFriction()) * 0.5f;
                                                                     penetration - POSITIONSLOP, 0.0f)
            float maxFriction;
54
                                                                     ):
                                                                 float lambda = K > 0.0f ? -C / K :
55
                                                     87
            lambda = CalculateLagrangian(JN, vA,
                                                                     0.0f;
56
                vB, kn, bias);
                                                     88
                                                                 glm::vec3 P = lambda * normal;
57
                                                     89
            oldImpulse = impulseSumN;
                                                     90
                                                                 pA.c -= P * A->GetInvMass();
58
            impulseSumN = glm::max(0.0f,
                                                     91
                                                                 pA.q += 0.5f * glm::quat(0.0f, A->
59
                impulseSumN + lambda);
                                                                     GetInvInertia() * glm::cross(rA,
            lambda = impulseSumN - oldImpulse;
                                                                     -P)) * pA.q;
60
61
                                                     92
                                                                 pA.q = glm::normalize(pA.q);
            vA.v += lambda * JN.L1 * A->
                                                     93
                                                                 pB.c += P * B->GetInvMass();
62
                GetInvMass();
                                                                 pB.q += 0.5f * glm::quat(0.0f, B->
            vA.w += lambda * JN.A1 * A->
                                                                     GetInvInertia() * glm::cross(rB,
63
                GetInvInertia();
                                                                     P)) * pA.q;
            vB.v += lambda * JN.L2 * B->
                                                     95
                                                                 pB.q = glm::normalize(pB.q);
64
                GetInvMass():
                                                     96
                                                        }
65
            vB.w += lambda * JN.A2 * B->
                                                     97
                GetInvInertia();
                                                     98
                                                        void Manifold::SolveVelocities()
66
                                                     99
                                                        {
67
            for (int i = 0; i < 2; i++)
                                                    100
                                                             Body* A = contacts[0].A;
                                                    101
                                                             Body* B = contacts[0].B;
68
            {
                lambda = CalculateLagrangian(JT[i102
69
                    ], vA, vB, kt[i], 0.0f);
                                                             if (A->GetInvMass() != 0 && B->GetInvMass
                                                                 () != 0)
70
71
                oldImpulse = impulseSumT[i];
                                                    104
                                                            {
                maxFriction = uf * impulseSumN;
                                                                 if (!A->IsAwake() || !B->IsAwake())
72
                                                    105
                    //uf * kt[i] * 9.8f;
                                                    106
                                                                 {
                impulseSumT[i] = glm::clamp(
                                                    107
                                                                         B->SetAwake(true);
73
                    impulseSumT[i] + lambda, -
                                                    108
                                                                         A->SetAwake(true);
                    maxFriction, maxFriction);
                                                    109
                                                                 }
74
                lambda = impulseSumT[i] -
                                                    110
                                                            }
                    oldImpulse;
                                                    111
75
                                                    112
                                                             glm::vec3 v1 = A->GetVelocity();
                vA.v += lambda * JT[i].L1 * A->
                                                             glm::vec3 w1 = A->GetAngularVelocity();
76
                                                    113
                    GetInvMass();
                                                    114
                                                             glm::vec3 v2 = B->GetVelocity();
77
                vA.w += lambda * JT[i].A1 * A->
                                                    115
                                                             glm::vec3 w2 = B->GetAngularVelocity();
                    GetInvInertia();
                                                             Velocity vA(v1, w1);
                                                    116
78
                vB.v += lambda * JT[i].L2 * B->
                                                    117
                                                             Velocity vB(v2, w2);
                    GetInvMass();
                                                    118
79
                vB.w += lambda * JT[i].A2 * B->
                                                    119
                                                             for (int i = 0; i < contacts.size(); i++)
                    GetInvInertia();
                                                    120
                                                    121
                                                                 contacts[i].SolveVelocities(vA, vB);
80
            }
                                                    122
81
   }
                                                             }
82
                                                    123
```

```
8 • Name: Liu Yifei
    email: liuyf7@shanghaitech.edu.cn
    Name: Yan Yiheng
    email: yanyh1@shanghaitech.edu.cn
A->SetVelocity(vA.v);
124
125
         A->SetAngularVelocity(vA.w);
         B->SetVelocity(vB.v);
126
127
         B->SetAngularVelocity(vB.w);
128
129
130
    void Manifold::SolvePositions()
131
         Body* A = contacts[0].A;
132
133
         Body* B = contacts[0].B;
134
         glm::vec3 c1 = A->GetCentroid();
135
         glm::quat q1 = A->GetOrientation();
136
         glm::vec3 c2 = B->GetCentroid();
137
         glm::quat q2 = B->GetOrientation();
138
         Position pA(c1, q1);
139
         Position pB(c2, q2);
140
         for (int i = 0; i < contacts.size(); i++)</pre>
141
142
         {
                   contacts[i].SolvePositions(pA, pB
143
                       );
         }
144
145
146
         A->SetCentroid(pA.c);
147
         A->SetOrientation(pA.q);
         B->SetCentroid(pB.c);
148
149
         B->SetOrientation(pB.q);
150
```

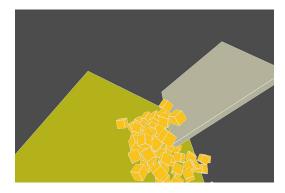


Fig. 4. Massive Boxes simulation

### 3.1 Division of labor

In this project, Liu implemented the physical mechanics simulation system and wrote the report, while Yan implemented the basic environment, including camera, geometry, etc., and created the PPT.

## 3 RESULTS

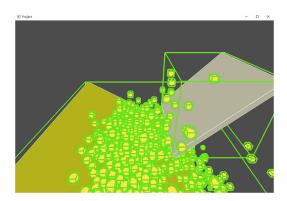


Fig. 3. Massive Balls simulation

, Vol. 1, No. 1, Article . Publication date: January 2023.