2D Rigidbody and Joints

Physic Simulation

2020/4/1

Outline

- Demo
- Project overview
- Scoring criteria
- Objective and explanation
- Submission
- Hint and Reminder
- Q&A

Demo

Project Overview

- Project folder (Rigidbody2D)
  - include (header files)
  - src (source code)
  - vendor (third-party dependencies)
- Environment
  - IDE：Visual studio 2017 / 2019
- Find the “TODO” tags and complete the functions in src folder.

Scoring Criteria

- Collision detection - 20%
  - AABB to AABB - 5%
  - Circle to circle - 5%
  - Circle to AABB - 10%
- Resolve impulse - 20%
  - Impulse - 15%
  - Friction Calculation - 5%
- Joint constraint - 20%
  - Spring joint - 10%
  - Distance joint - 10%

Scoring Criteria (cont’d)

- Intergrator - 20%
  - Explicit Euler - 5%
  - Runge Kutta 4th - 15%
- Report - 20%

Objective & Explanation

Collision Detection

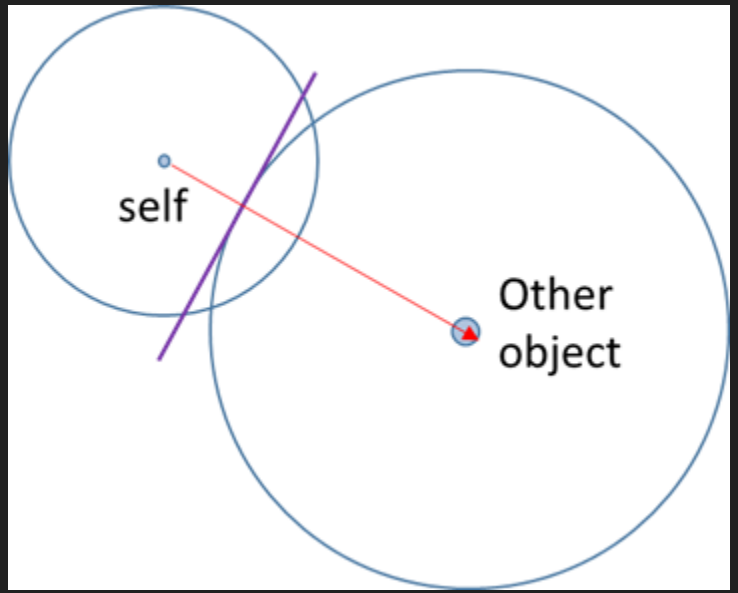
- In this homework, there are two types of objects
  - AABB (Axis Aligned Bounding Box)
  - Circle
- Compute the relationship between two objects
  - Manifold AABB::visitAABB(· · ·)
  - Manifold Circle::visitCircle(· · ·)
  - Manifold CollisionHelper::GenerateManifold(· · ·)

Collision Detection (cont'd)

- Store collision informations into Manifold
  - The normal should be pointing from self to another object and normalized
- Remember to deal with overlapping
  - Overlap happens when two objects collide with each other
  - Compute penetration along normal, that is the overlap part

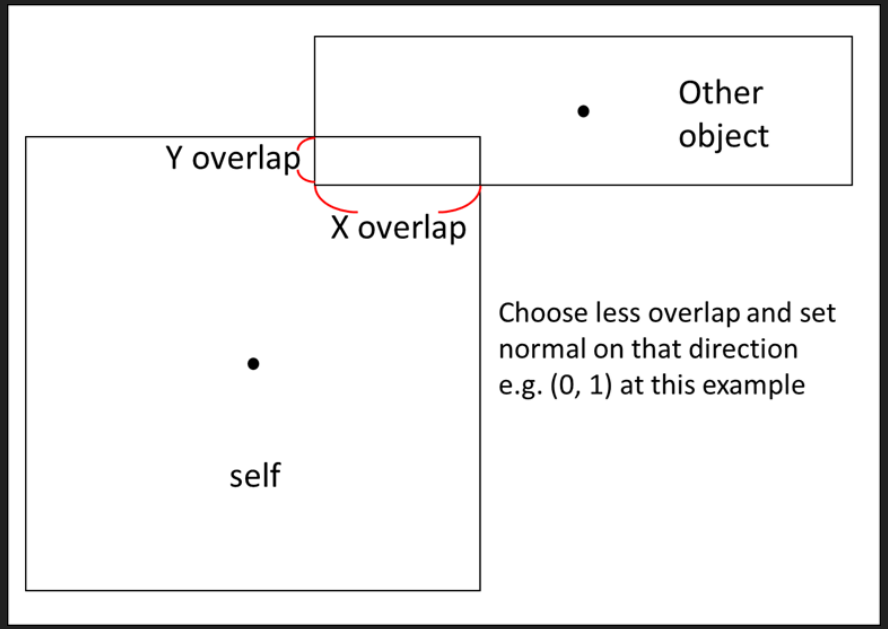
Collision Detection (cont'd)

- Compute collision normal
  - Circle to circle



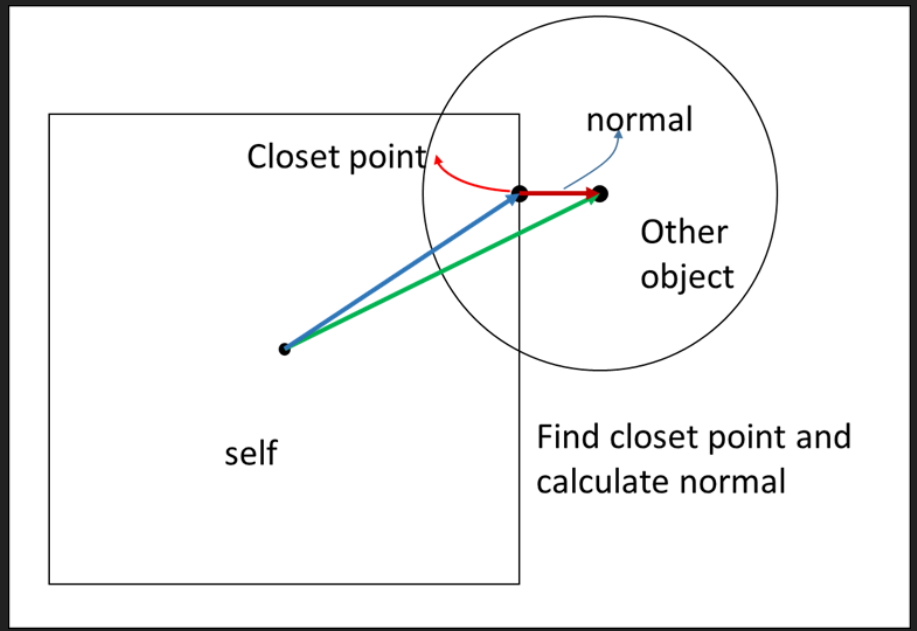
Collision Detection (cont'd)

- Compute collision normal
  - AABB to AABB



Collision Detection (cont'd)

- Compute collision normal
  - AABB to circle



Resolve Impulse

- Compute the force between two objects if collision happens
  - void Manifold::Resolve()
  - Compute impulse
  - Then compute friction
    - The friction factor at the contact point are the square root of two objects' friction factors
    - Both static and dynamic friction are based on the given formula

Resolve Impulse (cont'd)

- Resolve impulse
  - Compute relative velocity along normal
  - Compute reflection vector according to the ratio of their masses
    - "e" is the minimum value of restitution between two objects

$$V_a = V_a - (1+e)*N*((V_b-V_a) \cdot N)*(M_b / (M_a+M_b))$$

$$V_b = V_b - (1+e)*-N*((V_b-V_a) \cdot -N)*(M_a / (M_a+M_b))$$

Resolve Impulse (cont'd)

- Some tricks

$$(M_a+M_b)/(M_a*M_b) = 1/M_a + 1/M_b$$

$$I = (1+e)*N*(V_r \cdot N) / (1/M_a + 1/M_b)$$

$$V_a -= I * 1/M_a$$
$$V_b += I * 1/M_b$$

Resolve Impulse (cont’d)

- Compute friction along the tangent direction
  - Compute velocity change and apply as before
  - Follow the law of maximum static friction and dynamic friction

Joint Constraint

- In this homework, there are two types of joints
  - Spring joint
  - Distance joint
- Resolve the constraint between two bodies connected by the joint
  - void SpringJoint::ApplyConstraint()
  - void DistanceJoint::ApplyConstraint()

Joint Constraint (cont’d)

- Spring Joint
  - The connection between two bodies is spring
  - Review “particle.ppt” from p.9 to p.13
  - Damper coeff is given as (1 / 30) * m_stiffness

Joint Constraint (cont'd)

- Distance Joint
  - The distance between two joints must be smaller than a given length
  - Apply velocity to stop the joint from extending if the distance is greater than the given length
  - The maximum length of joint is given as m_restLength
- Reference
  - https://reurl.cc/oLeMLq

Integrator

- In this homework, there are two types of integrator
  - Explicit Euler
    - Review “ODE_basic.ptt” from p.15 to p16
  - Runge Kutta 4th
    - Review “ODE_basic.ptt” p.21
    - Or “Physically Based Modeling” p.B5- p.B6
  - You should consider gravity = (-9.8 m/s^2) on Y direction in this system

Integrator (cont'd)

- Runge Kutta 4th
  - Refer to the format below as an example of storing essential data for RK4

```
// this stores the absolute value of position and velocity
std::vector<StateStep> currentState(_bodies.size());
// below four arrays store the delta value of each state
std::vector<StateStep> deltaK1State(_bodies.size());
std::vector<StateStep> deltaK2State(_bodies.size());
std::vector<StateStep> deltaK3State(_bodies.size());
std::vector<StateStep> deltaK4State(_bodies.size());
```

Suggested Outline of Report

- Introduction/Motivation
- Fundamentals
- Implementation
- Result and Discussion
    - the difference between Explicit Euler and RK4
- Conclusion

Submission

- Compress all the materials into a zip file
  - Naming rule：CA1_StudentID_Version
    - e.g. CA1_0516221_v001.zip
- Your zip file should contain
  - Source code (ensure your project can be built successfully)
  - Report in pdf or MS word format, no more than 10 pages
- Upload all your materials to New E3
  - The latest version will be your final submission

Late and Cheating Penalty

- Late penalty
  - Penalty of 10 points of the value of the assignment/day
- Cheating policies
  - 0 points for any cheating on assignments
  - Allowing another student to examine your code is also considered as cheating
  - DONOT upload your project to any Git hosting service before this semester

Deadline

- 2020/4/21 23:55

Hint and Reminder

- Run in Release mode for better performance, in Debug mode the system will be very laggy
- You can change the deltatime at main.cpp to test integrator
  - When deltatime is big, RK4 is more stable than Euler

Hint and Reminder (cont'd)

- Please fill your student ID at the top of main.cpp
- How to contact TAs?
  - Please send email via new E3
  - IMPORTANT：please sort out and arrange your questions, so we can help you without wasting time on trivial matters

Q & A