Institut für Informatik — Technische Universität München SS 2016
Scientific **C**omputing in **C**omputer **S**cience
S. Rettenberger, R. Wittmann

Bachelor Lab Scientific Computing (Game Physics) Worksheet 2: Collisions (sphere-sphere, sphere-plane) & Interpenetration

On the present worksheet, we will implement collision detections for sphere-sphere and sphere-plane collisions and introduce a simple collision resolution method. For testing your implementations, you can refer to the scenes 1 and 9 of your physics engine.

Assignment 1: Sphere-sphere collision detection

File: *cPhysicsIntersections.cpp*,
Method: *CPhysicsIntersections::sphereSphere*

This collision test is one of the easiest ones. Use the radii of the 2 spheres and their midpoints to compute whether there is an intersection or not. You also need to compute the collision normal and the interpenetration depth (see presentation slides). For further purposes, it is important to set all values in the class of type CPhysicsCollisionData given as a reference to the collision detection function. Return 'true' in case of an intersection:

- physics_object1 // reference to first colliding object
- physics_object2 // reference to second colliding object
- collision_point1 // collision point at first object (in the world space)
- collision_point2 // collision point at second object (in the world space)
- collision_normal // collision normal
- interpenetration_depth // the overall interpenetration depth of both objects

Do not forget that you can use the *debug mode* in the SBNDEngine and the output to the console to test and validate successful collisions. Scene 9 has 4 spheres with one of them fixed in space which allows you to test the correct interpenetration.

Also insert an appropriate call to the method *getCollisions()* which iterates over all possible colliding objects and stores the collision information.

Assignment 2: Sphere-plane collision detection

File: *cPhysicsIntersections.cpp*,
Method: *CPhysicsIntersections::spherePlane*
There are many ways to implement a collision detection. One possible way is to transfer the sphere to the plane space with the x- and z-coordinate aligned at the planes axes. Then the y-coordinate and the radius of the sphere can be used to run an early "non-collision test". Further simple collision tests have to be implemented which take different cases into account as described in the presentation slides. Since infinite planes do not exist in reality, we have to handle interpenetrations at the plane edges appropriately. For this case, compute the collision normal according to the slides of the last presentation! A further problem is posed by spheres that are *fully placed on the outer side* of the plane, that is spheres which "overhang" the computational domain.

Implement the collision tests between a sphere and a plane according to the descriptions and distinction of cases from above! Also fill in the collision data (see assignment 1) and find a solution to the "Overhang"-problem.

Assignment 3: Interpenetration resolution

File: *cPhysicsEngine_Private.cpp*,
Method: *cPhysicsEngine_Private::resolveInterpenetrations*

Resolving interpenetration

Implement appropriate methods to resolve interpenetrations between colliding objects as described on the presentation slides during the meeting. Include an assertion to check whether the collision is resolved.

Since the objects cannot be accelerated yet by possible collisions (see next worksheet), the spheres in scene 1 should stop on the plane for a few frames. Starting the simulation with scene 9, one sphere should stop on the sphere with infinite mass.

After having stopped, the spheres in both scenes soon start jittering and finally disappear after approx. 20 seconds. Why does this happen? How can you account for that?

Iterative solving

Resolving interpenetrations only once per timestep can lead to further interpenetrations since we only consider binary collisions at each timestep. Therefore, the collision detection as well as the solving of the interpenetration has to be called several times. Implement a respective solution into the timestep methods. You can use the member-variable *max_global_collision_solving_iterations* as an upper limit for the iterative solver steps.

Be prepared to answer questions about this non-binary collision problem!

Submission

Please submit a **compileable** snapshot of your (team-)work to Moodle along with a visual proof (e.g. set of pictures, short length video) to demonstrate the progress of your work.

Good luck,

Roland & Sebastian