DigiPen
INSTITUTE OF TECHNOLOGY

Lecture 7 | Static Collision

CS230
Game
Implementation
Techniques

1. ASTEROIDS – Collision

Collision is the interaction between solid objects in the real word. In computer simulations like 3D graphics and video games, these boundary surfaces do not really exist, so objects are free to move through one another. In simulations, you must define where any object needs to *simulate* collision.

In 2D games, when we want to check if 2 sprites collide, we check if their "collision data" are intersecting. Rectangle and Circle are 2 of the most commonly used collision data. A rectangle is good enough for most sprites, but sometimes a circle is used for circular game objects like bullets or wheels.

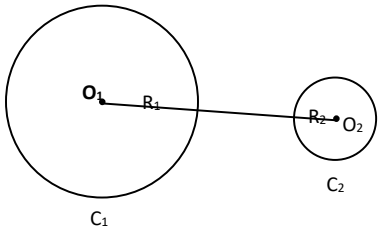
1.1. Point-Circle Collision

- Point-Circle collision can be used for small object, where it would be enough to use the object's corners as collision data.
- To check for collision between a circle and point, we must see if that point is:
  - Inside the circle
  - Outside the circle
  - On the perimeter of the circle
- In order to do this, we must compare the distance separating the circle's center the point with the circle's radius.
- Example: The circle's center is C, its radius is R and the point is P
  - |CP| < R: The point is inside the circle -> Collision
  - |CP| > R: The point is outside the circle -> No collision
  - |CP| = R: The point lies on the perimeter of the circle. The circle and the point can be considered as either colliding or non-colliding, but it's the programmer responsibility to be **consistent.**

1.2. Circle-Circle Collision

- Using a circle as collision data is great for circular game objects like a car's wheels for example.
- It can be very inaccurate in some situations, especially when an object has a relatively large size in a direction and a much smaller one in another direction.
- When we want to check collision among 2 game objects having circles as their collision data, we should check if these 2 circles overlap.
- 5 cases arise when comparing 2 circles (C_1 & C_2):

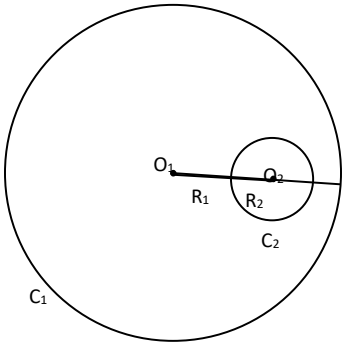
    - Case 1: Exterior circles
      This case arises when C_1 & C_2 don't intersect and none of them contains the other.



As you can see in the figure above, the distance separating the circles' centers is greater than the sum of the radii.

$$\|\overrightarrow{o_1 o_2}\| > R_1 + R_2$$

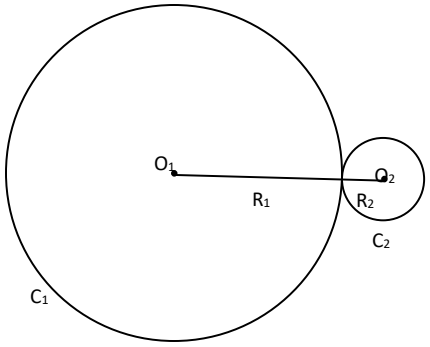
    - Case 2: Interior circles
      This case arises when C_1 & C_2 don't intersect but one of them is contained inside the other.



Looking at the figure above, it's clear that the distance separating the circles' centers is less than the absolute value of the difference between the radii.

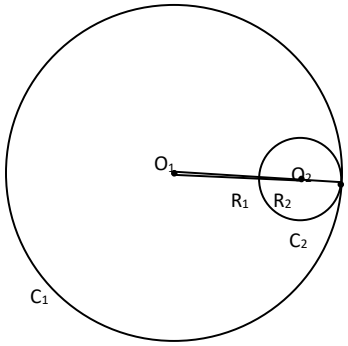
$$\left\|\overrightarrow{O_1 O_2}\right\| < |R_1 - R_2|$$

- Case 3: Externally tangent circles
  This case arises when C1 & C_2 intersect in one point but none of them contains the other.



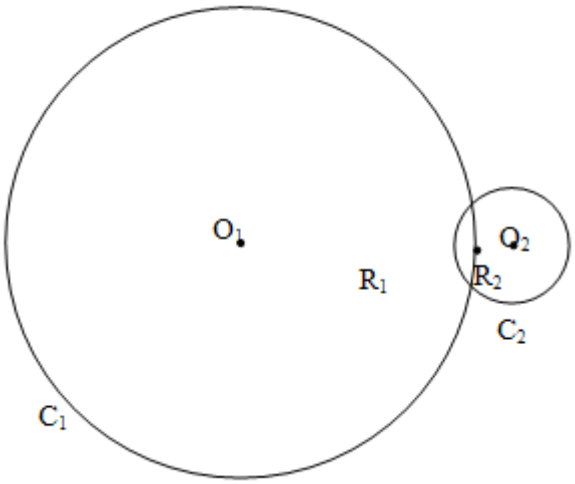
  Here we have the distance separating the circles' center equal to the sum of the radii.
  $\left\|\overrightarrow{O_1O_2}\right\| = R_1 + R_2$

- Case 4: Internally tangent circles
  C_1 & C_2 are internally tangent if they intersect in one point and one of them contains the other.



  This occurs when the distance separating the circles' centers is equal to the absolute value of the difference between the 2 radii.

$$\left\|\overrightarrow{O_1O_2}\right\| = |R_1 - R_2|$$



- Case 5: Secant Circles
  This case arises when C_1 & C_2 intersect in two points.

  This situation occurs when distance separating the 2 centers is:
  - Greater than the absolute value of the difference between the 2 radii
  - Less than the sum of the circles' radii

$$|R_1 - R_2| < \left\|\overrightarrow{O_1O_2}\right\| < R_1 + R_2$$

- Usually in games we don't need this much information about the 2 circles in order to determine if a collision occurred.

- Case 2 to 5 can be considered as colliding circles, which leaves case 1 as the only non-colliding option.

- Algorithm:
  Point centerA, centerB
  float radiusA, radiusB

  RadiusSum = radiusA + radius
  CentersDistance = Length(centerA, centerB)

  if(CentersDistance <= RadiusSum)
  Collision
  else
  No Collision

- **Optimization:**
  - In all circle-circle and point-circle collision checks, we need to calculate and compare the distance between the circles' centers and the sum (or difference) of the radii.
  - Computing the distance requires a square root operation, which is expensive computation wise.
  - To avoid this operation:
    - Point-Circle: Compare the square of the distance separating the circle's center and the point with the square of the radius
    - Circle-Circle: Compare the square of the distance between the 2 centers with the square of the radii's sum/difference.

1.3. Point-Rectangle Collision

- Point collision can be used for small objects or for particular objects where it is enough to test objects' corners.
- A rectangle is defined as 4 values: top, bottom, left and right.
- A point outside the rectangle is considered as non-colliding.
- A point inside the rectangle is considered as colliding.
- A point on the perimeter of the rectangle can be considered as either colliding or non-colliding. It is up to the programmer to decide which way to choose
- Algorithm:
  Point P;
  float left, right, top, bottom;

  if(P.X < left) then no collision
  if(P.X > right) then no collision
  if(P.Y < bottom) then no collision
  if(P.Y > top) then no collision

  If all the above conditions fail, then the point lies within the rectangle.

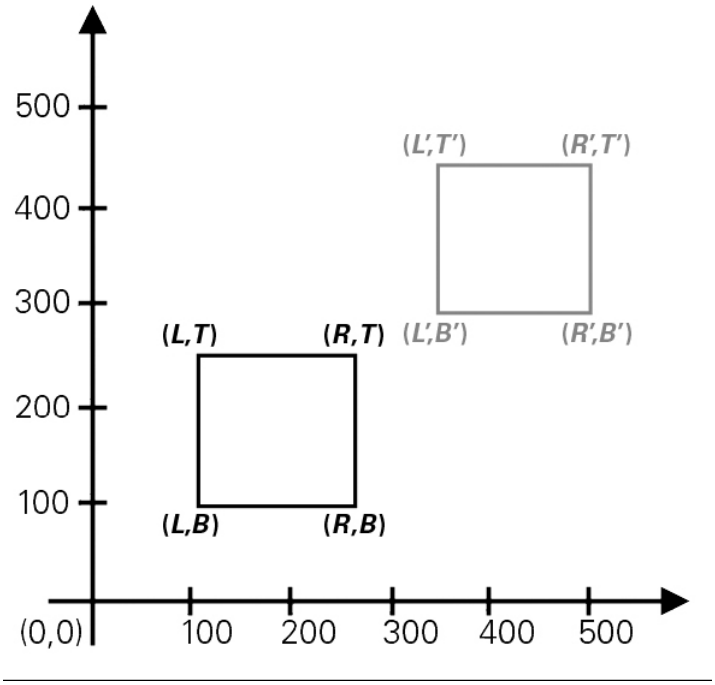
1.4. Rectangle-Rectangle Collision

- Most 2D games use a bounding rectangle around the sprite.
- This rectangle should be as small as possible, but it must still contain the actual game object (which can be of any shape inside the rectangular image).
- A bounding rectangle is defined using 4 values: top, bottom, left and right.
- When we want to check if 2 sprites collide, we should check if there 2 bounding rectangles overlap each other.
- Algorithm:
  float leftA, leftB
  float rightA, rightB
  float topA, topB
  float bottomA, bottom

  if(leftA > rightB) then no collision
  if(leftB > rightA) then no collision
  if(topA < bottomB) then no collision
  if(topB < bottomA) then no collision

  If all the above conditions fail, then the 2 rectangles overlap, and a collision between the 2 sprites occurred

- Example: No collision



- Example: Collision