Moving Circle vs Static Line – Pseudo Code

Introduction:

LNS is a line segment with end points **P0** and **P1** and outward normal $\hat{\boldsymbol{N}}$.

Circle is centered by **B** and has radius **R**. It is moving with velocity $\vec{V}$ per one frame.

$\mathbf{B_s}$ is the starting position of **B**, $\mathbf{B_e}$ is the end position of **B**, $\mathbf{B_i}$ is the intersection position of **B** (if any collision).

Problem:

Detect the collision time, the collision position of Circle, and the reflected position after bouncing of LNS.

Solution:

The following is a pseudo-code.

We need to distinguish between 2 cases:

1 - The Circle might hit the body of LNS first (2 sub-cases = 2 sides (normal's side and opposite normal's side))

2 - The circle might hit one of the edges (end points) of LNS.

MovingCircleVsStaticLine()

{ //**N** is normalized

if($\hat{N}$.**B**$_s$ - $\hat{N}$.**P0** <= -**R**) //Here we consider we have an imaginary line LNS1, distant by –**R** (opposite $\hat{N}$ direction)

//Check if the velocity vector $\vec{V}$ is within the end points of LNS1

//$\vec{M}$ is the outward normal to Velocity $\vec{V}$. Compute P0' and P1'

P0' = **P0** – **R***$\hat{N}$ and **P1'** = **P1** – **R***$\hat{N}$ //To simulate LNS1 line edge points

if($\vec{M}$.**B**$_s$**P0'** * $\vec{M}$.**B**$_s$**P1'** < 0)

T$_i$ = ($\hat{N}$.**P0** - $\hat{N}$.**B**$_s$ - **R**) / ($\hat{N}$.$\vec{V}$) //We are sure $\hat{N}$.$\vec{V}$!= 0

if(0 <= **T**$_i$ <= 1)

B$_i$ = **B**$_s$ + $\vec{V}$*(**T**$_i$)

B'$_e$ = **ApplyReflection**(-$\hat{N}$, **B**$_i$**B**$_e$) //Normal of reflection is -$\hat{N}$

else

CheckMovingCircleToLineEdge(false)

else if($\hat{N}$.**B**$_s$ - $\hat{N}$.**P0** >= **R**) //Here we consider we have an imaginary line LNS2 distant by +**R** (Same $\hat{N}$ direction)

//Check if the velocity vector $\vec{V}$ is within the end points of LNS2

//$\vec{M}$ is the outward normal to Velocity $\vec{V}$. Compute P0' and P1'

P0' = **P0** + **R***$\hat{N}$ and **P1'** = **P1** + **R***$\hat{N}$ //To simulate LNS2 line edge points

if($\vec{M}$.**B**$_s$**P0'** * $\vec{M}$.**B**$_s$**P1'** < 0)

T$_i$ = ($\hat{N}$.**P0** - $\hat{N}$.**B**$_s$ + R) / ($\hat{N}$.$\vec{V}$) //We are sure $\hat{N}$.$\vec{V}$!= 0

if(0 <= **T**$_i$ <= 1)

B$_i$ = **B**$_s$ + $\vec{V}$*(**T**$_i$)

B'$_e$ = **ApplyReflection**($\hat{N}$, **B**$_i$**B**$_e$) //Normal of reflection is $\hat{N}$

else

CheckMovingCircleToLineEdge(false)

else //The circle's starting position **B**$_s$, is between both lines LNS1 and LNS2.

CheckMovingCircleToLineEdge(true)

}

CheckMovingCircleToLineEdge(bool withinBothLines)

```
{
    if(withinBothLines) //When it's true, is to say that Bs is starting from between both imaginary lines
        //Check which edge may collide first?
        if(BsP0.P0P1 > 0) //P0 side
            if(m = BsP0.V̂ > 0) //Otherwise no collision
                //Reaching here means the circle movement is facing P0
                //M̂ is normalized outward normal of V⃗
                float dist0 = BsP0.M̂  //Same as P0.M̂ - Bs.M̂ (Shortest distance from P0 to V⃗)
                if(abs(dist0) > R)
                    return no collision
                //Reaching here means the circle movement is going towards P0
                //The next line assumes the circle at collision time with P0
                Compute: H = sqrt(R*R - dist0*dist0)
                float ti = (m – H) / V.Length();
                if(ti <= 1)
                    Bi = Bs + V⃗*ti
                    //Normal of reflection is P0Bi normalized
                    B'e = ApplyReflection(P0Bi, BiBe)
        else //(BsP1.P0P1 < 0) //P1 side
            if(m = BsP1.V̂ > 0) //Otherwise no collision
                //Reaching here means the circle movement is facing P1
                //M̂ is normalized outward normal of V⃗
                float dist1 = BsP1.M̂    //Same as P1.M̂ - Bs.M̂
                if(abs(dist1) > R)
                    return no collision
                //Reaching here means the circle movement is going towards P1
```

```
                //The next line assumes the circle at collision time with P1
                Compute: H = sqrt(R*R – dist1*dist1)

                float t_i = (m – H) / V.Length();

                if(t_i <= 1)

                        B_i = B_s + V⃗*t_i

                        //Normal of reflection is P1B_i normalized

                        B'_e = ApplyReflection(P1B_i, B_iB_e)

else //else of: if(withinBothLines)

        //Check which line edge, P0 or P1, is closer to the velocity vector V⃗?

        bool P0Side = false

        float dist0 = B_sP0.M̂//Same as P0.M̂ - B_s.M̂  (M̂ is normalized outward normal of V⃗)

        float dist1 = B_sP1.M̂//Same as P1.M̂ - B_s.M̂


        float dist0_absoluteValue = abs(dist0)

        float dist1_absoluteValue = abs(dist1)

        if(dist0_absoluteValue > R) && (dist1_absoluteValue > R)

                return No Collision

        else if(dist0_absoluteValue <= R) && (dist1_absoluteValue <= R)

                float m0 = BsP0.V̂

                float m1 = BsP1.V̂

                float m0_absoluteValue = abs(m0)

                float m1_absoluteValue = abs(m1)

                if(m0_absoluteValue < m1_absoluteValue)

                        P0Side = true

                else

                        P0Side = false

        else if(dist0_absoluteValue <= R)

                P0Side = true
```

```
else //if(dist1_absoluteValue <= R)
    P0Side = false

if(P0Side) //circle is closer to P0
    if(m = BsP0.V̂ < 0)
        return No Collision //moving away
    else
        //Reaching here means the circle movement is going towards P0
        //The next line assumes the circle at collision time with P0
        Compute H: = sqrt(R*R - dist0*dist0)
        float ti = (m – H) / V.Length();
        if(ti <= 1)
            Bi = Bs + V⃗*ti
            //Normal of reflection is P0Bi normalized
            B'e = ApplyReflection(P0Bi, BiBe)
else // circle is closer to P1
    if(m = BsP1.V̂< 0)
        return No Collision //moving away
    else
        //Reaching here means the circle movement is going towards P1
        //The next line assumes the circle at collision time with P1
        Compute H: = sqrt(R*R – dist1*dist1)
        float ti = (m – H) / V.Length();
        if(ti <= 1)
            Bi = Bs + V⃗*ti
            //Normal of reflection is P1Bi normalized
            B'e = ApplyReflection(P1Bi, BiBe)
}
```

Point2D ApplyReflection(Vector2D normal, Vector2D penetration)

```
{
    return Bi + penetration - 2(penetration . normal) * normal;
}
```