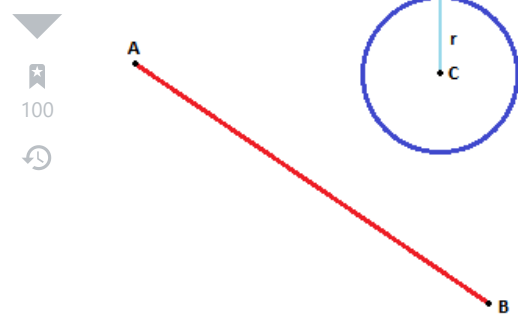


# Circle line-segment collision detection algorithm?

Asked 13 years, 1 month ago Modified 5 months ago Viewed 172k times

I have a line from A to B and a circle positioned at C with the radius R.

218



What is a good algorithm to use to check whether the line intersects the circle? And at what coordinate along the circles edge it occurred?

algorithm math line collision-detection geometry

Share Follow

edited Dec 6, 2014 at 23:51

smci  
30.3k ● 18 ● 110 ● 144

asked Jul 2, 2009 at 9:15

Mizipzor  
49.3k ● 22 ● 94 ● 138

4 Hmm. One question: are you talking about the infinite line through A and B, or the finite line segment from A to B? – Jason S Jul 2, 2009 at 19:07

2 In this case, its the finite line segment. Is "line" called something else depending on if its finite or infinite? – Mizipzor Jul 3, 2009 at 13:19

1 Is there a performance requirement ? Should it be a fast method ? – chmike Jul 7, 2009 at 6:32

At this point, no, all the algorithms here that Ive tried doesnt slow the application down noticeably. – Mizipzor Jul 7, 2009 at 10:53

15 @Mizipzor yes, they are called something else: line *segments*. If you just say "line" it's implied an infinite one. – MestreLion Aug 6, 2014 at 7:59

## 29 Answers

Sorted by: Highest score (default)

Trending sort available

Taking

223



1. **E** is the starting point of the ray,
2. **L** is the end point of the ray,
3. **C** is the center of sphere you're testing against
4. **r** is the radius of that sphere

Compute:

**d** = L - E ( Direction vector of ray, from start to end )

**f** = E - C ( Vector from center sphere to ray start )

Join Stack Overflow to find the best answer to your technical question, help others answer theirs.

Sign up with email

Sign up with Google

Sign up with GitHub

Sign up with Facebook



This is a parametric equation:

$$P_x = E_x + t d_x$$

$$P_y = E_y + t d_y$$

into

$$(x - h)^2 + (y - k)^2 = r^2$$

(h,k) = center of circle.

Note: We've simplified the problem to 2D here, the solution we get applies also in 3D

to get:

1. **Expand**  $x^2 - 2xh + h^2 + y^2 - 2yk + k^2 - r^2 = 0$
2. **Plug**  $x = e_x + t d_x$   
 $y = e_y + t d_y$   
 $(e_x + t d_x)^2 - 2(e_x + t d_x)h + h^2 + (e_y + t d_y)^2 - 2(e_y + t d_y)k + k^2 - r^2 = 0$
3. **Explode**  $e_x^2 + 2e_x t d_x + t^2 d_x^2 - 2e_x h - 2t d_x h + h^2 + e_y^2 + 2e_y t d_y + t^2 d_y^2 - 2e_y k - 2t d_y k + k^2 - r^2 = 0$
4. **Group**  $t^2(d_x^2 + d_y^2) + 2t(e_x d_x + e_y d_y - d_x h - d_y k) + e_x^2 + e_y^2 - 2e_x h - 2e_y k + h^2 + k^2 - r^2 = 0$
5. **Finally**,  $t^2(\mathbf{d} \cdot \mathbf{d}) + 2t(\mathbf{e} \cdot \mathbf{d} - \mathbf{d} \cdot \mathbf{c}) + \mathbf{e} \cdot \mathbf{e} - 2(\mathbf{e} \cdot \mathbf{c}) + \mathbf{c} \cdot \mathbf{c} - r^2 = 0$   
 Where  $\mathbf{d}$  is the vector  $\mathbf{d}$  and  $\cdot$  is the dot product.
6. **And then**,  $t^2(\mathbf{d} \cdot \mathbf{d}) + 2t(\mathbf{d} \cdot (\mathbf{e} - \mathbf{c})) + (\mathbf{e} - \mathbf{c}) \cdot (\mathbf{e} - \mathbf{c}) - r^2 = 0$
7. **Letting**  $\mathbf{f} = \mathbf{e} - \mathbf{c}$   $t^2(\mathbf{d} \cdot \mathbf{d}) + 2t(\mathbf{d} \cdot \mathbf{f}) + \mathbf{f} \cdot \mathbf{f} - r^2 = 0$

So we get:

$$t^2 * (\mathbf{d} \cdot \mathbf{d}) + 2t * (\mathbf{f} \cdot \mathbf{d}) + (\mathbf{f} \cdot \mathbf{f} - r^2) = 0$$

So solving the quadratic equation:

```
float a = d.Dot( d );
float b = 2*f.Dot( d );
float c = f.Dot( f ) - r*r ;

float discriminant = b*b-4*a*c;
if( discriminant < 0 )
{
    // no intersection
}
else
{
    // ray didn't totally miss sphere,
    // so there is a solution to
    // the equation.

    discriminant = sqrt( discriminant );

    // either solution may be on or off the ray so need to test both
    // t1 is always the smaller value, because BOTH discriminant and
    // a are nonnegative.
    float t1 = (-b - discriminant)/(2*a);
    float t2 = (-b + discriminant)/(2*a);

    // 3x HIT cases:
    //      -o->      --|--> |      |      --|-->
    // Impale(t1 hit,t2 hit), Poke(t1 hit,t2>1), ExitWound(t1<0, t2 hit),

    // 3x MISS cases:
    //      -> o      o ->      | -> |
    // FallShort (t1>1,t2>1), Past (t1<0,t2<0), CompletelyInside(t1<0, t2>1)

    if( t1 >= 0 && t1 <= 1 )
```

Join Stack Overflow to find the best answer to your technical question, help others answer theirs.

Sign up with email

Sign up with Google

Sign up with GitHub

Sign up with Facebook

X

Share Follow

edited Nov 30, 2020 at 21:59

answered Jul 5, 2009 at 21:54



anatolyg

25.2k ● 8 ● 55 ● 123



bobobobo

62.3k ● 58 ● 250 ● 349

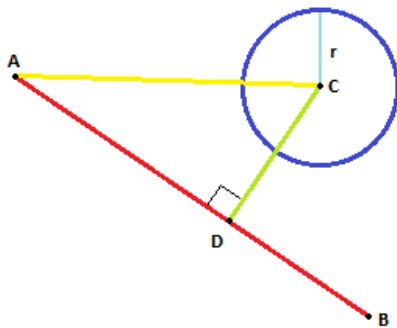
- 1 Seems to work if I do straight copy and paste, but I'm looking to understand it to. In  $(x-h)^2 + (y-k)^2 = r^2$  what is  $h$  and  $k$ ? Is  $k$  to constant by which the line/ray increases on  $y$  over  $x$ ? And what is  $t$ ? Looking at the code it seems you have assumed its 1 (so its just "removed"). Do these formulas have a name or something? Maybe I can look them up in detail on Wolfram. – Mizipzor Jul 6, 2009 at 12:17
- 3  $h$  and  $k$  are the center of the circle that you're intersecting against.  $t$  is the parameter of the line equation. In the code,  $t_1$  and  $t_2$  are the solutions.  $t_1$  and  $t_2$  tell you "how far along the ray" the intersection happened. – bobobobo Jul 6, 2009 at 13:22
- 1 Ok, got it. The dot product is simply computed over the three elements  $(x,y,z)$  vectors. I will switch my code to this algorithm. – chmike Jul 10, 2009 at 6:23
- 24  $P = E + t * d$  What is  $t$ ? – Derek 朕會功夫 Jun 9, 2012 at 1:49
- 3 Not sure why, but the code doesn't seem to work for the Impale case. It does when I add if  $t_1 \leq 0 \ \&\& \ t_1 \geq -1 \ \&\& \ t_2 \leq 0 \ \&\& \ t_2 \geq -1$  as true condition, but then it also gives a false positive on one side of the finite line, when the circle is on the "infinite" part. I don't understand the math yet, but copy/pasters, beware. – Nicolas Mommaerts Apr 3, 2013 at 22:54

No one seems to consider projection, am I completely off track here?

154

Project the vector  $AC$  onto  $AB$ . The projected vector,  $AD$ , gives the new point  $D$ .  
If the distance between  $D$  and  $C$  is smaller than (or equal to)  $R$  we have an intersection.

Like this:



## Community Edit:

For anyone stumbling across this post later and wondering how such an algorithm can be implemented, here is a general implementation written in JavaScript using common vector manipulation functions.

```
/**
 * Returns the distance from line segment AB to point C
 */
function distanceSegmentToPoint(A, B, C) {
    // Compute vectors AC and AB
    const AC = sub(C, A);
    const AB = sub(B, A);

    // Get point D by taking the projection of AC onto AB then adding the offset
    // of A
    const D = add(proj(AC, AB), A);

    const AD = sub(D, A);
    // D might not be on AB so calculate k of D down AB (aka solve AD = k * AB)
    // We can use either component, but choose larger value to reduce the chance
    // of dividing by zero
```

Join Stack Overflow to find the best answer to your technical question, help others answer theirs.

[Sign up with email](#)
[Sign up with Google](#)
[Sign up with GitHub](#)
[Sign up with Facebook](#)


```

    return Math.sqrt(hypot2(C, B));
}

return Math.sqrt(hypot2(C, D));
}

```

For this implementation I used a couple common vector manipulation functions that you are likely to already have provided in whatever environment you might be working in. However if you do not already have these functions available, here is how they can be implemented.

```

// Define some common functions for working with vectors
const add = (a, b) => ({x: a.x + b.x, y: a.y + b.y});
const sub = (a, b) => ({x: a.x - b.x, y: a.y - b.y});
const dot = (a, b) => a.x * b.x + a.y * b.y;
const hypot2 = (a, b) => dot(sub(a, b), sub(a, b));

// Function for projecting some vector a onto b
function proj(a, b) {
  const k = dot(a, b) / dot(b, b);
  return {x: k * b.x, y: k * b.y};
}

```

Share Follow

edited Jan 22 at 0:51



Locke

5,311 ● 1 ● 20 ● 35

answered Jul 3, 2009 at 13:57



Mizipzor

49.3k ● 22 ● 94 ● 138

- 10 There are many details to take into consideration: does D lie between AB? Is C perpendicular distance to the line larger than the radius? All of these involve magnitude of vector, i.e. square root. – ADB Sep 1, 2010 at 14:42
- 17 Good idea, but how do you then compute the two intersection points? – Ben Feb 29, 2012 at 15:39
- 4 @Spider it doesn't matter. In general, since this is a variant of sphere-line intersection problem, Mizipzor's strategy is perfectly valid. CD is a projection, it is perpendicular by definition. – user719662 Dec 7, 2014 at 0:03
- 2 It's an old question, but there's a good resource on this and related algorithms at this website: [paulbourke.net/geometry/pointlineplane](http://paulbourke.net/geometry/pointlineplane) – Andrew Jan 10, 2015 at 11:00
- 3 Isn't this answer incomplete? It finds whether a circle and line intersect, not a line segment. In my opinion correct check should be something like: (projectionPointInCircle and projectionPointOnLine) or endPointsInCircle Last check is necessary since line segment may penetrate circle and finish before it goes past center. endPoints refer to starting and ending points of line segment. – unlut Jul 18, 2018 at 17:58

53 I would use the algorithm to compute the distance between a point (circle center) and a line (line AB). This can then be used to determine the intersection points of the line with the circle.

Let say we have the points A, B, C. Ax and Ay are the x and y components of the A points. Same for B and C. The scalar R is the circle radius.



This algorithm requires that A, B and C are distinct points and that R is not 0.

Here is the algorithm

```

// compute the euclidean distance between A and B
LAB = sqrt( (Bx-Ax)²+(By-Ay)² )

// compute the direction vector D from A to B
Dx = (Bx-Ax)/LAB
Dy = (By-Ay)/LAB

// the equation of the line AB is x = Dx*t + Ax, y = Dy*t + Ay with 0 <= t <= LAB.

// compute the distance between the points A and E, where
// E is the point of AB closest the circle center (Cx, Cy)

```

Join Stack Overflow to find the best answer to your technical question, help others answer theirs.

Sign up with email

Sign up with Google

Sign up with GitHub

Sign up with Facebook



```
// compute the euclidean distance between E and C
LEC = sqrt((Ex-Cx)^2+(Ey-Cy)^2)

// test if the line intersects the circle
if( LEC < R )
{
    // compute distance from t to circle intersection point
    dt = sqrt( R^2 - LEC^2 )

    // compute first intersection point
    Fx = (t-dt)*Dx + Ax
    Fy = (t-dt)*Dy + Ay

    // compute second intersection point
    Gx = (t+dt)*Dx + Ax
    Gy = (t+dt)*Dy + Ay
}

// else test if the line is tangent to circle
else if( LEC == R )
```

Share Follow

edited Oct 26, 2018 at 12:03

answered Jul 6, 2009 at 16:55



chmike

19.5k ● 21 ● 77 ● 99

if any line that is not intersecting the circle and its both point p1 and p2 are inside circle. in this case how your algorithm work?? – Prashant Dec 12, 2014 at 10:33

- 1 You have to test  $t-dt$  and  $t+dt$ . If  $t-dt < 0$ , then p1 is inside the circle. If  $t+dt > 1$ , then p2 is inside the circle. This is true if  $LEC < R$  of course. – chmike Dec 14, 2014 at 13:13

Thanks. I liked this pgm comments as explanation because there was no use of the words "dot product" since my math is rusty. However  $t$  and  $dt$  are not between 0..1 so while changing this to python I changed  $t$  to be divided by  $LAB^{**2}$ . My understanding is the first division by  $LAB$  is to project the center of the circle to the line AB, and the second division by  $LAB$  is to normalize it into the range 0..1. Also the  $dt$  needs to be divided by  $LAB$  so it is also normalized. Thus "if ( $t-dt > 0.0$ )" first intersection exists "if ( $t+dt <= 1.0$ )" second intersection exists. This worked with testing. – punchcard May 12, 2015 at 14:15

- 2 Because the intersection point with the circle are at "distance"  $t+dt$  and  $t-dt$  on the line.  $t$  is the point on the line closest to the center of the circle. The intersection points with the circle are at a symmetric distance from  $t$ . The intersection points are at "distances"  $t-dt$  and  $t+dt$ . I quoted distance because it's not the euclidian distance. To get the euclidian distance from  $A$  where  $t=0$ , you have to multiply the value by  $LAB$ . – chmike May 27, 2015 at 15:57
- 1 @Matt W You mean "How to determine if the intersection occurs outside the end points of the line section AB"? Just think about  $t$  as a measure of distance along the line. Point A is at  $t=0$ . Point B at  $t=LAB$ . When both intersection points ( $t1=t-dt$  and  $t2=t+dt$ ) have negative values than intersections are outside the section (behind point A looking from the section side of the point). When  $t1$  and  $t2$  are bigger than  $LAB$  then they are outside too (this time behind B point). Intersection  $t1$  (or  $t2$ ) occurs between A and B only when  $t1$  (or  $t2$ ) it is between 0 and  $LAB$ . – Marconius Dec 8, 2015 at 18:59

Okay, I won't give you code, but since you have tagged this [algorithm](#), I don't think that will matter to you. First, you have to get a vector perpendicular to the line.

22

You will have an unknown variable in  $y = ax + c$  ( $c$  will be unknown)

To solve for that, Calculate it's value when the line passes through the center of the circle.



That is,

Plug in the location of the circle center to the line equation and solve for  $c$ .

Then calculate the intersection point of the original line and its normal.

This will give you the closest point on the line to the circle.

Calculate the distance between this point and the circle center (using the magnitude of the vector).

If this is less than the radius of the circle - voila, we have an intersection!

Join Stack Overflow to find the best answer to your technical question, help others answer theirs.

[Sign up with email](#)
[Sign up with Google](#)
[Sign up with GitHub](#)
[Sign up with Facebook](#)


- 2 That was, in fact, what I wanted. I want the theory, a google search of line-circle collision algorithm turns up only code as far as I can see.  
– Mizipzor Jul 2, 2009 at 9:25
- Ok,  $c$  is unknown in your equation, but what is "a"? The other answers seem to refer to that variable as "alpha" and "t". Although, this is just a linear function ( $y=kx+m$ ), quite basic math, so I suddenly feel a bit rusty. Isn't  $k$  also unknown? Or do you mean we can assume  $m=0$  and solve  $k$ ? Wouldn't then  $m$  (that is,  $c$ ) always be zero for our solved  $k$ ? – Mizipzor Jul 3, 2009 at 13:26
- 1 Oh, sorry - I am using the simple equation of a line with a gradient and offset (the cartesian equation). I assumed that you were storing the line as such an equation - in which case you use the negative of the gradient for  $k$ . If you don't have the line stored like this, you could calculate  $k$  as  $(y_2-y_1)/(x_2-x_1)$  – a\_m0d Jul 3, 2009 at 21:22
- 1 We don't assume that  $m$  is zero; we calculate the gradient first (so that the equation of the line then looks like  $y=2x+m$  as an example), and then once we have the gradient we can solve for  $m$  by plugging in the center of the circle for  $y$  and  $x$ . – a\_m0d Jul 3, 2009 at 21:24
- 1 +1 Awesome explanation! But I think this assumes a line, not a line segment. So, if the nearest point on this line to the circle's center wasn't between points A and B, it would still be counted. – user377628 Jun 1, 2012 at 22:01

▲  
15

Another method uses the triangle ABC area formula. The intersection test is simpler and more efficient than the projection method, but finding the coordinates of the intersection point requires more work. At least it will be delayed to the point it is required.



The formula to compute the triangle area is :  $\text{area} = bh/2$



where  $b$  is the base length and  $h$  is the height. We chose the segment AB to be the base so that  $h$  is the shortest distance from C, the circle center, to the line.

Since the triangle area can also be computed by a vector dot product we can determine  $h$ .

```
// compute the triangle area times 2 (area = area2/2)
area2 = abs( (Bx-Ax)*(Cy-Ay) - (Cx-Ax)(By-Ay) )

// compute the AB segment length
LAB = sqrt( (Bx-Ax)^2 + (By-Ay)^2 )

// compute the triangle height
h = area2/LAB

// if the line intersects the circle
if( h < R )
{
    ...
}
```

#### UPDATE 1 :

You could optimize the code by using the fast inverse square root computation described [here](#) to get a good approximation of  $1/LAB$ .

Computing the intersection point is not that difficult. Here it goes

```
// compute the line AB direction vector components
Dx = (Bx-Ax)/LAB
Dy = (By-Ay)/LAB

// compute the distance from A toward B of closest point to C
t = Dx*(Cx-Ax) + Dy*(Cy-Ay)

// t should be equal to sqrt( (Cx-Ax)^2 + (Cy-Ay)^2 - h^2 )

// compute the intersection point distance from t
dt = sqrt( R^2 - h^2 )

// compute first intersection point coordinate
```

**Join Stack Overflow** to find the best answer to your technical question, help others answer theirs.

Sign up with email

Sign up with Google

Sign up with GitHub

Sign up with Facebook



```
Fx = Ax + (t+dt)*Dx
Fy = Ay + (t+dt)*Dy
```

If  $h = R$  then the line AB is tangent to the circle and the value  $dt = 0$  and  $E = F$ . The point coordinates are those of E and F.

You should check that A is different of B and the segment length is not null if this may happen in your application.

Share Follow

edited Jun 22, 2012 at 16:18

answered Jul 7, 2009 at 7:05



**chmike**  
19.5k ● 21 ● 77 ● 99

2 I like the simplicity in this method. Maybe I could adapt some of the surrounding code to not need the actual collision point itself, Ill see what happens if I use A or B rather than the calculated point in between. – [Mizipzor](#) Jul 7, 2009 at 11:00

1  $t = Dx*(Cx-Ax) + Dy*(Cy-Ax)$  should read  $t = Dx*(Cx-Ax) + Dy*(Cy-Ay)$  – [Stonetip](#) Jun 21, 2012 at 15:02 ✎

just edited -- first line calculates triangle area using a *cross* product, not a dot product. verified with code here:  
[stackoverflow.com/questions/2533011/...](http://stackoverflow.com/questions/2533011/) – [ericsoco](#) Jun 1, 2013 at 15:26

5 note also, the first half of this answer tests for intersection with a line, not a line segment (as asked in the question). – [ericsoco](#) Jun 21, 2013 at 18:40

An image to go with this would be very helpful to understand it more – [WDUK](#) Jun 19, 2021 at 22:50

I wrote a small script to test intersection by projecting circle's center point on to line.

9

```
vector distVector = centerPoint - projectedPoint;
if(distVector.length() < circle.radius)
{
    double distance = circle.radius - distVector.length();
    vector moveVector = distVector.normalize() * distance;
    circle.move(moveVector);
}
```

<http://jsfiddle.net/ercang/ornh3594/1/>

If you need to check the collision with the segment, you also need to consider circle center's distance to start and end points.

```
vector distVector = centerPoint - startPoint;
if(distVector.length() < circle.radius)
{
    double distance = circle.radius - distVector.length();
    vector moveVector = distVector.normalize() * distance;
    circle.move(moveVector);
}
```

<https://jsfiddle.net/ercang/menp0991/>

Share Follow

edited Aug 19, 2015 at 11:33

answered Aug 19, 2015 at 11:25



**ercang**  
91 ● 2 ● 3

This solution I found seemed a little easier to follow then some of the other ones.

6

Taking:

p1 and p2 as the points for the line, and

**Join Stack Overflow** to find the best answer to your technical question, help others answer theirs.

[Sign up with email](#)
[Sign up with Google](#)
[Sign up with GitHub](#)
[Sign up with Facebook](#)


```
p3 = p1 - c
p4 = p2 - c
```

By the way, whenever I subtract points from each other I am subtracting the `x`'s and then subtracting the `y`'s, and putting them into a new point, just in case someone didn't know.

Anyway, I now solve for the equation of the line with `p3` and `p4`:

```
m = (p4_y - p3_y) / (p4_x - p3) (the underscore is an attempt at subscript)
y = mx + b
y - mx = b (just put in a point for x and y, and insert the m we found)
```

Ok. Now I need to set these equations equal. First I need to solve the circle's equation for `x`

```
x^2 + y^2 = r^2
y^2 = r^2 - x^2
y = sqrt(r^2 - x^2)
```

Then I set them equal:

```
mx + b = sqrt(r^2 - x^2)
```

And solve for the quadratic equation ( $\theta = ax^2 + bx + c$ ):

```
(mx + b)^2 = r^2 - x^2
(mx)^2 + 2mbx + b^2 = r^2 - x^2
0 = m^2 * x^2 + x^2 + 2mbx + b^2 - r^2
0 = (m^2 + 1) * x^2 + 2mbx + b^2 - r^2
```

Now I have my `a`, `b`, and `c`.

```
a = m^2 + 1
b = 2mb
c = b^2 - r^2
```

So I put this into the quadratic formula:

```
(-b ± sqrt(b^2 - 4ac)) / 2a
```

And substitute in by values then simplify as much as possible:

```
(-2mb ± sqrt(b^2 - 4ac)) / 2a
(-2mb ± sqrt((-2mb)^2 - 4(m^2 + 1)(b^2 - r^2))) / 2(m^2 + 1)
(-2mb ± sqrt(4m^2 * b^2 - 4(m^2 * b^2 - m^2 * r^2 + b^2 - r^2))) / 2m^2 + 2
(-2mb ± sqrt(4 * (m^2 * b^2 - (m^2 * b^2 - m^2 * r^2 + b^2 - r^2)))) / 2m^2 + 2
(-2mb ± sqrt(4 * (m^2 * b^2 - m^2 * b^2 + m^2 * r^2 - b^2 + r^2))) / 2m^2 + 2
(-2mb ± sqrt(4 * (m^2 * r^2 - b^2 + r^2))) / 2m^2 + 2
(-2mb ± sqrt(4) * sqrt(m^2 * r^2 - b^2 + r^2)) / 2m^2 + 2
(-2mb ± 2 * sqrt(m^2 * r^2 - b^2 + r^2)) / 2m^2 + 2
(-2mb ± 2 * sqrt(m^2 * r^2 + r^2 - b^2)) / 2m^2 + 2
(-2mb ± 2 * sqrt(r^2 * (m^2 + 1) - b^2)) / 2m^2 + 2
```


This is almost as far as it will simplify. Finally, separate out to equations with the  $\pm$ :

```
(-2mb + 2 * sqrt(r^2 * (m^2 + 1) - b^2)) / 2m^2 + 2 or
```

**Join Stack Overflow** to find the best answer to your technical question, help others answer theirs.

Sign up with email

 Sign up with Google

 Sign up with GitHub

 Sign up with Facebook

X



```
function interceptOnCircle(p1,p2,c,r){
  //p1 is the first line point
  //p2 is the second line point
  //c is the circle's center
  //r is the circle's radius

  var p3 = {x:p1.x - c.x, y:p1.y - c.y} //shifted line points
  var p4 = {x:p2.x - c.x, y:p2.y - c.y}

  var m = (p4.y - p3.y) / (p4.x - p3.x); //slope of the line
  var b = p3.y - m * p3.x; //y-intercept of line

  var underRadical = Math.pow((Math.pow(r,2)*(Math.pow(m,2)+1)),2)-Math.pow(b,2)); //the value under the square root
  sign

  if (underRadical < 0){
    //line completely missed
    return false;
  } else {
    var t1 = (-2*m*b+2*Math.sqrt(underRadical))/(2 * Math.pow(m,2) + 2); //one of the intercept x's
    var t2 = (-2*m*b-2*Math.sqrt(underRadical))/(2 * Math.pow(m,2) + 2); //other intercept's x
    var i1 = {x:t1,y:m*t1+b} //intercept point 1
    var i2 = {x:t2,y:m*t2+b} //intercept point 2
    return [i1,i2];
  }
}
```

I hope this helps!

P.S. If anyone finds any errors or has any suggestions, please comment. I am very new and welcome all help/suggestions.

Share Follow

edited Mar 1, 2014 at 5:14

answered Mar 1, 2014 at 4:42



Alexey  
2,502 ● 2 ● 29 ● 51



multitaskPro  
539 ● 4 ● 13

If possible, also post with some sample values so that we can quickly grasp the flow. – Prabindh Mar 1, 2014 at 5:01

2 With `underRadical` extra ')' – byJeevan Feb 25, 2016 at 6:54

▲ Here's an implementation in Javascript. My approach is to first convert the line segment into an infinite line then find the intersection point(s). From there I check if the point(s) found are on the line segment. The code is well documented, you should be able to follow along.

5



You can try out the code here on this [live demo](#). The code was taken from my [algorithms repo](#).



**Join Stack Overflow** to find the best answer to your technical question, help others answer theirs.

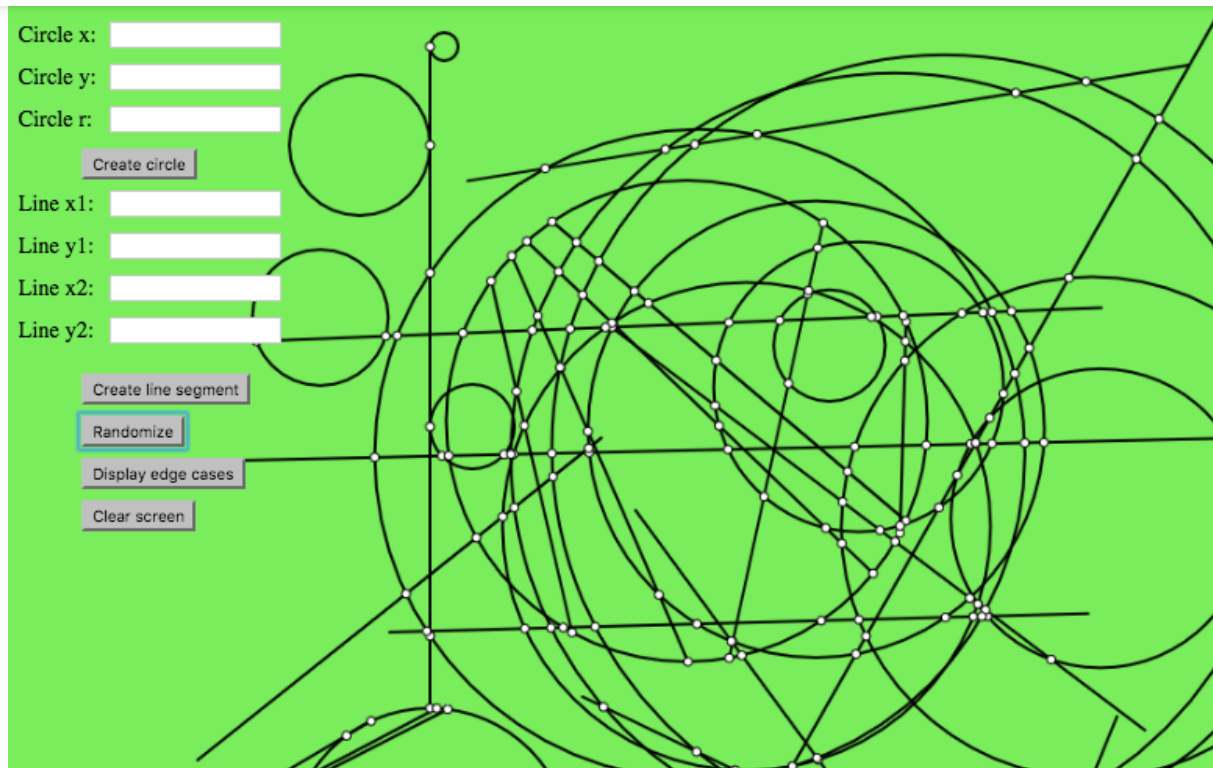
Sign up with email

Sign up with Google

Sign up with GitHub

Sign up with Facebook





```
// Small epsilon value
var EPS = 0.0000001;

// point (x, y)
function Point(x, y) {
  this.x = x;
  this.y = y;
}

// Circle with center at (x,y) and radius r
function Circle(x, y, r) {
  this.x = x;
  this.y = y;
  this.r = r;
}

// A line segment (x1, y1), (x2, y2)
function LineSegment(x1, y1, x2, y2) {
  var d = Math.sqrt((x1-x2)*(x1-x2) + (y1-y2)*(y1-y2));
  if (d < EPS) throw 'A point is not a line segment';
  this.x1 = x1; this.y1 = y1;
  this.x2 = x2; this.y2 = y2;
}


// An infinite line defined as: ax + by = c
function Line(a, b, c) {
  this.a = a; this.b = b; this.c = c;
  // Normalize line for good measure
  if (Math.abs(b) < EPS) {
    c /= a; a = 1; b = 0;
  } else {
    a = (Math.abs(a) < EPS) ? 0 : a / b;
    c /= b; b = 1;
  }
}

// Given a line in standard form: ax + by = c and a circle with
// a center at (x,y) with radius r this method finds the intersection
// of the line and the circle (if any).
function circleLineIntersection(circle, line) {
```

Join Stack Overflow to find the best answer to your technical question, help others answer theirs.

Sign up with email

 Sign up with Google

 Sign up with GitHub

 Sign up with Facebook

×

```

expand to obtain quadratic:
// (a^2 + b^2)x^2 + (2abY - 2ac + - 2b^2X)x + (b^2X^2 + b^2Y^2 - 2bcY + c^2 -
b^2r^2) = 0
// Then use quadratic formula  $X = (-b \pm \sqrt{a^2 - 4ac})/2a$  to find the
// roots of the equation (if they exist) and this will tell us the intersection
points

// In general a quadratic is written as:  $Ax^2 + Bx + C = 0$ 
// (a^2 + b^2)x^2 + (2abY - 2ac + - 2b^2X)x + (b^2X^2 + b^2Y^2 - 2bcY + c^2 -
b^2r^2) = 0
var A = a*a + b*b;
var B = 2*a*b*y - 2*a*c - 2*b*b*x;
var C = b*b*x*x + b*b*y*y - 2*b*c*y + c*c - b*b*r*r;

// Use quadratic formula  $x = (-b \pm \sqrt{a^2 - 4ac})/2a$  to find the
// roots of the equation (if they exist).

var D = B*B - 4*A*C;
var x1,y1,x2,y2;

// Handle vertical line case with b = 0
if (Math.abs(b) < EPS) {

    // Line equation is  $ax + by = c$ , but  $b = 0$ , so  $x = c/a$ 
    x1 = c/a;

    // No intersection
    if (Math.abs(x-x1) > r) return [];

    // Vertical line is tangent to circle
    if (Math.abs((x1-r)-x) < EPS || Math.abs((x1+r)-x) < EPS)
        return [new Point(x1, y)];

    var dx = Math.abs(x1 - x);
    var dy = Math.sqrt(r*r-dx*dx);

    // Vertical line cuts through circle
    return [
        new Point(x1,y+dy),
        new Point(x1,y-dy)
    ];

    // Line is tangent to circle
} else if (Math.abs(D) < EPS) {

    x1 = -B/(2*A);
    y1 = (c - a*x1)/b;

    return [new Point(x1,y1)];

    // No intersection
} else if (D < 0) {

    return [];

} else {

    D = Math.sqrt(D);

    x1 = (-B+D)/(2*A);
    y1 = (c - a*x1)/b;

    x2 = (-B-D)/(2*A);
    y2 = (c - a*x2)/b;

    return [
        new Point(x1, y1),
        new Point(x2, y2)
    ];


}


}


```

Join Stack Overflow to find the best answer to your technical question, help others answer theirs.

Sign up with email

 Sign up with Google

 Sign up with GitHub

 Sign up with Facebook



```

}

// Checks if a point 'pt' is inside the rect defined by (x1,y1), (x2,y2)
function pointInRectangle(pt,x1,y1,x2,y2) {
  var x = Math.min(x1,x2), X = Math.max(x1,x2);
  var y = Math.min(y1,y2), Y = Math.max(y1,y2);
  return x - EPS <= pt.x && pt.x <= X + EPS &&
         y - EPS <= pt.y && pt.y <= Y + EPS;
}

// Finds the intersection(s) of a line segment and a circle
function lineSegmentCircleIntersection(segment, circle) {

  var x1 = segment.x1, y1 = segment.y1, x2 = segment.x2, y2 = segment.y2;
  var line = segmentToGeneralForm(x1,y1,x2,y2);
  var pts = circleLineIntersection(circle, line);

  // No intersection
  if (pts.length === 0) return [];

  var pt1 = pts[0];
  var includePt1 = pointInRectangle(pt1,x1,y1,x2,y2);

  // Check for unique intersection
  if (pts.length === 1) {
    if (includePt1) return [pt1];
    return [];
  }

  var pt2 = pts[1];
  var includePt2 = pointInRectangle(pt2,x1,y1,x2,y2);

  // Check for remaining intersections
  if (includePt1 && includePt2) return [pt1, pt2];
  if (includePt1) return [pt1];
  if (includePt2) return [pt2];
  return [];
}

```

Share Follow

edited Jul 7, 2017 at 17:56

answered Jul 7, 2017 at 16:02



will.fiset

1,464 ● 1 ● 17 ● 28

4

You can find a point on a infinite line that is nearest to circle center by projecting vector AC onto vector AB. Calculate the distance between that point and circle center. If it is greater that R, there is no intersection. If the distance is equal to R, line is a tangent of the circle and the point nearest to circle center is actually the intersection point. If distance less that R, then there are 2 intersection points. They lie at the same distance from the point nearest to circle center. That distance can easily be calculated using Pythagorean theorem. Here's algorithm in pseudocode:

```

{
  dX = bX - aX;
  dY = bY - aY;
  if ((dX == 0) && (dY == 0))
  {
    // A and B are the same points, no way to calculate intersection
    return;
  }

  dL = (dX * dX + dY * dY);
  t = ((cX - aX) * dX + (cY - aY) * dY) / dL;

  // point on a line nearest to circle center
  nearestX = aX + t * dX;
  nearestY = aY + t * dY;

  dist = point_dist(nearestX, nearestY, cX, cY);
}

```

**Join Stack Overflow** to find the best answer to your technical question, help others answer theirs.

Sign up with email

Sign up with Google

Sign up with GitHub

Sign up with Facebook

X

```

if (t < 0 || t > 1)
{
    // intersection point is not actually within line segment
}
else if (dist < R)
{
    // two possible intersection points

    dt = sqrt(R * R - dist * dist) / sqrt(dl);

    // intersection point nearest to A
    t1 = t - dt;

```

EDIT: added code to check whether found intersection points actually are within line segment.

Share Follow

edited Sep 2, 2010 at 10:41

answered Jul 3, 2009 at 13:59



Juozas Kontvainis

9,161 ● 6 ● 53 ● 65

You missed one case since we are talking about a line segment: when the segment ends in the circle. – ADB Sep 1, 2010 at 14:47

@ADB actually my algorithm only works for infinite lines only, not line segments. There are many cases that it does not handle with line segments. – Juozas Kontvainis Sep 2, 2010 at 6:52

The original question was about line-segments, not circle-line intersection, which is a much easier problem. – msumme Jan 13, 2017 at 18:35

4 Weirdly I can answer but not comment... I liked Multitaskpro's approach of shifting everything to make the centre of the circle fall on the origin. Unfortunately there are two problems in his code. First in the under-the-square-root part you need to remove the double power. So not:

```
var underRadical = Math.pow((Math.pow(r,2)*(Math.pow(m,2)+1)),2)-Math.pow(b,2));
```



but:

```
var underRadical = Math.pow(r,2)*(Math.pow(m,2)+1)) - Math.pow(b,2);
```

In the final coordinates he forgets to shift the solution back. So not:

```
var i1 = {x:t1,y:m*t1+b}
```

but:

```
var i1 = {x:t1+c.x, y:m*t1+b+c.y};
```

The whole function then becomes:

```

function interceptOnCircle(p1, p2, c, r) {
    //p1 is the first line point
    //p2 is the second line point
    //c is the circle's center
    //r is the circle's radius

    var p3 = {x:p1.x - c.x, y:p1.y - c.y}; //shifted line points
    var p4 = {x:p2.x - c.x, y:p2.y - c.y};

    var m = (p4.y - p3.y) / (p4.x - p3.x); //slope of the line
    var b = p3.y - m * p3.x; //y-intercept of line

    var underRadical = Math.pow(r,2)*Math.pow(m,2) + Math.pow(r,2) - Math.pow(b,2); //the value under the square root
    sign

```

Join Stack Overflow to find the best answer to your technical question, help others answer theirs.

Sign up with email

Sign up with Google

Sign up with GitHub

Sign up with Facebook



```

var t2 = (-m*b - Math.sqrt(Math.pow(m,2) + 1))/(Math.pow(m,2) + 1); //other intercept's x
var i1 = {x:t1+c.x, y:m*t1+b+c.y}; //intercept point 1
var i2 = {x:t2+c.x, y:m*t2+b+c.y}; //intercept point 2
return [i1, i2];
}
}

```

Share Follow

answered Aug 13, 2015 at 23:42

Duq  
51 ● 3

- 1 suggestions: First, have it handle the case where line segment is vertical (i.e. has infinite slope). Secondly, have it only return those points that actually fall within the range of the original line segment -- i believe it's happily returning all points that fall on the infinite line, even if those points lie outside of the line segment. – Gino Jan 12, 2016 at 18:05

Note: This works well for lines, but does not work for line segments. – Mike Feb 22, 2019 at 14:05

▲ You'll need some math here:

3

Suppose  $A = (X_a, Y_a)$ ,  $B = (X_b, Y_b)$  and  $C = (X_c, Y_c)$ . Any point on the line from A to B has coordinates  $(\alpha X_a + (1-\alpha)X_b, \alpha Y_a + (1-\alpha)Y_b) = P$



If the point P has distance R to C, it must be on the circle. What you want is to solve



```
distance(P, C) = R
```

that is

```

(alpha*Xa + (1-alpha)*Xb)^2 + (alpha*Ya + (1-alpha)*Yb)^2 = R^2
alpha^2*Xa^2 + alpha^2*Xb^2 - 2*alpha*Xb^2 + Xb^2 + alpha^2*Ya^2 + alpha^2*Yb^2 - 2*alpha*Yb^2 + Yb^2=R^2
(Xa^2 + Xb^2 + Ya^2 + Yb^2)*alpha^2 - 2*(Xb^2 + Yb^2)*alpha + (Xb^2 + Yb^2 - R^2) = 0

```

if you apply the ABC-formula to this equation to solve it for alpha, and compute the coordinates of P using the solution(s) for alpha, you get the intersection points, if any exist.

Share Follow

answered Jul 2, 2009 at 9:29

Martijn  
5,281 ● 4 ● 36 ● 50

▲ Just an addition to this thread... Below is a version of the code posted by pahlevan, but for C#/XNA and tidied up a little:

3



```

/// <summary>
/// Intersects a line and a circle.
/// </summary>
/// <param name="location">the location of the circle</param>
/// <param name="radius">the radius of the circle</param>
/// <param name="lineFrom">the starting point of the line</param>
/// <param name="lineTo">the ending point of the line</param>
/// <returns>true if the line and circle intersect each other</returns>
public static bool IntersectLineCircle(Vector2 location, float radius, Vector2 lineFrom, Vector2 lineTo)
{
    float ab2, acab, h2;
    Vector2 ac = location - lineFrom;
    Vector2 ab = lineTo - lineFrom;
    Vector2.Dot(ref ab, ref ab, out ab2);
    Vector2.Dot(ref ac, ref ab, out acab);
    float t = acab / ab2;

```

**Join Stack Overflow** to find the best answer to your technical question, help others answer theirs.

[Sign up with email](#)
[Sign up with Google](#)
[Sign up with GitHub](#)
[Sign up with Facebook](#)


```

Vector2 h = ((ab * t) + lineFrom) - location;
Vector2.Dot(ref h, ref h, out h2);

return (h2 <= (radius * radius));
}

```

Share Follow

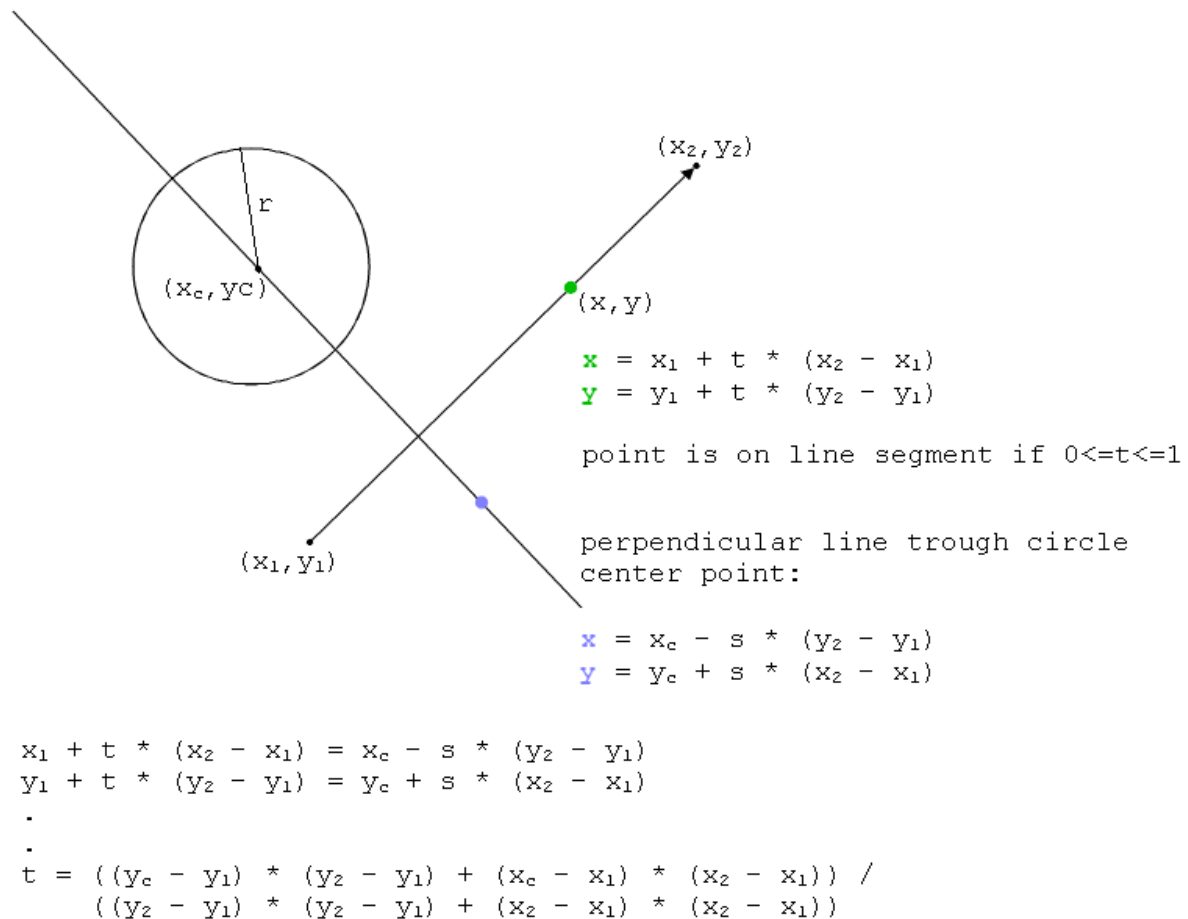
answered May 1, 2012 at 2:30



Rob

1,659 ● 3 ● 22 ● 34

In C#/XNA you can use [Ray.Intersects\(BoundingSphere\)](#) – bobobobo Nov 26, 2012 at 20:34



VB.NET - Code

```

Function CheckLineSegmentCircleIntersection(x1 As Double, y1 As Double, x2 As Double, y2 As Double, xc As Double, yc
As Double, r As Double) As Boolean
    Static xd As Double = 0.0F
    Static yd As Double = 0.0F
    Static t As Double = 0.0F
    Static d As Double = 0.0F
    Static dx_2_1 As Double = 0.0F
    Static dy_2_1 As Double = 0.0F

    dx_2_1 = x2 - x1
    dy_2_1 = y2 - y1

    t = ((yc - y1) * dy_2_1 + (xc - x1) * dx_2_1) / (dy_2_1 * dy_2_1 + dx_2_1 * dx_2_1)

```

**Join Stack Overflow** to find the best answer to your technical question, help others answer theirs.

[Sign up with email](#)
[Sign up with Google](#)
[Sign up with GitHub](#)
[Sign up with Facebook](#)


```

    Return d <= r
Else
    d = Math.Sqrt((xc - x1) * (xc - x1) + (yc - y1) * (yc - y1))
    If d <= r Then
        Return True
    Else
        d = Math.Sqrt((xc - x2) * (xc - x2) + (yc - y2) * (yc - y2))
        If d <= r Then
            Return True
        Else
            Return False
        End If
    End If
End If
End Function

```

Share Follow

edited Feb 27, 2014 at 10:43

answered Feb 24, 2014 at 13:34



A.J. Bauer

2,665 ● 1 ● 25 ● 34

I have created this function for iOS following the answer given by [chmike](#)

3

```

+ (NSArray *)intersectionPointsOfCircleWithCenter:(CGPoint)center withRadius:(float)radius toLinePoint1:(CGPoint)p1
andLinePoint2:(CGPoint)p2
{
    NSMutableArray *intersectionPoints = [NSMutableArray array];

    float Ax = p1.x;
    float Ay = p1.y;
    float Bx = p2.x;
    float By = p2.y;
    float Cx = center.x;
    float Cy = center.y;
    float R = radius;

    // compute the euclidean distance between A and B
    float LAB = sqrt( pow(Bx-Ax, 2)+pow(By-Ay, 2) );

    // compute the direction vector D from A to B
    float Dx = (Bx-Ax)/LAB;
    float Dy = (By-Ay)/LAB;

    // Now the line equation is x = Dx*t + Ax, y = Dy*t + Ay with 0 <= t <= 1.

    // compute the value t of the closest point to the circle center (Cx, Cy)
    float t = Dx*(Cx-Ax) + Dy*(Cy-Ay);

    // This is the projection of C on the line from A to B.

    // compute the coordinates of the point E on line and closest to C
    float Ex = t*Dx+Ax;
    float Ey = t*Dy+Ay;

    // compute the euclidean distance from E to C
    float LEC = sqrt( pow(Ex-Cx, 2)+ pow(Ey-Cy, 2) );

    // test if the line intersects the circle
    if( LEC < R )
    {

```

Share Follow

answered Nov 24, 2014 at 13:30



Aqib Mumtaz

4,868 ● 1 ● 33 ● 33

Circle is really a bad guy :) So a good way is to avoid true circle, if you can. If you are doing collision check for games you can go

**Join Stack Overflow** to find the best answer to your technical question, help others answer theirs.

Sign up with email

Sign up with Google

Sign up with GitHub

Sign up with Facebook







First, i would consider renaming and switching coordinate system to avoid excessive data:

```
s0s1 = B-A;
s0qp = C-A;
rSqr = r*r;
```

Second, index h in hvec2f means than vector must favor horisontal operations, like dot()/det(). Which means its components are to be placed in a separate xmm registers, to avoid shuffling/hadd'ing/hsub'ing. And here we go, with most performant version of simpliest collision detection for 2D game:

```
bool fat_point_collides_segment(const hvec2f& s0qp, const hvec2f& s0s1, const float& rSqr) {
    auto a = dot(s0s1, s0s1);
    //if( a != 0 ) // if you haven't zero-length segments omit this, as it would save you 1 _mm_comineq_ss()
    instruction and 1 memory fetch
    {
        auto b = dot(s0s1, s0qp);
        auto t = b / a; // length of projection of s0qp onto s0s1
        //std::cout << "t = " << t << "\n";
        if ((t >= 0) && (t <= 1)) //
        {
            auto c = dot(s0qp, s0qp);
            auto r2 = c - a * t * t;
            return (r2 <= rSqr); // true if collides
        }
    }
    return false;
}
```

I doubt you can optimize it any further. I am using it for neural-network driven car racing collision detection, to process millions of millions iteration steps.

Share Follow

edited Feb 25, 2018 at 17:13

answered Feb 25, 2018 at 16:58



xakepp35

2,598 ● 6 ● 22 ● 47

If the line segment intersects the circle but only slightly so it does not pass its center point, won't this function return false when it should return true? The t value might be outside the range 0..1. – Chris Jun 17, 2020 at 13:44



3



In this post circle line collision will be checked by checking distance between circle center and point on line segment (lpoint) that represent intersection point between normal N (Image 2) from circle center to line segment.

**Join Stack Overflow** to find the best answer to your technical question, help others answer theirs.

Sign up with email

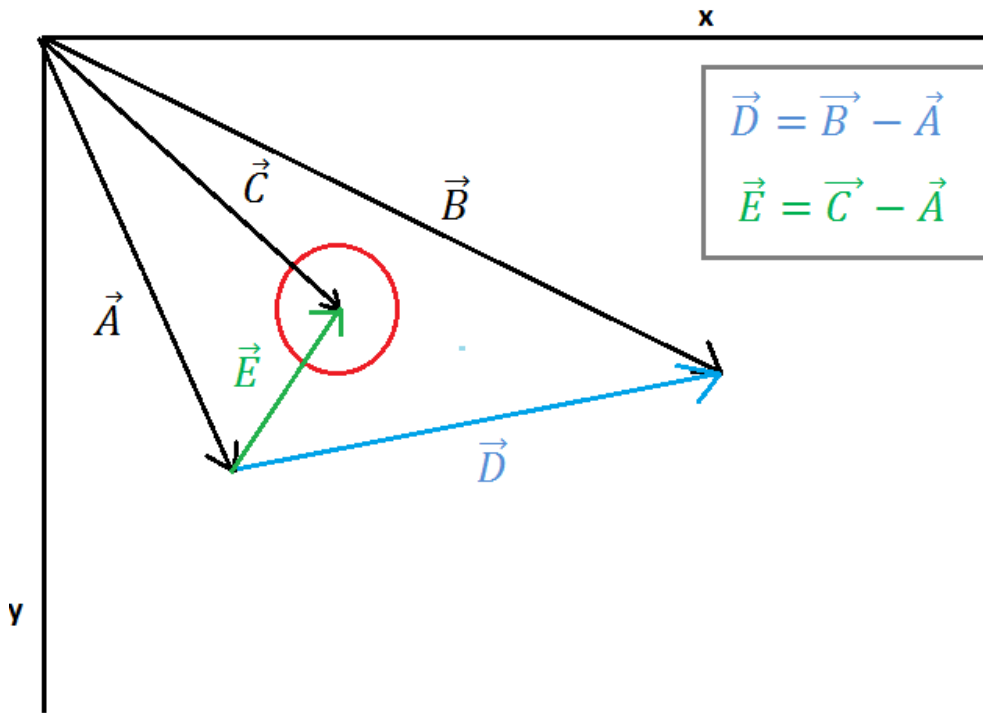
Sign up with Google

Sign up with GitHub

Sign up with Facebook

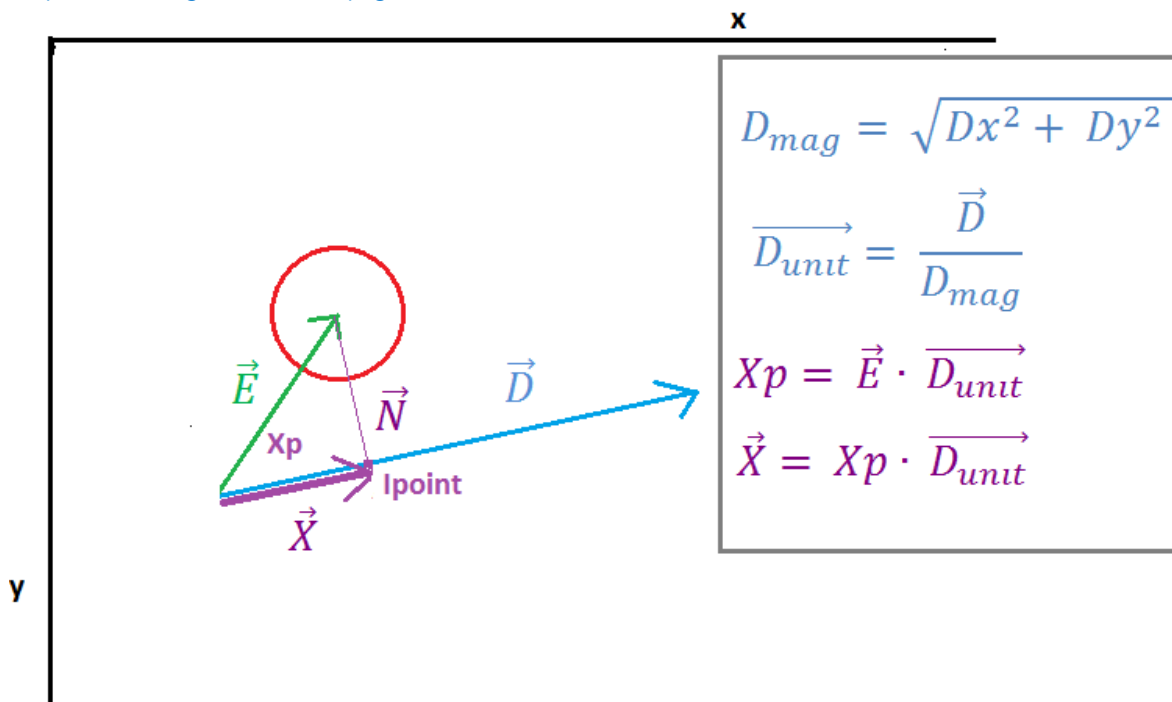


(<https://i.stack.imgur.com/3o6do.png>)




On image 1 one circle and one line are shown, vector A point to line start point, vector B point to line end point, vector C point to circle center. Now we must find vector E (from line start point to circle center) and vector D (from line start point to line end point) this calculation is shown on image 1.


(<https://i.stack.imgur.com/7098a.png>)




**Join Stack Overflow** to find the best answer to your technical question, help others answer theirs.

Sign up with email

 Sign up with Google

 Sign up with GitHub

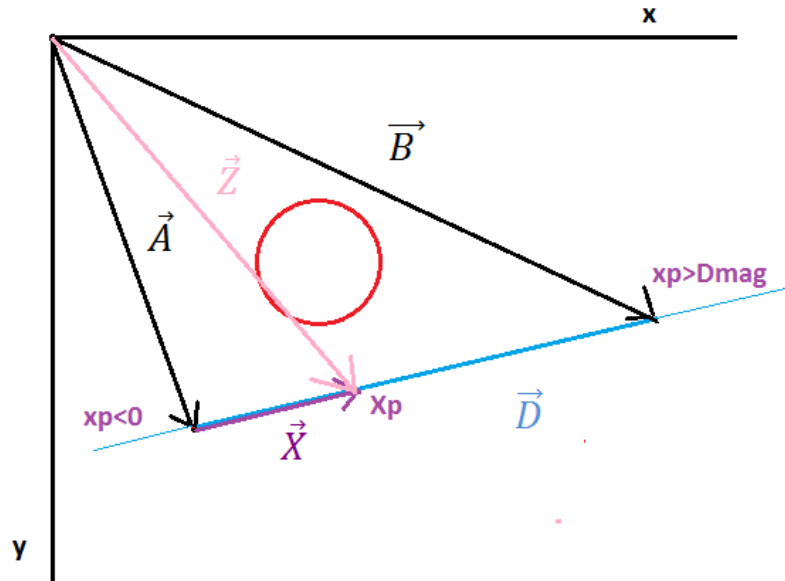
 Sign up with Facebook

X

At image 2 we can see that vector E is projected on Vector D by "dot product" of vector E and unit vector D, result of dot product is scalar  $X_p$  that represent the distance between line start point and point of intersection (lpoint) of vector N and vector D. Next vector X is found by multiplying unit vector D and scalar  $X_p$ .

Now we need to find vector Z (vector to lpoint), its easy its simple vector addition of vector A (start point on line) and vector X. Next we need to deal with special cases we must check is lpoint on line segment, if its not we must find out is it left of it or right of it, we will use vector closest to determine which point is closest to circle.

(<https://i.stack.imgur.com/p9Wlr.png>)



```

 $\vec{Z} = \vec{A} + \vec{X}$ 
IF( $X_p < 0$ ) {
     $\vec{Closest} = \vec{A}$ 
} ELSE IF( $X_p > D_{mag}$ ) {
     $\vec{Closest} = \vec{B}$ 
} ELSE {
     $\vec{Closest} = \vec{Z}$ 
}

```

When projection  $X_p$  is negative lpoint is left of line segment, vector closest is equal to vector of line start point, when projection  $X_p$  is greater then magnitude of vector D then lpoint is right of line segment then closest vector is equal to vector of line end point in any other case closest vector is equal to vector Z.

Now when we have closest vector, we need to find vector from circle center to lpoint (dist vector), its simple we just need to subtract closest vector from center vector. Next just check if vector dist magnitude is less then circle radius if it is then they collide, if its not there is no collision.

Join Stack Overflow to find the best answer to your technical question, help others answer theirs.

Sign up with email

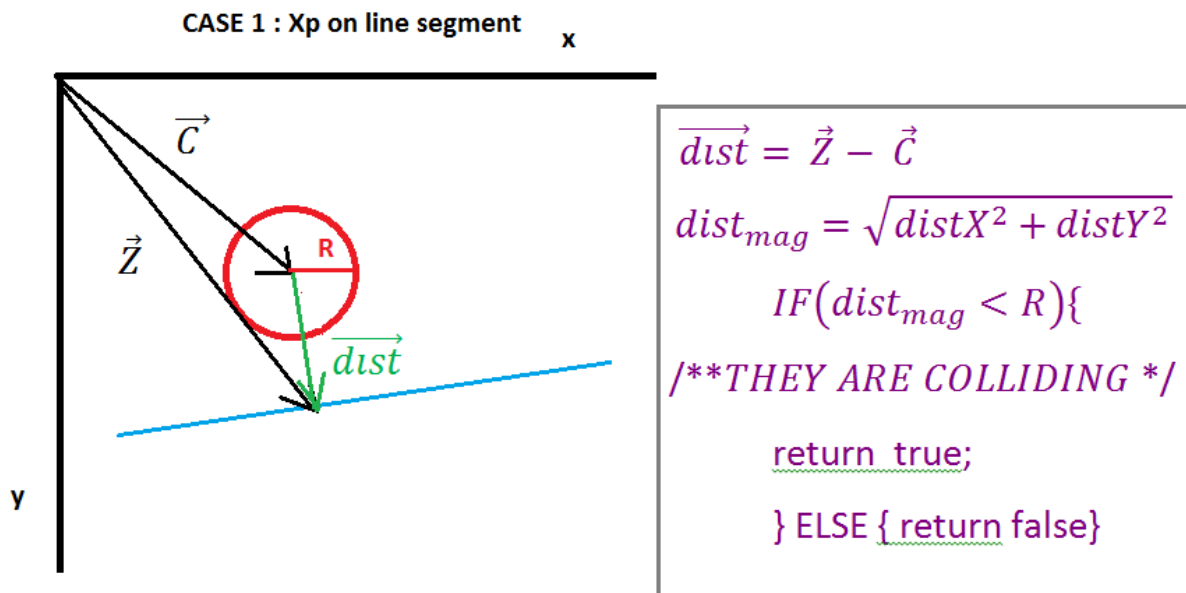
Sign up with Google

Sign up with GitHub

Sign up with Facebook

X

(<https://i.stack.imgur.com/QJ63q.png>)



For end, we can return some values for resolving collision , easiest way is to return overlap of collision (subtract radius from vector dist magnitude) and return axis of collision, its vector D. Also intersection point is vector Z if needed.

Share Follow

edited Mar 18, 2018 at 22:19

answered Mar 17, 2018 at 13:10



FunDeveloper89

31 ● 2

If the line's coordinates are A.x, A.y and B.x, B.y and the circles center is C.x, C.y then the lines formulae are:

$$x = A.x * t + B.x * (1 - t)$$

$$y = A.y * t + B.y * (1 - t)$$

where  $0 \leq t \leq 1$

and the circle is

$$(C.x - x)^2 + (C.y - y)^2 = R^2$$

if you substitute x and y formulae of the line into the circles formula you get a second order equation of t and its solutions are the intersection points (if there are any). If you get a t which is smaller than 0 or greater than 1 then its not a solution but it shows that the line is 'pointing' to the direction of the circle.

Share Follow

edited Jul 2, 2009 at 9:30

answered Jul 2, 2009 at 9:24



Gábor Hargitai

534 ● 2 ● 4

If you find the distance between the center of the sphere (since it's 3D I assume you mean sphere and not circle) and the line, then check if that distance is less than the radius that will do the trick.

2

**Join Stack Overflow** to find the best answer to your technical question, help others answer theirs.

Sign up with email

Sign up with Google

Sign up with GitHub

Sign up with Facebook

X

Distance between a point and a line:

<http://mathworld.wolfram.com/Point-LineDistance3-Dimensional.html>

Share Follow

edited Jul 2, 2009 at 21:17

answered Jul 2, 2009 at 9:20



Martin

12.3k ●12 ●61 ●125

1 It's in 2D, not 3D; as you say, this doesn't really matter – [Martijn](#) Jul 2, 2009 at 9:21

I'm no mathematician so I thought I'd be better outlining a general approach and leaving it to others to figure out specific maths (although I does look rather trivial) – [Martin](#) Jul 2, 2009 at 9:30

2 +1 with a strong upvote. (though I would have linked to another site, the pbourke site looks confusing) All the other answers so far are overcomplicated. Although your comment "That point is also the intersection point on the line" is incorrect, there is no point that has been constructed in the computation process. – [Jason S](#) Jul 2, 2009 at 19:03

2 [mathworld.wolfram.com/Point-LineDistance3-Dimensional.html](http://mathworld.wolfram.com/Point-LineDistance3-Dimensional.html) and [mathworld.wolfram.com/Point-LineDistance2-Dimensional.html](http://mathworld.wolfram.com/Point-LineDistance2-Dimensional.html) are better & from a more reputable site – [Jason S](#) Jul 2, 2009 at 19:05

I explained a little better about the closest point, and linked to mathworld instead of pbourke :) – [Martin](#) Jul 2, 2009 at 21:19

Another one in c# (partial Circle class). Tested and works like a charm.

```
public class Circle : IEquatable<Circle>
{
    // *****
    // The center of a circle
    private Point _center;
    // The radius of a circle
    private double _radius;

    // *****
    /// <summary>
    /// Find all intersections (0, 1, 2) of the circle with a line defined by its 2 points.
    /// Using: http://math.stackexchange.com/questions/228841/how-do-i-calculate-the-intersections-of-a-straight-
    line-and-a-circle
    /// Note: p is the Center.X and q is Center.Y
    /// </summary>
    /// <param name="linePoint1"></param>
    /// <param name="linePoint2"></param>
    /// <returns></returns>
    public List<Point> GetIntersections(Point linePoint1, Point linePoint2)
    {
        List<Point> intersections = new List<Point>();

        double dx = linePoint2.X - linePoint1.X;

        if (dx>AboutEquals(0)) // Straight vertical line
        {
            if (linePoint1.X>AboutEquals(Center.X - Radius) || linePoint1.X>AboutEquals(Center.X + Radius))
            {
                Point pt = new Point(linePoint1.X, Center.Y);
                intersections.Add(pt);
            }
            else if (linePoint1.X > Center.X - Radius && linePoint1.X < Center.X + Radius)
            {
                double x = linePoint1.X - Center.X;

                Point pt = new Point(linePoint1.X, Center.Y + Math.Sqrt(Radius * Radius - (x * x)));
                intersections.Add(pt);
            }
        }
    }
}
```

Required:

```
using System;
```

**Join Stack Overflow** to find the best answer to your technical question, help others answer theirs.

[Sign up with email](#)

[Sign up with Google](#)

[Sign up with GitHub](#)

[Sign up with Facebook](#)



```
// http://stackoverflow.com/questions/2411392/double-epsilon-for-equality-greater-than-less-than-less-than-
// or-equal-to-gre

/// <summary>
/// Compare two double taking in account the double precision potential error.
/// Take care: truncation errors accumulate on calculation. More you do, more you should increase the
epsilon.
public static bool AboutEquals(this double value1, double value2)
{
    if (double.IsPositiveInfinity(value1))
        return double.IsPositiveInfinity(value2);

    if (double.IsNegativeInfinity(value1))
        return double.IsNegativeInfinity(value2);

    if (double.IsNaN(value1))
        return double.IsNaN(value2);

    double epsilon = Math.Max(Math.Abs(value1), Math.Abs(value2)) * 1E-15;
    return Math.Abs(value1 - value2) <= epsilon;
}

// *****
// Base on Hans Passant Answer on:
// http://stackoverflow.com/questions/2411392/double-epsilon-for-equality-greater-than-less-than-less-than-
// or-equal-to-gre

/// <summary>
/// Compare two double taking in account the double precision potential error.
/// Take care: truncation errors accumulate on calculation. More you do, more you should increase the
```

Share Follow

edited May 13, 2016 at 16:57

answered May 12, 2016 at 17:44



Eric Ouellet

10.3k ● 10 ● 80 ● 112

Here is good solution in JavaScript (with all required mathematics and live illustration) <https://bl.ocks.org/milkbread/11000965>

2 Though `is_on` function in that solution needs modifications:

```
function is_on(a, b, c) {
    return Math.abs(distance(a,c) + distance(c,b) - distance(a,b)) < 0.000001;
}
```

Run code snippet

[Expand snippet](#)

Share Follow

edited Jun 8, 2017 at 11:19

answered Jun 8, 2017 at 11:07



nazar kuliye

1,155 ● 1 ● 11 ● 13

I know it's been a while since this thread was open. From the answer given by chmike and improved by Aqib Mumtaz. They give a good answer but only works for a infinite line as said Aqib. So I add some comparisons to know if the line segment touch the circle, I write it in Python.

```
def LineIntersectCircle(c, r, p1, p2):
    #p1 is the first line point
    #p2 is the second line point
    #c is the circle's center
    #r is the circle's radius

    p3 = [p1[0]-c[0], p1[1]-c[1]]
    p4 = [p2[0]-c[0], p2[1]-c[1]]
```

**Join Stack Overflow** to find the best answer to your technical question, help others answer theirs.

Sign up with email

Sign up with Google

Sign up with GitHub

Sign up with Facebook

X

```

if (underRadical < 0):
    print("NOT")
else:
    t1 = (-2*m*b+2*math.sqrt(underRadical)) / (2 * math.pow(m,2) + 2)
    t2 = (-2*m*b-2*math.sqrt(underRadical)) / (2 * math.pow(m,2) + 2)
    i1 = [t1+c[0], m * t1 + b + c[1]]
    i2 = [t2+c[0], m * t2 + b + c[1]]

    if p1[0] > p2[0]:
        if (i1[0] < p1[0]) and (i1[0] > p2[0]):
            if p1[1] > p2[1]:
                if (i1[1] < p1[1]) and (i1[1] > p2[1]):
                    print("Intersection")
            if p1[1] < p2[1]:
                if (i1[1] > p1[1]) and (i1[1] < p2[1]):
                    print("Intersection")
        if p1[0] < p2[0]:
            if (i1[0] > p1[0]) and (i1[0] < p2[0]):
                if p1[1] > p2[1]:
                    if (i1[1] < p1[1]) and (i1[1] > p2[1]):
                        print("Intersection")
                if p1[1] < p2[1]:
                    if (i1[1] > p1[1]) and (i1[1] < p2[1]):
                        print("Intersection")

#Si el punto 1 es mayor al 2 en X
#Si el punto iX esta entre 2 y 1 en X
#Si el punto 1 es mayor al 2 en Y
#Si el punto iy esta entre 2 y 1

#Si el punto 2 es mayo al 2 en Y
#Si el punto iy esta entre 1 y 2

#Si el punto 2 es mayor al 1 en X
#Si el punto iX esta entre 1 y 2 en X
#Si el punto 1 es mayor al 2 en Y
#Si el punto iy esta entre 2 y 1

#Si el punto 2 es mavo al 2 en Y

```

Share Follow

answered Dec 4, 2019 at 2:24



Jorge Eduardo G  
21 ● 1

This Java Function returns a DVec2 Object. It takes a [DVec2](#) for the center of the circle, the radius of the circle, and a Line.

1

```

public static DVec2 Circline(DVec2 C, double r, Line line)
{
    DVec2 A = line.p1;
    DVec2 B = line.p2;
    DVec2 P;
    DVec2 AC = new DVec2( C );
    AC.sub(A);
    DVec2 AB = new DVec2( B );
    AB.sub(A);
    double ab2 = AB.dot(AB);
    double acab = AC.dot(AB);
    double t = acab / ab2;

    if (t < 0.0)
        t = 0.0;
    else if (t > 1.0)
        t = 1.0;

    //P = A + t * AB;
    P = new DVec2( AB );
    P.mul( t );
    P.add( A );

    DVec2 H = new DVec2( P );
    H.sub( C );
    double h2 = H.dot(H);
    double r2 = r * r;

    if(h2 > r2)
        return null;
    else
        return P;
}

```

Share Follow

answered Jul 19, 2009 at 11:12



pahlevan  
43 ● 7

**Join Stack Overflow** to find the best answer to your technical question, help others answer theirs.

[Sign up with email](#)
[Sign up with Google](#)
[Sign up with GitHub](#)
[Sign up with Facebook](#)


1 plane XY



```

fn projectPoint p1 p2 p3 = --project p1 perpendicular to the line p2-p3
(
  local v= normalize (p3-p2)
  local p= (p1-p2)
  p2+((dot v p)*v)
)
fn findIntersectionLineCircle CircleCenter CircleRadius LineP1 LineP2=
(
  pp=projectPoint CircleCenter LineP1 LineP2
  sideA=distance pp CircleCenter
  --use pythagoras to solve the third side
  sideB=sqrt(CircleRadius^2-sideA^2) -- this will return NaN if they don't intersect
  IntersectV=normalize (pp-CircleCenter)
  perpV=[IntersectV.y,-IntersectV.x,IntersectV.z]
  --project the point to both sides to find the solutions
  solution1=pp+(sideB*perpV)
  solution2=pp-(sideB*perpV)
  return #(solution1,solution2)
)

```

Share Follow

answered Jul 24, 2018 at 19:53

martin  
11 ● 2

Here is my solution in TypeScript, following the idea that @Mizipzor suggested (using projection):

1



```

/**
 * Determines whether a line segment defined by a start and end point intersects
 * with a sphere defined by a center point and a radius
 * @param a the start point of the line segment
 * @param b the end point of the line segment
 * @param c the center point of the sphere
 * @param r the radius of the sphere
 */
export function lineSphereIntersects(
  a: IPoint,
  b: IPoint,
  c: IPoint,
  r: number
): boolean {
  // find the three sides of the triangle formed by the three points
  const ab: number = distance(a, b);
  const ac: number = distance(a, c);
  const bc: number = distance(b, c);

  // check to see if either ends of the line segment are inside of the sphere
  if (ac < r || bc < r) {
    return true;
  }

  // find the angle between the line segment and the center of the sphere
  const numerator: number = Math.pow(ac, 2) + Math.pow(ab, 2) - Math.pow(bc, 2);
  const denominator: number = 2 * ac * ab;
  const cab: number = Math.acos(numerator / denominator);

  // find the distance from the center of the sphere and the line segment
  const cd: number = Math.sin(cab) * ac;

  // if the radius is at least as long as the distance between the center and the
  // line
  if (r >= cd) {
    // find the distance between the line start and the point on the line closest
    // to
    // the center of the sphere
    const ad: number = Math.cos(cab) * ac;
    // intersection occurs when the point on the line closest to the sphere

```

Join Stack Overflow to find the best answer to your technical question, help others answer theirs.

Sign up with email

Sign up with Google

Sign up with GitHub

Sign up with Facebook





```

}

export function distance(a: IPoint, b: IPoint): number {
  return Math.sqrt(
    Math.pow(b.z - a.z, 2) + Math.pow(b.y - a.y, 2) + Math.pow(b.x - a.x, 2)
  );
}

export interface IPoint {
  x: number;
  y: number;
  z: number;
}

```

Share Follow

edited Apr 16, 2019 at 5:52

answered Apr 16, 2019 at 5:35



Joe Skeen

1,669 • 16 • 17

Solution in python, based on @Joe Skeen

1

```

def check_line_segment_circle_intersection(line, point, radius):
    """ Checks whether a point intersects with a line defined by two points.

    A `point` is list with two values: [2, 3]

    A `line` is list with two points: [point1, point2]

    """
    line_distance = distance(line[0], line[1])
    distance_start_to_point = distance(line[0], point)
    distance_end_to_point = distance(line[1], point)

    if (distance_start_to_point <= radius or distance_end_to_point <= radius):
        return True

    # angle between line and point with law of cosines
    numerator = (math.pow(distance_start_to_point, 2)
                 + math.pow(line_distance, 2)
                 - math.pow(distance_end_to_point, 2))
    denominator = 2 * distance_start_to_point * line_distance
    ratio = numerator / denominator
    ratio = ratio if ratio <= 1 else 1 # To account for float errors
    ratio = ratio if ratio >= -1 else -1 # To account for float errors
    angle = math.acos(ratio)

    # distance from the point to the line with sin projection
    distance_line_to_point = math.sin(angle) * distance_start_to_point

    if distance_line_to_point <= radius:
        point_projection_in_line = math.cos(angle) * distance_start_to_point
        # Intersection occurs when the point projection in the line is less
        # than the line distance and positive
        return point_projection_in_line <= line_distance and point_projection_in_line >= 0
    return False

def distance(point1, point2):
    return math.sqrt(
        math.pow(point1[1] - point2[1], 2) +

```

Share Follow

answered Apr 20, 2020 at 20:11



Ian Douglas

21 • 2

Maybe there is another way to solve this problem using rotation of coordinate system.

1

Normally, if one segment is horizontal or vertical, which means parallel to x or y axis, it's quite easy to solve the intersection point

Join Stack Overflow to find the best answer to your technical question, help others answer theirs.

Sign up with email

Sign up with Google

Sign up with GitHub

Sign up with Facebook



Let's take an example of circle  $x^2+y^2=1$  and segment  $P_1-P_2$  with  $P_1(-1.5,0.5)$  and  $P_2(-0.5,-0.5)$  in x-y system. And the following equations to remind you of the rotation principles, where  $\theta$  is the angle anticlockwise, x'-y' is the system after rotation :

$$x' = x \cdot \cos(\theta) + y \cdot \sin(\theta)$$

$$y' = -x \cdot \sin(\theta) + y \cdot \cos(\theta)$$

and inversely

$$x = x' \cdot \cos(\theta) - y' \cdot \sin(\theta)$$

$$y = x' \cdot \sin(\theta) + y' \cdot \cos(\theta)$$

Considering the segment  $P_1-P_2$  direction ( $45^\circ$  in terms of -x), we could take  $\theta=45^\circ$ . Taking the second equations of rotation into circle's equation in x-y system :  $x^2+y^2=1$  and after simple operations we get the 'same' equation in x'-y' system :

$$x'^2+y'^2=1.$$

Segment endpoints become in x'-y' system using the first equations of rotation  $\Rightarrow P_1(-\sqrt{2}/2, \sqrt{2}/2), P_2(-\sqrt{2}/2, 0)$ .

Assuming the intersection as P. We have in x'-y'  $P_x = -\sqrt{2}/2$ . Using the new equation of circle, we get  $P_y = +\sqrt{2}/2$ . Converting P into original x-y system, we get finally  $P(-1,0)$ .

To implement this numerically, we could firstly have a look at segment's direction : horizontal, vertical or not. If it belongs to the two first cases, it's simple like I said. If the last case, apply the algorithms above.

To judge if there is intersection, we could compare the solution with the endpoints coordinates, to see whether there is one root between them.

I believe this method could be also applied to other curves as long as we have its equation. The only weakness is that we should solve the equation in x'-y' system for the other coordinate, which might be difficult.

Share Follow

answered Nov 13, 2020 at 14:39



Here is a solution written in golang. The method is similar to some other answers posted here, but not quite the same. It is easy to implement, and has been tested. Here are the steps:

0

1. Translate coordinates so that the circle is at the origin.
2. Express the line segment as parametrized functions of t for both the x and y coordinates. If t is 0, the function's values are one end point of the segment, and if t is 1, the function's values are the other end point.
3. Solve, if possible, the quadratic equation resulting from constraining values of t that produce x, y coordinates with distances from the origin equal to the circle's radius.
4. Throw out solutions where t is  $< 0$  or  $> 1$  ( $\leq 0$  or  $\geq 1$  for an open segment). Those points are not contained in the segment.
5. Translate back to original coordinates.

The values for A, B, and C for the quadratic are derived here, where (n-et) and (m-dt) are the equations for the line's x and y coordinates, respectively. r is the radius of the circle.

**Join Stack Overflow** to find the best answer to your technical question, help others answer theirs.

Sign up with email

Sign up with Google

Sign up with GitHub

Sign up with Facebook



Therefore  $A = ee + dd$ ,  $B = -2(en + dm)$ , and  $C = nn + mm - rr$ .

Here is the go lang code for the function:

```
package geom

import (
    "math"
)

// SegmentCircleIntersection return points of intersection between a circle and
// a line segment. The Boolean intersects returns true if one or
// more solutions exist. If only one solution exists,
// x1 == x2 and y1 == y2.
// s1x and s1y are coordinates for one end point of the segment, and
// s2x and s2y are coordinates for the other end of the segment.
// cx and cy are the coordinates of the center of the circle and
// r is the radius of the circle.
func SegmentCircleIntersection(s1x, s1y, s2x, s2y, cx, cy, r float64) (x1, y1, x2, y2 float64, intersects bool) {
    // (n-et) and (m-dt) are expressions for the x and y coordinates
    // of a parameterized line in coordinates whose origin is the
    // center of the circle.
    // When t = 0, (n-et) == s1x - cx and (m-dt) == s1y - cy
    // When t = 1, (n-et) == s2x - cx and (m-dt) == s2y - cy.
    n := s2x - cx
    m := s2y - cy

    e := s2x - s1x
    d := s2y - s1y

    // lineFunc checks if the t parameter is in the segment and if so
    // calculates the line point in the unshifted coordinates (adds back
    // cx and cy.
    lineFunc := func(t float64) (x, y float64, inBounds bool) {
        inBounds = t >= 0 && t <= 1 // Check bounds on closed segment
        // To check bounds for an open segment use t > 0 && t < 1
        if inBounds { // Calc coords for point in segment
            x = n - e*t + cx
            y = m - d*t + cy
        }
        return
    }

    intersects = false
    x1, y1, x2, y2 = lineFunc(0), lineFunc(0), lineFunc(1), lineFunc(1)
    return
}
```

I tested it with this function, which confirms that solution points are within the line segment and on the circle. It makes a test segment and sweeps it around the given circle:

```
package geom_test

import (
    "testing"

    . "*/put your package path here*"
)

func CheckEpsilon(t *testing.T, v, epsilon float64, message string) {
    if v > epsilon || v < -epsilon {
        t.Error(message, v, epsilon)
        t.FailNow()
    }
}


func TestSegmentCircleIntersection(t *testing.T) {
    epsilon := 1e-10 // Something smallish
    x1, y1 := 5.0, 2.0 // segment end point 1
    x2, y2 := 50.0, 30.0 // segment end point 2
    cx, cy := 100.0, 90.0 // center of circle
    r := 80.0


    segx, segy := x2-x1, y2-y1

    testCnt, solutionCnt := 0, 0
```

**Join Stack Overflow** to find the best answer to your technical question, help others answer theirs.

[Sign up with email](#)

 [Sign up with Google](#)

 [Sign up with GitHub](#)

 [Sign up with Facebook](#)



```
s1y, s2y := y1+float64(j), y2+float64(j)

sclx, scly := s1x-cx, s1y-cy
seg1Inside := sclx*sclx+sclx*sclx < r*r
sc2x, sc2y := s2x-cx, s2y-cy
seg2Inside := sc2x*sc2x+sc2y*sc2y < r*r
```

Here is the output of the test:

```
=== RUN   TestSegmentCircleIntersection
--- PASS: TestSegmentCircleIntersection (0.00s)
geom_test.go:105: Tested 40000 examples and found 7343 solutions.
```

Finally, the method is easily extendable to the case of a ray starting at one point, going through the other and extending to infinity, by only testing if  $t > 0$  or  $t < 1$  but not both.

Share Follow

edited Mar 21, 2018 at 5:47

answered Mar 20, 2018 at 23:26



Steller  
31 ● 2

Another solution, first considering the case where you don't care about collision location. Note that this particular function is built assuming vector input for  $x_B$  and  $y_B$  but can easily be modified if that is not the case. Variable names are defined at the start of the function

```
#Line segment points (A0, Af) defined by xA0, yA0, xAf, yAf; circle center denoted by xB, yB; rB=radius of circle, rA
= radius of point (set to zero for your application)
def staticCollision_f(xA0, yA0, xAf, yAf, rA, xB, yB, rB): #note potential speed up here by casting all variables to
same type and/or using Cython

    #Build equations of a line for linear agents (convert  $y = mx + b$  to  $ax + by + c = 0$  means that  $a = -m$ ,  $b = 1$ ,  $c =$ 
    -b
    m_v = (yAf - yA0) / (xAf - xA0)
    b_v = yAf - m_v * xAf
    rEff = rA + rB #radii are added since we are considering the agent path as a thin line

    #Check if points (circles) are within line segment (find center of line segment and check if circle is within
    radius of this point)
    segmentMask = np.sqrt( (yB - (yA0+yAf)/2)**2 + (xB - (xA0+xAf)/2)**2 ) < np.sqrt( (yAf - yA0)**2 + (xAf - xA0)**2
    ) / 2 + rEff

    #Calculate perpendicular distance between line and a point
    dist_v = np.abs(-m_v * xB + yB - b_v) / np.sqrt(m_v**2 + 1)
    collisionMask = (dist_v < rEff) & segmentMask

    #return True if collision is detected
    return collisionMask, collisionMask.any()
```

If you need the location of the collision, you can use the approach detailed on this site, and set the velocity of one of the agents to zero. This approach works well with vector inputs as well: <http://twobitcoder.blogspot.com/2010/04/circle-collision-detection.html>

Share Follow

answered Nov 14, 2020 at 4:36



DivideByZero  
119 ● 1 ● 11

Although I think using [line-circle intersection](#), then checking if intersection points are between the endpoints is better and probably cheaper, I would like to add this more visual solution.

I like to think about the problem as a "point in sausage problem" which should work in any number of dimensions without

**Join Stack Overflow** to find the best answer to your technical question, help others answer theirs.

Sign up with email

Sign up with Google

Sign up with GitHub

Sign up with Facebook



(I use "less than", but "less than or equal to" could also be used depending on what we're testing.)

1. Make sure the Circle\_Point is less than Radius distance from the infinite line. (Use favorite method here).
  2. Calculate distance from both Segment\_Points to the Circle\_Point.
  3. Test if the **bigger** Circle\_Point-Segment\_Point distance is less than  $\sqrt{\text{Segment\_Length}^2 + \text{Radius}^2}$ . (This is the distance from a segment-point to a theoretical point, which is radius-distance from the other segment-point **and** the infinite-line (right-angle). See image.)
- 3T. If true: Circle\_Point is inside sausage.
  - 3F. If false: If the **smaller** Circle\_Point-Segment\_Point distance is less than Radius, then Circle\_Point is inside sausage.

[Image: Thickest line segment is the selected segment, there is no example circle. A bit crude, some pixels misalign slightly.](#)

```
function boolean pointInSausage(sp1,sp2,r,c) {
  if ( !(pointLineDist(c,sp1,sp2) < r) ) {
    return false;
  }
  double a = dist(sp1,c);
  double b = dist(sp2,c);
  double l;
  double s;
  if (a>b) {
    l = a;
    s = b;
  } else {
    l = b;
    s = a;
  }
  double segLength = dist(sp1,sp2);
  if ( l < sqrt(segLength*segLength+r*r) ) {
    return true;
  }
  return s < r;
}
```

Tell if any issues are found and I'll edit or retract.

Share Follow

edited Mar 22 at 16:02

answered Mar 19 at 17:13





Mater


1 ● 2

**Join Stack Overflow** to find the best answer to your technical question, help others answer theirs.

Sign up with email

 Sign up with Google

 Sign up with GitHub

 Sign up with Facebook

