

How to find the intersection point between a line and a rectangle?

Asked 15 years, 2 months ago Modified 8 months ago

Viewed 86k times



70

I have a line that goes from points A to B; I have (x,y) of both points. I also have a rectangle that's centered at B and the width and height of the rectangle.



I need to find the point in the line that intersects the rectangle. Is there a formula that gives me the (x,y) of that point?



algorithm

geometry

line

intersection

Share

Improve this question

Follow

edited Mar 5, 2016 at 18:33



Cimballi

11.4k ● 1 ● 42 ● 74

asked Oct 18, 2009 at 17:39



John Petterson

703 ● 1 ● 5 ● 4

3 Can we assume the rectangle is aligned with the axes and not tilted? – [Grandpa](#) Oct 18, 2009 at 18:00

21 To those voting to close: traditionally we have allowed these kind of math questions as being close enough to

programming problems and common enough in both real life programming and programming education. The thing I would look for on this questions is the real possibility that it is a duplicate. – [dmckee](#) --- [ex-moderator kitten](#) Oct 19, 2009 at 0:00

14 Answers

Sorted by:

Highest score (default)



34



```
/**
 * Finds the intersection point between
 *     * the rectangle
 *     with parallel sides to the x and y axes
 *     * the half-line pointing towards (x,y)
 *     originating from the middle of the rectangle
 *
 * Note: the function works given min[XY] <= max[XY]
 * even though minY may not be the "top" of the rectangle
 * because the coordinate system is flipped.
 * Note: if the input is inside the rectangle,
 * the line segment wouldn't have an intersection
 * but the projected half-line does.
 * Warning: passing in the middle of the rectangle w
itself
 *     there are infinitely many half-lines pro
 *     so let's just shortcut to midpoint (GIGO
 *
 * @param x:Number x coordinate of point to build th
 * @param y:Number y coordinate of point to build th
 * @param minX:Number the "left" side of the rectangle
 * @param minY:Number the "top" side of the rectangle
 * @param maxX:Number the "right" side of the rectangle
 * @param maxY:Number the "bottom" side of the rectangle
 * @param validate:boolean (optional) whether to tre
error
 * @return an object with x and y members for the in
 * @throws if validate == true and (x,y) is inside t
 * @author TwiStErRob
 * @licence Dual CC0/WTFPL/Unlicence, whatever float
```

```

* @see <a href="http://stackoverflow.com/a/31254199
* @see <a href="http://stackoverflow.com/a/18292964
*/
function pointOnRect(x, y, minX, minY, maxX, maxY, v
    //assert minX <= maxX;
    //assert minY <= maxY;
    if (validate && (minX < x && x < maxX) && (m
        throw "Point " + [x,y] + "cannot be
            + "the rectangle: " + [minX, min
".";

    var midX = (minX + maxX) / 2;
    var midY = (minY + maxY) / 2;
    // if (midX - x == 0) -> m == ±Inf -> minYx/
    ±Inf = ±0)
    var m = (midY - y) / (midX - x);

    if (x <= midX) { // check "left" side
        var minXy = m * (minX - x) + y;
        if (minY <= minXy && minXy <= maxY)
            return {x: minX, y: minXy};
    }

    if (x >= midX) { // check "right" side
        var maxXy = m * (maxX - x) + y;
        if (minY <= maxXy && maxXy <= maxY)
            return {x: maxX, y: maxXy};
    }

    if (y <= midY) { // check "top" side
        var minYx = (minY - y) / m + x;
        if (minX <= minYx && minYx <= maxX)
            return {x: minYx, y: minY};
    }

    if (y >= midY) { // check "bottom" side
        var maxYx = (maxY - y) / m + x;
        if (minX <= maxYx && maxYx <= maxX)
            return {x: maxYx, y: maxY};
    }

    // edge case when finding midpoint intersect
    if (x === midX && y === midY) return {x: x,

    // Should never happen :) If it does, please

```

```

        throw "Cannot find intersection for " + [x,y]
          + " inside rectangle " + [minX, minY] +
    }

(function tests() {
    var left = 100, right = 200, top = 50, bottom = 150;
    really
    var hMiddle = (left + right) / 2, vMiddle = 100;
    function intersectTestRect(x, y) { return pointOnRect(
        x, y, left, top, right, bottom, true); }
    function intersectTestRectNoValidation(x, y) { return pointOnRect(
        x, y, left, top, right, bottom, false); }
    function checkTestRect(x, y) { return function() {
        return intersectTestRect(x, y); }; };
    QUnit.test("intersects left side", function() {
        var leftOfRect = 0, closerLeftOfRect = 0;
        assert.deepEqual(intersectTestRect(left, top,
        y:75}, "point above top");
        assert.deepEqual(intersectTestRect(left, top,
        {x:left, y:80}, "point in line with top");
        assert.deepEqual(intersectTestRect(left, top,
        y:90}, "point above middle");
        assert.deepEqual(intersectTestRect(left, top,
        {x:left, y:100}, "point exact middle");
        assert.deepEqual(intersectTestRect(left, top,
        y:110}, "point below middle");
        assert.deepEqual(intersectTestRect(left, top,
        {x:left, y:120}, "point in line with bottom");
        assert.deepEqual(intersectTestRect(left, top,
        y:125}, "point below bottom");
    });
    QUnit.test("intersects right side", function() {
        var rightOfRect = 300, closerRightOfRect = 300;
        assert.deepEqual(intersectTestRect(right, top,
        y:75}, "point above top");
        assert.deepEqual(intersectTestRect(right, top,
        {x:right, y:75}, "point in line with top");
        assert.deepEqual(intersectTestRect(right, top,
        y:90}, "point above middle");
        assert.deepEqual(intersectTestRect(right, top,
        {x:right, y:100}, "point exact middle");
        assert.deepEqual(intersectTestRect(right, top,
        y:110}, "point below middle");
        assert.deepEqual(intersectTestRect(right, top,
        {x:right, y:120}, "point in line with bottom");
    });
});

```

```

{x:right, y:125}, "point in line with bottom");
    assert.deepEqual(intersectTestRect(r
y:125}, "point below bottom");
});
QUnit.test("intersects top side", function(a
    var aboveRect = 0;
    assert.deepEqual(intersectTestRect(8
y:top}, "point left of left");
    assert.deepEqual(intersectTestRect(l
y:top}, "point in line with left");
    assert.deepEqual(intersectTestRect(1
y:top}, "point left of middle");
    assert.deepEqual(intersectTestRect(h
y:top}, "point exact middle");
    assert.deepEqual(intersectTestRect(1
y:top}, "point right of middle");
    assert.deepEqual(intersectTestRect(r
y:top}, "point in line with right");
    assert.deepEqual(intersectTestRect(2
y:top}, "point right of right");
});
QUnit.test("intersects bottom side", functio
    var belowRect = 200;
    assert.deepEqual(intersectTestRect(8
y:bottom}, "point left of left");
    assert.deepEqual(intersectTestRect(l
y:bottom}, "point in line with left");
    assert.deepEqual(intersectTestRect(1
y:bottom}, "point left of middle");
    assert.deepEqual(intersectTestRect(h
y:bottom}, "point exact middle");
    assert.deepEqual(intersectTestRect(1
y:bottom}, "point right of middle");
    assert.deepEqual(intersectTestRect(r
y:bottom}, "point in line with right");
    assert.deepEqual(intersectTestRect(2
y:bottom}, "point right of right");
});
QUnit.test("intersects a corner", function(a
    assert.deepEqual(intersectTestRect(l
y:top}, "intersection line aligned with top-left cor
    assert.deepEqual(intersectTestRect(r
y:top}, "intersection line aligned with top-right co
    assert.deepEqual(intersectTestRect(l

```

```

{x:left, y:bottom}, "intersection line aligned with
    assert.deepEqual(intersectTestRect(r
{x:right, y:bottom}, "intersection line aligned with
    });
    QUnit.test("on the corners", function(assert)
        assert.deepEqual(intersectTestRect(l
"top-left corner");
        assert.deepEqual(intersectTestRect(r
y:top}, "top-right corner");
        assert.deepEqual(intersectTestRect(r
y:bottom}, "bottom-right corner");
        assert.deepEqual(intersectTestRect(l
y:bottom}, "bottom-left corner");
    });
    QUnit.test("on the edges", function(assert)
        assert.deepEqual(intersectTestRect(h
y:top}, "top edge");
        assert.deepEqual(intersectTestRect(r
y:vMiddle}, "right edge");
        assert.deepEqual(intersectTestRect(h
{x:hMiddle, y:bottom}, "bottom edge");
        assert.deepEqual(intersectTestRect(l
y:vMiddle}, "left edge");
    });
    QUnit.test("validates inputs", function(asse
        assert.throws(checkTestRect(hMiddle,
inside/, "center");
        assert.throws(checkTestRect(hMiddle-
inside/, "top left of center");
        assert.throws(checkTestRect(hMiddle-
inside/, "left of center");
        assert.throws(checkTestRect(hMiddle-
inside/, "bottom left of center");
        assert.throws(checkTestRect(hMiddle,
inside/, "above center");
        assert.throws(checkTestRect(hMiddle,
inside/, "center");
        assert.throws(checkTestRect(hMiddle,
inside/, "below center");
        assert.throws(checkTestRect(hMiddle+
inside/, "top right of center");
        assert.throws(checkTestRect(hMiddle+
inside/, "right of center");
        assert.throws(checkTestRect(hMiddle+

```

```

inside/, "bottom right of center");
        assert.throws(checkTestRect(left+10,
inside/, "right of left edge");
        assert.throws(checkTestRect(left+10,
inside/, "right of left edge");
        assert.throws(checkTestRect(left+10,
inside/, "right of left edge");
        assert.throws(checkTestRect(right-10
inside/, "left of right edge");
        assert.throws(checkTestRect(right-10
inside/, "left of right edge");
        assert.throws(checkTestRect(right-10
inside/, "left of right edge");
        assert.throws(checkTestRect(hMiddle-
inside/, "below top edge");
        assert.throws(checkTestRect(hMiddle,
inside/, "below top edge");
        assert.throws(checkTestRect(hMiddle+
inside/, "below top edge");
        assert.throws(checkTestRect(hMiddle-
inside/, "above bottom edge");
        assert.throws(checkTestRect(hMiddle,
inside/, "above bottom edge");
        assert.throws(checkTestRect(hMiddle+
inside/, "above bottom edge");
    });
    QUnit.test("doesn't validate inputs", function() {
        assert.deepEqual(intersectTestRectNo
vMiddle-10), {x:left, y:top}, "top left of center");
        assert.deepEqual(intersectTestRectNo
vMiddle), {x:left, y:vMiddle}, "left of center");
        assert.deepEqual(intersectTestRectNo
vMiddle+10), {x:left, y:bottom}, "bottom left of cen
        assert.deepEqual(intersectTestRectNo
vMiddle-10), {x:hMiddle, y:top}, "above center");
        assert.deepEqual(intersectTestRectNo
vMiddle), {x:hMiddle, y:vMiddle}, "center");
        assert.deepEqual(intersectTestRectNo
vMiddle+10), {x:hMiddle, y:bottom}, "below center");
        assert.deepEqual(intersectTestRectNo
vMiddle-10), {x:right, y:top}, "top right of center"
        assert.deepEqual(intersectTestRectNo
vMiddle), {x:right, y:vMiddle}, "right of center");
        assert.deepEqual(intersectTestRectNo

```

```
vMiddle+10), {x:right, y:bottom}, "bottom right of c  
});  
})();
```

```
<link href="https://code.jquery.com/qunit/qunit-2.3.  
<script src="https://code.jquery.com/qunit/qunit-2.3  
<div id="qunit"></div>
```



Run code snippet

[Expand snippet](#)

Share Improve this answer

edited Mar 29, 2018 at 22:42

Follow

answered Jul 6, 2015 at 19:37



TWiStErRob

46.4k ● 28 ● 180 ● 271

Excellent answer. I just shamelessly stole your function for [this question](#) and worked like a charm. – [Mark](#) Jan 6, 2017 at 18:43

3 @Mark Attribution is never shameless, and way better than a link-only answer ;) – [TWiStErRob](#) Jan 6, 2017 at 23:00

That's neat, it's what I need ;) – [canbax](#) Nov 30, 2021 at 8:14



31

The point A is always outside of the rectangle
and the point B is always at the center of the
rectangle



Assuming the rectangle is axis-aligned, this makes things pretty simple:



The slope of the line is $s = (A_y - B_y)/(A_x - B_x)$.



- If $-h/2 \leq s * w/2 \leq h/2$ then the line intersects:
 - The right edge if $A_x > B_x$
 - The left edge if $A_x < B_x$.
- If $-w/2 \leq (h/2)/s \leq w/2$ then the line intersects:
 - The top edge if $A_y > B_y$
 - The bottom edge if $A_y < B_y$.

Once you know the edge it intersects you know one coordinate: $x = B_x \pm w/2$ or $y = B_y \pm h/2$ depending on which edge you hit. The other coordinate is given by $y = B_y + s * w/2$ or $x = B_x + (h/2)/s$.

Share Improve this answer

edited Oct 26, 2016 at 11:39

Follow

answered Oct 18, 2009 at 18:18




Joren


14.7k ● 3 ● 52 ● 54


-
- 4 Thanks Joren, I've made a fiddle of this algorithm: jsfiddle.net/524ctnfh It seems right-left and top-bottom edges are swapped-around, so it should be: *right*: $A_x < B_x$; *left*: $A_x > B_x$; *top*: $A_y < B_y$; *bottom*: $A_y > B_y$; – Johnner Feb 3, 2015 at 9:33
-

- 3 Sorry, I've made some mistakes in the script, here is fixed version: jsfiddle.net/524ctnfh/1 – Johnner Feb 4, 2015 at 16:53

An implementation of a similar one in JavaScript:

stackoverflow.com/a/31254199/253468 – TWiStErRob Jul 6, 2015 at 19:43 

@Johnner: Assuming a standard coordinate system where x increases left-to-right, then $A_x < B_x$ definitely implies that point A is to the *left* of the rectangle with center B (and $A_x > B_x \Rightarrow$ to the right). Top-bottom could indeed be flipped depending on your coordinate system convention. I'm using a right-handed coordinate system where y increases bottom-to-top (as is standard in mathematics), while you're probably thinking of a left-handed coordinate system where y increases top-to-bottom (as is standard in graphics & UI programming). – Joren Jul 7, 2015 at 13:32 

-
- 1 This answer is incomplete. OP says he "need[s] to find the *point* in the line that intersects the rectangle" - not just which side of the rectangle it intersects. – cp.engr Oct 25, 2016 at 21:35 



19



You might want to check out [Graphics Gems](#) - this is a classic set of routines for graphics and includes many of the algorithms required. Although it's in C and slightly dated the algorithms still sparkle and it should be trivial to transfer to other languages.

For your current problem the just create the four lines for the rectangle and see which intersect your given line.

Share Improve this answer

edited Jun 29, 2015 at 4:18

Follow



Ken Y-N

15k ● 21 ● 77 ● 119

answered Oct 18, 2009 at 18:31



peter.murray.rust

38k ● 46 ● 159 ● 224

4 This is too far from what the OP asked. – [TWiStErRob](#) Jul 6, 2015 at 18:37



11



Here is a solution in Java that returns true if a line segment (the first 4 parameters) intersects an axis aligned rectangle (the last 4 parameters). It would be trivial to return the intersection point instead of a boolean. It works by first checking if completely outside, else using the line equation $y=m*x+b$. We know the lines that make up the rectangle are axis aligned, so the checks are easy.

```
public boolean aabbContainsSegment (float x1, float y1
float minX, float minY, float maxX, float maxY) {
    // Completely outside.
    if ((x1 <= minX && x2 <= minX) || (y1 <= minY && y
    && x2 >= maxX) || (y1 >= maxY && y2 >= maxY))
        return false;

    float m = (y2 - y1) / (x2 - x1);

    float y = m * (minX - x1) + y1;
    if (y > minY && y < maxY) return true;

    y = m * (maxX - x1) + y1;
    if (y > minY && y < maxY) return true;

    float x = (minY - y1) / m + x1;
    if (x > minX && x < maxX) return true;
```

```
x = (maxY - y1) / m + x1;  
if (x > minX && x < maxX) return true;  
  
return false;  
}
```

It is possible to shortcut if the start or end of the segment is inside the rectangle, but probably it is better to just do the math, which will always return true if either or both segment ends are inside. If you want the shortcut anyway, insert the code below after the "completely outside" check.

```
// Start or end inside.  
if ((x1 > minX && x1 < maxX && y1 > minY && y1 < maxY)  
    maxX && y2 > minY && y2 < maxY)) return true;
```

Share Improve this answer

Follow

edited Jul 6, 2015 at 19:47



TWiStErRob

46.4k ● 28 ● 180 ● 271

answered Aug 17, 2013 at 20:20



NateS

5,876 ● 4 ● 54 ● 59

2 Great thanks!, this is what I was looking for. I moved it to javascript, here is the fiddle I used to test it jsfiddle.net/pjnovas/fPMG5 cheers! – [pjnovas](#) Sep 28, 2013 at 16:36

i can spot couple potential divide by zeros here – [gzmask](#) Feb 8, 2015 at 19:45

1 @gzmask It's true, but the method still appears to return the correct values for all inputs (in Java and JavaScript

$x/0=\text{Infinity}$ and $x/\text{Infinity}=0$). See [here](#). – NateS

Feb 8, 2015 at 23:21 

I added a specialized version of this with all "trivial" stuff and "shortcuts": stackoverflow.com/a/31254199/253468

– TWiStErRob Jul 6, 2015 at 19:41

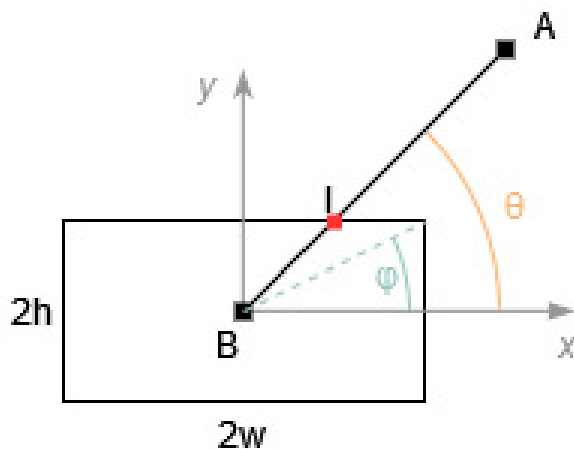
- 1 Warning: this returns false if the line crosses exactly the corner. jsfiddle.net/obgxhyku – Ti Hausmann Aug 27, 2021 at 21:50



8



Given the original question, I think that @ivanross answer is the most concise and clear so far, and I found myself using the same approach.



If we have a rectangle

- centered in B
- with sides parallel to x and y axes

we can use a bit of trigonometry to get:

- $\tan \varphi (\text{phi}) = h/w$

- $\tan \theta (\text{theta}) = (y_B - y_A) / (x_B - x_A)$

and some trivial math to get in which quadrant (of the x-y plane centered in B) the point A is.

finally we compare the angles and use the tangents to calculate the coordinates of the intersection point, applying again basic trigonometry principles.

```
/**
 * Finds the intersection point between
 *     * a rectangle centered in point B
 *     with sides parallel to the x and y axes
 *     * a line passing through points A and B (the
 *
 * @param width: rectangle width
 * @param height: rectangle height
 * @param xB; rectangle center x coordinate
 * @param yB; rectangle center y coordinate
 * @param xA; point A x coordinate
 * @param yA; point A y coordinate
 * @author Federico Destefanis
 * @see <a href="https://stackoverflow.com/a/3125419
 */

function lineIntersectionOnRect(width, height, xB, y

var w = width / 2;
var h = height / 2;

var dx = xA - xB;
var dy = yA - yB;

//if A=B return B itself
if (dx == 0 && dy == 0) return {
    x: xB,
    y: yB
};

var tan_phi = h / w;
```

```

var tan_theta = Math.abs(dy / dx);

//tell me in which quadrant the A point is
var qx = Math.sign(dx);
var qy = Math.sign(dy);

if (tan_theta > tan_phi) {
    xI = xB + (h / tan_theta) * qx;
    yI = yB + h * qy;
} else {
    xI = xB + w * qx;
    yI = yB + w * tan_theta * qy;
}

return {
    x: xI,
    y: yI
};
}

var coords = lineIntersectionOnRect(6, 4, 0, 0, 1, 0
console.log(coords);

```



Run code snippet



[Expand snippet](#)

Share Improve this answer

edited Jan 4, 2022 at 17:52

Follow

answered Jan 4, 2022 at 15:15



Federico Destefanis

1,058 ● 2 ● 17 ● 31

2 It works fine. Intersection point is correct – [Shashi3456643](#)
Jan 5, 2022 at 7:32



Here is a solution that works for me. I assume that the rect is aligned to the axes.

6

Data:



```
// Center of the Rectangle
let Cx: number
let Cy: number
// Width
let w: number
// Height
let h: number

// Other Point
let Ax: number
let Ay: number
```

Now translate point A by the center of the rectangle so the rect is centered in O(0,0) and consider the problem in the first quarter (i.e. $x > 0$ and $y > 0$).

```
// Coordinates Translated
let Px = Math.abs(Ax - Cx)
let Py = Math.abs(Ay - Cy)

// Slope of line from Point P to Center
let Pm = Py / Px

// Slope of rectangle Diagonal
let Rm = h / w

// If the point is inside the rectangle, return the ce
```



```

let res: [number, number] = [0, 0]

// Check if the point is inside and if so do not calcu
if (!(Px < w / 2 && Py < h / 2)) {

    // Calculate point in first quarter: Px >= 0 && Py
    if (Pm <= Rm) {
        res[0] = w / 2
        res[1] = (w * Pm) / 2
    } else {
        res[0] = h / (Pm * 2)
        res[1] = h / 2
    }

    // Set original sign
    if (Ax - Cx < 0) res[0] *= -1
    if (Ay - Cy < 0) res[1] *= -1
}

// Translate back
return [res[0] + Cx, res[1] + Cy]

```

Share Improve this answer

answered Apr 25, 2018 at 14:37

Follow



ivanross

103 ● 1 ● 4



5



I am not a math fan nor do I particularly enjoy translating stuff from other languages if others have already done so, so whenever I complete a boring translation task, I add it to the article that led me to the code. To prevent anyone doing double work.



So if you want to have this intersection code in C#, have a look here



<http://dotnetbyexample.blogspot.nl/2013/09/utility-classes-to-check-if-lines-andor.html>



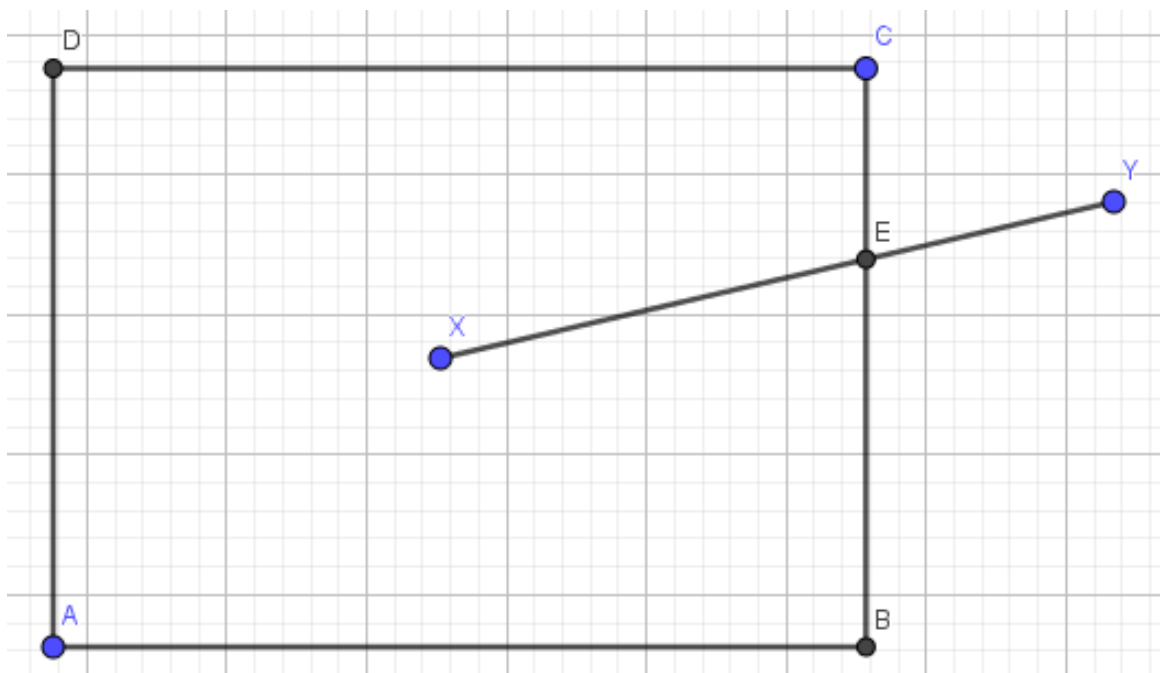
Let's make some assumptions :

4

Points A and C are given, such that they define a rectangle $ABCD$ aligned with the traditional axes. Assume that A is the bottom-left corner, and C is the top-right (*i.e.* $x_A < x_C$ and $y_A < y_C$).

Assume that X and Y are two points given such that X lies *inside* the rectangle (ie $x_A < x_X < x_C$ && $y_A < y_X < y_C$) and Y lies *outside* (*i.e.* $\text{not}(x_A < x_Y < x_C \text{ \&\& } y_A < y_Y < y_C)$).

This allows us to define a **unique** intersection point E between the segment $[X, Y]$ and the rectangle $\partial ABCD$.



The trick is to look for a certain $0 < t < 1$ such that $t \cdot Y + (1-t) \cdot X$ is on the rectangle $\partial ABCD$. By re-writing the condition $\Gamma(t) \in ABCD$ as :

$$(x_Y - x_X) \cdot t \in [x_A - x_X, x_C - x_X] \text{ and } (y_Y - y_X) \cdot t \in [y_A - y_X, y_C - y_X],$$

it is now possible to unwind all the scenarios. This yields :

```
var t = 0;

if(xY == xX) {
    t = max((yA - yX)/(yY - yX), (yC - yX)/(yY - yX));
} else {
    if(yY == yX) {
        t = max((xA - xX)/(xY - xX), (xC - xX)/(xY - xX));
    } else {
        if(xY > xX) {
            if(yY > yX) {
                t = min((xC - xX)/(xY - xX), (yC - yX)/(yY - yX));
            } else {
                t = min((xC - xX)/(xY - xX), (yA - yX)/(yY - yX));
            }
        } else {
            if(yY > yX) {
                t = min((xA - xX)/(xY - xX), (yC - yX)/(yY - yX));
            } else {
                t = min((xA - xX)/(xY - xX), (yA - yX)/(yY - yX));
            }
        }
    }
}
```

```
xE = t * xY + (1 - t) * xX;  
yE = t * yY + (1 - t) * yX;
```

Share Improve this answer

edited Jun 20, 2020 at 9:12

Follow



Community Bot

1 • 1

answered May 20, 2020 at 1:15



Anthony

223 • 3 • 14

There is an error I cannot track inside the `(xY > xX)`
– user1908746 May 21, 2021 at 7:12

-
- 1 @Lara wdyd by and error you "cannot track" ? Do you mean an error upon compilation, or an error regarding the result yielded ? Have you c/p'ed the code, or have you translated to your language of choice ? Are you sure your points are all in positions compatible with the assumptions I made to the problem ? – Anthony May 21, 2021 at 8:30

The code works when the line crosses above and below but not when the line crosses from the left or right of the rectangle. In that case, `yE` is correctly calculated but `xE` is not (it becomes displaced increasingly away). I cannot figure out why, i.e., cannot track down the error other than it is at that `if`. My mistake somehow, no doubt. Here is my implementation of your algorithm: pastebin.com/6xPnKMAB
– user1908746 May 21, 2021 at 22:02

There is a lot of duplication in this code, which makes it harder to understand. Subexpressions `"(xY - xX)"` and `"(yY - yX)"` seem to appear in every code path (or could easily be made to). I recommend assigning these to variables (e.g. `dx` and `dy`) before the main 'if' statements, and simplifying the code inside the 'if' statements to use these variables (also, `"if (xY == xX)"` becomes `"if (dx == 0)"`, etc.). Also, you would not

need the various "t =" assignments if you put the 'if' statements in a function and assigned t one time to be the function's result. That way, t could be 'const'. – [Some Guy](#)

Apr 15 at 20:04

@SomeGuy I don't think the issues you address are genuine issues. I gave a proof-of-concept with an explanation, and the code does the job of illustrating that approach. Anyhow, feel free to edit my answer with improvements if you feel like the need to. – [Anthony](#) Apr 16 at 5:46



I'll not give you a program to do that, but here is how you can do it:

3



- calculate the angle of the line
- calculate the angle of a line from the center of the rectangle to one of it's corners
- based on the angles determine on which side does the line intersect the rectangle
- calculate intersection between the side of the rectangle and the line



Share Improve this answer

Follow

answered Oct 18, 2009 at 17:52



[Lukáš Lalinský](#)

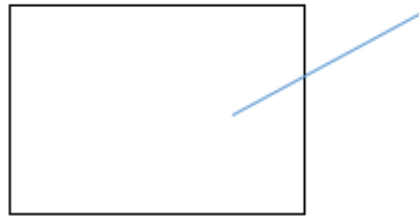
41.2k ● 6 ● 107 ● 128



3



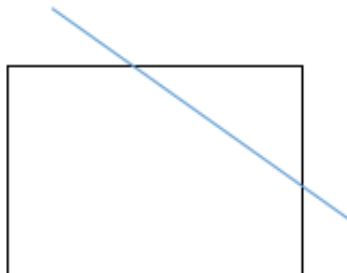
Line Intersection Possibilities



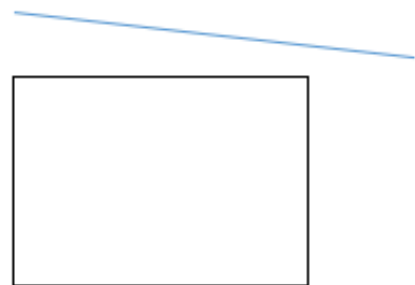
I). One point Inside and other point Outside the Rectangle



II). Both Points inside the Rectangle



III). Both Point Outside the Rectangle And intersecting



IV). Both Point Outside the Rectangle And not intersecting

Hope It works 100%

I am also had this same problem. So after two days of hard effort finally I created this method,

Main method,

```
enum Line
{
    // Inside the Rectangle so No Intersection
    Point(Both Entry Point and Exit Point will be
    Null)
    InsideTheRectangle,

    // One Point Inside the Rectangle another
    Point Outside the Rectangle. So it has only Entry
    Point
    Entry,
```

```

        // Both Point Outside the Rectangle but
        Intersecting. So It has both Entry and Exit Point
        EntryExit,

        // Both Point Outside the Rectangle and
        not Intersecting. So doesn't has both Entry and
        Exit Point
        NoIntersection
    }

    // Tuple<entryPoint, exitPoint, lineStatus>
    private Tuple<Point, Point, Line>
    GetIntersectionPoint(Point a, Point b, Rectangle
    rect)
    {
        if (IsWithinRectangle(a, rect) &&
        IsWithinRectangle(b, rect))
        {
            // Can't set null to Point that's why
            I am returning just empty object
            return new Tuple<Point, Point, Line>
            (new Point(), new Point(),
            Line.InsideTheRectangle);
        }
        else if (!IsWithinRectangle(a, rect) &&

```

Supporting methods,

```

    private Point
    GetEntryIntersectionPoint(Rectangle rect, Point a,
    Point b)
    {
        // For top line of the rectangle
        if (LineIntersectsLine(new Point(0, 0),
        new Point(rect.Width, 0), a, b))
        {
            return GetPointFromYValue(a, b, 0);
        }
        // For right side line of the rectangle
        else if (LineIntersectsLine(new
        Point(rect.Width, 0), new Point(rect.Width,
        rect.Height), a, b))
        {

```

```

        return GetPointFromXValue(a, b,
rect.Width);
    }
    // For bottom line of the rectangle
    else if (LineIntersectsLine(new Point(0,
rect.Height), new Point(rect.Width, rect.Height),
a, b))
    {
        return GetPointFromYValue(a, b,
rect.Height);
    }
    // For left side line of the rectangle
    else
    {
        return GetPointFromXValue(a, b, 0);
    }
}

public bool LineIntersectsRectangle(Point p1,
Point p2, Rectangle r)
{
    return LineIntersectsLine(p1, p2, new
Point(r.X, r.Y), new Point(r.X + r.Width, r.Y));
}

```

Share Improve this answer

edited Jun 5, 2021 at 5:30

Follow

answered May 9, 2021 at 17:45



Maari

51 ● 7

Note that the question specifically being asked here is only a special case of case (I) in your list (specifically when the endpoint of the line is at the center of the rectangle). It's possible that the rest of your code might be useful to other people reading this question who have different needs, however. – [Some Guy](#) Apr 15 at 20:12



2



Another option that you can consider especially if you are planning on testing many lines with the same rectangle is to transform your coordinate system to have the axes align with diagonals of the rectangle. Then since your line or ray starts at the center of the rectangle you can determine the angle then you can tell which segment it will intersect by the angle (i.e. $<90^\circ$ seg 1, $90^\circ < <180^\circ$ seg 2 etc...). Then of course you have to transform back to the original coordinate system

Although this seems like more work the transformation matrix and its inverse can be calculated once and then reused. This also extends to higher dimensional rectangles more easily where you would have to consider quadrants and intersections with faces in 3D and so on.

Share Improve this answer

answered Jan 20, 2016 at 7:21

Follow



Ivajlo Donev

195 ● 2 ● 10



1



I don't know if this is the best way, but what you could do is to figure out the proportion of the line that is inside the rectangle. You can get that from the width of the rectangle and the difference between the x coordinates of A and B (or height and y coordinates; based on the width and height you can check which case applies, and the other case will be on the extension of a side of the rectangle). When you have this, just take that proportion of the vector from B to A and you have your intersection point's coordinates.

Share Improve this answer

answered Oct 18, 2009 at 17:56

Follow



JaakkoK

8,367 ● 2 ● 35 ● 50



1



Here is a slightly verbose method that returns the intersection intervals between an (infinite) line and a rectangle using only basic math:

```
// Line2      - 2D line with origin (= offset from 0,0)
// Rectangle2 - 2D rectangle by min and max points
// Contacts    - Stores entry and exit times of a line
```

```
Contacts findContacts(const Line2 &line, const Rectang
Contacts contacts;
```

```
// If the line is not parallel to the Y axis, find o
// the limits of the rectangle horizontally
```

```
if(line.Direction.X != 0.0f) {
    float leftTouch = (rect.Min.X - line.Origin.X) / l
    float rightTouch = (rect.Max.X - line.Origin.X) /
    contacts.Entry = std::fmin(leftTouch, rightTouch);
    contacts.Exit = std::fmax(leftTouch, rightTouch);
} else if((line.Offset.X < rect.Min.X) || (line.Offs
return Contacts::None; // Rectangle missed by vert
}
```

```
// If the line is not parallel to the X axis, find o
// the limits of the rectangle vertically
```

```
if(line.Direction.Y != 0.0f) {
    float topTouch = (rectangle.Min.Y - line.Offset.Y)
    float bottomTouch = (rectangle.Max.Y - line.Offset
```

```
// If the line is parallel to the Y axis (and it g
// the rectangle), only the Y axis needs to be tak
```

```
if(line.Direction.X == 0.0f) {
    contacts.Entry = std::fmin(topTouch, bottomTouch
    contacts.Exit = std::fmax(topTouch, bottomTouch)
} else {
```

```

float verticalEntry = std::fmin(topTouch, bottom
float verticalExit = std::fmax(topTouch, bottomT

// If the line already left the rectangle on one
// on the other, it has missed the rectangle.
if((verticalExit < contacts.Entry) || (contacts.
    return Contacts::None;
}

// Restrict the intervals from the X axis of the
// the line is also within the limits of the rec
contacts.Entry = std::fmax(verticalEntry, contac
contacts.Exit = std::fmin(verticalExit, contacts
}
} else if((line.Offset.Y < rect.Min.Y) || (line.Offs
    return Contacts::None; // Rectangle missed by hori
}

return contacts;
}

```

This approach offers a high degree of numerical stability (the intervals are, in all cases, the result of a single subtraction and division) but involves some branching.

For a line segment (with start and end points), you'd need to provide the segment's start point as the origin and for the direction, `end - start`. Calculating the coordinates of the two intersections is as simple as `entryPoint = origin + direction * contacts.Entry` and `exitPoint = origin + direction * contacts.Exit`.

Share Improve this answer

Follow

edited Jul 6, 2015 at 19:46



TWiStErRob

46.4k ● 28 ● 180 ● 271

answered Mar 4, 2014 at 10:45



Cygon

9,610 ● 8 ● 45 ● 50



With a little math, you can solve this problem in a much easier fashion than most of these answers suggest.

1

Strategy:



1. Transform the line and rectangle so that the center of the rectangle is at the origin (0,0).
2. Scale both line and rectangle by the width/2 and height/2 of the rectangle (called x_{radius} and y_{radius} in my code sample below), so the problem becomes essentially "find the intersection point of [a line between point p and the origin] with [a square of size 2 which is centered on the origin]".
3. We can now find the intersection point on the nearest side of the square by scaling down the coordinates of point p so that it lies on the square (by making either of the coordinates equal to +1 or -1).
4. Since the problem is now symmetrical, if we take the absolute value of x and y, we can transform (mirror) the problem into only "Quadrant 1" of the grid, where x and y are both positive. This avoids having to handle each side of the box individually.
5. In Quadrant 1, find the intersection point. If p is above the line $x=y$, we need to divide by y to put the point on the nearest line, and if it's below that line we need to divide by x instead. Note that even though

the scaling factor was calculated with the absolute values of the x and y coordinates, it can be used to scale the original coordinates also.

6. Now that we have the intersection point, inverse-transform it to undo the transformations from step 2 and step 1.

In C++, this looks something like:

```
struct point { double x, y; };
struct rect { point center; double xradius, yradius; }
point intersect(const point& p, const rect& r) {
    // Steps 1 and 2: Transform to origin and scale by
    point q{ (p.x - r.center.x) / r.xradius, (p.y - r.
    // Steps 3 and 4: Transform to Quadrant 1 and find
    double f = max(abs(q.x), abs(q.y));
    // Step 5: Divide by scaling factor to find inters
    point intersect{ q.x/f, q.y/f };
    // Step 6: Inverse transformations from steps 1 an
    return {intersect.x * r.xradius + r.center.x, inte
    r.center.y};
}
```

if you have access to a true vector and matrix class with operator overloading, the code becomes even easier:

```
point intersect(const point& p, const rect& r) {
    matrix m = scale_matrix(r.xradius, r.yradius)*tran
    r.center)
    point q = m * p;
    return m.inverse() * q/(max(abs(q.x), abs(q.y)))
}
```

Warning: This code doesn't handle the case where

`p.x==r.centerx && p.y==r.centery` which will result in a

divide-by-zero error in step 5. Handling that error condition will be left as an exercise to the reader. :-)

Share Improve this answer

edited Apr 15 at 5:35

Follow

answered Jan 20, 2023 at 12:21



Some Guy

509 ● 8 ● 17
