

3D Line-Plane Intersection

Asked 11 years, 4 months ago Modified 10 months ago Viewed 72k times

39 ▲ If given a line (represented by either a vector or two points on the line) how do I find the point at which the line intersects a plane? I've found loads of resources on this but I can't understand the equations there (they don't seem to be standard algebraic). I would like an equation (no matter how long) that can be interpreted by a standard programming language (I'm using Java).

21



Share Follow

edited Dec 24, 2014 at 9:40



ideasman42

36.8k ● 33 ● 176 ● 293

asked Apr 14, 2011 at 16:11



jt78

842 ● 2 ● 10 ● 21

possible duplicate of stackoverflow.com/questions/4382591/line-plane-intersection – Cobra_Fast Mar 3, 2015 at 15:52

1 Its not a duplicate, 4382591 isn't asking about the general case. – ideasman42 Oct 23, 2015 at 5:22

8 Answers

Sorted by: Trending sort available ⓘ Highest score (default) ▾

38 ▲ Here is a Python example which finds the intersection of a line and a plane.

38 Where the plane can be either a point and a normal, or a 4d vector (normal form), In the examples below (*code for both is provided*).



Also note that this function calculates a value representing where the point is on the line, (called `fac` in the code below). You may want to return this too, because values from 0 to 1 intersect the line segment - which may be useful for the caller.

Other details noted in the code-comments.

Note: This example uses pure functions, without any dependencies - to make it easy to move to other languages. With a `Vector` data type and operator overloading, it can be more concise (included in example below).

```
# intersection function
def isect_line_plane_v3(p0, p1, p_co, p_no, epsilon=1e-6):
    """
    p0, p1: Define the line.
    p_co, p_no: define the plane:
        p_co Is a point on the plane (plane coordinate).
        p_no Is a normal vector defining the plane direction;
            (does not need to be normalized).

    Return a Vector or None (when the intersection can't be found).
    """

    u = sub_v3v3(p1, p0)
    dot = dot_v3v3(p_no, u)

    if abs(dot) > epsilon:
        # The factor of the point between p0 -> p1 (0 - 1)
        # if 'fac' is between (0 - 1) the point intersects with the segment.
        # Otherwise:
        # < 0.0: behind p0.
        # > 1.0: infront of p1.
```

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 None

# -----
# generic math functions

def add_v3v3(v0, v1):
    return (
        v0[0] + v1[0],
        v0[1] + v1[1],
        v0[2] + v1[2],
    )

def sub_v3v3(v0, v1):
    return (
        v0[0] - v1[0],
        v0[1] - v1[1],
        v0[2] - v1[2],
    )

def dot_v3v3(v0, v1):
    return (
        (v0[0] * v1[0]) +
        (v0[1] * v1[1]) +
        (v0[2] * v1[2])
    )

def len_squared_v3(v0):
    return dot_v3v3(v0, v0)

def mul_v3_f1(v0, f):
    return (
        v0[0] * f,
        v0[1] * f,
        v0[2] * f,
    )

```

If the plane is defined as a 4d vector (*normal form*), we need to find a point on the plane, then calculate the intersection as before (see `p_co` assignment).

```

def isect_line_plane_v3_4d(p0, p1, plane, epsilon=1e-6):
    u = sub_v3v3(p1, p0)
    dot = dot_v3v3(plane, u)

    if abs(dot) > epsilon:
        # Calculate a point on the plane
        # (divide can be omitted for unit hessian-normal form).
        p_co = mul_v3_f1(plane, -plane[3] / len_squared_v3(plane))

        w = sub_v3v3(p0, p_co)
        fac = -dot_v3v3(plane, w) / dot
        u = mul_v3_f1(u, fac)
        return add_v3v3(p0, u)

    return None

```

For further reference, this was taken from Blender and adapted to Python. `isect_line_plane_v3()` in [math_geom.c](#)

For clarity, here are versions using the [mathutils](#) API (*which can be modified for other math libraries with operator overloading*).

```


# point-normal plane
def isect_line_plane_v3(p0, p1, p_co, p_no, epsilon=1e-6):
    u = p1 - p0
    dot = p_no * u
    if abs(dot) > epsilon:

```

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

×

```
# normal-form plane
def isect_line_plane_v3_4d(p0, p1, plane, epsilon=1e-6):
    u = p1 - p0
    dot = plane.xyz * u
    if abs(dot) > epsilon:
        p_co = plane.xyz * (-plane[3] / plane.xyz.length_squared)

        w = p0 - p_co
        fac = -(plane * w) / dot
        return p0 + (u * fac)

    return None
```

Share Follow

edited Oct 25, 2021 at 13:17

answered Aug 31, 2013 at 0:25



ideasman42

36.8k ● 33 ● 176 ● 293

- 3 Finally, an intersection method that actually works! Man.. I was banging my head on the wall for so long trying to fix my code for point of contact, thank you sir. – [Dan Bechard](#) Jan 29, 2018 at 9:48
- 1 it's infinite in the sense that this method doesn't impose a start/end point for the line. In practice it's limited by float precision. – [ideasman42](#) Feb 27, 2019 at 11:10

Here is a method in Java that finds the intersection between a line and a plane. There are vector methods that aren't included but their functions are pretty self explanatory.

28



```
/**
 * Determines the point of intersection between a plane defined by a point and a normal vector and a line defined by a point
 * and a direction vector.
 *
 * @param planePoint A point on the plane.
 * @param planeNormal The normal vector of the plane.
 * @param linePoint A point on the line.
 * @param lineDirection The direction vector of the line.
 * @return The point of intersection between the line and the plane, null if the line is parallel to the plane.
 */
public static Vector lineIntersection(Vector planePoint, Vector planeNormal, Vector linePoint, Vector lineDirection) {
    if (planeNormal.dot(lineDirection.normalize()) == 0) {
        return null;
    }

    double t = (planeNormal.dot(planePoint) - planeNormal.dot(linePoint)) / planeNormal.dot(lineDirection.normalize());
    return linePoint.plus(lineDirection.normalize().scale(t));
}
```

Share Follow

edited Apr 16, 2019 at 16:21

answered Oct 8, 2018 at 23:23



ZGorlock

515 ● 4 ● 11

- 1 N.B. LineDirection *must* be normalised – [smirkingman](#) Mar 4, 2019 at 8:10

I searched this answer for way to long! Great code example, easy to understand. – [m_power](#) Apr 12, 2019 at 21:29

@smirkingman You are correct, thank you! In my code it had been normalized before being passed into the method, I have edited the post to include this. – [ZGorlock](#) Apr 16, 2019 at 16:18

Just an update as this does not work for all case. Take a point `{0,0,0}` ray `{-1,0,0}`, use YZ plane at `x = 10` so plane point `{10,0,0}` and normal be either `{-1,0,0}` or `{1,0,0}` it does detect a hit even if the ray is casting away from the plane – [Franck](#) Nov 19, 2019 at 18:37

@Franck Good observation! This code only works for lines, not rays. – [ZGorlock](#) Feb 7, 2020 at 19:16

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



- Plane is not parallel to line (one point of intersection)
- Plane contains the line (line intersects at every point on it)

You can express the line in parameterized form, like here:

<http://answers.yahoo.com/question/index?qid=20080830195656AA3aEBr>

The first few pages of this lecture do the same for the plane:

<http://math.mit.edu/classes/18.02/notes/lecture5compl-09.pdf>

If the normal to the plane is perpendicular to the direction along the line, then you have an edge case and need to see whether it intersects at all, or lies within the plane.

Otherwise, you have one point of intersection, and can solve for it.

I know this isn't code but to get a robust solution you'll probably want to put this in the context of your application.

EDIT: Here's an example for which there's exactly one point of intersection. Say you start with the parameterized equations in the first link:

```
x = 5 - 13t
y = 5 - 11t
z = 5 - 8t
```

The parameter t can be anything. The (infinite) set of all (x, y, z) that satisfy these equations comprise the line. Then, if you have the equation for a plane, say:

```
x + 2y + 2z = 5
```

(taken from [here](#)) you can substitute the equations for x , y , and z above into the equation for the plane, which is now in only the parameter t . Solve for t . This is the particular value of t for that line that lies in the plane. Then you can solve for x , y , and z by going back up to the line equations and substituting t back in.

Share Follow

edited Apr 14, 2011 at 18:24

answered Apr 14, 2011 at 16:38



John

16k ● 9 ● 67 ● 109

1 Thanks, that's very helpful, but I'm still a little baffled. I understand how to get the equation for the line, that's pretty simple. But from there on I'm stuck. In lamen's terms, what exactly does the 'equation' of a plane mean? And what do I do to work out the point of intersection once both equations are solved? – [jt78](#) Apr 14, 2011 at 17:25

What information do you know about your plane? Points in the plane, for example? – [John](#) Apr 14, 2011 at 17:36

Pretty much anything, I think. I know at least 4 points on the plane, and it's defined independently so I should be able to find out almost anything. – [jt78](#) Apr 14, 2011 at 17:53

Using numpy and python:

17

```
#Based on http://geomalgorithms.com/a05-_intersect-1.html
from __future__ import print_function
import numpy as np

epsilon=1e-6

#Define plane
```

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

```

rayPoint = np.array([0, 0, 10]) #Any point along the ray

ndotu = planeNormal.dot(rayDirection)

if abs(ndotu) < epsilon:
    print ("no intersection or line is within plane")

w = rayPoint - planePoint
si = -planeNormal.dot(w) / ndotu
Psi = w + si * rayDirection + planePoint

print ("intersection at", Psi)

```

Share Follow

edited Feb 2, 2017 at 17:22

answered Sep 10, 2016 at 8:29



TimSC

1,421 ● 16 ● 19

If you have two points p and q that define a line, and a plane in the general cartesian form $ax+by+cz+d = 0$, you can use the parametric method.

5

If you needed this for coding purposes, here's a javascript snippet:

```

/**
 * findLinePlaneIntersectionCoords (to avoid requiring unnecessary instantiation)
 * Given points p with px py pz and q that define a line, and the plane
 * of formula ax+by+cz+d = 0, returns the intersection point or null if none.
 */
function findLinePlaneIntersectionCoords(px, py, pz, qx, qy, qz, a, b, c, d) {
    var tDenom = a*(qx-px) + b*(qy-py) + c*(qz-pz);
    if (tDenom == 0) return null;

    var t = - ( a*px + b*py + c*pz + d ) / tDenom;

    return {
        x: (px+t*(qx-px)),
        y: (py+t*(qy-py)),
        z: (pz+t*(qz-pz))
    };
}

// Example (plane at y = 10 and perpendicular line from the origin)
console.log(JSON.stringify(findLinePlaneIntersectionCoords(0,0,0,0,1,0,0,1,0,-10)));

// Example (no intersection, plane and line are parallel)
console.log(JSON.stringify(findLinePlaneIntersectionCoords(0,0,0,0,0,0,0,1,0,-10)));

```

Run code snippet

Expand snippet

Share Follow

answered Jul 27, 2017 at 22:32



DNax

1,391 ● 1 ● 18 ● 29

- 1 Should "var t = - (apx + bpy + cpz + d) / tDenom;" actually be subtracting d ? so ... var t = - (apx + bpy + cpz - d) / tDenom; – Russell Okamoto May 2, 2018 at 22:48

d is only the independent term with nothing more special about it. In the example, you can see that a plane $y=10$ is represented with $d=-10$ (first console log). If it is still not clear I may write down the formula development/solution. – DNax May 3, 2018 at 12:32

What if the line is **contained** within the plane? This function only seems to return a point or null, and in this case I would expect it to return the line itself (because that's what intersects the plane). And can I conclude that when null is returned that the line never intersects the plane?

– Mark Miller Aug 20, 2018 at 21:37

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

4

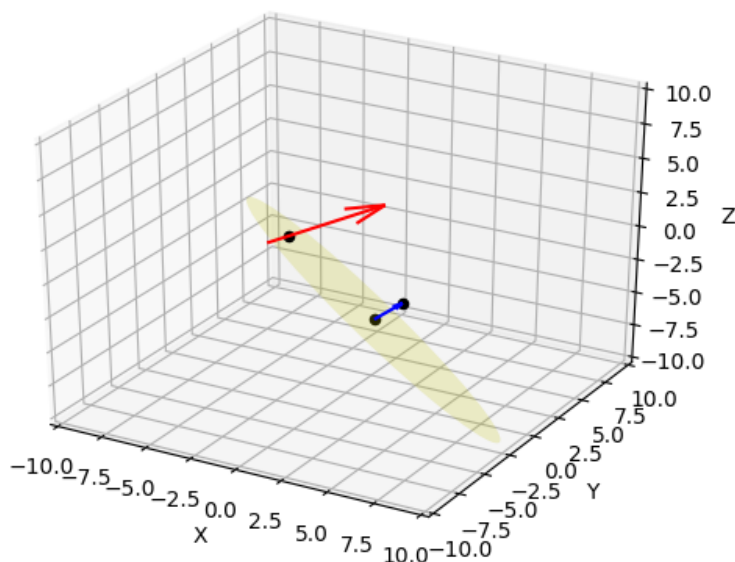
```
# n: normal vector of the Plane
# V0: any point that belongs to the Plane
# P0: end point 1 of the segment P0P1
# P1: end point 2 of the segment P0P1
n = np.array([1., 1., 1.])
V0 = np.array([1., 1., -5.])
P0 = np.array([-5., 1., -1.])
P1 = np.array([1., 2., 3.])

w = P0 - V0;
u = P1 - P0;
N = -np.dot(n,w);
D = np.dot(n,u)
sI = N / D
I = P0 + sI*u
print I
```

Result

```
[-3.90909091  1.18181818 -0.27272727]
```

I checked it graphically it seems to work,



This I believe is a more robust implementation of the link shared [before](#)

Share Follow

edited Apr 18, 2017 at 11:12

answered Apr 18, 2017 at 9:58



BBsysDyn

4,189 ● 8 ● 45 ● 60

This question is old but since there is such a much more convenient solution I figured it might help someone.

3

Plane and line intersections are quite elegant when expressed in homogeneous coordinates but lets assume you just want the solution:

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: $x = L * p = \{x_0, x_1, x_2, x_3\}$

$x_intersect = (\{x_0, x_1, x_2\} / x_3)$ where if x_3 is zero there is no intersection in the euclidean sense.

Share Follow

edited Jul 26, 2017 at 15:07

answered Feb 7, 2017 at 14:48



midji
374 ● 3 ● 9

Just to expand on [ZGorlock's](#) answer, I have done the dot product, addition and scaling of 3D Vectors. The references for these calculations are [Dot Product](#), [Add two 3D vectors](#) and [Scaling](#). Note: Vec3D is just a custom class which has points: x, y and z.

```
/**
 * Determines the point of intersection between a plane defined by a point and a normal vector and a line defined by a point
 * and a direction vector.
 *
 * @param planePoint    A point on the plane.
 * @param planeNormal    The normal vector of the plane.
 * @param linePoint    A point on the line.
 * @param lineDirection The direction vector of the line.
 * @return The point of intersection between the line and the plane, null if the line is parallel to the plane.
 */
public static Vec3D lineIntersection(Vec3D planePoint, Vec3D planeNormal, Vec3D linePoint, Vec3D lineDirection) {
    // ax x bx + ay x by
    int dot = (int) (planeNormal.x * lineDirection.x + planeNormal.y * lineDirection.y);
    if (dot == 0) {
        return null;
    }

    // Ref for dot product calculation: https://www.mathsisfun.com/algebra/vectors-dot-product.html
    int dot2 = (int) (planeNormal.x * planePoint.x + planeNormal.y * planePoint.y);
    int dot3 = (int) (planeNormal.x * linePoint.x + planeNormal.y * linePoint.y);
    int dot4 = (int) (planeNormal.x * lineDirection.x + planeNormal.y * lineDirection.y);

    double t = (dot2 - dot3) / dot4;

    float xs = (float) (lineDirection.x * t);
    float ys = (float) (lineDirection.y * t);
    float zs = (float) (lineDirection.z * t);
    Vec3D lineDirectionScale = new Vec3D( xs, ys, zs);

    float xa = (linePoint.x + lineDirectionScale.x);
    float ya = (linePoint.y + lineDirectionScale.y);
    float za = (linePoint.z + lineDirectionScale.z);

    return new Vec3D(xa, ya, za);
}
```

Share Follow

edited Aug 14, 2021 at 19:26

answered Nov 13, 2018 at 13:35



GreenHapi
3 ● 2



Muhammad Qumail
1 ● 3

This is clearly incorrect: the dot product of 3D vector is $(v_0.x * v_1.x + v_0.y * v_1.y + v_0.z * v_1.z)$ – [Depau](#) May 21 at 21:02

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