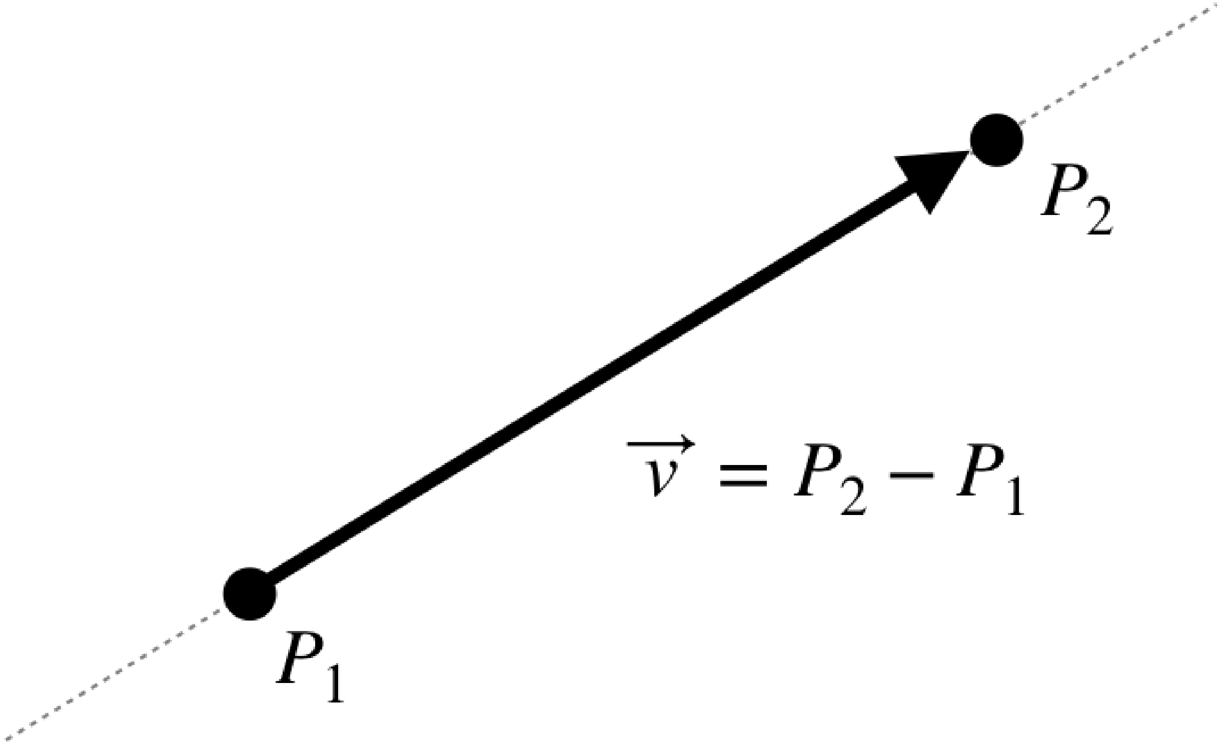
Line Equation in 3D

Related Topics: Plane Equation
Download: StarGenerator.zip



Graph of a line

The equation of a line is defined with 2 known points. This page describes a line equation using parametric form; a point P_1 on the line and the direction vector $\vec{v}$ of the line.

The direction vector can be computed by subtracting 2 known points;
$$\vec{v} = P_2 - P_1 = (x_2 - x_1, y_2 - y_1, z_2 - z_1)$$

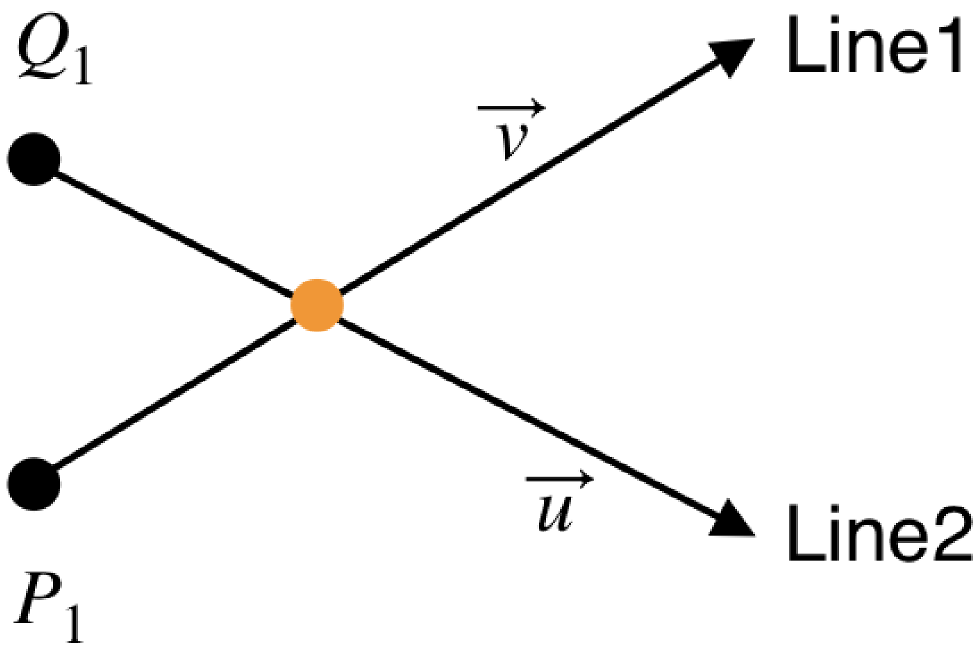
Therefore, the parametric form of the line equation is;
$$\begin{aligned}\text{Line} &= P_1 + t\vec{v} \\ &= (x_1, y_1, z_1) + t(x_2 - x_1, y_2 - y_1, z_2 - z_1)\end{aligned}$$

Or, each element (*x, y, z*) of the line can be written by the parameter *t*;
$$\begin{cases} x = x_1 + t(x_2 - x_1) \\ y = y_1 + t(y_2 - y_1) \\ z = z_1 + t(z_2 - z_1) \end{cases}$$
Notice that it is same as linear interpolation formula on each axis. If $0 < t < 1$, it is an interpolation, otherwise extrapolation.

Also note that 2D line equation is frequently represented with slope-intercept form $y = mx + b$ or standard form $ax + by + c = 0$. However these forms are not suitable for computer algorithms because the slope of a vertical line is undefined (±∞), and plus the slope-itercept form cannot be used in 3D space.

Intersection of 2 Lines

LOG



Intersection of 2 Lines

The intersection point of 2 lines is solving the linear system of 2 lines using vector algebra.

$$\begin{cases} \text{Line1} = P_1 + t\vec{v} \\ \text{Line2} = Q_1 + s\vec{u} \end{cases}$$

First, solve the linear system, Line1 = Line2 for *t*. Then substitute *t* into Line1 equation to find the intersection point (x, y, z).

(Reference: Intersection of two lines in three-space, Ronald Goldman, Graphics Gems, page 304, 1990)

$$\begin{aligned}
P_1 + t\vec{v} &= Q_1 + s\vec{u} && (\because \text{Line1} = \text{Line2}) \\
t\vec{v} &= (Q_1 - P_1) + s\vec{u} && (\because \text{subtract } P_1 \text{ on both sides}) \\
t\vec{v} \times \vec{u} &= (Q_1 - P_1) \times \vec{u} + s\vec{u} \times \vec{u} && (\because \text{cross product } \vec{u} \text{ on both sides}) \\
t\vec{v} \times \vec{u} &= (Q_1 - P_1) \times \vec{u} && (\because \vec{u} \times \vec{u} = 0) \\
t(\vec{v} \times \vec{u}) \cdot (\vec{v} \times \vec{u}) &= (Q_1 - P_1) \times \vec{u} \cdot (\vec{v} \times \vec{u}) && (\because \text{dot product } \vec{v} \times \vec{u} \text{ on both side}) \\
t &= \frac{(Q_1 - P_1) \times \vec{u} \cdot (\vec{v} \times \vec{u})}{(\vec{v} \times \vec{u}) \cdot (\vec{v} \times \vec{u})} && (\because \text{divide by } (\vec{v} \times \vec{u}) \cdot (\vec{v} \times \vec{u}))
\end{aligned}$$

If two lines are parallel, the cross product of 2 direction vectors of the lines becomes zero. You can determine if two lines are intersect or not with;

$$\vec{v} \times \vec{u} = 0$$

Here is C++ code to find the intersection point of 2 lines. Please see the details in Line.cpp.

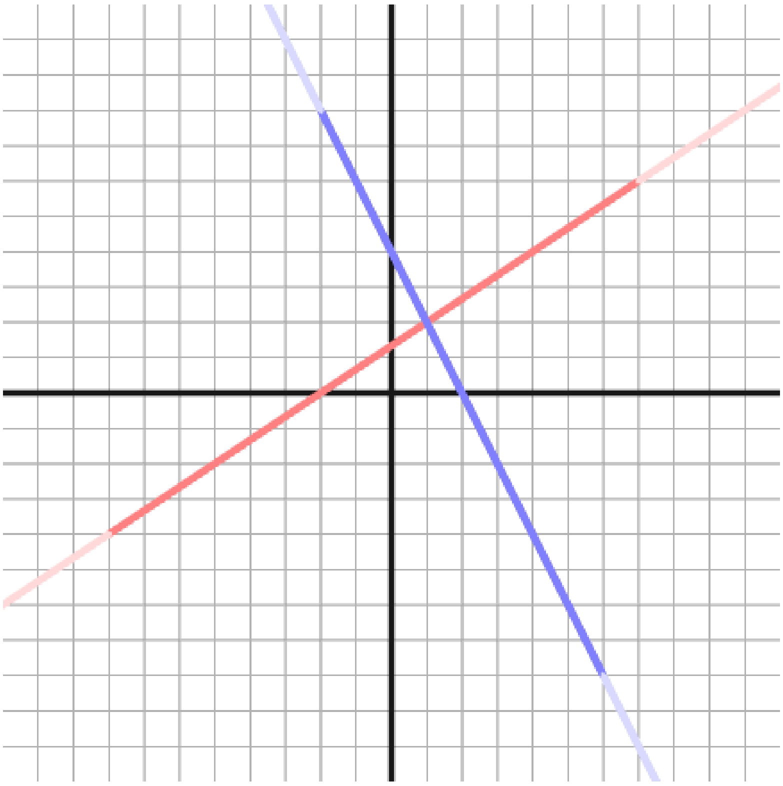
```
// dependency: Vector3, Line
struct Vector3
{
    float x;
    float y;
    float z;
};

class Line
{
    Vector3 direction;                  // (Vx, Vy, Vz)
    Vector3 point;                      // (Px, Py, Pz)
};


Vector3 intersect(Line& line1, Line& line2)
{
    Vector3 p = line1.point;            // P1
    Vector3 v = line1.direction;        // v
    Vector3 q = line2.point;            // Q1
    Vector3 u = line2.direction;        // u

    // find a = v x u
    Vector3 a = v.cross(u);             // cross product
```

LOG

```
    // find dot product = (v x u) . (v x u)
    float dot = a.dot(a);              // inner product

    // if v and u are parallel (v x u = 0), then no intersection, return NaN point
    if(dot == 0)
        return Vector3(NAN, NAN, NAN);

    // find b = (Q1-P1) x u
    Vector3 b = (q - p).cross(u);      // cross product

    // find t = (b.a)/(a.a) = ((Q1-P1) x u) .(v x u) / (v x u) . (v x u)
    float t = b.dot(a) / dot;

    // find intersection point by substituting t to the line1 eq
    Vector3 point = p + (t * v);
    return point;
}
```

Example: Intersection of 2 Lines in 2D (Interactive Demo)

The following JavaScript interactive demo is finding the intersection point from 2 lines in 2D. It requires WebGL enabled browsers.



Line 1

P1: (-8, -4)

x: -8

y: -4

P2: (7, 6)

x: 7

y: 6

Parametric Form: (-8.0, -4.0) + t(15.0, 10.0)

Slope-Intercept Form: y = 0.67x + 1.33

Standard Form: -0.67x + y - 1.33 = 0

Line 2

Q1: (-2, 8)

x: -2

y: 8

Q2: (6, -8)

x: 6

y: -8

Parametric Form: (-2.0, 8.0) + t(8.0, -16.0)

Slope-Intercept Form: y = -2.00x + 4

Standard Form: 2.00x + y - 4 = 0

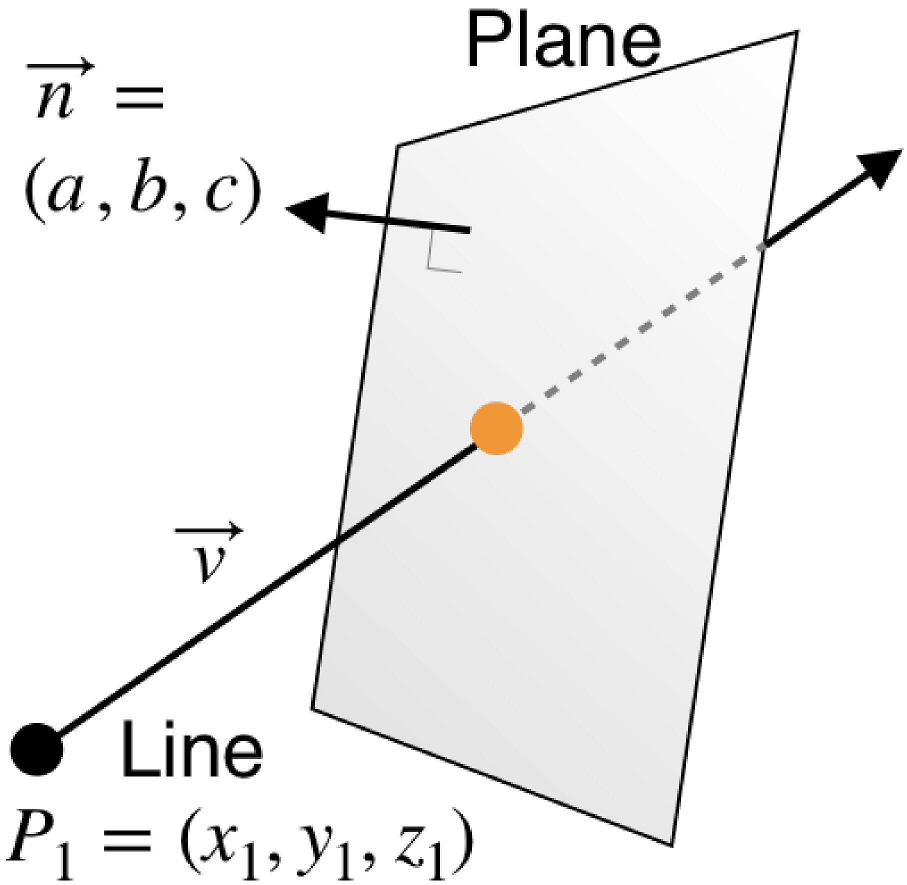
LOG

Intersection Point

(1.000, 2.000)

Reset Points

Intersection of Line and Plane



Intersection of Line and Plane

A plane equation in 3D is defined with its normal vector (a, b, c) and a known point on the plane;
$ax + by + cz + d = 0$

Please refer to [Plane Equation] to see how to derive the plane equation.

Finding the intersection point of a line and plane is solving a linear system of a line and plane.

$$\begin{cases} \text{Line:} & P_1 + t\vec{v} = (x_1 + tv_x, y_1 + tv_y, z_1 + tv_z) \\ \text{Plane:} & ax + by + cz + d = 0 \end{cases}$$

First, substitute (x, y, z) of the plane equation with $(x_1 + tv_x, y_1 + tv_y, z_1 + tv_z)$ from the line equation. Then, solve for *t*;

$$\begin{aligned} ax + by + cz + d &= 0 \\ a(x_1 + tv_x) + b(y_1 + tv_y) + c(z_1 + tv_z) + d &= 0 \quad (\text{substitute } (x_1 + tv_x, y_1 + tv_y, z_1 + tv_z)) \\ ax_1 + by_1 + cz_1 + d + t(av_x + bv_y + cv_z) &= 0 \\ t(av_x + bv_y + cv_z) &= -(ax_1 + by_1 + cz_1 + d) \\ t &= \frac{-(ax_1 + by_1 + cz_1 + d)}{(av_x + bv_y + cv_z)} \end{aligned}$$

If the denominator part is zero, there is no intersection. And the sum of multiplications of the above equation can be replaced with dot products;

LOG

$$\vec{n} \cdot P_1 = ax_1 + by_1 + cz_1$$
$$\vec{n} \cdot \vec{v} = av_x + bv_y + cv_z$$

$$\therefore \; t = \frac{-(\vec{n} \cdot P_1 + d)}{\vec{n} \cdot \vec{v}}$$

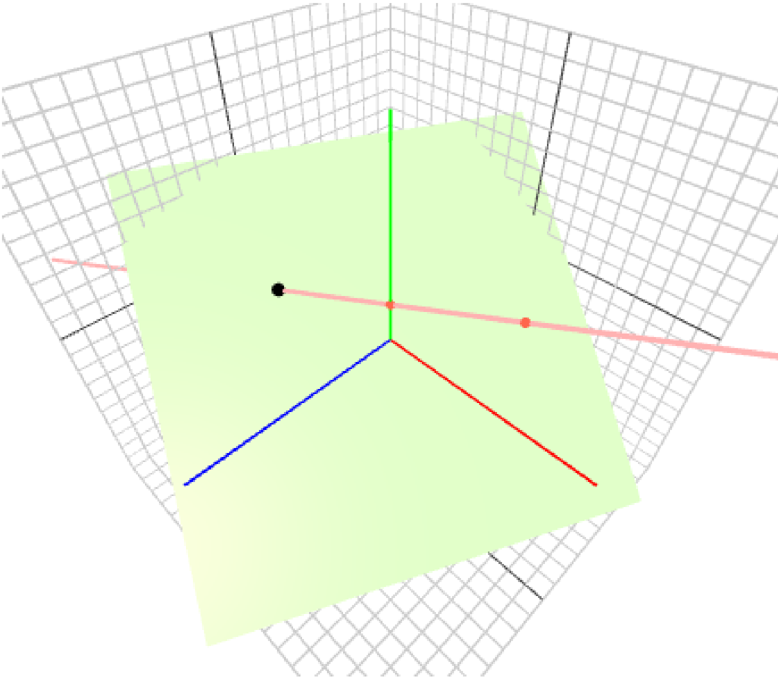
Here is C++ code to find the intersection point of a line and a plane.

```
// dependency: Vector3, Line, Plane
struct Vector3
{
    float x;
    float y;
    float z;
};

class Line
{
    Vector3 direction;                 // v
    Vector3 point;                     // p
};

class Plane
{
    Vector3 normal;                    // (a, b, c)
    Vector3 d;                         // constant term: d = -(a*x0 + b*y0 + c*z0)
};


Vector3 intersect(Line& line, Plane& plane)
{
    // from line = p + t * v
    Vector3 p = line.point;            // (x1, y1, z1)
    Vector3 v = line.direction;        // (Vx, Vy, Vz)

    // from plane: ax + by + cz + d = 0
    Vector3 n = plane.normal;          // (a, b, c)
    Vector3 d = plane.d;               // constant term of plane

    // dot products
    float dot1 = n.dot(v);             // a*Vx + b*Vy + c*Vz
    float dot2 = n.dot(p);             // a*x1 + b*y1 + c*z1

    // if denominator=0, no intersect
    if(dot1 == 0)
        return Vector3(NAN, NAN, NAN); // return NaN point

    // find t = -(a*x1 + b*y1 + c*z1 + d) / (a*Vx + b*Vy + c*Vz)
    float t = -(dot2 + d) / dot1;

    // find intersection point by substituting t to line eq
    return p + (t * v);
}
```

Example: Intersection of Line and Plane (Interactive Demo)

The following JavaScript interactive demo is finding the intersection point from a line and a plane in 3D. Use left and right mouse buttons to rotate the view, or to zoom in and out. It requires WebGL enabled browsers.

LOG



Plane

2.0x + 3.0y + 1.0z + 5.0 = 0

a: 2

b: 3

c: 1

d: 5

Line

(5.0, 3.0, -2.0) + t(-5.000, -1.000, 2.000)

P1: (5, 3, -2)

x: 5

y: 3

z: -2

P2: (0, 2, 0)

x: 0

y: 2

z: 0

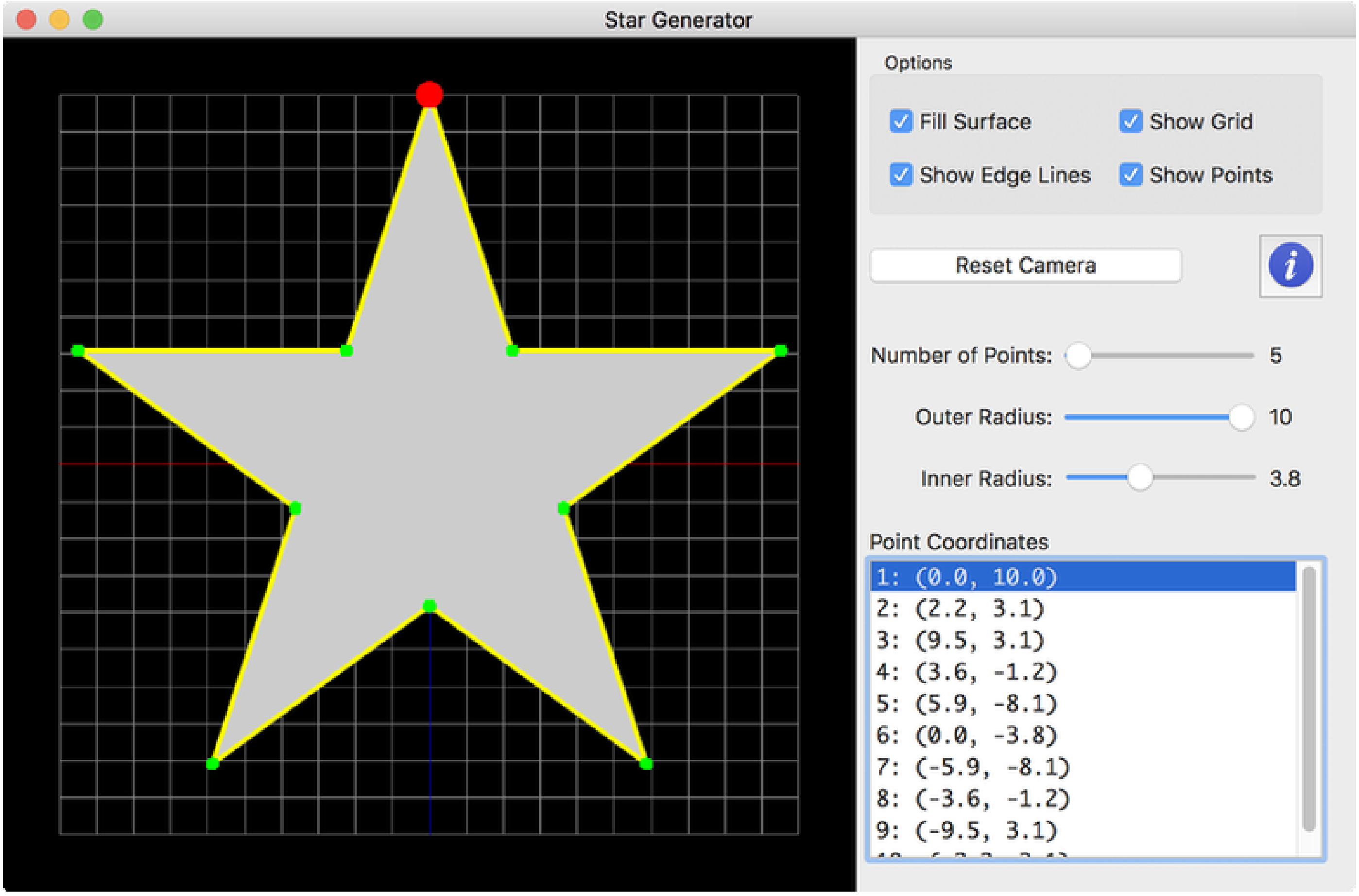
Intersection Point

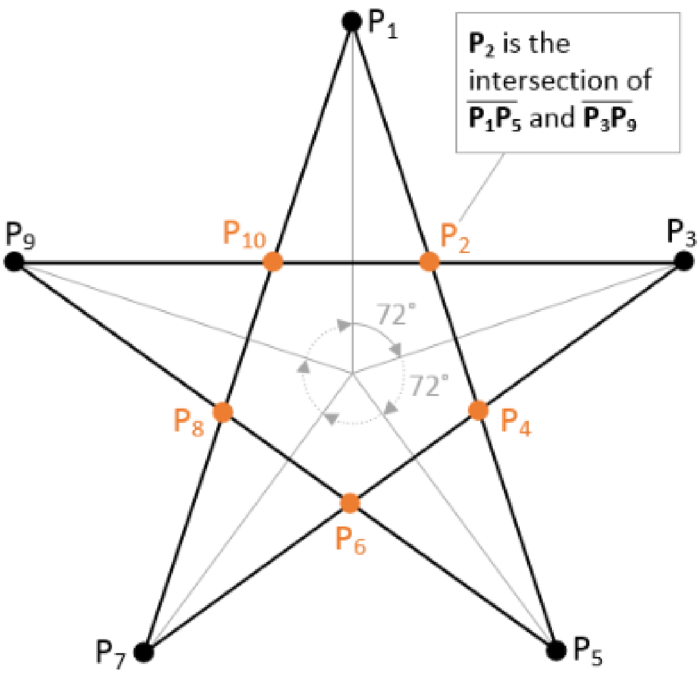
(-5.000, 1.000, 2.000)

Reset

Example: Drawing N-point Star

LOG





P_2 is the intersection of 2 lines; P_1P_5 and P_3P_9

Download source and binary:

[StarGenerator.zip] *(Updated: 2020-07-17)*

[StarGenerator_mac.zip] *(Updated: 2023-04-25)*

Drawing a N-point star is a practical application using line intersection algorithm. The steps for drawing a star are;

1. Define the top outer point with a given radius, for example, if the radius is 10, then the top point, P_1 will be (0, 10).
2. Compute the other outer points P_3, P_5, ... P_{n-1}, by rotating the top point by N / 360 degree repeatedly, for example, 72 degree for 5-point star.

LOG

3. Find inner points P_2, P_4, ... P_n, which are intersection points from the lines by connecting the outer points. For example P_2 is the intersection point of the 2 lines; P_1P_5 and P_3P_9.

←Back