Math for 3D/Games Programmers

4. Geometrical Objects Equations

Table of Contents

Line

- A **straight line** can be represented mathematically in multiple ways
- The simplest way is to use **linear function,** which is an example of an **explicit equation**:

$$y = f(x) = ax + b$$

- Another way is to use the **implicit equation**:

$$ax + by + c = 0$$

- Finally there is the **parametric equation**:

$$p(t) = p_0 + \vec{v}t$$

Line

- Each of those forms has its own unique advantages
- The linear function and implicit equation allow only for representation of a 2D line.
  The parametric equation can represent both a 2D and 3D line
- In this subsection we will find a straight line equation in all those three forms, given points $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$
- We will find equations of two distinct lines and find their intersection point

Line – Linear Function

- The classical explicit equation, a function of one variable $y = f(x)$, which we all know from school:

$$y = f(x) = ax + b$$

- This form is sometimes called **slope-intercept form** or **explicit form**
- a is called **slope of the straight line**. Often denoted as m
- b is called **y-intercept**. Also called **free term** or **constant**
- (x, y) are coordinates of a point that lies on the line

Line – Linear Function

- Given points p_1 and p_2 let's find the linear function of a line that passes via those points
- The points satisfy the following system of equations:

$$\begin{cases} y_1 = \boldsymbol{a}x_1 + \boldsymbol{b} \\ y_2 = \boldsymbol{a}x_2 + \boldsymbol{b} \end{cases}$$

- After solving the above system we get a and b
- The solution is:

$$\begin{cases} a = \dfrac{y_2 - y_1}{x_2 - x_1} \\ b = y_1 - ax_1 \end{cases}$$

Line – Linear Function

- We have equations of two lines:

$$y = a_1 x + b_1$$

$$y = a_2 x + b_2$$

- We're looking for their intersection point (p_x, p_y)
- Intersection point satisfies both equations at the same time, so:

$$\begin{cases} \boldsymbol{p_y} = a_1 \boldsymbol{p_x} + b_1 \\ \boldsymbol{p_y} = a_2 \boldsymbol{p_x} + b_2 \end{cases}$$

Line – Linear Function

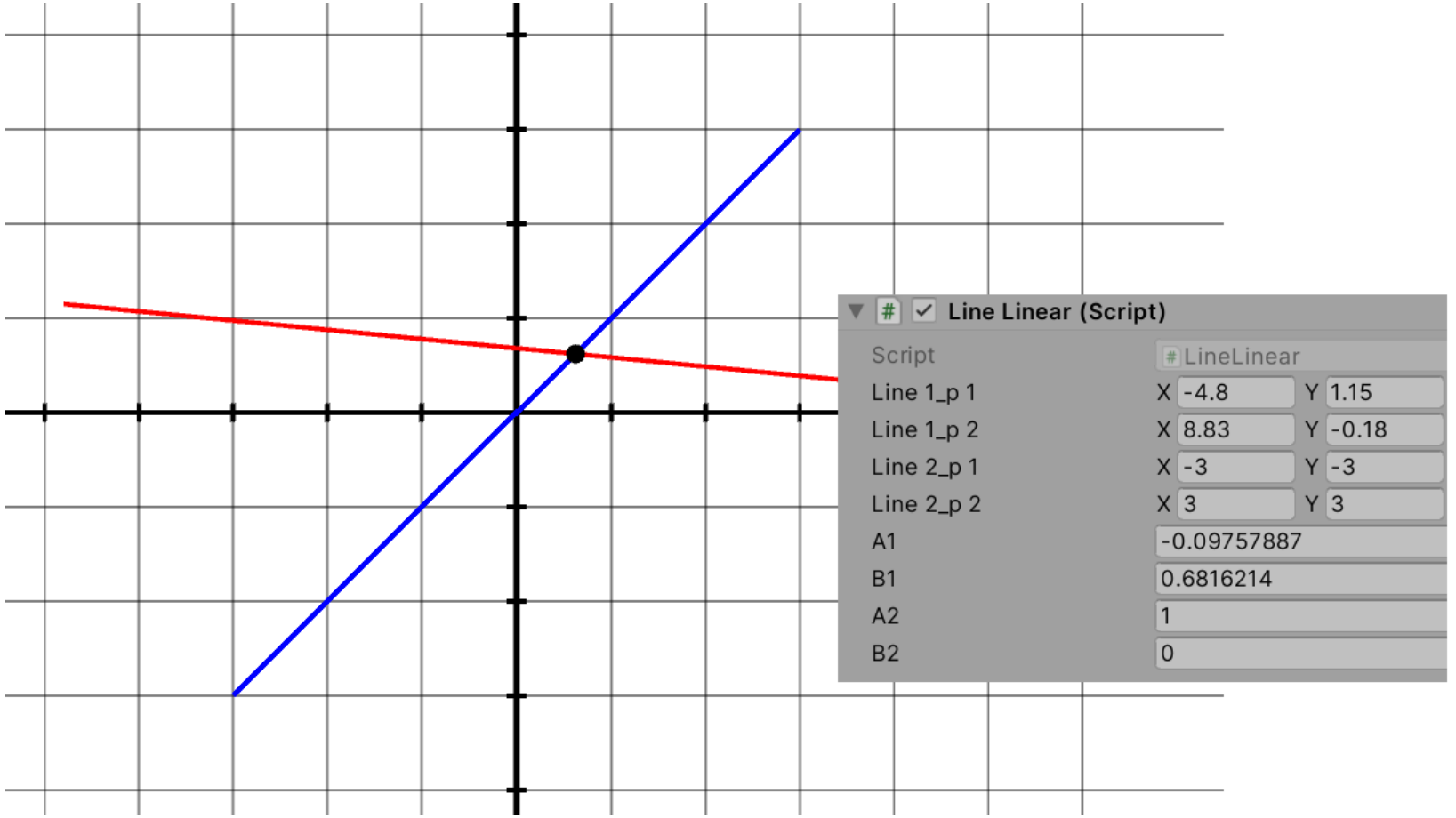
- System to solve:

$$\begin{cases} \boldsymbol{p_y} = a_1 \boldsymbol{p_x} + b_1 \\ \boldsymbol{p_y} = a_2 \boldsymbol{p_x} + b_2 \end{cases}$$

- A solution to this system is:

$$\begin{cases} p_x = \dfrac{b_2 - b_1}{a_1 - a_2} \\ p_y = a_1 p_x + b_1 \end{cases}$$

Line – Linear Function



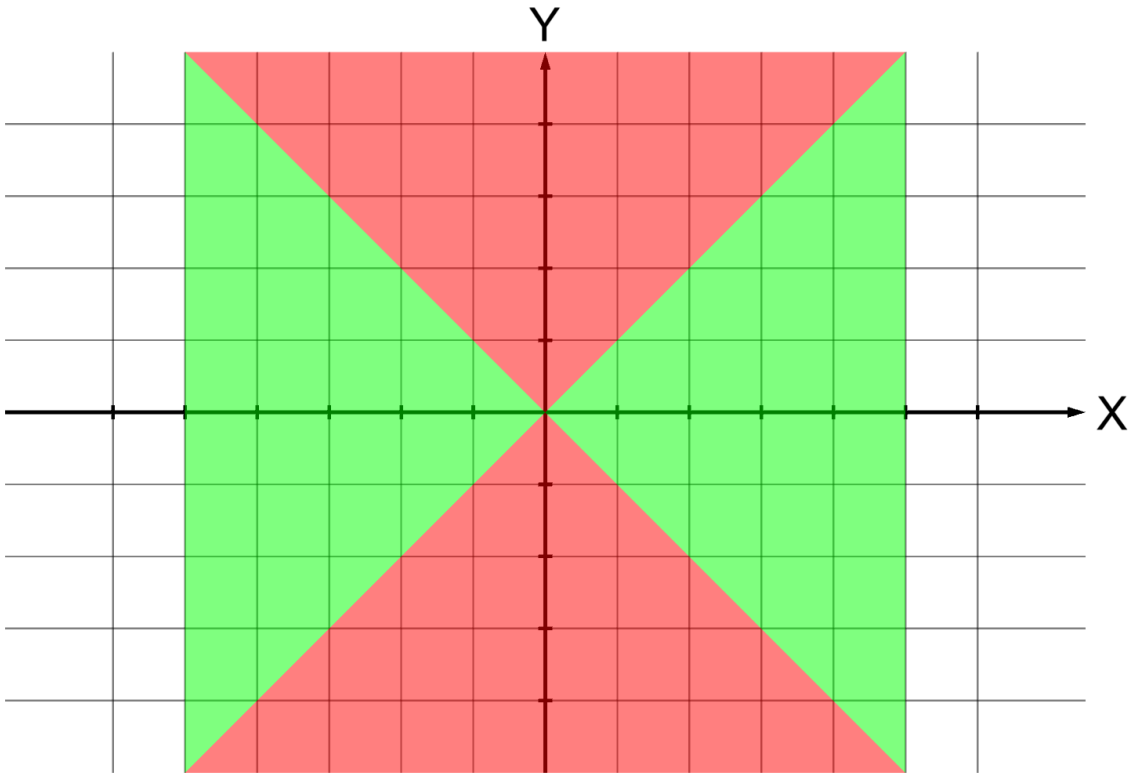
Line – Linear Function

- The biggest disadvantage of this straight line representation is the inability to represent a vertical line
- Additionally, the more vertical a line gets, the less precise are calculations that use it
- A constant change in argument x leads to a „step” of varying distance on a line, that depends on the parameter a
- To represent a line we only need two values
- Works only in 2D

Line – Linear Function – Vertical Line

- A solution to the vertical line problem may be an alternative equation:

$$x = ay + b$$



Line – Linear Function – Tangent

- The following formula is true:

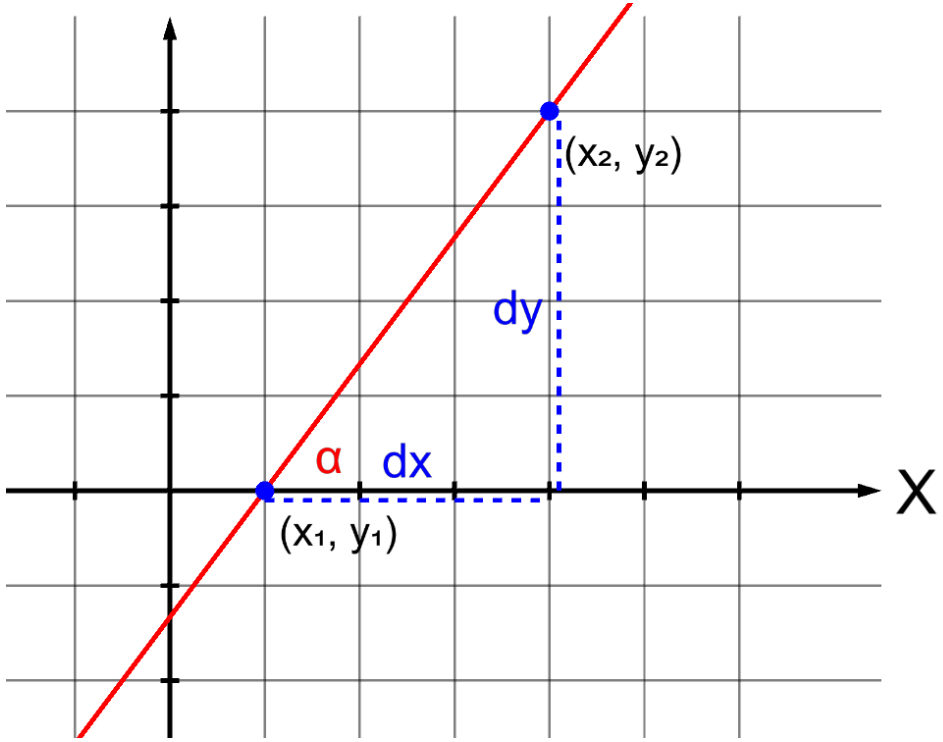
$$a = \tan(\alpha) \qquad \alpha - \text{angle between the line and the X axis}$$

$$dx = x_2 - x_1$$

$$dy = y_2 - y_1$$

$$a = \frac{y_2 - y_1}{x_2 - x_1}$$

$$\tan(\alpha) = \frac{dy}{dx}$$



Line – Linear Function – Linear Mapping

- The linear function's meaning extends beyond its geometric interpretation
- The linear function, in general, can represent any „linear mapping" – a function which linearly maps one interval $[a, b]$ onto another $[c, d]$

Line – Linear Function – Linear Mapping

- As an example, with the linear function we can easily find a formula that „maps” the interval $[10, 50]$ onto $[0, 1]$
- Value of 10 is to be mapped to 0, while 50 to 1.
  Values in-between are mapped linearly, so for example 30 is mapped to 0.5
- Such a mapping can be used, for example, to implement how a light’s or sound’s intensity changes with distance from the source

Line – Linear Function – Linear Mapping

- Let’s find a linear function which determines the aforementioned mapping
- The function is of form:

$$y = f(x) = ax + b$$

- We know that:

$$f(10) = 0$$
$$f(50) = 1$$

- Therefore:

$$0 = a * 10 + b$$
$$1 = a * 50 + b$$

Line – Linear Function – Linear Mapping

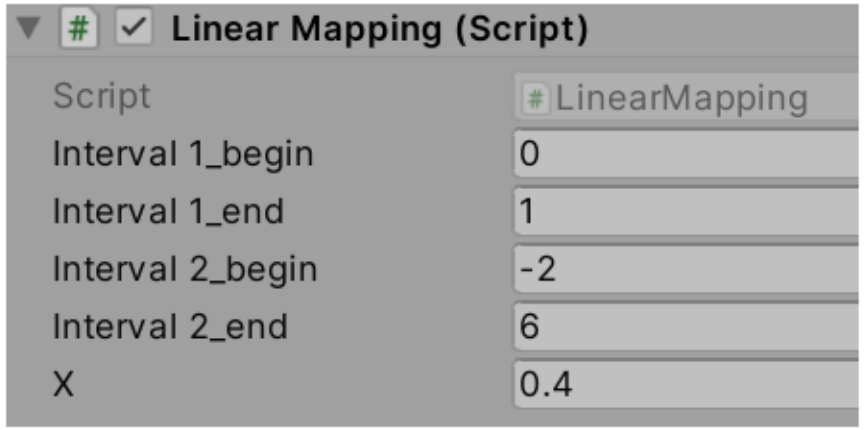
- A solution is the formula that we've seen before:

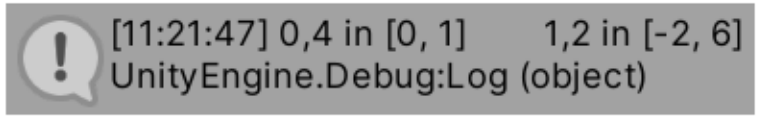
$$\begin{cases} a = \dfrac{y_2 - y_1}{x_2 - x_1} \\ b = y_1 - ax_1 \end{cases}$$

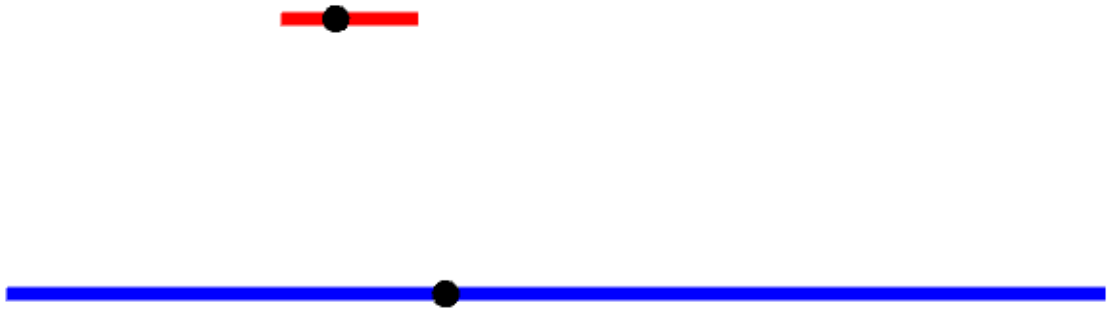
$$\begin{cases} a = \dfrac{1 - 0}{50 - 10} \\ b = 0 - a * 10 \end{cases}$$

$$y = ax + b = \frac{1}{40}x - 0.25$$

Line – Linear Function – Linear Mapping







Line – Implicit Equation

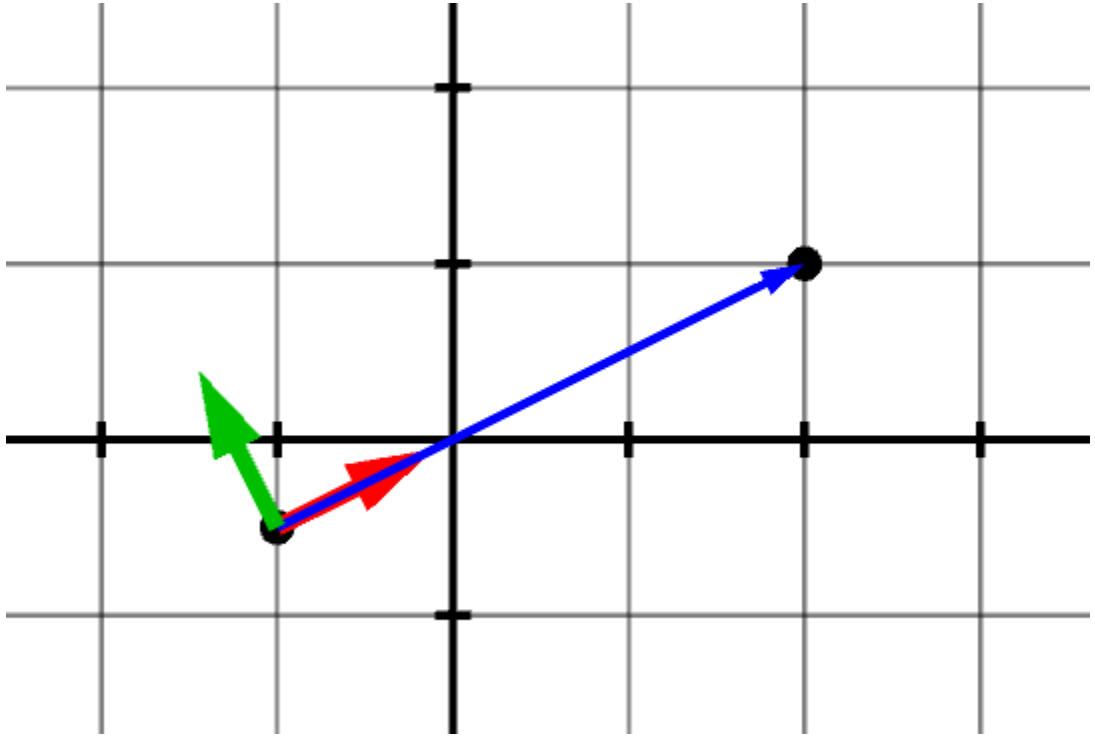
- The equation of form:

$$ax + by + c = 0$$

- This form is also called **general** or **standard**
- Implicit equations do not give as „explicit solutions” to variables/functions.
  Solutions (in this case points that belong on the line) are „embedded” in the equation (x and y coordinates)
- $[a, b]$ is the normal vector of the line.
  It doesn’t have to be normalized, although it usually is
- c is the distance of the line from the origin
- (x, y) are the coordinates of any point that lies on the line

Line – Implicit Equation

- To determine the equation of the implicit line we start by determining a and b:



Line – Implicit Equation

- We find $\vec{v} = p_2 - p_1$. We normalize it
- We find the normal $[a, b] = [-\vec{v}_y, \vec{v}_x]$
- Now we can find c:

$$ax + by + c = 0$$

$$c = -(ax + by)$$

$$c = -(ax_1 + by_1)$$

- If $[a, b]$ is normalized, then the expression $ax + by + c$ gives us the **„signed”** distance of point (x, y) from the line

Line – Implicit Equation

- We have two distinct lines:

$$a_1 x + b_1 y + c_1 = 0$$

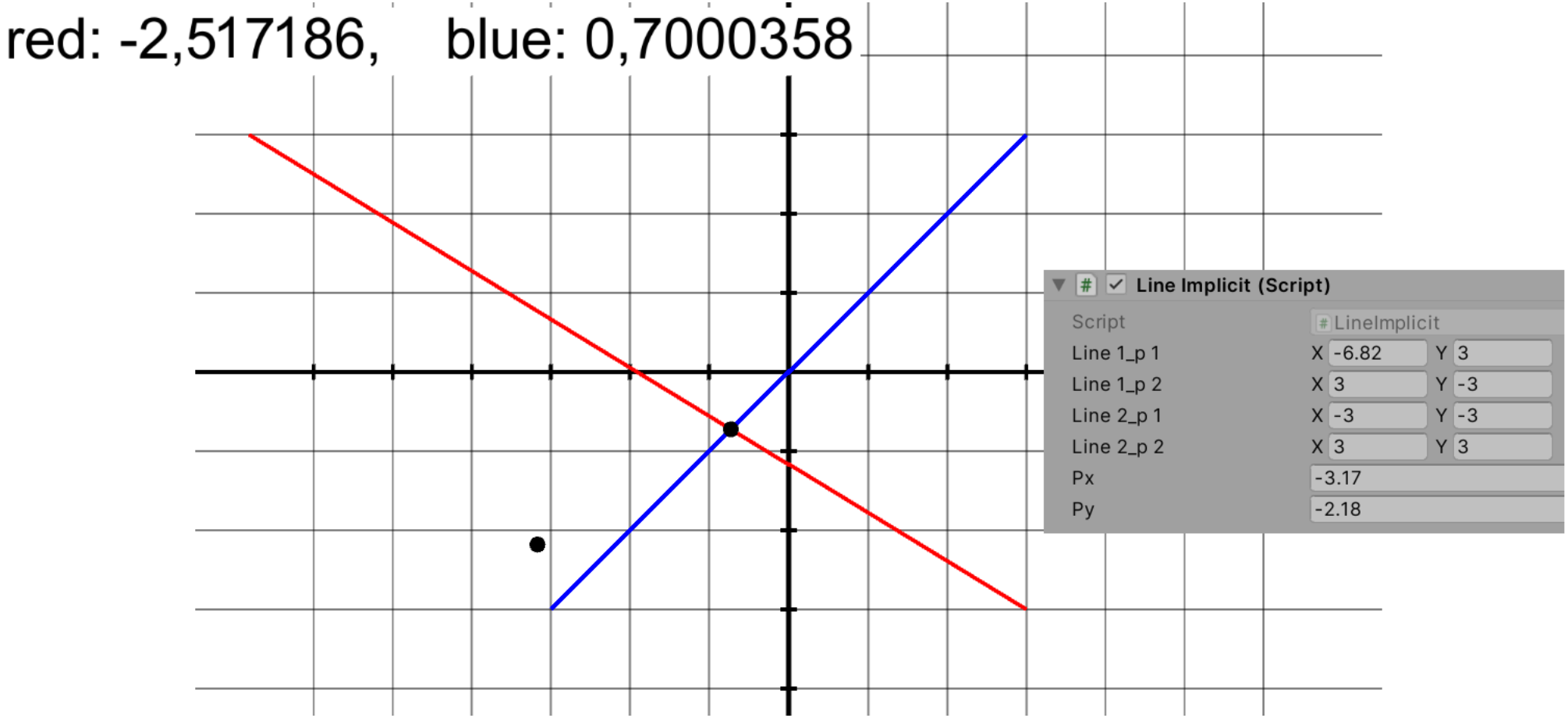
$$a_2 x + b_2 y + c_2 = 0$$

- Finding (x, y) in this system gives us the intersection point of those two lines:

$$\begin{cases} a_1 \boldsymbol{x} + b_1 \boldsymbol{y} + c_1 = 0 \\ a_2 \boldsymbol{x} + b_2 \boldsymbol{y} + c_2 = 0 \end{cases}$$

- Wolfram

Line – Implicit Equation



Line – Implicit Equation

- We've just discussed the implicit line equation:

$$ax + by + c = 0$$

- It's worth knowing though that quite often we can find this equation written as:

$$Ax + By + C = 0$$

Line – Implicit Equation

- The implicit equation is a generalization of the linear function form:

$$Ax + By + C = 0$$

$$By = -(Ax + C)$$

$$y = -\frac{Ax + C}{B}$$

$$y(x) = -\frac{A}{B}x + \frac{C}{B}$$

$$a = -\frac{A}{B} \qquad b = \frac{C}{B}$$

Line – Implicit Equation

- As an example, let's take an implicit equation the represents a certain line:

$$3x - 5y + 1 = 0$$

- The linear function, which represents the exact same line is:

$$a = -\frac{A}{B} \qquad b = \frac{C}{B}$$

$$a = -\frac{3}{-5} \qquad b = \frac{1}{-5}$$

$$y = ax + b = \frac{3}{5}x - \frac{1}{5}$$

Line – Implicit Equation

- It’s easy to determine on which side of a line a points is
- It’s easy to find the distance of a point from a line
- Certain formulas/applications require that a geometric object be in implicit form
- Points on a line can be generated by solving the equation for one of the coordinates
- It’s possible to represent a vertical line
- Needs three values to represent a line
- Works only in 2D

Line – Parametric Equation

- The equation of form:

$$p(t) = p_0 + \vec{v}t$$

- p are coordinates of a point on the line
- p_0 are coordinates of some fixed (arbitrary) point on the line
- $\vec{v}$ is the line's **direction vector**. Depending on our needs it's normalized or not
- t is a **parameter**, whose different values generate us different points on the line.
  For $t \geq 0$ the line reduces to a **ray**.
  When $t \in [a, b]$ we have a **segment**
- Allows us to represent a line in both 2D and 3D

Line – Parametric Equation

- **Vector** form of parametric line equation:

$$p(t) = p_0 + \vec{v}t$$

- We can also write this equation in the **scalar** form (2D):

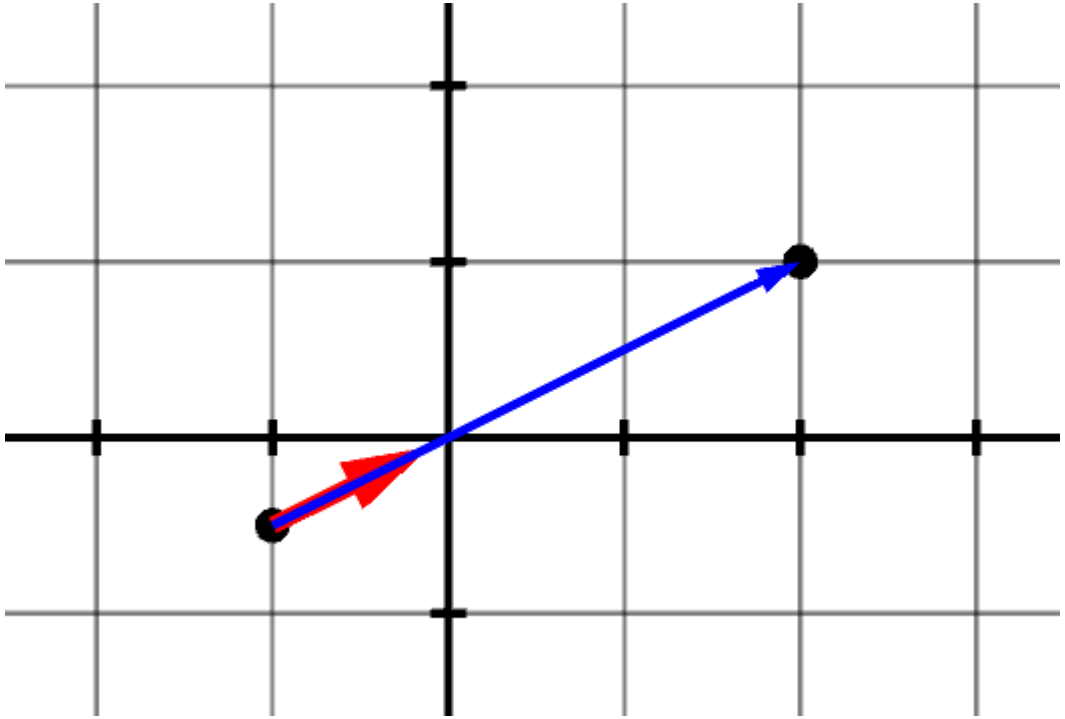
$$\begin{cases} x(t) = x_0 + \vec{v}_x t \\ y(t) = y_0 + \vec{v}_y t \end{cases}$$

- Or (3D case):

$$\begin{cases} x(t) = x_0 + \vec{v}_x t \\ y(t) = y_0 + \vec{v}_y t \\ z(t) = z_0 + \vec{v}_z t \end{cases}$$

Line – Parametric Equation

- The direction vector $\vec{v}$ can be normalized or not:



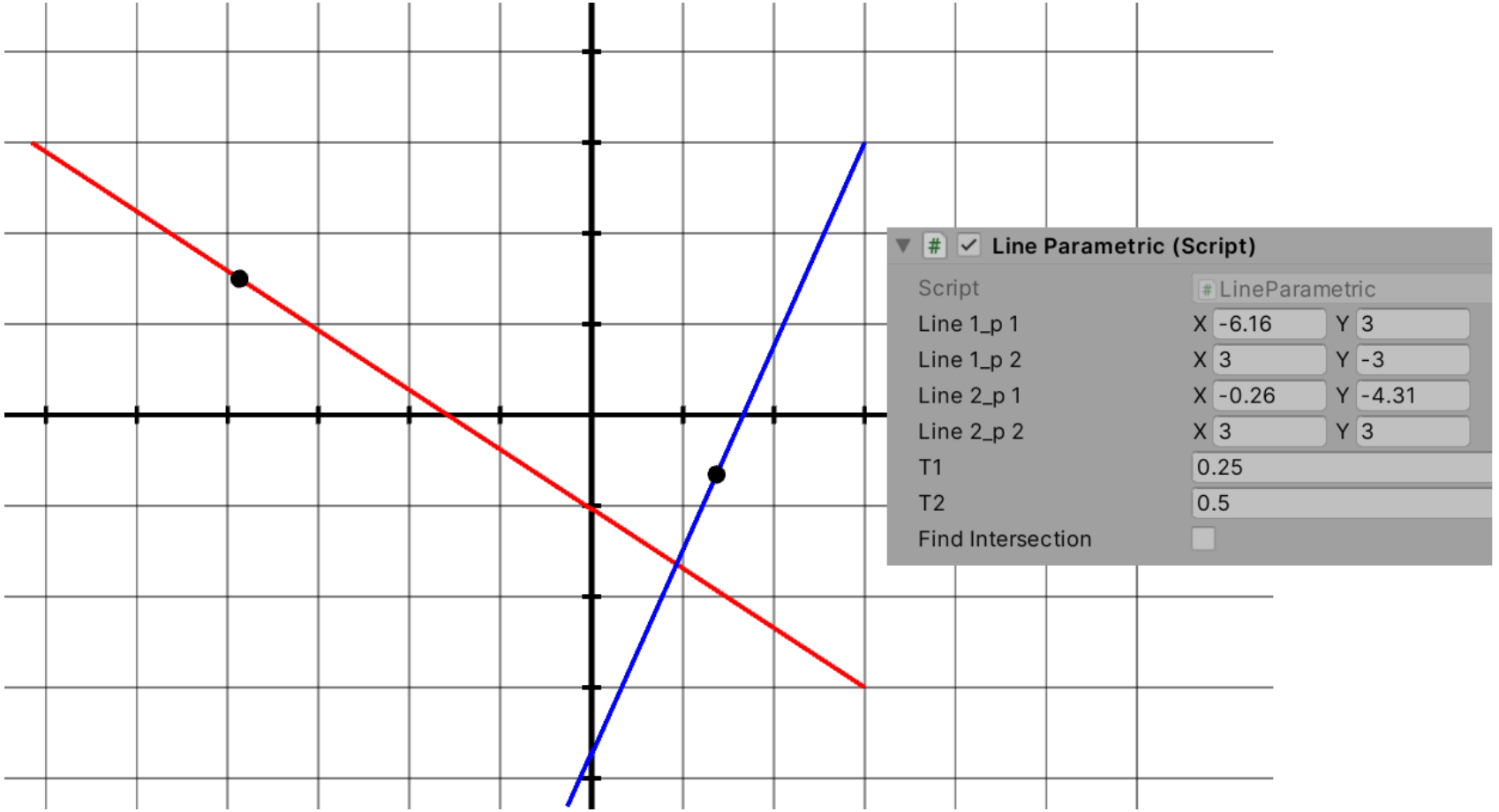
Line – Parametric Equation

- Let's take two points p_1 and p_2 and find a parametric line equation for them
- For p_0 we will take p_1
- Let's now find $\vec{v} = p_2 - p_1$. We decide not to normalize
- Now we have:

$$p(t) = p_1 + (p_2 - p_1)t \qquad t \in [0, 1]$$

  We have defined a segment from p_1 to p_2
- If $\vec{v}$ is normalized then t (for a segment) is in range from 0 to distance between p_1 and p_2

Line – Parametric Equation



Line – Parametric Equation

- Let’s find the intersection point
- We have two lines A and B:

$$A_p = A_{p_0} + A_{\vec{v}}\boldsymbol{A_t}$$
$$B_p = B_{p_0} + B_{\vec{v}}\boldsymbol{B_t}$$

$$A_{p_0} + A_{\vec{v}}\boldsymbol{A_t} = B_{p_0} + B_{\vec{v}}\boldsymbol{B_t}$$

- One equation and two unknowns $\boldsymbol{A_t}$ i $\boldsymbol{B_t}$?

Line – Parametric Equation

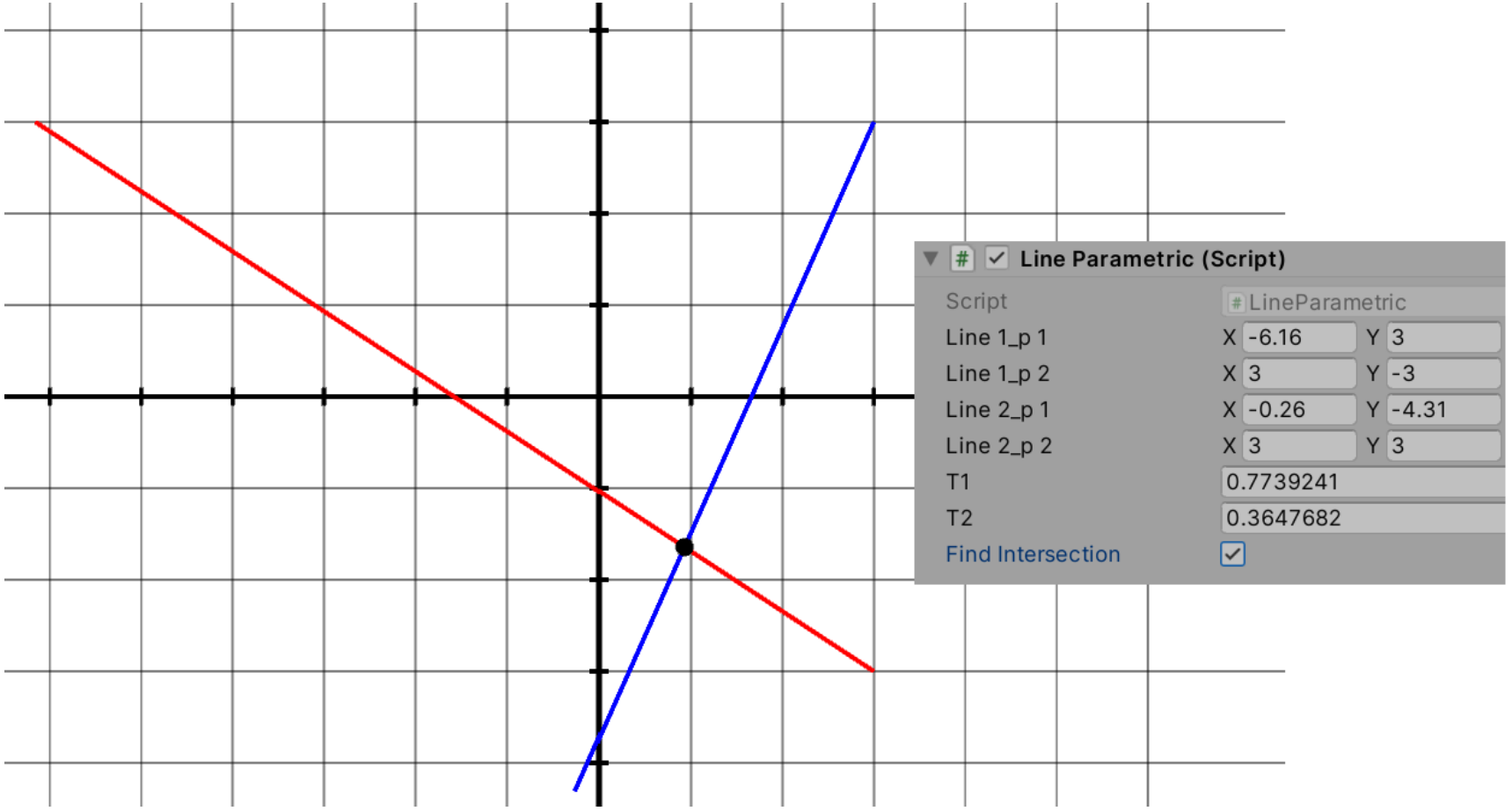
- There are actually two equations (one for x, and one for y):

$$\begin{cases} A_{p_{0x}} + A_{\vec{v}_x}\boldsymbol{A_t} = B_{p_{0x}} + B_{\vec{v}_x}\boldsymbol{B_t} \\ A_{p_{0y}} + A_{\vec{v}_y}\boldsymbol{A_t} = B_{p_{0y}} + B_{\vec{v}_y}\boldsymbol{B_t} \end{cases}$$

- Wolfram

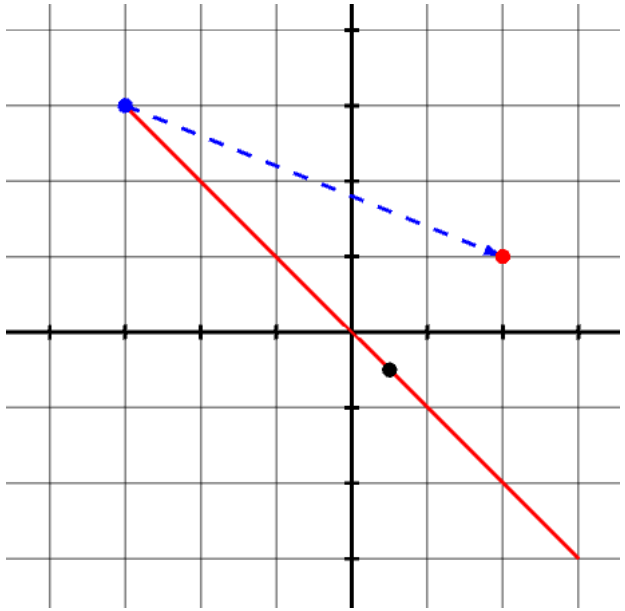
```
float a = line1_p1.x;
float b = line1_v.x;
float c = line2_p1.x;
float d = line2_v.x;
float e = line1_p1.y;
float f = line1_v.y;
float g = line2_p1.y;
float h = line2_v.y;
```

Line – Parametric Equation



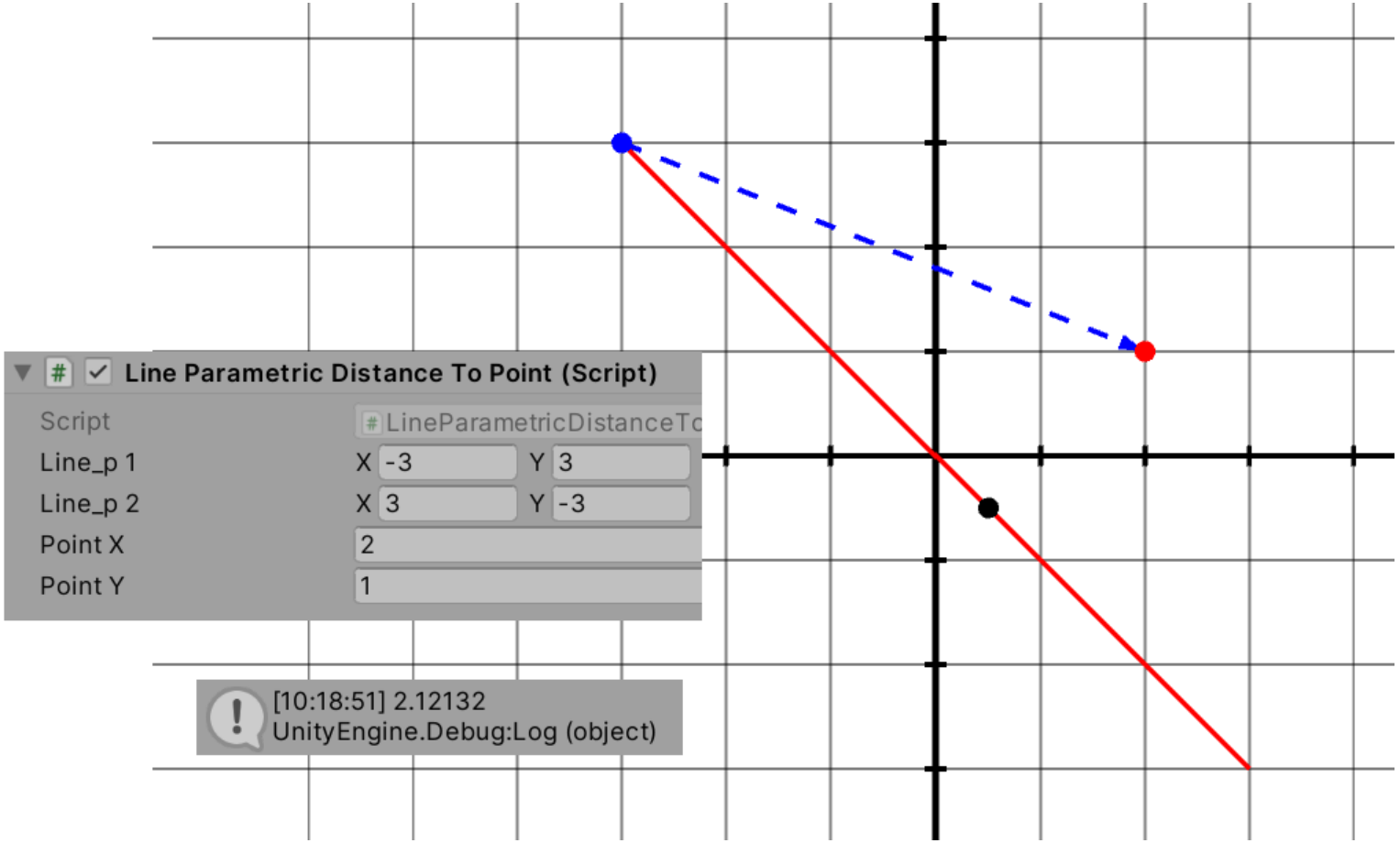
Line – Parametric Equation

- We will now find the distance of a point from a line (via vector projection):



- Please note that with the implicit equation that distance can be calculated directly from the equation

Line – Parametric Equation



Line – Parametric Equation

- It’s easy to generate points on a line/ray/segment
- Fairly easy to find intersection points of a line with different geometric objects
- A constant change in argument t leads to a „step” of the same distance on a line
- (2D) If the direction vector is not normalized, a line requires four values for representation. If normalized then three is enough
- Works in all dimensions!

Circle

- A **circle** is usually represented in two ways
- First is the **implicit equation**:

$$(x-a)^2+(y-b)^2=r^2$$

- Second is the system of two **parametric equations**:

$$\begin{cases} x(\theta) = a + r\cos(\theta) \\ y(\theta) = b + r\sin(\theta) \end{cases}$$

Circle – Implicit Equation

- The equation of form:

$$(x-a)^2+(y-b)^2=r^2$$

- (a,b) are the coordinates of the center of the circle, whose radius is r
- (x,y) are the coordinates of any point that belongs to the circle
- Alternatively we can write this as:

$$(x-a)^2+(y-b)^2-r^2=0$$

Circle – Implicit Equation

- Let’s find the intersecion point of a circle and a line
- The circle equation:

$$(x-a)^2+(y-b)^2=r^2$$

- The parametric line equation:

$$\begin{cases} x(t) = x_0 + \vec{v}_x t \\ y(t) = y_0 + \vec{v}_y t \end{cases}$$

- The intersection point of the line and circle is such a point, which (obviously) lies on both the line and circle

Circle – Implicit Equation

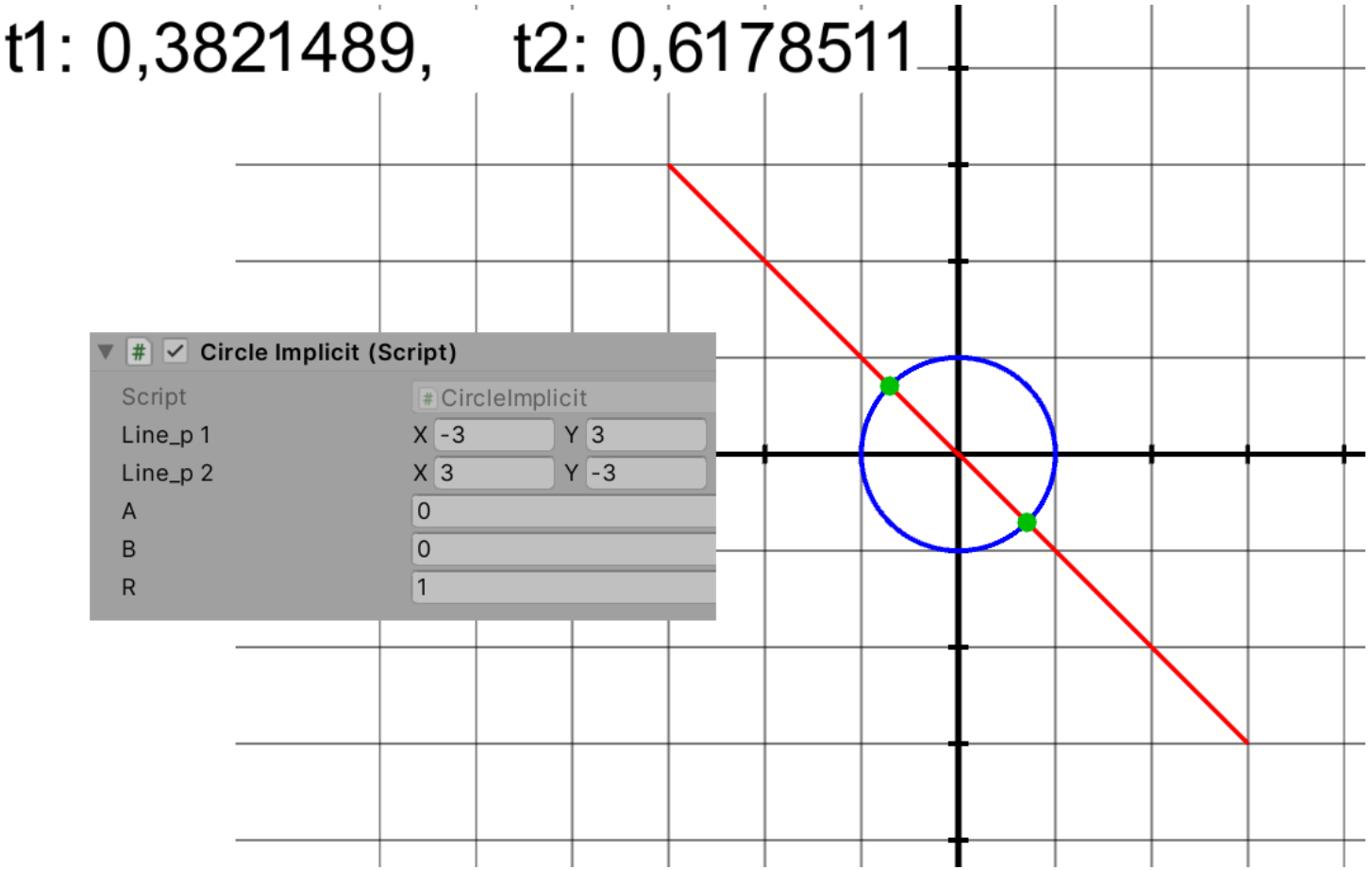
- We now substitute the coordinates of a point that lies on the line into the circle equation:

$$(x-a)^2+(y-b)^2=r^2$$

$$((x_0+\vec{v}_x\boldsymbol{t})-a)^2+((y_0+\vec{v}_y\boldsymbol{t})-b)^2=r^2$$

- We need to solve the above for t
- Wolfram. The result is not particularly pleasant...

Circle – Implicit Equation



Circle – Implicit Equation

- It’s easy to determine if a point is inside a circle or outside
- It’s not possible to use this form to calculate the distance of point from a circle!
- Points on a circle can be generated by solving the equation for one of the coordinates

Circle – Implicit Equation

- To generate points on a circle we can solve the implicit equation for y:

$$(x-a)^2+(y-b)^2=r^2$$

$$\left(\boldsymbol{y^2}\right)-(2b\boldsymbol{y})+(x^2+a^2-2ax+b^2-r^2)=0$$

- We then end up with two **explicit** functions $y=f(x)$, one of which represents the upper hemi-circle and the second one represents the bottom hemi-circle
- Wolfram
- We can also do the reverse – solve for x as a function of y

$$x=f(y)$$

Circle – Parametric Equation

- The equation of form:

$$\begin{cases} x(\theta) = a + r\cos(\theta) \\ y(\theta) = b + r\sin(\theta) \end{cases}$$

- The **parameter** is θ.
  This value in range $[0, 2\pi)$ gives us different coordinates (x, y) of points lying on a circle
- (a, b) is the center of a circle, and r is its radius
- This equation originates from polar coordinates

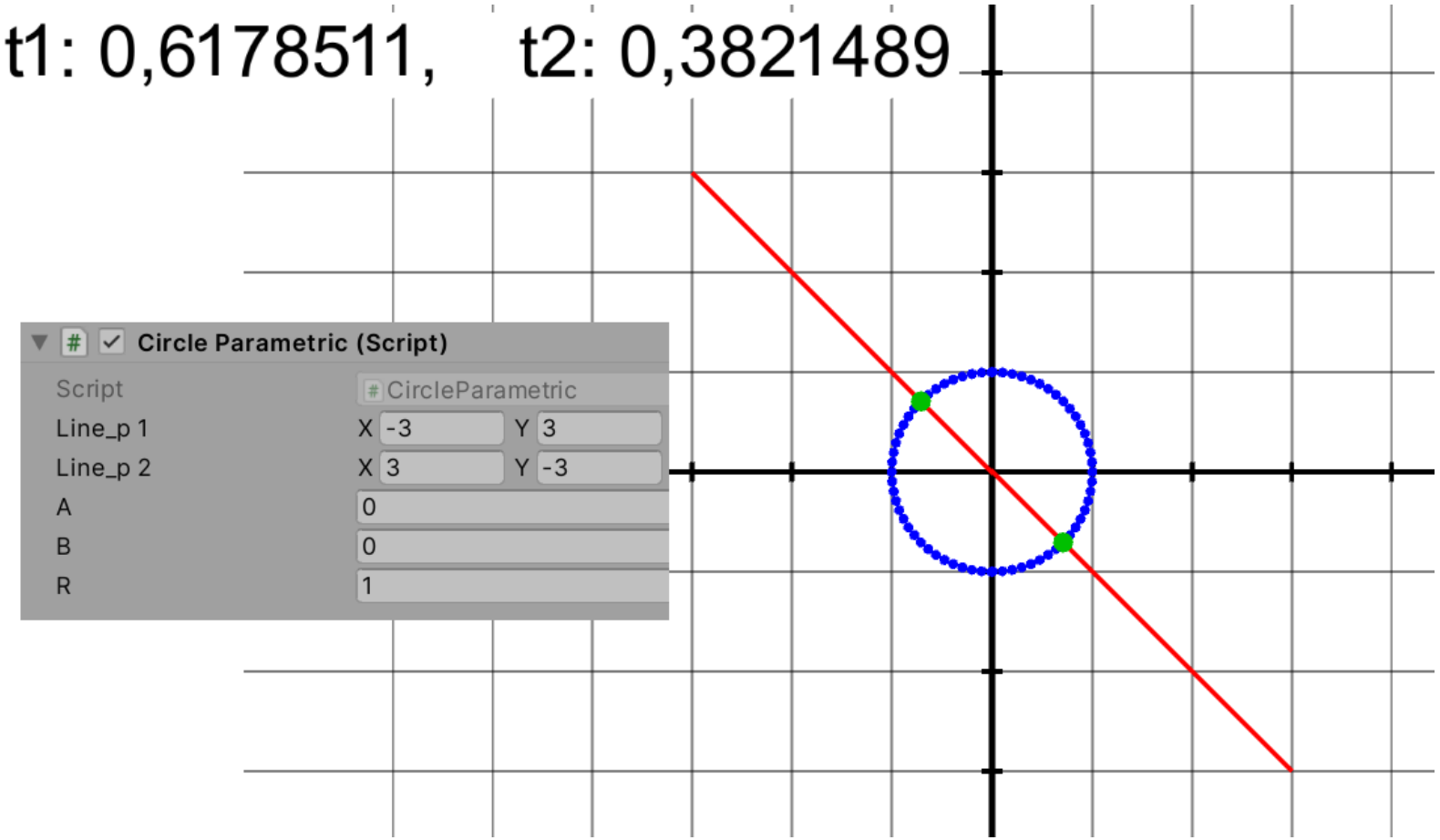
Circle – Parametric Equation

- We will now find the intersection point of a line with a parametric circle, by equating the parametric line equation to the circle parametric equation:

$$\begin{cases} x_0 + \vec{v}_x \boldsymbol{t} = a + r\cos(\boldsymbol{\theta}) \\ y_0 + \vec{v}_y \boldsymbol{t} = b + r\sin(\boldsymbol{\theta}) \end{cases}$$

- In this system the unknowns are t and θ
- Wolfram. Let's limit ourselves to t...

Circle – Parametric Equation



Circle – Parametric Equation

- It’s easy to generate points on a circle
- Calculation of intersection points with other objects can be involving
- A constant change in argument θ leads to a „step” of the same distance on a circle

Sphere

- A **sphere** is a „circle in 3D”
- The implicit equation is a natural extension of the circle equation:

$$(x-a)^2+(y-b)^2+(z-c)^2=r^2$$

- We already know the parametric equation – it’s the spherical coordinates:

$$\begin{cases} x(\theta,\varphi) = a + r\sin(\theta)\cos(\varphi) \\ y(\theta,\varphi) = b + r\sin(\theta)\sin(\varphi) \\ z(\theta,\varphi) = c + r\cos(\theta) \end{cases}$$

- The intersection point is found in the same way as it is with a circle and a 2D line

Plane

- Just like a circle/sphere, a **plane** can be represented in two different ways
- The first one is the **implicit equation**:

$$ax + by + cz + d = 0$$

- The second one is the **parametric equation**:

$$p(u, v) = o + u\vec{t} + v\vec{b}$$

Plane – Implicit Equation

- As a reminder this is the implicit equation of line:

$$ax + by + c = 0$$

- Implicit equation of plane:

$$ax + by + cz + d = 0$$

- A plane is a 3D object. In „some sense” it is an extension of line into 3D
- $[a, b, c]$ is the plane’s normal vector
  It doesn’t have to be normalized, although it usually is
- d is the distance of the plane from the origin

Plane – Implicit Equation

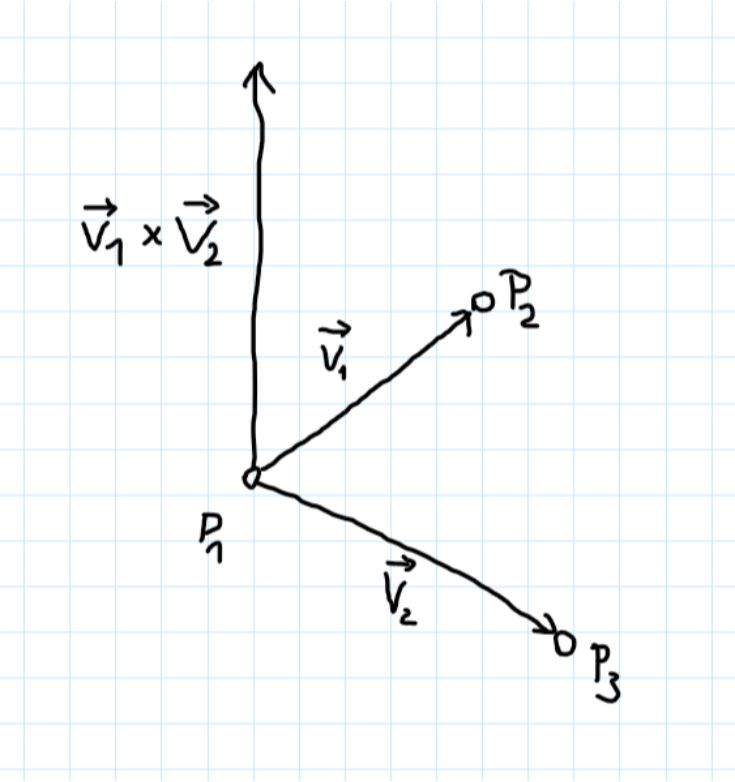
- Usually we have coordinates of a point belonging to a plane (p_x, p_y, p_z) and the plane's normal vector $[a, b, c]$
- Let's find d:

$$d = -(ap_x + bp_y + cp_z)$$

- If $[a, b, c]$ is normalized then $ax + by + cz + d$ gives us **signed distance** of point (x, y, z) from the plane

Plane – Implicit Equation

- Alternatively, instead of having coords of a point and the normal vector we might have coords of three (or more) points belonging to the plane
- In that case we can easily find the normal vector using the cross product




```
static public Vector3 GetNormal(Vector3 p1, Vector3 p2, Vector3 p3)
{
    return Vector3.Cross(p2 - p1, p3 - p1).normalized;
}
```

Plane – Implicit Equation

- We will find the intersection point of a plane and a line in 3D
- A 3D line is described using the following parametric equation:

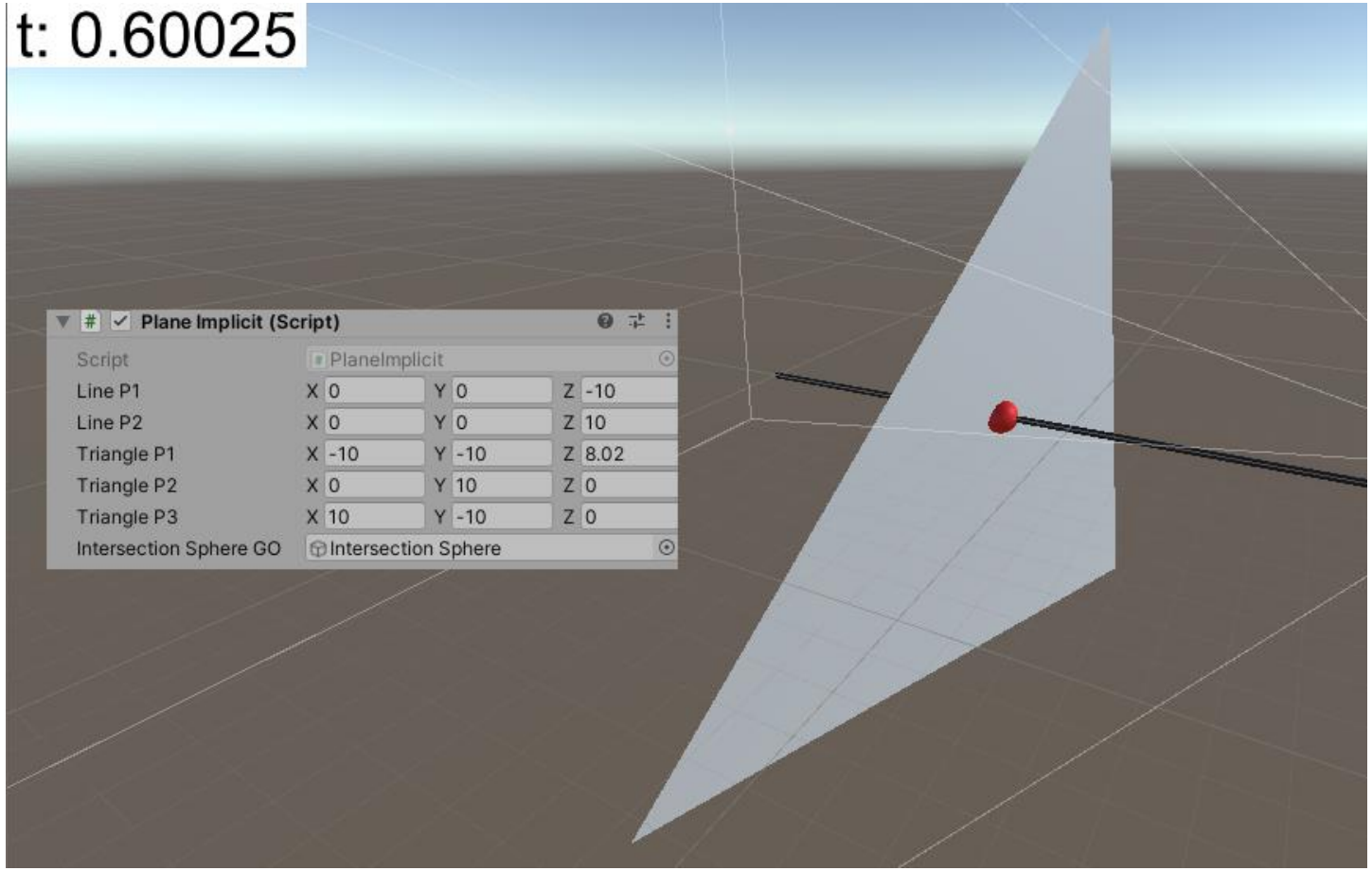
$$\begin{cases} x = x_0 + \vec{v}_x t \\ y = y_0 + \vec{v}_y t \\ z = z_0 + \vec{v}_z t \end{cases}$$

- We plug these into the plane equation:

$$a(x_0 + \vec{v}_x \boldsymbol{t}) + b\left(y_0 + \vec{v}_y \boldsymbol{t}\right) + c(z_0 + \vec{v}_z \boldsymbol{t}) + d = 0$$

- Wolfram

Plane – Implicit Equation



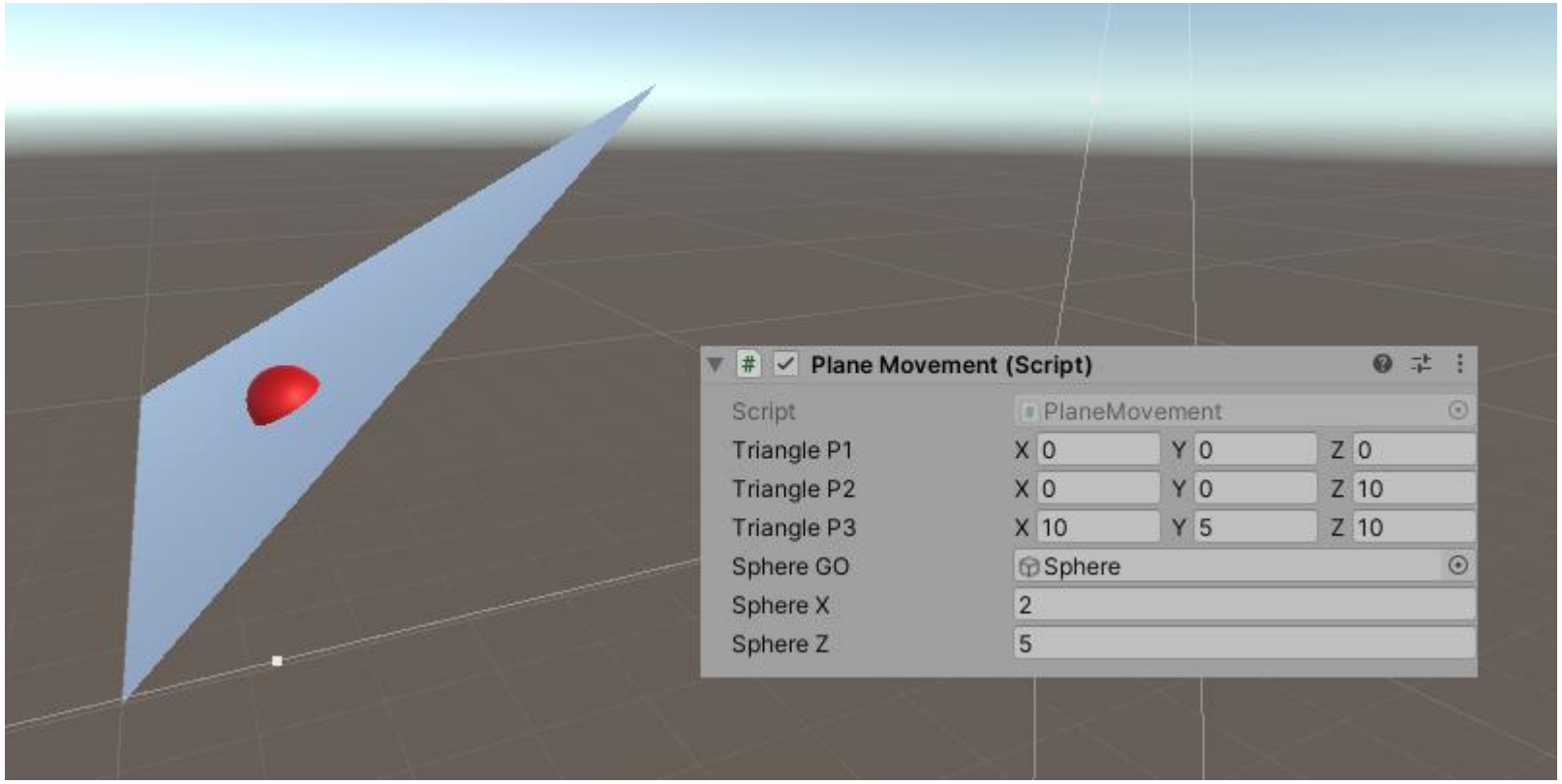
Plane – Implicit Equation

- Implicit plane equation makes it easy to implement movement on a plane
- Let's say we have (x, z) coords of a point and we want to find the height y on a plane
- In order to do that all we need to do is to solve for y the plane equation:

$$ax + by + cz + d = 0$$

$$y = y(x, z) = -\frac{(ax + cz + d)}{b}$$

Plane – Implicit Equation



Plane – Implicit Equation

- We've discussed the implicit plane equation in the form:

$$ax + by + cz + d = 0$$

- It's worth knowing though that quite often we can find this equation written as:

$$Ax + By + Cz + D = 0$$

Plane – Implicit Equation

- It’s easy to determine on which side of a plane a point is
- It’s easy to find the distance of a point from a plane
- Points on a plane can be generated by solving the equation for one of the coordinates

Plane – Parametric Equation

- As a reminder here is the parametric line equation:

$$p(t) = p_0 + \vec{v}t$$

- Again, a plane is in some sense an extension of line into 3D
- The parametric plane equation:

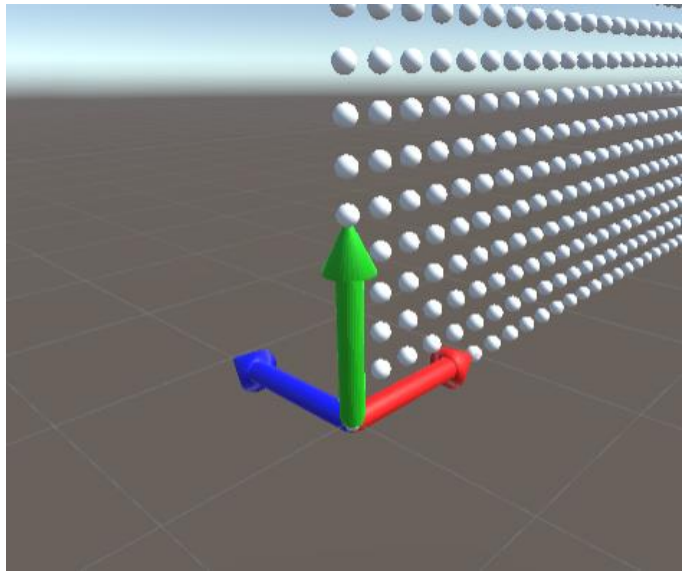
$$p(u, v) = o + u\vec{t} + v\vec{b}$$

- o are coordinates of some fixed (arbitrary) point on the plane
- $\vec{t}$ and $\vec{b}$ to are the plane's **tangent vectors**
- u and v are parameters, which we use to generate points on the plane

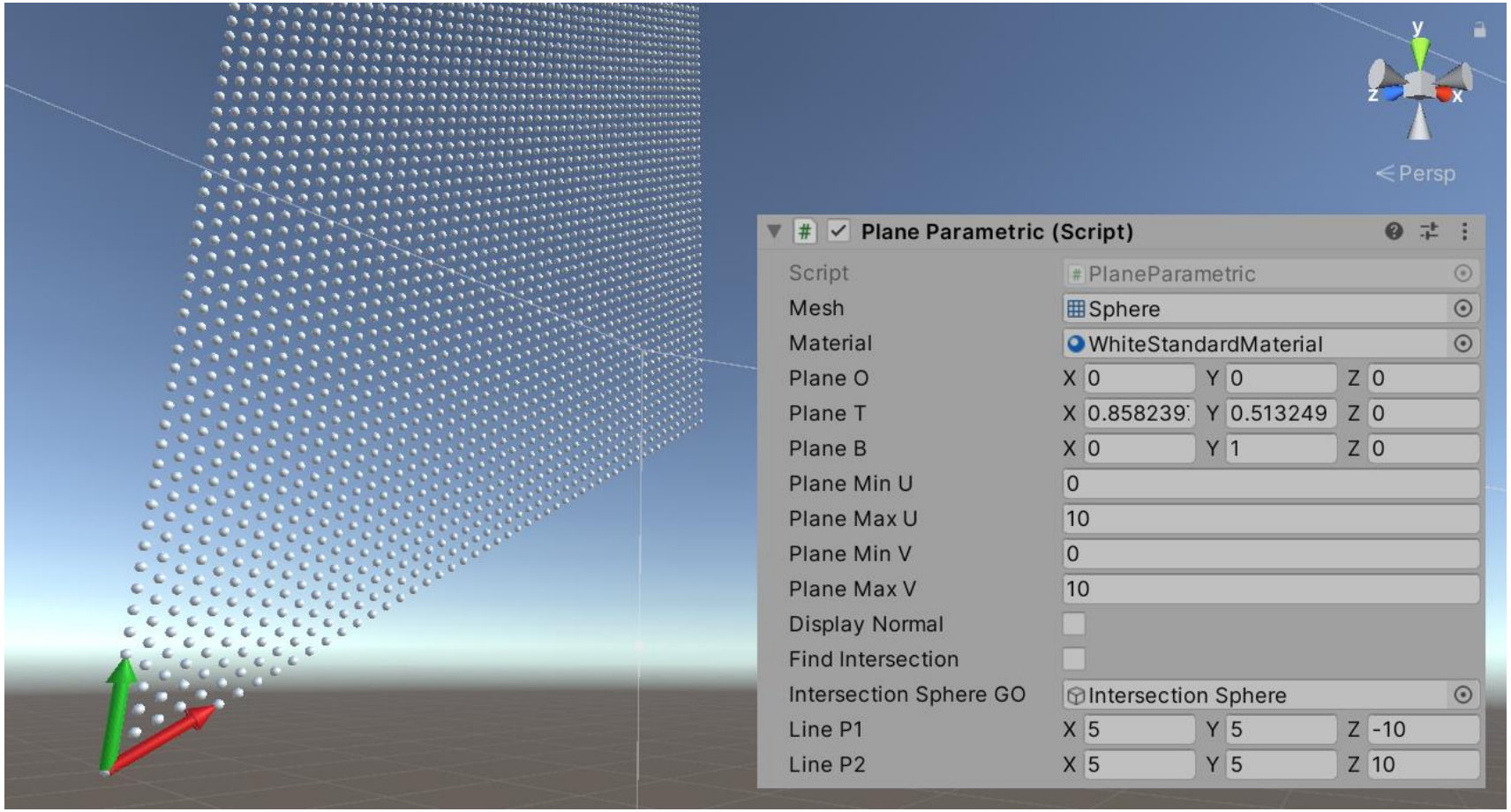
Plane – Parametric Equation

- The parametric equation is defined using two tangent vectors: $\vec{t}$ (tangent vector) and $\vec{b}$ (bitangent vector, sometimes mistakenly called „binormal”)
- The normal vector can be determined by taking the cross product of the tangent vectors:

$$\vec{n} = \vec{t} \times \vec{b}$$



Plane – Parametric Equation



Plane – Parametric Equation

- Let’s find the intersection point of a plane and a line in 3D
- A line in 3D is described with the parametric equation:

$$\begin{cases} x = x_0 + \vec{v}_x t \\ y = y_0 + \vec{v}_y t \\ z = z_0 + \vec{v}_z t \end{cases}$$

- The parametric plane equation broken down into components is:

$$\begin{cases} x = o_x + u\vec{t}_x + v\vec{b}_x \\ y = o_y + u\vec{t}_y + v\vec{b}_y \\ z = o_z + u\vec{t}_z + v\vec{b}_z \end{cases}$$

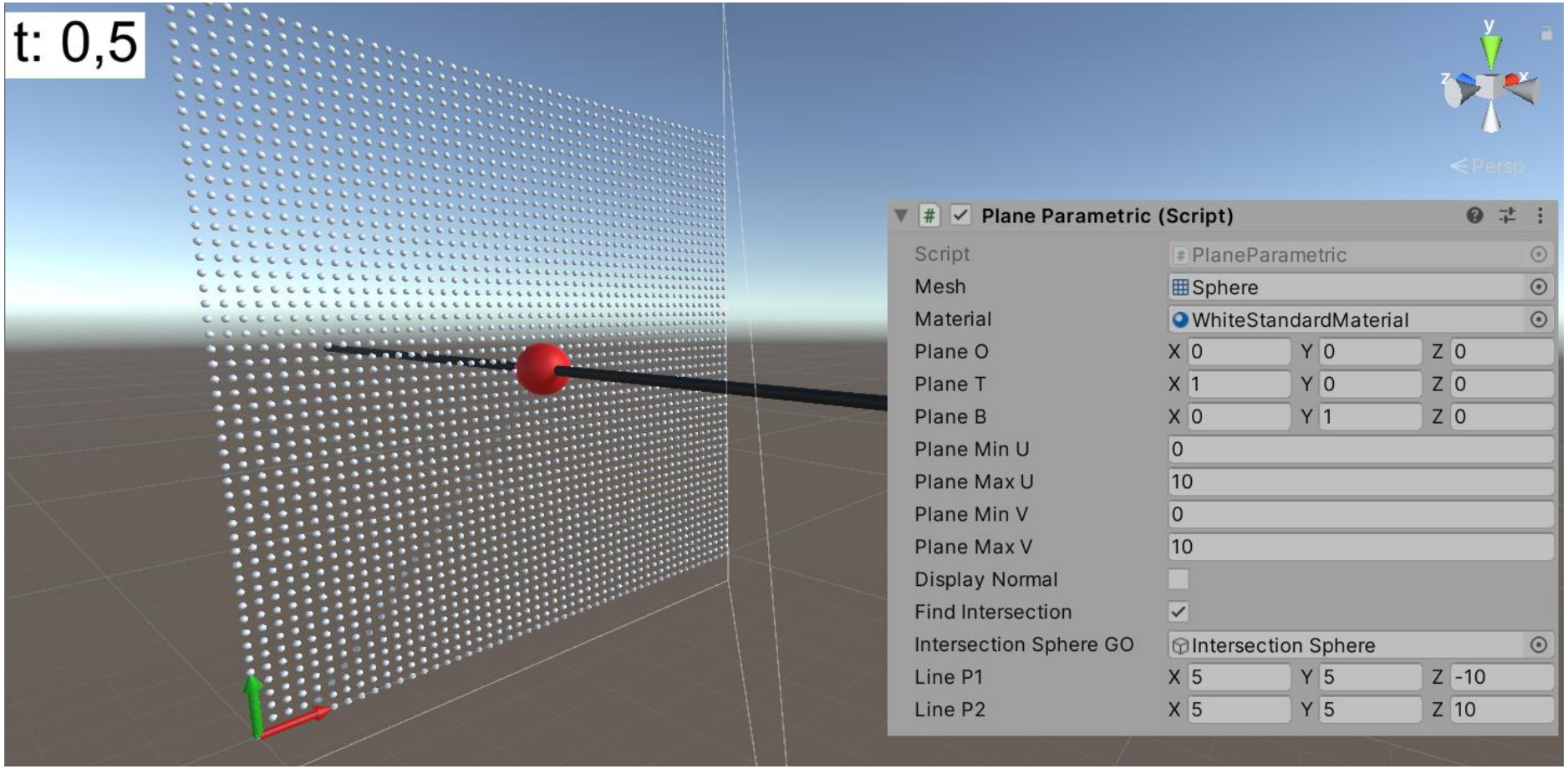
Plane – Parametric Equation

- We equate the coordinates, coming up with a system:

$$\begin{cases} x_0 + \vec{V}_x \boldsymbol{t} = o_x + \boldsymbol{u}\vec{T}_x + \boldsymbol{v}\vec{B}_x \\ y_0 + \vec{V}_y \boldsymbol{t} = o_y + \boldsymbol{u}\vec{T}_y + \boldsymbol{v}\vec{B}_y \\ z_0 + \vec{V}_z \boldsymbol{t} = o_z + \boldsymbol{u}\vec{T}_z + \boldsymbol{v}\vec{B}_z \end{cases}$$

- Wolfram. We only need t

Plane – Parametric Equation



Plane – Parametric Equation

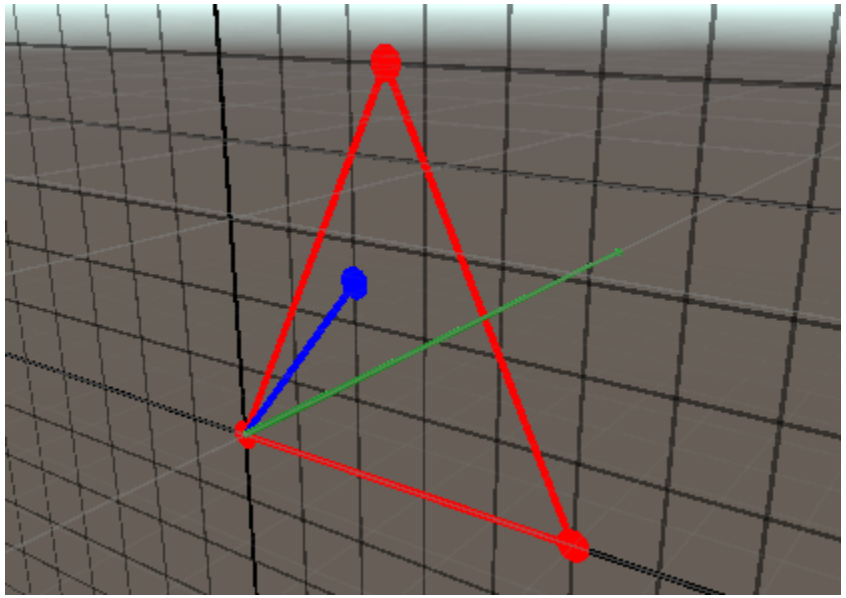
- Easy to generate points on a plane
- Calculating intersection points with other objects can be involving
- A constant change in arguments u and v leads to a „step” of the same distance along the tangent vectors

Triangle

- So far we have learned to find intersection points of a line with various primitives
- We have not discussed though how to find the intersection point between a line and a **triangle**, one of the most commonly used 3D primitives
- A triangle always lies in one plane. So we construct a plane equation (from the coords of the points that make up the triangle) and we are looking for the intersection with the line
- When we've found the intersection all we need to do is check if the intersection point lies within the triangle, i.e. if it is bounded by all three sides of the triangle

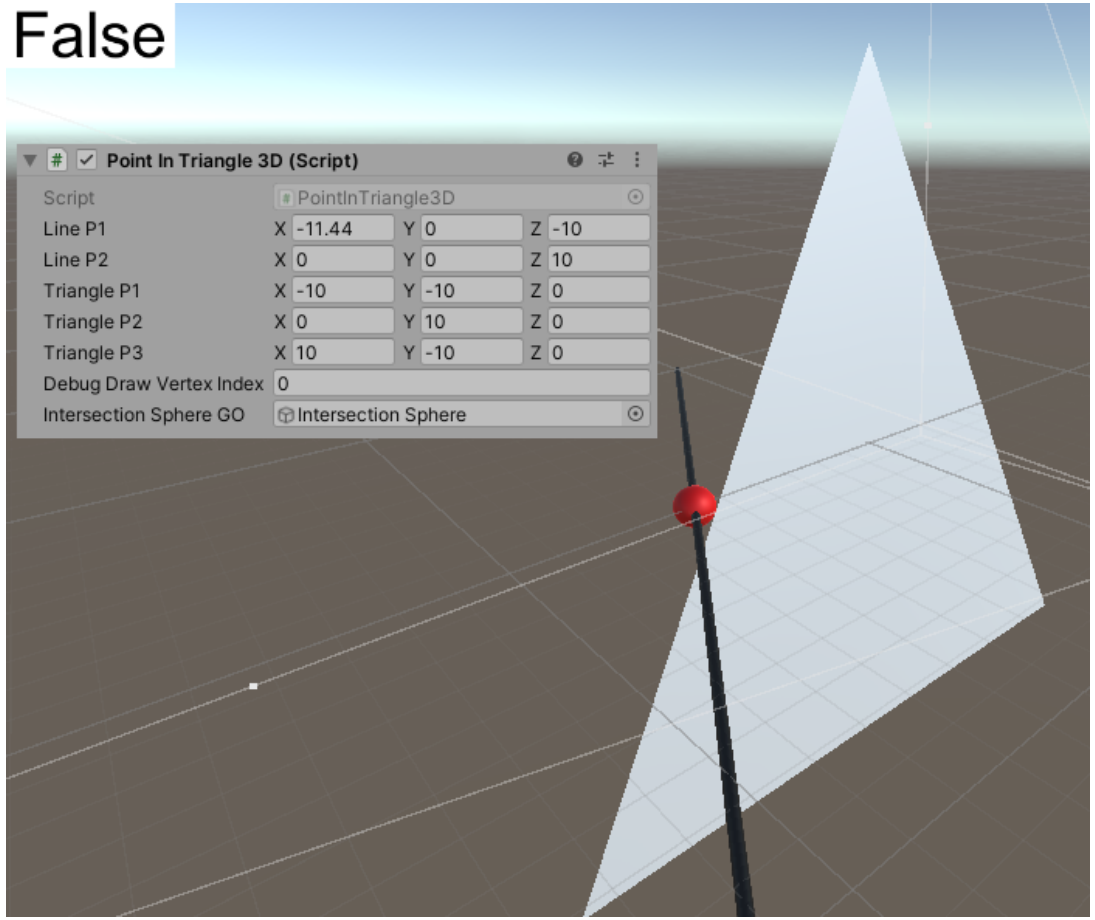
Triangle

- In chapter 3 we discussed a program where we checked if a point is inside a triangle in 2D:



- We can check the 3D case in a very similar way

Triangle



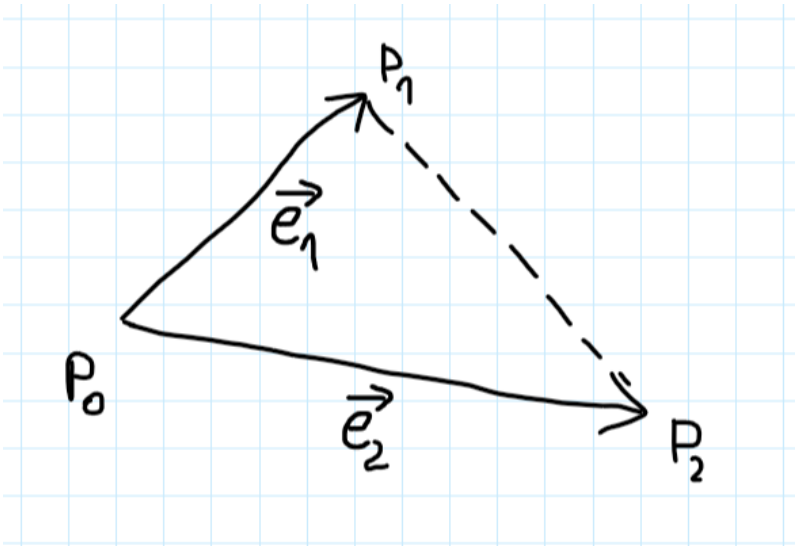
Triangle – Barycentric Coordinates

- A concept that is worth discussing while we are talking about triangles are **barycentric coordinates**
- Let's say we have a triangle with points p_0, p_1 and p_2.
  Barycentric coordinates (b_1, b_2) let us select any point p that lies in the triangle's plane:

$$p(b_1, b_2) \quad = \quad p_0 + b_1(p_1 - p_0) + b_2(p_2 - p_0) \quad = \quad p_0 + b_1\vec{e}_1 + b_2\vec{e}_2$$

$$\vec{e}_1 = p_1 - p_0$$
$$\vec{e}_2 = p_2 - p_0$$

- Vectors $\vec{e}_1$ and $\vec{e}_2$ are known as **edge vectors**



Triangle – Barycentric Coordinates

- The definition of barycentric coordinates is very similar to that of the parametric plane equation:

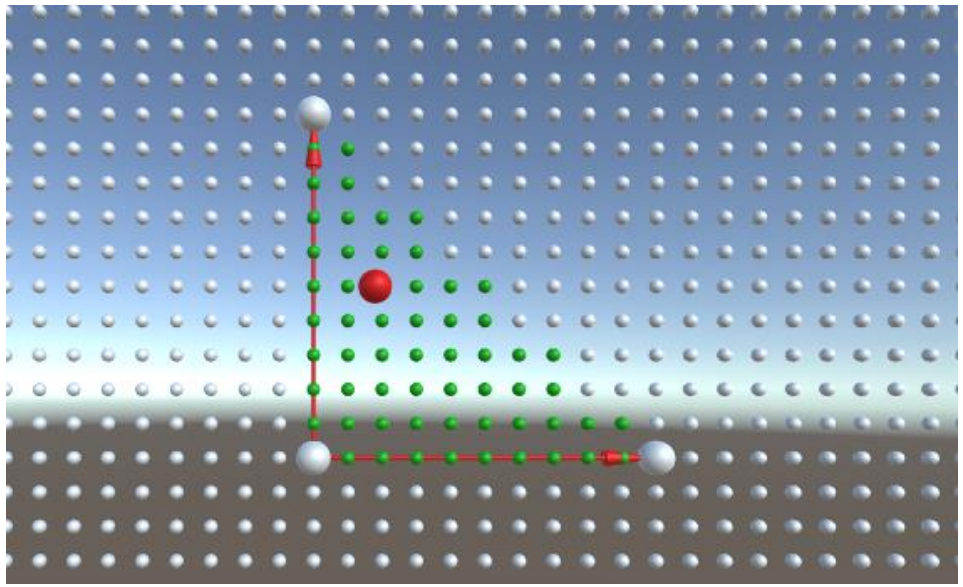
$$p(u, v) = o + u\vec{t} + v\vec{b}$$

- The main difference comes from the fact that vectors $\vec{t}$ and $\vec{b}$ are usually normalized and the only point of reference is the plane's fixed point o.
  On the other hand with barycentric coordinates the vectors $\vec{e}_1$ and $\vec{e}_2$ additionally, implicitly define points p_1 oraz p_2
- In some sense it's the barycentric coordinates (b_1, b_2) that are „normalized"

Triangle – Barycentric Coordinates

- The fact that barycentrics are „normalized” lets us specify generic constraints, whose meeting can determine if a point lies inside a triangle
- Point p with barycentric coordinates (b_1, b_2) is inside a triangle if:

$$b_1 \geq 0 \qquad b_2 \geq 0 \qquad b_1 + b_2 \leq 1$$



Triangle – Barycentric Coordinates – Solution in 2D

- We will now derive a formula for calculating barycentric coordinates (b_1, b_2) of a given point p, what will let us check if the point is inside a triangle.
  We will start with the 2D case
- Our goal is to solve the following system and find b_1 and b_2:

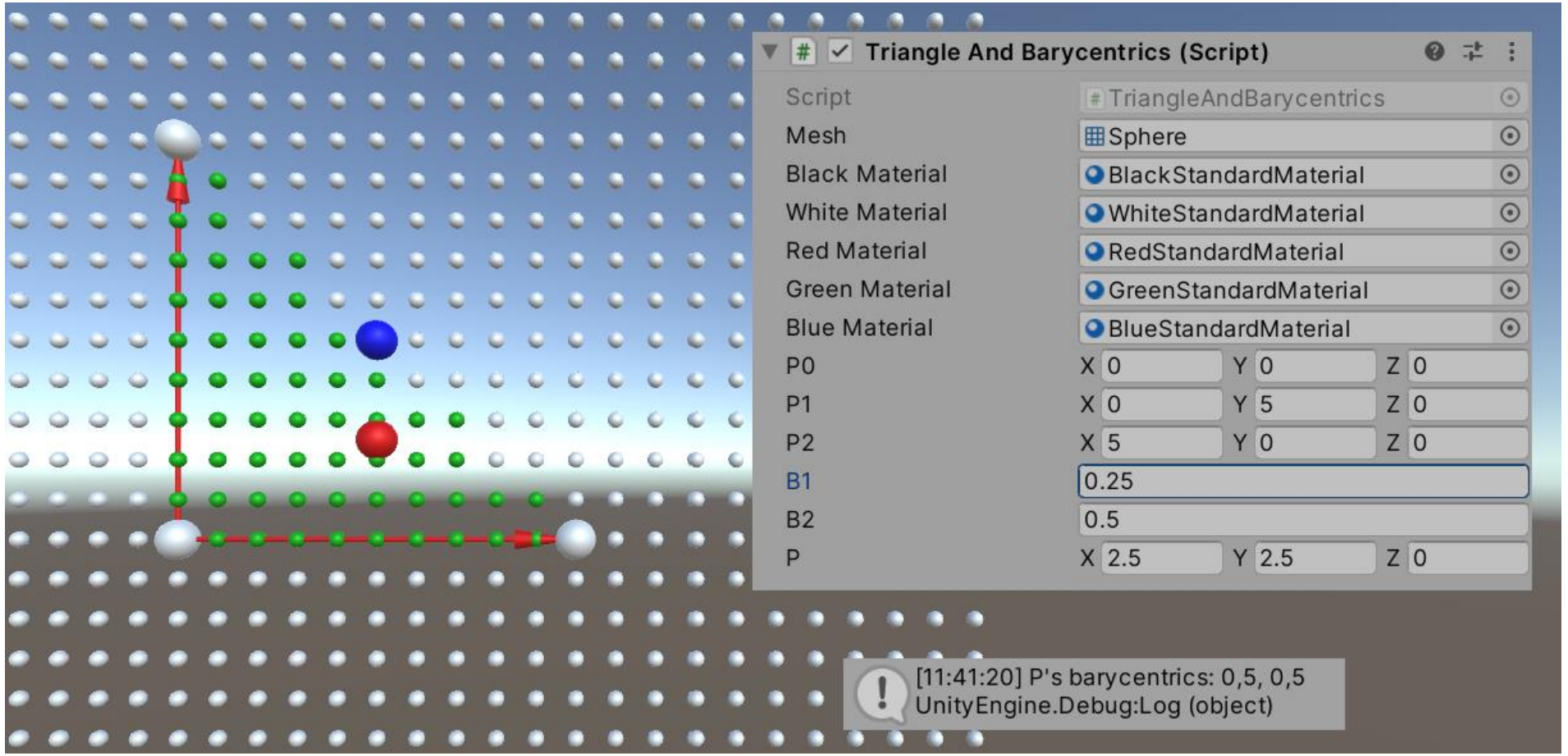
$$\begin{cases} p_x = p_{0x} + \boldsymbol{b_1}(p_{1x} - p_{0x}) + \boldsymbol{b_2}(p_{2x} - p_{0x}) \\ p_y = p_{0y} + \boldsymbol{b_1}(p_{1y} - p_{0y}) + \boldsymbol{b_2}(p_{2y} - p_{0y}) \end{cases}$$

- Wolfram

```
private Vector2 BarycentricsXY(Vector3 p0, Vector3 p1, Vector3 p2, Vector3 p)
{
    // https://www.wolframalpha.com/input?i=a_1+%2B+u*%28b_1-a_1%29+%2B+v*%28c_1-a_1%29+%3D+d_1%2C+a_2+%2B+u*%28b_2-a_2%29+%
    float a1 = p0.x;
    float b1 = p1.x;
    float c1 = p2.x;
    float d1 = p.x;
    float a2 = p0.y;
    float b2 = p1.y;
    float c2 = p2.y;
    float d2 = p.y;
    //
    float u = (a2 * (c1 - d1) + a1 * (d2 - c2) + c2 * d1 - c1 * d2) / (a2 * (c1 - b1) + a1 * (b2 - c2) - b2 * c1 + b1 * c2);
    float v = (a2 * (d1 - b1) + a1 * (b2 - d2) - b2 * d1 + b1 * d2) / (a2 * (c1 - b1) + a1 * (b2 - c2) - b2 * c1 + b1 * c2);

    return new Vector2(u, v);
}
```

Triangle – Barycentric Coordinates – Solution in 2D



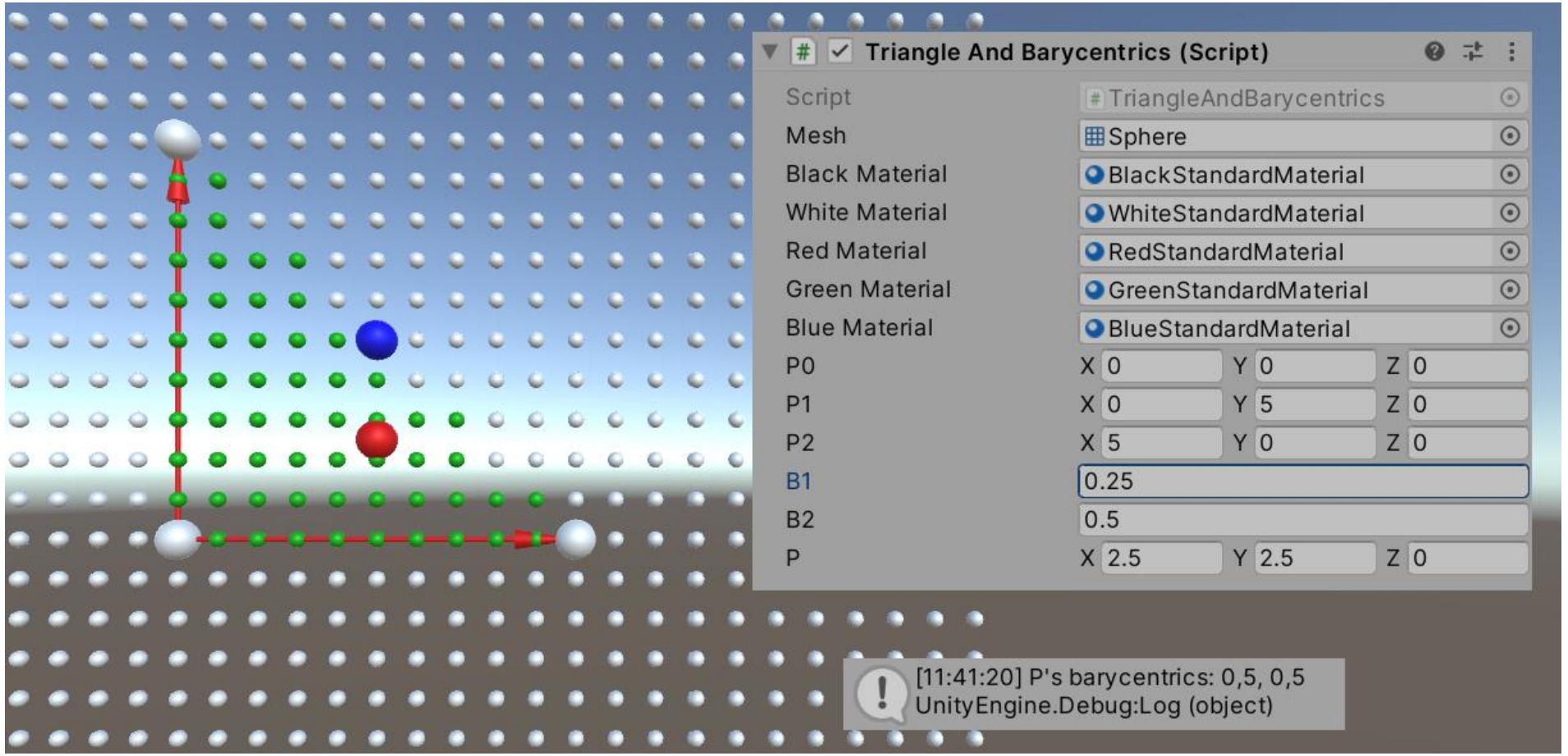
Triangle – Barycentric Coordinates – Solution in 3D

- Let's now solve the same problem in 3D:

$$\begin{cases} p_x = p_{0x} + \boldsymbol{b_1}(p_{1x} - p_{0x}) + \boldsymbol{b_2}(p_{2x} - p_{0x}) \\ p_y = p_{0y} + \boldsymbol{b_1}(p_{1y} - p_{0y}) + \boldsymbol{b_2}(p_{2y} - p_{0y}) \\ p_z = p_{0z} + \boldsymbol{b_1}(p_{1z} - p_{0z}) + \boldsymbol{b_2}(p_{2z} - p_{0z}) \end{cases}$$

- Here we have 3 equations and 2 unknowns
- This „incompatibility" results from the fact that barycentrics „live in 2D" – they determine coords of points that all lie on one plane in 3D. Hence the right hand side of the above equations can only express points lying in that plane
- On the left hand side of the equations above is any point in „full" 3D
- In practice this means that to find (b_1, b_2) we can use any 2 equations of the 3, but we have to pick them with consideration...

Triangle – Barycentric Coordinates – Solution in 3D



Triangle – Barycentric Coordinates – Solution in 3D

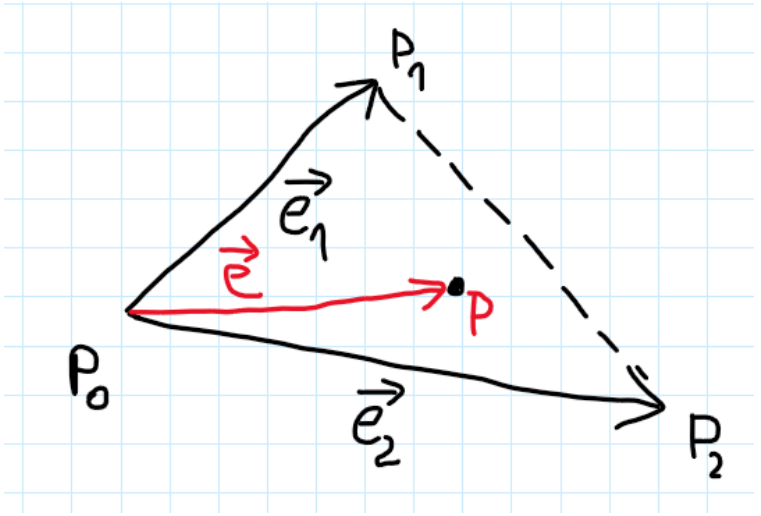
- We will now again derive a solution in 3D, but differently:

$$p(b_1, b_2) = p_0 + b_1\vec{e}_1 + b_2\vec{e}_2$$

$$p - p_0 = b_1\vec{e}_1 + b_2\vec{e}_2 \qquad \vec{e} = p - p_0$$

$$\vec{e} = b_1\vec{e}_1 + b_2\vec{e}_2 \mid \circ \vec{e}_1/\vec{e}_2$$

$$\begin{cases} \vec{e} \circ \vec{e}_1 = \boldsymbol{b_1}\vec{e}_1 \circ \vec{e}_1 + \boldsymbol{b_2}\vec{e}_2 \circ \vec{e}_1 \\ \vec{e} \circ \vec{e}_2 = \boldsymbol{b_1}\vec{e}_1 \circ \vec{e}_2 + \boldsymbol{b_2}\vec{e}_2 \circ \vec{e}_2 \end{cases}$$



- We ended up with a system of two equations
- We „created” those two equations out of one (actually we made two out of three...)

Triangle – Barycentric Coordinates – Solution in 3D

- Let's solve ($u = b_1$ and $v = b_2$):

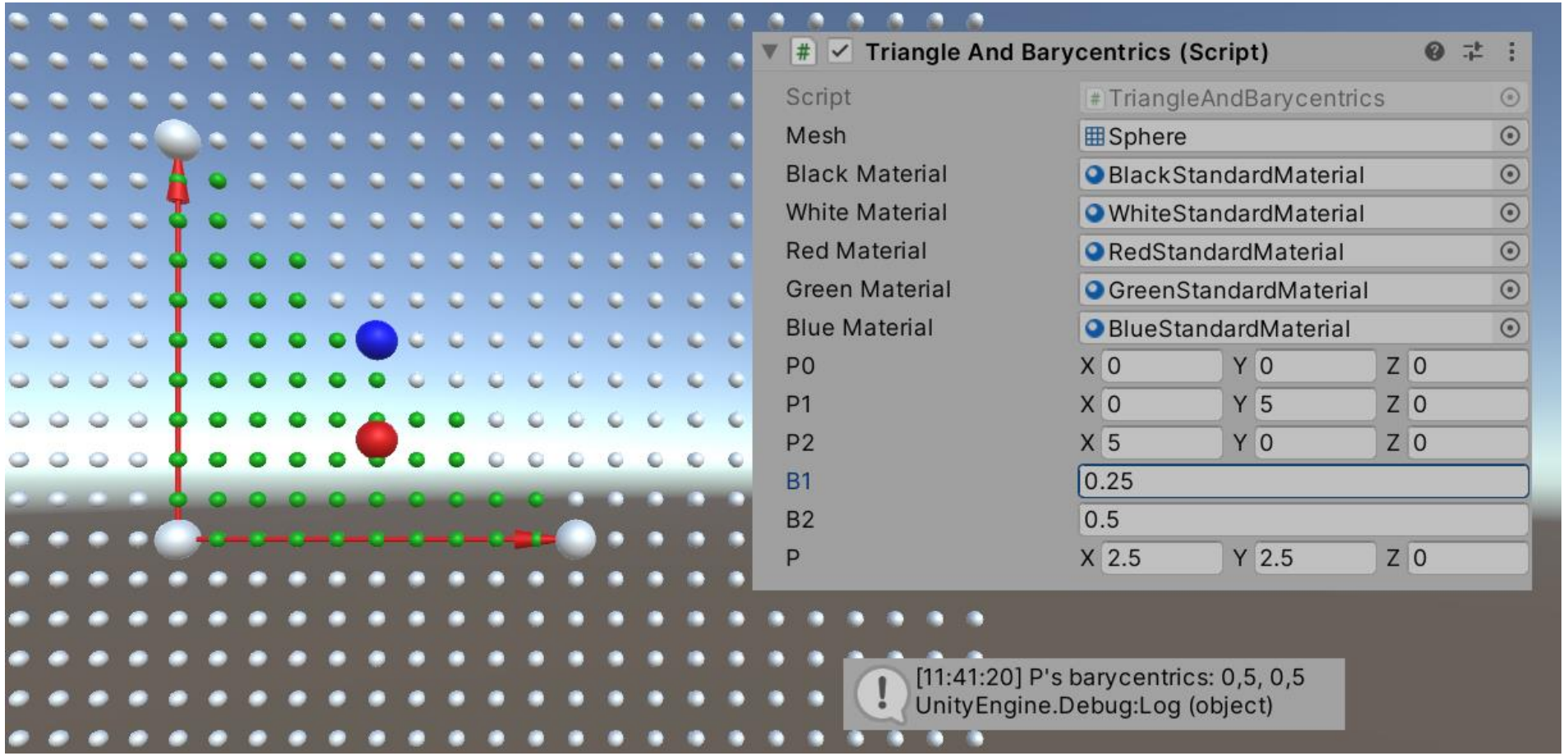
$$\boldsymbol{u}A + \boldsymbol{v}B = C$$
$$\boldsymbol{u}D + \boldsymbol{v}E = F$$

$$A = \vec{e}_1 \circ \vec{e}_1 \qquad B = \vec{e}_2 \circ \vec{e}_1 \qquad C = \vec{e} \circ \vec{e}_1$$

$$D = \vec{e}_1 \circ \vec{e}_2 \qquad E = \vec{e}_2 \circ \vec{e}_2 \qquad F = \vec{e} \circ \vec{e}_2$$

- Wolfram

Triangle – Barycentric Coordinates – Solution in 3D



Triangle – Barycentric Coordinates – Intersection with Parametric Line

- Let's now plug, on the left hand side, coords of a point that belongs to a parametric line:

$$\begin{cases} x_0 + \vec{v}_x \boldsymbol{t} = p_{0x} + \boldsymbol{b_1}(p_{1x} - p_{0x}) + \boldsymbol{b_2}(p_{2x} - p_{0x}) \\ y_0 + \vec{v}_y \boldsymbol{t} = p_{0y} + \boldsymbol{b_1}(p_{1y} - p_{0y}) + \boldsymbol{b_2}(p_{2y} - p_{0y}) \\ z_0 + \vec{v}_z \boldsymbol{t} = p_{0z} + \boldsymbol{b_1}(p_{1z} - p_{0z}) + \boldsymbol{b_2}(p_{2z} - p_{0z}) \end{cases}$$

- Now we have exactly 3 equations and 3 unknowns because both on the right and left hand side the point is „constrained"
- On the left hand side we have some point on a line, and on the right hand side we have some point on a plane
- This system solves simultaneously for the intersection point (parameter t) and barycentric coordinates of that point (b_1, b_2)

Triangle – Barycentric Coordinates – Intersection with Parametric Line

- We covered an identical problem when we discussed the parametric plane equation:

$$\begin{cases} x_0 + \vec{V}_x \boldsymbol{t} = o_x + \boldsymbol{u}\vec{T}_x + \boldsymbol{v}\vec{B}_x \\ y_0 + \vec{V}_y \boldsymbol{t} = o_y + \boldsymbol{u}\vec{T}_y + \boldsymbol{v}\vec{B}_y \\ z_0 + \vec{V}_z \boldsymbol{t} = o_z + \boldsymbol{u}\vec{T}_z + \boldsymbol{v}\vec{B}_z \end{cases}$$

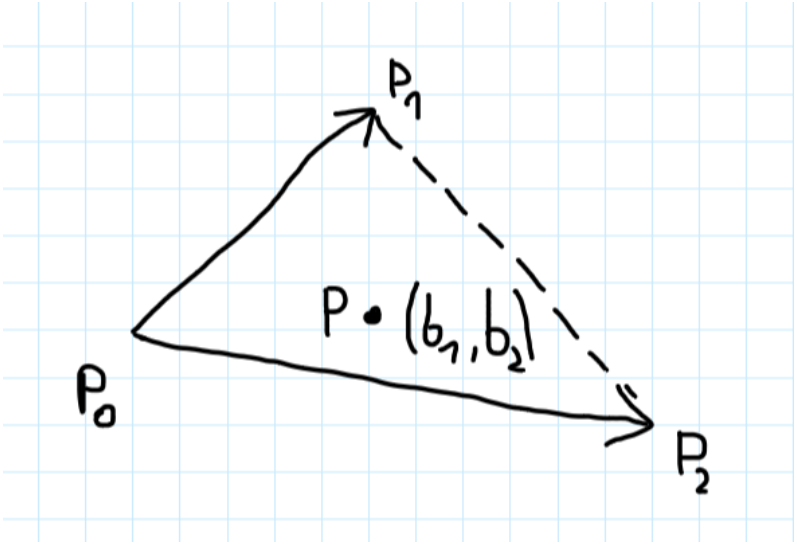
  The difference here is that vectors $\vec{T}$ and $\vec{B}$ are (usually) normalized

- Möller-Trumbore Algorithm is an efficient implementation of the solution to the above system

Triangle – Barycentric Coordinates – Attributes Interpolation

- An important bonus that comes from using barycentric coordinates is that we can easily calculate/interpolate various parameters/attributes within a triangle
- We've got barycentric coords (b_1, b_2) of some point p
- We may also have values of some attributes (like color or normal vector) denoted as a_0, a_1, a_2 at vertices p_0, p_1 and p_2
- We can now calculate the value of attribute a at point p as:

$$a = a_0 + b_1(a_1 - a_0) + b_2(a_2 - a_0)$$



Triangle – Barycentric Coordinates

- It's worth knowing that sometimes barycentric coords are derived from the following:

$$p(b_0, b_1, b_2) = b_0 p_0 + b_1 p_1 + b_2 p_2 \qquad b_0 + b_1 + b_2 = 1$$

- Point p is simply a weighted average of points p_0, p_1, p_2
- A point outside the triangle will have one of those three coordinates negative
- This definition is equivalent with the previous one:

$$p(b_0, b_1, b_2) = (1 - b_1 - b_2)p_0 + b_1 p_1 + b_2 p_2$$

$$p(b_0, b_1, b_2) = p_0 + b_1(p_1 - p_0) + b_2(p_2 - p_0)$$

Triangle – Barycentric Coordinates

- Pretty much every solid book on math for 3D games covers barycentric coordinates
- The first 3D algorithm that we covered here comes from the book 3D Math Primer for Graphics and Game Development
- The second 3D algorithm that we covered is discussed in books Mathematics for 3D Game Programming and Computer Graphics and Real-Time Collision Detection
- Overall these three books cover barycentric coordinates in great detail

Implicit Geometry

- On a few examples we've learned how implicit geometry is defined
- Geometry in 2D is defined like this:

$$f(x, y) = c$$

  where typically $c = 0$
- Geometry in 3D is defined similarly:

$$f(x, y, z) = c$$

Implicit Geometry

- Note that for example the implicit plane equation:

$$ax + by + cz + d = 0$$

  is actually a function:

$$f(x, y, z) = ax + by + cz + d$$

- We can also make it explicit (and loose some freedom!):

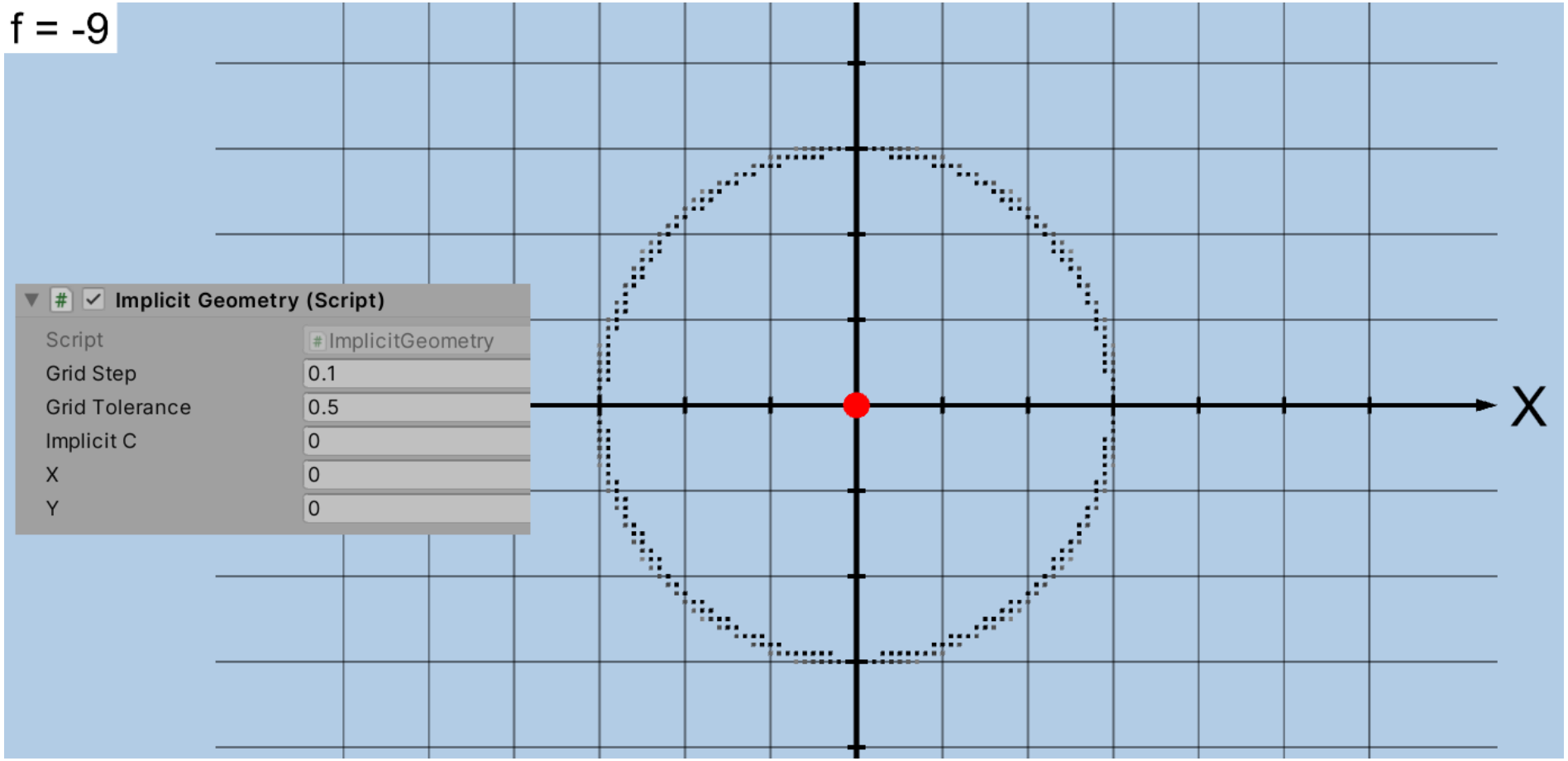
$$ax + by + cz + d = 0$$

$$z(x, y) = -\frac{ax + by + d}{\boldsymbol{c}}$$

Implicit Geometry

- In chapter 8 we will learn how to calculate normal vectors for generic implicit geometry
- Typically implicit definition allows us to work with more „freeform” geometry.
  It makes many operations, like smooth blending of geometry, much easier
- Usually visualization is the hard part (turning into explicit works only for simple cases).
  One way to do it is to **polygonize** the implicit geometry using
  marching squares algorithm (in 2D) or
  marching cubes (in 3D)
- Book Fundamentals of Computer Graphics has a great entire chapter on that subject

Implicit Geometry



Equations Optimization

- Some formulas in this chapter were quite complicated
- For example, to find the intersection point of a parametric line with an implicit circle required us to solve the following:

$$((x_0 + \vec{v}_x \boldsymbol{t}) - a)^2 + ((y_0 + \vec{v}_y \boldsymbol{t}) - b)^2 = r^2$$

  for t

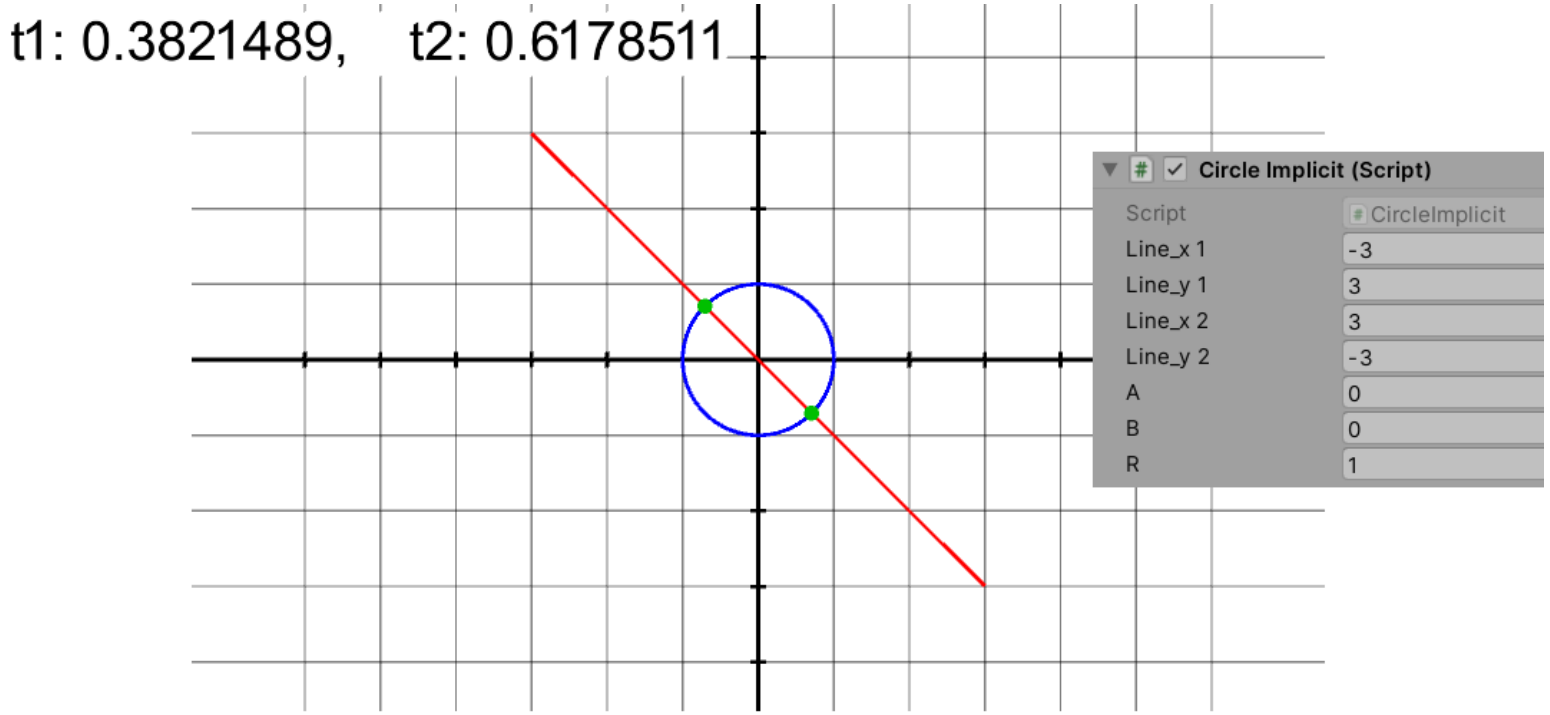
- Wolfram

```
float delta = 8.0f*b*X*(Y*(a - x0) + X*y0) - 4.0f*(Y*(a - x0) + X*y0)*(Y*(a - x0) + X*y0) - 4.0f*b*b*X*X + 4.0f*r*r*(X*X + Y*Y);
float t1 = 1.0f / (X*X + Y*Y) * (-0.5f*Mathf.Sqrt(delta) + a*X + b*Y - x0*X - y0*Y);
float t2 = 1.0f / (X*X + Y*Y) * (0.5f*Mathf.Sqrt(delta) + a*X + b*Y - x0*X - y0*Y);
```

Equations Optimization

- Let’s examine this problem once again, but with an assumption that the circle is at the origin:

$$(x_0 + \vec{v}_x \boldsymbol{t})^2 + (y_0 + \vec{v}_y \boldsymbol{t})^2 = r^2$$



Equations Optimization

- Assuming that $a = 0$ and $b = 0$ the equation simplifies substantially:

$$((x_0 + \vec{v}_x \boldsymbol{t}) - a)^2 + ((y_0 + \vec{v}_y \boldsymbol{t}) - b)^2 = r^2$$

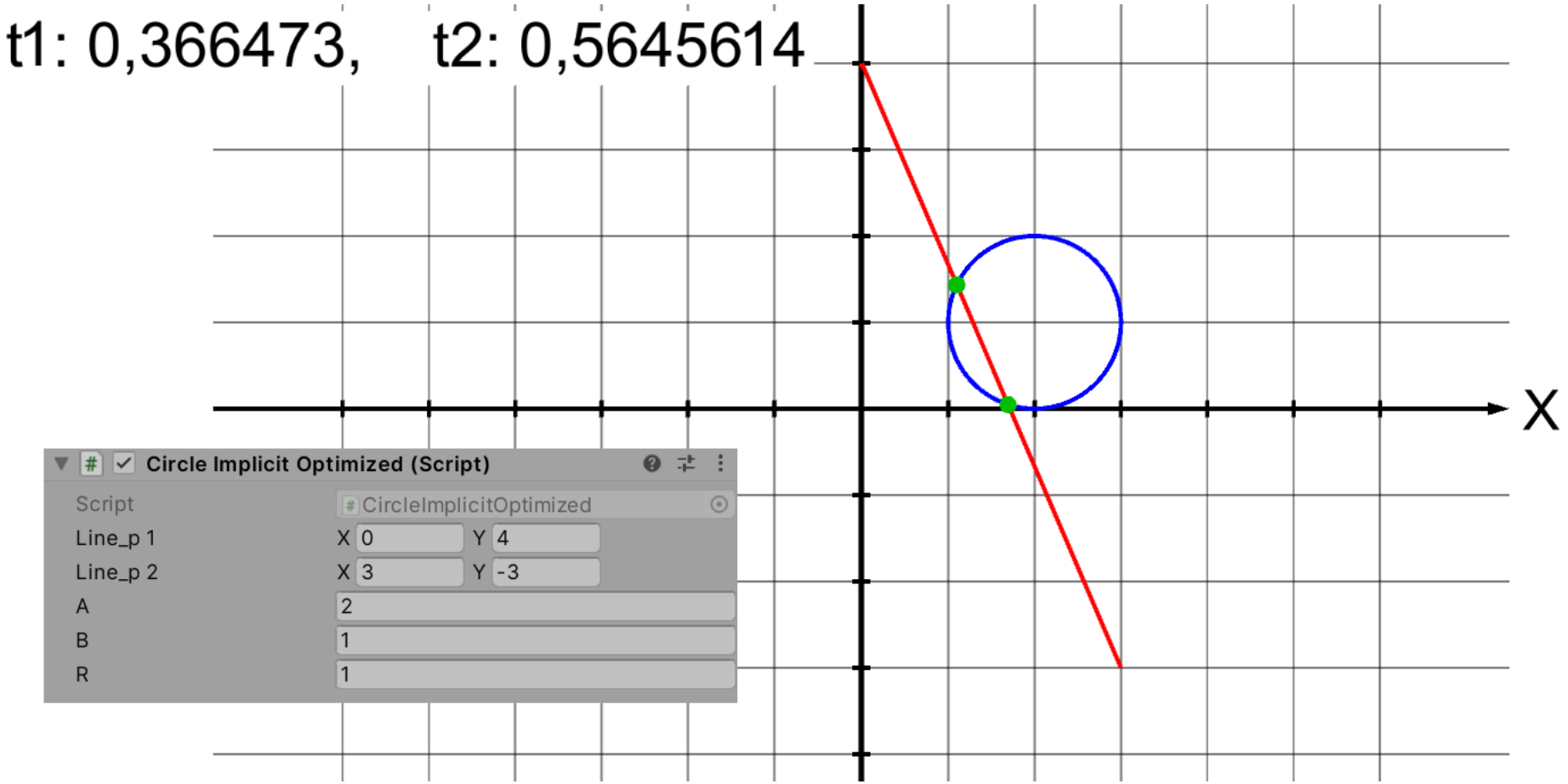
$$(x_0 + \vec{v}_x \boldsymbol{t})^2 + (y_0 + \vec{v}_y \boldsymbol{t})^2 = r^2$$

- Wolfram

```
float delta = r*r*(X*X + Y*Y) + 2.0f*x0*X*y0*Y - x0*x0*Y*Y - X*X*y0*y0;
float t1 = -(Mathf.Sqrt(delta) + x0*X + y0*Y) / (X*X + Y*Y);
float t2 = (Mathf.Sqrt(delta) - x0*X - y0*Y) / (X*X + Y*Y);
```

- We can first „offset” (just for the calculations) all objects so that the center of the circle is at the origin, and after the calculations we can „cancel”/reverse that offset

Equations Optimization



Equations Optimization

- In geometrical problems like this one it’s always worth to look for ways of „offseting” calculations so that they take place around the origin (we will revisit that in chapter 6)

Exercises

1. In program LineImplicit make sure that the intersection point is not displayed if the segments do not intersect (slide 22)
2. In program LineParametricDistanceToPoint extract the calculation of the distance to a dedicated/separate function (slide 36)
3. Write a program that finds the distance of a point from a circle (slide 38)
4. Write a program that generates a few points on a circle described with parameters a, b and r. Do not use the parametric equation (slide 44)
5. Write a program that calculates the intersection point between a line and a sphere (slide 49)
6. Write a program that finds the intersection point of a line with a triangle in 3D.
   Find the barycentric coordinates of the intersection point and use them to check if the point is inside the triangle (slide 81)
7. In ImplicitGeometry program visualize an implicit line (slide 88)