PHStudios.com Tutorials

Vectors: Math and XNA

Text Edition

Mathematics

To fully understand vectors, we first need to have a look about the mathematics and physics aspect of them. Vectors are defined as being quantities that can be broken down into components. A vector also has magnitude and direction. In mathematics, we look at vectors in an X-Y Coordinate system.

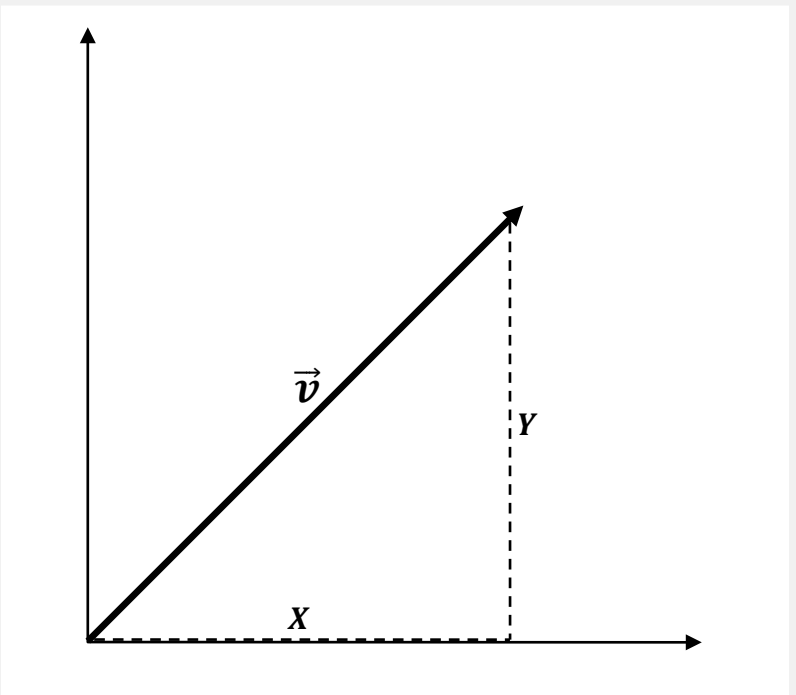

Figure 1 – Vector Diagram in X-Y System

The above figure shows the vector in an X-Y Coordinate System. The vector, designated $\vec{v}$, has two components X and Y. The X component holds the X position, and the Y component holds the Y position. Most of your games will implement this approach. Games operate a bit differently than the standard coordinate system. Let's see what a vector looks like in the game's coordinate system.

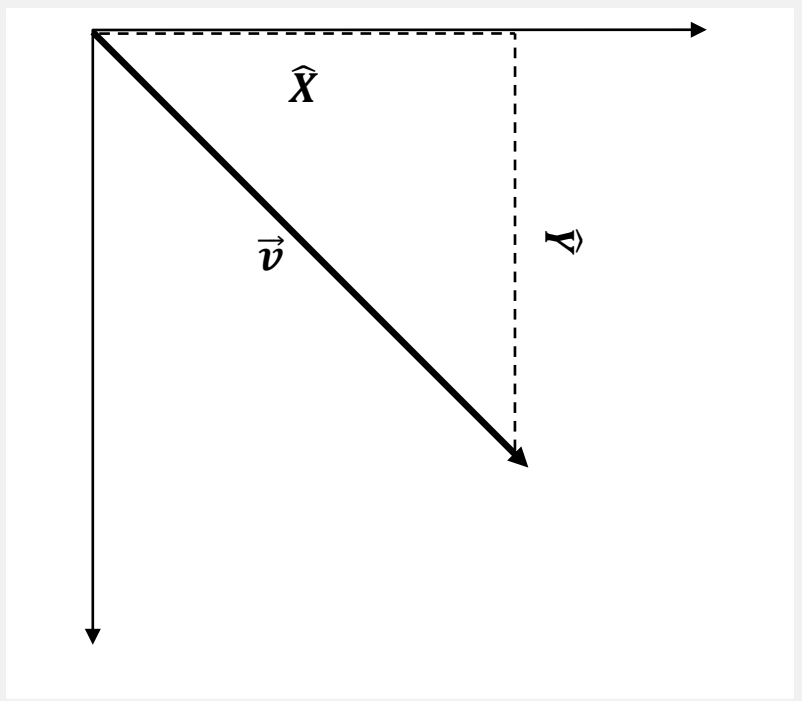

Figure 2 – Vector Diagram in XNA System

The only difference here is the positive Y is pointing down instead of up. If you have done any game programming and experimenting, you would know that +Y is down, and –Y is up in games.

Notation

When you look at a vector in a mathematics or physics book, they show the components. Notation is very important for vectors. It allows us to see what each component is in a simple way. There are several ways to look at a vector, and I will list the three most common ways. Depending on what book, website, or program you are running, you might find the vector representation different. We represent any given vector as **v** or $\vec{v}$, I will use the notation with the arrow. Vectors can be any letter or combination of letters. Later on I will use more vectors, so I will use different letters. So here are the three most common ways to represent vectors in 2D (for 3D simply add a Z to the components).

Ordered Set Notation

$\vec{v} = (x, y)$

$\vec{v} = \langle x, y \rangle$

X and Y are components of ***V***

Matrix Notation

$\vec{v} = [x \quad y]$

$\vec{v} = \begin{bmatrix} x \\ y \end{bmatrix}$

X and Y are components of ***V***

Engineering/Physics Notation

A vector in engineering or physics is represented with unit vectors $\hat{\imath}$, $\hat{\jmath}$, and $\hat{k}$ (k is for 3D)

$\hat{\imath} = (1, 0)$

$\hat{\jmath} = (0, 1)$

$\vec{v} = x\hat{\imath} + y\hat{\jmath}$

X and Y are components of ***V***

3-Dimensional Vectors

Note: Skip this section is you are only interested in 2D vectors and games for now.

2D is all fine and good, but what about 3D? 3-Dimensional vectors operate the same way, except a new component is added for depth ***Z***. There is no standard 3-dimensional diagram to follow; some books have the 3D coordinate system set one way, where other books have it set another. I will use the DirectX way to illustrate a 3D vector, instead of showing you the mathematic way first which would only complicate things. This version also makes the most sense with Z pointing away from the user (negative Z pointing toward the user). This is known as the Left-handed Cartesian Coordinate System. The Right Handed version has the Z pointing the opposite way.

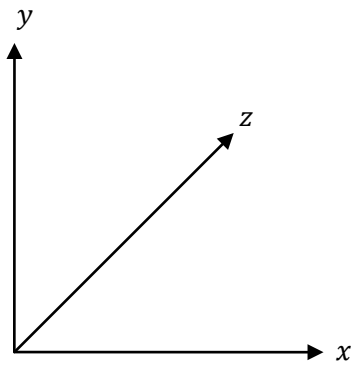

Figure 3 – DirectX 3D Coordinate System

Now let's illustrate a point on the coordinate system. We need to throw some variables out in order to illustrate this properly. Now if *P* is any given point, we let *a* be the distance along the X-axis, let *b* be the distance along the Y-axis, and we let *c* be the distance along the Z-axis.

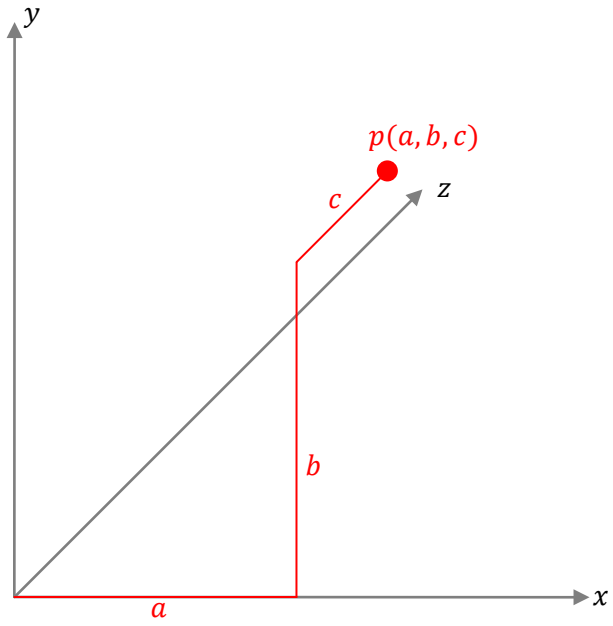

Figure 4 – General Point in 3D

It is very difficult to illustrate a 3D diagram properly on a 2D plane (monitor or paper). Let's take a look at two diagrams. One will have a z component of 0, the other will combine the two. We will keep X and Y as *a* and *b* respectively.

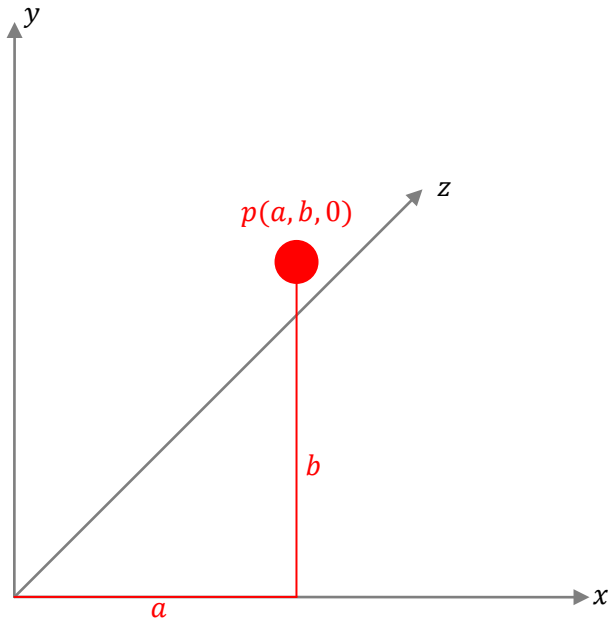

Figure 5 – Point with a Zero Z Component

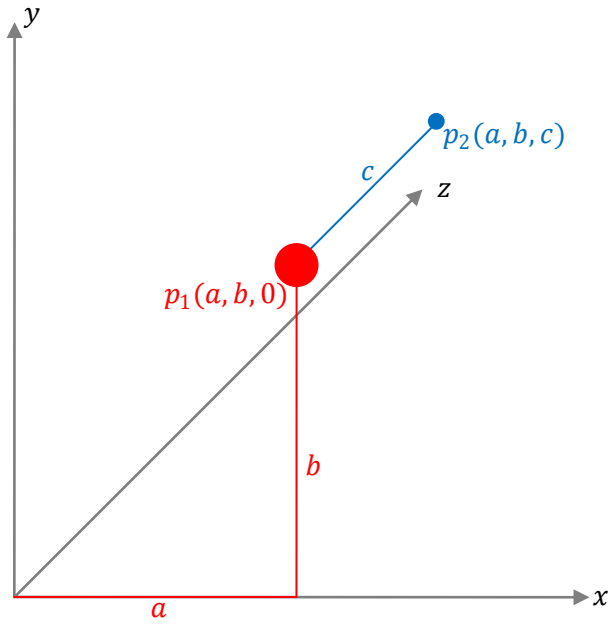

Figure 6 – Combination of both points

As the Z component of the point increases, the size of the object decreases. Again, it is very difficult to illustrate this on paper, just experiment with the 3D space as much as you can. Just remember, as Z increases, the object will get smaller. Let's take a look at a final figure showing several points in a rectangular box. This gives us an even better glace at the 3D side.

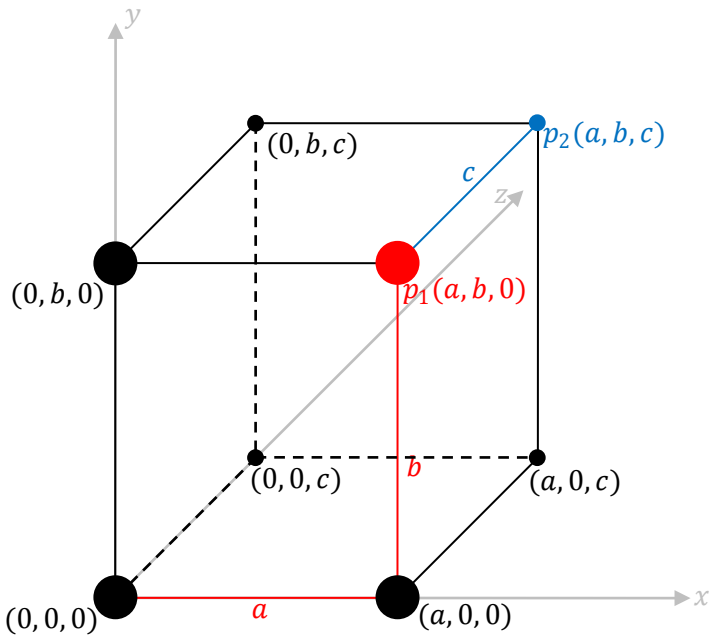

Figure 7 – Several Points in a Rectangular Box

Let's finally look at vectors in 3D instead of general points. We will look at the first point in vector form. We will write any given vector as $\vec{v}$ with components v_x, v_y, and v_z.

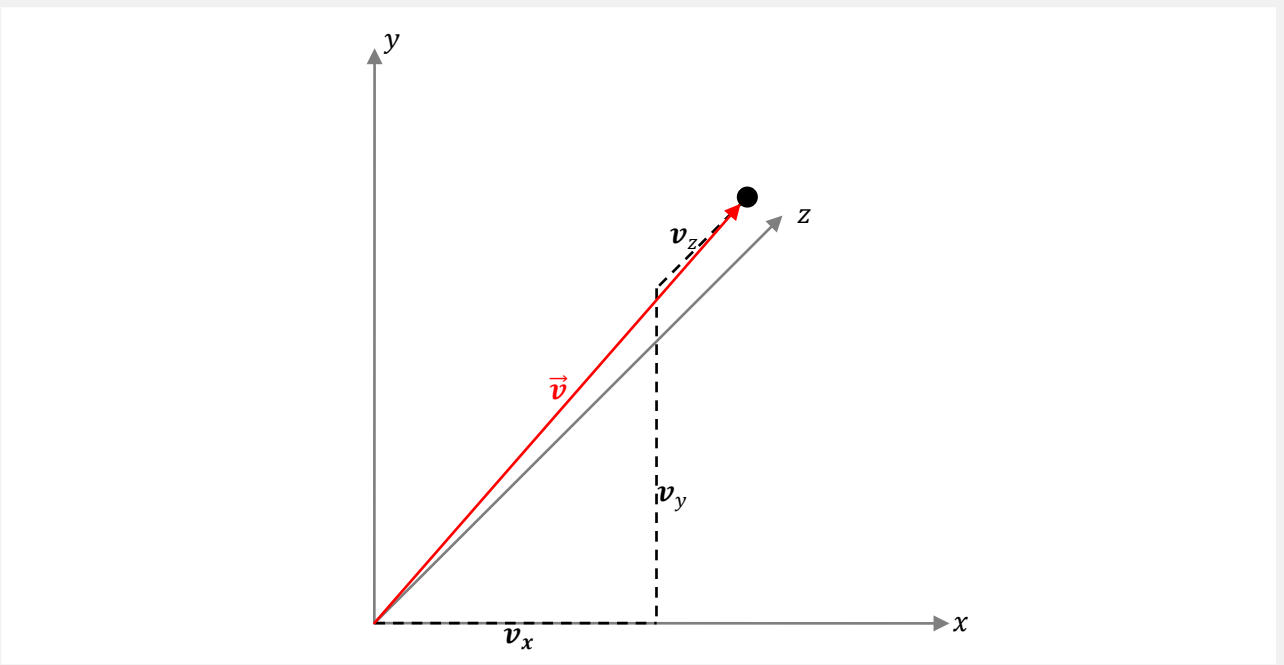

Figure 8 – General Vector in 3D

Vector Operations

Now that you saw visuals of vectors and are fairly confident that you have the basics down, let's take a look at the operations and properties of vectors. Some of this stuff will get a bit "mathy", so if you are not that into math, you can pick and choose what operations you want. This section is mostly for those that are curious as to how vectors work. If you do not care about the math, at least check out the dot product and cross product sections, as they will prove useful in more complicated games. I will put the most important things in a red box, and proofs in blue boxes. All figures will be in 2D, but math will be in both most of the time. Keep in mind, whatever vectors I use here can have any label you choose, including subscripts. I am just using generic labels. Also, a scalar in math is any number that cannot be resolved into components (3 is a scalar, so is 18.654).

Vector Addition and Scalar Multiplication

If $\vec{\boldsymbol{a}} = \langle a_x, a_y \rangle$, $\vec{\boldsymbol{b}} = \langle b_x, b_y \rangle$, and c is any given scalar

$$\vec{\boldsymbol{a}} + \vec{\boldsymbol{b}} = \langle a_x + b_x, a_y + b_y \rangle \qquad \vec{\boldsymbol{a}} - \vec{\boldsymbol{b}} = \langle a_x - b_x, a_y - b_y \rangle$$
$$c\vec{\boldsymbol{a}} = \langle ca_x, ca_y \rangle$$

If $\vec{\boldsymbol{a}} = \langle a_x, a_y, a_z \rangle$, $\vec{\boldsymbol{b}} = \langle b_x, b_y, b_z \rangle$, and c is any given scalar

$$\vec{\boldsymbol{a}} + \vec{\boldsymbol{b}} = \langle a_x + b_x, a_y + b_y, a_z + b_z \rangle$$
$$\vec{\boldsymbol{a}} - \vec{\boldsymbol{b}} = \langle a_x - b_x, a_y - b_y, a_z - b_z \rangle$$
$$c\vec{\boldsymbol{a}} = \langle ca_x, ca_y, ca_z \rangle$$

Graphically Combining Vectors

When you do any form of programming, you should test most things on paper before you code them. One of those things will be combining vectors for positioning. There are two ways to combine vectors graphically. If we have vectors $\vec{u}$ and $\vec{v}$, we move $\vec{v}$ so it's tail is connected to the tip of $\vec{u}$ and define the sum of $\vec{u}$ and $\vec{v}$ as the vector from the initial point of $\vec{u}$ to the terminating point of $\vec{v}$.

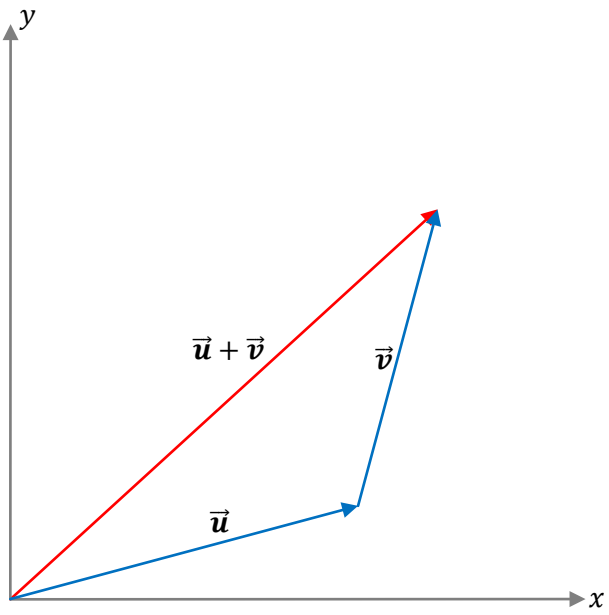

Figure 9 – Combining Vectors (Triangle Law)

The other way to combine two vectors is to have both $\vec{u}$ and $\vec{v}$ start at the same place, copy the $\vec{u}$ vector and place its tail to the end of $\vec{v}$, copy $\vec{v}$ and place its tail at the end of $\vec{u}$, and finally, the sum of $\vec{u}$ and $\vec{v}$ is the diagonal of the parallelogram.

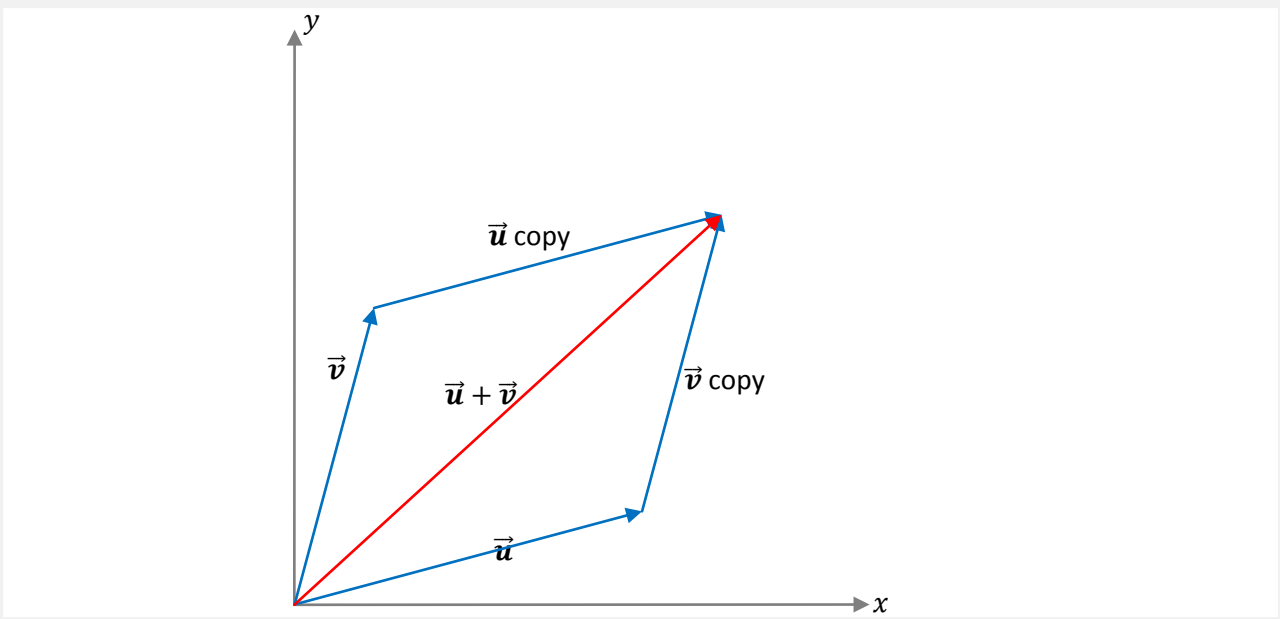

Figure 10 – Combining Vectors (Parallelogram Law)

Subtracting vectors requires one additional step. If we have two vectors $\vec{u}$ and $\vec{v}$ and subtract $\vec{v}$ from $\vec{u}$, the result would be $\vec{u} - \vec{v}$ as shown in the algebraic section. We can also write this operation a second way $\vec{u} + (-\vec{v})$, this means exactly the same thing, but allows us to graphically combine them easier. The easier way of the two is the Parallelogram Law, so that is what I will cover for subtraction.

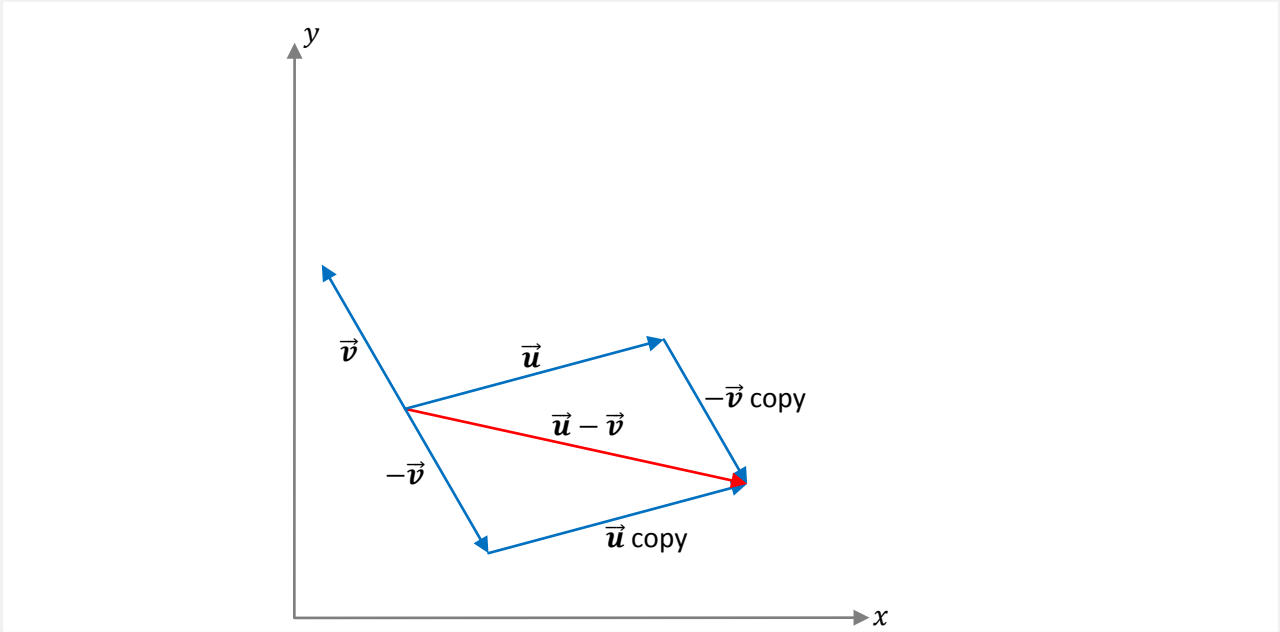

Figure 11 – Subtracting Vectors (Parallelogram Law)

Vector Between Points

Sometimes you want an object to follow the other. There are complicated ways of doing this, and easier ways. The easiest ways to do this is to grab a vector that goes from point A to point B (or two other vectors) and **normalize**, which we will get into later. **This is not built** in, so you will need to do this manually. Here is how to create a vector between two points or two other vectors.

Given the vectors $\vec{\boldsymbol{a}} = \langle a_x, a_y, a_z \rangle$ and $\vec{\boldsymbol{b}} = \langle b_x, b_y, b_z \rangle$, the vector $\vec{\boldsymbol{v}}$ represents the vector between $\vec{\boldsymbol{a}}$ and $\vec{\boldsymbol{b}}$.

$$\vec{\boldsymbol{v}} = \langle b_x - a_x, b_y - a_y, b_z - a_z \rangle$$

For two-dimensions:

$$\vec{\boldsymbol{v}} = \langle b_x - a_x, b_y - a_y \rangle$$

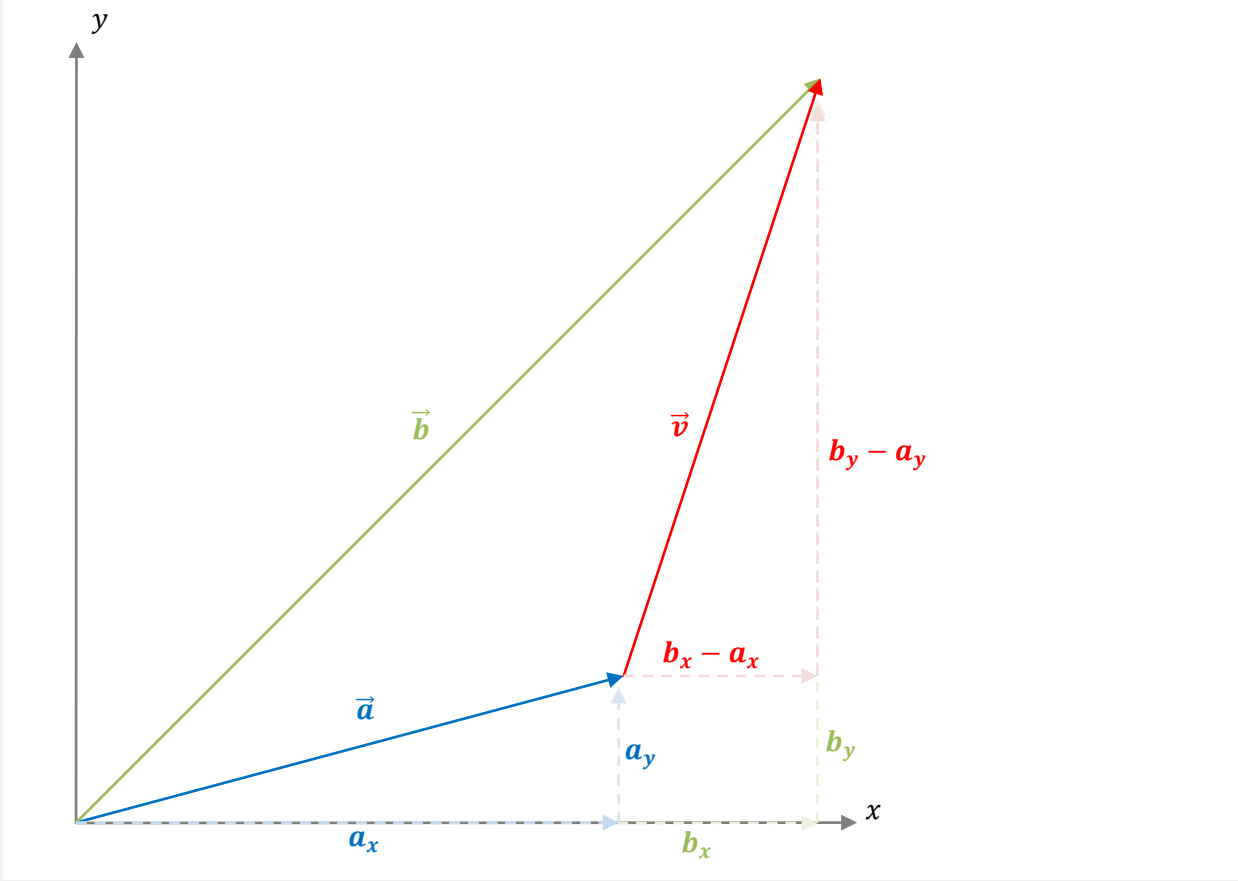

Figure 12 – Calculating a Vector Between Two Points

Vector Magnitude

A vector's magnitude, also known as the length, is the value of the longest side of the vector (in 2D, this is the diagonal). You can use this to calculate the distance an object has traveled (collision detection correction), or to determine an objects speed or acceleration. The notation for length for any given vector is $|\vec{v}|$ or $\|\vec{v}\|$.

For two-dimensions, a vector $\vec{a} = \langle a_x, a_y \rangle$ has a length of

$$|\vec{a}| = \sqrt{a_x^2 + a_y^2}$$

For three-dimensions, a vector $\vec{a} = \langle a_x, a_y, a_z \rangle$ has a length of

$$|\vec{a}| = \sqrt{a_x^2 + a_y^2 + a_z^2}$$

Math Examples

Skip this if you are not that interested in the math.

If $\vec{a} = \langle 4, 7 \rangle$ and $\vec{b} = \langle 3, 2 \rangle$, find the following:

$$|\vec{a}| = \sqrt{4^2 + 7^2} = \sqrt{\mathbf{65}} \approx \mathbf{8.062}$$

$$\vec{a} + \vec{b} = \langle 4, 7 \rangle + \langle 3, 2 \rangle = \langle 4 + 3, 7 + 2 \rangle = \langle \mathbf{7}, \mathbf{9} \rangle$$

$$3\vec{a} = 3\langle 4, 7 \rangle = \langle 3(4), 3(7) \rangle = \langle \mathbf{12}, \mathbf{21} \rangle$$

Vector Normalizing

To normalize a vector, you must turn it into a **unit vector**, which is a vector with a length of 1. This is essential to create objects that follow another so it will not jump to the destination right away. All you need to do is simply divide the entire vector by the length. Here is how you can calculate the unit vector of any given vector.

If $\vec{v}$ is any given vector, then let $\vec{u}$ be its unit vector.

$$\vec{u} = \frac{1}{|\vec{v}|}\vec{v} = \frac{\vec{v}}{|\vec{v}|}$$

$$|\vec{u}| = 1$$

Just for fun, I will prove that the length of the unit vector is indeed 1

Proof: If $\vec{v}$ is any given vector, then let $\vec{u}$ be its unit vector.

$$|\vec{u}| = \left|\frac{1}{\vec{v}}\vec{v}\right| = \left|\frac{1}{\vec{v}}\right| |\vec{v}| = \frac{1}{|\vec{v}|}|\vec{v}| = \frac{|\vec{v}|}{|\vec{v}|} = 1$$

Finding the Angle of a Vector

This is another important aspect of vectors and gaming, especially 2D. You want objects to rotate in 2D, maybe not all, but at least a few objects. When an object follows another, you do not want that object to always look up; it should be looking at the object it is following. We do this using trigonometry. Do not worry if you are not that good with math, I will explain and show you how. Any given vector has an angle attached to it, which is what makes a vector a vector by definition. **This is also not built in**. So let's see any given vector with any given angle (we use θ (theta) for any given angle)

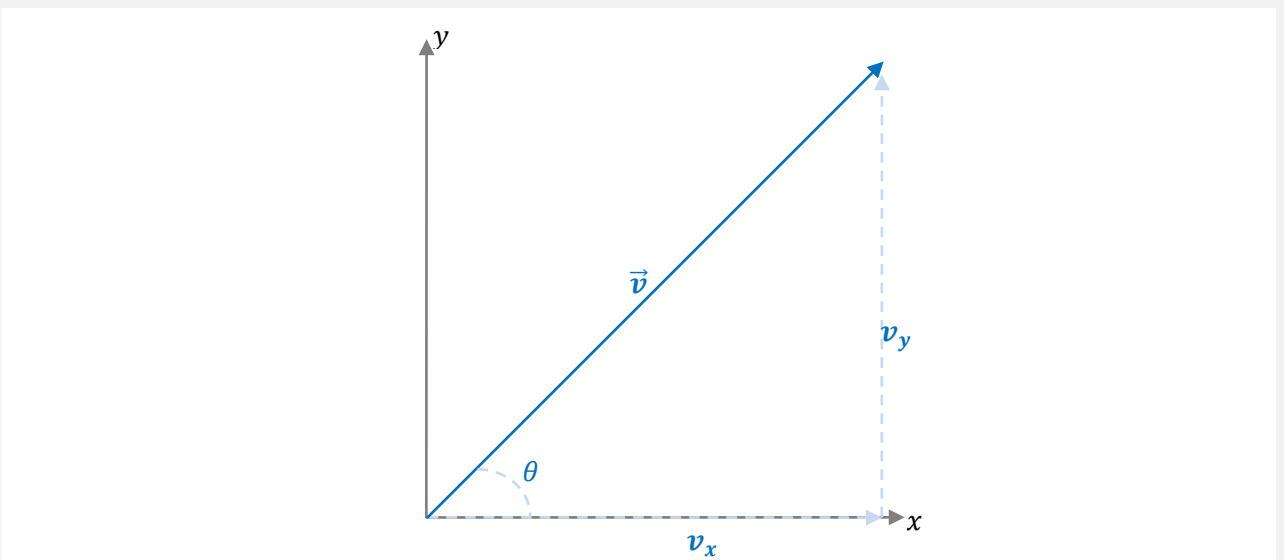

Figure 13 – The Angle of Any Given Vector

The angle is very simple to find if you know a few other parts of the vector. There are several ways to find the angle. The first way is the most common in game programming, and the one you will probably use 90% of the time.

For any given vector $\vec{\boldsymbol{v}}$, its angle v_θ can be determined one of three ways.

$$v_\theta = \tan^{-1}\left(\frac{v_y}{v_x}\right)$$
$$v_\theta = \cos^{-1}\left(\frac{v_x}{|\vec{\boldsymbol{v}}|}\right)$$
$$v_\theta = \sin^{-1}\left(\frac{v_y}{|\vec{\boldsymbol{v}}|}\right)$$

Sometimes you will have the angle and some components, but you wish to find the others. Below is a listing on how to find a component using the angle.

Trigonometry Functions

$$\tan\theta = \left(\frac{v_y}{v_x}\right)$$
$$\cos\theta = \left(\frac{v_x}{|\vec{\boldsymbol{v}}|}\right)$$
$$\sin\theta = \left(\frac{v_y}{|\vec{\boldsymbol{v}}|}\right)$$

Finding Vector Components from Trig. Functions

$$v_x \tan\theta = v_y$$

$$\frac{v_y}{\tan\theta} = v_x$$

$$|\vec{\boldsymbol{v}}| \cos\theta = v_x$$

$$\frac{v_x}{\cos\theta} = |\vec{\boldsymbol{v}}|$$

$$|\vec{\boldsymbol{v}}| \sin\theta = v_y$$

$$\frac{v_y}{\sin\theta} = |\vec{\boldsymbol{v}}|$$

The Dot Product

Now we get to the good stuff. The only way to multiply two vectors together is to use the dot product. When we multiply two vectors together using the dot product, the end result is a scalar **not** a vector. So let's see how it is done!

For any two given vectors, $\vec{v} = \langle v_x, v_y, v_z \rangle$ and $\vec{u} = \langle u_x, u_y, u_z \rangle$, then the dot product of $\vec{v}$ and $\vec{u}$ is given by $\vec{v} \bullet \vec{u}$.

$$\vec{v} \bullet \vec{u} = v_x u_x + v_y u_y + v_z u_z$$

For two-dimensional vectors, $\vec{v} = \langle v_x, v_y \rangle$ and $\vec{u} = \langle u_x, u_y \rangle$.

$$\vec{v} \bullet \vec{u} = v_x u_x + v_y u_y$$

We can also use the angle when computing the dot product of two vectors. Keep in mind the angle has to be the angle **between** the two vectors. Let's see how this works in a visual first.

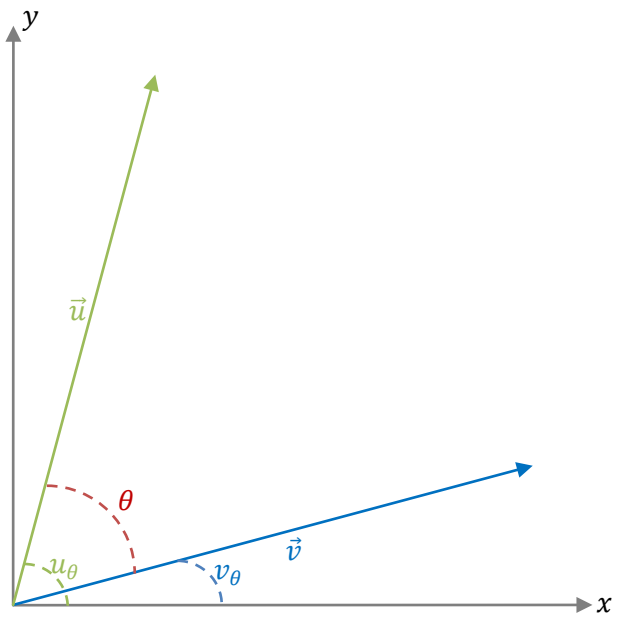

Figure 14 – The Angle between Vectors

For two any given vectors, $\vec{v}$ and $\vec{u}$, there exists an angle, θ, between them. Then

$$\vec{v} \bullet \vec{u} = |\vec{v}||\vec{u}| \cos\theta$$

Changing things around a bit we get this corollary

$$\frac{\vec{v} \bullet \vec{u}}{|\vec{v}||\vec{u}|} = \cos\theta$$

Finally,

$$\theta = \cos^{-1}\left(\frac{\vec{v} \bullet \vec{u}}{|\vec{v}||\vec{u}|}\right)$$

The Cross Product

Another method of multiplying vectors is the cross product, and it very useful for only three-dimensional vectors (sorry 2D readers). Unlike the dot product, this method is a vector. The cross product is a very long process to get to the formula from the beginning, so I will just jump straight to the formula this time.

Given the vectors $\vec{a} = \langle a_x, a_y, a_z \rangle$ and $\vec{b} = \langle b_x, b_y, b_z \rangle$, the cross product of $\vec{a}$ and $\vec{b}$ is the vector

$$\vec{a} \times \vec{b} = \langle a_y b_z - a_z b_y, a_z b_x - a_x b_z, a_x b_y - a_y b_x \rangle$$

Fun fact: $\vec{a}$ and $\vec{b}$ are parallel if and only if

$$\vec{a} \times \vec{b} = 0$$

There are a few more mathematic aspects of Vectors, but you do not need to worry about these when writing games, unless they are very complicated. If you want to learn more, check out a Calculus or higher level math books for more information.

The Fourth Dimension

XNA comes with a 4-dimensional vector (Vector4) class to use with your games. What is the fourth-dimension you might ask? Well it has two definitions. A fourth-dimension is a three-dimensional space with an extra dimension (obvious I know). This dimension is either considered an extra space dimension, or more commonly considered as time (also known as Minkowski spacetime). Einstein used this space in his theories of general relativity and special relativity. I will not go into how to implement this in your games, mostly because I have not gone into 4D spacetime yet, but it would be an interesting game to have implemented a 4D with time component properly. Search online for the fourth dimension or spacetime if this interests you. Of course you can use the Vector4 class for other things in your game like holding window dimensions.

XNA

Most of the stuff I covered above has already been implemented in the appropriate Vector classes. I do not want to list every single method and property in each vector class, but I will highlight what I have covered above. Take a look at the following cheat sheet for vectors.

Class - Vector2 Let v be an object of Vector2	
v.Length()	Returns the length of the vector
v.LengthSquared()	Returns the squared length of the vector.
v.Normalize()	Makes the length of the vector 1
Vector2.Add(Vector2, Vector2)	Returns the result of adding the two vectors
Vector2.Subtract(Vector2, Vector2)	Returns the result of subtracting the two vectors.
Vector2.Distance(Vector2, Vector2)	Returns the float distance of the two vectors.
Vector2.Dot(Vector2, Vector2)	Performs the dot product of two vectors and returns the result.

The Vector3 class has all the above, but with a few more things including the cross product.

Class – Vector3 Let v be an object of Vector3	
Vector3.Cross(Vector3, Vector3)	Returns the cross product of the two vectors.

Play around with vectors until you understand them.

End of Guide

This is the end of the Vectors guide. I hope you understand the insides of how vectors work and what cool things they can do. The math portion was mainly meant for curious readers and those wanting to build off the Vector class, or create your own. As always, play around with vectors as much as you can to fully understand them. They are quite extraordinary for gaming.