

# Particle System Mathematics © Matthew Bennet

So how do you make a particle system? Actually it's not too hard. Basically (and this applies to most of the scripts I write) you just think about how a particular object(s) moves, and see if you can break it into manageable pieces.

For a particle system, what is actually going on? Well, pretty much it can be thought of as a bunch of object's being thrown at random velocities, and in random directions.

Let's break that down a little. First, we will only worry about one object. So now we have one object being thrown in a random direction, and at a random velocity. So for simplicity, let's say it is being thrown straight up. Since this is my own little world, there are no friction or wind forces to mess up my perfect little equations.

So for a ball being tossed up, it has an initial velocity. Let's say its initial velocity is 50 m/s. Then if we ignore gravity for a minute, we can see that every second the object moves another 50 meters. (I am going to use frames and seconds as being equal for simplicity)

time distance traveled

0	0
1	50
2	100
3	150

So since in TS, the Z axis is up, then we could represent the position at time T to be:

$$Z\_position = t * 50$$

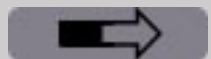
But let's get gravity back in the game. Gravity pulls the object down. So it's force is negative. That means we will be subtracting something from the Z\_position.

Acceleration due to gravity is  $9.8m/s^2$ . So what we have is the following:

$$Z\_position = t * 50 - 9.8 * t^2$$

time	position
0	0
1	40.2
2	60.8
3	61.8
4	43.5
5	5
6	-52.8

So we can see that the object indeed goes up, and then comes back down. That's great, gravity now works!



The next step is to add in a direction. The horizontal motion is actually independent of the vertical motion. I remember a question my dad used to ask me when I was a kid: "If you shoot a rifle parallel to the ground, and at the same time drop a bullet next to the rifle, which bullet will hit the ground first? The one that was shot, or the one that was dropped". The answer is that they will land at the same time, because the vertical distance traveled is the same.

So if we were to only worry about motion horizontal motion in the X direction (for now), and we are keeping our system such that there is no resistance due to atmospheric effects, then all we need for the X motion is the desired velocity. So let's say that is 2 m/s. Our position equations now look like:

$$Z\_position = t * 50 - 9.8 * t^2$$

$$X\_position = t * 2$$

Not too bad so far. Now we could easily just add in motion in the Y direction by adding in:

$$Y\_position = t * 3 \text{ (where this time I picked 3m/s as the velocity in the Y direction).}$$

So then changing the X and Y velocities, we could obtain any direction. But that is sort of hard to visualize. It works, but it's not pretty. And let's face it, math is pretty. You must at least partially agree or you wouldn't have read this far.

What I really would like to get to, is where we can specify an initial velocity, a direction, and the angle at which the projectile is propelled. The initial velocity we

already have. Remember that's our '50' in the  $Z\_position$  equation. So what about the direction. If you stick your arm straight out and spin in a circle, your hand points in all the possible directions we could through something. So let's use degrees as the direction.

To plot out a circle, you can use the following two equations:

$$x = \sin(\theta)$$

$$y = \cos(\theta)$$

Where  $\theta$  ranges from 0 to 360.

For our case,  $\theta$  will be picked and held constant. However we do want to adjust the velocity in the given direction. That's easy, just multiply both equations by the desired velocity and time. So if our horizontal velocity is 5m/s, then we have the following:

$$X\_position = t * 5 * \sin(\theta)$$

$$Y\_position = t * 5 * \cos(\theta)$$

To prove this works, if time is 1, then we should have traveled a total of 5 meters. Let's say  $\theta = 30$  degrees. Then the X motion is  $5 * \sin(30) = 2.5$ , and the Y motion is  $5 * \cos(30) = 4.33$ . Then using a little Pythagorean theorem, we have  $\sqrt{2.5 * 2.5 + 4.33 * 4.33} = \sqrt{6.25 + 18.75} = \sqrt{25} = 5$ .

Pretty cool! So now we can control the direction by simply specifying the desired angle.

So the next piece is to figure out how to get the angle of attack. Our angle will be from 0 (parallel to the

ground) to 90 (straight up). What that means is that only part of our initial velocity will be in the upwards direction, and the other part will be in the horizontal direction. Let's tackle the vertical part first.

Lets say we want to toss the projectile at 60 degrees. The majority of the velocity will be vertical, while the rest will be in the horizontal direction. To be precise, the vertical component can be determined by multiplying by the cos of the angle. Our original equation was:

$$Z\_position = t * 50 - 9.8 * t^2$$

where 50 was the velocity. We now have:

$$Z\_position = t * 50 * \cos(60) - 9.8 * t^2$$

So basically all we have done is reduce the effect of the velocity based on the angle. But we need to put the rest of that velocity into the horizontal motion. Let's call our horizontal motion 'Horizontal\_Velocity' for now.

Our Horizontal\_Velocity would then be the following:

$$Horizontal\_Velocity = t * 50 * \sin(60)$$

Where taking the sin(60) gives us the leftovers from the cos(60). Now if we consider that the Horizontal\_Velocity is really the distance travelled in a given direction, that can also be considered as the radius of a circle. Our equations for X and Y from before were:

$$X\_position = t * 5 * \sin(\theta)$$

$$Y\_position = t * 5 * \cos(\theta)$$

Where now we want to replace the  $t * 5$  (which was our arbitrarily chosen horizontal velocity), with our new

Horizontal\_Velocity equation. So we now have:

$$X\_position = t * 50 * \sin(60) * \sin(\theta)$$

$$Y\_position = t * 50 * \sin(60) * \cos(\theta)$$

Then to finally get to a nice set of general equations, let's have our Initial Velocity be  $V$ , acceleration due to gravity be  $g$ , direction be  $\theta$ , and our angle of attack be  $\nu$ . We then have the following:

$$Z\_position = t * V * \cos(\nu) - g * t^2$$

$$X\_position = t * V * \sin(\nu) * \sin(\theta)$$

$$Y\_position = t * V * \sin(\nu) * \cos(\theta)$$

One last bit, these equations are only good for objects starting at point  $(0,0,0)$ . So to get our particle system to start from any initial position, we need merely to add in the  $x, y$  and  $z$  initial positions. Let's call those  $(x_i, y_i, z_i)$ . So finally we have:

$$Z\_position = z_i + t * V * \cos(\nu) - g * t^2$$

$$X\_position = x_i + t * V * \sin(\nu) * \sin(\theta)$$

$$Y\_position = y_i + t * V * \sin(\nu) * \cos(\theta)$$

In these examples,  $t$  was taken to be seconds. That doesn't quite work if you are using this in TS. In TS, our  $t$  represents frames. So we have two options, we can divide every instance of  $t$  by 30 (or whatever frame rate the user wants), or you can just adjust the velocity and gravity so your particle system looks proper - slow them down by a factor of 30. I prefer the later being that the user can then enter more realistic values for the velocity and gravity settings. So let's add in one more little variable:

$$fps = 30$$

$$Z\_position = z_i + (t/fps) * V * \cos(\nu) - g * (t/30)^2$$

$$X\_position = x_i + (t/fps) * V * \sin(\nu) * \sin(\theta)$$

$$Y\_position = y_i + (t/fps * V) * \sin(\nu) * \cos(\theta)$$

So that's the math behind the basic particle system. The main thing is if you can visualize what's happening, then you can create it. That's really the whole trick to math. What better way to visualize, and to see the beauty in math then by running a particle system in trueSpace!!

[Close](#)