How to Write Your Own Particle Engine

Phaelax (phaelax@hotmail.com)

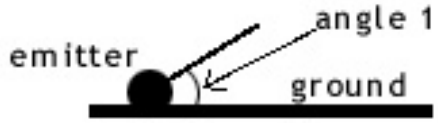
First of all, what are particles? Particles are tiny fragments of something. Slam a rock into the ground, and it may burst into smaller fragments, or create a small dust cloud from the dirt on the ground it hit. In video games, when we see those smaller rock fragments and dust clouds, we call them particles. Though, in terms of code, particles are just a bunch of objects. The rock fragments could be just an imported X model. As for the dust cloud, which really has no visible chunks in it, could be a ghosted plane with a nice smoke-like texture on it. In the example that I'm going to walk you through, the particles will be planes.

To start things off, we need to know what kind of attributes a particle has. Particles have a location. They have one angle for 2D, and two angles for 3D. I'll explain that further down. They have a velocity, as well as a time of creation, and a time of death. I'll explain that in a bit as well. There are also forces that can act upon the particle system, gravity and time. Time is related to the creation and death of the particles. There can be other forces, such as wind, but for simplicity I won't discuss them.

There will be a time index array, which will take the place for the creation and death times. Also, you need to know how many particles you'll have. For this, you make another variable, because hard-coding a value is just bad practice. Plus, it's easier if you want to change something later on. Your variable list should look something like this.

- particleTotal = 100
- pd# distance from the emitter
- px# X coordinate of particle
- py# Y coordinate of particle
- pz# Z coordinate of particle
- dim pa1#(particleTotal) angle 1
- dim pa2#(particleTotal) angle 2
- dim pt#(particleTotal) time index
- dim pv#(particleTotal) velocity
- gravity# = 2
- emitterX# X coordinate of the emitter
- emitterY# Y coordinate of the emitter
- emitterZ# Z coordinate of the emitter

Angle 1 represents the angle at which the particle shoots out of the emitter. This determines if the particle goes straight up, and shoots straight out to the side.



SIDE VIEW

Angle 2 represents the direction in space along the X and Z axis in which the particle will move.

Now how do we make the particles move? For my example, I'm going to use a projectile trajectory equation. The equation looks like this.

x = v*cos(a)*t
y = v*sin(a)*t - ½gt²

Where **v** is velocity, **a** is angle 1, **t** is time index, and **g** is gravity.

If you haven't noticed, there's no Z value anywhere. I'll show you how to add that direction by using angle 2.

To write those equations in DB using the variables we defined above, it should look something like the following below. We'll need to use a **FOR** statement to run through all the particles.

```
for i = 1 to particleTotal
  1. pt#(i) = pt#(i) + 0.5
  2. pd# = pv#(i) * cos(pa1#(i)) * pt#(i)
  3. py# = pv#(i) * sin(pa1#(i)) * pt#(i)-(0.5*gravity#)*(pt#(i)^2) + emitterY#
  4. px# = newxvalue(emitterX#, pa2#(i), pd#)
  5. pz# = newzvalue(emitterZ#, pa2#(i), pd#)
  6. check for particle death, then reinitialize variables
  7. position object i, px#, py#, pz#
  8. set object to camera orientation i
next I
```

Now here's a walk-through of what each line does.

1. Updates the time index of the particle. The higher the number with which you increment the variable by, the faster the particles with move. Think of the time index as a frame number in an animation sequence.
2. Normally, this would be the X coordinate, but because we're using 3D instead of 2D, it's going to be the distance the particle is at from the emitter. This value is only temporarily stored, and its value changed on the next pass through the loop.
3. This is the current height of the particle's arc in the air.
4. Temporarily stores the particle's X value based on the emitter's position.
5. Temporarily stores the particle's Z value based on the emitter's position.
6. This was left as pseudo-code, because this statement depends on your own preference. A particle can die by a number of ways. Its lifespan has ended, it hits something, or it hits the ground or drops below a specific point. When the particle has died, you must reinitialize the variables of that particle so that the particle can be used again by the emitter.
7. Now we position the particle.
8. This line is only needed if your particles are planes. This will ensure that we can always see the particle, regardless of the camera's rotation.

That about covers how to move the particles. You can play around with the gravity value and the amount to add onto the time index and different velocity values, until you get the look you want. I plan on doing a follow up tutorial, which discusses how to move the emitter, while having the particles still fall from its original location of birth. I will also explain how to make your particles bounce on the ground before their lifespan has ended. Below, is the code in a full working demo. Also, here is the particle image used in the demo.

filename: particle_blue.jpg

```
Rem Project: particles
Rem Created: 1/17/2003

Rem ***** Main Source File *****

REM Create a grassy looking image
for x=1 to 32
  for y=1 to 32
    dot x, y, rgb(0,rnd(255),0)
  next y
next x
get image 1,1,1,32,32

REM Create a gravel looking image
for x=1 to 32
  for y=1 to 32
    a = rnd(128)+64
    dot x, y, rgb(a,a,a)
  next y
next x
get image 2,1,1,32,32


sync on
sync rate 60
autocam off
color backdrop 0
hide mouse
```

```
REM VARIABLE DECLARATIONS
particleTotal = 300
dim pa1#(particleTotal)
dim pa2#(particleTotal)
dim pt#(particleTotal)
dim pv#(particleTotal)
pd# = 0
px# = 0
py# = 0
pz# = 0
gravity# = 2
emitterX# = 500
emitterY# = 15
emitterZ# = 500

REM IMAGES
load image "particle_blue.jpg", 3

REM CREATE PARTICLES
REM Initialize variables
for t = 1 to particleTotal
  make object plain t, 5, 5
  pv#(t) = rnd(5)+10
  pa1#(t) = rnd(120)+30
  pa2#(t) = rnd(360)
  texture object t, 3
  set object t, 1,1,0,0,0,0
  ghost object on t
next t

REM Make a cylinder to represent a fountain
make object cylinder 15000, 50
position object 15000, 500,0,500
set object 15000, 1,0,0
texture object 15000,2

REM Create a matrix and cover it with our grassy image
make matrix 1,1000,1000,20,20
prepare matrix texture 1,1,1,1
randomize matrix 1, 20
update matrix 1


REPEAT

   REM Particle loop
   for i = 1 to particleTotal
     pt#(i) = pt#(i) + 0.3
```

```
    pd# = pv#(i) * cos(pa1#(i)) * pt#(i)
    py# = pv#(i) * sin(pa1#(i)) * pt#(i)-(0.5*gravity#)*(pt#(i)^2) + emitterY#
    px# = newxvalue(emitterX#, pa2#(i), pd#)
    pz# = newzvalue(emitterZ#, pa2#(i), pd#)
    REM Particle dies if it drops below the matrix
    REM Reinitialize variables after death
    if py# < get ground height(1, px#, pz#)
      pt#(i) = 0
      pv#(i) = rnd(5)+10
      REM Shoots particle out. Not quite straight up, not quite straight out
sideways.
      pa1#(i) = rnd(120)+30
      REM Angle of direction in 3D space. Use 360 to have a full look.
      pa2#(i) = rnd(360)
    endif
    position object i, px#, py#, pz#
    set object to camera orientation i
  next I

  gosub _Camera_Control

  set cursor 0,0
  print screen fps()

fastsync
UNTIL SPACEKEY()

for t = 1 to particleTotal
  delete object t
next t
end


REM Basic First-Person camera movement
_Camera_Control:
  oldcx#=cx#
  oldcz#=cz#
  if upkey()=1
    cx#=newxvalue(cx#,a#,11)
    cz#=newzvalue(cz#,a#,11)
  endif
  if downkey()=1
    cx#=newxvalue(cx#,a#,-11)
    cz#=newzvalue(cz#,a#,-11)
  endif
  if leftkey()=1
    cx#=newxvalue(cx#,wrapvalue(a#-90.0),10.0)
    cz#=newzvalue(cz#,wrapvalue(a#-90.0),10.0)
```

```
 endif
 if rightkey()=1
  cx#=newxvalue(cx#,wrapvalue(a#+90.0),10.0)
  cz#=newzvalue(cz#,wrapvalue(a#+90.0),10.0)
 endif


 a#=wrapvalue(a#+(mousemovex()/3.0))
 cxa#=cxa#+(mousemovey()/3.0)
 if cxa#<-90.0 then cxa#=-90.0
 if cxa#>90.0 then cxa#=90.0
 position camera cx#,get ground height(1,cx#,cz#)+100,cz#
 rotate camera wrapvalue(cxa#),a#,0
RETURN
```