fxguide tip sheet

particle functions

The following functions have QuickTime reference movies associated with them, so you can see in a basic way how they work. We've also included the action particle functions, which should be placed on your sgi machine (the `/usr/discreet/user/effects/[username]/action` directory is a good place to keep them). That way, the functions can be easily loaded from within action. The easiest way is to just copy the functions directory from the CDROM into your action directory.

QuickTime examples: particles/quicktimes/
Flame/Inferno files: particles/functions.tar

Transparency functions

For transparency functions, make sure particle generator transparency is set to 0% under the Surface:Geometry menu (unless otherwise noted).

simple-fade-off
transparency = lifetimel

(evenly from start)

simple-fade-up
transparency = (1-lifetimel)

(evenly from start. Particles pop on and then fade up, however)

approx-fade-off
transparency = sin (lifetimel)

normal-fade-off
transparency =sin (lifetimel*1.5)

normal-fade-up
transparency = cos (lifetimel*1.5)

fade-in-out (over life)
transparency = sin(lifetimel*3.14)

fade-in-out+cnt
transparency = power * (sin(lifetimel * 3.14))

This is better than "transparency = lifetimel" because with the previous formula you would have particles 'pop' on and then fade. I had some popping occur when I was trying to make smoke from a chimney, but it gave this interesting puffing effect.

fade-midway+cnt
transparency = transparency / sin(lifetimel) * power

Surface geometry transparency must be set at 0. Fade down in the middle of the particles life – control with power value (between 0 and 1).

fade-up-off-fast
transparency = -0.05(pow(tan((lifetimel-0.5)/0.37, 2)) +1

Produces a very steep up flat top , very steep down curve, no explict control.

trans-near-a-point
transparency = 1 - smoothstep(100,150, length(pos-(60,40,30))))

In this case, the coordinates 60,40,30 define the location of the transparency (shown in the QuickTime with a sphere). The numbers 100 and 400 define the distances close and far - that limit the transparency. This could be used in an application such as wanting rain to light up around a street light. Unlike a matte, it works in 3D space - very nice

fade-off+cnt
transparency=pow(lifetimel,power)

fade-up-off-random
transparency = sin(lifetimel + turbulence(pos,1))

A great function that acts more naturally in fading off particles

linear-fade-up-off+cnt
transparency = (((1-lifetimel)<magnitude)*smoothstep(0,1,(1-lifetimel)/magnitude)) + (((1-lifetimel)>=magnitude)*smoothstep(0,1,lifetimel/(1-power)))

This amazing function allows you to move the fadeup (via magnitude) and the out (via power) try magnitude = .1 and power = .6

hold-then-off
transparency = (((1-lifetimel)<magnitude)*smoothstep(0,1,(1-lifetimel)/magnitude)) + (lifetimel<=0.5)*0.5

*(lifetimel>0.5) returns 1 when lifetimel is greater than 0.5 and zero when lifetimel is less than 0.5
and likewise (lifetimel<=0.5) returns 1 when lifetimel is less than or equal to 0.5 and zero when lifetimel is greater than 0.5 thus multiplying the "then" and "else" functions by their respective conditional expressions and adding them together results in transparency fading down to 0.5 until halfway through the particles life and then remaining at 0.5 till the end....*

fade-up-to-hold
transparency = (((1-lifetimel)<magnitude)*smoothstep(0,magnitude,1-lifetimel)*power)+((1-lifetimel)>=magnitude)*power

This function has the particles fade up to your desired transparency (power) at a designated time (magnitude) and hold for the duration of the particles life.

Color functions

yellow-green-blue
rgb = (red*lifetimel, sin(lifetimel), 25)

Colors shift through the range during their life

fire-yellow-red-black
rgb=(red*lifetimel, green*pow(lifetimel,4), 0)

In Surface/Geometry, keep the diffuse/specular/ambient colours as default (white/white/black). The first frame of the particles will be white, but then jumps to yellow because of the above expression. The green channel drops faster than the red channel, producing orange before red and then black. (tip: great for fire)

red-purple
rgb = ((1,0,0)-(0,0,1))*smoothstep(0,1,lifetimel))+(0,0,1)

This will make the color go from red to purple

color-pos
rgb=((magnitude-smoothstep(minSpeed,maxSpeed,length(pos-(50,0,0)))),(power-smoothstep(minSpeed,maxSpeed,length(pos-(50,0,0)))),(damping-smoothstep(minSpeed,maxSpeed,length(pos-(50,0,0)))))

This will allow you to make the color of a particle reach a certain value around a certain point. The target color is defined using the magnitude (red), power (blue), and damping (green) values. For magnitude and power, values are between 0 and 1. For damping, the values are between 0 and 100. This allows for easy changing and even animation of color. If you don't want to deal with the different scale methods, simply replace the magnitude, power, and damping words in the function with r,g,b values. The position of the color change is set in the function – in the example shown it is (50, 0, 0). Finally, the MinSpeed and MaxSpeed manage the area of influence of the color change. The Surface:Geometry setting of the particle generator determines the initial color.

color-fade
rgb = (1*lifetimel, .8*lifetimel, .2*lifetimel)

This will make the color go from the initial color to black over the lifetime of the particle. Enter the initial color both in the function and in the Surface:Geometry menu so that the color doesn't pop on. For instance, in the formula above, you would also enter values of 100, 80, and 20 in the Geometry menu.

Turbulence functions

speed-noise+cnt1
speed = speed + cross ((power, power, power), noise3(pos))

Changing the power and speed affect the turbulence...try power=3.0

speed-noise+cnt2
speed = speed + cross ((power, power, power), noise3(pos)) * magnitude

*Changing the power, speed and magnitude all affect the turbulence... * note:In the above two examples, you can achieve some very interesting effects by keyframing the power, speed or magnitude channels. For instance, start out with very little turbulence and then ramp up into something really chaotic!*

speed-turbulence+cntl1
speed = speed + cross((power,power,power), turbulence3(pos,1)

Same as above but using turbulence rather than the more 'random' noise function. note it is turbulence3 as we need the vector version.

turb1
speed=(speed+cross(power*((1-lifetimel)/2), power *((1-lifetimel)/2), power*((1-lifetimel)/2)), turbulance3(pos,1)))

Speed is defined as speed = speed + another speed vector ,.. plus it scales down the final vector by lifetime.

turb2
speed=(speed+cross((power*(1-(lifetimel/2)), power*(1-(lifetimel/2)), power*(1-(lifetimel/2))), turbulence3(pos,1)))/(magnitude,magnitude,magnitude)

As the speed is defined as speed = speed + another speed vector ,.. we also need to scale down the final vector. So you need to divide everything by say (magnitude,magnitude,magnitude)

small-turbulence
speed=speed*0.95+turbulence3(pos*0.01,1)

When rendering out thisone with let's say a full light with 180 spread on it as the generator what you will see are the obvious flows and eddies that the particles move through (use lines to see this, only one manipulator for the function).

small-turbulence2
speed=speed*0.95+turbulence3((pos+(0,0,frame*20))*0.01,1)

In this formula you begin to see the flows and eddies and then they are suddenly disrupted. In the first formula the variables are 0.95 and 0.01. In the second the 20 is also a variable. A lower number should make the swirls evolve at a different rate. The 0.01 value in both lines effects the apparent "lattice" scale. A smaller number makes for a bigger lattice, creating bigger/looser swirls. The range for this will be 0.1-0.001.

small-turbulence3
speed = speed + (sin(lifetimel/2 + 3*turbulence(pos,1)), sin(lifetimel/2 + 3*turbulence(pos,1)), sin(lifetimel/2 + 3*turbulence(pos,1)))