Particle Systems

I'm sure you love playing computer games, especially those ones which involve lots of fighting, blood, blasts, rockets, etc. You sure enjoy blasting a rocket which leaves a smoky trail behind it and you also love to watch a fire effect showing the burning of the fuel. When this rocket hits a monster, its pieces fly away realistically as if the rocket pieces are obeying physics laws.

Indeed, they are obeying physics laws. Welcome to the world of particle systems. All the effects such as fire, water, smoke, blood, etc. are at a basic level particle systems. Now what the heck is a particle system? It can be a collection of individual particles of smoke (or for that matter any other effect) behaving according to a set of rules. When the (physics) rules mix and match with the particle's properties such as velocity, weight, acceleration, etc., it creates wonders. You get a totally realistic effect without much effort. All you have to do is to define characteristic properties for the particles of the desired effect and apply the basic rudimentary laws such as gravity, momentum transfer, etc. to it. Yes, of course you can add more laws such as viscous drag, etc. to your particle system. More the laws you integrate better the realism is.

You may say: "WOW! That is wonderful. I wish I could get my hands dirty on an actual particle system". Behold, your wish is granted. We are actually going to build *our own* basic particle system in C++. It can be implemented in any compiler of C++ supporting graphics. But, here we are going to use Turbo C++ 3.1. But, note that I have also implemented this system using Allegro library. But, you are free to use any other graphics library such as DirectX or OpenGL.

Firstly, note the download location for the source code. It is
http://www.paraschopra.com/sourcecode/particle.zip

Due to space restrictions, we will just be discussing code snippets here and not the whole code itself. But whatever we discuss will be sufficient for you to follow the whole code. First let us define data structures required.

```
enum ParticleType { Create,Update };

struct Vector2d{
        double x;
        double y;
};

struct Particle {
        Vector2d Pos; //Position of the particle
        Vector2d Vel; //Velocity of the particle
        int age; //Current age of the particle
        int LifeSpan; //Age after which the particle dies
        int color;
        int size;
};
```

Here we have a structure defining the properties of a particle viz. Position, velocity, age, color and size. As we are running our particle system in a 2D world, we will just need two coordinates; X & Y, which are given by the structure *Vector2d*. Note that here the origin is top-left corner of the monitor and X is positive to the east (or right) of origin and Y is positive to the south (or downwards) of the origin.

The *Create* constant defines whether an existing particle has died and should it be replaced by creating a new particle elsewhere.

Now what a particle system consists is a huge lot collection of particles. Each individual particle defined by structure Particle. So when the various forces are applied to them, they behave in different manner. This behavior is different because different particles have different properties such as locations or positions, so when a force acts on them, the same force produces different changes.

Now, let us take a look at the Initialization function *Init()* of the Particle system class.

```
void Init(long num_part,long num_forces, Vector2d Forces[], ParticleType part_type,Vector2d vel,
Vector2d pos1,Vector2d pos2, int lifespan, int color, int size){
                    Particles = new Particle[num_part];
                    ParticleNum=num_part;
                    for(int i=0;i<num_forces;i++){
                              TotForce.x+=Forces[i].x;
                              TotForce.y+=Forces[i].y;
                    }
                    Particle_Type=part_type;
                    Vel=vel;
                    Pos1=pos1;
                    Pos2=pos2;
                    LifeSpan=lifespan;
                    Color=color;
                    Size=size;
                    randomize();
                    InitParticles();
          }
```

Pos1 and *Pos2* (in the arguments) indicate the position of the two opposite corners of the rectangle between which particles are generated. This rectangle basically restricts the initial position of a newly generated particle within it. However, the position of individual particles is random provided it lies within the rectangle. This can be seen in the function InitParticles() which initializes all the particles.

The array *Forces* indicate the various forces acting on the system. These forces are universal, that is they do not have position values but have magnitude values only. They act in the same way on all of the particles, irrespective of their respective positions. These forces are then added together to represent a single force *TotForce*.

The real core of a particle system is the main loop of execution; it is where the updation of the particles' properties occurs. It means the particles' properties such as location, age,

velocity, etc. are updated according to the forces and some other criteria like age, etc. Let us analyze our own loop function *run()*.

```
void Run(){
        while(!kbhit()){
                delay(10);
                cleardevice();
                //delay(10);
                for(int i=0;i<ParticleNum;i++){
                        if(Particles[i].age >= 0){
                        setfillstyle(1,Particles[i].color);
                        setcolor(Particles[i].color);
                        //circle((int)Particles[i].Pos.x,(int)Particles[i].Pos.y,Particles[i].size);
        fillellipse((int)Particles[i].Pos.x,(int)Particles[i].Pos.y,Particles[i].size,Particles[i].size);
                        Particles[i].Vel.x+=TotForce.x;
                        Particles[i].Vel.y+=TotForce.y;
                        Particles[i].Pos.x+=Particles[i].Vel.x;
                        Particles[i].Pos.y+=Particles[i].Vel.y;
                        Particles[i].age++;
                        if(Particles[i].age > Particles[i].LifeSpan){
                        if(Particle_Type == Create) {
                        InitParticle(i);
                         } else {
                         Particles[i].age = -1;
                         }   }

                        }
                }
        }   }
```

The function *kbhit()* ensures that the particle system is run until any of the keys is pressed. You may observe here that the value of *TotForce* is added to the value of the velocity, hence increasing it and the value of the particle's velocity is added to its position. This increase may actually be decrease if the velocity or *TotForce* is negative.

For each particle, after the position and velocity are updated, an ellipse (or circle) is plotted on the screen at the location specified by its position variable.

Also note that the particle's age is increased by 1 each time it is updated in the loop and when its age increases its maximum age allowed, a new particle is initialized. Note that the maximum age is different for different particles to achieve more realism.

We have used the base class *ParticleSystem* to derive a class called *Bouncy*. It just overrides the *run()* function so as to integrate a virtual walls enclosing the screen from which particles bounce and lose some of their velocity (10% in this case). It is implemented as follows:

```
if(Particles[i].Pos.y>getmaxy() || Particles[i].Pos.y<0) Particles[i].Vel.y*=-0.9;
if(Particles[i].Pos.x>getmaxx() || Particles[i].Pos.x<0) Particles[i].Vel.x*=-0.9;
```

You may try experimenting with other values instead of just -0.9. Try using -0.3 or different values for x and y. Trust me this thing is fun.

Now for the last part, we create an actual particle system by creating an object for class *Bouncy* (or *ParticleSystem*)

```
void main(){
        Bouncy n;
        //ParticleSystem n;
        int n_forces=1;
        Vector2d f[1];
        f[0].x=0.0;
        f[0].y=0.5;
        Vector2d vel,pos1,pos2;
        vel.x=5;
        vel.y=5;

        n.InitGraphics();
        setbkcolor(BLACK);
        pos1.x=getmaxx()/2;
        pos1.y=getmaxy()/2;
        pos2.x=getmaxx()/2;
        pos2.y=getmaxy()/2;
        n.Init(1000,n_forces,f,Create,vel,pos1,pos2,100,EGA_GREEN+EGA_RED,1);
        n.Run();

        getch();
}
```

Here we are creating a particle system with 1000 particles of orange color and size 1. We just have a single vertical force of value 0.5. This is to emulate a kind of gravity. We initialize the positions of all the particles at the center of the screen. We thus have our particle system ready to roll.

That's it. Enjoy the show. You have all the freedom to play with your newly created cute little particle system. You may try adding an X component of velocity also, or you may place the initial source of particles at the bottom of the screen and then giving the particles negative Y velocity, thus creating a fire effect.

You may also add some more features into the existing system by deriving classes from it. You may have multiple particle systems operating simultaneously or a system which reduces particles size or color with time. The diverse effects offered by this system are only limited by your creativity. Unleash it and rule the world.

At the end I would like you to tingle with your grey matter by trying to simulate various effects such as viscous drag, fire, smoke, a blast and flow of blood or any other effect you see in a Hollywood flick or the latest games.

Web location of this tutorial: http://www.paraschopra.com/tutorials/particle-systems/