Assignment 3 - *Worth 25% of overall grade.*

This assignment is intended to show your understanding of the “Particle Systems” material presented in Weeks 6, and 7, and to give you a chance to flex your graphics muscles with something that is both challenging and fun to write. You will build upon the material covered in Weeks 6 and 7 to implement your own particle system which will be flexible and data driven.

You are required to:

1) Write a class, *Effect*, which is a container for your particle *Emitters*. This class will contain a list of emitters which are the real body of work in this module. The Effect will delegate work down to this collection, for example to Update all the emitters during a call to Update on the effect. The Effect class must be able to be given a parent transform (position, scale and rotation) that will be applied to all emitters it contains. Usage of the Effect class would look something like:

```
Effect* pExplosion = new Effect(“explosion.pfx”);
pExplosion->SetPos(vCarExplodedPos); // builds transform that is passed to
                                     // all emitters as their parent
                                     // transform.
pExplosion->Update(fDelta); // updates all emitters
pExplosion->Render();       // renders all emitters
```

2) As seen in the code example above, the Effect class must load its description from an XML file (in this case, with the extension .pfx but it’s still just XML inside). This XML file should be a simple list of the various emitters that make up this overall effect, plus any extra data you might want. Each emitter defined in the file should be able to be given a positional offset from the effect’s position that is applied to it. The XML may look something like:

```
<effect name="explosion">
   <emitter file="sparks.emitter" offset="0,0,1"/>
   <emitter file="debris.emitter" offset="0,0,0"/>
   <emitter file="fire.emitter" offset="0,0,0"/>
   <emitter file="smoke.emitter" offset="0,4,0"/>
</effect>
```

3) Write a class, *Emitter*, which is the core of this assignment. This emitter is responsible for the core areas covered in the lectures: particle management, spawning particles and giving them initial properties, updating particles, and finally rendering them. These will be covered in the following points (4-7). The Emitter must be able to load itself from an XML data definition. This XML *may* look like the following, although you are free to come up with whatever representation suits you, so long as it is valid XML.

```
<emitter name="smoke" num_particles="100"
       duration="2.0" type="continuous"
       birthrate="20">

  <spawn_property name="velocity" type="random" min="0,0,0" max="10,0,0"/>
  <spawn_property name="color" type="constant" value="1,1,0,1"/>
  <spawn_property name="size" type="random" min="16" max="24"/>

  <affector type="scale">
```

```
        <property name="mode" value="over_life"/>
        <property name="start" value="1"/>
        <property name="end" value="2"/>
    </affector>

    <affector type="add_velocity"/>
</emitter>
```

4) Your Emitter class should manage its particles from a constant pool. The Emitter should create a pool of X particles at initialization time, where X is defined in the XML file (via an attribute such as *num_particles*). No more particles should be allocated beyond this initial pool during the lifetime of this emitter. Your particle management should follow the linked list system presented in the lectures in order to recycle particles when needed.

5) Your Emitter class must support continuous and burst mode for emission types. Burst mode does not need to have the ability to repeat bursts, although it is encouraged. When a particle is spawned, it must be given initial values for the following properties: *color*, *size*, *velocity*, *lifetime*, and *fade*. Additional properties are encouraged but not required. Each property must be able to be given a value either through a *constant setter* or a *random range setter*, as described in the lectures. Which one it uses, and its parameters, should be defined in the XML description.

6) Your Emitter class must also contain a list of Affectors that it will apply to all particles that are currently spawned. These Affectors should follow the simple class hierarchy presented in the lectures to make implementation easy - in other words, don't focus on performance here, focus on easy to read and flexible code. Which Affectors are used in the emitter, and their properties, should be defined in the XML description as seen in the example above. When updating affectors, your Emitter class must also check if the particle's lifetime has expired and kill it if so.

   You must support at least the following Affectors: *AddVelocity, Scale* (grow and shrink), *Fade* (in and out) - although you may want to add others such as *ApplyGravity* or *ColorBlend*. The *Fade* Affector must allow a particle to be faded out over the entire lifetime of the particle, and the Scale affector must allow you to choose a *start* and *end* value for the particle's scale to interpolate over during its whole lifetime.

7) The Emitter class must render all active particles in one draw call. This should be done by filling the Vertex Buffer every frame based on the current values for the particles that are active. All particles need to be billboarded. All particles in a single emitter should be rendered using the same wolf::Material (although you will, of course, want different materials for each emitter).

8) Your Emitter class should have a *duration*, after which it will stop spawning particles (if it's a *continuous* emitter). The value of -1 for the duration should be interpreted as infinite.

9) A sample program must be provided showing off a particle effect of your own design, and a *fire* effect, with a key to toggle between the two. The fire effect should have two emitters: *smoke* and *fire*. The fire should use additive blending to give the impression of fire, and the smoke should start above the fire and drift upwards while scaling up and fading out. It should be possible for me to create a new effect in your system by writing only XML files.

You need to submit (to Moodle) the assignment as a zip file containing all the source code and assets required to build and run it. A readme file should also be provided inside the zip file with any additional items you want taken into consideration (if any). The program **must** be *submitted in a working format, or marks will be lost (i.e. it must be able to be built and run from visual studio "out of the box").*