Open Scene Graph Particle Effects

In this exercise you will learn how to create a particle emitter. We are going to build on the file used for the collision exercise

- Create a new Win32 console project called *OSG particle effects*
- Set the properties as for the lighting tutorial and ALSO add the library osgParticled.lib

Create geometry and display in the viewer

- Copy the entire program code you used for the collision exercise. Test the project
- Now add in the new header files (you will also need to add the particle library to your project)

```
#include <osg/PositionAttitudeTransform>
#include <osg/ref_ptr>

#include <osgParticle/Particle>
#include <osgParticle/ParticleSystem>
#include <osgParticle/ParticleSystemUpdater>
#include <osgParticle/ModularEmitter>

void moveBox...
```

- Next you need to create a particle system and add it to a geode node

```
redXform->addChild(redHex.get());
	myRoot->addChild(redXform.get()); // adds it all to the root node

	//*************************************************************//

	//******************** Create and initialize a particle system************//

      osgParticle::ParticleSystem *myParticleSystem = new osgParticle::ParticleSystem;
// Set the attributes 'texture', 'emmisive' and 'lighting'

	myParticleSystem->setDefaultAttributes("models/fire.png", false, false); //
This is a simple particle system  with default values. This holds the particles, and
will be added to a geode

	osg::Geode *particleGeode = new osg::Geode; // particle is from the drawable
class, so needs to be added to a geode

	myRoot->addChild(particleGeode);
	particleGeode->addDrawable(myParticleSystem);

	//*************************************************************************//

		//*******set up the viewer to display your scene***************//

	osgViewer::Viewer viewer; // Declare a 'viewer' which will display the scene
```

- Now you need to add an updater, so that the particle system is updated every traversal:

```
myRoot->addChild(particleGeode);
particleGeode->addDrawable(myParticleSystem);


osgParticle::ParticleSystemUpdater *psu = new osgParticle::ParticleSystemUpdater; //
now we need an updated added to the scene, to update the particle system every
traversal
    psu->addParticleSystem(myParticleSystem);
       myRoot->addChild(psu);

       osgParticle::Particle myParticle;  // create a particle to be used by our
system
    myParticle.setSizeRange(osgParticle::rangef(0.005,0.01)); // meters
    myParticle.setLifeTime(4); // seconds
    myParticle.setMass(0.01); // in kilograms
    myParticleSystem->setDefaultParticleTemplate(myParticle);   // Make this our
particle system's default particle


//*************************************************************************//

             //*******set up the viewer to display your scene***************//

       osgViewer::Viewer viewer; // Declare a 'viewer' which will display the scene
```

- Next create an emitter and a shooter for creation and control of the particles. Remember to test the project after each stage. You won't see anything different happening yet.

```
    myParticle.setMass(0.01); // in kilograms
    myParticleSystem->setDefaultParticleTemplate(myParticle);   // Make this our
particle system's default particle

osgParticle::ModularEmitter *emitter = new osgParticle::ModularEmitter; // a modular
emitter has a counter, a placer and a shooter, to set number, position and velocity of
particles
       emitter->setParticleSystem(myParticleSystem);
       osgParticle::RandomRateCounter *rateCounter =
static_cast<osgParticle::RandomRateCounter *>(emitter->getCounter()); // we are going
to use the existing particle counter
       rateCounter->setRateRange(5,10);    // generate 5 to 10 particles per second

    osgParticle::RadialShooter* myShooter = new osgParticle::RadialShooter(); // To
customize the shooter, create and initialize a radial shooter

    // Set properties of this shooter
    myShooter->setThetaRange(0.0, 3.14159/2); // radians, relative to Z axis.
    myShooter->setInitialSpeedRange(0.5,1); // meters/second
    emitter->setShooter(myShooter);// Use this shooter for our emitter

//*************************************************************************//

             //*******set up the viewer to display your scene***************//

       osgViewer::Viewer viewer; // Declare a 'viewer' which will display the scene
```

- Now we can attach the emitter to our scene. We are adding it to the same transform node which holds our blue hex box, so that they move together. When you test this project now, you should see your emitter working. Experiment with changing some of the parameters.

```
    myShooter->setInitialSpeedRange(0.5,1); // meters/second
    emitter->setShooter(myShooter);// Use this shooter for our emitter

blueXform->addChild(emitter); // we need to add the emitter into the system, then put
the partical system where we need it using the blue transform node

//*************************************************************************//

             //*******set up the viewer to display your scene***************//

       osgViewer::Viewer viewer; // Declare a 'viewer' which will display the scene
```

- To make this a bit more interesting, let's link the emitter behaviour to the hit test. Firstly disable the emitter in the main program:

```
blueXform->addChild(emitter); // we need to add the emitter into the system, then put
the particle system where we need it using the blue transform node

       emitter->setEnabled(false);


       //*************************************************************************//
```

- Now we will alter the headers for the moveBox function so that we can pass the emitter as an argument into the function:

```
#include <osgParticle/ModularEmitter>

void moveBox (osg::ref_ptr<osg::PositionAttitudeTransform> blueBox,
osg::ref_ptr<osg::PositionAttitudeTransform> redBox, osgParticle::ModularEmitter
*myEmitter); // this function is  going to move one of the boxes towards the other
until we get a hit


int _tmain(int argc, _TCHAR* argv[])
{



return 0;
}
```

- and:

```
void moveBox (osg::ref_ptr<osg::PositionAttitudeTransform> blueBox,
osg::ref_ptr<osg::PositionAttitudeTransform> redBox, osgParticle::ModularEmitter
*myEmitter)// we've altered this function to enable the emitter on collision
{
	osg::Vec3d boxPos = blueBox->getPosition();
	boxPos[0] +=0.002;
	blueBox->setPosition(boxPos);
```

- and we also need to change the function call in the main loop

```
    while (!viewer.done()) // until the end of the program
	{
		moveBox(blueXform, redXform, emitter);
		viewer.frame(); // update to next frame
```

- Finally, let's control the emitter activity via the collision (note that we have removed the red box shrinking behaviour)

```
if(bs1.intersects(bs2))
	{
		boxPos[0] -= 0.1;
		blueBox->setPosition(boxPos); // bounce the box left after a collision
		myEmitter->setEnabled(!(myEmitter->isEnabled())); // toggles the emitter
on and off on hit
	}
```

Now that you can make and control a simple emitter, see if you can make one a little more interesting (for example, a fountain which can be turned on or off; exhaust fumes from a moving car.....)