MTAT.03.015 Computer Graphics (Fall 2013)
Exercise session XIV: OGRE

Konstantin Tretyakov, Ilya Kuzovkin

December 9, 2013

In this exercise session we will have a look at high-level graphics engine called OGRE[1]. We will see how the concepts we know about are included into the OGRE engine, making our life easier: lighting, materials, shadows, environmental mapping and other techniques are made accessible by adding few lines of code, without the need to implement all annoying details on our own.

The solutions will have to be submitted as a zipped project directory. Please keep Windows libraries even if you work on Linux or Mac.

You can always seek for additional information and help in official tutorials `http://www.ogre3d.org/tikiwiki/tiki-index.php?page=Tutorials`.

1 Structure of the application

We start by comparing the structure of the application to the familiar structure we've been using so far. Please open `1_OgreTriangle` project and read through the code in the `triangle.cpp` file. Compare it to the GLUT-based applications we have seen before.

Exercise 1 (0.5pt). Add a small square which will fly around the triangle and rotate around it's own center. For that you will need to

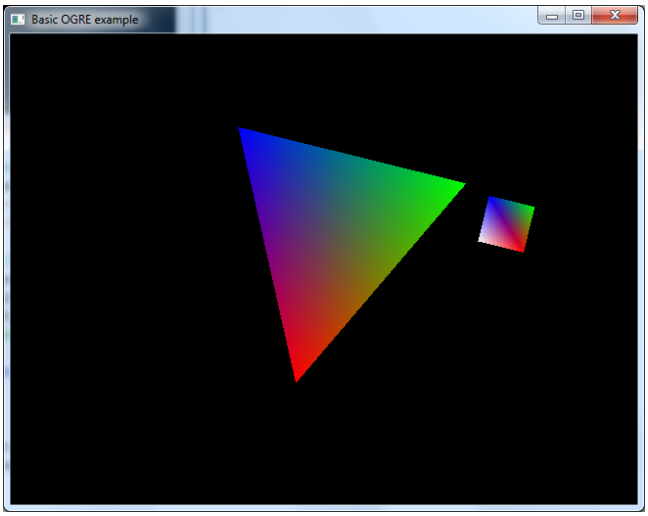
1. Create new `Ogre::ManualObject` object using `createManualObject()` method of the scene manager.
2. Describe vertices of your square. Look up in the documentation of the `Ogre::RenderOperation` class[2] which operation type you should use. Note that in OGRE we first create the vertex itself and then describe it's attributes.
3. Create `Ogre::SceneNode` and attach the new object to it.

[1] `http://www.ogre3d.org/`

[2] `http://www.ogre3d.org/docs/api/html/classOgre_1_1RenderOperation.html`

4. Use this `SceneNode` to animate our object (update it's potision) in the `frameRenderingQueued()` method, which is an analog of `idleFunc()` in GLUT.

The existing code for the triangle will serve you as example. The result should look something like this:



2 Lighting, materials and textures

Open project `2_OgreLighting`. The structure is same as before. Have a look at `createLitSphereScene()`, here we create a sphere, add materials to it and enable lighting. See how it is done. Now pay attention to these lines in the middle of `run()` function:

```
Ogre::ResourceGroupManager::getSingleton().
                    addResourceLocation("../data", "FileSystem");
Ogre::ResourceGroupManager::getSingleton().
                    initialiseAllResourceGroups();
```

Instead of writing material properties into the code, materials (and some other things) can be described in special *script* files. First line tells OGRE where to look for resources: various data files (textures, meshes, etc.) and scripts. Second line instructs it to read in resource descriptions and initialise *resource groups*.

Have a look at the `Examples.material` file in the `data` folder and try to apply some of materials described there to our sphere:

```
sphere->setMaterialName("Examples/WaterStream");
```

If you would like to try other examples you should download Ogre SDK[3] and add the resources needed for each particular example to our `data` folder.

Exercise 2 (0.5pt). Create new file `data/Sphere.material`. Here we will describe material for our sphere. Use `data/Examples.material` and `http://www.ogre3d.org/docs/manual/manual_16.html` to create a material script, which exactly reproduces material parameters we have specified in the code. The result should look exactly the same: red sphere with white specular spot on it. Note that you can use `WASD` keys to move the camera and mouse to look around.

Exercise 3 (0.5pt). In practice session #9, which was about different types of shadows, we implemented three different shadow techniques. You might remember that for implementing stencil shadows you first needed to create shadow volume objects and after that implement the logic by carefully playing with stencil, color and depth buffers. OGRE allows you to enable stencil shadows with literally one line of code[4]:

```
scene->setShadowTechnique(Ogre::SHADOWTYPE_STENCIL_ADDITIVE);
```

Create a plane (`Ogre::SceneManager::PT_PLANE`) below the sphere and enable shadows. Note that you can enable or disable whether specific object casts a shadow using `setCastShadows()` method of `Ogre::Entity` object.

Exercise 4 (0.5pt). Next let's see how we can map textures on our objects. The best way to do that is, again, by using `.material` scripts. Check out `data/Examples.material` for examples. Note that you can create animated textures just by adding one line (for example see `Examples/WaterStream`). Find your favourite picture in the internet and use it as texture for the plane.

Another popular use of texture is environmental mapping or cube mapping. Have a look at the tutorial[5] and add SkyBox or SkyDome to our scene.

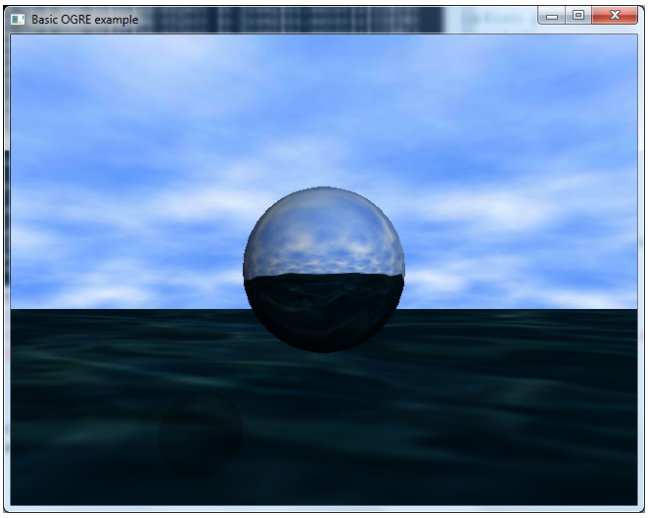
Exercise 5* (0.5pt). Look at the scene now. You might feel the urge to make sphere reflective. Study the code in `examples/CubeMapping/include/CubeMapping.h`. Here is an approximate description of the steps you need to do:

1. Make our `Application` class inherit `Ogre::RenderTargetListener`
2. Make use of `preRenderTargetUpdate()` method
3. Add some global variables
4. Create cube map texture in the same way as it is done in `createCubeMap()`

[3] `http://www.ogre3d.org/download/sdk`
[4] See more `http://www.ogre3d.org/tikiwiki/tiki-index.php?page=Basic+Tutorial+2`
[5] `http://www.ogre3d.org/tikiwiki/tiki-index.php?page=Basic+Tutorial+3`

5. Use `Examples/DynamicCubeMap` to create a new material in our `Sphere.material` script
6. Enable this material using `setMaterialName()` method

After completing exercises 3, 4 and 5 your scene will be something like this:



3 Meshes and animations

Now let us move to the next project and see how OGRE works with meshes. Open project `3_OgreMesh`. Meshes are handles as any other object we have seen before: they are entities, have `SceneNode` and you can specify materials and textures and shadows in the exactly same way as we have seen before. You can see that in `setupBody()`.

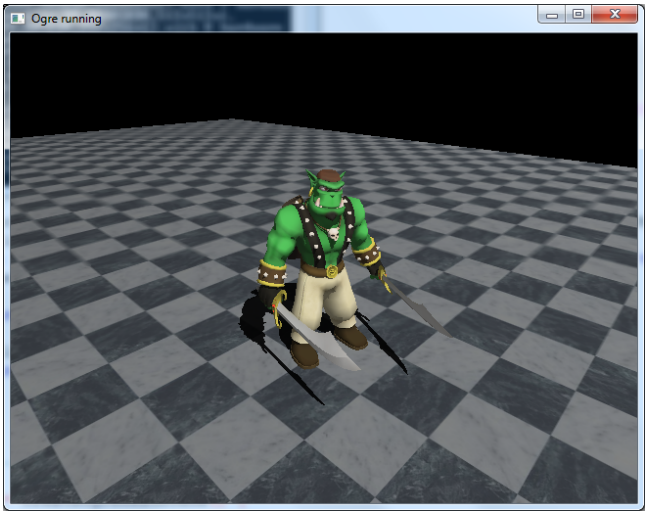
The code you see in this project is rather complicated and represents simplified and adopted version of `CharacterAnimation` OGRE sample application. You do not need to understand everything what is happening there, although it should not be hard.

Exercise 6 (0.5pt). Currently our character is only able to run forward and swing sword on left mouse click (this does not work until you draw swords out, which you will need to implement). Add the following functionality (coding should happen inside keyboard and mouse listener section and in `updateBody` method, you will see guiding comments in there):

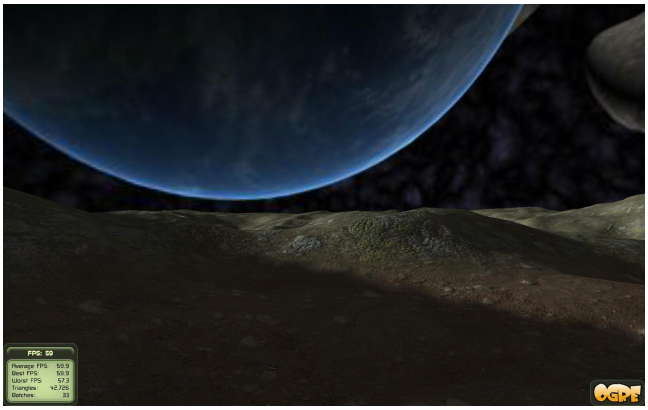
1. Enable movement in all four directions using `WASD` keys
2. Enable jumps on `Spacebar`
3. Draw swords on `Q`

4. On left mouse click the character swings sword vertically, but there is also horizontal swing animation available, make him do that on on right mouse click

5. Do now allow character to run off the floor



Exercise 7* (2pt). In OGRE official Basic Tutorial 3: "Terrain, Sky, and Fog"[6] you can learn how to create terrains, sky (as we already did in exercise 4) and fog. Like this:



Take character from the previous exercise and place him into such world. Choose details to your own liking: terrain, sky, light, shadows, textures, reflections, army of ogres, another creatures[7], mysterious reflective spheres in the middle of everything...

[6] `http://www.ogre3d.org/tikiwiki/tiki-index.php?page=Basic+Tutorial+3`
[7] Look inside `media/models` folder of OGRE SDK for more meshes and characters

4 Plugins

In OGRE you can get additional functionality by using plugins. Open project `4_OgrePlugins`. Have a look at the following line in the beginning of `run` function:

```
mRoot = new Ogre::Root("plugins.cfg");
```

It tells OGRE name of the file in `bin` directory where plugin configuration resides. Inside that file you will see how we enable plugin called ParticleFX. This plugin has a configuration file, which resembles `.material` files we have seen before. Open `data/Examples.particle` and study it.

Exercise 8 (0.5pt). To complete this exercise do the following:

1. Make particle source follow mouse movements. For that you will need to add mouse listener in the way analogous to the way how keyboard listener is added. See helpful comments in the code.
2. Play with different examples and their parameters, make the one you like best and describe it in new `.particle` file.