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 have 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.

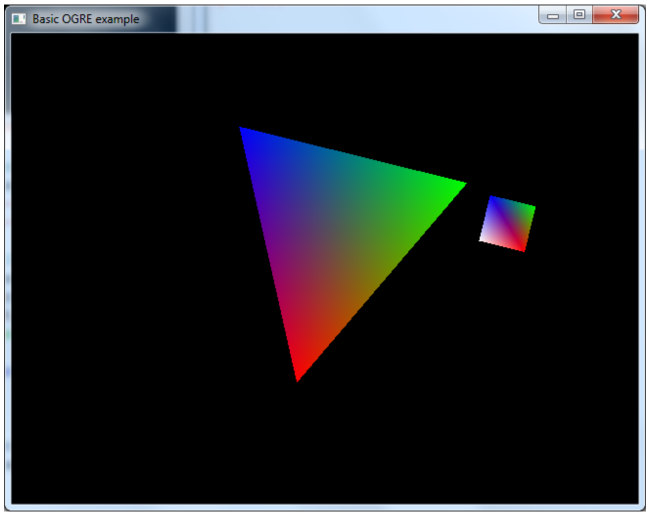
Exercise 1 (0.5pt). Add a small square which will fly around the triangle and rotate around its own center. For that you will need to

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 its 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 its 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`, where you 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 (1pt). Next, let us see how we can map textures on our objects. The best way to do that is, again, by using `.material` scripts.

1. 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 the texture for the plane.

2. Another popular use of texture is cube mapping. Have a look at the tutorial[4] and add SkyBox to our scene.

Exercise 4* (1pt). 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 to create dynamical cube mapping (texture is updated on each frame and you will see actual reflections of moving objects):

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()`

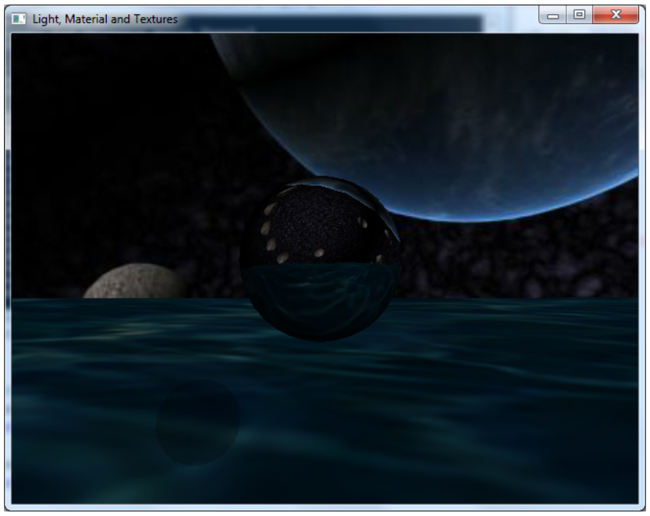
5. Use `Examples/DynamicCubeMap` to create a new material in our `Sphere.material` script

6. Enable this material using `setMaterialName()` method

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

[3] `http://www.ogre3d.org/download/sdk`

[4] `http://www.ogre3d.org/tikiwiki/tiki-index.php?page=Basic+Tutorial+3`



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 handled as any other object we have seen before: they are entities, they have `SceneNode` and you can specify materials, textures and shadows in the exactly same way as we have seen before. Have a look at `createOgreScene()` routine to see how mesh is loaded and entity is created.

Exercise 5 (0.5pt). Run the application to see an ogre standing in the middle of nowhere.

1. Create a plane (`Ogre::SceneManager::PT_PLANE`) and add any texture to it using new `.material` script.
2. Currently your texture is stretched along whole plane, use `texture_unit` attributes[5] to cover the surface with repeating images of the texture

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[6]:

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

3. Enable shadows for our scene

[5] `http://www.ogre3d.org/docs/manual/manual_17.html`

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

Note that you can control whether specific object casts a shadow using `setCastShadows()` method of `Ogre::Entity` object.

Exercise 6 (1pt). Next let us make our character move when we press `WASD` keys. Do the following:

1. Add new `Vector3` which will be used to store requested direction
2. Update this vector in `keyPressed` and `keyReleased` functions according to the user input
3. Update character position in `updateBody`

We also would like our character to turn his face to the same direction where he is moving. One way to that is:

1. Use `getOrientation().zAxis().angleBetween()` of object's `SceneNode` to calculate angle between current object orientation and requested direction.
2. Rotate the object using that angle.

Exercise 7 (1pt). In the `data/Character` directory among other things we have `Sinbad.skeleton` file. It describes armatures and 13 animations[7] for this character: `IdleBase`, `IdleTop`, `RunBase`, `RunTop`, `HandsClosed`, `HandsRelaxed`, `DrawSwords`, `SliceVertical`, `SliceHorizontal`, `Dance`, `JumpStart`, `JumpLoop`, `JumpEnd`. In this exercise we will implement `RunBase`:

1. Create a pointer which will be used to handle animation

   ```
   AnimationState* mRunningAnimation;
   ```

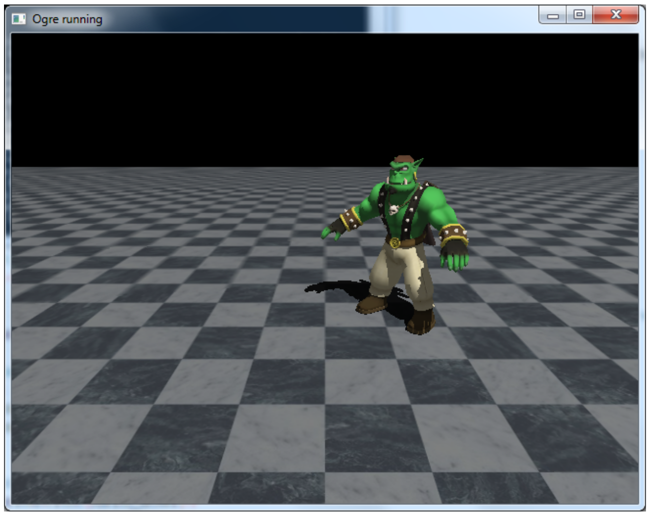
2. In the `createOgreScene()` routine get pointer to the animation named `"RunBase"`

   ```
   mRunningAnimation = mBodyEnt->getAnimationState("RunBase");
   ```

3. You can control phase of the animation with `addTime()` method of `AnimationState`. Use it in `animateObjects()` routine.
4. Use `setEnable()` method to enable and diable animation in `keyPressed()` and `keyReleased()` routines.

In total this animation can enabled with 5 lines of code (not counting few `if` statements, which you will probably need). Congratulations! You now have running ogre at your disposal.

[7] `http://www.ogre3d.org/docs/manual/manual_75.html`



The code you see in this project is simplified and adopted version of the OGRE sample application. You can find full version with all 13 animations, smooth transition between animations and floating camera in OGRE SKD under `Samples/CharacterAnimation/include/SinbadCharacterController.h`.

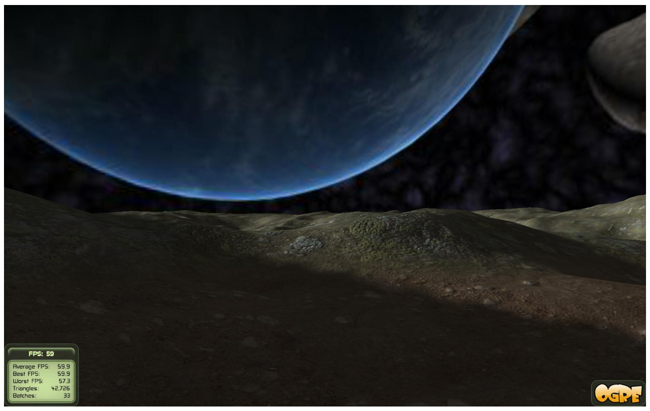
Exercise 8* (1pt). Study the code there and implement in our application idle animation, jump, smooth transitions and using the swords. Add camera behaviour if you like. This exercise relies heavily on your coding skills, if you and C++ are friends then this exercise should be mostly copy-pasting to correct places.

Complex meshes like the one we have been using, their skeletons and animations can be created with modeling tools like Blender and then exported to OGRE format using exporters[8] provided by OGRE.

Exercise 9* (1pt). It should be possible to export the mesh (and its animation) we have created on previous practice session to OGRE format and use in your application. Submit an application with self-made animated mesh to get a bonus point.

Exercise 10* (2pt). In OGRE official Basic Tutorial 3: “Terrain, Sky, and Fog”[9] you can learn how to create terrains, sky (as we already did in exercise 4) and fog. Like this:

[8] `http://www.ogre3d.org/tikiwiki/tiki-index.php?page=Blender+Exporter`
[9] `http://www.ogre3d.org/tikiwiki/tiki-index.php?page=Basic+Tutorial+3`



Take ogre character and place him into such world. Choose details to your own liking: terrain, sky, light, shadows, textures, reflections, army of ogres, another creatures[10] and mysterious reflective sphere in the middle of everything...

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 11 (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, create a particle system you like, describe it in a new `.particle` file and enable in the application.

[10]Look inside `media/models` folder of OGRE SDK for more meshes and characters