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:

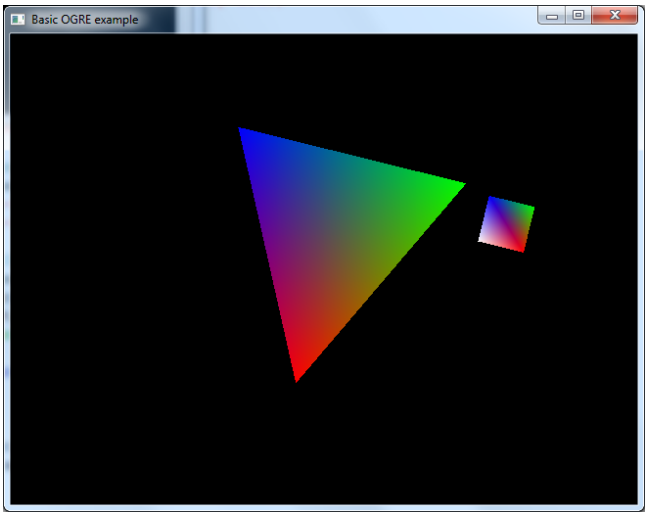
1. Create new `Ogre::ManualObject` object using `createManualObject()` method of the scene manager.
2. Describe vertices of your square. Look up in the documentation of the `Ogre::RenderOperation` class[2] which operation type you should use. Note that in OGRE we first create the vertex itself and then 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 configure our scene: create a sphere, add materials to it, enable lighting and shadows. See how it is done.

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[3]. Uncomment the following line to enable stencil shadows:

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

Now pay attention to these lines in the middle of `run()` function:

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

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

Instead of writing material properties in the code, materials (and some other things) can be described in a special *script* file. The first line tells OGRE where to look for the resources: data files (textures, meshes, etc.) and scripts. The 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[4] and add the resources needed for each particular example to our `data` folder.

Exercise 2 (1pt). Create two new files `data/Sphere.material` and `data/Plane.material`. Those will be used to describe materials of our objects.

1. Use `data/Examples.material` and `http://www.ogre3d.org/docs/manual/manual_16.html` to create a material script for the sphere, 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.

Next, let us see how we can use textures in OGRE. The best way to do that is, again, by using `.material` scripts.

2. See material examples in `data/Examples.material`, which have `texture_unit` section. Note how you can create animated textures just by adding one line in the material script (for example see `Examples/WaterStream`).

3. Describe the material with a texture in the `Plane.material` file. Find your favourite picture in the internet and use it as the texture for the plane.

Note that you can use `WASD` keys to move the camera and mouse to look around.

Exercise 3* (0.5pt). Another popular use of textures is cube mapping. For that you need 6 image files, which have common prefix `mytexture_` and six suffixes, one for each side of the cube: `bk`, `dn`, `fr`, `lf`, `rt`, `up`. See images `stevecube_*.jpg` in our `data` folder. You can use those or find/create your own. Have a look at the tutorial[5] and add SkyBox to our scene.

Exercise 4* (0.5pt). Look at the scene now. You might feel the urge to make the sphere reflective. Study the code in `examples/CubeMapping/include/CubeMapping.h`. Here is an approximate description of the steps you need to do to implement dynamical cube mapping:

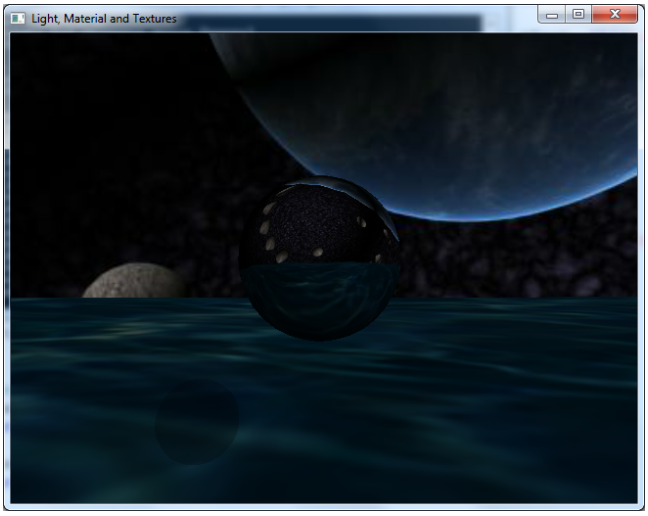
[4] `http://www.ogre3d.org/download/sdk`

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

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

Note how we use additional camera to update textures used for the cube. Since textures are updated on each frame you will get actual dynamic reflections.

After completing exercises 2, 3 and 4 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 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 (1pt). In the `data/Character` directory (which we use as resource location for this project) among other things we have `Sinbad.skeleton` file. It describes armatures and 13 animations[6] for this character: `IdleBase`, `IdleTop`,

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

`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 the animation

   ```
   AnimationState* mRunningAnimation;
   ```

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

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

3. Control phase of the animation with `addTime()` method of `AnimationState`. Use it in the `animateObjects()` routine.

4. Use `setEnable()` method to enable and disable the animation in `keyPressed()` and `keyReleased()` routines.

In total this animation can be configured with 4-5 lines of code (not counting few `if` statements, which you will probably need).

Exercise 6* (0.5pt). 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 the `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.

Congratulations! You now have running ogre at your disposal.

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 transitions between animations and floating camera in OGRE SDK under `Samples/CharacterAnimation/include/SinbadCharacterController.h`.

Complex meshes like the one we use here, their skeletons and animations can be created with modeling tools like Blender and then exported to OGRE format using exporters[7] provided by OGRE. Our Sinbad was created in Blender, so you can open `data/Character/Sinbad.blend` and do with it whatever you want. To access list of animations open "Dope Sheet" in one of the panels, Choose "Action Editor" and then you will see a drop-down list with all available animations.

Exercise 7* (2pt). This bonus exercise offers you a variety of interesting things you can do. Implement all of them, any of them or any combination of them. Depending on the amount of effort you can get 1-2 bonus points.

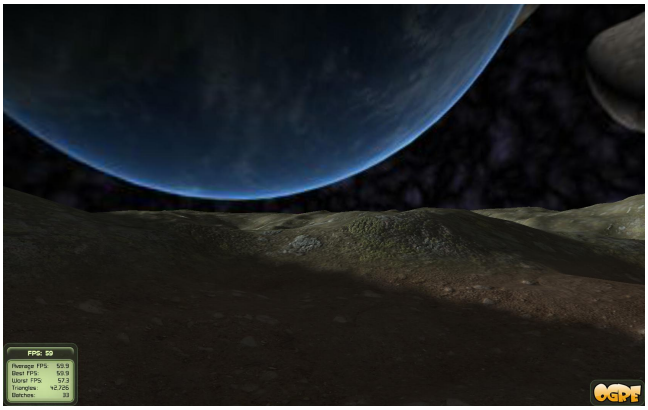
More animations
Study the code in ODRE SDK `Samples/Character` and add more animations to our application: idle animation, jump, smooth transitions, sword movements, etc. Add camera behaviour if you like. This exercise relies heavily on your coding skills, if you and C++ are friends then this exercise should be mostly copy-pasting to correct places.

Blender
It should be possible to export the mesh (and its animation) we have created

[7] `http://www.ogre3d.org/tikiwiki/tiki-index.php?page=Blender+Exporter`

during the previous practice session to OGRE format and use in it your application. Add self-made animated mesh to the scene.

Terrain
In OGRE official Basic Tutorial 3: "Terrain, Sky, and Fog"[8] you can learn how to create terrains, sky (as we already did in exercise 3) and fog resulting in a scene like this:



Take ogre character and place him into such world. Choose details to your own liking: terrain, sky, light, shadows, textures, reflections, army of ogres, other creatures[9] and mysterious reflective torus knot in the middle of everything...

4 Particle system plugin

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

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

It tells OGRE the name of the file in `bin` directory where plugin configuration is stored. 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.

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

2. Play with different examples and their parameters, make particles fly in random directions, describe it in a new `.particle` file and enable in the application.