Prof. G. Zachmann
P. Lange

University of Bremen
School of Computer Science
CGVR Group
June 24, 2014

Summer Semester 2014

Assignment on Advanced Computer Graphics - Sheet 5

Due Date 10. 07. 2014

Exercise 1 (Procedural Generation of Asteroids: Noisy Mesh, *4 Credits*)

Implement an algorithm which procedurally generates asteroid-like meshes. For reference, have a look at the picture below:

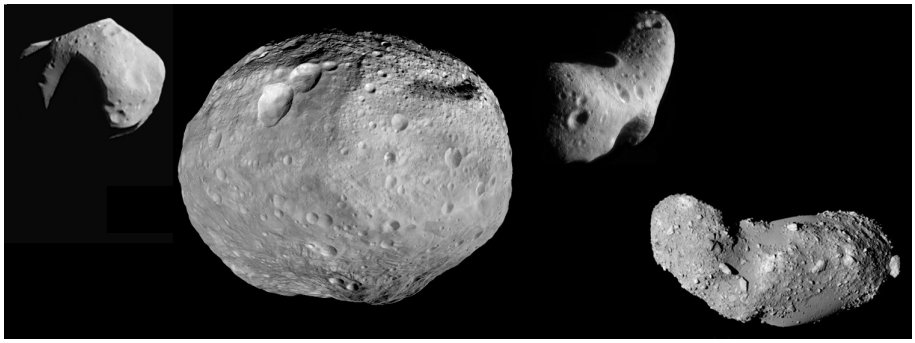

Figure 1: Real asteroids: Mathilde, Vesta, Eros and Itokawa.

The result of your algorithm from this task should look similar to this rendering:

Figure 2: Example rendering of an asteroid from the first exercise.

To do that, you can use the provided framework or you can implement your idea from scratch with your favourite rendering library.

The provided framework is written in Ogre3D version 1.9. You can find the corresponding download for your operating system at `http://www.ogre3d.org/download`. Additionally, a guide for configuring your IDE can be found at `http://www.ogre3d.org/tikiwiki/tiki-index.php?page=Setting+Up+An+Application&structure=Development`.

The framework is a minimalistic Ogre class named AsteroidFramework with the following functionalities, which are already implemented:

- Configuring Ogre: All configuration of Ogre is done in `AsteroidFramework::configureOgre(void)` like enabling ressource groups or starting the rendering window. A configuration window will be displayed where you can adjust your settings (resolution, graphics library, etc.).
- Inputs: The framework handles mouse and keyboard inputs. You can:
  -Move inside the environment with WASD movement.
  -You can orientate the field of view with your mouse.
  -Change the rendering view with 'R' to see the textured mesh,
  the vertices of the mesh or the points of the mesh. You can use this for debugging purposes.
- Scene Management: The overall scene is created in `AsteroidFramework::createScene()` which also already calls your asteroid generation methods. Additionally, the camera and lighting is already defined. The most important instance variable of the framework used here is the `Ogre::SceneManager`, which manages all scene nodes (`Ogre::SceneNode`).
- The Framework also comes with a noise function. You can use the noise function via using the class `SimpleNoise::GetHeight(x,y)` which returns the noise for a given x and y coordinate. You can also use the libnoise library, which can be downloaded at `http://libnoise.sourceforge.net/`.

Your tasks in detail:

1. Implement a suitable algorithm which procedurally generates asteroid-like meshes in `OgreFramework::createAsteroidMesh()`. Propose a suitable algorithm which gets as input several parameters: the maximum size (e.g. between 1 and 50), roughness (between 0.0 and 1.0 for very smooth and very rough) and elongations (a multiplier, e.g. between 1 and 3) for each of the three axes. Remark that the size and the elongations could also be achieved with a parent `Ogre::SceneNode` which could scale your asteroid but in this task you should really generate and operate on the mesh vertices.
2. You can find an example implementation for generating a sphere mesh in `OgreFramework::createSphere()`. It is probably helpful for you if you look at this example and start your implementation from there. It uses spherical coordinates to create sphere mesh and also creates texture coordinates.
3. With the proposed spherical approach you will get some spheres which will look like deformed ellipsoids. Use `SimpleNoise::getHeight()` to apply noise to your generated mesh to make the mesh more interesting.
4. The generated meshes should have a very strong resemblance to asteroids. Have a look at the reference pictures of real asteroids.
5. Generate 5 different asteroids and document your results with screenshots. Give also a range for each of your input parameters which creates meshes which have a strong resemblance to real asteroids.

Exercise 2 (Procedural Generation of Asteroids: Enhanced Mesh, *3 Credits*)

With the proposed method above, all of your generated meshes will have a strong resemblance of spheres, even with the noise applied. However, real asteroids can have also a banana-like form:

Figure 3: Close-up of Eros.

Your task:

- Improve your algorithm of exercise 1 to generate asteroid meshes that resemble real asteroids more closely. In the lab meeting, we will discuss some possibilities how to do that, but we will appreciate it very much if you invent your own methods. Probably, you will need to add more parameters to `OgreFramework::createAsteroidMesh()`.
- Generate 5 different asteroids and document your results with screenshots. Give also a range for each of your input parameters which creates meshes which have a strong resemblance to real asteroids.

Exercise 3 (Procedural Generation of Asteroids: Bump Mapping 1, *3 Credits*)

The Ogre framework shows in `AsteroidFramework::createScene()` how a texture can be applied to a mesh. Extend the implementation and the corresponding material file to a bump mapping approach which adds craters and more details to your generated asteroid-like mesh.

For that, procedurally generate suitable height maps which can have arbitrary amount of craters, varying in their size. The pictures below show how such a height map could look like.

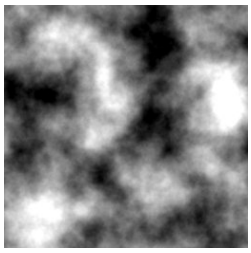

Figure 4: Example height map without craters. Similar height maps can be generated with the framework noise class or with a the libnoise library.

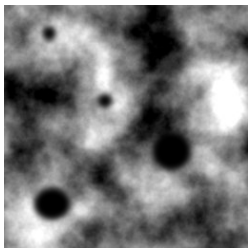

Figure 5: Example height map with some craters.

Your tasks in detail:

1. Implement a suitable algorithm which procedurally generates such a height map which can be applied as bump mapping for your generated asteroid in `OgreFramework::createAsteroidBumpMaps()`. Propose a suitable algorithm which gets as input several parameters: the amount of craters as well as the minimum and maximum size of each of them.
2. In the first step you should use a noise to generate a height map like in figure 4. Second, you should place some unit circles. For better results you should blur the transitions between generated circles and underlying height map.
3. In Ogre you can generate dynamic textures with `Ogre::TexturePtr`. You can define the pixels of the height map texture via calling `TexturePtr::getBuffer()`. For saving a dynamic texture you can use `Ogre::Image`. The functions `AsteroidFramework::saveImage` and `AsteroidFramework::generateTexture` show how you can use these Ogre classes.
4. To enable your generated texture, you have to implement a corresponding `.material` file, which has to be stored in the Ogre media folder. The framework itself comes with a Phong shading in `SpherePhong.material` which you can adapt and extend.
5. Apply your bump mapping to your 10 generated asteroids and document your results with screenshots.