Assignment 5 - Planetary System

Due Apr 24 by 11:59pm **Points** 100 **Submitting** a file upload **Available** after Apr 4 at 12am

Exercise 5 – Planetary System

Please read the following guidelines carefully before starting – I advise not to postpone it and start right away.

You can ask me any question to help you solve any misunderstanding or unclear specifications, or give you possible directions if needed.

General Guidelines:

- Submission should be made electronically as before.
- Any theory should be added in Word/PDF and sketches can be added as jpg attached to the document
- Submission of this exercise is **in couples as set**
- Partners should implement the project together with full knowledge of the entire code.

Submission Guidelines:

- The project should be submitted as a zip file with the exercise and the last name of the students – for example '**Exercise4_Maverick_Goose.zip**'.
- **The root directory within the zip should be the zip file name** as indicated in 1.
- As with previous assignments, the submitted zip file should contain**only files needed to build the project and run it**. In other words, the project files, the sources and headers and the resource files (textures, models…) - no intermediate files or directories (such as sdf, obj and others) should be included – review the zip files I created for the book projects as an example.
- A working release version of your application (.exe file) should be included in the zip and should not rely on any file on my computer to be able to run.

Submission Date:

Submission by end of Friday, April 24th 2015 midnight.

Overview

This exercise is the final exercise and covers many of the elements learned during the course and is a good prep before the final test.

The goal of the exercise is to create a planetary system consisted of a sun, several planets circling the sun, and each planet has one or several moons.

The exercise should be based on the previous project sources and enhances the sphere class.

Goals of the exercise:

- Implement normal mapping
- Create planetary system with usage of matrix hierarchy and objects trees (recursive matrix update and multiplication)
- Use height maps manipulate mesh vertices via the VS
- Use alpha blending, masks and blending modes
- **Have fun creating amazing visual effects!** Use your imagination to create variety of planets and moons of different textures, alpha masks, materials and blending effects:Create a new type of camera system
  - Fire (inner solid sphere of main file texture, several outer transparent spheres with additive blend)
  - Ice (color texture, height map, normal map)
  - Water (several spheres containment with various water alpha blended texture)
  - Earth-like / Moon / Mars / regular planet (color texture, height map, normals map)
  - Other blazing effects will add bonus points
-

Specifications

- **Planetary system:**
  - Should consist of single or double sun rotating around their center / each other.
  - At least 3 different planets rotating around the sun and revolving around their center.
  - All planets should have moons – at least one should have 2 moons.
  - The planets and moons should be lit by the sun (hence – the sun is the source of light), with exception of fire planet / moon.
  - The sun and one planet / moon should be 'fire' stars/planets (think about how to implement this – ask me ahead of time any question).

- **Inheritance and objects tree:**
  - The BaseObject should be enhanced to include a vector / list of pointers to the object's sub-tree of children.
  - Call this vector / list m_Children
  - Since every model has this children vector, and each child repeatedly has that – an entire 'hierarchical tree' can be implemented following this concept.
  - Each of the updating methods should go over the children and call this method – this way the entire planetary system can be updated.
  - Short explanation: in our planetary system the sun is the root, its children are the planets of this system, and their children are the moons surrounding each.

- **Required materials for the various object types**:

  Notice - each type should have a separate shader!

- Sun / fiery planet:
  - Simple material - No Height map or normal is required
  - Uses additional 2 or more external rotating spheres with semi-transparent noise/cloud textures that will simulate the fire (additive blend for glow – SrcAlpha, DstOne)
  - Ice / rock / earth planets:
    - This material and object should include full material capabilities: height, normals and the general equation of lighting with various types of shininess according to the desired visual looks. Geometric deviation should be applied via VS and lighting will include normal mapping via PS.
    - Water planet:
      - Usage of blending levels of normal maps / height maps is up to you.
      - Like in the fire case – use additional 2 geometric layers or more, this time the blend should be more regular and towards constant transparency (SrcAlpha, InvSrcAlpha), or constant (simple transparency for texture with some features).

Additional Requirements

- **Camera Design and UI Control**
  - Look at is the origin of the planet / star coordinate system.
  - The distance from the center should be set via parameter – name it m_CameraRadius.
  - Mouse movement should rotate the camera in regards to the local origin (of the moon / planet / star).
  - Mouse zoom will be done with center mouse roll and will changes the camera radius from center.
  - Moving between moon / planet / star as explained in the UI section.
  - Camera transition between the two objects should be smooth – traversing the camera from one point in space to another along a path that will be pleasing to the eye (you can apply one of many techniques / movements methods).

- **General Coding guidelines**
  - Creation and control parameters (radius / size, rotations, resolution, colors…) should be exposed through the constructor and setter methods – **they should not be hard coded in your equations!**

- **Rotation and movement rules:**
  - Any planetary object should have rotation around its axis
  - The sun center should be placed at the (0,0,0) of the world coordinates system, however, as any other object, the sun should also revolve around its axis.
  - The self-rotation **does not** affect the children – if a planet revolve around itself, it does not rotate the moons according to its rotation.
  - Each planetary object should have a rotation radius and speed in regards to its gravitational parent. Unlike self-rotation, this transformation **does** affect the children – as a planet circles the sun, the moons are moving along with it.
  - All transformations should be combined from class variables (for example – m_SelfRotationAxis, m_SelfRotationVelocity, m_OrbitAxis, m_OrbitRadius, m_OrbitVelocity).
  - All transformation should be represented and combined using matrices, for example – m_SelfMatrix, m_OrbitPlacementMatrix (question – why do we need two matrices?)

- **Lighting**
  - Should be implemented **via the pixel shader** (the Phong model – per pixel lighting)

- **Control Keys**
  - **W -** Switch between Solid render and Wireframe by pressing
  - **L –** switches between level (star/planet/moon)
  - **Number keys** - select between the viewed object at this level (‘1’, ‘2’, ‘3’)
  - **O / P** - change self-rotation angular velocity of the selected object
  - **K / L** - change orbit rotation velocity (around a parent) of the selected object
  - **Mouse wheel** – increase / decrease the distance of the camera from the selected object

Hints and Pointers

- Using blending:
  - Make sure to use image format that supports alpha channel when you are using it in the blending (png, dds, tga, enhanced bmp…). You can edit it using Photoshop.
  - Enable the blending render state and set the blend function for the source and destination. I suggest going over chapter 12 in the book.
    - Example (blend on, blend function creates transparency):

```
HR(gd3dDevice->SetRenderState(D3DRS_ALPHABLENDENABLE, true));
HR(gd3dDevice->SetRenderState(D3DRS_SRCBLEND, D3DBLEND_SRCALPHA));
```

HR(gd3dDevice->SetRenderState(D3DRS_DESTBLEND, D3DBLEND_INVSRCALPHA));

- Do not forget to disable the blend mode once you finished rendering all models that needed blend (so that models without blend will be rendered correctly)

- Texture read in the vertex stage:
  - Follow the terrain example to see how vertices position can be manipulated via texture reading.
  - You can use the function **tex2Dlod** to read texture in the vertex stage.
  - Make sure that both PS and VS are set to version 3.0 or higher to allow the support of this capability.

Grading (goes to 110%)

- **[15%] Correct zip, file structure and exe runs properly**
- [5%] Clean code standards
- [25%] Camera control and smooth transitions
- [10%] All UI modes and model fully functioning
- [45%] 15% per each planetary render techniques – all textures and effects should match and work together towards the specified look!
- [10%] Make me wow extra cool feature(s)!