Atmospheric Scattering
Practical Course Visual Computing

Michael Becher

This document describes a branch of space-lion that has been specifically created for an exercise given in a practical course at University of Stuttgart.

Manual

This section gives a brief overview of how to compile and run the program. The most recent source code and all necessary resources can be found at https://github.com/invor/space-lion/tree/fapra.

Compiling

Compiling the source code using the supplied solution file requires Visual Studio 2013. The solution file is located in the `vs2013` folder. For a successful compilation, all external dependencies (glfw, glew and glm) have to be present inside the `2013/include` and `vs2013/lib` folder. The `include` folder should contain the following:

- `GL/glew.h`
- `GL/glxew.h`
- `GL/wglew.h`
- `GLFW/glfw3.h`
- `GLFW/glfw3native.h`
- `glm/...`
  (there are simply to many files in my glm folder, but it's really just the standard include folder that comes with glm)

The `lib` folder should contain the following:

- `glew32.lib`
- `glew32s.lib`
- `glfw3.lib`

Note to exercise examiners:
All files should already be present in the SVN repository. If you have downloaded or pulled a more recent revision from Github which will not contain these files, you can still look up the required files and folder structure in the SVN.

The solution file features three projects: engine, fapra and fault-tolerant-visualization. Each of those can be compiled and run on its own. However, only the fapra project has to be compiled for this exercise. While compiling the others should generally work, it will most likely produce a lot of additional warnings not related to the engine core or the fapra project. Known and expected warnings during compilation of the fapra project are *warning C4503 : decorated name length exceeded, name was truncated* caused by the ResourceManager class and furthermore a couple of warnings caused by the fbx loader.

After compilation, the .exe file should be located in a `bin` folder, which is on the same level as the `vs2013` folder (i.e. top-level). Running the program requires the `glew32.dll`.

Usage

Make sure to have copied the `glew32.dll` to the `bin` folder, if it isn't globally accessible on your machine. Start the program using the `fapra.exe`. After the program has started, the application window will be white at first (during shader compilation, resource creation and atmosphere precomputation), but after a short time you should be able to just barely make out the terrain. This is due to the simple day-night cycle, that starts at midnight and takes 60 seconds for a full rotation around the sun (note: we aren't actually moving, the sun direction is). After about fifteen second the horizon should brighten.

If the program is compiled without the '`#define CAMERA_ANIMATION`' flag in `fapra_renderHub.h` it features basic mouse controls for an orbital camera:

- Rotate the camera around the focus point by holding the right mouse button while moving the mouse
- Move closer or away from the focus point by turning the mouse wheel
- Translate the camera on the view plane by holding the middle mouse button while moving the mouse

Implementation

The major (novel) parts of the implementation are compromised of four C++ classes embedded into space-lion, two normal shader programs as well as two compute shader programs.
Shader program sources are stored in `resources/shaders(/fapra)`.
The relevant C++ sources are stored in `src/fapra` and `src/engine/core`.

FapraRenderHub Class

Files: `fapra_renderHub.h` and `fapra_renderHub.cpp`.
Derived from the RenderHub class. This class takes care of window and context creation, as well as scene and framebuffer handling. It's *void run()* method loads the scene and all necessary resources and implements the main/render loop.

PlanetaryScene Class

Files: `planetaryScene.h` and `planetaryScene.cpp`.
Derived from the Scene class. A specialised scene that contains a terrain and atmosphere object and the required methods for initializing and rendering a planetary scene.

Terrain Class

Files: `terrain.h` and `terrain.cpp`.
This class stores the mesh object, terrain properties such as size and altitude range (currently without a meaningful metric) and pointers to the terrain heightmap and material. Naturally it also contains the methods for initializing and rendering the terrain.

Atmosphere Class

Files: `atmosphere.h` and `atmosphere.cpp`.
This class stores the atmosphere attributes, the precomputed tables, pointers to the pre-computation shader programs and the Mesh for a screen-filling quad. Again, it also contains all methods for initializing and rendering the atmosphere.

Terrain Program

Files: `terrain_v.glsl`, `terrain_tc.glsl`, `terrain_te.glsl` and `terrain_f.glsl`.
This shader program is used for rendering a heightmapped terrain to a framebuffer object with multiple render-targets (essentially a G-Buffer, but currently not very sophisticated). Uses tessellation for increased detail. Since a single, instanced quad is used for the terrain, the position of vertices is determined and set in the tessellation evaluation stage.
Note: To match the size used for rendering the atmosphere, the terrain is currently scaled down by a factor of 10 in the shader.

Atmosphere Program

Files: `genericPostProc_v.glsl` and `sky_f.glsl`.
This shader program renders a planets atmosphere using precomputed tables and directly composits it with the terrain rendered in a previous pass.

Transmittance Program

File: `transmittance_c.glsl`.
A compute shader program for approximating the transmittance integral.

Inscatter Program

File: `inscatter_c.glsl`.
A compute shader program that computes the inscatter integral for a single scattering event.

Changes in the engine core

Some modifications and additions to the core classes of space-lion were necessary during this project. Most notably this includes support for tessellation and compute

shaders in the ResourceManager. Other changes were limited to a small tweak of the Material class and the addition of several shader program types in the GLSLProgram class and ResourceManager.

Future Work

A quick selection of noteworthy (but not required) features and aspects that couldn't be implemented in time.

- Multiple Scattering to increase realism and accuracy
- Precompute irradiance table for terrain light (only direct sunlight is computed)
- Correct blending of atmosphere and terrain. Due to the limited terrain size, it fortunately makes little visual difference at the moment.
- Proper rendering the sun disc
- Generic handling of planetary bodies for arbitrary scenes (no special planetary scene)
- Generate high-resolution terrain heightmaps
- ...