Atmospheric Scattering
Practical Course Visual Computing

Michael Becher

Manual

Compiling

Compiling the source code using the supplied solution file requires Visual Studio 2013 due to several C++11 features that are in use.
For a successful compilation, all external dependencies (glfw, glew and glm) have to be present inside `2013/include` and `vs2013/lib`. The `include` folder should contain the following:

- `GL/glew.h`
- `GL/glxew.h`
- `GL/wglew.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 for 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 ilnside the solution has to be compiled. 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.

Usage

Make sure to have copied the `glew32.dll` to the `bin` folder. Start the program using the `fapra.exe`. After the program has started, the screen will stay black for a short time. 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 15 second the horizon should lit up.
The program 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.

FapraRenderHub Class

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 initializes the scene and all necessary resources and implement the main/render loop.

PlanetaryScene Class

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

Atmosphere Class

Terrain Program

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 details.

Atmosphere Program

This shader program renders a planets atmosphere using precomputed tables and directly composits it the terrain rendered in a previous pass.

Transmittance Program

A compute shader program for approximating the transmittance integral.

Inscatter Program

A compute shader program that computes the inscatter integral for a single scattering event.

Future Work

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

- Multiple Scattering to increase realism and accuracy
- Irradiance table for terrain light (only direct sunlight is computed)
- Rendering the sun disc
- Generic handling of planetary bodies for arbitrary scenes (no special planetary scene)
- ..