

## 1 Zielsetzung

Ziel war es einen virtuellen Globus mit den Methoden der Computergrafikvorlesung zu implementieren. Hierfür wurde C++ und OpenGL verwendet.

## 2 Implementierung

Startpunkt bildete das [GLFW Getting started](#), wobei inzwischen kaum noch etwas hiervon erhalten ist.

### 2.1 Kugel-Mesh

Die Kugel, welche für die Darstellung von Planete(n), Monde(n) und der Sonne gedacht ist, wird durch rekursives unterteilen der Dreiecke eines Ikosaeders generiert. Zusätzlich werden für anschließendes Texturemapping entlang eines Längengrades die Knoten dupliziert. All diese Algorithmen sind in `sphere.cpp` zu finden und wurden so implementiert, dass keine unnötigen Knoten erzeugt werden, welche die Performance einschränken.

Für die weitere Verarbeitung wurde ein `struct Mesh` implementiert, welches die Attribute der Kugel einem `VERTEX_ARRAY_OBJECT`, `VERTEX_BUFFER_OBJECT` und einem `INDEX_BUFFER_OBJECT` zuordnet. Anschließend kann ein Mesh dann relativ unkompliziert mit `glDrawElements` gezeichnet werden.

### 2.2 Cubemap

Um das umliegende Universum darzustellen wird eine standardmäßige Cubemap verwendet.

### 2.3 Texturen

Für die Texturen von Erde, Mond und Universum wurden öffentlich zugängliche Bilder von NASA genommen.

Idee war eine Erde mit dynamischem Wetter darzustellen, wofür eine [Simulation von NASA aus dem Jahr 2005](#) verwendet wurde. Die Auflösung beträgt 5760x2881 jede Stunde, was ungefähr 210GB entspricht. Für schnellere Updateraten wurden die Grafiken aber auch mit `ffmpeg` gedownscaled. Des Weiteren wurde auch eine [Lichtkarte](#) von NASA verwendet. Auch für das

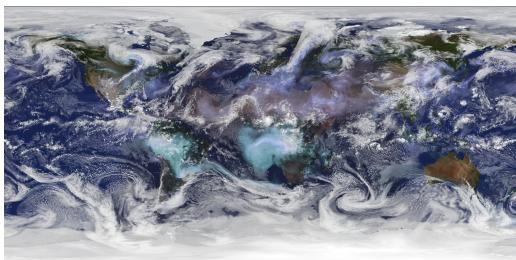


Abbildung 1: Erde bei Tag



Abbildung 2: Erde bei Nacht

[Universum](#) und den [Mond](#) wurden NASA-Daten verwendet.

Alle Texturen wurden mithilfe von `stbi` in den Funktionen `loadTexture` und `loadCubemap` geladen bzw. für die Wettersimulation wurde `updateTexture` verwendet.

## 2.4 Shading

Der Shadercode wurde in Files .vs (VertexShader) und .fs (FragmentShader) gespeichert und von `shader.h` compiliert und verwaltet. Bei der Erde wird linear die Wettersimulation mit der Lichtkarte bezüglich des Skalarproduktes von Normalen und Lichtvektor interpoliert. Somit entsteht abhängig von der Lichteinstrahlrichtung Tag und Nacht.

## 3 Ergebnisse



## 4 Ausblick

### 4.1 Kamera

Momentan ist die Kamera noch nicht frei steuerbar, da dies sich nur bei einer höheren Framerate anbietet. Mehr dazu [4.3](#)

### 4.2 Atmosphäre

Es wäre schön wenn optische Effekte, wie das [atmosphärische Streuen](#) im Shader implementiert wären.

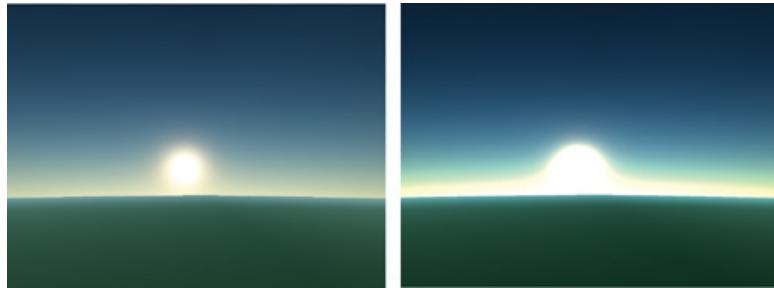
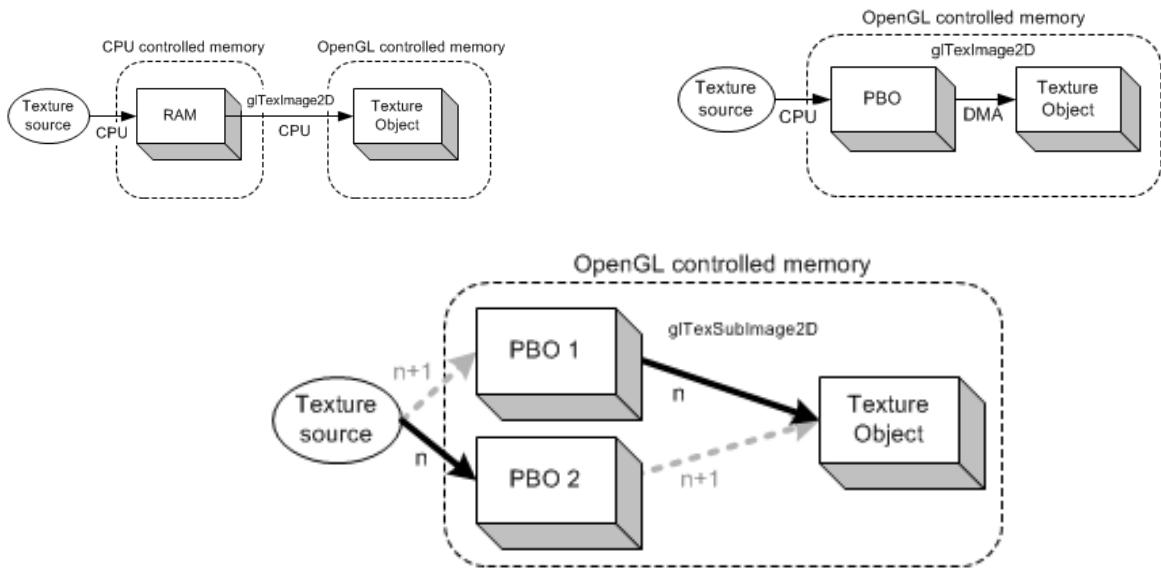


Abbildung 3: Atmosphärische Effekte

## 4.3 Performance

Bottleneck ist das Dekomprimieren/Laden zu CPU/GPU von den doch sehr großen Texturen. Lösung könnte das nicht komprimierte Speicher von den Texturen sein, was offensichtlicher Weise aber auch den Speicherverbrauch. Ansonsten wäre eine naheliegende Lösung das Dekomprimieren der Texturen zu parallelisieren. Um die Übertragung von RAM zur GPU zu beschleunigen wäre es noch ratsam zwei PixelBufferObjects zu verwenden. Hierbei beschleunigt sich das Laden durch asynchronen Direct Memory Access.



## 5 Source

## Code

Das Projekt findet man auf <https://github.com/halbrech/ComputergrafikProjekt>