Solarsystem

Dokumentation

Kuba & Erni

TGM

Inhaltsverzeichnis

1 Projektbeschreibung

1.1 Anforderung

Es soll eine einfache Animation unseres Sonnensystems erstellt werden!

In einem Team (2) sind folgende Anforderungen zu erfüllen.

- Ein zentraler Stern
- Zumindest 2 Planeten, die sich um die eigene Achse und in elliptischen Bahnen um den Zentralstern drehen
- Ein Planet hat zumindest einen Mond, der sich zusätzlich um seinen Planeten bewegt
- Kreativität ist gefragt: Weitere Planeten, Asteroiden, Galaxien,...
- Zumindest ein Planet wird mit einer Textur belegt (Erde, Mars,... sind im Netz verfügbar)

Events:

- Mittels Maus kann die Kameraposition angepasst werden: Zumindest eine Überkopf-Sicht und parallel der Planentenbahnen
- Da es sich um eine Animation handelt, kann diese auch gestoppt werden. Mittels Tasten kann die Geschwindigkeit gedrosselt und beschleunigt werden.
- Mittels Mausklick kann eine Punktlichtquelle und die Textierung ein- und ausgeschaltet werden.
- Schatten: Auch Monde und Planeten werfen Schatten.

Hinweise zu OpenGL und glut:

- Ein Objekt kann einfach mittels glutSolidSphere() erstellt werden.
- Die Planten werden mittels Modelkommandos bewegt: glRotate(), glTranslate()
- Die Kameraposition wird mittels gluLookAt() gesetzt
- Bedenken Sie bei der Perspektive, dass entfernte Objekte kleiner - nahe entsprechende größer darzustellen sind.
  Wichtig ist dabei auch eine möglichst glaubhafte Darstellung. gluPerspective(), glFrustum()
- Für das Einbetten einer Textur kann die Library Pillow verwendet werden! Die Community unterstützt Sie bei der Verwendung.

1.2 Teammitglieder

Andreas Ernhofer

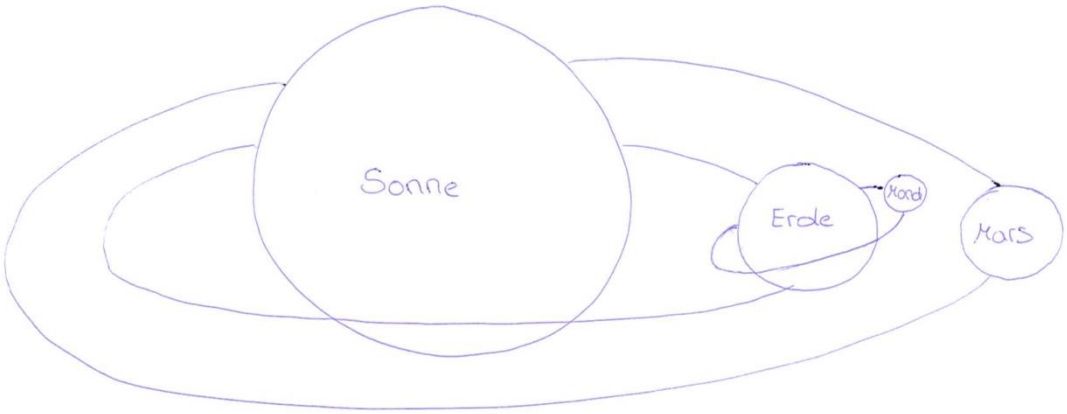
Jakub Kopec

1.3 Tools

Als IDE verwenden wir PyCharm. Um die Aufgabe erfüllen zu können, haben wir zusätzlich folgende Pakete installiert:

Package	Version	Latest
Pillow	3.0.0	3.0.0
PyOpenGL	3.1.0	3.1.1a1
PyOpenGL-accelerate	3.1.0	3.1.1a1
Sphinx	1.3.1	1.3.1
pygame	1.9.2a0	

2 GUI-Skizze



3 Evaluierung der Frameworks

3.1 Panda3D

Die Panda3D Game Engine nimmt Spieleentwicklern einen Großteil der immer wiederkehrenden Aufgaben ab. Zum Beispiel das Laden von Spielfiguren und Sounds, grundlegende Bewegungssteuerung und einiges mehr. Die Installation der Panda-Engine gestaltet sich einfach. Außerdem sind schon Funktionen wie beispielsweiße eine Kameraführung und Bewegungsabläufe bereitgestellt.

3.2 Pygame

Pygame ist eine Sammlung von Python-Modulen, die für das Schreiben von Spielen konzipiert wurden. Pygame verfügt über Funktionalitäten der SDL-Bibliothek. Dies ermöglicht die Erstellung von Spielen mit umfangreichen Funktionsumfang und Multimedia-Programmen in der Programmiersprache Python. Pygame ist sehr portabel und läuft auf nahezu jeder Plattform und Betriebssystem. Pygame selbst wurde bereits über eine Millionen Mal heruntergeladen und hat Millionen Besucher auf ihrer Website.
Pygame ist kostenlos. Unter der LGPL Lizenz veröffentlicht können Open-Source, kostenlose, Freeware, Shareware und kommerzielle Spiele kreiert werden.

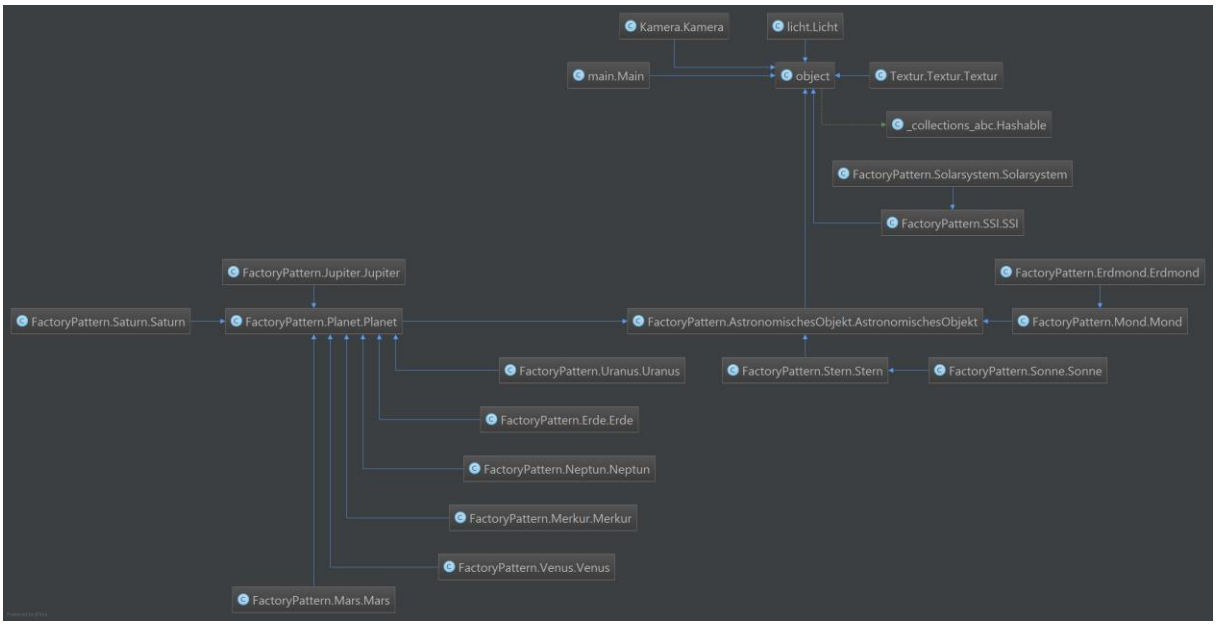
3.3 Conclusio

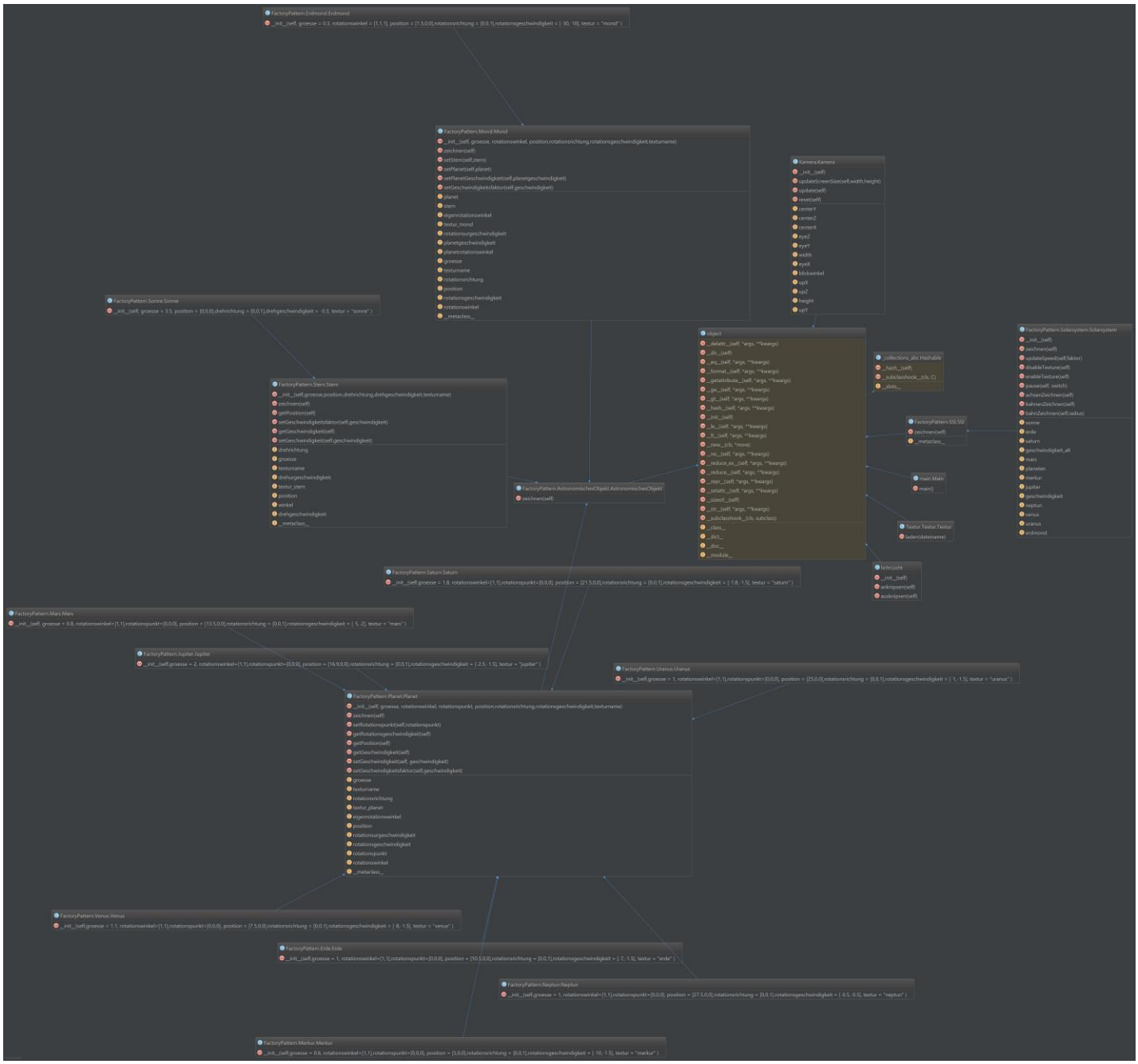
Der Einstieg mit Panda3D war ziemlich einfach und schnell. Da sich jedoch der Großteil der Klasse bereits für dieses Framework entschieden hat, dachten wir es könnte nicht schaden etwas Anderes auszuprobieren. Außerdem wollten wir neue Erfahrungen Sammeln und daher auch möglichst viel selber erstellen und nicht alles von bereits zur Verfügung gestellten Methoden verwenden. Unsere Wahl fiel daher auf **Pygame**.

4 Technische Dokumentation

4.1 Klassen

Übersicht:





siehe auch http://1drv.ms/1OTCf5y

4.2 Designpatterns

Factory Pattern

Für das Erstellen von Erde, Mars, etc. haben wir ein Factory Pattern verwendet. Als Grundvorlage liegt ein Astronomisches Objekt vor. Aus diesem haben wir Vorlagen für Planeten, Sterne und Monde erzeugt. Aus diesen erzeugen wir Sonne, Merkur, Erde, Erdmond, usw.

5 Umsetzung

Dieser Teil des Protokolls soll zeigen wie vorgegangen wurde.

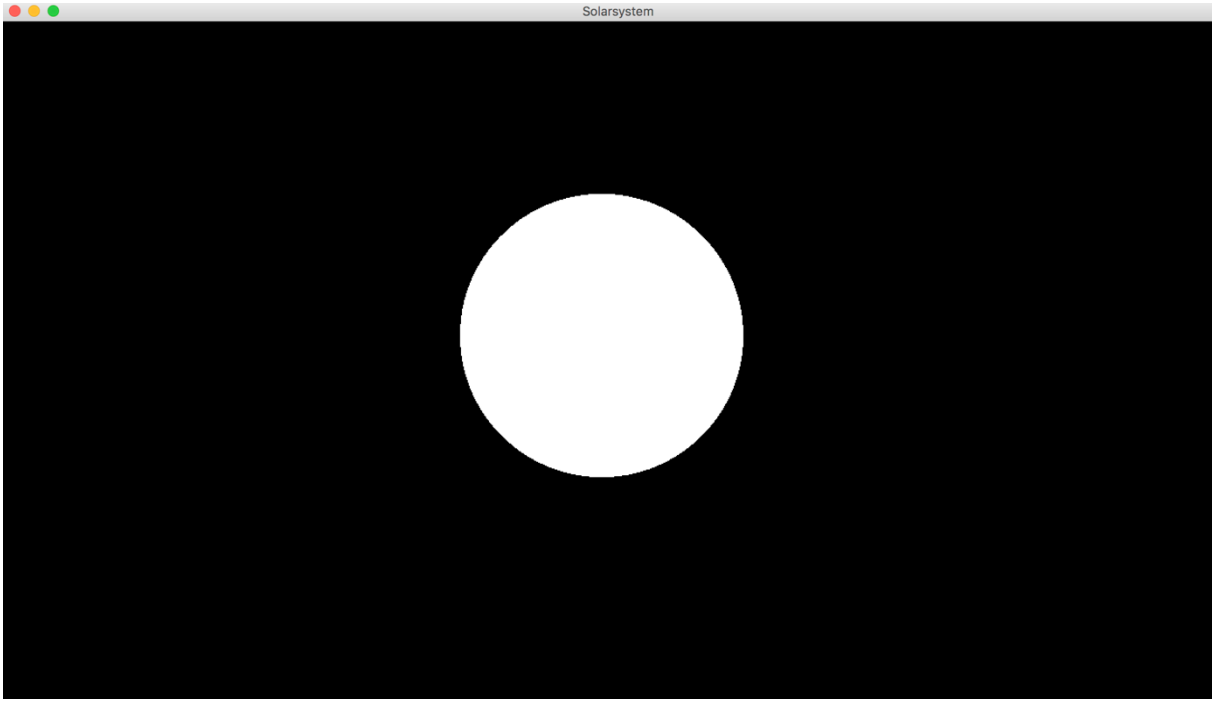
5.1 Zeichnen eines Objektes

Das zeichnen einer Kugel funktioniert (wie bereits in der Angabe erwähnt) mit folgendem Befehl:

```
glutSolidSphere(1,10,10)
```

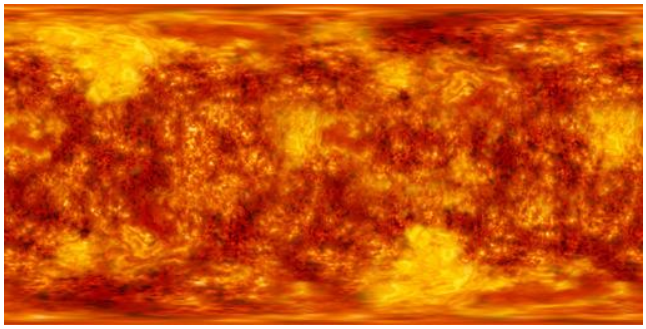
Der erste Parameter hierbei, bestimmt die Größe der Kugel und die anderen zwei bestimmen wie viele Kanten die Kugel besitzen soll. Je mehr Kanten, desto feiner die Kugel.

Der Output sollte ungefähr so aussehen:



5.2 Textur hinzufügen

Um einem Objekt eine Textur hinzufügen zu können muss man als erstes ein Bild haben, welches man für die Textur verwenden möchte. Für die Sonne haben wir beispielsweise dieses hier verwendet:



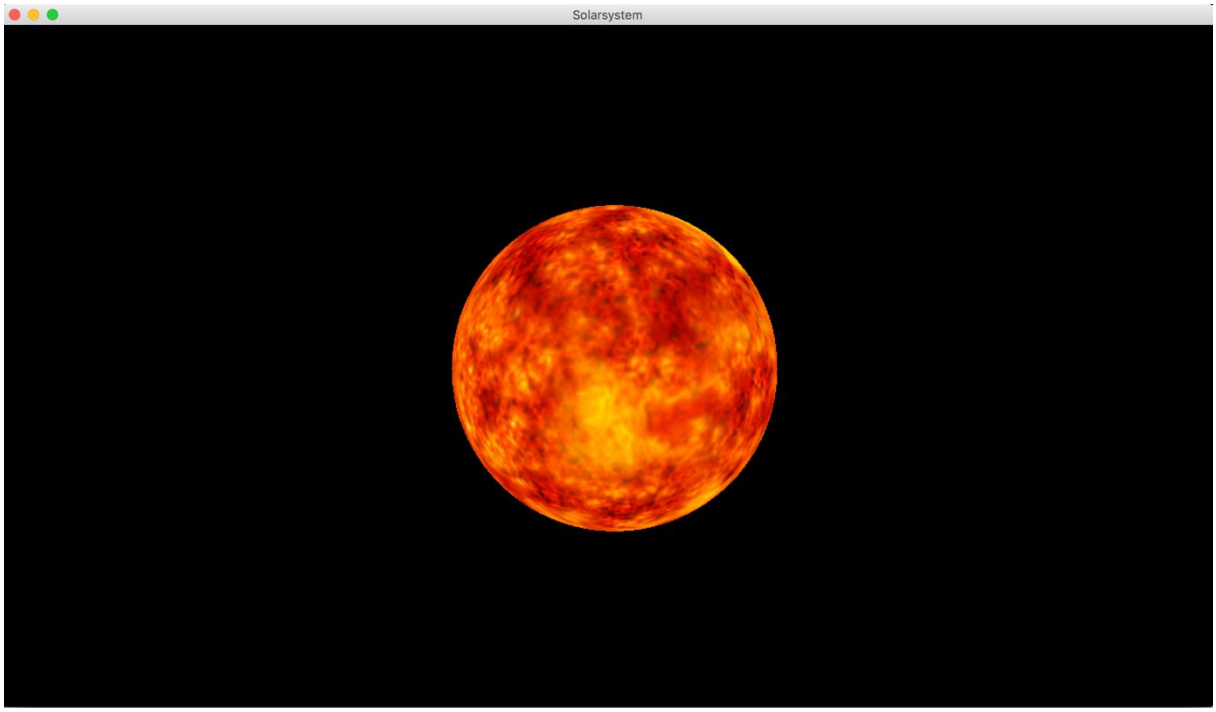
Um das nun an eine Kugel binden zu können, muss man als allererstes den vorherigen Befehl zur Erstellung einer Kugel verwerfen, und durch einen neuen ersetzen:

```
gluSphere(quadric,1,10,10)
```

Dieser nimmt eine Quadrik entgegen, auf welche man eine Textur binden kann. Bevor man diesen Befehl ausführt, sollte man vorher die Textur laden eine Quadrik erstellen und mit folgendem Befehl an das Objekt binden.

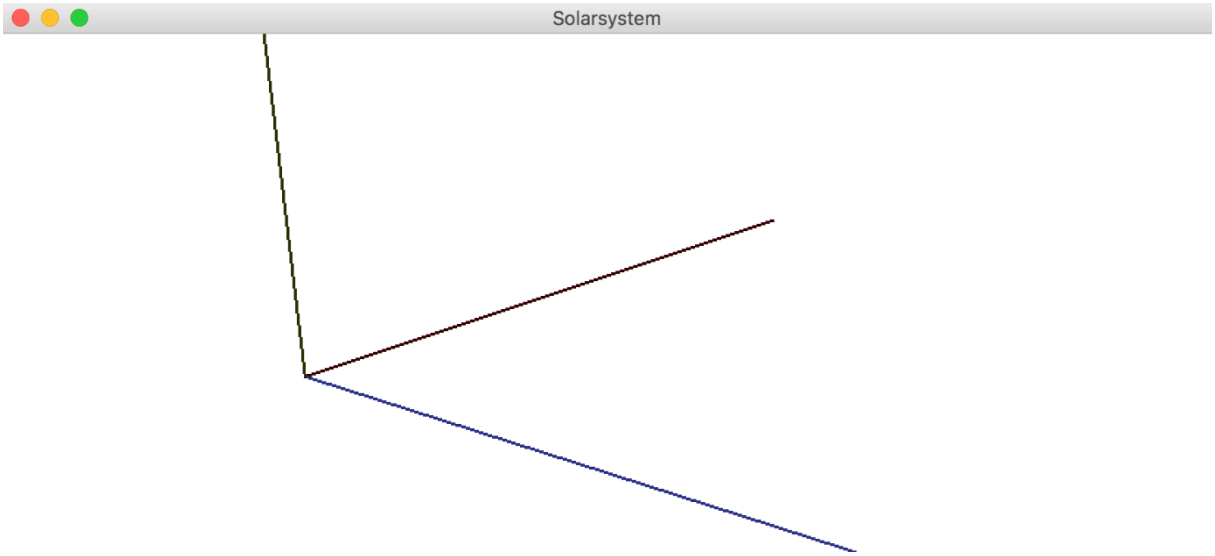
```
glBindTexture(GL_TEXTURE_2D, textur)
```

Der Output sollte ungefähr so aussehen:



5.3 Einzeichnen der Achsen

Damit wir überhaupt wissen wo wir uns gerade befinden beziehungsweise wohin wir die Objekte drehen beziehungsweise verschieben, haben wir uns Achsen eingezeichnet.



Das haben wir mit Linien bewerkstelligt, die wir vom Koordinatenursprung weg, zeichneten.

Dieser Code-Snippet stellt beispielsweise eine Linie vom Startpunkt (0,0,0) bis Endpunkt (0,100,0) dar, was konkret die Y-Achse darstellen soll:

```
glBegin(GL_LINES)
glVertex3f(0, 0, 0) #glVertex3f(x,y,z) <- Startpunkt
glVertex3f(0, 100, 0) #glVertex3f(x,y,z) <- Endpunkt
glEnd()
```

Auf diese Weise wurden auch die anderen Achsen eingezeichnet.

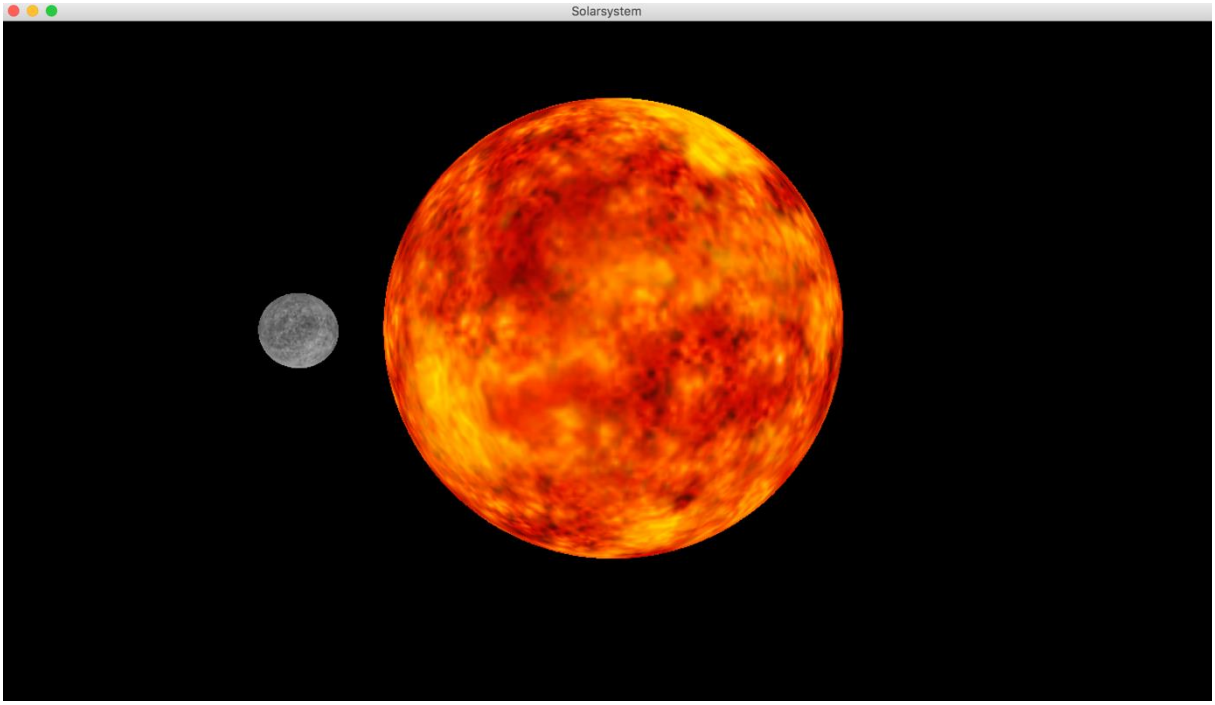
5.4 Objekte verschieben

Bis jetzt haben wir immer im Koordinatenursprung gezeichnet, wollen wir jedoch ein zweites Objekt zeichnen, werden wir feststellen, dass zwei Objekte im Koordinatenursprung vielleicht doch nicht die beste Idee ist. Das Problem kann man lösen, indem man folgenden Befehl verwendet:

```
glTranslatef(x,y,z)
```

Mit Hilfe dieses Befehls kann man ein Objekt auf die gewünschte Stelle im dreidimensionalen Raum verschieben. Hierbei ist zu beachten, dass die Identitätsmatrix neu geladen werden muss, um ausschließlich das aktuelle Objekt zu verschieben.

Somit ist nun folgendes möglich:



5.5 Objekte drehen

Hier wird es nun doch etwas komplizierter. Ein Objekt kann man mit folgendem Befehl rotieren:

```
glRotatef(rotationswinkel,x,y,z)
```

Der erste Parameter gibt an, um wieviel Grad sich das jeweilige Objekt drehen soll, was sich im Endeffekt darauf auswirkt, wie schnell sich das Objekt dreht. Die anderen drei Parameter bestimmen um welche Achse(n), vom Koordinatenursprung ausgehend, sich das Objekt drehen soll.

Gehen wir vom einfachsten Szenario aus: Wir wollen Sonne eine Eigenrotation verpassen. Das ist ziemlich einfach, da die Sonne sich im Koordinatenursprung befindet und wir einfach nur sagen müssen um welche Achse sich die Sonne drehen soll.

```
glRotatef(1,0,1,0)
```

Die Sonne dreht sich mit folgendem Befehl mit jeweils einem Grad um die Y-Achse vom Koordinatenursprung ausgehend, also um sich selbst, da die Y-Achse in diesem Fall die eigene Achse der Sonne ist. Mission erfolgreich ausgeführt.

Die nächste Schwierigkeitsstufe wäre es, einen Planeten um die Sonne kreisen zu lassen. Erstmal entfernen wir den Planeten etwas von der Sonne, damit man ihn sieht und lassen ihn dann um die Sonne rotieren.

```
glTranslatef(5,0,0)
glRotatef(1,0,1,0)
```

Das funktioniert, da wir den Planeten vom Koordinatenursprung wegschieben und dann um den Koordinatenursprung rotieren lassen. Alles gut und schön, bis auf eine Sache: Der Planet hat keine Eigenrotation.

Um einem Objekt, welches um den Koordinatenursprung kreist, eine Eigenrotation zu geben muss man das Objekt als erstes im Koordinatenursprung belassen, dort rotieren lassen, danach verschieben und dann erst um den Koordinatenursprung rotieren lassen.

```
glRotatef(1,0,1,0) #Eigenrotation
glTranslatef(5,0,0) #Translation
glRotatef(1,0,1,0) #Rotation um den Koordinatenursprung
```

Das funktioniert, da immer um die Achsen im Koordinatenursprung gedreht wird. Belässt man das Objekt im Koordinatenursprung so sind die Achsen im Koordinatenursprung gleich den Rotationsachsen des Objektes. Verschiebt man das Objekt jedoch vom Koordinatenursprung weg so ist dies nicht mehr gültig, die Achsen sind nun verschiedene und das Objekt dreht sich nicht um sich selbst.

Bei einem Mond steigert sich die Komplexität somit um noch eine Stufe.

5.6 Kamera

In OpenGL wird die Kamera mit folgendem Befehl eingestellt:

```
gluLookAt(eyeX,eyeY,eyeZ, centerX,centerY,centerZ, upX,upY,upZ)
```

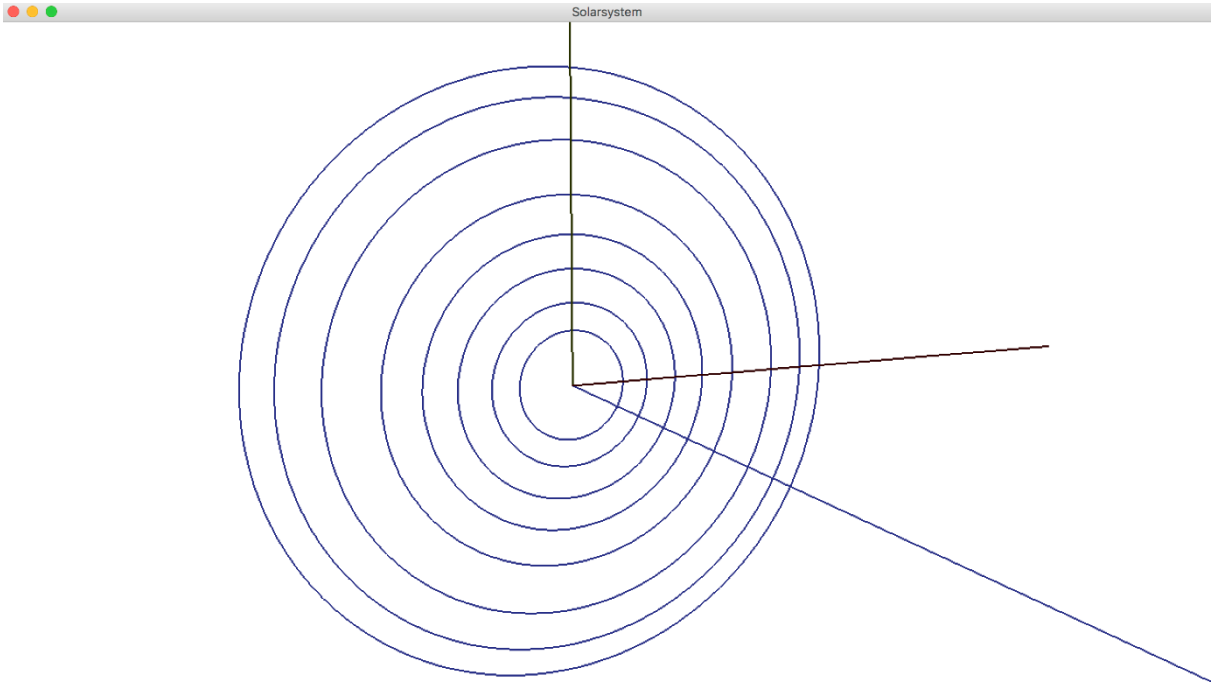
Die ersten drei Parameter bestimmen den Punkt, von welchem aus die Animation betrachtet wird. Die nächsten drei bestimmen den Punkt auf welchen geschaut wird und die letzten drei bestimmen von wo man darauf schaut. (wo oben ist)

Da dieser Befehl in Kombination mit anderen oftmals vorkommt, haben wir uns entschieden eine Kamera-Klasse zu bauen. Somit vermeiden wir redundanten Code und können die Parameter ganz einfach über Attribute verändern und die Kamera somit bewegen.

```
kamera.eyeY += 1
kamera.update()
```

5.7 Bahnen Zeichnen

Zur Übersichtlichkeit haben wir auch noch die Bahnen der Planeten eingezeichnet.



5.8 Licht

Mittels folgendem Befehl kann eine Lichtquelle aktiviert werden:

```
glEnable(GL_LIGHTING)
```

Position und Farbe werden folgendermaßen definiert:

Die ersten drei Parameter sind die Position und der 4 Parameter die Größe.

```
lightZeroPosition = (0, -5, 8, 1)#licht links oben erzeugen

lightZeroColor = (0.8, 1, 0.8, 1) #green tinged
```

Wenn man das Licht wieder deaktivieren möchte ist dies mit diesem Befehl möglich:

```
glDisable(GL_LIGHTING)
```

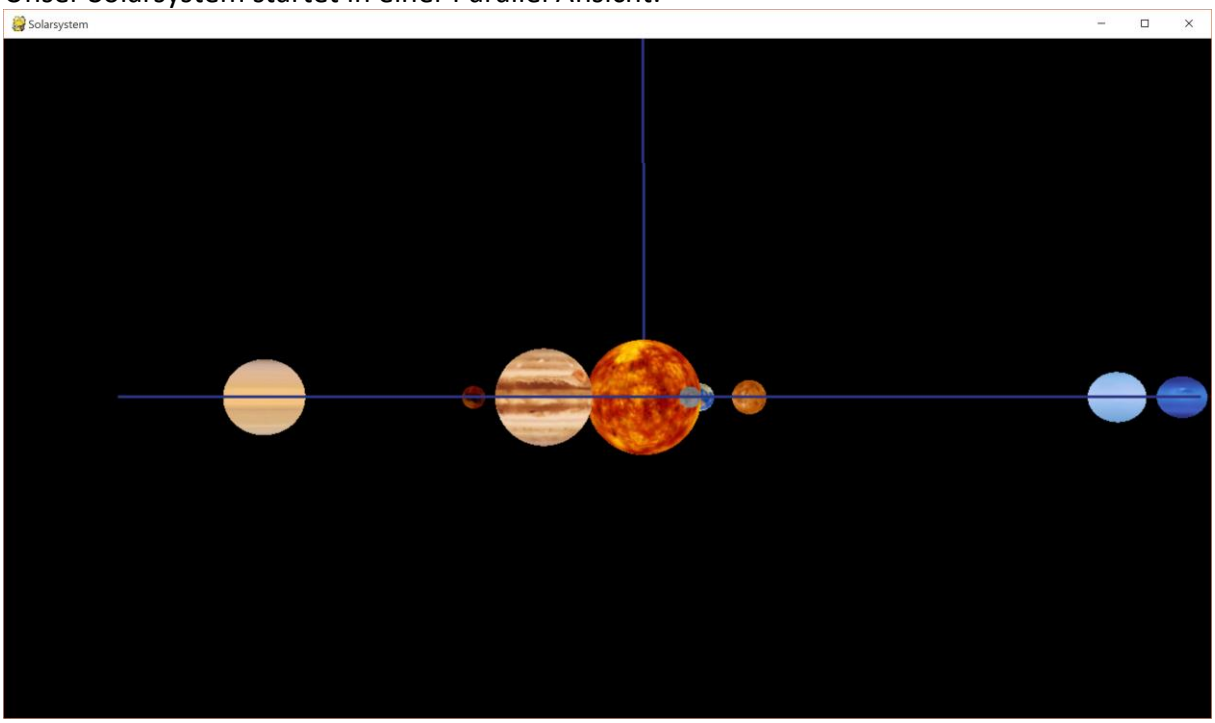
5.9 Steuerung

Funktion	**Tastenkombination**
Licht an/aus	Linksklick
Textur an/aus	Rechtsklick
Zoom	Mausrad oder +/- Tasten
Kamera bewegen	Maus bewegen
Schneller / langsamer	. / ,
Entlang der Achsen bewegen	WASD
Um ein Objekt drehen	Pfeiltasten
Kameraansicht ändern	K - Taste
Animation pausieren/fortsetzen	P - Taste
Fullscreen/Normalansicht	F11
Beenden	ESC - Taste

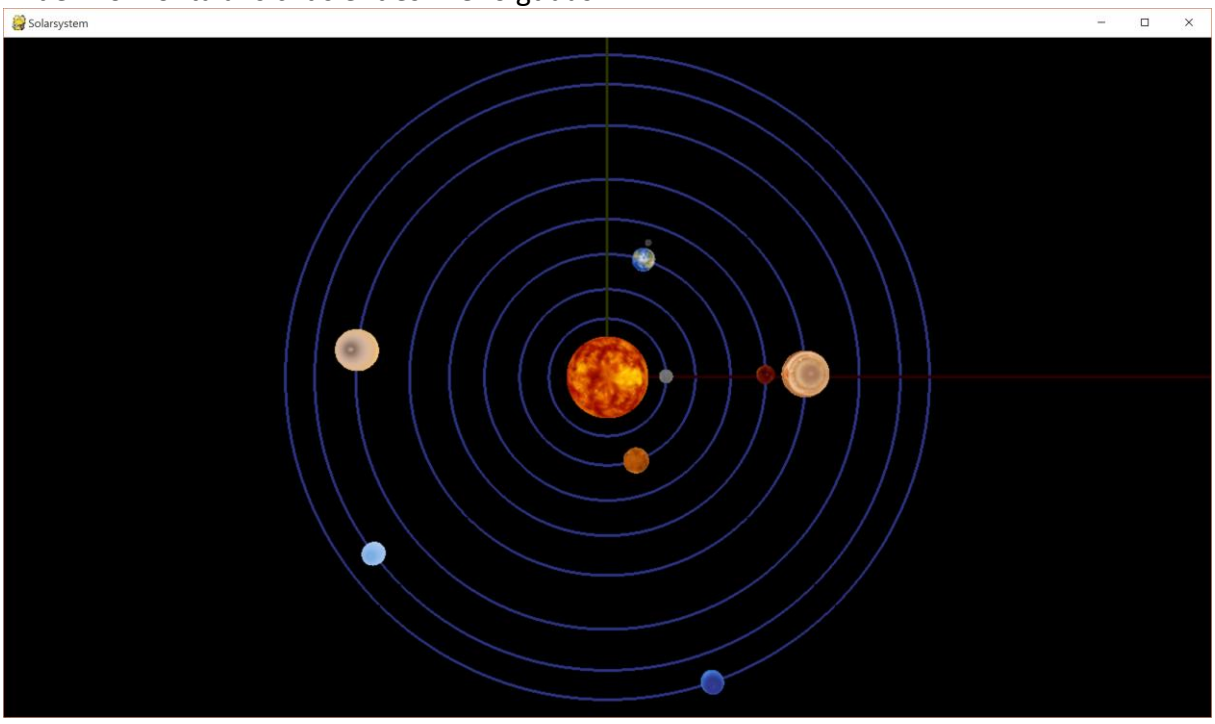
6 Derzeitiger Stand

Siehe: https://github.com/aernhofer-tgm/Solarsystem

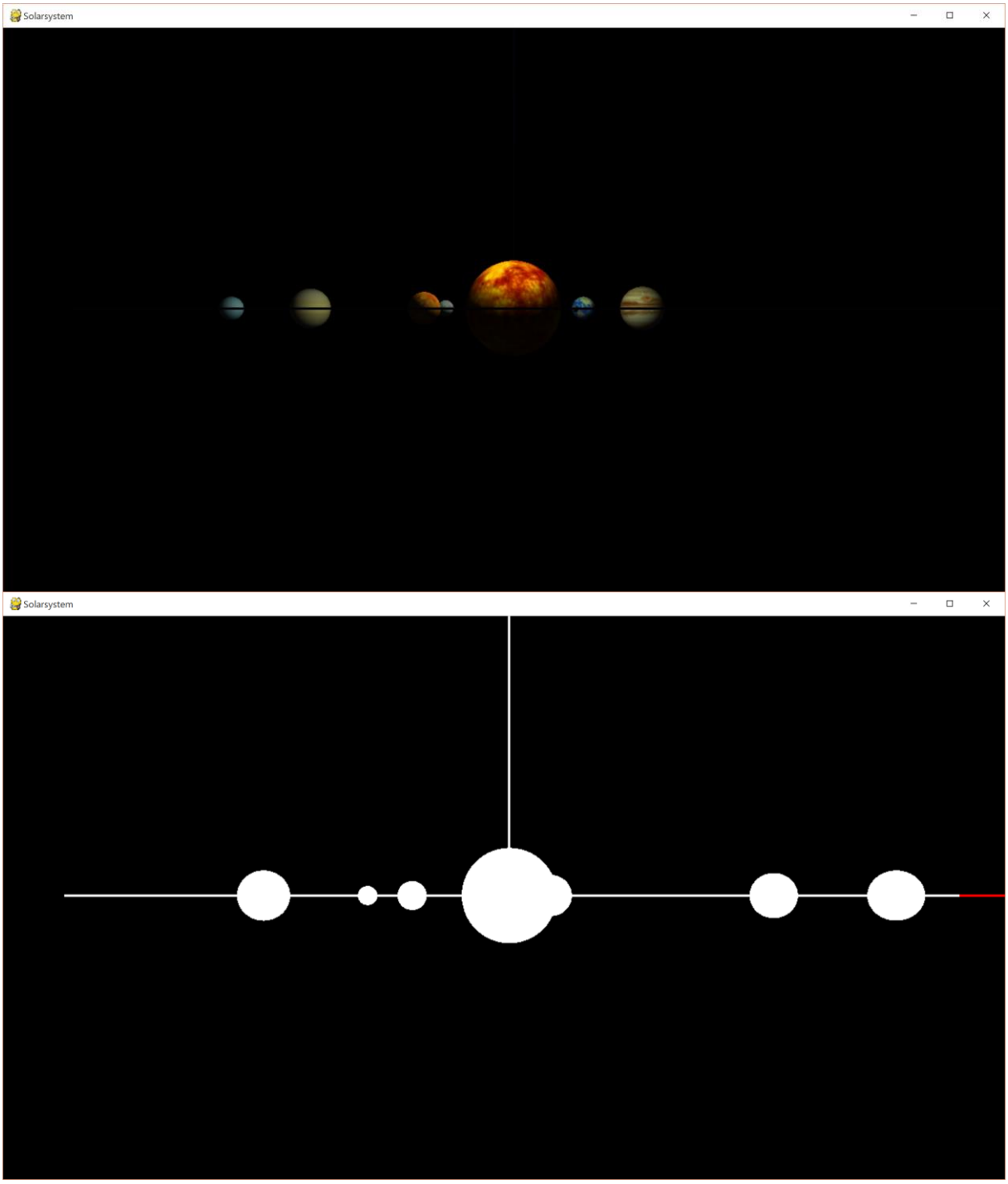
Unser Solarsystem startet in einer Parallel Ansicht:



In der Horizontalansicht sieht es wie folgt aus:



Außerdem ist es möglich die Texturen, sowie die Sonne als Lichtquelle ein bzw. auszuschalten:



Weitere Funktionen finden sich unter 5.Steuerung.

7 Probleme

7.1 Identitätsmatrix neu laden

Unser größtes Problem war es, dass wir anfangs die Identitätsmatrix nicht neu geladen haben und dadurch keine bestimmten Objekte verschieben beziehungsweise rotieren konnten, sondern immer nur das ganze System.

7.2 Texturen

Ein weiteres Problem, welches uns längere Zeit beschäftigt hat war, dass wir die Texturen pausenlos neu geladen haben, anstatt sie zu Beginn einmal zu laden und sie dann zu behalten. Dies führte natürlich zu enormen Rechenaufwänden.

7.3 Maussteuerung

Maus bleibt nicht in der Mitte. Sobald sie das Fenster verlässt wird die Ansicht nicht mehr aktualisiert.

7.4 Fullscreen

Ein weiteres Problem war die Umsetzung der Fullscreen Anzeige auf Apple Geräten.