Visit Our Solar System

SEW

Schwarz Stephan & Gala Mateusz

12/17/2015

Inhaltsverzeichnis

1. Aufgabenstellung

Wir wollen nun unser Wissen aus Medientechnik und SEW nützen um eine etwas kreativere Applikation zu erstellen.

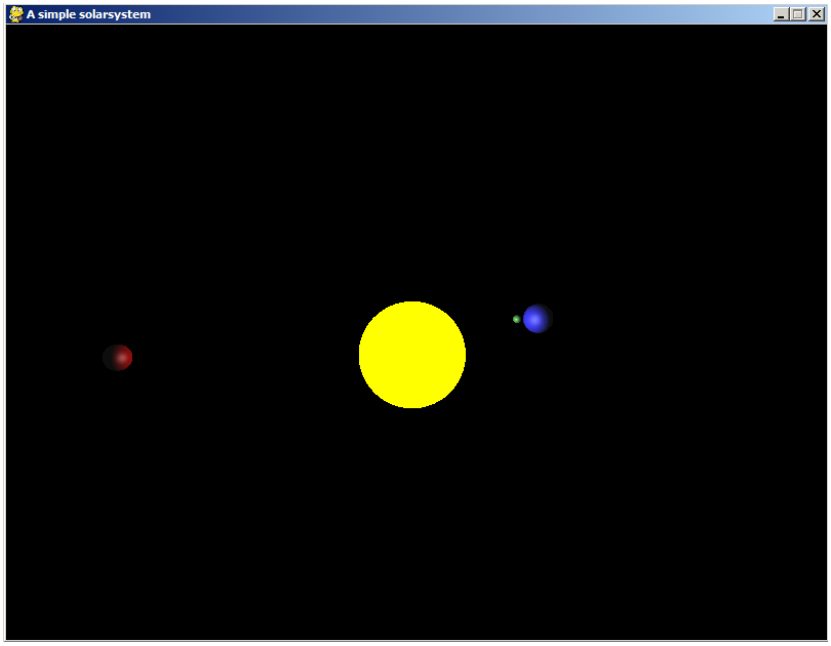
Eine wichtige Library zur Erstellung von Games mit 3D-Grafik ist Pygame. Die 3D-Unterstützung wird mittels PyOpenGL erreicht.

Die Kombination ermöglicht eine einfache und schnelle Entwicklung.

Während pygame sich um Fensteraufbau, Kollisionen und Events kümmert, sind grafische Objekte mittel OpenGL möglich.

Die Aufgabenstellung:

Erstellen Sie eine einfache Animation unseres Sonnensystems:



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:

- 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 wird die Library Pillow benötigt! Die Community unterstützt Sie bei der Verwendung.

Tutorials:

- Pygame: https://www.youtube.com/watch?v=K5F-aGDIYaM

Viel Erfolg!

Lighting:

```
zeros = (0.15, 0.15, 0.15, 0.3)
ones = (1.0, 1.0, 1.0, 0.3)
half = (0.5, 0.5, 0.5, 0.5)
lightposition = (1,1,1,1)

glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, zeros)
glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, half)
glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, 15)
glLightfv(GL_LIGHT0, GL_POSITION, lightposition)
glLightfv(GL_LIGHT0, GL_AMBIENT, zeros)
glLightfv(GL_LIGHT0, GL_DIFFUSE, ones)
glLightfv(GL_LIGHT0, GL_SPECULAR, half)
glEnable(GL_LIGHT0)
glEnable(GL_LIGHTING)
glColorMaterial(GL_FRONT_AND_BACK, GL_DIFFUSE)

glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP)
glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP)
glEnable(GL_TEXTURE_GEN_S)
glEnable(GL_TEXTURE_GEN_T)

glEnable(GL_COLOR_MATERIAL)
glEnable(GL_NORMALIZE)
glShadeModel(GL_SMOOTH)
```

2. Aufwandsabschätzung

2.1. Erwartet

	Schwarz	Gala
Recherche	5 h	8 h
Umsetzung	15 h	15 h
Dokumentation	30 min	10 min
Stunden	20h 30min	23 h 10 min

2.2. Tatsächlich

	Schwarz	Gala
Recherche	20 h	15 h
Umsetzung	10 h	15 h
Dokumentation	2 h	1 h
Stunden	32 h	31 h

3. Recherche

3.1. Pyglet

Begonnen haben wir mit Pyglet, mit welchem zwar die Erstellung eines Fensters und das Zeichnen einiger Objekte einfach fiel, jedoch die Erstellung von 3D-Objekten zu einem Problem wurde.

3.2. Pygame

Nach dem Umstieg auf Pygame musste es erst einmal installiert werden. Dies bereitete zuerst einige Probleme, da die Installation auf der Website nur für Python2.7 funktionierte. Daher musste die Python3.4 Version installiert werden. Danach musst die gedownloadete .whl Datei installiert werden.

```
pip install pygame-1.9.2a0-cp34-none-win32.whl
```

3.2.1. Youtube – Tutorial

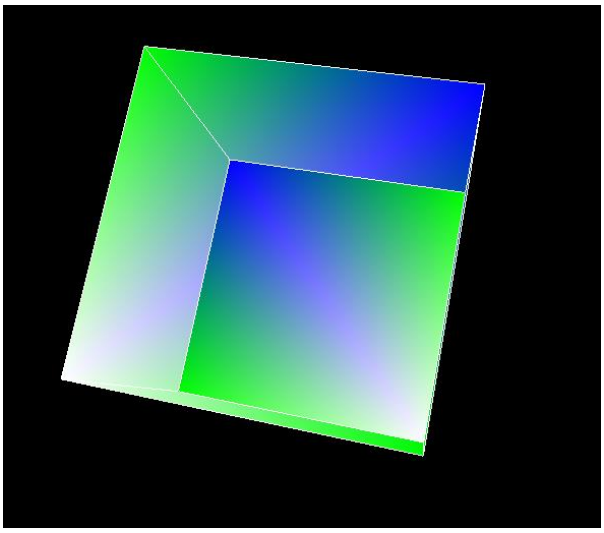
Für das Tutorial auf Youtube musste PyOpenGL installiert werden, welches allerdings GLUT nicht richtig installiert, sollte man es mittels

```
pip install PyOpenGL
```

installieren. Daher musst ebenfalls die Python3.4 Version gedownloadet und installiert werden.

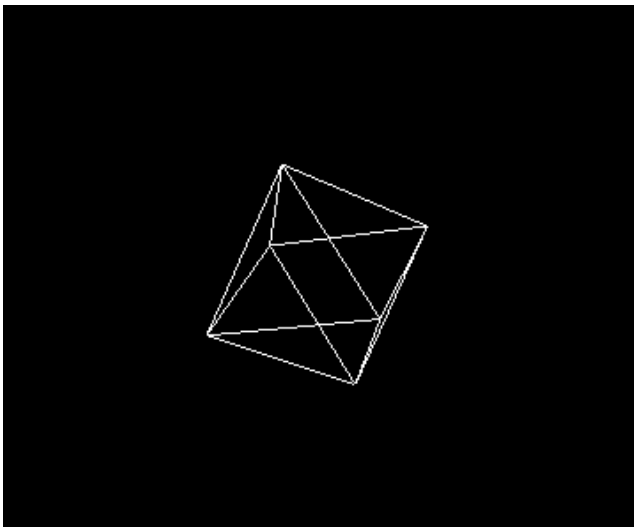
```
pip install PyOpenGL-3.1.1b1-cp34-none-win32.whl
```

Danach konnte dem Tutorial gefolgt und folgender Output erstellt werden:



3.2.2. First Sphere

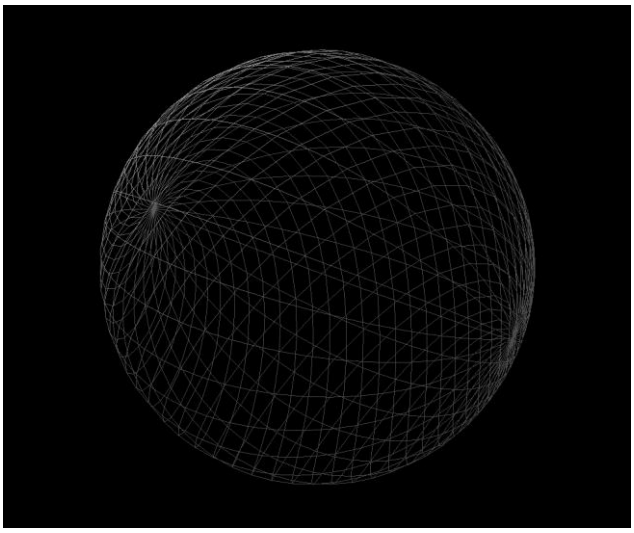
Der erste Versuch eine Sphere zu Zeichnen wurde mittels dieser Methode gestartet. Da jedoch schnell eine bessere Lösung gefunden wurde, wurde diese Methode nur bis zu diesem Output fortgeführt:



Die erste Sphere wurde ohne den GLUT erzeugt. Stattdessen wurde folgende for-Schleife verwendet:

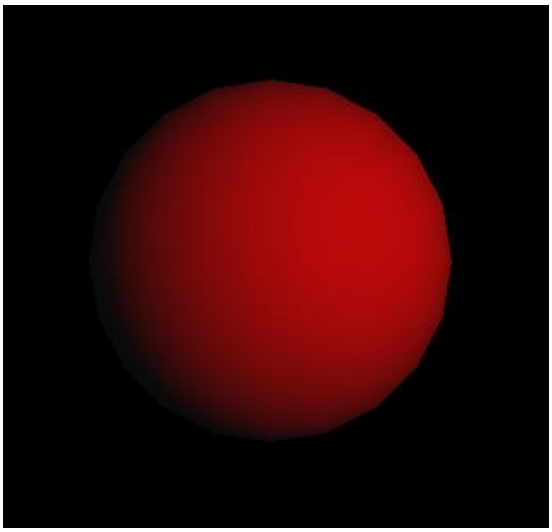
```
for i in range(self.lats + 1):
    lat0 = pi * (-0.5 + float(float(i - 1) / float(self.lats)))
    z0 = sin(lat0) * self.r
    zr0 = cos(lat0)

    lat1 = pi * (-0.5 + float(float(i) / float(self.lats)))
    z1 = sin(lat1) * self.r
    zr1 = cos(lat1)

    glBegin(GL_LINE_STRIP)

    for j in range(self.longs + 1):
        lng = 2 * pi * float(float(j - 1) / float(self.longs))
        x = cos(lng) * self.r
        y = sin(lng) * self.r

        glNormal3f(x * zr0, y * zr0, z0)
        glVertex3f(x * zr0, y * zr0, z0)
        glNormal3f(x * zr1, y * zr1, z1)
        glVertex3f(x * zr1, y * zr1, z1)

    glEnd()
```

Mit dieser ließ sich folgender Output erzeugen:

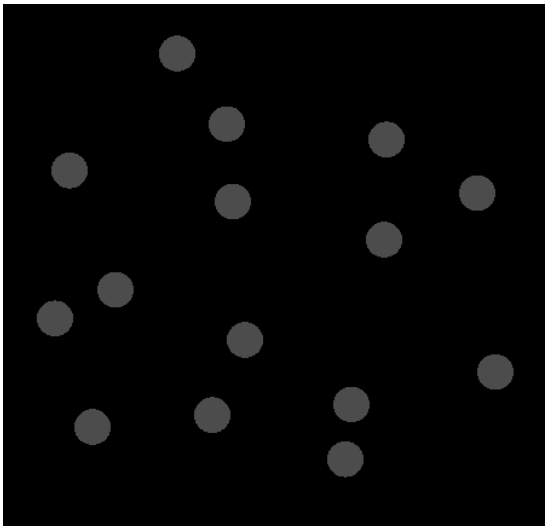


3.3. glutSolidSphere ohne Pygame

Die ersten Sphere mittels glutSolidSphere wurden ohne den Gebrauch von Pygame erstellt. Hierbei wurde in das Fenster mittels glutCreateWindow gezeichnet. Der erste Versuch eine Sphere mit Schattierung zu Zeichnen erzeugte folgenden Output:



Bei dem Versuch mehrere Spheres auf einmal zu Zeichnen fanden wir folgendes Programm, welches Sphers per Mausklick erzeugte:



Diesen fehlte jedoch die Schattierung, weswegen kein 3D-Effekt sichtbar ist.

3.4. glutSolidSphere mit Pygame

Das erzeugen einer glutSolidSphere in einem Pygame Fenster erzeugte die meisten Schwierigkeiten, da die Installation von GLUT Fehlerhaft war. Diese fehlerhafte Installation wurde jedoch nicht sofort gefunden und so sehr viel Zeit damit verschwendet den Source Code zu verändern. Um GLUT nun doch richtig zu installieren musste erst einmal die fehlerhafte Installation entfernt werden.

```
pip uninstall PyOpenGL
```

Danach musste es neu installiert werden, jedoch diesmal die normale Installation, welche zwar zu einer fehlerhaften Installation von GLUT führt, jedoch kann dieses danach manuel richtig installiert werden.

```
pip install PyOpenGL
```

Nun musste nur noch GLUT richtig installiert werden, was mittels dieser Anleitung erfolgte. Um GLUT manuell zu installieren musste das Nvidia Cg toolkit installiert werden. Danach musste das File glut32.dll in

```
python3.4/Lib/site-packages/OpenGL/DLLS
```

verschoben werden.

Danach konnte erstmals eine glutSolidSphere in ein Pygame Fenster gezeichnet werden.

3.5. Texture Mapping

Das Texture Mapping bereitete uns sehr lange Zeit sehr große Probleme. In einem Test Programm in funktionierte es nur auf eine einzelne Sphere mittels dieser Anleitung einwandfrei. Verwendeten wir dann denselben Code im Hauptprogramm funktionierte überhaupt nichts. Nach einiger Zeit schafften wir es, die Textur händisch auf die Sphere zu Mappen. Dies erfolgte mittels TexCoord3f() in der for-

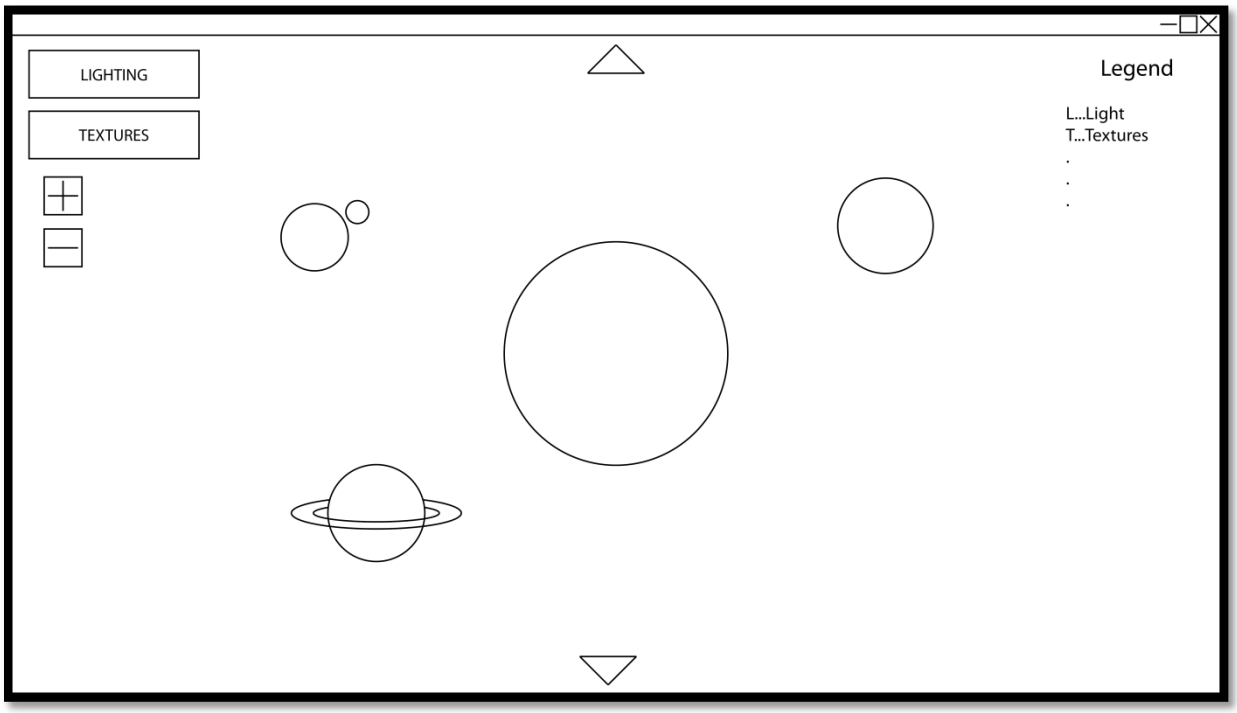
Schleife welche wir bislang zum Erstellen einer Sphere benutzten. Jedoch waren die Textures dann sehr ungenau gemappt, und das wurde bei den größeren Planeten wie der Sonne dann auch sichtbar. Also versuchten wir es mittels einer gluSphere, was ebenfalls nicht funktionierte. Erst als ich durch Zufall mal wieder unsere Klasse für das Licht öffnete sah ich einige Codezeilen, welche ganz offensichtlich zu dem Problem führten:

```
glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP)
glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP)
glEnable(GL_TEXTURE_GEN_S)
glEnable(GL_TEXTURE_GEN_T)
```

Nach dem Entfernen funktionierte alles einwandfrei.

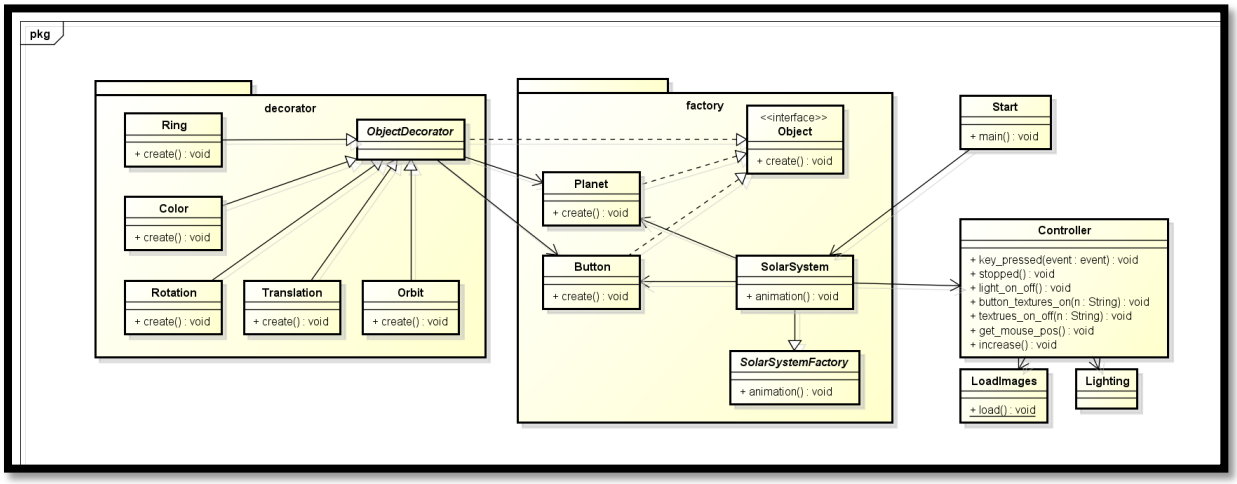
4. Umsetzung

4.1. Grafisches Design



- Button "LIGHTING" schaltet das Licht an und aus
- Button "TEXTURES" schaltet die Texturen an und aus
- "+" und "-" Buttons erhöhen die Rotationsgeschwindigkeit
- Pfeil-buttons lassen die Kamera auf die Positionen 0°, 45° und 90° einstellen
- Die Legende zeigt die Tastaturbefehle zur einfacheren Steuerung an

4.2. Technisches Design



4.3. Implementierung

Auf Grund der Recherche viel die eigentliche Erstellung des Programms nicht mehr allzu schwer. Einige Hindernisse, unerwartete Komplikationen oder einfach nur etwas komplexere Stellen waren:

4.3.1. Licht

Das Licht konnte zwar von der oben genannten Aufgabenstellung übernommen werden, jedoch war die Position noch falsch und musste noch geändert werden. Dies erfolgte mittels:

```
light_position = (1.0, 1.0, 1.0, 1.0)
glLightfv(GL_LIGHT0, GL_POSITION, light_position)
```

4.3.2. Laden der Texturen

Das Laden der Texturen führte eine lange Zeit zu abstürzen des Programmes nach kurzer Laufzeit. Der Fehler hierbei war, dass die entsprechende Methode in der "while True" aufgerufen wurde und somit die Texturen immer und immer wieder geladen wurden.

4.3.3. Ausschalten der Texturen

Bevor sämtliche Buttons existierten wurde das Ausschalten der Texturen beim Drücken der Taste "T" mittels

```
glDisable(GL_TEXTURE_2D)
```

gemacht. Nachdem jedoch die Buttons erstellt waren und sich Texturen auf ihnen befanden, wurden nicht nur die Texturen der Planeten ausgeschaltet, sondern auch die der Buttons. Dies konnte, dank dieser Anleitung, behoben werden, indem anstatt

```
glDisable(GL_TEXTURE_2D)
```

einfach die Standardtextur über die Planeten Textur geladen wird:

```
glBindTexture(GL_TEXTURE_2D, 0)
```

4.3.4. Pfeilbuttons

Mittels der beiden Pfeilbuttons kann man zwischen 3 verschiedenen Kamera Positionen wählen. Eine Seitenansicht von 0°, eine 3D-Ansicht von 45° und eine Vogelperspektive mit 90°.

Umgesetzt wurde das mittels 3 Boolean Variablen und einiger IF-Anweisungen:

```
mitte = True
oben = False
unten = False

if button hoch:
    if kamera zurzeit mittig:
        oben = True
        mitte = False
    if kamera zurzeit unten:
        mitte = True
        unten = False
    if kamera zurzeit oben:
        nichts passiert

if button runter:
    if kamera zurzeit mittig:
        unten = True
        mitte = False
    if kamera zurzeit oben:
        mitte = True
        oben = False
    if kamera zurzeit unten:
        nichts passiert
```

4.3.5. Kamera Steuerung

Die Kamera Steuerung bereitete allgemein sehr viele Probleme und wurde deshalb auch ganz am Ende implementiert. Bevor der Verwendung von gluLookAt wurde die Kamera einfach mittels gluPerspective erstellt. Da dann aber alles zu nah an der Linse war, wurde alles um 30 Einheiten in z-Richtung nach hinten verschoben.

Die Erste Erstellung einer Kamera mittels gluLookAt führte hauptsächlich zu dem Problem, dass man einfach nichts mehr sah. Was natürlich an den falschen Parametern lag, jedoch war es schwer im Internet sinnvolle und verständliche Erklärung für diese 9 Parameter zu finden. Hier eine kleine Auswahl der gefundenen Seiten aus denen dann verschiedene nützliche Informationen zu finden waren:

- www.opengl.org
- www.opengl.org
- www.opengl.org
- www.lighthouse3d.com
- www.toldo.info
- www.felixgers.de
- stackoverflow.com
- gamedev.stackexchange.com

Mittels dieser und noch einiger weiterer Infos konnte dann erstmals eine Kamera erstellt werden, welche die Planeten richtig zeigte. Nun mussten noch 2 weitere Einstellungen erstellt werden, welche die Planeten aus einer Vogelperspektive und einer Seitenansicht zeigten. Dies konnte vor Allem durch experimentieren mit den verschiedenen Parametern erreicht werden. Nun entstand jedoch ein weiteres Problem. Da es in OpenGL nicht möglich ist eine Kamera zu drehen, sondern man immer die gesamte Welt dreht, wurden auch die Buttons, welche ja nur 2D-Objekte sind, mitgedreht. Die Lösung dieses Problems nahm dann wieder sehr viel Zeit in Anspruch, konnte aber Letzen Endes mit einer zweiten Kamera für die 2D-Objekte gelöst werden.