|  |
| --- |
|  |
| Visit Our Solar System |
| **SEW** |
| Schwarz Stephan & Gala Mateusz |
| 12/11/2015 |
|  |
| **C:\Users\Stephan\Desktop\TGM_Logo_solo.png**C:\Users\Stephan\Desktop\it_logo.png |

**Inhaltsverzeichnis**

[1. Aufgabenstellung 1](#_Toc437269620)

[2. Lighting 3](#_Toc437269621)

[3. Aufwandsabschätzung 3](#_Toc437269622)

[4. Recherche 4](#_Toc437269623)

[4.1. Pyglet 4](#_Toc437269624)

[4.2. Pygame 4](#_Toc437269625)

[4.2.1. Youtube – Tutorial 4](#_Toc437269626)

[4.2.2. First Sphere 4](#_Toc437269627)

[4.3. glutSolidSphere ohne Pygame 6](#_Toc437269628)

[4.4. glutSolidSphere mit Pygame 7](#_Toc437269629)

[4.5. Texture Mapping 7](#_Toc437269630)

[5. Umsetzung 8](#_Toc437269631)

[5.1. Grafisches Design 8](#_Toc437269632)

[5.2. Technisches Design 8](#_Toc437269633)

[5.3. Implementierung 8](#_Toc437269634)

[5.3.1. Licht 9](#_Toc437269635)

[5.3.2. Laden der Texturen 9](#_Toc437269636)

[5.3.3. Ausschalten der Texturen 9](#_Toc437269637)

[5.3.4. Pfeilbuttons 9](#_Toc437269638)

[5.3.5. Kamera Steuerung 10](#_Toc437269639)

# 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)

# Aufwandsabschätzung

|  |  |  |
| --- | --- | --- |
|  | Schwarz | Gala |
| Recherche | 20 |  |
| Umsetzung | 5 |  |
| Dokumentation | 1 |  |
| Stunden |  |  |

# Recherche

## 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.

## 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](http://www.lfd.uci.edu/~gohlke/pythonlibs/#pygame) installiert werden. Danach musst die gedownloadete .whl Datei installiert werden.

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

## Youtube – Tutorial

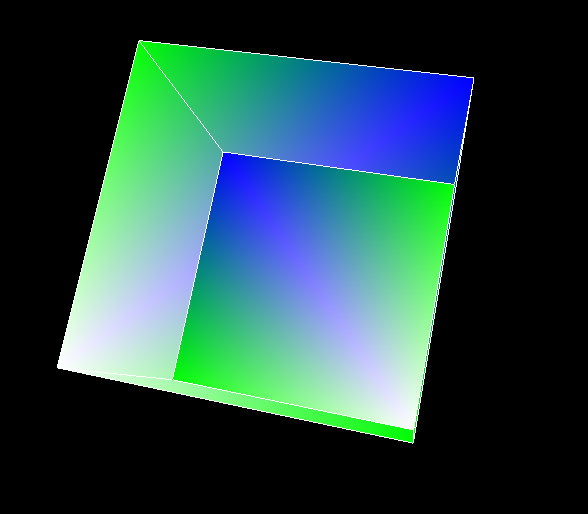
Für das Tutorial auf Youtube musste PyOpenGL installiert werden, welches allerdings GLUT nicht richtig installiert, sollte man es mittels

pip install PyOpenGL

installieren. Daher musst ebenfalls die [Python3.4 Version](http://www.lfd.uci.edu/~gohlke/pythonlibs/#PyOpenGL) gedownloadet und installiert werden.

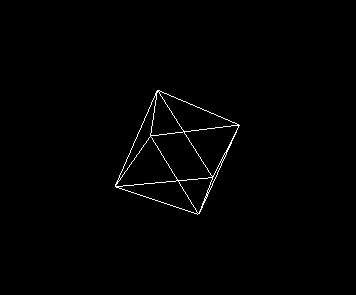
pip install PyOpenGL-3.1.1b1-cp34-none-win32.whl

Danach konnte dem Tutorial gefolgt und folgender Output erstellt werden:



## First Sphere

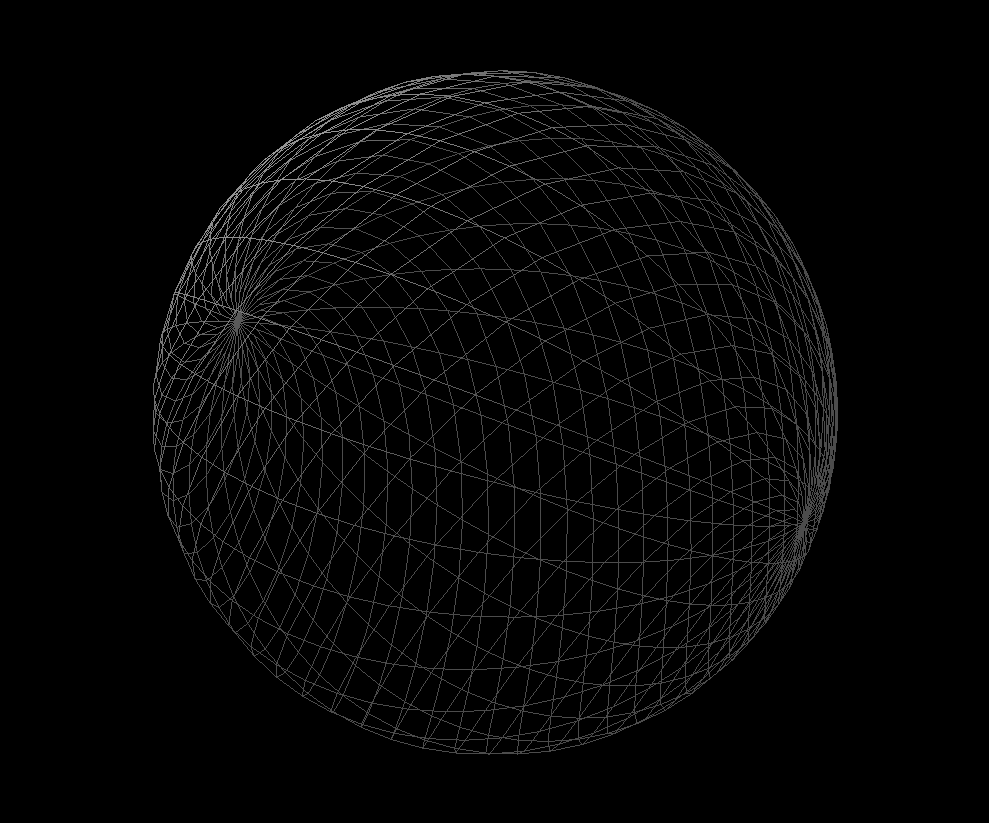
Der erste Versuch eine Sphere zu Zeichnen wurde mittels [dieser Methode](http://stackoverflow.com/questions/7687148/drawing-sphere-in-opengl-without-using-glusphere) gestartet. Da jedoch schnell eine bessere Lösung gefunden wurde, wurde diese Methode nur bis zu diesem Output fortgeführt:



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

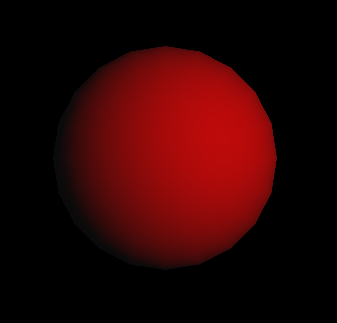
**for** i **in** range(self.lats + 1):  
 lat0 = pi \* (-0.5 + float(float(i - 1) / float(self.lats)))  
 z0 = sin(lat0) \* self.r  
 zr0 = cos(lat0)  
  
 lat1 = pi \* (-0.5 + float(float(i) / float(self.lats)))  
 z1 = sin(lat1) \* self.r  
 zr1 = cos(lat1)  
  
 glBegin(GL\_LINE\_STRIP)  
  
 **for** 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:

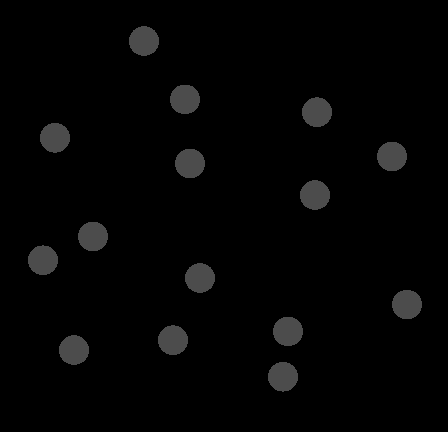


## glutSolidSphere ohne Pygame

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.

## 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](http://web.eecs.umich.edu/~sugih/courses/eecs487/glut-howto/#win) erfolgte. Um GLUT manuell zu installieren musste das [Nvidia Cg toolkit](https://developer.nvidia.com/cg-toolkit-download) 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.

## Texture Mapping

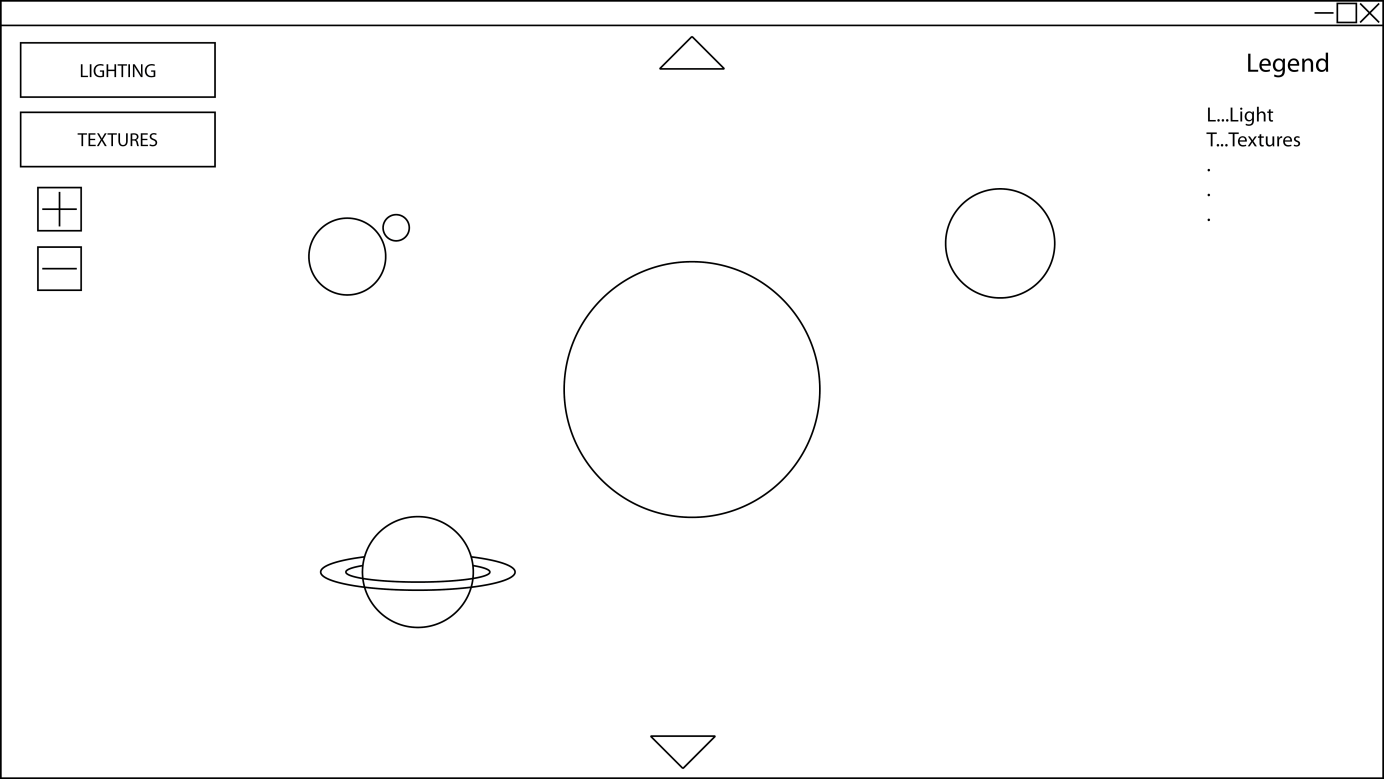
Das Texture Mapping bereitete uns sehr lange Zeit sehr große Probleme. In einem Test Programm in funktionierte es nur auf eine einzelne Sphere mittels [dieser Anleitung](http://pyopengl.sourceforge.net/context/tutorials/nehe6.html) einwandfrei. Verwendeten wir dann denselben Code im Hauptprogramm funktionierte 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.

# Umsetzung

## 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

## Technisches Design

## 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:

## 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)

## 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.

## 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 behoben werden, indem anstatt

glDisable(GL\_TEXTURE\_2D)

einfach die Standardtextur über die Planeten Textur geladen wird:

glBindTexture(GL\_TEXTURE\_2D, 0)

## 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

## Kamera Steuerung