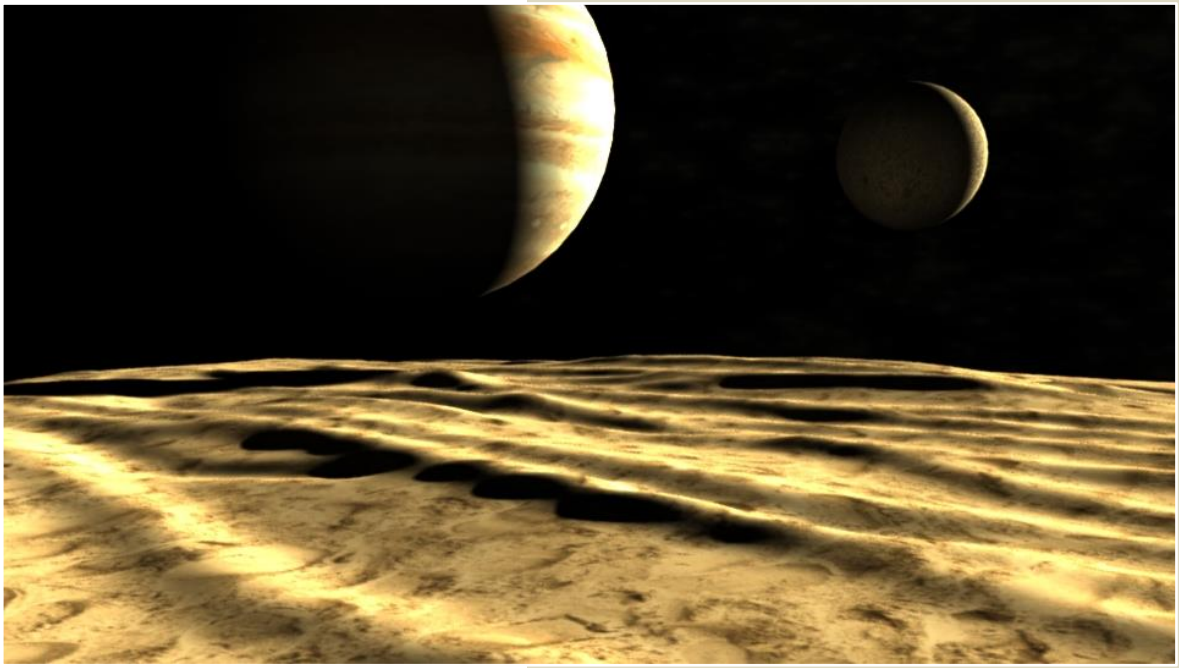
2015

Visit our solar system



Herczeg | Krickl

5BHITM

23.02.2015

Sonnensystem

Inhaltsverzeichnis

Aufgabenstellung 2
Zeitaufzeichnung 4
Design 6
Libraries und Versionen 6
UML 8
GUI 9
Problemstellungen 11
GUI 11
Texturierung 11
Lighting 12
Eventhandling 13
Animation 13
Neue Gestirne hinzufügen 14
Relationen der Gestirne 14
Tests 15
Ausführen und Steuerung 16
Quellen 17
Im Text verwendet 17
Nachschlagen 17

Aufgabenstellung

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

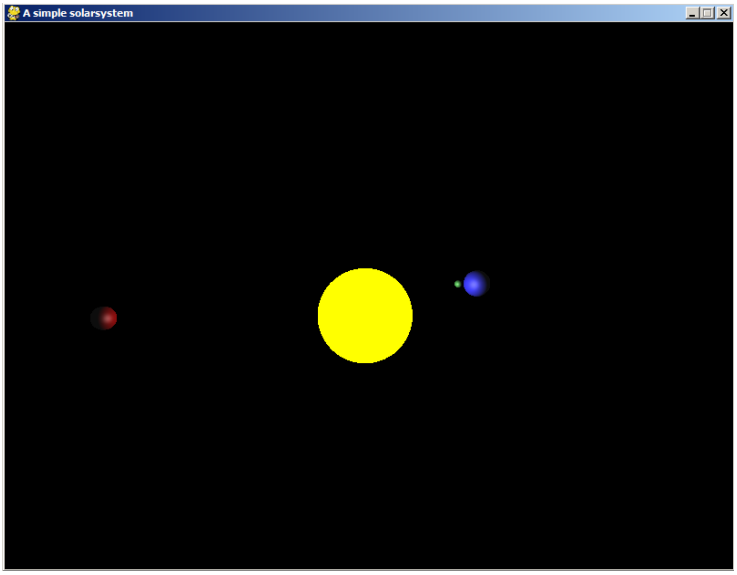
Eine wichtige Library zur Erstellung von Games mit 3D-Grafik ist Pygame. Die 3D-Unterstützung wird mittels PyOpenGL erreicht.

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

```
        def setupLighting():
                """ Initializing Lighting and Light0

                :return:
                """
                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)

                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_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)
```

Zeitaufzeichnung

Anforderung	Priorität	Verantwortlicher	Zeit (G) [min]	Zeit (T) [min]	Status (D,I,Te,Do,F, A)
Nicht funktional					
Recherche der Libraries und OpenGL	Mittel	K, H	120	130	F
Installation der Libraries	Hoch	K, H	30	40	F
Installation von OpenGL	Mittel	K, H	10	5	F
Texturen suchen und aufbereiten	Mittel	K	80	40	I
Planung					
GUI planen	Mittel	K	120	130	F
GUI erstellen	Hoch	K	140	150	F
3D-Raum erstellen. Dh.: Die GUI auf eine 3D-Ansicht vorbereiten	Hoch	K	40	60	F
Erstellen der 3D-Objekte	Niedrig	K	20	30	F
Klassendiagramme bzw. Software planen und erstellen	Hoch	H	120	100	F
Implementierung					
Implementierung eines Gestirns Interfaces	Hoch	H	30	20	F
Implementierung der Translation eines Planeten	Hoch	H	20	40	F
Implementierung der Rotation eines Planeten	Hoch	H	10	20	F
Implementierung einer Mond-Klasse	Hoch	H	30	50	F
Implementierung der Translation des Mondes	Hoch	H	10	20	F

Implementierung der Rotation des Monds	Hoch	H	30	35	F
Fixstern als Punktlicht implementieren	Mittel	K	60	70	F
Schattenberechnung implementieren	Niedrig	K, H	60	80	F
Kamera implementieren	Hoch	H	20	30	F
Steuerung und Animation					
Benutzersteuerung implementieren	Hoch	K	20	35	F
Kameraposition anpassbar machen	Hoch	K	20	30	F
Animationen implementieren	Hoch	K, H	10	40	F
Animation stoppbar/startbar machen	Hoch	H	10	30	F
Geschwindigkeit der Animation anpassbar machen	Hoch	H	10	20	F
Licht ein/ausschaltbar machen	Mittel	K	20	35	F
Test und Abnahme					
Prototyp fix-fertig lauffähig machen	Hoch	K, H	15	15	F
Testcases planen	Mittel	K, H	60	30	F
Testcases schreiben	Hoch	H	35	20	F
UACs planen	Mittel	K, H	60	30	F
UACs durchführen	Mittel	K	30	40	F
Beta-Tests durchführen	Niedrig	K, H	25	20	F
DAU das Programm ausführen lassen	Niedrig	K, H	15	10	F
Abnahme	Hoch	K, H	30	-	F
Summe Herczeg [min]			545	620	
Summe Krickl [min]			545	620	

G … Geschätzt
T … Tatsächlich
D … Design
I … Implementierung
Te … Test
Do … Dokumentation
F … Fertig
A … Abgenommen

H … Herczeg
K … Krickl

Design

Libraries und Versionen

Python 3.4.

Pygame 1.9.

Wird noch nicht verwendet.

Pillow 2.7.1

Library zum Einbinden von Texturen

```
    @staticmethod
    def LoadTexture(pic):
        # Textur
        ix = image.size[0]
        iy = image.size[1]
        image = image.tostring("raw", "RGBX", 0, -1)

        # Textur erstellen
        textures = glGenTextures(1)
        glBindTexture(GL_TEXTURE_2D, textures)  # 2d texture (x and y size)

        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR)
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_LINEAR_MIPMAP_NEAREST)
        gluBuild2DMipmaps(GL_TEXTURE_2D, 3, ix, iy, GL_RGBA,
GL_UNSIGNED_BYTE, image)

        return textures
```

PyOpenGL 3.x.

Library um die GPU des PCs zu verwenden. Leichtes erstellen und einbinden von Standardformen möglich.

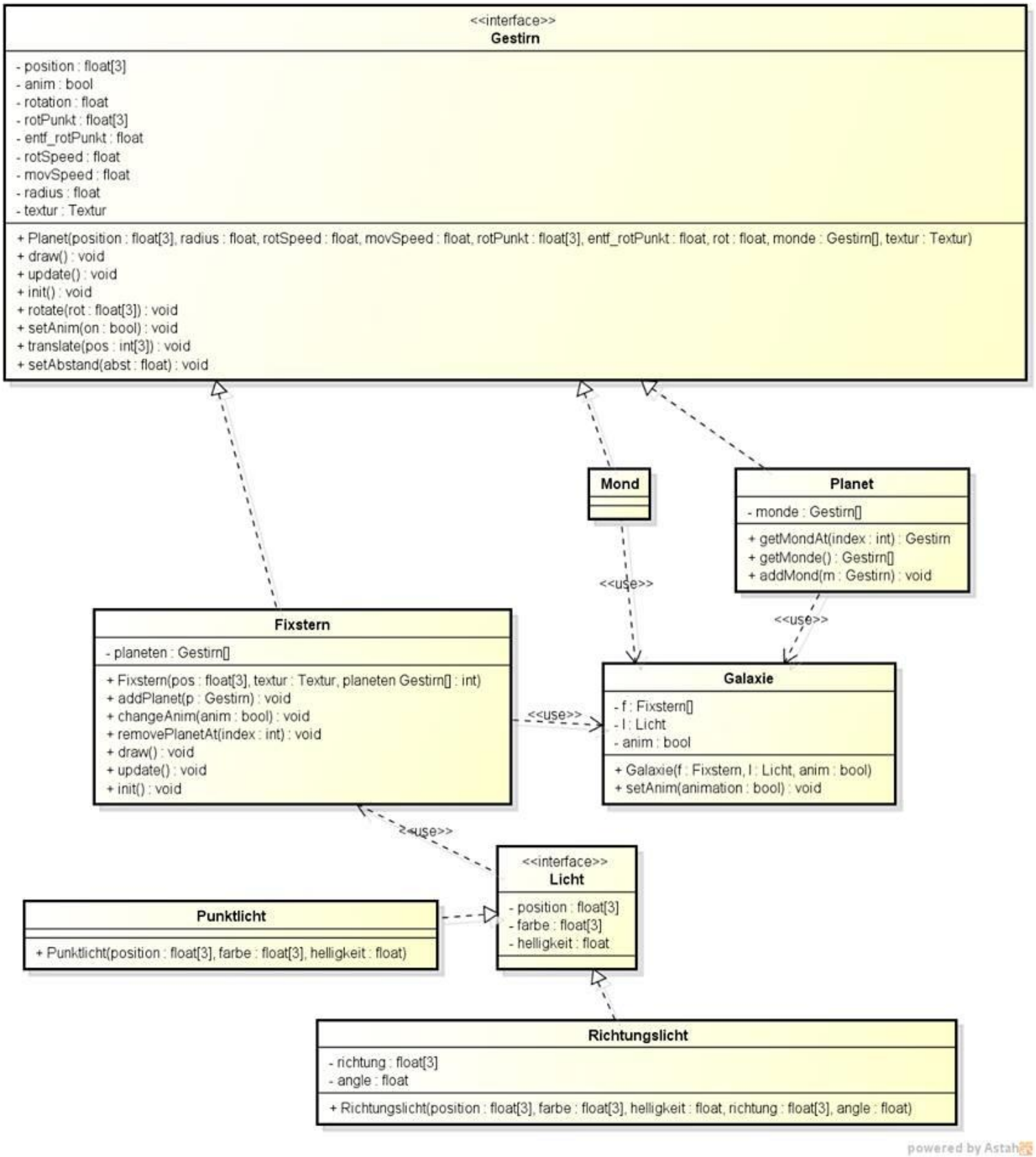
```
def InitGL(self, Width, Height):
        glEnable(GL_TEXTURE_2D)
        glClearColor(0.0, 0.0, 0.0, 0.0)    # Hintergrundfarbe
        glClearDepth(1.0)                   # Loeschen des Depth Buffers
        glDepthFunc(GL_LESS)                # The Type Of Depth Test To Do
```

```
        glEnable(GL_DEPTH_TEST)              # Enables Depth Testing
        glShadeModel(GL_SMOOTH)              # Enables Smooth Color Shading
        glMatrixMode(GL_PROJECTION)          # Reset The Projection Matrix
        glLoadIdentity()

        # camera
        gluPerspective(45.0, float(Width) / float(Height), 0.1, 100.0)
        glMatrixMode(GL_MODELVIEW)

    """
    Wenn die groesse vom Fenster geaendert wird
    """
    def ReSizeGLScene(self, Width, Height):
        glViewport(0, 0, Width, Height
        glMatrixMode(GL_PROJECTION)
        glLoadIdentity()
        # Perspektive
        gluPerspective(50.0, float(Width) / float(Height), 0.1, 100.0)
        glMatrixMode(GL_MODELVIEW)


    """
    szene zeichenn
    """
    def DrawGLScene(self):

        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
        # Planet P1
        self.rot_pl2 = self.gestirn.rotation(self.rot_pl2, 0, 0.04, 0)
        self.gestirn.DrawGLScene_P(0.5, self.rot_pl2, self.light,0.8,0,-10)

        glutSwapBuffers()  # zeichnen

    def main(sc):
        #solarsystem
        glutInit(sys.argv)

        glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH)
        glutInitWindowSize(1000, 600)
        glutInitWindowPosition(50, 50)
        glutCreateWindow(b'Solarsystem')
        glutDisplayFunc(sc.DrawGLScene)
        glutIdleFunc(sc.DrawGLScene)
        glutReshapeFunc(sc.ReSizeGLScene)
        sc.InitGL(640, 480)
        glutMainLoop()


s = universe()
main(s)
```

Freeglut 2.8.1 32bit

Wird alternativ zum OpenGL Utility Toolkit in Kombination mit PyOpenGL verwendet. Beispielcode von PyOpenGL lässt sich nur mit Freeglut verwenden.

UML



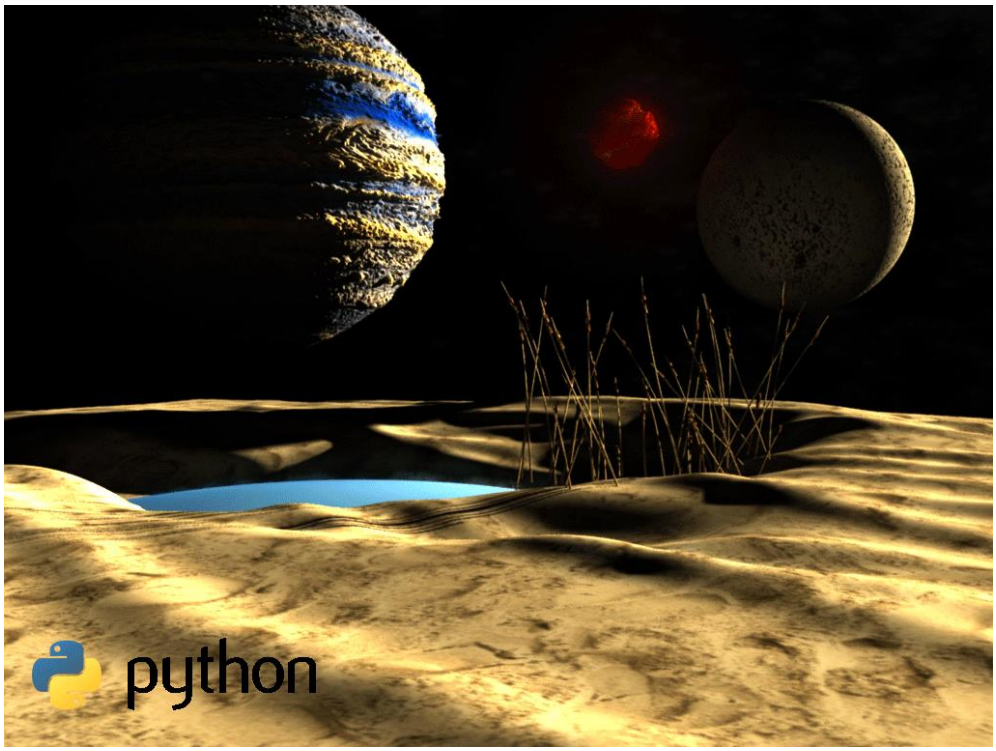
Eine Galaxie kann einen oder mehrere Fixsterne besitzen, um den Fixstern können sich Planeten drehen und um die Planeten können sich Monde bewegen. Da man ein Gestirn (Himmelkörper) verallgemeinern kann, da Monde, Planeten und Fixsterne sehr ähnliche Eigenschaften haben, gibt es

ein Interface ‚Gestirn', das diese Abstaktion umsetzt. Alle Himmelskörper implementieren dieses Interface und fügen eigene spezifische Methoden hinzu. Da das Licht vom Fixstern ausgeht, verwendet dieser auch das ‚Licht'-Interface, dass Punkt- und Richtungslicht implementiert hat.

GUI

Splashscreen [1]

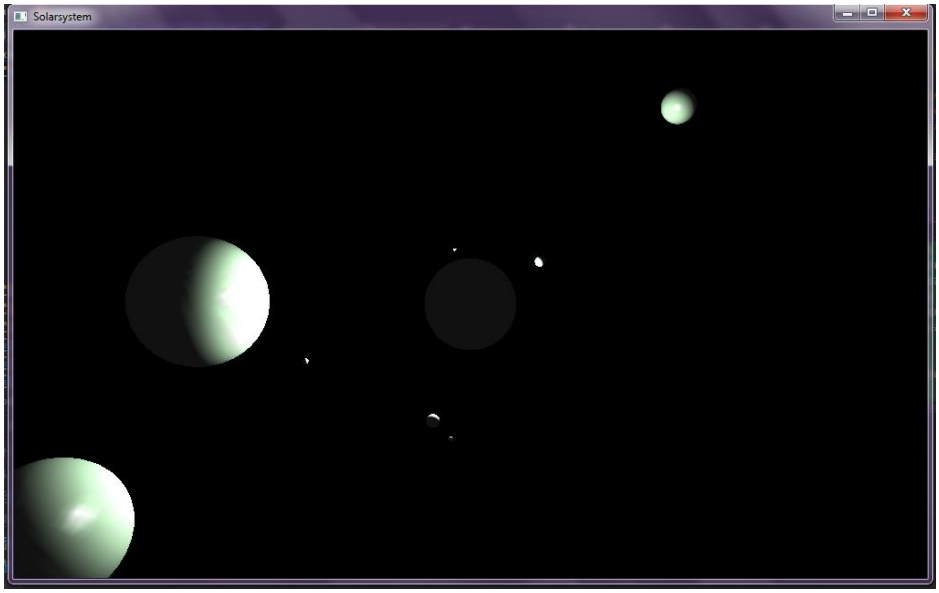
Wird ein paar Sekunden lang angezeigt, bevor die Animation startet.



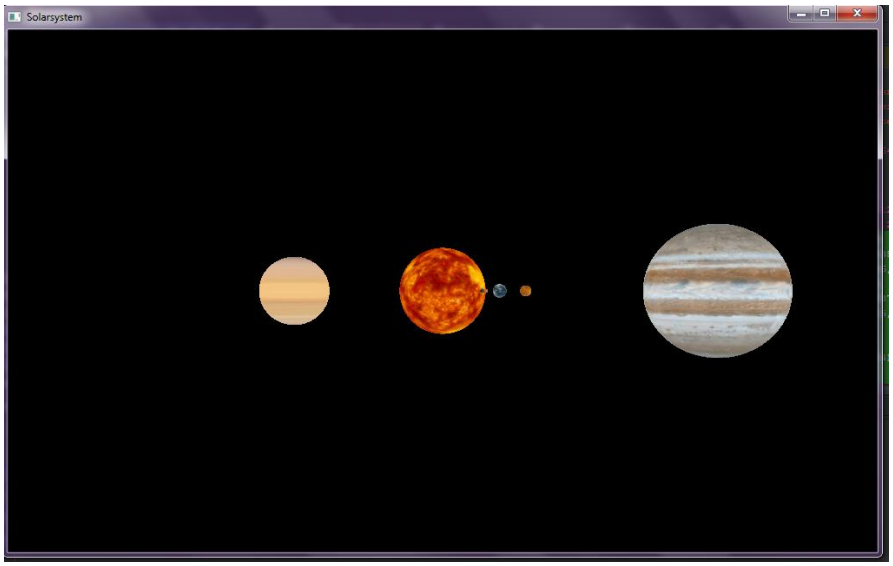
Perspektive von vorne. Taste ‚c' zum Perspektive ändern.

Ansicht von oben. Taste ‚c' um die Perspektive zu ändern.

Taste ‚t' um die Texturen auszuschalten.



Über ‚l' lässt sich die Belichtung an und ausschalten.



Taste ‚p' um die Animation zu pausieren und wieder anzuschalten. Über die Tasten w/s verändert sich die Geschwindigkeit der Animation und über die Tasten +/- kann man den Zoom verändern.

Problemstellungen

Installation von Pillow 2.7.1 auf Python 3.3 war nicht möglich, deshalb Umstieg auf Python 3.4, wo es problemlos möglich war.

GUI

Die Animation des Universums wird über PyOpenGL und Freeglut angezeigt. Der Splashscreen aber wurde mithilfe von Tkinter realisiert. Mithilfe von Tkinter wird ein Bild (.gif) großflächig am Bildschirm angezeigt [1].

Nach diesem Verfahren öffnet Tkinter allerdings ein Standardfenster, das man durch folgenden Methodenaufruf schließen kann, da wir es sonst nicht gebraucht haben.

```
tkRoot.withdraw()
```

Der Aufruf dieser Methode geschieht nach dem Aufruf der enter und exit Methode der Splashscreen Klasse. Danach werden alle Komponenten von PyOpenGL gestartet die das Universum anzeigen.

Texturierung

Alle Texturen wurden auf eine Breite von maximal 700px skaliert, da eine höhere Auflösung erstens die Animation sichtlich verlangsamt und zweitens man mit freiem Auge auch keinen Qualitätsunterschied erkennt. Für die Gestirne wurde das Bildformat ‚jpg' verwendet weil es PyOpenGL gut verarbeiten kann. Allerdings wurde für den Splashscreen das Format ‚gif' gewählt, da es schneller geladen und somit angezeigt werden kann, aber auch weil der Code des Splashscreens nicht in Verbindung mit PyOpenGL steht sondern im Tkinter und sich deshalb keine besonders gute und schnelle Bildverarbeitung angeboten hat.

Prinzipiell wurden die Texturen der Gestirne über eine eigene Texturen Klasse geladen. Die Textur wird über den Dateipfad angegeben.

```
class Texturen():
```

```
    @staticmethod
    def LoadTexture(pic):
        image = open(pic)

        # Textur
        ix = image.size[0]
        iy = image.size[1]
        image = image.tostring("raw", "RGBX", 0, -1)

        # Textur erstellen
        textures = glGenTextures(1) #textur ID
        glBindTexture(GL_TEXTURE_2D, textures)  # 2d texture (x and y size)

        # Planet P1
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR)
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_LINEAR_MIPMAP_NEAREST)
        gluBuild2DMipmaps(GL_TEXTURE_2D, 3, ix, iy, GL_RGBA,
GL_UNSIGNED_BYTE, image)

        return textures
```

Beim Erstellen eines Gestirns wird die Textur als Parameter angegeben und verwendet.

Lighting

Das Hauptlicht (Punktlicht) geht vom Fixstern aus/von den Fixsternen. Aufgrund dessen werden die Schatten dynamisch berechnet und somit werden die Planeten umso näher sie an der Kamera sind, dunkler. Das Licht lässt sich mit dem Tastaturbefehl ‚l' ein- und ausschalten.

In PyOpenGL wurde das folgendermaßen gelöst:

```
def init(self):
        glLightfv(GL_LIGHT0, GL_AMBIENT, (0.25, 0.25, 0.25, 1.0))# AmbientL
        glLightfv(GL_LIGHT0, GL_DIFFUSE, (1.0, 1.0, 1.0, 1.0)) # Diffuse L
        glLightfv(GL_LIGHT0, GL_POSITION, (0.0, 0.0, 2.0, 1.0)) # PositionL
        glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, zeros)
        glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, half)
        glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, 100)
        glEnable(GL_LIGHT0)
        glEnable(GL_LIGHTING)
        glColorMaterial(GL_FRONT_AND_BACK, GL_DIFFUSE)
        glEnable(GL_COLOR_MATERIAL)
        glEnable(GL_NORMALIZE)
        glShadeModel(GL_SMOOTH)
```

Durch Objektattribut in der Klasse Light wurde es ermöglicht das Licht ein- und auszuschalten in der draw Methode:

```
def draw(self):
        if self.enabled:
            glShadeModel(GL_SMOOTH)
            glEnable(GL_CULL_FACE)
            glEnable(GL_DEPTH_TEST)
            glEnable(GL_LIGHTING)
            glDepthFunc(GL_LESS)
            lightZeroPosition = self.position
```

```
            lightZeroColor = self.color
            glLightfv(GL_LIGHT0, GL_POSITION, lightZeroPosition)
            glLightfv(GL_LIGHT0, GL_DIFFUSE, lightZeroColor)
            glLightf(GL_LIGHT0, GL_CONSTANT_ATTENUATION, 0.01)
            glLightf(GL_LIGHT0, GL_LINEAR_ATTENUATION, 0.01)
            glEnable(GL_LIGHTING)
            glEnable(GL_LIGHT0)
        else:
            glDisable(GL_LIGHTING)
            glDisable(GL_LIGHT0)
```

Eventhandling

Das Eventhandling wurde von der Glut Library über die keyPressed Methode übernommen. [2]

```
def keyPressed(self, *args):
        if args[0] == b'c':
            self.changePos()

        if args[0] == b't':
            self.changeTextures()

        if args[0] == b'l':
            self.disable_light()
```

Hier wird der Tastaturbefehl entgegen genommen und ausgwertet. Den Tastatur-Input kann man mit b (für byte, gibt den byte Code des Nachfolgenden Strings zurück) ‚t' angegeben werden (hier wird auf den Buchstaben t geprüft). Wenn der Vergleich erfolgreich war wird die entsprechende Methode aufgerufen.

Die Steuerung über die Tastatur musste lediglich aktiviert werden über folgenden Methodenaufruf in der initGL Methode:

```
glutKeyboardFunc(self.keyPressed)
```

Hier gibt man an auf welche Methode bei einer gedrückten Taste referenziert werden soll. Da hier nur ein ‚call' angegeben wird, sind keine Klammern notwendig.

Animation

Ein Gestirn jedes Universums hat nicht nur eine Rotation. Die Gestirne drehen sich um den Fixstern, die Monde drehen sich um den jeweiligen Planeten und alle Gestirne drehen sich auch um sich selbst.

Prinzipiell wird beim Erstellen eines Gestrins übergeben ob sich dieses Bewegen soll, den Rest berechnet die jeweilige Gestrin-Klasse automatisch mit den Methoden setAnimation, animateAllChildrenSlower und animateAllChildrenFaster. Beispiel dafür

```
def animateAllChildrenFaster(self, factor, factorMov):
        self.rotSpeed += factor
        self.movSpeed += factorMov

        for i in range(0, len(self.monde)):
            self.monde[i].animateFaster(factor, factorMov)
```

Als kleines Feature wurden Kometen eingebaut, die durch die Galaxie hindurch fliegen.

Neue Gestirne hinzufügen

Der Code ist daraus ausgelegt schnell und einfach neue Gestirne hinzuzufügen (Planeten, Monde).

Um eine neue Sonne anzulegen müssen folgende Befehle ausgeführt werden.

```
# Parameter(Fixstern): (position, rotSpeed, textur, planeten, anim, licht,
radius, divisions)
self.sonne = Fixstern([0, 0, -80], 0.2, self.sonnenTextur, None, True,
self.light, 3, 64)
```

Wenn man eine Sonne hat, kann man dieser Planeten hinzufügen um sie um die Sonne rotieren zu lassen.

```
# Parameter(Planet): (position, anim, rotation, rotSpeed, rotPoint,
movSpeed, radius, textur, divisions, monde)
self.merkur = Planet([-5, 0, -80], True, [90, 0, 0], 0.05,
self.sonne.position, 0.0001, 1.25, self.merkurTextur, 32, None)


self.sonne.addPlanet(self.merkur)
```

Um einem Planeten einen Mond hinzuzufügen muss man folgenden Code ausführen (Planet muss wie im vorherigen Schritt beschrieben davor angelegt werden).

```
# Parameter(Mond): (anim, rotation, rotSpeed, parent, entf_rotPoint,
movSpeed, radius, textur, divisions)
self.mond = Mond(True, [-90, 0, -80], 0, self.erde, 5, -0.0005, 1,
self.mondTextur, 24)

self.erde.addMond(self.mond)
```

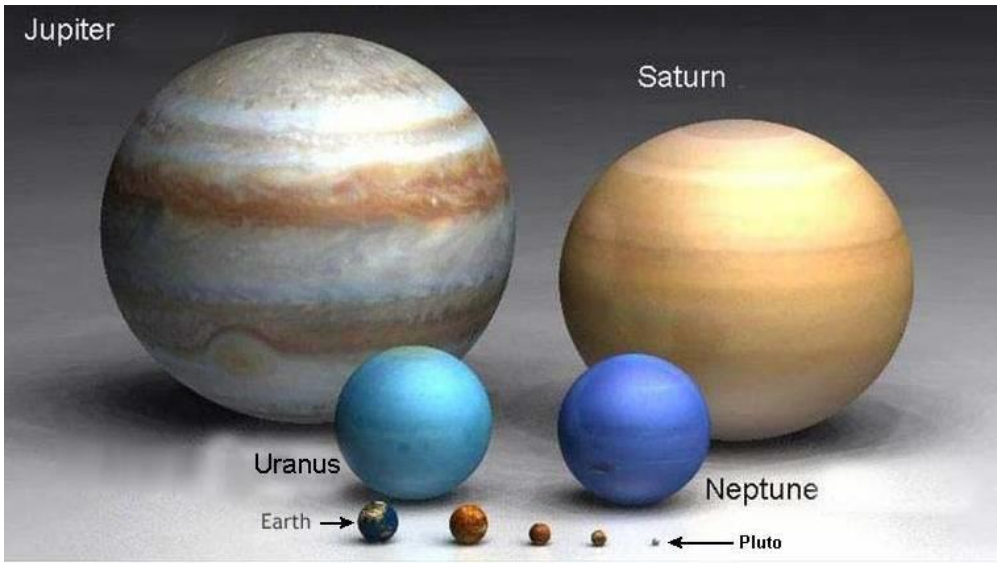
Relationen der Gestirne

Da die Sonne in unserem Sonnensystem im Gegensatz zu den anderen Planeten unwahrscheinlich groß ist wurde diese nicht in ihrem echten Verhältnis dargestellt (Fixstern).

Die Relationen der Planeten zueinander wurden bestmöglich an die echten Maßstäbe angepasst. [3]

Sonne	ca. 1.390.000 Kilometer	*Jupiter*	ca. 143.000 Kilometer
Merkur	ca. 4.900 Kilometer	*Saturn*	ca. 120.500 Kilometer
Venus	ca. 12.100 Kilometer	*Uranus*	ca. 51.100 Kilometer
Erde	ca. 12.800 Kilometer	*Neptun*	ca. 49.500 Kilometer
Mars	ca. 6.800 Kilometer		

Die Verhältnisse als Bild zur besseren Vorstellung:



Es werden allerdings nicht alle Planeten in unserem Sonnensystem zu sehen sein.

Tests

Unit Tests

Die wichtigsten Tests waren, ob das Programm richtig reagiert wenn man beim Anlegen eines bestimmten Gestirns falsche Werte übergibt.

Alle Testcases siehe Github
https://github.com/akrickl-tgm/solar01.git

UAC Tests

Test	Ergebnis	Kommentar
Funktional		
User kann Programm mit Anleitung starten	✓	
User kann zoomen	✓	Mithilfe der Anleitung aus dem Protokoll
User kann Texturen aus- und einschalten	✓	Mithilfe der Anleitung aus dem Protokoll
User kann Licht ein- und ausschalten	✓	Mithilfe der Anleitung aus dem Protokoll
User kann die Animation pausieren	✓	Mithilfe der Anleitung aus dem Protokoll
User kann die Kameraposition ändern	✓	Mithilfe der Anleitung aus dem Protokoll
User kann die Geschwindigkeit der Animation anpassen	✓	Mithilfe der Anleitung aus dem Protokoll
Nicht funktional		
User findet das Layout ansprechend	✓	
User findet die Animation ansprechend	×	Zu schnell anfangs
User hat Interesse daran das Programm zu entdecken	✓	
User würde das Programm gerne noch einmal starten	×	Nicht umfangreich genug

Ausführen und Steuerung

Um das Programm zu starten muss man im Package astrid die Datei start.py ausführen.

Die Steuerung funktioniert wie folgt.

Taste(n)	Aktion
l	Licht an/aus
t	Texturen an/aus
p	Animation an/aus
c	Kameraposition ändern & zurücksetzen
w/s	Animationen schneller/langsamer
+/-	Mehr/Weniger Zoom

Quellen

Kompletter Code siehe Github
https://github.com/akrickl-tgm/solar01.git

Im Text verwendet

[1] **Splashscreen with Python**
http://code.activestate.com/recipes/576936-tkinter-splash-screen/
02.03.2015

[2] **Tastatursteuerung**
https://www.opengl.org/documentation/specs/glut/spec3/node49.html
http://stackoverflow.com/questions/8272463/pyopengl-glut-input
16.03.2015

[3] **Größenverhältnisse der Planeten**
http://www.astronomie.de/astronomie-fuer-kinder/interessantes-fuer-lehrer-eltern/in-der-schule/groessenvergleich-der-planeten/
26.03.2015

Nachschlagen

PyOpenGL 3.x
http://pyopengl.sourceforge.net/
24.02.2015

Pillow 2.6.1
https://pypi.python.org/pypi/Pillow/2.6.1
24.02.2015

PyGame Download
http://www.pygame.org/download.shtml
24.02.2015

PyOpenGL Tutorial mcfeltch
http://bazaar.launchpad.net/~mcfletch/pyopengl-demo/trunk/view/head:/PyOpenGL-Demo/proesch/simple/simpleInteraction.py
03.03.2015

Rotation
http://nehe.gamedev.net/tutorial/arcball_rotation/19003/
04.03.2015