HTW Dresden
Wintersemester 2020/2021
Modul Computergrafik I
Beleg
Leitung: Prof. W. Oertel

Beleg

Vorgelegt von: Max Haufe
Studiengang: allg. Informatik
E-Mail: s80458@htw-dresden.de

Inhalt

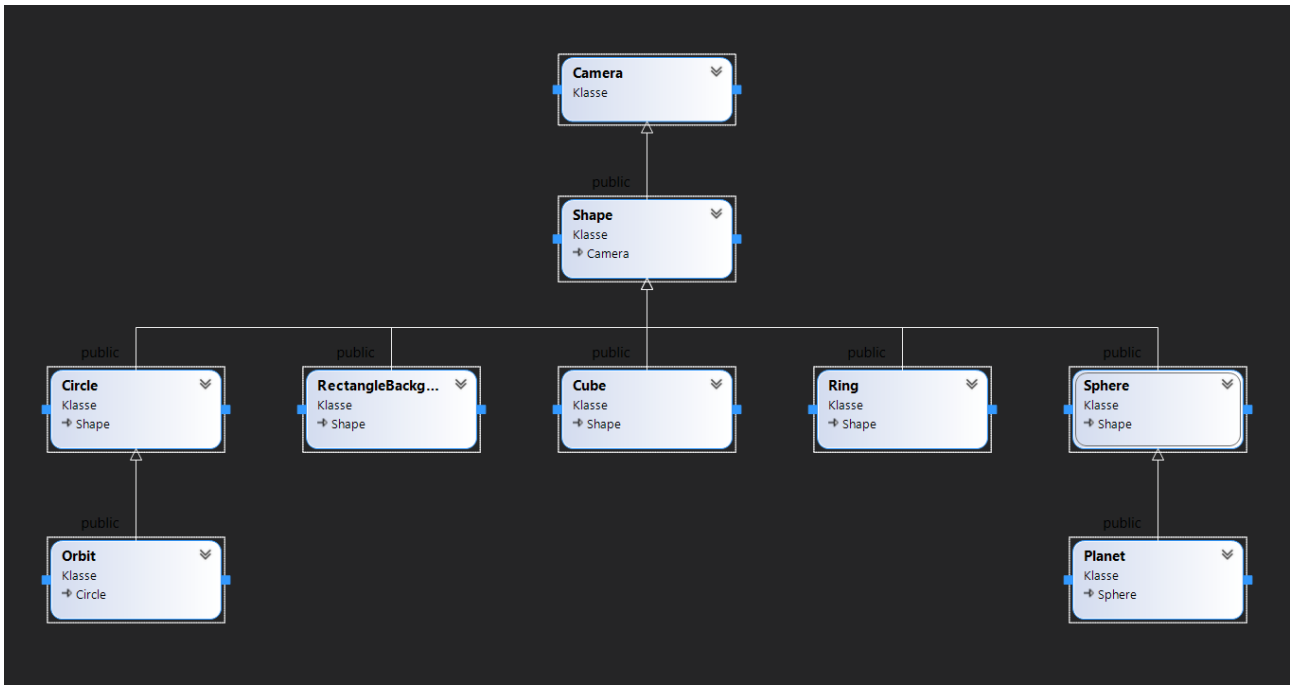
Aufgabenstellung

Die Aufgabe war, eine zeitlich animierte Szene mittels OpenGL, C++, sowie Fragment- und Vertex-Shadern zu realisieren.
Dabei waren folgende Rahmenbedingungen gegeben:

- Es muss verschiedenartige Lichtquellen geben, sodass unterschiedliche Beleuchtungseffekte sichtbar werden.
- Es muss mehr als zwei verschiedene dreidimensionale geometrische Objekte geben.
- Die Objekte müssen texturiert und farblich sein.
- Die Szene muss interaktiv sein.
- Die Szene muss in mehreren verschiedenen Ansichten und Projektionen dargestellt sein.

Lösungsansatz

Mein Ziel war von Anfang an, ein Sonnensystem darzustellen.
Daher brauchte ich ein neues dreidimensionales Objekt, eine Kugel. Dafür habe ich Klassen erstellt, die voneinander erben, weiteres dazu später.



Lösungsumsetzung

Die Klasse „Camera“ ist meine Basisklasse, welche die drei Vektoren, aus denen die View-Matrix gebildet wird, als private Member beinhaltet. Weiterhin enthält die Klasse die Lichtfarbe und Lichtposition, sowie einen Member, pointLightEnabled, welcher über das An und Aus des Punktlichtes entscheidet. Hat dieser Member den Wert false, so wird ein Richtungslicht angeschaltet. Im Falle des Punktlichts wird die Position des Lichts in der draw-Funktion der Klasse

„Shape“ automatisch festgelegt.

Die Klasse „Shape“ erbt von der Klasse „Camera“. Sie enthält den float-Wert textureEnable, welcher entweder den Wert 1.0f oder 0.0f erhält. Dieser Member dient als eine Art boolean, welcher an die Shader weitergegeben wird, um zu unterscheiden ob nun eine Textur oder Farbe gerendert werden soll. Der private Member path enthält den Pfad zu einem beliebigen Bild im jpg Format, welches auf ein dreidimensionales Objekt gemappt werden soll. Da sich die Bilder im gleichen Verzeichnis wie die Quelldateien befinden, ist der Pfad der Name des Bildes.
Des Weiteren sind 4 Arrays als Member enthalten, welche jeweils die Koordinaten, Farbe Normale und Texturkoordinaten eines Punktes speichern.
Die Klasse enthält weiterhin die Member singleColor, Model und Projection deren Bedeutung trivial ist. Es ist ein Member material des Typs Material enthalten, welcher aus drei vec3 Vektoren und einem float-Wert besteht und die ambienten, diffusen und spiegelnden Lichtanteile, sowie die Größe der Spiegelung beeinflusst.
Zusätzlich enthält die Klasse den Member image des Typs Image, welcher alle Parameter enthält, die für die Erzeugung der Textur nötig sind.
Mit dem Member draw() werden die Punkte aus dem Array vertices gezeichnet, davor werden jedoch mittels der Member pushMaterial() und pushMatrices() das Material, die ModelViewProjection Matrix, ModelView Matrix, Normal Matrix, Lichtposition, Lichtrichtung, sowie die EyeDirection an die Shader gegeben. Der Aufruf der Funktionen pushMaterial() und pushMatrices() erfolgt am Anfang von draw().
Mittels EnableTexture() und DisableTexture() können die Texturen an- und ausgeschalten werden.

Die Klasse „Sphere“ erbt von der Klasse „Shape“. Im Constructor werden die Arrays befüllt, welche Koordinaten, Farbe, Normale und Texturkoordinaten beinhalten. Eine Kugel lässt sich durch die Formel $r^2 = x^2 + y^2 + z^2$ beschreiben. Der Radius ist hierbei r. Genauere Informationen dazu befinden sich im Quellenverzeichnis.

Die Klasse „Planet“ erbt von der Klasse „Sphere“. Im Speziellen ist diese Klasse nur dafür da, die Sphären zu transformieren, skalieren und zu rotieren. Da dies für mehrere Planeten in der display() Funktion im Hauptprogramm zu unübersichtlich wird, habe ich diese Klasse erstellt.
Anmerkung: der Parameter rotationAngleOffset beschreibt die Neigung der Rotationsachse (z.B. Erde: 23.5°). Damit wird auch die Rotation mit dem Uhrzeigersinn möglich, wie z.B. bei der Venus.

Die Klasse „Circle“ erbt von der Klasse „Shape“. Die Funktionsweise ist aus dem Praktikum bekannt. Die diffusen Lichtanteile werden auf null gesetzt, während die ambienten Lichtanteile auf eins gesetzt werden.

Die Klasse „Orbit“ erbt von der Klasse „Circle“. Der Kreis wird jedoch so geneigt, dass er auf der xz-Ebene (OpenGL) bzw. xy-Ebene (mathematisch) liegt. Mittels der scale() – Funktion kann der Radius des Kreises angepasst werden.

Die Klasse „Ring“ erbt von der Klasse „Shape“. Sie ist dazu da, das Ringsystem des Saturn darzustellen. Der Ring besteht aus zwei Kreisen, deren Mittelpunkt derselbe ist, jedoch einen unterschiedlichen Radius haben. Somit kann man die Fläche zwischen den zwei Kreisen füllen und bekommt einen Ring.

Die Klasse „RectangleBackground“ erbt von der Klasse „Shape“ und dient dazu, ein Rechteck auf den gesamten Bildschirm zu zeichnen, auf den man später eine Textur als Hintergrundbild mappt.

Alternativ befindet sich im Projekt auch noch eine Klasse Cube, mit deren Hilfe ein Würfel dargestellt werden kann. Jedoch hat mir der Würfel gegen Ende des Projekts visuell eher wenig zugesagt, weshalb dieser im Endeffekt keine Verwendung im Projekt gefunden hat.

Die Shader orientieren sich im Wesentlichen an der Vorlesung.

Im Hauptprogramm werden die dreidimensionalen Objekte global definiert, damit nicht in jedem Aufruf von display() mehrere Kugeln neu ausgerechnet werden müssen.
In der init() Funktion werden die Texturen geladen. Die zeitliche Animation erfolgt wie in der über eine timer Funktion mittels einer global definierten Variablen, dem rotationAngle.
In der Funktion Planets() wird das Sonnensystem gezeichnet, die meisten Objekte erhalten nur Parameter, da sie der Klasse Planets angehören.
Der Mond hingegen dreht sich um ein sich bereits bewegendes Objekt, weshalb er nicht als Planet deklariert werden kann. Daher musste ich diese Einstellungen für diesen Sonderfall per Hand in der Funktion vornehmen.
In der Funktion CarbonAtom() wird ein Kohlenstoff-12 Atom gezeichnet. Für das Neigen der Elektronenbahnen habe ich Quaternions mit einem Euler-Vektor benutzt. Die Bahnen werden entlang der y-Achse gekippt und dann entlang der z-Achse um jeweils $\frac{\pi}{6}$ weitergedreht, sodass die Abstände der Bahnen gleichmäßig sind. Die kann man in ähnlicher Form auch auf die Elektronen anwenden, welche sich nun auf den Bahnen bewegen.
Timer-, Keyboard- und Mouse-Funktion sind in ähnlicher Art wie im Praktikum vorhanden.
Es gibt drei Program states, welche mittels der Tastendrücke 1, 2 oder 3 eingeleitet werden.
Im Program State 1 befindet sich das Planetenmodell im Vollbild und ist interaktiv zu bedienen.
Im Program State 2 befindet sich zwei Planetenmodell im Splitscreen und die Perspektive sowie der Betrachterpunkt ist fest im Programm eincodiert. Besonders schön ist das nicht, jedoch dient dieser Program State nur dazu, die Aufgabenstellung zu erfüllen.
Im Program State 3 wird das C-12 Atom angezeigt.
In der main Funktion wird lediglich zwischen den drei Program States unterschieden und es wird der Aufruf der richtigen Funktion mit den richtigen Projektionen und Beobachterpositionen getätigt.

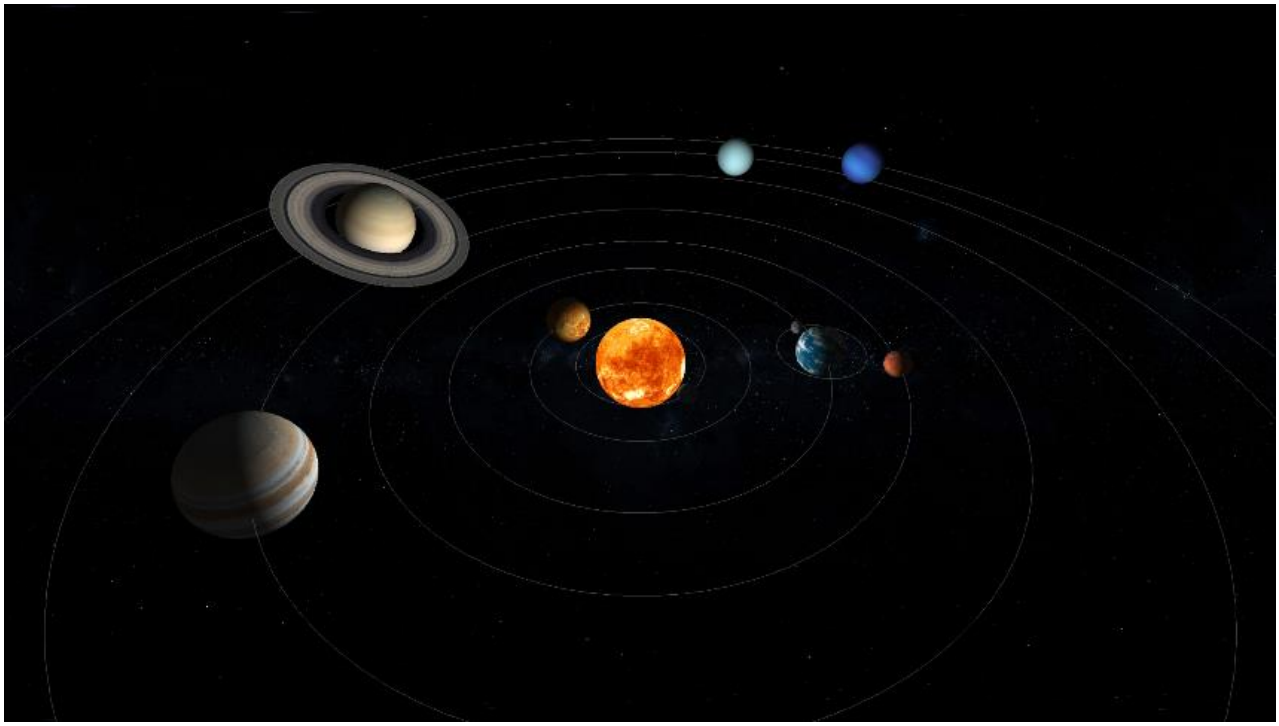
Installations- und Bedienungsanleitung

Für die Installation ist wichtig, dass die Schritte zur Einrichtung von OpenGL abgeschlossen sind (siehe Praktikum). Das mitgelieferte Archiv muss entpackt werden, danach muss auf die Verknüpfung zur Executable geklickt werden.

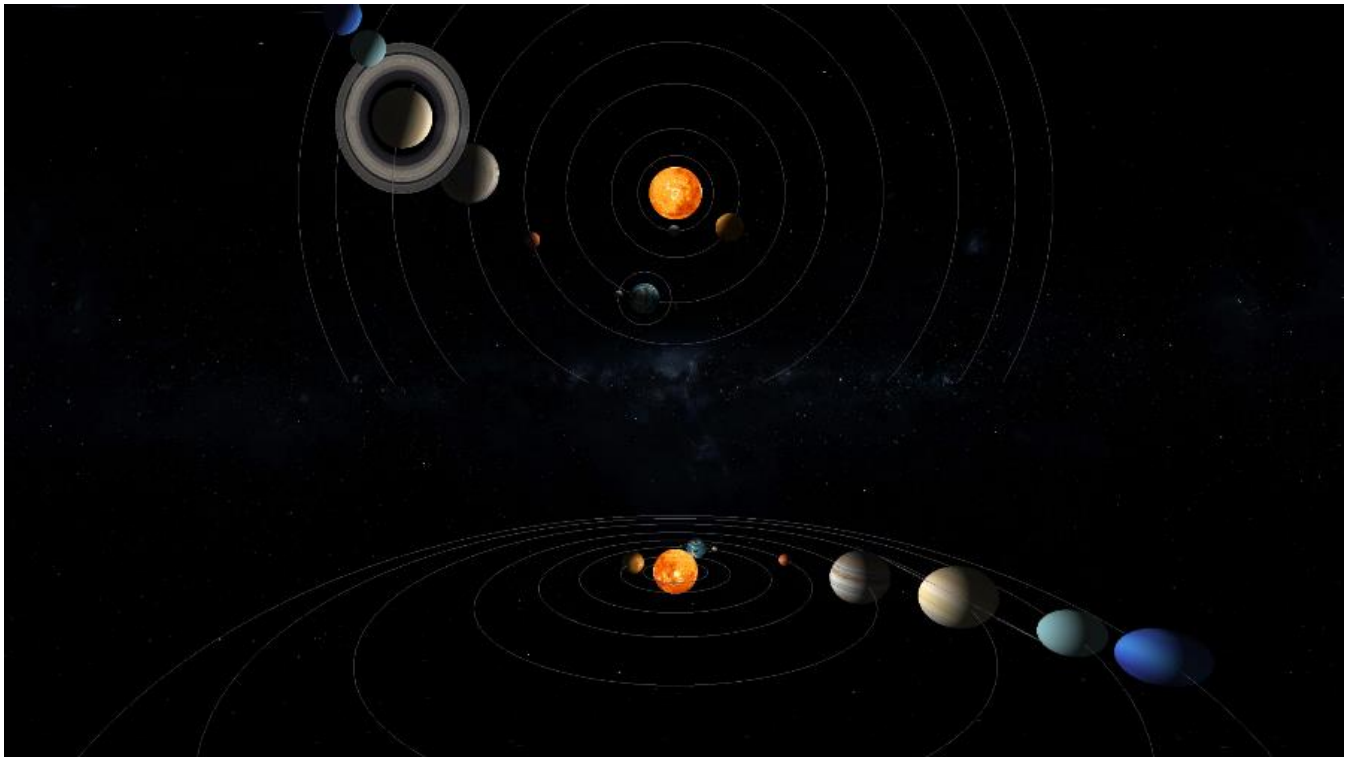
Für die Bedienung gibt es unterschiedliche Eingaben, welche via Tastatur oder Maus getätigt werden müssen, damit ein Event ausgelöst wird.
Das Bewegen der Szene funktioniert mit Maus und/oder Tastatur so, wie im Praktikum.

Im Quelltext befinden sich in der init() Funktion noch mehr auskommentierte Aufrufe der Funktion EnableTexture() mit anderen Bildern. Auf Nachfrage kann ich Ihnen diese Bilder auch zukommen lassen.

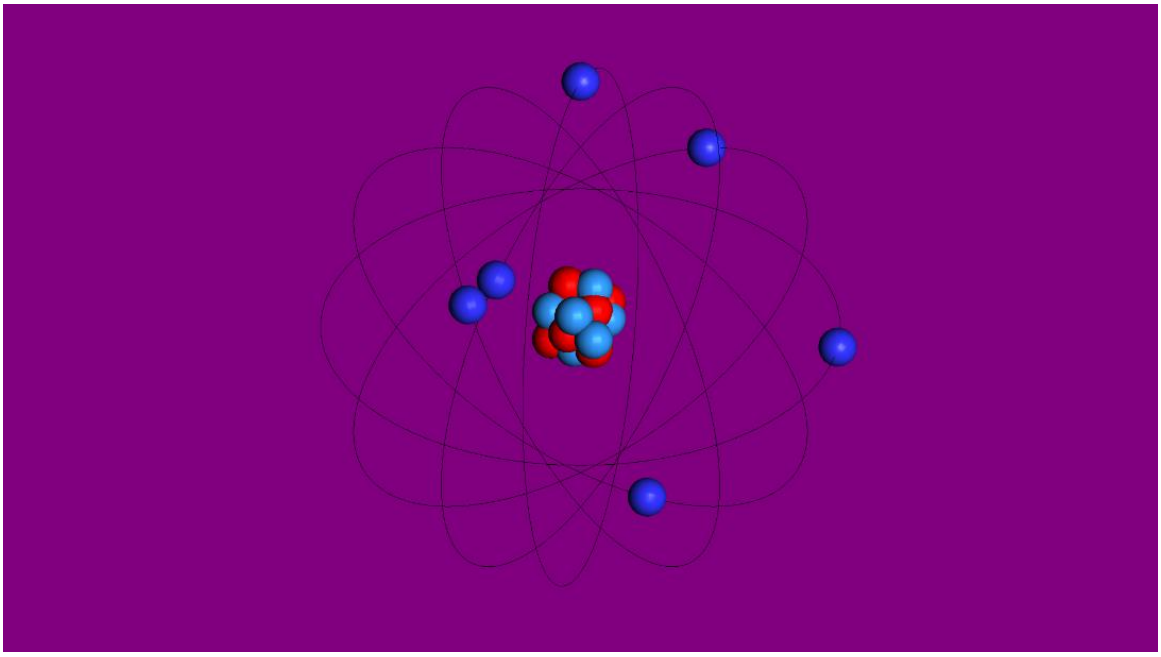
Mit der Taste „1“ wird das Sonnensystem dargestellt, man kann sich mit den Pfeiltasten und der Maus darin herumbewegen.



Mit der Taste „2“ wird das Fenster vertikal in zwei Viewports geteilt, im oberen befindet sich eine Orthogonale Draufsicht, im unteren eine leicht schräge perspektivische Sicht von vorn.

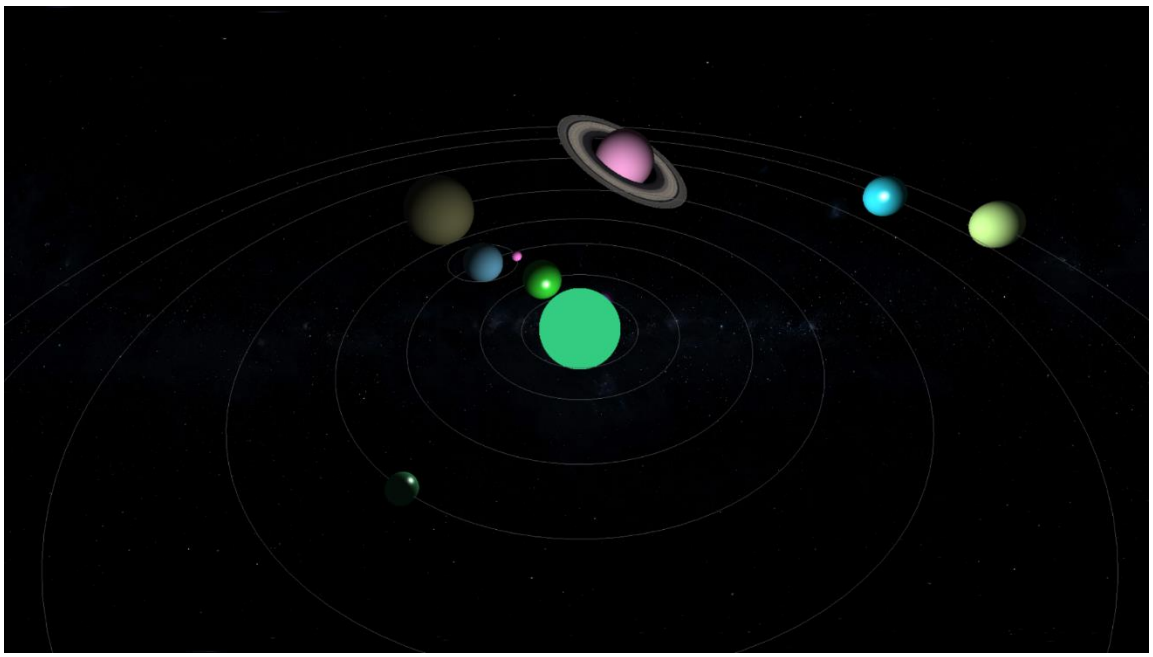


Mit der Taste „3“ sieht man das Atommodell.

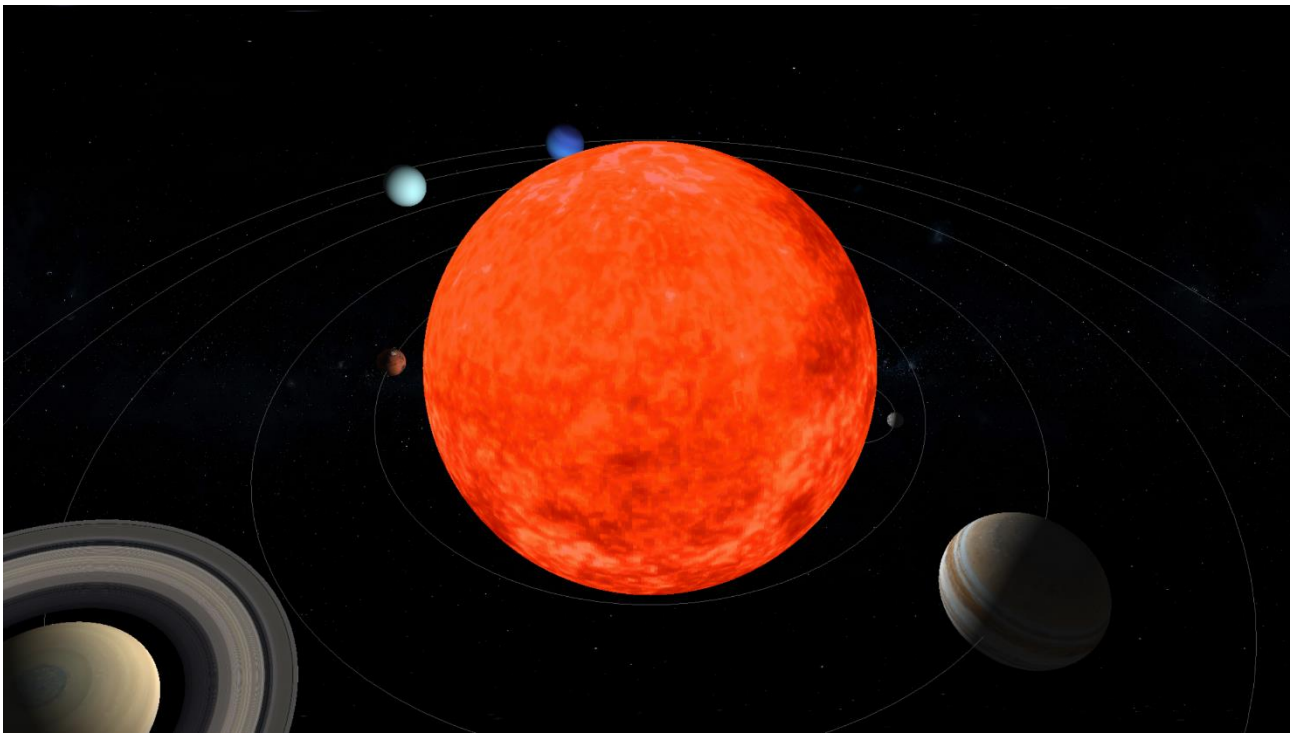


Mit der Taste „L“ schaltet man zwischen Punktlicht und Richtungslicht (von rechts) um. Standardmäßig ist das Punktlicht vom Ursprung aus eingestellt.

Mit der Taste „T“ schaltet man die Texturen der Himmelskörper aus/ein. Standardmäßig sind die Texturen angeschaltet. Die Farben und das Reflexionsverhalten beim Ausschalten der Texturen sind zufällig.



Mit der Taste „G“ wächst die Sonne zu einem Roten Riesen und verschlingt das Sonnensystem.



Mit der Taste „R“ werden die Elektronen beim Atommodell synchronisiert/nicht synchronisiert.

„R“ bezieht sich also nur auf die dritte Ansicht, das Atommodell, während sich die anderen Tasten auf die ersten beiden Ansichten beziehen.

Bekannte Bugs und Probleme

- Beim Anzeigen der zweiten Ansicht, also des Splitscreens mit den zwei Sonnensystemen läuft das Programm auf meinem Laptop nicht mehr flüssig (es „ruckelt“ und bewegt sich langsamer). Das liegt vermutlich an folgenden Ursachen:
    - Ich habe irgendwo sehr ineffizient programmiert.
    - Mein Laptop hat keine Grafikkarte.
- Hin und wieder funktioniert das Depth Buffering nicht richtig. Dieses Problem ist in der ersten Hälfte des Projekts selten aufgetreten, in der zweiten Hälfte gar nicht mehr. Ich habe diesbezüglich jedoch nichts gefixt, möchte es deswegen aufführen, falls es wieder auftritt.

Anmerkungen

Zum Sonnensystem

Die Abstände der Himmelskörper untereinander sind nicht maßstabsgerecht. Die Größe und Revolutionsgeschwindigkeit von Erde, Mars, Merkur und Venus sind maßstabsgerecht. Die Größe des Mondes ist maßstabsgerecht, die Revolutionsgeschwindigkeit ist im Modell nur halb so groß wie in Wirklichkeit. Dafür ist das Verhältnis von Revolutions- und Rotationsgeschwindigkeit beim Mond richtig.
Bei allen Planeten sind die Rotationsachsen korrekt, die Rotationsrichtung auch (siehe Venus).
Die Rotationsgeschwindigkeiten habe ich grob abgeschätzt, das Verhältnis untereinander spiegelt aber keineswegs die Realität wider.

Mir ist bewusst, dass die Sonne als Roter Riese nicht das gesamte Sonnensystem „verschluckt“, jedoch ist meine Darstellung des Ganzen aufregender.

Weiterhin ist mir bewusst, dass die Sonne kein Planet ist, trotz der Tatsache, dass sie in meinem Quelltext der Klasse der Planeten angehört. Das war übersichtlicher.

Pluto ist nicht enthalten, da kein Planet.

Auf Wunsch von Herrn Struck habe ich im Fragment Shader eine Zeile hinzugefügt. Wenn Sie die Kommentierung umkehren, dann passieren witzige Sachen: Voraussetzung dafür ist folgende:

- Die Texturen wurden vorher mindestens einmal aus- und angeschaltet (Mit der Taste „T“).

Wenn z.B. die Sonne durch das Ausschalten der Texturen eine grüne Farbe zugewiesen bekommen hätte, und man danach die Textur wieder anschaltet, dann bekommt die Sonne einen grünen Schimmer. Die Texturen werden sozusagen mit der Farbe multipliziert. Dadurch wird die Farbe der Planeten verändert.
Dabei handelt es sich um folgende zwei Zeilen in der Datei beleg.fs:

```
vec3 rgb=min(text.rgb*scatteredLight+reflectedLight,vec3(1.0,1.0,1.0));
//vec3 rgb=min(Color.rgb*text.rgb*scatteredLight+reflectedLight,vec3(1.0,1.0,1.0));
```

Zum Atommodell

Dies war eine Spielerei, auf die ich beim Bearbeiten des Belegs und bei einer Folge der Sitcom „The Big Bang Theory“ gekommen bin. Ich fand dies eigentlich schön und habe es daher mit in den Beleg mit aufgenommen.

Literatur- und Quellenverzeichnis

Die Berechnung aller Punkte einer Sphäre
http://www.songho.ca/opengl/gl_sphere.html#:~:text=In%20order%20to%20draw%20the,triangle%20strip%20cannot%20be%20used.

Aussehen eines Atommodells
https://www.welt.de/wissenschaft/article124313918/Geschrumpfte-Protonen-bereiten-Kopfzerbrechen.html

Wie man Quaternions und Euler Vektoren benutzt
http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-17-quaternions/

Hilfe bei der Auswahl der Farben
https://www.w3schools.com/colors/colors_picker.asp

Inspiration für das Planetenmodell
https://www.dlr.de/next/desktopdefault.aspx/tabid-10515/18251_read-42646/

Texturen für das Planetenmodell
https://www.solarsystemscope.com/textures/