Computergrafik 1 - Beleg

Dokumentation

Theresa Schüttig

15. Januar 2020

Inhaltsverzeichnis

1 Aufgabenbeschreibung **3**

2 Lösungsansatz **4**
2.1 Vorüberlegung 4
2.2 Skizze 5

3 Lösungsumsetzung **6**
3.1 Berechnungen 6
3.2 Programmstruktur 7

4 Installation **10**
4.1 Windows 10
4.2 Linux 10

5 Bedienung **10**
5.1 Entfernen und Wiederhinzufügen von Lichtquellen 10
5.2 Rotieren und Zoomen 10
5.3 Beenden des Programms 10

6 Screenshots **11**

7 Probleme **12**
7.1 Keine nahtlose Texturierung 12
7.2 Kein Schattenwurf 12
7.3 Blick in das Innere der Objekte möglich 12

8 Ergebnisse **13**

9 Literatur- und Quellenverzeichnis **14**
9.1 Literatur 14
9.2 Quellcode 14
9.3 Bilder 14

1 Aufgabenbeschreibung

Schreiben Sie ein Programm in C/C++, das unter Verwendung von OpenGL, Vertex- und Fragment-Shadern folgende Aufgaben realisiert.

Aufgabe 1
Geometrische Objekte: Erzeugen Sie eine interaktive zeitlich animierte Szene mit mehreren unterschiedlichen farblichen und texturierten dreidimensionalen geometrischen Objekten.

Aufgabe 2
Beleuchtung: Beleuchten Sie die Szene mit verschiedenartigen Lichtquellen so, dass auf den Objekten unterschiedliche Beleuchtungseffekte sichtbar werden.

Aufgabe 3 Ansicht: Stellen Sie die Szene gleichzeitig in verschiedenen Ansichten und Projektionen in mehreren Viewports des Anzeigefensters dar.

Aufgabe 4
Programm: Stellen Sie das komplette Programm in Quelltextform als Visual-Studio-C++- Projekt und in ausführbarer Form als exe-File derart bereit, dass die Lauffähigkeit auf den Computern des Praktikumslabors der Lehrveranstaltung gewährleistet ist.

Aufgabe 5
Dokumentation: Fertigen Sie eine Systemdokumentation in Form eines pdf-Dokumentes von etwa 10 Seiten an, die Deckblatt, Gliederung, Aufgabenbeschreibung, Lösungsansatz, Lösungsumsetzung, Installations- und Bedienungsanleitung, einige Bildschirm-Snapshots, Probleme, Ergebnisse, Literatur- und Quellenverzeichnis enthält.

Aufgabe 6
Abgabe: Demonstrieren Sie die Ergebnisse der Aufgaben 4 und 5 an einem Computer des Praktikumslabors der Lehrveranstaltung und übergeben Sie diese in einem Verzeichnis „Name_Vorname_Bibliotheksnummer“.

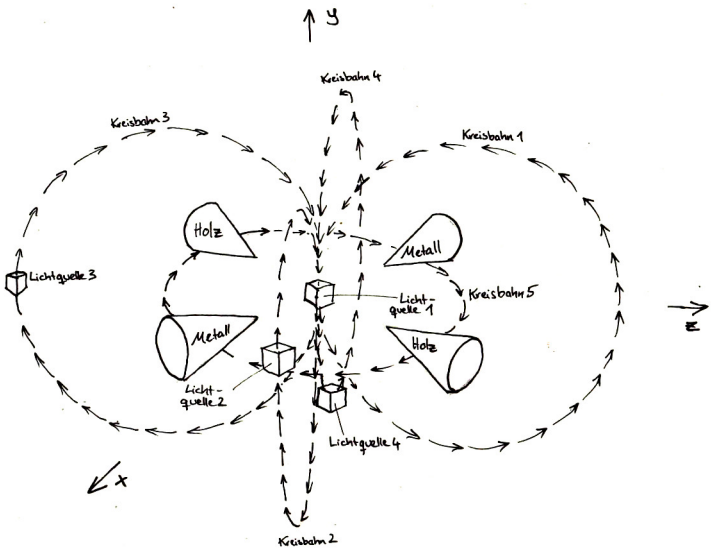
Zeitplan
Die Ausgabe der Aufgabenstellung erfolgt zu Beginn der Lehrveranstaltungszeit. Die Abgabe der Ergebnisse erfolgt spätestens zum Ende der Lehrveranstaltungszeit.

2 Lösungsansatz

2.1 Vorüberlegung

Das Programm soll 4 Kegel darstellen, die sich mit gleicher Geschwindigkeit und in gleicher Richtung auf einer Kreisbahn bewegen, auf der sie gleichmäßig verteilt sind. Der Radius der Kreisbahn schwankt in Abhängigkeit der Zeit gleichmäßig um einen festen Wert. Die Spitze der Kegel soll dabei stets auf den Mittelpunkt des Kreises gerichtet sein. Zwei der Kegel besitzen eine Holztextur, die anderen zwei Kegel besitzen eine Metalltextur und glänzen zusätzlich. Kegel aus dem gleichen Material befinden sich auf der Kreisbahn gegenüber.
Die Szene wird durch ein Richtungslicht und 4 verschiedenfarbige Punktlichte, welche als einfarbige Würfel ohne Schattierung dargestellt werden, beleuchtet. Jeder Würfel bewegt sich auf einer eigenen Kreisbahn mit festem Radius, die einen Punkt enthält, der Mittelpunkt der Kreisbahn der Kegel ist. Der Abstand zwischen dem Mittelpunkt einer Lichtkreisbahn und dem Mittelpunkt der Kegelkreisbahn ist bei jeder Lichtquelle gleich. Die Ebene, auf der die Kreisbahn einer Lichtquelle liegt und die Ebene der Kreisbahn, auf der die Kegel liegen, sind orthogonal zueinander. Die Schnittpunkte zwischen der Ebene einer Lichtkreisbahn und der Kegelkreisbahn sind gleichmäßig auf der Kegelkreisbahn verteilt. Alle Würfel befinden sich entweder über oder unter (abhängig vom Blickwinkel) der Ebene der Kegelkreisbahn, wenn sie sich auf dessen Mittelpunkt zubewegen. Die Lichtquellen sollen nacheinander mit gleichem Zeitabstand auf diesen Mittelpunkt treffen.
Die Bewegung aller Objekte erfolgt periodisch und die Größe der Kreisbahnen und Objekte sind so zu wählen, dass zu keinem Zeitpunkt zu einer Überlappung zwischen zwei Objekten kommt.
Im Fenster sollen vier verschiedene Viewports enthalten sein, von denen jeder die Szene aus einem anderen Blickwinkel darstellt. Für einen Viewport kommt eine orthographische Projektion zum Einsatz, für alle anderen Viewports wird eine perspektivische Projektion verwendet. Zudem kann der Nutzer in einem der Viewports zoomen und die Szene rotieren.

2.2 Skizze


y
Kreisbahn 4
Kreisbahn 3
Kreisbahn 1
Holz
Metall
Lichtquelle 3
Licht-
quelle 1
Kreisbahn 5
z
Metall
Holz
Licht-
quelle 2
Lichtquelle 4
x
Kreisbahn 2

3 Lösungsumsetzung

3.1 Berechnungen

Die Kegel rotieren um den Koordinatenursprung und besitzen folgende Koordinaten:

Kegel 1: K_1(a | 0 | -b)
Kegel 2: K_2(b | 0 | a)
Kegel 3: K_3(-a | 0 | b)
Kegel 4: K_4(-b | 0 | a)

a und b werden folgendermaßen berechnet:

$$a = sin(f(t)) \cdot (1,025 - 0,35 \cdot sin(2 \cdot f(t)))$$
$$b = cos(f(t)) \cdot (1,025 - 0,35 \cdot sin(2 \cdot f(t)))$$

f(t) ist eine von der Zeit abhängige Funktion, die den aktuellen Winkel im Gradmaß angibt und folgendermaßen definiert ist:

$$f : [0, \infty) \to [0, 359],\ t \mapsto t \cdot 10^{-2} \mod 360$$

Der Radius schwankt also periodisch zwischen $0,675$LE und $1,375$LE mit einer Periodendauer von 1,8s, wobei der Radius als Abstand zwischen dem Koordinatenursprung und dem Mittelpunkt eines Kegels definiert ist. Für die Höhe eines Kegels wurde 1LE festgelegt, so dass auf der x-z-Ebene ein Kreis mit Radius r = 0,175LE entsteht, durch den laut Satz des Pythagoras in jedem Fall ein Würfel mit einer Kantenlänge von bis zu $\sqrt{2 \cdot r^2}$ = 0,247LE über den Mittelpunkt der Kegelkreisbahn gelenkt werden kann, ohne dass es zur Überlappung mit einem Kegel kommt. Im Programm wurden für die Kantenlänge 0,25LE festgelegt. Da sich im Mittelpunkt der Kegelkreisbahn kein Würfel befindet, wenn die Kegelkreisbahn ihren minimalen Radius erreicht, kommt es mit diesem leicht höheren Wert zu keiner Überlappung.
Damit die Kegelspitzen in Richtung Koordinatenursprung zeigen, muss jeder Kegel um einen Vektor rotiert werden, der die Kegelmitte schneidet und parallel zur y-Achse verläuft. Der Winkel α_i, um den der Kegel i gedreht werden muss, wird folgendermaßen berechnet:
$\alpha_i = -f(t) + (i - 1) \cdot 90°$

Die Lichtquellen rotieren um folgende Koordinaten:
Licht 1: A(1,5 | 0 | 0)
Licht 2: B(0 | 0 | -1,5)
Licht 3: C(-1,5 | 0 | 0)
Licht 4: D(0 | 0 | 1,5)

Auf Basis der Angaben im Lösungsansatz lassen sich die von der Zeit abhängigen Koordinaten L_i, $i \in \{1, 2, 3, 4\}$ der Lichter $1-4$ folgendermaßen berechnen:

$$L_1 = A + \begin{pmatrix} -sin(f(t)) \cdot 1,5 \\ cos(f(t)) \cdot 1,5 \\ 0 \end{pmatrix}$$

$$L_2 = B + \begin{pmatrix} 0 \\ cos(f(t) + 90°) \cdot 1,5 \\ sin(f(t) + 90°) \cdot 1,5 \end{pmatrix}$$

$$L_3 = C + \begin{pmatrix} sin(f(t) + 180°) \cdot 1,5 \\ cos(f(t) + 180°) \cdot 1,5 \\ 0 \end{pmatrix}$$

$$L_4 = D + \begin{pmatrix} 0 \\ cos(f(t) + 270°) \cdot 1,5 \\ -sin(f(t) + 270°) \cdot 1,5 \end{pmatrix}$$

3.2 Programmstruktur

Main.cpp

- **void main(int argc, char *argv[])** Nimmt notwendige Initialiserungen zur Nutzung von OpenGL vor, erzeugt ein Fenster und legt bestimmte in *Program.cpp* definierte Funktionen als Callback-Funktionen fest.

Program.cpp

- **void init()** Generiert Texturen und VAOs, VBOs und Puffer, ermittelt die Positionen der Shader-Variablen, übergibt dem Fragment-Shader die Lichtfarben und aktiviert die Tiefenprüfung.
- **void display()** Leert den Farb- und Tiefenpuffer, befüllt diesen neu und erzeugt vier quadratische Viewports gleicher Größe. Vor dem Rendern eines Viewports werden die Matrizen *View* und *Projection* modifiziert.
- **void renderScene()** Berechnet einen von der Zeit abhängigen Winkel, der im Schnitt alle 10ms um 1° erhöht wird. Über diesen Winkel werden die Kegel- und Würfelpositionen berechnet. Für jedes Objekt werden die Matrizen *Model* und *ModelViewProjection* und *NormalMatrix* berechnet. Vor dem Rendern der Würfel wird dem Fragment-Shader die Nummer der jeweiligen Lichtquelle über das Setzen von *isLightSource* übergeben. Nach dem Rendern wird diese Variable wieder auf 0 gestetzt.
- **void loadTextures()** Bindet die Texturen, deren Pfade in *texturePaths* gespeichert sind, an *GL_ TEXTURE_ 2D*.

- **void setViewPoint()** Berechnet die Matrix *View* basierend auf der vom Nutzer festgelegten Betrachterposition und übergibt dem Fragment-Shader die Betrachterposition.
- **void reshape(int w, int h)** Aktualisiert die Variablen *width* und *height* für die Fenstergröße, wenn diese geändert wird.
- **void timer(int value)** Wird kontinuierlich nach minimal 10ms erneut aufgerufen und leitet den erneuten Aufruf von display ein.
- **void keyboard(unsigned char theKey, int mouseX, int mouseY)** Wird bei Betätigen einer Taste der Tastatur aufgerufen und verändert die Variablen, die Entfernung der Blickposition vom Mittelpunkt der Szene bestimmen.
- **void special(int specKey, int mouseX, int mouseY)** Wird aufgerufen, wenn eine Funktions- oder Pfeiltaste betätigt wird. Wurde eine Pfeiltaste betätigt, so verändert die Funktion Variablen, die die Blickposition bestimmen.
- **void motion(int mouseX, int mouseY)** Wird bei Mausbewegung aufgerufen, wenn mindestens eine Maustaste gedrückt gehalten wird und verändert Variablen, die die Blickposition bestimmen.

Data.hpp

Enthält die Bezeichner für die VAOs, VBOs und Buffer-Objekte, welche in *Cone.cpp*, *Cube.cpp* und *Program.cpp* verwendet werden.

Cube.cpp

- **void generateCube()** Erzeugt die Würfel- und Texturkoordinaten sowie Normalenvektoren, bindet diese an den *GL_ARRAY_BUFFER* und aktiviert die Attributarrays für den Vertex-Shader.
- **void drawCube(int texID)** Rendert einen Würfel mit der Textur, der *texID* als Bezeichner zugewiesen wurde.

Cone.cpp

- **void generateCone()** Erzeugt die Kegel- und Texturkoordinaten sowie Normalenvektoren, bindet diese an den *GL_ARRAY_BUFFER* und aktiviert die Attributarrays für den Vertex-Shader.
- **void drawCone(int texID)** Rendert einen Kegel mit der Textur, der *texID* als Bezeichner zugewiesen wurde.

- **void calcConeTexCoords(float h, float r, float vertices[][2])** Berechnet die Texturkoordinaten für einen Kegel mit der Höhe h und dem Radius r und speichert diese im Array *vertices*, welches mindestens eine Größe von 12· *NumVertices* besitzen muss.

LoadShader.cpp

- **GLuint LoadShaders(const char *vertexFilePath, const char *fragmentFilePath)** Kompiliert den Vertex- und Fragment-Shader, deren Pfade als Parameter übergeben werden.

main.vs

- **void main()** Berechnet *gl_Position* und übergibt dem Fragment-Shader den Normalenvektor und Textur- und Vertexkoordinaten.

main.fs

- **void main()** Summiert die Lichter aller Lichtquellen, und legt das Ergebnis als Farbe des Fragments fest, falls *isLightSource* 0 ist. Andernfalls erhält das Fragment eine leicht aufgehellte Variante der Farbe der Lichtquelle.
- **vec3 calcLight(int lightID)** Berechnet den ambienten, diffusen und spekularen Anteil der Lichtquelle mit der Bezeichnung *lightID*. Ist *useSpecularLight* 0, so wird der spekulare Anteil auf 0 gesetzt. Ist *lightID* 4, so wird *lightDir* als Richtungsvektor des Lichts verwendet, andernfalls wird dieser über die Lichtposition (*lightPos[lightID]*) und die Fragmentposition (*FragPos*) ermittelt. Zurückgegeben wird das Produkt aus der Texturfarbe und der Summe des ambienten, diffusen und spekularen Anteils.

4 Installation

Unabhängig vom Betriebssystem müssen vor der Installation die Bibliotheken *OpenGL*, *GLUT*, *GLM* und *FreeImage* installiert werden. Die ausführbare Datei befindet sich nach der Installation im Verzeichnis *bin*.

4.1 Windows

Unter Windows kann die Projektionmappendatei mit Visual Studio Community geöffnet und über Erstellen -> Projektmappe kompiliert werden.

5 Bedienung

5.1 Entfernen und Wiederhinzufügen von Lichtquellen

Jedes Punktlicht kann durch das Betätigen einer bestimmten Taste entfernt oder wieder hinzugefügt werden. Für jede Lichtquelle kann dieser Vorgang beliebig oft wiederholt werden.

Taste	Farbe der Lichtquelle
R	rot
O	orange
G	gelb
B	blau

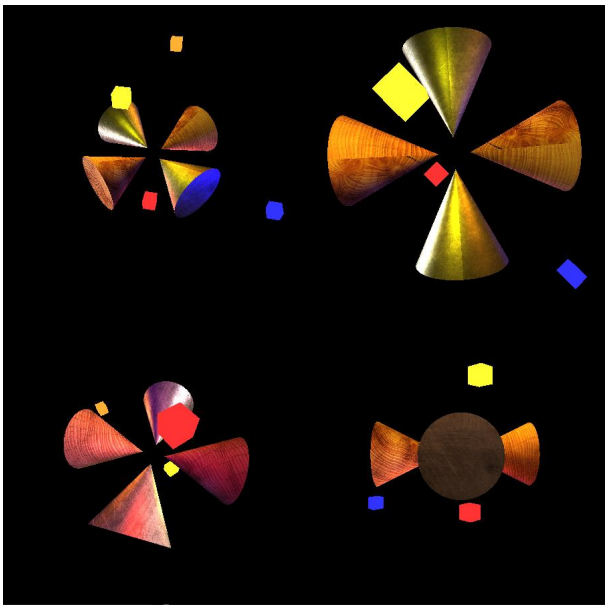
5.2 Rotieren und Zoomen

Im linken oberen Viewport kann der Nutzer in der Szene zoomen und rotieren. Das Zooming erfolgt über die Pfeiltasten oder über das Gedrückthalten einer Maustaste mit gleichzeitigem Bewegen der Maus. In Richtung Mittelpunkt kann über die V-Taste gezoomt werden, das Herauszoomen erfolgt über die Z-Taste.

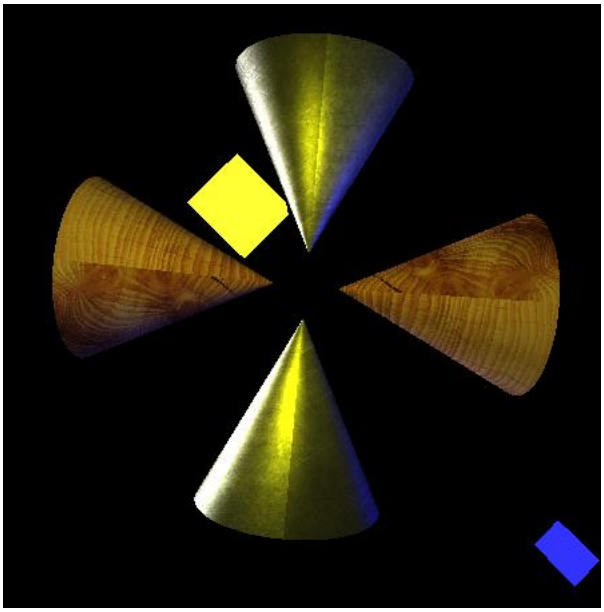
5.3 Beenden des Programms

Das Programm kann über das Schließen des Fensters oder das Betätigen der E-Taste beendet werden.

6 Screenshots



Alle Viewports mit allen Lichtquellen



Rechter unterer Viewport ohne rotes und oranges Punktlicht

Linker oberer Viewport ohne Punktlichter

7 Probleme

7.1 Keine nahtlose Texturierung

Auf der Mantelfläche und zwischen Mantel- und Grundfläche befinden sich Nahtstellen. Diese könnte man entfernen, indem man unter Verwendung der vom Programm erstellten Texturkoordinaten eine speziell für die Kegel nahtlose Textur erstellt.

7.2 Kein Schattenwurf

Die in der Szene dargestellten Objekte werfen keinen Schatten auf ein anderes Objekt, wenn sie sich zwischen diesem und einer Lichtquelle befinden, woraus ein Verlust von Realismus folgt. Zur Lösung des Problems bietet sich der Einsatz von Shadow-Mapping an.

7.3 Blick in das Innere der Objekte möglich

Durch geschicktes Zoomen und Rotieren der Szene kann der Betrachter die Blickposition in das Innere eines Objektes verschieben, was im Idealfall nicht möglich sein sollte. Um dies zu verhindern, kann im Hauptprogramm überprüft werden, ob die Position des Betrachters bei einer bestimmten Veränderung mit einem der Objekte kollidieren würde. In diesem Fall würde diese Änderung nicht vorgenommen werden.

8 Ergebnisse

Die Anfertigung des Belegs half, Kenntnisse von Grundlagen, insbesondere Licht und Texturen, zu festigen und zu verstehen, wie mehrere, sich bewegenden Lichtquellen in einer Anwendung implementiert werden können.

9 Literatur- und Quellenverzeichnis

9.1 Literatur

- OpenGL Programming Guide, Addison Wesley, 2013, 8. Auflage
- Vorlesungsskript
- https://learnopengl.com/

9.2 Quellcode

- einzelne Funktionen: Praktikumsunterlagen

9.3 Bilder

- Holztextur: https://pixabay.com/photos/wood-tree-spruce-picea-conifer-3212803/
- Metalltextur: https://pixabay.com/photos/background-texture-metal-scratches-1172581/