Hochschule für Technik und
Wirtschaft Dresden
University of Applied Sciences

Schachbrett - Beleg Computergrafik I

Lotte Richter

30.03.2025

Contents

Einleitung

Computergrafik bildet die technische Grundlage für die Darstellung interaktiver 3D-Szenen in Bereichen wie Spieleentwicklung und Simulationen. Im Rahmen des Moduls Computergrafik I (I-240) bei Prof. Dr.-Ing. habil. W. Oertel wurde ein 3D-Schachbrett entwickelt, das moderne Rendering-Techniken mit OpenGL und GLUT implementiert. Das Projekt verfolgte drei Hauptziele:

– Realistische Darstellung eines Schachbretts mit texturierten Figuren

– Dynamische Beleuchtung mit verschiedenen Lichtquellen

– Multi-Viewport-Darstellung mit synchronisierten Ansichten

Motivation

Bei der Themenwahl für den Beleg habe ich die im Praktikum erstellten Programme durchgesehen. Dabei fiel mir die Übung zur Texturierung besonders auf, insbesondere das Schachbrettmuster. Da ich durch das Praktikum bereits eine solide Grundlage hatte, entschied ich mich, ein vollständiges 3D-Schachbrett zu programmieren.

Aufgabenbeschreibung

Die Aufgabe bestand darin, ein Programm in C/C++ unter Verwendung von OpenGL sowie Vertex- und Fragment-Shadern zu schreiben. Das Programm sollte mehrere unterschiedlich farbige und texturierte dreidimensionale geometrische Objekte erzeugen und die Szene mit verschiedenen Lichtquellen beleuchten, sodass die Beleuchtungseffekte sichtbar werden. Zudem sollte die Szene in mehreren Viewports mit unterschiedlichen Projektionen dargestellt werden.

Ich entschied mich für ein Schachbrett als zentrale Szene, auf dem Spielfiguren erzeugt werden.

Lösungsansatz und -umsetzung

Der Aufbau der Szene wurde systematisch in mehrere Komponenten unterteilt, die schrittweise implementiert wurden.

Ich habe den Code aus dem Praktikum als Grundlage genommen, wo wir mit Texturen gearbeitet haben. Dadurch hatte ich eine grobe Vorstellung, wie das Programm am Ende aussehen muss.

Schachbrett und Texturierung

Zunächste habe ich das Schachbrett als grundlegende 2D-Textur erstellt, die ein 8x8-Felder-Muster in den Farben Weiß und Braun darstellt. Später habe ich die Farben von

weiß-braun zu weiß-schwarz geändert, da mir das Braun nicht so gut gefallen hatte. Dabei wurde die RGBA-Textur verwendet, um Transparenz und Farbkanläle flexibel steuern zu können.

Das Schachbrett wurde anschließend auf einen 3D-Quader projiziert, um ihm eine physische Präsenz in der Szene zu verleihen.
Zuerst habe ich dem Schachbrett einfach einen Rand hinzugefügt, der so breit war, wie ein Kästchen, allerdings sah das nicht so gut aus und ich habe mich für den Quader entschieden.
Als Untergrund dient dieser braune Quader mit Holztextur, der dem Brett einen realistischen Rahmen gibt, dementsprechend ist der Quader ein wenig größer, als das Schachbrett-Muster. Die Textur wurde mithilfe der FreeImage-Bibliothek geladen und mittels OpenGL auf die Oberflächen appliziert.

Damit die verschiedenen Objekte auch die richtigen Texturen zugewiesen bekommen, habe ich im Fragment-Schader die Berechnung in Anhängigkeit der Objekt-ID, mit folgenden Zuweisungen gemacht:
0 – Brett (Board)
1 – Schachbrett-Muster (Checkerboard)
2 – Weiße Schachfiguren
3 – Schwarze Schachfiguren

Im Hauptprogramm habe ich Funktionen hinzugefügt, die die Texturen laden. Diese Funktionen erzeugen die Textruen und übergeben diese an die Shader, wo die Texturen dann an die Objekte gebunden werden können.
Die Funktionen werden im Hauptprogramm bei `init()` aufgerufen.

Das Schachbrett sah nun so aus:

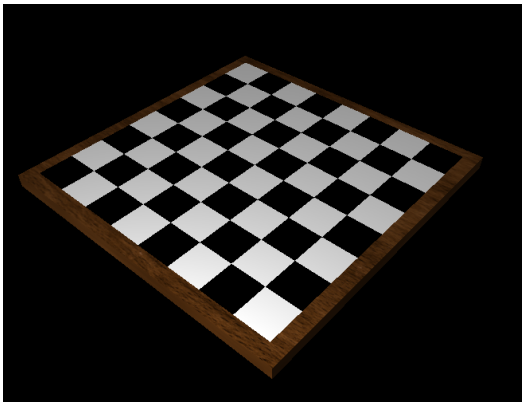

Abbildung 1: Schachbrett ohne Figuren

Modellierung der Schachfiguren

Nachdem das Schachbrett erzeugt wurde, habe ich angefangen die Schachfiguren zu erzeugen.
Dafür habe ich Zylinder erstellt, die ein Metall-Textur haben und diese Figuren habe ich auf das Schachbrett gestellt.
Da die Positionierung auf dem Schachbrett zu Herausforderungen geführt hatte, habe ich die Funktion `getChessboardPosition` hinzugefügt, welche die Koordinaten der Figur

zurückgibt, je nachdem auf welchem Feld sie steht. Dabei ist die Position links unten (0,0) und rechts oben (7,7).

Beleuchtung und Materialien

Da man die Metall-Textur der Schachfiguren schlecht erkennen konnte, habe ich mich dazu entschieden, erst ein mal die Beleuchtungen hinzuzufügen, bevor ich die restlichen Schachfiguren erzeuge.

Ich habe ambientes, diffuses und specular Licht hinzugefügt, bis das Schachbrett folgendermaßen aussah:

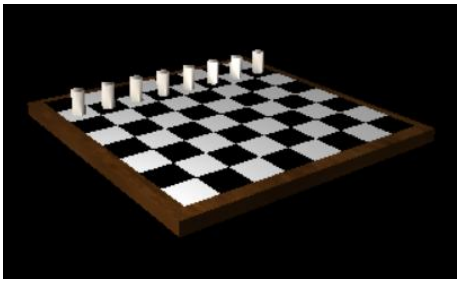

Abbildung 2: Ambientes und Diffuses Licht

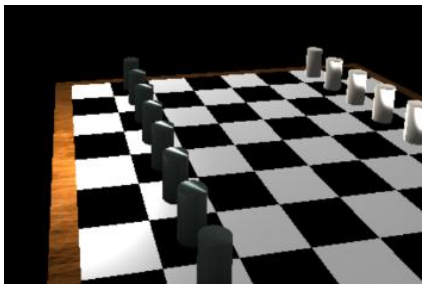

Abbildung 3: Specular Licht

In Abbildung 1 kann man das diffuse Licht gut erkennen und in Abbildung 2 ist das speculare Licht gut an den Reflexionen der Spielfiguren erkennen.

Als ich die Texturen der Spielfiguren einigermaßen gut erkennen konnte, habe ich mich dazu entschieden auch direkt noch ein viertes Licht hinzuzufügen, welches ein Spotlicht ist. Dieses habe ich zuerst von oben auf das Schachbrett scheinen lassen, allerdings hat sich das mit dem specularen Licht gebissen und es war alles ein wenig zu hell. Deswegen habe ich mich dazu entschieden, das Licht einfach von unten auf das Spielbrett scheinen zu lassen.

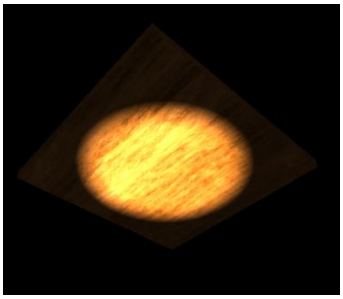

Abbildung 4: Spotlicht

Die Lichtberechnungen erfolgen im Fragment-Shader unter Verwendung des Phong-Beleuchtungsmodells, das realistische Reflexionen ermöglicht.

Modellierung der restlichen Schachfiguren

Nachdem das Licht erfolgreich implementiert wurde, habe ich die restlichen Schachfiguren hinzugefügt. Zuert habe ich eine Textur für die schwarzen Figuren geladen, welche ein dunkles Metall ist.
Durch eine Fallunterscheidung in den draw-Funktionen der Figuren werden diese entweder mit der hellen oder dunkeln Texturen erzeugt.

Nachdem die hellen und dunklen Bauern auf dem Feld standen, habe ich für die restlichen Spielfiguren Kegel erzeugt, die etwas höher und breiter als die Zylinder für die Bauern sind. Bei den Figuren habe ich ebenfalls eine Fallunterscheidung für die jeweilige Textur hinzugefügt.

Animation

Ein Feature des Projekts ist die Animation eines Bauern, der auf Knopfdruck über das Brett bewegt wird. Die Implementation nutzt eine lineare Interpolation zwischen Start- und Zielposition, wobei der Fortschritt der Animation über eine Zeitvariable gesteuert wird. Die Steuerung erfogt über Tastatureingaben, wobei GLUT zur Erfassung der Benutzerinteraktion genutzt wird.

Wenn man die Taste ‘a‘ drückt, bewegt sich der Bauer von Position (1,4) auf (3,4) und diese Bewegung findet in der Funktion `updateAnimation()` statt.

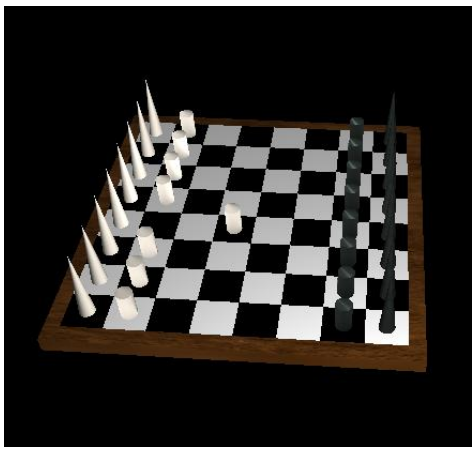

Abbildung 5: Animation

Multi-Viewport-Darstellung

Um die Szene aus verschiedenen Perspektiven betrachten zu könne, wurde ein Multi-Viewport-System implementiert. Die Ansichten umfassen eine perspektivische Hauptkamera, eine Draufsicht, eine Sicht von unten und eine Seitenansicht. Jeder Viewport nutzt eigene Projektionsmatrizen, während die Objekte in allen Ansichten konsistent bleiben.

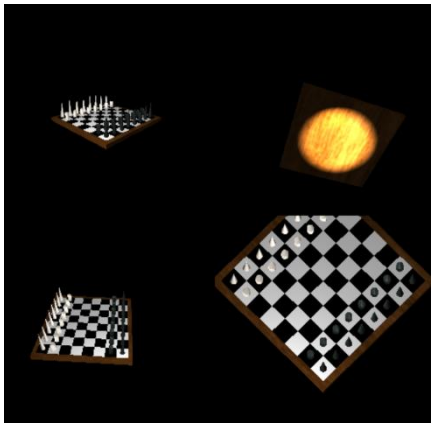

Abbildung 6: Multi-Viewport-Darstellung

Die perspektivische Hauptkamera ist der View unten links, mit ihr kann man weiterhin die Szene mit der Maus aus weiteren Perspektiven ansehen und auch zoomen.

Die `display()`-Funktion unterteilt das Fenster in die vier Bereich. Die Szene wird in jedem Viewport neu gerendert, wobei Shader und Lichtparameter erhalten bleiben.

Installations- und Bedienungsanleitung

Man kann das Programm ausführen, wenn man die Solution s85482_cg1_beleg.sln mit Visual Studio öffnet und den Installation so befolgt, wie wir es im ersten Praktikum des Moduls gemacht haben, mit den Hinzufügen der Bibliotheken. Wenn man dies getan hat, kann man das Main Programm auführen.

Sobald man das Programm ausgeführ hat, öffnet sich das Fenster mit den vier Viewports. Die Viewports oben und unten rechts bewegen sich nicht, außer bei der Animation. Die Kamera des Viewport unten links lässt sich allerdings mit der Maus oder der Tastatur steuern.

Taste	**Aktion**
a	Animation
v/z	Zoom +/-
c	Rückseitenentfernung an/aus
t	Tiefenpuffer an/aus
e	Exit
Pfeiltasten	Kamera rotieren
Maustasten	Kamera bewegen

Die meisten Tasten- und Maussteuerungen habe ich aus den Praktika übernommen, da man so die erzeugten Objekte am besten ansehen kann und schauen, welche Dinge in dem Programm noch verbessert werden müssen.

Probleme

Texturierung und Umrandung des Schachbretts

Die Umrandung des Schachbretts war anfänglich nicht wie gewünscht. Die Erweiterung des 8x8-Musters auf ein 10x10-Muster war zwar keine schlechte Idee, allerdings sah das komisch aus und nicht sehr natürlich.

Lösung: Ich habe einen Quader, den ich unter das Schachbrett gelegt habe auch als Rand verwendet, indem ich den Quader größer erzeugt habe, als das Schachbrett-Muster. Dadurch sah das Schachbrett natürlicher aus.

Schwierigkeiten bei der Positionierung von Figuren

Die Positionierung der Figuren auf dem Schachbrett war schwer erkennbar, da die ganzen Koordinaten allgemein nicht sonderlich gut greifbar sind.

Lösung: Die Funktion getChessboardPosition() wurde implementiert, um die Positionen der Figuren zu berechnen und korrekt auf dem Brett zu platzieren. Diese Funktion gibt die Koordinaten der entsprechenden Felder auf dem Schachbrett zurück.

Aktuelle Probleme

Begrenzte Steuerung durch GLUT

Ich wollte zuerst die Animation etwas ausgefeilter machen, sodass man steuern kann, wo die Figur sich hin bewegt. Zum Beispiel wenn man die Tasten a+1 drückt, bewegt sich die ausgewählte Figur zu A1. Allerdings unterstützt GLUT keine Erkennung gleichzeitiger Eingabe mehrerer Tasten, weshalb ich die Animation statisch gelassen habe.

Lösung: Das Problem wurde nicht wirklich gelöst, da GLUT dies technisch nicht unterstützt. Eine bessere Steuerung könnte man mit einer anderen Bibliothek umgesetzt werden.

Kein Feedback oder Interaktivität für Nutzende

Das Programm bietet keine Möglichkeit, mit den Schachfiguren zu interagieren, z. B. durch Ziehen und Setzen von Figuren.

Lösung: Eine vollständige Schachspiel-Logik oder eine Benutzeroberfläche zur Auswahl und Bewegung von Figuren wäre notwendig, um Interaktivität zu ermöglichen. Das müsste noch entwickelt werden.

Eingeschränkte Perspektiven und Kamera-Steuerung

Die verschiedenen Perspektiven sind festgelegt und bieten keine flexible Steuerung durch den Nutzenden. Desweiteren ist die Bewegung der Kamera durch die Maus sehr unintuitiv und das Bild überschlägt sich manchmal.

Lösung: Man könnte eine größere Auswahl an Perspektiven anbieten.

Schlecht erkennbare Metalltexturen der Figuren

Die Metalltextur der Figuren war schwer erkennbar, was zu einer schlechten visuellen Darstellung führte. Komplett zufrieden bin ich immernoch nicht, da die Mantelflächen der Figuren die Texturen irgendwie nicht gut wiedergeben können. Wenn man eine gemusterte Textur nehmen möchte für die Figuren, lässt sich das Muster kaum erkennen.

Lösung: Man könnte eine andere Berechnung und für die Erzeugung der Schachfiguren verwenden, worauf sich die Texturen besser abbilden lassen.

Ergebnisse

Das entwicklete 3D-Schachbrettprogramm hat alle grundlegenden Anforderungen erfolgreich umgesetzt und demonstriert die praktische Anwendung zentraler Computergrafik-Konzepte. Als wichtigste Ergebnisse lassen sich folgende Punkte hervorheben:

Das Programm zeigt ein vollständiges texturiertes Schachbrett mit realistischer Holzmaserung und metallischen Spielfiguren, die sich deutlich vom Untergrund abheben.

Durch die Implementierung verschiedener Lichtquellen, inklusive Umgebungslicht, diffusem Licht, Glanzlichtern und einem gezielten Spotlight, entsteht eine plastische und ansprechende Darstellung der 3D-Szene.

Ein Erfolg ist die Multi-Viewport-Darstellung, die es ermöglicht, das Schachbrett gleichzeitig aus vier verschiedenen Perspektiven zu betrachten. Die perspektivische Hauptansicht, Draufsicht und Seitenansicht bieten dem Nutzenden unterschiedliche Einblicke in die Szene, während alle Viewports konsistent und synchron bleiben.

Die Animation der Spielfiguren, insbesonder die Bewegung eines Bauern auf Knopfdruck, funktionier flüssig und veranschaulicht die korrekte Implementierung von Transformationsmatrizen und zeitbasierten Interpolationen. Die Steuerung per Tastatur ermöglich es, zwischen verschiedenen Ansichten zu wechseln und bestimmte Rendering-Effekte wie Transparenz oder Tiefenpuffer ein- und auszuschalten.

Zusammenfassend zeigt das Projekt, wie gut sich grundelgende Techniken der 3D-Grafikprgrammierung prakitsch umsetzen lassen, und bildet eine solide Basis für weiterführende Entwicklungen. Die Ergebnisse beweisen die Funktionsfähigkeit des Konzepts und liefern wertvolle Erkenntnisse für zukünftige Projekte in der Computergrafik.

Weitere Bilder:

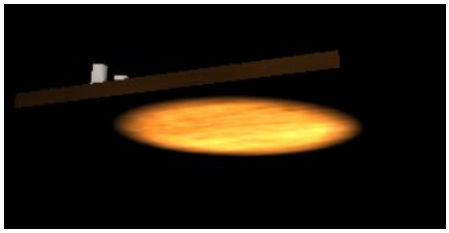

Abbildung 7: ohne ambientes Licht

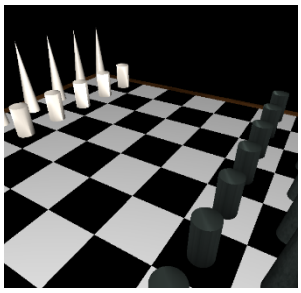

Abbildung 8: Zoom

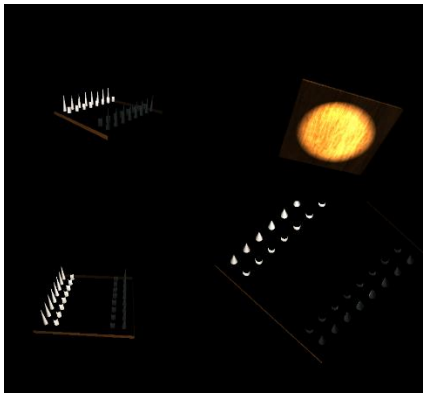

Abbildung 9: Tiefenpuffer

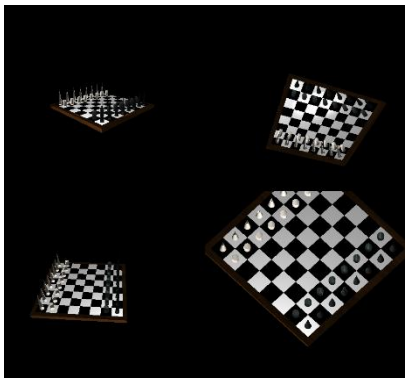

Abbildung 10: Rückseitenentfernung

Quellenverzeichnis

- https://www.opengl.org/resources/libraries/glut/spec3/spec3.html
- https://ambientcg.com/
- https://learnopengl.com/
- Mitschriften aus Vorlesung + Praktikum
- Deepseek(https://www.deepseek.com/) + ChatGPT (https://chatgpt.com/?ref=dotcom)
    - Die generativen KIs habe ich genutzt für die Generierung von Koordinaten, wie z.B. für die Schachbrett-Textur