Informatik
Hauptcampus

H O C H
S C H U L E
T R I E R

Rubik's Cube

Projektdokumentation

Benedikt Pischinger, MatrNr.: 960025

Hausarbeit zur Vorlesung Spielekonsolenprogrammierung

Inhaltsverzeichnis

1

Einleitung

Bei dem vorliegenden Projekt handelt es sich um ein kleines Rubik?s Cube Spiel. Das Spiel wurde in C++ und OpenGL geschrieben, unter Verwendung der Graphics Library GLFW3, der Math Library GLM und irrKlang zum Abspielen von Soundeffekten. Der Würfel selbst kann mit Hilfe der Tastatur gedreht und die sechs Seiten des Wurfels verschoben werden. Der Rubik's Cube ist texturiert und wird mit Hilfe eines Light Shaders eingefärbt. Der ist innerhalb einer Cubemap dargestellt, welche als Skybox dient. Außerdem sind alle Bewegung durch ein Audiofeedback bestätigt und es kann ein freier Kamermodus aktiviert werden.

2

Theoretischer Ansatz

2.1 Rubik's Cube

Es wurde zunächst überlegt, wie der Rubik's Cube am besten umgesetzt werden sollte. Hier wurde die Entscheidung getroffen den Würfel aus 27 kleinen Würfeln (*Cubies*) zusammenzubauen. Ein solcher Cubie befindet sich im Ursprung und dient als Kern des Würfels. An diesen wurde anschließen die sechs einfarbigen Mittel-Cubies angehängt und zu diese wurden wiederum die Kanten- und Ecken-Cubies hinzugefügt. Als nächstes wurde überlegt wie Rotationen an dem Würfel durchgeführt werden. Mathematisch kann die Rotation durch Quaternions durchgeführt werden. Hierzu wird die Achse, um welche rotiert werden soll, der Rotationswinkel und ein zu rotierenden Punkt benötigt.

Achse = (rx, ry, rz), Winkel = φ, P = (px, py, pz), Quaternion = $(i, j, k, Realteil)$

$Q_R = cos(\frac{\varphi}{2}) + sin(\frac{\varphi}{2}) * ((i * rx) + (j * ry) + (k * rz))$

$P_Q = ((i * rx) + (j * ry) + (k * rz))$

$P_R = Q_R * P_Q * Q_R^{-1}$

Diese Rechnung kann dann wiederum auf einen Cubie angewendet werden.

$Cubie_R = Q_R * Cubie_Q * Q_R^{-1}$

Wird also zum Beispiel der gesamte Würfel gedreht, müssen alle Cubies neu berechnet werden. Wird wiederum nur eine Seite verschoben, müssen nur die betroffenen Cubies neu berechnet werden.

2.2 Steuerung

Als nächstes wurde überlegt, wie die Tastatursteuerung am besten umgesetzt wird. Es wurde unter anderem ein Selektor in Betracht gezogen, sodass eine bestimmte

Reihe oder Spalte an Cubies ausgewählt und verschoben werden kann. Es wurde aber die Entscheidung getroffen, dass die bessere Lösung ist einfach einer bestimmten Taste eine Seite zuzuordnen. Auf diese Seite kann der Würfel relativ schnell manipuliert werden und es muss nicht erst vor jedem Zug eine Seite gewählt werden. Somit ergab sich folgende Steuerung.

Escape.....Schließt das Spiel.

F .....Schaltet die freie Kamera an und aus.

L Umschalt.....Gedrückt halten für inverse Rotationen.

Oben.....Dreht den Rubik's Cube um 90° nach oben.
Unten.....Dreht den Rubik's Cube um 90° nach unten.
Links.....Dreht den Rubik's Cube um 90° nach links.
Rechts.....Dreht den Rubik's Cube um 90° nach rechts.

W .....Dreht die obere Seite des Rubik's Cube um 90°.
A .....Dreht die linke Seite des Rubik's Cube um 90°.
S .....Dreht die untere Seite des Rubik's Cube um 90°.
D .....Dreht die rechte Seite des Rubik's Cube um 90°.
Q .....Dreht die vordere Seite des Rubik's Cube um 90°.
E .....Dreht die hintere Seite des Rubik's Cube um 90°.

2.3 Spieldesign

Anfänglich wurde überlegt den Würfel mit den Pfeiltasten frei drehbar zu machen. Hierbei entstanden aber zwei Probleme. Bei dem ersten handelt es sich um ein technisches Problem mit der Steuerung. Eine Taste rotiert eine spezifische Seite. Die W-Taste zum Beispiel rotiert die obere Seite des Würfels um 90° im Uhrzeigersinn. Wäre der Würfel selbst nun frei in alle Richtungen drehbar, so hätten entweder die Achsen und Seiten des Würfels, oder die Tastatursteuerung jedes mal neu definiert werden müssen, wenn der Würfel um mehr als 45° in eine Richtung gedreht wird. Bei dem zweiten Problem handelt es sich um ein Spieldesign bezogenes. Im Normalfall erwartet der Spieler auf eine gewisse Aktion eine bestimmte Reaktion, welche bestmöglich erfüllt werden sollte. Dreht der Spieler nun also den Würfel in eine komische Position (zum Beispiel auf eine Ecke) und versucht anschließend die obere Seite zu rotieren und es wird stattdessen auf einmal die untere Seite rotiert, ist dies eine nicht optimale Reaktion. Aus diesen beiden Gründen wurde die Entscheidung getroffen den Würfel selbst statisch darzustellen und nur Drehungen um 90° nach rechts, links, oben und unten zu erlauben. Auf diese Weise hat der Würfel immer klar definierte Seiten und die Achsen können durch die limitierte Anzahl an Drehmöglichkeiten einfacher definiert und nach Drehung aktualisiert werden. Um

dem Spieler dennoch die Möglichkeit zu geben den Würfel frei von allen Richtungen zu betrachten, besitzt das Spiel einen freien Kameramodus, mit welchem der Spieler sich frei um den Würfel herum bewegen kann. Weiterhin wurde entschieden die Pfeiltasten zu sperren so lange Seiten verschoben werden, um zu verhindern, dass die Achsen durcheinander kommen. Die Eingabe für Seitenrotation kann gebuffert werden und erlaubt somit eine schnelle Manipulation des Würfels, ähnlich zu einem echten Rubik's Cube. Das Spiel enthält außerdem Audioeffekte. Jeder Zug wird also mit einem Audiofeedback bestätigt.

3

Technische Aspekte

3.1 Shader

3.1.1 LightingShader

Für Beleuchtung und Farbdarstellung wurde ein Lighting Shader verwendet. Der Shader selbst besteht aus einem Directional Light, welches aus der Richtung eines Sterns der Skybox kommt, sowie sechs Point Lights, die jeweils an einer Seite des Würfels positioniert wurden. Der Shader überprüft zunächst welche Pixel farbig und welche schwarz sind. Für farbige Pixel wird das Material auf Ambient, Diffuse und Specular gesetzt. Mit Hilfe dieser Einstellungen kann eine bestimmte Oberfläche simuliert werden. Ambient Light ist das passive Licht, welches immer in einer Umgebung vorhanden ist. Diffuse Light simuliert Licht, welches aus einer bestimmten Richtung auf ein Objekt fällt. Specular Light dient dazu ein Material glänzend erscheinen zu lassen. Licht welches auf die Oberfläche scheint wird quasi von dem Material reflektiert und spiegelt sich als leuchtender Punkt auf der Oberfläche wieder. Auf schwarze Pixel wird das Material nur auf Specular gesetzt, um den Rahmen der Würfelseiten einen leichten Glanz zu geben. Mit den richtigen Zahlen kann hier nun die Oberfläche eines echten Rubik's Cubes simuliert werden. Im nächsten Schritt werden die genauen Farbwerte, unter Beeinflussung der gesetzten Lichtwerte, berechnet und nach einer Gammakorrektion in der *FragColor*-Variable gespeichert.

3.1.2 SkyboxShader

Hier handelt es sich um einen einfachen Shader. Es werden lediglich die Farben der eingelesen Skybox Texturen übernommen.

3.2 Header Files

3.2.1 CameraController.h

Dieses File dient zur Implementation der Kamera. Die Kamera bestimmt den Sichtpunkt des Spielers, welcher mit Hilfe einer Reihe von Variablen definiert wird. Außerdem enthält der Header eine Methode zur Verarbeitung von Tastatur und Mausinputs für den freien Kameramodus.

3.2.2 Shader.h

Dieser Header enthält zunächst eine Reihe von Hilfsmethode für die Konfiguration der verwendeten Shadern innerhalb des Programms. Außerdem werden mit diesem File die jeweiligen Instanzen der Shader kreiert. Wie zuvor erwähnt wurde in diesem Projekt ein Lighting und Skybox Shader verwendet. Die Files der beiden Shader werden also mit Hilfe des Header Files eingelesen und in das Programm integriert.

3.2.3 RubiksCube.h

Dieser Header enthält ebenfalls eine Reihe von Hilfsmethoden und statischen Variablen. Hier sind die Farben für die Würfelseiten, die Vertex Daten für den Würfel und die Skybox, sowie die Positionen für die Lichtquellen definiert. Ein weiteres wichtiges Element ist die *faceLookup* Tabelle. Hier sind die aneinander liegenden Farben des Würfels gespeichert. Wenn der gesamte Würfel gedreht wird, kann hier nachgeschaut werden, wie sich die Farben ändern müssen. Wird der Würfel zum Beispiel nach oben gedreht, kann unter Benutzung der Tabelle festgestellt werden, dass die blaue Vorderseite gelb werden muss. Weiterhin wird hier ein Struct für die logische Darstellung von Zügen definiert. Die letzte Methode dient zum Laden von Texture Files für die Texturierung der Würfelseiten.

3.3 Source Files

3.3.1 glad.c

Hierbei handelt es sich um einen Loader für OpenGL. Dieser wurde mit einem Online Tool generiert.

3.3.2 stb_image.cpp

Bei diesem File handelt es sich um einen Image Loader, welcher benutzt wird um die Würfel Texture Files einzulesen. Die Dokumentation hatte empfohlen den Loader mit einem eigenen CPP File zu implementieren.

3.3.3 RubiksCube.cpp

Hier handelt es sich um das Hauptprogramm des Spiels. Es besitzt zunächst eine Cubie Klasse, welche die einzelnen Cubies des Rubik's Cube definiert. Jeder Cubie hat unter anderem sechs Farben für die jeweiligen Seiten, sowie eine bestimmte Position. Die Klasse enthält weiterhin eine *DrawCubie()*-Methode mit welcher die Cubies unter Benutzung der Shader gezeichnet und texturiert werden. Die *Rotate()*-Methode implementiert die zu Beginn erwähnte Matrixtransformation zur Berechnung der Rotationen.
InitialiseOpenGL() kreiert das Fenster, in welches gezeichnet werden soll und konfiguriert diverse OpenGL Settings. Hier werden untere anderem Features wie Depth

Testing, Blending, Face Culling und Multisampling aktiviert.
InitialiseLighting dient zur Konfiguration des Lighting Shaders. Es werden die diverse Werte für das Directional und die Point Lights an das Shader Programm übergeben, um die Würfeloberfläche einzufärben und ein relativ realistisches Material zu simulieren.
InitialiseCamera() setzt die initialen Parameter für die Kameraposition. *LoadTextures()* lädt die jeweiligen Textures Files von den angegebenen Pfaden.
BindVertexData() bindet die Vertices für den Rubik's Cube und die Skybox, welche im Header File definiert sind, an die jeweiligen Vertex Buffer Arrays und Vertex Buffer Object.
BuildRubiksCube() baut den den Würfel aus 27 Cubies auf. Hierzu wird zu erst ein komplett schwarzer Cubie als Kern des Würfels angelegt. Die sechs einfarbigen mittleren Cubies jeder Seite werden dann an diesen Kern gehängt. Anschließen werden die Eck- und Kanten-Cubies um diese mittleren Cubies herum angereiht. Dies wird später wichtig, wenn Seiten rotiert werden.
Draw() zeichnet nun die Skybox und den Cube in den Framebuffer. Diese Methode wird wird in einer *while()*-Schleife innerhalb der *main()*-Methode konstant aufgerufen, um die graphische Darstellung des Würfels immer aktuell zu halten.
ProcessKeyboard() kümmert sich um Tastatureingaben und führt die jeweiligen Züge, welche mit einer bestimmten Taste assoziiert sind, aus.
RotateRubiksCube() aktualisiert die Seiten des Würfels, wenn der gesamte Würfel gedreht wird. Wie zuvor erwähnt wird hier die *faceLookup* Tabelle verwendet. Wenn zum Beispiel der Würfel nach oben gedreht wird und die aktuelle Vorderseite "blau" und die Oberseite "weiß" ist, kann der Tabelle entnommen werden, dass "blau" nun die neue Oberseite und "gelb" die neue Vorderseite ist. Außerdem werden die Achsen entsprechend des Zuges gekippt, um die Ausrichtung des Würfels zu aktualisieren. Am Ende der Methode wird die Würfelrotation noch als *currentMove* weitergegeben.
Die *PerformTurnMove()*-Methode dient zum Drehen einer bestimmten Seite. Wenn zum Beispiel die W-Taste gedrückt wird, wird die *EnqueueTurnMove()*-Methode aufgerufen und die Art des Zuges übergeben. *EnqueueTurnMove()* überprüft dann welcher Mittel-Cubie sich aktuell auf der Seite befindet, welche gedreht werden soll und fügt den Zug dann mit der jeweiligen *faceID* der *moveQueue* hinzu. Diese *moveQueue* wird in der *Update()*-Methode konstant abgearbeitet. Wenn es sich bei dem auszuführenden Zug nun um eine Seitenrotation handelt, wird die Methode für die Seite aufgerufen und der Zug wird aus der *moveQueue* entfernt. Die *PerformTurnMove()*-Methode überprüft nun zunächst welche Cubies rotiert werden müssen. Es werden zunächst alle Cubies, welche sich auf der zu drehenden Seite befinden bestimmt werden, indem die Position jedes Cubies mit der zu drehenden Seite verglichen werden. Mit der *AttachRingCubies()*-Methode werden diese Cubies nun an den Mittel-Cubie der zu drehenden Seite kopiert und der Zug wird nun als *currentMove* weitergereicht. Wenn der Zug nun also von der *Update()*-Methode durchgeführt wird, kann der Mittel-Cubie, sowie die acht äußeren Cubies, welche an den Mittel-Cubie kopiert wurden, über die *Rotate()*-Methode rotiert werden. Die Methode rotiert hierzu zunächst den Mittel-Cubie und iteriert

dann durch alle Cubies, welche an dem Mittel-Cubie hängen und rotiert diese ebenfalls.
Animiert wird der Rubik's Cube mit Hilfe der *Update()*-Methode. Diese läuft ebenfalls in einer *while()*-Schleife in der *main()*-Methode. Muss nun ein Zug durchgeführt werden, wird zunächst ein *animationAngle* aus der *deltaTime* berechnet und aufaddiert. Dieser bestimmt um wie viele Grad die Cubies jede Frame rotiert werden sollen. Wenn der *animationAngle* dann größer als 90° wird, wird die Rotation abgeschlossen und der *currentMove* entfernt. Durch die stückweise Rotation jede Frame kann so eine flüssige Animation erzielt werden.