Informatik
Hauptcampus

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

Rubiks Cube Dokumentation

Florian Wößner

Hausarbeit

Betreuer: Prof. Dr. Christoph Lürig

Trier, 02.02.2025

Inhaltsverzeichnis

1

Einleitung

Das Rubik's Cube Projekt wurde im Rahmen des Moduls Spieleprogrammierung - Vertiefung entwickelt. Ziel des Projekts war es, einen interaktiven Rubik's Cube zu simulieren, der mithilfe von Maus- und Tastatureingaben gesteuert werden kann. Die technische Umsetzung erfolgte unter Verwendung von C++ und OpenGL, wobei mathematische Konzepte wie Quaternionen und Transformationen zur Anwendung kamen.

In dieser Hausarbeit werden die Funktionsweisen der wichtigsten Klassen und die grundlegenden Lösungsideen dokumentiert. Der Fokus liegt auf den entwickelten Ansätzen sowie einer oberflächlichen Beschreibung der Implementierung.

1.1 Steuerung

Der Rubik's Cube wird vollständig durch Maus- und Tastatureingaben gesteuert:

- Rechte Maustaste gedrückt: Durch Ziehen wird der gesamte Cube rotiert.
- Linke Maustaste gedrückt: Ermöglicht das Drehen einzelner Slices (Zeilen und Spalten).
- Leertaste: Setzt den Würfel in den Ausgangszustand zurück.
- Scrollrad: Kann benutzt werden, um mit der Kamera herein- oder herauszuzoomen.

2

Umsetzung

In diesem Kapitel werden die wichtigsten Klassen und deren Funktionsweisen erläutert. Jede Klasse wird kurz beschrieben, wobei der Fokus auf ihrer Rolle im Gesamtsystem und ihrer Implementierung liegt.

2.1 Klassenübersicht

Das Klassendiagramm zeigt die Beziehungen der wichtigsten Komponenten des Rubik's Cube Projekts. Die Basis bildet das Interface `IGameInterface`, das von `RubiksGameInterface` implementiert wird. Diese zentrale Klasse koordiniert das Zusammenspiel von Eingaben (`InputSystem`, unterstützt durch `KeyboardObserver`), der Logik (`RubiksCube`) und der Darstellung (`CubieRenderer`).

Der Würfel selbst besteht aus `Cubie`-Objekten, die von `RubiksCube` organisiert werden. Zur Unterstützung von Rendering-Aufgaben wird die Klasse `ShaderUtil` genutzt, welche die Shader-Verwaltung übernimmt. Abbildung 2.1 verdeutlicht die Beziehungen der Klassen.

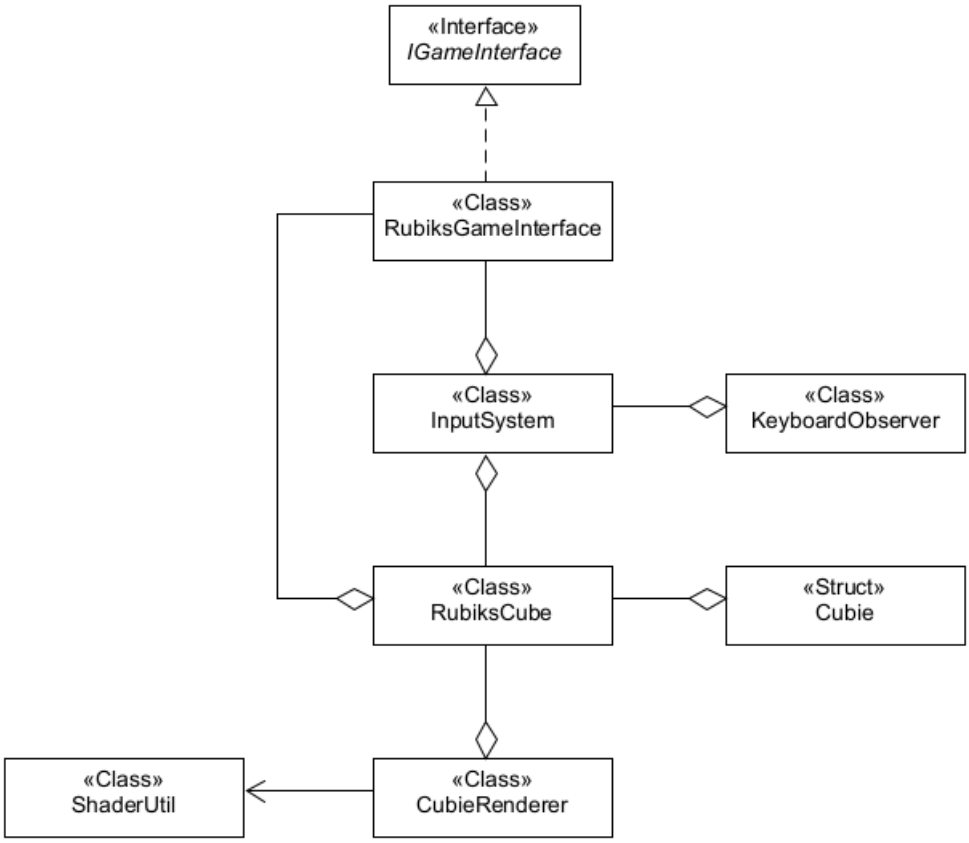

Abbildung 2.1: Vereinfachtes Klassendiagramm

2.2 main.cpp

Der Code in `main.cpp` dient als Einstiegspunkt des Projekts. Hier wird *GLFW* initialisiert, ein Fenster erstellt und die *GLEW*-Bibliothek geladen, um moderne OpenGL-Funktionen nutzen zu können. Ein Objekt vom Typ `RubiksGameInterface` wird erstellt und als aktuelle Schnittstelle festgelegt. Der Game-Loop verarbeitet Benutzereingaben und rendert das Fenster. Ein zusätzlicher Mechanismus stellt sicher, dass das Programm bei minimiertem Fenster nicht abstürzt. Nach dem Schließen des Fensters werden mit der Methode `ShutdownSystem()` alle Ressourcen freigegeben und *GLFW* ordnungsgemäß beendet.

2.3 RubiksGameInterface und IGameInterface

Die Klasse `RubiksGameInterface` implementiert das Interface `IGameInterface` und stellt die zentrale Schnittstelle für die Verwaltung des Spiels dar. Die Methode `Initialize(GLFWwindow* window)` kümmert sich um die Initialisierung des Spielfensters und des Eingabesystems. Dabei wird das Fensterobjekt in `m_window` gespeichert und das Eingabesystem durch den Aufruf von `m_input.Initialize(window)` eingerichtet. Der Rubik's Cube wird durch `m_rubiksCube.Initialize(*this)` initialisiert. Zusätzlich wird mit `m_input.ObserverKey(GLFW_KEY_SPACE)` ein Observer für die Leertaste erstellt.

Die Methode `Render(float aspectRatio)` ist für das Zeichnen des Fensters und des Rubik's Cube verantwortlich. Sie berechnet bei Bedarf die Projektions- und View-Matrizen neu, wenn die Kameraposition oder das Seitenverhältnis des Fensters sich geändert haben. Anschließend wird die Renderlogik auf den Rubik's Cube angewendet, um dessen aktuellen Zustand mit dem Aufruf von `m_rubiksCube.Render(m_projection * m_view)` darzustellen.

Die Methode `Update(double deltaTime)` verarbeitet die Benutzereingaben und aktualisiert den Rubik's Cube. Wenn die Leertaste gedrückt wird, erfolgt ein Zurücksetzen des Rubik's Cubes, andernfalls wird `m_rubiksCube.Update(*this)` aufgerufen. Änderungen der Kameraentfernung, die durch das Mausrad bedingt sind, werden in Echtzeit aktualisiert, indem der Wert von `m_CameraDistance` angepasst wird. Der Kameraabstand wird durch `if`-Abfragen auf bestimmte Werte begrenzt. Die Methode `ClearResources()` gibt die Ressourcen des `m_rubiksCube`-Objektes frei.

Zusätzlich verfügt die Klasse über die Methode `QueueMatrixRecalculation()`, die es ermöglicht, das Neuberechnen der Projektions- und View-Matrizen gezielt anzustoßen. Weitere Getter-Methoden erlauben den Zugriff auf `m_deltaTime` und `m_input`.

Abbildung 2.2 zeigt das Klassendiagramm zu diesem Abschnitt.

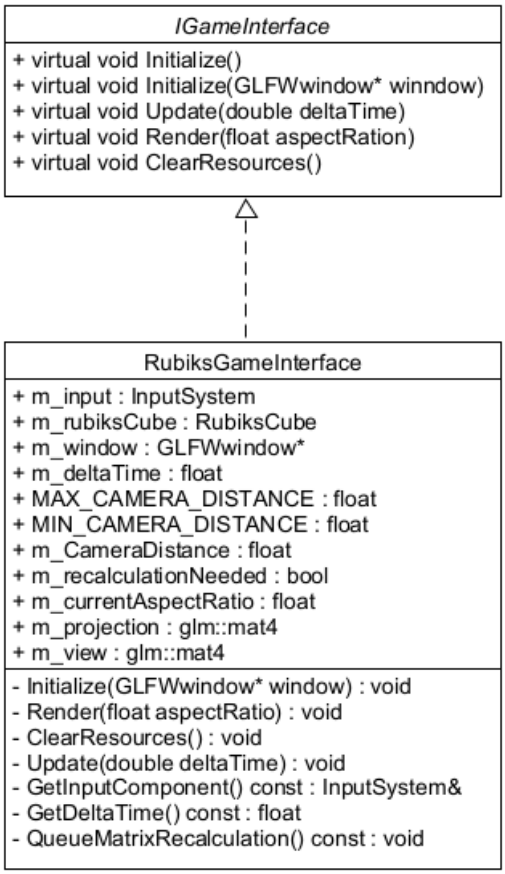

Abbildung 2.2: Klassendiagramm RubiksGameInterface

2.4 InputSystem und KeyboardObserver

2.5 CubieRenderer und ShaderUtil

2.6 RubiksCube und Cubie

3

Zusammenfassung