Praxisprojektbericht

3D-Code

Autor: Dennis Goßler

Matrikel-Nr.: 11140150

Adresse: Oswald-Greb-Str. 7

42859 Remscheid

[dennis.gossler@smail.th-koeln.de](mailto:dennis.gossler@smail.th-koeln.de)

eingereicht bei: Alexander Dobrynin

Remscheid, 01.01.1001

Abstract

Inhaltsverzeichnis

[Abstract 2](#_Toc89791860)

[Inhaltsverzeichnis 3](#_Toc89791861)

[Tabellenverzeichnis 4](#_Toc89791862)

[Abbildungsverzeichnis 5](#_Toc89791863)

[1 Einleitung 6](#_Toc89791864)

[1.1 Relevanz 6](#_Toc89791865)

[1.2 Zielsetzung 6](#_Toc89791866)

[1.3 Planung 6](#_Toc89791867)

[1.3.1 Definierung der Programmiersprache 6](#_Toc89791868)

[1.3.2 Planung der Aufbereitung des 3D Outputs 6](#_Toc89791869)

[1.3.3 Planung des Thesenüberprüfung 7](#_Toc89791870)

[1.4 Grundaufbau 7](#_Toc89791871)

[1.5 Veranschaulichung 7](#_Toc89791872)

[2 Hauptteil 8](#_Toc89791873)

[2.1 Anpassung/ Erweiterung der Sprache 8](#_Toc89791874)

[2.1.1 Feature Auflistung 8](#_Toc89791875)

[2.1.2 Einfache Typen 8](#_Toc89791876)

[2.1.3 Objektspeicherung 8](#_Toc89791877)

[2.1.4 Interne Funktionen 8](#_Toc89791878)

[2.1.5 Generische Klassen/ Funktionen 8](#_Toc89791879)

[2.1.6 Dateiverwaltung 8](#_Toc89791880)

[2.2 Zusammenführung und Rendering 8](#_Toc89791881)

[2.3 Implementierung der Anwendungsszenarien 8](#_Toc89791882)

[2.4 Vergleich zu Alternativen 8](#_Toc89791883)

[3 Fazit 9](#_Toc89791884)

[Literaturverzeichnis 10](#_Toc89791885)

[Anhang 11](#_Toc89791886)

Tabellenverzeichnis

Abbildungsverzeichnis

# Einleitung

Das Praxisprojekt 3D-Code befasst sich mit dem Thema der schnellen Ausgabe von 3D Objekten, um gewisse Problemstellungen mit wenig Aufwand in einer dreidimensionalen Umgebung zu projizieren. Hierfür wird eine eigene objektorientiere Programmiersprache entwickelt, die es erlaubt Objekte für den Renderprozess zu erstellen. Die Programmiersprache besitzt generische Funktionen und Klassen, welche es ermöglicht Objekte jeder Art in Kollektionen zu speichern.

## Relevanz

Das sogenannte „Debuggen“ von Code ist seit den Anfängen der Computergeschichte ein umfassendes und wichtiges Thema. Gerade Anfängern fällt es sehr schwer Algorithmen auf Fehlern zu untersuchen und diese zu beheben. Da eine Visualisierung meistens nur durch Ausgabe von Zeichenketten erfolgt. Vor allem dreidimensionale Problemstellung lassen sich nur schwer mit einem 2-dimensionalen Output visualisieren.

Im Zuge dessen ist diese Projektidee entstanden, welche es ermöglicht mit nur minimalem Aufwand ein dreidimensionaler Output von farbigen Objekten zu erzeugen.

## Zielsetzung

Ziel des Praxisprojektes ist herauszufinden, ob eine solche Art der Programmierung Vorteile gegenüber dem konventionellen Ansatz zeigt. Außerdem sind Probleme und aufkommende Fragestellungen aufzulisten. Um dies zu ermitteln ist eine objektorientiere Programmiersprache zu entwerfen/ entwickeln die es ermöglicht einen dreidimensionalen Output zu erzeugen. Dieser sollte nur mit einer minimalen Anzahl von Zeilen Code auskommen. Dennoch ist es wichtig, dass die entworfene Programmiersprache einfach zu lernen und zu verstehen ist.

## Planung

Die Planung ist in mehreren Schritten zerlegt. Je Meilenstein ist das weitere Vorgehen agil geplant. Das Praxisprojekt ist von einer Person geplant, entwickelt und getestet. Außerdem ist das gesamte Projekt ist in vier Hauptphasen unterteilt. Da dieses Projekt auf zwei weiteren bereits entwickelten Projekten aufbaut, wird oftmals nur von einer Anpassung oder Ergänzung berichtet. Das Grundmodell, bestehend aus den zwei erwähnten Projekten, dies Thema wird im Kapitel „“Grundaufbau““ ausführlich behandelt.

### Definierung der Programmiersprache

Die erste Phase besteht zum Großteil aus der Entwicklung der Programmiersprache. Hierbei werden alle Features definiert und getestet. Da das Grundgerüst der zu entwickelnden Programmiersprache in der Sprache Kotlin geschrieben wurde, wird diese Programmiersprache weiterhin beibehalten. Ein wesentlicher Bestandteil ist die Planung der Umsetzung von einer prozeduralen Programmiersprache in die gewünschte objektorientieren Programmiersprache.

### Planung der Aufbereitung des 3D Outputs

Das Framework des dreidimensionalen Outputs ist die „Lightweight Java Game Library“ in kurz „LWJGL“. Da in einem anderen Praktikum bereits eine Grundstruktur in Kotlin entstanden, gilt es diese anzupassen.

### Planung des Thesenüberprüfung

Um zu überprüfen, ob die „User Experience“ einen wesentlichen Vorteil gegenüber der herkömmlichen Art bietet, werden verschiedene Algorithmen implementiert. Diese sollen zeigen welche Vorteile, die entwickelte Sprache bietet.

## Grundaufbau

Der Grundaufbau besteht aus zwei fertiggestellten Projekten, die als Grundlage dienen.

Das erste Projekt ist im Rahmen des Moduls „Programmiersprachen und Compilerbau“ entstanden. Dieses Projekt hatte das Ziel eine Programmiersprache ähnlich wie „C“ von Grund auf zu entwickeln. Hierfür musste ein Lexer der den Quellcode in logische zusammengehörige Einheiten zerlegt entwickelt werden. Darauffolgend wurden diese Tokens durch einen Parser in einen Abstract syntax tree, kurz „AST“, umgewandelt. Dieser Teil des Modulprojektes wurde von dem Studenten Lukas Momberg entwickelt. Weiterdessen wurde der „AST“ evaluiert und auf seine Typisierung überprüft. *Projektlink*

Der Ablauf gilt, wie in folgender Abbildung veranschaulicht, trotz Änderung der Programmiersprachenart nicht anzupassen.

***ABBILDUNG***

Das zweite Projekt ist im Modul „Computergrafik und Animation“ entstanden. In diesem Modul wurde eine dreidimensionale Weltraumsimulation geschaffen, welche es ermöglicht verschiedene Sonnensysteme zu generieren und diese zu animieren. Das Projekt wurde in Zusammenarbeit mit Frau Anastasia Chouliaras erstellt. *Projektlink*.

Es ist geplant nur die grobe Struktur des Ladens und Rendern der Objekte für dieses Projekt zu verwenden. Der Rest gilt abzuändern oder zu verwerfen.

## Veranschaulichung

Nach der Fertigstellung der Programmierumgebung sind verschiedene Veranschaulichungen zu entwickelt. Als Inspiration wurde zum Beispiel das YouTube Video von „TimoBingmann“ Quelle einfügen genutzt, welches 15 verschiede Sortieralgorithmen grafisch darstellt. Ein weiteres Beispiel ist die Visualisierung von Perlin Noise unter verschiedenen Parametern.

# Hauptteil

Der Hauptteil beschäftigt sich mit der Erstellung und den Problemstellungen der einzelnen Implementierungsphasen.

## Anpassung/ Erweiterung der Sprache

Wie im Kapitle LINK beschrieben, baut das Projekt auf der eigen entwickelten Programmiersprache „C--“ auf. Dieser Abschnitt beschäftigt sich mit der Anpassung und den Erweiterungen, die vorgenommen wurden.

### Feature Auflistung

Folgende Aufzählung beinhaltet alle Features, die die „3D Code“ Sprache besitzt, welche nicht in der Sprache „C--“ schon enthalten waren.

* Variablen besitzen kein Anfangszeichen, wie z.B. „$“
* Semikolon optional
* Überladene Funktionen
* Fließkommazahl
* Division
* Additionszuweisungsoperator
* For-Schleife
* Klassen definieren
* Überladene Konstruktoren
* Nullwerte
* Generische Funktionen/ Klassen
* Arrays
* Inkludierung von Dateien
* Listen / Verlinkte Listen
* Tulpe / Vector3f
* Mathebibliothek

### Umgebung

Die Umgebung speichert und verwaltet globale und lokale Variablen. Jede Datei besitzt eine globale Umgebung. Weiterdessen besitzt jede Codeabschnitt, wie zum Beispiel eine Funktion oder ein Block eine eigene Umgebung. Ein Block in einer Funktion besitzt somit eine Teilmenge der Funktionsumgebung.

Jedes Laufzeitobjekt besitzt dementsprechend auch eine eigene Umgebung, die bei Funktionsaufrufen genutzt werden kann.

### Variablen Einfache Typen / Objekte

Variablen mit einfachen Typen wie zum Beispiel: Integer, Fließkommerzahlen und Strings werden als Konstante Werte in der aktuellen Umgebung gespeichert. Im Gegensatz zu Objekten, die mit einem Konstruktor erzeugt werden. In diesem Fall wird nur ein Pointer in der Variabel abgelegt. Dies hat zur Folge das mehrere Objekte auf dasselbe Objekt und dessen Inhalt zeigen könnten.



Abbildung 1 [Variablen Zuweisung]

Der Code in der oberen Abbildung würde folgendes ausgeben:

*Ausgabe:*

*6 | 7*

*b | b*

### Überlagerte Funktionen

Überladene Funktionen besitzen den gleichen Namen aber unterschiedliche Parameter. Es werden alle Funktionen einer Datei in Form einer Map, bestehend aus dem Namen und dessen Deklaration, gespeichert.

Wenn nun eine Funktion aufgerufen wird, die mehrfach mit dem gleichen Namen in einer Datei vorhanden ist, muss nun der Evaluator die Funktion herausfiltern, die die gleichen Parameter besitzt, wie die aufgerufene.

### Generische Klassen/ Funktionen

### Dateiverwaltung

### Interne Funktionen

## Zusammenführung und Rendering

## Implementierung der Anwendungsszenarien

## Vergleich zu Alternativen

# Fazit

Literaturverzeichnis

Anhang