Praxisprojektbericht
3D-Code

Autor:	Dennis Goßler
Matrikel-Nr.:	11140150
Adresse:	Oswald-Greb-Str. 7
	42859 Remscheid
	dennis.gossler@smail.th-koeln.de

eingereicht bei: Alexander Dobrynin

Remscheid, 16.12.2021

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Das Praxisprojekt 3D-Code befasst sich mit dem Thema der schnellen Ausgabe von 3D Objekten, um die Ergebnisse von 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 Containern zu speichern.

1.1 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 Fehler zu untersuchen und diese zu beheben. Da eine Visualisierung meistens nur durch Ausgabe von Zeichenketten erfolgt, können vor allem dreidimensionale Problemstellung sich nur schwer visualisieren lassen.

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

1.2 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, durch das Aufrufen einer eigenentwickelten Applikation, zu erzeugen. Die Sprache 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.

1.3 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 [1.4 Grundaufbau der Anwendung S.6] ausführlich behandelt.

1.3.1 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.

1.3.2 Planung der Aufbereitung des 3D Outputs

Das Framework des dreidimensionalen Outputs ist die *Lightweight Java Game Library* in kurz *LWJGL* [Q10 Lightweight Java Game Library S.17]. Da in einem anderen Praktikum bereits eine Grundstruktur in Kotlin entstanden ist, gilt es diese anzupassen.

1.3.3 Planung der Thesenüberprüfung

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

1.4 Grundaufbau der Anwendung

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. [Q7 Projekt PuC_C-- S.16]

Der Ablauf gilt, wie in folgender Abbildung veranschaulicht anzupassen.

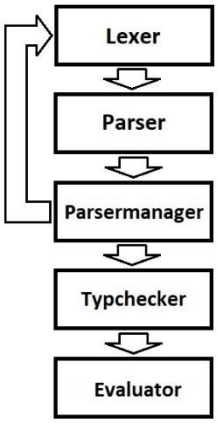

Abbildung 1 [Compileraufbau]

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. [Q3 Projekt OuterSpace S.16]

Es ist geplant nur die grobe Struktur des Ladens und Rendern der Objekte für dieses Projekt zu verwenden.

1.5 Veranschaulichung

Nach der Fertigstellung der Programmierumgebung sind verschiedene Veranschaulichungen zu entwickelt. Als Inspiration wurde zum Beispiel das YouTube Video von *TimoBingmann* [Q2 15 Sorting Algorithms in 6 Minutes S.16] genutzt, welches 15 verschiede Sortieralgorithmen grafisch darstellt. Ein weiteres Beispiel ist die Visualisierung von Perlin Noise unter verschiedenen Parametern.

2 Hauptteil

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

2.1 Anpassung und Erweiterung der Sprache

Wie im Kapitle [1.4 Grundaufbau der Anwendung S.6] beschrieben, baut das Projekt auf der eigen entwickelten Programmiersprache *C*-- auf. Dieser Abschnitt beschäftigt sich mit der Anpassung und den Erweiterungen der Sprache.

2.1.1 Feature Auflistung

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

- **Syntax**
  - Variablen besitzen kein Anfangszeichen, wie z.B. ‘$‘
  - Semikolon optional
  - Inkludierung von Dateien
  - Additionszuweisungsoperator
- **Schleifen**
  - For-Schleife
- **Definitionen**
  - Klassen definieren
  - Private Klassenattribute/-funktionen
  - Überladene Funktionen
  - Überladene Konstruktoren
  - Generische Funktionen/ Klassen
- **Typen**
  - Float
  - Nullwerte
  - Arrays
  - Listen / Verlinkte Listen
  - Pair, Tripple und Vector3f
- **Fließkommazahl Arithmetik**
  - Division
- **Sonstiges**
  - Mathebibliothek

2.1.2 Kontext

Der Kontext speichert und verwaltet globale und lokale Variablen. Jede Datei besitzt einen globalen Kontext. Weiterdessen besitzt jeder Codeabschnitt, wie zum Beispiel eine Funktion oder ein Block einen eigenen Kontext. Ein Block in einer Funktion besitzt somit eine Teilmenge des Funktionskontext.

Jedes Laufzeitobjekt besitzt dementsprechend auch einen eigenen Kontext, der bei Funktionsaufrufen genutzt werden kann.

2.1.3 Variablen: Einfache Typen und Objekte

Variablen mit einfachen Typen wie zum Beispiel: Integer, Fließkommerzahlen und Strings werden als konstante Werte in dem aktuellen Kontext gespeichert. Im Gegensatz zu Objekten, die mit einem Konstruktor erzeugt und als *DynamicValue* gespeichert werden. In diesem Fall wird nur eine Referenz in der Variablen abgelegt. Dies hat zur Folge, dass mehrere Referenzen auf dasselbe Objekt, und dessen Inhalt, zeigen könnten.

Die Abbildung [Abbildung 2 [Variablen Zuweisung] S.8] zeigt Beispiele der Zuweisung und Verwendung der verschiedenen Variablentypen.

```
//Code
/*
Objektdefinitionen
…
*/

Void Main(){
        Int x1 = 5
        Int x2 = x1
        Obj o1 = Obj()
        Obj o2 = o1

        x1 += 1
        x2 += 2

        o1.attribute = „a“
        o2.attribute = „b“

        Println(x1.ToString() + „ | “ + x2.ToString())
        Println(o1.attribute + „ | “ + o2.attribute)
}
```

Abbildung 2 [Variablen Zuweisung]

Der Code in der oberen Abbildung, würde folgendes ausgeben:

Ausgabe:

6 | 7

b | b

2.1.4 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 existiert, muss nun der *Evaluator* die Funktion herausfiltern, die die gleichen Parameter besitzt, wie die aufgerufene.

2.1.5 Generische Klassen und Funktionen

Die Sprache *3DCode* enthält die Möglichkeit Klassen oder Funktionen mit generischen Eigenschaften zu deklarieren. Hierfür ist der *Type Checker* besonders angepasst. Dieser muss zunächst prüfen, ob die generischen Typen richtig verwendet werden. Der *Evaluator* ist dementsprechend auch angepasst, indem er nun bei überlagerten Funktionen auch den generischen Anteil berücksichtigen muss.

2.1.6 Dateiverwaltung

Da es möglich ist in mehrere Dateien zu programmieren und diese sich jeweils inkludieren können, muss vor dem Evaluieren und der Typüberprüfung eine Verlinkung stattfinden. Hierfür wurde ein neues Modul namens *Parsermanager* eingeführt. Dieser durchläuft jeweils eine Datei und bei einem *include* wird die jeweilige Datei auch überprüft. Somit werden alle Dateien einmalig rekursiv überprüft und miteinander verlinkt.

2.1.7 Interne Funktionen

Da es oftmals sehr aufwendig oder unmöglich ist manche Features in der Sprache zu entwickeln, wird auf gewisse *Kotinfeatures* zurückgegriffen.

Ein wichtiges Beispiel wäre hierfür das Array. Diese Klasse ist in der *AST* Syntax vorhanden und wird bei einem Include durch den *Parsermanager* hinzugefügt. Beim Aufruf dessen Konstruktors, wird wiederrum zusätzlich eine integrierte Funktion namens *_integratedFunctionSetArray* aufgerufen. Diese erzeugt ein *DynamicValue* Objekt, welches ein *Kotlin* Array enthält.

2.2 Zusammenführung und Rendering

Dieser Abschnitt erläutert wie die Schnittstelle zwischen Programmiersprache und der Anzeige von den dreidimensionalen Objekten funktioniert.

2.2.1 Initialisierung

Beim Starten der Anwendung wird eine Datei namens *App* gesucht und alle Inhalte geparst. Die Datei und alle Verbunddateien werden, in der *AST Form*, zwischengespeichert und nach der erfolgreichen Typüberprüfung evaluiert.

2.2.2 Update und OnKey

Wenn eine Tastatureingabe stattfindet, wird jedes Mal die Funktion *OnKey*, mit dem vorherigen Kontext, evaluiert. Durch dieses Vorgehen kann die statische dreidimensionale Welt animiert werden. Dafür können die Eigenschafften der erschaffenen Objekte angepasst werden.

Nach dem *Update/ OnKey* Event werden die Objekte jeweils in ein *Renderable* umgewandelt und dem Nutzer ausgegeben.

2.2.3 Umwandlung von Objekten

Im Umwandlungsprozess werden alle Eigenschaften des erstellten Objektes so umgewandelt, dass Sie kompatible mit der erstellten 3D-Engine sind. Hierfür werden Attribute wie die Skalierung, Rotation, Position und Einfärbung genutzt.

2.3 Einfaches Beispiel

Das folgende Beispiel soll veranschaulichen, wie man ein dreidimensionales Objekt erschafft und dieses Transformiert. Dafür wird ein Stern geladen und dieser gelb eingefärbt. In jedem Updatezyklus wird dieser um 1° in seiner Y-Achse gedreht.

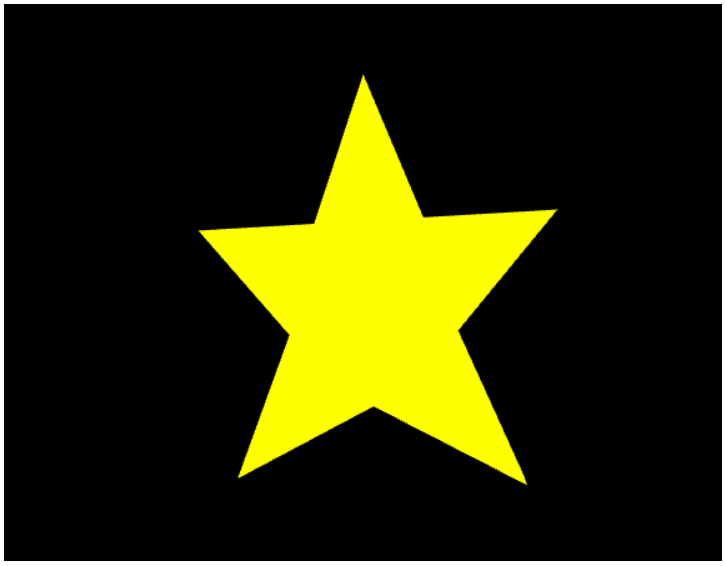

Abbildung 3 [Einfaches Beispiel]

Folgender Code wurde für die obere Ausgabe genutzt.

```
include "lib.List"
include "lib.Vector3f"
include "Object"

List objects<Object> = List<Object>(1)

Void Init(){
        Object o = Object("assets/objects/star.obj")
        o.color = Vector3f(255.0,255.0,0.0)
        o.position = Vector3f(0.0,0.0,-30.0)
        objects.Add(o)
}

Void Update(Float deltaTime, Float time){
        Object o = objects.Get(0)
        o.rotation = Vector3f(0.0,o.rotation.y + 1.0,0.0)
}
```

2.4 Implementierung der Anwendungsszenarien

In diesem Abschnitt werden die entwickelten Beispiele erklärt und dessen Ausgabe erläutert. Weitergehend werden die Erkenntnisse, die bei der Programmierung entstanden sind, dargestellt und an Ihren Beispielen erläutert.

2.4.1 Sortieralgorithmen

Dieses Beispiel soll verdeutlichen, wie Veränderungen von bereits erschaffenen Objekten in der Programmiersprache realisiert werden könnte. Hierfür sind drei Sortieralgorithmen ausgewählt, die das Szenario auf verschiedensten Arten abbilden.

Dafür ist eine Klasse entstanden, namens *SortObj*, die sich bei Ihrer Konstruktion selbst einen *Score*, eine zugehörige Farbe und eine Skalierung zuweist. Sodass beim Sortieren nur die Position verändert werden muss.

Der *Bubblesort* Algorithmus wurde mittels 26 Zeilen Code implementiert, wobei nur zwei Zeilen für das Aktualisieren der Position benötigt werden. [A4 Bubblesortfunktion S.19]

Bei dem zweiten Sortieralgorithmus handelt sich es um *Mergesort*. Dieser Algorithmus sortiert die Elemente zunächst in kleinen Paketen, wenn zwei Pakete der gleichen Größe sortiert wurden, werden diese zu einem größeren Paket zusammengefasst, welches dementsprechend auch sortiert wird. Dieses Verfahren wird so lange wiederholt, bis ein Paket der Größe der sortieren Menge vorliegt. Der Algorithmus besitzt eine Komplexität von O(n*log(n)).[1] Der implementierte Algorithmus [A5 Mergesortfunktion S.20] ist allerdings auf Sortiergrößen von glatten zweier Potenzen limitiert.

Der dritte Algorithmus ist eine Abwandlung von *Combsort*. Diese Implantation vergleicht wie der *Bubblesortalgorithmus* Elemente, die sich nebeneinander befinden, und vertauscht diese, wenn nötig. Der große Unterschied ist, dass dieser Algorithmus das größere Element bis zur nächsten schleifen Iteration nichtmehr verschiebt. [A6 Combosortfunktion S.21]

Die Ausführung, von den oben genannten drei Algorithmen, ist per Videoform [Q4 3DCode – 3 Sorting Algorithms S.16] abrufbar.

2.4.2 Perlin Noise

Das zweite Beispiel beschäftigt sich mit dem *Perlin Noise* Algorithmus. Dieser Algorithmus soll zeigen, wie sich komplexe Formen konstruieren und sich in der dreidimensionalen Umgebung betrachteten lassen.

Der Algorithmus lässt sich wie folgt beschreiben:„*Perlin Noise ist eine Rauschfunktion auf der Basis von seudozufälligen Gradientenwerten an Gitterpunkten. […] Perlin Noise wird häufig in der Bildsynthese eingesetzt, um natürliche Phänomene wie Wolken, Landschaftstopologien oder Wasser zu simulieren.“* [2]

Q1 [1] [Q8 OpenGL Webseite

Khronos Group. (o. D.). OpenGL - The Industry Standard for High Performance Graphics. Copyright (c) 2000 - 2021 Khronos Group. Abgerufen am 16. Dezember 2021, von https://www.opengl.org/

Merge Sort Algorithm], S.19

Q2 [2] [Q11 Matplotlib documentation

Matplotlib development team. (o. D.). Matplotlib documentation — Matplotlib 3.5.1 documentation. Matplotlib 3.5.1 documentation. Abgerufen am 16. Dezember 2021, von https://matplotlib.org/stable/index.html

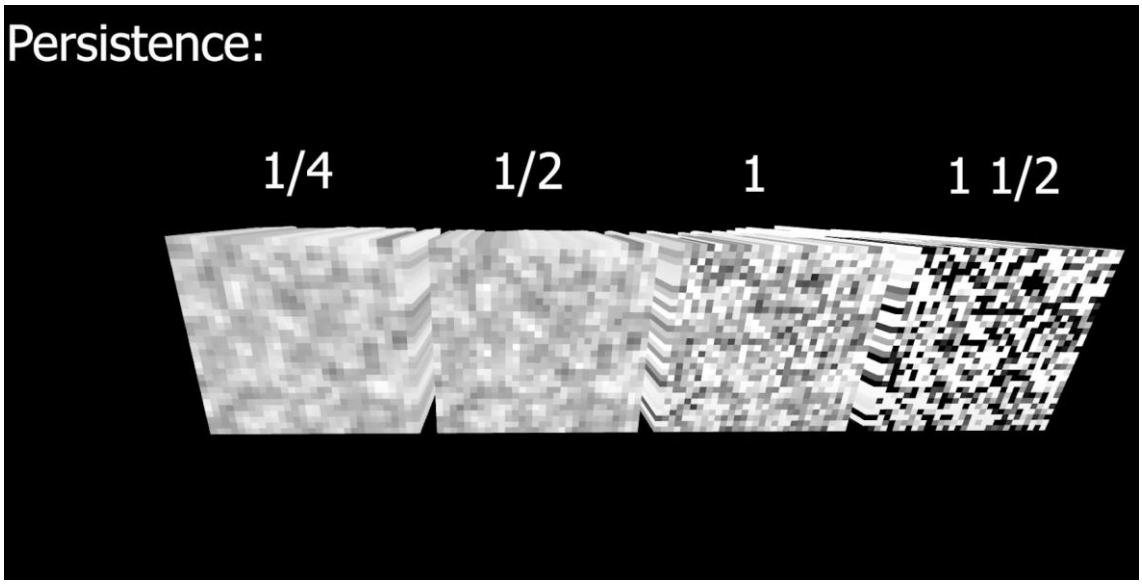

Abbildung 4 [Perlin Noise Output]

Die Abbildung [Abbildung 4 [Perlin Noise Output]] zeigt die Ausgabe des Algorithmus. Hierbei sind vier verschiedene Werte für die Persistenz gewählt. Das Video [Q5 3DCode – Perlin Noise S.16] veranschaulicht zusätzlich die dreidimensionale Ansicht, die aus einem anderen Blickwinkel hervorgeht.

Die Implementierung und das generelle Verständnis gehen aus Quelle [Q1 Perlin Noise Web.Archive.Org S.16] hervor. Der entwickelte Code ist im Anhang zu finden [A7 Codebeispiel Link [Perlin Noise] S.22].

2.4.3 Random Walk

Der Random Walk ist als drittes Beispiel gewählt, da er ermöglicht eine stetig verändernde Welt dazustellen.

Bei jedem Aufruf der Updatefunktion wird ein neuer Block generiert. Dieser kann an jeder Seite und an jeder Ecke seines Vorgängers generiert werden. Somit wird bei jedem Aufruf eine von 26 möglichen zufälligen Position gewählt.[3]

Q3 OpenGL Wikipedia

Seite „OpenGL“. In: Wikipedia – Die freie Enzyklopädie. Bearbeitungsstand: 8. November 2021, 14:54 UTC.
URL: https://de.wikipedia.org/w/index.php?title=OpenGL&oldid=217099756 (Abgerufen : 16. Dezember 2021, 22:44 UTC)

Perlin-Noise Wikipedia], siehe zusätzlich [Q1 Perlin Noise Web.Archive.Org]

[3] Zugehöriger Code: [A8 Codebeispiel Link [Random Walk 3D] S.21]

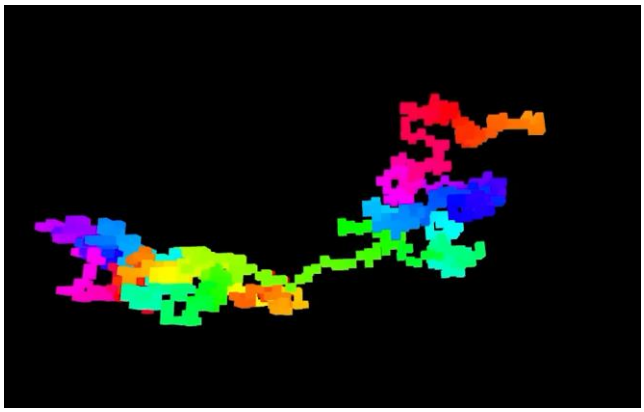

Abbildung 5 [Random Walk 3D]

Die Abbildung [Abbildung 5 [Random Walk 3D] S.13] stellt die Ausgabe des oben beschrieben Algorithmus im Zeitpunkt t+2min 10s da. Die animierte Ausgabe ist unter [Q6 3DCode – Random walk S.16] zu finden.

2.5 Testfälle

Um nach Veränderungen des Programmcodes weitergehend sicherzustellen, dass die Veränderung keine Fehler auslösen, sind insgesamt 170 Tests entstanden. Die Tests decken einen Großteil der Funktionalitäten der Programmiersprache ab. Über folgenden Link sind alle Tests abrufbar [A2 Testfälle S.18].

2.6 Vergleich zu Alternativen

Es befinden sich mehrere Tools auf dem freien Markt, um dreidimensionale Objekte zu rendern und diese anzuzeigen. Die Unterkapitel erklären drei andere Ansätze und listen Vor und -Nachteile auf.

2.6.1 OpenGL

„OpenGL […] ist eine Spezifikation einer plattform- und programmiersprachenübergreifenden Programmierschnittstelle (API) zur Entwicklung von 2D- und 3D-Computergrafikanwendungen. Der OpenGL-Standard beschreibt etwa 250 Befehle, die die Darstellung komplexer 3D-Szenen in Echtzeit erlauben.“ [4] OpenGL ist als vergleich genutzt, da es erlaubt zwei und -dreidimensionale Objekte sehr hardwarenah abzubilden. Dieser Ansatz erfordert wiederrum ein erweitertes mathematisches Wissen, da er keine vorgefertigten Kameras oder Beleuchtungsmedien beinhaltet und somit nur sehr rudimentäre Features bietet. OpenGL wird oft in der

---

Q4 [4] [Q11 Matplotlib documentation

Matplotlib development team. (o. D.). Matplotlib documentation — Matplotlib 3.5.1 documentation. Matplotlib 3.5.1 documentation. Abgerufen am 16. Dezember 2021, von https://matplotlib.org/stable/index.html

OpenGL Wikipedia S.16] siehe auch [Q8 OpenGL Webseite S.15]

Videospielindustrie eingesetzt, um sehr komplexe und ressourcenintensive Szenen darzustellen.

2.6.2 Three-Dimensional Plotting in Matplotlib

Matplotlib ist eine *Python* Bibliothek. Die Bibliothekswebseite beschreibt sich übersetzt als: „*Matplotlib ist eine umfassende Bibliothek zur Erstellung statischer, animierter und interaktiver Visualisierungen in Python.*“ [5] Sie ermöglicht ähnlich wie dieses Projekt, das Darstellen von dreidimensionalen Szenen. Diese Bibliothek wird, aber eher für eine frontale zweidimensionale Ansicht genutzt und ist weitestgehend hierfür auch optimiert. Außerdem ist es nicht möglich eigene dreidimensionale Objekte zu nutzen. (Stand: 10.12.2021)

2.6.3 Unity

Das dritte Vergleichsobjekt ist die Spiel-Engine Unity. Die Engine bietet eine Vielzahl von vorgefertigten Features wie zum Beispiel Kameras, Bedeutung, Materialien und vieles weitere. Es kann wie dieses Projekt auch eine dreidimensionale Welt darstellen und interaktiv verändern. Wer in Unity ein Projekt realisieren möchte, muss dies mit der Unity-Applikation machen. Da Unity eine sehr große Anzahl von Features enthält kann, dies leicht abschreckend wirken.

[5] [Q11 Matplotlib documentation S.16]

3 Fazit

Dieses Kapitel berichtet über aufgetretene Schwachstellen, sowie Vorteile, die die entwickelte Sprache im Gegensatz zu den herkömmlichen Alternativen bietet. Außerdem wird über mögliche Features berichtet die zu signifikant Verbesserungen in der Benutzung führen könnten.

3.1 Ergebnisse

Grundsätzlich hat dieses Projekt gezeigt, dass es durchaus sinnvoll sein kann gewisse Problemstellung mit der Sprache *3D-Code* zu visualisieren, da sie ohne großes Vorwissen oder Kenntnisse im Bereich *Open GL* auskommt. Sie ermöglich auf verschiedene Weisen mit den selbstdefinierten Objekten zu interagieren und diese zu verändern. Außerdem lässt sich die Sprache auf verschiedensten Endgeräten ausführen, da sie auf *Kotlin* basiert.

3.2 Erkenntnisse

Wie die Beispiele [2.4 Implementierung der Anwendungsszenarien S.10] zeigen ist es mit wenig Aufwand möglich verschiedene komplexe Gebilde darzustellen. Dies wird erreicht, indem der Nutzer einer Liste Objekte hinzufügt, diese werden nach der Vollendung des Codes gerendert.

Ein wesentlicher Vorteil der eigenentwickelten Sprache ist, dass besondere Features sich leicht hinzufügen lassen, die sonst unüblich in normalen Programmiersprachen sind. Des Weiteren ist die Syntax leicht zu verstehen und komplexe System vor dem Nutzer abgekapselt. Ein Nachteil der Sprache *3Dcode* ist, dass sie ungefähr 938-mal langsamer ist als die Sprache, worauf *3D-Code* aufbaut. Dies ist durch mehrere Testdurchläufe Einens Priemzahlenermittlungsalgorithmus festgehalten. Siehe Anhang [A9 Performens Test [Kotlin vs 3D-Code] S.23]

In Anbetracht der oben genannten Punkte, wäre eine andere valide Option einer Entwicklung einer *Kotlin* Bibliothek, die ähnlich wie die entwickelte Sprache fungiert. Diese Option hätte zu dem jetzigen Entwicklungsstand noch Vorteile gegenüber der selbstentwickelten Sprache. Durch zusätzliche Features der Sprache *3D-Code* und weiteren Optionen könnte, könnten die Vorteile kompensiert werden. Siehe [3.3 Ausblick S.15]

3.3 Ausblick

Damit die Sprache *3D Code* einen wesentlichen Vorteil gegenüber anderen Optionen [2.6 Vergleich zu Alternativen S.13] hätte, müsste sie in manchen Punkten ergänzt werden.

- Statische Funktionen / Klassen
- Vererbung/ Polymorphie
- Beleuchtung und Kollisionserkennung
- Injektion von eigenen Shaderprogrammen
- Überschreiben der Kamera Bewegung
- Umwandlung des *3D-Codes* in Maschinensprache, statt Evaluation in Kotlin
- Das manuelle Aktualisieren des Anzeigebildes, während der Code Ausführung
- Ausarbeitung der Bibliotheken
- Eine Entwicklungsumgebung

Quellenverzeichnis

Q1 Perlin Noise Web.Archive.Org

Alexa Crawls. (2016, 30. Juni). Perlin Noise. Web.Archive.Org. Abgerufen am 10. Dezember 2021, von http://web.archive.org/web/20160530124230/http://freespace.virgin.net/hugo.elias/models/m_perlin.htm

Q2 15 Sorting Algorithms in 6 Minutes

Bingmann, T. [Timo Bingmann]. (2013, 20. Mai). 15 Sorting Algorithms in 6 Minutes [Video]. YouTube. https://www.youtube.com/watch?v=kPRA0W1kECg&feature=youtu.be

Q3 Projekt OuterSpace

Goßler, D. & Chouliaras, A. (2021, 22. August). GitHub - DennisGoss99/Prj_OuterSpace: 3D Space game. OuterSpace. Abgerufen am 16. Dezember 2021, von https://github.com/DennisGoss99/Prj_OuterSpace

Q4 3DCode – 3 Sorting Algorithms

Goßler, D. G. [DennisGoss99]. (2021a, Dezember 9). *3DCode - 3 Sorting Algorithms* [Video]. YouTube. https://www.youtube.com/watch?v=IJR2lAxHhA4&feature=youtu.be

Q5 3DCode – Perlin Noise

Goßler, D. G. [DennisGoss99]. (2021b, Dezember 9). *3DCode - Perlin Noise* [Video]. YouTube. https://www.youtube.com/watch?v=qcyR2wNYwds&feature=youtu.be

Q6 3DCode – Random walk

Goßler, D. G. [DennisGoss99]. (2021c, Dezember 9). *3DCode - Random walk 3D* [Video]. YouTube. https://www.youtube.com/watch?v=OaKN4KQeAzc&feature=youtu.be

Q7 Projekt PuC_C--

Goßler, D. & Momberg, L. (21–07-28). GitHub - DennisGoss99/PuC_C--: mini c. PuC_C--. Abgerufen am 16. Dezember 2021, von https://github.com/DennisGoss99/PuC_C--

Q8 OpenGL Webseite

Khronos Group. (o. D.). OpenGL - The Industry Standard for High Performance Graphics. Copyright (c) 2000 - 2021 Khronos Group. Abgerufen am 16. Dezember 2021, von https://www.opengl.org/

Q9 Merge Sort Algorithm

Kumar, A., Dutt, A. & Saini, G. (2014, 11. Dezember). *Merge Sort Algorithm*. citeseerx.ist.psu.edu. Abgerufen am 10. Dezember 2021, von https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.845.6329&rep=rep1&type=pdf

Q10 Lightweight Java Game Library

LWJGL - Lightweight Java Game Library. (o. D.). LWJGL. Abgerufen am 15. Dezember 2021, von https://www.lwjgl.org/

Q11 Matplotlib documentation

Matplotlib development team. (o. D.). Matplotlib documentation — Matplotlib 3.5.1 documentation. Matplotlib 3.5.1 documentation. Abgerufen am 16. Dezember 2021, von https://matplotlib.org/stable/index.html

Q12 OpenGL Wikipedia

Seite „OpenGL“. In: Wikipedia – Die freie Enzyklopädie. Bearbeitungsstand: 8. November 2021, 14:54 UTC.
URL: https://de.wikipedia.org/w/index.php?title=OpenGL&oldid=217099756 (Abgerufen : 16. Dezember 2021, 22:44 UTC)

Q13 Perlin-Noise Wikipedia

Seite „Perlin-Noise“. In: Wikipedia – Die freie Enzyklopädie. Bearbeitungsstand: 11. Oktober 2021, 14:27 UTC. URL: https://de.wikipedia.org/w/index.php?title=Perlin-Noise&oldid=216289760 (Abgerufen: 10. Dezember 2021, 17:37 UTC)

Anhang

A1 Github repository Link

https://github.com/DennisGoss99/PraxProj_3DCode

A2 Testfälle

https://github.com/DennisGoss99/PraxProj_3DCode/tree/main/3DCode/src/test/kotlin

A3 Codebeispiel Link [3 Sorting Algorithms]

https://github.com/DennisGoss99/PraxProj_3DCode/blob/main/3DCode/code/Sort_App.3dc

A4 Bubblesortfunktion

```
Void BubbleSortStep(){
        if(!finished){
                if(sortPos < sortSize){
                        SortObj x = sortObjects.Get(sortPos)
                        SortObj x1 = sortObjects.Get(sortPos + 1)

                        //Swap Position
                        if(x.score > x1.score){
                                sortObjects.InsertAt(sortPos, x1)
                                sortObjects.InsertAt(sortPos + 1, x)

                                x.Move(Vector3f( 3.0, 0.0, 0.0))
                                x1.Move(Vector3f( -3.0, 0.0, 0.0))
                        }

                        sortPos += 1

                }else{

                        if(sortSize <= 1){
                                finished = true
                                Println("Finished sorting: BubbleSort")
                        }

                        sortSize -= 1
                        sortPos = 0
                }
        }
}
```

A5 Mergesortfunktion

```
Void MergeSortStep(){
        if(!finished){
        if(sortPos < sortSize){
                SortObj x = sortObjects.Get(sortPos)
                SortObj x1 = sortObjects.Get(sortPos + 1)

                if(x.score > x1.score){
                        sortObjects.InsertAt(sortPos, x1)
                        sortObjects.InsertAt(sortPos + 1, x)

                        x.Move(Vector3f( 3.0, 0.0, 0.0))
                        x1.Move(Vector3f( -3.0, 0.0, 0.0))
                }
                sortPos += 2

                for(Int i = 4 ; i <= elementCount ; i *= 2){

                        if(m.Mod(sortPos,i) == 0){
                                Int pos1 = sortPos - 1
                                Int pos2 = (sortPos - (i/2)) - 1
                                Int pos1L = i / 2
                                Int pos2L = i / 2
                                Int count = i - 1

                        Array tempArray<SortObj> = Array<SortObj>(i)

                                SortObj x = sortObjects.Get(pos1)
                                SortObj x1 = sortObjects.Get(pos2)

                                while(pos1L >= 1 || pos2L >= 1){
                                        if(pos1L >= 1){
                                                x = sortObjects.Get(pos1)
                                        }
                                        if(pos2L >= 1){
                                        x1 = sortObjects.Get(pos2)
                                        }

                        if(pos1L >= 1 && (pos2L < 1 || x.score > x1.score))
                                tempArray.Set(count,x)
                                pos1 -= 1
                                pos1L -= 1
                        }else{
                        if(pos2L >= 1 && (pos1L < 1 || x.score <= x1.score)){
                                tempArray.Set(count,x1)
                                pos2 -= 1
                                pos2L -= 1
                        }
                        }
                        count -= 1
                }
                for(Int j = 0 ; j < tempArray.size ; j += 1){
                 SortObj sObj = tempArray.Get((tempArray.size - 1) - j)
                 sObj.SetMove( Vector3f((((sortPos - 1) - j) * 3) -
((elementCount / 2.0) * 3), sObj.pos.y, sObj.pos.z))
                 sortObjects.InsertAt((sortPos - 1) - j, sObj)
                }

                }
                }
                }else{
                        finished = true
                        Println("Finished sorting: MergeSort")
                }
        }
}
```

A6 Combosortfunktion

```
Bool combSortSortet = true

Void CombSortStep(){
        if(!finished){
                if(sortPos < sortSize){
                        SortObj x = sortObjects.Get(sortPos)
                        SortObj x1 = sortObjects.Get(sortPos + 1)

                        //Swap Position
                        if(x.score > x1.score){
                                sortObjects.InsertAt(sortPos, x1)
                                sortObjects.InsertAt(sortPos + 1, x)

                                x.Move(Vector3f( 3.0, 0.0, 0.0))
                                x1.Move(Vector3f( -3.0, 0.0, 0.0))
                                combSortSortet = false
                                sortPos += 1
                        }

                        sortPos += 1

                }else{

                        if(combSortSortet){
                                finished = true
                                Println("Finished sorting: Comb Sort")
                        }

                        combSortSortet = true
                        sortPos = 0
                }
        }
}
```

A7 Codebeispiel Link [Perlin Noise]

https://github.com/DennisGoss99/PraxProj_3DCode/blob/main/3DCode/code/PerlinNoise_App.3dc

A8 Codebeispiel Link [Random Walk 3D]

https://github.com/DennisGoss99/PraxProj_3DCode/blob/main/3DCode/code/RandomWalk_App.3dc

A9 Performens Test [Kotlin vs 3D-Code]

Um die beiden Sprachen miteinander zu vergleichen, werden Primzahlen in einem gewissen Bereich gesucht. Beide Sprachen erhalten ein Programm mit der gleichen Code Semantik.

```
Int Mod(Int n, Int k)
{
    while(n >= k){
        n -= k
    }
    return n
}

Int Main()
{
    Int x = 2
    Int i = 2
    Bool quitFlag = true
    Int foundPrimes = 0

    while(x <= $limit)
    {
        while(i <= x && quitFlag)
        {
            if((Mod(x,i) == 0) && (x != i))
            {
                quitFlag = false // break
            }
            else
            {
                if(i == x)
                {
                    foundPrimes += 1
                }
            }

            i += 1
        }
        quitFlag = true
        i = 2
        x += 1
    }
    return foundPrimes
}
```

Bereich von 0 bis	Kotlin	3D-Code	Faktor
5000	10 ms	9912 ms	991x
7500	22 ms	22372 ms	1016x
10000	53 ms	38952 ms	734x
15000	93 ms	90467 ms	972x
20000	169 ms	161270 ms	954x
25000	251 ms	161270 ms	954x
30000	370 ms	304294 ms	822x
40000	670 ms	655132 ms	977x
50000	1092 ms	1119378 ms	1025x

Die obere Tabelle zeigt das 3D-Code ungefähr im Durchschnitt 938-mal langsamer ist als Kotlin.