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

Inhaltsverzeichnis

[Inhaltsverzeichnis 2](#_Toc90060354)

[Tabellenverzeichnis 4](#_Toc90060355)

[Abbildungsverzeichnis 5](#_Toc90060356)

[1 Einleitung 6](#_Toc90060357)

[1.1 Relevanz 6](#_Toc90060358)

[1.2 Zielsetzung 6](#_Toc90060359)

[1.3 Planung 6](#_Toc90060360)

[1.3.1 Definierung der Programmiersprache 6](#_Toc90060361)

[1.3.2 Planung der Aufbereitung des 3D Outputs 6](#_Toc90060362)

[1.3.3 Planung des Thesenüberprüfung 7](#_Toc90060363)

[1.4 Grundaufbau 7](#_Toc90060364)

[1.5 Veranschaulichung 7](#_Toc90060365)

[2 Hauptteil 8](#_Toc90060366)

[2.1 Anpassung/ Erweiterung der Sprache 8](#_Toc90060367)

[2.1.1 Feature Auflistung 8](#_Toc90060368)

[2.1.2 Umgebung 8](#_Toc90060369)

[2.1.3 Variablen Einfache Typen / Objekte 8](#_Toc90060370)

[2.1.4 Überlagerte Funktionen 9](#_Toc90060371)

[2.1.5 Generische Klassen/ Funktionen 9](#_Toc90060372)

[2.1.6 Dateiverwaltung 9](#_Toc90060373)

[2.1.7 Interne Funktionen 10](#_Toc90060374)

[2.2 Zusammenführung und Rendering 10](#_Toc90060375)

[2.2.1 Initialisierung 10](#_Toc90060376)

[2.2.2 Update / OnKey 10](#_Toc90060377)

[2.2.3 Umwandlung von Objekten 10](#_Toc90060378)

[2.3 Implementierung der Anwendungsszenarien 10](#_Toc90060379)

[2.3.1 Sortieralgorithmen 10](#_Toc90060380)

[2.3.2 Perlin Noise 11](#_Toc90060381)

[2.3.3 Random Walk 12](#_Toc90060382)

[2.4 Testfälle 12](#_Toc90060383)

[2.5 Vergleich zu Alternativen 12](#_Toc90060384)

[3 Fazit 13](#_Toc90060385)

[Quellenverzeichnis 14](#_Toc90060386)

[Q1 Perlin Noise Web.Archive.Org 14](#_Toc90060387)

[Q2 3DCode – 3 Sorting Algorithms 14](#_Toc90060388)

[Q3 3DCode – Perlin Noise 14](#_Toc90060389)

[Q4 3DCode – Random walk 14](#_Toc90060390)

[Q5 Merge Sort Algorithm 14](#_Toc90060391)

[Q6 Perlin-Noise Wikipedia 14](#_Toc90060392)

[Anhang 15](#_Toc90060393)

[A1 Github repository Link 15](#_Toc90060394)

[A2 Codebeispiel Link [3 Sorting Algorithms] 15](#_Toc90060395)

[A3 Codebeispiel Link [Perlin Noise] 15](#_Toc90060396)

[A4 Codebeispiel Link [Random Walk 3D] 15](#_Toc90060397)

Abbildungsverzeichnis

[Abbildung 1 [Compileraufbau] 6](file:///C:\Users\Merdo\Desktop\--%20PraxPro\PraxProj_3DCode\Documentation\Praxisprojektbericht.docx#_Toc90061338)

[Abbildung 2 [Variablen Zuweisung] 8](#_Toc90061339)

[Abbildung 3 [Perlin Noise Output] 11](#_Toc90061340)

[Abbildung 4 [Random Walk 3D] 12](#_Toc90061341)

# 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*

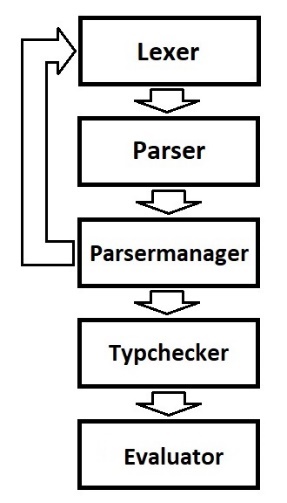
Der Ablauf gilt, wie in folgender Abbildung veranschaulicht, trotz Änderung der Programmiersprachenart nicht anzupassen.

Abbildung 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. *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 der Sprache.

### 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
* Private Klassenattribute/-funktionen
* 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 und als „DynamicValue“ gespeichert werden. In diesem Fall wird nur ein Pointer in der Variabel abgelegt. Dies hat zur Folge, dass mehrere Objekte auf dasselbe Objekt, und dessen Inhalt, zeigen könnten.



Abbildung 2 [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 existiert, muss nun der Evaluator die Funktion herausfiltern, die die gleichen Parameter besitzt, wie die aufgerufene.

### Generische Klassen/ 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 muss, 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.

### Dateiverwaltung

Da es möglich ist in mehrere Dateien zu programmieren und diese sich jeweils inkludieren können, musste 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.

### Interne Funktionen

Da es oftmals sehr aufwendig oder unmöglich ist manche Features in der Sprache zu entwickeln, werden diese in die Sprache eingebettet.

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 Array enthält.

## Zusammenführung und Rendering

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

### 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. Nach dem erfolgreichen „Type checken“ evaluiert.

### Update / OnKey

Wenn ein Tastaturinput stattfindet, wird jedes Mal die Funktion „OnKey“, mit der vorherigen Umgebung, evaluiert. Durch dieses Vorgehen kann die statische dreidimensionale Welt animiert werden. Hierfür 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.

### 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.

## 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.

### Sortieralgorithmen

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

Hierfür 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“ wurde mittels 26 Zeilen Code implementiert, wobei nur zwei Zeilen für das Aktualisieren der Position benötigt werden. QUELLE

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]](#footnote-1) Der implementierte Algorithmus LINK ist allerding aus komplexitätsgründen auf Sortiergrößen von 2n limitiert.

Der dritte Algorithmus ist eine Abwandlung von Combsort. Diese Implantation vergleich 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.

Die Ausführung, von den oben genannten drei Algorithmen, ist per Videoform LINK abrufbar.

### Perlin Noise

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

Der Algorithmus lässt sich wie folgt beschreiben:„Perlin Noise ist eine [Rauschfunktion](https://de.wikipedia.org/wiki/Rauschen_(Physik)) auf der Basis von seudozufälligen [Gradientenwerten](https://de.wikipedia.org/wiki/Gradient" \o "Gradient) an Gitterpunkten. […] Perlin Noise wird häufig in der [Bildsynthese](https://de.wikipedia.org/wiki/Bildsynthese) eingesetzt, um natürliche Phänomene wie Wolken, Landschaftstopologien oder Wasser zu simulieren.“ *[[2]](#footnote-2)*

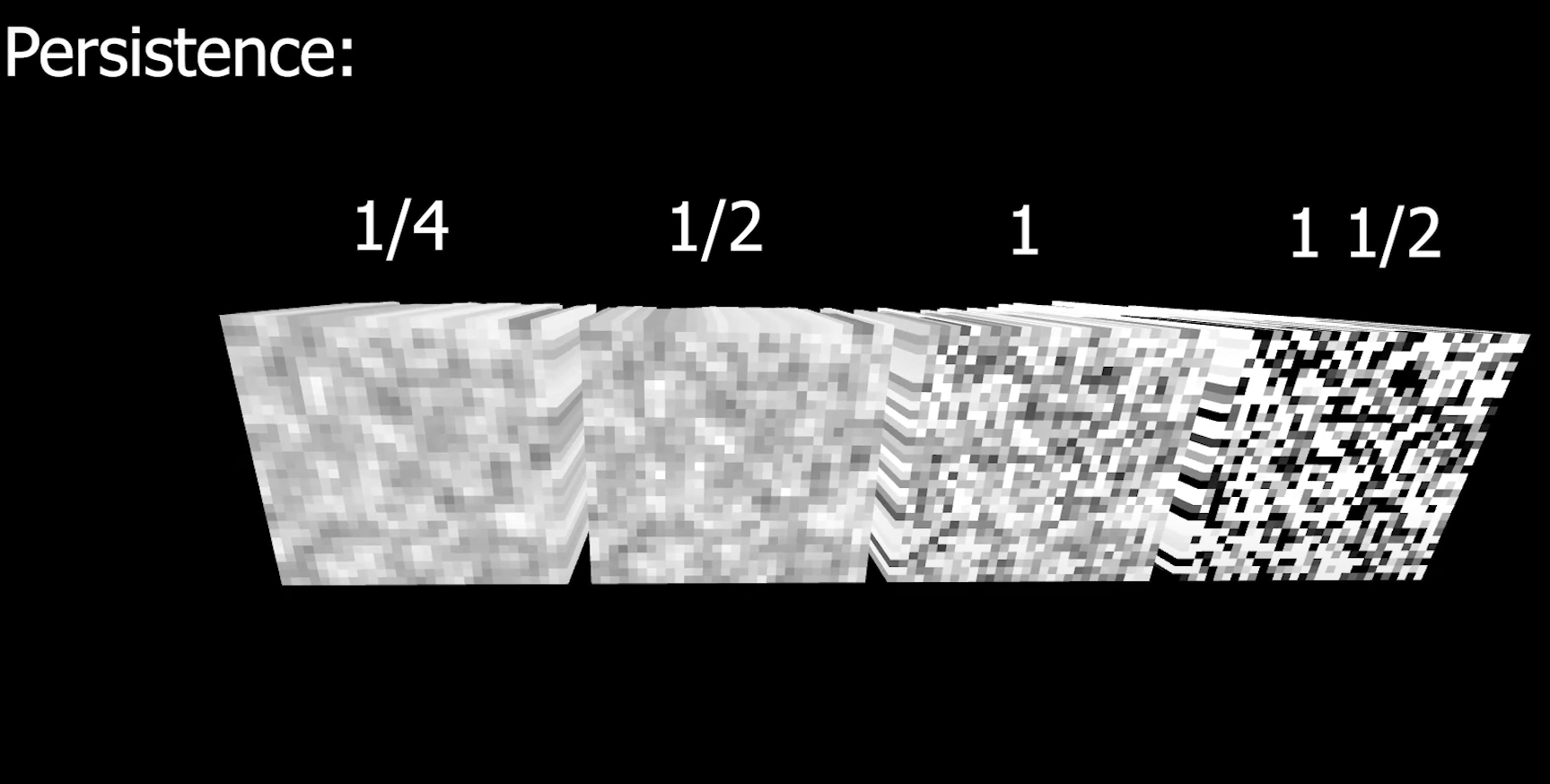


Abbildung 3 [Perlin Noise Output]

Die Abbildung LINK zeigt die Ausgabe des Algorithmus. Hierbei wurde vier verschiedene Werte für die Persistenz gewählt. Das Video LINK veranschaulicht zusätzlich die dreidimensionale Ansicht, die aus einem anderen Blickwinkel hervorgeht.

Die Implementierung und das generelle Verständnis gehen aus Quelle LINK hervor. Der entwickelte Code ist im Anhang zu finden LINK.

### 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]](#footnote-3)

Ein Bild, das Karte enthält.

Automatisch generierte Beschreibung

Abbildung 4 [Random Walk 3D]

Die Abbildung LINK stellt die Ausgabe des oben beschrieben Algorithmus im Zeitpunkt t+2min 10s da. Die animierte Ausgabe ist unter LINK zu finden.

## Testfälle

Um nach Veränderungen des Programmcodes weitergehend sicherzustellen, dass die Veränderung keine Fehler ausgelöst, sind insgesamt 170 Tests entstanden. Die Tests decken einen Großteil der Funktionalitäten der Programmiersprache ab. Über folgenden Link sind alle Tests abrufbar LINK.

## Vergleich zu Alternativen

# Fazit

Quellenverzeichnis

1. 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](http://web.archive.org/web/20160530124230/http:/freespace.virgin.net/hugo.elias/models/m_perlin.htm)

1. 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>

1. 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>

1. 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>

1. 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>

1. 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

1. Github repository Link

<https://github.com/DennisGoss99/PraxProj_3DCode>

1. Codebeispiel Link [3 Sorting Algorithms]

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

1. Codebeispiel Link [Perlin Noise]

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

1. Codebeispiel Link [Random Walk 3D]

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

1. Q4 Merge Sort Algorithm, S.19 [↑](#footnote-ref-1)
2. Q5 Perlin-Noise Wikipedia [↑](#footnote-ref-2)
3. Zugehöriger Code: Anhang…. [↑](#footnote-ref-3)