HAW HAMBURG

BACHELORTHESIS
Minh Lê

Prozedural generierte Inhalte in Augmented Reality Anwendungen am Beispiel der Generierung eines eingebetteten Gleissystems mittels Wave Function Collapse

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Minh Lê

Prozedural generierte Inhalte in Augmented Reality Anwendungen am Beispiel der Generierung eines eingebetteten Gleissystems mittels Wave Function Collapse

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Angewandte Informatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Philipp Jenke
Zweitgutachter: Prof. Dr. Thomas Lehmann

Eingereicht am: 02. Oktober 2023

Minh Lê

Thema der Arbeit

Prozedural generierte Inhalte in Augmented Reality Anwendungen am Beispiel der Generierung eines eingebetteten Gleissystems mittels Wave Function Collapse

Stichworte

Augmented Reality, Procedural Content Generation, Gleisgenerierung, Mobilgerät, Umgebungsbewusste Content Generierung, Wave Function Collapse

Kurzzusammenfassung

Augmented Reality Anwendungen können von Automatisierung durch Procedural Content Generation profitieren. Derzeit sind viele Mobilgeräte Augmented Reality fähig. Augmented Reality Anwendungen sind jedoch nicht weit verbreitet. Procedural Content Generation kann die Entwicklungskosten und den Zeitaufwand der Erstellung von Augmented Reality Inhalten verkürzen. In dieser Arbeit wird ein modifizierter Wave Function Collapse Algorithmus verwendet in einer Augmented Reality Anwendung, um aus aufbereiteten Tiefendaten ein Gleis prozedural zu generieren. Der Gleisverlauf und die Platzierung werden von den Tiefendaten bestimmt. Die entwickelte Anwendung für Android Mobilgeräte zeigt, dass die Integration von prozedural generierten Inhalten in Augmented Reality Anwendungen sinnvoll ist. Jedoch ist Optimierung in Genauigkeit und Leistung der entwickelten Lösung notwendig.

Minh Lê

Title of Thesis

Procedurally generated content in augmented reality applications illustrated by generating an embedded rail system using wave function collapse

Keywords

augmented reality, procedural content generation, rail generation, mobile device, spatially aware content generation, wave function collapse

Abstract

Augmented reality applications can benefit from automation via procedural content generation. Currently many mobile devices are capable of augmented reality. However augmented reality applications are not widely available. Procedural content generation can reduce the development cost and time of augmented reality content creation. In this work, a modified wave function collapse algorithm is used in an augmented reality application to procedurally generate rails with processed depth data. The course and placement of the rails are determined by the depth data. The developed application for Android mobile devices shows that the integration of procedurally generated content in augmented reality applications is useful. However, optimization in accuracy and performance of the suggested solution is necessary.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungen

AR Augmented Reality.

HAW Hochschule für Angewandte Wissenschaften.

HMD Head-Mounted Display.

Lidar Light imaging, detecting and ranging.

MVC Model View Controller.

PCG Procedural Content Generation.

PRNG Pseudo-Random Number Generators.

RGB-D Red Green Blue-Depth.

SA Spatial Algorithms.

SLAM Simultaneous Localization and Mapping.

ToF Time-of-Flight.

VR Virtual Reality.

VST Video See-Through.

WFC Wave Function Collapse.

1 Einleitung

Augmented Reality (AR) ist eine spannende Technologie, die die Wahrnehmung mithilfe von Sensoren analysieren, täuschen und erweitern kann. Die Vision besteht darin, virtuelle Inhalte nahtlos in die Realität des Benutzers zu integrieren. Die Technologie findet in verschiedenen Bereichen Anwendung, darunter Medizin, Bau und Wartung von Maschinen, Bildung sowie Unterhaltung. Bis Anfang 2000 war sie aufgrund hoher Kosten und Komplexität hauptsächlich in Forschungsinstitutionen und der Industrie eingesetzt worden. In den letzten Jahren wurden jedoch leistungsfähigere AR-Bibliotheken veröffentlicht, die auch Einsteigern die Entwicklung von AR-Anwendungen ermöglichen, indem sie einfachen Zugang zu Umgebungsdaten und Tracking bieten. Zunehmend gibt es mehr Endnutzergeräte für AR wie Head-Mounted Display (HMD), leistungsfähigere Mobilgeräte und verbesserte Bibliotheken, die die Nutzung von AR in mobilen Anwendungen unter einer breiteren Masse ermöglichen. Die Verwendung von Umgebungsdaten birgt Potenzial für die Schaffung immersiverer AR-Anwendungen, doch erst in den letzten Jahren wurden Verfahren entwickelt, die auf vielen Mobilgeräten nutzbar sind.

Die Procedural Content Generation (PCG) ist ein Fachgebiet, das eine Vielzahl von Methoden umfasst, um Objekte und Inhalte mithilfe von Algorithmen zu erzeugen. Diese Inhalte reichen von Musik und Bildern bis hin zu virtuellen Welten. In diesem Bereich werden stetig innovative Methoden und komplexe Parametermengen erforscht und in Algorithmen eingebracht, um dynamische Inhalte zu erstellen.

Bei der PCG geht es darum, Virtuelles zu erstellen und zu gestalten, während AR darauf abzielt, die Realität zu analysieren und Virtuelles zu verwenden, um die Wahrnehmung zu beeinflussen. AR kann von der PCG profitieren, da die generierten Inhalte adaptiv sind und die Immersion steigern können. Zusätzlich können durch die massenhafte Generierung von Inhalten Zeit und Aufwand gespart werden, vor allem bei automatischer Platzierung von Inhalten.

AR kann der PCG vielfältige Anwendungsmöglichkeiten eröffnen und neue inspirierende Ansätze bieten. Die Umgebungsdaten bieten eine große Parametermenge, deren Einfluss

sich in Echtzeit, in den Inhalten widerspiegeln kann, da AR eine starke Verbindung zwischen der Bewegung des Benutzers und den Umgebungsdaten herstellt.

In dieser Arbeit wird ein Prototyp für Android-Mobilgeräte und ein PCG-Verfahren entwickelt, um AR und PCG zu kombinieren. Der Prototyp soll ein visuelles AR-System bieten, das PCG-Verfahren nutzen, Umgebungsdaten dafür aufbereiten und den generierten Inhalt, wie in der Realität eingebettet, anzeigen.

Das PCG-Verfahren soll Umgebungsdaten in Form von Tiefendaten nutzen, um ein Gleissystem zu generieren. Das Verfahren nutzt den WFC-Algorithmus, der seit seiner Entwicklung hohe Anwendung findet. Dieser Algorithmus arbeitet mit räumlichen Strukturen wie Gittern oder Graphen. Das Gleissystem wird als Teil eines AR-Erlebnisses vorgestellt und soll auf einer Ebene erscheinen.

Ein Benutzer wird mit der Anwendung 'Vergleiser' in einer statischen Umgebung Tiefendaten um eine Ebene herum sammeln, indem er sich mit seinem Mobilgerät im Raum bewegt. Dann kann er die Generierung eines Gleissystems starten. Die Anwendung erzeugt dann ein Gleissystem auf den Freiflächen dieser Ebene und lässt einen Zug fahren.

Die Arbeit ist in weitere fünf Kapitel unterteilt. Im nächsten Kapitel werden Grundlagen und Beispiele für AR, PCG, Tiefendaten und WFC-Algorithmus vorgestellt sowie themenverwandte Arbeiten angesprochen. Im dritten Kapitel werden die Anforderungen für die Anwendung vorgestellt. Dann wird die Tiefendaten-Verarbeitung und Funktionsweise des PCG-Verfahrens erklärt. Das vierte Kapitel beschäftigt sich mit der Struktur, dem Verhalten und der Implementierung der Anwendung. Schließlich folgt im fünften Kapitel eine Evaluation des entwickelten PCG-Verfahrens und der Anwendung. Die Arbeit endet in einem Fazit, das die Ergebnisse zusammenfasst und einen Ausblick gibt.

2 Stand der Technik

Dieses Kapitel befasst sich mit Augmented Reality, Tiefendaten, Procedural Content Generation, dem Wave Function Collapse Algorithmus und themenverwandten Arbeiten.

Der Abschnitt Augmented Reality handelt von gängigen Abläufen, Visualisierung und Tracking. Es folgt eine Vorstellung von Anwendungsfällen für Tiefendaten. Im nächsten Abschnitt Procedural Content Generation folgt ein Überblick über generierbare Inhalte und verwendete Methoden. Schlussendlich wird der WFC erläutert und themenverwandte Arbeiten hervorgehoben.

2.1 Augmented Reality

R.T. Azuma hat 1997 in seiner Studie den damaligen Stand der Forschung im Fachbereich Augmented Reality (AR) erfasst. Die Studie wird fort folgend regelmäßig vielfach zitiert für die generelle und prägnante Charakterisierung von AR-Systemen. AR-Systeme besitzen demnach die folgenden drei Eigenschaften [4]:

1. Ein AR-System kombiniert Reales und Virtuelles.
2. Ein AR-System ist interaktiv in Echtzeit.
3. Ein AR-System ist in 3D registriert.

Im Folgenden werden die drei Eigenschaften näher erläutert.
(1.) Bei den meisten AR-Systemen handelt es sich bei den zu kombinierenden realen und virtuellen Inhalten, um visuelle oder auditive Inhalte. Diese Definition beschränkt die Inhalte nicht darauf. Je nach Medium unterscheiden sich die Ansätze, wie Real und Virtuell kombiniert werden.
(2.) AR-Systeme bieten dem Benutzer ein navigierbares Erlebnis. Freie Körperbewegung oder Kontrolle des Blickpunkts sind häufige Interaktionsmöglichkeiten.

(3.) Registrierung in 3D bedeutet, dass virtuelle und reale Inhalte möglichst zeitnah aneinander angeglichen werden. **Photometrische Registrierung** meint die Angleichung der Beleuchtung von virtuellen Objekten an reale Lichtverhältnisse. Dafür müssen Informationen über die realen Lichtquellen bekannt sein. **Geometrische Registrierung** bezieht sich auf die Angleichung von Koordinatensystemen und ist die übliche Bedeutung von Registrierung. Um die Illusion von virtuellen Objekten in der Realität zu erzeugen, werden virtuelle und reale Objekte in einem einheitlichen Koordinatensystem beschrieben.

2.1.1 Komponenten eines Augmented Reality Systems

Mögliche Komponenten für AR-Systeme werden im Folgenden aus Eigenschaft (2.) nach D. Schmalstieg und T. Höllerer [23, Kapitel 1] hergeleitet.

Die Eigenschaft (2.) fordert Echtzeit. Zwar bedeutet Echtzeit je nach Anwendung etwas anders, doch ein interaktives System in Echtzeit erfordert immer einen stark gekoppelten Feedback Loop zwischen Benutzer und System.

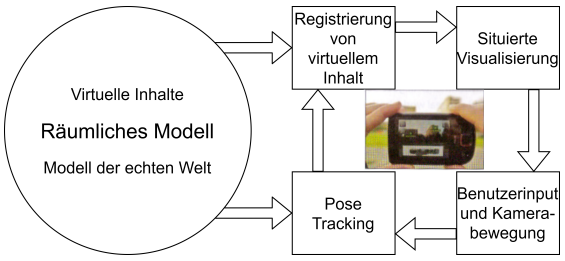


Abbildung 2.1: AR nutzt einen Feedback Loop zwischen Benutzer und System. Der Benutzer beobachtet den AR-Bildschirm und kontrolliert den Blickpunkt. Das System trackt den Blickpunkt des Benutzers, registriert die Pose in der echten Welt mit den virtuellen Inhalten und präsentiert situierte Visualisierungen. Die Abbildung wurde übersetzt und stammt von Figure 1.1 [4].

Mögliche Inhalte eines Feedback-Loops für visuelle AR-Systeme werden auf Abb. 2.1 abgebildet. Der Benutzer navigiert ständig die AR-Szene. Das System trackt die Veränderung, z.B. um die Blickrichtung, die Position oder die Orientierung (dies nennt sich gemeinsam Pose) des Benutzers zu ermitteln. Sollen die virtuellen Inhalte in der Realität registriert sein, muss das System die Visualisierung an die neue Benutzerpose anpas-

sen. Das System präsentiert dem Benutzer eine Visualisierung, die in der echten Welt registriert ist (situierte Visualisierung).

Abb. 2.1 zeigt links eine Modellkomponente, die die virtuellen Inhalte und Informationen der realen Welt verwaltet. Letztere dienen dem Tracking als Referenz.

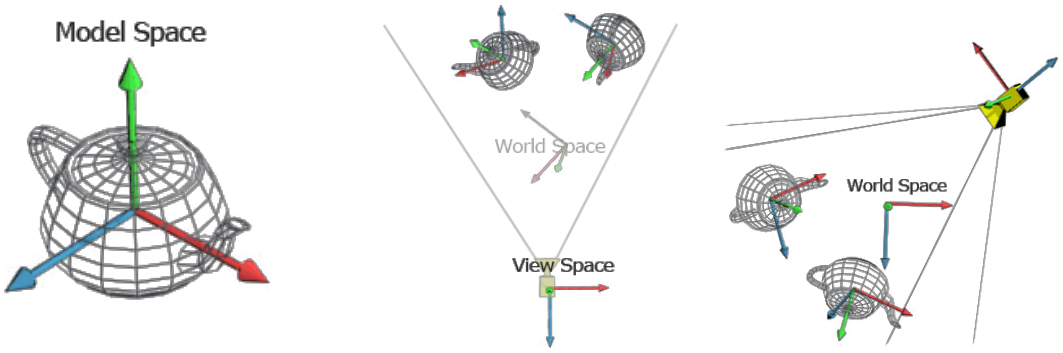
Aus dem Feedback-Loop lässt sich ableiten, dass sich ein visuelles AR-System in die drei Komponenten Tracking, Registrierung und Visualisierung aufteilen lässt. Lediglich Visualisierung und Tracking werden im Folgenden behandelt. **Registrierung** ist bereits eng mit Tracking gekoppelt, weil Trackingverfahren selbst Ergebnisse aus unterschiedlichen Koordinatensystemen vereinheitlichen.

2.1.2 Visualisierung

In jedem Frame müssen reale und virtuelle Objekte kombiniert dargestellt werden. Bei dem Video See-Through (VST) Ansatz, der auch in dieser Arbeit genutzt wird, erfolgt dies über ein Display und eine Kamera. Die Kamera nimmt die Umgebung auf und das Display zeigt das Video an. Über dem Video werden dann virtuelle Objekte gezeichnet. Das Zeichnen virtueller Objekte (das Rendern) ist Forschungsgegenstand der Computergrafik. AR-Anwendungen müssen virtuelle Objekte aus verschiedenen Blickpunkten wie z.B. die des Benutzers rendern. Dafür werden die Objekte in einer virtuellen Szene beschrieben. Diese Szene wird mithilfe einer Abfolge von standardmäßigen Prozessen, der Rendering Pipeline, gerendert. Sie ermöglicht eine flexible Visualisierung der Objekte.

Virtuelle Szene und 3D-Modelle

Eine virtuelle Szene enthält 3D-Modelle, Kameras, Lichtquellen und wird häufig als Szenengraph hierarchisch repräsentiert sowie verwaltet. 3D-Modelle sind abstrakte Datenstrukturen, die Informationen darüber enthalten, wie ein Objekt zu rendern ist. Dazu gehören ein Mesh und üblicherweise ein Material. Die Realisierung des Materials hängt vom benutzten Beleuchtungsmodell ab. Kameras sind Punkte, aus deren Sicht eine Szene gerendert werden kann. Sie haben sog. interne/intrinsische Parameter (linsenspezifische optische Parameter wie Brennweite etc.) und externe/extrinsische Parameter (die Kamerapose).



(a) (b) (c)

Abbildung 2.2: Überblick über verschiedene Koordinatensysteme. Für jedes Koordinatensystem wurde der Ursprung eingezeichnet. 2.2a zeigt das Mesh eines Teekessels im lokalen Koordinatensystem (Model Space). 2.2b zeigt die Szene im globalen Koordinatensystem (World Space / Weltkoordinatensystem), mit zwei Teekesseln und einer Kamera (gelb). Für alle Objekte ist ihr lokaler Ursprung eingezeichnet. 2.2c zeigt eine Draufsicht derselben Szene, nachdem sie in das Kamera-Koordinatensystem (View Space) überführt wurde [1, Fig. 6, Fig. 8].

Rendering Pipeline und Transformationen

Eine Rendering Pipeline enthält standardmäßige Abläufe, um eine Szene von 3D auf einen Bildschirm in 2D zu rendern. Viele Abläufe können teilweise hardwarebeschleunigt ausgeführt werden (variiert je nach Framework und Grafikkartenhersteller) und ermöglichen sogar auf leistungsarmen Mobilgeräten ein effizientes Rendern. Die Pipeline bietet eine Möglichkeit Objekte effizient in der Szene zu transformieren, d.h. Transformationen auf sie anzuwenden wie Translation, Rotation und Skalierung. Hier ist eine Transformation eine 4×4 Matrize um Punkte zu transformieren. Die Punkte haben dementsprechend vier Koordinaten (x, y, z, w) sog. homogene Koordinaten und beschreiben 3D-Punkte. In homogenen Koordinaten können u.a. perspektivische Projektionen als Matrizen dargestellt werden.

In visuellen AR- und Virtual Reality (VR)-Systemen wird mit vier Koordinatensystemen gearbeitet. Das globale Weltkoordinatensystem wird genutzt um die Szene und die Benutzerpose zu beschreiben. Die Benutzerposition bestimmt zu Anfang den Koordinatenursprung. Der Bildschirm hat ein eigenes Koordinatensystem. Virtuelle Objekte werden in ihren eigenen lokalen Koordinatensystemen beschrieben. Die Kamera ist auch ein virtuelles Objekt und ihr lokales Koordinatensystem wird hier Kamera-Koordinatensystem

genannt. Auf Abb. 2.2a ist das Mesh eines Teekessels in einem Koordinatensystem beschrieben. Auf Abb. 2.2b sieht man nun zwei Teekesseln mit verdrehten Koordinatensystemen, da diese Abbildung eine Draufsicht des Kamera-Koordinatensystems ist. Um das Teekessel-Koordinatensystem in das Kamera-Koordinatensystem zu überführen werden Transformationen genutzt. Für vier Koordinatensysteme gibt es drei Transformationen, welche als Teil der Pipeline nacheinander angewendet werden. Das sind die Modell-Transformation, Kamera-Transformation[1] und Projektions-Transformation. Jede Transformation wird erläutert, speziell für Modell-Transformation und Kamera-Transformation folgt ein Tracking-bezogener Textabschnitt.

Modell-Transformation Da 3D-Modelle in lokalen Koordinatensystemen beschrieben werden, müssen sie in das Weltkoordinatensystem überführt werden. Für jedes 3D-Modell gibt es eine Modell-Transformation, die die lokalen Koordinaten in das Weltkoordinatensystem abgebildet. Die Matrize platziert das Objekt üblicherweise mithilfe von Rotation, Skalierung und Translation.

Virtuelle Objekte werden von der Anwendung verwaltet und müssen nicht getrackt werden. Statische reale Objekte kann man als Teil des Weltkoordinatensystems betrachten und werden nicht getrackt. Wenn sie dynamisch sind und sich ihre Pose verändern kann, müssen sie ggf. getrackt werden. Letztendlich müssen virtuelle und reale Objekte in dasselbe Weltkoordinatensystem abgebildet werden, um eine situierte Visualisierung zu ermöglichen.

Kamera-Transformation Anschließend werden alle Weltkoordinaten in das Kamera-Koordinatensystem über die Kamera-Transformation abgebildet. Diese Transformation erleichtert die Projektion auf die Bildebene aus Kamerasicht.

Die Platzierung der Kamera in das Weltkoordinatensystem verhält sich umgekehrt zum Platzieren von Weltkoordinaten in das Kamera-Koordinatensystem. D.h. die Kamera-Transformation ist die inverse Modell-Transformation für die Kamera und lässt sich aus der Kamerapose (im Weltkoordinatensystem) berechnen.

[1] Die Kamera-Transformation ist auch als Augenpunkt-Transformation bekannt und nennt sich auf Englisch 'view transformation'. In der OpenGL Spezifikation wird diese Transformation mit der Modell-Transformation zur ModelView-Transformation zusammengefasst.

Wenn dem Benutzer erlaubt wird, sich frei zu bewegen und den Blickpunkt zu verändern, muss dieser getrackt werden, um eben die Kamera-Transformation zu aktualisieren. Ansonsten würden Objekte aus anderen Blickpunkten dargestellt werden. Das Tracking der Kamerapose ist eine der wichtigsten Aufgaben von AR-Systemen.

Projektions-Transformation Die Projektions-Transformation bildet alle Koordinaten auf die Bildebene ab. Um realistische Bilder zu erstellen, wird eine perspektivische Projektion verwendet, die von den internen Kamera-Parametern und dem Bildschirm abhängt. Diese werden im Rahmen einer Kalibrierung ermittelt. Z.B. ändern sich die Bildschirm- und Bildmaße beim Drehen eines Mobilgeräts. Die internen Kamera-Parameter werden in einer Matrix K angegeben. Sie hat die Form $K = \begin{pmatrix} f_x & 0 & x_0 & 0 \\ 0 & f_y & y_0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$. Die Werte f_x und f_y bilden die Brennweite als zwei Werte in Pixeln ab. x_0 und y_0 bilden einen Pixel-Offset zum optischen Mittelpunkt. K hat eine Relevanz beim Interpretieren von Tiefendaten und wird im Abschnitt 2.2.2 (Rekonstruktion von Punkten) verwendet.

Visuelle Effekte

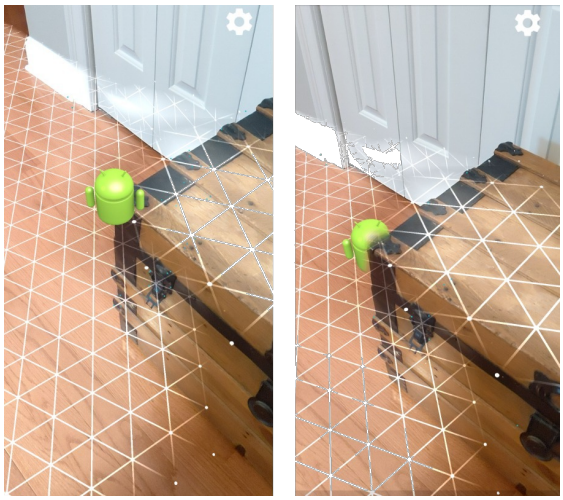


Abbildung 2.3: Beim linken Bild wurde die Verdeckung nicht berechnet. Beim rechten Bild wurde die Verdeckung berechnet und Teile des Android Maskottchens sind ausgeblendet [9].

Viele weitere Konzepte können beim Rendern eingebracht werden, um die Immersion in die AR-Szene zu erhöhen, wie z.B. die Verdeckung von virtuellen Objekten durch reale Objekte (Occlusion, Abb. 2.3), photometrische Registrierung, virtuelle Lichtquellen, Ausblendung von realen Objekten (Diminished Reality), eine stilisierte Darstellung [23].

Bei einem visuellen AR-System mit VST-Ansatz, ist die Verdeckung innerhalb realer Objekte natürlich gegeben. Die Verdeckung innerhalb virtueller Objekte wird beim Rendern bereits beachtet. Die Verdeckung eines virtuelles Objekt durch ein reales Objekt muss berechnet werden. Dafür müssen Tiefendaten oder eine geometrische Repräsentation der realen Szene bekannt sein z.B. durch 3D-Scans. Verdeckte Teile können transparent dargestellt werden.

2.1.3 Optisches Tracking

Ein Objekt zu tracken bedeutet, seine Pose möglichst genau und ohne Verzögerung aufzuzeichnen. Die Identifizierung von Zielobjekten (Objekterkennung) ist nicht Teil des Prozesses.

Die Pose muss aus Messungen geschätzt werden. Für das Tracking der Kamerapose auf einem Mobilgerät gibt es einige gängige Sensoren wie Beschleunigungssensoren, Magnetometer und Gyroskop. In spezieller Anordnung bilden diese Sensoren eine inertiale Messeinheit, welche in Mobilgeräten der Orientierungsmessung dient. Außerdem können Mobilgeräte oft mit WLAN und GPS ihre Position schätzen.

Diese Informationen allein reichen nicht für eine akkurate Registrierung. Mit Zunahme der hohen Informationsdichte von Kamerabildern kann optisches Tracking aus dem Bereich Computer Vision akkuratere Ergebnisse bieten, wenn Licht für ausreichende Helligkeit und Kontrast vorhanden ist. Für robusteres Tracking können Kameras mit weiteren Lichtspektren wie Infrarot oder Ultraviolett erweitert werden. Auf der einen Seite können sog. RGB-D Kameras, wie in der Microsoft Kinect eingebaut, Lichtmuster auf die Umgebung projizieren, um die Geometrie der Umgebung zu analysieren. Auf der anderen Seite kann mit Zeitmessung eine Abstandsmessung betrieben werden (Light imaging, detecting and ranging (Lidar)/Time-of-Flight (ToF) Kameras). In beiden Fällen können die geometrischen Rückschlüsse zur Positionsbestimmung der zu trackenden Kamera genutzt werden.

Nur wenige Mobilgeräte besitzen diese Sensoren. Stattdessen nutzen AR-Anwendungen auf Mobilgeräten meist markerloses Model-free Tracking wie z.B. visuelle Simultaneous Localization and Mapping (SLAM)-Systeme, die mit herkömmlichen Kameras funktionieren.

Model-based Tracking vs. Model-free Tracking

Um Kamerabilder zu interpretieren, können gegebene Daten z.B. eine geometrische Repräsentation, ein 3D-Modell oder ein Muster als Referenz dienen. Beim **Model-based** Tracking ist so eine Referenz bereits gegeben. Beim **Model-free** Tracking muss ansonsten eine Referenz während des Trackings erstellt werden z.B. via 3D-Scanning. Model-free Tracking ist sehr beliebt, weil kein extensives Vorwissen benötigt wird. Jedoch ist der benötigte Scan-Schritt nicht benutzerfreundlich und je nach Größe der Umgebung sehr aufwendig.

Marker vs. Natural Features

Es können im Bild vorkommende Feature (Natural Feature) oder künstliche Marker getrackt werden. Einige Oberflächeneigenschaften von Zielobjekten erschweren Natural Feature Tracking.

1. Einfarbigkeit und abwesende Textur erschweren die zuverlässige Identifizierung von Features.
2. Objekte mit hohem Glanz ändern ihr Erscheinungsbild stark bei Bewegung.
3. Repetitive Muster erhöhen die Mehrdeutigkeit der Position von Features.

Künstliche Marker können diese Schwierigkeit umgehen und erlauben robustes Tracking und robuste Identifikation von Zielobjekten. Jedoch ist die Hinzunahme eines Markers je nach Anwendungsfall unerwünscht.

Simultaneous Localization and Mapping

Model-free Natural Feature Tracking wird in der Robotik und anderen Bereichen als SLAM realisiert. Gleichzeitig wird das Problem selbst, eine unbekannte Umgebung zu kartieren und sich zu lokalisieren, SLAM genannt. SLAM-Systeme erzeugen inkrementell eine Karte der Umgebung (ein Mesh oder eine Punktwolke) und tracken gleichzeitig die Benutzerpose darin, was einen Scan der Umgebung erfordert.

SLAM-Algorithmen ähneln prinzipiell dem Visual Odometry Algorithmus [21, 23].

Grober Ablauf von Visual Odometry

- Pro Frame werden Feature erkannt.
- Für jedes Feature wird versucht dasselbe Feature in vergangenen Frames zu ermitteln.
- Mit mehreren Bildern aus verschiedenen Perspektiven und vielen Feature-Korrespondenzen kann die relative Kamerapose berechnet werden.
- Die Korrespondenzen werden genutzt, um die 3D Position eines Features zu triangulieren.

Wesentliche Verbesserungen von SLAM Algorithmen beinhalten die Reduzierung von akkumuliertem Drift mittels Bundle Adjustment, das Erkennen von Trackingverlust und dessen Neustart (sog. Relocalization), das Verhindern der Rekartierung von bekannten Bereichen (sog. Loop Closure). Relocalization und Loop Closure sind essenziell, um die Konsistenz und Verwendbarkeit der Tracking-Daten so lange wie möglich zu bewahren.

2.2 Tiefendaten

Tiefendaten sind Entfernungsschätzungen von einem Blickpunkt, wie einer Kamera, zu Punkten auf Objektoberflächen in der realen Szene und liegen als Punktwolke oder Tiefenbild vor. Die Daten stammen aus dedizierten Tiefensensoren (Infrarot, Lidar, ToF Kamera) oder Algorithmen für das optische Tracking wie SLAM. Auf Geräten mit einer einzelnen Kamera liegen die Daten als ein Tiefenbild (Abb. 2.4) aus Kamerasicht vor, wobei jedes Pixel keinen Farbwert, sondern eine Entfernungsschätzung repräsentiert.

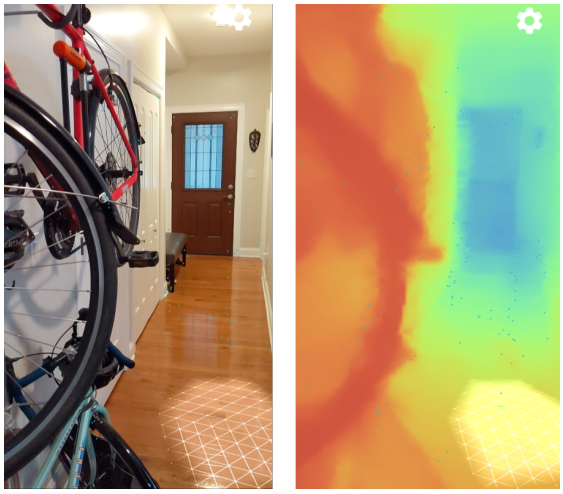


Abbildung 2.4: RGB Bild vs. Tiefenbild eines Flurs mit Fahrrad. Je röter, desto näher ist der Punkt an der Kamera. Je bläulicher, desto entfernter ist der Punkt [9].

2.2.1 Anwendungsfälle

Ein Tiefenbild wird üblicherweise direkt oder für eine Punktwolken-Rekonstruktion genutzt. Tiefenbilder können auch als Eingaben für komplexe Algorithmen dienen.

In direkten Pixelvergleichen kann ein Tiefenbild verwendet werden, um bspw. Verdeckung zwischen virtuellen Objekten (Pixel im Rendering) und der realen Szene (Pixel im Tiefenbild) zu berechnen. Da Tiefenbilder oft von niedriger Auflösung sind, muss für den direkten Vergleich Subsampling der Rendering-Pixel oder ein Upsampling des Tiefenbilds vorgenommen werden.

Ein Tiefenbild ergänzt fehlende Informationen bei der Rekonstruktion von Punktwolken, denn bei der Bildgebung gehen Informationen über die Tiefe verloren. Mit den Informationen aus dem Tiefenbild können die Punkte rekonstruiert werden, die ursprünglich auf das Tiefenbild abgebildet wurden. Das Ergebnis ist eine Punktwolke. Mit Punkten aus verschiedenen Winkeln können Oberflächen rekonstruiert werden. Dafür muss die Anwendung über einige Zeit Daten sammeln. Die Punktwolken werden dann vereinigt. Ggf. ist die Vereinigung eine komplexe Rekonstruktions-Pipeline, um die finale Punktwolkenqualität zu verbessern. Möglicherweise müssen doppelte Punkte oder Rauschen entfernt werden. Die Punktwolke kann weiterverarbeitet werden, um die Umgebung geometrisch abzubilden und hat viele Verwendungszwecke, z.B. für Visualisierungen, Objekterkennung oder Oberflächensimulation.

R. Du et al. haben in ihrem DepthLab viele Algorithmen vorgestellt und Anwendungsfälle demonstriert, um Tiefenbilder in Echtzeit zu nutzen [5]. Sie ermöglichen Interaktionsmöglichkeiten in den drei Kategorien Localized Depth, Surface Depth und Dense Depth. Localized Depth arbeitet mit dem Tiefenbild und erlaubt Abfragen der Entfernung und Normalvektor-Schätzungen beliebiger Bildschirm-Pixel. Damit sollen Messungen und einfache Kollisionsprüfungen ermöglicht werden. Surface Depth benutzt das Tiefenbild, um die Knoten von einem stark tesseliertes Rechteck zu verformen, um ein Mesh der Szenenoberfläche zu erzeugen. Das Verfahren ist ähnlich zur Rekonstruktion einer Punktwolke. Damit sollen ein erweitertes Umgebungsverständnis (durch Zugriff auf ein Mesh) wie Kollisionsprüfungen, weitere physikalische Simulationen und spezifischere visuelle Effekte ermöglicht werden. Dense Depth versucht die Auflösung der Tiefenkarte mit einem speziellen Antialiasing Algorithmus stark zu erhöhen. Damit können komplexere Effekte erzielt werden, wie die Simulation von virtuellen Lichtquellen.

DepthLab zeichnet sich vor allem durch Geschwindigkeit aus, da sämtliche vorgestellten Anwendungsfälle keine Vereinigung von Punktwolken benötigen. Andererseits ist das Tiefenverständnis lediglich für den derzeitigen Blickpunkt gültig.

2.2.2 Rekonstruktion von Punkten

Hier werden 3D Punkte als Spaltenvektor in homogenen Koordinaten repräsentiert und sollen mit einer Tilde gekennzeichnet werden. Es gilt $p = (\frac{x_1}{x_n}, \frac{x_2}{x_n}, \ldots, \frac{x_{n-1}}{x_n})^T$ und $\tilde{p} = (x_1, x_2, x_3, \ldots, x_n)^T$. Z.B. wird der Punkt $p_{Camera} = (x, y, z)^T$ durch Anhängen einer 1 zu $\tilde{p}_{Camera} = (p_{Camera}, 1)^T$. Umgekehrter Weise wird die letzte Komponente entfernt, und die verbliebenen Komponenten dadurch geteilt.

Die perspektivische Projektion von einem Punkt im Kamera-Koordinatensystem p_{Camera} zu einem Bildpunkt p_{Image} wird mit der Matrixmultiplikation mit den internen Kameraparametern K modelliert.

$$K \cdot \tilde{p}_{Camera} = \tilde{p}_{Image} = \begin{pmatrix} f_x & 0 & x_0 & 0 \\ 0 & f_y & y_0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \cdot f_x + z \cdot x_0 \\ y \cdot f_y + z \cdot y_0 \\ z \end{pmatrix}$$

Das ergibt den Bildpunkt $p_{Image} = \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} \frac{x \cdot f_x + z \cdot x_0}{z} \\ \frac{y \cdot f_y + z \cdot y_0}{z} \end{pmatrix}$.

Nun liefert ein Tiefenbild für jeden Bildpunkt p_{Image} eine Schätzung z' für z. Das ergibt $\tilde{p}'_{Image} = \begin{pmatrix} u \cdot z' \\ v \cdot z' \\ z' \end{pmatrix} \approx \begin{pmatrix} x \cdot f_x + z \cdot x_0 \\ y \cdot f_y + z \cdot y_0 \\ z \end{pmatrix}$.

Ersetzt man auf der rechten Seite z durch z' und löst nach x und y auf, kann p_{Camera} approximiert werden als ${p_{Camera}}' = \begin{pmatrix} \frac{z' \cdot (u - x_0)}{f_x} \\ \frac{z' \cdot (v - y_0)}{f_y} \\ z' \end{pmatrix} \approx \begin{pmatrix} x \\ y \\ z \end{pmatrix}$.

Mithilfe der Kamerapose kann ${p_{Camera}}'$ in Weltkoordinaten überführt werden.

2.3 Procedural Content Generation

Mit Procedural Content Generation (PCG) ist die automatische Erzeugung von Inhalten (Content) mithilfe von Algorithmen gemeint [27]. Damit ist ein PCG-Verfahren eine konkrete Lösung für ein Content-Generierungs-Problem. PCG findet Verwendung in unterschiedlichen Bereichen wie Film[2] oder Kunst. Der Großteil der Forschung findet im Kontext von Spielen statt.

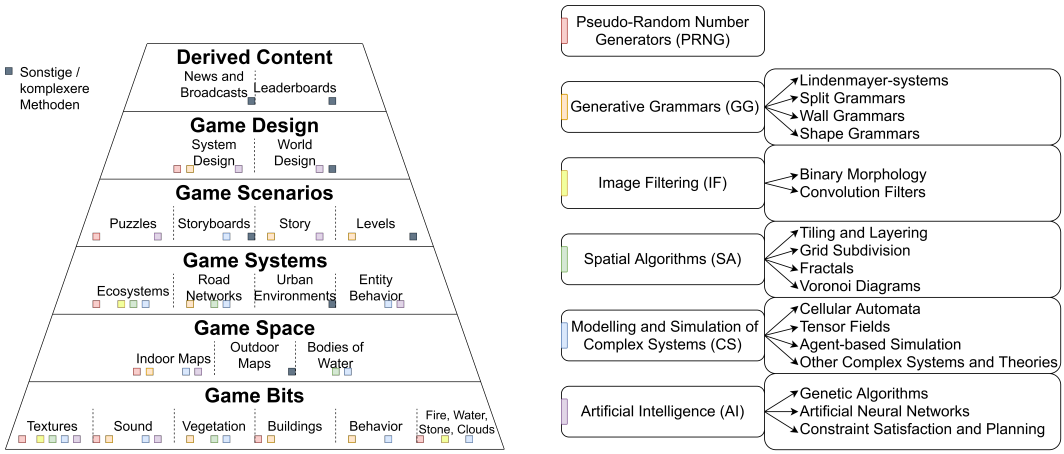
PCG-Verfahren werden aus verschiedenen Gründen entwickelt und eingesetzt, u.a. um Entwicklungszeit und -kosten zu sparen, den Wiederspielwert zu erhöhen, adaptive Spiele zu ermöglichen, Designer zu unterstützen sowie Kreativität und Spieldesign zu untersuchen [25, Kapitel 1].

2.3.1 Überblick über Game Content Gruppen

Hendrikx et al. haben in ihrer Studie generierbaren Content in sechs Gruppen aufgeteilt und als Pyramide hierarchisch angeordnet, welche Abb. 2.5a zeigt. Content aus höheren Gruppen können Content aus tieferen Gruppen enthalten. Die prozedurale Generierung der oberen Gruppen Game Scenarios, Game Design, Derived Content ist nicht gängig. Für jede Gruppe wurden wesentliche Typen an Content identifiziert z.B. für die Gruppe Game Space, die Typen Indoor Maps, Outdoor Maps, Bodies of Water [12]. Für jede

[2] Die Anwendung MASSIVE erstellt prozedural generierte Menschenmassen und wurde für 'Der Herr der Ringe' entwickelt https://www.massivesoftware.com/film.html (Abruf: 20.02.2023).

Gruppe könnte überlegt werden, wie sie in AR genutzt werden kann. Auch außerhalb von Spielen kann das Konzept der Hierarchie genutzt werden.



(a) Game Content Pyramide (b) Gängige PCG-Verfahren

Abbildung 2.5: (a) Diese Pyramide zeigt auf jeder Stufe eine Gruppe an generierbarem Game Content. Für jede Gruppe gibt es gängige Vertreter. Für jeden Vertreter ist farblich markiert, welches PCG-Verfahren gängig ist.
(b) Fünf Gruppen an PCG-Verfahren und einige Vertreter.
Beide Abbildungen sind adaptiert aus [12].

Game Bits sind elementare Bausteine, um Game Content zu erstellen, wie z.B. Texturen, Pflanzen, Gebäude. Sie agieren u.a. als Dekoration oder Items und werden am häufigsten generiert.

Game Space beinhaltet die Umgebung, in der das Spiel stattfindet. Dazu gehört auch Content, um in der Umgebung zu navigieren wie Treppen, Portale, Ein-, Ausgänge.

Mit **Game Systems** sind Modelle oder Simulationen gemeint, um das Spiel zu bereichern wie z.B. Simulation von Ökosystemen um Vegetation zu platzieren.

Game Scenarios sind Szenarien, die beschreiben wie und in welcher Reihenfolge Ereignisse im Spiel ablaufen.

Game Design bezeichnet einen Spielentwurf, welcher Spielregeln und Thema enthält.

Derived Content ist Content, der aus dem Zustand des Spiels entsteht. Z.B. Nachrichtenmeldungen, die sich auf die Aktionen des Spielers beziehen.

2.3.2 Überblick über PCG Verfahren

In derselben Studie wurden fünf Gruppen an gängigen PCG-Verfahren identifiziert, welche Abb. 2.5b auflistet.

Viele PCG-Verfahren benötigen indirekt Zufall. Deswegen sind **Pseudo-Random Number Generators (PRNG)** in den einzelnen PCG-Verfahren integriert und zählen in dieser Gruppierung nicht als eigenständige Gruppe. Perlin Noise ist ein sehr verbreitetes Verfahren das mit PRNG Rauschen erzeugt und weiterverwendet wird z.B. für Bild- und Terraingenerierung.

Generative Grammars / Grammatiken aus dem Bereich der formalen Sprachen werden in verschiedenen Formen genutzt. Weitverbreitete Verfahren sind Lindenmayer-Systeme, Shape Grammars um Vegetation bzw. Gebäude zu generieren.

Image Filtering umfasst mehrere Bildverarbeitungs-Verfahren, um Bilder zu analysieren oder neue Bilder mittels Filter zu generieren.

Die Gruppe **Spatial Algorithms** generiert Content mittels Manipulation bestehender Strukturen wie Gitter, Netze oder Oberflächen.

Modelling and Simulation of Complex Systems umfasst Theorien über komplexe Systeme. Z.B. die Simulation eines Ökosystems oder eine agentenbasierte Simulation für das Verhalten von Figuren.

Artificial Intelligence umfasst Machine Learning Verfahren, Constraint Satisfaction und Planungsprobleme sowie Suchalgorithmen wie z.B. genetische Algorithmen, Monte Carlo Tree Search.

2.3.3 Klassifikation von PCG Verfahren

Shaker et al. stellen eine Taxonomie vor, um PCG-Verfahren nach sieben Aspekten zu bewerten [25].

Online versus offline unterscheidet den Zeitpunkt des Aufrufs des PCG-Verfahrens. Online meint, dass der Content generiert wird während der Spieler spielt. Offline meint, dass der Content außerhalb der Spielzeit generiert wird. Vor dem Deployment oder vor einem Game Scenario.

Necessary versus optional unterscheidet die Notwendigkeit eines PCG-Verfahrens. Ein Verfahren kann notwendigen Content erstellen wie ein Game Scenario. Der Content ist dann zwingend korrekt, wohingegen optionaler generierter Content wie z.B. unterstützende Items nicht korrekt sein müssen.

Degree and dimensions of control meint, wie viel Kontrolle über Content ein PCG-Verfahren über seine Parameter bietet. Ein einzelner Seed für Pseudo-Random Number Generators (PRNG) bietet einen geringen Grad an Kontrolle, wohingegen viele oder komplexe Parameter, wie z.B. gesamte Strukturen bei Spatial Algorithms (SA), einen hohen Grad bieten.

Generic versus adaptive charakterisiert den erzeugten Content. Berücksichtigt ein PCG-Verfahren das Verhalten des Spielers bei der Content-Generierung, so erzeugt er adaptiven Content. Sonst ist der Content generisch.

Stochastic versus deterministic meint die Determiniertheit des Verfahrens. Ein determiniertes PCG-Verfahren erzeugt mit denselben Argumenten dieselbe Ausgabe. Bei einem stochastischen PCG-Verfahren ist die Ausgabe fast nicht reproduzierbar.

Constructive versus generate-and-test unterscheidet Generierungsansätze. Ein konstruktives PCG-Verfahren erzeugt den Content in einem Lauf. Ein Generate-and-Test PCG-Verfahren nutzt einen Test und erzeugt Content so lange, bis der Test bestanden ist.

Automatic generation versus mixed authorship charakterisiert, inwieweit ein Mensch mit dem Verfahren zusammenarbeitet. Üblicherweise integriert der Entwurf von PCG-Verfahren keine Interaktion mit einer Person (wie einen Game Designer), da die Content-Generierung automatisch erfolgen soll. Jedoch gibt es den Ansatz, die Person bei der Content-Generierung zu unterstützen. Solch ein PCG-Verfahren kann angefangenen Content vervollständigen oder verbessern und versichert Konsistenz mit Constraints.

2.4 Wave Function Collapse

Der Wave Function Collapse (WFC)[3] ist ein Bildgenerierungs-Algorithmus der aus kleinen Bildern, große Bilder generiert und wurde 2016 von M. Gumin entwickelt. Er kann

[3] Der Algorithmus ist sehr lose an das Konzept der Wellenfunktion und seinem Kollaps aus der Quantenmechanik angelehnt.

verwendet werden, um zusammensetzbare Bilder, sog. Kacheln (Tile) zu einem großen Bild zusammenzusetzen oder, um aus einem kleinen Beispielbild ein großes Bild zu generieren.

Im letzteren Fall generiert der Algorithmus aus einem Eingabebild ein Ausgabebild lokaler Ähnlichkeit zum Eingabebild (z.B. Abb. 2.6). Die lokale Ähnlichkeit wird über kleine Bildausschnitte ($N \times N$ Muster) definiert und erfüllt folgende zwei Bedingungen:

C1: Das Ausgabebild enthält nur $N \times N$ Muster, aus dem Eingabebild.

C2: Die Verteilung von $N \times N$ Muster einer ausreichend großen Anzahl an Ausgabebildern gleicht der Verteilung von $N \times N$ Mustern im Eingabebild.

Abbildung 2.6: Das Ausgabebild rechts ist lokal ähnlich zum Eingabebild links. Alle 3×3 Muster im Ausgabebild stammen aus dem Eingabebild. Das Ausgabebild enthält mehr graue Blöcke und Linien als schwarze Bereiche genau ähnlich zum Eingabebild. Das Bild stammt aus dem WFC GitHub Repo [10].

Beim WFC wird eine Struktur aufgebaut, die `wave` heißt und dieselben Dimensionen wie die Ausgabe besitzt. WFC erlaubt jeder $N \times N$ Region im Ausgabebild ein beliebiges $N \times N$ Muster des Eingabebilds zu enthalten. Dafür wird in `wave` für jede Region eine Liste an allen $N \times N$ Mustern verwaltet. Im ursprünglichen Algorithmus ist jeder Listeneintrag mit einem booleschen Wert (sog. Koeffizient) assoziiert, der bestimmt, ob das jeweilige $N \times N$ Muster für diese $N \times N$ Region erlaubt ist. Im Laufe des Algorithmus werden Koeffizienten auf `false` gesetzt. Eine Region ist unobserviert, wenn alle seine Koeffizienten `true` sind. Eine Region ist in einem definiten Zustand, wenn nur ein Koeffizient `true` ist. Wenn kein Koeffizient `true` ist, ist der Zustand widersprüchlich. Der Zustand eines Ausgabebild-Pixels ist abhängig von seiner Region und wird beschrieben als eine Superposition aus allen Pixeln der erlaubten Muster.

Der WFC Algorithmus besteht aus zwei Phasen, um alle Superpositionen zu einem definiten Zustand zu verkleinern (kollabieren). In der ersten Phase (Observation) wird eine $N \times N$ Region nach bestimmten Regeln ausgewählt. Der Zustand der Region wird mithilfe der Verteilung der $N \times N$ Muster im Eingabebild kollabiert, womit C2 statistisch erfüllt

wird. In der zweiten Phase (Propagierung) wird die Kollapsinformation propagiert, um die Konsistenz von Bedingung C1 aufrechtzuerhalten.

2.4.1 Algorithmus

Der WFC, wie von M. Gumin beschrieben [10]:

1. Lese Eingabebild und zähle $N \times N$ Muster.
    i. (Optional) Augmentiere Datensatz mit Rotationen und Spiegelungen.
2. Sei `wave` ein Array mit den Dimensionen vom Ausgabebild. Jedes Element von `wave` repräsentiert den Zustand einer $N \times N$ Region im Ausgabebild. Solch ein Zustand ist eine Liste aller $N \times N$ Mustern, zusätzlich mit booleschen Koeffizienten assoziiert. Koeffizient `false` bzw. `true` bedeutet, dass das $N \times N$ Muster in der $N \times N$ Region nicht erlaubt bzw. erlaubt ist.
3. Initialisiere `wave` als unobserviert, d.h. alle booleschen Koeffizienten in `wave` sind `true`.
4. Wiederhole folgende Schritte:
    i. Observation
        a) Finde das Element in `wave` mit der kleinsten Entropie ungleich 0. Gibt es keins, geh zu Schritt 5.
        b) Kollabiere das Element in einen definiten Zustand, anhand seiner Koeffizienten und der Häufigkeitsverteilung der Muster.
    ii. Propagierung: Propagiere die Veränderung aus der Observation.
5. Entweder sind nun alle Elemente in `wave` observiert, dann gebe `wave` als Ergebnis zurück. Oder die Elemente sind in einem widersprüchlichen Zustand, dann gebe nichts zurück.

Bei der Propagierung (Schritt 4b) wird geprüft, ob Elemente aus `wave` in Konflikt zu C1 stehen. Dafür werden in vielen Varianten vom WFC in Schritt 1 Nachbarschaftsregeln berechnet. $N \times N$ Muster, die sich um ein Pixel versetzt, überlappen, können benachbart sein ohne C1 zu verletzen.

D.h. die Koeffizienten von nicht benachbarten $N \times N$ Mustern in zwei benachbarten Elementen aus `wave`, können bei der Propagierung auf `false` gesetzt werden.

Im Abschnitt 3.3.1 wird beschrieben, wie `wave` in Schritt 5 initialisiert werden kann, um mehr Kontrolle über die Ausgabe zu erhalten.

2.4.2 Modelle im Algorithmus

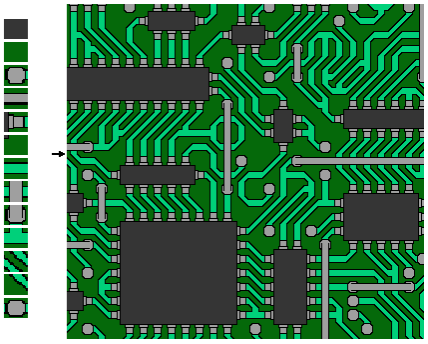


Abbildung 2.7: Der WFC im Simple Tiled Model erzeugt Ausgabebilder aus mehreren Eingabebildern.

Der WFC kann mit zwei Modellen betrieben werden. Das **Overlap Model** (Überlappungsmodell) unterscheidet sich nicht vom vorgestellten Algorithmus und zeichnet sich dadurch aus, dass die Nachbarschaftsregeln aus dem Eingabebild berechnet werden (Abb. 2.6). Beim **Simple Tiled Model** (einfaches Kachelmodell) muss der Benutzer ein Tileset (Menge mit zusammensetzbaren Bildern) und Nachbarschaftsregeln übergeben. Nachbarschaftsregeln werden nicht berechnet. Das Tileset selbst stellt die $N \times N$ Muster dar und kann mehrere Bilder umfassen (Abb. 2.7). Da viele Kacheln symmetrisch sind, kann die Anzahl an Nachbarschaftsregeln sehr lang werden. Beim Simple Tiled Model kann man für jede Kachel eine Symmetrie angeben, sodass weniger Nachbarschaftsregeln angeben werden müssen.

2.4.3 Modifikationen

Unabhängig davon, welches Modell gewählt wird, übersetzt der Algorithmus ein Textursynthese-Problem in ein Constraint-Satisfaction-Problem. Der WFC baut stark auf die Beispielbasierten Textursynthese-Algorithmen Synthese mittels Markov-Ketten [6] und Model

Synthesis [16] auf. Für das Propagieren der Constraints wird im WFC-Original AC-4 (Arc Consistency Algorithm 4) [20] genutzt.

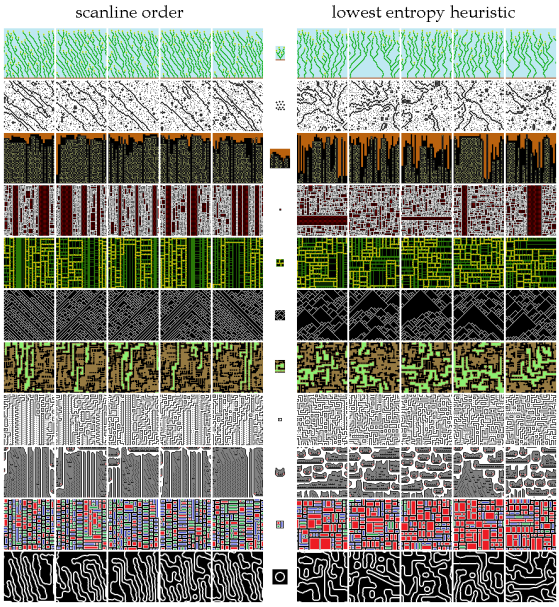


Abbildung 2.8: Verschiedene Observations-Heuristiken führen Bias bei der Musterverteilung ein. Auf diesen elf Beispielen ist links weniger Variation zu sehen als rechts. Links wird eine Heuristik genutzt, bei der die erste Zelle mit Entropie ungleich 0 ausgewählt wird. Rechts wird die minimale Entropie Heuristik genutzt. Das Bild stammt aus dem GitHub Repo [10].

Außerdem nutzt der WFC im Original eine minimale Entropie Heuristik für die Observation. Es gibt viele Varianten, die unterschiedliche Verfahren für die Propagierung und Heuristiken nutzen, wobei unterschiedliche Heuristiken zu Biases in der Musterverteilung in den Ausgabebildern führen können (Abb. 2.8).

Der WFC kann auch angepasst werden um in höheren Dimensionen oder mit anderen Datentypen als Bildern zu arbeiten. In dieser Arbeit wird ein generischer WFC genutzt, welcher in 1D bis 3D arbeiten kann und generische benutzerdefinierte Daten akzeptiert, wie z.B. Voxel oder Strings.

2.5 Themenverwandte Arbeiten

Azad et al. stellen zwei Prototypen in VR für AR-Platformer-Spiele vor, in denen hoch adaptive Level generiert werden [3]. Das Generate-and-Test PCG-Verfahren nutzt Heuristiken über nutzbare Spielfläche und den Spieler für die Levelgestaltung. Die Spielfiguren navigieren eigenständig durch die Umgebung und reagieren auf platzierte Objekte. Ein Umgebungsscan via Kinect 2.0[17] wird vom Benutzer erfordert.

Holo Worlds Infinite [15] ist ein AR-Projekt für die Microsoft HoloLens[19], dass den freien Platz auf dem Fußboden mit einem prozedural generierten Strassennetzwerk und Gebäuden füllt. Das Raumverständnis ist durch mehrere Tiefensensoren der Microsoft HoloLens gegegeben. Die Strassennetzwerk-Generierung erfolgt über L-Systeme.

Im Rahmen der Bachelorarbeit von J. Helbig [11], wurde eine AR-Anwendung entwickelt, die ein Smartphone als VST-Display nutzt um reale Ebenen mit einem Gleis und Zug zu erweitern. Aus einem vorliegendem Satz an Gleisteilen und einer Zeichnung des Gleisverlaufs durch den Benutzer, kann die Anwendung ein Gleissystem erstellt. Im Gegensatz zu dieser Arbeit nutzt die Generierung keine Tiefendaten.

Sra et al. stellen ein VR-Projekt vor, in dem ein Raum-Scan als Grundlage für prozedural generierte Welten dient. Der Scan wird nach begehbaren Flächen und Hindernissen analysiert. Für das Projekt wird ein Tablet mit RGB-D Kamera zuerst für den Scan und dann als HMD genutzt [26]. Viele Projekte erforschen die Erhöhung der Immersion in VR durch die reale Welt.

S. Hugo und C. Verbrugge haben den WFC leicht modifiziert, um im Overlap Model Pfade einfacher zu erzeugen. Es werden Vor- und Nachverarbeitungsschritte vorgestellt und beschrieben, inwiefern Eingabebilder die erzeugten Pfade beeinflussen können [24].

In der Bachelorarbeit von F. Rett [22] wurde ein Verfahren entwickelt, mit dem ein tieferes Verständnis der realen Umgebung ermöglicht werden soll. Die Pipeline nutzt Tracking Daten, um Meshes von realen Objekten zu erstellen. Die Pipeline stellt für diese Arbeit eine Möglichkeit dar, um Hindernisse zu erkennen.

3 Konzeption

Dieses Kapitel stellt die Anforderungen an die Anwendung und die Ansätze für die Umgebungsanalyse und Gleisgenerierung vor.

Die Anwendung akkumuliert eine Punktwolke aus Tiefendaten, approximiert durch Clustering Oberflächen von Objekten und projiziert sie auf die Gleisebene, um eine grobe diskretisierte Schätzung der Hindernisgrundflächen zu erhalten.

Anpassungen an den WFC ermöglichen die Übergabe eines Initialzustands. Der Initialzustand modelliert die gerasterten Grundflächen auf der Gleisebene als 'Leer'-Kachel.

3.1 Anforderungen

Die mobile Anwendung soll wie angesprochen vier Aufgaben erfüllen, um prozeduralen Content und Umgebung bzgl. Gleisverlauf und Platzierung eng zu koppeln. Sie muss ein visuelles **AR-System bieten**, ein **PCG-Verfahren nutzen**, **Umgebungsdaten** für das PCG-Verfahren **aufbereiten** und den generierten **Content anzeigen**.

Das folgende beispielhafte Anwendungsszenario soll Anforderungen an die Anwendung veranschaulichen. Der Benutzer ist in einem beleuchteten Raum, in dem sich ein Tisch befindet. Auf dem Tisch steht eine Flasche.

1. Der Benutzer startet die Anwendung auf seinem Mobilgerät.
2. Die Anwendung zeigt einen Umgebungsausschnitt als Video.
3. Die Anwendung weist den Benutzer an, das Gerät auf eine ebene Fläche zu richten und sich um diese Ebene zu bewegen.
4. Der Benutzer richtet das Gerät auf den Tisch und bewegt sich um den Tisch herum.
5. Der Benutzer drückt einen Knopf, um die Gleisgenerierung zu starten.

6. Nach einiger Zeit erscheinen virtuelle Gleise auf dem Tisch. Ein virtueller Zug beginnt auf einem Gleis zu fahren.
7. Das Gleis verläuft nicht in die Flasche.
8. Bewegt sich der Benutzer, dann scheinen Gleis und Zug wirklich auf dem Tisch zu liegen.

Zusammenfassend erstellt die Anwendung Gleise auf Freiflächen, und versucht sie wie reale Objekte anzuzeigen. Punkt 7 verdeutlicht, dass die Anwendung adaptiven Content erzeugt, der die Umgebung berücksichtigt. Dafür erkennt die Anwendung die Tischoberfläche und Objektgrundflächen und benötigt einen manuellen Umgebungsscan durch den Benutzer (Punkt 1 bis Punkt 4). Für die Benutzbarkeit werden dem Benutzer Nachrichten angezeigt, da aus Benutzersicht während dieser Zeitspanne nur eine Videowiedergabe der Umgebung abgespielt wird.

Es wurden sechs funktionale Anforderungen und drei nichtfunktionale Anforderungen ermittelt, die in der Tabelle 3.1 (mit R und Q) aufgelistet sind und erläutert werden.

ID	**Funktionale Anforderung**
R1	Das System generiert ein Gleis mittels des Wave Function Collapse Algorithmus
R2	Das System berücksichtigt Hindernisse bei der Gleisgenerierung
R3	Das System erhebt Tiefendaten und erkennt daraus Grundflächen
R4	Das System bietet dem Nutzer ein Augmented Reality Erlebnis
R5	Das System berücksichtigt die Verdeckung virtueller Objekte beim Zeichnen
R6	Das System erkennt Ebenen
ID	**Nichtfunktionale Anforderung**
Q1	Das System läuft mit einer konsistenten Framerate zwischen 10 und 30 FPS
Q2	Das System unterweist den Benutzer in die Verwendung der Anwendung
Q3	Die Gleisgenerierung ist zeitlich begrenzt auf 1 Minute

Tabelle 3.1: Anforderungen

R1: Das System generiert ein Gleis mittels des Wave Function Collapse Algorithmus
Das System ist in der Lage mit dem WFC ein Gleis zu generieren.

Hier soll 'Gleis' für eine Abstraktion eines realen Gleises stehen. Ein Gleis besteht aus einer graphischen Repräsentation und einer Pfadkomponente. Die Repräsentation zeigt Querlatten und zwei Schienen, die einen Pfad entlang laufen. Die Pfadkomponente soll diesen Pfad modellieren. Es besteht kein Anspruch auf eine realistische Nachempfindung der Gleise.

Der WFC soll im Simple Tiled Model betrieben werden, also müssen u.a. Tileset und Nachbarschaftsregeln übergeben werden. Das Tileset wird optisch Querlatten und Schienen enthalten. Die Nachbarschaftsregeln müssen ein optisches Zusammenhängen erzwingen. Der Pfad, auf dem der Zug fahren soll, kann nicht direkt vom WFC generiert werden und muss separat berechnet werden.

R2: Das System berücksichtigt Hindernisse bei der Gleisgenerierung
Das gewählte PCGVerfahren soll mit Umgebungsdaten parametrisiert werden, z.B. mit Positionen und Oberflächen von Hindernissen. Beim WFC kann dafür eine Anpassung von S. Hugo und C. Verbrugge [24] übernommen werden, bei der ein Initialzustand übergeben werden kann. So kann verhindert werden, dass Zellen, die Hindernisse enthalten, zu Gleisabschnitten kollabieren. Die betrachtete reale Szene soll statisch sein, d.h. für ein Anwendungsszenario werden keine Objekte hinzugefügt oder entfernt.

R3: Das System erhebt Tiefendaten und erkennt daraus Grundflächen
Die Tiefendaten sollen genutzt werden, um das Umgebungsverständnis zu verbessern und eine enge Kopplung zwischen prozeduralem Content und Umgebung herzustellen. Indem aus den Tiefendaten Grundflächen erkannt werden, können Platzierung und Gleisverlauf von Content an die Umgebung angepasst werden. Das Berechnen der Tiefendaten übernimmt die AR Bibliothek.

Die Tiefendaten von ARCore haben eine theoretische Reichweite von 65 Metern, eine optimale Genauigkeit zwischen 50 cm und 15 m von der Kamera entfernt, mit zuverlässigen Tiefenschätzungen bis 25 m. Dieser Bereich genügt für diese Anwendung.

Das Erkennen der Grundflächen kann durch verschiedene Verfahren umgesetzt werden. Ein Objekt sollte größer sein als ein Würfel mit 3 cm Kantenlänge (willkürliches Mindestmaß), damit seine Grundfläche erkannt wird.

R4: Das System bietet dem Nutzer ein Augmented Reality Erlebnis
Es soll eine AR-Anwendung entwickelt werden, folglich muss sie die drei Eigenschaften aus Abschnitt 2.1 (Augmented Reality) aufweisen. Die Anwendung soll offline auf einem Mobilgerät betrieben werden. Visuelle AR-Systeme, die den VST-Ansatz nutzen haben sind nicht ergonomisch und bieten relativ kurze AR-Erlebnisse, da der Benutzer das Mobilgerät konstant auf etwa Augenhöhe tragen muss. Zudem wird die Anwendung viel Leistung nutzen, trotz geringerer Tracking Genauigkeit und die Erwärmung des Mobilgeräts kann unangenehm sein. Diese Arbeit verfolgt nicht einen Prototyp zu entwickeln, der ein angenehmes AR-Erlebnis bietet.

R5: Das System berücksichtigt die Verdeckung virtueller Objekte beim Zeichnen
Eine AR-Anwendung muss eine Illusion einer erweiterten Realität aufrechterhalten. Sie ist auf eine AR-Bibliothek angewiesen, um virtuelle Objekte über die Zeit konsistent an einer designierten Position, in der realen Welt registriert, zu zeichnen.

Um die Illusion zu verbessern, soll die Verdeckung von virtuellen Objekten durch reale Objekte mit Tiefendaten berechnet werden. Ansonsten ist zu erwarten, dass die Anwendung in vielen weiteren Aspekten optisch nicht korrekt erscheinen wird. Bspw. wird ein, auf dem Display sichtbarer, Spiegel keine Reflexion von virtuellen Objekten anzeigen.

R6: Das System erkennt Ebenen
Die Anwendung beschränkt sich auf das Erweitern auf Ebenen wie Tischen, Fluren, Wänden. Die Ebenenerkennung soll von der AR-Bibliothek übernommen werden und als Ergebnis die Ebene und ihre Eckpunkte ausgeben.

Q1: Das System läuft mit einer konsistenten Framerate zwischen 10 und 30 FPS
Das System soll eine konsistente relative niedrige Framerate beibehalten und eine sehr dynamische Framerate verhindern, z.B. durch das Nutzen von asynchronen statt synchronen Prozessen. Eine dynamische Framerate zeigt sich als starkes Stocken der Anwendung und stört den visuellen Fluss der Anwendung und das Benutzererlebnis erheblich.

Q2: Das System unterweist den Benutzer in die Verwendung der Anwendung
Die Anwendung weist den Benutzer mithilfe von Benachrichtigungen und visuellen Clues an, wie das AR-Erlebnis zu navigieren ist.

Q3: Die Gleisgenerierung ist zeitlich begrenzt auf 1 Minute
Die Gleisgenerierung sollte innerhalb einer vorgegebenen Zeit stattfinden. Diese Qualität ist teilweise von der verwendeten WFC Implementierung abhängig. Sie hat eine geringe Priorität, aber die Generierung von Content sollte eine obere Laufzeitgrenze haben.

3.2 Umgebungsanalyse

Im Abschnitt 2.5 (Themenverwandte Arbeiten) wurde ein Verfahren für die Oberflächenrekonstruktion angesprochen, dass auf mobilen Geräten läuft. F. Rett hat in seinem Verfahren Punktwolken aus Feature-Punkten vereinigt und bereinigt, ein Clustering Verfahren angewendet sowie die Clusteroberflächen trianguliert [22]. Dieser Ansatz wird in dieser Arbeit mit einigen Unterschieden verfolgt, um Objektgrundflächen und damit Freiflächen für das Gleis zu erkennen.

Punktwolken aus Feature-Punkten sind in ARCore spärlich mit ca. 200 Punkten. Dafür enthalten sie identifizierbare Feature und können robust vereinigt werden. Ein Tiefenbild mit einer üblichen Auflösung von 160×120 in ARCore enthält 19200 Punkte und wird für meist (Abschnitt 2.2) in einem aufwendigen Verfahren vereinigt und bereinigt, weil viele Punktduplikate angesammelt werden können.

Hier wird eine Punktwolke aus Tiefenbildern akkumuliert, ein Clustering-Verfahren angewendet, die Cluster projiziert und die Clustergrundflächen approximiert.

3.2.1 Algorithmus

Die Umgebungsanalyse besteht aus der Punktwolkenakkumulation und Grundflächenerkennung.

Die Punktwolkenakkumulation erhält als Eingabe eine Punktwolke, ein Tiefenbild, interne Kameraparameter, eine Ebene und einen Mindesthöhenabstand y_{min} und erzeugt

Algorithm 1 Punktwolkenakkumulation

Eingabe: Punktwolke, Tiefenbild, interne Kameraparameter, Kamerapose, Ebene, y_{min}
Ausgabe: Punktwolke

1: Punktwolke aus Tiefenbild und Kamerapose rekonstruieren
2: Punktwolken vereinigen
3: Punkte unter der Ebene filtern
4: Punkte mit Abstand zur Ebene kleiner als y_{min} filtern **return** Punktwolke

als Ausgabe eine Punktwolke. Über mehrere Frames wird eine Punktwolke akkumuliert. Punkte unter der Ebene, und nahe der Ebene werden gefiltert. Ein Punkt ist nah an der Ebene, wenn der Abstand kleiner als y_{min} ist. Hier werden Ebenen nach ARCore-Konvention auf der XZ-Ebene definiert. In lokalen Koordinatensystemen von Ebenen entsprichen Y-Koordinaten somit dem vertikalen Abstand zu einer Ebene. Das soll verhindern, dass beim Clustering bei der Grundflächenerkennung, die Cluster über solche Punkte verbunden sind.

Algorithm 2 Erkennen und Rasterung von Grundflächen

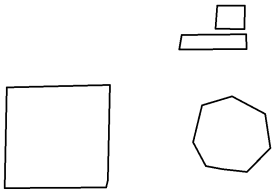
Eingabe: Punktwolke, Ebene, (Zeilenanzahl, Spaltenanzahl, Zellenhöhe, Zellenbreite), α
Ausgabe: Karte

1: Karte erstellen
2: Cluster in Punktwolke finden
3: Cluster auf Ebene projizieren
4: Polygone aus Projektionen erstellen
5: Polygone rastern und Karte befüllen **return** Karte

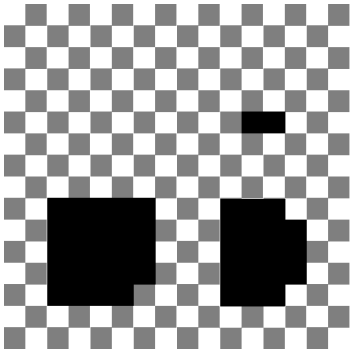
Die Grundflächenerkennung erhält als Eingabe eine Punktwolke, eine Ebene, eine gewünschte Auflösung und α und erzeugt als Ausgabe eine Karte. Es wird eine Karte mit der gewünschten Auflösung erstellt, die einer Draufsicht der Ebene mit über gelegtem Gitter entspricht. Dann werden Objekte als Cluster in der Punktwolke approximiert. Jedes Cluster wird auf die Ebene projiziert. Anschließend wird für jede Projektion ein einschließendes Polygon berechnet. Hier werden Alpha Shapes verwendet. Das sind Hüllen, die vom Parameter α abhängen. α bestimmt, wie eng die Hülle an den projizierten Punkte anliegt. Ein Wert von 1 berechnet die konvexe Hülle, ein Wert von 0 eine Hülle, die einem Spannbaum ähnelt. Jedes Polygon wird als Gitterzellen in entsprechender Auflösung diskretisiert. Entsprechenden Elementen in der Karte wird der Wert 1 zugewiesen.

Hier wurde ein gitter-basiertes Clusteringverfahren genutzt. Für die Rasterung der Polygone wurde der Kantenlisten-Algorithmus Polygon-Scanline verwendet.

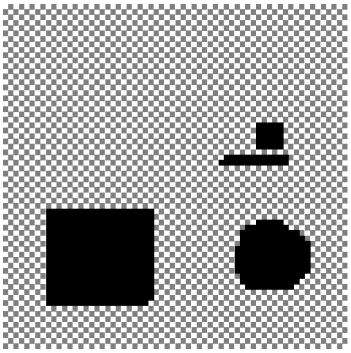
Auflösung der Hinderniskarte Die Hinderniskartenauflösung bestimmt die Gleisauflösung, wird durch Zeilen- und Spaltenanzahl und Zellendimension bestimmt und hat einen direkten Einfluss auf die Optik des Gleises.



(a) Ein Teilergebnis der Umgebungsanalyse sind Polygone.

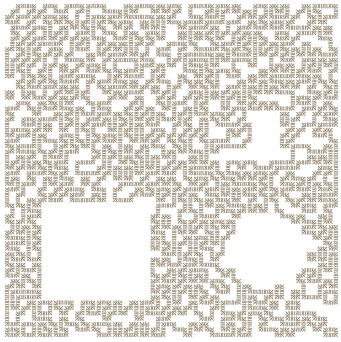


(b) Hinderniskarte mit einer Auflösung von 16 Zeilen und 16 Spalten.



(c) Hinderniskarte mit einer Auflösung von 64 Zeilen und 64 Spalten.

(d)



(e)

Je niedriger die Auflösung (weniger, größere Zellen), desto mehr Platz nehmen Hindernisse und Gleisteile ein und desto gröber werden die Grundflächen. Abb. 3.1b und Abb. 3.1d zeigen eine niedrige Auflösung. Je höher die Auflösung (mehr, kleinere Zellen), desto weniger Platz nehmen Hindernisse und Gleisteile ein und desto feiner werden die Grundflächen. Das Gleis kann wie auf Abb. 3.1e labyrinthartig erscheinen.

3.3 Gleisgenerierung

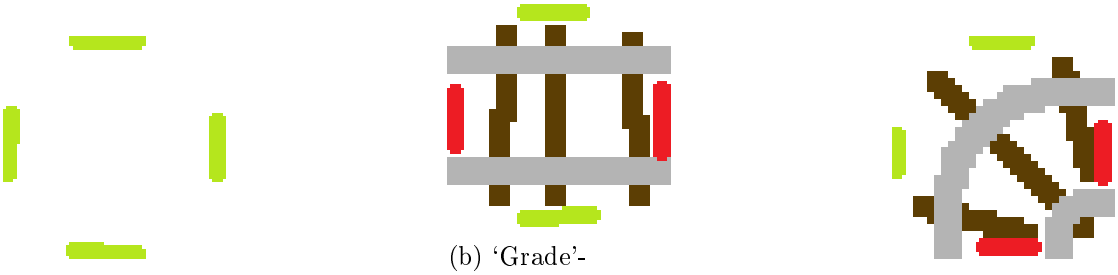
Die Gleisgenerierung erhält als Eingabe die WFC-Parameter sowie einen Initialzustand und erzeugt als Ausgabe eine Kachelanordnung und einen Pfad.

3.3.1 Anpassungen an den WFC

Die Eingabe des WFC wurde um einen Initialzustand erweitert und die Ausgabe auf die Kachelanordnung reduziert. Der Initialzustand besitzt dieselben Dimensionen, wie die Ausgabe und enthält als Elemente `Null` oder eine definite Kachel. Die Kachelanordnung ist abstrakter, als ein Bild und kann auch für die Pfadgenerierung genutzt werden.

Der WFC-Algorithmus wird nocheinmal aufgelistet, wobei der Zusatz vom Initialzustand `intialstate` fett markiert ist.

1. Lese Eingabebild und zähle $N \times N$ Muster.
    i. (Optional) Augmentiere Datensatz mit Rotationen und Spiegelungen.
2. Sei `wave` ein Array mit den Dimensionen vom Ausgabebild. Jedes Element von `wave` repräsentiert den Zustand einer $N \times N$ Region im Ausgabebild. Solch ein Zustand ist eine Liste aller $N \times N$ Mustern, zusätzlich mit booleschen Koeffizienten assoziiert. Koeffizient `false` bzw. `true` bedeutet, dass das $N \times N$ Muster in der $N \times N$ Region nicht erlaubt bzw. erlaubt ist.
3. Initialisiere `wave` mit `intialstate`, d.h. alle booleschen Koeffizienten in `wave` sind `true`, wenn das entsprechende Element in `intialstate` = `Null` ist. **Andernfalls repräsentiert das Element in `intialstate` einen definiten Zustand. Kollabiere das Element in `wave` mit dem Zustand.**
    i. **Sind die Elemente in einem widersprüchlichen Zustand, dann gebe nichts zurück.**
4. Wiederhole folgende Schritte:
    i. Observation
        a) Finde das Element in `wave` mit der kleinsten Entropie ungleich 0. Gibt es keins, geh zu Schritt 5.



(a) 'Leer'-Kachel, Symmetrie: X

(b) 'Grade'-Kachel, Symmetrie: I

(c) 'Kurve'-Kachel, Symmetrie: L

Abbildung 3.2: Dieses Tileset enthält drei Kacheln, um Gleise zu generieren. Um die Nachbarschaftsregeln zu veranschaulichen, wurden die Kanten eingefärbt. Sich berührende Kanten müssen mit derselben Farbe markiert sein. Grün markierte Kanten stellen Leere dar und rot markierte Kanten stellen Gleisverbindungen dar.

b) Kollabiere das Element in einen definiten Zustand, anhand seiner Koeffizienten und der Häufigkeitsverteilung der Muster.

ii. Propagierung: Propagiere die Veränderung aus der Observation.

5. Entweder sind nun alle Elemente in `wave` observiert, dann gebe `wave` als Ergebnis zurück. Oder die Elemente sind in einem widersprüchlichen Zustand, dann gebe nichts zurück.

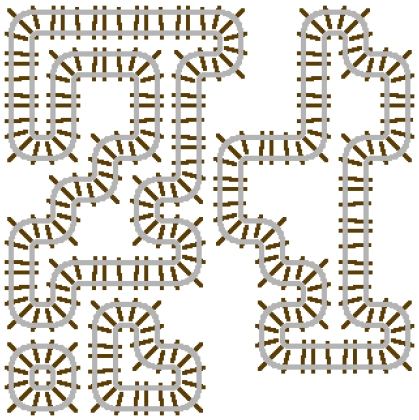
Nun kann mit `initialstate` ein Anfangszustand erzwungen werden, der durch die Hinderniskarte bestimmt wird.

3.3.2 Gleise mittels WFC erstellen

Ein Bild eines Gleises kann mit dem folgendem Tileset erstellt werden. Das Tileset besteht aus drei Tiles. Bestimmte Nachbarschaftsregeln erzwingen zusammenhängende Gleisabschnitte. Bei der Zeichnung des Gleises werden die 3D Modelle verwendet.

Beispiele für Pfade, die dadurch erzeugt werden können, sind auf Abb. 3.3a und Abb. 3.3b abgebildet. Dieses Tileset versucht geschlossene Gleisabschnitte zu erzwingen, indem jeder Gleisabschnitt eine gerade Anzahl an Verbindungsstellen besitzt. Nur eine Kachel am Bildrand kann einen offenen Pfad anfangen (bspw. Abb. 3.3a).

(a) Dieses Ausgabebild zeigt ein Gleis, das über den Bildrand verläuft.



(b) Wird der Bildrand mit 'Leer'-Kacheln befüllt, erzwingt das geschlossene Gleise.

Um dies zu verhindern, wird dem angepassten WFC eine Hinderniskarte übergeben, bei der der Bildrand mit 'Leer' Tiles gefüllt ist.

3.3.3 Pfadextraktion

Die Pfadextraktion erhält als Eingabe eine Kachelanordnung sowie Kurven für jede Kachel und erzeugt als Ausgabe einen Pfad. In einem ungerichteten Graph werden für jede Zelle Knoten und Kanten anhand der entsprechenden Kurve hinzugefügt. Der längste Pfad wird als Ergebnis ausgegeben. Nun kann die Zugpose linear interpoliert werden.

Die Kurven werden als Punktliste mit Koordinaten zwischen 0 und 1 übergeben (z.B. Abb 3.4). Die Zugposition könnte auch mit anderen Interpolationen berechnet werden, wobei sicherlich andere Kurvenkoordinaten hinterlegt werden müssen.

Andere Ansätze könnten auf Analysen der Meshes oder Bildanalysen der Texturen aufbauen, um damit explizite Pfaddefinitionen zu vermeiden. Bspw. haben S. Hugo und C. Verbrugge bei ihrer Pfadgenerierung mit dem WFC Kantendetektion auf dem Ausgabebild genutzt.

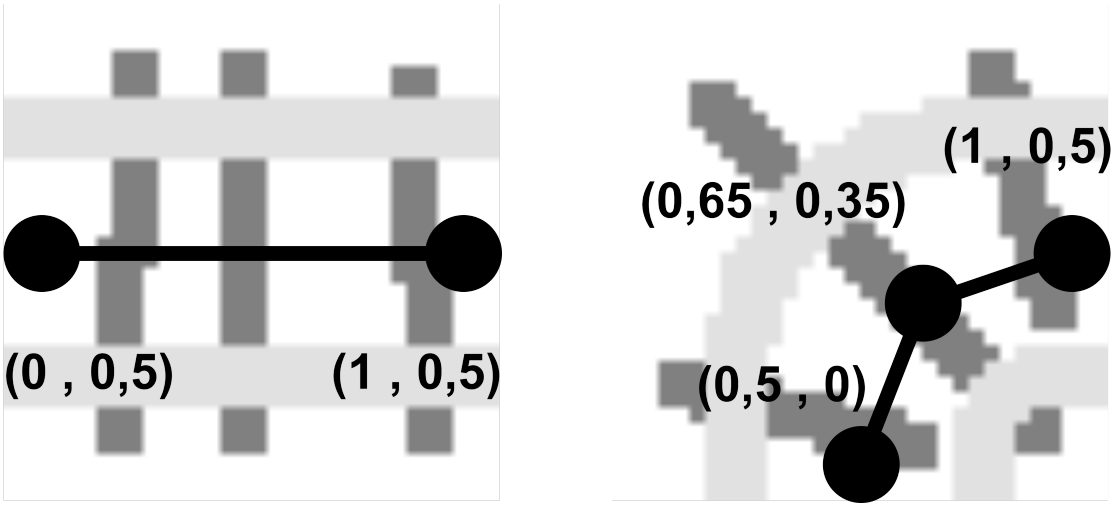


Abbildung 3.4: Beispielhafte Kurvendefinitionen für die Kacheln 'Gerade' und 'Kurve'.

3.3.4 Gleiszeichnung

Algorithm 3 Gleiszeichnung

Eingabe: arrangement, tiles, plane, depthImage
Ausgabe:

```
1: for x ← 0 to length(arrangement.columns) do
2:     for y ← 0 to length(arrangement.rows) do
3:         planePoint ← Point(x,y) ∗ plane.dimension
4:         translation ← transformPlanePointToWorldSpace(planePoint, plane)
5:         rotation ← plane.rotation
6:         pose ← Pose(translation, rotation)
7:         tile ← tiles[arrangement[x,y]]
8:         drawObject(tile, pose, depthImage)
9:     end for
10: end for
```

Das Gleis wird gemäß der Kachelanordnung gezeichnet. Der Ablauf wird als Pseudocode in Algorithmus 3 beschrieben. Da das Gleis auf der Ebene definiert wurde, wird für jede zu zeichnende Kachel die Ebenen-Koordinaten ermittelt. Mithilfe der Modell-Transformation der Ebene können die Koordinaten in das Weltkoordinatensystem überführt werden. An dieser Pose wird das entsprechende 3D-Modell der Kachel gezeichnet.

4 Software

Dieses Kapitel stellt den Kontext, die Struktur und das Verhalten der Anwendung vor. Abschließend wird die Implementierung besprochen.

4.1 Systemkontext

Die Anwendung bietet das Starten und Schließen der Anwendung, ein AR-Erlebnis, das Starten der Gleisgenerierung und das Konfigurieren der Umgebungsanalyse und Gleisgenerierung. Die GUI bietet dafür zwei Knöpfe. Die Tabelle 4.1 ordnet den Usecases die Anforderungen zu.

ID	zugehörige Anforderungen	Usecase
UC1	R4	Anwendung starten
UC2	R4	Anwendung stoppen
UC3	R1, R2, R3, R4, R5, R6	Erweiterte Umgebung navigieren und sehen
UC4	R1, R2	Gleisgenerierung starten
UC5	R1, R3	Anwendung konfigurieren

Tabelle 4.1: Jedem Usecase wurden die zugehörigen Anforderungen zugeordnet

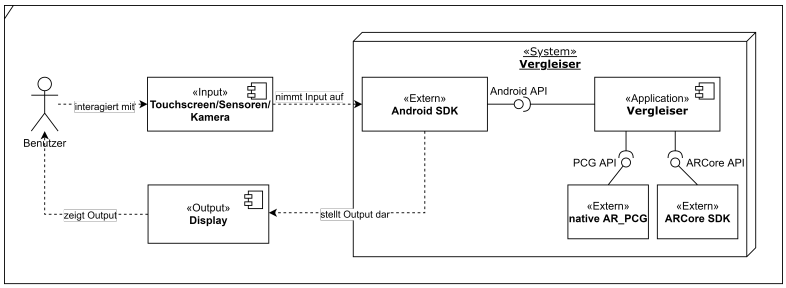


Abbildung 4.1: Die Anwendung ‘Vergleiser’ nutzt einige Bibliotheken.

Der Benutzer interagiert über den Touchscreen und den integrierten Sensoren mit der Anwendung. Abb. 4.1 zeigt die Anwendung als Blackbox. Vor allem nutzt die Anwendung die Android SDK, ARCore SDK und eine PCG Bibliothek. Die PCG-Bibliothek enthält eine spezielle WFC Implementierung. Dieser WFC ist generisch (statt Kacheln können beliebige Daten verwendet werden) und modular aufgebaut, was Modifikationen erleichtert.

4.2 Design

Die Struktur der Anwendung ist an einer Model View Controller (MVC) Referenzarchitektur angelehnt (Abb. 4.2). Verwandte Architekturen wie Model View ViewModel oder Model View Presenter sind bei Android Anwendungen beliebt. Model ist eine zustandsbehaftete Komponente und enthält die Implementierung für die Gleisgenerierung, der Bewegung vom Zug und die Anbindung an Bibliotheken wie ARCore oder der AR Bibliothek. Renderer agiert als View und übernimmt das Rendern von Umgebung und virtuellen Objekten und verwaltet Meshes der Gleisteile und des Zuges. MainActivity bietet den Einsprungspunkt der Anwendung und übernimmt die Verknüpfung von User Input und Callbacks.

Die Interaktion zwischen den Komponenten wird durch die Schnittstellen beschrieben. Renderer bezieht kontinuierlich Daten über das Gleis und den Zug aus Model. Diese Schnittstelle wird durch IModelView beschrieben. MainActivity kann über die Schnittstelle IModelController, die Gleisgenerierung in Model starten, Einstellungen in Model ändern. Über die Schnittstelle IView kann MainActivity Testansichten in Renderer freischalten.

Interface	Beschreibung
IModelView	Enthält Funktionen für das Abfragen von AR bezogenen Daten: getPointCloud, getD
IModelController	F7
IView	showMessage, showPlanes, showPointCloud, ..

Tabelle 4.2: Interfaces innerhalb der Anwendung

Nach dem Initialisieren der Anwendung erstellt Renderer einen Thread für das Rendering, auf dem Update und Draw in einer Endlosschleife laufen. Die Schleife erfüllt die Aufgaben, die auf Abb. 2.1 im Grundlagenkapitel vorgestellt wurden.

Im **Update** Schritt manipuliert Renderer den Zustand von Model. Es wird ein neues Kamerabild erworben, das Tracking und die Umgebungsanalyse wird aktualisiert. Falls

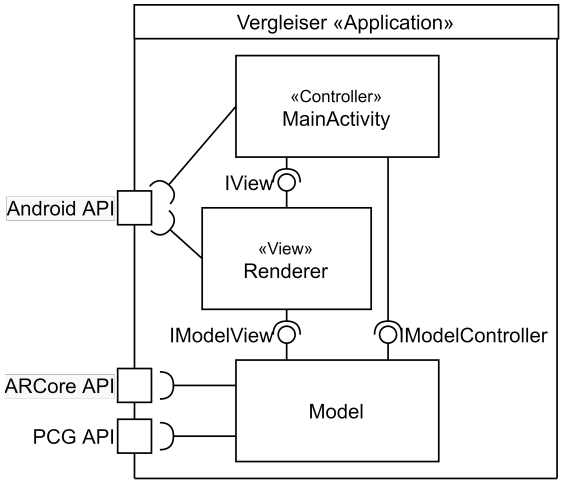


Abbildung 4.2: Die Anwendung ist in Model-, View- und Controller-Komponenten aufgeteilt.

möglich wird der Zug entlang des Gleises bewegt. Im **Draw** Schritt (Abb. 4.3) werden das Tiefenbild und eine erkannte Ebene abgefragt. Dann wird das aktuelle Kameraframe und ggf. Gleis und Zug gezeichnet.

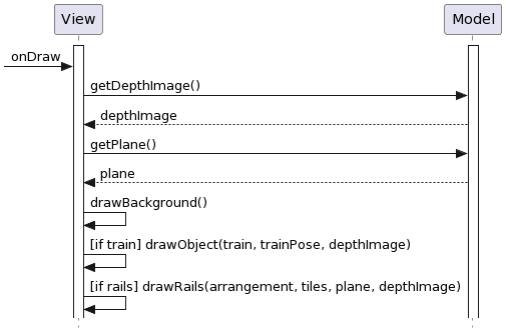


Abbildung 4.3: Während des Draw Schritts wird zunächst der Hintergrund gezeichnet, und dann die virtuellen Objekte.

Die Gleisgenerierung (Abb. 4.4) wird auf einem Thread gestartet. Die Ergebnisse der Umgebungsanalyse werden an den WFC übergeben, die Kachelanordnung und der längste Pfad werden generiert.

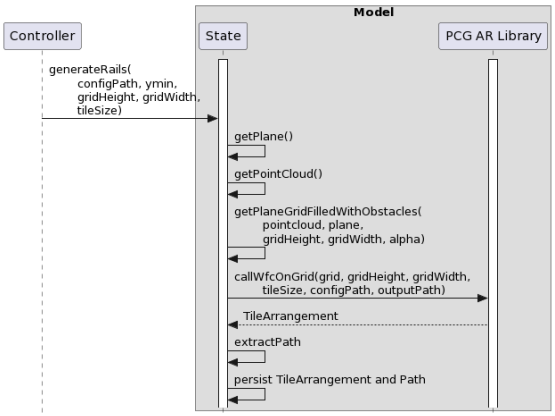


Abbildung 4.4: Die Gleisgenerierung wird von der GUI aus gestartet. Wenn die nötigen Daten gesammelt wurden, wird die Kachelanordnung von der WFC Implementierung erstellt.

4.3 Implementierung

Die Implementierung der Anwendung basiert auf einigen ARCore Beispielprojekten [8] und wurde in die vorgestellte Architektur umstrukturiert. Sie wurde in Kotlin und C++ mit Android Studio und Visual Studio entwickelt. Die verwendete WFC-Implementierung ist Teil einer PCG-Bibliothek für AR-Anwendungen einer Forschungsgruppe für Computergrafik an der Hochschule für Angewandte Wissenschaften (HAW) und wurde in C++ entwickelt.

ARCore muss mit einer Visualisierungsbibliothek genutzt werden, da sie nur Tracking und Registrierung übernimmt. Auf die Beispielprojekte aufbauend, wurde hier OpenGL ES 3.0 [13] verwendet, welche nativ auf der Android Platform verfügbar ist. Alternative Bibliotheken bauen auf OpenGL [18] auf oder nutzen andere Rendering APIs wie Vulcan [14]. Einige beliebte Frameworks wie Sceneform [7], seine Varianten und Scenekit [2] bündeln Bibliotheken für AR und Visualisierung.

Die Anwendung enthält einen Knopf, um die Gleisgenerierung zu starten, ein Knopf für Einstellungen und ein seitliches Menü für Testzwecke und einen Schieberegler für den Höhenabstand-Parameter y_{min}. ARCore erfordert ein Gerät mit Android-Version 7.0.

Es wurde Instanced Rendering genutzt, um die Gleiszeichnung zu optimieren. Die Anzahl der Zeichen-Aufrufe hängt nicht von der Anzahl der Zellen ab, sondern von der Anzahl der Kacheltypen.

Die genutzte WFC-Implementierung erwartet eine Beschreibung des verwendeten Tilesets in einer Konfigurationsdatei. Die Pfaddefinition wird auch in einer Konfigurationsdatei definiert. In den Anwendungseinstellungen kann ein Dateipfad für den Konfigurationsordner hinterlegt werden, der diese Konfigurationsdateien enthält.

5 Evaluation

Die Anwendung hat die Anforderungen prinzipiell erfüllt und kann die Grundflächen von entsprechend großen Objekten erkennen und ein Gleis generieren. Nur Q1 und Q3 wurden nicht erfüllt, die Frame-Rate der Anwendung variiert stark und kann unter 10 FPS fallen, außerdem dauert die Gleisgenerierung in manchen Fällen länger als zwei Minuten.

Die Anwendung wird in den folgenden Abschnitten unter den Aspekten Optik, Korrektheit und Performance untersucht.

5.1 Optik

Das entwickelte PCG-Verfahren erzeugt Gleise, die labyrinthartig erscheinen. Bei der Visualisierung kann die Verdeckung manchmal inkorrekt berechnet werden, so können Gleisteile reale Objekte verdecken, hinter welchen sie positioniert sein sollten. Der Zugpfad wird aus benutzerdefinierten Teilpfaden generiert. Diese Pfaddefinition wird für große Tilesets mühsam.

Die Gleisoptik erscheint blockartig, da das benutzte Tileset nur aus zwei sichtbaren Kacheln besteht. Das führt dazu, dass viele aufeinanderfolgende Ecken generiert werden. Ein anderes Tileset, das ebenfalls geschlossene Pfade erzeugt, könnte Gleise mit weniger Ecken generieren. Es könnte ein Verfahren angewendet werden, um einen Zugpfad aus dem erstellten Gleis zu extrahieren. Bei Verdeckungsfehlerm ist es schwierig die Ursache nachzuvollziehen. Denn sowohl Tracking- als auch Registrierungsfehler sind schwierig zu überprüfen. Tracking basiert auf der Detektion von Featuren. Bei einer fehlerhaften Einschätzung der Position ist das erzeugte Koordinatensystem schlecht zur Realität angeglichen. Subsequente Anfragen wie Tiefendaten oder Kamerapose erscheinen nicht korrekt. Aus Trackingfehlern folgen Registrierungsfehler, sodass die Anwendung schlecht einschätzen kann, ob virtuell nebeneinanderliegende Punkte auch in der Realität nebeneinander liegen. So kann es dazu kommen, das virtuelle Objekte falsch positioniert werden.

5.2 Performance

Die Performance wird unter den Aspekten Trackingverlust, Speicherverbrauch und Temperatur, WFC-Laufzeit und Frame-Laufzeit behandelt.

Bei Trackingverlust wird der Benutzer darüber benachrichtigt, wie die Szene zu verbessern ist, sodass Tracking-Neustart erfolgen kann. Befindet sich der Benutzer in einem dunklen Raum mit unzureichendem Kontrast, wird der Benutzer benachrichtigt in einen beleuchteten Raum zu wechseln. Wenn der Benutzer das Mobilgerät zu schnell bewegt, wird der Benutzer benachrichtigt sich langsamer zu bewegen. Eintönige weiße Wände führen immer zu Trackingverlust der Anwendung. Wenn das Umgangen werden soll, müssen Tiefensensoren verwendet werden, z.B. Ansätze mit strukturiertem Licht oder Lidar. Das Problem taucht verstärkter auf Spiegeln und spiegelnden Oberflächen auf. Wahrscheinlich werden Feature im Spiegelbild erkannt, die ihre relative Position bei jeder Bewegung stark verändern; wie bei hohem Glanz. Die Anwendung kann bei der Punktwolkenakkumulation so viel RAM verbrauchen, dass es zum Absturz kommt. Mit zunehmender Größe der Punktwolke steigt die Gerätetemperatur an. Die Laufzeit der Gleisgenerierung variiert je nach Auflösung und Dimension der erkannten Ebene zwischen 1 Sekunde und 2 Minuten. Ab 100×100 Zellen braucht der WFC ca. zwei Minuten. Wenn das verwendete Gitter nicht quadratisch ist, ist die Laufzeit bei gleicher Zellenanzahl kleiner. Bei einem Tisch mit den Maßen ein Meter mal zwei Meter und einer gewünschten Zellendimension von 3 cm braucht die Anwendung ca. acht Sekunden. Bei Auflösungen bis etwa 30 Zeilen und Spalten, beträgt die WFC-Laufzeit weniger als eine Sekunde. Auf dem Flur braucht der WFC, aufgrund der Größe, länger als eine Minute. Beim Start der Anwendung verlangsamen sich die ersten Frames auf über 100 ms wegen dem Laden von 3D-Modellen und dem Starten des Trackings. Nach wenigen Frames der Initialisierung, scheinen Tiefenbilder für jedes subsequente Frame bereit zu sein. Beim regulären Gebrauch der Anwendung stockt sie sehr. Im Grundlagenkapitel wurde erwähnt, dass Einfarbigkeit, Abwesenheit von Textur, hoher Glanz und repetitive Muster das optische Tracking stark beeinflussen. Drinnen kann die Szene und ihre Beleuchtung kontrolliert werden, daher ist diese Anwendung für drinnen gedacht. Trotzdem kann das Tracking durch spiegelnde Flächen gestört werden. Die hohen Leistungsanforderungen an AR zeigen sich als Wärmeanstieg und Stocken der Anwendung.

Die Leistungsfähigkeit des WFC sind ein Bottleneck der Anwendung. Will man die Anwendung ohne Wartezeit verwenden, muss eine optimierte WFC-Implementierung genutzt werden. Um dem Bottleneck entgegenzuwirken, könnte man das Echtzeit-Verfahren

wie DepthLab nutzen, da es keine Punktwolken akkumuliert. Dieser Ansatz ist für diese Anwendung nicht geeignet, da dort nur ein Tiefenbild verwendet wird. Man kann nicht erfahren, was hinter der Oberfläche eines realen Objekts liegt. Damit kann man keine Grundflächen erkennen. Prinzipiell kann auch diese Anwendung ohne Punktwolkenakkumulation genutzt werden, indem man die Gleisgenerierung bei einer Draufsicht startet, sodass die Grundflächen sichtbar sind.

5.3 Korrektheit

Die Grundflächenerkennung approximiert die Grundflächen nicht gut, regelmäßig werden reale Grundflächen überschätzt. Besonders bei hohen Objekten ist der Höhenabstand-Parameter y_{min} effektiv, weil sie herausragen, jedoch hat die Anwendung Probleme mit flachen Objekten. Außerdem rastert der Rasteralgorithmus die Konturen fehlerhaft. Mal fallen die Konturen innerhalb, mal außerhalb der Grundfläche. Ein anderer Rasteralgorithmus sollte genutzt werden, um die Konturen korrekt zu rastern.

Ein optimaler Höhenabstand y_{min} wurde nicht gefunden und muss vom Benutzer von Fall zu Fall bestimmt werden. Wenn der Wert zu klein ist, wird auch die Ebene als Grundfläche erkannt. Ist der Wert zu groß, werden keine Grundflächen erkannt. Die Tiefendaten zu bewerten ist ohne eine Referenz, wie ein Raumscan, nicht trivial. Sie könnten schlecht angeglichen sein, sodass die Punktwolke verzerrt ist oder starkes Rauschen enthalten, das als Hindernis erkannt wird. Ein leichtes Überschätzen der Grundflächen sollte sich nicht negativ auf die Gleisgenerierung auswirken.

6 Schluss

In dieser Arbeit wurden Tiefendaten zu einer Punktwolke akkumuliert, um die Anwesenheit von Objekten auf einer Ebene zu erkennen und für den WFC nutzbar zu machen. Dafür wurde eine einfache Anpassung an dem WFC übernommen. Im Abschnitt Stand der Technik wurden die Themen AR, Tiefendaten, PCG, WFC und themenverwandte Arbeiten vorgestellt. Dabei wurden die Komponenten eines AR-Systems erklärt und es wurden Visualisierung und verschiedene Methoden im Bereich optisches Tracking vorgestellt. Das optische Tracking ermöglicht erst die situierte Visualisierung und Abfrage von Tiefendaten auf Geräten mit einer einzelnen Kamera. Beim Thema PCG wurde ein breiter Überblick über das Thema gegeben, weil es sich lohnt eine abstrakte Sicht auf das Gebiet zu werfen. Vor allem, weil PCG mehr eingesetzt, als erforscht wird. Der Ablauf und die Weiterentwicklungsmöglichkeiten vom WFC wurden vorgestellt. Bei den themenverwandten Arbeiten wurde eine Anzahl an Arbeiten ausgewählt, die auch in AR oder VR mit Tiefendaten arbeiten, um Content zu platzieren, zu generieren oder Tiefendaten nutzen, um Spielfiguren automatisch zu navigieren. Im Abschnitt Konzeption wurden die Anforderungen und die verwendeten Lösungsansätze zur Grundflächenerkennung und Gleisgenerierung genannt und erläutert. Im Abschnitt Software wurde das Verhalten der Komponenten und ihre Interaktion miteinander erklärt und die Implementierung beleuchtet.

Content Platzierung ist in AR besonders aufwändig, da dieser nicht, ohne selbes Setup der Szene, geplant werden kann. Jeder Raum ist anders, daher ist die Nutzung von PCG-Verfahren besonders sinnvoll, welche Aspekte der Szene nutzen können, um reaktiv virtuelle Objekte zu platzieren. Da die Anwendung keiner Marker nutzt, kann der breiteren Masse die Benutzung zugänglicher gemacht werden. Dafür wurde auf Robustheit verzichtet, welche Modelbased Trackingverfahren anbieten. Generell funktioniert die Anwendung, aber nicht alle nichtfunktionalen Anforderungen wurden erfüllt. Die Grundflächenerkennung ist schnell und hohe Objekte können gut erkannt werden. Generell gibt es ein Problem mit der Leistungsfähigkeit der Algorithmen und Genauigkeit der Tiefendaten. Sie werden jedoch schnell erzeugt.

Die Anwendung könnte unter folgenden Aspekten weiterentwicklet werden. Zusätzlich zu den Tiefendaten könnte man die RGB Bilder verarbeiten. Die Umgebungsanalyse könnte durch Verfahren der Bildanalyse bereichert und robuster gestaltet werden. RGB Bilder bieten mehr Informationen für PCG-Verfahren. Adaptiver Content könnte generiert werden, der auf Farben oder Feature reagiert. Die Gleisgenerierung beschränkt sich auf 2D. Der Anwendungsfall könnte in 3D überführt werden. Dafür müsste eine Oberflächenerkennung oder Objekterkennung statt der Grundflächenerkennung genutzt werden. Content in vollständigem 3D erlaubt ein umfangreicheres AR-Erlebnis. Es animiert den Benutzer dazu, sich mehr um den Content zu bewegen, um ihn von allen Seiten zu betrachten. Es könnte eine Tiefendaten-Referenz, wie z.B. ein Raumscan, in die Umgebungsanalyse integriert werden. Diese Referenz könnte als Teil eines extensiven Umgebungsscans durch den Benutzer erhoben werden. Die Anwendung würde mehr Robustheit und Geschwindigkeit im Austausch von Spontanität erlangen. Mit einem anderen Tileset, kann man die Gleisoptik ändern. Mit einem automatischen Verfahren für die Extraktion des Zugpfads, würde die Nutzung von weiteren Tilesets vereinfacht werden. Das Overlap Model vom WFC würde ein alternatives PCG-Verfahren inspirieren. Die Ausgabestruktur des Overlap Model ist auf Pixel Basis, sodass ein weiterer Algorithmus nötig ist, der ein Gleis mit einer gewissen Breite generiert. Die generierten Gleise wären optisch interessanter.

Literaturverzeichnis

[1] Alamia, Marco: *Article - World, View and Projection Transformation Matrices*. http://www.codinglabs.net/article_world_view_projection_matrix.aspx. (letzter Abruf 10.05.2023)

[2] Apple: *SceneKit*. https://developer.apple.com/documentation/scenekit/. (letzter Abruf 06.06.2023)

[3] Azad, Sasha ; Saldanha, Carl ; Gan, Cheng-Hann ; Riedl, Mark: Mixed Reality Meets Procedural Content Generation in Video Games. In: *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* 12 (2016), Nr. 2, S. 22–26. – ISSN 2334-0924

[4] Azuma, Ronald T.: A Survey of Augmented Reality. In: *Presence: Teleoperators and Virtual Environments* 6 (1997), August, Nr. 4, S. 355–385. – ISSN 1054-7460

[5] Du, Ruofei ; Turner, Eric ; Dzitsiuk, Maksym ; Prasso, Luca ; Duarte, Ivo ; Dourgarian, Jason ; Afonso, Joao ; Pascoal, Jose ; Gladstone, Josh ; Cruces, Nuno ; Izadi, Shahram ; Kowdle, Adarsh ; Tsotsos, Konstantine ; Kim, David: DepthLab: Real-time 3D Interaction with Depth Maps for Mobile Augmented Reality. In: *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*. New York, NY, USA : Association for Computing Machinery, Oktober 2020 (UIST '20), S. 829–843. – ISBN 978-1-4503-7514-6

[6] Efros, A.A. ; Leung, T.K.: Texture Synthesis by Non-Parametric Sampling. In: *Proceedings of the Seventh IEEE International Conference on Computer Vision* Bd. 2, September 1999, S. 1033–1038 vol.2

[7] Google: *Sceneform*. https://developers.google.com/sceneform/develop. (letzter Abruf 06.06.2023)

[8] Google: *Codelabs - Google Developer Tutorial*. https://codelabs.developers.google.com/?text=arcore. (letzter Abruf 19.07.2023)

[9] Google: *Depth adds realism.* https://developers.google.com/ar/develop/depth. (letzter Abruf 25.11.2022)

[10] Gumin, Max: *WFC GitHub Repository.* https://github.com/mxgmn/WaveFunctionCollapse. (letzter Abruf 03.12.2022)

[11] Helbig, Jascha: *Entwicklung einer Augmented Reality Modelleisenbahn*, TECHNISCHE UNIVERSITÄT HAMBURG-HARBURG, Bachelor's thesis, April 2018

[12] Hendrikx, Mark ; Meijer, Sebastiaan ; Van Der Velden, Joeri ; Iosup, Alexandru: Procedural Content Generation for Games: A Survey. In: *ACM Transactions on Multimedia Computing, Communications, and Applications* 9 (2013), Februar, Nr. 1, S. 1:1–1:22. – ISSN 1551-6857

[13] Inc, The Khronos G.: *OpenGL ES Overview.* https://www.khronos.org/opengles/. (letzter Abruf 05.04.2023)

[14] Inc, The Khronos G.: *Vulkan Platform Support.* https://www.vulkan.org/. (letzter Abruf 06.06.2023)

[15] Lawrence, Louise M. ; Hart, Jonathon D. ; Billinghurst, Mark: Holo Worlds Infinite: Procedural Spatial Aware AR Content. In: *ICAT-EGVE 2017 - Posters and Demos* (2017), S. 2 pages. ISBN 9783038680529

[16] Merrell, Paul ; Manocha, Dinesh: Model Synthesis: A General Procedural Modeling Algorithm. In: *IEEE Transactions on Visualization and Computer Graphics* 17 (2011), Juni, Nr. 6, S. 715–728. – ISSN 1941-0506

[17] Microsoft: *Kinect for Windows.* https://learn.microsoft.com/en-us/windows/apps/design/devices/kinect-for-windows. (letzter Abruf 01.12.2022)

[18] Microsoft: *OpenGL Headline News.* https://www.opengl.org/. (letzter Abruf 05.04.2023)

[19] Microsoft: *Microsoft HoloLens 2.* https://www.microsoft.com/en-us/hololens. (letzter Abruf 16.02.2023)

[20] Mohr, Roger ; Henderson, Thomas C.: Arc and Path Consistency Revisited. In: *Artificial Intelligence* 28 (1986), März, Nr. 2, S. 225–233. – ISSN 0004-3702

[21] Nister, D. ; Naroditsky, O. ; Bergen, J.: Visual Odometry. In: *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.* Bd. 1, Juni 2004, S. I–I. – ISSN 1063-6919

[22] Rett, Florian: *Using Mobile AR Tracking Data to Generate 3D Geometry for Gameplay Interaction*, TECHNISCHE UNIVERSITÄT MÜNCHEN, Bachelor's Thesis, Juni 2019

[23] Schmalstieg, Dieter ; Höllerer, Tobias: *Augmented Reality - Principles and Practice.* Addison-Wesley Professional, Juni 2016. – ISBN 978-0-321-88357-5

[24] Scurti, Hugo ; Verbrugge, Clark: Generating Paths with WFC. (2018)

[25] Shaker, Noor ; Togelius, Julian ; Nelson, Mark J.: *Procedural Content Generation in Games.* Cham : Springer International Publishing, 2016 (Computational Synthesis and Creative Systems). – ISBN 978-3-319-42714-0 978-3-319-42716-4

[26] Sra, Misha ; Garrido-Jurado, Sergio ; Schmandt, Chris ; Maes, Pattie: Procedurally Generated Virtual Reality from 3D Reconstructed Physical Space. In: *Proceedings of the 22nd ACM Conference on Virtual Reality Software and Technology.* New York, NY, USA : Association for Computing Machinery, November 2016 (VRST '16), S. 191–200. – ISBN 978-1-4503-4491-3

[27] Togelius, Julian ; Kastbjerg, Emil ; Schedl, David ; Yannakakis, Georgios N.: What Is Procedural Content Generation? Mario on the Borderline. In: *Proceedings of the 2nd International Workshop on Procedural Content Generation in Games.* New York, NY, USA : Association for Computing Machinery, Juni 2011 (PCGames '11), S. 1–6. – ISBN 978-1-4503-0872-4

Glossar

Augmented Reality Augmented Reality, auf Deutsch Erweiterte Realität bezeichnet eine Technologie, um die Wahrnehmung des Benutzers zu analysieren und zu manipulieren. Die Analyse erfolgt über verschiedenste Sensoren und die Manipulation (Erweiterung) kann über Bilder, Töne, Gerüche oder Empfindungen wie Vibrationen und Temperatur erfolgen..

Content Digitale Inhalte, z.B. Bilder, ein 3D Modell, Video, Musik..

Feature Ein Feature ist ein Bildbereich mit identifizierbaren Eigenschaften. Methoden um Feature zu beschreiben sind SIFT, SURF, Harris Corners. Feature sind auch als Interest Points bekannt..

Head-Mounted Display Ein Head-Mounted Display ist ein an den Kopf gebundenes System mit einem Bildschirm. Der Umfang des Systems kann stark variieren..

Material Ein Material beinhaltet mehrere Texturen, die verschiedene Werte auf Meshes abbilden. Z.B. Farbe, Rauheit..

Mesh Ein dreidimensionales polygonales Netz zur Beschreibung von Oberflächen. Dreiecksnetz, Viereckснetz. Es ist häufig Teil eines 3D Modells..

Platformer In einem Platformer (Plattformspiel) bewegen sich Spielfiguren von Plattform zu Plattform und umgehen dabei Hindernisse[1]. Prominente Beispiele sind Super Mario Bros. und Donkey Kong..

[1] https://en.wikipedia.org/wiki/Platform_game (Abruf 16.02.2023).

Procedural Content Generation Procedural Content Generation ist auch bekannt als Procedural Generation mit der Abkürzung procgen, auf Deutsch Prozedurale Content-Generierung bzw. Prozedurale Generierung. Das ist eine Vielzahl an Methoden um Inhalte vorallem in Spielen zu generieren..

Video See-Through Video See-Through bezeichnet einen Ansatz für visuelle Augmented Reality Systeme. Der Ansatz nutzt mind. einen Bildschirm, eine Kamera und einen Prozessor. Der Benutzer kann oft nur den Bildschirm sehen (im Gegensatz zum Ansatz Optical See-Through, bei dem *durch* den Bildschirm gesehen wird). Um die Umgebung zu erweitern wird eine Aufnahme der Umgebung (das Video) mit digitalen Bildern erweitert und auf dem Bildschirm angezeigt..

Virtual Reality Virtual Reality, auf Deutsch Virtuelle Realität..

Erklärung zur selbstständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original