Jade Hochschule Oldenburg

Fachbereich: Bauwesen Geoinformation Gesundheitstechnologie

Studiengang: B.Sc. Angewandte Geodäsie

Bachelorarbeit

Möglichkeiten der Herstellung synthetischer Trainingsdaten für KI-Modelle in einer Game-Engine aus Geodaten

Eingereicht von: **Jannik Fröchtenicht (Mat.-Nr. 6033689)**
Elveser Weg 11
37154 Northeim
Telefon: 05551-2915
E-Mail: jannik.froechtenicht@student.jade-hs.de

Erarbeitet im: 8. Semester

Abgegeben am: 16.02.2024

Prüfer: Prof. Dr. Roland Pesch
Jade Hochschule Oldenburg
Ofener Str. 16/19
26121 Oldenburg
Telefon: 0441-7708 3248

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

ALKIS – Amtliches Liegenschaftskatasterinformationssystem

API – Programmierschnittstelle

Bzw. - beziehungsweise

DGM – Digitale Geländemodell

DHHN2016 – Deutsches Haupthöhennetz 2016

DLM – Digitales Landschaftsmodell

DOM – Digitale Oberflächenmodell

DOP – Digitales Orthofoto

ETRS89 – Europäisches Terrestrisches Referenzsystem 1989

GIS – Geoinformationssystem

KI – Künstliche Intelligenz

KNN – Künstliche neuronale Netzwerke

LGLN – Landesamt für Geoinformation und Landesvermessung Niedersachsen

LIDAR – Light Detection and Ranging

LoD – Level of Detail

RGB – Rot Grün Blau

UAV - Unmanned Aerial Vehicle

UTM – Universale Transversale Mercatorprojektion

z.B. – zum Beispiel

Zusammenfassung

Für das Training von KI-Modellen werden große und vielfältige Mengen an klassifizierten Trainingsdaten benötigt. Traditionelle Herstellungsmethoden für diese sind zeit- und arbeitsintensiv. Zusätzlich ist es nicht immer möglich, für alle Grenzfälle gutes und ausreichendes Datenmaterial vorliegen zu haben. In dieser Arbeit wird eine Methode beschrieben werden, um synthetische Trainingsdaten für ein KI-Modell zu erzeugen. Es soll eine Möglichkeit der Erzeugung von Trainingsdatensätzen für die Semantische Segmentierung von Luftbildern mit Esri Produkten und der Unity Game-Engine untersucht werden.

Dazu werden bereits vorhandene Datensätze öffentlicher Stellen genutzt werden, umso den Erfassungsaufwand zu minimieren. Mit diesen Daten wird eine 3D-Szene in der Unity Game-Engine erstellt und aus dieser können dann die Trainingsdaten abgeleitet werden. Demonstriert wird diese Vorgehensweise für die Erkennung von Dachflächen in Luftbildern mit LoD2-Gebäudemodellen. Trotz verbleibender Fragestellungen zeigt sich, dass die beschriebene Methode zur Herstellung synthetischer Trainingsdaten funktioniert und weiteres Potenzial für ähnliche Anwendungsfälle bietet.

Abstract

There is a need for big and varied amounts of annotated data for the training of AI-models. Traditional production methods for these are time-consuming and labor-intensive. Additionally, it isn’t always practical to have enough data material for every edge case on hand. This paper describes a method for generating synthetic training data for an AI model. The aim is to demonstrate a way of generating training data sets for the semantic segmentation of aerial images using Esri products and the Unity game engine.

Existing data sets from public institutions are to be used for this purpose to minimize the effort involved in data collection. This data is used to create a 3D scene in the Unity game engine, from which the training data can then be derived. This procedure is demonstrated for the detection of roof surfaces in aerial images with LoD2 building models. Despite remaining questions, it is clear that the method described for producing synthetic training data works and offers further potential for similar applications.

1. Einleitung

KI ist ein Schlagwort, das derzeit in aller Munde und in der Presse zu lesen ist. Es handelt sich um ein Teilgebiet der Informatik und beschreibt Anstrengungen, deren Ziel es ist, Maschinen intelligent zu machen. Im weitesten Sinne wird Intelligenz in diesem Kontext verstanden als die Fähigkeit eines Wesens, angemessen und vorausschauend in seiner Umgebung zu agieren. Beispielsweise geht es darum, Umgebungsdaten wahrzunehmen und darauf zu reagieren, Informationen aufzunehmen, zu verarbeiten und als Wissen abzuspeichern, Sprache zu verstehen und zu erzeugen, Probleme zu lösen und Ziele zu erreichen.

Vor dem Hintergrund dieser großen Bandbreite an Problemstellungen und vielfältigen Lösungsansätzen, unterteilt sich die KI-Forschung in mehrere, relativ eigenständige Teilgebiete: Mustererkennung (z.B. Spracherkennung), Wissensmodellierung, Expertensysteme, Maschinelles Lernen, Computer Vision, Robotik und Universelle Spieleprogramme.

Diese Arbeit ist dem Teilgebiet des maschinellen Lernens zuzuordnen. Bei diesem Aspekt der KI-Forschung geht es um die Generierung von Wissen aus Erfahrung. Das heißt ein künstliches System lernt aus Beispielen und kann diese nach Beendigung der Lernphase verallgemeinern. Es sollen nicht nur Beispiele auswendig gelernt werden, sondern es sollen Muster und Gesetzmäßigkeiten in den Lerndaten erkannt werden, die dann auf einen neuen Datensatz angewandt werden können.

Die Verfügbarkeit großer und vielfältiger Datensätze ist entscheidend für die Leistungsfähigkeit von KI-Modellen. Allerdings stoßen viele Entwickler auf Herausforderungen, wenn es darum geht, ausreichende und repräsentative reale Daten zu sammeln. Dieses Dilemma wird besonders in Anwendungsgebieten deutlich, in denen die Datenerfassung teuer, zeitaufwendig oder ethisch problematisch ist.

Die Vorteile KI basierter Methoden sind es schnell und effizient, Informationen aus großen Datenmengen gewinnen zu können. Diese Vorteile werden auch zunehmend für die Verarbeitung von Geodaten genutzt (KUMARI ET. AL. 2023). Geodaten und insbesondere Fernerkundungsdaten sind nicht nur sehr umfangreich und müssen häufig aktualisiert werden, sondern stellen auch besondere Herausforderungen für das Training von KNN dar: Sie sind geografisch korreliert, was bedeutet, dass ein Datensatz von einem bestimmten Standort möglicherweise nicht vollständig für einen anderen Standort repräsentativ ist. Zusätzlich kommen neue Anwendungsfälle auf und für diese müssen neue Klassifikationen erstellt werden. Ein Beispiel aus dem LGLN wären Bestrebungen, die Materialabdeckung der Erdoberfläche im Zusammenhang mit der Berechnung des Entsiegelungspotenzials aus Luftbildern zu erfassen.

Die Motivation für die Erforschung synthetischer Trainingsdatensätze liegt in der Überwindung dieser Hindernisse und der Schaffung eines umfassenderen, vielseitigeren Datensatzes durch die Ergänzung realer Trainingsdaten durch synthetische Daten. Diese werden künstlich erzeugt, um die Lücken in realen Datensätzen zu schließen, die Modellleistung zu verbessern und den Anwendungsbereich von KNN zu erweitern. Diese Herangehensweise ermöglicht eine effizientere Nutzung von KNN in Szenarien, in denen der Zugang zu ausreichend realen Daten begrenzt ist.

Mit der in dieser Arbeit beschriebenen Methode sollen synthetische Trainingsdatensätze für das Trainieren eines KNN für die automatische Auswertung von Luftbildern generiert werden. Dabei soll auf bereits vorhandene Datensätze der Katasterverwaltung Niedersachsen zurückgegriffen werden, um den Aufwand der Datenerfassung zu minimieren. Dazu soll das Potenzial der Game-Engine Technologie in Verbindung mit GIS aufgezeigt werden. In dieser Arbeit wird eine Herstellungsmethode präsentiert, die die Unity Game-Engine und ArcGIS Pro umfasst, um realistische Luftbilder und die dazugehörigen Klassifizierungsmasken für Deep-Learning-Segmentierungsmodelle zu erzeugen.

1.1 Struktur

Die Bachelorarbeit wird in mehreren Abschnitten strukturiert sein, um eine systematische Untersuchung des Themas zu gewährleisten. Der zweite Abschnitt wird sich mit den Grundlagen von KNN, die Anforderungen an Trainingsdatensätze und den für die Erzeugung der 3D-Szene erforderlichen Programmen beschäftigen.

Im dritten Abschnitt wird ausführlich auf die Datengrundlage der Arbeit eingegangen. Es geht um die Datenherkunft, die Genauigkeiten und die Raumbezüge.

Im vierten Abschnitt wird beschrieben, wie die 3D-Szene erzeugt wird und wie aus dieser die synthetischen Trainingsdaten erzeugt werden können.

Im fünften Abschnitt werden die Ergebnisse der Arbeit präsentiert.

Anschließend folgt im sechsten Abschnitt eine Diskussion der Ergebnisse. Es werden Vorteile der in der Arbeit präsentierten Methode genannt und es wird auf Nachteile und Herausforderungen eingegangen

Abschließend folgt ein Fazit im siebten Abschnitt.

2. Theoretischer Hintergrund

In diesem Abschnitt werden die Grundlagen von KNN dargestellt und Anforderungen an Trainingsdaten werden beleuchtet. Zusätzlich werden die zentralen Programme, die zur Erzeugung dieser Arbeit verwendet wurden, erläutert.

2.1 Erläuterungen zu KNN

KNN haben sich zu einer Schlüsseltechnologie in verschiedenen Bereichen der Informatik entwickelt. Sie werden durch Trainingsdaten trainiert und verwendet, um beispielsweise Beziehungen und Muster in Datensätzen zu erkennen (Data Mining) oder zur Mustererkennung in Bild und Ton. Inspiriert von biologischen neuronalen Netzwerken simulieren KNN die Art und Weise, wie das menschliche Gehirn Informationen verarbeitet. In diesem Abschnitt soll auf die Grundlegende Funktionsweise von KNN eingegangen werden und es sollen Anwendungen von KNN in der Praxis beleuchtet werden.

KNN bestehen aus miteinander verbundenen Neuronen auch als künstliche Neuronen oder Knoten bezeichnet. Jedes Neuron nimmt Eingaben entgegen, verarbeitet sie und gibt eine Ausgabe weiter. Die Verbindungen zwischen den Neuronen haben Gewichtungen, die den Einfluss einer Eingabe auf die Ausgabe steuern. In einem KNN sind diese Neuronen in Schichten organisiert: Eingangsschicht, versteckte Schichten und Ausgangsschicht.

Der Informationsfluss in einem KNN beginnt mit den Eingabewerten, die durch die Eingangsschicht laufen. Diese Eingaben werden mit den Gewichtungen verrechnet und an die Neuronen der nächsten Schicht weitergegeben. Dieser Prozess setzt sich durch die versteckten Schichten fort, bis die Ausgabeschicht erreicht wird. Das Netzwerk gibt dann die endgültige Ausgabe aus. Der Lernprozess in KNN wird durch Anpassen der Gewichtungen zwischen den Neuronen erreicht. Es gibt verschiedene Trainingsverfahren, doch im Folgenden wird auf das Überwachte Lernen eingegangen.

2.1.1 Überwachtes Lernen

Beim Überwachten Lernen wird dem KNN ein Eingangsmuster gegeben und nach dem Durchlaufen von diesem wird die Ausgabe mit dem Wert verglichen, der eigentlich ausgegeben werden soll. Durch den Vergleich der Soll- und Ist-Ausgabe kann darauf geschlossen werden, welche Änderungen an der versteckten Schicht bzw. den Gewichtungen zwischen den Neuronen vorgenommen werden müssen. „In einem typischen Deep-Learning-System können hunderte Millionen dieser einstellbaren Gewichtungen und hunderte Millionen [Trainingsdatensätze] sein, mit welchen die Maschine trainiert wird“ (LeCun et al. 2015: 436).

Da es nicht immer möglich ist, zu jedem Eingabedatensatz einen passenden Ausgabedatensatz zu definieren oder zum Training zur Verfügung zu haben, kann es sein,

dass die Notwendigkeit besteht, den Erfolg eines KNN durch mehrere Kennzahlen bewerten zu müssen. Beim Bestärkendem Lernen erfolgt diese Bewertung durch die Verlust-Funktion. Auf der Grundlage dieser können dann die Gewichtungen der Neuronen in der versteckten Schicht angepasst werden, um das KNN zu optimieren.

Die Anwendungen von KNN erstrecken sich über verschiedene Bereiche. Im nächsten Abschnitt geht es um die Verwendungsmöglichkeiten von KNN für die Verarbeitung von Geodaten

2.1.2 Anwendungen von KNN für Geodaten

Die zunehmende Verfügbarkeit und Aktualität von Geodaten in Form von Fernerkundungsdaten wie z.B. Luft- und Satellitenbilder macht diese für eine Vielzahl von Anwendungen interessant: Kartografie, Stadtplanung, Krisenmanagement, Umwelt, Verkehr, nachhaltige Landwirtschaft und Tourismus, um nur eine Handvoll möglicher Anwendungsgebiete zu nennen. KNN ermöglichen es, Informationen automatisiert aus Bilddaten zu erfassen und können daher einen Informationsgewinn ermöglichen. Verschiedene Aufgaben können mit KNN gelöst werden, die im Folgenden erläutert werden.

2.1.3 Semantische Segmentierung

KNN können Luft- und Satellitenbilder in verschiedene semantische Klassen wie Gebäude, Straßen, Vegetation, Gewässer und Landbedeckungstypen segmentieren. Die semantische Segmentierung weist jedem Pixel im Bild eine Klasse zu, was eine detailliertere Analyse der Landschaft ermöglicht.

Im Bereich der Geodaten zeigen sich für semantische Segmentierungsverfahren zahlreiche Anwendungsmöglichkeiten.

Für die Klassifizierung von Landnutzung können Luft- und Satellitenbilder in Landnutzungskategorien segmentiert werden, z.B. in stätische Gebiete, landwirtschaftliche Flächen, Wälder und Gewässer. Diese Informationen sind für die Stadtplanung, die Umweltüberwachung und das Ressourcenmanagement von entscheidender Bedeutung (VOELSEN ET AL. 2021).

Infrastrukturkartierung kann durch die Segmentierung von Bildern in Merkmale wie Gebäude, Straßen, Brücken und Schienenwege ermöglicht werden. Dies ist hilfreich für die Planung und das Management von Infrastruktur (HENRY ET AL. 2021).

Bei der Vegetationsanalyse kann die Segmentierung von Luft- und Satellitenbildern in verschiedene Arten von Vegetation wie Wälder, Wiesen oder landwirtschaftliche Nutzflächen die Identifizierung erleichtern (KATTENBORN ET AL. 2021). Dies ist für die Forstwirtschaft, Agrarwirtschaft und den Umweltschutz von Interesse.

Semantische Segmentierung kann es bei der Gewässererkennung ermöglichen, Flüsse, Seen, Teiche und andere Wasserkörper zu identifizieren und zu überwachen (ZHANG ET AL. 2021). Dies ist relevant für das Wasserressourcenmanagement und weitere umweltbezogene Fragestellungen.

2.1.4 Änderungserkennung

KNN können Paare von Satellitenbildern analysieren, die zu verschiedenen Zeiten aufgenommen wurden, um Änderungen in der Landschaft zu erkennen. Dies ist entscheidend für die Überwachung von städtischem Wachstum (FYLERIS ET AL. 2022), Abholzung, Veränderungen der Landbedeckung und Naturkatastrophen. KNN können lernen, Bereiche signifikanter Veränderung zu identifizieren, indem sie pixelweise die Unterschiede zwischen Bildern vergleichen.

2.1.5 Objekterkennung

KNN können Objekte von Interesse innerhalb von Luft- und Satellitenbildern erkennen und lokalisieren. Dies umfasst die Identifizierung einzelner Gebäude, Fahrzeuge, Infrastruktur, Vegetation und anderer Merkmale (DING ET AL. 2021). Objekterkennungsalgorithmen geben in der Regel einen Begrenzungsrahmen um erkannte Objekte sowie deren entsprechende Klassifizierungen aus.

2.1.6 Trainingsdaten

Für das Training eines jeden KNN werden Trainingsdatensätze benötigt. Im Folgenden wird als Beispiel das Scenario einer Pixelklassifikation verwendet, welche durch ein KNN automatisch jeden Pixel in einem Luftbild als ein Haus oder Hintergrund klassifizieren soll. Eine Möglichkeit, dieses Netzwerk durch Überwachtes Lernen zu trainieren, wäre eine Sammlung von Bildpaaren (Eingabebild und Klassifikationsmaske) zu erstellen. Das Eingabebild ist das realitätsgetreue Bild, aufgenommen durch eine Kamera, in dem jedem Pixel ein RGB-Wert zugeordnet wird. Das zweite Bild hat für jeden Pixel anstelle eines RGB-Wertes eine Klassifizierung, eine sogenannte Klassifizierungsmaske. In diesem Beispiel wären die zwei Klassen Gebäude und Hintergrund denkbar. Die realitätsgetreuen Bilder sind aufzunehmen oder anderwärtig zu beschaffen. Arbeitsaufwändig ist die Erstellung der Klassifizierungsmaske, welche traditionellerweise durch die manuelle Markierung eines jeden Pixels als eine der vorgegebenen Klassen erfolgt. „Anwendungen, in denen die Trainingsdaten Beispiele der [Eingangsdaten] und der korrespondierenden [Klassifizierungsdaten] enthalten, sind bekannt als Überwachtes Lernen“ (BISHOP 2006: 3). Das Ziel der Arbeit ist es also, Trainingsdaten für die Trainingsmethode des Überwachten Lernens für KNN zu generieren.

2.1.7 Anforderungen an Trainingsdatensätze

Im Optimalfall besteht der Datensatz, der zum Trainieren des KNN verwendet wird, aus einer Vielzahl von Datenbeispiele, die dazu führen, dass das Modell so viele Muster wie möglich aus den Daten lernt bzw. erkennen kann. Das bedeutet, dass der Trainingsdatensatz eine große Variation oder Diversität und damit ein Abbild der realen Welt sein sollte. Die Leistung des Modells hängt auch von der Anzahl der verfügbaren Datenbeispiele ab. In der Regel werden Deep-Learning-Modelle in Anwendungen der Bilderkennung mit Millionen Datenbeispielen (Bildpaaren) trainiert, um eine höhere Modellgeneralisierung zu gewährleisten.

Da für das Trainieren eines solchen KNN eine große Menge von Trainingsdaten benötigt werden, gibt es den Ansatz, diese synthetisch zu erzeugen. In diesem Falle heißt dies, eine 3D-Szene in einem Editor zu erstellen, von welcher im Anschluss Screenshots gemacht werden können. Der Aufbau einer möglichst realitätsnahen, simulierten Szene in einem 3D-Editor ist nicht trivial, jedoch ergibt sich die Arbeitszeitersparnis durch die automatische Ableitung der Klassifizierungsmaske.

Mehr Trainingsdaten heißt im Allgemeinen auch eine bessere Performance des KNN. In der Vergangenheit wurde versucht durch immer größere Datensätze und eine immer höhere Anzahl an Gewichtungen die Genauigkeit und die Aufgaben, die ein KNN erfüllen kann zu erhöhen. An dieser Stelle soll darauf hingewiesen werden, dass die Größe des Trainingsdatensatzes nicht der alleinige Erfolgsgarant für ein KNN ist, welches durch Überwachtes Lernen trainiert wird. Die Methode der Anpassung der Gewichte hat ebenfalls einen großen Einfluss auf die Performance eines KNN. In einer Untersuchung zu dieser Thematik im Hinblick auf die Klassifikation von Landnutzungsdaten in durch Satelliten erfassten Bildern wird folgendes Fazit gezogen. „Wir empfehlen daher, dass eine bewährte Vorgehensweise für jedes Projekt, bei dem es um überwachte Klassifizierungen von Fernerkundungsdaten geht, darin besteht, mehrere Klassifizierungsmethoden für maschinelles Lernen zu untersuchen, da einige Klassifikatoren für maschinelles Lernen je nach Größe des Trainingssatzes möglicherweise eine bessere Genauigkeit bieten als andere“ (RAMEZAN ET AL 2021: 21).

In den meisten Anwendungsfällen sind die synthetisch erzeugten Trainingsdaten als Ergänzung für andere Datensätze gedacht, welche auf herkömmliche Weise erzeugt worden sind. Interessant sind synthetische Trainingsdaten speziell auch für Nischen Anwendungen, für welche nicht bereits große Sammlungen von Trainingsdaten vorhanden sind und für dessen Erzeugung es wirtschaftlich nicht sinnvoll wäre, dies auf die traditionelle Weise zu tun.

Ein Beispiel aus der Forschung ist eine Arbeit des Institutes für Neuroinformatik der Ruhr-Universität Bochum. In einem Forschungsprojekt sollte ein KNN zur Erkennung von belegten

oder freien Parklücken trainiert werden, welche durch eine Überwachungskamera gefilmt wurden. Die Trainingsdaten hierfür sollen auf Videoaufnahmen einer animierten 3D-Szene beruhen, welche in der Unreal-Engine 4 erstellt wurde. Die Szene war hierbei eine digitale Nachbildung eines realen Parkplatzes. Aus den Aufnahmen wurden Screenshots erstellt und diese wurden anschließend für das Training eines KNN verwendet. „Das System erreichte eine Klassifikationsgenauigkeit von 99,05%“ (HORN & HOUBEN 2018: 5). In diesem Fall wurde das KNN ausschließlich mit synthetischen Trainingsdaten trainiert und die Validierung der Ergebnisse erfolgte anschließend durch zwei reale Videosequenzen. Dies zeigt, dass die Verwendung von synthetisch erzeugten Trainingsdaten als Grundlage für das Training von KNN dienen kann.

2.1.8 Erstellung von Trainingsdatensätzen

Üblicherweise werden Trainingsdatensätze durch manuelle Klassifizierung erzeugt. Die manuelle Beschriftung durch menschliche Experten liefert qualitativ hochwertige Trainingsdatensätze, ist jedoch sehr teuer und arbeitsintensiv. Es gibt mehrere öffentliche Trainingsdatensätze für verschiedene Aufgaben wie semantische Segmentierung (GARIOUD ET AL. 2023) und Objekterkennung (DING ET AL. 2021).

2.1.9 Generierung synthetischer Datensätze

Game-Engines wie Unity und Unreal Engine werden zunehmend zur Erzeugung synthetischer Trainingsdaten für unterschiedliche Anwendungen eingesetzt. Diese Entwicklungsumgebungen ermöglichen die Erstellung realistischer 3D-Szenen, die reale Landschaften imitieren, und erlauben die Generierung verschiedener Trainingsdatensätze für Aufgaben wie semantische Segmentierung und Objekterkennung. Dieses Forschungsthema ist jedoch neu, insbesondere für Geodaten. LAUX ET AL. (2023) stellen eine Methode zur Generierung syntetischer Trainingsdatensätze unter Verwendung von Unreal Engine und AirSIM zur Erkennung von Personen in UAV-Luftbildern vor. AirSim ist eine Simulationsumgebung für autonome Fahrzeuge unter Verwendung der Game-Engine Unreal Engine. Sie testen die Qualität des generierten Trainingsdatensatzes, indem sie die Objekterkennungsmodelle YOLOv4 trainieren. Für das Training verwenden sie sowohl reale als auch synthetische Daten. Sie kommen zu dem Schluss, dass die erzeugten Bilder zwar von hoher Qualität sind, aber oft Menschen im Schatten oder durch Hindernisse verdeckt zeigen und daher für das anfängliche Training zu schwierig sind.

In dieser Arbeit soll eine Möglichkeit der Erzeugung von Trainingsdatensätzen für die Segmentierung von Luftbildern mit Esri Produkten und der Unity Game-Engine demonstriert werden.

2.2 ArcGIS Pro

ArcGIS Pro ist ein GIS, das von Esri entwickelt wurde und es Benutzenden ermöglicht, räumliche Daten zu erstellen, zu verarbeiten, zu analysieren, zu präsentieren und zu teilen. Es handelt sich um eine Weiterentwicklung von ArcGIS Desktop und bietet eine moderne, benutzerfreundliche Benutzeroberfläche sowie erweiterte Funktionen für Geodatenverarbeitung und -analyse.

Eine der Hauptfunktionen von ArcGIS Pro ist die Möglichkeit, verschiedene Arten von Geodaten zu verarbeiten, darunter Vektor-, Raster- und 3D-Daten. Anwendende können komplexe räumliche Analysen durchführen, um Muster zu erkennen, Beziehungen zu verstehen und fundierte Entscheidungen zu treffen. Die Software bietet eine Vielzahl von Werkzeugen und Funktionen für Geoverarbeitung, Geostatistik, Geokodierung, Routing, Geländeanalyse und mehr.

Als ein GIS unterstützt ArcGIS Pro auch die Erstellung ansprechender Karten und Visualisierungen. Benutzende können Kartenelemente anpassen, Layouts entwerfen und interaktive Karten für Präsentationen oder den Einsatz im Web erstellen. Die Integration mit ArcGIS Online ermöglicht es Benutzenden, ihre Karten und Analysen online zu teilen und mit anderen zu kollaborieren.

Darüber hinaus bietet ArcGIS Pro Funktionen für die Datenverwaltung und -organisation. Anwendende können Geodatenbanken erstellen, Datenquellen verwalten, Metadaten hinzufügen und Geodaten in verschiedenen Formaten importieren und exportieren.

ArcGIS Pro ist der Standard für EVAP-GIS-Systeme. Für diese Arbeit wird die Version 3.2.1 verwendet.

2.3 Unity Game-Engine

Die Unity Game-Engine ist für diese Arbeit von zentraler Bedeutung, da die 3D-Szene, aus welcher später die Bildpaare abgeleitet werden sollen, in dieser erstellt wird. Im folgenden Abschnitt soll die Entwicklungsumgebung daher näher betrachtet werden.

Ursprünglich entwickelt wurde Unity als eine Game-Engine. Hierbei handelt es sich um ein spezielles Framework zur Entwicklung von Computerspielen, das den Spielverlauf steuert und für die visuelle Darstellung des Spielablaufes verantwortlich ist. In den letzten Jahren hat sich Unity zunehmend zu einem hilfreichen Werkzeug für die Entwicklung interaktiver Anwendungen im Bereich der virtuellen und der erweiterten Realität entwickelt und die Plattform unterstützt eine breite Palette von Endgeräten und Betriebssystemen, darunter PCs, mobile Geräte, Webbrowser und VR-Headsets.

Die Unity Game-Engine hat eine komponentenbasierte Architektur. Diese erleichtert die Strukturierung und Organisation von Spieleobjekten, welche nicht als monolithische Entitäten betrachtet werden, sondern als Ansammlung wiederverwendbarer Einzelteile. Diese Einzelteile haben spezifische Funktionalitäten wie z.B. Grafiken, Animation, Physik oder Beleuchtung. Dabei bietet die komponentenbasierte Architektur den Vorteil, dass jede Komponente unabhängig voneinander entwickelt, bearbeitet und eventuell auch ausgetauscht werden kann, was die Flexibilität und Wartbarkeit des Codes verbessert.

Die Beleuchtung und Texturierung von Objekten erfolgt in der Unity Game-Engine durch sogenannte Shader. Diese werden beispielsweise einem 3D-Objekt zugeordnet und geben dann vor, wie eine Lichtquelle jenes beleuchtet. Mit Shadern können auch Animationen realisiert werden.

Interessant für die Ausgestaltung und die Arbeit mit Unity sind der Unity Asset Store und verschiedene Foren. Hierüber können Komponenten und auch ganze Bibliotheken mit Objekten, Texturen oder Skripten in ein bereits vorhandenes Projekt integriert werden. In den Foren und durch die Entwickler bereitgestellte Tutorials können sich ein Anwendende die Entwicklung mit Unity selbst aneignen.

Ein entscheidender Vorteil der Game-Engine ist auch, dass das Programm für nicht kommerzielle Zwecke kostenlos ist und trotzdem alle Funktionen für Anwendende zur Verfügung stehen. Sollte es später zu einem gewerblichen Gebrauch der Software kommen, kann eine Lizenz erworben werden.

Die Unity Game-Engine bildet die Grundlage der 3D-Darstellung dieser Arbeit und ist in Verbindung mit dem ArcGIS Maps SDK for Unity, welches über den Package-Manager eingefügt werden kann, ein zentraler Bestandteil. Für diese Arbeit wird die Version 2022.3.8f1 verwendet.

2.4 ARCGIS Maps SDK for Unity

Das ArcGIS Maps SDK for Unity ist ein neues Entwicklungsframework, dessen erste Release Version im Juni 2022 veröffentlicht wurde. Es ermöglicht eine direkte Integration von geografischen Informationen in Unity-basierte Spiele und Anwendungen. Anwendende können mit dieser Schnittstelle Geoinformationen nutzen, um immersive, realitätsbezogene Spiele und Szenen zu erstellen, die von realen geografischen Daten unterfüttert sind.

Für die Integration der vorliegenden Geodaten in die 3D-Szene in Unity ist dieses Modul unersetzlich und soll daher im Folgenden näher beschrieben werden.

Eine für Entwickelnde ohne Hintergrund in der Geodäsie interessante Funktion ist die einfache Integration von Karten und GIS-Daten in Unity-Projekte. Dies wird ermöglicht durch den

„ArcGIS Living Atlas of the World“. Hierbei handelt es sich um eine Vielzahl von Online-Diensten, die von Esri in einem zentralen Portal zur Verfügung gestellt werden. Voraussetzung ist ein ArcGIS Online Zugang. Diese Dienste können direkt über die graphische Benutzeroberfläche als Layer in das Projekt eingefügt werden.

Das ArcGIS Maps SDK for Unity hat drei verschiedene Methoden, um eine 3D-Szene zu konfigurieren. Die Methode über das grafische Interface bietet den Vorteil, dass Anwendende ohne Kenntnisse im Programmieren schnell und einfach eine Szene erstellen können. Neben den von Esri zur Verfügung gestellten Diensten können hier auch eigene Geodaten als Layer hinzugefügt werden. Es ist jedoch wichtig, dass die betreffenden Daten das richtige Datenformat haben und Rasterdaten müssen auch das richtige Bezugssystem aufweisen. 3D-Ebenen und 3D-Objekte können durch das SDK transformiert und anschließend dargestellt werden.

Anwendende, die sich in der Unity-Architektur auskennen, können die Gestaltung der 3D-Szene durch Komponenten vornehmen. Die dritte Möglichkeit für Anwendende ist die Konfiguration durch die API-Schnittstelle. Hier können durch Skripte noch tiefergreifende Funktionen angesteuert werden, die durch die ersten beiden Optionen nicht möglich sind.

Das SDK ermöglicht die Interaktion mit Karten in 2D und 3D und bietet eine breite Palette von Funktionen zur Anpassung und Steuerung von Kartenvisualisierungen. Entwickler können Benutzerinteraktionen implementieren, wie das Zoomen, Verschieben und Klicken auf Kartenobjekte, um ein immersives Erlebnis zu schaffen.

Für die Zwecke dieser Arbeit interessant ist außerdem, dass das ArcGIS Maps SDK for Unity es ermöglicht, 3D-Ebenen und 3D-Objekte zu manipulieren. Zur Verfügung stehen raumbasierte Filterung, Mesh-Modifikation und die Anwendung von Materialien nach Attributen. Bei der raumbasierten Filterung können Teile einer Ebene entfernt werden, nachdem der Datensatz bereits hinzugefügt worden ist. Bereits bebaute Flächen können so beispielsweise digital geräumt werden und ein neues Gebäude kann dargestellt werden. Dieselbe Funktion steht für die Manipulation von Meshs zur Verfügung. Mithilfe von Shadern können 3D-Modelle mit unterschiedlichen Materialien drapiert werden. Ein Shader kann dabei verschiedene Texturen drapieren, je nach Attribut des zugrunde liegenden Objektes.

Das ArcGIS SDK for Unity bildet in Zusammenarbeit mit der Unity Game-Engine die Grundlage dieser Arbeit und es wird die Version 1.3.0 verwendet.

2.5 Shader in der Unity Game-Engine

Im Kontext dieser Arbeit wird bei der Nennung von Shadern, von Software basierten Anwendungen gesprochen. Als Hardware-Shader werden kleine Recheneinheiten bezeichnet,

die in moderne Graphikchips integriert sind, um spezialisierte, sehr oft in der Berechnung von Grafikeffekten wiederkehrende Aufgaben effizienter zu verarbeiten.

Shader oder Shader Programme sind eigenständige Anwendungen oder Anwendungsmodule, die in der Regel auf der Grafikkarte eines PCs laufen. Ursprünglich wurden sie entwickelt, um die Schattierung von 3D-Szenen zu ermöglichen. Es handelte sich um eine Reihe von Regeln und Gleichungen, um die korrekte Menge an Licht, Dunkelheit und Farbe in einem Bild zu ermitteln. Heutzutage können Shader eine Vielzahl an Funktionen erfüllen, beispielsweise Spezialeffekte in der Computergrafik oder auch Nachbearbeitung von Videos.

In Unity sind alle Shader Teil der Shader-Klasse. Ein Skript dieser Klasse wird als Shader-Objekt bezeichnet und enthält Informationen über sich selbst wie z.B. den eigenen Namen, einen optionalen Ausweich-Shader und einen oder mehrere Sub-Shader. Shader-Objekte können auf unterschiedliche Weisen bearbeitet werden. Die Unity Game-Engine bietet die Möglichkeit, Shader in einer grafischen Benutzeroberfläche zu bearbeiten, dem sogenannten *Shader Graph*.

Shader Graph bietet die Möglichkeit, Shader-Objekte visuelle zu kreieren, anstatt die Programmierung in Code schreiben zu müssen. Einzelne Funktionen erscheinen als Boxen und diese können dann an vorgegebenen Knotenpunkten miteinander verbunden werden. Diese Funktionen beinhalten beispielsweise Gleichungen, Regeln, Farbeffekte und Logikelemente.

Auf die Konfiguration der Shader für die Zwecke dieser Arbeit wird in einem späteren Teil eingegangen.

3. Datengrundlage

Als Datengrundlage dieser Arbeit dienen Daten, die bereits durch das LGLN und andere Stellen erhoben worden sind. Vom LGLN werden hier die 3D-Gebäudemodelle und die DOPs verwendet. Von der Landeshauptstadt Hannover stammt das DGM.

Der Hauptgrund für diese Vorgehensweise liegt darin, dass die erzeugten Bildpaare als ergänzende Trainingsdaten für das Trainieren eines KNN gedacht sind. Es soll möglichst aufwandsarm sein, die erforderliche 3D-Szene in Unity zu erstellen. Auch wenn die Vorprozessierung der verschiedenen Daten nicht immer unkompliziert ist, so ist dieser Ansatz deutlich effizienter. Eine Alternative wäre beispielsweise eine Szene manuell herzustellen, indem jedes Gebäude per Hand modelliert und texturiert wird. Da insbesondere die 3D-Gebäudemodelle in akzeptabler Qualität vorliegen, und die DOPs im gleichen System georeferenziert sind, besteht keine Notwendigkeit eigene Daten zu erheben. Da die beiden zuvor genannten Datensätze außerdem für das gesamte Landesgebiet Niedersachsens vorliegen, kann durch die in dieser Arbeit beschriebene Methode mit geringen Anpassungen eine 3D-Szene an einer beliebigen Stelle des Landesgebietes erstellt werden und damit können dann im Anschluss Bildpaare für Trainingszwecke erstellt werden. Insbesondere könnten so ländliche und urbane Szenen unkompliziert generiert werden. Es wäre lediglich notwendig ein Geländemodell zu beschaffen, da das in dieser Arbeit verwendete DGM der Landeshauptstadt Hannover nur jene abbildet.

Die 3D-Gebäudemodelle liegen in der Qualität LoD2 vor. Hergestellt werden diese in automatisierter Form und als Grundriss der Gebäudemodelle werden die ALKIS Gebäudegrundrisse verwendet. Die Ableitung der Höhe, sowie einer standardisierten Dachform erfolgt aus Laserscandaten aus einer Befliegung per Flugzeug für das ganze Landesgebiet. Die Aktualität der ALKIS Gebäudegrundrisse entspricht jeweils dem tagesstand der automatischen Ableitung. Die Laserscandaten werden derzeit nicht regelmäßig neuerfasst und daher stammen die Daten aus Befliegungen zwischen 2015 und 2018. Die Lagegenauigkeit der 3D Gebäudemodelle entspricht der Genauigkeit der zugrundeliegenden Gebäudegrundrisse aus ALKIS. Die Höhengenauigkeit beträgt größtenteils 1 m, jedoch kann es durch komplexe Dachformen zu groben Abweichungen kommen. Das Lagebezugssystem für die 3D-Gebäudemodelle ist ETRS89 projiziert im UTM-Koordinatensystem und das Höhenbezugssystem ist das DHHN2016 mit Normalhöhen-Null. Die LoD2-Modelle stehen auf der OpenGeoData-Internetseite des LGLN zum Download bereit im Dateiformat 3D-Shape. Der Objekttyp in ArcGIS Pro ist ein Multipatch-Feature.

Die DOPs werden ebenfalls durch das LGLN erfasst und alle drei Jahre wird das gesamte Landesgebiet beflogen. Aus den orientierten Luftbildern wird im ersten Schritt ein DOM

gebildet (ein sogenanntes bDOM) und anschließend werden die DOPs aus dem Verschnitt des bDOM und der Luftbilder errechnet. Die DOPs haben eine Bodenauflösung von ungefähr 20 cm und sind im ETRS89 georeferenziert. Die DOPs stehen auf der OpenGeoData-Internetseite des LGLN zum Download zur Verfügung und liegen als TIFF-Dateien vor.

Die Elevationsdaten wurden durch die Abteilung Geoinformation der Landeshauptstadt Hannover zur Verfügung gestellt. Sie sind Online auf der Seite der Region Hannover abrufbar und liegen als GeoTIFF im ETRS89/UTM Zone 32 System vor. Erhoben wurden die Daten im Jahr 2010 durch eine Befliegung mittels Laserscanning und anschließend wurde aus der Punktwolke ein Geländemodell mit einer Gitterweite von 1m gerechnet. Zur Genauigkeit und Zuverlässigkeit der Daten werden keine Angaben gemacht.

Die Ausdehnung aller drei zuvor genannten Datensätze ist der Tabelle 1 zu entnehmen.

Tabelle 1: Ausdehnung des Bearbeitungsgebietes in den relevanten Koordinatensystemen (Quelle: Eigene Darstellung)

	ETRS89/UTM Zone32		WGS 1984 Web Mercator	
Ecke	East in m	North in m	East in m	North in m
links oben	546.000	5.808.000	1.077.021	6.876.477
rechts unten	556.000	5.798.000	1.093.541	6.859.916

Im ETRS89/UTM-Zone32-System decken die Daten ein Quadrat mit einer Kantenlänge von 10km ab. Durch die Projektion in das WGS-1984-Web-Mercator-System entsteht durch Scherung und Streckung ein Rechteck mit den Seitenlängen 16,52km und 16,56km.

4. Implementierung

Auch wenn die eigentliche Herstellung der Bildpaare in der Unity Entwicklungsumgebung nicht mehr besonders arbeitsintensiv ist und die Generierung der 3D-Szene weitgehend automatisch abläuft, so müssen die Daten zuerst in ein geeignetes Datenformat und ein von der ArcGIS Maps SDK for Unity verarbeitbares Referenzsystem gebracht werden.

4.1 Vorprozessierung der Daten in ArcGIS Pro

Die Aufbereitung der Daten erfolgt in ArcGIS Pro. Noch bevor die Datensätze transformiert werden, müssen sie auf Vollständigkeit geprüft werden. Dazu werden die 3D-Gebäudemodelle, die DOPs und die Elevationsdaten in ein Projekt eingeladen, in einer Karte dargestellt und anschließend visuell überprüft. Hierbei ist es wichtig festzustellen, ob alle Kacheln vorhanden sind und die Grenzen der abgebildeten Gebiete übereinstimmen. Die Überprüfung hat ergeben, dass die drei verschiedenen Datensätze das in der Tabelle 1 angegebene Gebiet vollständig abdecken im ETRS89/UTM-Zone-32-System.

4.1.1 Vorprozessierung 3D-Gebäudemodelle

Im nächsten Schritt werden zuerst die 3D-Gebäudemodelle aufbereitet. Mit der *Zusammenführen*-Funktion von ArcGIS Pro werden die 25 Einzelkacheln zu einem Layer zusammengefügt.

Der entstandenen zusammengefügten Ebene werden die Felder *Ausrichtung*, *Neigung*, *TEXTURE* und *CLASS* in der Attributtabelle hinzugefügt. Diese werden in den folgenden Schritten mit Inhalt gefüllt.

Um später die Texturen korrekt auf den Dachflächen ausrichten zu können, muss die Richtung jedes einzelnen Gebäudemodelles zum Nordazimut ermittelt werden. Dazu müssen die 3D-Modelle mit der *Multipatch-Footprint* Funktion zu Polygonen transformiert werden. Diese Polygone bilden den Grundriss eines jeden Multipatch-Feature und werden in einer separaten Ebene gespeichert. Diese Polygone können nun mit der Funktion *Hauptwinkel von Polygon berechnen* verarbeitet werden. Hiermit wird die Ausrichtung eines Polygons im Verhältnis zum Nordazimut bestimmt. Abgespeichert werden die Daten für jedes Objekt im Feld *Ausrichtung*. Dieses Feld kann nun an die eigentliche Ebene, welche die Multipatch-Features beinhalten, übertragen werden.

Senkrechte Wände sollen später mit einer weißen Füllfarbe texturiert werden. Die notwendige Information hierzu kann mit der Funktion *Z-Informationen hinzufügen* in ArcGIS Pro ermittelt werden, denn es kann die Neigung einer Multipatch-Oberfläche bestimmt und in dem Feld *Neigung* gespeichert werden. Senkrechten Multipatch-Oberflächen wird ein Wert von minus eins zugewiesen.

Da das ArcGIS Maps SDK for Unity nur ein Attribut pro Ebene an die Unity Game-Engine weitergeben kann, sollen nun alle Informationen in einem einzigen Attribut codiert werden. Kombiniert werden sollen die Attribute *Dachform, Ausrichtung* und *Neigung* und abgespeichert werden soll die Zahlensequenz in den Feldern *TEXTURE* und *CLASS*. Mit der Funktion *Feld berechnen* können kleine Code-Blöcke zur Berechnung eines Feldwertes ausgeführt werden. Die Programmiersprache ist Python und in der Abbildung 1 ist die Methode dargestellt, die zur Codierung der Felder verwendet wurde.

```
def cod(Dachform, Ausrichtung, Neigung):
    if Dachform != 0:
        x = str(Dachform)
    else:
        x = str(1000)

    if Ausrichtung > 0:
        if Ausrichtung >= 10:
            y = "0" + str(Ausrichtung)
        else:
            y = "00" + str(Ausrichtung)
    elif Ausrichtung < 0:
        if Ausrichtung <= -10:
            y = "1" + str(abs(Ausrichtung))
        else:
            y = "10" + str(abs(Ausrichtung))
    else:
        y = "000"

    if Neigung == -1:
        z = str(1)
    else:
        z = str(0)

    return int(x + y + z)

```

Abbildung 1: Methode für die Kodierung des TEXTURE-Attributfeldes (Quelle: Eigene Darstellung)

Es ist zu sehen, dass jeder Ausgabewert eine Länge von acht Ziffern hat. Das Format ist folgendermaßen aufgebaut: XXXXYYYZ. Die ersten vier Ziffern stehen für die Dachform. Diese ist bereits im Datensatz der LoD2-Modelle der Landesvermessung enthalten und ist immer vier Ziffern lang außer der null Wert, welcher den Wert 1000 zugeordnet bekommt.

Der nächste Teil der Ziffernfolge bildet die Ausrichtung des Modells in Bezug auf den Nordazimut ab. Eine Ausrichtung nach Norden ergibt einen Wert von 000, eine Ausrichtung nach Nordost ergibt einen Wert von 045 und eine Ausrichtung nach Nordwest einen Wert von 145. Die Ziffer eins ist hierbei eine Verschlüsselung des Minuszeichens und die null steht für ein Pluszeichen. Eine Ausrichtung nach Südwest ist einer Ausrichtung nach Nordost gleichzusetzen und damit treten nur Werte zwischen -90° und +90° bzw. 190 und 090 auf.

Die letzte Ziffer des Codes kann als ein Boolean verstanden werden. Ein Wert von eins steht für wahr oder in diesem Fall eine senkrechte Wand. Der Wert null für falsch oder keine senkrechte Wand.

Somit enthalten die Felder *TEXTURE* und *CLASS* nun achtstellige Zahlencodes, welche später an die Unity Game-Engine weitergegeben werden können.

Anschließend wird die Ebene mit der *3D-Objekt-Szenen-Layer-Inhalt-erstellen*-Funktion als ein *Szenen-Layer-Paket* exportiert. Gleichzeitig werden die Objekte in das WGS-1984-Web-Mercator-System transformiert. Als Höhensystem wird weiterhin das DHHN2016 verwendet. Das Datenformat *Szenen-Layer-Paket* kann bereits durch das ArcGIS Maps SDK for Unity verwendet werden.

4.1.2 Vorprozessierung DOPs

Die DOPs werden zunächst in ein sogenanntes Mosaik zusammengefasst. Mit der Funktion *Mosaik zu neuem Raster* können die 25 Einzelbilder zu einem großen DOP zusammengefasst werden und gleichzeitig wird das entstehende Mosaik in das WGS-1984-Web-Mercator-System überführt. Dieses kann nun in einem Kartendokument mit dem richtigen Koordinatensystem dargestellt werden. Es ist wichtig hierbei kein Höhensystem zu definieren, da der Rasterdatensatz sonst später auf der Höhe Null Meter in der Unity Game-Engine dargestellt wird. Mit dem Kartendokument wird im nächsten Schritt ein Kachelschema erstellt. Dieses gibt sowohl die Kacheldimensionen als auch die Maßstabsebenen und die Orientierung der Kacheln im Raum vor. In ArcGIS Pro geht dies mit der Funktion *Kachelschema für Kartenserver-Cache erstellen*. Definiert werden die Maßstäbe 1:1.000 bis 1:256.000 bei einer Auflösung von 254dpi. Die Kacheln die später generiert werden sollen, werden eine Größe von 256x256px haben. Der Ursprungspunkt der Kacheln wird bei East: 1.000.000m und North: 6.900.000m im WGS-1984-Web-Mercator-System definiert.

Nachdem das Kartenschema definiert wurde, kann nun ein Kartenkachelpaket mit der Funktion *Kartenkachelpaket erstellen* erstellt werden. Zuerst wird das Kartendokument ausgewählt, welches das Mosaik der 25 Einzelbilder beinhaltet. In der Karte dürfen nur die DOPs dargestellt sein, da alle Karteninhalte zu einem Rasterdatenobjekt zusammengefasst werden. Das bedeutet auch, dass die DOPs nicht in der Darstellung abgewählt sein dürfen, da sonst ein leeres Kartenkachelpaket entsteht. Als Kachelschema wird das zuvor definierte Schema gewählt und als Kachelformat das PNG-Format. Die minimale Detailierungsebene ist der Maßstab 1:256.000 und die maximale Detailierungsebene ist der Maßstab 1:1.000. Als Ergebnis liegt nun ein Kartenkachelpaket im Dateiformat *.tpkx* vor, welches später in das Unity-Projekt eingebunden werden kann.

4.1.3 Vorprozessierung DGM

Die Aufbereitung des DGM für die Verwendung in der Unity Game-Engine folgt ähnlichen Schritten. Zuerst werden die 400 Einzelbilder in ein lokales Szenendokument geladen. Jedes Bild hat eine Größe von 500x500px und jeder Pixel ist einen Quadratmeter groß im ETRS89/UTM-Zone-32-System. Damit ergibt sich eine Ausdehnung von 500x500m pro Bild. Wie schon zuvor bei der Aufbereitung der DOPs, werden die Einzelbilder zunächst mit der

Funktion *Mosaik zu neuem Raster* in ein einziges Rasterdatenobjekt überführt und gleichzeitig erfolgt die Transformation in das WGS-1984-Web-Mercator-System.

Mit der Funktion *Kachelschema für Kachel-Cache erstellen* wird nun auch für das DGM ein Kachelschema erstellt. Grundlage bildet das Mosaik, welches im Schritt zuvor erstellt worden ist. Wie schon für die DOPs werden auch hier die Maßstabsebenen von 1:1.000 bis 1:256.000 definiert und als Ursprungspunkt für die Kacheln wird der Punkt East: 1.000.000m und North: 6.900.000m im WGS-1984-Web-Mercator-System gewählt. Die Auflösung beträgt 254dpi und die Größe der zu generierenden Kacheln 256x256px. Damit das DGM später in der Unity Game-Engine verwendet werden kann, muss als Kachelformat die LERC-Komprimierung ausgewählt werden. „Limited Error Raster Compression“ ist eine effiziente, aber verlustbehaftete Komprimierungsmethode, die vor allem für Höhendaten verwendet wird. Der LERC-Fehler wird mit null Millimeter definiert.

Im nächsten Schritt wird mit der Funktion *Kachel-Cache verwalten* ein Kachel-Cache erzeugt. Dieser muss an einem Ort gespeichert werden, wo er für die spätere Verwendung wiederauffindbar ist. Als Datenquelle dient das erzeugte Mosaik und als Kachelschema muss das zuvor Erstellte ausgewählt werden. Mit der Funktion *Kachel-Cache exportieren* werden nun die im Kachel-Cache abgelegten Kacheln in ein Kachelpaket exportiert. Dieses hat ebenfalls das Dateiformat *.tpkx* und kann durch das ArcGIS Maps SDK for Unity gelesen werden.

Die drei Datensätze, mit denen die 3D-Szene erstellt werden soll, liegen nun in den benötigten Datenformaten *Szenen-Layer-Paket (.slpk)* und *Kartenkachelpaket (.tpkx)* vor und können durch das ArcGIS Maps Sdk for Unity verarbeitet werden.

4.2 Erstellung des Unity-Projektes

Für diese Arbeit wird die Unity-Editor Version 2022.3.8f1 verwendet. Als erstes wird ein Unity-Projekt in der *3D-High-Definition-Rendering-Pipeline* erstellt. Dieses bietet Komptabilität mit dem ArcGIS Maps SDK for Unity, welches auf der Esri Webseite heruntergeladen werden kann. Verwendet wurde die Version 1.3.0. In der Unity Game-Engine kann das ArcGIS Maps SDK for Unity mit dem Package-Manager zum Projekt hinzugefügt werden und die Versuchsdaten hinzugefügt werden.

4.3 Konfiguration der 3D-Szene

Das ArcGIS Maps SDK for Unity bietet drei verschiedene Optionen, die 3D-Szene zu konfigurieren. Es gibt ein graphisches Interface, welches die einfache Darstellung von vordefinierten Esri-Diensten und eigenen Inhalten ermöglicht. Von Interesse sind hier vor allem das Elevationsmodell sowie verschiedene Luftbilddienste, die es ermöglichen, ohne große

Vorkenntnisse eine einfache 3D-Szene zu erzeugen. Für Anwendende, die mit der Unity Architektur vertraut sind, gibt es die Möglichkeit mit Komponenten zu arbeiten. In dieser Arbeit soll nicht weiter auf diese Methode eingegangen werden, da sie für mit der Unity Game-Engine vertrauten Entwicklern eine einfache Alternative darstellt, schnell eine 3D-Szene zu erstellen, jedoch hat sie dieselben Limitierungen wie die Methode über die graphische Benutzeroberfläche. In beiden Fällen können 3D-Objekte nicht mit eigenen Shadern manipuliert werden und somit sind sie nicht geeignet für das Ziel dieser Arbeit.

Verwendet wird also die dritte Methode, die API-Methode. In der Unity Game-Engine können mit C#-Skripten 3D-Szenen erstellt und manipuliert werden. Als Vorlagen dienen das *SampleAPIMapCreator*-Skript, die Dokumentation auf der ArcGIS Entwickler Webseite sowie das *Sample3DAttributesComponent*-Skript.

```
18
19  public class API_Hannover : MonoBehaviour
20  {
21      //API
22      public string APIKey =
          "AAPK5411673525944aae9c4a539b748feaf7JmIHbFmgSiknm39s7bWYKEc2BLW9iKgul
          gRw8n4DrwGuBC0ZGQ4Din-oEtWwtp2b";
23
```

Abbildung 2: Ausschnitt aus dem C#-Skript API-Hannover Nr.1 (Quelle: Eigene Darstellung)

Wie in Abbildung 2 zu sehen, wird zunächst eine Klasse als *API-Hannover* deklariert. In dieser wird im ersten Schritt eine Variable für den API-Schlüssel definiert, welcher für die Authentifizierung benötigt wird und dieser kann mit einem ArcGIS Entwicklerkonto generiert werden.

```
23
24      //map
25      private ArcGISMapComponent arcGISMapComponent;
26      private ArcGISPoint geographicCoordinates = new ArcGISPoint(1084000,
          6868000, 0, ArcGISSpatialReference.WebMercator());
27
28      //camera
29      private ArcGISCameraComponent cameraComponent;
30
```

Abbildung 3: Ausschnitt aus dem C#-Skript API-Hannover Nr.2 (Quelle: Eigene Darstellung)

In Abbildung 3 wird in Zeile 25 die Kartenobjektkomponente und in Zeile 29 die Kameraobjektkomponente zum Unity-Projekt hinzugefügt. Zudem wird in Zeile 26 ein Punktobjekt für die spätere Verwendung definiert. Dieses hat die Koordinaten East: 1.084.000m und North: 6.868.000m im WGS-1984-Web-Mercator-System.

```
30
31      //3D-Attributes
32      public delegate void SetLayerAttributesEventHandler
          (Esri.GameEngine.Layers.ArcGIS3DObjectSceneLayer layer);
33      public event SetLayerAttributesEventHandler OnSetLayerAttributes;
34
```

Abbildung 4: Ausschnitt aus dem C#-Skript API-Hannover Nr.3 (Quelle: Eigene Darstellung)

Der Code der Zeilen 32 und 33 in der Abbildung 4 dient dazu, später 3D-Ebenen definieren zu können, welche durch Shader in der 3D-Szene texturiert werden können.

```
34
35      private void CreateArcGISMapComponent()
36      {
37          arcGISMapComponent = FindObjectOfType<ArcGISMapComponent>();
38
39          if (!arcGISMapComponent)
40          {
41              var arcGISMapGameObject = new GameObject("ArcGISMap");
42              arcGISMapComponent =
                  arcGISMapGameObject.AddComponent<ArcGISMapComponent>();
43          }
44
45          arcGISMapComponent.OriginPosition = geographicCoordinates;
46          arcGISMapComponent.MapType =
              Esri.GameEngine.Map.ArcGISMapType.Local;
47
48          arcGISMapComponent.MapTypeChanged += new
              ArcGISMapComponent.MapTypeChangedEventHandler(CreateArcGISMap);
49      }
50
```

Abbildung 5: Ausschnitt aus dem C#-Skript API-Hannover Nr.4 (Quelle: Eigene Darstellung)

In der Abbildung 5 ist weiterer Code dargestellt. In den Zeilen 35 bis 43 wird die Kartenobjektkomponente modifiziert. Als Name wurde *ArcGISMap* vergeben und in der Zeile 45 wird als Zentrum der Karte das zuvor definierte Punktobjekt definiert. In Zeile 46 wird als Kartentyp *local* gesetzt. In der Unity Game-Engine kann eine lokale oder globale Szene erstellt werden. Für die Zwecke dieser Arbeit wird ein kleiner Ausschnitt von Hannover visualisiert und daher reicht es aus, eine lokale Szene zu erstellen. In einer lokalen Szene kann außerdem der Darstellungsausschnitt begrenzt werden und damit kann die Performance der Szene verbessert werden. Mit dem Code der Zeile 48 wird das Kartenobjekt erstellt mithilfe einer Methode, die im Folgenden definiert wird.

```
50
51      //map
52      public void CreateArcGISMap()
53      {
54          var arcGISmap = new Esri.GameEngine.Map.ArcGISMap
              (arcGISMapComponent.MapType);
55
56          //add Elevation-Modell
57          arcGISmap.Elevation = new Esri.GameEngine.Map.ArcGISMapElevation
              (new Esri.GameEngine.Elevation.ArcGISImageElevationSource("C:/
              Users/jannik/Documents/Test Hannover/DGM_TIFF/
              DGM_MERCATOR.tpkx", "Terrain 3D", ""));
58
59          //add aerial Photographie
60          var layer_1 = new Esri.GameEngine.Layers.ArcGISImageLayer("C:/
              Users/jannik/Documents/Test Hannover/DOP/DOP_MERCATOR.tpkx",
              "DOP", 1.0f, true, "");
61          arcGISmap.Layers.Add(layer_1);
62
63          if (counter2 % 2 == 1)
64          {
65              layer_1.IsVisible = true;
66          }
67          else
68          {
69              layer_1.IsVisible = false;
70          }
71
72          //add 3D-Building-Modells
73          var buildingLayer = new
              Esri.GameEngine.Layers.ArcGIS3DObjectSceneLayer("C:/Users/
              jannik/Documents/Test Hannover/LoD2/LoD2_MERCATOR.slpk",
              "Building Layer", 1.0f, true, "");
74          arcGISmap.Layers.Add(buildingLayer);
75
```

Abbildung 6: Ausschnitt aus dem C#-Skript API-Hannover Nr.5 (Quelle: Eigene Darstellung)

In der Zeile 54 der Abbildung 6 wird das Kartenobjekt der 3D-Szene erstellt als ArcGISMap-Objekt. In der Zeile 57 der Abbildung 6 wird als Elevationsmodell das DGM eingefügt, welches in ArcGIS Pro angepasst wurde. In der Zeile 60 und 61 wird als Grundkarte das Kartenkachelpaket hinzugefügt, welches die DOPs beinhaltet und benannt wird es als *layer_1*. In den Zeilen 73 und 74 werden die 3D-Gebäudemodelle als *buildingLayer* hinzugefügt. Damit sind nun alle drei zuvor verarbeiteten Datensätze in das Unity-Projekt integriert. Die Zeilen 63 bis 70 dienen der Abfrage, ob die DOP-Ebene gerade sichtbar sein soll oder ob die Ebene gerade nicht dargestellt werden soll. In einem späteren Teil wird die Variable *counter2* definiert und an der Stelle wird auch der Zweck dieser Abfrage deutlich.

```
75
76          //3D-Attributes
77          if (OnSetLayerAttributes != null)
78          {
79              OnSetLayerAttributes(buildingLayer);
80          }
81
82          //extend
83          arcGISMapComponent.EnableExtent = true;
84
85          var extentCenter = geographicCoordinates;
86          var extent = new ArcGISExtentCircle(extentCenter, 3000);
87
88          arcGISmap.ClippingArea = extent;
89
90          arcGISMapComponent.View.Map = arcGISmap;
91
```

Abbildung 7: Ausschnitt aus dem C#-Skript API-Hannover Nr.6 (Quelle: Eigene Darstellung)

Mit den Zeilen 77 bis 80 in der Abbildung 7 wird die Ebene mit dem Namen *buildingLayer* als attributierbare 3D-Ebene gekennzeichnet. Dies ermöglicht es die 3D-Modelle in der Ebene zu texturieren und zu klassifizieren. Diese Ebene enthält die 3D-Gebäudemodelle. Der letzte Teil der Methode in den Zeilen 83 bis 90 definiert die Ausdehnung des in der Unity Game-Engine dargestellten Ausschnittes, welcher für diese Arbeit eine kreisrunde Darstellung mit einem Radius von 3000m zeigt.

```
96      private void CreateArcGISCamera()
97      {
98          cameraComponent =
              Camera.main.gameObject.GetComponent<ArcGISCameraComponent>();
99
100         if (!cameraComponent)
101         {
102             var cameraGameObject = Camera.main.gameObject;
103
104             cameraGameObject.transform.SetParent
                  (arcGISMapComponent.transform, false);
105
106             cameraComponent =
                  cameraGameObject.AddComponent<ArcGISCameraComponent>();
107
108             cameraGameObject.AddComponent<ArcGISCameraControllerComponent>
                  ();
109
110             cameraGameObject.AddComponent<ArcGISRebaseComponent>();
111         }
```

Abbildung 8: Ausschnitt aus dem C#-Skript API-Hannover Nr.7 (Quelle: Eigene Darstellung)

Der Code der Zeilen 96 bis 111, in der Abbildung 8 zu sehen, dient dazu, das Kameraobjekt zu erstellen. Dies geschieht auf Grundlage der in Zeile 29 eingefügten Kamerakomponente.

```
112
113         var cameraLocationComponent =
              cameraComponent.GetComponent<ArcGISLocationComponent>();
114
115         if (!cameraLocationComponent)
116         {
117             cameraLocationComponent =
                  cameraComponent.gameObject.AddComponent<ArcGISLocationCompon
                  ent>();
118
119             cameraLocationComponent.Position = geographicCoordinates;
120             cameraLocationComponent.Rotation = new ArcGISRotation(0, 0,
                  0);
121         }
122     }
123
```

Abbildung 9: Ausschnitt aus dem C#-Skript API-Hannover Nr.8 (Quelle: Eigene Darstellung)

In den Codezeilen der Abbildung 9 wird die Position der Kamera in der Zeile 119 definiert. Es handelt sich um denselben Punkt wie das Zentrum der Karte. In Zeile 120 wird die voreingestellte Ausrichtung der Kamera definiert. Mit den zu sehenden Einstellungen schaut die Kamera senkrecht nach unten und in Richtung Norden.

```
123
124     //sky
125     private void CreateSkyComponent()
126     {
127 #if USE_HDRP_PACKAGE
128         var currentSky = FindObjectOfType<UnityEngine.Rendering.Volume>();
129         if (currentSky)
130         {
131             ArcGISSkyRepositionComponent skyComponent =
                  currentSky.gameObject.GetComponent<ArcGISSkyRepositionCompon
                  ent>();
132
133             if (!skyComponent)
134             {
135                 skyComponent =
                      currentSky.gameObject.AddComponent<ArcGISSkyRepositionCo
                      mponent>();
136             }
137
138             if (!skyComponent.arcGISMapComponent)
139             {
140                 skyComponent.arcGISMapComponent = arcGISMapComponent;
141             }
142
143             if (!skyComponent.CameraComponent)
144             {
145                 skyComponent.CameraComponent = cameraComponent;
146             }
147         }
148 #endif
149     }
150
```

Abbildung 10: Ausschnitt aus dem C#-Skript API-Hannover Nr.9 (Quelle: Eigene Darstellung)

Mit dem Code in der Abbildung 10 wird eine *Skybox* hinzugefügt. Diese gibt einen Hintergrund und hat durch Partikeleffekte einen Einfluss auf die Beleuchtung der Szene. Später kann in dieser Komponente das Nebelvolumen und die Ausrichtung der Sonne bearbeitet werden. Für die Zwecke dieser Arbeit wird ein voreingestelltes Profil von Esri verwendet.

```
150
151     void Start()
152     {
153         CreateArcGISMapComponent();
154         CreateArcGISCamera();
155         CreateSkyComponent();
156         CreateArcGISMap();
157     }
158
```

Abbildung 11: Ausschnitt aus dem C#-Skript API-Hannover Nr.10 (Quelle: Eigene Darstellung)

In der Abbildung 11 sind die Zeilen 151 bis 157 zu sehen. Mit diesen Befehlen werden die einzelnen Komponenten am Anfang der Simulation gestartet. Im Gegensatz dazu wird der Code, welcher in der Abbildung 12 zwischen den Zeilen 163 und 177 steht bei jedem Frame der Simulation neu ausgeführt. In diesem Fall ist damit eine Screenshot Funktion implementiert worden. Mit den Zeilen 171 bis 176 in Verbindung mit dem Code, welcher in den Zeilen 63 bis 70 steht, kann die DOP-Ebene ein- und ausgeblendet werden. Damit ergibt sich für die Klassifizierungsmaske ein neutraler Hintergrund.

```
158
159     //Screenshot-Function
160     int counter = 1;
161     int counter2 = 1;
162     void Update()
163     {
164         if (Input.GetMouseButtonDown(0))
165         {
166             ScreenCapture.CaptureScreenshot("./Bilder/Texture_" +
                  counter.ToString() + ".png");
167             Debug.Log("Click");
168             counter++;
169         }
170
171         if (Input.GetKeyDown("c"))
172         {
173             counter2++;
174             CreateArcGISMap();
175             Debug.Log("c");
176         }
177     }
178 }
```

Abbildung 12: Ausschnitt aus dem C#-Skript API-Hannover Nr.11 (Quelle: Eigene Darstellung)

Hiermit ist die Basis für die 3D-Szene geschaffen. Um das Skript in der Unity Game-Engine darstellen zu können, muss es als Komponente zum *MainCamera*-Objekt hinzugefügt werden. Als Ergebnis wird im Unity-Editor nun das DGM angezeigt, über welches die DOPs als Ebene drapiert sind und die LoD2-Gebäusdemodelle sind bisher noch als einfache weiße Blöcke mit Dachform dargestellt. Die Beleuchtung kann so angepasst werden, dass der Schattenwurf aus der Unity-Simulation den natürlichen Schattenwurf aus der Luftbildfotografie abdeckt.

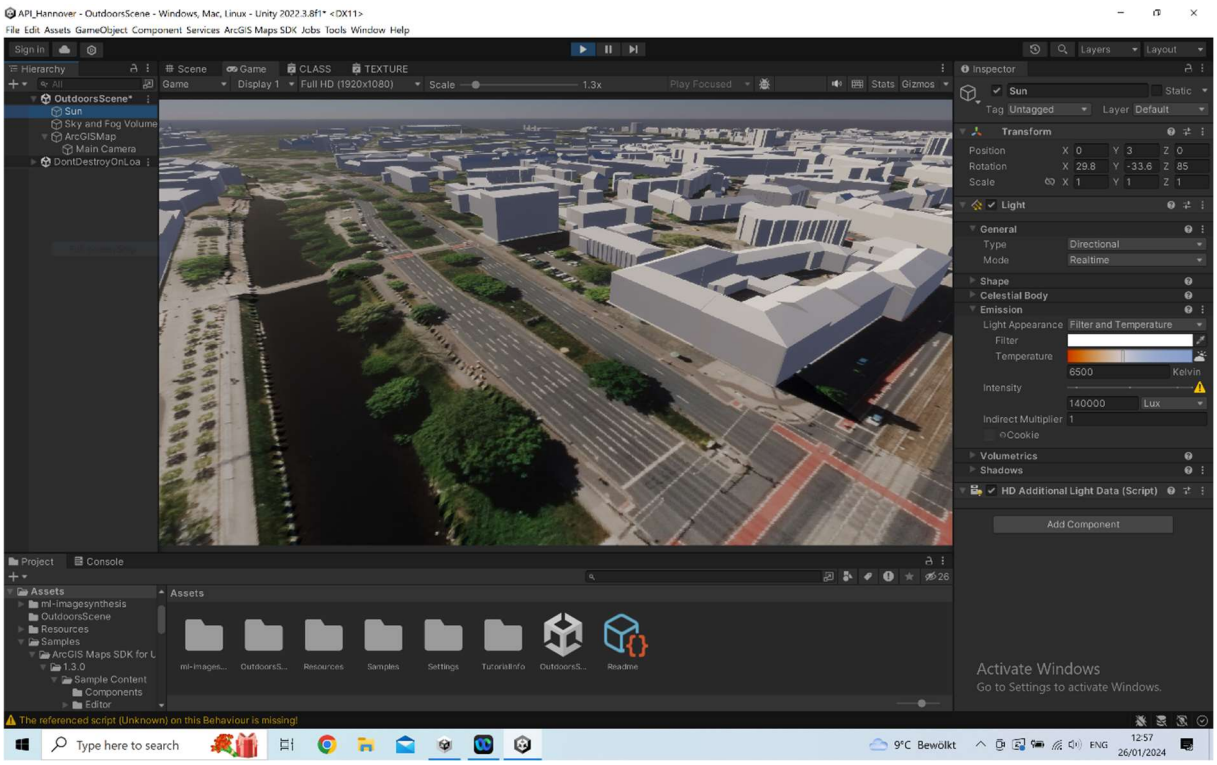


Abbildung 13: Bearbeitungsstand in der Unity Game-Engine nach der Erstellung des API-Hannover-Skriptes (Quelle: Eigene Darstellung)

In Abbildung 13 ist der derzeitige Bearbeitungsstand zu sehen. Im Vordergrund auf der linken Seite des Simulationsausschnittes ist zu erkennen, dass das Gelände zum Gewässer hinabfällt und einen Graben bildet. Über dem DGM ist das Kartenkachelpaket mit den DOPs drapiert. Ebenso sind auf der rechten Bildseite die LoD2-Gebäudemodelle zu sehen, welche derzeit noch keine Texturen oder Klassifizierungen aufweisen.

4.4 Anpassung des 3D-Attribute-Srikptes

Dieser Abschnitt behandelt die notwendigen Schritte, um die Darstellung von Texturen auf den 3D-Gebäudemodellen zu ermöglichen. Als Vorlage hierfür dient das *Sample3DAttributesComponent*-Skript, welches unter den Beispiel-Dateien des ArcGIS Maps SDK for Unity zu finden ist.

In der Abbildung 14 ist der erste Teil des Codes zu sehen. Zuerst wird die Klasse *Sample3DAttributesComponent* definiert. Anschließend wird eine Liste mit den Werten „None“, „TEXTURE“ und „CLASS“ angelegt. Dies sind später die Auswahlmöglichkeiten, die in der Unity Game-Engine zur Verfügung stehen werden, wobei „None“ der derzeitigen, texturlosen Ansicht entspricht, „TEXTURE“ die generischen Texturen über die 3D-Modelle drapiert darstellen wird und „CLASS“ die Klassifizierung zeigen wird.

```
14
15  [ExecuteAlways]
16  public class Sample3DAttributesComponent : MonoBehaviour
17  {
18      //Define Attributes
19      public enum AttributeType
20      {
21          None,
22          TEXTURE,
23          CLASS
24      };
25
```

Abbildung 14: Ausschnitt aus dem C#-Skript Sample3DAttributesComponent Nr.1 (Quelle: Eigene Darstellung)

In den Codezeilen 33 bis 44 der Abbildung 15 wird eine Verbindung zu dem zuvor erstellten Skript hergestellt. Mit diesem Teil wird der im letzten Abschnitt eingefügte Code-Teil aktiviert.

```
31
32      //Connection to API_Hannover
33      private API_Hannover sampleMapCreator;
34
35      private void Awake()
36      {
37          //API_Hannover
38          sampleMapCreator = GetComponent<API_Hannover>();
39
40          if (!sampleMapCreator)
41          {
42              Debug.LogError("SampleAPIMapCreator not found");
43              return;
44          }
45
```

Abbildung 15: Ausschnitt aus dem C#-Skript Sample3DAttributesComponent Nr.2 (Quelle: Eigene Darstellung)

Zeilen 76 bis 84 und 86 bis 94 der Abbildung 16 definieren jeweils Methoden für eine Verbindung zu den spezifischen Materialien. In einem späteren Teil dieser Arbeit wird auf die Erstellung der Materialien und Shadern eingegangen.

```
75
76      private void Setup3DAttributesTexture
          (Esri.GameEngine.Layers.ArcGIS3DObjectSceneLayer layer)
77      {
78          var layerAttributes = ArcGISImmutableArray<String>.CreateBuilder();
79          layerAttributes.Add("TEXTURE");
80          layer.SetAttributesToVisualize(layerAttributes.MoveToArray());
81          Debug.Log("Attribute (TEXTURE) geladen");
82
83          layer.MaterialReference = new Material(Resources.Load<Material>
              ("Materials/" + DetectRenderPipeline() + "/TEXTURE"));
84      }
85
86      private void Setup3DAttributesClass
          (Esri.GameEngine.Layers.ArcGIS3DObjectSceneLayer layer)
87      {
88          var layerAttributes = ArcGISImmutableArray<String>.CreateBuilder();
89          layerAttributes.Add("CLASS");
90          layer.SetAttributesToVisualize(layerAttributes.MoveToArray());
91          Debug.Log("Attribute (CLASS) geladen");
92
93          layer.MaterialReference = new Material(Resources.Load<Material>
              ("Materials/" + DetectRenderPipeline() + "/CLASS"));
94      }
95
```

Abbildung 16: Ausschnitt aus dem C#-Skript Sample3DAttributesComponent Nr.3 (Quelle: Eigene Darstellung)

In der Abbildung 17 wird überprüft, welche Auswahlmöglichkeit als Darstellung gewählt wurde und dann wird die jeweilige Methode angesprochen.

```
58      //Setup Connection to Shaders
59      private void Setup3DAttributes(ArcGIS3DObjectSceneLayer buildingLayer)
60      {
61          if (buildingLayer == null)
62          {
63              return;
64          }
65
66          if (layerAttribute == AttributeType.TEXTURE)
67          {
68              Setup3DAttributesTexture(buildingLayer);
69          }
70          else if (layerAttribute == AttributeType.CLASS)
71          {
72              Setup3DAttributesClass(buildingLayer);
73          }
74      }
75
```

Abbildung 17: Ausschnitt aus dem C#-Skript Sample3DAttributesComponent Nr.4 (Quelle: Eigene Darstellung)

Zum Schluss wird in der Abbildung 18 noch Code dargestellt, der zur Erkennung der Rendering-Pipeline dient. In dieser Arbeit wurde die *Highdefinition Render Pipeline (HDRP)* verwendet und daher wird für alle Darstellungsoptionen diese verwendet.

```
95
96      //Detecting Rendering Pipeline
97      private string DetectRenderPipeline()
98      {
99          if (GraphicsSettings.renderPipelineAsset != null)
100         {
101             var renderType = GraphicsSettings.renderPipelineAsset.GetType
                  ().ToString();
102
103             if (renderType ==
                  "UnityEngine.Rendering.Universal.UniversalRenderPipelineAsset"
                  )
104             {
105                 return "URP";
106             }
107             else if (renderType ==
                  "UnityEngine.Rendering.HighDefinition.HDRenderPipelineAsset" )
108             {
109                 return "HDRP";
110             }
111         }
112
113         return "Legacy";
114     }
115 }
116
```

Abbildung 18: Ausschnitt aus dem C#-Skript Sample3DAttributesComponent Nr.5 (Quelle: Eigene Darstellung)

Das fertig programmierte Skript kann nun dem *Main Camera* Objekt der 3D-Szene des Unity-Editors hinzugefügt werden. Zu finden ist dies im linken Teil des Editorfensters unter dem Herachiepunkt *ArcGISMap*. Es kann entweder per Drag-and-Drop oder über die *Komponenten hinzufügen* Funktion angehangen werden. Es kann nun der Shader ausgewählt werden, mit welchem die 3D-Gebäudemodelle dargestellt werden sollen.

4.5 Shader Graph Konfiguration

Die Konfiguration der Shader-Objekte erfolgt in *Shader Graph*. In diesem Abschnitt wird auf die erforderlichen Schritte eingegangen, um sowohl den Texturierungs- als auch den Klassifizierungs-Shader zu erzeugen.

Als Vorlage für beide Shader-Objekte dienen der *BuildingNameRenderer* und der *ConstructionYearRenderer*. In *Shader Graph* werden mathematische, verlinkende oder anderwärtige Manipulationen von Eingangswerten in Fenstern dargestellt. Diese Fenster

haben verschiedenste Schnittstellen, die dann durch Verbindungen miteinander in Beziehung gebracht werden können.

4.5.1 Konfiguration des TEXTURE-Shaders in Schader Graph

In der Abbildung 19 ist der Anfang des *TEXTURE*-Shader in Shader Graph zu sehen. Um die einzelnen Komponenten besser erklären zu können, ist die Darstellung aufgeteilt. Eine Vollständige Darstellung ist im Anhang 1 zu finden. Der im Folgenden beschriebene Teil ist für die Überprüfung auf senkrechte Wände zuständig.

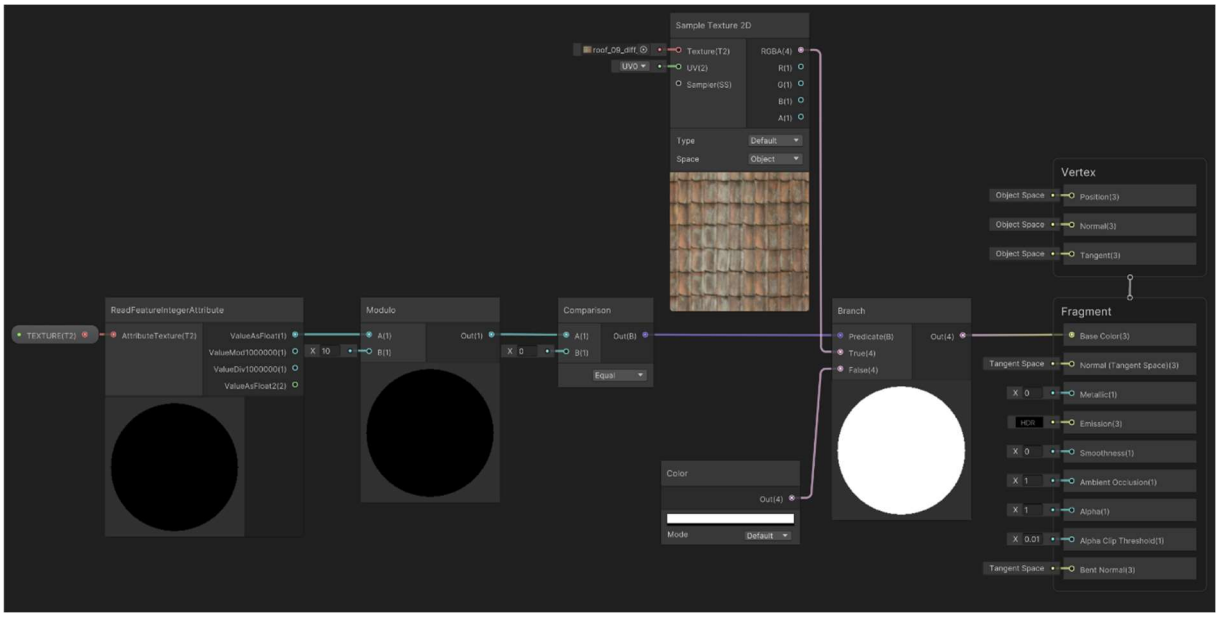


Abbildung 19:Übersicht über die Abfrage "Senkrechte Wand?" in Shader Graph (Quelle: Eigene Darstellung)

Auf der linken Seite erfolgt die Übergabe des *TEXTURE* Attributfeldes. Dies erfolgt durch die Funktion *ReadFeatureIntegerAttribute*. Um die letzte Ziffer des Codes mit der Form XXXXYYYZ zu erhalten, wird die Zahl durch zehn geteilt mit dem Modulo Operator. Der Betrag am Ausgabeknoten der Modulo Funktion ist durch den Aufbau des Codes entweder null oder eins. Daher wird mit der nächsten Funktion ermittelt, ob der Wert gleich null ist mit der Funktion *Comparison*. Der Ausgabeknoten dieser Funktion hat den Datentyp Boolean und gibt ein *falsch* aus, wenn der Eingabewert den Wert eins hat. Mit der nächsten Funktion *Branch* wird nun eine Entscheidung getroffen. Wenn der Booleanwert am Eingabeknoten den Wert *falsch* hat, dann wird als Textur eine einfache weiße Füllfarbe ausgegeben. Sollte der Booleanwert *wahr* sein, dann wird eine Dachtextur aufgebracht. Als Beispiel ist hier eine rote Dachziegeltextur zu sehen. Die Auswahl der Textur wird im nächsten Schritt beschrieben. Der rechte Teil der Abbildung enthält die Knoten, die am Ende die Darstellung der Textur auf dem 3D-Modell veranlassen. Für diese Funktion reicht es den Vektor mit vier Werten des Ausgabeknoten der *Branch* Funktion mit dem *Base Color* Eingangsknoten zu verbinden.

In der Abbildung 20 ist der Anfang der Zuordnung der Textur nach Attributwert der Dachform zu sehen. Wie schon zuvor erfolgt die Übergabe des *TEXTURE* Attributfeldes durch die Funktion *ReadFeatureIntegerAttribute*. Von dort wird der Attributwert durch 10.000 geteilt und abgerundet, damit nun der Dachform Code vorliegt.

Im unteren Teil der Abbildung wird die Textur vorbereitet. Mit der Funktion *Position* wird die folgende Textur als ein 3D-Objekt definiert. Mit der Funktion *RotateAboutAxis* wird die Richtung angepasst, aus welcher die Textur aufgebracht wird. Standardmäßig wird die Textur senkrecht in Nordrichtung aufgebracht. Sollte ein Beobachter aus der Vogelperspektive auf die Szene schauen, wie es bei einem Luftbild der Fall wäre, so würde es auf den Dächern der 3D-Gebäudemodelle zu Verzerrungen der Textur kommen. Daher wird mit der genannten Funktion die Hauptrichtung so geändert, dass die Texturen in der Horizontalebene aufgebracht werden. Es ist also eine Rotation um die x-Achse des Raumes um 90° erforderlich. Mit der nächsten Funktion kann die Kachelung der Textur angepasst werden. Die Größe einer Texturkachel kann individuell in der x- und y-Ebene eingestellt werden. In diesem Beispiel wurde mit der *TilingAndOffset* Funktion die Textur in beide Richtungen mit dem Wert 0,3 skaliert.

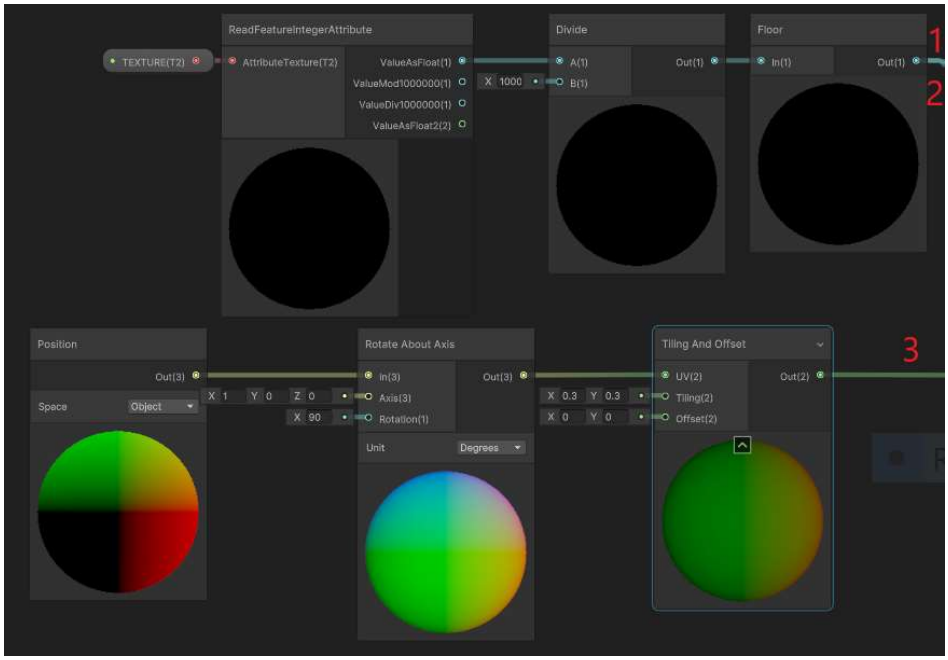


Abbildung 20: Übersicht über die Texturzuordnung in Abhängigkeit von der Dachform in Shader Graph Teil 1 (Quelle: Eigene Darstellung)

Der Rest der Texturzuordnung ist in der Abbildung 21 zu sehen. Anhand der roten Zahlen ist zu erkennen, wo die Pfade aus der vorherigen Abbildung anschließen.

Im unteren Teil werden die Vorgaben für eine Textur mit dem Pfad der Ziffer drei übergeben. Die Funktion *Rotate* erhält den Betrag, um den die Textur rotiert werden soll aus dem Code des Attributfeldes *TEXTURE*. Die Dekodierung dieses Teils des Codes wird in einem späteren

Teil dieser Arbeit beschrieben und die Übergabe dieses Wertes erfolgt am Eingabeknoten *Rotation*. In der nächsten Funktion *SampleTexture2D* wird die eigentliche Textur eingebracht.

Mit dem oberen Teil der Abbildung wird überprüft, ob der richtige Zahlenwert vorliegt. In diesem Beispiel muss die Zahl 3100 vorliegen. Dieser Wert steht für ein Satteldach. Mit dem Pfad eins und zwei wird der reduzierte Wert aus dem Attributfeld *TEXTURE* eingebracht. Wie im Abschnitt zuvor wird durch die *Comparision* Funktion ein Boolean Wert geschaffen. Dieser wird in der *Branch* Funktion überprüft. Sollte der Wert *wahr* sein, dann wird als Textur die Dachziegel-Textur eingefügt.

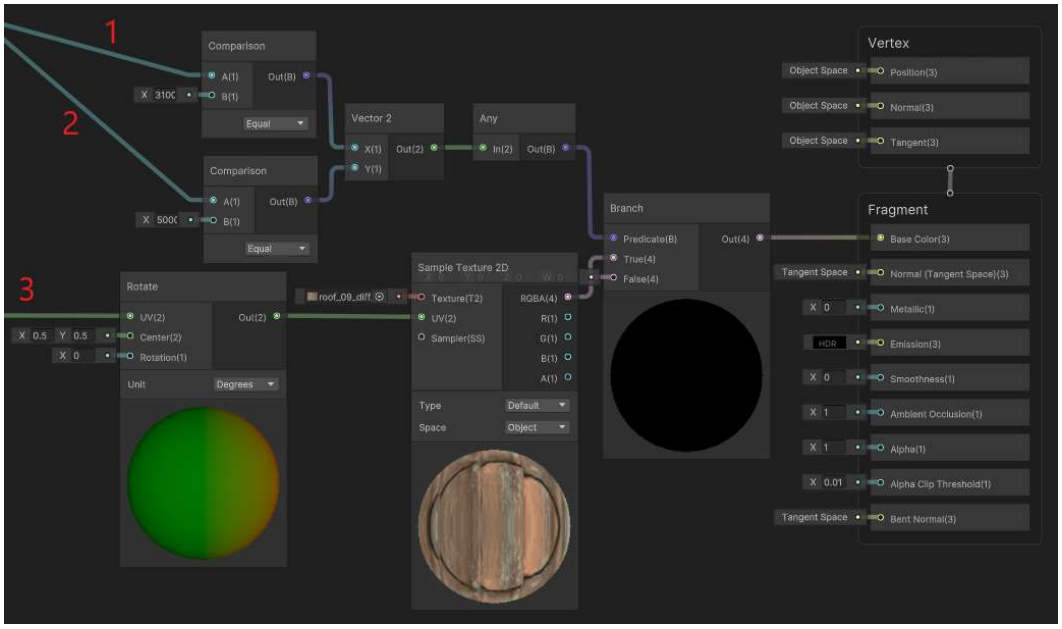


Abbildung 21: Übersicht über die Texturzuordnung in Abhängigkeit von der Dachform in Shader Graph Teil 2 (Quelle: Eigene Darstellung)

Mit der nächsten Abbildung wird der Rotationswert dekodiert. Zuerst wird der Z-Wert des XXXXYYYZ- Codes entfernt. Anschließend wird mit dem *Modulo* Operator die erste Ziffer der verbliebenen Zahlenfolge übertragen. Auch im Folgenden werden durch den *Divide*, den *Floor* und den *Modulo* Operator die einzelnen Ziffern herausgestellt. Im rechten Teil der Abbildung 22 wird noch überprüft, ob es sich um einen negativen oder einen positiven Wert handelt. Die Kombination aller Zwischenergebnisse ergibt am Ende den Rotationswert, um den die Textur auf dem spezifischen 3D-Modell rotiert werden muss.

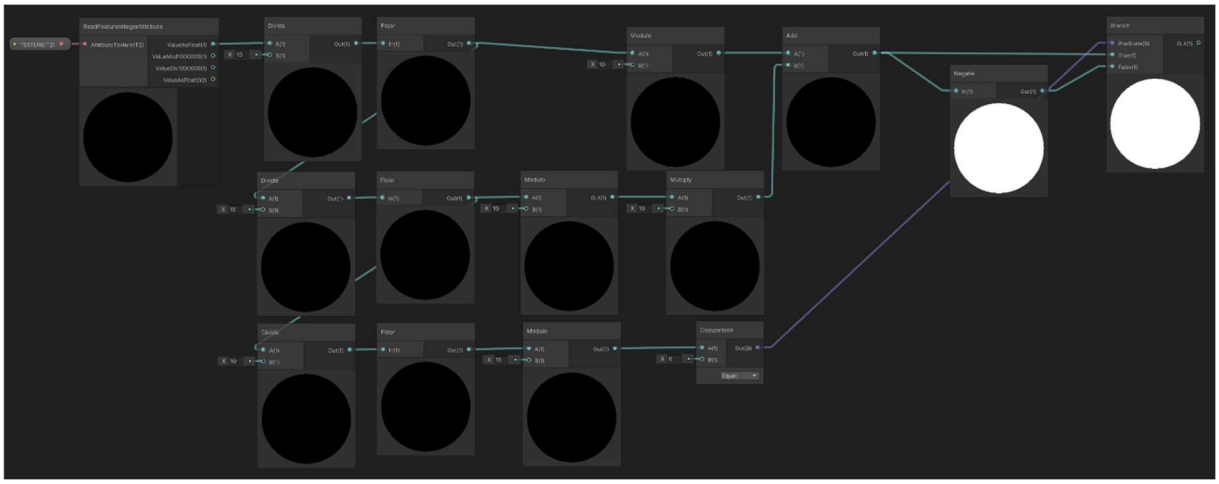


Abbildung 22: Übersicht über die Dekodierung des Rotationswertes in Shader Graph (Quelle: Eigene Darstellung)

Der Dekodierte Rotationswert wird an den Eingangsknoten der *Rotate* Funktion übergeben, der in der Abbildung 21 zu sehen ist. Mit der *Add* Funktion werden die einzelnen Ausgabewerte der Texturabfragen mit dem *BaseColor*-Knoten der Masterfunktion verbunden. Durch diese werden die Texturen am Ende auf den 3D-Modellen dargestellt.

In der Unity-Logik wird ein Shader einem Material zugeordnet. Dieses kann dann über ein 3D-Modell drapiert werden. Für die Zwecke dieser Arbeit reicht es aus, das Material *ConstructionYearRenderer* aus den Beispieldateien des ArcGIS Maps SDK for Unity zu kopieren und umzubenennen. Umbenannt wird es zu *TEXTURE* und als Shader wird der *TEXTURE*-Shader ausgewählt. Unter dem *MainCamera*-Objekt kann nun in der *Sample3DAttributesComponent*-Komponente der Listeneintrag *TEXTURE* gewählt werden. Damit ist die Konfiguration der Texturdarstellung abgeschlossen.

In der Abbildung 23 ist der derzeitige Stand der 3D-Szene zu sehen. Gut zu erkennen ist, dass die Dachziegel der Textur in die Hauptrichtung des Daches verlaufen. Ebenso haben die Flachdächer eine andere Textur und die senkrechten Wände im Hintergrund haben eine weiße Füllfarbe. Im Vordergrund ist zu erkennen, dass die Darstellung dieser Wände nicht immer richtig ist. Mehr dazu ist in einem späteren Teil dieser Arbeit zu finden.

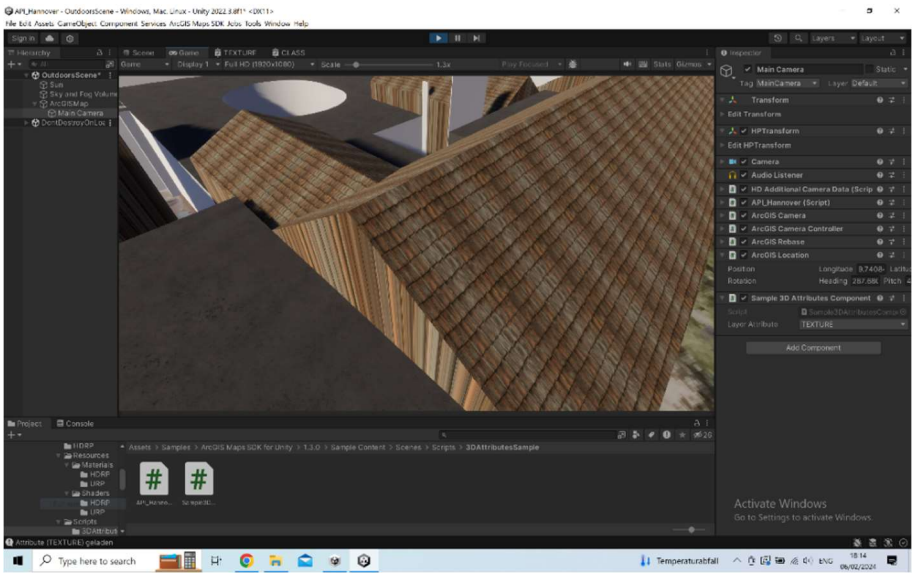


Abbildung 23: Bearbeitungsstand in der Unity Game-Engine nach der Anpassung des TEXTURE-Shaders (Quelle: Eigene Darstellung)

4.5.2 Konfiguration des CLASS-Shaders in Shader Graph

Die Konfiguration des *CLASS*-Shader ist eng mit dem des *TEXTURE*-Shader abgestimmt. Im Gegensatz zu der real gestalteten 3D-Szene werden für diesen Shader unbeleuchtete Materialien verwendet. Damit ergibt sich für eine Klasse nur ein Farbwert und es muss nicht zusätzlich auf Schattierungseffekte geachtet werden.

Die Dekodierung des Codes im Attributfeld *CLASS* entspricht jenem aus dem vorherigen Abschnitt. In der Abbildung 24 ist eine vereinfachte Darstellung des Shader zu sehen. Auch hier ist das vollständige Shader-Graph-Fenster im Anhang 2 zu finden.

Im unteren Teil des Shader wird zunächst durch den Modulo-Operator überprüft, ob es sich um eine senkrechte Wand handelt. Je nach Ausgabe der *Comparision* Funktion wird entweder der Input aus dem oberen Teil der Abbildung weitergegeben oder es wird eine magentafarbene Füllfarbe als Klassifizierung für eine senkrechte Wand in den Haupt *BaseColor*-Knoten eingefügt.

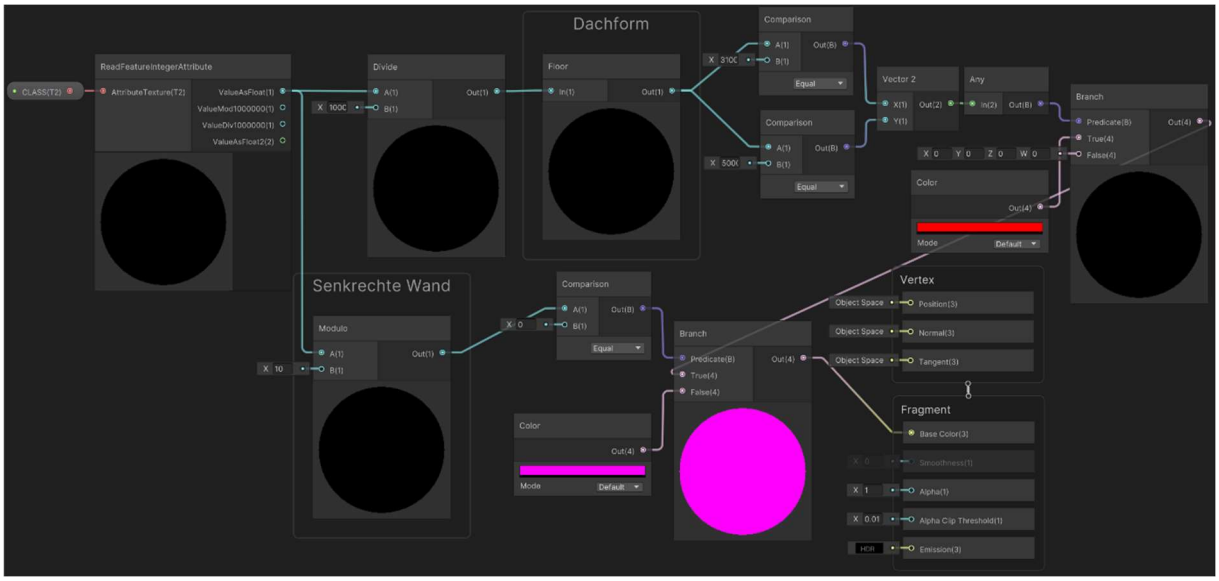


Abbildung 24: Übersicht über den CLASS-Shader in Shader Graph (Quelle: Eigene Darstellung)

Im oberen Teil der Abbildung 24 wird der Code mit der Form XXXXYYYZ durch 10.000 geteilt und abgerundet. Damit bleiben nur noch die vorderen vier Ziffern, welche die Codierung der Dachform enthalten. Analog der Zuweisung der Dachziegeltextur erfolgt auch hier ein Vergleich mit jedem Wert, der als Dachformcode auftreten kann. In diesem vereinfachten Beispiel wird einmal mit dem Wert 3100 und mit dem Wert 5000 verglichen. Da für beide Werte dieselbe Dachziegeltextur vergeben wird, soll auch in diesem Beispiel für die beiden Einträge dieselbe Farbe gezeigt werden, in diesem Fall ein rot.

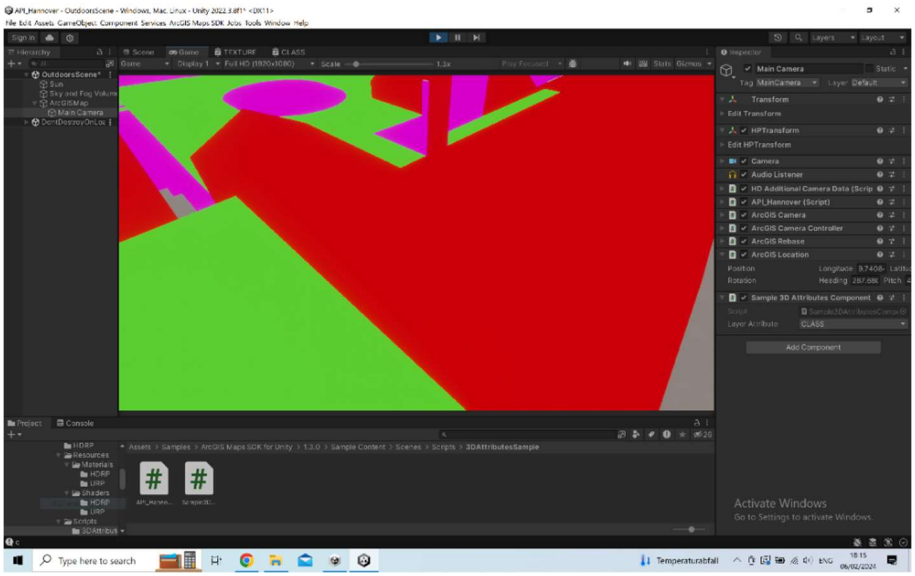


Abbildung 25: Bearbeitungsstand in der Unity Game-Engine nach der Anpassung des CLASS-Shaders (Quelle: Eigene Darstellung)

In der Abbildung 25 ist der Zwischenstand in der Unity Game-Engine zu sehen, wenn die Klassifizierungseinstellung gewählt ist. Speziell, wenn man diese Ansicht mit jener in der Abbildung 23 vergleicht, fällt auf, dass diese Szene keinen Schattenwurf aufweist.

5. Ergebnisse

In diesem Teil der Arbeit sollen die Ausgaben des Programms präsentiert werden. Als Ergebnis werden die erzeugten Bildpaare gesehen, die später genutzt werden können, um KNN zu trainieren.

Ein Bildpaar besteht hierbei zum einen aus dem Screenshot einer realistisch beleuchteten 3D-Szene mit DGM, DOP und texturierten Gebäudemodellen und zum anderen aus der Klassifizierungsmaske.

Wenn die Szene aktiviert ist, kann durch einen Klick der linken Maustaste ein Screenshot geschossen werden und anschließend kann auf die Klassifizierungsansicht gewechselt werden. Dann kann an derselben Stelle ein weiterer Screenshot erstellt werden, womit dann beide Teile des Bildpaares vorliegen.

In der Abbildung 26 ist das Opernhaus Hannover in der 3D-Szene zu sehen. In der Bildmitte ist zu erkennen, dass das Satteldach mit roten Dachziegeln texturiert ist. An den Seiten des Gebäudes sind LoD2-Modelle mit Flachdächern angeordnet. Zu den Seiten hin fallen die Flachdächer ab und werden zu Mansarden, die eine schwarze Dachziegeltextur haben. Im linken Teil des Bildes sind weiße Wände sowie auch im Innenhof zu erkennen. Das Elevationsmodell ist hier recht gleichmäßig und daher sind keine Vertiefungen im Erdboden zu erkennen, dafür ist das DOP, welches über das DGM drapiert ist, gut zu erkennen.

Abbildung 26: Teil 1 eines Beispielbildpaares (Quelle: Eigene Darstellung)

Anhand dieses Beispiels ist auch ein Problem mit der 3D-Szene erkennbar. Die Wände des Hauptgebäudes in der Mitte und die Außenwände des Nebengebäudes auf der rechten vorderen Seite weisen die jeweilige Dachtextur auf, obwohl diese eigentlich weiß sein sollten. Zusätzlich ist die Textur extrem verzerrt, da sie von oben auf das Gebäudemodell projiziert

wird. Es ergibt sich ein Streifeneffekt, indem der RGB-Wert des Pixels, der auf die Dachkante fällt, für die ganze Länge der Wand angewandt wird. Eine Überprüfung der Multipatch-Feature in ArcGIS Pro ergibt, dass die Wände den richtigen Neigungswert von minus eins aufweisen und das auch die Codierung im *TEXTURE*-Feld korrekt ist. Es ist davon auszugehen, dass der Fehler beim Import des Szenen Layer Paketes in die Unity Game-Engine durch das ArcGIS Maps SDK for Unity auftritt. Möglicherweise werden hierbei einzelne Multipatch-Feature kombiniert.

Diese Verzerrung tritt ebenfalls in der Klassifizierungsansicht auf. In der Abbildung 27 ist die Klassifizierungsmaske für den zuvor gezeigten Screenshot zu sehen. Die roten Teile stehen für die rote Dachziegeltextur, Grün für die Flachdachtextur, Blau für die Textur der schwarzen Dachziegel und Magenta für die Wände.

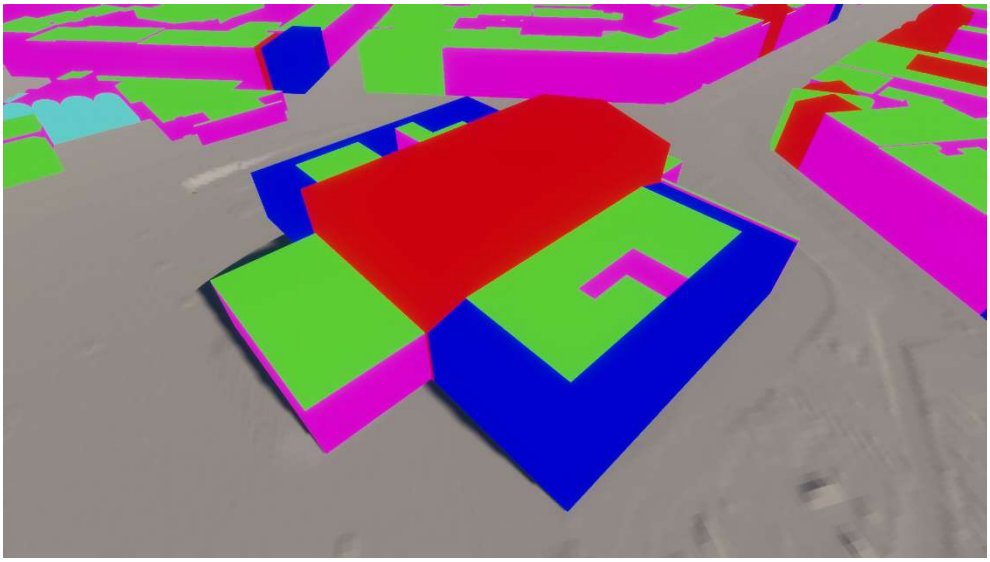


Abbildung 27: Teil 2 eines Beispielbildpaares (Quelle: Eigene Darstellung)

Auf dem Boden ist ohne die DOP-Ebene auch das DGM zu erkennen. Da hier nur Grautöne auftreten können, ist es nicht nötig, diesen gesondert zu klassifizieren. Solche Farben dürfen lediglich nicht als Kodierung für eine Klasse verwendet werden.

In dem kleinen Innenhof auf der vorderen rechten Gebäudeseite ist außerdem zu sehen, dass die verwendeten Materialien keinen Schattenwurf auf ihrer Oberfläche zulassen. Somit ist die Voraussetzung für eine Verwendung als KNN-Trainingsdaten gegeben, da jeder Klasse nur ein Farbwert zugeordnet wird.

6. Diskussion der Ergebnisse

In diesem Abschnitt der Arbeit sollen die präsentierten Ergebnisse diskutiert und näher erörtert werden. Es wird auf die Vorteile der synthetischen Produktionsmethode hingewiesen und weitere Problemstellungen aufgezeigt.

6.1 Vorteile der Herstellungsmethode

Grundsätzlich ist festzuhalten, dass die Ableitung einer 3D-Szene und der dazu gehörigen Klassifizierung aus bereits vorhandenen Geodaten öffentlicher Stellen funktioniert. Mit dem in dieser Arbeit beschriebenen Ansatz lassen sich Bildpaare als synthetische Trainingsdaten für ein KNN erzeugen. Gerade für eine Nischenanwendung ist diese Art der Erweiterung eines Trainingsdatensatzes interessant.

Als Basis dieser Arbeit dienen 3 Datensätze. Ein DGM, herausgegeben durch die Landeshauptstadt Hannover, Digitale Orthofotos, herausgegeben durch das LGLN und 3D-Gebäudemodelle der Qualitätsstufe LoD2, welche ebenfalls durch das LGLN herausgegeben werden. Die DOPs und die LoD2-Modelle sind flächendeckend für das gesamte Land Niedersachsen vorhanden. Lediglich das DGM ist auf den Raum Hannover beschränkt. Trotzdem ist es mit den verwendeten Datensätzen bereits möglich, Bildpaare an beliebigen Orten der Landeshauptstadt zu erstellen. Ein DGM des gesamten Landesgebietes mit einer Gitterweite von einem Meter liegt dem LGLN vor und könnte also auch als Basis für die Darstellung des gesamten Landesgebietes Niedersachsens dienen.

Die erzeugten Trainingsdaten bieten sich als Ergänzung für Anwendungen an, die Luftbilder des Landesgebietes Niedersachsen als Eingangsdaten haben. Hierdurch ist die Korrelation der Trainingsdaten und Eingangsdaten höher.

Die Unity Game-Engine ermöglicht es, sowohl die Beleuchtung als auch die Kamerawinkel und die Position der Kamera frei zu variieren. Es ist daher auch möglich, der Kamera eine Liste von Koordinaten und Kamerawinkeln zuzuweisen, die dann automatisch zuerst in der texturierten 3D-Szene und anschließend in der Klassifizierungsansicht aufgenommen werden können. Somit ist die Produktion großer Mengen an Bildpaaren ohne großen Zeitaufwand zu realisieren.

6.2 Herausforderungen und Nachteile

Im Gegensatz dazu steht die aufwändige Anpassung der Shader an die Gegebenheiten des jeweiligen 3D-Datensatzes. Die Texturierung und die Klassifizierung der einzelnen Gebäudeoberflächen werden durch die Attribute der Oberfläche vorgegeben. Daher ist es notwendig, für jeden möglichen Attributwert eine Textur und eine Klasse vorzugeben. Für die Szene in dieser Arbeit wurden fünf verschiedene Texturen gewählt, die zehn Dachformcodes

abdecken. Für eine vielfältigere Darstellung der 3D-Szene könnten noch mehr unterschiedliche Texturen eingebracht werden. Es ist auch denkbar für eine Dachform, mehr als eine Textur zuzuweisen, die dann durch das Zufallsprinzip ausgewählt werden. Der Dachformcode 3100 beispielsweise codiert ein Satteldach und tritt daher häufig auf. Es wäre also denkbar, diesem einen Code mehrere Texturen zuzuweisen, umso für mehr Vielfalt in der 3D-Szene zu sorgen. Für diesen Zweck bietet Unity umfangreiche Texturbibliotheken, die einfach in ein Projekt integriert werden könnten. Diese sind meist kostenpflichtig, beinhalten aber eine große Bandbreite verschiedener Materialien und die Dateiformate sind direkt mit Shader Graf kompatibel.

Eine Beschränkung dieser Herstellungsmethode für synthetische Trainingsdaten ist die Tatsache, dass ein zu texturierender und klassifizierender 3D-Datensatz sinnvoll attributiert sein muss. Der LoD2-Datensatz ist hierfür ein gutes Bespiel. Jedem Gebäudemodell ist eine Dachform zugeordnet. Wenn für das Training eines KNN, also beispielsweise Beispielbilder für die Unterscheidung von Flachdächern und Dächern mit Giebeln benötigt werden, dann können die Flachdächer und alle anderen Dächer mit unterschiedlichen Texturen versehen werden. Im Klassifizierungsschader können nun die Klassen Flachdach, nicht Flachdach und Hintergrund definiert werden und so können für diesen sehr spezifischen Anwendungsfall hilfreiche Trainingsdaten erzeugt werden. Wenn für ein KNN im Gegensatz dazu mehr Trainingsdaten benötigt werden, um Dachstühle auf Dächern in Luftbildern zu erkennen, so eignen sich die LoD2-Daten nicht für diesen Zweck. Zum einen sind die Daten nicht in dieser Hinsicht attributiert und zum anderen enthalten die Gebäudemodelle keine Modellierung ebendieser. Es folgt also, dass eine anwendungsbezogene Attributierung der 3D-Daten unerlässlich ist. Natürlich besteht auch die Möglichkeit, dies selbst vorzunehmen. Es muss dabei jedoch erneut abgewogen werden, ob der Aufwand zu rechtfertigen ist oder ob eine andere Herstellungsmethode vielleicht sinnvoller erscheint.

Eine weitere Problemstellung ist die Tatsache, dass die DOP schon einen natürlichen Schattenwurf beinhalten. Das Problem ist in der Abbildung 28 dargestellt. Auf der linken Seite ist die Belichtung der Szene anhand der natürlichen Schatten ausgerichtet. Gut zu erkennen ist der durch Unity modellierte Schattenwurf auf dem kleineren Gebäude. Dies ist der simulierte Schattenwurf auf die Texturen, der so gewollt ist, um die Realitätsnähe der 3D-Szene zu erhöhen.

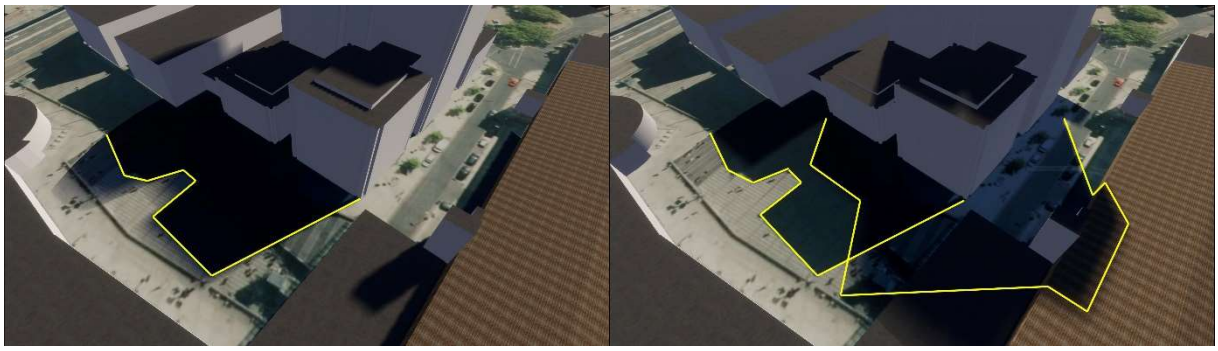


Abbildung 28: Illustration des Problems „Schattenwurf“ (Quelle: Eigene Darstellung)

Auf der rechten Seite der Abbildung 28 ist zu sehen was passiert, wenn die Beleuchtung nicht anhand der natürlichen Schatten ausgerichtet ist. Es entstehen um die Gebäude herum zusätzliche Schattenzonen, die die realitätsnähe verringern. Verhindern lässt sich dieses Problem derzeit nur, indem die Beleuchtung in der Unity Game-Engine immer anhand der natürlichen Schatten ausgerichtet wird.

Die generierte 3D-Szene beinhaltet keine Modellierung der Vegetation. Eine Dachfläche kann im Luftbild durch Bäume verdeckt werden und auch wenn ein solcher Grenzfall nicht häufig auftritt, so ist er mit der beschriebenen Methode nicht reproduzierbar. Gerade in ländlichen Gebieten könnte eine solche Erkennung wichtig sein, um Dachflächen richtig identifizieren zu können.

6.3 Stand der Entwicklung ArcGIS Maps SDK for Unity

Das ArcGIS Maps SDK for Unity ist im Juni 2022 erschienen und befindet sich noch in der Weiterentwicklung. Diese Arbeit wurde mit der Version 1.3.0 erstellt und im Dezember 2023 ist bereits die neue Version 1.4.0 erschienen. Es ist daher nicht verwunderlich, dass die Software nicht ausgereift ist und ein Problem ist die Tatsache, dass Multipatch-Feature-Oberflächen zusammengefasst werden. Diese Problematik scheint beim Import des *Szenen-Layer-Paketes* in die Unity Game-Engine durch das ArcGIS Maps SDK for Unity zu entstehen. In der Abbildung 29 ist wieder das Opernhaus Hannover dargestellt. Auf der linken Seite ist ein Screenshot aus Unity und auf der rechten Seite einer aus ArcGIS Pro zu sehen, welcher das *Szenen-Layer-Paket* beinhaltet. Es handelt sich um genau jenes, welches auch in die 3D-Szene geladen wird.

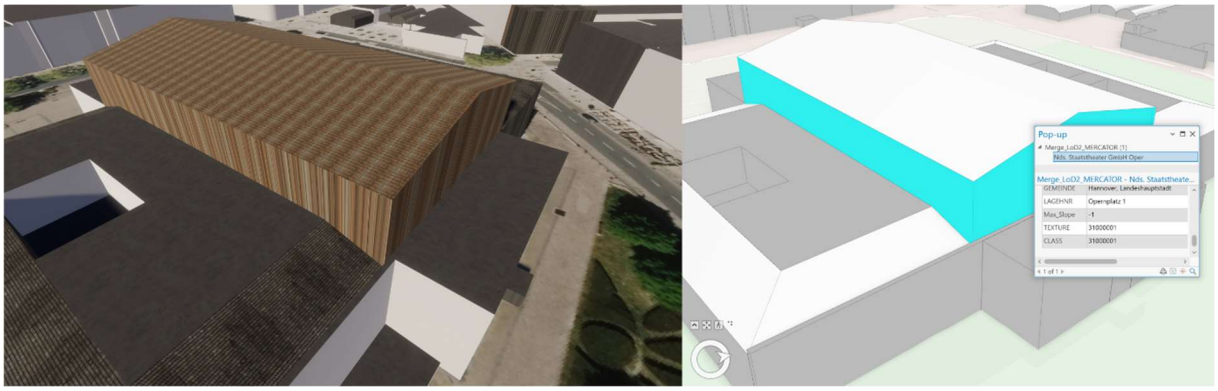


Abbildung 29: Illustration des Problems „Zusammengefügte Multipatch-Elemente“ (Quelle: Eigene Darstellung)

Die blau gefärbte Multipatch-Feature-Oberfläche ist die Selektion, von welcher in dem kleinen Fenster auch noch die Attributtabelle dargestellt ist. Es ist klar zu erkennen, dass die Dachfläche nicht zu diesem Objekt gehört. Außerdem ist in der Attributtabelle der TEXTURE-Wert zu sehen und dieser ist 31000001. Die letzte Ziffer ist eine eins und somit sollte der Texturierungsshader eine weiße Wand anzeigen. Auf der linken Abbildungsseite ist jedoch klar zu sehen, dass die verzehrte Dachziegeltextur dargestellt wird. Im unteren Teil der Abbildung ist eine richtige Darstellung zu sehen, in welcher die Wand weiß eingefärbt wird. Die falsche Zusammenfassung der Einzelobjekte passiert analog auch im Klassifizierungsshader. Es kommt also nicht zu fehl Klassifikationen.

Eine Funktion, die für die Zwecke dieser Arbeit von Interesse sein könnte, wäre die Fähigkeit des ArcGIS Maps SDK for Unity 2D-Ebenen und -Elemente mit Shadern manipulieren zu können. Derzeit ist dies ausschließlich mit *Szenen-Layer-Paketen* möglich. Diese beinhalten Multipatch-Feature-Objekte, die einen festen Höhenwert haben und sich daher nicht über das DGM drapieren lassen. Eine Höhe von 0m anzugeben ist hier keine Lösung, da die Objekte dann unter dem Elevationsmodell angezeigt werden. Wenn es möglich wäre manipulierbare Vektor-Ebenen einzufügen, dann könnte beispielsweise das Basis-DLM des LGLN eingefügt werden. Mit den Shadern der Unity Game-Engine könnten dann Trainingsdaten für die Klassifizierung von Landnutzungsdaten anhand der Klassen des Basis-DLM produziert werden. Dies könnte Trainingsdaten für die Erkennung von Landnutzungsdaten aus Luftbildern für das Landesgebiet Niedersachsen oder ähnliche Anwendungen liefern.

7. Fazit

In dieser Arbeit wird eine Möglichkeit untersucht, synthetische Trainingsdaten in einer Game-Engine für das Training eines KNN herzustellen. Dieses KNN soll die Automatische Auswertung von Luftbildern unterstützen und es sollten bereits vorhandene Geodaten der Katasterverwaltung Niedersachsen genutzt werden, um den Erfassungsaufwand zu minimieren.

Es wird gezeigt, dass diese Herangehensweise grundsätzlich funktioniert und es können brauchbare Bildpaare erstellt werden. Durch die Zusammenführung und Aufbereitung der drei verschiedenen Datensätze, zuerst durch ArcGIS Pro und dann in der Unity Game-Engine ist eine flexible 3D-Szene entstanden, die zusätzlich angepasst werden kann. Die Texturierung und Klassifizierung leitet sich aus den Attributierungen der Eingangsdaten ab.

Die in dieser Arbeit beschriebene Methode der Herstellung synthetischer Trainingsdaten kann dazu verwendet werden, Deep-Learning-Segmentierungsmodelle für die automatische Auswertung von Luftbildern zu verbessern. Sie eignet sich dazu große Mengen an Bildpaaren automatisiert herzustellen.

Im vorherigen Abschnitt wird auf die Limitierungen dieser Methode der Herstellung von synthetischen Trainingsdaten hingewiesen und es werden Möglichkeiten der Weiterentwicklung aufgezeigt.

Die in dieser Arbeit beschriebene Methode ermöglicht die Herstellung einer flexiblen 3D-Szene, die durch Beleuchtung und Partikeleffekte angepasst werden kann. In Kombination mit den Freiheiten der Kameraeinstellung können so Trainingsdaten für Grenzfälle erstellt werden, für die zu wenige oder keine anderwärtigen Datenquellen vorliegen.

Bedingt durch die zur Erstellung der 3D-Szene verwendeten Datensätze eigenen sich die generierten Bildpaare besonders, um die Gebäudeerkennung eines KNN im Gebiet des Landes Niedersachsens zu verfeinern.

Zusätzlich wurde das Potenzial der Weiterentwicklung dieses Produktionsansatzes hervorgehoben. Zu Nennen wäre hier vor allem eine mögliche Weiterentwicklung des ArcGIS Maps SDK for Unity, um 2D-Ebenen durch Shader manipulieren zu können. Vorstellbar wäre dann eine Szene in der Unity Game-Engine, die das Basis-DLM des LGLN und die DOPs enthält. Diese Szene könnte nun zur Erstellung von Trainingsdaten für KNN verwendet werden, die die Erkennung von Landbedeckung in Luftbildern unterstützen sollen.

Literaturverzeichnis

BISHOP, C.M. (2006): Pattern Recognition and Machine Learning. (Springer) New York.

DING, J., XUE, N., XIA, G.S., BAI, X., YANG, W., YANG, M., BELONGIE, S., LUO, J., DATCU, M., PELILLO, M., ZHANG, L. (2022): Object detection in aerial images: A large-scale benchmark and challenges. In: IEEE transactions on pattern analysis and machine intelligence, 44(11), S. 7778-7796.

FYLERIS, T., KRIŠČIŪNAS, A., GRUŽAUSKAS, V., ČALNERYTĖ, D., BARAUSKAS, R. (2022): Urban Change Detection from Aerial Images Using Convolutional Neural Networks and Transfer Learning. In: ISPRS International Journal of Geo-Information, 11(4): 246.

GARIOUD, A., PEILLET, S., BOOKJANS, E., GIORDANO, S., WATTRELOS B. (2022): FLAIR #1: semantic segmentation and domain adaptation dataset.
DOI: https://doi.org/10.13140/RG.2.2.30183.73128/1 (Abgerufen am 10.01.2024).

HENRY, C., FRAUNDORFER, F., VIG, E. (2021): Aerial Road Segmentation in the Presence of Topological Label Noise. In: IEEE (Hg.): 2020 25th International Conference on Pattern Recognition (ICPR) (Curran Associates) Mailand, S. 2336-2343.

HORN, D. & HOUBEN, S (2018): Evaluation of Synthetic Video Data in Machine Learning Approaches for Parking Space Classification. In: IEEE (Hg.): 2018 IEEE Intelligent Vehicles Symposium (IV 2018). (Curran Associates) Changshu, S. 2157-2162.

KATTENBORN, T., LEITLOFF, J., SCHIEFER, F., HINZ, S. (2021): Review on Convolutional Neural Networks (CNN) in vegetation remote sensing. In: ISPRS journal of photogrammetry and remote sensing 173, S. 24-49.

KUMARI, M. & KAUL, A. (2023): Deep learning techniques for remote sensing image scene classification: A comprehensive review, current challenges, and future directions. In: Concurrency Computation - Practice and Experience, 35(22) (John Wiley & Sons) Hoboken.

LAUX, L., SCHIRMER, S., SCHOPFERER, S., DAUER, J. (2022): Build Your Own Training Data - Synthetic Data for Object Detection in Aerial Images. In: Software Engineering 2022 Workshops. (Gesellschaft für Informatik e.V.) Bonn, S. 182-190.

LECUN, Y.; BENGIO, Y.; HINTON, G (2015): Deep Learning. In: Nature 521, S.436-444.

RAMEZAN, C.A.; WARNER, T.A.; MAXWELL, A.E.; PRICE, B.S. (2021): Effects of Training Set Size on Supervised Machine-Learning Land-Cover Classification of Large-Area High-Resolution Remotely Sensed Data. In: Remote Sensing, 13(3): 368.

VOELSEN, M., LOBO TORRES, D., QUEIROZ FEITOSA, R., ROTTENSTEINER, F., HEIPKE, C. (2021): Investigations on feature similarity and the impact of training data for land cover classification. In: ISPRS Annals of the Photogrammetry - Remote Sensing and Spatial Information Sciences 3, S. 181–189.

ZHANG, Z., LU, M., JI, S., YU, H., NIE, C. (2021): Rich CNN Features for Water-Body Segmentation from Very High Resolution Aerial and Satellite Imagery. In: Remote Sensing, 13(10): 1912.

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Alle Ausführungen, die anderen veröffentlichten oder nicht veröffentlichten Schriften wörtlich oder sinngemäß entnommen wurden, habe ich kenntlich gemacht.

Die Arbeit hat in gleicher oder ähnlicher Fassung noch keiner anderen Prüfungsbehörde vorgelegen.

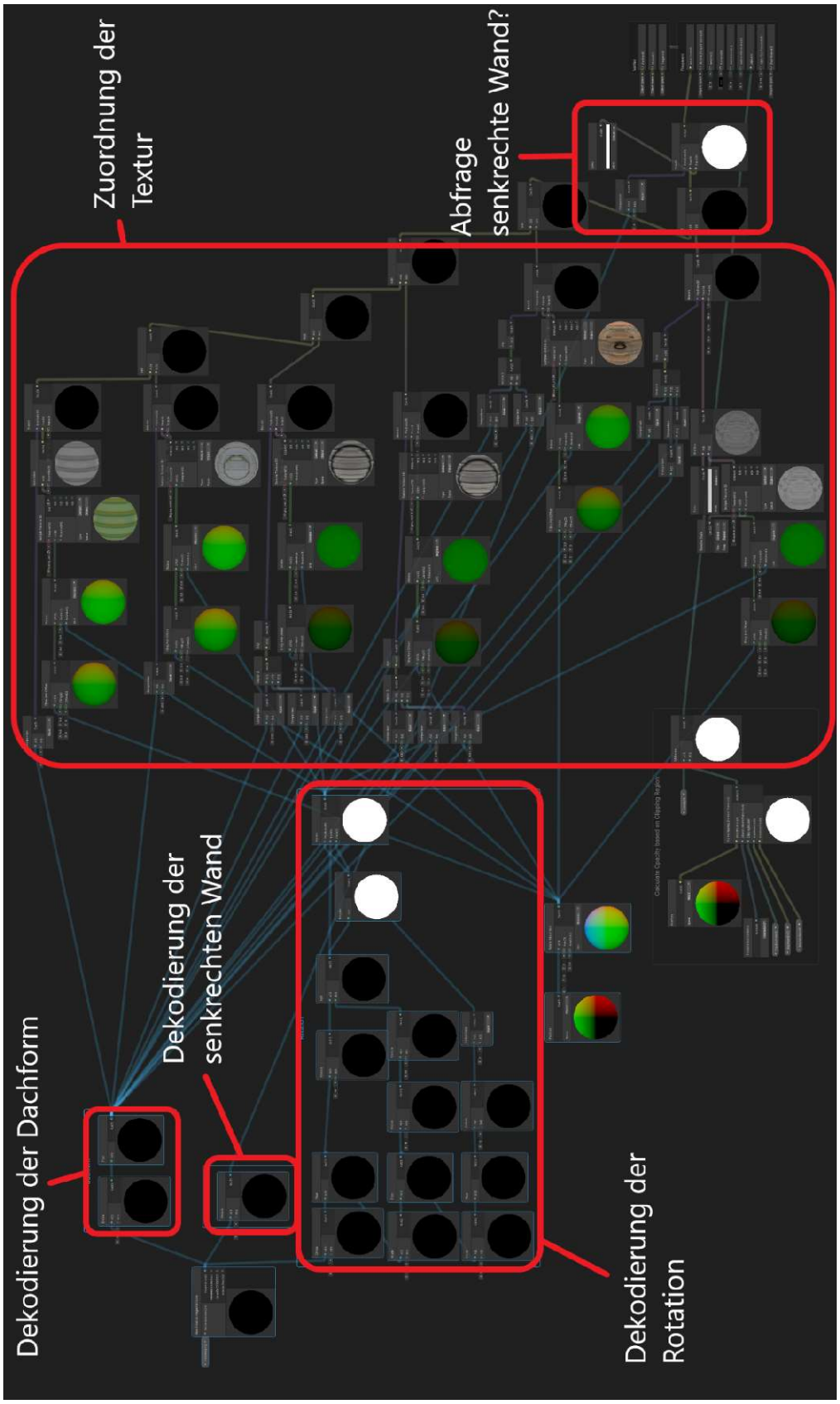
Northeim, 14.02.2024

Ort, Datum

Unterschrift

Anhang

Anhang 1: Übersicht über den TEXTURE-Shader



Anhang 2: Übersicht über den CLASS-Shader