universität wien

MASTERARBEIT | MASTER'S THESIS

Titel | Title

Effiziente Texturierung von 3D-Stadtmodellen durch die Integration prozeduraler Texturen in Multiresolutions-Texturatlanten

verfasst von | submitted by

Rainhard Neuhauser BSc

angestrebter akademischer Grad | in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien | Vienna, 2025

Studienkennzahl lt. Studienblatt | Degree programme code as it appears on the student record sheet:	UA 066 856
Studienrichtung lt. Studienblatt | Degree programme as it appears on the student record sheet:	Masterstudium Kartographie und Geoinformation
Betreut von | Supervisor:	Ass.-Prof. Mag. Dr. Andreas Riedl

Zusammenfassung

Hochaufgelöste, photorealistische Texturen für 3D-Stadtmodelle sind in der Erstellung wie im Rendering ressourcenintensiv: Sie beruhen häufig auf kostenaufwendiger Datenerhebung (Photogrammetrie, Laserscans) und verursachen hohe Speicher- und Rechenlast, insbesondere bei Nahansichten. Viele Anwendungen – etwa Simulationen, analytische Visualisierungen oder interaktive Stadtmodelle – benötigen jedoch keine streng naturgetreue Darstellung, sondern eine lesbare, optisch konsistente und performante Texturierung.

Diese Arbeit verfolgt daher das Ziel, einen Workflow zur effizienten Texturierung von 3D-Stadtmodellen zu entwickeln, der auf prozeduralen Fassadentexturen basiert und diese in Multiresolution-Texturatlanten organisiert. Prozedurale Module wurden in Substance Designer erstellt, in separaten Atlanten je Auflösungsstufe zusammengeführt und in einer Beispielszene in Blender über UV-Mapping auf ein Stadtmodell angewendet. Die Evaluation erfolgte durch einen visuellen Vergleich mit den Standardtexturen des Blender-OSM-Add-ons sowie durch Messung von Renderzeiten und Speicherbedarf. Ergänzend wurde eine Variante mit Parallax Occlusion Mapping (POM) betrachtet.

Die Ergebnisse zeigen ein klares Muster: In mittleren und weiten Ansichten liefern die Atlanten Effizienzgewinne bei mindestens vergleichbarer, häufig besserer visueller Darstellung. In Nahansichten steigen die Renderzeiten, werden jedoch durch deutlich höhere Detail- und Materialwirkung gerechtfertigt. POM reduziert geometrische Komplexität und erleichtert die Arbeit in großen Szenen.

Insgesamt bietet der Workflow eine flexible Möglichkeit, je nach Anwendung zwischen Effizienz und visueller Qualität zu gewichten. Weiterführende Arbeiten sollten die Parametrisierung der Texturmodule sowie die Automatisierung von UV-Unwrapping und Texturzuweisung vorantreiben.

Abstract

High-resolution, photorealistic textures for 3D city models are both costly to create and computationally demanding. They often rely on expensive data acquisition methods such as photogrammetry or laser scanning and can cause significant memory and performance issues, especially in close-up views. However, many applications—such as simulations, analytical visualizations, or interactive city models—do not require a fully photorealistic representation but instead demand a visually coherent and efficient texturing approach.

This thesis aims to develop a workflow for the efficient texturing of 3D city models by integrating procedural façade textures into multiresolution texture atlases. Procedural textures were created in Substance Designer, organized into separate atlases for each resolution level, and applied to a sample city scene in Blender using UV mapping. The evaluation combines a qualitative visual comparison with the standard textures of the Blender-OSM add-on and a quantitative performance analysis, including render times and memory usage. Additionally, a variant incorporating Parallax Occlusion Mapping (POM) was tested to reduce geometric complexity.

The results reveal a clear pattern: in medium and distant views, the multiresolution atlases offer significant performance improvements with comparable or superior visual quality. Close-up views lead to slightly increased render times but deliver much greater detail and realism, which justifies the trade-off. POM further enhances efficiency for large scenes.

Overall, the developed workflow offers a flexible solution that balances visual quality and performance. Future work may focus on extending parameterization and automating UV mapping and texture assignment.

Inhaltsverzeichnis

Abkürzungsverzeichnis

2D – Zweidimensional

2,5D – Zweieinhalbdimensional

3D – Dreidimensional

API – Application Programming Interface

AO – Ambient Occlusion

BIM – Building Information Modeling

bSI – buildingSMART International

CAD – Computer Aided Design

CGA – Computer Generated Architecture

DGM – Digitales Geländemodell

ESRI – Environmental Systems Research Institute

GIMP – GNU Image Manipulation Program

GIS – Geoinformationssystem

GML – Geography Markup Language

GPU – Graphics Processing Unit

HDRI – High Dynamic Range Image

IFC – Industry Foundation Classes

ISO – International Organization of Standardization

LOD – Level of Detail

LiDAR – Light Detection and Ranging

MIP – multum in parvo (viel in kleinem Raum)

NP – Komplexitätsklasse Nichtdeterministische Polynomialzeit

OGC – Open Geospatial Consortium

OSM – OpenStreetMap

PNG – Portable Network Graphics

PLM – Product Lfecycle Management

PBR – Physically Based Rendering

POM – Parallax Occlusion Mapping

RGB – Red Green Blue

SfM – Structure from Motion

SIFT – Scale-Invariant Feature transform

SIG 3D – Special Interest Group 3D

sRGB – standard RGB Farbraum

SSIM - Structural Similarity Index

UV – u/v-Texturkoordinaten

1. Einleitung

1.1. Motivation und Zielsetzung

Die Motivation für diese Arbeit ergibt sich aus der Herausforderung, die bei der Darstellung digitaler Stadtmodelle besteht. Während für Aufnahmen aus der Ferne niedrigauflösende Texturen meist ausreichen, sind für Nahaufnahmen höher aufgelöste Texturen mit mehr Detailreichtum notwendig, um sichtbare Pixel zu vermeiden und eine ansprechende Darstellung zu gewährleisten. Oftmals ist es schwierig eine angemessene Balance zwischen visueller Qualität und Leistungseffizienz zu finden. Insbesondere hochauflösende photorealistische Texturen mit hohem Detailgrad können zu langen Renderzeiten oder sogar zu einer Überlastung der Rechenkapazität führen.

Das Ziel dieser Arbeit ist es, einen Workflow zu entwickeln, der speziell für die Texturierung von 3D-Stadtmodellen geeignet ist. Im Mittelpunkt steht dabei die Entwicklung eines Multiresolution-Texturatlas aus prozeduralen Texturen. Es soll eine optimale Balance zwischen visueller Qualität und Leistungseffizienz in der Texturierung von 3D-Stadtmodellen gewährleistet werden, sowie eine Vielzahl kreativer Möglichkeiten für visuelle Gestaltung bieten.

1.2. Stand der Forschung

In den vergangenen Jahren ist die Nachfrage nach digitalen Stadtmodellen rasant gestiegen, was ihre wachsende Bedeutung und ihre vielfältigen Anwendungsmöglichkeiten unterstreicht. Digitale Stadtmodelle spielen heute eine entscheidende Rolle in Bereichen wie Stadtplanung, Immobilienentwicklung, Umweltsimulation, Navigationssystemen, Tourismus und vielen anderen (vgl. BUCHHOLZ 2006).

Die Wahl der geeigneten Darstellung von digitalen Stadtmodellen hängt in erster Linie vom geplanten Anwendungsbereich ab. So werden beispielsweise für die Stadtplanung und den Tourismus naturgetreue und photorealistische Texturen benötigt. Diese werden oft durch Methoden wie photogrammetrische Texturierung oder LiDAR-Texturierung erzeugt, bei denen Luftbildaufnahmen oder Laserscan-Daten verwendet werden, um eine genaue

Darstellung der städtischen Umgebung zu ermöglichen (vgl. FRUEH et al. 2004; KAHRAMAN & KARAS 2014).

Für Anwendungen wie Umweltsimulationen oder Navigationssysteme hingegen sind abstraktere Darstellungen erforderlich, die auf semantischen Unterscheidungen und vereinfachten Texturen basieren. In solchen Fällen werden häufig prozedurale Texturen verwendet (vgl. ALIZADEHASHRAFI et al. 2022).

Prozedurale Texturen werden mithilfe mathematischer Modelle generiert, sind besonders vielseitig und können eine breite Palette von Oberflächenstrukturen simulieren. Anstatt auf vorgefertigte Bilder oder Daten zurückzugreifen, werden prozedurale Texturen durch die Manipulation von Parametern oder die Verwendung von Zufallswerten generiert, oft in einer graphischen Benutzeroberfläche namens „Nodes“. Dieser Ansatz ermöglicht es neue Texturen zu erstellen, indem vorhandene Texturen weiterverarbeitet oder kombiniert werden (vgl. DONG et al. 2020).

Zusätzlich zu diesen Methoden zur Texturerzeugung spielt auch die Organisation der Texturen eine wichtige Rolle. Ein Beispiel dafür ist der Einsatz von sogenannten Textur-Atlanten. Bei dieser Technik werden mehrere Texturen in einem einzigen Bild (Atlas) organisiert. Dies ermöglicht es, die Anzahl der Aufrufe zum Laden der Texturen zu reduzieren und so die Effizienz des Renderings zu verbessern. Der Multiresolutions-Texturatlas ist eine Erweiterung dieses Konzepts, bei dem verschiedene Auflösungsstufen derselben Textur in einem einzigen Atlas kombiniert werden. Dadurch kann das Rendering je nach Detailanforderungen der Szene dynamisch zwischen verschiedenen Auflösungsstufen wechseln, was sowohl die Leistung als auch den Speicherbedarf optimiert (vgl. BUCHHOLZ 2006; ZHANG et al. 2021).

1.3. Problemstellung und Relevanz der Arbeit

Die meisten bestehenden Ansätze fokussieren sich auf naturgetreue, photorealistische Stadtmodelle, deren Erstellung jedoch mit hohem Aufwand und erheblichen Kosten verbunden ist. Insbesondere basieren solche Modelle meist auf Photogrammetrie oder Laserscanning-Daten, für die teure Befliegungen oder umfangreiche Erhebungen durchgeführt werden müssen. Die Kosten können dabei schnell mehrere zehntausend Euro

für kleinere Areale oder bis in den sechsstelligen Bereich für größere städtische Gebiete betragen (vgl. BUCHHOLZ 2006; FRUEH et al. 2004).

Neben diesen kostenintensiven Verfahren gibt es auch Ansätze, wie etwa von Zhang et al. (2021), die auf bereits texturierten Stadtmodellen aufbauen und deren Texturen vor allem im Hinblick auf Performance und Automatisierung optimieren. Die vorliegende Arbeit setzt hingegen einen Schritt davor an und beginnt mit untexturierten Blockmodellen. Anstatt vorhandene Texturen lediglich effizienter zu organisieren, liegt der Fokus hier auf der prozeduralen Erzeugung realistischer Fassadentexturen sowie deren Organisation in Multiresolution-Texturatlanten.

Daraus ergibt sich folgende zentrale Forschungsfrage:

„Wie effizient ist die Texturierung von 3D-Stadtmodellen durch die Integration prozeduraler Texturen in Multiresolution-Texturatlanten, insbesondere in Bezug auf Leistungseffizienz, Skalierbarkeit und visuelle Qualität?“

1.3. Methodischer Ansatz

Im Mittelpunkt dieser Arbeit steht die Erstellung prozeduraler Fassadentexturen für 3D-Stadtmodelle sowie deren Organisation in einem Multiresolution-Texturatlas. Als Werkzeug zur Texturerstellung wird Substance Designer eingesetzt, da es durch seine Node-basierte Arbeitsweise eine flexible und parametergesteuerte Generierung komplexer Texturen ermöglicht.

Die erstellten Texturen werden anschließend in mehrere Texturatlanten zusammengeführt, wobei für jede Auflösungsstufe und Texturart ein eigener Atlas erstellt wird. Zur Evaluierung wird eine Beispielszene in Blender aufgebaut, in die die Texturatlanten integriert werden. Die Texturen werden dabei mittels UV-Mapping auf ein 3D-Stadtmodell projiziert und gerendert.

Die Bewertung der Ergebnisse erfolgt sowohl qualitativ als auch quantitativ. Zum einen werden die Renderergebnisse subjektiv hinsichtlich der optischen Qualität mit den Standardtexturen des Blender-OSM-Add-ons verglichen. Zum anderen werden im Rahmen einer Performance-Analyse die Renderzeiten sowie der Speicherbedarf während des Renderings gemessen und den Werten herkömmlicher Texturen gegenübergestellt.

1.4. Aufbau der Arbeit

Die Arbeit gliedert sich in einen theoretischen und einen praktischen Teil.

Zunächst werden die theoretischen Grundlagen vermittelt. Dabei werden die zentralen Aspekte von 3D-Stadtmodellen behandelt, beginnend mit ihrer Definition, den gängigen Standards und Formaten, die Level of Detail sowie den unterschiedlichen Methoden zur Erstellung solcher Modelle. Anschließend folgt eine Einführung in die Grundlagen der Texturierung, die von technischen Grundlagen über verschiedene Texturtypen – bildbasiert und prozedural – bis hin zu Methoden des Texture-Mappings reicht. Darüber hinaus werden Techniken zur Optimierung des Texturmanagements wie Texturatlanten, Multiresolutions-Texturatlanten und Mipmapping erläutert.

Im praktischen Teil der Arbeit wird der entwickelte Workflow zur Integration prozeduraler Texturen beschrieben. Schritt für Schritt werden die durchgeführten Arbeitsschritte dargestellt: vom Sammeln von Referenzbildern über die Erstellung der prozeduralen Texturen in Substance Designer, die Zusammenführung der Texturen zu Atlanten in GIMP, bis hin zur Integration dieser Atlanten in eine Blender-Szene. Abschließend werden die Ergebnisse evaluiert, bevor ein Ausblick auf mögliche Weiterentwicklungen gegeben wird.

2. Grundlagen und theoretischer Rahmen

2.1. Einführung in 3D-Stadtmodelle

Der Einsatz von digitalen 3D-Stadtmodellen hat in den letzten Jahren stark zugenommen. In Bereichen wie etwa der Stadtplanung, der Umweltanalyse, bei Mobilitätsfragen oder auch im Katastrophenschutz können 3D-Stadtmodelle bei kritischen Entscheidungsprozessen helfen, da sie mehr Informationen enthalten als klassische zweidimensionale Karten (siehe Abbildung 1) (vgl. BILJECKI et al. 2015: 2842f.; LEI et al. 2023: 788f.; FORKERT o.J.).

Abbildung 1: Beispiel - 3D-Stadtmodell
Quelle: virtualcitysystems GmbH, unter: https://vc.systems/ letzter Zugriff am 13.05.2025

Mit der steigenden Nachfrage haben sich auch die Methoden zur Datenerfassung und Modellierung weiterentwickelt. Vor allem im Bereich der automatisierten Gebäudemodellierung wurden erhebliche Fortschritte erreicht. Moderne Verfahren ermöglichen heute Kosteneinsparungen von bis zu 70 % im Vergleich zu früheren Jahren (vgl. FORKERT o.J.), unter anderem durch die effizientere Nutzung bereits vorhandener Geodaten. Trotzdem ist die Erhebung der Daten weiterhin ein wesentlicher Kostenfaktor (vgl. BILJECKI et al. 2015: 2842f.; LEI et al. 2023: 788f.).

Trotz ihrer weiten Verbreitung gibt es weiterhin Herausforderungen: Viele 3D-Stadtmodelle werden unabhängig voneinander und mit unterschiedlichen Zielen entwickelt. Sie unterscheiden sich stark in Struktur und Inhalt, was Vergleiche erschwert. Die Standardisierung von 3D-Stadtmodellen ist bereits ein wichtiger Schritt, um dieses Problem zu lösen (vgl. BILJECKI et al. 2015: 2842f.; LEI et al. 2023: 788f.; FORKERT o.J.).

Das folgende Kapitel gibt eine grundlegende Einführung in das Thema 3D-Stadtmodelle. Es erklärt zentrale Begriffe, stellt die wichtigsten Formate und Standards vor und zeigt typische Datenquellen und Erfassungsmethoden auf.

2.1.1. Definition und Abgrenzung

2.1.1.1. Begriffserklärung – Was ist ein 3D-Stadtmodell?

„Ein 3D-Stadtmodell ist eine Darstellung einer urbanen Umgebung mit einer dreidimensionalen Geometrie von typischen städtischen Objekten und Strukturen, wobei Gebäude das hervorstechendste Merkmal sind.“ (BILJECKI et al. 2015: 2843) Es handelt sich dabei also um eine digitale, georeferenzierte Repräsentation urbaner Räume, die neben Gebäuden auch Geländeformen, Verkehrsinfrastruktur, Vegetation und weitere städtische Objekte umfasst. Mit diesen Modellen können komplexe urbane Strukturen abgebildet und analysiert werden (vgl. BILJECKI et al. 2015: 2843f.; DÖLLNER et al. 2006: 107).

Ein typisches Merkmal von 3D-Stadtmodellen ist, dass sie verschiedene Geodaten in einer gemeinsamen, visuellen und strukturierten Umgebung kombinieren. Dadurch ermöglichen sie nicht nur eine realistische Visualisierung der Stadt, sondern auch eine umfassende Analyse von räumlichen und thematischen Informationen, zum Beispiel im Bereich Stadtplanung, Katastrophenschutz oder Immobilienbewertung (vgl. ebd.).

Die Geometrie stellt dabei die Form der Objekte dar, während Texturen für ein realistischeres Aussehen sorgen. Viele 3D-Stadtmodelle enthalten außerdem zusätzlich semantische Informationen, die nicht direkt für die visuelle Darstellung wichtig sind, sondern zusätzliche Beschreibungen liefern, wie beispielsweise zur Nutzung eines Gebäudes oder seiner Eigentümerstruktur. Diese semantische Ebene wird immer wichtiger, besonders im Zusammenhang mit thematischen Analysen und Smart Cities (vgl. DÖLLNER et al. 2006: 107; UGGLA 2015: 3f.).

3D-Stadtmodelle haben sich von ihrer anfänglichen visuellen Nutzung, etwa in Form von Architekturvisualisierung oder digitalen Stadtansichten, zu einer Vielzahl technischer und analytischer Anwendungen entwickelt. Heute werden sie nicht nur zur Darstellung verwendet, sondern dienen auch als Grundlage für simulationsgestützte Entscheidungsprozesse (siehe Abbildung 2). Damit zeigen sie ihr Potential als digitale Werkzeuge zur Erfassung, Analyse und Vermittlung urbaner Informationen (vgl. BILJECKI et al. 2015: 2843f.).

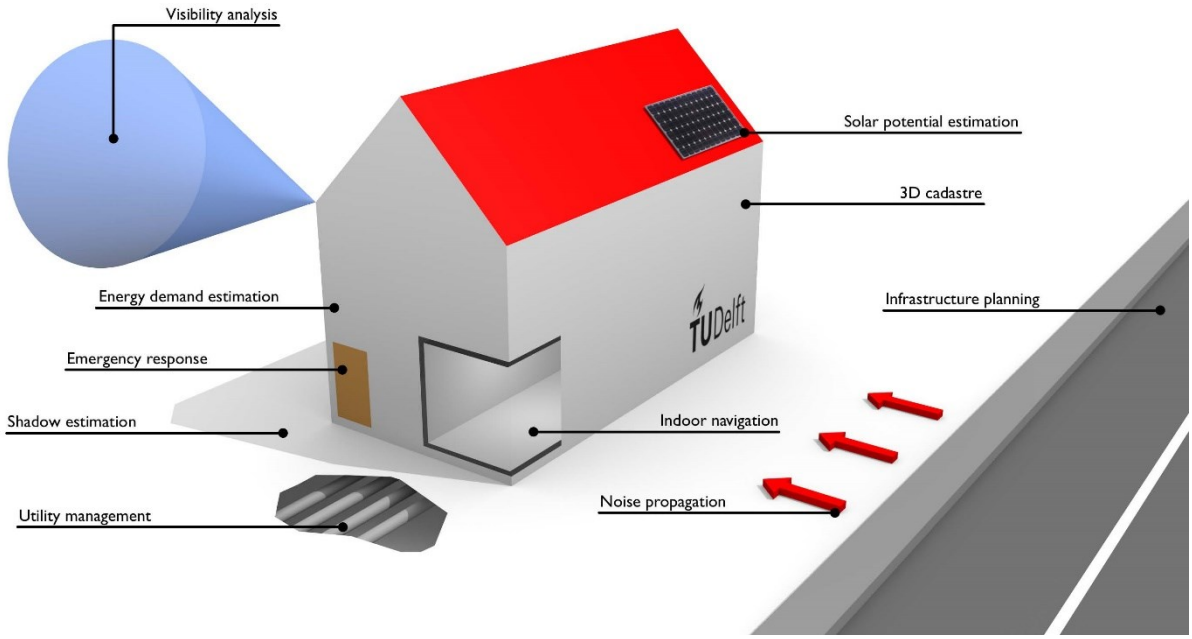

Abbildung 2: Einsatzmöglichkeiten von 3D-Stadtmodellen
Quelle: BILJECKI et al. 2015

2.1.1.2. Modellansätze und verwandte Konzepte

Geometrische und Semantische Modelle

3D-Stadtmodelle basieren grundsätzlich auf zwei Modelltypen, welche sich gegenseitig ergänzen können, nämlich geometrische und semantische Modelle. Geometrische Modelle (siehe Abbildung 3) beschreiben die Form, Position und Ausdehnung städtischer Objekte, wie beispielsweise Gebäude, Straßen oder Vegetation, meist in Form von Punkten, Linien oder Flächen. Diese Modelle stützen sich häufig auf standardisierte Schemata wie dem sogenannten ISO 19107 „Spatial Schema“, welche grundlegende Konzepte zur Modellierung räumlicher Objekte sowie deren geometrische und topologische Beziehungen definiert (vgl. HERRING 2001; EL-MEKAWY 2010: 13f.).

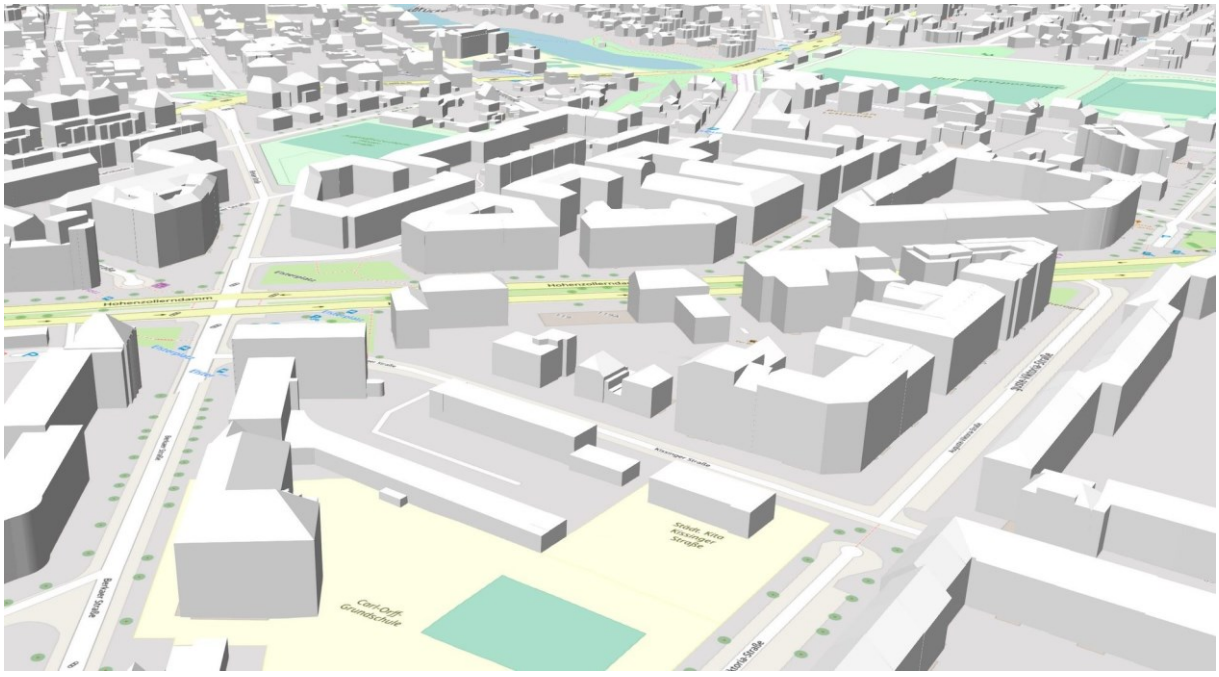

Abbildung 3: Beispiel – Geometrisches Stadtmodell (LoD-2)
Quelle: virtualcitysystems GmbH, unter: https://vc.systems/ letzter Zugriff am 13.05.2025

Semantische Modelle hingegen bringen eine zusätzliche Bedeutungsebene mit sich. Sie weisen den Objekten Eigenschaften, Kategorien oder funktionale Zusammenhänge zu. Beispiele dafür sind etwa die Gebäudeart, die Nutzungsform oder räumlich-funktionale Beziehungen zu anderen Objekten im Stadtraum (siehe Abbildung 4) (vgl. EL-MEKAWY 2010: 13f.).

In der Praxis stammen geometrische Daten meist aus CAD-Zeichnungen, Laserscans, Vermessungen oder photogrammetrischen Verfahren. Semantische Informationen werden dagegen aus Planunterlagen extrahiert oder durch andere Erhebungsmethoden ergänzt. Die

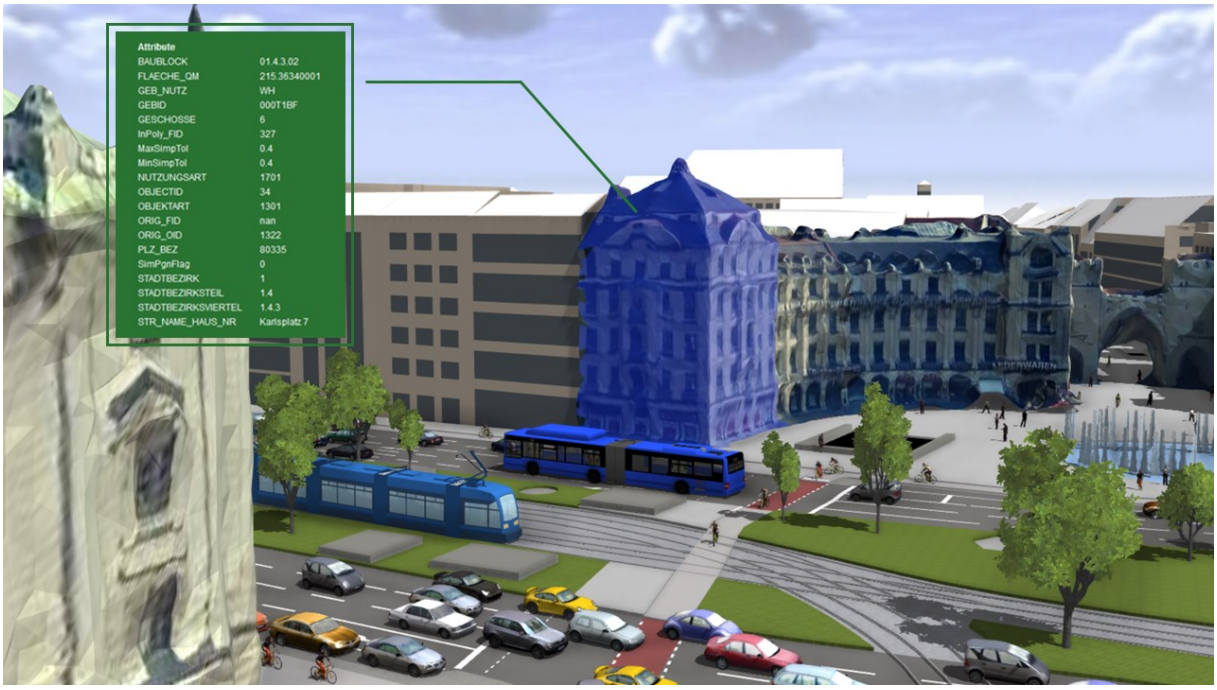

Abbildung 4: Beispiel – Semantisches Stadtmodell
Quelle: München. Digital., unter: https://muenchen.digital/ letzter Zugriff am 13.05.2025

konkreten Methoden zur Erstellung von 3D-Stadtmodellen werden in einem späteren Kapitel ausführlicher behandelt (vgl. SUN et al. 2020: 235f.; EL-MEKAWY 2010: 13f.).

Lange Zeit lag der Fokus in der digitalen Modellierung, insbesondere in Architektur und Bauwesen, vor allem auf der Geometrie. Klassische CAD-Modelle in 2D oder 3D dienten primär der Formdarstellung und enthielten in der Regel keine inhaltlichen Zusatzinformationen. Mit dem Aufkommen von Konzepten wie Product Lifecycle Management (PLM) und Building Information Modeling (BIM) rückten semantische Aspekte stärker in den Vordergrund (vgl. EL-MEKAWY 2010: 13f.).

Im Gegensatz dazu konzentrieren sich geodatenbasierte Stadtmodelle auf die Abbildung urbaner Räume in ihrer Gesamtheit. Sie erfassen reale Objekte der Stadtumgebung wie beispielsweise Gebäude, Infrastruktur oder Geländeformen und bilden diese in vereinfachter geometrischer Form ab, zum Beispiel 3D oder 2,5D, einer reduzierten Darstellung der vertikalen Dimension (vgl. ebd.).

Ein Beispiel für ein Datenformat, welches beide Modellarten integriert, ist CityGML. Es ermöglicht eine standardisierte Repräsentation urbaner Objekte die sowohl geometrische als auch semantische Eigenschaften umfasst. CityGML wird in einem späteren Kapitel ausführlicher behandelt (vgl. SUN et al. 2020: 235ff.).

Geoinformationssysteme (GIS)

Geographische Informationssysteme (GIS) sind computergestützte Informationssysteme zur Erfassung, Verwaltung, Analyse und Präsentation raumbezogener Daten. Im Zentrum steht dabei die Verarbeitung georeferenzierter Informationen. GIS-Systeme bestehen typischerweise aus einer Kombination von Hardware, Software, Methoden, den vorliegenden Daten sowie dem Anwender, auch als Brainware bezeichnet (siehe Abbildung 5) (vgl. EL-MEKAWY 2010: 15f.; MIERZEJOWSKA/POMYKOL 2019: 1f.).

Abbildung 5: GIS-Komponenten
Quelle: MIERZEJOWSKA und POMYKO 2019

Mit dem technologischen Fortschritt und der zunehmenden Verfügbarkeit hochauflösender Geodaten hat sich GIS auch im Bereich der 3D-Modellierung von Städten etabliert. So lassen sich beispielsweise mit der „CityEngine“ von ArcGIS Pro vollwertige 3D-Stadtmodelle erzeugen (ESRI Inc. 2025). Diese erfassen nicht nur die Lage und Geometrie, sondern integrieren auch semantische Informationen (vgl. EL-MEKAWY 2010: 15f.; KHAYYAL et al. 2022: 105f.).

Computer-Aided Design (CAD)

Bei Computer-Aided Design (CAD) handelt es sich um die rechnergestützte Erstellung technischer Modelle, die vor allem in Bereichen wie Architektur, Bauwesen und Maschinenbau Verwendung finden. Im Fokus stehen hierbei Form, Struktur und technische Details im Rahmen konkreter Konstruktionsprozesse. CAD-Modelle enthalten daher in der Regel keine umfassenden Kontextinformationen zum städtischen Raum (vgl. NAVRATIL et al. 2020: 129).

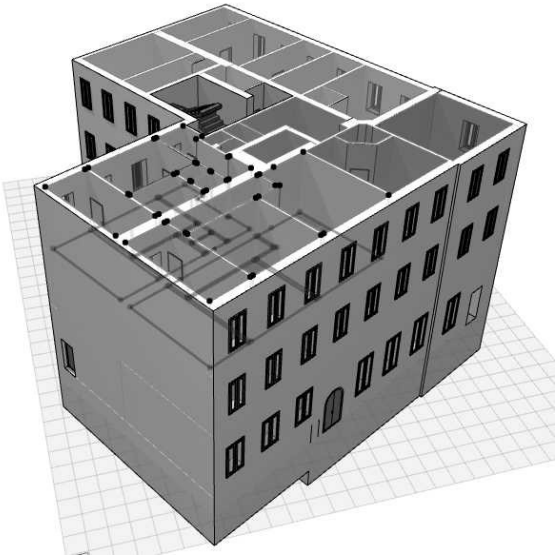

Abbildung 6: CAD-Visualisierung
Quelle: NAVRATIL et al. 2020

Building Information Modeling (BIM)

Building Information Modeling (BIM) beschreibt einen Arbeitsvorgang, in dem sämtliche Aspekte eines Gebäudezyklus zentral erfasst und verwaltet werden (siehe Abbildung 7). Dazu zählen unter anderem die Planung, der Bau und die Instandhaltung. Dabei werden Gebäudeelemente wie Wände, Fenster oder Stützen als intelligente 3D-Objekte modelliert, die neben den geometrischen Informationen auch semantische Eigenschaften, wie Materialien und technische Parameter enthalten (vgl. EL-MEKAWY 2010: 14f.; ESKEF 2020: 12; NAVRATIL et al. 2020: 129f.; SUN et al. 2020: 235ff.).

Abbildung 7: Eigenschaften des BIM Systems
Quelle: ESKEF 2020

Im Unterschied zu 3D-Stadtmodellen, die den urbanen Raum ganzheitlich abbilden und dabei auch Infrastruktur und Vegetation berücksichtigen, konzentriert sich BIM in der Regel auf einzelne Gebäude und deren technische Aspekte. BIM hat sich dabei insbesondere in der Architektur- und Bauindustrie als zentraler Standard etabliert.

2.1.2. Standards und Formate

Aufgrund der Vielzahl an Einsatzmöglichkeiten von 3D-Stadtmodellen existieren auch viele Standards und Datenformate, da ein einziges nicht für alle Anwendungsbereiche angewendet werden kann. Ein Datenformat beschreibt dabei die technische Struktur, also wie die Daten innerhalb einer Datei abgespeichert werden. Standards hingegen legen fest, welche dieser Formate offiziell anerkannt bzw. empfohlen werden, um das Austauschen von Daten effizienter zu machen. Diese beiden Begriffe werden häufig synonym verwendet (vgl. KAUFMANN 2021: 12). Im folgenden Kapitel werden die gängigsten davon näher erläutert, um einen Überblick zu verschaffen.

2.1.2.1. Open Geospatial Consortium (OGC)

Die OGC ist eine Internationale Organisation tätig im Bereich Geoinformation, um einheitliche Standards und Austauschformate zu entwickeln und die geodatenbezogene Interoperabilität zu fördern. Sie vereint über 500 Mitgliedern, bestehend aus verschiedensten Regierungsorganisationen, Forschungseinrichtungen sowie einzelnen Unternehmen aus der Privatwirtschaft (vgl. UGGLA 2015: 10f.).

2.1.2.2. Industry Foundation Classes (IFC)

IFC steht für Industry Foundation Classes und ist ein offener Standard im Bauwesen, welcher von buildingSMART International (bSI) definiert wird. Die IFC ist das bedeutendste Austauschformat für BIM, mit dem Ziel eine einheitliche Sprache für Bauprojekte zur besseren Zusammenarbeit zu schaffen. Neben den materiellen Aspekten eines Gebäudes, wie etwa Fenster, Türen, Wände, werden auch semantische Informationen wie zum Beispiel Kosten, Zeitpläne oder Energieverbrauch eingebunden (vgl. ESKEF 2020: 17f.; EL-MEKAWY 2010: 24ff.; PLUME/MITCHELL 2005: 35f.).

2.1.2.3. CityGML

CityGML ist sowohl ein Datenformat als auch ein internationaler Standard zur Darstellung und zum Austausch von digitalen 3D-Stadtmodellen. Die Abkürzung GML (Geography Markup Language) bezeichnet dabei eine Programmiersprache die auf der sogenannten XML-Grammatik (Extensible Markup Language) basiert und ermöglicht die Modellierung, Speicherung und Übermittlung von geographischen Informationen. Die CityGML gehört zum OGC-Standard und soll einen reibungslosen Austausch von Stadtmodell-Daten in verschiedenen Anwendungen ermöglichen (vgl. EL-MEKAWY 2010: 17ff.; KOLBE 2009: 17f.).

Modellierte Aspekte

Die Grundstruktur des CityGML besteht aus vier verschiedenen Aspekten. Der Semantik, der Geometrie, der Topologie und dem Erscheinungsbild. Diese vier Modellierungsaspekte sind nicht isoliert, sondern in den Aufbau integriert und werden in vertikale und horizontale Module unterteilt. Diese Trennung ermöglicht es themenspezifisch oder übergreifend zu modellieren. Abbildung 8 zeigt eben diesen modularen Aufbau mit dem technischen Fundament GML. Direkt darüber liegt das zentrale Erweiterungsmodul CityGML Core, welches die Schnittstelle zwischen GML und den weiteren Modulen bildet. Die Vertikalen Module sind thematische Objektlassen welche sowohl semantische Informationen als auch geometrische und topologische Strukturen beinhalten. Darüber im Appearance - Modul werden visuelle Aspekte wie Texturen, Farben oder Materialien definiert und an der Spitze im Generics – Modul können benutzerdefinierte Attribute und Objekte hinzugefügt werden (vgl. KOLBE 2009: 17f.).

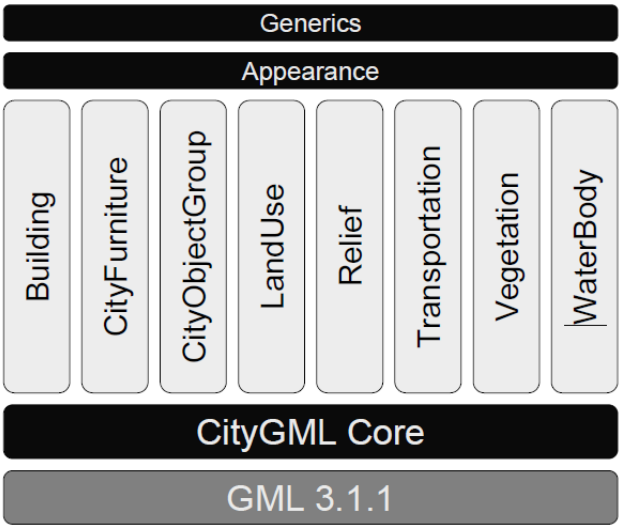

Abbildung 8: Modularisierung von CityGML
Quelle: KOLBE 2009

2.1.3. Klassifikation nach Level of Detail (LOD)

Mit den sogenannten Levels of Detail (LOD) können in CityGML verschiedene Detaillierungsgrade von 3D-Modellen dargestellt werden. Dieses LoD-Konzept wurde ursprünglich vom SIG 3D, einer Arbeitsgruppe der Initiative Geodateninfrastruktur Deutschland, für CityGML entwickelt. Es gibt aber auch außerhalb von CityGML diverse Ansätze, LOD-Konzepte für Gebäude zu definieren, wie beispielsweise die Arbeiten von BILJECKI 2017. CityGML unterscheidet fünf verschiedene Level of Detail (siehe Abbildung 9) – LOD0, LOD1, LOD2, LOD3 und LOD4 (vgl. BILJECKI 2017: 3ff.; KAUFMANN 2021: 17f.; KOLBE 2009: 18ff.).

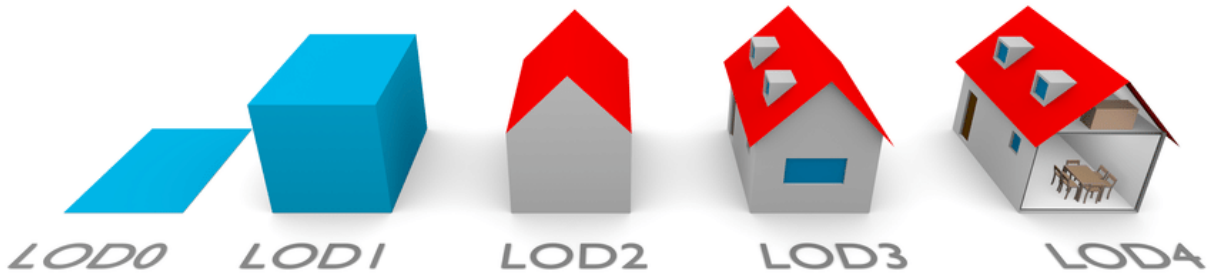

Abbildung 9: Levels of Detail in CityGML
Quelle: BILJECKI et al. 2016

2.1.3.1. LOD0 – Geländemodell

Das LOD0 stellt die einfachste und grobste Detailstufe in CityGML dar (siehe Abbildung 10). Es besteht hauptsächlich aus einem digitalen Geländemodel (DGM), also einer Höhenbeschreibung der Erdoberfläche. Die Gebäude werden hierbei als 2D-Grundrisse dargestellt und auf das Gelände projiziert. Häufig wird über das Geländemodell ein Luftbild gelegt, um das Gesamtbild anschaulicher zu machen. Das LOD0 eignet sich besonders gut für großflächige Übersichten, bei denen einzelne Gebäude und deren Form eine weniger wichtige Rolle spielen (vgl. KAUFMANN 2021: 17; KOLBE 2009: 18f.).

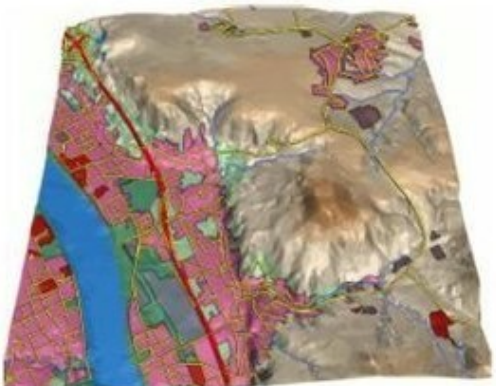

Abbildung 10: LOD0 – Geländemodell
Quelle: Open Geospatial Consortium (OGC), unter: https://www.ogc.org/standards/citygml/ letzter Zugriff am 13.05.2025

2.1.3.2. LOD1 – Blockmodell

Beim LOD1 werden die einzelnen Gebäude bereits in 3D-Form dargestellt. Aus den flachen Grundrissen wird durch die Extrusion der Kanten das sogenannte Blockmodell erstellt, die die ungefähre Höhe eines Gebäudes abbilden (siehe Abbildung 11). Die Dachformen oder Details in der Fassade werden hierbei noch zur Gänze. Stadtobjekte, wie etwa Bäume oder Infrastrukturobjekte können vereinzelt und in simpler geometrischer Form dargestellt werden. Für viele Anwendungen, bei denen es eher um die räumliche Ausdehnung von Gebäuden geht, reicht das LOD1 bereits aus (vgl. KAUFMANN 2021: 18).

Abbildung 11: LOD1 – Blockmodell
Quelle: Open Geospatial Consortium (OGC), unter: https://www.ogc.org/standards/citygml/ letzter Zugriff am 13.05.2025

2.1.3.3. LOD2 – Dachmodell

Im Level of Detail 2 wird das Stadtmodell um erste architektonische Merkmale erweitert. Im Fokus stehen dabei die generalisierten Dachformen, die sich nun klar in Höhe und Geometrie vom LOD1 unterscheiden (siehe Abbildung 12). Zusätzlich kommen hier bereits die ersten Texturen für Fassaden und Dächer zum Einsatz. Neben Gebäuden lassen sich jetzt auch vermehrt Vegetations- und Infrastrukturobjekte darstellen, was das Gesamtbild gegenüber vorherigen Stufen deutlich realistischer macht (vgl. KAUFMANN 2021: 18; KOLBE 2009: 18f.).

Abbildung 12: LOD2 – Dachmodell
Quelle: Open Geospatial Consortium (OGC), unter: https://www.ogc.org/standards/citygml/ letzter Zugriff am 13.05.2025

2.1.3.4. LOD3 – Detailmodell

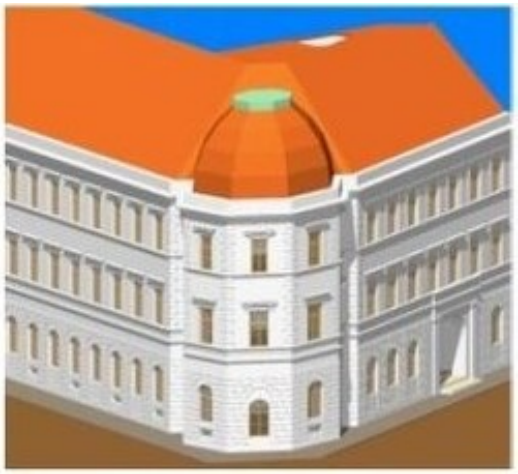

Abbildung 13: LOD3 – Detailmodell
Quelle: Open Geospatial Consortium (OGC), unter: https://www.ogc.org/standards/citygml/ letzter Zugriff am 13.05.2025

Mit dem LOD3 erreicht das Stadtmodell eine deutlich höhere Detailtiefe. Die Fassaden und Dachstrukturen werden genauer modelliert und mit zusätzlichen Elementen erweitert, darunter Fenster, Türen, Schornsteine. Auch größere Bauelemente wie Innenhöfe, Balkone und Außentreppen können hier modelliert und hinzugefügt werden (siehe Abbildung 13). Texturen mit höherer Auflösung werden auf die Gebäude projiziert und durch spezifische Materialien können beispielsweise Reflexionen in Fensterglas simuliert werden, wodurch das Stadtmodell noch realistischer dargestellt wird. Neben der Gebäudemodellierung erlaubt LOD3 auch eine differenzierte Darstellung von Vegetation, Stadtobjekten und Infrastrukturobjekten. Aufgrund des hohen Modellierungsaufwands wird diese Detaillierungsstufe vor allem für kleinere Gebiete eingesetzt, da eine flächendeckende Erstellung bislang nicht automatisiert möglich ist (vgl. EL-MEKAWY 2010: 19; KAUFMANN 2021: 19; KOLBE 2009: 18).

2.1.3.5. LOD4 – Innenraummodell

Abbildung 14: LOD4 – Innenraummodell
Quelle: Open Geospatial Consortium (OGC), unter: https://www.ogc.org/standards/citygml/ letzter Zugriff am 13.05.2025

Im Level of Detail 4 wird das Stadtmodell durch Innenrauminformationen erweitert (siehe Abbildung 14). Gebäude enthalten jetzt nicht nur Außenstrukturen, sondern auch modellierte Innenräume, welche auch mit modellierten Möbel oder technischen Installationen ausgestattet werden können. Das Ziel vom LOD4 ist es, die Verknüpfung von Innen- und Außenraum darzustellen und so ein umfassendes und virtuell begehbares Gebäudemodell zu schaffen. Auch Vegetation und Gewässer lassen sich in LOD4 detaillierter abbilden. Allerdings handelt es

sich dabei lediglich um eine Verfeinerung der Geometrie, was keine zusätzlichen inhaltlichen Informationen liefert (vgl. KAUFMANN 2021: 19; KOLBE 2009: 18f.).

Mit der Einführung von CityGML 3.0 wurde das LOD4-Konzept überarbeitet. Anstelle einer eigenen Stufe gibt es nun die Möglichkeit Innen- und Außenelemente innerhalb jedes LOD zu integrieren (vgl. BILJECKI et al. 2016: 26f.).

2.1.4. Datenquellen und Erfassungsmethoden

Die Erstellung von 3D-Stadtmodellen stützt sich vor allem auf vier zentrale Methoden. Die Nutzung von 2D-Karten, photogrammetrische Verfahren, Laserscanning sowie prozedurale Modellierungsverfahren. Welche Methode verwendet wird, hängt dabei stark von dem Anwendungsziel, den verfügbaren Ausgangsdaten und den technischen sowie wirtschaftlichen Rahmenbedingungen ab. In vielen größeren Städten liegen häufig bereits gewisse Geodaten vor. Beispielsweise digitale Geländemodelle, Orthofotos oder Gebäudegrundrisse aus Katasterkarten. Diese Daten stammen meist aus amtlichen Vermessungen und eignen sich besonders gut als Grundlage für die Modellierung einfacher Stadtmodelle. In diesem Kapitel werden die wichtigsten Verfahren zur Datenerfassung vorgestellt und näher erläutert. (vgl. DÖLLNER 2006: 105ff.; STOTER et al. 2020: 2f.; UGGLA 2015: 4f.).

2.1.4.1. Photogrammetrie

Photogrammetrie ist eine Methode der Fernerkundung, bei der Fotos genutzt werden, um die geometrischen Eigenschaften von sichtbaren Objekten zu bestimmen. Dabei können Größe, Form und Position eines Objektes durch die Auswertung vieler überlappender Bilder ermittelt werden. Die Auswertung wird dabei meist automatisiert über Algorithmen in spezieller Software durchgeführt, kann aber auch manuell durchgeführt werden.

Ein zentrales Prinzip der Photogrammetrie ist die Stereoskopie, also die Gewinnung räumlicher Information aus zwei oder mehr Bildern, die aus unterschiedlichen Blickwinkeln aufgenommen wurden. Dieses Verfahren ähnelt dem menschlichen Sehen, bei dem beide Augen aus verschiedenen Perspektiven auf dasselbe Objekt blicken und so räumliche Tiefe wahrnehmen können (vgl. UGGLA 2015: 5f.).

Grundsätzlich wird jeder reale 3D-Punkt von der Kamera auf genau einen 2D-Punkt im Bild projiziert, um aber die Tiefe, also die räumliche Lage dieses Punktes zu bestimmen, braucht man mindestens zwei Bilder, sowie die Parameter und Position der Kamera.

In Bezug auf 3D-Stadtmodellen wird Photogrammetrie bereits häufig und in verschiedenen Größenordnungen verwendet. Am bedeutendsten ist hier die sogenannte Airborne Photogrammetrie, bei der die Bilder aus der Luft aufgenommen werden, zum Beispiel von einem Flugzeug oder einer Drohne aus (siehe Abbildung 15). Mit diesen hochauflösenden Bilder lassen sich anschließend neben großmaßstäbige topographische Karten auch 3D-Modelle erstellen. Diese Methode eignet sich besonders gut für große Stadtgebiete, da die Befliegung gegenüber der terrestrischen Vermessung sehr zeitsparend und kosteneffizient ist (vgl. JEBUR et al. 2017: 3f.; UGGLA 2015: 4f.).

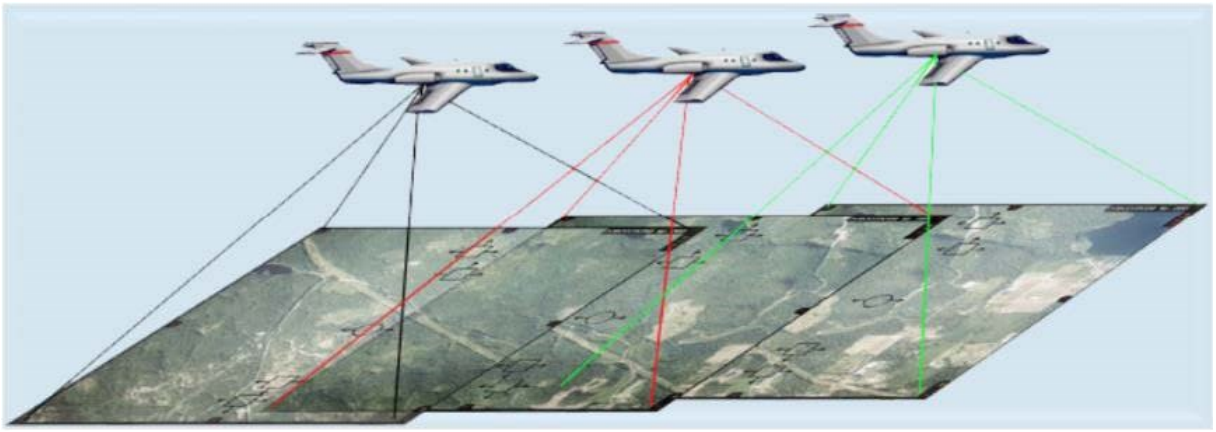

Abbildung 15: Airborne Photogrammetrie
Quelle: JEBUR et al. 2017

Scale-Invariant Feature Transform (SIFT)

SIFT ist ein Algorithmus, der dabei hilft, markante Punkte in Bildern zu erkennen. Diese Punkte nennt man Merkmale oder auch Schlüsselmerkmale. Der Algorithmus wurde von David G. Lowe entwickelt und analysiert die Bildstruktur, wie beispielsweise Kanten, Ecken oder Kontraste. Aus dieser Analyse werden dann sogenannte Feature Keys erstellt und wenn solche Merkmale in mehreren überlappenden Bildern gefunden werden und übereinstimmen, kann man davon ausgehen, dass es sich um denselben Punkt im Raum handelt. Wenn mindestens drei dieser Merkmale in zwei Bildern übereinstimmen, lassen sich daraus Rückschlüsse auf die Position der Kamera ziehen. Diese Information ist wichtig, um später die Lage der Bilder zueinander bestimmen zu können und in weiterer Folge daraus ein 3D-Modell zu erstellen (vgl. LOWE 1999: 1150ff.; UGGLA 2015: 5f.).

Der Vorteil von SIFT ist, dass diese Merkmale trotz Veränderungen im Bild, welche durch unterschiedliche Perspektiven oder Helligkeitsunterschieden entstehen können, wiedererkannt werden können. SIFT wird deshalb oft als erster Schritt in der automatisierten Auswertung von Bilddaten eingesetzt (vgl. LOWE 1999: 1150ff.; UGGLA 2015: 5f.).

Structure from Motion (SfM)

Structure from Motion ist ein Verfahren, mit dem man aus vielen einzelnen Fotos eine 3D-Szene rekonstruieren kann. Die Umgebung selbst bewegt sich dabei nicht, sondern nur die Kamera. Deshalb funktioniert SfM gut mit Drohnenaufnahmen oder Luftbildern.

Der Ablauf ist in mehreren Schritten organisiert. Zuerst werden in jedem Bild markante Punkte erkannt, zum Beispiel mithilfe von SIFT. Dann werden diese Punkte in verschiedenen Bildern miteinander verglichen und zugeordnet. Sobald genügend passende Punkte gefunden werden können die Kamerapositionen und die 3D-Koordinaten der Punkte im Raum berechnet werden. Hierbei wird darauf geachtet, dass der Unterschied zwischen den echten Bildpunkten und den projizierten Punkten möglichst klein ist. Das wird auch als Minimierung des Reprojektionsfehlers bezeichnet. Das Ergebnis dieser Berechnungen ist eine sogenannte Punktwolke (siehe Abbildung 16). Dabei handelt es sich um eine große Menge einzelner Punkte im dreidimensionalen Raum, die jeweils eine bestimmte Position in der Szene repräsentieren. Je mehr Bilder und Übereinstimmung der Bildpunkte vorhanden sind, desto besser und dichter ist die Punktwolke. In späteren Schritten lässt sich daraus dann ein vollständiges 3D-modell mit Flächen und Texturen ableiten (vgl. JEBUR et al. 2017: 23ff.; UGGLA 2015: 6f.).

Abbildung 16: Punktwolke
Quelle: All3DP 2025, unter: https://all3dp.com/de/1/photogrammetrie-programm-3d-scan/ letzter Zugriff am 13.05.2025

2.1.4.2. Laserscanning

Laserscanning ist ebenfalls eine Methode der Fernerkundung und wie bei der Photogrammetrie entsteht dabei am Ende eine Punktwolke. Der Unterschied zwischen diesen beiden Methoden liegt im Messprinzip. Während Photogrammetrie aus Fotos die geometrischen Informationen ableitet, werden diese beim Laserscanning direkt über Lichtimpulse gemessen.

Das Verfahren basiert auf dem Prinzip LiDaR (Light Detection and Ranging). Dabei werden Laserscanner verwendet, um gebündelte Lichtimpulse auszusenden. Diese Lichtstrahlen werden vom Zielobjekt reflektiert und vom Scanner wieder empfangen. Aus der Zeit, die das Licht für den Hin- und Rückweg braucht, kann die Entfernung zum Objekt abgeleitet werden. Da sich Licht mit konstanter Geschwindigkeit ausbreitet, ist diese Art der Messung sehr genau. Damit nicht nur ein einzelner Punkt gemessen wird, sondern große Flächen, muss der Boden systematisch abgetastet werden. Dafür wird der gebündelte Lichtstrahl mittels beweglicher Spiegel in verschiedene Richtungen gelenkt. Je nach System entstehen dabei verschiedene Scanmuster, was die Verteilung und Dichte der Punktwolke beeinflusst (vgl. JEBUR et al. 2017: 26ff.; STOTER et al. 2020: 2f.; UGGLA 2015: 8ff.).

Beim Airborne Laserscanning befindet sich der Scanner, wie bei der Airborne Photogrammetrie in einem Flugzeug oder einer Drohne und tastet das Gelände Stück für Stück ab (siehe Abbildung 17). Auch hier ist eine gewisse Überlappung der Flugstreifen nötig. Besonders in unregelmäßigem Gelände oder in städtischen Gebieten ist das wichtig, um eine gleichmäßige und vollständige Abdeckung zu erreichen (vgl. ebd.)

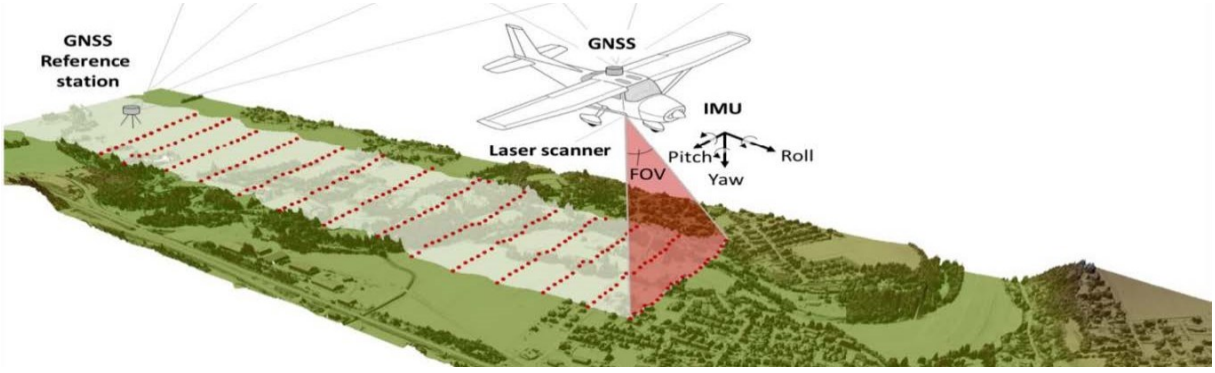

Abbildung 17: Airborne Laserscanning
Quelle: Spatial Media LLC 2025, unter: https://lidarmag.com/2024/12/30/airborne-lidar-a-tutorial-for-2025/ letzter Zugriff am 13.05.2025

2.1.4.3. Vektorbasierte Modellierung

Abbildung 18: Digitale Mehrzweckkarte
Quelle: FORKERT o.J.

Eine der einfachsten Methoden zur Erstellung von 3D-Stadtmodellen nutzt bereits vorhandene 2D-Vektordaten. Das können zum Beispiel Gebäudegrundrisse aus Katasterkarten oder Vektordaten aus Stadtplänen sein (siehe Abbildung 18). Diese Daten liegen in vielen Städten bereits digital vor, allerdings fehlen dabei häufig Höhenangaben der Gebäude. Trotzdem lassen sich daraus einfache 3D-Modelle erzeugen, indem die Gebäudegrundrisse extrudiert werden. Daraus entstehen die bereits erwähnten LOD1 – Blockmodelle (siehe Kapitel 2.1.3.2). Die Höhe der Gebäude wird dabei aus anderen Quellen bezogen, oder wenn keine Daten vorhanden sind, wird eine Standardhöhe verwendet, was für die meisten Anwendungen ausreicht (vgl. DÖLLNER et al. 2006: 107f.).

Die Vorteile dieser Methode sind vor allem die Effizienz und die automatisierte Erstellung. Man ist zwar begrenzt auf LOD1-Modelle, aber für Analysen im Bereich der Stadtplanung kann dies bereits ausreichen.

2.1.4.4. Prozedurale Modellierung

Prozedurale Modellierung ist ein skriptbasiertes Verfahren für die automatisierte Erstellung von 3D-Stadtmodellen. Im Gegensatz zu den bisher genannten Methoden wie Photogrammetrie oder Laserscanning, wird hierbei nicht die reale Welt erfasst und rekonstruiert, sondern eine virtuelle Geometrie mittels Algorithmen erzeugt. Die Grundlage dafür bilden sogenannte Regelwerke, meist in Form von Skripten, die beschreiben, wie Gebäude aufgebaut werden sollen (vgl. PARISH/MÜLLER 2001: 301ff.; SCHINKO et al. 2015: 469ff.; WATSON et al. 2008: 18f.).

Grundsätzlich besteht dieses Verfahren aus drei Komponenten. Dem Regelwerk, den Parametern, welche nach Wunsch konfiguriert, werden können und optionale Eingabedaten, wie beispielsweise 2D-Vektordaten. Das Regelwerk ist häufig ein Skript, geschrieben in speziellen Programmiersprachen wie etwa CGA (Computer Generated Architecture), welche

für automatische Architekturmodellierungen entwickelt wurde. In diesen Regeln können unter anderem Bedingungen festgelegt werden. Beispiele für solche Regeln könnten sinngemäß lauten: Wenn ein Gebäude höher ist als 10 Meter, füge ein zusätzliches Stockwerk hinzu; Setze alle 3 Meter entlang der Fassade ein Fenster. Die verwendeten Parameter erlauben zusätzliche Kontrolle von Größe, Material oder Form einzelner Gebäudeelemente und das Ziel ist es, mit möglichst wenigen Eingaben größere Mengen an 3D-Geometrie zu erzeugen (vgl. PARISH/MÜLLER 2001: 301f.; SCHINKO et al. 2015: 469f.).

Für die Umsetzung steht spezialisierte Software zur Verfügung, wie unter anderem die CityEngine von ESRI, Houdini, aber auch kostenfreie Varianten wie Blender mithilfe entsprechender Plugins. Die Erstellung von prozeduralen Stadtmodellen finden vor allem breite Anwendung in der Film- und Videospielindustrie, um fiktive, aber realistisch wirkende Städte/Umgebungen zu erzeugen. Allerdings hat dieses Verfahren auch seine Grenzen Insbesondere der Detailgrad ist stark abhängig von der Komplexität des Regelwerks, da einfach formulierte Skripten zu künstlich wirkenden und sich wiederholenden Mustern führen kann (siehe Abbildung 19) (vgl. PARISH/MÜLLER 2001: 302ff.; WATSON et al. 2008 20f.).

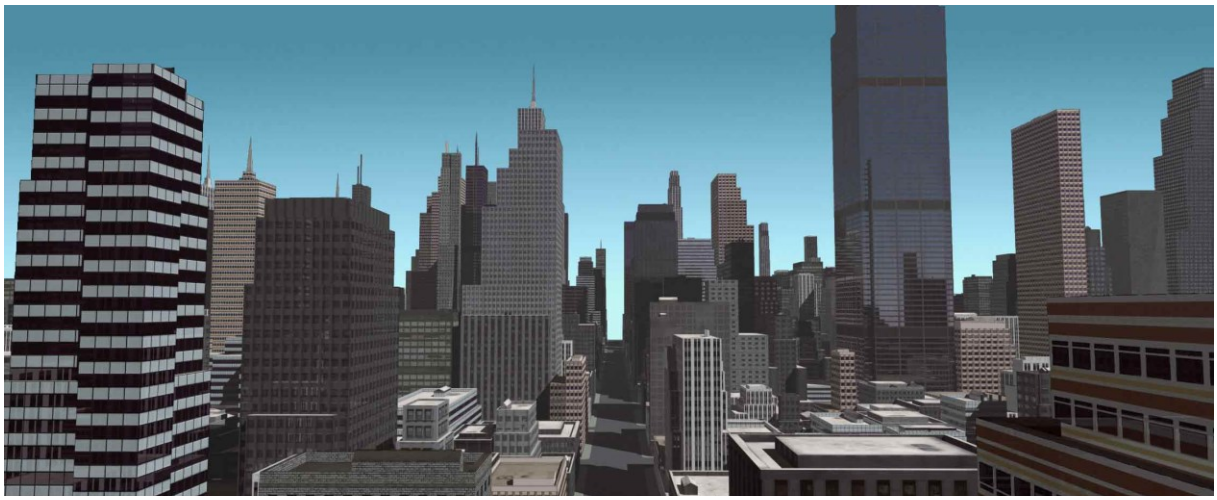

Abbildung 19: Beispiel einer prozedural modellierten Stadt
Quelle: PARISH/MÜLLER 2001

2.2. Grundlagen der Texturierung

Texturierung bezeichnet im Grunde den Prozess, bei dem zweidimensionale Bildinformationen, in der Regel Texturen genannt, auf dreidimensionale Geometrien projiziert werden. Die so erzeugte „optische Hülle“ vermittelt Materialeigenschaften wie Mauerwerk, Fenster, Fassaden usw. und verleiht auch einfachen geometrischen Modellen ein überzeugend realistisches Erscheinungsbild. Dabei geht es nicht nur um die Farbgebung. Durch spezielle Texturtypen wie unter anderem Bump-, Normal-, Roughness-Maps lassen sich auch feine Oberflächendetails und Lichtwechselwirkungen simulieren. Abbildung 20 zeigt anschauliche Beispiele für diese speziellen Texturen (siehe Kapitel 2.2.2.) (vgl. HECKBERT 1986: 56ff.; WEINHAUS/DEVARAJAN 1997: 325f.).

Besonders in großflächigen Szenen wie 3D-Stadtmodellen spielt die korrekte und effiziente Texturierung eine zentrale Rolle, denn sie erlaubt detaillierte Darstellungen bei vergleichsweise geringer Rechenlast. In den folgenden Kapiteln werden die technischen Grundlagen, die wichtigsten Texturtypen, sowie die häufigsten Methoden des Texture-Mappings näher erläutert (vgl. HECKBERT 1986: 57).

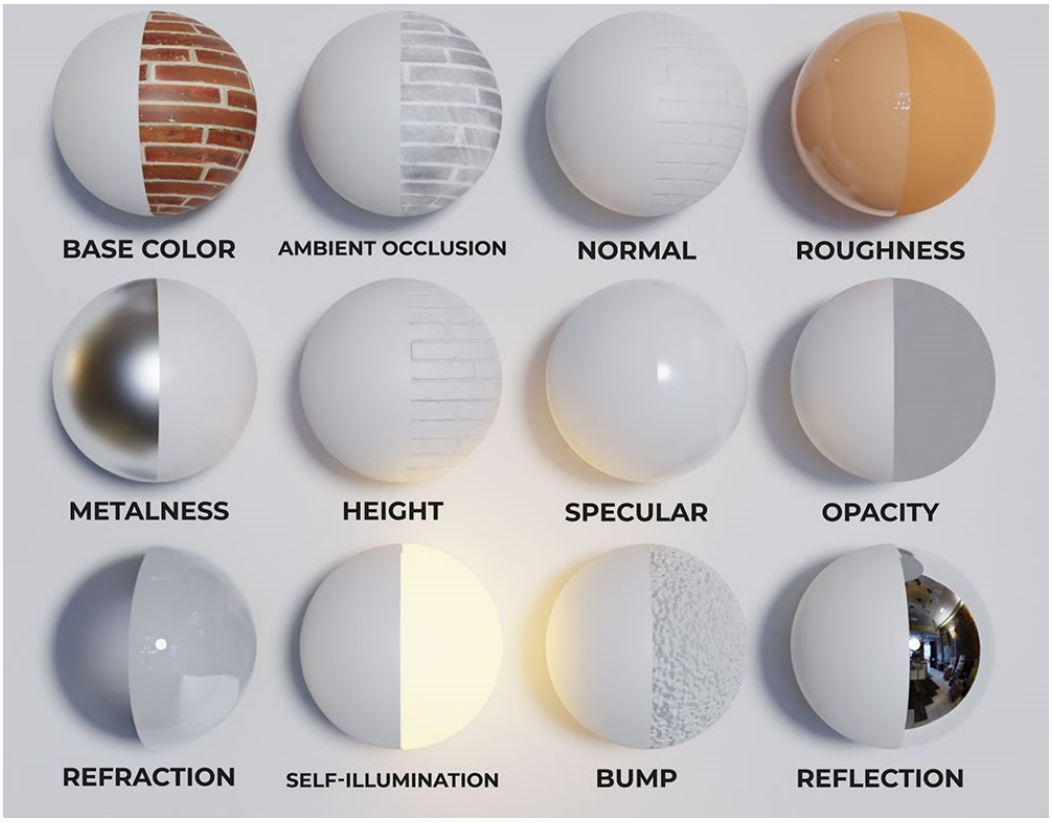

Abbildung 20: Texture-Maps
Quelle: 2014 – 2025 3D Studio, unter https://3dstudio.co/de/3d-texture-mapping/ letzter Zugriff am 13.05.2025

2.2.1. Technische Grundlagen

2.2.1.1. Texturkoordinaten

Die Texturkoordinaten sind ein wichtiger Bestandteil des Texture-Mappings, denn sie definieren, wie eine Textur auf das 3D-Objekt projiziert wird. Jedem Vertex, also einem einzelnen Eckpunkt eines 3D-Objekts, wird neben seiner Position zusätzlich eine Texturkoordinate im sogenannten UV-Raum zugeteilt. Die Bezeichnungen *u* und *v* stehen dabei für die Koordinatenachsen innerhalb der Textur. Neben den 2D-Texturen gibt es auch prozedurale und volumetrische Texturen, welche zusätzlich die *w* Achse als dritte Koordinatenachse verwenden (vgl. WEINHAUS/DEVARAJAN 1997: 329f.).

Bei der Darstellung zweidimensionaler Texturen wird häufig ein sogenanntes normiertes Koordinatensystem verwendet. Hierbei markieren die Koordinaten (0,0) die linke obere Ecke der Textur und (1,1) die rechte untere Ecke (siehe Abbildung 21). Dabei gilt es zu beachten, dass sich je nach Grafiksystem die Ausrichtung der V-Achse umkehren kann (vgl. WEINHAUS/DEVARAJAN 1997: 334f.).

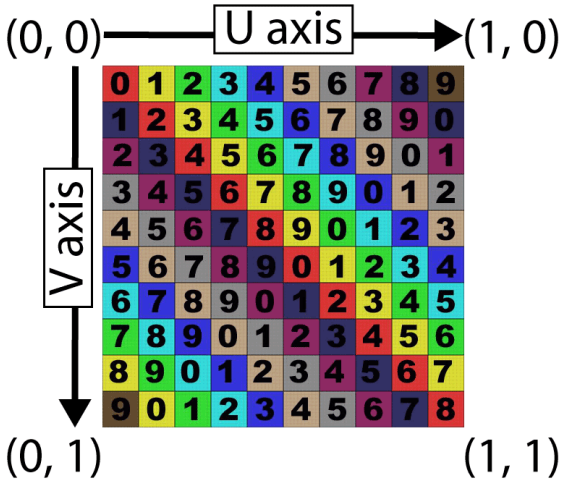

Abbildung 21: Normalized Texture Coordinates
Quelle: 3D Game Engine Programming, unter https://www.3dgep.com/learning-directx-12-4/#Texture_Coordinates letzter Zugriff am 13.05.2025

Die Verwendung solcher normierten Texturkoordinaten hat den Vorteil, dass sie unabhängig von der tatsächlichen Auflösung der Textur funktionieren. Anstelle von festen Pixelpositionen werden relative Positionen innerhalb der Textur angegeben. Dadurch müssen beim Rendern die genauen Pixelmaße nicht berücksichtigt werden, was besonders hilfreich ist, wenn sich die Auflösung der Textur ändert (vgl. ebd.).

2.2.1.2. Texturprojektionen

Bevor eine Textur auf ein 3D-Objekt projiziert werden kann, muss definiert werden, wie sie sich über die Oberfläche legen soll. Dafür gibt es verschiedene Projektionsmethoden, welche sowohl von der Geometrieform als auch dem Anwendungsfall abhängen. Die drei gängigsten sind die planare, zylindrische und sphärische Projektion.

Planare Projektion

Die planare Projektion ist die einfachste Form der Texturzuweisung. Hierbei wird die Textur senkrecht von einer gedachten Ebene aus auf das Modell projiziert (siehe Abbildung 22). Die Strahlen verlaufen alle parallel zur gewählten Achse. Mathematisch betrachtet läuft diese Projektion über eine affine Transformation, bei der die zweidimensionalen Koordinaten der Textur auf die Geometrie übertragen werden. Diese Methode ist vor allem dann sinnvoll, wenn es um relativ ebene Flächen geht. Bei stark gekrümmten oder unregelmäßigen Formen, wie in Abbildung 22 zu sehen, kommt es zu Verzerrungen, vor allem an Stellen, die schräg zur Projektionsebene stehen (vgl. HAEBERLI/MARK 1993: 264f.; HECKBERT 1986: 60f.).

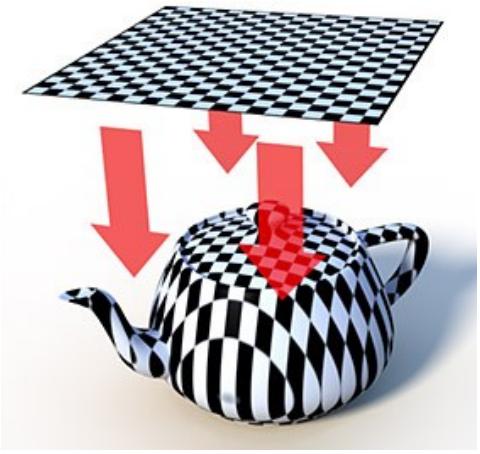

Abbildung 22: Planare Projektion
Quelle: The Foundry Visionmongers Ltd., unter: https://learn.foundry.com letzter Zugriff am 13.05.2025

Zylindrische Projektion

Bei der zylindrischen Projektion wird die Textur um das Objekt herumgewickelt (siehe Abbildung 23). Dafür wird die Fläche der Textur in ein Zylinderkoordinatensystem umgerechnet. Typischerweise wird die Zylinderachse entlang der X-, Y- oder Z-Achse definiert, je nachdem, wie das 3D-Objekt im Raum orientiert ist. Diese Projektion eignet sich besonders gut für Formen, die eine gewisse Rotationssymmetrie besitzen, wie zum Beispiel Säulen. Aber auch hier kann es zu starken Verzerrungen kommen, etwa bei Flächen, welche senkrecht zur Zylinderachse stehen (vgl. HAEBERLI/MARK 1993: 264f.).

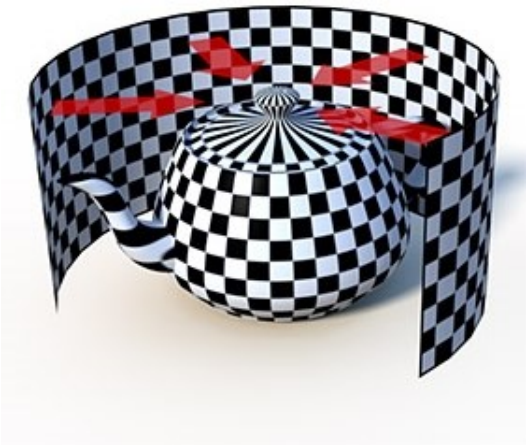

Abbildung 23: Zylinder Projektion
Quelle: The Foundry Visionmongers Ltd., unter: https://learn.foundry.com letzter Zugriff am 13.05.2025

Sphärische Projektion

Bei der sphärischen Projektion wird die Textur auf eine Kugel projiziert (siehe Abbildung 24). Dabei werden die Koordinaten für die Textur aus zwei Winkeln berechnet. Dem Azimut, also dem Längengrad und dem Polarwinkel, dem Breitengrad. Dazu wird angenommen, dass das Objekt im Zentrum einer virtuellen Kugel liegt, auf deren Oberfläche die Textur aufgelegt wird. Diese Methode eignet sich vor allem für Objekte, die in irgendeiner Form rund oder kugelig sind und wird häufig bei der Darstellung von Panoramen oder sogenannten Himmelskugeln verwendet. Der Nachteil hierbei ist, dass die Texture um die Pole herum stark gestaucht werden, weil viele Punkte auf engem Raum zusammenlaufen. Auch gibt es häufig eine Nahtstelle entlang eines Längengrades, an der sich die Textur schließt (vgl. WEINHAUS/DEVARAJAN 1997: 330ff.).

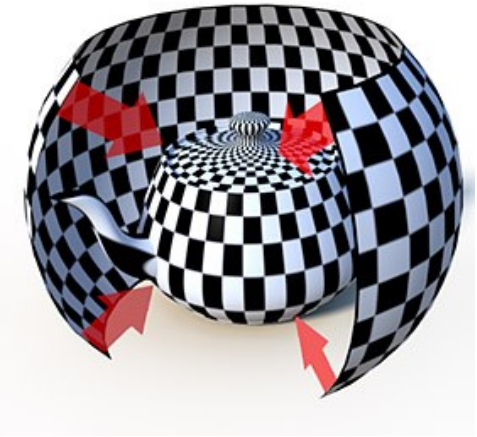

Abbildung 24: Sphärische Projektion
Quelle: The Foundry Visionmongers Ltd., unter: https://learn.foundry.com letzter Zugriff am 13.05.2025

UV-Unwrapping

Die bisher erwähnten geometriebasierten Projektionen sind in den meisten Fällen nur für einfache oder teilflächige 3D-Objekte geeignet. Für komplexere Modelle kommt das sogenannte UV-Unwrapping zum Einsatz. Hierbei wird die Geometrie des Objekts gewissermaßen aufgeschnitten und abgewickelt, sodass sie sich auf einer zweidimensionalen Ebene abbilden lässt. Diese Darstellung wird als UV-Inseln oder auch Charts bezeichnet (siehe Abbildung 25) und repräsentiert einzelne Bereiche der 3D-Oberfläche auf der Texturfläche. Der Vorteil hierbei ist, dass auch gekrümmte und komplexere Formen präzise und ohne Verzerrungen abgebildet werden können (vgl. VERMANDERE et al. 2024: 2f.).

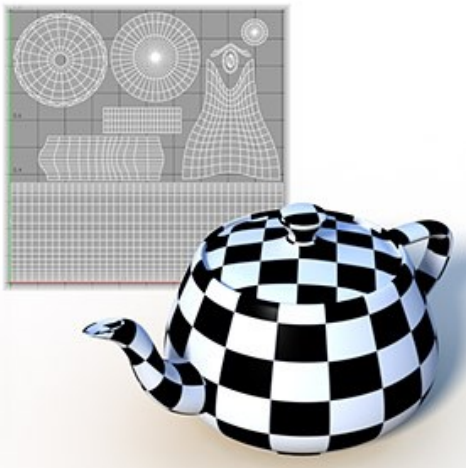

Abbildung 25: UV-Map
Quelle: The Foundry Visionmongers Ltd., unter: https://learn.foundry.com letzter Zugriff am 13.05.2025

2.2.1.3. Auflösung, Skalierung, Kachelung und Aliasing

Begriffe wie Auflösung, Skalierung Kachelung (Tiling) oder Aliasing spielen im Bereich der Texturierung eine zentrale Rolle für die korrekte Darstellung von Texturen.

Die Auflösung, also die Anzahl an Pixeln in einer Textur, auch Texel genannt, bestimmt maßgeblich den Detailgrad. Eine hohe Texelzahl, 2048x2048 oder 4096x4096 erlaubt es feine Strukturen, wie etwa die Details in Gebäudefassaden oder Mauerwerke auch bei Nahansicht realistisch darzustellen. Bei zu geringer Auflösung kann dieselbe Textur schnell unscharf oder pixeliert (Pixelation) erscheinen (siehe Abbildung 26), ein blockartiger Effekt, bei dem einzelne Texel als grobe Blöcke sichtbar werden (vgl. AKENINE-MÖLLER et al. 2018: 175ff.).

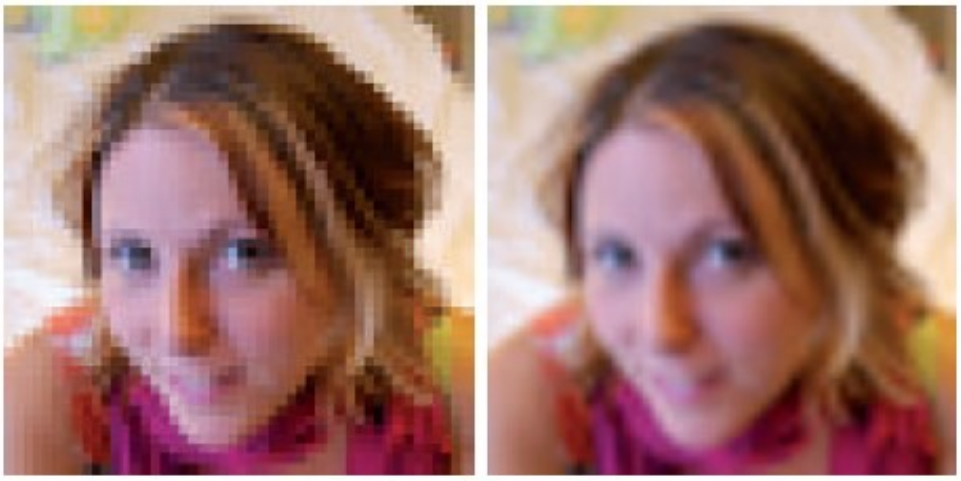

Abbildung 26: Pixelation – Beispiel
Quelle: AKENINE-MÖLLER et al. 2018

Skalierung beschreibt das Verhältnis zwischen Texturgröße und der Fläche, auf die sie projiziert wird. Bei Magnifikation, also einer starken Vergrößerung der Textur kann es ebenfalls zur Pixelation kommen. Bei einer Minifikation, also einer Verkleinerung der Textur entstehen dagegen oft sogenannte Moiré-Muster (siehe Abbildung 27). Dieser Effekt entsteht, wenn hochfrequente Texturmuster nicht ausreichend gesampelt werden und sich das Raster der Texel mit dem Raster der Bildpunkte überlagert (vgl. AKENINE-MÖLLER et al. 2018: 178f.).

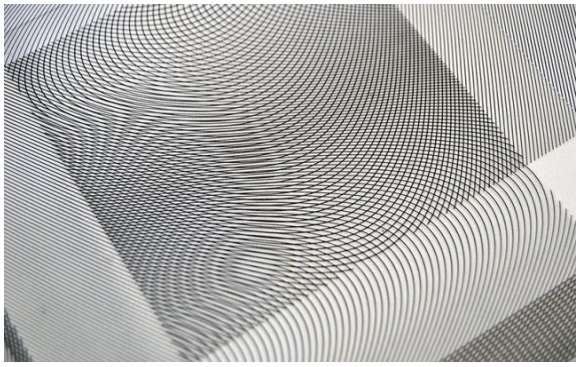

Abbildung 27: Moiré-Muster
Quelle: GeroWP, unter https://bewerbungsfotos-stuttgart.de/blog/der-moire-effekt-bei-kleidung/ letzter Zugriff am 13.05.2025

Die Kachelung oder auch Tiling im englischen ermöglicht es eine einzige kleine Texture mehrfach über eine größere Fläche zu legen. Dabei wird vor allem der Speicher gespart und die Grafikkarte (GPU) entlastet. Allerdings ist dabei darauf zu achten, dass die Ränder der Texturen nahtlos anschließen, sonst entstehen sichtbare Übergänge und Musterwiederholungen (siehe Abbildung 28). Heutzutage lassen sich diese Kacheln auch mithilfe von Parametern konfigurieren, um Rotationen oder Versetzungen einzubauen, sodass das Gesamtbild heterogener wirkt (vgl. AKENINE-MÖLLER et al. 2018: 175).

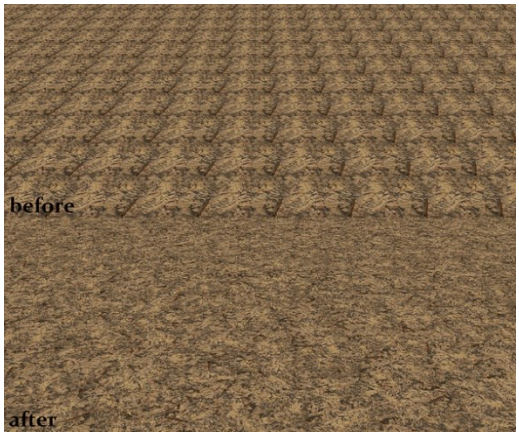

Abbildung 28: Texture-Tiling
Quelle: blenderartists.org, unter: https://blenderartists.org/t/huge-landscape-texture-tiling/1218489/4 letzter Zugriff am 13.05.2025

Das sogenannte Aliasing bezeichnet Bildartefakte, die auftreten, wenn beispielsweise feine Texturmuster oder Linien nicht ausreichend abgetastet werden. Das kann dazu führen, dass Informationen verloren gehen oder falsch interpretiert werden. Dadurch entstehen gezackte Kanten, auch als „Jaggies“ bezeichnet, flimmernde Bereiche oder störende Muster im Bild (siehe Abbildung 29). Verschiedene Techniken zur Reduzierung solcher Artefakte wie etwa Antialiasing werden in einem späteren Kapitel zur Texturoptimierung behandelt (vgl. AKENINE-MÖLLER et al. 2018: 130f.).

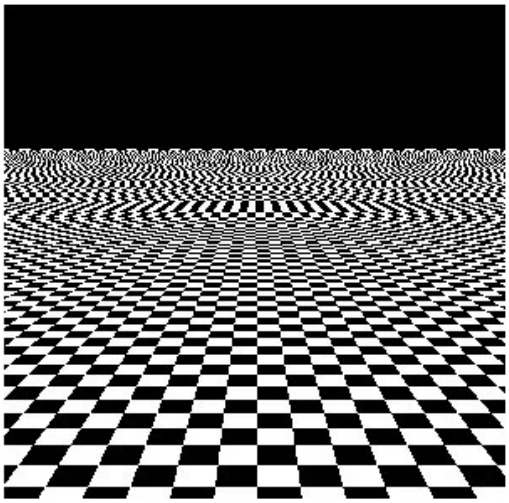

Abbildung 29: Aliasing Effekt
Quelle: Roldan Fritz, unter: https://textureingraphics.wordpress.com/what-is-texture-mapping/anti-aliasing-problem-and-mipmapping/ letzter Zugriff am 13.05.2025

2.2.2. Texturtypen

Wie bereits in der Einleitung zu Kapitel 2.2 erwähnt, gibt es neben der reinen Farbgebung noch eine Vielzahl weiterer Texturtypen, die gezielt bestimmte Materialeigenschaften beeinflussen, wie etwa Glanz, Rauigkeit, Transparenz oder Oberflächenstruktur. Diese Texturen bestimmen, wie das Licht mit dem Objekt interagiert und tragen wesentlich dazu bei

3D-Modelle realistischer darzustellen. Grundsätzlich lassen sich die Texturtypen in zwei Gruppen einteilen. Bildbasierte und prozedurale Texturen (vgl. AKENINE-MÖLLER et al. 2018: 167f.).

Bildbasierte Texturen basieren auf externen Bilddateien und im Kontext von 3D-Stadtmodellen können das beispielsweise Fotos von Gebäudefassaden sein. Sie werden häufig im Rahmen des sogenannten Physically Based Rendering (PBR) eingesetzt, einem weit verbreiteten Ansatz in der Computergrafik. Prozedurale Texturen hingegen werden nicht aus bestehenden Bildern übernommen, sondern werden algorithmisch erzeugt (vgl. DONG et al. 2020: 1; MCDERMOTT 2018: 45f.).

In den folgenden Kapiteln werden zunächst die gängigsten bildbasierten Texturtypen im PBR-Kontext vorgestellt und erläutert und im Anschluss folgt eine ausführliche Betrachtung prozeduraler Texturen und wie sie erstellt werden.

2.2.2.1. Bildbasierte Texturen

„Physically Based Rendering (PBR) ist ein Verfahren zur Schattierung und Darstellung, das eine genauere Abbildung davon ermöglicht, wie Licht mit Oberflächen interagiert.“ (MCDERMOTT 2018: 45). Hierbei kommen mehrere Texturtypen zum Einsatz, um verschiedene Aspekte eines Materials detailliert zu definieren. Diese Texturen liefern ihre Informationen an sogenannte Shader, kleine Programme innerhalb einer Render-Engine, die das Zusammenspiel von Licht und Material auf Grundlage dieser Daten visuell berechnet. Bei traditionellen Rendering Methoden wurden die Lichtreflexionen auf Oberflächen oft vereinfacht über zwei Parameter gesteuert, nämlich über Diffuse und Specular, welche die matte Grundfarbe, sowie Glanz und Spiegelungen beschreiben. PBR ersetzt diese vereinfachte Herangehensweise, indem das Verhalten von Licht realitätsnah simuliert wird (vgl. MCDERMOTT 2018: 22ff).

Abbildung 30 veranschaulicht, wie sich ein Lichtstrahl verhält, wenn er von einem Medium in ein anderes übergeht und dabei auf eine Oberfläche trifft. Abhängig von deren Beschaffenheit kann ein Teil des Lichts direkt reflektiert werden. Insbesondere bei glatten Flächen geschieht das in einem definierten Winkel (specular reflection). Bei raueren Oberflächen wird das Licht diffus in verschiedene Richtungen gestreut, was auch als diffuse Reflexion bezeichnet wird. Ein weiterer Teil des Lichts kann auch in das Material eindringen. Dort wird er absorbiert oder gestreut und mit veränderter Intensität oder Richtung wieder abgegeben. Dieser Vorgang wird

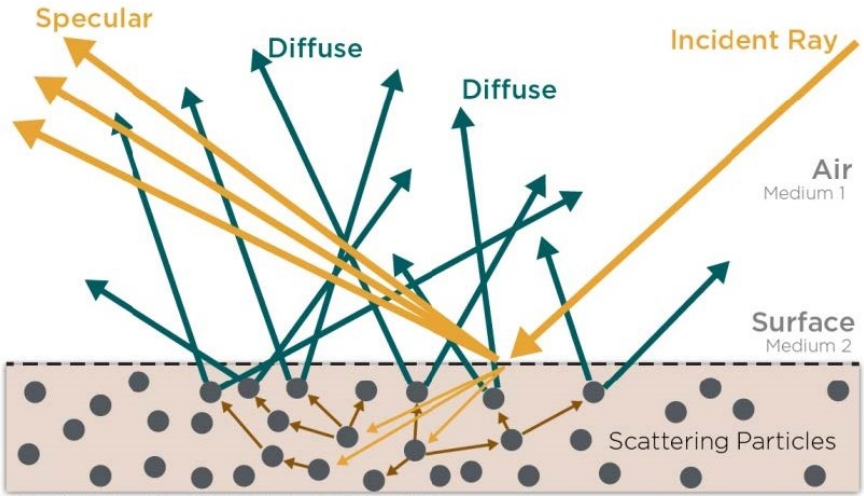

Abbildung 30: Verhalten von Licht an einer Oberfläche
Quelle: MCDERMOTT 2018

auch als Subsurface Scattering bezeichnet und zeigt sich besonders bei durchscheinenden Materialien wie Haut oder Wachs (vgl. MCDERMOTT 2018: 22ff.; NEPPIUS 2022: 16f.).

PBR berücksichtigt bei der Darstellung solcher Lichtverhalten zentrale physikalische Prinzipien. Dazu gehört das Prinzip der Energieerhaltung, das besagt, dass reflektiertes Licht niemals heller sein kann als das einfallende. Sowie der sogenannte Fresnel-Effekt, welcher beschreibt, dass Oberflächen bei flachen Einfallswinkeln mehr Licht reflektieren, was besonders bei glänzenden oder spiegelnden Materialien deutlich wird. Um diese Eigenschaften im Rendering umzusetzen, kommen verschiedene Texturtypen zum Einsatz. Zu den PBR-Texturen zählen grundsätzlich Base Color, Ambient-Occlusion, Normal, Roughness, Metallic sowie Height (vgl. MCDERMOTT 2018: 43ff.).

Base Color / Albedo Map

Die Base Color Texture, oft auch als Albedo-Map bezeichnet bildet die grundlegende Farbverteilung einer Oberfläche ab. Sie enthält ausschließlich Farbinformationen, also die sogenannte Eigenfarbe eines Objekts, ohne jegliche Licht- oder Schatteninformationen (siehe Abbildung 31). Im PBR-Kontext wird diese Textur linear interpretiert und sollte daher keine eingebetteten Lichtinformationen enthalten. Also keine Schlagschatten, keine spekulare Glanzlichter, keine Umgebungslichteinfälle. Das ist

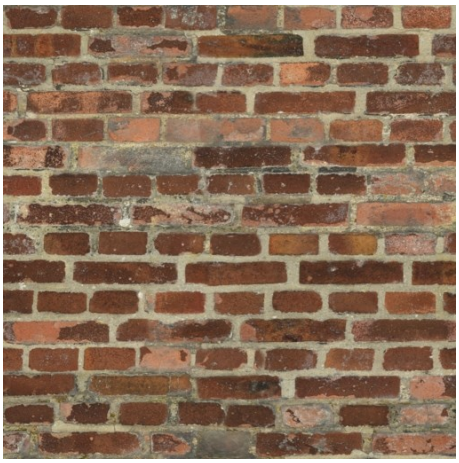

Abbildung 31: Base Color – Beispiel
Quelle: Quelle: ambientCG, unter: https://ambientcg.com/view?id=Bricks097 letzter Zugriff am 13.05.2025

wichtig, da solche Beleuchtungseffekte in einem späteren Schritt dem sogenannten Shader berechnet werden (vgl. AKENINE-MÖLLER et al. 2018: 201; MCDERMOTT 2018: 51ff.).

Da die Base Color die reine Materialfarbe beschreibt, sollten ihre RGB-Werte in einem realistischen Bereich liegen. Selbst sehr dunkle Oberflächen reflektieren noch einen kleinen Anteil des Lichts. Ein Wert von 0 sRGB, also absolut schwarz, wäre deshalb nicht korrekt und würde im Rendering unnatürlich aussehen. Um das zu vermeiden, empfiehlt sich ein Wertebereich von etwa 30 bis 240 sRGB. Das gleiche gilt auch bei sehr hellen Flächen (vgl. MCDERMOTT 2018: 51).

Metallic Map

Die Metallic Map ist eine Graustufentextur, die festlegt, welche Bereiche eines Materials als metallisch interpretiert werden. Dabei unterscheidet man zwischen leitenden (metallischen) und nicht-leitenden (dielektrischen) Oberflächen. Die Textur enthält keine realen Materialeigenschaften, sondern dient dem Shader als Informationsquelle zur Interpretation der Base Color (vgl. AKENINE-MÖLLER et al. 2018: 201; MCDERMOTT 2018: 47f.).

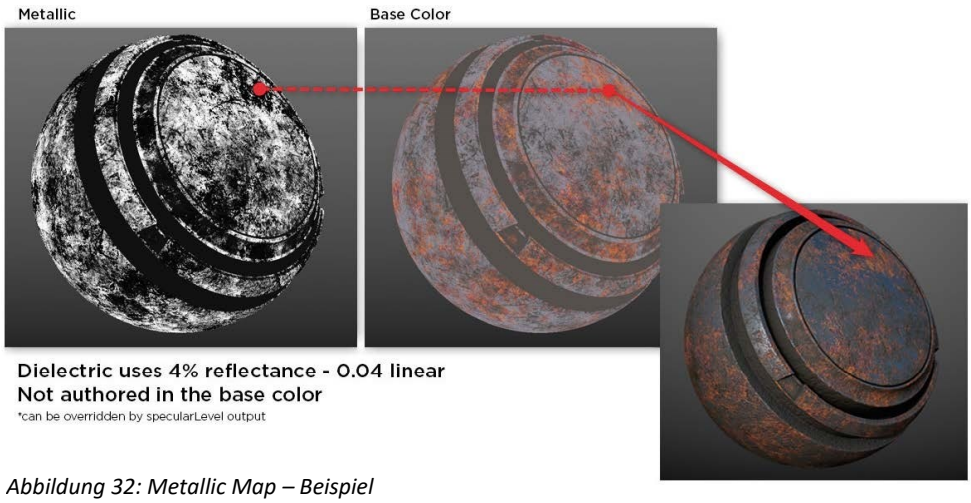

Abbildung 32: Metallic Map – Beispiel
Quelle: MCDERMOTT 2018

Ein Wert von 0 sRGB (Schwarz) steht für nicht-metallische Bereiche und ein Wert von 255 sRGB (Weiß) für rohe Metallflächen. In der Praxis ist die Metallic Map meist binär, also entweder Metall oder Nicht-Metall (siehe Abbildung 32). Graustufenwerte dazwischen können jedoch Übergangsbereiche darstellen, wie etwa bei verschmutzten oder rostigen Metallen. In solchen Fällen gilt, je geringer der Wert in der Metallic Map, desto stärker muss auch der Reflexionswert in der Base Color reduziert werden (vgl. MCDERMOTT 2018: 54f.; NEPPIUS 2022: 17f.).

Um den typischen Reflexionsgrad polierter Metalle realistisch darzustellen, sollten metallische Bereiche in der Base Color Helligkeitswerte zwischen 180 und 255 sRGB aufweisen. In der Praxis wird die Metallic Map entweder manuell erstellt, etwa durch Bildbearbeitung, bei der aus einem Farbfoto gezielt eine Graustufenmaske erzeugt wird, oder mithilfe von Generatoren in Texturing Tools wie Substance Designer automatisch erzeugt (vgl. MCDERMOTT 2018: 55).

Roughness Map

Die Roughness Map ist ebenfalls eine Graustufentextur, und beschreibt im Grunde die Rauheit einer Oberfläche, also wie stark das einfallende Licht gestreut oder reflektiert wird. Schwarz steht hierbei für eine glatte Oberfläche, auf der das Licht scharf und fokussiert reflektiert wird und weiß steht für eine raue Oberfläche, die das Licht diffus streut und matt erscheinen lässt. Im Gegensatz zur meist binären Metallic Map arbeitet die Roughness Map in der Regel mit kontinuierlichen Abstufungen zwischen Schwarz und Weiß, um feine Materialunterschiede darzustellen (siehe Abbildung 33) (vgl. AKENINE-MÖLLER et al. 2018: 369ff.; MCDERMOTT 2018: 59ff.).

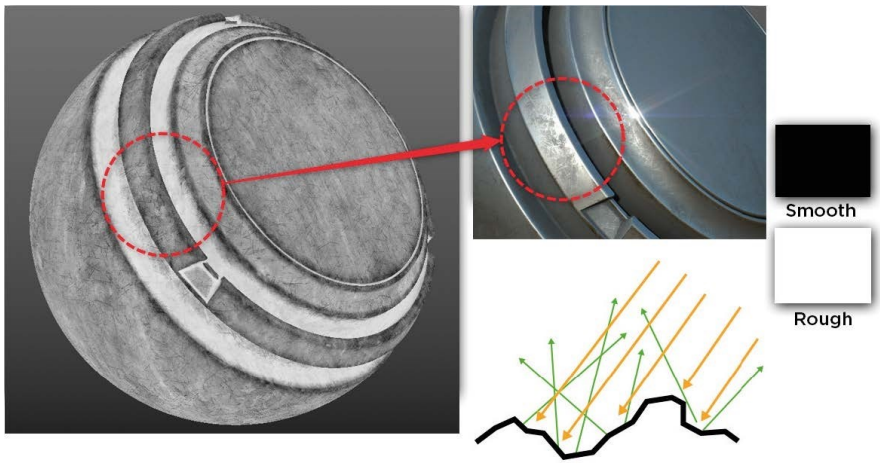

Abbildung 33: Roughness Map – Beispiel
Quelle: MCDERMOTT 2018

Im Kontext von 3D-Stadtmodellen spielt die Roughness Map nicht nur bei der Fassadengestaltung eine Rolle, sondern insbesondere auch bei Fenstergläsern. Auch wenn Glas meist transparente Eigenschaften aufweist, sind Fensterflächen moderner Gebäude häufig stark reflektierend und spiegeln ihre Umgebung. In solchen Fällen wird der Roughness-Wert auf nahe 0 definiert, damit die Oberfläche spiegelglatt erscheint.

Für eine realistische Darstellung spielt neben der Qualität der Roughness Map auch ihre technische Umsetzung eine wichtige Rolle. Besonders bei Materialgrenzen oder Übergängen, wie beispielsweise zwischen metallischen Fensterrahmen und nichtmetallischen Fassadenbereichen, kann es durch Texturinterpolation zu sogenannten „White Edge Artefacts“ kommen (siehe Abbildung 34). Diese entstehen, wenn stark unterschiedliche Werte in der Roughness- und Base Color-Map interpoliert werden und sich helle Ränder zeigen. Solche Artefakte sind vor allem bei Nahaufnahmen sichtbar und werden durch zu geringe Auflösung oder schlecht skalierten UVs zusätzlich verstärkt (vgl. MCDERMOTT 2018: 60f.).

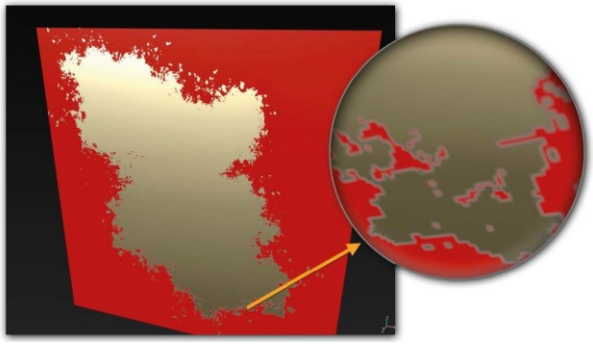

Abbildung 34: White Edge Artifacts
Quelle: MCDERMOTT 2018

Ambient Occlusion Map (AO)

Die Ambient Occlusion Map (AO) ist ebenfalls eine Graustufentextur und definiert, wie stark das Umgebungslicht an einer Oberfläche zugänglich ist. Sie simuliert dabei vor allem die Abschattung in Ecken, Vertiefungen und anderen engen Bereichen, wo das Umgebungslicht durch nahe liegende Geometrie teilweise blockiert wird. Dadurch entsteht ein realistischer Tiefeneffekt, der die Oberfläche plastischer und detailreicher wirken lässt. AO beeinflusst dabei nur den diffusen Lichtanteil und sollte die spekulare Reflexion nicht abschwächen (vgl. AKENINE-MÖLLER et al. 2018: 446ff.; MCDERMOTT 2018: 74f.).

Schwarz bedeutet hierbei, dass die Fläche komplett abgeschattet ist, also kaum Umgebungslicht empfängt und Weiß bedeutet, dass die Fläche vollständig beleuchtet wird. Auch hier werden wieder kontinuierliche Abstufungen eingesetzt, um natürliche Übergänge zu erzeugen. Abbildung 35 zeigt eine AO-Map einer Gebäudefassade.

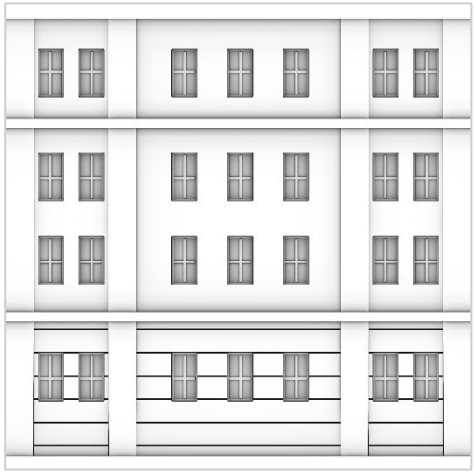

Abbildung 35: Ambient Occlusion Map
Quelle: Eigene Darstellung

In modernen Shadern wird die AO-Map als eigene Textur genutzt und mit der Umgebungsbeleuchtung

multipliziert. Alternativ kann die AO-Information auch direkt in die Base Color Map eingebettet werden. Dadurch wird zwar Speicher gespart und die Anzahl der Texturen reduziert, aber man verliert auch Flexibilität, da die AO nicht mehr unabhängig von der Base Color im Shader konfiguriert werden kann. Die AO-Map lässt sich entweder direkt aus dem 3D-Objekt erzeugen oder durch Filter, die auf Normal- oder Höhenkarten basieren (vgl. MCDERMOTT 2018: 74f.).

Normal Map

Die Normal Map ist eine RGB-Textur, mit der Oberflächendetails simuliert werden können, ohne zusätzliche Geometrie zu erzeugen. Hierbei werden die Richtung der Oberflächennormalen als RGB-Farbwerte gespeichert und die Farbkanäle entsprechen den X-, Y- und Z-Komponenten der Normale. In der Praxis entspricht beispielsweise der Farbwert [128, 128, 255] (hellblau) einer Normale, die senkrecht nach oben zeigt, also einer flachen Oberfläche ohne Neigung. Die Abweichungen davon definieren Vertiefungen bzw. kleine Wölbungen (siehe Abbildung 36) (vgl. AKENINE-MÖLLER et al. 2018: 211f.; MCDERMOTT 2018: 78f.).

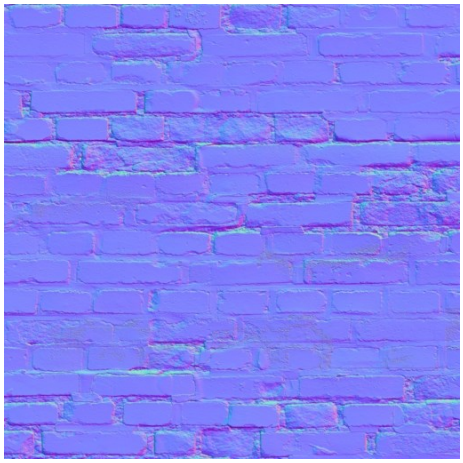

Abbildung 36: Normal Map – Beispiel
Quelle: ambientCG, unter: https://ambientcg.com/view?id=Bricks097 letzter Zugriff am 13.05.2025

Normal Maps kommen typischerweise bei sogenannten Low-Poly-Modellen zum Einsatz, um den Detailgrad eines High-Poly-Modells auf ein einfacheres Modell zu übertragen (siehe Abbildung 37). Dabei steht Poly für Polygon, den geometrischen Flächen, aus denen ein 3D-Modell besteht. Je höher die Anzahl der Polygone, desto detailreicher, aber auch rechenintensiver ist ein 3D-Modell (vgl. NEPPIUS 2022: 12f.).

Abbildung 37: High-Poly-Modell (links), Low-Poly-Model (mitte), Normal Map (rechts)
Quelle: PolycountWiki, unter: http://wiki.polycount.com/wiki/Normal_map letzter Zugriff am 13.05.2025

Die gängigste und qualitativ hochwertigste Methode zur Erstellung von Normal Maps ist das sogenannte Baking, also das Einbacken von Details. Hierfür werden die geometrischen Details eines komplexen Modells präzise in Normaldaten übertragen. Programme wie Substance Designer oder Blender bieten dafür integrierte Baking-Funktionen. Alternativ können sich Normal Maps auch durch die Konvertierung eine Höhenkarte mittels spezieller Nodes erzeugen (vgl. MCDERMOTT 2018: 78; NEPPIUS 2022: 12f.).

Da Normal Maps direkt in die Beleuchtungsberechnung auf Pixelebene eingreifen, ist ihre Verarbeitung technisch anspruchsvoll. Besonders bei nichtlinearen Lichtmodellen, wie spekularem Glanz, kann eine unsaubere Filterung zu unschönen Effekten führen. Auch die Komprimierung von Normal Maps kann problematisch sein. Um Speicher zu sparen, werden oft nur zwei der drei Farbkanäle gespeichert und der dritte später rekonstruiert. Das funktioniert allerdings nur zuverlässig, wenn die Normalvektoren korrekt normalisiert sind. Andernfalls kann es zu sichtbaren Fehlern oder Artefakten kommen (vgl. AKENINE-MÖLLER et al. 2018: 213f.).

Vor dem Aufkommen von Normal Maps war Bump Mapping die verbreitetste Methode zur Simulation von Oberflächenstruktur. Eine Graustufentextur dient hier zur Darstellung von Höhenunterschieden (siehe Abbildung 38). Aus diesen Werten werden bei der Beleuchtung lokale Abweichungen der Normalen approximiert, ohne die Geometrie dabei zu verändern. Da aber keine echten Richtungsinformationen gespeichert werden, sind die Ergebnisse in ihrer Aussagekraft begrenzt, vor allem bei komplexen Lichtverhältnissen oder sehr flachen Betrachtungswinkeln. In heutigen 3D-Enginse sind Bump Maps weitestgehend durch Normal Maps ersetzt worden (vgl. AKENINE-MÖLLER et al. 2018: 208ff.).

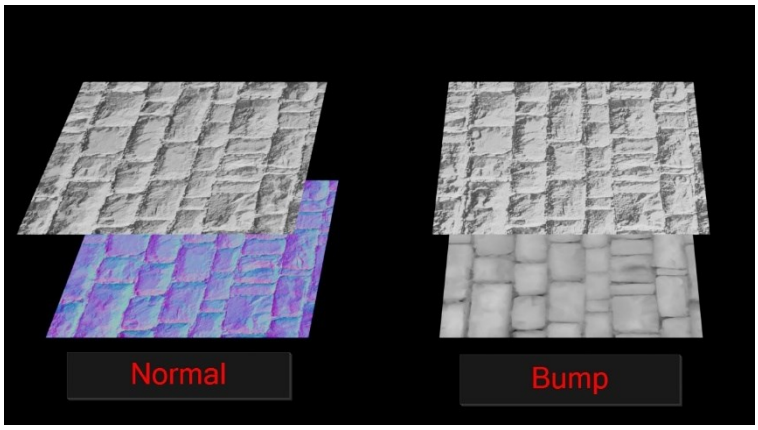

Abbildung 38: Normal Map vs. Bump Map
Quelle: 3DModels LTD, unter: https://3dmodels.org/blog/normal-maps-in-blender-guide/ letzter Zugriff am 13.05.2025

Height Map

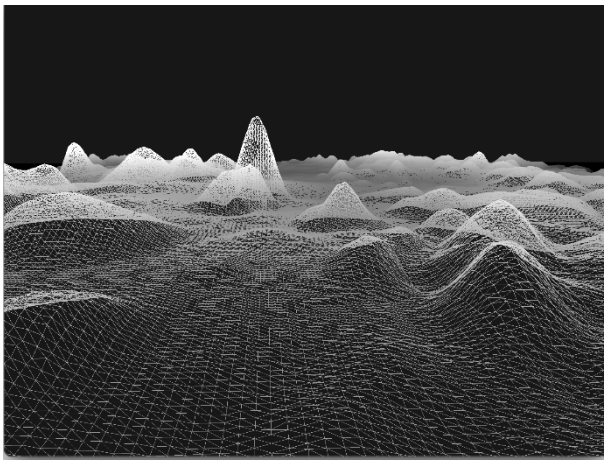

Abbildung 39: Geländemodell mit einer Height Map Quelle: Learn OpenGL, unter: https://learnopengl.com/Guest-Articles/2021/Tessellation/Height-map letzter Zugriff am 13.05.2025

Die Height Map ist ebenfalls eine Graustufentextur und speichert Höheninformationen. Weiß steht dabei für die höchsten Stellen der Oberfläche und schwarz für die tiefsten. Anders als bei der Normal oder Bump Map wird hier aber keine Illusion von Plastizität simuliert, sondern tatsächlich die Geometrie der Oberfläche verformt. Damit dieses Displacement überhaupt sichtbar wird, muss das zugrunde liegende Modell über ausreichend viele Polygone verfügen. Aus einer groben Fläche mit wenigen Vertices lässt sich keine feine Geometrie ableiten. In Programmen wie Blender kann man die nötige Polygonanzahl zum Beispiel über einen Subdivide-Befehl erhöhen. In Echtzeit-Engines wird stattdessen häufig die sogenannte Tessellation verwendet, um die Geometrie des 3D-Modell dynamisch zu erhöhen, was aber auch mehr Rechenleistung benötigt (vgl. MCDERMOTT 2018: 75f.).

2.2.2.2. Prozedurale Texturen

Texturen müssen nicht zwangsläufig aus Bilddateien bestehen, sondern können auch prozedural generiert werden, also mithilfe von mathematischen Algorithmen. Statt Pixelwerte fest in einem Bild zu speichern, definiert ein Algorithmus, wie der Farb- oder Höhenwert an jeder beliebigen Koordinate zu berechnen ist. Solche Verfahren erzeugen unter anderem verschiedene Rauschmuster, zufällige Verteilungen oder fraktale Strukturen, welche dann wiederum in nodebasierten Workflows kombiniert werden können, um komplexe Texturen zu erzeugen (vgl. DONG et al. 2020: 1f.; EBERT et al. 2002: 11f.).

Prozedurale Texturen bieten eine Reihe wichtiger Vorteile gegenüber bildbasierten Texturen. Sie sind äußerst speichereffizient, da ihre mathematische Definition meist nur wenige Kilobyte umfasst, während Bildtexturen schnell mehrere Megabyte benötigen. Zudem sind prozedurale Texturen auflösungsunabhängig, das heißt sie liefern auch bei starker Vergrößerung detaillierte Ergebnisse, ohne sichtbare Pixelartefakte. Ein weiterer Vorteil ist ihre unbegrenzte Ausdehnung, da sie theoretisch unendlich große Flächen abdecken können,

ohne Wiederholungsmuster oder sichtbare Übergänge zwischen den Abschnitten. Außerdem lassen sich prozedurale Texturen über Parameter steuern, sodass sich aus einem einzigen Algorithmus eine Vielzahl verwandter Variationen erzeugen lässt (vgl. EBERT et al. 2002: 14f.).

Allerdings bringen sie auch Herausforderungen mit sich. Die Erstellung und das Debugging prozeduraler Texturen erfordert oft komplexes Programmieren, was den Einstieg erschweren kann. Zudem ist das Ergebnis nicht immer leicht vorhersehbar. Je nach Perspektive kann diese Eigenschaft als kreative Freiheit oder als Mangel an Kontrolle gesehen werden. Auch in Bezug auf Rechenleistung gibt es Nachteile, denn das Auswerten von Algorithmen zur Laufzeit kann langsamer sein als das einfache Laden einer Bilddatei (vgl. ebd.).

In modernen nodebasierten Workflows wie Substance Designer oder Blender sind prozedurale Texturen zentral. Dort kommen Bausteine wie unter anderem Noise, Voronoi, Wave oder Gradient zum Einsatz. In den folgenden Kapiteln werden die wichtigsten prozeduralen Texturtypen vorgestellt und näher erläutert.

Noise Texture

Die Noise Texture zählt zu den grundlegendsten und vielseitigsten prozeduralen Texturen. Sie basiert auf zufallsähnlichen Mustern mit weichen Übergängen und dient als Ausgangspunkt für vielfältige visuelle Effekte. Die Entwicklung der Noise Texture geht auf den sogenannten Perlin Noise zurück (siehe Abbildung 40), welchen Ken Perlin in den 1980er Jahren einführte. Diese Form des gradientenbasierten Rauschens war ein Meilenstein in der prozeduralen Texturerzeugung (vgl. DONG et al. 2020: 10ff.; EBERT et al. 2002: 590ff.).

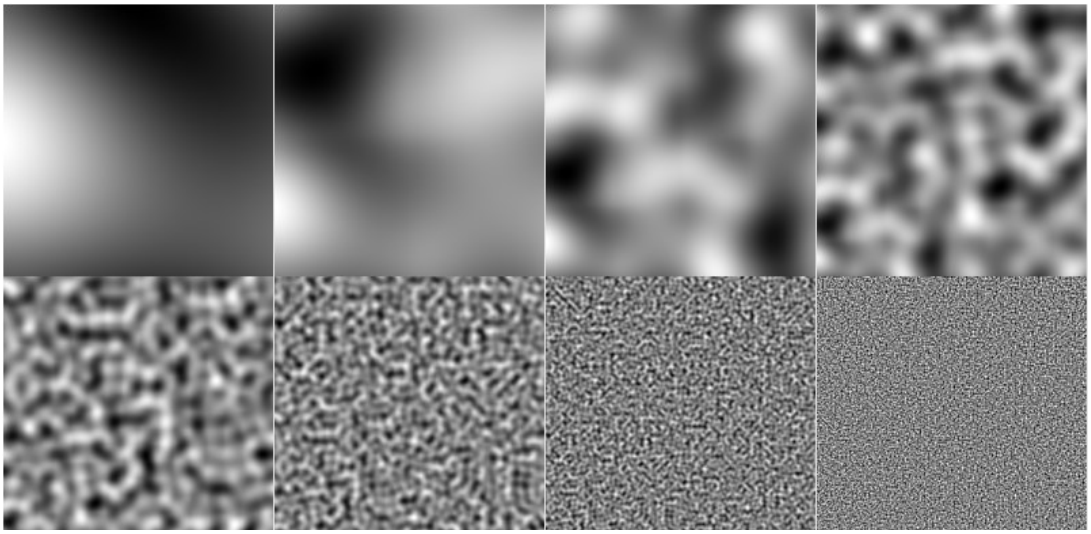

Abbildung 40: Perlin Noise – Different Frequencies
Quelle: Tomáš Bouda, unter: https://medium.com/100-days-of-algorithms/day-88-perlin-noise-96d23158a44c letzter Zugriff am 13.05.2025

Die Erscheinung der Noise Texture lässt sich durch verschiedene Parameter flexibel steuern. Diese Parameter ermöglichen eine präzise Anpassung und Steuerung der Textur für gestalterische oder technische Anforderungen (vgl. DONG et al. 2020: 2ff.; EBERT et al. 2002: 69ff.):

- **Scale** beeinflusst die Frequenz bzw. Größe des Musters. Kleine Werte erzeugen grobe Strukturen, große Werte feinere Details.
- **Detail** legt die Anzahl kombinierter Fraktalebenen fest und erhöht somit den Detailgrad der Textur.
- **Distortion** führt gezielte Verzerrungen ein, um das Muster abwechslungsreicher und organischer wirken zu lassen
- **Roughness** bestimmt die Ausprägung feiner Strukturen über mehrere Ebenen hinweg und sorgt für ein raueres Erscheinungsbild.

Neben der bereits erwähnten Perlin Variante existieren eine Vielzahl weiterer Noise-Typen, die entweder direkt auf der Fläche oder im Frequenzbereich generiert werden. Dazu zählen unter anderem:

- **Fractal Noise** (z. B. fractional Brownian motion)
- **Wavelet Noise** (zur Reduktion von Aliasing und Detailverlust)
- **Anisotropic Noise** (mit gerichteter Frequenzverteilung)
- **Fourier Noise** (auf Basis kontrollierter Power-Spektren)
- **Sparse Convolution Noise** (für detailreiche Strukturen bei geringem Rechenaufwand)
- **Spot Noise** (für Spezialfälle wie Strömungs-Visualisierung)

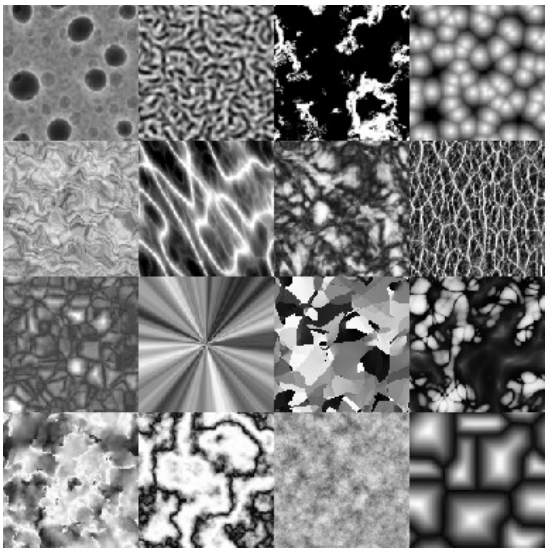

Abbildung 41: Noise Textures – Variationen
Quelle: Screaming Brain Studios, unter: https://opengameart.org/content/700-noise-textures letzter Zugriff am 13.05.2025

Typische Anwendungen umfassen unter anderem oxidierte oder beschädigte Metalloberflächen, verwitterte Steine, Fassadenoberflächen, oder abstrakte, organische Muster. Auch atmosphärische Effekte wie Nebel, Wolken oder Wasseroberflächen lassen sich mithilfe von Noise Texturen gestalten. Für stilisierte Zwecke kommen sie bei der Erzeugung von Grunge-Effekten, Scanlines oder Masken zum Einsatz. Besonders durch die Kombination mehrerer Noise-Schichten entsteht ein hohes Maß an gestalterischer Flexibilität (vgl. EBERT et al. 2002: 69ff.).

Voronoi Texture

Die Voronoi Texture gehört zu den geometriebasierten Verfahren und erzeugt zellenartige Muster, die sich an der natürlichen Struktur von Materialien von Stein, Haut oder Eis orientieren. Sie basiert auf dem sogenannten Voronoi-Diagramm. Hierbei wird der Raum in Bereiche aufgeteilt, die jeweils einem von mehreren zufällig verteilten Punkten am nächsten liegen (siehe Abbildung 42). Diese Punkte werden auch Feature-Points bezeichnet. Die dadurch entstehenden Zellen weisen klare Kanten auf und erzeugen ein visuell charakteristisches, polygones Muster. Die Voronoi Texture ist damit eine spezielle Form der sogenannten Cellular Texture und lässt sich ebenfalls durch verschiedene Parameter steuern (vgl. AKENINE-MÖLLER et al. 2018: 144; EBERT et al. 2002: 136ff.):

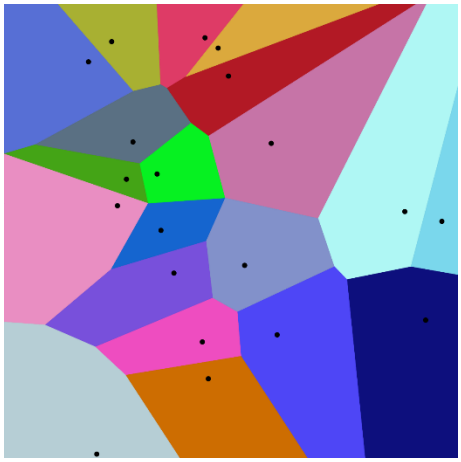

Abbildung 42: Voronoi-Diagramm
Quelle: Wikimedia Foundation Inc., unter: https://en.wikipedia.org/wiki/Voronoi_diagram letzter Zugriff am 13.05.2025

- **Feature Density** bestimmt, wie viele Zellen auf einer Flächen entstehen.
- **Distance Metric** legt fest, wie die Entfernung zwischen den Punkten berechnet wird (z.B. euklidisch oder Manhattan-Distanz)
- **Combination Method** beschreibt, wie die verschiedenen Abstandsinformationen zusammengeführt werden (z.B. Differenz oder Summe mehrerer Abstände)
- **Jitter** führt eine gezielte Unregelmäßigkeit in der Positionierung der Feature Points ein, um natürliche Variationen zu erzeugen.

Die erstellten Muster werden insbesondere zur Erzeugung zellenartiger, modulhafter oder organisch wirkender Muster eingesetzt. In 3D-Stadtmodellen eignen sich die Voronoi-Muster gut für die Texturierung von gepflasterten Gehwegen oder für einzelne Fassadenelemente. Darüber hinaus eignen sie sich für stilisierte Muster, wie etwa Risse in einem Material, welche in Kombination mit einer Normal Map zu einer realistischeren Materialdarstellung beitragen (vgl. DONG et al. 2020: 11; EBERT et al. 2002: 136ff.).

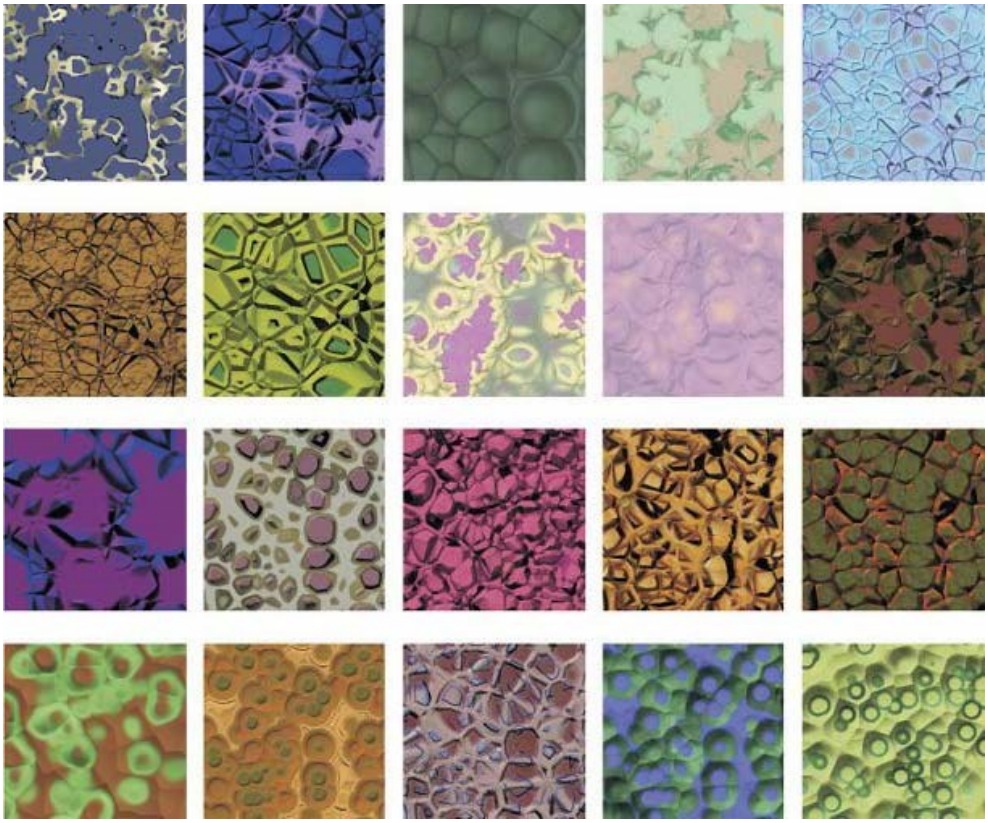

Abbildung 43: Voronoi Muster – Variationen
Quelle: EBERT et al. 2002

Wave Texture

Die Wave Texture basiert auf periodischen Wellenfunktionen wie Sinus und Kosinus und erzeugt dabei regelmäßige, richtungsgebundene Muster. Im Gegensatz zu den zufallsbasierten Noise-Texturen bietet die Wave Texture ein festes, regelmäßig wiederholbares Muster, welches darauf basiert, dass jede beliebige Funktion durch eine Summe von Sinus- und Kosinusfunktionen dargestellt werden kann. Dieses Prinzip liegt der sogenannten Fourier-Analyse zugrunde (vgl. DONG et al. 2020: 14f.; EBERT et al. 2002: 48ff.).

Die bekannteste Form der Wave Texture beruht auf der Fourier Spectral Synthesis, bei der verschiedene Wellen unterschiedlicher Frequenz, Amplitude und Phase überlagert werden

und dadurch komplexe Strukturen erzeugen. Hierbei werden gezielt sinusförmige Komponenten summiert, sodass die Textur sowohl reguläre als auch variantenreiche Muster aufweisen kann. Zur Steuerung der Wave Texture stehen, ähnlich wie bei der Noise Texture, mehrere Parameter zur Verfügung (vgl. DONG et al. 2020: 14f.; EBERT et al. 2002: 48ff.).

- **Scale** kontrolliert die Frequenz des Musters.
- **Detail** regelt den Grad an überlagerten Wellenkomponenten, was zu einem komplexeren Muster führt.
- **Distortion** erzeugt Unregelmäßigkeiten in der Wellenform.
- **Phase Offset** ist ein spezifischer Parameter der Wave Texture und beeinflusst die Verschiebung der Wellen entlang einer Achse. Dies kann insbesondere bei Animationen oder überblendenden Texturen genutzt werden, um dynamische Effekte zu erzeugen.

Bei geringer Verzerrung durch den Distortion Parameter entstehen klar definierte, regelmäßige Strukturen, die im Kontext von 3D-Stadtmodellen bei der Texturierung von Fassadenelementen wie Lamellen, Rippen oder metallischen Paneelen verwenden werden können. Erhöht man den Distortion-Wert, ergeben sich unregelmäßigere, organischere Musterformen, welche beispielsweise für Wasseroberflächen verwendet werden können (vgl ebd.).

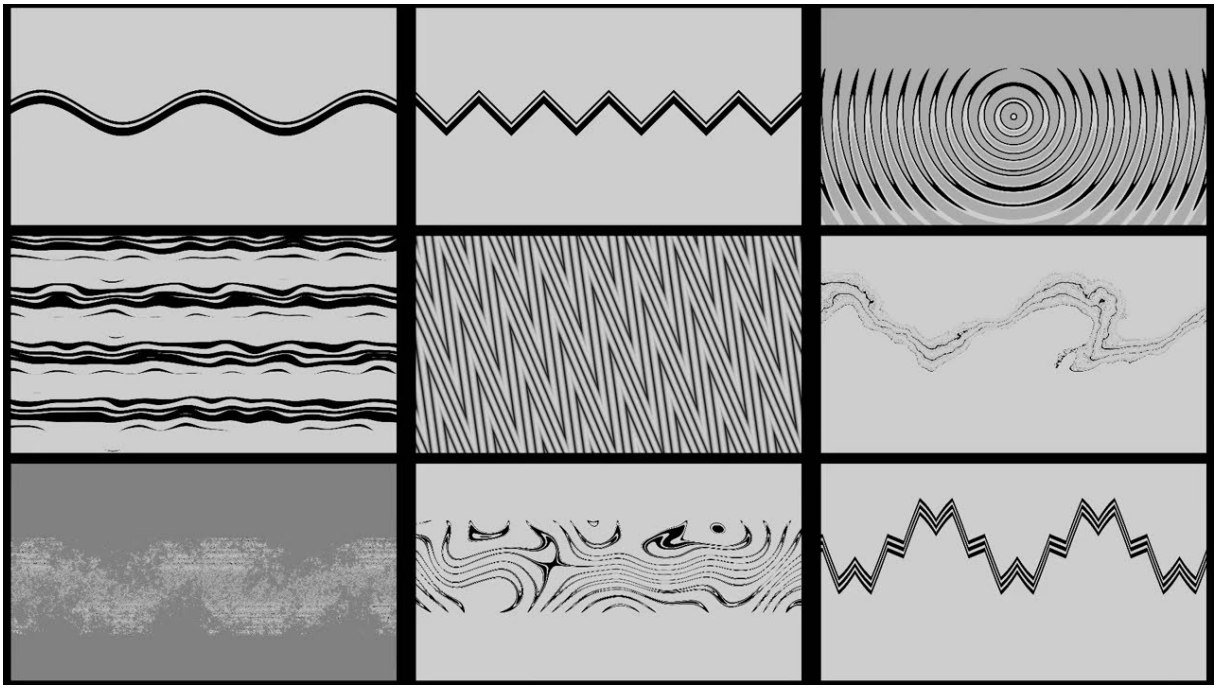

Abbildung 44: Wave Texture – Beispiele
Quelle: Sam Bownman, unter: https://www.youtube.com/watch?app=desktop&v=cBoYiQtxbuQ letzter Zugriff am 13.05.2025

Gradient Texture

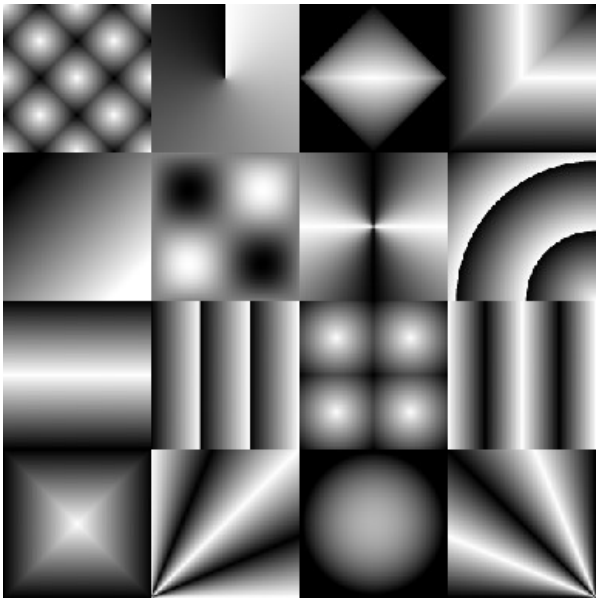

Abbildung 45: Gradient Texture – Beispiele
Quelle: Screaming Brain Studios, unter: https://opengameart.org/content/300-gradient-textures letzter Zugriff am 13.05.2025

Die Gradient Texture basiert auf kontinuierlichen Farbverläufen, die sich entlang definierter Richtungen oder Flächen erstrecken. Sie erzeugt sanfte Übergänge zwischen Farben oder Helligkeitswerten und dient meist als Basis für weitere Texturierungen oder als Überlagerungseffekt. Anders als bei periodischen Mustern wie der Wave Texture entsteht hier kein wiederholendes Muster, sondern ein linear oder radial verlaufender Farb- oder Helligkeitsgradient (siehe Abbildung 45). Die Erzeugung von Gradient Textures basiert häufig auf Gradient Noise, bei dem jedem Gitterpunkt ein zufälliger Richtungsvektor zugewiesen wird. Im Gegensatz zu Value Noise, das feste Werte an Gitterpunkten definiert, werden hier die Zwischenräume durch Interpolation dieser Gradienten gefüllt (vgl. EBERT et al. 2002: 181f.).

Zur Steuerung der Gradient Texture stehen wieder einige Parameter zur Verfügung:

- **Gradient Type** bestimmt die Form und Richtung des Gradientenverlaufs. Typische Varianten sind Linear, Radial, Quadratisch, Spherical und Diagonal (siehe Abbildung 45).
- **Scale** beeinflusst die räumliche Ausdehnung des Gradienten und steuert, wie schnell sich die Werte über eine Fläche verändern.
- **Interpolation** steuert die Art des Übergangs zwischen den Werten des Gradienten. Mögliche Interpolationen sind Linear, Ease oder Constant.

Die Qualität von Gradient Texturen hängt nicht nur von der Erzeugungsmethode ab, sondern auch von der Art der Farbdarstellung im Ausgabesystem. Farbverläufe können unter dem sogenannten Banding-Effekten leiden, wenn die Farbtiefe der Anzeige begrenzt ist. Diese Phänomen zeigt sich als sichtbare Stufen im Farbverlauf. Je nach verwendeter Farbtiefe im

Farbbuffer (z.B. 16-Bit High Color, 24-Bit True Color oder 30+ Bit Deep Color) variiert die Präzision der Farbwiedergabe. Um Banding zu minimieren, werden häufig Techniken wie Dithering eingesetzt, bei denen feines Rauschen hinzugefügt wird, um die Wahrnehmung eines glatteren Verlaufs zu erzeugen (siehe Abbildung 46) (vgl. AKENINE-MÖLLER et al. 2018: 1009ff.; EBERT et al. 2002: 181ff.).

Gradient Texturen finden vielseitige Anwendungen, insbesondere auch im Kontext von 3D-Stadtmodellen und der Gestaltung von Gebäudefassaden. Sie dienen häufig als Masken oder als Übergänge, um die Alterung oder Verschmutzung von Fassen natürlich wirken zu lassen.

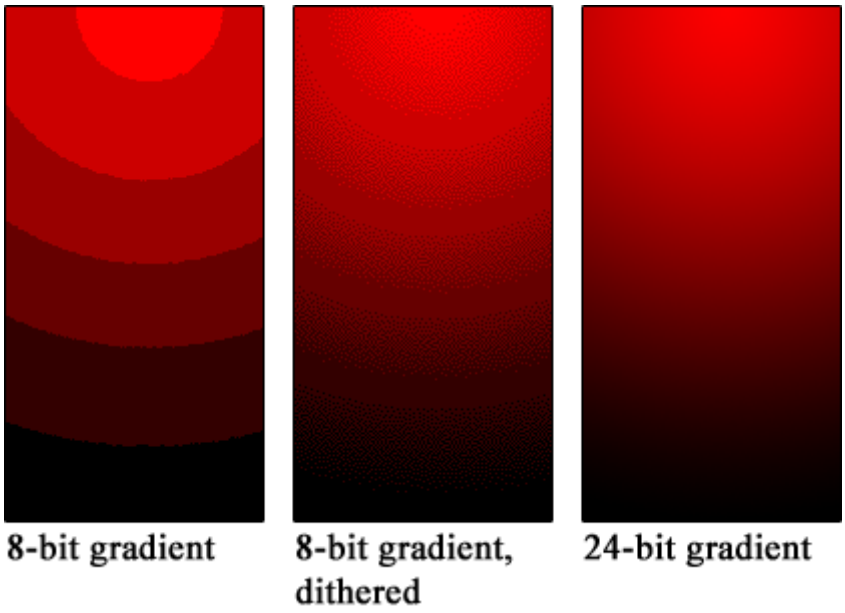

Abbildung 46: Gradient Texture: Banding and Dithering
Quelle: Wikimedia Foundation Inc., unter:
https://en.wikipedia.org/wiki/Colour_banding letzter Zugriff am 13.05.2025

Grid Texture

Die Grid Texture basiert auf der regelmäßigen Anordnung von geometrischen Grundelementen, auch Textons genannt, innerhalb eines kartesischen Gitters. Wie auch Wave und Gradient gehört sie zu den strukturierenden Texturverfahren und eignet sich besonders zur Darstellung regelmäßig aufgebauter Oberflächen wie Fassadenraster. Das zugrundeliegende Prinzip beruht auf der Platzierung sich wiederholender Formen (z.B. Kreise, Ellipsen, Quader) an den Schnittpunkten oder innerhalb der Zellen eines Rasters (vgl. DONG et al. 2020: 7ff.).

Die Erscheinung der Grid Texture kann durch verschiedene Parameter, welche sowohl die geometrischen Eigenschaften der einzelnen Textons als auch deren Anordnung und Kombination, gesteuert werden (vgl. DONG et al. 2020: 7ff.):

- **Texton Shape** bestimmt die Grundform der platzierten Elemente, z.B. Rechteck, Kreis, oder benutzerdefinierte Geometrien.
- **Grid Resolution** legt die Dichte des Rasters fest und beeinflusst damit den Abstand zwischen den Textons.
- **Size (a, b, c)** definiert die Dimensionen der einzelnen Textons entlang der jeweiligen Raumachsen.
- **Orientation** erlaubt die Rotation einzelner Elemente, um eine größere visuelle Vielfalt zu erzeugen.

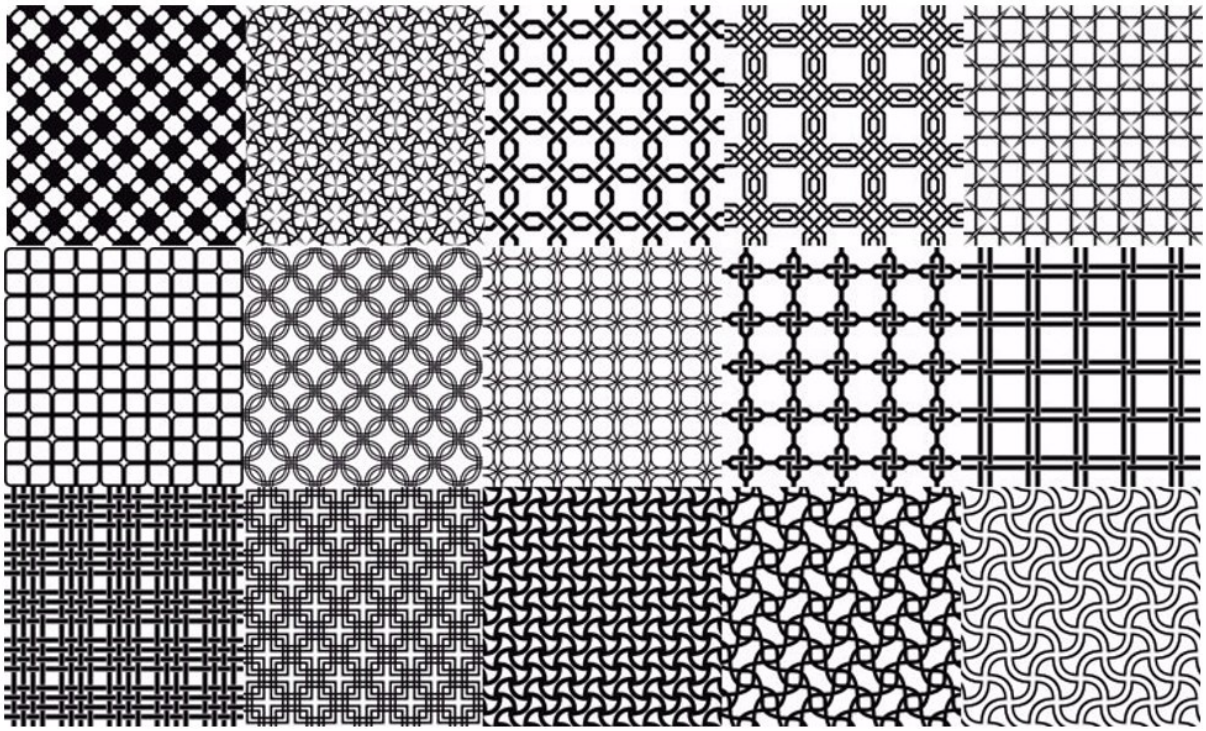

Abbildung 47: Gradient Texture – Beispiele
Quelle: DavidZydd, unter https://designbundles.net/davidzydd/9725-15-seamless-grid-patterns-eps-ai-svg-jpg-5000x5000 letzter Zugriff am 13.05.2025

Eine besondere Variante stellt das Random Grid dar. Hierbei werden die regelmäßig platzierten Textons durch zufällige Versätze gestört. Dadurch wird das starre Raster aufgebrochen und es entsteht ein organischeres, aber dennoch strukturiertes Erscheinungsbild (vgl. DONG et al. 2020: 8f.).

2.2.3. Methoden des Texture Mappings

In der Computergrafik existieren verschiedene Verfahren, um Texturen auf 3D-Modelle zu projizieren. Zu den wichtigsten Methoden zählen unter anderem UV-Mapping, prozedurales Mapping, Triplanar Mapping sowie die fortgeschrittene Parallax Occlusion Mapping Methode.

2.2.3.1. UV-Mapping

Die grundlegenden Prinzipien des UV-Mappings wurden bereits in den Kapiteln 2.2.1.1. und 2.2.1.2. erläutert. Im Folgenden werden diese noch einmal kurz zusammengefasst und darüber hinaus weiterführende Aspekte des UV-Mappings thematisiert.

Das UV-Mapping ist ein Verfahren, bei dem die Oberfläche eines 3D-Objektes in einer zweidimensionalen Darstellung, dem sogenannten UV-Raum, aufgefaltet wird. Dabei werden jedem Punkt der 3D-Geometrie koordinatenbasierte Referenzen (U- und V-Koordinaten) zugewiesen, die es ermöglichen, eine 2D-Textur präzise auf die komplexe Modelloberfläche zu projizieren. Diese Methode ist insbesondere bei komplexen Formen unverzichtbar, da einfache Projektionsverfahren wie planare, zylindrische oder sphärische Projektion oft zu Verzerrungen oder Überlappungen führen.

Eine zentrale und äußerst wirkungsvolle Anwendung des UV-Mappings stellt das sogenannte Texture Baking dar. Hierbei werden prozedural generierte Oberflächeneigenschaften, die normalerweise direkt auf dem 3D-Modell erzeugt werden, auf eine zweidimensionale Textur übertragen, indem für jeden Punkt der Oberfläche über die UV-Koordinaten die exakten Farb- und Materialwerte berechnet und auf einer Textur gespeichert werden (siehe Abbildung 48).

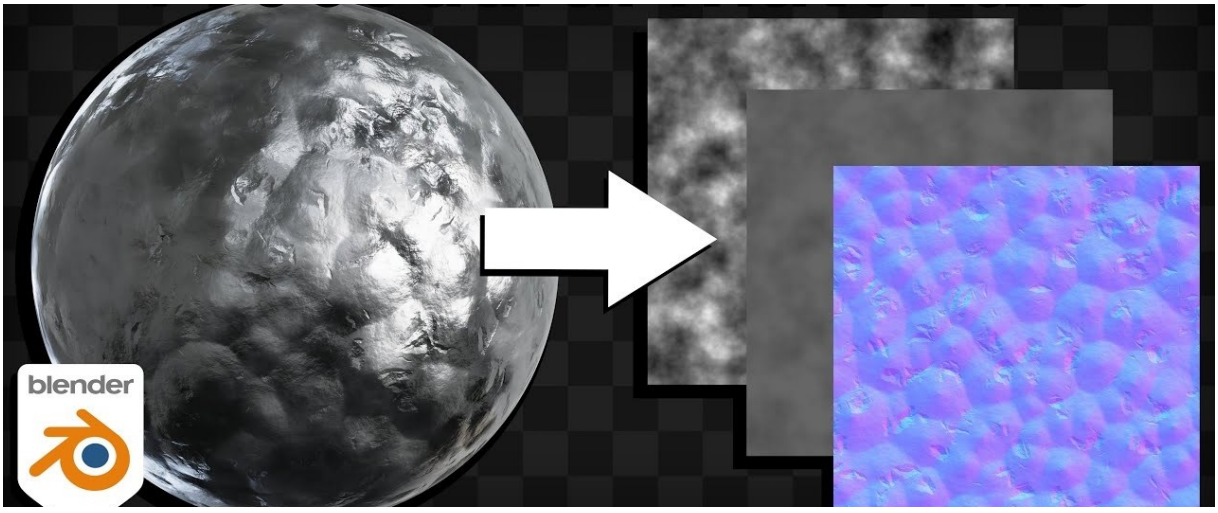

Abbildung 48: Texture Baking – Beispiel
Quelle: Ryan King Art, unter: https://www.youtube.com/watch?v=B2kFeMBBBjc letzter Zugriff am 13.05.2025

So erhält man für genau dieses Modell eine maßgeschneiderte Textur, die die komplexen Details der prozeduralen Shader abbildet. Dieses Verfahren ist besonders effektiv, weil es die Vorteile prozeduraler, dynamischer Texturen mit der Effizienz von statischen 2D-Texturen kombiniert (vgl. EBERT et al. 2002: 197f.).

Trotz der Vorteile ist das Erstellen effizienter und verzerrungsarmer UV-Layouts eine komplexe und zeitaufwendige Aufgabe. Manuelles Unwrapping erfordert Erfahrung, um Nahtstellen, sogenannte Seams, geschickt zu platzieren. Zur Vereinfachung und Verbesserung kommen zunehmend Methoden aus dem Bereich des maschinellen Lernens zum Einsatz, um das UV-Unwrapping zu automatisieren. Ansätze wie Graphseam nutzen neuronale Netze, um UV-Inseln semantisch sinnvoll zu gruppieren und Nahtstellen intelligent zu platzieren (vgl. VERMANDERE et al. 2024: 2).

2.2.3.2. Procedural Mapping

Procedural Mapping beschreibt einen prozessgesteuerten Vorgang, mit dem Prozedurale Texturinhalte auf eine 3D-Oberfläche übertragen werden. Anders als beim UV-Mapping werden hier jedoch keine zweidimensionalen Texturen bzw. Bilddaten verwendet, sondern durch formelbasierte Regeln und Funktionen, die direkt zur Laufzeit ausgewertet werden müssen (vgl. AKENINE-MÖLLER et al. 2018: 198ff.).

Wichtig ist hierbei die Unterscheidung zwischen prozeduralen Texturen und dem Procedural Mapping. Während die prozeduralen Texturen die Inhalte algorithmisch erzeugen, zielt Procedural Mapping auf deren Platzierung und Verteilung auf der Objektoberfläche ab. Das Mapping erfolgt dabei dynamisch und ohne UV-Koordinaten (vgl. AKENINE-MÖLLER et al. 2018: 198ff.; vgl. EBERT et al. 2002: 287ff.).

Um das ganze besser zu verstehen hier ein anschauliches Beispiel: Wird eine prozedurale Holzmaserung vorab in einer 2D-Textur berechnet und anschließend auf ein Objekt projiziert, spricht man vom Mapping einer Prozeduralen Textur. Wird hingegen für jede Position auf der Oberfläche zur Laufzeit bestimmt, welche Maserung dort erscheinen soll, ist dies Procedural Mapping (vgl. AKENINE-MÖLLER et al. 2018: 199.).

Dieses Verfahren bietet mehrere Vorteile. Es entstehen keine Nähte von angrenzenden Texturen oder Verzerrungen. Die Projektion ist auflösungsunabhängig und somit skalierbar

und speichereffizient. Zudem lässt sie sich flexibel an Geometrieänderungen oder dynamische Parameter bei Animationen anpassen. Allerdings gibt es auch Herausforderungen. Dadurch, dass die Textur nach jeder Bewegung neu berechnet werden muss, braucht man für dieses Verfahren eine hohe Rechenleistung. Während es für das klassische UV-Mapping Möglichkeiten gibt, Ladezeiten und Speicherkapazitäten zu reduzieren (siehe Kapitel 2.2.4.) ist das Procedural Mapping hauptsächlich durch die verfügbaren Rechenressourcen limitiert und benötigt vor allem bei großen Projekten wie 3D-Stadtmodellen eine leistungsstarke Hardware (vgl. AKENINE-MÖLLER et al. 2018: 198ff.; vgl. EBERT et al. 2002: 287ff.).

2.2.3.3. Triplanar Mapping

Beim Triplanar Mapping werden die Texturen nicht über vordefinierte UV-Koordinaten auf das Modell projiziert, sondern entlang der drei Raumachsen X, Y und Z gleichzeitig (siehe Abbildung 49). Die Texturprojektion wird dabei dynamisch zur Laufzeit berechnet und ermittelt für jeden Punkt der Oberfläche, wie stark die jeweilige Achsenprojektion zur finalen Texturierung beträgt. Die Grundlage dafür bildet die Orientierung der jeweiligen Normalenvektoren, anhand derer gewichtete Übergänge zwischen den Projektionen entstehen (vgl. WEISS et al. 2020: 1ff.).

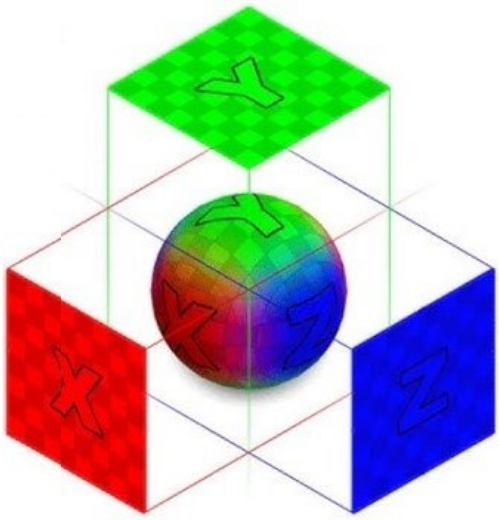

Abbildung 49: Triplanar Mapping
Quelle: mohsen zare, unter: https://www.youtube.com/watch?v=YwnVl2YHXBc letzter Zugriff am 13.05.2025

Triplanar Mapping lässt sich einfach in Shadern implementieren und eignet sich sowohl für klassische Texturen als auch für Normal- und Displacement-Maps. Besonders für unregelmäßige oder steile Geometrien können dadurch Artefakte oder Verzerrungen vermieden werden. Ähnlich wie beim Procedural Mapping spielt auch hier wieder eine leistungsstarke Hardware eine wichtige Rolle. Zudem sind in Bereichen mit stark wechselnden Normalenrichtungen kleinere Projektionsartefakte möglich, wenngleich sie in der Regel deutlich weniger auffallen als bei UV-Mapping Problemen (vgl. WEISS et al. 2020: 3f.).

2.2.3.4. Parallax Mapping

Das Parallax Mapping ist eine fortgeschrittene Technik aus der Computergrafik und dient vor allem dazu bei Oberflächen mit wenig Geometrie einen Eindruck von Tiefe und Unebenheiten zu erzeugen. Bei Normal- oder Bump Maps werden nur die Lichtreflexion auf der Oberfläche verändert und die sichtbaren Unebenheiten bleiben dabei immer an der gleichen Stelle, unabhängig vom Blickwinkel. Parallax Mapping hingegen berücksichtigt die Perspektive des Betrachters und erzeugt so einen dynamischen Effekt, bei dem sich Details je nach Sichtwinkel verschieben (vgl. AKENINE-MÖLLER et al. 2018: 214ff.; KANEKO et al. 2001: 205ff).

Die Grundlage dafür bildet eine Height Map und beim Rendern wird für jeden Bildpunkt der Wert dieser Height Map ausgelesen und dazu verwendet, die Texturkoordinaten zu verschieben. Ziel ist es dabei, die tatsächliche Position auf der Oberfläche zu bestimmen, indem ermittelt wird, wo der Blickvektor die Höhenkarte durchdringt (siehe Abbildung 50). Parallax Mapping nähert dies durch eine erste Approximation an. Es wird die Höhe an der aktuellen Position ausgewertet und basierend darauf, eine neue, angepasste Texturkoordinate berechnet. Dadurch entsteht ein plastischer Eindruck, der das Modell realistischer wirken lässt, ohne die Geometrie zu verändern (vgl. AKENINE-MÖLLER et al. 2018: 215f.).

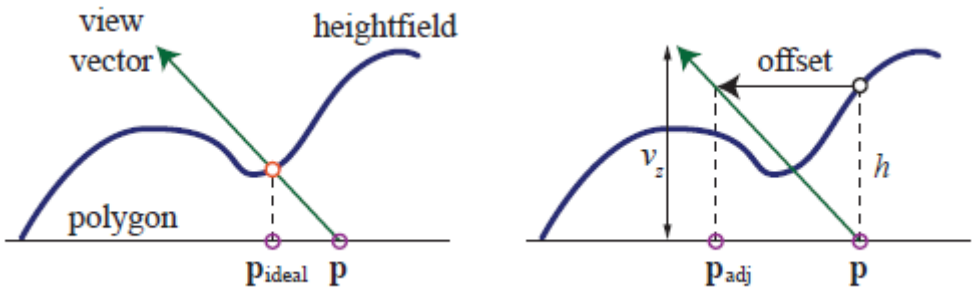

Abbildung 50: Parallax Mapping
Quelle: AKENINE-MÖLLER et al. 2018

Der wesentliche Vorteil dieser Methode liegt darin, dass sie ohne zusätzliche Geometrie auskommt und somit vergleichsweise wenig Rechenleistung benötigt. Bei 3D-Stadtmodellen, insbesondere dem LOD1-Blockmodell lassen sich die Gebäudeoberflächen optisch aufwerten. Weil das Parallax Mapping auf einer Annäherung beruht, wirkt der Effekt schlechter, je steiler der Blickwinkel auf das Objekt ist. Dann können unerwünschte Effekte, wie Flimmern oder Verzerrungen entstehen (vgl. AKENINE-MÖLLER et al. 2018: 214ff.; KANEKO et al. 2001: 206ff).

Parallax Occlusion Mapping

Parallax Occlusion Mapping (POM) stellt eine Erweiterung und präzisere Variante des Parallax Mappings dar. Hierbei wird ein sogenanntes Ray-Marching Verfahren angewandt, um die tatsächliche Schnittstelle zwischen Sichtstrahl und Höhenfeld zu bestimmen. In Abbildung 51 wird der grüne Blickstrahl (Eye Ray) auf die Ebene der Oberfläche projiziert. Entlang dieser Projektion wird die Oberfläche in regelmäßigen Abständen abgetastet, wobei an den violetten Stichprobenpunkten die Höhenwerte ausgelesen werden. Das Verfahren bestimmt dann die erste Stelle, an der der Blickstrahl die approximierten, schwarzen Liniensegmente des Höhenfeldes schneidet. Diese Schnittstelle definiert die sichtbare Position auf der Oberfläche (vgl. AKENINE-MÖLLER et al. 2018: 216f.).

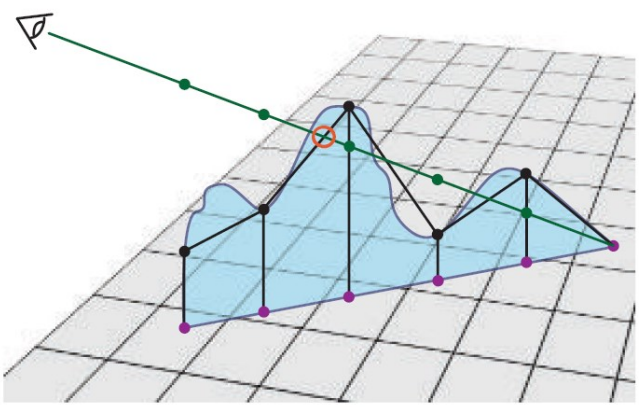

Abbildung 51: Parallax Occlusion Mapping
Quelle: AKENINE-MÖLLER et al. 2018

POM ermöglicht zudem die Simulation von Selbstabschattung, also von Schatten, die Teile einer Oberfläche auf sich selbst werfen, sowie von Überhängen, bei denen vorgelagerte Strukturen dahinterliegende Bereiche verdecken. Weiters kann durch adaptive Abtastraten für unterschiedliche Sichtwinkel die Balance zwischen der Bildqualität und dem Rechenaufwand optimiert werden, denn vor allem bei flachen Winkeln sorgt eine höhere Abtastrate dafür, dass Schnittpunkte nicht übersehen werden. Nachteile des Parallax Occlusion Mappings sind erneut höhere Rechenleistung, da hier mehr Texturzugriffe und komplexere Berechnungen notwendig sind. Abbildung 52 veranschaulicht die visuellen Unterschiede zwischen einer Normal Map, dem Standard Parallax Mapping und dem fortgeschrittenen Parallax Occlusion Mapping (vgl. AKENINE-MÖLLER et al. 2018: 217ff.).

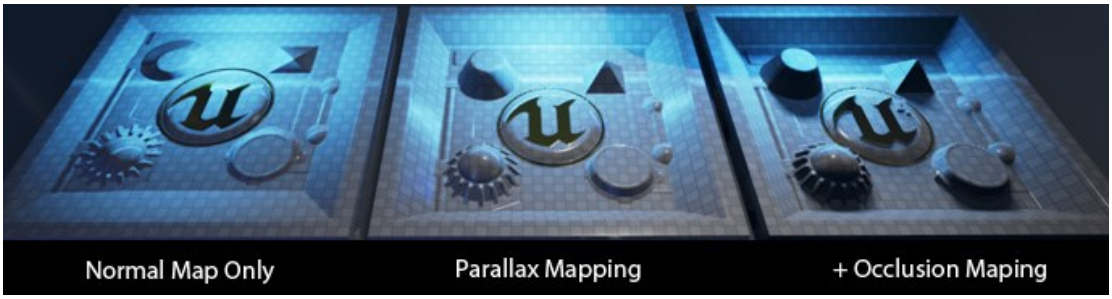

Abbildung 52: Vergleich – Normal Map / Parallax Mapping / POM
Quelle: Ryan Brucks, unter: https://shaderbits.com/blog/curved-surface-parallax-occlusion-mapping letzter Zugriff am 13.05.2025

2.2.4. Optimierungstechniken für Texturenmanagement

In heutigen 3D-Anwendungen reicht es nicht mehr aus, einfach nur Texturen auf Objekte zu bringen. Insbesondere bei großen Projekten stößt man schnell an technische Grenzen, wenn viele verschiedene Texturen auf einmal geladen werden müssen. Hier setzten verschiedene Optimierungstechniken an, die sich vor allem in der Spieleindustrie als Standard etabliert haben, wo möglichst effizient mit Ressourcen gearbeitet werden muss.

Zu den gängigen Methoden zählen unter anderem Texturatlanten, Mipmapping, Texture Streaming oder der Einsatz von Texture Arrays. Die folgenden Abschnitte geben einen Überblick über die relevantesten und zeigen auf, wie sie helfen, die Performance in texturintensiven Szenen zu verbessern.

2.2.4.1. Texturatlas

Bei Texturatlanten werden mehrere Einzeltexturen in einer einzigen großen Textur-Bilddatei zusammengefasst (siehe Abbildung 53). Ziel ist es, insbesondere beim Echtzeitrendering die Anzahl der sogenannten Texture Switches pro Frame zu reduzieren, also der Wechsel von einer aktiven Textur zur nächsten während des Renderprozesses. Das ist vor allem in grafikintensiven Anwendungen wie Spiel-Engines, 3D-GIS-Systemen oder webbasierten Visualisierungen, da jeder Texture Switch einen Performanceaufwand verursacht (vgl. AKENINE-MÖLLER et al. 2018: 190f.; BUCHHOLZ 2006: 9f.).

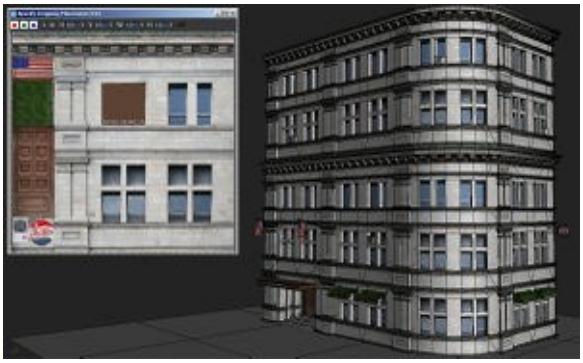

Abbildung 53: Texturatlas – Beispiel
Quelle: Stefan Morrell, unter: http://wiki.polycount.com/wiki/Texture_atlas letzter Zugriff am 13.05.2025

Die grundlegende Funktionsweise eines Texturatlas lässt sich in drei Schritte gliedern: Segmentierung, Parametrisierung und Packing. Dabei ähneln die ersten beiden Schritte stark dem bereits erläuterten UV-Unwrapping (siehe Kapitel 2.2.1.2.), das meist objektbezogen erfolgt, während die Atlas-Erstellung viele verschiedene Texturen in einem gemeinsamen Texturraum verpackt. Zunächst wird das 3D-Modell in einzelne Teilbereiche unterteilt, sogenannten Charts, die in ihrer Topologie einer Scheibe ähneln. Diese Charts werden anschließend im Rahmen der Parametrisierung in den zweidimensionalen Texturraum

überführt. Der dritte Schritt, das Packing, besteht darin, die Charts möglichst effizient anzuordnen, um eine möglichst hohe Atlas Coverage, also ein möglichst geringer Anteil ungenutzter Bildbereiche, zu erhalten (vgl. BUCHHOLZ 2006: 9f.; LÉVY et al. 2002: 362ff.).

Ein Zentrales Problem bei der Verwendung von Texturatlanten tritt auf, wenn eine Textur über eine Fläche hinweg mehrfach wiederholt werden soll, wie es auch beim Tiling von Fassadenstrukturen üblich ist (siehe Kapitel 2.2.1.3.). Normalerweise verwendet man dafür Texturkoordinaten, die über den Bereich von [0,1] hinausgehen, um die Wiederholung zu steuern. Innerhalb eines Atlas funktioniert das jedoch nicht ohne Weiteres, da alle Einzeltexturen gemeinsam in einem Bild gespeichert sind. Sobald Koordinaten diesen Bereich verlassen, können benachbarte, völlig andere Texturen hineinlaufen, was zu fehlerhaften Darstellungen führt. Eine Möglichkeit dieses Problem zu umgehen, ist die mehrfache Ablage derselben Textur nebeneinander im Atlas, um so eine Wiederholung zu simulieren. Alternativ kann man die betroffene Fläche in kleinere Teile unterteilen und jedem Abschnitt eine eigene Textur zuweisen, was allerdings die Modellkomplexität erhöht. Am effizientesten ist der Einsatz eines sogenannten Fragment Shaders, der die Wiederholung der Textur direkt auf der Grafikkarte steuert. Diese Lösung benötigt zwar moderne Shader-Technologie, vermeidet aber zusätzlichen Speicherverbrauch und erhält die Darstellungsqualität (vgl. BUCHHOLZ 2006: 10f.; CARR/HART 2002: 106ff.).

Segmentierung

Die Segmentierung stellt den ersten Schritt bei der Erstellung eines Texturatlas dar. Abbildung 54 zeigt diesen Prozess anhand eines 3D-Modells, bei dem die Segmentierung entlang geometrisch sinnvoller Merkmale erfolgt. Der Vorgang kann entweder manuell durch den Nutzer oder automatisiert mithilfe geeigneter Algorithmen erfolgen. Die bereits erwähnten Charts sollen dabei zwei wesentliche Anforderungen erfüllen (vgl. LÉVY et al. 2002: 366ff.).

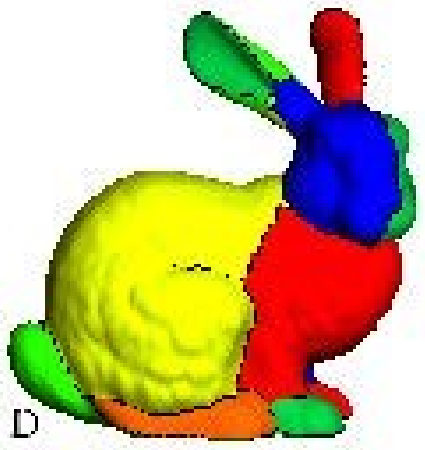

Abbildung 54: Segmentierung – Beispiel
Quelle: LÉVY et al. 2002

- **Positionierung der Grenzen:** Die Grenzen der Charts sollten möglichst in Bereichen hoher Krümmung liegen. In solchen Zonen treten bereits starke Helligkeitsänderungen durch das Shading auf, wodurch potentielle Texturartefakte weniger auffallen.
- **Topologische Eignung:** Jeder Chart muss topologisch einer Scheibe entsprechen, um später möglichst verzerrungsfrei parametrisiert werden zu können.

Zur Umsetzung dieser Anforderungen kommen automatische Verfahren zum Einsatz, die sich an der manuellen Modellzerlegung orientieren. Dabei wird die Geometrie auf Merkmale wie hohe Krümmung hin analysiert, um sogenannte Feature-Kurven zu erkennen. Diese bilden die natürlichen Grenzen zwischen Charts. Anschließend wachsen die Charts iterativ von ausgewählten Startpunkten (Seeds) entlang der Geometrie, bis sie sich an den erkannten Kanten treffen (vgl. LÉVY et al. 2002: 366ff.).

In besonders komplexen Fällen, etwa bei zylindrischen Strukturen oder nicht-konvexen Flächen, können zusätzliche Schnitte notwendig sein, um eine Parametrisierbarkeit zu gewährleisten. In der Praxis wird die resultierende Segmentierung häufig auf Basis geometrischer Kriterien wie Verhältnis von Fläche zu Umfang oder lokaler Verzerrung validiert und bei Bedarf nachjustiert (vgl. ebd.).

Parametrisierung

Die Parametrisierung beschreibt den Übergang von den zuvor segmentierten Charts in den zweidimensionalen Texturraum. Dieser Prozess entspricht im Wesentlichen dem bereits erläuterten UV-Unwrapping (siehe Kapitel 2.2.1.2.). Damit diese Abbildungen erfolgreich gelingt, müssen zwei zentrale Anforderungen erfüllt sein (vgl. LÉVY et al. 2002: 367).

- **Winkel- und Flächentreue:** Die Parametrisierung soll möglichst geringe Verzerrungen aufweisen. Idealerweise bleiben sowohl Winkel als auch Flächenverhältnisse erhalten. In der Praxis ist dies meist ein Kompromiss und je nach anwendungsfall wird entweder Winkel- oder Flächentreue priorisiert.
- **Überlappungsfreiheit:** Die Charts dürfen sich im Texturraum nicht überlappen, da dies zu fehlerhaften Texturierungen führen würde.

Abbildung 55 veranschaulicht typische Herausforderungen der Parametrisierung, wie etwa Überlappung durch selbstschneidende Ränder (A) und zeigt, wie diese durch das Aufteilen eines Charts (B) behoben werden können (vgl. LÉVY et al. 2002: 367).

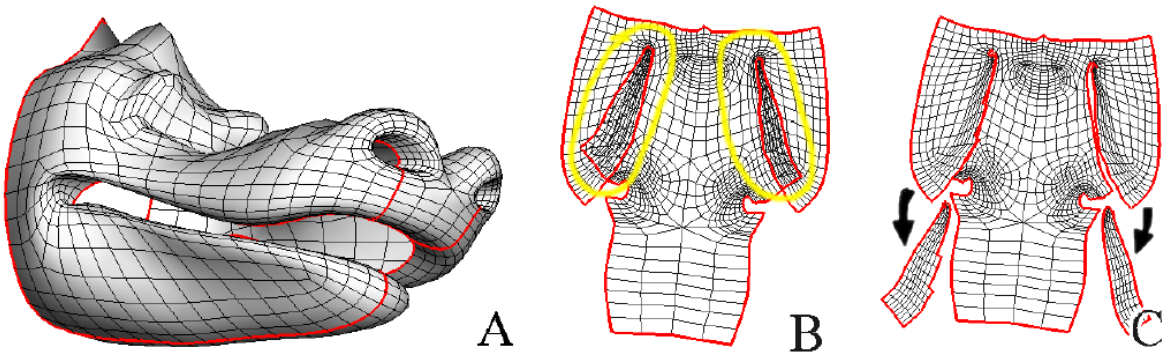

Abbildung 55: Parametrisierung – Beispiel
Quelle: LÉVY et al. 2002

Bei der in LÉVY et al. (2002) beschriebenen Methode erfolgt die Parametrisierung automatisch im Anschluss an das Chart-Growing. Da die Charts jeweils topologisch einer Scheibe entsprechen, können sie unabhängig voneinander parametrisiert werden. Die Qualität der Parametrisierung wird anschließend anhand zweier Kriterien validiert werden (vgl. ebd.).

- **Selbstüberschneidung:** Überlappungen entlang der Ränder können mithilfe von sogenannten Stencil-Tests erkannt werden, bei denen mehrfach belegte Pixel im UV-Raum identifiziert werden. Bei detektierten Selbstschnitten erfolgt eine Korrektur durch das automatische Aufteilen des betroffenen Charts entlang geeigneter Kanten.
- **Flächenverzerrung:** Die Verzerrung wird über das Verhältnis von Modellfläche zu Texturfläche gemessen. Überschreitet dieses Verhältnis einen definierten Schwellenwert, wird das entsprechende Chart unterteilt.

Packing

Im letzten Schritt erfolgt das Packing. Hierbei werden alle parametrisierten Teilbereiche innerhalb eines gemeinsamen Texturbildes so anzuordnen, dass keine Überlappungen auftreten und der verfügbare Platz möglichst effizient genutzt wird (siehe Abbildung 56).

Das zugrunde liegende Problem gehört zur Klasse der sogenannten NP-schweren Optimierungsprobleme. Das bedeutet, dass keine bekannten Algorithmen existieren, die bei wachsender Komplexität des Modells in vertretbarer Zeit immer eine optimale Lösung garantieren können. Aus diesem Grund werden in der Praxis vereinfachte Verfahren

eingesetzt, die eine gute Balance zwischen Laufzeit und Packungsqualität bieten. Eine verbreitete Methode ist, die Charts anhand ihrer begrenzenden Rechtecke zu approximieren und diese zu packen. Allerdings führt diese Näherung häufig zu suboptimaler Raumausnutzung, da die tatsächliche Form der Charts unberücksichtigt bleibt. Eine verbesserte Strategie berücksichtigt direkt die exakten Umrisse der Charts und nutzt einen iterativen Einfügealgorithmus, der auf der Verwaltung eines sogenannten Horizonts basiert. Der Ablauf gestaltet sich dabei folgendermaßen (vgl. BUCHHOLZ 2006: 58; LÉVY et al. 2002: 367ff.):

- **Flächenorientierung:** Alle Charts werden so skaliert, dass ihre Flächen im zweidimensionalen Texturraum der Fläche im 3D-Modell entspricht.
- **Ausrichtung und Sortierung:** Die Charts werden entlang ihrer maximalen vertikalen Ausdehnung ausgerichtet und absteigend nach Größe sortiert.
- **Horizontbasierte Platzierung:** Für jeden Chart werden obere und untere Horizontlinien berechnet, welche seine tatsächliche Form im Texturraum beschreiben. Die Einfügeposition wird so gewählt, dass der Zwischenraum zwischen der unteren Horizontlinie des Charts und dem aktuellen Horizont des Atlas minimiert wird.
- **Aktualisierung des Horizonts:** Nach dem Einfügen eines Charts wird der Horizont des Atlas entsprechend der neuen Geometrie angepasst.

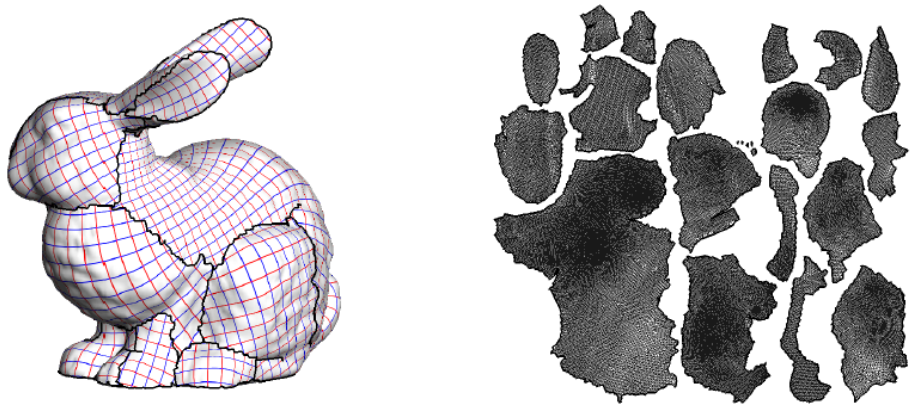

Abbildung 56: Packing – Beispiel
Quelle: LÉVY et al. 2002

Die Horizontlinien werden als diskrete Werte auf Basis der Texturauflösung gespeichert. Zusätzlich wird ein Sicherheitsabstand zwischen den Charts eingefügt, um Mip-Mapping-Artefakte zu vermeiden. So lassen sich auch komplexe Modelle mit vielen nicht-konvexen Charts effizient in einem einzelnen Texturatlas unterbringen LÉVY et al. 2002: 368f.).

2.2.4.2. Mip Mapping

Der Begriff „Mip“ stammt aus dem Lateinischen „multum in parvo“ und bedeutet so viel wie „vieles in kleinem Raum“, was die Funktionsweise dieser Methode bereits gut beschreibt. Mip mapping ist eine weit verbreitete Technik zur Antialiasing-Reduzierung bei Texturen (siehe Kapitel 2.2.1.3.). Hierbei wird die ursprüngliche Textur durch mehrfaches Herunterskalieren in kleinere Versionen überführt, die gemeinsam eine sogenannte Mipmap-Kette bilden (vgl. AKENINE-MÖLLER et al. 2018: 184ff.).

Der Erstellungsprozess einer Mipmap-Kette beginnt mit der Ausgangstextur auf Level 0, deren Auflösung auf jedem weiteren Level jeweils halbiert wird. Dabei wird jeder neue Texel-Wert als Durchschnitt der jeweils vier benachbarten Texel der vorherigen Textur berechnet. Dieses Herunterskalieren erfolgt rekursiv, bis eine Dimension der Textur nur noch einen Texel aufweist. Abbildung 57 veranschaulicht diese pyramidenartige Struktur. Die vertikale Achse stellt dabei die dritte Texturkoordinate d dar, welche nicht linear ist, sondern angibt, welche zwei Texturlevel für die Interpolation eines Samples verwendet werden (vgl. AKENINE-MÖLLER et al. 2018: 186ff.; BUCHHOLZ 2006: 87).

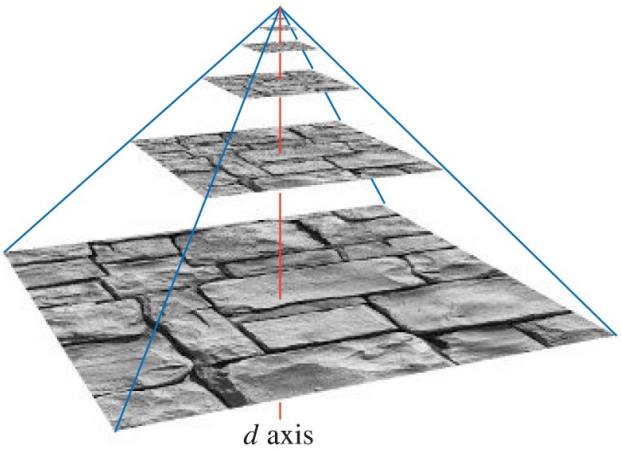

Abbildung 57: Mipmap – Pyramide
Quelle: AKENINE-MÖLLER et al. 2018

Wichtig für die Qualität einer Mipmap ist die Filterung und die Berücksichtigung der sogenannten Gamma-Korrektur, bei der Helligkeitswerte nicht linear, sondern im korrigierten Farbraum gemittelt werden, um Farbverfälschungen zu vermeiden. Besonders relevant ist dies bei Texturen, die in einem nichtlinearen Farbraum wie sRGB kodiert sind. Moderne Grafik-APIs (Application Programming Interface) unterstützen deshalb die Konvertierung in den linearen Raum vor der Mipmap-Erzeugung und speichern die fertigen Mipmaps anschließend wieder im sRGB-Farbraum (vgl. AKENINE-MÖLLER et al. 2018: 185f.).

Während des Renderings wird die geeignete Mipmap-Stufe automatisch durch die Render-Engine bestimmt. Grundlage dafür ist die Berechnung des Level of Detail, der angibt, welcher Mipmap-Level für einen gegebenen Bildschirmpixel verwendet werden soll. Dies wird anhand

der Veränderung der Texturkoordinaten über die Bildschirmpixel berechnet, wobei verschiedene Methoden existieren, beispielsweise die Bestimmung des größten Differentials. Wichtig ist hierbei anzumerken, dass in diesem Kontext die Level of Detail eine andere Bedeutung haben als jene in Kapitel 2.1.3. Die eigentliche Abtastung erfolgt durch trilineare Interpolation. Hier werden die zwei benachbarten Mipmap-Level bilinear abgefragt und anschließend linear zwischen ihnen interpoliert (vgl. AKENINE-MÖLLER et al. 2018: 185f.).

2.2.4.3. Multiresolution-Texturatlas

Ein Multiresolution-Texturatlas (MTA) ist im Grunde eine Kombination aus dem klassischen Texturatlas und der Mipmapping Technik. Ziele wie etwa die Reduktion von Speicherbedarf, Minimierung von Texture Switches und die effiziente Verwaltung von Texturen bleiben gleich, allerdings bietet der MTA zusätzliche Funktionalitäten, die insbesondere in großen Echtzeitanwendungen, oder bei wechselnden Level of Detail Anwendungen zum Tragen kommen. Im Unterschied zum klassischen Texturatlas speichert der MTA für jede enthaltene Textur mehrere Auflösungsstufen in einer gemeinsamen Atlasstruktur, wodurch eine direkte und effiziente Auswahl der passenden Detailstufe ohne zusätzliche Speicherzugriffe oder Texturwechsel ermöglicht wird (vgl. BUCHHOLZ 2006: 73ff; CARR/HART 2002: 118ff.).

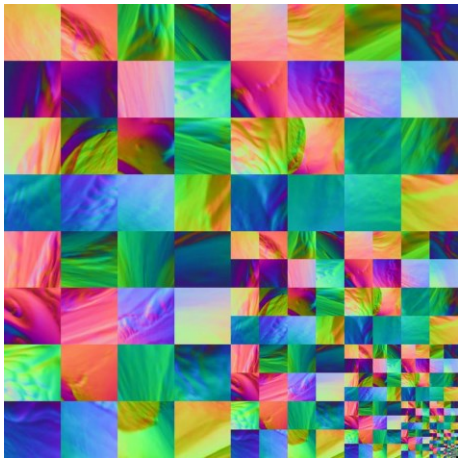

Abbildung 58: Multiresolution Texturatlas – Beispiel
Quelle: PURNOMO et al. 2004

Abbildung 58 zeigt ein Beispiel für einen nahtlosen Texturatlas, in dem mehrere verschiedene Texturen in einem gemeinsamen Mipmap-Muster angeordnet sind. Die Texturen liegen dabei nicht nur in ihrer höchsten Auflösung (Level 0) vor, sondern sind zusätzlich in mehreren sukzessive verkleinerten Auflösungsstufen vorhanden, die kompakt im unteren Quadranten der Atlas Textur organisiert sind (siehe Abbildung 59). Ausgehend des Bildes auf

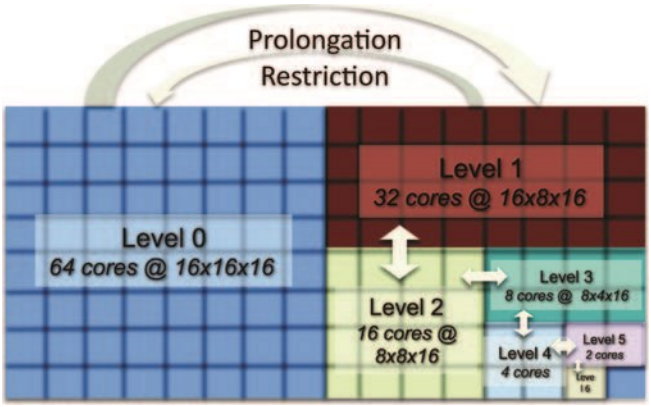

Abbildung 59: Speicherlayout eines MTA im Mipmap-Stil
Quelle: WHENER et al. 2011

Level 0 lassen sich die entsprechenden Koordinaten für die höheren Mipmap-Stufen im Fragment-Shader berechnen (vgl. PURNOMO et al. 2004: 69; WEHNER et al. 2011: 14).

Im Gegensatz zu Ansätzen, die auf ein dynamisches Nachladen einzelner Texturkacheln während der Laufzeit angewiesen sind, setzt der MTA auf eine dauerhaft im Speicher verfügbare, hierarchisch organisierte Texturstruktur. Alle Auflösungsstufen werden direkt im Grafikspeicher gehalten, ohne zeitkritische Streaming-Vorgänge. Das ist insbesondere bei dynamischen Szenen mit schnellen Kamerabewegungen ein wichtiger Vorteil (vgl. BUCHHOLZ 2006: 85f.; PURNOMO et al. 2004: 362ff.).

Trotz der Vorteile eines klassischen MTA-Ansatzes stoßen dessen Strukturen in der Praxis jedoch auch an gewisse Grenzen. Vor allem bei sehr großen oder heterogenen Texturdatenmengen kann die Organisation innerhalb einer einzigen Atlasstruktur zu Ineffizienzen führen. So erfordert etwa das Packing vieler verschiedener Texturen in mehreren Auflösungsebenen in eine einzelne Bilddatei viel Aufwand und sorgfältige Planung. Zudem erschwert die starre Gesamtstruktur das nachträgliche Hinzufügen oder Austauschen einzelner Texturblöcke (vgl. BUCHHOLZ 2006: 86f.; ZHANG et al. 2021: 1ff.).

Als Alternative zum Multiresolution-Texturatlas, bei dem sämtliche Auflösungsstufen innerhalb einer gemeinsamen Atlasstruktur gespeichert werden, existieren auch Ansätze, die diese Trennung explizit über mehrere eigenständige Atlasdateien hinweg organisieren. Die Grundlage dieses Ansatzes ist eine adaptive Texturverwaltung und diese modulare Aufteilung erlaubt eine gezieltere Kontrolle der Speicherressourcen und eine bessere Anpassung an Hardware Limitierungen. Niedrigauflösende Atlanten können frühzeitig geladen werden, während hochauflösende Inhalte nur bei Bedarf nachgeladen werden. Diese Flexibilität verbessert die Handhabbarkeit bei Erweiterungen und Updates. Die erhöhte Anzahl an Texture Switches, da nicht mehr nur auf einen einzelnen Atlas zurückgegriffen wird, ist im Vergleich zum erheblichen Mehraufwand der Erstellung eines komplexen MTA, vernachlässigbar (vgl. CARR/HART 2002: 118ff.; ZHANG et al. 2021: 3ff.).

3. Praktische Umsetzung

Die zuvor behandelten Kapitel zu 3D-Stadtmodellen und Texturierung liefern die theoretische Grundlage für die praktische Umsetzung. Sie zeigen die relevanten Konzepte und Methoden auf, die für die Erstellung realistischer und effizient handhabbarer 3D-Stadtmodelle notwendig sind. Auf dieser Basis wird im praktischen Teil der Arbeit ein Ansatz entwickelt, der diese Erkenntnisse zusammenführt und in einem eigenen Workflow erprobt.

3.1. Struktureller Überblick

Ziel ist es, das Konzept eines Multiresolution-Texturatlas auf Basis von prozedural erzeugten Fassadentexturen umzusetzen und hinsichtlich der Effizienz zu evaluieren. Dabei wird nicht die klassische Methode verwendet, bei der alle Auflösungsstufen in einem einzigen, hierarchisch aufgebauten Atlas organisiert sind. Stattdessen wird der zuvor bereits erwähnte modulare Ansatz umgesetzt. Für jede Auflösungsstufe wird ein eigener Texturatlas erstellt. Grund für die Entscheidung dieser Methode ist vor allem der modulare Aspekt. So können je nach Belieben weitere Auflösungsstufen hinzugefügt oder entfernt werden, ohne bestehende Strukturen wesentlich anzupassen. Darüber hinaus ermöglicht dieser Ansatz einen klar nachvollziehbaren und übertragbaren Workflow, der von Dritten übernommen und in eigene Projekte integriert werden kann. Dieses Ziel der Wiederverwendbarkeit und Skalierbarkeit spiegelt sich auch in der Art und Weise wider, wie die prozeduralen Texturen erstellt werden.

In den folgenden Kapiteln werden zunächst die verwendeten Programme eingeführt und kurz auf ihren Einsatz im Rahmen der Arbeit hingewiesen. Im Anschluss werden die wichtigsten Arbeitsschritte des gesamten Workflows näher beschrieben. Weiters werden die resultierenden Ergebnisse präsentiert und evaluiert.

3.1.1. Verwendete Programme

Für die praktische Umsetzung dieser Arbeit sind Programme erforderlich, die bestimmte grundlegende Anforderungen erfüllen. Dazu zählen die effiziente Erstellung prozeduraler Texturen, die Modellierung und Visualisierung von 3D-Szenen sowie die Möglichkeit, Bilddateien in unterschiedlichen Auflösungen zusammenzuführen. Auf Grundlage dieser

Kriterien werden drei Programme ausgewählt, die im Folgenden beschrieben und hinsichtlich der getroffenen Entscheidung sowie möglicher Alternativen erläutert werden.

3.1.1.1. Substance Designer

Substance Designer ist ein spezialisiertes, kostenpflichtiges Programm zur prozeduralen Erstellung von Materialien und Texturen und wird hauptsächlich in der Spieleentwicklung und Filmproduktion eingesetzt. Entwickelt wurde das Programm ursprünglich von der Firma Allegorithmic und seit 2019 gehört es zu Adobe. Es ist Teil der Substance 3D-Reihe, zu der auch

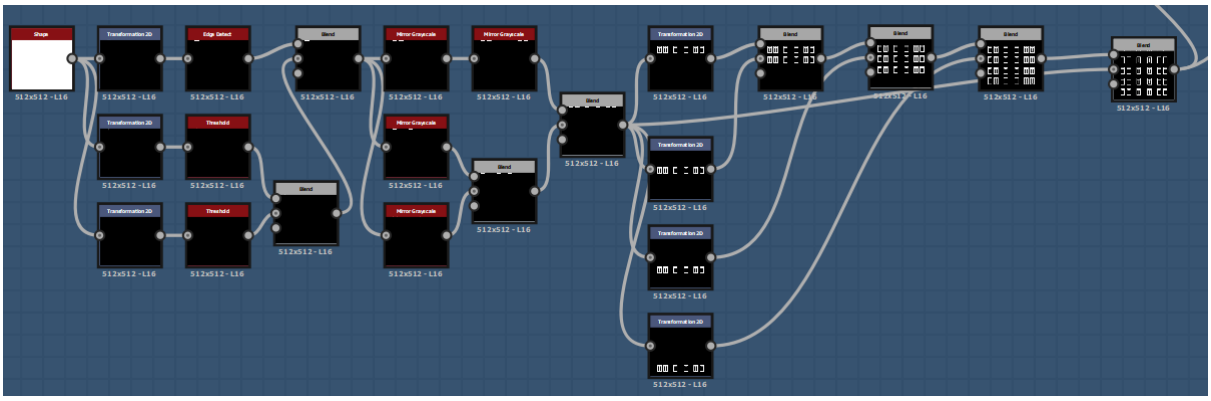

Abbildung 60: Graphenbasierter Workflow – Beispiel
Quelle: Eigene Aufnahme, 21.06.2025

weitere Programme wie Substance Painter, Substance Sampler sowie Substance Modeler gehören, welche alle unterschiedliche Schwerpunkte im 3D-Bereich haben.

Substance Designer arbeitet auf Basis eines nicht-destruktiven, graphenbasierten Workflows. Das bedeutet, dass einzelne Abschnitte in Form von sogenannten Nodes miteinander verknüpft werden (siehe Abbildung 60), wodurch sich komplexe Materialdefinitionen modular und nachvollziehbar aufbauen lassen. Der nicht-destruktive Ansatz bedeutet, dass jederzeit auf vorherige Arbeitsschritte zurückgegriffen und diese verändert werden können, ohne den gesamten Erstellungsprozess erneut durchführen zu müssen. Dadurch wird ein möglichst flexibler und effizienter Workflow möglich.

Im Rahmen dieser Arbeit wird Substance Designer verwendet, um die Fassadentexturen für den Texturatlas zu erstellen. Dabei werden die im Kapitel 2.2.2.2. bereits erläuterten prozeduralen Texturen eingesetzt und über die Node-Trees miteinander kombiniert und variiert. Alternativ zu Substance Designer würde sich auch Blender anbieten, das ebenfalls grundlegende Funktionen zur prozeduralen Texturerstellung bereitstellt. Allerdings ist Blender eher als Allrounder Programm ausgelegt und bietet im Bereich der Texturerstellung

weniger spezialisierte Werkzeuge. Substance Designer erweist sich daher aufgrund seiner Spezialisierung als die geeignetere Lösung.

3.1.1.2. Blender

Blender ist eine kostenlose Open-Source 3D-Grafiksoftware, die ein breites Spektrum an Funktionen rund um die Erstellung, Bearbeitung und Visualisierung von 3D-Inhalten bietet. Es wurde von der Blender Foundation entwickelt und obwohl es frei verfügbar ist, gehört es zu den leistungsfähigsten Tools im Bereich der digitalen 3D-Produktion und wird sowohl im professionellen als auch im akademischen Umfeld eingesetzt.

Blender deckt zahlreiche Anwendungsbereiche ab, darunter Modellierung, Scultping, Rigging, Animation, Simulation, Texturierung, Rendering und Compositing. Zusätzlich bietet es eine flexible Programmierschnittstelle, über die Entwickler eigene Erweiterungen und Werkzeuge in Form von Add-ons und Skripten erstellen können.

Für den praktischen Teil dieser Arbeit wird Blender eingesetzt, um eine vereinfachte städtische Testszene zu erstellen und die Texturatlanten hinsichtlich ihrer Renderzeiten zu testen. Als Alternative bietet sich hier auch die Unreal Engine an, die eine ähnlich leistungsfähige Umgebung für 3D-Szenen bereitstellt. In Bezug auf die Funktionalität bestehen keine gravierenden Unterschiede zu Blender. Ausschlaggebend für die Entscheidung ist vielmehr die bereits bestehende Erfahrung im Umgang mit Blender, wodurch ein zusätzlicher Einarbeitungsaufwand in ein neues Programm vermieden wird.

3.1.1.3. GIMP

GIMP (GNU Image Manipulation Programm) ist ein kostenfreies Open-Source Bildbearbeitungsprogramm und gilt als eine der bekanntesten Alternativen zu kommerzieller Software wie Adobe Photoshop. Die Software bietet unter anderem Werkzeuge für Image Editing, Layer-Bearbeitung sowie Farbkorrektur.

Im Rahmen dieser Arbeit wird GIMP verwendet, um die Texturatlanten zu erstellen. Hierfür werden die generierten Fassadentexturen in einer 2048x2048 Auflösung innerhalb einer 8K-Bilddatei nebeneinander zusammengefügt. Diese Vorgehensweise wird für sämtliche Texturtypen, sowie in drei verschiedenen Auflösungsstufen durchgeführt.

Eine Alternative stellt Adobe Illustrator dar, das ebenfalls leistungsfähige Werkzeuge für Bildbearbeitung und Layout bietet. Allerdings ist Illustrator kostenpflichtig und in diesem Kontext nicht zwingend erforderlich. Da die Arbeitsschritte für die Erstellung der Texturatlanten vergleichsweise einfach sind und keine tiefergehenden Spezialfunktionen benötigt werden, erweist sich die kostenfreie Open-Source-Variante als die praktikablere Lösung.

3.2. Workflow

Abbildung 61 gibt einen kompakten Überblick über den gesamten Workflow. Er umfasst die Sammlung von Referenzmaterial, die Erstellung der Fassadentexturen in Substance Designer, die Zusammenführung zu Texturatlanten in GIMP und die abschließende Projektion auf das 3D-Stadtmodell in Blender. Die einzelnen Schritte werden in den folgenden Kapiteln detailliert erläutert.

Zudem wird ein Testsetting in Blender konzipiert. Hierfür wird ein kleiner Ausschnitt der Stadt Wien gewählt, da die Referenzbilder vor Ort verfügbar sind. Ziel der Szene ist es, die Texturatlanten in verschiedenen Zoomstufen zu präsentieren und ihre Effizienz anhand von Renderzeiten zu vergleichen. Für die Geometrie wird ein LoD1-Modell der Stadt Wien verwendet, welches über das Blender-OSM Add-on geladen wird.

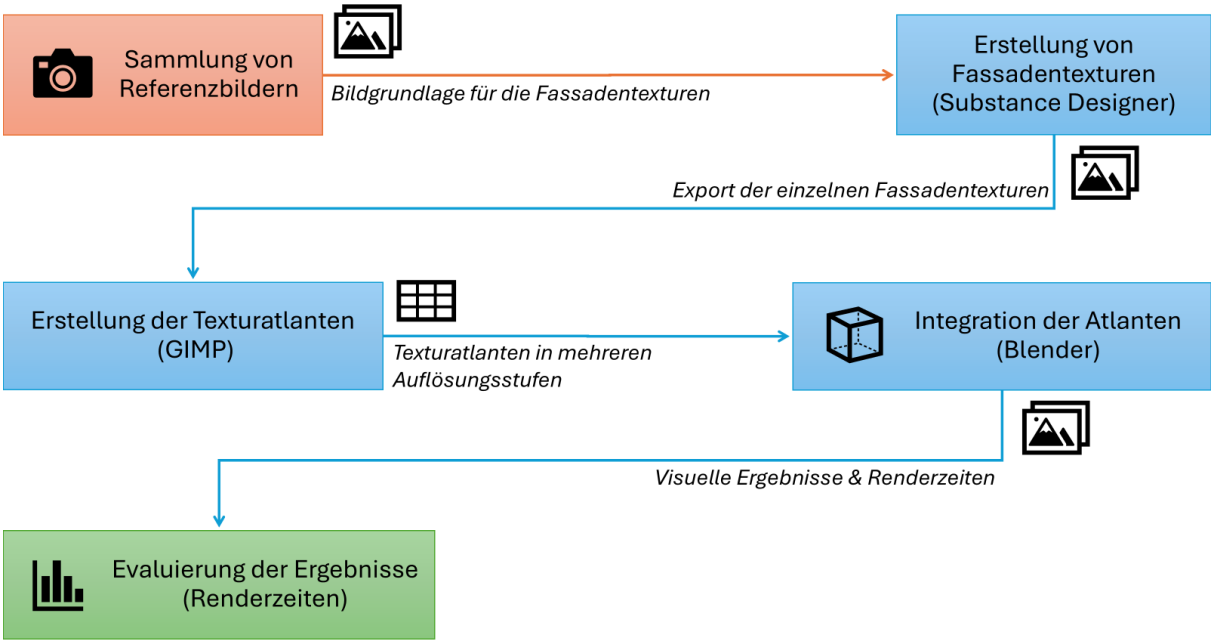

Abbildung 61: Übersicht des Workflows
Quelle: Eigene Darstellung 10.08.2025

3.2.1. Sammlung von Referenzbildern

Das Erstellen von prozeduralen Texturen in Substance Designer ist eine Kombination aus technischen und künstlerischen Prozessen. Die Struktur der Textur wird zwar durch algorithmische Abläufe erzeugt, die Auswahl von Formen, Farben und Details unterliegen jedoch einer gestalterischen Entscheidung. Aus diesem Grund werden vor dem eigentlichen Erstellen der Texturen Referenzbilder gesammelt, die als Vorlage dienen, um realistische Fassadentexturen zu erzeugen. Die Bilder werden im Stadtgebiet von Wien aufgenommen und zeigen typische städtische Fassaden (siehe Abbildung 62).

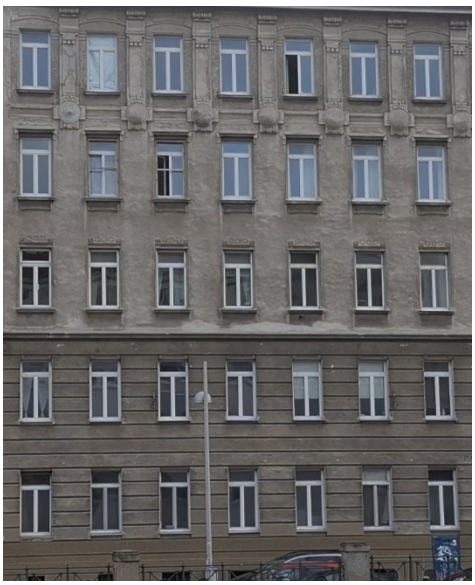

Abbildung 62: Referenzbild – 1
Quelle: Eigene Aufnahme, 17.02.2025

Wichtig bei der Auswahl der Referenzbilder ist, dass sich die Fassaden optisch voneinander unterscheiden und eine gewisse Vielfalt aufweisen, aber gleichzeitig keine zu komplexen Strukturen aufweisen. Die Referenzbilder dienen vor allem als kreative Inspirationsquelle, weshalb technische Aspekte wie eine frontale Aufnahme oder konstante Lichtverhältnisse weniger wichtig sind. Die Bilder wurden größtenteils aus einer Straßenperspektive aufgenommen, wodurch perspektivische Verzerrungen und Schatten entstehen können, was aber für den Erstellungsprozess später nicht relevant ist.

Insgesamt werden 18 verschiedene Referenzbilder gesammelt, aus denen später acht unterschiedliche Fassaden mit jeweils zwei Farbvarianten entwickelt werden (siehe Abbildung 63). Teilweise werden auch mehrere Referenzen kombiniert oder einzelne Elemente frei interpretiert, um größere Vielfalt zu ermöglichen.

Abbildung 63: Referenzbilder
Quelle: Eigene Aufnahme, 17.02.2025

3.2.2. Erstellung der Fassadentexturen

Neben der Erstellung der Fassadentexturen spielt auch die Wiederverwendbarkeit und Erweiterbarkeit eine zentrale Rolle. Deshalb wird die Erstellung der Texturen modular aufgebaut, das heißt in einzelne, leicht austauschbare und veränderbare Blöcke unterteilt.

Ein Beispiel dafür sind die horizontalen Querbalken, die Gebäudefassaden optisch in verschiedene Stockwerke unterteilen (siehe Abbildung 64). Hierfür werden die separaten Module in Form einer „Fundament-Linie“, „1-Stock-Linie“ sowie einer „Dach-Linie“ (siehe Abbildung 65). Um weitere Stockwerke hinzuzufügen, kann der Block der 1-Stock-Linie einfach kopiert und vertikal nach oben oder unten verschoben werden.

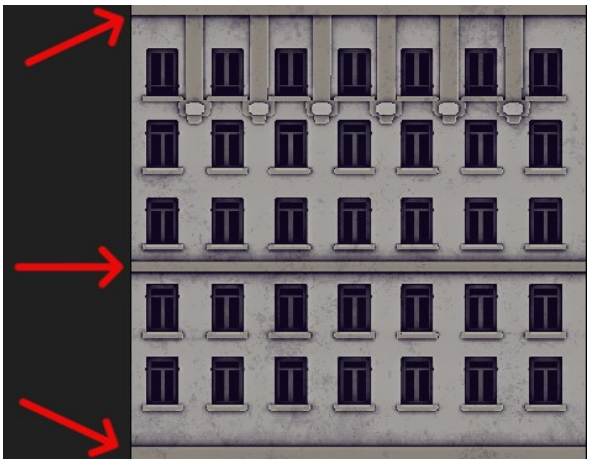

Abbildung 64: Textur-Modul – Beispiel
Quelle: Eigene Aufnahme, 21.06.2025

In den folgenden Kapiteln werden die einzelnen Arbeitsschritte in Substance Designer, sowie die verwendeten prozeduralen Texturen, die als Grundlage dienen, beschrieben. Die Gliederung erfolgt dabei entlang der modularen Struktur, das heißt, die einzelnen Modulblöcke werden jeweils separat erläutert.

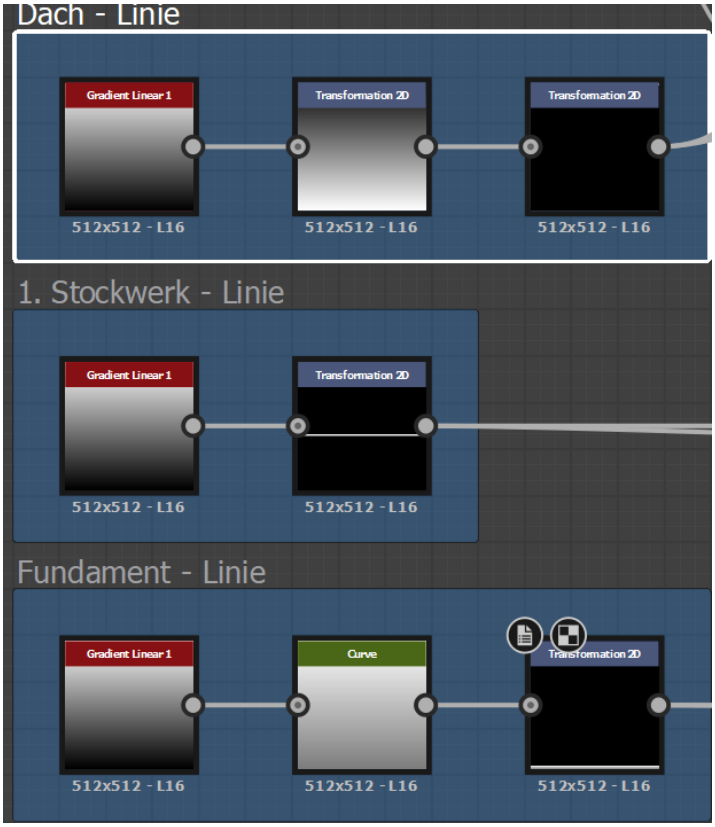

Abbildung 65: Modulblock – Stockwerke
Quelle: Eigene Aufnahme, 21.06.2025

3.2.2.1. Fenster-Modul

Fenster – Basis

Die Fenster-Basis bildet die Grundstruktur der Fensteröffnungen innerhalb der Fassade. Für die Erstellung wird zunächst über eine Shape-Map ein einfaches rechteckiges Element verwendet, das die Grundform eines Fensters repräsentiert (siehe Abbildung 66). Dieses

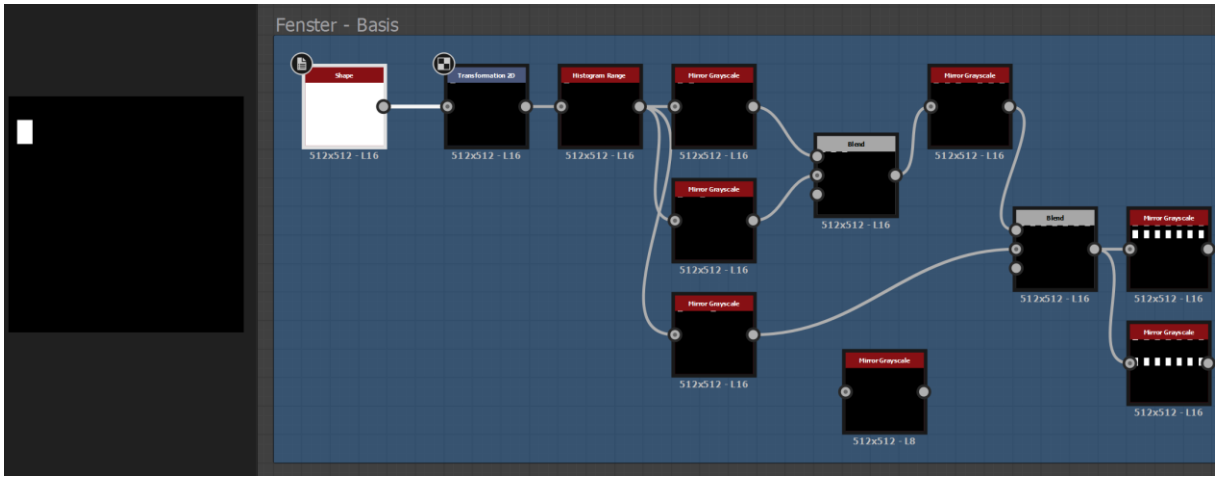

Abbildung 66: Fenster – Basis Modul (rechts), Vorschau der Textur (links)
Quelle: Eigene Aufnahme, 22.06.2025

Rechteck wird anschließend in seiner Größe und Position angepasst, sodass die Proportionen zu einer typischen Fassadenstruktur passen. Um eine gleichmäßige Fensteranordnung über die gesamte Fassadenfläche zu erzielen, wird das Rechteck mithilfe von Mirror-Nodes mehrfach gespiegelt und die separaten Reihen über eine Blend-Node miteinander kombiniert. Das Ergebnis ist eine Schwarz-Weiß Textur, in der die weißen Bereiche die Position der Fensteröffnungen markieren (siehe Abbildung 67).

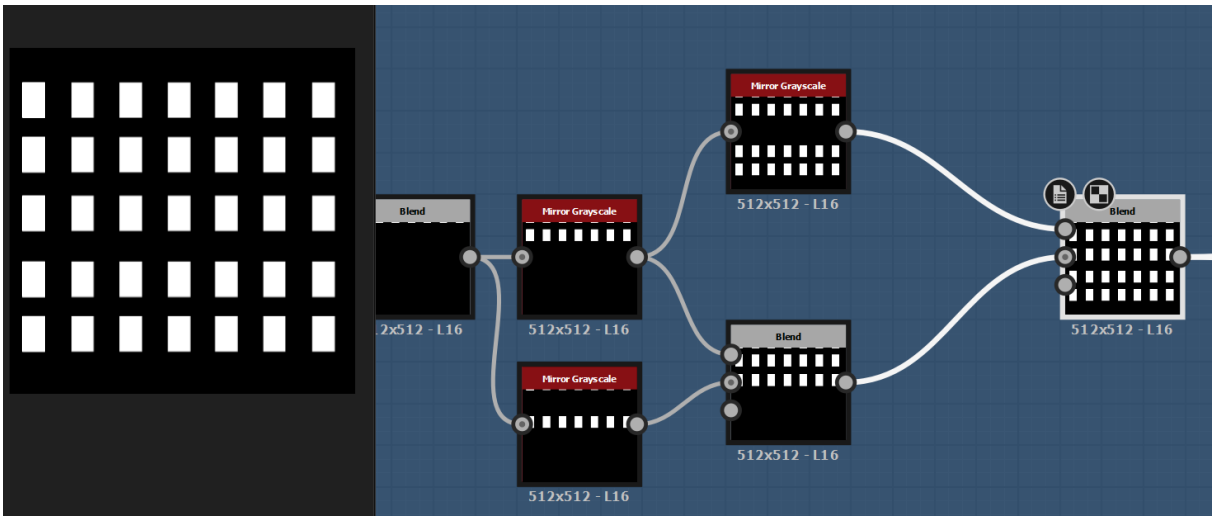

Abbildung 67: Ergebnis – Fenster-Basis
Quelle: Eigene Aufnahme, 21.06.2025

Im Anschluss durchläuft die Textur die allgemeinen Verarbeitungsschritte, die auch bei allen anderen Modulen durchgeführt werden (siehe Abbildung 68). Zunächst wird die Textur in eine Histogram-Range-Node weitergeleitet, in der die Graustufenwerte angepasst werden, um die Höhen- bzw. Tiefenausbuchtungen der Struktur im späteren Material zu steuern. Da es sich bei den Fenstern um Vertiefungen handelt, wird die Textur vor diesem Schritt invertiert. Im nächsten Block werden die angepassten Texturen in einer Height-Blend-Node mit den anderen Modulen kombiniert und in die Finale Height-Map weitergeleitet.

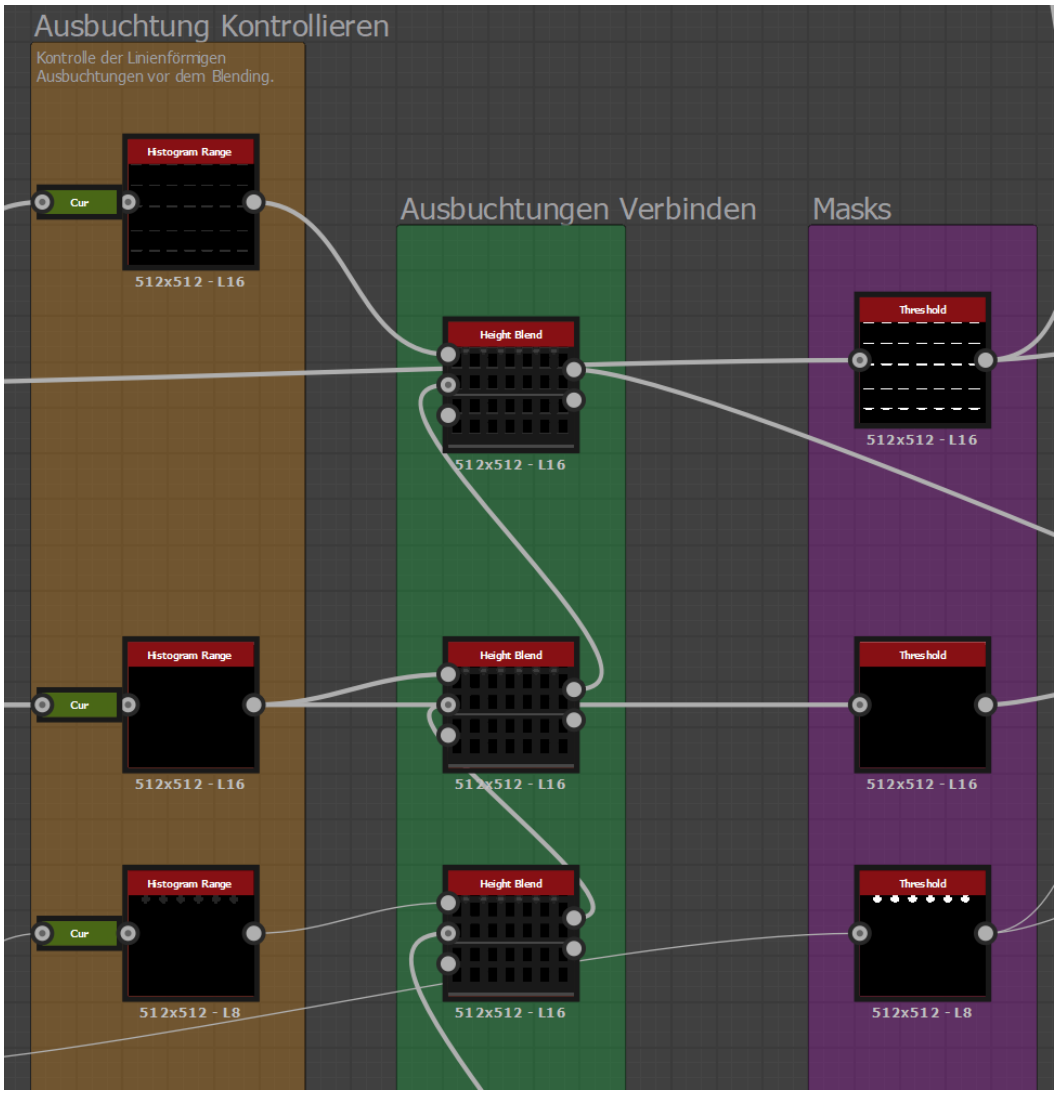

Abbildung 68: Allgemeine Verarbeitungsschritte
Quelle: Eigene Aufnahme, 23.06.2025

Zusätzlich wird die invertierte Schwarz-Weiß-Textur als sogenannte Maske exportiert. Diese Maske dient als Grundlage für die nachfolgenden Materialdefinitionen wie Color-, Roughness- und Normal-Map. Durch diese Trennung bleibt der Workflow konstant, sodass Anpassungen an der Fensterstruktur direkt in allen nachfolgenden Materialebenen übernommen werden.

Fenster – Rahmen

Der Fensterrahmen-Block ergänzt die Fenster-Basis um die Rahmen- und Strebenstrukturen, welche die Fensterflächen optisch unterteilen und einfassen. Wie bei der Fenster-Basis werden auch hier einfache Formen als Grundlage genutzt. Shape-Nodes werden entsprechend skaliert und mithilfe von Mirror- und Blend-Nodes zu einem regelmäßig angeordneten Gittermuster zusammengefasst (siehe Abbildung 69).

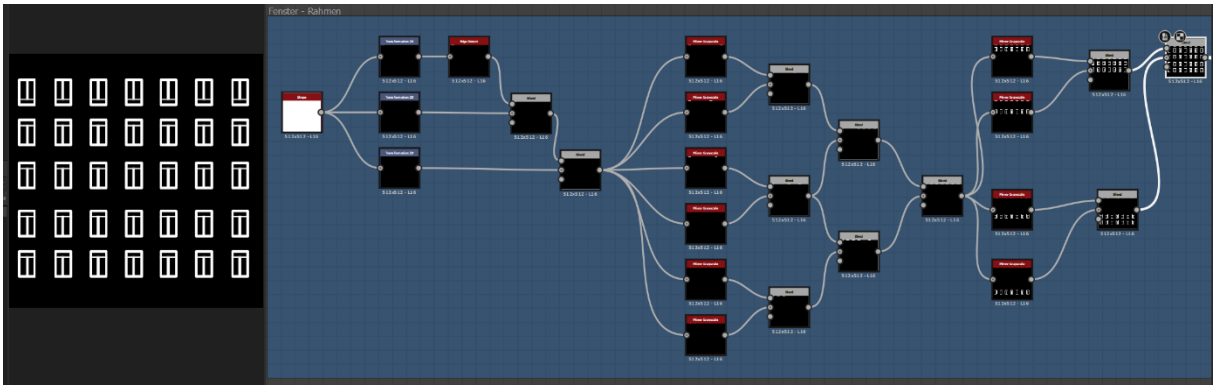

Abbildung 69: Fenster-Rahmen Modul
Quelle: Eigene Aufnahme, 23.06.2025

Die Trennung von Fenster-Rahmen und Fenster-Basis erfolgt nicht nur aus strukturellen, sondern auch aus materialtechnischen Gründen. Während die Fenster-Basis die Glasflächen darstellt und daher eine glatte und stark reflektierende Oberfläche erhält, werden die Fensterrahmen mit einer matteren, leicht rauen Oberfläche versehen, um den typischen Materialmix aus lackiertem Metall oder behandeltem Holz realistisch abzubilden. Durch diese Aufteilung können die Materialeigenschaften beider Elemente unabhängig voneinander angepasst werden.

Fensterbänke

Der Block für die Fensterbänke stellt das dritte und letzte Element des Fenster-Moduls dar. Auch hier wird als Ausgangspunkt wieder eine einfache Shape-Node verwendet und in weiterer Folge korrekt skaliert und positioniert. Anders als beim Fensterrahmen-Block dient die Aufteilung hier nicht hauptsächlich den Materialeigenschaften, sondern zur gezielten Steuerung der Höhenwerte über die Height-Map. So lässt sich genau festlegen, wie stark die Fensterbank aus der Fassadenfläche hervorgehoben wird.

3.2.2.2. Stockwerk-Linien – Modul

Wie bereits im Kapitel 3.2.2. beispielhaft erläutert, bilden die horizontalen Fassadenlinien ein zentrales Element der modularen Texturstruktur. Sie dienen dazu, die einzelnen Stockwerke voneinander zu trennen und der Fassade eine klare Gliederung zu geben.

Für die Umsetzung wird eine Gradient-Map verwendet, die in Größe und Position angepasst wird (siehe Abbildung 70). Im Gegensatz zu einer Shape-Node, die klare, scharfkantige Formen erzeugt, ermöglicht die Gradient-Map weichere Kantenverläufe, was in diesem Anwendungsfall zu besseren Ergebnissen führt. Ein weiterer Grund liegt in der besseren Kachelbarkeit der Textur. Während harte Kanten beim wiederholten Tiling der Textur schnell zu sichtbaren Übergängen führen würden, ermöglicht der weiche Verlauf der Gradient-Map nahtlose Wiederholung auf der Fassadenfläche.

Je nach gewünschtem architektonischem Stil können die einzelnen Linien in ihrer Breite und Höhe angepasst oder weitere Stockwerk-Linien hinzugefügt werden, indem das Modul dupliziert und vertikal verschoben wird.

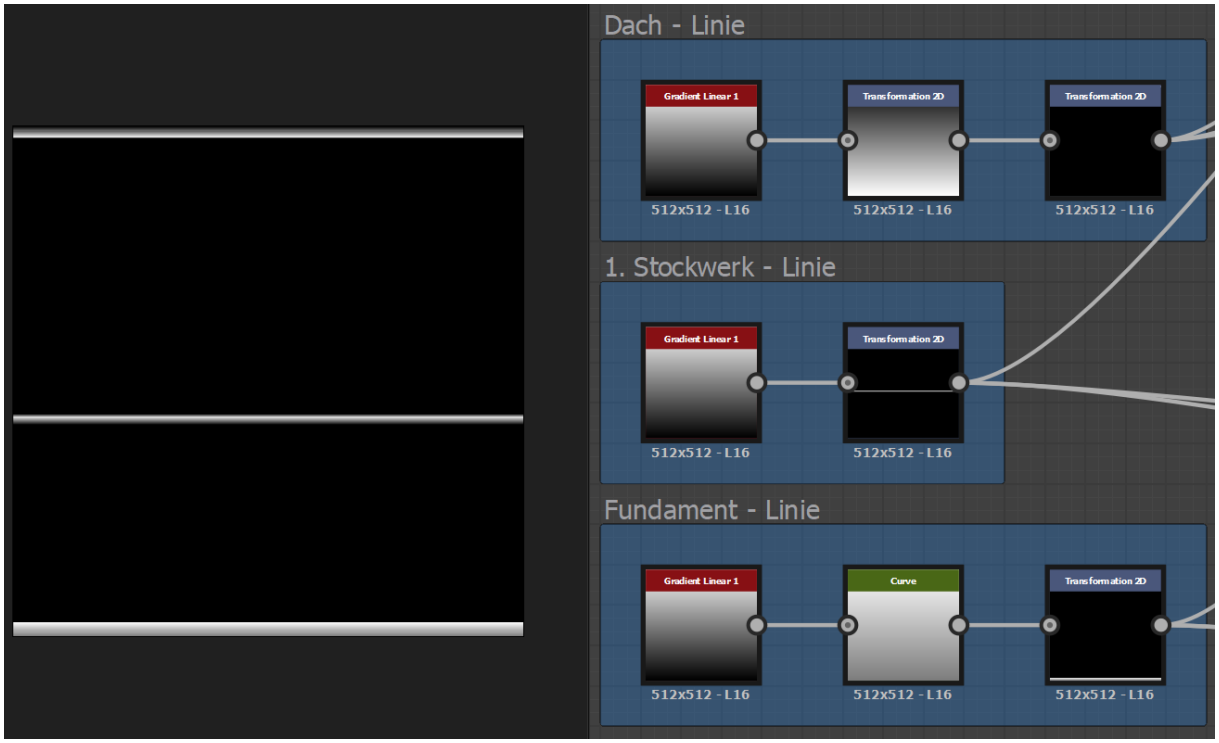

Abbildung 70: Stockwerk-Linien – Modul
Quelle: Eigene Aufnahme, 23.06.2025

3.2.2.3. Verzierungs – Modul

Neben den zuvor beschriebenen Modulblöcken, die bei allen Fassaden in ähnlicher Form vorkommen, stellt das Verzierungs-Modul einen Block dar, der je nach Fassadentyp individuell angepasst oder vollständig weggelassen wird. Hier werden verschiedene dekorative Strukturen wie Säulen, geometrische Ornamente wie Kreise oder Quadrate oder andere gestalterische Elemente integriert werden (siehe Abbildung 71).

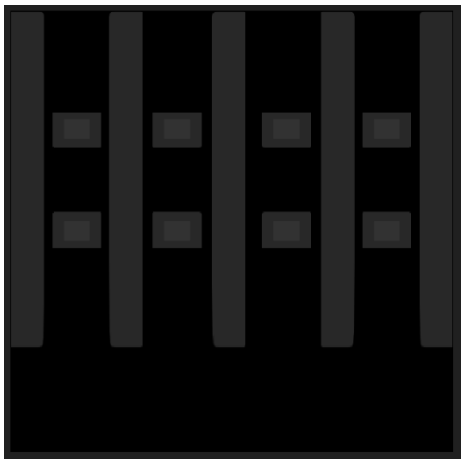

Abbildung 71: Verzierungen – Beispiel
Quelle: Eigene Aufnahme, 25.06.2025

Die Umsetzung erfolgt auch hier wieder mithilfe von Shape-Nodes, die in Form, Größe und Position angepasst werden. Im Unterschied zu den zuvor behandelten Modulen werden hier jedoch teilweise deutlich komplexere Strukturen aufgebaut, die aus mehreren Formen bestehen. Dies macht den gesamten Block in seiner Erstellung sowohl zeitintensiver als auch technisch anspruchsvoller.

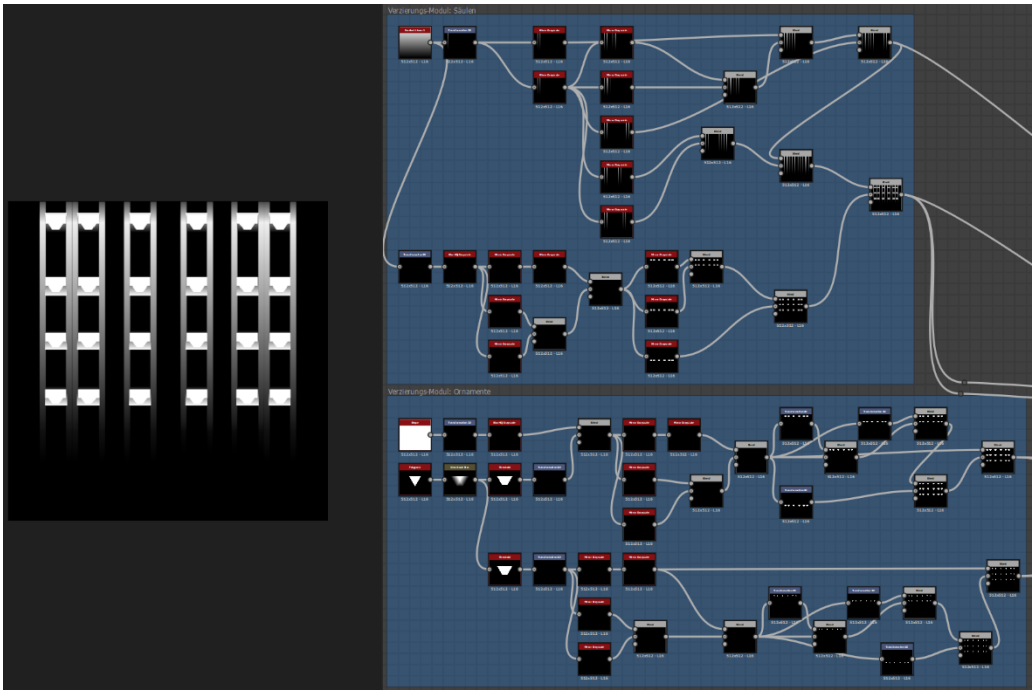

Abbildung 72: Verzierungs-Modul
Quelle: Eigene Aufnahme, 25.06.2025

Abbildung 72 zeigt das Verzierungs-Modul am Beispiel einer Fassade mit besonders vielen dekorativen Elementen. In diesem Fall ist das Modul in zwei separate Blöcke unterteilt. Zum einen die vertikalen Säulen, die entlang der Fenster verlaufen und zum anderen die

Ornamentstrukturen, die unterhalb der Fensterbänke platziert sind. Eine besondere Herausforderung liegt in der korrekten Anordnung der einzelnen Ebenen für die Height-Map. Die Säulen treten bereits aus der Fassadenfläche hervor, während die Ornamente zusätzlich auf den Säulen liegen und somit in der Height-Map noch weiter hervorgehoben werden müssen.

3.2.2.4. Bossenwerk – Modul

Das Bossenwerk stellt eine charakteristische Struktur dar, die häufig im unteren Bereich von Fassaden zu finden ist, insbesondere im Erdgeschoss. Hierbei handelt es sich um großflächige, rechteckige Fassadenelemente, die durch horizontale und gelegentlich auch vertikale Fugen optisch voneinander getrennt sind. Während bei klassischem Bossenwerken grob gehauene Natursteinquader verwendet werden, handelt es sich in städtischen Fassaden meist um eine deutlich einfachere Ausführung, den sogenannten Quaderputz (siehe Abbildung 73). Hier wird die Struktur der Steine lediglich in den Putz eingeritzt, sodass der Eindruck massiver Steinblöcke entsteht, ohne tatsächlich Werksteine zu verbauen. Diese Technik ist besonders in mitteleuropäischen Städten weit verbreitet und verleiht dem Sockelbereich eine robuste und architektonisch abgesetzte Wirkung (vgl. KOEPF/BINDING 2005: 380f.).

Abbildung 73: Quaderputz – Beispiel
Quelle: Eigene Aufnahme, 17.02.2025

Ausgangspunkt für die Erstellung bildet eine Grid-Textur, bei der die vertikalen Linien entfernt werden, sodass nur noch die horizontalen Fugen erhalten bleiben. Anschließend wird diese Textur invertiert und mit einer zusätzlichen Schwarz/Weiß-Maske kombiniert. Diese Maske wird über eine Shape-Node erstellt und steuert, in welchen Bereichen der Quaderputz sichtbar ist. Beim Verblenden mit der Grid-Textur bleiben die Linien nur in den gewünschten Bereichen erhalten. Zusätzlich wird die Fenster-Basis-Maske eingebunden, sodass auch die Fensteröffnungen von der Struktur ausgespart werden und die Linien nicht durch die Fensterflächen verlaufen (siehe Abbildung 74).

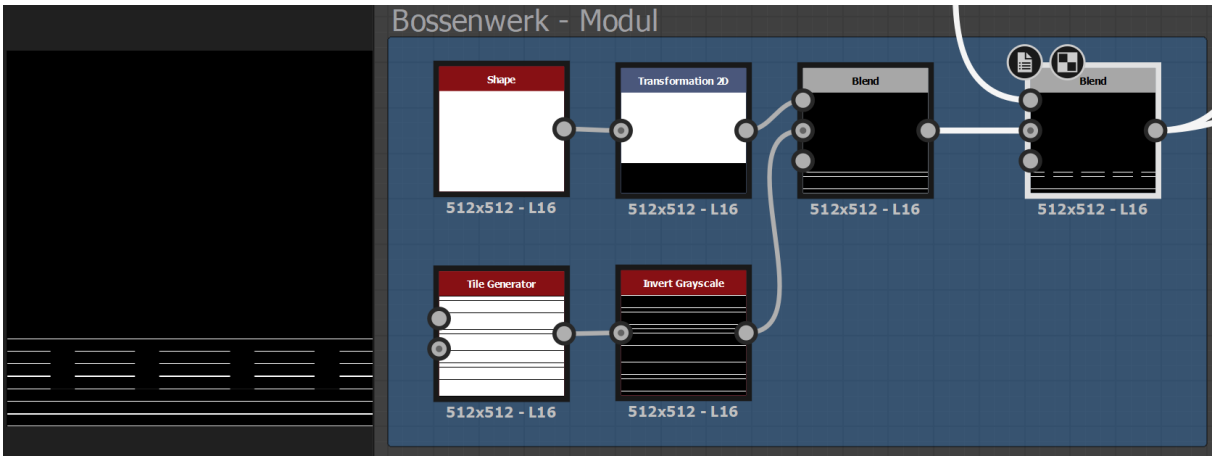

Abbildung 74: Bossenwerk – Modul
Quelle: Eigene Aufnahme, 25.06.2025

Ziegelmuster

Bei einigen Fassaden wird anstelle einer glatten Putzfläche ein Ziegelmuster verwendet. Die Erstellung dieses Musters orientiert sich am Aufbau des Bossenwerk-Moduls, unterscheidet sich jedoch in der Ausgangstextur. Hier wird eine Brick-Textur verwendet, die bereits vorgefertigt von Substance Designer bereitgestellt wird (siehe Abbildung 75).

Über die Parameter der Brick-Node lassen sich Ziegelgröße, Fugenbreite und Reihenversatz gezielt steuern, um verschiedene Ziegelmuster zu erstellen. Auch hier wird über eine Schwarz/Weiß-Maske gesteuert, in welchen Bereichen der Fassade das Ziegelmuster sichtbar ist.

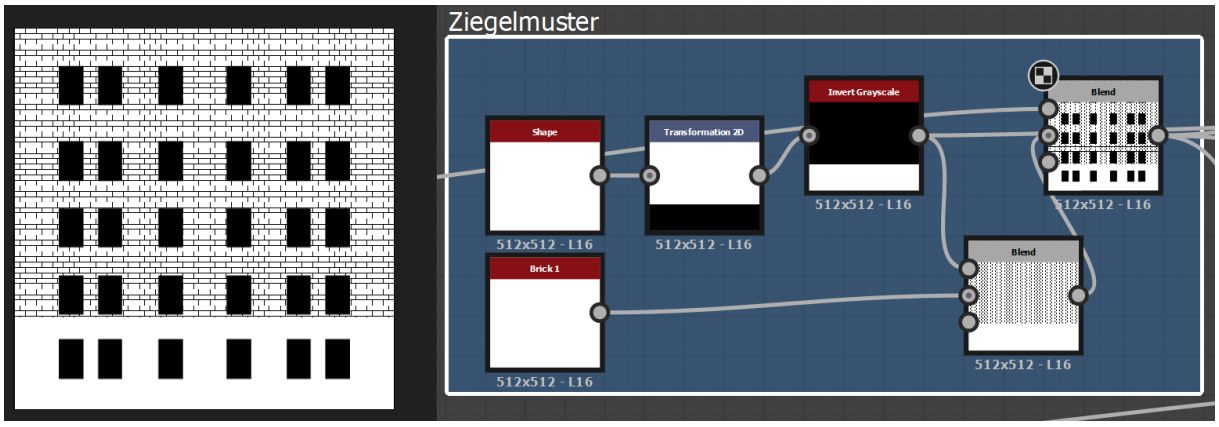

Abbildung 75: Ziegelmuster-Block
Quelle: Eigene Aufnahme, 25.06.2025

3.2.2.5. Textur – Modul

Nachdem in den vergangenen Abschnitten die einzelnen Fassadenelemente beschrieben wurden, folgen nun die Texturmodule, die der Fassade mehr Tiefe und Realismus verleihen. Dazu gehören die Normal-, Color-, sowie die Roughness Map. Die grundlegenden Funktionen dieser Texturen wurden bereits in Kapitel 2.2.2.1. erläutert.

Diese drei Texturen definieren zusammen die wichtigsten optischen Eigenschaften der Fassade. Auf zusätzliche Material-Texturen wie Metallic Map oder Ambient Occlusion Map wird bewusst verzichtet. Die Metallic Map wäre in diesem Fall lediglich für einzelne kleine Bereiche wie die Fensterrahmen relevant, die erst in der höchsten Detailstufe sichtbar sind. Da der metallische Charakter auch über die Roughness-Werte ausreichend angedeutet werden kann, wird auf eine separate Metallic Map verzichtet. Ähnlich verhält es sich mit der Ambient Occlusion Map.

Normal Map

Die Normal Map dient dazu, Unebenheiten auf der Oberfläche zu simulieren, ohne die Geometrie des Modells zu verändern. Für die Fassadenstrukturen bedeutet dies, dass feine Unebenheiten wie Putzstrukturen oder kleinere Materialunregelmäßigkeiten realistisch dargestellt werden können.

Für die Erstellung der Normal Map werden verschiedene Noise-Texturen verwendet (siehe Abbildung 76). Dabei werden grobe und feine Noise-Muster kombiniert, um unterschiedliche Unregelmäßigkeiten abzubilden. Um einen harmonischen Übergang zu erzeugen, wird die kombinierte Textur mithilfe einer Blur-Node weichgezeichnet. Anschließend wird die fertige Normal Map mit den jeweiligen Masken der einzelnen Module verknüpft, sodass die Oberflächenunebenheiten gezielt nur dort sichtbar sind, wo sie gewünscht werden.

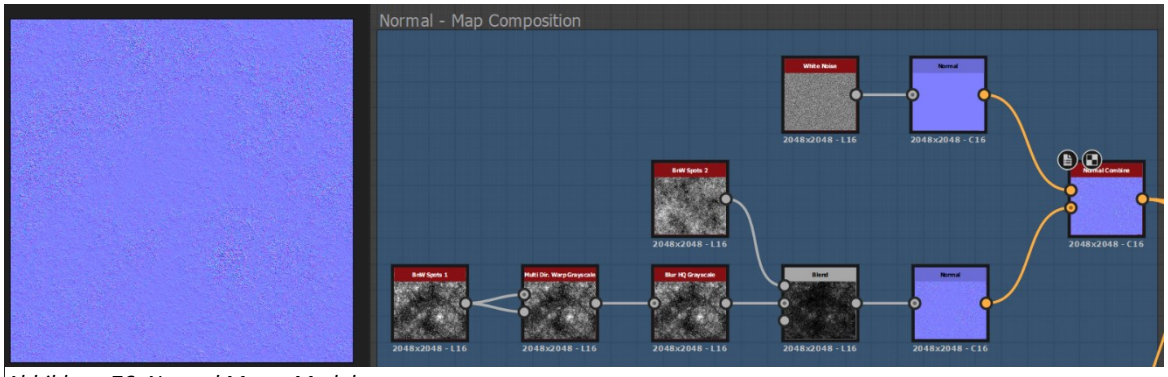

Abbildung 76: Normal Map – Modul
Quelle: Eigene Aufnahme, 26.06.2025

Color Map

Die Color Map legt die farbliche Gestaltung der Fassadenoberfläche fest. Die unterschiedlichen Fassadenelemente werden farblich differenziert, um ihre Gestaltung klar voneinander abzugrenzen.

Zur Erstellung der Color Map werden die Schwarz-Weiß-Masken der einzelnen Fassadenelement-Module genutzt. Jede Maske wird separat mit einer sogenannten Uniform Color-Node über eine Blend-Node miteinander verknüpft, wodurch jedem Fassadenelement eine spezifische Farbe zugewiesen wird (siehe Abbildung 77). Die Farbwahl orientiert sich an den Referenzfassaden und bildet typische Farbtöne städtischer Architektur ab.

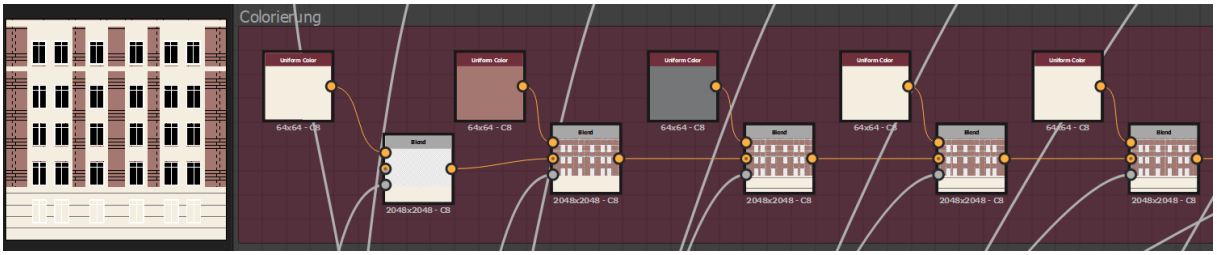

Abbildung 77: Color Map – Modul
Quelle: Eigene Aufnahme, 26.06.2025

Roughness Map

Die Erstellung der Roughness Map folgt nach dem gleichen Prinzip wie bei der Color Map, jedoch werden statt Farbwerten verschiedene Graustufen eingesetzt, um den Reflexionsgrad der jeweiligen Oberflächen zu steuern (siehe Abbildung 78).

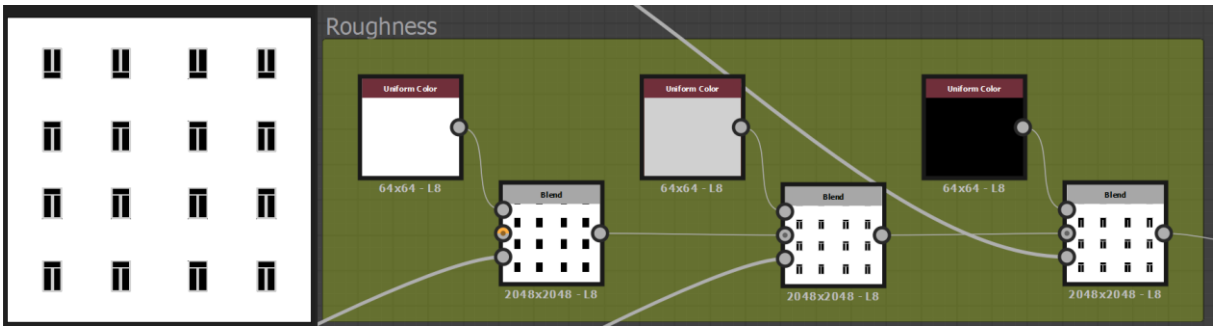

Abbildung 78: Roughness Map – Modul
Quelle: Eigene Aufnahme, 26.06.2025

Die Fenster erhalten einen schwarzen Wert, was in der Roughness Map einer maximal spiegelnden Oberfläche entspricht. Die übrigen Fassadenelemente werden in einem hellen Grauton eingefärbt, da die Fassadenoberfläche nur gering reflektiert. Vollständig weiße Werte, die für eine absolute Lichtstreuung stehen würden, kommen in der Realität nur selten vor.

Height Map

Wie bereits im Kapitel 3.2.2.1. erläutert, werden alle Module nach ihrer individuellen Erstellung durch gemeinsame Verarbeitungsschritte in eine einheitliche Height Map überführt. Die Schwarz-/Weiß-Masken der Module werden zunächst in einer Histogram-Range-Node angepasst, um den Grauwertbereich und damit die spätere Höhenwirkung zu steuern. Anschließend werden alle Module mit Height-Blend-Nodes miteinander verbunden und in eine Levels-Node überführt, wo eine Feinabstimmung des gesamten Wertebereichs folgt. Zum Abschluss wird die fertige Height Map als Material-Output ausgegeben (siehe Abbildung 79).

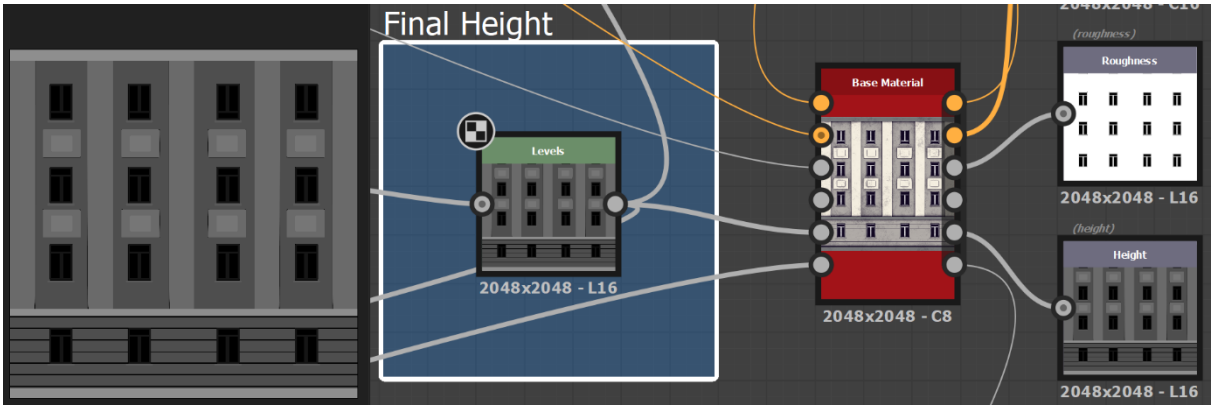

Abbildung 79: Height Map
Quelle: Eigene Aufnahme, 26.06.2025

Dirt - Overlay

Zu guter Letzt wird ein Dirt-Overlay erstellt, das Schmutz- und Alterungsspuren an den Kanten und Übergängen der Fassade simuliert. Dafür wird zunächst eine separate Normal Map aus der Height Map berechnet. Diese dient der Dirt-Node als Grundlage, um die Verschmutzungen korrekt zu platzieren. Im Anschluss wird die Dirt-Node mit der Base Color-Map kombiniert und in den Material-Output weitergeleitet (siehe Abbildung 80).

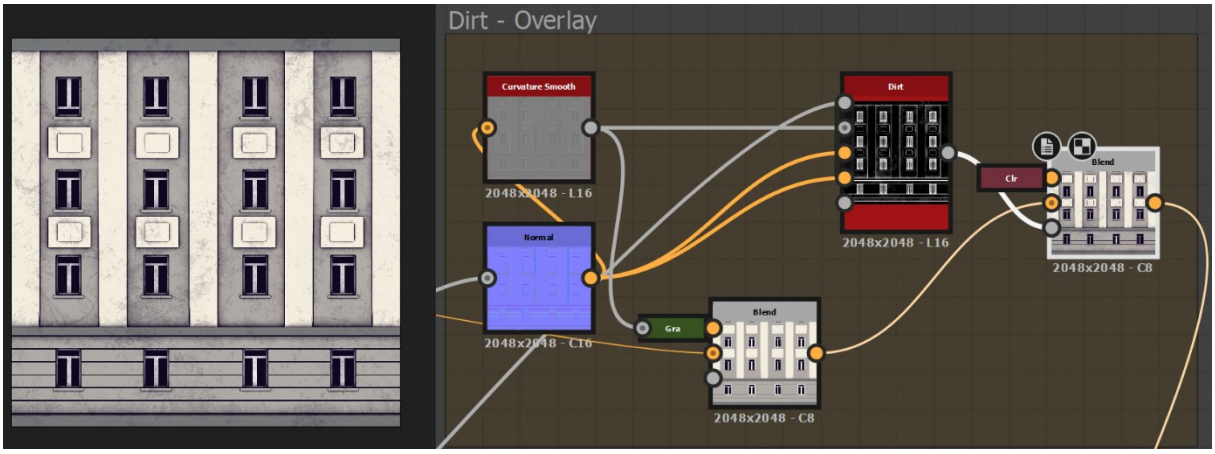

Abbildung 80: Dirt-Overlay
Quelle: Eigene Aufnahme, 26.06.2025

3.2.3. Erstellung der Texturatlanten

Nach der Erstellung der einzelnen Fassadentexturen werden diese in Texturatlanten zusammengefügt. Grundlage dafür bilden acht verschiedene Fassadentexturen, die jeweils in zwei Farbvarianten vorliegen. Daraus ergeben sich insgesamt 16 Einzeltexturen pro Auflösungsstufe. Für die Erstellung der Atlanten wird das Programm GIMP verwendet (siehe Kapitel 3.1.1.3.) und die Texturen werden manuell in der vorgesehenen Anordnung platziert, sodass die Zielauflösung des Atlasses vollständig ausgenutzt wird.

Für jede Auflösungsstufe und jeden Texturtyp wird ein separater Atlas erstellt. Insgesamt entstehen 8 Atlanten:

- **8K-Atlanten:** Hier werden die 16 Fassadentexturen mit jeweils 2048 × 2048 Pixel Auflösung zu einem 8192 × 8192 Pixel großen Atlas kombiniert. Für jede Texturart – Base Color, Roughness, Normal und Height – entsteht ein eigener Atlas.
- **4K-Atlanten:** In dieser Stufe werden die Texturen mit jeweils 1024 × 1024 Pixel Auflösung in einem 4096 × 4096 Pixel großen Atlas zusammengefasst. Es werden Atlanten für Base Color und Roughness erstellt.
- **2K-Atlanten:** Hier werden die Texturen in einer 512 × 512 Pixel Auflösung ausgegeben und zu einem 2048 × 2048 Pixel großen Atlas zusammengefügt. Auch in dieser Auflösung entstehen Atlanten nur für Base Color und Roughness.

Die Anordnung der Texturen bleibt für alle Atlanten hinweg identisch, was für den späteren UV-Unwrapping Prozess relevant ist. Nach dem Zusammenfügen werden die Atlanten im PNG-Format exportiert, um eine möglichst verlustfreie Speicherung bei gleichzeitig akzeptabler Dateigröße zu gewährleisten. Die resultierenden Dateigrößen variieren je nach Texturtyp. Abbildung 81 zeigt eine Übersicht der erstellten Atlasdateien. Die Werte zeigen, dass sich je nach Texturtyp teils erhebliche Unterschiede in der Dateigröße ergeben. So beanspruchen Base Color- und

Texturtyp	Auflösung	Dateigröße
Base Color	8K	43 MB
Normal Map	8K	42 MB
Height Map	8K	450 KB
Roughness	8K	344 KB
Base Color	4K	13 MB
Roughness	4K	137 KB
Base Color	2K	4 MB
Roughness	2K	72 KB

Abbildung 81: Übersicht: Dateigröße der Texturatlanten
Quelle: Eigene Darstellung, 26.06.2025

Normal-Atlanten durch ihre Farbdaten vergleichsweise viel Speicher, während Roughness- und Height-Maps deutlich kompakter sind, da sie lediglich Graustufen- bzw. Schwarzweißinformationen enthalten.

Abbildung 82 zeigt exemplarisch den fertigen Base Color Atlas in 4k Auflösung, d.h. die einzelnen Texturen haben eine Auflösung von 1024x1024 Pixel. Die restlichen Atlanten werden im Anhang dokumentiert.

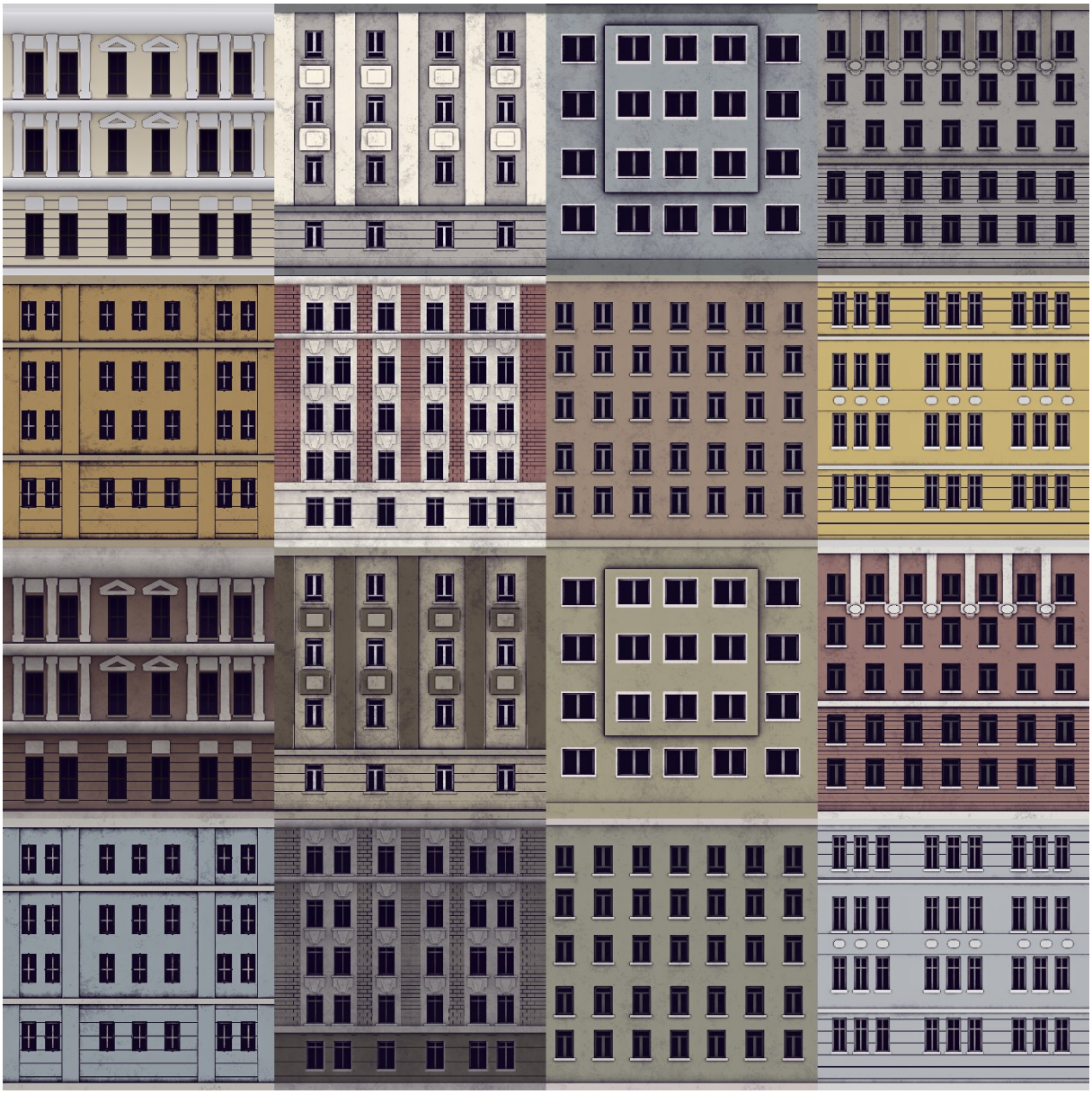

Abbildung 82: Base Color – Texturatlas in 4k Auflösung
Quelle: Eigene Aufnahme, 26.06.2025

3.2.4. Integration der Texturatlanten in Blender

Um die erstellten Texturatlanten hinsichtlich ihrer Effizienz zu testen, wird eine exemplarische Szene in Blender erstellt. Hierfür wird mit dem Blender-OSM (OpenStreetMap) Add-on ein Ausschnitt eines kleinen Stadtteils von Wien importiert. Grundlage bildet zunächst eine flache Ebene mit einem Orthofoto als Textur, um die Bodenfläche darzustellen. Anschließend werden die Gebäudemodelle im LOD1, also dem Blockmodell importiert (siehe Abbildung 83).

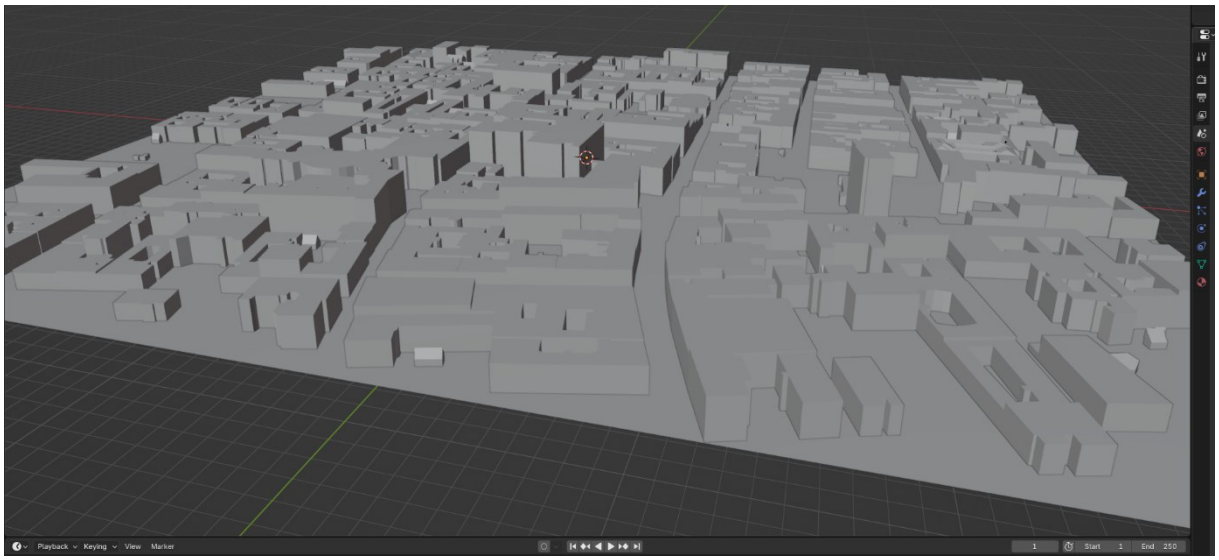

Abbildung 83: Ausschnitt 3D-Stadtmodell – LoD1
Quelle: Eigene Aufnahme, 28.06.2025

Beim Import der Gebäudemodelle werden automatisch simple Fassadentexturen auf die Gebäude projiziert. Diese stammen vom Add-on selbst und werden über interne Python-Skripte direkt und automatisiert beim Import zugewiesen und haben alle eine einheitliche Auflösung von 1024x1024 Pixel. Die Qualität dieser Texturen ist vergleichsweise gering, da sie sehr generisch wirken und wenn überhaupt nur für Szenen aus größerer Entfernung passend.

Im nächsten Schritt erfolgt das manuelle UV-Unwrapping der Gebäude, um die Texturatlanten korrekt zuzuweisen. Dabei werden die UV-Inseln der Gebäudemodelle so angepasst, dass sie exakt mit den Koordinaten der entsprechenden Fassadenbereiche im Texturatlas übereinstimmen (siehe Abbildung 84). Da die Anordnung der Texturen in allen Atlanten identisch ist, muss dieser Prozess nur einmal durchgeführt werden und bleibt für alle Auflösungsstufen gleich. Die erstellten UV-Maps können somit für sämtliche Texturatlanten direkt übernommen werden.

Abbildung 84: UV-Unwrapping der Gebäudemodelle
Quelle: Eigene Aufnahme, 28.06.2025

Zur Beleuchtung der Szene wird eine HDRI-Umgebungstextur (High Dynamic Range Image) verwendet, die auf Lichtinformationen realer Fotos basiert und realistische Himmels- sowie Umgebungsbeleuchtung ermöglicht. Zusätzlich wird ein Panoramabild der Stadt Wien als Hintergrund eingefügt, um einen passenden Übergang zum 3D-Modell zu schaffen und zu vermeiden, dass die Szene nach hinten hin abgeschnitten wirkt.

Für den späteren Rendervergleich werden drei Kameras aus verschiedenen Entfernungen platziert. Je nach Entfernung kommt einer der erstellten Texturatlanten zum Einsatz und wird mit den integrierten Texturen des Blender-OSM Add-ons verglichen. Für die Nahaufnahme kommen zwei unterschiedliche Methoden zum Einsatz, um die Geometrie der Gebäudemodelle darzustellen. Da die importierten Modelle im LoD1 vorliegen, bestehen die Flächen lediglich aus einfacher, flacher Geometrie. Das wiederum bedeutet, dass die Height Map nicht verwendet werden kann, um die Fassadenelemente hervorzuheben. Aus diesem Grund werden zwei verschiedene Methoden angewendet, um dennoch eine räumliche Tiefenwirkung zu erzielen.

Im ersten Ansatz wird die Geometrie einiger Gebäudemodelle mittels Subdivision verfeinert. Hierbei wird die Anzahl der Polygone um ein Vielfaches erhöht, um der Height Map genügend Auflösung zu geben (siehe Abbildung 85). Da in Nahaufnahmen meist nur wenige Gebäude im Bild sind, bleibt der Einfluss auf die Performance und Renderzeit begrenzt.

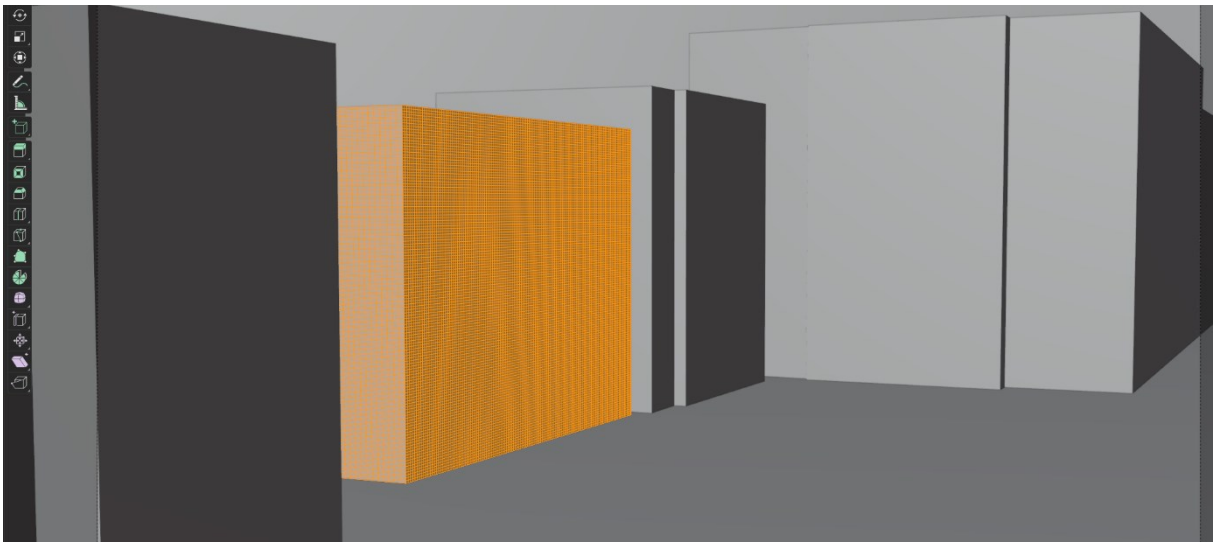

Abbildung 85: Erhöhte Geometrie
Quelle: Eigene Aufnahme, 28.06.2025

Die zweite Methode ist das im Kapitel 2.2.3.4. beschriebene Parallax Occlusion Mapping. Hierbei wird mithilfe eines Ray-Marching Verfahrens der Schnittpunkt zwischen Blickstrahl und Höhenfeld approximiert. In der praktischen Umsetzung wird ein vorgefertigter Shader eines Drittanbieters verwendet (XEOFRIOS 2024), der auf Basis der Height Map eine Tiefenillusion erzeugt, ohne die Geometrie des Modells tatsächlich zu verändern.

Der verwendete POM-Shader ist in Form eines komplexen Node-Tree-Systems aufgebaut, das aus mehreren verschachtelten Math-Nodes besteht (siehe Abbildung 86). Trotz der hohen internen Komplexität ist die Handhabung für den Anwender vergleichsweise einfach. Für die jeweilige Fassade müssen lediglich die Color- und Height-Map zugewiesen werden. Ein großer Nachteil ist jedoch, dass kein zusätzlicher Input für die Roughness Map vorgesehen ist. Dadurch können beispielsweise Fenstergläser keine realistischen Lichtreflexionen simulieren.

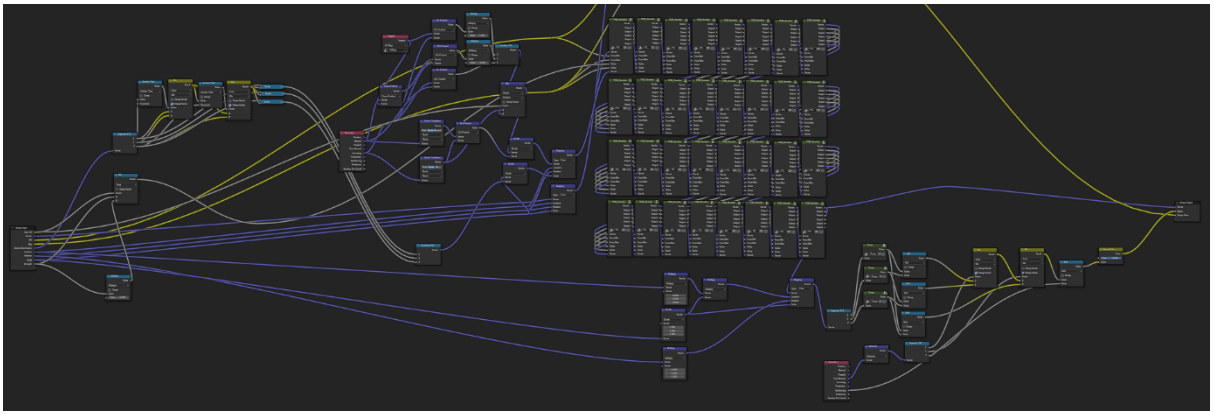

Abbildung 86: POM-Shader
Quelle: Eigene Aufnahme, 28.06.2025

4. Ergebnisse und Bewertung

Im folgenden Kapitel werden die erzielten Ergebnisse in Form der gerenderten Bilder der Beispielszene präsentiert und in Bezug auf die Leistungseffizienz und der visuellen Qualität bewertet. Als Vergleichsmaterial dienen die bereits erwähnten Texturen, die im Blender-OSM Add-on integriert sind.

Abbildung 87: (oben) 8K-Texturatlas mit erhöhter Geometrie / (unten) Blender-OSM Texturen
Quelle: Eigene Aufnahme, 05.07.2025

Abbildung 87 (oben) zeigt das Ergebnis der ersten Nahaufnahme mit erhöhter Geometrie der Modelle unter Verwendung der 8K-Texturatlanten. Es ist deutlich zu sehen, wie die einzelnen Fassadenelemente dreidimensional hervortreten und auch wie die Fenstergläser das

einfallende Licht reflektieren. Zum Vergleich zeigt der Abschnitt unten dieselbe Szene mit den OSM-Texturen. Anzumerken ist, dass die Gebäudehöhen in den beiden Darstellungen leicht variieren. Das liegt daran, dass die Standardtexturen nur in Kombination mit der Importoption „3D-realistisch“ verfügbar sind, während für die eigenen Texturatlanten die Option „3D-simpel“ verwendet wurde. Die Unterschiede in der Höhe haben allerdings keinen Einfluss auf die Texturqualität.

Abbildung 88: 8K-Texturatlas mit Parallax Occlusion Mapping Shader
Quelle: Eigene Aufnahme, 05.07.2025

Abbildung 88 zeigt das alternative Ergebnis mit der Parallax Occlusion Mapping Variante. Auf der links im Bild sichtbaren Fassadenfläche erzeugt der POM-Shader eine überzeugende Tiefenillusion. Die Wirkung ist jedoch stark blickwinkelabhängig. Bei Fassaden, die annähernd frontal zur Kamera stehen wie hier rechts im Bild, reduziert sich der Tiefeneindruck deutlich. Auch bei flachen Betrachtungswinkeln nimmt der 3D-Effekt ab bzw. kann verzerrt erscheinen.

Der wesentliche Vorteil des Parallax Occlusion Mappings liegt in der Viewport-Performance bei großen Szenen. Da die Gebäudemodelle hierbei geometrisch einfach bleiben muss Blender deutlich weniger Polygongeometrie verwalten. Navigation, Auswahl und allgemeines Arbeiten in umfangreichen Stadtmodellen kann auch mit leistungsstarker Hardware oft langsam werden, dem mit dem Einsatz von POM entgegengewirkt werden kann.

Abbildung 89 zeigt die Beispielszene aus mittlerer Distanz. Auch hier wieder oben die Version mit prozeduralem Texturatlas, diesmal in 4K-Auflösung und unten die Version mit den Blender-OSM Texturen. Hier kommt bereits ausschließlich die Base Color und Roughness Map zum Einsatz, da aus dieser Entfernung die feinen Reliefinformationen der Normal- und Height-Map kaum erkennbar sind. Die Lichtreflexionen hingegen sind bei dieser Entfernung noch deutlich zu erkennen, weshalb die Roughness Map durchaus Sinn macht. Da die OSM-Texturen ausschließlich aus Base Color Maps bestehen, fehlen hier eben jene Materialunterschiede.

Abbildung 89: (oben) 4K-Texturatlas / (unten) Blender-OSM Texturen
Quelle: Eigene Aufnahme, 05.07.2025

Abbildung 90 zeigt die Szene aus der dritten und weitesten Perspektive. Hier wird der Texturatlas mit 2K-Auflösung verwendet, bei der die einzelnen Texturen nur noch 512x512 Pixel groß sind und somit kleiner als die Standardtexturen des Blender Add-ons sind. Aufgrund der Perspektive ist dieser Unterschied visuell kaum relevant und hat praktisch keine Auswirkung auf die Bildqualität.

Wie bereits bei der mittleren Zoomstufe werden auch hier ausschließlich Base Color und Roughness Map verwendet. Die Lichtreflexionen der Fenstergläser sind zwar noch erkennbar, der Effekt lässt aber bereits deutlich nach. Die geringere Texturgröße wirkt sich positiv auf den Speicherbedarf aus und trägt damit zu kürzeren Renderzeiten bei.

Abbildung 90: (oben) 2K-Texturatlas / (unten) Blender-OSM Texturen
Quelle: Eigene Aufnahme, 05.07.2025

Abbildungen 91 & 92 fassen noch einmal alle Renderaufnahmen mit den jeweiligen Texturvarianten, Auflösungen sowie den zugehörigen Messwerten wie Speicherbedarf während des Renderings und der eigentlichen Renderzeit.

Zoomstufe	Texturvariante	Auflösung (effektiv)	Verwendete Maps	Besonderheiten (Geom./POM)	Speicherbedarf [MB]	Renderzeit [mm:ss.ms]
Nahaufnahme	8K - Texturatlas	2048x2048	Base Color / Roughness / Height / Normal	Erhöhte Geometrie	1229,04MB	03:22.31
Nahaufnahme	8K - Texturatlas	2048x2048	Base Color / Height	Parallax Occlusion Mapping	779,10MB	04:14.10
Nahaufnahme	OSM-Standardtexturen	1024x1024	Base Color	Standard	739.12MB	01:27.03
Mittlere Distanz	4K - Texturatlas	1024x1024	Base Color / Roughness	Standard	739.04MB	00:58.01
Mittlere Distanz	OSM-Standardtexturen	1024x1024	Base Color	Standard	739.13MB	01:14.44
Weiteste Distanz	2K - Texturatlas	512x512	Base Color / Roughness	Standard	739.04MB	00:32.62
Weiteste Distanz	OSM-Standardtexturen	1024x1024	Base Color	Standard	739.13MB	01:03.61

Abbildung 92: Leistungsvergleich der verschiedenen Testaufnahmen
Quelle: Eigene Darstellung, 08.07.2025

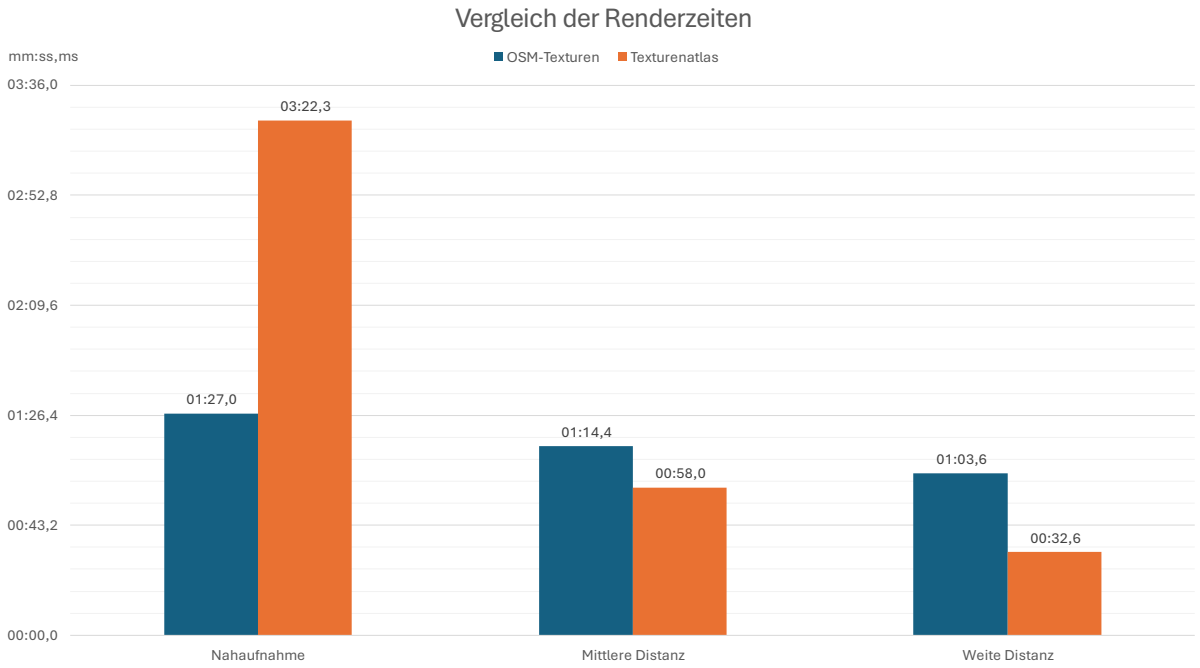

Abbildung 91: Leistungsvergleich – Diagramm
Quelle: Eigene Darstellung, 12.08.2025

Bei der Nahaufnahme mit erhöhter Geometrie zeigt sich, dass der Speicherbedarf mit 1229 MB und die Renderzeit mit 3:22 Minuten im Vergleich zu den Standardtexturen (739 MB und 1:27 Minuten) deutlich höher ausfallen, jedoch ist gerade hier die optische Aufwertung durch

den Einsatz einer Height Map am größten. Die POM-Variante liegt mit 779 MB Speicherbedarf bereits deutlich näher an den Standardtexturen, benötigt jedoch mit 04:14 Minuten die längste Renderzeit. Bezüglich der optischen Qualität positioniert sich diese Methode in der Mitte, da sie zwar realistischere Ergebnisse als die OSM-Texturen liefert, jedoch nicht das Niveau der Variante mit erhöhter Geometrie erreicht. Betrachtet man die Ergebnisse der erhöhten Geometrie Variante im direkten Vergleich zu den Standardtexturen zeigt sich, dass die zusätzliche Rechenzeit von ca. zwei Minuten eine deutliche Steigerung der optischen Qualität ermöglicht.

Bei den Aufnahmen aus mittlerer Distanz zeigt sich, dass der 4K-Texturatlas im Vergleich zu den Standardtexturen eine gute Balance bietet. Die Renderzeit liegt mit etwa 58 Sekunden zwar nur geringfügig unter den rund 01:14 Minuten der OSM-Variante, aber dafür überzeugt der 4K-Texturatlas vor allem durch die deutlich bessere und realistischere Optik. Der Speicherbedarf ist bei beiden Versionen nahezu identisch.

Da bei den Aufnahmen aus der weitesten Distanz die 2K-Texturatlanten verwendet werden, bei dem alle einzelnen Texturen eine Auflösung von 512x512 Pixeln haben und somit kleiner sind als die Standardtexturen ist hier der Unterschied in der Renderzeit am größten. Mit nur 32 Sekunden rendert der Atlas fast doppelt so schnell wie die OSM-Texturen (01:03 Minuten). Der Speicherbedarf ist mit 739,04 MB gegenüber 739,13 MB praktisch identisch und optisch bietet der Texturatlas weiterhin die ansprechendere Variante. Auch wenn die Lichtreflexionen durch die Roughness Map aus dieser Distanz weniger stark wahrnehmbar sind, tragen sie dennoch zur Verbesserung des Gesamtbildes bei.

4.1. Schwächen und Einschränkungen

Neben den positiven Ergebnissen zeigen sich auch verschiedene Limitationen des Workflows, die dessen Flexibilität und Anwendbarkeit einschränken. Diese Schwächen betreffen sowohl die technische Struktur der Texturen als auch die Modellierungs- und Arbeitsprozesse.

Die im Workflow erstellten Texturen liegen in einer statischen Form vor. Änderungen oder Anpassungen können derzeit nur über die Originaldateien in Substance Designer erfolgen. Dieser Umstand limitiert die schnelle Anpassbarkeit der Texturen innerhalb von Blender und führt zu einem Bruch im Workflow: Soll etwa eine Variation einer Fassade erstellt werden,

erfordert dies den erneuten Export aus Substance Designer. Für umfangreiche Projekte oder Szenarien mit mehreren Varianten ist dies unpraktisch und zeitaufwändig.

Ein weiterer Punkt betrifft die Gebäudestrukturen. Die erzeugten Texturen enthalten derzeit keine modellierten Dachaufbauten, was für manche 3D-Stadtmodelle unproblematisch ist, für andere jedoch ein klarer Nachteil. Zusätzlich fehlen modellierte Erdgeschosse mit Details wie Eingangstüren. Dies liegt zum einen am erhöhten Aufwand bei der Erstellung und zum anderen am Problem des Tilings. Bei der Texturprojektion kann es sonst dazu kommen, dass sich Elemente wie eben Eingangstüren unerwünscht vertikal wiederholen.

Drittens wurde das UV-Unwrapping und Mapping der Texturen manuell durchgeführt, was einen erheblichen Mehraufwand bedeutet. Im Gegensatz dazu bietet das Blender-OSM-Add-on eine automatisierte Lösung, die gerade für große Stadtmodelle besser skaliert.

Darüber hinaus ist auch die Bewertung der optischen Qualität hier kritisch zu betrachten. Die bisherige Einschätzung der visuellen Wirkung basiert größtenteils auf subjektiven Beobachtungen, was die Vergleichbarkeit und Nachvollziehbarkeit einschränkt. Eine objektivere Bewertung könnte beispielsweise über bildbasierte Metriken wie den Structural Similarity Index (SSIM) erfolgen. Der SSIM quantifiziert die Ähnlichkeit zwischen zwei Bildern, indem er Unterschiede in Helligkeit, Kontrast und strukturellen Details analysiert und so die menschliche Wahrnehmung der Bildqualität nachbildet. In Bezug auf die hier untersuchten Renderings von 3D-Stadtmodellen kann der SSIM genutzt werden, um objektiv zu erfassen, wie stark die selbst erstellten Fassadentexturen im Vergleich zu den standardmäßigen OSM-Texturen die Realitätsnähe der Gebäude verbessern (vgl. MOORTHY et al. 2011).

5. Fazit und Ausblick

Ziel der Arbeit war es, eine kostengünstige und flexible Methode zur Texturierung von 3D-Stadtmodellen zu entwickeln. Dabei stand insbesondere die Frage im Mittelpunkt:

„Wie effizient ist die Texturierung von 3D-Stadtmodellen durch die Integration prozeduraler Texturen in Multiresolution-Texturatlanten, insbesondere in Bezug auf Leistungseffizienz, Skalierbarkeit und visuelle Qualität?“

Die Ergebnisse dieser Arbeit zeigen, dass die entwickelten Multiresolution-Texturatlanten eine gute Balance zwischen Bildqualität und Effizienz bieten. Über alle drei getesteten Zoomstufen ergibt sich ein klares Muster. Bei den Aufnahmen aus weiter und mittlerer Entfernung stehen vor allem die Effizienzgewinne im Vordergrund. Die Renderzeiten fallen kürzer aus und die optische Darstellung reicht von mindestens vergleichbar bis deutlich verbessert. Die Ergebnisse der Nahaufnahmen haben zwar erhöhte Renderzeiten, liefern dafür jedoch deutlich mehr Detail und Realismus. Dieser Mehraufwand ist durchaus gut vertretbar, vor allem wenn hochwertige Nahaufnahmen gefragt sind. Ergänzend kann Parallax Occlusion Mapping eingesetzt werden, um Geometrie niedrig zu halten und dennoch qualitativ ansprechende Ergebnisse zu erzeugen, was vor allem die Handhabung bei sehr großen Szenen erleichtert.

Insgesamt bestätigt die Arbeit, dass die entwickelte Workflow die Forschungsfrage weitgehend beantwortet und eine flexible Lösung bietet, mit der je nach Anforderungen unterschiedliche Prioritäten zwischen Effizienz und visueller Qualität gesetzt werden können.

5.1. Zukünftige Forschung und Weiterentwicklung

Die zuvor angeführten Limitationen bilden die Grundlage für mögliche Weiterentwicklung des Workflows. Dabei lassen sich vor allem zwei zentrale Ansatzpunkte identifizieren, die sowohl die Flexibilität der Texturen als auch die Effizienz des Workflows erheblich verbessern könnten.

Ein vielversprechender Ansatz zur Effizienzsteigerung besteht darin, die Module in Substance Designer als sogenannte Parameter-Nodes zu exportieren und diese direkt in Blender zu importieren. Dadurch könnten Fassadenelemente unmittelbar in Blender verändert werden,

ohne dass ein erneuter Export der Texturen erforderlich ist. Diese Methode würde den Workflow deutlich flexibler und schneller machen, da Anpassungen in Echtzeit möglich wären. Allerdings setzt dies einen wesentlich komplexeren modularen Aufbau in Substance Designer voraus, um die Parameter sinnvoll zu strukturieren und die Kompatibilität mit Blender zu gewährleisten. Die Herausforderung liegt darin, die modulare Struktur so zu gestalten, dass sie einerseits alle notwendigen Anpassungsmöglichkeiten bietet und andererseits praktikabel und übersichtlich bleibt.

Ein weiterer wichtiger Entwicklungsschritt betrifft die Automatisierung verschiedener Arbeitsschritte im Texturierungsprozess. So könnten beispielsweise das UV-Unwrapping und Mapping mithilfe von Python-Skripten automatisiert werden, ähnlich wie es bereits im Blender-OSM Add-on realisiert ist. Auch die Erstellung der Texturatlanten ließe sich durch entsprechende Skripte weitgehend automatisieren. Durch diese Automatisierungen könnten repetitive und zeitaufwändige Aufgaben entfallen, was nicht nur die Effizienz erhöht, sondern auch die Konsistenz und Qualität der Ergebnisse verbessert. Langfristig bietet dies die Möglichkeit, den gesamten Workflow von der Modellierung bis zur finalen Texturierung stärker zu integrieren und zu optimieren.

6. Literaturverzeichnis

Alizadehashrafi, Behnam / Coors, Volker / Rahman, Alias Bin Abdul (2022): *Photorealistic versus Procedural Texturing of the 3D Buildings in Virtual Smart Cities.* Bezogen unter: doi:10.22059/eoge.2022.345196.1119

Akenine-Möller, Tomas / Haines, Eric / Hoffman, Naty / Pesce, Angelo / Iwanicki, Michał / Hillaire, Sébastien (2018): *Real-time rendering.* In: A Balkema book, Fourth edition, first issued in hardback, Boca Raton London New York: CRC Press. Bezogen unter: doi:10.1201/b22086

Biljecki, Filip (2017): *Level of detail in 3D city models.* Delft University of Technology. Bezogen unter: doi:10.4233/UUID:F12931B7-5113-47EF-BFD4-688AAE3BE248

Biljecki, Filip / Ledoux, Hugo / Stoter, Jantien (2016): *An improved LOD specification for 3D building models.* In: Computers, Environment and Urban Systems, 59/-/25–37. Bezogen unter: doi:10.1016/j.compenvurbsys.2016.04.005

Biljecki, Filip / Stoter, Jantien / Ledoux, Hugo / Zlatanova, Sisi / Çöltekin, Arzu (2015): *Applications of 3D City Models: State of the Art Review.* In: ISPRS International Journal of Geo-Information, 4/4/2842–2889. Bezogen unter: doi:10.3390/ijgi4042842

Buchholz, Henrik (2006): *Real-time visualization of 3D city models* [Dissertation, Universität Potsdam]. Bezogen unter: https://publishup.uni-potsdam.de/opus4-ubp/frontdoor/index/index/year/2007/docId/1253

Carr, Nathan A. / Hart, John C. (2002): *Meshed atlases for real-time procedural solid texturing.* In: ACM Transactions on Graphics, 21/2/106–131. Bezogen unter: doi:10.1145/508357.508360

Döllner, Jürgen / Baumann, Konstantin / Buchholz, Henrik (2006): *Virtual 3D City Models as Foundation of Complex Urban Information Spaces.* In: CORP 2006 & Geomultimedia06, 107-112. Bezogen unter: https://www.semanticscholar.org/paper/e9aff2a83086dcbc9abea029e95f21e15bcd4788

Dong, Junyu / Liu, Jun / Yao, Kang / Chantler, Mike / Qi, Lin / Yu, Hui / Jian, Muwei (2020): *Survey of Procedural Methods for Two-Dimensional Texture Generation.* In: Sensors, 20/4/1135. Bezogen unter: doi:10.3390/s20041135

Ebert, David S. / Musgrave, Kenton F. / Peachey, Darwyn / Perlin, Ken / Worley, Steven (2002): *Texturing & Modeling: A Procedural Approach.* In: Academic press. Bezogen unter: https://www.researchgate.net/publication/311272393_Texturing_and_Modeling_A_Procedural_Approach_Third_Edition

El-Mekawy, Mohamed (2010): *Integrating BIM and GIS for 3D City Modelling: The Case of IFC and CityGML* [Master's Thesis, Royal Institute of Technology (KTH)]. Bezogen unter: https://www.semanticscholar.org/paper/2315beacdb74d4f20579b75bc26b4cc19757d648

Eskef, Ali (2020): *Aggregation und Deduktion zwischen BIM und GIS-Modellen* [Master's Thesis, Universität Duisburg-Essen]. Bezogen unter: doi:10.13140/RG.2.2.28738.81603

ESRI Inc. (2025): *ArcGIS CityEngine – A modern 3D city design solution.* Bezogen unter: https://www.esri.com/en-us/arcgis/products/esri-cityengine/overview

Forkert, Gerald (o.J.): *3D Stadtmodellierung.*

Frueh, Christian / Sammon, R. / Zakhor, Avideh (2004): *Automated texture mapping of 3D city models with oblique aerial imagery.* In: Proceedings. 2nd International Symposium on 3D Data Processing, Visualization and Transmission, 2004. Bezogen unter: doi:10.1109/TDPVT.2004.1335266

Haeberli, Paul / Segal, Mark (1993): *Texture Mapping as a Fundamental Drawing Primitive.* In: Proc. Fourth Eurographics Workshop on Rendering, 259-266. Bezogen unter: https://www.researchgate.net/publication/2635152_Texture_Mapping_as_a_Fundamental_Drawing_Primitive

Heckbert, Paul S. (1986): *Survey of Texture Mapping.* In: IEEE Computer Graphics and Applications, 6/11/56–67. Bezogen unter: doi:10.1109/MCG.1986.276672

Herring, John (2001): *The OpenGIS Abstract Specification, Topic 1: Feature Geometry (ISO 19107 Spatial Schema), Version 5.* OGC Document Number 01-101.

Jebur, Ahmed / Fanar M. Abed / Mohammed, Mamoun Ubaid (2017): *3D City Modelling by Photogrammetric Techniques.* Bezogen unter: doi:10.13140/RG.2.2.11494.06722

Kahraman, İdris / Karaş, İsmail Rakıp (2014): *Texturing Techniques in 3D City Modeling.* Bezogen unter: https://www.isites.info/pastconferences/isites2014/ISITES2014/papers/B12-ISITES2014ID155.pdf

Kaneko, Tomomichi / Takahei, Toshiyuki / Inami, Masahiko / Kawakami, Naoki / Yanagida, Yasuyuki / Maeda, Taro / Tachi, Susumu (2001): *Detailed Shape Representation with Parallax Mapping.* In: Proceedings of the 11th International Conference on Artificial Reality and Tele-Existence (ICAT), 205-208. Bezogen unter: https://www.researchgate.net/publication/228583097_Detailed_shape_representation_with_parallax_mapping

Kaufmann, Thomas (2021): *Entwicklung eines neuen Erscheinungsbildes für 3D-Stadtmodelle am Beispiel der Stadt Wien* [Master's Thesis, Universität Wien]. Bezogen unter: https://utheses.univie.ac.at/detail/60651#

Khayyal, Heba K. / Zeidan, Zaki M. / Beshr, Ashraf A.A. (2022): *Creation and Spatial Analysis of 3D City Modeling based on GIS Data.* In: Civil Engineering Journal, 8/1/105–123. Bezogen unter: doi:10.28991/CEJ-2022-08-01-08

Koepf, Hans / Binding, Günther (2005): *Bildwörterbuch der Architektur.* Bezogen unter: doi:10.11588/jfk.2003.3.34235

Kolbe, Thomas H. (2009): *Representing and Exchanging 3D City Models with CityGML.* In: Lecture Notes in Geoinformation and Cartography, 15–31. Bezogen unter: doi:10.1007/978-3-540-87395-2_2

Lei, Binyu / Stouffs, Rudi / Biljecki, Filip (2023): *Assessing and benchmarking 3D city models.* In: International Journal of Geographical Information Science, 37/4/788–809. Bezogen unter: doi:10.1080/13658816.2022.2140808

Lévy, Bruno / Petitjean, Sylvain / Ray, Nicolas / Maillot, Jérome (2002): *Least Squares Conformal Maps for Automatic Texture Atlas Generation.* In: ACM Transactions on Graphics, 21. 362-371. Bezogen unter: doi:10.1145/566654.566590

Liu, Ruijun / Li, Haisheng / Lv, Zhihan (2023): *Modeling Methods of 3D Model in Digital Twins.* In: Computer Modeling in Engineering & Sciences, 136/2/985–1022. Bezogen unter: doi:10.32604/cmes.2023.023154

Lowe, David G. (1999): *Object Recognition from Local Scale-Invariant Features.* In: The Proceedings of the Seventh IEEE International Conference on Computer Vision 2, 1150-1157. Bezogen unter: doi:10.1109/ICCV.1999.790410

McDermott, Wes (2018): *The PBR-Guide: A Handbook for Physically Based Rendering.* In: Allegorithmic 2018. Bezogen unter: https://cgvr.cs.uni-bremen.de/teaching/cg_literatur/PBR%20Guide%20- %20Wes%20McDermott,%20allegorithmic,%202018.pdf

Mierzejowska, Aleksandra / Pomykol, Maciej (2019): *Calibration of raster image using GIS class software - accuracy analysis.* In: IOP Conference Series: Earth and Environmental Science, 261/-/012033. Bezogen unter: doi:10.1088/1755-1315/261/1/012033

Moorthy, Anush K. / Wang, Zhou / Bovik, Alan C. (2011): *Visual Perception and Quality Assessment*. In: Optical and Digital Image Processing. 1. Auflage, Wiley, 419–439. Bezogen unter: doi:10.1002/9783527635245.ch19

Navratil, G. / Konturek, P. / Giannopoulos, I. (2020): *INTERACTING WITH 3D MODELS – 3D-CAD VS. HOLOGRAPHIC MODELS.* In: ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, VI-4/W1-2020/-/129–134. Bezogen unter: doi:10.5194/isprs-annals-VI-4-W1-2020-129-2020

Neppius, Ale (2022): *3D Game Texturing: Comparative Analysis Between Hand-painted and PBR pipelines* [Bachelor's Thesis, South-Eastern Finland University of Applied Sciences]. Bezogen unter: https://urn.fi/URN:NBN:fi:amk-2022052411546

Open Geospatial Consortium (OGC). (2025): *Leading the Future of Geospatial Innovation.* Bezogen unter: https://www.ogc.org/who-we-are/

Parish, Yoav I.H. / Müller, Pascal (2001): *Procedural Modeling of Cities.* In: Proceedings of the 28th annual conference on Computer graphics and interactive techniques, 301-308. Bezogen unter: doi:10.1145/383259.383292

Plume, Jim / Mitchell, John (2007): *Collaborative design using a shared IFC building model—Learning from experience.* Automation in Construction, 16(1), 28–36. Bezogen unter: doi: 10.1016/j.autcon.2005.10.003

Purnomo, Budirijanto / Cohen, Jonathan D. / Kumar, Subodh (2004): *Seamless texture atlases.* In: Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing, 65–74. Bezogen unter: doi:10.1145/1057432.1057441

Schinko, Christoph / Krispel, Ulrich / Ullrich, Torsten / Fellner, Dieter W. (2015): *BUILT BY ALGORITHMS – STATE OF THE ART REPORT ON PROCEDURAL MODELING.* In: The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, XL-5/W4/-/469–479. Bezogen unter: doi:10.5194/isprsarchives-XL-5-W4-469-2015

Stoter, Jantien E. / Arroyo Ohori, Ken A. / Dukai, Balázs / Labetski, Anna / Kumar, Kavisha / Vitalis, S. / Ledoux, Hugo. (2020): *State of the art in 3D city modelling: Six challenges facing 3D data as a platform.* In: GIM International: The Worldwide Magazine for Geomatics, 34. Bezogen unter: https://www.gim-international.com/content/article/state-of-the-art-in-3d-city-modelling-2

Sun, Jing / Olsson, Perola / Eriksson, Helen / Harrie, Lars (2020): *Evaluating the geometric aspects of integrating BIM data into city models.* In: Journal of Spatial Science, 65/2/235–255. Bezogen unter: doi:10.1080/14498596.2019.1636722

Uggla, Gustaf (2015): *3D City Models – A Comparative Study of Methods and Datasets* [Master's Thesis, Royal Institute of Technology (KTH)]. Bezogen unter: https://api.semanticscholar.org/CorpusID:127154276

Vermandere, Jelle / Bassier, Maarten / Vergauwen, Maarten (2024): *Semantic UV mapping to improve texture inpainting for indoor scenes.* In: Computer Graphics and Visual Computing (CGVC). Bezogen unter: doi:10.2312/cgvc.20241221

Watson, Benjamin / Müller, Pascal / Veryovka, Oleg / Fuller, Andy / Wonka, Peter / Sexton, Chris (2008): *Procedural Urban Modeling in Practice.* In: IEEE Computer Graphics and Applications, 28/3/18–26. Bezogen unter: doi:10.1109/MCG.2008.58

Wehner, Michael F. / Oliker, Leonid / Shalf, John / Donofrio, David / Drummond, Leroy A. / Heikes, Ross / Kamil, Shoaib / Kono, Celal / Miller, Norman / Miura, Hiroaki / Mohiyuddin, Marghoob / Randall, David / Yang, Woo-Sun (2011): *Hardware/software co-design of global cloud system resolving models.* In: Journal of Advances in Modeling Earth Systems, 3/4. Bezogen unter: doi:10.1029/2011MS000073

Weinhaus, Frederick M. / Devarajan, Venkat (1997): *Texture mapping 3D models of real-world scenes.* In: ACM Computing Surveys, 29/4/325–365. Bezogen unter: doi:10.1145/267580.267583

Weiss, Sebastian / Bayer, Florian / Westermann, Rüdiger (2020): *Triplanar Displacement Mapping for Terrain Rendering.* Bezogen unter: doi:10.2312/egs.20201016

Xeofrios (2024): *Parallax Occlusion Mapping Test.* [Forumpost]. Bezogen unter: https://blenderartists.org/t/parallax-occlusion-mapping-test/1513622

Zhang, Xuequan / Liu, Wei / Liu, Bing / Zhao, Xin / Hu, Zihe (2021): *Size-Adaptive Texture Atlas Generation and Remapping for 3D Urban Building Models.* In: ISPRS International Journal of Geo-Information, 10/12/798. Bezogen unter: doi:10.3390/ijgi10120798

Zlatanova, Sisi / Stoter, Jantien / Isikdag, Umit (2012): *Standards for Exchange and Storage of 3D Information: Challenges and Opportunities for Emergency Response.* Bezogen unter: https://gdmc.nl/publications/2012/Standards_Exchange_Storage_3D_Information.pdf

7. Hilfsmittelverzeichnis

KI-Hilfsmittel	Einsatz	Betroffene Teile der Arbeit	Bemerkungen (Beispiele)
OpenAI ChatGPT 4	Übersetzung von Textpassagen	Kap. 2.1.2. Kap. 2.2.1. Kap. 2.2.3.3. Kap. 2.2.3.4. Kap. 2.2.4.2.	Abschnitte einiger Fachliteratur wurde zum besseren Verständnis ins Deutsche übersetzt.
OpenAI ChatGPT 4	Zusammenfassung von Fachliteratur	Kap. 2.1.2.3. Kap. 2.2.1.2. Kap. 2.2.4.2.	Die Zusammenfassungen dienten der schnelleren Filterung der relevanten Fachliteratur
OpenAI ChatGPT 4	Vorschlag für die Erstellung der Gliederung von Kapiteln	Kap. 2.2.3. Kap. 2.2.4.	Um inhaltliche Wiederholungen zu minimieren. Die Vorschläge wurden in Folge überarbeitet.

8. Abbildungsverzeichnis

9. Anhang