HAW HAMBURG

Erstellung modularer Gebäude für den Einsatz in Videospielen

Bachelor-Thesis
zur Erlangung des akademischen Grades B.Sc.

im Studiengang Media Systems

Nikolai Stecker 2240250

Erstprüfer: Prof. Ralf Hebecker

Zweitprüferin: Prof. Dr.-Ing. Sabine Schumann

Hamburg, 14.08.2019

Hochschule für Angewandte Wissenschaften Hamburg
Fakultät Design, Medien und Information
Department Medientechnik

Inhaltsverzeichnis

1 Einleitung **3**
- 1.1 Motivation . . . 3
- 1.2 Zielsetzung . . . 4

2 Einführung in modulare Assets **5**
- 2.1 Begriffserklärung und Definition . . . 5
  - 2.1.1 Modularität . . . 5
  - 2.1.2 Kit . . . 6
  - 2.1.3 Asset . . . 6
  - 2.1.4 Game-Engine . . . 7
- 2.2 Geschichte von Modularität in Videospielen . . . 7
- 2.3 Modularität in der Realität . . . 12

3 Grundlagen der Asset-Erstellung **14**
- 3.1 Planung . . . 14
- 3.2 Modellierung . . . 15
  - 3.2.1 Texturierung . . . 17
- 3.3 Implementierung . . . 20

4 Generierung modularer Assets **23**
- 4.1 Anforderungen an modulare Assets . . . 23
  - 4.1.1 Visuelle Qualität . . . 23
  - 4.1.2 Nutzerfreundlichkeit . . . 24
  - 4.1.3 Aufwand . . . 24
  - 4.1.4 Performance . . . 24
  - 4.1.5 Zusammenfassung . . . 25
- 4.2 Methoden für die Generierung modularer Assets . . . 26
  - 4.2.1 Projektübergreifende Planung . . . 26
  - 4.2.2 Modularitätsstufen . . . 26
  - 4.2.3 Raster . . . 28
  - 4.2.4 Positionierung und Nutzen des Pivot Points . . . 30
  - 4.2.5 Kit-spezifische Planung . . . 32
  - 4.2.6 Weitere Techniken . . . 35
  - 4.2.7 Texturierung von modularen Assets . . . 36

5 Beispielanwendung **37**
- 5.1 Das Spiel . . . 37
- 5.2 Planung . . . 38
- 5.3 Modellierung . . . 42
- 5.4 Texturierung . . . 46
- 5.5 Implementierung . . . 48

6 Evaluation **50**
- 6.1 Kit . . . 50
- 6.2 Methoden . . . 54

7 Fazit und Ausblick **57**
- 7.1 Fazit . . . 57
- 7.2 Ausblick . . . 58

Abbildungsverzeichnis **59**

Tabellenverzeichnis **61**

Literaturverzeichnis **62**

Abstract

This thesis deals with the study and application of modular design in video game development. In particular, methods are developed which are used for the creation of a modular kit. The focus is on the generation of buildings without interiors. The basics of modularity and asset development are explained at the beginning of this thesis. In addition, methods are developed that enable the use of modular design. Furthermore criteria are developed which can be used to evaluate modular assets.

By applying the methods, a set of assets could be created, with which theoretically infinitely many different buildings could be generated. The assets were created with Blender and implemented in Unreal Engine. Afterwards the applied methods could be evaluated by their practical use and benefit.

The bachelor thesis addresses both beginners and advanced users of the application of modular design in asset development. In addition, the topic is interesting for smaller game studios to optimize asset development.

Zusammenfassung

Diese Thesis beschäftigt sich mit der Untersuchung und Anwendung von modularem Design in der Videospielentwicklung. Im speziellen werden Methoden erarbeitet, die für die Erstellung eines modularen Kits eingesetzt werden. Der Fokus liegt dabei auf der Generierung von Gebäuden ohne Innenräume. Zu Beginn der Arbeit werden Grundlagen von Modularität und der Asset-Erstellung erläutert. Darüber hinaus werden Methoden erarbeitet, die den Einsatz von modularem Design ermöglichen. Es werden auch Kriterien erarbeitet, die modulare Assets bewerten können.

Durch Anwendung der Methoden konnte ein Set von Assets angefertigt werden, mit dem theoretisch unendlich viele verschiedene Gebäude generiert werden können. Die Assets wurden mit Blender erstellt und in Unreal Engine implementiert. Die angewandten Methoden konnten nach dem praktischen Einsatz auf ihren Nutzen hin bewertet werden.

Die Bachelorarbeit wendet sich sowohl an Einsteiger als auch an Fortgeschrittene der Anwendung von modularem Design in der Assetentwicklung. Zusätzlich ist das Thema für kleinere Game-Studios interessant, um die Asset-Entwicklung zu optimieren.

1 Einleitung

1.1 Motivation

Videospiele haben eine lange Entwicklung hinter sich. Nicht nur haben sich Gameplay und Storytelling in den letzten Jahrzehnten weiterentwickelt, sondern vor allem die Grafik hat sehr große Fortschritte gemacht. Ältere Spiele wie *Pong* (Atari, 1972) bestanden nur aus weißen Strichen und Punkten vor einem schwarzen Hintergrund, heutige Titel wie *God of War* (Santa Monica Studio, 2018) und *Far Cry New Dawn* (Ubisoft Montreal, 2019) sind der Realität näher als je zuvor. Eine Methode, die genutzt wird, um die immer größeren und imposanteren Welten zu erschaffen ist Modularität (Burgess & Purkeypile, 2013).

Modularität wird in der Videospielentwicklung in vielen Bereichen auf verschiedene Arten genutzt:

- Für das Erzeugen von Musik, die immer zum aktuellen Geschehen passt (Medina-Gray, 2014, S. 12),
- in der Programmierung, wo einmal programmierte modulare Komponenten auf verschiedene Arten genutzt werden, um diverse Aufgaben zu übernehmen (Coss, o.D.),
- für die Generierung von Items, um eine große Auswahl mit individuellen Eigenschaften zu erschaffen (Crecente, 2012),
- für die Entwicklung von Modellen, bei der vorgefertigte Teile immer wieder eingesetzt werden, um den Arbeitsprozess zu beschleunigen (Durand, 2019),
- und das Erzeugen von ganzen Leveln oder Levelelementen, die direkt in der Game Engine zusammengefügt werden können (Burgess & Purkeypile, 2013).

Letzteres ist für *Raw Vengeance Studios*, das Unternehmen in dessen Kontext diese Thesis ausgearbeitet wird, besonders interessant. Das Start-up entwickelt einen Third Person Cartoon Shooter namens *Renegade Line*. Ich bin Aktuell der einzige 3D-Artist der an der Entwicklung beteiligt ist und kann nicht genügend Modelle produzieren, um beide Level-Designer sinnvoll zu beschäftigen. Ein Aspekt der Modularität, welcher von großem Vorteil für das Unternehmen wäre, ist, dass es die Möglichkeit bieten soll, mit wenigen 3D-Artists viel Inhalt zu generieren (Burgess & Purkeypile, 2013). Dabei werden verhältnismäßig wenige von Artists erstellte Modelle von Level-Designern genutzt, um viel und abwechslungsreichen Content zu erstellen (Burgess & Purkeypile, 2013).

1.2 Zielsetzung

In dieser Arbeit soll aufgezeigt werden, welche Methoden für die Erstellung modularer Assets genutzt werden und welche Vorteile durch den Einsatz von modularem Design in Videospielen erzielt werden können. Dabei wird ein Set aus Modellen erstellt, mit dessen Hilfe eine Vielzahl an unterschiedlichen Gebäuden ohne Innenräume generiert werden kann.

Zu diesem Zweck werden zunächst Anwendungen von modularem Design in der Videospielentwicklung vorgestellt und es wird diskutiert, wie diese sich im Verlauf der Zeit entwickelt haben. Des Weiteren werden, anhand von Gebäudebau und Lego, Beispiele aus der Realität dargestellt, in denen Modularität Vielfältigkeit und Effizienz ermöglicht. Für ein besseres Verständnis der modularen Methoden werden im Anschluss Grundlagen der Asset-Erstellung erörtert. Darauf aufbauend werden die Methoden, mit denen modulares Design ermöglicht wird, aufgelistet und erklärt. Es wird außerdem darauf eingegangen, wie modulare Modelle bewertet werden können.

Nachdem alle theoretischen Aspekte abgehandelt wurden, werden die erarbeiteten Methoden in einem Projekt angewendet. In dem Projekt werden Modelle mit Blender erstellt und in Unreal Engine (kurz UE) implementiert. Mithilfe der zuvor erarbeiteten Kriterien werden im Anschluss die erstellten Modelle bewertet. Auch die genutzten Methoden werden auf ihre Nützlichkeit hin ausgewertet.

Abschließend wird ein Fazit aus den erlangten Informationen gebildet und ein Ausblick gewährt, welche Aspekte noch behandelt werden könnten und wie die Forschung diesbezüglich fortgesetzt wird.

2 Einführung in modulare Assets

In diesem Kapitel werden grundlegende Aspekte der Modularität erläutert.

2.1 Begriffserklärung und Definition

Folgende Begriffe sind für die Thesis relevant und werden deshalb in diesem Abschnitt definiert und erläutert:

- Modularität
- Kit
- Asset
- Game-Engine

2.1.1 Modularität

Wikipedia definiert Modularität wie folgt:

> „Modularität (auch Baustein- oder Baukastenprinzip) ist die Aufteilung eines Ganzen in Teile, die als Module (...) oder Bausteine bezeichnet werden. Bei geeigneter Form und Funktion können sie zusammengefügt werden oder über entsprechende Schnittstellen interagieren. Bei einem modularisierten Aufbau werden Systeme aus Bauteilen entlang definierter Stellen (...) zusammengesetzt. “ (Wikipedia, 2019)

Bei der Entwicklung von Videospielen kommt Modularität beispielsweise zum Einsatz, um aus kleinen, wiederverwendbaren Einzelteilen verschiedene Permutationen größerer Gebilde zu erzeugen (Meler, 2018). Diese Elemente können, wie im obigen Zitat beschrieben, an definierten Stellen miteinander verbunden werden.

Die Verwendung sogenannter „Kits“ erlaubt die Integration von Modularität in die Spieleentwicklung (Burgess & Purkeypile, 2013).

2.1.2 Kit

Ein „Kit“ (engl., Bausatz) ist eine Sammlung aus Elementen, die nach einem definierten System miteinander verbunden werden können. Ein Kit kann beispielsweise aus mehreren Elementen eines Rohrs bestehen, durch dessen Kombination verschiedene, größere und komplexere Systeme gebaut werden können (vgl. Abbildung 2.1). (Burgess & Purkeypile, 2013)

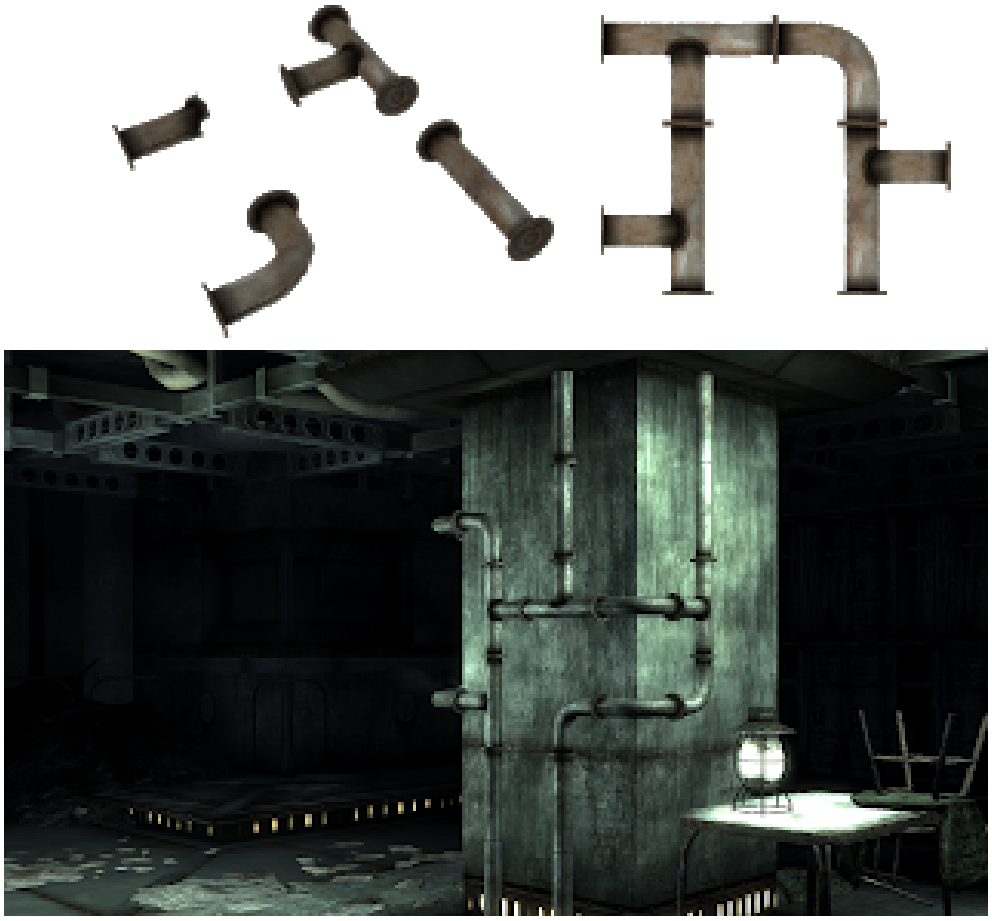


Abbildung 2.1: Beispielhaftes modulares Kit eines Rohr-Systems aus *Fallout 3* (Bethesda Softworks, 2008), (Burgess & Purkeypile, 2013).

2.1.3 Asset

Die Elemente die ein Kit ausmachen werden „Assets“ genannt und von G. Rehfeld folgendermaßen definiert:

> „Assets sind die benötigten Bestandteile eines Spiels. Bei einem Analogspiel die unterschiedlich farbigen Pöppel und Karten, das Spielbrett und die Würfel. In einem Digitalspiel handelt es sich um alle Objekte, die im Spiel auftauchen: Die Modelle und animierten Figuren, die Spieler und Lebewesen der Welt darstellen, feste Objekte der Spielwelt wie Autos, Bäume, Häuser, Gegenstände wie Waffen, Kleidung etc. Also alles, was man im Film als „Prop“ oder „Requisite“ bezeichnet und dazu dient, die Atmosphäre des Spiels so klar wie möglich darzustellen.“ (Rehfeld, 2013, S. 53-54)

Nach dieser Definition umfasst „Asset“ viele verschiedene Elemente. In dieser Arbeit wird „Asset“ hauptsächlich für die Bezeichnung fester Objekte in der Spielwelt genutzt und umfasst somit beispielsweise Gebäude, Höhlen oder Raumschiffe.

2.1.4 Game-Engine

Für die Entwicklung von Videospielen werden Game-Engines genutzt. Zu den aktuell bekanntesten gehören Unity und UE (GameDesigning, 2019). Jeff Ward, Spieleprogrammierer und Dozent, definiert die Leistung und den Nutzen von Game-Engines wie folgt:

> „...it exists to abstract the (sometime platform-dependent) details of doing common game-related tasks, like rendering, physics, and input, so that developers (artists, designers, scripters and, yes, even other programmers) can focus on the details that make their games unique.
>
> Engines offer reusable components that can be manipulated to bring a game to life. Loading, displaying, and animating models, collision detection between objects, physics, input, graphical user interfaces, and even portions of a game's artificial intelligence can all be components that make up the engine.“ (Ward, 2008)

Durch Spiele-Engines wird das Entwickeln von Spielen also erheblich erleichtert, da sie viele grundlegende Funktionen bieten und die Komplexität der Spieleentwicklung reduzieren. (Ward, 2008)

2.2 Geschichte von Modularität in Videospielen

Ein frühes Beispiel für Modularität in Videospielen ist *Super Mario Bros.* (Nintendo, 1987) für das Nintendo Entertainment System. Die Level des Spiels wurden auf Kästchenpapier entworfen, also auf einem Raster, mit dem die digitalen Koordinaten für das Spiel definiert werden konnten (vgl. Abbildung 2.2). Nach dem Designprozess wurden die Level von Programmierern Block für Block in das Spiel implementiert (Nintendo, Miyamoto, S., Tezuka T., 2015).

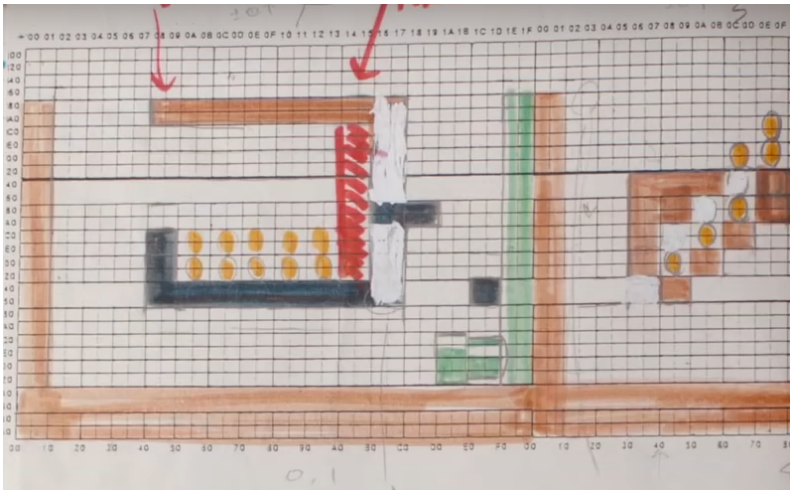


Abbildung 2.2: Ausschnitt eines Original Entwurfs eines *Super Mario Bros.* Levels. In Anlehnung an (Nintendo, Miyamoto, S., Tezuka T., 2015).

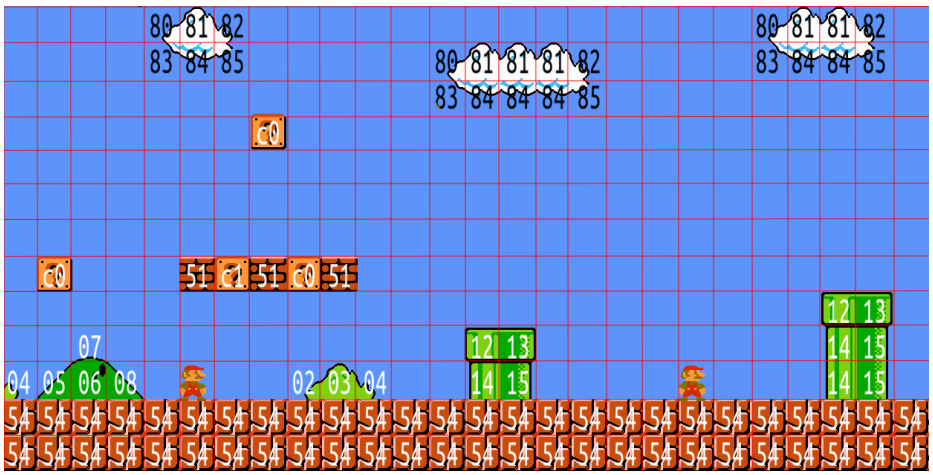


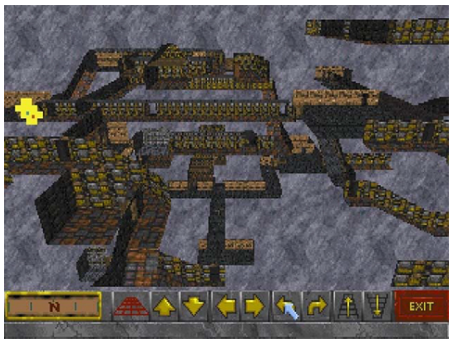
Abbildung 2.3: Analyse der Tiles in *Super Mario Bros.*. In Anlehnung an (Earl, 2018).

Alle Elemente der Level wurden mit Hilfe von Tiles[1] dargestellt. Wie in der Abbildung 2.3 zu sehen, besteht dieser Ausschnitt des Levels aus nur wenigen verschiedenen Tiles. Sie sind in einem Raster angeordnet und wurden so aufgebaut, dass sie nahtlos aneinanderpassen. So ließen sich mit nur wenigen Sprites[2] beliebig große Objekte erstellen und ganze Level bauen. Das grüne Rohr kann zum Beispiel mit den Stücken 14 und 15 beliebig verlängert werden. Das gleiche gilt für die Wolken und die Tiles 81 und 84. Diese modulare Herangehensweise erlaubte es den Entwicklern trotz der Hardware-Speicher-Limitierungen von damals größere Level zu entwickeln (Altice, 2015).

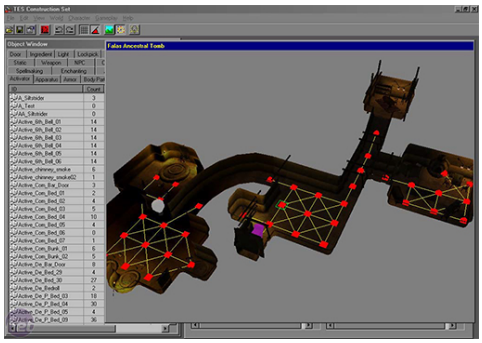
Die fortschreitende Technologie brachte die Möglichkeit mit sich, Spiele im dreidimensionalen Raum zu entwickeln, was als logischen Schluss hatte, dass die meisten Spiele fortan in 3D entwickelt wurden (Williams, 2017). Der Schritt in die dritte Dimension brachte sowohl in der Programmierung als auch beim Design der Spiele neue Hindernisse mit sich, die es zu überwinden galt.

Trotz der fortgeschrittenen Technik waren Speicherplatz und Rechenleistung noch nicht so ausgereift wie heute und stellten weiterhin ein Problem dar, wenn große Welten entwickelt werden sollten. Spiele wie *The Elder Scrolls II: Daggerfall*[3] (Bethesda Softworks, 1996) wussten sich durch den Einsatz der Modularität und Prozeduralität zu helfen und umgingen dieses Problem somit (Burgess & Purkeypile, 2016b). Die geschickte Kombination von Modularität und Prozeduralität erlaubte den Entwicklern mit Hilfe weniger modularer Dungeon-Blöcke eine Vielzahl unterschiedlich wirkender Dungeons zu generieren (Burgess & Purkeypile, 2016b). Die algorithmisch erstellten Level sorgten so für sehr viel mehr Abwechslung, wodurch in nur kurzer Zeit etwa 15.000 Orte generiert werden konnten (Elderscrollsportal, 2019). Die Welt von *Daggerfall* bestand aus zweidimensionalen Sprites, die in einer dreidimensionalen Umgebung platziert wurden. Obwohl diese Technik sich deutlich von der heutigen unterscheidet, ähneln die Modularitätsansätze in *Daggerfall* dennoch denen aktueller Spiele.

[1] Ein Tile ist eine Grafik, die mosaikartig zusammengesetzt ein größeres Gesamtbild ergibt.
[2] Ein Sprite ist eine zweidimensionale Grafik.
[3] Daggerfall ist ein Open-World-Action-Rollenspiel welches aus der Egoperspektive gespielt wird.



(a) Übersicht eines Dungeons aus *Daggerfall*.



(b) Übersicht eines Dungeons aus *Morrowind*.

Abbildung 2.4: Beispiele für Modulare Dungeons (Burgess & Purkeypile, 2016a).

The Elder Scrolls III: Morrowind (2002) von Bethesda Softworks ist der erste Teil der *Elder Scrolls*-Reihe, der komplett dreidimensionale Assets nutzte und neben der Egoperspektive auch über eine Third-Person-Perspektive verfügte. Auch in diesem Teil wurden die Dungeons mit Hilfe modularer Assets erstellt (Burgess & Purkeypile, 2016b). Dieses Mal wurden sie allerdings nicht prozedural generiert, sondern von Hand erstellt und genau von den Level-Designern geplant. Folglich war die Welt von *Morrowind* kleiner, aber realistischer aufgebaut, als die ihres Vorgängers (Kane, 2019). *Morrowind* hatte nur 0,01 % der Größe seines Vorgängers (Elderscrollsportal, 2019), was ein Beispiel für die Möglichkeiten von Modularität und prozeduraler Generierung ist. Ähnlich zu dem System aus *Super Mario Bros.* (Abschnitt 2.2) arbeitet der Level-Editor aus *Morrowind* mit einem Raster, auf dem verschiedene modulare Tiles aneinander gesteckt werden können. Dadurch entstehen aus den einzelnen Modulen komplette Höhlensysteme (Burgess & Purkeypile, 2016b).

Der in *Morrowind* genutzte Ansatz wurde auch bei *The Elder Scrolls IV: Oblivion* (Bethesda Softworks, 2006) angewandt. Für einen abwechslungsreicheren Look wurden modulare Assets auch mit vielen individuellen Assets kombiniert. (Burgess & Purkeypile, 2016b)

Für *The Elder Scrolls V: Skyrim* (Bethesda Game Studios, 2012) wurde das in *Morrowind* und *Oblivion* genutzte System für Modularität weiter verbessert. Zu Beginn der Entwicklung wurden Kits definiert, die jeweils einen Bereich der Spielwelt abdecken sollten. Die Kits beinhalteten Assets, mit denen natürlichere Formen, wie Höhlen und Tunnel, dargestellt werden konnten. Dadurch war es möglich, sich von dem kastenförmigen Erscheinungsbild, welches modulare Ansätze vorher prägte, zu entfernen.

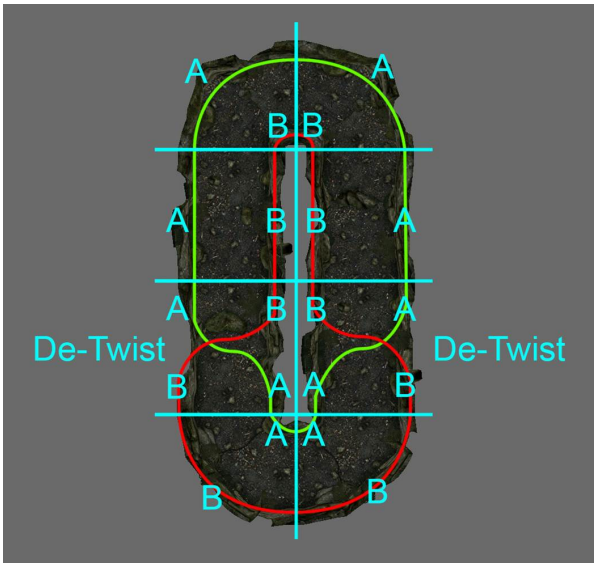


Abbildung 2.5: Asymmetrische Tunnelmodule aus *Skyrim* (Burgess & Purkeypile, 2013).

Hierfür wurden unter anderem asymmetrische Module entwickelt, die sich nur in eine Richtung miteinander verbinden ließen. Die Nutzung asymmetrischer Assets verursacht mehr Arbeit, diese wird aber durch die höhere visuelle Qualität gerechtfertigt. Durch für alle Kits definierte Regeln konnten die Assets der verschiedenen Kits auch gut untereinander kombiniert werden. Dank dieser Technik reichten schon sieben verschiedene Kits, für genügend Abwechslung in den über 400 Leveln *Skyrims*. Bethesdas Hauptziel beim Einsatz von modularem Design war, trotz weniger Mitarbeiter möglichst viele Inhalte generieren zu können. Von den zur Zeit der Entwicklung 90 Mitarbeitern im Entwicklungsteam waren nur zehn für die Level-Erstellung zuständig. (Burgess & Purkeypile, 2013)

Mittlerweile ist Modularität eine weit verbreitete Technik in der Videospielindustrie, die je nach spezifischem Einsatzgebiet sehr unterschiedliche Ansätze und Ziele verfolgt. Allen gemeinsam ist die Idee der Wiederverwertung getaner Arbeit, sowie des Einsparens von Speicher und der Optimierung der Performance (Meler, 2018).

Ein aktuelleres Beispiel für modulares Design in Videospielen ist das Online-Hack-and-Slay *For Honor* (2017) von Ubisoft. In diesem Spiel können sich Spieler mit verschiedenen Kriegern bekämpfen. Diese Kämpfe finden auf verschiedenen, voneinander getrennten Karten, in einem von drei Settings (Ritter, Samurai und Wikinger) statt (Ubisoft Entertainment, o.D.).

Für das Entwickeln der Karten werden modulare Kits genutzt. Es gibt Kits, die nur für spezifische Orte auf einer Karte genutzt werden. Die Modularität dieser Kits verliert dadurch an Wert. Ein Vorteil von großer Bedeutung für Ubisoft bleibt aber bestehen: Der Aufbau der Level ist durch ihre Modularität leicht veränderbar. Dies ermöglicht es nach Tests, Gebiete oder einzelne Räume ohne große Umstände für verbessertes Gameplay anzupassen (Durand, 2019). Das ist für ein Kampfspiel, besonders wichtig.

Abbildung 2.6: Modulare Assets in *For Honor*. In Anlehnung an (Durand, 2019).

Abbildung 2.6 zeigt einen Raum, der aus einem solchen Kit entstanden ist. Ergänzend wurde er mit einigen weiteren Assets ausgestaltet. Alle Module in For Honor folgen festen Regeln, die zu Beginn der Entwicklung festgelegt wurden. Diese Regeln umfassen die genutzte Größeneinheit, unterschiedliche Größen für verschiedene modulare Teile und das Verhalten der Assets auf dem genutzten Raster. (Durand, 2019)

Auch viele der nicht modularen Assets in *For Honor* wurden mit Hilfe der Modularität erzeugt. Der Lead-Level-Designer hat zu Beginn des Projektes ein Kit aus Steinen und eines aus Holzbrettern angefertigt. Die anderen 3D-Artists konnten diese Kits nutzen, um die ihnen zugewiesenen Modelle zu erstellen. So konnten die Assets auch bei hoher Entwicklungsgeschwindigkeit mit einem großen Detailgrad entworfen werden. (Durand, 2019)

Modularität findet mittlerweile nicht mehr nur für die Gestaltung von Levels Anwendung, sondern auch in anderen Bereichen, wie z. B. der Erstellung von Assets. Wie aufgezeigt wurde, haben sich die dafür genutzten Methoden über die letzten Jahrzehnte durchgehend geändert und verbessert. Die meisten modularen Systeme basieren allerdings immer noch auf der Nutzung von Rastern. Hauptsächlich die Methoden, wie Raster genutzt werden, haben sich weiterentwickelt. Auch aktuell werden immer neue Methoden, beispielsweise die Verwendung künstlicher Intelligenz, in die Produktionsabläufe integriert, die die Vorteile von Modularität weiter ausreizen.

2.3 Modularität in der Realität

Nicht nur in Videospielen, sondern auch in der echten Welt werden die Vorteile der Modularität ausgenutzt.

Ein frühes Beispiel für Modularität ist der Gebäudebau. Eines der ersten Systeme welches den modernen Ansätzen ähnelt, wurde ca. 1830 von Henry Manning entwickelt (Smith, 2011, Abschnitt 1.1.1). Manning verkaufte vorproduzierte Elemente, welche mit einfachen Werkzeugen zu kleinen Häusern zusammengebaut werden konnten (Smith, 2011, Abschnitt 1.1.1).

(a) Ein Gebäude auf Basis eines Grundgerüsts (Barrett Byrd Associates, 2011).

(b) Ein Gebäude, dessen Stabilität durch die Wände der Module entsteht (Triumph Modular, o.D.).

Abbildung 2.7: Beispiele der zwei genannten Methoden für den Bau modularer Gebäude.

Heutzutage werden für den Bau modularer Gebäude hauptsächlich zwei verschiedene Methoden genutzt. Erstere findet vor allem bei größeren Bauwerken Verwendung. Hier wird zuerst ein Grundgerüst gebaut, in das die Module eingesetzt werden. Die Lastverteilung verläuft hier über Eckpfeiler. Die andere Methode arbeitet mit Modulen, deren Lasten über die Wände verteilt werden. Letztere kann nur eine Höhe von 4 bis 8 Stockwerken erreichen. (Lawson & Ogden, 2010)

Der Produktionsablauf unterscheidet sich in vielen Punkten von der normalen Art Gebäude zu errichten, bei welcher der Großteil der Arbeit vor Ort auf der Baustelle durchgeführt wird. Die genutzten Module werden in einer Fabrik, abseits der Baustelle, angefertigt und dann vor Ort mit dem Fundament und anderen nicht-modularen Bauelementen verbunden. So ist es (gegenüber der herkömmlichen Methode) möglich, die Bauzeit um bis zu 50% zu verkürzen. So können die neuen Räumlichkeiten schneller genutzt werden und das Umfeld der Baustelle wird nicht so lange belastet, wie es normalerweise der Fall ist. (Lawson & Ogden, 2010)

Mit der Anzahl wiederverwendeter Module sinken die Kosten, weil das Modul nur einmal entworfen werden muss und dann immer wieder auf die gleiche Art produziert werden kann (Lawson & Ogden, 2010). Dieser Vorteil gleicht der Wiederverwendbarkeit der Module in Videospielen, die ebenfalls nur einmal designt und dann beliebig oft eingesetzt werden können.

Das wohl offensichtlichste Beispiel sind LEGO-Steine. LEGO hat ein System entwickelt bei dem sich fast alle der verschiedenen produzierten Teile miteinander kombinieren lassen und das auf viele verschiedene Arten. Sechs gleichfarbige Sechser-Blöcke können auf 915.103.765 Arten (Hugo, 2018, S. 10) verbunden werden, um verschiedene Formen darzustellen. Auch die Teile der LEGO-Figuren lassen sich beliebig miteinander kombinieren.

Es existieren LEGO-Sets, die standardmäßig darauf ausgelegt sind, dass drei unterschiedliche Modelle, aus den im Set vorhandenen Teilen, gebaut werden können. Zusätzlich lassen sich, wie bei allen LEGO-Sets, noch weitere Modelle bauen, die nicht durch die Anleitung vorgegeben sind (The LEGO Group, o.D. a). Seit 2007 gibt es modulare Sets, die diesen Ansatz weiter verfolgen, in denen zusammengebaute kleinere Teile auf verschiedene Weisen miteinander verbunden werden können um das Aussehen des Modells auf einfache Art abzuwandeln (The LEGO Group, o.D. b).

Abbildung 2.8: Das Raster nach dem alle LEGO-Steine Entwickelt werden. In Anlehnung an (Hugo, 2018, S. 11).

Der modulare Ansatz ermöglicht es dem Unternehmen neue Sets zu entwerfen, die zum Beispiel den Eiffelturm darstellen, ohne dass dafür neue Teile entwickelt werden müssen. Stattdessen kann auf die bestehende Palette von LEGO-Steinen zurückgegriffen werden. Damit dies funktioniert, müssen alle Steine Regeln befolgen und auf einem bestimmten Raster entworfen werden (vgl. Abbildung 2.8). Hierdurch sind alle jemals entwickelten Teile zumindest stückweise miteinander kompatibel (Hugo, 2018, S. 11).

Es existieren viele weitere Anwendungsgebiete für Modularität. Die hier aufgezeigten Beispiele sollen genügen, um einen Eindruck zu vermitteln, welche Möglichkeiten und Vorteile Modularität mit sich bringen kann.

3 Grundlagen der Asset-Erstellung

In diesem Kapitel wird auf die grundlegenden Vorgehensweisen für die Erstellung und Implementierung von 3D-Modellen für Videospiele eingegangen. Da die Vorstellung aller Workflows rund um die Erstellung dreidimensionaler Modelle den Rahmen dieser Arbeit übersteigt, konzentriert sich der nachfolgende Abschnitt lediglich auf den traditionellen Ansatz.

3.1 Planung

Der erste Schritt in der Entwicklung eines Assets ist die Planung. Die Länge dieser Phase ist abhängig vom jeweiligen Projekt und dessen Fortschritt. Sind der Stil und das Setting des Spiels noch nicht definiert, wird diese Phase mehr Zeit in Anspruch nehmen.

Entscheidend sind auch die Komplexität des Assets und sein geplanter Nutzen. Handelt es sich zum Beispiel um ein Fahrzeug, das der Spieler benutzen soll, müssen vorab Regeln definiert werden, die das Modell erfüllen muss, um mit der Spielfigur und der Umwelt kompatibel zu sein. Dazu gehören das Zusammenspiel mit der Spielphysik und die Beeinflussung durch die Programmierung. Für Gebäude sollte zum Beispiel die Größe der Spielfigur für Türen und Abstände in den Räumen beachtet werden. Um Problemen mit der Kamera vorzubeugen, sollte das Verhalten der Kamera bei der Planung von Decken und Wänden bedacht werden. Jedes Modell-, beziehungsweise jeder Modell-Typ, bedarf spezifischer Planung, damit es im Spiel ordnungsgemäß funktioniert. Um große Überarbeitungen in späteren Revisionen des Modells zu verhindern, sollte bereits zu Beginn sichergestellt werden, dass das geplante Modell die Regeln des Projektes einhält.

Um die Umsetzung des Assets zu beschleunigen und möglichst wenig Iterationsschritte zu benötigen, sollte vor der eigentlichen Modellierungsphase ein oder mehrere Konzeptgrafiken oder wenigstens andere Referenzgrafiken vorliegen.

3.2 Modellierung

In diesem Abschnitt werden der Aufbau eines 3D-Modells und die Arbeitsschritte in seinem Produktionsablauf erklärt. Der Abschnitt basiert im wesentlichen auf dem Handbuch von T. Beck (Beck, 2017) und fasst die hier relevanten Aspekte zusammen. Bei der vorgestellten Methodik wird erst das Modell erzeugt und danach eine passende Textur generiert. Es ist auch möglich, diese Schritte zu vertauschen und Modelle von einer Textur ausgehend zu erzeugen. Auf dieses Verfahren wird hier aber nicht eingegangen[4].

Grundlagen und Begriffe

Grundlegend bestehen 3D-Modelle aus einer Ansammlung von Punkten in einem dreidimensionalen Raum. Diese Punkte heißen Vertex (Singular), bzw. Vertices (Plural). Eine Verbindung zwischen Zweier dieser Punkte heißt Kante oder Edge. Bilden mehrere Edges eine geschlossene Fläche, so spricht man von einem Face.

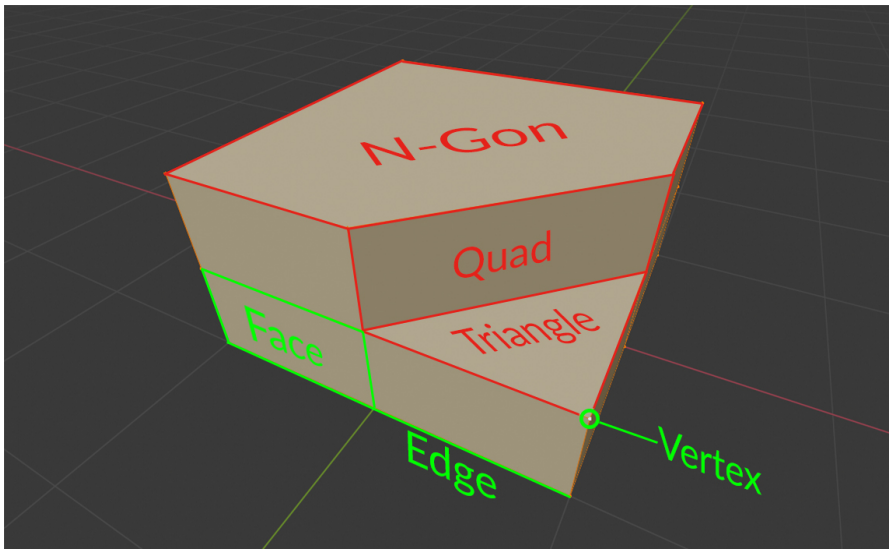


Abbildung 3.1: Darstellung der Bestandteile eines Meshes.

Wie in Abbildung 3.1 ersichtlich, kann die Anzahl der verwendeten Vertices zur Bildung eines Faces variieren. Man unterscheidet in der Regel zwischen Triangles (Face mit drei Vertices), Quads (Face mit vier Vertices) und N-Gons (Face mit fünf oder mehr Vertices). Im Hintergrund bestehen alle Flächen aus Dreiecken, egal, wie viele Eckpunkte sie haben (Beck, 2017, S. 82-83). Während des Modellierens ist es von Vorteil mit Quads zu arbeiten, sie lassen sich leichter unterteilen und wieder zu einem großen Quad zusammenführen. N-Gons sollten vermieden werden, weil diese zu Fehlern in der Darstellung und Manipulation führen können.

Während Vertices und Edges die Form eines Modells definieren, repräsentieren Faces die sichtbaren Flächen des Objekts. Die Richtung, aus der eine Fläche zu sehen ist, ist abhängig von ihrer Flächennormalen. Wenn die Flächennormale nicht manipuliert wurde steht sie senkrecht auf der zugehörigen Fläche (Beck, 2017, S.88).

[4] Der interessierte Leser findet hierfür ein Beispiel in dem „Modular Building Breakdown“ von Jacob Norris (Norris, 2015).

Von der der Flächennormalen abgewendeten Seite aus ist die Fläche unsichtbar. Es gibt in den meisten Engines eine Funktion für beidseitiges Rendern von Faces, dies ist aber sehr Performanceintensiv.

Das sogenannte Shading bestimmt das Erscheinungsbild eines Meshes. Zwei spezielle Formen die direkt mit der Topologie des Models arbeiten, sind Flat und Smooth Shading.

Bei der normalen Darstellung (Flat) wird jedes Face für sich gerendert, ohne die angrenzenden Faces zu beachten. Für Smooth Shading wird zwischen den Flächennormalen interpoliert, wodurch eine glattere/weichere Erscheinung des Meshes entsteht. Diese beiden Modi können auf einzelne Faces oder das gesamte Mesh angewendet werden. Dadurch sind sowohl weiche als auch harte Flächen auf einem Objekt darstellbar. (Beck, 2017, S. 388-394)

Nach den Grundlagen folgt jetzt der eigentliche Prozess der Modell-Erstellung.

Prozess der Modell-Erstellung

Für die Modell-Erstellung kommen nun die Konzeptgrafiken und Referenzen zum Einsatz. Gute Konzepte können als Blaupause im 3D-Tool genutzt werden, wodurch das Konzept detailgetreu nachgebildet werden kann. Auch Referenzen eignen sich, wie bereits erwähnt, um das Modell nachzubauen.

Durch Manipulation der Vertices, Edges und Faces kann jede mögliche Form modelliert werden. In der Videospielproduktion wird häufig auf Low-Poly-Modelle zurückgegriffen. Das heißt, dass bei der Modellierung die Anzahl der Polygone, so gering wie möglich gehalten werden. Hierdurch verlieren die Modelle zwar an Detail, die Darstellungsgeschwindigkeit wird aber erhöht. Die fehlenden Details können teilweise im Nachhinein mit Hilfe einer Normal-Map (siehe Abschnitt 3.2.1) ausgeglichen werden.

Auch für den nächsten Arbeitsschritt hat der 3D-Artist zwei Möglichkeiten. Entweder wird zuerst das Low-Poly-Modell erstellt und aus einer Kopie dessen dann ein High-Poly-Modell, oder es wird direkt das High-Poly-Modell erstellt. Ein High-Poly-Modell ist ein Asset mit einer hohen Auflösung, in Bezug auf die genutzten Triangles. Bei letzterer Variante wird im Nachhinein durch Retopologie ein Low-Poly-Modell erzeugt.

In dieser Arbeit wird der erste Ablauf zur Erstellung der 3D-Modelle genutzt.

Es empfiehlt sich, die gewünschten Formen zuerst mit primitiven Objekten nachzubilden, die dann z. B. durch Unterteilen, Auftrennen, Zusammenfügen und Verformen[5] so angepasst werden können, dass das Modell den Vorgaben entspricht. Die Möglichkeiten der Mesh-Bearbeitung hängen vom genutzten 3D-Tool ab, aber alle Tools decken prinzipiell die gleichen Funktionen ab.

[5] Dies ist nur eine Auswahl der Möglichkeiten, die zur Manipulation von Meshes zur verfügen stehen. Eine ausführliche Liste bietet das Blender Manual (Blender Documentation Team, o.D.) im Kapitel Mesh Editing.

In wie weit die Details der Vorgaben nachgebildet werden, hängt vom Stil, den Fähigkeiten des 3D-Artist und dem Budget[6] des Models ab.

Mit der Low-Poly Variante können schon erste Tests in der Engine durchgeführt werden, wie beispielsweise die Überprüfung der Vorgaben und Regeln des Projektes. Funktioniert das Modell wie geplant, kann eine Kopie, die als Basis der High-Poly-Variante dient, angelegt werden. Funktionen und Tools wie der Remesh Modifier aus Blender (vgl. Abbildung 3.2) können gleichmäßige Verteilungen von Flächen in Meshes erzeugen. Diese Gleichmäßige Verteilung ist wichtig um beim sogenannten Sculpting und auch sonstigen weiteren Anpassungen des Meshes einen konstanten Detailgrad zu erhalten (Burrows, 2015).

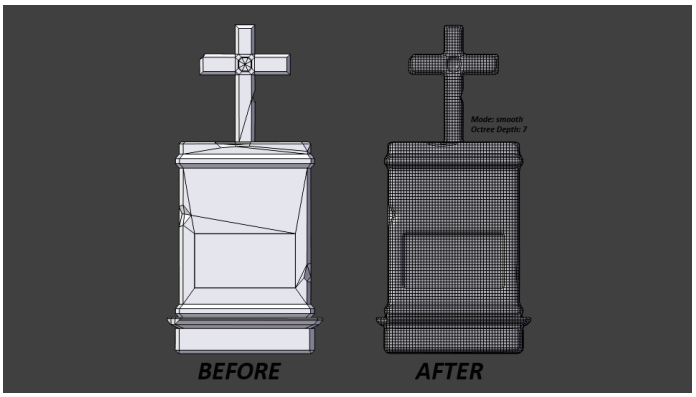


Abbildung 3.2: Ein Grabstein vor und nach dem Anwenden des Blender Remesh Modifierers (Burrows, 2015).

Neben der Technik das ganze Mesh so oft zu unterteilen, bis es den gewünschten Detailgrad abbilden kann, ist es auch möglich mit dynamischer Topologie zu arbeiten. Bei diesem Verfahren wird nur an den modifizierten Stellen des Objekts neue Topologie erschaffen. So kann die Anzahl der Triangles des Meshes relativ gering gehalten werden.

Unabhängig von der gewählten Methode kann das Mesh anschliessend, wie Ton bearbeitet werden. Dies ermöglicht organische und detailreiche Modelle (Beck, 2017, S. 172-189).

3.2.1 Texturierung

In diesem Abschnitt wird auf Möglichkeiten eingegangen das äußere Erscheinungsbild eines Models zu verändern. Dies ist mit Hilfe von Texturen und Materialien möglich. Es werden verschiedene Arten von Texturen und ihr Einfluss auf Modelle erläutert.

Die Darstellung des Models ist abhängig von dem oder den zugewiesenen Materialien. Materialien (englisch Materials) sind in allen 3D-Tools und Engines verschieden aufgebaut, funktionieren im Prinzip aber gleich. Es gibt Slots, die mit verschiedenen Texturen gefüllt werden können, und Parameter, mit denen die Erscheinung des Materials weiter beeinflusst werden kann.

[6] Das „Budget“ bezeichnet die Menge an Ressourcen, die ein Objekt oder eine Szene verbrauchen darf, ohne dass die Performance des Spiels ungewollt beeinflusst wird.

Neben den „normalen“ Materialien existieren Physically Based Rendering (PBR) beziehungsweise Physically Based Shading (PBS) Materialien, der Einfachheit halber wird ab hier PBR als Überbegriff genutzt. PBR-Materialien erlauben es, annähernd physikalisch korrekte Oberflächen darzustellen. Hierfür werden Texturen mit Messwerten echter Materialien benötigt (Beck, 2017, S. 490-492). Dies verringert die für die Material-Erstellung benötigte Zeit, weil mögliche schrittweise Annäherung entfallen (Beck, 2017, S. 490-492). Des Weiteren sind PBR-Systeme in allen PBR unterstützenden 3D-Anwendungen ähnlich aufgebaut, wodurch sie unabhängig von der Anwendung gleich aussehen, ohne einer speziellen Konfiguration zu bedürfen (Beck, 2017, S. 490-492).

Im Folgenden werden die geläufigsten Texturen aufgelistet und ihre Funktion erklärt.[7] Zudem wird darauf eingegangen, ob es bei der jeweiligen Textur Unterschiede für PBR und nicht PBR gibt.

Diffuse / Albedo Map

Die Diffuse, (bzw. bei PBR-Materialien Albedo) Map, sorgt für die grundlegende Farbe des Modells. Für nicht PBR Materials sind in dieser Textur neben der Farbe auch Schattierungen und Ambient Occlusion enthalten (Beck, 2017, S. 495). Ambient Occlusion oder Umgebungsverdeckung erzeugt an Kanten und anderen Flächen mit geringem Abstand Verdunkelungen. Dies beruht auf dem Prinzip, dass weniger Licht an Orte gelangt, die wenig Platz für Streuung zulassen (Beck, 2017, S. 362). Diese Texturen können durch direktes Zeichnen auf dem Modell in einem 3D-Tool oder in einem externen Grafikprogramm wie *Adobe Photoshop* angefertigt werden.

Normal Map

Enthält ein Material Normal Maps, kann die äußere Erscheinung des Models verändert werden, ohne seine Geometrie anzupassen. So ist es möglich, Low-Poly-Modelle hochwertiger und teilweise sogar wie High-Poly-Modelle aussehen zu lassen. Die Normal Map manipuliert die Ausrichtung der Normals auf den Faces und kann so beliebige Oberflächenstrukturen vortäuschen, auch wenn das Modell eine glatte Oberfläche besitzt. Jeder Pixel der Map repräsentiert einen, über einen RGB-Wert definierten Richtungsvektor, der mit den bestehenden Normals des Models verrechnet wird. Durch diese Manipulation wird die Interaktion von Licht und Schatten mit der Oberfläche des Objektes verändert. (Beck, 2017, S. 421-422)

Bump Map

Bump Maps ähneln in ihrer Funktion den Normal Maps. Auch sie ändern das Erscheinungsbild des Models, ohne die Geometrie zu verändern. Der Effekt wird bei Bump Maps durch Graustufen erreicht. Schwarz stellt eine maximale Vertiefung und Weiß eine maximaler Erhöung dar, 50-prozentiges Grau lässt die Oberfläche unverändert. (Beck, 2017, S. 420-421)

[7] Eine Übersicht mit Beispielen und weiteren Maps ist auf dem Youtube Kanal von FlippedNormals, in dem Video Texture Maps Explained (FlippedNormals, 2019), zu finden.

Displacement Map

Das letzte Beispiel für Texturen, welche die Form der Models verändern, ist die Displacement Map. Diese verändert im Gegensatz zu den vorherigen Maps wirklich die eigentliche Geometrie des unterliegenden Objektes. Sie arbeitet wie die Bump Map mit Graustufen, Weiß führt zu einer maximalen Erhöhung und Schwarz zu einer maximalen Vertiefung, 50-prozentiges Grau steht wieder für keine Veränderung. Für eine akkurate Abbildung der Details aus der Textur benötigt das Mesh eine hohe Auflösung, folglich ist diese Methode für Spiele eher ungeeignet. Sie wird hauptsächlich für den Erstellungsprozess genutzt, um dem Modell weitere Details hinzuzufügen, die später in eine Normal Map übertragen werden. (Beck, 2017, S. 422-423)

Specular Map

Specular Maps werden für nicht PBR-Materialien genutzt, um zu definieren, an welchen Stellen das Modell Glanzpunkte erzeugen kann. Auch die Specular Map ist in Graustufen aufgebaut. Die Stärke des Glanzes wird durch den Grauwert bestimmt: schwarze Stellen glänzen gar nicht und weiße am stärksten. (Beck, 2017, S. 423)

Für PBR-Materialien werden Specular Maps nur in Sonderfällen genutzt. Eis, Wasser und Haut sind Beispiele hierfür. Die Map wird hier genutzt, um die richtige Lichtbrechung zu definieren (Hider, 2017)

Roughness / Gloss Map

Für PBR-Materialien werden Roughness Maps genutzt, um raue und glatte Bereiche auf einem Modell zu erzeugen. Roughness und Gloss Maps sind im Prinzip das Gleiche, aber die Auswertung des Helligkeitswerts der Textur ist invertiert. Bei einer Gloss Map repräsentiert ein heller Pixel eine sehr glatte Fläche, bei einer Roughness Map wäre die Fläche sehr rau und würde nur wenig bis keine Reflexion erzeugen. (Beck, 2017, S. 496-497)

Texturen können nur auf ein Modell angewendet werden, wenn dieses entsprechend vorbereitet wurde. Dafür muss es den Prozess des UV-Unwrappings durchlaufen. Hier wird das Mesh aufgetrennt und auf eine zweidimensionale Fläche ausgebreitet. In den meisten 3D-Tools gibt es mehrere Möglichkeiten, das Mesh automatisch aufzutrennen, in der Regel ist es aber besser, diesen Vorgang manuell durchzuführen. Hierfür werden die Edges, an denen das Modell zerschnitten werden soll, markiert und im Anschluss der Unwrap-Vorgang gestartet, der das UV-Layout erzeugt. Über das UV-Layout wird definiert, welcher Ausschnitt einer Textur auf welcher Fläche des Meshes abgebildet wird. (Beck, 2017, S. 445-455)

Texturen wie die Normal Map können „gebacken“ werden. Für diesen Vorgang werden eine High- und eine Low-Poly Version des gleichen Models benötigt (Beck, 2017, S. 422). Die Low-Poly- Version auf welche die Details des High-Poly-Meshes gebacken werden, muss unwrapped sein, damit die Details auf eine 2D-Ebene projiziert werden können (Beck, 2017, S. 513-518). Während des „Backens“ werden die Oberflächennormalen des High-Poly-Models in die zugehörigen RGB-Werte übersetzt und in einer Normal Map gespeichert (Beck, 2017, S. 422).

Für Spiele wie zum Beispiel *World of Warcraft* (Blizzard Entertainment, 2004) werden nur Low-Poly- Modelle mit Diffuse Texturen genutzt, die nur die Farbe des Modells enthalten (Vergne, 2017). Für die Erstellung dieser Texturen werden High-Poly-Meshes genutzt, deren Details dann auf verschiedene Texturen gebacken werden und anschließend in einer Diffuse Map zusammen geführt werden (Vergne, 2017). Mit dieser Methode wird ein spezieller Look erzeugt und die Speichernutzung reduziert.

Mit den sogenannten 3D-Zeichenprogrammen, wie beispielsweise *Substance Painter* , können Materialien mit allen benötigten Texturen auf 3D Modelle angewendet werden. So können bestehende Materialien manipuliert oder neue angelegt werden. Die Materialien können auf die Beschaffenheiten des Modells eingehen und zum Beispiel Verschleiß an Kanten simulieren. So können in kurzer Zeit detailreiche, realistische Materialien erzeugt werden.

Alle Materialien müssen vor dem Export aus dem 3D-Tool allen zugehörigen Faces des Objekts zugewiesen sein und die Quads des Models sollten in Triangles übersetzt werden („Triangulation“). Game-Engines arbeiten nur mit Triangles und übersetzen das Modell selber in Triangles wenn es aus Quads und N-Gons besteht. Bei diesem Vorgang kann es zu Fehlern mit den Flächennormalen kommen. Wird die Triangulation vor dem Export durchgeführt, sind diese Fehler vermeidbar.

3.3 Implementierung

Für die Erstellung eines Videospiels mit den angefertigten Assets wird eine Game-Engine wie z. B. UE benötigt. Für ein performantes Spiel und eine fehlerfreie Darstellung müssen tiefgreifende Aspekte der Engine verstanden werden. Daher wird im Folgenden auf die Funktionsweise von Realtime Rendering eingegangen, welches für die Darstellung in Spielen genutzt wird.

Alle in der Engine verwendeten Modelle werden in einen Verbund aus Dreiecken übersetzt, weil Grafikkarten (GPUs) mit Dreiecken rechnen (O’Conor, 2017). Jedes Mesh wird in der Game Engine in zwei Teilen abgespeichert, dem Vertex Buffer und dem Index Buffer (O’Conor, 2017). Der Vertex Buffer enthält eine Liste aller Vertices des Meshes (O’Conor, 2017). Dazu gehören Position des Vertex, UV-Koordinaten, Ausrichtung der Normals und weitere (O’Conor, 2017). Der Index Buffer enthält Informationen darüber, welche Vertices miteinander Dreiecke bilden und so das Mesh formen (O’Conor, 2017). Zusätzlich zu dem Index und Vertex Buffer werden dem Mesh ein oder mehrere Materialien zugewiesen (O’Conor, 2017).

Die Materialien werden in der Engine selbst erstellt und mit den vorgesehenen Texturen konfiguriert. Sie können nicht aus den 3D-Tools, mit denen das Mesh erstellt wurde, übernommen werden. Die Material-Systeme der Engines und die der 3D-Tools sind in der Regel nicht kompatibel.

Für die Darstellung des Meshes auf dem Bildschirm wird für jeden Frame eine Folge von Schritten durchlaufen, die sich wie folgt zusammenfassen lassen:

1. Im ersten Schritt, dem *Input Assembly*, werden anhand des Index Buffers die Vertices aus dem Vertex Buffer miteinander zu Dreiecken verbunden.

2. Dann findet das *Vertex Shading* statt, bei dem die aktuelle Ausrichtung und Position des Meshes in Relation zur Kamera mit allen Vertices verrechnet wird. So wird die Gesamtansicht bestimmt und welche Teile des Objektes zu sehen sind.

3. Die *Rasterization* erzeugt aus jedem Triangle eine gerasterte Fläche, über die die Vertex-Normalen, UV-Koordinaten und weitere Eigenschaften der Vertices interpoliert werden.

4. Der Schritt des *Pixel Shading* ordnet jedem Pixel in der gerasterten Fläche abhängig von Material, Licht und anderen Einflüssen eine Farbe zu.

5. Zuletzt wird *Render Target Output* durchgeführt, um die berechneten Werte für die Darstellung weiterzuleiten. Es gibt einen Zwischenschritt, der überprüft, ob die errechneten Pixel angezeigt werden sollen. Es wird unter anderem überprüft, ob Pixel nicht sichtbar sind, weil das zugehörige Objekt von einem anderen Objekt verdeckt wird oder ob ein Pixel durchsichtig ist. Dann wird der dahinterliegende Pixel genutzt.
   (O'Conor, 2017)

Das Rendering ist damit abgeschlossen.

Die Grafikkarte wird bei der Bearbeitung von Bildern von der CPU unterstützt. Diese sendet Draw Calls an die Grafikkarte und übergibt der Grafikkarte die benötigten Daten, um die Draw Calls auszuführen. Ein Draw Call enthält Information über das zu rendernde Mesh, sein Material und seine sonstigen Eigenschaften. Jeder Draw Call wird vom Grafikkarten-Treiber übersetzt und auf Fehler überprüft. Dadurch entstehen für jeden Draw Call Grundkosten, die sich in verbrauchter Zeit widerspiegeln. Für jedes Material wird ein eigener Draw Call abgesetzt; das heißt, wenn ein Mesh mit drei Materials ausgestattet ist, löst es auch drei Draw Calls aus.

Weil die Grafikkarte von der CPU abhängig ist und die CPU jeden Draw Call durcharbeiten muss, haben diese einen direkten Einfluss auf die Framerate. Die Anzahl an Draw Calls sollte folglich gering gehalten werden. (O'Conor, 2017)

In der Engine kann der Aufbau eines Objektes nicht geändert werden, lediglich Skalierung und Rotation können angepasst werden. Das hat aber keinen Einfluss auf die Struktur der Vertices, Faces und Edges. Objekte können für leichtere Handhabung und das Verringern von Draw Calls mit einander verknüpft werden. Dies hat keinen Einfluss auf die Struktur der verbundenen Objekte.

Moderne Spiele können nur die, von der Spielerkamera erfassten, Objekte rendern. Dieser Prozess nennt sich auch Occlusion Culling. Deshalb sollten Objekte nicht unbegrenzt verbunden werden. Ist nur ein kleiner Teil des Objektes zu sehen,

wird dieses komplett gerendert und sorgt so für einen weiteren kompletten Draw Call, wodurch gegebenenfalls Performance verschwendet wird. Ob ein Objekt zu sehen ist, wird am Anfang des Prozesses mit einer Bounding Box überprüft[8]. Es kann dann passieren, dass zwar die Bounding Box gesehen wird, das Objekt an sich aber nicht. So würden wieder unnötige Draw Calls gestartet werden. (O'Conor, 2017)

Eine Möglichkeit Draw Calls weiter zu verringern, ist das wiederholte Benutzen des gleichen Objektes mit *Instancing*. Aktuelle Engines erkennen dies automatisch. Es wird dann nur ein Draw Call abgesetzt, der die verschiedenen Konfigurationen, Position, Rotation etc. der einzelnen Objekte beinhaltet. (O'Conor, 2017)

Durch das Verbinden von Texturen zu einer größeren Textur, einem so genannten Textur Atlas, können die Draw Calls weiter reduziert werden. Mit Textur-Atlanten können mit einem Material verschiedene Oberflächen dargestellt werden. So können mehrere Objekte mit dem gleichen Material ausgestattet werden, für das aber nur ein Draw Call erzeugt wird. (O'Conor, 2017)

Die Grundlagen der Asset-Erstellung und -Implementierung sind hiermit abgeschlossen. Im folgenden Kapitel wird auf die Anforderungen und mögliche Vor- und Nachteile von modularen Assets eingegangen. Des Weiteren werden Methoden, zur Erstellung und Verwendung modularer Assets behandelt.

[8] Eine Bounding Box ist ein unsichtbarer Quader um die maximalen Maße des Objektes.

4 Generierung modularer Assets

Für die Bewertung der Ergebnisse dieser Arbeit werden im nächsten Kapitel Qualitätsmerkmale modularer Assets aufgestellt. Der zweite Teil des Kapitels zeigt exemplarisch, inwiefern diese Kriterien erfüllbar sind.

4.1 Anforderungen an modulare Assets

Modulare Assets müssen für den Einsatz in realen Videospiel-Produktionen verschiedene Eigenschaften erfüllen. Diese werden in diesem Abschnitt erläutert.

4.1.1 Visuelle Qualität

Ein wichtiger Aspekt bei der Entwicklung von Assets ist ihre visuelle Qualität. Sie müssen den restlichen Ansprüchen des Spiels genügen und zu dessen Ästhetik passen.

Insbesondere bei modularen Assets wird dies zu einem Problem, da diese dazu neigen, Elemente häufig zu wiederholen. Man spricht hier von der sogenannten Kunstermüdung (*Art Fatigue*). Sie führt dazu, dass Spieler sich langweilen und das Gefühl haben, bereits alles gesehen zu haben. (Burgess & Purkeypile, 2013)

Weiterhin führt die Repetition modularer Assets schnell zu einer Art Orientierungslosigkeit, bei der Spieler nicht mehr genau bestimmen können, wo sie sich gerade befinden. Grund hierfür ist der Mangel eindeutig identifizierbarer Orientierungspunkte. (Lin, Lentz, Sturgill & Reed, o.D.).

Zudem gilt es bei der Konzeption modularer Kits darauf zu achten, dass keine Lücken oder Überlagerungen entstehen, da diese zu grafischen Fehlern führen (Mader, 2005). Sollte es Teile in einem Kit geben, die Lücken zulassen, muss es passende Gegenstücke geben, die diese wieder schließen können (Burgess & Purkeypile, 2013).

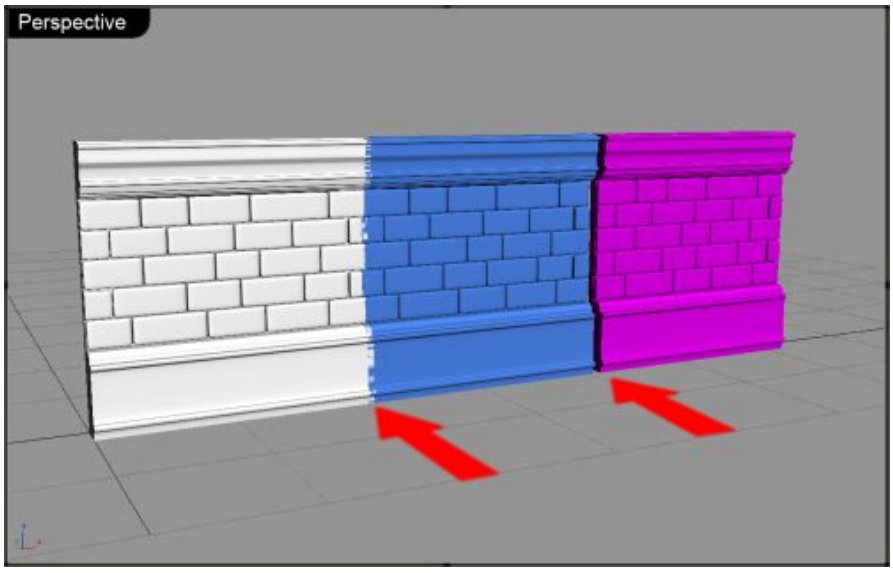


Abbildung 4.1: Beispiel für Überlappung und Lücken zwischen Modulen (Mader, 2005).

4.1.2 Nutzerfreundlichkeit

Damit ein modulares Kit effizient nutzbar ist, muss das Arbeiten mit diesem so angenehm und einfach wie möglich sein. Hierbei ist sehr wichtig, dass sich die einzelnen Teile im Editor einfach bewegen und verbinden lassen. Nach Möglichkeit sollten diese automatisch einrasten, sobald sie nah genug zusammengeführt werden. (Mader, 2005).

Auch die Flexibilität trägt erheblich zur Nutzerfreundlichkeit bei. Hierzu zählt, wie kompatibel die einzelnen Teile untereinander sind und wie viel Freiheit sie den Level-Designern lassen. (Lin et al., o.D.).

Zusätzlich lässt sich die Nutzerfreundlichkeit daran messen, wie schnell und einfach größere Strukturen erstellt werden können. (Mader, 2005).

4.1.3 Aufwand

Kosten spielen bei der Erstellung von Videospielen, wie in jeder Branche, eine große Rolle. Um diese möglichst gering zu halten ist es wünschenswert, den benötigten Arbeitsaufwand und die Zeit zu reduzieren. Effektiv entworfene modulare Assets sind hierfür eine gute Hilfe.

In *Skyrim* wurden über 300 Dungeons[9] von nur zehn Personen entwickelt, darunter nur zwei 3D-Artists. Der Einsatz modularer Assets erlaubte dem Team innerhalb der Entwicklungszeit eine große Menge diverser Spielinhalte zu erstellen. (Burgess & Purkeypile, 2013)

Ein weiteres Beispiel ist der bereits in Abschnitt 2.2 erwähnte Einsatz von Modularität in *For Honor*, bei dem diese genutzt wird, um schnell und einfach Level-Design-Iterationen durchzuführen. (Durand, 2019)

Zwar benötigt die Planung modularer Assets vorab mehr Zeit als die Planung herkömmlicher Assets, jedoch ist die Zeitersparnis in der Design-Phase um einiges größer (Meler, 2018). Ein gutes modulares Asset muss demnach flexibel genug sein, um Level-Design-Iterationen zu beschleunigen.

4.1.4 Performance

Durch das bereits erwähnte *Instancing* (vgl. Abschnitt 3.3) können modulare Assets helfen, die Performance des Spiels zu verbessern und somit eine höhere Bildfrequenz zu erreichen.

Entscheidend hierfür sind die Modularitätsstufen (vgl. Abschnitt 4.2.2) und die Wiederverwertbarkeit der Assets, damit das Instancing effektiv genutzt werden kann.

[9] Ein Dungeon ist ein in sich geschlossenes Level, welches einen Abschnitt der Spielwelt repräsentiert. Das kann z. B. ein verlassener Tempel, eine Höhle oder ein Verlies sein

4.1.5 Zusammenfassung

Die Qualität modularer Assets lässt sich über mehrere Kriterien bewerten. Für den weiteren Verlauf der Arbeit sind die genannten Kriterien ein ausschlaggebendes Bewertungsmaß der erstellten Assets. Hier eine Übersicht der aufgestellten Kriterien:

- **Visuelle Qualität**: Lücken und Überlappungen sollten vermieden werden. Um die Kunstermüdung zu vermeiden, sollte das Kit über ausreichend Abwechslung oder musterbrechende Teile verfügen. Starke Wiederholungen sollten vermieden werden.
- **Nutzerfreundlichkeit**: Ein gutes modulares Kit ermöglicht einfaches Arbeiten. Zusammenfügen und Bewegen der Teile ist einfach und intuitiv. Die Teile erlauben den Bau größerer Strukturen ohne großen Mehraufwand. Das Kit ist untereinander flexibel und kompatibel.
- **Aufwand**: Die Assets helfen in der späteren Projektphase Level Design Iterationen zu verkürzen und ermöglichen die Erstellung einer Vielzahl von Inhalten mit möglichst wenigen Assets.
- **Performance**: Das modulare Kit kostet nicht mehr Performance, als ein herkömmliches und erzielt bestenfalls durch Instancing eine noch bessere Performance.

4.2 Methoden für die Generierung modularer Assets

Um die Planung und die Produktion modularer Kits so effektiv wie möglich zu gestalten, sollten bestimmte Regeln und Vorgehensweisen beachtet werden. Im Folgenden werden die wichtigsten davon erläutert.

4.2.1 Projektübergreifende Planung

Planung ist nach Klafke und Meler einer der wichtigste Aspekt bei der Entwicklung modularer Assets (Klafke, 2010; Meler, 2018).

Sie kann in zwei Stufen eingeteilt werden. Während die erste der beiden Phasen nur einmal im Verlauf des gesamten Projektes durchgeführt werden muss[10], muss die zweite für jedes neue Kit im Projekt ausgeführt werden. (Burgess & Purkeypile, 2013)

In der ersten Planungsphase werden grundlegende Eigenschaften der angestrebten Modularität festgelegt. Hierzu gehören die verwendeten Modularitätsstufen, die Abstimmung einheitlich verwendeter Rasters (Perry, 2002) und das Setzen der Pivot Points (Mader, 2005). Auf Grund ihrer Bedeutung und Komplexität werden diese im Folgenden detailliert beleuchtet. Sollten alle erstellten Kits diesen gemeinsamen Regeln folgen, sind sie auch untereinander kombinierbar und können somit für mehr Diversität sorgen (Burgess & Purkeypile, 2013).

4.2.2 Modularitätsstufen

Entsprechend des Grades ihrer Aufspaltung in Einzelteile teile L. Perry modular angelegte Assets in verschiedene Stufen (im Original „Scales“) ein (Perry, 2002).

Die verwendeten Modularitätsstufen sollten zu Beginn der Planung festgelegt werden, damit die folgenden Schritte einfacher konkretisiert werden können (Lin et al., o.D.). Die gewählte Stufe ist abhängig von der Spielwelt und dem geplanten Grad der Modularität (Lin et al., o.D.). Wurden diese festgelegt, können die benötigten Gebilde leichter in Module unterteilt werden, die der benötigten Stufe entsprechen. Die Ausprägungen der Stufen werden jetzt erläutert.

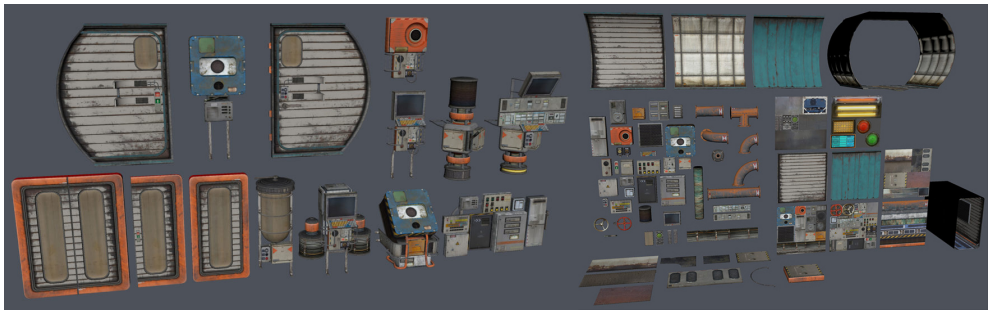


Abbildung 4.2: Module für den Innenraum einer Raumstation. In Anlehnung an (Olson, 2018).

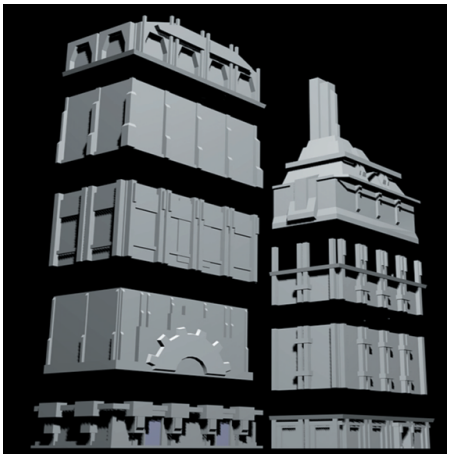
Für Innenräume empfiehlt sich ein stark granulares Kit, damit Wände, Böden und andere Teile der Umgebung detail- und abwechslungsreich zusammengesetzt wer-

[10] Dies gilt unter der Annahme, dass alle Kits nach den gleichen Regeln erzeugt werden. Andernfalls muss auch die erste Phase für jedes Kit wiederholt werden.

den können. Beispiel für einen solchen Innenraum wäre eine verlassene Raumstation, die nur schleichend erkundet wird (Perry, 2002).

Abbildung 4.3a zeigt ein modulares Kit mit mittlerer Modularitätsstufe. Jedes Element aus dem Kit ist eine eigene Wand, mit dessen Hilfe ein Gebäude erstellt werden kann. Eine Mischung aus einer solchen Modularitätsstufe und einer kleineren wurde von L. Durand für *For Honor* genutzt (vgl. Abbildung 2.6).

(a) Übersicht eines Kits mit mittlerer Modularität.



(b) Übersicht eines Kits mit geringer Modularität.

Abbildung 4.3: Beispiele für verschiedene Modulraitätsstufen eines Gebäude Kits (Perry, 2002).

Ein Beispiel für eine geringe Modularitätsstufe zeigt 4.3b. Mit Hilfe dieses Kits können Gebäude aus verschiedenen, vorgefertigten Etagen zusammengesetzt werden. Sie erlauben schnell eine Vielzahl unterschiedlicher Gebäude zu erstellen, jedoch sind diese aufgrund der geringen Modularität recht einseitig.

In der nächstgrößeren Stufe würden ganze Häuser genutzt, um daraus Häuserblöcke zu errichten. Eine weitere Steigerung wäre aus diesen Häuserblöcken eine Stadt zu errichten. Diese Modularitätsstufe eignet sich beispielsweise für den Einsatz in einem Rennspiel. (Perry, 2002)

Innerhalb eines Projekts können durchaus verschiedene Stufen verwendet und gemischt werden. Besitzt ein Spiel beispielsweise offene und geschlossene Bereiche, so ist es sinnvoll, deren Kits unterschiedlich modular zu gestalten. Das gleiche gilt für Bereiche, die der Spieler aus verschiedenen Distanzen sieht. So könnten bei einem Fußweg durch die Stadt beispielsweise die Häuserreihen weniger modular sein als die Straße, über die der Spieler läuft, da letztere deutlich präsenter und näher ist. (Klafke, 2010)

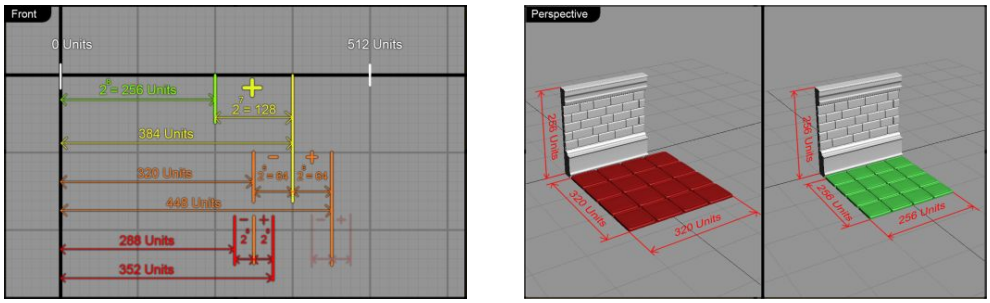
Beim Arbeiten mit nur einer einzigen Stufe kann die Repetition durch die manuelle Platzierung kleinerer Objekte aufgebrochen werden (Perry, 2002).

Die Modularitätsstufen sollten zudem projektspezifisch betrachtet werden. Für ein Rennspiel im Weltall, dessen kleinste Einheit Raumschiffe sind, ist es wenig sinnvoll die gleichen Modularitätsstufen zu nutzen wie für ein Fantasy-Rollenspiel mit detailliertem Interieur. Die Einteilung der Stufen ist immer auch abhängig von der Spielwelt und Perspektive.

4.2.3 Raster

Nach L. Durand ist einer der wichtigsten Aspekte, für die Umsetzung eines Projekts mit modularem Design, das Raster (Durand, 2019).

Jedes 3D-Tool arbeitet intern mit einem Raster[11]. Sie können sich in der verwendeten Maßeinheit (Meter, Zentimeter, Zoll, etc.), der Orientierung des Koordinatensystems, den Unterteilungsstufen und anderen Eigenschaften unterscheiden. Im Raster ist es 3D-Artists möglich, Objekte punktgenau im Raum zu platzieren. Dies ermöglicht die Erstellung modularer Assets mit lückenlosen Übergängen. (Mader, 2005)



(a) Beispielhafte Kalkulation der Raster-Einteilungen.

(b) Beispiel für die Angleichung von Modulen.

Abbildung 4.4: Beispiele für die Anwendung verschiedener Rastergrößen (Mader, 2005).

Das Raster sollte möglichst in Stufen von Zweierpotenzen eingeteilt sein. Nur durch das Nutzen von geraden Zahlen kann das Raster in kleinere Stufen aufgeteilt werden. Dies kann für kleinere Objekte nötig sein. Kann die Größe eines Elements nicht durch die standardmäßig genutzte Zweierpotenz dargestellt werden, ist es möglich, kleinere Zweierpotenzen zu addieren oder subtrahieren. Abbildung 4.4a zeigt Beispiele für mögliche Einteilungen des Rasters. (Mader, 2005)

Zu kleine Einteilungen des Rasters, erhöhen jedoch den Aufwand des Zusammenbaus von Elementen. Die Module müssen dann in immer kleineren Schritten über das Raster bewegt werden, wodurch das Einrasten erschwert wird. (Mader, 2005)

Alle Elemente, die untereinander kompatibel sein sollten, wie beispielsweise Wände, Böden und Decken, sollten die gleichen Grundmaße besitzen, da sie ansonsten nur schwer untereinander kombinierbar sind. (Mader, 2005)

Abbildung 4.4b illustriert dieses Problem: Die Wand und die rote Bodenfläche sind untereinander nicht kompatibel, ohne dass eines der beiden Elemente größer bzw. kleiner skaliert oder fünf Wand- und vier Bodenelemente benutzt werden. Eine solche Inkompatibilität sorgt für erhöhten Zeitaufwand und eine Einschränkung der kreativen Freiheit der Level-Designer. So verliert sich der Mehrwert der angestrebten Modularität.

Burgess und Purkypile bezeichnen die Grundmaße eines Kits als Footprint (deutsch Fußabdruck). Passt ein Element nicht in den definierten Fußabdruck, sollte es entweder ein Vielfaches oder einen Teiler der Größe des Fußabdruck als Größe nutzen (Burgess & Purkeypile, 2013).

[11] Das Raster ist hier ein dreidimensionales Koordinatensystem.

Beispielsweise könnte die grüne Bodenfläche aus Abbildung 4.4b auch die Größe 128x128 besitzen. Dann müssten vier Elemente genutzt werden, um die Fläche vor einer Wand zu füllen, die gewünschte Modularität bliebe zu einem gewissen Grad erhalten.

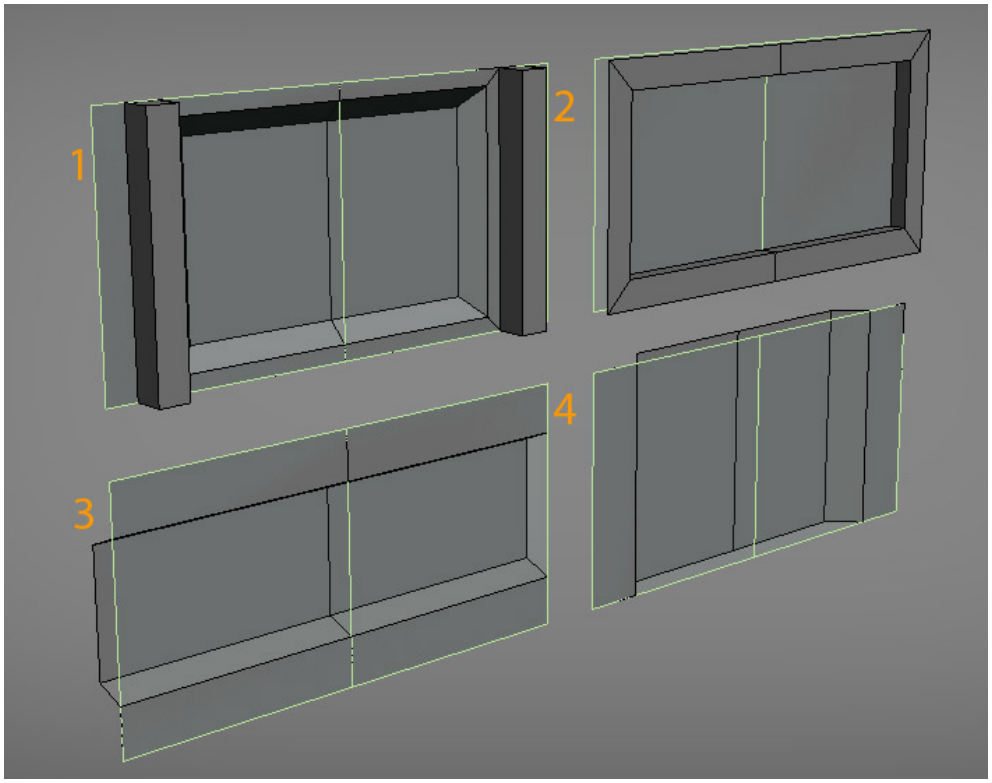


Abbildung 4.5: Beispiel für verschiedene Qualitäten von Kontaktpunkte (Klafke, 2010).

Thiago Klafke arbeitet mit *Bounding Boxes*. Alle Module in seinem Kit müssen in eine vorgegebene Box passen. Sie muss nicht in alle Richtungen gefüllt sein, aber ihre Kanten sollten nach Möglichkeit soweit ausgearbeitet sein, dass keine Lücken zwischen den Modulen entstehen. Haben Module eine Tiefe, sollten alle Elemente an den Rändern der *Bounding Box* auf der gleichen Ebene liegen. Abbildung 4.5 verdeutlicht dies: Werden z. B. die Elemente 1 und 3 horizontal mit einander verbunden entstehen Lücken. Modul 2 ist in diesem Fall weder mit 1 noch mit 3 oder 4 kombinierbar, da es mit keinem dieser Module verbunden werden kann, ohne Lücken zu erzeugen. (Klafke, 2010)

Die Einteilung der Raster-Größe und seiner Aufteilung sollten abhängig von der Größe der Spielfigur, der Animationen und der Bewegung des Spielers in der Welt entschieden werden (Perry, 2002). Wenn vorhanden, muss auch die Interaktion von Computer gesteuerten Charakteren mit der Umwelt mit einbezogen werden (Burgess & Purkeypile, 2013). *Skyrim* beispielsweise nutzt als minimale Größe eines begehbaren Elements die Breite von zwei Charakteren, wodurch sichergestellt wird, dass sich alle Figuren frei bewegen können (Burgess & Purkeypile, 2013).

Um die Fehleranfälligkeit zu verringern und den Arbeitsprozess zu beschleunigen, sollten die unterschiedlichen Rasters aller verwendeten 3D-Tools aneinander angeglichen werden. Diese Angleichung sollte optimalerweise direkt im Anschluss an die Konzeption des Rasters erfolgen. (Durand, 2019)

4.2.4 Positionierung und Nutzen des Pivot Points

Der Pivot Point (Drehpunkt) oder auch der Origin Point (Ursprung) kann beliebig für jedes Objekt gesetzt werden (Beck, 2017, S. 85). Über ihn wird definiert, wo sich das Objekt im Raum befindet und wie sich Rotation, Skalierungen und Spiegelungen darauf auswirken (Beck, 2017, S. 85 & S.102).

Ohne korrekt gesetzten Pivot Point wird die zielgenaue Platzierung eines Objektes im dreidimensionalen Raum erschwert bzw. ist nur eingeschränkt möglich. Ein falsch positionierter Pivot Point schränkt die Vorteile der Modularität ein und beeinflusst den Workflow negativ. (Mader, 2005)

Im Folgenden werden verschiedene Beispiele gezeigt, anhand derer Regeln zur korrekten Platzierung des Pivot Points, für jedes Objekt bzw. Objektgruppe, abgeleitet werden können. Sind Objekte auf einer oder mehreren Achsen symmetrisch, sollte der Pivot Point am Ursprung des lokalen Koordinatensystems des Objekts platziert werden. Dadurch kann mit diesem in der Engine freier gearbeitet werden. (Mader, 2005)

Wie zu fast allem gibt es auch zu der Platzierung des Pivot Points einige fallspezifische Ausnahmen.

Zusätzlich muss beachtet werden, dass eine nachträgliche Änderung des Pivot Points dazu führt, dass jede Instanz des Objektes im Spiel angepasst werden muss, da z.b. eine Translation des Pivot Points auch zu einer Translation des Objektes in der Spielwelt führt. (Burgess & Purkeypile, 2013)

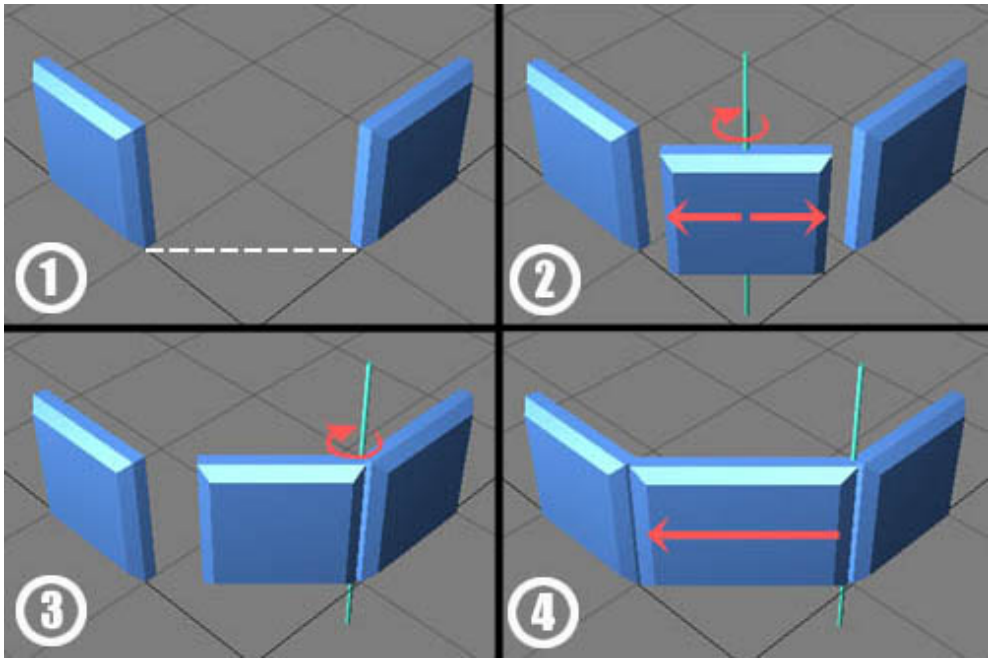


Abbildung 4.6: Beispiel für den Einsatz eines am Ende einer Mauer platzierten Pivot Points (Lin, Lentz, Sturgill & Reed, o.D.).

Bei Objekten, die entlang einer Achse wiederholt werden sollen, sollte der Pivot Point möglichst an einem der beiden Enden der Achse positioniert werden (Abbildung 4.6). (Mader, 2005)

Für sich wiederholende Objekte sollte der Pivot Point bei allen Objekten der gleichen Art auf der gleichen Seite der entsprechenden Achse liegen, damit eine einfache Verbindbarkeit der Module gewährleistet ist (Mader, 2005).

Ein weiteres Beispiel, warum es von Vorteil ist, für Mauer- oder Wandstücke den Pivot Point an einer Seite zu haben, ist das Überbrücken einer Lücke mit einem Winkel. Abbildung 4.6 stellt diesen Fall dar. Wäre der Pivot Point in der Mitte des Moduls, wäre das Anpassen an die Größe der Lücke und das Setzen an die richtige Position mit viel Aufwand verbunden. (Perry, 2002)

Bei, auf dem Boden oder einer anderen Fläche platzierten, Elementen, sollte der Pivot Point möglichst an der Unterseite des Objektes positioniert werden, damit diese bei aktivierter Einrastfunktion auch wirklich mit dem Grund der Spielwelt abschließen. Durch diese Technik können spätere händische Eingriffe verhindert werden. Außerdem ändern diese Objekte dadurch auch nicht ihre Position auf dem Boden, wenn sie skaliert werden. (Mader, 2005)

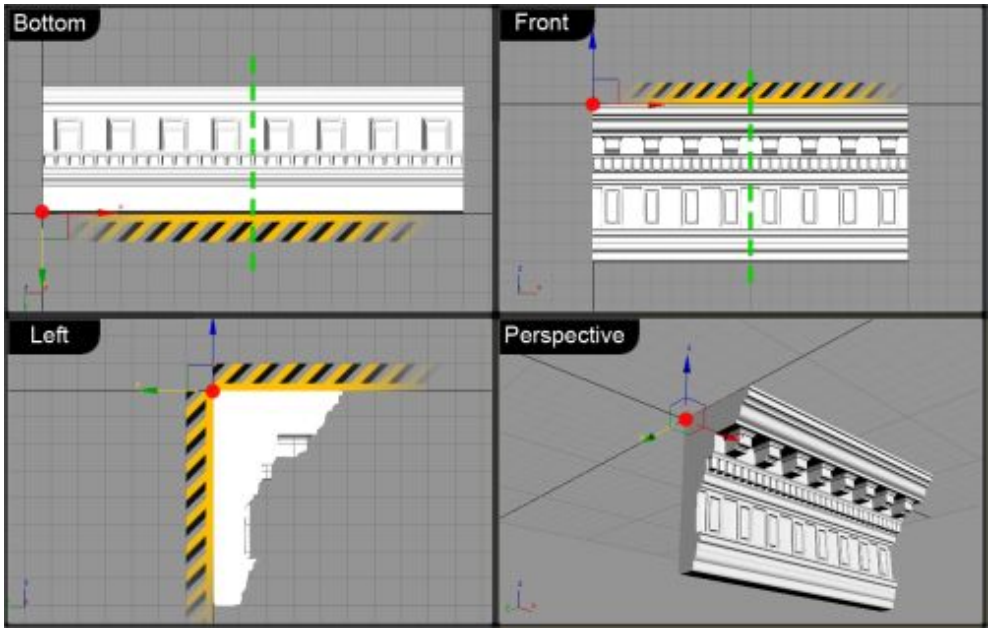


Abbildung 4.7: Beispiel für die Positionierung eines Pivot Points an einer Stuckleiste (Mader, 2005).

Für Elemente, die unter oder direkt neben anderen platziert werden, sollte der Pivot Point an die Ecke gesetzt werden, die mit den benachbarten Objekten in Kontakt kommen sollen. So kann das Element in genau dieser Ecke einrasten. Ein Beispiel hierfür könnte eine Stuckleiste sein, die zwischen Decke und Wand eingesetzt wird. (Meler, 2018)

Diese Regeln können teilweise auch untereinander kombiniert werden. Soll beispielsweise die Leiste aus dem vorigen Beispiel noch horizontal erweiterbar sein, muss der Pivot Point zusätzlich an einer der äußeren Ecken platziert werden (Abbildung 4.7). (Meler, 2018)

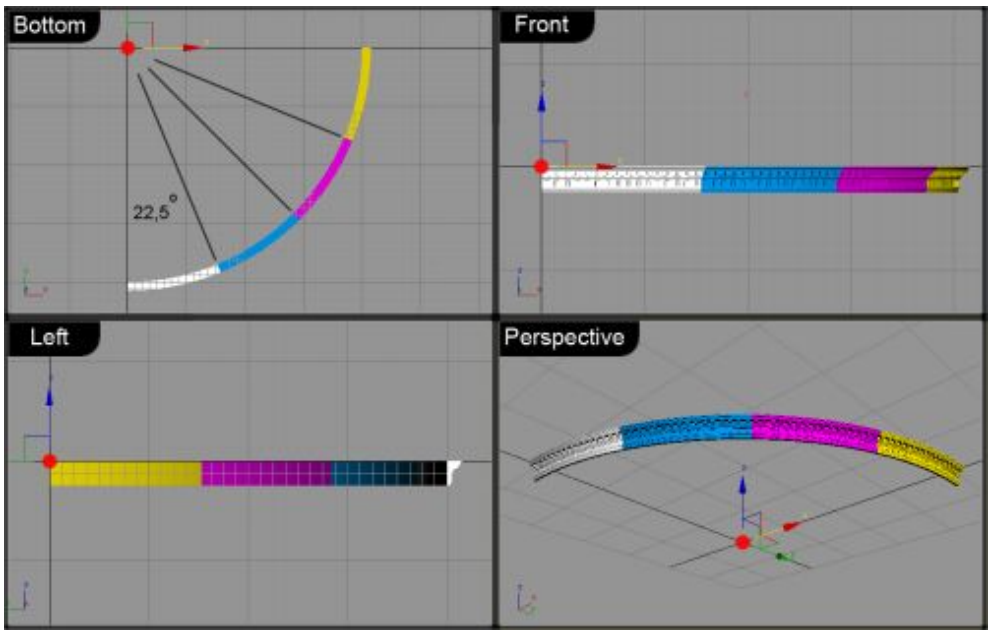


Abbildung 4.8: Beispiel für einen Pivot Point im Zentrum eines Kreises (Mader, 2005).

Eine etwas unintuitive Ausnahme bilden Objekte mit einer Krümmung in ihrer Oberfläche. Für diese empfiehlt es sich, den Pivot Point in das Zentrum des Kreises der Krümmung zu setzen (Abbildung 4.8). Dadurch ist er zwar weit weg vom eigentlichen Objekt, allerdings ergibt sich durch die Krümmung ein nützlicher Vorteil: Wenn die Pivot Points der einzelnen Elemente überlagern, können diese durch simple Anpassung der Rotation sehr leicht aneinander angelegt werden. (Mader, 2005)

4.2.5 Kit-spezifische Planung

In der zweiten Planungsphase werden die Details des Kits ausgearbeitet und spezifische Regeln definiert. Während dieser Phase wird das Setting festgelegt, welches das Kit abbilden soll (Burgess & Purkeypile, 2013). Das können beispielsweise vereiste Höhlen oder das Innere eines Raumschiffs sein. Außerdem sollte geklärt werden, wie und mit welchem Zweck das Kit im Spiel eingesetzt wird, damit sein Umfang und seine Möglichkeiten entsprechend planbar sind (Burgess & Purkeypile, 2013). Nach Perry ist es eine gute Methode dafür eine Liste mit allen Eigenschaften, die das Kit erfüllen soll anzufertigen. Dabei geht es primär um die Kernelemente des Kits, welche für den Großteil des Einsatzgebietes genutzt werden (Perry, 2002).

Einzigartige Modelle, so genannte *Hero Pieces*, sollten erst im späteren Verlauf entwickelt werden. Sie trügen in der frühen Phase wenig zum Fortschritt des Kits bei, benötigten aber viel Arbeit und ihr Fehlen verzögerte im Gegensatz zu den Kernelementen des Kits nur an wenigen Stellen den Arbeitsablauf. (Burgess & Purkeypile, 2013)

Joel Burgess fordert in einem GDC-Talk zu validieren, welche Kits eine Existenzberechtigung haben.Können Kits zusammengefasst werden, um ein generalisiertes Kit mit vielseitigerem Nutzen zu erzeugen, sei dies empfohlen. Dies verringere die Anzahl und die Planungszeit der benötigten Kits und so auch die Komplexität des Projekts. (Burgess & Purkeypile, 2016b)

Perry, Burgess und Purkeypile empfehlen einheitlich, dass Kits Elemente enthalten sollten, mit denen sie untereinander verbunden werden könnten (Burgess & Purkeypile, 2013; Perry, 2002). So könne z. B. eine einheitliche Größe für Türrahmen genutzt werden, um den Übergang von einem in das nächste Kit zu schaffen, ohne für beide spezifische Teile anfertigen zu müssen (Burgess & Purkeypile, 2013). Perry erwähnt zusätzlich, dass Kits abgrenzende Elemente benötigten. So solle für einen modularen Fluss beispielsweise ein Wasserfall oder eine Mündung in ein Gitter existieren, damit der Fluss beendet werden könne (Perry, 2002). So lange ein Kit nicht für die Generierung endloser Objekte/Level gedacht sei, sei die Einplanung solcher Teile zwingend.

Für das weitere Vorgehen gibt es verschiedene Methoden, bei denen teilweise die Planungsphase in die Erstellungsphase über geht.

Klafke und Norris empfehlen beide, mit der Erstellung von Texturen und Materialien zu beginnen, bevor die ersten Modelle existieren (Klafke, 2010; Norris, 2015). Obwohl sie hier einig sind, unterscheiden sich ihre Vorgehensweisen:

Norris erzeugt für seine Kits bereits voll ausgearbeitete Texturen mit allen Details, die er abbilden möchte, bevor er die Modelle erstellt (Norris, 2015). Klafke hingegen arbeitet etwas freier. Für ihn ist ein Mockup der späteren Textur ausreichend, um die Modelle passend zu unwrappen und dadurch im weiteren Verlauf Zeit zu sparen (Klafke, 2010).

Der Aufbau der genutzten Texturen ist bei beiden ähnlich und besteht aus großen, kachelbaren Texturen, um Wände abzubilden (Klafke, 2010; Norris, 2015). Klafke nutzt diese zusätzlich auch für Bodenflächen (Klafke, 2010). Für alle im Kit darzustellenden Details erstellen sie eine, beziehungsweise zwei Texturen mit einer Verbindung aus Trims, so genannte Trim Sheets[12] (Klafke, 2010; Norris, 2015).

Eine weitere Vorgehensweise wird von Burgess und Purkeypile beschrieben. Hier wird mit der Erstellung einfacher Modelle begonnen, anhand derer alle aufgestellten Regeln und erarbeiteten Ideen ausgiebig getestet werden. Es ist möglich, dass in dieser Phase viele Modelle entstehen, die verworfen werden, bevor eine Herangehensweise gefunden wird, um passende Assets zu erstellen. In dieser Zeit können die Regeln des Kits noch einfach verändert und auf die Bedürfnisse des Spiels angepasst werden. (Burgess & Purkeypile, 2013)

Die nächste Stufe dieses Prozesses ist das sogenannte *Grayboxing*. In dieser Phase werden die Kernelemente des Kits mit einem geringen Detailgrad und ohne Texturen erstellt. Das *Grayboxing* dient weiterhin dazu, das Konzept des Kits sowie die Funktionen der einzelnen Module zu überprüfen und solange zu verfeinern, bis alle Elemente auf dem gewünschten Niveau sind. Die visuelle Qualität der Elemente spielt dabei noch keine große Rolle, der Fokus liegt auf der Funktionalität. Können die Elemente des Kits alle Probleme lösen, für die es konzipiert wurde, kann die Phase beendet werden. (Burgess & Purkeypile, 2013)

[12] Trim Sheets sind Texturen, die mehrere unabhängige und meist kachelbare Details enthalten (Meler, 2018).

Nach der *Graybox*-Phase werden alle Assets ausgearbeitet. Aus den vorher entworfenen Elementen werden detailreiche, texturierte Assets erstellt und für die weitere Verarbeitung freigegeben. Zusätzlich werden weniger prominente Teile erstellt, die in der vorherigen Phase ausgelassen wurden. Existieren Varianten von Assets, sollte erst eine Version fertig gestellt werden, um final testen zu können, dass es keine Probleme gibt. So kann vermieden werden, dass alle Varianten überarbeitet werden müssen, wenn es zu Fehlern kommt. Sowohl während dieses Vorgangs als auch danach werden alle Assets verfeinert. Es werden bestehende Fehler und Unreinheiten ausgebessert und die Kits, falls nötig, um Module erweitert, bis das Projekt beendet ist. (Burgess & Purkeypile, 2013)

Eine dritte Vorgehensweise wird von Perry angeführt. Ihm zufolge sollten erst einfache modulare Modelle erstellt und dann nach und nach weitere komplexere Teile hinzugefügt werden. Unterstützend zu dieser Arbeit soll eine Asset-Review eingeführt werden, bei der regelmäßig getestet wird, ob alle bisher produzierten Modelle wie vorgesehen untereinander kompatibel sind und funktionieren. Dies soll gewährleisten, dass am Ende keine fertigen Assets erzeugt werden, die nicht ohne aufwändige Überarbeitungen funktionieren. (Perry, 2002)

Ein bisher unerwähnter Aspekt ist die Namensgebung der modularen Assets. Bereits früh im Projekt sollte eine passende Namenskonvention eingeführt werden, die auf alle Kits anwendbar ist. Dies sorgt dafür, dass alle Artists, Designer und Programmierer immer genau wissen, welchen Zweck ein bestimmtes Modul hat und damit schneller und effizienter arbeiten können. (Burgess & Purkeypile, 2013)

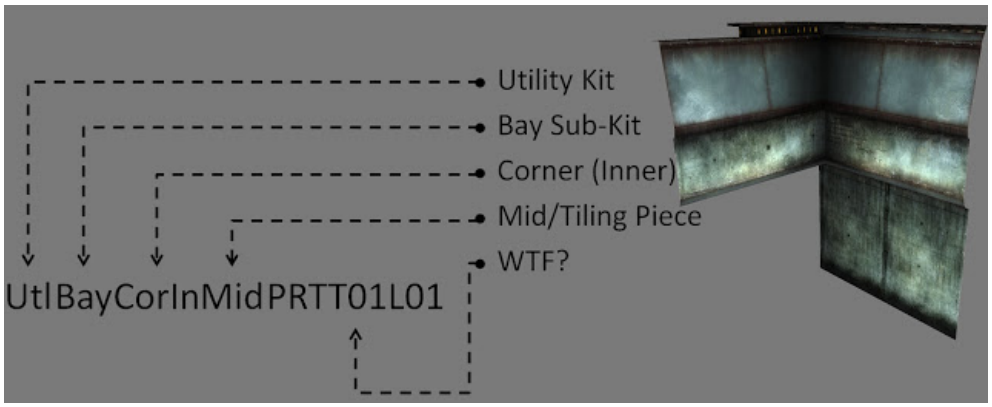


Abbildung 4.9: Beispiel für Namensgebung eines modularen Assets aus *Fallout 3* (Burgess & Purkeypile, 2013).

Abbildung 4.9 zeigt beispielhaft die Kodierung einer Namenskonvention. Der Name sollte möglichst deskriptiv und einfach zu entschlüsseln sein. Zudem ist es sinnvoll, eine bei „01“ beginnende und aufsteigende Nummerierung einzuführen. Dies erleichtert die spätere Implementierung von Asset-Varianten. Durch die Einhaltung der Namenskonvention teilen sich alle Elemente einer Art einen Namen. Dies reduziert die Komplexität des Systems und Elemente können durch einfaches Heraufzählen ausgetauscht werden. (Burgess & Purkeypile, 2013)

Alle wichtigen Aspekte der Planung und Grundlagen für Modularität sind jetzt erklärt. In den folgenden Abschnitten werden weitere, für die Generierung modularer Assets hilfreiche Aspekte erläutert.

4.2.6 Weitere Techniken

In diesem Abschnitt werden weitere Methoden erläutert, die keinem der obigen Themen zuzuordnen oder nur in Randfällen anwendbar sind.

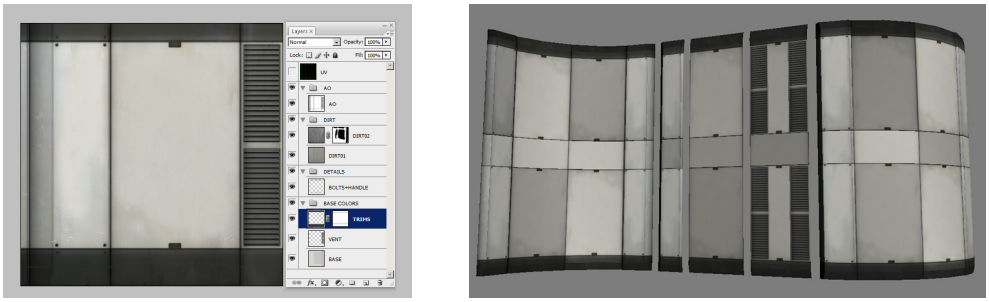
Bei der Nutzung modularer Assets sollte man auch außerhalb der vorgegebenen Regeln denken und Module von Zeit zu Zeit zweckentfremden. So könne beispielsweise eine Zimmerpflanze auch als Baum eingesetzt werden, wenn die Distanz zum Spieler ausreichend sei. (Lin et al., o.D.)

Module können direkt so gestaltet werden, dass sie mehrere aufgaben erfüllen können. Zum Beispiel könnte die Decke einer Höhle gleichzeitig auch als Boden genutzt werden (Perry, 2002). Dadurch könnten mit etwas mehr Planung Ressourcen und Arbeitszeit eingespart werden (Lin et al., o.D.).

Um das rasterartige Erscheinungsbild modularer Assets zu mildern, kann an manchen Stellen das Raster verlassen werden. Dies ist allerdings nur möglich, wenn der Editor es auch technisch erlaubt. Für das Platzieren von Modulen im Editor kann definiert werden, dass Objekte an einem alternativen Raster, welches durch die Rotation und Position eines gewählten Elementes definiert wird, ausgerichtet werden. (Burgess & Purkeypile, 2013)

Kits sollten außerdem Elemente enthalten, die eventuelle Lücken oder unschöne Übergänge[13] verbergen. Diese helfen auch repetitive Muster des Kits zu brechen. Beispiele dafür sind Säulen, Pfeiler, Steinformationen oder auch Pflanzen. (Burgess & Purkeypile, 2013; Perry, 2002)

Falls geplant ist das normale Raster zu verlassen, müssen zusätzlich Elemente entworfen werden, die Lücken und Kanten verbergen. Werden beispielsweise in verschiedenen Winkeln verbundene Tunnel-Elemente genutzt, müssen Übergänge für alle Möglichkeiten erstellt werden, wie diese verbunden werden können. (Burgess & Purkeypile, 2013)



(a) Textur ohne den Einfluss von Vertex Farben.

(b) Assets unter dem Einfluss Vertex Farben.

Abbildung 4.10: Beispiele für Einfluss von Vertex Farben (Klevestav, 2009).

In 3D-Tools kann Vertices eines Meshes ein Farbwert zugeordnet werden. Diese Farbwerte heißen Vertex-Farben. Durch diese Technik kann, ohne großen Aufwand und ohne neue Objekte eine größere Anzahl Modelle erzeugt werden. In den

[13] Diese können zum Beispiel durch Texturen entstehen, die nicht zueinander passen oder durch Module die nicht füreinander konzipiert wurden.

Material-Einstellungen des Objekts muss angegeben werden, dass für die Farbgebung des Models die Vertex-Farben einkalkuliert werden sollen. Die Abbildungen 4.10a 4.10b zeigen, wie sich diese auf das Aussehen einer Textur auswirken können. (Klafke, 2010; Klevestav, 2009)

4.2.7 Texturierung von modularen Assets

In dieser Thesis liegt der Fokus auf der Arbeit an den Modellen. Texturierung wird demzufolge nur angeschnitten.

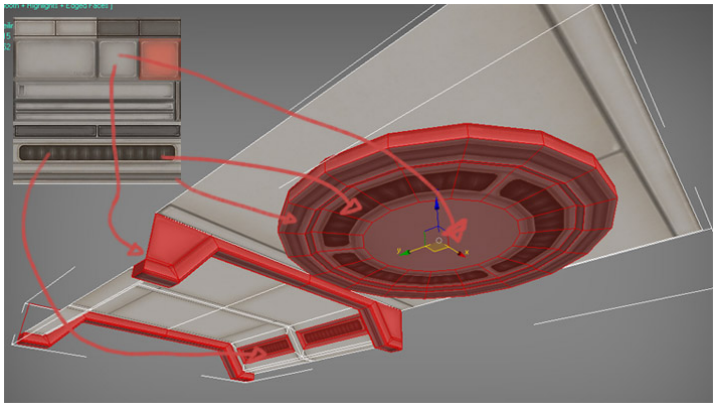


Abbildung 4.11: Beispiel für Mehrfachnutzung eines Trimsheets (Klafke, 2010).

Modulare Elemente können wie normale Assets texturiert werden. Texturen von gemeinsam genutzten Objekten sollten aus Performance-Gründen auf einem Textur-Atlas verbunden werden. Nutzt ein Projekt Elemente, die mehrfach aneinandergefügt werden sollen, bietet es sich an, kachelbare Texturen zu nutzen. Auf diese Weise können große Flächen erzeugt werden, ohne Kanten oder sichtbare Nähte. Dies ist vor allem für modulare Assets von großem Vorteil. Da sie vielseitig nutzbar sind und nur einmal geladen werden müssen, sparen solche Texturen außerdem Ressourcen. (Meler, 2018)

Abbildung 4.12: Ein Beispiel für das Potential austauschbarer Texturen. In Anlehnung an (Burgess & Purkeypile, 2016a).

Werden austauschbare Texturen genutzt, die dem gleichen Aufbau folgen, kann ein Modell unterschiedlich aussehen. (Norris, 2015)

Hiermit sind die Grundlagen für die praktische Ausarbeitung dieser Thesis abgeschlossen. Die nachfolgenden Kapitel wenden diese in der eigenen Umsetzung modularer Assets an.

5 Beispielanwendung

5.1 Das Spiel

Dieser Abschnitt soll einen grundlegenden Überblick über das Spiel gewähren, für welches die modularen Gebäude erstellt werden. Die folgenden Abschnitte behandeln die Anwendung der zuvor erarbeiteten Methoden.

Grundlagen

Bei *Renegade Line* handelt es sich um einen Third Person Cartoon Shooter, in dem bis zu 16 Spieler über das Internet in verschiedenen Spielmodi auf unterschiedlichen Karten gegeneinander spielen können. In den meisten Spielmodi werden die Spieler in zwei Teams mit jeweils acht Spielern eingeteilt, die im Anschluss um den Sieg kämpfen. Spieler können während des Spiels eine von vier verschiedenen Klassen wählen, die spezielle Fähigkeiten und passende Waffen besitzen.

Das Spiel befindet sich zum Zeitpunkt dieser Thesis im Early Access und wird von Raw Vengeance Studios entwickelt. Raw Vengeance ist ein kleines Start-up aus Hamburg. Aktuell wird eine neue Karte für einen neuen Spielmodus entwickelt. Zusammen mit der neuen Karte soll auch ein neues Setting etabliert werden. Bisher wurden alle Modelle für die neue Karte von mir erstellt.

Der Stil

Der Stil des Spieles ist ein stilisierter Cartoon Look. Alle Elemente sind Abwandlungen ihrer realen Gegenstücke. Meist werden prägnante Formen stark hervorgehoben. Das bisherige Setting aller Assets ist leicht stereotypisierend an eine italienische Kleinstadt angelehnt (vgl. Abbildung 5.1). Das neue Setting soll das feudale Japan repräsentieren. Auch hier wird mit Stereotypen gearbeitet, die eindeutig und leicht zu erkennen sein sollen. In *Renegade Line* können Gebäude nicht betreten werden, alle Handlungen finden im Freien stattfinden. Dies reduziert die Komplexität der Modelle und die Arbeitszeit ihrer Erstellung.

Abbildung 5.1: Ausschnitt eines aktuellen Levels in dem bisherigen Setting.

5.2 Planung

In der Planungsphase wurde in Zusammenarbeit mit dem Lead-Level-Designer definiert, welche Anforderungen das Kit erfüllen soll. Es wurde über den generellen Aufbau der Gebäude gesprochen. Dies umfasst die Größe der Grundrisse, die Anzahl der Stockwerke, welche Elemente eine Etage ausmachen sollen und welche Sonder-Elemente es geben soll. Außerdem wurde das Design der verschiedenen Elemente angesprochen. Die Umsetzung der Anforderungen habe ich selbst ausgearbeitet.

Diese Anforderungen wurden festgelegt:

Grundriss

- Gebäude sollen nicht nur rechteckig, sondern auch in anderen Formen aufgebaut sein (L-Form, Kreuz etc..) und
- die Grundfläche eines Gebäudes soll nach Möglichkeit unlimitiert sein.

Etagen

- Es sollen mindestens fünf oder besser unbegrenzt viele Etagen möglich sein und
- in der untersten Etage soll ein Steinfundament mit einer Treppe eingesetzt werden können.

Details

- Es soll Elemente für ein Vordach geben, das eine Art Umrandung um das Gebäude bildet (Abbildung 5.3 auf Seite 40),
- Wandstücke sollen in verschiedenen Ausführungen erstellt werden: mit unterschiedlichen Holzmustern, Fenstern und Türen und
- es soll Elemente für einen Balkon geben, wenn möglich.

Design

- Die Farbgebung soll sich an Abbildung 5.2 (Folgeseite) und dem bisherigen Farbspektrum orientieren und
- das grundlegende Design der Gebäude soll sich an Abbbildung 5.2 (Folgeseite) und Abbildung 5.3 (Seite 40) orientieren.

Abbildung 5.2: Konzept für Farbgebung und Aufbau der Gebäude (Pampin, 2018).

Nach der konzeptionellen Planung wurde geprüft, welche Möglichkeiten die genutzten Programme bieten, um das Projekt umzusetzen. Zudem wurden erste Einstellungen vorgenommen, um einen fehlerfreien Ablauf nachfolgender Prozesse zu gewährleisten. Die Maßeinheiten von Blender und UE wurden aufeinander abgestimmt und die *Snapping*-Funktion in UE getestet.[14]

In UE wurde bisher mit Zentimetern gearbeitet, dies wird so beibehalten. Für das Arbeiten mit einem modularen System wird das Raster von klassisch (mit Schritten von 1 m, 2 m, 5 m usw.) auf die Basis von Zwei und dessen Potenzen eingestellt. Abstände, die so entstehen, lassen sich, wie in Abschnitt 4.2.3 erwähnt, besser halbieren ohne das ungerade Zahlen entstehen, die das genaue Platzieren von Objekten deutlich erschweren würden. Durch eine einfache Einstellung im Viewport[15] kann die Raster Größe (2 cm, 4 cm, 8 cm, ... 128 cm, 256 cm, usw.) angepasst werden.

Auch in Blender kann die Maßeinheit auf Zentimeter eingestellt und das selbe Raster wie in UE konfiguriert werden. Das Raster wird hier allerdings nur wenig benutzt: Beim Modellieren ist es eher hinderlich, wenn Bewegungen nur in definierten Schritten durchgeführt werden können, da viele Details der Modelle dem Raster nicht folgen werden.

Die Planung der Elemente basiert auf den Eigenschaften bestehender Assets. Durch dieses Vorgehen soll sichergestellt werden, dass sich die neuen Modelle in den bestehenden Stil einfügen.

Die bisherigen Gebäude wurden für ein anderes Setting entwickelt, die genutzten Maße gelten dennoch als Orientierung für die neuen Assets.

[14] Für dieses Projekt wurde mit Blender 2.79b und Unreal Engine 4.21.2 gearbeitet

[15] Der Viewport ist eine grafische Oberfläche, in dem das aktuelle Level betrachtet und bearbeitet werden kann

Größen bestehender Elemente

- Türen sind 2,8 m hoch und 1,8 m breit,
- Fenster sind zwischen 1,5 m und 2,4 m hoch und breit,
- eine Etage der alten Gebäude ist ca. 4,3 m hoch,
- Treppenstufen an Gebäuden haben eine Höhe von ca. 24 cm und eine Tiefe von ca. 27 cm,
- Treppenstufen in der Welt haben eine Höhe von ca. 42 cm und eine Tiefe von ca. 56 cm und
- die Spielfiguren sind 195 cm hoch und ca. 70 cm breit.

Als Fußabdruck (siehe Abschnitt 4.2.3) für die neuen Elemente wird eine Größe von 5,12 m x 5,12 m x 5,12 m genutzt. Kein erstelltes Element soll den so definierten Würfel überschreiten.

Auf Grundlage der bestehenden Gebäude wurde sich für eine Etagenhöhe von 5,12 m entschieden. Grund hierfür war die ursprüngliche Etagenhöhe von 4,3 m, zu der für die neuen Gebäude noch die Höhe des Vordaches addiert wird. Nach längerer Planung habe ich mich für Elemente entschieden, die so aufgebaut sein sollen, wie in Abbildung 5.3 eingezeichnet. In der Modelierungsphase wurde erkannt, dass dieser Ansatz eher ungeeignet ist um das volle Potenzial von Modularität auszunutzen. Im nächsten Abschnitt wird auf den alternativen Lösungsansatz eingegangen.

Damit das Kit mehr Abwechslung erzeugen kann, wurden zudem Elemente in halber Höhe (2,56 m) und halber Breite (2,56 m) geplant. Ein schon in der Planungsphase erkennbares Problem, war das Zusammenlaufen der Etagen. Der Prozess zur Lösung dieses Problems wurde in die Modellierungsphase verschoben. Es erschien leichter, eine Lösung im dreidimensionalen Raum zu finden, statt mit Skizzen zu arbeiten.

Abbildung 5.3: Skizze der zunächst definierten Module. In Anlehnung an (Muza-chan, 2012).

Für das Projekt wurde eine mittlere Stufe von Modularität gewählt. Weitere Varianz wird durch verschiedene Versionen der gleichen Elemente erzeugt. Zusätzlich werden kleinere Module erstellt, um Muster aufzubrechen und den Gebäuden mehr Details zu verleihen. Bisher wurden neun verschiedene Gebäude für alle Level genutzt, aufgrund der Perspektive und der Art des Gameplays ist dies bisher nur gering negativ aufgefallen. Folglich sollte es unproblematisch sein, wenn die neuen Gebäude, aus der genutzten Spielperspektive, aus größeren Elementen bestehen, auch wenn es weniger Abwechslung bedeutet.

Das Kit wurde für eine bessere Übersicht und leichtere Handhabung in sieben Subkits aufgeteilt. Die Sets entsprechen jeweils einer Ebene des Gebäudes.

Subkit	**Bedeutung**
Fundament	Elemente für das optionale Steinfundament
Wall	Wandelemente
Balcony	Balkonelemente
Roof	Elemente für das optionale Dach
TopRoof	Dachelemente
Joist	Dachbalkenelemente, die auf das Dach gesetzt werden können
Beam	Holzbalkenelemente, die an Wandübergängen platziert werden können

Tabelle 5.1: Übersicht aller Subkits und deren Bedeutung

Die Benennung der Elemente, ist wie in Abschnitt 4.2.5 erwähnt, ein wichtiger Faktor für effizientes Arbeiten mit modularen Kits. Für die Bezeichnung wurden verschiedene Begriffskategorien definiert, mit deren Hilfe die Elemente genau zugeordnet werden können. Der Name eines Moduls setzt sich aus den folgenden Kategorien zusammen:

Subkit	Höhe	Form	Breite	Variante
Ro(of)	Ha(lf)	In(ner Corner)	Si(ngle)	04

Tabelle 5.2: Bezeichnungsschemata der genutzten Module mit einem Beispiel

Zuerst wurde versucht die Bezeichnungen der Elemente nur aus den Anfangsbuchstaben der Wörter zu bilden, die ein Modul ausmachen (z. B. „FSBS01“). Es stellte sich heraus, dass eine so starke Vereinfachung schwer zu lesen beziehungsweise zu erlernen ist. Aus diesem Grund wurden für die Benennung der Elemente die ersten beiden Buchstaben genutzt, um die Namen der Elemente zu generieren. Aus „FSBS01“ wurde „FuSiBaSi01“, welches einfacher in „Fundament Single Basic Single 01“ übersetzt werden kann.

Die Bezeichnung der Höhe wurde so weit vorne im Namen platziert, damit bei einer Suche im Editor zusammengehörige Elemente leicht gefunden werden können. Die verschieden hohen Elemente können auf einer Ebene nicht miteinander verbunden werden. Durch dieses Schema können die nicht kompatiblen Elemente leicht voneinander getrennt werden.

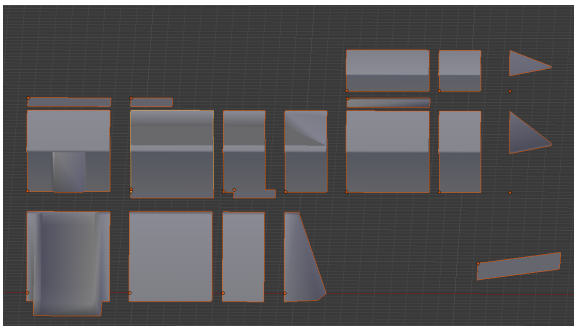
Der Abschnitt zur Planung ist hier beendet, die Planung insgesamt jedoch noch nicht und wird während der Modellierung fortgeführt. Die geplanten Konzepte müssen mit einfachen Modellen getestet und eventuell angepasst werden. Nur so können im späteren Verlauf Fehler vermieden werden.

5.3 Modellierung

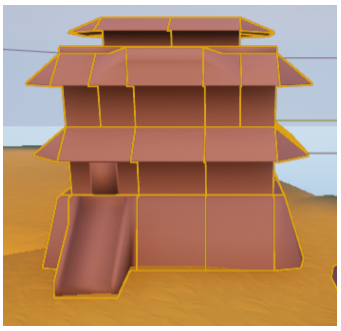
Zu Beginn der Modelierungsphase wurden Prototypen der konzipierten Modelle erstellt und auf ihre Funktion hin überprüft.

Das zuvor entwickelte System funktionierte. Durch das Einhalten der definierten Größen und das Platzieren der Pivot Points an den unteren Eckpunkten der Module ließen sich die Objekte leicht miteinander verbinden ohne das Lücken entstanden.

Um zu überprüfen, ob das Kit auch in der UE funktioniert, wurden die Module testweise importiert und zu einem Gebäude verbunden (Abbildung 5.4b). Auch in der Engine traten keine Probleme mit dem Kit auf. Das Raster in UE wurde auf 1.28 m eingestellt, was das Zusammensetzen des Gebäudes enorm beschleunigte.



(a) Alle Prototypen-Module in Blender.



(b) Aus Modulen erstelltes Gebäude in UE.

Abbildung 5.4: Erste Prototypen der geplanten Module.

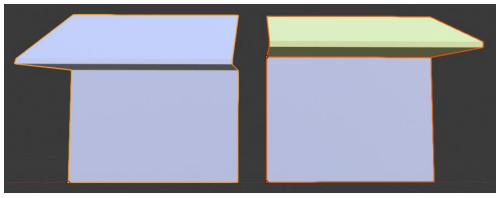


Abbildung 5.5: Ursprüngliches Wandelement (links) und überarbeitete Version mit abgelöstem Dach (rechts).

Im Anschluss wurde begonnen, die Module weiter auszubauen. Dabei wurde klar, dass die geplante Struktur der Module die Modularität des Kits einschränken und den Arbeitsaufwand vergrößern würde. Es war geplant, dass jedes Wandmodul mit einem Dach oder einem Dach und einem Balkon erstellt werden sollte. Ein besserer Ansatz ist es, die Dächer und Balkone von den Wänden zu trennen, was zur Folge hat, dass die Wände nicht mehr eine Höhe von 5,12 Metern haben. Um das Raster nicht zu verlassen, wurde die normale Größe von Wänden auf 3,84 Metern gesetzt. Zusammen mit Dachelementen, die 1,28 Meter hoch werden, kann die Etagenhöhe von 5,12 Metern wieder eingehalten werden. Die zuvor geplanten halbhohen Wandelemente werden mit einer Höhe von 2,56 Metern erstellt.

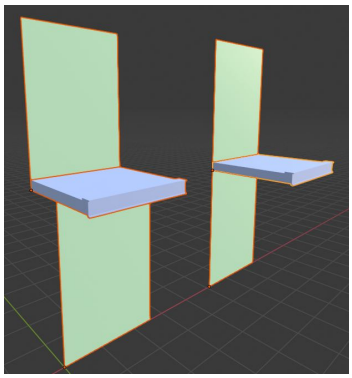


Abbildung 5.6: Mögliche Kombinationen von Balkon- und Wandelementen.

Durch diese Änderung konnte die Anzahl der benötigten Module verringert und die Modularität des Kits erhöht werden. Bei Bedarf können Wandelemente auch ohne Dachelemente aufeinander gesetzt werden, um größere Wände zu erzeugen. Auch Balkonmodule können durch diese Entwicklung vielseitiger genutzt werden. Sie können entweder an die Wand gesetzt werden oder auch 1,28 Meter nach innen in das Gebäude verschoben werden, um die nachfolgende Etage zu verkleinern, ohne dass hierfür verschiedene Module benötigt werden.

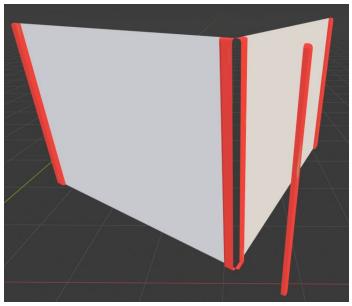


Abbildung 5.7: In Wandelemente eingesetzte Holzbalken mit Sonderelement.

Nach der Überarbeitung des Systems wurden die neuen Wandelemente mit einem halben vertikalen Holzbalken am linken und rechten Rand erweitert. Werden zwei Elemente miteinander verbunden, bildet sich am Übergang ein vollständiger Holzbalken. Die Holzbalken wurden eingesetzt, um das Kit dem Setting anzupassen. Wurde mit Wandmodulen eine Ecke gebildet, entstand eine Lücke zwischen den Holzbalken, die mit einem Sonderelement gefüllt werden musste.

Bei einem weiteren Test in UE wurden Probleme mit Schatten gefunden. An Stellen, bei denen Module aufeinander treffen und keinen glatten Übergang bilden, kommt es zu sehr starken Helligkeitsunterschieden, die falsch aussehen. Dies passierte bei dem Element zum Füllen der Lücke zwischen Holzbalken und bei der bis dahin genutzten Form von Eckelementen der Dachmodule. Dieses Problem kann auch bei glatten Übergängen auftreten, ist dort aber weniger auffällig.

Die Form der Dachmodule wurde daraufhin so abgeändert, dass die Übergänge zu geraden Modulen eine glatte Fläche bilden.

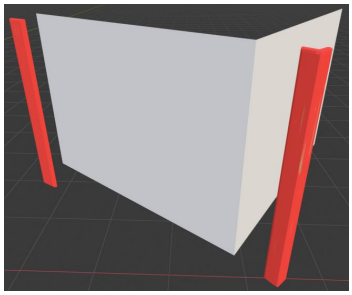


Abbildung 5.8: Überarbeitete Wandelemente und Holzbalken aus neuem Subkit.

Die zuvor in die Wände eingesetzten Holzbalken wurden wieder von den Wänden gelöst und wurden in ein eigenes Subkit verschoben. In diesem Subkit gibt es Holzbalken für die verschiedenen hohen Module und auch für Außen- und Innenecken. Durch diese Änderungen wurden die Probleme mit den Schatten gelöst. Zusätzlich können so verschiedene Versionen von Holzbalken genutzt werden, wodurch eine sichtbare Wiederholung vermindert wird. Auch ist es möglich Wände ohne Holzbalken zu bilden, was weitere Möglichkeiten eröffnet.

Nach diesen Anpassungen wurden weitere Tests durchgeführt. Bei diesen wurde festgestellt, dass bei einer bestimmten Kombination von Dach- und Balkonelementen im Dach Lücken entstehen können, die nicht mit bestehenden Modulen gefüllt werden können. Die Lücken hatten eine Breite von 1,28 Metern, dies entspricht einem Viertel der normalen Breite aller Elemente.

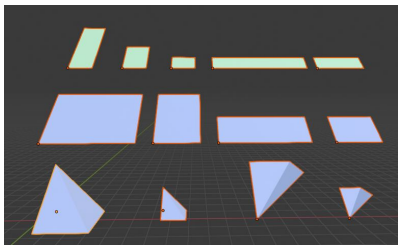


Abbildung 5.9: Ursprüngliches Elemente für den Abschluss des Daches (blau) und neue Elemente für das schließen spezifischer Lücken (grün).

Nach mehreren erfolglosen Versuchen diese Lücken, ohne viele neue Elemente zu vermeiden, wurde dem Subkit, für die Schließung des Gebäudes, weitere Elemente hinzugefügt. Auf Basis der bestehenden Elemente wurden neue Elemente in verschiedenen Kombinationen aus einem Viertel der Höhe und einem Viertel der Breite erstellt.

Mit diesen neuen Modulen war das Kit vollständig (Abbildung 5.10) und alle Modelle konnten weiter ausgearbeitet werden.

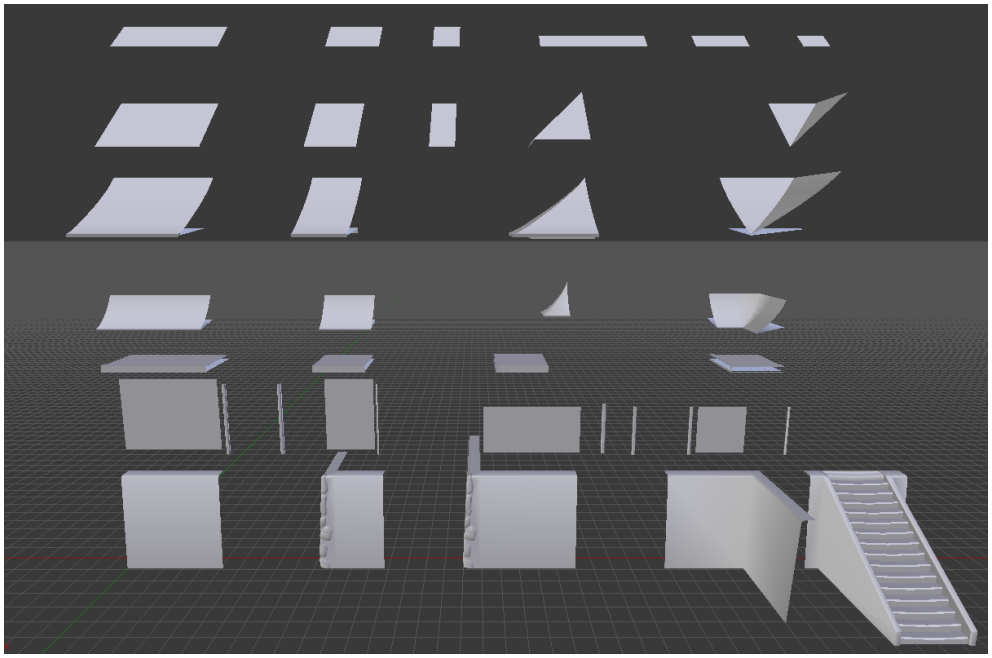


Abbildung 5.10: Prototypen des weiterentwickelten Systems.

Die finalen Module wurden von unten nach oben entwickelt. Zuerst wurde das Fundament fertiggestellt, dann die Wände, die Vordächer, die Balkone und zum Schluss die Dachelemente, die das Gebäude abschließen. Auf diese Weise konnte immer überprüft werden, ob die neuen Elemente ohne Probleme auf die unterliegende Schicht passen.

Aufgrund der speziellen Architektur der Gebäude konnte keine vollständige Modularität erreicht werden. Module sind horizontal immer nur mit Modulen des gleichen Subkits kombinierbar. Vertikal können alle Subkits miteinander verbunden werden. Ausschließlich das Fundament-Subkit kann nur als unterste Ebene fungieren, da es auf Grund seiner speziellen Form nicht auf andere Elemente gesetzt werden kann.

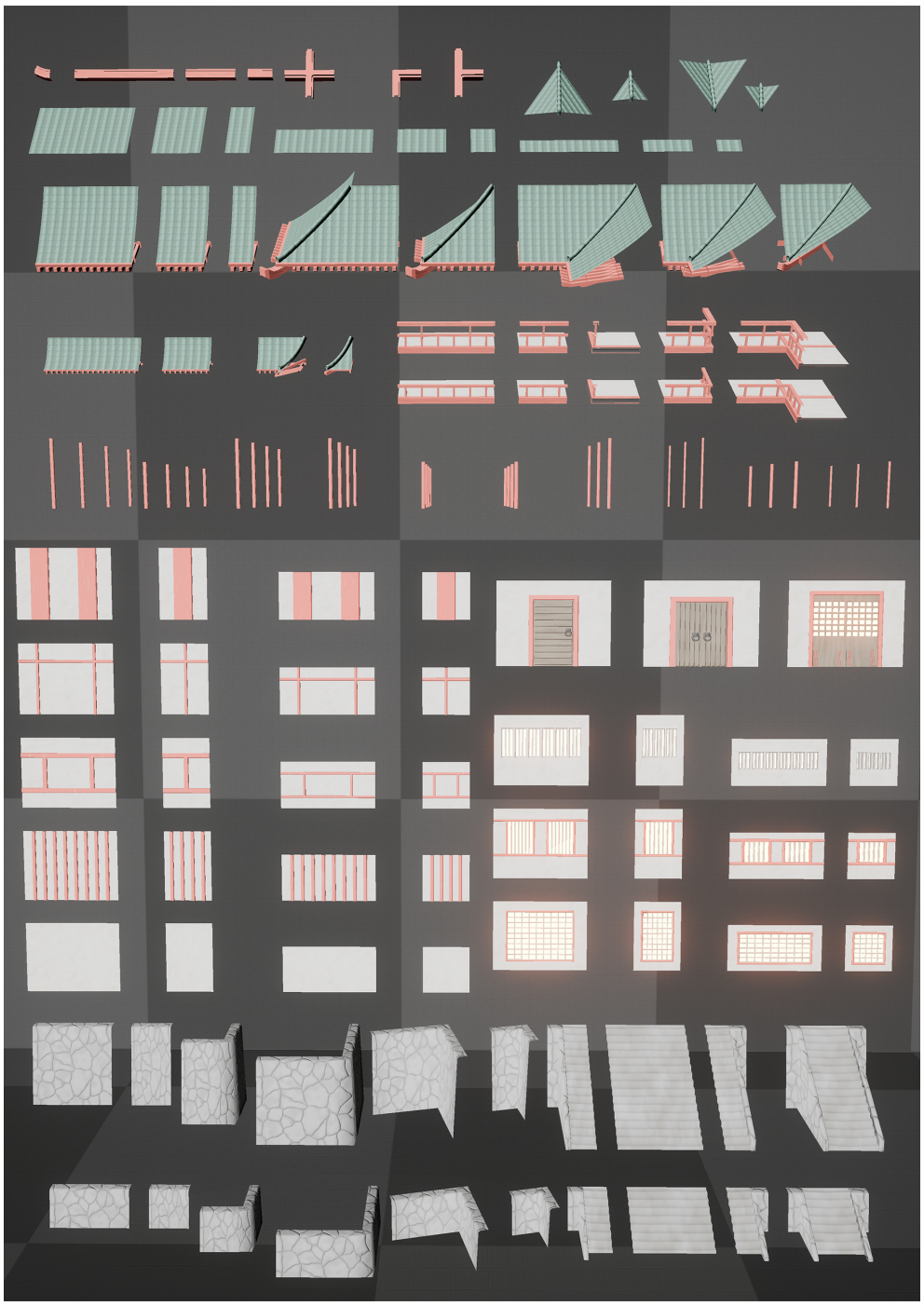


Abbildung 5.11: Alle Module mit angewandten Texturen.

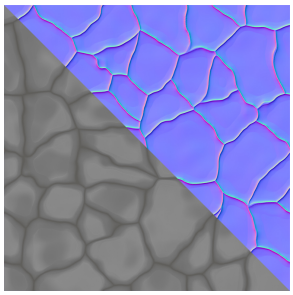
Für das Projekt wurden insgesamt 76 verschiedene Elemente erzeugt. Von den Wand-, Balkon- und Holzbalkenelementen wurden zusätzliche Varianten erzeugt, um eine abwechslungsreichere Gestaltung zu ermöglichen. Zusammen mit diesen entstanden insgesamt 126 Module. Abbildung 5.11, zeigt alle erstellten Module in ihrem finalen Zustand.

5.4 Texturierung

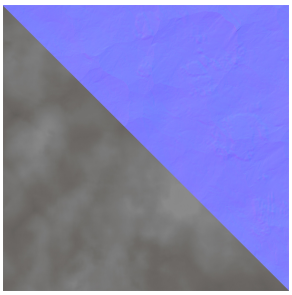
Wie zuvor erwähnt, steht in dieser Arbeit die Erstellung der Modelle im Vordergrund; infolgedessen wird in diesem Abschnitt nur skizziert, wie für die Texturierung vorgegangen wurde.

Nach der Planungsphase wurde für alle benötigten Texturen eine Vorabversion erstellt. So konnten für die Modelle schon passende UV-Maps erstellt werden. Hier war zu beachten, dass die Ränder der Modelle mit den Rändern der Textur verbunden wurden. Nur so kann ein nahtloser Übergang zwischen Modulen gewährleistet werden. Durch das Texturieren der Modelle konnte schon früh ein Eindruck vermittelt werden, wie die Module später aussehen werden. Die finale Version der Texturen wurde nach Abschluss der Modellierphase erstellt.

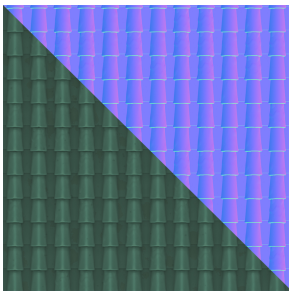
Um zu erfahren, welche verschiedenen Texturen benötigt werden, wurden Bilder der nachzubildenden Gebäude auf genutzte Baumaterialien hin analysiert. Zusammen mit dem vorliegenden Konzept (Abblidung 5.2) konnte so eine Liste mit benötigten Texturen ausgearbeitet werden.



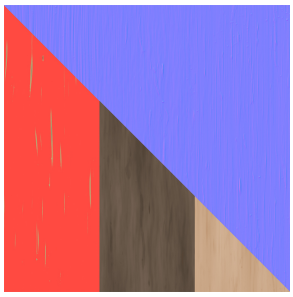
(a) Steintextur für Fundamentelemente.



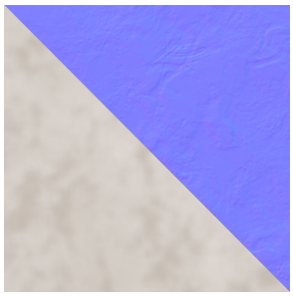
(b) Steintextur für Treppenstufen.



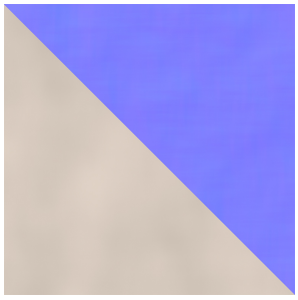
(c) Dachziegeltextur für Dachelemente.



(d) Holztexturen für Holzelemente.



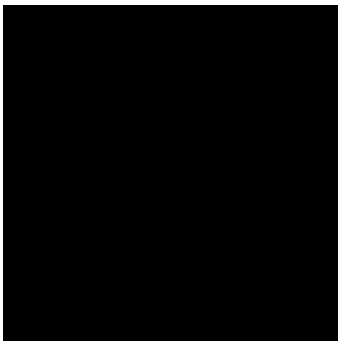
(e) Steintextur für Wandelemente.



(f) Papiertextur für Fenster.

Abbildung 5.12: Ausschnitte aller benötigten Diffuse- und Normalmaps.

Im Anschluss wurden alle Texturen mit *Substance Designer* erstellt. Werden Texturen mit den Standardeinstellungen von *Substance Designer* erstellt, sind diese immer so ausgelegt, dass sie in alle Richtungen kachelbar sind. Diesbezüglich mussten also kein spezielles Vorgehen beachtet werden.



(a) Specularmap.

(b) Rougnessmap.

Abbildung 5.13: Die für die meisten Modelle genutzten Rougness- und Specularmaps.

In *Renegade Line* werden für die Materialien Albedo-, Normal-, Roughness- und Specularmaps genutzt. Die Roughness- und Specularmap werden von den bestehenden Modellen übernommen. Diese sind größtenteils einfarbig und sorgen für ein eher stumpfes Erscheinungsbild der Modelle. Dies ist für den angestrebten Cartoon Look sehr wichtig. Die Albedo- und Normalmap wurde für jede Textur von Grund auf neu erzeugt. Für die Griffe an den Türen wurde ein schon existierendes Metall Material genutzt.

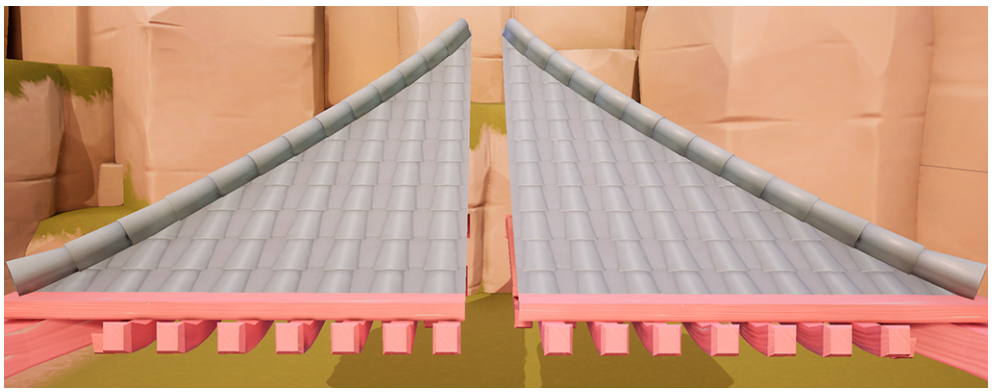


Abbildung 5.14: Einfluss der Specular- und Roughnessmap (links) im Vergleich mit den Standartwerten aus UE (rechts).

Von den Holztexturen für Holzbalken und Leisten an Türen, Fenstern und Dächern wurden drei Versionen erstellt: Eine, zum Konzept passende Rote sowie eine mit hellem und eine mit dunklem Holz (Abbildung 5.12d). So kann ein Gebäude durch Tausch einer Textur einen ganz neuen Look erhalten.

Versuche einen Texturatlas zu nutzen wurden nach Tests in UE nicht weiter verfolgt. Durch Mip Mapping, ein Vorgang bei dem Texturen komprimiert werden, um eine bessere Spielperformance zu gewährleisten, wurden nebeneinander liegende Texturen vermischt. Diese Vermischung führte zu sichtbaren Kanten auf den Modellen. Zusätzlich wäre es komplizierter gewesen die verschiedenen Holzarten zu nutzen, wenn diese nicht in verschiedene Materialien aufgeteilt worden wären.

5.5 Implementierung

Nachdem alle Tests bezüglich der Funktion und des Aussehens des Kits abgeschlossen waren, wurden alle Testdateien aus dem UE-Projekt entfernt.

Im Anschluss wurden für Texturen, Materialien, Modelle und fertige Gebäude Ordner angelegt. Der Ordner für die Modelle wurde mit Unterordnern für die Subkits gefüllt. Die fertigen Modelle sollten in Blueprints[16] abgespeichert und in dem dafür vorgesehenen Ordner gesichert werden.

Zuerst wurden alle Texturen importiert und auf deren Grundlage neue Materialien erstellt. Dann wurden die Modelle importiert und jedem Modell wurde ein Standardset an Materialien zugeordnet. Alle Modelle mit Holzelementen wurden mit dem roten Holz ausgestattet, dessen Verwendung als Preset vorgesehen ist.

Nach der Implementierung wurden für Tests erste Gebäude in UE erstellt. Die schon zuvor festgestellten Belichtungsprobleme an glatten Übergängen konnten durch Anpassungen der Lichtberechnung in den Welteinstellungen und dem Vergrößern der Lightmap gelöst werden. Die Lightmap ist ein Bild in der die Schattenwerte gespeichert werden, die durch die Berechnung des Lichtes ermittelt wurden. Die Modelle nutzen nun eine Lightmap-Größe von 128 x 128 Pixeln. Die vorkonfigurierten 64 x 64 Pixel waren zu klein und führten an manchen Stellen zu Überlappungen zwischen den Faces.

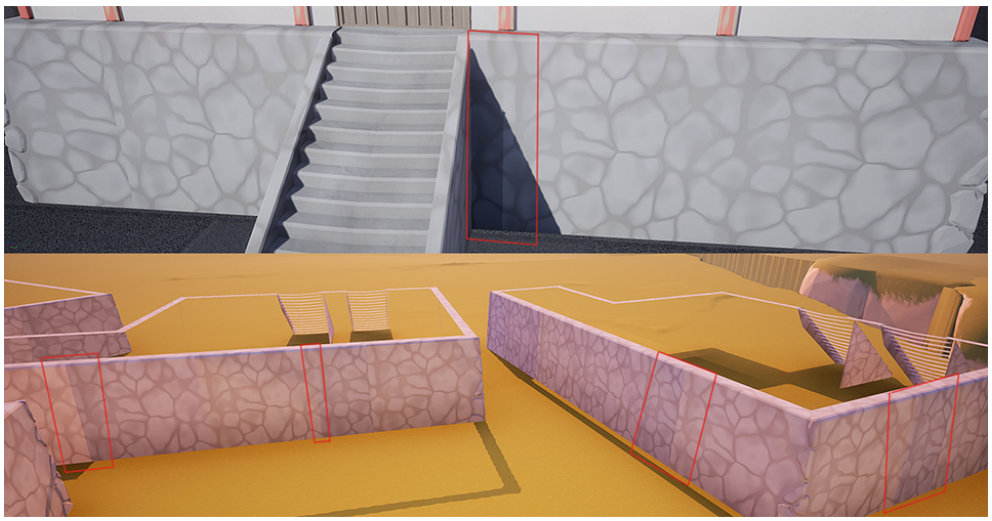


Abbildung 5.15: Harte Kanten an Modulübergängen von Modulen.

Das Ändern der Lichtberechnungseinstellungen verlangsamte die Lichtberechnung. Da dieser Prozess vor dem Spielen ausgeführt wird, hat dies keinen negativen Einfluss auf die Spielperformance, erhöht aber die Dauer von Tests in der Entwicklung. Nachdem die Probleme der Beleuchtung gelöst waren, wurden weitere Gebäude erstellt.

[16] In Unreal Engine sind Blueprints eine Art leerer Container der Code ausführen und Objekte enthalten kann.

Während dessen wurden keine Probleme festgestellt. Alle Module konnten wie geplant ohne Fehler verbunden werden. So wurden zehn verschiedene Gebäude erstellt, um mögliche Formen und Größen der Gebäude zu testen. Um zu überprüfen, ob sie beziehungsweise das modulare Kit in die alte Spielwelt passen, wurden die Gebäude in ein bestehendes Level eingefügt.

Abbildung 5.16: Aus dem Kit erstellte Gebäude in einem alten Level.

Die neuen Gebäude fügen sich sehr gut in den bestehenden Spielstil ein, auch wenn die übrigen Assets ein anderes Setting repräsentieren. Das an dem Spiel beteiligte Team war sich über die Diskussion der Ästetik hinaus einig, dass die neuen Gebäude zu dem bestehenden Stil passen.

Der praktische Teil der Arbeit ist hier beendet, im Folgenden werden die Ergebnisse dieses Projektes zusammengefasst und ausgewertet.

6 Evaluation

6.1 Kit

In diesem Abschnitt wird das unter Anwendung der erarbeiteten Methoden entstandene modulare Kit bewertet.

Performance

Für eine bessere Bewertung des Kits wurde ein Performance-Test durchgeführt, der ernüchternde Ergebnisse lieferte.

Die Tests wurden auf einem sehr leistungsstarken Desktop-PC[17] durchgeführt.

Es wurden zwei verschiedene Gebäude in zwei Versionen für die Tests genutzt: Jeweils eine, aus Modulen in UE zusammengesetzte Version und eine, die in Blender erstellt und als ein einzelnes Objekt in UE importiert wurde. Ein Gebäude ist einfach aufgebaut und soll vielfach in einem Level auftauchen. Während des Spielens sieht der Spieler ungefähr zehn der einfach aufgebauten Gebäude auf einmal, diese Zahl kann je nach Level und Position des Spielers variieren. Das andere Gebäude ist größer und komplexer aufgebaut, es repräsentiert eine Art Palast der pro Level nur einmal vorkommen soll.

Die Komplexität der Modelle lässt sich wie folgt darstellen:

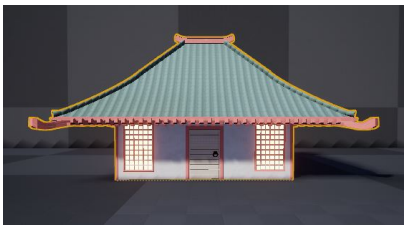
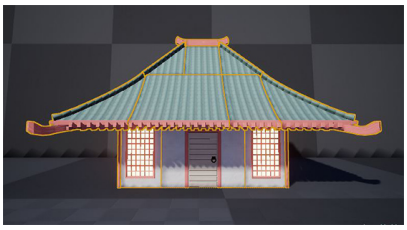
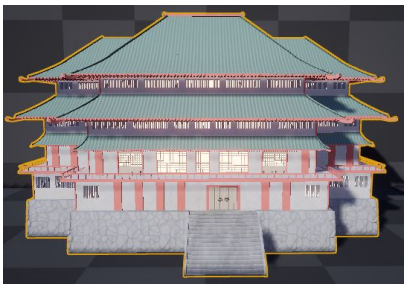
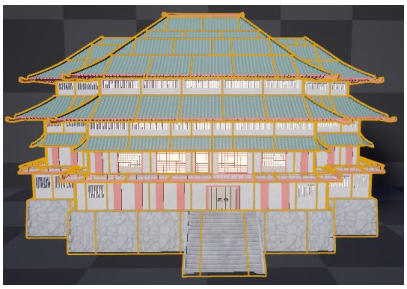
Kleines Gebäude	 26.000 Triangles, 7 Materialien	 40 Module
Großes Gebäude	 155.000 Triangles, 8 Materialien	 342 Module
	Normale Ausführung	Modulare Ausführung

Tabelle 6.1: Aufbau und Eigenschaften der für die Performance-Test genutzten Gebäude.

[17] Der PC ist aus folgenden Komponenten zusammengesetzt: INTEL CORE i7-4790K 4.0GHz, GeForce GTX 970 4GB Grafikkarte, 16GB Arbeitsspeicher und 1TB HDD Festplatte.

Für die Tests wurde ein leeres Level erstellt, in das eine Bodenebene und ein direktionales Licht eingefügt wurde. Mit Hilfe von Blueprints wurde das Level mit verschieden viel Gebäuden einer Art gefüllt. Nach Berechnung des Lichts wurden im Playmode verschiedene Werte gemessen. Der Test konnte auf der verwendeten Hardware mit 500 der großen Gebäude nicht durchgeführt werden.

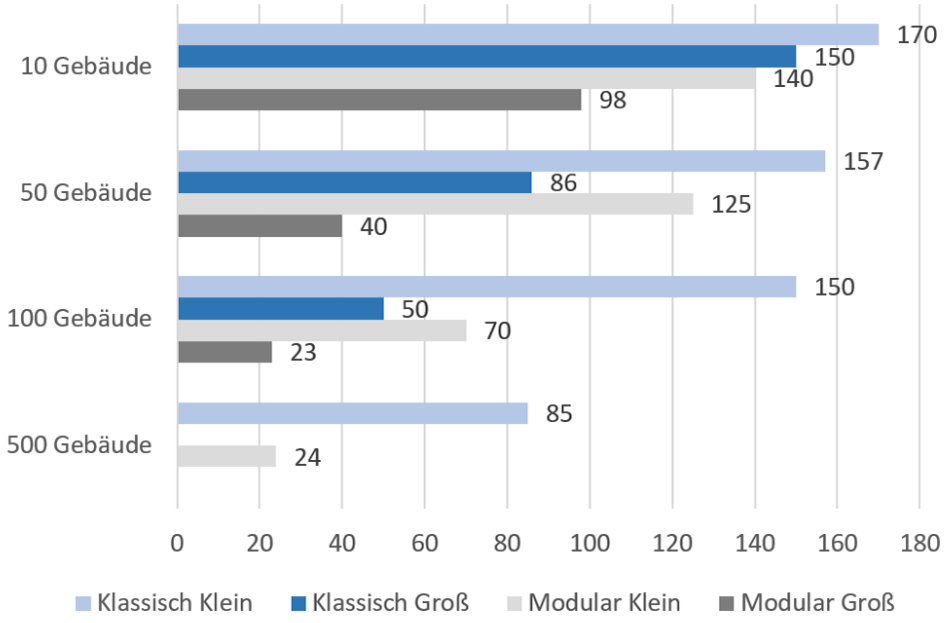


Abbildung 6.1: Übersicht der Bildfrequenzen aus den durchgeführten Tests.

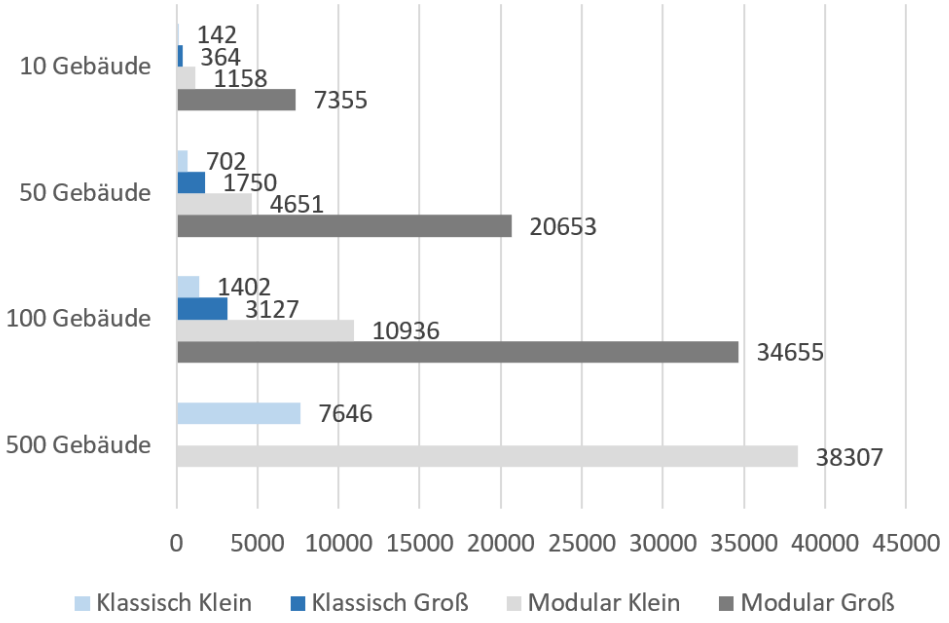


Abbildung 6.2: Übersicht benötigter Draw Calls aus den durchgeführten Tests.

Die Diagramme (Abbildung 6.1 und 6.2) zeigen, dass die Performance der nichtmodularen Gebäude deutlich besser ist. Die nötigen Draw Calls und die resultierende Last auf das System sind für die modularen Gebäude um ein Vielfaches höher als bei den nichtmodularen Gebäuden. Auch die Bildwiederholungsrate ist bei den klassischen Gebäuden bedeutend höher.

Nach den Tests wurde versucht, die Performance der modularen Elemente zu verbessern. Das in Abschnitt 3.3 beschriebene *Instancing* wird von der genutzten Version 4.21 von UE nur teilweise automatisch angewendet. Die Modelle werden nur einmal in den Speicher geladen, aber jedes Modell erzeugt für sich und seine Materialien eigene Draw Calls.

Mit Hilfe des Tools „AutoInstance“[18] aus dem Unreal Engine Marketplace wurde versucht das Instancing der Module zu verbessern.

Das Tool konnte die Draw Calls drastisch verringern. Trotzdem konnte die Performance nicht verbessert werden, da durch das *Instancing* an anderer Stelle neue Last entstand. Zudem wurde die Zuweisung der Materialien durch das Instanziieren der Module verändert und konnte nicht mehr angepasst werden.

Die Instanziierung des Tools kann auch manuell durchgeführt werden, jedoch würde dies zu unverhältnismäßig viel Arbeit führen und müsste für jedes Level neu durchgeführt werden.

In UE können mehrere Objekte zu einem verbunden werden. Bei Tests dieser Funktion zeigte sich, dass bei der Verbindung von Modulen alle Material Slots erhalten bleiben. Dies hat zur Folge, dass wieder mehr Draw Calls erzeugt werden, als wenn das Gebäude in einem Stück in UE importiert werden würde. Das große Gebäude aus dem Performance-Test hatte nach dem Verbinden in der Engine 737 zugewiesene Materialien anstelle von 8.

Da die Performance-Unterschiede auch schon bei einer für das Spiel normalen bis geringen Anzahl von gleichzeitig sichtbaren Gebäude deutlich waren und es zu diesem Zeitpunkt keine Lösung gab, dies zu ändern, wurde davon abgesehen, das Kit in UE zu nutzen.

Die schon in UE erstellten Gebäude wurden in Blender nachgebaut und in UE importiert.

Da nach den Tests in UE normale Modelle genutzt werden, ist der Einfluss auf die Performance identisch mit dem von normal erstellten Assets.

Visuelle Qualität

Gebäude, die mit dem neuen Kit erstellt wurden, erreichen das gleiche Level an Detail und Qualität, wie die schon bestehenden Assets und fügen sich sehr gut in den Spielstil ein.

[18] Das Tool kann unter: https://www.unrealengine.com/marketplace/en-US/slug/auto-instance abgerufen werden.

Tests bezüglich der Kunstermüdung (siehe Abschnitt 4.1.1) können aufgrund von fehlender Assets für ein gesamtes Level und der begrenzten Zeit nicht durchgeführt werden. Alle bisherigen Level wurden auf der Basis von neun Gebäuden erstellt. Mit dem modularen Set können weit mehr als neun verschiedene Gebäude erstellt werden. Das Nutzen des Kits sollte demnach keinen negativen Effekt auf das optische Erscheinungsbild des Levels haben.

Durch weitere Varianten der Dachmodule könnte das Kit noch verbessert werden. Die Dächer aller Gebäude sehen gleich aus, auch wenn sie andere Formen haben.

Durch das Zusammensetzen der Module in Blender kann jedes Modell, neben dem Kombinieren verschiedener Module, auch mit einem individuellen Look versehen werden, indem das erstellte Gebäude durch Modifikationen der Meshes abgeändert wird. Hierdurch lässt sich die Qualität der Module weiter erhöhen, es wird aber auch weitere Arbeitszeit benötigt.

Die von Burgess und Purkeypile erwähnte Boxartigkeit von modularen Assets ist auch in diesem Set zu erkennen. Dies ist jedoch zum Teil auf die Bauweise der Gebäude zurückzuführen, welche durch das Kit abgebildet werden sollten und fällt in diesem Fall nicht negativ auf.

Nutzerfreundlichkeit

Die Nutzerfreundlichkeit des Kits sollte durch einen Level-Designer geprüft werden. Da die Modelle nach dem Performance-Test in Blender vorgefertigt werden müssen, wäre das Erstellen von Gebäuden in der Engine durch den Level-Designer überflüssige Arbeit und wurde ausgelassen. Durch die durchgeführten Tests und die dadurch entstandenen Gebäude, wurde dennoch ein guter Eindruck der Handhabung vermittelt.

Gebäude von kleiner bis mittlerer Größe lassen sich schnell erstellen. Steigt die Größe des Gebäudes weiter an, erhöht sich die benötigte Zeit jedoch drastisch. Für größere Gebäude wäre es von Vorteil, wenn das Kit größere Elemente besitzen würde. Da aber hauptsächlich kleine bis mittlere Gebäude erstellt werden sollen, fällt dies nur gering negativ auf.

Da in Blender die gleichen Methoden wie in UE genutzt werden können und zusätzlich weitere Methoden zur Verfügung stehen, wurde die Nutzerfreundlichkeit in diesem Aspekt noch verbessert.

Aufwand

Das Erstellen der Assets hat viel Zeit gekostet. Vor allem zu Beginn war es schwierig das Konzept der Modularität zu greifen und die Gebäude richtig in passende Module aufzuteilen. Hierfür wurde die meiste Zeit der Planung aufgebracht.

Nur für das eine Level wäre es vermutlich schneller gewesen, einzelne Gebäude zu erstellen. Werden noch weitere Level in dem gleichen Setting entwickelt, kann der Arbeitsaufwand wohl ausgeglichen werden.

Könnte das Kit, wie geplant, in UE genutzt werden, hätte die benötigte Arbeitszeit, um Gebäude zu erstellen, besser auf das Team verteilt werden können. Das Erstellen und Anpassen der Gebäude aus den vorgefertigten Modulen wäre von den Level-Designern übernommen worden. So wäre es möglich gewesen, dass der 3D-Artist ohne Unterbrechungen an weiteren Assets für das Level arbeitet.

Durch das Vorbauen der Modelle in Blender kann die Aussage von L. Durand bezüglich der leichten und schnellen Anpassung der Gebäude auf das Gameplay nicht überprüft werden.

Die Bewertung des Kits ist hier beendet. Im Folgenden wird zuerst darauf eingegangen, welche der zuvor erarbeiteten Methoden im Verlaufe des Projektes angewandt wurden und welchen Nutzen diese hatten.

6.2 Methoden

Raster

Ohne die Nutzung eines Rasters wäre das passgenaue Platzieren der Module so zeitaufwändig gewesen, dass es keinen Sinn ergeben würde ein modulares Kit zu nutzen.

Auch bei der Erstellung der Module hat das Raster eine entscheidende Rolle gespielt. Ohne Anhaltspunkte, die durch das Raster gegeben waren, wäre es sehr schwer gewesen, ein modulares System zu entwickeln, durch welches alle Teile miteinander verbunden werden können.

Die von Burgess und Purkeypile erwähnte Methode, das Raster zu verlassen (siehe Abschnitt 4.2.6) und an vorher platzierten Objekten auszurichten, wurde nicht genutzt. Dies hätte die Erstellung vieler neuer Module und Modifikationen des Kits gefordert. Die Bauweise der nachzubildenden Gebäude bedient sich in der Regel nur an rechten Winkeln, welche mit dem Kit dargestellt werden können. Es gab also auch keinen Grund diese Methode zu nutzen.

Pivot Point

Nur durch die richtige Platzierung des Pivot Points konnte das volle Potenzial des Rasters genutzt werden. Für die Positionierung des Pivot Points wurden die Regeln[19] von P. Mader genutzt. Diese waren sehr leicht anzuwenden und hilfreich, um eine praktische Position für Pivot Points zu finden.

Sein Hinweis darauf den Pivot Point bei Objekten, die einen Kreis formen, in das Zentrum des Kreises zu setzen, wurde für dieses Objekt auch bei Eckmodulen angewandt. Dies beschleunigt die Neupositionierung und Ausrichtung dieser Elemente.

[19] Eine Übersicht der Regeln kann in einem *Gamasutra*-Artikel (Mader, 2005) von P. Mader unter dem Abschnitt *Pivot Placement* gefunden werden.

Textur

Der Einsatz von kachelbaren Texturen war für das Kit unabdinglich. Darüber hinaus musste nicht viel beachtet werden außer, dass in der UV-Map die Ränder der Modelle mit den Rändern der Textur verbunden werden. Nur so war gewährleistet, dass keine sichtbaren Kanten zwischen den Modulen entstehen, die den Eindruck stören würden, dass die Gebäude aus einem Stück bestehen.

Planung und Tests

Ein weiterer wichtiger Aspekt des Projektes war das Planen und Testen. Auch wenn dies keine praktisch anwendbaren Methoden sind, wie die anderen hier aufgelisteten, sind sie nicht weniger wichtig.

Ohne die vorherige Planung und diverse Tests, hätte die Entwicklung des Kits mehr Zeit in Anspruch genommen und hätte vermutlich keine so hohe Qualität erreicht.

Mit der gewonnenen Erfahrung kann die Planungsphase bei einem nachfolgenden Projekt voraussichtlich schneller und besser durchgeführt werden. Viele der nötigen Änderungen am Kit hätten durch mehr Praxiserfahrung schon in der ersten Planungsphase erkannt werden können, was den generellen Ablauf des Projektes beschleunigt hätte.

Auf Tests und eventuelle Weiterentwicklungen des Kits sollte dennoch nicht verzichtet werden. Die in der Planungsphase erstellten Regeln sollten angepasst werden, wenn dadurch das Kit verbessert wird, auch wenn dadurch mehr Arbeit entsteht. Je nach Projekt wird das Kit an vielen Stellen und für lange Zeit genutzt und sollte daher so gut wie möglich ausgearbeitet sein.

Varianten

Die Subkits bestehen aus allen nötigen Teilen, um verschiedene Gebäude zu bauen. Diese hätten alle ein sehr ähnliches Erscheinungsbild, wenn das Kit keine Varianten der Module enthalten würde. Bisher wurden nur von den Wand-, Balkon- und Holzbalkenelementen Varianten erstellt. Nur durch diese lassen sich schon viele individuell aussehende Gebäude erzeugen. Die Erzeugung weiterer Varianten nimmt nur wenig Zeit in Anspruch, da von einem Basis Modell aus gearbeitet werden kann, welches schon an die Regeln des Kits angepasst wurde.

Das Nutzen von Varianten ist also ein effizienter Weg ein Kit zu ergänzen.

Modularitätsstufen

In diesem Projekt wurde eine mittlere bis kleine Stufe der Modularität gewählt. Folglich ist es möglich mit einer kleinen Auswahl an Modulen viele verschiedene Objekte zu erstellen. Würde ein Gebäude aus nur drei Modulen bestehen, müssten viel mehr Module erstellt werden, um die gleiche Abwechslung zu gewährleisten.

Wie schon erwähnt wurde die Spielperformance durch die vielen genutzten Elemente negativ beeinflusst. Die Lösung, die Gebäude in Blender vorzufertigen, verringert nicht die positiven Aspekte der gewählten Stufe, da diese auch in Blender

genutzt werden können.

Benennung

Eine gut konzipierte Benennung der Module ist, wie von Burgess und Purkeypile angeführt, ein sehr wichtiger Aspekt, um Arbeitsabläufe mit dem Kit zu beschleunigen und die Kommunikation zu verbessern. Wie zuvor beschrieben, muss darauf geachtet werden, eine sinnvolle Bezeichnung der Module zu entwickeln, die leicht zu interpretieren und zu erlernen ist. Sind bei Tests Probleme mit Modulen aufgefallen, konnten diese leicht durch ihren Namen zugeordnet werden.

Struktur

Zusammen mit der Benennung war das Einführen einer Ordnerstruktur sehr hilfreich, um den Prozess der Gebäudeerstellung in UE zu beschleunigen. Auch wenn dies nur zu einem gewissen Grad getestet wurde. Des Weiteren konnte durch das Einteilen der Module in Subkits der Erstellungsprozess besser organisiert und die Übersicht in UE dadurch deutlich verbessert werden.

Vertex Farben

Das Einfärben von Vertices, um die Erscheinung der Texturen bzw. die des Models zu verändern, wurde nicht angewandt. Hierfür hätten weitere Module erstellt werden müssen. Da das Kit auch ohne diese Methode schon genug Abwechslung aufwies, wurde darauf verzichtet. Aufgrund des veränderten Arbeitsablaufes könnte dies ohne großen Aufwand für neue Gebäude genutzt werden, um mehr Abwechslung zu schaffen ohne neue Module zu entwickeln.

7 Fazit und Ausblick

7.1 Fazit

Durch das Aufzeigen des Einflusses von Modularität auf die Entwicklung von Videospielen und die Realität, konnte ein erster Eindruck von dessen Nutzen gewonnen werden. Zudem wurde dargelegt, das dieses Prinzip schon lange angewandt wird und stetig weiterentwickelt wird. Die Erarbeitung des normalen Ablaufs der Asseterstellung und wie dieser durch spezielle Methoden auf die Anwendung von modularem Design angepasst wird, hat gezeigt, welche Tiefe dieses Konzept besitzt. Durch die Anwendung der Methoden und Konzepte konnten diese bewertet und eine Einschätzung von modularem Design erstellt werden.

Eine ausführliche Bewertung der genutzten Methoden wurde schon im vorherigen Kapitel durchgeführt. Zusammenfassend lässt sich sagen, dass die wichtigsten Methoden, um modulares Design zu realisieren, die folgenden sind:

- Raster,
- Pivot Point,
- Planung,
- Tests und
- kachelbare Texturen.

Alle weiteren Methoden verbessern den Prozess, aber ohne die zuvor genannten, ist die praktische Umsetzung eines modularen Kits nicht möglich. Für andere Ansätze von Modularität kann diese Bewertung abweichen.

Durch die vielen schon vorhandenen Methoden, welche trotz ihres Alters alle noch relevant sind und meist gut miteinander funktionieren, ist es leicht einen Einstieg in modulares Design zu finden. Ohne viel Zeit in die gründliche Planung zu investieren, lässt sich das Konzept jedoch nur schwer umsetzen.

Auf Grund dieser großen Zeitinvestition vorab, muss für jedes Projekt geprüft werden, bis zu welchem Grad es sinnvoll ist Modularität zu nutzen. Lassen sich Assets nicht wieder verwerten oder wird immer nur eine geringe Anzahl ähnlicher Objekte genutzt, sollte von einem modularen Kit abgesehen werden. Dennoch kann für die Erstellung der Assets, wie von L. Durand (siehe Abschnitt 2.2) beschrieben, Modularität genutzt werden, um diesen Prozess zu beschleunigen. Dies wurde im praktischen Abschnitt dieser Arbeit nicht genau getestet. Da für die Erstellung der Varianten von Modulen aber ein ähnliches Vorgehen genutzt wurde, kann dies zu einem gewissen Umfang verifiziert werden.

Das modulares Design einen hohen Nutzen hat, kann nur bestätigt werden. Mit Hilfe des erstellten Kits lassen sich beliebig viele Kombinationen an Gebäuden erstellen und durch die mögliche Erweiterung, mit weiteren Varianten der Module wird ein abwechslungsreiches Erscheinungsbild gewährt. Für alle weiteren Level, die dem gleichen Setting folgen, können Gebäude aus dem Kit erstellt werden.

Dies bedeutet eine sehr große Zeitersparnis für die Zukunft, ohne dass immer die exakt gleichen Gebäude für die Level genutzt werden müssen, wie es bisher der Fall war.

Da bei diesem Projekt zum ersten Mal mit modularem Design gearbeitet wurde und wenig Praxiserfahrung vorhanden ist, können die Ergebnisse bei der Bearbeitung durch ein erfahreneres Team abweichen. Die, von vielen Autoren angesprochene, Performance-Verbesserung konnte nicht verifiziert werden. Dies Lag mit aller Wahrscheinlichkeit an der Nutzung einer falschen Modularitätsstufe oder einem anderen Fehler, der mit mehr Erfahrung vielleicht nicht passiert wäre. Die Effektivität der angewandten Methoden konnte dennoch gut eingeschätzt werden.

Tests bezüglich der visuellen Qualität des Ergebnisses waren auf Grund der zur Verfügung stehenden Mittel nur begrenzt möglich. Für eine bessere Evaluierung der visuellen Qualität der Assets, hätte ein fertiges Level mit den erstellten Assets von Testpersonen getestet werden müssen.

Die erarbeiteten Methoden und Ergebnisse können für ähnliche Projekte sicherlich hilfreich sein, auch wenn die Performance-Problematik nicht gelöst wurde.

Die Zusammenfassung der Ergebnisse ist hier abgeschlossen. Im folgenden Abschnitt wird darauf eingegangen, wie mit den Ergebnissen verfahren wird und ein Ausblick in die Zukunft präsentiert.

7.2 Ausblick

Grundlegend kann gesagt werden, dass die Möglichkeit modulares Design anzuwenden eine wichtige Erweiterung der Fähigkeiten eines 3D-Artists ist, um dem Trend der immer größeren und realistischeren Spielwelten gewachsen zu sein.

Für die weitere Entwicklung von *Renegade Line* wird versucht mit der Version 4.22 von UE noch einmal den modularen Ansatz in UE zu verfolgen. Mit dieser neuen Version wurde die Instanziierung von Objekten verbessert (Wassmer, 2019). Zuvor muss das Spiel allerdings auf diese Version portiert werden.

Zudem ist geplant, in der Zukunft Gebäude mit Innenräumen auszustatten, um ein komplexeres Spielerlebnis zu ermöglichen. Es bietet sich an für dieses Vorgehen Modularität zu nutzen. Mithilfe der erarbeiteten Erkenntnisse kann dies voraussichtlich leichter umgesetzt werden, auch wenn für dieses Anwendungsgebiet sicherlich Anpassungen vorgenommen werden müssen.

Generell wäre es spannend, die erlernten Methoden an anderen Projekten anzuwenden, die einem anderen Stil folgen und unterschiedliche Stufen von Modularität benötigen. Die Möglichkeiten, die dieses Thema bietet, sind, genau wie Modularität selbst, unbegrenzt.

Abbildungsverzeichnis

Wenn nicht anders angegeben, eigene Grafik.

2.1 Beispielhaftes modulares Kit eines Rohr-Systems aus *Fallout 3* (Bethesda Softworks, 2008), (Burgess & Purkeypile, 2013). 6

2.2 Ausschnitt eines Original Entwurfs eines *Super Mario Bros.* Levels. In Anlehnung an (Nintendo, Miyamoto, S., Tezuka T., 2015). 7

2.3 Analyse der Tiles in *Super Mario Bros.*. In Anlehnung an (Earl, 2018). 8

2.4 Beispiele für Modulare Dungeons (Burgess & Purkeypile, 2016a). . 9

2.5 Asymmetrische Tunnelmodule aus *Skyrim* (Burgess & Purkeypile, 2013). 10

2.6 Modulare Assets in *For Honor*. In Anlehnung an (Durand, 2019). . . 11

2.7 Beispiele der zwei genannten Methoden für den Bau modularer Gebäude. 12

2.8 Das Raster nach dem alle LEGO-Steine Entwickelt werden. In Anlehnung an (Hugo, 2018, S. 11). 13

3.1 Darstellung der Bestandteile eines Meshes. 15

3.2 Ein Grabstein vor und nach dem Anwenden des Blender Remesh Modifierers (Burrows, 2015). 17

4.1 Beispiel für Überlappung und Lücken zwischen Modulen (Mader, 2005). 23

4.2 Module für den Innenraum einer Raumstation. In Anlehnung an (Olson, 2018). 26

4.3 Beispiele für verschiedene Modulraitätsstufen eines Gebäude Kits (Perry, 2002). 27

4.4 Beispiele für die Anwendung verschiedener Rastergrößen (Mader, 2005). 28

4.5 Beispiel für verschiedene Qualitäten von Kontaktpunkte (Klafke, 2010). 29

4.6 Beispiel für den Einsatz eines am Ende einer Mauer platzierten Pivot Points (Lin, Lentz, Sturgill & Reed, o.D.). 30

4.7 Beispiel für die Positionierung eines Pivot Points an einer Stuckleiste (Mader, 2005). 31

4.8 Beispiel für einen Pivot Point im Zentrum eines Kreises (Mader, 2005). 32

4.9 Beispiel für Namensgebung eines modularen Assets aus *Fallout 3* (Burgess & Purkeypile, 2013). 34

4.10 Beispiele für Einfluss von Vertex Farben (Klevestav, 2009). 35

4.11 Beispiel für Mehrfachnutzung eines Trimsheets (Klafke, 2010). . . 36

4.12 Ein Beispiel für das Potential austauschbarer Texturen. In Anlehnung an (Burgess & Purkeypile, 2016a). 36

5.1 Ausschnitt eines aktuellen Levels in dem bisherigen Setting. 37

5.2 Konzept für Farbgebung und Aufbau der Gebäude (Pampin, 2018). 39

5.3 Skizze der zunächst definierten Module. In Anlehnung an (Muzachan, 2012). 40

5.4 Erste Prototypen der geplanten Module. 42

5.5 Ursprüngliches Wandelement (links) und überarbeitete Version mit abgelöstem Dach (rechts). 42

5.6 Mögliche Kombinationen von Balkon- und Wandelementen. 43

5.7 In Wandelemente eingesetzte Holzbalken mit Sonderelement. . . 43

5.8 Überarbeitete Wandelemente und Holzbalken aus neuem Subkit. . 43

5.9 Ursprüngliches Elemente für den Abschluss des Daches (blau) und neue Elemente für das schließen spezifischer Lücken (grün). 44

5.10 Prototypen des weiterentwickelten Systems. 44

5.11 Alle Module mit angewandten Texturen. 45

5.12 Ausschnitte aller benötigten Diffuse- und Normalmaps. 46

5.13 Die für die meisten Modelle genutzten Rougness- und Specularmaps. 47

5.14 Einfluss der Specular- und Roughnessmap (links) im Vergleich mit den Standartwerten aus UE (rechts). 47

5.15 Harte Kanten an Modulübergängen von Modulen. 48

5.16 Aus dem Kit erstellte Gebäude in einem alten Level. 49

6.1 Übersicht der Bildfrequenzen aus den durchgeführten Tests. . . . 51

6.2 Übersicht benötigter Draw Calls aus den durchgeführten Tests. . . 51

Tabellenverzeichnis

5.1 Übersicht aller Subkits und deren Bedeutung 41
5.2 Bezeichnungsschemata der genutzten Module mit einem Beispiel 41

6.1 Aufbau und Eigenschaften der für die Performance-Test genutzten Gebäude. 50

Literaturverzeichnis

Letzter Zugriff auf alle Websites am 08.08.2019

Altice, N. (2015). The long shadow of Super Mario Bros. Zugriff unter https://www.gamasutra.com/view/news/253377/The_long_shadow_of_Super_Mario_Bros.php

Barrett Byrd Associates (2011). The prefabricated bedroom modules sit on a two storey steel frame. Zugriff unter https://www.newsteelconstruction.com/wp/modular-construction-checks-in/

Beck, T. (2017). *Blender 2.7 : Das umfassende Handbuch*. Rheinwerk.

Blender Documentation Team (o.D.). Blender 2.80 Reference Manual. Zugriff unter https://docs.blender.org/manual/en/latest/modeling/meshes/editing/index.html

Burgess, J. & Purkeypile, N. (2013). Skyrim's Modular Approach to Level Design. Zugriff unter http://blog.joelburgess.com/2013/04/skyrims-modular-level-design-gdc-2013.html

Burgess, J. & Purkeypile, N. (2016a). [Präsentation] "Fallout 4's"Modular Level Design. Zugriff unter https://gdcvault.com/play/1022930/-Fallout-4-s-Modular

Burgess, J. & Purkeypile, N. (2016b). [Video] "Fallout 4's"Modular Level Design. Zugriff unter https://gdcvault.com/play/1023202/-Fallout-4-s-Modular

Burrows, A. (2015). What To Know When Creating Next Gen Assets. Zugriff unter https://cgmasters.net/free-tutorials/what-to-know-when-creating-next-gen-assets/

Coss, D. (o.D.). Making modular videogames. Zugriff unter https://codewords.recurse.com/issues/three/making-modular-videogames

Crecente, B. (2012). Borderlands 2's amazing weapons and their bizarre inspirations, from Apple to Austrian cars. Zugriff unter https://www.polygon.com/gaming/2012/4/4/2924678/borderlands-2s-amazing-weapons-and-their-bizarre-inspirations

Durand, L. (2019). Environment Design for AAA Games with Laurie Durand. Zugriff unter https://80.lv/articles/environment-design-for-aaa-games-with-laurie-durand/

Earl, M. (2018). Extracting Super Mario Bros levels with Python. Zugriff unter http://matthewearl.github.io/2018/06/28/smb-level-extractor/

Elderscrollsportal (2019). Daggerfall:The Elder Scrolls II: Daggerfall. Zugriff unter https://www.elderscrollsportal.de/almanach/Daggerfall:The_Elder_Scrolls_II:_Daggerfall

FlippedNormals (2019). [Video] Texture Maps Explained - Essential for All Texture Artists. Zugriff unter https://www.youtube.com/watch?v=ZOHNRlrd1Ak

GameDesigning (2019). The Top 10 Video Game Engines. Zugriff unter https://www.gamedesigning.org/career/video-game-engines/

Hider, J. (2017). Real Time Rendering. Zugriff unter https://jesshiderue4.wordpress.com/real-time-rendering-an-overview-for-artists/

Hugo, S. (2018). *LEGO – Absolut alles, was du wissen musst: Das Beste aus dem LEGO® Universum* (1. Aufl.). Dorling Kindersley.

Kane, A. (2019). Morrowind: An oral history. Zugriff unter https://www.polygon.com / 2019 / 3 / 27 / 18281082 / elder - scrolls - morrowind - oral - history - bethesda

Klafke, T. (2010). Creating modular environments in UDK. Zugriff unter http://www.thiagoklafke.com/modularenvironments.html

Klevestav, P. (2009). Working with modular sets. Zugriff unter http://www.philipk.net/tutorials/modular_sets/modular_sets.html

Lawson, R. & Ogden, R. (2010). Sustainability and Process Benefits of Modular Construction. Zugriff unter https://www.irbnet.de/daten/iconda/CIB18783.pdf

Lin, Lentz, Sturgill & Reed (o.D.). Using Workflow Techniques and Modularity. Zugriff unter https://api.unrealengine.com/udk/Two/WorkflowAndModularity.html

Mader, P. (2005). Creating Modular Game Art For Fast Level Design. Zugriff unter https://www.gamasutra.com/view/feature/130885/creating_modular_game_art_for_fast_.php

Medina-Gray, E. (2014). Modular Structure and Function in Early 21st-Century Video Game Music. Zugriff unter https://www.academia.edu/7594328/Modular_Structure_and_Function_in_Early_21st-Century_Video_Game_Music

Meler, M. (2018). Modular Concepts for Game and Virtual Reality Assets. Zugriff unter https://software.intel.com/en-us/articles/modular-concepts-for-game-and-virtual-reality-assets

Muza-chan (2012). Matsumoto Castle, Matsumoto. Zugriff unter https://muza-chan.net/japan/index.php/blog/japanese-castle-architecture-renketsushiki-style

Nintendo, Miyamoto, S., Tezuka T. (2015). [Video] Super Mario Bros. 30th Anniversary Special Interview ft. Shigeru Miyamoto & Takashi Tezuka. Zugriff unter https://youtu.be/DLoRd6_a1CI?t=163

Norris, J. (2015). Modular Building Set. Zugriff unter http://images.purepolygons.com/building_tut/Building_Breakdown.pdf

O'Conor, K. (2017). GPU Performance for Game Artists. Zugriff unter http://fragmentbuffer.com/gpu-performance-for-game-artists/

Olson, M. (2018). Modular Sci-Fi Level in UE4. Zugriff unter https://80.lv/articles/004adk-modular-sci-fi-level-in-ue4-by-matt-olson/

Pampin, M. (2018). Artstation Submission Feudal Japan: The Shogunate Environment Design. Zugriff unter https://www.artstation.com/contests/feudal-japan/challenges/54/submissions/35789?sorting=latest

Perry, L. (2002). Modular Level and Component Design. Zugriff unter https://api.unrealengine.com/udk/Three/rsrc/Three/ModularLevelDesign/ModularLevelDesign.pdf

Rehfeld, G. (2013). *Game Design und Produktion: Grundlagen, Anwendungen und Beispiele*. Carl Hanser Verlag. Zugriff unter https://books.google.de/books?id=c7dPAgAAQBAJ

Smith, R. E. (2011). *Prefab Architecture: A Guide to Modular Design and Construction*. Wiley.

The LEGO Group (o.D. a). [Online Shop] The Lego Group Internetshop Unterkategorie Creator 3 in 1. Zugriff unter https://shop.lego.com/de-DE/category/creator-3-in-1

The LEGO Group (o.D. b). LEGO® Creator Expert – Modular Buildings 10th anniversary. Zugriff unter https://www.lego.com/en-us/themes/creatorexpert/explore/modular-buildings-10th-anniversary

Triumph Modular (o.D.). Modular design: Sedona's Economic Future. Zugriff unter https://triumphmodular.com/blog/modular-design-sedona-economic-future/

Ubisoft Entertainment (o.D.). For Honor Ubisoft. Zugriff unter https://forhonor.ubisoft.com/game/en-us/game-info/index.aspx

Vergne, F. (2017). The Strings of Environment Design. Zugriff unter https://80.lv/articles/the-strings-of-environment-design/

Ward, J. (2008). What is a Game Engine? Zugriff unter http://www.gamecareerguide.com/features/529/what_is_a_game.php

Wassmer, M. (2019). [Video] Refactoring the Mesh Drawing Pipeline for Unreal Engine 4.22 | GDC 2019 | Unreal Engine. Zugriff unter https://youtu.be/qx1c190aGhs?t=2128

Wikipedia (2019). Modularität - Wikipedia, Die freie Enzyklopädie. Zugriff unter %5Curl%7Bhttps://de.wikipedia.org/w/index.php?title=Modularit%C3%A4t&oldid=189371860%7D

Williams, A. (2017). *History of Digital Games: Developments in Art, Design and Interaction*. CRC Press. Zugriff unter https://books.google.de/books?id=nLsrDwAAQBAJ

Ich versichere, die vorliegende Arbeit selbstständig ohne fremde Hilfe verfasst und keine anderen Quellen und Hilfsmittel als die angegebenen benutzt zu haben. Die aus anderen Werken wörtlich entnommenen Stellen oder dem Sinn nach entlehnten Passagen sind durch Quellenangaben eindeutig kenntlich gemacht.

Hamburg, 14.08.2019 Nikolai Stecker