Höhere Technische Bundeslehranstalt Wien 3, Rennweg
Rennweg 89b
A-1030 Wien, Tel +43 1 24215-10

Diplomarbeit

CitySketch

CitySketch – das Blender Add-on zum Generieren von 3D Städten

ausgeführt an der
Höheren Abteilung für
Informationstechnologie

im Schuljahr 2020/2021

durch
Kruk Kamil

Rabas Felix

San Diego Franz-Steven

Tran Quoc Huy

unter der Anleitung von

Prof. Matejowsky Peter

Prof. Bayandor Mitra

Prof. Weiss Florian

Wien, Mai 2021

Kurzfassung

Durch den immer steigenden Konsum an digitalen Medien, eine Entwicklung, die in absehbarer Zeit nicht aufhören wird, werden digitale 3D-Grafiken immer häufiger angewendet. Eine der wichtigsten Komponenten dieser Entwicklung, ist die Gestaltung realitätsnaher 3D Städte welche in der Film-, Architektur- und Videospielindustrie nicht mehr wegzudenken sind. Das Erstellen solcher Städte wird noch immer mit einer besonders repetitiven und zeitintensiven Aktivität verbunden, bei der die gestalterische Seite leicht verloren geht.

Ziel dieser Diplomarbeit ist es eine Erweiterung für die populäre sowie kostenfreie open-source 3D-Software Blender zu programmieren, die den Benutzer bei der Umsetzung solcher Städte maßgeblich unterstützt. Durch vormodellierte 3D-Assets erreichen wir eine Simplifikation des gesamten Prozesses auf nur wenige Klicks, die es sogar einer unerfahrenen Person erlaubt, unsere Erweiterung zu verwenden.

Im Laufe der Diplomarbeit wurden Generatoren für die auffallendsten Strukturen einer echten Stadt: Gebäude, Straßen, Grünzonen sowie Flüsse programmiert und als Gesamtpaket in einer Erweiterung umgesetzt. Passend dazu wurden die nötigen 3D-Assets in zwei verschiedenen Stilen modelliert, texturiert und animiert.

Abstract

Due to the significant rise in consumption of digital media, a development that is expected to continue in the foreseeable future, digital 3D-Graphics have seen a spiking increase in usage. One of the most important spots of this development can be attributed to the design of realistic 3D-cities that have become major components in the film, architecture, and video game industry. Regrettably, the process of designing a realistic city still involves many repetitive tasks and is therefore very time consuming.

The goal of this diploma thesis is the development of an extension for the popular and free open-source 3D software Blender, that significantly eases this process. We achieve this by providing premade 3D assets that dramatically simplify this task and ultimately allow anyone with no prior experience to generate realistic cities with a few clicks.

During this thesis, several generators for the most striking visual elements of a city have been developed. This includes generators for buildings, roads, green zones, and rivers that have been implemented in a single extension. Additionally, the appropriate 3D assets have been modelled, textured, and animated in two distinguished styles.

Ehrenwörtliche Erklärung

Ich versichere,

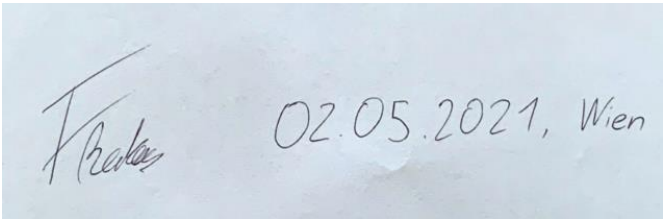
- dass ich meinen Anteil an dieser Diplomarbeit selbstständig verfasst habe,
- dass ich keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe
- und mich auch sonst keiner unerlaubten Hilfe bzw. Hilfsmittel bedient habe.

Wien, am 02.05.2021

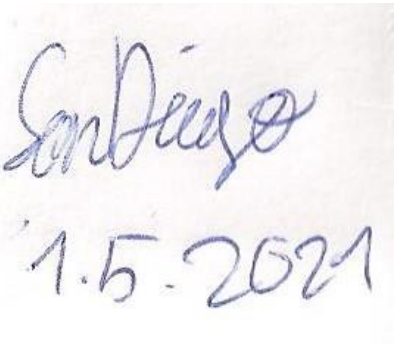
Kruk Kamil

Signiert von: Kamil Kruk
Datum: 02.05.2021 13:11:26
Dieses mit einer qualifizierten elektronischen Signatur versehene Dokument hat gemäß Art. 25 Abs. 2 der Verordnung (EU) Nr 910/2014 vom 23. Juli 2014 ("eIDAS-VO") die gleiche Rechtswirkung wie ein handschriftlich unterschriebenes Dokument.
Dieses Dokument ist digital signiert!
Prüfinformation:
Informationen zur Prüfung der elektronischen Signatur finden Sie unter:
www.handy-signatur.at
www.a-trust.at
A-TRUST einfach sicher

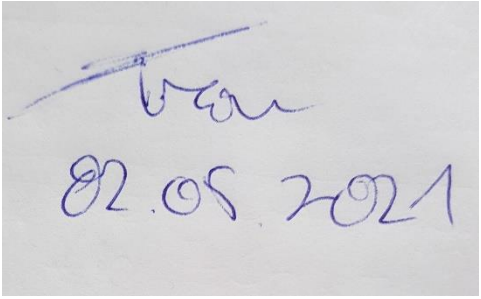
Rabas Felix



San Diego Franz-Steven



Tran Quoc Huy



Präambel

Die Inhalte dieser Diplomarbeit entsprechen den Qualitätsnormen für „Ingenieurprojekte“ gemäß § 29 der Verordnung des Bundesministers für Unterricht und kulturelle Angelegenheiten über die Reife- und Diplomprüfung in den berufsbildenden höheren Schulen, BGBl. Nr. 847/1992, in der Fassung der Verordnungen BGBl. Nr. 269/1993, Nr. 467/1996 und BGBl. II Nr. 123/97.

Liste der betreuenden Lehrer
Prof. Peter Matejowsky
Prof. Bayandor Mitra
Prof. Weiss Florian

Liste der Kooperationspartner:
- ❖ INCO Innovative Computerlösungen Gesellschaft m.b.H.

Inhaltsverzeichnis

1 Ziele

1.1 Einleitung

Die Idee für „CitySketch – das Blender Add-on zum Generieren von 3D Städten“ ist während der Arbeiten mit Blender entstanden. Beim Modellieren einer vollständigen 3D-Stadt kommen zahlreiche Aufgaben vor, die man mit einem Add-on optimieren und beschleunigen kann.

CitySketch setzt hier an. Das Add-on generiert automatisch eine vollständige 3DStadt, ermöglicht dem Benutzer aber optional die Einstellungen und Parameter zu verändern. Entscheidend ist die „One-Click“ Generierung, die dem Benutzer erlaubt, mit nur einem Mausklick, die Stadt zu generieren.

CitySketch unterschiedet sich von anderen City-Generator Add-ons dadurch, dass man die Umrisse der Stadt, sowie wichtige Stadtzentren, mit einem Stift einzeichnen kann.

1.2 Individuelle Zielsetzung

Jedes Ziel ist einem hier angeführten Hauptverantwortlichen zugeordnet, trotzdem garantiert dies nicht dessen eigenständige Umsetzung durch diese Person, da die Methodik der „Paar-Programmierung“ oftmals angewendet wurde. Nichtsdestotrotz liegt die Verantwortung der vollständigen und korrekten Umsetzung eines Zieles in den Händen des Verantwortlichen.

Straßensystem Generator (RE-H 1, Rabas)

Das Add-on ist in der Lage komplexe Straßensysteme unter Berücksichtigung der Kreuzungen durch einen selbstgeschriebenen Algorithmus zu erstellen und aus den Straßen-3D-Assets zusammenzubauen.

Gebäude Generator (RE-H 2, Kruk)

Das Add-on ist in der Lage aus einem vordefinierten Pool an Gebäuden, unter Berücksichtigung der Dimensionen und der Priorität die passenden auszuwählen und auf einer Fläche, unter Berücksichtigung der Straßen zu Platzieren.

Grünzonen Generator (RE-H 3, Rabas)

Das Add-on ist in der Lage Grünzonen an den vom Benutzer markierten Stellen zu erzeugen.

Animationen (RE-H 4, Tran)

Das Add-on unterstützt animierte Assets wie Autos und Flugzeuge/Hubschrauber. Die Assets sind im High-Poly Asset Pack enthalten.

High-Poly Asset Pack (RE-H 5, San Diego)

Ein Asset Pack mit mindesten 5 realistischen Gebäuden mit vielen Details und passenden Texturen sowie allen nötigen Grünzonen-Modellen für RE-H 3 und zusätzlichen Assets die eine Stadt ausmachen wie Straßenlaternen, Schilder, Mistkübel, usw.., welche in Cycles gerendert werden, ist vorhanden.

Low-Poly Asset Pack (RE-H 6, Tran)

Ein Asset Pack mit mindestens 5 Gebäuden die wenige „Vertices“ (Eckpunkte) besitzen und in Cycles gerendert werden können, ist vorhanden.

Blender Add-on (RE-H 7, Kruk)

Ein Blender Add-on mit entsprechendem Interface zum Bedienen der Module „Straßensystem Generator“, „Gebäude Generator“ und „Fluss Generator“ ist vorhanden und funktioniert ab der Blender Version 2.9x.

Projekt Management (RE-H 8, Kruk)

Der Scrum-Master leitet das Projekt nach den Richtlinien der agilen Projektmanagement Methode Scrum.

Produkt Video (RE-H 9, San Diego)

Ein 2-minütiges Produkt Video, welches die Funktionalität des Add-on erklärt, sowie einen Rundflug um die generierte Stadt zeigt, ist vorhanden.

Fluss Generator (RE-O 1, Rabas)

Das Add-on ist in der Lage, einen Fluss entlang einer vom Benutzer skizzierten Linie zu generieren.

Gerenderte Bildergalerie (RE-O 2, San Diego)

Mindestens 5 Bilder mit einer vom Add-on generierten Stadt im Vordergrund wurden gerendert.

Produkt Website (RE-O 3, Tran)

Die Website wurde zu einer zum Kauf/Download anregenden Produkt Website umgestaltet, auf der nun ein Startup Guide, sowie mit dem Add-on gerenderte Bilder und Videos zu sehen sind.

1.3 Ergebnisse

Das Ergebnis der Diplomarbeit ist ein in sich vollständiges, funktions- und installationsfähiges Add-on für das 3D-Programm Blender welches ab der Version 2.9 kompatibel ist.

2 Assets

2.1 High-Poly Asset Pack

2.1.1 Gebäudestil

Ein Hauptbestandteil der Diplomarbeit ist das realistische High-Poly Asset Pack, welches dem Benutzer eine Grundlage für den Stadtaufbau bietet. Aufgrund der großen Varianz an Gebäudearten, konzentriert sich CitySketch hauptsächlich auf den Baustil einer klassischen Großstadt des 21. Jahrhunderts.

2.1.2 Modellierung

2.1.2.1 Allgemein

Ein 3D-Objekt in Blender lässt sich in drei Bestandteile aufteilen:

- Punkte (Vertices)

  Ein Vertex Ist der kleinste selektierbare Bestandteil eines Objektes.

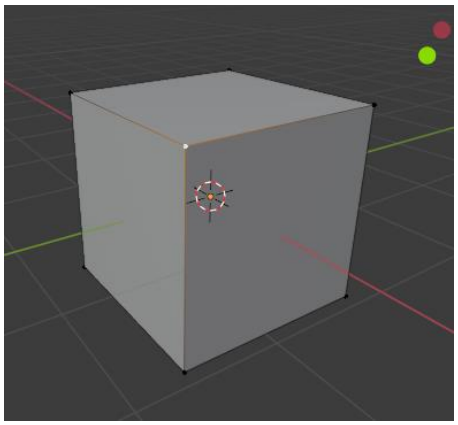


Abbildung 1: Vertex

❖ Linien (Edges)

Linien sind zwei verbundene Punkte.

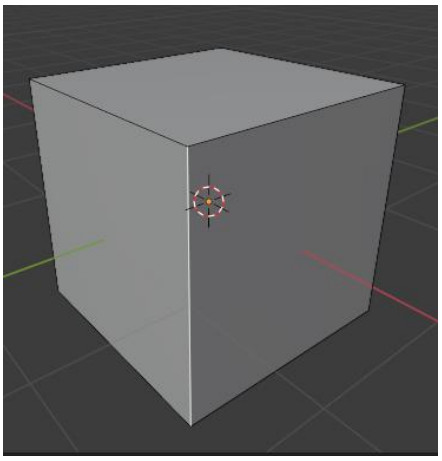


Abbildung 2: Edge

❖ Flächen (Faces)

Flächen entstehen aus der Vernetzung mehrerer Linien. Die kleineste Fläche ist ein Dreieck bestehend aus drei Punkten. Modelliert wird allerdings mit Vierecken, da viele Werkzeuge darauf ausgelegt sind.

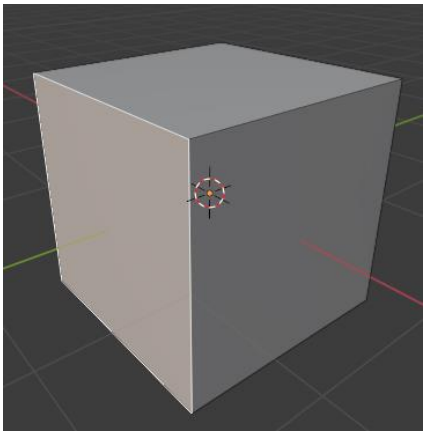


Abbildung 3: Face

2.1.2.2 Modi

Für das Bearbeiten eines Objektes, gibt es in Blender verschiedene Modi. Jeder Modus hat eine eigene Arbeitsweise, vom Skulpturieren bis zum Bemalen. In CitySketch sind allerdings nur zwei Modi relevant, der „Object Mode“ und „Edit Mode“.

Der „Object Mode“ ist der Standardmodus in Blender, indem es möglich ist, Primitiven, wie Würfeln, Kugeln oder auch Lichtquellen hinzuzufügen. Die erstellten Objekte kann man beliebig verschieben, skalieren oder rotieren. Der hauptsächliche Zweck des „Object Mode“ ist es, die Komposition der Szene zu gestalten.

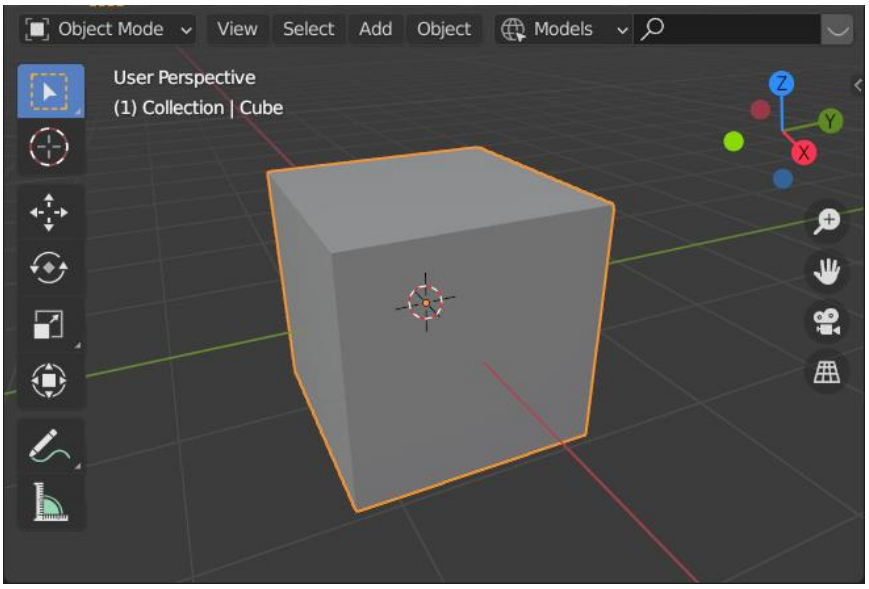


Abbildung 4: Blender im "Object Mode"

Im Edit Mode lässt sich die Geometrie genauer manipulieren. In diesem Modus kann man die einzelnen Bestandteile, wie Vertices, Edges oder Faces eines Objektes bearbeiten.

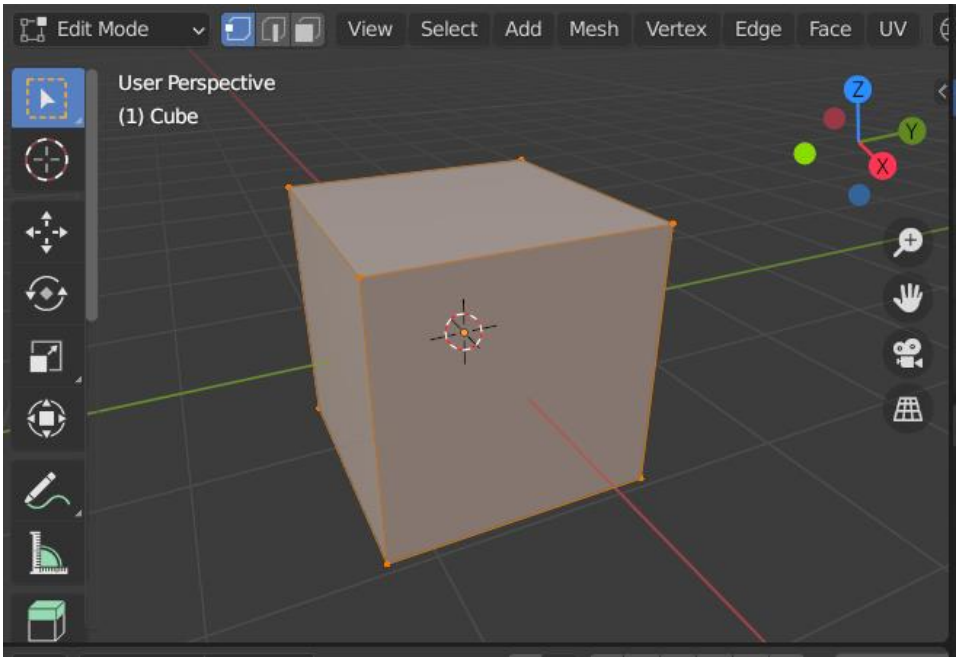


Abbildung 5: Blender im "Edit Mode"

2.1.2.3 Shading Modi

❖ Solid Mode

Der Solid Mode ist die Standardansicht im Viewport. Sie benützt die „Workbench Engine“ von Blender, welche Farben und Licht einfach darstellt.

❖ Wireframe

Im Wireframe Modus kann man nur die Edges und Vertices sehen. Dadurch lassen sich Fehler leichter erkennen und beheben.

❖ Material Preview

Die Material Preview verwendet den „Eevee“ Render, eine sogenannte „real-time“ Render Engine, die als Vorschau Ansicht verwendet wird.

❖ Rendered View

Ähnelt der Material Preview, allerdings hat man in dieser Render Ansicht die Wahl zwischen den zwei Hauptrenderengines in Blender (Cycles/Eevee). Der hauptsächliche Unterschied zwischen den beiden Render Engine sind ihre Performance und Qualität.

2.1.2.4 Basic Mesh Modelling

Bei dieser Methode wird zuerst mit Primitiven begonnen, um die generelle Form und Richtung des Gebäudes zu skizzieren. Darüber hinaus findet die Platzierung von Fenstern und Türen statt. Diese werden mit Hilfe des Boolean Operators in das Objekt hineingeschnitten, um die Details zu platzieren. Andernfalls könnte man die Löcher auch mit Loop-Cuts machen. Gegenüber den Boolean erzielen Loop-Cuts saubere Resultate, die jedoch unnötige bzw. ungewollte Loop-Cuts erstellen. Die Details, wie z.B Fenster, werden ebenfalls aus einfachen Formen modelliert, die allerdings keine weiteren Operatoren brauchen. Anschließend werden alle Teile zusammengefügt und zu einem Objekt vereint.

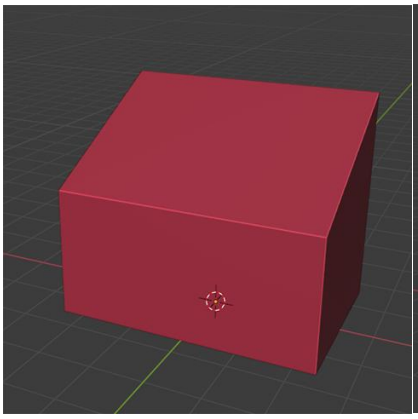


Abbildung 6: Mock-Up vom Gebäude

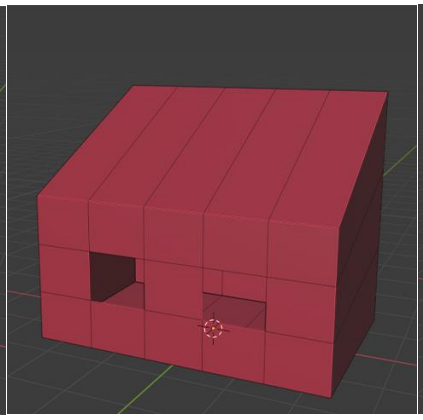


Abbildung 7: Loop-Cut Einschnitte

Abbildung 8: Gebäude mit Fenster und Tür

2.1.2.6 Image Plane Modelling

Für das Image Plane Modelling wird, wie es im Namen schon steht, eine „Image Plane“ gebraucht, welche nur eine einfache Textur ist. In unserem Fall benötigen wir ein Foto von einem echten Gebäude.

Um die Fotos in Blender zu importieren, kann man das in Blender bereits installierte Add-on “Import Image as Plane” verwenden, welches automatisch eine Plane mit der gewünschten Textur erstellt. Anderseits kann man auch im Material Property Tab manuell einem Objekt eine Textur hinzufügen, das erfordert jedoch mehrere Schritte und kostet viel Zeit.

Die texturierte Fläche muss im Edit Mode mit Loop-Cuts bearbeitet werden. Dabei versucht man die markantesten Stellen (z.B Fenster/Türen/Säulen) der Textur mit einem Loop-Cut zu markieren. Diese Bereiche werden mit dem „Extrude“ hervorgehoben, welche dem Bild ihre dreidimensionale Tiefe verleihen.

Abbildung 7: "Brick Building" Textur

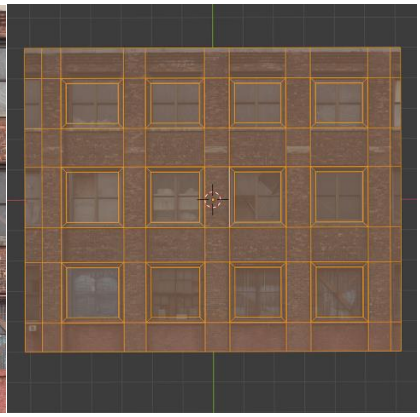


Abbildung 8: Plane nach "Loop-Cuts"

Abbildung 6: gerendertes Gebäude

Damit das Bild im Viewport zu sehen ist, sollte die Render Ansicht auf Rendered oder Material Preview gewechselt werden. Beide Modi ermöglichen es das Gebäude mit dessen Textur vorzuschauen. Allerdings werden dabei auch die Wege vom Licht berechnet, welche die Viewport Performance deutlich beeinträchtigen. Alternativ kann man in den Viewport Shading Einstellung die Farbe im Solid Mode, auf die der Textur wechseln.

Am Anfang der Methode unterteilt man das Gebäude in drei Sektoren, die Fassaden, Dächer und Erdgeschosse. Jeder Sektor ist Teil einer gemeinsamen Textur, die nur in Fällen, in denen man ein gezieltes Aussehen verfolgt, unterschiedlich sein kann. Oftmals beginnt man mit der Fassade, da diese hauptsächlich den Stil des Gebäudes widerspiegelt.

2.1.2.7 Basic Mesh Modelling & Image Texture Modelling

Eine Mischung aus beiden Varianten mit deren Vorteilen. Es wird mit der Basic Mesh Modelling Methode begonnen, um das Aussehen der Gebäude ungefähr zu formen. Etliche Details kommen erst beim Hinzufügen der Texturen hinzu. Jedoch werden die Bilder nicht als eine Image Plane importiert, sondern mit "Cube Projection", einer UV Mapping Methode auf das Objekt entpackt. Anschließend kann man je nach Belieben die Textur im UV Editor Fenster verschieben.

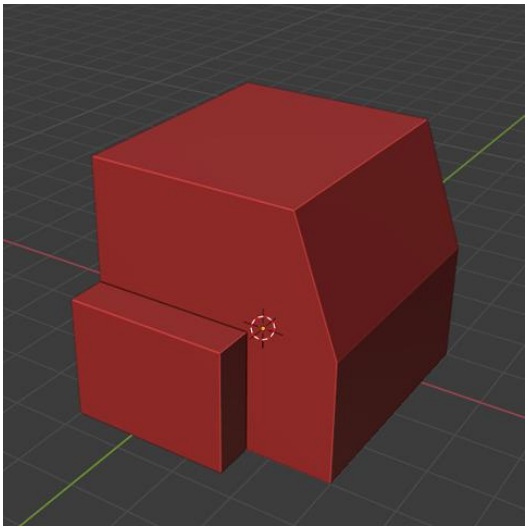


Abbildung 12: Gebäude Skizzieren

Abbildung 13: Gebäude mit Textur

2.1.3 UV Mapping

2.1.3.1 Einleitung

Mit UV Mapping ist es in Blender möglich Texturen auf eine Mesh zu projizieren bzw. anzupassen. Wie schon vorhin in der Modelling Einleitung erwähnt, besitzt jedes Gebäude Flächen, Linien und Punkte und eine UV-Map besitzt ebenfalls diese Eigenschaften, allerdings nur im 2D-Bereich. Anstatt der X-/Y- Achse wird bei „UV-Maps" mit der U bzw. V Achse gearbeitet, daher auch der Name „UV". Man könnte diesen Prozess mit einer Milchpackung vergleichen. Es ein normales 3D Objekt, jedoch wenn man es ausfaltet, ist es nur ein flaches Stück Karton. Das gleiche Prinzip gilt auch in Blender oder in jeder anderen 3D Software.

Um einer Mesh eine gewisse Textur zuzuweisen muss man sie erstmal in 2D in ihre Bestandteile aufteilen.

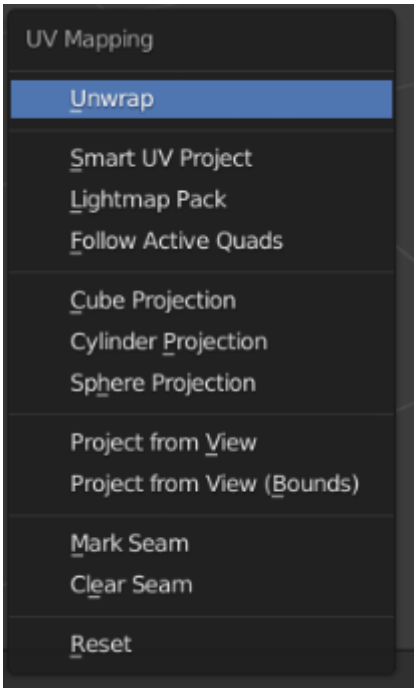


Abbildung 9: Unwrap Methoden in Blender

2.1.3.2 Standard Unwrap

Diese Methode ist einer der simpleren Wege, um eine UV-Map zu erstellen. Sie schaut wie die einzelnen Punkte miteinander verbunden sind und stellt dementsprechend die Faces der UV her. Außerdem versucht sie für jede Face eine freie Fläche zu finden.

Ein weiteres Kriterium, auf welches das Standard Unwrap schaut sind die sogenannten „Seams“ oder auch auf Deutsch „Nähte“. Diese „Nähte“ kann man mit denen eines Stofftieres vergleichen. Wie auch beim Stofftier verbindet das „Seam“ zwei Flächen miteinander. Allerdings ist es in Blender ein Indikator zweier UV-Faces.

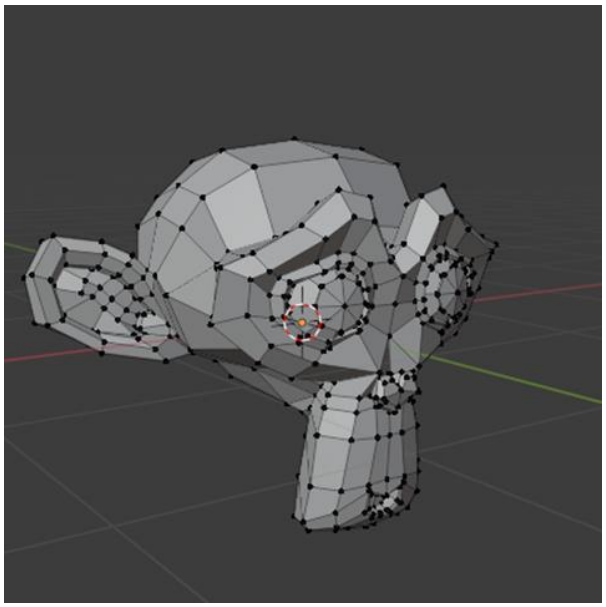


Abbildung 15: "Suzanne"

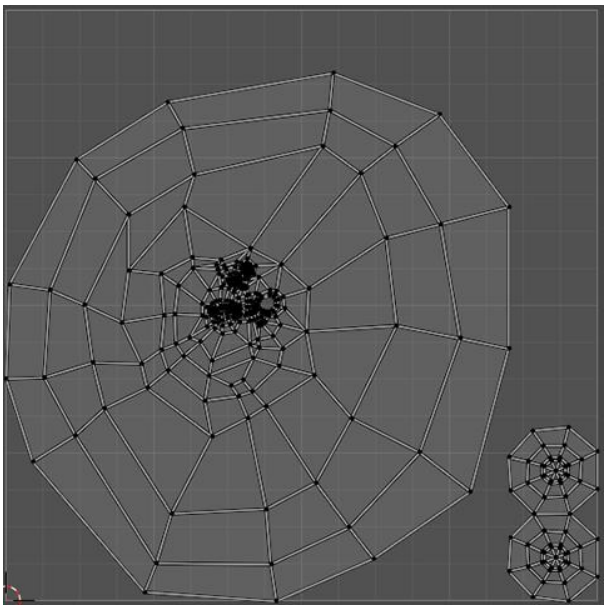


Abbildung 16: UV-Map mit der Standard Unwrap Methode

2.1.3.3 Smart UV Projekt

Im Gegensatz zum Standard Unwrap, rechnet sich die Smart UV Project Methode aus den Verbindungen zwischen den Faces, automatisch die nötigen „Seams“ aus. Vor Allem für CitySketch Gebäude wurde diese Methode verwendet, da Smart UV Project primitivere Formen, leichter bearbeiten kann. Allerdings entstehen meistens viele kleinere Flächen, die bei einer Mesh mit mehreren Flächen bzw. „Vertices“ ein Problem darstellen könnten. Die Smart UV Project Methode sucht für jede „Face“ einen freien Platz auf dem Bild. Da es nicht möglich ist, in einem begrenzten Bereich unendlich Raum zu schaffen, müssen die Flächen für die Map proportional kleiner sein. Ansonsten würde Blender beim Kalkulieren der „Faces“ überfordert werden und abstürzen.

Abbildung 11: "Korean Building"

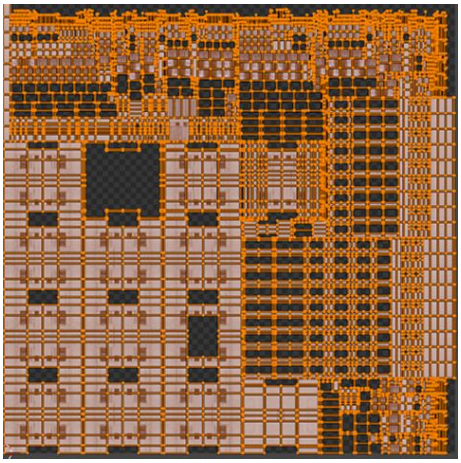


Abbildung 10: Smart UV Map vom "Koran Building"

2.1.3.4 Cube Projection

Cube Projection nimmt die jeweilige Mesh und projiziert sie auf die sechs Seiten eines Würfels, welche darauf aufgefaltet wird. Die daraus entstehenden „Faces“ überlappen sich können aber per Hand nach Belieben verschoben werden. Diese Methode wurde in diesem Projekt verwendet, um kleinere Bereiche eines Gebäudes zu “entpacken”, damit mit man ohne viel Aufwand eine brauchbare “UV” für das Texturieren hat. Dasselbe Prinzip gilt auch für die ähnlichen Unwrap Methoden „Cylinder Projection“ und „Sphere Projection“.

Diese Methode wird hauptsächlich bei der Mesh + Image Texture Modelling Methode benützt. Sie bietet eine gute Grundlage für das weiter Verarbeiten der Texture Map an.

2.1.3.5 Project from View

Eine der schilchteren UV Unwrap Methoden, sie projiziert die UV-Map in dem Winkel, indem man momentan auf die Mesh schaut. Mit „Project from View“ kann man Fehler in der „UV“ leicht manuell ausbessern.

Das größte Problem bei der Image Plane Modelling Methode, sind verschmierte Texturen beim Extrudieren. Blender kann ihre UV Maps nicht automatisch aktualisieren, darum wird neu erzeugte Geometry nicht richtig interpretiert.

2.1.4 Texturierung

Texturieren gehört, neben dem Modellieren, zu den wichtigsten Schritten, um realistische Objekte zu erstellen. Schließlich bekommt das Gebäude in dieser Phase die Farbe und den Charakter.

Es gibt verschieden Wege um eine Textur zu bekommen. Einerseits kann man sie im Internet herunterladen oder auch selbst machen. In jedem größerem Studio werden hauptsächlich „Hausgemachte“ Texturen benützt. Für diese Diplomarbeit wurden eigene Bilder erstellt, aber auch vom Internet welche benützt.

2.1.4.1 Procedural Materials

Hauptsächlich kann das größte Potenzial der „Nodes“ bei „Procedural“-Materialen ausgeschöpft werden, welche mehr oder weniger Computergeneriete Texturen sind. Jedoch benötigt jedes der handgemachten Materialen für ihr Aufbau oftmals eine von Blender eigenen dynamischen Texturen. Diese haben Vorteil, dass man sie je nach Belieben verändern kann, welche es erlaubt alle verschieden fotorealistischen Texturen „händisch“ nachzumachen. Allerdings benötig man für die Erstellung ein äußerst aufwendiges „Node“-Netzwerk. In CitySketch wurde beispielsweise für das Fluss-Asset eine „Procdural“-Textur angefertigt.

2.1.5 Shader Entwicklung

Die Shader Entwicklung geht Hand in Hand mit dem Texturieren, da Beides aufeinander aufbaut. Texturen braucht man, um dem Objekt Farbe zu verleihen daraufhin braucht man Shaders, damit die Texturen mit dem 3D Umfeld interagieren können.

2.1.5.1 Princepled Shader

Der Princepled Shader umfasst alle anderen Shaders die es in Blender gibt. Damit ist es möglich eine Vielzahl an verschiedenen Materialen zu erstellen. Außerdem wird dieser universell vom verschieden 3D-Softwaren unterstützt.

2.1.5.2 Nodes

Der effizienteste Weg, um in Blender mit Materialen zu arbeiten, ist im „Shader-Editor"-Fenster. Darin wird mit sogenannten „Nodes" gearbeitet. Diese sind auch in anderen 3D-Softwaren, wie zum Beispiel in Cinema-4D oder Maya, üblich. Mit „Nodes" ist es möglich komplexe Shader-Strukturen aufzubauen. Vergleichen könnte man das „Node"-System mit den Logikgattern in der Elektrotechnik. „Nodes" sind in der gleichen logischen Weise wie bei der Logischen Schaltung aufgebaut Es gibt mehreren Eingängen beziehungsweise Ausgänge, die in Kombination mit mehreren „Nodes" verschiedene Ergebnisse liefern können. Dadurch ist es möglich im „Shader-Editor" komplizierte Rechnung, allein durch das Verbinden mehrerer „Nodes" aufzustellen.

2.1.5.3 Arbeiten mit Materialen

Im „Shader"-Editor gibt es diverse Typen von „Nodes", die ihren eigenen Zweck haben. Dabei sind die „Shader"-Nodes der Hauptbestandteil jedes Materials, da ohne sie, keine Texturen es erlaubt mit anderen Objekten oder Lichter zu interagieren. Damit Texturen versetzt oder skaliert werden, sorgen die „Vector" Nodes. Diese nehmen die Position, beziehungsweise der Vektor eines Bildes und manipuliert sie je nach Belieben. Dadurch ist es möglich außerhalb vom UV-Editor Texturen zu bewegen. Ein weiterer essenzieller Teil jedes Materials sind die „Converter". Durch diese ist können Werte von „Node"-Ausgängen vergrößert, verkleinert oder begrenzt werden.

2.1.6 Point of Interest

Für die Erstellung und Modellierung eines „Point of Interest“ Gebäudes benötigt man einen anderen Ansatz. Wie der Name schon sagt, muss das Gebäude so gestaltet werden, dass es sich von den anderen auf Anhieb unterscheidet und sofort auffällt. Außerdem wird dieses durch einen rundum generierten freien Bereich betont.

Aufgrund der komplexeren Form der Kirche, wurde für die Erstellung des Gebäudes die „Basic Mesh Modelling“ Methode verwendet. Der Grundriss ist nötig, damit man sich bei gröberen Fehlern jederzeit ausbessern kann.

2.1.7 Gebäude

Das High-Poly Asset Pack beinhaltet 10 unterschiedliche realistische Gebäude. Jedes Asset hat grundsätzlich dieselben Elemente (Fassaden, Dächer und Erdgeschosse). Dennoch kann man alle Gebäude in Gruppen aufteilen. Die meisten Gebäude sind normale Wohnhäuser, aber auch einige Hochhäuser, sowie eine Schule sind im Pack vertreten.

2.1.7.1 Zusammenstellen der Teile

Das Zusammenfügen von der Gebäudestücken ist wichtig, da man dadurch einige Probleme aus dem Weg gehen kann. Dafür sollte man zuerst alle Modifikatoren anwenden, damit beim Verbinden der Teile nicht unabsichtlich den Effekt zu entfernen, beziehungsweise dem ganzen Objekt zu übertrage. Für das Addon ist es notwendig das Gebäude als ein einziges Objekt zu übergeben, da das Positionieren eines Assets aus mehreren Teilen, äußert aufwendig ist. Der Grund dafür sind die verschiedenen Maße, die den Ankerpunkt jedes einzelnes Objekts beeinflussen. Dadurch könnte man zwei verschiedene Teile durch denselben Wert vergrößern und man würde zwei unterschiedliche Ergebnisse bekommen.

Eine Alternative zum herkömmlichen Zusammenstellen mehrerer Teile sind die in Blender 2.8 neu erschienenen „Instance Collections“, welche es erlaubt eine Gruppe aus mehreren Objekten zusammenzufassen, so dass man sie nur noch als ein Einziges sieht. Im Viewport kann man die Kollektion nur noch Skalieren und Positionieren. Allerdings ist es Möglich mit „Instance Collection“ jeder Zeit einzelne Teile des Gebäudes zu variieren, ohne da man ein Back-Up braucht.

2.1.7.2 Modellierung

Für das Modellieren der Gebäude wurde hauptsächlich die Image Plane Modelling Methode verwendet. In der aus einem Foto ein 3D-Modell gemacht wird. Mit dieser Technik ist es möglich ohne viel Texturen-/Shader-Arbeit ein realistisches Gebäude zu modellieren.

2.1.8 Straßen

Im Straßen Asset Pack gibt es vier Straßenstücke, Kreuzungen, T-Kreuzungen, Links-/Rechtkurven und normale gerade Straßen.

2.1.8.1 Modellierung

Bei der Modellierung der Straßen wurde die Mesh Modelling Methode verwendet. Damit alle Teile, wie Puzzlestücke, übergangslos aneinandergereiht werden können, müssen die Gehsteigsegmente, sowie die Straßenendungen die gleiche Länge haben. Dafür werden alle Stücke in ein 6x6 Grid-System unterteilt (1/3 Gehsteig, 2/3 Straße).

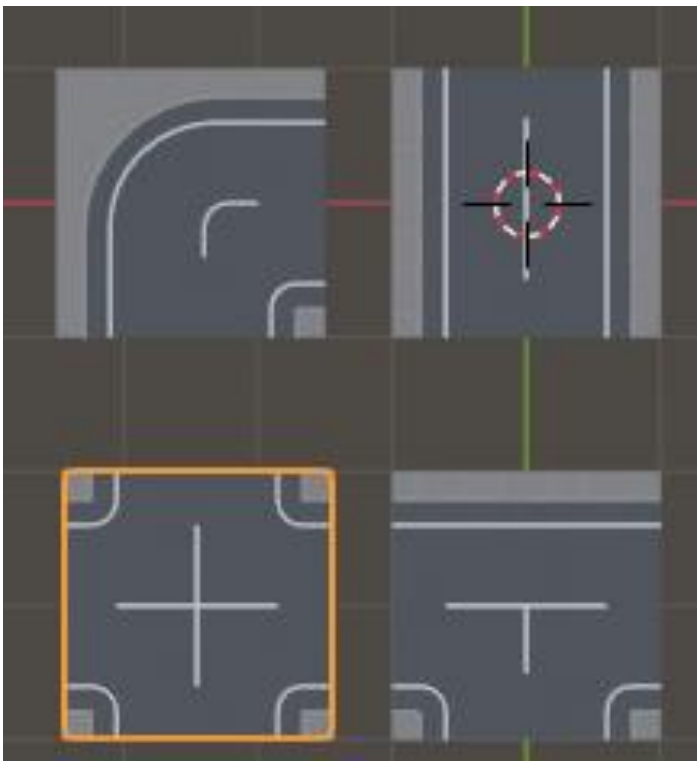


Abbildung 12: X-/I-/L-/T-Straßen Asset

2.2 Low-Poly Asset Pack

2.2.1 Modellierung

Beim Erstellen der Modelle wurde nicht nur auf die Einfachheit geachtet, sondern auch auf das Konzept eines realistischen Gebäudes und dessen Form. Das Ziel der Modellierung ist es mithilfe eines einzigen Würfels ein Bauwerk zu modellieren, so dass die Erscheinung noch dem eines Gebäudes entspricht. Hierbei reflektieren wir jedoch die Fassaden der Realität, da im Low Poly Stil auf Details und Feinheiten verzichtet werden, da diese vom Entwurf weg neigen würden.

Die Grundvoraussetzungen, die einem Haus entsprechen (Fenster, Dach und Tür) sollen bleiben. Die große Schwierigkeit bei der Modellierung des Low-Poly Packs ist es gleiche Merkmale zwischen den verschiedenen Gebäuden zu entwickeln. Das Asset Pack beinhaltet unterschiedliche Modelle, die von der Größe und dem minimalistischen Design differenzieren.

Ziel des Assets ist es die Augen des Betrachters auf das ganze Objekt zu richten. Farben sollen keine große Rolle spielen. Das Asset Pack stellt die Gebäude in einem klaren und sauberen Weiß zur Verfügung. Weiß wirkt schöner und moderner fürs Auge und lenkt nicht auf die Details.

Dadurch, dass Feinheiten vermieden werden sollen, wurde bei der Modellierung in Blender mit Tools wie Loopcuts gearbeitet. Loopcuts sind ein gutes Werkzeug um Objekte zu „schneiden“. Das Objekt wird dann in mehrere, gleich große Flächen geschnitten damit beim Modellieren keine ungleichen Ebenen entstehen.

2.2.1.1 Die Erstellung eines Gebäudes

Bevor man mit einem Low-Poly Gebäude beginnt, muss man sich vergewissern, wie es am Ende ausschauen soll. Der Stil ist recht vielfältig und es gibt viele Methoden Objekte in solch einer Art zu erstellen. Ein positiver Aspekt ist die Kreativität, Gebäude müssen nicht der Realität bzw. einer echten Stadt entsprechen.

In Blender bekommt man standardweise einen Würfel, wenn man das Programm öffnet. Die meisten Gebäude des Asset Packs wurden nur mithilfe des Würfels überarbeitet.

Jedes Gebäude wurde auf 1x2 oder 1x1 hochskaliert, damit es zu keinen Ungenauigkeiten kommt (wenn die Stadt generiert wird und zwei Gebäude ineinander platziert werden, die Höhe ist in diesem Zusammenhang unwichtig).

2.2.2 Formen

Einer der größten Probleme, die wir beim Planen von den Gebäuden hatten, ist, welche Form die Gebäude einnehmen sollen. Wir wollen keine langweiligen, sich ähnelnden Gebäuden modellieren, da die Stadt dann sehr einseitig wirkt.

2.2.3 Point of Interest

Ein Stadtzentrum mit einer großen Fläche soll auch im Low-Poly Pack existieren. Wie auch beim High Pack haben wir uns dazu entschieden, eine Kirche als POI zu erstellen. Die Kirche soll nicht die einzige Möglichkeit als Point of Interest sein, hierbei geben wir in CitySketch die Möglichkeit neben die eigenen Gebäude auch ein eigenes POI in das Add-on zu importieren. Die Inspiration eine Kirche als Hauptgebäude zu nehmen war aufgrund vom Stephansplatz mit dem Stephansdom in Österreich, Wien. Die Kirche war groß und einzigartig und wurde wegen der offenen Fläche sehr gut als POI repräsentiert. Daher wird auch in CitySketch rund um den Point of Interest eine offene Fläche generiert (Keine Straßenlinien).

Abbildung 13: Point of Interest des Low-Poly Asset Packs: Kirche

2.2.4 Gebäude

Das Asset Pack beinhaltet 10 verschiedene und selbsterstellte 3D-Gebäude, die beim Herunterladen von CitySketch mitinkludiert sind. Ein großes Problem war, das Konzept der Gebäude. Sollen es realistische Häuser in Form von Low Poly sein, oder können wir auch etwas Einzigartiges machen was man nicht allzu Tage sieht? Wir sind dann zu der Lösung gekommen, eine Mischung aus beiden Aspekten zu machen, wobei einige der Realität entsprechen und manche eine Eigenkreation sind. (Alle Modelle wurden vom mir erstellt und es wurden keine anderen Add-ons oder Tools außerhalb von Blender verwendet).

Diese Gebäude sind im Asset Pack enthalten:

Three Corner ist ein eigenes Konzept und ist ein Wohngebäude, dieses Gebäude hat bewusst keine Tür. Der Kernpunkt dieses Models ist es sich in der Stadtmenge einzubauen und von oben auf die Dächer hinunterschauen zu können. Das Gebäude ist von Fenster umrundet, so dass keine Seite leer ausschaut.

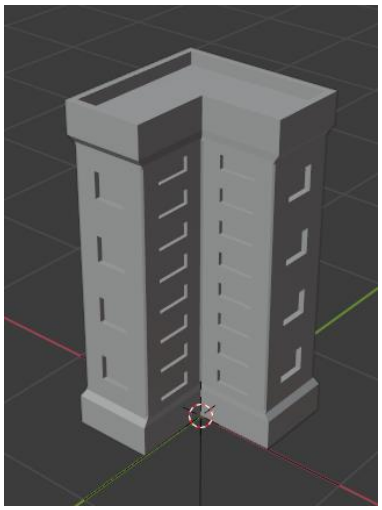


Abbildung 14: Three Corner

Der **New Yorker Office**, wie bereits im Namen erläutert, wurde von den Business Hochhäusern in der Stadt New York City inspiriert. Das Gebäude stellt ein Bürohaus dar, welches bis zu 50. Stockwerke hoch ist. Dieses Gebäude passt sehr gut in eine dichtere Stadt und soll auch das Prinzip des Low-Polys erfüllen: Einen Überblick über die gesamte Stadt.

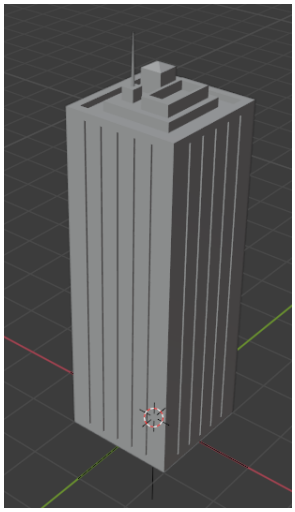


Abbildung 15: New Yorker Office

Das **Family Flathouse** ist ein eigenes Konzept und gehört zu den Wohnhäusern. Anders als bei den anderen Gebäuden, hat dieses im Erdgeschoss größere Fenster als bei den darüber folgenden Geschossen. Das große Fenster soll ein Geschäft repräsentieren was sich

unter einem Wohnhaus befindet. Die Geschäftsidee wäre zum Beispiel ein Restaurant oder ein Kaffeehaus.

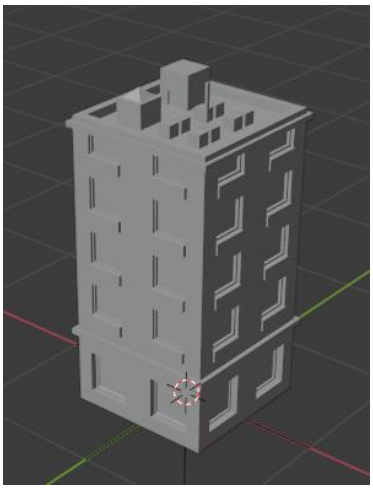


Abbildung 18: Family Flat-house

Main Centre Tower ist eine Inspiration vom Brandenburger Tor in Deutschland, Berlin. Der 1x2 Baustil sorgt für mehr Spannung und Einzigartigkeit in der Stadt. Dieses Haus dient als ein Bürohaus im Bereich der Informationstechnologie und repräsentiert das Moderne Design und die Zukunft der Technik.

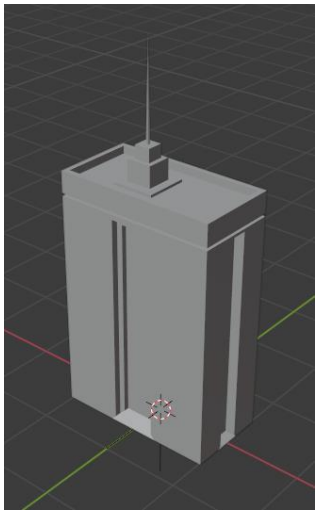


Abbildung 16: Main Centre Tower

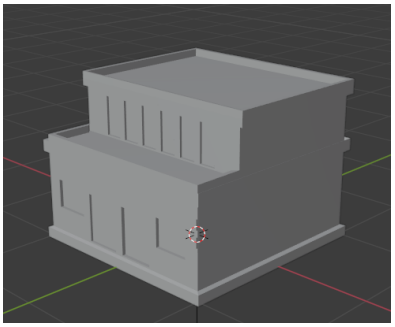


Abbildung 17: Summerhouse

Das Summerhouse ist ein Gebäude inspiriert von Videospielen im Bereich Open World. Diese Art von Spielen beihalten meisten sehr viele saisonale Gebäude von Winter bis hin zu Sommer beziehungsweise Strandhäusern. Daher sollte in CitySketch sowas nicht fehlen!

Die Idee eine Stadtbücherei zu modellieren kommt von der Wiener Nationalbibliothek und der New Yorker Public Library. Auch im Low Poly Pack soll es zahlreiche Orte wie **The Library** zum Besuchen geben.

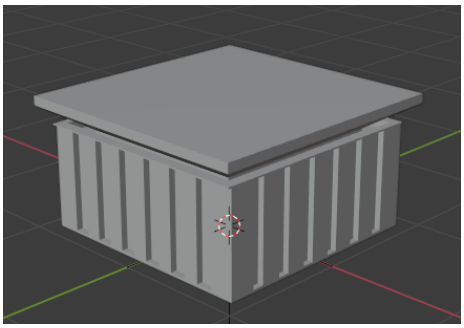


Abbildung 19: The Library

The Office ist ein Businesshaus einer Telekomfirma. Ein modernes Gebäude mit vielen Fenstern und einem großen Dach mit vier kleinen Türmen auf jeder Ecke.

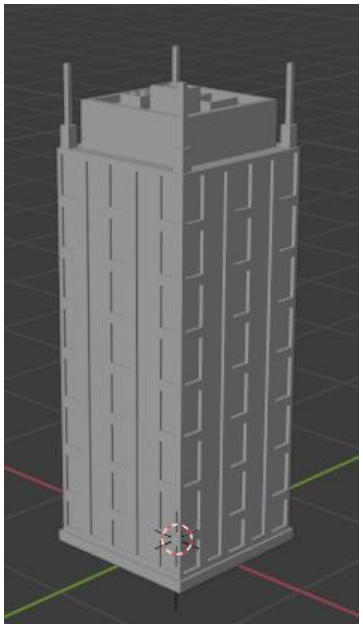


Abbildung 20: The Office

Das Pillarhouse ist ein eigenes Konzept und gehört zu den Familienwohnungen. Es besteht aus 5 Stockwerken und 2 Säulen für einen interessanten Stil.

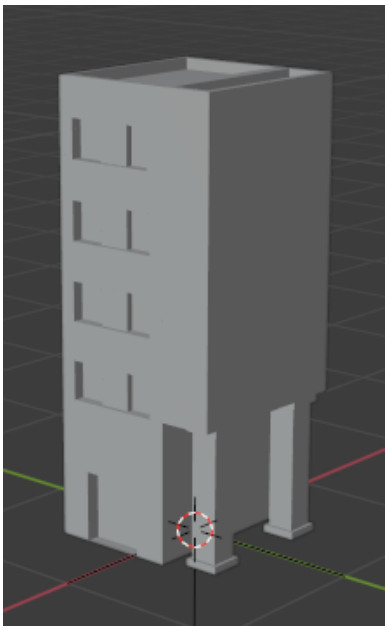


Abbildung 22: Pillarhouse

Old Rowhouse repräsentiert einen älteren Baustil von vor 50 Jahren. Es unterscheidet sich sehr von den moderneren Häusern des Low Poly Packs.

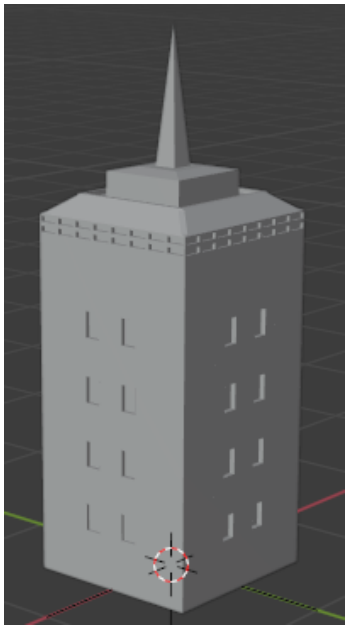


Abbildung 21: Old Rowhouse

Der Watertank ist die einzige Form, die von den realistischen Konzepten der Häuser heraussticht. Das Gebäude ist auf Grund weniger Details trotz dessen einen Blick durch den besonderen Stil wert. Watertank ist auch ein gutes Prinzip, um die Stadt auf einen gesamten Blick betrachten zu können.

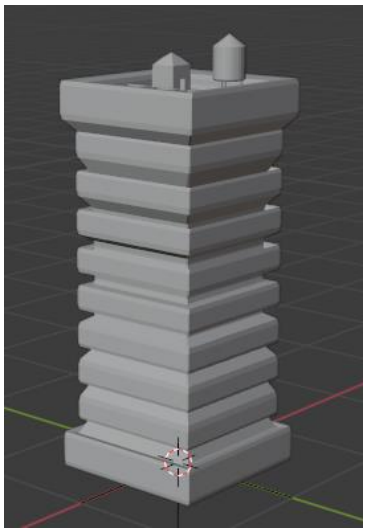


Abbildung 23: Watertank

2.2.5 Straßen

Bei der Erstellung der Straßen haben wir uns an die Buchstaben X-T-L-I orientiert. Das bedeutet, dass die Straßenrichtungen den jeweiligen Zeichen entlang gehen. Die Straßen sollen sowie die Low-Poly Gebäude keine detaillierten Feinheiten wie Straßenrisse, oder Pflastersteine besitzen, da diese nicht zum Konzept vom Asset-Pack passen.

Für die Erstellung der Straßen wurde vom Mesh eine Plane erstellt. Die Plane wurde im Edit Mode mithilfe von Loop Cuts in drei Teile geschnitten. Im Object Mode wird dann mithilfe der Face Selection die linke und die rechte Face ausgewählt und wieder im Edit Mode mithilfe vom Shortcut (E) für Extrude, hochskaliert. Es bildet sich dann eine Rasterform, diese Methode nutzt man für alle Straßen und skaliert nur die Bereiche, für die man braucht. (Bsp: Für eine L-Straße werden x Flächen hochskaliert [siehe Abildung]).

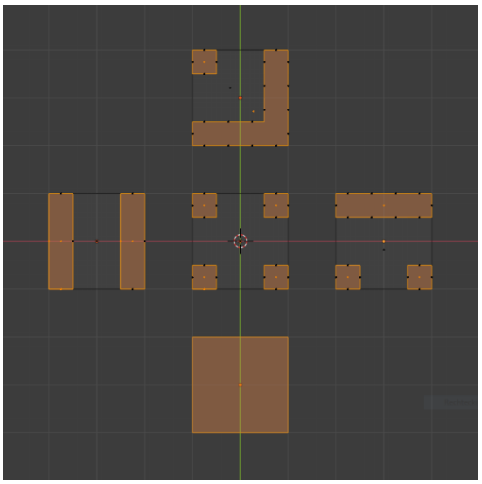


Abbildung 24: Straßenmodelle

2.3 Grünzonen Assets

2.3.1 Gras

2.3.1.1 Plane Modelling

Die Form eines Grashalms ist eine einfache „Plane“ mit einer extrudierten Spitze. Mithilfe des „Array“ Modifikator kann man eine Mesh mehrmals duplizieren und in Kombination „Randomize Transform“, einer Funktion um mehre Objekte zufällig zu verschieben und rotieren. Dadurch werden kleinere Grasfelder gemacht.

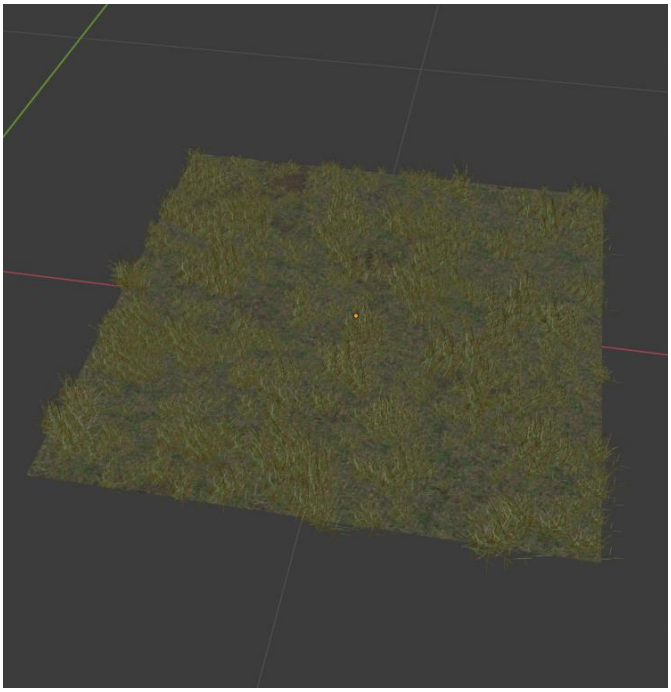


Abbildung 25:Grassfeld

2.3.1.2 Partikel System

Das Partikel System in Blender ist eine weitere Möglichkeit, eine Mesh zu kopieren. Jedoch, im Vergleich mit dem „Array“ Modifikator besitzt es weit aus mehreren Optionen, um das Generierverhalten zu beeinflussen.

2.3.2 Bäume

Das Baum Asset wurde mit dem „Sapling Tree Gen“ Add-on modelliert. Das Aussehen der Bäume kann man mit dem Add-on durch Regler bestimmen. Es gibt allerlei Einstellungen, um die Anzahl der Äste oder die Dicke des Stammes zu verstellen.

Abbildung 26: Tree Asset

2.3.3 Büsche

2.3.3.1 Blätter

Für die Blätter werden PNGs verwendet, die in Blender mit dem „Image as Plane“ Addon importiert wird. Allerdings benötigt die Plane für ein detaillierteres Aussehen, mehr Geometrie. Dafür werden im Edit Mode drei bis vier Loop-Cuts, jeweils in der Höhe und der Breite hinzugefügt. Dabei ist es wichtig die Anzahl der Loop-Cuts so gering wie möglich zu halten, da die Performance davon stark beeinträchtigt wird.

Für Unregelmäßigkeiten in den Blättern sorgt der „Displace“ Modifikator. Dieser verschiebt die Vertices der Objekte, anhand einer beliebigen Textur. Grundsätzlich ist der „Displace“-Effekt stärker je heller das Bild ist. Normalerweise benützt man dafür eine sogenannte „Displacement Map“. Im Gegensatz zu herkömmlichen Bildern, besitzt eine „Displacement Map“ Non-Color-Data, welche zum Beispiel Vektor Informationen speichert. Ein Beispiel für eine „Displacement Map“ wäre die „Clouds“-Textur, die bei der Modellierung der Blätter verwendet wurde.

Damit nicht alle Blätter den gleichen „Displace“-Effekt haben, kann man im Modifikator die Koordinaten der Textur versetzten. Dabei gibt es die Möglichkeit sie entweder manuell zu steuern oder in den Koordinaten Einstellungen auf „Global“ zu setzten, welche die Textur anhand der Position vom „Displace“-Objekt verschiebt.

2.3.3.2 Partikel System

Die modellierten Blätter werden mittels eines Partikel System dupliziert. Jedes Partikel System braucht ein Objekt als Emitter. Dieser wird benötigt, um den Partikeln einen Punkt zum Starten zu geben.

Der Emitter für den Busch ist eine in die Länge gezogene Sphäre, worauf ein „Displace“-Modifikator angewendet wurde. Für den „Displace“ wurde die gleiche „Cloud“-Textur wie bei den Blättern benützt. Dadurch sieht die Sphäre unebener aus und ähnelt bereits in diesem Stadium einem Busch.

Danach fügt man dem Partikel System, die Blätter hinzu. Am Anfang sieht der Busch aus wie ein komischer Igel, da alle Blätter normal zum Emitter, in die gleiche Richtung schauen. Um das zu vermeiden sollte man in den „Scale Random“ Einstellungen des Partikel Systems, den Wert erhöhen. Das Gleich sollte ebenfalls in den Rotier-Einstellungen gemacht werden.

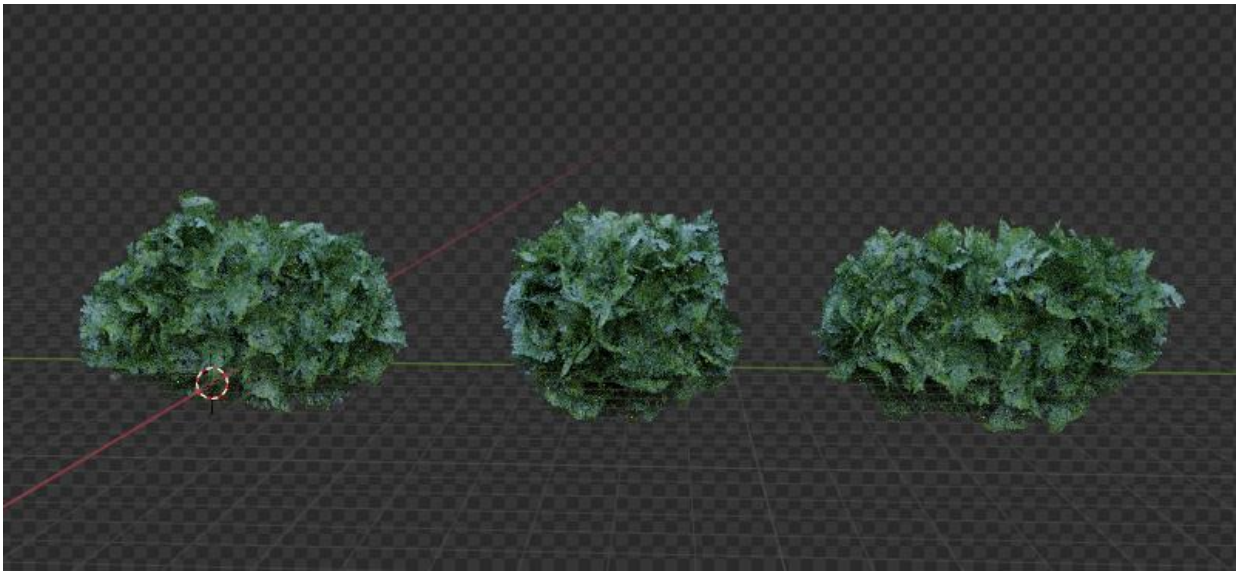


Abbildung 27: Busch Assets

2.4 Animierte Assets

2.4.1 Extrahierung der Pfade

Eine originelle Funktionalität ist die Möglichkeit der Pfad-Extrahierung. Ein generiertes Straßennetz wird, unabhängig von dessen Komplexität, in einen Pfad mit der durch den Benutzer voreingestellten Länge umgewandelt. Die besondere Eigenschaft dieses Pfades ist die realitätsnahe Abbildung eines möglichen Straßenverkehres. Der Pfad verfolgt das Straßennetz unter Berücksichtigung der grundlegenden Verkehrsregeln. Der Pfad folgt aus seiner Sicht ausschließlich der rechten Straßenseite, Verhält sich auf Kreuzungen korrekt und versucht eine möglichst große unerforschte Fläche abzudecken – ganz ohne Rekursion und abrupten Knicken.

Dies wird erreicht, indem ein fiktives Auto (im Code), ausgehend von einem zufälligen Straßenstück, durch die Stadt fährt. Durch das Evaluieren der möglichen Routen anhand einer Fitness Funktion entscheidet sich das Auto rekursionslos für die beste lokale Route. Die beste globale Route zu kalkulieren ist ohne Rekursion nicht möglich, da dazu alle Routen geprüft werden müssen. Die Fitness Funktion bewertet eine Route anhand der Besuchshäufigkeit und dem nötigen Aufwand, um zu Ihr zu gelangen. Eine Route, die erfordert, dass das Auto um 180 Grad umdreht, besitzt einen sehr hohen Aufwand im Vergleich zu einer Route, die geradeaus verläuft. Letztere wird in diesem Fall bevorzugt.

Während das fiktive Auto fährt, werden sogenannte Marker (Punkte) auf der Route platziert, die im Anschluss verwendet werden, um eine Non-uniform rational B-spline (NURBS) Kurve zu erzeugen. Diese Form der Kurve eignet sich aufgrund der Interpolation zwischen ihren Punkten optimal für unseren Anwendungsfall, denn es ist möglich das Verhältnis dieser Interpolation pro Punkt zu steuern. Dies ist entscheidend für die weiche Bewegung beim Abbiegen und an den Kreuzungen.

Der Pfad kann im Anschluss verwendet werden, um einen Straßenverkehr zu simulieren. Nachdem ein Auto-Modell die Blender Funktion „Follow Path“ auf den extrahierten Pfad anwendet, beginnt dieses dem Pfad entlangzufahren. Wiederholt man diesen Prozess mit mehreren Autos und lässt diese an verschiedenen Stellen starten, erhält man eine überzeugende Verkehrssimulation.

2.4.2 Path Animationen

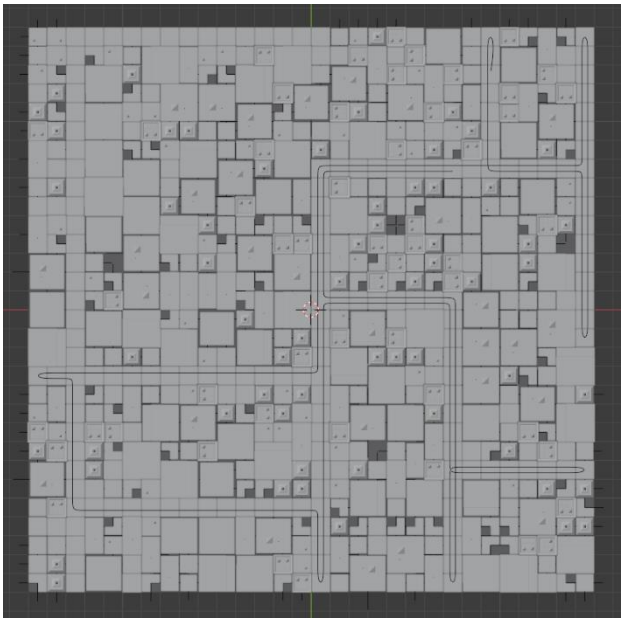


Abbildung 28: Generierter Pfad

Path Animation ist eine Erweiterte Einstellung für die Generierung von Linien, die auf den Straßen abgebildet werden. Diese Linien werden von Autos entlang der ganzen Stadt befahren. Dasselbe Prinzip gilt auch für den Helikopter, der sich mithilfe des Pfades um die Stadt herumbewegt. Es kann eingestellt werden, ob nur die Straßenpfade oder auch ein Kreispfad erstellt wird. Ein Kreispfad dient als eine rotierende Linie für mögliche Fliegermodelle. (Beispielsweise im unserem Add-on ein Helikopter.)

Generierung des Pfades

Um in CitySketch einen Pfad zu generieren, muss man in die "Advanced Generator Settings" gehen. Hier befinden sich die Möglichkeiten "Extract Path" oder "Extract Circle zu betätigen. Insbesondere kann die Größe beziehungsweise die Länge der Pfade eingegeben werden. Für die Pfade bedeutet dies, wie viele Linien in der Stadt generiert werden sollte sprich wie viele Runden oder Wege ein Autofahren kann. Für Extract Circle kann man die Höhe einstellen. Hiermit kann der Benutzer entscheiden, wie hoch der Helikopter über die Stadt fliegen soll.

Objekt an dem Pfad anlegen

Um das gewollte Objekt auf den Pfad anzulegen, müssen wir diesem einen sogenannten Constraint geben. Mit Constraints können wir die Bewegung eines Objektes kontrollieren. Für unseren Zweck benutzen wir dies um in dem Fall ein Auto und einen Helikopter entlang unseres erstellten Pfades entlang fliegen beziehungsweise entlangfahren zu lassen. Beim

Auswählen des Constraints nehmen wir die Methode “Follow Path”. Nachdem ein Constraint erstellt wurde können wir uns einen Pfad aussuchen. Hierbei geht man auf “Target:” und gibt einer der generierten Linien an. Als nächsten Schritt muss auf “Animate Path” gedrückt werden und die Checkbox “Follow Curve” aktiviert werden. Nun sollten Autos und Helikopter in der Stadt auf die entsprechenden Pfade fahren und fliegen.

Wie schnell sind die Objekte?

Die Geschwindigkeit kann vom Nutzer geändert werden. Im Bereich “Object Data Properties”, unter “Path Animation” ist eine Checkbox die aktiviert werden muss. Bei “Frames” kann eingestellt werden, wie schnell die Modelle sich bewegen sollen. Je höher die Framesrate, umso langsamer bewegen sie sich. Ein Problem ist es, dass wenn bei einem Objekt eine niedrigere Framerate eingestellt wurde, dann wird dieser zum Ursprungsort platziert und macht womöglich keine vollen Runden. (Beispiel: Der Helikopter hat 2000 Frames und die Autos haben 5000, beim Abspielen der Animation wird bei Frame Nummer 2001 der Helikopter zum Ursprünglichen Ort zurückgesetzt wo er bei Frame Nummer 1 war).

2.4.3 Die Assets

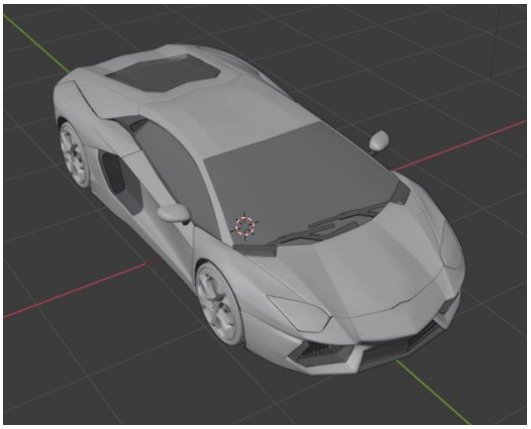


Abbildung 30: Lamborghini Aventador von BlenderKit

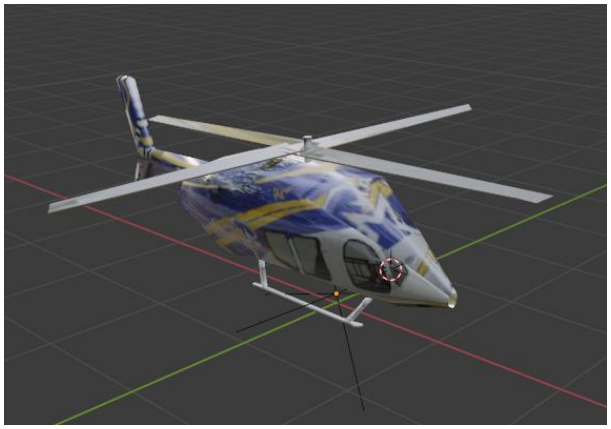


Abbildung 29: Helikopter von CitySketch

In CitySketch gibt es zwei verschiedene Assets, die im Animierten Asset Ordner beinhaltet sind. Eines davon ist ein selbstmodellierter Helikopter. Der Propeller des Helikopters dreht sich bei der Animationsführung damit das Fliegen realistischer wirkt. Grundsätzlich war geplant, die Assets vom Blender Add-on “Blenderkit” zu nutzen. Blenderkit ist eine Blendererweiterung die viele fertiggestellte Modelle lizenzfrei zur Verfügung stellt, darunter befinden sich auch Transportmodelle wie Autos und Helikoptern. Das Auto, welches sich genauso im Ornder befindet, wurde vom BlenderKit übernommen und ist das sogenannte “Lamborghini Aventador” Modell. Kleine Änderungen wurden jedoch für das Auto gemacht,

denn genauso wie beim Helikopter soll das Auto ein realistisches Gefühl geben und mit rollenden Rädern auf den Straßen fahren. BlenderKit bietet zwei Möglichkeiten an, wie die Modelle importiert werden sollen. Das Modell wird als ein einziges Objekt importiert, ein Nachteil wäre, dass die Einzelteile, die in einer Gruppe zusammengebunden sind aufgelöst werden müssten. Deswegen gibt es die zweite Möglichkeit Append, die genauso dieses Prinzip hat. Der Lamborghini wurde mit Append importiert. Um die Reifen zu drehen, wurde unter "Items" die X-Achsen Rotation geändert. Mithilfe vom Befehl #frame oder frame bewegen sich die Räder um jeden Frame, dies bewirkt eine durchgehende Drehung und illustriert ein fahrendes Auto.

Anfangs war es geplant ein Helikoptermodell von BlenderKit mit hinein zu importieren, jedoch kam es zu unbekannten Fehlern, die wir in der Zeit nicht finden konnten und so haben uns für die Alternative entschieden und selber eines erstellt. Wir vermuten, dass es sich um ein Versionsproblem handeln könnte, da Blender 2.83 und 2.9 für das Projekt verwendet wurden.

2.5 Detail Assets

2.5.1 Straßenschilder und Ampeln

2.5.1.1 Allgemein

Für die Erstellung der Straßenschilder bzw. Ampeln wurde eine Methode verwendet, die oft in VFX-Projekten verwendet wird. Darin wird ein 3D-Objekt aus einem Foto erstellt. Man könnte den Prozess, mit dem vom Abpausen vergleichen.

2.5.1.2 FSPY

Fspy ist eine Open Source Software, um die Kamera Perspektive von einem Bild zu reproduzieren. In dieser Software hat man vier Hilfslinien, jeweils zwei für eine Achse (entweder xy/xz/zy). Mit diesen Linien versucht man entlang dem Fluchtpunkt im „Raum“ zu richten. In unserem Fall werden die Achsen entlang den Seiten des Objekts gerichtet. Die Fspy Datei wird mithilfe des Fspy-Addons in Blender importiert. Dabei wird automatisch eine Kamera mit den richtigen Einstellungen generiert, worin man in der Kamera Perspektive das Originalbild, leicht transparent sieht.

2.5.1.3 Abpausen in Blender

In Blender versucht man die grundsätzliche Form des Objektes mit einer Primitive zu überlappen. Als Beispiel nehmen wir jetzt das „Elektrokasten“ Asset. Im Referenzfoto kann man erkennen das der Kasten wie ein hochgestellter Quader aussieht. Deswegen erstellen wir als erstes einen „Cube“. Mit dem Würfel im Edit Mode passt man alle Ecken an die vom Kasten an. Der weitere Prozess besteht aus dem Anordnen der Kante und Punkte in den richtigen Positionen.

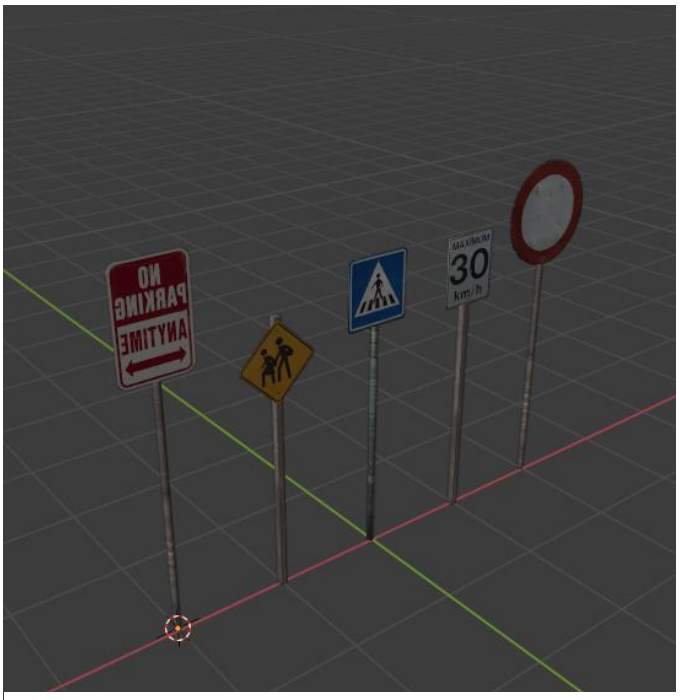


Abbildung 31: Straßen Assets

2.6 Optimierung der Assets

2.6.1 Texturen Baken

Mit dem „Baken“ ist es möglich mehrere Texturen eines Objektes in einer Datei zusammenzufassen. Diese schneidet alle ungebrauchten Materiale aus, welche die Perfomance deutlich erhöht.

Das Problem mit den meisten komplexeren Gebäuden ist, dass diese viele verschiedene Texturen brauchen (für Fenster, Fassaden, Türen, Dächer …). Dadurch sammeln sich im Projekt Ordner immer mehr Dateien an, die durchschnittlich 1-3 MB haben. Dadurch werden mit mehreren Gebäuden, die Blender-Datei bzw. das Add-on deutlich größer.

Um alle Texturen in eine Einzige zu exportieren benötigt man zuerst eine gemeinsame UV-Map. Diese wird mithilfe der „Smart UV Projection“-Methode erstellt. „Smart UV Projection“ generiert platzeffizient eine „Texture Map“ ohne dass sich Flächen überlappen. Daraufhin muss man in jedem Material eine neue „Image Texture“-Node hinzufügen. Davor muss man noch die Auflösung, sowie den Namen des Bakes festlegen.

Momentan ist das „Baken“ in Blender nur in der „Cycles“ Render Engine möglich. Bevor man den Prozess startet, muss man zuerst in den darin vorgesehenen Einstellungen die „Bake Type“ auf „Diffuse“ setzten, welche nur die Farben des Gebäudes berücksichtigt.

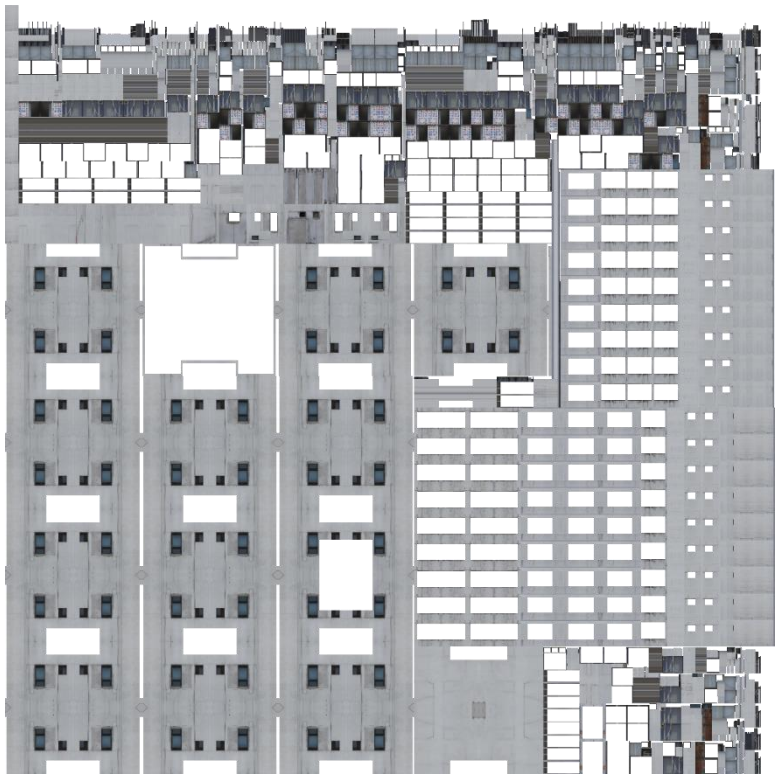


Abbildung 32: Textur nach dem Bake

3 Programmierung

3.1 Blender Add-on

3.1.1 BPY: die Python – Blender Schnittstelle

Die Blender Foundation stellt in Blender eine mächtige Python Schnittstelle zur Verfügung die den Umgang mit digitalen 3D (dreidimensionalen) Objekten abstrahiert und somit deutlich vereinfacht. Es gilt die Regel: Was man in Blender Interface machen kann, funktioniert genauso gut oder besser mit der Blender Python Schnittstelle.

Blender bietet einen signifikanten Vorteil beim programmirischen Umgang mit 3D Objekten im Vergleich zu anderen Low-Level Lösungen wie OpenGL oder DirectX, da grundsätzlich das Berechnen und Rendern von digitalen Formen extrem Rechenintensiv und Komplex ist, in Blender allerdings über eine benutzerfreundliche Schnittstelle abstrahiert wurde. Blender wurde genau für diesen Zweck mithilfe fortgeschrittener und standardisierter Algorithmen optimiert, zudem verfügt Blender über die Möglichkeit, bei spezifischen Anwendungsfällen auf Low-Level Funktionen zurückzugreifen. Aus diesem Grund haben wir uns für Blender als Host Programm für unsere Software entschieden.

Alle Skripte, die in Blender ausgeführt werden, haben auf das globale Modul bpy (Blender Python) Zugriff. Die einzige Voraussetzung ist das Importieren dieses Modules mittels des folgenden Codes:

```
import bpy
```

Nun steht das vollständige Arsenal an Funktionen und Werkzeugen für den Umgang mit digitaler Geometrie zur Verfügung. So kann man zum Beispiel einen Würfel mit dem folgenden Code erzeugen:

```
bpy.ops.mesh.primitive_cube_add(
    size=2,
    enter_editmode=False,
    align='WORLD',
    location=(0, 0, 0),
    scale=(1, 1, 1)
)
```

Hier wird der primitive Würfel mittels eines Operators (= eine vordefinierte Funktion mit einem bestimmten Zweck) an den Koordinaten (0|0|0) instanziiert. Mithilfe solcher Operators ist es möglich, die Generierung komplexer Szenerien zu automatisieren.

Zusätzlich ist es durch bpy möglich die Blender Benutzeroberfläche zu steuern und zu erweitern. Diese Eigenschaft ist für die Programmierung einer Erweiterung unbedingt notwendig. Verständlicherweise gilt Blender aus diesem Grund als beliebtes und höchst modernes System für die Entwicklung von 3D Grafikmanipulierenden Erweiterungen wie unserer.

Nichtsdestotrotz ist das Entwickeln in bpy nicht immer eine einfache und verständliche Angelegenheit. Durch die oftmals fehlende, unvollständige oder unverständliche Dokumentation muss man bereit sein, selbst im Quellcode von Blender nach Antworten und Lösungsansetzen zu stöbern.

3.1.2 Übersicht der Dateistruktur

Ein Blender Add-on setzt sich aus drei wichtigen Dateien zusammen

❖ Operators

Ein Operator ist eine Klasse, die eine oder mehrere definierte Aufgaben durchführt. Ähnlich wie beim Funktionellen Programmieren, erwartet man von einem Operator beim mehrmaligen Aufrufen immer dasselbe Ergebnis.

❖ Panels

Ein Panel beinhaltet die mit einem Operator verbundenen Benutzeroberflächenelemente. In einem Panel können Texte, Knöpfe, Select Boxen sowie diverse andere Elemente definiert und positioniert werden.

❖ Init-Datei

Eine Init-Datei ist der Ausgangpunkt jedes Blender Add-ons. Diese wird beim Installieren des Add-ons aufgerufen und dient dazu, alle notwendigen Python Module zu laden und zu registrieren.

Alle Komponenten des Add-ons bestehen somit aus zwei Klassen, die sich gegenseitig referenzieren, dem Operator (_op.py Endung) und dem Panel (_pl.py Endung).

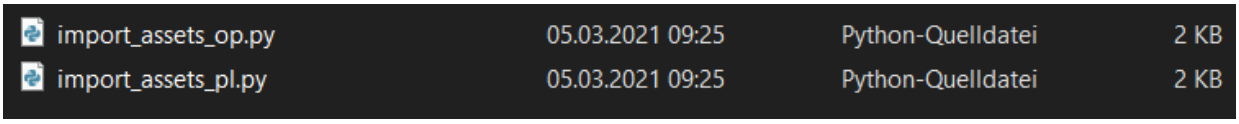


Abbildung 33: Die Operator- und Panel-Klasse des Asset-Importer

Damit ein Add-on als solches von Blender erkannt wird, sind zwei weitere Angaben in der Init-Datei notwendig:

❖ bl_info

bl_info ist ein Python Dictionary welches die wichtigsten Add-on Metadaten beinhaltet

```
bl_info = {
    "name" : "CitySketch",
    "author" : "CitySketch",
    "description" : "Generate large cities, road networks and buildings",
    "blender" : (2, 80, 0),
    "version" : (1, 0, 0),
    "location" : "",
    "warning" : "",
    "category" : "Generic"
}
```

Abbildung 34: Metadaten des CitySketch Add-ons

❖ Module die Registriert werden müssen

Beim Installieren eines Blender Add-ons registriert Blender alle notwendigen Module, sodass diese falls gebraucht sofort verfügbar sind. Dies ist notwendig, da Blender nicht weiß, wo es nach diversen selbstgeschriebenen Modulen suchen soll. Die dazugehörige unregister Methode entfernt diese Module beim Deinstallieren des Add-ons. Dies ist ein Best Practise. (Add-on Tutorial - Blender Manual, 2021)

```
classes = (
            OBJECT_OT_realistic_generator,
            OBJECT_OT_generate_city,
            PANEL_PT_generate_city,
            OBJECT_OP_import_assets,
            PANEL_PT_import_assets,
            PANEL_PT_import_options,
            PANEL_PT_generate_city_semi_random,
            PANEL_PT_generate_city_panel_realistic,
            PANEL_PT_generate_city_panel_singular_street,
            PANEL_PT_generate_city_panel_advanced,
            PANEL_PT_point_of_interest,
            PANEL_PT_details,
            PANEL_PT_green_areas,
            PANEL_PT_river,
            PANEL_PT_generate_options,
            OBJECT_OT_draw_sketch,
            OBJECT_OT_disable_drawing,
            OBJECT_OT_clear_sketch

        )

register, unregister = bpy.utils.register_classes_factory(classes)
```

Abbildung 35: Register und Unregister Methode für die Klassen von CitySketch

3.1.3 Externe Python Module

Das Einbinden von externen Python Modulen in ein Blender Add-on ist oftmals eine Herausforderung. Blender verfügt über eine vorinstallierte Python Distribution, um unabhängig vom Betriebssystem und der Konfiguration zu funktionieren. Um ein externes Modul zu verwenden muss man dieses in das vorinstallierte Python einschleusen.

CitySketch setzt für das korrekte Funktionieren die Bibliothek Shapely voraus. Diese ist allerdings nicht Teil der vorinstallierten Distribution und muss daher beim Aktivieren des Add-on nachinstalliert werden. Um die Benutzerfreundlichkeit nicht zu beinträchtigen, muss dies automatisch geschehen. Im Laufe dieser Diplomarbeit wurden mehrere Arten der Moduleinbindung getestet. Als beste Möglichkeit erwies sich das Installieren bei Aktivierung mithilfe von Sub-Prozessen.

Dabei wird mithilfe eines Sub-Prozesses der Python Package Manager (kurz **pip**, ein rekursives Akronym für „Pip installiert Packages“) der Blender Python Distribution aufgerufen, welcher die aktuelle sowie passende Version des Moduls installiert.

```
subprocess.call([
     str(py_exec),
     "-m",
     "pip",
     "install",
     f"--target={str(py_exec)[:-14]}" + "lib",
     "shapely"
 ])
```

Dies ist allerdings nur möglich, sofern der Package Manager pip verfügbar ist. In manchen Blender Versionen ist dies allerdings nicht der Fall (insbesondere Linux Distributionen). Aus diesem Grund muss pip bereits davor installiert beziehungsweise aktualisiert werden. Dies geschieht mithilfe folgender Befehle:

```
subprocess.call([
     str(py_exec),
     "-m","ensurepip",
     "--user"
 ])
 subprocess.call([
     str(py_exec),
     "-m",
     "pip",
     "install",
     "--upgrade",
     "pip"
 ])
```

3.2 Einlesen von Benutzerzeichnungen

3.2.1 Übersicht

Ein wichtiger Aspekt von *CitySketch* ist die Zeichenfunktion, die dem Benutzer ermöglicht das Grundlayout der Stadt wie auf einem Blatt Papier zu skizzieren. Dies ist ein origineller und intuitiver Weg die Generierung zu beeinflussen.

Um dies zu erreichen, verwenden wir selbstgeschriebene Operatoren die intern das Grease Pencil Tool von Blender aufrufen und bedienen. Dazu wurden drei Operatoren geschrieben, die das folgende ermöglichen:

- Zeichenmodus aktivieren
- Zeichenmodus deaktivieren
- Zeichenfläche zurücksetzen

Das besondere bei diesen Operatoren ist, dass diese ausschließlich im Kontext des eigenen Layers ausgeführt werden, dadurch wird erreicht, dass das Zurücksetzen des Layers „Grünzonen“, andere Layer nicht beeinflusst, obwohl diese optisch übereinander liegen.

Im Anschluss wird die Zeichnung evaluiert und beim Generieren der Stadt zur Positionsbestimmung für die jeweiligen Strukturen angewendet.

3.2.2 Blender Grease Pencil

Das Grease Pencil Tool von Blender ist ein eingebautes Werkzeug zum Zeichnen von 2D Grafiken im 3D Raum. Es unterstützt verschiedene Layer und Farben, die es erlauben, die Zeichnungen zu strukturieren und zu unterteilen. Grease Pencil kann als Annotationstool, sowie für Zeichentrickfilm-Animationen genutzt werden. Im Fall von CitySketch wird dem Tool ein neuer Nutzen verliehen: als Schnittstelle zwischen Benutzer und Add-on. (Grease Pencil: blender.org, 2020)

3.2.3 Auswerten der Zeichnungen

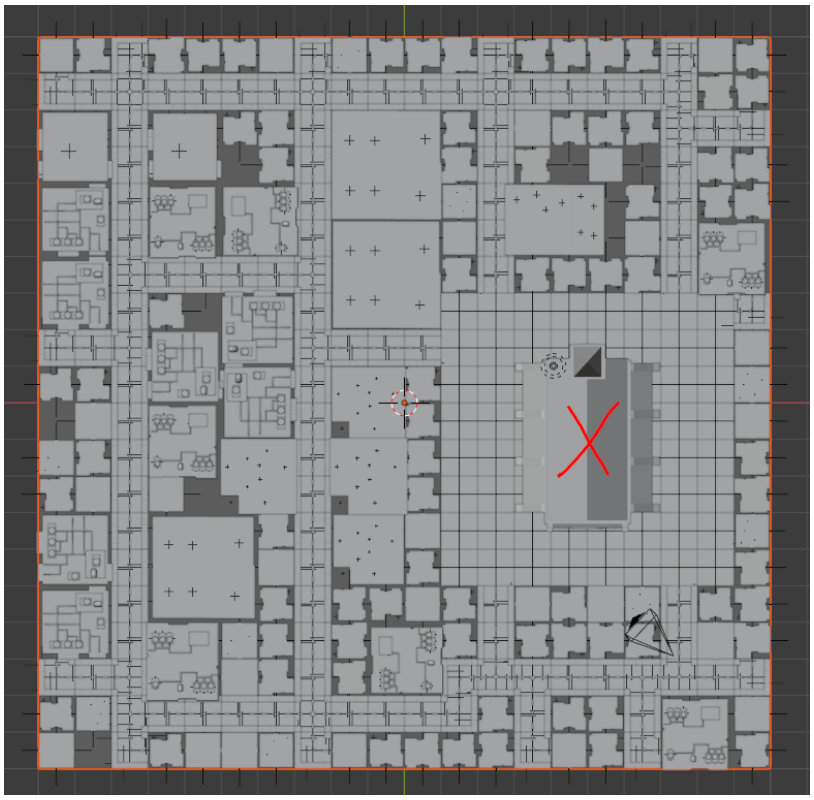


Abbildung 36: Eingezeichneter Point of Interest

Nachdem der Benutzer auf *Generieren* drückt, werden die eingezeichneten Skizzen evaluiert. Je nachdem welche Strukturen im Interface des Add-on eingeschalten wurden, werden die passenden Skizzen berücksichtigt. Dies ist möglich, da die Zeichnungen auf unterschiedlichen Ebenen in Blender gespeichert sind (Erkennbar an der Farbe des Stiftes, zb.: grün für Grünzonen) und so voneinander differenziert werden können.

Zuerst wird der Point of Interest bestimmt. Dazu werden die Layer für den Point of Inte werden, gilt die Positionsbestimmung als erfolgreich. Ist dies der Fall, werden die Koordinaten des nächstgelegenen Gitterfeldes berechnet. Zuletzt wird der Point of Interest im Mittelpunkt dieses Feldes instanziiert. Um den Point of Interest wird außerdem ein Bereich, mit der im User Interface definierten Größe reserviert. Dieser wird im Anschluss mit leeren Betonflächen-Assets befüllt.

Das Überprüfen der Kriterien für einen Grease Pencil Layer geschieht mittels des folgenden Codes:

```
#PROCESS DRAWING IF EXISTS
gp = bpy.data.grease_pencils.get("CitySketch")
if gp is not None:
    layer = gp.layers.get("Point of Interest")
    if layer is not None:
```

```
#individual lines the user has drawn
strokes = layer.active_frame.strokes

#check if at least 2 lines are present
if len(strokes) >= 2:
```

Die vom Benutzer eingezeichneten Strokes („Linien“) befinden sich in dem Active_Frame Objekt eines Layers, welcher sich in dem vom Benutzer verwendeten Grease Pencil Objekt befindet.

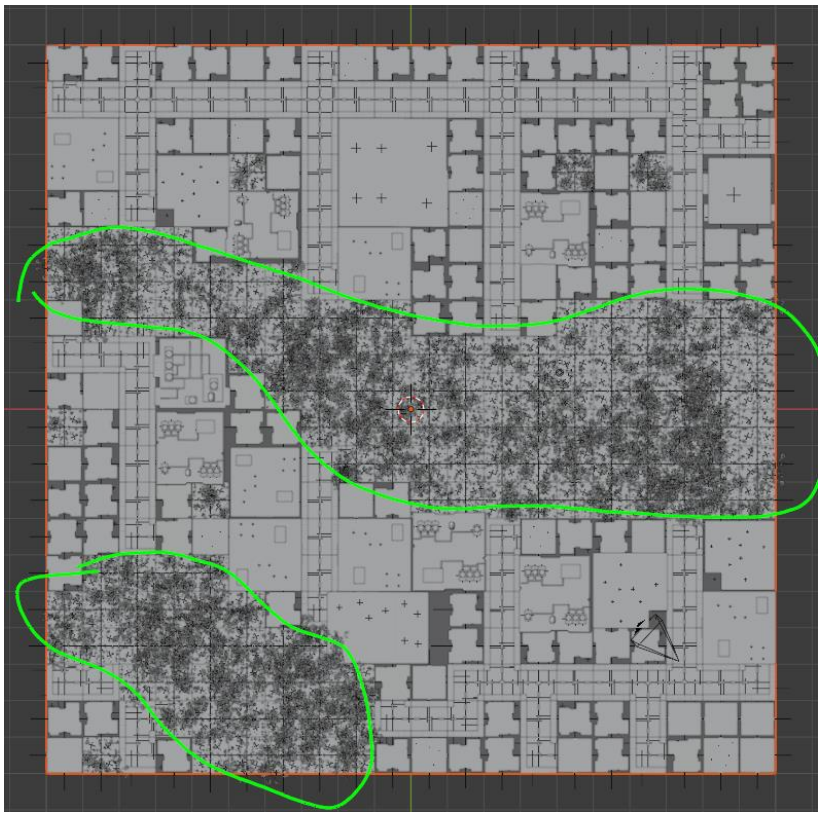


Abbildung 37: Zwei eingezeichnete Grünzonen/Parks

Ein ähnlicher Prozess geschieht beim Generieren der Grünzonen. Der Unterschied ist der für die Evaluierung verwendete Algorithmus. Im ersten Schritt werden die Strokes zu geometrischen Objekten umgewandelt. Dafür wird die populäre Python Bibliothek Shapely verwendet. Diese wandelt einen Array an Koordinaten in Shapely-Formen um, die dann mithilfe geometrischer Funktionen verarbeitet werden können. Als nächstes wird festgestellt, welche Gitterpunkte innerhalb der Formen liegen, sollten welche gefunden werden, wird das dazugehörige Gitterfeld als Grünzone markiert.

In diesem Schritt werden noch keine Bäume/Büsche instanziiert, denn dies erfolgt über den Grünzonen Generator, welcher im späteren Verlauf genauer beschrieben wird. Der Grund dafür ist die Unabhängigkeit der Grünzonenobjekte vom Gitter.

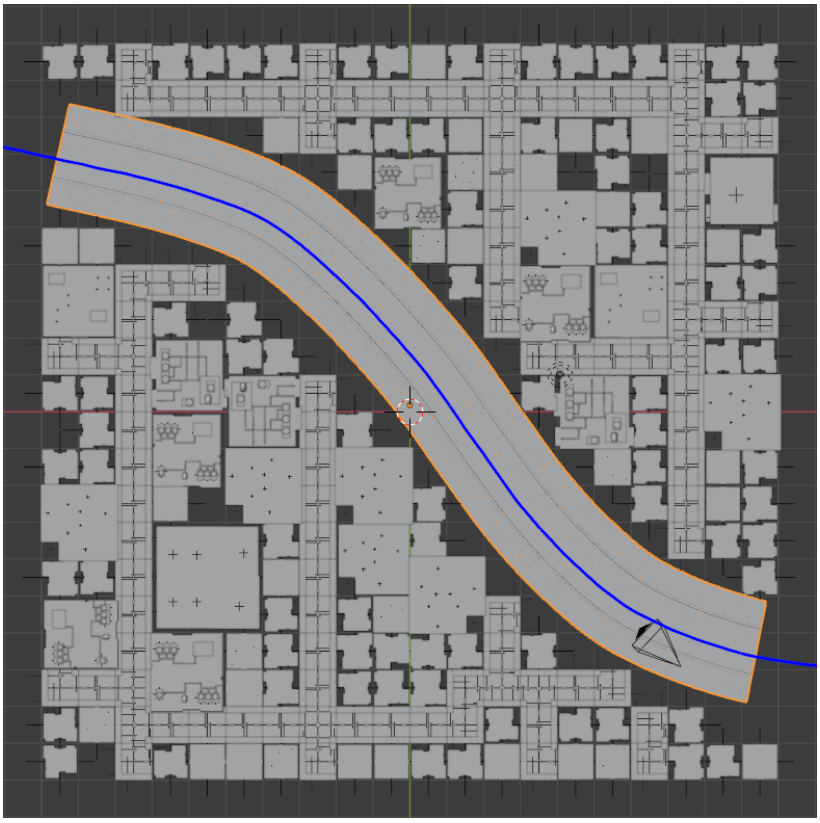


Abbildung 38: Eingezeichneter Flussverlauf

Die Auswertung der Fluss Zeichnungen funktioniert, indem die Strokes des Benutzers in die einzelnen Punkte, aus denen die Linie besteht, unterteilt werden. Anschließend werden die Punkte zu einem Shapely LineString umgewandelt der es erlaubt, die kürzeste Distanz zu einem beliebigen Punkt auf der 2D Ebene zu berechnen. Diese Distanz wird verwendet um den Bereich rund um den Fluss zu bestimmen, der für den Fluss reserviert bleiben muss.

3.3 Noise Funktionen

3.3.1 Übersicht

Im Laufe des Codes wird immer wieder auf sogenannte Noise Funktionen zurückgegriffen. Eine solche Funktion ermöglicht die Generierung pseudo-zufälliger Zahlen, die von einer anderen benachbarten Zahl auf einer fiktiven zwei-dimensionalen Ebene abhängen. So lassen sich großräumige Muster beim Betrachten mehrerer solcher Zahlen erkennen. Diese Muster werden als Grundlage der Positionsbestimmung für einzelne Objekte verwendet.

Ein gutes Beispiel ist die Generierung von Grünzonen. Das Ziel ist es, zufällige Wälder zu generieren, aber gleichzeitig die Makroformationen zu behalten. Eine Mischung aus Zufall und Muster ist also notwendig. Für diese Zwecke sind Noise Funktionen optimal, denn diese erfüllen die gerade genannten Voraussetzungen.

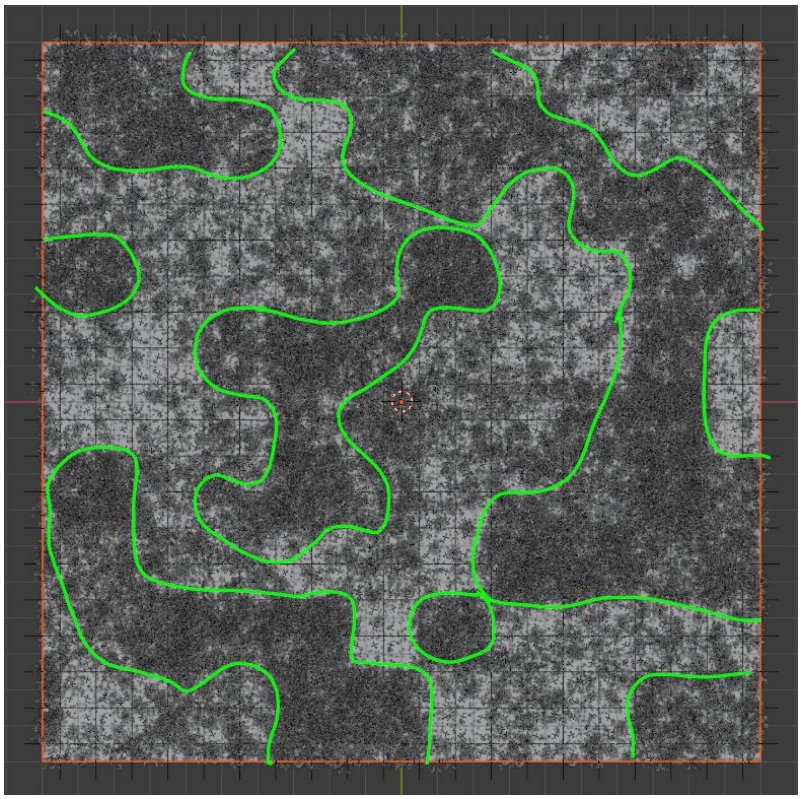


Abbildung 39: Noise Funktionen im Einsatz beim Generieren von Grünzonen

Auf dem obigen Bild sieht man eine 20 mal 20 Blender Units große Grünzone. Man erkennt (hier nochmal grün markiert) Dichteunterschiede bei der Verteilung der einzelnen Bäume. Ebenso bemerkenswert ist das Nichtvorhandensein abrupter Dichteübergänge. Dies ist eine weitere Eigenschaft diverser Noise Funktionen.

(Noise Utilitites Blender, 2021)

3.4 Straßensystem Generator

3.4.1 Übersicht

Der Straßensystem Generator legt fest an welchen Positionen Straßen-Assets platziert werden sollen und ist dementsprechend dafür verantwortlich, wie das Straßennetz in jeder generierten Stadt aussieht. Die Straßennetze werden generiert, basierend auf den vom Benutzer festgelegten Parametern “Street Ratio”, “Symmetry” und “Size”.

Der Street Ratio Parameter hat einen ganzzahligen Wert von null bis vier, wobei null keine Straßen und vier sehr viele Straßen im Straßennetz erzeugen.

Symmetry hat einen ganzzahligen Wert von eins bis drei, wobei eins ein sehr organisches und drei Straßennetz aus parallelen und rechtwinkeligen Straßen ergibt, wie man es zum Beispiel aus New York kennt.

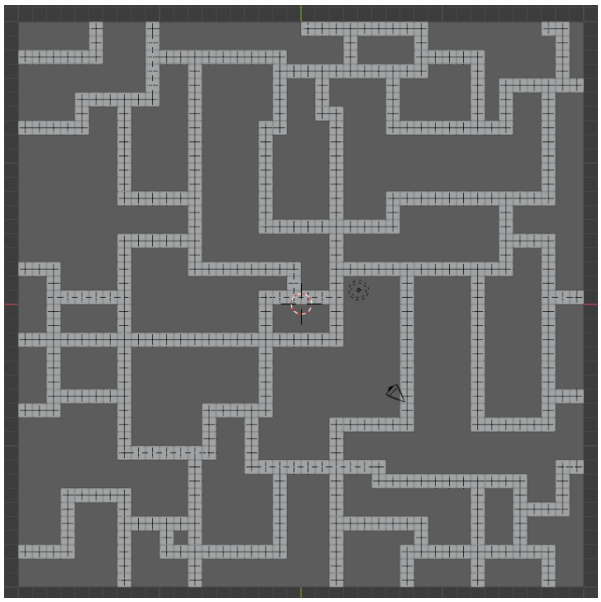


Abbildung 41: Straßennetz bei Symmetry "1"

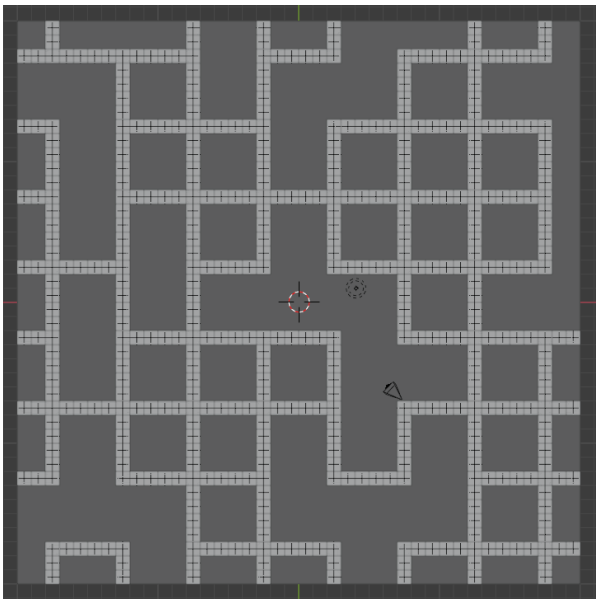


Abbildung 40: Straßennetz bei Symmetry "3"

Size hat einen ganzzahligen positiven Wert und bestimmt die Seitenlänge der Stadt.

3.4.2 Ansätze

Folgende Ansätze wurden bei der Umsetzung des Straßensystem Generators in Betracht gezogen:

3.4.2.1 Grease Pencil Verfahren

Das Grease Pencil Verfahren würde dem Benutzer ähnlich wie bei dem Grünzonen- und Fluss Generator die Möglichkeit geben, mithilfe des Grease Pencil Tools in Blender alle Straßen selber einzuzeichnen. Der Vorteil hierbei wäre, dass der Benutzer vollständige Entscheidungsfreiheit darüber hätte, wie sein Straßennetz aussehen würde. Dies würde jedoch, vor allem bei größeren Städten, viel zu lange dauern. In Abbildung 33 sieht man eine vergleichsweise kleine Stadt, in welcher es eine halbe Minute gedauert hat, alle Straßen

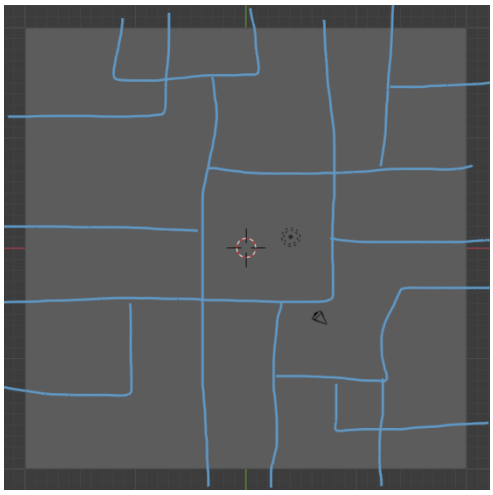


Abbildung 42: Beispiel für eine mögliche Einzeichnung eines Straßensystems in der Stadt

sinnvoll einzuzeichnen. Das würde in diesem Fall selbst bei so einer kleinen Stadt schon länger dauern als sich mit allen anderen Einstellungsmöglichkeiten zusammen auseinanderzusetzen. Da einer der Grundprinzipien von CitySketch ist, so schnell und simpel wie möglich zu funktionieren, wurde dieser Ansatz nicht in die finale Version implementiert.

3.4.2.2 Rekursion

Die nächste Version, welche als sekundärer Generator “Semi-Random” in CitySketch implementiert ist, basiert darauf, dass vom Mittelpunkt der Stadt aus in jede der vier Himmelsrichtungen eine gerade Straße, Zelle für Zelle generiert wird. Bei jedem neuen Straßenstück wird mit einer voreingestellten Wahrscheinlichkeit eine Kreuzung generiert, bei welcher neue Straßen in andere Richtungen mithilfe des gleichen Prozesses generiert werden, wodurch eine Rekursion entsteht. Der Vorteil bei diesem Ansatz ist, dass die resultierenden Straßennetze, auf Grund der ausschließlich chancenbasierenden Generierung, eine sehr hohe Vielfalt haben. Jedoch entstehen dementsprechend auch bei weitem öfters, als bei anderen Ansätzen, weniger brauchbare Ergebnisse (siehe Bild). Des Weiteren wird die maximale bei Rekursionstiefe bei einer Stadtgröße von ungefähr 150x150 erreicht, wodurch diese Lösung weniger optimal ist und dementsprechend nur der sekundäre Generator ist.

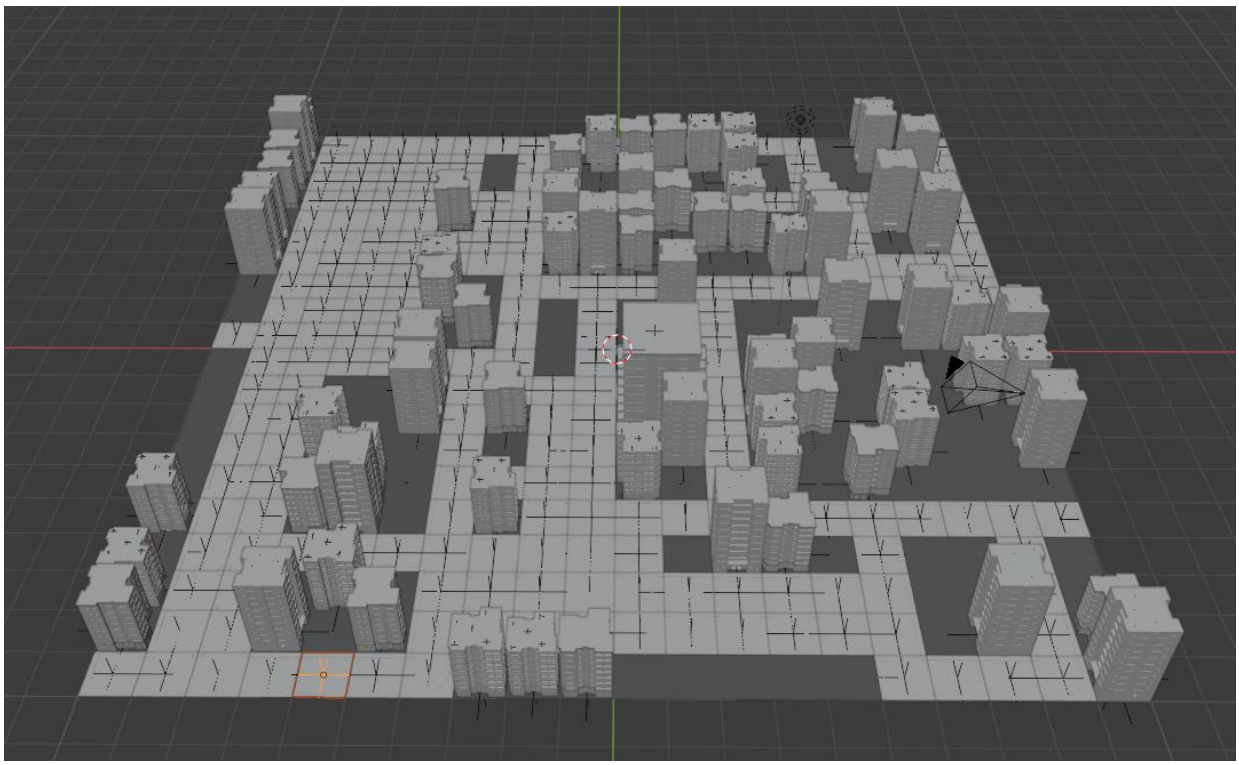


Abbildung 43: Suboptimales Straßennetz bei der Benutzung des "semi-random" Generators

3.4.2.3 Template System / Iterative Methode

Der schlussendlich umgesetzte Ansatz basiert auf vordefinierten Straßensegmenten welche passend aneinandergefügt werden. Diese Straßensegmente werden alle per Hand “gezeichnet” (siehe “4.4.3 Segmente”) und werden so aneinandergefügt, dass keine ungewollten Sackgassen entstehen und ein realistisches Straßennetz entsteht. Diese Lösung hat den großen Vorteil, dass, durch die eigene Erstellung der Segmente, das Risiko ein unbrauchbares Straßennetz zu generieren eliminiert wird, da sehr genau im Vorhinein festgelegt werden kann, wie mögliche Straßennetze aussehen werden. Der einzige Nachteil an dieser Methode ist, dass der Benutzer das Straßennetz weniger mitgestalten kann. Dieses Problem wurde jedoch mit der Einführung der beiden Parameter “Symmetry” und “Street Ratio” behoben, da der Benutzer dadurch wieder mehr Entscheidung darüber hat, wie das Straßennetz schlussendlich aussieht.

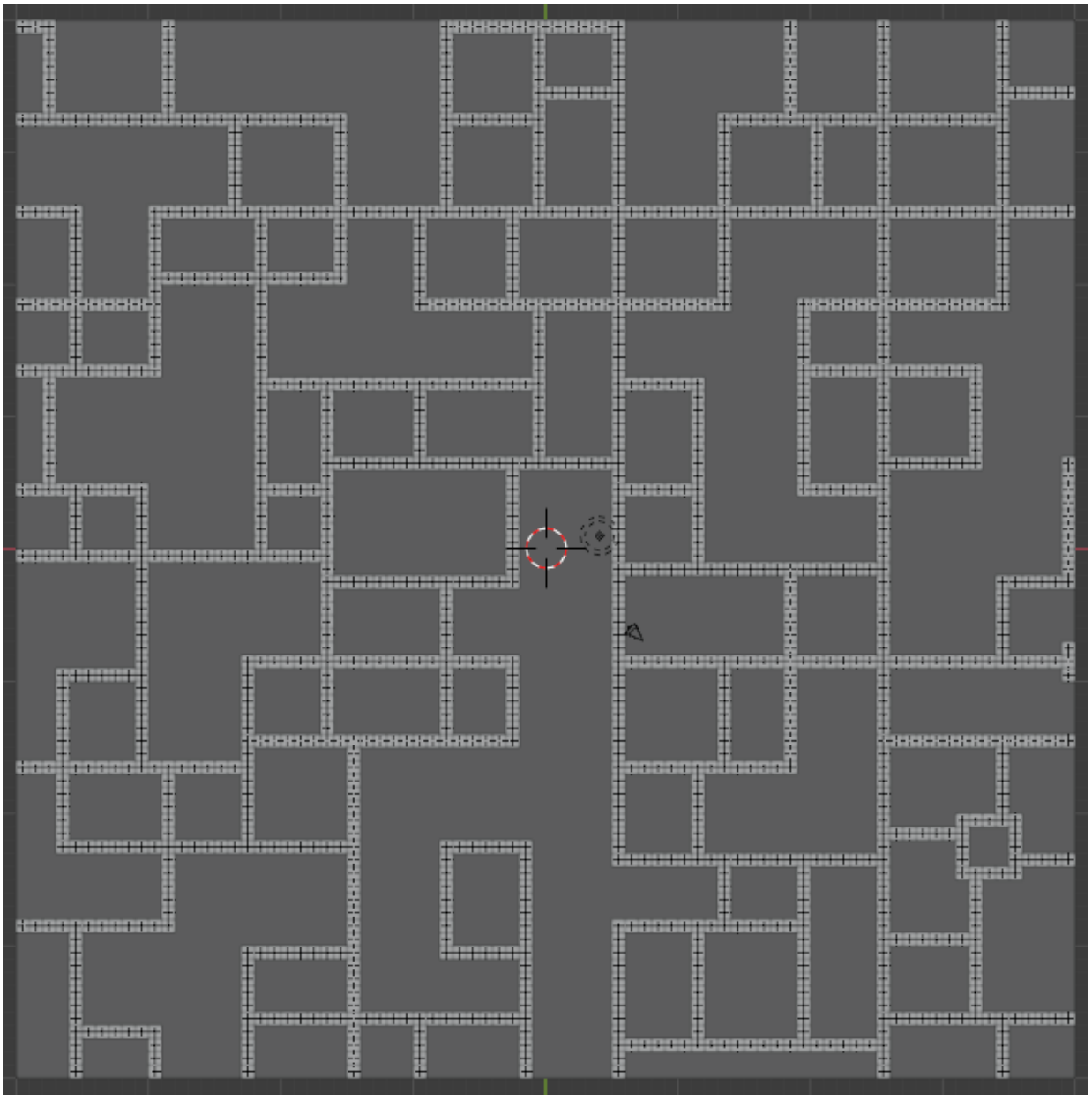


Abbildung 44: Straßennetz generiert durch den primären "Realistic"Generator

3.4.3 Segmente

Die Straßensegmente für den Straßensystem Generator sind speziell zugeschnitten auf genaue Kombinationen aus den Werten von Street Ratio und Symmetry.

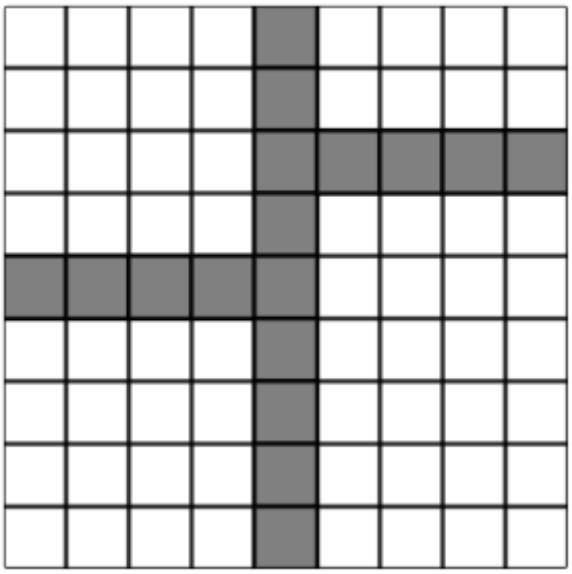


Abbildung 45: Beispiel für ein Segment für Symmetry "2" und Street Ratio "2"

Da es für den Straßensystem Generator neun verschiedene Kombinationsmöglichkeiten der beiden Parameter gibt (jeweils von eins bis drei, da Street Ratio auf "0" bedeuten würde, dass keine Straßen generiert werden), gibt es neun verschiedene Kategorien, in denen die jeweiligen Segmente für die korrespondierende Kombination aus den Parametern gespeichert werden. Alle Segmente wurden per Hand in der eigens dafür programmierten Web-Anwendung auf https://getcitysketch.com/road-segment-drawer/ eingezeichnet und als JSON-Datei exportiert, um später schnellstmöglich eingelesen werden zu können. (Nurseitov, Paulson, Reynolds, & Izurieta, 2009)

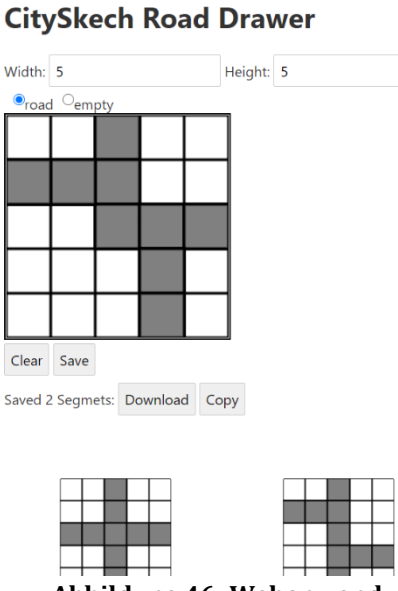


Abbildung 46: Webanwendung

Im Code stehen sie dann als “quadratische” 2D-Arrays mit dem Wert “0” an Stellen, an der sich keine Straße befindet und dem Wert “2” an Stellen, wo sich eine Straße befindet, dem Straßensystem Generator zur Verfügung.

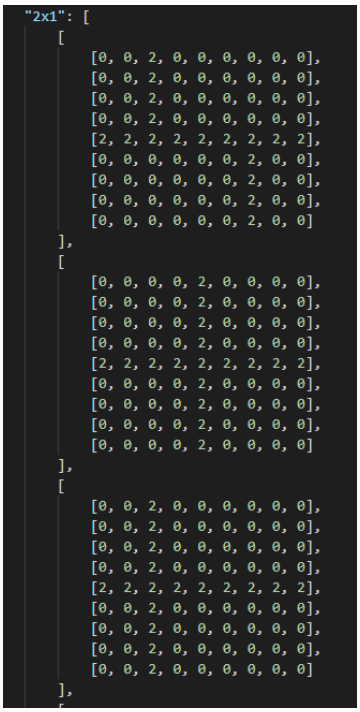


Abbildung 47: Beispiel für die ersten drei Segmente in der Kategorie Symmetry “2” und Street Ratio “1”

Grundsätzlich gilt jedoch bei jedem Segment, dass niemals eine Straße in einer Ecke sein kann. Bei der Erstellung der Segmente gibt es jedoch noch weitere verschiedene Regeln, je nach den Werten der Parameter, nach denen sich alle Segmente halten müssen. Ein Beispiel dafür wäre, dass sich die Seitenlänge der Segmente mit steigendem Street Ratio verringern muss, um somit ein dichteres Straßennetz zu erzeugen.

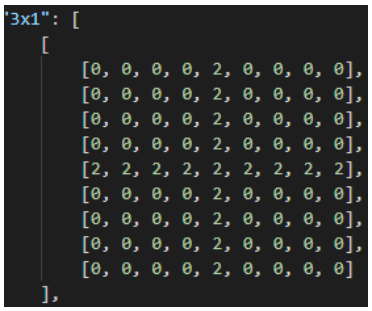


Abbildung 46: Beispiel Straßensegmente der Symmetry "3" mit Street Ratio "1"

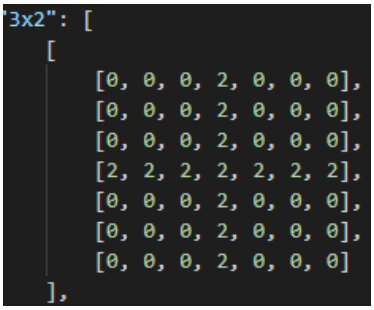


Abbildung 48: Beispiel Straßensegmente der Symmetry "3" mit Street Ratio "2"

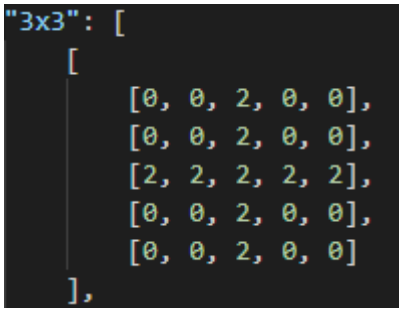


Abbildung 48: Beispiel Straßensegmente der Symmetry "3" mit Street Ratio "3"

Bei den Segmenten der Symmetry “3” wurden nur symmetrische Straßensegmente verwendet, welche an allen Rändern entweder keine Punkte hatten an denen Straßen fortführen oder nur in der Mitte der jeweiligen Seite.

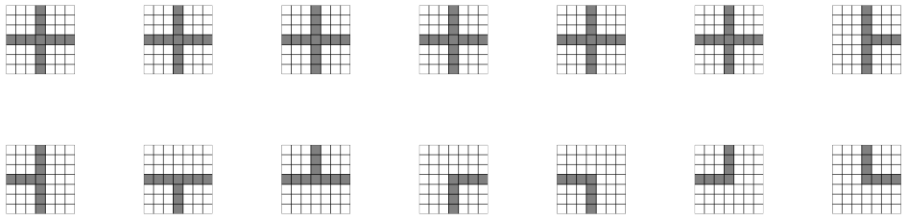


Abbildung 49: Beispiel Straßensegmente der Kategorie Symmetry "3" und Street Ratio "2"

Außerdem ist in diesem Fall zu erkennen, dass die ersten sechs Segmente alle die gleichen sind. Das ist der Fall damit sie eine erhöhte Wahrscheinlichkeit haben verwendet zu werden, um zu vermeiden, dass es bei größeren Städten zu mehreren abgetrennten Straßenbereichen geführt.

Beim Aneinanderfügen der Segmente wird, worauf später bei “4.4.4 Technische Aspekte” noch genauer eingegangen wird, darauf geachtet, dass 2 Segmente neben- oder übereinander immer die gleichen Ein- und Ausgangspunkte haben, um ungewollte Sackgassen zu vermeiden. Das führt dazu, dass beim Erstellen der Straßensegmente darauf geachtet werden muss, dass alle Kombinationen aus Ein- und Ausgangspunkten gleich oft verwendet werden, um sicherzustellen, dass es in größeren Städten nicht vermehrt zur Wiederholung von Sequenzen aus Straßensegmenten kommt. Bei den Segmenten der Symmetry “2” und “1” wurden deswegen auf jeder Seite immer drei Punkte festgelegt bei denen Straßen Ein- und Ausgänge entstehen konnten, wobei alle Segmente einer gleichen Kategorie immer die gleiche Seitenlänge und idente Ein- und Ausgangspunkte haben.

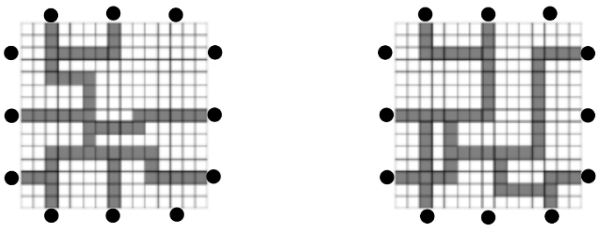


Abbildung 50: Zwei Segmente von Symmetry “1” und Street Ratio “2” bei denen die möglichen Ein- und Ausgangspunkte gekennzeichnet wurden

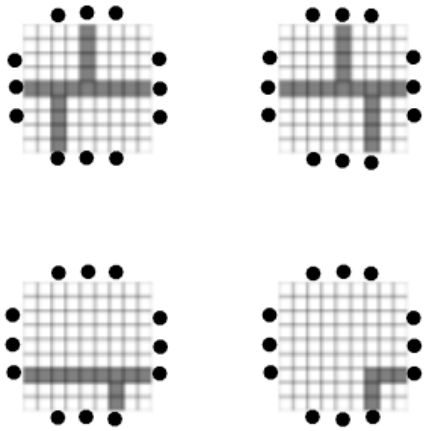


Abbildung 51: Vier Segmente von Symmetry "2" und Street Ratio "2" bei denen die möglichen Ein- und Ausgangspunkte gekennzeichnet wurden

Während alle Segmente mit der Symmetry "2" auf jeder Seite immer entweder einen oder keinen Punkt verwendet haben, wurden bei Segmenten der mit der Symmetry "1" immer genau zwei Punkte verwendet. Der Grund dafür ist, dass Symmetry "1" Straßensegmente immer in größeren Feldern gezeichnet wurden, um mehr Freiraum beim Einzeichnen von organischen Straßensystem zu bieten und dementsprechend bei den anderen Symmetry Werten ausgeglichen werden musste, um bei gleichem Street Ratio auch gleich viele Straßen zu haben.

Alle Segmente der Symmetry "2" wurden nach dem Prinzip erstellt jede mögliche Kombination aus Ein- und Ausgangspunkten zu benutzen ohne dabei mehr Abzweigungen als nötig zu verwenden. Dabei wurden in diesem Fall von Symmetry "2" und Street Ratio "2" insgesamt 129 verschiedene Straßensegmente erstellt.

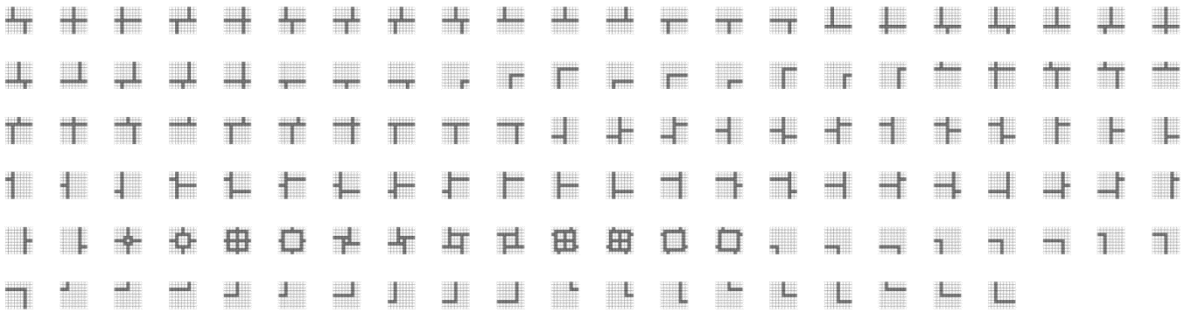


Abbildung 52: Beispiel Segmente von Symmetry "2" und Street Ratio "2"

Da die Segmente von Symmetry "1" um einiges größer waren, war es nicht nötig so viele zu erstellen jedoch musste auch hier darauf geachtet werden alle Ein- und Ausgangspunkte gleich oft zu verwenden. Schlussendlich wurden hier die Segmente so erstellt, dass jede Kombination aus linken und oberen Punkten (oder rechten und unteren Punkten) drei Mal mit Verschiedenen Punkten auf der gegenüberliegenden Seite existiert. Da es auf zwei Seiten neun verschiedene Kombinationen mit jeweils zwei von drei verwendeten Punkten

gibt, wurden insgesamt 27 verschiedene Segmente für jedes der drei "Segmentpacks" mit der Symmetry "1" erstellt. Bis auf die Randpunkte gab es für den restlichen Teil der Straßensegmente keine definierten Regeln bis auf dass die Straßen möglichst unterschiedlich zu anderen Straßensegmenten in der gegebenen Kategorie und organisch aussehen sollten.

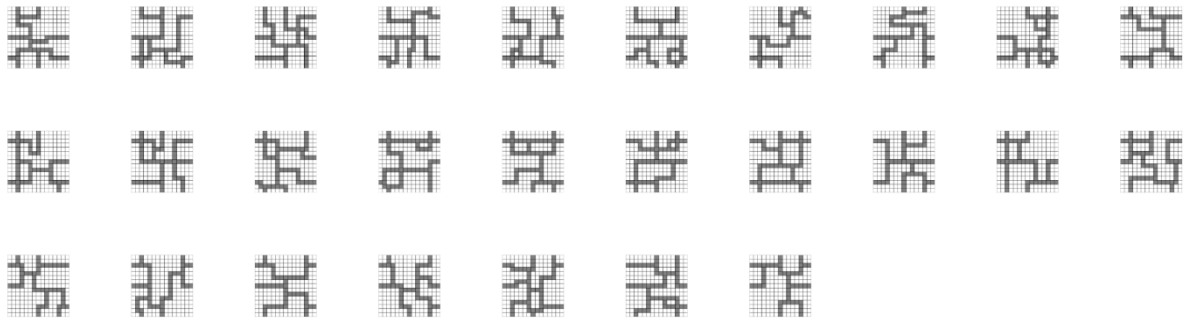


Abbildung 53: Beispiel Segmente von Symmetry "1" und Street Ratio "2"

Insgesamt wurden über 500 Segmente erstellt.

3.4.4 Technische Aspekte

Die Generatoren in CitySketch basieren auf dem Prinzip, dass anfangs ein 2D Array, welches die gleiche Seitenlänge wie die Stadt hat, generiert wird. Jedes Feld im Array steht für das korrespondierende Feld im Raster der Stadt und wird anfangs an jeder Stelle mit einer Null befüllt.

```
#get the dimensions for the city and generate the main array
w, h = city_size, city_size
city = []
. . .
city = [[0 for x in range(w)] for y in range(h)]
```

Eine Null steht dafür, dass in diesem Feld in der Stadt nichts eingefügt wird. Anschließend läuft jeder Generator nacheinander über das Array und befüllt die Stellen, an denen er ein Asset seiner Kategorie einfügen will, mit der Zahl, die für diese Kategorie steht. Ein Zweier steht für eine Straße und dementsprechend befüllt der Straßenssystem Generator jede Stelle im Array mit einer Zwei, wo eine Straße später eingefügt werden soll.

Als ersten Schritt lädt der Straßensystem Generator die JSON-Datei, die alle Straßensegmente enthält, und überprüft die vom Benutzer übergebenen Werte von Street Ratio und Symmetry um dann aus der JSON-Datei alle Segmente herauszufiltern, welche für die Kombination der beiden Parameter gedacht sind. Hierbei ist auch eine Exit-Condition eingebaut, falls der Street Ratio den Wert "0" hat, da in diesem Fall keine Straßen generiert werden müssen.

```
#return empty city if
        if street_ratio == 0:
            return city
```

Alle relevanten Straßensegmente werden dann in der Liste “loadedSegmentList” gespeichert, welche im gesamten weiteren Prozess verwendet wird, um passende Segmente zu finden. Dann beginnt der Straßensystem Generator die Segmente in den Haupt-Array der Stadt einzufügen. Dazu werden zwei pointer-Variablen für die derzeitige x- und y-Koordinate erstellt, welche jeweils bei 0 (ganz links und ganz oben) beginnen und immer, nachdem ein Segment eingefügt wird erhöht sich der x-pointer um die Seitenlänge des eingefügten Segments, bis der x-pointer die Seitenlänge der Stadt überschreitet. Dann wird der x-pointer auf 0 zurückgesetzt und der y-pointer wird um die Seitenlänge der eingefügten Segmente erhöht und der ganze Prozess wiederholt sich. Das führt dazu, dass die Segmente Zeile für Zeile von links nach rechts eingefügt werden bis schlussendlich der y-pointer die Seitenlänge der Stadt überschreitet, woraufhin der Prozess endet. Das Ganze wird mithilfe von zwei while-Schleifen gemacht:

```
while y < height:
            while x < width:
. . .
                    x += segmentWidth
            y += segmentWidth
            x = 0
return city
```

Um ein Segment einzufügen wird zuerst die Liste loadedSegmentList in die neue Liste currentSegmentList kopiert. Danach wird in einer `while True:` Schleife, Ein zufälliges Segment aus currentSegmentList entfernt, und in einer neuen Variable randomSegment eingefügt wird.

```
currentSegmentList = loadedSegmentList.copy()
   while True:
         rnd = r.randint(0, len(currentSegmentList)-1)
         randomSegment = currentSegmentList[rnd]
         currentSegmentList.pop(rnd)
```

Bevor jedoch ein Segment eingefügt wird, wird noch überprüft, ob die Straßen an den Rändern mit dem Segment welches sich darüber, und dem welches sich links von dem neuen Segment befindet, zusammenpassen. Dazu werden alle Werte des Segments, welches sich am linken und oberen Rand befinden mit allen Werten des rechten Randes des linken

Segments und des unteren Randes des oberen Segments (der oberste Punkt im rechten Segment und der linkste Punkt im oberen Segment werden zu einem Wert zusammengefasst, da sich bei Segmenten so oder so niemals eine Straße in einer Ecke befindet) verglichen.

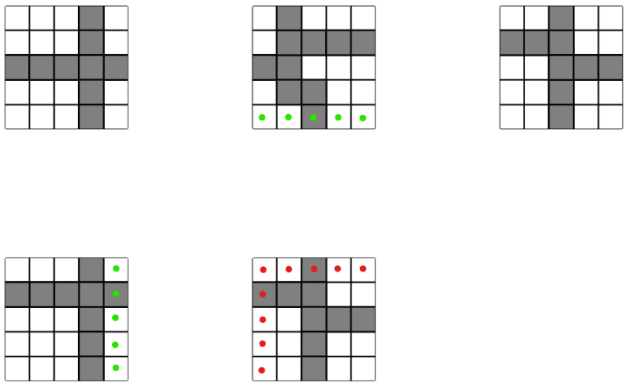


Abbildung 54: Die Werte mit den roten Punkten im neuen Segment werden mit den Rändern der be-reits eingefüllten Segmente (grüne Punkte) verglichen

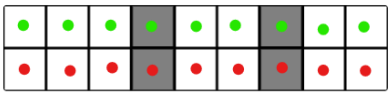


Abbildung 55: Veranschaulichung wie die Werte verglichen werden. In diesem Fall wäre es ein kompatibles Segment

Im Code werden für diesen Vergleich zuerst zwei Arrays mit der doppelten Seitenlänge des Segments minus eins (da die Ecke nicht zweimal verwendet wird) erstellt und an jeder Stelle mit einer "0" befüllt:

```
innerOutline = [0] * (segmentWidth*2 - 1)
outerOutline = [0] * (segmentWidth*2 - 1)
```

"innerOutline" steht für den Umriss des neuen Segments und "outerOutline" für die Ränder der existierenden Segmente. Diese werden jeweils in einfachen for-Schleifen befüllt mit if-statements richtig befüllt:

```
for i in range(segmentWidth + segmentWidth - 1):
    if i < segmentWidth:
            innerOutline[i] = randomSegment.grid[segmentWidth - i - 1][0]
            if x > 0 and len(city) > (y + segmentWidth - i - 1):
                    outerOutline[i] = city[y + segmentWidth - i - 1][x - 1]
            else:
                    outerOutline[i] = -1
    else:
```

```
            innerOutline[i] = randomSegment.grid[0][i - segmentWidth + 1]
            if y > 0 and len(city[y - 1]) > (x + i - segmentWidth + 1):
                  outerOutline[i] = city[y - 1][x + i - segmentWidth + 1]
            else:
            outerOutline[i] = -1
```

Hierbei werden alle Werte, beginnend links unten, bis nach rechts oben von dem jeweiligen Segment kopiert und in den Array für den jeweiligen Umriss eingefügt. Außerdem wird überprüft, ob sich die x- oder y-pointer am linken oder oberen Rand der Stadt befinden, denn das würde bedeuten, dass es kein Segment für die outerOutline an dieser Stelle gibt und es deswegen egal ist, ob das neue Segment an diesen Stellen eine Straße hat oder nicht.

Nach diesem Schritt werden noch die beiden Umrisse mithilfe der Funktion compareOutlines verglichen:

```
def compareOutlines(self, innnerOutline, outerOutline):
    for i in range(len(innnerOutline)):
        if outerOutline[i] != -1 and outerOutline[i] != innnerOutline[i]:
            return False
    return True
```

Bei einem erfolgreichen Vergleich wird die `while True:` Schleife abgebrochen und "randomSegment" wird an der Stelle eingefügt an der sich die x- und y-pointer befinden. Das geschieht mithilfe der Funktion insertSegment welche ebenfalls überprüft, ob das eingefügte Segment über den rechten oder unteren Rand der Stadt hinaus gehen würde und verhindert somit eine out-of-range Exception:

```
def insertSegment(self, city, segment, x, y):
        for i in range(y, segment.height + y):
            if i < len(city):
                for j in range(x, segment.width + x):
                    if j >= len(city[0]):
                        break
                    city[i][j] = segment.grid[i - y][j - x]
        return city
```

Falls der Vergleich jedoch nicht erfolgte, wiederholt sich die `while True:` Schleife ohne dem derzeitigem "randomSegment", jedoch überprüft der Straßensystem Generator zur Error-Vermeidung zuerst ob in der Liste currentSegmentList noch Elemente enthalten sind. Ist dies nicht der Fall (was eigentlich nicht vorkommen sollte), nimmt er ein zufälliges Segment aus der ursprünglichen Liste loadedSegmentList und breakt die `while True:` Schleife.

```
if len(currentSegmentList) <= 0:
  #pick any segment for now, shouldnt really happen
  rnd = r.randint(0, len(loadedSegmentList) - 1)
  randomSegment = loadedSegmentList[rnd]
break
```

3.4.5 Alternative Generatoren

Alternativ zu dem primären "Realistic" Straßensystem Generator gibt es auch noch den "Semi-Random" Generator, auf welchen in "4.4.2.2 Rekursion" eingegangenen wurde und den "Singular Road" Generator.

3.4.5.1 Singular Road Generator

Der Singular Road Generator generiert als einziger Generator eine Rechteckige Stadt, bei der die Breite immer fünf Felder sind und der Benutzer nur die Länge bestimmen kann. Dieser Generator ist dafür gedacht, dass man schnell Bilder aus der Stadt rendern kann, ohne gleich eine ganze Stadt rundherum mitzuladen.

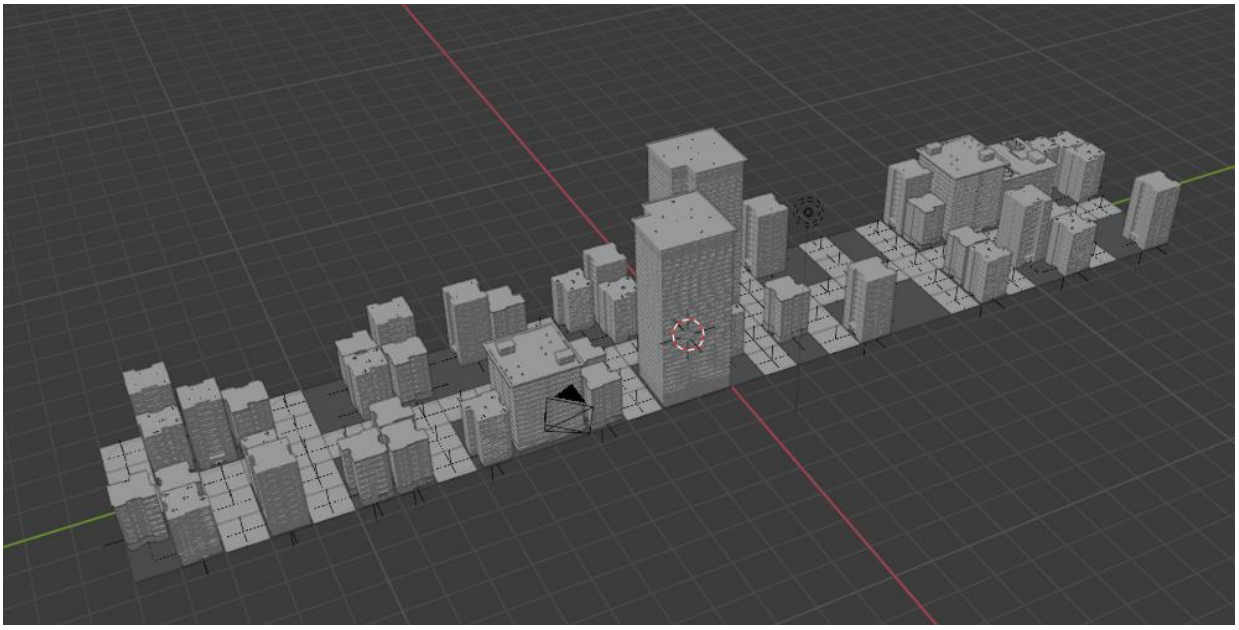


Abbildung 56: Beispiel Singular Road Stadt

Diese Städte haben immer eine lange gerade Straße in der Mitte, von der sich je nach dem benutzerdefinierten Wert "Density of side roads" mehr oder weniger Seitenstraßen abzweigen. Der Wert dieses Parameters geht ebenfalls wie Street Ratio von null bis drei wobei null ebenfalls wieder keine Seitenstraßen bedeuten würde und drei sehr viele Seitenstraßen.

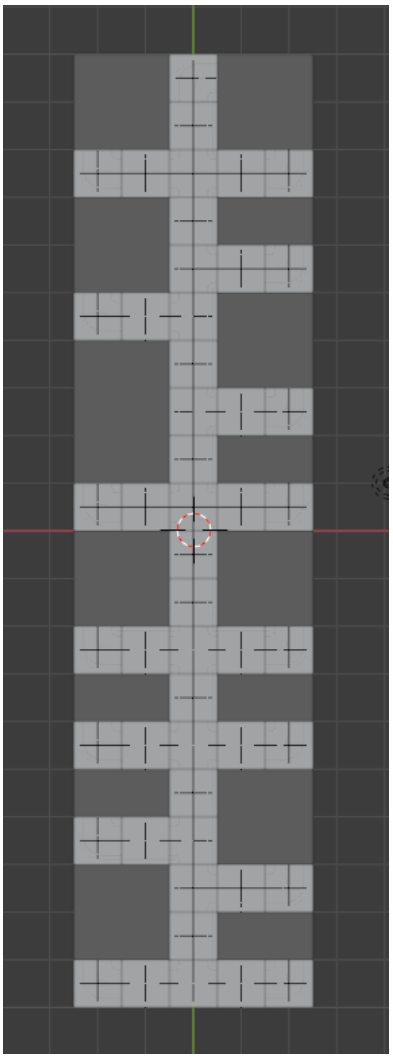


Abbildung 58: Density of side roads "3"

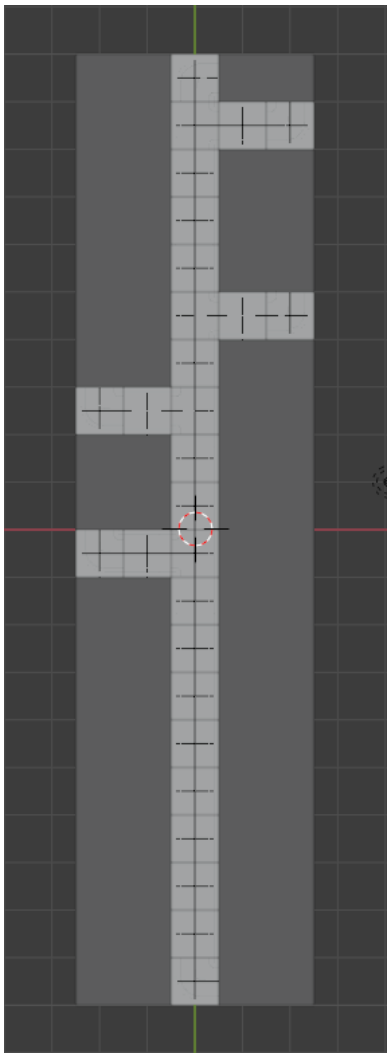


Abbildung 57: Density of side roads "1"

Wie alle anderen Generatoren beginnt der Singular Road Generator einen 2D-Array für die Stadt zu erstellen und and jeder Stelle mit dem Wert “0” zu befüllen:

```
w = 5
h = city_size
city = [[0 for x in range(h)] for y in range(w)]
```

Anschließend wird die Hauptstraße in der Mitte der Breite generiert:

```
#generate main road
            for i in range(h):
                city[2][i] = 2
```

Zuletzt wird noch der Wert von “Density of side roads” überprüft, und mit der dementsprechenden Wahrscheinlichkeit werden zuerst auf der linken, dann der rechten Seite Seitenstraßengeneriert. Dabei wird noch überprüft, ob die Reihe davor eine Seitenstraße enthält um zwei Seitenstraßen nebeneinander zu vermeiden und ebenfalls werden am Ende noch Index Errors abgefangen.

```
#left or right side
for i in range(2):
   #height coordinate
   for j in range (h):
         try:
               if (city[1+2*i][j-1] != 2 and city[1+2*i][j+1] != 2 and
         r.randint(0, 100) < junctionChance):
               city[1+2*i][j] = 2
               city[0+4*i][j] = 2
         except IndexError:
               continue
```

3.4.5.2 Semi-Random Generator

Der Semi-Random Straßensystem Generator beginnt ebenfalls damit, ein 2D-Array mit den Dimensionen der Stadt zu erstellen und an jeder Stelle den Wert "0" einzufügen.

Danach wird eine zufällige x- und y-Koordinate ausgewählt, um eine Zelle als Startpunkt für den Generator festzulegen.

```
roadXstart = r.randint(1, city_size)
roadYstart = r.randint(1, city_size)
```

Anschließend wird die rekursive Funktion createRoadNetwork in alle 4 Richtungen vom Anfangspunkt aus aufgerufen

```
#generate roads
city = self.createRoadNetwork(0, roadXstart, roadYstart, city, 1, 0,
city_size*5)
city = self.createRoadNetwork(0, roadXstart, roadYstart, city, 0, 1,
city_size*5)
city = self.createRoadNetwork(0, roadXstart, roadYstart, city, -1, 0,
city_size*5)
city = self.createRoadNetwork(0, roadXstart, roadYstart, city, 0, -1,
city_size*5)
```

Zuerst wird in der Funktion überprüft, ob sie entweder gerade in einer Straße oder außerhalb der Stadt aufgerufen wurde und wird dementsprechend beendet:

```
#hit a wall
if (x >= len(city) or y >= len(city) or x < 0 or y < 0):
   return city

if (city[x][y] == 2 and depth > 1):
   return city
```

Außerdem wird überprüft, ob die maximale Rekursionstiefe erreicht wurde:

```
if (depth >= maxDepth):
   return city
```

Falls all dies nicht der Fall ist, beginnt die Funktion damit, auf dem derzeitigen Standort eine Straße zu platzieren:

```
city[x][y] = 2
```

Anschließend wird mit einer 30-prozentigen Wahrscheinlichkeit eine neue Abzweigung in eine zufällige Richtung der momentanen Straße generiert. Da immer nur zwei der vier Richtungen infrage kommen, da die neue Abzweigung sofort unterbrochen wird, wenn sie in die Gegenrichtung generiert wird und mit einem if-Statement kontrolliert wird, ob es nicht die gleiche Richtung wie die jetzige ist, ist die reale Wahrscheinlichkeit für eine neue Abzweigung 15%. Zuletzt rufen die Abzweigung, falls vorhanden, und danach die eigentliche Straße die Funktion rekursiv neu auf:

```
if (r.randint(0, 100) < 30):
. . .
   if (not (directionX == newDirectionX and directionY == newDirectionY)):
      city = self.createRoadNetwork(depth + 1, x + newDirectionX, y +
   newDirectionY, city, newDirectionX, newDirectionY, maxDepth)

city = self.createRoadNetwork(depth + 1, x + directionX, y + directionY,
city, directionX, directionY, maxDepth)

return city
```

3.5 Gebäude Generator

3.5.1 Übersicht

Der Gebäude Generator, ist jener Generator, der die Position und Orientierung der Gebäude bestimmt. Es werden vier unterschiedliche Stufen der Gebäudedichte unterstützt sowie verschiedengroße Gebäudemodelle. Auch hier kommen Noise Funktionen zum Einsatz, da eine komplett zufällige Verteilung an Gebäuden optisch nicht besonders überzeugend wirkt. Durch Noise Funktionen lassen sich hingegen großräumige Strukturen erkennen, auf diese später genauer eingegangen wird.

3.5.2 Funktionen

Die primäre, durch den Generator unterstützte Einstellung, ist die Gebäudedichte. Im pseudo-zufälligen Modus des Generators entspricht die Gebäudedichte einfachgesagt der Prozentzahl der leeren Gitterfelder, die zu einem Gebäudefeld umgewandelt werden. So bedeutet eine Gebäudedichte von 0 eine Stadt komplett ohne Gebäude. Hingegen eine Dichte von 80 würde eine Stadt, die zu 80% aus Gebäuden besteht produzieren. Diese Einstellungsmöglichkeit gibt dem Benutzer die Möglichkeit genau zu kontrollieren wie viele Gebäude in einer Stadt existieren sollen.

Durch zahlreiche Tests und Versuche haben sich bei dem System allerdings mehrere signifikante Probleme ergeben. Einerseits ist die Möglichkeit, eine genaue Prozentzahl einzustellen nicht bedeutsam, da der Unterschied zwischen einer Dichte von 60% bzw. 65% nicht erkennbar ist. Diese unnötige Komplikation könnte manche User überfordern, da Änderungen manchmal eben nicht erkennbar wären. Andererseits sieht eine Stadt, dessen Gebäude per Zufall verteilt sind unrealistisch aus, da keine Makrostrukturen erkennbar sind.

Abbildung 59: Pseudo-Zufällige Gebäudeverteilung

Die Lösung für dieses Problem bietet der Realistische Gebäudegenerator. Dabei werden nur noch vier unterschiedliche Gebäudedichten angeboten, die allerdings deutliche sowie sinnvolle Unterschiede produzieren. Die zwei Extremwerte, also keine Gebäude, sowie überall Gebäude wurden beibehalten, allerding gibt es zwei weitere Einstellungen „Gebäudeinseln“ sowie „Gebäudeübergangsinseln“. Diese neuen Einstellungen ersetzen die aussagelose Prozentzahl.

Eine Insel ist eine Anhäufung von Gebäuden. Zwischen den Inseln erscheinen entweder keine oder nur sehr wenige alleinstehende Gebäude.

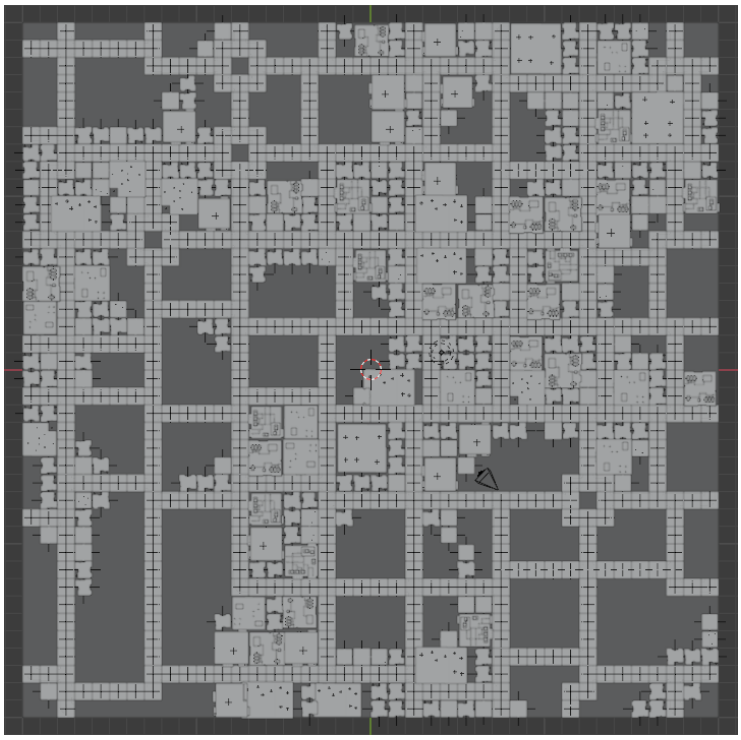


Abbildung 60: Gebäudestreuung mittels der VORONOI F1 Noise Funktion

Zusätzlich zu der Gebäudedichte, kann der Benutzer die folgenden Einstellungen kontrollieren:

- Zufällige Variation der Gebäudegröße
- Erscheinen der Gebäude ausschließlich neben Straßen
- Halbe Gebäudegröße (lässt die Straßen doppelt so breit wirken)
- Noise Funktion für die Streuung der Gebäude (10 verschiedene zur Auswahl)
- Skalierungsfaktor der Noise Funktion

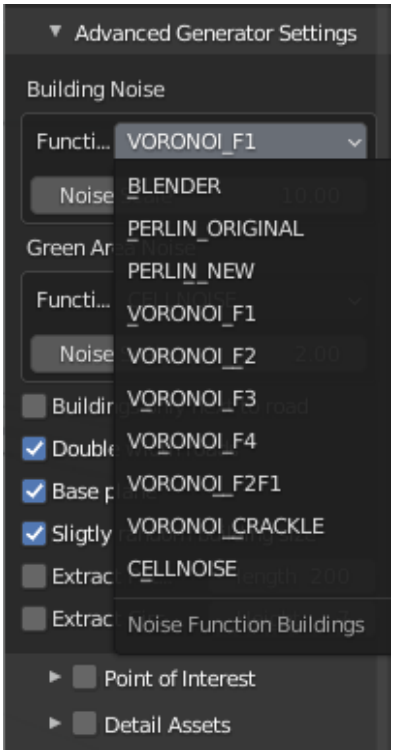


Abbildung 61: Auswahlmöglichkeiten für Noise Funktionen

3.5.3 Technische Aspekte

Entscheidend für die realistische Streuung der Gebäude ist das Anwenden von Noise Funktionen. Dadurch entsteht ein von weitem erkennbares Muster von Gebäudeanhäufungen und Leerflächen. Diese Leerflächen werden im weiteren Verlauf zu Grünzonen/Parks aus Büschen und Bäumen umgewandelt.

Je nachdem welche Gebäudedichte ausgewählt wird, kommt eine andere Formel für die Bestimmung der Gebäudeposition zum Einsatz. Nehmen wir als Beispiel die Dichte 1:

```
#generate building islands
if building_ratio is 1:
    for i in range(len(city)):
        for j in range(len(city[i])):
            if ((mathutils.noise.noise((j/noise_scale_buildings, i/noise_
scale_buildings, 0), noise_basis=noise_function_buildings)+1)*50 > 50 and
 city[i][j] != 2):
                city[i][j] = 1
```

Zuerst wird überprüft, ob die Gebäudedichte 1 eingestellt wurde. Wenn das der Fall ist, wird durch „city“, ein zweidimensionales Array, welches die komplette Stadt beinhaltet iteriert. Anschließend wird für jede mögliche Gebäudeposition eine Formel ausgeführt, die bestimmt ob an dieser Gitterstelle ein Gebäude erscheinen wird oder nicht. Dies geschieht, wenn der Noise Wert, an der jeweiligen Koordinate eine bestimmte Schwelle überschreitet.

Als Koordinate wird ein 3-dimensionaler Vektor erstellt, dessen X und Y Komponente der aktuellen Position im 2D Array entsprechen, so wird garantiert, dass die Muster der Noise Funktion auf die Stadt übertragen werden. Die Z Koordinate ist in diesem Zusammenhang irrelevant und wird aus diesem Grund auf 0 gesetzt. Nachdem der Noise Wert an der Position des 3D Koordinatenvektors bestimmt wird, vergleicht die Funktion diesen mit dem Schwellenwert 50. Theoretisch bedeutet das, dass man von einer 50% Besetzung aller möglichen Positionen für Gebäude ausgehen kann, dies ist nur wahr, wenn eine entsprechend große Stadt generiert wird.

3.6 Grünzonen Generator

3.6.1 Übersicht

Der Grünzonen Generator ist in CitySketch dafür zuständig, alle Wiesen, Bäume und Büsche an ihren Plätzen einzufügen. Er ist per Default deaktiviert und generiert bei der Aktivierung Grünflachen, je nach Einstellung, einerseits an leeren Stellen in der Stadt und andererseits in vom Benutzer mithilfe des Grease Pencils definierten Flächen.

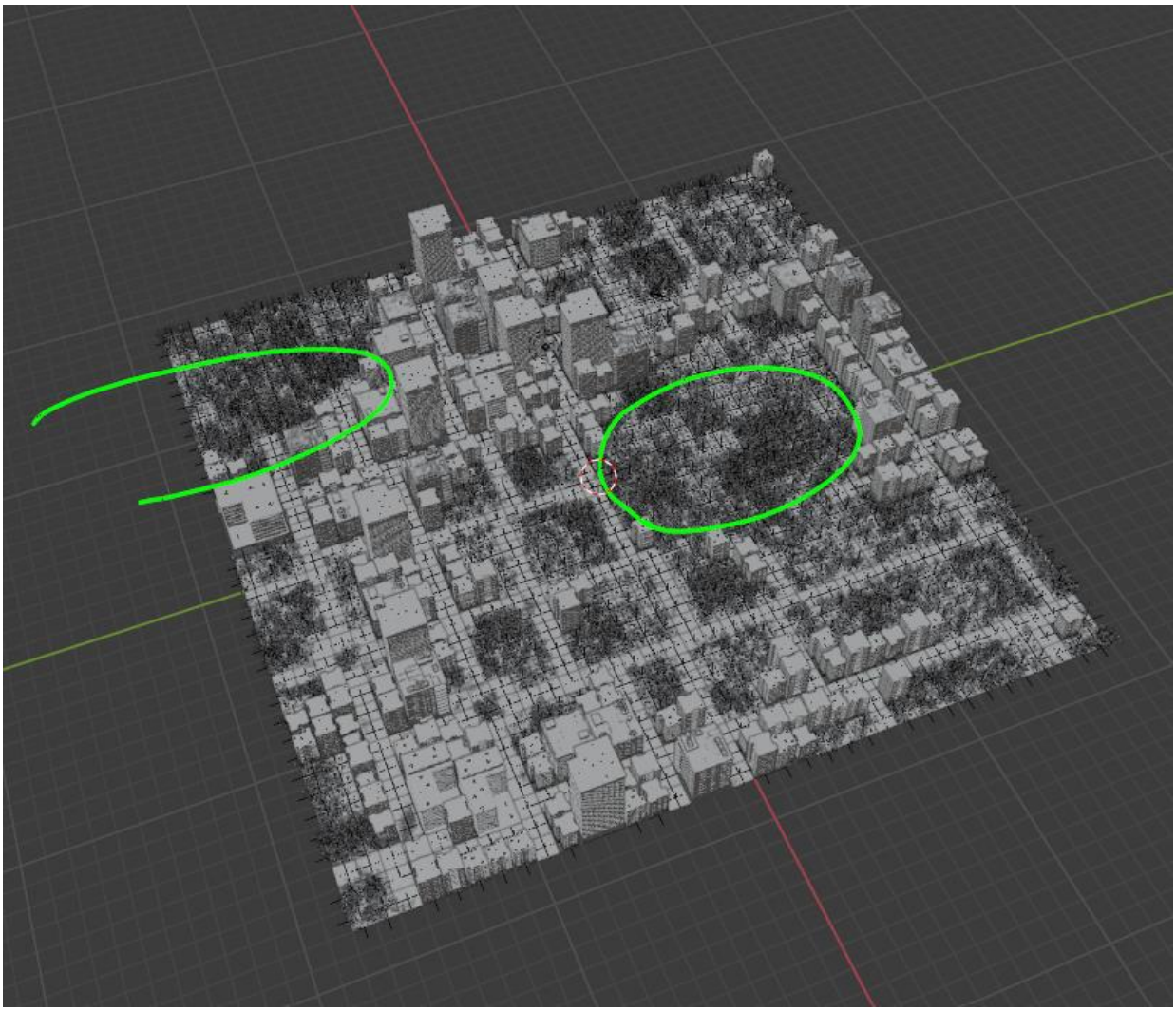


Abbildung 62: Grünflächen an leeren und eingezeichneten Stellen

3.6.2 Funktionen

Der Grünzonen Generator kann im CitySketch-Interface in Blender aktiviert und deaktiviert werden und man kann ein Dropdown-Menü aus- und einklappen.

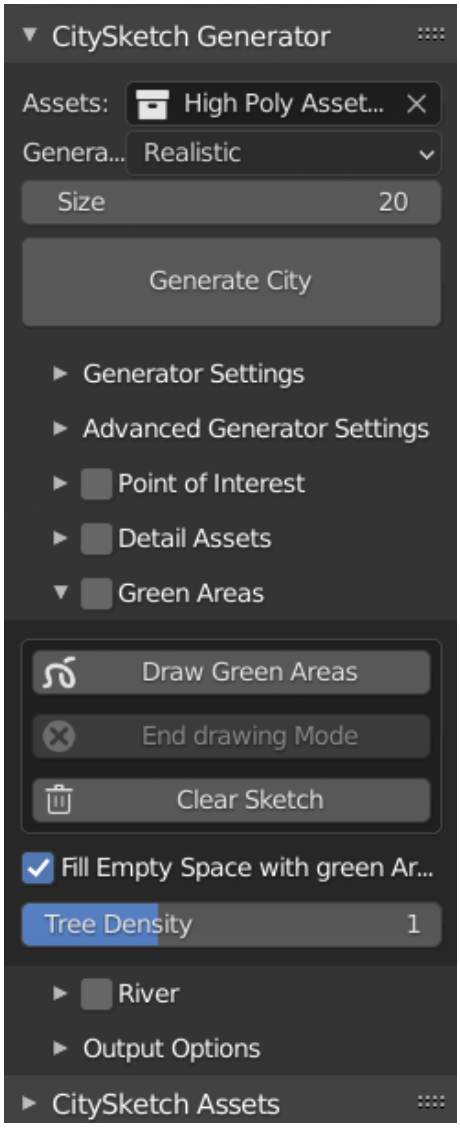


Abbildung 63: Ausgeklapptes Menü für den Grünzonen Generator

Im Dropdown Menü hat man einerseits die drei Optionen im Zusammenhang mit dem Grease Pencil welche auch bei Point of Interest und beim Fluss Generator zu sehen sind:

- "Draw Green Areas" => Der grüne Grease Pencil wird ausgewählt, welcher dafür verwendet wird, um Grünzonen in der Stadt einzuzeichnen.
- "End Drawing Mode" => Entfernt den Grease Pencil wieder und setzt den Benutzer in den normalen Cursor Modus zurück.
- "Clear Sketch" => Entfernt alle Zeichnungen, welche mit dem Grease Pencil für Grünzonen gemalt wurden.

Weiters kann man noch die Option auswählen, ob leere Stellen in der Stadt mit Grünzonen gefüllt werden sollen und zuletzt kann man noch eine "Tree Density" von null bis drei auswählen, wobei null keine Bäume und drei sehr viele Bäume wären.

3.6.3 Technische Aspekte

Der Grünzonen Generator beginnt damit, zu überprüfen, ob er aktiviert wurde und falls ja, ob eine Grease Pencil Zeichnung existiert und ob die Option, leere Flächen mit Grünzonen zu füllen, aktiviert wurde:

```
if generate_green_areas:

   #Process drawings if exists
   gp = bpy.data.grease_pencils.get("CitySketch")
   if gp is not None:
   . . .

   #fill empty space with green areas
   if fill_empty_green_area:
   . . .
```

Wie bei "4.2.4 Auswerten der Zeichnungen" bereits erklärt wurde, werden am Anfang die Zeichnungen mit dem Grease Pencil ausgewertet:

```
layer = gp.layers.get("Green Areas")

if layer is not None:
   #individual lines the user has drawn
   strokes = layer.active_frame.strokes
```

Anschließend wird mithilfe der for-Schleife `for stroke in strokes:` der Grünzonen Generator für jede eingezeichnete Grünzone des Benutzers aufgerufen.

In der Schleife werden die Koordinaten aller Punkte einer eingezeichneten Grünzone in einem Array gespeichert und dann mithilfe der von Shapely zur Verfügung gestellten Funktion `Polygon(x)` zu einem Polygon umgewandelt.

```
polygon_points = []

for stroke_point in stroke.points:
   polygon_points.append([stroke_point.co[0], stroke_point.co[1]])

polygon = Polygon(polygon_points)
```

Zuletzt wird noch durch alle Zellen in der Stadt durchgeloopt und es wird überprüft ob sich deren Mittelpunkte in dem gegebenen Polygon befinden und falls dies der Fall ist wird an dieser Stelle im 2D-Array der Stadt der Wert “5” eingetragen um hier eine Grünzone zu makieren:

```
for i in range(len(city)):
   for j in range(len(city[i])):

         point = Point(i-w/2+0.5, j-h/2+0.5)

         if polygon.contains(point):
               city[i][j] = 5
```

Nach diesem Schritt wiederholt sich das Ganze für jede weitere Zeichnung des Benutzers.

Um hingegen die leeren Felder mit Grünflächen zu befüllen wird lediglich, nach erfolgreicher Überprüfung, ob der Benutzer diese Option ausgewählt hat, jeder Wert im 2D-Array der Stadt, welcher auf “0” ist, mit einer “5” ersetzt.

```
for i in range(len(city)):
   for j in range(len(city[i])):
         if (city[i][j] == 0):
               city[i][j] = 5
```

3.7 Fluss Generator

3.7.1 Übersicht

Der Fluss Generator ist in CitySketch dafür zuständig, alle Flüsse an den vom Benutzer eingezeichneten Stellen einzufügen. Er ist per Default deaktiviert und generiert bei der Aktivierung Kanal-artige Flüsse entlang den vom Benutzer eigenzeichneten Stellen.

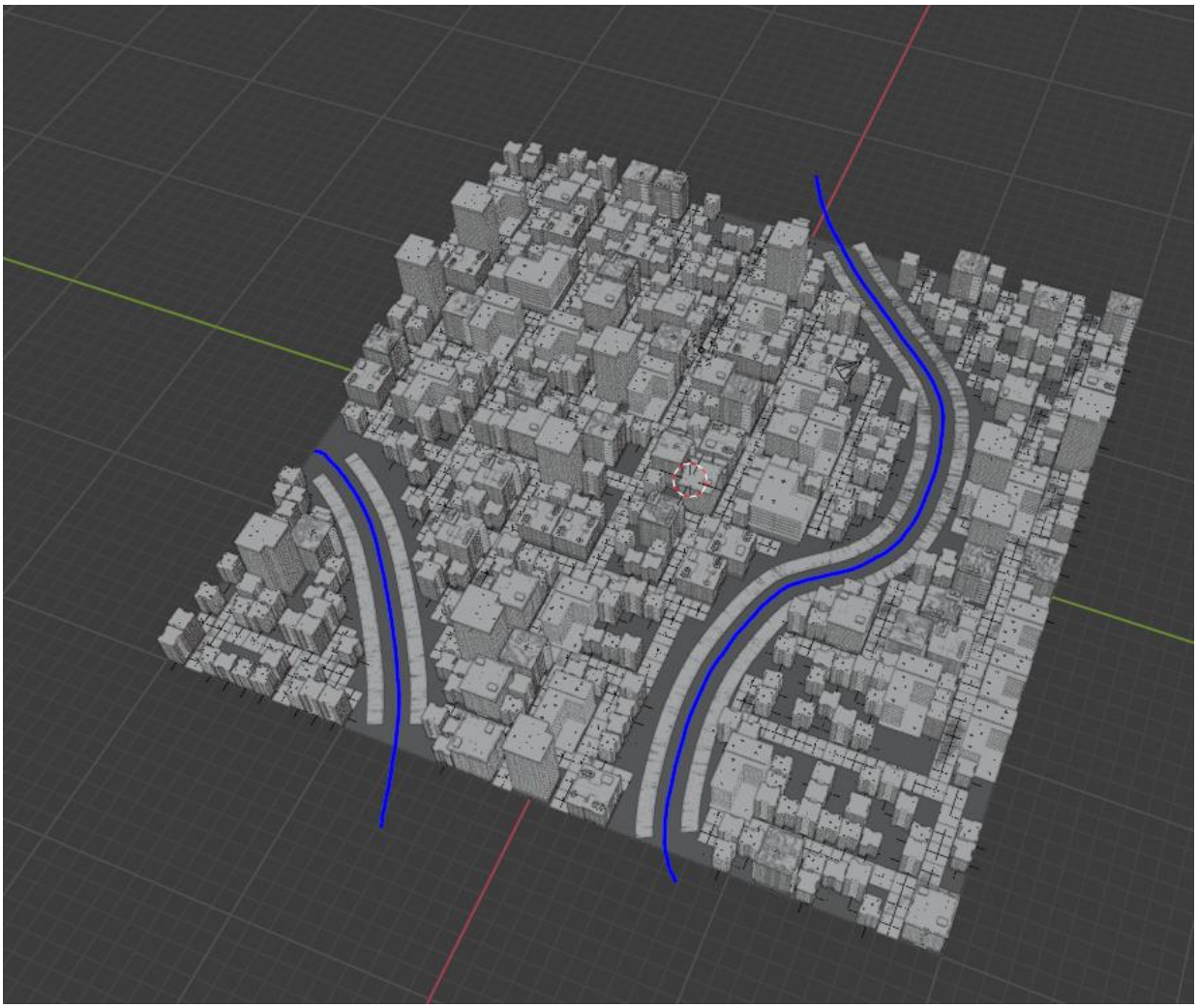


Abbildung 64: Flüsse entlang der vom Benutzer eingezeichneten Striche

3.7.2 Funktionen

Der Flussgenerator kann ähnlich wie der Grünzonengenerator im CitySketch-Interface in Blender aktiviert und deaktiviert werden und es kann auch wieder ein Dropdown-Menü mit weiteren Funktionen aus- und eingeklappt werden.

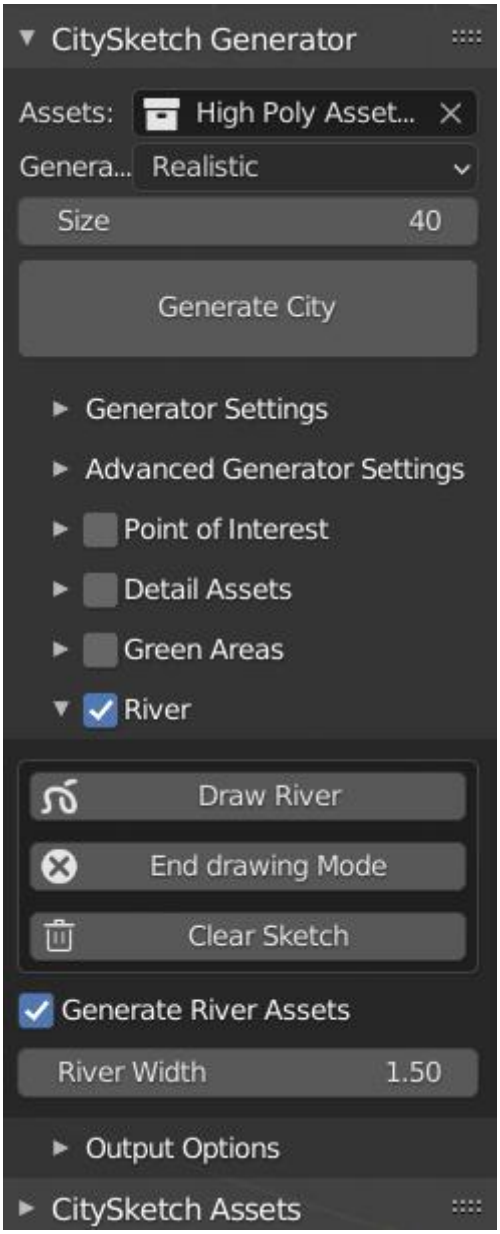


Abbildung 65: Ausgeklapptes Menü für den Fluss Generator

Im Dropdown-Menü hat man anfangs die gleichen drei Möglichkeiten im Zusammenhang mit dem Grease Pencil welche schon in 4.7.2 erklärt wurden mit dem einzigen Unterschied, dass der Benutzer bei “Draw River” nicht dem Umriss eines Flusses zeichnen soll, sondern stattdessen eine Linie, entlang welcher dann der Fluss generiert werden soll.

Ebenfalls kann der Benutzer durch das ein- und ausschalten der Option “Generate River Assets” entscheiden, ob die Assets für Flüsse mitgeneriert werden. Dadurch kann der Benutzer unter anderem selber die Gewichtung von Qualität und Performance in den gene-

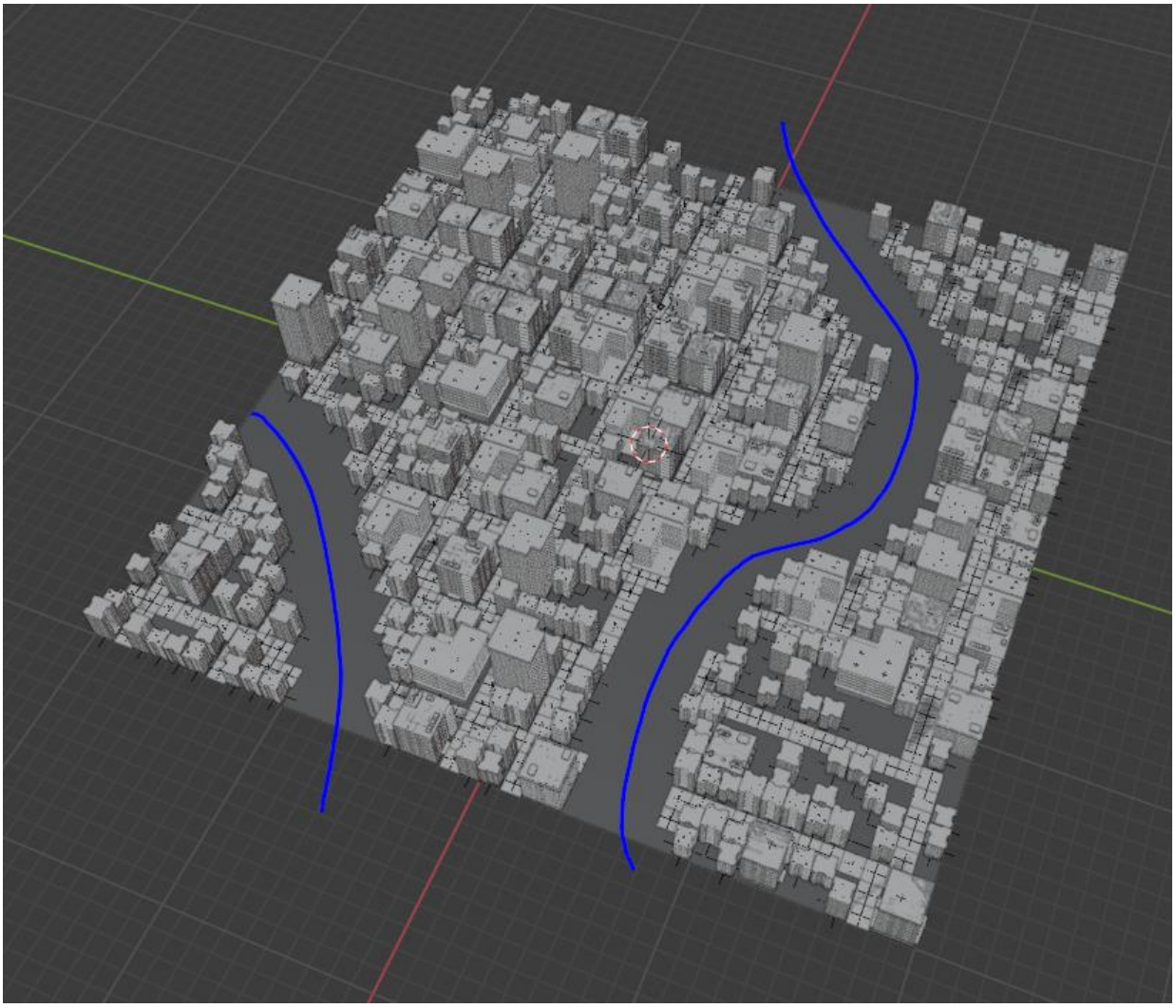


Abbildung 66: Beispiel Stadt ohne generierte Fluss-Assets

rierten Städten mitentscheiden.

Als letztes gibt es noch die Möglichkeit bei “River Width” die Breite des Flusses in Blender Units zu verstellen. Standardmäßig ist dieser Wert bei 1,5.

3.7.3 Technische Aspekte

Zuerst wird, wie beim Grünzonen Generator und Point of Interest, mithilfe eines einfachen if-Statements überprüft, ob der Benutzer die Option zur Generierung von Flüssen aktiviert hat. Dann wird, ähnlich wie bei Grünzonen Generator erwähnt und in “4.2.4 Auswerten der Zeichnungen” erklärt, die Zeichnung des Benutzers eingelesen:

```
gp = bpy.data.grease_pencils.get("CitySketch")
if gp is not None:
   layer = gp.layers.get("River")
   if layer is not None:

         #individual lines the user has drawn
         strokes = layer.active_frame.strokes
```

Auch hier wird wieder mithilfe einer for-Schleife der ganze folgende Prozess für jeden eingezeichneten Strich wiederholt. Zuerst wird überprüft, ob jeder Strich mindestens 2 Punkte hat, denn ist dies nicht der Fall, ist es unmöglich eine Kurve zu bilden und dementsprechend würde dieser Strich ignoriert werden.

Als nächstes wird eine neue Nurbs Kurve erstellt, da sich diese besonders dafür eignen, weiche Kurven aus vielen Punkten zu generieren.

```
crv = bpy.data.curves.new('Extracted River ' + str(index), 'CURVE')
                        crv.dimensions = '3D'


                        spline = crv.splines.new(type='NURBS')
```

Weiters wird für jeden Punkt überprüft, ob er sich in der Stadt befindet und falls ja wird jeder vierte Punkt in die eigentliche Kurve übernommen, um eine weichere Kurve zu generieren. Da eine Nurbs Kurve immer standardmäßig schon einen Punkt besitzt wird beim ersten Punkt der gezeichneten Kurve kein neuer Punkt zur Nurbs Kurve hinzugefügt und stattdessen die Koordinaten des ersten Punktes verändert.

```
first_point=true
for i, stroke_point in enumerate(stroke.points):
   point_x = stroke_point.co[0]
   point_y = stroke_point.co[1]


   if(abs(point_x) < city_size/2 + treshold and abs(point_y) < city_size/2
   + treshold):
         if first_point:
               first_point = False
         elif i % 4 == 0:
               spline.points.add(1)
         spline.points[-1].co = (point_x, point_y, 0, 1)
```

Im nächsten Schritt wird die Nurbs Kurve zu einem Blender Container Objekt umgewandelt, mit welchem Blender dann arbeiten kann. Dieses Objekt wird dann außerdem noch zur Szene gelinkt.

```
riverObj = bpy.data.objects.new('Extracted River ' + str(index), crv)
   riverObj.location = (0, 0, 0.02)
   generatedCollection.objects.link(riverObj)
```

Als nächstes wird überprüft, ob der Benutzer die Option, dass Fluss-Assets platziert werden sollen, ausgewählt hat und platziert dann dementsprechend die Assets. Die Idee ist, dass das Fluss-Asset nur eine dünne Scheibe ist und entlang der Kurve deformiert und platziert wird. Dazu wird das Asset geladen und es wird zuerst der Modifier "Array" mit dem Type "Fit_Curve" hinzugefügt, um das Asset so oft hintereinander zu platzieren, wie es Punkte in der Curve gibt. Danach wird der Modifier "Curve" hinzugefügt, um die Assets entlang der Kurve zu platzieren und deformieren.

Zuletzt werden noch einmal die Koordinaten aller Punkte des eigenzeichneten Striches in ein Array gefüllt und anschließend wird für den Mittelpunkt jeder Zelle der Stadt überprüft, ob er sich innerhalb einer gewissen Reichweite vom nächsten Punkt des Striches befindet. Falls dies der Fall ist, wird an dieser Stelle im 2D-Array der Stadt der Wert "6" eingetragen, um diesen Platz für Flüsse zu reservieren.

```
river_points = []
                        for stroke_point in stroke.points:
                            point_x = stroke_point.co[0]
                            point_y = stroke_point.co[1]
                            river_points.append([point_x, point_y])
                        river = LineString(river_points)


                        for i in range(len(city)):
                            for j in range(len(city[i])):


                                point = Point(i-w/2+0.5, j-h/2+0.5)


                                if(river.distance(point)                <=
river_width+0.6):
                                    city[i][j] = 6
```

3.8 Optimierung

3.8.1 Überblick

Beim Generieren von 3D Städten mit CitySketch wird oft mit Dateigrößen im Gigabyte-Bereich sowie Millionen von geometrischen Punkten, die insgesamt mehrere hundert Tausend Objekte ergeben gearbeitet. Der Prozess funktioniert nur aufgrund signifikanter Optimierungen und strenger Codevorschriften.

3.8.2 Instance Collections

Die wohl bedeutsamste Optimierung ist das Anwenden von sogenannten Instance Collections. Dies ist ein in Blender 2.8 (Anfang 2020) hinzugefügtes Feature, welches in erster Linie diese Diplomarbeit ermöglicht hat.

Instance Collections ermöglichen das Klonen von „Objektsammlungen“, ohne dass die Daten dupliziert werden, es entsteht lediglich eine neue Referenz, die auf dieselben Daten zeigt. Dies ist gut vergleichbar mit einem Pointer aus diversen Programmiersprachen, ebenfalls einer Referenz, die nicht auf die Daten, sondern auf deren Ort im Speicher zeigt.

Jedes einzelne 3D Objekt von CitySketch liegt in einer oftmals gleichnamigen Collection, die ausschließlich per Referenz instanziiert wird. So kann man das neue Instance Collection System maximal ausnutzen. Dadurch wird die kleinstmögliche Dateigröße erreicht und der Generierungsprozess um einen erheblichen Faktor beschleunigt. Ein Test hat gezeigt, dass die Generierzeit einer kleinen Stadt von knapp 2 Minuten auf unbedeutsame 2 Sekunden fällt, eine Verbesserung um den Faktor 60.

Instance Collections haben außerdem den Effekt, dass die Viewport Leistung bei hohen Anzahlen an gleichen Objekten (>10.000) hoch bleibt, da das Rendern dieser Collections gecached werden kann.

3.8.3 Rekursive vs. iterative Algorithmen

Ein ebenso wichtiger Aspekt ist die Anwendung iterativer Algorithmen. Dies beschleunigt einerseits die Generierzeit, schafft andererseits aber auch das Größenlimit für eine Stadt ab.

Die Rekursion ist ein mächtiges Programmierwerkzeug, mit dem viele Problemstellungen leicht lösbar sind. Jedoch besitzt dieses abgesehen von der Unlesbarkeit des Codes die dadurch entsteht auch zahlreiche andere Nachteile. Ausschlaggebend für unseren anwendungsfall ist allerdings die maximale Rekursionstiefe.

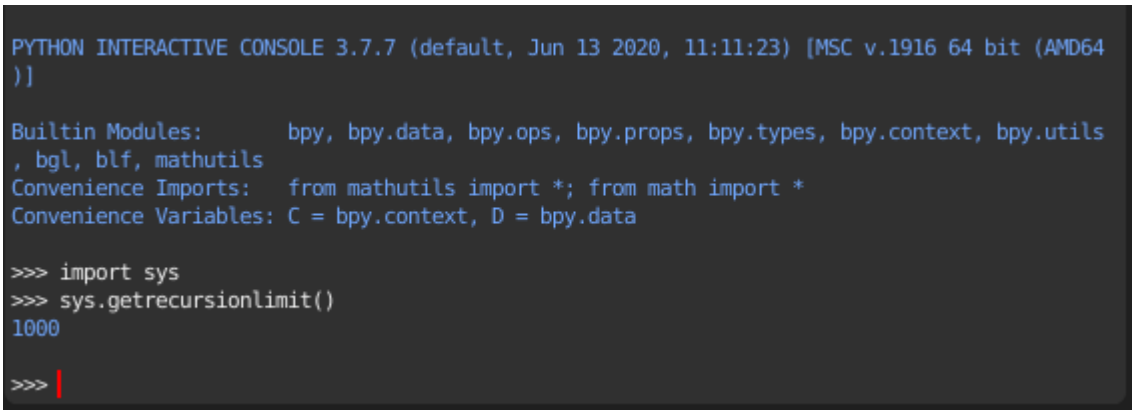


Abbildung 67: Rekursionstiefe der Blender Python Distribution bestimmen

Die Maximale Rekursionstiefe der in Blender mitgelieferten Python Distribution beträgt 1000 (ident mit der normalen Python Rekursionstiefe). Dies bedeutet, dass eine rekursive Funktion maximal 1000 Mal in sich selbst aufgerufen werden kann. Ab dem 1001 Mal wird ein „maximum recursion error“ hervorgerufen. Diese Rekursionstiefe ist zwar keine echte Hardwaregrenze und ließe sich mit dem Befehl „sys.setrecursionlimit()“ verändern, doch dann kann es leicht passieren, dass der Stack Speicher überfüllt wird, denn dieser muss alle bisherigen Rekursionsebenen gespeichert haben. Das Rekursionslimit ist also eine unüberwindbare Grenze, die im Fall von CitySketch, wie sich durch wiederholtes Testen ergeben hat, ab einer Stadtgröße von 150 Blender Units erreicht wäre.

Aus diesem Grund werden beim realistischen Generator ausschließlich iterative Funktionen und Algorithmen angewendet, besonders wichtig ist dies beim Straßensystemgenerator. Iterative Algorithmen können den Zustand der letzten Epoche verwerfen, da dieser zu der aktuellen geführt hat und somit nicht mehr gebraucht wird. Aus diesem Grund können iterative Operationen theoretisch unendlich lange laufen. Der Nachteil ist, dass dies bei manchen Problemstellungen zu unnötig komplizierten Lösungen führt, beziehungsweise schlicht unmöglich ist. (Shannon, 2019)

3.8.4 Viewport Sichtbarkeit

Eine weitere bemerkenswerte Optimierungstechnik, auf die wir während dem Testen gestoßen sind, ist die Viewport Sichtbarkeit der Blender Szene. Indem die Objekte zuerst unsichtbar gemacht werden, dann der Reihe nach instanziiert werden und im Anschluss gemeinsam wieder sichtbar gemacht werden, verbessert sich die Generierungszeit um das 6-Fache. Dies liegt daran, dass Blender im Hintergrund nicht mehrere tausendmal versucht den Viewport zu rendern, sondern nur einmal am Schluss.

Außerdem hat der Benutzer die Möglichkeit mittels einer Auswahlbox das Sichtbarkeit der generierten Stadt zu kontrollieren. Eine besonders praktische Einstellung ist die „hide output“ Option, die das Ergebnis im Viewport versteckt, beim Rendern allerdings wieder sichtbar macht. Dies erlaubt es, besonders große Städte zu generieren, denn die Sichtbarkeit der Objekte im Viewport ist die Haupteinschränkung bei großen Städten.

4 Projektmanagement

4.1 Überblick

Heutzutage haben sich mehrere Projektmanagement Methoden für softwarelastige Projekte etabliert. Man unterscheidet allerdings zwischen zwei verschiedenen Grundideen, die das Management betreffen. In der einen, etwas traditionelleren Variante wird die gesamte Planung vor der tatsächlichen Umsetzung durchgeführt. Diese Methode nennt sich Wasserfallmethode. Ein modernerer Ansatz sind Agile Methoden, diese Methoden setzten auf zahlreiche begleitende Prozesse und kontinuierliche Anpassungen.

„Theoretisch ist Wasserfall eine perfekt logische Methode, denn es wird alles genau geplant, sodass nur noch ein detaillierter Plan befolgt werden muss. Allerdings gibt es eine große Schwäche: Menschen sind involviert.

Oft kommt es vor, dass gute Ideen erst im Laufe des Projektes entstehen, um diese berücksichtigen zu können, braucht man einen anderen Ansatz.“
(Deemer, Benefield, Larman, & Vodde, 2010)

Besonders bei Softwareprojekten können sich Requirements im Laufe des Projektes ändern, um das Produkt besser an den Markt anzupassen. Es ist also kein Wunder das Agile Projektmanagementmethoden in diesem Zusammenhang bevorzugt werden.

Scrum ist eine dieser Methoden, die vor allem auf kontinuierliche Produktinkremente, häufiges Feedback und gemeinsame Entscheidungen setzt. (Sliger, 2011)

Diese Diplomarbeit wurde mithilfe der Scrum Methodik durchgeführt. Scrum ist ein begleitender Prozess und eignet sich optimal für Projekte dessen exakte Durchführung nicht definierbar ist, da sie von veränderlichen Faktoren abhängt. Bei CitySketch hatten wir eine genaue Vorstellung der Ziele und konnten diese auch präzise definieren, die Lösungsansätze blieben allerdings oftmals vage beschrieben, denn das Erfahrungsniveau, um zu entscheiden, wie ein gewisses Problem konkret gelöst werden soll hat gefehlt. Zum Beispiel kannten wir mögliche Vorgehensweisen beim Erstellen der Straßen, wir wussten es ist möglich die Straßen als eine Blender Bezier Kurve zu behandeln, als Pfade, als gesamte Objekte sowie als Anordnungen aus verschiedenen Puzzlestücken, konnten aber nicht sagen welcher von den Lösungsansätzen der Beste ist. Durch Scrum war es uns möglich in den frühen Phasen des Projektes mit den Lösungsansätzen in Form von Prototypen zu „experimentieren“ und anschließend diesen Prototypen Sprint für Sprint inkrementell zu verbessern. Bei der Management Methode Wasserfall, ist ein derartiges inkrementelles Experimentieren und Verbessern nicht möglich, da die genaue Lösungsweise bereits vor Beginn der Umsetzung bekannt sein muss.

4.2 Rollen

Ausschlaggebend für Scrum sind die drei Rollen Scrum Master, Product Owner sowie das development Team.

❖ Scrum Master

Der Scrum Master sorgt dafür, dass die Scrum Methodik im Projekt von allen korrekt durchgeführt wird. Das Einhalten der Scrum Vorgaben und anwenden der Prozesse lässt das Projekt möglichst effizient vorankommen und minimiert das Risiko von Abweichungen, beziehungsweise alarmiert früh genug wenn etwas schiefgehen sollte. Erst wenn jedes Mitglied des Projektes mit den Scrum Vorgängen vertraut ist und sich an diese hält, können die Vorteile einer agilen Entwicklungsweise ausgenutzt werden, dadurch kann der Projektablauf optimal und effizient verlaufen. Es ist also wichtig, als Scrum Master, alle Mitglieder in das Scrum System einzubeziehen und falls notwendig auf die Methodik einzuschulen. Oft kann dies auch über externe Kurse und Trainer vor dem Projektstart geschehen.

Während dem Projekt achtet der Scrum Master vor allem auf das Einhalten der Scrum Ereignisse und Wohlgefühl der Mitarbeiter. Der Scrum Master soll nicht das Projekt leiten, sondern dem Projekt dienen und dieses währen der gesamten Laufzeit begleiten.

Außerdem stärkt der Scrum Master die Kommunikation des Teams und regt häufige Gespräche an, um Probleme einzelner Mitglieder aktiv zu diskutieren, denn dies führt zu möglichst schnellen Problemlösungen, oder lässt das Team unüberwindbare Barrieren früh genug erkennen, um an anderen Möglichen Strategien zu arbeiten. Entscheidend dafür ist der sehr wichtige Informationsfluss zwischen mehr und weniger erfahrenen Mitarbeitern der oft von allein nicht entsteht.

❖ Product Owner

Der Product owner kennt die Ziele des Projektes und weiß wie das Ergebnis maximal und optimal genutzt werden kann. Der Product Owner kommuniziert die Ziele und Anforderungen an das Team und ist verantwortlich für das dadurch entstandene Ergebnis. Er muss auf die Transparenz und Aktualität der Ziele achten, dies geschieht in Scrum über den Product Backlog. Die Ziele kommen entweder vom Product Owner direkt oder einer überstehenden Person. Im letzteren Fall, gilt der Product Owner als informierter Mittelmann zwischen dem Kunden und dem Team und musst sich mit beiden Seiten intensiv beschäftigen um sich in diese Versetzen zu können. Falls notwendig schlägt der Product Owner Veränderungen vor. Im Vordergrund steht immer die Diskussion mit dem erfahrenen Team, um den größten Wert aus den Anforderungen schöpfen zu können.

❖ Das development Team

Das Team besteht aus den Entwicklern, die die Pläne zum Ergebnis führen müssen. Die Entwickler sind Profis in Ihren Gebieten und sind für Ihren Teil verantwortlich. Zu den Aufgaben des Teams zählt das Erstellen und Planen der Sprints unter Einbeziehung ihrer Expertise, sowie das Lernen und Ziehen von Entschlüssen aus vergangenen Sprints.

4.3 Artefakte

Artefakte sind Entitäten, die wichtige Informationen zur Beurteilung der „Gesundheit“ des Projektes darstellen. Diese Informationen werden genutzt, um notwendige Entscheidungen zu treffen und das Projekt an die vorherrschende Situation bestmöglich anzupassen. Nur wenn die Artefakte nach den Scrum Richtlinien ausgearbeitet werden, kann garantiert werden das die abgelesenen Informationen wertvoll sind. Artefakte bieten die Möglichkeit unkompliziert Kerninformation über das Projekt zu ermitteln, die statistisch gesehen im Großteil der Projekte relevant sind.

❖ Product Backlog

Der Product Backlog ist eine Sammlung an Zuständen die notwendig ist, um das Produkt zu Verbessern und zum Ziel zu führen.

❖ Product Increment

Ein Product Increment ist die nächste, verbesserte Version eines Produktes. Dieses wird im Laufe des Sprints verbessert und durch neue Features erweitert, sodass am Ende ein erkennbares Inkrement existiert.

4.4 Management Tools

Im Verlauf dieser Diplomarbeit wurden diverse Tools zum Verwalten gewisser Aspekte des Managements verwendet.

❖ Toggl Track (https://toggl.com/track/)

Eine Webanwendung für das mitprotokollieren des Zeitverbrauches. Es werden unterschiedliche Diagramme und Ansichten zur Beurteilung der Zeitaufteilung angeboten, sodass Abweichungen früh erkennbar sind und gegebenenfalls angemessen gehandelt werden kann.

❖ Taiga (https://taiga.io/)

Eine Webanwendung die das Verwalten eines Backlogs erleichtert. Es gibt die Möglichkeit alle User Stories inklusive der Tasks in der Anwendung zu definieren und anschließend den jeweiligen Sprints und Personen zuzuordnen. Eine Analyse der Sprint Velocity informiert den Scrum Master rechtzeitig über potenzielle Hindernisse und Hemmungen.

❖ Sharepoint (sharepoint.com)

Ein Intranet zum Verwalten von Dokumenten und Dateien innerhalb eines Projektteams. Die Zugriffkontroller ermöglicht das Steuern der Datenvisibilität für bestimmte Teammitglieder.

❖ Github (github.com)

Ein Codeverwaltungstool zum gemeinsamen bearbeiten eines Code Repositories.

4.5 Risiken

5.2 Risikoportfolio

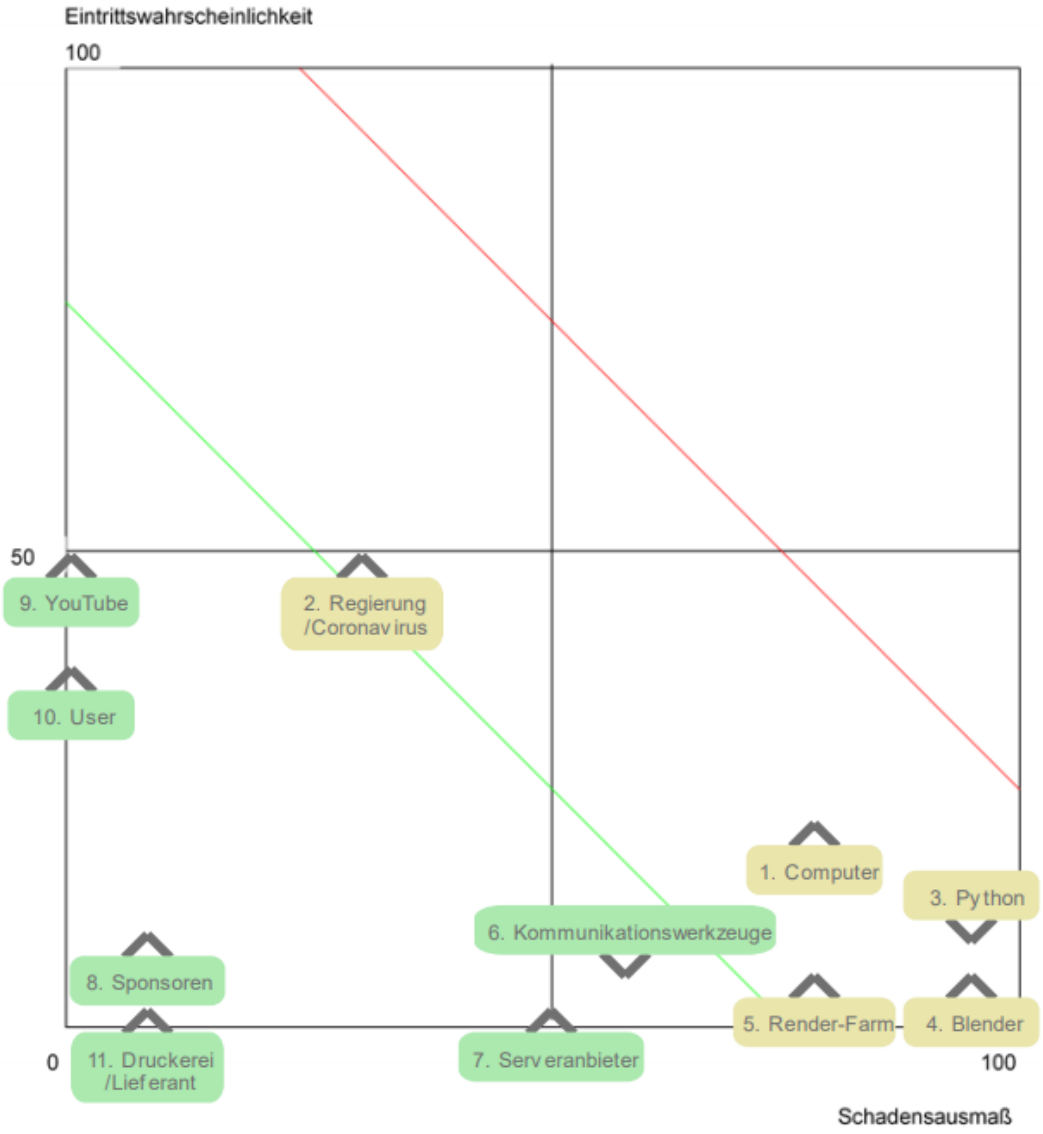


Abbildung 68: Risikoportfolio für CitySketch

In der Planungsphase des Projektes wurden die wichtigsten Risiken in einem Risikoportfolio verschriftlich und deren Eintrittswahrscheinlichkeit wurde abgeschätzt. Wie erwartet

sind einige dieser Risiken im Laufe des Projektes eingetreten und entsprechende Maßnahmen mussten gezogen werden. Das Höchstbewertete Risiko, die unzureichende Leistungskraft der uns zur Verfügung stehenden Computer für das Rendern des Product Videos ist eingetreten. Wie geplant konnte allerdings eine Kooperation mit unserem Sponsor arrangiert werden. Ein starker Rendercomputer wurde bereitgestellt und hat in einem Prozess, der mehrere hundert Stunden gedauert hat, das Rendern der Szenen übernommen. Dies ist nur möglich gewesen, da vor dem Projekt eine konkrete Vorgangweise definiert wurde, was geschehen soll, wenn das Risiko eintritt. Dies wurde ebenfalls mit dem Sponsor besprochen. Große Unternehmen haben oft die Möglichkeit Arbeitsmitteln bereitzustellen, sofern dies rechtzeitig besprochen, mittgeteilt und verschriftlicht wird, denn in solchen Unternehmen kann nicht spontan auf Ereignisse reagiert werden.

5 Marketing

5.1 Product Video

5.1.1 Übersicht

Wir haben das Product Video in drei Teile unterteilt, dem Intro, Hauptteil und Outro. Das Intro ist ein „Showreel“, um die Möglichkeiten und Qualität des Addons vorzustellen. Darin werden Shots von den Gebäuden, sowie von den Point of Interest Gebäuden und Grünzonen gezeigt. Im Hauptteil werden die einzelnen Schwerpunkte der Diplomarbeit gezeigt. Die Funktionen und Asset Packs werden nacheinander in einer Form von Breakdown und Beispiel gezeigt. Zum Schluss im Outro kommen die „End Credits“, während die Kamera langsam aus einer Stadt heraus zoomt.

5.1.2 Szene vorbereiten

Für das Setting der Shots werden HDRIs (High Dynamic Range Imaging) verwendet. HDRIs sind 360° Panorama Fotos von Landschaften oder Städten. Normalerweise werden dafür Bilder von Orten mit einer guten Belichtung geschossen. Dadurch lässt sich die Szene leichter beleuchten. Außerdem dienen sie ebenfalls als ein Hintergrund.

5.1.3 Post Production

Für das Post-Processing des Videos wird die kostenlose Software Davinci Resolve verwendet. Die gegenüber anderen Videoschnittprogramme ein „All-in-One“ Packet mit den wichtigsten Programmen für die Videoproduktion anbietet. Mit dieser Software kann man Colorgraden, Schneiden, „Compositing“ machen und Audios bearbeiten.

5.2 Social Media Accounts

Eines der größten Faktoren und die Garantie für den Erfolg des Add-ons ist die Verbreitung und die Steigung des Berühmtheitsgrades vom CitySketch. Letzten Endes ist es ein Ziel für uns, unser Add-on auf den sogenannten Blender Market anbieten zu können. Wir wollen mithilfe von Social Media unser Produkt präsentieren und zukünftigen CitySketch Nutzern erklären, welche Features beinhaltet sind und welche Ergebnisse zustande kommen können. Unser Fokus liegt auf der App Instagram, einer der weltweit berühmtesten Onlinedienste, um ein Privat- oder Unternehmens-Profil zu erstellen und mit diesem, für seine Produkte oder seine Marke zu werben.

Auf unserem Profil laden wir seit der Erstellung des Instagram-Kontos Bilder hoch. Man erkennt auch die Verbesserung und die Entwicklung des Projektes.

Abbildung 69: Instagram Account von CitySketch

Uns war bewusst, dass wir mithilfe der Webseite alleine nicht viel mehr Aufmerksamkeit bekommen konnten Daher haben wir viele verschiedene Arten von Städte im Low- und Highpoly Stil generiert und diese auf unserem Profil hochgeladen. Um das Interesse des Betrachters zu wecken zeigen wir auch die Features wie die Road Generation, Detail Assets (unser eigenes Logo als Schildtafel auf den Gebäuden, Straßenschildern, Grünzonen) sowie den Fluss Generator die CitySketch so einzigartig machen, um auch von der Konkurrenz heraus zu stechen.

Auf dem YouTube Kanal CitySketch befindet sich ein Step-to-Step Tutorial auf Deutsch von einer älteren Version, wo erklärt wird, wie CitySketch funktioniert und wie in Sekundenschnelle eine Stadt generiert werden kann.

5.3 Gerenderte Bildergalerie

Die Bildergalerie zeigt die hohe Qualität der Städte bzw. des Addons. Hierbei wurde mit einer ähnlichen Arbeitsweise wie beim Product Video gearbeitet. Im Gegensatz zum Video kann man in einem Render das Niveau der Gebäude genauer darstellen.

5.3.1 Szene vorbereiten

Wie auch beim Product Video wurde für die Beleuchtung ein HDRI Bild im Hintergrund verwendet. Allerdings wird die Szenze ohne Hintergrund gerendert. Dafür muss man in dem „Scene Property"-Fenster die Transparenz unter „Film" auswählen.

5.3.2 Post Production

Die Renderings werden in Photoshop bearbeitet. Zu dem Bild wird eine beliebige Himmel Textur hinzugefügt, die mit „Curves" und „Hue and Saturation" an das Farbschema von dem Gebäude-Render bearbeitet wird. Je nach Szene wird zusätzlich eine „Depth Map" über alle Layers gelegt. Die „Depth Map" bekommt Blender, wenn man im „Compositing"-Fenster den Z-Input rendert. Dieser repräsentiert (in Schwarzweiß) die Bildtiefe und wird oft als Nebel oder für Tiefenschärfe verwendet.

Abbildung 70: Gerendertes Bild mit CitySketch

5.4 Product Website

5.4.1 Übersicht

Die Product Website dient dazu, unseren interessierten Kunden aus der Blender Community das Add-on CitySketch zum Download anzubieten. Neben dieser Möglichkeit bieten wir dem Besucher mehr über uns zu erfahren.

Bei der Eingabe unseres Links getcitysketch.com wird man auf unsere Startseite geleitet. Das erste Merkmal wird der sich bewegende Hintergrund sein, der die Werke und Erarbeitungen von Citysketch zeigt. Im Vordergrund befindet sich dann eine E-Mail-Eingabe, wo man hier den Newsletter für aktuelle Neuigkeiten abonnieren kann. Darunter befinden sich unsere Social Media Profile mit einem Copyright Abzeichen. Wenn auf “Find out more!” gedrückt wird, kommt man auf die eigentliche Homepage.

Auf der Homepage wird erklärt was die Idee und die Nutzung von Citysketch ist und was die Blender Erweiterung alles anbietet. Wir möchten den Nutzern darauf hinweisen, dass das Add-on hochqualitative und selbsterstellte Modelle anbietet. "CitySketch” soll bedeuten, dass dem Benutzer frei überlassen ist sich seine eigene Stadt zu designen.

Scrollt man hinunter, werden die Low- und High Poly Packs anzeigt. Man sieht eine generierte Stadt in dem jeweiligen Stil, inklusive Erklärungen über den Inhalt der Asset Packs und wo man sie anwenden kann. Es wird auch vorgewarnt, dass die Renderzeiten beim High-Poly Pack deutlich länger andauern können als beim Low-Poly Pack.

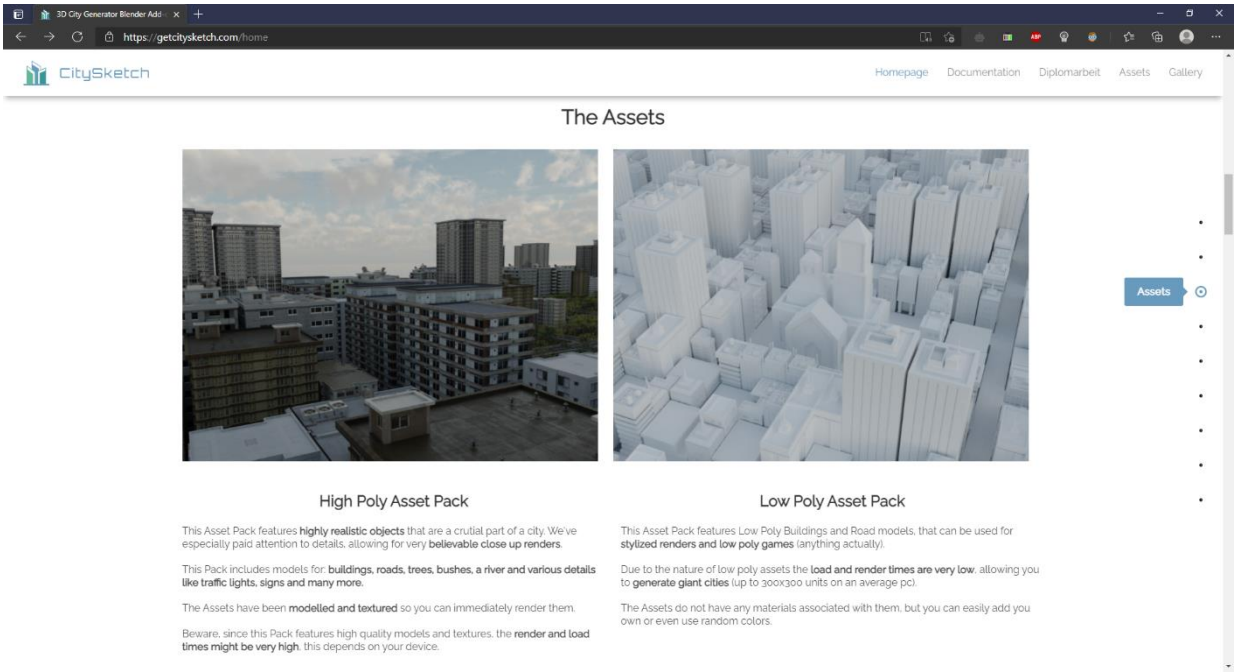


Abbildung 71: Vergleich der verfügbaren Asset Packs

Als nächstes erklären wir dem User Schritt für Schritt was genau gemacht werden muss, um eine Stadt mit ohne jegliche Bearbeitungen bzw. Erweiterte Einstellungen zu generieren. Beim Öffnen des Add-on muss ein Asset Pack importiert werden. Wie bereits erwähnt wird von uns 2 Arten der Assets zur Verfügung gestellt. Man kann natürlich auch seine eigenen importieren. Wichtig: Die Assets müssen in Blender funktionieren.

Wir gehen nochmal etwas genauer in den Punkt “Turn Sketches into Cities” ein und haben beim nächsten Punkt eine Grafik, die das Bild einer fertig generierten Stadt in der Vogelperspektive und die Zeichnung des Sketches. Das Zeichnen soll den Spaßfaktor beim Erstellen der Stadt erhöhen, dadurch dass freihändige Tätigkeiten meist offener und kreativere Wirkungen haben können als Rastern und Linien. Flüsse, Grünzonen und Point of Interest können selber gezeichnet und platziert werden.

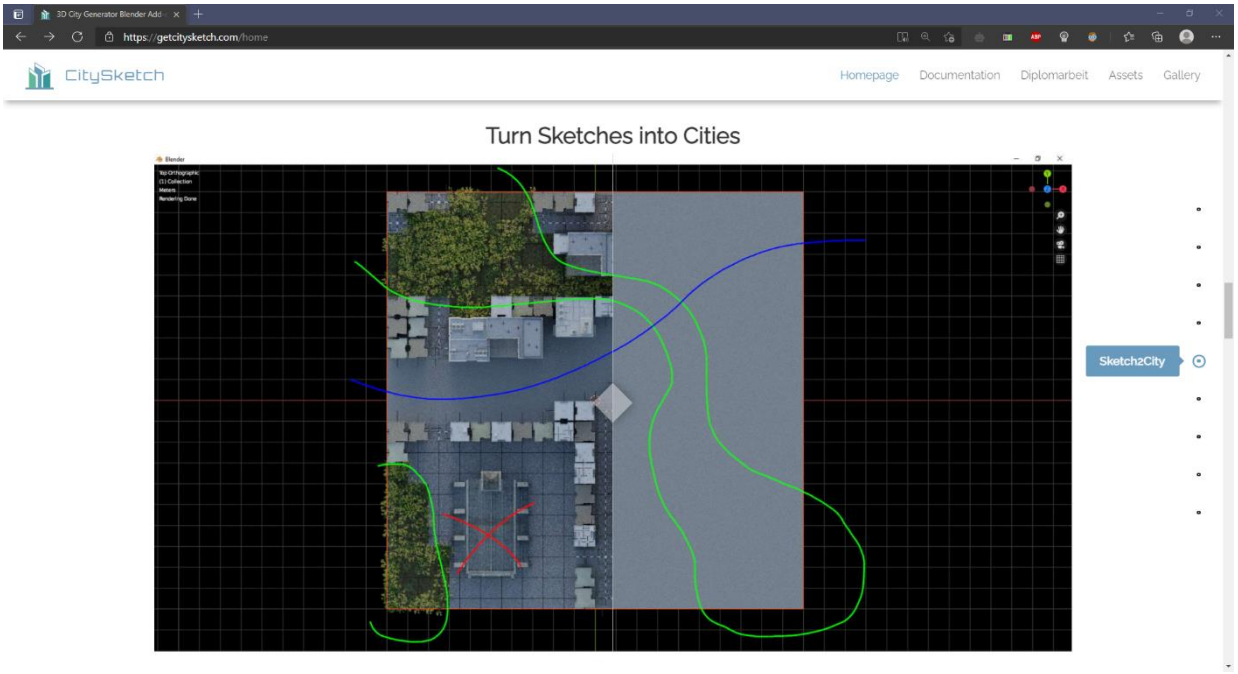


Abbildung 72: Before/After Vergleich des Generierens mit Zeichnugnen

Neben dem Sketch erklären wir das Grundprinzip unseres Advanced Road Systen. Wir achten auf benutzerfreundliche Einstellungen und haben daher 2 Optionen für das Straßensystem zusammengefasst: Symmetry oder Density. Insbesondere ist es umso wichtiger, dass das Straßenmuster logisch korrekt generiert wird (das bedeutet keine abgehackten Wege bzw. Sackgassen, sondern richtig entlangfahrende Pfade). Auch hier stellen wir selbsterstellte Road Assets zur Verfügung, die im Add-on beinhaltet sind. Natürlich kann man sowie bei den Gebäuden auch bei den Straßen seine eigenen importieren.

Eine Idee wo CitySketch seinen Nutzen genauso finden könnte wäre im Bereich 2D. Eine Stadt in der Vogelperspektive von “Top to down” bietet eine Möglichkeit für 2D Videospiele oder auch Architektenpläne. Hierbei rendert man die Stadt von oben.

5.4.2 Blogposts

Letzten Endes kommen wir zum CitySketch' Blog. Der Blog ist ein Art Tagebuch und beinhaltet Artikeln und Fortschrittsmitteilungen über das Projekt von null bis jetzt. Jeder Blog Eintrag umfasst einen Sprint, den wir gemacht haben, welches 3 Wochen entspricht. Im ersten Blog handelt es um Änderungen des Zieles und der Vision. Anfangs waren wir uns über das Konzept des Zeichnens nicht sicher. Der Grundgedanke dahinter war es auch die Positionen der Straßen und Gebäuden selber zu platzieren. Daher haben wir uns entschieden mit einem Grid zu arbeiten, was bedeutet, kein Freies Zeichnen, sondern nur nach Koordinaten.

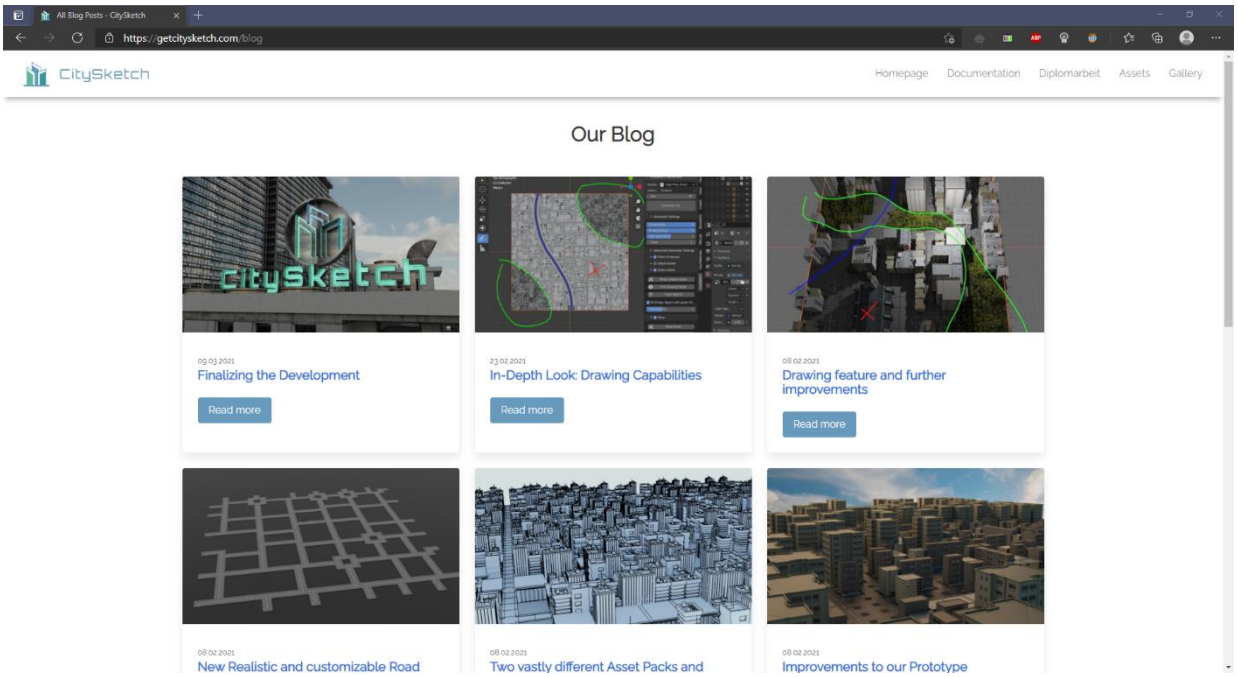


Abbildung 73: Übersicht aller Blogposts

Um das neue Ziel umzusetzen erstellten wir einen Prototyp, um den Proof of Concept zu bestätigen. Die ersten Blogs beinhalten Mitteilungen über die Planung, sowohl als auch um die Erweiterungen die neuen Prototypen mithilfe von Features. Nebenbei wurde mit der Erstellung der Asset Packs, Low- und High Poly begonnen. In den darauffolgenden Artikeln kam es zu Verbesserungen des Straßensystems. Es war nun möglich verschiedene Optionen in CitySketch einzustellen, von der Dichte bis hin zu der Symmetrie (Wie viele Gebäude? 2 Blockhäuser oder jedes Gebäude getrennt?). In der darauffolgenden Zeit wollten wir wieder die ursprüngliche Idee für CitySketch probieren und haben einen Alternativweg umsetzen können: Jetzt war es möglich, Flüsse, Grünzonen und den Point of Interesst per Hand zu zeichnen. Nicht nur die Generierzeiten wurden optimiert, sondern auch das eigentliche Konzept vom Add-on wurde wieder ins Leben gebracht. Im letzten Blog und somit auch im letzen Sprint bereiten wir uns auf die erste offizielle Veröffentlichung des Add-ons. Hierbei

ist es umso wichtiger unsere Profile auf Social Media mit guten Resultaten zu befüllen, um unsere Community zu vergrößern.

Am Ende des Projektes wurde auch unser Product Video auf die Startseite hochgeladen, welches auch auf YouTube unter unserem Kanal zu finden ist.

Auf der Webseite befindet sich auch eine Online-Dokumentation über CitySketch. Hier im Overview erfahren die Besucher über die benötigten Ressourcen und die Schritte die durchgeführt werden um CitySketch in Blender vewenden zu können. Ein wichtiger Tipp: Bei der finalen Version des Add-on wird eine Blender Version von 2.9 oder höher benötigt.

Es wird Schritt für Schritt angeboten vom Download einer .zip Datei bis hin zum Importieren in Blender. (Blender Erweiterungen werden beim Downloaden auf einer Website oder im Internet nicht direkt in Blender eingebunden. Diese müssen bei den Einstellungen nochmals manuell vom Explorer importiert und aktiviert werden! Außerdem darf die .zip Datei keinesfalls umbenannt werden, da im Code nur der Name CitySketch.zip und kein anderer erkannt wird).

Im Bereich “Example” erklären wir wie eine einfache Stadt generiert wird und welche Grundeinstellungen von Größe, Stadtdichte bis hin zum Einzeichnen des POI, Flüsse oder der Grünzonen erstellt werden.

Unsere Features werden ebenfalls erklärt, da wir denken das zum Beispiel die Road Generation ein sehr komplexes Thema ist und auch eventuell den ein oder anderen Blender Benutzer interessiert.

Literaturverzeichnis

Add-on Tutorial - Blender Manual. (25. 2 2021). Von blender.org: https://docs.blender.org/manual/en/latest/advanced/scripting/addon_tutorial.html abgerufen

Blender, F. (31. 8 2020). *Blender 2.90 Manual*. Von https://docs.blender.org/manual/en/2.90/ abgerufen

Deemer, P., Benefield, G., Larman, C., & Vodde, B. (2010). *brianidavidson.com.* Von http://www.brianidavidson.com/agile/docs/scrumprimer121.pdf abgerufen

Grease Pencil: blender.org. (2020). Von https://www.blender.org/features/grease-pencil/ abgerufen

Noise Utilitites Blender. (25. 03 2021). Von Blender Website: https://docs.blender.org/api/current/mathutils.noise.html abgerufen

Nurseitov, N., Paulson, M., Reynolds, R., & Izurieta, C. (2009). Von https://www.cs.montana.edu/izurieta/pubs/caine2009.pdf abgerufen

ScrumGuides.org. (2020). Von https://scrumguides.org/scrum-guide.html abgerufen

Shannon, M. (05. 12 2019). *The one million limit*. Von python.org: https://www.python.org/dev/peps/pep-0611/#recursion-depth abgerufen

Sliger, M. (22. 10 2011). *pmi.org*. Von https://www.pmi.org/learning/library/agile-project-management-scrum-6269 abgerufen

Abbildungsverzeichnis

Stichwortverzeichnis