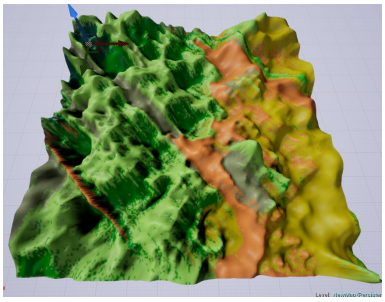
Universität Bremen

Fachbereich 3: Mathematik / Informatik
Studiengang: Informatik

Prozedurale Generierung von Multi-Biom-Landschaften

Mit Asset-Distribution auf Basis von Klimasimulation in der Unreal Engine 4

Masterarbeit



Roland Fischer
s_8ix2ba@uni-bremen.de
Matrikelnummer: 2484425

9. Februar 2019

1. Gutachter: Prof. Dr. Gabriel Zachmann
2. Gutachter: Prof. Dr.-Ing. Udo Frese
Betreuer: M.Sc. Philipp Dittmann

Erklärung

Ich versichere, die Masterarbeit ohne fremde Hilfe angefertigt zu haben. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen sind, sind als solche kenntlich gemacht.

Bremen, den 9. Februar 2019

.....

(Roland Fischer)

Inhaltsverzeichnis

Kapitel 1

Einleitung

Diese Arbeit setzt sich mit der prozeduralen Generierung von Multi-Biom-Landschaften auseinander, indem auf Basis von Heightmaps, Digital Elevation Models (DEMs) und diversen Parametern neue Heightmaps und Entitätenverteilungen, wie z.B. Vegetation, generiert werden. Die Visualisierung erfolgt in der *Unreal Engine 4*. Zur Berechnung wurden beobachtete Naturverhalten in Form einer Klimasimulation vereinfacht nachimplementiert. Berücksichtigt wurden Temperatur, Wind, Feuchtigkeit, Niederschläge und Höhe. Aus Aufwandsgründen wurden nicht alle bekannten, physikalische Modelle integriert, wie Plattentektonik, mehrere Luftschichten, Planetenrotation, Flüsse/Seen, Gezeiten, Erosion. Im Verlauf der Arbeit wird gezeigt, dass auch ohne diese Modelle beliebige, plausibel wirkende Multi-Biom-Landschaften mit mehreren hunderttausend Quadratkilometern Größe und ebenso vielen Entitäten in wenigen Minuten generiert werden können, die sich für diverse Zwecke weiterverwenden lassen.

Dieses Kapitel befasst sich mit der Motivation, der Problemstellung und dem Ziel, sich mit der prozeduralen Generierung von Multi-Biom-Landschaften auseinanderzusetzen. Abschließend wird eine detaillierte Kapitelübersicht über den weiteren Verlauf der Arbeit gegeben.

1.1 Motivation

Durch die zunehmende Digitalisierung in vielen Bereichen des Lebens und der Industrie sowie den stetigen Fortschritten in der Computergrafik besteht vermehrt der Wunsch nach bzw. steigt der Bedarf und Einsatz von 3D-Visualisierungen im Allgemeinen und der Darstellung von sehr großen Landschaften im Speziellen. Die Einsatzgebiete und Anwendungsfelder sind weit gestreut. Von Computerspielen, wo sich ein Trend zu sogenannten Open-World-Titeln abzeichnet, über Filmproduktionen mit Computer Generated Imagery (CGI)-Einsatz, bis hin zu Simulationen aller Art. Beispielhaft sei die Generierung von Asteroiden oder ganzen Planeten im Kontext der Weltraumexploration genannt. Diese Entwicklung zu immer größeren darzustellenden Landschaften und 3D-Umgebungen führt allerdings zu dem Problem des stetig wachsenden Erstellungsaufwandes, was schließlich zu steigendem Zeitaufwand und

höheren Kosten führt, wie (Amato, 2017, pp. 15-18) beschreibt. Abhilfe schaffen, bzw. das Problem abschwächen, können jedoch die Methoden der prozeduralen Generierung. Mit ihrer Hilfe werden digitale Inhalte automatisch anhand definierter und parametrisierter Regeln und Algorithmen erstellt, was den Bedarf an langwieriger und kostspieliger manueller Erstellung verringert. Somit werden die Designer entlastest und bei ihrer Arbeit unterstützt. Diese prozeduralen Methoden können sowohl alleinstehend oder auch unterstützend eingesetzt werden, eine Kombination aus von Hand erstellten Inhalten und Layouts bietet meist den besten Kompromiss aus Aufwand und abwechslungsreichen, interessanten und realistischen 3D-Umgebungen. Mit dem innerhalb dieser Arbeit entwickelten Systems ist dies möglich, da für das Terrain Heightmaps in einem Standardformat verwendet werden und so eine nachträgliche Manipulation durchgeführt werden kann. Zudem ist so ebenfalls eine hohe Kompatibilität zu anderen Programmen, z.B. zur Visualisierung oder Weiterverarbeitung, gegeben.

1.2 Problemstellung

Das Forschungsfeld der prozeduralen Generierung ist relativ alt, mit entsprechend vielen bestehenden Forschungsergebnissen. Zuerst eingesetzt wurde sie Anfang der 80er Jahre in Spielen wie *Elite* und *Rogue* (Amato, 2017, pp. 17-18,21-24), auf welche in Abschnitt 2.1 noch näher eingegangen wird. Es wurden viele verschiedene Methoden und Ansätze entwickelt, vgl. (Hendrikx et al., 2013, pp. 1, 21-24), jedoch gibt es noch viel Verbesserungspotenzial. Die meisten dieser Methoden können nur eine oder wenige der Anforderungen an ein solches System zufriedenstellend abdecken oder haben anderweitige Defizite, wie in (Smelik et al., 2014, pp. 3-4) beschrieben. Noise-Methoden z.B. sind zwar performant und flexibel, aber durch die wenig intuitiven Parameter schwer zu kontrollieren und oft sind Selbstähnlichkeit oder eine sich regulär wiederholende Erscheinung nicht gewünscht. Hingegen können Algorithmen, die natürliche Phänomene, wie z.B. Erosionsprozesse, präzise simulieren, realistischere Ergebnisse liefern, sind aber vergleichsweise teuer und auf eben diese Prozesse eingeschränkt. Die vier entscheidenden zu erfüllenden Kriterien sind in der Regel: realistisch wirkende Ergebnisse, eine hohe Benutzbarkeit, Flexibilität, was die möglichen Ergebnisse angeht, und hinreichende Performanz. Problematisch ist, dass diese sich teils gegenseitig widersprechen. Ein höherer Grad an Realismus geht meist mit niedriger Performanz einher, Flexibilität dagegen geht zu Lasten der Benutzbarkeit. Drei Arten von prozeduralen Techniken, die alle verschiedene Vor- und Nachteile bieten und unterschiedliche Schwerpunkte bezüglich der Anforderungen setzen, sind die sogenannten synthetischen, physikalisch-basierten und Beispiel-basierten Methoden (vgl. (Génevaux et al., 2013, p. 1) und (Viitanen, 2012, p. 2)).
Das Erzeugen von großen, realistischen Terrains, welche aus mehreren Biomen bestehen, ist ein relativ wenig erforschter Teilbereich der prozeduralen Generierung, der aber durch vermehrtes Auftreten dieser Aufgabe an Bedeutung gewinnt. Ein bekanntes Beispiel für diesen Anwendungsfall ist das Spiel *Minecraft*, was ebenfalls später in Abschnitt 2.1.1 thematisiert wird. Zusätzliche Schwierigkeiten bei diesem Anwendungsfall sind das Erzeugen einer

plausiblen Biomverteilung, die Übergänge zwischen ihnen und anstatt eines homogenen, realistischen Terrains müssen nun diverse, zusammenhängende Subbereiche erzeugt werden, die für sich genommen realistisch erscheinen, aber auch in ihrer Gesamtheit. Zudem haben die Biome Einfluss auf die jeweils vorkommenden Objekte, was für ein überzeugendes Resultat beachtet werden muss. Beispielhaft kommen in einer Savanne verstärkt hitze- und trockenheitsresistentere Gräser, Sträucher und vereinzelt Bäume und Baumgruppen vor, während in den gemäßigten Klimazonen von Europa z.B. dichte, sommergrüne Laubwälder üblich sind.

1.3 Zielsetzung

Ziel dieser Arbeit ist es, ein Konzept für ein prozedurales System zu entwerfen, welches die Aufgabe hat, das oben genannte Problem zu lösen, große, realistisch wirkende und aus mehreren Biomen bestehende Landschaften zu generieren und dabei einen möglichst guten Kompromiss aus den vier Anforderungen zu erzielen. Dies sind, wie im vorherigen Abschnitt aufgeführt: realistisch wirkende Ergebnisse, Benutzbarkeit, Flexibilität und Performanz. Um dies zu erreichen, werden bestehende Methoden der prozeduralen Generierung betrachtet sowie neue Methoden aus anderen Forschungs- und Anwendungsgebieten, etwa des wachsenden Machine-Learning-Themenfeldes, speziell der Neuronalen Netze, in diesem Kontext auf Nutzbarkeit geprüft. Unter Verwendung und durch Kombination einer Auswahl dieser Techniken und Methoden werden eine prototypische Implementierung erstellt und die Ergebnisse analysiert. Die Aufgabe schließt das prozedurale Erzeugen von Biomen, dem Terrain selbst und das Verteilen und Platzieren von Objekten ein, nicht jedoch das automatische Generieren ihrer Meshes bzw. Geometrie. Ebenfalls ist das Erstellen von Texturen des Terrains nicht Teil dieser Arbeit.
Vielversprechend für das Erreichen des Ziels ist vor allem die Kombination der Techniken für die jeweiligen Unteraufgaben, was Synergieeffekte ergeben und die jeweiligen Stärken hervorheben sollte. In diesem Zusammenhang sind auch Algorithmen aus den Bereichen der Bildverarbeitung, des maschinellen Sehens und, eng damit verbunden, der Textursynthese relevant. Als Basis für die Biomverteilung soll eine einfache Klimasimulation dienen, um durch Nachahmen der realen, physikalischen Prozesse plausible und für den Nutzer leichter nachvollziehbare Verteilungen zu produzieren. Anhand der so generierten Biomverteilung werden weitere Algorithmen angewendet, um das Terrain zu generieren und die Objekte zu platzieren. Die prototypische Implementierung wird unter Zuhilfenahme der 3D-(Spiele)-Engine *Unreal Engine 4* entwickelt, was einige peripher relevante Aufgaben der Computergrafik abnimmt und das Grundgerüst für eine visuell ansprechende Darstellung bietet.

1.4 Struktur der Arbeit

Die Arbeit gliedert sich in sechs Kapitel. In Kapitel 2 werden für diese Arbeit relevante grundlegende Techniken und Konzepte erläutert, Begriffe definiert sowie die jeweils aktuellen

und bedeutsamen Publikationen thematisiert. Im darauffolgenden Kapitel 3 werden theoretische Überlegungen über die vorgestellten Techniken angestellt, ihre Eignung für diese Arbeit und mögliche Kombinationsmöglichkeiten erörtert und schließlich ein finales Konzept vorgestellt. In Kapitel 4 werden die hier in einem Prototyp eingesetzten Algorithmen und Datenstrukturen besprochen, vertieft und anschließend die Implementierungsdetails detailliert beschrieben. Dabei wird ebenfalls auf alternative Methoden und Optimierungspotenzial hingewiesen. Das Kapitel schließt mit einer ausführlichen Komplexitätsanalyse. In Kapitel 5 werden die Ergebnisse der Implementierung vorgestellt und evaluiert. Die Arbeit endet mit Kapitel 6, in dem zum einen ein Fazit gezogen wird und zum anderen mögliche Verbesserungs- und Erweiterungsmöglichkeiten dargelegt werden.

Kapitel 2

Grundlagen und aktueller Stand der Forschung

In diesem Kapitel werden die Grundlagen für diese Arbeit vermittelt, abschnittsweise relevante Techniken und Algorithmen vorgestellt und jeweils auf wichtige Publikationen hingewiesen. Im ersten, hierauf folgenden Abschnitt wird eine Einführung über prozedurale Generierung gegeben. Danach wird auf prozedurale Terraingenerierung (PTG) im Speziellen eingegangen und in den anschließenden Abschnitten werden die verschiedenen Methoden und Disziplinen beschrieben, die in der prozeduralen Content-Generierung und PTG zum Einsatz kommen oder die anderweitig von Bedeutung sind.

2.1 Prozedurale Generierung

Dieser Abschnitt beginnt zunächst mit der Begriffsdefinition der prozeduralen Generierung, gefolgt von einer Übersicht über dessen Anwendungsgebiete und schließlich wird eine grobe Einteilung und Vorstellung der möglichen Techniken gegeben.
Im Allgemeinen versteht man unter dem Begriff der prozeduralen Generierung das automatische Erzeugen von Daten. Oft wird in der Unterhaltungsindustrie und entsprechenden wissenschaftlichen Veröffentlichungen allerdings der Term prozedurale Content-Generierung (PCG), engl. Procedural Content Generation, verwendet, um zu verdeutlichen, dass ein Anwendungsinhalt der einen oder anderen Art erstellt wird. Beispielhaft wird der Begriff in (Amato, 2017) und (Hendrikx et al., 2013) genutzt. Prozedural bedeutet „durch Prozeduren/Funktionen“. Content ist ein in der Spiele- und Filmbranche verwendeter Begriff für Assets (digitale Objekte aller Art), bzw. Inhalt eines Spiels, Films oder sonstiger Anwendungen. Content sind in diesem Zusammenhang z.B. Modelle, Level, Spielregeln, Geschichten, Texturen, Animationen etc., siehe (Shaker et al., 2016, p. 1). (Smelik et al., 2011, pp. 352-363) definieren PCG als „any kind of automatically generated asset based on a limited set of user-defined input parameters“, (Freiknecht and Effelsberg, 2017, p. 5) sprechen etwas

präzisierend von „Procedural content generation is the automatic creation of digital assets for games, simulations or movies based on predefined algorithms and patterns that require a minimal user input“. Letztendlich wird Content automatisch nach einem definierten und vom Nutzer parametrisierten Algorithmus erzeugt.

Das prominenteste Anwendungsgebiet dafür ist die Spielentwicklung, wo die PCG bereits eine lange Geschichte hat und bis in die 80er zurückreicht, nachzulesen in (Amato, 2017, pp. 17-18,21-24). Einige der ersten Spiele, welche die PCG einsetzten, um endlose (oder endlos viele verschiedene) Welten zu generieren und Platz durch Laufzeitgenerierung zu sparen, waren das Weltraum-Handelsspiel *Elite*[1], dargestellt in Abbildung 2.1, und das Rollenspiel *Rogue*[2].

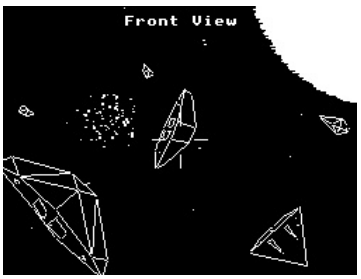


Abbildung 2.1 Eine Kampfszene aus dem Spiel *Elite* (Bell, 2015).

In *Elite* wurden die Galaxien mitsamt den Planeten und allen relevanten Daten, wie Positionen, Namen, Beschreibungen und Preisen der Handelsgüter, prozedural generiert, in *Rogue* die Spiellevel, in diesem Fall Dungeons, sowie Gegner- und Item-Positionen. In 2.2 (unten links) ist ein Bild von dem neueren Spiel *Spelunky*[3] dargestellt, was ganz ähnliche Level wie das damalige *Rogue* generiert. In vielen Rollenspielen werden weiterhin die sammelbaren Gegenstände mit ihren Eigenschaften prozedural generiert, so beispielhaft in *Borderlands*[4]. Andere Spiele lassen bestimmte Spielobjekte oder Quests (zum Teil) automatisch generieren, um vor allem Arbeitszeit und damit Kosten zu sparen, so ist z.B. der Gebrauch der Middleware *SpeedTree*[5] sehr verbreitet, um prozedural Bäume und Vegetation zu erstellen, wie das Bild oben rechts in Abbildung 2.2 zeigt. Neuere Beispiele für den umfangreichen Einsatz von PCG sind, wie in (Stangl, 2017, p. 1) beschrieben, z.B. *Minecraft*[6], *No Mans Sky*[7] und *Dwarf Fortress*[8], in denen ganze Spielwelten prozedural generiert werden. Angefangen mit *Minecraft*, in dem zur Laufzeit eine endlose Voxel-Welt erzeugt wird, über *No Mans Sky*, wo in einem Universum unzählige 3D-Planeten, inklusive Flora und Fauna, generiert werden, beispielhaft zu sehen oben links in Abbildung 2.2, bis hin zu *Dwarf Fortress*, was nahezu alle Aspekte der Spielwelt prozedural erstellt und berechnet. Dazu gehören zum Beispiel das Terrain und

[1] `http://www.iancgbell.clara.net/elite/` (Abgerufen am 11.12.2018)
[2] `https://www.mobygames.com/game/rogue` (Abgerufen am 11.12.2018) (Keine offizielle Website verfügbar.)
[3] `https://www.spelunkyworld.com/` (Abgerufen am 11.12.2018)
[4] `https://borderlands.com/en-US/` (Abgerufen am 11.12.2018)
[5] `https://store.speedtree.com/` (Abgerufen am 11.12.2018)
[6] `https://minecraft.net/de-de/` (Abgerufen am 11.12.2018)
[7] `https://www.nomanssky.com/` (Abgerufen am 11.12.2018)
[8] `http://www.bay12games.com/dwarves/` (Abgerufen am 11.12.2018)

die Biome der Welt, basierend auf Faktoren wie Temperatur und Niederschlag, Rohstoff- und Populationsverteilungen, die Geschichte der Welt und Religion. Unten rechts in Abbildung 2.2 wird anhand eines Ausschnitts von *Civilization V*[9] illustriert, wie durch prozedural generierten Terrain und Rohstoffverteilungen, in diesem Fall Hexagon-basiert, Spielwelten erzeugt werden.

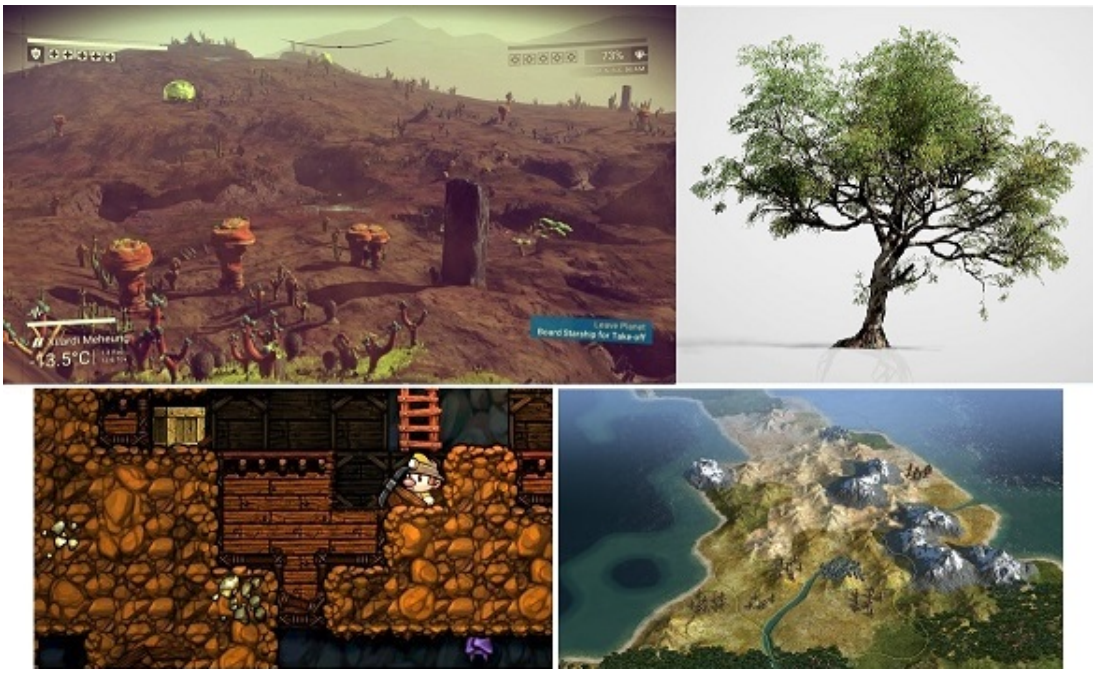


Abbildung 2.2 Vier Beispielszenen prozeduraler Generierung. Oben links: Prozedural generierter Planet mit Terrain, Lebewesen und Vegetation in *No Mans Sky* (Baines, 2016). Oben rechts: In *SpeedTree* prozedural erstellter Baum (SpeedTree, 2017). Unten links: Prozedural erzeugtes 2D-Level in *Spelunky*, auch Gegner, Händler und Items werden automatisch platziert (Moss, 2016). Unten rechts: Prozedural generierte Landmassen und Rohstoffverteilung in *Civilization V* (Totilo, 2010).

Auch im Film wird PCG erfolgreich eingesetzt, um, vergleichbar Computerspielen, große virtuelle Landschaften zu generieren oder sie mit vielen ähnlichen, aber leicht unterschiedlichen Objekten zu füllen, etwa Bäumen (auch hier oft mit der o.g. Software *SpeedTree*) oder Individuen einer großen Armee. Darüber hinaus kann prozedurale Generierung auch genutzt werden, um virtuelle Welten oder Layouts zu Simulationszwecken zu generieren, beispielsweise Straßennetzwerke für Fahrsimulatoren oder in (Campos et al., 2015, p. 1) Flughafenterminal-Layouts zur Evaluierung durch Crowd Simulation. Beliebige Texte, seien es Geschichten, Nachrichten oder Dialoge, können ebenfalls durch prozedurale Methoden, wie etwa Formale Grammatiken, erzeugt werden. Ein weiteres, eher unbekannteres Anwendungsfeld ist die prozedurale Generierung von Sound und Musik. Sie kann dabei helfen, dynamischen, adaptiven und sich nicht merklich wiederholenden Ton zu generieren, vgl. (Strout, 2017) und (Plans and Morelli, 2012, pp. 192-198). Speziell jedoch die Generierung von Landschaften bzw. Terrains

[9] `https://civilization.com/civilization-5/` (Abgerufen am 11.12.2018)

ist ein häufiger Anwendungsfall im Feld der Computergrafik, breit erforscht, und bildet einen eigenen Subbereich, die PTG, welche ausführlich in Abschnitt 2.1.1 vorgestellt wird.

Zur PCG werden viele verschiedene Methoden verwendet, die nicht ausschließlich aus der Computergrafik, sondern aus den unterschiedlichsten Disziplinen stammen, und die PCG so zu einem sehr interdisziplinären Feld machen. Eine einheitliche Einordnung und Klassifizierung der Methoden und Ansätze ist deshalb schwierig, je nach Anwendungsfall eignen sich zudem manche mehr als andere. So gibt es in der Literatur je nach Fachgebiet und Ziel oft ganz unterschiedliche Ansichten über mögliche Methodiken und entsprechend auch diverse Klassifizierungen derer. (Hendrikx et al., 2013, pp. 1, 21-24) geben jedoch einen guten Überblick, ohne zu sehr ins Detail zu gehen, und schlagen eine Einteilung in die folgenden sechs übergeordneten Gruppen vor:

- Pseudo-Random Number Generators, womit vor allem auch Noise(-Funktionen) eingeschlossen werden. Sie generieren Sets von „zufälligen“ Datenpunkten, um so die in der Natur augenscheinlich häufig auftretende Zufälligkeit nachzubilden. Mehr Informationen dazu sind im Abschnitt 2.1.3 nachzulesen.
- Generative Grammatiken sind Regelsets, die iterativ auf Eingaben angewendet werden, um grammatikalisch korrekte Ausgaben zu generieren (mehr dazu im Abschnitt 2.1.2).
- Image Filtering umfasst Techniken, die auf Bilder angewendet werden, um sie nach einem subjektiven Maß zu verbessern, bestimmte Details herauszuarbeiten oder generell zu verändern (siehe Abschnitt 2.3).
- Spatial Algorithms fasst Methoden zusammen, die spezielles Augenmerk auf die räumliche Ausdehnung und Anordnung von Objekten legen und diese ausnutzen. Oft wird Gebrauch von räumlichen Datenstrukturen gemacht, um etwa bestimmte relevante Bereiche dynamisch mit Details zu versehen. Zu den gebräuchlichsten Methoden gehören z.B. Tiling und Layering, bei denen Textur- oder Leveltiles prozedural aneinandergefügt werden oder in Schichten übereinander, um so neue, komplexere zu generieren.
- Modeling and Simulation of Complex Systems ist eine Kategorie, bei der zur Problemlösung große Berechnungsmodelle aufgestellt werden oder komplexe Verhalten, etwa aus der Natur, simuliert werden (siehe dazu Abschnitte 2.6 und 2.7). Ein Beispiel wäre Cellular Automata, ein diskretes Berechnungsmodell, bei dem für eine Menge von Zellen, die bestimmte Zustände annehmen können, in jedem Zeitschritt ihr neuer Zustand, abhängig von der Nachbarschaft und gewissen Regeln, berechnet wird.
- Artificial Intelligence (AI) wird meist zur Simulation von intelligentem bzw. menschlichem Verhalten angewendet, seit einiger Zeit wird sie aber auch in der PCG eingesetzt. Zum Beispiel Evolutionäre Algorithmen, welche versuchen, Optimierungsprobleme zu lösen, können möglichst optimale Lösung für ein Terrain aus einem Suchraum berechnen. Ein weiteres Beispiel aus dem AI-Bereich sind Neuronale Netze, die im entsprechenden Abschnitt 2.5.1 näher erläutert werden.

Ein übliches Unterscheidungskriterium im Bereich der PCG ist zudem, ob der Ansatz On- oder Offline ist, wie in (Togelius et al., 2011, p. 2) und (Shaker et al., 2016, pp. 7-8) beschrieben. Online bedeutet in diesem Fall, dass der Content prozedural zur Laufzeit der Anwendung generiert wird, während, wenn man von Offline spricht, die Generierung einmalig vor der Ausführung, z.B. während der Entwicklung, geschieht. Beide Varianten haben Vor- und Nachteile. Ein Online-Algorithmus kann theoretisch unendliche große Welten erzeugen und es ist keine dauerhafte Speicherung der generierten Inhalte notwendig. Offline-Generierung dagegen hat keine so gravierende zeitliche Beschränkung und kann so deutlich mehr Rechenzeit in detailliertere, realistischere Ergebnisse investieren. Ebenso besteht die Möglichkeit der Nachbearbeitung durch einen Designer.

2.1.1 Prozedurale Terraingenerierung

Wie bereits im vorherigen Abschnitt erwähnt, ist das automatische Generieren von Terrain, also der Landschaft bzw. dem Boden einer virtuellen Umgebung, eine häufige Aufgabe, der sich schon von vielen mit den unterschiedlichsten Methoden gewidmet wurde. Es ist eines der klassischen Gebiete der PCG und wird prozedurale Terraingenerierung (PTG) genannt. Dabei lassen sich zwei Varianten der Repräsentation des Terrains unterscheiden, wie (Smelik et al., 2014, p. 2) erläutern und in Abbildung 2.3 illustriert wird. Die verbreitetere Variante ist die durch sogenannte Heightmaps. Dies sind zweidimensionale, regelmäßige, rechtwinklige Gitter, bei dem jeder Vertex oder jede Zelle einen Wert enthält, der die Höhe an dieser Position beschreibt. Heightmaps sind leicht umzusetzen und zu handhaben. Durch ihre zweidimensionale Natur ist es allerdings mit ihnen nicht möglich Überhänge oder Höhlen darzustellen. Dazu sind andere Datenstrukturen nötig, etwa Voxel, welche einen Gitterpunkt, bzw. ein würfelförmiges Volumen, in einem dreidimensionalen Gitter darstellen.

Abbildung 2.3 Oben: Darstellungen von überwiegend prozedural erstellten Voxel-Terrains in *Minecraft* (man beachte die Überhänge im rechten Bild) (Bellic, 2013)(Milling, 2015). Unten: Ein mit *Houdini*[10]semi-prozedural erstelltes Terrain, basierend auf einer Heightmap (Dracott, 2017).

Bezüglich der Methoden zur (semi-)automatischen Generierung von Terrain lassen sich in der Literatur einige Autoren finden, die sie in drei elementare Gruppen gliedern. So sind dies etwa nach (Génevaux et al., 2013, p. 1) die „[...] procedural, physics-based, and sketch- or example-based [...]“ Methoden. Die Gruppe der prozeduralen Methoden umfasst klassische Methoden der PCG wie z.B. Noise und Fraktale. (Musgrave, 2003, p. 2:2) definiert Fraktale als „a geometrically complex object, the complexity of which arises through the repetition of some shape over a range of scales“. Das heißt geometrisch komplexe Objekte, bei denen sich die Form über diverse Detailebenen wiederholt. Viele Aspekte der Natur enthalten Fraktale bzw. lassen sich gut durch sie darstellen, etwa Berge oder Wolken. Daher werden sie oft zur PTG genutzt und etwa durch Fractal Noise (dazu mehr in Abschnitt 2.1.3) generiert. Physikalisch-basierte Methoden simulieren physikalische Prozesse der Natur, z.B. Plattentektonik oder Erosion, und werden daher auch oft in der Biologie und Geologie angewendet. Beispiel-basierte Techniken hingegen basieren darauf, aus Beispieldaten, Texturen oder Heightmaps, durch cleveres Replizieren und Anordnen von Teilbereichen, größere homogene Ausgaben zu erzeugen. Für die PTG wird meist auf sogenannte DEMs zurückgegriffen, was in der Regel durch Satelliten aufgenommene Höhenkarten des Erdterrains sind. Die Verallgemeinerung dieses Prinzips wird in der Computergrafik als Textur- bzw. Bildsynthese bezeichnet und auch viele Methoden der Bildverarbeitung finden in diesem Bereich Anwendung. Die Sketch-basierten Methoden gleichen den Beispiel-basierten, hier werden jedoch die Beispieldaten manuell durch Nutzer-

[10] `https://www.sidefx.com/` (Abgerufen am 11.12.2018)

sketches gezeichnet, welche anschließend verarbeitet und interpretiert werden müssen, um daraus das Terrain zu generieren.
(Guérin et al., 2017, p. 1) machen im Prinzip die gleiche Einteilung wie die eben vorgestellte von (Génevaux et al., 2013, p. 1), zählen aber die Sketch-basierten Methoden als eine eigene Gruppe auf und verwenden statt „physics-based“ den Begriff der Simulation. Diese Klassifizierungen implizieren, dass Beispiel- und physikalisch-basierte Methoden nicht prozedural wären. Dehnt man aber die im vorherigen Abschnitt gegebenen Definitionen aus, wären alle hier aufgeführten Methoden als Untergruppen im Bereich der PCG einzuordnen. Z.B. könnte man die physikalisch-basierten Methoden in der Einteilung von (Hendrikx et al., 2013, pp. 1, 21-24) im vorherigen Abschnitt dem Bereich der „Modeling and Simulation of Complex Systems“ zuordnen, die prozeduralen Methoden entsprächen vor allem den Grammatiken und Noise und die Beispiel-basierten, je nach konkretem Algorithmus, entweder “AI", „Spatial Algorithms“ oder „Image Processing“. Statt prozeduraler Methoden passt daher im Kontext dieser Gliederung besser der Begriff „synthetisch“, wie ihn (Viitanen, 2012, p. 2) zur Abgrenzung der physikalisch-basierten Methoden nutzt.
Jeder dieser drei Ansätze hat spezielle Vor- und Nachteile. Synthetische Methoden sind schnell und vergleichsweise einfach zu implementieren, dafür ist es jedoch schwer, realistisch wirkende Ergebnisse zu erzielen. Physikalische Methoden produzieren, vom geologischen Standpunkt aus gesehen, korrektere und damit realistischer aussehende Ergebnisse, sind aber schwieriger zu implementieren und langsamer zu berechnen. Beispiel-basierte Methoden schließlich können ebenfalls sehr realistische Terrains generieren, sind jedoch von den Eingabebeispielen abhängig und können nur schwerlich komplett neue Features generieren.

Im Folgenden werden einige klassische, bekannte Methoden zur synthetischen PCG vorgestellt, beginnend mit den Formalen Grammatiken.

2.1.2 Formale Grammatiken

Formale Grammatiken sind eine typische Technik der prozeduralen Generierung, dessen Verständnis für die später betrachteten L-Systeme relevant ist. Nach (Carpenter, 2011, pp. 3-4) und (Shaker et al., 2016, pp. 45-46, 74-75) bestehen Formale Grammatiken aus einer endlichen Menge von Produktions- bzw. Ersetzungsregeln und einem endlichen Alphabet, um aus einem Eingabestring einen neuen zu generieren. Dazu werden wiederholt einzelne Literale gemäß der definierten Regeln ersetzt. So könnte z.B. das Literal A durch eine Regel $A \Rightarrow B$ zu einem B werden und durch iterative Anwendung das B durch eine andere Regel $B \Rightarrow b$ durch b ersetzt werden. Es sind jedoch nicht alle Grammatiken zwingend deterministisch. Es gibt durchaus welche, bei denen mehrere Regeln gleichzeitig angewendet werden können und z.B. nach definierten Wahrscheinlichkeiten eine gewählt wird. Was ein String oder Literal im Kontext der PCG schließlich repräsentiert, ist komplett offen, es können somit z.B. Missionsstrukturen, Gebäude oder Höhlen generiert werden, wie in (Shaker et al., 2016, Kap. 5) beschrieben wird. Ein weiteres Beispiel sind (Larive and Gaildrat, 2006, pp. 429-437), die eine

Wall- und eine Split-Grammatik nutzen, um Gebäude bzw. spezifisch die Wände iterativ zu generieren.

L-Systeme

L-Systeme, auch Lindenmayer-Systeme, sind eine spezielle Form der Formalen Grammatiken, die parallel ausgeführt werden. Der Eingabestring wird iterativ ersetzt, jedoch werden pro Iteration möglichst viele Regeln gleichzeitig auf die verschiedenen Literale angewendet. Durch Interpretation der Literale als Zeichen- bzw. Modellierungsanweisungen und zwei speziellen, konstanten Literalen, die als Push- und Pop-Funktionen eines Stacks dienen, lassen sich leicht komplexe und verzweigte Strukturen erzeugen, vgl. Abb. 2.4.

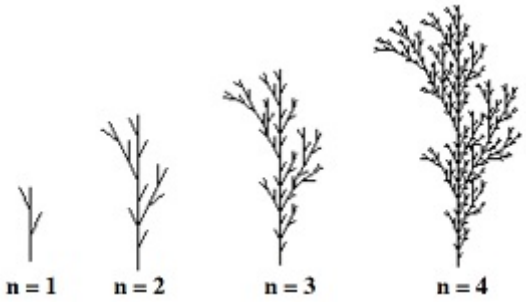


Abbildung 2.4 L-System nach vier Iterationen zur Modellierung von Pflanzen. Man erkennt gut die sich verzweigende Struktur. (Shaker et al., 2016, p. 78).

Mehr über L-Systeme findet sich in (Carpenter, 2011, p. 3) und (Shaker et al., 2016, pp. 75-77). L-Systeme wurden zur Modellierung von organischen Systemen entwickelt, eignen sich aber auch zur Generierung von Fraktalen. Heutzutage finden sie weite Verbreitung bei der Generierung von Pflanzen, beispielhaft seien (Lluch et al., 2003, pp. 31-38) und (Deussen et al., 1998) genannt, welche L-Systeme zur Modellierung und zum Rendering von Vegetation einsetzen.

2.1.3 Noise

Die hier beschriebenen Noise-Funktionen sind eine der ältesten und verbreitetsten Techniken zur PCG und speziell der PTG. Sie finden oft Anwendung zur Erzeugung von Terrain oder Texturen aller Art. Der Begriff Noise-Funktion bezeichnet nach (Hyttinen, 2017, pp. 1-8) im Kontext der Computergrafik und verwandter Gebiete eine Funktion, die einem n-dimensionalen Punkt im Raum einen pseudozufälligen, reellen Wert zuweist. Pseudozufällig bedeutet, dass die Ausgabe zufällig erscheint, aber es nicht ist. Sie hängt von einem Seed/Hash ab und bei gleicher Eingabe wird immer dasselbe herauskommen. Eine ideale Noise-Funktion ist kontinuierlich, nicht-periodisch und random accessible, was bedeutet, dass jeder Punkt jederzeit unabhängig von allen anderen berechnet werden kann. Es gibt eine Vielzahl von verschiedenen Noise-Funktionen, die unterschiedliche Muster produzieren, wenn man den

Output grafisch interpretiert, z.B. als Grauwertbild.
Zwei grundlegende Klassen von Noise sind Value Noise und Gradient Noise. Viele Noise-Funktionen sind zudem Gitter-basiert und zählen so zu der Kategorie Lattice Noise. Value Noise, eine der simpelsten Noise-Funktionen, besteht aus zwei Schritten: erst wird für Eckpunkte auf einem groben Gitter jeweils eine pseudozufällige Zahl generiert und dann werden für die Stellen zwischen den Gitterpunkten die Werte interpoliert. Gradient Noise basiert im Gegensatz zu Value Noise nicht auf zufälligen Werten, die etwa den Gitterpunkten zugewiesen werden, sondern zufälligen Gradienten, was weniger Artefakte produziert. Eine detaillierte Erklärung wird im nächsten Abschnitt anhand zweier sehr bekannter Gradient-Noise-Verfahren gegeben.
Um mit Noise komplexere Muster zu erzeugen, die z.B. Terrain besser nachbilden, kann die Noise-Funktion wiederholt mit verschiedenen Frequenzen und Amplituden aufgerufen und somit die Ergebnisse überlagert werden. Jede Iteration bildet eine sogenannte Oktave, bei der sich üblicherweise, aber nicht festgelegt, die Amplitude halbiert (Gain/Persistence) und die Frequenz verdoppelt (Lacunaty). Es entsteht eine Überlagerung von gleichartigen Mustern in verschiedenen Skalen. Diese Technik nennt sich Fractional Brownian Motion (FBM), bzw. spricht man in diesem Fall durch die Selbstähnlichkeit von Fractal Noise, wie in (Hyttinen, 2017, p. 21) und (Shaker et al., 2016, pp. 62-64) nachzulesen ist. In Abbildung 2.5 (rechts) wird dies beispielhaft illustriert. Im Bereich der PTG ist es weit verbreitet, nach ähnlichem Vorgehen verschiedene Noise-Funktionen in unterschiedlichen Oktaven zu kombinieren und ggf. einfache mathematische Operationen auf diese anzuwenden, um eine Vielfalt von unterschiedlichen, komplexen Ergebnissen zu erzielen und so verschiedenartige Terrains nachzubilden.

Perlin und Simplex Noise

Perlin Noise, vorgestellt von Ken Perlin im Jahr 1985, ist die erste und eine der bekanntesten Gradient-Noise-Funktionen. Sie ist laut (Hyttinen, 2017, pp. 11-12) und (Shaker et al., 2016, pp. 61-62) relativ günstig zu berechnen, einfach verständlich und vielseitig einsetzbar. Es wird wie bei Value Noise ein Gitter verwendet, das zur Vereinfachung ganze Integerzahlen als Raster verwendet. Hier allerdings wird statt jeweils einem pseudozufälligen Wert ein pseudozufälliger Vektor, der als Gradient dient, für einen Gitterpunkt berechnet. Das Verfahren läuft folgendermaßen ab: Für jeden zu berechnenden Eingabepunkt werden die benachbarten Gitterpunktkoordinaten und anschließend ihre Gradienten berechnet. Aus dem jeweiligen Gradient und der Distanz zum Eingabepunkt werden mittels Skalarprodukt die Werte berechnet, die sich nach jeweils einem der Nachbarn ergeben würden. Diese werden schlussendlich zu dem finalen Ergebnis interpoliert, vgl. Abbildung 2.5 (links). Es ergibt sich eine Komplexität von $O(2^n)$, wobei n die Dimensionalität beschreibt. Die ursprüngliche Implementierung nutzte eine Smooth-Step-Funktion zur Interpolation, was durch Diskontinuitäten in den Ableitungen zu visuellen Artefakten führte.

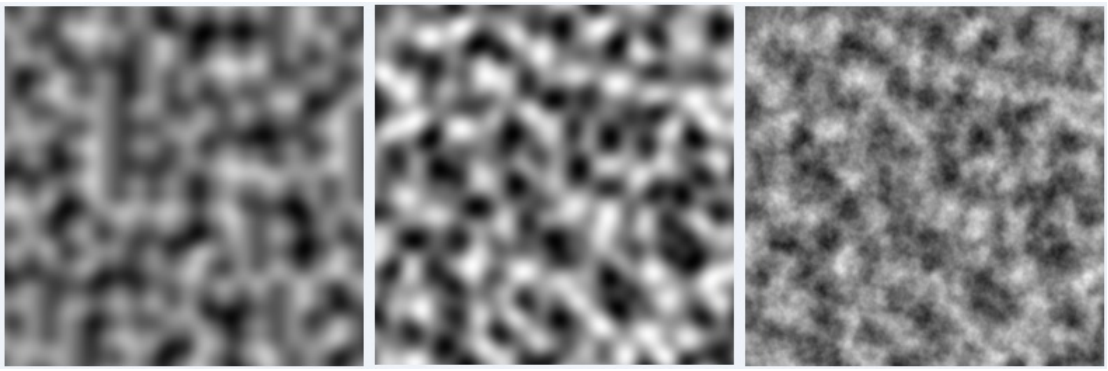


Abbildung 2.5 Drei Beispiele von Simplex- und Perlin-Noise-Arten mit vergleichbarer Auflösung. Links: Perlin Noise. Mitte: Simplex Noise. Rechts: Simplex-Fractal Noise.

Simplex Noise ist, wie (Hyttinen, 2017, pp. 12-14) beschreibt, eine verbesserte Version von Perlin Noise, die Perlin im Jahr 2001 vorstellte, siehe 2.5 (mittig). Es werden ungefähr die gleichen Muster erzeugt, aber zum einen ist sie visuell isotropisch, was direktionale Artefakte vermeidet, und zum anderen etwas performanter zu berechnen, speziell in höheren Dimensionen. Die direktionalen Artefakte entstanden in Perlin Noise durch die Verwendung eines Axis Aligned Lattice, wodurch die Ausrichtung des Koordinatensystems erkennbar wurde. Simplex Noise verwendet statt dem regulären Gitter ein Gitter bestehend aus den namensgebenden Simplices. Unter Simplex versteht man die Generalisierung eines Dreiecks in den n-dimensionalen Raum, oder anders ausgedrückt, die konvexe Hülle über n+1 affin unabhängige Vertices. Z.B. eine Linie im eindimensionalen Raum und ein Dreieck in 2D. Damit ist ein Simplex die einfachste geometrische Form, die sich im n-dimensionalen Raum aneinanderreihen lässt. Das führt durch die reduzierte Zahl zu berechnender Nachbarpunkte zu einer verringerten Komplexität von $O(n^2)$ gegenüber Perlin Noise. Durch die Verwendung des Simplex-Gitters und das Ersetzen der Interpolation durch eine Kernelsummation konnten die visuellen Artefakte beseitigt werden, unter denen Perlin Noise litt.

Cellular Noise

Cellular Noise, auch Worley oder Voronoi Noise, ist ein weiterer, verbreiteter Noise-Typ, mit dem sich Zellen-basierte Strukturen generieren lassen, etwa zur Texturierung bzw. Darstellung von Gestein, Felsen oder Wasser. Zuerst vorgestellt wurde die Technik von (Worley, 1996). Das Prinzip sowie das optische Resultat sind sehr ähnlich dem der Voronoi-Diagramme. Zuerst wird eine Menge von Feature-Punkten im Raum verteilt. Dies kann mittels einer beliebigen Verteilungen passieren, üblicherweise wird jedoch eine Poisson-Verteilung verwendet (siehe 2.7). Anschließend wird für jeden Punkt/Texel zur Ermittlung des Noise-Wertes die Distanz zu dem n-nächsten Feature-Punkt ermittelt. Im Normalfall wird der nächstgelegene gesucht, teils aber auch die Distanzen der m-nächstgelegenen, wie (Hyttinen, 2017, p. 18) erläutert. Wie in Abbildung 2.6 zu sehen ist, ergeben sich je nach genutzter Distanzfunktion verschiedene Muster.

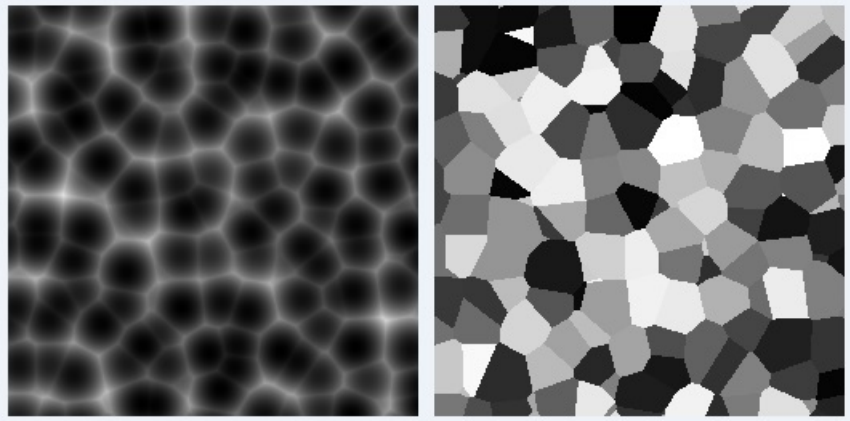


Abbildung 2.6 Verschiedene Varianten von Cellular Noise. Links wurde die euklidische Distanz genutzt, rechts stattdessen der Zellenwert zurückgegeben.

Zur Beschleunigung der Berechnung werden meist reguläre Gitter verwendet. Dies hat den Vorteil, dass nicht mehr die Distanz zu allen anderen Feature-Punkten geprüft werden muss, sondern nur noch zu denen in der Umgebung.

2.2 Textur/Bildsynthese

Bei der Textursynthese ist das Ziel, aus einer oder mehreren vom Nutzer bereitgestellten Beispieltexturen neue gleichartige oder größere Texturen zu erzeugen. Man unterscheidet zwischen zwei Hauptvorgehensweisen, zum einen der Pixel-basierten und zum anderen der Patch-basierten Synthese, wie in (Barnes and Zhang, 2017, pp. 1-2) beschrieben. Im ersten Fall wird für jeden Ausgabepixel aus der Beispieltextur der geeignetste kopiert, während die Patch-basierten Methoden kleine Regionen von Pixeln, Patches, auf einmal verarbeiten.
Viele Methoden basieren darauf, die Textur als Markov Random Field (MRF) zu modellieren, was ein probabilistisches, grafisches Modell ist, um ungerichtete Zusammenhänge zu beschreiben. Wie in (Kwatra et al., 2003, p. 1) nachzulesen ist, wird dabei jedem Knoten des Modells, welcher eine Zufallsvariable darstellt, ein oder mehrere Pixel des Ausgangsbildes zugeordnet und angenommen, dass der Wert einer Zufallsvariablen nur von seiner Nachbarschaft abhängt. Schließlich wird versucht, die Gesamtwahrscheinlichkeit über alle Variablen zu maximieren.
Ursprünglich wurde das Prinzip der Textursynthese in der Computergrafik rein zur Texturerzeugung genutzt, heutzutage wird es aber auch auf reguläre Bilder angewendet. Es eignet sich zudem sehr gut, um Löcher in Bildern, bzw. fehlende Bereiche in ihnen, aufzufüllen, sofern das Ursprungsmaterial strukturiert genug ist. Die Textursynthese findet zudem oft, wie früher bereits erwähnt, in der Beispiel-basierten Terraingenerierung Anwendung. Beispielhaft sei (Zhou et al., 2007) genannt. Eine artverwandte Aufgabe, für den sich Methoden der Textursynthese ebenfalls oft eignen, ist der sogenannte Texturtransfer, bei dem zwei Texturen oder allgemein Bilder überlagert werden, so dass im Resultat die Struktur des einen im anderen

möglichst gut widergespiegelt wird, mehr dazu im Abschnitt 2.5.1.
Fast alle Patch-basierten Methoden bestehen nach (Barnes and Zhang, 2017, pp. 2-3) aus zwei Schritten, dem Matching, wo die am besten passenden Bereiche aus dem Beispiel ermittelt werden, und dem Blending, bei dem die neuen Patches sinnvoll angeordnet und vor allem ein möglichst homogener Übergang zwischen ihnen geschaffen werden soll (siehe Abbildung 2.7).

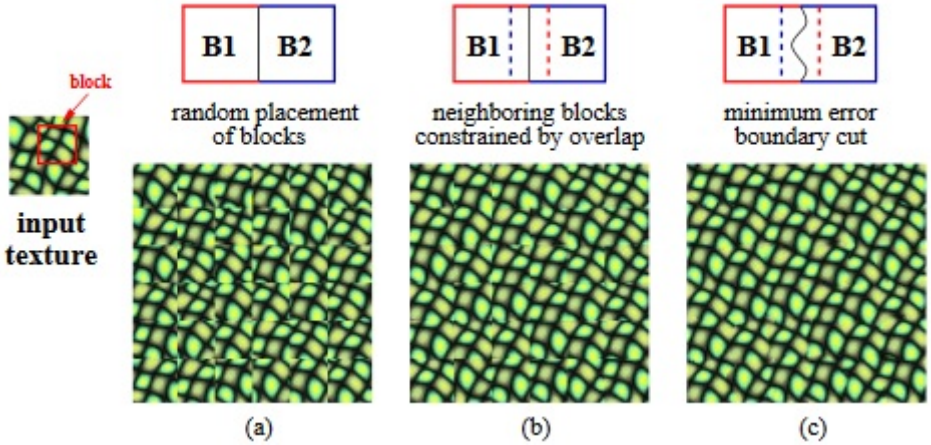


Abbildung 2.7 Links die Ausgangstextur, rechts die daraus synthetisierten, größeren Texturen: (a) eine zufällige Anordnung von Blöcken, (b) sich möglichst homogen überlappende Blöcke, (c) sich überlappende Blöcke mit optimalem Rand/Schnitt. (Efros and Freeman, 2001, p. 2).

Übliche Verfahren dafür sind Blending-Algorithmen, welche in Abschnitt 2.3.2 behandelt werden, oder Techniken zum Ermitteln eines Minimal Error Boundary Cuts. Letztere versuchen, in einem definierten Bereich eine Grenze zwischen (üblicherweise) zwei Bildern zu finden, so dass der „Fehler“ oder die „Energie“, hier vorstellbar als die visuellen Unterschiede, möglichst gering ausfällt. Diese Verfahren werden ausführlicher im Abschnitt 2.3.1 beschrieben.

2.3 Bildverarbeitung und Maschinelles Sehen

Bei der (digitalen) Bildverarbeitung handelt es sich um ein breites Fachgebiet, was sich mit der Analyse und vor allem Manipulation von Digitalbildern beschäftigt. Sie wird in vielen Bereichen zu unterschiedlichsten Zwecken eingesetzt, meist zur Vorverarbeitung oder nachträglichen Bearbeitung von Bildern aller Art. An dieser Stelle soll auch das Feld des maschinellen Sehens nicht unerwähnt bleiben, was der Bildverarbeitung sehr nahe steht und in dem der Schwerpunkt auf der Analyse, Klassifizierung und Erkennung von Bildinhalten liegt. Die beiden Disziplinen gehen fließend ineinander über und viele Methoden werden in beiden angewendet.
Allgemein lassen sich in der Bildverarbeitung Operationen, die das Eingangsbild transformieren, danach unterscheiden, wovon der Wert eines Ausgabespixels abhängt. Bei Punktoperationen hängt jeder Ausgabepixel nur von dem entsprechenden Eingabepixel ab. Bei lokalen

Operationen, sogenannten Filtern, hängt jeder Ausgabepixel dagegen von seinen Nachbarn bzw. einer lokalen Region und bei globalen Operationen von allen Pixeln des Eingangsbildes ab. Im Fall der Filterung wird eine entsprechend dimensionierte Filtermatrix, auch Convolution Matrix oder Filterkernel genannt, von Gewichten über das Bild „geschoben“ und jeder Ausgabepixel ergibt sich durch Verrechnung mit den darunterliegenden Eingabepixeln, wie in Abb. 2.8 dargestellt.

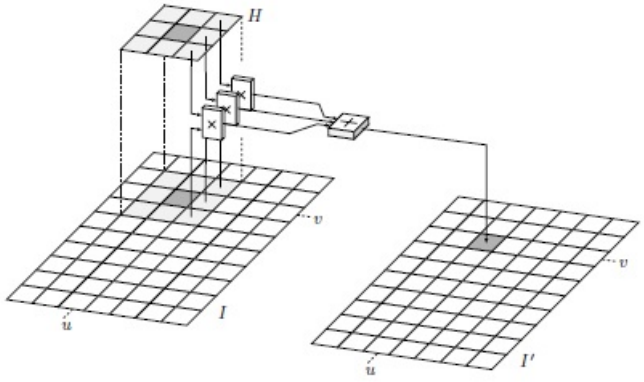


Abbildung 2.8 Filterkernel H wird über das Ausgangsbild geschoben, jeweils die Koeffizienten mit den Pixelwerten multipliziert und ergeben in der Summe den Ausgabepixel (Burger and Burge, 2006, p. 92).

Typische Filteroperationen sind z.B. das Glätten eines Bildes, sprich Entfernen von Rauschen mittels Mittelung, oder das Auffinden von Kanten und anderer sogenannter Features. Ein einfacher Glättungsfilter könnte z.B. einen 3×3 Filterkernel haben, bei dem jedes Element eine Gewichtung von 1/9 besitzt. Der Ausgabewert für den mittleren Pixel würde sich nun durch eine gewichtete Summe der Ursprungsdaten ergeben und entspräche dem Durchschnittswert der Region. Ein etwas komplexerer Glättungsfilter ist der Gaußfilter, bei dem die Gewichte des Kernels einer zweidimensionalen Gaußfunktion folgen. Abbildung. 2.9 zeigt beispielhaft eine Glättungsoperation.

Abbildung 2.9 Links das Ursprungsbild und rechts das Ergebnis nach Anwendung eines Glättungsfilters (Burger and Burge, 2006, p. 92).

Eine weitere Klasse von Filtern sind die morphologischen Filter, die im Stande sind, die Struktur eines Bildes zu verändern. Üblicherweise arbeiten sie auf Binärbildern, es gibt aber auch Verallgemeinerungen auf Grauwertbildern. Die Struktur des Bildes lässt sich durch iteratives Anwenden von den zwei Grundoperationen Erosion und Dilatation und einem hier

Strukturelement genannten Filterkernel erreichen. Die Form des Kernels entscheidet dabei allein über die Transformation. Durch Erosion werden selektiv gemäß dem Strukturelement Bildbereiche „abgetragen“ und bei Dilatation aufgetragen. Morphologische Filter sind nützlich, um z.B. kleine Bildelemente verschwinden zu lassen, Löcher bzw. Regionen zu schließen oder Konturbilder zu erzeugen. Mehr über Bildverarbeitung und speziell Filterung lässt sich in (Burger and Burge, 2006, Kap. 6-8,10) nachlesen.
Eine wichtige Aufgabe der Bildverarbeitung und vor allem des maschinellen Sehens ist die der Segmentierung. Hierbei wird das Bild in verschiedene Bereiche partitioniert, die gewisse Gemeinsamkeiten aufweisen, bzw. um sie von anderen Bildinhalten abzugrenzen. Übliche Methoden zur Segmentierung sind nach (Yuheng and Hao, 2017) z.B. das sogenannte Region Growing und Clustering Techniken, bei dem benachbarte Pixel ausgehend von ihren Eigenschaften zu mehreren Regionen zusammengefügt werden oder die weiter oben bereits erwähnte Kantendetektion.

2.3.1 Graph Cut

Graph-Cut-Algorithmen sind nach (Sinha, 2004, p. 3) eine beliebte Technik des maschinellen Sehens zur Segmentierung. Sie haben aber darüber hinaus auch viele andere Anwendungsgebiete und Ziele. Diese lassen sich in der Regel als Energy-Minimization-Problem formulieren, etwa das Entrauschen eines Bildes. Graph-Cut-Algorithmen werden regelmäßig in der Textursynthese eingesetzt, um eine optimale Grenze zwischen zwei Bildausschnitten zu finden, siehe z.B. (Zhou et al., 2007) und (Kwatra et al., 2003). Dazu wird die Aufgabe, welche oft als MRF (siehe Abschnitt 2.2) modeliert wird, in ein Maximal-Flow/Minimal-Cut-Problem umformuliert.
Das Maximal-Flow-Problem ist ein Graph-basiertes Optimierungsproblem, bei dem ein möglichst großer Fluss zwischen zwei Enden eines Graphens gesucht wird. Nach dem Maximal-Flow/Minimal-Cut-Theorem ist ein maximaler Fluss äquivalent zu einem Schnitt mit minimalen Kosten des Minimal-Cut-Problems, siehe (Sinha, 2004, p. 2). Dabei gibt es zwei spezielle terminale Knoten s und t, die die zu trennenden Seiten repräsentieren und eine endliche Menge regulärer Knoten zwischen ihnen. Jeder Kante wird nach einer Kosten- bzw. lokalen Energiefunktion ein Wert zugeordnet. Anschließend wird durch Algorithmen der kombinatorischen Optimierung ein Schnitt zwischen den Knoten gesucht, der s und t trennt und minimale Kosten aufweist, bzw. den maximalen Fluss über die Grenze ergibt, was einer Minimierung der globalen Energiefunktion entspricht. Diese globale Energiefunktion kann, je nach Problem und Umsetzung, zusätzlich zu der Summe der lokalen Terme auch einen übergeordneten, gebietsorientierten Term enthalten, der z.B. Vorinformationen bzgl. einer Zuordnung enthält, wie in (Soltow, 2010, p. 9) beschrieben.
Im Kontext der Textursynthese bilden, wie (Kwatra et al., 2003, p. 3) erläutern, die Terminalknoten s und t die nichtveränderlichen Patches der Bilder und die regulären Knoten eine überlappende Region von Pixeln. Den Kanten adjazent zu den Terminalknoten werden maximal-hohen Kosten zugewiesen, damit hier kein Schnitt angewendet wird. Für alle ande-

ren Kanten werden die Kosten nach einem Ähnlichkeitsmaß der zugehörigen Pixel berechnet, z.B. ausgehend vom Intensitätsunterschied des Pixelpaars. In Abb. 2.10 wird dies dargestellt.

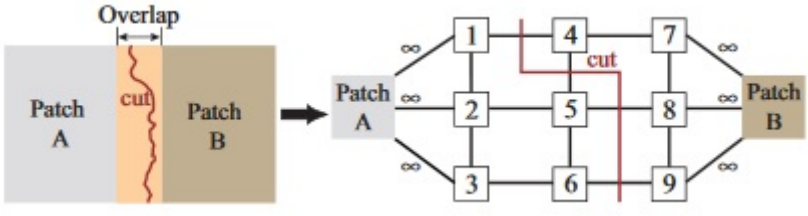


Abbildung 2.10 Links der schematische Aufbau der zwei sich überlappenden Patches A und B und einer idealen Grenze, rechts die Darstellung als Graph. A und B bilden dabei die terminalen Knoten s und t mit den unendlich gewichteten Kanten. (Kwatra et al., 2003, p. 3).

2.3.2 Blending-Techniken

Um in der Computergrafik oder Bildverarbeitung verschiedene Texturen bzw. Bilder zu überlagern und einen mehr oder weniger sanften Verlauf zu erzielen, verwendet man Blending-Techniken. Im Kontext der PTG können sie beispielhaft genutzt werden, um Verläufe zwischen Heightmaps oder verschiedenen Biomen zu erzeugen. Die einfachste Variante ist, wie auch (Abdullah and Agha, 2013, p. 1) erläutern, das sogenannte Alpha Blending, bei dem jeder Pixel des Ausgabebildes aus einer gewichten Summe der jeweiligen Eingabepixel besteht. Dabei werden die Pixelwerte der Eingabebilder nach einem Alphawert gewichtet, der in 2D oft als Maske vorliegt. Ein Pixel berechnet sich demnach durch

$$C = \text{alpha} \cdot x + (1 - \text{alpha}) \cdot y \tag{2.1}$$

Abbildung 2.11 veranschaulicht das Prinzip.

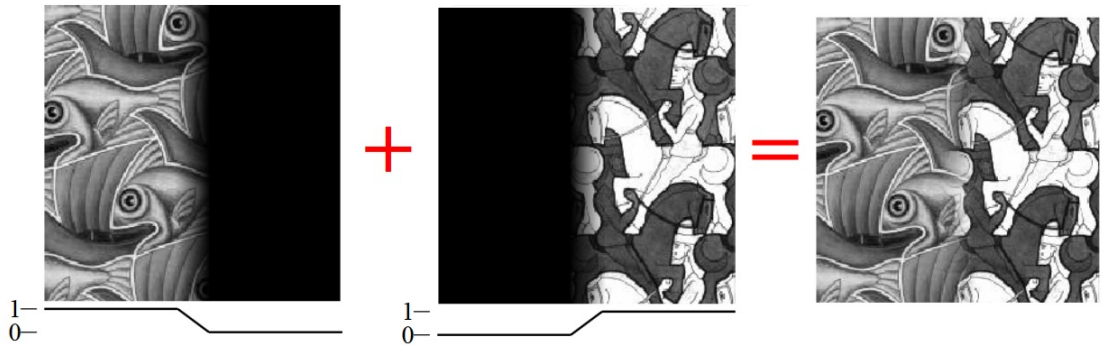


Abbildung 2.11 Zwei Bilder werden anhand eines Alphawertes überlagert. Dieser ist in den Kurven unter den Bildern angedeutet. (Efros, 2005, p. 15).

Die Wahl der Änderungsrate des Alphawertes, bzw. die Distanz im Bild über welche er sich ändert, ist je nach Fall zu wählen und hat großen Einfluss auf das visuelle Ergebnis, indem es bestimmt, wie hart oder weich der Übergang ist. Ein Problem ist, dass verschiedene

Frequenzanteile in den Bildern unterschiedlich große Übergangsbereiche für ein visuell ansprechendes Ergebnis benötigen. Bei breiten, weichen Übergängen gehen z.B. hohe Frequenzdetails verloren. Mittels Multi-Band- oder Pyramid-Blending-Techniken, welche später genauer beschrieben werden, kann dieses Problem umgangen werden. Basierend auf der Tatsache, dass Menschen Diskontinuitäten der Bildgradienten stärker wahrnehmen als Änderungen der reinen Intensitäten, siehe (Pérez et al., 2003, p. 313), wurden zudem Gradienten-basierte Blending-Algorithmen entworfen. Einer davon, Poisson Blending, wird in einem noch folgenden Teilabschnitt vorgestellt.

Laplacian Pyramid Blending

Laplacian Pyramid Blending ist ein Blending-Algorithmus, der verschiedene Blending-Stärken auf unterschiedliche Frequenzbereiche anwendet. Vorgestellt wurde das allgemeine Pyramiden-basierte Verfahren von (Burt and Adelson, 1983). Mit ihm können niedrige Frequenzen über einen größeren Bereich geblendet werden, während höhere Frequenzanteile über eine kürzere Distanz geblendet werden und so besser erhalten bleiben, wie in (Abdullah and Agha, 2013, p. 1) nachgelesen werden kann. Um dies zu erreichen muss zuerst eine Gauß-Pyramide von der Alphamaske erstellt werden. Eine Gauß-Pyramide besteht aus einer Reihe von aufeinanderfolgend Gauß gefilterten und um Faktor 2 herunter skalierten Versionen eines Bildes. Danach müssen für beide Eingangsbilder Laplace-Pyramiden erstellt werden. Diese sind der Gauß-Pyramide ähnlich, aber jedes Level bildet nur die Differenz zum vorherigen Level ab und enthält so nur einen Teil des Frequenzspektrums, vgl. Abb. 2.12.

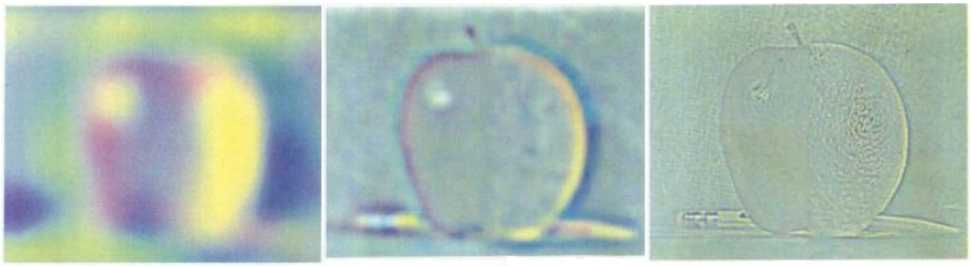


Abbildung 2.12 Geblendete Laplace-Pyramiden zweier Bilder (jeweils linke und rechte Hälfte) für die Pyramiden-Level 4,2,0 (Efros, 2005, p. 27).

Anschließend wird eine gemeinsame Laplace-Pyramide, mithilfe der Gauß-Pyramide der Alphawerte, über die bekannte Alpha-Blending-Formel berechnet. Schließlich wird diese Pyramide wieder zu einem einzelnen Bild skaliert und kombiniert. Abb. 2.13 zeigt so ein Ergebnis beispielhaft.

Abbildung 2.13 Zwei Bilder werden mittels Laplacian Pyramid Blending überlagert (Efros, 2005, p. 28).

Poisson Blending

Poisson Blending ist eine Gradienten-basierte Blending-Technik, zuerst vorgestellt von (Pérez et al., 2003), um Bildausschnitte möglichst homogen in anderen Bildern zu platzieren. Die Technik basiert darauf, die Gradienten der Farbwerte des Bildausschnittes zu nutzen, um neue Pixelwerte zu berechnen, welche die ursprünglichen, relativen Unterschiede widerspiegeln und so die ehemalige Struktur beibehalten, zum anderen aber auch einen möglichst sanften Übergang zum Rand des neuen Bildes ergeben. Die Pixelwerte aus dem Zielbild färben schließlich nach und nach vom Rand aus gemäß den Gradienten den eingefügten Bildausschnitt. Abbildung 2.14 illustriert den Aufbau.

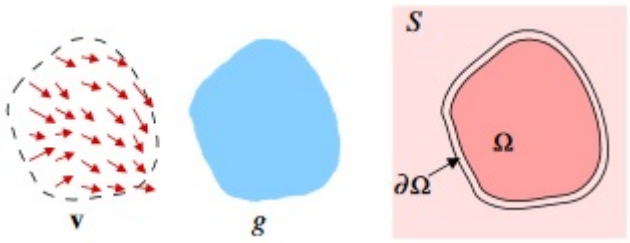


Abbildung 2.14 Der Bildausschnitt Ω des Zielbildes wird mittels der Gradientenvektoren v aus dem ursprünglichen Bildausschnitt g vom Rand $\delta\Omega$ ausgehend interpoliert (Pérez et al., 2003, p. 314).

Dies geschieht mithilfe eines Gradientenvektorfeldes und einer spärlich gefüllten Koeffizientenmatrix, die eine zu lösende Poisson-Gleichung bilden. Dabei dient jeder Pixel im zu blendenden Bereich als eine Unbekannte. Die Gradienten und Koeffizienten werden durch den diskreten Laplace-Operator bestimmt. Es ist hier zudem möglich, pro Pixel zu wählen, aus welchem Bild der Gradient stammt oder ob gar eine Kombination gewählt werden soll. Dies ermöglicht es auch, die Strukturen beider Bilder zu erhalten. Eine Variante ist es, den Gradienten mit dem höheren absoluten Wert zu wählen.
Poisson Blending findet meist in der einen oder anderen Variante in der Textursynthese Anwendung, um die Patch-Grenzen zu verbergen, wie z.B. in (Zhou et al., 2007) und (Abdullah and Agha, 2013). Über die Zeit wurde die Technik von anderen Autoren verbessert, um unter anderem die ursprünglich hohen Speicher- und Berechnungskosten zu reduzieren. (Szeliski

et al., 2011) nutzen dazu etwa Multi-Splines und kombinieren es mit einem kleinen Anteil von Laplacian Pyramid Blending.

2.4 A* Suche

A* ist ein bekannter Algorithmus der informierten Suche in Graphen, vorgestellt von (Hart et al., 1968). Suchalgorithmen suchen, ausgehend von einem Startzustand, in einem Zustandsraum bestimme Zielzustände. Werden dabei Informationen über den Raum und dessen Zustände bzw. eine Heuristik verwendet, nennt man dies informierte Suche. Ohne Zusatzinformationen ergibt sich eine uninformierte Suche. Eine Heuristik ist eine Strategie oder auch Schätzfunktion, um mit begrenztem Wissen und Aufwand Näherungen des optimalen Ergebnisses zu erhalten. Hier erleichtert sie das Auffinden der Lösung. A* wird oft zur Wegfindung eingesetzt, um den kürzesten oder schnellsten Weg von A nach B zu finden. Eine typische Heuristik ist dabei die Distanz zum Ziel als Luftlinie. Für diese Arbeit wäre z.B. eine Verwendung zur Erzeugung von Flüssen denkbar. Der Ablauf des Algorithmus ist so, dass nach und nach vom Startknoten aus ein Suchbaum aufgebaut wird, indem schrittweise nach einer Kostenfunktion

$$f(x) = g(x) + h(x) \tag{2.2}$$

der nächstbeste der bekannten, angrenzenden, also bereits explorierten, Knoten expandiert und evaluiert wird, bis das Ziel erreicht ist. $g(x)$ sind die Kosten vom Start bis zum aktuellen Knoten x und $h(x)$ die nach der Heuristik geschätzten Kosten zum Ziel.
A* findet garantiert die optimale Lösung, wenn die Heuristik zulässig ist, also die Kosten nie überschätzt. Ist die Heuristik zusätzlich monoton, was bedeutet, dass die geschätzten Kosten von einem Knoten k zum Ziel kleiner oder gleich den tatsächlichen Kosten zu einem Nachfolger k' sowie den von dort aus geschätzten Kosten sind, muss kein bereits expandierter Knoten erneut geprüft werden. Dies steigert die Effizienz.

2.5 Machine Learning

Wie (Goodfellow et al., 2016, pp. 66-105) beschreiben, ist Machine Learning ein Teilbereich der künstlichen Intelligenz, bei dem statistische Methoden angewendet werden, um automatisch anhand bestehender Beispieldaten zu „lernen“. Lernen bedeutet in diesem Zusammenhang, die Performanz oder Güte des Ergebnisses bzgl. eines definierten Maßes zu verbessern. Die Algorithmen erkennen während einer Lernphase Gesetzmäßigkeiten und Muster in den Beispielen, generalisieren diese und können das Wissen anschließend auf unbekannte Daten anwenden. So könnten diese Methoden z.B. anhand von realen Terrains/Biomen angelernt werden und neue, vergleichbare erzeugen. Typische Aufgaben sind Klassifikation und Regression, Vorhersage oder Clustering. Algorithmen des Machine Learnings lassen sich in drei Kategorien einteilen: überwachtes Lernen, bei dem die korrekten Ergebnisse zu den Beispiel-

daten während des Lernens zur Verfügung stehen, unüberwachtes Lernen und das bestärkende Lernen, bei dem nur durch nachträgliches Feedback gelernt wird.

2.5.1 Neuronale Netze

Neuronale Netze sind Techniken des Maschinellen Lernens, welche grob die biologischen neuronalen Netze im Gehirn und ihre Informationsverarbeitung nachbilden und gut dafür geeignet sind, großen Datensätzen zu verarbeiten. Sie bestehen aus einer Menge von verbundenen und in Schichten angeordneten Knoten, die Neuronen repräsentieren. Die erste Schicht nennt sich Input Layer, die letzte Output Layer und alle dazwischen liegenden sind die sogenannten Hidden Layer. Gibt es davon keine, spricht man von einem Perceptron(-Netzwerk), sind es dagegen viele Schichten, spricht man von Deep Learning. In Abb. 2.15 wird die Struktur eines simplen Neuronales Netzes illustriert.

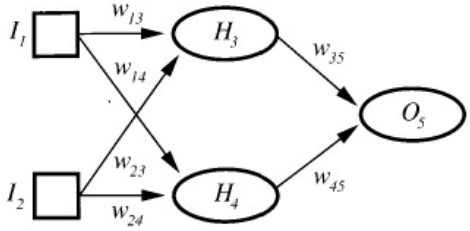


Abbildung 2.15 Neuronales Netz mit je einem Input, Hidden, und Output Layer. Die jeweiligen Knoten sind mit I, H und O markiert, W steht für die Gewichte. (Russell and Norvig, 2009, p. 572).

Jeder Knoten berechnet seinen Output über eine üblicherweise nichtlineare Aktivierungs- bzw. Transferfunktion, die auf die gewichtete Summe seiner Inputs angewendet wird. Zudem gibt es in der Regel noch einen Schwellwert, meistens abgebildet durch einen ebenfalls gewichteten Bias(-Knoten pro Layer), um die Flexibilität zu erhöhen. Abb. 2.16 zeigt einen Knoten und seine Bestandteile.

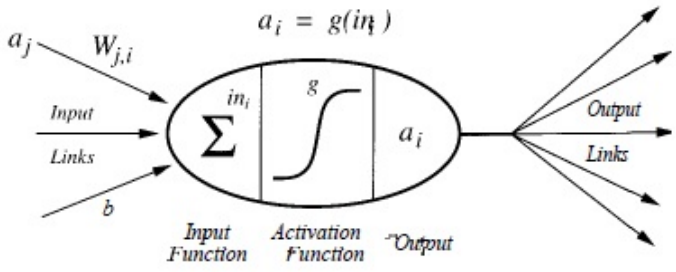


Abbildung 2.16 Aufbau eines künstlichen Neurons. Die Input-Funktion berechnet die gewichtete Summe der einzelnen Inputs und des Bias, die Aktivierungsfunktion berechnet daraus dann den Output des Knotens. (Russell and Norvig, 2009, p. 568).

Der Ablauf eines sogenannten Feed-Forward-Netzes ist typischerweise so, dass Rohdaten an

den Input Layer übergeben werden, diese Daten werden gemäß des Knotennetzes durch die Hidden Layer verarbeitet, weitergeleitet und erreichen schließlich den Output Layer. Hier wird das Ergebnis während der Lernphase nach einer Kostenfunktion bewertet und der Fehler mittels Back Propagation zurück durch das Netz propagiert, um die Gewichte der Knoten gemäß ihrem Beitrag zum Ergebnis anzupassen und letztendlich den Fehler durch eine Gradient-Descent-Optimierung zu minimieren. Dieser Vorgang wird wiederholt, bis die Lernphase abgeschlossen ist.
Allgemein lassen sich Neuronale Netze in generative und diskriminative Netze einteilen. Erstere versuchen, die zugrundeliegende Wahrscheinlichkeitsverteilung zu lernen und sind dadurch in der Lage, neue, ihr folgende Daten zu generieren, während letztere direkt versuchen, eine Lösung, etwa eine Klassifizierung, zu finden. (Russell and Norvig, 2009, pp.563,567-583).

Convolutional Neuronal Networks Eine spezielle Art von Neuronalen Netzen zur Verarbeitung von großen, gitterförmigen Datensets sind die die Convolutional Neural Networks (CNNs). Sie werden oft in der Bildverarbeitung und dem maschinellen Sehen eingesetzt, z.B. zur Objekterkennung, und verwenden eine Kombination von sogenannten Convolution und Pooling Layern. Erstere führen eine Faltung, ähnlich den Filtern in der Bildverarbeitung, auf den Eingabedaten durch und letztere verwerfen überflüssige Informationen. Ihr Aufbau wird in Abb. 2.17 illustriert.

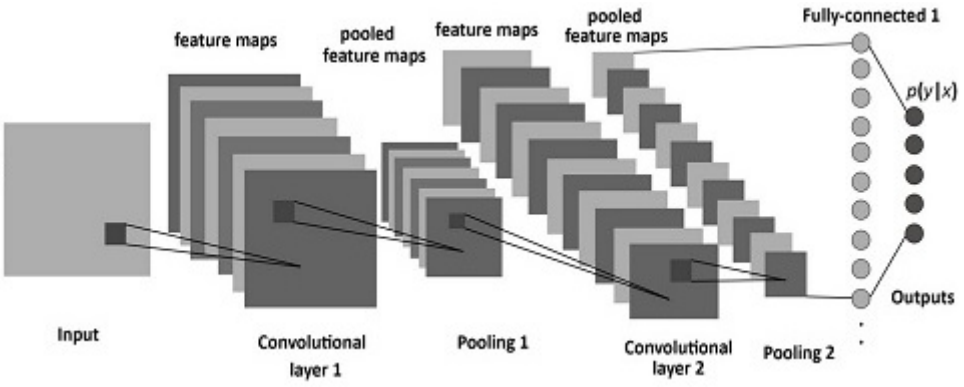


Abbildung 2.17 Aufbau eines CNN aus mehreren, sich abwechselnden Convolution und Pooling Layern, gefolgt von einem letzten Fully Connected Layer. (Albelwi and Mahmood, 2017, 5).

Entgegen regulärer Neuronaler Netze sind CNN Layer dreidimensional, was der Verarbeitung von (RGB-)Bildern als Eingabe geschuldet ist. Abb. 2.18(links) zeigt, wie jeder Convolution Layer dabei aus einem Volumen von künstlichen Neuronen besteht, die in Höhe und Breite nur entsprechend der Filterkerngröße mit dem vorherigen Layer verbunden sind. In die Tiefe bildet jede Schicht innerhalb des Convolution Layers im Prinzip einen eigenständigen Filter, der darauf trainiert wird, nach bestimmten Dingen zu suchen.

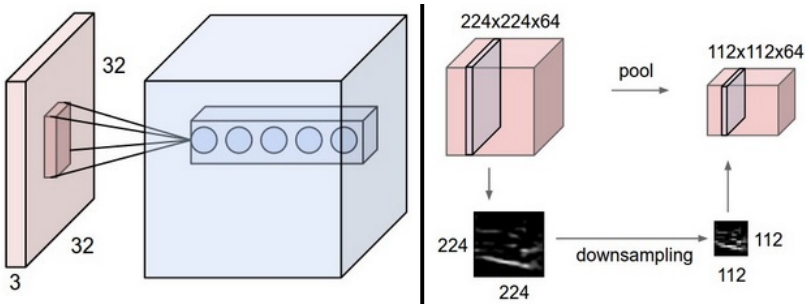


Abbildung 2.18 Links: Convolution Layer mit einer Tiefe von fünf Neuronen/Filtern. Rechts: Pooling Layer, der ein 2×2 Subsampling durchführt. (Karpathy, 2018).

Durch die lokale Konnektivität der Convolution Layer und geteilte Gewichte ergeben sich deutlich weniger Parameter und damit eine wesentlich effizientere Verarbeitung. Die Pooling Layer führen jeweils ein Subsampling in Breite und Höhe durch und verdichten so die Daten, wie in Abbildung 2.18 dargestellt wird. Weitere Informationen sind in (Goodfellow et al., 2016, pp. 326-339) zu finden.
Eine für diese Arbeit sehr interessante Anwendung von CNNs ist der sogenannte **Style Transfer**, vorgestellt von (Gatys et al., 2015) bzw. (Gatys et al., 2016). Dabei wird ein zur Bilderkennung trainiertes CNN (VGG-19) aus 16 Convolution und drei Pooling Layern genutzt, um den Stil eines Bildes mit dem semantischen Inhalt eines zweiten zu kombinieren, siehe 2.19. Was dort als Style Transfer bezeichnet wird, ist vom Konzept eine Art der Textur- bzw. Bildsynthese.

Abbildung 2.19 Style Transfer: Der künstlerische Stil des kleinen, mittleren Bildes wird mit dem Inhalt des linken kombiniert und ergibt das rechte Bild (Gatys et al., 2016, p. 2418).

Generative Adversial Networks Eine weitere wichtige Art der Neuronalen Netze, welcher in letzter Zeit viel Beachtung geschenkt wurde und welche ausführlicher in (Goodfellow et al., 2016, pp. 696-699) erläutert wird, sind die sogenannten Generative Adversial Networks (GANs). Sie basieren darauf, zwei unterschiedliche Neuronale Netze gegeneinander antreten zu lassen, mit dem Ziel, neue Daten, etwa Bilder, zu generieren, die den Beispieldaten entsprechen. Dabei produziert das erste, generative Netzwerk Samples, welche möglichst ähnlich

der Trainingsdaten sind und das zweite, als Diskriminator bezeichnete Netzwerk, hat die Aufgabe, sie von den Beispieldaten zu unterscheiden. Dieses Wechselspiel, bei dem beide Netze gleichzeitig auf ihre Aufgabe trainiert werden, geht im Idealfall solange vor sich hin und her, bis der Diskriminator die generierten Daten nicht mehr unterscheiden kann, was bedeutet, dass das generative Netz entsprechend gute Sample generieren kann. Diese Konvergenz tritt in der Praxis aber nicht immer ein.
Vorgestellt wurde dieses Konzept von (Goodfellow et al., 2014), anschließend wurden von vielen weiteren Autoren damit Experimente gemacht, versucht die GANs für viele spezielle Aufgaben zu nutzen und sie ggf. modifiziert und angepasst. So bauen z.B. (Ledig et al., 2016) auf ihnen auf und nutzen eine modifizierte Variante, die sie Super Resolution Generative Adversial Network (SRGAN) nennen für „Photo-Realistic Single Image Super-Resolution“ mit hohen Upscaling-Faktoren (4x). (Radford et al., 2015) kombinierten CNNs mit GANs und schufen damit die Klasse der Deep Convolutional Generative Adversial Networks (DCGANs), bei denen sowohl Generator- als auch Diskriminator-Netzwerk explizit ein CNN sind und damit das System speziell geeignet ist für die Generierung von Bildern. Kürzlich verwendeten (Beckham and Pal, 2017) und (Wulff-Jensen et al., 2018) DCGANs konkret zur PTG, indem das Neuronale Netz, angelernt durch DEMs, Heightmaps erzeugt. (Guérin et al., 2017) hingegen nutzen eine Reihe von conditional Generative Adversial Networks (cGANs), bei denen Generator und Diskriminator durch zusätzliche Eingaben konditioniert werden, um aus vor allem Nutzersketches realistische Heightmaps von Terrain zu generieren.

2.6 Physikalisch-basierte Simulation natürlicher Prozesse

Dieser Abschnitt widmet sich Methoden der physikalisch-basierten Simulation und anderer komplexer Systeme, die auch zur PCG eingesetzt werden.

Sollen natürliche Prozesse auf der Erde mit gewisser Präzision nachgebildet oder vorhergesagt werden, kommen meist sehr komplexe Modelle zum Einsatz, die auf den entsprechenden physikalischen Grundlagen beruhen und diese meist durch mathematische Formeln beschreiben. Je komplexer die zu simulierenden Vorgänge sind und je höher die gewünschte Präzision ist, desto höher ist auch der Rechenaufwand. Es kommt nicht selten vor, dass extrem leistungsfähige Supercomputer zum Einsatz kommen und/oder die Berechnungen mehrere Stunden oder gar Tage dauern.
Mit zu den bekanntesten Anwendungsgebieten dürften wohl die Klima- und Wettersimulationen zählen. In beiden Fällen wird ein meteorologisches Modell als Grundlage genommen, was numerisch die relevanten Gleichungen für jeden Zeitschritt löst. Lokale Modelle bieten mehr Präzision, globale umfassen dafür einen größeren räumlichen Bereich, bis hin zum ganzen Planeten. Ein Klimamodell bildet in seiner Gesamtheit im Gegensatz zur Wettersimulation deutlich mehr relevante Größen und Prozesse ab, wie z.B. die Ozeanzirkulation oder Dynamik von Eisschilden in den Polregionen. Dazu wird es oft durch entsprechende Sub-Modelle ergänzt. Einige Grundlagen über das Klima und relevante Vorgänge werden im folgenden Ab-

schnitt erläutert. Andere physikalisch-basierte Modelle, die Teil eines Klimamodells sein oder auch gesondert zur Anwendung kommen können, simulieren des Weiteren z.B. das korrekte Verhalten von Flüssigkeiten oder Gasen (Fluid- und Strömungssimulation) oder geologische Prozesse wie Plattentektonik und Erosion. Dies wird in den Abschnitten 2.6.2, 2.6.3 und 2.6.4 behandelt. In Abschnitt 2.7 wird ausführlich beschrieben, wie darüber hinaus in der Ökologie Populationen von Flora oder Fauna und teils ganze Ökosysteme simuliert werden.
In der Computergrafik, speziell Echtzeitanwendungen und Computerspielen und abseits wissenschaftlicher Anwendungen, werden aus Gründen der Performanz und Relevanz oft nur gewisse Teilbereiche simuliert und/oder nur oberflächlich in ihren Grundzügen, wobei die Verbreitung mit steigender allgemein zu Verfügung stehender Rechenleistung zunimmt.

2.6.1 Grundlagen des Klimas

Die folgenden Grundlagen basieren überwiegend auf (Goosse et al., 2008, Kap. 1). Zunächst werden einige wichtige Begriffe wie Klima, Biom und Ökosystem definiert, anschließend relevante Größen des Klimas und vor allem der Biomverteilung wie Temperatur und Niederschlag und zugehörige Prozesse und Wechselwirkungen erläutert. Beispielhaft wird auf Konvektion und atmosphärische Zirkulation sowie Winde eingegangen.

Das Klima beschreibt den durchschnittlichen Zustand relevanter meteorologischer Größen wie Wind, Niederschlag und Temperatur an einem bestimmten Ort und ihre Schwankungen über einen langen Zeitraum vieler Jahre. Ein Klimasystem, was durch ein Klimamodell in Gänze oder nur zum Teil simuliert wird, umfasst des Weiteren Komponenten wie Hydrosphäre, Kryosphäre, Pedosphäre und Biosphäre, womit der Einfluss von Wasser, Eis, Land und Lebewesen sowie ihre Prozesse berücksichtigt werden.
Die in einem großen Gebiet vorherrschende Gemeinschaft von Fauna und vor allem Flora und ihre gemeinsamen Ansprüche an die Umgebung und das Klima nennt man Biom. Die genaue Definition und vor allem Kategorisierung variiert allerdings je nach Quelle. Beispielsweise schreibt (Campbell, 1996) von „[...] the world's major communities, classified according to the predominant vegetation and characterized by adaptations of organisms to that particular environment“. Vor allem die Abgrenzung zu einem Ökosystem fällt nicht immer leicht. Ein Ökosystem beschreibt ebenfalls die Organismen und ihre Umwelt, bezogen auf ein definiertes Gebiet, legt aber einen größeren Schwerpunkt auch auf die Wechselwirkungen mit der physischen Umwelt. In (Vertragsparteien, 1992, Art. 2) wird es als „[...] dynamischer Komplex von Gemeinschaften aus Pflanzen, Tieren und Mikroorganismen sowie deren nicht lebender Umwelt, die als funktionelle Einheit in Wechselwirkung stehen“ beschrieben und (Encyclopaedia Britannica, 2018a) definiert es als „[...] the complex of living organisms, their physical environment, and all their interrelationships in a particular unit of space“. Zudem kann ein Ökosystem in beliebiger Größe betrachtet werden, meist bildet es einen deutlich kleineren Raum als ein Biom ab. In der Regel lässt sich also sagen, dass das Klima maßgeblich die Biome beeinflusst, welche wiederum aus vielen gleichartigen Ökosystemen bestehen.

Die wichtigsten Faktoren des Klimas, die die Biome und ihre Verteilung über die Erde beeinflussen, sind die Temperatur und der Niederschlag. Die Temperatur der Erdoberfläche hängt vor allem von der Höhe der eintreffenden Wärmestrahlung durch die Sonne ab, die um den Äquator am größten ist. Die Erdoberfläche strahlt die Wärme wiederum ab und erwärmt so ebenfalls die oberflächennahen Luftschichten. Weiterhin relevant für die Temperaturverteilung sind vor allem atmosphärischer und ozeanischer Wärmetransport sowie die Höhe über dem Meeresspiegel, denn bis zu etwa 11 Kilometern Höhe sinkt die Temperatur durchschnittlich um etwa 6,5 Grad Celsius pro Kilometer. Dafür verantwortlich sind überwiegend die mit der Höhe abnehmende Wärmerückstrahlung der Erdoberfläche und der ebenfalls abnehmende Luftdruck sowie Konvektionsprozesse.
Nahe der Erdoberfläche ist der Luftdruck am größten. Das hat zur Folge, dass die Luft hier mehr Wärmeenergie und Feuchtigkeit, also durch die Wärmestrahlung verdunstendes Wasser (Evaporation), und dadurch wiederum noch mehr Wärmeenergie aufnehmen kann als in höheren Luftschichten mit geringerem Druck. Steigt die feuchtere, erwärmte und dadurch leichter gewordene Luft auf, führt das durch den sinkenden Druck zum einen zu einer stetigen Temperaturabnahme (adiabatische Kühlung) und zum anderen zu einem geringeren relativen Wasserdampfgehalt, wodurch die Luft weniger Wärmestrahlung absorbieren kann. Weiterhin nimmt mit abnehmendem Druck und abnehmender Temperatur die maximal zu haltende Menge an Wasserdampf ab, was zu einer Übersättigung und damit zur Kondensation zu Wasser oder Eis, Wolkenbildung und schließlich Regen führen kann. Bei der Kondensation wird ebenfalls Wärmeenergie abgegeben. Kältere, trockenere und somit schwerere Luft sinkt gleichzeitig ab, um die aufgestiegene Luft zu ersetzen und den Kreislauf zu schließen. Diesen Prozess nennt man Konvektion. In Abbildung 2.20 werden diese Prozesse illustriert.

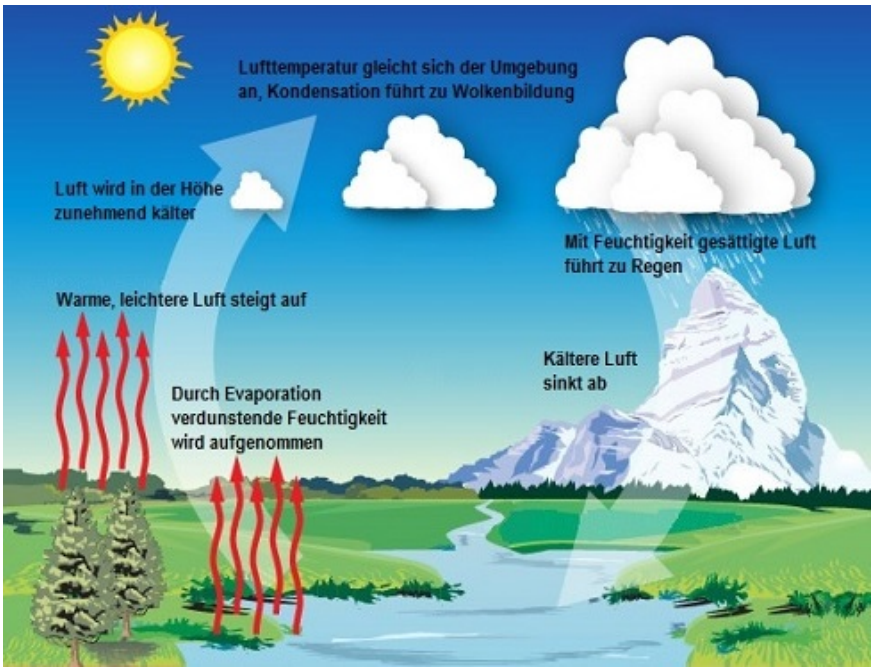


Abbildung 2.20 Wärme- und Feuchtigkeitstransport durch Konvektion in der Atmosphäre (National-Weather-Service, 2017).

Diese Konvektionsprozesse sind auch verantwortlich für die atmosphärische Zirkulation von Luftmassen und Erzeugung von Winden, welche wiederum für den Wärme-, aber auch Feuchtigkeitstransport und letztendlich Niederschlag verantwortlich sind und so eine große Rolle für die Biomverteilung spielen. Im Allgemeinen entsteht Wind durch Druckunterschiede, Luftmassen bewegen sich von Regionen mit hohem Druck (Hochdruckgebieten) in Regionen mit niedrigerem Druck (Tiefdruckgebieten), bis ein Ausgleich stattgefunden hat. Diese Druckunterschiede kommen in der Regel durch räumlich unterschiedlich starke Erwärmung und anschließende Konvektion zustande. Im Fall der globalen Zirkulation ist es vor allem die stärkere Erwärmung der Äquatorregion im Vergleich zu den Polregionen, wodurch mehrere, große und relativ stabile Tief- und Hochdruckgebiete und daraus Konvektionskreisläufe entstehen. Diese Kreisläufe nennt man Konvektionszellen. Auf der Erde gibt es in jeder Hemisphäre die Polarzelle, Ferrel-Zelle und die Headley-Zelle. Auch spielt die Rotation der Erde eine entscheidende Rolle, denn sie sorgt für eine zusätzliche seitliche Ablenkung der Winde. Abbildung 2.21 veranschaulicht die globale Zirkulation in der Atmosphäre und die Anordnung der Konvektionszellen.

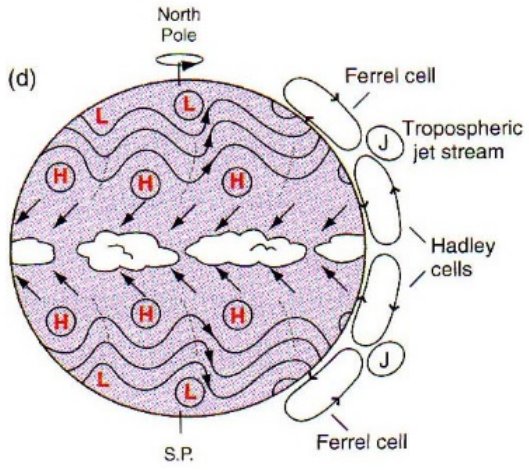


Abbildung 2.21 Globale atmosphärische Zirkulation, H und L stehen für Hoch-und Tiefdruckgebiete (Wallace and Hobbs, 2006, pp. 484).

Es gibt zudem lokale Winde, die durch das Terrain, wie z.B. Berge oder Küstenregionen, beeinflusst werden. Ein Beispiel dafür, wie durch Winde die Niederschlagsverteilung und damit Biome beeinflusst werden, ist das Phänomen des Regenschattens, wie er in (Stockham et al., 2018, p. 1) beschrieben und in Abb. 2.22 illustriert wird.

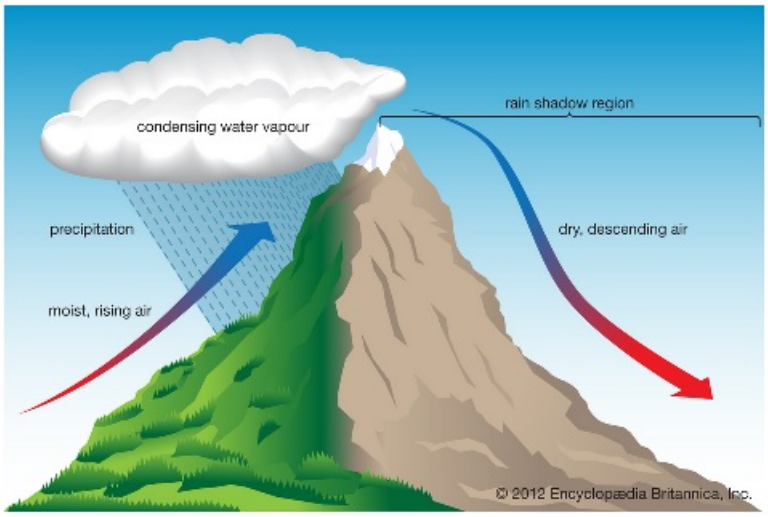


Abbildung 2.22 Hohe Niederschläge durch Orographic Lift auf einer Seite eines Gebirges und infolgedessen Regenschatten auf der anderen (Encyclopaedia Britannica, 2012b).

Trifft (warmer, feuchter) Wind auf ein Gebirge, wird er zum Großteil in die Höhe abgelenkt, um das Hindernis zu überwinden (Orographic Lift), wodurch sich die Luft abkühlt und es, wie weiter oben beschrieben, zu größerem Niederschlag kommt. Auf der anderen Seite des Gebirges sinkt die nun trockene Luft ab und es gibt mangels Feuchtigkeit kaum noch Niederschläge. Abbildung 2.23 zeigt ein reales Beispiel.

Abbildung 2.23 U.a. das kantabrische Gebirge in Nordspanien blockiert die atlantischen Winde (rote Linie), so dass auf der nordwestlichen Seite viele Regenfälle nieder gehen, während die südlichen Regionen sehr trocken sind (Google-Maps, 2018).

Zur letztendlichen Bestimmung der Biome bietet sich die oft genutzte Klassifizierung nach Whittacker an, der, wie viele andere, als maßgebende abiotische (unbelebte) Umweltfaktoren Temperatur und Niederschlag identifizierte. Andere abiotische Faktoren wären z.B. verfügbare Mineralstoffe oder Licht. Biotische Faktoren hingegen sind belebt, etwa Beutetiere oder Konkurrenz. Abb. 2.24 zeigt Whittackers Einteilung der Biome in Abhängigkeit der gewählten Faktoren.

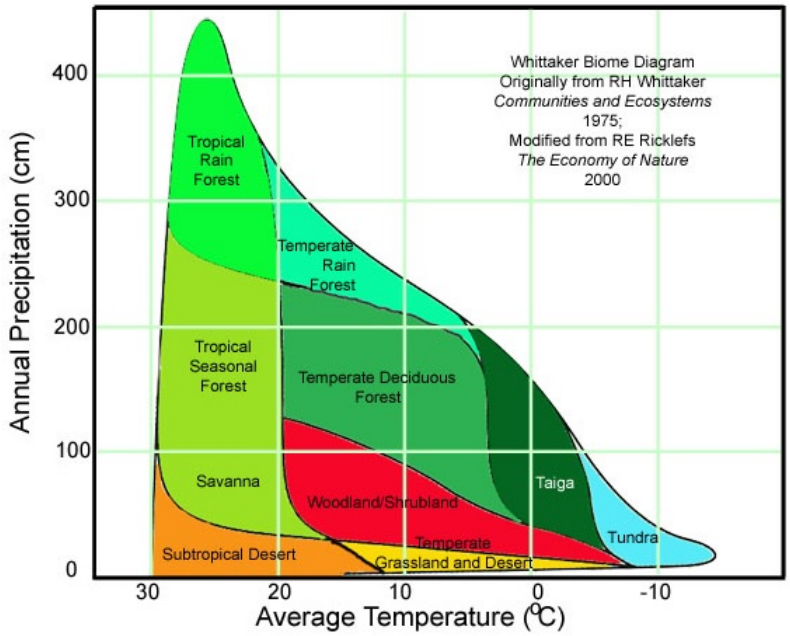


Abbildung 2.24 Biomklassifizierung anhand von Temperatur und Niederschlag nach Whittacker, modifiziert von Ricklefs (Ricklefs, 2000).

Andere Autoren definieren teils mehr, meist deutlich weniger Biome. Auch die genaue Abgrenzung, wo ein Biom aufhört und das nächste anfängt, ist zu einem gewissen Maß arbiträr gewählt, da sie in der Natur fließend ineinander übergehen. Abbildung 2.25 zeigt eine mögliche Einteilung der Erde in Biome.

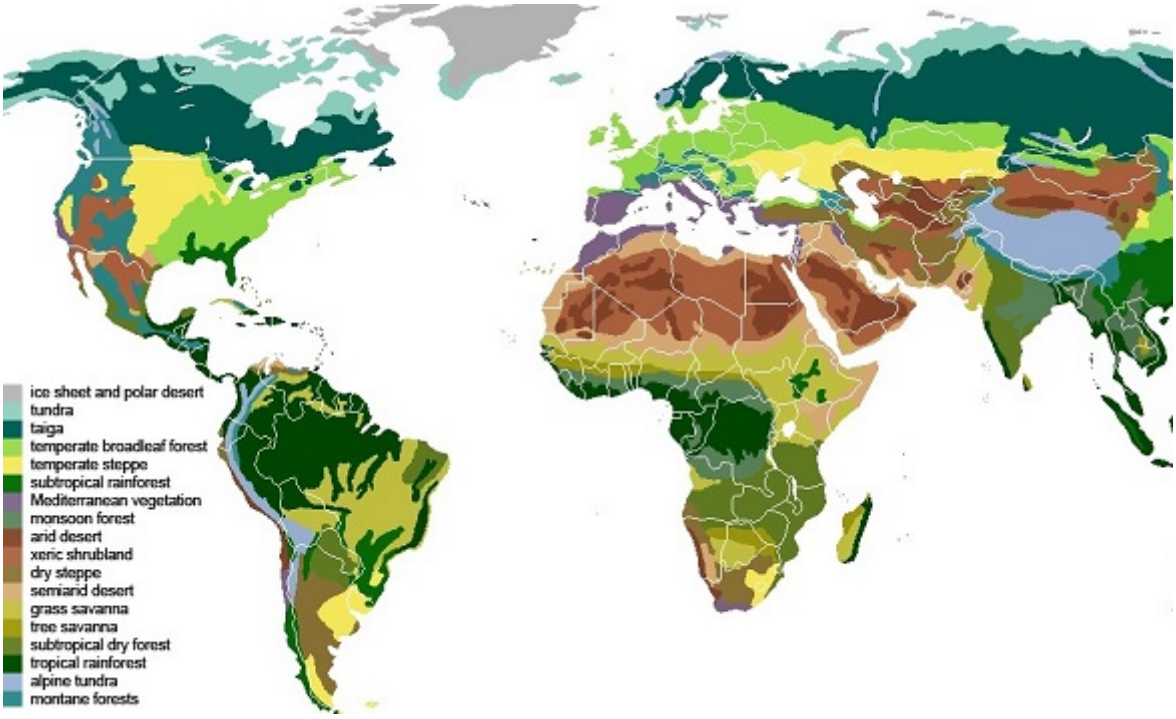


Abbildung 2.25 Ausschnitt der Erde, farbkodiert nach Biomen (Koistinen, 2007).

In der Computergrafik und in Computerspielen, die große Landmassen oder Welten beinhalten, einige Beispiele dazu werden in 3.1 genannt, spielen Biome oft eine zentrale Rolle für die Darstellung und konkrete Repräsentation der Gebiete. Entsprechend ist die Verteilung der Biome eine wichtige Aufgabe. Die Möglichkeiten dazu sind vielfältig, aber oft wird auf eine Kombination aus Temperatur und Niederschlag, wie es oben beschrieben ist, zurückgegriffen. Wie diese beiden Größen jedoch ermittelt werden, variiert stark.

2.6.2 Fluiddynamik

Die Fluiddynamik beschäftigt sich mit dem Verhalten und der Bewegung von Fluiden, sprich Flüssigkeiten und Gasen. In vielen Bereichen ist die Simulation dieser von großer Bedeutung, z.B. zur Ermittlung des Strömungsverhaltens von Flugzeugen oder der Simulation des Windes zur Wettervorhersage. Die Fluiddynamik beruht auf einer Reihe von Formeln und Erhaltungssätzen, elementar sind vor allem die Navier-Stokes-Gleichungen, die ein präzises, mathematisches Modell der Strömung von viskosen Flüssigkeiten und Gasen bilden. Dazu werden Geschwindigkeit, Druck, Dichte, Viskosität und äußere Kräfte betrachtet. Wie (Harris, 2003, pp. 10-12) schreibt, sind oft gemachte Vereinfachungen die Annahme eines inkompressiblen, homogenen Fluids, bei dem die Dichte konstant ist oder das Vernachlässigen der Viskosität, wodurch die Gleichungen den sogenannten Euler-Gleichungen der Strömungsmechanik entsprechen.
Die in der Computergrafik am häufigsten verwendeten Navier-Stokes-Formeln für inkompressible Fluide lauten nach (Harris, 2003, pp. 10-12):

$$\frac{\partial \mathbf{u}}{\partial t} = -(\mathbf{u} \cdot \Delta)\mathbf{u} - \frac{1}{\rho}\Delta p + \nu \Delta^2 \mathbf{u} + \mathbf{F} \tag{2.3}$$

$$\Delta \cdot \mathbf{u} = 0 \tag{2.4}$$

Dabei wird in Gleichung 2.3 das Fluid über ein Geschwindigkeitsvektorfeld $\mathbf{u}$ und ein Skalarfeld für den Druck p abgebildet, welche beide abhängig von der Zeit t sind. ρ ist die (konstante) Dichte, ν die Viskosität und F externe Kräfte. Gleichung 2.4 ist die sogenannte Kontinuitätsgleichung, die die Erhaltung von Masse fordert.
Um diese Gleichungen numerisch zu lösen müssen, wie (Harris, 2003, pp. 10-12,50-58) beschreibt, für jeden Zeitschritt folgende Komponenten (aufgelistet in Reihenfolge der Terme in Formel 2.3) berechnet werden: Advektion, Druck, Diffusion (der Viskosität) und externe Kräfte.
Advektion bezeichnet dabei die Bewegung von Fluiden oder den Transport von Substanzen durch diese. Die Bewegung bzw. Strömung von Fluiden wird durch ein Vektorfeld $\mathbf{u}$ beschrieben. Eigenschaften der simulierten Substanzen, die durch dieses Feld transportiert werden, wie z.B. eine Farbe einer Flüssigkeit oder die Dichte von Rauchpartikeln, werden in einem Skalarfeld gespeichert. Gemäß den Vektoren des Geschwindigkeitsfeldes werden mittels der Advektion die Skalarwerte des Eigenschaftsfeldes angepasst, quasi durch das Feld bewegt.

Soll ein dynamisches Vektorfeld simuliert werden, sprich, sich über die Zeit verändernde Strömungsrichtungen und -stärken, kann die Advektion auch auf das eigene Vektorfeld selbst angewendet werden. In diesem Fall spricht man von Self Advection bzw. Selbstadvektion.
Eine in der Computergrafik häufig angewendete Methode zur Berechnung der Advektion, die zwar nicht ganz so präzise, dafür aber schnell und stabil ist, basiert darauf, die Zellen als Partikel zu betrachten und ihre Bewegung über die Zeit zu verfolgen. Dabei wird allerdings für jede Zelle betrachtet, wo das Partikel gemäß des Vektors hergekommen sein muss, sprich, die Position einen Zeitschritt zuvor ermittelt. Dessen vier umliegende Gitterpunkte werden dann interpoliert, um den neuen Wert zu erhalten. Dieses Verfahren nennt sich Semi-Lagrangian Scheme und wird durch Abbildung 2.26 verdeutlicht.

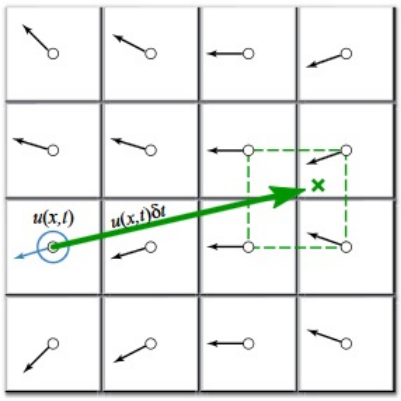


Abbildung 2.26 Berechnung der Advektion für die Zelle mit dem blauen Kreis. Das x markiert die gesuchte Ursprungsposition nach Zurückverfolgung des Vektors, die umliegenden Werte werden interpoliert. (Harris, 2003, p. 57).

Die (viskose) Diffusion hängt von der Viskosität ν des Fluids, landläufig Dickflüssigkeit genannt, ab und beschreibt die Resistenz bzgl. Bewegung und damit die Diffusion (bzw. den Verlust) der Geschwindigkeit. Die nächste Komponente mit Einfluss auf das Fluid ist die Anwendung von äußeren Kräften $\mathbf{F}$. Schließlich muss noch der Druck p beachtet werden, der ebenfalls auf die Fluidpartikel einwirkt und das Geschwindigkeitsfeld beeinflusst, denn Fluide bewegen sich in Richtung der Region mit geringstem Druck. Laut Kontinuitätsgleichung 2.4 muss das Geschwindigkeitsfeld in jedem Zeitschritt divergenzfrei sein, was bedeutet, dass in jede Zelle gleich viel ein- wie herausfließt. Andernfalls würde sich eine Änderung der Dichte ρ ergeben und damit es sich nicht mehr um ein inkompressibles Fluid handeln. Durch Anwendung der Druck-Komponente kann die nach der Advektion und den anderen Einwirkungen ggf. verlorene Divergenzfreiheit wiederhergestellt werden, indem eine Art Diffusionsprozess im Sinne des Dichteausgleichs zwischen Zellen stattfindet. Dies beinhaltet in der Regel die Berechnung der Divergenz, aus der der Druck und schließlich die Gradienten dessen ermittelt werden können.
In (Harris, 2003, pp. 59-60) läuft ein typischer Berechnungsschritt schließlich folgendermaßen ab: *advect*, *diffuse*, *addForces*, *computePressure*, *subtractPressureGradient*. Die letzten beiden Schritte werden in anderer Literatur meist unter dem Begriff Pressure oder Projection zu-

sammengefasst. (Harris, 2003) basiert damit wie viele andere auf der in der Computergrafik sehr bekannten Methode von Jos Stam. Dessen Berechnung, siehe (Stam, 1999) und (Stam, 2003), folgt demselben, oben genannten Schema, wobei die Reihenfolge der einzelnen Teilberechnungen variiert. Zudem war die ursprüngliche Implementierung im Gegensatz zu neueren noch auf Ausführung auf dem Prozessor ausgelegt.

2.6.3 Plattentektonik

Ein entscheidender Faktor für das Erscheinungsbild der Erdoberfläche im großen Maßstab ist ein Prozess namens Plattentektonik. Wie (Viitanen, 2012, pp. 6-20) erläutert, besteht die Erde aus mehreren Schichten. Die oberste, feste Schicht heißt Lithosphäre und besteht aus mehreren großen Platten, die sich auf weicheren, tieferen Erdschichten bewegen und gegenseitig beeinflussen. Der Begriff Plattentektonik beschreibt diese Dynamik. Die Lithosphäre selbst besteht wiederum aus einem oberen Teil, der Kruste, und dem unteren, dem lithosphärischen Mantel. Es gibt zwei Typen von lithosphärischen Platten, die sich durch ihre Krusten unterscheiden. Zum einen die sogenannten ozeanischen Platten, die eine dünnere, schwerere Kruste besitzen und den Meeresboden bilden, und zum anderen die kontinentalen Platten mit dickerer, aber leichterer Kruste, die die Landmassen bilden. Je nach Kombination und Bewegungsrichtungen der Platten kommt es zu verschiedenen Phänomenen.
Driften Platten auseinander, spricht man von divergierenden Platten. In die dabei entstehende Lücke steigt heißes Magma aus tieferen Schichten auf, verfestigt sich und bildet neuen Meeresboden. Eine kontinentale Platte kann zudem aufbrechen und eine sogenannte Riftzone bilden, dabei in zwei Teile gespalten werden, wodurch sich schließlich ein neuer Ozean bildet. Bei hingegen aufeinander zu driftenden, konvergierenden Platten wird eine ozeanische Platte unter eine andere ozeanische oder eine leichtere kontinentale Platte geschoben (Subduktion), wobei daraus eine Tiefseerinne und oft auch vulkanische Aktivität hervorgeht. Treffen zwei kontinentale Platten aufeinander, entstehen stattdessen Faltengebirge. Des Weiteren können sich Platten auch relativ zu der Plattengrenze bewegen. In Abbildung 2.27 sind diese Vorgänge der Plattentektonik schematisch dargestellt.

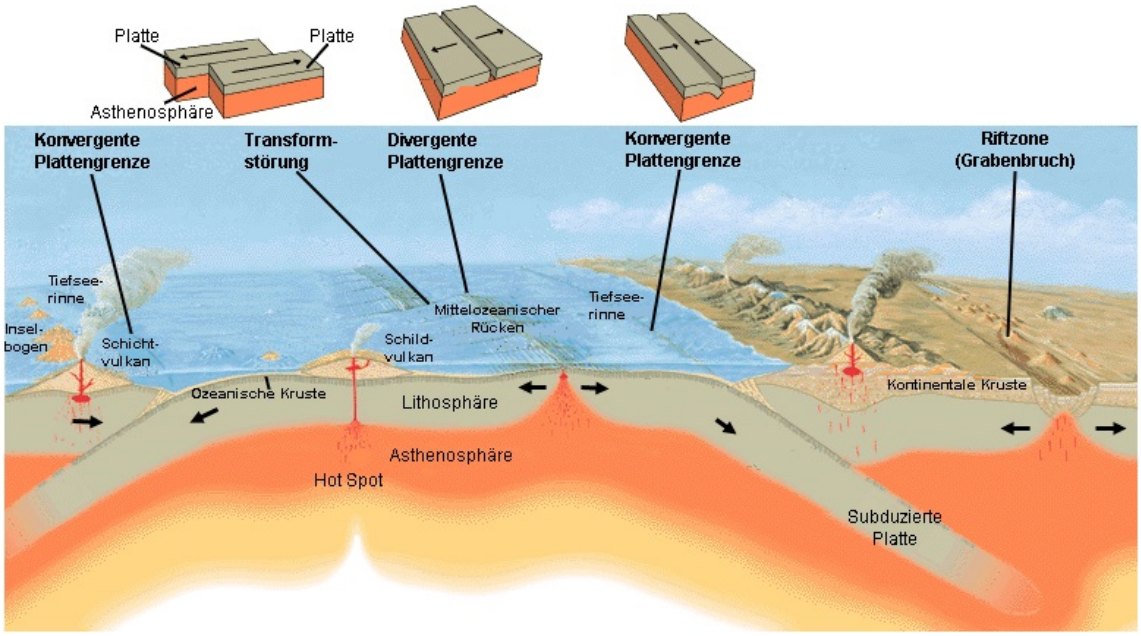


Abbildung 2.27 Prozesse der Plattentektonik (Vigil, 2011).

Dieser Prozess der Plattentektonik kann simuliert werden, um z.B. Vorhersagen über die Bewegung der Platten auf der Erde zu machen oder zur PTG, zwecks Erstellung sehr weitläufiger virtueller Umgebungen. So können nach geologischem Vorbild ganze Kontinente und Landmassen nachgebildet werden, wie es (Viitanen, 2012) und (Cordonnier et al., 2016) beschreiben. Beide Autoren kombinieren die Plattentektonik mit einer Simulation von Erosionsprozessen, welche im nachfolgenden Abschnitt erläutert werden. In (Viitanen, 2012) beginnt die Simulation mit einer Reihe von Platten, die alle relevanten Daten, inklusive einer Heightmap für die Topographie, enthalten und sich in zufällige Richtungen bewegen. Bei Kollisionen werden die entsprechenden Kräfte und Effekte simuliert, so dass sich Platten zusammenschließen, auseinanderbrechen und sich gegenseitig überlagern können, wodurch sich ihre Topographie ändern und Land entstehen oder verschwinden kann. Abbildung 2.28 zeigt ein Beispiel, links die Platten vor der Ausführung, rechts ein Endergebnis.

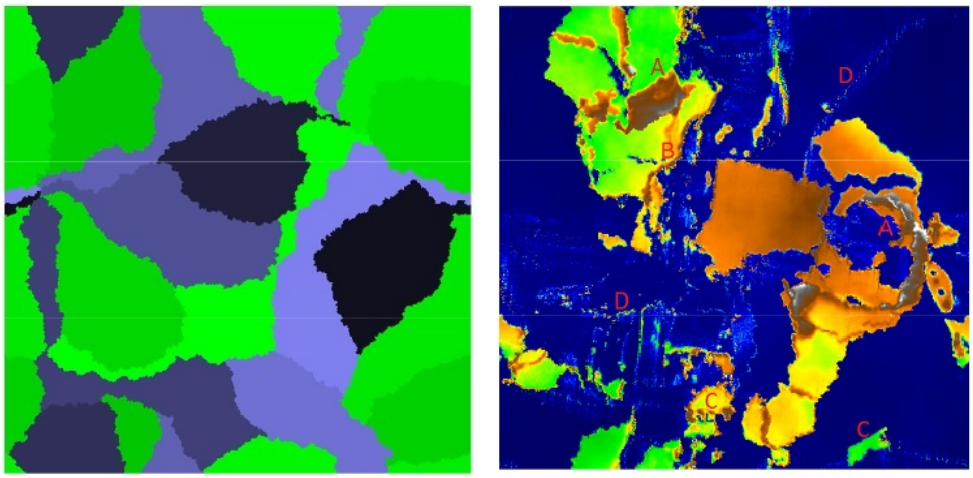


Abbildung 2.28 Links: Einteilung in Platten zum Start. Rechts: Ergebnis der Plattensimulation, die Buchstaben beschreiben verschiedene Landformen (Gebirgskette zwischen Kontinenten, Gebirgskette an Küste, Insel, Inselkette). (Viitanen, 2012, p. 50/53).

2.6.4 Erosion

Im Kontext der physikalisch-basierten PTG ist die Erosion ein häufig simulierter Prozess, wobei damit oft der vorhergehende Prozess der Verwitterung mit einbezogen wird.
Bei Verwitterung handelt es sich in der Geologie, wie in (Encyclopaedia Britannica, 2016) nachzulesen ist, um die Zersetzung und das Aufbrechen von Gestein in kleinere Teile durch äußere Einflüsse. Diese können physischer oder chemischer Natur sein. Zum Beispiel bei der thermalen Verwitterung, engl. Thermal Wheatering oder auch Thermal Erosion, wird durch große Temperaturunterschiede Gestein aufgebrochen, wohingegen bei der Frostverwitterung das gefrierende Wasser in Gesteinsporen durch Ausdehnung für Aufsprengungen sorgt. Ganz ähnlich passiert es bei der Salzverwitterung in trockenen Gegenden durch wachsenden Druck sich ausdehnender Salzkristalle. In der PTG werden meist alle Verwitterungsprozesse unter Thermal Weathering/Erosion zusammengefasst und beschreiben nach (Musgrave et al., 1989, pp. 5-6) das Losbrechen von Gestein an steilen Hängen sowie das darunterliegende Anhäufen des Materials, wodurch sich an der Basis von Klippen, Steilhängen etc. sogenannte Talus Slopes mit stetigem, flachem Winkel bilden.
Der Begriff Erosion bezeichnet hingegen das Abtragen, den Transport und die Ablagerung von (verwitterten) Material durch fließendes Wasser (hydraulische bzw. fluviale Erosion genannt), Wind oder Gletscher. Abbildung 2.29 zeigt beispielhaft einige Arten der Erosion.

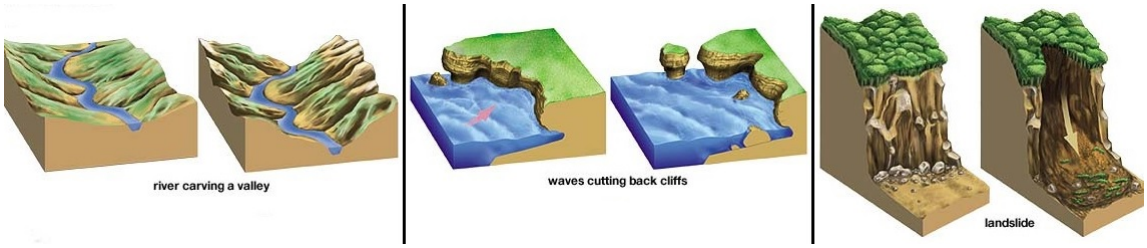


Abbildung 2.29 Einige Arten der Erosion: Links fluviale Erosion durch einen Fluss, in der Mitte Meereserosion und rechts Thermal Weathering/Erosion (Encyclopaedia Britannica, 2006).

Oft wird unterschieden in flächenhafte und linienhafte Erosion, wobei die Grenzen fließend sind. Eine der Arten von Erosion mit dem größten Einfluss auf die Landschaftsbildung ist nach (Encyclopaedia Britannica, 2018b) und (Encyclopaedia Britannica, 2012a) die fluviale Erosion. Sie ist stark verantwortlich für das Bilden von Tälern, Schluchten, Canyons und Flussbetten. Der Vorgang läuft üblicherweise so ab, dass sich Regen- oder Quellwasser durch die Gravitation zu temporären Rinnsalen oder zunehmend größeren Fließgewässern zusammenschließt und bei dem Herabströmen über den Boden vor allem verwittertes Material wie Steine oder Sand- und Gesteinspartikel ablöst, aufnimmt, mit sich transportiert und später wieder an anderer Stelle ablagert. Wie stark die Erosionswirkung ist, hängt von der Bodenbeschaffenheit und dessen Erosionsresistenz ab, der Wassermenge, der Menge des bereits mitgeführten Materials, der Fließgeschwindigkeit, welche von dem Gefälle abhängt, und der Höhe über dem Meeresspiegel.
Ein Beispiel für den Einfluss der Bodenbeschaffenheit ist die Bildung von Wasserfällen, die u.a. dort langsam, aber stetig durch Erosion entstehen, wo in Fließrichtung auf hartes Gestein deutlich weicherer Boden folgt. Dieser wird dann schneller erodiert und es bildet sich eine Stufe, wo das Wasser herabfällt. Die Erosionsprozesse sind auch maßgeblich für die Bildung und den Verlauf von Flüssen verantwortlich. Im sogenannten Oberlauf, nahe der Quelle, die sich i.d.R. in einer bergigen Region befindet, schneidet sich das Wasser durch starke Erosion tief in den Boden ein. Im Mittellauf, wo das Gefälle niedriger ist, kommt es verstärkt zu Seitenerosion und auch Ablagerungen, was breitere, sich windende Flüsse erzeugt. Im Unterlauf ist die Fließgeschwindigkeit am geringsten und es findet vor allem Ablagerung von transportiertem Material statt. Es bilden sich Flussdeltas. Abbildung 2.30 illustriert die drei Abschnitte eines Flusssystems.

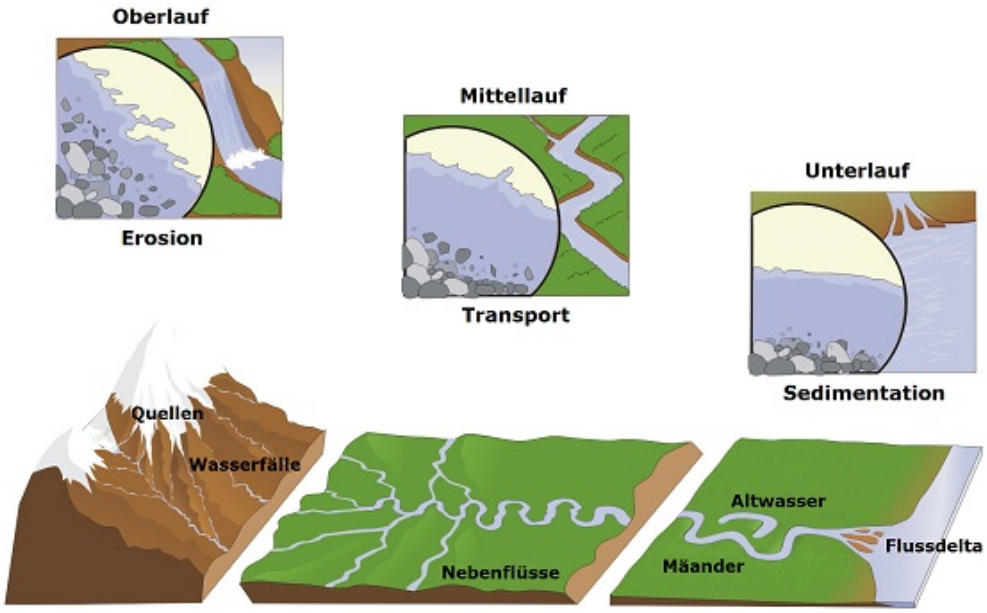


Abbildung 2.30 Die drei Abschnitte eines Flusssystems. Erst die junge Phase, Oberlauf genannt, mit starker (Tiefen-)Erosion aufgrund von u.a. hoher Fließgeschwindigkeit. Danach der Mittellauf, wo vor allem der Transport des Gesteins sowie Seitenerosion stattfindet, und schließlich der flache Unterlauf, in dem überwiegend die Sedimentation überwiegt und damit z.B. Deltas gebildet werden. (Shaker et al., 2016, p. 78).

Zur Simulation von hydraulischer Erosion für die PTG wurden viele verschiedene Algorithmen entwickelt, die teils auf bestehenden Flussnetzwerken beruhen, aber häufiger nachträglich auf bestehende Landschaften angewendet werden. Der allgemeine Ablauf der letzteren Algorithmen lässt sich wie folgt beschreiben: Im ersten Schritt wird Wasser in Form von gleichmäßig oder zufällig verteiltem Regen oder Flussquellen der Landschaft zugeführt. Danach findet der Erosionsprozess statt und anschließend eine mehr oder weniger aufwändige und realistische Simulation des Wassers und dessen Fluss über die Landschaft, wobei Transport und Ablagerung von Sediment nachgebildet werden. Schließlich wird ein Teil des Wassers wieder evaporiert, bevor der Zyklus von neuem beginnt. Je nach Modell ist es auch üblich, erst den Fluss zu berechnen und basierend darauf den Erosions- und Ablagerungsprozess durchzuführen.

(Musgrave et al., 1989) waren die Ersten, die Algorithmen für thermische und hydraulische Erosion vorstellten. Diese dienten anschließend als Basis und Inspiration für weitere Autoren, wie (Olsen, 2004), die sie optimierten oder erweiterten, etwa um den Evaporierungsschritt oder ausgefeiltere Simulation des Wassers. (Neidhold et al., 2005) z.B. nutzen ein Modell auf Basis vereinfachter newtonscher Physik zur Fluidsimulation und (Mei et al., 2007) verwenden die sogenannten Shallow Water Equations, welche von den Navier-Stokes-Gleichungen abgeleitet sind, und ein Virtual-Pipe-Modell zwecks paralleler Ausführung auf der Grafikkarte. Abbildung 2.31 zeigt ein Beispiel. (Jákó, 2011) baut darauf wiederum auf und adaptiert das System auch zur parallelen Berechnung von thermaler Erosion.

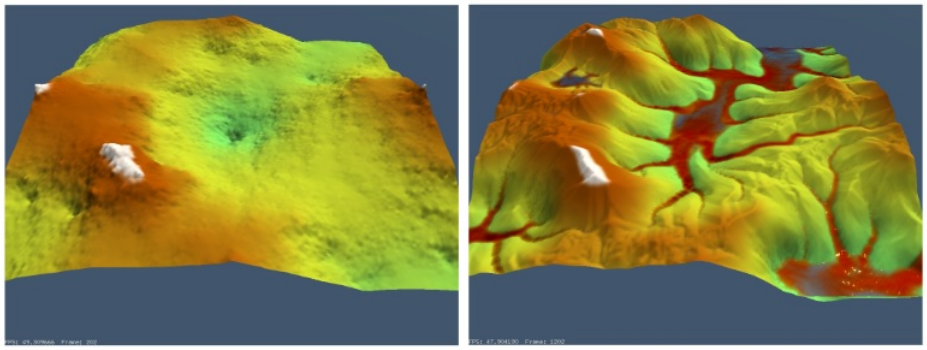


Abbildung 2.31 Links ein initiales Terrain, rechts thermische und hydraulische Erosion nach 1000 Iterationen (Jákó, 2011, 6).

2.7 Prozedurale Asset Distribution

Die prozedurale Verteilung von Assets aller Art, seien es organische Objekte, wie verschiedene Pflanzenarten, oder anorganische Objekte, wie Steine und von Menschen geschaffene Gegenstände, in großen (ggf. auch prozedural generierten) Umgebungen erfährt heutzutage zunehmend Beachtung, denn nicht nur die Erstellung riesiger Terrains, sondern auch das plausible Befüllen dieser mit Assets bedeutet großen Aufwand. Zur Lösung dieser Aufgabe, im Folgenden vorwiegend bezogen auf die Distribution von Pflanzen, lassen sich zwei grundlegende Ansätze identifizieren, (Lane et al., 2002, pp. 1-2) nennen sie Local-to-Global und Global-to-Local.
Bei dem Local-to-Global-Ansatz, den man sich auch abstrakter als Bottom-up-Strategie vorstellen kann, werden einzelne Pflanzen und ihre Wachstumsansprüche simuliert, woraus sich dann mit der Zeit eine entsprechend emergente Verteilung einstellt, siehe beispielhaft in Abbildung 2.32.

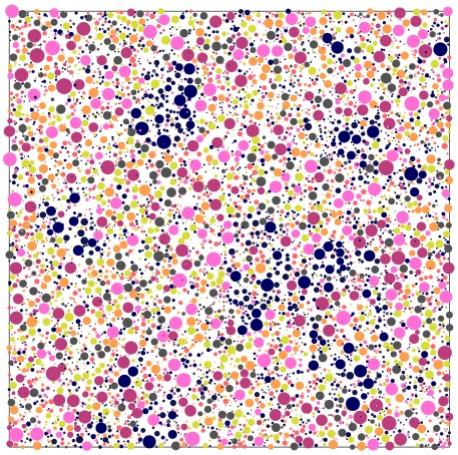


Abbildung 2.32 Simulierte Verteilung von acht Pflanzenarten, blaue Punkte repräsentieren Pflanzen, die feuchte Gebiete bevorzugen (Deussen et al., 1998, p. 6).

Diese Methode führt zu relativ realistischen Ergebnissen und wird daher oft zur Simulation

von Ökosystemen angewendet, ist allerdings auch vergleichsweise rechen- und damit zeitintensiv. (Deussen et al., 1998, pp. 5-6) beschreiben ein solches Modell, bei dem jeder Pflanzenart u.a. eine Wachstumsgeschwindigkeit, Maximalgröße, Bodenfeuchtigkeit und andere simulierte Größen zugeordnet sind. Alte Pflanzen sterben dort mit der Zeit ab, neue werden in der Nähe bestehender gepflanzt und stärkere verdrängen jüngere, wodurch globale Effekte wie die sogenannte Selbstausdünnung nachgebildet werden.
Der Global-to-Local-Ansatz hingegen setzt als Eingabe eine gegebene Verteilung voraus, etwa durch eine Graustufenkarte, die die lokale Dichte der Pflanzen repräsentiert. Nach dieser Verteilung werden dann direkt Positionen für einzelne Individuen generiert. Die Methode ist schneller und bietet mehr Kontrolle für den Nutzer, ist aber tendenziell weniger realistisch.

In der Natur und speziell bei Lebensgemeinschaften von Tieren und Pflanzen lassen sich drei grundlegende Arten von Verteilungen beobachten, wie (Ludwig et al., 1988, pp. 13-14) beschreiben. Zum einen gibt es die eher selten auftretende zufällige Verteilung, bei der scheinbar keine äußeren Umstände Einfluss hatten, des Weiteren die am häufigsten auftretende geclusterte Verteilung, wo sich Individuen in Regionen mit den besten Umweltfaktoren zusammenfinden oder andere räumliche Prozesse eine Rolle spielen und zu Häufungen führen. Schließlich gibt es zum anderen die uniforme Verteilung, bei der die Individuen durch gegenseitiges Abstoßen und Konkurrenz um Nährstoffe gleichmäßig über die Region verteilt auftreten. Abb. 2.33 veranschaulicht die drei Verteilungen.

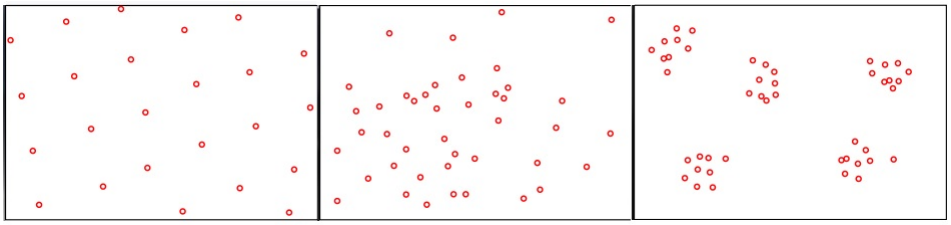


Abbildung 2.33 Von links nach rechts: Zufällig verteilte Samples, geclusterte Samples und uniform verteilte Samples (Yerpo, 2008).

Eine beliebte Technik, um solche uniformen Verteilungen durch Zufallssamples zu generieren, ist das Poisson Disk Sampling, bei dem jedes Sample einen wählbaren Mindestabstand zu den anderen aufrecht erhält, siehe Abb. 2.34.

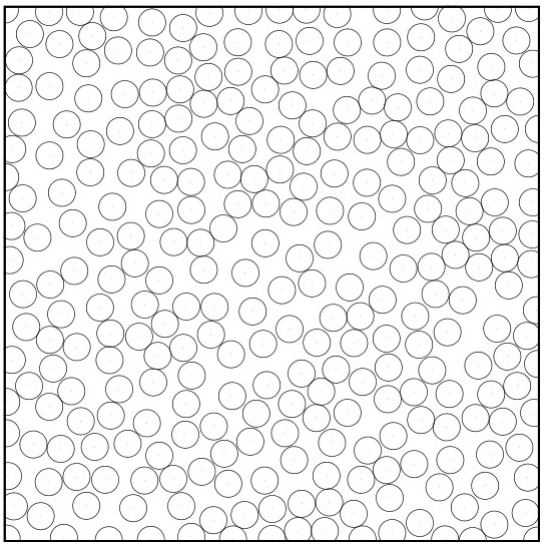


Abbildung 2.34 Poisson Disk Sampling in 2D, die Kreise repräsentieren die Minimalradii um die jeweiligen Samples (Gamito and Maddock, 2009, p. 2).

Ist dabei die verfügbare Fläche maximal ausgenutzt, so dass kein weiteres Sample mehr hinzugefügt werden kann, spricht man von Maximal Poisson Disk Sampling, wie (Gamito and Maddock, 2009, pp. 1-2) beschreiben. Durch z.B. Dichtekarten oder verschiedene Klassen von Samples kann zudem der Mindestradius dynamisch angepasst werden. Die triviale, aber ineffiziente Implementierung von Poisson Disk Sampling nennt sich laut (Jones, 2006, p. 2) Dart Throwing. Dabei werden fortlaufend zufällige Positionen gesampled und falls der Mindestabstand nicht eingehalten wird, das gezogene Sample zurückgewiesen. Da der Algorithmus schlecht mit der Anzahl bestehender Samples und vor allem dem Anteil noch freier Fläche skaliert, wurden diverse effizientere Algorithmen vorgestellt. Sie basieren meist auf räumlichen Datenstrukturen, um zu kodieren, welche Bereiche noch frei und welche bereits belegt sind. So erreichen z.B. (Gamito and Maddock, 2009) und (Jones, 2006) eine Komplexität von O(n log n).

Im folgenden Kapitel wird auf Basis der hier ausführlich diskutierten Grundlagentechniken und des aktuellen Forschungsstandes ein Konzept für das PTG-System erstellt.

Kapitel 3

Konzept des PTG-Systems

Dieses Kapitel widmet sich der Ausarbeitung eines Konzepts für ein optimales System von Algorithmen bzw. einer umfangreichen Pipeline zur prozeduralen Generierung von Multi-Biom-Landschaften. Dazu werden in Abschnitt 3.1 zuerst theoretische Überlegungen über die Ziele, Anforderungen, Einschränkungen und mögliche Methodiken dargelegt und das Für und Wider von den in Kapitel 2 vorgestellten Techniken und Algorithmen erörtert. Darauf folgt in Abschnitt 3.2 die Vorstellung des erarbeiteten Konzepts, was im Laufe der Arbeit teil-implementiert wurde.

3.1 Theoretische Überlegungen

Um, wie zuvor beschrieben, ein Konzept aus vielversprechenden Algorithmen zur PTG zu entwickeln und später möglichst optimale Ergebnisse zu erzielen, müssen zuerst erneut die Zielsetzung besprochen und konkrete Einschränkungen definiert werden, bevor anschließend auf die in Abschnitt 2.1.1 genannten drei übergeordneten Methodiken zur PTG und ihre Vor- und Nachteile eingegangen werden kann. Schließlich werden Kernaufgaben identifiziert und die einzelnen Techniken aus Kapitel 2 und ihre Eignung für diese Arbeit diskutiert.

Zielsetzung und Einschränkungen Wie im Unterkapitel 1.3 bereits erwähnt, ist das Ziel dieser Arbeit das prozedurale Generieren großer Landschaften, d.h. das Terrain, bestehend aus verschiedenen Biomen und passend verteilter Assets, etwa der Vegetation. Auch Gewässer, sprich Meere, Flüsse, Bäche und Seen, sind Teil der Landschaft und spielen somit eine Rolle bei der Terraingenerierung. Die Landschaften sollen dabei möglichst realistisch aussehen. Um diese vage Zielsetzung zu präzisieren, müssen weitere Einschränkungen und Anforderungen definiert werden. Fragen, die sich direkt aus der obigen Beschreibung ergeben, wären z.B., was unter „groß“ verstanden wird und ob eine Echtzeitgenerierung (Online) angepeilt wird oder nicht.

Um einen hohen Grad an Realismus zu erreichen, wird auf eine Echtzeitgenerierung verzichtet

und stattdessen ein Offline-System angestrebt. So steht, wie in Abschnitt 2.1 beschrieben, deutlich mehr Rechenzeit für komplexere Berechnungen zur Verfügung. Daraus ergibt sich weiterhin, dass keine endlos große Welt generiert werden kann. Ein sinnvoll erscheinender Umfang wäre hingegen etwa ein- bzw. einige hundert Quadratkilometer. Dies ist eine Größe, die die der meisten Spiellevel bei weitem übersteigt und etwa der Bereich, den auch die größeren, modernen sogenannten Open-World-Spiele wie z.B. *Grand Theft Auto 5*[1] mit ca. 81 km^2, *The Witcher 3*[2] mit ca. 136 km^2, *Ghost Recon: Wildlands*[3] mit ca. 440 km^2 und *Just Cause* 2 und 3[4] mit ca. 1000 km^2 in first-person bieten. Einige Rennspiele sind noch größer. Nachzulesen ist dies in (Herold, 2017) und (Veltin, 2017). Zudem reicht diese Größe aus, um, in moderat verkleinertem Maßstab, glaubhaft mehrere Biome zu kombinieren.
Ein weiterer Vorteil der Offline-Generierung, der hier genutzt werden soll, ist die (nachträgliche) Modifizierbarkeit. Das System soll zwar einen möglichst hohen Grad der Automatisierung erzielen, aber eine wirklich realistische und vor allem auch interessante und abwechslungsreiche Welt 100% prozedural zu generieren ist (noch) utopisch, daher sollte die Möglichkeit für den Nutzer (Designer) bestehen, nachträgliche manuelle Änderungen und Ergänzungen vorzunehmen, wobei die vorher prozedural generierte Landschaft als Vorlage und Basis dient.

Weitere zu priorisierende Anforderungen sind die Benutzbarkeit und Flexibilität. Um ihnen Rechnung zu tragen wird das System so konzipiert, dass es aus einer Reihe von aufeinanderfolgenden Schritten, bzw. Modulen, besteht und der Nutzer jeweils direkt die entstehenden Auswirkungen verdeutlicht bekommt, Änderungen vornehmen sowie Schritte wiederholen kann. Diese Generierungsphase findet demnach während der Entwicklungs-/Editorphase und damit vor der Ausführung der Endanwendung statt. Mit Bezug auf die Implementierung bietet sich die Nutzung einer Open-Source-3D-(Spiele-)Engine wie der *Unreal Engine 4*[5], *CryEngine V*[6] oder *Unity*[7] an, um von möglichst vielen bestehenden Funktionen bzgl. der Fähigkeit zur Darstellung großer Terrains mittels Techniken wie Level of Detail (LOD), ggf. Level Streaming und Instancing zu profitieren.
Eine weitere wichtige zu treffende Entscheidung ist, ob das Terrain mittels Heightmaps oder Voxeln repräsentiert werden soll, vgl. Abschnitt 2.1.1. Eine Voxel-Darstellung hätte den Vorteil der möglichen Darstellung von Überhängen, Höhlen und ähnlichen Landschaftsformen, würde also mehr generative Freiheit bieten und komplexere Terrains erlauben. Heightmaps hingegen sind die deutlich verbreitetere Technik mit mehr Unterstützung in z.B. den 3D-Engines. Sie sind deutlich performanter zu berechnen, ressourcenschonender und eventuell ausreichend, was die Ergebnisse betrifft.
Was das Performance Target angeht, so müssen die Berechnungen zwar nicht in Echtzeit ausgeführt werden, sie sollten aber in einem zumutbaren Rahmen bleiben, um eine dynamische

[1] `https://www.rockstargames.com/V` (Abgerufen am 11.12.2018)
[2] `https://thewitcher.com/en/witcher3` (Abgerufen am 11.12.2018)
[3] `https://ghost-recon.ubisoft.com/wildlands` (Abgerufen am 11.12.2018)
[4] `https://square-enix-games.com/de_DE/games/just-cause-3` (Abgerufen am 11.12.2018)
[5] `https://www.unrealengine.com` (Abgerufen am 11.12.2018)
[6] `https://www.cryengine.com` (Abgerufen am 11.12.2018)
[7] `https://unity3d.com` (Abgerufen am 11.12.2018)

Entwicklung durch den Nutzer zu gewährleisten, angestrebt wird eine Gesamtberechnungsdauer von mehreren Sekunden oder einigen Minuten.

Zusätzlich zu der Terraingenerierung ist, wie zuvor erwähnt, auch eine prozedurale Asset-Verteilung Teil der Zielsetzung, da sie wichtig zur Veranschaulichung der unterschiedlichen Biome und eng mit ihrer Existenz und Distribution verknüpft ist, wie in dem Unterkapitel 2.6.1 erläutert wurde. Die prozedurale Generierung der Assets selbst ist jedoch kein Bestandteil dieser Arbeit, dies ist eine andere, unabhängige Aufgabenstellung, der sich hier nicht gewidmet wird.

Methodiken und ihre Eigenschaften Die Priorisierung der o.g. Anforderungen an das prozedurale System hat direkten und entscheidenden Einfluss auf die in Frage kommenden Techniken und die Eignung bestimmter Algorithmen. Abbildung 3.1 veranschaulicht die identifizierten Anforderungen Performanz, Realismus, Benutzbarkeit/Komplexität und Flexibilität.

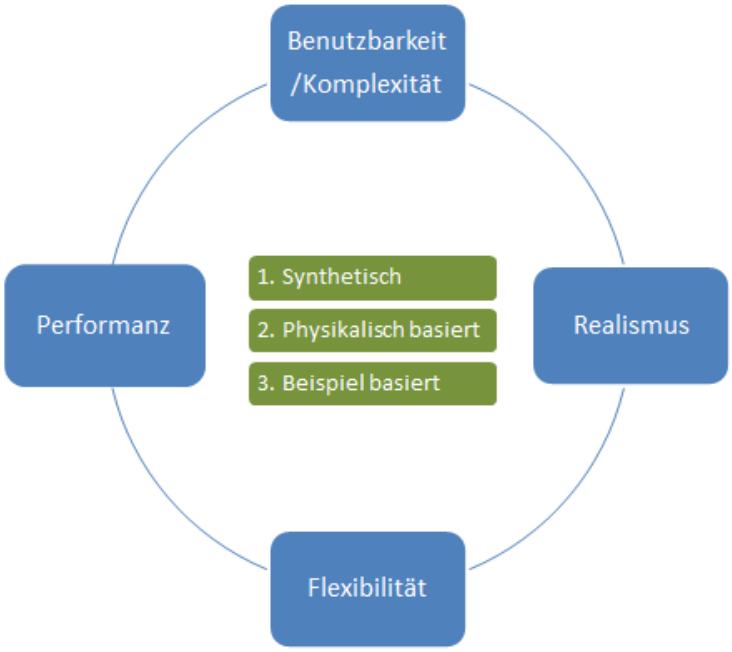


Abbildung 3.1 Diametral zueinander stehende Anforderungen an das PCG-System sowie die drei übergeordneten Methodiken der PTG mit unterschiedlichen Schwerpunkten.

Sie stehen überwiegend diametral zueinander und jede der in Unterkapitel 2.1.1 eingeführten drei großen Berechnungstypen der PTG setzt unterschiedliche Schwerpunkte. Der synthetische Ansatz, etwa durch Einsatz klassischer Noise-Funktionen, bietet eine sehr hohe Performanz und ist zudem flexibel. Jedoch ist die Benutzbarkeit eher gering, speziell wenn versucht wird, realistische Ergebnisse zu erzielen. Beispielhaft lassen sich durch verschiedene Kombinationen von unterschiedlichem Noise mit vergleichsweise geringem Rechenaufwand viele Terrainarten und mit entsprechendem Feintuning auch relativ realistische oder zumin-

dest glaubhafte Resultate erzeugen. Jedoch müssen dazu viele verschiedene, nicht intuitive Parameter eingestellt und aufeinander abgestimmt sowie die entsprechenden einzelnen Noise-Funktionen passend überlagert werden. Dies ist der Benutzbarkeit abträglich und kann schnell sehr komplex für den Nutzer werden. Beschränkt man sich auf ein weniger ausgefeiltes System an Noise-Funktionen mit nur wenigen Schichten und Parametern, können keine komplexen und realistischen Ergebnisse erzielt werden.
Physikalisch-basierte Systeme bieten einen sehr hohen Grad an Realismus zu Lasten des Berechnungsaufwandes. Sie sind zwar in der Regel sehr komplex mit ggf. ebenfalls vielen Parametern, aber es ist denkbar, dass deren Auswirkungen intuitiver nachzuvollziehen sind und so die Benutzbarkeit besser ist. Jedoch müssen hier voraussichtlich auch Abstriche bzgl. der Flexibilität gemacht werden, da intern mehr Bedingungen, feste Formeln und Prozesse abgebildet werden.
Die dritte Variante, die Beispiel-basierte PTG, ist vielleicht die interessanteste und am schwierigsten einzuordnende, basierend auf der Differenz der Methoden und ihrer teils relativen Neuheit, speziell wenn man an die Verwendung Neuronaler Netze denkt. Im Kern lässt sich sagen, dass mit Verwendung von realen Beispielen sehr realistische, wenn nicht sogar die realistischsten, Terrains generiert werden können und das mit einem minimalen Rechenaufwand. Die Flexibilität und vor allem Benutzbarkeit sind jedoch deutlich eingeschränkt, denn man kann nur das nachbilden, was als Beispiel zur Verfügung steht und dies muss manuell zusammengesucht werden. Durch Verwendung von statistischen und generativen Methoden, neuerdings speziell auch Neuronaler Netze, wird die Freiheit, was aus den Beispielen generiert werden kann, erhöht, was der Flexibilität zugutekommt, aber auch schnell den Rechenaufwand in die Höhe schießen lässt. Beispiel-basierte Verfahren, die auf Sketches beruhen (Sketch-basiert), würden zwar mehr Kontrolle und Freiheit bringen, aber das manuelle Zeichnen für die angestrebte Größe des Terrains würde schnell mühselig werden, zudem mangelt es bei diesen Methoden wiederum an Realismus sowie Kontrolle über die Terraindetails.

Kernaufgaben und mögliche Techniken Die Forschungsfrage, die sich stellt, ist, können die verschiedenen Ansätze eventuell kombiniert werden, so dass die jeweiligen Vorteile zur Geltung kommen und in Summe ein besseres Ergebnis erzielt werden kann, bezogen auf die Gesamtheit der Anforderungen? Betrachtet man das Ziel der Arbeit, die Generierung von Multi-Biom-Landschaften, steht fest, dass die jeweiligen Biome durch (stark) unterschiedliche Terrainbeschaffenheit gekennzeichnet sind. Daraus ergibt sich, dass Kernaufgaben das Ermitteln der Biome, bzw. derer Verteilung über das Terrain, sowie das Erzeugen der biomspezifischen Landschaft sind. Leicht zu identifizierende Schwierigkeiten sind dabei der Übergang zwischen den Biomen sowie die Dualität zwischen Biomen und dem gegebenen Terrain. Denn, wie in Unterabschnitt 2.6.1 dargelegt, hängt das vorherrschende Biom indirekt vom Terrain ab, das Terrain jedoch ebenso vom Biom. Denkbar wäre hier ein übergeordneter zweiteiliger, iterativer Algorithmus, der abwechselnd das Terrain bzgl. des Bioms und darauffolgend die Biome anhand des neuen Terrains anpasst, bis sich evtl. eine Art Konvergenz einstellt oder

eine gewisse Anzahl an Iterationen vergangen ist.

Die Verteilung der Assets ist eine weitere Kernaufgabe, die jedoch ausschließlich vom Terrain und dem Biom abhängig ist und so leicht separiert werden und zum Schluss ausgeführt werden kann. Denkbar wäre für diese Aufgabe der in Abschnitt 2.7 erwähnte Global-to-Local-Ansatz, idealerweise mit zur Verfügung stehenden tatsächlichen Distributionsdaten von Pflanzengattungen. Interessant wäre es auch, ein Machine-Learning-Algorithmus oder Neuronales Netz darauf zu trainieren, die Verteilung aus einer gegebenen, sei es eine reale oder vom Nutzer manuell vorgegebene, zu erzeugen. Da allerdings solche realen Daten schwierig zu erhalten sind und es sehr arbeitsintensiv für den Nutzer wäre, die große benötigte Menge an Trainingsdaten zu erzeugen, ist der Local-to-Global-Ansatz voraussichtlich besser geeignet. Hierbei könnte der Nutzer pro Asset einen Satz an plausibel erscheinenden Grunddaten eingeben und durch Simulation von Wachstum und Verdrängung über die Zeit oder alternativ schlichtes wiederholtes Platzieren unter Berücksichtigung der Constraints ergäbe sich eine glaubhafte, emergente Verteilung. Für die Bestimmung von möglichen Positionen, vor allem bei einem solchen Local-to-Global-Ansatz, könnte man auf Techniken aus dem Poisson Disk Sampling zurückgreifen, die sehr verbreitet für so eine Aufgabe sind, im einfachsten Fall mittels Dart Throwing. Man könnte hingegen auch über die Nutzung von L-Systemen nachdenken, was zwar unüblich erscheint, aber in (Lane et al., 2002) angewendet wird und für geclusterte Verteilungen durch die verzweigende Ausbreitung tatsächlich geeignet sein könnte.

Es wäre evtl. sogar denkbar, im selben Sinne L-Systeme bzw. Grammatiken im Allgemeinen für die Aufgabe der Biomverteilung zu nutzen. So könnten sich verschiedene Biome über die Karte ausbreiten und „wachsen“, bis diese vollständig bedeckt ist und bestimmte Constraints dafür sorgen, dass plausible Nachbarschaften entstehen. Die trivialste Lösung hingegen wäre die Verwendung einer bzw. mehrerer Noise-Funktionen, um die Biome über das Terrain zu verteilen, dies erscheint aber nicht allzu vielversprechend im Hinblick auf eine realistische Verteilung. Hier würde es sich anbieten, statt einem synthetischen Ansatz zu folgen besser auf ein physikalisch-basiertes System zurückzugreifen. Eine vollständige, präzise Simulation des Klimas und seiner Bestandteile, inklusive Fluidsimulation, ähnlich der Wettervorhersage oder wissenschaftlicher Vorhersagemodelle, wäre jedoch deutlich zu rechenintensiv und deshalb nicht geeignet für die in dieser Arbeit gewünschte, vergleichsweise kurze Ausführungszeit. Ein manchmal genutzter Ansatz ist, siehe beispielhaft (Turner, 2017), (Voyagergames, 2015) und (Cepero, 2016), schrittweise die einzelnen entscheidenden Komponenten und Prozesse für die Biombildung, sprich Temperatur, Wind und Feuchtigkeit/Niederschlag, abzubilden und aus ihrer Kombination die Biome nach einer Lookup Table zu klassifizieren. Die einzelnen Komponenten könnten dabei entweder für sich genommen und beschränkt auf die Eingabedaten realistisch, physikalisch simuliert oder simpel synthetisch erzeugt werden. Auch ein irgendwie gearteter Kompromiss wäre möglich und vielleicht am geeignetsten. So ein System hätte den entscheidenden Vorteil, zumindest oberflächlich die in der Realität existierenden Prozesse nachzubilden und somit, je nach konkreter Ausprägung und Komplexität, einen guten Kompromiss aus physikalischer Korrektheit, Nachvollziehbarkeit und damit zum Teil

Benutzbarkeit und ggf. Performanz zu erreichen.

Bezüglich der Kernaufgabe der Terraingenerierung scheint allgemein eine Simulation der Plattentektonik als erster Schritt für die Landmassenverteilung, auf dem andere Algorithmen aufbauen können, sinnvoll zu sein. Jedoch ist bei der hier anvisierten Weltgröße dies voraussichtlich unnötig, da die Welt zu klein ist, als dass verschiedene Platten und die sich daraus ergebenden Landformen zu gebrauchen wären. Stattdessen würde sich ein terrainübergreifendes, grobes Noise oder alternativ ein zu verarbeitender Benutzersketch als Basis anbieten, was durch andere Methoden biomspezifisch verfeinert wird und in Kombination somit realistischere Ergebnisse zu erwarten sind, als durch Noise alleine.
Für die Terraindetails könnten zwar, je Biom, wieder ausgesuchte Noise-Funktionen sorgen, aber vielversprechender wären mit Blick auf einige neuere Veröffentlichungen, die sehr beeindruckende Ergebnisse demonstrieren, z.B. (Jákó, 2011) und (Zhou et al., 2007), sicher Erosionsalgorithmen oder eine Beispiel-basierte Methode, die auf realen Daten beruht. Mit diesen Ansätzen würde man sehr natürliche, plausible und vom physikalischen und geologischen Standpunkt aus korrektere Ergebnisse erzielen. Auch speziell im Bereich der Neuronalen Netze gab es in den letzten Jahren große Fortschritte, was generative Methoden betrifft, und es wäre einen Versuch wert zu prüfen, ob man zuverlässig aus DEM-Daten neue, den Realdaten entsprechende Heightmaps generieren könnte, wie es einige Publikationen nahe legen, siehe dazu Abschnitt 2.5.1. So hätte man einen theoretisch unendlichen Satz an möglichen Daten, die als Detail-Layer für das Terrain der Biome dienen könnten. Eine Frage, die sich dabei jedoch stellt, ist, ob die Auflösung der frei zur Verfügung stehenden Daten ausreichend ist. Aber auch für dieses etwaige Problem könnten die Neuronalen Netze die Lösung bieten in Form von Superresolution, wie (Ledig et al., 2016) sie vorstellen. Weiterhin könnte man den sogenannten Stiltransfer als Alternative zu den anderen Textursynthese- und Blending-Verfahren erproben, siehe dazu Abschnitt 2.2 und 2.3.2, um die Detaildaten mit bestehenden zu kombinieren, sofern für diesen Schritt ein Beispiel-basiertes Verfahren genutzt wird. Die Idee, den Stiltransfer auf Heightmaps anzuwenden, wurde von (Snider-Held, 2017) vorgestellt und sie erscheint sehr vielversprechend, speziell, wenn man sich die Ergebnisse des Stiltransfers im Allgemeinen anschaut und sich veranschaulicht, dass das Ziel des Verfahren exakt der hier gegebenen Aufgabenbeschreibung entspricht.
Um die Biomgrenzen und ihr jeweiliges Terrain, unabhängig davon, wie es biomspezifisch generiert wurde, homogen ineinander übergehen zu lassen, bietet sich eine Kombination aus Blending- und dem Graph-Cut-Verfahren an, wie es so oft auch in der Textursynthese für ähnliche Aufgaben angewendet wird (vgl. 2.2). Mit den möglichst cleveren Blending-Verfahren wird versucht, die Biomgrenze zu verstecken. Das Graph-Cut-Verfahren oder andere Segmentierungsmethoden aus der Bildverarbeitung dienen dagegen dazu, den Rand, in einem gewissen lokalen Rahmen, so zu legen bzw. modifizieren, dass er von vornherein möglichst wenig ausgeprägt ist.

Eine letzte Frage wäre nun, wie die Gewässer erzeugt und in das Terrain eingebunden werden. In der Realität besteht zwischen den Gewässern und dem Terrain eine ähnliche wechselseitige

Beziehung wie, weiter oben in diesem Kapitel beschrieben, zwischen Terrain und Biomen. Für eine korrekte Simulation wäre damit wieder eine abwechselnde Berechnung und Anpassung ideal, allerdings auch sehr aufwendig. Eine klassische und hingegen triviale Möglichkeit zur Darstellung von Gewässern wäre ein einfacher Schwellwert, alles unter diesem wäre je nach Anwendung Meer, See oder Fluss. Der offensichtliche Nachteil wäre allerdings, dass sich alle Gewässer auf einer Höhe befinden würden. Um ebenfalls sich aus einem Gebirge herunterschlängelnde Flüsse, höher gelegene Bäche oder Bergseen darstellen zu können, sind andere Techniken notwendig. Eine Möglichkeit wäre, direkt im Zuge der ggf. angewendeten Erosionsalgorithmen und ihrer Wassersimulation die Gewässer zu generieren. Wird keine Erosion verwendet, ließe sich dennoch das Prinzip explizit zur Gewässerberechnung verwenden, um zu ermitteln, wo bei gegebenem Terrain das Wasser entlang fließen und sich sammeln würde. Eine alternative, auch interessante Idee, die nicht physikalisch-basiert, sondern rein synthetisch wäre, ist die Nutzung eines Wegfindungsalgorithmus wie A* mit u.a. der Terrainhöhe als Kostenheuristik und dem nächstgelegenen Gewässer auf Meereshöhe als Ziel. Mehr zu A* in 2.4. Vor allem das Vorgehen von (Cepero, 2013) klingt vielversprechend. Der Kernpunkt dabei ist, dass an all den Stellen ein Fluss oder See vorherrscht, die der Algorithmus im Zuge der Wegfindung von in Gebirgen verteilten Quellen hin zum Meer exploriert hat, statt nur den Stellen, die zum letztendlichen Weg gehören. Dies findet vor allem dort statt, wo das Terrain überwiegend flach ist. So ergeben sich halbwegs natürlich wirkende Gewässer über verschiedene Höhen im Terrain.

3.2 Das PTG-System als Pipelinemodell

Nachdem im vorherigen Abschnitt die Ziele und Beschränkungen dieser Arbeit ausformuliert und über die verschiedenen Lösungsmöglichkeiten diskutiert wurde, wird im Folgenden das Konzept vorgestellt, was sich schlussendlich daraus ergeben hat und kurz die Entscheidungsfindung begründet.

Zum einen wurde sich für die Verwendung von Heightmaps statt Voxeln entschieden, da der Vorteil der leichteren Nutzung und höheren Kompatibilität die fehlenden Repräsentationsmöglichkeiten überwiegt. Des Weiteren kommt die *Unreal Engine 4* zum Einsatz. Für sie spricht, dass sie eine moderne, zunehmend verbreitete Open-Source-3D-(Spiele-)Engine ist, die u.a. zur nicht-kommerziellen Nutzung keine Gebühren verlangt und visuell beeindruckende Ergebnisse liefert. Zudem unterstützt sie viele Techniken wie Level Streaming, LOD-Generierung, Instancing und das Erzeugen bzw. Importieren von großen Landschaften. Somit ist die Engine gut geeignet, um als Umgebung und Basis für das System zur PTG zu dienen.
Um zu versuchen, die jeweiligen Vorteile der synthetischen, physikalisch-basierten und Beispielbasierten PTG zu kombinieren, werden für die verschiedenen Teilaufgaben Methoden aus allen drei Kategorien eingesetzt und auf Synergieeffekte gebaut. Kernpunkte des Konzepts sind zum einen das Verwenden einer oberflächlichen, einfach zu bedienenden Klimasimula-

tion zur Biomverteilung und zum anderen physikalisch- sowie Beispiel-basierte Methoden zur biomspezifischen Terraingenerierung. In Abbildung 3.2 ist das vollständige Konzept, als Pipelinemodell aufgebaut, illustriert.

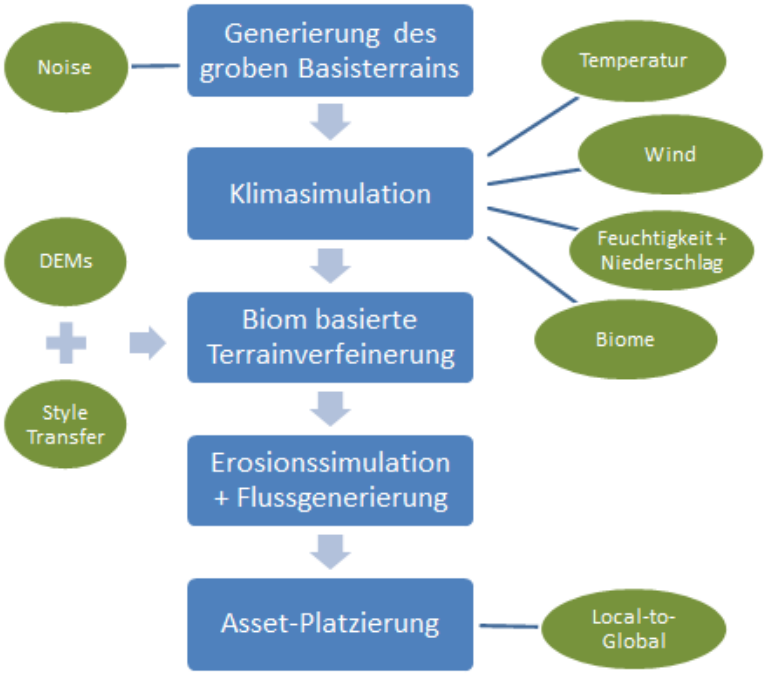


Abbildung 3.2 Das aus den theoretischen Überlegungen hervorgegangene Pipelinemodell des PTG-Systems.

Der erste Schritt der Pipeline ist das Generieren eines groben Basisterrains, um das allgemeine Layout der virtuellen Welt zu beschreiben, wozu auf klassische, einfache Noise-Funktionen zurückgegriffen wird. Auf dieser abstrakten Ebene ist kein langwieriges Experimentieren mit Parametern und verschiedenen Funktionen notwendig und die Flexibilität und Schnelligkeit des synthetischen Noise-Ansatzes kommen zur Geltung. Auf Sketch-basierte Methoden wird verzichtet, da sie zu viel Vor- und Nachbearbeitung und Nutzeraufwand nach sich ziehen. Anschließend wird anhand des generierten Basisterrains und unter Berücksichtigung realer, physikalischer Prozesse die Biomverteilung ermittelt. Dazu wird der Ansatz einer schrittweisen, angedeuteten Klimasimulation aufgegriffen, die jedoch nicht zu präzise und tiefgreifend ist, um Rechenzeit zu sparen, intuitiver zu bedienen zu sein und dem Nutzer mehr Freiheiten zu lassen.
Die Klimasimulation beginnt mit der Erzeugung einer Temperaturkarte nach Nutzervorgaben und unter Berücksichtigung des Basisterrains. Anschließend folgt die Bestimmung von vorwiegenden Winden, welche die Luftmassen bewegen. Danach wird die Feuchtigkeit der Luftmassen und die sich daraus ergebenden Niederschläge berechnet und mithilfe der Winde verteilt. Hierbei werden einfache Bedienbarkeit durch den Nutzer und hohe Performanz der physikalischen Korrektheit und Komplexität der Modelle vorgezogen. Die oberflächliche Anlehnung an die realen Prozesse reicht für glaubhafte Ergebnisse aus. Schließlich wird anhand der so

gewonnen Daten die Biomverteilung erzeugt, wozu eine Lookup Table, bestehend aus der berechneten Temperatur und Niederschlagsmenge, ähnlich des Whittacker-Diagramms, herangezogen wird. Dieses Verfahren der Biomdistribution ist im Kern ein physikalisch-basiertes System, mit dem Vorteil, intuitiv realistische Ergebnisse zu produzieren, wobei je nach Ausprägung der jeweiligen Schritte auch synthetische Einflüsse zu mehr Flexibilität und schnellerer Berechnung führen können als bei einer durch und durch physikalischen Simulation, so wie hier umgesetzt.
Um nun biomspezifisch das Terrain anzupassen und auf ein detailliertes, realistisches Niveau zu heben, wird auf die Nutzung von Beispieldaten in Form von DEMs zurückgegriffen. Im Speziellen werden Beispiel-basierte Verfahren angewendet, die experimentell mittels Neuronaler Netze, konkret dem sogenannten Style Transfer, oder alternativ anderer Textursynthese-Verfahren DEMs über das bestehende Terrain überlagern. Um die in jedem Fall auftretenden Diskrepanzen zwischen den Biomgrenzen zu verstecken, werden gleichzeitig ebenfalls klassische Verfahren aus der Textursynthese und der Bildverarbeitung für diesen Anwendungszweck, etwa Blending-Techniken, eingesetzt. Durch die Anwendung des Beispiel-basierten Ansatzes wird hier ein hoher Grad an Realismus erreicht, der im Fall von Neuronaler Netze nur in der Vorverarbeitung mehr Arbeit und Recheneinsatz erfordert, nicht aber während der konkreten Terraingenerierung. Mittels anderer, generativer Neuronale Netze können zudem vorher beliebig neue realistische Beispiel-Heightmaps aus den Ursprungs-DEMs generiert werden, so dass nicht mehr direkt auf immer manuell zu suchende DEMs zurückgegriffen werden muss.

Auf die im vorherigen Abschnitt hingewiesene Option, Biomverteilung und biomspezifische Terraingenerierung iterativ und abwechselnd auszuführen, wird hier aus Gründen der Komplexität vorerst verzichtet, wobei dies bei tatsächlich festgestelltem Bedarf eine Möglichkeit für weitergehende Forschung wäre.
Nach der Terrainverfeinerung durch DEMs folgt ein Erosionsschritt, der die Niederschlagsverteilung der Klimasimulation als Eingabe nutzen kann, um einerseits das Terrain mit noch feineren Details, die über die Auflösung der DEMs hinausgehen, anzureichern und anderseits den Verlauf des Wassers auf dem Terrain zu verfolgen, um Bäche, Flüsse und Seen zu erzeugen. Die Erosionsalgorithmen, bzw. deren Wassersimulation, zu benutzen ist in dieser Arbeit sinnvoller als der Ansatz mittels Wegfindung, zumal die Ergebnisse realistischer sind.
An dieser Stelle macht sich zudem die Klimasimulation bezahlt und man erkennt die sich ergebenden Synergieeffekte der verschiedenen Algorithmen. So können die berechneten Klimadaten nicht nur zur Biomspezifikation, sondern auch als natürliche Eingabe für andere Algorithmen, wie der Erosion, dienen. Die Erosion wiederum kann inhärent durch die teils komplexe Fluidsimulation zur Erzeugung von Flüssen und Gewässern verwendet werden. Ergänzend wird hier die Schwellwertmethode zur Darstellung eines Basislevels der Gewässer, z.B. eines Meeres, genutzt.
Der letzte Schritt ist schließlich die Verteilung von Assets auf der generierten Landschaft, je nach vorherrschendem Biom. Dazu wird auf ein Local-to-Global-Modell zurückgegriffen,

jedoch vorerst auf eine fortlaufende Simulation verzichtet, um Rechenzeit einzusparen. Stattdessen werden, angelehnt an Poisson Disk Sampling, wiederholt und unter diversen einstellbaren und an reale Faktoren angelehnte Constraints die Asset-Positionen ermittelt. So hat man den Vorteil von viel Flexibilität durch den Nutzer, nicht stark steigenden Berechnungskosten und gleichzeitig glaubhaften, sich emergent entwickelnden Verteilungen.

Im nachfolgenden Kapitel 4 wird die prototypische Implementierung des oben beschriebenen Konzepts ausführlich erläutert.

Kapitel 4

Algorithmen und Implementierung

Nachdem im letzten Kapitel das Konzept und die groben Schritte der Pipeline vorgestellt wurden, wird in diesem nun vertiefend auf diese eingegangen und die dabei verwendeten Algorithmen und Datenstrukturen beschrieben (4.1), konkrete Implementierungsdetails besprochen (4.2) und schließlich die Komplexität analysiert (4.3).

4.1 Abstrakte Algorithmen und Datenstrukturen

Dieser Abschnitt beginnt mit einem High-Level-Überblick über die letztendlich tatsächlich implementierten Schritte der Pipeline. Danach folgt in Unterabschnitt 4.1.1 die grobe Erläuterung der Datenstrukturen und anschließend in 4.1.2 die detailliertere Beschreibung der einzelnen eingesetzten Algorithmen der jeweiligen Pipelineschritte.

Im Zuge dieser Arbeit wurden, gemäß des Konzeptes des vorherigen Kapitels, Experimente mit Neuronalen Netzen zum Stiltransfer durchgeführt, um empirisch ihre Tauglichkeit für den Terrainverfeinerungs-Schritt der Pipeline zu überprüfen. Im Speziellen wurde versucht, so die Heightmaps der DEMs mit dem bestehenden Terrain zu kombinieren. Details zu dem Vorgehen, den verwendeten Implementierungen und den Resultaten werden in Abschnitt 5.1 beschrieben. An dieser Stelle soll jedoch vorweggenommen werden, dass im Laufe der Versuche kein zufriedenstellendes Ergebnis erzielt werden konnte und somit auf ein alternatives, klassischeres und simpleres Verfahren zur Terrainkombination zurückgegriffen werden musste. Dieses wird daher in den in diesem Kapitel folgenden Algorithmen, Diagrammen und Beschreibungen aufgeführt und es wird nicht weiter auf die ursprünglich geplante Nutzung der Neuronalen Netze eingegangen.

In Abbildung 4.1 sind die hier schließlich umgesetzten Schritte in Form eines Unified Modeling Language (UML)-Aktivitätsdiagramms illustriert. Nach der Generierung des Basisterrains folgt die mehrschrittige Klimasimulation, gefolgt von der Terrainfinalisierung und schließ-

lich der Asset-Platzierung. Leider konnten aus Zeitgründen nicht alle Teile aus dem Konzept umgesetzt werden, daher wurden die als weniger wichtig identifizierten Komponenten der Pipeline von einer Implementierung ausgespart. Sie könnten jedoch im Zuge zukünftiger Erweiterungen leicht umgesetzt und integriert werden. Dies betrifft vor allem den Schritt der Erosionssimulation und Gewässergeneration. Ebenfalls mussten manche (Teil-)Algorithmen der Pipelineschritte, ähnlich der Neuronalen Netze, vorläufig durch simplere Alternativen ersetzt werden bzw. konnten nicht ausführlicher experimentell untersucht werden, um den Umfang dieser Arbeit nicht zu überreizen. Auf die möglichen Erweiterungen und Verbesserungspotenzial wird jedoch im jeweiligen Abschnitt hingewiesen.

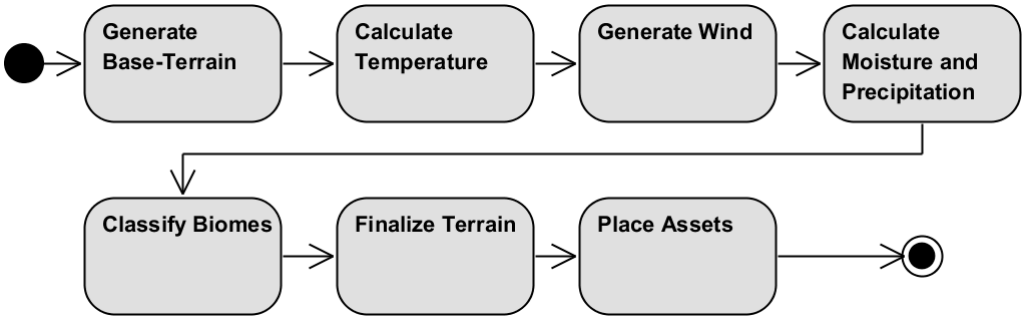


Abbildung 4.1 UML-Aktivitätsdiagramm der implementierten Pipeline und ihre Einzelschritte.

Algorithmus 1 zeigt den Ablauf des implementierten Systems. Wie in dem im vorherigen Kapitel beschriebenen Konzept werden zuerst anhand einiger Parameter eine Heightmap für das grobe Terrain erzeugt und Vorbereitungen für weitere Schritte getroffen. Anschließend beginnt die Klimasimulation mit der Berechnung der von u.a. der Terrainhöhe abhängenden Temperatur, gefolgt von den Winden. Mithilfe von diesen beiden generierten Datensätzen und weiterer Parameter können danach die Feuchtigkeits- und Niederschlagskarten erzeugt werden. Die Klimasimulation endet mit der Biomklassifizierung unter Zuhilfenahme einer Lookup Table. Darauf folgt die Verfeinerung des groben Terrains anhand der Biome mittels DEMs. Abschließend werden die Positionen der Assets ermittelt und die Objekte bzw. Instanzen gespawnt. Das manuelle Importieren des Terrains in die *Unreal Engine* ist nicht im Algorithmus abgebildet und kann vor oder nach dem Generieren der Asset-Positionen erfolgen.

Algorithmus 1 Grober Ablauf des implementierten PTG-Systems

function GenerateBaseMap
generate rough base terrain, water map and do preparations
function CalculateTemperature
calculate height-based temperature
function CalculateWind
calculate prevailing wind
function CalculatePrecipitation
inject and distribute moisture and calculate precipitation
function SelectBiomes
select biomes from lookup table using temperature and precipitation
function FinalizeMap
distort biome borders and combine DEMs with base terrain for final terrain
function DistributeAssets
calculate asset positions based on user data and generate instances

Zu beachten gilt, dass vor Durchführung der einzelnen Schritte der Nutzer die jeweiligen Parameter einstellen kann und muss, um die gewünschten Ergebnisse zu erzielen. Weiterhin ist erwähnenswert, dass jederzeit Schritte wiederholt und vorherige Daten überschrieben werden können. Die DEMs müssen zudem vor dem Verfeinerungsschritt vorliegen, deren Erzeugung oder Erhalt ist nicht Teil des hier implementierten Prototyps.

Nachfolgend werden die verwendeten Datenstrukturen beschrieben.

4.1.1 Datenstruktur

Die Pipeline des in dieser Arbeit implementierten Systems beinhaltet, wie zuvor beschrieben, viele verschiedenen Schritte und Algorithmen. Daher gibt es eine Vielzahl an zu speichernden Daten und jeweils unterschiedliche Anforderungen, wie dies optimal geschehen sollte. Um die Implementierung einfach zu halten, werden hauptsächlich Uniform Grids (deutsch: reguläre Gitter) als Datenstruktur eingesetzt. Da Heightmaps und keine Voxel benutzt werden und auch kein anderer Schritt eine weitere Dimension benötigt, sind die Gitter zweidimensional. Tatsächlich ist diese Datenstruktur vielseitig einsetzbar und die meisten Anforderungen der Algorithmen lassen sich mit ihr ausreichend gut und performant erfüllen. Zu Anfang waren auch Quadtrees und, speziell im Bereich der Klimasimulation, Voronoidiagramme in der engeren Auswahl. Jedoch hätten diese, bei erst zu prüfendem tatsächlichem Effizienzvorteil, zusätzlichen Implementierungsaufwand nach sich gezogen und durch die Verwendung von Heightmaps wären zudem mehrmals Konvertierungen von und zu diesen notwendig gewesen. Vor allem bei Nutzung von Voronoidiagrammen und anderen Graphenstrukturen wäre dies zum Tragen gekommen. Um auch ohne Nutzung von etwa Quadtrees verschiedene Präzisionsstufen über verschiedene Pipelineschritte und nach Nutzerpräferenz zu ermöglichen, werden verschiedene Uniform Grids in unterschiedlichen Größen verwendet, wobei sie immer ein Vielfaches bzw. eher ein Bruchteil des Grundgitters sind, welches die Höhendaten enthält.

So ergibt sich über die Schritte der Pipeline hinweg nach und nach eine Art Stapel von Gittern verschiedener Auflösungen, die für jeden Punkt der Welt alle nötigen Daten, vor allem der Klimasimulation, enthalten. Diese Struktur ist in Abbildung 4.2 dargestellt.

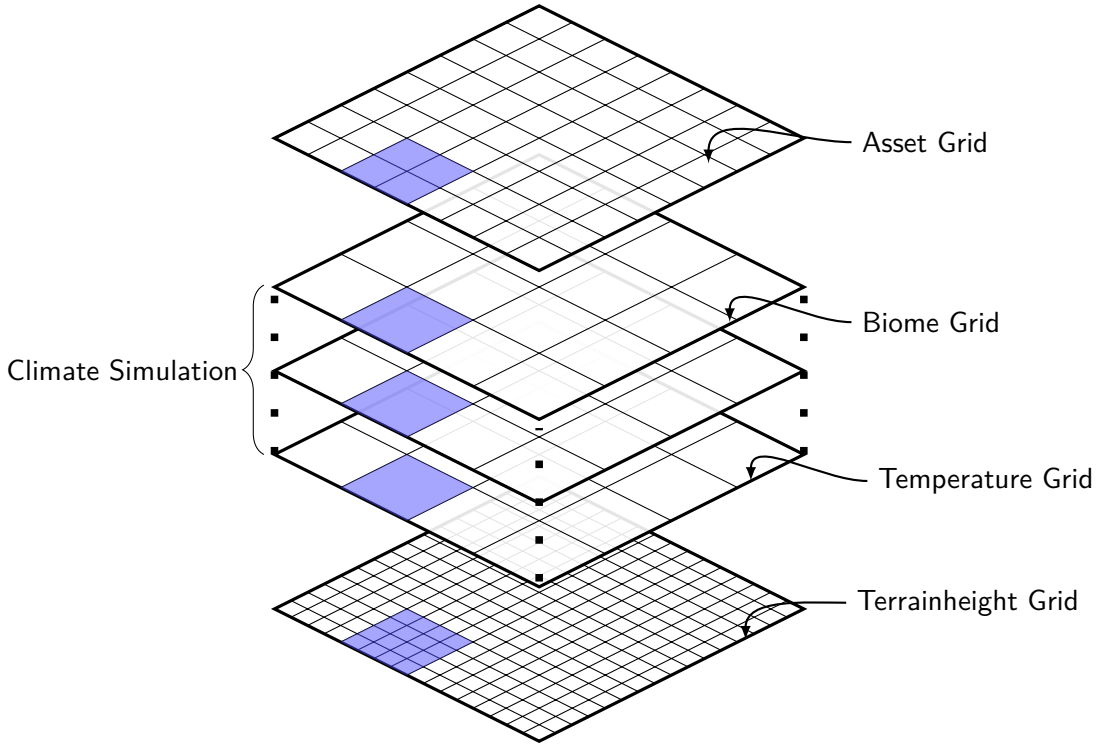


Abbildung 4.2 Grober Aufbau der verwendeten Datenstrukturen: Diverse 2D-Uniform Grids überlagern die einzelnen Daten über das Terrain. Die Auflösung aller Gitter ist ein Bruchteil des Terrainhöhen-Gitters.

Die DEM-Daten liegen als Bilddateien im PNG-Format vor und die letztendlich generierten Heightmaps werden ebenfalls als PNGs abgespeichert. Da zwecks Level Streaming das *Tiled-Landscape*-System der *Unreal Engine* genutzt wird, wird statt einer großen Heightmap-Datei eine Menge kleinerer, sich überlappender erzeugt. Diese werden später wieder importiert. Die generierten Asset-Positionen werden ebenfalls in einem Uniform Grid, als räumliche Beschleunigungsdatenstruktur, abgespeichert, bevor die entsprechenden Instanzen in der Welt generiert werden. Um zu bestimmen, welche Assets in welchen Biomen unter welchen Bedingungen vorkommen sollen, steht dem Nutzer im *Unreal Engine*-Editor eine Liste von Biomen zur Verfügung, in die er selbst zu erstellende und zu parametrisierende Asset-Klassen einfügen kann. Darüber hinaus sind alle Parameter der jeweiligen Algorithmen im Editor einstellbar.

Im Folgenden werden die Pipelineschritte auf einer algorithmischen Ebene vorgestellt.

4.1.2 Algorithmen der Pipelineschritte

Unter Bezugnahme auf das weiter oben abgebildete Diagramm 4.1 werden nun nachfolgend die einzelnen Schritte und ihre Algorithmen sowie Abhängigkeiten genauer diskutiert und jeweils mit einem Sub-Aktivitätsdigramm verdeutlicht. Zu beachten ist, wie aus der zuvor erläuterten

Datenstruktur hervorgeht, dass die Algorithmen der Pipelineschritte auf 2D-Gittern arbeiten und sich demnach einige Aktionen auf diese, bzw. eine Ausführung pro Zelle, beziehen.

Basisterrain Gemäß dem im vorherigen Kapitel 3.2 vorgestellten Pipelinemodell wird als erster Schritt das grobe Terrain, also eine Höhenkarte, mittels Noise-Funktion generiert. Abbildung 4.3 zeigt, wie dieser Vorgang abläuft. Als Eingabe dienen diverse Parameter bzgl. Terrain- und Berechnungsgröße sowie der Noise-Funktion, welche vom Nutzer eingestellt werden können. Auch wird auf eine externe Noise-Bibliothek zurückgegriffen, in der diverse Noise-Funktionen effizient implementiert sind. Der Algorithmus beginnt damit, dass ein 2D-Noise generiert wird, woraus sich anschließend gemäß Parametern die Terrainhöhe ergibt. Darauffolgend wird ebenfalls eine Wasser-Map erstellt, die für spätere Schritte verwendet wird. Schließlich wird auch dynamisch ein Proxy Mesh erzeugt, was das Terrain und dessen Eigenschaften im Verlauf der Pipeline visualisiert, bevor das finale generiert wird.

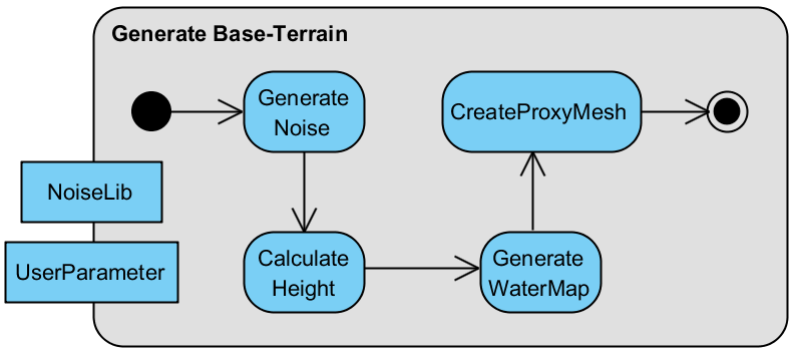


Abbildung 4.3 Sub-Aktivitätsdiagramm des Pipelineschrittes der Basisterrain-Generierung.

Eine mögliche Erweiterung innerhalb des Kontextes der Noise-Generierung könnte es sein, dem Nutzer die Möglichkeit zur Verfügung zu stellen, mehrere und verschiedene Noise-Funktionen, die von der Library (lib) unterstützt werden, zu generieren und kombinieren, um komplexere und weniger homogene Basisterrains zu erzeugen. Zu bedenken ist jedoch, dass das Ziel dieses Pipelineschrittes ist, nur ein vergleichsweise simples Basisterrain zu erzeugen.

Temperatur Auf Basis der zuvor erzeugten Höhendaten beginnt die Klimasimulation mit der Berechnung der Temperatur. Auch hier kommen als zusätzliche Eingabe Nutzerparameter zum Einsatz, die bestimmen, wie bzw. nach welcher räumlichen Funktion sich die Temperatur berechnet. Abbildung 4.4 zeigt, wie für jede Zelle zuerst die Basistemperatur ermittelt und anschließend ein von der Terrainhöhe abhängiger Temperaturabfall berechnet und davon subtrahiert wird. Dadurch wird der Fakt abgebildet, dass die Temperatur mit zunehmender Höhe abnimmt, wie in Kapitel 2.6.1 erläutert wurde.

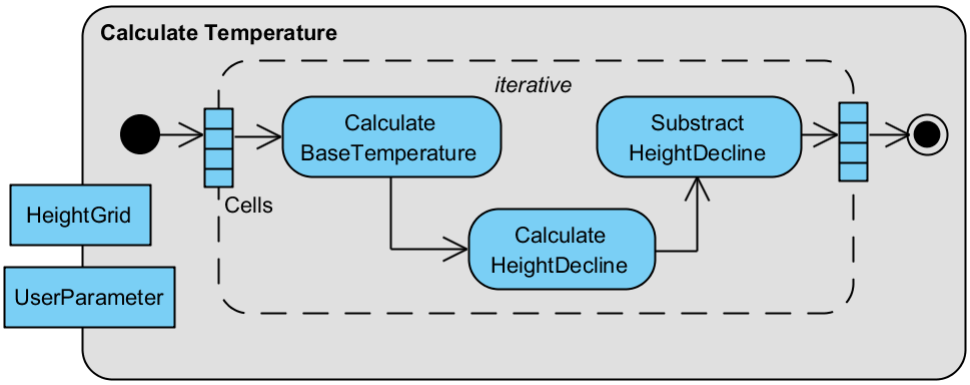


Abbildung 4.4 Sub-Aktivitätsdiagramm des Pipelineschrittes der Temperaturberechnung.

Eine Unterscheidung zwischen Land und Wasser und gesonderten Temperaturen wird nicht gemacht, wäre aber eine denkbare Erweiterung.

Wind Auf die Temperaturberechnung folgt die Erzeugung von Wind, wobei diese unabhängig von den vorherigen Schritten ist. Konkret wird hier nur die Windrichtung berechnet, die Stärke wird vorerst nicht berücksichtigt. Auch wird zwecks Vereinfachung auf die Betrachtung von verschiedenen Höhenschichten und Druck als relevanter Größe verzichtet. In Abbildung 4.5 ist der hier verwendete Algorithmus dargestellt. Er ist inspiriert durch die Methode von Jos Stam, die in Kapitel 2.6.2 vorgestellt wurde, und dem Semi-Lagrangian Scheme, ist aber deutlich vereinfacht. Es werden nur eine Art von Selbstadvektion und externe Kräfte nachgebildet. Zuerst werden alle Windrichtungen des Gitters initialisiert, daraufhin iterativ die Windrichtungen des Gitters berechnet und angepasst, wobei die Windrichtung jeder Zelle mit der gemittelt wird, auf die sie zeigt. Über die Iterationen hinweg wird so die Windrichtung entlang der initialen propagiert und angepasst und es nähert sich insgesamt ein homogener Verlauf an. Zusätzlich werden innerhalb jeder Iteration Windrichtungen ein wenig randomisiert und mit den externen Kräften gemittelt.

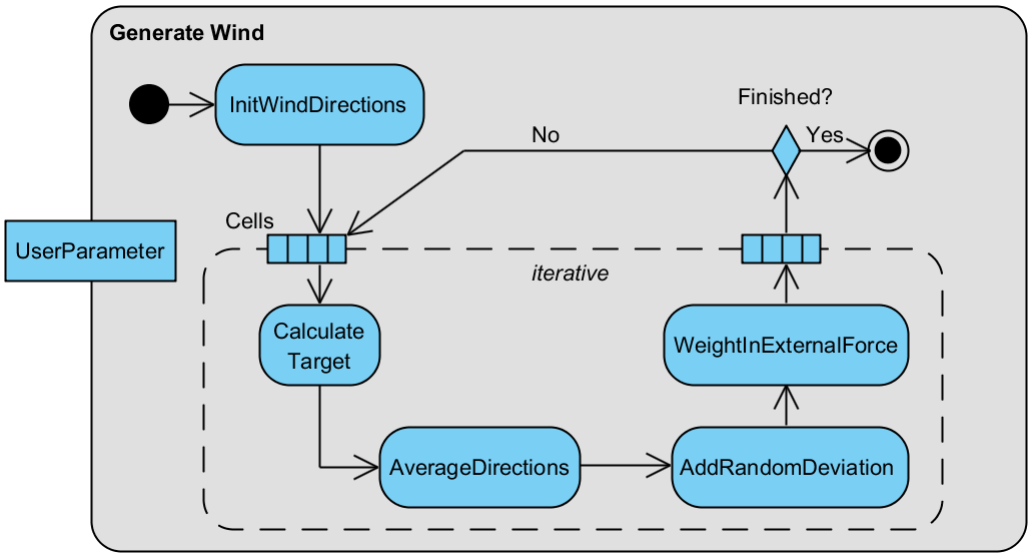


Abbildung 4.5 Sub-Aktivitätsdiagramm des Pipelineschrittes der Windberechnung.

Für eine realistischere Windberechnung könnte man im einfachsten Fall die vernachlässigte Windstärke einführen, das System auf mehrere Höhenschichten erweitern und das Terrain mit einbeziehen oder stattdessen eine vollständige Fluidsimulation einsetzen. Den gesteigerten Realismus und die physikalische Korrektheit würde man sich jedoch mit erhöhter Komplexität und geringerer Performanz erkaufen, was im Endeffekt eine Abwägung nach den Anforderungen ergibt.

Feuchtigkeit und Niederschlag Feuchtigkeit und der daraus entstehende Niederschlag werden in einem gemeinsamen Algorithmus ermittelt, der in Abbildung 4.6 dargestellt ist. Als Eingabe dienen zum einen die zuvor erstellte Wasserkarte, Windrichtungen und Temperaturen und zum anderen einige Parameter des Nutzers. Vor der eigentlichen Berechnung findet ein vorbereitender Schritt statt, in dem die vom Wind abhängigen, statischen Bewegungsmuster bzw. Interaktionen der Zellen des Feuchtigkeitsgitters vorberechnet und gespeichert werden. Daraufhin wird iterativ und für jede Zelle ermittelt, wie viel Feuchtigkeit zu ihr transportiert wird, wie groß der jeweilige Anteil der Feuchtigkeit letztendlich und nach einer Form von Dichteausgleich konkret ist und wie viel Feuchtigkeit sich zusätzlich durch Kondensation über Wasserzellen ergibt. Die Höhe der Kondensation einer Zelle hängt von der Temperatur und Nutzerparametern ab. Anhand der Feuchtigkeitswerte wird innerhalb der Iteration je Zelle zudem ein Basisniederschlag und ein von der Temperatur abhängiger Niederschlag berechnet. Der Basisniederschlag hängt von der absoluten Höhe der vorhandenen Feuchtigkeit und Zufallsparametern ab, der temperaturabhängige Teil hingegen davon, wie hoch das Temperaturdelta bei dem Feuchtigkeitstransport war. Durch den iterativen Prozess wird simuliert, wie Feuchtigkeit dem System hinzugefügt wird, etwa durch die Kondensation über Gewässern, sie sich schrittweise gemäß dem Wind über der Landschaft verbreitet und wie hohe Niederschläge sich dabei jeweils ergeben.

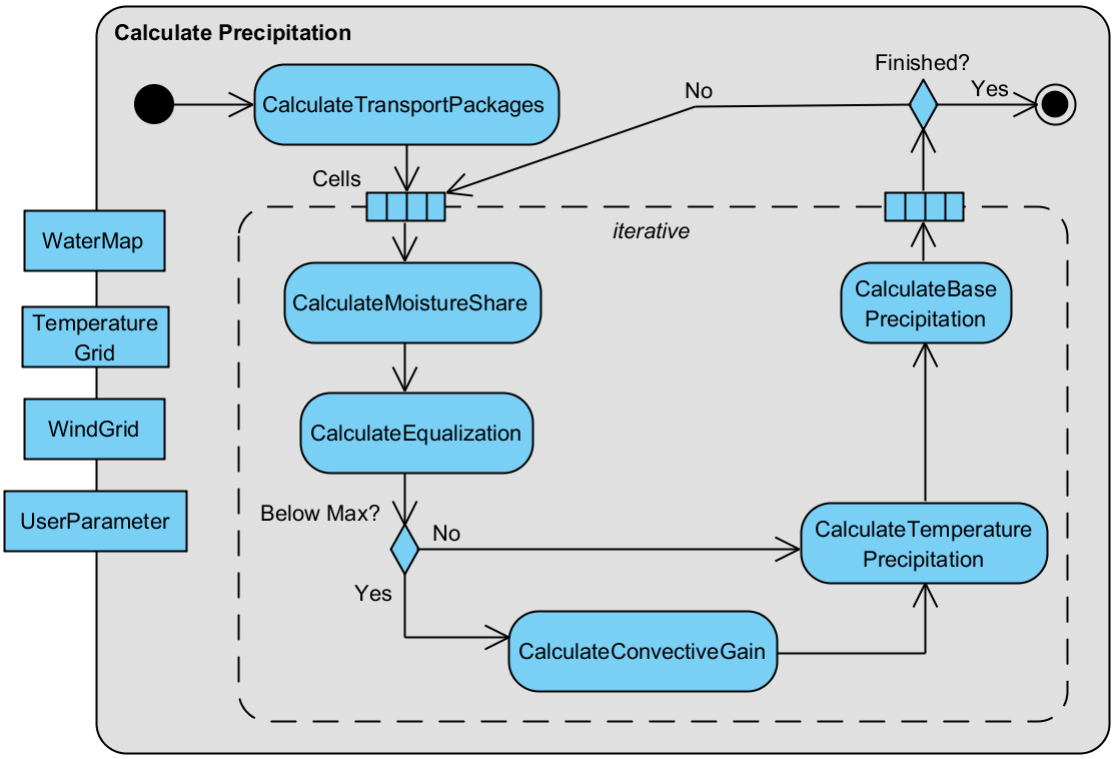


Abbildung 4.6 Sub-Aktivitätsdiagramm des Pipelineschrittes der Feuchtigkeits- und Niederschlagsberechnung.

Mit dem obigen Algorithmus und den eingesetzten Regeln und Formeln für Feuchtigkeit und Niederschlag können die realen Begebenheiten und Phänomene, wie z.B. Regenschatten, plausibel nachgebildet werden.

In Zukunft wäre es denkbar, die Berechnungen auszuweiten und physikalisch korrektere Formeln einzusetzen, um realistischere und präzisere Ergebnisse zu erzielen. Speziell das Berechnen und die Verwendung des Taupunktes wären interessant.

Biomklassifizierung Nachdem alle relevanten Größen zur Klimasimulation vorliegen, kann die Biomklassifizierung erfolgen. Sie basiert auf den zuvor bestimmten Temperatur- und Niederschlagswerten. Abbildung 4.7 zeigt den simplen Ablauf. Es wird eine Lookup Table, welche im Prinzip ein diskretisiertes, modifiziertes und anpassbares Whittacker-Diagramm darstellt, eingelesen und anhand der Wertekombination das passende Biom ausgelesen.

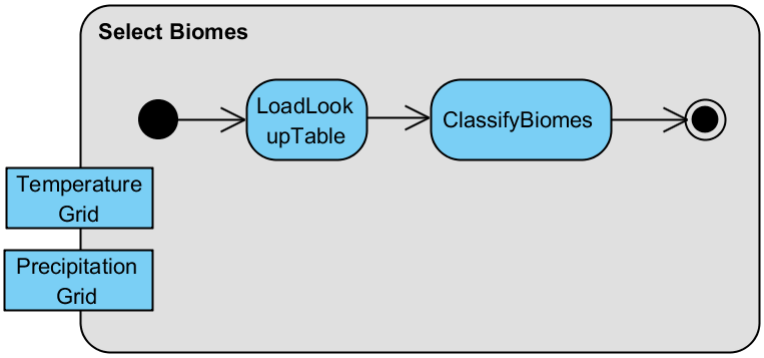


Abbildung 4.7 Sub-Aktivitätsdiagramm des Pipelineschrittes der Biomklassifizierung.

Terrainverfeinerung Auf Basis der bisherigen Terrainhöhe, der Biomverteilung und einigen Nutzerparametern wird das Terrain nun verfeinert, wie in Abbildung 4.8 gezeigt wird. Dazu werden zuerst zuvor in Form von Heightmaps hinterlegte DEMs für jedes Biom eingelesen. Danach wird die Biom-Map auf eine höhere Auflösung skaliert und anschließend, mittels eines generierten Noise, die Biomgrenzen verzerrt. Dies dient dazu, die Grenzen natürlicher erscheinen zu lassen und die harten, schachbrettförmigen Übergänge zu verstecken. Anhand der neuen Biomkarte werden für alle Zellen die jeweiligen Biomanteile, für ein anschließendes Blending der zugehörigen DEMs, ermittelt. Als Letztes wird der aus den DEMs gemittelte Wert mit dem bisherigen Terrainhöhenwert verrechnet.

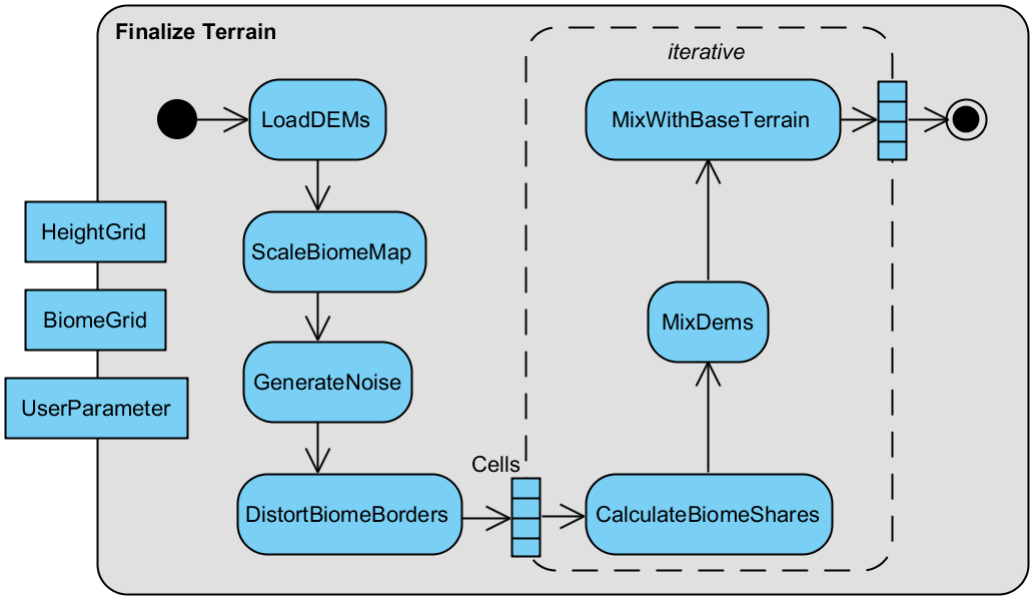


Abbildung 4.8 Sub-Aktivitätsdiagramm des Pipelineschrittes der Terrainfinalisierung.

In diesem Schritt gibt es einiges Verbesserungspotenzial. Zum einen könnte der simple Blending-Part durch ausgefeiltere Algorithmen wie Poisson Blending oder hierarchisches Blending ersetzt werden, siehe 2.3.2, und zum anderen wäre es auch eine vielversprechende Möglichkeit, die Biomgrenzen mittels Graph-Cut-Verfahren (2.3.1) statt dem Noise zu modifizieren.

Asset-Platzierung Der letzte Schritt der Pipeline besteht aus dem Generieren der Asset-Positionen und dem Spawning der zugehörigen Objekte bzw. Instanzen. Wie in Diagramm 4.9 abgebildet, basiert der Algorithmus auf der zuvor erzeugten Biomverteilung. Zunächst müssen vom Nutzer die möglichen Asset-Klassen erstellt und ihre Parameter gewählt werden. Anschließend werden sie den Biomen zugeordnet, in denen sie vorkommen sollen, sowie die angestrebte Häufigkeit bestimmt. Daraufhin werden, sortiert nach Abhängigkeitshierarchie, für alle Vorkommen aller Assets die Positionen innerhalb ihrer jeweiligen Biome gesampled (vereinfacht dargestellt) und diverse Contraints nach den Asset-Parametern abgeprüft. Valide Positionen werden übernommen, die anderen verworfen, bis die gewünschte Anzahl an Positionen erreicht ist. Schließlich werden die Asset-Objekte mittels Instancing an den Positionen gespawnt.

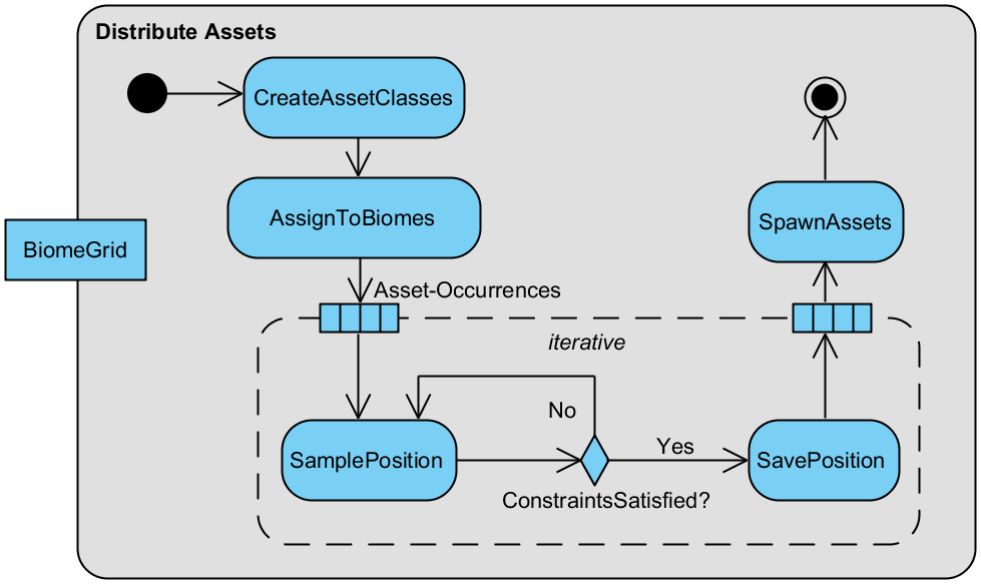


Abbildung 4.9 Sub-Aktivitätsdiagramm des Pipelineschrittes der Asset-Platzierung.

Durch den verfolgten Local-to-Global-Ansatz, der hier durch Dart Throwing und Constraint Checking umgesetzt ist, ergeben sich letztlich emergente, komplexe Verteilungen, die, vom Nutzer nachvollziehbar, durch Wahl der Asset-Parameter modifiziert werden können. Bezüglich der Performanz ließen sich durch Verwendung eines optimierten Sampling-Prozesses und weiterer räumlicher Datenstrukturen Verbesserungen erzielen, speziell bei dichten Populationen. Eine mögliche Alternative zum sequentiellen Samplen wäre eine Simulation über die Zeit, bei dem die einzelnen Asset-Objekte sich gegenseitig beeinflussen, verdrängen und teils auch wieder verschwinden. Dieses Verfahren käme der Realität näher, wäre aber auch langsamer.

Nach den oben erläuterten Algorithmen wird im nächsten Abschnitt auf die Implementierungsdetails eingegangen.

4.2 Implementierungsdetails

In den nachfolgenden Unterabschnitten werden ausführliche Implementierungsdetails zu den verwendeten Datenstrukturen und Pipelineschritten gegeben, die über die bisherigen Erläuterungen hinaus gehen, sowie der Programmaufbau dargestellt.

4.2.1 Programmaufbau

In diesem Abschnitt wird der konkrete Programmaufbau anhand eines UML-Klassendiagramms dargestellt. Es werden alle Klassen und ihre gegenseitigen Beziehungen veranschaulicht. Der Übersicht halber sind jedoch nur die wichtigsten Funktionen innerhalb der einzelnen Klassen aufgeführt.

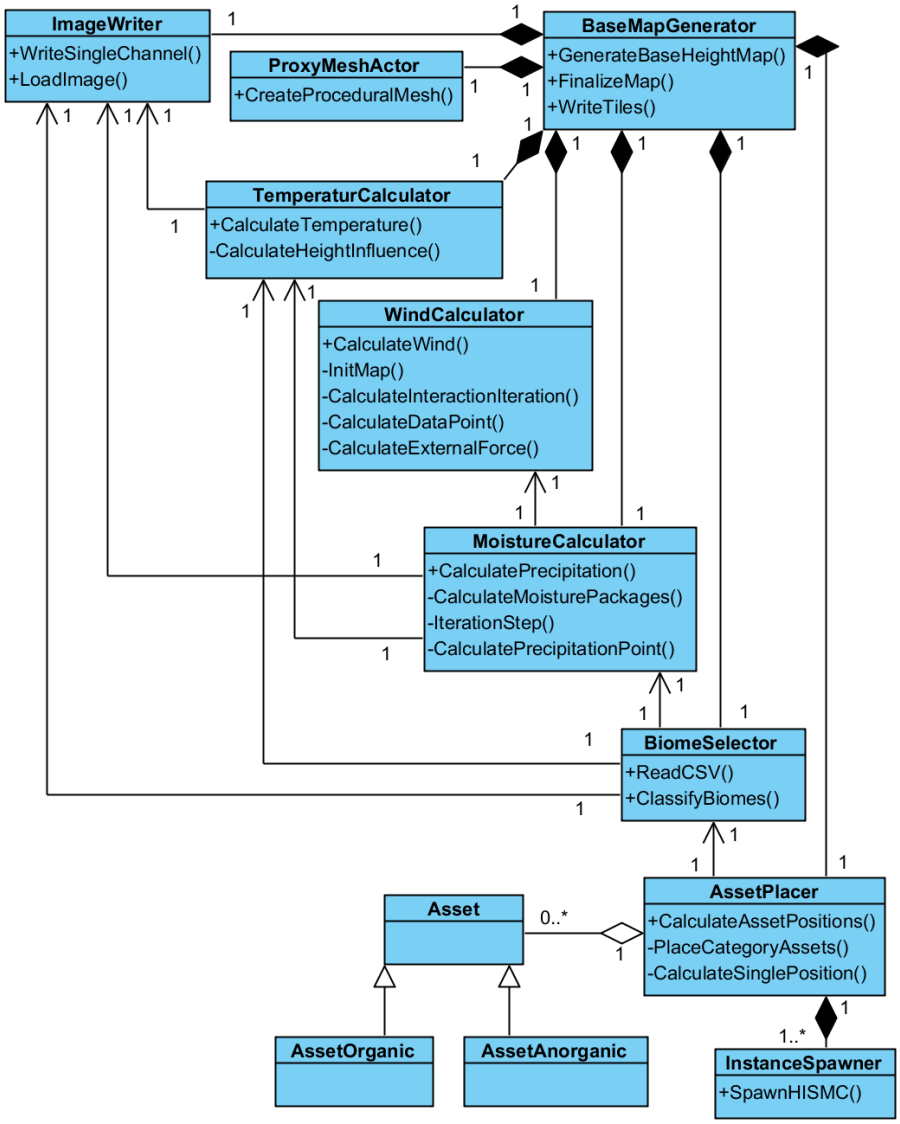


Abbildung 4.10 Der Programmaufbau, veranschaulicht als UML-Klassendiagramm. Der Übersicht halber wurden Rückgabetypen, Funktionsargumente und Klassenvariablen weggelassen.

Es ist ersichtlich, dass die Klasse *BaseMapGenerator* den Ausgangspunkt des Systems bildet und der Großteil der anderen Klassen mit ihr interagieren. Dies sind einerseits Hilfsklassen, wie *ProxyMeshActor* und *ImageWriter*, als auch alle Klassen, die jeweils einen Schritt der Pipeline implementieren, vom *TemperatureCalculator* bis hin zum *AssetPlacer*. Zwischen diesen Klassen gibt es teilweise wiederum Assoziationen, was den sequentiellen Aufbau und die daraus resultierenden Abhängigkeiten widerspiegelt. Darüber hinaus ist die verbreitete Verbindung zum *ImageWriter* und die hierarchische Beziehung der Asset-Klassen erkennbar.

Auf den soeben erläuterten Programmaufbau folgt im Weiteren eine vertiefende Beschreibung der Datenstrukturen.

4.2.2 Details der Datenstrukturen

In diesem Abschnitt werden die Datenstrukturen vertieft und entsprechende Details zur Umsetzung und Anwendung besprochen.

Das primäre Uniform Grid enthält die anfangs vorläufig nur grob generierten Höhendaten und dient als Basisauflösung. Ein weiteres Gitter gleicher Größe enthält explizit die Information, wo Wasser und wo Land ist. Für die Komponenten der Klimasimulation - Temperatur, Wind, Feuchtigkeit, Niederschlag und Biom - wird jeweils ein weiteres Gitter erzeugt, wobei deren Auflösung ggf. nur einen Bruchteil des Basisgitters darstellt. Diese Komponenten greifen bei ihren Berechnungen zudem auf dafür generierte, entsprechend niedriger aufgelöste Versionen von Basis- und Gewässergitter zu. Im Zuge der Wind-, Feuchtigkeits- und Niederschlagsberechnung kommen zudem temporär weitere Gitter und Arrays zum Einsatz, da u.a. eine Art Double-Buffering-Verfahren für die Feuchtigkeitsberechnung angewendet wird und initial sogenannte Feuchtigkeitstransportpakete nach Windrichtung ermittelt werden.
Die Lookup Table für die Biomklassifizierung liegt als Datei im CSV-Format vor und stellt eine diskretisierte und inhaltlich leicht modifizierte Version des Whittacker-Diagramms dar. Abbildung 4.11 stellt den Aufbau vor, die Zeilen repräsentieren die Niederschlagsmenge, die Spalten die Temperatur. Der numerische Eintrag selbst bildet das jeweilige Biom ab, negative Ziffern sind nicht vorgesehen und die erste Zeile sowie Spalte sind für die Beschriftung angedacht.

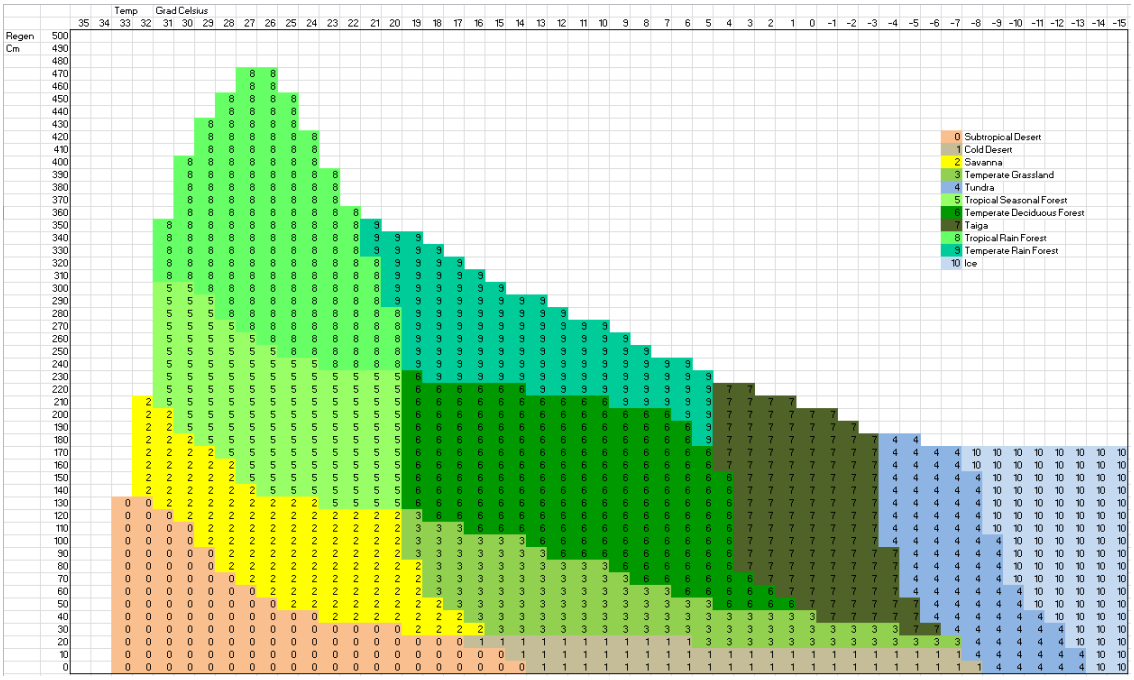


Abbildung 4.11 Lookup Table für die Biomklassifizierung.

Die DEMs liegen, wie zuvor beschrieben, als PNG-Dateien vor und haben das Format 16 Bit Single-Channel. Je Biom wird ein gleichnamiger Unterordner in „/dems/“ erwartet, wo die zugehörigen Dateien hinterlegt sind. Ebenfalls werden gemäß den Anforderungen der *Unreal Engine* die letztlich generierten Heightmap Tiles in diesem Format unter „/Content/Levels/“ abgespeichert und korrekt benannt, damit sie später wieder importiert und die *Tiled Landscapes* durch die Engine daraus erzeugt werden können. Das bedeutet, die sich überlappenden Heightmaps werden bei dem Import zu einem optisch zusammenhängenden Terrain zusammengefügt, bilden aber jeweils ein eigenes Objekt, was durch das Level Streaming-System automatisch ein- und ausgeblendet und so aus dem Speicher genommen werden kann. Für das Einlesen und Schreiben der PNG-Dateien wird die freie Bildverarbeitungs-Programmbibliothek *OpenCV*[1] verwendet.

Für die Platzierung von Assets werden Konstrukte, bestehend aus verschachtelten Arrays und Structs, verwendet, um zum einen dem Nutzer die Möglichkeit zu geben, jedem Biom die zu platzierenden Asset-Klassen sowie ihre Anzahl zuzuordnen und andererseits die letztendlichen Positionsdaten in Abhängigkeit des Asset-Typs abzuspeichern. Diese werden wiederum in einem Uniform Grid, als räumlicher Beschleunigungsdatenstruktur, verwaltet. Jeder Asset-Typ bildet eine Klasse mit vom Nutzer einzustellenden Parametern. Alle Parameter der jeweiligen Pipelineschritte stehen dem Nutzer während des Programmablaufs als frei veränderliche Klassenvariablen im Editor zur Verfügung.

Nachdem zuvor die verwendeten Datenstrukturen hinreichend diskutiert wurden, folgt nun abschnittsweise eine algorithmische Erläuterung der Einzelschritte der umgesetzten Pipeline.

[1] `https://opencv.org/` (Abgerufen am 11.12.2018)

4.2.3 Erzeugen des Basisterrains

Der initiale Schritt der Pipeline ist das Generieren des Basisterrains. In Pseudocode 2 ist der Ablauf dargestellt. Zuallererst werden vom Nutzer einige Parameter eingestellt, betreffend vor allem die Noise-Funktion (*NoiseF*, *NoiseOctaves*), die finalen Terraindimensionen (*WorldLength*, *WorldHeightMin*, *WorldHeightMax*) und verschiedene Berechnungsauflösungen (*CalculationSize*, *BiomeDivisor*, *HeightMapProxyDivisor*). Gemäß den Nutzerparametern für Berechnungsauflösung, Frequenz und Anzahl der Oktaven wird mittels der verwendeten Programmbibliothek *FastNoiseSIMD*[2] von Jordan Peck ein 2D-Fractal-Simplex Noise berechnet, welches als grobe Terrrainhöhe dient. Die erzeugten Werte liegen im Bereich -1 bis 1 und werden zur leichteren Handhabung direkt nach 0 bis 2 verschoben.

Algorithmus 2 Ablauf der Generierung des Basisterrains

```
function GenerateBaseHeightMap
    user adjusts parameters
    generate 2D-simplex-fractal noise using library
    for all heightGridCells do
        height = noise value
        shift height value above 0
    for all waterGridCells do
        calculate real height value h
        if h <= OceanHeightBias then
            mark as water-cell
    generate low precision maps for following steps
    divide heightmap in tiles
    write tiles into files
    generate other classes and initialize them
    generate proxy mesh
```

Anschließend wird aufgrund der Höhenwerte und der eingegebenen Nutzerparameter eine Wasser-Map erstellt, die zwischen Gewässer und Land unterscheidet. Alle Höhenwerte, die unter oder auf dem vom Nutzer vorgegebenem Meeresniveau *OceanHeightBias* liegen, zählen als Gewässer. Um aus den internen Höhendaten im Intervall (0,2) die tatsächlichen, hier h genannt, zu berechnen, wird ein Skalierungsfaktor aus Minimal- und Maximalhöhe ermittelt, mit ihm multipliziert und schließlich die Minimalhöhe als Offset aufaddiert. Daraufhin werden niedriger aufgelöste Height- und Watermaps durch Downsampling mittels Box-Filter berechnet und gespeichert, die in den folgenden Pipelineschritten genutzt werden. An dieser Stelle kann die Karte des Terrains zudem schon auf Nutzerwunsch in sich überlappende Tiles aufgeteilt und in entsprechende Heightmap-Dateien geschrieben werden. Diese können dann von der *Unreal Engine* am Ende importiert und das 3D-Terrain erzeugt werden. Zwar wird das Terrain später in der Regel noch verfeinert, aber diese Dateien können im Zweifel als Fallback dienen. Die Tilegröße kann vom Nutzer vorgegeben werden, wobei, um nicht exakt

[2] `https://github.com/Auburns/FastNoiseSIMD` (Abgerufen am 11.12.2018)

passende Größen zu handhaben, ein Offset verwendet wird und so ggf. der äußere Bereich der Map nicht verwendet und exportiert wird. Schließlich werden alle weiteren Klassen erstellt und initialisiert sowie ein Proxy Mesh samt Overlay-Texturen zur dynamischen Visualisierung des späteren Terrains generiert. Da die Berechnung des Proxy Meshes sehr rechenintensiv ist, steht hier ein weiterer Parameter (Teiler) zur Verfügung, um die Anzahl der verwendeten Vertices in Abhängigkeit der Basispräzision zu bestimmen.

4.2.4 Berechnen der Temperatur

Für die Temperaturberechnung stehen dem Nutzer zwei Modi zur Verfügung. Zum einen eine bidirektionale, lineare Interpolation von vier vom Nutzer vorgegebenen Temperaturwerten an den Ecken des Terrains und zum anderen eine vertikal ausgerichtete Sinusfunktion, die mit einem Temperaturfaktor sowie einem Temperaturoffset parametrisiert wird. Der letztgenannte Modus lehnt sich an den Temperaturverlauf auf der Erde an. Damit hat der Nutzer ausreichend Möglichkeiten, verschiedene Temperaturverläufe zu generieren, ohne eine Vielzahl von Parametern optimieren zu müssen. In Pseudocode 3 ist der Ablauf dargestellt.

Algorithmus 3 Ablauf der Temperaturberechnung

```
function CalculateTemperature
    user chooses mode of calculation (sineMode)
    user adjusts parameters
    if sineMode then
        for all temperatureGridCells do
            temperature = offset + sine function
    else
        for all temperatureGridCells do
            temperature = lerp(cornerValues)
    for all temperatureGridCells do
        calculate height-based temperature-decline td
        temperature -= td
```

Zuerst wählt der Nutzer den Modus *sineMode*, daraufhin stellt er die jeweiligen Parameter ein und je nach Modus findet anschließend zellenweise die Temperaturberechnung statt. Unabhängig vom Modus wird nach der Temperaturberechnung wieder zellenweise ein von der Terrainhöhe abhängiger Wert *td* berechnet und subtrahiert, um, wie in 2.6.1 beschrieben, die in der Realität mit zunehmender Höhe sinkende Temperatur abzubilden. Der Faktor, wie viel Grad die Temperatur je 100 m Höhe abnimmt, lässt sich vom Nutzer direkt über einen Parameter einstellen. Zwecks Auslesen der Höhendaten wird auf das niedrig aufgelöste Gitter des *BaseMapGenerators* zugegriffen, was eine 1-zu-1-Relation der jeweiligen Zellen zulässt. Für die nachträgliche Berechnung des Temperaturabfalls in einer separaten Schleife wurde sich aufgrund besserer Visualisierungs- und Debuggingmöglichkeiten entschieden, obwohl dies technisch weniger effizient ist, wobei dieser Nachteil hier nicht ins Gewicht fällt.

4.2.5 Generieren des Windes

Der Pseudocode 4 zeigt den Ablauf der Windrichtungsberechnung. Nachdem der Nutzer alle Parameter, vor allem die vorherrschende Windrichtung als Vektor an den vier Ecken des Terrains (wc), eingestellt hat, werden alle Zellen des Gitters initialisiert. Dazu wird jeweils ermittelt, in welchem der vier Quadranten sich die Zelle befindet und welche Ecke mit ihrem Wind wc demnach die nächstgelegene ist. Die Distanz zur Ecke in Form der Manhattan-Distanz, sowie deren Windrichtung, wird für die spätere Berechnung der externen Kräfte abgespeichert und die Windrichtung für die aktuelle Zelle übernommen.

Algorithmus 4 Ablauf der Windberechnung

```
function CalculateWind
    user adjusts parameters
    for all windGridCells do
        determine and store nearest corner wind wc
        wind vector = wc
        calculate and store distance to wc
    for all iterations do
        create temporary 2D grid wt for wind
        for all windGridCells_wt do
            calculate current angle a in degree
            determine targetCell with a
            if targetCell is valid then
                get wind direction of targetCell
                average own and targetCells direction
                wind vector = average
            calculate and add random deviation
            calculate force weight based on distance to wc
            set direction as weighted average between wcs and current one
        replace main wind grid with wt
```

Anschließend beginnt die eigentliche, iterative Windberechnung, wobei die Anzahl der Iterationen standardmäßig der Anzahl der Zellen pro Dimension entspricht. In jeder Iteration wird vorerst in ein neues Gitter wt geschrieben, um die alten Daten nicht zu überschreiben, solange sie noch zur Berechnung der neuen Iteration benötigt werden. Erst danach ersetzt der neue den alten Datensatz.

In jeder Iteration wird pro Zelle zuerst anhand der bisherigen Windrichtung in Vektorform der Winkel in Grad a und anschließend daraus die angrenzende Zielzelle, in dessen Richtung der Wind der aktuellen Zelle bläst, ermittelt. Dabei wird die Achternachbarschaft verwendet, d.h., auch diagonale Nachbarzellen werden betrachtet. Die neue Windrichtung der aktuellen Zelle ergibt sich nun als Durchschnitt aus der alten Richtung und der Windrichtung der Zielzelle. Ist, bzw. wäre, die Zielzelle außerhalb des Terrains, bleibt die Windrichtung unverändert. Danach wird die Windrichtung mit einer zufälligen Wahrscheinlichkeit um eine zufällige Anzahl an Grad rotiert, um kleinere, lokale Abweichungen des Windes nachzuahmen. Bei-

de Wahrscheinlichkeiten stehen als Parameter zur Verfügung. Abschließend werden externe Kräfte berücksichtigt, indem die finale Windrichtung der Zelle durch Interpolation zwischen iterativ berechneter und der Basiswindrichtung der Ecke bestimmt wird. Das Gewicht der Interpolation wird mit einer Funktion berechnet, die auf der abgespeicherten Distanz der nächstgelegenen Ecke beruht. Je näher die Zelle der Ecke ist, desto größer ist der Einfluss der an der Ecke vorherrschenden Windrichtung, und je weiter die Zelle weg ist, desto stärker wird die berechnete Richtung gewichtet.

4.2.6 Berechnen der Feuchtigkeit und des Niederschlages

Die Feuchtigkeits- und Niederschlagsberechnung hängt von der gegebenen Windrichtung, den Temperaturen und dem Terrain ab. Wie Pseudocode 5 zeigt, stellt wie üblich der Nutzer zuerst alle nötigen Parameter ein, woraufhin vor der eigentlichen Berechnung zuerst, mittels der statischen Windrichtung, die Interaktionen zwischen benachbarten Zellen des Feuchtigkeitsgitters ermittelt und abgespeichert werden. Dazu werden *MoisturePackage* genannte Structs verwendet, die die nötigen Informationen enthalten und den entsprechenden Empfängern zugeordnet werden.
Konkret ist der Ablauf, dass für jede Zelle erst der Winkel der Windrichtung bestimmt wird. Bläst der Wind auf eine valide andere Zelle, wird daraufhin ein *MoisturePackage mp* angelegt. Die eigene Zelle wird dem Paket als Absender zugeordnet, die Zielzelle als Primärempfänger, die beiden benachbarten Zellen aus Sicht des Senders ermittelt und als Sekundärempfänger hinterlegt sowie deren Gewichtung berechnet und gespeichert. Die Gewichtung wird anhand des aus dem Windvektor berechneten Winkels bestimmt, je höher die Abweichung von der Mittelposition der Zielzelle, desto stärker die Gewichtung des entsprechenden Nachbars auf dieser Seite. Die ID des generierten Pakets wird bei jedem Empfänger in eine entsprechende Liste eingehängt.

Algorithmus 5 Ablauf der Niederschlagsberechnung

```
function CalculatePrecipitation
    user adjusts parameters
    for all moistureGridCells do
        calculate angle from wind vector
        create moisture package mp and set own id as sender
        calculate and set mps receiver ids and shares using angle
        store mp in receiver cells
    for all iterations do
        for all moistureGridCells do
            if linear then
                calculate potential moisture gain pmg using linear formula
            else
                calculate potential moisture gain pmg using exponential formula
            if waterCell then
                add pmg to moisture gain mg
            else if is terrain border and borderMoisture set then
                add pmg multiplied by factor to moisture gain mg
            for all receivingMoisturePackages do
                calculate average of previous moisture over all receiving cells
                calculate moisture share using expansion parameters and average
                calculate moisture m from sender using share
                add m to second moisture grids cell and array ma
            if value of second moisture grids cell below max then
                add as much of mg as possible
            if ma not empty then
                for all receivingMoisturePackages and ma do
                    calculate temperature difference from sender
                    if linear then
                        calculate temperature precipitation tp using linear formula
                    else
                        calculate temperature precipitation tp using exponential formula
                    add tp to precipitation grids cell
                    subtract tp from second moisture grids cell
            calculate random amount of precipitation p
            add p to precipitation grids cell
            subtract p from second moisture grids cell
        swap moisture grids and clear second one
```

Nach diesem vorbereitenden Schritt beginnt die iterative Berechnung von Feuchtigkeit und den daraus folgenden Niederschlägen. Hierzu wird ein Double-Buffering-Verfahren angewendet, was bedeutet, dass am Ende jeder Iteration zwischen zwei Feuchtigkeitsgittern getauscht und das „neue“ geleert wird. Das erste Gitter enthält die bisherige Feuchtigkeit, also die, welche in vergangenen Iterationen berechnet wurde, und das zweite dient der Berechnung des daraus resultierenden neuen Wertes in der aktuellen Iteration. So wird nach und nach die Feuchtigkeit gemäß Windrichtung über die Map propagiert.

Innerhalb einer Iteration wird für jede Zelle damit begonnen, die durch Kondensation potentiell hinzukommende Feuchtigkeit *pmg* zu ermitteln und ggf. in *mg* zu speichern. Je nach Nutzerparameter kann eine von zwei Formeln zur Berechnung gewählt werden, die erste ist linear, die zweite exponentiell.
Für alle Zellen, die laut der Klasse *BaseMapGenerator* ein Gewässer beinhalten, wird im linearen Fall eine gewichtete Summe aus einem Basiswert sowie einem Temperaturfaktor gebildet, der mit der dortigen Temperatur multipliziert wird. Im exponentiellen Fall bildet die Temperatur den Exponenten des Temperaturfaktors, der mit dem Basiswert multipliziert wird. Höhere Temperaturen führen in jedem Fall zu vermehrter Kondensation. Zellen, die zwar kein Gewässer sind, aber den Kartenrand darstellen, können, differenziert nach Terrainseite und vom Nutzer auswählbar, ebenfalls für Feuchtigkeitszufluss nach obigem Schema sorgen. In diesem Fall jedoch ergänzt um einen weiteren einstellbaren Faktor, der die Stärke bestimmt. Somit wird von außerhalb der dargestellten und simulierten Welt durch die vorherrschenden Winde zuströmende Feuchtigkeit nachgeahmt.
Anschließend wird die Feuchtigkeit berechnet, die durch den Transport von Nachbarfeldern aus der aktuellen Zelle zugeführt wird. Dazu wird für alle hinterlegten *MoisturePackages* ermittelt, wie groß der Anteil an der im Sender in der letzten Iteration enthaltenen Feuchtigkeit ist und somit dieser Zelle hinzugefügt werden kann.
Der Anteil errechnet sich nach der Anzahl der Zellen, auf die die Feuchtigkeit aufgeteilt wird, der konkreten Windrichtung, Nutzerparametern und den bisherigen Feuchtigkeitswerten der empfangenden Zellen. Mittels des Parameters *ExpansionFactor* wird festgelegt, wie viel Feuchtigkeit auf die bis zu zwei Nachbarzellen der (nach Windrichtung) eigentlichen Zielzelle entweicht. Somit beeinflusst er, wie gerichtet die Bewegung ist. Die exakte Windrichtung hat Einfluss auf die Verteilung unter den beiden Nachbarzellen und ist im Paket bereits hinterlegt. Ein zweiter Parameter ist der *MoistureRedistributionFactor*, mit dem die Stärke eines Feuchtigkeitsausgleichs aller bis zu drei Zielzellen eines *MoisturePackages* skaliert werden kann. Dazu werden die bisherigen Feuchtigkeitswerte innerhalb der Zellen ermittelt und ihr Durchschnitt berechnet, welchem sich die letztendliche Feuchtigkeit annähern soll. In Kombination der beiden Parameter wird so eine Form von Expansion und Feuchtigkeits- bzw. Konzentrantionsausgleich nachgeahmt.
Die jeweiligen Feuchtigkeitsmengen, die der Zelle zugeführt werden, werden in einem Array abgespeichert und ihre Summe in der zweiten Feuchtigkeits-Map gespeichert. Ist diese Summe unterhalb eines Maximum-Parameters, wird die noch verfügbare Differenz mit der zuvor berechneten Konvektions-Feuchtigkeit aufgefüllt. Diese Abfrage verhindert, dass neue Feuchtigkeit dem System und speziell bereits (über-)vollen Zellen zugefügt wird.
Der nächste Schritt besteht daraus, falls es Feuchtigkeit von Nachbarn gab, für jedes *MoisturePackage* und den entsprechenden Feuchtigkeitsanteil die Menge an Niederschlag zu berechnen, der bei der Luftbewegung entsteht. Die Berechnung ist temperaturabhängig und es gibt wieder zwei mögliche Formeln, eine lineare und eine exponentielle. Entweder wird die Temperaturdifferenz zwischen Start- und Zielzelle mit einem wählbaren Faktor multipliziert und zwischen 0 und 1 geclamped, um den Anteil des Niederschlags der Feuchtigkeit zu ermitteln,

oder die Temperaturdifferenz dient als Exponent für den obigen Faktor.
Schließlich wird der so berechnete Niederschlag vom zweiten Feuchtigkeitsgitter subtrahiert und auf den verbleibenden Rest eine durch wählbare Zufallsparameter bestimmte Menge an Niederschlag berechnet. Dieser wird ebenfalls von der Feuchtigkeitsmenge subtrahiert und die beiden Niederschlagswerte werden in die entsprechende *PrecipitationMap* geschrieben.

4.2.7 Klassifizieren der Biome

Auf Basis der berechneten Temperatur und Niederschläge und unter Zuhilfenahme einer Lookup Table werden die Biome wie in Pseudocode 6 klassifiziert.

Algorithmus 6 Ablauf der Biomklassifizierung

```
function SelectBiomes
    if loadCSV then
        for all CSVEntries do
            if valid then
                store biome id in grid g
            else
                store -1 in g
    for all biomeGridCells do
        get corresponding temperature t and precipitation p
        classify biome according to t and p using g
        if valid then
            cell value = id
        else
            cell value = 0 and release warning
```

Den ersten Schritt stellt das Einlesen der Lookup Table als CSV-Datei und Übertragen der IDs in das Gitter g dar. Sie wird zeilenweise eingelesen, dabei die erste Reihe und Spalte, welche die Beschriftung enthalten, übersprungen und invalide Einträge (z.B. leer oder nicht vom Typ Integer) erhalten den Wert -1. Danach werden für jede Zelle des Biomgitters Temperatur- und Niederschlagswert aus den Gittern der jeweiligen Klassen ausgelesen und daraus das Biom durch Nachschlagen in Gitter g ermittelt. Werte außerhalb des hier festgelegten Wertebereichs bekommen das Biom mit der ID 0 zugewiesen und lösen eine Warnung aus. Alle anderen bekommen das für die Wertekombination hinterlegte Biom, bzw. dessen ID, wobei wiederum alle IDs außerhalb des vorgesehenen Wertbereichs durch 0 ersetzt werden. Dem Nutzer steht es mittels Parameter frei, bei erneuter Klassifikation nicht erneut die Lookup Table einzulesen, was Zeit spart.

Prinzipiell wären ohne viel Aufwand auch eine größere Anzahl an Biomen, andere Wertebereiche oder Auflösungen der Diskretisierung für Temperatur und Niederschlag möglich und bedürften nur einer Änderung der Lookup Table und einiger im Quellcode fest definierter Werte. In diesem Fall wäre stattdessen die Verwendung von einstellbaren Parametern angebracht.

4.2.8 Finalisieren des Terrains

Nachdem die Biome klassifiziert wurden, können die biomspezifischen Terraindetails erzeugt und hinzugefügt werden. Es wird vorausgesetzt, dass der Nutzer Heightmaps bzw. DEMs für die jeweiligen Biome korrekt abgelegt hat, siehe 4.2.2. Wie üblich werden, wie Pseudocode 7 zeigt, zunächst alle nötigen Parameter vom Nutzer eingestellt und als nächstes je Biom eine zufällige der verfügbaren Heightmaps eingelesen und in einem Array abgespeichert. Ebenfalls werden die DEM-Werte falls nötig so verschoben, dass der Minimalwert bei 0 liegt.

Algorithmus 7 Ablauf der Terrainverfeinerung

```
function FinalizeMap
    user adjusts parameters and deposits DEMs
    for all biomes do
        load random DEM file of current biome type
        shift DEM minimal height to zero
    create scaled up (base precision) biome map bs
    create 2D fractal noise
    for horizontal, vertical do
        for all biomeGridCells do
            if biome not equals upper neighbour then
                for all borderCells in bs do
                    calculate amplitude using noise
                    get biome at amplitude-cell ac in bs
                    for all orthoCells between borderCell and ac do
                        set biome in bs similar to ac
    for all heightGridCells do
        calculate kernelSize
        for all kernelCells in heightGrid do
            increment counter c
            increment biome counter bc in array
        for all bc in array do
            calculate biome share by ratio c to bc
            multiply share with DEM value and add it to mixedDEM
        combine mixedDEM and base terrain data using factors
    divide heightmap in tiles
    write tiles into files
```

Danach wird, um die Biomgrenzen unauffälliger und organischer verlaufen zu lassen, die Biom-Map der Klasse *BiomeSelector* auf die höhere Basisauflösung interpoliert und als Gitter *bs* gespeichert, wozu das Nearest-Neighbor-Verfahren verwendet wird, und anschließend die Grenzen selbst mittels Noise verzerrt. Hierzu wird wieder ein 2D-Fractal-Simplex Noise erzeugt, für das der Nutzer die Frequenz einstellen kann.
Der Vorgang des Verzerrens verläuft zweischrittig, erst werden die horizontal, danach die vertikal verlaufenden Biomgrenzen bearbeitet. Jeweils wird für jede Biomregion geprüft, ob es eine entsprechende Grenze nach oben bzw. rechts zu einem anderen Biom gibt. Mit Biomregi-

on werden hier die interpolierten Zellen bezeichnet, die vorher aus einer einzigen bestanden. Ist dies der Fall, wird für jede zur Grenze adjazente Zelle innerhalb der Biomregion die Verschiebung der Grenze anhand des Noise-Wertes, multipliziert mit einem Amplitudenparameter, berechnet. Alle Zellen zwischen der Grenzzelle und der Zelle unter der Amplitude (ac) bekommen ihre Biom-ID, wodurch die Verschiebung realisiert wird.
Anschließend wird das Basisterrain mit den DEM-Daten kombiniert. Dazu wird zellenweise ein Filterkernel auf die detaillierte Biomkarte angewendet, der in einem vom Nutzer wählbaren Umkreis ($kernelSize$) die Vorkommen bc aller Biome aufzählt, um daraus eine Gewichtung zu ermitteln. Die Höhenwerte der DEMs der einzelnen Biome werden daraufhin gemäß der Gewichtung gemittelt, woraus sich der gemeinsame Wert $mixedDEM$ ergibt. Zu beachten gilt, dass für jedes DEM nach Möglichkeit die Koordinaten der Terrain-Heightmap ausgelesen werden und demnach jedes DEM auch mindestens dieselbe Auflösung besitzen sollte. Ist die DEM-Größe geringer, wird das DEM als Fallback so oft wie nötig an den Rand gespiegelt.
Als letztes wird der gemittelte DEM-Höhenwert $mixedDEM$ mit dem des Basisterrains verrechnet, wozu dem Nutzer zwei Gewichte zur Verfügung stehen. Die sich daraus ergebenden Produkte werden hier addiert. So steht dem Nutzer frei, in welcher Gewichtung das Basisterrain und die DEM-Daten zum Tragen kommen, ob der Durchschnitt genommen wird oder eher ein stempelartiger Aufdruck der DEMs. Allerdings kann so das finale Terrain ggf. auch stark von den Basisterrain abweichen.

4.2.9 Platzieren der Assets

Der letzte Schritt der Pipeline besteht aus dem Ermitteln der Asset-Positionen. Der Ablauf ist in Pseudocode 8 abgebildet. Zuerst müssen vom Nutzer die jeweiligen Asset-Klassen im *Unreal Engine*-Editor angelegt und ihre Parameter eingestellt werden. Es wir aufgrund der jeweiligen Anforderungen und Parameter unterschieden zwischen organischen- und anorganischen Assets. Ist dies geschehen, kann der Nutzer sie in einer Liste von den möglichen Biomtypen referenzieren und die zugehörige, angepeilte, Häufigkeit und deren Abweichung pro gleichartigem Biom angeben. Anschließend wird die Datenstruktur zum Speichern der zu generierenden Asset-Positionen initialisiert. Dazu wird für jede Zelle des Positionsgitters, was ein Vielfaches des Biomgitters ist, und für jedes dem Biom zugeordneten Asset ein Struct angelegt, was die jeweiligen Positionen enthalten wird.

Algorithmus 8 Ablauf der Asset-Plazierung

```
function DistributeAssets
    user adds asset classes and adjusts their parameters
    for all biomeTypes do
        user assigns asset classes and their amount
    for all assetGridCells do
        for all assignedAssetClasses do
            initialize by creating struct
    for all 12 AssetCases do
        for all biomeCells do
            for all assetTypes do
                if asset category matches case then
                    if interactSameCategory then
                        increase postponed spawns counter ps
                        calculate randomized final number of spawns fs
                        store asset type and fs in array ats
                    else
                        calculate randomized final number of spawns fs
                        for fs do
                            CalculatePosition()
            for ps do
                select random asset class of ats
                decrease fs of ats
                CalculatePosition()
    for all assetGridCells do
        for all assetClasses do
            store transforms in joint asset class array aca
    for all assetClasses in aca do
        create HISMC
        for all assetTransforms of assetClass do
            spawn instance using HISMC
```

Danach findet das eigentliche Generieren der Asset-Positionen statt, wobei dies ein geschachtelter, sequentieller Prozess ist. Zuerst werden alle anorganischen Assets positioniert, danach erst die organischen. Dies hat den Grund, dass die organischen Objekte keine Kollisionen mit anorganischen haben sollten. Innerhalb der beiden Gruppen findet eine Staffelung nach Größenkategorie statt, erst „Groß“, dann „Mittel“ und schließlich „Klein“, wodurch sichergestellt wird, dass Abhängigkeiten seitens kleinerer Objekte bezüglich der größeren beachtet werden können. In einer dritten, tieferen Ebene wird unterschieden zwischen Assets, die Abhängigkeiten innerhalb ihrer Größenkategorie haben, und denen, die keine haben. Assets mit Abhängigkeiten dürfen erst nach denjenigen platziert werden, die diese nicht haben.
In jedem der sich daraus ergebenden 12 Schritte wird über alle Biome der groben Biom-Map und innerhalb dieser über alle dort zu platzierenden Asset-Klassen iteriert, für die zum Schritt zugehörigen Asset-Klassen die konkrete Anzahl an zu generierenden Positionen nach den gesetzten Parametern berechnet und schließlich jeweils einmal die Subfunktion zur konkreten

Positionsgenerierung ausgeführt. In denjenigen der obigen Schritte, in denen die Assets Abhängigkeiten innerhalb der Größenkategorie haben, wird von dem eben genannten Schema abgewichen und vorerst nur vorgemerkt, wie viele Positionen welcher Asset-Klasse innerhalb der Biomzelle generiert werden sollen und wie hoch die Gesamtzahl an Asset-Positionen ist. Erst nachdem über alle Asset-Klassen einer Biomzelle iteriert wurde, werden die vorgemerkten Generierungen in zufälliger Reihenfolge ausgeführt und jeweils der Zähler herabgesetzt, bis die Gesamtzahl an Positionen aller Assets erreicht wurde.

Algorithmus 9 Ablauf der Positionsgenerierung

```
function CalculatePosition
    determine clustering according to probability
    for all tries do
        if clustering then
            for all spawned asset classes in biomeCell do
                if correct class then
                    add positions to array ppa
            select random parent from ppa
            if interactSameCategory then
                calculate average cluster radius ac with parent
            sample random position in ac around parent
        else
            sample random position in biomeCell
        get height at position
        if height not in valid range then
            abort current try
        calculate terrain angle at position
        if angle out of valid range then
            abort try
        set transform
        for all assetGridCells in 3 × 3 block do
            for all correct asset classes do
                add positions to array nap
        for all entries in nap do
            calculate distance
            if interactSameCategory then
                average repel radii
            if distance < repelRadius then
                abort try
        do above 2 loops roughly again for anorganic repulsion and shadow evaluation
        save transform in assetGridCell
```

Das Generieren einer einzelnen Position läuft iterativ nach dem Dart-Throwing-Prinzip ab, wobei bei jedem Versuch die diversen Constraints nach Asset-Parametern geprüft werden. Der Nutzer kann über den Parameter *tries* bestimmen, wie viele Versuche pro zu generierender Position ausgeführt werden, bevor diese endgültig verworfen wird. Bevor die Schleife

ausgeführt wird, wird jedoch erst anhand des Wahrscheinlichkeitsparameters ermittelt, ob die Asset-Position zufällig oder geclustert gesampled werden soll. Im Fall des clusterings wird nach einem zufälligen, bereits in der Biomzelle platzierten Objekt der gleichen Asset-Klasse oder Asset-Kategorie gesucht, welches als Ausgangsbasis der neuen Position dient. Dazu werden alle Positionen aller in Frage kommender Asset-Klassen in allen Asset-Positions-Zellen der Biomzelle aufgelistet und anschließend zufällig eins ausgewählt. Als möglicher Platzierungsbereich dient dann der Durchschnitt der Parameter *ClusterRadius* beider Objekte. Wird nicht geclustert gesampled, entspricht der Platzierungsbereich der Biomzelle. In beiden Fällen wird eine zufällige Position in dem jeweiligen Bereich gesampled.

Darauf folgt das Prüfen aller Constraints, zuerst, ob Terrainhöhe und Winkel zur Z-Achse an der potentiellen Position im, gemäß den Asset-Parametern, zulässigen Bereich sind. Hierbei wird zusätzlich ein Gradient definiert, in dem die Wahrscheinlichkeit des Verwerfens der Position zum Rand hin, linear zunehmend, ansteigt. Sind Höhe oder Winkel nicht zulässig, wird der aktuelle Platzierungsversuch abgebrochen. Zwischen der Gradientenwahrscheinlichkeit und der Anzahl der Platzierungsversuche gibt es leider eine ungewollte Abhängigkeit, je mehr Versuche es gibt, desto höher ist die Endwahrscheinlichkeit der Platzierung im Gradientenbereich.
An dieser Stelle werden, gemäß der eingestellten Parameter, eine zufällige uniforme Skalierung, Rotation um die sowie Neigung an der Z-Achse durchgeführt und der *Transform* des Objektes abgespeichert.
Danach folgt das Prüfen der Constraints betreffend anderer platzierter Objekte. Zunächst werden innerhalb des umliegenden 3×3 Blocks von Asset-Positions-Zellen alle generierten Asset-Positionen der gleichen Asset-Klasse bzw. der gleichen Größenkategorie aufgelistet, um sie anschließend auf eine Überschneidung, gemäß der *RepelRadius*-Parameter, zu prüfen. Ist die berechnete Distanz zwischen der beiden Asset-Positionen kleiner als der durchschnittliche Abstoßungsradius, wird die neue Position verworfen. Auch hier kommt zusätzlich ein linearer Gradient über die zulässige Distanz hinzu.

Das gleiche Prinzip wird noch zwei Mal in ähnlicher Form angewendet. Einmal werden alle anorganischen Objekte der Größen „Groß“ und „Mittel“ in derselben Umgebung aufgelistet und wie zuvor auf ein Verletzen der Abstoßungsradii geprüft. Schließlich wird nochmals nach allen im 3×3 Block platzierten Objekten gesucht, die in eine größere Kategorie fallen, nach Parameter *ShadeRadius* einen Schatten werfen und deren Distanz zur aktuellen Position kleiner als dieser Schattenradius ist. Für alle platzierten Objekte, auf die dies zutrifft, wird die Stärke des auf die neue Asset-Position geworfenen Schattens berechnet und aufsummiert. Dies geschieht anhand des Parameters *ShadeValue*, der Distanz und eines weiteren Gradienten. Letztlich wird nach bekanntem Schema geprüft, ob der Schattenwert den zulässigen Bereich über- oder unterschreitet oder in den Grenzbereichen nach berechneter Wahrscheinlichkeit die Position ebenfalls verworfen wird.
Wurde die generierte Position des Versuchs nicht verworfen, wird das *Transform* im entsprechenden Gitter abgespeichert.

Um an den gespeicherten Asset-Positionen die Objekte zu spawnen, wird Instancing, konkret *Unreal Engines UHierarchicalInstancedStaticMeshComponent*, verwendet. Es wird pro Mesh eine benötigt, die alle Instanzen des Objektes an den jeweiligen Positionen erzeugt, daher müssen die gespeicherten *Transforms* umstrukturiert werden. Dazu wird über alle Zellen und deren Assetklassen iteriert, jeweils geprüft, ob bereits ein Asset derselben Art vorkam und entsprechend entweder die aktuelle Liste an *Transforms* an der korrekten Stelle in das bestehende Array *aca* eingefügt oder das Array um diese Klasse mit ihren *Transforms* erweitert. So kann schließlich je Asset-Klasse eine *UHierarchicalInstancedStaticMeshComponent* erzeugt und in einer Schleife alle Instanzen gemäß der *Transforms* generiert werden.

Nachdem die Implementierung hinreichend diskutiert wurde, folgt eine ausführliche Komplexitätsanalyse.

4.3 Komplexitätsanalyse

In diesem Abschnitt wird die Komplexität der Pipeline und der einzelnen Schritte, gemäß der zuvor beschriebenen Implementierung, analysiert. Dazu werden nach und nach die relevanten Variablen eingeführt und nach folgendem Schema definiert:

Variable	Beschreibung
CX_{noise}	Komplexität der Noise-Berechnung
n_b	Anzahl Zellen des Basisgitters pro Dimension
C_{calcH}	Kosten der Höhenberechnung einer Zelle
$Cell_{calcWt}$	Kosten der Wasserberechnung einer Zelle
CX_{proxy}	Komplexität des Erstellens des Proxy Meshes
CX_h	Komplexität der gesamten Basisterrain-Generierung

Anhand der jeweils definierten Variablen wird dann für jeden Pipelineschritt die Komplexität bezüglich der Laufzeit ermittelt und anschließend daraus die Gesamtkomplexität berechnet. Ebenfalls wird am Ende auf die Speicherplatzkomplexität eingegangen.

Den Anfang macht die Komplexität der Generierung des Basisterrains:

$$CX_h = CX_{noise} + n_b^2 \cdot C_{calcH} + n_b^2 \cdot C_{calcWt} + CX_{proxy} = O(n_b^2)$$

Hierbei wird für die Simplex-Noise-Generierung CX_{noise} eine quadratische Laufzeit angenommen, wie in 2.1.3 genannt, und da sie die gleiche Auflösung wie n_b besitzt, kann sie als Faktor 2 vernachlässigt werden.
Die Laufzeit der Generierung des Proxy Meshes CX_{proxy} ist nicht bekannt, kann aber außen vor gelassen werden, da es kein zwingender Bestandteil der Berechnungen ist und nur der Visualisierung dient.

Variable	Beschreibung
n_c	Anzahl der Zellen der Klimasimulation pro Dimension
C_{calcT}	Kosten der Temperaturberechnung einer Zelle
C_{calcTd}	Kosten der Berechnung des Temperaturabfalls einer Zelle
CX_t	Komplexität der gesamten Temperaturberechnung

Die Zeitkomplexität der Temperaturberechnung ergibt sich nach:

$$CX_t = n_c^2 \cdot C_{calcT} + n_c^2 \cdot C_{calcTd} = O(n_c^2)$$

Variable	Beschreibung
C_{initW}	Kosten der Initialisierung einer Windzelle
C_{calcW}	Kosten der Berechnung der Windrichtung einer Zelle
CX_w	Komplexität der gesamten Windberechnung

Die Zeitkomplexität der Windberechnung hingegen durch folgende Formel:

$$CX_w = n_c^2 \cdot C_{initW} + n_c^3 \cdot C_{calcW} = O(n_c^3)$$

Variable	Beschreibung
C_{calcTP}	Kosten der Berechnung der Transportpakete einer Zelle
C_{calcP}	Kosten der Berechnung der Feuchtigkeit und des Niederschlages einer Zelle
CX_p	Komplexität der gesamten Niederschlagsberechnung

Die Zeitkomplexität der Niederschlagsgenerierung berechnet sich durch:

$$CX_p = n_c^2 \cdot C_{calcTP} + n_c^3 \cdot C_{calcP} = O(n_c^3)$$

Variable	Beschreibung
n_{csv}	Anzahl der Einträge der CSV-Tabelle pro Dimension
C_{read}	Kosten des Einlesens eines CSV-Eintrags
$C_{classify}$	Kosten des Klassifizierens einer Biomezelle
CX_b	Komplexität der gesamten Biomklassifizierung

Die Zeitkomplexität der Biomklassifizierung ergibt sich wie folgt:

$$CX_b = n_{csv}^2 \cdot C_{read} + n_c^2 \cdot C_{classify} = O(n_{csv}^2 + n_c^2)$$

Variable	Beschreibung
n_{bio}	Anzahl der Biomtypen
$C_{loadImg}$	Kosten des Einlesens einer Bilddatei und Verschieben der Werte
C_{scale}	Kosten des Skalierens einer Biomzelle
$C_{distort}$	Kosten des Berechnens der Biomgrenzen-Verzerrung einer Biomzelle
n_{amp}	Amplitude in Zellen des Basisgitters
$C_{setBiome}$	Kosten des Biomwechsels einer Zelle
C_{calcHN}	Kosten des Berechnens der neuen Höhe einer Zelle
n_{ker}	Anzahl der Zellen des Kernels pro Dimension in Basisauflösung
C_{count}	Kosten des Setzens von Zählern
C_{avg}	Kosten der Durchschnittsberechnung
CX_f	Komplexität der gesamten Terrainfinalisierung

Die Zeitkomplexität der Terrainfinalisierung wird durch folgende Formeln beschrieben:

$$C_{distort} = \frac{n_b}{n_c} \cdot n_{amp} \cdot C_{setBiome}$$

$$C_{calcHN} = n_{ker}^2 \cdot C_{count} + n_{bio} \cdot C_{avg}$$

$$\begin{aligned} CX_f &= n_{bio} \cdot C_{loadImg} + n_c^2 \cdot C_{scale} + CX_{noise} + 2 \cdot n_c^2 \cdot C_{distort} + n_b^2 \cdot C_{calcHN} \\ &= O(n_b^2 \cdot n_{ker}^2) \end{aligned}$$

Wobei $C_{loadImg}$ eine quadratische Laufzeit bezogen auf die Seitenlänge in Pixeln hat. Da diese als vergleichbar der von n_b angenommen werden kann, entfällt sie als Faktor 2 des Ergebnisses.

Variable	Beschreibung
n_a	Anzahl der Zellen des Asset-Gitters pro Dimension
n_{ab}	Anzahl der Asset-Klassen eines Bioms
C_{initAC}	Kosten des Initialisierens einer Zelle
n_{pab}	Anzahl der zu generierenden Positionen eines Assets einer Biomzelle
CX_{calcP}	Komplexität der Berechnung einer Asset-Position
CX_a	Komplexität der gesamten Asset-Platzierung
n_t	Anzahl der Versuche, eine valide Asset-Position zu generieren
C_{const}	Kosten des Berechnens und Prüfens von Höhen- und Winkelanforderungen
C_{constB}	Kosten des Berechnens und Prüfens von Distanz- und Schattenanforderungen
C_{spawn}	Kosten des Spawnens einer Asset-Instanz
CX	Komplexität der gesamten Pipeline

Die Zeitkomplexität der Asset-Platzierung berechnet sich nach:

$$CX_a = n_a^2 \cdot n_{ab} \cdot C_{initAC} + 12 \cdot n_c^2 \cdot n_{ab} \cdot n_{pab} \cdot CX_{calcP} + n_c^2 \cdot n_{ab} \cdot n_{pab} \cdot C_{spawn}$$

mit $CX_{calcP} = n_t \cdot (C_{const} + 4 \cdot n_{ab} \cdot n_{pab} \cdot C_{constB})$
woraus sich ergibt: $O(n_a^2 + (n_{ab} \cdot n_{pab})^2 + n_c^2)$

Durch Kombination der jeweiligen Komplexitäten lässt sich die Gesamtkomplexität bzgl. der Zeit formulieren durch:

$$\begin{aligned} CX &= CX_h + CX_t + CX_w + CX_p + CX_b + CX_f + CX_a \\ &= n_b^2 + n_c^2 + n_c^3 + n_c^3 + n_{csv}^2 + n_c^2 + n_b^2 \cdot n_{ker}^2 + n_a^2 + (n_{ab} \cdot n_{pab})^2 + n_c^2 \\ &= O(n_b^2 \cdot n_{ker}^2 + n_c^3 + n_{scv}^2 + n_a^2 + (n_{ab} \cdot n_{pab})^2) \\ &= O(n_c^3) \end{aligned}$$

Für die Gesamtspeicherkomplexität CX_s ergibt sich über alle Pipelineschritte, unter Vernachlässigung temporär erzeugter Daten und einzelner Variablen und Referenzen:

$$\begin{aligned} CX_s = &(n_b^2 + n_c^2) \cdot (f + i) \\ &+ n_c^2 \cdot f \\ &+ n_c^2 \cdot 7 \cdot f \\ &+ n_c^2 \cdot f + n_{csv}^2 \cdot i + n_{bio} \cdot fn \\ &+ n_a^2 \cdot n_{ab} \cdot (p + n_{pab} \cdot t) + n_{bio} \cdot (fn + n_{ab} \cdot (f + i + p)) \\ &= (10 \cdot n_c^2 + n_b^2 + n_{bio} \cdot n_{ab}) \cdot f + (n_b^2 + n_c^2 + n_{csv}^2 + n_{bio} \cdot n_{ab}) \cdot i + 2 \cdot n_{bio} \cdot fn \\ &+ n_a^2 \cdot n_{ab} \cdot n_{pab} \cdot t + (n_a^2 + n_{bio}) \cdot n_{ab} \cdot p \\ &= O(n_c^2 + n_b^2 + n_{scv}^2 + n_a^2) \end{aligned}$$

wobei f für einen *float*, i für einen *integer*, b für einen *bool*, fn für einen *FName*, t für einen *FTransform* und p für einen *Pointer* steht.
Dazu kommen die DEMs, die Asset-Objekte, ein paar Texturen und Farbwerte zur Visualisierung sowie andere von/in der *Unreal Engine* erzeugte Objekte wie dem Proxy Mesh und den *HISMC*s. Ein Asset-Objekt besteht aus einem Mesh, sowie $i + b + 20 \cdot f$.

Nachdem die Implementierung besprochen und die Komplexität untersucht wurde, folgt im nächsten Kapitel die ausführliche Darstellung, Analyse und Diskussion der Ergebnisse.

Kapitel 5

Ergebnisse und Evaluation

In diesem Kapitel werden die Ergebnisse der selbstgeschriebenen Implementierung vorgestellt, analysiert und ausgewertet. Es wird damit begonnen, in Abschnitt 5.1, die Versuchsdurchführung und die Resultate bezüglich des Stiltransfers mit Neuronalen Netzen vorzustellen. Danach folgt in Abschnitt 5.2 die Präsentation der Ergebnisse der einzelnen Pipelineschritte sowie deren ausführliche Auswertung. Anschließend wird in Abschnitt 5.3 die implementierte Pipeline ganzheitlich diskutiert und evaluiert sowie mit existierenden Anwendungen verglichen.

Alle Tests wurden auf einem System bestehend aus einem Intel Xeon X3440 (4 Kerne, 8 Threads, 3,4 GHz, SSE 4.2) Prozessor, einer NVIDIA GeForce GTX 1060 6 GB Grafikkarte, 8 GB Arbeitsspeicher und Windows 7 x64 durchgeführt.

5.1 Experimente mit Neuronalen Netzen

Im Verlauf dieser Arbeit wurde, wie bereits geschrieben, die Eignung von Neuronalen Netzen zur Kombination von unterschiedlichen Heightmaps erprobt. Konkret wurde die Technik des Stiltransfer mittels CNNs angewendet. Mehr über den Stiltransfer und CNNs in Abschnitt 2.5.1. Hierzu wurden zwei bestehende Tensorflow-Implementierungen verwendet, *neural-style*[1] von Anish Athalye und *Fast Style Transfer*[2] von Logan Engstrom. Beide Versionen nutzen das vortrainierte VGG19-Netz.
Als Trainingsdatensatz wurden einerseits reale DEM-Daten von zufällig über der Erde gewählten Ausschnitten und andererseits selbst generierte Fractal-Simplex und Cellular Noisemaps genutzt. Alle Bilder lagen als 16 Bit Graustufen-Heightmap vor, insgesamt ca. 5000 Stück.

Das Ziel war es, die Neuronalen Netze darauf zu trainieren, den „Stil“ von Heightmaps, etwa Erosionsmuster, typische Wellenmuster von Hügelketten oder Dünen einer Wüste, auf eine bestehende grobe Heightmap zu transferieren und dabei ihre Struktur beizubehalten. Damit

[1] `https://github.com/anishathalye/neural-style` (Abgerufen am 11.12.2018)
[2] `https://github.com/lengstrom/fast-style-transfer` (Abgerufen am 11.12.2018)

ist das Ziel in dieser Arbeit ganz ähnlich des eigentlichen und in den jeweiligen Veröffentlichungen vorgestellten Einsatzzweckes, den Malstil eines Bildes auf die grobe Struktur eines anderen zu übertragen.

In Abbildung 5.1 sind einige Resultate dargestellt. Die obere Reihe zeigt die Quelldaten, die untere die Ergebnisse.

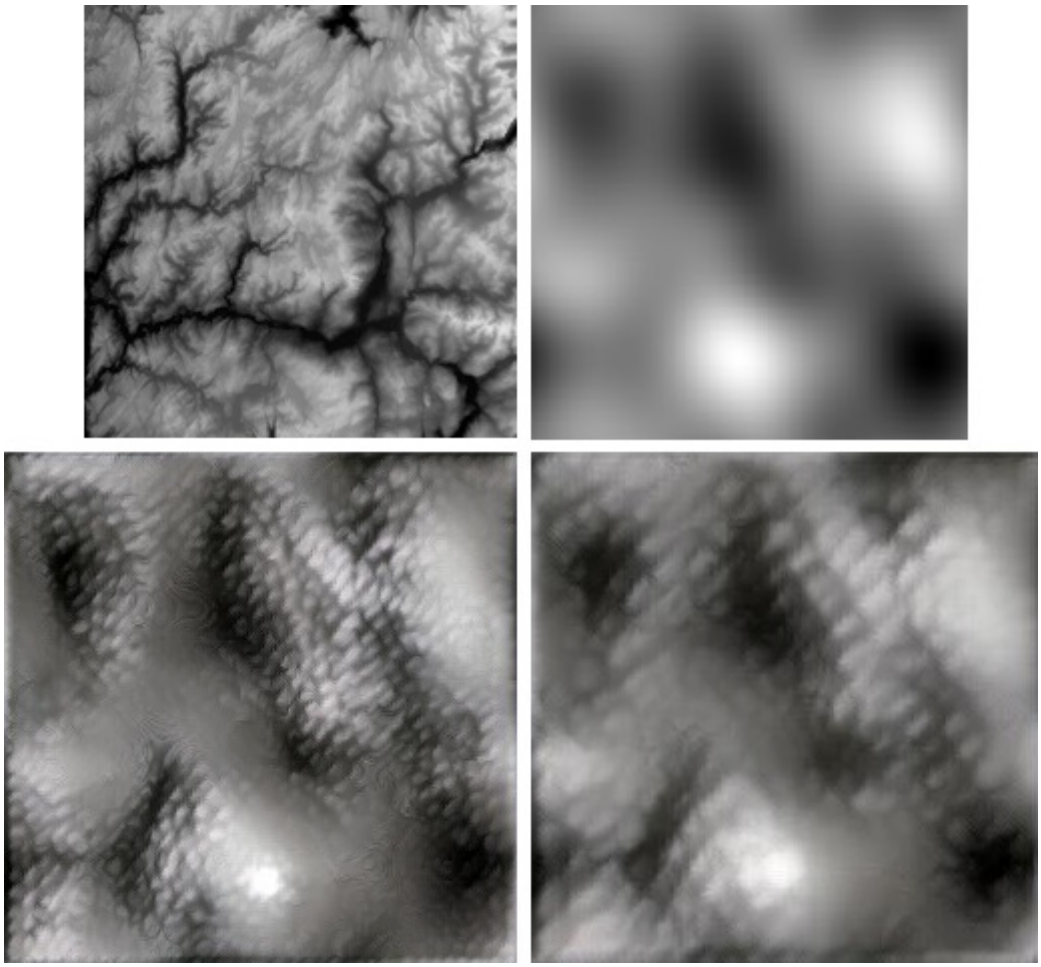


Abbildung 5.1 In der unteren Reihe zwei Ergebnisse in unterschiedlichen Auflösungen von *Fast Style Transfer* zur Heightmap-Kombination. Angewendet auf das Quellbild oben rechts mit dem Stilbild wie oben links.

Es ist deutlich erkennbar, wie das Neuronale Netz das Quell- und Stilbild genutzt hat, um die neuen Bilder zu erzeugen. Die Grundstruktur des Quellbildes ist erhalten geblieben, im Direktvergleich ist leicht zu sehen, dass die größeren weißen und schwarzen Regionen auch im Zielbild in ähnlicher Form vorhanden sind. Hier nun allerdings um ein Muster erweitert und überlagert. Dieses ähnelt den schwarzen, sich verzweigenden Erosionsmustern des Stilbildes. Das Neuronale Netz hat damit seine Aufgabe erfüllt und den Stil eines Bildes auf die grobe Struktur eines anderen transferiert, das generierte Bild entspricht dem zu erwartenden und erhofften Ergebnis.

Jedoch genauer betrachtet muss man sagen, dass das übertragende Muster nicht wirklich wie

Erosionsspuren aussieht. Im Detail fallen die Unterschiede auf, es ist zu regelmäßig, mit zu vielen, gleichförmigen Verzweigungen, die wieder zusammen führen. In der Struktur gleicht es etwas einem Cellular Noise. Weiterhin sind Artefakte erkennbar, die sich vor allem in den mittelgrauen Übergangsregionen von auslaufenden, großen schwarzen Bereichen bilden. Besonders gut zu sehen sind die Artefakte in dem linken Ergebnisbild welches eine höhere Auflösung besitzt. Ein drittes, stark auffallendes Problem sind die weißen Fehlbereiche, hier im unteren Bereich sichtbar. Offensichtlich treten sie in Bereichen mit hoher Helligkeit im Quellbild auf.
Speziell zur Nutzung als Heightmap und der direkten Landschaftsgenerierung, wie es in diesem Anwendungsfall vorgesehen ist, lassen die drei genannten Probleme eine Verwendung der so generierten Bilder nicht zu, da die Bildfehler und der unrealistische Terrainstil hier sehr deutlich negativ auffallen würden.
Die oben genannten Probleme treten so oder in ähnlicher Form auch mit anderen Quellbildern und anderen Parametrisierungen auf. Die fehlerhaften, weißen Regionen sind wahrscheinlich ein Resultat des Umstandes, dass die verwendeten Netze auf die Verarbeitung von RGB-Bildern mit drei Farbkanälen ausgelegt sind, hier aber auf Grauwertbilder mit einem Kanal angewendet werden. Mit Anpassungen der Netze und Algorithmen ist das Problem jedoch voraussichtlich behebbar. Bezüglich der Artefakte und dem nur suboptimal gelernten Muster lässt sich sagen, dass das konkrete Ergebnis immer stark abhängig von den verwendeten Quell- und Stilbildern sowie den beim Training genutzten Parametern ist und in diesem Bereich deutlich mehr Tests und Experimente nötig sind, um bessere Resultate zu erzielen.
Was die Laufzeit betrifft, so hat das Training mehrere Stunden gedauert, wobei die Laufzeit stark von den Parametern und der Menge an Trainingsdaten abhängig ist. Die Anwendung eines gelernten Stils auf ein neues Bild dauert hingegen, wie bei Neuronalen Netzen üblich, nur wenige Sekunden und wäre somit im Rahmen des in dieser Arbeit gesteckten Zielhorizontes. Ein Training eines Stils pro Biom wäre als Vorverarbeitungsschritt einmalig nötig und vertretbar, wenn die Fehler in Zukunft behoben bzw. umgangen werden können und das System genutzt werden sollte.

Leider stellte sich während der begrenzten Versuche, wie man an den gezeigten Beispielbildern erkennt, heraus, dass die bestehenden Stiltransfer-Implementierungen nicht ohne Modifizierungen für den hier bestehenden Anwendungsfall geeignet sind. Es ist jedoch deutlich das Potenzial erkennbar. Mit ausgedehnteren, gesonderten Versuchen, ggf. tiefgreifenderen Änderungen an dem Aufbau der Netze und umfangreichen Testreihen mit verschiedenen Parametern können sicher bessere Ergebnisse erzielt werden.

Nachdem hier die durchwachsenen Ergebnisse mit Neuronalen Netzen dargelegt wurden, werden im weiteren Verlauf die Ergebnisse aller Pipelineschritte ausgewertet.

5.2 Einzelergebnisse und Auswertung

Nachfolgend werden in diesem Abschnitt die Ergebnisse aller Pipelineschritte demonstriert und analysiert, sowohl visuell als auch laufzeittechnisch.

Die grafische Auswertung der Zwischenschritte der Pipeline wird anhand des implementierten Proxy Meshes (mit 128^2 Vertices), samt seiner Overlay-Texturen, durchgeführt. Für alle Laufzeitmessungen wurden jeweils drei (oder mehr) vollständig voneinander unabhängige Durchläufe im Editor durchgeführt, mit dem internen Profilingtool der *Unreal Engine* die Laufzeiten aufgezeichnet und der Median der jeweiligen Messwerte zur weiteren Analyse herangezogen. Zuwachsfaktoren beziehen sich, falls nicht anders angegeben, immer auf die vorherige Auflösung.

Erzeugen des Basisterrains In Abbildung 5.2 ist das Proxy Mesh mit einem generierten Basisterrain dargestellt, welches beispielhaft für das Ergebnis dieses Pipelineschrittes steht. Durch die Farben Gelb und Blau wird das Land und Wasser, klassifiziert durch einen Schwellwert, visualisiert. Anhand des Terrains ist deutlich die Generierungsmethode (Simplex Noise) erkennbar. In diesem Fall mit einer moderat hohen Frequenz, jedoch für diesen Anwendungsfall schon eher am oberen Rand des sinnvollen Bereichs. Für sich genommen wäre dies kein gutes, realistisches Terrain, aber da dies nicht das Endresultat ist, sondern nur als grobe Basis für weitere Schritte dient, ist dies kein Problem. Eventuell könnte man dennoch für die Zukunft über eine Kombination mit einem 2. Noise, wie z.B. Cellular Noise, nachdenken, um es als eine Art Maske zu nutzen und die Regelmäßigkeit zu umgehen sowie dem Nutzer mehr Gestaltungsmöglichkeiten zu bieten.

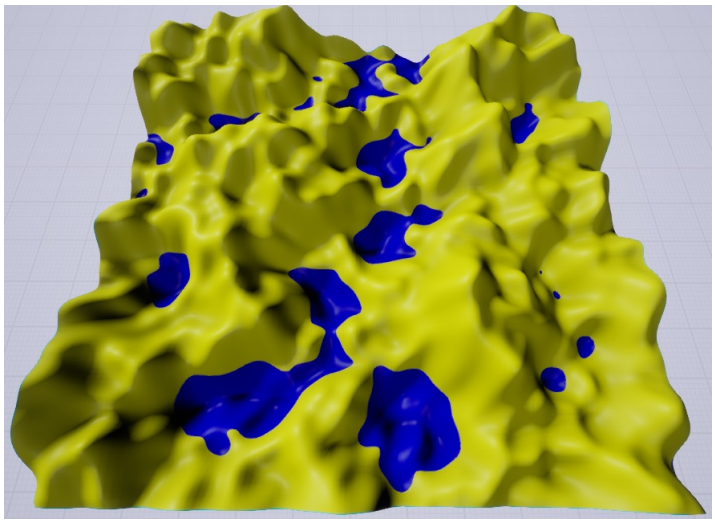


Abbildung 5.2 Das generierte, grobe Terrain, dargestellt durch das Proxy Mesh mit den auch in allen weiteren Schritten voreingestellten 128^2 Vertices. Blau markiert sind Bereiche, die Wasser darstellen.

In der unten abgebildeten Tabelle 5.1 sind die Messungen des Zeitverhaltens der Generierung des Basisterrains aufgeführt. Es sind über drei unterschiedliche Auflösungen die Gesamtlaufzeit[3], die Berechnung der Höhe, inklusive der Noise-Generierung, und das Erzeugen der niedriger aufgelösten, bzw. unterabgetasteten Version („Unterab.“) aufgeführt. Zudem werden in der Tabelle auch die jeweiligen Zuwachsfaktoren angegeben. Es ist festzustellen, dass das Berechnen der Höhendaten in beiden Versionen in etwa ein Drittel der Gesamtlaufzeit ausmacht, wobei der Großteil davon auf das Berechnen der hoch aufgelösten Version entfällt. Zudem werden etwa 80% davon wiederum für die Noise-Generierung verwendet. Die restlichen zwei Drittel der Gesamtlaufzeit verbrauchen Visualisierungsaufgaben, wie das Erzeugen von Overlay-Texturen (knapp ein Drittel der restlichen Zeit) und das Schreiben von der Heightmap in eine Bilddatei. Letzteres wäre nicht zwingend notwendig und das Weglassen würde etwa knapp 25% der restlichen Zeit einsparen. Wie nach der Komplexitätsanalyse in Abschnitt 4.3 festgestellt, liegt durchweg ein quadratisches Wachstum vor, ersichtlich an den Wachstumsfaktoren von durchschnittlich etwa 4.

	Zellen pro Achse				
	1024	2048		4096	
Schritte	Zeit (ms)	Zeit (ms)	Zuwachsfaktor	Zeit (ms)	Zuwachsfaktor
Gesamt[3]	240,5	956,0	3,97	3940	4,12
Höhe	72,75	294,4	4,05	1258	4,27
Unterab.	15,3	52,17	3,41	211,5	4,05

Tabelle 5.1 Laufzeit der Generierung des Basisterrains über mehrere Auflösungen hinweg und die resultierenden Zuwachsfaktoren (diese bezogen auf die jeweils vorige Auflösung).

Das Diagramm 5.3 veranschaulicht die Daten der obenstehenden Tabelle.

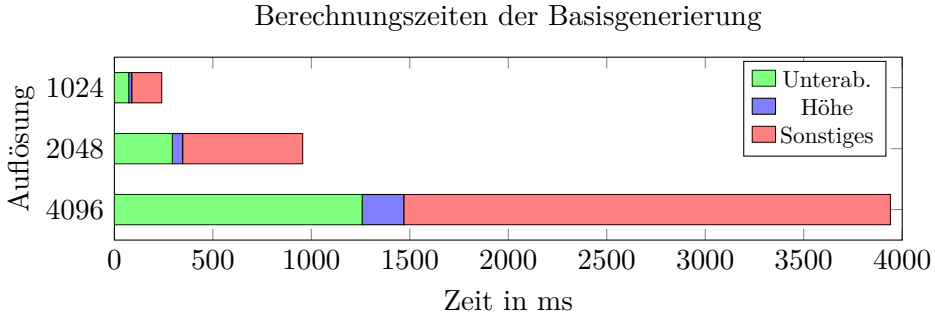


Abbildung 5.3 Diagramm der Tabelle 5.1.

Die Generierung des Proxy Meshes ist voreingestellt auf maximal 128^2 Vertices, was im Mittel 3726 ms dauert. Bei 64^2 Vertices ergibt sich hingegen eine Zeit von 265,6 ms, was einen Faktor von 14,0 ergibt. Damit ist die Skalierung der Laufzeit des Proxy Meshes extrem schlecht und es wird verdeutlicht, warum es wichtig ist, den entsprechenden Parameter für den Faktor der

[3]Was in der oben sogenannten Gesamtlaufzeit noch nicht enthalten ist, ist die Laufzeit der Generierung des Proxy Meshes und des optionalen Erzeugens und Schreibens der Heightmap Tiles.

Proxy Mesh-Auflösung im Auge zu behalten. Mehr als die eingestellten 128^2 Vertices werden nicht empfohlen. Tatsächlich entfällt intern fast die gesamte Zeit auf das Aktualisieren der Kollision, eventuell ist es möglich, dies in Zukunft zu verhindern, da diese nicht notwendig ist.
Das Schreiben der optionalen Heightmap Tiles hingegen skaliert zum einen quadratisch mit der Gesamtauflösung und zum anderen linear mit der Anzahl der Tiles pro Achse. Es empfiehlt sich, die Zahl der Tiles möglichst gering zu halten, um Rechenzeit zu sparen. Das Schreiben von $4^2 = 16$ Tiles mit einer Gesamtauflösung von 2048 dauert nur 249,9 ms, verachtfacht man im Extremfall die Anzahl der Tiles pro Seite auf $32^2 = 1024$, ergibt sich eine deutlich längere Zeit von 2065,5 ms. Bei einer Auflösung von 4096 und $3^2 = 9$ Tiles dauert das Schreiben der Tiles hingegen 878,1 ms.

Temperaturberechnung Die Ergebnisse der Temperaturberechnung sind in Abbildung 5.4 illustriert, links einmal mit der Interpolationsmethode, rechts mit dem Sinusmodus. Die Temperaturen sind farblich über dem Proxy Mesh kodiert, Blau ist am kältesten, gefolgt von Grün, Gelb und schließlich Rot, was die wärmsten Bereiche repräsentiert. Die generell dunkleren Bereiche sind Schatten des Terrains, hellere dagegen Reflexionen und beide zu ignorieren.

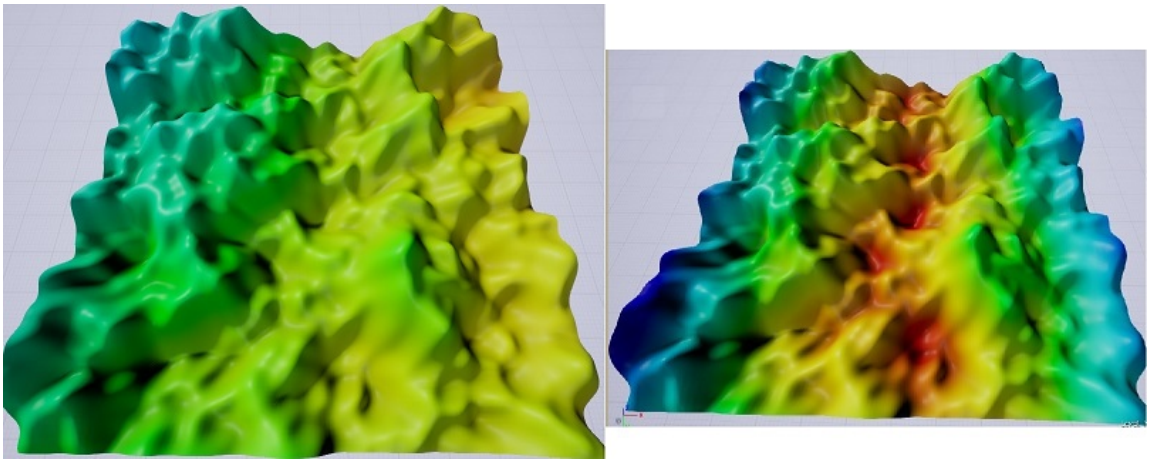


Abbildung 5.4 Links: Die mit bilinearer Interpolation berechneten Temperaturen werden farblich repräsentiert auf dem Proxy Mesh dargestellt. Man beachte den Temperaturabfall auf Berggipfeln. Rechts: Eine alternativ mit dem Sinusmodus und verstärktem Temperaturabfall in der Höhe erzeugte Temperaturkarte.

Man kann anhand der Farbverläufe gut erkennen, wie zum einen die Temperaturen von den Ecken her interpoliert werden und sich über dem Terrain homogen verteilen, bzw. im Sinusmodus sich ein hier horizontaler Verlauf einstellt, und zum anderen, wie die Temperatur an höheren Stellen lokal abnimmt und in Tälern tendenziell höher ausfällt. Somit bildet der Algorithmus, vernünftige Nutzerparameter vorausgesetzt, das höhenabhängige Verhalten in der Realität sehr gut ab.

In Tabelle 5.2 wird das Laufzeitverhalten der Temperaturberechnung aufgeführt. Es werden die Zeiten des gesamten Pipelineschrittes sowie der konkreten Temperaturberechnung für drei Auflösungen und die daraus resultierenden Skalierungsfaktoren dargestellt. Das Berechnen der Temperatur hat einen vergleichsweise sehr kleinen Anteil an der Gesamtlaufzeit, wobei alle Werte absolut betrachtet sehr gering sind. Die Skalierung schwankt stark, aber um den theoretischen, quadratischen Faktor von $2^2 = 4$ herum. Die hohen Schwankungen kommen sicherlich dadurch zustande, dass die Werte absolut nur wenige Millisekunden betragen und hier kleine Unterschiede im Verhalten des Systems, des Editors oder des internen Profilling-tools zum Tragen kommen. Zudem beinhalteten die Daten ursprünglich das Schreiben zweier Bilder zur Visualisierung, die hier zwar separat gemessen und herausgerechnet wurden, aber durch ihrerseits merkbare Schwankungen die hier abgebildeten Daten leicht beeinflusst haben können.

	Zellen pro Achse				
	128	256		512	
Schritte	Zeit (ms)	Zeit (ms)	Zuwachsfaktor	Zeit (ms)	Zuwachsfaktor
Gesamt	1,427	7,224	5,06	36,268	5,02
Temperatur	0,223	0,856	3,84	3,868	4,52

Tabelle 5.2 Laufzeit der Temperaturberechnung über mehrere Auflösungen hinweg und die resultierenden Zuwachsfaktoren.

Das Diagramm 5.5 illustriert die Daten der obenstehenden Tabelle.

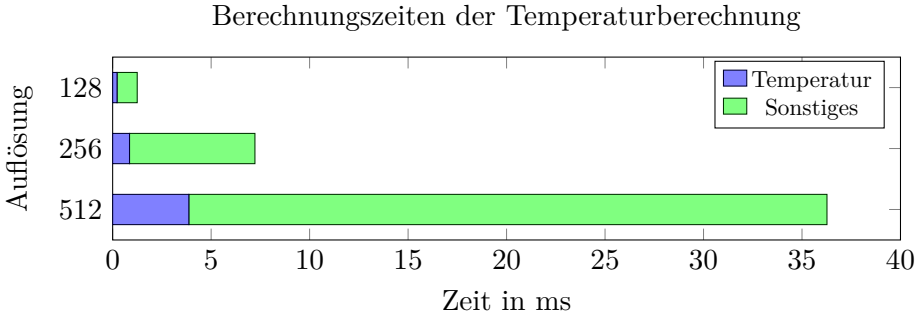


Abbildung 5.5 Diagramm der Tabelle 5.2.

Windberechnung Die berechneten Windrichtungen werden durch ein Array von Pfeilen über dem Terrain dargestellt, wie in Abbildung 5.6 (links) ersichtlich. Im rechten Bild ist eine Draufsicht mit flach dargestellter Landschaft zur besseren Sicht auf das Pfeil-Array und den Verlauf der Windrichtungen illustriert. Die Länge der Pfeile, und damit die Windstärke, wird in den Berechnungen nicht berücksichtigt.

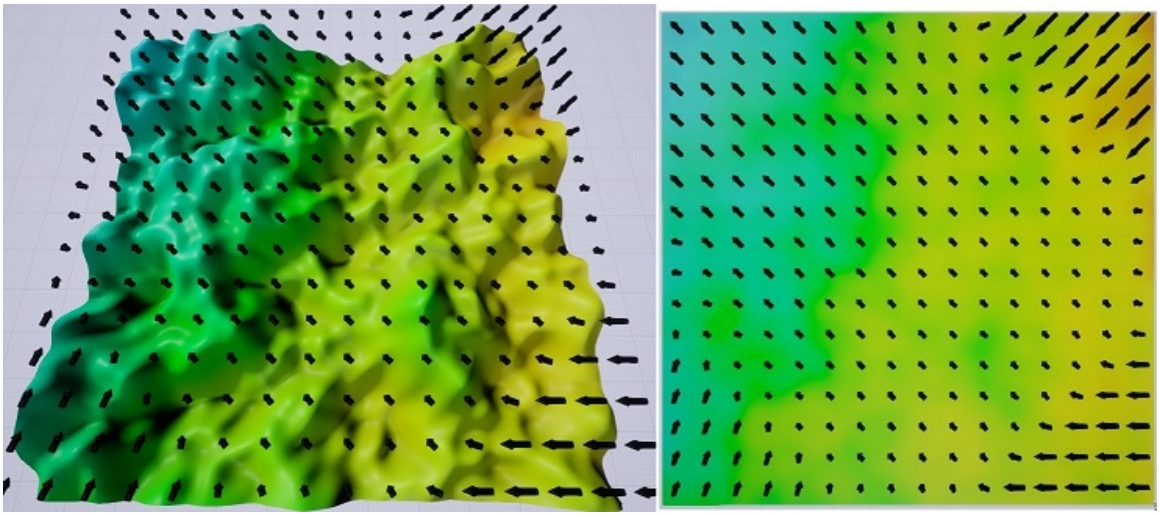


Abbildung 5.6 Links: Die berechneten Windrichtungen werden durch ein Gitter von Pfeilen über dem Terrain dargestellt. Rechts: Dieselben Daten als Draufsicht und mit flacher Landschaftsdarstellung.

Man erkennt gut, dass die eingestellten Windrichtungen an den Ecken sich zur Mitte hin homogen und relativ plausibel vermischen und die drei eingehenden Windströmungen umgeleitet werden zur ausgehenden in der linken, oberen Ecke. Die zufällige Randomisierung von Windrichtungen ist hier nicht ersichtlich, da die Pfeile zur Visualisierung jeweils mehrere Zellen und ihre Windrichtungen mitteln. Verbesserungswürdig ist jedoch die distanzabhängige Funktion zur Betrachtung der externen Kräfte, denn der Verlauf von den Ecken zur Mitte hin scheint recht abrupt zu sein.

In Tabelle 5.3 sind die Laufzeitmessungen für diesen Schritt über die drei Auflösungen aufgelistet. Die reine Laufzeit der eigentlichen, iterativen Berechnung entsprach nahezu der hier dargestellten Gesamtlaufzeit des Pipelineschrittes, weswegen sie nicht extra dargestellt wird. Die hier mit eingerechneten Verwaltungs- und Visualisierungsaufgaben machen bloß einen vernachlässigbaren Anteil von unter 1%, bzw. in der geringsten Auflösung einen kleinen einstelligen Prozentsatz, am Gesamtanteil aus. Die Daten passen gut zu der im Abschnitt 4.3 festgestellten kubischen Komplexität, man beachte, dass die Zuwachsfaktoren nahe dem theoretischen Wert von $2^3 = 8$ liegen, bzw. sich ihm mit steigender Auflösung annähern.

	Zellen pro Achse				
	128	256		512	
Schritte	Zeit (ms)	Zeit (ms)	Zuwachsfaktor	Zeit (ms)	Zuwachsfaktor
Gesamt	148,7	1127	7,58	8840	7,84

Tabelle 5.3 Laufzeit der Windberechnung über mehrere Auflösungen hinweg und die resultierenden Zuwachsfaktoren.

Diagramm 5.7 illustriert die gemessenen Daten.

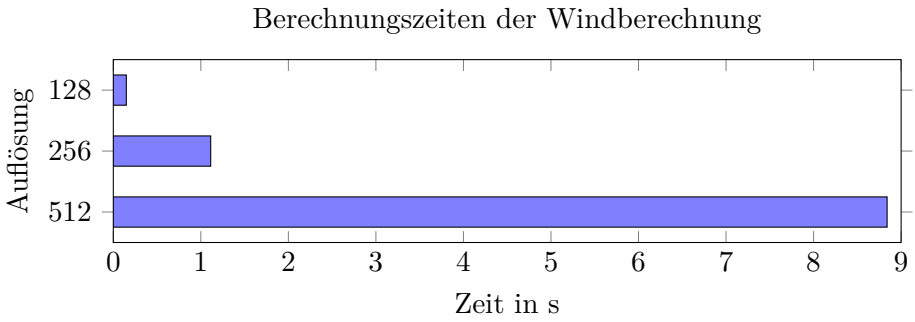


Abbildung 5.7 Diagramm der Tabelle 5.3.

Niederschlagsberechnung Die Verteilung bzw. Höhe der Feuchtigkeit und Niederschläge, die sich nach der Berechnung ergibt, ist wieder farblich kodiert über dem Proxy Mesh dargestellt, vgl. Abbildung 5.8. Hier jedoch sind die Farben in umgekehrter Reihenfolge, Rot stellt die niedrigsten, Blau die höchsten Werte dar. Das linke Bild zeigt die berechnete Feuchtigkeit und das rechte die sich daraus ergebenden Niederschläge.

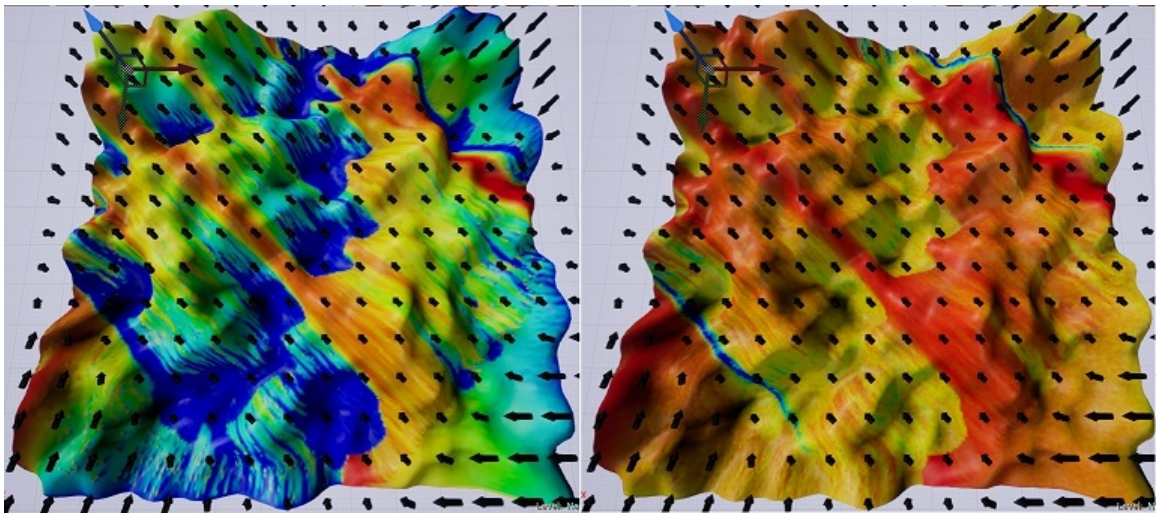


Abbildung 5.8 Links wird die berechnete Feuchtigkeitsmenge farblich kodiert über dem Terrain abgebildet, rechts dagegen die daraus resultierenden Niederschlagsmengen. Blau steht für hohe Werte, rot für niedrige, man beachte die erhöhten Werte auf den Wind zugewandten Berghängen.

Die dunkelblauen Bereiche im linken Bild stellen überwiegend die Gebiete mit Wasser dar, in denen Konvektion vorkommt, vgl. Abbildung 5.2. Zusätzlich ist vom Kartenrand einströmende Feuchtigkeit eingestellt. Diese Feuchtigkeit wird durch die Winde überwiegend in Richtung Nordwesten transportiert. Es ist ersichtlich, dass die Feuchtigkeit mit der Zeit durch Niederschläge abnimmt und dass diese verstärkt an den Wind zugewandten Berghängen vorkommt (hellgelbe und grüne Bereiche im rechten Bild) und auf den Gipfeln und abgewandten Seiten geringer ausfällt, was dem Phänomen des Regenschattens entspricht.

Zudem ist anhand der farblich kodierten Werte deutlich die Windrichtung zu erkennen. Eventuell wäre eine stärkere unidirektionale Expansion der Feuchtigkeit sinnvoll, was auch die beiden länglichen, unplausiblen Bereiche mit starken Niederschlägen (rechtes Bild) abmildern würde. Diese kommen wohl dadurch zustande, dass sich hier Feuchtigkeit aus mehreren Richtungen (speziell durch den orthogonal auftreffenden Wind von den Ecken aus) vereinigt.

Tabelle 5.4 zeigt die Laufzeitmessungen für die Feuchtigkeits- und Niederschlagsberechnung in den bekannten drei Auflösungen. Die reine Laufzeit der eigentlichen, iterativen Berechnung entsprach wie bei der Windberechnung in etwa der hier dargestellten Gesamtlaufzeit des Pipelineschrittes, weswegen sie ebenfalls nicht extra dargestellt wird. Es ist auch hier, wie erwartet wurde, eine etwa kubische Skalierung der Laufzeit zu verzeichnen, wobei hier die Zuwachsfaktoren sogar leicht über dem theoretischen Wert liegen.

	Zellen pro Achse				
	128	256		512	
Schritte	Zeit (ms)	Zeit (ms)	Zuwachsfaktor	Zeit (ms)	Zuwachsfaktor
Gesamt	878,9	7309	8,32	59388	8,13

Tabelle 5.4 Laufzeit der Niederschlagsberechnung über mehrere Auflösungen und die resultierenden Zuwachsfaktoren.

In Diagramm 5.9 wird die Laufzeit veranschaulicht.

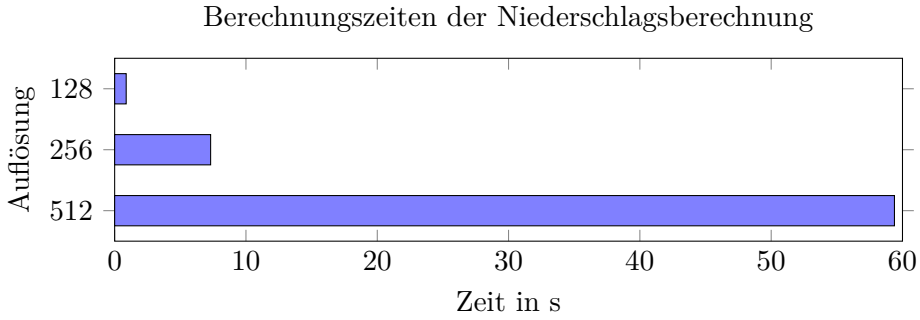


Abbildung 5.9 Diagramm der Tabelle 5.4.

Biomklassifizierung Das Resultat einer Biomklassifizierung anhand der vorher berechneten Daten wird in Abbildung 5.10 dargestellt, jedem Biom ist eine eigene Farbe zugeordnet. Orange entspricht wüstenartigen Regionen, Olivgrau sogenannten kalten Wüsten, Gelb Savannen, Hellgrün Grasland und Dunkelgrün sommergrünen Laubwäldern.

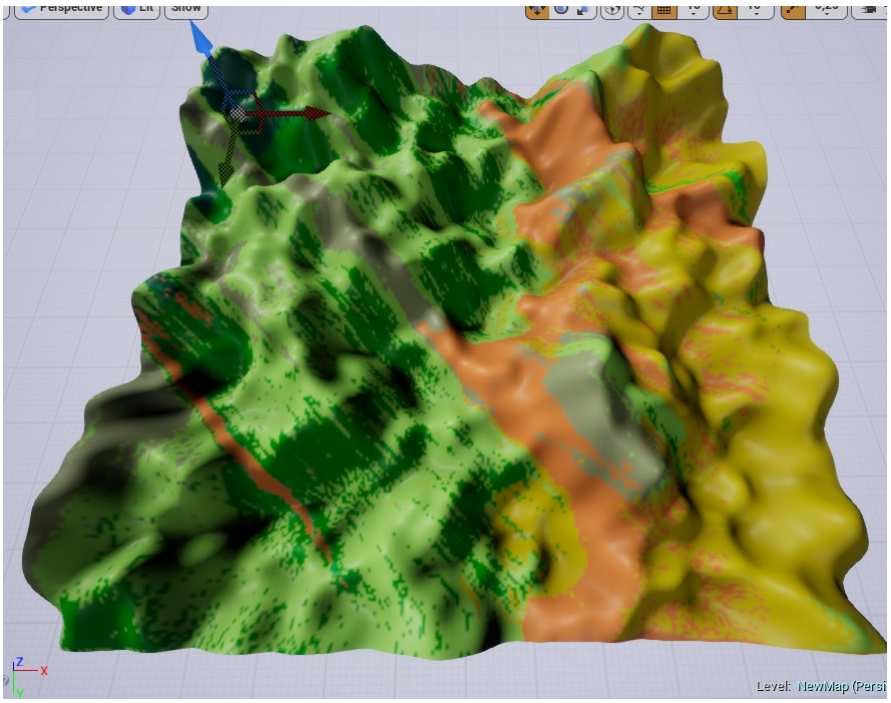


Abbildung 5.10 Die Biomverteilung, dargestellt durch verschiedene Farben je Biom. Gut zu erkennen sind die durch Regenschatten entstehenden Verteilungsmuster.

Die Biomverteilung ist eindeutig von den zuvor berechneten und dargestellten Daten beeinflusst und insgesamt plausibel, wenn man von den zwei länglichen, streifenartigen Bereichen absieht. Diese würde man in der Realität eher weniger erwarten. Man erkennt zudem die Auswirkungen der Regenschatten, auf den feuchten Berghängen bilden sich sommergrüne Laubwälder, während auf den trockeneren Gipfeln und Rückseiten kalte Wüsten oder Grasland vorherrschen.

Nachfolgend wird in Tabelle 5.5 das Laufzeitverhalten der Biomklassifizierung abgebildet. Die Gesamtlaufzeit setzt sich aus dem Einlesen der Lookup Table, der eigentlichen Klassifizierung und sonstigen, zu vernachlässigenden Arbeiten, wie der Ausgabe von Warnungen, zusammen. Zu Testzwecken wurde darauf geachtet, die Lookup Table jedes Mal neu einzulesen. Es ist deutlich ersichtlich, dass fast die gesamte Zeit für das Einlesen der Lookup Table verwendet wird und die Klassifizierung selbst so gut wie keine Zeit kostet. Das Skalierungsverhalten ist daher interessant: Wie erwartet wächst das Klassifizieren quadratisch, das Einlesen ist hingegen konstant, wobei hier systembedingt einige Schwankungen vorherrschen, woraus folgt, dass die Gesamtskalierung ebenfalls nahezu konstant ist.

	Zellen pro Achse				
	128	256		512	
Schritte	Zeit (ms)	Zeit (ms)	Zuwachsfaktor	Zeit (ms)	Zuwachsfaktor
Gesamt	766,6	740,9	0,97	812,9	1,10
CSV	763,3	730,1	0,96	769,2	1,05
Klassifizierung	0,154	0,636	4,13	2,728	4,29

Tabelle 5.5 Laufzeit der Biomklassifizierung, inklusive des Einlesens der Lookup Table, über mehrere Auflösungen und die resultierenden Zuwachsfaktoren.

Das Diagramm 5.5 illustriert die Daten der obenstehenden Tabelle.

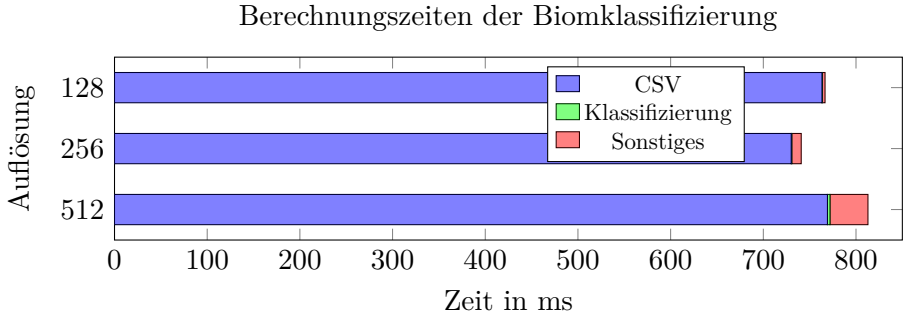


Abbildung 5.11 Diagramm der Tabelle 5.5.

Finalisieren des Terrains Nachfolgend werden die Ergebnisse der Terrainfinalisierung besprochen. In Abbildung 5.12 ist im linken Bild das finalisierte Terrain durch das Proxy Mesh dargestellt. Die rechten Bilder zeigen die endgültige Biomkarte, welche im Laufe dieses Pipelineschrittes entstanden ist, das grobe Terrain als Heightmap und schließlich das aus diesen und mithilfe von DEMs generierte Terrain, ebenfalls als Heightmap. Für die DEMs wurden Daten von *terrain.party*[4] verwendet, was eine Kombination von Satellitendaten von *ASTER*[5], *USGS NED*[6] und *SRTM30+*[7] mit Genauigkeiten von meist ∼30 m und teils von ∼10 m bietet. Die von der Quelle vorgenommene Höhenanpassung wird durch das implementierte System wieder rückgängig gemacht, so dass der Minimalwert tatsächlich einem Grauwert von 0 entspricht.

[4] `https://terrain.party/` (Abgerufen am 19.9.2018)
[5] `https://asterweb.jpl.nasa.gov/gdem.asp` (Abgerufen am 10.01.2019)
[6] `https://www.usgs.gov/core-science-systems/national-geospatial-program/national-map` (Abgerufen am 10.01.2019)
[7] `https://topex.ucsd.edu/WWW_html/srtm30_plus.html` (Abgerufen am 10.01.2019)

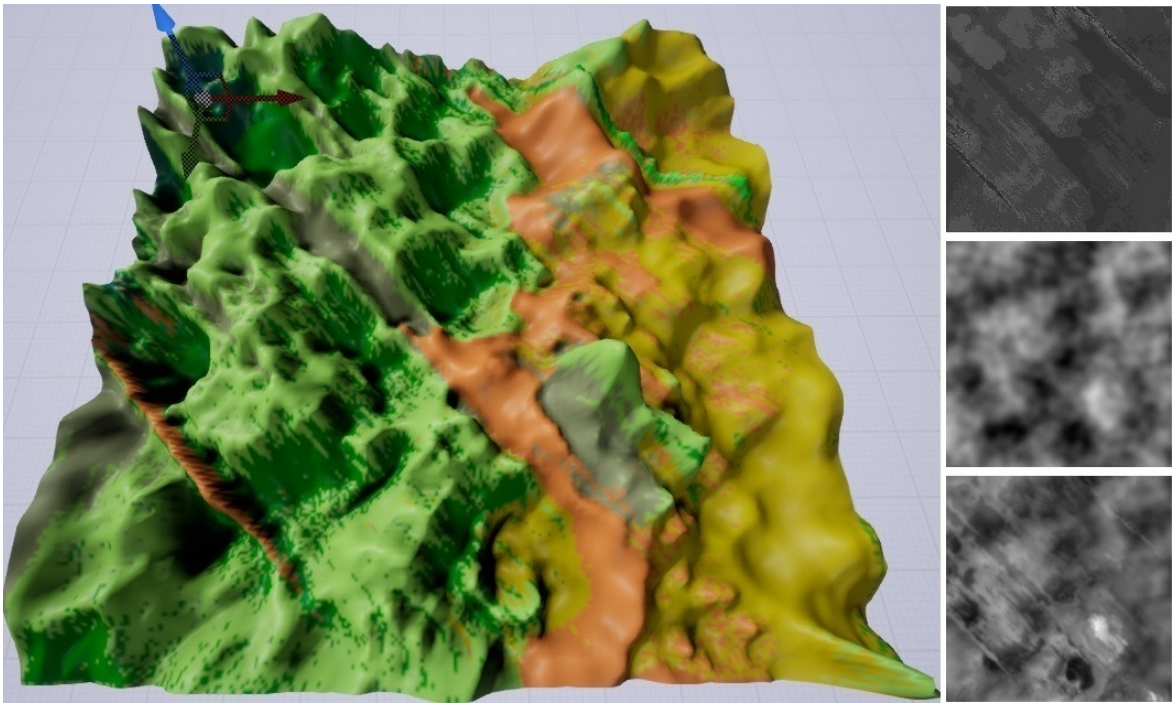


Abbildung 5.12 Links: Das Terrain nach der Finalisierung durch DEMs. Rechts von oben nach unten: Eine Karte der endgültigen Biomverteilung (ein Grauwert je Biomtyp), die Heightmap des groben Terrains, die Heightmap des finalisierten Terrains.

Im Vergleich zu dem vorherigen Terrain, siehe Abbildung 5.10, fällt der höhere Detailgrad des Terrains auf. Auch gibt es nun verschiedene Terrainarten und -formationen, es wirkt weniger regelmäßig und natürlicher. Allerdings scheinen die bereits bekannten zwei Problemstellen auch hier durch abrupte, unterbrechende Höhenformationen, die sich aus der Biomverteilung ergeben, unnatürliche Ergebnisse zu erzielen. Wobei man sagen muss, dass es das Terrain als Ganzes auch interessanter macht. Letztlich ist das konkrete Ergebnis von vielen Parametern und Faktoren der vorherigen Schritte, der DEM-Gewichtung und nicht zuletzt von den gewählten DEMs abhängig und sehr variabel. Es können schnell sehr unterschiedliche Resultate erzielt werden, nicht alle sind jedoch realistisch.
Der Teilschritt des Verzerrens der Biomgrenzen und die DEM-eigenen Höhenmuster kommen in diesem Beispiel interessanterweise für das visuelle Endergebnis weniger zum Tragen als die eigentliche Biomverteilung, da die Biomauflösung hier vergleichsweise hoch ist und die zusammenhängenden Biomregionen teilweise recht klein.

Tabelle 5.6 zeugt die Laufzeitdaten des Finalisierens des Terrains über drei Auflösungen mit einem Biomteiler, einer Kernelgröße und Verzerrungsamplitude von jeweils 8. Die Gesamtlaufzeit[8] der Tabelle setzt sich aus dem Einlesen und Verarbeiten der DEMs, dem Verzerren der Biomregionen sowie dem Blenden und Kombinieren der DEMs und des groben Terrains zusammen. Das Einlesen der DEMs dauert bei 11 Biomen und damit DEMs und jeweils einer

[8]Was nicht in der Gesamtlaufzeit betrachtet wird, aber zwingend hinzukommt, ist das erneute Generieren des Proxy Meshes, was bei 128^2 Vertices wieder, wie am Anfang des Abschnitts geschrieben, knapp 4 Sekunden dauert.

Auflösung von 1081 Pixeln etwa 600 ms, die Skalierung stellte sich als konstant heraus. Damit ist diese Aufgabe bei kleineren Auflösungen ein größerer Posten, dessen Anteil sinkt aber mit zunehmender Auflösung merklich. Das Verzerren skaliert dagegen quadratisch, aber die absolut benötigte Zeit ist durchweg vernachlässigbar klein. Der Großteil der Zeit wird für das Blenden und Kombinieren der DEMs verbraucht, was ebenfalls ein quadratisches Wachstum besitzt.
Weitere, nicht explizit erfasste Aufgaben, die in die Gesamtzeit eingerechnet sind, sind das Schreiben von 16 Heightmap Tiles, was in der Region von 250 ms für die Auflösung 2048 und knapp einer Sekunde für 4096 liegen sollte, wenn man die Zeiten am Anfang dieses Abschnitts betrachtet. Dazu kommt das Schreiben von zwei Bildern in Dateien (was ebenfalls wieder eingespart werden könnte) und das Generieren von Noise (vergleichbare Zeiten sind im Anfang des Abschnitts ersichtlich).

	Zellen pro Achse				
	1024	2048		4096	
Schritte	Zeit (ms)	Zeit (ms)	Zuwachsfaktor	Zeit (ms)	Zuwachsfaktor
Gesamt[8]	1615	4933	3,05	17567	3,56
Einlesen	605,9	597,3	0,99	603,3	1,01
Verzerren	1,444	6,179	4,28	25,07	4,06
Kombinieren	839,1	3754	4,47	14674	3,9

Tabelle 5.6 Laufzeit der Finalisierung des Terrains über mehrere Auflösungen und die resultierenden Zuwachsfaktoren.

Diagramm 5.13 veranschaulicht die oben stehenden Daten.

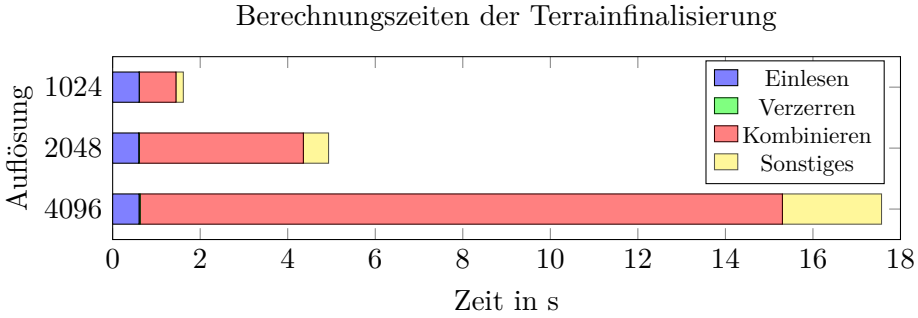


Abbildung 5.13 Diagramm der Tabelle 5.6.

Tabelle 5.7 zeigt die Auswirkungen von anderen Biomauflösungen und Kernelgrößen auf die relevanten Teilschritte. Halbiert man die Biomauflösung, ohne aber Kernelgröße und Amplitude zu verändern, wie in der mittleren Spalte abgebildet, verkürzt das die Zeit des Verzerrens um ca. 40%, alle anderen Teilschritte werden jedoch kaum oder gar nicht beeinflusst, und da die Zeit des Verzerrens ohnehin sehr gering ist, hat die Biomauflösung selbst damit kaum einen Einfluss auf die Performanz. Was jedoch einen enormen Einfluss hat, ist die Kernelgröße. Wird sie verdoppelt, wie in der rechten Spalte zu sehen, steigt die Zeit des Kombinierungs-

schrittes um ca. Faktor 2,87 an, was, wenn man bedenkt, dass dieser Teilschritt sehr teuer ist, schnell kritisch wird.

	4096 Zellen pro Achse				
	Bt 8, Kg 8	Bt 16, Kg 8		Bt 16, Kg 16	
Schritte	Zeit (ms)	Zeit (ms)	Zuwachsfaktor	Zeit (ms)	Zuwachsfaktor
Verzerren	25,07	15,21	0,61	21,70	0,87
Kombinieren	14674	14894	1,01	42107	2,87

Tabelle 5.7 Laufzeit der Finalisierung mit unterschiedlichen Biomauflösungen und Kernelgrößen. Bt steht für den Biomteiler von der Grundauflösung und damit die Biomauflösung und Kg für die Kernelgröße in Zellen der Grundauflösung.

Asset-Platzierung Die Ergebnisse des letzten Schrittes, der Asset-Platzierung, werden in diesem Abschnitt behandelt. Zur Verdeutlichung der Möglichkeiten, vor allem durch die vielen Parameter und die sich daraus emergent ergebenden Verteilungen, ist in Abbildung 5.14 eine Beispielverteilung dargestellt. Zur besseren Illustration wurden die Asset-Instanzen über das Proxy Mesh transliert, wobei sie leicht über dem Mesh „schweben“, was zu ignorieren ist. Als Testbeispiel wurde eine praxisnahe, aber vereinfachte Konstellation gewählt, in der drei Asset-Klassen in einem Biom und viele Constraints in moderatem Einfluss vorkommen. So liegen die drei Asset-Klassen in verschiedenen Größenkategorien, beschatten sich, stoßen sich ab und treten teilweise geclustert auf. Die Kombination der konkreten Assets und dem Biom ist dagegen willkürlich für diesen Test gewählt und stellt keinen Anspruch an eine plausible Konstellation (Nadelbäume in Savanne).

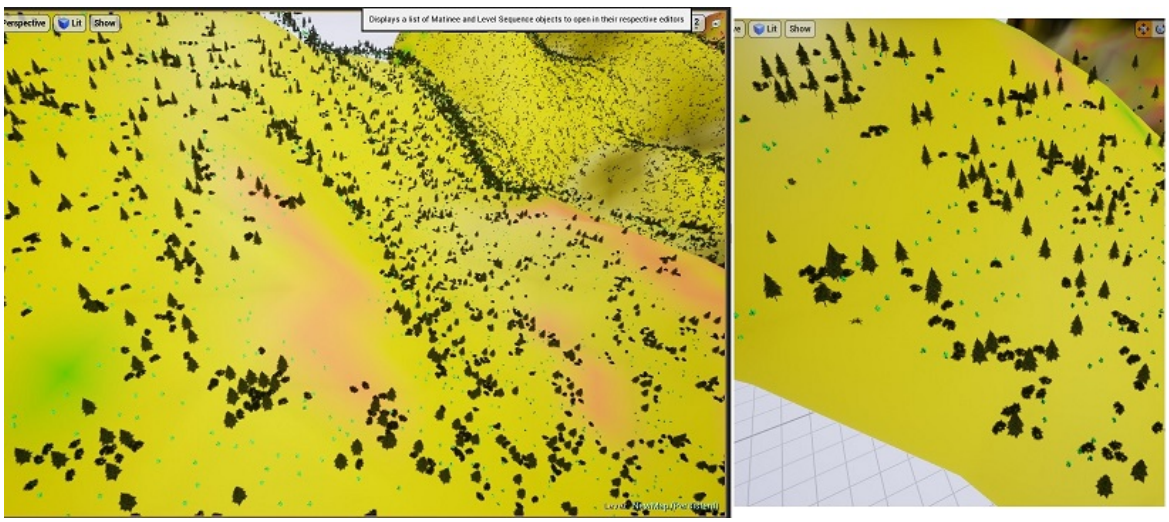


Abbildung 5.14 Zwei Ausschnitte von den beispielhaft platzierten Assets über dem Proxy Mesh, wobei die Translation nicht exakt ist, daher die „schwebenden“ Assets. Man beachte jedoch die gute Verteilungen der Assets untereinander sowie die hohe Anzahl an Instanzen. Die konkreten Asset Meshes dienen nur zu Testzwecken und sind in diesem Biom nicht realistisch.

Eingestellt wurde, dass sich alle Instanzen innerhalb ihrer Asset-Klasse verdrängen, Büsche und Bäume sich dabei auch gegenseitig beeinflussen, Büsche stark und Bäume moderat geclustert auftreten. Zudem verursachen Bäume Beschattung und die kleinen hellgrünen Objekte (etwa Blumen, oder Gräser) benötigen einen vollsonnigen Standort.
Anhand der Bilder wird deutlich, dass nicht nur erfolgreich eine hohe Anzahl an Instanzen generiert und diese der Biomverteilung nach über dem Terrain platziert werden kann, sondern auch, dass die konkrete Verteilung der Assets untereinander ebenfalls dem angestrebten Ziel entspricht. Damit können, realistische Eingaben vorausgesetzt, plausible, visuell ansprechende Verteilungen großflächig erzeugt werden. Wie hier etwa abgebildet das überwiegend zusammenhängend, gruppierte Auftreten von Bäumen und Büschen sowie Pflanzen, die eher dazwischen und in Freiflächen vorkommenden.

Die Messung und vor allem die Analyse des Laufzeitverhaltens gestaltet(e) sich in diesem Schritt schwieriger und komplexer als in den anderen, denn es gibt hier viele ineinander greifende und sich gegenseitig beeinflussende Parameter. Als Testbeispiel diente wieder dieselbe Konstellation, wie weiter oben zur grafischen Veranschaulichung beschrieben.
Tabelle 5.8 zeigt die gemessenen Zeiten über drei unterschiedliche Zielmengen von Positionen pro Biomzelle (sprich Asset-Dichten) hinweg. Die Zahl der Biomzellen betrug stets 15022. Ebenfalls angegeben wird jeweils die Anzahl der dabei tatsächlich gespawnten Asset-Instanzen. Die etwas überraschenden Zielmengen rühren daher, dass sie sich aus der Summe der Asset-Klassen zusammensetzen und nur der Anteil von zweien pro Fall verdoppelt wurde. Als erstes fällt auf, dass über alle Testfälle hinweg eine relativ zum jeweiligen Ziel konstant hohe und nicht abfallende Anzahl an Positionen generiert wurde. Die verwendete Zahl von Versuchen pro Position von 4 war damit ausreichend. Bezüglich der Aufteilung der Gesamtzeiten auf die Teilschritte ist zu sehen, dass die Vorbereitung vernachlässigbar klein und auch der Zuwachsfaktor mit 1,29 gering ist. Interessanterweise ist das Spawnen an sich sehr teuer und macht den Großteil der Zeit aus. Außerdem weicht der Zuwachsfaktor, der theoretisch exakt dem Zuwachs an generierten Instanzen entsprechen sollte, deutlich ab und scheint stark zu schwanken. Das liegt wahrscheinlich daran, dass die Engine-interne Instanzgenerierung ihrerseits anscheinend nicht konstant bezüglich der Zeit ist. Was die Positionsberechnung betrifft, ist diese moderat teuer und daraus, dass der Zuwachsfaktor dauerhaft über dem der generierten Instanzen liegt und seinerseits mit der Zahl derer steigt, lässt sich schließen, dass die Kosten der Berechnung einer Position steigen. Dies entweder direkt durch die höhere Objektdichte, was mehr Vergleiche nötig macht, oder indirekt durch mehr Versuche, die durch vermehrtes Fehlschlagen der Constraints auftreten.

	Asset-Instanzen pro Biomzelle / Zuwachsfaktor				
Ziel:	15	25	1,67	45	1,8
Generiert:	13,9	23,2	1,67	41,3	1,78
Schritte	Zeit (ms)	Zeit (ms)	Zuwachsfaktor	Zeit (ms)	Zuwachsfaktor
Gesamt	7180	12242	1,71	24327	1,99
Vorbereitung	45	58	1,29	75	1,29
Positionsberechnung	1451	2599	1,79	5457	2,1
Spawning	4810	9582	1,99	15131	1,58

Tabelle 5.8 Laufzeit der Berechnung der Asset-Positionen und des Spawnings der Assets über verschiedene Zielmengen hinweg. Die Zielmengen beziehen sich auf Assets pro Biomzelle, von denen es 15022 gab.

Durch Diagramm 5.15 werden die Daten der obenstehenden Tabelle sowie des letzten Testfalls der folgenden Tabelle veranschaulicht.

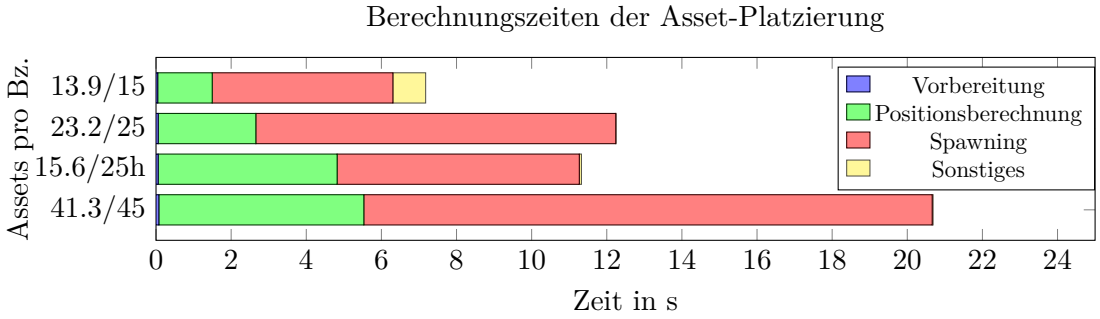


Abbildung 5.15 Diagramm der Tabelle 5.8 und des letzten Testfalls der Tabelle 5.9, markiert mit „h“.

Nachfolgend werden die Auswirkungen anderer entscheidender Parameter, konkret der Biom- und damit Asset-Grid-Auflösung und der Anzahl an Positionierungsversuche untersucht. In Tabelle 5.9 sind daher drei weitere Testfälle dargestellt. Als Referenz diente der mittlere Testfall der obenstehenden Tabelle mit 25 Positionen pro Biom als Zielmenge.
Im ersten Testfall wurde die Auflösung des Biomgitters halbiert und somit die jeweilige Größe vervierfacht. Damit die Objektdichte über die Fläche unverändert bleibt, wurde ausgleichend die Anzahl der Instanzen pro Biomzelle ebenfalls vervierfacht. Die Zahl der so tatsächlich erzeugten Instanzen hat sich sogar etwas mehr als vervierfacht, was darauf hindeutet, dass durch die größeren Biomzellen die Constraints etwas leichter zu erfüllen waren. Die Zeit, die der Vorbereitungsschritt benötigte, sank, was naheliegend ist, da dieser u.a. von der nun gesunkenen Anzahl der Biomzellen abhängig ist. Die Zeit des Spawnings kann hingegen, wenn man die generell hohe Schwankung bedenkt, als etwa konstant angenommen werden, obwohl sie hier, auf die Zahl der Instanzen bezogen, etwas stieg (1,09 fache Zeit bei insgesamt $\frac{96}{4} \cdot 23,2 = 1,03 = 3\%$ mehr Asset-Instanzen). Der interessanteste Schritt schließlich, das Berechnen der Positionen, dauerte deutlich länger, was darauf zurückzuführen ist, dass nun die Zellengröße größer ist und damit mehr Assets bei der Positionierung abgeprüft werden müssen als zuvor.

	Biomaufl. halbiert		Versuche verdoppelt		V. ·2 + härtere Constr.	
	Asset-Instanzen pro Biomzelle / Zuwachsfaktor					
Ziel:	100	4,0	25	1,0	25	1,0
Generiert:	96	4,14	23,3	1,0	15,6	0,67
Schritte	Zeit (ms)	Zwfkt.	Zeit (ms)	Zwfkt.	Zeit (ms)	Zwfkt.
Gesamt	14502	1,18	12386	1,01	11293	0,92
Vorbereitung	42	0,72	56	0,97	57	0,97
Einzelpositionen	4054	1,56	2645	1,02	4764	1,83
Spawning	10416	1,09	9672	1,01	6448	0,67

Tabelle 5.9 Laufzeit der Platzierung der Assets bezüglich verschiedener Parameter. Als Referenzwerte für die Zuwachsfaktoren dienten die des mittleren Versuchs in Tabelle 5.8.

Der zweite Testfall zeigt den Einfluss einer verdoppelten Anzahl an Positionierungsversuchen. Da alle Ergebnisse in etwa unverändert sind, muss man zu dem Schluss kommen, dass, wie zuvor vermutet, die vorherige Anzahl an Versuchen ausreichend war und die Erhöhung somit keinen nennenswerten Einfluss hatte. Im dritten Testfall wurden entsprechend die Constraints verschärft, so dass es schwieriger und damit unwahrscheinlicher wurde, eine valide Position zu samplen. Die Auswirkungen sind deutlich. Trotz der erhöhten Anzahl an Versuchen werden nun deutlich weniger Positionen erfolgreich generiert ($\frac{15,6}{25} = 0,624 = 62,4\%$ der Zielanzahl, was 33% weniger sind als zuvor). Die benötigte Gesamtlaufzeit sinkt jedoch nur um 8%, denn obwohl Vorbereitungs- und besonders die Spawningzeit wie erwartet sanken, stieg die Dauer der Positionsberechnung sogar stark an. Damit steigt auch ihr Anteil an der Gesamtzeit erheblich. Die offensichtliche Ursache für den Zeitanstieg ist, dass nun mehr Versuche benötigt werden, um valide Positionen zu generieren. Tatsächlich scheinen die 8 im Mittel noch nicht auszureichen und es wären noch mehr nötig.

Insgesamt wurde deutlich, dass das Spawning einen erstaunlich hohen Anteil an der Gesamtdauer und eine ebenfalls hohe absolute Berechnungsdauer hat. Ebenfalls hat sich gezeigt, dass die Zahl der Versuche pro Positionierungsversuch und (vor allem in Kombination) die durch die Constraints bestimmte Wahrscheinlichkeit einer validen Positionsfindung einen entscheidenden Einfluss auf die benötigte Zeit haben.

5.3 Gesamtevaluation

Nach der Einzelanalyse der Pipelineschritte im vorherigen Abschnitt soll in diesem nun das System als Gesamtes evaluiert und reflektiert werden.

Wie ein generiertes Terrain schlussendlich aussieht, bzw. aussehen kann, ist in Abbildung 5.16 dargestellt. Sie zeigt das zuvor erstellte Terrain als *Unreal Engine Tiled Landscape*, skaliert auf drei Kilometer Seitenlänge, auf Basis von den erzeugten 16 Tile Heightmaps.

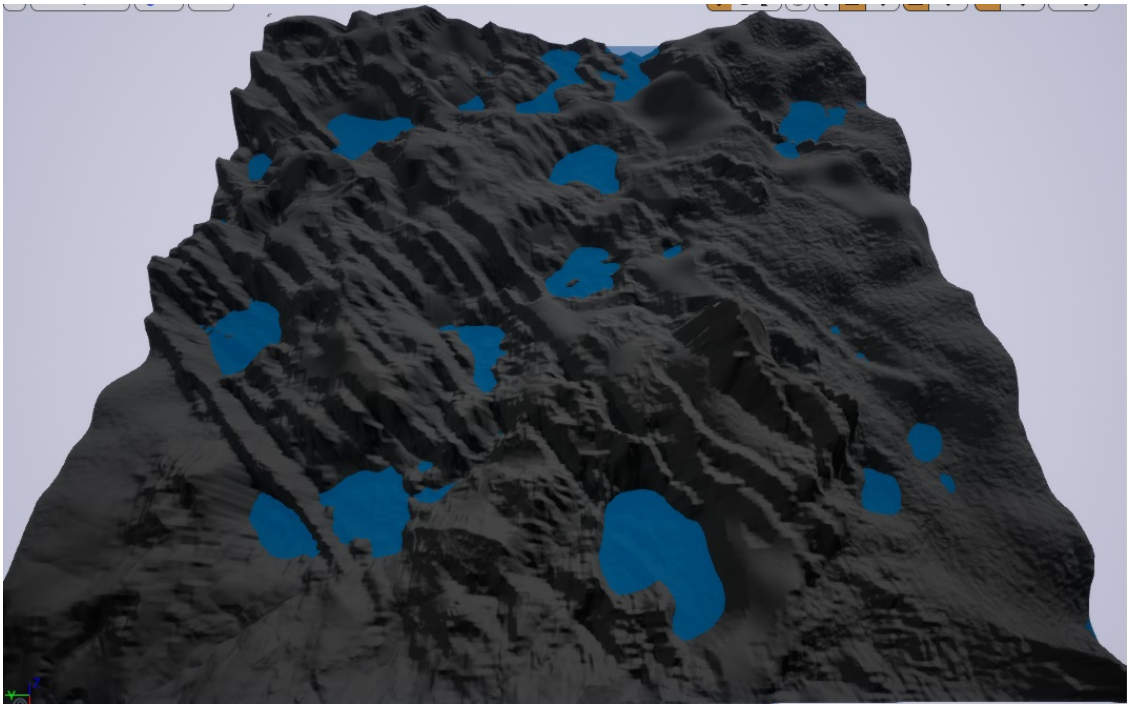


Abbildung 5.16 Das finalisierte Terrain aus dem Datensatz mit 1000 Pixeln pro Achse als *Tiled Landscape* mit 4×4 Tiles und dargestellt mit ca. drei Kilometern Seitenlänge. Gewässer visualisiert durch eine Ebene.

Was positiv auffällt sind die unterschiedlichen Terrainformen und „Oberflächenmuster“, die sich durch die Nutzung der DEMs für die verschiedenen Biome ergeben. So wirkt die Landschaft sehr organisch und interessant. Auch ist die durch die Klimasimulation berechnete Biomverteilung sehr natürlich und im Großen und Ganzen plausibel, was in Kombination mit dem Grundterrain für ein gutes Zusammenspiel der Biome über der Gesamtlandschaft sorgt. Ebenfalls wurde festgestellt, dass durch die Pipelinearchitektur, die vielen Einstellungsmöglichkeiten innerhalb der Pipelineschritte sowie die direkte Visualisierung der Ergebnisse das finale Terrain vergleichsweise schnell und einfach angepasst werden kann. Dies erfüllt die erhofften Erwartungen. Die Güte der Ergebnisse ist dabei natürlich immer abhängig von den gewählten Parametern und auch den genutzten DEMs. An einigen Beispielen wurde ersichtlich, dass eine ungünstige Kombination schnell zu unschönen oder unrealistischen Bereichen und Teilergebnissen führen kann, welche sich dann durch den sequentiell Pipelineprozess bis zum Endergebnis durchziehen.

Die Positionsberechnung der Assets fügt sich gut in das Gesamtsystem ein, wie Abbildung 5.17 zeigt. Der Anwender hat viele Möglichkeiten, das Platzierungsverhalten der Assets zu beeinflussen und die Endergebnisse sind vielversprechend, optisch ansprechend und vor allem entsprechen sie den Erwartungen gemäß der eingestellten Parameter.

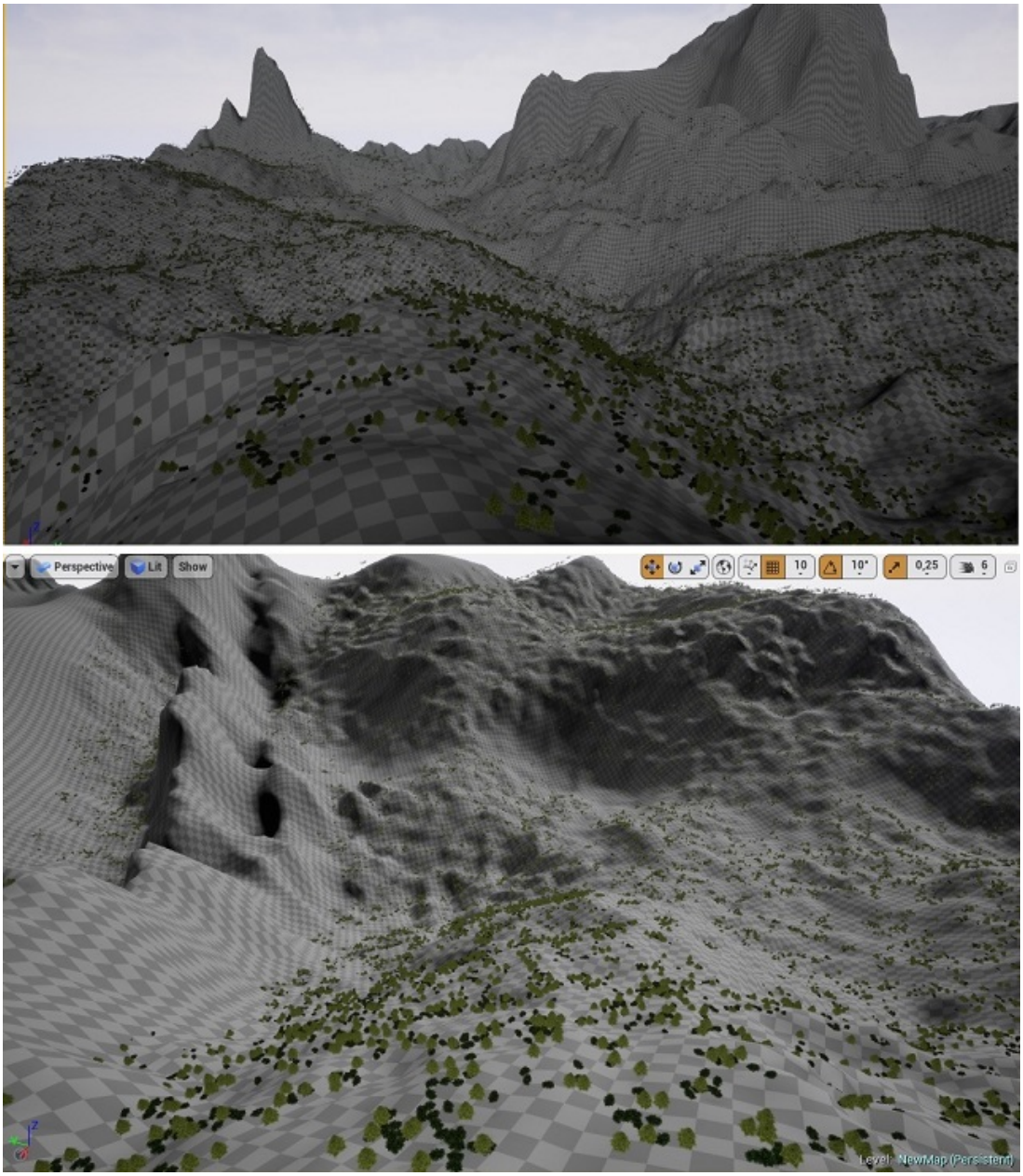


Abbildung 5.17 Zwei Ausschnitte des finalisierten Terrains (gleiche Bedingungen wie im Bild 5.16) mit in einem Biom platzierten Assets nach Verteilung, wie in Abschnitt 5.2 „Asset-Platzierung“ beschrieben.

Ein Problem, was jedoch besteht, ist, dass die Ausrichtung und Positionierung der Assets als Gesamtes nicht exakt mit dem letztlich importierten Mesh übereinstimmt. Dies ist zum einen ein Skalierungsproblem und zum anderen liegt es an der Triangulation, die bei der vorherigen Höhenberechnung nicht zur Verfügung steht. Zu beheben wäre dieses Problem, wenn bei der Platzierung der Assets direkt alle nachfolgend von der Engine ausgeführten Operationen, um das finale Mesh zu erstellen, bekannt wären und bereits berücksichtigt werden würden. Alternativ könnten die Assets erst nachträglich unter Verwendung des Meshes und evtl. Raycasting platziert werden.

Einen Überblick über die Laufzeiten des Systems zeigt Diagramm 5.18, wo die Gesamtlaufzeiten der Pipelineschritte, wie sie im vorherigen Abschnitt einzeln besprochen wurden, in jeweils drei Auflösungsstufen aufgeführt sind. Im Direktvergleich wird verdeutlicht, wie unterschiedlich zum einen die Laufzeiten sind und zum anderen auch ihr Skalierungsverhalten ist. Besonders lang dauert und mit hohem, weil kubischem, Wachstum ist die Niederschlagsberechnung. Ebenfalls sehr aufwendig ist die Terrainfinalisierung und in hoher Auflösung auch die Windberechnung durch ihr ebenfalls kubisches Wachstum. Dagegen sind Temperaturberechnung und Biomklassifizierung (letzteres durch das konstante Wachstum, vor allem relativ gesehen, in höheren Auflösungen) sehr schnell und günstig.

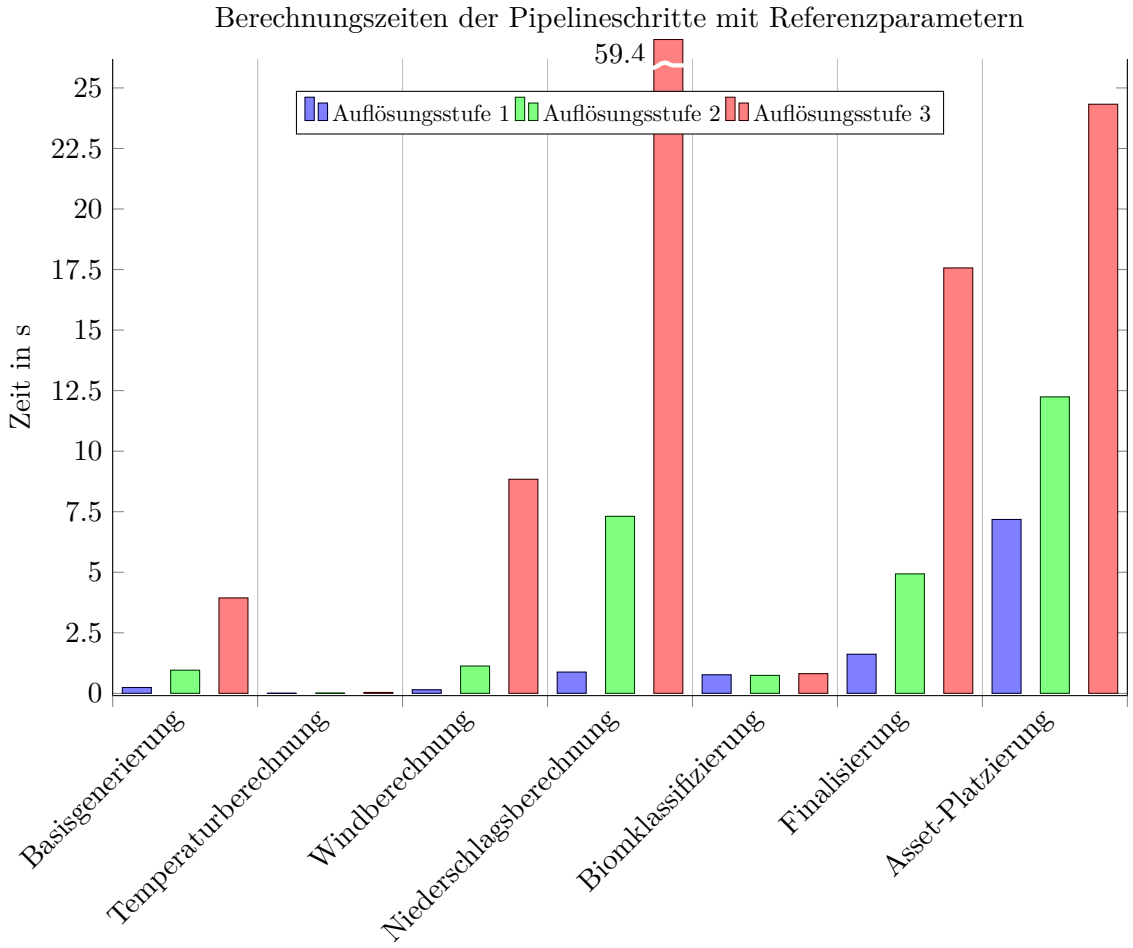


Abbildung 5.18 Vergleich der Gesamtberechnungszeiten der Algorithmusschritte in drei Auflösungsstufen. Für die Basisgenerierung und Finalisierung sind dies 1024, 2048 und 4096. Im Fall der Asset-Platzierung entsprechen sie einer Zielmenge von 15, 25 und 45 Assets pro Biomzelle. Für alle anderen Schritte gelten 128, 256 und 512. Jeweils in den Referenzeinstellungen, wie im vorherigen Kapitel aufgeführt, Werte siehe Tabellen 5.1, 5.2, 5.3 , 5.4, 5.5, 5.6 und 5.8 . Zu beachten gilt, dass der Wert der Niederschlagsberechnung in der 3. Stufe knapp 60 Sekunden beträgt und damit der Balken mehr als doppelt so hoch wäre, wie hier abgebildet.

Damit bestätigen die Daten die Erwartungen, die sich nach der Komplexitätsanalyse eingestellt hatten. Anhand der Verarbeitung von Daten in einem zweidimensionalen Gitter herrscht vor allem quadratisches Wachstum vor, bei den Schritten mit iterativer Berechnung entsprechend kubisches, was zu langen Ausführungszeiten mit steigender Auflösung führt. Etwas unerwartet und ärgerlich sind die hohen Berechnungskosten für das Erstellen des Proxy Meshes. Jedoch ist anzumerken, dass die jeweiligen Zeiten teils stark von unterschiedlichen Parametern abhängig sind und somit viel Spielraum gegeben ist. Zudem ist zu bedenken, dass es ebenfalls noch viel Potenzial zur Optimierung gibt, sowohl was Speicher als auch vor allem die Laufzeiten betrifft. Für die Platzierung und Generierung der Assets gelten im Prinzip dieselben Punkte der Parametrisierung und Optimierung. Allerdings kommt hier auch der erstaunlich zeitfressende Punkt des Spawnens hinzu, der sich nicht verhindern lässt und direkt von der Zahl der Instanzen abhängt.
Insgesamt kann man aber sagen, dass, je nach gewählten Parametern, alle Pipelineschritte ausreichend schnell sind und in einem zu erwartenden Rahmen liegen, speziell für diese nicht durchoptimierte prototypische Implementierung.

Durch die Nutzung eines schnelleren, aktuelleren Prozessors (Intel i7-7800X, 6 Kerne, 3.5/4 GHz, AVX-512) konnten, je nach Pipelineschritt, Steigerungen der Berechnungsgeschwindigkeit zwischen Faktor 1,5 (Windberechnung) und 2,3 (Assetplatzierung) erzielt werden. Die meisten Schritte konnten ca. um Faktor 2 beschleunigt werden. Bei der Biomklassifizierung konnten jedoch keine Leistungsgewinne verzeichnet werden, da diese Input-Output-limitiert ist.

Theoretische Limitierungen des Systems sind im Allgemeinen die Laufzeit, die der Nutzer bereit ist zu warten, die Menge an verbautem Arbeitsspeicher und schließlich der Wertebereich der verwendeten Datentypen. Je nach Pipelineschritt sind Laufzeit oder Speicherplatz die gravierendsten Limits, die Datentypen kommen erst deutlich später, in der Praxis meist gar nicht zum Tragen. Es bräuchte z.B. mehr Asset-Instanzen oder eine größere Seitenlänge der Gitter, als mit einem *int* repräsentierbar ist. Selbiges gilt für die darzustellende Größe des Terrains und den Zahlenraum eines *floats*.
Empirisch festgestellt wurde, dass im Schritt der Asset-Positionierung sowohl Laufzeit als auch Speicherverbrauch in etwa linear mit der Anzahl an Instanzen skalieren. Die Verwendung von vielen Instanzen macht diesen Schritt, unabhängig des Positionierungsalgorithmus, zu einem der teureren bzgl. des Speicherverbrauchs. Beispielhaft wurde mit ca. 350 Tausend Instanzen ein Gesamtspeicherverbrauch der Anwendung von etwa 1,7 GB festgestellt, bei ca. 1,1 Millionen Instanzen waren es bereits 4 GB. Die Rendering-Performanz hängt verständlicherweise dabei maßgeblich vom verwendeten LOD und der Zahl an gleichzeitig sichtbaren Instanzen ab, stellt also kein grundsätzliches Problem dar.
Die meisten der anderen Pipelineschritte brauchen nur temporär eine in der Regel vernachlässigbar kleine Menge an zusätzlichem Speicherplatz. Bei diesen Schritten spielt eher die steigende Laufzeit eine größere Rolle.
Der Schritt der Basisgenerierung, der zudem diverse Verwaltungs- und Visualisierungsauf-

gaben übernimmt, kostet dagegen ebenfalls viel Speicher, die konkrete Menge wird von der Berechnungsauflösung beeinflusst. Bei einer Auflösung von 2048 betrug der Gesamtspeicherverbrauch ca. 850 MB, bei 8192 waren es etwa 4,7 GB und bei einer weiteren Verdopplung reichte der verfügbare Speicher (ca. 6,3 GB) nicht mehr aus und es resultierte ein Absturz. Durch Tests konnte ermittelt werden, dass der Großteil des dauerhaft benötigten Speichers für die Overlay-Texturen verbraucht wird. Dies ist sehr ärgerlich, da diese nur peripher relevant sind, jedoch ließe sich der Verbrauch sicher noch stark optimieren. Ohne die Texturen hingegen konnte eine Berechnung des Schrittes mit der Auflösung von 16384 erfolgreich mit nur etwa 4,7 GB Speicherverbrauch durchgeführt werden. Weiterhin beträgt die maximale Texturauflösung 8192 Pixel, bei größeren Berechnungsauflösungen müssten diese von der Texturgröße entkoppelt und letztere limitiert werden. Alternativ könnten mehrere kleine Texturen verwendet oder eine Komprimierung vorgenommen werden.

Nach der soeben dargelegten Evaluation des Gesamtsystems wird im nächsten Abschnitt versucht, ein Vergleich mit bestehenden Anwendungen zu ziehen.

Vergleiche

Ein Vergleich mit anderen, bereits existierenden Lösungen gestaltet sich schwierig, da es einerseits kaum vergleichbare Systeme gibt, die sich in ihren Ergebnissen und Anforderungen ähneln, und andererseits die Anwendungen selten öffentlich zugänglich sind. Hinzu kommt, dass die visuelle Bewertung überwiegend subjektiver Natur ist und es keine konkreten Metriken zur Beurteilung der Güte gibt.

Wissenschaftliche Veröffentlichungen mit vergleichbarem Inhalt und Ziel sind mir nicht bekannt. Daher wird im Folgenden versucht, einige Teilbereiche gesondert zu vergleichen sowie auch kommerzielle Tools und private Projekte betrachtet.
Bezüglich dem prozeduralen Platzieren von Assets, speziell der Vegetation, kann man (Lane et al., 2002) und (Deussen et al., 1998) als Vergleich heranziehen. Beide verfolgen, wie auch diese Anwendung, u.a. einen Local-to-Global-Ansatz, sind aber insoweit komplexer, dass sie eine zeitliche Simulation durchführen und somit Effekte wie Selbstausdünnung ermöglichen. Ihre Resultate sind in etwa visuell vergleichbar zu dieser Arbeit und ebenfalls vielversprechend. Performanzvergleiche sind u.a. aufgrund der deutlich unterschiedlichen Systeme nicht angebracht. Weiterhin kann man sagen, dass rein auf Poisson Disk Sampling oder auf zufällig generierten Verteilungen beruhende Systeme nicht die Flexibilität und Plausibilität bieten, wie sie hier in der Arbeit möglich sind.
Im Laufe der Arbeit wurden einige Paper zur PTG vorgestellt, die verschiedene Techniken nutzen, und bereits erläutert, dass sie verschiedene Stärken und Schwächen besitzen. Eine reine Terraingenerierung durch Noise ist wenig realistisch und nicht intuitiv benutzbar. Die Kombination mit Erosionsalgorithmen jedoch ist deutlich vielversprechender, z.B. (Jákó, 2011) hat gezeigt, dass so realistische, erodierte Landschaften generiert werden können. Nachteil jedoch ist, dass diese Variante alleine relativ rechenintensiv ist und zudem insofern un-

flexibel, dass andere Prozesse und Landschaftsformen nicht hinreichend nachgebildet werden können. Daher war in dem Konzept dieser Arbeit Erosion als eine Teilkomponente des Gesamtsystems vorgesehen und eine zukünftige Implementierung von Vorteil. Beeindruckende Ergebnisse lieferten auch (Zhou et al., 2007) mit der DEM-basierten Terrainsynthese. Durch die Verwendung der realen Daten werden sehr realistische Ergebnisse erzielt. Diese Methode ist, wie alle Beispiel-basierten, stark abhängig von den gegebenen DEM-Daten und ihrer Komplexität. Dennoch wäre es für die Zukunft sinnvoll zu versuchen, einige Teilschritte- und Techniken in dieser Arbeit, speziell dem noch eher rudimentären Schritt der Terrainverfeinerung durch DEMs, zu integrieren, wie in Kapitel 3 diskutiert. Dies würde allerdings auch deutlich mehr Berechnungszeit nach sich ziehen.
Einige private Projekte, die man evtl. auf Web-Blogs oder *YouTube* findet, oder auch das eine oder andere Spiel wie *Minecraft* beinhalten das prozedurale Generieren von 2D/3D-Landschaften aus unterschiedlichen Biomen. Diese basieren jedoch meist rein auf synthetischen Methoden, z.B. Temperatur und Feuchtigkeit durch Noise, und bieten damit nicht so realistische Ergebnisse bei hoher Benutzbarkeit wie diese Arbeit hier. Dafür sind sie voraussichtlich performanter. Einige wenige andere Projekte basieren, wie diese Arbeit, ebenfalls auf grundlegenden, physikalischen Prozessen zur Biomverteilung, sind dann aber dabei oft nicht so flexibel und generieren nur die Biomverteilung, kein 3D-Terrain.
Kommerzielle Grafik- und Terrain-Tools wie *Houdini* sind in der Lage fotorealistische Landschaften zu generieren. Sie basieren dabei in der Regel auf einem semi-automatischen Arbeitsablauf. Im Gegensatz zu dieser Arbeit wird jedoch mehr manuelle Arbeit, Interaktion und Design vom Nutzer verlangt und kombinierend Gebrauch von prozeduralen Algorithmen gemacht, die in Form von z.B. Erosionsfiltern angewendet werden. Eine automatische, simulierte Biomverteilung und -generierung bietet meines Wissens nach aber keins dieser Tools.

Generell lässt sich damit feststellen, dass das in dieser Arbeit entwickelte System in seiner Gesamtheit vergleichsweise einzigartig ist und viele gute Techniken, Ansätze und Komponenten vereint.

Nachdem in diesem Kapitel das entwickelte System ausgedehnt evaluiert wurde, wird im nächsten und letzten Kapitel das Fazit gezogen und noch einmal auf Verbesserungsmöglichkeiten hingewiesen.

Kapitel 6

Fazit und Ausblick

Ziel dieser Arbeit war es, ein prozedurales System zu schaffen, sowohl konzeptionell als auch prototypisch, was es ermöglicht, semi-automatisch sehr große, realistisch wirkende Landschaften zu erzeugen, welche aus mehreren plausibel ineinander übergehenden Biomen besteht und leicht mit passenden Objekten befüllt werden kann. Dabei sollte nicht nur der Realismus, sondern auch Benutzbarkeit, Flexibilität und auch die Performanz im Vordergrund stehen und ein guter Kompromiss aus diesen vier Anforderungen erzielt werden.
Dies ist voll umfänglich gelungen, wie die Auswertung der Ergebnisse im vorherigen Kapitel zeigt. Es wurden im Zuge der Arbeit viele mögliche Methoden, ihre Vor- und Nachteile sowie Kombinationsmöglichkeiten analysiert. Das daraus entstandene Pipelinemodell, bestehend aus initialer Terraingenerierung, mehrschrittiger Klimasimulation, Beispiel-basierter Terrainverfeinerung und Objektplatzierung, und die zugehörige Implementierung haben sich als sehr vielversprechend herausgestellt. Mithilfe der physikalisch-basierten Klimasimulation werden anhand von Temperatur, Wind und Niederschlag plausible Biomverteilungen generiert und durch DEMs biomspezifisch das anfangs erstellte Basisterrain finalisiert. So entstehen realistisch wirkende Terrains, bestehend aus unterschiedlichen Landschaftsformen mit bis zu mehreren Hundert Quadratkilometern Größe. Dabei kommt der Stiltransfer mittels Neuronaler Netze nicht wie erhofft zur Anwendung, da sich gezeigt hat, dass dieser hier nicht ohne Weiteres nutzbar ist. Auch die Constraint-basierte Platzierung von vielen Hunderttausend Assets mit unterschiedlichen Verteilungen über die Biome hinweg ist durch das verwendete Local-to-Global-Modell leicht möglich und es ergeben sich gute Resultate. Das entwickelte System hat gezeigt, dass es möglich ist, durch Kombination der unterschiedlichen prozeduralen Ansätze, synthetisch, physikalisch- und Beispiel-basiert, die Vorteile zu vereinen und gute Leistungen in allen Anforderungskategorien zu erzielen. Das System ist durch die dem Anwender innerhalb der jeweiligen Pipelineschritte zur Verfügung stehenden Parameter sehr flexibel und durch wenige Veränderungen dieser können schnell unterschiedliche Landschaften produziert werden. Diese kommen dabei meist der Realität recht nahe und sind im Allgemeinen plausibel. Naheliegenderweise führt bei der Vielzahl von Parametern und der so gebotenen Flexibilität nicht jede Kombination zu einem guten Resultat, dies wäre nur mit drastischen

Einschränkungen zu garantieren. Weiterhin konnte festgestellt werden, dass durch das sequentielle Pipelinemodell, die direkte Visualisierung und die gute Nachvollziehbarkeit eine hohe Benutzbarkeit gewährleistet wird. Die Auswirkungen von Änderungen sind gut verständlich und schnell ersichtlich. Vor allem durch Nutzung von Heightmaps ist das System sehr kompatibel und die Resultate auch in externen Anwendungen verwendbar. Eine nachträgliche, manuelle Bearbeitung ist damit gewährleistet. Allerdings können somit auch keine Überhänge oder Höhlen erzeugt werden. Letztlich hat die Auswertung gezeigt, dass auch die Anforderungen an die Performanz erfüllt werden, alle Pipelineschritte sind für sich genommen ausreichend schnell, wobei dies selbstverständlich zum Teil von den Einstellungen abhängt, speziell, da manche Pipelineschritte ein kubisches Laufzeitverhalten aufweisen. Üblicherweise dauert ein Schritt jedoch nur wenige Sekunden bis maximal eine Minute.

In dieser Arbeit steckt jedoch noch viel unausgeschöpftes Potenzial, da durch den großen Umfang und die begrenzte Zeit nicht alle Techniken ausreichend evaluiert und implementiert werden konnten. So bestehen über alle Abstraktionsebenen hinweg, vom Konzept bis hin zu der konkreten Implementierung, noch viele Verbesserungs- und Erweiterungsmöglichkeiten. Die hier vorgestellte Arbeit dient in der jetzigen Form mehr als Grundlagenforschung und veranschaulicht die Möglichkeiten.
Die prototypische Implementierung ist als solche nicht oder nur sehr rudimentär optimiert. An vielen Stellen können noch Laufzeit- und Speicherverbrauchsverbesserungen erzielt werden. Besonders eine Parallelisierung zur Nutzung mehrerer Threads oder gar eine teilweise Ausführung auf der Grafikkarte würden deutliche Gewinne bringen, zumal große Teile des Systems gut parallelisierbar sind. Die Visualisierung verbraucht zudem, wie in Kapitel 5 beschrieben, unnötig viel Rechenzeit und vor allem Speicherplatz und sollte daher optimiert werden. Auch bleibt leider das Problem der nicht exakt platzierten, bzw. nicht korrekt an dem finalen, importierten Mesh ausgerichteten Assets zu beheben. Algorithmisch könnten an fast allen Teilschritten Verbesserungen vorgenommen oder Alternativen exploriert werden, wie bereits ausführlich in den entsprechenden Abschnitten (vor allem Kapitel 4, aber auch 3 und 5) erläutert wurde. Um nur einige bedeutsamere zu nennen, wäre da zuerst die Windberechnung. Dessen Richtungsermittlung ließe sich verbessern, mehrere Höhenschichten einführen oder man könnte die Windberechnung durch eine komplette Fluidsimulation ersetzen. Auch ist die Kombination von Basis- und biomspezifischem Terrain ausbaufähig, hier kam bedauerlicherweise der vorerst nicht zufriedenstellend funktionierende Styletransfer mit den Neuronalen Netzen zum Tragen. In Zukunft können in diesem Bereich hoffentlich bessere Resultate erzielt werden. Alternativ könnten die leider aus Zeitgründen nicht mehr implementierten, clevereren Algorithmen der Textursynthese, wie Hierarchical und Poisson Blending sowie Graph Cut, Abhilfe schaffen. Auf der konzeptionellen Ebene könnte die angesprochene, mögliche Meta-Iteration zwischen Biom- und Terraingenerierung noch realistischere Gesamtergebnisse erzielen, wobei dies mit dem jetzigen System die Berechnungskosten stark in die Höhe treiben würde. Als zusätzlichen Schritt könnte man über die Generierung einer Karte der Bodenbeschaffenheit nachdenken, bei dem verschiedene Gesteinsarten und -härten Ein-

fluss auf die generierte Landschaft nehmen. Speziell bei der Erosions- und Flussberechnung, die es leider auch nicht mehr in den Prototyp geschafft hat, würde dies einen entscheidenden Vorteil bringen.

Abschließend lässt sich feststellen, dass die Arbeit sehr erfolgreich war und ein gutes und effektives prozedurales System geschaffen wurde, um große Multi-Biom-Landschaften in der *Unreal Engine 4* zu generieren. Der entwickelte Prototyp ist darüber hinaus noch ausbaufähig, denn es bestehen noch einige vielversprechende Erweiterung- und Optimierungsmöglichkeiten.

Glossar

Instancing

Eine Technik der Echtzeit-Computergraphik, bei der zur Darstellung vieler Objekte mit gleichem Mesh nur ein gemeinsamer, gebündelter Aufruf an die Grafikkarte (Drawcall) stattfindet. Dies spart sehr viel Leistung. Für jede gerenderte Instanz des Meshes wird eine eigene Transformationsmatrix verwendet, um z.B. die Position zu bestimmen. 43, 48, 61, 78

Level Streaming

Eine Technik zum dynamischen Ein-und Ausblenden von Spielleveln oder Teilen eines Levels. Nur die zurzeit relevanten Teile liegen im Speicher, was Ressourcen spart. 43, 48, 55, 65

Mesh

Die physische Geometrie eines 3D-Objektes. Ein Mesh besteht aus diversen Polygonen, die zusammen die Form des Objektes beschreiben. 3, 78, 81, 96, 101, 102, 108

Proxy Mesh

In der Computergraphik ein Objekt (Mesh), welches stellvertretend für ein anderes steht, meist eine niedriger aufgelöste Version. 56, 67, 78, 81, 85–87, 90, 93, 95, 96, 103

spawn

(to) spawn bezeichnet in der Computergraphik das Erzeugen und Platzieren eines Spielers oder Objektes in einem Spiellevel.. 53, 61, 78, 80, 97–99, 104

Abkürzungen und Akronyme

AI

Artificial Intelligence (deutsch: künstliche Intelligenz). 8, 11

cGAN

conditional Generative Adversial Network. 26

CGI

Computer Generated Imagery. 1

CNN

Convolutional Neural Network. 24–26, 82

DCGAN

Deep Convolutional Generative Adversial Network. 26

DEM

Digital Elevation Model. 1, 10, 26, 47, 50, 52–55, 60, 65, 73, 74, 81, 82, 93–95, 100, 105

FBM

Fractional Brownian Motion. 13

GAN

Generative Adversial Network. 25, 26

LIB

Library (deutsch: Programmbibliothek). 56

LOD

Level of Detail. 43, 48, 104

MRF

Markov Random Field. 15, 18

PCG

Prozedurale Content-Generierung. 5–12, 26, 44

PTG

Prozedurale Terraingenerierung. 5, 8–10, 12, 13, 26, 35, 36, 38, 41, 42, 44, 45, 48, 49, 54, 105

SRGAN

Super Resolution Generative Adversial Network. 26

UML

Unified Modeling Language. 53, 62, 63

Literaturverzeichnis

Abdullah, A. and Agha, Z. (2013). Efficient poisson blending for seamless image stitching. https://pdfs.semanticscholar.org/9366/68c90bc5439fe42cfeb00000d790c984a41e.pdf, Abgerufen am: 08.11.2018.

Albelwi, S. and Mahmood, A. (2017). A framework for designing the architectures of deep convolutional neural networks. *Entropy*, 19(6). https://www.mdpi.com/1099-4300/19/6/242/pdf, Abgerufen am: 08.11.2018.

Amato, A. (2017). Procedural content generation in the game industry. In *Game Dynamics*, pages 15–25. Springer. https://www.springer.com/cda/content/document/cda_downloaddocument/9783319530871-c2.pdf?SGWID=0-0-45-1604224-p180609223, Abgerufen am: 08.11.2018.

Baines, T. (2016). Maths, no man's sky, and the problem with procedural generation. Bild von No Man's Sky aus Online-Artikel auf Thumbsticks;https://www.thumbsticks.com/no-mans-sky-problem-procedural-generation/, Abgerufen am: 08.11.2018.

Barnes, C. and Zhang, F.-L. (2017). A survey of the state-of-the-art in patch-based synthesis. *Computational Visual Media*, 3(1):3–20. http://www.connellybarnes.com/work/publications/2016_patch_survey.pdf, Abgerufen am: 08.11.2018.

Beckham, C. and Pal, C. (2017). A step towards procedural terrain generation with gans. *arXiv preprint arXiv:1707.03383*. https://arxiv.org/pdf/1707.03383.pdf, Abgerufen am: 08.11.2018.

Bell, I. (2015). Elite. Bild von Ian Bell's Elite Homepage; http://www.elitehomepage.org/gallery/scrn/battle.gif, Abgerufen am: 12.01.2019.

Bellic, N. (2013). Minecraft 1.7 what's new (terrain, flowers & ground cover). Bild von Mincecraft in Artikel auf WonderHowTo;https://minecraft.wonderhowto.com/news/minecraft-1-7-whats-new-terrain-flowers-ground-cover-0148941/, Abgerufen am: 08.11.2018.

Burger, W. and Burge, M. (2006). *Digitale Bildverarbeitung: Eine algorithmische Einführung mit Java*. X.media.press. Springer Berlin Heidelberg.

Burt, P. J. and Adelson, E. H. (1983). A multiresolution spline with application to image mosaics. *ACM Transactions on Graphics (TOG)*, 2(4):217–236. http://persci.mit.edu/pub_pdfs/spline83.pdf, Abgerufen am: 08.11.2018.

Campbell, N. A. (1996). *Biology*. Benjamin-Cummings Pub Co, 4rd edition.

Campos, C., Leitão, J. M., Pereira, J. P., Ribas, A., and Coelho, A. F. (2015). Procedural generation of topologic road networks for driving simulation. In *2015 10th Iberian Conference on Information Systems and Technologies (CISTI)*, pages 1–6. IEEE.

Carpenter, E. (2011). *Procedural Generation of Large Scale Gameworlds*. PhD thesis, University of Dublin, Trinity College. https://www.scss.tcd.ie/publications/theses/diss/2011/TCD-SCSS-DISSERTATION-2011-056.pdf, Abgerufen am: 08.11.2018.

Cepero, M. (2013). Water bodies. Blogpost über die Generation der Gewässer in "Voxelfarm"; http://procworld.blogspot.com/2013/10/water-bodies.html, Abgerufen am: 08.11.2018.

Cepero, M. (2016). Geometry is destiny part 2. Blogpost über Welt-Generation und Biome in "Voxelfarm"; http://procworld.blogspot.com/2016/07/geometry-is-destiny-part-2.html, Abgerufen am: 08.11.2018.

Cordonnier, G., Braun, J., Cani, M.-P., Benes, B., Galin, E., Peytavie, A., and Guérin, E. (2016). Large scale terrain generation from tectonic uplift and fluvial erosion. In *Computer Graphics Forum*, volume 35(2), pages 165–175. Wiley Online Library. https://hal.inria.fr/hal-01262376/document, Abgerufen am: 08.11.2018.

Deussen, O., Hanrahan, P., Lintermann, B., Měch, R., Pharr, M., and Prusinkiewicz, P. (1998). Realistic modeling and rendering of plant ecosystems. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 275–286. ACM. https://kops.uni-konstanz.de/bitstream/handle/123456789/6414/Realistic_Modeling_and_Rendering_of_Plant_Ecosystems_1998.pdf, Abgerufen am: 08.11.2018.

Dracott, A. (2017). Procedural landscape generation for game environments. Bild von mit Houdini erstelltem Terrain in Artikel auf 80 Level, bearbeitet (Ausschnitt);https://80.lv/articles/procedural-landscape-generation-for-game-environments/, Abgerufen am: 08.11.2018.

Efros, A. (2005). Image pyramids and blending. Bilder aus Vorlesungsfolien, bearbeitet (Ausschnitte und Anordnung); http://graphics.cs.cmu.edu/courses/15-463/2005_fall/www/Lectures/Pyramids.pdf, Abgerufen am: 08.11.2018.

Efros, A. A. and Freeman, W. T. (2001). Image quilting for texture synthesis and transfer. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 341–346. ACM. https://people.eecs.berkeley.edu/~efros/research/quilting/quilting.pdf, Abgerufen am: 08.11.2018.

Encyclopaedia Britannica, E. (2006). Erosion. Bild, bearbeitet (Ausschnitte und Anordnung); https://www.britannica.com/science/erosion-geology/media/191809/148678, Abgerufen am: 08.11.2018.

Encyclopaedia Britannica, E. (2018a). Ecosystem. *Encyclopaedia Britannica (Online)*. https://www.britannica.com/science/ecosystem, Abgerufen am: 08.11.2018.

Encyclopaedia Britannica, E. (2018b). Erosion. *Encyclopaedia Britannica (Online)*. https://www.britannica.com/science/erosion-geology, Abgerufen am: 08.11.2018.

Encyclopaedia Britannica, F. (2012a). Fluvial process. *Encyclopaedia Britannica (Online)*. https://www.britannica.com/science/fluvial-process, Abgerufen am: 08.11.2018.

Encyclopaedia Britannica, O. (2012b). Orographic lift. *Encyclopaedia Britannica (Online)*. Bild, https://www.britannica.com/science/orographic-precipitation/media/433062/140263, Abgerufen am: 08.11.2018.

Encyclopaedia Britannica, W. (2016). Weathering. *Encyclopaedia Britannica (Online)*. https://www.britannica.com/science/weathering-geology, Abgerufen am: 08.11.2018.

Freiknecht, J. and Effelsberg, W. (2017). A survey on the procedural generation of virtual worlds. *Multimodal Technologies and Interaction*, 1(4):27. hpcg.purdue.edu/papers/Smelik14CGF.pdf, Abgerufen am: 08.11.2018.

Gamito, M. N. and Maddock, S. C. (2009). Accurate multi-dimensional poisson-disk sampling. *ACM Transactions on Graphics (TOG)*, 29(1):8:1–8:19. http://staffwww.dcs.shef.ac.uk/people/s.maddock/research/gamito/papers/poisson.pdf, Abgerufen am: 08.11.2018.

Gatys, L. A., Ecker, A. S., and Bethge, M. (2015). A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576*. https://www.robots.ox.ac.uk/~vgg/rg/papers/1508.06576v2.pdf, Abgerufen am: 08.11.2018.

Gatys, L. A., Ecker, A. S., and Bethge, M. (2016). Image style transfer using convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2414–2423. https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Gatys_Image_Style_Transfer_CVPR_2016_paper.pdf, Abgerufen am: 08.11.2018.

Génevaux, J.-D., Galin, É., Guérin, E., Peytavie, A., and Benes, B. (2013). Terrain generation using procedural models based on hydrology. *ACM Transactions on Graphics (TOG)*, 32(4):143. https://arches.liris.cnrs.fr/publications/articles/SIGGRAPH2013_PCG_Terrain.pdf, Abgerufen am: 08.11.2018.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. https://www.deeplearningbook.org/, Abgerufen am: 08.11.2018.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680. https://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf, Abgerufen am: 08.11.2018.

Google-Maps (2018). Sattelitenaufnahme von nordspanien. Bild, bearbeitet (rote Linie); https://www.google.de/maps/place/Spanien/@41.3065343,-4.9660376,997770m/data=!3m1!1e3!4m5!3m4!1s0xc42e3783261bc8b:0xa6ec2c940768a3ec!8m2!3d40.463667!4d-3.74922, Abgerufen am: 08.11.2018.

Goosse, H., Barriat, P., Lefebvre, W., Loutre, M., and Zunz, V. (2008). *Introduction to climate dynamics and climate modeling.* Online. Online Buch verfuegbar unter http://www.climate.be/textbook, Abgerufen am: 08.11.2018.

Guérin, E., Digne, J., Galin, E., Peytavie, A., Wolf, C., Benes, B., and Martinez, B. (2017). Interactive example-based terrain authoring with conditional generative adversarial networks. *ACM Trans. Graph.*, 36(6):228:1–228:13. https://hal.archives-ouvertes.fr/hal-01583706/file/tog.pdf, Abgerufen am: 08.11.2018.

Harris, M. J. (2003). *Real-time cloud simulation and rendering.* PhD thesis, University of North Carolina at Chapel Hill. http://www.markmark.net/dissertation/harrisDissertation.pdf, Abgerufen am: 08.11.2018.

Hart, P. E., Nilsson, N. J., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107. https://www.cs.auckland.ac.nz/courses/compsci709s2c/resources/Mike.d/astarNilsson.pdf, Abgerufen am: 08.11.2018.

Hendrikx, M., Meijer, S., Van Der Velden, J., and Iosup, A. (2013). Procedural content generation for games: A survey. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 9(1):1. http://www.st.ewi.tudelft.nl/iosup/pcg-g-survey11tomccap_rev_sub.pdf, Abgerufen am: 08.11.2018.

Herold, M. (2017). Spiele-rekorde - die größten open-worlds der spielegeschichte. Online-Artikel der GameStar;https://www.gamestar.de/artikel/spiele-rekorde-die-groessten-open-worlds-der-spielegeschichte,3311516.html, Abgerufen am: 08.11.2018.

Hyttinen, T. (2017). Terrain synthesis using noise. Master's thesis, University of Tampere. https://tampub.uta.fi/bitstream/handle/10024/101043/GRADU-1494236249.pdf, Abgerufen am: 08.11.2018.

Jákó, B. (2011). Fast hydraulic and thermal erosion on gpu. In *Eurographics 2011 - Short Papers.* http://old.cescg.org/CESCG-2011/papers/TUBudapest-Jako-Balazs.pdf, Abgerufen am: 08.11.2018.

Jones, T. R. (2006). Efficient generation of poisson-disk sampling patterns. *Journal of Graphics Tools*, 11(2):27–36. https://pdfs.semanticscholar.org/676a/8b52e96c2d494cbb020ba61eb40b8efb5c4c.pdf, Abgerufen am: 08.11.2018.

Karpathy, A. (2018). Cs231n: Convolutional neural networks for visual recognition. Bilder aus Kursnotizen, bearbeitet (Ausschnitte und Anordnung); http://cs231n.github.io/convolutional-networks/, Abgerufen am: 08.11.2018.

Koistinen, V. (2007). Vegetation. Lizensiertes Bild unter CC BY-SA 3.0, bearbeitet (Ausschnitt); https://commons.wikimedia.org/wiki/File:Vegetation.png, Abgerufen am: 08.11.2018.

Kwatra, V., Schödl, A., Essa, I., Turk, G., and Bobick, A. (2003). Graphcut textures: image and video synthesis using graph cuts. *ACM Transactions on Graphics (ToG)*, 22(3):277–286. https://www.think-cell.com/en/career/talks/pdf/think-cell_article_siggraph2003.pdf, Abgerufen am: 08.11.2018.

Lane, B., Prusinkiewicz, P., et al. (2002). Generating spatial distributions for multilevel models of plant communities. In *Graphics Interface 2002 Conference*, pages 69–80. Citeseer. https://pdfs.semanticscholar.org/ef0e/f687f159099ebc5fda02cda4306846e48f3a.pdf, Abgerufen am: 08.11.2018.

Larive, M. and Gaildrat, V. (2006). Wall grammar for building generation. In *Proceedings of the 4th International Conference on Computer Graphics and Interactive Techniques in Australasia and Southeast Asia*, pages 429–437. ACM.

Ledig, C., Theis, L., Huszár, F., Caballero, J., Cunningham, A., Acosta, A., Aitken, A. P., Tejani, A., Totz, J., Wang, Z., et al. (2016). Photo-realistic single image super-resolution using a generative adversarial network. http://openaccess.thecvf.com/content_cvpr_2017/papers/Ledig_Photo-Realistic_Single_Image_CVPR_2017_paper.pdf, Abgerufen am: 08.11.2018.

Lluch, J., Camahort, E., and Vivó, R. (2003). Procedural multiresolution for plant and tree rendering. In *Afrigraph*.

Ludwig, J. A., Quartet, L., and Reynolds, J. F. (1988). *Statistical Ecology: A Primer in Methods and Computing*, volume 1. John Wiley & Sons.

Mei, X., Decaudin, P., and Hu, B.-G. (2007). Fast hydraulic erosion simulation and visualization on gpu. In *Computer Graphics and Applications, 2007. PG'07. 15th Pacific Conference on*, pages 47–56. IEEE. http://www-ljk.imag.fr/Publications/Basilic/com.lmc.publi.PUBLI_Inproceedings@117681e94b6_fff75c/FastErosion_PG07.pdf, Abgerufen am: 08.11.2018.

Milling, J. (2015). Voxel engines. Bild von Mincecraft in Blogeintrag;http://jeffmilling.blogspot.com/2015/03/voxel-engines.html, Abgerufen am: 08.11.2018.

Moss, R. (2016). 7 uses of procedural generation that all developers should study. Bild von Spelunky aus Online-Artikel auf Gamasutra;http://www.gamasutra.com/view/news/262869/7_uses_of_procedural_generation_that_all_developers_should_study.php, Abgerufen am: 08.11.2018.

Musgrave, F. K. (2003). Procedural fractal terrains. *Texturing & Modeling: A Procedural Approach*, pages 489–506. https://www.classes.cs.uchicago.edu/archive/2015/fall/23700-1/final-project/MusgraveTerrain00.pdf, Abgerufen am: 08.11.2018.

Musgrave, F. K., Kolb, C. E., and Mace, R. S. (1989). The synthesis and rendering of eroded fractal terrains. In *ACM Siggraph Computer Graphics*, volume 23(2), pages 41–50. ACM. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.27.8939&rep=rep1&type=pdf, Abgerufen am: 08.11.2018.

National-Weather-Service (2017). The basic hydrologic (water) cycle. Bild von Website des National Weather Service - JetStream, bearbeitet (Ausschnitt, Beschriftungen ersetzt, Pfeile entfernt);https://www.weather.gov/jetstream/hydro, Abgerufen am: 08.11.2018.

Neidhold, B., Wacker, M., and Deussen, O. (2005). Interactive physically based fluid and erosion simulation. In *Proceedings of the First Eurographics Conference on Natural Phenomena*, pages 25–33. http://graphics.uni-konstanz.de/publikationen/Neidhold2005InteractivePhysicallyBased/Neidhold2005InteractivePhysicallyBased.pdf, Abgerufen am: 08.11.2018.

Olsen, J. (2004). Realtime procedural terrain generation. http://web.mit.edu/cesium/Public/terrain.pdf, Abgerufen am: 08.11.2018.

Pérez, P., Gangnet, M., and Blake, A. (2003). Poisson image editing. *ACM Transactions on graphics (TOG)*, 22(3):313–318. https://xeds.eu/clone/poisson.pdf, Abgerufen am: 08.11.2018.

Plans, D. and Morelli, D. (2012). Experience-driven procedural music generation for games. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(4):192–198. http://axon.cs.byu.edu/~dan/673/papers/plans.pdf, Abgerufen am: 12.01.2019.

Radford, A., Metz, L., and Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*. https://arxiv.org/pdf/1511.06434.pdf, Abgerufen am: 08.11.2018.

Ricklefs, R. E. (2000). *The Economy of Nature*. W. H. Freeman. Bearbeitet Version von Whittakers Communities and ecosystems (1975).

Russell, S. and Norvig, P. (2009). *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, 3rd edition.

Shaker, N., Togelius, J., and Nelson, M. J. (2016). *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer. http://pcgbook.com/, Abgerufen am: 08.11.2018.

Sinha, S. N. (2004). Graph cut algorithms in vision, graphics and machine learning an integrative paper. https://pdfs.semanticscholar.org/1e3e/e2bfb3535d79d0dea81c0c5ce08174c6a994.pdf, Abgerufen am: 08.11.2018.

Smelik, R. M., Tutenel, T., Bidarra, R., and Benes, B. (2014). A survey on procedural modelling for virtual worlds. In *Computer Graphics Forum*, volume 33(6), pages 31–50. Wiley Online Library. http://hpcg.purdue.edu/papers/Smelik14CGF.pdf, Abgerufen am: 08.11.2018.

Smelik, R. M., Tutenel, T., de Kraker, K. J., and Bidarra, R. (2011). A declarative approach to procedural modeling of virtual worlds. *Computers & Graphics*, 35(2):352–363.

Snider-Held, S. (2017). Neural networks and the future of 3d procedural content generation. Online-Artikel; https://towardsdatascience.com/neural-networks-and-the-future-of-3d-procedural-content-generation-a2132487d44a, Abgerufen am: 08.11.2018.

Soltow, E. (2010). Bildsegmentierung mittels graph-cuts. Master's thesis, Universität zu Lübeck. https://www.mic.uni-luebeck.de/fileadmin/mic/publications/student_projects/bachelor_thesis/SoltowBachelor.pdf, Abgerufen am: 08.11.2018.

SpeedTree (2017). Produktbild von speedtree cinema 8 evaluation. Bearbeitet (Ausschnitt);https://store.speedtree.com/store/speedtree-cinema-8-evaluation/, Abgerufen am: 08.11.2018.

Stam, J. (1999). Stable fluids. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 121–128. ACM Press/Addison-Wesley Publishing Co. http://www.dgp.toronto.edu/people/stam/reality/Research/pdf/ns.pdf, Abgerufen am: 08.11.2018.

Stam, J. (2003). Real-time fluid dynamics for games. In *Proceedings of the Game Developer Conference*. https://pdfs.semanticscholar.org/847f/819a4ea14bd789aca8bc88e85e906cfc657c.pdf, Abgerufen am: 08.11.2018.

Stangl, R. (2017). Procedural content generation: Techniques and applications. https://umm-csci.github.io/senior-seminar/seminars/spring2017/stangl.pdf, Abgerufen am: 08.11.2018.

Stockham, A. J., Schultz, D., Fairman, J., and Draude, A. (2018). Quantifying the rain-shadow effect: Results from the peak district, british isles. *American Meteorological Society. Bulletin*, 99(4). https://journals.ametsoc.org/doi/pdf/10.1175/BAMS-D-17-0256.1, Abgerufen am: 08.11.2018.

Strout, J. (2017). Procedural audio in unity. Blogpost; https://www.gamasutra.com/blogs/JoeStrout/20170223/292317/Procedural_Audio_in_Unity.php, Abgerufen am: 12.01.2019.

Szeliski, R., Uyttendaele, M., and Steedly, D. (2011). Fast poisson blending using multi-splines. In *Computational Photography (ICCP), 2011 IEEE International Conference on*, pages 1–8. IEEE. http://www.cs.cmu.edu/~ICCP2011/papers/Paper%2016/p16.pdf, Abgerufen am: 08.11.2018.

Togelius, J., Yannakakis, G. N., Stanley, K. O., and Browne, C. (2011). Search-based procedural content generation: A taxonomy and survey. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(3):172–186. http://julian.togelius.com/Togelius2011Searchbased.pdf, Abgerufen am: 08.11.2018.

Totilo, S. (2010). Civilization v preview: Small changes, big differences. Bild von Civilization V aus Online-Artikel auf Kotaku;https://kotaku.com/5489814/civilization-v-preview-small-changes-big-differences, Abgerufen am: 08.11.2018.

Turner, S. (2017). Saving for a rainy day. Blogpost über Biomgeneration in "Dragons Abound"; https://heredragonsabound.blogspot.com/2017/03/saving-for-rainy-day.html, Abgerufen am: 08.11.2018.

Veltin, T. (2017). Offene spielwelten - die wahrscheinlich größten open-world-spiele. Online-Artikel der GamePro; https://www.gamepro.de/artikel/offene-spielwelten-die-wahrscheinlich-groessten-open-world-spiele,3311519.html, Abgerufen am: 08.11.2018.

Vertragsparteien (1992). Übereinkommen über die biologische vielfalt. Definition Ökosystem in Artikel 2 Begriffsbestimmungen. Übersetzung der Schweizerischen Bundesverwaltung; https://www.admin.ch/opc/de/classified-compilation/19920136, Abgerufen am: 08.11.2018.

Vigil, J. F. (2011). Plattengrenzen. Lizensiertes Bild unter Public Domain, von José F. Vigil (USGS), bearbeitet von Eurico Zimbres und TomCatX; https://commons.wikimedia.org/wiki/File:Plattengrenzen.png, Abgerufen am: 08.11.2018.

Viitanen, L. (2012). Physically based terrain generation: Procedural heightmap generation using plate tectonics. Master's thesis, Helsinki Metropolia University of Applied Sciences. http://www.theseus.fi/bitstream/handle/10024/40422/Viitanen_Lauri_2012_03_30.pdf, Abgerufen am: 08.11.2018.

Voyagergames (2015). Procedural world generation. Blogpost über Welt-Generation und Biome in Öuter Colony"; https://voyagergames.com/procedural-world-generation/, Abgerufen am: 08.11.2018.

Wallace, J. and Hobbs, P. (2006). *Atmospheric science: an introductory survey.* Academic Press, 2nc edition.

Worley, S. (1996). A cellular texture basis function. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 291–294. ACM. http://weber.itn.liu.se/~stegu/TNM084-2017/worley-originalpaper.pdf, Abgerufen am: 08.11.2018.

Wulff-Jensen, A., Nerup Rant, N., Nordvig Møller, T., and Aksel Billeskov, J. (2018). Deep convolutional generative adversarial network for procedural 3d landscape generation based on dem. In *6th EAI International Conference on Arts and Technology, Interactivity & Game Creation*, pages 85–94. http://vbn.aau.dk/files/272397944/corrected_2017_A_Wulff_Jensen_N_N_Rant_T_N_M_ller_and_J_A_Billeskov_Deep_Convolutional_Generative_Adversarial_Network_for_Procedural_3D_Landscape_Generation_Based_on_DEM.pdf, Abgerufen am: 08.11.2018.

Yerpo (2008). Population distribution. Lizensiertes Bild unter CC BY-SA 4.0, bearbeitet (Anordnung); https://commons.wikimedia.org/wiki/File:Population_distribution.svg, Abgerufen am: 08.11.2018.

Yuheng, S. and Hao, Y. (2017). Image segmentation algorithms overview. *CoRR*, abs/1707.02051. https://arxiv.org/ftp/arxiv/papers/1707/1707.02051.pdf, Abgerufen am: 08.11.2018.

Zhou, H., Sun, J., Turk, G., and Rehg, J. M. (2007). Terrain synthesis from digital elevation models. *IEEE transactions on visualization and computer graphics*, 13(4):834–848. https://www.cc.gatech.edu/~turk/my_papers/terrain_synth_tvcg.pdf, Abgerufen am: 08.11.2018.