

Masterarbeit
in der angewandten Informatik
AI-2019-MA-007

Prozedurale Verteilung von Vegetation auf einem Höhenfeld-basiertem Terrain

Martin Lesser
martin.lesser@mailbox.org

Abgabedatum: 9. September 2019
SS 2019

Prof. Dr. Jörg Sahm
Prof. Dr. Uwe Altenburg

Inhaltsverzeichnis

Abbildungsverzeichnis	IV
Tabellenverzeichnis	VI
Listings	VII
1 Kurzfassung	1
2 Einleitung	2
2.1 Aufbau der Arbeit	3
3 Anforderungen	4
4 Theoretische Grundlagen	6
4.1 Prozedurale Synthese	6
4.1.1 Rauschen	6
4.1.2 Fraktale	7
4.1.3 Placement Algorithmen	7
4.2 Geobotanik	8
4.2.1 Abiotische Standortfaktoren	9
4.2.2 Einstrahlungstypus & Albedo	9
4.3 Zusammenfassung	10
5 Anwendungsgebiete	11
5.1 Videospiele	11
5.1.1 Vegetation	11
5.1.2 Terrain	13
5.2 Filme	13
5.3 Medizin	14
5.4 Zusammenfassung	14
6 State-of-the-Art	15
6.1 GeNa	15
6.2 Unreal Engine 4: Procedural Foliage Tool	16
6.3 Horizon: Zero Dawn	17
6.4 Weitere Lösungen	18
6.4.1 Vegetation Studio	18
6.4.2 Far cry 5	19
6.5 Zusammenfassung	20
7 Abgrenzung	21
7.1 Gewählte Merkmale	21

7.2 Übersicht	21
7.3 Auswertung	22
8 Konzept	23
8.1 Daten	23
8.1.1 Vegetationsart	23
8.1.2 Bodenart	23
8.1.3 Biom	24
8.1.4 Map	24
8.2 Sonnenbestrahlung	25
8.2.1 Atmosphäre und Albedo	25
8.2.2 Sonnenstand und Sonnenwanderung	25
8.3 Relief	27
8.4 Bodentiefe	29
8.5 Wasservorrat & Verdunstung	29
8.6 Wachstumswahrscheinlichkeiten	30
8.7 Gesamtsystem	30
8.8 Zusammenfassung	32
9 Implementierung	33
9.1 Data	33
9.1.1 Yaml-Dateien	33
9.1.2 Sonne	35
9.1.3 Image	35
9.2 Logic	36
9.2.1 Controller	36
9.2.2 Insolation	38
9.2.3 Tageslichtstunden	38
9.2.4 Tatsächliche Energiemenge	40
9.2.5 Reflektion	41
9.2.6 Orography	41
9.2.7 Edaphology	42
9.2.8 Hydrology	42
9.2.9 Probabilities	43
9.3 GUI	43
9.4 Zusammenfassung	44
10 Ergebnisse	46
10.1 GUI	46
10.2 Bedienung	47
10.3 Farbkodierung	48
10.4 Ergebnisbilder	49
10.5 Zusammenfassung	53
11 Ausblick	54
11.1 Wärme	54
11.2 Erosion	54
11.3 Ausbreitungssimulation	54

11.4 Biotische Faktoren	55
12 Selbstständigkeitserklärung	56
Literaturverzeichnis	57

Abbildungsverzeichnis

4.1	Beispiele von Perlin Noise. (links) Perlin's berühmte erste prozedurale Rauschfunktion. (rechts) Perlin's berühmte Mamor Vase, eine der ersten prozeduralen Texturen. [Per85]	7
4.2	(links) Fraktale Strukturen im Romanesco. [rom] (rechts) Mandelbrot Fraktal von Benoit Mandelbrot. [BK17]	7
4.3	Bioregionen des Bundesstaates Victoria in Australien. [evc]	8
4.4	Toleranzbreite der Intensität eines Faktors und dessen Auswirkung auf das Wachstum einer Vegetation. [Wit12, S. 105]	9
4.5	Berechnung der Sonnenstrahl-Energiemenge, die den Erdboden erreicht. [Wal86, S. 92]	10
4.6	Liste der Albedo-werte von verschiedenen Stoffen [Oke02]	10
5.1	Minecraft (links [Min]) und No man's sky (rechts [NoM]) lassen ihre Spielwelten komplett prozedural erzeugen.	11
5.2	Das Anfangszeichen b wird mehrmals anhand von Ersetzungsregeln ausgewechselt. [PL12]	12
5.3	Drei L-Systeme und ihre Turtle-Grafiken. Ein L-System besteht aus der Anzahl der Ersetzungen (n), dem Winkel für Drehungen (δ), einem Anfangsbuchstaben (X) und den Ersetzungsregeln (z.B. $F \rightarrow FF$).[PL12]	12
5.4	Erzeugung eines 3D-Terrains anhand eines Grauwertbildes (Heightmap), das mittels Perlin Noise erzeugt wurde. [EL15]	13
5.5	Das Intro zu Avatar zeigt eine riesige Landschaft, die mit Speedtree Vegetationen gefüllt wurde. [Spea]	13
6.1	Ein User platziert einen Felsen in Unity 3D und lässt GeNa anschließend weitere prozedural hinzufügen. [GeN]	15
6.2	Mit dem Unreal Engine Procedural Foliage Tool werden zu Beginn ein paar Pflanzen gesetzt. [Unr]	17
6.3	Anschließend wird das Wachstum und die Verbreitung der Pflanzen simuliert. [Unr]	17
6.4	Horizon: Zero Dawn ohne Echtzeit-Platzierung. [Hor]	18
6.5	Horizon: Zero Dawn nach Echtzeit-Platzierung in derselben Szene. [Hor]	18
6.6	Einstellbare Regeln für den Platzierungsalgorithmus von Vegetation Studio. [veg]	19
6.7	Die Far cry 5 Engine berechnet auf Basis eines Höhenfeldes verschiedene Maps, welche dann zur Bestimmung des Sub-Bioms verwendet werden. [far]	19
8.1	Der Sonnenstand wird durch Azimut und Elevation (Höhe) definiert. [Azi]	25
8.2	Die Abbildung von Polarkoordinaten zu kartesischen Koordinaten. [Pol]	26
8.3	Das gewählte Koordinatensystem. [coo]	27

8.4 Die Normale (grün) des ausgewählten Scheitelpunkts (rot) wird über die Summe der Normalen (gelb) seiner Nachbarn (blau) ermittelt. Die Darstellung ist grob und nicht exakt. (Quelle: eigene Darstellung)	28
8.5 Normalen Vektoren auf einer 3D-Fläche. [nor]	28
8.6 Der Winkel zwischen zwei Normalen wird mittels Punktprodukt ermittelt. [ang]	29
8.7 Ablauf der Berechnungen der benötigten Informationen für die Wahrscheinlichkeitsberechnung. (Quelle: eigene Darstellung)	31
8.8 Ablauf der Berechnungen für die Wachstumswahrscheinlichkeit einer Vegetationsart. (Quelle: eigene Darstellung)	32
 10.1 Hauptmenü der Applikation. (eigene Darstellung)	46
10.2 Auflistung der angelegten Vegetationsarten. (eigene Darstellung)	47
10.3 Erstellung einer neuen Vegetationsart. (eigene Darstellung)	47
10.4 Das Laden einer Map und die Einstellungsmöglichkeiten für die Berechnungen. (eigene Darstellung)	48
10.5 Aus einem Höhenfeld (links) wird mittels Posterisation eine Soil-IDs-Map (rechts) erzeugt. (eigene Darstellung)	49
10.6 Die Ergebnisse eines 512 x 512 Pixel großen Höhenfeldes. (eigene Darstellung)	50
10.7 Die Ergebnisse eines 1024 x 1024 Pixel großen Höhenfeldes. (eigene Darstellung)	51
10.8 . (eigene Darstellung)	52
10.9 Die Ergebnisse eines 2048 x 2048 Pixel großen Höhenfeldes. (eigene Darstellung)	53

Tabellenverzeichnis

7.1 Übersicht der Leistungsmerkmale aller vorgestellten Softwarelösungen. 22

Listings

9.1	Source-Code der Vegetation-Klasse.	33
9.2	Beispiel für eine Yaml-Datei.	34
9.3	Auszug aus der bioms.yml Datei.	34
9.4	Auszug aus der soil_types.yml Datei.	34
9.5	Auszug aus der maps.yml Datei.	34
9.6	Auszug aus der vegetations_types.yml Datei.	35
9.7	Implementierung der Sonne.	35
9.8	Implementierung der Speicher- und Ladenfunktion der Image-Klasse.	36
9.9	Die Funktion zum Laden der Vegetationsarten.	37
9.10	Diese Funktion lässt die Berechnung der Insolation Klasse ausführen und lässt das Ergebnis einerseits in der GUI anzeigen und andererseits als PNG speichern.	37
9.11	Die Berechnung der Winkel, die auf den Azimut- und Elevation-Wert der Sonne pro Stunde addiert bzw. subtrahiert werden.	39
9.12	Die Schleife zur schrittweisen Verfolgung des Sonnenlichtstrahls. Falls das Terrain den Strahl blockiert, bricht der Algorithmus ab und der Pixel erhält für diese Stunde kein Sonnenlicht.	40
9.13	Ein Teil der Energie durch die Sonnenstrahlen geht durch die Atmosphäre und Reflektion verloren.	40
9.14	Ein Teil der Reflektion der Bodenoberfläche wird an dessen angrenzenden Pixel abgestrahlt.	41
9.15	Die Funktion berechnet die Bodentiefe an jedem Punkt der Map anhand des Winkels und der maximalen Bodentiefe.	42
9.16	Berechnung der Wassermenge an einem Punkt der Karte.	42
9.17	Die Funktion zur Berechnung der Wahrscheinlichkeit anhand der vorhandenen und benötigten Menge (z.B. Wasser oder Energie).	43
9.18	Alle Fenster-Klassen werden angelegt und in einer Liste gespeichert. Jedes Fenster-Objekt erhält eine Referenz auf das Hauptfenster.	44
9.19	Methoden für das Wechseln und Aktualisieren eines Fensters.	44

1 Kurzfassung

Die Entwicklung neuer Blockbuster in der Film- und Videospielindustrie drängt auf immer höher liegende Maßstäbe und stellt die Entwickler vor immer größer werdende Herausforderungen, die zunehmenden Vorgaben zu erfüllen. Das Einstellen von mehr Entwicklern sprengt dabei schnell das Budget und kann mit dem Wachstum der Zielsetzungen nicht mithalten. Deshalb sind neue Techniken gefragt, die dieses Problem lösen können und die prozedurale Synthese ist der vielversprechendste Ansatz.

Bei der prozeduralen Generierung geht es um die Entwicklung von Algorithmen zur Content-Erzeugung. Anstatt beispielsweise einen Künstler eine Textur zeichnen zu lassen, kann ein Algorithmus Texturen durch Berechnungen generieren. Die größte Herausforderung ist dabei eine hohe Qualität der Ergebnisse der Algorithmen zu gewährleisten, um mit der Arbeit eines Künstlers mithalten zu können. Ein Teil des zunehmenden Arbeitsvolumens stellt das Platzieren von Vegetation auf riesigen Terrains dar.

Diese Arbeit untersucht deshalb die Möglichkeit, einen prozeduralen Algorithmus zu entwickeln, der die Wahrscheinlichkeit für das Platzieren von Vegetation auf Basis eines Höhenfeldes (von einem Terrain) übernimmt. Das Ergebnis soll realistisch, stimmig und organisch sein, sodass es mit der Qualität der Erzeugnisse eines Künstlers gleichkommen kann.

2 Einleitung

Virtuelle Welten gewinnen im Kontext von Spielen und Simulationen zunehmend an Bedeutung. Ein Beispiel ist das Videospiel *Horizon: Zero Dawn* (2017) [Gam17]. Die Größe und der Detailgrad der Spielwelt von Horizon sind so hoch, dass der Speicherbedarf pro Quadratkilometer circa 40 MB¹ beträgt. Folglich ist der Ressourcen-Aufwand zur Entwicklung des Spiels beträchtlich, sodass die Schaffung der Inhalte mit herkömmlichen Mitteln nicht zu bewältigen ist. Die Entwickler mussten deshalb prozedurale Generierung (auch procedural content generation, kurz PCG genannt) einsetzen, um die Anforderungen bewerkstelligen zu können.² Spielstudios begegnen deshalb häufig zwei Problemen:

- Erschaffung einer realistischen und großen Spielwelt mit hohem Detailgrad und genügend Spielinhalten, um Spieler ausreichend lang zu beschäftigen und den Spielpreis zu rechtfertigen [FE17]
- Einhaltung der Zeit und des Budgets [FE17]

Der Videospiel-Designer Will Wright (Sims (2000) [Max00], SimCity (1989) [Max89]) nannte dieses Problem *The Mountain of Content Problem* [FE17]. Dies hat dazu geführt, dass Entwickler Methoden erfanden, um Texturen, 3D-Modelle oder Terrains mittels Algorithmus, also prozedural, zu generieren. Auch die Platzierung von Vegetation per Hand durch einen Grafik-Designer kostet viel Zeit und ist zudem eine monotone Arbeit, vor allem wenn große virtuelle Welten „bepflanzt“ werden müssen. Deshalb ist die prozedurale Platzierung von Vegetation ein wichtiges Hilfsmittel, um den zunehmend hohen Anforderungen gerecht zu werden. Mittels PCG kann die Platzierung von Vegetation automatisiert werden, sodass größere Welten für dasselbe Budget gebaut werden und sich Grafik-Designer auf wichtigere Inhalte konzentrieren können. Dabei ist es sehr wichtig, dass die PCG gute Ergebnisse liefert, die mit der Qualität, die ein Designer schafft, gleichkommt. Denn nur dann kann die PCG diese Arbeit abnehmen.

Damit ein Platzierungs-Algorithmus gute Qualität liefert, ist es notwendig, die Gegebenheit des zugrundeliegenden Terrains beim Platzieren der Pflanzen zu beachten. Daraus ergeben sich folgende Anforderungen:

- Berücksichtigung der Sonneneinstrahlung über einen Tag verteilt
- Berücksichtigung des Reliefs
- Berücksichtigung der Feuchtigkeit
- Berücksichtigung der Bodenart und Bodentiefe
- Realistische Ergebnisse

¹<https://www.guerrilla-games.com/read/gpu-based-procedural-placement-in-horizon-zero-dawn>

²Ebd.

Die gegebenen Informationen für ein Terrain sind zum einen das Höhenfeld und zum anderen die Bodenart an jedem Punkt des Terrains. Aus diesen beiden Informationen müssen die daraus abgeleiteten Daten für die Sonneneinstrahlung, Relief, Feuchtigkeit und Bodentiefe berechnet werden, welche wiederum als Grundlage für den Platzierungsalgorithmus dienen.

2.1 Aufbau der Arbeit

Diese Arbeit geht zuerst auf die selbst gestellten Anforderungen für einen Platzierungs-Algorithmus, der gute Ergebnisse liefern soll, ein. Danach werden die notwendigen theoretischen Grundlagen erörtert. Diese sind die prozedurale Synthese, dessen Definition und Arten, und die Geobotanik, um zu klären welche Faktoren für das Wachstum einer Pflanze entscheidend sind. Danach werden die Anwendungsbereiche der PCG aufgezeigt und im Kapitel State-of-the-Art werden die bekanntesten Vertreter für die prozedurale Platzierung von Objekten vorgestellt. Anschließend werden die Vertreter verglichen und aufgezeigt, welche Anforderungen, die in dieser Arbeit aufgestellt worden, nicht erfüllt werden. Zur Realisierung der eigenen Lösung wird zunächst das Konzept der Applikation vorgestellt und daraufhin die Implementierung von dieser gezeigt. Zuletzt werden die erzeugten Ergebnisse präsentiert und im Kapitel Ausblick die Möglichkeiten zum Ausbau der entwickelten Applikation aufgelistet.

3 Anforderungen

Anhand einer Benutzeroberfläche sollen alle notwendigen Daten zur Berechnung der Wahrscheinlichkeiten zur Platzierung einer Pflanzenart eingegeben, gespeichert und später geladen werden können. Auch die Ergebnisse der Berechnungen sollen gespeichert und geladen werden können. Die Berechnungen zur Wahrscheinlichkeit der Platzierung einer Vegetation soll auf Grundlage der Bedürfnisse der Pflanze und den Gegebenheiten des Terrains stattfinden. Die Gegebenheiten sind die Sonneneinstrahlung, Wassermenge, Bodenart und Bodentiefe pro Pixel. Diese Daten werden anhand der Höhenfeld-Map und der Bodenart-Map berechnet. Jeder Pixel dieser Map enthält also einen Höhenwert bzw. die ID für eine Bodenart. Aus diesen Daten ergeben sich folgende Anforderungen:

1. Sonneneinstrahlung pro Pixel pro Stunde in Kilokalorien berechnen. Unter folgenden Berücksichtigungen:
 - a) Solarkonstante
 - b) Absorption durch die Atmosphäre
 - c) Wolkenreflektion
 - d) Diffusion der Atmosphäre
 - e) Albedowert des Bodens
 - f) Reflektion auf umliegende Pixel
 - g) Sonnenstand (definiert aus Azimut und Elevation) je Stunde
2. Relief/Normal Map anhand des Höhenfeldes berechnen.
3. Bodentiefe anhand des Reliefs berechnen.
4. Wasservorrat pro Quadratmeter berechnen. Unter folgenden Berücksichtigungen:
 - a) Regenmenge pro Tag
 - b) Grundwassermenge
 - c) Bodentiefe
 - d) Wasserabsorption der Bodenart
 - e) Verdunstung durch Sonneneinstrahlung
5. Wahrscheinlichkeit für das Wachsen einer Pflanzenart. Unter folgenden Berücksichtigungen:
 - a) benötigte Kalorienmenge
 - b) benötigte Bodentiefe

- c) benötigte Wassermenge
 - d) benötigte Bodenart
6. GUI zum Anlegen, Speichern und Laden von Biomen, Bodenarten, Maps und Pflanzenarten.
- a) Anlegen der Biominformationen: atmosphärische Diffusion, atmosphärische Absorption, Wolkenreflektion, durchschnittliche Regenmenge pro Tag und Grundwassermenge pro Quadratmeter
 - b) Anlegen der Bodenartinformationen: ID, Albedowert, Wasserabsorption
 - c) Anlegen der Mapinformationen: Name, Pfad zum Höhenfeldbild, Pfad zum Bodenartbild, Wert für die Höhenwertkonversion, maximale Bodentiefe, Pixelgröße
 - d) Anlegen der Pflanzenartinformationen: benötigte tägliche Energiemenge, benötigter täglicher Wasserbedarf, benötigte Bodenart, benötigte Bodentiefe
7. GUI zum Berechnen, Speichern und Laden der Wahrscheinlichkeiten für eine Pflanzenart für eine gegebene Map.

4 Theoretische Grundlagen

Dieses Kapitel behandelt die prozedurale Synthese, dessen Definition und Beispiele. Weiterhin werden die Geobotanik und die Standortfaktoren einer Vegetation betrachtet.

4.1 Prozedurale Synthese

Ruben M. Smelik et al. definieren die prozedurale Synthese (auch prozedurale Generierung genannt) folgendermaßen:

„Procedural generation is an umbrella term for software algorithms that can (semi-)automatically generate a specific type of content (e.g., a 3D model of a tree) based on a limited set of user input parameters.“ [STKB11]

Nach dieser Definition charakterisiert die Erzeugung von Inhalten mittels Algorithmen anhand einiger Parameter den Begriff. Ferner grenzen Togelius et al. die prozedurale Generierung ein. So sind die Vorhersehbarkeit und Genauigkeit bei der Anwendung der Algorithmen entscheidend, ob diese als prozedural einzuführen sind.

„In contrast, a [...] PCG system involves a non-trivial amount of computation (the procedural part of PCG) between the user input and the content change; the change might not be immediate, not local and not direct in the sense that it is not straightforward for the human to predict the exact changes that will come about as a result as a particular input.“ [TKSY11]

Deshalb ist die Erzeugung von Inhalten mittels Spiel-Editoren keine Form der prozeduralen Synthese. Des Weiteren stellen Togelius et al. klar, dass der Begriff nicht mit Zufall verbunden sein muss. Zufall kann Teil von PCG sein, z.B. wenn eine Spielkarte erzeugt wird, muss dieses Element nicht zwangsläufig enthalten sein. Der Algorithmus kann deterministisch und prozedural sein.

4.1.1 Rauschen

Rauschen ist eine Reihe von unstrukturierten Zufallszahlen, die deterministisch erzeugt werden können. Anhand von Parametern erzeugt eine Rauschfunktion eine Reihe von Zufallszahlen. Das Ergebnis kann grafisch dargestellt werden. Die bekannteste Rauschfunktion ist *Perlin Noise* und wurde von Ken Perlin 1985 erfunden [LLC⁺10]. Abbildung 4.1 zeigt die Anwendung der Perlin Noise Funktion zur Erzeugung von prozeduralen Texturen. In *Improving Noise* [Per02] veröffentlichte Perlin 2002 eine optimierte Version des Perlin Noise Algorithmus. Eine Übersicht weiterer Noise Algorithmen lassen sich in der Präsentation *The Importance of Being Noisy: Fast, High Quality Noise*¹ von Natalya Tatarchuk finden.

¹[https://developer.amd.com/wordpress/media/2012/10/Tatarchuk-Noise\(GDC07-D3D_Day\).pdf](https://developer.amd.com/wordpress/media/2012/10/Tatarchuk-Noise(GDC07-D3D_Day).pdf)

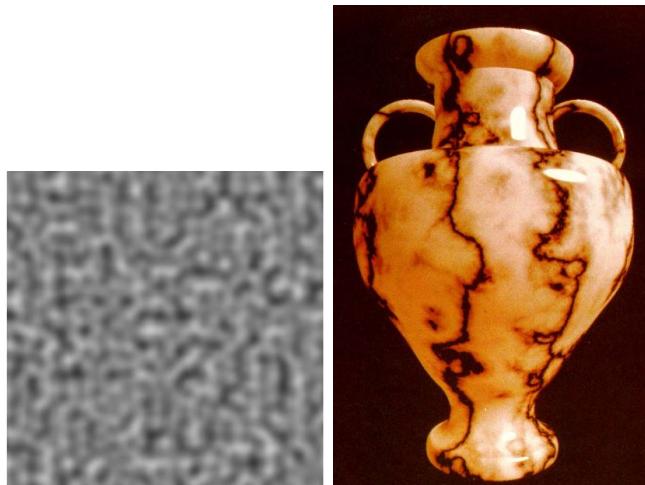


Abbildung 4.1: Beispiele von Perlin Noise. (links) Perlin's berühmte erste prozedurale Rauschfunktion. (rechts) Perlin's berühmte Mamor Vase, eine der ersten prozeduralen Texturen. [Per85]

4.1.2 Fraktale

Fraktale sind Strukturen, die sich im Detail selbst wiederholen, aber immer mit einer kleinen Änderung. In der Natur sind viele Strukturen fraktal. Deshalb kann eine mathematische Beschreibung von Fraktalen genutzt werden, um diese Strukturen prozedural zu erzeugen. Romanesco, Küsten oder Gebirge sind Beispiele für Fraktale in der Natur. Das berühmteste mathematische Fraktal ist das Mandelbrot (siehe Abbildung 4.2), das von Benoit Mandelbrot erfunden wurde [BK17].

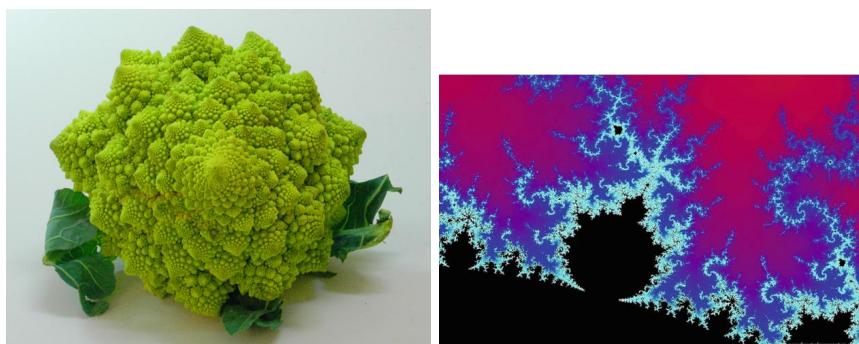


Abbildung 4.2: (links) Fraktale Strukturen im Romanesco. [rom] (rechts) Mandelbrot Fraktal von Benoit Mandelbrot. [BK17]

4.1.3 Placement Algorithmen

Zur Platzierung von Objekten können Maps benutzt werden, die Informationen enthalten, welche relevant für die Platzierung sind. Beispielsweise kann ein Höhenfeld als Grauwertbild dargestellt sein und anhand des Grauwertes kann eine Entscheidung getroffen werden wo eine Vegetation wachsen kann [Ham01]. Auch Informationen eines Geoinformationssystems (GIS) können verwendet werden. Beispielsweise gibt es eine Bioregion-Karte (siehe Abbildung 4.3) des Bundesstaates Victoria in Australien, welches die Regionen nach Ecological Vegetation

Classes (EVC) auflistet. Ein EVC ist eine Region, die durch bestimmte Vegetationstypen definiert wird. Anhand dieser Informationen kann ein prozeduraler Algorithmus entscheiden wo welche Pflanzen platziert werden können.

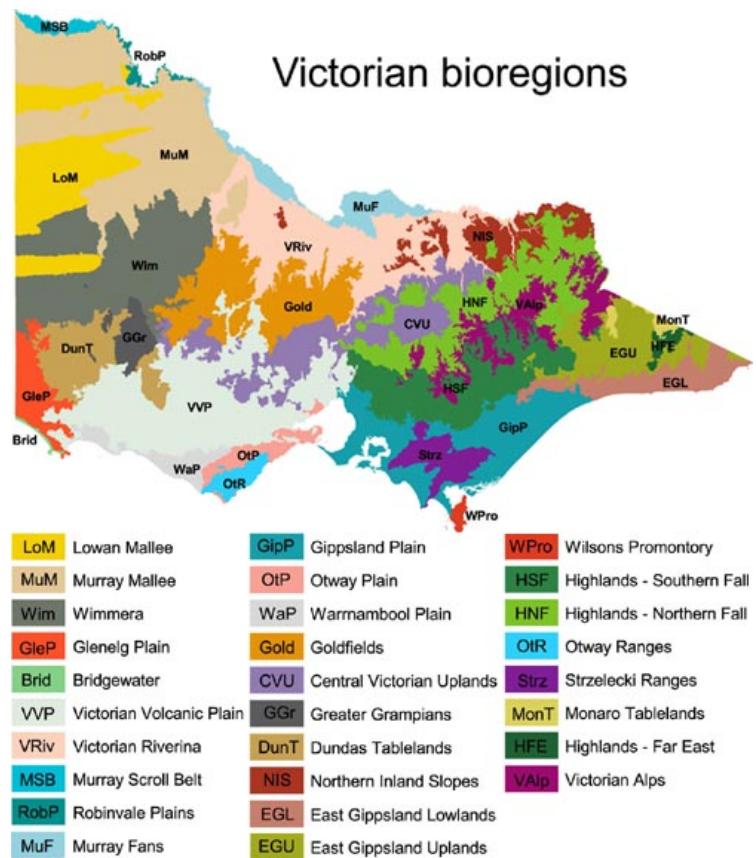


Abbildung 4.3: Bioregionen des Bundesstaates Victoria in Australien. [evc]

4.2 Geobotanik

Die Geobotanik ist die Wissenschaft zur Erforschung des Verhaltens von Pflanzen an ihrem natürlichen Standort [Wal86, S. 9]. Während die Botanik Pflanzen, losgelöst von ihrem Vorkommen in der Natur, untersucht, beschäftigt sich die Geobotanik mit dem Zusammenhang von Pflanzen und ihrer natürlichen Umgebung. Teilgebiete der Geobotanik sind die floristische, historische und ökophysiologische Geobotanik. Die floristische Geobotanik erforscht die in einem Gebiet vorkommende Flora und versucht diese aufzulisten und ihre Verbreitung festzustellen. Die Aufgabe der historischen Geobotanik ist es die heutige Verbreitung der Pflanzen im Kontext der historischen Entwicklung, die diese hervorgebracht hat, zu untersuchen. Die Wechselbeziehung zwischen Pflanzen und deren Umwelt ist Forschungsobjekt der ökophysiologischen Geobotanik. Dabei gilt dem Wirkungsgefüge zwischen den Pflanzen und den abiotischen Umweltfaktoren (Klima, Wasser, Boden) besondere Aufmerksamkeit.

4.2.1 Abiotische Standortfaktoren

Zum Überleben einer Pflanze an einem Standort spielen die Standortfaktoren die wichtigste Rolle. Standortfaktoren sind die Gegebenheiten eines Gebiets, die auf die Pflanze einwirken. Diese können biotisch oder abiotisch sein. Die biotischen Faktoren sind Auswirkungen von Lebewesen (Flora oder Fauna) auf die Vegetation. Dazu zählen Konkurrenz, Feinddruck und Nahrungsangebot. Die abiotischen Faktoren sind physiko-chemische Gegebenheiten wie Feuchtigkeit, Bodenart, Lichteinstrahlung und Temperatur [Wit12, S. 105]. Bezuglich der Auswirkungen von abiotischen Faktoren auf eine Pflanze herrscht eine Toleranzbreite, die sich von Art zu Art unterscheidet. Das *Gesetz der Toleranz* besagt, dass die Intensität eines Faktors eine Bandbreite besitzt für die die Auswirkung auf die Pflanze optimal ist. Ist die Intensität über oder unter der optimalen Bandbreite, so sinkt die positive Wirkung. Abbildung 4.4 veranschaulicht dieses Gesetz.

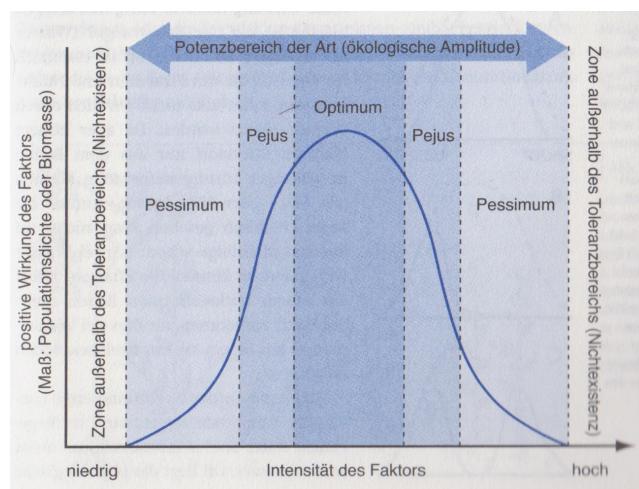


Abbildung 4.4: Toleranzbreite der Intensität eines Faktors und dessen Auswirkung auf das Wachstum einer Vegetation. [Wit12, S. 105]

Wird die Auswirkung aller vorhandenen Standortfaktoren betrachtet, so besagt das *Ge setz des Minimums* von *Sprengel-Liebig* (1840 bis 1855), dass der Standortfaktor mit der negativsten Auswirkung auf die Pflanze der Entscheidende ist [PK⁺99]. Selbst wenn eine Vegetation optimal Licht erhält, optimale Temperaturen und Bodenbedingungen hat aber zu wenig Wasser vorhanden ist, wird das Wachstum maßgeblich vom Faktor Feuchtigkeit beschränkt.

4.2.2 Einstrahlungstypus & Albedo

Der abiotische Faktor Licht hängt vom Einstrahlungstypus ab. Die Lichtstrahlen der Sonne treffen auf die Atmosphäre und werden von dieser teilweise absorbiert, reflektiert und gestreut. Von den gestreuten Lichtstrahlen erreicht ein Teil dennoch den Erdboden. Der Rest wird jedoch in den Weltraum gestreut. Das reflektierte Licht entsteht zum einen durch die Reflektion an der Wolkendecke und zum anderen durch die Reflektion des Bodens. Die konkreten Anteile hängen von der Klimazone ab. Abbildung 4.5 zeigt den Einstrahlungstypus von Mitteleuropa [Wal86, S. 93].

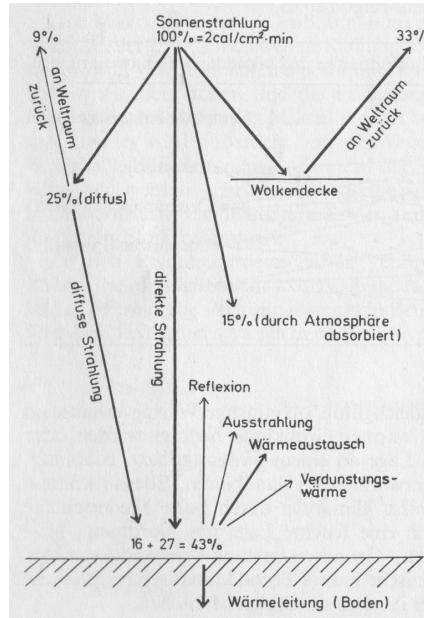


Abbildung 4.5: Berechnung der Sonnenstrahl-Energiemenge, die den Erdboden erreicht.
[Wal86, S. 92]

Der Albedo spiegelt den Reflektionswert des Bodens wider. Der Wert ist der Prozentteil des Lichts, der reflektiert wird. Reflektiert die Bodenart viel Licht, wie es beispielsweise bei Schnee der Fall ist, dann ist der Albedowert hoch. Im Fall von Schnee entspricht dies etwa 84% [Wal86, S. 93]. Abbildung 4.6 zeigt unter anderem die Albedowerte von Gras, Stein, Schiefer und Wasser.

	Surface	Emissivity	Albedo
Ground	Asphalt	0.95	0.125
	Grass	0.93	0.205
Facades	Brick	0.91	0.3
	Concrete	0.805	0.225
	Wood	0.9	0.15
	Stone	0.9	0.275
	Glass	0.895	0.305
	Tile	0.9	0.225
	Slate	0.9	0.1
	Corrugated iron	0.205	0.13
Roofs	Tar roof	0.92	0.13
Natural	Forests	0.97	0.15
	Water	0.97	0.5

Abbildung 4.6: Liste der Albedowerte von verschiedenen Stoffen [Oke02]

4.3 Zusammenfassung

Die prozedurale Synthese erlaubt es, mittels eines Algorithmus, Inhalte zu generieren. So ist es möglich einen Algorithmus zu entwickeln, der Vegetationen anhand von Informationen (Maps) und Regeln auf einem Terrain platziert. Eine Entscheidungshilfe über die Abhängigkeit des Pflanzenwachstums bietet die ökophysiologische Geobotanik. Diese definiert abiotische Standortfaktoren und wie diese ermittelt werden können.

5 Anwendungsgebiete

Die prozedurale Synthese findet überall da Anwendung wo große Mengen an Inhalten erzeugt werden sollen, die durch Handarbeit von Menschen zu viele Ressourcen kosten würden. Vor allem in der Videospielbranche finden sich viele Anwendungsgebiete der PCG. Aber auch die Film- und Medizinbranche können sich dieser Technik bedienen.

5.1 Videospiele

In der Videospielindustrie wird die prozedurale Generierung häufig verwendet, um mehr Inhalt in dem Produkt anbieten zu können. Durch den wirtschaftlichen Wettbewerb versuchen die Spieleentwickler mehr zu bieten als die Konkurrenz. Zum Beispiel wird oft die Größe der Spielwelt als besonderes Leistungsmerkmal (auch als Feature bezeichnet) aufgeführt. Vor allem Rollenspiele versuchen mit noch nie zuvor gesehener Spielweltgröße zu werben. Um aber dieses Feature erschaffen zu können, müssen prozedurale Algorithmen eingesetzt werden, da dieselbe Arbeit mit Grafik-Designern zu stemmen schnell das Budget für einen Titel übersteigen würde. Die eingesetzten Algorithmen müssen das Terrain generieren und die Objekte auf diesem platzieren. Zu den Objekten gehören die Flora, Fauna und von Menschen geschaffene Artefakte wie Straßen, Häuser, Zäune etc. Neben der Erschaffung von mehr Content kann die prozedurale Generierung auch als Spielmechanik dienen. So ist es bei Sandbox-Spielen wie *Minecraft* (2009) [Moj09] oder *No man's Sky* (2016) [Gam16] Teil des Gameplays neue Welten generieren zu lassen und diese zu erforschen.



Abbildung 5.1: Minecraft (links [Min]) und No man's sky (rechts [NoM]) lassen ihre Spielwelten komplett prozedural erzeugen.

5.1.1 Vegetation

Die prozedurale Generierung von Vegetation findet ihre Wurzeln bereits 1968 durch die Forscher Prusinkiewicz und Lindenmayer. Diese entwickelten erste mathematische Modelle zur Beschreibung von Strukturen in der Natur, im speziellen Blumen. In ihrem Werk *The*

Algorithmic Beauty of Plants (1990) [PL12] beschreiben sie das sogenannte L-System (Lindenmayer System). Dieses System ist ein Ersetzungssystem, d.h. dass ausgehend aus einem Anfangswort alle Buchstaben anhand verschiedener Ersetzungsregeln ausgetauscht werden können. Beispielsweise kann es die Regeln $a \rightarrow ab$ und $b \rightarrow a$ geben. Die Abbildung 5.2 veranschaulicht eine Reihe von Ersetzungen auf der Basis des Anfangszeichens b und den oben genannten Regeln.



Abbildung 5.2: Das Anfangszeichen b wird mehrmals anhand von Ersetzungsregeln ausgetauscht. [PL12]

Um solche erzeugten Zeichenketten zu visualisieren, wird das Turtle Model herangezogen, das von der funktionalen Programmiersprache LOGO verwendet wurde, um einfache Grafiken zeichnen zu können. Anhand einer Kette von Befehlen kann einem Stift (Turtle) vorgeschrieben werden wie ein Bild anhand von Linien gezeichnet werden soll. Beispielsweise bedeutet die Zeichenkette $F+F-F$: *male eine Linie (F für Forward), drehe dich 90° nach Rechts (+), male eine Linie (F), drehe dich 90° nach Links (-) und male eine Linie (F)*. Die Kombination aus L-System und Turtle Model erzeugt Strukturen, die denen von Pflanzen ähneln. Abbildung 5.3 zeigt drei L-Systeme und ihre Turtle-Grafiken.

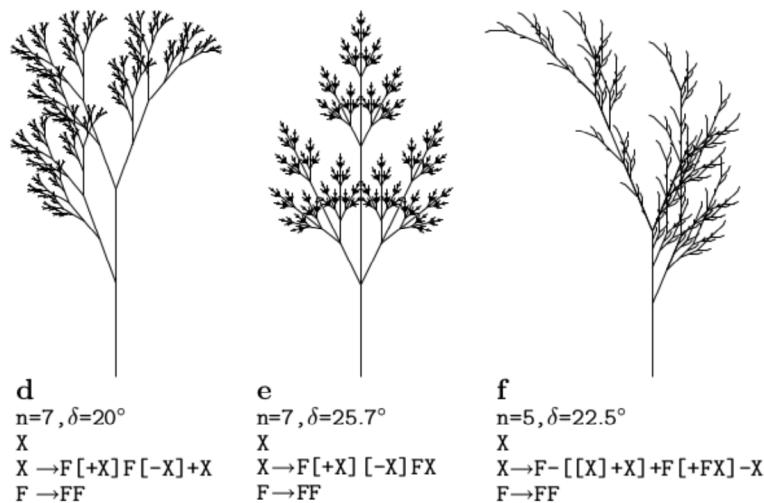


Abbildung 5.3: Drei L-Systeme und ihre Turtle-Grafiken. Ein L-System besteht aus der Anzahl der Ersetzungen (n), dem Winkel für Drehungen (δ), einem Anfangsbuchstaben (X) und den Ersetzungsregeln (z.B. $F \rightarrow FF$).[PL12]

5.1.2 Terrain

Die einfachste Art, Terrain prozedural zu generieren, ist es Heightmaps zu erzeugen. Heightmaps sind Grauwertbilder, d.h. sie besitzen nur einen Kanal, der den Wert der Höhe für einen Pixel widerspiegelt. Anhand der Heightmap kann ein Polygonnetz erzeugt werden. Die Koordinaten der Scheitelpunkte richten sich dann nach den Pixelkoordinaten und dem Wert des Pixels. Abbildung 5.4 zeigt ein Beispiel für eine, mittels Perlin Noise, erzeugte Heightmap, welche dann für die Generierung eines Terrains benutzt wird.

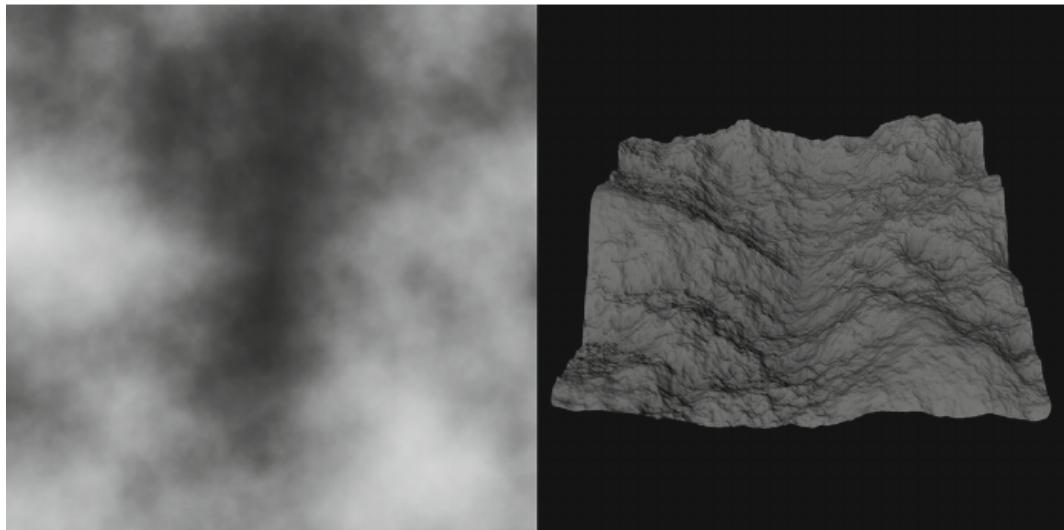


Abbildung 5.4: Erzeugung eines 3D-Terrains anhand eines Grauwertbildes (Heightmap), das mittels Perlin Noise erzeugt wurde. [EL15]

5.2 Filme

In der Filmindustrie werden prozedurale Algorithmen eingesetzt, um riesige virtuelle Filmszenen zu generieren. Die Software Speedtree kommt dabei häufig zum Einsatz. Sie erlaubt das Generieren hochwertiger realistischer Vegetationen. Einsatz fand die Software in Filmen wie *Avatar* (2009), *The Great Gatsby* (2013) und *The Wolf of Wall Street* (2014) [speb].



Abbildung 5.5: Das Intro zu Avatar zeigt eine riesige Landschaft, die mit Speedtree Vegetationen gefüllt wurde. [Spea]

5.3 Medizin

Duffy und Wang [DUF15] haben in ihrem Paper *Application of Procedural Generation as a Medical Training Tool* eine Möglichkeit aufgezeigt prozedurale Generierung zu benutzen, um medizinische Trainingssimulationen zu erzeugen. Dabei werden Patientendaten auf Basis einer zugrunde liegenden Krankheit erzeugt. Angehende Ärzte können dann anhand der Patientendaten die Diagnosestellung üben.

5.4 Zusammenfassung

Das größte Anwendungsfeld der PCG findet sich in der Videospielbranche. In dieser werden Terrains, Vegetationen, Levels und Modelle prozedural generiert. Ferner findet die prozedurale Synthese auch in der Filmindustrie Anklang, indem Szenen mit riesigen, virtuellen Welten prozedural erzeugt werden müssen, da eine händische Erzeugung durch Künstler extrem viel Ressourcen kosten würde. Des Weiteren sind auch Einsätze in weiteren Branchen möglich wie in der Medizin, um Trainingsdaten zu erzeugen.

6 State-of-the-Art

In der Videospielindustrie kommen eine Vielzahl von prozeduralen Systemen zum Einsatz, um Terrains mit Objekten, im speziellen Vegetation, zu befüllen. Die drei wichtigsten Vertreter werden im Folgenden beleuchtet.

6.1 GeNa

GeNa ist Teil der Terrain-Generierungs-Software namens *Gaia* [GeN]. Wobei *GeNa* das generierte Terrain mit Objekten wie Vegetation, Steinen, Gebäuden etc. prozedural befüllt. Beide Softwareprogramme sind Erweiterungen für die Spiel-Engine¹ *Unity 3D* und werden von der Firma *Procedural Worlds* entwickelt und verkauft. Anhand von benutzerdefinierter Regeln werden Assets in der Welt platziert. Dabei achtet die Software darauf, wo der User bestimmte Objekte platziert und versucht anschließend, weitere dieser Objekte an ähnlichen Stellen auf dem Terrain zu platzieren. Abbildung 6.1 zeigt wie ein User einen Felsen auf einem Terrain platziert und anschließend *GeNa* an ähnlichen Stellen weitere Felsen hinzufügt.

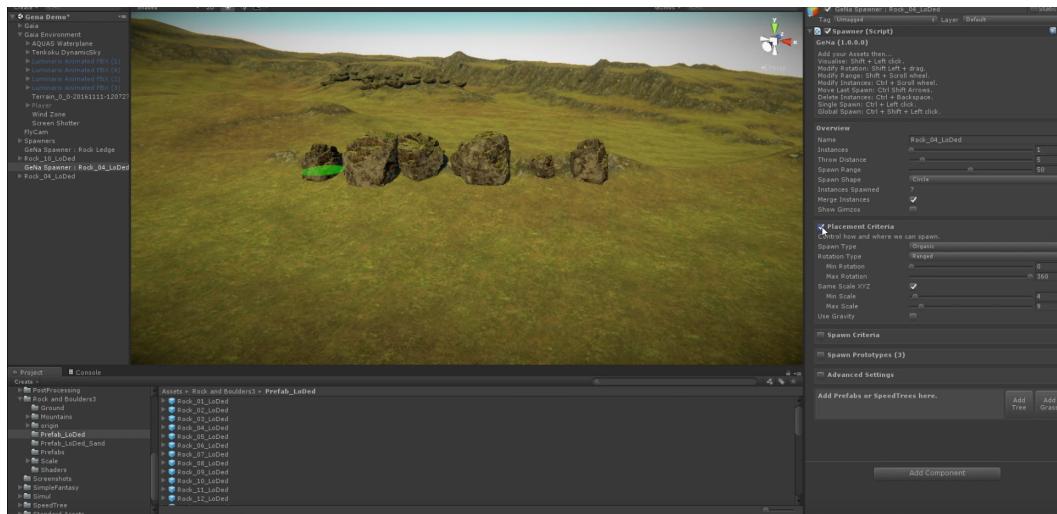


Abbildung 6.1: Ein User platziert einen Felsen in Unity 3D und lässt GeNa anschließend weitere prozedural hinzufügen. [GeN]

Bei der Platzierung der Objekte werden folgende Kriterien beachtet:

- Kollision
- Höhe
- Steigung

¹Spiel-Engines sind Frameworks zur Steuerung des Spielablaufs, Darstellung der Grafik und zum Management der Spielressourcen.

- Textur
- Maskeninformationen (vom User vorgegebene Schablonen, die definieren wo auf dem Terrain etwas wachsen darf) [GeN]

6.2 Unreal Engine 4: Procedural Foliage Tool

Die Spiel-Engine *Unreal Engine 4* bietet zum prozeduralen Erzeugen von Vegetation auf einem Terrain das *procedural foliage tool* an². Dazu wird eine Vegetationsart ausgewählt und dazu die Parameter Kollisionsradius, Schattenradius, Streudichte, Streuradius und Anzahl der Simulationsschritte definiert. Die Engine führt dann eine Wachstums- und Ausbreitungssimulation durch. Dabei breitet sich die Pflanzenart Schritt für Schritt über das Terrain aus und verdrängt dabei auch andere Vegetationen, falls diese in den Schattenradius geraten. Das Ergebnis wird daher sehr organisch und realistisch. Im Folgenden sind die Faktoren der Simulation aufgelistet:

- Kollision
- Schattenradius
- Clustering (Wachstums- und Verbreitungssimulation)
 - Anzahl Iterationen (eine Iteration = ein Jahr)
 - Initiale Samendichte
 - Ausbreitungsdistanz
 - neue Samen pro Durchlauf
- Wachstum
 - Kann im Schatten wachsen (ja/nein)
 - Maximales Alter
 - Prozedurale Skalierung

Abbildung 6.2 zeigt die initiale Platzierung einiger Pflanzen und Abbildung 6.3 zeigt die anschließende Wachstums- und Verbreitungssimulation.

²<https://docs.unrealengine.com/en-US/Engine/OpenWorldTools/ProceduralFoliage/QuickStart/index.html>

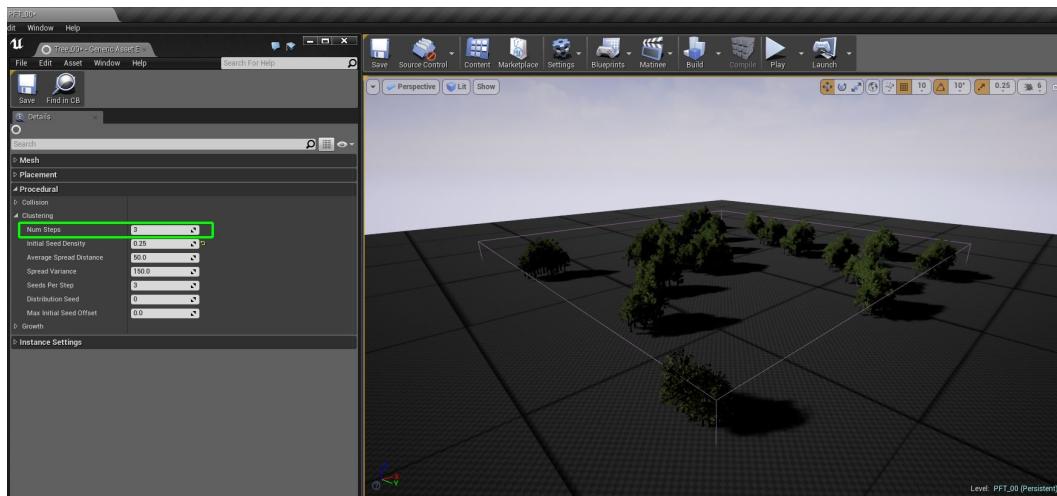


Abbildung 6.2: Mit dem Unreal Engine Procedural Foliage Tool werden zu Beginn ein paar Pflanzen gesetzt. [Unr]

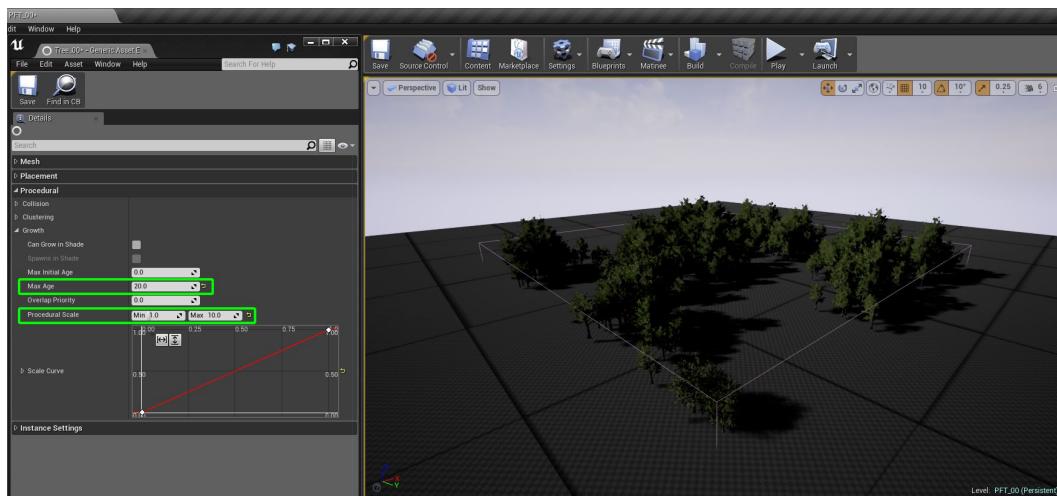


Abbildung 6.3: Anschließend wird das Wachstum und die Verbreitung der Pflanzen simuliert. [Unr]

6.3 Horizon: Zero Dawn

Das Videospiel *Horizon: Zero Dawn* aus dem Jahr 2017 vom Entwickler *Guerrilla Games* hat in vielerlei Hinsicht neue Meilensteine gesetzt. So auch in der Echtzeit-Prozedural-Generierung. Bei der Entwicklung des Spiels stellten die Entwickler früh fest, dass die gewünschte Detail-Dichte der Spielwelt zu hoch war und zu viel Speicherbedarf kosten würde³. Die Lösung war die prozedurale Platzierung der Spielwelt-Objekte während der Spielzeit. Während die oben genannten Beispiele der prozeduralen Platzierung alle offline, d.h. während der Entwicklung des Spiels, geschehen, werden in *Horizon* die Objekte während des Spielens platziert. Dies stellt besondere Ansprüche an die Performance der Platzierungsalgorithmen, da diese so schnell geschehen müssen, dass ein Spieler davon nichts bemerkt.

³<https://www.guerrilla-games.com/read/gpu-based-procedural-placement-in-horizon-zero-dawn>

Die Algorithmen laufen deshalb auf der GPU⁴. Der Platzierungsprozess ist in zwei Phasen geteilt: 1. die Berechnung der Wahrscheinlichkeitswolken und 2. die Diskretisierung zu einer Punktfolge. Der Algorithmus bedient sich folgender Maps, um die Platzierung zu bestimmen:

- Höhenfeld Map
- Erosions Map
- Masken für Gewässer und Flüsse
- Masken (Zur Festlegung wo was wachsen darf)

Abbildung 6.4 zeigt eine Szene aus dem Spiel in dem das System zur prozeduralen Platzierung deaktiviert ist und Abbildung 6.5 zeigt die selbe Szene mit aktiver prozeduraler Platzierung. Die Umschaltung bedarf weniger als eine Sekunde⁵.



Abbildung 6.4: Horizon: Zero Dawn ohne Echtzeit-Platzierung. [Hor]



Abbildung 6.5: Horizon: Zero Dawn nach Echtzeit-Platzierung in derselben Szene. [Hor]

6.4 Weitere Lösungen

6.4.1 Vegetation Studio

Vegetation Studio ist ein Tool für Unity 3D und platziert eine Vegetation anhand definierter Regeln (siehe Abbildung 6.6) wie Höhe über dem Meer, Steilheit des Terrains, Perlin Noise (siehe Unterabschnitt 4.1.1), Rotation, Skalierung und Textur-basierte Regeln (Wachstum nur auf bestimmten Texturen).

⁴Eine GPU (graphics processing unit) ist ein Grafikprozessor, der Berechnungen mit Hilfe von vielen einfachen Prozessoren aufteilt und parallel berechnet. Meist werden GPUs zur Grafikberechnung verwendet.

⁵<https://www.guerrilla-games.com/read/gpu-based-procedural-placement-in-horizon-zero-dawn>

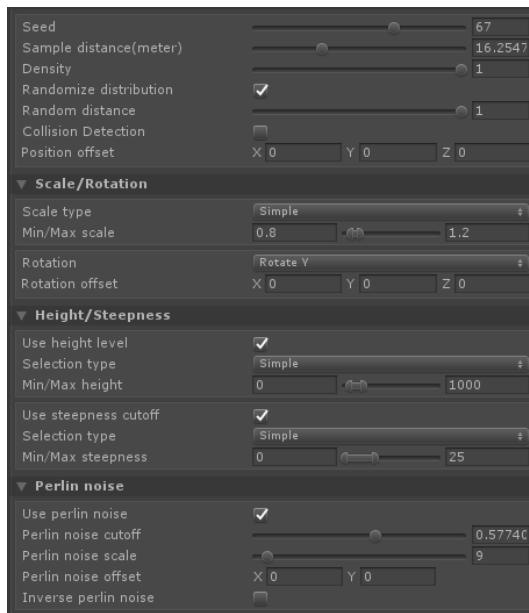


Abbildung 6.6: Einstellbare Regeln für den Platzierungsalgorithmus von Vegetation Studio.
[veg]

6.4.2 Farcry 5

Bei der Entwicklung von Farcry 5 wurden prozedurale Algorithmen verwendet, um die Flora der Spielwelt zu generieren. Die zugrundeliegenden Daten für die Generierung waren dabei Höhenfeld-, Biom- und Textur-Maps. Die Biom-Map gibt das Makroklima eines Gebietes vor, jedoch wird das tatsächliche Mikroklima (Sub-Biom) von dem Höhenfeld und den daraus berechneten Maps (Occlusion, Flow, Slope, Curvature und Illumination [siehe Abbildung 6.7])⁶ bestimmt. Zum Beispiel ist für ein Gebiet das Biom *Mountain* vorgegeben und an den Stellen, wo das Relief Baumwachstum zulässt entsteht das Sub-Biom *Mountain Forest* und an den anderen Stellen (ohne Baumwachstum) entsteht das Sub-Biom *Mountain Grass*. Die Platzierung von Pflanzen hängt vom Sub-Biom ab.

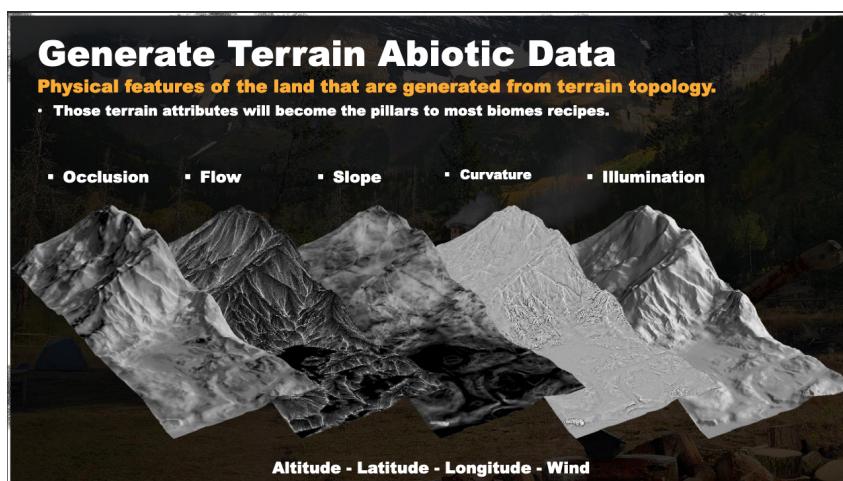


Abbildung 6.7: Die Farcry 5 Engine berechnet auf Basis eines Höhenfeldes verschiedene Maps, welche dann zur Bestimmung des Sub-Bioms verwendet werden. [far]

⁶<https://twvideo01.ubm-us.net/o1/vault/gdc2018/presentations/ProceduralWorldGeneration.pdf>

6.5 Zusammenfassung

Prozedurale Platzierungsalgorithmen laufen meistens während der Entwicklung von Videospielen. Das Ergebnis wird dann gespeichert und beim Spielen geladen. Jedoch hat das Videospiel Horizon: Zero Dawn gezeigt, dass es zum einen Sinn machen kann, den Algorithmus während der Spielzeit laufen zu lassen (um z.B. eine höhere Objektdichte zu erreichen) und zum anderen, dass die Performance eines prozeduralen Systems so gut sein kann, dass diese zur Laufzeit ohne Stockungen in der Framerate ausgeführt werden kann.

7 Abgrenzung

In diesem Kapitel werden die Softwarelösungen aus Kapitel 6 anhand ausgewählter Leistungsmerkmale verglichen und aufgezeigt welche Anforderungen aus Kapitel 3 von keiner Softwarelösung erfüllt werden können.

7.1 Gewählte Merkmale

Um die verschiedenen Systeme der prozeduralen Vegetationsverteilung zu vergleichen, müssen zuerst Merkmale für den Vergleich bestimmt werden. Anhand der gewählten Leistungsmerkmale wird dann ein Vergleich durchgeführt und anschließend beleuchtet, welche Anforderungen aus Kapitel 3 fehlen. Im Folgenden werden die gewählten Leistungsmerkmale erläutert, die zum Vergleich der Softwarelösungen aus Kapitel 6 herangezogen wurden.

1. Masken: Der Einsatz von Maskenbildern, um die Möglichkeit des Wachstums verschiedener Vegetationen einzuschränken, z.B. Bodentexturen, Straßen, Gewässer.
2. Steigung: Berücksichtigung der Steigung an einem Punkt der Karte. Maximale Steigung für das Wachstum einer Vegetationsart.
3. Biom: Das Biom definiert die Makro- oder Mikroklimazone, welche wiederum bestimmt, welche Vegetationen wachsen können.
4. Ausbreitungssimulation: Die Simulation der Ausbreitung von Pflanzen über ein Terrain im Laufe der Zeit.
5. Sonneneinstrahlung: Beachtung der Sonnenbestrahlung (welche Gebiete liegen im Schatten, welche in der Sonne).
6. Bodentiefe: Wird die Bodentiefe berechnet und mit der benötigten Bodentiefe einer Pflanze verglichen?
7. Bodenfeuchtigkeit: Die Berechnung des Wasservorkommens im Boden, abhängig von der Regenmenge, Grundwasser, Verdunstung etc..

7.2 Übersicht

Die Tabelle 7.1 zeigt, welche Software welche Merkmale erfüllt. Dabei fällt auf, dass das *procedural foliage tool* der Unreal Engine 5 als Einzige eine Simulation ausführt, um die Ausbreitung der Pflanzen über die Zeit nachzubilden. Allerdings erfüllt das Tool keines der restlichen Merkmale. Die meisten Leistungsmerkmale werden von Far cry 5 erfüllt. Neben dem Einbeziehen von Masken, Steigung und Biom berücksichtigt die Engine den abiotischen

	GeNa	Unreal Engine 5: Procedural Foliage Tool	Horizon: Zero Dawn	Vegetation Studio	Farcry 5
Masken/Boden-information	ja	nein	ja	ja	ja
Relief/Steigung	ja	nein	ja	ja	ja
Biom	nein	nein	ja	nein	ja
Ausbreitungs-simulation	nein	ja	nein	nein	nein
Sonnen-einstrahlung	nein	nein	nein	nein	ja
Bodentiefe	nein	nein	nein	nein	nein
Bodenfeuchtigkeit	nein	nein	nein	nein	nein

Tabelle 7.1: Übersicht der Leistungsmerkmale aller vorgestellten Softwarelösungen.

Standortfaktor Sonneneinstrahlung, welche von der Engine berechnet wird. Jedoch werden die Faktoren Bodentiefe und Bodenfeuchtigkeit nicht berücksichtigt.

7.3 Auswertung

Alle vorgestellten Lösungen definieren keine Standortfaktoren der Pflanzen. GeNa, Unreal Engine 5, Vegetation Studio und Horizon verwenden lediglich Masken, um festzulegen, wo welche Pflanzen wachsen dürfen. Farcry 5 bestimmt Mikrobiome und das Biom bestimmt, welche Pflanzen darin wachsen können. Das Definieren der Standortfaktoren und die Berechnung der vorhandenen Faktoren auf einer Karte hat den Vorteil, dass das Wachstum der Pflanzen frei ist und prinzipiell jede Pflanze auf einer Karte wachsen kann, sofern die benötigten Faktoren auf irgendeiner Weise vorhanden sind. Dies kommt der Realität näher und könnte mehr Abwechslung und organischere Ergebnisse erzeugen. So wäre es beispielsweise möglich, dass bei angemessenen Bedingungen Kiwis in den Alpen wachsen können [HL02, S. 177]. Die abiotischen Faktoren Sonneneinstrahlung, Bodentiefe und Wassermenge müssen in einem solchen System pro Vegetationsart definiert und aus den vorhandenen Daten (wie Höhenfeld und Bodentexturen) berechnet werden.

8 Konzept

Um die Wahrscheinlichkeit für das Wachstum einer Vegetationsart zu einem Punkt auf einer Karte zu bestimmen, muss zuerst festgelegt werden, welche Bedürfnisse eine Vegetation hat, um daraufhin zu überprüfen, inwiefern diese Bedürfnisse für einen Standort gedeckt werden können. Jedoch werden die vorhandenen Standortfaktoren nicht einfach vorgegeben. Lediglich die *Höhenwerte* und die *Bodenarten* einer Map stehen zur Verfügung. Daher müssen alle weiteren benötigten Informationen berechnet werden.

8.1 Daten

8.1.1 Vegetationsart

Wie im Unterabschnitt 4.2.1 beschrieben, sind die Hauptbedürfnisse einer Pflanze die abiotischen Faktoren Sonnenbestrahlung, Boden und Wasser. Deshalb wird in der Applikation von dieser Arbeit eine Pflanze durch folgende Daten charakterisiert:

- Menge der benötigten Energie (in Kilokalorien) durch Sonneneinstrahlung pro Tag
- Menge des benötigten Wassers (in Liter) pro Tag
- Benötigte Bodenart
- Benötigte Bodentiefe (in cm)

Die Bodenart wird anhand eines Bildes vorgegeben, welches aus nur einem Kanal besteht (Grauwertbild), sodass jeder Pixel nur die ID von dessen Bodenart enthält. Dies macht es notwendig, die Bodenarten und deren IDs festzulegen. Diese IDs müssen mit den IDs in der Bodenart-Textur übereinstimmen.

8.1.2 Bodenart

Die Bodenart wird anhand folgender Merkmale beschrieben:

- ID
- Wasserabsorption
- Albedowert

Die ID ist, wie im Unterabschnitt 8.1.1 erklärt, für die Zuordnung der in dem Bodenart-Bild vorzufindenden IDs notwendig. Der Wasserabsorptionswert gibt an wie viel Prozent der Boden durch Regen und Grundwasser aufnehmen kann. Ferner gibt es einen Albedowert für jede Bodenart (siehe Unterabschnitt 4.2.1).

8.1.3 Biom

Um die Werte für die Sonnenbestrahlung und das Wasservorkommen berechnen zu können, ist es notwendig, die Klimazone für die Map zu bestimmen. Das Biom beschreibt dazu die folgenden Eigenschaften:

- Atmosphärische Absorption (Prozentzahl zwischen eins und null)
- Atmosphärische Diffusion (Prozentzahl zwischen eins und null)
- Wolkenreflektion (Prozentzahl zwischen eins und null)
- Durchschnittliche Regenmenge (in Liter) pro Tag
- Grundwasser (in Liter) pro Kubikmeter

Die atmosphärische Absorption ist ein Prozentwert, der die Lichtabsorption durch die Atmosphäre widerspiegelt. Die atmosphärische Diffusion ist ebenfalls ein Prozentwert und gibt den Teil des Streulichts an, der durch die Atmosphäre abgelenkt und ins Weltall zurückgeführt wird. Die Wolkenreflektion ist der Prozentanteil des Lichts, welcher durch die Wolken ebenfalls ins Weltall zurück gelenkt wird. Die Regenmenge und das Grundwasser dienen als Grundlage für die Berechnung der tatsächlichen Wassermenge an jedem Punkt.

8.1.4 Map

Die Karte (auch Map genannt) ist ein zweidimensionaler Abschnitt eines Terrains, für das berechnet werden soll, an welchen Punkten auf dem Terrain eine Vegetationsart wachsen kann. Dazu sind folgende Informationen notwendig, um eine Map zu bestimmen:

- Name
- Maximale Bodentiefe
- Biom
- Höhenkonversionszahl
- Pixelgröße
- Pfad zur Höhenfeld-Map
- Pfad zur Bodenart-Map

Der Name ist notwendig, um zu bestimmen, für welche Karte die Berechnungen stattfinden sollen. Die maximale Bodentiefe wird für das Bodentiefe-Bild (siehe Abschnitt 8.3) benötigt. Das Biom ist, wie oben beschrieben, notwendig, um die Sonnenbestrahlung und die Wassermenge zu berechnen. Die Höhenkonversion gibt an, wie ein gegebener Höhenwert aus der Höhenfeldkarte umgerechnet werden muss, um den realen Höhenwert in Meter zu erhalten. Der Wert gibt also an, mit welcher Genauigkeit die Höhe abgetastet wurde, um aus einem Terrain das Höhenfeldbild zu erzeugen. Entspricht der Konversionswert beispielsweise 0.1 so bedeutet dies, dass die Höhe mit einer Genauigkeit von 0.1 Meter, also 10 cm, abgetastet wurde. Ähnlich verhält es sich mit der Pixelgröße. Diese gibt an, mit welchem Abstand das Terrain abgetastet wurde, um die Heightmap zu erhalten. Falls die Pixelgröße beispielsweise

100m beträgt, wurde das Terrain alle 100m abgetastet. Der Höhenkonversionswert und die Pixelgröße sind für die Sonnenbestrahlung und für die Berechnung des Reliefs erforderlich (siehe weiter unten).

8.2 Sonnenbestrahlung

Der Ausgangswert für die Sonnenbestrahlung an jedem Punkt auf der Map bildet die Solarkonstante. Diese beträgt ca. 2 Kalorien/cm² pro Minute [Bha14, Kapitel 2]. Auf den Meter hochgerechnet, sind das 20 Kilokalorien pro Minute, pro Stunde also 1200 Kilokalorien pro Quadratmeter. Die Berücksichtigung der Solarkonstante erfüllt die Anforderung 1.a aus Kapitel 3.

8.2.1 Atmosphäre und Albedo

Allerdings ist dies nicht die tatsächliche Menge Energie, die auf dem Boden ankommt, da das Licht von Wolken reflektiert und von der Atmosphäre absorbiert oder gestreut wird (siehe Unterabschnitt 4.2.1). Zudem reflektiert der Boden einen Teil davon zurück ins All. Die Prozentwerte zu diesen Faktoren werden über das Biom bestimmt (siehe Unterabschnitt 8.1.3).

Das Licht, das den Erdboden erreicht, wird zum Teil reflektiert. Der Reflektionsanteil wird durch den Albedowert bestimmt. Abbildung 4.6 zeigt eine Übersicht von verschiedenen Albedowerten. Ein Teil der reflektierten Sonnenstrahlen wird an die angrenzenden Flächen abgegeben. Die Berücksichtigung der Atmosphäre und Bodenreflektion erfüllt die Anforderungen 1.b bis 1.f aus Kapitel 3.

8.2.2 Sonnenstand und Sonnenwanderung

Der obige Abschnitt beschreibt die Berechnung der Sonnenenergie an einem Punkt, sofern der Sonnenstrahl diesen Punkt erreicht. Dies hängt vom Stand der Sonne und vom Gelände ab. Der Sonnenstand wird durch Azimut und Elevation beschrieben. Azimut ist eine Gradzahl zwischen null und dreihundertsechzig Grad und definiert in welcher horizontalen Richtung die Sonne steht. Elevation gibt die vertikale Höhe des Sonnenstands an. Dieser Wert liegt zwischen null und neunzig Grad. Abbildung 8.1 veranschaulicht die Definition des Sonnenstands durch Azimut und Elevation.

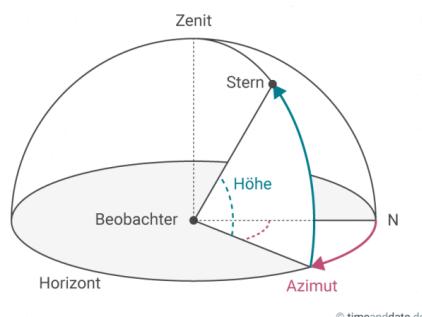


Abbildung 8.1: Der Sonnenstand wird durch Azimut und Elevation (Höhe) definiert. [Azi]

Wenn zwischen einem Punkt auf der Map und der Sonne Gelände liegt, das höher ist und somit den Sonnenstrahl blockiert, dann erhält dieser Punkt kein Sonnenlicht und somit keine Energie. Allerdings bezieht sich dies auf den Sonnenstand zu einer bestimmten Stunde. Im Laufe des Tages bewegt sich jedoch die Sonne und der Punkt könnte dann wieder Licht erhalten. In der Applikation muss deshalb angegeben werden wie viele Sonnenstunden es gibt. Danach wird für jede Stunde der Sonnenstand ermittelt und anschließend für jeden Punkt der Karte berechnet, ob dieser Sonnenlicht erhält. Um ein Ergebnis zu erhalten, wird der Richtungsvektor aus Azimut und Elevation berechnet. Der Richtungsvektor gibt also an, in welcher Richtung die Sonne steht. Das Bild 8.2 zeigt die Abbildung von Polarkoordinaten in kartesische Koordinaten.

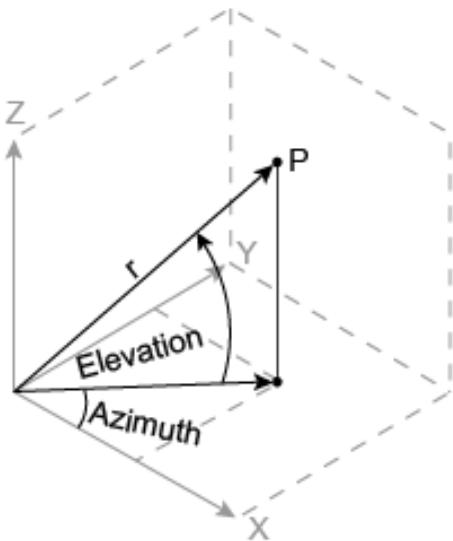


Abbildung 8.2: Die Abbildung von Polarkoordinaten zu kartesischen Koordinaten. [Pol]

Mathematisch wird die Umrechnung durch die Formel 8.1 durchgeführt.

$$\begin{aligned} x &= r * \cos(elevation) * \cos(azimuth) \\ y &= r * \cos(elevation) * \sin(azimuth) \\ z &= r * \sin(elevation) \end{aligned} \quad (8.1)$$

Um festzustellen, ob ein Punkt von einem Sonnenstrahl erreicht wird, wird dem Richtungsvektor gefolgt und an jedem Punkt die momentane Höhe des Lichtstrahls mit der Höhe der Heightmap verglichen. Falls der Wert des Höhenfeldes höher liegt, bedeutet dies, dass der Anfangspunkt zu diesem Zeitpunkt nicht beleuchtet wird und im Schatten liegt. Liegt der Höhenwert darunter, so wird dem Lichtstrahl solange gefolgt, bis entweder das Terrain den Strahl blockiert, der Strahl die Map verlässt oder der Höhenwert des Lichtstrahls höher liegt als der höchste Punkt der Höhenmap. Tritt eine der letzten beiden erwähnten Bedingungen ein, so erhält der Anfangspunkt Licht, also Energie in der berechneten Höhe, wie weiter oben beschrieben. Die Berücksichtigung des Sonnenstandes erfüllt die Anforderung 1.g aus Kapitel 3.

8.3 Relief

Um die Bodentiefe zu ermitteln, ist es nötig, das Relief (auch als Normalmap bezeichnet) zu berechnen. Jeder Punkt auf dem berechneten Reliefbild entspricht dann dem Richtungsvektor an diesem Punkt. Der Richtungsvektor gibt an, in welche Richtung die Fläche des Terrains an diesem Punkt zeigt. Ist das Terrain an einem Punkt eben, so ist der Richtungsvektor wie in der Formel 8.2 dargestellt.

$$\begin{bmatrix} x = 0 \\ y = 0 \\ z = 1 \end{bmatrix} \quad (8.2)$$

Das gewählte Referenzkoordinatensystem definiert die beiden horizontalen Dimensionen als X- und Y-Achse, und die vertikale Dimension als Z-Achse. Abbildung 8.3 veranschaulicht dieses System.

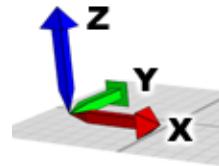


Abbildung 8.3: Das gewählte Koordinatensystem. [coo]

Um die Normalmap zu berechnen, sind zwei Ansätze möglich. Die eine ist einen Sobel-Filter anzuwenden. Die beiden Matrizen in der Gleichung 8.3 werden dabei auf jeden Pixel und dessen Nachbarn angewendet.

$$S_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \quad S_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \quad (8.3)$$

Bei der anderen Variante muss ein Scheitelpunktnetz auf Basis der Heightmap erstellt werden. Dabei ist es wichtig, dass die X- und Y-Koordinaten der Scheitelpunkte nicht den Pixelkoordinaten des Höhenbildes entsprechen, sondern den tatsächlichen Koordinaten des zugrundeliegenden Terrains. Dies liegt darin begründet, dass die Höhenwerte in die tatsächlichen Höhen umgerechnet werden. Für die Scheitelpunktkoordinaten sind deshalb die beiden Werte für die Höhenkonversion und Pixelgröße (siehe Unterabschnitt 8.1.4) notwendig. Mit Hilfe des Scheitelpunktnetzes werden die Normalen aller angrenzenden Flächen eines Scheitelpunkts berechnet, aufsummiert und normalisiert. Das Ergebnis ist die Normale an diesem Punkt auf dem Terrain. Für die Berechnung der Nachbar-Normalen eines Scheitelpunkts werden aus den beiden Nachbar-Scheitelpunkten (Ortsvektoren) beide Richtungsvektoren durch Subtraktion ermittelt. Anschließend wird mittels Kreuzprodukt aus den beiden Richtungsvektoren die Normale berechnet. Abbildung 8.4 zeigt diesen Sachverhalt.

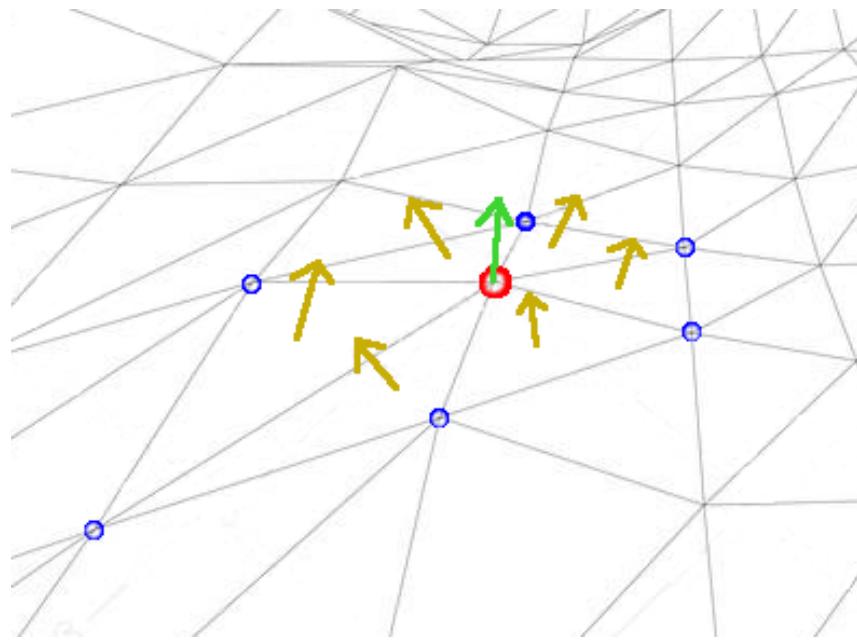


Abbildung 8.4: Die Normale (grün) des ausgewählten Scheitelpunkts (rot) wird über die Summe der Normalen (gelb) seiner Nachbarn (blau) ermittelt. Die Darstellung ist grob und nicht exakt. (Quelle: eigene Darstellung)

Die Berechnung der Normalen wird für jeden Scheitelpunkt wiederholt. Das Ergebnis ähnelt dem aus der Abbildung 8.5 wobei in dem Bild nicht alle Normalen an jedem Scheitelpunkt zu sehen sind. Die Berechnung der Normalmap erfüllt die Anforderung 2 aus Kapitel 3.

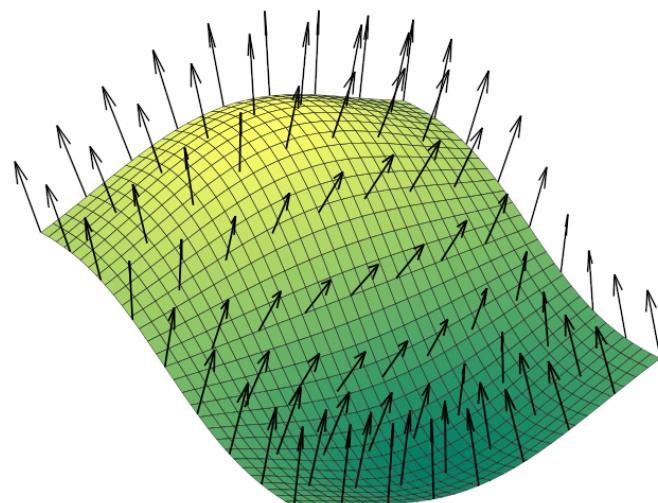


Abbildung 8.5: Normalen Vektoren auf einer 3D-Fläche. [nor]

8.4 Bodentiefe

Die Bodentiefe kann mit Hilfe der Normalmap (siehe Abschnitt 8.3) berechnet werden. Dazu wird die Normale an einem Punkt mit der Normalen, die senkrecht nach oben zeigt (siehe Vektor 8.2), per Punktprodukt verrechnet. Das Ergebnis ist der Winkel zwischen den beiden Vektoren. In der Abbildung 8.6 wird dies verdeutlicht. Je größer der Winkel, desto niedriger die Bodentiefe, da das Gelände an dieser Stelle steil ist.

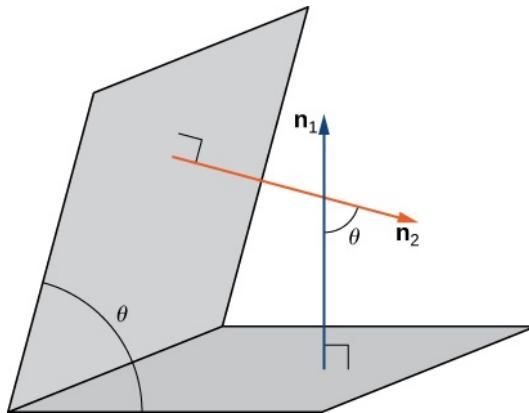


Abbildung 8.6: Der Winkel zwischen zwei Normalen wird mittels Punktprodukt ermittelt.
[ang]

Die Berechnung der Bodentiefe erfüllt die Anforderung 3 aus Kapitel 3.

8.5 Wasservorrat & Verdunstung

Auf Basis der Bodentiefe und der durch das Biom vorgegebenen Regen- und Grundwassermenge wird dann die Wassermenge an jedem Punkt berechnet. Von dieser Grundmenge wird danach die Verdunstungsmenge abgezogen. Die Verdunstungsmenge berechnet sich aus der vorhandenen Energiemenge, der Sonneneinstrahlung und der benötigten Menge Energie, um ein Milliliter Wasser zu verdunsten. Die benötigte Energiemenge liegt bei $2.5 \cdot 10^6 \text{ J kg}^{-1}$ [Ger11, S. 20]. Umgerechnet auf ein Gramm sind das $2.5 \cdot 10^3 \text{ J g}^{-1}$. Ein Gramm Wasser entspricht dem Volumen von einem Milliliter. Die Umrechnung von einer Kilokalorie zu Joule beträgt 4184 [Mec73, S. 11]. Umgekehrt beträgt der Konversionsfaktor von einem Joule zu Kilokalorie demnach $\frac{1}{4184} = 2.39 \cdot 10^{-4}$. Die benötigte Kilokaloriemenge, um ein Milliliter Wasser zu verdampfen, ist demnach $2.5 \cdot 10^3 \frac{\text{J}}{\text{ml}} \cdot 2.39 \cdot 10^{-4} \frac{\text{kcal}}{\text{J}} = 0.54 \frac{\text{kcal}}{\text{ml}}$. Dieser Wert wird mit der vorhandenen Energiemenge pro Pixel verrechnet, um die Menge an Wasser zu ermitteln, die theoretisch verdampfen kann. Das Ergebnis wird also von der vorhandenen Wassermenge abgezogen. Falls die Verdampfungsmenge größer als die vorhandene Menge ist, wird die Wassermenge auf null gesetzt. Die Berechnung der vorhandenen Wassermenge unter Berücksichtigung der Verdampfung erfüllt die Anforderung 4.a bis 4.e aus Kapitel 3.

8.6 Wachstumswahrscheinlichkeiten

Wenn alle benötigten Informationen berechnet wurden (Energiemenge, Bodentiefe und Wassermenge), können die Wahrscheinlichkeiten ermittelt werden. Für jeden Standortfaktor einer Vegetation wird für jeden Punkt auf der Karte die Wahrscheinlichkeit für das Wachstum berechnet. Anschließend wird die geringste Wahrscheinlichkeit als die endgültige Wahrscheinlichkeit festgelegt. Dies liegt im *Gesetz des Minimums* begründet (siehe Unterabschnitt 4.2.1). Das Gesetz besagt, dass der Standortfaktor einer Pflanze, welcher im geringsten Maße vorhanden ist, das Wachstum einer Pflanze bestimmt. Prinzipiell können die Wahrscheinlichkeiten mit der einfachen Formel $\frac{\text{vorhandeneFaktormenge}}{\text{benötigteFaktormenge}}$ ermittelt werden. Allerdings soll die Wahrscheinlichkeit wieder abnehmen, falls die vorhandene Faktormenge zu groß wird. In dem Fall kann die Formel $1 - (\frac{\text{vorhandeneFaktormenge}}{\text{benötigteFaktormenge}} - 1)$ verwendet werden. Die Ausnahme bildet die Wahrscheinlichkeitsberechnung für die benötigte Bodenart. Diese beträgt entweder 100% oder 0%. Die Berechnung der Wahrscheinlichkeit unter Berücksichtigung der Standortfaktoren einer Vegetation erfüllt die Anforderung 5.a bis 5.d aus Kapitel 3.

8.7 Gesamtsystem

Um die benötigten Informationen zu berechnen, muss zuerst die Sonneneinstrahlung (Insolation) auf Basis der Heightmap berechnet werden. Anschließend wird die Normalmap (Orography) berechnet, welche ebenfalls die Heightmap benötigt. Auf Basis der Normalmap wird danach die Bodentiefe (Edaphology) kalkuliert. Zuletzt findet die Generierung der Wassermapping (Hydrology) statt, welche die Bodentiefe- und die Insolationmap voraussetzt. In der Abbildung 8.7 wird dieser Prozess verdeutlicht.

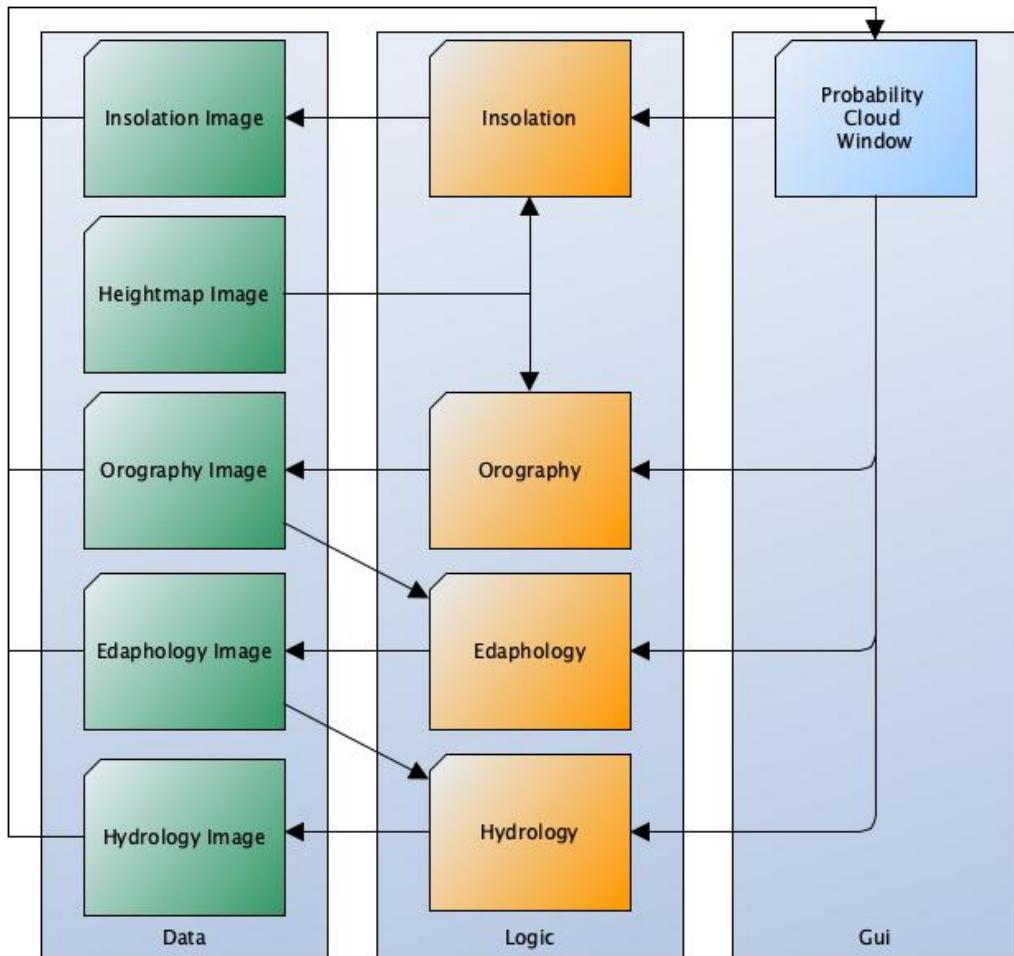


Abbildung 8.7: Ablauf der Berechnungen der benötigten Informationen für die Wahrscheinlichkeitsberechnung. (Quelle: eigene Darstellung)

Wenn alle Informationen vorliegen, können die Wahrscheinlichkeiten einer Pflanze ermittelt werden. Abbildung 8.8 visualisiert diesen Ablauf.

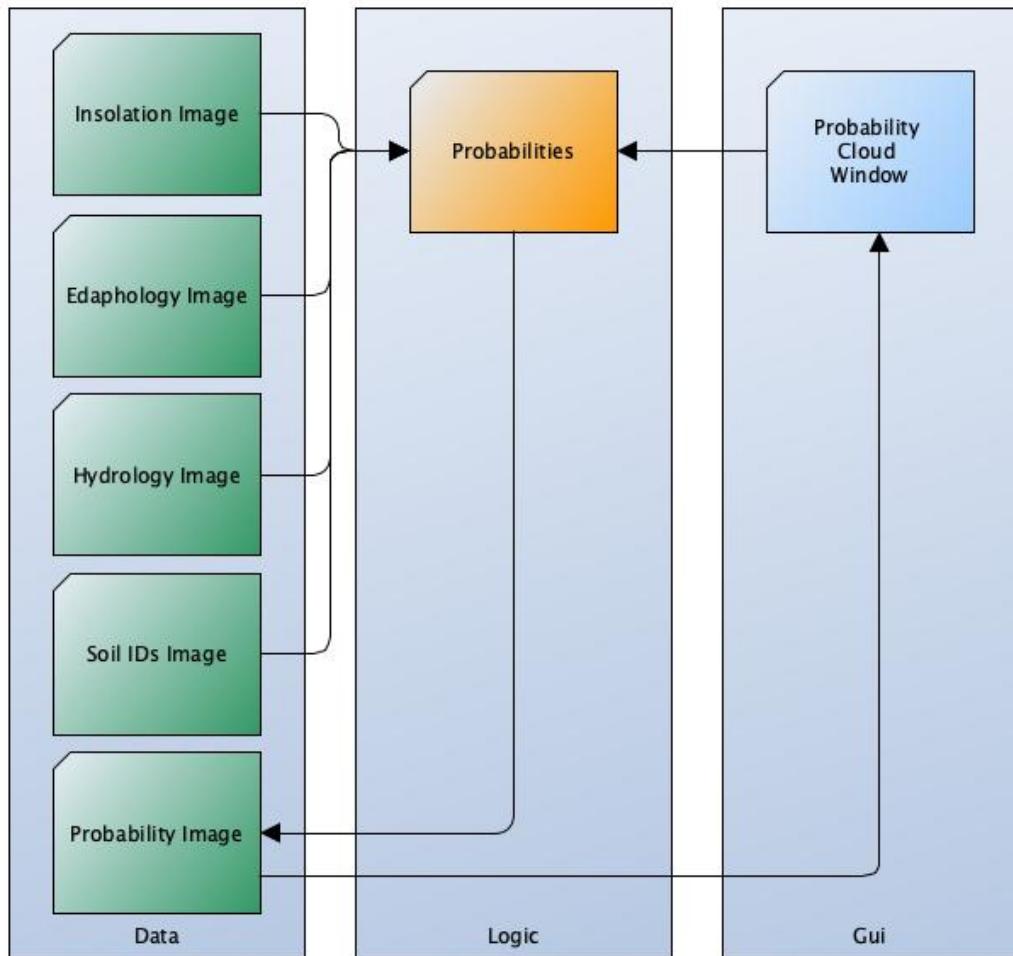


Abbildung 8.8: Ablauf der Berechnungen für die Wachstumswahrscheinlichkeit einer Vegetationsart. (Quelle: eigene Darstellung)

8.8 Zusammenfassung

Die Berechnung der Wahrscheinlichkeit für das Wachstum einer Pflanze an einem Punkt auf einer Karte hängt von der Wassermenge, der Lichteinstrahlung, der Bodenart und -tiefe ab. Diese Werte müssen anhand des Höhenfeldes und der Bodenart-Karte errechnet werden. Die Sonnenbestrahlung wird anhand der Solarkonstante abzüglich der Absorption durch die Atmosphäre berechnet, jedoch erhält ein Punkt auf der Karte nur dann Energie, falls das Terrain zu einer bestimmten Stunde nicht den Lichtstrahl unterbricht. Das Relief wird anhand des Scheitelpunktnetzes ermittelt, indem die Normalen der angrenzenden Flächen von jedem Scheitelpunkt aufsummiert und normalisiert werden. Mit Hilfe des Reliefs werden die Bodentiefe und die Wassermenge bestimmt. Durch Verdampfung wird die Wassermenge reduziert.

9 Implementierung

Die Applikation wurde mit Python implementiert, da in dieser die größte Expertise des Autors liegt. Im Folgenden wird auf die Implementierung der Daten, Logik und GUI eingegangen.

9.1 Data

Für alle Daten, die in Abschnitt 8.1 aufgelistet worden, wurden Klassen implementiert, um die benötigten Informationen zu speichern. Zur dauerhaften Speicherung werden die Daten in Dateien im Yaml-Format abgelegt und beim Start der Applikation geladen. Das Listing 9.1 zeigt die Implementierung der Vegetation-Klasse und enthält neben dem Konstruktor, zum Anlegen der Daten, eine Funktion *save_vegetation* zur persistenten Speicherung der Daten als Yaml-Datei.

Listing 9.1: Source-Code der Vegetation-Klasse.

```

1 class Vegetation:
2     def __init__(self, name, energy_demand, water_demand, soil_demand,
3                  soil_depth_demand):
4         self.name = name
5         self.energy_demand = energy_demand
6         self.water_demand = water_demand
7         self.soil_demand = soil_demand
8         self.soil_depth_demand = soil_depth_demand
9
10    def save_vegetation(self):
11        data = {self.name: {
12            'energy_demand': self.energy_demand,
13            'water_demand': self.water_demand,
14            'soil_demand': self.soil_demand,
15            'soil_depth_demand': self.soil_depth_demand}}
16
17        with open('resources/data/vegetation_types.yml', 'a') as
18            outfile:
19                yaml.dump(data, outfile, default_flow_style=False)

```

9.1.1 Yaml-Dateien

Das Yaml-Format zeichnet sich durch besonders wenig Syntax aus. Datenblöcke werden durch Einrückungen deklariert. Deshalb ist es wichtig, die Einrückungen und Zeilenumbrüche

genau zu beachten. Im Yaml-Format werden alle Informationen als Liste oder assoziative Liste angelegt. Diese können außerdem miteinander kombiniert werden. Das Listing 9.2 zeigt ein Beispiel für eine Yaml-Datei.

Listing 9.2: Beispiel für eine Yaml-Datei.

```

1 # Dies ist ein Kommentar
2 # Es folgt eine Liste
3 - Auto
4 - Katze
5 - Mond
6 # Es folgt eine assoziative Liste
7 name: Bernd
8 alter: 27
9 studium: Feng-Shui-Wissenschaften
10 # Es folgt eine Kombination aus assoziativer Liste und Liste
11 compiler:
12   - c
13   - c++
14   - fortran
15 interpreter:
16   - python
17   - ruby
18   - javascript

```

Die *biom.yml* (siehe Listing 9.3) enthält die Biome, welche in der Applikation angelegt wurden.

Listing 9.3: Auszug aus der bioms.yml Datei.

```

1 PolarZone:
2   atmospheric_absorption: '45'
3   atmospheric_diffusion: '8'
4   avg_rainfall_per_day: '1.5'
5   cloud_reflection: '40'
6   groundwater: '1.0'

```

Das Listing 9.4 zeigt die gespeicherten Information der angelegten Bodenarten.

Listing 9.4: Auszug aus der soil_types.yml Datei.

```

1 Dirt:
2   id: '40'
3   albedo: '0.25'
4   water_absorption: '0.38'

```

Es folgen Beispiel-Auszüge zu den Maps-Informationen (siehe Listing 9.5) und den Vegetationsarten (siehe Listing 9.6).

Listing 9.5: Auszug aus der maps.yml Datei.

```

1 Big_Terrain:
2   biom: TemperateZone
3   height_conversion: 0.1

```

```

4 height_map_path: C:\Users\Thorus\PycharmProjects\Prozedurale-
    Vegetations-Verteilung\resources\maps\16
    bitHeightMap512x512Terrain8km.png
5 max_soil_depth: 150.0
6 pixel_size: 16.0
7 texture_map_path: C:\Users\Thorus\PycharmProjects\Prozedurale-
    Vegetations-Verteilung\resources\maps\16
    bitHeightMap512x512Terrain8km_soil_ids.png

```

Listing 9.6: Auszug aus der vegetations_types.yml Datei.

```

1 Appletree:
2     energy_demand: '2000'
3     soil_demand: Dirt
4     soil_depth_demand: '80'
5     water_demand: '0.7'

```

9.1.2 Sonne

Die Implementierung der Sonne erfolgte nach dem Konzept wie bereits in Unterabschnitt 8.2.2 beschrieben. Neben den Informationen zu Azimut und Elevation gibt es eine Funktion (siehe Listing 9.7) zur Umrechnung in uv-Koordinaten (ebenfalls in Unterabschnitt 8.2.2 beschrieben).

Listing 9.7: Implementierung der Sonne.

```

1 class Sun:
2     def __init__(self, elevation, azimuth):
3         self.elevation = elevation
4         self.azimuth = azimuth
5
6     def convert_to_uv_coordinates(self):
7         u = math.cos(math.radians(self.azimuth)) * math.cos(math.
    radians(self.elevation))
8         v = math.sin(math.radians(self.azimuth)) * math.cos(math.
    radians(self.elevation))
9         w = math.sin(math.radians(self.elevation))
10        u = round(u, 4)
11        v = round(v, 4)
12        w = round(w, 4)
13        return u, v, w

```

9.1.3 Image

Zum Einladen der Informationen aus der Heightmap und der Soil-IDs-Map wurde die Klasse *Image* erstellt. Ferner dient diese Klasse der Speicherung der Ergebnisse, sowohl der Berechnungen, die für die Wahrscheinlichkeit benötigt werden, als auch für die Ergebnisse der Wahrscheinlichkeitsberechnungen selbst. Da die Berechnungen, abhängig von der Bildgröße, sehr viel Zeit in Anspruch nehmen können, ist es sinnvoll, die Zwischenergebnisse zu

speichern. Die so gespeicherten bzw. geladenen Bilder unterteilen sich in zwei Datentypen: Integer und Float. Wobei die Bilder alle als Graubild PNG mit 8- oder 16-Bit-Tiefe gespeichert werden. Die Bilder, die Integer-Werte enthalten (Heightmap, Soil-IDs-Map und Insolation), bedürfen keiner weiteren Nachverarbeitung und können direkt gespeichert werden. Die Float-Bilder (Edaphic, Water und Probability) müssen jedoch zuerst in Integer-Werte umgewandelt und beim Einladen wieder in Float-Werte umgewandelt werden. Dazu werden die Werte beim Speichern mit dem Wert Eintausend multipliziert und beim Laden mit diesem Wert dividiert. Das bedeutet, dass die Float-Werte nach der vierten Nachkommastelle abgeschnitten werden. Die Implementierung wird in Listing 9.8 gezeigt.

Listing 9.8: Implementierung der Speicher- und Ladenfunktion der Image-Klasse.

```

1 def load_image(self, path):
2     self.image = imageio.imread(path)
3     self.size = self.image.shape[0]
4     if self.dtype == np.float:
5         new_image = np.full(shape=(self.size, self.size), fill_value
6                             =0.0, dtype=np.float)
6         for y in range(self.size):
7             for x in range(self.size):
8                 new_image[x][y] = self.image[x][y] / 1000.0
9         self.image = new_image
10    self.is_image_loaded = True
11
12 def save_image(self, path):
13     if not os.path.exists(os.path.dirname(path)):
14         os.makedirs(os.path.dirname(path))
15     if self.dtype == np.float:
16         image = np.zeros(shape=(self.size, self.size), dtype=np.uint16)
17         for y in range(self.size):
18             for x in range(self.size):
19                 image[x][y] = self.image[x][y] * 1000
20         imageio.imwrite(path, image)
21     else:
22         imageio.imwrite(path, self.image)

```

Des Weiteren wurden zwei Funktionen zu Erstellung von Soil-IDs-Maps implementiert. Die Funktion *filter_unique_numbers_from_2d_array* listet alle in einem Bild vorkommenden Werte auf. Auf Basis dieser Information kann dann das Bild mittels der Funktion *transform_image_to_valid_soils* in der Art umgewandelt werden, das es nur IDs enthält, die in der soil-types.yml angelegt wurden.

9.2 Logic

9.2.1 Controller

Das Zentrum der Applikation ist die Controller Klasse. Bei Programmstart lädt der Controller alle Daten aus den Yaml-Dateien. Das Listing 9.9 zeigt beispielsweise die Funktion zum Laden der Vegetationsarten, welche in der *vegetations_types.yml* gespeichert sind.

Listing 9.9: Die Funktion zum Laden der Vegetationsarten.

```

1 def load_vegetations(self):
2     vegetations = {}
3     vegetations_file = Path("resources/data/vegetation_types.yml")
4     if vegetations_file.is_file():
5         with open(vegetations_file, 'r') as stream:
6             try:
7                 vegetations_dict = yaml.safe_load(stream)
8                 if vegetations_dict is not None:
9                     for vegetation_name, vegetation_values in
10                         vegetations_dict.items():
11                             vegetations[vegetation_name] = Vegetation(
12                                 vegetation_name, float(vegetation_values['
13                                     energy_demand']), float(vegetation_values['
14                                     water_demand']), self.soils[
15                                         vegetation_values['soil_demand']], float(
16                                         vegetation_values['soil_depth_demand']))
17             except yaml.YAMLError as exc:
18                 print(exc)
19             self.vegetations = vegetations

```

Nach dem Laden aller angelegten Daten wird die GUI gestartet. Werden interaktive Elemente der GUI bedient, so werden Funktionen aus der Controller Klasse ausgeführt, welche die Funktionen der weiteren Logik-Klassen (Insolation, Orography, Edaphology und Hydrology) startet. Die Controller Klasse enthält die Objekte der Berechnungsklassen und die Objekte der Image-Klasse (Heightmap, Insolationmap etc.). Außerdem gibt es Befehle zur Steuerung der GUI. Zum Beispiel wird die Funktion *prepare_insolation_calculation* (siehe Listing 9.10) beim Klick auf den Button *Calculate Insolation* im Probability Fenster ausgeführt, welche die Berechnung der Insolation durchführt und das Ergebnis in der GUI anzeigt und als PNG speichern lässt.

Listing 9.10: Diese Funktion lässt die Berechnung der Insolation Klasse ausführen und lässt das Ergebnis einerseits in der GUI anzeigen und andererseits als PNG speichern.

```

1 def prepare_insolation_calculation(self, map_name, daylight_hours,
2                                     sun_start_elevation, sun_start_azimuth, sun_max_elevation,
3                                     reflection_coefficient):
4     map = self.maps[map_name]
5     self.image_insolation_map = Image(size=self.image_height_map.size)
6     insolation = Insolation(self)
7     self.image_height_map.load_image(map.height_map_path)
8     self.image_insolation_map = insolation.calculate_actual_insolation(
9         map, daylight_hours, sun_start_elevation, sun_start_azimuth,
10        sun_max_elevation, reflection_coefficient)
11    self.main_window.frames['ProbabilityCloudWindow'].draw_insolation_image(
12        self.image_insolation_map)
13    save_path = "resources/results/" + map_name + "/" + map_name + "_"
14        + str(daylight_hours) + "daylight_hours_insolation_image.png"
15    self.image_insolation_map.save_image(save_path)

```

9.2.2 Insolation

Die Insolation Klasse berechnet für jeden Pixel, wie viel Energie dieser durch Sonneneinstrahlung erhält. Der Prozess unterteilt sich in drei Phasen:

1. Berechnung der rohen Energiemenge pro Pixel (ohne Energieverlust durch die Atmosphäre etc.) pro Stunde
2. Berechnung der tatsächlichen Energiemenge an jedem Pixel (durch Abzug der Absorption der Atmosphäre usw.)
3. Zusätzliche Energiemenge durch Reflektion der Nachbarpixel

Die notwendigen Berechnungen sind sehr umfangreich und nehmen mit größeren Bildern quadratisch zu, sodass die Berechnungen auf einem Prozessorkern für ein 1024 mal 1024 Pixel großes Höhenfeld ca. 3 Stunden benötigen. Da die Berechnungen unabhängig voneinander sind, können diese aufgeteilt und mittels Multithreading¹ oder auf der GPU berechnet werden. Das GPU-Toolkit Cuda² von NVIDIA bietet die Möglichkeit, Funktionen auf der GPU laufen zu lassen. Um dies nutzen zu können, wäre es notwendig die kritischen Funktionen in Shader³ zu refaktorisieren. Allerdings ist der Aufwand dafür sehr groß, da es zu diesem Thema wenig Dokumentation gibt und die Refaktorisierung zu Shadern in diesem Fall nicht trivial gewesen wäre. Dies hat den Autor dazu bewogen die Applikation auf einem Kern laufen zu lassen und die gewonnene Zeit dazu zu nutzen, die Ergebnisse für die großen Höhenfeldbilder berechnen zu lassen.

9.2.3 Tageslichtstunden

Die erste Phase wird durch die Funktion `calculate_insolation_for_daylight_hours` realisiert. In dieser wird zuerst die Winkelwanderung für den Azimut und Elevation der Sonne pro Stunde berechnet. In der GUI ist einstellbar, bei welchem Azimut- und Elevation-Winkel die Sonne startet. Über die `daylight_hours` Variable wird dann berechnet, wie viel Gradzuwachs diese beiden Werte pro Stunde erhalten. Startet die Sonne beispielsweise im Osten (Azimut = 0°) und die Anzahl der Tageslicht-Stunden beträgt dreizehn, so wandert die Sonne pro Stunde um $\frac{180^\circ}{12} = 15^\circ$. Für die Elevation kann zusätzlich ein maximaler Gradwert angegeben werden, der angibt bis zu welchem vertikalen Winkel die Sonne wandert. In der Realität erreicht die Sonne nur im Äquator zur Mittagsstunde die vollen 90°. Weiter südlich oder nördlich sind es weniger und eine noch niedrigere Gradzahl ist im Winter vorzufinden. Über den maximalen Elevationwert können also der Breitengrad und die Jahreszeit simuliert werden. Nur bei 90° erhalten definitiv alle Pixel auf einer Map Sonnenstrahlung. Bleibt der maximale Elevationwert bei standardmäßigen 90° so beträgt der Wert für die stundenmäßige Sonnenwanderung $\frac{90^\circ}{13/2} = 13,85^\circ$. Wobei dieser Wert zuerst bis zur Mittagszeit addiert und danach subtrahiert wird. Das Listing 9.11 zeigt den Algorithmus für diese Berechnungen. Bei dem obigen Beispiel steht demnach in der ersten Tageslichtstunde die Sonne am Horizont im Osten, zur siebten Stunde (Mittag) genau senkrecht über der Karte und in der dreizehnten Stunde im Westen am Horizont. Zudem kann für die Elevation ein Startwert in

¹Multithreading bezeichnet das Aufteilen eines Prozesses in mehrere Threads, welche parallel berechnet werden können.

²<https://developer.nvidia.com/cuda-toolkit>

³Shader sind Programme, die auf GPUs laufen und sich durch Einfachheit, Determinismus und Kürze auszeichnen.

der GUI angegeben werden, wobei der Algorithmus dann so funktioniert, dass zur letzten Tageslichtstunde die anfängliche Elevationgradzahl wieder erreicht wird.

Listing 9.11: Die Berechnung der Winkel, die auf den Azimut- und Elevation-Wert der Sonne pro Stunde addiert bzw. subtrahiert werden.

```

1      # elevation
2      if daylight_hours == 1:
3          elevation_per_hour = 0
4      elif daylight_hours == 2:
5          elevation_per_hour = sun_max_elevation -
6              sun_start_elevation
7      else:
8          if daylight_hours % 2 == 1:
9              elevation_per_hour = ((sun_max_elevation -
10                 sun_start_elevation) / math.ceil(
11                     (daylight_hours - 2) / 2))
12      else:
13          elevation_per_hour = (sun_max_elevation -
14              sun_start_elevation) / (daylight_hours / 2)
15      # azimuth
16      if daylight_hours == 1:
17          azimuth_per_hour = 0
18      elif daylight_hours == 2:
19          azimuth_per_hour = 180 - 2 * sun_start_azimuth
20      else:
21          azimuth_per_hour = 180 / (daylight_hours - 1)

```

Anschließend wird pro Stunde für jeden Pixel die rohe Energiemenge mit der Funktion *calculate_raw_insolation* berechnet. Diese wandelt zuerst den momentanen Sonnenstand (Polarkoordinaten) in einen Richtungsvektor (kartesische Koordinaten) um. Alle folgenden Berechnungen für die Sonnenstrahlposition geschehen im Real space, d.h. die Pixelkoordinaten werden anhand der Pixelgröße in die realen Koordinaten umgerechnet. Dies ist notwendig, da die Höhen aus der Heightmap mit diesen Koordinaten verglichen werden. Anschließend wird in einer Schleife dem Richtungsvektor der Sonne gefolgt, was dem Folgen des Lichtstrahls von einem Punkt zur Sonne gleichen würde. Bei der Verfolgung des Lichtstrahls wird die momentane Höhe der Karte an diesem Punkt mit der Höhe des Lichtstrahls an dem momentanen Punkt verglichen. Ist die Höhe des Lichtstrahls kleiner als die Höhe des Terrains an diesem Punkt wird der Lichtstrahl vom Terrain blockiert und der Pixel erhält zu dieser Stunde kein Sonnenlicht und somit auch keine Energie. Dem Lichtstrahl wird solange gefolgt bis entweder dieser vom Terrain verdeckt wird, der Lichtstrahl die Karte verlässt oder der Lichtstrahl eine Höhe hat, die höher ist als der höchste Punkt auf dem Terrain. Tritt einer der letzten beiden genannten Bedingungen ein erreicht der Sonnenlichtstrahl diesen Pixel und erhält folglich Energie. Die Energiemenge wird über die Tageslichtstunden akkumuliert. Um den Algorithmus zu beschleunigen, wird berechnet, wie weit dem Lichtstrahl gefolgt werden muss bis eine neue Pixelkoordinate erreicht wird und somit sich auch die momentane Terrainhöhe ändert. Diese berechnete Schrittweite ist in der Variable *t_step* wieder zu finden. Das Listing 9.12 zeigt den Code zu dieser Schleife. Je nachdem in welcher Richtung (x- oder y-Achse) die nächste Änderung der Pixelkoordinate erfolgt, wird die Variable *t_step* berechnet. Der Vergleich macht die Umrechnung der Realkoordinaten in Pixelkoordinaten

notwendig.

Listing 9.12: Die Schleife zur schrittweisen Verfolgung des Sonnenlichtstrahls. Falls das Terrain den Strahl blockiert, bricht der Algorithmus ab und der Pixel erhält für diese Stunde kein Sonnenlicht.

```

1 while 0 <= x_real_world_pos < map_x_y_boundary and 0 <=
2     y_real_world_pos < map_x_y_boundary and z_real_world_pos <
3         heightmap_max_height:
4     if x_step != 0.0 or y_step != 0.0:
5         if x_real_world_pos % pixel_size >= y_real_world_pos % 
6             pixel_size and x_step != 0:
7             t_step = (pixel_size - (x_real_world_pos % pixel_size)) /
8                 x_step
9         else:
10            t_step = (pixel_size - (y_real_world_pos % pixel_size)) /
11                y_step
12    else:
13        break # sun stands in zenith so every pixel will receive light
14    t_step = abs(t_step)
15    t += t_step
16    x_real_world_pos = x_start_world_pos + t * x_step
17    y_real_world_pos = y_start_world_pos + t * y_step
18    z_real_world_pos = z_start_world_pos + t * z_step
19    x_pixel_pos = int(x_real_world_pos / pixel_size)
20    y_pixel_pos = int(y_real_world_pos / pixel_size)
21
22    if x_pixel_pos < 0 or y_pixel_pos < 0 or x_pixel_pos > map_size - 1
23        or y_pixel_pos > map_size - 1:
24        break
25    terrain_height = self.controller.image_height_map.image[y_pixel_pos]
26        ][x_pixel_pos] * height_conversion
27    line_height = z_real_world_pos
28    if terrain_height > line_height:
29        sun_beam_reaches_pixel = False
30        break # something blocks the light from the sun for that pixel
31    if sun_beam_reaches_pixel:
32        self.insolation_image.image[y][x] +=
33            SOLAR_CONSTANT_K_CALORIES_PER_HOUR

```

9.2.4 Tatsächliche Energiemenge

Nachdem die theoretische Energiemenge pro Pixel berechnet wurde, wird im Anschluss daran die tatsächliche Energie ermittelt. Eine Menge der Energie geht durch die Atmosphäre verloren, wird von Wolken oder von der Bodenoberfläche reflektiert. Die Bodenreflektion hängt von dessen Albedowert ab. Listing 9.13 zeigt den Source-Code zur Berechnung der tatsächlichen Energiemenge pro Pixel.

Listing 9.13: Ein Teil der Energie durch die Sonnenstrahlen geht durch die Atmosphäre und Reflektion verloren.

```

1 | for y in range(self.controller.image_height_map.size):
2 |     for x in range(self.controller.image_height_map.size):
3 |         pixel_raw_insolation = self.insolation_image.image[y][x]
4 |         cloud_reflection_loss = pixel_raw_insolation * map.biom.
5 |             cloud_reflection / 100
6 |         atmospheric_absorption_loss = pixel_raw_insolation * map.biom.
7 |             atmospheric_absorption / 100
8 |         atmospheric_diffusion_loss = pixel_raw_insolation * map.biom.
9 |             atmospheric_diffusion / 100
10 |        soil_id = self.controller.soil_ids_map.image[y][x]
11 |        soil = self.controller.search_soil(soil_id)
12 |        self.insolation_image.image[y][x] = (pixel_raw_insolation -
13 |            cloud_reflection_loss - atmospheric_absorption_loss -
14 |            atmospheric_diffusion_loss) * (1.0 - soil.albedo)

```

9.2.5 Reflektion

Ein Teil der Bodenreflektion wird an dessen Umfeld abgestrahlt. Dies wird so realisiert, dass bei jedem Pixel die Menge der Energie seiner angrenzenden Nachbarpixel aufsummiert und durch die Anzahl der Nachbarn geteilt wird, um einen Durchschnitt zu erhalten. Anschließend erhält der Pixel einen Teil dieses Durchschnitts. Der Anteil ist in der GUI einstellbar.

Listing 9.14: Ein Teil der Reflektion der Bodenoberfläche wird an dessen angrenzenden Pixel abgestrahlt.

```

1 | for y in range(1, self.controller.image_height_map.size + 1):
2 |     print("Reflection: " + str(y))
3 |     for x in range(1, self.controller.image_height_map.size + 1):
4 |         neighbor_insolation_sum = 0
5 |         neighbor_insolation_sum += padded_insolation_image[y][x+1]
6 |         neighbor_insolation_sum += padded_insolation_image[y+1][x+1]
7 |         neighbor_insolation_sum += padded_insolation_image[y+1][x]
8 |         neighbor_insolation_sum += padded_insolation_image[y+1][x-1]
9 |         neighbor_insolation_sum += padded_insolation_image[y][x-1]
10 |        neighbor_insolation_sum += padded_insolation_image[y-1][x-1]
11 |        neighbor_insolation_sum += padded_insolation_image[y-1][x]
12 |        neighbor_insolation_sum += padded_insolation_image[y-1][x+1]
13 |        added_reflection_insolation = neighbor_insolation_sum / 8 *
14 |            reflection_coefficient
15 |        self.insolation_image.image[y-1][x-1] +=
16 |            added_reflection_insolation

```

9.2.6 Orography

Zur Erstellung des Normalen-Bildes wird zuerst ein Scheitelpunktnetz erzeugt. Die x- und y-Koordinaten orientieren sich an den Pixelkoordinaten und der Pixelgröße der Map. Die Höhe (z-Koordinate) der Scheitelpunkte ist durch den Höhenwert der Heightmap und des Höhenkonversionswertes (siehe Abschnitt 8.3) bestimmt. Anschließend wird zu den sechs

angrenzenden Nachbarflächen die Normale berechnet. Diese werden aufsummiert und normalisiert. Das Ergebnis ist die Normale zu jedem Pixel der Map.

9.2.7 Edaphology

Auf Basis der Normalmap kann zu jedem Pixel berechnet werden wie eben oder schräg dieser Punkt auf der Karte ist. Dazu wird das Produkt aus der Normalen und des Vektors, welcher gerade nach oben zeigt (siehe Abschnitt 8.3), berechnet. Anhand dieses Winkels und der maximalen Bodentiefe, welche in den Map-Informationen gespeichert ist, wird die Bodentiefe an einem Punkt bestimmt.

Listing 9.15: Die Funktion berechnet die Bodentiefe an jedem Punkt der Map anhand des Winkels und der maximalen Bodentiefe.

```

1 def calculate_soil_depth(map, size, angles):
2     soil_depths = Image(size=size)
3     x = 0
4     y = 0
5     for rows in angles:
6         for angle in rows:
7             depth = (1 - angle/90) * map.max_soil_depth
8             soil_depths.image[y][x] = depth
9             x += 1
10            y += 1
11            x = 0
12    return soil_depths

```

9.2.8 Hydrology

Die Berechnung der Wassermenge an einem Punkt hängt von der Bodentiefe, der Wasser- aufnahme des Bodens, der Grundwassermenge und der durchschnittlichen Regenmenge ab. Bei niedriger Bodentiefe von weniger als einem Meter kann der Boden nur einen Teil der Wassermenge aufnehmen. Wiederum kann der Boden nur so viel Wasser aufnehmen, wie dessen Absorptionskraft es zulässt. Außerdem wird berechnet, wie viel Wasser durch die Sonneneinstrahlung verdampft. Dazu wird die vorhandene Energiemenge mit der benötigten Energiemenge, um ein Milliliter Wasser zu verdampfen, verrechnet (siehe Abschnitt 8.5). Das Listing 9.16 zeigt den entsprechenden Algorithmus.

Listing 9.16: Berechnung der Wassermenge an einem Punkt der Karte.

```

1 if depth >= 100:
2     depth_coefficient = 1.0
3 else:
4     depth_coefficient = depth / 100
5 water_supply = (biom.groundwater + biom.avg_rainfall_per_day) *
6                 depth_coefficient * soil.water_absorption
7 evaporated_water = (image_insolation_map.image[y][x] *
8                      K_CALORIES_NEEDED_TO_EVAPORATE_1_G_WATER) / 1000
9 if evaporated_water > water_supply:
10    evaporated_water = water_supply

```

```
9 | water_supply -= evaporated_water
```

9.2.9 Probabilities

Zur Berechnung der Wachstumswahrscheinlichkeit einer Pflanze werden zuerst die einzelnen Wahrscheinlichkeiten für jeden Standortfaktor der Pflanze berechnet und anschließend wird die geringste Wahrscheinlichkeit als die Finale gewertet (siehe Abschnitt 8.6). Für die Standortfaktoren Licht und Wasser wird die folgende Funktion (Listing 9.17) verwendet. Falls die vorhandene Menge (Energie oder Wasser) niedriger als die benötigte Menge ist, nimmt die Wahrscheinlichkeit linear ab. Liegt der Wert darüber so nimmt sie umgekehrt ab und ergibt ab der doppelt benötigten Menge null. Dies realisiert eine Spanne der Wachstumswahrscheinlichkeit.

Listing 9.17: Die Funktion zur Berechnung der Wahrscheinlichkeit anhand der vorhandenen und benötigten Menge (z.B. Wasser oder Energie).

```
1 def calculate_probability(needed, available):
2     if available <= needed:
3         probability = available / needed
4     elif available <= needed * 2:
5         probability = 1 - (available / needed - 1)
6     else:
7         probability = 0
8     return probability
```

9.3 GUI

Die Realisierung der GUI wurde mit Hilfe des Toolkits *Tkinter* erstellt. Mit diesem ist es einfach, grafische Benutzeroberflächen für Windows, Mac OS und Linux zu erstellen. Tkinter bietet die Möglichkeit verschiedene Bedienelemente wie Buttons, Schieberegler oder Labels in ein Fenster einzubinden und diese mittels eines Layouts zu positionieren. Der Layout-Manager bietet dazu drei mögliche Layouts: Grid, Pack und Place. Das Gridlayout positioniert die Elemente wie in einer Tabelle, d.h. die Reihe und Spalte können jedem Element zugeordnet werden. Das Pack-Layout nimmt dem Entwickler die Arbeit ab und entscheidet selbst, wo es welches Element platziert. Für die pixelgenaue Platzierung kann der Place-Manager verwendet werden. Für die Applikation dieser Arbeit wurde das Grid-Layout gewählt, da dieses das passende Mittelmaß aus Genauigkeit der Platzierung und der einfachen Verwendung bietet.

Für alle Daten (siehe Abschnitt 9.1) wurde sowohl ein Fenster zur Auflistung aller bisher angelegten Informationen als auch ein Fenster zu Erstellung von neuen Objekten angelegt. Alle Fenster werden über das *MainWindow* verwaltet. Dieses legt ein Objekt von jeder Fenster-Klasse an, übergibt dabei eine Referenz auf das Hauptfenster und speichert die angelegten Fenster in einer assoziativen Liste. Siehe Listing 9.18.

Listing 9.18: Alle Fenster-Klassen werden angelegt und in einer Liste gespeichert. Jedes Fenster-Objekt erhält eine Referenz auf das Hauptfenster.

```

1 self.container = tk.Frame(self)
2 self.container.pack(side="top", fill="both", expand=True)
3 self.container.grid_rowconfigure(0, weight=1)
4 self.container.grid_columnconfigure(0, weight=1)
5
6 self.frames = {}
7 for F in (MenuWindow, MapWindow, NewMapWindow, SoilWindow,
8     NewSoilWindow, BiomWindow, NewBiomWindow, VegetationWindow,
9     newVegetationWindow, ProbabilityCloudWindow):
10    page_name = F.__name__
11    frame = F(self.container, self, self.controller)
12    self.frames[page_name] = frame
13    frame.grid(row=0, column=0, sticky="nsew")

```

Um von einem Fenster zum nächsten zu wechseln, wird die *show_frame* Methode aufgerufen. Diese führt die *tkraise* Methode des Fensters auf, um es in den Vordergrund zu bringen. Ferner ist es notwendig, die Fenster zu aktualisieren, wenn z.B. eine neue Vegetation angelegt wurde und diese in der Auflistung erscheinen soll. Die Aktualisierung eines Fensters übernimmt die Methode *update_frame*. In Listing 9.19 ist die Implementierung beider oben beschriebenen Methoden zu sehen.

Listing 9.19: Methoden für das Wechseln und Aktualisieren eines Fensters.

```

1 def show_frame(self, page_name):
2     frame = self.frames[page_name]
3     frame.tkraise()
4
5 def update_frame(self, page_name):
6     frame = page_name(self.container, self, self.controller)
7     self.frames[page_name.__name__] = frame
8     frame.grid(row=0, column=0, sticky="nsew")

```

Die Implementierung einer GUI zum Anlegen aller erforderlichen Daten und zur Berechnung und Darstellung der Ergebnisse erfüllen die Anforderungen 6 und 7 aus Kapitel 3.

9.4 Zusammenfassung

Die für die Berechnungen notwendigen Daten wie Biom, Map, Bodenart und Vegetation müssen zuerst angelegt werden. Diese werden als Yaml-Dateien persistent gespeichert. Bevor die Berechnung der Wahrscheinlichkeiten gestartet werden kann, müssen die notwendigen Informationen aus dem Höhenfeld und der Soil-Map ermittelt werden. Zuerst wird die Sonneninstrahlung berechnet. Dies ist die komplizierteste Berechnung, da über mehrere Stunden zu jedem Pixel ermittelt werden muss, ob dieser Licht erhält oder im Schatten liegt. Außerdem wird die Reflektion der Nachbarpixel dazu addiert. Als nächstes erfolgt die Erstellung der Normalmap, welche das Relief der Karte widerspiegelt und im nächsten Schritt für die Bodentiefe benötigt wird. Je steiler das Terrain an einem Punkt, desto weniger Bodentiefe

ist vorzufinden. Die Bodentiefe wird zuletzt verwendet, um die Wassermenge zu berechnen. Neben der Regenmenge und dem Grundwasser spielt hierbei auch die Sonneneinstrahlung eine Rolle, da diese einen Teil der vorhandenen Wassermenge verdampfen lässt. Das Auflisten und Anlegen neuer Daten und die Darstellung der Berechnungen werden mittels der GUI grafisch dargestellt und vom Nutzer gesteuert.

10 Ergebnisse

Dieses Kapitel zeigt die Ergebnisse der Applikation dieser Arbeit. Es wird die entwickelte GUI und dessen Bedienung erklärt und die Endergebnisse sowie deren Farbkodierungen aufgezeigt.

10.1 GUI

Die implementierte GUI startet zuerst in einem Hauptmenü (siehe Abbildung 10.1). In diesem hat der Benutzer die Möglichkeit alle bisher angelegten Biome, Bodenarten, Vegetationsarten und Maps auflisten zu lassen (siehe Abbildung 10.2) bzw. neue anzulegen (siehe Abbildung 10.3).

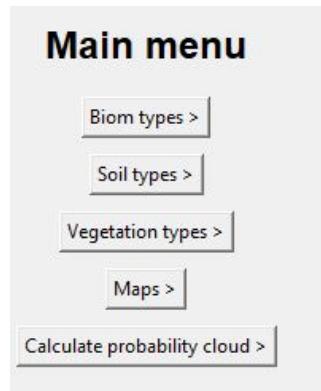


Abbildung 10.1: Hauptmenü der Applikation. (eigene Darstellung)

[< Back to menu](#)

Vegetation types:

[Create new vegetation type >](#)

Appletree

Energy demand: 2000.0 kcal/day
 Water demand: 0.7 l/day
 Soil demand: Dirt
 Soil depth demand: 80.0 cm

Shrub

Energy demand: 1200.0 kcal/day
 Water demand: 0.3 l/day
 Soil demand: Dirt
 Soil depth demand: 20.0 cm

Moss

Energy demand: 1000.0 kcal/day
 Water demand: 0.3 l/day
 Soil demand: Stone
 Soil depth demand: 1.0 cm

Abbildung 10.2: Auflistung der angelegten Vegetationsarten. (eigene Darstellung)

[< Back to vegetation types](#)

New vegetation type:

Name:

Energy demand (in kcal/day):

Water demand (in l/cm²):

Soil demand:

Soil depth demand (in cm):

Abbildung 10.3: Erstellung einer neuen Vegetationsart. (eigene Darstellung)

10.2 Bedienung

Das *Calculate probability cloud* Fenster erlaubt eine angelegte Map zu laden (siehe Abbildung 10.4). Daraufhin wird das Höhenfeld und die Soil-IDs map angezeigt. Außerdem werden bereits berechnete Maps (z.B. Insolation Map) mit geladen, damit diese nicht neu berechnet werden müssen. Beim Laden spielt die Anzahl der Tageslichtstunden eine wichtige Rolle, denn diese beeinflusst das Ergebnis der Insolation Map und der Hydrology Map. Deshalb

wird die `daylight_hours` Variable mit in den Dateinamen der berechneten Insolation und Hydrology geschrieben und beim Laden wird nach diesem Dateinamen gesucht.

The screenshot shows a user interface for calculating a probability cloud. At the top, it says "Calculate probability cloud:". Below that is a dropdown menu labeled "Map:" with a downward arrow. Underneath is a text input field labeled "Daylight hours:" containing the value "13". A button labeled "Load map" is positioned below the input field. There are five more input fields stacked vertically: "Sun start elevation (0-90°): 0", "Sun start azimuth (0-360°): 0", "Sun max elevation (0-90°): 80", and "Reflection (0.0 - 1.0): 0.1".

Abbildung 10.4: Das Laden einer Map und die Einstellungsmöglichkeiten für die Berechnungen. (eigene Darstellung)

Des Weiteren kann, wie in Abschnitt 9.2.3 beschrieben, die Start- und Max-Elevation sowie der Start-Azimut-Winkel festgelegt werden. Ferner kann der Anteil der Nachbarpixel-Reflektion angegeben werden. Durch drücken des *Calculate and save all maps* werden alle Berechnungen nacheinander ausgeführt und automatisch abgespeichert. Es ist auch möglich die Berechnungen einzeln auszuführen, falls es nicht nötig ist alle Berechnungen zu starten. Dazu gibt es unterhalb jeder Map einen *Calculate <map>* Button. Liegen alle Berechnungen vor, so kann mit Hilfe des Dropdowns *Vegetation type* die Vegetationsart ausgewählt werden für die die Wahrscheinlichkeiten berechnet werden sollen. Das Ergebnis ist in der Probability Map zu sehen und kann mittels darunter liegenden Button gespeichert werden. Die gespeicherten Ergebnisse landen in einem eigenen Ordner unter dem Pfad *resources/results/* gefunden werden, wobei der Ordnername dem Namen der Map entspricht.

10.3 Farbkodierung

Die Ergebnis-Bilder sind farbkodiert. Die Insolation Map hat eine Farbkodierung bei der niedrige Werte (wenig Kilokalorien) dunkelbraun, höhere Werte gelb und die höchsten Werte weiß dargestellt werden. Die Normalmap hat die typische Farbkodierung, d.h. Vektoren, die nach Süden weisen werden grün, nach Norden zeigende rot und vertikal nach oben weisende lila dargestellt. Die Farbkodierung der Edaphic Map erstreckt sich von Dunkelbraun (hohe Bodentiefe) über Ocker bis Hellgelb (niedrige Bodentiefe). Bei der Hydrology Map ist es von weiß bis dunkel blau, d.h. je dunkel blauer der Pixel, desto mehr Wasser ist vorhanden. Das Ergebnis der Probability Map erstreckt sich von schwarz (Wahrscheinlichkeit von null Prozent) über grau (etwa 50%) bis weiß (100%).

10.4 Ergebnisbilder

Es ist schwer Informationen zu den Bodenarten eines Gebietes zu finden. Deshalb wurde eine Workaround-Lösung benutzt, um die Bodenarten-Bilder zu erzeugen. Die Soil-IDs-Maps wurden durch Posterisation des Höhenfeldes erzeugt. Posterisation ist ein Effekt zur Farbreduktion, sodass ein Bild anschließend aus einer geringeren Menge von Farben besteht. Das Grafikbearbeitungsprogramm GIMP¹ wurde benutzt, um mittels Posterisation aus den Höhenbildern die Soil-IDs-Maps zu erzeugen. Die so erzeugten Soil-IDs hängen demnach mit der Höhe des Höhenfeldes zusammen. Die Abbildung 10.5 zeigt eine Heightmap und die durch Posterisation erstellte Soil-IDs-Map.

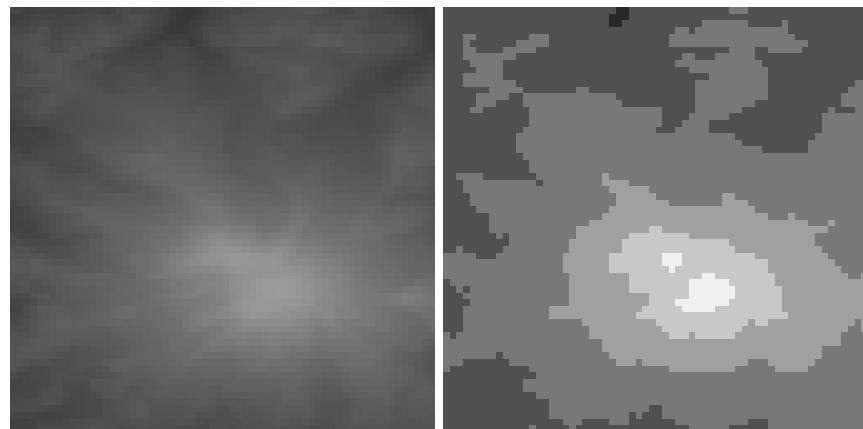


Abbildung 10.5: Aus einem Höhenfeld (links) wird mittels Posterisation eine Soil-IDs-Map (rechts) erzeugt. (eigene Darstellung)

Die Abbildung 10.6 zeigt die Ergebnisse eines 512 mal 512 Pixel großen Höhenfeldes. Bei diesem Höhenfeld erstreckt sich ein Bergkamm von Westen nach Osten. Deshalb sind im Norden der Insolation Map dunklere Flecken zu sehen, da der nördliche Teil durch den Gebirgskamm zu frühen und späten Tagesstunden im Schatten liegt. Im Relief-Bild ist gut das Gebirge zu erkennen, das kann helfen zu verstehen wo z.B. Schatten entstehen. Die Edaphology Map weist an den ebenen Stellen des Gebirges die größte Bodentiefe auf. Die Hydrology Map ist ähnlich zu der Edaphology Map, da die Wassermenge mit der Bodentiefe korreliert. Allerdings weicht diese ab, da Stellen mit viel Licht auch mehr Wasser verdampfen lassen. Deshalb ist im Norden der Karte die größte Wassermenge zu finden, da sich dort die Bodentiefe und Schatten ergänzen.

Das Ergebnis der Probability Map ist für eine Tanne (Fir) berechnet worden. Die Tanne benötigt Ton (Clay) als Boden und kann deshalb nur in diesen Bereichen wachsen (auf der Soil-IDs Map in mittel-grüner Farbe). Die höchste Wahrscheinlichkeit ist im Süden zu finden, da die Tanne viel Energie (2200 Kilokalorien) am Tag benötigt, welches im schattigen Norden nicht ausreichend vorzufinden ist.

¹<https://www.gimp.org/>

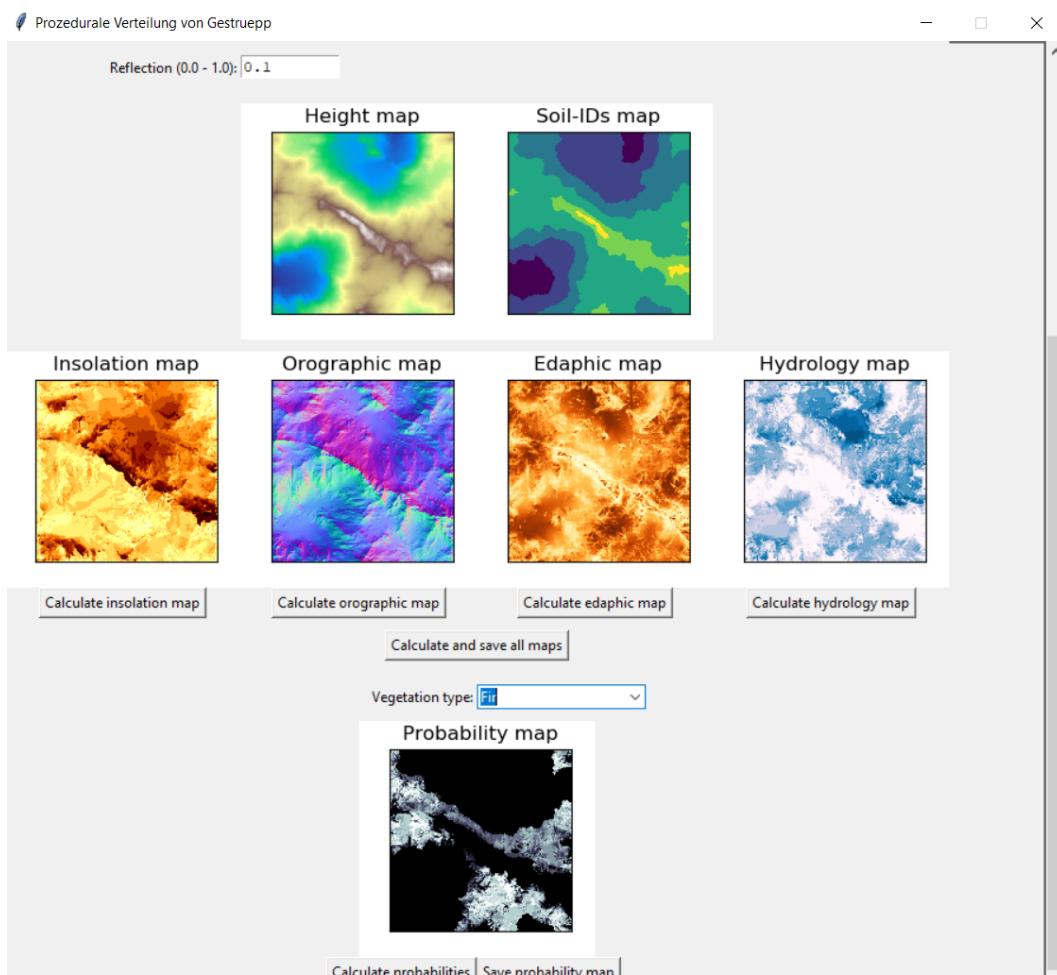


Abbildung 10.6: Die Ergebnisse eines 512 x 512 Pixel großen Höhenfeldes. (eigene Darstellung)

Abbildung 10.7 zeigt das Ergebnis einer 1024 Pixel großen Textur. Das Höhenfeld wurde über die Webseite terrain.party² erstellt. Die Seite ermöglicht es das Höhenfeld einer Region von überall auf der Erde zu erstellen. In diesem Fall wurde das Gebiet bei Weißbach bei Lofer in Österreich gewählt, um das Höhenfeld zu erstellen. Im Süden befindet sich ein Gebirgszug mit einem langen Hang nach Norden. Darauf folgt ein kurzes Stück ebener Fläche, welches sich von Nordwesten nach Südosten schlängelt, dahinter folgt ein Hang, der sich nach Nordosten erstreckt. Da das Terrain sehr hügelig ist, weist die Insolation Map viel Schatten auf, da die meisten Bereiche im Laufe des Tages im Schatten liegen. Das ebene schlängenförmige Tal weist die größte Bodentiefe und durch den Schatten auch das meiste Wasser auf. Das Ergebnis der Probability Map wurde für Buchen berechnet, welche nur auf Lehmboden wachsen können (helleres Blau in der Soil-IDs-Map). Die größte Wachstumswahrscheinlichkeit ist da zu finden, wo der Boden am tiefsten und genügend Wasser zu finden ist. In dem Fall ist das ein Plateau nördlich des Tals in der Mitte der Karte.

²<https://terrain.party/>

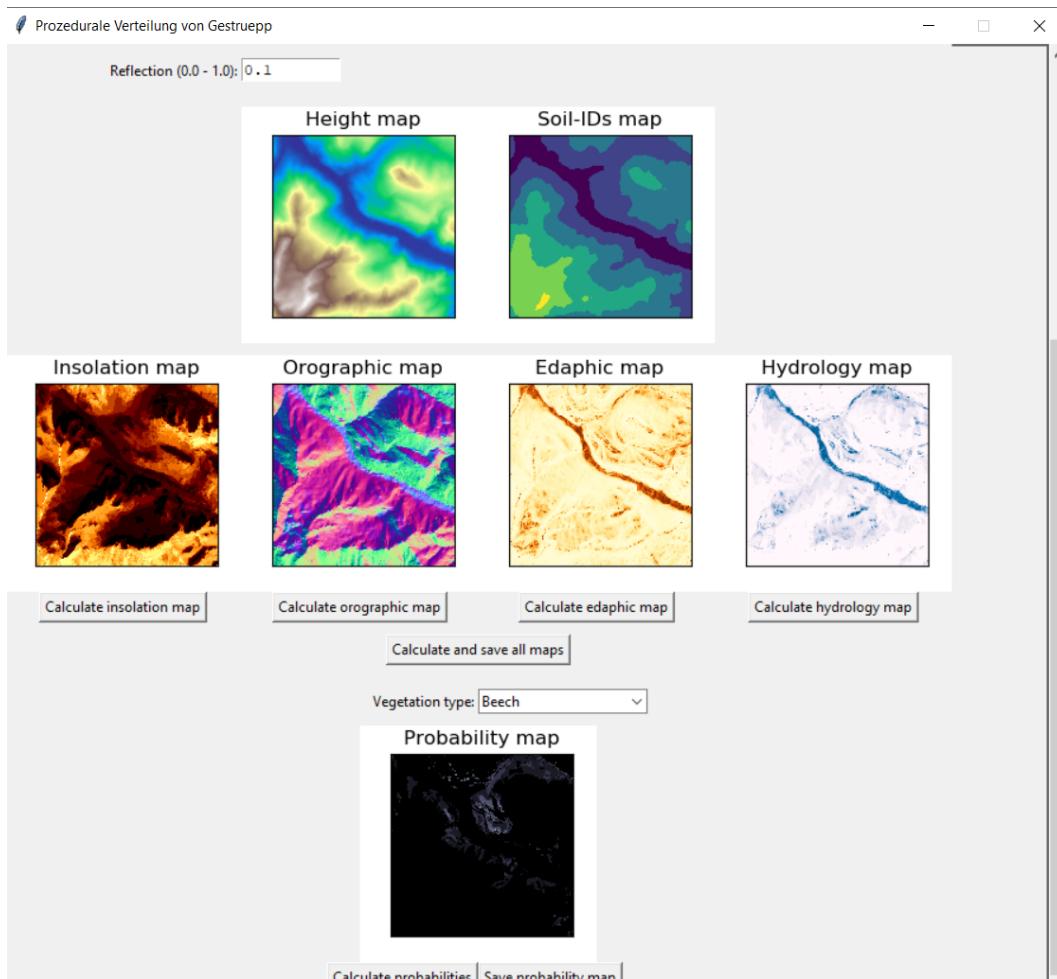


Abbildung 10.7: Die Ergebnisse eines 1024 x 1024 Pixel großen Höhenfeldes. (eigene Darstellung)

Das nächste Beispiel (siehe Abbildung 10.8) enthält zu allen Maps eine Legende, die erklärt welchen Wert eine Farbe repräsentiert. Die Wahrscheinlichkeiten wurden für Apfelbäume berechnet.

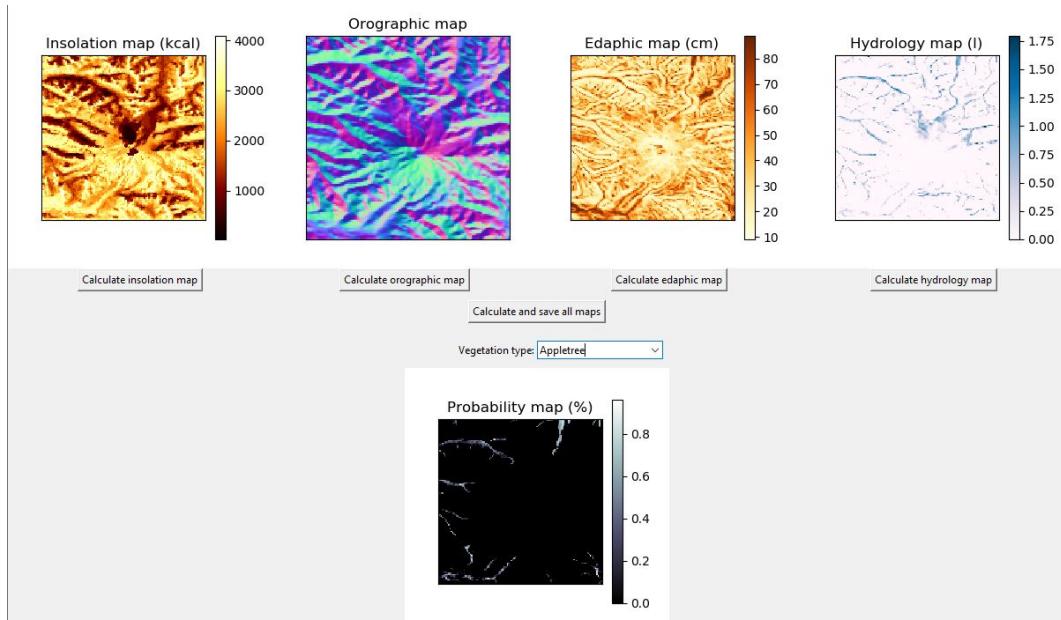


Abbildung 10.8: . (eigene Darstellung)

In Abbildung 10.9 werden die Ergebnisse einer 2048 mal 2048 Pixel großen Textur gezeigt. Da die Map als subtropisches Biom eingestellt wurde, ist die Absorption durch die Atmosphäre gering und das Terrain erhält sehr viel Energie. Der Großteil des Wassers ist verdampft und erlaubt nur wenigen Pflanzen zu wachsen. Deshalb wurde der Kaktus als Vegetationsart hinzugefügt, aber selbst diese findet nur wenige Stellen zum Wachsen (siehe Probability Map).

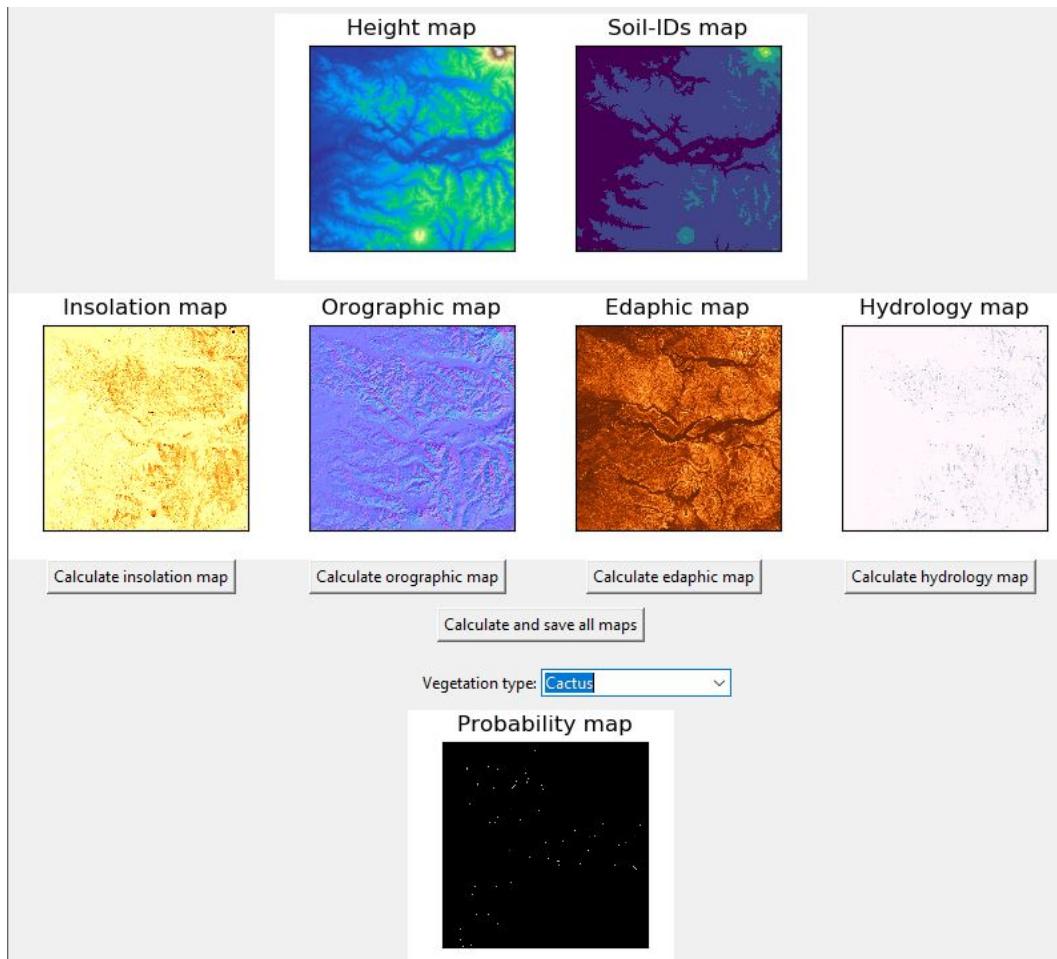


Abbildung 10.9: Die Ergebnisse eines 2048 x 2048 Pixel großen Höhenfeldes. (eigene Darstellung)

10.5 Zusammenfassung

Die GUI bietet die Möglichkeit zum Anlegen aller benötigten Daten (Bodenarten, Biome, Maps und Vegetationsarten). Außerdem bietet die GUI eine Oberfläche zur Berechnung der benötigten Informationen (Sonneneinstrahlung, Relief, Bodentiefe und Feuchtigkeit). Die Ergebnisse werden farbkodiert dargestellt. Anhand der ermittelten Informationen können schließlich die Wachstumswahrscheinlichkeiten einer Vegetation auf der gesamten Map berechnet werden. Die Resultate wurden teilweise auf Basis realer Höhendaten erstellt.

11 Ausblick

Die in dieser Arbeit vorgestellte Applikation kann erweitert werden, um die Qualität und Glaubwürdigkeit der Ergebnisse zu erhöhen.

11.1 Wärme

Die vorgestellte Applikation berechnet die durch die Sonneneinstrahlung vorzufindende Kalorienanzahl an jedem Punkt, jedoch spiegelt dies nicht die benötigte Wärme einer Vegetation wider. Die Wärme könnte nicht nur durch die Klimazone spezifiziert werden, sondern auch durch das Mikroklima, das durch die Aufnahme und Abgabe des Bodens, Gestein und Gewässer, erzeugt wird. So ist es in kleinen Nischen mit den passenden Bedingungen möglich, dass eine Vegetation in einer Klimazone wachsen kann, obwohl diese vom Makroklima zu kalt oder zu heiß wäre. Steine nehmen z.B. tagsüber Wärme auf und geben diese über die Nacht langsam wieder ab. Dieses Prinzip machen sich auch Landwirte zu nutzen, um Südfrüchte wie Kiwis in den Alpen anbauen zu können [HL02, S. 177].

11.2 Erosion

Erosion ist die kontinuierliche Abtragung von Boden durch Regen und Wind. Durch Simulation dieser Faktoren könnte berechnet werden an welchen Stellen es weniger oder mehr Boden gibt. Bei der Definition einer Map könnte die häufigste Windrichtung angegeben werden und auf Basis dessen kann der Wind-Vektor bestimmt werden. Anschließend wird der Winkel zwischen diesem Vektor und den Vektoren der Normalmap berechnet. Ist der berechnete Winkel an einem Punkt auf der Map groß so liegt der Punkt entgegen der Windrichtung und umso mehr Boden wird abgetragen.

11.3 Ausbreitungssimulation

Um organischere Ergebnisse zu erhalten, würde eine Ausbreitungssimulation helfen. Bei der Bestimmung der Wachstumspunkte einer Vegetation würde dann die Zeit als weitere Komponente dazu kommen. Auf Basis einer pro Vegetation bestimmbarer Ausbreitungsdichte und Geschwindigkeit werden pro Zeitintervall neue Wachstumspunkte bestimmt werden. Das Ergebnis würde somit realistischer und organischer werden. Eine solche Ausbreitungssimulation bietet die *Unreal Engine 4* mit dem *Procedural Foliage Tool* (siehe Abschnitt 6.2).

11.4 Biotische Faktoren

In dieser Arbeit wurden nur die abiotischen Faktoren, wie Licht, Wasser und Boden, für das Wachstum der Pflanzen berücksichtigt. Jedoch spielen biotische Faktoren, wie Konkurrenz zu anderen Pflanzen oder der Mensch, ebenfalls eine wichtige Rolle für das Pflanzenwachstum. Gerade eine Wachstumssimulation mit Verdrängungen zwischen den Vegetationsarten könnte die Qualität der Ergebnisse bezüglich ihrer Natürlichkeit und Glaubwürdigkeit verbessern. Auch die Auswirkungen der Menschheit hat sehr großen Einfluss auf die Natur und in Applikationen, die die Anwesenheit des Menschen integrieren (z.B. Videospiele), wäre deshalb die Berücksichtigung des Menschen obligatorisch. Ein einfaches Beispiel wären Straßen, die über ein Terrain verlaufen und das Wachstum von Vegetation beeinflussen.

12 Selbstständigkeitserklärung

Ich, Martin Lesser, versichere hiermit, dass ich die vorliegende Masterarbeit mit dem Thema

Prozedurale Verteilung von Vegetation auf einem Höhenfeld-basiertem Terrain

selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Erfurt, 9. September 2019

Ort, Datum

Martin Lesser

Literaturverzeichnis

- [ang] *Winkel zwischen Normalen.* <https://www.quora.com/How-are-angles-between-two-planes-equal-to-the-angle-between-their-normals-Why-do-we-need-the-normal-direction-for-defining-a-plane>, . – Aufruf: 22.08.2019
- [Azi] *Azimut Vertikalwinkel: Horizontales Koordinatensystem.* <https://www.timeanddate.de/astronomie/horizontales-koordinatensystem>, . – Aufruf: 22.08.2019
- [Bha14] BHATIA, SC: *Advanced renewable energy systems, (Part 1 and 2)*. WPI Publishing, 2014
- [BK17] BLATZ, Michael ; KORN, Oliver: A Very Short History of Dynamic and Procedural Content Generation. In: *Game Dynamics*. Springer, 2017, S. 1–13
- [coo] *Reference Coordinate System.* <http://docs.autodesk.com/3DSMAX/15/ENU/3ds-Max-Help/index.html?url=files/GUID-0F3E2822-9296-42E5-A572-B600884B07E3.htm&topicNumber=d30e57159>, . – Aufruf: 22.08.2019
- [DUF15] DUFFY, Z. J. ; W. J. ; WANG: Application of Procedural Generation as a Medical Training Tool. In: *Multimodal Technologies and Interaction* (2015), S. 223
- [EL15] ECK, Wim ; LAMERS, Maarten: Biological Content Generation: Evolving Game Terrains Through Living Organisms, 2015
- [evc] *Ecological Vegetation Classes.* <http://clmstudents.weebly.com/evcs3.html>, . – Aufruf: 05.09.2019
- [far] *Farcry 5.* <https://twvideo01.ubm-us.net/o1/vault/gdc2018/presentations/ProceduralWorldGeneration.pdf>, . – Aufruf: 30.08.2019
- [FE17] FREIKNECHT, Jonas ; EFFELSBERG, Wolfgang: A survey on the procedural generation of virtual worlds. In: *Multimodal Technologies and Interaction 1* (2017), Nr. 4, S. 27
- [Gam16] GAMES, Hello: *No Man's Sky*. Guildford, England, 2016. – Video game
- [Gam17] GAMES, Guerrilla: *Horizon: Zero Dawn*. Amsterdam, Niederlande, 2017. – Video game
- [GeN] *Gena: Rock Gravity Spawning in Unity 3D.* <https://www.youtube.com/watch?v=71irCiARX2I>, . – Aufruf: 19.08.2019

- [Ger11] GEROSA, Giacomo: *Evapotranspiration: From Measurements to Agricultural and Environmental Applications*. BoD–Books on Demand, 2011
- [Ham01] HAMMES, Johan: Modeling of ecosystems as a data source for real-time terrain rendering. In: *Digital Earth Moving*. Springer, 2001, S. 98–111
- [HL02] HOLZER, Sepp ; LIEBCHEN, Konrad: *Agrar-Rebell*. Stocker, 2002
- [Hor] Horizon: *Zero Dawn: GPU-based Procedural Placement in Horizon Zero Dawn*. <https://www.youtube.com/watch?v=Fyj974PdGIU>, . – Aufruf: 19.08.2019
- [LLC⁺10] LAGAE, Ares ; LEFEBVRE, Sylvain ; COOK, Rob ; DEROSE, Tony ; DRETTAKIS, George ; EBERT, David S. ; LEWIS, John P. ; PERLIN, Ken ; ZWICKER, Matthias: A survey of procedural noise functions. In: *Computer Graphics Forum* Bd. 29 Wiley Online Library, 2010, S. 2579–2600
- [Max89] MAXIS: *SimCity*. Emeryville, Vereinigte Staaten, 1989. – Video game
- [Max00] MAXIS: *Die Sims*. Emeryville, Vereinigte Staaten, 2000. – Video game
- [Mec73] MECHTLY, EA: The International System of Units: Physical Constants and Conversion Factors. Second Revision. (1973)
- [Min] Minecraft. <https://www.planetminecraft.com/blog/minecraft-awesome-world-seeds/>, . – Aufruf: 05.09.2019
- [Moj09] MOJANG: *Minecraft*. Stockholm, Sweden, 2009. – Video game
- [NoM] No man's sky. <https://blog.cdkeys.com/no-mans-sky-visions-update/>, . – Aufruf: 05.09.2019
- [nor] Normal vectors. [https://en.wikipedia.org/wiki/Normal_\(geometry\)#/media/File:Surface_normals.svg](https://en.wikipedia.org/wiki/Normal_(geometry)#/media/File:Surface_normals.svg), . – Aufruf: 22.08.2019
- [Oke02] OKE, Timothy R.: *Boundary layer climates*. Routledge, 2002
- [Per85] PERLIN, Ken: An image synthesizer. In: *ACM Siggraph Computer Graphics* 19 (1985), Nr. 3, S. 287–296
- [Per02] PERLIN, Ken: Improving noise. In: *ACM transactions on graphics (TOG)* Bd. 21 ACM, 2002, S. 681–682
- [PK⁺99] PLOEG, Rienk R. d. ; KIRKHAM, MB u. a.: On the origin of the theory of mineral nutrition of plants and the law of the minimum. (1999)
- [PL12] PRUSINKIEWICZ, Przemyslaw ; LINDENMAYER, Aristid: *The algorithmic beauty of plants*. Springer Science & Business Media, 2012
- [Pol] Transform spherical coordinates to Cartesian. <https://de.mathworks.com/help/matlab/ref/sph2cart.html>, . – Aufruf: 22.08.2019
- [rom] Fractal Food: Self-Similarity on the Supermarket Shelf. <https://www.fourmilab.ch/images/Romanesco/>, . – Aufruf: 05.09.2019
- [Spea] Speedtree in Avatar. https://www.vision4d.de/htmldocs/speedtree/images-speedtree/avatar_02.jpg, . – Aufruf: 05.09.2019

- [speb] *SpeedTree Takes Hollywood.* <https://columbiametro.com/article/speedtree-takes-hollywood/>, . – Aufruf: 02.09.2019
- [STKB11] SMELIK, Ruben M. ; TUTENEL, Tim ; KRAKER, Klaas J. ; BIDARRA, Rafael: A declarative approach to procedural modeling of virtual worlds. In: *Computers & Graphics* 35 (2011), Nr. 2, S. 352–363
- [TKSY11] TOGELIUS, Julian ; KASTBJERG, Emil ; SCHEDL, David ; YANNAKAKIS, Georgios N.: What is procedural content generation?: Mario on the borderline. In: *Proceedings of the 2nd international workshop on procedural content generation in games* ACM, 2011, S. 3
- [Urn] *Unreal Engine Procedural Foliage Tool: How to set up and use the Procedural Foliage tool.* <https://docs.unrealengine.com/en-US/Engine/OpenWorldTools/ProceduralFoliage/QuickStart/index.html>, . – Aufruf: 19.08.2019
- [veg] *Parameter Vegetation Studio.* <https://www.awesometech.no/index.php/home/vegetation-studio/>, . – Aufruf: 30.08.2019
- [Wal86] WALTER, Heinrich: *Allgemeine Geobotanik: als Grundlage einer ganzheitlichen Ökologie; 22 Tabellen.* Ulmer, 1986
- [Wit12] WITTIG, Rüdiger: *Geobotanik.* UTB, 2012 (3753)