Hinweise zu dieser Version

Die Originalversion dieser Bachelorarbeit enthält einige Bilder, die aus einem Höhendatensatz des Laubacher Waldes in Hessen generiert wurden. Die Rechte an diesem Datensatz liegen bei dem Landesamt für Denkmalpflege Hessen, Abteilung Archäologie und Paläontologie. Die betroffenen Bilder wurden aus dieser Version der Bachelorarbeit entfernt.

Jonathan Klein

Terrain Shading

Bachelorarbeit

im Fach Informatik

vorgelegt von

Jonathan Klein

Geboren am 08. Oktober 1987 in Siegen

Angefertigt am

Lehrstuhl für Computergraphik und Multimediasysteme
Dept. ETI / Nat.-Wiss. Fak.
Universität Siegen

Betreuer:

Prof. Dr. A. Kolb, Lehrstuhl Computergraphik und Multimediasysteme, Universität Siegen

Dr.-Ing. Martin Lambers, Lehrstuhl Computergraphik und Multimediasysteme, Universität Siegen

Beginn der Arbeit: 22. September 2011

Abgabe der Arbeit: 20. Januar 2012

Eidesstattliche Erklärung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Siegen, den 20. Januar 2012

Zusammenfassung

Diese Arbeit untersucht die Umsetzbarkeit, digitale Geländemodelle in Echtzeit zu visualisieren. Dadurch soll die Möglichkeit geschaffen werden, diese interaktiv zu analysieren. Grundlage dafür bietet eine Reihe Verfahren aus der Kartografie, die aus den Höhenwerten Farbwerte berechnen. Herkömmliche Analysewerkzeuge nutzen diese Verfahren, um aus dem digitalen Geländemodell ein zweidimensionales Ausgabebild zu berechnen.

Im Rahmen dieser Arbeit ist ein Programm entstanden, das eine Auswahl dieser Verfahren implementiert und auf ein dreidimensionales Terrain anwendet. Dieses kann dann durch eine frei bewegliche Kamera erkundet werden. Des Weiteren ist es möglich, die Auswahl der Verfahren sowie deren Parameter in Echtzeit zu verändern. Dadurch wird eine neue Form der Terrainanalyse geschaffen.

Es hat sich gezeigt, dass sich alle ausgewählten Verfahren sinnvoll in Echtzeit berechnen lassen. Die interaktive Einstellung der Parameter ermöglicht darüber hinaus die Anpassung der Ausgabe an jeden beliebigen Geländetyp. Bedingt durch die großen Datenmengen treten allerdings auch einige Probleme auf, die sich teilweise mit erhöhtem Aufwand lösen ließen, aber teilweise auch prinzipieller Natur sind.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Dank der Rechenkraft moderner Computersysteme ist es möglich, digitale Höhenmodelle (*digital evelation Model, DEM*) auf vielfältige Arten zu analysieren. Dafür gibt es eine Reihe unterschiedlicher Verfahren, die verschiedene Variablen visualisieren und ihre spezifischen Vor- und Nachteile haben.

Im Rahmen dieser Arbeit ist ein Programm zur interaktiven Analyse dieser Höhenmodelle entwickelt worden. Die Ausgabe ist dabei im Gegensatz zu herkömmlichen Karten dreidimensional. Es sollen Vertreter der wichtigsten Verfahren implementiert werden und in Echtzeit auf das Gelände angewandt werden können. Es existieren zwar bereits sehr fortschrittliche Terrainanalyse-Werkzeuge, diese arbeiten allerdings nicht in Echtzeit.

Der Begriff *Shading* wird in der Computergraphik allgemein für die Berechnung der Oberflächenfarbe eines Objektes benutzt. *Terrain Shading* bedeutet in diesem Zusammenhang also die Anwendung dieser Verfahren auf ein dreidimensionales Geländemodell, mit dem Ziel dieses untersuchen und analysieren zu können.

Die durch die Echtzeitberechnung entstehende Interaktivität ermöglicht dem Benutzer, die verschiedenen Verfahren vergleichen zu können, sowie deren Stärken und Schwächen zu beurteilen. Außerdem kann durch die interaktive Wahl der Geländeparameter die Ausgabe optimal auf die jeweils interessanten Aspekte des Terrains angepasst werden. Darüber hinaus erhält man durch die frei bewegliche Kamera schneller einen Eindruck über die Struktur des Terrains als dies bei zweidimensionalen Karten der Fall ist.

Das erste Kapitel beschäftigt sich mit den theoretischen Grundlagen der Kartografie und der Analyse von DEMs und listet die verschiedenen Verfahren auf. Im zweiten Kapitel werden einige theoretischen Aspekte der Entwicklung des *Terrain-Explorer* genannten Terrain-Viewers diskutiert sowie die Verfahren detailliert vorgestellt und analysiert. Außerdem wird die Benutzeroberfläche des entwickelten Terrain-Explorers beschrieben.

Das dritte Kapitel beschreibt die generelle Struktur des Terrain-Explorers sowie die Probleme bei dessen Entwicklung. Im letzten Kapitel werden schließlich die implementierten Verfahren verglichen und generelle Vor- und Nachteile eines Echtzeit-Terrain-Viewers besprochen.

Kapitel 2

Grundlagen

2.1 Von Landkarten zur DEM Analyse

Die Ursprünge der DEM-Analyse (*digital elevation model analysis*) liegen in der Kartographie. Landkarten waren schon vor langer Zeit von großer Bedeutung, wenn es darum ging, Länder zu verwalten, Feldzüge zu planen, oder Länderbesitz zu dokumentieren. Mit der Zeit entwickelten sich unterschiedliche Arten von Karten, je nachdem, was dargestellt werden sollte. Wanderkarten (Bild 2.1) beispielsweise verzeichnen verschiedene Rundwege mit Wegpunkten und geben über Schattierungen an, um welches Gelände es sich handelt (Wald, landwirtschaftliche genutzte Flächen, Dörfer etc.). Flurkarten (Bild 2.2) hingegen haben einen kleineren Maßstab und verzeichnen einzelne Grundstücke.

An diesen Beispielen wird deutlich, wie unterschiedlich Landkarten je nach Verwendungszweck aussehen können.

2.1.1 Relief Shading

Relief Shading bezeichnet verschiedene Methoden, um Gebiete in einer natürlichen, ästhetischen und intuitiven Art darzustellen [JR]. Der Betrachter soll anhand einer zweidimensionalen Darstellung eine möglichst gute Vorstellung der Beschaffenheit des Geländes bekommen. Dabei geht es zunächst hauptsächlich um die Topographie, also die Form des Geländes, und nicht um eine Interpretation der Gebiete in Form von Bewuchs, Bebauung oder Besitz.

Relief Shading wurde schon lange vor der Zeit moderner Computersysteme betrieben und war dementsprechend eher ein Kunsthandwerk. Der Zeichner musste durch den Einsatz verschiedener Techniken einen möglichst plastischen Eindruck vermitteln, darüber hinaus sollte die fertige Karte auch einen ästhetischen Gesamteindruck hinterlassen. Auch wenn heutzutage Computer aus Höhendaten ansehnliche Darstellungen generieren können, ist immer noch eine Nachbearbeitung per Hand nötig, um vergleichbare Ergebnisse zu erhalten [JR].

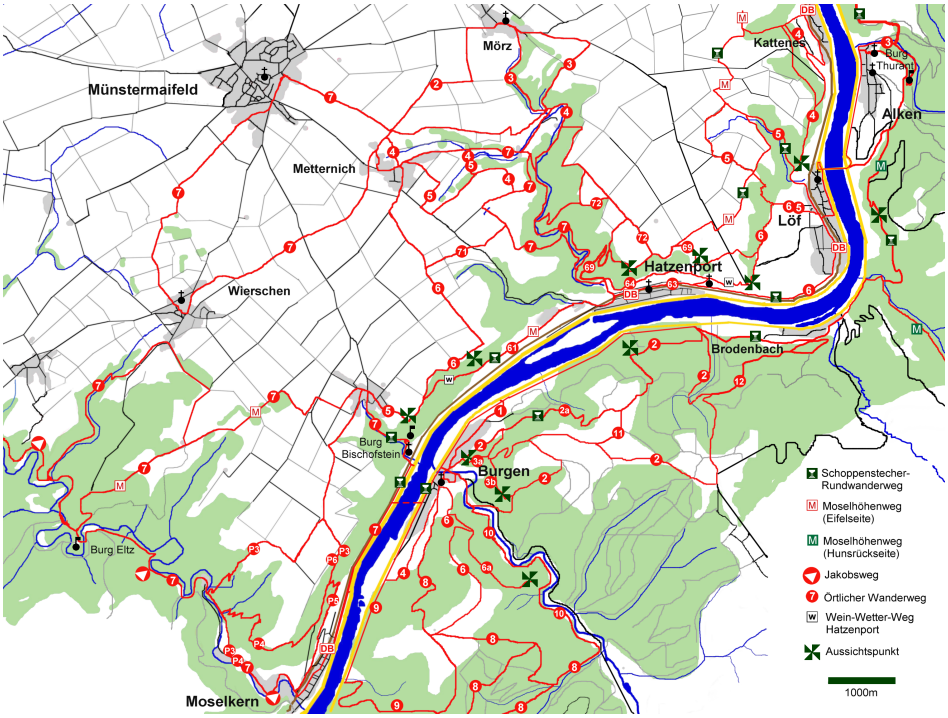

Bild 2.1: Wanderkarte Umgebung Burg Bischofstein
`http://de.wikipedia.org/w/index.php?title=Datei:Karte_Burg_Bischofstein.png`, gemeinfrei

2.1.2 Wissenschaftliche Visualisierung mit DEMs

Mit Hilfe moderner Vermessungsverfahren ist es möglich geworden, wissenschaftliche Visualisierungen am Computer zu berechnen. Grundlage dafür bietet dabei oft ein DEM, also ein Datensatz, der die Höhe des Geländes an jedem Punkt beschreibt. Um diese Modelle zu erstellen, können beispielsweise Laserscanner in Flugzeugen eingesetzt werden [SBDM+10].

Abbildung 2.4 zeigt schematisch die Approximation eines Geländes durch ein DEM. Man sieht jeweils 3 mal den selben Geländeabschnitt aus der Seitenansicht. Die oberste Reihe zeigt das echte Gelände, die mittlere eine Abtastung mit variabler Genauigkeit, die untere eine Abtastung in gleichmäßigen, aber (in diesem Fall) recht großen Abständen. Solche DEMs können beispielsweise als monochromes Bitmap gespeichert werden (gegebenenfalls mit Zusatzinformationen über die Lage des Terrains und Angaben zur Interpretation der Daten).

Prinzipbedingt kann eine digitale Darstellung eines kontinuierlichen Geländes immer nur eine endliche Genauigkeit bzw. Auflösung haben. Dies betrifft sowohl die Anzahl der Messpunkte, als auch deren Genauigkeit (ob Höhenwerte nur auf einige Meter oder wenige Zentimeter genau sind). Da eine höhere Auflösung (die Anzahl der Messpunkte und deren Genauigkeit) durch größeren Speicherbedarf die Archivierung und Verarbeitung der gesammelten Daten erschwert, muss stets abgewogen werden, welche Auflösung benötigt wird.

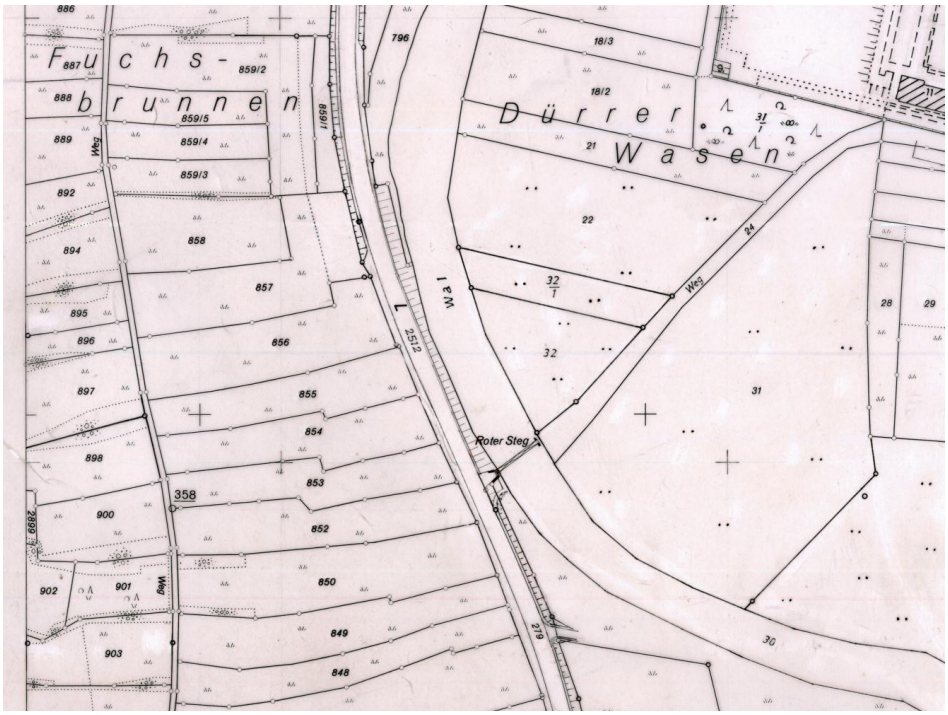

Bild 2.2: Württembergischen Flurkarte im Maßstab 1:2500
http://de.wikipedia.org/w/index.php?title=Datei:Flurkarte.jpg, CC BY-SA 3.0 Schorle

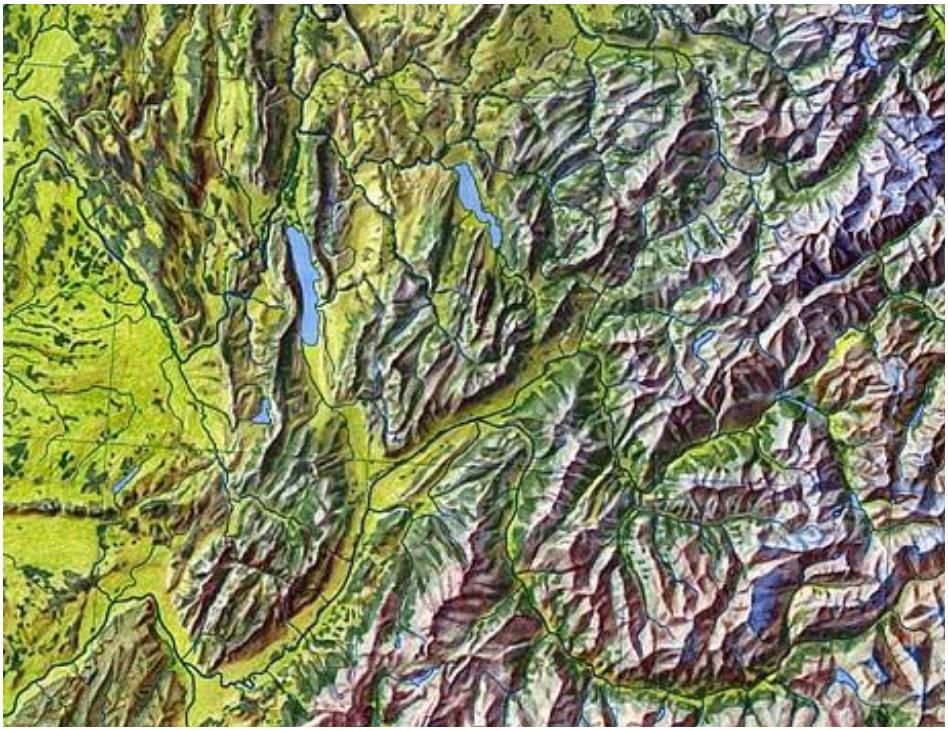

Bild 2.3: „Die Alpen” (Ausschnitt) von Kümmerly & Frey, Berne, 1:750,000.
http://www.reliefshading.com/examples/alpen.html

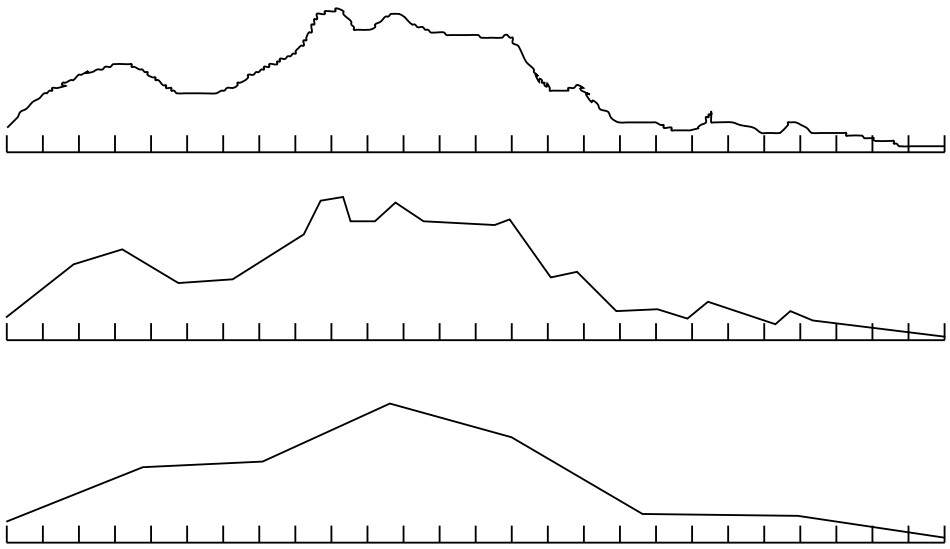

Bild 2.4: zweidimensionale Darstellung einer Geländeapproximation durch ein DEM

In manchen Fällen erschwert eine höhere Auflösung aber auch die Analyse. Untersucht man beispielsweise die Steigung des Geländes, möchte man diese eigentlich eher global betrachten und nicht jede lokale Unebenheit berücksichtigen. In Abbildung 2.4 erkennt man, dass die höher aufgelöste erste Digitalisierung Stellen von extremer Steigung hat, während die zweite bis auf einen Hang relativ flach bleibt. Dies hat natürlich maßgebliche Auswirkungen auf die Interpretation, da man unterscheiden muss, ob ein Gelände wirklich steil oder nur sehr uneben ist. Desweiteren kommt es bei der Vermessung zwangsläufig zu Messfehlern, so dass es oft sinnvoll ist, aus einer hochaufgelösten Messung ein glatteres (und dementsprechend detailärmeres) Modell zu berechnen, in dem es keine Ausreißer mehr gibt.

Interpretation

Aus diesen Höhendaten können durch unterschiedliche Algorithmen verschiedene Größen berechnet werden. Diese werden dann in der Regel auf einer zweidimensionalen Landkarte visualisiert. Dabei müssen stets einige Modellannahmen getroffen werden, die sich später auf die Interpretation auswirken können.

Ein gängiges Verfahren ist beispielsweise die Beleuchtung des Terrains durch eine künstliche Lichtquelle [Ken09]. Da das Auge sehr gut darauf trainiert ist, Licht und Schatten zu interpretieren, entsteht so relativ einfach eine sehr intuitive Darstellung des Geländes. Es gibt allerdings in der Computergraphik eine ganze Reihe von Beleuchtungsmodellen, von denen keines von vornherein richtig oder falsch ist. Man muss die Ergebnisse untersuchen und vergleichen, welche Geländecharakteristik durch welches Verfahren besonders gut sichtbar wird.

Während Beleuchtung als „Menge des einfallenden Lichtes" noch physikalische Grundlagen hat, ist beispielsweise die Rauheit (*roughness*) eines Geländes zwar intuitiv verständlich, aber sehr schwer

allgemein zu definieren. Daher gibt es eine Reihe von unterschiedlichen Algorithmen zur Bestimmung der Rauheit (siehe 3.2.5), die alle leicht unterschiedliche Ergebnisse liefern. Die Beurteilung, welche besser oder schlechter sind, muss der Benutzer von Fall zu Fall entscheiden.

Auch bei Geländeeingenschaften, wie beispielsweise der Steigung, die sich mathematisch präzise definieren lassen, treten Probleme auf. Nur die lokale Steigung zu betrachten, führt, wie zuvor beschrieben, zu einem sehr unruhigen Bild und erhöht den Einfluss von Messfehlern. Daher möchte man meistens die durchschnittliche Steigung in einem kleinen Gebiet berechnen, dessen optimale Größe aber je nach Situation variiert.

Darüber hinaus ist es wichtig, den Wertebereich der Berechnungen möglichst passend auf die Ausgabefarben abzubilden; eine automatische Anpassung ist zwar möglich, aber da der Rechner nicht weiß, welche Details den Benutzer gerade interessieren, werden die Ergebnisse meist nicht optimal sein.

Diese Beispiele deuten schon an, dass man zur bestmöglichen Interpretation eines Geländes sowohl die richtigen Verfahren, als auch deren Parameter geschickt auswählen muss. Ein interaktives System kann hier eine große Hilfe sein, da der Benutzer die Ergebnisse direkt beurteilen kann und so in jeder Situation schnell zu einer günstigen Darstellung des Geländes kommt.

2.2 Die Verfahren

Von den hier vorgestellten Verfahren existieren oft eine Reihe von Variationen, die im nächsten Kapitel genauer untersucht werden.

Höhengradient: Jedem Pixel wird seiner Höhe entsprechend eine Farbe aus einem Farbverlauf zugewiesen, ohne dass benachbarte Pixel beachtet werden

Direktionale Beleuchtung: Das Gelände wird von einer fiktiven Lichtquelle aus beleuchtet

Steigung: Die Steilheit des Geländes, ohne Betrachtung der Ausrichtung. Entspricht der 1. Ableitung

Krümmung: Die Konkavität oder Konvexität des Geländes entlang bestimmter Richtungen. Entspricht der 2. Ableitung

Rauheit: Die Unebenheit des Geländes

Offenheit: Anteil des Himmels der von einem Punkt aus sichtbar ist

Kantenerkennung: Verfahren aus der allgemeine Bildverarbeitung um markante Kanten im Bild zu finden

Normalvektor: Vektor, der senkrecht auf einer Ebene steht, die an das Terrain an dieser Stelle angelegt ist

Richtung: Ausrichtung des Terrains, ohne Berücksichtigung der Steigung

Höhenlinien: Linien, die entlang verschiedener, aber pro Linie fester Höhe verlaufen und so die Konturen von Bergen auf verschiedenen Ebenen nachzeichnen

Die Namensgebung der einzelnen Variationen ist oft sehr vielfältig, obwohl sich an der grundsätzlichen Idee wenig ändert. Das liegt zum einen an den verschiedenen möglichen Interpretationen einer Variable (*Openess* beschreibt die Offenheit, *Sky-View-Factor* die Menge des sichtbaren Himmels), zum anderen an abweichenden Fachwörtern verschiedener Anwendungsgebiete (*hill-shading* in der Kartografie, *direktionale Beleuchtung* in der Computergraphik).

Kapitel 3

Analyse und Konzeption

Ziel dieser Arbeit ist es, einen Terrain-Viewer zu implementieren, der es dem Benutzer erlaubt, mit verschiedenen Visualisierungsverfahren und deren Parametern zu experimentieren. Ein besonderer Fokus wird dabei auf die Interaktivität gelegt, so dass alle Anpassungen des Benutzers in Echtzeit umgesetzt werden müssen. Der Benutzer soll auf einfache Art und ohne viel Zeitaufwand die verschiedenen Details des Terrains gut erkennen können.

Wichtig ist dabei eine frei bewegliche Kamera, die es dem Benutzer ermöglicht, das dreidimensionale Terrain aus einem beliebigen Blickwinkel zu betrachten. Im Gegensatz zu normalen Landkarten kann man also nicht nur die Zoomstufe ändern, sondern auch die Kamera kippen und so beispielsweise einen Berg von der Seite betrachten. Durch die dreidimensionale, perspektivische Darstellung erhöht sich außerdem die Interpretierbarkeit, da das Auge relativ unabhängig von dem konkret zur Visualisierung benutzen Verfahren die grobe Form des Terrains erkennen kann.

Die implementierten Verfahren sollten sich möglichst vielseitig auf das Terrain anwenden lassen. Da die meisten Verfahren nur einen einzelnen Wert liefern, liegt es nahe, mehrere Verfahren in verschiedenen Farbkanälen darzustellen, um so dem Benutzer die Möglichkeit zu bieten, diese miteinander vergleichen zu können. Dabei ist es sinnvoll, sich nicht nur auf den RGB-Farbraum zu beschränken, sondern auch Alternativen wie den HSV Farbraum anzubieten. Darüber hinaus sollte es aber auch möglich sein, einzelne Verfahren durch den gesamten Farbraum darzustellen.

Um Ladezeiten gering zu halten und die enormen Datenmengen eines DEMs handhaben zu können, wird darüber hinaus ein LOD-Verfahren (siehe 3.1.2) benötigt.

3.1 Technische Grundlagen

Im Folgenden werden einige technische Aspekte von DEMs erläutert, die für diese Arbeit von Bedeutung sind.

3.1.1 Ableitung des DEM

Für einige Verfahren (etwa die Steigung oder Krümmung) ist es nötig, Ableitungen des Terrains zu berechnen. Da das DEM zunächst als Treppenfunktion (für jede Position eine Höhe) gespeichert wird, ist es aber zunächst nicht möglich, es direkt abzuleiten. Man benötigt also eine Funktion höherer Ordnung, die das DEM interpoliert und sinnvoll ableitbar ist. Diese Funktion muss dabei nicht das ganze Gelände abbilden, es genügt, wenn sie jeweils in der Nähe des untersuchten Punktes dem Terrain ähnlich ist (es gibt also für jeden Punkt eine eigene Funktion).

Eine Lösung dieses Problems bietet ein aus der Computergraphik bekanntes Verfahren zur Berechnung von Normalen eines Polygonnetzes. Aus dem Abtastraster des DEM wird ein Polygonnetz (wie bei einer Heightmap) aufgebaut, mithilfe des Kreuzproduktes kann man dann die Normalvektoren einzelner Dreiecke berechnen und damit schließlich jedem Punkt einen aus den anliegenden Dreiecken interpolierten Normalvektor zuordnen. Dieses Verfahren wird beispielsweise in [GSR11] verwendet um die Rauheit des Terrains anhand der Unterschiede in den Normalvektoren seiner Umgebung zu bestimmen. Das Terrain wird also in jedem Punkt durch eine anliegende Ebene beschrieben.

Ein rechenaufwändigeres, aber weitaus flexibleres Verfahren wird in [WOB+07] vorgestellt. Dabei wird für jeden Punkt versucht, die Umgebung durch eine zweidimensionale Parabel zu interpolieren, wie Bild 3.1 zeigt.

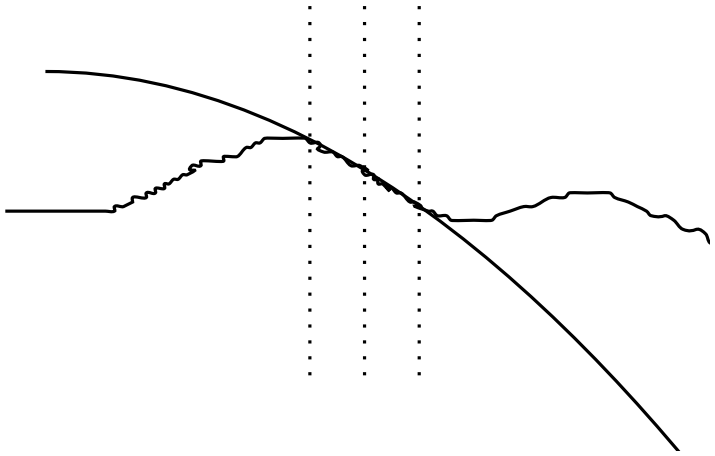

Bild 3.1: Approximation eines Teils des Terrains durch eine Parabel

Man erkennt gut, dass die Parabel den Abschnitt zwischen den Linien sehr gut approximiert, außerhalb der Linien aber einen vollkommen anderen Verlauf als das Terrain hat. Daher muss diese Parabel für jeden Pixel neu berechnet werden.

Im Zweidimensionalen müssen statt der 3 Koeffizienten in $Z = aX^2 + bX + c$ die folgenden 6 Koeffizienten berechnet werden:

$$Z = aX^2 + bY^2 + cXY + Dx + eY + f \tag{3.1}$$

Diese Koeffizienten müssen dabei so bestimmt werden, dass die Parabel die Umgebung möglichst

gut approximiert. Ein einfaches Verfahren dafür ist die *Gaußsche Methode der kleinsten Quadrate*, bei der die Summe der Quadrate der Fehler minimiert wird. Der Vorteil dieses Verfahrens besteht vor allen Dingen darin, dass es nach einiger Vorarbeit nur noch eine Matrixmultiplikation pro Punkt erfordert.

Die gaußsche Methode der kleinsten Quadrate löst ein überbestimmtes Gleichungssystem derartig, dass die Summe der Quadrate der Fehler der einzelnen Gleichungen minimal wird. Das zugehörige Gleichungssystem $A \cdot x = b$ ergibt sich aus Einsetzen verschiedener Terrainpunkte eines Kernels (Umgebung des aktuellen Punktes), der im Bild durch die gestrichelten Linien gekennzeichnet wird, in die Parabelgleichung. Für den Fall eines quadratischen 3×3 Kernels sieht das folgendermaßen aus:

$$\begin{pmatrix} 1 & 1 & -1 & -1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & -1 & -1 & 1 \\ 0 & 1 & 0 & 0 & -1 & 1 \\ 1 & 1 & -1 & 1 & -1 & 1 \end{pmatrix} \cdot \begin{pmatrix} a \\ b \\ c \\ d \\ e \\ f \end{pmatrix} = \begin{pmatrix} (-1,1) \\ (0,1) \\ (1,1) \\ (-1,0) \\ (0,0) \\ (1,0) \\ (-1,-1) \\ (0,-1) \\ (1,-1) \end{pmatrix} \tag{3.2}$$

Der Ergebnisvektor beinhaltet die Höhenwerte an den 9 Punkten des Kernels, die hier jeweils mit ihren Koordinaten aus $\{-1,0,1\} \times \{-1,0,1\}$ gekennzeichnet sind. Die Werte der Matrix ergeben sich, indem man die Koordinaten in die Parabelgleichung einsetzt (die erste Spalte ist also beispielsweise das Quadrat der X Koordinate, die dritte das Produkt aus der X und Y Koordinate, die sechste konstant 1 usw.)

Das Gaussche Verfahren der kleinsten Quadrate besagt nun, dass

$$A^T A x = A^T b$$

ein neues, eindeutig lösbares Gleichungssystem ist, dessen Lösung die beste Näherung für das ursprüngliche Gleichungssystem darstellt. Das gesuchte Ergebnis erhält man also durch einige Umformungsschritte aus

$$x = (A^T \cdot A)^{-1} \cdot A^T \cdot b$$

In [WOB+07] wird für die Umgebung ein quadratischer Kernel mit ungerader Kantenlänge vorgeschlagen, wie man ihn in Bild 3.2 sehen kann. Denkbar wären allerdings auch eine kreisförmige Verteilung, oder eine, die einige Punkte mehrfach beinhaltet um diese stärker zu gewichten.

Der Vorteil dieses Verfahrens besteht nun zum einen darin, dass man die Parabelgleichung zwei-

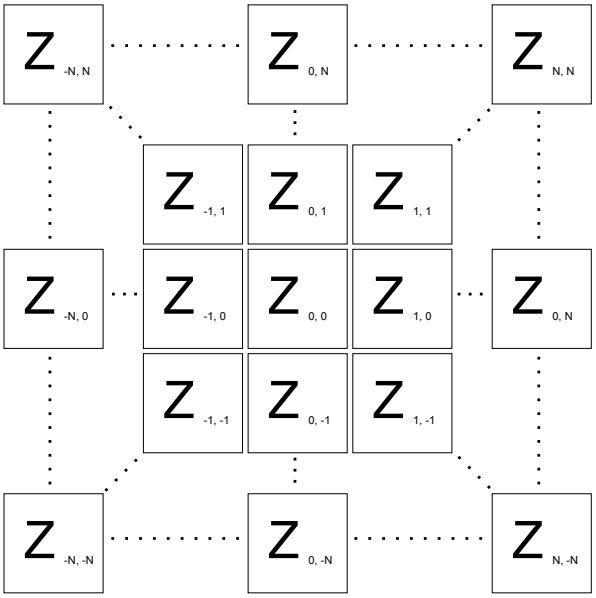

Bild 3.2: Kernel der Größe $2N + 1$

mal ableiten kann (im Gegensatz zum Normalvektor, durch den man nur die erste Ableitung erhalten kann), und zum anderen, dass man durch unterschiedliche Kernelgrößen unterschiedlich lokale oder globale Ableitungen berechnen kann.

3.1.2 Level of Detail

Gebräuchliche DEMs können eine Größe von mehreren Gigabytes erreichen. Während diese Datenmenge zwar für heutige PCs normalerweise kein Problem darstellt, sind sie für den Echtzeiteinsatz viel zu groß, da neue Bilder innerhalb von Sekundenbruchteilen generiert werden müssen und der Speicher aktueller Grafikkarten immer noch sehr beschränkt ist. Um solch große Terrains dennoch in Echtzeit darstellen zu können, wird ein sogenanntes LOD-Verfahren (*Level of Detail*) benötigt.

Bei diesem Verfahren existiert ein Modell in verschiedenen Auflösungen. Wenn ein Objekt weit von der Kamera entfernt ist, kann man eine geringer aufgelöste Version anzeigen (was Rechenzeit spart), ohne dass der Benutzer einen allzu großen Unterschied feststellen kann. Bewegt sich das Objekt näher an die Kamera (bzw. wird am Bildschirm größer dargestellt), wird eine detailliertere Version benutzt. Bei Terrains sind einige Teile in der Regel recht nahe an der Kamera und andere weiter entfernt. Es bietet sich also an, das Terrain in sogenannte Tiles zu unterteilen und dann nicht nur die Auflösung des gesamten Terrains zu verändern, sondern einzelne Tiles getrennt zu betrachten.

Wie man in Bild 3.3 sehen kann, ähnelt die Struktur sehr der eines Quadtrees. In diesem Bild befindet sich die Kamera unten rechts, so dass das Terrain in der Nähe der Kamera stärker unterteilt wurde, um dort mehr Details zu bieten. Bei jeder Unterteilung wird ein Tile durch 4 Tiles halber Seitenlänge ersetzt.

Auch wenn ein LOD-Verfahren größere Terrains erst darstellbar macht, hat es doch einige Nach-

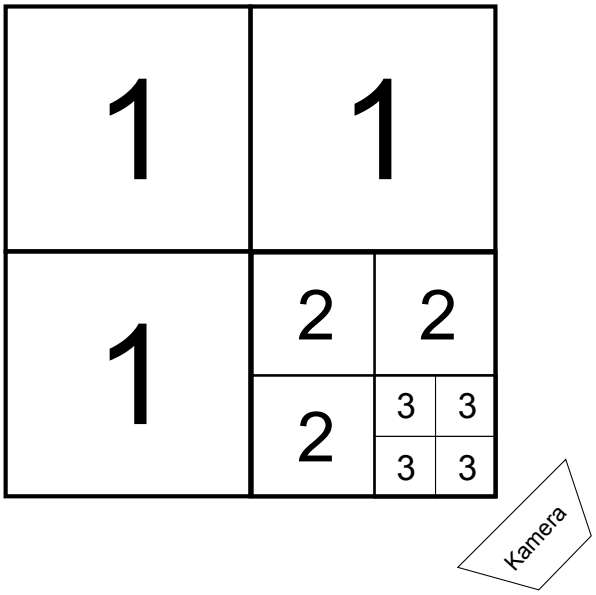

Bild 3.3: Hierarchische Aufteilung eines Terrains in LOD-Stufen

teile, deren Umgehung oft einige Mühe kosten kann. Heikel ist dabei vor allen Dingen der Übergang zwischen den Detailstufen, bei dem nach Möglichkeit kein sichtbarer Übergang entstehen sollte. Dies betrifft nicht nur das plötzliche Umschalten von einer Stufe zur nächsten bei einem kontinuierlichen Zoom, sondern auch das gleichzeitige Darstellen mehrerer Stufen nebeneinander (wenn ein Tile weiter von der Kamera entfernt ist, als ein anderes). Dabei treten folgende Probleme auf:

- Auch wenn Tiles verschiedener LOD-Stufen die selbe Meshauflösung haben, ist die effektive räumliche Auflösung unterschiedlich, da die Tilestufen unterschiedlich viel Terrain beinhalten. Dadurch können Lücken im Terrainmesh entstehen, das aus den einzelnen Tiles zusammengesetzt wird.

- Aus dem selben Grund unterscheidet sich auch die Auflösung der Texturen, so dass ein plötzlicher Wechsel von scharfen zu unscharfen Texturen auftreten kann.

- Durch das immer feinere Aufteilen des Terrains wird auch die Textur in immer kleinere Teile unterteilt. Dadurch entstehen zahlreiche Texturkanten, an denen es zu Fehlern bei der Filterung von Randtexeln kommen kann.

Insgesamt ist ein LOD-Verfahren für die interaktive Darstellung großer DEMs zwar zwingend nötig, liegt aber nicht im Fokus dieser Arbeit, da es hierfür bereits zahlreiche, gut funktionierende Lösungen gibt.

3.1.3 Zugriff auf die Pixel eines Kernels

Für einige Verfahren (Approximation durch quadratische Gleichung, Rauhheit, Offenheit) müssen die Höhendaten in der Nähe des aktuellen Pixels betrachtet werden. Idealerweise kennt man den aktuellen Pixel und möchte jetzt die genauen Werte der umliegenden Pixel berechnen.

Allerdings kann man nur über Texturkoordinaten auf die Pixel einer Textur zugreifen. Liegen diese zwischen zwei Pixeln, wird der Rückgabewert aus den umliegenden Pixeln interpoliert (je nach Texturfiltereinstellung). Kennt man allerdings die Texturgröße, kann man die Texturkoordinaten um $1/Size$ verschieben und erhält so die Koordinaten des Nachbarpixels. Wenn allerdings die ursprünglichen Texturkoordinaten zwischen 2 Pixeln lagen, trifft man auch nicht die Nachbarpixel genau. Man könnte durch einige zusätzliche Berechnungen zwar die Pixelmittelpunkte berechnen, würde dadurch aber auch den Vorteil der Texturfilterung verlieren.

3.2 Bekannte Verfahren

Dieser Abschnitt stellt einen Überblick über die gängigen und im Terrain-Explorer implementieren Verfahren dar.

3.2.1 Höhenfarbverlauf

Eine der einfachsten Möglichkeiten, ein dreidimensionales Gebiet auf eine zweidimensionale Karte abzubilden ist es, jeder Höhe eine andere Farbe zuzuordnen. Gängige Farbgradiente gehen von einem Grünton in den untersten Gebieten über Gelb und Rot bis zu Weiß in den Bergspitzen. Ein Beispiel kann man in Abbildung 2.3 sehen, wo die Täler hellgrün sind, die Hänge dunkelgrün und die Bergspitzen grau bis rot.

Im Terrain-Explorer kann man die minimale und maximale Höhe einstellen. So kann die Genauigkeit der Ausgabe an das untersuchte Gebiet angepasst werden, Werte die außerhalb dieses Bereiches liegen, sind dann aber nicht mehr unterscheidbar.

3.2.2 Direktionale Beleuchtung

Die direktionale Beleuchtung ist eine Vereinfachung des klassischen Phong-Beleuchtungsmodells aus der Computergraphik. Dabei setzt sich die Helligkeit eines Objektes aus dem Ambient- (Grundhelligkeit), dem Diffuse- (direkte Beleuchtung durch eine Lichtquelle) und dem Spekularanteil (Spiegelung der Lichtquelle im Objekt) zusammen.

Bei der direktionalen Terrainbeleuchtung wird die Menge an Licht berechnet, die von einer Lichtquelle auf dem Objekt eintrifft, also der Diffuseanteil des Phongmodelles ohne Berücksichtigung von Materialeigenschaften oder Lichtfarbe. Da von einer unendlich weit entfernten Lichtquelle ausgegangen wird, hängt die Lichtstärke nicht von der Position ab (da die Strahlen nicht schwächer werden

und parallel sind) sondern nur von dem Winkel zwischen Oberflächennormale und Lichteinfall. Somit ergibt sich die Helligkeit direkt aus dem Skalarprodukt dieser 2 Vektoren ($I = \vec{L} \cdot \vec{N}$).

Dadurch entsteht allerdings das Problem, dass Geländemerkmale, die parallel zum Lichteinfall liegen, deutlich schlechter sichtbar sind also solche, die senkrecht zu ihm liegen [SC05]. Dieses Problem kann man beispielsweise lösen, indem man 2 Ausgabebilder erzeugt, jeweils eins mit der Lichtrichtung parallel und orthogonal zu den Hauptcharakterzügen [SC05]. Eine andere Möglichkeit ist es, mehrere Farbkanäle in einem einzelnen Bild zu benutzen, worunter aber die Übersicht leiden kann und wodurch ein Farbbild benötigt wird.

Der Terrain-Explorer erlaubt die Einstellung der Himmelsrichtung und Höhe der virtuellen Sonne. Dadurch lassen sich beide Problemlösungen anwenden, da der Terrain-Explorer sowohl das schnelle Umschalten der Lichtrichtung als auch das gleichzeitige Visualisieren mehrerer Lichtquellen unterstützt.

3.2.3 Steigung

Die Steigung des Geländes ergibt sich aus der ersten Ableitung des Geländes, also aus 3.1.

$$\frac{\delta Z}{\delta X} = S_x = 2aX + cY + d$$
$$\frac{\delta Z}{\delta Y} = S_y = 2bY + cX + e$$

Wertet man diese Funktion im Nullpunkt aus (was immer der Fall sein wird, da man für jeden Pixel einen neuen Kernel berechnet und der Pixel immer in der Mitte des Kernels liegt), vereinfachen sich die Formeln zu $S_x = d$ und $S_y = e$. Nach [WOB+07] ergibt sich die Steigung in diesem Fall aus:

$$slope = \arctan(\sqrt{d^2 + e^2})$$

Im Terrain-Explorer kann man eine Skalierung der Steigung einstellen, um so die Ausgabe an unterschiedliche steile Gebiete anzupassen. Außerdem kann man die Kernelgröße wählen.

3.2.4 Krümmung

Die Krümmung des Geländes ergibt sich aus der zweiten Ableitung von Gleichung 3.1. Allerdings kann man einem Pixel keinen eindeutigen Krümmungswert zuordnen, da man das Gelände in verschiedene Richtungen ableiten kann; man muss sich also für eine entscheiden. Die beiden am häufigsten verwendeten sind die Profil-Krümmung (*profile curvature*) und die Plankrümmung (*plan curvature*) [WOB+07]. Die Profilkrümmung ist die Krümmung des Geländes entlang der steilsten Neigungsrichtung, die Plankrümmung die Krümmung entlang einer Höhenlinie. Die entsprechenden Formeln lauten nach [WOB+07]:

$$profc = \frac{-2(ad^2 + be^2 + cde)}{(e^2 + d^2)(1 + e^2 + d^2)^{1.5}} \tag{3.3}$$

$$planc = \frac{2(bd^2 + ae^2 - cde)}{(e^2 + d^2)^{1.5}} \tag{3.4}$$

(In [WOB+07] werden diese Formeln jeweils noch mit 100 multipliziert, damit das Ergebnis der Steigung des Geländes in % entspricht.)

Im Terrain-Explorer ist die Krümmung entlang der steilsten Neigungsrichtung implementiert und kann mit einem Skalierungsparameter an das jeweilige Gelände angepasst werden. Ein Ausgabewert von 0.5 bedeutet dabei ein ebenes Gelände, kleinere Werte zeigen konkave Gebiete an, größere konvexe.

3.2.5 Rauheit

Die Rauheit eines Geländes beschreibt, wie uneben es ist, bzw. wie stark sich die Höhe eines Pixels von der Höhe umliegender Pixel unterscheidet. Die Rauheit eines Geländes kann beispielsweise Aufschluss über das Alter geben, da Geländestrukturen mit der Zeit verwittern und somit glatter werden [GSR11].

Es gibt dazu in der Literatur eine ganze Reihe unterschiedlicher Definitionen und Algorithmen, von denen viele in [GSR11] beschrieben und miteinander verglichen werden.

Eine Methode ist beispielsweise die aus der Mathematik bekannte Standardabweichung, die man auf die Höhendifferenz der umliegenden Pixel und den zu untersuchenden Pixel anwendet.

$$SD = \frac{1}{k} \cdot \sqrt{\sum_{i=1}^{k} (H_i - H)^2}$$

Eine Variante ist es, nicht die tatsächliche Höhendifferenz zu verwenden, sondern die Entfernung zu einer an diesem Punkt am besten anliegenden Ebene zu berechnen. Auf diese Weise werden konstanten Steigungen nicht als rau erkannt. Bild 3.4 verdeutlicht diesen Unterschied. Eine ähnliche Variante ist es, die Standardabweichung auf den Unterschied zwischen dem ursprünglichen und dem geglätteten Gelände anzuwenden [GSR11].

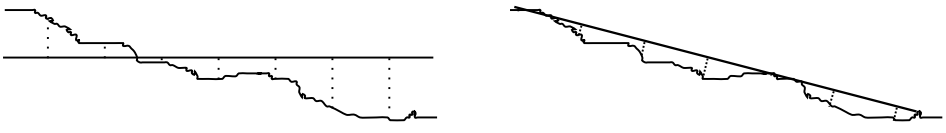

Bild 3.4: Unterschied zwischen Höhendifferenz und Entfernung zur anliegenden Ebene bei der Rauheitsberechnung

Desweiteren kann man die Krümmung, also die zweite Ableitung des Geländes untersuchen, um Sprünge in der Steigung festzustellen [GSR11].

Weitere Methoden sind die Analyse der Normalvektoren, die Bestimmung der fraktalen Dimension der Oberfläche, Fourier Analyse oder Geostatistik [GSR11]. In [PRBY] werden noch einige weitere Rauheitsverfahren beschrieben; nämlich die mittlere Differenz zwischen dem untersuchten und den umgebenden Pixeln, die Differenz zwischen dem untersuchten und dem Mittelwert der umgebenden Pixel und die größte Entfernung des untersuchten Pixels zu einem der Nachbarpixel.

3.2.6 Kantenerkennung

Mittels Kantenerkennung (engl. Edge Detection) können Kanten in Bildern gefunden werden. Dies ist beispielsweise nützlich, um Rastergrafiken in Vektorgrafiken umzuwandeln, Objekte in Bildverarbeitungsprogrammen freizustellen, oder Formen zu erkennen. Ein bekanntes Verfahren dazu ist der Sobel-Filter.

Bei der Konzeption dieser Arbeit wurde überlegt, ob es sinnvoll ist, dieses Verfahren zu implementieren. Allerdings gibt es in natürlichen Landschaften in der Regel keine harten Kannten (diese würden sich beispielsweise in dem Höhenbild einer Stadt finden), so dass kaum sinnvolle Ergebnisse zu erwarten sind. Wie sich aber herausstellte, liefert die Krümmung Bilder, die der Anwendung eines Kantenerkennungsfilters ähneln.

3.2.7 Offenheit

Die Offenheit eines Geländes gibt an, ein wie großer Teil des Himmels zu sehen ist. Die Ergebnisse ähneln einer diffusen Beleuchtung, bei dem das Terrain von einem bewölkten Himmel gleichmäßig beleuchtet wird, so dass Täler dunkler erscheinen als Bergspitzen. Im Gegensatz zur direktionalen Beleuchtung wird hierbei aber keine explizite Lichtquelle benötigt, wodurch keine Details durch eine schlechte Lichtquellenwahl verschwinden können [ZOK11].

Zur Berechnung der Offenheit gibt es unterschiedliche Algorithmen mit teilweise unterschiedlichen Bezeichnungen, etwa *Openess* [YSP02] oder *Sky-View-Factor* [ZOK11]. Im Folgenden wird die Offenheit nach der Definition von [YSP02] beschrieben.

Die Offenheit eines Punktes ergibt sich aus 2 Werten, dem maximalen Zenitwinkel und dem maximalen Nadirwinkel, die jeweils für 8 Richtungen berechnet und gemittelt werden. Der Zenitwinkel gibt an, auf welcher Höhe aus Sicht eines Betrachters der Boden aufhört und der Himmel anfängt, der Nadirwinkel beschreit das selbe auf der Unterseite des Terrains, wie man in Bild 3.5 sehen kann.

Dabei wird jedoch nur der maximale Winkel bis zu einer gewissen Entfernung berechnet, im Bild werden 5 Abtastpunkte betrachtet. Zu jedem dieser Punkte wird ein Winkel θ durch folgende Formel berechnet:

$$\theta = \tan^{-1}\left(\frac{H_B - H_A}{P}\right) \tag{3.5}$$

Das Maximum und Minimum dieses Winkels wird dabei über die Schritte hinweg gespeichert. Im Bild ist zu erkennen, dass der maximale Winkel eigentlich etwas größer sein müsste, aber das Gelände

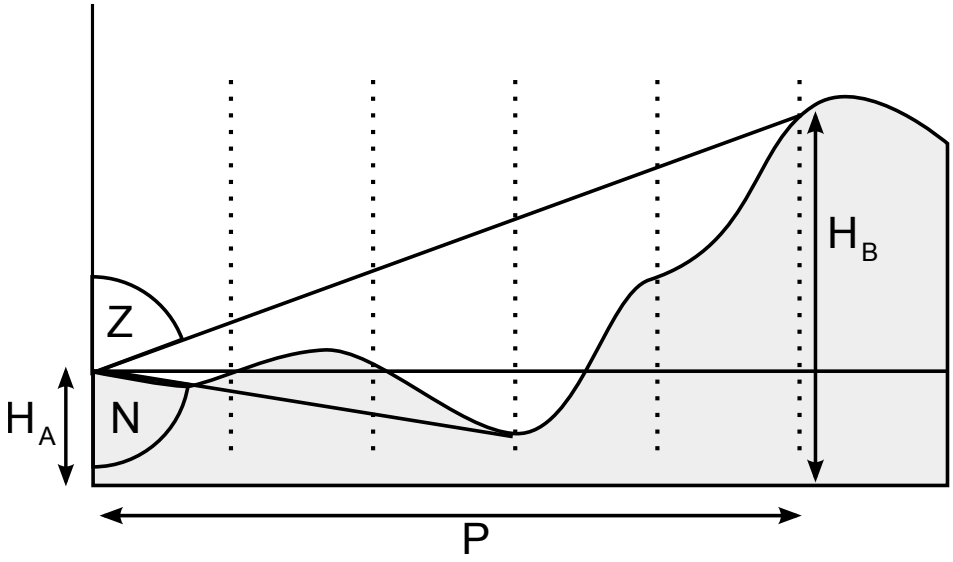

Bild 3.5: Bestimmung des maximalen Zenitwinkels

nicht weit genug abgetastet wird, um die Bergspitze in Betracht zu ziehen.

Der Zenitwinkel ergibt sich dann durch $Z = 90 - max(\theta)$, der Nadirwinkel aus $Z = 90 - min(\theta)$ (man beachte, dass der minimalen Winekl θ in der Regel negativ ist).

Die Offenheit des Geländes ergibt sich schließlich aus dem durchschnittlichen Zenitwinkel oder Nadirwinkel der 8 Himmelsrichtungen.

Im Terrain-Explorer wurde der Zenitwinkel implementiert. Über einen Parameter kann die Variable P skaliert werden (dies beeinflusst nicht, welche Pixel benutzt werden, sondern nur das Ergebnis von θ), wodurch die Ausgabe an unterschiedlich steile oder flache Gebiete angepasst werden kann.

3.2.8 Richtung

Die Richtung ist ein Wert zwischen $0°$ und $360°$ und repräsentiert den Azimuth (horizontale Ausrichtung des Geländes) [PRBY]. Konsequenterweise ist die Richtung in horizontalen Flächen undefiniert.

Die Richtung des Geländes wird über dessen Normalvektor berechnet. Zuerst wird dessen Z Komponente auf 0 gesetzt, das Ergebnis wird anschließend normalisiert.

Da 0 Grad Richtung Norden zeigt, muss der Winkel zur Y-Achse bestimmt werden. Den Winkel zwischen 2 Vektoren kann man mithilfe des Skalarproduktes berechnen, da der Richtungsvektor der Y-Achse allerdings $(0, 1)$ ist, erhält man den Winkel direkt aus dem Kosinus des Normalvektors (da beide Vektoren bereits die Länge 1 haben). Allerdings muss noch das Vorzeichen der Y-Komponente überprüft werden, um herauszufinden, in welchem Halbkreis der aktuelle Winkel liegt.

Alternativ kann man die Richtung des Geländes auch mithilfe der Funktion atan2 und der X- und Y-Komponente des Normalvektors als Parameter berechnen. Man benötigt allerdings trotzdem eine trigonometrische Funktion und eine Fallunterscheidung.

```
1  Direction(Normal) =
2    Direction=(Normal.x, Normal.y, 0)
3    normalize(Direction)
4
5    if(Direction.y >= 0)
6      angle = acos(Direction.x) / 2*PI;
7    else
8      angle = -acos(Direction.x) / 2*PI;
9
10   return angle
```

Listing 3.1: Berechnung der Richtung

```
1  Direction(Normal) =
2    angle = atan(Normal.y, Normal.x)
3    if(angle < 0)
4      angl e= angle + 2*PI
5    angle = angle / 2*PI
6
7    return angle
```

Listing 3.2: Alternative Berechnung der Richtung

3.2.9 Normalvektor

Der Normalvektor ist ein Vektor der senkrecht auf dem Terrain steht, auf einer an dieser Stelle anliegenden Ebene. Obwohl er meist als Richtungsvektor mit 3 Koordinaten gespeichert wird, ist es nur eine zweidimensionale Größe, da er ja immer die Länge 1 hat. Eine Visualisierung dieser 3 Koordinaten entspricht einer klassischen Normalmap aus der Computergraphik.

Dieses Verfahren wurde im Terrain-Explorer nicht implementiert, da es aufgrund seiner Zweidimensionalität schwer ist, ihm einen einzelnen Wert im Intervall $[0..1]$ zuzuordnen. Da sich seine zwei Dimensionen aber aus der Richtung und der Steigung des Terrains zusammensetzen, können die Informationen, die der Normalvektor liefert, trotzdem visualisiert werden.

3.3 Der Terrain-Explorer

3.3.1 Programmiersprache und Bibliotheken

Als Programmiersprache stand von Anfang an C++ fest. Gründe dafür sind zum einen, dass C++ in Grafikanwendungen weite Verbreitung findet und einige der benutzten Bibliotheken für C++ geschrieben wurden, zum anderen die praktische Erfahrung des Autors aus vergangenen Projekten in C++.

Darüber hinaus fiel die Wahl schnell auf *Qt* als GUI-Framework. Qt bietet viele Highlevel-Funktionen zur Entwicklung grafischer Applikationen und ist darüber hinaus auf allen großen Plattformen verfügbar. Außerdem verfügt Qt über ausgereifte Tools, etwa den Qt-Designer, mit dem die Programmoberfläche

in einem WYSIWYG-Editor bearbeitet werden kann.

Für zusätzliche Komfortfunktionen wird neben der C++ Standardbibliothek auch *boost* benutzt, insbesondere die darin enthaltene Serialisierungsbibliothek zum Speichern der Konfigurationen.

Für die grafische Ausgabe fiel die Wahl nach einiger Zeit auf die *Open Scene Graph*-Bibliothek. Deren Hauptaugenmerk liegt zwar auf der Verwaltung komplexer Szenenstrukturen, (was in diesem Fall eher uninteressant ist, da das Terrain das einzige Objekt in der Szene ist) bietet aber auch zahlreiche Hilfsfunktionen, etwa eine fertige Kamerasteuerung oder eine Klasse zum Verwalten von Terrains.

Ein grundsätzliches Problem, das bei der Auswahl der Werkzeuge adressiert werden musste, war die Tatsache, dass viele DEMs extrem viel Speicherplatz benötigen. Moderne Grafikkarten haben nur eine Speicherkapazität in der Größenordnung eines Gigabytes, für DEMs ist die zehnfache Größe aber durchaus üblich. Zudem sollte die Ladezeit natürlich nicht unverhältnismäßig lang sein.

Daher schied es von Anfang an aus, das komplette Terrain beim Programmstart zu laden. Überhaupt sieht der Benutzer ja immer nur einen Ausschnitt des Terrains oder hat relativ weit herausgezoomt, so dass ein *LOD*-Verfahren sinnvoll ist. Und OSG bietet mit seiner Terrainklasse ein solches. Dennoch musste zunächst überprüft werden, ob OSG auch die anderen Anforderungen erfüllen kann.

Ein sehr wichtiger Punkt ist dabei natürlich die Integration in Qt, denn normalerweise erstellt der OsgViewer ein eigenes Hauptfenster um die Szene anzuzeigen. Glücklicherweise gibt es aber auch eine Osg-Qt Schnittstelle, die einen *osgViewer* in ein *QWidget* integriert. Danach waren nur noch ein paar Anpassungen nötig, insbesondere um die aktuelle Pixelfarbe für das Interpretationsfenster auslesen zu können. Außerdem musste sichergestellt werden, dass die Geländedaten korrekt geladen werden und es möglich ist, beliebige Shader zu setzen.

Ein Verzicht auf OSG hätte zwar die Einarbeitung in eine komplexe Bibliothek vermieden, aber auch gleichzeitig bedeutet, dass ein eigenes LOD-System implementiert werden muss. Obwohl dies sicherlich eine reizvolle Aufgabe ist, wäre dabei doch ein erheblicher Zeitaufwand entstanden, der aber keine für diese Arbeit relevanten Erkenntnisse erbracht hätte.

Desweiteren wurde CMake für die Projektverwaltung benutzt. Mithilfe von CMake ist es möglich, die externen Abhängigkeiten und Projekteinstellungen automatisch zu konfigurieren. Obwohl das Projekt komplett unter Windows entwickelt wurde, war es dank CMake und der Beschränkung auf plattformunabhängige Bibliotheken möglich, den Terrain-Explorer mit minimalen Änderungen unter Linux zu kompilieren.

3.3.2 Die Oberfläche

Dieses Kapitel beschreibt die Benutzeroberfläche des Terrain Explorers und erklärt dessen Funktionen.

Bild 3.6 zeigt das Hauptfenster des Terrain Explorers. Neben der Terrainansicht in der rechten Fensterhälfte befindet sich auf der linken Seite eine Toolbar über die alle Einstellungen getroffen werden können:

Visualization Height: Über diesen Schieberegler kann die Höhe des Terrains geändert werden, also

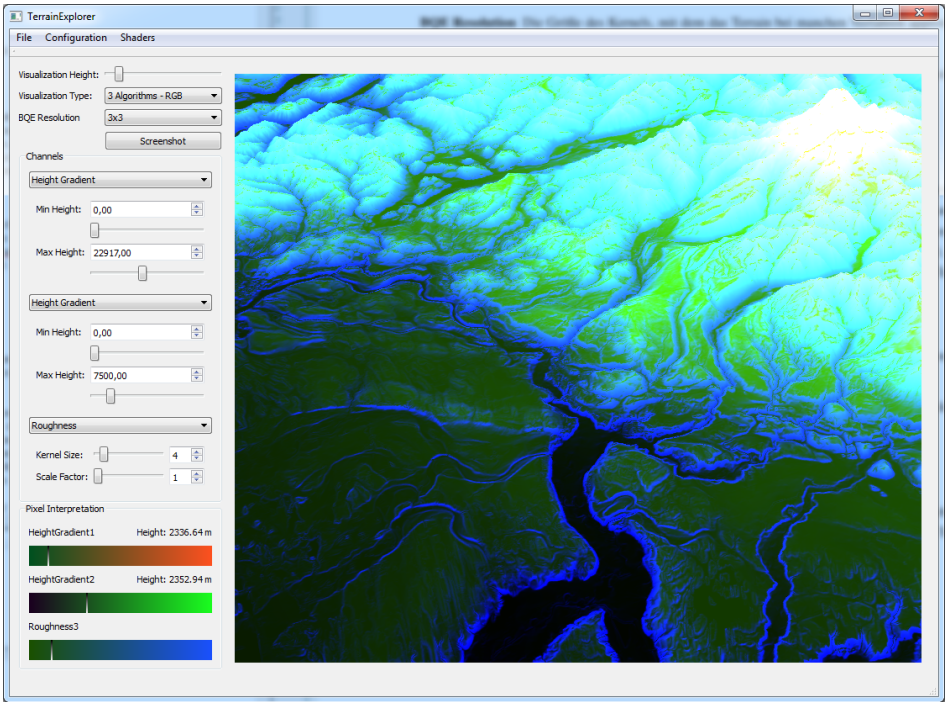

Bild 3.6: Die Programmoberfläche des Terrain Explorers

der Faktor um den das Terrain entlang der Z-Achse gestreckt wird. Das ist nützlich, wenn das Gelände falsch exportiert wurde und zu steil oder zu flach ist, oder um beispielsweise einen eigentlich flachen Hügel in der 3D Ansicht deutlicher sichtbar zu machen. Die Visualisierungshöhe beeinflusst dabei nur die Form des Terrains und beeinflusst nicht die Berechnung der unterschiedlichen Verfahren.

Visualization Type: Hier kann der Modus des Viewers ausgewählt werden. Näheres über die Modi findet man in Kapitel 3.3.3.

BQE Resolution: Die Größe des Kernels, mit dem das Terrain bei manchen Verfahren approximiert wird (siehe 3.1.1).

Screenshot: Eine Komfortfunktion um die aktuelle Ansicht des Viewers als PNG-Datei zu speichern.

Die weiteren Bedienelemente sind von dem aktuellen Modus abhängig.

Die Kamera wird komplett über Mausgesten im Ansichtsfenster gesteuert. Ein ziehen des Mauszeigers bewirkt eine Drehung der Kamera, mithilfe des Mausrades kann herein- und herausgezoomt werden.

3.3.3 Die verschiedenen Modi

Ein zentraler Aspekt des Terrain Explorers ist die Möglichkeit, Verfahren beliebig miteinander zu kombinieren. So kann einerseits der Zusammenhang zwischen verschiedenen Größen analysiert wer-

den und andererseits eine bessere Interpretation der Geländestruktur ermöglicht werden. Die Verfahren wurden dabei so implementiert, dass jedes über eine einheitliche Schnittstelle einen Wert aus dem Intervall $[0..1]$ liefert. Der Betrachter implementiert dabei 3 verschiedene Modi um die endgültige Farbe des Terrains zu berechnen:

RGB: Es wird jedem Farbkanal (rot, grün blau) ein Verfahren zugeordnet

HSV: Ähnlich wie bei dem RGB Modus, werden die 3 Werte dem Farbton (*Hue*), der Sättigung (*Saturation*) und der Helligkeit (*Value*) zugeordnet

Gradient: Hier wird nur ein Verfahren benutzt, dessen Wert als Texturkoordinate einer Gradiententextur interpretiert wird.

Es wäre auch denkbar, weitere Farbräume zu implementieren, etwa CMYK oder HSL. Diese sind aber sehr ähnlich zu den beiden implementierten Farbräumen, so dass aus deren Implementation keine Verbesserung der Farbinterpretation zu erwarten ist.

3.3.4 Gleichzeitige Visualisierung mehrerer Verfahren

Die ersten beiden Modi unterscheiden sich nur in der Art und Weise, wie die endgültige Farbe aus den 3 Ausgabewerten der gewählten Verfahren berechnet wird; die Kontrollelemente in der Toolbar sind die selben.

Für jeden der 3 Kanäle kann man ein Verfahren auswählen und dessen Parameter ändern. Die Kontrollelemente für die Parameter passen sich dabei dynamisch an das gewählte Verfahren an. Viele Parameter haben mehrere Kontrollelemente, so kann man beispielsweise die minimale und maximale Höhe des Höhengradienten sowohl direkt als Zahl eintragen, als auch über einen Schieberegler einstellen. Letzteres ist zwar etwas ungenauer, bietet aber die Möglichkeit den Wert sehr schnell auf die ungefähr gewünschte Größe einzustellen. Ähnliches gilt auch für andere Parameter.

Visualisiert man 3 unterschiedliche Verfahren, kann das Bild sehr wirr aussehen und es wird schwierig zu verstehen, was genau man eigentlich sieht. Bild 3.6 zeigt beispielsweise 2 verschiedene Höhengradiente und die Offenheit des Terrains. Aus den Einstellungen weiß der Benutzer aber nur, dass die Werte auf den Rot-, Grün- und Blaukanal abgebildet werden, aber nicht, welche Farben bei welchen Kombinationen auftreten würden.

Da der Farbraum dreidimensional ist, ist es nicht möglich, eine Skala in einer zweidimensionalen Darstellung anzugeben. Stattdessen wird gewissermaßen die Ableitung der Farbe unter der aktuellen Cursorposition nach den 3 Kanälen berechnet, also wie sich die Farbe verändern würde, wenn man den Wert eines Kanales ändern würde.

Die oberste Anzeige ist dem ersten Verfahren und damit dem Rotkanal zugeordnet. Der Verlauf zeigt an, wie sich die Farbe an der Stelle des Mauszeigers ändern würden, wenn die Berechnung des ersten Verfahrens andere Werte liefern würde. Der Grünton auf der linken Seite enthält demzufol-

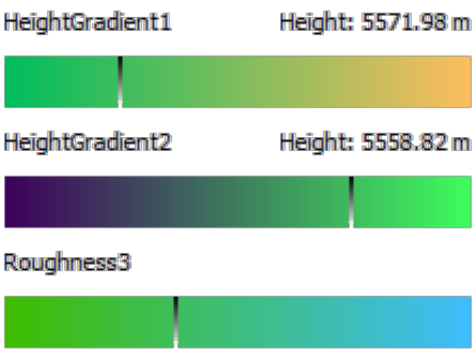

Bild 3.7: Die Ausgabe der Interpretierungsanzeige

ge überhaupt kein Rot, während im Orangeton auf der rechten Seite der Rotanteil maximal ist. Der vertikale Schwarzweißverlauf zeigt dabei die aktuelle Farbe an.

Bei manchen Verfahren ist es möglich, anhand der Ausgabefarbe den tatsächlich berechneten Wert zu bestimmen. Dieser wird dann bei dem entsprechenden Kanal im Interpretationsfenster angezeigt. In 3.7 gibt es beispielsweise 2 Höhengradiente, die unterschiedliche Höhenbereiche abdecken. Da der Höhenbereich des ersten Gradienten wesentlich höher ist, ist der Rotwert an der selben Stelle deutlich niedriger als der Grünwert, da dieser sein Maximum sehr viel früher erreicht. Die berechnete Höhe ist aber dennoch in etwa gleich, die Unterschiede resultieren nur aus der geringen Farbtiefe des Ausgabebildes (Der Höhenbereich umfasst einige tausend Meter, es können aber nur 256 Rotwerte angenommen werden).

Anhand der Gradienten kann man nicht nur ablesen, welche Werte der aktuelle Pixel hat, sondern auch abschätzen, welche Farben andere Kombinationen ergeben. So hat zum Beispiel ein hellblauer Pixel die selbe Höhe, aber das Gelände in seiner Umgebung ist deutliche rauer.

Bewegt man den Mauszeiger, ändern sich die 3 Gradienten der neuen Farbe entsprechend:

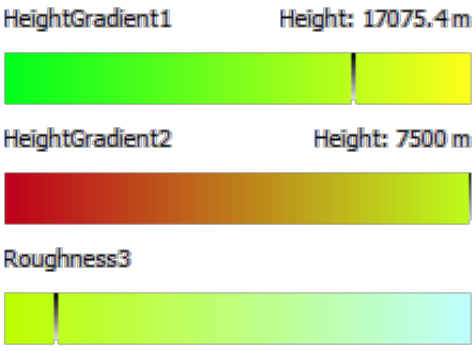

Bild 3.8: Eine weitere Ausgabe der Interpretierungsanzeige an einer anderen Stelle

Dieser Pixel liegt offenbar wesentlich höher. Da der zweite Höhengradient sein Maximum bereits erreicht hat, kann der berechnete Höhenwert nicht mehr stimmen und liegt daher wesentlich unter

dem Ergebnis des ersten Gradienten. Da die ersten beiden Werte voneinander abhängig sind, wird im Ausgabebild der Orangeton aus 3.7 nie erreicht. Anders sieht es mit den Farben aus dem dritten Gradienten aus, in Bild 3.6 erkennt man deutlich die Hellblau- und Hellgrün-Töne in den höheren Lagen, die dementsprechend die unterschiedlich rauen, aber gleich hohen Bergregionen markieren.

Dadurch, dass sich die Gradienten der Verfahren bei jeder Mausbewegung ändern, ist das Interpretationsfenster zunächst recht verwirrend, aber nach einiger Eingewöhnung kann man dank ihm die Farben sehr viel schneller interpretieren, anstatt im Kopf zu überlegen, wie die einzelnen Farbmischungen wohl entstehen mögen.

Im HSV-Modus funktioniert das Interpretationsfenster analog, nur sehen die Gradienten entsprechend anders aus:

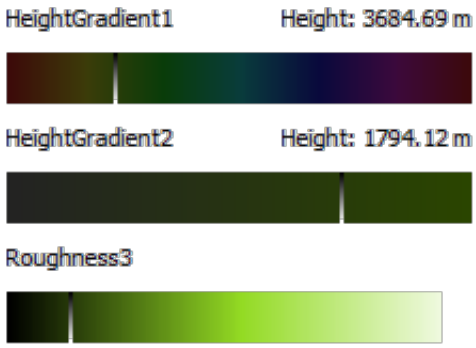

Bild 3.9: Ausgabe der Interpretierungsanzeige im HSV-Modus

An den Gradienten gut zu erkennen sind die 3 Dimensionen des HSV-Farbraums:

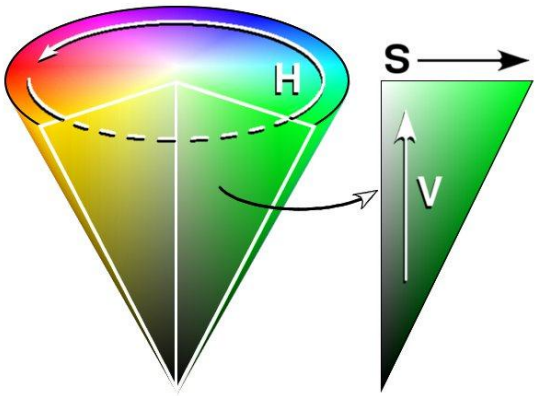

Bild 3.10: Der HSV-Farbraum
`http://de.wikipedia.org/w/index.php?title=Datei:HSV_cone.jpg`, CC BY-SA 3.0 Fanghong

Der erste Gradient ist der Farbton, also der Winkel im Kegel. Der zweite ist die Sättigung, also der Abstand vom Mittelpunkt und der dritte die Helligkeit, also die Höhe.

3.3.5 Gradientenvisualisierung

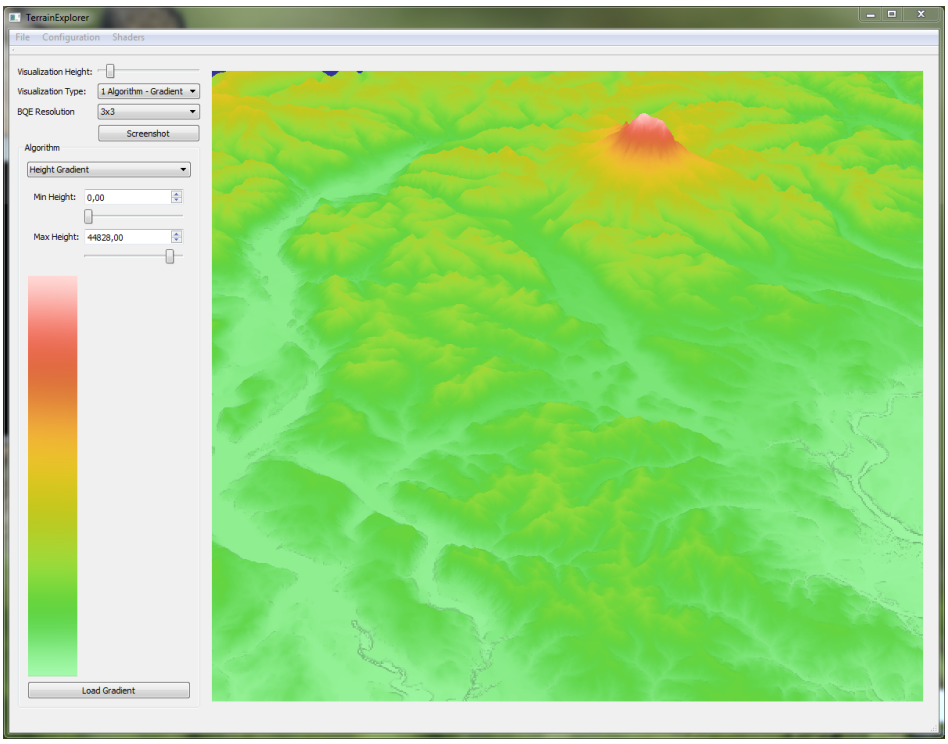

Bild 3.11: Die Programmoberfläche des Terrain Explorers im Gradientenmodus

In diesem Modus kann nur ein einzelnes Verfahren visualisiert werden. Der Benutzer kann das Verfahren auswählen, dessen Parameter ändern und einen Gradienten (in Form einer Bilddatei) laden. Der Ausgabewert des Verfahrens wird dann als Texturkoordinate benutzt.

Da in diesem Modus einerseits keine Farben gemischt werden und andererseits eine Farbe im Gradienten mehrfach vorkommen kann (und somit keine eindeutige Umkehrfunktion existieren muss), entfällt das Interpretationsfenster in diesem Modus.

3.3.6 Erweiterungsmöglichkeiten durch den Shader Editor

Der Shader Editor bietet die Möglichkeit, die generierten Vertex- und Fragmentshader anzuschauen und zu bearbeiten. Zwar können dabei keine neuen Parameter definiert werden, die man dann über GUI-Elemente steuern könnte, aber man kann Konstanten im Shadereditor jederzeit neu setzen und auch die vorhandenen Parameter neu verwenden.

Für größere Änderungen ist es natürlich sinnvoller, das Programm selbst zu erweitern, aber durch den Shader Editor können neue Ideen schnell getestet werden (Prototyping).

Im folgenden Beispiel wird die Steigung des Geländes in x und y Richtung visualisiert. Wie in 3.2.3 erläutert, folgt diese direkt aus den Koeffizienten der Approximationsparabel, was zu folgendem Quellcode führt:

```
1 void main()
2 {
3         InitializeBQE(gl_TexCoord[0], 256, Height);
4         vec3 Color=vec3(bqeD(), bqeE(), Constant3());
5         gl_FragColor=vec4(Color, 1.0);
6 }
```

Listing 3.3: Visualisierung der Geländesteigung in X und Y Richtung

Bild 3.12 zeigt das Ergebnis.

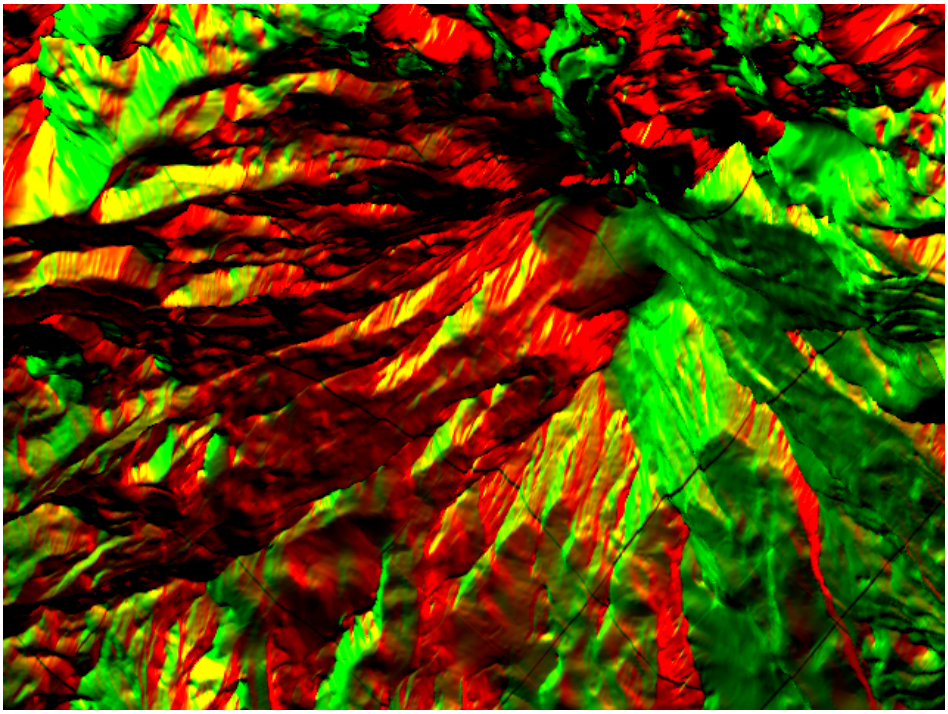

Bild 3.12: Visualisierung der Geländesteigung in X und Y Richtung

3.3.7 Speichern und Laden von Konfigurationen

Über das Programmmenü ist es möglich, verschiedene Konfigurationen zu speichern. Unter einer Konfiguration wird die Auswahl der Verfahren, sowie deren Parameter verstanden. Wenn man also alle Parameter der Landschaft entsprechend angepasst hat und somit eine Konfiguration gefunden hat, die bestimmte Eigenschaften des Geländes gut hervorhebt, kann man diese speichern und später wieder laden. Ebenso ist es möglich, schneller zwischen verschiedenen Konfigurationen hin und her zu wechseln, so dass man sie interaktiv miteinander vergleichen kann.

Kapitel 4

Implementation

Dieses Kapitel beschreibt die Implementation des Terrain Explorers.

4.1 Aufbereiten und Laden des Terrains

Eine wichtige Frage ist es, in welchem Format das Terrain gespeichert werden soll. Glücklicherweise bietet OSG dafür ein *osgdem* genanntes Tool an, dass eine Vielzahl an Eingabeformaten lesen kann und eine (bzw. mehrere) von OSG lesbare Ausgabedatei erstellt. Das tatsächliche Laden des Modells beschränkt sich damit auf nur noch wenige Zeilen Code:

```
1 ref_ptr<Node> model=osgDB::readNodeFile(Filename.toStdString());
2 m_Transf->addChild(model);
3 m_Viewer->setSceneData(m_Transf);
```

Listing 4.1: Laden eines Terrains in OSG

Da nur ein einzelnes Terrain (auch wenn es intern aus vielen Tiles und LOD-Stufen besteht) gerendert wird, besteht die Szenenhierarchie nur aus dem Terrain selber und eines Transformationsknotens, der benutzt wird, um das Terrain entlang der Z-Achse zu strecken.

Für die Berechnung der einzelnen Verfahren benötigt man in irgendeiner Weise Zugriff auf die Höheninformationen. Nach dem Laden liegt das Terrain zwar als Polygonnetz im Speicher, aber die Auflösung ist verhältnismäßig gering. OSG lädt zwar selbständig eine LOD Stufe, die so groß ist, dass die Silhouette keine auffallenden Kanten beinhaltet, die einzelnen Dreiecke sind allerdings trotzdem mehrere Pixel groß. Außerdem kann man im Vertexshader nur auf den aktuellen Vertex zugreifen, was für die meisten Verfahren nicht ausreicht.

Daher ist es nötig, die Höheninformationen in einer Textur zu speichern, auf die man dann im Fragmentshader vollen Zugriff hat. Dabei ist darauf zu achten, dass diese eine ausreichende Präzision hat, es werden also Fließkommatexturen benötigt. Den entsprechenden osgdem-Aufruf sieht man in Listing 4.2.

Die einzelnen Parameter haben folgende Bedeutung:

```
1 osgdem -l 8 -v 0.02 -d C:\TerrainExplorer\puget\ps_height.png \
2         --RGB32F --image-ext tiff -t \
3         C:\TerrainExplorer\puget\heights.tiff \
4         -o C:\TerrainExplorer\puget\Testing.osg
```

Listing 4.2: osgdem Aufruf

-l legt die maximale Tiefe der Tilehierarchie fest. Es werden also maximal $4^{8-1} = 16384$ Tiles erzeugt (Da die oberste Ebene aus einem und nicht vier Tiles besteht).

-v legt den Skalierungsfaktor entlang der Z-Achse fest. Auch wenn diese nachträglich im Terrain Explorer angepasst werden kann, ist es sinnvoll, einen passenden Startwert zu wählen.

-d legt die Eingabedatei der Höhendaten fest.

–RGB32F –image-ext tiff gibt das Format ($3 \cdot$ 32bit float Kanäle pro Pixel) und den Typ (tiff) für die nächsten Texturen an.

-t definiert eine Textur für das Terrain.

-o legt die Ausgabedatei fest. Dies ist nur die Hauptdatei, die von OSG später geladen wird, für die einzelnen Tiles werden weitere Dateien in einem Unterordner erstellt, auf die die Hauptdatei verweist.

Wie man sehen kann, wird das Terrain aus einer Heightmap erstellt, in diesem Fall handelt es sich um eine 16bit Graustufen PNG-Datei. Leider ging diese hohe Genauigkeit beim Konvertieren verloren, so dass später in OSG nur noch 8bit Grauwerte vorhanden waren, womit keine sinnvollen Berechnungen mehr möglich sind.

Um präzisere Texturen in OSG zu erhalten, musst leider ein Umweg über tiff Bilder gegangen werden. Dabei wurde die Heightmap zunächst mit GTA in eine 32bit float tiff-Datei umgewandelt:

```
1 gta from-gdal FILE.png FILE.gta
2 gta component-convert -c float32 FILE.gta | \
3 gta to-gdal FILE.tiff
```

Listing 4.3: Konvertierung der Heightmap mit gta

Außerdem musste der tiff-Loader von OSG angepasst werden, da bei einer Umwandlung die Farbtiefe wieder auf 8bit reduziert wurde.

Nach diesen Schritten war es möglich im Fragment auf die Höhendaten als 32bit float-Wert zuzugreifen.

4.2 Die Programmstruktur

Die wichtigsten Komponenten des Terrain-Explorers sind:

- Das Grundgerüst für das Programm stellt die **TerrainExplorer** Klasse dar. Hier wird das Viewer-Widget und die Verfahrensauswahl verwaltet sowie das Interpretationswidget implementiert.
- Die **ViewerWidget** Klasse für die Darstellung des Terrains ist im wesentlichen aus dem OsgQt Beispiel übernommen, mit Ausnahme einiger zusätzlicher Eventhandler für Mausbewegungen.
- Der **ShaderGenerator** ist für die Erstellung der Vertex- und Fragmentshader zuständig. Diese müssen den Code für die verwendeten Verfahren sowie nötige *Uniform*-Parameter und natürlich die Main-Funktion, in der die endgültige Farbe zusammengesetzt wird, beinhalten.
- Die einzelnen Verfahren werden über abgeleitete Klassen der **ChannelFunction** Klasse implementiert. Sie erstellen jeweils ein Widget zur Bearbeitung der Parameter, liefern die benötigten Codestücke für den Shader Generator und besitzen Methoden zur Serialisierung der Parameter.
- Der **Shader Editor** ist ein einfacher Dialog zum Bearbeiten, Speichern und Laden des aktuellen Vertex- und Fragmentshaders.
- Das Interpretationswidget benutzt die Funktion **GradientGeneration**, um die jeweiligen Farbverläufe zu generieren. Dabei muss unterschieden werden, ob man sich im RGB oder HSV Modus befindet, und gegebenenfalls die Funktionen zum Konvertieren von RGB in HSV und umgekehrt benutzt werden.

Im Folgenden werden die interessanten Programmteile näher beschrieben.

4.3 Shader Generator

Der Shader Generator wird jedesmal aufgerufen, wenn sich die aktuelle Konfiguration ändert. Er besteht lediglich aus 2 Funktionen, die die aktuellen Verfahren und einige zusätzliche Parameter annehmen und den ShaderCode als std::string zurückliefern. Diese werden vom TerrainExplorer kompiliert und als Shader gesetzt.

Damit es nicht zu Namenskonflikten von Variablen aus unterschiedlichen Verfahren kommt, besitzt jedes eine ID, die der Kanalnummer entspricht (diese IDs sind eindeutig, da einem Kanal zu jeder Zeit immer nur ein Verfahren zugeordnet sein kann). Diese ID wird jeder Variable und Funktion als Suffix angehängt, wodurch Namenskollisionen unmöglich werden.

Im wesentlichen beinhaltet der Shader Generator den Shader Quelltext als eine Art Lückentext in den die Codefragmente der einzelnen Verfahren eingesetzt werden. Desweiteren sind einige längere Funktionen (zur Bestimmung der quadratischen Gleichung und Hilfsfunktionen für den HSV-Farbraum) als strings in eine weitere Datei ausgelagert, um den Code übersichtlich zu halten.

Der Einfachheit halber wird der Code zur Approximation des Geländes durch eine quadratische Gleichung immer in den Shader geschrieben, da er von mehreren Verfahren benutzt wird. Sollte er nicht benötigt werden, kümmert sich der Shaderkompiler in einem Optimierungsschritt um dessen Entfernung.

4.4 Die einzelnen Verfahren

Jedes Verfahren definiert ein Widget mit Bedienelementen, über das sich die Parameter in Echtzeit steuern lassen. Für jeden Parameter gibt es dementsprechend eine *uniform* Variable, die bei den entsprechenden Events gesetzt wird.

Auch wenn für die Serialisierung *boost::serialization* benutzt wird, wird nicht das gesamte Objekt gespeichert, sondern nur dessen Parameter. Es werden also insbesondere die Zeiger auf das Hauptprogramm, das Parent-Widget und die Uniforms nicht gespeichert. Diese Variablen mit all ihren Abhängigkeiten zu speichern wäre ein sehr großer und völlig unnötiger Aufwand. Da man diese Zeiger auf jeden Fall später setzen müsste, ist es weitaus einfacher und übersichtlicher, die Objekte selber mit den passenden Werten zu erstellen, anstatt die Objekterstellung der Serialisierungsbibliothek zu überlassen.

Im folgenden werden nur diejenigen Verfahren aufgeführt, die aus Sicht der Implementation interessant sind.

4.4.1 Höhenlinien

Die technische Schwierigkeit dieses Verfahrens liegt darin, dass man nicht einfach Linien zeichnen kann, sondern einzelne Pixel zeichnen muss und für jeden entscheiden muss, ob dieser auf einer Höhenlinie liegt oder nicht. Dadurch, dass man keinerlei Informationen über die Nachbarpixel hat, wird es sehr schwierig, an allen Stellen eine gleichmäßige Liniendicke zu gewährleisten. Auch das hier benutzt Verfahren ist nicht perfekt, bietet aber in den meisten Situationen mit einiger Feinjustierung der Parameter brauchbare Ergebnisse.

Für jeden Pixel wird geprüft, ob er sich in der Nähe einer Höhenlinie befindet, also seine Höhe in einem Intervall um die Höhe einer Höhenlinie liegt. Die Intervallgröße kann über einen Parameter gesteuert werden und entscheidet, wie dick die Höhenlinien werden. Schaut man das Terrain allerdings nicht von oben, sondern von der Seite an, bzw. kippt die Ansicht zu stark, werden aufgrund der perspektivischen Projektion Objekte im Vordergrund größer dargestellt. Somit wären die Linien im Vordergrund deutlich dicker, als diejenigen im Hintergrund. Um im Hintergrund überhaupt noch Linien sehen zu können, müsste man die Intervallgröße so stark anheben, dass die Linien im Vordergrund viel zu dick würden.

Die Lösung liegt nun darin, die Intervallgröße dynamisch von der Entfernung des Pixels zur Kamera abhängig zu machen. Dafür wird die aktuelle Pixelposition ganz normal transformiert wodurch die Entfernung zur Kamera in der neuen Z-Koordinate steht.

Als weitere Verbesserung könnte man zusätzlich die Steigung des Geländes und die Neigung der Kamera betrachten. Da nur die Entfernung zur nächsten Höhenlinie entlang der Z-Achse berechnet wird und nicht die tatsächliche, liegen in steilen Kameraeinstellungen in flachen Gebieten sehr viel mehr Pixel nahe einer Höhenlinie, als in steilen. Ist die Kamera allerdings in einem sehr flachen Winkel, sieht man sehr viel mehr Pixel an den Berghängen, als von den flacheren Stellen.

4.4.2 Offenheit

Die Berechnung der Offenheit benutzt eine Hilfsfunktion, die den maximalen Zenitwinkel entlang einer Himmelsrichtung bestimmt.

```
1  OpenessInDirection(TexCoord, Direction, Steps) =
2    Max = 0
3    Origin = TexturSample(TexCoord)
4    P = 0
5
6    float l = length(Direction)*256
7
8    while Steps > 0
9      TexCoord += Direction
10     P += Scale1 / l
11
12     Current = TexturSample(TexCoord)
13     Angle = atan( (Current-Origin) / PI)
14
15     Max=max(Max, Angle)
16     Steps -= 1
17
18   return PI/2 - Max
```

Listing 4.4: Hilfsfunktion zur Berechnung der Offenheit

In Zeile 6 von Listing 4.4 wird die Länge des Richtungsvektors ermittelt und als Faktor für den Abstand P (siehe Abbildung 3.5) benutzt. Die Richtung ist also kein normalisierter Vektor, sondern gibt den Offset zum nächsten Pixel an, wodurch die Berechnung der neuen Texturkoordinate (Zeile 9) einfacher wird. Außerdem kann man über den *Scale* Parameter die Entfernung zweier Pixel bestimmen, was letztendlich die Helligkeit beeinflusst und gut geeignet ist, um die Ausgabe an verschiedene LOD-Stufen anzupassen (siehe 5.2).

In der Hauptfunktion wird *OpenessInDirection* dann für jede Richtung (aus Bild 4.1) aufgerufen (Listing 4.5).

Bei den letzten 4 Himmelsrichtungen (die nicht parallel zu einer Achse liegen), wird die Anzahl der Schritte mit $1/\sqrt{2}$ multipliziert, da unabhängig der Richtung Pixel für Pixel vorgegangen wird und die diagonalen Linien dementsprechend weniger Pixel haben. Am Ende wird das Ergebnis von dem Intervall $[0...2\pi]$ auf $[0...1]$ abgebildet, da jedes Verfahren einen Wert zwischen 0 und 1 ausgeben soll.

4.5 Approximation des Terrains durch eine quadratische Gleichung

Wie schon in Kapitel 3.1.1 gezeigt, werden die tatsächlichen Kernelwerte b erst am Ende der Rechnung benötigt. Man kann die Matrix $(A^T \cdot A)^{-1} \cdot A^T$ also vorberechnen und fest in den Code einbauen.

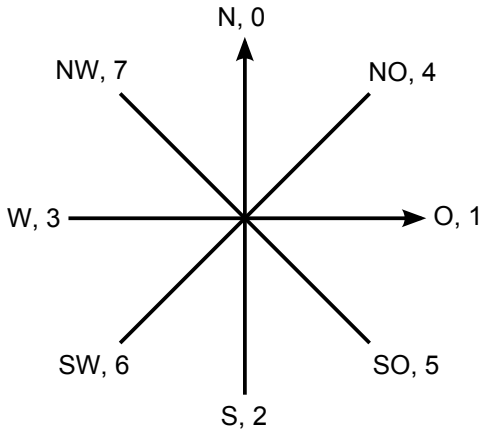

Bild 4.1: Die 8 Himmelsrichtungen

```
1  Openess() =
2    Openess = 0
3    PixelSize = 1 / 256
4
5    Openess +=
6      OpenessInDirection(TexCoord, (0, PixelSize), Steps)
7    ...(Direction 1-3)
8
9    Openess +=
10     OpenessInDirection(TexCoord, (PixelSize, PixelSize), Steps * 1/sqrt(2) )
11   ...(Direction 5-7)
12
13         return (Openess / 8) / (PI/2)
```

Listing 4.5: Berechnung der Offenheit

Damit reduziert sich das Problem auf eine Matrixmultiplikation und kann im Fragmentshader berechnet werden.

Dies wurde für die 3×3 Matrix in Gleichung 3.2 und analog für eine 5×5 Matrix gemacht. Die 5×5 Matrix hat bereits 125 Elemente, die nächstgrößere 7×7 Matrix hätte bereits $7^2 \cdot 6 = 294$ Elemente, weswegen keine weiteren Größen umgesetzt wurden. Ein allgemeines Verfahren für beliebige Kernelgrößen wäre zwar möglich, allerdings wäre dann die Codegenerierung für den Fragmentshader sehr viel komplexer (da unter anderem die obige Matrix berechnet werden müsste und man sehr viele indizierte Variablen benötigen würde) und wurde deshalb für diese Arbeit nicht umgesetzt.

Da zum einem GLSL keinen eingebauten Datentyp für diese Matrixgrößen bietet und zum anderen nicht immer alle Koeffizienten berechnet werden müssen, existiert für jeden Koeffizienten eine Funktion, die dessen Wert berechnet, sowie eine Initialisierungsfunktion, die die verwendeten Kernelvariablen mit Werten aus der Höhentextur füllt, wie Listing 4.6 zeigt.

```
1  float   bqe1, bqe2, bqe3,
2          bqe4, bqe5, bqe6,
3          bqe7, bqe8, bqe9;
4
5  float i6=1.0/6.0, i3=1.0/3.0, i4=1.0/4.0;
6
7  float bqeA()
8  {
9      return (i6*bqe1 + -i3*bqe2  + i6*bqe3   +
10             i6*bqe4 + -i3*bqe5  + i6*bqe6   +
11             i6*bqe7 + -i3*bqe8  + i6*bqe9)*bqeScale;
12 }
13
14 //... similar functions for B-F
15
16 void InitializeBQE(vec2 Coord, float TexSize, sampler2D Tex)
17 {
18     float PixelSize=1/TexSize;
19     bqe1=texture2D(Tex, Coord+vec2(-PixelSize, PixelSize)).x;
20     bqe2=texture2D(Tex, Coord+vec2(0.0,        PixelSize)).x;
21     bqe3=texture2D(Tex, Coord+vec2(+PixelSize, PixelSize)).x;
22
23     bqe4=texture2D(Tex, Coord+vec2(-PixelSize, 0.0)).x;
24     bqe5=texture2D(Tex, Coord+vec2(0.0,        0.0)).x;
25     bqe6=texture2D(Tex, Coord+vec2(+PixelSize, 0.0)).x;
26
27     bqe7=texture2D(Tex, Coord+vec2(-PixelSize, -PixelSize)).x;
28     bqe8=texture2D(Tex, Coord+vec2(0.0,        -PixelSize)).x;
29     bqe9=texture2D(Tex, Coord+vec2(+PixelSize, -PixelSize)).x;
30 }
```

Listing 4.6: Berechnung der Parabelkoeffizienten in GLSL

Kapitel 5

Ergebnisse

5.1 Die einzelnen Verfahren

In diesem Unterkapitel wird untersucht, welche Geländeeigenschaften man mit den unterschiedlichen Verfahren besonders gut feststellen kann und für welche Geländetypen sie sich eignen. Dazu wurde in zwei verschiedenen DEMs jeweils der selbe Geländeabschnitt mit den unterschiedlichen Verfahren visualisiert.

Bei dem ersten Gelände (Bild 5.1) handelt es sich um das *Puget Sound Terrain*, einem Beispieldatensatz, veröffentlicht von der University of Washington (`http://www.cc.gatech.edu/projects/large_models/ps.html`). Das Gelände weist neben einem größeren flachen Abschnitt ein zerklüftetes Gebirge mit hohen Gipfeln auf.

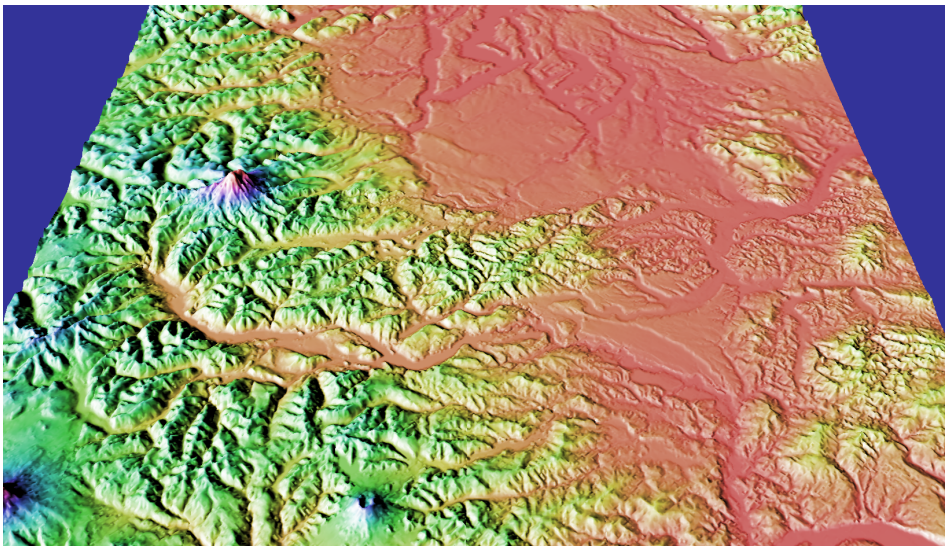

Bild 5.1: Puget Sound Terrain DEM

Das zweite Gelände (Bild 5.2) ist der Laubacher-Wald aus Hessen (Deutschland) und stammt von dem Landesamt für Denkmalpflege Hessen. Es ist wesentlich höher aufgelöst und beinhaltet Details

wie Straßen und Gebäudegrundrisse. Im Vergleich zu dem Puget Sound Terrain ist es wesentlich flacher und sanfter.

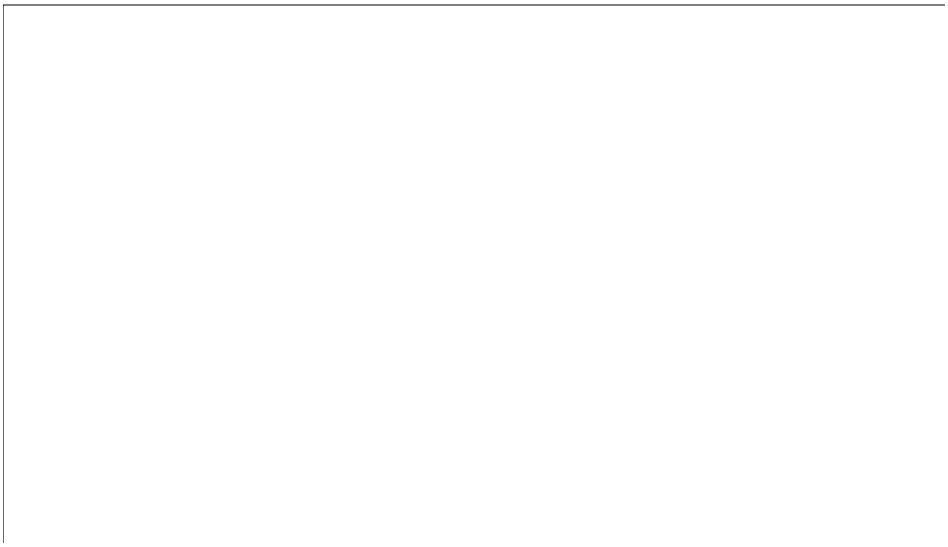

Bild 5.2: DEM von Hessen

Diese zwei sehr unterschiedlichen DEMs bieten eine gute Grundlage um, die implementierten Verfahren in verschiedenen Situationen miteinander vergleichen zu können. Dabei wurden die Parameter der Verfahren jeweils so angepasst, dass ein möglichst aussagekräftiges Gesamtbild entsteht. Im Folgenden sieht man jeweils auf der linken Seite das Hessen DEM und auf der rechten Seite das Pudget Sound DEM.

5.1.1 Höhengradient

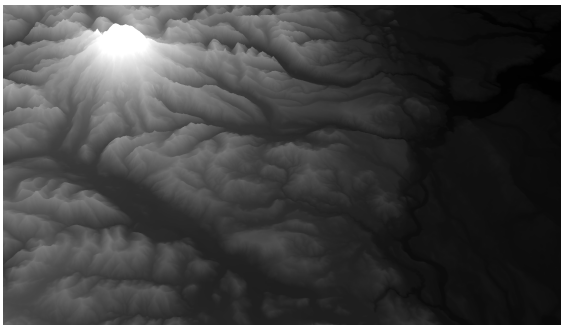

Bild 5.3: Höhengradient

In dem sehr gleichmäßigen Gebiet aus Hessen, ist der Höhengradient nur ein diffuser Farbverlauf, auf dem keinerlei Details erkennbar sind. Die Berge und Täler des Puget Sound Terrains sind dagegen deutlich zu erkennen, auch wenn feinere Details fehlen.

Der Höhengradient alleine ist also nur sehr bedingt zu gebrauchen. Eine sinnvolle Anwendung sieht man in Bild 5.1 und 5.2, wo der Höhengradient im HSV Modus den Farbton bestimmt und die Helligkeit durch direktionale Beleuchtung berechnet wird. Die eigentlichen Details entstammen dabei der direktionalen Beleuchtung, durch die unterschiedliche Farbtöne kann man aber die Höhen etwas besser einordnen. Der Höhengradient eignet sich damit also gut zur Unterstützung anderer Verfahren.

5.1.2 Direktionale Beleuchtung

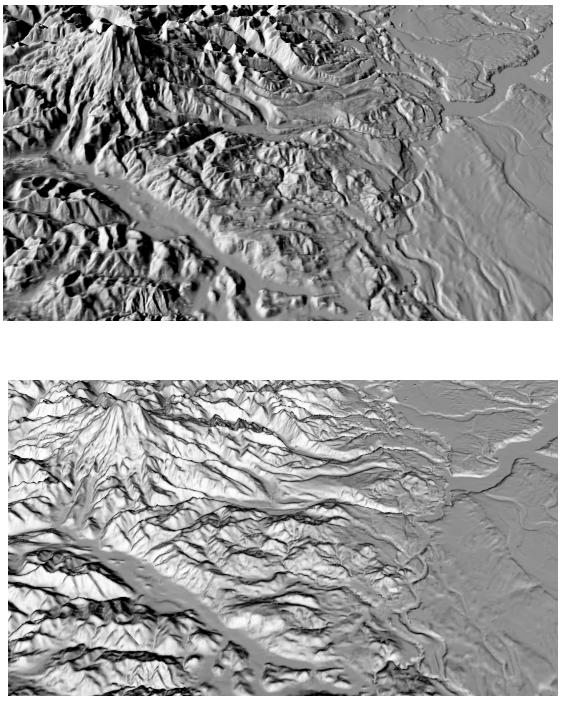

Bild 5.4: Direktionale Beleuchtung

Direktionale Beleuchtung erzeugt einen sehr guten intuitiven Eindruck von der Topologie des Terrains. Das Bild wirkt sehr plastisch und man erkennt nicht nur eine grobe Struktur sondern auch zahlreiche kleinere Details. Allerdings ändert sich der Gesamteindruck des Bildes dramatisch mit der Wahl der Lichtquelle, obwohl die Landschaft immer klar erkennbar bleibt. Manche Details sind allerdings bei ungünstigen Lichteinstellungen schwer zu erkennen und können so leicht übersehen werden.

Auch wenn das flachere DEM von Hessen durch die Beleuchtung recht strukturiert erscheint, fällt es schwer, die Höhe der Berge einzuschätzen. Durch die sehr unterschiedlichen Steigungen im Puget Sound DEM wirkt dieses detaillierter und die Form der Landschaft ist sehr viel leichter einzuschätzen.

Die direktionale Beleuchtung ist also ein einfaches Standardverfahren, das bis auf sehr flache oder gleichmäßige Terrains einen sehr guten Ersteindruck des Geländes ermöglicht. Daher wurde es auch für die Überblickbilder 5.1 und 5.2 zusammen mit dem Höhengradienten benutzt.

5.1.3 Steigung

Bild 5.5: Steigung

Im Vergleich zur direktionalen Beleuchtung ist die Steigung wesentlich weniger intuitiv, auch wenn man die Geländeform noch recht gut erkennen kann. Dafür kann man schon aus dem Farbwert alleine Ergebnisse erhalten. Beispielsweise erkennt man im Hessen DEM gut, dass sich die steilsten Stellen am rechten Hügel befinden. Unregelmäßigkeiten an den Hängen treten hier auch etwas deutlicher hervor, als bei der direktionalen Beleuchtung und man kann die künstlichen Strukturen am rechten Hügel besser erkennen.

Im Puget Sound DEM ist ein wesentlich höherer Kontrast zu erkennen, obwohl die Parameter für beide Bilder jeweils angepasst wurden. Flachland und Gebirge sind farblich klar voneinander getrennt. Außerdem kann man sehr deutlich erkennen, wie schroff die Gebirgsgrate sind, da sie von beiden Seiten eine sehr hohe Steigung haben, die am höchsten Punkt dann plötzlich auf 0 abfällt.

Die Steigung eignet sich also nur bedingt, um das Gelände als Ganzes oder kleinere, lokale Details darzustellen, hier sind andere Verfahren besser geeignet. Aber durch die einfache Analyse der Steigung kann man andere, interessante Schlüsse ziehen (z.B. wo das Gelände am steilsten ist).

5.1.4 Krümmung

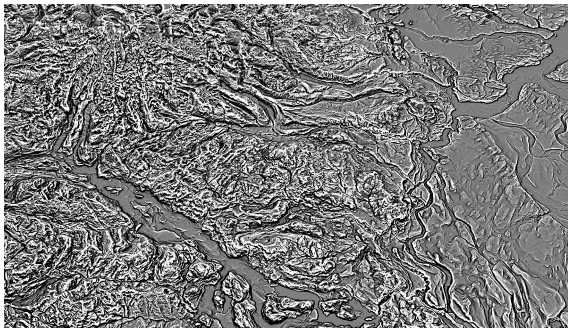

Bild 5.6: Krümmung

Die Krümmung erzeugt ein äußerst unruhiges Bild, in dem die Form des Terrains kaum mehr zu erkennen ist. Insgesamt gleicht das Bild dem Ergebnis eines Kantenerkennungsfilters, da sich entlang der Details deutliche Linien bilden.

Im Hessen DEM kann man anhand der einzelnen Linien die Form der Hügel erkennen. Dies wird zumindest im linken Teil des Puget Sound DEM nahezu unmöglich, hier erkennt man nur noch einige markante Täler. Das Flachland in der rechten Hälfte ist dagegen durch klare Linien beschrieben und wirkt wesentlich ruhiger.

Die Krümmung eignet sich gut dazu, markante Kanten im Gelände zu finden. Die tatsächliche Interpretation der Farbwerte ist dagegen sehr schwer, vor allen Dingen, da die meisten Stellen nahezu unabhängig von der Form des Geländes ähnliche Krümmungswerte habe, es sei denn es handelt sich um sehr flache Gebiete.

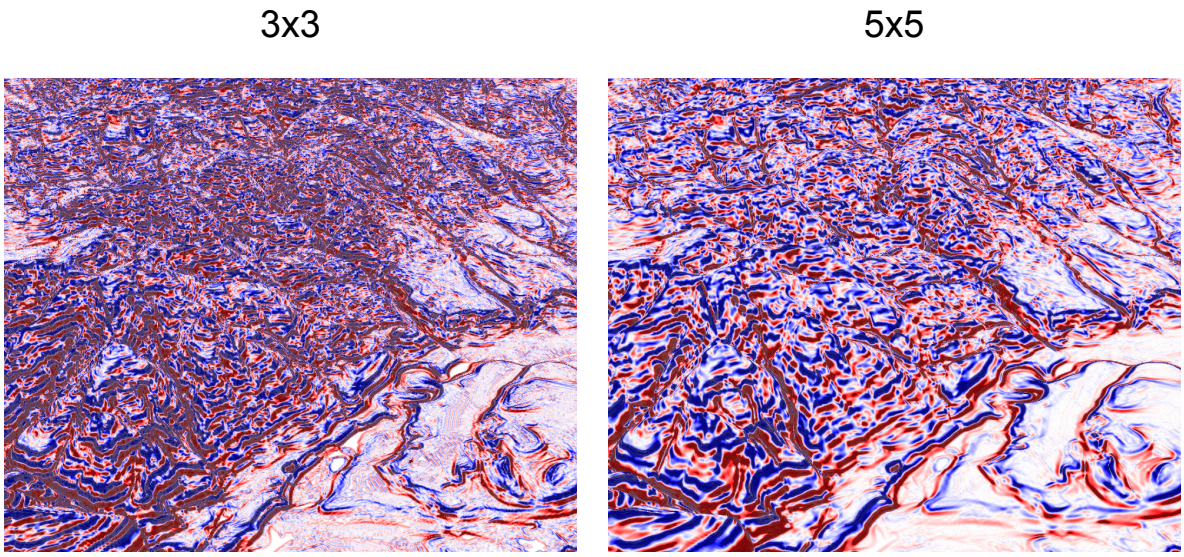

Bild 5.7: Krümmung mit Rot-Blau Gradient

Eine etwas intuitivere Darstellung des Terrains erhält man, wenn man einen 2 farbigen Gradienten zur Visualisierung benutzt. In Bild 5.7 wurde ein Rot-Weiß-Blau Gradient (rot im unteren Bereich, weiß in der Mitte, blau im oberen Bereich) benutzt, so dass konvexe Stellen blau erscheinen und konkave rot. Deutlich sieht man das beispielsweise am Übergang von dem Tal in der Mitte zu den Bergen (konkav, rot) oder an den Berggraten (konvex, blau), dort sind die Farben jeweils besonders intensiv.

Es zeigt sich, dass die Krümmung dazu neigt, extreme Werte anzunehmen. Eine mögliche Ursache könnten die recht kleinen Kernel $(3, 5)$ sein, die implementiert wurden, wobei die 5 schon um einiges ruhiger ist. Durch größere Kernels (die, wie in 4.5 begründet, nicht implementiert wurden) könnte so vermutlich ein deutlich besseres Bild erzeugt werden.

Der Benutzer muss allerdings schon anhand von anderen Bildern einen guten Eindruck davon

bekommen haben, wie das Gelände geformt ist, um die Bergformen identifizieren zu können. Dies ist gerade in einem Echtzeitsystem sehr leicht möglich, wodurch die Krümmung gut in Kombination mit anderen Verfahren genutzt werden kann.

5.1.5 Offenheit

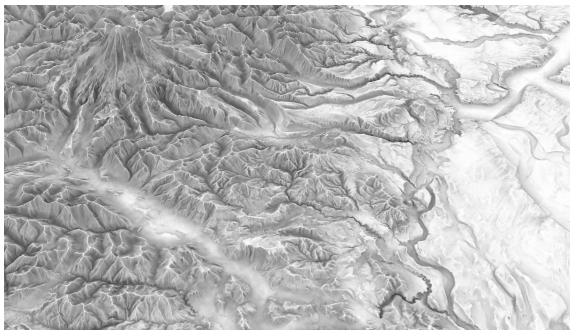

Bild 5.8: Offenheit

Die Offenheit bietet einen ähnlich guten intuitiven Eindruck des Geländes wie die direktionale Beleuchtung, zeigt aber gleichzeitig mehr Details und ist unabhängig von irgendwelchen Lichrichtungen. Erstaunlicherweise funktioniert sie nicht nur in den zerklüfteten Tälern des Pudget Sound DEMs sehr gut, sondern zeigt bei richtiger Kalibrierung der Parameter auch im generell sehr offenen Hessen DEM sehr gute Ergebnisse.

Bei unterschiedlichen Kernelgrößen (es sei noch einmal erwähnt, dass nicht der gesamte Kernel gefiltert wird, sondern nur 8 Linien entlang der 8 Richtungen in Bild 4.1) ändert sich die Darstellung der Offenheit, wie Bild 5.9 zeigt. Die kleineren Details sind auch bei geringen Kernelgrößen gut zu erkennen, je größer der Kernel allerdings wird, desto mehr werden enge Täler abgedunkelt. Insgesamt kann man diese so besser erkennen und es entsteht ein besserer räumlicher Eindruck, wobei sich die Darstellung kleinerer Details nur minimal verschlechtert. Dadurch werden die Ergebnisse mit größeren Kernels generell besser, als mit kleinen.

Durch die sehr großen Kernels tritt hier allerdings das in 5.3 beschriebene Problem der Tileränder besonders stark auf. Außerdem steigt natürlich die Rechenzeit, so dass die Offenheit das einzige Verfahren ist, bei dem die Framerate spürbar absinkt. Durch die interaktive Einstellung der Parameter stellt die Framerate allerdings kaum ein Problem dar, da man mit kleineren Kernels eine interessante Stelle suchen kann und diese dann mit größeren Kernels und einer etwas niedrigeren Framerate betrachten kann.

5.1.6 Rauheit

Die Ausgabe der Rauheit ähnelt auf den ersten Blick stark der Ausgabe der Steigung. Das Gelände ist gut erkennbar, wenn auch nicht so klar wie bei der direktionalen Beleuchtung oder der Offenheit.

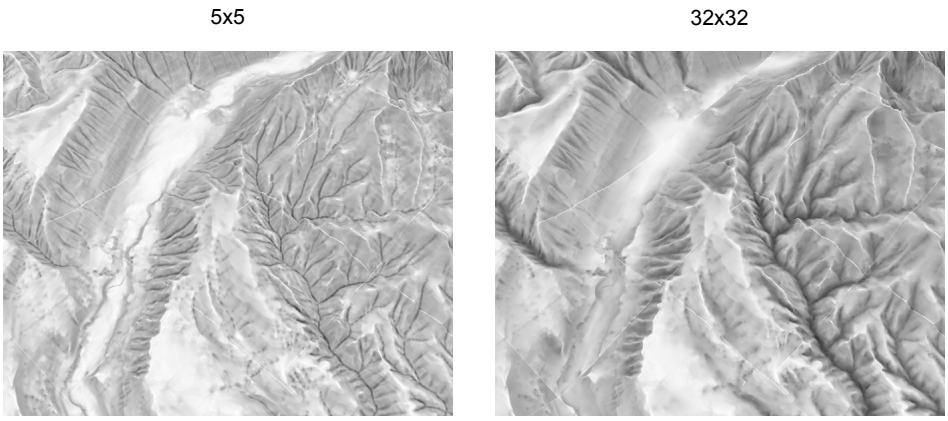

Bild 5.9: Auswirkung des Skalierungsparameters bei der Offenheit

Bild 5.10: Rauheit

Im direkten Vergleich zur Steigung sind leicht unterschiedliche Details erkennbar aber die grundsätzlichen Charakteristika sind jeweils die gleichen. Allerdings ist Rauheit ein sehr unterschiedlich definierter Begriff, so dass die anderen in 3.2.5 vorgestellten Verfahren ganz andere Ergebnisse zeigen können. Diese unterschiedlichen Rauheitsverfahren zu vergleichen, liegt allerdings außerhalb des Fokus dieser Arbeit.

Bei größeren Kernels erwartet man ein insgesamt helleres Bild (da ein großes Gebiet normalerweise mehr Höhenunterschiede beinhaltet, als ein kleines) und ein Auswaschen der feineren Details (da diese mit einer kleineren Gewichtung in das Endergebnis einfließen). Wie Bild 5.11 zeigt, treten beide Effekte wie erwartet ein. In der Praxis sind dadurch nur die kleinen Kernel interessant, da die großen Details zerstören, ohne neue zu schaffen. Unterschiedlich lokale oder globale Rauheitswerte erreicht man besser durch ein Zoomen des Bildausschnittes, da die gleiche Kernelgröße (in Pixeln) auf weniger detaillierten LOD-Stufen ein größeres Terrain abdeckt. Dadurch fällte es auch nicht mehr ins Gewicht, dass bei größeren Kernels die Framerate einbricht, da man diese in der Regel nicht benutzt.

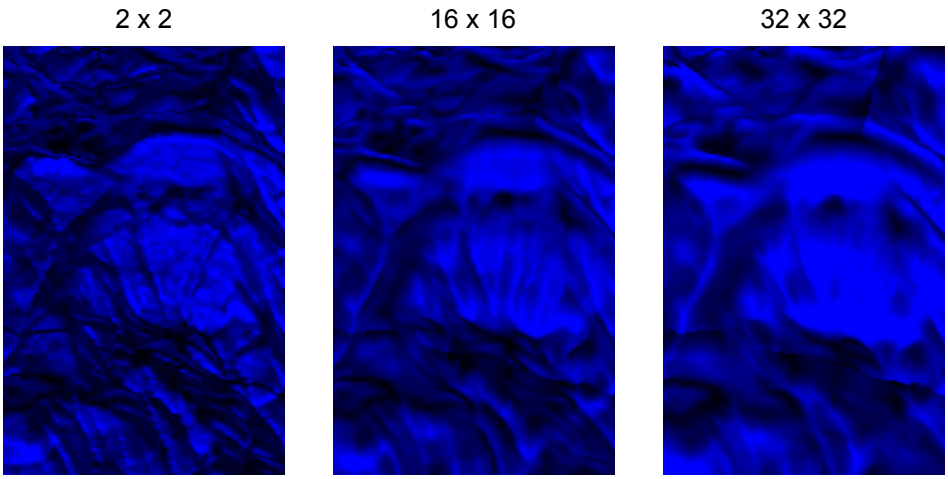

Bild 5.11: Unterschiedliche Kernelgrößen bei der Rauheit

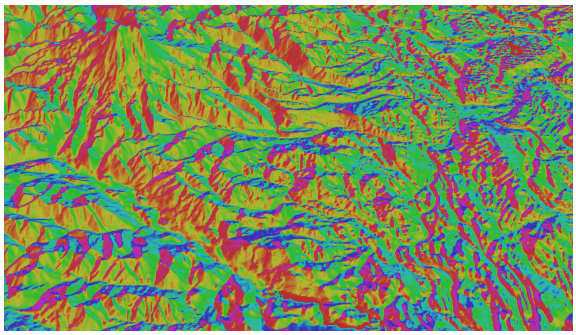

Bild 5.12: Richtung

5.1.7 Richtung

Zur Visualisierung der Richtung eignet sich der HSV-Modus am besten. Da die Ausrichtung des Geländes von $[0..360]$ Grad auf einen Ausgabewert von $[0..1]$ abgebildet wird, entsteht an der Grenze von 0 und 360 eine hässliche Kante. Wie Bild 3.10 zeigt, beginnt der Farbton im HSV-Farbraum bei einem rot Ton und endet dort auch wieder, wodurch dieser hässliche Übergang im HSV-Modus verschwindet, wenn man die Richtung im Farbtonkanal berechnet.

Im Gegensatz zur Richtung ergibt sich das Ergebnis der direktionalen Beleuchtung aus der Richtung und Steigung des Punktes, dafür werden entlang der Lichtrichtung gegenüberliegenden Geländeausrichtungen aber der selbe Wert zugeordnet. Im diesen Sinne zeigt die Richtung also etwas sehr ähnliches, aber insgesamt weniger, da man bei der direktionalen Beleuchtung aus dem Zusammenhang gleichfarbige Pixel auf gegenüberliegenden Berghängen als entgegengesetzt ausgerichtet interpretieren kann. Man kann auch bei dem Richtungsverfahren analog zur Lichtrichtung bei der direktionalen Beleuchtung einen Offset einstellen, allerdings bewirkt dieser nur eine Farbverschiebung und es werden keinerlei Details mehr oder weniger sichtbar.

Die Richtung erlaubt eine mäßige Interpretation der Geländekonturen. Während man im flachen Hessen DEM die Hügel noch gut erkennen kann und auch einige kleinere Details sieht, wirkt das Bild beim Puget Sound DEM besonders in der rechten Hälfte sehr unruhig. Es ist nicht mehr zu erkennen, dass dieses Gebiet sehr flach ist. In solchen Gebieten können schon minimale Geländeunterschiede zu komplett verschiedenen Farben führen. Der Gipfel in der linken Hälfte ist allerdings klar erkennbar. Bei solchen größeren Konturen kann man durch die Farbzuordnung das Gelände im Hinblick auf seine Ausrichtung gut interpretieren.

5.1.8 Höhenlinien

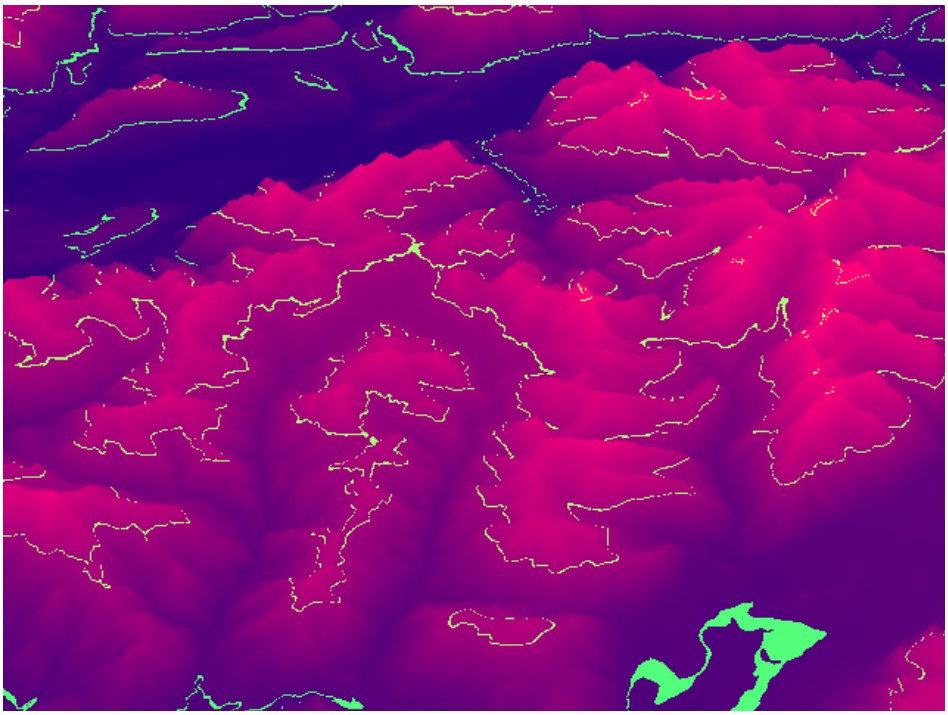

Bild 5.13: Höhenlinien

Zur besseren Verständlichkeit wurde in dem Bild 5.13 zusätzlich zu den Höhenlinien im Grünkanal noch ein Höhengradient im Rotkanal visualisiert, da ansonsten die Form der Berge sehr schlecht zu erkennen sind.

Im unteren Teil erkennt man deutlich das in 4.4.1 angesprochene Problem an sehr flachen Stellen, die Höhenlinie ist hier so dick, dass sie beinahe wie ein See wirkt. Auch ansonsten schwankt die Linienbreite, manche Stellen sind sehr dick und an anderen wird die Linie gar unterbrochen. Das Verfahren ist in dieser Implementierung zwar gut dazu zu gebrauchen, zu erkennen, welche Punkte auf einer Höhe liegen, erzeugt aber keine sonderlich ansprechenden Bilder und benötigt einiges an Feintuning seitens des Benutzers.

Von oben betrachtet zeichnen die Höhenlinien sehr schön die Form der Berge nach. Da sie nur

kleine Teile des Bilder betreffen, dort aber deutlich zu erkennen sind, lassen sie sich gut zur Unterstützung anderer Verfahren einsetzen, haben aber alleine betrachtet zu wenig Aussagekraft.

5.1.9 Zusammenfassung

Es hat sich gezeigt, dass sich überraschenderweise alle Verfahren für verschiedene Geländetypen so anpassen lassen, dass sie sinnvolle Ergebnisse liefern. Dabei liefern einige einen sehr guten Überblick über die Struktur und generelle Eigenschaften des Terrains, andere liefern eher spezielle Informationen oder sind nur in wenigen Fällen sinnvoll. Durch die Echtzeitberechnung und der daraus folgenden schnellen Umschaltbarkeit und Kalibrierbarkeit der Verfahren kann man diese sinnvoll kombinieren und so einen maximalen Informationsgewinn erzielen.

Durch die Parameter der Verfahren kann man diese nicht nur an den Geländetyp anpassen, sondern auch die Art der Ausgabe verändern. Während man bei der direktionalen Beleuchtung so grundsätzlich verschiedene Bilder erhält, gibt es bei der Offenheit oder Rauheit klar bevorzugte Werte. Für die Steigung und Krümmung wäre eine flexiblere Kerneleinstellung sicherlich wünschenswert und könnte interessante neue Ergebnisse liefern, allerdings war dies zu aufwändig zu implementieren (siehe 4.5).

5.2 Verschiedene LOD-Stufen

Bei der Berechnung der Rauheit oder Offenheit eines Geländes ist die Größe des verwendeten Kernels ein wichtiger Faktor. Da bei der Rauheit der Höhenunterschied des aktuellen Pixels mit den umgebenden Pixeln berechnet wird, wird dieser in der Regel größer ausfallen, wenn diese Pixel weiter entfernt sind.

Die Größe des Kernels meint dabei nicht nur dessen Auflösung, sondern auch die tatsächliche, räumliche Größe. Bei herkömmlichen Verfahren ist die Bildgröße und Auflösung konstant, bei dem Terrain-Explorer trifft dies aufgrund des LOD-Verfahrens aber nicht mehr zu. Jedes Tile hat zwar die gleiche Texturgröße, aber da die Tiles an sich unterschiedlich groß sind (je nach aktuellem Level), deckt ein Pixel in der Textur unterschiedlich viel Terrain ab. Trotz gleichbleibender Kernelgröße ändert sich die vom Kernel abgedeckte Fläche also in den verschiedenen Zoomstufen, wodurch Sprünge in der Rauheit des Geländes entstehen.

Beim Hereinzoomen in das Terrain wird die Rauheit bei den Sprüngen von einem Tile-Level in den nächsten immer weiter abnehmen (weswegen die Implementierung auch einen Skalierungsfaktor als Parameter hat, mit der man diesen Effekt für einzelne Bilder von Hand ausgleichen kann). Kippt man allerdings die Kamera, sieht man im allgemeinen mehrere Tilelevel auf einmal, wodurch der Vordergrund des Bildes dunkler wirkt als der Hintergrund. Bild 5.14 zeigt diesen Effekt, man erkennt im Vordergrund, der Bildmitte und dem Hintergrund 3 verschiedene Helligkeiten, die sprunghaft wechseln.

Zur Lösung dieses Problems könnte man die Kernelgröße dem aktuellen LOD-Level anpassen. Dann würden allerdings die einzelnen Abtastpunkte nicht mehr mit den Pixeln übereinstimmen; man

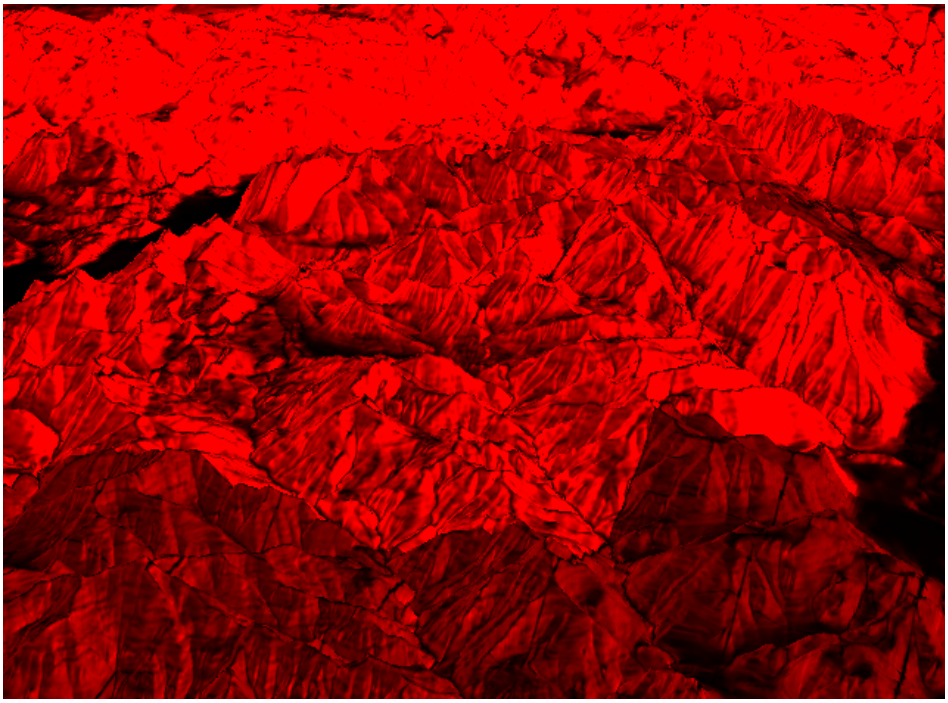

Bild 5.14: Rauhheitssprünge an unterschiedlichen Tileleveln

überspringt entweder einige Pixel, oder sampelt einige mehrfach, wie Bild 5.15 zeigt.

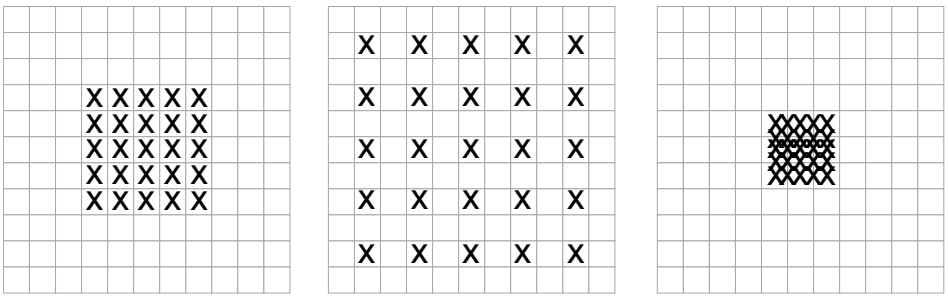

Bild 5.15: Sampeln der Textur mit verschiedenen Kernelskalierungen

Beim Überspringen mancher Pixel, wird das Ergebnis ungenauer (da man nur noch auf einer Stichprobe arbeitet), beim mehrfachen Sampeln erhöht man den Rechenaufwand, ohne mehr Details zu bekommen.

Es wäre theoretisch auch denkbar, immer auf der detailliertesten Textur zu arbeiten. Deren Größe war allerdings gerade der Grund, das LOD-Verfahren zu benutzen, daher könnte man eine solche Berechnung nicht in Echtzeit ausführen, sondern müsste sie vorberechnen, was allerdings gerade der Idee eines Echtzeitbetrachters widerspricht.

Letztendlich entsteht der vermeintliche Fehler dadurch, dass der Rauheitswert genau wie das Gelände in unterschiedlichen Detailstufen dargestellt wird, was ja auch sehr sinnvoll ist. Bei mehr

Details als Pixeln, würde die Darstellung flimmern (weswegen man bei statischen Texturen MipMaps verwendet), bei weniger verwaschen aussehen.

5.3 Tile Grenzen

Bei manchen Verfahren treten sichtbare Kannten zwischen den einzelnen Tiles auf, wie Bild 5.16, das die Terrainhöhe (Farbton) und Rauheit (Helligkeit) im HSV Modus visualisiert, zeigt.

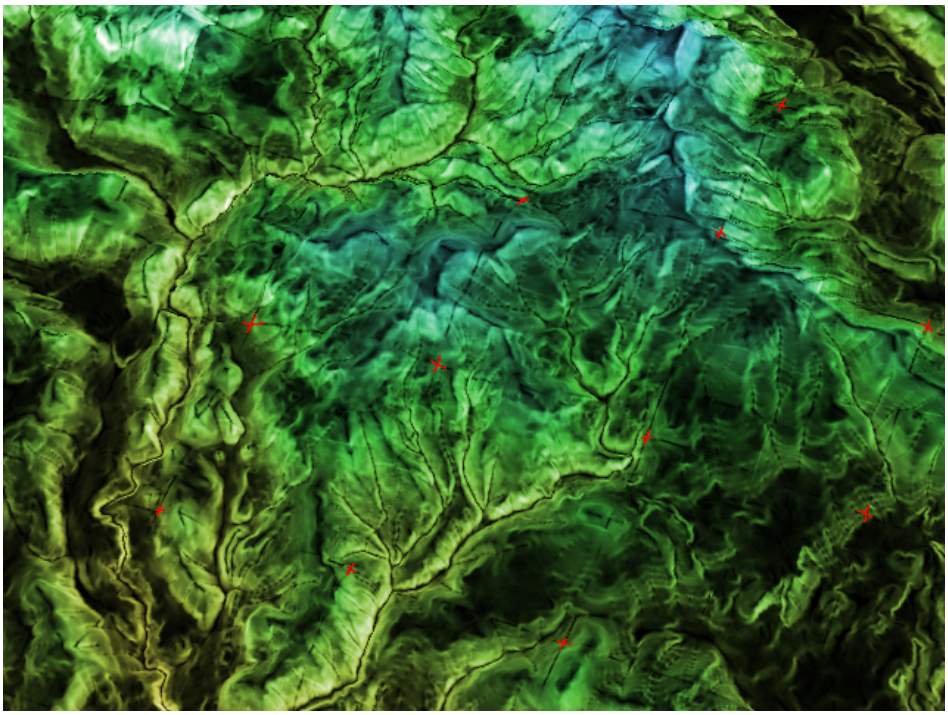

Bild 5.16: Sichtbare Tileübergänge

Diese Grenzen sind sehr dunkel, was auf eine geringe Rauigkeit hinweist. Diese entsteht dadurch, dass außerhalb der Textur gesammpelt wird, wo keine Informationen über das Terrain mehr vorliegen, so dass alle Pixel die gleiche Farbe haben. Zur Lösung dieses Problems müsste man entweder auf die Texturen der Nachbartiles zugreifen, was im Shader nur sehr schwer und ineffizient möglich wäre, oder aber die Texturen überlappen lassen, wie Bild 5.17 zeigt.

In diesem Bild ragen sowohl das Rechteck als auch die Spirale über die Texturgrenzen hinaus. Zur bloßen Darstellung ist das noch kein Problem, da die Nachbartiles die benötigten Informationen enthalten. Versucht man aber den Kernel für die Rauhigkeitsberechnung am Rand auszuwerten, fehlen diese Informationen dort. Wenn man nun aber die Textur etwas verkleinert und am Rand Teile von den angrenzenden Texturen einfügt, kann der gesamte Kernel korrekt ausgewertet werden. Allerdings müssen auch die Texturkoordinaten angepasst werden, damit sich der eigentliche Inhalt nach wie vor über das gesamte Tile erstreckt.

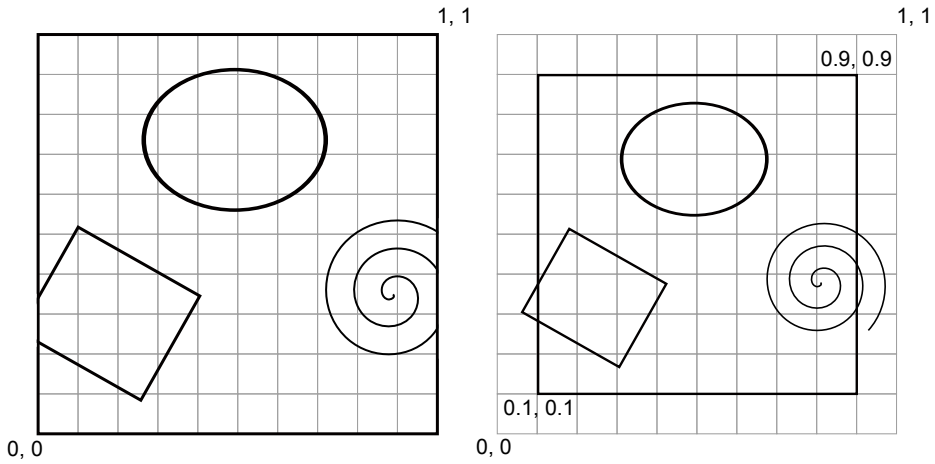

Bild 5.17: Anpassung der Textur, so dass Überlappungen mit dem nächsten Tile entstehen

Diese angepassten Texturen müssten allerdings schon beim Importieren des Terrains berechnet werden und ihre Überlappung müsste groß genug sein, um auch die maximale Kernelgröße sampeln zu können. Es handelt sich also um ein fehlendes Feature in Open Scene Graph, das im Terrain-Explorer nur mit beträchtlichen Aufwand implementiert werden könnte.

5.4 Gleichzeitige Visualisierung mehrerer Verfahren

Die Möglichkeit, mehrere Verfahren auf einmal in einem Bild anzuzeigen, soll es ermöglichen, bestimmte Verfahren und Parameterkonfigurationen sehr einfach zu vergleichen. In der Praxis zeigt sich allerdings, dass diese Bilder meist sehr unübersichtlich werden, so dass man letztendlich weniger erkennen kann, als wenn man sich die Bilder getrennt anschaut.

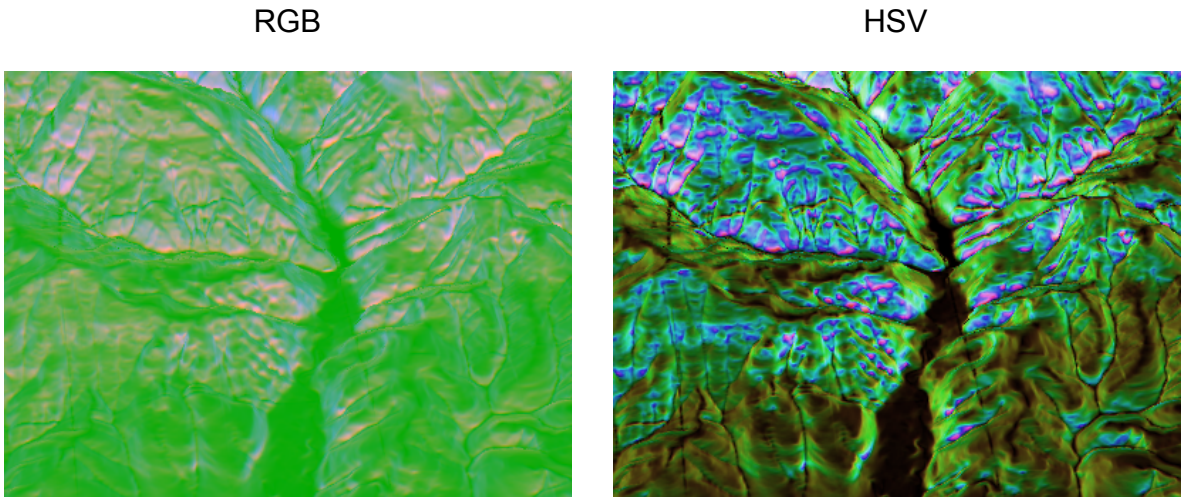

Bild 5.18: Vergleich zwischen RGB und HSV Kanalmischung

Die Kanaleinstellungen sind in den beiden Bildern (5.18) identisch, lediglich der Modus wurde umgeschaltet. Im ersten Kanal (rot, bzw. Farbton) wird die Steigung visualisiert, im dritten (blau, bzw. Helligkeit) die Rauigkeit. Das menschliche Auge ist relativ gut darin, einem Pixel eine Farbe und Helligkeit zuzuordnen, eine Farbmischung erkennt es aber nur sehr schwer. Mithilfe des Interpretationswidgets kann man allerdings einzelne Farbmischungen analysieren und so nach einiger Eingewöhnung Farben besser erkennen.

Ein Problem des HSV-Modus ist es, dass bei sehr kleinen Helligkeitswerten ein Pixel weiß oder schwarz ist, ganz gleich, welchen Wert der Farbton oder Sättigungskanal hat. Ebenso erscheinen ungesättigte Pixel immer grau, egal welchen Farbton sie haben. Diese Eigenschaften des HSV-Farbraums erschweren die Interpretation merklich, und auch das Interpretationswidget kann hier nicht helfen.

Insgesamt stellt sich heraus, dass die Möglichkeit, mehrere Verfahren gleichzeitig zu visualisieren nur bedingt sinnvoll ist. Spätestens bei 3 Verfahren wird es sehr schwierig, noch etwas Sinnvolles zu erkennen, so dass man sich ernsthaft überlegen muss, ob nicht der Vergleich mehrerer Bilder schnellere Ergebnisse liefert.

Kapitel 6

Zusammenfassung und Ausblick

Im Rahmen dieser Arbeit wurde mit dem Terrain-Explorer ein Programm entwickelt, das die interaktive Untersuchung eines DEMs ermöglicht.

Es hat sich gezeigt, dass alle Verfahren erfolgreich in Echtzeit implementiert werden können und trotz einiger Einschränkungen, wie etwa in der Größe beschränkte Kernel, sinnvolle Ergebnisse liefern. Durch die Verwendung der OSG-LOD Implementierung entstehen bei vielen Bildern sichtbare Fehler, dies ist aber kein prinzipielles Problem, sondern könnte durch mehr oder weniger große Anpassungen in OSG behoben werden.

Die Möglichkeit, mehrere Verfahren gleichzeitig anzuzeigen, ist in der Praxis weniger hilfreich als erhofft, da es schwer fällt, die dadurch entstehenden Bilder zu interpretieren. Hilfreich ist aber die Möglichkeit, schnell zwischen den einzelnen Verfahren hin und her schalten zu können, wodurch dieses Problem wieder ein Stück weit ausgeglichen wird.

Sinnvolle Erweiterungen dieser Arbeit wären die Behebung der LOD-Probleme, Verbesserungen in der Benutzeroberfläche des Terrain-Explorers wie zum Beispiel eine komfortablere Konfigurationsverwaltung, sowie die Implementierung weiterer Verfahren. Desweiteren könnte eine variable Kernelgröße für die Steigung und Krümmung neue Erkenntnisse über diese Verfahren liefern.

Literaturverzeichnis

[GSR11] C.H. Grohmann, M.J. Smith, and C. Riccomini. Multiscale analysis of topographic surface roughness in the midland valley, scotland. *Geoscience and Remote Sensing, IEEE Transactions on*, 49(4):1200 –1213, april 2011.

[JR] Bernhard Jenny and Stefan Räber. reliefshading.com. `http://www.reliefshading.com/index.html`.

[Ken09] Patrick J. Kennelly. Hill-shading techniques to enhance terrain maps. In *24th International Cartographic Conference, Santiago, Chile, 15-21 November 2009*, 2009.

[PRBY] Matthew Perry, Even Rouault, Howard Butler, and Chris Yesson. Gdal: gdaldem. `http://www.gdal.org/gdaldem.html`.

[SBDM+10] Cornelis Stal, Jean Bourgeois, Philippe De Maeyer, Guy De Mulder, Alain De Wulf, Rudi Goossens, Timothy Nuttens, and Birger Stichelbaut. Kemmelberg (Belgium) case study : comparison of DTM analysis methods for the detection of relicts from the first world war. In Rainer Reuter, editor, *Remote sensing for science, education, and natural and cultural heritage: proceedings of the EARSeL Symposium 2010*, pages 65–72. European Association of Remote Sensing Laboratories (EARSeL), 2010.

[SC05] Mike J. Smith and Chris D. Clark. Methods for the visualization of digital elevation models for landform mapping. *Earth Surface Processes and Landforms*, 30(7):885–900, 2005.

[WOB+07] Margaret F. J. Wilson, Brian O'Connell, Colin Brown, Janine C.Guinan, and Anthony J. Greham. Multiscale terrain analysis of multibeam bathymetry data for habitat mapping on the continental slope. *Marine Geology*, 30:3–35, 2007.

[YSP02] Ryuzo Yokoyama, Michio Shirasawa, and Richard J. Pike. Visualizing topography by openness: A new application of image processing to digital elevation models. *Photogrammetric Engineering and Remote Sensing*, 68:257–265, 2002.

[ZOK11] Klemen Zakšek, Kristof Oštir, and Žiga Kokalj. Sky-view factor as a relief visualization technique. *Remote Sensing*, 3(2):398–415, 2011.