Visual Analytics in der Fernerkundung: Entwicklung und Evaluation von Werkzeugen für die Untersuchung von zeitlichen Veränderungen in Satellitenbildern

Bachelorarbeit

für die Prüfung zum

Bachelor of Engineering

des Studienganges Informationstechnik

an der Dualen Hochschule Baden-Württemberg Mannheim

von
Tom Smukalski

11. September 2017

Bearbeitungszeitraum	19.06. - 11.09.2017
Matrikelnummer, Kurs	7492092, TINF14ITIN
Ausbildungsfirma	DLR, Braunschweig
Betreuer der Ausbildungsfirma	M.Sc. Simon Schneegans
Gutachter der Dualen Hochschule	Prof. Dr. Harald Kornmayer

Eidesstattliche Erklärung

gemäß § 5 (3) der „Studien- und Prüfungsordnung DHBW Technik“ vom 29. September 2015.

Ich habe die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

Braunschweig, den 8. September 2017

Zusammenfassung

Das Ziel der vorliegenden Bachelorarbeit war es, Softwarewerkzeuge zu entwickeln und zu evaluieren, um den Nutzer beim Erkennen von zeitlichen Veränderungen in Satellitendaten zu unterstützen. Dazu wurden zwei Prototypen entwickelt, einer zur Change Detection in Satellitenbildern und einer zur Change Detection in Höhenfeldern.

Der Prototyp zur Change Detection in Satellitenbildern wurde mit HTML und JavaScript entwickelt. Die Satellitenbilder erhält die Anwendung vom MapServer, einem auf Geodaten spezialisierten Webservice. Mit dem Prototypen konnte ein Sandsturm in den Bilddaten der HRSC-Kamera der Mars Express Mission gefunden werden.

Der Hauptteil der Arbeit beschäftigt sich mit der Change Detection in den Höhenfeldern von Satellitenbildern. Der dafür entwickelte Prototyp stellt die Höhenfelder in einer interaktiven 3D-Szene dar. Die Anwendung ist mit C++ und OpenGL entwickelt und erhält die Satellitendaten ebenfalls vom MapServer. Zur Unterstützung des Nutzers bei der Erkennung von zeitlichen Veränderungen, wurden verschiedene statische sowie eine dynamische Visualisierungsmethode entwickelt. Zur Evaluierung des Prototypen wurden bekannte Verdickungen des isländischen Dyngjujökull-Gletschers mit verschiedenen Visualisierungsmethoden nachgewiesen.

Abstract

The aim of the present bachelor thesis was to develop and evaluate software tools to support the user in the detection of temporal changes in satellite data. Two prototypes have been developed for this purpose. One for change detection in satellite images and the other for change detection in heightfields in satellite images.

The prototype for change detection in satellite images was developed with HTML and JavaScript. The application receives the satellite images from the MapServer which is a web service specialized in geodata. The prototype was able to detect changes in the image data. Specifically a sandstorm was found in the HRSC data acquired by the Mars Express Mission.

The main part of the bachelor thesis deals with the change detection in the elevation data of satellite images. The prototype developed for this purpose visualizes the heightfields in an interactive 3D scene. The application is developed with C++ and OpenGL and receives the satellite data from the MapServer. To support the user in the recognition of temporal changes, different static and dynamic visualization methods have been developed. For the evaluation of the prototype the well-known thickening of the Dyngjujökull glacier has been observed with different visualization methods.

Inhaltsverzeichnis

Abkürzungsverzeichnis **III**

Abbildungsverzeichnis **IV**

1 Einleitung **1**
1.1 Umfeld . . . 1
1.2 Motivation . . . 1
1.3 Ziel der Arbeit . . . 2
1.4 Gliederung der Arbeit . . . 2

2 Grundlagen **4**
2.1 Visual Analytics . . . 4
2.1.1 Grundlagen der optischen Wahrnehmung . . . 5
2.1.2 Change Detection in Satellitenbildern . . . 6
2.1.3 Visualisierung von topografischen Veränderungen . . . 9
2.2 Echtzeitvisualisierung von Terraindatensätzen . . . 10
2.2.1 Datensätze . . . 10
2.2.2 MapServer . . . 12
2.2.3 Terrain Renderer . . . 12
2.2.4 Pipeline zur Echtzeitvisualisierung von Terraindatensätzen . . . 13
2.3 Programmierwerkzeuge . . . 14
2.3.1 Programmierung von Webseiten . . . 14
2.3.2 C++ . . . 16
2.3.3 OpenGL . . . 16

3 Entwicklung eines Prototyps zur Change Detection in Satellitenbildern **18**
3.1 Anforderungen . . . 18
3.2 Konzept . . . 19
3.3 Implementierung . . . 21
3.4 Evaluierung der Ergebnisse . . . 23

4 Entwicklung eines Prototyps zur Change Detection in Höhendaten **25**
4.1 Anforderungen . . . 25
4.2 Konzept . . . 26

4.3 Implementierung 29
4.3.1 Entwicklung der Benutzeroberfläche 30
4.3.2 3D-Rendering der Höhendaten eines Satellitenbildes 31
4.3.3 3D-Rendering mehrerer Höhenfelder 33
4.3.4 Entwicklung statischer Visualisierungsmethoden 35
4.3.5 Entwicklung dynamischer Visualisierungsmethoden 41
4.4 Evaluierung der Ergebnisse 41

5 Ergebnisse und Ausblick **46**
5.1 Ergebnisse 46
5.2 Ausblick 46

Literatur **I**

Abkürzungsverzeichnis

DEM **D**igital **E**levation **M**odel

DHBW **D**uale **H**ochschule **B**aden-**W**ürttemberg

DLR **D**eutsches Zentrum für **L**uft- und **R**aumfahrt e.V.

ESA **E**uropean **S**pace **A**gency

GUI **G**raphical **U**ser **I**nterface

HRSC **H**igh **R**esolution **S**tereo **C**amera

VR **V**irtual **R**eality

WFS **W**eb **F**eature **S**ervice

WMS **W**eb **M**ap **S**ervice

Abbildungsverzeichnis

2.1 Visual Analytics Prozess 5
2.2 Landsat Satellitenbild 7
2.3 Thermalsatellitenbild 8
2.4 TerraSAR-X Aufnahme 9
2.5 Verschiedene Arten von Terrain Visualisierungen 10
2.6 Terrain Renderer 12
2.7 Healpix Gitternetz 13
2.8 Pipeline der Terraindatensätze 14
2.9 Programmierung von Webseiten 15
2.10 OpenGL Pipeline 17

3.1 Mockup des Prototyps zur Change Detection in Satellitenbildern 20
3.2 Sequenzdiagramm der Anwendung-MapServer-Kommunikation 21
3.3 Prototyp zur Change Detection in Satellitenbildern 23
3.4 Sandsturm in Arcadia Planitia 24

4.1 Mockup des Prototyps zur Change Detection in Höhendaten 28
4.2 Benutzeroberfläche des Prototyps 30
4.3 In 3D gerendertes Höhenfeld 33
4.4 In 3D gerenderte Höhenfelder 34
4.5 Gerendertes Gitternetz 36
4.6 Gerenderte Gitterlinien 36
4.7 Gerenderte Regenbogenskala 38
4.8 Histogramm und Differenzkarte 39
4.9 Differenzkarte mit Trends 40
4.10 Höhenveränderungen des Bildes im Fokus 41
4.11 Höhenunterschiede am Vatnajökull 42
4.12 Differenzkarte am Dyngjujökull 43
4.13 Höhenlinien am Dyngjujökull 44
4.14 Gletschererhöhungen am Dyngjujökull 45

1 Einleitung

1.1 Umfeld

Bei der vorliegenden Arbeit handelt es sich um eine Bachelorarbeit zur Erlangung des „Bachelor of Engineering“in der Studienrichtung Informationstechnik. Das Studium wurde an der Dualen Hochschule Baden-Württemberg (DHBW) in Mannheim in Kooperation mit dem Deutschen Zentrum für Luft- und Raumfahrt e.V. (DLR) absolviert. Die Praxisphasen wurden am DLR Standort Braunschweig in der Gruppe Interaktive Visualisierung der Einrichtung Simulations- und Softwaretechnik durchgeführt.

Das DLR ist das Forschungszentrum Deutschlands für die Luft- und Raumfahrt. An 20 Standorten wird in den Bereichen Luftfahrt, Raumfahrt, Energie, Verkehr und Sicherheit geforscht und entwickelt. Zudem ist das DLR für die Planung und Umsetzung der deutschen Raumfahrtaktivitäten verantwortlich[1].

Die Aufgaben der DLR-Einrichtung Simulations- und Softwaretechnik teilen sich in die Bereiche Forschung und Entwicklung auf dem Gebiet von Software-Engineering-Technologien sowie die Anwendung und Bereitstellung dieser Software auf. Themenschwerpunkte sind aktuell Softwareentwicklung für verteilte und mobile Systeme, Software für eingebettete Systeme, Visualisierung und High Performance Computing[2]. Der Arbeitsschwerpunkt der Gruppe Interaktive Visualisierung liegt in der Untersuchung und Integrierung von interaktiven Visualisierungsaspekten.

Die 2009 gegründete DHBW ist die erste und einzige staatliche duale Hochschule in Deutschland. An neun Standorten und in Kooperation mit über 9.000 Unternehmen bietet die DHBW Studiengänge in den Bereichen Wirtschaft, Technik und Sozialwesen an. Mit über 34.000 Studenten ist die DHBW die größte Hochschule Baden-Württembergs[3].

1.2 Motivation

Weltweit gibt es eine Vielzahl von Weltraummissionen im Bereich der Erdbeobachtung, Fernerkundung oder auch Planetenforschung. Diese Missionen produzieren riesige Datenmengen von Satellitenbildern. Vor allem für die Erde und den Mars gibt es viele Bilder, die über Jahre hinweg aufgenommen worden sind und sich teilweise oder komplett überlagern. Das Erkennen von Veränderungen in der Struktur oder Textur der Oberflächen kann zu wertvollen Ergebnissen führen. Die Struktur von Oberflächen ist das Relief, also die Form des Geländes. Die Textur ist das reine Bild der Oberfläche, hier können zum Beispiel die

Vegetation oder Wolken erkannt werden, da nur die Farbinformationen betrachtet werden. Mittels Vorher-Nachher Aufnahmen lassen sich die Ausmaße von Naturkatastrophen abschätzen, Landschaftsveränderungen feststellen oder auch mögliche Landeplätze auf anderen Planeten selektieren.

Die automatische Erkennung solcher Veränderungen ist jedoch äußerst kompliziert. Die einzelnen Bildaufnahmen lassen sich nur schwer vergleichen, da sie sich häufig sehr stark voneinander unterscheiden. Dies ist vor allem durch unterschiedliche Licht- und Schatten-Verhältnisse, verschiedene Aufnahmewinkel oder auch Wolken bedingt.

Um dieses Problem zu lösen kann auf Visual Analytics gesetzt werden. Visual Analytics ist ein Ansatz, der die Fähigkeit des Menschen schnell Muster und Trends visuell zu erfassen mit den Stärken der automatischen Datenanalyse kombiniert.

1.3 Ziel der Arbeit

Im Rahmen der Bachelorarbeit sollen Softwarewerkzeuge entwickelt und evaluiert werden, um Veränderungen in Satellitenbildern zu erkennen. Besonderer Wert soll dabei auf die Visualisierung von Veränderungen in Höhenfeldern gelegt werden. Die zentrale Frage der Bachelorarbeit lautet: „Mit welchen Methoden lassen sich Veränderungen in der Struktur (und Textur) von Oberflächen visualisieren?“

1.4 Gliederung der Arbeit

Die vorliegende Arbeit teilt sich grob in fünf Bereiche auf:

- **Grundlagen:** Hier werden Visual Analytics und die Echtzeitvisualisierung von Terraindaten näher erläutert. Darüber hinaus werden die benötigten und verwendeten Softwarewerkzeuge zur Entwicklung der Prototypen erklärt.
- **Entwicklung eines Prototyps zur Change Detection in Satellitenbilder:** In diesem Kapitel wird ein Prototyp zur Erkennung von Veränderungen in Satellitenbildern als Einstieg in die Change Detection entwickelt. Die Anwendung soll in der Lage sein, dem Nutzer beim Erkennen von zeitlichen Veränderungen in der Textur von Oberflächen zu unterstützen.
- **Entwicklung eines Prototyps zur Change Detection in Höhendaten:** Dies wird der Hauptteil der Bachelorarbeit. In diesem Kapitel wird der Prototyp zur Erkennung von zeitlichen Veränderungen in Höhendaten implementiert. Dafür werden verschiedene Visualisierungsmethoden entwickelt die den Nutzer beim Erkennen von Veränderungen unterstützen sollen. Der Nutzer soll mit der Oberfläche interaktiv in 3D interagieren können.

- **Ergebnisse und Ausblick:** Im abschließendem Kapitel werden die in der Bachelorarbeit erreichten Ergebnisse zusammengefasst. Es wird außerdem ein Ausblick gegeben welche Aufgaben im Rahmen der Bachelorarbeit nicht durchgeführt werden konnten und wie die Entwicklung der Prototypen in Zukunft fortschreiten soll.

2 Grundlagen

Im folgendem Kapitel werden die Grundlagen von Visual Analytics, Echtzeitvisualisierungen von Terraindaten und den verwendeten Softwarewerkzeugen näher betrachtet.

2.1 Visual Analytics

In der heutigen Zeit haben wir einen drastischen Anstieg von Datenmengen zu verzeichnen. Aufgrund der sich immer verbessernden Speichermöglichkeiten und der Vereinfachung des Erzeugens und Sammeln von Daten änderte sich der Umgang mit diesen. Während die Möglichkeiten zum Sammeln von Daten schnell ansteigen, wachsen die Möglichkeiten der Verarbeitung dieser Daten langsamer. Schon heute wird ein Großteil der Daten „roh“, also ohne Filterung und Verbesserung, gespeichert. Dieser Informationsüberfluss kann dazu führen, dass sich in ungeeigneten Daten verloren wird. Solche Daten sind zum Beispiel:

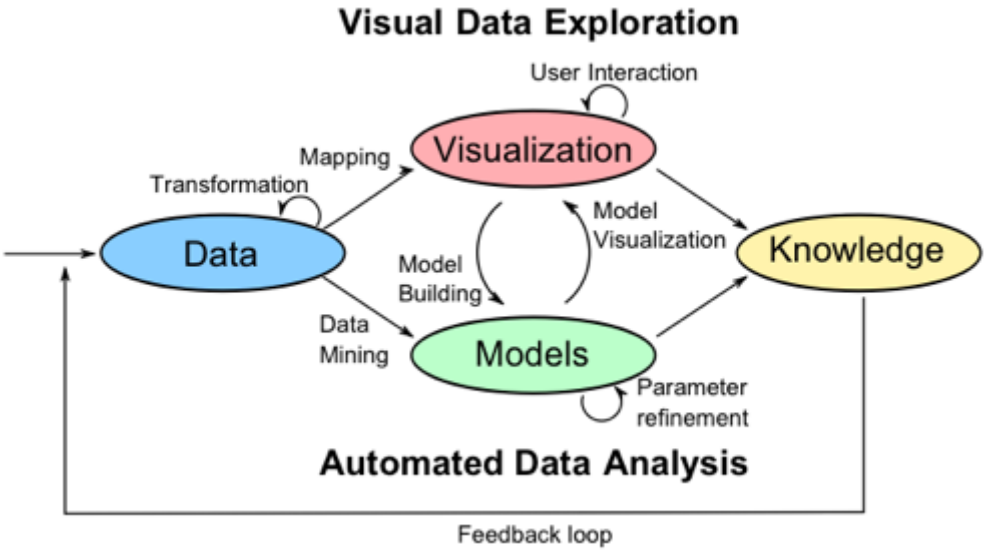
- irrelevant zur Erledigung der Aufgabe
- unsachgemäß verarbeitet
- unsachgemäß präsentiert[4].

An dieser Stelle setzt Visual Analytics mit dem Ziel an, aus großen und komplexen Datensätzen Erkenntnisse zu gewinnen. Dabei wird auf eine Kombination der Kreativität, Flexibilität sowie dem Hintergrundwissen von Menschen mit der enormen Speichermenge und Rechenleistung von Computern gesetzt. Mit visuellen Schnittstellen kann der Menschen mit der Analyse interagieren und ist so in der Lage, komplexe Probleme dem Computer sachkundig abzunehmen.

Der Prozess von Visual Analytics verbindet automatische und visuelle Analysemethoden in Verbindung mit menschlicher Interaktion. Abbildung 2.1 zeigt den Prozess mit den verschiedenen Abschnitten (durch Ovale dargestellt) und Übergängen (durch Pfeile dargestellt). Bevor die Daten verarbeitet werden können, müssen sie vorverarbeitet werden (*Data*). Anschließend entscheidet der Analyst, ob zuerst automatische (*Models*) oder visuelle (*Visualization*) Analysemethoden angewendet werden. Wenn zuerst die automatische Analyse angewendet wird, wird Data-Mining verwendet, um Modelle aus den Originaldaten zu gewinnen. Sobald ein Modell vorhanden ist, kann der Analyst dieses evaluieren und weiter verfeinern, indem er die Parameter anpasst oder andere

Algorithmen anwendet. Der ständige Wechsel der beiden Methoden ermöglicht die kontinuierliche Verfeinerung und Verifizierung der Ergebnisse. Wenn zuerst die visuelle Analyse angewendet wird, muss der Analyst seine Hypothese von einer automatischen Analyse bestätigen lassen[5].

Die Vorgehensweise orientiert sich dabei an dem Paradigma: „Analyse First – Show the Important – Zoom, Filter and Analyse Further – Details on Demand“ [1].



Quelle: `http://www.visual-analytics.eu/wp-content/uploads/visual-analytics-process12.png`

Abbildung 2.1: **Visual Analytics Prozess**

2.1.1 Grundlagen der optischen Wahrnehmung

„Der wichtigste Sinn des Menschen ist der Sehsinn, der Mensch ist ein 'Augentier'. [Es] gelangen 80% aller Informationen über die Augen ins Gehirn.“ [2] Change Detection ist eine Fähigkeit, die Veränderungen erkennen lässt. Dies ist eine Fähigkeit die für den Menschen überlebensnotwendig ist. Zum Beispiel im Straßenverkehr müssen Postions-, Richtungs- und Geschwindigkeitsveränderungen sofort wahrgenommen werden, um schnellstmöglich auf einzelne Ereignisse reagieren zu können. Dadurch, dass dies auch im Alltag eine große Rolle spielt, ist der Mensch, im Gegensatz zu Computern, sehr gut im Erkennen von Veränderungen[6].

[1] D. A. Keim, F. Mansmann, J. Schneidewind, J. Thomas, H. Ziegler: Visual analytics: Scope and challenges. Visual Data Mining, 2008, S. 82.

[2] M. Dahm: Grundlagen der Mensch-Computer-Interaktion, 2005, S. 41.

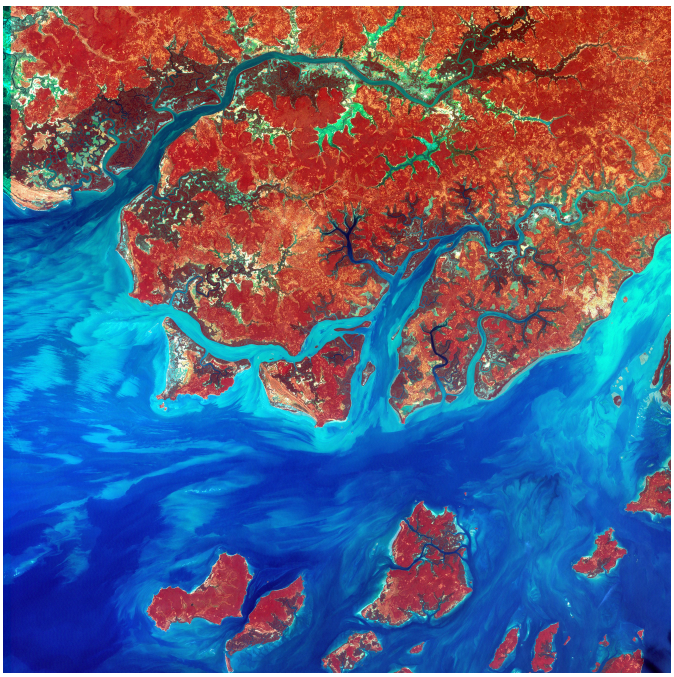
Dafür gibt es mehrere Gründe:

- **Hintergrundwissen:** Wenn ein Mensch zwei Bilder miteinander vergleicht, verwendet er sein Hintergrundwissen, um nach erwarteten Veränderungen zu suchen. Ungewohnte oder unerwartete Objekte fallen einem Menschen meist direkt auf.
- **Flexibilität:** Unterschiedliche Beleuchtungsstärken, Aufnahmewinkel, Schatten oder Störungen, wie Wolken, schränken einen Menschen in der Veränderungserkennung nicht ein (solange sie keine relevanten Bereiche verdecken). Für einen Computer ist es schwierig zwischen realen Veränderungen der Oberfläche und Störungen zu unterscheiden, da Vergleiche meist nur pixelweise unternommen werden.
- **Übung:** Wie schon eingangs erwähnt, spielt die Veränderungserkennung im Alltag für den Menschen eine große Rolle. Er ist von Geburt an darauf trainiert Veränderungen zu erkennen und zu deuten.

2.1.2 Change Detection in Satellitenbildern

Satellitenbilder sind Bilder von der Erde oder anderen Planeten, die mit Hilfe von Fernerkundungsverfahren von Satelliten erstellt werden. Sie können mittels verschiedener Sensoren erstellt werden. Beispiele hierfür sind Multispektralkameras, Thermalbildkameras oder Radarsysteme. Deren unterschiedliche Daten können anschließend für verschiedene Forschungszwecke genutzt werden.

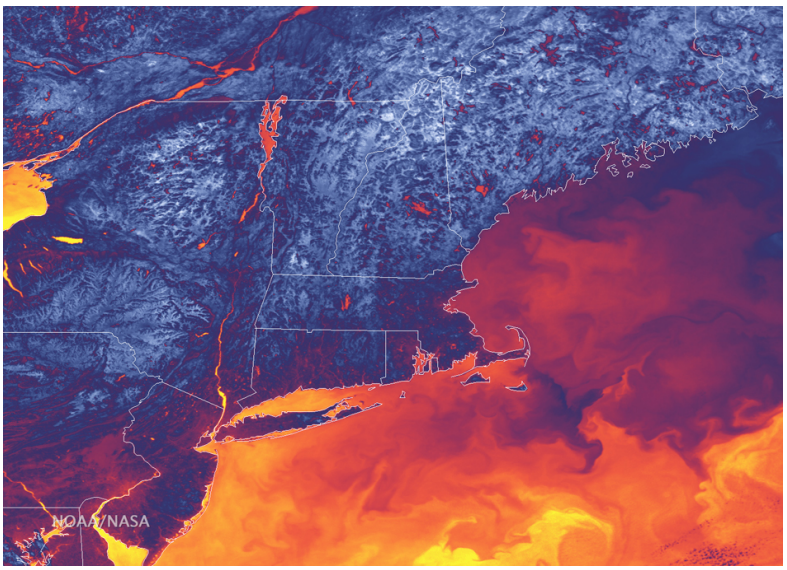
Mittels Multispektralkameras werden multispektrale Bilder aufgenommen, das sind Bilder welche Informationen über reflektierende oder emittierende elektromagnetische Strahlung unterschiedlicher Wellenlängen enthalten. Meist werden Sensoren genutzt, die blaues, rotes und infrarotes Licht aufnehmen. Bei den Fernerkundungsmissionen Landsat [7] werden Multispektralbilder aufgenommen, die für verschiedene wissenschaftliche Zwecke verwendet werden (siehe Abbildung 2.2). Beispiele für solche Forschungen sind Veränderungen in der Oberfläche, wie Entwaldung [8], Städtewachstum [9] oder Gletscherschmelzen [10]. Diese können aufgrund der hohen Genauigkeit der Bilder erkannt werden.



Quelle: `https://www.nasa.gov/sites/default/files/images/330946main_landsat_art_guinea_lrg_full.jpg`

Abbildung 2.2: **Landsat Satellitenbild:**
Dieses Multispektralbild wurde von der Kamera des Landsat-7 Satelliten in Guinea-Bissau aufgenommen. Das Bild wurde mittels Sensoren für die Farben rot, blau und infrarot am 12. Januar 2000 erstellt.

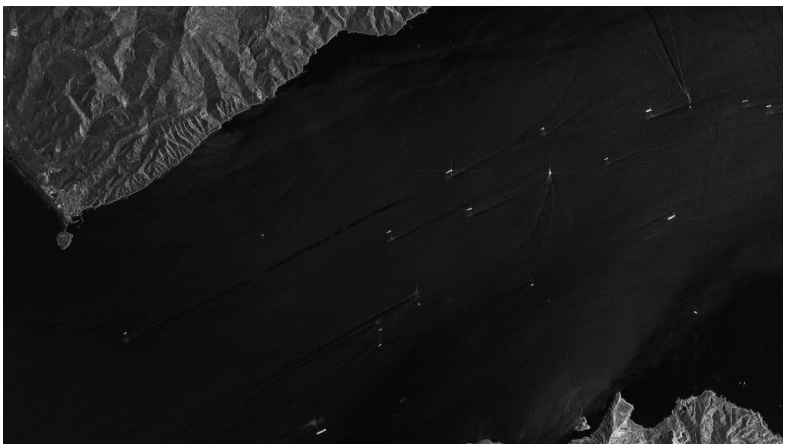
Thermalbildkameras ähneln herkömmlichen Kameras, empfangen jedoch Infrarotstrahlung. Die Bilder werden in vielen verschiedenen Bereichen verwendet, so finden sich Anwendungen zum Beispiel in der Medizin (für diagnostische Zwecke), bei der Feuerwehr (Aufspüren von Brandherden) oder im Militär (Beobachtung/Aufklärung bei schlechter Sicht oder Dunkelheit). Thermale Satellitenbilder können unter anderem zur Untersuchung von Erdbeben verwendet werden[11]. In dieser Studie wurden Thermalbilder vor, während und nach Erdbeben in China und Japan zwischen 1997 und 1999 miteinander verglichen, um die Erdbeben zu untersuchen. Hierbei wurde festgestellt, dass die Temperatur bereits 6-24 Tage (in China) beziehungsweise 7-10 Tage (in Japan) Anomalien aufwies. Die Bilder wurden von einem Satelliten der National Oceanic and Atmospheric Administration (NOAA) erstellt. Ein solches Satellitenbild ist in Abbildung 2.3 zu sehen.



Quelle: `http://spaceref.com/earth/thermal-satellite-imagery-shows-variations-across-northeastern-united-states.html`

Abbildung 2.3: **Thermalsatellitenbild:**
Dieses Satellitenbild wurde vom NOAA/NASE Soumi NPP Satelliten aufgenommen. Es zeigt die Oberflächentemperaturen an der US-amerikanischen Ostküste. Die blauen und weißen sind die kältesten, die gelben die wärmsten Gebiete. Die Aufnahme entstand am 13.9.2016.

Bei Radarsystemen wird die zu beobachtende Oberfläche mittels elektromagnetischer Wellen abgetastet. Radaraufnahmen besitzen gegenüber Kameras, die mit sichtbarem Licht arbeiten, mehrere Vorteile. Die Aufnahmen sind unter anderem leicht interpretierbar. Außerdem können die Aufnahmen unabhängig von der Beleuchtung erstellt werden, ebenso spielt die Witterung keine Rolle, da die Mikrowellen nicht von Wolken oder ähnlichen Störfaktoren beeinflusst werden. TerraSAR-X ist ein deutscher Erdbeobachtungssatellit, der in einer Kooperation zwischen dem *DLR* und *Airbus Defence and Space* entstanden ist. Die Aufnahmen weisen eine sehr hohe Auflösung (1 m bei einer Szenegröße von 10 km x 5 km) auf, ein von TerraSAR-X aufgenommenes Bild ist in Abbildung 2.4 zu sehen[12]. Aufgrund der hohen Auflösung eignen sich die Bilder für verschiedene Anwendungsfälle, so können unter anderem Schiffe überwacht werden[13]. Die Bilder eignen sich auch für Change Detection Ansätze, so können die Ausmaße von Überflutungen in urbanen Gegenden abgeschätzt werden[14].



Quelle: `http://www.dlr.de/dlr/desktopdefault.aspx/tabid-10387/#gallery/333`

Abbildung 2.4: **TerraSAR-X Aufnahme:**
Dieses Satellitenbild wurde vom TerraSAR-X Satelliten aufgenommen. Auf dem Bild ist die Straße von Gibraltar abgebildet. Die weißen Stellen repräsentieren Schiffe. Das Bild wurde am 9.7.2007 mit einer Auflösung von 3 m aufgenommen.

2.1.3 Visualisierung von topografischen Veränderungen

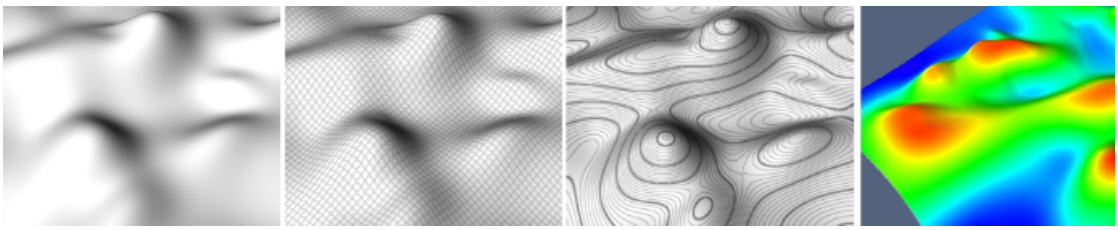
Möglichkeiten der Visualisierung von topografischen Veränderungen zu entwickeln und zu evaluieren soll der Schwerpunkt dieser Arbeit sein. Für die Visualisierung von Geländeveränderungen gibt es verschiedene Möglichkeiten. Das Verwenden von prozeduralen Texturen vereinfacht das Erkennen ungemein. Diese Texturen können unterschiedliche Formen annehmen.

Drei Beispiele sind im folgenden aufgelistet und in Abbildung 2.5, im Vergleich zum Terrain ohne darüber liegende Texturen (l.), abgebildet:

- **Gitternetz:** Über das Terrain wird eine Gitterstruktur gelegt, die Gitter passen sich der Geländeneigung in Relation zum Betrachter an. In Abbildung 2.5 ist das Gitternetz das zweite Bild von links.

- **Höhenlinien:** Höhenlinien oder auch Niveaulinien symbolisieren Punkte auf gleicher Höhe in einem Gelände. Meist werden verschiedene Linienarten benutzt, es wird zum Beispiel jede fünfte oder zehnte Linie hervorgehoben, um die Höhe besser ermitteln zu können. In Abbildung 2.5 ist das Gitternetz das zweite Bild von rechts.

- **Farbskala:** Bei der Verwendung einer Farbskala werden die einzelnen Punkte auf einer Höhe in einer Farbe eingezeichnet. Dadurch ergibt sich ein durchgängiger Farbverlauf, der entweder die verschiedenen Helligkeitsstufen einer Farbe darstellen oder auch den Regenbogenverlauf annehmen kann. In Abbildung 2.5 wird die Farbskala im rechten Bild verwendet.



Quelle: Effectiveness of Structured Textures on Dynamically Changing Terrain-like Surfaces [15]

Abbildung 2.5: **Verschiedene Arten von Terrain Visualisierungen:**
In dieser Abbildung sind verschiedene Visualisierungsmöglichkeiten von Terrain dargestellt. Von links nach rechts: Standardterrain ohne Visualisierung, Gitternetz, Höhenlinien und Farbskala.

Verschiedene wissenschaftliche Studien haben sich schon mit dem Thema der Visualisierung von topographischen Veränderungen beschäftigt, um herauszufinden, welche Texturunterstützung Veränderungen am besten erkennen lässt[15] [16] [17]. Gitternetze und Höhenlinien gelten als die effektivsten Visualisierungsmöglichkeiten.

Sweet und Ware [16] fanden 2004 heraus, dass Gitternetze effektiver sind als Höhenlinien. Weiterhin stellten sie fest, dass nicht nur die Textur die Erkennung von Veränderungen beeinflusst, sondern auch der Blickwinkel des Betrachters. In manchen Blickwinkeln eignen sich Höhenlinien besser als Gitternetze. In der Studie wurden jedoch nur die statische Terrainänderung betrachtet. Das bedeutet es wurden nur zwei einzelne Bilder verglichen auf denen Terrainänderungen vorkamen, jedoch nicht der Verlauf zwischen dieser Änderung.

Butkiewicz und Stevens untersuchten 2016 die dynamische Veränderung von Gelände[15]. Bei der dynamischen Veränderung muss neben der Unterstützung sichergestellt werden, dass kleine Änderungen aufgrund der sich ständig wechselnden Szene nicht übersehen werden. In der Studie wurde festgestellt, dass Höhenlinien für dynamische Szenen am effektivsten sind.

2.2 Echtzeitvisualisierung von Terraindatensätzen

2.2.1 Datensätze

Die beiden Prototypen sollen die Change Detection von Satellitendaten in verschiedenen Datensätzen implementieren. Zur Evaluierung und zu Testzwecken der Anwendung wurden die Datensätze von ArcticDEM[18] und die der HRSC[19] ausgewählt, da diese jeweils eine hohe Auflösung besitzen und zudem frei verfügbar sind. Der ArticDEM Datensatz wurde bei der Entwicklung des Prototyps zur Change Detection in Höhendaten verwendet, der HRSC Datensatz bei der Entwicklung des Prototyps zur Change Detection in Satellitenbildern.

ArcticDEM

ArcticDEM ist ein Projekt, dass es sich zur Aufgabe gemacht hat ein hochauflösendes und hochqualitatives digitales Oberflächenmodell der Arktis aufzunehmen. Dieses Modell wird mittels Stereo-Satellitenbildern erstellt. Das Projekt ist eine Initiative der National Science Foundation (NSF) und der National Geospatial-Intelligence Agency (NGA). Die NSF ist eine US-amerikanische Behörde deren Aufgabe es ist, Forschung und Bildung auf allen Feldern der Wissenschaft finanziell zu unterstützen[20]. Sie ist vergleichbar mit der Deutschen Forschungsgemeinschaft. Die NGA ist die US-Behörde für militärische, geheimdienstliche und kommerzielle kartografische Auswertungen und Aufklärung[21].

Das Projekt wurde mit dem Zweck geschaffen hochauflösende Terraindaten entfernter Regionen aufzunehmen und die Möglichkeiten der Technologie aufzuweisen um solche große Datenmengen zu verarbeiten. Weiterhin liefert es eine Lösung für die Notwendigkeit topografische Veränderungen genau messen zu können. Nach Ablauf des Projektes soll jede Landfläche nördlich des 60. Breitengrades sowie das komplette Territorium von Grönland, Alaska sowie die Kamtschatka-Halbinsel erfasst werden[18].

Die Bilder haben eine Auflösung von rund 0,5 m. Zur Erstellung der Daten wurde das panchromatische Band der WorldView-1, WorldView-2 und WorldView-3 Satelliten verwendet. Panchromatische Aufnahmen weisen eine Empfindlichkeit im Bereich des sichtbaren Lichtes auf, jedoch ohne Differenzierung einzelner Spektralbereiche. Die World-View Satelliten sind kommerzielle Erdbeobachtungssatelliten der US-Firma DigitalGlobe[22].

High Resolution Stereo Camera

Die High Resolution Stereo Camera (HRSC) ist Deutschlands wichtigster Beitrag zu Mars Express. Mars Express ist eine Weltraummission der Europäischen Weltraumorganisation (ESA). Die Sonde wurde am 2. Juni 2003 gestartet und ist die erste europäische Mission zum Mars. Ziel ist es auf dem Mars nach Spuren von Wasser und Anzeichen von Leben zu suchen[23].

Die HRSC wurde vom DLR entwickelt und soll die Oberfläche des Mars hochauflösend in 3D und Farbe aufnehmen. Mit den Bildern sollen Fragen zur geologischen und klimatischen Geschichte des Planeten geklärt werden. Zusätzlich besitzt die HRSC noch einen *Super Resolution Channel* (SRC), mit diesem können Aufnahmen mit einer Auflösung von bis zu 2-3 m pro Pixel erstellt werden. Das Mars Express Raumschiffe fliegt in höher Höhe von mindestens 270 km über den Mars und nimmt dabei Bildstreifen in einer Breite von 52 km und einer Länge von mindestens 300 km auf. Die Länge ist dabei von der Datenspeicher- und Übertragungskapazität abhängig. Der SRC erstellt 2,3 km x 2,3 km große Bilder in der Mitte der Bildstreifen. Mit den Streifen soll so nach und nach die komplette Oberfläche des Mars aufgenommen werden[19].

2.2.2 MapServer

Der MapServer [24] ist ein Open Source Mapserver-Projekt der Open Source Geospatial Foundation (OSGeo). Mapserver sind Server, die auf Geodaten spezialisierte Webservices anbieten. Sie werden auch Geodienste genannt und dienen zur Verarbeitung von Kartenausschnitten und ortsbezogenen Informationen. Der MapServer ist konform zu den OGC-Standards und implementiert Web Coverage Service (WCS), Web Feature Service (WFS), Web Map Service (WMS) und Sensor Observation Service (SOS). Die OGC-Standards werden vom Open Geospatial Consortium (OGC) entwickelt mit dem Ziel, im Bereich der raumbezogenen Informationsverarbeitung allgemeingültige Standards festzulegen, um Interoperabilität zu erreichen.

Der MapServer wurde 1994 von der University of Minnesota, in Zusammenarbeit mit der NASA, veröffentlicht, mit dem Ziel die von der NASA erstellten Satellitenbilder der Öffentlichkeit zugänglich zu machen. Die Software ist plattformübergreifend und in der Programmiersprache C geschrieben, mittels der *Mapscript*-Schnittstelle jedoch auch in anderen Sprachen wie Java, Python oder PHP verfügbar.

2.2.3 Terrain Renderer

Der Terrain-Renderer ist ein Datenmanagement- und Computergrafikwerkzeug, das im DLR in der Einrichtung Simulations- und Softwaretechnik zum Einsatz kommt. Er ermöglicht es riesige Höhenfelddatensätze von Planeten interaktiv zu erkunden. Dies wird unter anderem für geologische Studien benutzt. Aktuell befindet sich der Terrain-Renderer in der zweiten Version (Next Generation Terrain-Renderer). In Abbildung 2.6 ist die Erde im Terrain-Renderer dargestellt.

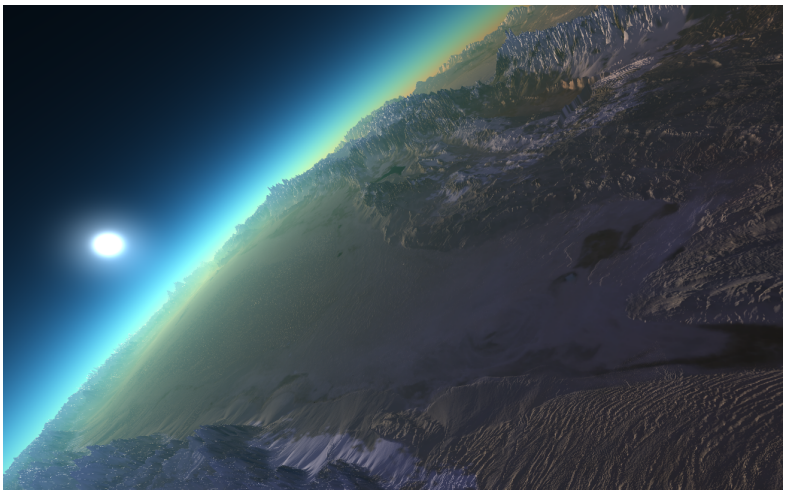
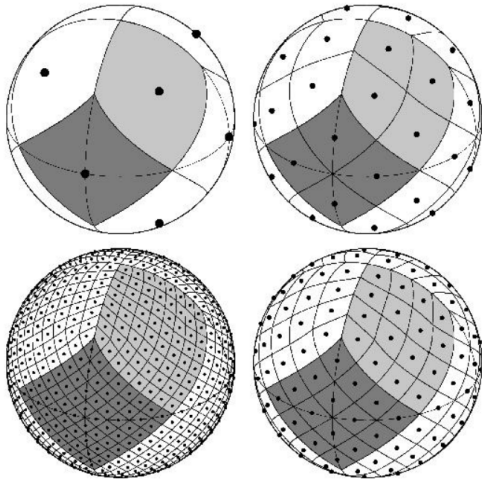


Abbildung 2.6: **Terrain-Renderer**

Das Ziel ist es, das Gelände von Planeten hochaufgelöst und interaktiv zu rendern. Die Planeten bestehen dabei aus verschiedenen Satellitenbildern in unterschiedlichen Auflösungen, meist sind es mehrere tausend Bilder. Diese riesigen Datenmengen, meist mehrere hundert Gigabytes, können nicht gleichzeitig auf dem Computer dargestellt werden. Aus diesem Grund wird eine sogenannte *Level of Detail (LOD)*-Struktur verwendet. Sie passt die Auflösung der einzelnen Bilder der aktuell für den Nutzer sichtbaren Szene an, ist er nah am Planeten, erhöht sich die Auflösung, ist er weiter entfernt, verringert sie sich wieder. Dabei werden die Bilder auf ein Gitternetz projiziert, hierfür wird Healpix verwendet.

Healpix [25] ist ein von der NASA entwickeltes Prinzip, dabei lässt sich eine Kugel in zwölf annähernd gleich große Vierecke teilen, welche sich wiederum rekursiv vierfach unterteilen lassen. Eine solche Healpix Kugel ist in Abbildung 2.7 zu sehen.



Quelle: HEALPix — a Framework for High Resolution Discretization, and Fast Analysis of Data Distributed on the Sphere[25]

Abbildung 2.7: **Healpix Gitternetz:**
Eine mit Healpix unterteilte Kugel. Die gesamte Kugel lässt sich in 12 Vierecke unterteilen, wobei jedes davon rekursiv 4 Kinder hat. So lässt sich eine Kugel nur in Vierecken darstellen.

2.2.4 Pipeline zur Echtzeitvisualisierung von Terraindatensätzen

In Abbildung 2.8 ist die Pipeline zur Echtzeitvisualisierung von Terraindatensätzen von der Aufnahme der Bilder im Satelliten bis zur Darstellung in der Anwendung zu sehen.

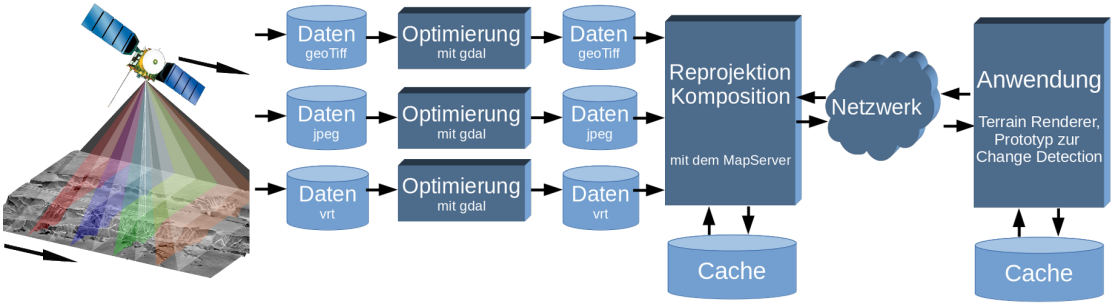


Abbildung 2.8: **Pipeline der Terraindatensätze:**
In der Abbildung ist die Pipeline zur Darstellung von Terraindatensätzen zu sehen. Die Pipeline reicht von der Aufnahme der Bilder vom Satelliten (links) bis zur Darstellung in der Anwendung (rechts).

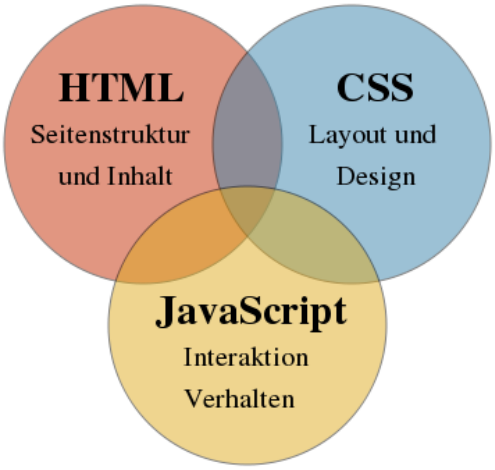
Im linken Teil der Abbildung 2.8 werden die Bilder vom Satelliten aufgenommen. Der Beobachtungssatellit umkreist dabei den Planeten in seiner Umlaufbahn und erstellt Bilder von dem Gebiet unter sich. Auf diese Art und Weise entstehen die typischerweise länglichen Satellitenbilder (bei der HRSC 52 km Breite und mindestens 300 km Länge, siehe 2.2.1). Die damit erstellten Datensätze werden anschließend vorprozessiert, dabei werden die Daten mit dem Softwaretool gdal [26] optimiert. Die Daten werden vom MapServer verwaltet, welcher für die Reprojektion und Komposition verantwortlich ist. Diese Daten werden auf dem Cache des Servers zwischengespeichert. Die Anwendung stellt über das Netzwerk Anfragen an den MapServer nach Bildern innerhalb zweier gewählter Koordinaten, der Bounding Box. Der MapServer antwortet mit einer JSON-Datei welche Informationen zu den Bildern im gewählten Bereich enthält. Diese können anschließend von der Anwendung heruntergeladen werden, alle heruntergeladenen Bilder werden lokal von der Anwendung gecacht.

2.3 Programmierwerkzeuge

Im folgenden werden die verschiedenen Softwarewerkzeuge vorgestellt die bei der Implementierung der Prototypen verwendet wurden.

2.3.1 Programmierung von Webseiten

In der modernen Webentwicklung haben die Webtechniken Hypertext Markup Language (HTML), Cascading Style Sheets (CSS) und JavaScript jeweils eine bestimmte Rolle (siehe Abbildung 2.9). Dies entspricht dem Prinzip der Trennung von Zuständigkeiten. HTML organisiert die Struktur der Webseite, CSS das Design und JavaScript die Interaktion.



Quelle: `https://wiki.selfhtml.org/wiki/Datei:HTML-CSS-JS.svg`

Abbildung 2.9: **Programmierung von Webseiten:**
Die Grafik zeigt die Trennung von Zuständigkeiten in der Programmierung von Webseiten.

HTML

HTML ist eine Dokumentbeschreibungssprache, die beschriebenen Dokumente bilden die Grundlage des Internets. Ein HTML-Dokument enthält den Text einer Website und zusätzlich HTML-Kommandos (Tags, Marken) zu seiner Formatierung, Bilder oder Querverweise auf andere Dokumente (Hyperlinks). Browser führen die Kommandos aus und zeigen die HTML-Seite an. HTML wurde entwickelt, um Forschungsergebnisse des CERN zwischen ihren Standorten auszutauschen, da es nicht möglich war, Informationen auf digitalem Weg einfach, schnell und strukturiert zwischen mehreren Personen zu verschicken. 1992 erschien die erste Version der HTML-Spezifikation.

CSS

Die Gestaltung einer Website erfolgt meist mittels CSS. CSS ist ein sogenannter lebender Standard, das bedeutet er wird ebenso wie HTML beständig vom World Wide Web Consortium (W3C) weiterentwickelt. CSS wurde entwickelt, um das Design einer Website von seinen Inhalten zu trennen.

JavaScript

JavaScript ist eine Skriptsprache, die entwickelt wurde, um in Webseiten Benutzerinteraktionen auszuwerten, Inhalte zu verändern, nachzuladen oder zu generieren. Die Sprache

wurde 1995 von Netscape entwickelt und findet heutzutage auch außerhalb von Browsern Anwendung, zum Beispiel in Microcontrollern. Die Sprache wurde ursprünglich unter dem Namen LiveScript entwickelt, 1996 jedoch in JavaScript umbenannt, um die Popularität von Java zu nutzen.

2.3.2 C++

C++ ist eine objektorientierte Programmiersprache, die von Bjarne Stroustrup als Erweiterung zur Programmiersprache C entwickelt wurde. Bei der Objektorientierung wird ein System durch das Zusammenspiel kooperierender Objekte beschrieben. Die Sprache wird seit 1979 entwickelt und wurde 1985 kommerziell veröffentlicht[27]. Seit dem wird sie kontinuierlich weiterentwickelt und befindet sich aktuell in der Version C++14. Aufgrund der vielen Einsatzmöglichkeiten zählt C++ heute zu den am meisten verwendeten Programmiersprachen (Platz 3 auf dem TIOBE Index August 2017[28]).

Einer der größten Vorteile ist, dass der übersetzte Code als sehr schnell gilt, da C++ eine strikt standardisierte Sprache ist. Die Standardisierung erlaubt es Annahmen über die Semantik des Codes zu treffen, um überflüssige Aufrufe wegzuoptimieren. Im Gegensatz zu anderen Programmiersprachen kompiliert der Compiler direkt in Maschinencode. Dadurch werden Geschwindigkeitsvorteile gegenüber anderen Sprachen erreicht, im Gegensatz zu Java (Platz 1 auf dem TIOBE Index August 2017) wird der Code nicht in einer virtuellen Maschine ausgeführt[29].

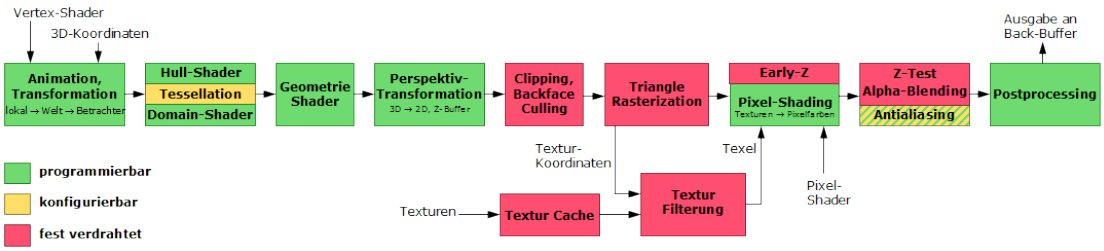
Der Prototyp zur Change Detection in Höhendaten (siehe 4.3) wird in C++ programmiert.

2.3.3 OpenGL

OpenGL steht für Open Graphics Library, was auf deutsch Offene Grafikbibliothek bedeutet. Es ist eine Programmierschnittstelle, die die Entwicklung von 2D- und 3D-Grafikprogrammen unterstützt. Die Software ist in der Programmiersprache C geschrieben und ist sowohl programmiersprachen- als auch plattformunabhängig. Die erste Version wurde 1992 von Mark Segal und Kurt Akeley verfasst, seitdem wird OpenGL weiterentwickelt. Seit 2006 wird es von der Khronos Group weiterentwickelt, die aktuelle Version ist 4.6, welche am 30. Juli 2017 veröffentlicht wurde[30].

OpenGL wurde als Zustandsautomat entworfen, dies bedeutet das Parameter nicht bei jedem Funktionsaufruf übergeben werden müssen. Dadurch bleiben sie aktiv bis ein neuer Zustand gesetzt wird. Farben werden zum Beispiel nur einmal festgelegt und nicht für jeden Vertex neu gesetzt. Für die Verwendung dieses Designs gibt es mehrere Gründe. Jede Änderung eines Zustandes hätte eine aufwendige Reorganisation der Grafikpipeline zur Folge (siehe Abbildung 2.10). Darüber hinaus müssen manche Zustände selten, beziehungsweise nie neu gesetzt werden. Für den Programmierer wäre es mit einem erheblichen Mehraufwand verbunden, wenn alle Parameter für jeden Vertex neu gesetzt

werden müssen. Ein großer Vorteil von OpenGL ist die Erweiterbarkeit. Grafikhersteller können die Zustandsmaschine um eigene Zustände erweitern[31].



Quelle: `https://upload.wikimedia.org/wikipedia/commons/5/54/3D-Pipeline.png`

Abbildung 2.10: **OpenGL Pipeline:**
Zuerst werden die Vertexdaten der Primitiven und Darstellungslisten berechnet, dies geschieht durch verschiedene Transformationen. In OpenGL gibt es aktuell drei Primitive. Primitive sind einfache geometrische Formen, die als Grundflächen verwendet werden. Diese sind Punkte, Linien sowie Dreiecke. Anschließend werden sie zu Fragmenten gerastert. Fragmente sind Pixeldaten welche zusätzliche Informationen wie zum Beispiel Texturkoordinaten beinhalten. Im letzten Schritt werden die Daten aus dem Frame Buffer geladen und auf dem Bildschirm dargestellt.

3 Entwicklung eines Prototyps zur Change Detection in Satellitenbildern

Zuerst sollte ein Prototyp implementiert werden, welcher die Erkennung von Veränderungen in Satellitenbildern unterstützen sollte. Dafür waren vier Arbeitsschritte notwendig. Zuerst sollten die Anforderungen festgehalten werden, danach musste ein Konzept erarbeitet werden. Im Anschluss daran sollte der Prototyp implementiert werden. Im letzten Schritt sollten die erzielten Ergebnisse evaluiert werden.

3.1 Anforderungen

Im ersten Schritt sollten die Anforderungen an den Prototypen festgelegt werden. Der Prototyp sollte dazu in der Lage sein den Nutzer bei der Erkennung von Veränderungen in der Textur von Satellitenbildern zu unterstützen. Um Veränderungen in den Satellitenbildern zu bemerken schickt der Nutzer eine Anfrage an den MapServer, um alle Bilder innerhalb eines bestimmten lokalen (Bounding Box) und temporalen Bereiches zu erhalten. Im Standardanwendungsfall wird er anschließend mit einem Slider alle Bilder von alt nach neu betrachten und nach Änderungen suchen, dabei wird er einzelne Bilder genauer vergleichen. Wenn er interessante Stellen gefunden hat wird er die Bounding Box verfeinern, um die Stellen genauer untersuchen zu können. Die entsprechenden Anforderungen an den Prototypen lauten:

- **Dynamisches Laden von Satellitenbildern:** Die Anwendung soll die Satellitenbilder von einer Schnittstelle beziehen können. Diese Schnittstelle soll Datensätze verschiedener Quellen zur Verfügung stellen. Die Datensätze sollen dabei mit möglichst geringem Implementierungsaufwand ausgetauscht werden können.
- **Unterschiedliche Gewichtung der einzelnen Bilder:** Der Nutzer sollte die Möglichkeit haben die einzelnen Satellitenbilder mit einer unterschiedlichen Gewichtung in die Szene einfließen zu lassen. Damit soll sichergestellt werden, dass es möglich ist, jedes Bilder mit jedem zu vergleichen.
- **Unterstützung des Standardanwendungsfalls:** Es sollte für den Nutzer möglichst einfach und intuitiv sein, die häufigste Bedienweise der Anwendung durchzuführen. In dieser werden alte Bilder mit den neuen Bildern überlagert.

- **Konfigurationsmöglichkeiten:** Der Nutzer soll die Szene über verschiedene Konfigurationsmöglichkeiten beeinflussen können.
- **Entwicklung einer wiederverwendbaren GUI:** Die zu entwickelnde Benutzeroberfläche sollte so erstellt werden, dass sie in der zweiten Aufgabe (siehe 4.3) wiederverwendet werden kann. Darüber hinaus muss die Benutzeroberfläche dazu in der Lage sein, die vorherigen Anforderungen zu erfüllen.

3.2 Konzept

Im zweiten Schritt der Entwicklung des Prototyps wurde das Konzept erarbeitet um die Anforderungen zu erfüllen.

Dynamisches Laden von Satellitenbildern

Zuerst wird das dynamische Laden der Satellitenbilder betrachtet. Damit der Implementierungsaufwand beim Austausch der Datensätze möglichst gering bleibt, soll eine Schnittstelle verwendet werden, die die Daten zur Verfügung stellt. Hierfür wird der MapServer verwendet (siehe 2.2.2). Dieser stellt einen Webservice zur Verfügung, mit welchem die einzelnen Datensätze geladen werden können. Wenn die Datensätze anschließend ausgetauscht werden sollen, sind lediglich geringe Änderungen in der Anwendung notwendig. Es muss die URL bei der Serveranfrage angepasst werden sowie in einigen Fällen der JSON Parser, welcher benötigt wird, um die einzelnen Anfragen genauer aufzuschlüsseln.

Unterschiedliche Gewichtung der einzelnen Bilder

Um jedes Bild mit jedem zu vergleichen, ist es notwendig die Transparenz jedes einzelnen Bildes anpassen zu können. Damit dies ermöglicht wird, sollen alle Bilder in dem angefragtem Bereich in einer Liste mittels einer kleinen Vorschau dargestellt werden. Neben dieser Vorschau sollte zusätzlich das Aufnahmedatum stehen, um den Zeitpunkt der Veränderung herauszufinden. Zur Veränderung der Transparenz dieses Bildes erhält jedes Listenelement einen Slider.

Unterstützung des Standardanwendungsfalls

Beim Standardanwendungsfall werden die alten Bilder mit dem jeweils neuerem überlagert. Um das möglichst einfach zu gestalten, soll ein großer Slider verwendet werden. Dieser enthält für jedes in der Szene vorkommendes Bild eine Position und überlagert alle älteren Bilder mit dem aktuellem Bild der aktuellen Position. Neben diesen Slider soll noch ein zweiter Slider implementiert werden, der das aktuell ausgewählte Bild in der Zeitspanne zwischen dem ältestem und neustem Bild einordnet.

Konfigurationsmöglichkeiten

Um die Szene zu beeinflussen, sollen verschiedene Einstellungsmöglichkeiten implementiert werden. Es soll die Möglichkeit gegeben werden die Satellitenbilder zeitlich zu filtern. Außerdem soll es möglich sein, eine neue Bounding Box zu definieren, um einen anderen Ausschnitt aus den Daten zu sehen.

Entwicklung einer wiederverwendbaren GUI

Zur Entwicklung der Benutzeroberfläche soll auf die Sprachen HTML und JavaScript zurückgegriffen werden. Mittels dem Toolkit ViSTA können HTML-Seiten als Benutzeroberfläche verwendet werden. Da der in Kapitel 4 zu entwickelnde Prototyp eine ähnliche GUI besitzen soll, kann so garantiert werden, dass die GUI wiederverwendbar ist. Um die oben beschriebenen Anforderungen zu erfüllen, wurde ein Mockup erarbeitet. Dieser ist in Abbildung 3.1 zu sehen.

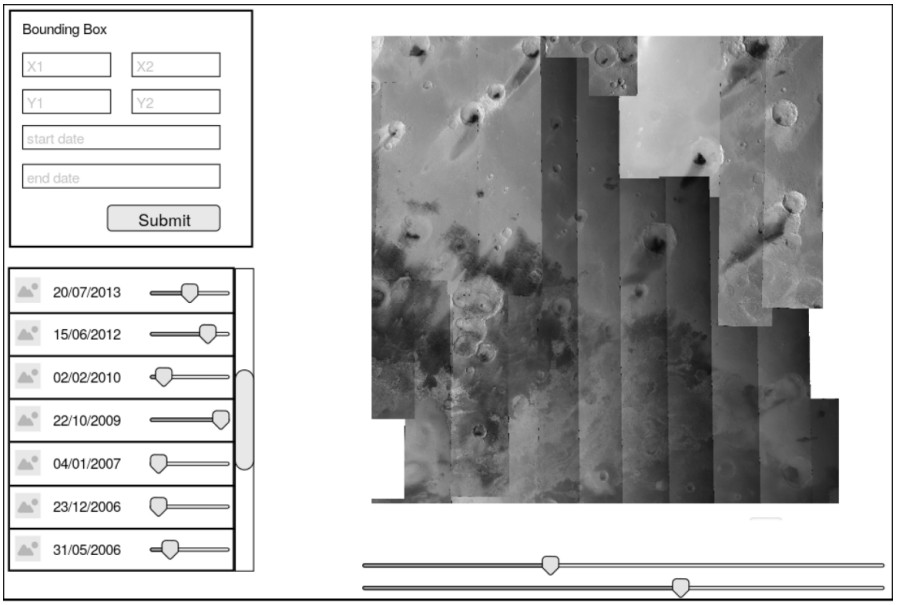


Abbildung 3.1: **Mockup des Prototyps zur Change Detection in Satellitenbildern**

Das Fenster wurde in zwei Bereiche unterteilt, links die Steuerung und rechts die aus den Satellitenbildern zusammengesetzte Szene. Im linken oberen Bereich befindet sich ein Formular, in diesem lässt sich die Bounding Box festlegen und die Bilder lassen sich nach Zeitraum filtern. Es werden nur Bilder in der Szene angezeigt, die zwischen dem Start- und Enddatum aufgenommen wurden. Im unteren Bereich in der Steuerung befindet sich eine Liste. In dieser werden alle Bilder in einer kleinen Vorschau angezeigt, die sich innerhalb der Bounding Box und der Zeitspanne zwischen Start- und Enddatum

befinden. Daneben befindet sich das Aufnahmedatum sowie ein Slider der die Transparenz der Bilder steuert. Im rechten Bereich des Fensters befinden sich zwei Slider sowie die verschiedenen Bilder der Szene. Die beiden Slider besitzen für jedes Bild eine Position, beim unteren Slider sind diese gleichmäßig verteilt. Sie sind beide miteinander verknüpft, wenn ein Slider bewegt wird, bewegt sich der andere mit. Beim oberen Slider sind die einzelnen Positionen mit den zugehörigen Bildern relativ zum Aufnahmedatum verteilt. Über den Slidern wird die Szene dargestellt, neue Bilder überlagern ältere Bilder, wenn ihre Transparenz nicht reduziert wird.

3.3 Implementierung

Im nächsten Schritt musste das Konzept umgesetzt und der Prototyp implementiert werden. Hierfür wurde zuerst die GUI implementiert. Die Textfelder sind Standard HTML Inputs. Der Liste wird jeweils ein Element hinzugefügt, welches ein Bild für die Vorschau, ein Text für das Datum sowie einen Slider (*range*) für die Transparenz enthält. Alle zu ladenden Bilder werden in einem Container absolut übereinander platziert. Dadurch wird jeweils das Bild in der unteren Schicht sichtbar, wenn die Transparenz des überlagernden Bild verringert wird. Sowohl die kleinen Slider, als auch die großen Slider unter dem Container, sind *noUiSlider*[32]. *noUISlider* sind ebenso wie *range* HTML5 Slider, bieten zusätzlich jedoch noch diverse Möglichkeiten, den Slider zu designen. Zum Design der übrigen Elemente wird das MaterializeCSS [33] verwendet, da in der Terrain-Rendering Software ebenfalls auf dieses Framework gesetzt wird. Das Material Design wurde ursprünglich von Google entwickelt und gestaltet.

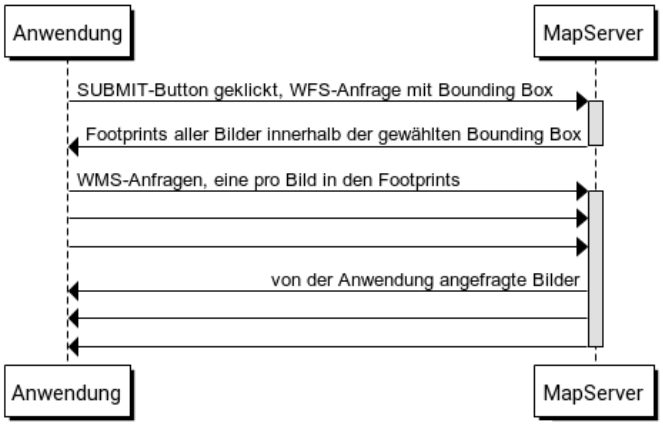


Abbildung 3.2: **Sequenzdiagramm der Anwendung-MapServer Kommunikation**

Die im folgenden beschriebene Anwendung-MapServer-Kommunikation ist in Abbildung 3.2 dargestellt. Nach einem Klick auf den *SUBMIT*-Button wird zuerst ein URL gebaut.

Dieser enthält eine WFS-Anfrage an den MapServer, in dieser muss der Datensatz angegeben werden, welcher angefragt werden soll, sowie die Bounding Box. Mittels jQuery wird anschließend eine WFS-Anfrage an den MapServer geschickt. Dieser antwortet mit einer JSON-Datei, welche alle Footprints in diesem angefragtem Bereich enthält. Footprints sind die Metainformationen der Bilder, sie enthalten die Umrisse des Bildes sowie weitere Informationen wie Aufnahmezeitpunkt, Name oder Aufnahmeort. Mit den in der JSON-Datei hinterlegten Bildnamen werden einzelne WMS-Anfragen für jedes Bild an den MapServer gestellt. Die Bilder werden sortiert, um das Start- und Enddatum des Datensatzes zu bestimmen. Für jedes Bild wird der Container anschließend erweitert und es wird ein neuer Eintrag in der Liste hinzugefügt.

Um die Benutzung der Anwendung für den Nutzer zu erleichtern, ist in der rechten oberen Ecke die aktuelle Mausposition über der Szene in geographischen Koordinaten angegeben. Zusätzlich wird das zugehörige Bild in der Szene farblich hervorgehoben wenn sich der Mauszeiger über dem zugehörigen Listenelemente befindet. Dies ist, neben der fertigen Anwendung, in Abbildung 3.3 zu sehen.

Im Laufe der Programmierung des Prototyps wurde geringfügige Änderungen an der Benutzeroberfläche vorgenommen. Die Liste mit den einzelnen Bildern befindet sich nicht mehr unter dem Formular, sondern rechts daneben. Dies hat mehrere Gründe. Wenn die Liste weiterhin unter dem Formular liegen würde, müssten die Bilder in dem Container entweder größer angefragt oder skaliert werden. Das größere Anfragen würde darin resultieren, dass das Laden der Seite deutlich mehr Zeit in Anspruch nimmt. Beim Skalieren würden die einzelnen Bilder verzerrt werden. Außerdem müsste die Liste in der Länge gekürzt werden.

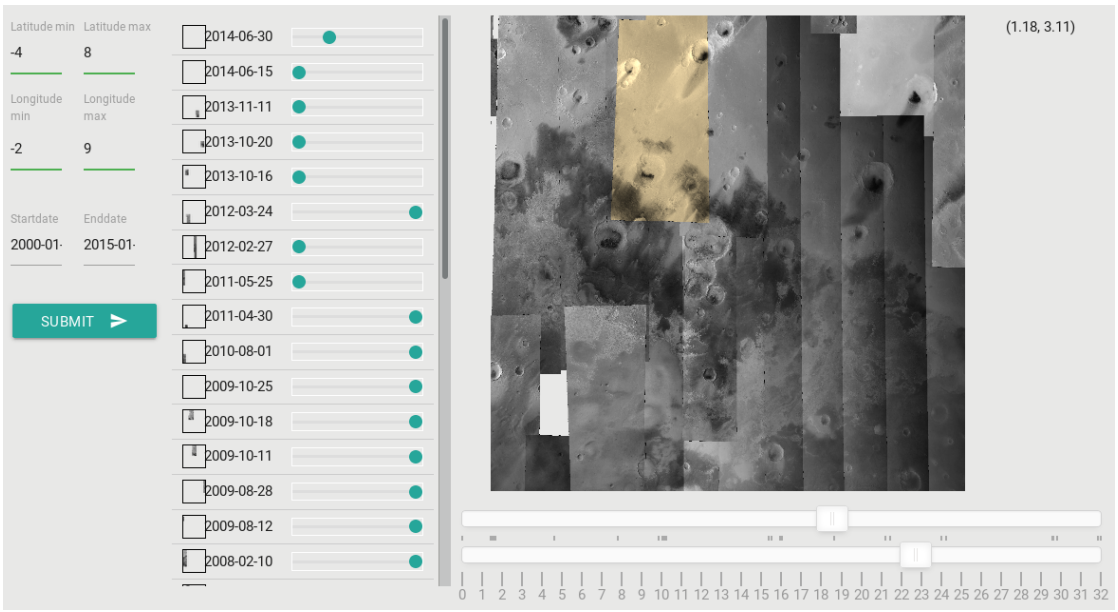


Abbildung 3.3: **Prototyp zur Change Detection in Satellitenbildern:**

3.4 Evaluierung der Ergebnisse

Der Prototyp war nach der Implementierung dazu in der Lage, Satellitenbilder in einem bestimmten lokalen und temporalen Bereich vom MapServer anzufragen und sie übereinander gelagert in einem Bild-Container darzustellen. Um dem Nutzer bei der Erkennung von Veränderungen in den Bilddaten zu unterstützen, kann die Transparenz jedes Bildes beliebig konfiguriert werden.

Anschließend an die Implementierung sollte evaluiert werden, inwieweit der Prototyp dem Nutzer bei der Erkennung von Veränderungen unterstützt. In dem verwendeten Datensatz der HRSC gestaltete sich das Erkennen von Terrainänderungen als schwierig, da für geringe Veränderungen die Auflösung, für große Veränderungen der Aufnahmezeitraum zu gering ist. Der Prototyp ermöglicht es jedoch Wetterereignisse wie zum Beispiel Sandstürme auf dem Mars zu erkennen.

Im Sommer 2011 wurden mit der HRSC Sandstürme in Arcadia Planitia aufgenommen[34]. Arcadia Planitia ist eine von Lavaströmen durchflossene Ebene auf dem Mars. Zur Evaluierung sollte im Prototypen die Region angefragt und der Sandsturm gefunden werden. Nach der ersten großen Anfrage der Region, wurde der Bildbereich nach und nach verfeinert, in Abbildung 3.4 ist der Sandsturm auf dem rechten Bild zu sehen. Das entsprechende Bild in der Szene wurde hervorgehoben.

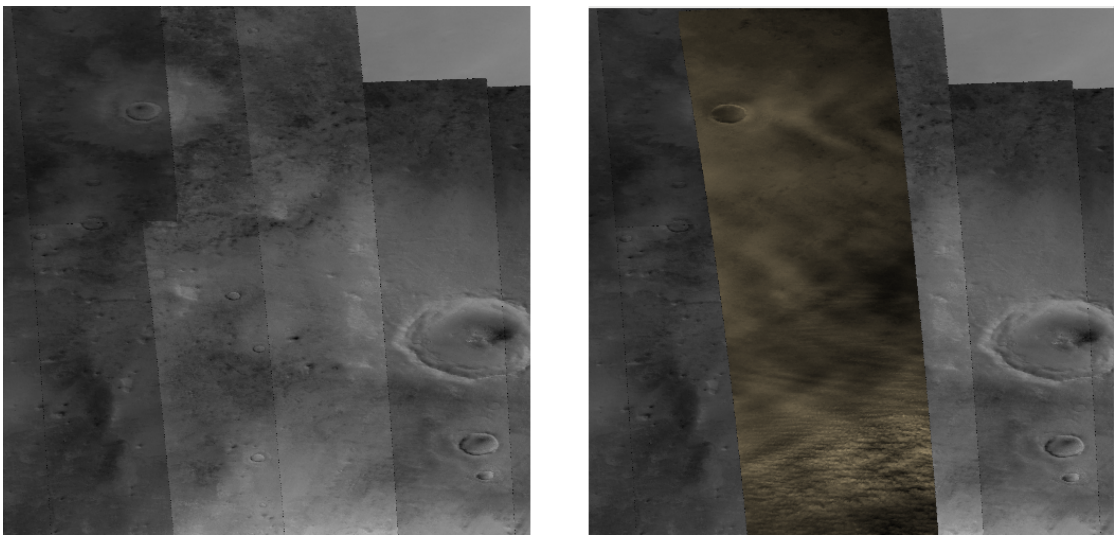


Abbildung 3.4: **Sandsturm in Arcadia Planitia:**
Auf dem rechten Bild ist der Sandsturm auf der entsprechenden Satellitenaufnahme hervorgehoben. Im Gegensatz zum linken Bild werden kleinere Krater im unteren Bildbereich überlagert. Ebenso ist die wellenförmige Struktur des Sturmes erkennbar.

Der Prototyp eignet sich also durchaus zum Erkennen von Veränderungen in Satellitenbildern. Im HRSC-Datensatz können so Sandstürme und andere Wetterereignisse gefunden werden. Die Erkennung von Terrainänderungen stellte sich aufgrund der geringen Auflösung als schwierig heraus.

4 Entwicklung eines Prototyps zur Change Detection in Höhendaten

Im nächsten Schritt sollte der Prototyp zur Change Detection in Höhendaten entwickelt werden. Dieser Bereich soll den größten Teil der Bachelorarbeit einnehmen. Um den Nutzer bei der Erkennung von Veränderungen möglichst gut zu unterstützen, sollten verschiedene Visualisierungsmethoden erarbeitet werden. Dafür waren vier Arbeitsschritte notwendig. Zuerst sollten die Anforderungen definiert und anschließend das Konzept erarbeitet werden. Dann sollte der Prototyp implementiert und zum Abschluss evaluiert werden.

4.1 Anforderungen

Der Prototyp soll den Nutzer dabei unterstützen, Veränderungen in Höhendaten zu erkennen. Im Zuge dessen sollten verschiedene Visualisierungsmethoden erarbeitet werden. Die Anforderungen an solch einen Prototypen lauten:

- **Rendern des Terrains:** Die wohl wichtigste Aufgabe des Prototyps ist es, das Terrain zu rendern. Unter Rendern versteht man die Erzeugung eines Bildes aus Rohdaten, zum Beispiel aus geometrischen Beschreibungen. Die Höhendaten der einzelnen Bilder sollten mittels OpenGL in 3D in die Szene übertragen werden, so dass es möglich wird, das Gelände interaktiv zu erkunden.

- **Anpassung der Benutzeroberfläche des vorherigen Prototyps:** Eine Anforderung des *Prototyps zur Change Detection in Satellitenbildern* (siehe 3.1) war es die Benutzeroberfläche so zu entwickeln, dass sie in dieser Aufgabe wiederverwendet werden kann. Dadurch, dass die Anwendung nun aber nicht mehr 2D-Bilder, sondern 3D-Höhendaten anzeigen sollte, mussten einige Änderungen in der GUI vorgenommen werden. Die Benutzeroberfläche sollte im Vordergrund stehen, während die 3D-Szene im Hintergrund gerendert wird. Da zusätzlich verschiedene Visualisierungsmethoden entwickelt werden sollten, ist eine zusätzliche Anpassung der Benutzeroberfläche notwendig.

- **Entwicklung verschiedener Visualisierungsmethoden:** Um Höhenänderungen in den Terraindaten möglichst effektiv zu erkennen, sollten verschiedene Visualisierungsmethoden entwickelt werden. Hierbei sind verschiedene statische oder

dynamische Methoden denkbar. Die verschiedenen Methoden sollen im Konzept (siehe 4.2) herausgearbeitet werden und danach implementiert werden (siehe 4.3).

- **Konfigurationsmöglichkeiten:** Hierbei sind zwei verschiedene Fälle zu betrachten. Die Szene sollte hinsichtlich der Position oder dem Aufnahmedatum der einzelnen Bilder anpassbar sein. Die Visualisierungsmethoden sollten konfigurierbar sein, um möglichst gute Ergebnisse in verschiedenen Gebieten auf der Erde oder anderen Planeten zu gewährleisten.

- **Erfüllung der weiteren Anforderungen aus 3.1:** Der Prototyp sollte ebenfalls die noch fehlenden Anforderungen aus 3.1 unterstützen. Dazu gehört das dynamische Laden der Satellitendaten, die unterschiedliche Gewichtung der Bilder sowie die Unterstützung der Standardanwendungsfälle.

4.2 Konzept

Im nächsten Schritt sollte das Konzept erarbeitet werden um die Anforderungen an den Prototypen zu erfüllen.

Rendern des Terrain

Das Terrain in 3D zu rendern ist die wohl wichtigste Aufgabe des Prototyps. Die fertige Anwendung soll aus insgesamt drei C++ Klassen, einer UserInterface-, DataManager- und ChangeDetection-Klasse, sowie der in HTML und JavaScript programmierten Benutzeroberfläche bestehen.

Die UserInterface-Klasse soll das Window erstellen und die Integration der HTML/JavaScript Benutzeroberfläche in die Anwendung implementieren. Um Informationen zwischen der C++ Anwendung und der Benutzeroberfläche auszutauschen, müssen verschiedene Callbacks implementiert werden.

Der DataManager soll die Verwaltung der benötigen Satellitendaten implementieren. Er erhält von dem UserInterface alle benötigten URLs der einzelnen Satellitenbilder in einem String. Der DataManager ist für das Parsen der URLs sowie den Download der Bilder verantwortlich. Die Bilder werden dabei im *geo Tagged Image File Format* (geoTIFF) empfangen. geoTIFF ist ein Bildformat indem eine Georeferenz, zusätzlich zu den sichtbaren Rasterdaten, in die Bilddatei eingebettet wird. Ebenso wird im DataManager das Caching implementiert.

Die Klasse ChangeDetection soll das Rendering des Terrain implementieren. Um das Terrain zu rendern, werden die Pixel der einzelnen Satellitenbilder durchiteriert und es wird die Höhe aus ihnen entnommen. Diese soll anschließend in 3D mit OpenGL gerendert werden. Die zu entwickelnden statischen und dynamischen Visualisierungsmethoden werden ebenso in der ChangeDetection-Klasse implementiert.

Die Benutzeroberfläche registriert die Benutzereingaben und die Interaktion mit den einzelnen Bedienungselementen und überträgt diese Daten an die C++ Anwendung. Sie stellt eine Anfrage an den MapServer nach den Footprints der Bilder, die sich innerhalb der vom Nutzer gewählten Bounding Box befinden. Die entsprechenden URLs werden an das UserInterface weitergereicht.

Anpassung der Benutzeroberfläche des vorherigen Prototyps

Die größte Änderung der Benutzeroberfläche ist notwendig, da die Szene nun in 3D gerendert wird und nicht mehr nur 2D-Bilder angezeigt werden. Der in 3.3 verwendete Container für die Bilder wird nicht mehr benötigt, da die Szene in 3D hinter der GUI gerendert wird.

Um die Visualisierungsmethoden zu benutzen, musste die Oberfläche der Anwendung angepasst werden. Unter der Liste mit den einzelnen Bildern, sollte eine eine zweite Liste mit den verschiedenen Visualisierungsmöglichkeiten entstehen. Diese sollten sich mittels Checkboxen Ein- und Ausschalten lassen. Um weitere Konfigurationsmöglichkeiten zu ermöglichen, befindet sich nun außerdem ein *SETTINGS*-Button in der Anwendung. Dieser öffnet ein Menü, in dem die Visualisierungsmethoden angepasst werden können. Der *HIDE*-Button soll es ermöglichen, die Benutzeroberfläche auszublenden und das Terrain im Vollbildmodus anzuzeigen. Weitere Buttons die im Laufe der Implementierung erforderlich sind, sollen neben dem *SETTINGS/HIDE*-Button platziert werden.

Das Mockup für die Benutzeroberfläche ist in Abbildung 4.1 zu sehen.

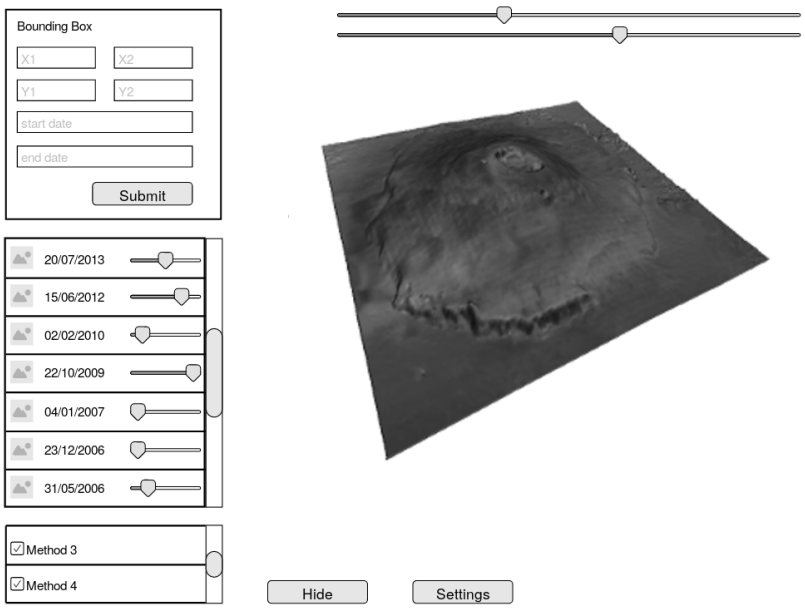


Abbildung 4.1: **Mockup des Prototyps zur Change Detection in Höhendaten**

Entwicklung verschiedener Visualisierungsmethoden

Die Entwicklung der Visualisierungsmethoden soll den Hauptteil der Arbeit einnehmen. Die Visualisierungsmethoden sollen sich in zwei verschiedene Gruppen aufteilen lassen, statische und dynamische Visualisierungsmethoden. Die statischen Methoden sind dabei unabhängig von dem zuletzt betrachtetem Satellitenbild. Die dynamischen Methoden machen jeweils Unterschiede, zwischen dem aktuell betrachtetem Bild zu den vorherigen in der Szene deutlich.

Bei den statischen Methoden sollen zuerst die implementiert werden, die sich in der Grundlagenrecherche als besonders geeignet herausstellten (siehe 2.1.3). Hierzu zählt unter anderem Implementierung einer Farbskala, die die Pixel je nach Höhe unterschiedlich einfärbt, sowie die Verwendung eines Gitternetzes. Da es sich um eine dynamische Veränderung des Geländes handelt, sollen auch Höhenlinien implementiert werden, da sie sich laut Butkiewicz und Stevens [15] als am effektivsten bewiesen haben. Weitere mögliche statische Methoden sollen im Laufe der Implementierung erarbeitet werden.

Bei den dynamischen Methoden soll zuerst eine Methode entwickelt werden, die die Unterschiede des aktuell beobachtetem Bild zu allen vorherigen Bildern verdeutlichen soll. Weitere mögliche dynamische Methoden können im Laufe der Implementierung erarbeitet werden.

Konfigurationsmöglichkeiten

Um die Szene anpassen zu können, soll es wie in 3.3 ein Formular geben, in dem die Bounding Box sowie das Start- und Enddatum festgelegt werden können. Um die einzelnen Visualisierungsmethoden anzupassen, soll sich nach dem Klick auf den *SETTINGS*-Button im Mockup 4.1 ein neues Formular öffnen. In diesem sollen die Methoden angepasst werden, zum Beispiel kann hier der Abstand zwischen Höhenlinien oder den Linien im Gitternetz festgelegt werden.

Erfüllung der weiteren Anforderungen aus 3.1

Es wird auf die gleiche Anwendung-MapServer Architektur (siehe Abbildung 3.2) gesetzt, um die Satellitendaten dynamisch zu laden. Der einzige Unterschied besteht im Laden der einzelnen Höhenfeldern aus den Satellitenbildern, dies soll jetzt in der C++ Anwendung stattfinden. JavaScript fragt lediglich die Footprints im ausgewählten Bereich an und übermittelt anschließend die URLs zu den einzelnen Bildern. Beim Austausch von Datensätzen muss nur der MapServer-Request in der GUI angepasst und einzelne Änderungen am JSON-Parser vorgenommen werden.

Um die Höhenfelder in der Szene mit unterschiedlichem Gewicht einzubringen, erhält jedes Höhenfeld einen Slider mit dem die Gewichtung festgelegt wird. Der Wert bestimmt, mit welcher Prozentzahl es in die vorhandene Szene gemixt wird.

Um den Standardanwendungsfall zu unterstützen, werden die gleichen zwei Slider wie bei der Implementierung des vorherigen Prototyps verwendet (siehe 3.3). Neue Höhenfelder überlagern dabei die alten komplett.

4.3 Implementierung

Die Hauptaufgabe war nun den Prototypen nach dem Konzept und konform zu den Anforderungen zu implementieren. Hierfür waren fünf Arbeitsschritte vorgesehen:

1. Entwicklung der Benutzeroberfläche
2. 3D-Rendering der Höhendaten eines Satellitenbildes
3. 3D-Rendering mehrerer Höhenfelder
4. Entwicklung statischer Visualisierungsmethoden
5. Entwicklung dynamischer Visualisierungsmethoden

4.3.1 Entwicklung der Benutzeroberfläche

Da es eine Anforderung in 3.1 war, die GUI möglichst wiederverwendbar zu entwickeln, mussten nur kleine Anpassungen vorgenommen werden. Der Container für die Bilder wurde entfernt, es wurde eine zweite Liste für die Visualisierungsmethoden und Buttons zum Verstecken der GUI sowie für die Einstellungen hinzugefügt. Die Präsentation der Daten findet nicht mehr in der Benutzeroberfläche statt, sondern komplett im Hintergrund dieser. Die Liste zum Ein- und Ausschalten der Methoden enthält jeweils für jede Methode eine eigene Checkbox. Beim Design wurde das Material-Design verwendet, welches auch schon im vorherigen Prototyp benutzt wurde.

Die Benutzeroberfläche ist in insgesamt fünf Bereiche (siehe Abbildung 4.2) aufgeteilt.

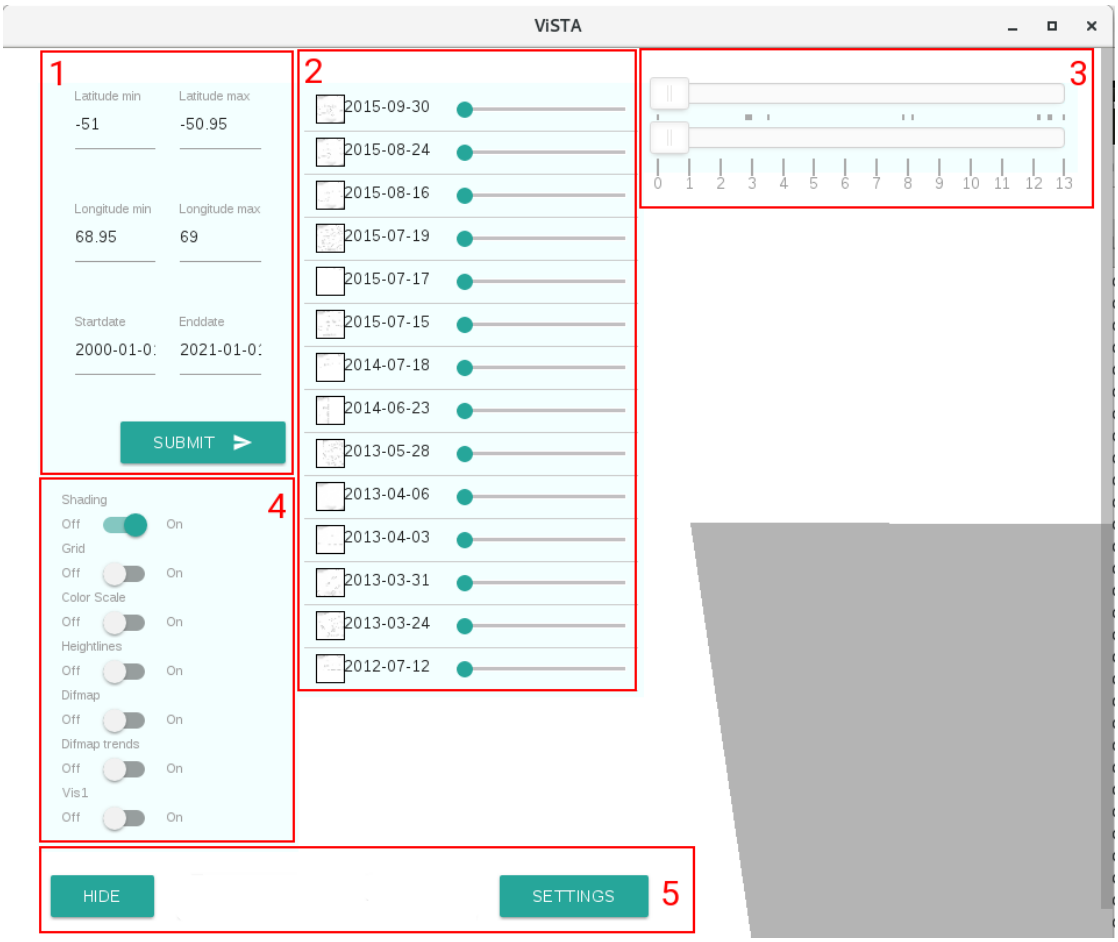


Abbildung 4.2: **Benutzeroberfläche des Prototyps:**
In der Abbildung ist die Benutzeroberfläche des Prototyps zu sehen. Im unteren rechten Bildbereich ist eine leere gerenderte Szene zu sehen.

- **Bereich 1:** Über die Felder *Latitude min, Latitude max, Longitude min* und *Longitude max* wird die Bounding Box festgelegt. Über die Felder *Startdate* und *Enddate* wird die Zeitspanne festgelegt, in welcher die Bilder aufgenommen worden sind. Nach Klick auf den *SUBMIT*-Button wird mittels JavaScript der URL gebaut, um die Bilder, die innerhalb der Bounding Box liegen, auf dem MapServer anzufragen. Der MapServer liefert als Antwort eine JSON-Datei, die Informationen zu Bildern im angefragten Bereich enthält. Anschließend werden diese Bilder zeitlich geordnet und Bilder die außerhalb des Start- und Enddatums liegen aussortiert. Alle noch relevanten Bilder werden nun an die in Bereich 2 liegende Liste übergeben. Die URLs der Bilder werden mittels eines Callbacks von JavaScirpt an die UserInterface-Klasse der C++ Anwendung übergeben.
- **Bereich 2:** Die in Bereich 2 liegende Liste enthält alle Höhenfelder, die innerhalb der Bounding Box und zwischen dem Start- und Enddatum liegen. Es gibt für jedes Höhenfeld einen Eintrag, dieser enthält eine Vorschau, das Aufnahmedatum und einen Slider. Der Slider bestimmt das Verhältnis, mit welchem das Höhenfeld und die vorherige Szene gemixt werden. Wenn die einzelnen Slider bewegt werden, wird über einen Callback die ID und der Wert des Sliders an die C++ Anwendung übergeben.
- **Bereich 3:** Der dritte Bereich enthält zwei Slider. Mit diesen Slidern wird die in den Anforderungen geforderte Unterstützung des Standardanwendungsfalls erfüllt. Die Slider enthalten alle relevanten Höhenfelder, im unteren Slider sind sie gleichmäßig verteilt und im oberen relational zum Aufnahmedatum. Es werden alle Höhendaten der Satellitenbilder bis zur aktuellen Position der Slider gerendert, neuere Bilder überlagern dabei die älteren.
- **Bereich 4:** In Bereich 4 befinden sich verschiedene Checkboxes für die einzelnen Visualisierungsmethoden. Diese sind als Kippschalter (toggle switches) designt. Wenn die Stellung des Schalters geändert wird, wird die neue Schalterstellung an die C++ Anwendung übergeben.
- **Bereich 5:** Im fünften Bereich befinden sich zusätzliche Buttons. Der *HIDE*-Button macht die komplette Benutzeroberfläche unsichtbar, um die gerenderte Szene in Vollbild anzuzeigen. Dies wird über die jQuery-Methode *$.hide()* erreicht, nach erneuten Klick auf den Button die GUI durch *$.show()* wieder sichtbar gemacht. Über den *SETTINGS*-Button wird ein neues Formular geöffnet. In diesem können die Parameter der einzelnen Visualisierungsmethoden angepasst werden.

4.3.2 3D-Rendering der Höhendaten eines Satellitenbildes

Das 3D-Rendering der Höhendaten findet komplett in der C++ Anwendung statt. Die C++ Anwendung besteht aus den UserInterface, DataManager und ChangeDetection

Klassen. In der Main-Methode wurde als erstes ein neues VistaSystem initialisiert. ViSTA ist eine Software-Toolkit, welches wissenschaftliche Anwendungen mit Methoden und Techniken der Visualisierung unterstützen soll. ViSTA ermöglicht es auch einer Szene sogenannte *VistaGuiItems* hinzuzufügen, diese können auch HTML-Dokumente sein. Über diese Funktion wird die mit HTML entwickelte GUI der Anwendung hinzugefügt. In diesen *VistaGuiItems* können Callbacks registriert werden. Ein solcher Callback wird in der UserInterface-Klasse auf die JavaScript Methode *SubmitClicked()* registriert, die Methode wird nach einem Klick auf den *SUBMIT*-Button aufgerufen. Wenn der Callback ausgelöst wird, wird aus der Benutzeroberfläche ein langer String an die Main-Methode übergeben. Dieser enthält alle URLs zu den Satellitenbildern aus den die Höhendaten entnommen werden, jeweils durch einen Seperatorzeichen getrennt. Die URLs können nicht in einer Liste übergeben werden, da es nur möglich ist, Strings, Doubles und Boolsche Variablen zu übermitteln. Mit diesem URL wird eine neue Instanz der ChangeDetection-Klasse initialisiert.

Im Konstruktor der ChangeDetection-Klasse wird zuerst der lange URL-String in eine Liste mit allen Strings zerlegt. Da zuerst nur ein einziges Bild gerendert werden sollte, war es nur notwendig, das erste Element der Liste herunterzuladen, dieses wurde mittels cURLpp [35] geladen. cURLpp ist ein C++ Wrapper für libcURL [36], welches eine Bibliothek zum Herunter- und Hochladen von Daten ist. Um den Vertex-Buffer zu binden, wird anschließend eine Struktur angelegt, diese hat die gleiche Größe wie die vom MapServer angefragten Bilder, 500 mal 500 Pixel. Die Höhendaten werden als 2D-Textur auf die Grafikkarte geladen, um sie dort zu rendern. Um diese Textur zu erstellen, wird durch jedes Pixel des heruntergeladenen geoTIFF-Bildes iteriert und die Höhe in diesem Pixel einem C++ Vektor hinzugefügt. Über den OpenGL Befehl *glTexImage2D()* wird die Textur für das Shader-Programm lesbar. Die *Do()*-Methode der ChangeDetection Klasse wird jeden Frame aufgerufen. Mittels GLSL-Uniforms wird die Textur dem Shaderprogramm übergeben und auf die GPU geladen.

Im Vertex-Shader wird zuerst der Normalenvektor jedes Pixels bestimmt, dieser wird im Vertex-Shader für die Berechnung des Lichteinfalls benötigt. Der Normalenvektor wird durch das Kreuzprodukt der direkten Nachbarn bestimmt. Außerdem wird die genaue Position des Pixels im Raum berechnet. Der Normalenvektor und die Position werden an den Fragment-Shader übergeben. In diesem wird nur der Lichteinfall in der Szene berechnet, um das Gelände gut sichtbar zu machen und nicht nur in einer Farbe zu rendern.

Das gerenderte Bild ist in Abbildung 4.3 zu sehen.

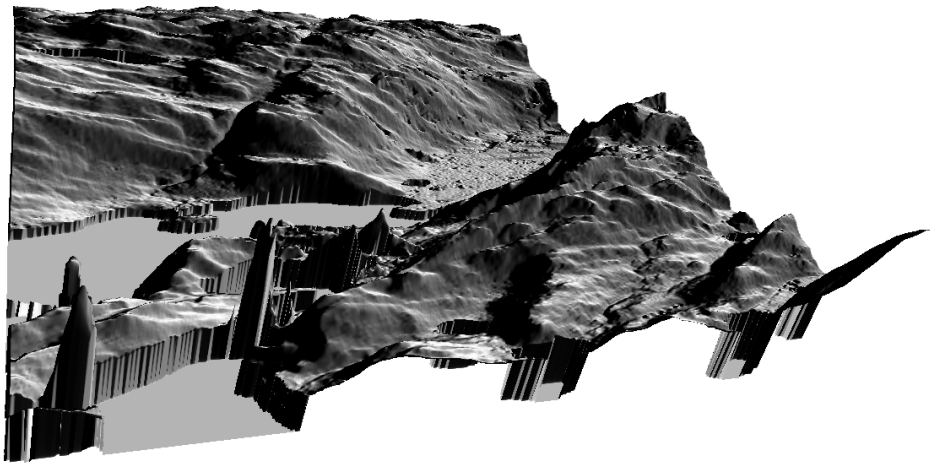


Abbildung 4.3: **In 3D gerendertes Höhenfeld:**
In dieser Abbildung ist ein Ausschnitt des Gebietes rund um den Jakobshavn Gletscher in Grönland zu sehen. An den komplett glatten Stellen im linken Bildbereich sind keine Satellitendaten vorhanden.

4.3.3 3D-Rendering mehrerer Höhenfelder

Nachdem ein Höhenfeld gerendert werden konnte, sollte die Anwendung dahingehend angepasst werden, dass nun auch mehrere Höhenfelder in einer Szene gerendert werden können. Hierfür mussten mehrere Änderungen im Quellcode vorgenommen werden. Statt eines Höhenfeldes, werden nun alle aus der URL-Liste heruntergeladen.

Hier wurde schnell festgestellt, dass der Download aller Bilder einer Szene bei jedem Programmstart sehr lange dauern kann. Aus diesem Grund wurde Caching implementiert. Um auch Bilder aus Anfragen mit unterschiedlichen Bounding-Boxes über verschiedene Starts der Anwendung hinweg zu cachen, werden die Bilder mit der jeweiligen URL als Dateinamen gespeichert.

Da nun mehrere Bilder auf die Grafikkarte geladen werden müssen, reicht eine 2D-Textur nicht mehr aus. Da jedoch im OpenGL-Standard festgelegt ist, dass das Laden von nur minimal 16 2D-Texturen gleichzeitig unterstützt werden muss [37], ist es nicht möglich jedes Bild als eine 2D-Textur auf die Grafikkarte zu laden. In manchen Anfragen können deutlich mehr als 16 Bilder pro Region verfügbar sein.

Stattdessen wird ein 2D-Textur-Array verwendet. Wie beim Rendern eines Höhenfelds (siehe 4.3.2) wird zuerst der Vertex-Buffer gebunden, dies jedoch nun für alle Höhenfelder und nicht nur für eins. Es wird ebenso durch die einzelnen Pixel des geoTIFF-Bildes iteriert, um die Höhe zu bestimmen. Die Höhenfelder werden jedoch nicht mehr durch den Befehl *glTexImage2D()* für den Shader lesbar gemacht, sondern durch *glTexSubImage3D()*.

glTexSubImage3D() redefiniert zusammenhängende Teilregionen einer 3D- oder 2D-Array-Textur. Es wird in dem 2D-Array für jedes Bild eine neue Ebene mit den Höhenwerten definiert. Dieses 2D-Array wird über GLSL-Uniforms an den Vertex-Shader übergeben. Zusätzlich zu dem 2D-Array wird nun jedoch auch ein Array, bestehend aus floats, an den Shader übergeben. Dieses enthält die Prozentanteile der Höhenfelder, wie stark sie in die Szene mit eingerechnet werden. Der Prozentanteil wird durch die einzelnen Slider für die Höhenfelder bestimmt (siehe Abbildung 4.2, Bereich 2).

Mit einer for-Schleife wird durch jede Ebene dieses 2D-Array iteriert. Dabei wird die Position (*newPositon*) und der Normalenvektor (*newNormal*) jedes Pixel mit allen vorherigen (*position* und *normal*) mit dem jeweiligen prozentualem Anteil (*opacity [level]*) interpoliert (siehe Listing 4.1). Pixel an denen keine Höhendaten vorhanden sind, werden nicht betrachtet. So wird die Szene nach und nach aufgebaut.

Listing 4.1: Mixen des aktuellen Höhenfelds mit allen vorherigen

```
position = mix(position, newPosition, opacity[level]);
normal = mix(normal, newNormal, opacity[level]);
```

Die Position und der Normalenvektor werden anschließend an den Fragment-Shader weitergegeben, in diesem mussten keine Änderungen vorgenommen werden. In Abbildung 4.4 sind insgesamt 14 gerenderte Höhenfelder zu sehen.

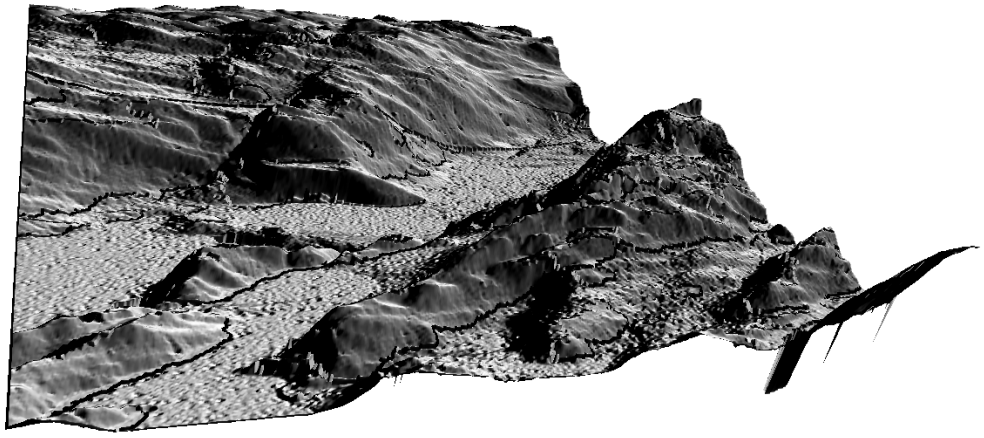


Abbildung 4.4: **In 3D gerenderte Höhenfelder:**
In dieser Abbildung ist ein Ausschnitt des Gebietes rund um den Jakobshavn Gletscher in Grönland zu sehen. Insgesamt sind 14 verschiedene Höhenfelder für die Berechnung dieses Bildes verwendet worden. Die glatten Stellen in Abbildung 4.3 wurden von neueren Bildern überlagert und Datenfehler im vorderen Bereich entfernt.

4.3.4 Entwicklung statischer Visualisierungsmethoden

Der Prototyp ist in der Lage Anfragen an den MapServer zu stellen, anschließend die angefragten Satellitendaten herunterzuladen und die Höhendaten mit unterschiedlichen Gewichtungen zu rendern. Um nun zeitliche Veränderungen im Gelände zu erkennen, sollten statische Visualisierungsmethoden entwickelt werden.

Gitterlinien

Nach den Höhenlinien gelten die Gitterlinien nach Butkiewicz und Stevens [15] als effektivste Visualisierungsmethode zur Change Detection von sich dynamisch verändernden Terrain. Sie sollten als erstes implementiert werden. Hierfür soll alle x Pixel sowohl auf der Länge, als auch auf der Breite eine Linie gezogen werden. X wird dabei vom Nutzer bestimmt und lässt sich über den *SETTINGS*-Button konfigurieren, der Wert wird mittels GLSL-Uniforms an den Fragment-Shader übergeben. Die Berechnung der Linien ist jedoch aufwendiger als vermutet, da keine absoluten Werte für die Linienbreite festgelegt werden können, da es beim Herauszoomen aus der Szene zu Darstellungsfehlern kommt. Diese treten auf weil die Gesamtszene eine Größe von 500 mal 500 Pixel aufweist, die Linienbreite aber in Bildschirmpixel angegeben wird und nicht in der Größe der Szene. Wenn die Szene sehr klein wird, färben die Linien diese komplette schwarz.

Aus diesem Grund muss ein anderer Ansatz gewählt werden. Die Implementierung der Breitenlinien ist in Listing 4.2 angegeben. Zur Berechnung dieser muss die Entfernung (*dis*) zwischen der Position (*position.x*) und dem Linienintervall (*interval*) relativ zum Bildschirm berechnet werden. Die Farbe des Pixels (*pixelColor*) wird anschließend als Farbverlauf zwischen weiß und schwarz bestimmt. Umso näher das Pixel an dem Linienursprung liegt, desto dunkler wird es. Die Berechnungen werden anschließend ebenso für die Längenlinien durchgeführt. Die gerenderten Gitterlinien sind in Abbildung 4.5 zu sehen.

Listing 4.2: Berechnung der Gitterlinien

```
float dif = mod(position.x, interval);
float difInv = dif > h * 0.5 ? h - dif : dif;
float dx = fwidth(vPosition.x);
float dis = clamp(difInv / dx, 0.0, 1.0);
pixelColor = pixelColor - mix(vec3(0.0), vec3(1.0), 1 - dis);
```

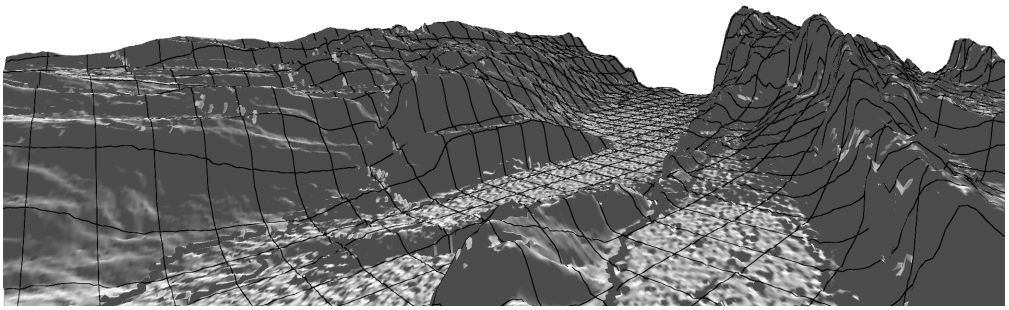


Abbildung 4.5: **Gerendertes Gitternetz:**
In dieser Abbildung ist ein Ausschnitt des Gebietes rund um den Jakobshavn Gletscher in Grönland mit gerenderten Gitterlinien zu sehen. Die Gitterlinien werden alle 15 Pixel gezeichnet.

Höhenlinien

Nach den Gitterlinien sollten die Höhenlinien als statische Visualisierungsmethode implementiert werden. Im Gegensatz zu den Gitterlinien, wird das Intervall der Höhenlinien nicht in Pixeln angegeben sondern Metern. Das Intervall lässt sich ebenfalls nach Klick auf *SETTINGS*-Button konfigurieren. Die Berechnung der Höhenlinien gleicht dabei der Berechnung der Gitterlinien. Statt dem x-Wert der Position wird jedoch der y-Wert (die Höhe) gewählt. Die Höhenlinien sind in Abbildung 4.6 zu sehen.

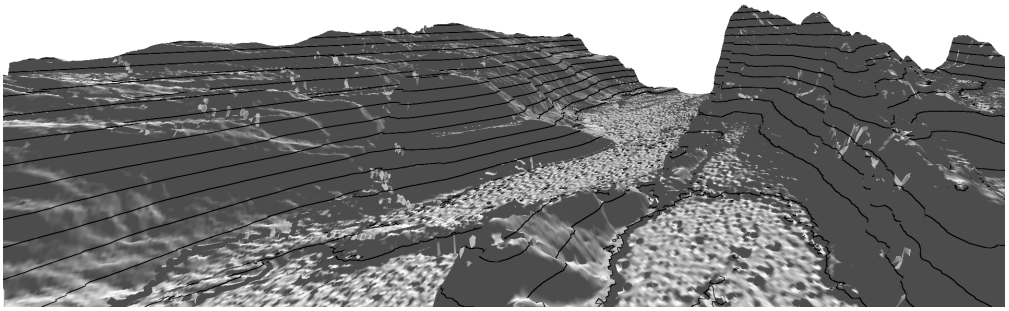


Abbildung 4.6: **Gerenderte Gitterlinien:**
In dieser Abbildung ist ein Ausschnitt des Gebietes rund um den Jakobshavn Gletscher in Grönland mit gerenderten Höhenlinien zu sehen. Die Höhenlinien werden alle 30 m gezeichnet.

Regenbogenskala

Nach den Gitterlinien und den Höhenlinien sollte im nächsten Schritt eine Regenbogenskala implementiert werden. Dabei sollten die einzelnen Höhenebenen jeweils in einer Farbe mit einem durchgehenden Übergang eingefärbt werden. Die Höhen sollen dabei von rot über gelb, grün, türkis, blau nach magenta eingefärbt werden. Dabei sind die tiefsten Punkte rot und die höchsten Punkte magenta gefärbt. Die Berechnungen für einen geradlinigen Farbverlauf sind in Listing 4.3 angegeben. Da die Extremwerte der Höhe des Terrains nicht im Shader berechnet werden können, mussten diese vorher von der CPU berechnet werden und über GLSL-Uniforms an den Fragment-Shader gegeben werden.

ratio ist das Verhältnis zwischen der Höhe des Pixels und der maximalen Höhe des Geländes. Wenn die Höhe im unteren Fünftel der maximalen Gesamthöhe liegt, (*ratio <0.2*), geht die Farbe des Pixels mit steigender Höhe von rot (*vec3(1, 0, 0)*) in orange (*vec3(1, 1, 0*) über. Wenn die Höhe im zweiten Fünftel der maximalen Gesamthöhe liegt, nimmt der rote Farbkanal mit steigender Höhe wieder ab und die Farbe geht von orange in gelb über. Auf diese Art und Weise wird das komplette Farbspektrum des Regenbogens aufgebaut. Eine Ausnahme bildet die Höhe 0, bei dieser wird das Pixel in grau gerendert, um Datenlücken oder die Meeresfläche zu visualisieren.

In der Abbildung 4.7 ist das Terrain in Regenbogenfarben gerendert.

Listing 4.3: Berechnung der Pixelfarbe bei der Regenbogenskala

```
1  if(ratio < 0.2 && ratio != 0) {
2          value = (1.0 / max) * (h * 5);
3          color = vec3(1 , value , 0 );
4  } else if(ratio >= 0.2 && ratio < 0.4) {
5          value = 1 - ((1.0 / max) * ((h - max / 5.0) * 5.0));
6          color = vec3(value, 1 , 0 );
7  } else if(ratio >= 0.4 && ratio < 0.6) {
8          value = (1.0 / max) * ((h - 2 * ( max / 5.0 )) * 5.0);
9          color = vec3(0, 1 , value);
10 } else if(ratio >= 0.6 && ratio < 0.8) {
11         value = 1-((1.0 / max) * (h - 3 * ( max / 5.0 )) * 5.0);
12         color = vec3(0, value , 1);
13 } else if(ratio >= 0.8 ){
14         value = (1.0 / max) * (h - 4 * ( max / 5.0 )) * 5.0;
15         color = vec3(value , 0, 1);
16 } else if (ratio == 0) {
17         color = vec3(0.5);
18 }
```

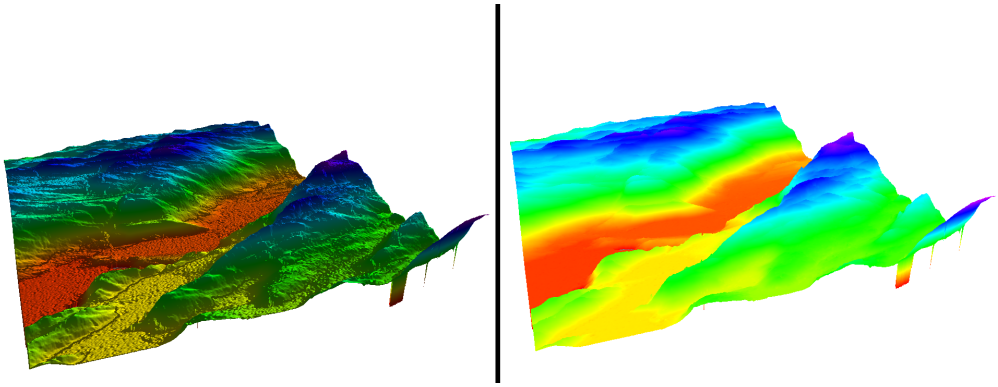


Abbildung 4.7: **Gerenderte Regenbogenskala:**
In dieser Abbildung ist ein Ausschnitt des Gebietes rund um den Jakobshavn Gletscher in Grönland zu sehen. Auf der linken Seite ist das Terrain mit Schattierungen gerendert, auf der rechten Seite ohne.

Differenzkarte und Histogramm

Als nächste Visualisierungsmethode war eine Differenzkarte sowie ein Histogramm der Differenzen angedacht. Mit der Differenzkarte sollten interessante Regionen hervorgehoben werden, dies sind Regionen, in denen besonders viele Änderungen auftreten. Um solche Regionen zu finden, werden die Höhenunterschiede der einzelnen Pixel zwischen aufeinanderfolgen Höhenfeldern addiert. Das Histogramm soll die Häufigkeitsverteilung der addierten Höhendifferenzen visualisieren. Anhand dieser Verteilung soll der Nutzer nachvollziehen können, welche Höhendifferenzen durch fehlerbehaftete Satellitendaten nicht korrekt sind.

Um die Differenzkarte zu erstellen, werden die Höhenunterschiede pixelweise im Vertex-Shader berechnet. Dabei wird zu der gesamten Höhendifferenz die absolute Differenz, zwischen der Höhe des aktuellen Pixels und der Höhe des letzten Pixels, addiert. Die maximale und minimale Höhendifferenz wird vorher auf der CPU berechnet und über Uniforms an den Fragment-Shader übergeben. Sie wird benötigt, um einen Farbverlauf zwischen niedrigen und hohen Differenzen zu erzeugen. Pixel mit wenig Veränderungen werden grün angezeigt, Pixel mit vielen Veränderungen rot.

Um das Histogramm anzuzeigen, wurde ein *HISTOGRAM*-Button in die Benutzeroberfläche integriert. Nach einem Klick auf diesen wird mittels D3.js ein Histogramm der Höhendifferenzen erzeugt. D3.js [38] ist eine JavaScript-Softwarebibliothek zum Erzeugen von dynamischen und interaktiven Datenvisualisierungen. Das findet komplett in der GUI statt, die einzelnen Höhendifferenzen werden, im Gegensatz zur Differenzkarte, auf der CPU berechnet. Über die Methode *CallJavascript()* werden die Differenzen an die Benutzeroberfläche geschickt und dort verarbeitet.

Da in den Datensätzen größere Fehler, als anfangs vermutet, vorhanden waren, musste

dem Histogramm noch eine Funktion zum Ausblenden von Werten hinzugefügt werden. So waren in manchen Pixeln Höhenunterschiede von über 400 m pro Bild vorhanden, obwohl in mehr als 98% der Bildern diese maximal 10 m pro Bild betrugen. Aufgrund dessen waren keine Farbunterschiede zwischen den gültigen Stellen erkennbar. Um minimale und maximale Differenzen festzulegen, wurde ein Slider implementiert. Alle Werte die außerhalb dieses Sliders liegen, werden blau gefärbt. In Abbildung 4.8 ist links das Histogramm und rechts die Differenzkarte abgebildet.

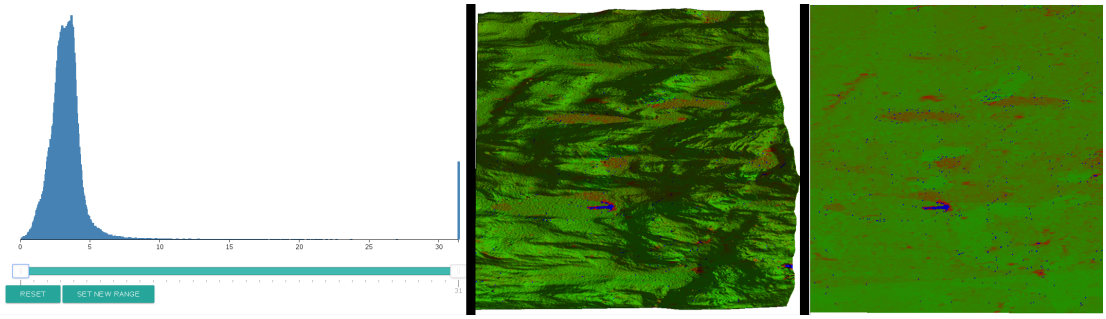


Abbildung 4.8: **Histogramm und Differenzkarte:**
Auf der linken Seite der Abbildung ist das Histogramm der Höhendifferenzen zu sehen, in der Mitte die Differenzkarte mit gerendertem Terrain und auf der rechten Seite die Differenzkarte ohne gerendertem Terrain. Die Erhöhung auf der rechten Seite im Histogramm sind alle Pixel die Höhenunterschiede von mehr als 30 m haben. Die roten Stellen in der Differenzkarte weisen besonders viele Höhenänderungen auf. An den blauen Stellen sind entweder keine Werte vorhanden oder die Höhendifferenzen liegen außerhalb des gewählten Bereiches im Histogramm (0 bis 30 m).

Differenzkarte mit Trends

Mit der entwickelten Differenzkarte war es nun für den Nutzer möglich, interessante Stellen schnell zu finden. Diese Visualisierung zeigt jedoch nur Stellen an, an denen viele Veränderungen auftreten. Damit kann keine Aussage getroffen werden ob das Terrain an dieser Stelle wächst, sinkt oder eventuell sogar stagniert.

Um diese Trends zu erkennen, sollte noch noch eine zweite Differenzkarte implementiert werden. In dieser sollten Erhöhungen des Geländes grün und Senkungen rot gefärbt werden. Umso höher oder tiefer die Veränderungen sind, desto stärker sollten die Pixel in der jeweiligen Farbe eingefärbt werden. Um den Trend der Veränderungen zu erfassen, mussten im Vertex-Shader die einzelnen Höhendifferenzen zwischen den Höhenfeldern addiert werden. Wenn viele Höhendifferenzen auftreten, insgesamt die Höhe des Pixels aber weder stark steigt oder sinkt, sollen die Pixel weiß gefärbt. Wenn wenig Differenzen auftreten sollen die Pixel schwarz gefärbt werden. Die Berechnung der absoluten Höhendifferenzen konnte aus der einfach Differenzkarte übernommen werden.

Die Berechnungen für die endgültige Pixelfarbe finden im Fragment-Shader statt und sind in Listing 4.4 zu sehen. In Zeile 1 wird überprüft, ob die addierten, absoluten Höhenunterschiede (*difAbs*) innerhalb der durch den Slider im Histogramm definierten Grenzen (*difMin* und *difMax*) liegen, wenn nicht, wird das Pixel direkt in blau gerendert. *dif* ist die aufaddierte Höhendifferenz des Pixels, wenn diese positiv ist, werden grün (*vec3(0,1,0)*) und weiß (*vec3(1)*) zur Festlegung der Pixelfarbe linear interpoliert. Umso größer *dif* ist, desto stärker wird der grüne Anteil in der Pixelfarbe. Wenn *dif* negativ ist, werden rot und weiß linear interpoliert. In Zeile 8 wird die erstellte Farbe dann noch einmal mit schwarz interpoliert. Umso geringer die aufaddierten, absoluten Höhenunterschiede zwischen den einzelnen Bilder sind, desto dunkler wird das Pixel.

In Abbildung 4.9 ist die Differenzkarte mit Trends dargestellt.

Listing 4.4: Berechnung der Differenzkarte mit Trends

```
1  if(difAbs > difMin && difAbs < difMax) {
2    float range = difMax - difMin;
3    if(dif > 0) {
4      color = mix(vec3(1), vec3(0,1,0), dif / range);
5    } else {
6      color = mix(vec3(1), vec3(1,0,0), abs(dif) / range);
7    }
8    color = mix(vec3(0), color, difAbs / range);
9  } else {
10   color = vec3(0,0,1);
11 }
```

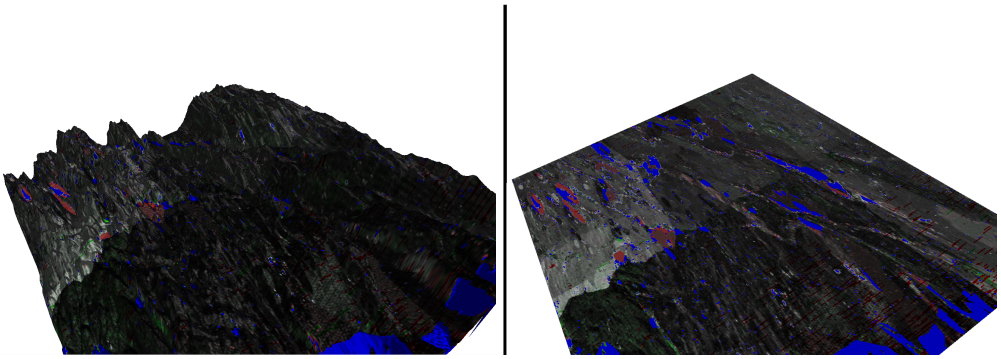


Abbildung 4.9: **Differenzkarte mit Trends:**
In der Abbildung ist die Differenzkarte mit den Trends zu sehen. Auf der linken Seite wurde es mit Terrain gerendert und auf der rechten Seite ohne. Man kann in der Karte erkennen, dass sich das Terrain im linken oberen Bereich deutlich stärker verändert als im Rest des Bildes. Dies wird anhand der helleren Farbtöne sichtbar.

4.3.5 Entwicklung dynamischer Visualisierungsmethoden

Bei der Veränderungserkennung sind besonders die Veränderungen des Höhenfeldes interessant, welches sich aktuell im Fokus befindet. Höhenfelder sind im Fokus, wenn ihr Anteil in der Szene als letztes geändert wurde. Dies ist immer dann der Fall, wenn der Slider des Bildes in der Liste oder der Hauptslider bewegt wird. Wenn der Hauptslider bewegt wird, ist das Bild an der eingestellten Sliderposition im Fokus. In dieser Visualisierungsmethode sollen die Veränderungen vom Höhenfeld, welches sich aktuell im Fokus befindet, zum Rest der Szene verdeutlicht werden.

Dabei werden alle Pixel eingefärbt, bei denen sich die Höhe des Bildes im Fokus um mehr als einen bestimmten Wert, im Vergleich zum Rest der Szene, ändert. Dieser Wert wird vom Nutzer in den Einstellungen der Anwendung festgelegt. Falls das Pixel höher liegt wird es grün gefärbt, wenn es tiefer liegt rot.

Die Visualisierungsmethode ist in Abbildung 4.10 zu sehen.

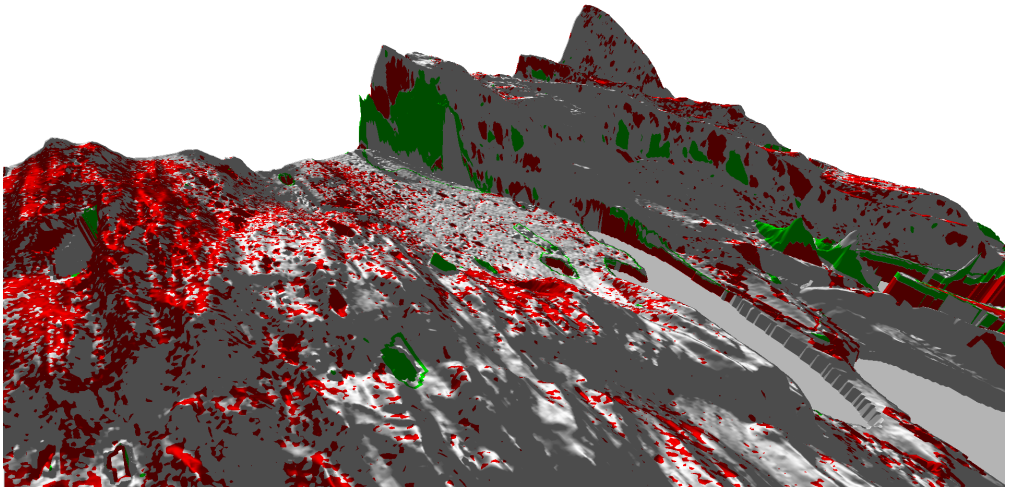


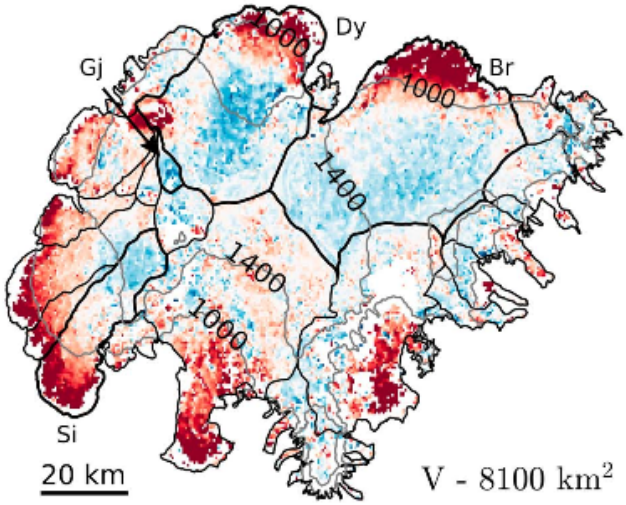
Abbildung 4.10: **Höhenveränderungen des Bildes im Fokus:**
In dieser Abbildung ist ein Ausschnitt des Gebietes rund um den Jakobshavn Gletscher in Grönland zu sehen. Die rot markierten Bereichen verlieren an Höhe, während die grün markierten Bereiche an Höhe gewinnen.

4.4 Evaluierung der Ergebnisse

Der Prototyp war nach der fertigen Implementierung in der Lage geoTIFF-Bilder vom MapServer zu laden und die Höhendaten dieser in 3D zu rendern. Um das Entdecken von Veränderungen im Terrain für den Nutzer zu erleichtern, wurden statische und eine dynamische Visualisierungsmethoden entwickelt. Im letzten Schritt der Entwicklung sollte

Prototyp getestet werden.

Um den Prototyp zu testen, wurde eine Region auf der Erde ausgewählt, in der bekannte Höhenveränderungen vorkommen. Hierfür wurde das Gebiet Dyngjujökull ausgewählt. Dyngjujökull ist Teil des Vatnajökull, dem größten Gletscher Islands. Diese Region wurde ausgewählt, da in dieser Region die Höhenunterschiede von L. Foresta, N. Gourmelen, F. Pálsson, P. Nienow, H. Björnsson und A. Shepherd mit Daten des CryoSat-2 ermittelt wurden [39]. CryoSat-2 ist ein Forschungssatellit der ESA zur Beobachtung der Eismassen auf der Erde. Die Höhenveränderung in Dyngjujökull zwischen 2010 und 2015 ist in Abbildung 4.11 zu sehen.



Quelle: Surface elevation change and mass balance of Icelandic ice caps derived from swath mode CryoSat-2 altimetry [39]

Abbildung 4.11: **Höhenunterschiede am Vatnajökull:**
In der Abbildung sind die Höhenunterschiede am Vatnajökull zwischen 2010 und 2015 zu sehen. Die roten Stellen sind Höhenverringerungen um bis zu 3 m pro Jahr und die blauen Erhöhungen um bis zu 3 m pro Jahr. Zur Evaluierung wurde die Region Dyngjujökull (Dy) ausgewählt. In Dyngjujökull in der Höhenregion um 1000 m schmilzt der Gletscher, in höheren Regionen wächst der Gletscher.

Auffällig in Dyngjujökull ist, dass in der Region um 1000 m der Gletscher schmilzt und an Höhe verliert. In den höheren Regionen kommt es zu einer Gletscherverdickung, hier steigt die Höhe jedes Jahr um bis zu 3 m an. Diese Gletscherverdickungen sollten mit dem Prototypen entdeckt werden. Hierfür wurde eine Teilregion (64.775°N , -17.031°W / 64.878°N, -16.816°W) des Gletschers angefragt. Im ersten Schritt wurden mittels der Differenzkarte mit Trends geschaut, in welcher Region welche Veränderungen vorliegen. Mit dieser Visualisierungsmethode wurden in der höheren Region des Ausschnittes positive

Geländeveränderungen gefunden. Die Differenzkarte ist in Abbildung 4.12 zu sehen.

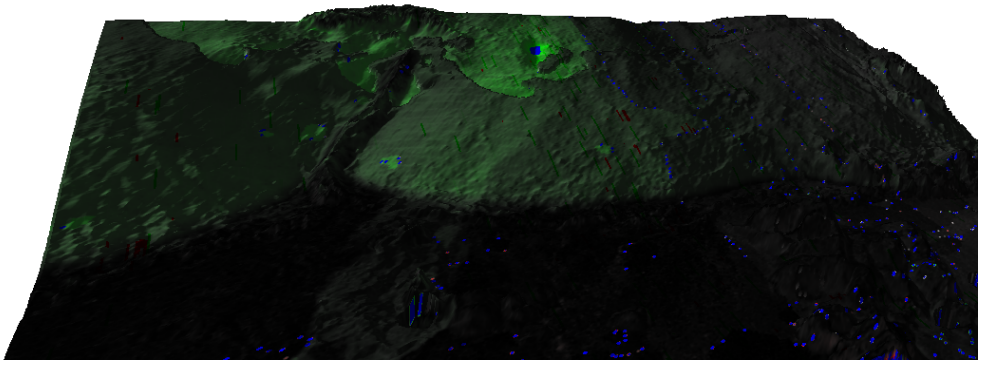


Abbildung 4.12: **Differenzkarte am Dyngjujökull:**
In der Differenzkarte werden Gletschererhöhungen verdeutlicht, diese sind grün eingefärbt. Die Veränderungen sind jedoch nur relativ gering, aus diesem Grund ist das grün sehr dunkel gehalten. In der flachen Region vor der Kante finden kaum Änderungen statt.

Im nächsten Schritt sollten diese Erhöhungen mittels der anderen Visualisierungsmethoden gefunden werden. Zuerst sollten die Erhöhungen mit den Höhenlinien gefunden werden. Die Veränderungen finden dabei im Zeitraum zwischen Februar 2013 und November 2015 statt. In der Abbildung 4.13 ist zu erkennen wie die Höhenlinien nach vorne „wandern“, das Terrain wächst also. Zur Verdeutlichung wurde die erste Höhenlinie des Terrains im Nachhinein rot markiert. Der komplette Gletscher wächst, da die Linien sich im ungefähr gleichen Abstand wie zur roten Vergleichslinie mit nach vorne bewegen. Mit Bildern ist dies sehr schwer zu beschreiben, die Bewegung der Höhenlinien ist in der interaktiven Visualisierung sehr gut zu erkennen.

Im nächsten Schritt sollte die Höhenveränderung mit Hilfe der Regenbogenskala erkannt werden. Hierbei wurde jedoch festgestellt, dass aufgrund der geringen absoluten Höhenveränderung (maximal 9 m Wachstum in drei Jahre, siehe 4.11) keine signifikanten Farbunterschiede festzustellen waren.

Als vielversprechend galt die dynamische Visualisierungsmethode, mit dieser sollte die Gletscherverdickung besonders gut visualisiert werden. Die Ergebnisse dieser Visualisierungsmethode sind in Abbildung 4.14 zu sehen.

Mit der dynamischen Visualisierungsmethode wird die Gletschererhöhung sehr gut sichtbar. Die drei Bilder zeigen jeweils Erhöhungen die mindestens 5 m, 10 m und 15 m betragen (von oben nach unten). Anhand der eingefärbten Flächen an der steilen Kante

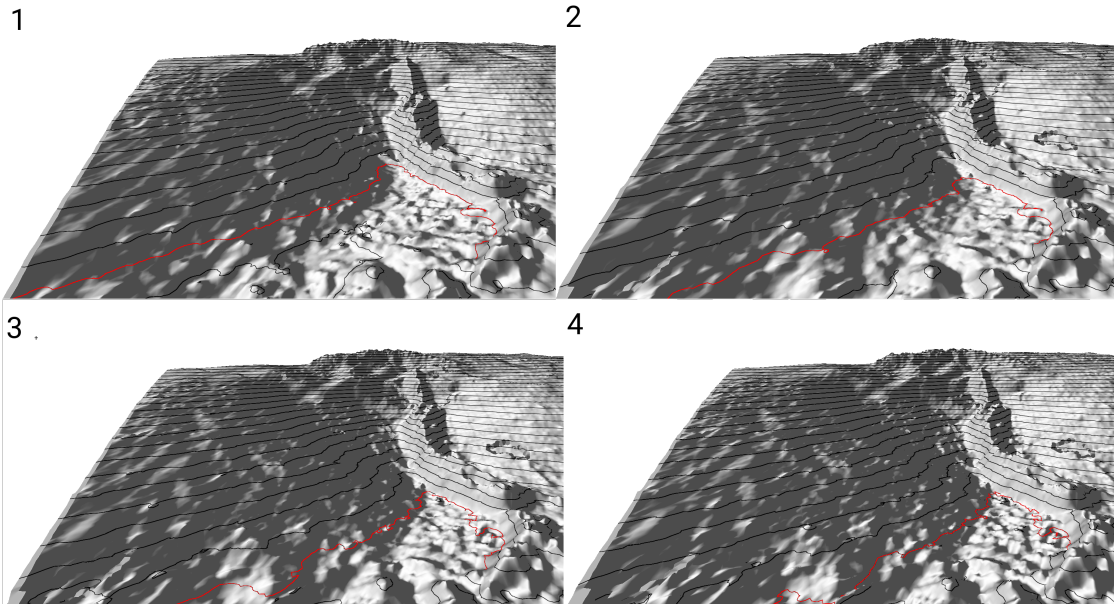


Abbildung 4.13: **Höhenlinien am Dyngjujökull:**
In dieser Abbildung wurde die Höhenveränderung am Dyngjujökull mittels Höhenlinien erkannt. Die Höhenlinien haben einen Abstand von jeweils 10 m. Um das Wachstum der Höhe zu verdeutlichen, wurde eine Höhenlinie rot markiert.

kann erkannt werden das der Gletscher „wandert“. Im Datensatz sind auch Erhöhungen von über 10 m innerhalb der 2,75 Jahre (Februar 2013 bis November 2015) vorhanden (Abbildung 4.14, Bild 2). Diese dürften jedoch maximal ca. 9 m betragen (siehe Abbildung 4.11, [39]). Es ist also davon auszugehen, dass der ArticDEM-Datensatz ungenauer ist als, die in 2.2.1 angegeben, 0,5 m.

Aufgrund des Datensatzes war es leider nicht möglich andere Gletscher aus dem Paper zu untersuchen. Hierfür gab es vier Gründe:

- **Zu wenig Bilder:** Für die angefragte Region waren zu wenig Bilder vorhanden. Die Gletscher waren teilweise nur von einem oder zwei Satellitenaufnahmen abgedeckt. Stellenweise gab es auch größere Lücken in dem Datensatz.
- **Ungünstige Aufnahmedaten:** Häufig kam es dazu, dass die Bilder innerhalb eines kurzen Zeitraumes aufgenommen wurden. Innerhalb der kurzen Zeit kam es zu keinen signifikanten Veränderungen im Terrain. Außerdem lagen manche Aufnahmedaten außerhalb dem Zeitraum der Studie. Diese betrachtete nur Höhenunterschiede zwischen Oktober 2010 und September 2015.
- **Fehlerbehaftete Aufnahmen:** In einigen Gebieten Islands sind große Fehler

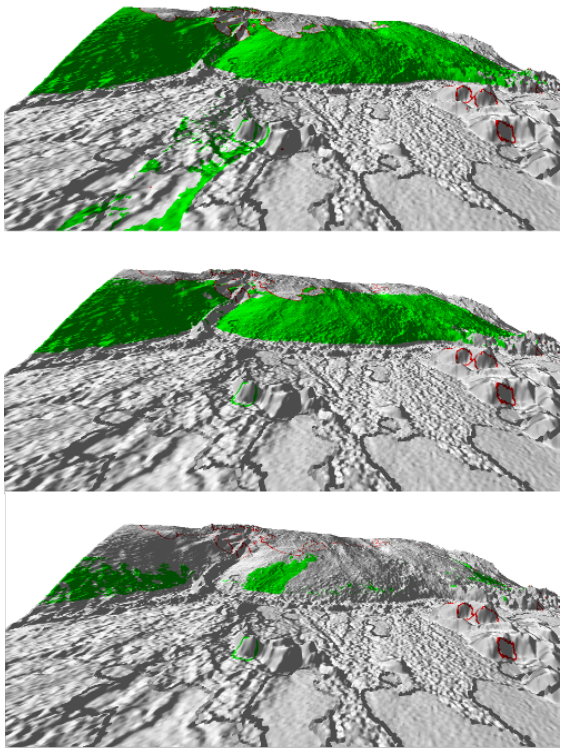


Abbildung 4.14: **Gletschererhöhungen am Dyngjujökull:**
In der Abbildung sind die Gletschererhöhungen am Dyngjujökull zu sehen. In der Abbildung ist das Vorstoßen des Gletschers in grün dargestellt. Die Höhenunterschiede fanden zwischen Februar 2013 und November 2015 statt. Im ersten Bild sind alle Erhöhungen ab 5 m zu sehen, im zweiten ab 10 m und im dritten ab 15 m.

innerhalb des Datensatzes vorhanden. Das Gelände veränderte sich hier innerhalb weniger Monate um mehrere hundert Meter, sowohl in die positive, als auch in die negative Richtung. Mit diesen Daten konnte keine qualitative Aussage über die Geländeveränderung getroffen werden.

- **Niederschlag:** Während in der Studie Niederschlag in Form von Schnee über die durchschnittliche Höhe herausgerechnet wurde, konnte das im ArticDEM-Datensatz nicht vorgenommen werden, da zu wenig Bilder vorhanden waren. So werden die Höhen im ArticDEM-Datensatz durch das Fallen und Abschmelzen des Schnees verfälscht.

5 Ergebnisse und Ausblick

5.1 Ergebnisse

Abschließend betrachtet war die Bachelorarbeit erfolgreich. Während der zwölf Wochen Bearbeitungszeit sollten insgesamt zwei Prototypen implementiert werden, zur Change Detection in Satellitenbildern und in Höhendaten.

Mit dem Prototypen zur Change Detection in Satellitenbildern wurde eine HTML und JavaScript Anwendung geschaffen, mit welcher Satellitenbilder vom MapServer angefragt werden können. Diese konnten anschließend mit unterschiedlicher Transparenz in die Gesamtszene eingehen. Mit dem Prototypen konnten erfolgreich Veränderungen in Form von Sandstürmen in den Satellitenbildern nachgewiesen werden. Der Nachweis von Geländeveränderungen im HRSC-Datensatz konnte aufgrund der relativ geringen Auflösung und dem kurzen Zeitraum der Aufnahmen nicht vollbracht werden. Als Einstieg in die Change Detection und die MapServer-Kommunikation, erwies sich der Prototyp als sehr hilfreich.

Der Prototyp zur Change Detection in Höhendaten ist eine C++ Anwendung mit einer in HTML und JavaScript programmierten Benutzeroberfläche. Die Anwendung fragt über den MapServer die Höhenfelder von Satellitenbildern an und stellt diese interaktiv in 3D dar. Die einzelnen Höhenfelder gehen dabei mit einer durch den Nutzer definierten Gewichtung ein. Um den Nutzer bei der Erkennung von zeitlichen Terrainänderungen zu unterstützen, wurden verschiedene statische und eine dynamische Visualisierungsmethode implementiert. Mit diesen Methoden konnten Verdickungen des Dyngjujökull-Gletschers in Island nachgewiesen werden. Die Veränderungen konnten sowohl mit den statischen als auch mit der dynamischen Visualisierungsmethode gezeigt werden. Die Höhenentwicklung anderer isländischer Gletscher konnte aufgrund von fehlerbehafteten oder unvollständigen Datensätzen nicht mit dem Prototypen bewiesen werden.

5.2 Ausblick

Da der Prototyp zur Change Detection in Satellitenbilder nur als Einstieg in die Change Detection und MapServer-Kommunikation gedacht war, ist in Zukunft nur eine weitere Evaluierung vorstellbar. Die Anwendung kann noch mit weiteren Datensätzen getestet werden.

Im Prototypen zur Change Detection in Höhendaten sind deutlich mehr Möglichkeiten

der Weiterentwicklung denkbar. Diese teilen sich grundlegend in die Bereiche VR-Integration, Weiterentwicklung der Visualisierungsmethoden und Evaluierung auf.

Die Visualisierung der Höhenfelder in VR ist eine Möglichkeit der Weiterentwicklung, die den Nutzer noch interaktiver mit der Anwendung umgehen lässt. Dafür kann die Anwendung mit der Terrain-Rendering Software verknüpft werden. Der Nutzer könnte dabei einen Bereich des Planeten markieren. Die Anwendung zur Change Detection soll sich anschließend im Vordergrund öffnen und die Change Detection ermöglichen.

Um Höhenveränderungen effektiver zu erkennen, können weitere Visualisierungsmethoden implementiert werden. Hierbei ist der Fokus besonders auf zusätzliche dynamische Methoden gerichtet. Für die vorhandenen Visualisierungsmethoden sind weitere Konfigurationsmöglichkeiten denkbar, in den Höhenlinien können zum Beispiel Beschriftungen für die absolute Höhe hinzugefügt werden.

Im Bereich der Evaluierung können verschiedene Datensätze in Verbindung mit der Anwendung getestet werden, um qualifiziertere Aussagen über die Unterstützung des Nutzers bei der Erkennung von Höhenveränderungen treffen zu können. Derzeit ist es auch nicht möglich die Visualisierungsmethoden untereinander qualitativ zu vergleichen. Um darüber Aussagen treffen zu können, sind Nutzerstudien denkbar um festzustellen welche Methode zeitliche Veränderungen in Satellitenbilder am besten visualisiert.

Literatur

[1] *DLR im Überblick*. URL: http://www.dlr.de/dlr/de/desktopdefault.aspx/tabid-10002/#/DLR/Start/About (besucht am 12. 07. 2017).

[2] *Simulations- und Softwaretechnik*. URL: http://www.dlr.de/sc/ (besucht am 12. 07. 2017).

[3] *Wir über uns*. URL: http://www.dhbw.de/die-dhbw/wir-ueber-uns.html (besucht am 12. 07. 2017).

[4] Daniel Keim u. a. *Mastering the Information Age Solving Problems with Visual Analytics*. 2010. ISBN: 978-3-905673-77-7.

[5] *What is Visual Analytics?* URL: http://www.visual-analytics.eu/faq/ (besucht am 12. 07. 2017).

[6] Ronald A Rensink. *Change Detection*. 2002.

[7] *Landsat Mission*. URL: https://landsat.usgs.gov/ (besucht am 19. 07. 2017).

[8] Pol R Copp1n und Marvin E Bauer. "Digital Change Detection in Forest Ecosystems with Remote Sensing Imagery". In: *Remote Sensing Reviews* 13 (), S. 207–234.

[9] "Urban built-up land change detection with road density and spectral information from multi-temporal Landsat TM data". In: (2017). DOI: 10.1080/01431160110104728. URL: http://www.tandfonline.com/action/journalInformation?journalCode=tres20.

[10] Tobias Bolch, Brian Menounos und Roger Wheate. "Landsat-based inventory of glaciers in western Canada, 1985–2005". In: *Remote Sensing of Environment* 114.1 (Jan. 2010), S. 127–137. ISSN: 00344257. DOI: 10.1016/j.rse.2009.08.015. URL: http://linkinghub.elsevier.com/retrieve/pii/S0034425709002661.

[11] Andrew A. Tronin, Masashi Hayakawa und Oleg A. Molchanov. "Thermal IR satellite data application for earthquake research in Japan and China". In: *Journal of Geodynamics* 33.4-5 (Mai 2002), S. 519–534. ISSN: 02643707. DOI: 10.1016/S0264-3707(02)00013-3. URL: http://linkinghub.elsevier.com/retrieve/pii/S0264370702000133.

[12] *TerraSAR-X*. URL: http://www.dlr.de/dlr/desktopdefault.aspx/tabid-10377/565_read-436/#/gallery/350 (besucht am 21. 07. 2017).

[13] Stephan Brusch u. a. “Ship surveillance with TerraSAR-X”. In: *IEEE Transactions on Geoscience and Remote Sensing* (2011). ISSN: 01962892. DOI: 10.1109/TGRS.2010.2071879.

[14] Laura Giustarini u. a. “A Change Detection Approach to Flood Mapping in Urban Areas Using TerraSAR-X”. In: *IEEE TRANSACTIONS ON GEOSCIENCE AND REMOTE SENSING* 51.4 (2013). DOI: 10.1109/TGRS.2012.2210901.

[15] Thomas Butkiewicz und Andrew H. Stevens. “Effectiveness of Structured Textures on Dynamically Changing Terrain-like Surfaces”. In: *IEEE Transactions on Visualization and Computer Graphics* (2016). ISSN: 10772626. DOI: 10.1109/TVCG.2015.2467962.

[16] Colin Ware und Graeme Sweet. “View Direction, Surface Orientation and Texture Orientation for Perception of Surface Shape”. In: (2004). URL: http://scholars.unh.edu/ccom%20http://scholars.unh.edu/ccom/325.

[17] Sunghee Kim, Haleh Hagh-Shenas und Victoria Interrante. “Showing shape with texture – two directions seem better than one”. In: *Electronic Imaging* (2003). DOI: 10.1117/12.477368.

[18] *ArcticDEM*. URL: https://www.pgc.umn.edu/data/arcticdem/ (besucht am 24. 08. 2017).

[19] *HRSC*. URL: http://www.dlr.de/dlr/desktopdefault.aspx/tabid-10364/548_read-400/#/gallery/657 (besucht am 24. 08. 2017).

[20] *NSF*. URL: https://www.nsf.gov/about/ (besucht am 24. 08. 2017).

[21] *NGA*. URL: https://www.nga.mil/About/Pages/Default.aspx (besucht am 24. 08. 2017).

[22] *DigitalGlobe*. URL: https://www.digitalglobe.com/about/our-constellation (besucht am 24. 08. 2017).

[23] *Mars Express*. URL: http://www.esa.int/Our_Activities/Space_Science/Mars_Express/The_mission (besucht am 24. 08. 2017).

[24] *MapServer-About*. URL: http://mapserver.org/de/about.html (besucht am 04. 08. 2017).

[25] K M Górski u. a. “HEALPix — a Framework for High Resolution Discretization, and Fast Analysis of Data Distributed on the Sphere”. In: (2004). arXiv: 0409513v1 [astro-ph].

[26] *GDAL*. URL: http://www.gdal.org/ (besucht am 06. 09. 2017).

[27] *C++ Geschichte*. URL: http://www.cplusplus.com/info/history/ (besucht am 14. 08. 2017).

[28] *Tiobe Index*. URL: https://www.tiobe.com/tiobe-index/ (besucht am 14. 08. 2017).

[29] *C++ Beschreibung*. URL: http://www.cplusplus.com/info/description/ (besucht am 14. 08. 2017).

[30] *OpenGL About*. URL: https://www.opengl.org/about/ (besucht am 14. 08. 2017).

[31] *OpenGL Extensions*. URL: https://www.khronos.org/opengl/wiki/OpenGL_Extension (besucht am 14. 08. 2017).

[32] *noUiSlider*. URL: https://refreshless.com/nouislider/ (besucht am 08. 08. 2017).

[33] *MaterializeCss*. URL: http://materializecss.com/about.html (besucht am 08. 08. 2017).

[34] *Dust lifting over Arcadia Planitia, Mars*. URL: http://www.planetary.org/blogs/guest-blogs/2016/1104-capturing-martian-weather-in-motion.html (besucht am 05. 09. 2017).

[35] *cURLpp*. URL: http://www.curlpp.org/ (besucht am 06. 09. 2017).

[36] *libcurl*. URL: https://curl.haxx.se/libcurl/ (besucht am 06. 09. 2017).

[37] *OpengGL Minimum unterstützte 2D-Texturen*. URL: https://www.khronos.org/opengl/wiki/Shader#Resource_limitations (besucht am 05. 09. 2017).

[38] *D3.js*. URL: https://d3js.org/ (besucht am 31. 08. 2017).

[39] L. Foresta u. a. "Surface elevation change and mass balance of Icelandic ice caps derived from swath mode CryoSat-2 altimetry". In: *Geophysical Research Letters* (2016). ISSN: 19448007. DOI: 10.1002/2016GL071485.