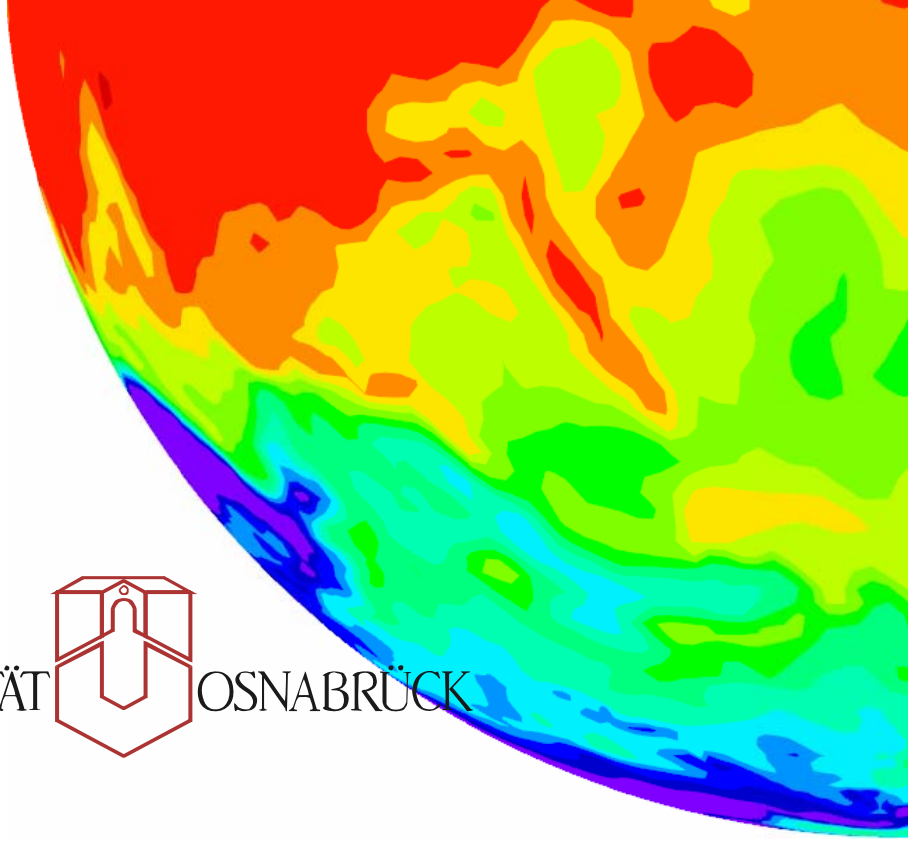
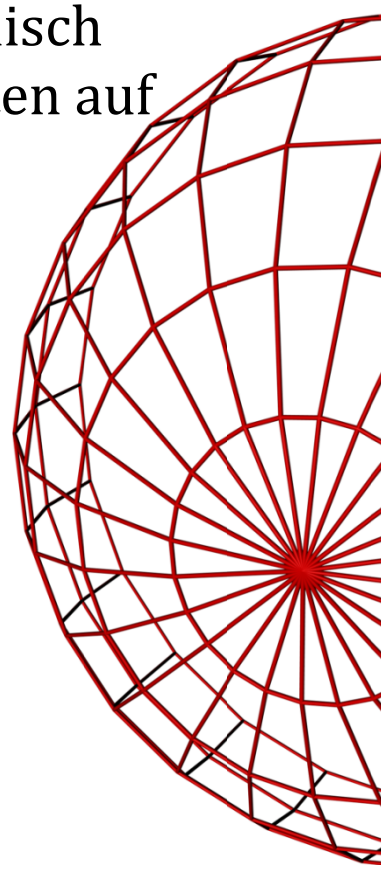


UNIVERSITÄT OSNABRÜCK

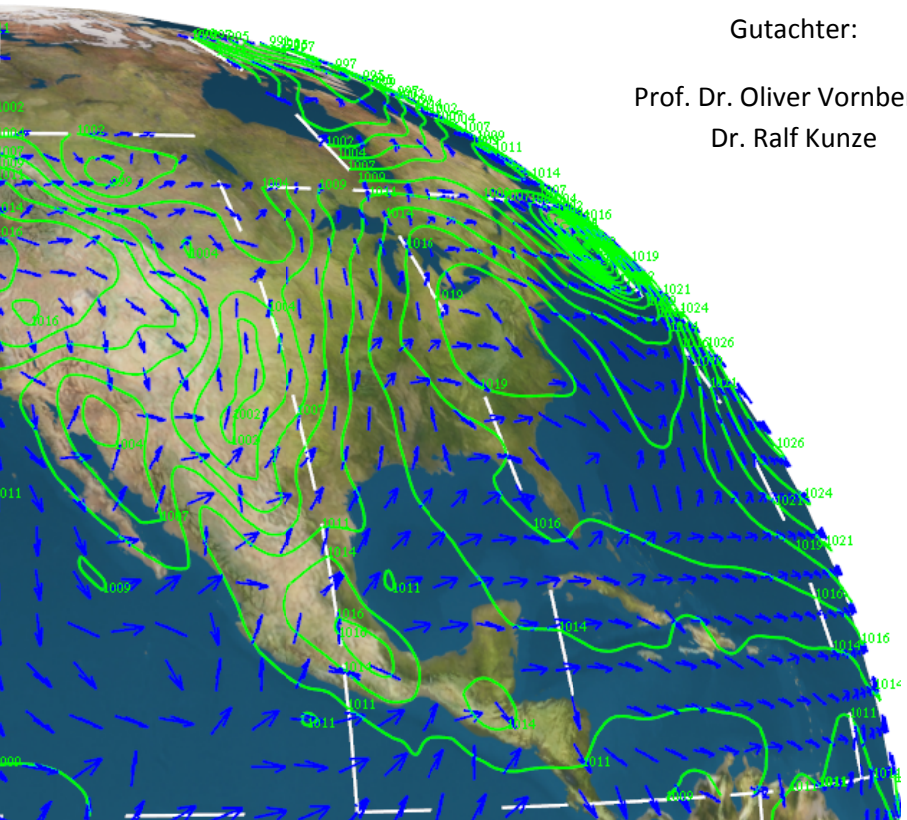
Earth Weather 3D

Visualisierung und Animation dynamisch bestimmter Teilmengen von Wetterdaten auf Webseiten mit OpenGL



Masterarbeit von Henning Wenke

Juni 2007



Gutachter:

Prof. Dr. Oliver Vornberger
Dr. Ralf Kunze

Zusammenfassung

Gegenstand dieser Masterarbeit ist ein Projekt zur Visualisierung und Animation von orts- und zeitabhängigen Wetterdaten mit Grafikhardwareunterstützung im Web. Bislang existiert kein vergleichbarer Ansatz, der die grundlegenden Funktionalitäten von Google Earth bietet (Wahl der Kameraposition, selektives Laden der Daten je Bildausschnitt und Zoomstufe) und mit Animationen über die Zeit kombiniert. Daher wurde im Rahmen dieser Untersuchung ein Java Applet entwickelt, mit dem solche Daten interaktiv und dreidimensional animiert im Internet dargestellt werden können. Zur Visualisierung der einzelnen Wetterelemente kommen verschiedene Techniken wie Farbverläufe oder Isolinien zum Einsatz. In diesem Zusammenhang ist insbesondere auf eine neu entwickelte Erweiterung des Marching Square Algorithmus hinzuweisen, welche in linearer Laufzeit aus den durch den Marching Square Algorithmus erzeugten Liniensegmenten echte Polygone erzeugen kann.

Abstract

This master's thesis deals with a project for webbased visualization and animation of time and space depending weather data. Therefore a program delivering the basic functionality of Google Earth (free camera position choice, selective data loading for current zoom and camera position) has been developed. Additionally, it supports animation and can be embedded in a website due to the platform independent implementation as a java applet. A spatial and interactive 3 day weather forecast and is available at:

http://www.henning-wenke.de/

The applet utilizes the OpenGL API via JOGL for efficient 3D-Graphics rendering. According to the different data types, a variety of visualization techniques (e.g. isolines for air-pressure and color gradient for temperature) is permitted. Especially noteworthy is a newly developed marching square algorithm modification, which delivers real polygons instead of line-segments, offering the possibility of smoothing these e.g. with a subdivision-curve-technique. The algorithms complexity is linear with respect to the data resolution, thus it has minimal impact on the runtime behavior and can be used in this real time environment.

1 Danksagung

Ich bedanke mich hiermit bei allen Personen, die mich während der Masterarbeit unterstützt haben. Ganz besonderer Dank gilt:

- Herrn Prof. Dr. Oliver Vornberger für die sehr gute Betreuung und die Unterstützung auch über die Masterarbeit hinaus
- Herrn Dr. Ralf Kunze für die sehr gute Betreuung, die vielen konstruktiven Vorschläge und Anmerkungen sowie für das Korrekturlesen
- Friedhelm Hofmeyer für das Einrichten der Datenbank
- Meiner Familie für die Unterstützung und für das Korrekturlesen

Osnabrück, im Juni 2007

Henning Wenke

Inhaltsverzeichnis

2 Einleitung

2.1 Motivation

In der heutigen Zeit gewinnt die Präsentation von Produkten, Ideen aber auch von wissenschaftlichen Ergebnissen zusehends an Bedeutung. Insbesondere gilt dies für Forschungszweige, in denen es große Datenmengen zu überblicken gilt. Hier können durch grafisches Darstellen der Messwerte selbst kleine Besonderheiten oder auch Fehler unmittelbar erkannt werden. Außerdem kann auf diese Weise auch ein Laie in einem Maße Einblick in die aktuelle Forschung nehmen, der ihm sonst aufgrund der Komplexität der Materie verwehrt bliebe. Wer kennt nicht beispielsweise Google Earth [Google], das Programm, mit dem Sattelitenbilder der Erdoberfläche im Internet auf eindrucksvolle Weise angezeigt werden können? Es ist dabei sowohl die Betrachtung der gesamten Erde als auch von Ausschnitten aus einem beliebigen Winkel möglich. Bewegt sich die Kamera näher an die Erde, wird diese mit immer detaillierteren Oberflächentexturen versehen, welche dynamisch über das Netz nachgeladen werden. Bislang wurde allerdings bei Google Earth die Veranschaulichung der zeitlichen Änderung der Daten, die gerade beim Klima von höchster Bedeutung ist, vernachlässigt. Dies bleibt weiterhin dem Fernsehwetterbericht und einigen Webseiten mit zumeist gering aufgelösten Filmchen wie beispielsweise animierten GIFs vorenthalten, bei denen jedoch keinerlei Interaktion möglich ist. Daher soll die vorliegende Arbeit einen Weg aufzeigen, Klima- und Wetterdaten angemessen der breiten Öffentlichkeit im Internet präsentieren zu können.

2.2 Eigene Vorarbeit

Für meine im März 2005 fertiggestellte Bachelorarbeit [Wenke] ist ein Programm entstanden, welches orts- und zeitabhängige Klimasimulationsdaten (Temperatur, Bewölkungsdichte und Wind) in Echtzeit einlesen und auf eine Erdkugel projiziert darstellen kann. Das Programm ist in C++ geschrieben und verwendet die OpenGL API [OGL] zum effizienten Rendern der 3D-Grafik. Damit ist die Applikation in der Lage, Klimadaten aus lokal vorliegenden Dateien durch eine optisch ansprechende Animation anzuzeigen.

2.3 Ziele der Arbeit

Google Earth stand zusammen mit meiner Bachelorarbeit Pate für ein im Rahmen dieser Masterarbeit entstandenes Programm, welches wie Google Earth die gewünschten Daten über das Internet bezieht und zusätzlich um die zeitliche Dimension erweitert wurde. Bedingt durch die im Vergleich zu einer lokalen Festplatte geringe Datenübertragungsrate aktueller DSL-Internetanschlüsse ergibt sich die Minimierung der zu übertragenden Datenmenge als zentrales Ziel der Arbeit. Sinnvoll ist in diesem Zusammenhang vor allem ein Datenlademechanismus, der ausschließlich die u.a. in Abhängigkeit des Sichtfelds benötigte Teilmenge der Wetterdaten anfordert und nicht etwa sequentiell aus einer Datei auslesen muss. Außerdem soll das Programm plattformunabhängig sein und in eine Webseite eingebettet werden können, weshalb C++ als Programmiersprache ausscheidet. Stattdessen wird ein von Grund auf neu entwickeltes Java-Applet, das mithilfe von JOGL [JOGL] die OpenGL API nutzt, eingesetzt. Abbildung 1 zeigt das in einem Browser laufende Applet. Zweiter zentraler Punkt ist die Entwicklung fortschrittlicherer Visualisierungsalgorithmen zur Verbesserung der Darstellungsqualität insbesondere bei geringer aufgelösten Daten und zum deutlicheren Hervorheben der jeweiligen datenspezifischen Merkmale. Da die verschiedenen Klima- und Wetterelemente

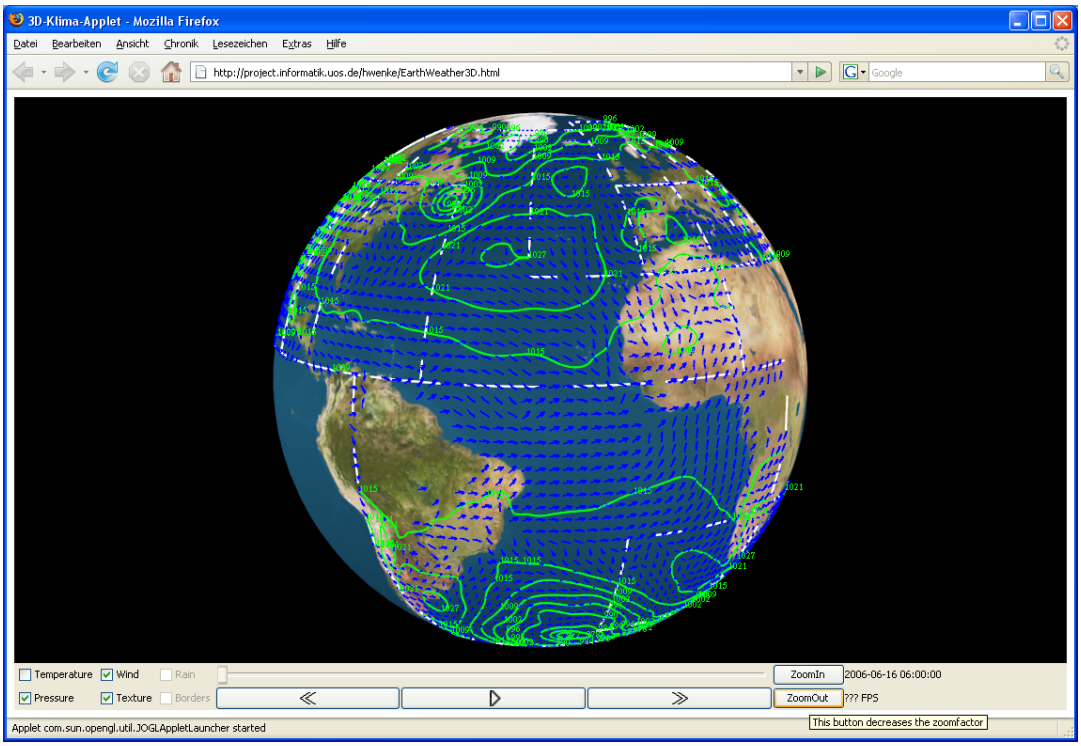


Abbildung 1: Anzeige des Applets im Firefox

sich wechselseitig beeinflussen, müssen sie zum vollständigen Verständnis wahlweise auch kombiniert dargestellt werden können. Daher wurden für die unterschiedlichen Wetterdaten jeweils geeignete Visualisierungstechniken wie z.B. Farbverläufe für Temperaturen und Isolinien für Luftdruck gewählt. Das Applet liefert eine dreitägige Wettervorhersage und ist zu erreichen unter:

http://www.henning-wenke.de/

3 Werkzeuge

In diesem Kapitel wird ein kurzer Einblick in Technologien gegeben, welche zur Umsetzung des Projekts unverzichtbar sind.

3.1 OpenGL (Open Graphics Library)

OpenGL stellt eine sehr schlanke API zum Rendern von 2D und 3D Computergrafik in Applikationen dar, die im Gegensatz zu Microsofts DirectX für viele verschiedene Plattformen verfügbar ist. Dabei soll der Zugriff auf die Grafikhardware auf der geringstmöglichen Ebene erfolgen, welche noch Hardwareunabhängigkeit gewährleistet. OpenGL unterstützt einen variablen Grad der Hardwarebeschleunigung, der sich nach dem Funktionsumfang der jeweiligen Implementation der API richtet. Diese wird in der Regel als Teil der Grafikkarten-Treiber ausgeliefert. Aktuelle Grafikhardware kann vom Rastern der Primitive bis hin zur Berechnung und Transformation der Geometrie einen Großteil der Viewing Pipeline ausführen. Auf der Grafikkarte nicht vorhandene Funktionen müssen allerdings unter deutlichen Performanceeinbußen durch die CPU emuliert werden. Die Entwicklung von OpenGL begann 1992 durch Silicon Graphics Inc. (SGI), wird aber seit dem 31. Juli 2006 von der Khronos Group weitergeführt. Die aktuelle Version von OpenGL ist 2.1 und wurde 2. August 2006 herausgegeben. [Shreiner] [Davis] [Rost] [OGL]

3.1.1 Bibliotheken

Die OpenGL API gliedert sich in drei Bibliotheken, welche im Folgenden erläutert werden:

Core OpenGL Library (`gl`) enthält ca. 200 Befehle zum Verwalten und Rendern von elementaren geometrischen Primitiven wie Punkten, Linien, Polygonen und Bézierkurven. Diese Basisbefehle haben einen sehr geringen Abstraktionsgrad und sind demnach geeignet, mit Hardwarebeschleunigung implementiert zu werden.

OpenGL Utility Library (`glu`) enthält ca. 50 Befehle, welche die weniger abstrakten Kommandos der Core OpenGL Library nutzen, um komplexere Operationen auszuführen. Darunter fallen unter anderem das vereinfachte Manipulieren der Projektionsmatrizen, das Rendern komplexerer Oberflächen (etwa NURBS), Mipmapping und die Tesselierung von Polygonen. Das OpenGL Utility Toolkit ist üblicherweise ebenfalls im Standard OpenGL Paket enthalten.

OpenGL Utility Toolkit (`glut`) enthält ca. 30 Befehle und setzt auf OpenGL, GLU und betriebssystemspezifischen Funktionen auf, um eine einfache Anbindung der von OpenGL gerenderten Ausgabe an das jeweilige Fenstersystem zu ermöglichen. Außerdem dient es der Verarbeitung von Benutzereingaben und enthält einige weitere Primitive. Das OpenGL Utility Toolkit gehört allerdings nicht zum Standard OpenGL Paket und wird nicht mehr weiterentwickelt. Ältere Versionen sind trotzdem für die meisten Plattformen verfügbar.

3.2 Java Applet

Java ist eine plattform- und betriebssystemunabhängige objektorientierte Programmiersprache der Firma Sun Microsystems [Sun]. Bei Java Applets handelt es sich um kleine in einem Web-Browser ausführbare Javaprogramme. Sie können über das Internet auf einen beliebigen Zielrechner übertragen und dort ohne jeden Portierungs- oder Installationsaufwand ausgeführt werden.

Java Native Interface (JNI) Java ist plattformunabhängig entworfen worden, sodass alle Funktionen auf allen Systemen lauffähig sind. Systemnahe Eigenschaften lassen sich daher durch Java nicht ohne weiteres nutzen. Um diese verwenden zu können, bietet sich ein Zugriff auf native Methoden über JNI an. Diese sind üblicherweise nicht in Java, sondern in einer hardwarenahen Programmiersprache (etwa C++) implementiert.

Sicherheit Werden Applets im Browser ausgeführt, so haben sie aufgrund strengerer Sicherheitsrichtlinien weniger Rechte als gewöhnliche Programme. Beispielsweise dürfen sie nicht auf lokale Dateien zugreifen oder Netzwerkverbindungen zu Rechnern aufbauen, von denen sie nicht heruntergeladen wurden. Soll das Applet trotzdem sicherheitskritische Aktionen ausführen können, muss es signiert werden. Die Signatur wird dem Benutzer angezeigt und ist durch ihn zu akzeptieren, um das Applet auszuführen.

3.3 Java Anbindungen an OpenGL

Java selbst ist unter anderem durch den hardwarefernen Ansatz zum effizienten Rendern von 3D-Grafik ungeeignet. Es existieren jedoch einige Anbindungen von Java an OpenGL, mit denen dieses Problem behoben werden kann. Einige davon werden im Folgenden hinsichtlich ihrer Eignung zur Verwendung in diesem Projekt verglichen.

3.3.1 Java 3D

Java 3D [Java 3D] ist eine Bibliothek von Java-Klassen zur Darstellung dreidimensionaler Grafiken in Java-Applikationen und -Applets. Es handelt sich dabei um eine objektorientierte Abstraktion von OpenGL und DirectX Funktionen auf Basis eines Szenengraphen. Dieser bildet den logischen Aufbau der darzustellenden Objekte auf eine gleichartig aufgebaute, baumähnliche Struktur ab. Ein direktes Ausführen der OpenGL bzw. DirectX-Funktionen ist nicht vorgesehen. Daher können neue Funktionen der APIs erst verwendet werden, nachdem sie in Java 3D gekapselt wurden. Aufgrund der stetigen Weiterentwicklung der Videohardware hat dies wiederholt dazu geführt, dass neu unterstützte Leistungsmerkmale in Java 3D nur verzögert oder gar nicht verfügbar waren [Wp2].
Java 3D wurde seit 1997 von Sun Microsystems entwickelt, war zwischenzeitlich eingestellt und ist seit Sommer 2004 als Open Source freigegeben. Die aktuelle Version ist 1.5.0 und wurde im Dezember 2006 herausgegeben.

3.3.2 Lightweight Java Game Library (LWJGL)

LWJGL [LWJGL] ist eine Programmbibliothek, mit der ein Java-Programmierer über JNI auf OpenGL- und OpenAL-Funktionen zugreifen kann. Zusätzlich ist auch die Ansteuerung

von Spielecontrollern möglich, sodass es eine plattformunabhängige Alternative zu Microsofts DirectX darstellt. Der Fokus von LWJGL liegt auf der Spieleentwicklung u.a. auch für mobile Geräte, weshalb die API sehr schlank gehalten wurde. Ein Nachteil von LWJGL ist allerdings, dass die Benutzung aus einem Applet heraus nicht möglich ist. Zur Verwendung über das Internet bietet sich daher Java Webstart an [Sun2]. Dabei handelt es sich um eine Technologie von Sun Microsystems, die es ermöglicht, Java-Applikationen über das Netz mit nur einem Klick zu starten. Im Unterschied zu Java-Applets benötigen Java-Web-Start-Anwendungen jedoch keinen Browser, um ablaufen zu können. Im Gegensatz zum objektorientierten Java 3D ist die Syntax von LWJGL der von OpenGL extrem ähnlich, sodass sich die Einarbeitungszeit für Programmierer mit OpenGL Kenntnissen in Grenzen hält und bestehender OpenGL Code mit minimalem Aufwand konvertiert werden kann. Vor allem ermöglicht der direkte Zugriff auf OpenGL Funktionen eine sehr hohe Performance. Die aktuelle Version der von einem unabhängigen Programmiererteam als Open Source entwickelten LWJGL API ist 1.1. Sie wurde am 30. April 2007 veröffentlicht und unterstützt OpenGL bis zur Version 2.1 sowie die wichtigsten Extensions.

3.3.3 Java OpenGL (JOGL)

JOGL [JOGL] ermöglicht ebenso wie LWJGL den direkten Zugriff auf die OpenGL API mittels JNI. Eine weitere Gemeinsamkeit ist der Verzicht auf die Objektorientiertheit. Stattdessen werden wenige spezielle Java-Wrapperklassen bereitgestellt, die Schnittstellen zu den nativen Funktionen von OpenGL bieten. Vorteile dieses geringen Abstraktionsgrades sind ebenfalls das hervorragende Laufzeitverhalten und die einfache Portierbarkeit bestehenden OpenGL Codes. Außerdem kann ein Großteil des JOGL Codes automatisch aus den OpenGL C Header Files generiert werden, weshalb Änderungen der OpenGL API sehr schnell in JOGL übernommen werden können. Ein Nachteil ist der OpenGL Programmierstil, welcher darauf basiert, den globalen Grafikzustand zu manipulieren. Dadurch wird eine Strukturierung des Java Codes in sinnvolle Klassen stark erschwert.
Die Entwicklung von JOGL wurde ursprünglich von Kenneth Russell und Chris Kline begonnen und wird mittlerweile von der Game Technology Group von Sun Microsystems als Open Source weiterentwickelt und stellt die Referenzimplementation von JSR-231 (Java Bindings for OpenGL) dar. Vorteile von JOGL gegenüber LWJGL sind die denkbare standardmäßige Integration in einer späteren Java Version und die Verwendungsmöglichkeit in Java Applets. Dazu muss der Benutzer lediglich die Signatur der von Sun zertifizierten JOGL Bibliothek akzeptieren. Die aktuelle Version von JOGL ist JSR-231 1.1.0 vom 22. April 2007, welche OpenGL bis zur Version 2.1 mit den wichtigsten Extensions unterstützt.

3.3.4 Performance

Gegenstand dieses Kapitels ist der Performancevergleich von Java 3D, LWJGL, JOGL und purem OpenGL anhand eines simplen Java bzw. C++ Programms, dessen Funktionsweise mit der des späteren Hauptprogramms vergleichbar ist. Hierbei ist allerdings zu beachten, dass die Tests im Vorfeld dieser Arbeit (Frühjahr 2006) mit den zu diesem Zeitpunkt aktuellen Versionen der Bibliotheken durchgeführt wurden. Außerdem kommt die Implementation eventuell einzelnen APIs stärker entgegen als anderen, weshalb gerade das deutlich mehr Portierungsaufwand erfordernde Java 3D benachteiligt sein könnte. Bei der dargestellten Szene handelt es sich um ein Gitter aus 320x160 Vertices, welche unterschiedliche Farbwerte erhal-

ten. Anschließend wird das Gitter triangulisiert und mit Farbverläufen eingefärbt. Es wurden dabei zwei unterschiedliche Messreihen durchgeführt:

Statische Szene Darstellung einer zeitlich unveränderlichen Szene, die aber trotzdem für jedes Bild völlig unabhängig gerendert wird.

Dynamische Szene Die einzelnen Vertices nehmen zu den einzelnen diskreten Zeitpunkten verschiedene Farben an, zwischen denen zeitlich interpoliert wird, um eine Animation zu erzeugen. Damit erhöht sich die Anzahl der durch das Programm selbst durchgeführten Berechnungen erheblich.

Die mit dem Testrechner[1] gemessenen Ergebnisse sind in Abbildung 2 zu sehen. Es lassen

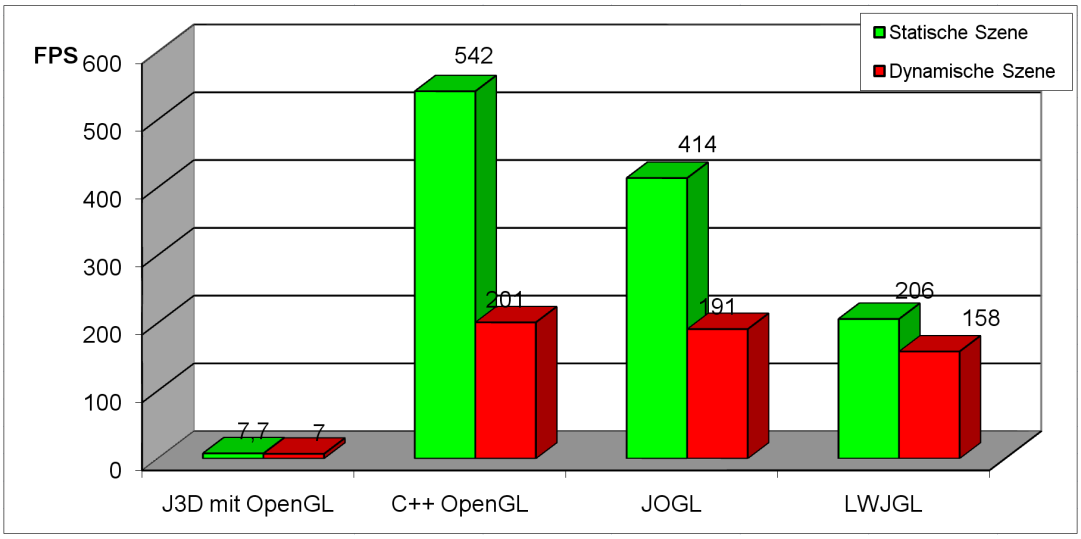


Abbildung 2: Performancevergleich verschiedener OpenGL Anbindungen

sich unter anderem die folgenden Beobachtungen machen:

- Java 3D ist völlig abgeschlagen.
- JOGL ist etwas schneller als LWJGL.
- JOGL ist nicht viel langsamer als das reine OpenGL Programm.
- Bei der dynamischen Szene liegen JOGL, OpenGL und LWJGL etwa gleichauf.

Wie bereits erwähnt, erheben diese Vergleichstests nicht den Anspruch auf Allgemeingültigkeit und sollen nicht etwa JOGL als LWJGL generell überlegen herausstellen. Trotzdem beeindruckt natürlich gerade der geringe Performanceunterschied zwischen der JOGL Applikation und dem C++ Programm, welches OpenGL direkt ansprechen kann. Interessanterweise nähern sich die Geschwindigkeiten der einzelnen Anbindungen im dynamischen Modus, in dem der Anteil der internen Berechnungen der Programme wesentlich höher ausfällt, deutlich. Demnach muss der etwa im JOGL Programm verwendete Java Code ähnlich performant ausführbar sein wie der C++ Code. Eine mögliche Erklärung ist unter anderem der Verzicht auf Speicheranforderungs- und Freigabeoperationen zur Laufzeit, die in Java bislang noch nicht sehr effizient ausgeführt werden.

[1] Systemkonfiguration: Athlon 64 3500+, GeForce 6800 GT, 1GB Ram, Windows XP

3.4 Zusammenfassung

Insgesamt stellen sich Java und JOGL als ideale Werkzeuge heraus, um eine Webapplikation mit hardwarebeschleunigter 3D-Grafik zu erstellen. Diese Lösung steht einem C++ Programm, welches OpenGL direkt ansprechen kann, weder in der Performance noch im Funktionsumfang wesentlich nach. Von seinem direkten Konkurrenten LWJGL hebt sich JOGL in erster Linie durch die Verwendbarkeit in Applets und durch die etwas bessere Performance ab. Eine nachträgliche Portierung ist aufgrund der nahezu identischen Syntax sowie der an OpenGL angelehnten Klassen- Felder- und Methodennamen mit minimalem Aufwand möglich.

4 Daten

4.1 Ausgangsdaten

Die in dieser Arbeit verwendbaren georeferenzierten Daten lassen sich grob in statische und dynamische einteilen. Daten ohne Zeitkomponente wie etwa Küstenlinien, Ländergrenzen, etc. geben geographische Merkmale wieder und werden primär benötigt, um dem Benutzer die räumliche Zuordnung der betrachteten Region zu ermöglichen. Sie werden üblicherweise im Shapefile Format der Firma ESRI [ESRI] hinterlegt. Zusätzlich können auch Erdoberflächentexturen als Orientierungshilfe verwendet werden.
Der Fokus dieses Projekts liegt auf der Animation dynamischer Klima- und Wetterdaten, welche zusätzlich zum räumlichen Bezug über eine Zeitkomponente verfügen. Von diesen existieren unabhängig voneinander messbare Ausprägungen wie Wind, Temperatur, Luftdruck, etc., welche sich natürlich gegenseitig beeinflussen und in ihrer Gesamtheit für das Verständnis des Klimas bzw. Wetters benötigt werden. Aus diesem Grund müssen die einzelnen Datenarten sowohl beliebig kombiniert als auch einzeln darstellbar sein, um den Betrachter nicht zu überfordern.

Wetter- und Klimadaten unterscheiden sich in erster Linie durch den Zeitraum, der bei einer Wettervorhersage üblicherweise auf wenige Tage beschränkt ist, während das Klima über Zeiträume von einigen Jahrzehnten ausgewertet wird. Des Weiteren werden Wetterberichte zumeist für kleinere Regionen erstellt, um örtliche Besonderheiten besser in die Prognosen einbeziehen zu können. Im Gegensatz dazu kann die Entwicklung des Klimas wesentlich besser weltweit verstanden und verdeutlicht werden.

4.1.1 GRIB-Files

Die Wetterdienste weltweit übertragen ihre Daten in der Regel mittels GRIB (Gridded Binary) Files [NCEP]. In GRIB-Files werden Wetterdaten in einem regelmäßigen rechteckigen Gitter gespeichert, welches sich einer bestimmten geographischen Region zuordnen lässt. Die eigentlichen Daten werden innerhalb eines GRIB-Files in einzelnen Records strukturiert. Ein Record beinhaltet eine bestimmte eindimensionale Kenngröße (Temperatur, Regenmenge, u-Komponente der Windgeschwindigkeit etc.) zu einem festgelegten Zeitpunkt. Ein GRIB-File kann aus beliebig vielen Records der einzelnen Wetterausprägungen zu verschiedenen Zeitpunkten bestehen. Bei nicht skalaren Daten existiert zu jedem Zeitpunkt ein Record für jede einzelne Komponente. Die Winddaten werden beispielsweise in je einem Record für die Windgeschwindigkeit in u-Richtung und in v-Richtung abgelegt. Folgende Informationen sind zu jedem Record eines GRIB-Files abrufbar:

- Art der Daten
- Extrema
- Zeitpunkt
- Zwei Winkelpaare, die die Begrenzung des Gitters angeben und ggf. eine Höheninformation
- Spalten- und Zeilenanzahl

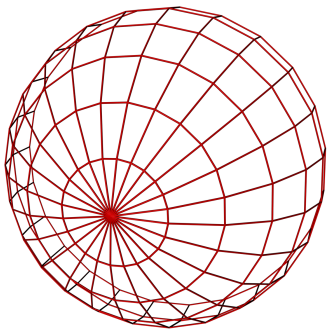
Neben Wetter- und Klimadaten werden auch andere Daten mit Zeit- und Raumbezug in GRIB-Files abgespeichert. Aufgrund der vielseitigen Verwendbarkeit des Formats zur Abspeicherung von äußerst verschiedenen Daten sind mit der Zeit unterschiedliche Dialekte entstanden. Dadurch wird das automatisierte Auslesen und Verarbeiten erschwert, da für GRIB-Files unterschiedlicher Herkunft ggf. das Ausleseprogramm an den jeweiligen Dialekt angepasst werden muss. Demnach sind trotz der hohen Verbreitung des Formats nur wenige Programme zum Auslesen der GRIB-Files verfügbar. Diese sind in der Regel nicht plattformunabhängig wie z.B. WGRIB [WGRIB] oder PINGO [PINGO] und folglich zur Verwendung in dieser Arbeit nicht besonders gut geeignet, weil somit die Installation mehrerer verschiedener Programme unausweichlich wäre. Aus diesem Grund wird ein an der Universität Osnabrück entwickelter `GRIBReader` [Kunze] verwendet, der in Java implementiert ist und sehr gut zur Integration in dieses Projekt geeignet ist.

4.1.2 Projektion der Rasterdaten

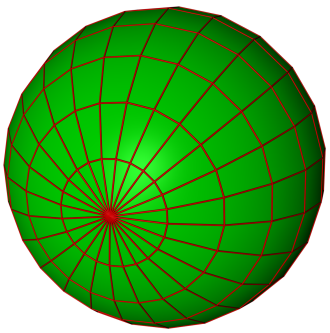
Gegenstand dieses Kapitels ist die Beschreibung eines Verfahrens zur Projektion der in den GRIB-Files enthaltenen Rasterdaten auf eine Geometrie, welche die Erdoberfläche repräsentiert. Zur Vereinfachung wird die Erde in dieser Arbeit als kugelförmig angenommen. Für jeden Punkt des Gitters können die Kugelkoordinaten $\varphi \in [0, 2\pi[$, $\theta \in [0, \pi]$ und r ermittelt werden. Diese lassen sich über

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = r \cdot \begin{pmatrix} cos(\varphi) \cdot sin(\theta) \\ sin(\varphi) \cdot sin(\theta) \\ cos(\theta) \end{pmatrix}$$

auf kartesische Koordinaten abbilden. Diese mathematischen Punkte auf der Oberfläche der mathematischen Kugel haben keine Ausdehnung und sind folglich nicht sichtbar. Für die verschiedenen Datenarten kommen nun jeweils geeignete Visualisierungsmechanismen zum Einsatz, welche auf unterschiedliche Art die Kugeloberfläche einfärben. Eine Möglichkeit besteht darin, die einzelnen Punkte zunächst zu einem Gitter zusammenzufassen (siehe Abb. 3(a))und die einzelnen Flächen des Gitters anschließend einzufärben (siehe Abb. 3(b)). Von



(a) Zusammenfassen der Punkte zu einem Gitter



(b) Einfärben der Gitterflächen

Abbildung 3: Visualisierung von Rasterdaten

der so erzeugten Oberfläche befinden sich allerdings nur die ursprünglichen Punkte auf der Kugeloberfläche. Der Rest befindet sich im Kugelinneren, wie in Abbildung 4(a) anhand

von Kugelsilhouetten veranschaulicht wird. Dort befindet sich die durch ein Gitter angenäherte Kugel (rot) mit Ausnahme ihrer Eckpunkte innerhalb einer mathematischen Kugel (grün). Grundsätzlich sollen verschiedene Klimaausprägungen wie beispielsweise Tempera-

(a) Näherung eines Gitters(rot) an eine Kugel(grün)

(b) Unterschiedliche Gitter schneiden sich

Abbildung 4: Veranschaulichung des Problems der Kugelnäherung durch ein Gitter anhand von Kugelsilhouetten

turen und Luftdruck gleichzeitig anzeigbar sein. Werden mehrere Schichten kombiniert, die sich etwa aufgrund unterschiedlicher Auflösungen nicht auf ein identisches Gitter abbilden lassen, durchdringen sich die erzeugten Gitter gegenseitig (siehe Abbildung 4(b)). Dieses generelle Problem kann zwar zu einer Reihe unschöner Effekte führen, wird aber in diesem Projekt ignoriert, da eine befriedigende Lösung im gegebenen Zeitrahmen nicht möglich ist (siehe Kapitel 10). Stattdessen erhalten die einzelnen Kugeln leicht unterschiedliche Radien, sodass bei den vorliegenden Daten und den verwendeten Visualisierungsmechanismen keine sichtbaren Darstellungsfehler auftreten.

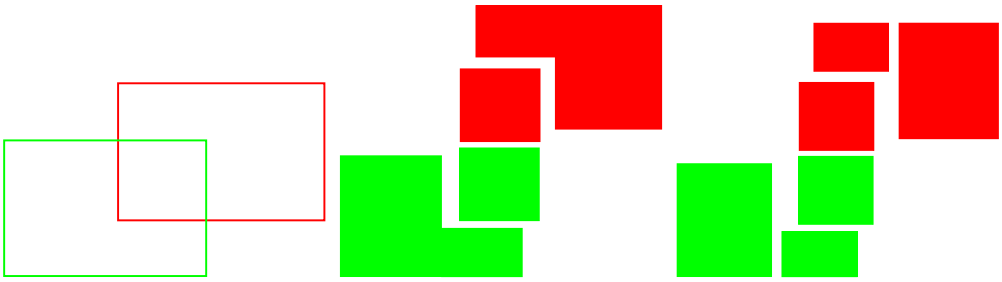
4.1.3 Verwendete Daten

Die aktuell verwendeten Daten stammen von Raymarine.com [Ray] und liegen im GRIB-File Format vor. Sie liefern eine dreitägige Wettervorhersage mit sechsstündigen Zeitintervallen und werden täglich aktualisiert. Darin enthalten sind Daten für Temperatur, Luftdruck, Wind und Niederschlag, welcher allerdings bislang nicht visualisiert wird. Einzelne GRIB-Files überdecken zumeist nur Teile der Erdoberfläche (etwa Europa oder Skandinavien).

4.2 Kombination von Kugelausschnitten

Da üblicherweise kein globales Modell für Wetterdaten verfügbar ist, müssen einzelne GRIB-Files geeignet kombiniert dargestellt werden können, um einen Großteil der Erdoberfläche abzudecken. In diesem Zusammenhang ist die Möglichkeit des räumlichen Schneidens verschiedener GRIB-Files zu beachten. Demnach müssen die Datensätze derart zerschnitten werden, dass sie sich nahtlos aneinander fügen. In den von mehreren GRIB-Files überdeckten Bereichen soll der Benutzer auswählen können, welcher Datensatz angezeigt wird. Schließlich können in einzelnen GRIB-Files die regionalen Besonderheiten besser zur Geltung kommen als in anderen, sodass nur der Anwender unter Berücksichtigung der für ihn interessantesten Region die Auswahl treffen kann.

Um den Anforderungen einer möglichst effizienten Visualisierung gerecht zu werden, soll beim Zerschneiden mehrerer Datensätze eine minimale Anzahl rechteckiger Messpunktanordnungen entstehen. Würde das Prinzip des regelmäßigen rechteckigen Gitters aufgegeben, müssten zu jedem Punkt die Koordinaten einzeln übertragen und verarbeitet werden. Die Folgen wären eine deutlich erhöhte Beanspruchung der Übertragungsbandbreite und eine verringerte Effizienz der Visualisierungsalgorithmen, da diese um einige Fallunterscheidungen erweitert werden müssten. Ein Nachteil der Einteilung in rechteckige Anordnungen ist allerdings die höhere Anzahl der erzeugten Teilstücke. Bei einer Zerschneideoperation zweier Rechtecke wird zunächst die Schnittmenge aus beiden Rechtecken bestimmt und von beiden abgetrennt (siehe Abbildung 5(a) und Abb.5(b)). Falls es sich bei den Teilstücken nicht um Rechtecke handelt, müssen sie in einem zweiten Schritt weiter zerteilt werden (siehe Abb.5(c)). Der Algorithmus zum Schneiden zweier Rechtecke erzeugt bei allen möglichen



(a) Vor dem Zerschneiden (b) Ausschneiden der Schnittmenge aus beiden Rechtecken (c) Rechteckseinteilung

Abbildung 5: Zerteilen zweier Rechtecke

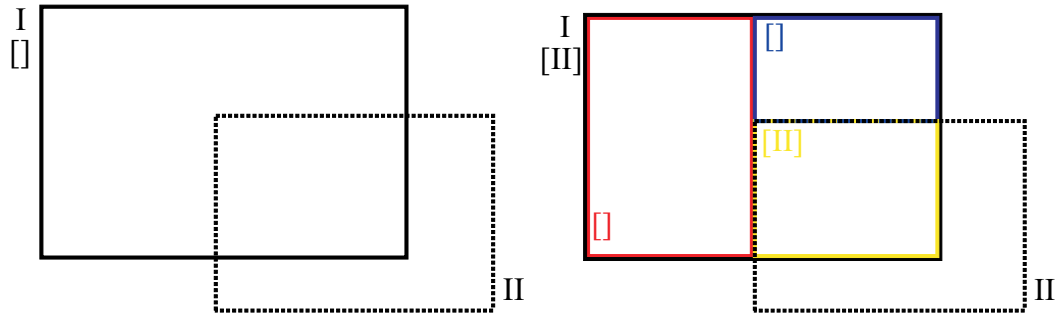
Arten der Überlagerung eine minimale Anzahl von Teilrechtecken. Weil beim Zerschneiden zweier Rechtecke auf die beschriebene Weise ausschließlich Rechtecke entstehen, bietet sich die Repräsentation eines Rechtecks durch eine rekursive Datenstruktur an. Ein solches Rechteck enthält die folgenden Informationen:

- Räumliche Grenzen
- Index des GRIB-Files, zu dem es gehört
- Eine Menge von Indices der es überlagernden GRIB-Files

Dann kann mithilfe des im Folgenden erläuterten Verfahrens das Zerschneiden beliebig komplizierter Überlagerungen von GRIB-Files auf Vergleiche von je zwei Rechtecken zur Zeit reduziert werden:

- Alle GRIB-Files werden paarweise verglichen
- Schneiden sich zwei noch unzerteilte Rechtecke (≈GRIB-Files), wird wie oben beschrieben verfahren. Die neu erzeugten Rechtecke werden in den jeweiligen ursprünglichen Rechtecken abgespeichert. Davon vermerken die Rechtecke aus dem Überlappungsbereich den Index des jeweils anderen GRIB-Files (siehe Abb.6).
- Schneidet sich ein bereits zerteiltes Rechteck mit einem GRIB-File, so werden die darin enthaltenen Rechtecke mit diesem GRIB-File verglichen. Lediglich solche Rechtecke, die sich mit dem anderen GRIB-File schneiden werden weiter untersucht.

Diese Vergleiche laufen rekursiv durch, bis Rechtecke verglichen werden, welche keine weiteren enthalten und somit auf die beschriebene Art geschnitten werden können (siehe Abb.7).

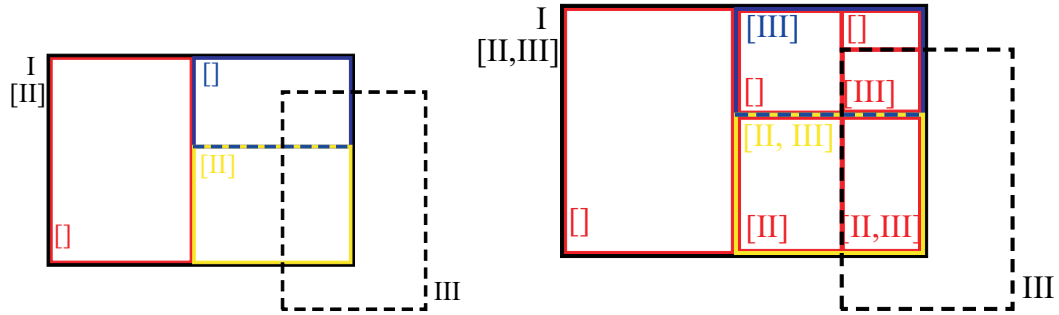


(a) Vor dem Zerschneiden des GRIB-Files I durch II.

(b) Es entstehen drei Rechtecke, von denen eins durch das GRIB-File II überdeckt wird.

Abbildung 6: Zerschneiden eines unzerteilten Rechtecks. Zahlen stehen für den GRIB-File Index. Indices in Klammern sind von GRIB-Files, die ein Rechteck schneiden.

Das Ergebnis dieses Algorithmus ist genau ein Wurzel-Rechteck für jedes GRIB-File, welches ggf. eine Menge weiterer rekursiver Rechtecke enthält. Dabei ist die resultierende Einteilung unabhängig von der Reihenfolge der Vergleiche der GRIB-Files und jeder Vergleich von zwei Rechtecken erzeugt eine minimale Anzahl von Teilrechtecken.

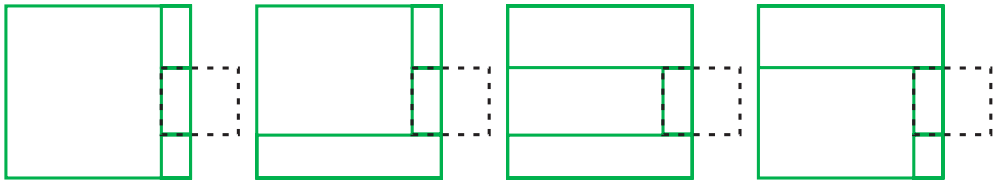


(a) Vor dem Zerschneiden des GRIB-Files I durch III.

(b) Zwei der in I enthaltenen Rechtecke werden erneut unterteilt.

Abbildung 7: Zerschneiden des bereits einmal geschnittenen Rechtecks.

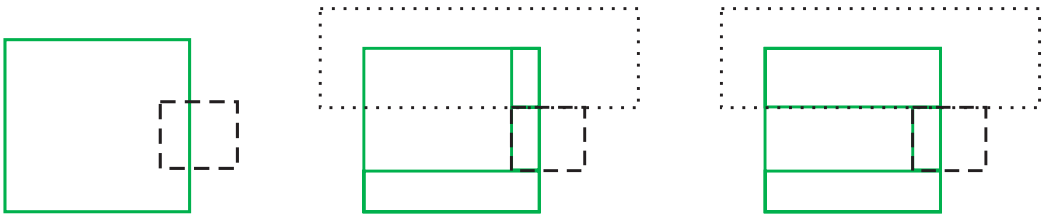
Verbesserungsmöglichkeit Die Art der Zerteilung beim Schneiden zweier Rechtecke kann nicht immer eindeutig festgelegt werden, wie anhand des Beispiels in Abbildung 8 erkennbar ist. Ein denkbares Entscheidungskriterium wäre das Auffinden einer Gesamtlösung mit einer minimalen Anzahl von Rechtecken. In dem in Abbildung 9(a) gezeigten Beispiel wird das grüne Rechteck von dem schwarz gestrichelten geschnitten. Die möglichen Einteilungen ergeben sich analog zu denen aus Abbildung 8. Schneidet nun ein weiteres GRIB-File das bereits zerteilte Rechteck, so wird in der Regel mindestens ein darin enthaltenes Rechteck weiter zerschnitten (siehe Abb. 9(b)). In diesem Beispiel kann jedoch eine Aufteilung gewählt werden, bei der alle Rechtecke entweder vollständig innerhalb oder außerhalb des



(a) Fall 1 (b) Fall 2 (c) Fall 3 (d) Fall 4

Abbildung 8: Alternative Zerlegungen des geschnittenen Rechtecks in eine minimale Zahl von Rechtecken

schneidenden GRIB-Files liegen (siehe Abb. 9(c)). Tritt dieser Sonderfall ein, ist ein weiteres Unterteilen der Rechtecke nicht nötig.
Davon abgesehen könnten Rechteckaufteilungen bevorzugt werden, deren schmalstes Rechteck am breitesten ist. Auf diese Weise kann eine größere Zahl von Zoomstufen bereitgestellt werden, da diese abhängig von der kleinsten Kantenlänge ist.
Des Weiteren sind Lösungen mit Rechtecken, die über sehr unterschiedliche Kantenlängen verfügen, schlechter zu bewerten. Grund dafür ist ihre im Verhältnis zum Flächeninhalt überaus große Wahrscheinlichkeit, zumindest teilweise im sichtbaren Bereich zu liegen. Folglich sinkt die Wahrscheinlichkeit des Clippens dieser Rechtecke, was in einer erhöhten zu ladenden und zu verarbeitenden Datenmenge resultiert.
Die letzten beiden Kriterien können auch lokal ausgewertet werden, allerdings garantiert nur eine globale Entscheidung ein optimales Ergebnis. Insgesamt rechtfertigen diese marginalen Verbesserungen der Rechteckseinteilung den mit einem globalen Algorithmus verbundenen Aufwand nicht, weshalb in diesem Projekt das bereits beschriebene lokale Verfahren zum Einsatz kommt.



(a) Schnitt mit erstem GRIB-File (b) Zerteilen nötig bei Schnitt mit weiterem GRIB-File (c) Kein Zerteilen bei Schnitt mit weiterem GRIB-File

Abbildung 9: Beispiel zum Auffinden einer globalen Lösung mit minimaler Rechteckzahl

4.3 Datenstruktur

Ein zentrales Problem des Projekts liegt in der begrenzten Bandbreite, mit der die Daten über das Internet nachgeladen werden können. Folglich muss der Minimierung der zu ladenden Datenmenge eine hohe Priorität gegeben werden. Wünschenswert ist dabei die Erfüllung der folgenden Kriterien:

- Räumliche Teilmenge
  Geladen werden sollen idealerweise ausschließlich Daten, welche im sichtbaren Be-

reich liegen. Insbesondere soll keine Anforderung von Daten aus folgenden Gebieten erfolgen:

  - Auf der Erdkugelrückseite
  - Außerhalb des Zoomfensters
  - Von anderen GRIB-Files verdeckten Bereiche

- Zoomstufen
  Auswahl der Daten in einer in Abhängigkeit von der Kameraentfernung zur Oberfläche optimalen Auflösung.

- Zeitliche Teilmenge
  Eine Animation beschreibt den zeitlichen Verlauf zwischen den Daten des aktuellen Zeitpunkts zu denen des nächsten. Demnach muss die Möglichkeit bestehen, Daten zu einzelnen Zeitpunkten zu laden und nicht etwa alle Zeitpunkte vor dem aktuellen ebenfalls auslesen zu müssen.

- Teilmenge benötigter Wetterelemente
  Die einzelnen Wetterausprägungen müssen auch einzeln anzeigbar sein. Somit sollten beispielsweise bei der Anzeige der Temperatur ausschließlich Temperaturdaten geladen werden.

- Kompression
  Verwendung geeigneter Algorithmen zur Reduzierung der Datenmenge

Dateien stellen ein eindimensionales Datenformat dar, weshalb die vierdimensionalen Wetterdaten (phi, theta, Zeitpunkt, Datenart) hintereinander darin abgelegt werden müssen. Das Auslesen der Dateien erfolgt ausschließlich sequentiell, sodass ein effizienter Zugriff auf Teilmengen unmöglich ist, da alle vor den gewünschten Werten liegenden Daten nicht übersprungen werden können. Außerdem enthalten die einzelnen GRIB-Files weder Informationen über ihre gegenseitige Überlappung noch vorberechnete Zoomstufen, weswegen ein direktes Auslesen der GRIB-Files einen völlig ungeeigneten Ansatz darstellt. Stattdessen muss eine Möglichkeit des strukturierten Abspeicherns der Daten gefunden werden, welche die obengenannten Anforderungen erfüllt.

4.3.1 DBMS (Database Management System)

Eine DBMS besteht aus einer Menge von gespeicherten Daten und Programmen zur Verwaltung dieser Datenbasis. Unter anderem ermöglichen sie das strukturierte Abspeichern hochdimensionaler Daten und bieten effiziente Zugriffsmethoden darauf an.

Das Relationale Modell Gegeben sind n nicht notwendigerweise unterschiedliche Wertebereiche $D_1, ..., D_n$, welche lediglich unstrukturierte atomare Werte enthalten. Dann ist eine Relation R definiert als Teilmenge des kartesischen Produkts der n Wertebereiche (Domänen) [Kemper]:

$$R \subseteq D_1 \times D_2 \times ... \times D_n$$

Dabei wird zwischen dem Schema einer Relation, welches durch die n Domänen gegeben ist und der aktuellen Ausprägung (Instanz) unterschieden. Ein Element der Menge R, dessen Stelligkeit sich aus dem Relationenschema ergibt, wird als Tupel bezeichnet. Anfragen an ein relationales Datenbanksystem sind auf verschiedene Arten möglich:

- Relationenkalkül
  Beschreibt unter Verwendung der Prädikatenlogik erster Stufe deklarativ welche Bedingungen die gewünschten Daten erfüllen sollen, nicht jedoch wie eine Anfrage ausgewertet werden kann.

- Relationale Algebra
  Verknüpft konstruktiv die vorhandenen Relationen durch Operatoren wie $\cup, \cap,$. Die Sprache ist stärker prozedural orientiert, d.h. ein Relationalalgebraischer Ausdruck beinhaltet implizit einen Abarbeitungsplan, wie die Anfrage abzuarbeiten ist. Aus diesem Grund spielt die Relationale Algebra eine größere Rolle bei der Realisierung von Datenbanksystemen und insbesondere bei der Anfrageoptimierung.

- SQL
  Stellt eine in Umgangssprache gegossene Mischung aus Relationaler Algebra und Relationenkalkül dar und wird von den meisten kommerziellen Relationalen Datenbanksystemen verwendet.

Straightforward Datenbankstruktur Eine Möglichkeit wäre die Abspeicherung in Form von sechsdimensionalen Datenpunkten der Art [Messwertdichte, phi, theta, Zeitpunkt, GRIB-File-Id, Datenart] in einer Relationalen Datenbank. Dieser Ansatz erfüllt von der Kompression abgesehen alle geforderten Kriterien. Allerdings beanspruchen Suchanfragen extrem viel Rechenzeit des Servers, da eine große Tupelmenge unter Auswertung einer Vielzahl von Vergleichen gefunden und anschließend nach allen sechs Dimensionen sortiert werden muss. Ein ähnlicher Ansatz mit lediglich vierdimensionalen Tupeln wurde an der Universität Osnabrück bereits getestet [CGP]. Die dort aufgetretenen mehrminütigen Wartezeiten bei Datenbankanfragen belegen die extrem schlechte Eignung dieses Ansatzes.

Teilerhalt der räumlichen Ordnung beim Speichern Bei Betrachtung der beiden vorherigen Ansätze werden folgende Punkte deutlich:

- Aufgrund der sequentiellen Anordnung der Werte in Dateien ist ein effizientes Auslesen von Teilmengen nicht möglich. Da jedoch räumlich zusammenhängende Werte sequentiell abgespeichert sind, können diese Bereiche effizient ausgelesen werden und ein nachträgliches Sortieren entfällt.

- Die beschriebene Datenbank dagegen ermöglicht das Auslesen von genau den benötigten Werten. Da aufgrund der mengenwertigen Abspeicherung alles Wissen über Nachbarschaftsbeziehungen verloren geht, sind die Suchanfragen extrem aufwändig.

- Gefragt sind immer größere räumlich zusammenhängende Bereiche, wobei diese Region üblicherweise nicht der in einem GRIB-File enthaltenen entspricht.

- Angefordert wird immer genau ein Zeitpunkt, eine bestimmte Wertdichte und eine beliebige Kombination von Datenarten. Folglich macht ein Erhalt der Ordnung in diesen Dimensionen keinen Sinn.

Offensichtlich wird eine Datenbank benötigt, die das Auslesen von Teilmengen in jeder Dimension gestattet. Die Effizienz kann massiv gesteigert werden, wenn die räumliche Ordnung erhalten bleibt. Da die betrachteten Bereiche zumeist nicht mit dem durch ein GRIB-File abgedeckten Gebiet übereinstimmen, sollten die Daten in kleinere räumlich zusammenhängende Gebiete zerteilt werden, um das Auslesen von räumlichen Teilbereichen zu ermöglichen. Das Verfahren funktioniert folgendermaßen:
Zunächst werden die GRIB-Files wie in Kapitel 4.2 beschrieben gemäß ihrer Überlappung zerschnitten. Die resultierenden rechteckig angeordneten Datenpunkte (Abbildung 10(a)) werden zu atomaren Rechtecken unter Beibehaltung ihrer räumlichen Ordnung zusammengefasst (Abbildung 10(b)). Zu beachten ist hierbei, dass eine zu grobe Rechteckseinteilung das Laden einer unnötig großen Datenmenge bewirkt, während durch eine zu feine Aufteilung die Datenbankanfragen wesentlich aufwändiger werden. Folglich ist ein Kompromiss in Abhängigkeit der verfügbaren Datenbandbreite und der erwarteten Serverbelastung zu finden. Nach Festlegung der räumlichen Zerteilung wird für jedes dieser Rechtecke aus jedem

(a) Ausgangsrechteck (b) Atomare Rechtecke

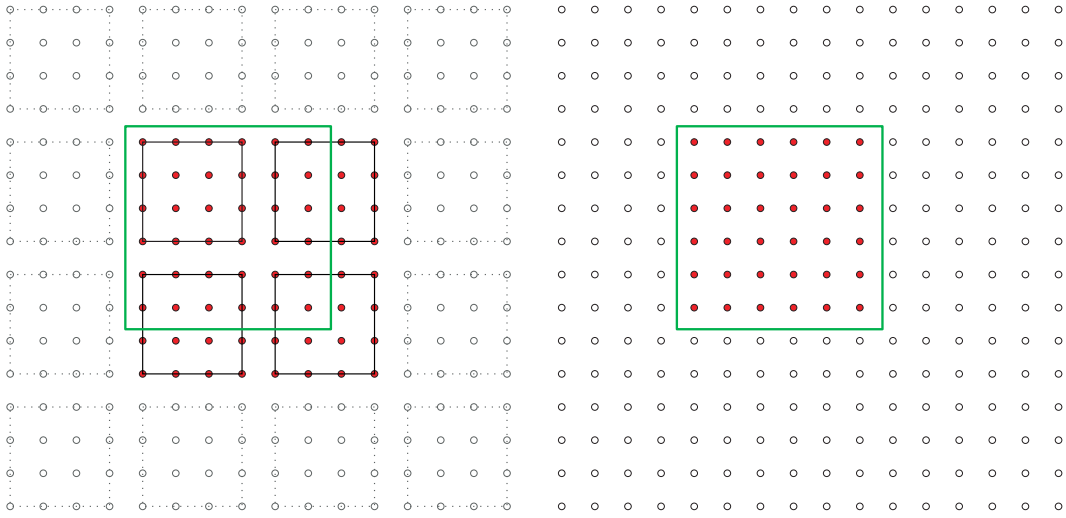
Abbildung 10: Zerteilen in atomare Rechtecke

GRIB-File für jede Datenart zu jedem Zeitpunkt in jeder Wertdichte ein Tupel in die Datenbank eingefügt.
Eine Suchanfrage bestimmt zunächst diejenigen Rechtecke, welche innerhalb des Sichtfensters liegen oder es schneiden (siehe Abb. 11(a)). Diese werden für den gewünschten Zeitpunkt und die beste Wertdichte für jede gewünschte Datenart aus einem obenliegenden GRIB-File vollständig geladen. Angezeigt wird davon lediglich die sichtbare Teilmenge der Werte (siehe Abbildung 11(b)).

4.3.2 Datenkompression

Unter Kompression wird ein Vorgang verstanden, welcher eine Repräsentation der Ausgangsdaten bestimmt, die weniger Speicherplatz benötigt. Dies kann beispielsweise durch Verminderung der Redundanz in den Daten erreicht werden. Generell wird zwischen verlustbehafteten und verlustfreien Kompressionsverfahren unterschieden. Im Rahmen dieser Arbeit wurde die Verwendung folgender Verfahren geprüft:

- Zip
  Das von Phil Katz entwickelte Zip-Dateiformat[Pkware] basiert auf dem von Ziv und Lempel entwickelten Algorithmus LZ78 [Wp5] zur Kompression beliebiger Zeichen-



(a) Laden sichtbarer Rechtecke (durchgezogene Linien)

(b) Anzeige sichtbarer Daten (rot)

Abbildung 11: Veranschaulichung des Datenlademechnismus

folgen. Dieser basiert auf der Idee, in einer sogenannten Präfix-Tabelle die Anfangsstücke bereits gelesener Strings zu speichern und wiederholtes Auftauchen derselben Strings durch Verweise in die Tabelle zu kodieren. Das Verfahren erzielt folglich gerade bei Daten mit hohem Redundanzgrad sehr gute Ergebnisse. In der Praxis ermöglicht dieser Algorithmus die verlustfreie Verringerung der Wetterdatengröße auf im Mittel etwa 33%. Die zusätzliche CPU-Belastung beim Dekomprimieren kann als vernachlässigbar eingestuft werden, weshalb die Zip-Kompression verwendet wird.

- Verlustbehaftete Kompression
  Nicht verlustfreie Verfahren entscheiden zumeist, welcher Anteil der Information für den Empfänger weniger wichtig bzw. idealerweise unnötig ist. Die Möglichkeit der verlustfreien Wiederherstellung der Ausgangsdaten kann dabei nicht garantiert werden. Ein Beispiel für ein solches Verfahren soll im Folgenden erläutert werden:
  In den GRIB-Files sind die einzelnen Werte als 32 Bit Gleitkommazahlen hinterlegt. Die meisten Visualisierungstechniken bilden diese Werte auf diskrete Farben oder Symbole ab. Der Bewölkungsgrad wird beispielsweise über 256 Graustufen visualisiert, die vorberechnet und dann in 8 Bit kodiert werden könnten. Folglich ist diese verlustbehaftete Kompression in Bezug auf die resultierende Darstellung verlustfrei. Allerdings hat diese Technik einige Nachteile:

  - Durch die Vorbestimmung der Visulisierung verringert sich der Interaktivitätsgrad
  - Das Verfahren funktioniert beispielsweise bei der Isoliniendarstellung nicht
  - Bei zeitlicher Interpolation kann die Exaktheit nicht garantiert werden

  Insgesamt stellt das Verfahren eine Ergänzung des Zip-Algorithmus bei sehr hohen Datenauflösungen dar. Aktuell sind solche Auflösungen nicht verfügbar, weshalb das Verfahren aufgrund der beschriebenen Nachteile nicht verwendet wird.

4.3.3 Umsetzung

Die Umsetzung der beschriebenen Datenbankstruktur könnte auch durch ein Dateisystem erfolgen, welches für jedes Rechteck eine eigene Datei anlegt. Da der Zugriff über das Internet auf viele kleine Dateien zu weit größeren Zeitverzögerungen führt als eine einzelne Datenbankanfrage, wird eine MySQL-Datenbank [MySQL] zum Verwalten der Daten verwendet. Außerdem bleibt so die Option einer weitergehenden Verwendung der Datenbankfunktionalität in Zukunft erhalten.

Tabellenstruktur In der Datenbank existieren zum einen Tabellen mit den dynamischen Daten und zum anderen Tabellen mit statischen Informationen, welche die dynamischen Daten beschreiben. Im Folgenden wird die Struktur der wichtigsten Tabellen erläutert. Die atomaren Rechtecke werden als Binary Large Object (Blob) kodiert und in die folgende Tabelle eingefügt:

DataRectangle : {[rectangleID, dateID, valDensity, values : Blob]}

Welche GRIB-Files ein gegebenes Rechteck überlagern ist der Tabelle

HideRelationship : {[rectangleID, gribFileID]}

zu entnehmen. Geographische Informationen und Zugehörigkeit zu einem GRIB-File lassen sich aus der Tabelle

RectangleInf : {[gribFileID, rectangleID, phiMin, phiMax, thetaMin, thetaMax, rowCnt, colCnt]}

ablesen. Tabellen dieser Art existieren für jeden Wetterdatentyp separat, sodass hierfür kein weiterer Index benötigt wird.

Füllen der Datenbank Die Aktualisierung der Daten soll in folgenden Schritten erfolgen:

1. Ein automatisiertes Script überprüft in regelmäßigen Abständen die Webseite des Datenanbieters auf aktualisierte Ausgangsdaten. Liegen Daten vor, die aktueller als die in der Datenbank enthaltenen sind, werden sie heruntergeladen und die folgenden Schritte angestoßen.
2. Der `GribReader` liest die Daten aus.
3. Die GRIB-Files werden ihrer Überlappung entsprechend zerschnitten.
4. Die resultierenden Rechtecke werden weiter zu den atomaren Rechtecken zerteilt.
5. Von den atomaren Rechtecken werden solange geringer aufgelöste erzeugt, bis eine bestimmte Wertdichte unterschritten wird.
6. Die Datenrechtecke werden komprimiert (siehe Kapitel 4.3.2).
7. Einfügen der Daten und der die Daten beschreibenden Tabellen auf die beschriebene Weise in die Datenbank.

Der Prozess ist außerdem in dem in Abbildung 12 dargestellten Diagramm erkenntlich. Allerdings ist das automatisierte Script aus Zeitgründen noch nicht implementiert, weshalb der Prozess bislang manuell gestartet werden muss.

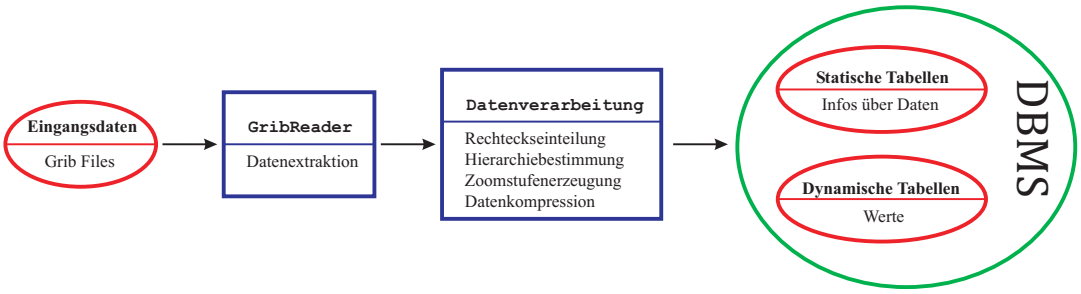


Abbildung 12: Datenflussdiagramm zur Veranschaulichung des Datenbankfüllprozesses

Auslesen der Daten Das Auslesen der Daten,, wird mit der folgenden Sequenz initiiert:

1. Das Applet stellt mithilfe eines JDBC-Treibers eine Verbindung zur MySQL Datenbank her.
2. Die statischen Tabellen der Datenbank werden ausgewertet. Danach ist dem Applet die Struktur der dynamischen Daten bekannt und diese können direkt über ihre Indices angefordert werden. Somit muss keine Suche im eigentlichen Sinn mehr durchgeführt werden.
3. Laden der benötigten Daten für die ersten beiden Zeitpunkte.

Anschließend läuft eine Schleife, in der bei einer Zustandsänderung (Kameraposition, Zeitpunkt, GRIB-File-Hierarchie, etc.) überprüft wird, ob alle benötigten Daten vorliegen. Ist dies nicht der Fall, werden die fehlenden Daten aus der Datenbank angefordert. Der Prozess ist in dem in Abbildung 13 enthaltenen Diagramm dargestellt.

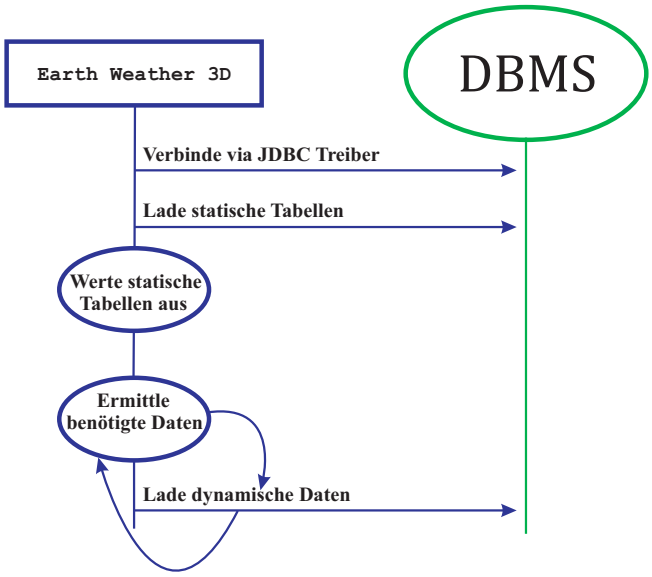


Abbildung 13: Diagramm zur Veranschaulichung der Kommunikation zwischen dem Applet und der Datenbank des Servers

4.3.4 Bewertung

Die in diesem Kapitel beschriebene Datenstruktur ermöglicht das Laden der Daten für nahezu ausschließlich den sichtbaren Bereich der Erdoberfläche. Durch diesen Ansatz kann die Fülle der zu übermittelnden Daten etwa konstant gehalten werden, indem diese in einer Dichte angefordert werden, die umgekehrt proportional zur Größe des sichtbaren Bereichs ist. Folglich resultiert eine höhere Maximalauflösung auf dem Server nicht in einer gestiegenen Beanspruchung der Bandbreite, sondern ermöglicht weitere Zoomstufen. Durch den Teilerhalt der räumlichen Ordnung ist ein Ausführen der Datenbankanfragen mit sehr geringer Beanspruchung der Rechenleistung des Servers ermöglicht worden. Das Laden eines im Vergleich mit dem Sichtfenster oft etwas größeren Bereichs ist kein Nachteil, da auf diese Weise eine leichte Änderung der Kameraposition nicht zwangsläufig in einer neuen Datenbankquery resultiert. Insgesamt ermöglicht die verwendete Datenstruktur das ausreichend schnelle Nachladen der Daten zur Laufzeit und stellt somit einen der wichtigsten Punkte für das Gelingen des Projekts dar. In der Praxis ist ein herkömmlicher DSL 1000 Anschluss für eine vollständig flüssige Animationsgeschwindigkeit bereits mehr als ausreichend. Die Verwendung der *Straightforward Datenbank* oder ein direktes Auslesen der GRIB-Files dagegen hätten das Projekt zum Scheitern verurteilt. Trotzdem wäre es bei einer alternativen Desktop-Applikation durchaus sinnvoll, eigene GRIB-Files wahlweise auch direkt einlesen zu können.

5 Animation

Schwerpunkt dieses Projekts ist die Visualisierung zeit- und raumbezogener Daten. Die Veranschaulichung der zeitlichen Veränderung, beispielsweise des Klimas, ist nur mithilfe der Animation möglich. Um den Eindruck einer flüssigen Animation zu erzeugen, müssen die folgenden Punkte erfüllt sein:

- Animationsgeschwindigkeit
  Die Zeitdifferenz zwischen zwei Animationsschritten darf nicht zu groß ausfallen. Wünschenswert ist eine Framerate von mehr als 24 Bildern pro Sekunde.

- Gleichmäßige Geschwindigkeit
  Läuft die Animation nicht mit konstanter Geschwindigkeit ab, wird dem Betrachter ein falsches Zeitgefühl vermittelt.

- Zeitliche Stetigkeit
  Die einzelnen Visualisierungsalgorithmen müssen sicherstellen, dass sich die Darstellungen zweier beliebig dicht liegender Zeitpunkte beliebig wenig unterscheiden. Ist dieses Kriterium nicht erfüllt, wirkt die Animation selbst bei hoher Framerate nicht flüssig und gleichmäßig.

5.1 Funktionsweise

Für die Animation zwischen zwei Zeitpunkten t_n und t_{n+1} werden die Werte für den aktuellen Zeitpunkt t_{akt} durch lineare zeitliche Interpolation der Daten ermittelt und anschließend visualisiert. Dementsprechend benötigt das Programm lediglich zwei Zeitpunkte auf einmal. Somit kann die Animation bereits nach dem Laden der ersten beiden Zeitpunkte gestartet werden, während die Anforderung weiterer Zeitpunkte erst bei Bedarf erfolgt. Allerdings würde das Laden der Daten unmittelbar vor der Verwendung eine ungleichmäßige Geschwindigkeit bewirken, da die Animation angehalten werden muss bis die Daten verfügbar sind. Das Problem kann gelöst werden, indem der Zeitpunkt t_{n+2} bereits geladen wird, während die Animation zwischen den Zeitpunkten t_n und t_{n+1} verläuft. Damit dies parallel im Hintergrund geschehen kann, muss das Programm mehrläufig ausgelegt sein. Folglich sollte sich ein extra Thread ausschließlich um das Laden und Aufbereiten (dekomprimieren etc.) der Daten kümmern. Auf diese Weise kann eine sehr gleichmäßige Animationsgeschwindigkeit erreicht und auf Mehrkernprozessorsystemen durch die gleichmäßigere Auslastung der Kerne sogar die Performance verbessert werden.

5.2 Probleme

Für die Animation wird vereinfachend eine lineare zeitliche Änderung der Werte angenommen. Dies ist gerade bei der geringen zeitlichen Auflösung von 6h der aktuell verwendeten Daten äußerst problematisch. So könnte beispielsweise die Temperatur drei Stunden vor Sonnenuntergang näherungsweise konstant bleiben und in den drei Stunden danach drastisch abfallen. Offensichtlich wird dies bei einer linearen Interpolation nicht berücksichtigt, weshalb die Visualisierung stark von der Realität abweicht. In diesem Zusammenhang wird auch deutlich, dass ausschließlich GRIB-Files kombiniert werden sollten, welche Records zu identischen Zeitpunkten enthalten.

6 Darstellungsformen der Wetterdaten

Allgemein ist die Qualität der Visualisierung verschiedener Daten maßgeblich von den in Abbildung 14 dargestellten Faktoren abhängig. Diese stehen oft zumindest teilweise im ge-

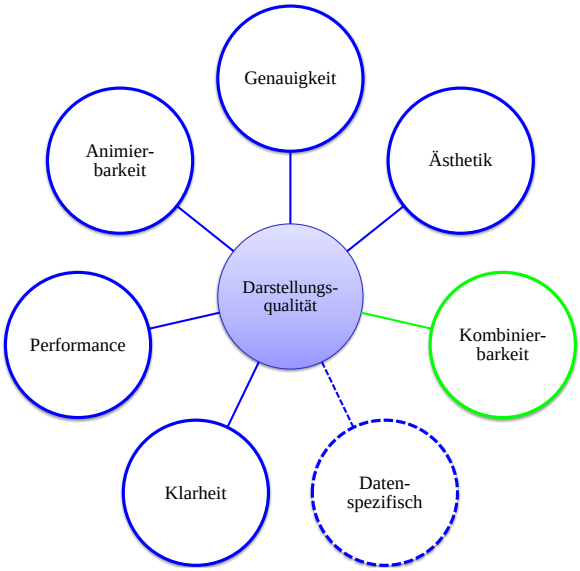


Abbildung 14: Qualitätskriterien der Darstellungstechniken

genseitigen Widerspruch, sodass ein geeigneter Kompromiss zu finden ist. So muss häufig zugunsten der Klarheit bzw. Anschaulichkeit auf eine maximale mathematische Genauigkeit der Darstellung verzichtet werden. Beispiel dafür ist die bereits in Kapitel 5 erläuterte Animation, welche zur Veranschaulichung der Dynamik des Klimas benötigt wird. Die dazu verwendete zeitliche Interpolation der Messwerte kann jedoch zu einer deutlichen Abweichung von der Realität führen. Des Weiteren muss jede Darstellungstechnik zeitlich stetig animierbar sein, wie in Kapitel 5 gefordert. Von diesen jeweils bei der Visualisierung eines Datentyps relevanten Kriterien abgesehen, die in Abbildung 14 blau eingekreist dargestellt sind, ist auf eine gute Kombinierbarkeit der Visualisierungstechniken verschiedener Daten zu achten. Zusätzlich zu diesen allgemeinen Kriterien sind bei den verschiedenen Wetterelementen jeweils spezifische Charakteristika besonders zu verdeutlichen, was im Folgenden näher erläutert wird.

6.1 Luftdruck

Der Luftdruck stellt eine fundamentale Größe für die Wetter- und Klimaentwicklung dar. Entscheidend sind vor allem große räumliche Druckgefälle, durch die Winde oder Stürme entstehen können [Walch]. Bei zusätzlicher starker Temperaturänderung besteht außerdem eine hohe Gewitterwahrscheinlichkeit. Davon abgesehen kennzeichnen Gebiete hohen Luftdrucks eine stabile Wetterlage, während Tiefdruckgebiete für wechselhaftes Wetter stehen. Damit stellt der Luftdruck ein Maß für die Verlässlichkeit einer Wettervorhersage dar.
Die Visualisierung des Luftdrucks, gegeben durch einen skalaren Wert, erfolgt mithilfe von Isolinien, welche Punkte gleichen Wertes auf der Wetterkarte verbinden. Werden Punkte gleichen Luftdrucks verbunden spricht man von einer Isobare. Durch die Abstände der Isobaren

ist das Luftdruckgefälle direkt ablesbar und auch die Hoch- bzw. Tiefdruckgebiete können unmittelbar erkannt werden. Damit ist dieses Verfahren besonders gut geeignet, die entscheidenden Eigenschaften des Luftdrucks zu vermitteln. Da nicht die gesamte Erdoberfläche einfärbt wird, ist die Kombination mit anderen Visualisierungstechniken problemlos möglich. Die Isoliniendarstellung des Luftdrucks ist in Abbildung 15 zu sehen, das Verfahren zu ihrer Erstellung wird in Kapitel 7 erläutert.

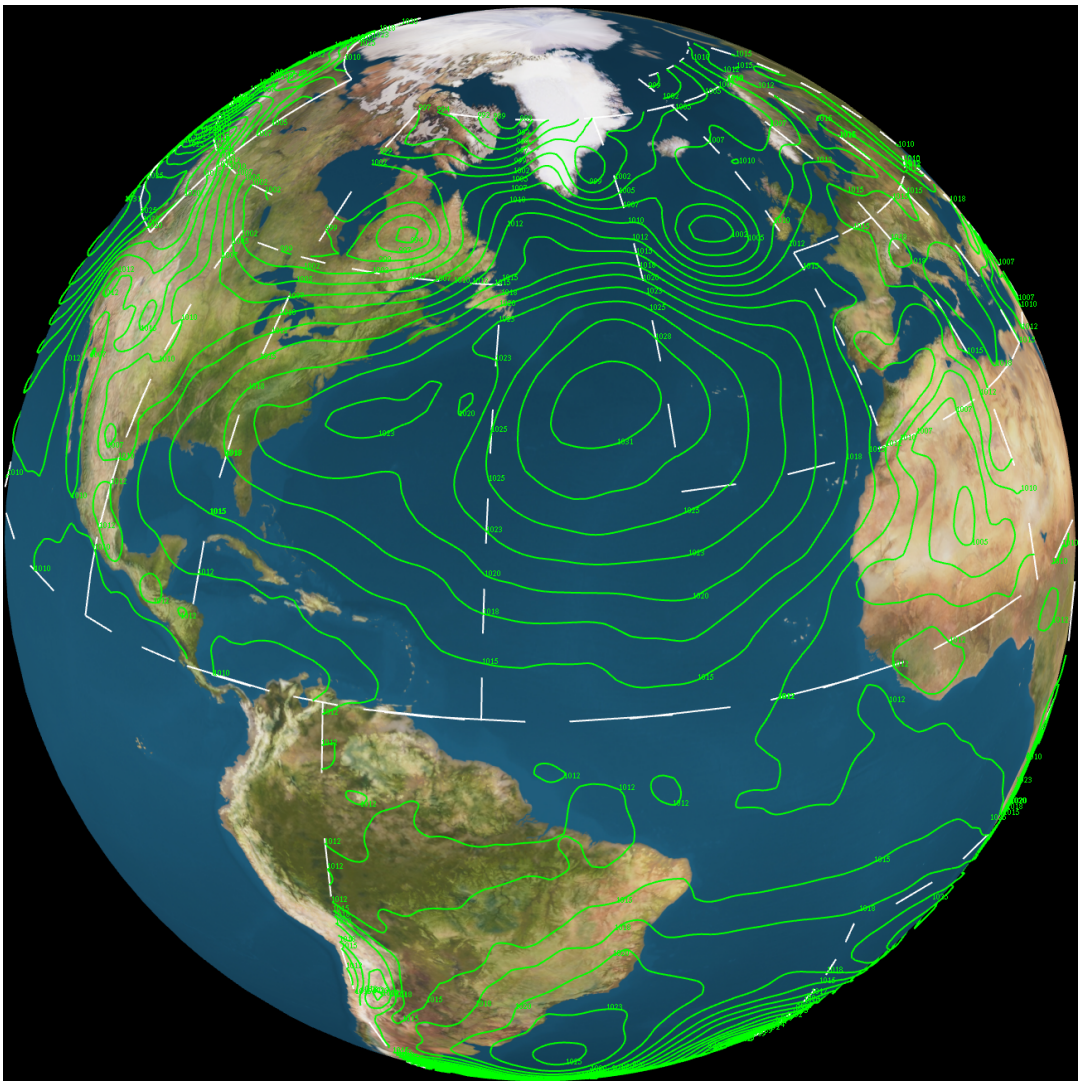


Abbildung 15: Darstellung des Luftdrucks durch Isolinien

6.2 Temperatur

Die Temperatur stellt eine weitere wesentliche Größe für die Wetter- bzw. Klimaentwicklung dar. So bewirken Temperaturunterschiede durch aufsteigende Warmluft oder durch absinkende Kaltluft thermische Tief- und Hochdruckgebiete [Walch]. Ferner spielt die Betrachtung der Temperatur in der Klimatologie aufgrund ihrer Auswirkung auf das Polkappenschmelzen und als Indiz für den klimatischen Wandel eine tragende Rolle.

Wie der Luftdruck ist auch die Temperatur ein skalarer Wert. Zur Darstellung von unterschiedlichen Temperaturen werden diese in der Regel auf Farben abgebildet, da der Mensch von Natur aus bestimmten Temperaturen eine Farbe zuordnet. So ist eine hohe Temperatur aufgrund der Assoziation zur Farbe des Feuers meist rötlich, während kalt dagegen meist mit Blau, der Farbe des Eises, assoziiert wird. Dies wird in der Regel bei der Darstellung der Temperaturwerte berücksichtigt. Isolinien würden sich prinzipiell dazu eignen eine Temperatur darzustellen. Weil bei der Temperaturdarstellung im Vergleich mit der Luftdruckdarstellung nicht primär das Gefälle, sondern gerade die absoluten Werte interessant sind, wird zumeist die gesamte Erdoberfläche eingefärbt. Außerdem wären unterschiedlich eingefärbte dünne

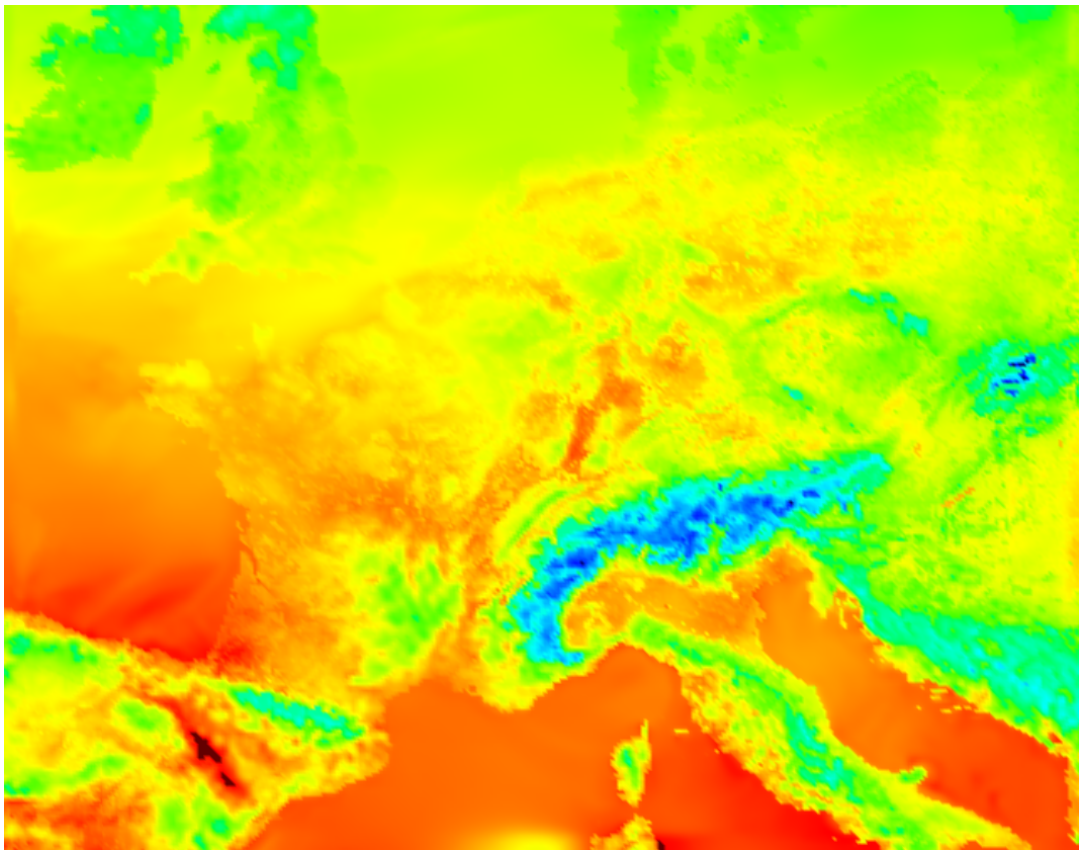


Abbildung 16: Darstellung hochaufgelöster Temperaturdaten in Europa durch Farbverläufe

Isolinien nur schwer zu erkennen bzw. zu unterscheiden. Nachteil dieser Technik ist die schlechte Kombinierbarkeit mit anderen Verfahren, die ebenfalls die gesamte Erdoberfläche einfärben. Einzige Lösung wäre das Übereinanderlegen von transparenten Ebenen. Dadurch würden aber die Farben vermischt, wodurch eine eindeutige Bestimmung nicht mehr möglich ist. Aus diesem Grund wird keine andere Wetterausprägung über die Farbe visualisiert. Zum Einfärben der Erdoberfläche sind zwei verschiedene Verfahren untersucht worden:

Farbverläufe Eine Möglichkeit der Oberflächeneinfärbung ist eine Abbildung aller Werte auf Farben eines bestimmten Spektrums. Anschließend werden die Flächen zwischen den Punkten triangulisiert und mit Farbverläufen eingefärbt. Vorteile des Verfahrens sind eine genaue Wiedergabe der Daten und durch die Ausführung auf der Grafikkarte eine hervorra-

gende Performance. Außerdem kann auf diese Weise bei hohen Datenauflösungen eine außerordentliche Bildqualität erreicht werden (vergl. Abb.16). Bei gröberen Auflösungen dagegen können sich benachbarte Messwerte stärker unterscheiden. Die Folge sind wenig klare Farbverläufe, welche für den Betrachter nur schwer zu interpretieren sind (siehe Abb.17 links).

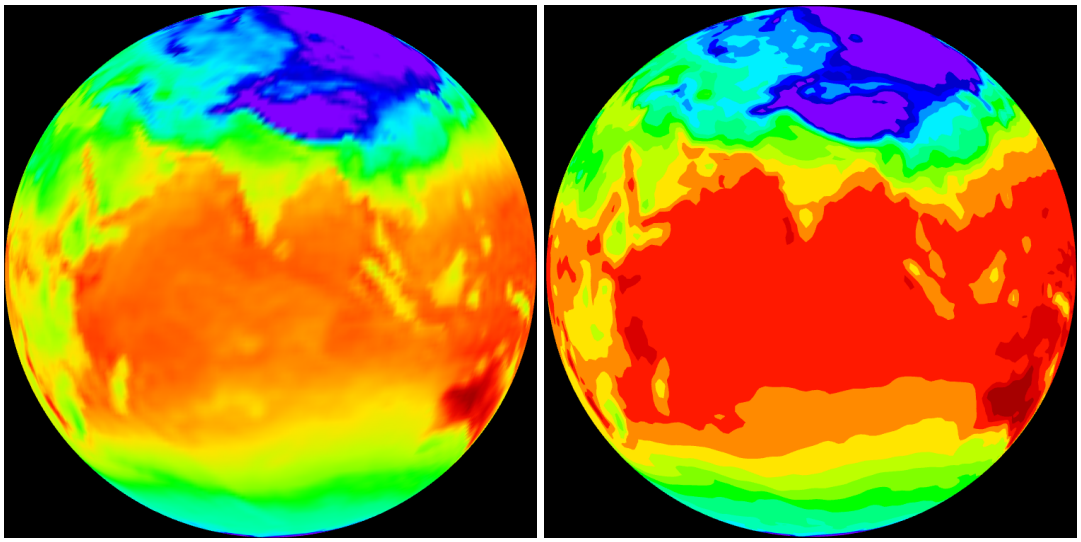


Abbildung 17: Temperaturdarstellung durch Farbverläufe (links) und Isoflächen (rechts)

Isoflächen entstehen durch Abbildung bestimmter Temperaturintervalle auf diskrete Farben (Abb.17 rechts). Dies geschieht durch Einfärbung der durch jeweils zwei Isolinien I_n und I_{n+1} umschlossenen Erdoberflächenausschnitte mit einer Farbe. Isoflächen wirken auf den Betrachter wesentlich übersichtlicher und verständlicher als die Farbverläufe (Abb.17 links). Allerdings ist mit der Verwendung dieses Verfahrens ein Genauigkeitsverlust verbunden, da im Inneren einer Fläche nicht mehr der exakte Wert feststellbar ist. Dieses Manko kann allerdings zum Teil durch den interaktiven Ansatz des Programms relativiert werden, der die numerische Ausgabe des Wertes des durch den Mauszeiger anvisierten Punktes ermöglicht. Die bislang getesteten Algorithmen ermöglichen keine Glättung der Isoflächenränder (siehe Kapitel 10) und liefern keine zufriedenstellende Performance, weshalb die Isoflächendarstellungstechnik in der aktuellen Version des Programms nicht zum Einsatz kommt.

6.3 Wind

Der Wind entsteht durch Druckausgleich und ist somit eine Auswirkung der Luftdruck- und indirekt auch der Temperaturverhältnisse auf der Erde [Walch]. Die Windvorhersage ist vor allem für die Schifffahrt aufgrund der Auswirkungen auf den Seegang von großer Bedeutung. Darüber hinaus werden Windvorhersagen benötigt, um vor stärkeren Stürmen Absicherungsmaßnahmen treffen zu können.
Die Windgeschwindigkeit ist ein zwei- oder dreidimensionaler Vektor, welcher den durch die Luft pro Zeiteinheit im Raum zurückgelegten Weg beschreibt. Für die Visualisierung des Windes wurden zwei Verfahren untersucht.

6.3.1 Windsymbole

Eine Möglichkeit der Windvisualisierung ist die Verwendung von Symbolen, die der Windrichtung entsprechend ausgerichtet werden (siehe Abbildung 18). Die Windstärke wird durch

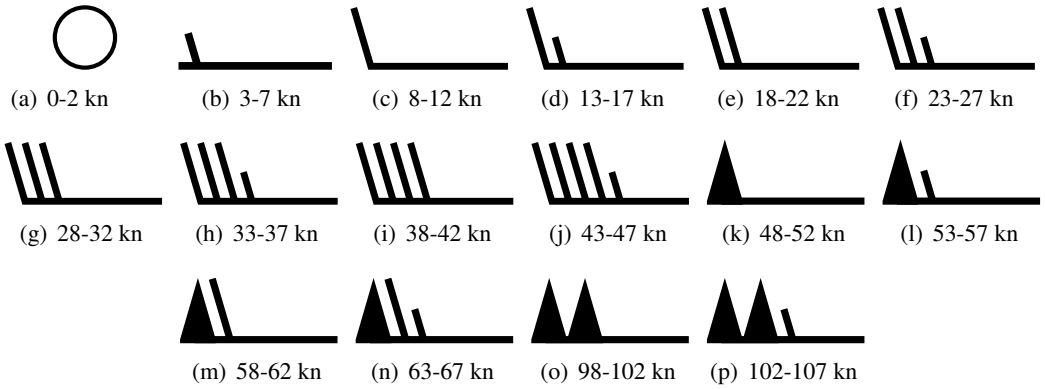


Abbildung 18: Windsymbole mit Angabe der Geschwindigkeit in Knoten

eine Kombination mehrerer Striche und Dreiecke gekennzeichnet, die am Ende der Windherkunftsrichtung eingezeichnet werden. Dabei entspricht ein halber Strich fünf Knoten, ein ganzer zehn und ein Dreieck 50 Knoten. Ergebnis der Summation über die Werte dieser Symbole ist der Betrag der Windgeschwindigkeit. Diese Darstellungstechnik ist zwar bei der professionellen Visualisierung des Wetters weit verbreitet, besitzt aber in den verbreiteten Fernsehwetterberichten nahezu keine Relevanz, weshalb ein Großteil der potentiellen Nutzer sie eventuell nicht versteht. Entschärft werden könnte das Problem durch die zusätzliche Angabe einer Legende. Ein weitaus größeres Problem ist die Abbildung der Windstärke auf diskrete Symbole, weshalb keine zeitlich stetige Animation zu erreichen ist. Folglich ist diese Darstellungstechnik nur für statische Wetterkarten, wie sie beispielsweise in SVG Weather [Kunze] Verwendung finden, akzeptabel (siehe Abbildung 19).

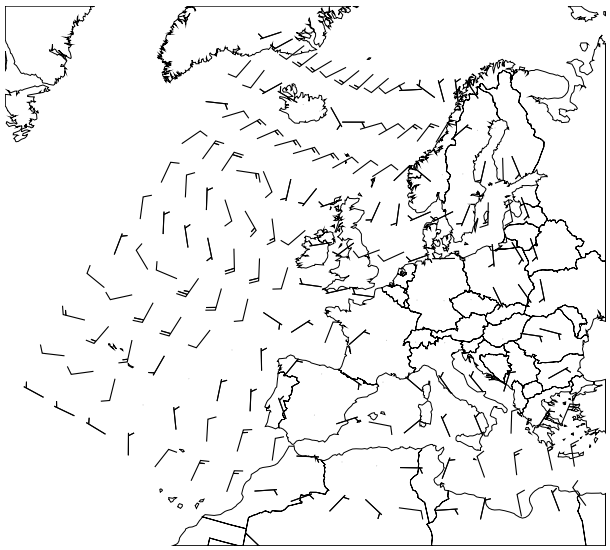


Abbildung 19: Winddarstellung durch Symbole in SVG Weather [Kunze]

6.3.2 Windpfeile

Eine weitere Möglichkeit der Windvisualisierung ist die Verwendung von Pfeilen, die der Windrichtung entsprechend ausgerichtet werden und deren Längen proportional zur Windstärke sind. Eine solche Darstellung kann zeitlich stetig animiert werden und ist intuitiv verständlich. Allerdings ist die Abschätzung der absoluten Windgeschwindigkeit nur schwer möglich. Ein weiteres Problem ist die ungleichmäßige Dichte mit der die Werte auf der Erdoberfläche angeordnet sind. So ist die Wertdichte beispielsweise in Polnähe wesentlich höher als beim Äquator. Daher müssten die maximalen Pfeillängen auf die maximale Länge der Pfeile im Polbereich normiert werden. Dies würde jedoch in sehr kurzen Pfeilen resultieren, die für den Betrachter kaum zu erkennen geschweige denn zu interpretieren wären. Alternativ könnte die Pfeildichte auf der Erdoberfläche durch geeignete Interpolation näherungsweise konstant gehalten werden. Nachteil dieses Ansatzes ist die Abbildung einer stark variierenden Anzahl von Werten auf jeweils ein Windsymbol. Abbildung 20 zeigt die Winddarstellung durch Pfeile, deren Dichte auf der Erdoberfläche in etwa konstant ist. Das dazu verwendete Verfahren ist in [Wenke] beschrieben.

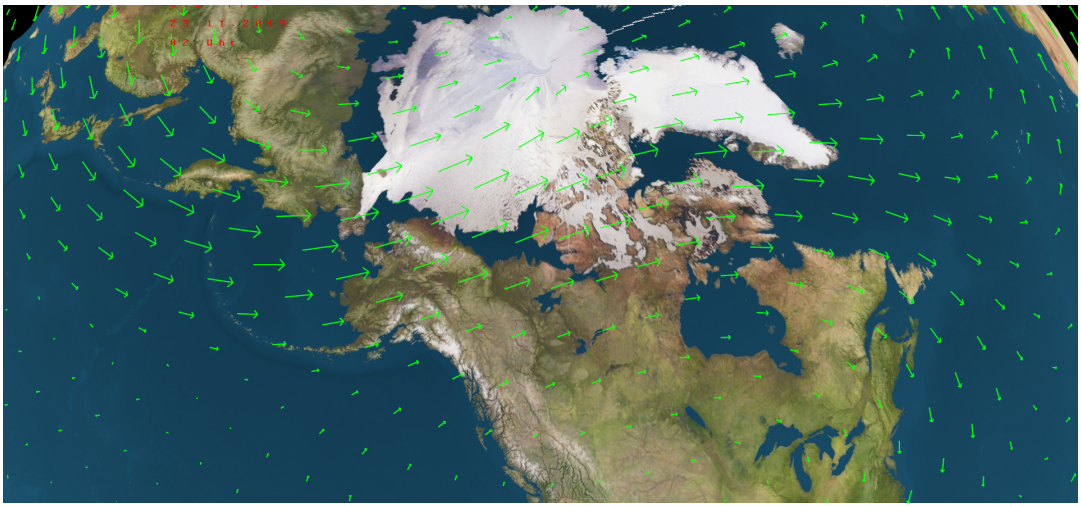


Abbildung 20: Winddarstellung durch Pfeile

6.3.3 Bewertung

Die durch ein Windsymbol dargestellte Windgeschwindigkeit kann nicht animiert werden. Darüber hinaus benötigt der Betrachter sehr lange, um sich einen Überblick über die Windgeschwindigkeiten zu verschaffen, da er diese zunächst ”berechnen” muss. Windpfeile dagegen können animiert werden und zumindest das Ablesen der relativen Windgeschwindigkeiten ist unmittelbar möglich. Demnach eignen sich die Windpfeile besser zur Veranschaulichung der relativen Geschwindigkeiten in der Animation, während die Windsymbole eher zur Darstellung der absoluten Werte in statischen Karten geeignet sind. Denkbar wäre daher die Verwendung der Windpfeile während der Animationslaufzeit und ein Einsatz der Windsymbole, sobald die Animation pausiert wurde. Alternativ könnte dem Benutzer die Wahl der Visualisierungsmethode überlassen werden. Insgesamt erscheinen jedoch alle Ansätze aufgrund der beschriebenen Probleme zur Verwendung noch nicht ausgereift genug.

6.4 Weitere Daten

Von den beschriebenen Wetterelementen abgesehen existieren noch weitere wie Bewölkungsgrad, Luftfeuchtigkeit und Niederschlag. Eine Betrachtung ist jedoch aus Zeitgründen und mangels entsprechender Daten ausgeblieben. Die Bewölkung kann beispielsweise über Grautöne mit einer zum Bewölkungsgrad proportionalen Opacity visualisiert werden. Abbildung 21 zeigt das Ergebnis und das dazu verwendete Verfahren ist in [Wenke] beschrieben.

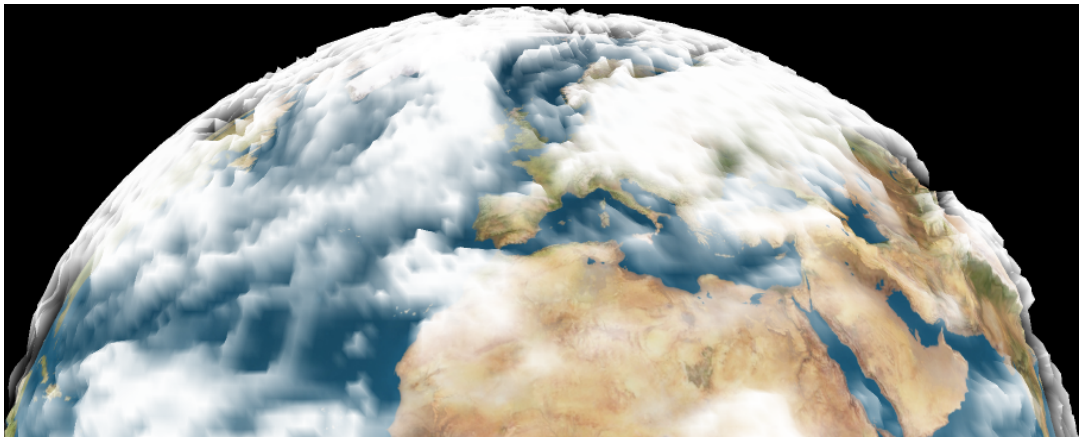


Abbildung 21: Wolkendarstellung durch transparente Grauschleier

7 Isoliniendarstellung

Gegenstand dieses Kapitels ist die Erläuterung verschiedener lokaler Verfahren zum Extrahieren von Isolinien bzw. Isoflächen aus den Ausgangsdaten. Isolinien verbinden Punkte gleichen Wertes in einem skalaren Feld. Diese müssen zunächst durch Vektorisierung aus den zugrundeliegenden Daten extrahiert werden.

7.1 Elementare Contourplot Verfahren

Die zu visualisierenden Rasterdaten liegen in einem regelmäßigen rechteckigen Gitter vor, jeder Gitterpunkt hat folglich vier direkte Nachbarpunkte. Zur Vereinfachung wird davon ausgegangen, dass sich der Wert eines Parameters zwischen zwei direkt benachbarten Punkten linear verhält. Daher können für zwei Punkte durch lineare Interpolation Zwischenwerte

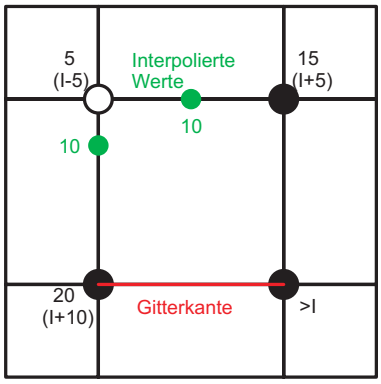


Abbildung 22: Interpolation auf Gitterkanten

errechnet werden, wie in Abbildung 22 dargestellt. Für diese und alle weiteren Abbildungen von Gitterzellenanordnungen gelten folgende Konventionen:

- Ein Kreis mit weißer Füllung symbolisiert einen Punkt mit einem Wert unterhalb des Isolevels.
- Ein schwarzer Kreis steht für einen Punkt mit einem Wert oberhalb des Isolevels. Anmerkung: Werden alle weißen Punkte unter Beibehaltung ihres Abstands zum Isolevel durch schwarze ersetzt und umgekehrt, entsteht ein identischer Linienverlauf.
- Ein Kreis mit roter Füllung repräsentiert einen Punkt dessen relative Lage zum Isolevel für diese Abbildung irrelevant ist.
- Zahlen neben Kreisen geben den Wert des jeweiligen Punktes an.
- Beschriftungen der Art $I + Zahl$ geben den Unterschied eines Punktes zum Isowert an.

Um die Daten zu vektorisieren gibt es verschiedene elementare Algorithmen, welche jeweils den Linien- bzw. Flächenverlauf in einer Gitterzelle lokal bestimmen. Von diesen werden einige im Folgenden näher untersucht und verglichen.

7.1.1 Marching Cubes Algorithmus

Das Marching Cube Verfahren [Lorensen] aus dem Jahre 1987 dient zur Visualisierung dreidimensionaler Daten mittels Isoflächen. Die Daten müssen als dreidimensionales skalares Punktfeld vorhanden sein, den so genannten Voxeln. Aus jeweils acht Voxeln wird ein Würfel definiert, wobei an den acht Ecken des Würfels die skalaren Werte eingetragen sind. Für einen vom Benutzer angegebenen Isowert wird untersucht, welche der Eckpunkte über bzw. unter dem Isowert liegen. Im nächsten Schritt wird berechnet, wo Schnittpunkte auf den Würfelkanten zu finden sind. Mit diesen Schnittpunkten werden einzelne Flächen gebildet. Insgesamt werden 256 verschiedene Anordnungen unterschieden, die durch Rotation oder Symmetrie auf 15 Grundtypen zurückgeführt werden können (Abbildung 23). Damit die

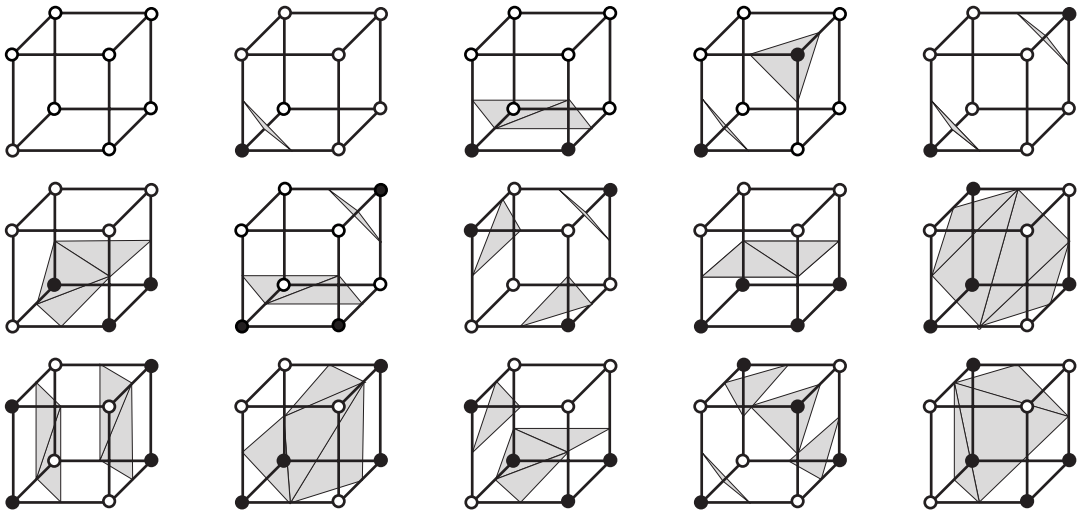


Abbildung 23: 15 Grundtypen beim Marching Cube Algorithmus

Analyse eines Würfels sehr schnell erfolgen kann, werden die 256 verschiedenen Anordnungen in Tabellen gespeichert und für jeden Würfel kann sofort die Lage der enthaltenen Flächen ermittelt werden. Diese Vorgehensweise wird für jeden einzelnen Würfel wiederholt. Der Algorithmus marschiert demnach durch die einzelnen Würfel, wodurch sich auch der Name des Verfahrens erklärt. Das Marching Cube Verfahren wird unter anderem verwendet, um medizinische Daten zu visualisieren, die beispielsweise bei einer Computertomographie gewonnen werden (Abbildung 24), um dreidimensionale Laserscans darzustellen oder um beliebige andere dreidimensionale Messwerte darzustellen.

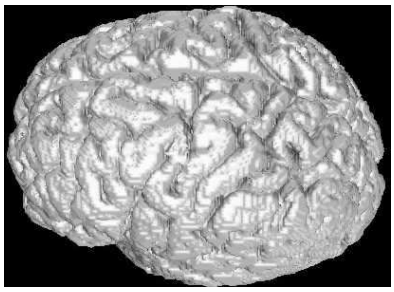


Abbildung 24: Visualisierung eines Gehirns aus 128984 Voxeln [Lingrand]

7.1.2 Marching Square Algorithmus

Das Marching Square Verfahren [Lorensen] stellt einen Spezialfall des Marching Cubes Algorithmus dar und dient zur Visualisierung zweidimensionaler Daten mittels Konturlinien. Die Daten müssen als zweidimensionales skalares Punktfeld vorhanden sein. Aus jeweils vier Punkten wird ein Quadrat definiert, wobei an den vier Ecken der Gitterzelle die skalaren Werte eingetragen sind. Für einen gegebenen Isowert wird untersucht, welche der Eckpunkte über und unter dem Isowert liegen. Liegt bei einer Kante ein Wert über und einer unter dem Isowert, so existiert ein Schnittpunkt, der durch lineare Interpolation bestimmt wird. Durch Verbinden dieser Punkte entstehen 6 verschiedene Linientypen, aus denen sich durch alle möglichen Kombinationen insgesamt 16 verschiedene Anordnungstypen ergeben (Abbildung 25). Zwei Fälle sind allerdings nicht eindeutig. Bei Fall 5 und Fall 10 ist nicht klar, wie

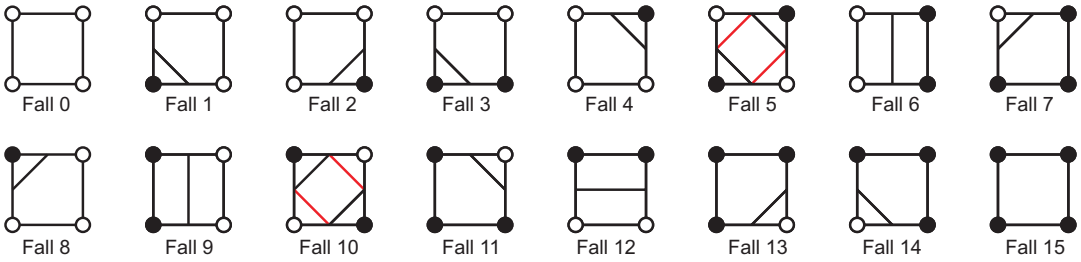


Abbildung 25: 16 Fälle beim Marching Square Verfahren

die Linie gezogen werden soll (Abbildung 26). Entweder entsteht eine zusammenhängende Konturlinie oder zwei einzelne. Da keine eindeutige Zuordnung möglich ist, muss festgelegt werden, wie in einem solchen Fall vorgegangen werden soll.
Auf diese Weise werden alle Gitterzellen für einen Isowert der Reihe nach abgearbeitet. Nachdem dies vollendet ist erhält man einzelne unzusammenhängende Liniensegmente der zugehörigen Konturlinien. Da eine Kante im Gitterinneren zu genau zwei Gitterzellen gehört, wird für beide der gleiche Schnittpunkt bestimmt. Aus diesem Grund enden die Liniensegmente benachbarter Gitterzellen im gleichen Punkt und die Konturlinien weisen keine Lücken auf. Anschließend läuft das Verfahren für den nächsten Isowert durch.

Abbildung 26: Nicht eindeutige Fälle beim Marching Square Verfahren

Entscheidungskriterien im zweideutigen Fall In dieser Arbeit sind zwei Entscheidungskriterien für die Wahl des Linienverlaufs im zweideutigen Fall zu beachten, welche im Folgenden erläutert werden:

I. Darstellungsqualität zu einem einzelnen Zeitpunkt:

Als Maß für die Ansehnlichkeit könnte man beispielsweise die Summe der erzeugten Konturlinienlängen verwenden. Dann könnte jeweils der Fall mit dem geringeren Wert dieser Summe gewählt werden, da dieser für den Betrachter korrekter wirkt (siehe Abb. 27).

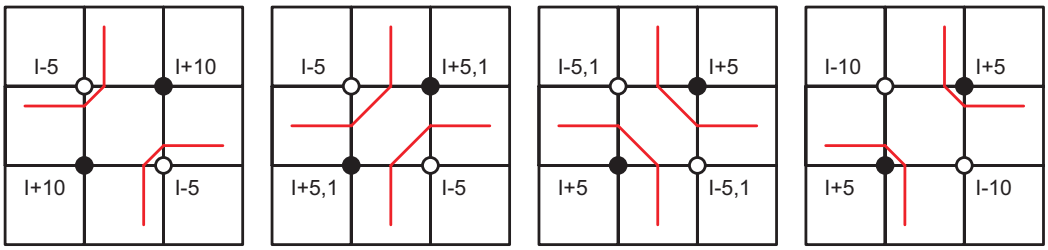
(a) Kurze Linien ergeben ein ansehnliches Ergebnis

(b) Lange Linien sind unansehnlicher

Abbildung 27: Statische Bestimmung des Linienverlaufs in einem zweideutigen Fall

II. Stetige zeitliche Änderung der Linien:

Auf die zuvor beschriebene Weise kann lediglich für den aktuellen Zeitpunkt eine befriedigende Lösung gefunden werden. Allerdings sind beim zeitlichen Verlauf sehr starke Sprünge möglich, da der Linienverlauf in einer Gitterzelle von einem Zeitpunkt zum nächsten fast invertiert werden kann (Siehe Abb. 28(b) und Abb. 28(c)). Eine denkbare Alternative wäre das



(a) Der Isolevel ist den weißen Punkten deutlich näher

(b) Der Isolevel ist den weißen Punkten noch minimal näher

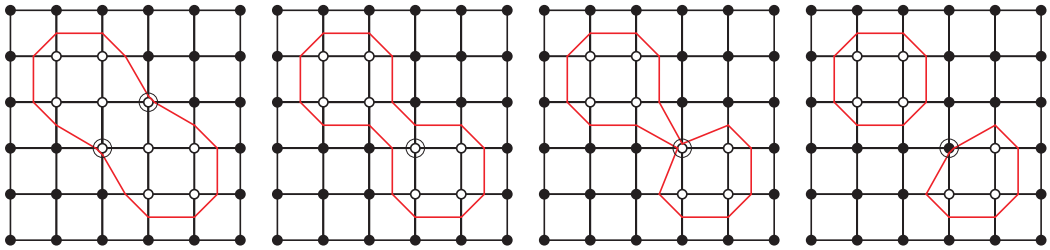
(c) Einen Schritt weiter sind die alternativen Linien kürzer

(d) Der Isolevel ist den schwarzen Punkten deutlich näher

Abbildung 28: Animationsproblem bei Minimierung der Gesamtlinienlänge.

Festhalten an der Art des initialen Linienverlaufs. Jedoch kann hier beim Übergang zu einem eindeutigen Fall ein Sprung entstehen, wie im Folgenden an einem Beispiel verdeutlicht wird. In der Ausgangssituation (siehe Abbildung 29(a)) ist der Linienverlauf noch eindeutig. Beim Übergang zum zweideutigen Fall kann durch geeignete Wahl des Linienverlaufs ein Sprung verhindert werden (siehe Abbildung 29(b)). Das Festhalten an dieser Art des Linienverlaufs ermöglicht eine stetige zeitliche Änderung der Konturlinien, solange der zweideutige Fall

bestehen bleibt (siehe Abbildung 29(c)). Wird der zweideutige Fall verlassen, ist der Linienverlauf wieder eindeutig festgelegt und es kann zu einem Sprung kommen. Dies ist in den Abbildungen 29(c) und 29(d) zu sehen, die den Linienverlauf unmittelbar vor und direkt nach dem Übergang zum eindeutigen Fall zeigen. Da keine Wahlmöglichkeit mehr bleibt, ist ein Verhindern des Sprungs nicht erreichbar. Folglich vermag der Marching Square Algorithmus eine zeitlich stetige Änderung des Linienverlaufs nicht zu garantieren.



(a) Ausgangssituation: Der Linienverlauf ist noch eindeutig

(b) Festlegen des Linienverlaufs im zweideutigen Fall

(c) Kurz vor Verlassen des zweideutigen Falls

(d) Übergang zum eindeutigen Linienverlauf erzwingt Sprung

Abbildung 29: Problem der Animation des Linienverlaufs im zweideutigen Fall. Es ändern sich jeweils nur die Werte der eingekreisten Punkte.

Bewertung Der Marching Square Algorithmus stellt ein außerordentlich effizientes Verfahren zur Berechnung von Konturlinien dar. Aufgrund des undefinierten Verhaltens im zweideutigen Fall kann eine stetige zeitliche Änderung der Konturlinien nicht garantiert werden. Dieses Problem ist durch den Marching Square Algorithmus nicht lösbar.

7.1.3 CONREC Algorithmus

Beispielhaft für ein dem Marching Square Algorithmus sehr ähnliches Verfahren wird der CONREC-Algorithmus [Bourke] von Paul Bourke aus dem Jahr 1987 vorgestellt. Dieser betrachtet ebenfalls jeweils vier rechteckig angeordnete Gitterpunkte gleichzeitig. Dabei wird zusätzlich zu den Eckpunkten A, B, C, D der Mittelpunkt M eingeführt, dessen Wert m gleich dem ungewichteten Mittelwert der Werte a, b, c, d der entsprechenden Eckpunkte ist. Anschließend wird das Rechteck in vier Dreiecke unterteilt, indem zwei Diagonalen von Eckpunkt zu Eckpunkt durch den Mittelpunkt gezogen werden, wie in der Abbildung 30 zu sehen

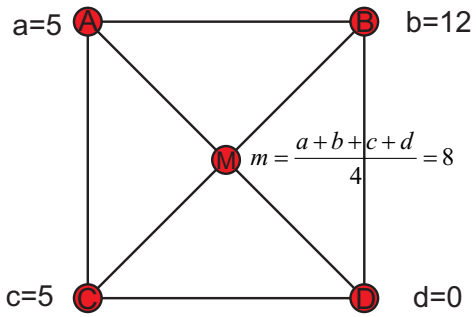


Abbildung 30: Triangularisierung im CONREC Algorithmus

ist. Analog zum Marching Square Algorithmus ergeben sich 16 Grundtypen von Gitterzellen, je nachdem welche Eckpunkte der Gitterzelle über oder unter dem Isowert liegen. Auf den entsprechenden Kanten wird ebenfalls ggf. durch lineare Interpolation ein Schnittpunkt berechnet. Das äußere Verhalten der Gitterzellen ist folglich mit dem des Marching Square Algorithmus identisch, lediglich der Linienverlauf im Inneren ist unterschiedlich. Dafür wird zusätzlich der Mittelwert mit dem Isolevel verglichen und ggf. auf den inneren Kanten weitere Schnittpunkte erzeugt, sodass es insgesamt 30 verschiedene Gitterzellentypen gibt (siehe Abbildung 31).

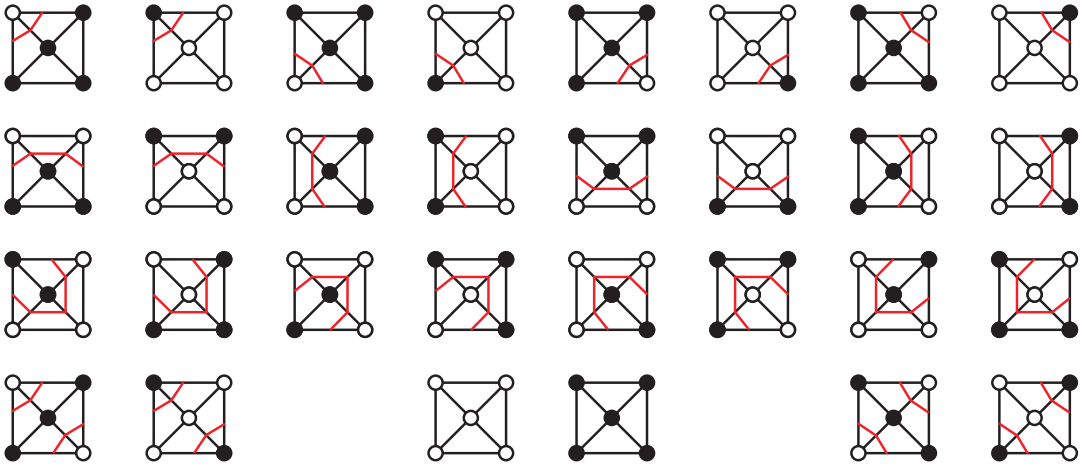
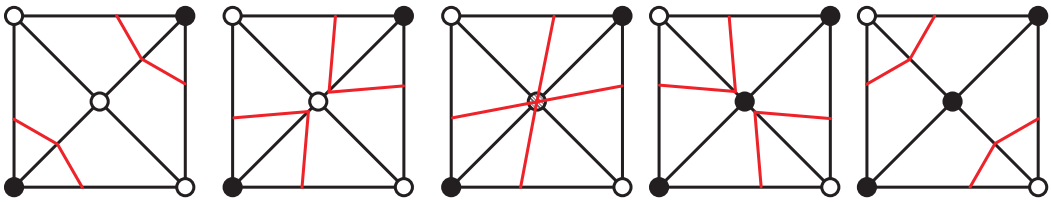


Abbildung 31: Die verschiedenen Fälle beim CONREC Algorithmus

Lösung des zweideutigen Falls Der CONREC Algorithmus löst das Problem des zweideutigen Falls des Marching Square Algorithmus, indem beide möglichen Linienverläufe stetig ineinander übergehen können. Dies wird am Beispiel eines Falles erläutert, der beim Marching Square Algorithmus nicht eindeutig lösbar wäre (siehe Abbildung 32). Darin ist der Linienverlauf zu jedem Zeitpunkt durch den mittleren Wert eindeutig festgelegt. In diesem Beispiel hat der mittlere Punkt anfänglich einen geringeren Wert als der Isolevel und wird als einziger Punkt seinen Zustand ändern und alle anderen werden ihn beibehalten. Nähert sich der Mittelwert dem Isolevel, krümmen sich die Linien zur Mitte (siehe Abb.32(a) und 32(b)). Wenn der Mittelwert mit dem Isolevel identisch ist, treffen die Linien im Mittelpunkt aufeinander (32(c)). Nach der Zustandsänderung teilen sich die Liniensegmente so auf,



(a) $m \ll Iso$ (b) $m < Iso$ (c) $m = Iso$ (d) $m > Iso$ (e) $m \gg Iso$

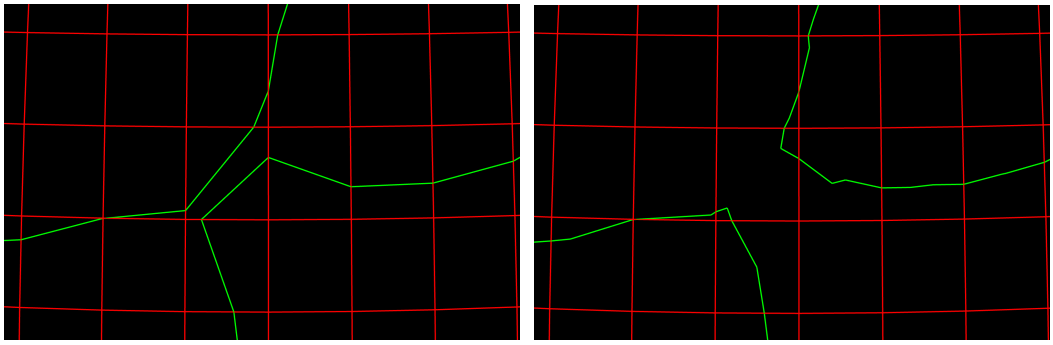
Abbildung 32: Veranschaulichung der stetigen zeitlichen Änderung des Linienverlaufs. Entscheidend ist hier das Verhältnis des Mittelwerts (m) zum Isowert (Iso).

dass die entstehenden Linien dem alternativen Linienverlauf des Marching Square Algorithmus entsprechen (Abb. 32(d) und 32(e)). Dabei werden bevorzugt nahe beieinanderliegende Schnittpunkte verbunden, sodass zu jedem Zeitpunkt eher kurze Linien entstehen. Folglich erfüllt der CONREC Algorithmus beide für den Marching Square Algorithmus geforderten Entscheidungskriterien im zweideutigen Fall.

Nachteile Der CONREC Algorithmus bestimmt den Linienverlauf innerhalb einer Gitterzelle unter Auswertung aller Eckpunkte. Dabei werden die Schnittpunkte auf den Kanten im Inneren durch lineare Interpolation zwischen einem Eckpunkt und dem Mittelwert bestimmt. Dieses ungebräuchliche Interpolationsverfahren zeigt einige Ungereimtheiten bezüglich der Gewichtung der Eckpunkte:

- Jeder Eckpunkt wird auf der Kante zum Mittelpunkt überaus stark gewichtet, da zwischen ihm und dem Mittelwert, in dem er ebenfalls enthalten ist, interpoliert wird.
- Die Gewichtungsänderung im Mittelpunkt ist sehr unstetig.
- Bei der Interpolation auf einer Diagonalenhälfte ändern sich auf dem Weg zum Mittelpunkt, von der Gewichtung des direkt anliegenden Eckpunktes abgesehen, alle anderen Gewichtungen gleich. Dabei ändert sich der räumliche Abstand zum Punkt auf der zweiten Diagonalenhälfte bei weitem stärker als die Abstände zu den anderen beiden Eckpunkten.

Aufgrund dieses Interpolationsverfahrens sehen die durch den CONREC Algorithmus erzeugten Linien (siehe Abbildung 33(b)) trotz höherer Vertexzahl nicht besser aus als die durch den Marching Square Algorithmus erzeugten (Abb. 33(a)). Die Punkte im Inneren



(a) Marching Square (b) CONREC

Abbildung 33: Vergleich der durch CONREC und Marching Square Algorithmus erzeugten Konturlinien

der Gitterzellen erzeugen des Öfteren Treppeneffekte, anstatt sich stimmig in den Linienverlauf einzufügen. Lediglich der beim Marching Square Algorithmus zweideutige Fall ist beim CONREC Algorithmus besser gelöst.

Bewertung Der CONREC Algorithmus löst das Problem des zweideutigen Falls des Marching Square Algorithmus und ermöglicht eine stetige Animation. Trotz der unter Auswertung aller Eckpunkte bestimmten Krümmung der Linien innerhalb der Gitterzellen wird die

statische Darstellungsqualität aufgrund des ungeeigneten Interpolationsmechanismus nicht verbessert. Zusätzlich ist die Berechnung einer größeren Anzahl von Schnittpunkten mit einem Performanceverlust verbunden. Das asymptotische Verhalten gleicht allerdings bis auf einen konstanten Faktor dem des Marching Square Algorithmus.

7.1.4 CONREC Algorithmus mit bilinearer Interpolation

Die Schwächen des CONREC Algorithmus können durch den Einsatz eines alternativen Interpolationsalgorithmus ausgeglichen werden. Dabei bleibt die Grundidee unverändert, lediglich die Schnittpunktbestimmung erfolgt anders.

Bilineare Interpolation Bei bilinearer Interpolation werden die Werte der vier Eckpunkte einer Gitterzelle in jeder Achse (u und v) jeweils linear interpoliert. Dazu werden zunächst die beiden Werte zwischen den oberen und den unteren Eckpunkten entlang der u-Achse linear interpoliert. Mit den Bezeichnungen aus Abbildung 34 gilt für den Wert eines Punktes auf $\overline{AB}$:

$$f_{AB}(u) = a + \frac{u}{w} \cdot (b - a) \tag{1}$$

Für einen Punkt auf $\overline{CD}$ gilt:

$$f_{CD}(u) = c + \frac{u}{w} \cdot (d - c) \tag{2}$$

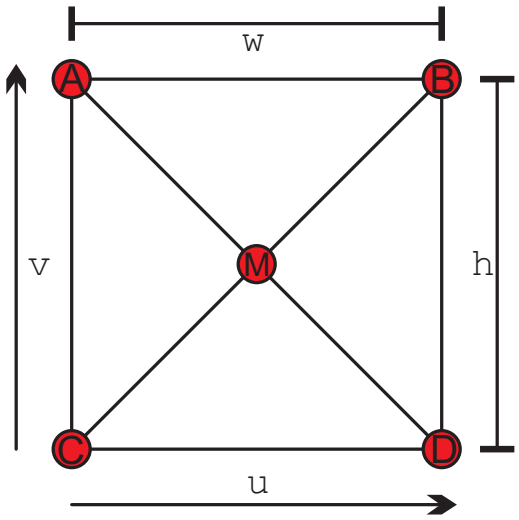


Abbildung 34: Bezeichnungen einer CONREC Gitterzelle

Dabei stehen a, b, c, d für die Werte in den entsprechenden Eckpunkten. Die so ermittelten Wertepaare werden anschließend entlang der v-Achse erneut linear interpoliert. Diese zweistufige lineare Interpolation gibt der bilinearen Interpolation ihren Namen. So ergibt sich für einen Punkt innerhalb des Rechtecks:

$$f_{ABCD}(u, v) = c + \frac{u}{w} \cdot (d - c) + \frac{v}{h} \cdot \left(a + \frac{u}{w} \cdot (b - a) - \left(c + \frac{u}{w} \cdot (d - c)\right)\right) \tag{3}$$

Dieses Verfahren ist sehr verbreitet und wird üblicherweise eingesetzt, um bei der Texturierung zwischen vier Texeln zu interpolieren. Aufgrund seiner elementaren Bedeutung wird die bilineare Filterung seit dem 1996 erschienenen Voodoo Graphics Chipsatz [Wp4] von jeder wesentlichen Grafikkarte in Hardware unterstützt. Wichtigste Eigenschaften:

- Symmetrie: Das Ergebnis ist unabhängig davon, ob zunächst in u und dann in v-Richtung interpoliert wurde oder umgekehrt.
- Der Mittelpunkt stellt genau den Mittelwert der Werte der Eckpunkte dar.
- Auf den äußeren Kanten wird genau wie bei den anderen Verfahren zwischen den beiden anliegenden Punkten interpoliert.

Für den CONREC Algorithmus müssen lediglich Punkte auf den Diagonalen bestimmt werden. Mit $\frac{u}{w} = \frac{v}{h} = \lambda$ folgt aus Gleichung 3 für die Diagonale $\overline{CB}$:

$$f_{CB}(\lambda) = c + \lambda \cdot (d - 2c + a) + \lambda^2 \cdot (b + c - a - d) \tag{4}$$

Analog ergibt sich die Diagonale $\overline{AD}$ aus Gleichung 3 mit $\frac{u}{w} = \lambda$ und $\frac{v}{h} = 1 - \lambda$ zu:

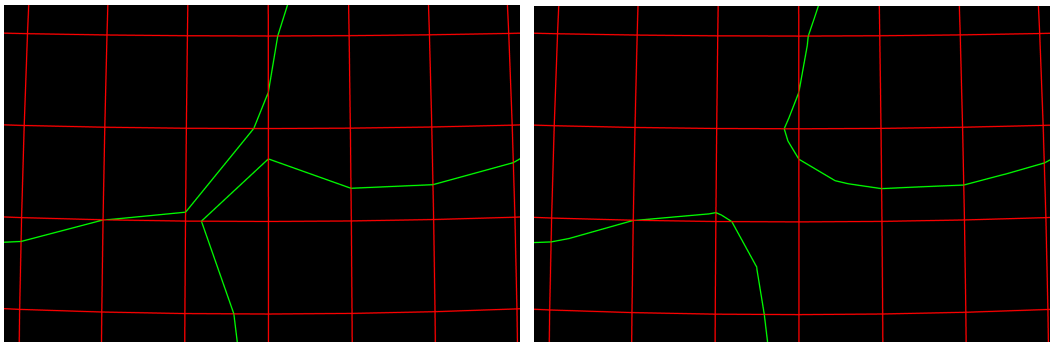
$$f_{AD}(\lambda) = a + \lambda \cdot (b - 2a + c) + \lambda^2 \cdot (a + d - b - c) \tag{5}$$

Bei 4 und 5 handelt es sich um stetige Gleichungen zweiter Ordnung. Diese lassen sich für einen gegebenen Isowert I_0 am besten mit der $P - Q - Formel$ lösen [Lampen], welche zwei Lösungen liefert, die nicht notwendigerweise reell und verschieden sind. Damit dieses Interpolationsverfahren beim CONREC Algorithmus verwendet werden kann, muss es genau dann auf einer Kante einen Schnittpunkt erzeugen, wenn das vom CONREC Algorithmus verwendete Verfahren einen erzeugt. Auf den äußeren Kanten einer Gitterzelle liefern beide Verfahren identische Ergebnisse. Bei einer Kante von einem Eckpunkt zum Mittelpunkt ist zu zeigen, dass die bilineare Interpolation genau dann genau einen Schnittpunkt erzeugt, wenn der beteiligte Eckpunkt und der Mittelpunkt einen unterschiedlichen Zustand haben. Der Beweis wird anhand der Kante $\overline{CM}$ geführt.
Befinde sich I_0 zwischen $c = f_{CB}(0)$ und $m = f_{CB}\left(\frac{1}{2}\right)$. Dann gibt es auf der Kante $\overline{CM}$ aufgrund der Stetigkeit der Gleichung 4 mindestens einen Schnittpunkt (Zwischenwertsatz [Forster]). Generell möglich wären $1 + n \cdot 2$ Schnittpunkte (mit $n = 0, 1, 2, ...$). Da eine quadratische Gleichung nicht mehr als zwei Lösungen haben kann, gibt es genau einen Schnittpunkt.
Befinde sich I_0 außerhalb von $c = f_{CB}(0)$ und $m = f_{CB}\left(\frac{1}{2}\right)$. Dann wären grundsätzlich $n \cdot 2$ Schnittpunkte möglich (mit $n = 0, 1, 2, ...$). Um die Nichtexistenz eines Schnittpunktes auf $\overline{CM}$ zu zeigen, wird die gesamte Kante $\overline{CB}$ betrachtet. Hier kann eine Fallunterscheidung bezüglich der Lage von B zum Isowert getroffen werden:

- 1. Fall: Liegt I_0 zwischen $b = f_{CB}(1)$ und $m = f_{CB}\left(\frac{1}{2}\right)$, muss es aufgrund der Stetigkeit der Funktion auf dem Abschnitt $\overline{MB}$ ebenfalls einen Schnittpunkt geben. Da für eine quadratische Gleichung nicht mehr als zwei Lösungen existieren, kann es auf $\overline{CM}$ keinen Schnittpunkt geben.
- 2. Fall: Sei nun I_0 außerhalb von $b = f_{CB}(1)$ und $m = f_{CB}\left(\frac{1}{2}\right)$. Dies ergibt sich, indem im Vergleich zum ersten Fall b verändert wird. Da b in die Gleichung 4 nur positiv eingeht, verändert sich $f_{CB}(\lambda)$ in gleicher Richtung. Folglich kann im Vergleich mit dem ersten Fall kein Schnittpunkt hinzu kommen.

Damit gibt es genau dann genau einen Schnittpunkt auf $\overline{CM}$, wenn sich I_0 zwischen $c = f_{CB}(0)$ und $m = f_{CB}\left(\frac{1}{2}\right)$ befindet. Aufgrund der Spiegelsymmetrie gilt der Beweis für die übrigen inneren Kanten ebenfalls. □

Ergebnis Die durch den CONREC Algorithmus mit bilinearer Interpolation erzeugten Linien (Abbildung 35(b)) sehen durch das geeignetere Interpolationsverfahren besser aus als die durch den Marching Square Algorithmus erzeugten (Abb. 35(a)). Der Algorithmus kann



(a) Marching Square (b) CONREC mit bilinearer Interpolation

Abbildung 35: Vergleich der durch CONREC und Marching Square Algorithmus erzeugten Konturlinien

mithilfe des vom CONREC Algorithmus geerbten Verhaltens den beim Marching Square Algorithmus zweideutigen Fall eindeutig lösen und ermöglicht so eine zeitlich stetige Animation. Außerdem gibt der geeignetere Interpolationsmechanismus die Daten besser wieder und die unter Auswertung aller vier Eckpunkte bestimmte Linienkrümmung innerhalb der Gitterzellen verbessert den visuellen Eindruck etwas. Einziger Nachteil sind die gegenüber dem CONREC Algorithmus um einen konstanten Faktor gestiegenen Kosten zur Bestimmung eines Schnittpunktes.

7.1.5 Abschließender Vergleich elementarer Contourplot Verfahren

Die beschriebenen Algorithmen werden aktuell lediglich für die Luftdruckdarstellung verwendet. Diese Werte ändern sich erfahrungsgemäß aufgrund der geringeren Abhängigkeit vom Sonnenstand und von den Meer-Kontinent Übergängen zeitlich- bzw. räumlich wesentlich weniger abrupt als beispielsweise Temperaturen. Durch die daraus resultierenden geringen Krümmungen der Linien bedingt unterscheiden sich die Ausgaben der Verfahren in der Praxis nicht signifikant. Außerdem tritt aus diesem Grund der zweideutige Fall beim Luftdruck nur sehr selten auf, weshalb die Sprünge bei subjektiver Betrachtung der Animation in der Praxis kaum auffallen. Die Performance leidet dagegen deutlich unter der erhöhten Anzahl der durch die CONREC Algorithmen erzeugten Schnittpunkte. Daher wird bei der Luftdruckdarstellung der performante Marching Square Algorithmus verwendet. Andere Daten könnten dagegen eventuell stärker von den Vorteilen des CONREC Algorithmus mit bilinearer Interpolation profitieren und sollten dementsprechend durch ihn visualisiert werden.

7.1.6 Anmerkungen zur Implementation des Marching Square Algorithmus

Eine Kante kann pro Isolevel maximal einen Schnittpunkt enthalten. Eine Datenstruktur zur Gitterzellenrepräsentation muss demnach vier ggf. undefinierte Schnittpunktindices, zwei ggf. undefinierte Datenstrukturen zur Linienrepräsentation und einen Index für den Gitterzelletyp enthalten. Ein Nachteil des Marching Square Algorithmus ist die Betrachtung von jeweils einer Gitterzelle zur Zeit. Daher wird jeder Schnittpunkt im Gitterinneren zweimal unabhängig berechnet und abgespeichert. Auf diese Weise wird nicht nur unnötig Rechenleistung und Speicher verschwendet, sondern auch das Auffinden geschlossener Linienzüge erschwert. Schließlich müssen gemeinsame Punkte der Linien benachbarter Gitterzellen erst gefunden werden. In dieser Arbeit wird daher ein modifizierter Marching Square Algorithmus verwendet. Dieser betrachtet sequentiell alle Kanten, berechnet ggf. Schnittpunkte und übergibt den zugehörigen Index den benachbarten Gitterzellen. Da zu keiner Zeit alle vier Eckpunkte betrachtet werden, ist ein besonderes Verfahren zur Bestimmung des Gitterzellentyps notwendig. Wie in Abbildung 25 auf Seite 33 ersichtlich ist, kann der Gitterzellentyp in Abhängigkeit der gesetzten Eckpunkte folgendermaßen bestimmt werden:

- 0 bei keinem Eckpunkt über dem Isolevel
- $1, 2, 4$ oder 8 bei genau einem gesetzten Eckpunkt
- Die Summe der Werte bei mehreren gesetzten Eckpunkten

Demnach wird der Gitterzellentyp zunächst auf Null gesetzt. Anschließend wird für jeden gesetzten Eckpunkt einer waagerechten Kante der Gitterzellentyp um den entsprechenden Wert erhöht.

7.1.7 Allgemeine Bewertung elementarer Contourplot Verfahren

Die beschriebenen Contourplot Verfahren sind sehr effizient, sowohl in der Zeichengeschwindigkeit als auch in der Speicherbelastung. Da jede Gitterzelle für jeden Isolevel jeweils mit konstantem Aufwand betrachtet werden muss, ergibt sich das Laufzeitverhalten in Abhängigkeit der Gitterzellenzahl n und Isolevelzahl m zu: $O(n \cdot m)$. Die Algorithmen können so modifiziert werden, dass nur Teile des Gitters eingelesen und verarbeitet werden müssen, da immer nur ein Rechteck aus vier Punkten betrachtet wird. Folglich könnten selbst beliebig große Datenraster mit geringem Speicheraufwand vektorisiert werden. Außerdem wäre bei entsprechender Hardware die parallele Verarbeitung beliebig vieler Gitterzellen möglich.
Es entstehen in der Ausgabe geschlossene Konturlinien, die allerdings nur aus Liniensegmenten bestehen und keine geschlossenen Polygone darstellen. Aus diesem Grund können sie nicht geglättet und beschriftet werden. Um zu verdeutlichen, für welchen Wert eine Isolinie steht, müsste jedes Liniensegment beschriftet werden, wodurch die Darstellung absolut unübersichtlich würde. Alternativ könnte der Wert über die Farbe dargestellt werden, was zu Problemen mit anderen Darstellungen führt, die ebenfalls Werte über Farben visualisieren. Insgesamt liefern die elementaren Contourplot Verfahren keine befriedigende Lösung, weshalb ein weiteres Verfahren benötigt wird, um aus den Teilstücken zusammenhängende Polygone zu bilden.

7.2 Linefollowing Algorithmen

William V. Snyder beschreibt im Jahr 1978, ausgehend von einem zweidimensionalen Array mit skalaren Werten und der Angabe von Isowerten, allgemein das Zeichnen von Konturlinien bzw. Isolinien [Snyder]. Er stellt dabei unter anderem eine Möglichkeit vor, um geschlossene Isolinien zu finden. Dabei werden Isolinien ausgehend von einem Startpunkt durch das gesamte Raster verfolgt, bis sie einen geschlossenen Linienzug bilden oder den Rand des Gitters schneiden.

7.2.1 Berechnung der Isolinien

Der Linefollowing Algorithmus von Snyder basiert auf einem zweidimensionalen Raster in Form eines Arrays und einer Liste von Isowerten. Der Algorithmus lässt sich in vier wesentliche Schritte einteilen:

1. Das Raster wird zeilenweise durchlaufen. Jede Kante einer Rasterzelle wird auf Schnittpunkten mit Isolinien untersucht. Liegt ein gesuchter Isowert zwischen den Werten zweier Rasterpunkte, existiert ein Schnittpunkt.

2. Falls eine Gitterkante mit einem gesuchten Isowert gefunden wurde, werden die Koordinaten des Schnittpunktes durch lineare Interpolation berechnet. Danach wird vermerkt, dass auf der Gitterkante der Isowert gefunden wurde, damit die Kante nicht erneut untersucht wird.

3. Der Algorithmus untersucht nun die zweite Rasterzelle, die an der Gitterkante mit dem gefundenen Isowert angrenzt, nach einer weiteren Gitterkante mit demselben Isowert. Ist die Kante gefunden, wird auch hier der Schnittpunkt wie in Schritt zwei berechnet und die Schnittpunkte werden mit einer geraden Linie verbunden. Dieser Schritt wird wiederholt ausgeführt bis keine unmarkierte Gitterkante mit dem Isowert gefunden wird (Abbildung 36). Der dafür nötige Aufwand für eine Gitterzelle ist somit linear von der Gitterauflösung abhängig.

4. Die Schritte eins bis drei werden je Isowert solange wiederholt bis alle Gitterkanten mit einem Schnittpunkt markiert sind. Es ergibt sich folglich ein proportional zum Quadrat der Messpunktzahl steigendes Laufzeitverhalten.

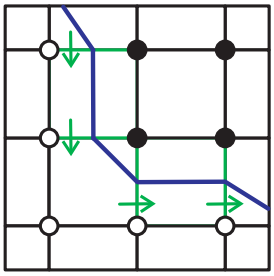


Abbildung 36: Funktionsweise eines Linefollowing Algorithmus

Insgesamt kann das Verfahren zur Erzeugung geschlossener Linienzüge verwendet werden. Allerdings ist das asymptotische Verhalten dem des Marching Square Algorithmus deutlich

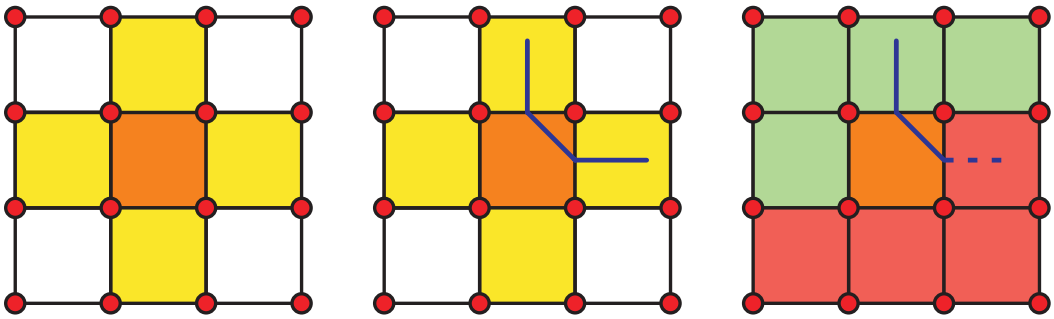
unterlegen, da die Kosten pro Isolevel quadratisch zur Anzahl der Gitterzellen bzw. Messwerte sind. Weil das Verfahren unabhängig für jeden Isolevel durchgeführt werden muss, ergibt sich die Laufzeit in Abhängigkeit der Gitterzellenzahl n und Isolevelzahl m zu: $O\left(n^2 \cdot m\right)$.

7.3 Marching Square SE

Um die beschriebenen Defizite des Marching Square Algorithmus zu beheben ist im Rahmen dieser Arbeit ein Verfahren entwickelt worden, mit dessen Hilfe echte Polygone auf Basis des Marching Square Algorithmus gebildet werden können. Dieser Algorithmus wird im Folgenden Marching Square SE genannt.

7.3.1 Arbeitsweise

Ausgangssituation der Erweiterung des Marching Square Algorithmus sind die Gitterzellen mit zusammenhängenden Liniensegmenten, die der Marching Square Algorithmus für einen Isolevel erzeugt hat. Diese enthalten einen Index für den Gitterzellentyp, zwei zunächst nicht initialisierte Linien und pro Kante einen ggf. undefinierten Schnittpunktindex. Jede Gitterzelle hat vier Kanten und damit vier direkte Nachbarn (siehe Abb. 37(a)). Des Weiteren können Methoden definiert werden, die in Abhängigkeit des Gitterzellentyps für jede in der Gitterzelle enthaltene Linie bestimmen, in welchen Nachbargitterzellen sie fortgesetzt wird (vergl. Abb. 37(b)). Aufgrund dieses Wissens entfällt eine aufwändige Suche nach einer Fortsetzung der Linie. Der Algorithmus verarbeitet die Gitterzellen zeilenweise von oben nach unten. In den einzelnen Zeilen werden die Gitterzellen von links nach rechts betrachtet. Bedingt durch diese Verarbeitungsrichtung sind die Gitterzellen links und über der jeweils aktuellen Gitterzelle ihre Vorgänger und zu diesem Zeitpunkt bereits verarbeitet (siehe Abb. 37(c)). Aus diesem Grund werden Vergleiche nur mit diesen Vorgängergitterzellen durchgeführt. Für jede in der jeweils aktuellen Gitterzelle enthaltene Linie wird in Abhängigkeit des Linientyps eine der folgenden Aktionen durchgeführt:



(a) Die vier direkten Nachbarn (gelb)

(b) Rekonstruktion des Linienverlaufs anhand des Gitterzellentyps

(c) Die bereits bearbeiteten Gitterzellen (grün) und die noch nicht betrachteten (rot)

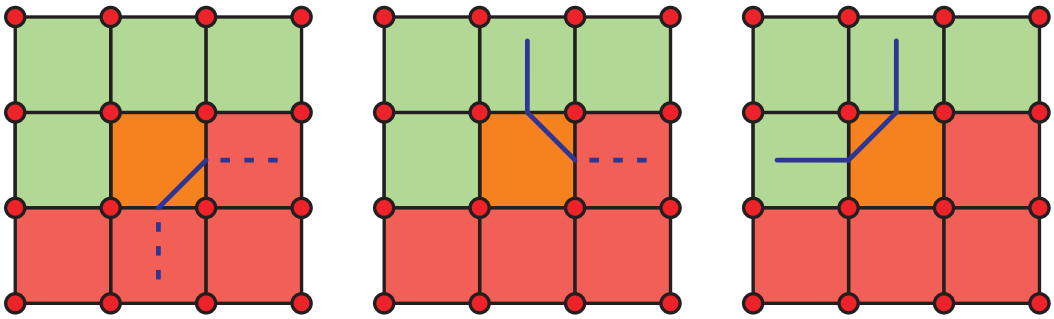
Abbildung 37: Nachbarschaftsbeziehungen der aktuellen Gitterzelle (orange)

Neue Linie Eine neue Linie muss immer dann erzeugt werden, wenn eine Gitterzelle ein Liniensegment enthält, welches keine gemeinsame Kante mit einer bereits bearbeiteten Gitterzelle hat und folglich nicht an eine existierende Linie angehängt werden kann

(siehe Abb. 38(a)). Eine neue Linie wird bei den Gitterzellentypen 2, 5 und 13 erzeugt. Wenn sich Gitterzellen anderen Typs am Rand des Gitters befinden, muss eventuell auch hier eine neue Linie erzeugt werden.

Id einhängen Enthält eine Gitterzelle ein Liniensegment, welches genau eine gemeinsame Kante mit einer bereits verarbeiteten Gitterzelle hat, so kann es an eine Linie in dieser Gitterzelle angehängt werden (siehe Abb. 38(b)). Unter Berücksichtigung der Typen der beteiligten Gitterzellen und ihrer Lage zueinander wird entschieden, an welchen Linienzug das Liniensegment anzuhängen ist. Abschließend wird ein Verweis auf die Linie an die aktuelle Gitterzelle übergeben und dort gespeichert. Eine Einfügeoperation in die obere Gitterzelle kommt bei den Gitterzellentypen 4, 6, 9 und 11 vor, sofern sie sich nicht am oberen Gitterrand befinden. Eine Gitterzelle des Typs 1, 3, 12 und 14 fügt einen Index in die links von ihr gelegene Gitterzelle ein, falls sie sich nicht am linken Rand befindet.

Linien vereinigen Enthält eine Gitterzelle eine Linie, welche zwei bereits untersuchte Gitterzellen verbindet, so muss hier der Linienzug geschlossen werden (siehe Abb. 38(c)). Enthalten die Nachbargitterzellen jeweils ein Ende der gleichen Linie, so muss diese lediglich geschlossen werden. Andernfalls handelt es sich um unabhängig voneinander erzeugte Linien, die vereinigt werden müssen, indem die eine an die andere angehängt wird. Eine Linienvereinigung wird bei Gitterzellen des Typs 5, 7 und 8 durchgeführt, sofern sie sich nicht am oberen oder linken Rand befinden.



(a) Erzeugen einer neuen Linie nötig

(b) Anhängen des Liniensegments an bestehende Linie

(c) Schließen einer Linie

Abbildung 38: In Abhängigkeit des Gitterzellentyps wird entschieden, wie mit der Linie der aktuellen Gitterzelle (blau) zu verfahren ist. Durchgezogene Linien aus fertigen Gitterzellen (grün) existieren bereits, gestrichelte Linien aus noch nicht verarbeiteten Gitterzellen (rot) existieren noch nicht.

7.3.2 Datenstruktur zur Linienrepräsentation

Eine Datenstruktur zur Beschreibung einer Linie muss die folgenden Linieneigenschaften unterstützen:

- Eine Linie hat genau zwei Enden.
- Die Anzahl der Elemente bei der Erzeugung ist zwei. Die Gesamtzahl ist zu diesem Zeitpunkt nicht zu bestimmen.

- An beiden Enden müssen nach Erzeugung Elemente eingefügt werden können.
- Der Algorithmus geht bei der Linienerzeugung zeilenweise durch das Gitter und folgt nicht dem Verlauf einer Linie. Daher müssen unabhängig voneinander erzeugte Teilstücke einer Isolinie nachträglich vereinigt werden können.

Es sind mehrere Lösungsansätze denkbar, die im Folgenden verglichen werden.

Gerichtete einfach verlinkte Liste
Diese minimalen Anforderungen werden durch eine einfach verlinkte Liste erfüllt, die allerdings noch um eine Methode zum Anhängen einer zweiten Linie erweitert werden muss. Bei dieser Operation ist zu beachten, dass eine Liste mit Head und Tail per Definition über eine Richtung verfügt. Diese entsteht durch die Reihenfolge in der die Indices eingehängt wurden (siehe Abb. 39). Treffen zwei unterschiedlich gerichtete Listen aufeinander, ist eine Verei-

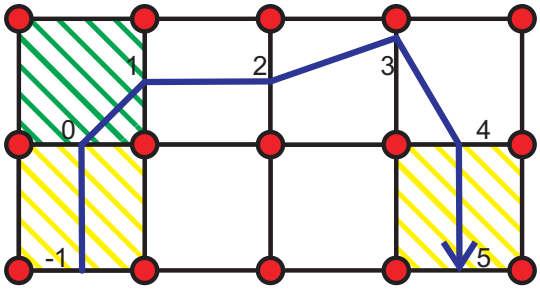
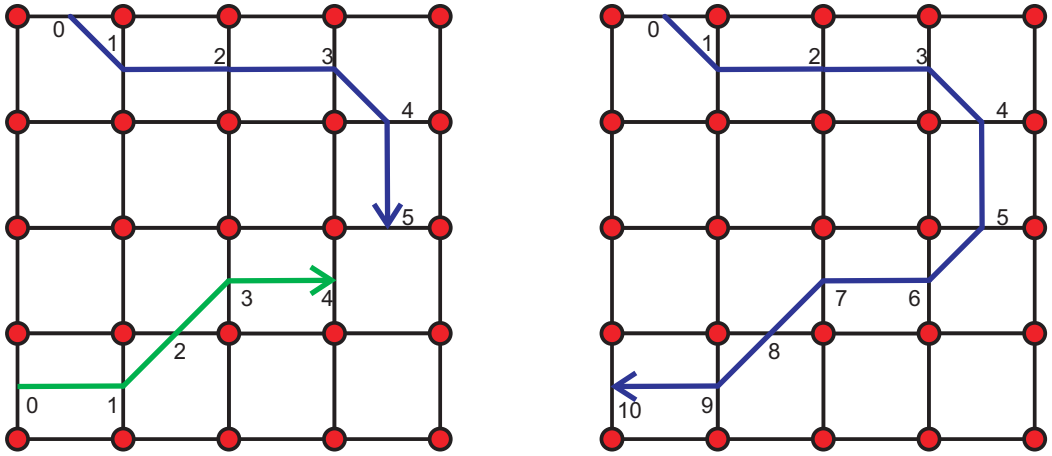


Abbildung 39: Veranschaulichung der Richtungsentstehung durch die Einfügereihenfolge (aufsteigend numeriert). In der grünen Gitterzelle wurde die Linie mit den initialen Elementen 0 und 1 erzeugt. Die gelben Gitterzellen enthalten die Linienenden.

nigung äußerst problematisch, da ein Anhängen der einen Liste an die Andere unmöglich ist (siehe Abb. 40(a)). Stattdessen muss eine Liste invertiert werden, indem ihre Elemente in umgekehrter Reihenfolge an die andere Liste angehängt werden (siehe Abb. 40(b)). Der Vorgang verursacht Kosten, die proportional zur Länge der kürzeren Liste sind. Da einer von sechs möglichen Linientypen ein Anhängen erzwingt, muss eine Isolinie im Laufe ihrer Erzeugung in der Regel mehrmals invertiert werden. Die mittlere Anzahl der Invertierungsoperationen sowie deren jeweilige Kosten sind jeweils proportional zur Anzahl der Elemente einer Liste. Diese wiederum wird durch die Gitterauflösung bestimmt, sodass die Kosten einer Anhängeoperation mit dem Quadrat der Messwertanzahl steigen. Zusätzlich muss der Algorithmus jede Gitterzelle einmal betrachten, sodass der Gesamtaufwand in der dritten Ordnung von der Gitterauflösung abhängig ist.
Außerdem kann eine Gitterzelle nicht lokal bestimmen, welches Ende einer Liste zu ihr gehört. So hat eine der in Abb. 39 dargestellten gelben Gitterzellen an ihrer Unterseite den Kopf, die andere den Schwanz. Da beide vom gleichen Gitterzellentyp sind, kann dieser nicht zur Bestimmung der Richtung dienen. Folglich muss sich eine Gitterzelle zu jeder ihrer Linien die Richtung merken und alle bislang erwähnten Operationen müssen um eine weitere Fallunterscheidung ergänzt werden. Diese Richtungsinformation muss bei Invertierungsoperationen am nicht beteiligten Linienende angepasst werden, weshalb eine Linie ihre Endgitterzellen kennen muss. Auch wenn diese zusätzlichen Operationen keine Auswirkungen auf das asymptotische Verhalten haben, tragen sie sicherlich nicht zur Eleganz des Algorithmus bei und verringern die Performance in der Praxis etwas.



(a) Ausgangssituation: Zwei Listen treffen mit dem gleichen Ende aufeinander.

(b) Anhängen aller Elemente einer Liste in umgekehrter Reihenfolge an die Andere.

Abbildung 40: Beispiel einer Vereinigung unterschiedlich gerichteter Listen. Die Zahlen geben die Elementreihenfolge und damit die Richtung an.

Ungerichtete einfach verlinkte Liste

Deutlich eleganter ist eine Bestimmung des richtigen Linienendes durch die Liste selbst. Dazu wird ausgenutzt, dass zusammengehörige Liniensegmente benachbarter Gitterzellen in einem Punkt enden, dessen Index beiden Gitterzellen bekannt ist. Beim Anhängen eines Liniensegments an eine andere Linie müssen beide Indices dieser Linie übergeben werden. Dabei dient der Index der gemeinsamen Kante dieser Gitterzellen der Identifizierung des Linienendes, an welches der andere Index angehängt wird. Das Auffinden der richtigen Enden bei der Vereinigung zweier Linien funktioniert ebenfalls über Vergleiche der Randindices. Da die ungerichtete Liste zwei gleichberechtigte Enden hat, entfällt der zum Verwalten der Richtung nötige Aufwand.

Wird eine Liste invertiert, ändert sich nicht die Ordnung (Nachbarschaftsbeziehung) der Knoten, sondern die Richtung. Dies geschieht, indem alle Verweise umgedreht werden. Beispielsweise wird eine Liste $E_1 \rightarrow E_2 \rightarrow ... \rightarrow E_{n-1} \rightarrow E_n$ durch Invertieren in $E_1 \leftarrow E_2 \leftarrow ... \leftarrow E_{n-1} \leftarrow E_n$ überführt. Die extrem ineffizienten Listeninvertierungen, die bei Verbundoperationen auftreten können, sind bedingt durch die Richtung der Knoten in der Liste. Diese Richtung steht in keinem Zusammenhang mit der Linie, sondern entsteht durch die Reihenfolge, in der die Indices eingefügt wurden. Die für die Linienrepräsentation benötigte Information ist lediglich die Ordnung der Punkte. Folglich ist die einfach verlinkte Liste keine geeignete Datenstruktur zur Repräsentation einer Linie.

Doppelt verlinkte ungerichtete Liste ungerichteter Knoten

Das Ziel ist eine Datenstruktur, die unabhängig von der durch die Linienerzeugung entstandenen Reihenfolge ist und nur die Ordnung der Indices verwaltet. Die Lösung ist eine doppelt verlinkte ungerichtete Liste ungerichteter Knoten (siehe Abb. 41). Von den Randknoten abgesehen hat darin jeder Knoten zwei gleichwertige Nachbarknoten. Durch die doppelte Verlinkung kann die Liste an beiden Enden an eine andere Liste angehängt werden. Das wäre bei

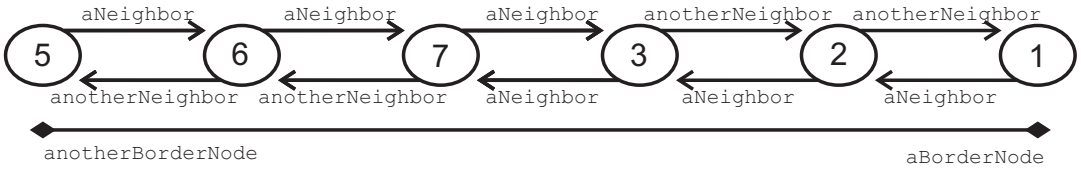


Abbildung 41: Beispiel einer doppelt verlinkten ungerichteten Liste ungerichteter Knoten. Die Zahlen in den Knoten stehen für den enthaltenen Schnittpunktindex.

doppelt verlinkten gerichteten Knoten zwar auch möglich, jedoch sind nach dem Anhängen an eine andere Liste eventuell Vorgänger und Nachfolger aller Knoten vertauscht. Anschließend müsste wieder die ganze Liste durchgegangen werden und jeglicher Vorteil ginge verloren. Wird auf eine Richtung verzichtet, muss lediglich der Auslesemechanismus angepasst werden. Dieser beginnt willkürlich an einem Listenende und erkennt das jeweils nächste Element daran, dass es nicht das Vorherige ist. Eine Verknüpfungsoperation wird weiterhin beim Auftreten eines bestimmten Linientyps initiiert. Allerdings müssen die Linien selbst festlegen, wie dies geschehen soll. Vor der Verknüpfungsoperation zweier Listen wird zunächst jeweils das Ende mit dem gemeinsamen Index ausgewählt (siehe Abb. 42 links). Ein Knoten, der zuvor auf den Knoten der eigenen Liste mit dem gemeinsamen Index verwiesen hat, verweist nun auf den Knoten der anderen Liste mit dem gemeinsamen Index (siehe Abb. 42 Mitte). Nachdem der verbliebene Knoten mit dem gemeinsamen Index einen Rückverweis auf diesen erhalten hat und die Ränder der Liste neu definiert worden sind, ist die Operation beendet (siehe Abb. 42 rechts). Würde in diesem Beispiel jeweils `anotherNeighbor` als

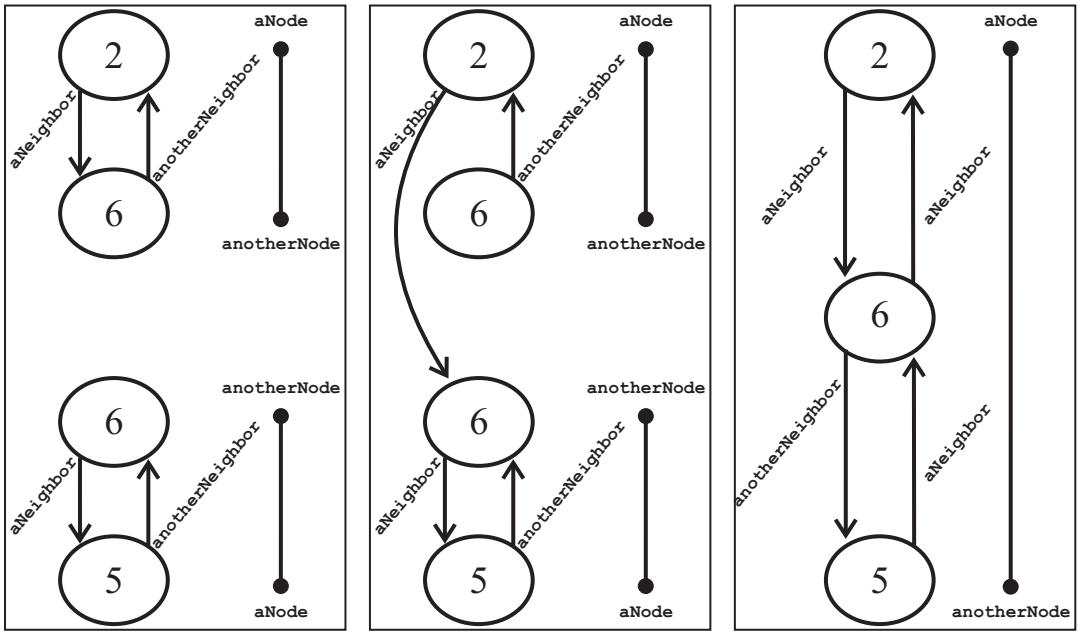


Abbildung 42: Beispiel einer Verbindungsoperation zweier Listen ungerichteter Knoten. Die Zahlen in den Knoten stehen für den enthaltenen Schnittpunktindex.

Vorgänger und `aNeighbor` als Nachfolger angesehen, müssten bei der unteren Liste nach dem Anhängen alle Verweise ausgetauscht werden. Insgesamt ermöglicht diese Liste die Ver-

einigung zweier Listen mit konstantem Aufwand, der insbesondere nicht von der Länge der Linien abhängt, da keine Verweise angepasst werden müssen.

7.3.3 Bewertung

Alle Operationen, die in einer Gitterzelle durchgeführt werden können, verursachen konstante Kosten. Da der Algorithmus alle Gitterzellen genau einmal betrachtet, verhält sich die Laufzeit für einen Isolevel linear zur Anzahl der Gitterzellen bzw. Messwerte. Isolinien zu verschiedenen Isoleveln können keine gemeinsamen Punkte haben, weshalb die Berechnungen zu verschiedenen Isoleveln nicht voneinander profitieren können. Die Laufzeit in Abhängigkeit der Gitterzellenzahl n und Isolevelzahl m ist demnach: $O(n \cdot m)$. Weil jeder Isolevel jeder Gitterzelle mindestens einmal betrachtet werden muss, ist die Laufzeit des Algorithmus nicht mehr verbesserbar. Der Speicherbedarf ergibt sich analog zu $O(n \cdot m)$, denn nichts befindet sich mehrfach im Speicher. Somit ist das asymptotische Verhalten mit dem des einfachen Marching Square Algorithmus identisch und in der Praxis verringert sich die Animationsgeschwindigkeit kaum, wie Tabelle 1 in Kapitel 8 zu entnehmen ist. Insgesamt kann festgehalten werden, dass der erweiterte Marching Square Algorithmus aufgrund seiner linearen Laufzeit lediglich vernachlässigbare Auswirkungen auf das Laufzeitverhalten hat und folglich zur Anwendung in dieser Echtzeitumgebung bestens geeignet ist. Durch die so ermöglichte Glättung und Beschriftung der Isolinien konnte die Ästhetik der Darstellung des Programms entscheidend verbessert werden. Darüber hinaus kann der Algorithmus auch als Erweiterung der anderen hier beschriebenen Contourplot Verfahren implementiert werden, da sich diese nur durch den Linienverlauf innerhalb einer Gitterzelle vom Marching Square Algorithmus unterscheiden.

7.4 Glättung der Isolinien

Zur Visualisierung von Wetter- und Klimadaten werden hauptsächlich Isolinien und Isoflächen verwendet. Diese werden bei der Vektorisierung in Form von eckigen Polygonen aus den Rasterdaten extrahiert. Eine solche kantige Darstellung ist nicht sehr ästhetisch und vor allem bei einer starken Vergrößerung werden die Ecken sehr deutlich. Für eine harmonischere Darstellung sollen die Polygone geglättet werden. Dabei ist es wichtig die Originalwerte nicht zu sehr zu verändern und dennoch eine ansehnliche Kurve zu erzeugen. In diesem Kapitel werden mehrere Verfahren vorgestellt, mit denen ein Linienzug geglättet werden kann.

7.4.1 Bézierkurven

Eine Bézierkurve wird durch Stützpunkte definiert, wobei die Punkte in Anker- und Kontrollpunkte unterschieden werden. Die Kurve verläuft durch die beiden Endpunkte der Kurve und die Ankerpunkte beeinflussen den Verlauf. Dabei liegt die Kurve in der konvexen Hülle des Kontrollpolygons [Schwarz]. Für $0 \leq t \leq 1, t \in \mathbb{R}; i = 0, ..., n\, n \in \mathbb{Z}$ wird eine Bézierkurve n-ter Ordnung mit den Stützpunkten P_i wie folgt definiert:

$$P(t) = \sum_{i=0}^{n} P_i B_{i,n}(t)$$

Dabei entsprechen die $B_{i,n}$ den Bersteinpolynomen

$$B_{i,n}(t) = \begin{pmatrix} n \\ i \end{pmatrix} t^i (1-t)^{n-i}$$

Die wichtigsten Eigenschaften der Bernsteinpolynome sind:

- der Wertebereich ist positiv auf dem Intervall $[0; 1]$.
- für $0 < t < 1, t \in \mathbb{R}$ sind die Bernsteinpolynome $\neq 0$.
- für ein festes t gilt: $\sum_{i=0}^{n} B_{i,n}(t) = 1$
- $B_{i,n}(t) = B_{n-i,n}(1-t)$
- Die Maxima von $B_{i,n}(t)$ in $[0; 1]$ liegen an den Stellen $t = \frac{i}{n}$, mit $i = 0, ..., n$

Die Bernsteinpolynome geben an, wie stark ein Stützpunkt P_i die Kurve beeinflusst. Dabei verläuft die Kurve durch den Startpunkt P_0 tangential zur Geraden $\overline{P_0P_1}$ und endet im Stützpunkt P_n tangential zur Geraden $\overline{P_{n-1}P_n}$

Bézier Kurven in OpenGL In OpenGL können Bézierkurven mithilfe der Evaluators der `gl` Bibliothek definiert und gerendert werden. Dabei ist die maximale Ordnung der Kurve vom verwendeten OpenGL Treiber abhängig. Die Implementation ist äußerst effizient in Hardware möglich und erlaubt die sehr genaue Auswertung der Kurven, da kein Iterationsverfahren verwendet wird [Davis].

Bewertung Aufgrund der begrenzten Ordnung der Bézierkurven in OpenGL ist ein Aneinandersetzen mehrerer Kurven unvermeidbar. Dies führt zu Unstetigkeiten in höheren Ableitungen der Kurven an den Übergängen, weshalb diese Stellen deutlich erkennbar sind (siehe Abb. 43). Wie bereits beschrieben ändert sich die Form der Polygone außer im zweideutigen

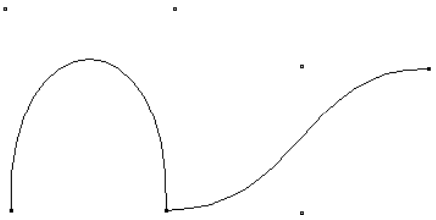


Abbildung 43: Zwei aneinanderliegende kubische Bézierkurven [Vor]

Fall zeitlich stetig. Allerdings kann durch die Gitterzellenstruktur bedingt die Stützstellenzahl von einem Zeitpunkt zum nächsten stark variieren. Folglich können sich die Übergangsstellen zu jedem Zeitpunkt an völlig anderen Orten befinden. Das Resultat ist eine extrem unstetige zeitliche Änderung der Form der geglätteten Kurven. Damit ist die Darstellungsqualität insgesamt mangelhaft.

7.4.2 Splines

B-Splines Bei B-Splines ist der Grad der Polynome unabhängig von der Zahl der Stützpunkte und nicht alle Stützpunkte haben Einfluss auf den gesamten Kurvenverlauf [Vor]. Für $n+1$ Stützpunkte $P_0, ..., P_n$ und einen Knotenvektor $T = (t_0, t_1, ..., t_{n+k})$, mit $t_j \leq t_{j+1}$ wird ein B-Spline wie folgt definiert:

$$P(t) = \sum_{i=0}^{n} P_i N_{i,k}(t)$$

Darin sind die Polynome $N_{i,k}(t)$ die Basis des B-Splines:

$$N_{i,1}(t) = \begin{cases} 1 & \text{falls } t_i \leq t \leq t_{i+1} \\ 0 & \text{sonst} \end{cases}$$

$$N_{i,k}(t) = \frac{t-t_i}{t_{i+k-1}-t_i} \cdot N_{i,k-1}(t) + \frac{t_{i+k}-t}{t_{i+k}-t_{i+1}} \cdot N_{i+1,k-1}(t), k > 1$$

Sie sind abschnittsweise vom Grad $k-1$ und stellen die Gewichte der Punkte P_i dar, welche sich maximal auf k Kurvenabschnitte auswirken können. Einige wesentliche Eigenschaften der B-Splines:

- Sie verlaufen innerhalb des aus den Stützpunkten gebildeten konvexen Polygons.
- Sie sind invariant unter affinen Abbildungen.
- Aufgrund der Unabhängigkeit des Polynomgrades von der Stützpunktanzahl kann eine Kurve zur Glättung eines beliebig komplexen Polygons verwendet werden.
- Sichtbare Übergänge wie bei den Bézierkurven entfallen damit.

Ein Beispiel für eine B-Spline Kurve ist in Abbildung 44 zu sehen.

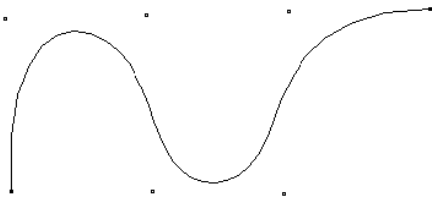


Abbildung 44: Eine B-Spline Kurve [Vor]

NURBS Bei NURBS (Non Uniform Rational B-Splines) handelt es sich um die allgemeinsten Formen zum Zeichnen von Kurven und Flächen [Vor]. Eine NURBS Kurve ist folgendermaßen definiert:

$$P(t) = \sum_{i=0}^{n} P_i R_{i,k}(t) \text{ mit:}$$

$$R_{i,k}(t) = \frac{h_i N_{i,k}(t)}{\sum_{j=0}^{n} h_j N_{j,k}(t)}$$

Einige wesentliche Eigenschaften der NURBS sind:

- Invarianz bzgl. perspektivischer Projektion
- NURBS sind eine Verallgemeinerung anderer Kurven wie Bézierkurven und B-Splines. Für $h_i = 1 \; \forall i$ reduziert sich die NURBS Kurve zur entsprechenden B-Spline Kurve.
- NURBS ermöglichen die analytische Beschreibung von Standardformen (z.B. Kegelschnitte).

Die `glu` Bibliothek von OpenGL ermöglicht die Auswertung von NURBS Kurven und Flächen. Es handelt sich dabei um einen Software Aufsatz, der auf den in Hardware ausführbaren Evaluators der `gl` Bibliothek basiert. Die Verwendung der NURBS würde höchsten ästhetischen Ansprüchen genügen und vermutlich auch ausreichend performant sein. Allerdings ist bislang eine Umsetzung in JOGL ausgeblieben, weshalb in dieser Arbeit keine NURBS eingesetzt werden. Für die Zukunft dagegen ist eine Implementation in JOGL nicht ausgeschlossen.

7.4.3 Subdivision Surface Verfahren

Das Subdivision Surface Verfahren wird in der Computergrafik eingesetzt, um aus beliebigen dreidimensionalen Polyedern glatte Flächen zu bilden. Die Anwendung in der Computergrafik wurde 1978 durch Edwin Catmull und Jim Clark [Catmull], sowie Daniel Doo und Malcolm Sabin [Doo] bekannt. Das Verfahren verwendet eine wiederholte Verfeinerung eines initialen Polyeders, wodurch eine Sequenz mit immer mehr Polygonflächen entsteht, welche gegen eine glatte Oberfläche konvergiert (Abbildung 45). Durch das Verfahren wird die Modellierung komplizierter glatter Formen stark vereinfacht.

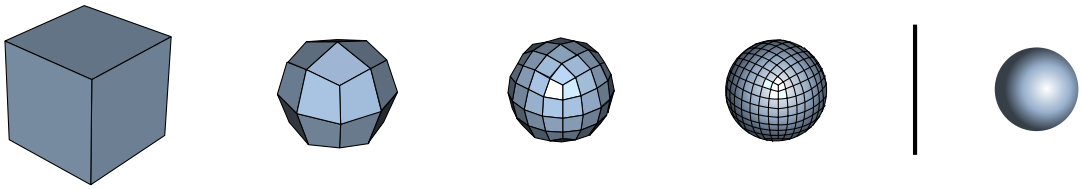
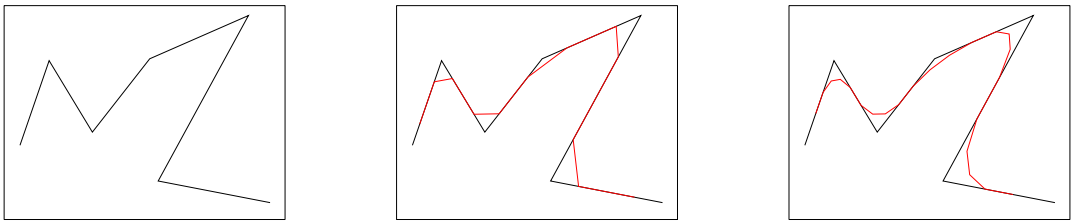


Abbildung 45: Ausgehend von einem Würfel, über die ersten drei Schritte der Catmull-Clark Subdivision, hin zum resultierenden Subdivision Surface [Wp3]

7.4.4 Subdivision Curve Verfahren

Ein Verfahren zur Glättung von Linienzügen stellt das Subdivision Curve Verfahren dar, eine zweidimensionale Vereinfachung des dreidimensionalen Subdivision Surface Verfahrens. Auch hier werden einem Polygonzug durch Unterteilung der Linien immer weitere Punkte hinzugefügt. Die Grundidee zum Subdivision Curve Verfahren stammt bereits aus dem Jahre 1947 von G. Rahm [Rahm] und führte zum ersten Corner Cutting Verfahren von Chaikin im Jahr 1974 [Chaikin]. Chaikin verwendet ein stationäres Unterteilungsschema, das heißt in jedem Interationsschritt $k = 1, 2, ...$ wird dieselbe Methode genutzt, um die Strecke zwischen zwei Punkten weiter aufzuteilen (corner cutting). Dazu werden in ein Teilstück zwei Zwischenpunkte bei $\frac{1}{4}$ und $\frac{3}{4}$ eingefügt und anschließend die Punkte der vorherigen Iterationsschritte gelöscht. Der Linienzug wird so bei jedem Iterationsschritt durch weitere Punkte geglättet (Abbildung 46). Für $k \to \infty$ konvergiert das Verfahren gegen eine quadratische B-Spline Kurve.

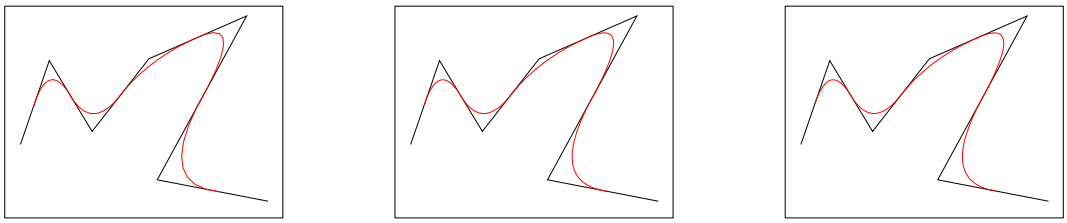
Bewertung Das Verfahren stellt einen effizienten Algorithmus für die Kantenglättung eines Polygons beim Rendering dar. Weil die Polygone nicht abschnittsweise sondern als Ganzes geglättet werden können, gibt es keine unansehnlichen Übergangsstellen. Damit ist auch eine gleichmäßigere Animation der Linien möglich. In Abbildung 47 ist eine mit 3 Iterationsschritten des Chaikin Algorithmus geglättete Kurve im Vergleich mit dem ungeglätteten



(a) Ungeglättes Polygon: 7 Stützpunkte

(b) 1. Iteration: 12 Eckpunkte

(c) 2. Iteration: 22 Eckpunkte



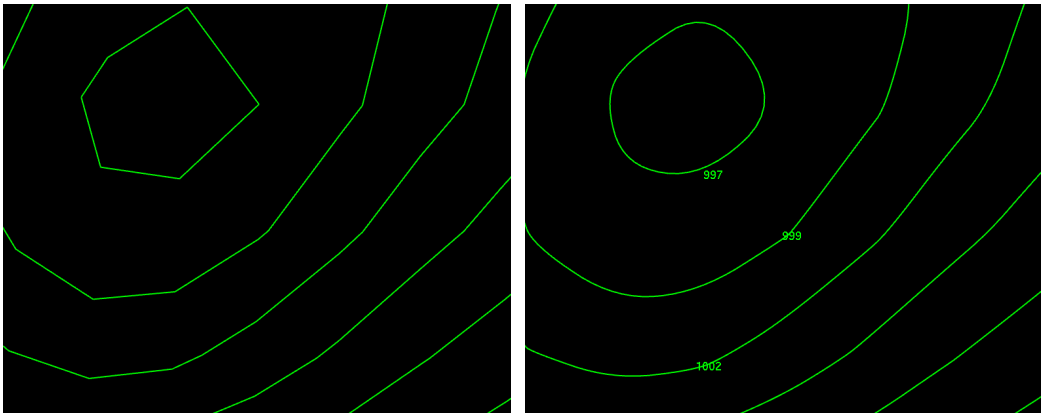
(d) 3. Iteration: 42 Eckpunkte

(e) 4. Iteration: 82 Eckpunkte

(f) 5. Iteration: 162 Eckpunkte

Abbildung 46: Chaikin Verfahren zur Kantenglättung [Kunze]

Polygon zu sehen. Ein weiterer Vorteil des Verfahrens ist der durch die Iterationsschrittzahl einstellbare variable Glättungsgrad. Insgesamt ist das Verfahren sehr gut zum Einsatz in dieser Arbeit geeignet.



(a) Ungeglättetes Pologon

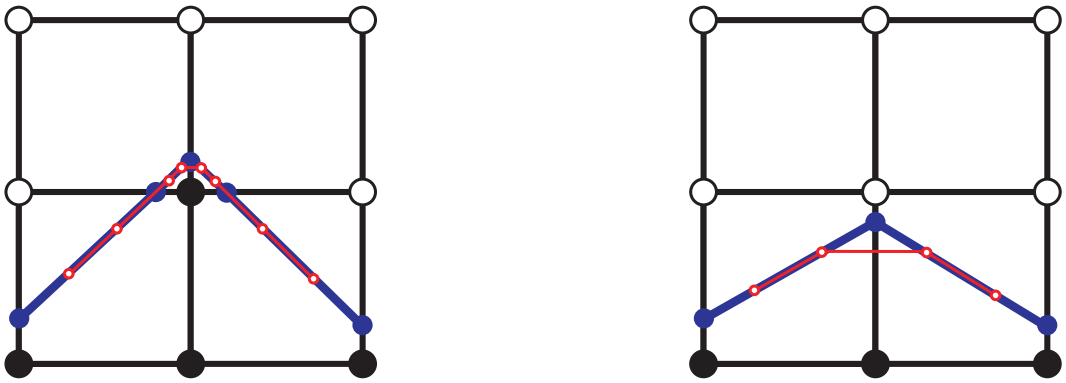
(b) Geglättetes Polygon

Abbildung 47: Vergleich der Originalpolygone mit den durch den Chaikin Algorithmus geglätteten

7.4.5 Zeitliche Stetigkeit

Mithilfe des Marching Square SE Algorithmus können Polygone gebildet werden, deren Verlauf sich außer im zweideutigen Fall zeitlich stetig ändert. Allerdings können von einem Zeitpunkt zum Nächsten neue Stützpunkte hinzukommen oder verschwinden. Dies hat natürlich

starken Einfluss auf die Glättung der Kurve, wie anhand des folgenden Beispiels verdeutlicht wird. In Abbildung 48(a) liegt der Wert des mittleren Gitterpunktes minimal oberhalb des Isolevels, sodass sich in diesem Bereich auf engstem Raum drei Schnittpunkte (dargestellt durch blaue Kreise) befinden. Die erste Iteration des Chaikin Algorithmus erzeugt zwischen je zwei Stützpunkten des Polynoms jeweils zwei neue Punkte (dargestellt durch rote Kreise). Einen minimalen Zeitschritt später liegt der mittlere Gitterpunkt nicht mehr oberhalb des Isolevels, weshalb sich in diesem Bereich die Stützpunktanzahl auf einen reduziert (siehe Abbildung 48(b)). Als Folge daraus wird das geglättete Polygon wesentlich weniger in die obere Ecke gezogen. Somit kommt es im Moment der Zustandsänderung des mittleren Gitterpunktes zu einer Unstetigkeit im zeitlichen Verlauf des geglätteten Polygons aber nicht des Ausgangspolygons.
Dieses Verhalten tritt bei anderen Glättungsalgorithmen ebenfalls auf, da die Stützpunkte immer hohen Einfluss auf die Kurvenform haben. In der Praxis allerdings sind die Sprünge im Animationsverlauf zumindest bei den verwendeten Daten subjektiv kaum bemerkbar.



(a) Situation direkt vor Zustandsänderung des mittleren Punktes

(b) Unmittelbar nach Zustandsänderung des mittleren Punktes

Abbildung 48: Animationsproblem bei Glättung der Polygone. Vergleich der Originalpolygone (blau) mit den durch eine Chaikin-Iteration geglätteten (rot).

7.4.6 Problem der Abhängigkeit der Polygone von der Gitterstruktur

Aufgrund der lokalen Erzeugung der Isoliniensegmente in Abhängigkeit der Gitterzellenstruktur können gerade oder gering gekrümmte Linien entstehen, die durch mehr als zwei Schnittpunkte definiert sind. Rahmen zwei solcher Linien eine Ecke wie in Abbildung 48(a) ein, liefern die Glättungsalgorithmen sehr unbefriedigende Ergebnisse. Beispielsweise kann der Chaikin Algorithmus die Ecke in Abbildung 48(a) selbst mit sehr vielen Iterationen nur in den oberen beiden Gitterzellen glätten, in den unteren beiden dagegen werden lediglich neue Punkte gebildet, welche wiederum genau auf den Geraden liegen. Dieses Problem ist eng mit dem in Kapitel 7.4.5 geschilderten verwandt und tritt bei anderen Glättungsalgorithmen in ähnlicher Form auf. Eine Lösungsmöglichkeit wäre das nachträgliche Entfernen von näherungsweise auf einer Geraden liegenden Stützpunkten. Dies würde jedoch einen hohen zusätzlichen Rechenaufwand erfordern und die Genauigkeit der Darstellung weiter verringern. Allerdings könnten dadurch einige Sprünge in der Animation abgeschwächt werden, die gerade aus diesem Effekt resultieren.

7.4.7 Bewertung der Kantenglättung

Die Isolinienglättung verringert die mathematische Genauigkeit der Darstellung, kostet zusätzliche Rechenzeit und lässt die Animation weniger flüssig wirken. Ein weiteres Problem ist das Vermitteln des Eindrucks von wesentlich höher aufgelösten Daten als in Realität vorliegen. Trotzdem werden in der aktuellen Version des Programms die Isolinien durch den Chaikin Algorithmus geglättet, da die Darstellung gerade in der Nahansicht einen deutlich ästhetischeren Eindruck hervorruft. Darüber hinaus wirkt die Visualisierung durch die Glättung der vielen kleinen Ecken und Kanten übersichtlicher und damit auch verständlicher.

7.5 Zusammenfassung

In diesem Kapitel wurden mehrere elementare Verfahren zum Vektorisieren der Rasterdaten vorgestellt bzw. neu entwickelt. Von diesen liefert zwar der CONREC Algorithmus mit bilinearer Interpolation die besten Ergebnisse, aufgrund der besseren Performance wurde trotzdem der Marching Square Algorithmus favorisiert. Die so erzeugten Liniensegmente können mithilfe der Erweiterung des Marching Square Algorithmus mit in Abhängigkeit von der Gitterauflösung linearer Laufzeit in Polygone überführt werden. Dadurch wird eine Beschriftung und Glättung der Isolinien beispielsweise durch ein Subdivision-Curve-Verfahren ermöglicht. Alternativ würde allerdings der CONREC Algorithmus mit bilinearer Interpolation und ohne zusätzliche Glättung ein mathematisch exakteres Ergebnis liefern, das außerdem auch ohne jegliche Sprünge animierbar wäre. Trotzdem ist der praxisbezogene Kompromiss der Kombination des Marching Square SE Algorithmus zur Isolinienerzeugung und der anschließenden Glättung durch den Chaikin Algorithmus hervorragend zur effizienten und detaillierten Visualisierung des Luftdrucks geeignet. Dies beweist Abbildung 15 auf Seite 25 eindrucksvoll.

8 Performance

Gegenstand dieses Kapitels sind Performancetests zentraler Algorithmen und Techniken, sowie des Applets selbst anhand realer Wetterdaten in der Praxis.

Algorithmenperformance In diesem Abschnitt werden einige der verwendeten Techniken und Algorithmen in der Praxis anhand realer Daten getestet. Dazu werden Luftdruckdaten eines Gitters mit $77 \cdot 57$ Werten mittels verschiedener Techniken visualisiert. Genauere Angaben zu den Daten und zur verwendeten Hardwarekonfiguration sind im Anhang (Kapitel 12) beschrieben. Allgemein ist zu beachten, dass die Messergebnisse aufgrund des variierenden Optimierungsgrades der verschiedenen Algorithmen nur tendenziellen Charakter haben. In der Tabelle 1 werden verschiedene Algorithmen zur Isoliniengenerierung und zur Glättung derselben verglichen. Darin werden der Chaikin und der Bézier Algorithmus mit je drei

Algorithmus	FPS	Relativer Unterschied
Marching Square	285	± 0 %
Marching Square SE	250	-12 %
Marching Square SE + Chaikin (Faktor: 2)	237	-17 %
Marching Square SE + Chaikin (Faktor: 4)	225	-21 %
Marching Square SE + Chaikin (Faktor: 8)	205	-28 %
Marching Square SE + Bézier (Faktor: 2)	185	-35 %
Marching Square SE + Bézier (Faktor: 4)	162	-43 %
Marching Square SE + Bézier (Faktor: 8)	130	-54 %

Tabelle 1: Vergleich der Algorithmen zur Isoliniengenerierung und zur Glättung derselben. Die relativen Unterschiede beziehen sich auf die erste Einstellung.

Glättungsgraden getestet, welche durch Faktoren gegeben sind, die das Verhältnis der erzeugten Vertices zur Anzahl der Stützpunkte angeben. Hier wird insbesondere der geringe Performanceverlust des Marching Square SE Algorithmus gegenüber dem Marching Square Algorithmus von nur 12% deutlich. Bei zusätzlicher Glättung mit drei Iterationen des Chaikin Algorithmus (Faktor 8) hält sich der Gesamtperformanceverlust mit 28% ebenfalls in Grenzen. Wird stattdessen eine Bézierkurve vergleichbaren Glättungsgrades (Faktor 8) verwendet, sinkt die Performance mit 54% deutlich stärker. Möglicherweise ist dies allerdings auf eine geringe Optimierung der Technik zurückzuführen, da sie aufgrund der mangelhaften Darstellungsqualität (vergl. Kapitel 7.4.1) nicht weiter untersucht wurde.
Weitere Performancetests sind der Tabelle 2 zu entnehmen. Auffällig ist die Unabhängigkeit der Framerate vom Grad des Multisamplings, welches eine deutliche Zusatzbelastung für die Grafikkarte darstellt. Dagegen ist die Prozessorauslastung nahezu konstant 100% und bei einer Zweikern CPU aufgrund des verwendeten Multithreadings sogar bis zu 60%, folglich ist das Programm stark CPU limitiert. Insgesamt decken sich diese Beobachtungen mit den bisherigen Untersuchungen zur Performance [Wenke].

Testeinstellung	FPS	Relativer Unterschied
Chaikin (Faktor: 8)	205	± 0 %
Chaikin (Faktor: 8) mit $16x$ Multisampling	205	± 0 %
Chaikin (Faktor: 8) mit Internetverbindung (DSL 1500)	205	± 0 %
Chaikin (Faktor: 8) mit Internetverbindung (56k Modem)	53	-72 %

Tabelle 2: Weitere Performancetests. Die relativen Unterschiede beziehen sich auf die erste Einstellung.

Des Weiteren verringert auch das Nachladen der Daten über das Internet die Performance zumindest in dieser Szene nicht. Dies ist durch das parallele Nachladen der Daten in einem extra Thread zu begründen und gilt nur, wenn die angeforderten Daten vor Erreichen des nächsten Zeitpunkts vollständig geladen werden können. Ist das beispielsweise aufgrund einer höheren Datenauflösung oder einer schlechteren Internetverbindung nicht möglich, kann die Performance durchaus drastisch sinken (Tabelle 2). Außerdem muss die Animation regelmäßig angehalten werden bis die benötigten Daten verfügbar sind, wodurch eine sehr ungleichmäßige Geschwindigkeit entsteht.

Gesamtperformance In einem zweiten Performancetest wird nun noch die Geschwindigkeit des Applets selbst, welches unter der URL

http://project.informatik.uos.de/hwenke/EarthWeather3D.html

zu erreichen ist, ermittelt. Dazu wird die Animation in der initialen Kameraposition, bei der für knapp zwei Drittel der sichtbaren Erdoberfläche Daten vorliegen, mit aktivierter Temperatur-, Wind- und Luftdruckdarstellung gestartet. Zunächst fällt hierbei die recht lange Startzeit des Applets von über 10 sec auf, bei der lediglich 170 kByte heruntergeladen werden. Da beim Start des Applets noch keine Wetterdaten angefordert werden, entspricht die geringe Datenmenge den Erwartungen. Die Animation selbst läuft bei 80 Animationsstufen zwischen je zwei Zeitpunkten auf der beschriebenen Systemkonfiguration mit 50 FPS. Insgesamt beläuft sich die während der Animation über 11 Zeitpunkte der drei Wetterausprägungen heruntergeladene Datenmenge auf 430 kByte. Dies entspricht ca. einem Fünftel der in der Datenbank enthaltenen Datenmenge und deckt sich folglich in etwa mit den Erwartungen.

Zusammenfassung Die Performancetests bestätigen die außerordentliche Effizienz der verwendeten Algorithmen wie etwa des Marching Square SE Algorithmus in der Praxis. Außerdem werden die geringen Ansprüche des Programms an die Internetverbindung und die Grafikkarte deutlich. Einen großen Nachteil stellt allerdings die unverhältnismäßig hohe Beanspruchung der CPU dar. Trotzdem reicht die Performance insgesamt aus, um den Eindruck einer vollständig flüssigen Animation zu erzeugen.

9 Bewertung der Eigenschaften des Ansatzes

In diesem Kapitel werden die wesentlichen Eigenschaften des Projekts bewertet und denkbare Nutzungsmöglichkeiten aufgezeigt.

Räumlichkeit Die Möglichkeit der 3D-Darstellung des Klimas auf der gesamten Erde ist keine technische Spielerei sondern absolut unverzichtbar. Schließlich macht das Klima nicht vor Länder- und Kontinentgrenzen halt, sondern kann nur global verstanden werden. Darüber hinaus erleichtert gerade die räumliche Darstellung dem Betrachter die Orientierung und damit auch die Navigation. Des Weiteren liegen Wetterdaten nicht nur für die Erdoberfläche, sondern auch in verschiedenen Höhenstufen vor (siehe Kapitel 10.4). Eine brauchbare gleichzeitige Visualisierung mehrerer Atmosphärenschichten ist in 2D kaum vorstellbar.

Animation Die Dynamik des Klimas kann nicht durch statische Bilder, sondern nur durch Animation auf äußerst eindrucksvolle Weise vermittelt werden. Von zentraler Bedeutung für das Projekt sind demnach die effizienten Algorithmen zum Laden und Visualisieren der Daten, da sonst eine ausreichend flüssige Darstellung des zeitlichen Verlaufs unmöglich wäre.

Interaktivität Die Interaktivität wird in erster Linie benötigt, um dem Benutzer die genaue Auswahl der für ihn interessanten Daten zu ermöglichen. Dies geschieht beispielsweise durch die Positionierung der Kamera und indem eine bestimmte Kombination der Wetterelemente ausgewählt wird.
Außerdem müssen bei vorberechneten Datenvisualisierungen (etwa Fernseh- oder Zeitungswetterberichte) stets gestalterische Entscheidungen getroffen werden, welche visuelle Umsetzung geeignet ist und welche Aspekte besonders hervorgehoben werden sollen. Ein generelles Problem ist folglich die damit unweigerlich verbundene Interpretation der Ausgangsdaten. Da in diesem Projekt dagegen die Daten zum Client übermittelt werden und die Darstellung dort dynamisch berechnet wird, kann der Benutzer in den Visualisierungsprozess eingreifen und die für ihn interessanten Eigenschaften der Daten besonders herausheben. Auf diese Weise erreicht das Programm einen bislang unübertroffenen Interaktionsgrad.

Skalierbarkeit Das Programm ermöglicht es, die Menge der zu übermittelnden Daten etwa konstant zu halten, indem diese in einer Dichte angefordert werden, die umgekehrt proportional zur Größe des sichtbaren Bereichs ist. Folglich resultiert eine höhere Maximalauflösung auf dem Server nicht in einer gestiegenen Beanspruchung der Bandbreite, sondern ermöglicht weitere Zoomstufen. Somit ist ein herkömmlicher DSL 1000 Anschluss für eine vollständig flüssige Animationsgeschwindigkeit bereits mehr als ausreichend.

Vielseitige Verwendbarkeit Der hier beschriebene Ansatz ist nicht ausschließlich auf die Visualisierung von Klima- bzw. Wetterdaten festgelegt, sondern kann zur Darstellung beliebiger georeferenzierter Daten dienen. Dazu zählen u.a. Arbeitslosigkeit, Demographie, CO_2-Ausstoß, Bevölkerungsdichte, Straßenkarten, Umfrage/Wahlergebnisse und Meeresströmungen.

Allgemeine Verfügbarkeit In diesem Projekt werden ausschließlich freie und plattformunabhängige Werkzeuge verwendet. Allerdings sind in der Praxis trotzdem plattformabhängige Eigenarten zu beachten. So gibt es beispielsweise auf Macs noch Probleme mit dem JOGL-Applet, die jedoch mangels Verfügbarkeit eines Macs nicht lokalisiert werden konnten. Theoretisch genügt jedoch zur Verwendung des Programms ein beliebiger internetfähiger Rechner. Eine 3D-Grafikkarte verbessert zwar die Animationsgeschwindigkeit und kann durch optionales Anti-Aliasing bzw. Anisotropische Filter die Bildqualität erhöhen, ist aber nicht vorausgesetzt.

Webfähigkeit Durch die Implementation als plattformunabhängiges Java-Applet ist die Einbettung in eine Webseite möglich (siehe Abbildung 49). Der Benutzer kann diese einfach

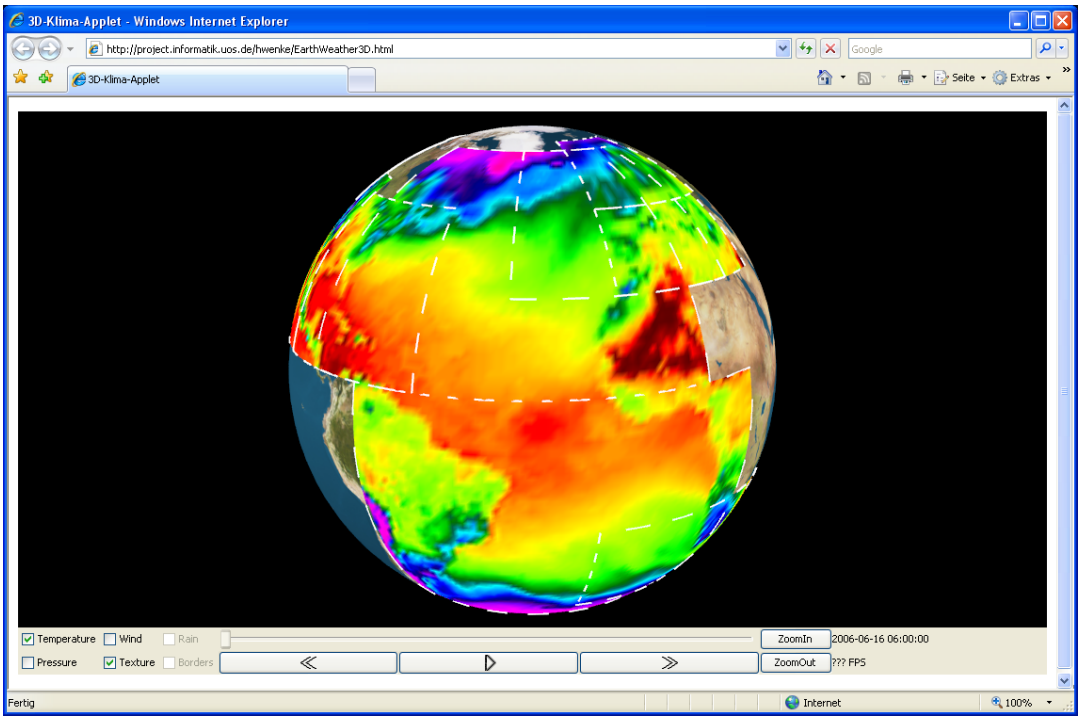


Abbildung 49: Anzeige des Applets im Internet Explorer

über die URL aufrufen und das Applet startet ohne das etwas gesondert installiert werden müsste. Ein Nachteil gegenüber einer Webseite, die Wetterdaten durch Pixelgrafiken darstellt ist die wesentlich längere Startzeit des Applets. Da während des Startvorgangs noch keine Wetterdaten geladen werden, handelt es sich hierbei um ein generelles Problem des Applets, welches nicht durch weitere Verbesserungen des Datenlademechanismus lösbar ist. Ein weiterer Nachteil ist die Voraussetzung der Java Virtual Machine. Dagegen können simple statische Pixelgrafiken bzw. animierte GIFs von jedem auch nur annähernd aktuellen Browser angezeigt werden. Insgesamt ist das Applet den herkömmlichen Webseiten somit bezüglich der Funktionalität massiv überlegen, kann aber nicht ganz deren Verfügbarkeit erreichen und benötigt eine deutlich längere Startzeit.

10 Ausblick

Dieses Kapitel soll Anregungen geben, inwieweit die im Rahmen dieser Arbeit durchgeführte Untersuchung noch vertieft werden könnte. Neben zahlreichen Detailverbesserungen des Programms zeichnen sich bereits fünf Schwerpunkte ab, die im Folgenden genauer beschrieben werden.

10.1 Geometrieunabhängige Oberflächeneinfärbungsalgorithmen

Bei den hier verwendeten herkömmlichen Methoden der räumlichen Datenvisualisierung entstehen Geometrien, deren Beschaffenheit direkt von den zugrundeliegenden Daten abhängen. Dabei werden die Rasterdaten zunächst auf ein Gitter abgebildet, dessen Gitterpunkte auf einer Kugeloberfläche liegen. Bei Kombination mehrerer Schichten unterschiedlicher Auflösung durchdringen sich diese Gitter gegenseitig, was zu unschönen Effekten führen kann.
Abgesehen von diesem generellen Problem der Rasterdatenvisualisierung tritt bei bestimmten Techniken der Isoflächendarstellung ein weiterer Effekt auf, der nachfolgend beschrieben wird. Im Rahmen dieser Arbeit ist testweise ein Verfahren zur Datenvisualisierung durch Isoflächen auf Basis des Marching Square Algorithmus implementiert worden. Dazu werden die Flächen in den einzelnen Gitterzellen, welche jeweils durch die Linien zu zwei Isoleveln und den Gitterzellenrand begrenzt sind, mit verschiedenen Farben eingefärbt. Diese lokale Vorgehensweise ermöglicht jedoch keine Glättung der Isoflächenränder. Alternativ könnte dem-

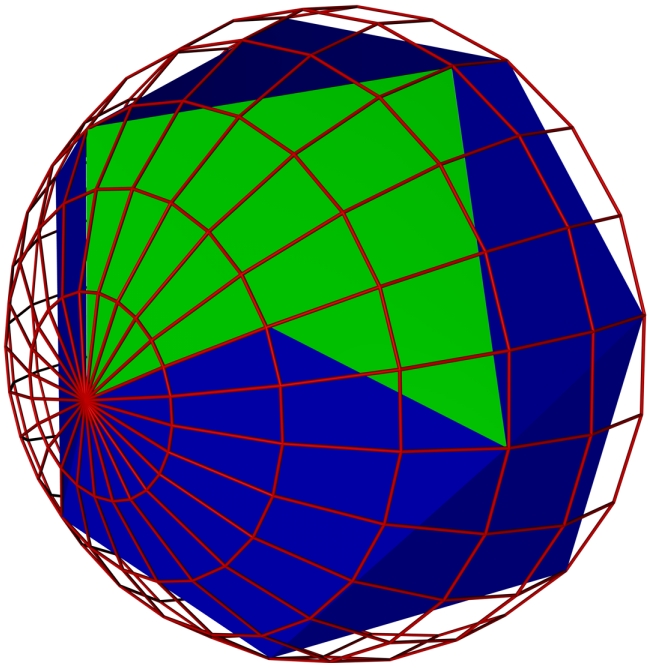


Abbildung 50: Veranschaulichung des Problems der Isosurfacedarstellung mit zwei verschiedenen Isosurfaces (blau und grün). Die Kugelform wird vollständig zerstört.

nach wiederum der Marching Square SE Algorithmus zum Erkennen der globalen Isoflächen eingesetzt werden. Allerdings tritt bei der Visualisierung solcher Isoflächen ein Effekt auf, welcher aus der dynamischen Erzeugung der einzelnen Polygongeometrien in Abhängigkeit

der Daten resultiert. Die so erzeugten Polygone liegen mit Ausnahme ihres Randes im Kugelinneren, da sämtliche Informationen über die Krümmung im Polygoninneren verloren sind (siehe Abb.50). Nach einer Glättung der Polygone liegen auch deren Ränder nicht zwangsläufig auf der Kugeloberfläche. Ein denkbarer Ansatz zur Lösung dieses Problems wäre eine nachträgliche Anpassung der Isoflächengeometrie an die Kugelgeometrie. Weil dies für jedes einzelne Bild mit erheblichem Aufwand geschehen müsste, ist eine erträgliche Animationsgeschwindigkeit nicht erwartbar. Sehr wünschenswert wäre dagegen ein Verfahren, welches die Oberfläche einer beliebigen Geometrie direkt auf Basis beliebiger Rasterdaten geeignet einfärben kann. Insbesondere soll hierbei keine neue Geometrie erzeugt werden, die sich eventuell sehr stark von der Geometrie des einzufärbenden Körpers unterscheidet.

10.2 Grafikhardware

Wie sich im Kapitel 8 herausgestellt hat, beanspruchen die aktuell verwendeten Visualisierungsalgorithmen den Prozessor übermäßig stark während die Grafikkarte völlig unausgelastet bleibt. Daher ist die Entwicklung alternativer grafikhardwaregestützter Visualisierungsalgorithmen bzw. das Auslagern von aktuell in Software implementierten Algorithmen auf die Grafikkarte wünschenswert. Denkbar ist beispielsweise das Ausführen des Marching Square Algorithmus in den programmierbaren Shadereinheiten moderner Grafikhardware. Da sich der Algorithmus nahezu beliebig parallelisieren lässt, könnte der extreme Parallelisierungsgrad aktueller Grafikkarten optimal ausgenutzt werden und folglich eine massive Performancesteigerung ermöglichen. Ferner könnte die Kugeloberfläche mithilfe von GPU-basierten Procedural-Texture-Shadern direkt gemäß den Isoflächen bzw. Rasterdaten eingefärbt werden, um die in Kapitel 10.1 beschriebenen Probleme zu lösen. Procedural-Texture-Shader werden zur Einfärbung von Oberflächen mit mathematisch beschreibbaren Eigenschaften mit beliebiger Genauigkeit verwendet. Es existiert allerdings keine scharfe Trennung zu Nonprocedural-Texture-Shadern, welche Oberflächen primär durch Auswertung von Daten einfärben [Rost]. Somit wäre es durchaus vorstellbar, die Isoflächen in einer Art zu beschreiben, dass diese Daten durch Procedural-Texture-Shader effizient ausgewertet werden können, um die Farben der einzelnen Fragments zu berechnen. Im Idealfall würden die Shader die Rasterdaten, welche aufgrund ihrer rechteckigen Anordnung problemlos wie eine Textur zu kodieren sind, als Eingabe erhalten und anschließend alle benötigten Operationen in Hardware ausführen.

10.3 Entwicklung einer Erweiterung des Marching Cubes Algorithmus zum Erkennen geschlossener Oberflächen

Es sollte überprüft werden, ob der Marching Cubes Algorithmus so erweitert werden kann, dass er lediglich durch Vergleiche direkt benachbarter Boxen zusammenhängende Flächen effizient erkennen kann. Für diese Vergleiche werden allerdings wesentlich mehr Fallunterscheidungen als beim Marching Square Algorithmus nötig, da es in drei Dimensionen 256 statt 16 verschiedene Gitterzellentypen gibt.

Schlüsselproblem Die Erweiterung des Marching Square Algorithmus verwendet eine speziell für dieses Problem entwickelte doppelt verlinkte Liste zur Linienrepräsentation. Da Linien immer genau zwei Enden haben, können Einfüge- und Linienverbundoperationen mit

konstantem Aufwand durchgeführt werden. Bei Flächen dagegen ist die Anzahl der Randpunkte von ihrer Größe und Form abhängig und demnach nicht konstant. Eine simple Erweiterung des Algorithmus, die lediglich mehr Fallunterscheidungen benötigt, ist demnach nicht möglich. Der Erfolg der Entwicklung dieses Algorithmus wird folglich maßgeblich von einer geeigneten Datenstruktur zur Repräsentation der Flächen abhängen.

Weitere Verwendungsmöglichkeiten Der Marching Cubes Algorithmus wird vorwiegend zur Visualisierung von 3D-Scans z.B. in der Computertomographie eingesetzt. Im Falle eines Erfolges können diese Bereiche von dem Algorithmus profitieren, indem etwa eine automatische Objekterkennung, einfachere Texturierung der Scans und eine effektive Ausdünnung der Geometrie möglich werden.

10.4 Visualisierung von Daten in mehreren Atmosphärenschichten

Wetterberichte werden nicht nur für die Erdoberfläche, sondern auch für verschiedene Höhenstufen erstellt. Beispielsweise liegen die vom Deutschen Wetter Dienst [DWD] durch das derzeitige Wettervorhersagemodell ermittelten Daten für Mitteleuropa in 35 verschiedenen Atmosphärenschichten vor. Besonders die Winddarstellung in den verschiedenen Höhenstufen ist für die Luftfahrt von großer Bedeutung [Walch]. Visualisierungstechniken wie etwa die Windsymbole sind zur gleichzeitigen Visualisierung aller Schichten in einer 3D-Darstellung problemlos geeignet, weil sich die Symbole nicht gegenseitig verdecken. Der Bewölkungsgrad könnte etwa mithilfe des Marching Cubes Algorithmus durch echte 3D Wolken visualisiert werden. Im Falle ausreichender Leistungsreserven könnte hier sogar der Marching Cubes SE Algorithmus zum Einsatz kommen. Aufgrund der im Vergleich zu einer Atmosphärenschicht extrem gestiegenen Anzahl der Werte und der aufwendigeren Visualisierungsalgorithmen, wie etwa dem Marching Cubes Algorithmus, wird eine Softwarelösung höchstwahrscheinlich keine ausreichend flüssige Echtzeitanimation ermöglichen. Folglich ist eine Auslagerung der Visualisierungsalgorithmen auf die programmierbare Grafikhardware unausweichlich. Davon abgesehen muss natürlich die deutlich höhere Menge der zu übertragenden Daten beachtet werden.

10.5 Testweise Implementation auf mobilen Endgeräten

In der letzten Zeit ist ein Trend zu erhöhter Multimediafähigkeit der Handys zu beobachten. So verfügt mittlerweile eine zunehmende Zahl mobiler Endgeräte über leistungsfähige GPUs (etwa NVIDIAs GoForce Reihe [nVidia]). An der Universität Osnabrück wird zur Zeit ein Projekt zur Klimadatenvisualisierung auf mobilen Endgeräten mit Digital Audio Broadcasting durchgeführt [Fox]. Es ist bereits ein Prototyp entwickelt worden, der die grundlegende Machbarkeit des Projektes zeigt. Eine Illustration zur Wetterdarstellung auf einem PDA ist in Abb. 51 zu sehen. Dabei wird SVG [W3C] zur Darstellung der vorberechneten Wetterkarten benutzt.
Alternativ könnte ein Verfahren getestet werden, Rasterdaten auf mobile Endgeräte zu übertragen und mithilfe von OpenGL ES [OGLES] zu visualisieren. Diese API stellt eine Untermenge von OpenGL dar und ist speziell für den Einsatz in eingebetteten Systemen wie Handys und PDAs entwickelt worden. Denkbare Vorteile wären eine Entlastung der CPU durch die GPU und eine aufgrund des binären Datenformats in der Praxis geringere Beanspruchung der Datenübertragungskapazität. Des Weiteren ist die Darstellung durch den Verzicht auf eine

Abbildung 51: Wetterdarstellung auf einem PDA (Illustration)

vorberechnete visuelle Umsetzung der Daten interaktiver. Auf diese Weise wäre ein aktueller Wetterbericht jederzeit und überall in einer bislang unerreichten Qualität abrufbar.

11 Fazit

Die vorliegende Untersuchung belegt die hervorragende Eignung von JOGL und Java Applets zur 3D-Visualisierung und Animation orts- und zeitabhängiger Wetterdaten mit Grafikhardwareunterstützung im Web. Das zentrale Problem der geringen Datenbandbreite aktueller Internetverbindungen konnte durch den in Kapitel 4.3 vorgestellten Datenlademechanismus sehr gut gelöst werden. Dieser ermöglicht das Laden der Daten für nahezu ausschließlich den sichtbaren Bereich der Erdoberfläche zur Animationslaufzeit im Hintergrund. Folglich entstehen für den Benutzer keinerlei Wartezeiten. Eine bislang mit interaktiven Wettervorhersagen im Web nicht erreichbare Darstellungsqualität, welche die jeweiligen datenspezifischen Charakteristika berücksichtigt, konnte mithilfe der in Kapitel 6 vorgestellten Techniken realisiert werden. Hierbei ermöglicht die gute Kombinierbarkeit der Darstellungsformen die gleichzeitige Visualisierung der einzelnen Wetter- bzw. Klimaelemente. Außerdem genügen alle Visualisierungstechniken den besonderen Anforderungen, die für eine effiziente und flüssige Echtzeitanimation erforderlich sind. Nur auf diese Weise kann die Dynamik des Wetters mit seinen mannigfaltigen sich wechselseitig beeinflussenden Ausprägungen hinreichend verstanden werden. Ferner gewährleisten die in Kapitel 7 beschriebenen Verfahren und insbesondere der in Abschnitt 7.1.4 vorgestellte CONREC Algorithmus mit bilinearer Interpolation eine sehr gute Darstellungsqualität selbst bei gering aufgelösten Daten. Dabei kann die Ästhetik durch die optionale Glättung der Polygone verbessert werden, welche mithilfe des in Abschnitt 7.3 vorgestellten Marching Square SE Algorithmus erzeugt werden können. Somit sind alle eingangs aufgeführten Ziele der Masterarbeit erreicht worden.

Das Programm kann durch seinen besonders anschaulichen und interaktiven Ansatz einen kleinen Teil dazu beitragen, dass Forschungsergebnisse, wie etwa die Entwicklung des Klimas, von größeren Teilen der Menschheit verstanden werden. Nicht zuletzt aufgrund der unaufhaltsamen Verbreitung des Internets steigt täglich die Zahl derer, die sich so über aktuelle Erkenntnisse informieren können. An dieser Stelle möchte ich noch einmal betonen, dass diese Arbeit keine technische Spielerei darstellt, sondern wie in Kapitel 9 beschrieben auf einen sinnvollen Einsatz neuer Technologien ausgerichtet ist. Darüber hinaus können einige im Rahmen der Masterarbeit entwickelte Algorithmen, wie etwa der Marching Square SE Algorithmus, auch außerhalb dieses Programms eingesetzt werden. Dies gilt in besonderem Maße für die in Kapitel 10 angedachte Generalisierung des Algorithmus auf drei Dimensionen. In seinem jetzigen Stadium bleibt das Projekt allerdings weit hinter dem bereits erkennbaren Potential zurück. Gerade die Perfektionierung fortschrittlicherer und innovativer Visualisierungsalgorithmen ist im Zeitrahmen einer Masterarbeit kaum durchführbar. Einige besonders interessante Anregungen hierfür sind im Kapitel 10 beschrieben.
Trotzdem ist das entstandene Applet auch im aktuellen Zustand in der Lage, optisch beeindruckende Echtzeitanimationen mit einem im Web bislang unerreichten Interaktivitätsgrad aus nüchternen Zahlen zu zaubern.

12 Anhang

Weitere Screenshots

Hier befinden sich noch einige Screenshots des Applets, das die Wettervorhersagedaten von Raymarine.com [Ray] visualisiert.

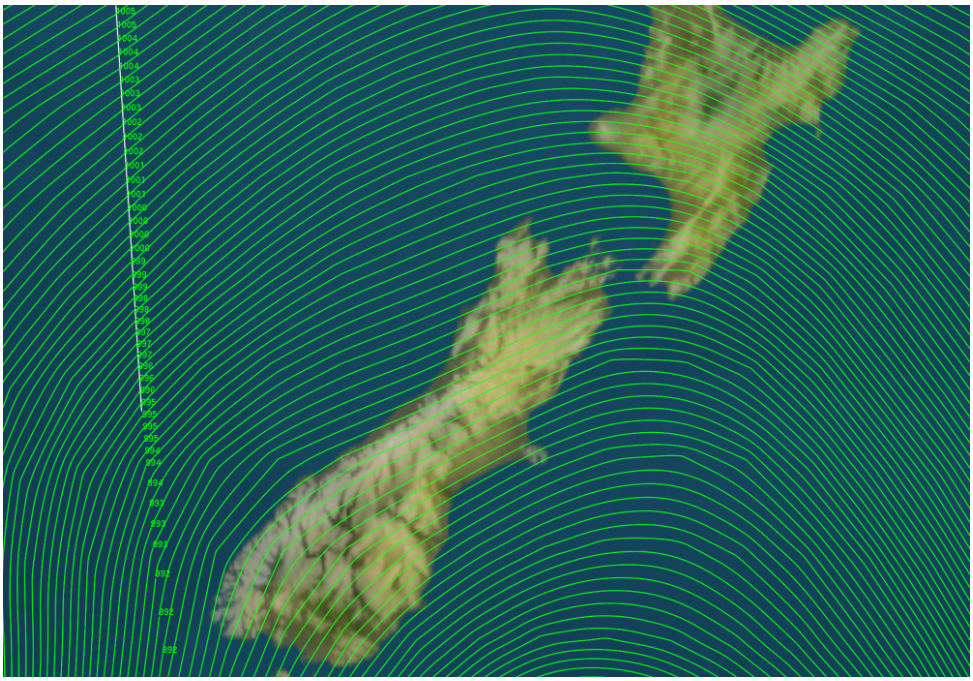


Abbildung 52: Darstellung des Luftdrucks mit einer sehr hohen Isolevelzahl über Neuseeland

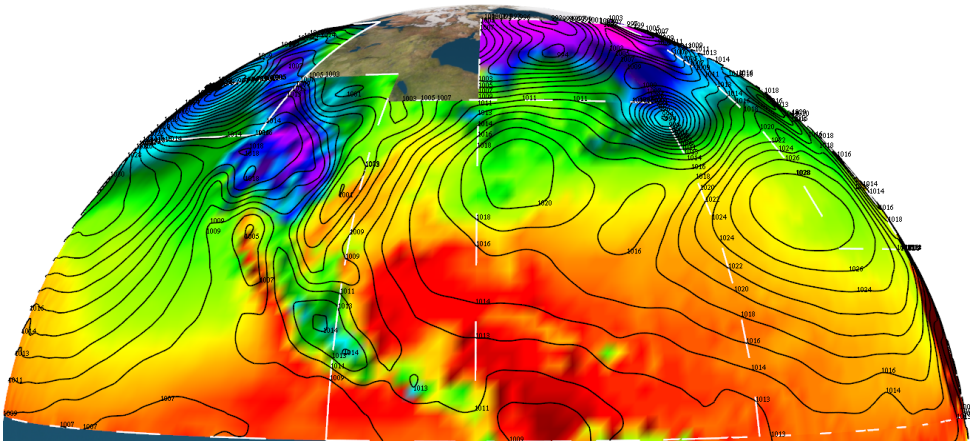
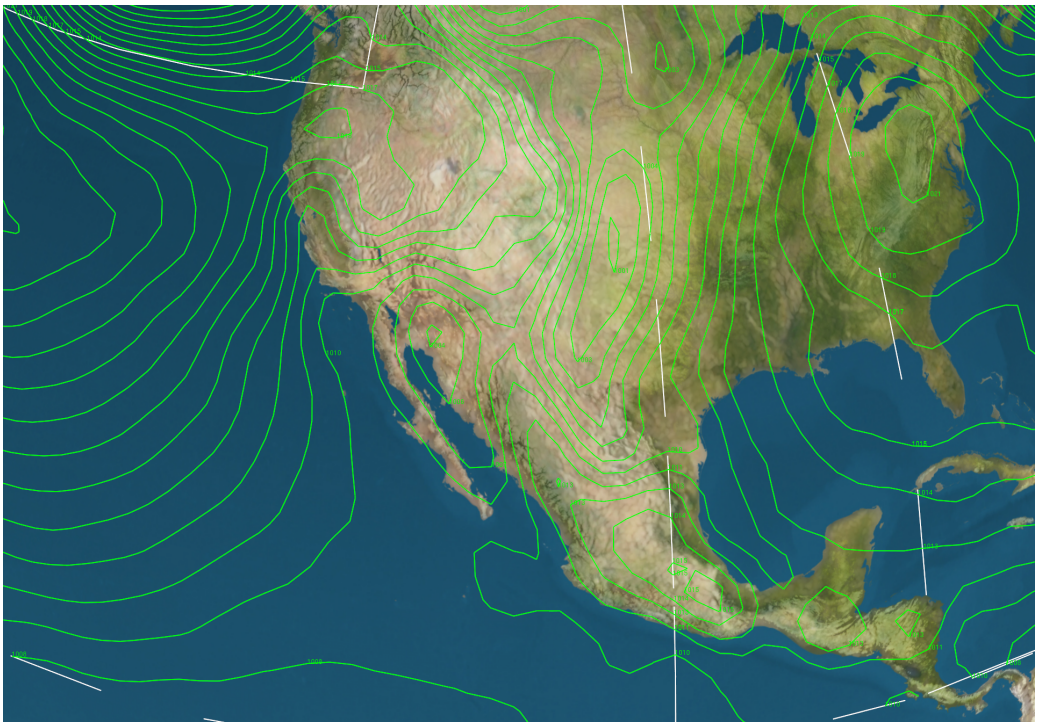
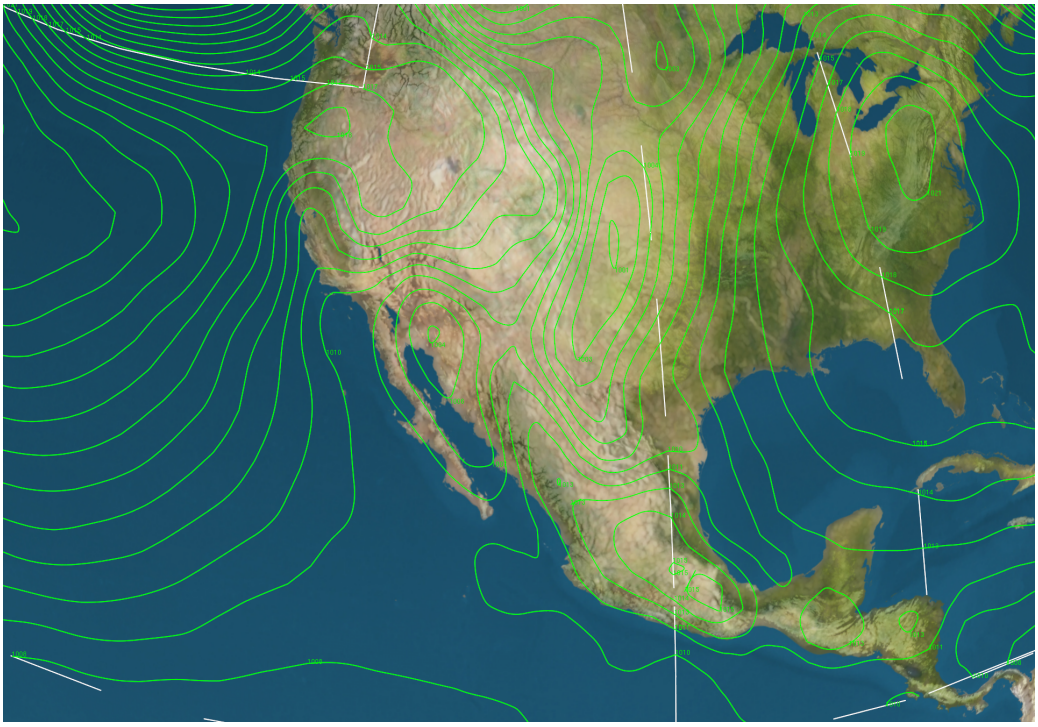


Abbildung 53: Kombinierte Darstellung von Temperatur und Luftdruck



(a) Ungeglättete Isolinien



(b) Durch drei Iterationen des Chaikin Algorithmus geglättete Isolinien

Abbildung 54: Vergleich zur Isolinienglättung aus mittlerer Entfernung anhand der Luftdruckdarstellung. Unterschiede sind nur bei stark gekrümmten Linien erkennbar.

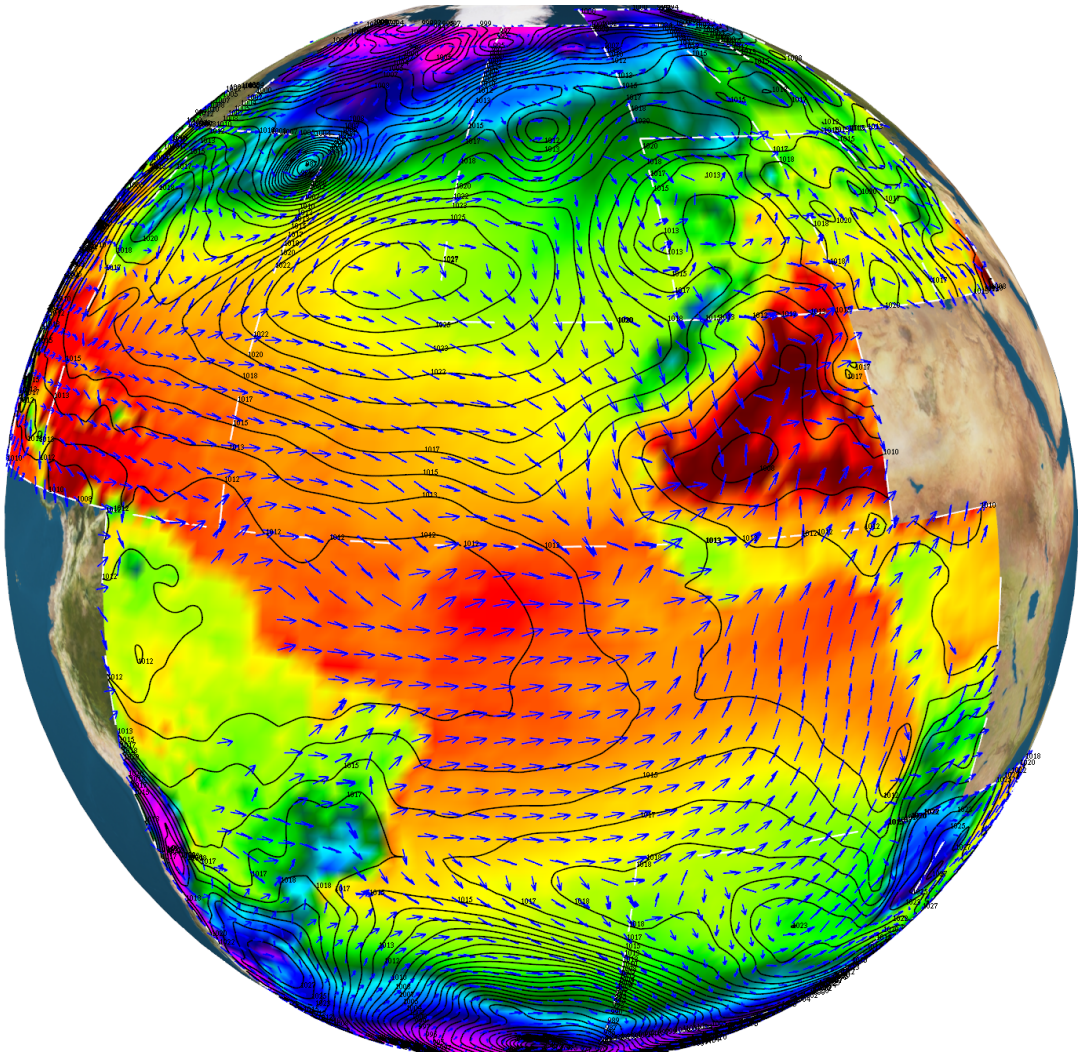


Abbildung 55: Kombinierte Darstellung von Temperatur, Wind und Luftdruck

Performancetesteinstellungsdetails

Für die Performancetests in Kapitel 8 wurde folgende Systemkonfiguration verwendet:

- **CPU: Athlon 64 3500**
- **Speicher: 2 GByte DDR 333 (4*512 MB), Cl 2-2-2-5**
- **Chipsatz: NForce 4 SLI**
- **Grafikkarte: GeForce 8800 GTS mit 320 MByte Speicher**
- **Grafikkartentreiber: ForceWare 158.22**
- **Fensterauflösung:** $1152 \cdot 864$
- **Betriebssystem: Windows XP mit SP 2**

Bei den im Abschnitt Algorithmenperformance des Kapitels 8 visualisierten Daten handelt es sich um Luftdruckdaten von Raymarine.com vom 16. Juni 6 Uhr bis 19.6 2006 6 Uhr in insgesamt 11 Zeitschritten. Dabei wurde ausschließlich der Bereich des GRIB-Files "South Pacific", der in sieben Rechtecke zerteilt ist, in der maximalen Gitterauflösung von $77 \cdot 57$ **für 10 äquidistante Isolevel visualisiert. Die Daten werden aus einer lokal vorliegenden Datenbank geladen, falls nichts anderes angegeben ist.**

Inhalt der CD

Im Anhang befindet sich eine CD mit den folgenden Ordnern:

- **Code**
  Enthält den Quellcode des Applets im Unterordner „EarthWeather3D“ und den Quellcode des Datenaufbereitungsprogramms im Unterordner „GribFile2DBMS“
- **Data**
  Enthält die GribFiles, mit denen das Programm getestet wurde
- **Thesis**
  Enthält dieses Dokument im PDF-Format
- **Tools**
  Enthält die verwendeten JOGL Bibliotheken und den zum Verbinden mit der MySQL Datenbank benötigten JDBC-Treiber

Abbildungsverzeichnis

Literatur

[Bourke] **Bourke, Paul D.**
Conrec - A contouring Subroutine,
BYTE (1987) Nr. 6, S. 143-150

[Catmull] **Catmull, Edwin; Clark, Jim**
Recursively generated B-Spline surfaces on arbitrary meshes,
Computer Aided Design, 1978, Vol. 10, Nr. 6, S. 350-355

[CGP] **Computergrafikpraktikum**
Universität Osnabrück, 2004
http://www-lehre.inf.uos.de/cgp/2004/

[Chaikin] **Chaikin, G.**
An Algorithm for high speed curve generation,
Computer Grafiks & Image Processing 3, 1974, S. 346-349

[Davis] **Davis, Tom; Neider, Jackie; Shreiner, Dave;Woo,Mason**
OpenGL Programming Guide (Redbook), 5.Edition
Addison Wesley, 2006

[Doo] **Doo, Daniel; Sabin, Malcolm**
Behaviour of recursive division surfaces near extraordinary points,
Computer Aided Design, 1978, Vol. 10, Nr. 6, S. 356-360

[DWD] **DeutscherWetterdienst, 2007**
http://www.dwd.de/de/de.htm

[ESRI] **ESRI**
ESRI - The GIS Software leader, 2007,
http://www.esri.com

[Forster] **Otto Forster**
Analysis I
vieweg, Auflage 6, 2001

[Fox] **P. Fox, R. Kunze, D. Langfeld, O. Vornberger**
Wetterdatenübertragung mit Digital Audio Broadcasting
36. Jahrestagung der Gesellschaft für Informatik, 2.-6.10.2006, TU Dresden http://www.inf.uos.de/papers_pdf/2006_06.pdf

[Google] **Google Inc.**
GoogleEarth, 2007
http://earth.google.com

[Java 3D] **Java 3D, 2007**
https://java3d.dev.java.net/

[JOGL] **JOGL**
The JOGL API Project, 2007
https://jogl.dev.java.net

[Kemper] **Kemper, Alfons; Eickler, André**
Datenbanksysteme,
Oldenbourg Verlag München Wien, Auflage 5, 2004

[Kunze] **Kunze, Ralf**
SVG Weather Entwicklung einer SVG Web Mapping Applikation zur Visualisierung von vierdimensionalen Daten am Beispiel von Wettervorhersagedaten
Dissertation an der Universität Osnabrück, 2006
http://elib.ub.uni-osnabrueck.de/cgi-bin/diss/user/catalog?search=sqn&sqn=603&publishid=nobody

[Lampen] **Werner Lampen**
Skript zur Veranstaltung "Analysis II"
Lingen, 1999

[Lingrand] **Lingrand, Diane**
The Marching Cubes, Université de Nice, 2002
http://www.polytech.unice.fr/ lingrand/MarchingCubes/applet.html

[Lorensen] **Lorensen,William E..; Cline, Harvey E.**
Marching Cubes: A High Resolution 3D Surface Construction Algorithm
Computer Graphics (Proceedings of SIGGRAPH 87), Vol. 21, Nr. 4, S. 163-169

[LWJGL] **Lightweight Java Game Library (LWJGL), 2007**
http://www.lwjgl.org/index.php

[Pkware] **Zip Format**
Pkware Inc., 2007
http://www.pkware.com/index.php?option=com_content&task=view&id=64&Itemid=107

[MySQL] **MySQL DBMS**
MySQL AB, 2007
http://www.mysql.com/

[NCEP] **Office Note 388 GRIB**
National Centers For Environmental Prediction, 2005
http://www.nco.ncep.noaa.gov/pmb/docs/on388/

[nVidia] **GoForce family of handheld GPUs**
NVIDIA Corporation, 2007
http://www.nvidia.com/page/handheld.html

[OGL] **OpenGL**
The OpenGL API, 2007
http://www.opengl.org/

[OGLES] **OpenGL for Embedded Systems**
The OpenGL ES API, 2007
http://www.khronos.org/opengles/

[PINGO] **Deutsches Klimarechenzentrum:Wegner, Jörg**
The PINGO Homepage, 2001
http://www.mad.zmaw.de/Pingo/pingohome.html

[Rahm] **de Rahm, G.**
Un peu de mathématique à propos d'une courbe plane
Elemente der Mathematik 2, 1947, S. 73-76 und 89-97

[Ray] **Raymarine Inc.**
3 Day RayTech Weather Forecast, 2007
http://www.raymarine.com

[Rost] **Randi J. Rost, et al.**
OpenGL Shading Language, Second Edition
Addison-Wesley, 2006

[Schwarz] **Schwarz, Hans Rudolf**
Numerische Mathematik, 5. Auflage
B.G. Teubner, Stuttgart 2004

[Shreiner] **Dave Shreiner**
OpenGL Reference Manual (Blue Book), Version 1.4
Addison Wesley, 2004

[Snyder] **Snyder,William V.**
Algorithm 531 - Contour Plotting,
ACM Trans. on Math. Software, Vol. 4, No. 3, S.290, September 1978

[Sun] **Sun Microsystems, 2007**
http://www.sun.com/

[Sun2] **Java Webstart**
Sun Microsystems, 2007
http://java.sun.com/products/javawebstart/

[Vor] **Vornberger, Oliver**
Computergrafik, 2006
Universität Osnabrück
http://www-lehre.inf.uos.de/ cg/2006/PDF/skript.pdf

[W3C] **WorldWideWeb Consortium**
Scalable Vector Graphics (SVG) - XML Graphics for the Web, 2003
http://www.w3.org/TR/SVG11/

[Walch] **Walch, Dieter; Frater, Harald**
Wetter und Klima: Das Spiel der Elemente - Atmosphärische Prozesse verstehen und deuten
Springer Verlag, Berlin Heidelberg 2004

[Wenke] **Wenke, Henning**
3D Klimadatenvisualisierung mit OpenGL

Bachelorarbeit an der Universität Osnabrück, 2005
http://www.inf.uos.de/prakt/pers/dipl/hwenke

[WGRIB] **Climate Prediction Center: Ebisuzaki,Wesley**
Climate Prediction Center - wgrib home page, 2005
http://www.cpc.ncep.noaa.gov/products/wesley/wgrib.html

[Wp1] **Wikipedia**
Displacement Mapping, 2007
http://en.wikipedia.org/wiki/Displacement_mapping

[Wp2] **Wikipedia**
Java 3D, 2007
http://de.wikipedia.org/wiki/Java_3D

[Wp3] **Wikipedia**
Subdivision surface, 2007
http://en.wikipedia.org/wiki/Subdivision_surfaceD

[Wp4] **Wikipedia**
3dfx Voodoo Graphics, 2007
http://de.wikipedia.org/wiki/Voodoo_Graphics

[Wp5] **Wikipedia**
LZ78, 2007
http://de.wikipedia.org/wiki/LZ78

Erklärung

Hiermit erkläre ich, dass ich die Bachelorarbeit selbstständig angefertigt und keine Hilfsmittel außer denen in der Arbeit angegebenen benutzt habe.

Osnabrück, den ..... **.....**

(Henning Wenke)