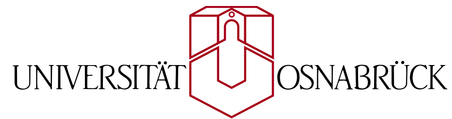


Institut für Informatik
AG Medieninformatik

Masterarbeit

Interaktive 3D Visualisierung von Wetterdaten mit Open GL ES und WebGL

Erik Wittkorn

Februar 2014

Erstgutachter:	Prof. Dr. Oliver Vornberger
Zweitgutachterin:	Prof. Dr.-Ing. Elke Pulvermüller

Danksagungen

Hier möchte ich mich bei allen Personen bedanken, die zur Fertigstellung der Arbeit beigetragen haben:

- Herrn Prof. Dr. Oliver Vornberger für die Tätigkeit als Erstgutachter und für die Bereitstellung der interessanten Thematik.
- Frau Prof. Dr.-Ing. Elke Pulvermüller, die sich als Zweitgutachterin zur Verfügung gestellt hat.
- Henning Wenke, M.Sc., der sich für die Betreuung der Arbeit viel Zeit nahm und mich durch offene und lockere Gespräche unterstützt hat.
- Alle Personen, die beim Korrekturlesen geholfen haben.

Zusammenfassung

Viele Menschen nutzen Smartphones und Notebooks, um sich täglich über die aktuelle Wettervorhersage zu informieren. Diese Arbeit beschreibt den Entwicklungsprozess eines Programms, das neuartige Formen der Darstellung einer animierten globalen Wettervorhersage bietet. Es stellt zum Beispiel die Temperatur anhand eines Farbspektrums und den Luftdruck durch Isolinien auf einem dreidimensionalen Globus dar. Ein Vorteil der Anwendung gegenüber vergleichbaren Angeboten ist ein hoher Grad an Interaktivität. Der Nutzer kann dadurch viele Eigenschaften der Visualisierung an seine persönlichen Bedürfnisse anpassen. Eine weitere Anforderung, die sich aus der Entwicklung für mobile Geräte ergibt, ist die ökonomische Nutzung der Datenverbindung.

Durch die Verwendung von modernen Technologien für die hardwarebeschleunigte Grafikdarstellung konnte diese Anwendung für iOS-Geräte und den Webbrowser umgesetzt werden. Da die Visualisierungen nicht vorberechnet sind, sondern in Echtzeit aus den Ursprungsdaten generiert werden, können Parameter wie Farben und die Darstellungsqualität zur Laufzeit verändert werden. Es wurde zusätzlich eine Technik entwickelt, um die Datenteilmengen zu bestimmen, die zur Anzeige und Animation der Wettervorhersage benötigt werden. Dadurch können sie gezielt nachgeladen werden, anstatt schon beim Start der Anwendung unnötig viel Datenvolumen zu beanspruchen. Eine Evaluation auf verschiedenen Plattformen zeigte abschließend, dass die Applikationen in guter Qualität auf mobilen Geräten genutzt werden können. Mit diesen Eigenschaften können die Ergebnisse dieser Arbeit für viele Menschen einen neuen Blick auf das globale Wettergeschehen ermöglichen.

Abstract

Many people use smartphones and notebooks to learn about the current weather forecast daily. This work describes the development process of a program which offers new forms of an animated global weather forecast. It presents, for example, the temperature on the basis of a color spectrum and the air pressure as isolines on a three-dimensional globe. A high degree of interactivity is an advantage in comparison to similar offers. The user can customize many parameters of the visualization to his personal needs. A further requirement in developing for mobile devices, is the economic use of the data connection.

Taking advantage of modern technology for hardware-accelerated graphics that render this application were implemented for iOS-devices and the web browser. Since the visualizations are not precomputed, but generated in realtime from the original data, parameters such as colors the quality of display can be changed at runtime. Additionally a technique was developed to determine the subsets of data that are required for the display and animation of the weather forecast. This allows them to be selectively loaded instead of using unnecessarily large volume of data at application startup. An evaluation on different platforms showed conclusively that the applications can be used in good quality on mobile devices. Based on these features, the results of this work can provide a new perspective on the global weather patterns for many people.

Inhaltsverzeichnis

1 Einleitung **1**
1.1 Motivation . . . 1
1.2 Vorarbeiten und Abgrenzung . . . 2
1.3 Zielsetzung und Vorgehen . . . 4

2 Wetterdaten und -visualisierung **6**
2.1 Beschaffenheit und Nutzung von Wetterdaten . . . 6
2.2 Verfügbarkeit der Daten . . . 8
2.3 Visualisierungsarten . . . 9

3 Stand der Technik **12**
3.1 Web Apps . . . 12
3.2 iOS Apps . . . 13

4 Konzeption **16**
4.1 Funktionale Anforderungen . . . 16
4.2 Anforderungen der Zielplattformen . . . 17
4.3 Grafische Oberfläche . . . 18
4.4 Server . . . 18
4.5 Fazit . . . 19

5 Technische Grundlagen **20**
5.1 OpenGL . . . 20
5.2 OpenGL ES und WebGL . . . 24
5.3 Objective-C und Cocoa Touch . . . 26
5.4 JavaScript und HTML5 . . . 30
5.5 Das GRIB-Format . . . 34

6 Umsetzung **35**
6.1 Server . . . 35
6.1.1 Abfrage und Verarbeitung der Daten . . . 35
6.1.2 Speicherung und Bereitstellung der Daten . . . 36
6.2 Programmablauf und -struktur . . . 37
6.2.1 Allgemeiner Ablauf und Klassenstruktur . . . 37
6.2.2 Zeitliche Interpolation der Daten . . . 42
6.2.3 Dynamische Bestimmung von sichtbaren Datenteilmengen . . . 44
6.3 Visualisierung der Wetterdaten . . . 46
6.3.1 Räumliche Interpolation und Approximation . . . 46

6.3.2 Farbspektrum 47
6.3.3 Isolinien 49
6.3.4 Partikel 51
6.3.5 Spezielle Symbole 53
6.3.6 Shadergenerierung 55
6.4 Benutzeroberfläche und Interaktion 56
6.4.1 Klassenstruktur der OpenGL-Oberfläche 56
6.4.2 Plattformunabhängige Elemente 58
6.4.3 Plattformspezifische Elemente 60
6.5 Kommentar zur Umsetzung auf den Zielplattformen 61
6.5.1 Web-Plattform 61
6.5.2 iOS-Plattform 62

7 Tests und Evaluation **65**
7.1 Testsysteme 65
7.2 Performanz 65
7.2.1 iOS-Plattform 66
7.2.2 WebGL-Plattform 67
7.2.3 Fazit zur Performanzanalyse 67
7.3 Datenübertragung 68

8 Fazit und Ausblick **70**

A Screenshots **72**

B Inhalt der CD und URL **74**

C Abbildungsverzeichnis **74**

D Literaturverzeichnis **76**

Abkürzungsverzeichnis

API Application Programming Interface
App Applikation
Apps Applikationen
ARC Automatic Reference Counting
ECMWF European Centre for medium-range weather forecasts
ER Entity Relationship
FS Fragment Shader
FPS Frames per second
GFS Global Forecast System
GLESSL OpenGL ES Shading Language
GLSL OpenGL Shading Language
GRIB GRidded Binary
GUI Graphical User Interface
iOS iPhone Operating System
JS JavaScript
JSON JavaScript Object Notation
JOGL Java Bindings for OpenGL
M3G Java Mobile 3D Graphics API
MVC Model View Controller
NDC Normalized Device Coordinates
NOAA National Oceanic and Atmospheric Administration
ObjC Objective-C
OpenGL Open Graphics Library
OpenGL ES OpenGL for Embedded Systems
PRMSL Pressure Reduced to Mean Sea Level
SGI Silicon Graphics, Inc.
TZ Texturzugriffe
TMP Temperature
TCDC Total Cloud Cover
WebGL Web Graphics Library
WMO World Meteorological Organization
VS Vertex Shader

Kapitel 1

Einleitung

Innovative Formen der Datenvisualisierung erfreuen sich eines anhaltenden Interesses, da ein Großteil der Bevölkerung täglich mit grafikfähigen Computern arbeitet und die Freizeit verbringt.[1] Diese Nachfrage wird auch an mobile Geräten gestellt, da ihre Verbreitung enorm steigt und sie mit leistungsfähigen Grafikprozessoren ausgestattet sind. Vor allem auf Reisen ist eine Information über die aktuelle Wetterlage oder Vorhersagedaten der nächsten Tage besonders nützlich. Mit der voranschreitenden Entwicklung des mobilen Internets ist außerdem die Übertragung von immer detaillierteren Daten möglich. Für die Darstellung von globalen Wetterdaten haben sich mit der Zeit verschiedene Techniken und Angebote etabliert. Diese Arbeit untersucht den aktuellen Stand dieser Visualisierungsformen und wie sie mit Hilfe von modernen Technologien auf verschiedenen mobilen Plattformen umgesetzt werden können. Dazu wird ein Programm zur Darstellung von globalen Wetterdaten konzipiert und auf unterschiedlichen mobilen Plattformen umgesetzt.

1.1 Motivation

Um sich über die Wettervorhersage der kommenden Tage zu informieren, werden typischerweise die Nachrichten des Fernsehens, der Zeitung, des Radios und des Internets genutzt. Letzteres wird immer häufiger auf modernen mobilen Computern wie Smartphones und Tablets abgefragt. Nach einer Studie ist der weltweite Verkauf von Smartphones vom zweiten Quartal 2012 zum zweiten Quartal 2013 mit einem Wachstum von 47 Prozent auf 230 Millionen Einheiten pro Quartal gestiegen.[2] In der vorliegenden Arbeit wird eine Applikation (App) für Apple-Geräte entwickelt, deren Anteil beim Smartphone-Verkauf im Q2 2013 bei 13,6 Prozent liegt. In den letzten 5 Jahren, seit denen der App Store als Marktplatz für Applikationen (Apps) auf Apple-Geräten existiert, belegt eine Wetter-App den siebten Platz der kostenlos geladenen Apps.[3]

Mit der wachsenden Popularität von mobilen Geräten geht auch die Entwicklung ihrer Hardware einher. Aktuell werden Quad-Core CPUs mit bis zu 1,9 GHz, Arbeitsspeicher bis zu 2 GB und

[1] http://www.bitkom.org/de/markt_statistik/64050_65137.aspx

[2] http://www.prnewswire.com/news-releases/strategy-analytics-global-smartphone-shipments-hit-record-230-million-units-in-q2-2013-217044481.html

[3] http://home.bt.com/techgadgets/techfeatures/app-store-turns-five-the-most-popular-apps-revealed-11363815200184

leistungsfähige Grafikprozessoren in Smartphones verbaut.[4] Seit 2007 steht mit OpenGL for Embedded Systems (OpenGL ES) die mobile Version der beliebten 3D-Grafik-Bibliothek Open Graphics Library (OpenGL) in der Version 2.0 zur Verfügung. Durch ihre Nutzung ergeben sich Möglichkeiten der hardwarebeschleunigten Datenvisualisierung, wie sie vor einigen Jahren auf mobilen Geräten kaum denkbar waren. Sie dient auch als Basis für die 2011 veröffentlichte Spezifikation der Web Graphics Library (WebGL). Mit dieser Bibliothek wird die Plattformunabhängigkeit von hardwarebeschleunigter 3D-Grafik vorangetrieben. Der Standard wird von vielen Browser-Herstellern implementiert und kann somit auf einer Reihe von Desktop- und mobilen Systemen mit diesen Browsern genutzt werden.

In dieser Arbeit sollen die Möglichkeiten der 3D Visualisierung in Echtzeit auf verschiedenen mobilen Geräten unter Einsatz der genannten Techniken untersucht werden. Die Anforderungen einer interaktiven Visualisierung von Wettervorhersagedaten über einen zeitlichen Verlauf eignen sich dabei gut, um verschiedene Systeme miteinander zu vergleichen und sind sinnvoll für den praktischen Gebrauch. Für diesen Vergleich soll jeweils eine App für die beiden genannten Plattformen entstehen, die für Endnutzer bedienbar ist und durch Nutzung von modernen Technologien gegenüber der existierenden Apps eine Innovation darstellt. Die Ergebnisse dieser Arbeit sollen anderen Entwicklern als Hinweise für die leistungsfähige Arbeit mit diesen Techniken dienen.

1.2 Vorarbeiten und Abgrenzung

In den letzten Jahren haben sich an der Universität Osnabrück andere Arbeiten mit der interaktiven 3D Visualisierung von Wetterdaten auf verschiedenen Plattformen beschäftigt. In diesem Abschnitt werden deren Ergebnisse und der Bezug zu der vorliegenden Arbeit zusammengefasst.

Earth Weather 3D - Visualisierung und Animation dynamisch bestimmter Teilmengen von Wetterdaten auf Webseiten mit OpenGL, 2007

Die Masterarbeit [13] von Wenke baut auf einem C++ Programm auf, das in seiner Bachelorarbeit [14] entwickelt wurde. Es wurde nun als Java-Applet umgesetzt, das auf java-fähigen Systemen aus dem Browser heraus gestartet werden kann. Mithilfe der Java Bindings for OpenGL (JOGL)-Bibliothek wurde mit OpenGL 1.5 gearbeitet.

Das Programm bietet, wie Abbildung 1.1 darstellt, unter anderem die Anzeige und zeitliche Animation von Daten einer Wettervorhersage auf einer 3D-Weltkugel in folgender Ausprägung:

Temperatur: eingefärbte Flächen der Erdoberfläche
Luftdruck: beschriftete Isolinien, die Flächen eines Luftdruckintervalls einschließen
Wind: Pfeile, deren Richtung und Länge die Windrichtung beziehungsweise -stärke darstellen
Bewölkungsgrad: beispielhafte Darstellung mit Grautönen als Schleier über der Erdoberfläche

Die Wetterdaten werden aus GRidded Binary (GRIB)-Dateien gelesen und nach der Einteilung in geografische Teilmengen in einer Datenbank abgelegt. Für eine stetige Animation einer Vorhersage wird zwischen aufeinander folgenden Zeitpunkten linear interpoliert und fehlende Daten für den aktuell sichtbaren Ausschnitt der Weltkugel dynamisch nachgeladen. Die Interaktion

[4] http://en.wikipedia.org/wiki/Comparison_of_smartphones

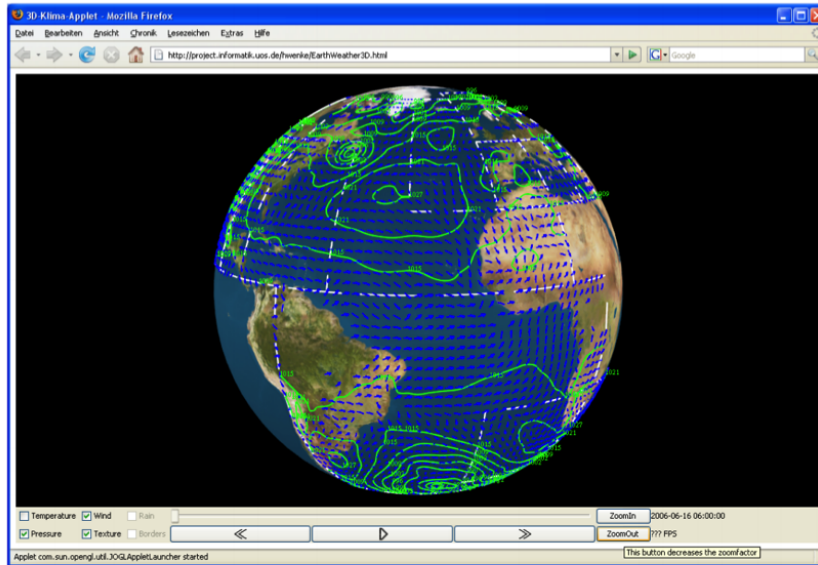


Abbildung 1.1: Visualisierung von globalen Wetterdaten (Luftdruck und Wind) nach [13]

erfolgt dabei durch die Maus mit Graphical User Interface (GUI)-Elementen. Damit kann der Nutzer einerseits die Anzeige von einzelnen Wetterelementen an- und ausschalten und andererseits den Blickwinkel auf den Globus steuern. Bei sich überlappenden Datensegmenten kann das dominierende eingestellt werden.

Interactive visualization of weather data on smartphones with M3G, 2009

In der Masterarbeit [3] von Engelhardt wurde die interaktive 3D-Visualisierung von Wetterdaten als Java MIDlet für das Nokia N95 mit der auf OpenGL ES 1.1 basierenden Java Mobile 3D Graphics API (M3G) 1.1 umgesetzt.

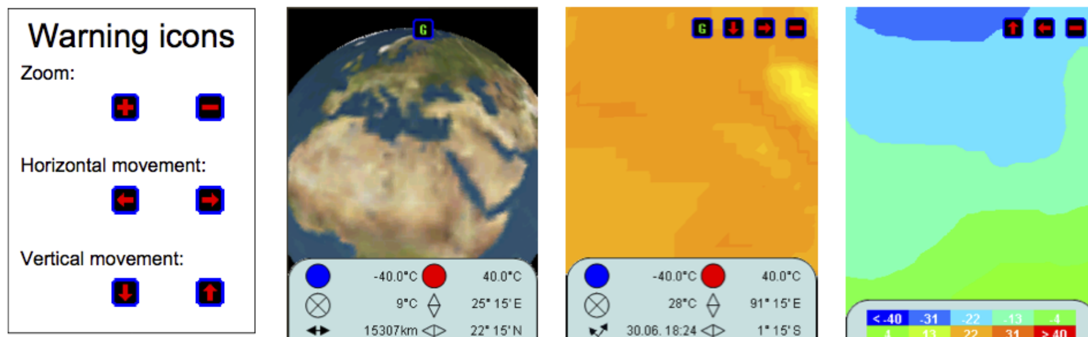


Abbildung 1.2
Ausschnitte der M3G-Applikation aus [3] (Interface, 3D-Globus, Temperatur, Temperatur)

Die Anzeige einer dreidimensionalen Weltkugel mit dynamisch erzeugter Wettervisualisierung ist trotz der eingeschränkten Hardware realisiert worden. In Abbildung 1.2 werden exemplarische Screenshots der App gezeigt. Als Teilmenge der Wetterdaten wird die Temperatur durch eingefärbte Flächen der Erdkugel dargestellt. Um die große Datenmenge anzeigen zu können, wurden Methoden entwickelt, die die verfügbaren Daten in kleinere Teilmengen einteilen und be-

stimmen, welche davon zur Anzeige des aktuell sichtbaren Ausschnitts der Weltkugel nötig sind. Für die weitere Leistungsoptimierung wird auch berechnet, welche Teilmengen für die Navigation zum nächsten Ausschnitt benötigt werden, um einen nahtlosen Übergang zu ermöglichen. Diese Daten werden über das mobile Internet geladen und auf dem Smartphone zwischengespeichert. Die Interaktion mit dem Programm geschieht über die Tasten des Smartphones, die eine räumliche Navigation um den Globus nach Norden, Süden, Osten und Westen und die Auswahl einer Zoomstufe ermöglichen.

Die vorliegende Arbeit stützt sich auf Erkenntnisse der vorgestellten Abhandlungen und ein enger Bezug ist an den entsprechenden Stellen kenntlich gemacht worden. Um zu neuen Erkenntnissen zu gelangen, gibt es folgende Abgrenzungskriterien, die im weiteren Verlauf der Arbeit erläutert werden:

- Die Techniken OpenGL ES 2.0 und und WebGL unterscheiden sich nach Weiterentwicklung der Technologien grundlegend von JOGL und M3G.
- Die Rechenleistung und Grafikqualität von mobilen Geräten hat sich in den vergangenen Jahren gesteigert. Dies soll genutzt werden, um neue Formen der Visualisierung umzusetzen.
- Die Implementierung für ein Touchscreen-Gerät, wie dem Apple iPhone, erfordert neue Überlegungen für die Nutzer-Interaktion mit dem Programm.
- Durch die Plattformunabhängigkeit von WebGL und der vielseitigen Familie der iPhone Operating System (iOS)-Geräte, die Smartphones und Tablets umfasst, ist ein Vergleich der Leistung von verschiedenen mobilen Geräten möglich.

1.3 Zielsetzung und Vorgehen

Wie in den vorherigen Abschnitten eingeleitet wurde, ist das Ziel dieser Arbeit zunächst die Analyse der aktuell genutzten Visualisierungstechniken und -angeboten von globalen Wettervorhersagedaten. Darauf aufbauend wird erarbeitet, wie moderne Technologien für hardwarebeschleunigte Grafikdarstellung, wie OpenGL ES 2.0 und WebGL, auf verschiedenen mobilen Systemen und Desktop-Plattformen in diesem Anwendungsbereich eingesetzt werden können, um vor allem einen neuen Grad der interaktiven Möglichkeiten zu erreichen. Abschließend soll evaluiert werden, wie sich die Anwendung auf verschiedenen mobilen Geräten verhält.

Für diesen Vergleich werden zwei Apps entwickelt. Zum einen wird eine native iPhone-App entwickelt, die OpenGL ES 2.0 für die hardwarebeschleunigte Darstellung nutzt. Diese App ermöglicht den Vergleich auf Geräten wie dem iPhone und iPad. Zum anderen wird eine App mit HMTL5 und WebGL umgesetzt und ermöglicht die Nutzung auf Desktop- und mobilen Systemen, mit der Voraussetzung, dass ein kompatibler Browser verwendet wird. Bei der Umsetzung soll auch betrachtet werden, welche Gemeinsamkeiten bei den Plattformen bestehen und wie hoch der Portierungsaufwand ist.

Um diese Ziele zu erreichen, wird zunächst erläutert, warum sich vor allem globale Wetterdaten für die hardwarebeschleunigte Visualisierung auf mobilen Geräten eignen. Dazu werden allgemeine Informationen über die Beschaffenheit von Wettervorhersagedaten und ihre aktuelle Verfügbarkeit dargestellt. Darauf aufbauend wird beschrieben, welche Techniken bei der Visualisierung dieser Daten angewandt werden können und einen Nutzen bringen. Abschließend wird der zentrale Begriff der interaktiven Wettervisualisierung definiert.

Im Stand der Technik werden aktuell verfügbare Apps präsentiert, die sich im Anwendungsbereich der interaktiven Wettervisualisierung befinden. Dazu werden Stellvertreter ausgewählt, die mit OpenGL ES für das iPhone beziehungsweise mit HTML5 und WebGL für den Web-Browser umgesetzt wurden.

Auf der Basis dieser Informationen werden die Anforderungen an die Apps herausgearbeitet, die bei der Umsetzung berücksichtigt werden. Dies sind zum einen Funktionalitäten, die aus Nutzersicht Sinn machen, und zum anderen Einschränkungen und Möglichkeiten der Plattformen, auf denen entwickelt werden soll. Ein Auszug daraus sind folgende Punkte:

- visuelle Qualität
- benötigtes Onlinedatenvolumen
- Möglichkeiten der Nutzerinteraktion

Mit diesem Wissen wird ein Fazit gezogen, in welchen Bereichen die Anforderungen dieser Arbeit über den Stand der Technik hinausgehen.

Im anschließenden Kapitel werden technische Grundlagen vorgestellt, die zum Verständnis der Umsetzung nötig sind. Die Grafikbibliothek OpenGL wird vorgestellt und eine Einordnung der mobilen Varianten OpenGL ES 2.0 und WebGL vorgenommen. Es wird zusammengefasst, welche speziellen Funktionen der verwendeten Programmiersprachen, Frameworks und integrierten Bestandteile der Plattform für die Apps nützlich sind. Unter den Sprachen ist zum einen Objective-C (ObjC) 2.0 auf den iOS Geräten und zum anderen die Verbindung von HTML5 und JavaScript für die Entwicklung von Browser-Apps. Abschließend wird das Format der verwendeten Wetterdaten vorgestellt.

Bei der Umsetzung wird zunächst gezeigt, welche Struktur den Apps zu Grunde liegt und nach welchen Richtlinien sie entworfen wurden. Die dabei entwickelten Komponenten kommunizieren zur Laufzeit, um die geplanten Funktionen zu realisieren. Die für die hardwarebeschleunigte Visualisierung wichtigen Komponenten werden anschließend nach dem Server gesondert betrachtet und ihre Funktionsweise beschrieben. Darauf folgend wird die Benutzeroberfläche vorgestellt und auf plattformspezifische Besonderheiten hingewiesen.

In der Evaluation werden die Programme unter normalen Benutzungsbedingungen getestet, um herauszufinden, in welcher Qualität sie auf aktueller Hardware verwendet werden können. Dazu wird zum einen die Performanz und zum anderen die benötigte Datenmenge für die Darstellung einer animierten Wettervisualisierung untersucht.

Abschließend werden die Ergebnisse dieser Arbeit im Hinblick auf den aktuellen Stand der Technik im Fazit zusammengefasst. Es wird zudem ein Ausblick mit technischen und funktionalen Perspektiven für die zukünftige Weiterentwicklung der entwickelten Technologien gegeben.

Kapitel 2

Wetterdaten und -visualisierung

„Als Wetter ... bezeichnet man den spürbaren Zustand der Atmosphäre ... an einem bestimmten Ort der Erdoberfläche, der unter anderem als Sonnenschein, Bewölkung, Regen, Wind, Hitze oder Kälte in Erscheinung tritt.“[5]

In den folgenden Abschnitten werden die Eigenschaften der Wetterdaten erläutert, die in dieser Arbeit dargestellt werden. Es wird kurz beschrieben, in welcher Form die Daten normalerweise vorliegen, welchen Bedingungen sie unterliegen und wie sie vom Endanwender genutzt werden können. Anschließend wird geklärt, von welcher Quelle entsprechende Daten bezogen werden können und darauf aufbauend die verschiedene Visualisierungsarten vorgestellt.

2.1 Beschaffenheit und Nutzung von Wetterdaten

Meteorologische Wetterdaten bestehen aus verschiedenen Elementen wie der Lufttemperatur, Windrichtung und -stärke oder der Bewölkung. Diese werden jeweils an unterschiedlichen Orten und Höhenlagen mit verschiedenen Geräten gemessen und in entsprechenden Einheiten (z. B. Temperatur in °*Celsius* und Luftdruck in *Pascal*) angegeben. Da diese Elemente einer Dynamik unterliegen und sich teilweise gegenseitig beeinflussen, ist die Speicherung des zeitlichen Verlaufs vor allem globaler Daten von Interesse. Aus den historischen Daten von mehreren Jahren lässt sich z. B. eine Klimaanalyse erstellen, um langfristige Veränderungen in großen Klimazonen, wie die globale Erwärmung und ihre Auswirkungen auf das sibirische Klima[6], zu erkennen. Außerdem wurden von Meteorologen verschiedene numerische Vorhersagemodelle entwickelt, die aufgrund historischer und tagesaktueller Wetterdaten durch computergestützte Simulation eine Einschätzung des zukünftigen Wetters geben können. In folgender Beschreibung werden die beiden bekanntesten Modelle für eine globale Vorhersage vorgestellt.

[5] de.wikipedia.org/wiki/Wetter

[6] http://de.wikipedia.org/wiki/Globale_Erw%C3%A4rmung#.C3.96rtliche_und_zeitliche_Verteilung_der_beobachteten_Erw.C3.A4rmung

European Centre for medium-range weather forecasts (ECMWF)-Modell: Dieses Modell ist nach Lynch [8] die wahrscheinlich wichtigste Einrichtung in der europäischen Meteorologie und weltweit führend in der numerischen Wettervorhersage. Seit der ersten Vorhersage im August 1979 wird das Modell ständig weiterentwickelt, um die Genauigkeit zu verbessern. Es erzeugt vielseitige Produkte, unter denen sich z. B. eine 10-Tages Vorhersage mit einer horizontalen Auflösung auf der Erdoberfläche von 25 km und eine Vorhersage über die nächsten 6 Monate mit einer Auflösung von 125 km befinden.

Global Forecast System (GFS)-Modell:[7] Dieses Modell wird von der amerikanischen Einrichtung National Oceanic and Atmospheric Administration (NOAA)[8] verwendet und weiterentwickelt. Mit diesem Modell wird alle sechs Stunden eine Vorhersage berechnet, welche eine horizontale Auflösung von 27 km hat und Daten im 3-Stunden-Takt für die nächsten acht Tage liefert. Die nächsten 16 Tage werden mit einer Auflösung von 35 km im 12-Stunden-Takt ausgeliefert. Für 2014 ist ein Upgrade auf eine Auflösung von 13 km für die nächsten zehn Tage geplant.

Die so berechneten Wettervorhersagedaten werden von verschiedenen Anbietern eingekauft und an den Endnutzer weitergegeben. In Abbildung 2.1 ist die Wettervorhersage der eingebauten App eines iPhones zu sehen. Sie zeigt die aktuellen Werte für Temperatur, Luftfeuchtigkeit, Regenwahrscheinlichkeit, Windrichtung und -stärke an einem Ort an. Diese Daten werden für die nächsten 12 Stunden im stündlichen Intervall und für die nächsten fünf Tage im täglichen Abstand zur Verfügung gestellt.

Abbildung 2.1
Wetter-App in iOS 7

Die Vielseitigkeit der angebotenen Wetterdaten lässt sich dadurch erklären, dass es verschiedene Interessengruppen bei der Nutzung von Wetterdaten gibt. Es gibt z. B. die Gruppe der Sportsegler oder -flieger, die sich vor allem für die Windrichtung und -stärke interessiert, wobei diese Information für einen Urlaubsreisenden nicht so wichtig ist, wie es die Temperatur am angestrebten Ort sein kann. Diese Nutzer profitieren besonders von der mobilen Aktualisierung ihrer Vorhersagedaten.

Neben der Auswahl einer interessanten Teilmenge der Wetterelemente, sollte ein Nutzer zur Wetterinformation neben der lokalen Auskunft für einen Ort auch globale Daten mit einbeziehen können. Die meisten verfügbaren Apps liefern lokale Wetterinformation, aber aufgrund der vorher genannten Wechselbeziehungen der Wetterelemente ist es oft sinnvoll, globale Zusammenhänge im zeitlichen Verlauf zu interpretieren. Mit Informationen über den vorherigen Bewölkungsgrad und Windverlauf lässt sich z. B. besser abschätzen, wann eine Regenfront einen entsprechenden Ort erreicht.

[8] `http://www.noaa.gov/`

2.2 Verfügbarkeit der Daten

GRIB Filter

For GRIB data you have to option to filter the data.

Extract Levels and Variables

You may select some or all levels and variables. The selections below represent common choices which may or may not be relevant to the files that you have selected. For example choosing RH (relative humidity) would be pointless in file of sea-surface temperatures. ...

Select the levels desired:

☐ all ☐ 1 mb ☐ 2 mb ☐ 3 mb ☐ 5 mb ☐ 7 mb ☐ 10 mb ☐ 20 mb ☐ 30 mb ☐ 50 mb ☐ 70 mb ☐ 100 mb ☐ 125 mb ☐ 150 mb ☐ 175 mb ☐ 200 mb ☐ 225 mb ☐ 250 mb ☐ 0-0.1 m below ground ☐ 0.1-0.4 m below ground ☐ 0.4-1 m below ground ☐ 1-2 m below ground ☐ surface ☑ 2 m above ground ☐ 10 m above ground ☐ 80 m above ground ☐ 100 m above ground ☐ mean sea level ☐ 305 m above mean sea level ☐ 457 m above mean sea level

Select the variables desired:

☐ all ☐ 4LFTX ☐ 5WAVA ☐ 5WAVH ☐ ABSV ☐ PRATE ☐ PRES ☑ PRMSL ☐ PWAT ☐ RH ☐ SHTFL ☐ SNOD ☐ SOILL ☐ SOILW ☐ SPFH ☐ SUNSD ☑ TCDC ☐ TMAX ☐ TMIN ☑ TMP ☐ TOZNE ☐ U-GWD ☐ UFLX ☐ UGRD ☐ ULWRF ☐ USTM ☐ USWRF ☐ V-GWD ☐ VFLX ☐ VGRD ☐ VRATE ☐ VSTM ☐ VVEL ☐ VWSH ☐ WATR ☐ WEASD

Abbildung 2.2: Auswahloptionen des Wettervorhersagedaten-WebService der NOAA, es sind Luftdruck, Wolkendichte und Temperatur in zwei Metern über der Meeresoberfläche ausgewählt[9]

Von den beiden vorgestellten Wetter-Organisationen stellt nur die NOAA ihre Berechnungen kostenlos zur Verfügung. Von der ECMWF können kostenlos nur vorberechnete Visualisierungen ohne Nutzen für die weitere Verarbeitung bezogen werden. Die nach dem GFS-Modell berechneten Daten werden über einen Web-Service online bereitgestellt. Sie beinhalten die Werte eines Wetterelements zu einem bestimmten Zeitpunkt mit der dazugehörigen geographischen Koordinate auf der Erde. In Abbildung 2.2 ist ein Ausschnitt der Web-Oberfläche des NOAA-Services zu sehen, in dem mögliche Datenfelder einer Vorhersage dargestellt sind. Die ausgewählten Wetterdaten werden aus einer Datei im GRidded Binary (GRIB)-Format (siehe Abschnitt 5.5) herausgefiltert und in diesem Format auch wieder ausgeliefert. Neben der Höhe über dem Meeresspiegel kann in dem Filter auch eingestellt werden, welche Wetterdaten aus den umfangreichen Vorhersagedaten herausgezogen werden sollen. In der Abbildung 2.2 sind der Luftdruck (Pressure Reduced to Mean Sea Level (PRMSL)), die Temperatur (Temperature (TMP)) und die absolute Wolkendichte (Total Cloud Cover (TCDC)) ausgewählt. In dieser Arbeit werden exemplarisch die eindimensionalen Daten für Temperatur, Luftdruck, Niederschlagswahrscheinlichkeit und Wolkendichte sowie die zweidimensionalen Daten für die Windrichtung genutzt, aus denen die Windstärke ermittelt werden kann.

[9] http://nomads.ncep.noaa.gov/cgi-bin/filter_gfs_hd.pl?dir=/gfs.2013121412/master

2.3 Visualisierungsarten

Spezielle Formen der Datenvisualisierung bringen besonders bei globalen Wetterdaten einen Mehrwert, da zwischen den einzelnen Wetterelementen globale Zusammenhänge existieren. In diesem Abschnitt werden verschiedene Visualisierungstechniken vorgestellt, die sich zur Anzeige dieser Daten eignen.

Farbspektrum: Für die Visualisierung von eindimensionalen Wetterelementen wie der Temperatur und dem Bewölkungsgrad bietet sich die Einfärbung von Bereichen eines Globus anhand eines Farbspektrums an. Dieses Spektrum bildet eine Gruppe von Farben mit verschiedenen Intervallen auf einen beliebigen Wertebereich ab. Diese Farben dienen der Unterscheidung verschiedener Wertebereiche der Daten und können diskret oder kontinuierlich ineinander übergehen. In Abbildung 2.3 ist eine Visualisierung von Vorhersagedaten der ECMWF zu sehen, die diese Technik verwendet. Sofern die Einprägung des Spektrums gelingt, können absolute Werte gut eingeschätzt werden.

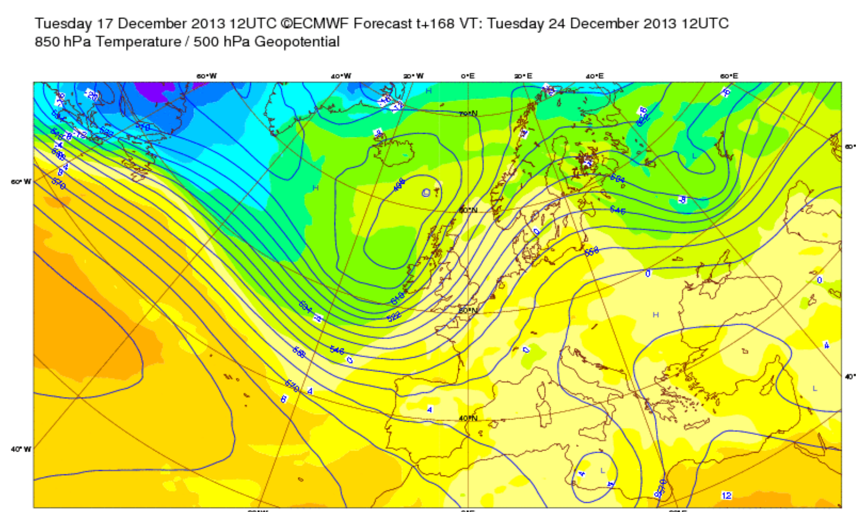


Abbildung 2.3: Wettervorhersage der ECMWF für Europa mit Visualisierung der Temperatur als Farbspektrum und des Luftdrucks mit Hilfe von Isolinien[10]

Isolinien: Eine weitere Möglichkeit, verschiedene Wertebereiche von eindimensionalen Daten darzustellen, ist die Einteilung durch Isolinien. Dabei umschließt eine Linie einen bestimmten Wertebereich der Daten. Eine Schachtelung von mehreren Linien ist sinnvoll, um feinere Abstufungen der Werte abzubilden. Damit werden normalerweise Luftdruckgebiete dargestellt (siehe Abbildung 2.3), weil bei ihnen der absolute Wert nicht so relevant ist wie der Abstand zwischen verschiedenen Bereichen. Die Linien lassen sich außerdem gut mit anderen Visualisierungsformen kombinieren.

Partikel: Zweidimensionale Wetterelemente wie die Windrichtung, stellen ein Vektorfeld dar. Eine Möglichkeit zur Darstellung der Bewegungsrichtung an einem bestimmten Punkt ist die Verwendung von Partikeln. Diese folgen von einer Ausgangsposition aus der Windrichtung und können zusätzlich ihre Bewegungsgeschwindigkeit dynamisch an die Windstärke anpassen. Auf diese Art sind relative Unterschiede der Windstärke zu erkennen, wohingegen absolute Werte kaum auszumachen sind. Die Windrichtung ist jedoch sofort ersichtlich.

[10] `http://www.ecmwf.int/products/forecasts/d/charts/medium/deterministic/msl_uv850_z500`

Spezielle Symbole: Einige Wetterelemente sollen für spezielle Nutzer von Wetterdaten in einer besonderen Form angezeigt werden. Dafür soll ein Symbol an einer Stelle des Globus platziert werden, das den aktuellen Wert oder Wertebereich eines oder mehrerer kombinierter Wetterelemente widerspiegelt. Als Beispiel kombinieren Windfahnen (siehe Abschnitt 6.3.5), wie sie von Seglern und Fliegern genutzt werden, die Werte von Windrichtung und -stärke. Eine einfachere Variante davon sind Pfeile, die durch verschiedene Größen und Richtungen die selben Informationen übermitteln. In einer Vorhersage von Windfinder[11] in Abbildung 2.4 wird diese Version verwendet.

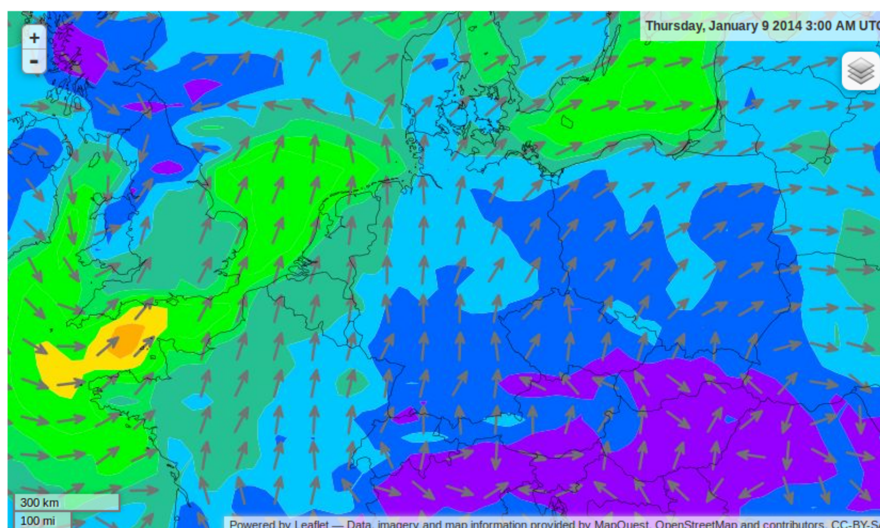


Abbildung 2.4: Wettervorhersage von `Windfinder.com` für Europa mit der Windrichtung als Vektorfeld und der Windstärke als Farbspektrum[12]

Textuelle Darstellung: Die einfachste Art, einen Wert an einem bestimmten Ort darzustellen, ist die textuelle Darstellung. Wie mit Abbildung 2.1 gezeigt wurde, greifen die meisten Apps bei der Präsentation von lokalen Wetterdaten auf diese Darstellungsform zurück. Aber auch bei globalen Wetterdaten ist z. B. eine textuelle Beschriftung einer Isolinie nützlich.

Die meisten Anbieter von Wetterinformationen geben dem Nutzer die Möglichkeit, die Entwicklung des Wetters als Animation über einen bestimmten Vorhersagezeitraum zu verfolgen. Dabei wird die Visualisierung mit einem festen Zeitintervall aktualisiert, um die Veränderungen zum nächsten Zeitpunkt darzustellen. Abstrakte Darstellungsformen wie die Einfärbung anhand eines Farbspektrums eignen sich dabei besser zur Veranschaulichung des zeitlichen Verlaufs als die textuelle Darstellung, bei der die einzelnen Werte kaum ablesbar sind, wenn sie in schneller Abfolge verändert werden.

In dieser Arbeit wird zwischen interaktiven und nicht manipulierbaren Visualisierungsangeboten unterschieden. Die bisher angeführten Beispiele der ECMWF und `Windfinder.com` fallen in die zweite Kategorie, da es keine Möglichkeit gibt, die generierten Bilder anzupassen. Bei einigen Angeboten ist eine freie Navigation durch die Zeitschritte der Vorhersage-Animation möglich, aber es gibt auch Wetter-Filme, die dies nicht erlauben. Im Gegensatz dazu werden Visualisierungen, die eine Manipulierbarkeit erlauben, in geringer Zeit aus den zu Grunde lie-

[11] `http://www.windfinder.com`

[12] `http://www.windfinder.com/weather-maps/forecast#5/51.399/9.668`

genden Wetterdaten generiert. Wenn diese Eigenschaft bei einem Angebot erfüllt ist, können z. B. die Farben eines Farbspektrums veränderbar oder die Anzahl der Verschachtelungen von Isolinien einstellbar sein. Eine derart freie Konfigurierbarkeit kann einerseits Nutzern helfen, bei der Visualisierung bestimmte Schwerpunkte zu setzen, und andererseits die Nutzung für Personen mit körperlichen Besonderheiten, wie z. B. einer Rot-Grün-Schwäche oder Kurzsichtigkeit, vereinfachen.

Aufgrund dieser sinnvollen Möglichkeiten beschäftigt sich diese Arbeit damit, wie moderne Technologien der hardwarebeschleunigten Visualisierung von Wetterdaten für eine animierte und interaktive Darstellung eingesetzt werden können.

Kapitel 3

Stand der Technik

In diesem Kapitel werden aktuell verfügbare Apps, die hardwarebeschleunigte Darstellung von globalen Wetterdaten auf einem dreidimensionalem Globus bieten, vorgestellt. Es werden dazu Stellvertreter für die Web- und iOS-Plattformen herangezogen.

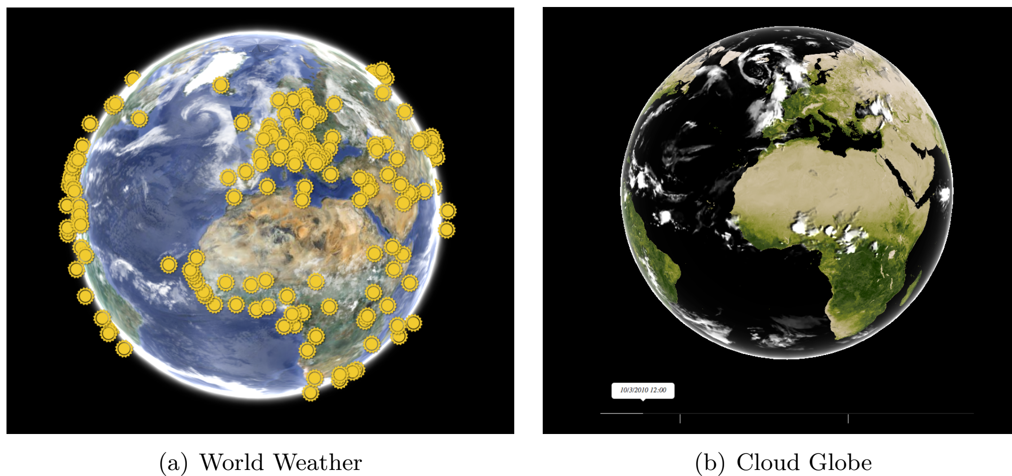
3.1 Web Apps

Als übergeordneter Marktplatz für Web-Applikationen kann nur der seit Dezember 2010 bestehende Chrome WebStore[13] bezeichnet werden. Er bietet die Möglichkeit, ein Angebot nach verschiedenen Produktnamen und Kategorien zu durchsuchen. Er ist fest in den Chrome Browser integriert und forciert auch die Möglichkeit der Offline-Installation von Web-Apps auf den Browser. Weitere Möglichkeiten, um an spezielle Web-Apps zu gelangen, bieten diverse Suchmaschinen, wobei der Suchende eine gute Vorstellung davon haben sollte, was für ein Programm er finden möchte. Zum aktuellen Zeitpunkt lassen sich im Chrome WebStore keine Apps finden, die über die zweidimensionale Darstellung von animierten, aber unmanipulierbaren Wetterinformationen hinausgehen. Deshalb werden im Folgenden zwei WebGL-Applikationen, die sich im Bereich der hardwarebeschleunigten 3D Visualisierung von Wetterdaten befinden, exemplarisch vorgestellt und es wird somit ein Einblick in die aktuell verfügbaren Angebote gegeben.

World Weather[14]

Diese App wurde von thespite entwickelt und stellt als Teilmenge der aktuellen globalen Wetterlage den Bewölkungsgrad auf einem dreidimensionalen Globus dar. Wie in Abbildung 3.1 (a) dargestellt, wird die Wolkendichte durch ein Farbspektrum visualisiert und es erscheinen Symbole wie eine Sonne oder eine Wolke an verschiedenen Orten, um die dortige Wetterlage anzuzeigen. Diese App wurde unter Verwendung von Frameworks wie dem Google WebGL Globe[15], das eine hardwarebeschleunigte Visualisierung globaler Daten auf einem Globus ermöglicht, und Three.js[16], einer allgemeinen Grafik-Bibliothek für WebGL-Apps entwickelt. Die App bietet neben der räumlichen Navigation keine interaktiven Elemente.

[13] `http://blog.chromium.org/2010/05/chrome-web-store.html`
[14] `http://www.clicktorelease.com/code/weather`
[15] `http://www.chromeexperiments.com/globe`
[16] `http://threejs.org/`



(a) World Weather (b) Cloud Globe

Abbildung 3.1: Interaktive WebGL-Applikationen für globales Wetter

Cloud Globe[17]

Die App Cloud Globe wurde im September 2012 vom Google Data Arts Team entwickelt und bietet eine Animation von globalen Bewölkungsdaten auf einem 3D-Globus über einen historischen Zeitraum vom Juli 2010 bis September 2012. Wie bei World Weather wurde der Google WebGL Globe als Grundlage der globalen Visualisierung genutzt. Die zeitliche Animation über die vorberechneten Wetterbilder erfolgt, wie in Abbildung 3.1 (b) dargestellt, in einer hohen visuellen Qualität. Die Interaktion des Nutzers ist neben der räumlichen Navigation durch die Auswahl des Zeitpunkts über einen Zeit-Slider möglich. Bemerkenswert ist der große Umfang des Zeitraums und die hohe Auflösung der Datenbilder.

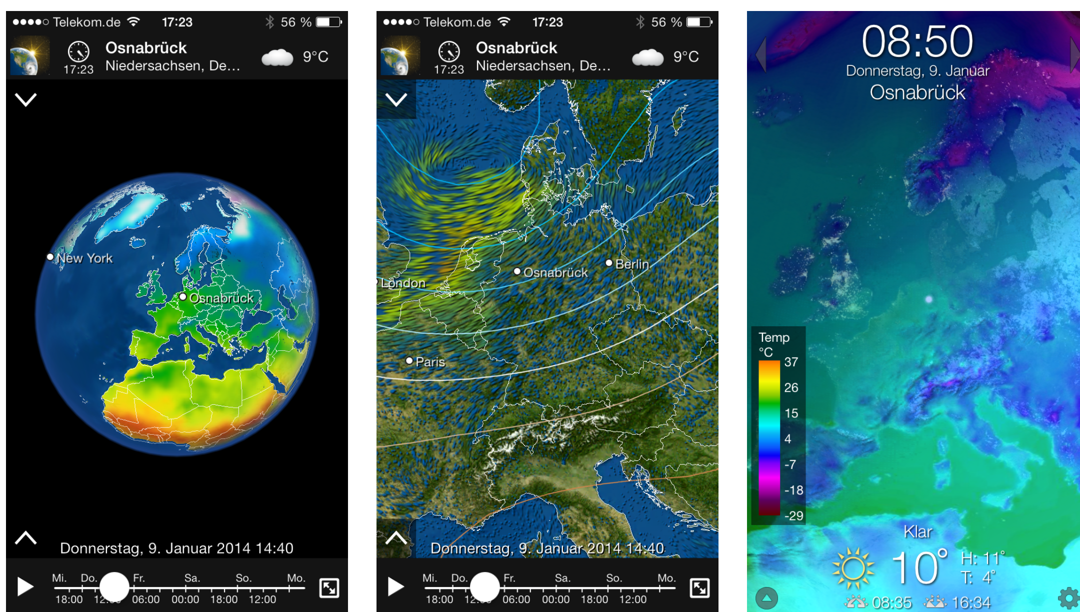
3.2 iOS Apps

Neue Apps für iOS-Geräte wie das iPhone, den iPod Touch und das iPad können Nutzer seit Juli 2008 über den App Store beziehen. Seit seiner Eröffnung wurde bis zum Mai 2013 der 50 millionste Download einer der 850.000 bis dahin veröffentlichten Apps getätigt.[18] Die dort angebotenen Apps sind in 24 Kategorien eingeteilt, unter denen eine ausschließlich „Wetter"-Apps führt. Die beliebtesten Programme in dieser Kategorie bieten überwiegend allgemeine Vorhersage-Informationen zu einem bestimmten Ort in verschiedenen Visualisierungsformen oder zielgruppenorientierte Informationen, wie z. B. Ski-Wetter oder Segel-Wetter. Die Visualisierungen gehen oft nicht über die textuelle Anzeige oder eine unmanipulierbare Radar-Animation einer Wetter-Information hinaus. Im Folgenden werden zwei Vertreter vorgestellt und analysiert, die eine Information über globale Wettervorgänge mit interaktiven Elementen anbieten.

[17] `http://workshop.chromeexperiments.com/cloudglobe/`
[18] `http://www.apple.com/pr/library/2013/05/16Apples-App-Store-Marks-Historic-50-Billionth-Download.html`

MeteoEarth

Diese App wurde im März 2013 vom deutschen Unternehmen MeteoGroup Deutschland GmbH erstmals veröffentlicht und bietet einen guten Funktionsumfang im Sinne dieser Arbeit. Als Visualisierungsarten von Wetterdaten wird die Kombination verschiedener Farbspektren (z. B. Temperatur + Bewölkung), die Anzeige von Isolinien (Luftdruck) und animierter Partikel (Windstärke kombiniert mit Windrichtung) angeboten. Die Daten liegen global vor und werden auf einem 3D-Globus angezeigt. Es gibt einen Vorhersagezeitraum von fünf bis acht Tagen, über den die Wetterinformationen animiert werden. Neben der Kombination verschiedener Visualisierungen bietet die App als interaktive Elemente z. B. eine freie Navigation per Gesten-Steuerung um den Globus und durch den Vorhersagezeitraum über einen Slider. In Abbildung 3.2 (a) ist die Anzeige der Temperatur mit einem Farbspektrum zu sehen. Die Auflösung und Farben des Spektrums sind nicht manipulierbar. Allerdings sind die in (b) gezeigten Partikel auch bei angehaltener zeitlichen Animation animiert, was eine dynamische Bewegung durch ein Vektor-Feld vermuten lässt. Weiterhin ist jede Wetterinformation nur mit einer vorgegebenen Visualisierungstechnik zu betrachten, die nicht an persönliche Anforderungen angepasst werden kann.



(a) MeteoEarth, Darstellung der Temperatur (Farbspektrum)

(b) MeteoEarth, Darstellung des Luftdrucks (Isolinien) und Windes (Partikel)

(c) Living Earth, Darstellung der Temperatur (Farbspektrum)

Abbildung 3.2: Interaktive iOS-Applikationen für globales Wetter

Living Earth

Eine andere, häufiger auftretende Form der globalen Wettervisualisierung wird von der App Living Earth repräsentiert. Sie wurde von Radiantlabs, LLC entwickelt und im September 2010 erstmals im App Store veröffentlicht. Im Gegensatz zu MeteoEarth bietet sie nur die Anzeige von Wetterdaten (wie z. B. Temperatur, Windstärke, Luftfeuchtigkeit) in Form eines Farbspektrums

an. Der Zeitpunkt innerhalb der Vorhersagedaten ist dabei fest eingestellt und eine Animation ist nicht möglich. Auch die Eigenschaften des Farbspektrums sind vorgegeben und nicht manipulierbar. Durch die Tatsache, dass diese App nur vorberechnete Informationsbilder auf einem 3D-Globus anzeigt, lässt sich die subjektiv bessere visuelle Qualität durch eine höhere Auflösung der Daten gegenüber der vielseitigeren und teilweise interaktiven Visualisierung von MeteoEarth erklären. Bemerkenswert ist auch der in Abbildung 3.2 (c) dargestellte große Mindestabstand vom Globus, der trotz hoch aufgelöster Daten eingehalten werden muss.

Kapitel 4

Konzeption

Die Möglichkeiten der hardwarebeschleunigten 3D-Darstellung und der aktuelle Stand der Technik bieten Inspiration für die Funktionalitäten des geplanten Programms. Um die Ziele dieser Arbeit zu erreichen, werden in diesem Kapitel die Anforderungen an eine geeignete App spezifiziert. Der erste Abschnitt über die funktionalen Anforderungen befasst sich mit den Features, die aus Nutzersicht interessant sind. In den Anforderungen der Zielplattformen wird auf die Unterschiede der mobilen Plattformen eingegangen, mit denen sich diese Arbeit befasst. Weiterhin spielt die grafische Oberfläche bei den Apps eine große Rolle, die anschließend konzipiert wird. Den letzten Abschnitt nimmt die Server-App ein, die für die Bereitstellung der Daten benötigt wird. Die Anforderungen wurden mit Blick auf ausreichende Vergleichsmöglichkeiten der verschiedenen Plattformen entwickelt und abschließend ein Fazit mit Blick auf den Stand der Technik gezogen

4.1 Funktionale Anforderungen

Zeitliche Animation der Wetterdaten: Die Daten einer Wetterprognose liegen für jedes Wetterelement an mehreren Zeitpunkten und Orten vor. Mit der App soll eine kontinuierliche Animation der Wetterdaten in chronologischer Reihenfolge möglich sein. Dazu müssen die Daten zwischen zwei Zeitpunkten in regelmäßigem Abstand interpoliert und das Ergebnis für die Darstellung verwendet werden. Die App soll dafür die Daten der folgenden Zeitpunkte dynamisch nachladen.

Interpolation der Wetterdaten: Die Wetterdaten liegen meist in einer geringeren räumlichen Auflösung vor, als es die Bildschirmauflösung erfordert, mit der sie angezeigt werden. Deshalb soll für die Zwischenräume der Daten eine Interpolation oder Approximation durchgeführt werden. Dafür gibt es verschiedene Berechnungsmethoden, die sich in Qualität und Komplexität voneinander unterscheiden.

Verschiedene Visualisierungstechniken: In Abschnitt 2.3 wurden verschiedene Visualisierungsarten von Wetterdaten vorgestellt. In dieser App sollen das Farbspektrum, die Isolinien, Partikel und spezielle Symbole Verwendung finden, da sie bei der dynamischen Animation von Nutzen sind. Da die Darstellungen aus den Ursprungsdaten generiert werden, ist eine hohe Variabilität bei der Kombination und Verwendung der Techniken möglich.

Navigation: Mit dieser App soll es möglich sein, sich einerseits räumlich durch die globalen Wetterdaten, andererseits aber auch zeitlich durch die Prognose zu navigieren. Für die

räumliche Navigation sollte der Nutzer in der Lage sein, die Kameraposition in einem Radius um den Globus zu verändern. Für die zeitliche Navigation soll es möglich sein, einen bestimmten Zeitpunkt auswählen zu können, um so die aktuelle Animation zu manipulieren.

Interaktive Parameter der Visualisierung: Bei der Animation und Visualisierung der Wetterdaten gibt es viele Parameter, die für die Personalisierung der Anzeige veränderbar gehalten werden sollen. Dies können z. B. die Geschwindigkeit der Animation, Genauigkeit der Visualisierung oder die Wahl der Farben eines Spektrums sein.

4.2 Anforderungen der Zielplattformen

Da die App für zwei sehr unterschiedliche und außerdem mobile Plattformen entwickelt wird, ergeben sich daraus Anforderungen, die durch durch Eigenschaften der verwendeten Hardware entstehen. In diesem Abschnitt werden Unterschiede der Zielplattformen in verschiedenen Kategorien herausgearbeitet, die für die App relevant sind.

Auflösung: Ein Browser passt den Inhalt einer Webseite automatisch an die Fenstergröße an, wenn es vergrößert oder verkleinert wird. Darauf soll die App auch reagieren können und bis zu einer minimalen Auflösung eine übersichtliche Darstellung bieten. Auf iOS-Geräten ist diese Eigenschaft ähnlich, da auch hier unterschiedliche Auflösungen und Pixeldichten der Displays existieren. Bei diesen Geräten wird auch eine Anpassung beim Wechsel von Vertikal- zu Horizontalstellung erwartet.

Bedienung: An einem Desktop-System oder Notebook wird ein Browser normalerweise mit Maus und Tastatur bedient. Auf mobilen Geräten der iOS-Familie hingegen werden Apps per Touch-Steuerung mit einem oder mehreren Fingern bedient. Diese Eigenschaft bringt neue Möglichkeiten wie die Gestensteuerung mit sich. Dadurch ist allerdings die Texteingabe erschwert und die Anzahl der interaktiven Elemente durch ihre zwanghafte Visualisierung auf einem kleineren Display eingeschränkt.

Hardware: Da Browser eine hohe Verbreitung anstreben, setzen sie meist nur geringe Hardwareanforderungen voraus. Die Web-App dieser Arbeit setzt auf Hardwarebeschleunigung mit WebGL, da vor allem die Anzahl von angezeigten Bildern pro Sekunde (Frames per second (FPS)) von der Plattform abhängig ist. Deswegen soll herausgefunden werden, mit welchen Einstellungen sich der beste Kompromiss zwischen Qualität und Leistung erreichen lässt. Bei iOS-Geräten besteht wegen verschiedener Grafikprozessoren eine ähnliche Herausforderung, die Variation ist aber deutlich geringer.

Datenverbindung: Besonders auf mobilen Geräten kann die Datenverbindung zu Störungen oder Abbrüchen der Übertragung führen. Dies soll erkannt und z. B. mit einem Hinweis für den Nutzer behandelt werden. Auch die Verbindungsgeschwindigkeit kann sich auf das Verhalten der App auswirken, da die benötigten Wetterdaten erst zur Laufzeit dynamisch nachgeladen werden sollen. Dies soll durch die Verfügbarkeit der Wetterdaten in verschiedenen Auflösungsstufen unterstützt werden, um somit die Menge der zu übertragenden Daten anpassen zu können.

4.3 Grafische Oberfläche

Die Inspiration für geeignete grafische Elemente der App hat sich aus der Analyse der verschiedenen Visualisierungstechniken und aktuell verfügbaren Apps auf den verschiedenen Plattformen ergeben. Deshalb werden in dieser Arbeit folgende Schwerpunkte bei der Entwicklung einer grafischen Oberfläche gesetzt.

Fokus auf die Datenvisualisierung: Im Zentrum der App steht die Darstellung der Wetterdaten mit verschiedenen Visualisierungstechniken. Dies soll auf einem dreidimensionalem Globus realisiert werden, um globale Zusammenhänge der Daten erkennbar zu machen. Diese Anzeige wird den Großteil der sichtbaren Fläche ausmachen, damit Details der Daten erkundet werden können.

Interaktive Elemente: Um die verschiedenen Parameter des Programms zur Laufzeit anzupassen, soll es z. B. Schaltflächen und einen Slider geben, um die Navigation durch die zeitliche Animation zu ermöglichen. Die Visualisierungstechnik eines Wetterelements soll mit verschiedenen Auswahlelementen eingestellt werden und das Spektrum möglichst mit direkter visueller Rückmeldung verändert werden können.

Plattformabhängigkeit: Da in dieser Arbeit für sehr unterschiedliche Plattformen entwickelt wird, werden sich die Nutzeroberflächen unterscheiden. Damit diese Unterschiede nicht zu groß ausfallen, sollen einige Elemente plattformunabhängig zu sehen sein und im Einzelfall auf spezielle Funktionen der Plattform zurückgegriffen werden. Bei iOS-Geräten gibt es eine Bibliothek mit grafischen Elementen, die ein einheitliches Aussehen der Apps ermöglichen sollen, wobei für Web-Apps eine Vielzahl dieser Sammlungen zur Verfügung steht und die Herausforderung in der Auswahl liegt.

4.4 Server

Um die genannten Anforderungen an die Apps realisieren zu können, soll ein unterstützender Server existieren. Die Aufgaben dieser Komponente sind in folgende Bereiche eingeteilt:

Beschaffung und Vorverarbeitung der Daten: Auf dem Server soll ein Programm existieren, das die Vorhersagedaten der relevanten Wetterelemente als GRIB-Daten vom NOAA Web-Service abfragen kann. Diese Daten müssen für die Nutzung in der App ausgelesen und vorverarbeitet werden.

Sicherung und Bereitstellung der Daten: Nach dieser Verarbeitung sollen die Wetterdaten geordnet und in einer Datenbank abgelegt werden. Eine Redundanz soll durch eindeutige Identifikation verhindert werden. Die Daten sollen von den Apps über die gleiche Web-Schnittstelle gezielt abgerufen werden können. Zur Initialisierung soll es möglich sein, Informationen über die auf dem Server verfügbaren Daten einzuholen, ohne diese selbst laden zu müssen. Zusätzlich sollen die Farbspektren in der Datenbank liegen, damit sie für die Apps einheitlich und leicht zu erweitern sind.

4.5 Fazit

Mit Blick auf die Analyse von aktuell verfügbaren Apps auf den Zielplattformen dieser Arbeit im Stand der Technik lassen sich folgende Schlußfolgerungen für die Ausrichtung dieser Arbeit formulieren:

- Unter den Apps auf iOS-Geräten gibt es einige wenige Vertreter, die den Anforderungen an eine interaktive Visualisierung animierter und globaler Wettervorhersagedaten nahe kommen. MeteoEarth ist hervorzuheben, da dieses Programm bereits eine breite Auswahl an verschiedenen Visualisierungsarten anbietet und teilweise auch Interaktion mit der Darstellung ermöglicht. Allerdings ist die Manipulierbarkeit der Visualisierungformen durch den Nutzer eingeschränkt. Andere Vertreter, wie Living Earth erfreuen sich trotz deutlich geringerem Funktionsumfang einer hohen Beliebtheit[19], was sich vermutlich auf die hohe visuelle Qualität zurückführen lässt.
- Bei den verfügbaren Web-Apps gibt es kein Angebot, das den beschriebenen Zielanforderungen annähernd gerecht wird. Die beiden vorgestellten Apps zeigen allerdings, was unter Einsatz von aktuellen Technologien möglich ist.

Mit diesen Erkenntnissen soll mit dieser Arbeit vor allem auf dem Gebiet der Web-Apps Innovation vorangetrieben und auf iOS-Geräten eine Lücke bei den interaktiven Elementen ausgefüllt werden.

[19]Living Earth: 3642 Nutzer-Wertungen im App Store und MeteoEarth: 362 Nutzer-Wertungen in ähnlicher Qualität

Kapitel 5

Technische Grundlagen

Im folgenden Kapitel werden die Werkzeuge vorgestellt, mit denen die Apps dieser Arbeit entwickelt wurden. Für die Realisierung der 3D Visualisierung wird die Grafik-Bibliothek OpenGL mit ihren Ablegern OpenGL ES und WebGL vorgestellt. Als Quelle dienten die OpenGL-Superbible [9] und der OpenGL ES-Programming Guide [10]. Für die Umsetzung auf den Zielplattformen iOS-Gerät und Web-Browser wurden die Programmiersprachen ObjC und JavaScript (JS) genutzt. Es wird ein Überblick der relevanten Features dieser Sprachen gegeben. Abschließend wird das Format der Wetterdaten vorgestellt, die in dieser Arbeit verwendet werden.

5.1 OpenGL

Mit OpenGL hat die Silicon Graphics, Inc. (SGI) im Jahr 1992 die erste Spezifikation der schlanken, plattformunabhängigen Application Programming Interface (API) für die hardwarebeschleunigten Darstellung von 3D Grafiken herausgegeben. Die Spezifikation enthält Details einer Zustandsmaschine für den Rendervorgang und Funktionen für ihre imperative Programmierung. Sie wurde seitdem von Herstellern wie AMD, NVIDIA, Intel, oder Apple in ihre Hardware und Treiber implementiert und somit ins Betriebssystem integriert. Nicht unterstützte Funkionen können auch in Software implementiert werden, die meist mit schlechterer Performanz ausgeführt werden. Den Herstellern ist es außerdem erlaubt, den Standard um Erweiterungen zu ergänzen. Für den Programmierer stehen die Funktionen von OpenGL in vielen Programmiersprachen durch entsprechende Bibliotheken zur Verfügung. Nach einer Übergabe im Jahre 2006 an die Khronos Group wurde 2008 die Spezifikation der Version 3.0 herausgegeben. Mit ihrer Einführung wurden viele bestehende Funktionen für die zukünftige Entfernung markiert. Sie ermöglichten die Abwärtskompatibilität zu alten Versionen, entsprachen aber nicht mehr der flexibleren Ausrichtung von OpenGL. Darunter waren zum Beispiel fest implementierte Beleuchtungsmodelle und Transformationen, die nur durch einige Parameter beeinflusst werden können. Seit der Einführung von Shadern - Programme, die parallel und optimiert auf der Grafikkarte ausgeführt werden - in OpenGL 2.0 sollen diese Funktionalitäten vom Programmierer selbst in der Programmiersprache OpenGL Shading Language (GLSL) formuliert werden. In den letzten Jahren wurde OpenGL kontinuierlich weiterentwickelt und steht seit Juli 2013 in der Version 4.4 zur Verfügung.

Anhand von Abbildung 5.1 soll die Arbeitsweise von OpenGL genauer erklärt werden. Sie

zeigt ein stark vereinfachtes Flußdiagram, das einzelne Komponenten der Zustandsmaschine von OpenGL ES 2.0 und die Reihenfolge ihrer Verarbeitung bis zum Setzen der Pixel in einer Zeichenfläche darstellt. Dieser Ablauf wird Rendering Pipeline genannt.

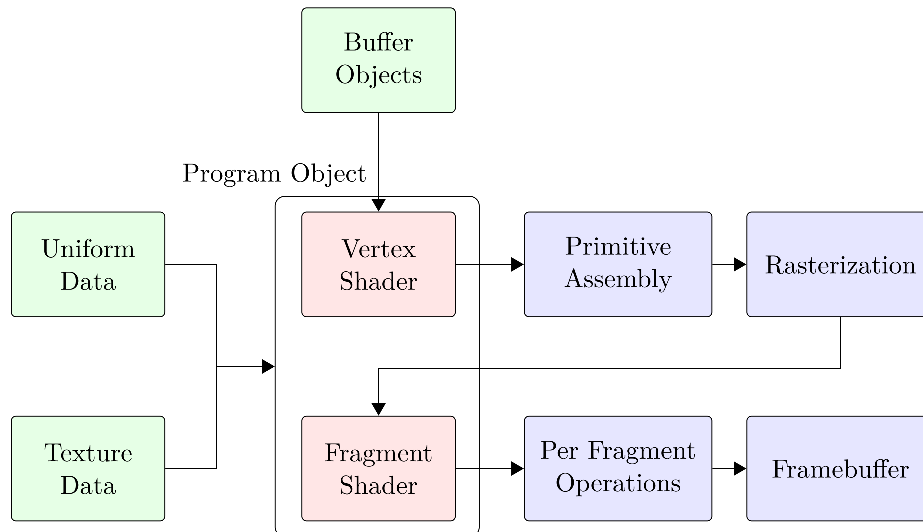


Abbildung 5.1: OpenGL ES 2.0 Rendering Pipeline. Die mit einem Shader programmierbaren Abschnitte der Pipeline sind rötlich hervorgehoben, die fest eingebauten Komponenten blau und die Datenbuffer grün.

Buffer Objects (Daten)

Ein Buffer Object ist ein Feld von Daten, die häuptsächlich zur Beschreibung von Geometrie durch Vertices genutzt werden. Ein Vertex ist ein Punkt im Raum, der neben der notwendigen Position beliebige weitere Informationen, wie einen Normalenvektor, eine Texturkoordinate oder eine Farbe als Vertexattribute enthalten kann. Ein Buffer mit mehreren Vertices wird Vertex Buffer genannt. In Kombination mit einer Topologie und einem Index Buffer, der die Reihenfolge der Vertices angibt, können beliebige Geometrien aus Punkten, Linien oder Dreiecken zusammengesetzt werden. OpenGL stellt spezielle Datentypen wie z. B. GLint, ein vorzeichenbehafteter 32-Bit integer, und GLclampf, eine vorzeichenbehaftete 32-Bit Fließkommazahl im Intervall [0, 1], zur Verfügung. Diese Daten können in der Form von Buffer Objects und Texturen an die Grafikkarte weitergegeben werden.

Eine Textur ist ein mehrdimensionales Feld eines Datentyps und wird wegen spezieller Verarbeitung gesondert betrachtet. Diese Datenstruktur ist für mehrdimensionale Zugriffsmuster optimiert und bietet Interpolationsmethoden wie Nearest Neighbor, oder Linear Interpolation für Werte zwischen den Datenpunkten an. Außerdem sind für jeden Wert bis zu vier Dimensionen, auch Farbkanäle genannt, vorgesehen. Wegen dieser Eigenschaften wird eine Textur oft als Speicher für ein- bis dreidimensionale Bilddaten genutzt. Da die globalen Daten der einzelnen Wetterelemente als zweidimensionales Datengitter vorliegen, bietet sich die Überführung in eine Textur an, um sie später hardwarebeschleunigt darstellen zu können.

Vertex Shader

Der Vertex Shader (VS) ist ein Programm, das für jeden Vertex ausgeführt wird, der an die Grafikkarte übermittelt wurde. Damit es genutzt werden kann, muss es mit einem Fragment Shader (FS) zu einem Program Object kombiniert werden. Der VS wird hauptsächlich für Vektoralgebra genutzt, um die Position eines Vertex durch Projektion und Transformation in das Normalized Device Coordinates (NDC) - Koordinatensystem von OpenGL, in dem die x-, y- und z-Koordinate im Intervall [-1, 1] liegt, zu überführen. Die für diese Berechnungen benötigten Matrizen werden durch Uniforms als Parameter des Program Objects gesetzt. Vom VS aus können auch die Attribute eines Vertex an die nächste programmierbare Komponente, den FS weitergegeben werden. Der VS wird in der C-ähnlichen Programmiersprache GLSL geschrieben und auf der Grafikkarte kompiliert. In Listing 1 ist der Code eines VS zu sehen, der die Position und Texturkoordinate eines Vertex erhält. Im Programm wird die Texturkoordinate unverändert an den FS weitergegeben und die Position mit einer Projektionsmatrix verrechnet, die als Uniform-Variable im gesamten Program Object verfügbar ist.

```
1  #version 100
2  attribute vec4 a_Position;    // Position des Vertex
3  attribute vec2 a_TexCoords;   // Texturkoordinate des Vertex
4  uniform mat4 u_ModelViewProj; // Zusammengesetzte Matrix als Uniform
5  varying vec2 v_TexCoords;     // Verbindung zum Fragment Shader
6  void main() { // Einstiegspunkt für das Shaderprogramm
7      // Texturkoordinaten an Fragment Shader weiterreichen
8      v_TexCoords = a_TexCoords;
9      // Berechnen der projezierten Vertexposition
10     gl_Position = u_ModelViewProj * worldPos;
11 }
```

Listing 1: Beispiel eines Vertex Shaders, der die Koordinaten der Vertices mit Hilfe einer Matrix transformiert (GLESSL)

Primitive Assembly

Im Abschnitt des Primitive Assembly werden die Vertices des Vertex Buffers mit ihrer zugehörigen Topologie und der Reihenfolge aus dem Index Buffer als geometrischen Grundformen interpretiert, die Punkte, Linien, oder Dreiecke sein können. Nach der Erstellung dieser Primitives wird eine Clipping-Methode verwendet und eventuell neue Vertices erstellt, wenn ein Primitive aus den Grenzen des NDC herausragt. Als nächster Schritt wird die Perspective Division durchgeführt, bei der die Koordinaten der Vertices homogenisiert werden. Zum Schluß werden die Primitives auf den zweidimensionalen Ausschnitt des Viewports projeziert. In der folgenden Abbildung 5.2 sind einige Topologien dargestellt, die in OpenGL verwendet werden können.

Rasterization

Bei der Rasterization wird entschieden, für welchen Pixel des Viewports sogenannte Fragments erstellt werden und welche Informationen sie tragen. Für jedes Primitive wird für jeden Pixel, den es überdeckt ein Fragment generiert. Dabei werden die Vertexattribute wie die Texturkoordinate oder der Normalenvektor für die Position des Fragments linear zwischen den Vertices interpoliert. Um Rechenzeit zu sparen, können hier bei aktiviertem Backface Culling Primitives vernachlässigt

werden, deren Vorderseite vom Betrachter wegzeigt. Zusätzlich erhält jedes Fragment einen Tiefenwert, der bei der Perspective Division gesichert wurde.

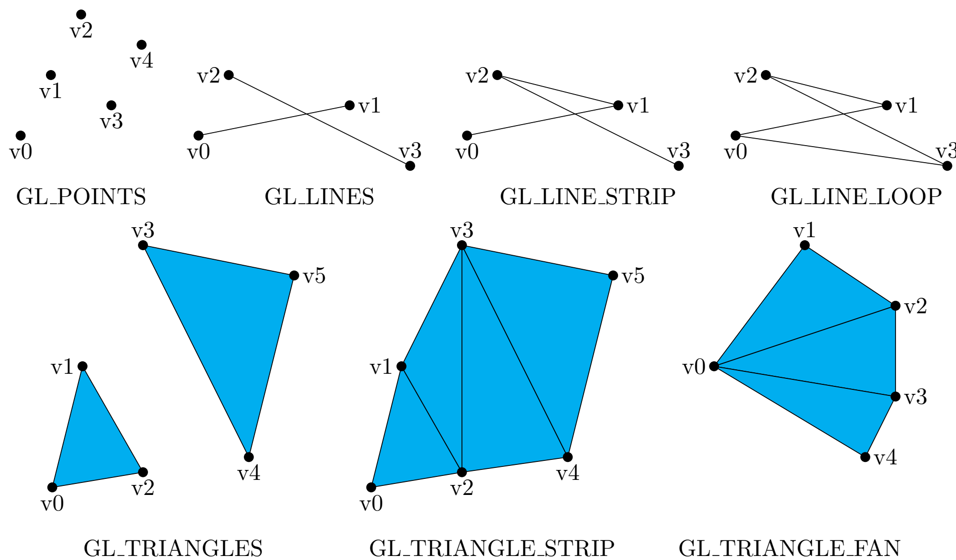


Abbildung 5.2: Primitives in OpenGL, zum Zeichnen ohne Index Buffer müssen die Vertices in aufsteigender Reihenfolge angegeben werden

Fragment Shader

Der Fragment Shader (FS) ist ähnlich dem VS ein Programm, das nicht für jeden Vertex, sondern für jedes erzeugte Fragment aufgerufen wird. Das Programm wird ebenfalls in GLSL geschrieben und hauptsächlich zur Berechnung der Farbe seines Fragments genutzt. Dazu können Texturen, verschiedene Beleuchtungsmodelle, oder auch prozedural erzeugte Farbgebung genutzt werden. Der Code eines FS in Listing 2 zeigt, wie mittels einer Textur und der vom VS übergebenen Texturkoordinate die Farbe eines Fragments gesetzt wird.

```
1  #version 100
2  // Mittlere Präzision für Datentyp float
3  precision mediump float;
4  varying vec2 v_TexCoords;     // 2D-Textur als Uniform
5  uniform sampler2D u_diffTexture;
6  void main() { // Einstiegspunkt für das Shaderprogramm
7      // Fragment-Farbe wird durch Texturzugriff bestimmt
8      gl_FragColor = texture2D(u_diffTexture, v_TexCoords);
9  }
```

Listing 2: Beispiel eines Fragment Shaders, der die Farbe für das Fragment einer Textur entnimmt (GLESSL)

Per Fragment Operations

Nach dem FS werden die Fragments weiteren Verarbeitungen unterzogen, die ihre letztendlichen Eigenschaften im Frame Buffer bestimmen. Mithilfe eines Depth Buffer Tests können Fragments

übersprungen werden, die an der gleichen XY-Position, aber hinter einem anderen Fragment liegen, um eine Überdeckung zu realisieren. Als Alternative kann dabei Blending genutzt werden, um die Farben zweier konkurrierender Fragments zu vermischen und so zum Beispiel Transparenzen zu erzeugen.

Framebuffer

Im Speicher der Grafikkarte ist der Framebuffer ein Bereich, der für die Weitergabe an ein Display reserviert ist. Der Framebuffer ist als eine zweidimensionale Zeichenfläche vorstellbar, die mit einer bestimmten Farbpalette und Auflösung den gesamten Inhalt der Ausgabe einer Grafikkarte enthält. Seit OpenGL 3.0 können auch nicht sichtbare Framebuffer-Objekte erstellt werden, deren Inhalt mit einer Textur verknüpft ist. Bei einer Animation, z. B. dem Scrollen von Text ist es oft performanter, den Framebuffer komplett zu leeren und den Inhalt neu zu zeichnen, als die Veränderung zwischen zwei Bildern zu analysieren und nur bestimmte Bereiche zu aktualisieren. Um dabei Artefakte wie Flackern und Schlierenbildung bei einer hohen Bildwiederholrate zu vermeiden wird Double Buffering[20] angewandt. Diese Technik verwendet zwei Buffer (Front- und Back-), bei denen einer angezeigt wird, während der andere aktualisiert wird. Diese beiden Buffer werden ausgetauscht, wenn der Monitor bereit ist, neue Inhalte der Grafikkarte anzuzeigen.

5.2 OpenGL ES und WebGL

Im Jahr 2003 veröffentlichte die Khronos Group die erste Version der OpenGL ES-Spezifikation. Sie wurde als Untermenge von OpenGL mit der speziellen Ausrichtung auf das 3D-Rendering auf eingebetteten Systemen und mobilen Geräten, wie Smartphones, Spielkonsolen und Fahrzeugen, entwickelt. Im Gegensatz zu Desktop-Hardware haben diese Geräte meist Defizite bei der Prozessorgeschwindigkeit, Speicherbandbreite und Hardware für Fließkomma-Operationen. Weiter wird bei diesen Geräten auf einen geringen Energieverbrauch zur Verlängerung der Akku-Laufzeit geachtet.

OpenGL ES 1.x

Die Version 1.0 von OpenGL ES wurde mit Bezug auf OpenGL 1.3 entwickelt. Die wichtigsten Punkte bei der Bestimmung der übernommenen Funktionen waren:

- die Entfernung von Redundanzen, um nur den performantesten Weg zu einem Ergebnis zu ermöglichen.
- die Bewahrung der Abwärtskompatibilität zu OpenGL.
- die Entfernung des double-Datentyps und Ersetzung der Fließkomma-Vertexattribute durch Festkomma-Werte.
- die Sicherung eines minimalen Standards bezüglich der Bildqualität, da diese bei kleinen Bildschirmen besonders wichtig ist.

Die Version 1.1 wurde im Jahr 2005 veröffentlicht, basiert auf OpenGL 1.5 und fügte einige neue Funktionen wie eine erweiterte Textur-Verarbeitung, Vertex Buffer Objects und mehr Abfragemöglichkeiten über den aktuellen Status der Zustandsmaschine hinzu.

[20] http://en.wikipedia.org/wiki/Double_buffering

OpenGL ES 2.0

Basierend auf OpenGL 2.0 verabschiedete sich die 2007 veröffentlichte Version 2.0 der OpenGL ES Spezifikation von der festen Rendering-Pipeline. Damit wurde auf die Abwärtskompatibilität zu den 1.x Versionen verzichtet, um den Speicherverbrauch des Treibers gering zu halten. Ein weiteres Argument war die Tatsache, dass die feste und die programmierbare Pipeline selten zusammen verwendet wurden, da die feste auch durch Shaderprogramme implementiert werden kann. Zur Programmierung der Shader wurde die Programmiersprache OpenGL ES Shading Language (GLESSL) eingeführt, die GLSL ähnelt. Als Veränderung bietet sie z. B. Parameter für verschiedene Genauigkeiten bei Shader-Berechnungen an, um eine verbesserte Performanz zu erreichen. Es wurde außerdem die aus OpenGL 3.0 bekannte Möglichkeit implementiert, die Pixel eines nicht sichtbaren Framebuffer-Objekts in den Programmspeicher zu übertragen.

OpenGL ES 3.0

Die aktuellste Version 3.0 wurde 2012 von der Khronos Group herausgegeben. In Anlehnung an OpenGL 3.1 wurden einige neue Features gegenüber der Vorgängerversion integriert:

- eingebaute Texturkompression mit hoher Bildqualität
- die Shader-Sprache GLESSL unterstützt 32 Bit Fließkomma-Operationen
- die Texturfunktionalitäten wurden erweitert, sodass z. B. float-Texturen, 3D-Texturen und Textur-Ausmaße, die nicht einer Zweierpotenz entsprechen, unterstützt werden
- per Transform Feedback kann die Ausgabe eines VS mit einem Buffer Object verknüpft und die weiteren Abschnitte der Rendering-Pipeline übersprungen werden

WebGL

Ein weiterer Ableger von OpenGL ist die Spezifikation von WebGL, die zum ersten Mal im Jahr 2011 von der Khronos Group herausgegeben wurde. Die Spezifikationen wurden weitgehend von OpenGL ES 2.0 übernommen. Der offensichtlichste Unterschied zu den anderen Ablegern ist, dass WebGL nur eine API für die Sprache JavaScript anbietet und ohne Plugin in kompatiblen Web-Browsern funktioniert. Die am meisten verbreiteten Browser[21] unterstützen WebGL und ermöglichen so dem Programmierer hardwarebeschleunigtes 3D-Rendering im Browser. Weitere Gemeinsamkeiten mit OpenGL ES 2.0 sind:[22]

- eingeschränkte Unterstützung von Texturen mit Dimensionen ungleich einer Zweierpotenz
- der 64-Bit Datentyp GL_DOUBLE wird nicht unterstützt
- das Fehlen des Geometry Shaders, um dynamisch Geometrie zu erzeugen

[21] `http://gs.statcounter.com/#browser-ww-monthly-200807-201307`
[22] `http://www.khronos.org/webgl/wiki_1_15/index.php/WebGL_and_OpenGL_Differences`

5.3 Objective-C und Cocoa Touch

Die Programmiersprache Objective-C (ObjC) wurde 1980 von Brad Cox entwickelt. Sie verfolgt ein objektorientiertes Konzept mit Klassen und Vererbung, das an das Nachrichten-Empfänger System der Sprache Smalltalk-80 angelehnt ist und wurde als Obermenge der Programmiersprache C implementiert. Deshalb sind alle Datentypen und Funktionalitäten von C enthalten. Im Jahr 1988 lizensierte die Firma NeXT Software die Sprache und entwickelte Bibliotheken und eine Entwicklungsumgebung mit dem Namen NEXTSTEP. Nach der Übernahme von Apple Computer im Jahr 1996 wurde die Sprache als Basis für das Betriebssystem Mac OS X verwendet. Mit der auf NEXTSTEP aufbauenden Entwicklungsumgebung Cocoa wird diese Sprache heute hauptsächlich für die Programmierung von nativen Apps für das Betriebssystem Mac OS und den mobilen Ableger iPhone Operating System (iOS) verwendet. Als Quelle für dieses Unterkapitel wurde [6] verwendet.

```
1  @interface Vector2i : NSObject <Vector>
2  // Die properties der Klasse werden definiert
3  @property (nonatomic) int x;
4  @property (nonatomic) int y;
5
6  -(id) initWithX: (int) xVal andY: (int) yVal;
7  -(double) performNumberFunc: (double (^)(Vector2i*)) numberFunc;
8  @end
9
10 @implementation Vector2i
11 @synthesize x, y; // Erstellen der Getter und Setter
12 -(id) initWithX: (int) xVal andY: (int) yVal {
13     self = [super init]; // Initialisierung der Oberklasse
14     if (self) {
15         [self setX: xVal];
16         [self setY: yVal]; // Zuweisung der Properties
17     }
18     return self;
19 }
20 // Instanz-Methode nimmt Funktionsblock als Parameter entgegen
21 -(double) performNumberFunc: (double (^)(Vector2i*)) numberFunc {
22      // Funktion ruft Block mit der Klasseninstanz als Parameter auf
23     return numberFunc(self);
24 }
25 @end
```

Listing 3: Deklaration und Implementation der Vektor-Klasse Vector2i (ObjC 2.0)

Eine Erweiterung von ObjC gegenüber C ist das Protocol. Es ist eine Definition eines Interface, das Klassen verabschieden können. Das ist nützlich, wenn der Entwickler kenntlich machen will, dass Objekte von beliebigen Klassen verarbeitet werden können, wenn sie bestimmte Funktionalitäten anbieten. Beispielsweise sollte in ObjC Programmen ein Objekt dem NSCoding-Protocol entsprechen, wenn es nach einem standardisiertem Verfahren archiviert werden soll. Bei der Umsetzung der iOS-App wurde diese Eigenschaft für die Umsetzung eines Event-Systems mit einheitlicher Schnittstelle der Eventlistener genutzt.

Zwei Schlüsselmerkmale, die aus Smalltalk-80 übernommen wurden, sind das Nachrichtensystem und die strenge, dynamische Typisierung. Bei der Arbeit mit Objekten in ObjC werden keine Methoden aufgerufen, sondern Nachrichten mit Argumenten an einen Empfänger-Objekte

gesendet. Dabei wird die Klasse des Empfängers, wie auch die auszuführende Methode erst zur Laufzeit bestimmt. Um falschen Aufrufen vorzubeugen, wenn eine Sammlung von Instanzen unterschiedlicher Klassen verarbeit wird, sollte vor dem Aufruf geprüft werden, ob eine entsprechende Methode implementiert ist. Diese Eigenschaften entstehen aus der dynamischen Typisierung, bei der erst zur Laufzeit über die Klassenzugehörigkeit eines Objekts entschieden wird. Einer Variablen, die eine Referenz auf ein beliebiges Objekt enthalten soll, kann deshalb der Universal-Pointer-Datentyp id zugewiesen werden. Optionale statische Informationen über die Klasse einer Variablen helfen dem Compiler bei der Fehlersuche. Der Typ einer Variable wird bei ihrer Deklaration festgelegt und kann sich nicht verändern. Diese Eigenschaft wird strenge Typisierung genannt. Eine Vektor-Klasse, die konform zu einem Vector-Protocol ist, ist in Listing 3 beispielhaft deklariert und implementiert.

Objective-C 2.0

Im Jahr 2007 wurde von Apple eine weitreichende Erweiterung von ObjC veröffentlicht, die verschiedene Änderungen der Syntax und neue Funktionalitäten beinhaltet.

Eine grundlegende Veränderung der Sprache ist die Einführung von Blöcken. Sie funktionieren wie anonyme Funktionen, die bei ihrer Definition den Wert von lokale Variablen aus ihrem Kontext kopieren können, um sie beim Aufruf zu nutzen. Um die Referenz zu kopieren und Variablen mit elementarem Datentyp im Kontext zu verändern, müssen sie mit dem __block-Qualifier ausgezeichnet werden. Blöcke werden in der Praxis als Callback-Funktion übergeben, die z. B. nach erfolgreicher Abfrage einer WebServer-Datenbank das Programm mit Inhalten füllt. Dies kann asynchron passieren, sodass der Programmablauf nicht durch eine andauernde Anfrage blockiert wird. Im Listing 3 wird für die Vector-Klasse eine Funktion implementiert, die einen übergebenen Block ausführt, der einen Vektor als Parameter verlangt. Und Listing 4 zeigt, wie damit gearbeitet wird.

```
1  Vector2i *vec1 = [[Vector2i alloc] initWithX: 1 andY: 2]];
2  Vector2i *vec2 = [[Vector2i alloc] initWithX: 1 andY: 5]];
3  double (^lengthSum)(Vector2i*); // Eine Block-Variable
4  // Definition eines Blocks, der die Länge eines übergebenen Vectors
5  // mit der Länge des lokalen vec1-Objekts addiert
6  lengthSum = ^ double (Vector2i* vec) {
7                      return [vec1 length] + [vec length];
8                  };
9  [vec2 performNumberFunc: lengthSum]; // Output: ...
```

Listing 4: Übergabe einer Block-Variable

Zur Vereinfachung der Syntax gibt es *properties*, mit denen die Instanzvariablen einer Klasse deklariert werden können. Mit dem Schlüsselwort @synthesize im Implementierungsteil der Klasse werden automatisch Setter- und Getter-Methoden für die Variable erzeugt. Jeder *property* können verschiedene Eigenschaften (wie z. B. *readonly* um Schreibzugriffe zu verhindern) zugewiesen werden. Eine weitere Vereinfachung ist die Fast-Enumeration, bei der die Iteration über eine Sammlung von Objekten durch optimierte Pointer-Arithmetik ersetzt wird und hat den Entwicklungsprozess der dynamischen App vereinfacht.

Eine neu eingeführte Garbage Collection sorgt dafür, dass Objekte, die nicht mehr referenziert werden, regelmäßig entfernt werden. Dies wird durch einen automatischen Referenzzähler pro

Objekt realisiert. Der Referenzzähler wird hochgezählt, wenn eine neue Referenz auf sein Objekt erstellt wurde und heruntergezählt, wenn auf eine Referenz nicht mehr zugegriffen werden kann. Die oft vergessene manuelle Verwaltung dieser Zähler wurde mit der Einführung von Automatic Reference Counting (ARC)[23] seit iOS 5 (2011) entfernt. Diese Eigenschaft sorgt für eine bessere Zugänglichkeit der Sprache.

Cocoa Touch und das Foundation Framework

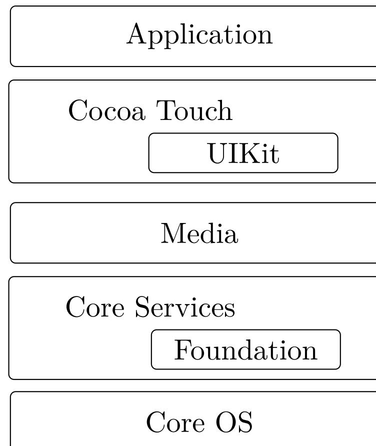


Abbildung 5.3: iOS Framework-Hierarchie

Wie in Abbildung 5.3 dargestellt ist die Architektur des Betriebssystems iOS in hierarchische Ebenen eingeteilt, wobei jede Ebene die Funktionen der darunterliegenden nutzt und damit der darüberliegenden neue ermöglicht.

Core OS: Hier befinden sich hardwarenahe Programme, wie der Kernel des Betriebssystems und das Energie- und Dateimanagement.

Core Services: In dieser Ebene sind grundlegende Techniken, wie String-Manipulation, Kontaktmanagement und die Systemeinstellungen implementiert. Außerdem greift diese Ebene auf die Spezial-Hardware des iPhones, wie den Kompass, GPS und das Gyroskop zu. Wichtig für die Apps sind das Foundation Framework, das vielseitige Klassen, wie NSString und NSArray, enthält und Core Data für die Archivierung von Daten nutzt.

Media: Die Bibliotheken dieser Ebene arbeiten intensiv mit dem Foundation Framework und bieten Funktionen für die graphische Darstellung an. Die Frameworks heißen z. B. Core Graphics, Core Text und Core Audio. Auch die OpenGL ES Implementation befindet sich in dieser Ebene.

Cocoa Touch: Vor allem die Klassen der Bibliothek UIKit machen sich die grafischen Funktionen zu Nutze und stellen eine Struktur für eine iOS-App mit User Interface und Touch-Bedienung dar.

Application: Alle Apps, die für das iPhone geschrieben werden, sind diesem Layer zuzuordnen. Neben dem Zugriff auf die darunterliegenden Funktionen gibt es für Entwickler die Möglichkeit der Nutzung von anderen Apps, z. B. durch das Map Kit, dem Game Kit und iAd, einem Framework, dass für die Integration von Werbung entwickelt wurde.

[23] `https://developer.apple.com/library/mac/releasenotes/ObjectiveC/RN-TransitioningToARC/Introduction/Introduction.html`

Unter dem Begriff Cocoa Frameworks werden das Foundation Framework und das UIKit zusammengefasst. Zusammen mit der Entwicklungsumgebung XCode, dem Interface Builder und dem iPhone-Simulator wird von der Entwicklung einer Cocoa-App gesprochen. Eigenschaften einer Cocoa-App sind z. B. die allgemeine Struktur, standardisierte Elemente für das User-Interface, Lokalisation und die Text-Darstellung.

Das Foundation Framework grenzt sich vom UIKit ab, indem die Objekte dieser Klassen nicht unbedingt im User-Interface sichtbar sind. Die Klassen sind von der Grundklasse NSObject abgeleitet und folgen dem Konzept, dass es, wie z. B. bei NSString und NSMutableString, eine unveränderliche und eine veränderliche Version gibt. Außerdem wird durch wenige abstrakte Klassen eine große Gruppe von konkreten Unterklassen definiert. Die wichtigsten Klassen des Frameworks, die bei der Umsetzung verwendet wurden, kümmern sich um die:

- Abstraktion der elementaren Datentypen
- String-Verarbeitung
- Netzwerk-Kommunikation
- Archivierung

Die wichtigste Ebene für iOS-Apps ist Cocoa Touch, zu der auch das UIKit Framework gehört. Es bietet ähnlich dem Vorbild AppKit, das für Desktop-Apps entwickelt wurde, vorgefertigte Objekte, um ein User-Interface zu erstellen, das dem Look-And-Feel einer iOS-App entspricht. Dazu wird intensiv das Model View Controller (MVC)-Pattern genutzt, um Datenquelle und visuelle Repräsentation voneinander zu trennen. Als Zusatz werden sogenannte Delegates genutzt, um eine Veränderung der Daten an den Controller weiterzugeben. Als weiteres Konzept wird das Template-Pattern genutzt und durch Überschreiben von vordefinierten Methoden allgemein gehaltener Klassen eine eigene Logik implementiert.
Die wichtigsten Aufgaben des UIKits sind somit:

- Das Event-Handling für Multi-Touch-Events, Gesten und iPhone Ereignissen, wie ein ankommender Anruf oder eine Akku-Warnung
- Allgemeine User-Interface-Klassen für Buttons, Switches, Image Views, Text Input, ...
- Darstellung von Text
- Archivierung von Daten mit dem Core Data Framework

5.4 JavaScript und HTML5

JS ist eine Skriptsprache, deren erste Version unter dem Namen LiveScript im Jahr 1995 von Brendan Eich herausgegeben wurde. Sie wurde seitdem von der Netscape Communication Cooperation und Mozilla Cooperation weiterentwickelt. Im Jahr 1997 gab Ecma international die erste Spezifikation unter dem Namen ECMAScript heraus, die seit 2011 in der Version 5.1 vorliegt. Ihre unterschiedlichen Implementierungen wie JavaScript (Mozilla und Google) und JScript (Microsoft) werden unter dem Namen JavaScript zusammengefasst. Da JS ein Teil des Internetbrowsers ist, wird es vor allem für die Manipulation von Webseiten genutzt und ermöglicht Client-seitige Interaktion mit dem Nutzer. Durch neu aufkommende performante virtuelle Maschinen und Plattformen wie Node.js[24] wird JS immer häufiger auch für Server-Apps, Spiele-Entwicklung und Desktop-Applikationen eingesetzt. Als Quelle für dieses Unterkapitel wurde [4] verwendet.

Trotz des Namens gibt es neben der C-ähnlichen Syntax und den eingebauten Klassen *Math* und *Date* nicht viele Ähnlichkeiten zur Programmiersprache Java. JS hat statt einer statischen, eine schwache, dynamische Typisierung, denn Variablen können - im Gegensatz zu Objective-C - zur Laufzeit ihren Typ ändern. Die Eigenschaft first-class-functions ermöglicht, dass eine Funktion selbst als Funktionsargument verwendet oder einer Variable zugewiesen werden kann. Durch die auf Prototypen basierte Programmierung kann mit JS objektorientiert, prozedural und auch funktional programmiert werden.

Viele dieser Eigenschaften resultieren daraus, dass sich fast jeder Datentyp auf den grundlegenden Typen *Object* zurückführen lässt. Eine *Object*-Instanz ist eine Sammlung von Schlüssel-Wert Paaren (im Weiteren *properties* genannt). Der Datentyp ähnelt damit dem Datentyp *HashMap* in Java oder *Dictionary* in der Skriptsprache Python. Das Object ist so tief in JS verwurzelt, dass selbst globale Variablen die *properties* eines globalen Objekts sind. Mit einer for/in-Schleife kann man durch die *properties* eines Objects iterieren. Die primitiven Datentypen in JS sind *Number*, *Boolean* und *String*. Mit Hilfe einer Konstruktorfunktion werden Instanzen dieser Typen z. B. bei Vergleichen und Operationen wie der Konkatenation automatisch in ein Objekt mit zusätzlichen Funktionen konvertiert. Spezielle Werte sind *null* und *undefined*, die die Abwesenheit eines Objekts bzw. einen nicht initialisierten Wert anzeigen. Weitere grundlegenden Klassen in JS sind:

Function: Eine Instanz enthält ausführbaren Code, der mit einem beliebigen Context-Objekt als *this*-Referenz ausgeführt werden kann. Bei Aufruf wird ein unsichtbares Ausführungsobjekt erstellt, das lokale Variablen der Funktion enthält.

Array: Eine geordnete Sammlung von Werten, bei der die Anzahl der Elemente automatisch verwaltet wird. Es existieren Methoden für das Hinzufügen und Entfernen von Objekten und das Sortieren.

Date: Die Repäsentation eines Datums. Kann in verschiedenen Formatierungen ausgegeben werden und als Timestamp.

RegExp: Ermöglicht das Arbeiten mit regulären Ausdrücken.

Error: Zeigt das Vorkommen eines Fehlers an.

Das Anlegen einer neuen Klasse und einer Vererbungshierarchie lässt sich in JS durch die *prototype-property*, die jedes Objekt besitzt, realisieren. Sie enthält die Referenz eines Objekts, auf das zugegriffen wird, wenn eine angeforderte *property* des referenzierenden Objekts nicht

[24] `http://nodejs.org/`

```
// Anlegen einer Konstruktorfunktion
var Vector2 = function(x, y) {
    this.x = x;
    this.y = y;
}
// Deklaration der Instanzvariablen
Vector2.prototype.x;
Vector2.prototype.y;
// Definition einer Instanzmethode
Vector2.prototype.length = function() {
    return Math.sqrt(this.x * this.x + this.y * this.y);
}
```

Listing 5: Eine eigene Klasse für Vektoren in JavaScript implementiert

existiert. Instanzen einer neuen Klasse können mit dem new-Operator und ihrer Konstruktorfunktion erstellt werden. Sie erben von deren Prototyp-Objekt der Klasse und besitzen die dafür definierten *properties* als Instanzvariablen und -methoden. Allerdings fehlt die Möglichkeit der Datenkapselung. Diese kann aber durch Closures realisiert werden, was im Folgenden noch erklärt wird. In Listing 5 ist beispielhaft zu sehen, wie eine Klasse für einen zweidimensionalen Vektor angelegt werden kann.

Im Gegensatz zu Objective-C hat JS keinen Block Scope sondern einen Function Scope. Im Block Scope befinden sich Variablen, die in einem Anweisungsblock zwischen zwei geschweiften Klammern definiert wurden. Sie sind für den Kontext automatisch unsichtbar. Der Function Scope in JS ist so implementiert, dass beim Aufruf einer Funktion automatisch ein Ausführungsobjekt erstellt wird, das für den Kontext unerreichbar ist. Lokale Variablen in der Funktion werden diesem Objekt als *property* hinzugefügt. In Listing 6, Zeile 10 ist die Variable *fscope* für den Kontext unerreichbar, weil sie sich im Function Scope einer anonymen Funktion befindet.

```
var funcs = [];
// Ein Array wird mit Funktionen
// gefüllt, die den Schleifenindex
// zurückgeben
for (var index = 0; index < 10; ++index) {
    funcs[index] = function() {
                    var fscope = 5;
                    return i;
                };
    fscope; // (Function Scope)
            // Reference Error
}
i; // (Block Scope) Wert: 10
funcs[4]();       // Wert: 10
funcs[6]();       // Wert: 10
```

Listing 6: Unerwartetes Verhalten einer Closure

```
function counter() {
    var n = 0;
    // Es wird ein Objekt zurückgegeben,
    // das eine Funktion enthält
    return {
        count: function() {
     // Durch den Function-Scope von counter
     // wird die Variable n geschützt
                return n++;
            }
    };
}
var c = counter();
c.count();    // Wert: 0
c.count();    // Wert: 1
```

Listing 7: Closure zum Kapseln von Variablen

Ein besonderes Verhalten in diesem Zusammenhang sind so genannte Closures. Sie bezeichnen die Eigenschaft, dass Funktionsobjekte bei ihrer Definition Variablen aus dem umliegenden Kontext einbinden können. Dabei kann unerwartetes Verhalten entstehen, wie Listing 6 verdeutlicht. Entgegen der Erwartung, dass der Wert von *index* kopiert wird, geben die Funktionen den Wert zurück, den *index* nach Ausführung der Schleife hat. Dieses Verhalten ist dadurch zu erklären, dass die Funktion bei ihrer Definition eine Referenz zum Kontext-Objekt erstellt und somit auf

lokale Variablen wie *index* zugreifen kann. Da im Beispiel jede Funktion eine Referenz auf das gleiche Objekt hat, sind auch die Werte identisch. Diese Eigenschaft kann genutzt werden, um Variablen vor Zugriff von außerhalb zu schützen, wie im Beispiel Listing 7 mit der Variable n geschehen. Damit lässt sich sogar eine Datenkapselung bei Klassen erreichen, indem lokale Variablen nur durch Getter- und Setter- verändert werden können. Dies wurde bei der Umsetzung genutzt, um Singleton-Objekte mit einer privaten und öffentlichen Schnittstelle zu realisieren.

ECMAScript 5

Mit der Revision 5 der ECMAScript Spezifikation[25] haben einige neue Bestandteile Einzug in die Sprache gefunden, die sich seit Version 3 in den Implementationen der Browser entwickelt haben. Eine wichtige Neuerung ist die Veränderbarkeit von *property attributes*. Vorher waren diese unsichtbar oder nicht vorhanden. Nun können für jede *property* verschiedene Parameter gesetzt werden, die folgende Auswirkungen haben:

value: Bestimmt den Wert einer *property*.
writable: Schreibzugriffe werden ignoriert, wenn dieser Parameter auf *false* steht.
get: Eine Getter-Funktion, die bei Lesezugriff aufgerufen wird und einen Wert zurückgibt.
set: Eine Setter-Funktion, die bei Schreibzugriff auf diese *property* aufgerufen wird.
configurable: Hiermit kann definiert werden, ob diese *property* gelöscht werden kann.
enumerable: Dieser Parameter kontrolliert, ob die *property* bei einer for/in-Schleife über das zugehörige Objekt auftaucht.

Mit diesen Möglichkeiten lassen sich die Eigenschaften eines Objekts vor einem Schreibzugriff von außen zu schützen, was eine Alternative zu Closures darstellt. Bei der Umsetzung werden durch Definition von *property attributes* dynamisch die Methoden einer Klasse generiert.

Als weitere Neuerung ist ein Parser für das JavaScript Object Notation (JSON)-Format von JavaScript Objekten integriert. Dieses Format wurde 2006 von Douglas Crockford entwickelt und stellt im Vergleich zu XML eine platzsparende Alternative für den Austausch von Objekten und Arrays dar. Es wird aktuell auch von einigen anderen Sprachen, wie z. B. PHP unterstützt und lässt sich zum Persistieren von Objekten oder Arrays mit Inhalten eines elementaren Datentyps, wie String, Number und Boolean nutzen.

Frameworks und Compiler

JS profitiert aktuell stark von einer Community, die frei verfügbare Plattformen wie Node.js für serverseitiges JS, Compiler für die Umwandlung anderer Sprachen nach JS und umfangreiche Frameworks entwickelt. Eines der beliebtesten Frameworks für die Erstellung von dynamischen Webseiten ist jQuery[26]. Diese Bibliothek vereinfacht das Ansprechen von Elementen einer Webseite und bietet erweiterte Funktionen für eingebaute Klassen und vor allem die Kompatibilität mit verschiedenen Browsern und deren Versionen. Es ist üblich, dass sich bestimmte Schlüsselwörter bei den verschiedenen Browser-Herstellern stark unterscheiden können. Ein Ableger dieses Frameworks ist jQuery UI, das die einheitliche Gestaltung von Benutzeroberflächen auf Desktop- und mobilen Geräten erleichtert. Neben Frameworks ermöglichen spezielle Compiler, wie asm.js, dass z. B. in C und C++ geschriebene Apps in optimierten JS-Code umgewan-

[25] `http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf`
[26] `http://jquery.com/`

delt werden. Für die vereinfachte Erstellung von dreidimensionalen Inhalten mit WebGL kann Three.js[27] verwendet werden.

In dieser Arbeit wird das Framework Google Closure Library - ein Teil der Closure Tools[28] - eingesetzt. Daraus werden vor allem mathematische Funktionen für die Arbeit mit OpenGL genutzt. Zusätzlich steht der Closure Compiler zur Verfügung, der JS-Code durch eine statische Analyse und Minimierung optimiert. Auf Three.js wird in dieser Arbeit verzichtet, um volle Kontrolle über die verwendeten OpenGL-Objekte und Funktionen zu behalten und die Vergleichbarkeit zur iOS-Version zu bewahren.

HTML5

Mit der Entwicklung von HTML5 seit dem ersten Arbeitsentwurf der Web Hypertext Application Technology Working Group haben viele neue Elemente in den Standard Einzug gehalten. Wegen der hohen Verbreitung des Standards wurden viele Faktoren, wie Kompatiblität, Sicherheit und Vereinfachung berücksichtigt. Die wichtigste Neuerung für diese Arbeit ist die Einführung des *canvas*-Elements. Es entspricht einer Zeichenfläche, auf der unmittelbares Zeichnen von 2D- und durch WebGL auch 3D-Inhalten möglich ist. Dabei kann von einer Hardwareunterstützung durch die Grafikkarte des Client-Systems profitiert werden. Weitere interessante Erweiterungen sind die Nutzung von Offline-Daten im Browser, eine API für Lokalisation für verschiedene Client-Systeme und Web Workers, die das Programmieren von parallelen Prozessoren mit JS ermöglichen.

[27] `http://threejs.org/`
[28] `https://developers.google.com/closure/`

5.5 Das GRIB-Format

Das GRIB[29]-Format wurde 1985 von der World Meteorological Organization (WMO) für den binären Austausch und die Speicherung von meteorologischen Informationen entwickelt, deren räumliche Anordnung ein gleichförmiges Gitter bildet. Nach der Spezifikation der zweiten Version von 2003[30] ist eine GRIB-Datei in neun Bereiche eingeteilt. Die Bedeutungen der wichtigsten Bereiche des Headers sind in folgender Aufzählung zusammengefasst:

Indicator & Identification Section (IS): In diesen Abschnitten lassen sich allgemeine Informationen zu den vorliegenden Daten finden. So kann unter anderem die Version, die Herkunft und der Typ (Analyse oder Vorhersage) bestimmt werden. Weiterhin kann die wichtige Information über den Berechnungs- und Startzeitpunkt der Vorhersage entnommen werden.

Grid Definition Section (GDS): Hier sind Informationen über die Beschaffenheit und Dimension des Datengitters hinterlegt. Es können dabei verschiedene Arten der Projektion auf eine Weltkarte genutzt werden, wie die übliche Längengrad/Breitengrad-, oder die Mercator-Projektion.

Product Definition Section (PDS): Dieser Bereich gibt an, um welche Wetterinformation es sich bei einem einzelnen Datum handelt. Das ist zum einen die Kombination von Höhenlevel und Typ, wie in Abbildung 2.2 abgebildet. Zum anderen ist das Datum und die Länge der Vorhersage angegeben.

Data Representation Section (DRS): Als letzte Meta-Information ist eine Definition des Datentyps der nachfolgenden Daten angegeben. Daran kann festgemacht werden, ob es sich um integer-, oder float-Werte mit einer bestimmten Präzision handelt. Außerdem kann eine Kompressionsart angegeben werden.

[29] `http://www.wmo.int/pages/prog/www/WMOCodes/Guides/GRIB/GRIB1-Contents.html`
[30] `http://www.wmo.int/pages/prog/www/WMOCodes/Guides/GRIB/GRIB2_062006.pdf`

Kapitel 6

Umsetzung

6.1 Server

Damit die beiden im Rahmen dieser Arbeit entwickelten Apps die Möglichkeit haben, ihre Wettervorhersagedaten mobil zu aktualisieren, wurde ein unterstützender Server realisiert. Er stellt die Daten zur Verfügung, die unabhängig von den Apps aktualisiert werden sollen.

In den folgenden Abschnitten wird kurz beschrieben, wie die Ursprungsdaten vom Server-Programm erst abgefragt, ausgelesen und abschließend vorverarbeitet zur Verfügung gestellt werden.

6.1.1 Abfrage und Verarbeitung der Daten

Neben dem in Abschnitt 2.2 vorgestellten Web-Service wird von der NOAA ein Python-Skript[31] zur Verfügung gestellt. Es ermöglicht nach Anzeige der vorhandenen Daten die gezielte Auswahl von bestimmten Wetterelementen und zeitlichen Bereichen aus umfangreichen Vorhersagedaten. Die Daten werden im GRIB-Format ausgeliefert. In dieser Arbeit wurden globale Vorhersagedaten in der derzeit höchsten räumlichen Auflösung von 721×361 Datenpunkten abgefragt. Der Speicherbedarf einer globalen Vorhersage beläuft sich somit auf $721 * 361\ (Datenpunkte) * 4\ Byte\ (Float32) = 0,99\ MB$ für die Daten eines Wetterelements zu einem Zeitpunkt.

Für die Verarbeitung dieser Daten wurde ein Java-Programm entwickelt. Die Bibliothek JGrib[32], dessen Entwicklung mit einer Diplomarbeit von Stark [11] an der Universität Osnabrück begann, wird als Parser für die GRIB-Datei verwendet. Mit dieser Bibliothek ist das Auslesen der Datenpunkte als float-Werte für jeden Wettertyp und Zeitpunkt in der Vorhersage möglich. Nach Ermittlung des globalen Minimum- und Maximum-Werts eines Wetterelements werden die Werte auf das Intervall $[0, 1]$ normiert und zu int-Werten im Intervall $[0, 255]$ konvertiert, um sie in den Apps als Texturdaten mit dem Datentyp GL_UNSIGNED_BYTE verwenden zu können. Diese Daten werden unter Nutzung der Bibliothek java-image-scaling[33] in niedrigere Auflösungen (Zoomstufen) skaliert. Zusätzlich müssen die Einträge aus der GRIB-Datei zeitlich sortiert werden.

[31] http://www.cpc.ncep.noaa.gov/products/wesley/get_gfs.html
[32] http://jgrib.sourceforge.net/
[33] http://code.google.com/p/java-image-scaling/

6.1.2 Speicherung und Bereitstellung der Daten

Die Wetterdaten aus der Vorverarbeitung werden per MySQL-JDBC-Treiber[34] in einer Datenbank abgelegt. Eine Redundanz wird durch die eindeutige Identifikation eines Datensatzes mit dem Wettertyp, dem Zeitpunkt innerhalb der Vorhersage und der Zoomstufe verhindert. In Abbildung 6.1 ist das Entity Relationship (ER)-Diagramm der verwendeten Datentabellen zu sehen, die in der nachfolgenden Beschreibung erklärt werden.

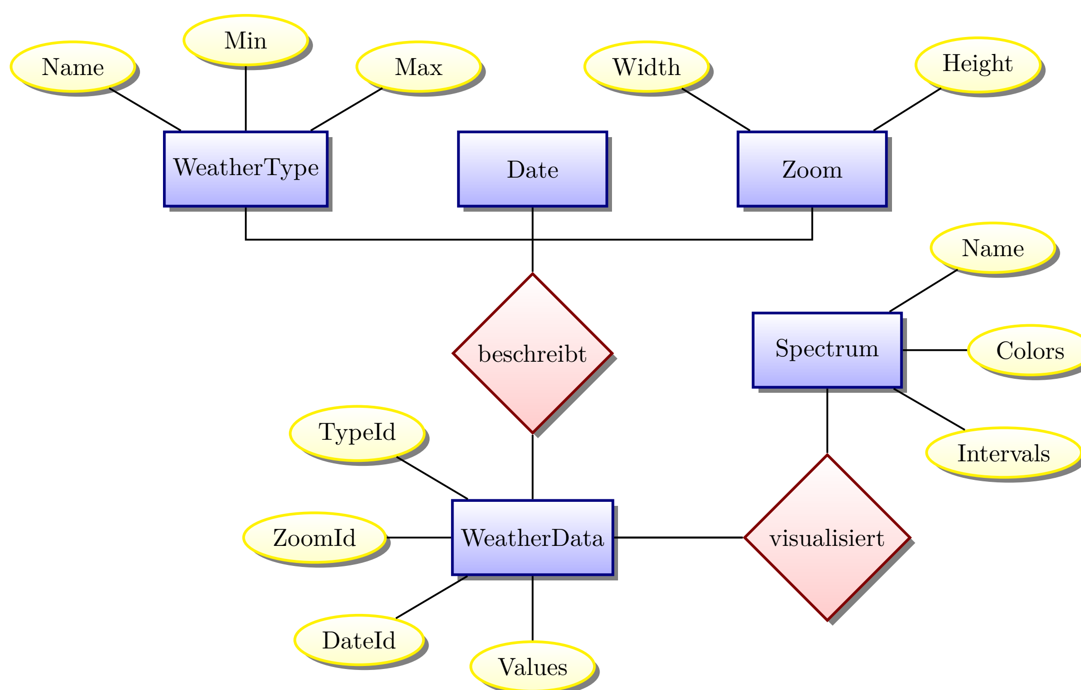


Abbildung 6.1: ER-Diagramm der Server-Datenbank

WeatherType: Ein WeatherType-Eintrag dient zur Information, für welches Wetterelement Daten auf dem Server verfügbar sind. Als Metainformation ist hinterlegt, mit welchen Extrema die Daten dieses Elements normiert worden sind.

Date: In dieser Tabelle ist eingetragen, für welche Vorhersagezeitpunkte Daten verfügbar sind. Dabei ist das Datum und die Uhrzeit des Zeitpunkts angegeben.

Zoom: Für jede Auflösung, auf die in der Vorverarbeitungsphase skaliert wurde, sind die Dimensionen in Breite und Höhe hinterlegt. Ein Eintrag entspricht somit einer Detailstufe, in der Daten auf dem Globus angezeigt werden können.

Spectrum: Ein Farbspektrum wird durch eine Liste von Farben und dazugehörigen Intervallen definiert. Die Farben sind im RGBA-Format angegeben und die Intervalle als ganzzahlige Werte. Die Größe eines Farbintervalls wird später im Programm genutzt, um jeder Farbe eine bestimmte Ausdehnung im Spektrum einzuräumen.

WeatherData: Ein Eintrag in dieser Tabelle ist eindeutig über die Indizes von Typ, Datum und Zoom referenziert. Er enthält die dazugehörigen Wetterdaten als Byte-Werte mit dem Spaltentyp BLOB (binary large object).

Die Wetterdaten sollen von den Apps über eine Web-Schnittstelle abgerufen werden. Dazu wurde ein PHP-Skript geschrieben, das die Einträge der einzelnen Tabellen im JSON-Format ausliefert.

[34] http://dev.mysql.com/downloads/connector/j/

Mit diesem Skript ist es möglich, nur eine Teilmenge der Wetterdaten abzufragen, wenn die Position und Dimension eines rechteckigen Ausschnitts mitgeteilt wird. In Listing 8 ist beispielhaft eine Anfrage zu sehen, die zwei rechteckige Ausschnitte der Größe 54×27 mit unterschiedlichen Positionen anfordert. Für die Übertragung per HTTP-Protokoll werden die binären Daten mit der Base64-Kodierung[35] in Text umgewandelt.

```
1 database?function=RecordValueRequests&requests=
2 [{"type":1,"zoom":2,"date":12,"rect":{"id":0,
3                                        "offset":{"x":0,"y":0},
4                                        "size":{"width":54,"height":27}}},
5  {"type":1,"zoom":2,"date":12,"rect":{"id":1,
6                                        "offset":{"x":54,"y":0},
7                                        "size":{"width":54,"height":27}}}]
```

Listing 8: PHP-Anfrage von zwei unterschiedlichen Teilmengen aus den Wetterdaten

Bei der Entwicklung wurden ein *Apache WebServer 2.4.7*, *MySQL 5.6.14* und *PHP 5.5.6* verwendet.

6.2 Programmablauf und -struktur

In den folgenden Abschnitten wird beschrieben, aus welchen Komponenten die Apps aufgebaut sind und in welcher Weise sie zusammenarbeiten, um die gewünschten Funktionen auszuführen. Die Beschreibungen sind weitgehend für beide Zielplattformen allgemein gehalten und gehen in den späteren Abschnitten auf Details der Implementationen ein. Es wird zunächst die allgemeine Klassenstruktur vorgestellt, um auf dieser Basis die zeitliche Interpolation der Daten und die dynamische Bestimmungen von Teilmengen als zentrale Bestandteile der Apps zu erklären.

6.2.1 Allgemeiner Ablauf und Klassenstruktur

Wie bei Anwendungen mit Echtzeit-Interaktion üblich und in Abbildung 6.2 dargestellt, wird nach Programmstart zunächst eine Initialisierungsphase durchgeführt, in der die benötigten Wetter- und Meta-Daten vom Server geladen werden. Anschließend geht das Programm in eine Schleife über, die für die gesamte Lebensdauer des Programms wiederholt ausgeführt wird. In einem Aufruf dieser Schleife werden zeitabhängige Komponenten, wie das Animations- und Event-System, aktualisiert. Sie verarbeiten unregelmäßige Ereignisse und die Daten, mit denen die Visualisierungs-Komponenten anschließend ein neues Bild zeichnen. Außerdem werden die sichtbaren Bereiche des Globus und eine passende Zoomstufe bestimmt, um die Größe der benötigten Daten zu reduzieren.

In diesem Abschnitt wird das Konzept für die objektorientierte Funktionsweise des interaktiven Visualisierungsprogramms vorgestellt. Die zentralen Funktionalitäten sind in die Packages „model“, „graphics“, „view“ und „controller“ eingeteilt. Diese Aufteilung wurde nach dem von Gamma et. al [5] bekannten MVC-Pattern gewählt. Im Folgenden werden die Bestandteile jedes Packages exemplarisch vorgestellt, um den Programmablauf zu erläutern. Das modulare Konzept hat bei der Entwicklung vor allem die Erweiterbarkeit und Portierung der Anwendung auf die Zielplattformen erleichtert.

[35] http://en.wikipedia.org/wiki/Base64

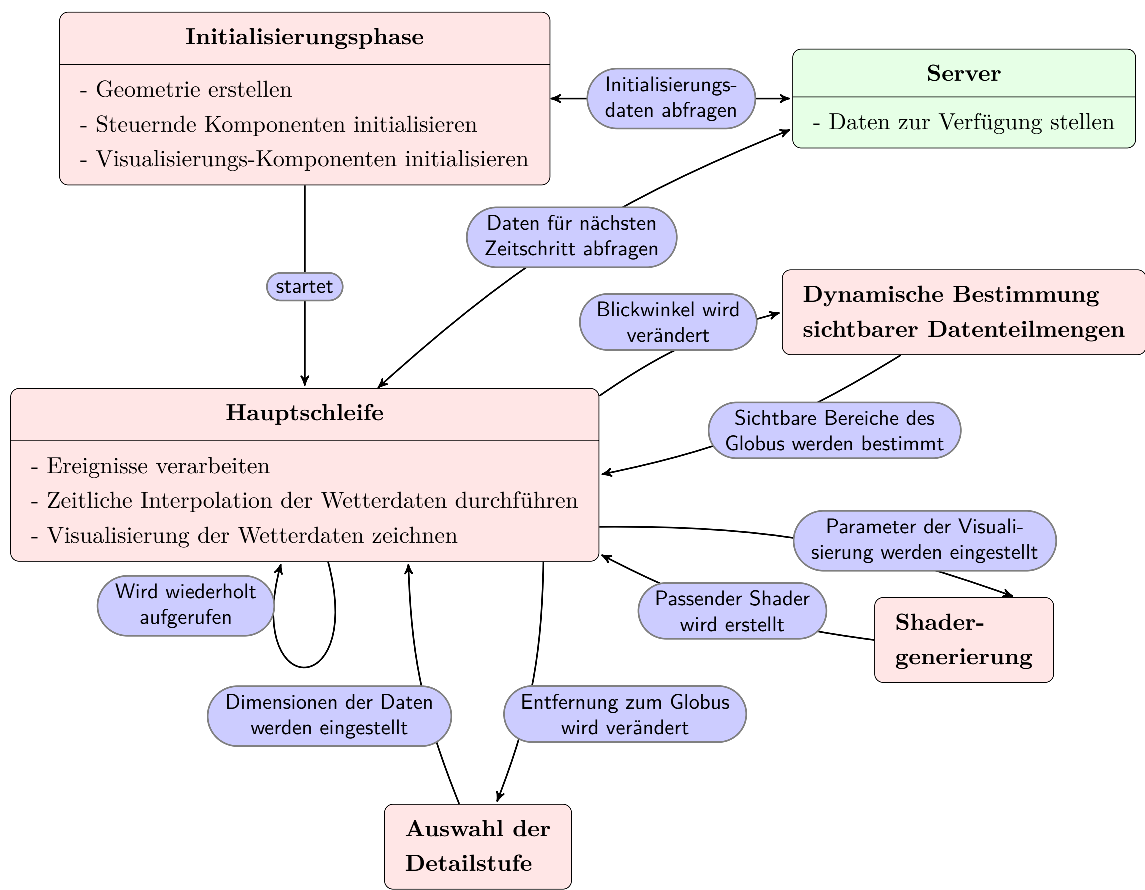


Abbildung 6.2: Diagramm: Ablauf des Programms. Nach einer Initialisierung geht das Programm in die Hauptschleife über, wobei verschiedene Ereignisse (blau) weitere Komponenten (rötlich) des Programms aktivieren oder Anfragen an den Server (grün) senden.

Das „model"-Package

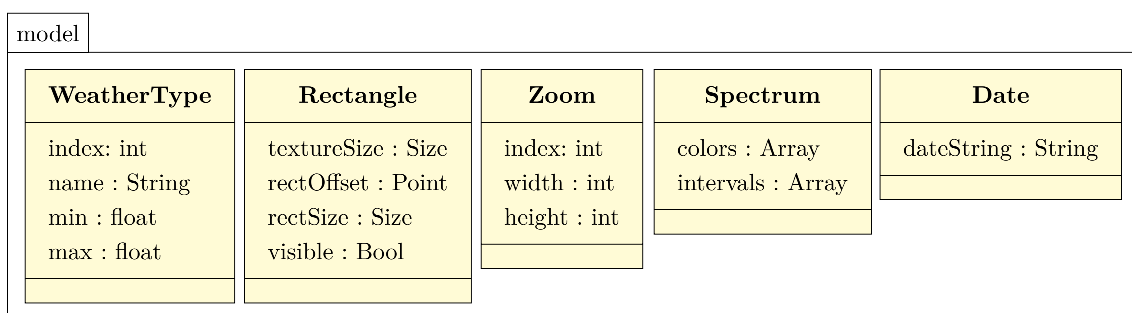


Abbildung 6.3: Klassendiagramm: „model"-Package

Die Klassen des in Abbildung 6.3 dargestellten „model"-Packages dienen zur Speicherung der in Abschnitt 6.1.2 vorgestellten Daten, die beim Programmstart vom Server geladen werden. Zusätzlich werden Objekte der *Rectangle*-Klasse generiert, deren Funktion nachfolgend beschrieben wird:

Rectangle: Objekte dieser Klasse werden generiert, um einen räumlichen Ausschnitt auf die Wetterdaten abzubilden. Für die Generierung werden prozentuale Faktoren angegeben, die die Breite, Höhe und Position eines Rechtecks in Relation zur räumlichen Auflösung der Wetterdaten definieren. Diese Objekte werden für die Bestimmung von sichtbaren Teilbereichen der Erdkugel benötigt (siehe Abschnitt 6.2.3), welche das selektive Nachladen von Teilmengen der Wetterdaten ermöglicht.

Das „view“-Package

In Abbildung 6.4 ist ein Ausschnitt des „view“-Packages dargestellt, dessen Klassen definieren, welche Inhalte auf dem Bildschirm sichtbar sind und wie der Nutzer mit ihnen interagieren kann. Im Folgenden wird die Kamera vorgestellt, mit der der Blickwinkel und Ausschnitt auf den Globus definiert werden kann. Weitere Elemente des Packages werden im Abschnitt 6.4.1 über die Benutzeroberfläche vorgestellt.

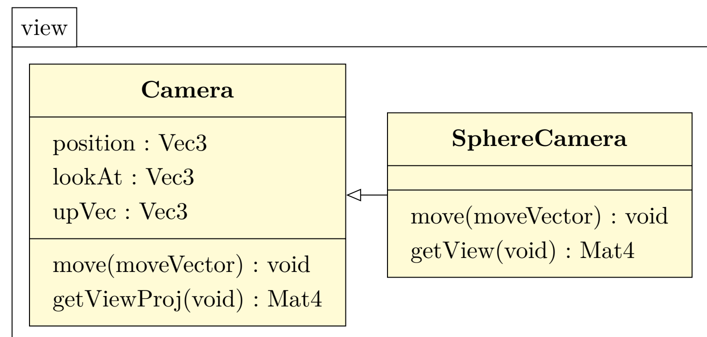


Abbildung 6.4
Klassendiagramm: „view“-Package (Ausschnitt 1)

Camera: Eine Instanz der *Camera*-Klasse dient mit Informationen über die Position, den anvisierten Punkt und eines nach oben zeigenden Vektors als virtuelle Kamera einer OpenGL-Szene. Diese Kamera kann in jeder Dimension bewegt werden und die Art der Projektion bestimmen. Als Projektionsart kommt zum einen die orthogonale Projektion in Frage, die Tiefeninformationen vernachlässigt und zum anderen die perspektivische Projektion, die weiter entfernte Objekte kleiner erscheinen lässt. Um die Funktionsweise der Kamera anzupassen, können Unterklassen erstellt werden.

SphereCamera: *SphereCamera* ist eine solche Unterklasse, welche die Kameraposition in Kugelkoordinaten konvertiert, um die Navigation um den Globus mit einem bestimmten Radius zu ermöglichen. Mit diesem Radius wird außerdem die aktuelle Zoomstufe der angezeigten Wetterdaten ermittelt, um bei geringer Entfernung mehr und bei großer Entfernung niedriger aufgelöste Daten anzuzeigen und im Hintergrund weniger Daten nachzuladen.

Das „graphics“-Package

Das in Abbildung 6.5 dargestellte „graphics“-Package enthält Klassen, die für die Darstellung von Objekten mit OpenGL zuständig sind. Nach dem Factory-Pattern gibt es entweder ein Singleton-Objekt oder statische Klassen-Methoden, die für die Initialisierung eines komplexen Objekts genutzt werden können. Diese Objekte kapseln die imperativen OpenGL-Befehle für die Verwendung in einer objektorientierten Umgebung.

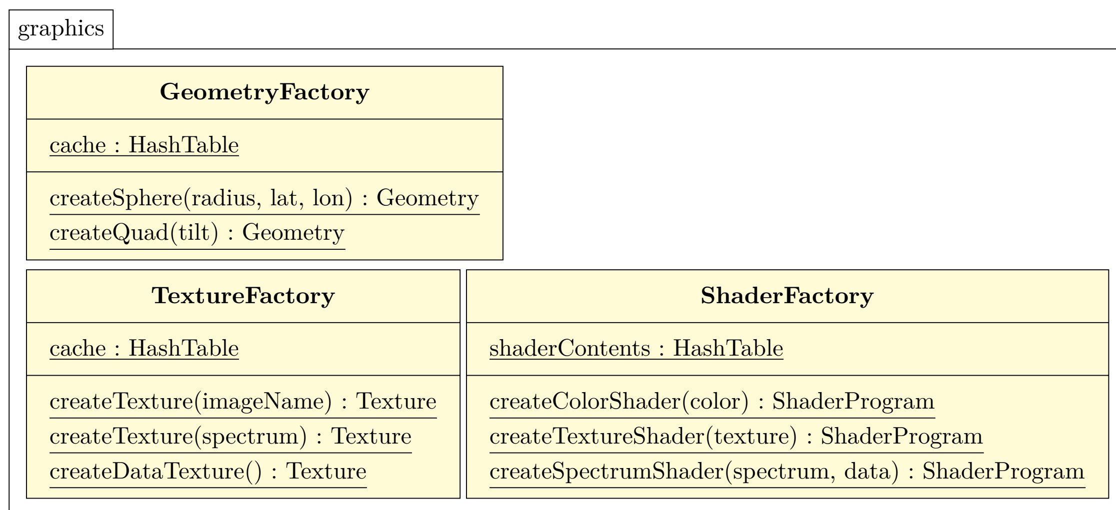


Abbildung 6.5: Klassendiagramm: „graphics“-Package

GeometryFactory: Die Klassenmethoden dieser Klasse dienen der Erstellung von Geometriedaten in Form von OpenGL-Objekten wie Vertex- und Index-Buffer. Es kann z. B. eine Kugelgeometrie oder die eines Quaders angefordert werden. Ein Cache stellt sicher, dass keine Geometrie doppelt erstellt wird.

TextureFactory: Texturen, die für das Einfärben von Geometrie genutzt werden, können mit Hilfe dieser Klasse erstellt werden. Dabei reicht der Funktionsumfang vom Laden einer Bilddatei bis zur Erstellung einer Spektrums-Textur aus den vorher genannten Spektrumsdaten. Von dieser Klasse erzeugte Objekte werden ebenfalls in einem Cache abgelegt, um redundante Generierungen zu verhindern.

ShaderFactory: In dieser Klasse sind Methoden für die einfache Generierung eines Shaderprogramms zu finden. Es kann z. B. ein Shaderprogramm erstellt werden, das die Geometrie mit einer konstanten Farbe einfärbt oder dazu verschiedene Farben aus Texturdaten nutzt.

Das „controller“-Package

Abbildung 6.6 zeigt das Diagramm eines Ausschnitts der Klassen des „controller“-Packages. Sie dienen zur Steuerung des Programmablaufs. Von jeder Klasse dieses Packages existiert zur Laufzeit genau ein Singleton-Objekt mit privaten Variablen und Methoden sowie einer öffentlichen Schnittstelle zu anderen Objekten. In folgender Beschreibung wird der EventController vorgestellt und die weiteren Bestandteile des Packages folgen in den Abschnitten 6.2.2 und 6.2.3.

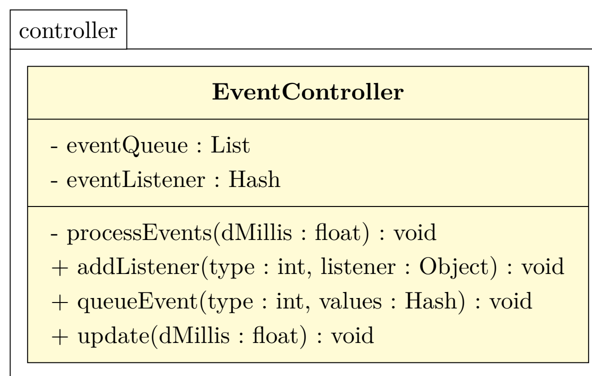


Abbildung 6.6: Klassendiagramm: „controller“-Package (Ausschnitt 1)

EventController: Mit dieser Klasse wird ein Event-System nach dem Observer-Pattern realisiert. Dieses System entkoppelt die Kommunikation der Objekte und sorgt für die erleichterte Erweiterbarkeit. Ein Event ist ein Objekt für den schnellen Austausch von Informationen und ist durch einen eindeutigen Event-Typ definiert. Optional können weitere Werte mitgegeben werden. In der Tabelle 6.1 ist ein exemplarischer Auszug der Event-Typen und ihrer Bedeutung für den Programmablauf aufgelistet. Jedes Objekt kann sich als Listener (Observer) für Events eines oder mehrerer Typen registrieren, wenn eine Schnittstelle implementiert ist, die bei Auftreten eines Events aufgerufen wird. Neue Events könen wiederum an beliebigen Stellen des Programms erzeugt werden und dann in einer Liste abgelegt, die bei jedem Aufruf der Hauptschleife abgearbeitet wird. Dieses System wird vor allem von den Objekten der Benutzeroberfläche genutzt, da dort unregelmäßig Events entstehen, deren Auswirkungen verschiedene Komponenten des Programms betreffen können.

Tabelle 6.1: Auszug der Events des Programms

Typ	Name	Bedeutung
0	ANIMATION_NEWDATE	Wird ausgelöst, wenn die zeitliche Animation ein neues Datum erreicht hat. Trägt als zusätzliche Werte das aktuelle und nächste Datum.
1	WINDOW_NEWSIZE	Signalisiert die Veränderung der Dimensionen des darstellenden Fensters.
2	DATA_NEWDATA	Wenn nach einem asynchronen Ladevorgang neue Wetterdaten bereitgestellt sind, wird dieses Event ohne zusätzliche Werte versendet.
3	GUI_CLICKED	Wird von einem GUI-Element bei einem Klick erzeugt. Zur Identifikation wird der Name und Zustand des sendenden Elements übertragen.

6.2.2 Zeitliche Interpolation der Daten

In diesem Abschnitt wird veranschaulicht, wie die lineare Interpolation der Wetterdaten zwischen zwei Zeitschritten bewerkstelligt wird. Dazu werden die beteiligten Klassen des in Abbildung 6.7 zu sehenden „controller“-Packages vorgestellt. Ihre Zusammenarbeit wird mit Hilfe eines Diagramms noch genau erläutert.

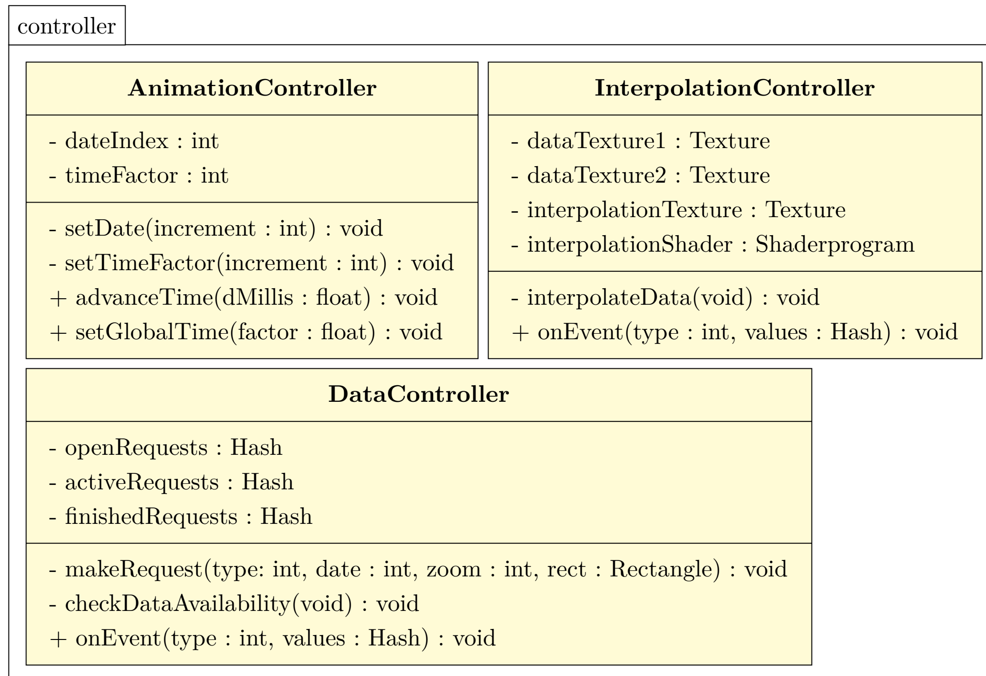


Abbildung 6.7: Klassendiagramm: „controller“-Package (Ausschnitt 2)

AnimationController: Diese Klasse steuert die zeitliche Interpolation durch die vorhandenen Zeitpunkte der Wetterdaten, durch Speicherung des aktuellen Zeitpunkts und eines Zeitfaktors, der den Abstand zum nächsten darstellt. Die *advanceTime*-Methode wird in jedem Frame aufgerufen und inkrementiert den aktuellen Zeitfaktor um einen bestimmten Wert. Parameter sind dabei die Auflösung zwischen zwei Zeitpunkten und die Zeit in Millisekunden, die für ein Inkrement vergehen soll. Dadurch kann der Übergang verfeinert und die Dauer der Animation gesteuert werden. Wenn der Zeitfaktor oder der aktuelle Zeitpunkt sich verändert, wird ein entsprechendes Event generiert. Es ist möglich, das Voranschreiten des Zeitfaktors zu pausieren, zu verlangsamen, oder die Richtung umzukehren.

InterpolationController: Der *InterpolationController* reagiert auf die Events des *AnimationControllers* und realisiert die zeitliche Interpolation der Wetterdaten in ein Textur-Objekt, das von den Visualisierungs-Komponenten genutzt wird. Dazu liegen Referenzen auf drei Textur-Objekte vor. *dataTexture1* und *dataTexture2* werden beim Wechsel des Zeitpunkts jeweils die Wetterdaten des aktuellen und nächsten Zeitpunkts zugewiesen. Die Daten eines Wetterelements belegen dabei einen von vier möglichen Farbkanälen. Diese Texturdaten werden von einem Shaderprogramm unter Berücksichtigung des aktuellen Zeitfaktors linear interpoliert. Das Ergebnis wird durch ein Framebuffer-Objekt in die *interpolationTexture* übertragen, die von anderen Shaderprogrammen genutzt werden kann.

DataController: Die asynchrone Beschaffung und Bereitstellung der Wetterdaten wird mit dem *DataController* umgesetzt. Wenn die Animation einen neuen Zeitpunkt erreicht hat,

wird mit der *makeRequest*-Methode eine Anfrage für neue Daten an den Servers gestellt. Jede Anfrage ist durch einen Rechtecks-Index, einen Wettertyp, einen Zeitpunkt und eine Zoomstufe eindeutig identifizierbar. Durch die Einteilung in offene, aktive und abgeschlossene Anfragen wird Redundanz bei der Datenabfrage verhindert. Die offenen Anfragen werden nach einem der vier Parameter gruppiert und gesammelt an den Server gestellt. Somit können z. B. mehrere Anfragen für einen bestimmten Zeitpunkt gesammelt und ohne wiederholten Verbindungsaufbau gestellt werden. Der *DataController* sorgt dafür, dass die Daten der aktivierten Wetterelemente der aktuellen Zoomstufe des aktuellen und nächsten Zeitpunkts vorhanden sind.

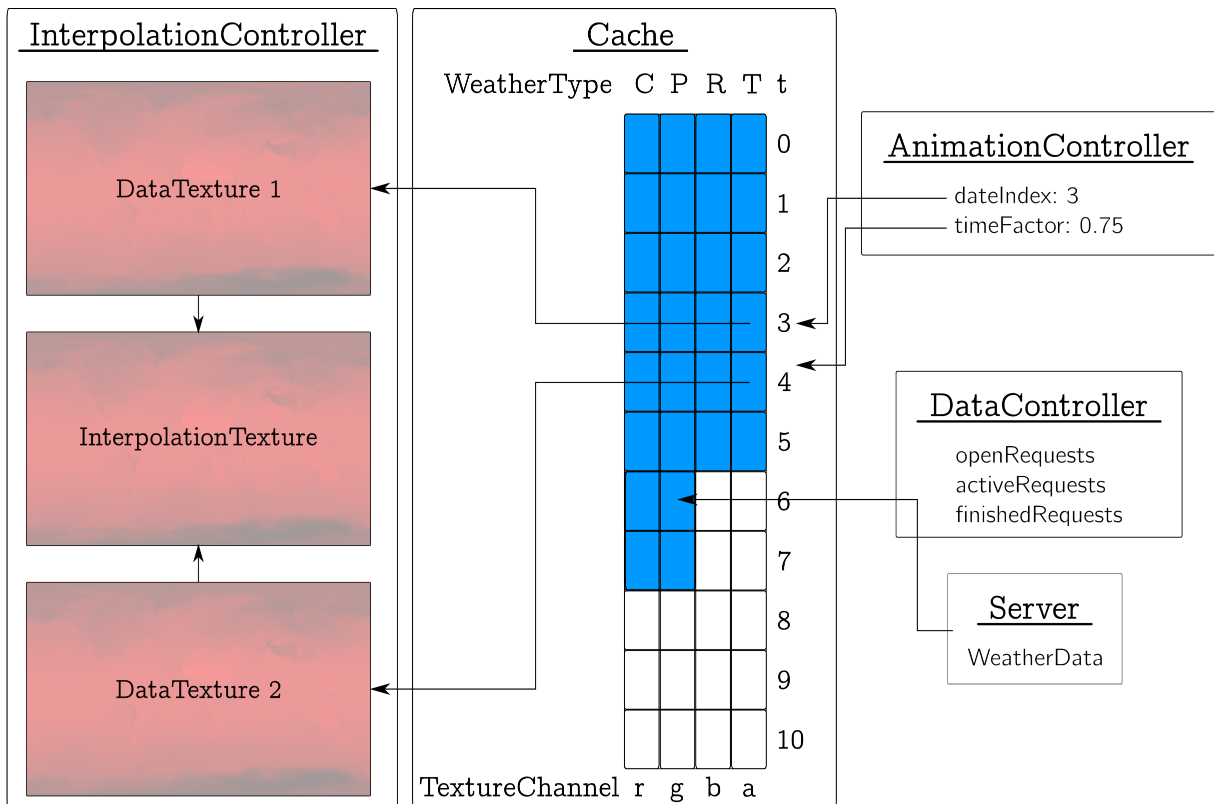


Abbildung 6.8: Diagramm: zeitliche Interpolation der Wetterdaten, blau markierte Bereiche der Daten liegen im Cache vor und werden als Texturdaten mit Hilfe eines Fragment Shaders interpoliert (bei den Texturen sind die Werte der Temperatur rötlich dargestellt)

In Abbildung 6.8 veranschaulicht ein Diagramm die Funktionsweise der vorgestellten Komponenten. Der in der Mitte befindliche Cache dient als Speicherbereich der Wetterdaten eines bestimmten Typs (C - Clouds, P - Pressure, R - Rain, T - Temperature) zu einem bestimmten Zeitpunkt t. Diese Daten werden wie beschrieben im Hintergrund durch den *DataController* vom Server abgerufen. In den farbig markierten Abschnitten liegen Daten vor. Der *AnimationController* bestimmt, an welchem Zeitpunkt (in der Abbildung ist $dateIndex = 3$) sich die Animation befindet und zu welchem Grad der nächste Zeitpunkt erreicht ist. Der *InterpolationController* führt mit dieser Information die lineare Interpolation zwischen den Zeitpunkten t und $t+1$ durch, deren Daten sich in *dataTexture1* und *dataTexture2* befinden. Dieser Vorgang wurde zunächst auf der CPU durchgeführt. Es hat sich allerdings herausgestellt, dass die Nutzung der GPU durch ein Shaderprogramm einen enormen Geschwindigkeitsvorteil bringt.

In Listing 9 ist der FS-Code des Shaderprogramms zu sehen, das die Interpolation mit Hilfe der GPU durchführt. Nachdem die Werte der beiden Datentexturen geladen wurden, wird die GLSL-Funktion *mix(...)* verwendet. Sie führt eine lineare Interpolation nach der Formel $(dataTexture1) * (1 - timeFactor) + (dataTexture2) * timeFactor$ durch. Dieses Shaderprogramm wird auf die Geometrie eines Quaders angewandt, um das Ergebnis über einen nicht sichtbaren Framebuffer in die *interpolationTexture* zu überführen.

```
1  precision highp float;  // hohe float-Präzision
2  varying vec2 v_TexCoords;
3  // Zeitfaktor zwischen den Zeitpunkten t und t+1
4  uniform float u_timeFactor;
5  // die beiden Datentexturen
6  uniform sampler2D u_dataTexture1;
7  uniform sampler2D u_dataTexture2;
8  void main(void) {
9      // Die Daten der Zeitpunkten t und t+1 werden geladen ...
10     vec4 color1 = vec4(texture2D(u_dataTexture1, v_TexCoords));
11     vec4 color2 = vec4(texture2D(u_dataTexture2, v_TexCoords));
12     // ... und per mix(...)-Funktion interpoliert
13     gl_FragColor = mix(color1, color2, u_timeFactor);
14 }
```

Listing 9: Code des Fragment Shaders für die Dateninterpolation

6.2.3 Dynamische Bestimmung von sichtbaren Datenteilmengen

Die Menge der zu übertragenden Wetterdaten an die mobilen Geräte soll so gering wie möglich sein. Da bei Nutzung des Programms zu jeder Zeit nur eine Hälfte des dreidimensionalen Globus sichtbar ist, kann die Übertragung der Daten für die Rückseite eingespart werden. Dafür wird in diesem Abschnitt die Vorgehensweise der dynamischen Bestimmung der benötigten Datenteilmengen vorgestellt. In Abbildung 6.9 ist ein Diagramm der dafür hauptverantwortlichen Klasse *VisibilityController* aus dem „controller“-Package zu sehen und nachfolgend beschrieben.

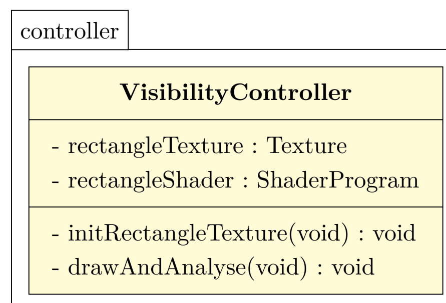
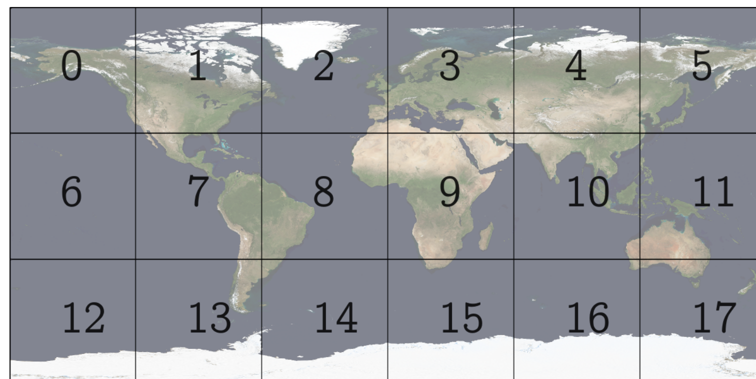


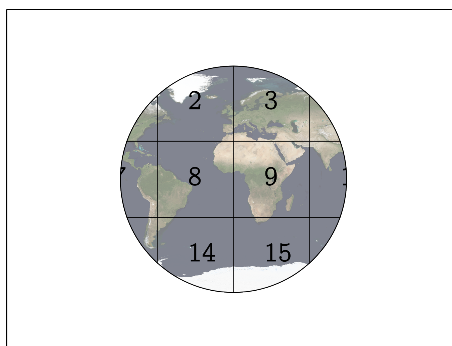
Abbildung 6.9: Klassendiagramm: „controller“-Package (Ausschnitt 3)

VisibilityController: Die Singleton-Instanz dieser Klasse generiert zunächst mit Hilfe der *TextureFactory* eine Textur, die verschiedenfarbige Rechtecke darstellt. Dabei repräsentieren die Farben und Dimensionen der Rechtecke die Indizes und Ausmaße der zuvor generierten *Rectangle*-Objekte. Unter Nutzung eines Shaderprogramms wird mit dieser Textur die Kugelgeometrie in ein nicht sichtbares Framebuffer-Objekt gezeichnet. Dabei wird die gleiche Kamera genutzt, die vom Nutzer um den dreidimensionalen Globus bewegt wird. Der Inhalt dieses Framebuffers wird per *glReadPixels(...)*-Methode vom Grafik- in den

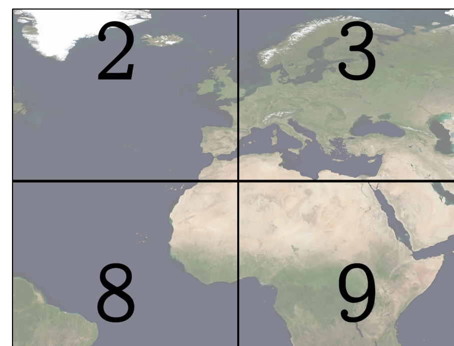
Programmspeicher geladen und analysiert. Die Rechtecke, die den enthaltenen Farben entsprechen, werden als sichtbar gekennzeichnet und fehlende als nicht sichtbar. Dieser Vorgang wird ausgeführt, wenn die Kamera per Event eine Bewegung signalisiert hat.



(a) Die Erdtextur mit 18 Rechtecks-Ausschnitten



(b) Beim Zeichnen der Kugel sind nur Rechtecke auf der Vorderseite sichtbar



(c) Bei hohem Zoomfaktor sind noch vier Rechtecke sichtbar

Abbildung 6.10: Darstellungen der Rechtecks-Textur für die dynamische Bestimmung von sichtbaren Datenteilmengen

Die Funktionsweise des Sichtbarkeitstests ist schematisch in Abbildung 6.10 dargestellt. In (a) ist die Textur zu sehen, die 18 generierten *Rectangle*-Objekten entspricht. Für dieses Beispiel wurde also der Faktor 1/6 für die Breite und 1/3 für die Höhe der Rechtecke ausgewählt. Dabei müssen die Rechtecke allerdings nicht gleiche Dimensionen haben, da die Breite der Rechtecks-Spalte am rechten Rand und Höhe der untersten Rechtecks-Zeile auf den verbleibenden Platz eingestellt wird. Die in (b) gezeichnete Kugel hat noch 12 Rechtecke auf ihrer Vorderseite und bei einem hohen Zoomfaktor wie in (c) sind nur noch vier Rechtecke sichtbar. Nach der Bestimmung der sichtbaren Rechtecke werden nur diese Teilmengen vom *DataController* nachgeladen. Mit dieser Technik wird die Menge der zu übertragenden Daten ohne funktionelle Einschränkungen reduziert.

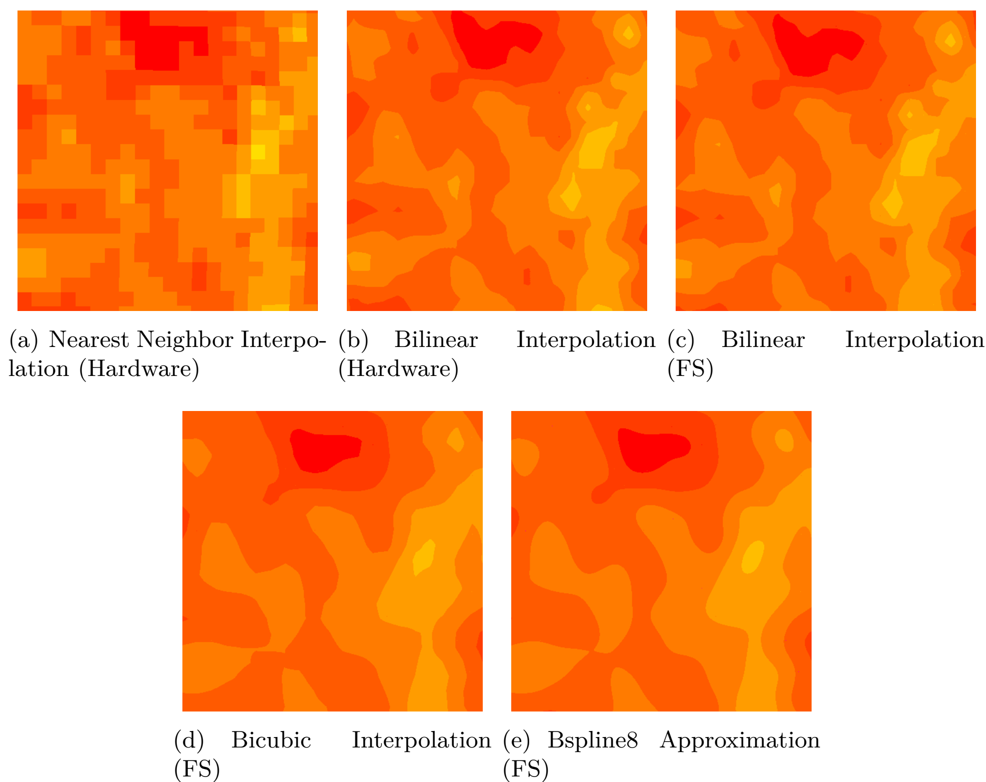
Damit auch die Rechtecks-Textur nur geringen Speicher belegt, wird nur ein Farbkanal für die Indizes der Rechtecke genutzt, wofür OpenGL das Format GL_ALPHA anbietet. Dies vereinfacht den Auslesevorgang aus der Textur, begrenzt aber die Anzahl der unterstützten Rechtecke auf 256, wenn der Datentyp GL_UNSIGNED_BYTE eingestellt ist. Zur Variation der Rechtecks-Textur ist zusätzlich die Möglichkeit gegeben, für jede Zoomstufe unterschiedliche Faktoren für die Dimensionen der Rechtecke zu bestimmen.

6.3 Visualisierung der Wetterdaten

Die Abschnitte dieses Unterkapitels stellen dar, wie die hardwarebeschleunigte Darstellung der Wetterdaten mit den vorgestellten Visualisierungstechniken umgesetzt wurde. Diese Beschreibung wird durch die Vorstellung der verwendeten Interpolations- und Approximations-Techniken eingeleitet und mit der Generierung eines entsprechenden Shaderprogramms abgeschlossen.

6.3.1 Räumliche Interpolation und Approximation

Die höchste Auflösung der aktuell kostenlos verfügbaren globalen Wetterdaten liegt bei 721×361 Datenpunkten. Aktuelle Desktop-Monitore bieten Auflösungen ab 1920×1080 und das iPhone seit der 6. Generation (iPhone 5) eine Auflösung von 1136×640, welche weit über die Datenauflösung hinausgehen. Um eine hohe visuelle Qualität der Daten auf diesen Geräten sicherzustellen, werden die Werte in den Zwischenräumen der Daten angenähert oder geglättet. In dieser Arbeit wurden verschiedene Techniken dazu verwendet. Der Hauptanteil davon findet im FS statt. Die Ausführungsgeschwindigkeit dieses Programms hängt neben der Anzahl der zu zeichnenden Fragments stark von der Anzahl der Texturzugriffe ab. Nachfolgend sind die verwendeten Techniken mit der Anzahl dieser Zugriffe und einer Einschätzung ihrer visuellen Qualität beschrieben (für mehr Details siehe [12]) und in Abbildung 6.11 ist in Ausschnitten der WebGL-Version zu sehen, wie sich die verschiedenen Techniken auf die Einfärbung mit einem diskreten Farbspektrum auswirken.



(a) Nearest Neighbor Interpolation (Hardware) (b) Bilinear Interpolation (Hardware) (c) Bilinear Interpolation (FS)

(d) Bicubic Interpolation (FS) (e) Bspline8 Approximation (FS)

Abbildung 6.11: Räumliche Interpolation und Approximation (WebGL)

- **Nearest Neighbor Interpolation** (Grafik-Hardware, schlechte Qualität)
  Bei der Nearest Neighbor-Interpolation wird der Wert des zur Zwischenposition nächstgelegenen Datenpunkts ausgewählt.
- **Bilineare Interpolation** (Grafik-Hardware, gute Qualität)
  Bei der linearen Interpolation wird der relative Abstand der Zwischenposition zwischen zwei Datenpunkten als Gewichtungsfaktor der Werte der Datenpunkte verwendet. Bei bilinearer Interpolation wird dieser Vorgang in zwei Richtungen (z. B. x- und y-Richtung in einer zweidimensionalen Textur) durchgeführt. Diese Methode ist in der OpenGL-Hardware integriert.
- **Bilineare Interpolation** (Shader, 4 Texturzugriffe, gute Qualität)
  Es wird nicht die integrierte bilineare Interpolation genutzt, sondern im FS berechnet. Diese Variante kann Ungenauigkeiten der integrierten Technik ausbessern.
- **Bikubische Interpolation** (Shader, 16 Texturzugriffe, sehr gute Qualität)
  Bei der in dieser Arbeit verwendeten bikubischen Interpolation werden zwei kubische Polynome durch jeweils vier Stützpunkte pro Richtung gelegt, mit dessen Hilfe der Wert der Zwischenposition bestimmt wird. Mit dieser Anzahl von Stützpunkten gehen die Werte zweier angrenzenden Datenpunkten stetig ineinander über.
- **B-Spline Approximation** mit 8 Kontrollpunkten (BSpline8) (Shader, 64 Texturzugriffe, sehr gute Qualität)
  Der Wert an der Zwischenposition wird mit Hilfe von zwei B-Spline-Kurven bestimmt, die jeweils durch acht Kontrollpunkte pro Richtung gesteuert werden. Im Gegensatz zu den Interpolationstechniken muss diese Kurve nicht durch die Stützpunkte verlaufen.

6.3.2 Farbspektrum

Die Einfärbung von verschiedenen Bereichen des Globus wurde mit Hilfe einer Spektrumstextur umgesetzt. Diese Textur wird aus den Farb- und Intervall-Daten eines *Spectrum*-Objekts generiert. Da die Wetterdaten auf das Intervall $[0, 1]$ normiert wurden, können sie im FS nach der räumlichen Interpolation aus der *interpolationTexture* als x-Wert der Texturkoordinate für die Spektrumstextur verwendet werden. Niedrige Werte erzeugen somit Farben im linken Bereich des in Abbildung 6.12 beispielhaft dargestellten Spektrums und hohe Werte im rechten Bereich. Wegen der in OpenGL-Hardware eingebauten Texturinterpolation ist es leicht möglich, ein diskretes oder kontinuierliches Spektrum zu realisieren. Diese Technik kann auch verwendet werden, um die Isolinien oder Partikel einzufärben.

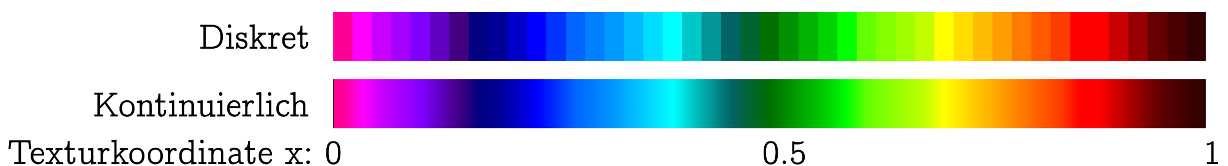


Abbildung 6.12: Diskretes und kontinuierliches Farbspektrum

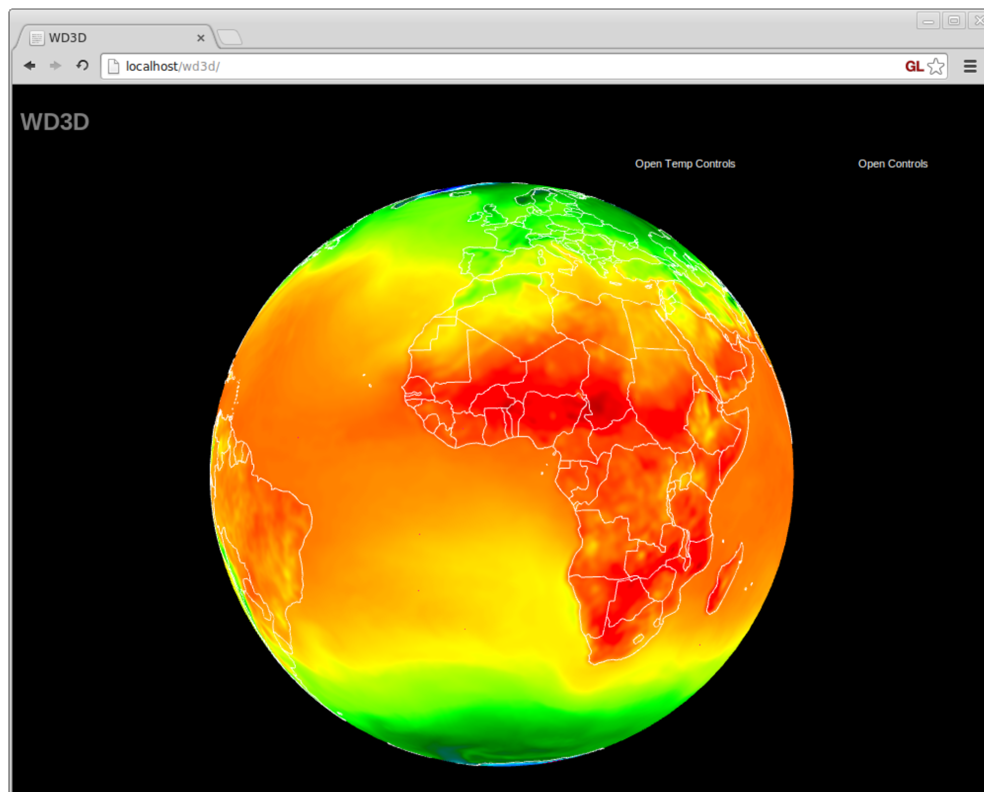


Abbildung 6.13: Umsetzung des Farbspektrums, mit dessen Hilfe die Temperatur dargestellt wird (WebGL/B-Spline8/Kontinuierlich)

In den Abbildungen 6.13 und 6.14 ist zu sehen, wie das Farbspektrum in den beiden Apps dieser Arbeit umgesetzt wurde.

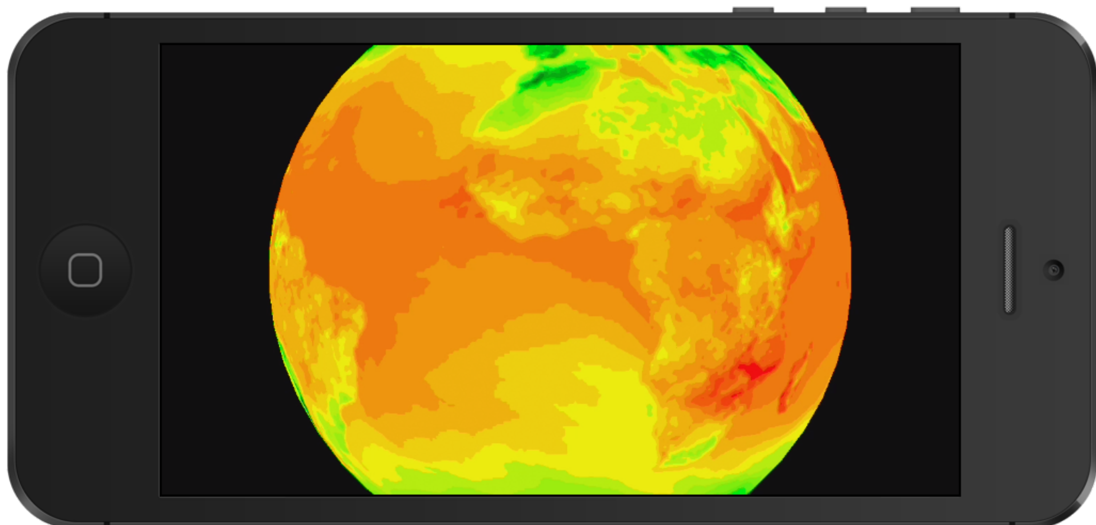


Abbildung 6.14: Umsetzung des Farbspektrums, mit dessen Hilfe die Temperatur dargestellt wird (iOS/Bilinear(Shader)/Diskret)

6.3.3 Isolinien

Eine Isolinie verbindet zwei Datenpunkte mit gleichem Wert, der Iso-Wert genannt wird. In einem Datengitter entstehen so durchgängige Linien, die zwei verschiedene Wertebereiche trennen und der Iso-Wert wird zum Schwellwert zwischen diesen Bereichen.

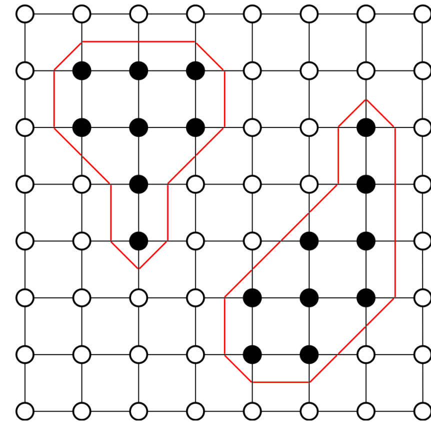


Abbildung 6.15
Liniengenerierung im Datengitter

Um solche Linien zu generieren, gibt es elementare Contour Plot Verfahren wie den Marching Squares [7]- oder CONCREC [1]-Algorithmus. In Abbildung 6.15 ist das Ergebnis eines solchen Verfahrens zu sehen. Schwarz ausgefüllte Kreise stehen in dem Beispiel für einen Wert, der über dem Iso-Wert liegt und die Werte der weiß ausgefüllten Kreise liegen darunter. Zwischen den Datenpunkten wird linear interpoliert. Die Eckpunkte der roten Linien innerhalb einer Zelle liefert der Marching Squares-Algorithmus. Dazu entscheidet er für jede Gitterzelle anhand der angrenzenden Datenpunkte, wie eine Linie durch sie verläuft. Mit diesen Informationen können die Linien als Geometrie gezeichnet werden.

Die beiden genannten Verfahren wurden in Wenkes Vorarbeit [13] hinsichtlich der Darstellungsqualität bei animierten Luftdrucklinien analysiert. Als Ergebnis wurde eine Variante des Marching Squares-Algorithmus entwickelt, aus dessen generierten Linien zusammenhängende Linienzüge extrahiert wurden. Diese konnten anschließend geglättet werden, um die Darstellungsqualität zu erhöhen.

```
1  bool isoline(sampler2D tex, vec2 texCoords) {
2      float ISOLINE_COUNT = 5.0; // bildet die Wetterwerte auf breiteren Bereich ab
3      // Der Abstand der Texturkoordinate zu den Ecken des Fragments
4      vec2 dTexCoord =  0.5 * fwidth(texCoords);
5      // Der Wert der unteren linken Ecke wird ausgelesen und konvertiert
6      float s = texCoords.s - dTexCoord.s; float t = texCoords.t - dTexCoord.t;
7      int iso0 = int(floor(ISOLINE_COUNT *
8                           interpolateFromTexture(tex, s, t)));
9      // ... die untere rechte Ecke
10     s = texCoords.s + dTexCoord.s; t = texCoords.t - dTexCoord.t;
11     int iso1 = int(floor(ISOLINE_COUNT *
12                          interpolateFromTexture(tex, s, t)));
13     // ... das Gleiche für die obere linke und obere rechte Ecke
14     // der Vergleich der Werte wird zurückgegeben
15     return (iso0 == iso1 && iso1 == iso2 && iso2 == iso3);
16 }
```

Listing 10: Isolinien-Funktion im Fragment Shader

In dieser Arbeit wurde das Marching Squares Verfahren testweise in der JS-App implementiert, allerdings aus zwei Gründen in der finalen Version verworfen. Einerseits entsprach die Laufzeit des Algorithmus für die Generierung der Liniengeometrie nicht den Echtzeit-Anforderungen dieser Arbeit, wenn sie in jedem Animationsschritt ausgeführt werden. Andererseits liegt es an der Entscheidung, die Wetterdaten zwischen zwei Zeitschritten mit Hilfe eines Shaderprogramms zu interpolieren und das Ergebnis in eine Textur zu überführen. Der Zugriff auf die berechneten

Daten ist somit wie bei der Bestimmung der sichtbaren Datenteilmengen in Abschnitt 6.2.3 nur über die *glReadPixels(...)*-Methode möglich, die allerdings sehr zeitintensiv ist.

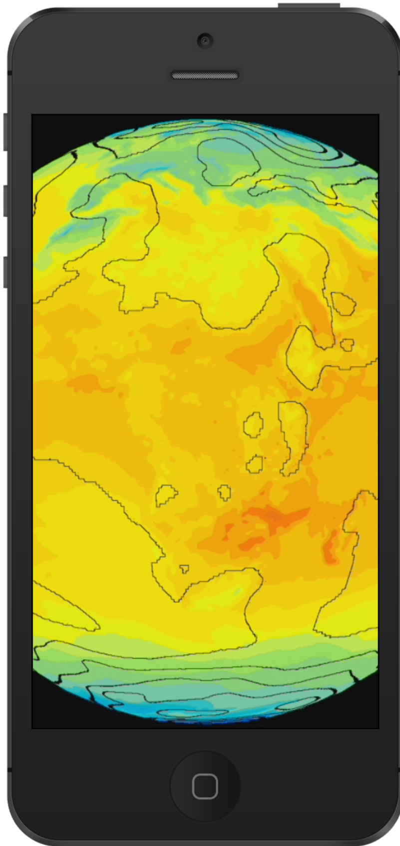


Abbildung 6.16
Umsetzung der Isolinien (+Temperatur) (iOS/Bilin(Shader))

Wegen dieser Erkenntnisse wurde das Zeichnen der Isolinien mit einem Shaderprogramm realisiert. In dem FS dieses Programms werden die Texturkoordinaten der Ecken eines Fragments ermittelt. Deren Wetterwert, wie z. B. der Luftdruck, werden unter Verwendung einer Interpolations-Methode aus der *interpolationTexture* ausgelesen. Nach einer Konvertierung aus dem Einheitsintervall auf einen größeren Wertebereich und anschließender Abrundung werden diese vier Werte miteinander verglichen. Wenn einer der Werte unterschiedlich ist, verläuft eine Iso-Linie durch dieses Fragment und es wird eingefärbt. Andernfalls wird es verworfen. In Listing 10 ist der GLESSL-Code der Funktion zu sehen, mit der dieser Test durchgeführt wird.

Ein Nachteil dieser Methode sind die vier Interpolationsvorgänge, die für die Werte der Ecken benötigt werden und somit hohen Aufwand verursachen. Damit mit dieser Methode anstelle einer gleichmäßigen Verteilung über einen Wertebereich, mehrere bestimmte Iso-Werte dargestellt werden, könnte sie z. B. durch ein Spektrum erweitert werden.

Abbildung 6.16 präsentiert, wie die Umsetzung der Isolinien auf dem iPhone gelungen ist.

6.3.4 Partikel

Um die Windrichtung- und stärke zu visualisieren, werden Partikel verwendet. Für ihrer Darstellung wurde die OpenGL-Topologie GL_POINTS ausgewählt und ein Partikelsystem implementiert, das anhand Abbildung 6.17 erläutert wird.

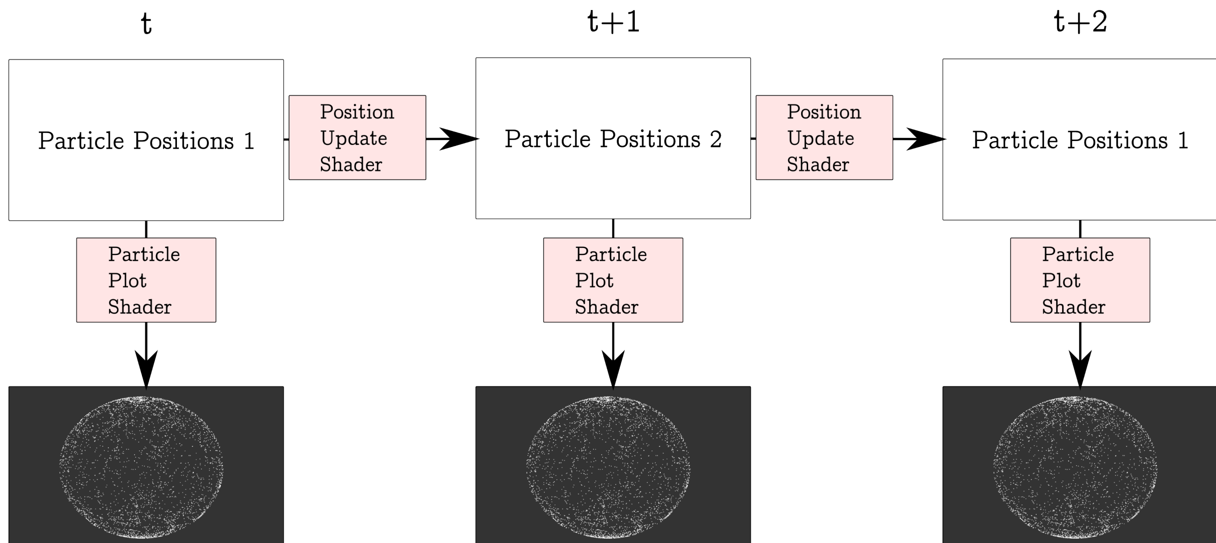


Abbildung 6.17: Ablauf des Partikelsystems

Bei der Initialisierung des Systems werden zunächst die beiden Texturen *Particle Positions 1* und *Particle Positions 2* generiert. Sie enthalten zweidimensionale Koordinaten im Intervall $[0, 1]$.

Das *Particle Plot*-Shaderprogramm wird zur Anzeige der Partikel verwendet. Es zeichnet die Vertices, denen per Texturkoordinate eine bestimmte Position in der *Particle Positions*-Textur zugeordnet ist. Die Koordinaten an diesen Positionen werden im VS aus der *Particle Positions*-Textur gelesen und daraus dreidimensionale Kugelkoordinaten berechnet. Jeder Vertex wird an diesen Punkt verschoben, damit sich die Partikel auf einer Kugeloberfläche befinden.

Damit die Partikel sich über die Zeit bewegen, existiert das *Position Update*-Shaderprogramm. In dessen FS werden die Positionen aus einer *Particle Positions*-Textur gelesen, verarbeitet und das Ergebnis über einen Framebuffer in die andere *Particle Positions*-Textur überführt. In Abbildung 6.17 ist zu sehen, wie sich die Texturen bei jedem Zeitschritt abwechseln und somit die Double Buffering-Technik ausüben. Das *Particle Plot*-Shaderprogramm positioniert und zeichnet zu jedem Zeitpunkt die Partikel mit der Textur, die mit den aktualisierten Positionen beschrieben wurde. Somit entsteht eine Animation der Partikel.

Für die Aktualisierung der Partikelposition werden die Werte der Windrichtung und -stärke aus der *interpolationTexture* an der aktuellen zweidimensionalen Position des Partikels ausgelesen. Die Windrichtung liegt als auf den Einheitsintervall normierter Winkel vor, der mittels Polarkoordinaten zu einem kartesischen Richtungsvektor umgerechnet wird. Dieser Vektor wird mit der Windstärke skaliert und auf die alte Position addiert. Durch diese Berechnungen werden die Partikel unter Berücksichtigung der korrekt zugeordneten Wetterdaten bewegt und stellen den Windverlauf dar.

```
1  #define M_PI 3.14159265358979323846264338327950
2  uniform float u_velocity; // globale Geschwindigkeit
3  uniform sampler2D u_positionTexture; // referenzierte Positions-Textur
4  uniform sampler2D u_interpolationTexture; // Wetterdaten
5  varying vec2 v_TexCoords;
6  void main(void) {
7      // Die aktuelle Partikel-Position wird ausgelesen
8      vec2 oldPosition = texture2D(u_positionTexture, v_TexCoords).rg;
9      vec2 interpolValue = texture2D(u_interpolationTexture, oldPosition.xy).rg;
10     float angle = interpolValue.x; // Der Winkel der Windrichtung
11     float strength = interpolValue.y; // Die Windstärke
12     // Der Verschiebungsvektor wird berechnet
13     vec2 direction = vec2(-sin(angle * (M_PI * 2.0)), -cos(angle * (M_PI * 2.0)));
14     // Die neue Position wird berechnet
15     vec2 newPosition = oldPosition + u_velocity * strength * direction;
16     // und in die zweite Positions-Textur geschrieben
17     gl_FragColor = vec4(mod(newPosition.x, 1.0), mod(newPosition.y, 1.0), 0, 0);
18 }
```

Listing 11: Ausschnitt des Position Update-FS

In Listing 11 ist ein vereinfachter Ausschnitt des FS gezeigt, der die beschriebenen Schritte für die Aktualisierung der Partikelpositionen durchführt.

Bei dieser Technik wurde statt des Datentyps GL_UNSIGNED_BYTE für die Positions-Texturen der Datentyp GL_FLOAT verwendet. Diese erhöhte Genauigkeit wird benötigt, weil die Bewegungen der Partikel sonst unstetig erscheinen. Außerdem wurde jedes Partikel mit einem zufälligen Wert versehen, der eine Zeit definiert, ab der die Position eines Partikels wieder auf den Ursprungswert gesetzt wird. Ohne diesen zusätzlichen Mechanismus kommt es vor, dass einige Partikel dem Windstrom für eine Weile folgen, danach aber an einer ungünstigen Position verharren.

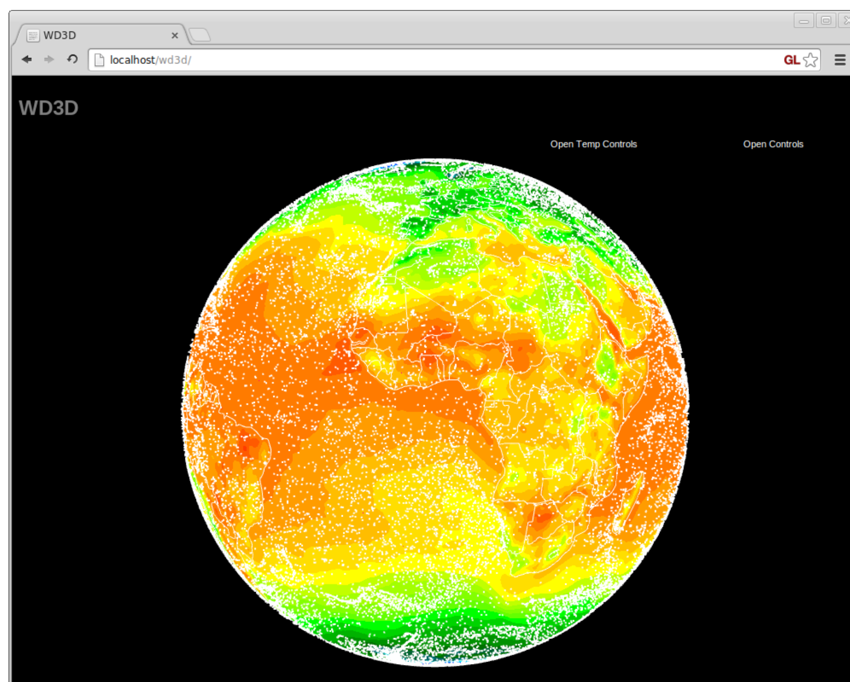


Abbildung 6.18: Umsetzung der Partikel (+Temperatur) (WebGL/BSpline8)

6.3.5 Spezielle Symbole

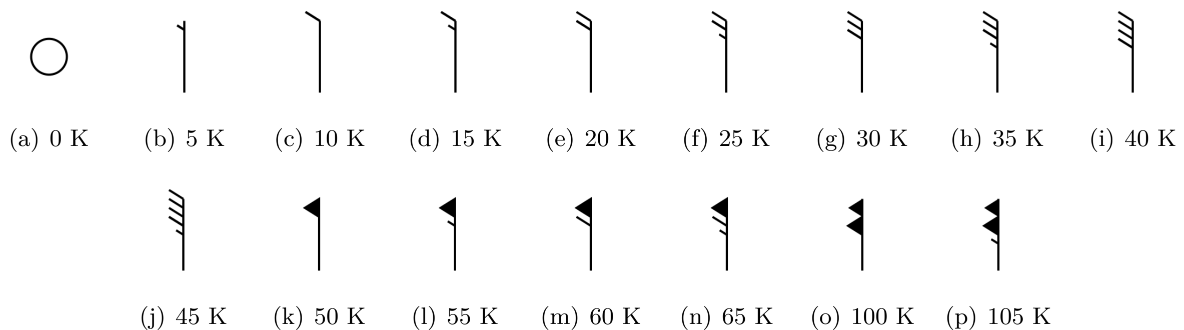


Abbildung 6.19: Windfahnen-Symbole, die verschiedene Windstärken in Knoten symbolisieren

Um spezielle Symbole zu visualisieren, wurde in dieser Arbeit eine Technik entwickelt, mit der z. B. Windfahnen dargestellt werden können, die von Seglern und Fliegern zum Ablesen der Windstärke und -richtung genutzt werden. Sie wurden mit einer texturierten Vierecks-Geometrie umgesetzt, die ähnlich den Partikeln an mehreren Positionen auf der Kugeloberfläche platziert und zusätzlich nach der Kugelnormalen an ihrer Position ausgerichtet wurde. Diese texturierten Vierecke werden in diesem Abschnitt auch Sprites genannt.
Das Muster, nach dem die Windgeschwindigkeit in Knoten von den Fahnen abgelesen kann, lässt sich aus Abbildung 6.19 ableiten. Ein kurzer Strich steht für 5 Knoten, ein langer für 10 und ein Dreieck für 50 Knoten. Diese Werte werden pro Fahne addiert. Die zusammengesetzten Fahnen-Symbole werden nacheinander in horizontaler Reihe in einer Symbol-Textur abgespeichert.

Das *Symbol*-Shaderprogramm stellt die Symbole auf dem Globus dar. Im Gegensatz zu den Partikeln wird die Position der Sprites nicht berechnet, sondern aus einer vorberechneten Geometrie entnommen. Die Hauptaufgabe des Shaderprogramms liegt bei dieser Technik in der Berechnung der richtigen Texturkoordinaten eines Sprites für die Symbol-Textur.

```
1  #define M_PI 3.14159265358979323846264338327795
2  // ... attribute Position, TexCoords
3  attribute vec2 a_SymbolCoords; // ist die Position auf der Kugel
4  uniform sampler2D u_interpolationTexture; // die Wetterdaten
5  // ... uniforms, varyings
6  void main(void) {
7  // ... Positions-Berechnung
8  float windStrength = texture2D(u_interpolationTexture, a_SymbolCoords).g;
9  float windDirection = texture2D(u_interpolationTexture, a_SymbolCoords).r;
10 float rotation = windDirection * M_PI * 2.0;
11 // Rotations-Matrix und Abstand werden berechnet und an den Fragment Shader geleitet
12 v_rotationMatrix = mat2(cos(rot), -sin(rot), sin(rot), cos(rot));
13 v_offset = vec2(floor(windStrength * 16.0)/16.0, 0.0);}
```

Listing 12: Ausschnitt des Vertex Shader-Code für spezielle Symbole

Dazu wird den vier Vertices eines Sprites eine zusätzliche Koordinate zugeordnet, welche die Position des Vierecks innerhalb der Textur der aktuellen Wetterdaten repräsentiert. Im VS (Listing 12) werden damit die Wetterdaten für Windstärke und -richtung ausgelesen. Mit der Windstärke wird ein Abstand zum linken Rand der Symbol-Textur berechnet und damit die richtige Windfahne ausgewählt. Um diesen Abstand bestimmen zu können, hat jedes Fahnensymbol in der

Textur die gleichen Dimensionen.
Mit der Windrichtung wird eine Rotationsmatrix generiert, mit der die Texturkoordinaten dem Wert entsprechend rotiert werden. Der Abstand und die Rotationsmatrix werden an den FS übergeben, der damit das Sprite mit dem richtigen rotierten Symbol einfärben kann. Die Umgebung der Symbole in der Textur ist transparent, damit sie nicht den Globus verdecken, auf dem sie gezeichnet werden. Abbildung 6.20 zeigt das Ergebnis der iOS-Implementation.

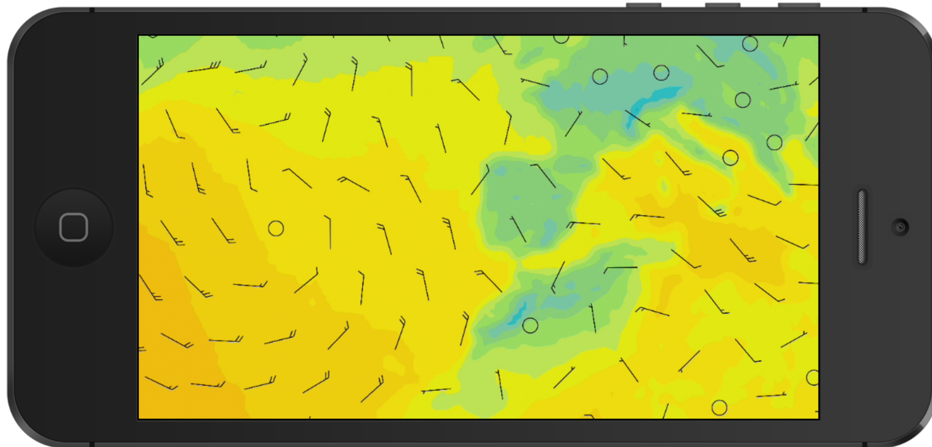
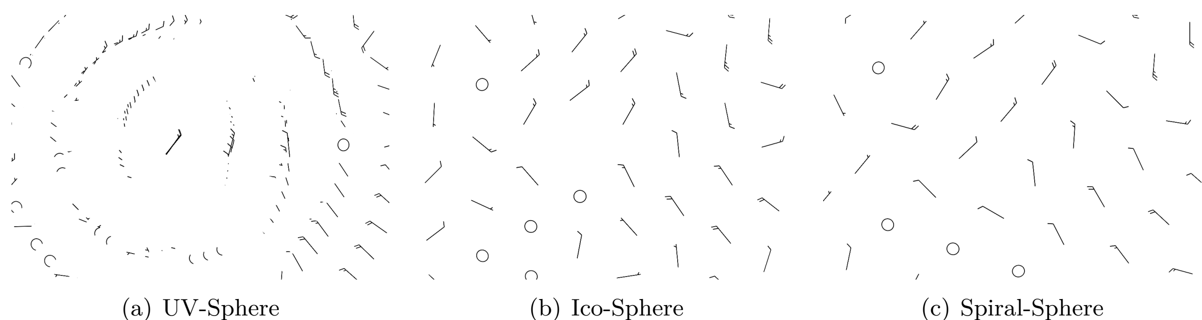


Abbildung 6.20: Umsetzung der Windfahnen (+Temperatur) (iOS/Bilinear(Shader))

Es hat sich bei der Umsetzung dieser Technik herausgestellt, dass die Geometrie, die zur Platzierung der Sprites verwendet wird, bestimmte Voraussetzungen erfüllen sollte. Damit die Symbole erkennbar bleiben, sollten sie mit ausreichendem Abstand zueinander und möglichst gleichmäßig auf der Kugel verteilt sein. In Abbildung 6.21 ist ein Vergleich von verschiedenen Geometrien mit jeweils dem gleichen Blickwinkel auf eine Polstelle gezeigt. Die Geometrie einer UV-Sphere (a), die für den Globus verwendet wird, leistet diese Eigenschaften nicht, da die Koordinaten vom Äquator zu den Polstellen enger aneinanderrücken. Eine Ico-Sphere (b), die durch mehrfaches Unterteilen eines Ikosaeders entsteht, eignet sich besser. Ein ähnliches Ergebnis wie die Ico-Sphere liefert auch eine Spirale (c), die vom Nordpol zum Südpol um den Globus gespannt wird. Die Komplexität der Geometrie kann adaptiv mit der Zoomstufe verändert werden, um eine übersichtliche Verteilung zu gewährleisten.



(a) UV-Sphere (b) Ico-Sphere (c) Spiral-Sphere

Abbildung 6.21: Verschiedene Geometrien für Positionen der Windfahnen-Symbole mit Blickwinkel auf die Polstelle (WebGL)

Zusätzlich hat sich zu Beginn der Implementierung gezeigt, dass texturierte Partikel mit der GL_POINTS-Geometrie für die Darstellung von speziellen Symbolen nur bedingt einsetzbar

sind. Diese Geometrie hat die Eigenschaft, dass die Seiten der erzeugten Vierecke immer parallel zu den Begrenzungen des Bildschirmausschnitts (Viewports) verlaufen. Dies führte bei Kamerabewegungen vor allem an den Polstellen zu einer verfälschten Rotation der Symbole, weshalb sie für einen sinnvollen Einsatz in einer dreidimensionalen Ansicht nicht geeignet sind. Bei einer zweidimensionalen Ansicht einer Wetterkarte würde dieses Problem nicht auftreten und GL_POINTS könnten einen Vorteil aufgrund der geringeren Anzahl von Vertices bringen.

6.3.6 Shadergenerierung

Aus den Anforderungen geht hervor, dass die verschiedenen Visualisierungstechniken kombinierbar und vielfältig einsetzbar sein sollen. Das heißt z. B. ein Wetterelement nicht auf eine Darstellungsform zu begrenzen. Von den umgesetzten Techniken können das Farbspektrum und die Isolinien für jedes eindimensionale Wetterelement eingesetzt und kombiniert werden. Zusätzlich gibt es unterschiedliche Interpolations- und Approximationstechniken mit verschiedenen Auswirkungen auf die Darstellungsqualität.

Eine Möglichkeit der Kombination dieser Darstellungsformen wäre, die Geometrie des Globus mehrfach hintereinander mit den entsprechenden Shaderprogrammen zu zeichnen und die Farbe mittels OpenGL-Blending mischen zu lassen. Diese Vorgehensweise hat allerdings folgende Nachteile:

- häufiger Wechsel des aktiven Shaderprogramms und Textur
- wiederholtes Zeichnen der komplexen Kugel-Geometrie
- Anzahl der Variationen und somit der verschiedenen Shaderprogramme ist recht groß
- die Farbmischung des OpenGL-Blending ist auf bestimmte Parameter begrenzt

Nach diesen Überlegungen wird mit der *ShaderFactory* für die Anzeige von Farbspektren und Isolinien ein einziges Shaderprogramm aus verschiedenen Bausteinen generiert. Partikel und spezielle Symbole lassen sich nicht integrieren, da sie mit ihrer eigenen Geometrie gezeichnet werden. Um für jedes Wetterelement eine individuelle Auswahl dieser Bausteine zu ermöglichen, wurde die *TypeConfig*-Klasse angelegt. In Abbildung 6.22 sind die Instanzvariablen dieser Klasse zu sehen, die kurz beschrieben werden:

TypeConfig
typeIndex : int name : String visType : String specIndex : int interpolType : String

Abbildung 6.22
Klassendiagramm: *TypeConfig*

typeIndex: Der eindeutige Index des Wetterelements.

name: Der Name des Wetterelements.

visType: Die Visualisierungsart kann „Spektrum“ oder „Isolinie“ sein.

specIndex: Der eindeutige Index des Spektrums, das verwendet werden soll.

interpolType: Berechnung der Werte im Shader per „Bilin“, „Bicubic“ oder „BSpline8“.

Das bestehende Shaderprogramm wird entfernt, wenn einer dieser Parameter vom Nutzer geändert wurde und mit den aktuellen Informationen neu generiert. Dieser Vorgang beansprucht wenig Zeit.

6.4 Benutzeroberfläche und Interaktion

In den folgenden Abschnitten wird erläutert, wie verschiedene Elemente der Benutzeroberfläche in den Apps umgesetzt wurden. Es wurden Elemente entwickelt, die z. B. den aktuellen Zustand der Visualisierung anzeigen und darüber hinaus dem Nutzer eine Möglichkeit geben, diesen zu manipulieren. Einige Elemente wurden mit Unterstützung von OpenGL ES bzw. WebGL realisiert. Dies hat den Vorteil von plattformunabhängiger Darstellung und Textur-Objekte aus dem Grafik-Speicher können direkt verwendet werden. Für andere komplexere Elemente, wie die effiziente Anzeige von Text und Menüs, wurden Bestandteile genutzt, die der Browser bzw. die iOS-Plattform zur Verfügung stellt, um das „Look-and-Feel“ der Plattform zu bewahren.

6.4.1 Klassenstruktur der OpenGL-Oberfläche

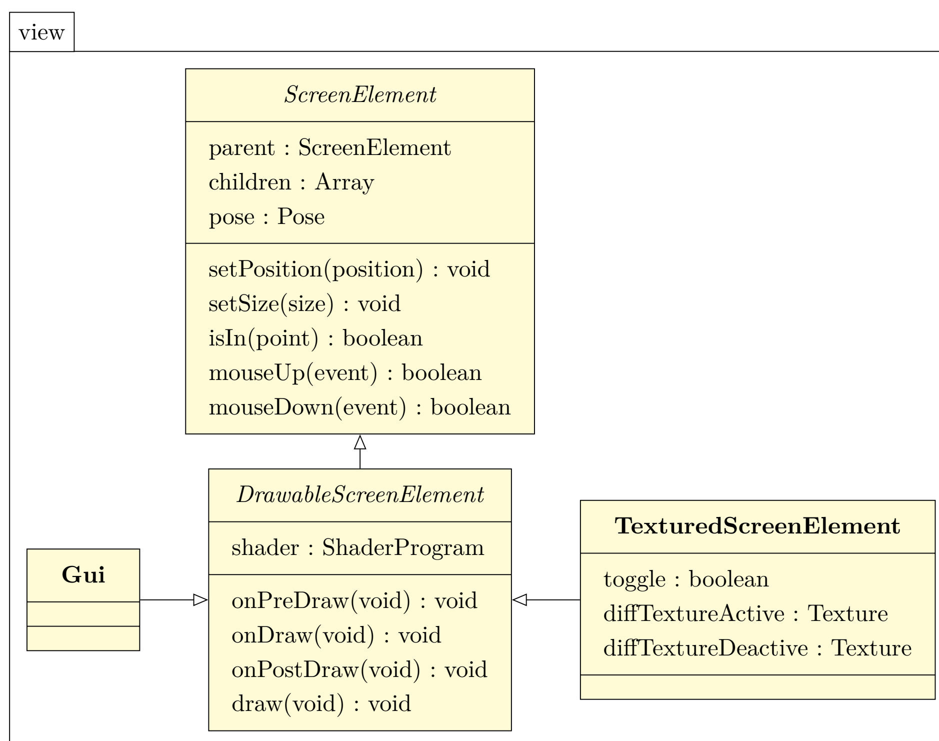


Abbildung 6.23: Klassendiagramm: „view“-Package (Ausschnitt 2)

In Abbildung 6.23 ist die Fortsetzung des Diagramms des „view“-Packages zu sehen. Mit Objekten dieser Klassen wurden Schaltflächen, Textanzeigen oder komplexere Elemente wie Slider realisiert. Nutzer-Aktionen, wie z. B. das Klicken oder Ziehen eines Elements mit der Maus werden über den *EventController* an die anderen Komponenten weitergeleitet.

ScreenElement: Die *ScreenElement*-Klasse dient als Oberklasse mit abstrakten Methoden, von der jedes dargestellte Objekt der Benutzeroberfläche erbt. Eine zentrale Information eines Objekts ist die Pose, mit der die Position und Dimensionen eines Elements gespeichert wird. Mit diesen Informationen wird ermittelt, ob ein Element per Maus oder Finger ausgewählt wurde. Um mit dieser Klasse eine Benutzeroberfläche mit mehreren Elementen aufzubauen, ist die Funktionsweise an das Composite-Pattern angelehnt. Mit

weiteren *ScreenElement*-Objekten kann eine Hierarchie angelegt werden, die einer Baum-Datenstruktur gleicht. Jedes Objekt besitzt ein Array, in dem Referenzen auf Kindknoten abgelegt werden können und jedes Element bis auf die Wurzel hat eine Referenz auf genau einen Vaterknoten. Durch diese Verknüpfung können beim Aufruf von Funktionen Veränderungen an die Kindknoten weitergegeben werden. Dadurch ist z. B. die relative Positionierung eines Elements innerhalb eines anderen möglich.

DrawableScreenElement: Von dieser abstrakten Klasse abgeleitete Klassen werden mit Hilfe eines Shaderprogramms dargestellt. Dazu können die Methoden mit dem Präfix *on* implementiert werden um nach dem Template Method-Pattern beim Zeichnen in einer festen Reihenfolge aufgerufen zu werden. Als zusätzliche Variable gibt es eine Referenz auf ein Shaderprogramm, das auch die Kindknoten verwenden können, um ihren Zeichenvorgang zu kontrollieren.

TexturedScreenElement: Ein *TexturedScreenElement* ist eine konkrete Unterklasse von *DrawableScreenElement*. Instanzen dieser Klasse haben Referenzen auf zwei Textur-Objekte. Eine Textur wird angezeigt, wenn sich das Element im aktiven Zustand befindet und die andere im inaktiven Zustand. Dies kann genutzt werden, um eine Schaltfläche zu erstellen, die nach Aktivierung die Textur verändert.

Gui: Von der Gui-Klasse existiert zur Laufzeit nur ein Objekt, das als Container für die Elemente der Benutzeroberfläche dient, sie initialisiert und als Kindknoten einträgt. Das Gui-Objekt wird mit einer Größe erstellt, die der Bildschirmauflösung entspricht. Beim justieren des Browser-Fensters oder Änderung der Orientierung des mobilen Geräts, wird dieses Objekt aktualisiert und somit die Information an die restlichen Elemente weitergegeben.

6.4.2 Plattformunabhängige Elemente

Für die Integration von plattformunabhängigen Elementen sollte vor allem auf eine Positionierung geachtet werden, die der jeweiligen Plattform entspricht. Dazu wurde die vorgestellte Klassenstruktur genutzt, die relative und absolute Größenangaben unterstützt. Nachfolgend werden die zum aktuellen Stand umgesetzten Elemente der OpenGL Benutzeroberfläche vorgestellt. In Abbildung 6.24 ist das Ergebnis der iOS-Version bzw. in Abbildung 6.25(a)-(b) der WebGL-Version zu sehen.

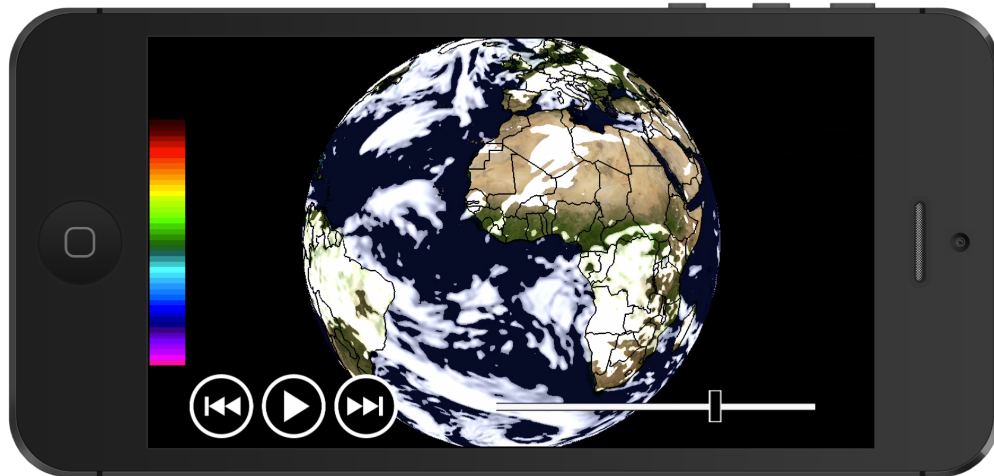
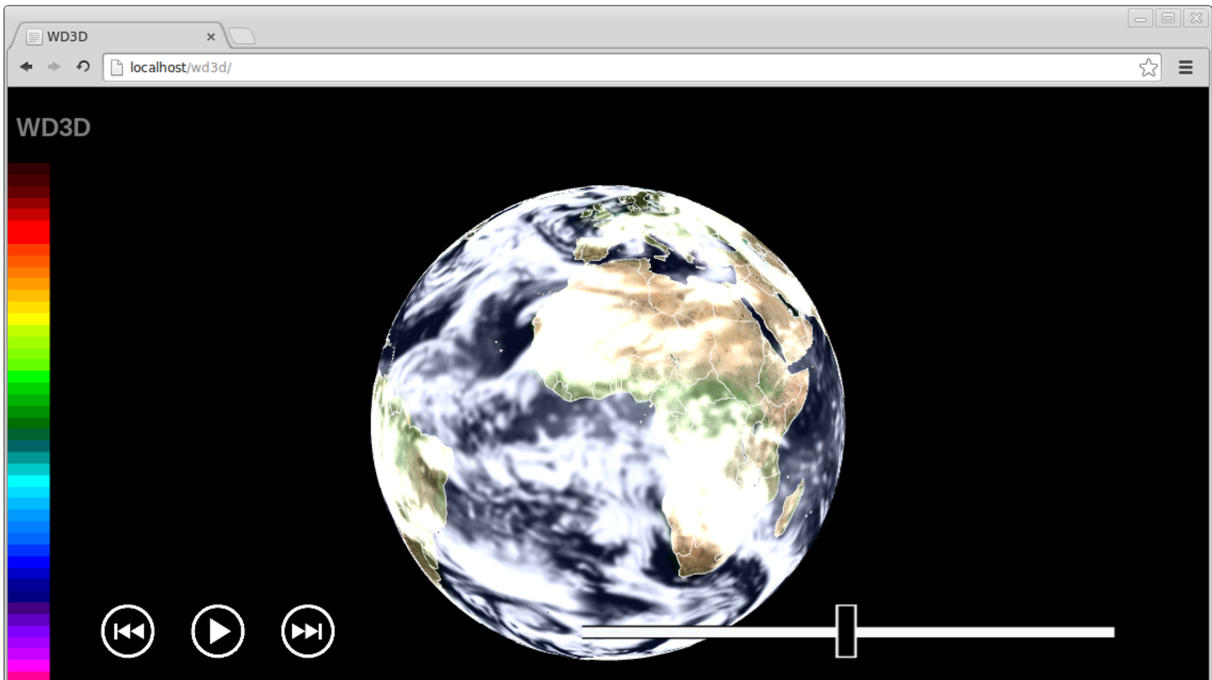


Abbildung 6.24: Umsetzung der OpenGL-Oberfläche mit OpenGL ES 2.0, links/mitte ist ein Farbspektrum zu sehen, links/unten Schaltflächen und rechts/unten ein Slider (+Erdtextur +Bewölkung)

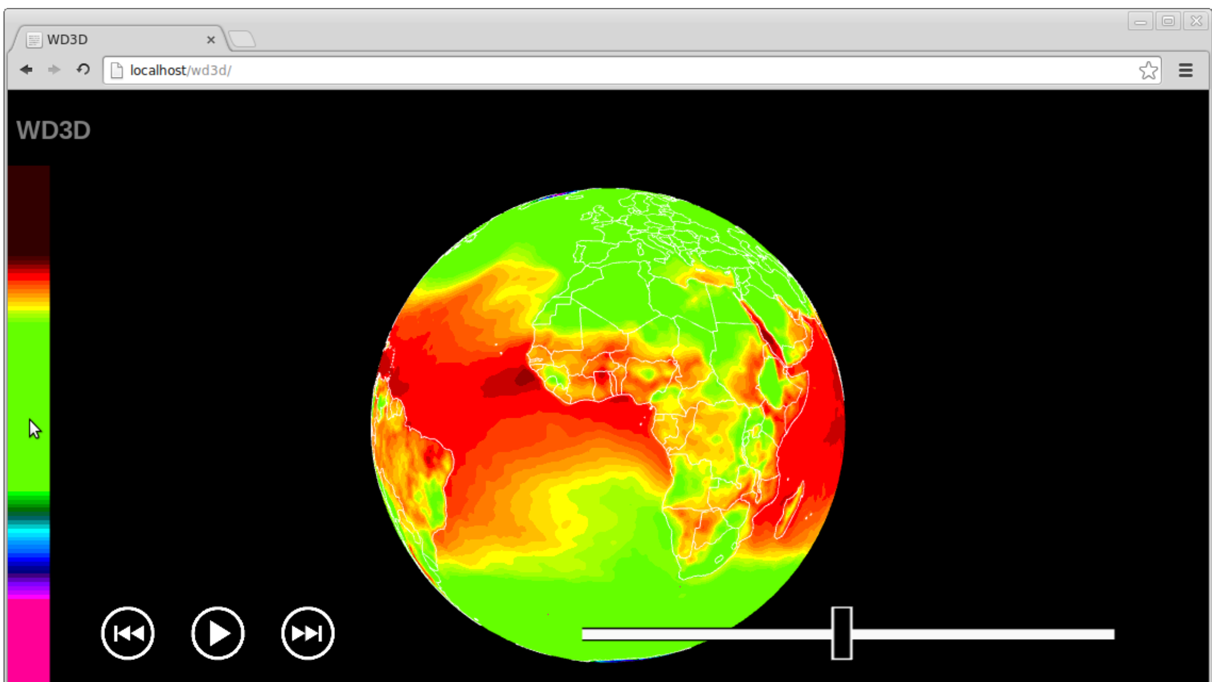
Modifizierbares Farbspektrum: Die Anzeige des Farbspektrums dient zur Zuordnung der Farben zu dem Wertebereich der Wetterdaten. Der Nutzer kann auf beiden Plattformen das Spektrum interaktiv verändern, indem er das obere und untere Intervall durch Ziehen mit der Maus bzw. mit dem Finger verändert und somit einen beliebigen Wertebereich in detaillierteren Abstufungen betrachten kann. Per Doppelklick bzw. zweimaliges Berühren wechselt das Spektrum zwischen diskreter und kontinuierlicher Darstellung. In der Web-App kann zusätzlich die Größe von einzelnen Intervallen mit dem Mausrad eingestellt werden.

Schaltfläche: Eine Schaltfläche kann durch einen Mausklick aktiviert bzw. deaktiviert werden. Beispielhaft wurden Schaltflächen umgesetzt, die das Pausieren/Starten der Animation und das Springen zu bestimmten Zeitpunkten ermöglichen.

Slider: Der Slider symbolisiert den Zeitraum der verfügbaren Wettervorhersage. Das bewegliche Element befindet sich an der aktuellen relativen Position der Animation. Dieses Element lässt sich mit der Maus bzw. dem Finger bewegen und erlaubt somit die zeitliche Navigation durch den Vorhersagezeitraum.



(a) Darstellung der Erdtextur und Bewölkung

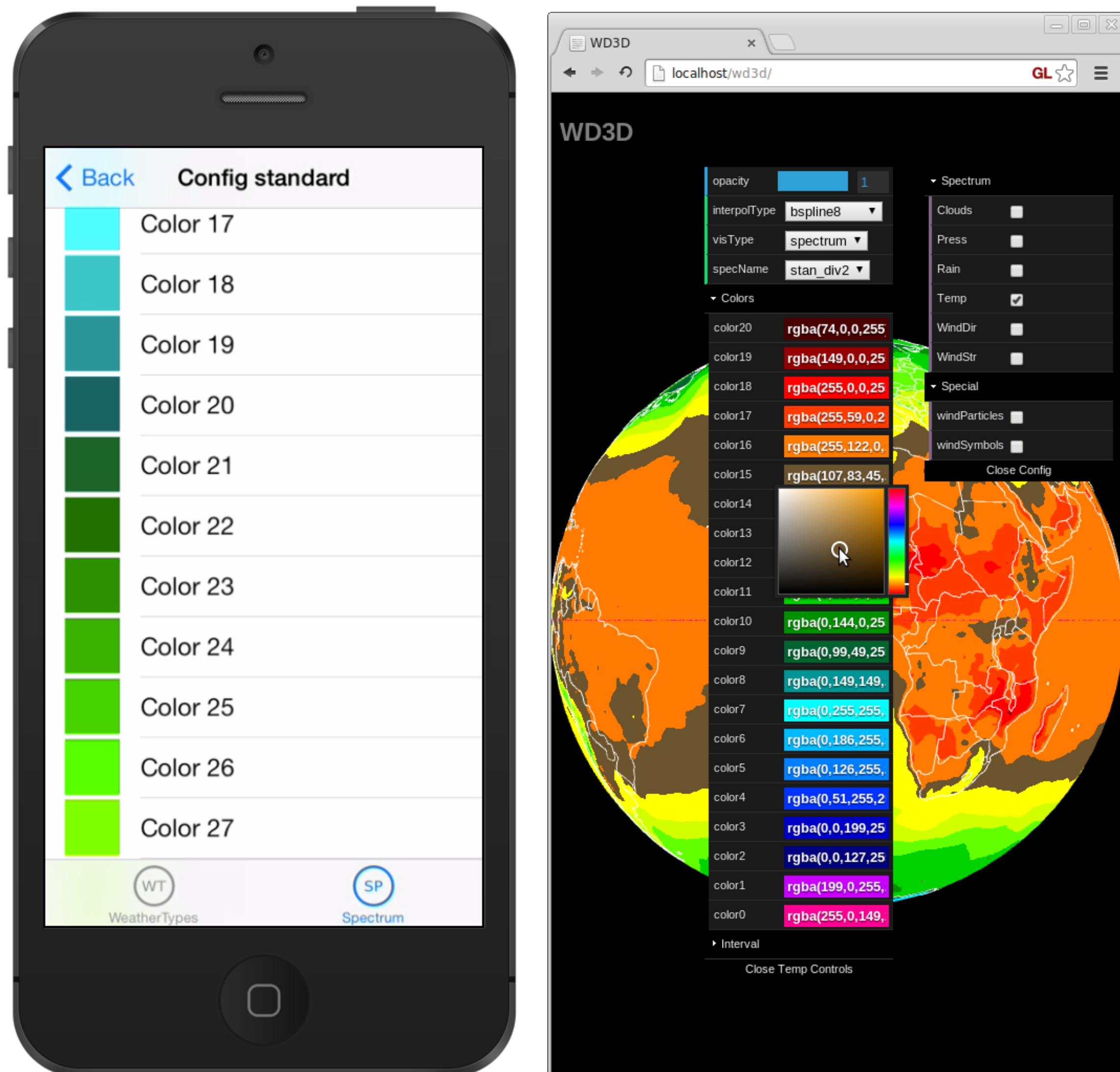


(b) Die Intervalle des Farbspektrums wurden durch Ziehen mit der Maus und dem Mausrad verändert

Abbildung 6.25: Umsetzung der OpenGL-Oberfläche mit WebGL, links/mitte ist ein Farbspektrum zu sehen, links/unten Schaltflächen und rechts/unten ein Slider

6.4.3 Plattformspezifische Elemente

Zur Darstellung von komplexeren interaktiven Elementen der Benutzeroberfläche wurden integrierte Bausteine und Frameworks der jeweiligen Zielplattformen genutzt. Die Abbildung 6.26 (a)-(b) zeigt exemplarisch, wie die Elemente auf beiden Plattformen aussehen.



(a) Detaillierte Konfiguration der Farben eines Farbspektrums auf iOS-Geräten

(b) Konfiguration der Anzeige von Wetterelementen (rechts/oben) und der einzelnen Farben eines Farbspektrums (links/mitte) in der Web-App

Abbildung 6.26: Plattformspezifische Elemente der Benutzeroberfläche der iOS- und Web-Version

Konfiguration der Darstellung von Wetterelementen: Es wurden interaktive Elemente umgesetzt, um die Parameter jedes Wetterelements anzupassen und nach ihrer Auswahl durch die Generierung eines entsprechenden Shaderprogramms (siehe Abschnitt 6.3.6) die Visualisierung zu aktualisieren. Die veränderbaren Parameter umfassen die Aktivierung/-Deaktivierung, Darstellung durch Spektrum oder Isolinie, Auswahl einer Interpolations- bzw. Approximations-Technik und Auswahl des Farbspektrums, das zur Einfärbung ge-

nutzt wird. Für den Browser wurden diese Elemente mit Hilfe des Frameworks *dat.GUI*[36] und für iOS-Geräte mit der integrierten Tabellenansicht umgesetzt.

Konfiguration eines Farbspektrums: Für jedes Farbspektrum besteht die Möglichkeit die aktuellen Farben und ihre Intervalle anzuzeigen und einzeln zu verändern. Auf den iOS-Geräten wurde der *NEOColorPicker*[37] für die Auswahl einer bestimmten Farbe verwendet, für den Browser ist diese Funktion schon in *dat.GUI* integriert. Nach Einstellung dieser Parameter wird die entsprechende Spektrums-Textur neu generiert.

6.5 Kommentar zur Umsetzung auf den Zielplattformen

Nach der Vorstellung der allgemeinen Konzeption und Umsetzung der wichtigsten Komponenten der Apps erfolgt in diesem Abschnitt eine zusammenfassende Beschreibung der Besonderheiten und Eigenschaften der Entwicklung auf zwei unterschiedlichen Zielplattformen. Dazu wird erläutert, welche Richtlinien und Erfahrungen die Arbeit mit JS und WebGL bestimmten und anschließend ObjC und die iOS-Plattform betrachtet.

6.5.1 Web-Plattform

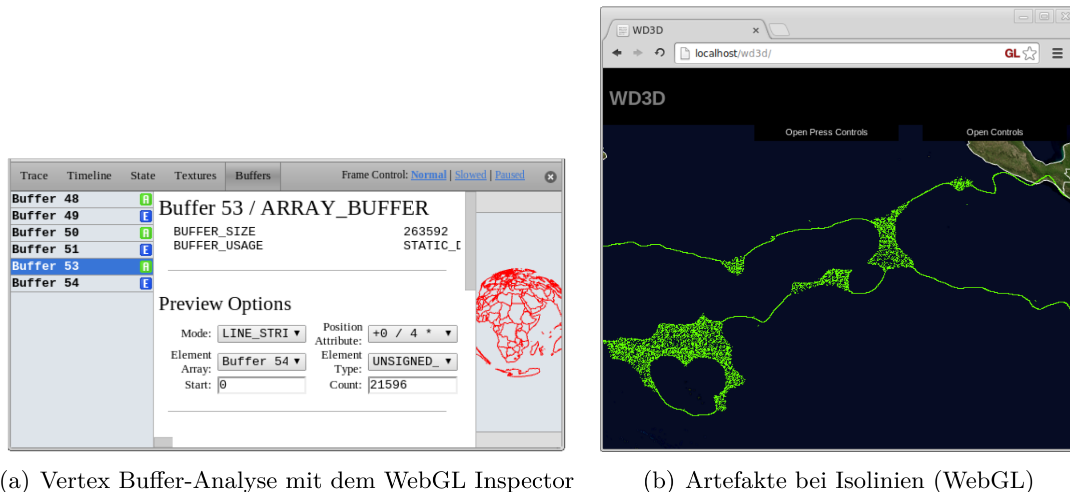
Bei der Entwicklung der Web-App wurde hauptsächlich der Browser *Google Chrome 31* genutzt, da er die beste Performanz bei der Ausführung von JS in Verbindung mit WebGL aufweist. Außerdem unterstützt er die meisten Funktionen der ECMA5-Spezifikation, von denen einige in dieser Arbeit verwendet wurden. Mit Hilfe der Developer Tools des Browsers ist es möglich, den JS-Cache auszuschalten, die Ausführungszeit und den Speicherverbrauch von einzelnen Bestandteilen zu messen und Breakpoints im Code zu setzen, um Anweisung für Anweisung Debugging durchzuführen. Diese letzte Funktion wurde aufgrund der asynchronen Datenanfragen und des Event-gesteuerten Programmablaufs im Laufe der Entwicklung unübersichtlich. Der JS-Code wurde zunächst auf Basis der Erkenntnisse und Hilfsfunktionen (z. B. für Vererbung und Anlegen einer Klassenmethode) entwickelt, die Crockford [2] zur Verfügung stellt. Außerdem wurde seine Online-Verifikation von „gutem“ JS- und JSON-Code[38] genutzt. Mit voranschreitender Entwicklung wurden allerdings aus folgenden Gründen von einigen Konventionen abgelassen und sich an den Stil der Google Closure Library angenähert:

- Die Abhängigkeiten zwischen Klassen sind klar verwaltet.
- Der Google Closure Compiler macht viele manuelle Optimierungen überflüssig.
- Debugger kann Namen von Klassenmethoden erkennen.
- Der JS-Code ist lesbarer und besser dokumentierbar.

Als weiteres neues Sprachfeature wurden Typed Arrays[39] genutzt, um die binären Daten der Wetter- und WebGL-Daten zu verwalten und zu manipulieren. Sie sind deutlich performanter als normale Array-Objekte und ermöglichen verschiedene Interpretationen wie z. B. Int8, Uint16 oder Float32 von binären Daten. CPU-Berechnungen, wie die zeitliche Interpolation der Wetterdaten, sind trotzdem sehr zeitaufwändig. Mit den neuen Möglichkeiten der ECMA5-Spezifikation wurden Funktionen dynamisch generiert und durch Closures Singleton-Objekte

[36] `https://code.google.com/p/dat-gui/`
[37] `http://kartech.github.io/colorpicker/`
[38] `http://www.jslint.com/` und `http://jsonlint.com/`
[39] `https://developer.mozilla.org/en-US/docs/Web/JavaScript/Typed_arrays`

mit privater und öffentlicher Schnittstelle realisiert. Die Entwicklung von Prototypen ging nach der Einarbeitungsphase effizient voran, was an der schwachen Typisierung, schlanken Syntax und leistungsfähigen String-Verarbeitung lag.



(a) Vertex Buffer-Analyse mit dem WebGL Inspector (b) Artefakte bei Isolinien (WebGL)

Abbildung 6.27: Besonderheiten bei der WebGL-Entwicklung

Bei der Arbeit mit WebGL stellte sich heraus, dass die Browser-API mit *Window.requestAnimationFrame(callbackFunction)*[40] eine deutlich stabilere Animation ermöglicht als die eingebauten JS-Timer. Diese Funktion ruft die Hauptschleife auf, wenn der Browser für einen neuen Zeichenvorgang bereit ist. Dies führt zu einer Animation mit maximal 60 FPS, die auch pausiert wird, wenn das Browser-Fenster nicht sichtbar ist und somit Rechenzeit und Energie spart. Die Chrome Erweiterung WebGL Inspector[41] (Abbildung 6.27 (a)) ist dabei sehr nützlich, um den OpenGL-Status und -Objekte auszulesen. Weitere Erfahrungen aus der Arbeit mit WebGL:

- Die Methode *glReadPixels(...)* zum Auslesen des Framebuffer-Inhalts ist sehr zeitaufwändig und zur Zeit nur mit allen vier Farbkanälen (GL_RGBA) kompatibel.
- Bei manchen Visualisierungstechniken treten vereinzelt auf Systemen Artefakte auf. (Abbildung 6.27 (b))
- Das Erstellen und Verändern der Dimensionen eines Framebuffer-Objekts ist unkompliziert.
- Die Veränderung der Fenster-Größe des Browsers muss durch Aktualisierung des Viewports und der Kamera behandelt werden, um auch den Vollbild-Modus zu unterstützen.

Abschließend lässt sich sagen, dass WebGL mit zukünftiger Weiterentwicklung ein guter Standard für die Anzeige von 3D-Inhalten und Berechnungen auf der GPU im Internet ist. Er bietet vor allem wegen der Plattform- und Plugin-Unabhängigkeit Vorteile genüber anderen Darstellungstechniken.

6.5.2 iOS-Plattform

Die iOS-App wurde auf dem Betriebssystem *Mac OS X 10.9.1* mit *XCode 5.0.2* entwickelt. Es wurde mit Geräten und dem Simulator der Betriebssysteme *iOS 5 - iOS 7* gearbeitet. Die Pro-

[40] `https://developer.mozilla.org/en-US/docs/Web/API/window.requestAnimationFrame`
[41] `http://benvanik.github.io/WebGL-Inspector/`

grammstruktur ließ sich gut umsetzen, da Datenkapselung und statische Methoden vorgesehen sind und die Funktionen von ObjC 2.0 die Portierung von JS-Code erleichtern. So entsprechen Blöcke den first-class-functions und lassen die Übergabe von Funktionen an Variablen und Callback-Funktionen zu. Eine Erleichterung bei der Programmierung stellen neben der guten Dokumentation auch die neu eingeführten Literale[42] dar, welche die benötigte Syntax bei der Arbeit mit Arrays und Dictionaries verringern. Berechnungen auf der CPU stellten sich wie bei JS als sehr zeitaufwändig heraus obwohl C-Arrays statt den integrierten Array-Objekten genutzt wurden.

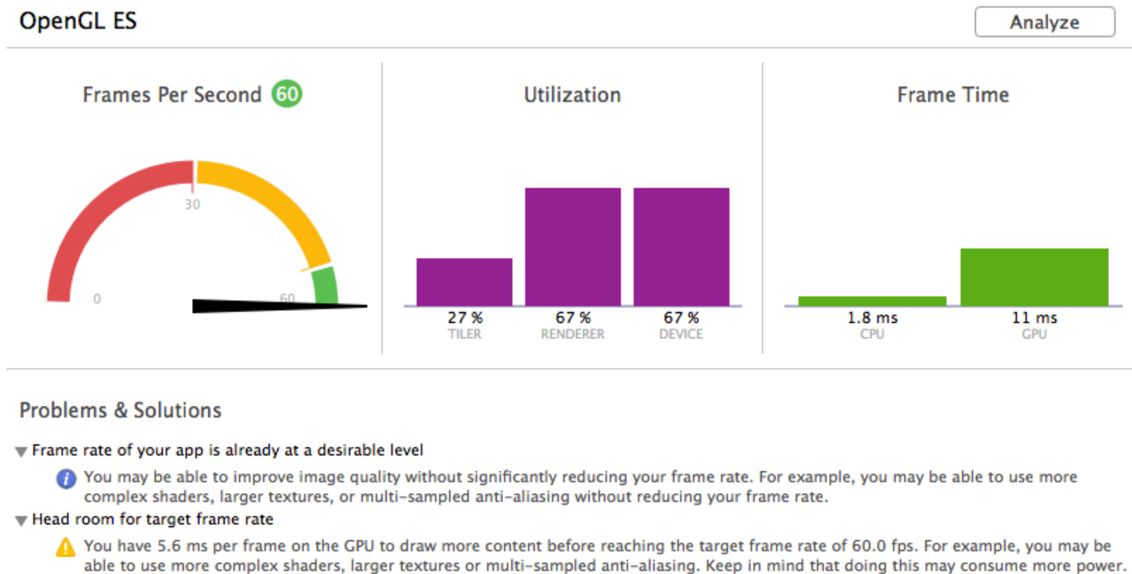


Abbildung 6.28: Analyse der OpenGL ES-Performanz (iOS)

Die im GLKit integrierten Datenstrukturen für Vektoren-Berechnung unterstützten die Arbeit mit OpenGL ES 2.0, da die C-Schnittstelle genutzt wurde und somit Vektoren und Matrizen direkt an OpenGL übergeben werden konnten. Der iOS-Simulator eignete sich dabei für die visuelle Ergebnisüberprufung. Für die genauere Analyse des Programms ist allerdings Hardware und somit eine kostenpflichtige Entwicklerlizenz notwendig. In Abbildung 6.28 ist das Ergebnis der integrierten OpenGL ES-Analyse zu sehen, die anzeigt wie sich die Rechenzeit für einen Frame auf CPU und GPU aufteilt und daraus Vorschläge zur Optimierung generiert. Daraus wurde unter anderem ersichtlich, dass Texturzugriffe mit berechneten Texturkoordinaten einen hohen Aufwand verursachen. Wie bei WebGL kann auch der Inhalt von OpenGL-Objekten analysiert werden. Weitere Bemerkungen zur Arbeit mit OpenGL ES 2.0:

- Die Methode *glReadPixels(. . .)* ist im Vergleich zu JS und WebGL nicht sehr aufwändig.
- Der Datentyp GL_FLOAT wird nur bei Texturen unterstützt, die nicht mit einem Framebuffer verknüpft sind. Stattdessen kann der Datentyp GL_HALF_FLOAT zur Arbeit mit einem Framebuffer verwendet werden.
- Deutlich weniger Artefakte im Gegensatz zu WebGL auf den in Abschnitt 7.1 vorgestellten Systemen.
- Die Pixeldichte des Geräts muss bei Einstellung der Renderbuffer und Viewports beachtet werden. Eine höhrere Pixeldichte sorgt für höhere optische Qualität bei den Interpolations- und Approximations-Techniken, allerdings müssen die Dimensionen von GL_LINES und

[42] http://clang.llvm.org/docs/ObjectiveCLiterals.html

GL_POINTS angepasst werden.

Einige der fehlenden Funktionen wie die volle Unterstützung von float-Texturen und Transform Feedback sind seit der Veröffentlichung des neuen iPhone 5S (September 2013) mit OpenGL ES 3.0 Integration inzwischen verfügbar. Diese Möglichkeiten würden vor allem beim Partikelsystem Optimierungen erlauben.

Kapitel 7

Tests und Evaluation

Damit die entwickelten Apps mit aktuell verfügbaren Angeboten der interaktiven Wettervisualisierung in Relation gesetzt werden können, wird in folgenden Auswertungen herausgefunden, wie sie sich bei normaler Nutzung verhalten und an welchen Stellen Potential für Optimierungen besteht. Dazu wurden repräsentative Testsysteme ausgewählt und eine Analyse der Laufzeit und der benötigten Datenmenge durchgeführt.

7.1 Testsysteme

Tabelle 7.1: Technische Spezifikationen der Testsysteme

Name	Macbook Pro (2011)	iPhone 5 (2012)	iPad 3 (2012)
Prozessor	Core i7 @ 2,2 GHz	A6	A5X
Grafikkarte	Rad. HD 6750M 600 MHz	PVR 543MP3 266 MHz	PVR 543MP4 200 MHz
Auflösung	1680×1050 @ 112 ppi	1136×640 @ 326 ppi	2048×1536 @ 264 ppi
Plattform	Google Chrome	iOS 7.04	iOS 7.04

7.2 Performanz

Auf den aufgeführten Systemen wurden Laufzeitanalysen durchgeführt, um herauszufinden, in welcher Qualität die entwickelten Techniken auf aktuellen Systemen nutzbar sind und ob sich das Verhalten einer Plattform von der anderen unterscheiden. Dazu wurden zum einen die Darstellungstechniken und zum anderen die vorgestellten Interpolations- und Approximations-Techniken variiert. Es wurden 1-4 Farbspektren kombiniert dargestellt und 1-2 Wetterelemente, die durch Isolinien dargestellt wurden. Zunächst dient die Interpolationstechnik Nearest Neighbor als Referenz, weil sie in die Hardware integriert ist. Darauf folgt Bilinear, die 4 Texturzugriffe (TZ) benötigt, Bicubic mit 16 TZ und abschließend die aufwändige BSpline8-Approximation mit 64 TZ. Zu der Isoliniendarstellung ist zu bemerken, dass sie im Vergleich zum Farbspektrum den 4-fachen Interpolationsaufwand bedeutet.

Bei diesen Tests wurde die Anzahl der Bilder pro Sekunde als Kennzahl gewählt, um eine ansprechende und interaktive Benutzung der Apps zu repräsentieren. Ein höherer Wert gilt als besser. Bei den Tests war jeweils die native Auflösung des Bildschirms mit dem 3D-Globus ausgefüllt und der gleiche Ausschnitt der Erdkugel zu sehen. Es wurden Wetterdaten des Datentyps GL_UNSIGNED_BYTE mit einer räumlichen Auflösung von 720 × 360 Datenpunkten verwendet. Bei jedem Durchlauf wurde eine zeitliche Animation gestartet, die über 10 Zeitschritte in einer realen Wettervorhersage verlief. Die zeitliche Auflösung zwischen zwei Zeitschritten betrug den Faktor 100. Die folgenden Ergebnisse wurden über mehrere Durchläufe gemittelt.

7.2.1 iOS-Plattform

Tabelle 7.2: Performanz-Analyse iPhone5

Darstellung	Nearest N.	Bilinear (4 TZ)	Bicubic (16 TZ)	BSpline8 (64 TZ)
1 Spektrum	59.8 FPS	59.6 FPS (±0.0%)	8.8 FPS (−85.3%)	5.8 FPS (−90.0%)
2 Spektren	59.6 FPS	58.6 FPS (−1.7%)	4.5 FPS (−92.4%)	2.2 FPS (−96.3%)
4 Spektren	58.2 FPS	40.9 FPS (−29.7%)	2.2 FPS (−96.2%)	0.3 FPS (−100.0%)
1 Isolinie	56.9 FPS	27.1 FPS (−52.4%)	1.8 FPS (−96.8%)	<0.1 FPS (−100.0%)
2 Isolinien	31.3 FPS	13.3 FPS (−57.5%)	0.9 FPS (−97.1%)	<0.1 FPS (−100.0%)

Die Ergebnisse der Testdurchläufe auf dem iPhone 5 sind in Tabelle 7.2 aufgeführt. Dabei ist zu beachten, dass auf iOS-Plattformen die Darstellung mit OpenGL ES auf 60 FPS begrenzt ist, die auch für eine interaktive Echtzeit-App ausreichend sind. In der Tabelle ist zu jeden Wert die prozentuale Veränderung gegenüber der Referenz der Nearest Neigbor-Interpolation aufgetragen. Aus den Ergebnissen lassen sich zwei Muster ableiten. Es ist zu erkennen, dass die Werte mit komplexer werdender Darstellung (in der Tabelle absteigend) etwas schlechter werden. Weiterhin bedeutet der Schritt von der bilinearen zur bikubischen Interpolation den größten Performanzverlust im Gegensatz zu den anderen. Bei einem Farbspektrum führen die 4-fachen TZ zu eine Verschlechterung um 85%. Oberhalb dieser Schwelle verhalten sich Verfahren wie 4 Spektren/Bilin, 1 Isolinie/Bilin und 2 Isolinien/NN mit gleicher Anzahl von TZ ähnlich und liegen bei ca 30 - 40 FPS.

Tabelle 7.3: Performanz-Analyse iPad3

Darstellung	Nearest N.	Bilinear (4 TZ)	Bicubic (16 TZ)	BSpline8 (64 TZ)
1 Spektrum	58.3 FPS	39.4 FPS (−32.4%)	2.5 FPS (−95.7%)	1.7 FPS (−97.1%)
2 Spektren	56.8 FPS	20.9 FPS (−63.2%)	1.4 FPS (−97.5%)	0.6 FPS (−98.9%)
4 Spektren	31.9 FPS	12.4 FPS (−61.1%)	0.8 FPS (−97.5%)	<0.1 FPS (−100.0%)
1 Isolinie	17.7 FPS	7.7 FPS (−56.5%)	0.6 FPS (−96.6%)	<0.1 FPS (−100.0%)

Auf dem iPad3 wurden die gleichen Tests mit anderer nativer Auflösung durchgeführt und die Ergebnisse in Tabelle 7.3 aufgetragen. Insgesamt sind die Werte etwas schlechter als beim iPhone5 ausgefallen. Auffällig ist der Performanzverlust bei einem Spektrum von Nearest Neighbor

zu Bilinear um 32% und bei 2 Spektren um 63%. Der große Abfall der Werte bei dem Schritt von Bilinear zu Bikubisch ist wieder zu beobachten.

7.2.2 WebGL-Plattform

Tabelle 7.4: Performanz-Analyse WebGL, Macbook Pro

Darstellung	Nearest N.	Bilinear (4 TZ)	Bicubic (16 TZ)	BSpline8 (64 TZ)
1 Spektrum	56.9 FPS	57.2 FPS (±0.0%)	57.2 FPS (±0.0%)	57.2 FPS (±0.0%)
2 Spektren	57.2 FPS	57.1 FPS (−0.1%)	56.2 FPS (−1.7%)	45.8 FPS (−19.9%)
4 Spektren	57.1 FPS	56.8 FPS (−0.5%)	32.6 FPS (−42.9%)	38.4 FPS (−32.7%)
1 Isolinie	57.3 FPS	57.3 FPS (±0.0%)	32.2 FPS (−43.8%)	16.2 FPS (−71.7%)
2 Isolinien	56.9 FPS	57.2 FPS (±0.0%)	5.9 FPS (−89.6%)	<0.1 FPS (−100.0%)

Die Messungen der Web-App wurden auf dem Macbook Pro mit dem Google Chrome Browser durchgeführt und Tabelle 7.4 stellt ihre Ergebnisse dar. Die Darstellung von WebGL ist ebenfalls auf 60 FPS begrenzt, weshalb sich bei der Darstellung von einem Spektrum keine Unterschiede erkennen lassen, da jeder Eintrag an diesen Höchstwert heranreicht. Erst bei 2 Spektren und der aufwändigen BSpline8-Approximation lässt sich eine Verschlechterung von ca. 20% ausmachen. Bei 4 Spektren hat sich die Performanz beim Schritt von bikubischer Interpolation zu BSpline8 um 10% verbessert, was vermutlich einen Messfehler darstellt, da bei einer Isolinie ein Performanzverlust von 30% auszumachen ist. Der Schwelleneffekt beim Übergang von bilinearer auf bikubische Interpolation fällt geringer als auf den iOS-Geräten aus. Bemerkenswert sind auch die nahezu exakt gleichen Werte bei 4 Spektren/Bicubic und 1 Isolinie/Bicubic, die auch der gleichen Anzahl von TZ entsprechen.

7.2.3 Fazit zur Performanzanalyse

Bei den iOS-Systemen lässt sich die allgemeine Verringerung der FPS-Werte vom iPhone5 zum iPad3 durch die Verwendung von ähnlicher Hardware bei der deutlich größeren Auflösung des iPads erklären. Die Leistungsschwelle, die beim Wechsel von bilinearer auf bikubische Interpolation ausgemacht wurde, lässt erkennen, wo aktuell für die Grafikprozessoren der iOS-Systeme die Grenzen bei der Anzahl von TZ mit im FS berechneter Texturkoordinate liegen. Schon bei der Entwicklung hat die OpenGL ES-Analyse (siehe Abschnitt 6.5.2) auf dieses Performanz-Problem hingewiesen. Trotzdem ist eine interaktive Nutzung der App in guter Qualität auf beiden Geräten möglich ist.

Die Messungen der WebGL-Version auf dem mobilen Macbook Pro zeigten ebenfalls, dass eine höhrere Anzahl von TZ zu einer schlechteren Performanz führen. Allerdings fällt die Veränderung von bilinearer zu bikubischer Interpolation geringer aus. Dies deutet auf die fortschrittlichere Entwicklung des Grafikprozessors hin, der in mobilen Rechnern verbaut wird. Auch hier lässt sich sagen, dass die App in guter Qualität auf einem mobilen Gerät lauffähig ist.

7.3 Datenübertragung

Vor allem auf mobilen Geräten ist die Menge der benötigten Daten ein Faktor, der die Nutzbarkeit eines Programms beeinflusst. Datentarife für die mobile Internetnutzung sind oft gedrosselt oder bei einer schnellen Übertragung auf ein bestimmtes Volumen begrenzt. In dem folgenden Test wird gemessen, wie sehr die Bestimmung von sichtbaren Datenteilmengen[43] die Datenmenge verringern kann. Dazu wurden mehrere Testläufe durchgeführt, bei denen wieder eine Animation über 10 Zeitschritte lief und nur ein Wetterelement mit einer Auflösung von 720×360 Datenpunkten angezeigt wurde. Es ist zu beachten, dass von dem Programm $n + 1$ Zeitschritte geladen werden, um eine fortlaufende Animation zu gewährleisten. Da die Daten auf dem Server zu int-Werte konvertiert wurden, wird jeder Datenpunkt durch ein Byte repräsentiert. Bei den einzelnen Testläufen wurden jeweils die Faktoren für die Größe der Rechtecke verändert um die Auflösung des Sichtbarkeitstests einzustellen. Bei jedem Testlauf war der gleiche Blickwinkel senkrecht zur Achse des Globus eingestellt und blieb unverändert.

Tabelle 7.5: Messung der übertragenen Daten bei veränderter Auflösung des Sichtbarkeitstests

Rechtecksdimensionen	Sichtbar	Datenmenge (Base 64)	Datenmenge (dekodiert)
100% × 100% (1 Rechteck)	1	3801600 Byte	2851200 Byte
50% × 50% (4 Rechtecke)	4 (100%)	3801600 Byte	2851200 Byte (±0%)
30% × 30% (16 Rechtecke)	9 (56%)	3079296 Byte	2309472 Byte (−19%)
10% × 10% (100 Rechtecke)	44 (44%)	1672704 Byte	1254528 Byte (−56%)
7% × 7% (195 Rechtecke)	78 (40%)	1431144 Byte	1072500 Byte (−62,4%)

Die Ergebnisse der Messungen der übertragenen Daten sind in der Tabelle 7.5 eingetragen. In der ersten Spalte sind die prozentualen Dimensionen eingetragen, die für die Generierung der Rechtecke angegeben wurden und die zweite Spalte gibt an, wie viele Rechtecke für den eingestellten Blickwinkel auf den Globus als sichtbar identifiziert wurden. Die dritte Spalte gibt an, wie groß die Datenmenge ist, die über das Internet mit Base64-Kodierung übertragen wurde und die vierte Spalte die Anzahl der Bytes nach der Dekodierung. Daraus lässt sich berechnen, dass die Kodierung der Daten einen Zuwachs von 25% ausmacht. Ab der zweiten Zeile ist jeweils angegeben, wie hoch der prozentuale Anteil der sichtbaren Rechtecke ist. Bei den Daten ist die prozentuale Veränderung des Wertes zur ersten Zeile angegeben, die als Referenz dient, weil die vollständige Anzahl von Daten $720 * 360(\text{Auflösung}) * 11(\text{Zeitschritte}) * 1Byte(\text{int}) = 2851200\ Byte$ geladen wurde. Ab einer Rechtecksgröße von $30\% \times 30\%$ werden fast 50% der Rechtecke als nicht sichtbar identifiziert, die Ersparnis bei den Daten liegt aber erst bei 19%, was durch die Größe der Rechtecke zu erklären ist. Bei Dimensionen von $7\% \times 7\%$ liegt die Ersparnis dann bei ca. 62%. Obwohl mehr als 50% der Daten nicht aktualisiert werden, ist eine gute Nutzung der Applikation möglich, da dies Randbereiche betrifft, die wegen der perspektivischen Darstellung der Kugel nicht sichtbar sind.

Die Analyse der Messungen zeigt, dass der entwickelte Sichtbarkeitstest für Teilmengen der Wetterdaten bei hochaufgelöster Einteilung der Erdoberfläche durch Rechtecke einen großen Beitrag zur Einsparung von zu übertragenden Daten leistet. In Kombination mit verschiedenen Zoomstufen lässt sich die Datenmenge dann bis zu einem gewissen Grad optimal an die Bedürfnisse

[43] Siehe Abschnitt 6.2.3

des Nutzers anpassen, indem bei unbeschränkter Übertragung die beste und andernfalls eine geringere Qualität der Daten ausgewählt wird.

Kapitel 8

Fazit und Ausblick

Diese Arbeit entstand in einem Umfeld, in dem die Entwicklung der technischen Möglichkeiten auf mobilen Geräten in kurzer Zeit mit großen Schritten voranschreitet. Entwickler können seit OpenGL ES 2.0 auf aktuellen Geräten der iOS-Familie, wie dem iPhone und iPad, eine Datenvisualisierung mit Hilfe von programmierbarer Grafik-Hardware realisieren. Mit ähnlichen Spezifikationen wurde WebGL entwickelt, um solche Darstellungen im Browser zu integrieren und somit auf Desktop- wie mobilen Notebook-Systemen zur Verfügung zu stellen. Einen besonderen Nutzen auf diesen Geräten bringt die Information über die aktuelle Wettervorhersage, wenn der Nutzer unterwegs ist. Mit diesen Voraussetzungen ließen sich für das Anwendungsgebiet der animierten Anzeige von globalen Wettervorhersagedaten neue Erkenntnisse sammeln.

Es wurde angenommen, dass die Visualisierung von globalen Wetterdaten einen Mehrwert bietet, weil die einzelnen Wetterelemente sich gegenseitig beeinflussen und einer Dynamik unterliegen, die einer lokalen Vorhersage nicht zu entnehmen ist. Als Visualisierungstechniken der einzelnen Elemente wurden das Farbspektrum, Isolinien, Partikel und spezielle Symbole, wie Windfahnen, vorgestellt. Jede Technik hat unterschiedliche Eigenschaften und kann teilweise mit anderen kombiniert werden. Eine hardwarebeschleunigte Darstellung der Ursprungsdaten kann gegenüber vorberechneten Bildern den Vorteil von interaktiver Einstellung verschiedener Parameter an die Bedürfnisse eines Nutzers bieten.

Die Analyse der aktuell verfügbaren Apps auf den Zielplattformen zeigte, dass für iOS-Systeme zwar Angebote existieren, die eine ansprechende Darstellung, aber wenig Möglichkeiten zur Personalisierung der Anzeige bieten. Mit WebGL wurde bis zum aktuellen Stand trotz der leistungsfähigen Technologie keine App entwickelt, die den Ansprüchen dieser Arbeit gerecht wird. Auf dieser Grundlage konnten Anforderungen an ein Programm entwickelt werden, das vor allem im WebGL-Umfeld eine Innovation zum Stand der Technik darstellt.

Bei der Umsetzung wurden Konzepte entwickelt, um die Anwendung an die mobilen Zielplattformen anzupassen. Es hat sich unter anderem gezeigt, dass durch Nutzung des Grafikprozessors die CPU der mobilen Geräte entlastet werden konnte, um eine ansprechende interaktive Nutzung zu realisieren. Zusätzlich wurde eine Technik entwickelt, um die Menge der benötigten Daten zu ermitteln und somit die Übertragungsmenge über das mobile Internet zu verringern. Die vorgestellten Visualisierungstechniken wurden auf unterschiedliche Art umgesetzt und vor allem bei den Partikeln und speziellen Symbolen machte sich das Fehlen moderner Techniken von Desktop-Grafikkarten, wie Transform Feedback und dem Geometry Shader, in den Spezifikationen von WebGL und OpenGL ES 2.0 bemerkbar. Mit ihrer Hilfe wäre eine noch effizientere

Darstellung der Daten möglich.

Für die Benutzeroberfläche der Apps konnten exemplarisch Elemente umgesetzt werden, die eine Navigation durch die Wettervorhersage und Anpassung der Visualisierung ermöglichen. Durch die Veränderbarkeit der Darstellungsqualität von einzelnen Wetterelementen und Auswahl der Farben eines Farbspektrums konnte auch gegenüber den aktuell verfügbaren iOS-Apps eine Innovation erzielt werden, da diese bereits ausgereifte Visualisierungstechniken bieten.

In der Evaluation konnte festgestellt werden, dass die im Rahmen dieser Arbeit entstandenen Apps auf verschiedenen Plattformen in guter Qualität verwendet werden können. Dazu wurde die Qualität der Animation und die Menge der übertragenen Daten gemessen.

Insgesamt ist es so gelungen, eine interaktive und vielseitige Visualisierung einer Wettervorhersage mit Hilfe von OpenGL ES 2.0 und WebGL zu entwickeln. Zusätzlich bietet das Programm sinnvolle Neuerungen gegenüber der aktuell verfügbaren Angebote. Der Entwicklungsprozess, der sich von der Analyse des aktuellen Stands der Technik über die allgemeine Konzeption bis zur Implementierung auf den Zielplattformen erstreckt, ist in dieser Arbeit dokumentiert.

Für die weitere Entwicklung der Apps bietet sich die Untersuchung der Möglichkeiten von OpenGL ES 3.0 und der kommenden Weiterentwicklung von WebGL an. Diese können zur Optimierung bzw. Weiterentwicklung der Visualisierungstechniken genutzt werden. Ein möglicher Ansatzpunkt könnte dabei die Nutzung von Wetterdaten in float-Genauigkeit statt der verwendeten int-Werte sein. Auch das Partikelsystem und die Anzeige von speziellen Symbolen können von den neuen Funktionen profitieren. Die Darstellungsqualität der Isolinien ließe sich durch effizientere Nutzung der CPU noch weiter verbessern.

Außerdem konnten im Rahmen dieser Arbeit nicht alle Funktionen integriert werden, welche die Nutzung von mobilen Geräten interessant machen. Neben der Standortbestimmung, die auf iOS-Geräten wie auch mit HTML5 durchgeführt werden kann, würde die Nutzung von Datenarchivierung und parallelen Verarbeitungsprozessen neue Verfahren und Auswertungen ermöglichen. Das Programm könnte zusätzlich als native App für Android- und Windows-Geräte umgesetzt werden um mehr Nutzer zu erreichen und spezielle Funktionen der Systeme einzusetzen. Ein Ziel bei der Entwicklung eines Programms für verschiedene Plattformen sollte die zentrale Konfiguration von dynamischen Komponenten wie z. B. Interpolationstechniken und Nutzer-Profilen für die Visualisierung sein. Die Server-Komponente dieser Arbeit hat durch die Verteilung der gleichen Spektrumsdaten an beide Apps dieses Vorhaben schon angedeutet.

Mit Blick auf die Erkenntnisse dieser Arbeit lässt sich abschließend sagen, dass das Interesse an innovativen Wetterdarstellungen auf mobilen Geräten sowohl aus Sicht der Nutzer als auch der Entwickler von hardwarebeschleunigten Visualisierungen in Zukunft noch anhalten wird.

Anhang A

Screenshots

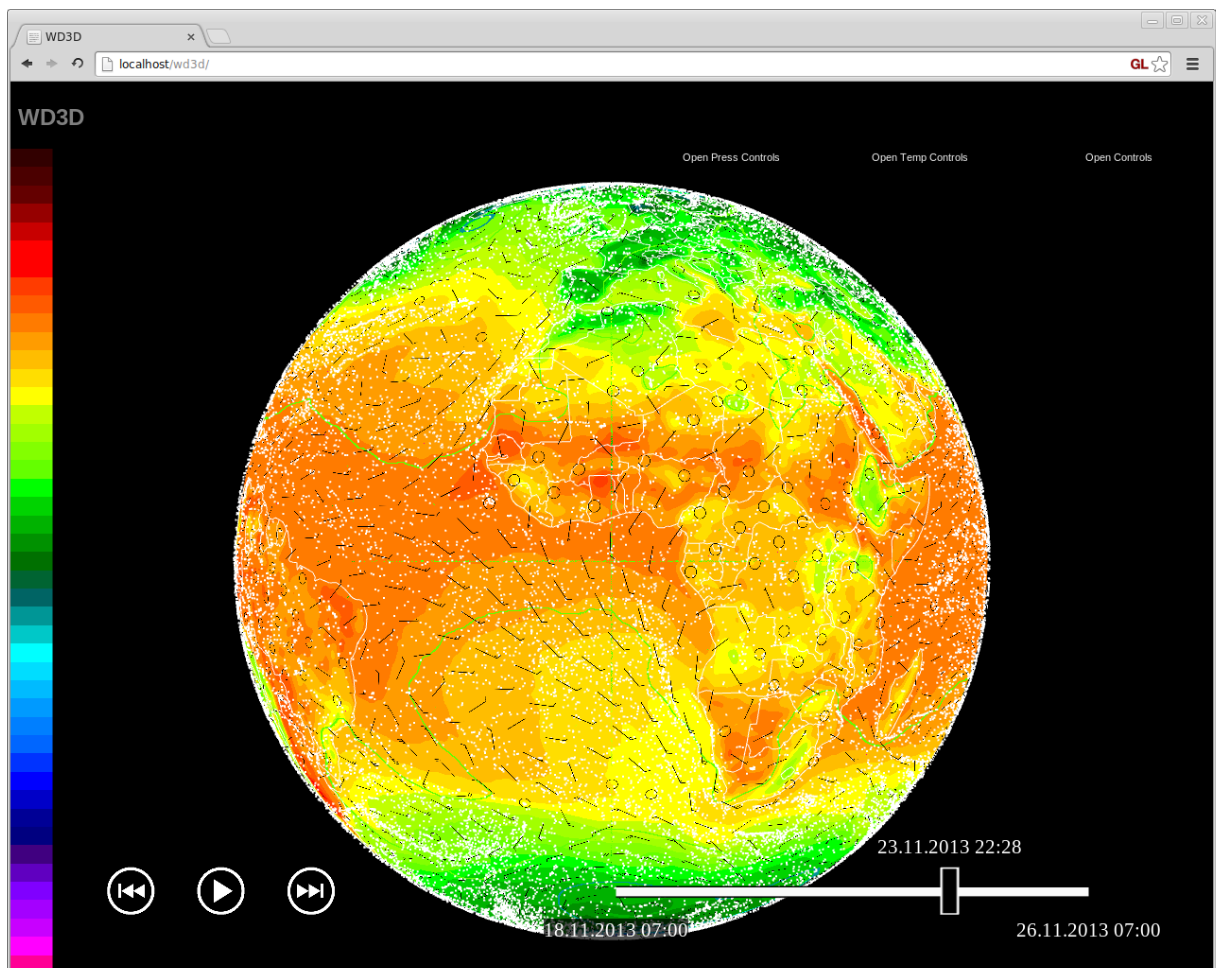


Abbildung A.1: Darstellung der Temperatur anhand eines Farbspektrums, des Luftdrucks durch Isolinien (grün), der Windrichtung und -stärke durch Partikel und Windfahnen, Küstenlinien und der Benutzeroberfläche (WebGL)

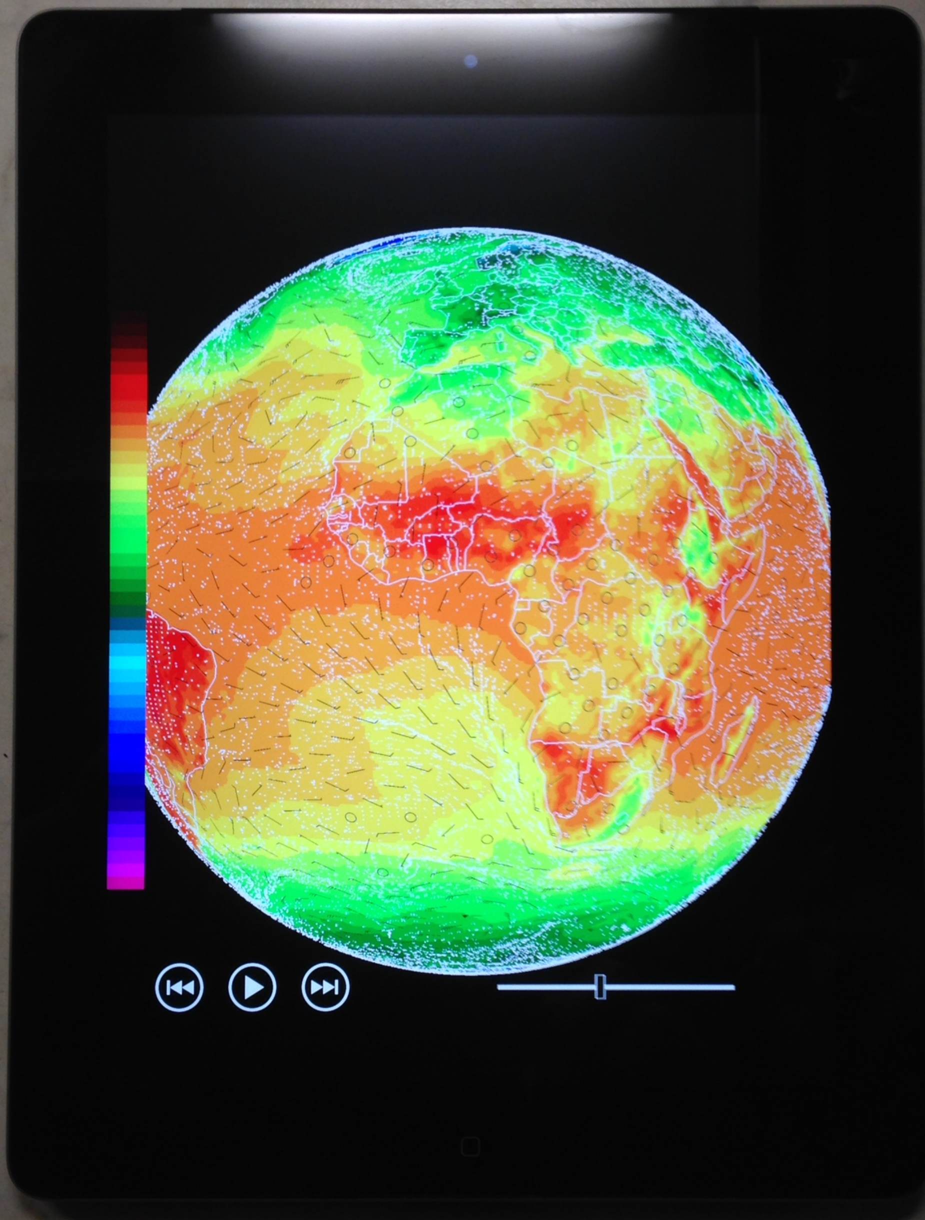


Abbildung A.2: Darstellung der Temperatur anhand eines Farbspektrums, der Windrichtung und -stärke durch Partikel und Windfahnen, Küstenlinien und der Benutzeroberfläche (iPad3 Foto)

Anhang B

Inhalt der CD und URL

Im Anhang befindet sich eine CD mit folgendem Inhalt:

Tabelle B.1: Inhalt der beigelegten CD

Unterordner	Inhalt
code	die Projekte der im Rahmen dieser Arbeit entwickelten Applikationen GribLoader, wd3d-ios, wd3d-web
data	Beispiel-Wetterdaten im GRIB-Format
docs	die JSDOC-Dokumentation von wd3d-web im HTML-Format
lib	die java-Bibliotheken und -Projekte, die GribLoader benötigt
sql	Beispiel-Wetterdaten als SQL-Befehle, um eine Datenbank zu füllen
thesis	diese Arbeit im pdf-Format
README.txt	eine detailliertere Beschreibung des Inhalts

Die WebGL-Version der Applikation ist zum Abgabezeitpunkt unter `wd3d.g-connection.org` zu erreichen.

C Abbildungsverzeichnis

1.1 Visualisierung von globalen Wetterdaten (Luftdruck und Wind) nach [13] 3
1.2 Ausschnitte der M3G-Applikation aus [3] (Interface, 3D-Globus, Temperatur, Temperatur) 3

2.1 Wetter-App in iOS 7 7
2.2 Auswahloptionen des Wettervorhersagedaten-WebService der NOAA, es sind Luftdruck, Wolkendichte und Temperatur in zwei Metern über der Meeresoberfläche ausgewählt 8
2.3 Wettervorhersage der ECMWF für Europa mit Visualisierung der Temperatur als Farbspektrum und des Luftdrucks mit Hilfe von Isolinien 9
2.4 Wettervorhersage von `Windfinder.com` für Europa mit der Windrichtung als Vektorfeld und der Windstärke als Farbspektrum 10

3.1 Interaktive WebGL-Applikationen für globales Wetter 13
3.2 Interaktive iOS-Applikationen für globales Wetter 14

5.1 OpenGL ES 2.0 Rendering Pipeline. Die mit einem Shader programmierbaren Abschnitte der Pipeline sind rötlich hervorgehoben, die fest eingebauten Komponenten blau und die Datenbuffer grün. 21
5.2 Primitives in OpenGL, zum Zeichnen ohne Index Buffer müssen die Vertices in aufsteigender Reihenfolge angegeben werden 23
5.3 iOS Framework-Hierarchie 28

6.1 ER-Diagramm der Server-Datenbank 36
6.2 Diagramm: Ablauf des Programms. Nach einer Initialisierung geht das Programm in die Hauptschleife über, wobei verschiedene Ereignisse (blau) weitere Komponenten (rötlich) des Programms aktivieren oder Anfragen an den Server (grün) senden. 38
6.3 Klassendiagramm: „model“-Package 38
6.4 Klassendiagramm: „view“-Package (Ausschnitt 1) 39
6.5 Klassendiagramm: „graphics“-Package 40
6.6 Klassendiagramm: „controller“-Package (Ausschnitt 1) 41
6.7 Klassendiagramm: „controller“-Package (Ausschnitt 2) 42
6.8 Diagramm: zeitliche Interpolation der Wetterdaten, blau markierte Bereiche der Daten liegen im Cache vor und werden als Texturdaten mit Hilfe eines Fragment Shaders interpoliert (bei den Texturen sind die Werte der Temperatur rötlich dargestellt) 43
6.9 Klassendiagramm: „controller“-Package (Ausschnitt 3) 44

6.10 Darstellungen der Rechtecks-Textur für die dynamische Bestimmung von sichtbaren Datenteilmengen . . . 45
6.11 Räumliche Interpolation und Approximation (WebGL) . . . 46
6.12 Diskretes und kontinuierliches Farbspektrum . . . 47
6.13 Umsetzung des Farbspektrums, mit dessen Hilfe die Temperatur dargestellt wird (WebGL/B-Spline8/Kontinuierlich) . . . 48
6.14 Umsetzung des Farbspektrums, mit dessen Hilfe die Temperatur dargestellt wird (iOS/Bilinear(Shader)/Diskret) . . . 48
6.15 Liniengenerierung im Datengitter . . . 49
6.16 Umsetzung der Isolinien (+Temperatur) (iOS/Bilin(Shader)) . . . 50
6.17 Ablauf des Partikelsystems . . . 51
6.18 Umsetzung der Partikel (+Temperatur) (WebGL/BSpline8) . . . 52
6.19 Windfahnen-Symbole, die verschiedene Windstärken in Knoten symbolisieren . . 53
6.20 Umsetzung der Windfahnen (+Temperatur) (iOS/Bilinear(Shader)) . . . 54
6.21 Verschiedene Geometrien für Positionen der Windfahnen-Symbole mit Blickwinkel auf die Polstelle (WebGL) . . . 54
6.22 Klassendiagramm: *TypeConfig* . . . 55
6.23 Klassendiagramm: „view“-Package (Ausschnitt 2) . . . 56
6.24 Umsetzung der OpenGL-Oberfläche mit OpenGL ES 2.0, links/mitte ist ein Farbspektrum zu sehen, links/unten Schaltflächen und rechts/unten ein Slider (+Erdtextur +Bewölkung) . . . 58
6.25 Umsetzung der OpenGL-Oberfläche mit WebGL, links/mitte ist ein Farbspektrum zu sehen, links/unten Schaltflächen und rechts/unten ein Slider . . . 59
6.26 Plattformspezifische Elemente der Benutzeroberfläche der iOS- und Web-Version 60
6.27 Besonderheiten bei der WebGL-Entwicklung . . . 62
6.28 Analyse der OpenGL ES-Performanz (iOS) . . . 63

A.1 Darstellung der Temperatur anhand eines Farbspektrums, des Luftdrucks durch Isolinien (grün), der Windrichtung und -stärke durch Partikel und Windfahnen, Küstenlinien und der Benutzeroberfläche (WebGL) . . . 72
A.2 Darstellung der Temperatur anhand eines Farbspektrums, der Windrichtung und -stärke durch Partikel und Windfahnen, Küstenlinien und der Benutzeroberfläche (iPad3 Foto) . . . 73

Alle Screenshots von iOS-Geräten wurden per Airplay-Streaming angefertigt.

D Literaturverzeichnis

[1] Bourke, Paul: CONREC - A Conturing Subroutine. In: *Byte, Nr. 6* (1987), June

[2] Crockford, Douglas: *JavaScript: The Good Parts.* O'REILLY, 2008

[3] Engelhardt, David: *Interactive visualization of weather data on smartphones with M3G.* Osnabrück, Universität Osnabrück, Masterarbeit, November 2009

[4] Flanagan, David: *JavaScript - The Definitive Guide.* O'REILLY, 2011

[5] Gamma, Erich et. a.: *Design Patterns. Elements of Reusable Object-Oriented Software.* Addison-Wesley, 1994

[6] Kochan, Stephen G.: *Programming in Objective-C 2.0.* Addison-Wesley, 2009

[7] Lorensen, Wlliam E.; Cline, Harvey E.: Marching Cubes: A high resolution 3D surface construction algorithm. In: *Computer Graphics, Volume 21, Number 4* (1987), July

[8] Lynch, Peter: The origins of computer weather prediction and climate modeling. In: *Journal of Computational Physics* (2007), Februar

[9] Munshi, Aaftab et. a.: *OpenGL ES 2.0 Programming Guide.* Addison-Wesley, 2008

[10] Richard S. Wright, Jr.: *OpenGL SuperBible Fifth Edition.* Addison-Wesley, 2011

[11] Stark, Benjamin: *Flash Weather - Vektorisierung von raum- und zeitbezogenen Daten zur Visualisierung mit Macromedia Flash.* Osnabrück, Universität Osnabrück, Diplomarbeit, Februar 2001

[12] Vince, John: *Mathematics for Computer Graphics.* Springer-Verlag, 2006

[13] Wenke, Henning: *Earth Weather 3D - Visualisierung und Animation dynamisch bestimmter Teilmengen von Wetterdaten auf Webseiten mit OpenGL.* Osnabrück, Universität Osnabrück, Masterarbeit, Juni 2007

[14] Wenke, Henning: *3D Klimadatenvisualisierung mit OpenGL.* Osnabrück, Universität Osnabrück, Bachelorarbeit, März 14, 2005

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe. Die Arbeit wurde bisher keiner Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Bohmte, 3. Februar 2014

Erik Wittkorn, B.Sc.