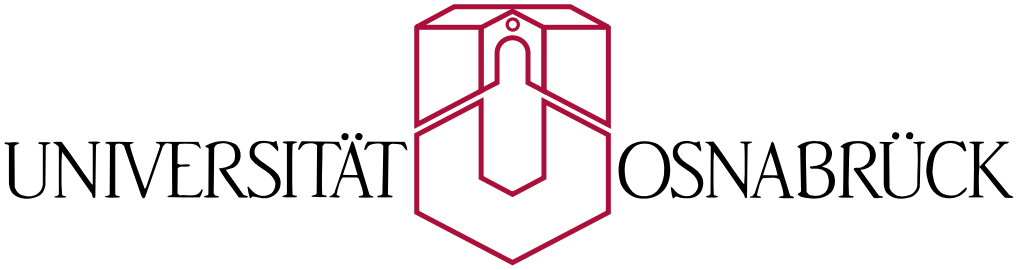


SVGWeather

Entwicklung einer SVG Web Mapping Applikation zur Visualisierung von vierdimensionalen Daten am Beispiel von Wettervorhersagedaten

Dissertation
von
Ralf Kunze

Gutachter
Prof. Dr. Oliver Vornberger
Prof. Dr. Heinrich Müller

Fachbereich Mathematik/Informatik
Universität Osnabrück

August 2006

Vorwort

Diese Dissertation entstand an der Universität Osnabrück am Institut für Informatik.

Danksagung

An dieser Stelle möchte ich folgenden Personen für ihre Unterstützung bei der Erstellung dieser Dissertation danken:

- Herrn Prof. Dr. Oliver Vornberger für die sehr gute Betreuung der Arbeit, die konstruktiven Hinweise und die vielen guten Anregungen
- Herrn Prof. Dr. Heinrich Müller für die Begutachtung der Arbeit und die konstruktiven Vorschläge
- Friedhelm Hofmeyer für die Bereitstellung der benötigten Hard- und Software im Institut für Informatik
- Allen Studenten, die durch die Mitarbeit an Projekten und Abschlussarbeiten zu dieser Arbeit beigetragen haben, insbesondere Sven Albrecht, Ingo Holzenkamp, Karl kleine Kruse, Dorothee Langfeld und Henning Wenke
- Dorothee Langfeld und Robert Mertens für das Korrekturlesen der Arbeit und für viele gute konstruktive Hinweise und Diskussionen rund um das Thema SVG
- Meiner Mutter Marianne Kunze, die mir das Studium ermöglicht hat und mich während der Arbeit an der Dissertation immer wieder motiviert und unterstützt hat

Warenzeichen

Inhaltsverzeichnis

Abbildungsverzeichnis

Quellcodeverzeichnis

I Prolog

1 Einleitung

„Das Wetter ist unabhängig am Werk ... immer auf der Suche nach neuen Mustern, mit denen es ausprobiert, ob sie sich auf den Menschen auswirken.“ (Mark Twain, 1835-1910, amerikanischer Schriftsteller)

Als Wetter bezeichnet man den spürbaren, kurzfristigen Zustand der Atmosphäre an einem bestimmten Ort. Wetter tritt unter anderem durch Sonnenschein, Bewölkung, Regen, Wind und Temperatur in Erscheinung. Wird das Wetter für einen bestimmten Ort über mehrere Tage charakterisiert, spricht man von der Witterung. Als Wetterlage wird der Zustand der Atmosphäre für ein größeres geographisches Gebiet bezeichnet. Das Klima beschreibt das durchschnittliche Wetter über längere Zeiträume, indem die Messwerte gemittelt werden. Dadurch kann für einen bestimmten Ort der typische Ablauf der Witterung angegeben werden. Zur Zeit wird oft auf einen möglichen Klimawandel hingewiesen. Hierbei werden eher globale Veränderungen der Temperatur und anderer Faktoren rückwirkend betrachtet und mittels Klimasimulationen für die Zukunft vorausberechnet.

1.1 Ausgangspunkt der Arbeit

Das Wetter ändert sich von Tag zu Tag, sogar von Minute zu Minute und seit den Anfängen wird alles Leben auf der Erde von Wetter und Klima bestimmt [Walch]. Die zukünftige Wetterentwicklung interessiert die Menschen bereits seit Jahrtausenden. Es wird schon lange versucht, das Wetter in Regeln zu fassen, um so besser auf das kommende Wetter vorbereitet zu sein. Ein Regelwerk sind die *Lostage*, eher unter dem Begriff *Bauernregeln* bekannt [Walch]. Es wird davon ausgegangen, dass das Wetter ausgehend von dem Zustand an einem Lostag einen bestimmten Verlauf nehmen würde, welcher aus früheren Wetteraufzeichnungen ermittelt wurde.

Um 1900 entstanden viele Wetterdienste, die auf wissenschaftlicher Basis versuchten, das Wetter vorherzusagen. Seit 1950 gab es die ersten mathematischen Modelle zur Wettervorhersage, die mit den Jahren durch bessere Modelle, zahlreiche Wetterstationen und durch Satellitenbilder immer mehr verbessert wurden. Alle Wettervorhersagen basieren auf der Betrachtung des aktuellen Zustands der Atmosphäre und auf regionalen geographischen Besonderheiten, wie Gebirgen, großen Seen oder großen Städten, die das Wettergeschehen beeinflussen können.

Zur Zeit sind Wettervorhersagen für die kommenden drei Tage zu 75 Prozent verlässlich. Zuverlässigere Wettervorhersagen sind nur schwer zu berechnen [Bergmann]. Dies liegt an verschiedenen Gründen:

- Ungenaue Datengrundlage: Das Netz zur Messung des aktuellen Wettergeschehens ist zu grobmaschig. Dadurch ist zu wenig über den aktuellen Zustand der Atmosphäre bekannt, was dazu führen kann, dass bestimmte Effekte nicht in eine Wettervorhersage eingehen.

- Chaotisches System: Das Wetter ist lediglich in größeren Regionen als stabil zu bezeichnen. Betrachtet man lokale Gebiete gibt es zu viele chaotische Faktoren, die nicht vorhergesagt werden können.
- Ungenaue Rechenmodelle: Die komplexe Struktur der Atmosphäre kann nur abstrahiert in Modelle zur Wettervorhersage eingehen. Außerdem sind die anfallenden Datenmengen sehr groß und die Rechenzeit stellt einen limitierenden Faktor dar. So nützt die Wettervorhersage nur, wenn sie schnell genug berechnet werden kann.

Lokale kurzfristige Wetterprognosen weisen eine wesentlich höhere Zuverlässigkeit auf. So kann das Wetter für die nächsten ein bis drei Stunden mit einer Genauigkeit von bis zu 90 Prozent vorhergesagt werden. Derartige Wettervorhersagen werden als *Nowcasting* bezeichnet.

Statistisch gesehen kann auch der Laie ohne besondere mathematische Rechenmodelle eine gute Wetterprognose für den nächsten Tag stellen. So trifft die Aussage „*heute wird's so wie gestern*“ im mitteleuropäischen Raum zu 60 Prozent zu und einfache Wetterstationen für den Hausgebrauch mit geeignetem Außenfühler können eine Prognose mit 60-75 Prozent Zutreffwahrscheinlichkeit bestimmen.

Fast alle Wetterprognosen sind so genannte numerische Wettervorhersagen. Die Prozesse in der Atmosphäre werden erfasst und als Messwerte einzelnen Gitterzellen zugeordnet. Die Gitterzellen erstrecken sich zum einen über die Erdoberfläche und zum anderen in die Höhe. Je kleiner die Gitterzellen sind, um so genauer kann der aktuelle Zustand der Atmosphäre wiedergegeben werden. Durch physikalische Gesetzmäßigkeiten, die in mathematische Gleichungen gefasst werden, lässt sich daraus die Entwicklung des Wetters bestimmen. Allerdings sind die Gleichungen so komplex, dass eine exakte (analytische) Lösung zur Bestimmung des zukünftigen Zustandes der Atmosphäre nicht möglich ist [DWD2]. Sie werden daher näherungsweise mit numerischen Verfahren gelöst. Das Ergebnis ist eine Wettervorhersage, die aus langen Zahlenreihen besteht, die der Mensch in dieser Form nur sehr schwer erfassen kann. Daher müssen die Ergebnisse geeignet aufbereitet werden. Dazu sind vor allem Wetterkarten geeignet, die in allen visuellen Medien zu finden sind (Kapitel 2).

1.2 Ergebnis der Arbeit

Thema dieser Arbeit ist die automatisierte grafische Aufarbeitung der Rohdaten einer Wetter- oder Klimaprognose für das Internet, um eine interaktive Anwendung zur Betrachtung der Daten zu erhalten. Im Internet finden sich zwar Wetterdarstellungen, die gute Ansätze aufweisen, allerdings werden interaktive und dynamische Techniken nur wenig genutzt. Daher wurde mit Hilfe von Bacheloranden und Diplomanden der Universität Osnabrück erstmalig eine Software erstellt, mit der eine komplexe Web Mapping Applikation generiert werden kann, die sowohl für die Darstellung von allgemeinen geographischen Daten, als auch für die Visualisierung von räumlichen Daten mit einem

zeitlichen Aspekt geeignet ist. Besonderen Wert wurde auf die Visualisierung von Wetterdaten gelegt, die in der Web Mapping Applikation komfortabel eingebunden werden können. Dadurch ist es möglich, eine interaktive und dynamische Visualisierung einer Wettervorhersage im Internet zu präsentieren. Zusätzlich wurde darauf geachtet, dass die Erstellung einer Wettervorhersage in Kombination mit der Web Mapping Applikation automatisiert erfolgt und dennoch sehr frei und einfach konfiguriert werden kann. Es ist keinerlei Fremdsoftware erforderlich, um eine solche Web Mapping Anwendung zu erzeugen. Lediglich die zu visualisierenden Daten müssen vorhanden sein.

Bei der Implementation der Software wurden konsequent alle Forderungen an eine interaktive Wettervorhersage (Kapitel 2.4) umgesetzt. So sind die in der Web Mapping Applikation vorhandenen Daten frei kombinierbar, Ausschnitte können gezielt betrachtet werden, eine Navigation durch die Zeit wird ermöglicht und die Wettervorhersagedaten können mit einer beliebig genauen Geographie versehen werden. Dadurch ist es erstmals gelungen eine komfortable Wettervorhersage für das Internet zu präsentieren, die eine umfangreiche Analyse der kommenden Wetterlage erlaubt.

Um die Anwendbarkeit der automatisierten Prozesskette zu demonstrieren, kann eine täglich aktualisierte interaktive Wettervorhersage für Europa unter folgender URL betrachtet werden:

http://www.svg-weather.de

1.3 Aufbau der Arbeit

Zu Beginn werden beispielhaft einige Arten von Wettervorhersagen vorgestellt. Dazu wird betrachtet, welche Art der Darstellung bisher in verschiedenen Medien verwendet wird und mit welchen Techniken eine Wettervorhersage visualisiert werden kann. Von diesen Betrachtungen ausgehend werden die Anforderungen an eine Wettervorhersage im Internet festgelegt. Im Folgenden werden die für die Arbeit notwendigen Grundlagen näher erläutert. Insbesondere wird das Vektorgrafikformat *Scalable Vector Graphics* näher betrachtet. Danach werden die Ausgangsdaten näher beschrieben und es wird erklärt wie die geographischen Daten der dreidimensionalen Erde auf zweidimensionalen Karten dargestellt werden können. Anschließend werden die Darstellungsformen verschiedener Wetterausprägungen betrachtet. Da Wetterkarten immer eine geographische Komponente aufweisen, wird als letzter Teil in den Grundlagen vorgestellt, wie geographische Karten interaktiv im Internet dargestellt werden können. Im zweiten Teil folgt die Realisierung. Hier wird vorgestellt, wie die Rohdaten einer Wetter- oder Klimaprognose aufbereitet werden können. Dazu gehört die Vektorisierung der Daten und die Kantenglättung von Polygonen. Danach wird die Erstellung der interaktiven Wettervorhersage erläutert. Im dritten Teil werden Projekte vorgestellt, die zusätzlich aus dieser Arbeit hervorgegangen und zum Teil noch nicht abgeschlossen sind. So wird gezeigt, wie beliebige georeferenzierte Daten in die Web Mapping Applikation integriert werden können. Außerdem wird erläutert, wie die Wetterdaten in GoogleEarth eingebunden werden können. In einem weiteren Kapitel wird ein zusätzlicher Verbreitungsweg neben dem Internet vorgestellt und es wird gezeigt, wie die Daten dreidimensional in einem Applet dargestellt werden können. Als letzter Teil folgt ein kurzer Ausblick und das Fazit.

2 Motivation

Für viele ist es wichtig, die zukünftige Wetterentwicklung abschätzen zu können, da beispielsweise Bauvorhaben vom Wetter abhängig sind, die Agrarwirtschaft vom Wetter abhängig ist, private Unternehmungen je nach Wetter geplant werden oder die Planung der Wasserversorgung von den zukünftigen Temperaturvorhersagen abhängt.

Die Entwicklung des zukünftigen Wetters hängt von vielen lokalen und globalen Faktoren ab, weshalb die Berechnung von Wettervorhersagen komplex und zeitintensiv ist. Das Ergebnis solcher Simulationen sind lange Zahlenkolonnen für unterschiedliche Wetterausprägungen, wie die Temperatur, die Bewölkung oder den Luftdruck. Derartige Daten sind selbst von Experten nur schwer zu überblicken. Daher gibt es unterschiedliche Darstellungsvarianten für Wettervorhersagen. Je nach Medium, in dem sie veröffentlicht werden, unterscheiden sie sich sehr stark in der Visualisierung. Es können drei Medientypen unterschieden werden, in denen Wettervorhersagen visualisiert werden: Printmedien, Fernsehen und Internet.

2.1 Printmedien

In Printmedien werden Wettervorhersagen sehr abstrahiert dargestellt. Da es sich um ein statisches Medium handelt, wird meist eine Karte des entsprechenden Landes oder der Region verwendet und an vorher definierten Stellen werden Symbole eingefügt, die das kommende Wetter darstellen sollen (Abbildung 2.1).

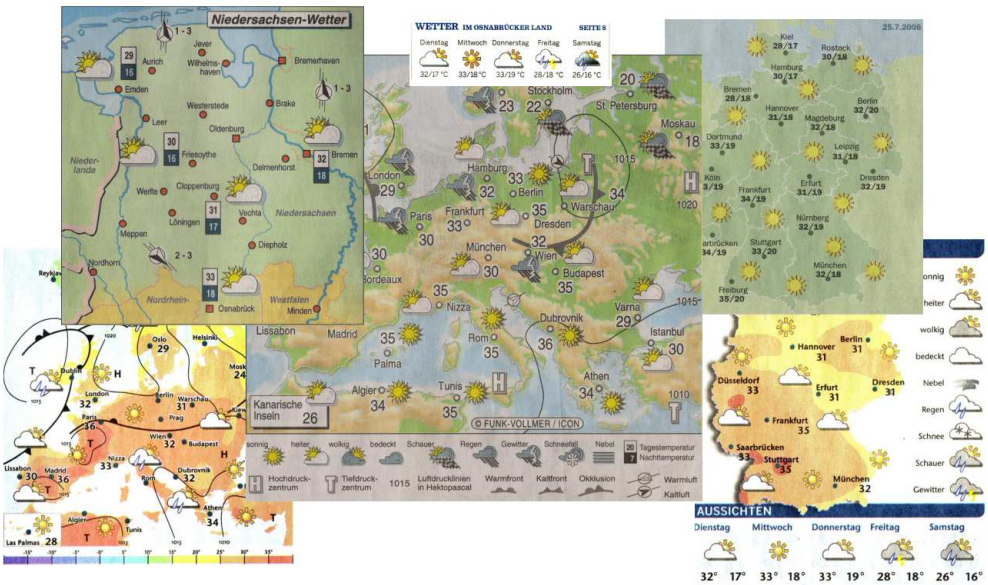


Abbildung 2.1: Darstellungsformen von Wettervorhersagen in Printmedien

Bei den Symbolen handelt es sich meist um eine stark vereinfachte Darstellung der voraussichtlichen Wetterlage. Es wird meist nur zwischen sonnig, leicht bewölkt, bewölkt,

leichte Schauer und Regen unterschieden. Dazu kommen noch Symbole für Gewitter oder Schneefall. Neben diesen Piktogrammen wird die Temperatur eingezeichnet. Da in einer Tageszeitung meist sorgsam mit dem Platz umgegangen werden muss, sind die Karten sehr vereinfacht. Es sind nur wenige Städte und nur die großen Flüsse zur Orientierung angegeben. Die Vorhersage für die kommenden Tage wird jeweils in eigenen Grafiken dargestellt.

Handelt es sich um eine überregionale Tageszeitung, wird meist eine Wettervorhersage für das gesamte Land dargestellt, die es dem Betrachter sehr gut ermöglicht, die Großwetterlage abzuschätzen. In Regionalzeitungen beziehen sich die Wettervorhersagen auf kleinere Gebiete, wodurch die Wetterlage detaillierter dargestellt werden kann.

Ein Nachteil der gedruckten Wettervorhersage ist ihre Ungenauigkeit. Durch die Verwendung von Symbolen kann nur ein grober Überblick über die Wetterlage gegeben werden. Wie das Wetter an einem bestimmten Ort sein wird, lässt sich nur schwer feststellen. Außerdem ändert sich die Wetterlage sehr häufig, was in einem Printmedium nicht berücksichtigt werden kann. Da das Medium statisch ist, können zukünftige Wetterentwicklungen nur schwer eingeschätzt werden, da die Betrachtung für die kommenden Stunden nicht möglich ist. So ist beispielsweise nicht erkennbar, in welche Richtung eine Regengebiet ziehen wird.

2.2 Fernsehen

Im Fernsehen werden Wettervorhersagen mit Symbolen, mit Farbverteilungen auf einer Karte oder mit Satellitenbildern dargestellt (Abbildung 2.2). Dabei sind die Grafiken zum Teil statisch und zum Teil animiert. So werden einzelne Symbole animiert dargestellt, die räumliche Veränderung von Regengebieten wird visualisiert oder es werden dreidimensionale Wetterflüge gezeigt. Dazu kommt, dass die Wettervorhersage fast immer mit einer verbalen Erläuterung versehen wird.

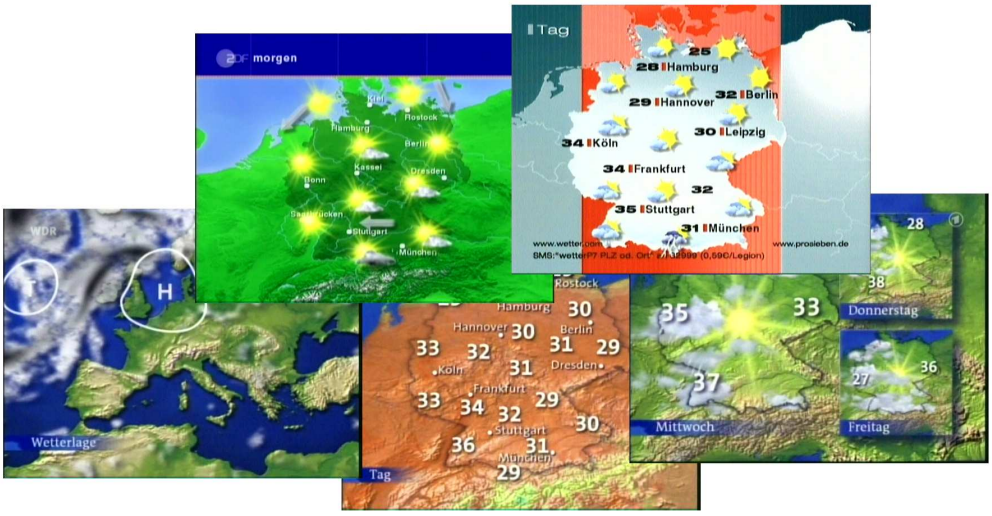


Abbildung 2.2: Darstellungsformen von Wettervorhersagen im Fernsehen

Wettervorhersagen im Fernsehen sind durch die Erläuterungen meist sehr gut verständlich und durch die Animationen sind zukünftige Wetterentwicklungen sehr gut erkennbar. Die Wettervorhersagen können mehrmals am Tag aktualisiert werden und es können aktuelle Satellitenbilder und andere aktuelle Messwerte mit in die Wettervorhersage übernommen werden. Da die Visualisierung der Wettervorhersage vorberechnet wird, können die Darstellungen optisch gut präsentiert werden.

Allerdings werden die Wettervorhersagen nur zu einem bestimmten Zeitpunkt ausgestrahlt, so dass man unter Umständen warten muss, um eine Vorhersage sehen zu können. Außerdem ist das Medium Fernsehen immer noch nicht mobil genug, so dass man unterwegs nicht ohne weiteres eine Wettervorhersage betrachten kann. Ein weiterer Nachteil ist die mangelnde Interaktivität. Der Betrachter hat also keine Möglichkeit die visualisierten Datensätze beliebig zu kombinieren.

2.3 Internet

Die Darstellungsformen von Wettervorhersagen im Internet (Abbildung 2.3) haben sich zunächst sehr stark an der Darstellung in den Printmedien orientiert. Dies ist nicht verwunderlich, da zu den Anfängen des World Wide Webs lediglich Text und statische Bilder vorhanden waren. Mit der Zeit hat sich das Internet sehr gewandelt. Dynamische Inhalte sind nun überall zu finden, dank Flash (Abschnitt 3.1) und SVG (Abschnitt 3.2) sind Webseiten bunter geworden und durch AJAX (Kapitel 4.5) ist die gesamte Struktur einer Webseite veränderbar. Diese Technologien wurden aber nur sehr zaghaft bei der Wettervorhersage eingesetzt. Die alten Systeme waren vorhanden und eine Neuerung ist mit Kosten verbunden. Außerdem ist die Erstellung einer dynamischen Anwendung wesentlich aufwändiger, als die Erstellung einer statischen Grafik.

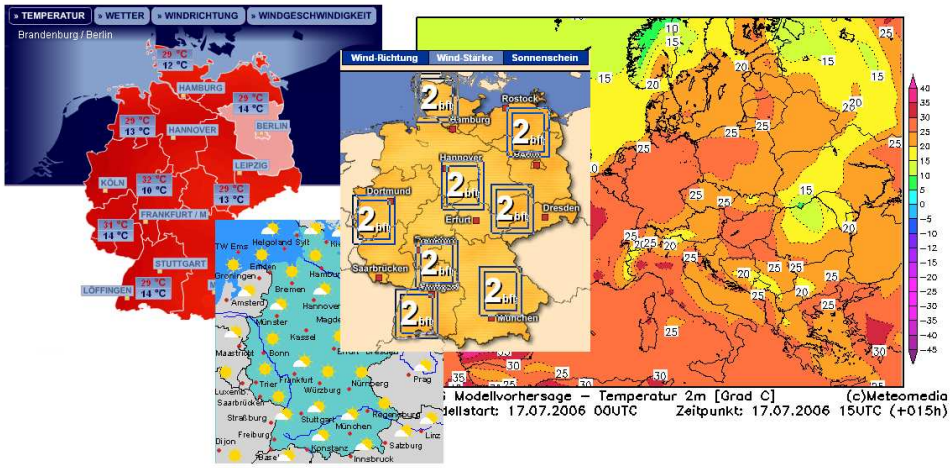


Abbildung 2.3: Darstellungsformen von Wettervorhersagen im Internet

Nach und nach wurden neuere Technologien zur Darstellung von Wettervorhersagen eingesetzt. So hat man auch bei den Grafiken Isolinien dargestellt und unterschiedliche Kar-

ten zu unterschiedlichen Wetterausprägungen wie z.B. Temperatur, Regenwahrscheinlichkeit oder Wind erstellt. Der nächste Schritt waren dann animierte GIF Grafiken. Hierbei werden mehrere statische Pixelgrafiken zu einer GIF-Animation kombiniert, welche dann im Webbrowser angezeigt wird. Dadurch wird deutlich, wie beispielsweise ein Regengebiet seine Lage verändert und ob es sich auf einen zu bewegt oder nicht.

Neuere Möglichkeiten bietet das Flash Vektorgrafik Format (Kapitel 3.1). Mittels Flash konnten aufwändigere Animationen platzsparend erstellt werden. Allerdings hat man diese Technologie meist nur dazu genutzt, um die althergebrachten Wetterkarten mit Symbolen aufzufrischen. So wurden die Symbole animiert, einzelne Bundesländer konnten visuell hervorgehoben werden und bei einem Klick auf ein Symbol wurden einige Zusatzinformationen eingeblendet. Wettervorhersagen im Internet werden immer multimedialer und interaktiver, allerdings wurden bisher noch längst nicht alle Möglichkeiten ausgenutzt.

Im Folgenden sollen einige Wettervorhersagen im Internet näher betrachtet werden, um einen aktuellen Überblick über die eingesetzten Techniken zu geben.

2.3.1 donnerwetter.de

Die Webseite www.donnerwetter.de stellt die Wettervorhersage für Deutschland klassisch mit einer Karte und Wettersymbolen dar (Abbildung 2.4(a)). Die Wetterkarte ist nicht

(a) Wettervorhersage für Deutschland

(b) Wettervorhersage für Osnabrück

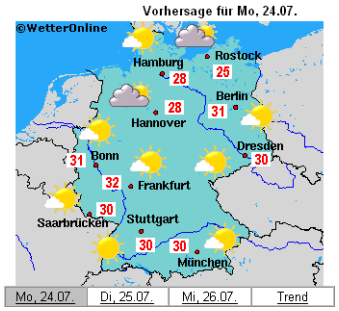
Abbildung 2.4: Wettervorhersagen auf donnerwetter.de

animiert und bietet keinerlei Interaktionsmöglichkeiten. Eine regionale Wettervorhersage ist über die Angabe eines Ortes oder einer Postleitzahl möglich. Hier erhält man keine Wetterkarte der Region, sondern nur Symbole, die das Wetter zu verschiedenen Tageszeiten charakterisieren sollen (Abbildung 2.4(b)). Sowohl bei der deutschlandweiten als auch bei der regionalen Wettervorhersage wird die Wetterlage kurz textuell erläutert.

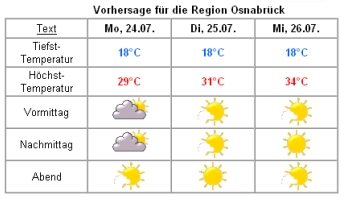
Animierte Grafiken sind auf der Webseite nicht zu finden. Dafür ist die Webseite einfach strukturiert, so dass die gewünschte Information leicht zu finden ist.

2.3.2 wetteronline.de

Auch wetteronline.de verwendet für die deutschlandweite Wettervorhersage eine Karte mit Symbolen und für die regionale Vorhersage eine Tabelle mit örtlichen Vorhersagewerten (Abbildung 2.5).



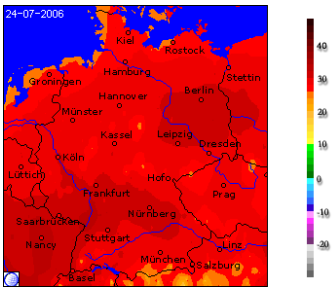
(a) Wettervorhersage für Deutschland



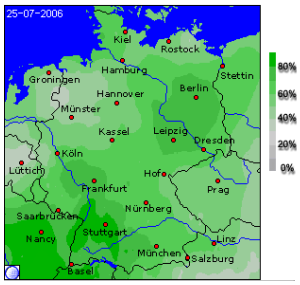
(b) Wettervorhersage für Osnabrück

Abbildung 2.5: Wettervorhersagen auf wetteronline.de

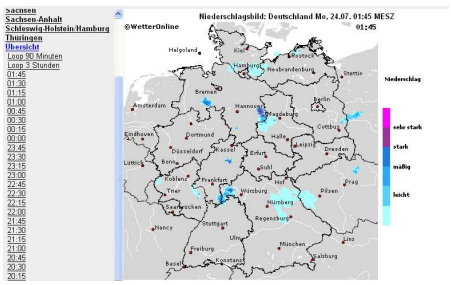
Dafür bietet wetteronline.de viele zusätzliche Karten an, auf denen die Wetterparameter wie Temperatur oder Sonnenscheindauer farbig eingezeichnet sind (Abbildung 2.6). Zudem können die Karten auch als animiertes GIF betrachtet werden, bei denen die ein-



(a) Temperatur



(b) Sonnenscheindauer



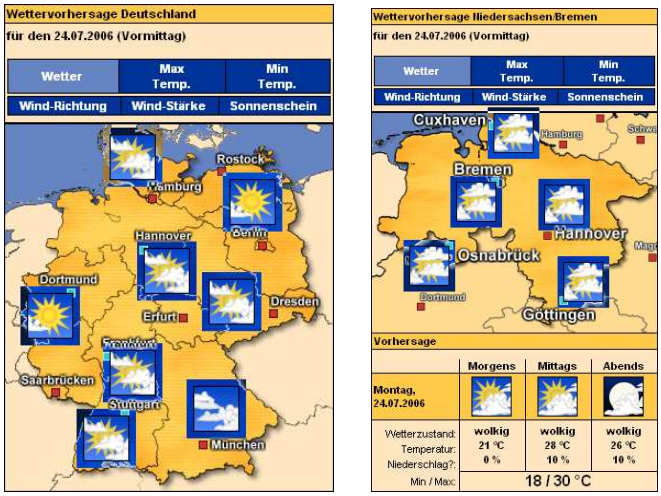
(c) Niederschlag

Abbildung 2.6: Animierte Grafiken auf wetteronline.de

zelnen Zeitpunkte der Reihe nach angezeigt werden. Auch der Niederschlag ist mit farbigen Karten dargestellt. Die Besonderheit hier ist, dass in einem animierten GIF die Niederschläge der letzten Stunden angezeigt werden können. Zusätzlich können auf der Webseite Satellitenbilder, Wasserstände, Pollenflugvorhersagen und viele weitere Informationen abgefragt werden. Wetteronline.de verfügt über ein breites Spektrum von Informationen, die grafisch recht gut aufbereitet sind. Allerdings bieten die Visualisierungen keine Interaktionsmöglichkeit, die Grafiken können nicht beliebig vergrößert werden und die in einer Grafik angezeigten Daten sind fest vorgegeben und können nicht frei kombiniert werden.

2.3.3 wetter.com

Wetter.com ist eine Webseite vom „Deutschen Wetter Fernsehen“. Auch hier werden Symbole zur Darstellung einer Wettervorhersage für Deutschland verwendet (Abbildung 2.7(a)), allerdings sind diese animiert. Neben einer Wettervorhersage für einen Ort mit Symbolen und Text, bietet wetter.com auch die Möglichkeit einzelne Bundesländer etwas genauer zu betrachten (Abbildung 2.7(b)).



(a) Wettervorhersage für Deutschland

(b) Wettervorhersage für Niedersachsen und Osnabrück

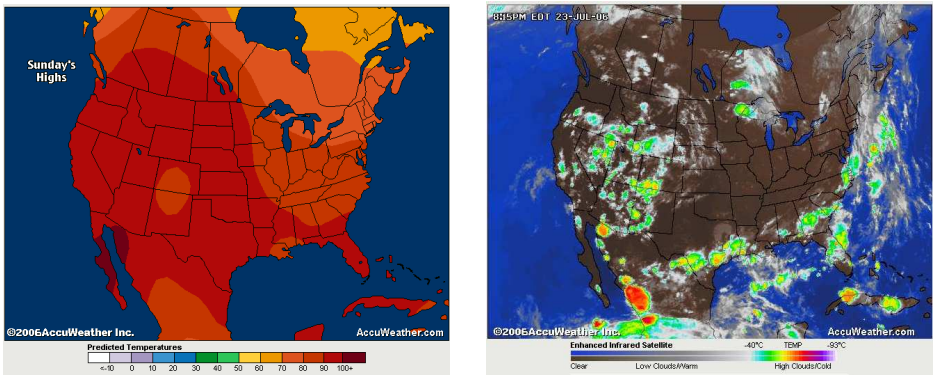
Abbildung 2.7: Wettervorhersagen auf wetter.com

Des Weiteren bietet wetter.com einige professionellere Karten an und Wetterprognosen für verschiedene Bereiche, wie Gesundheit, Heim und Garten, Auto und Verkehr oder Karten für das Agrarwetter. Zusätzlich werden die Wettervorhersagen des Webseitenbetreibers als Livestream zur Verfügung gestellt.

Leider sind auch hier die Informationen nicht frei kombinierbar und die Karten sind nicht interaktiv.

2.3.4 accuweather.com

AccuWeather ist der größte Wetterdienst der USA. AccuWeather legt großen Wert auf eine ansprechende Darstellung der Wetterprognosen und setzt eine Vielzahl von unterschiedlichen Techniken ein. Die Wetterkarten werden meist nicht mit Symbolen versehen. Statt dessen sind die Wetterparameter farblich in eine Karte eingezeichnet (Abbildung 2.8(a)). Fast alle Karten sind animiert und bieten so einen guten Überblick über die Wetterveränderungen. Unterstützt werden die Karten durch Satellitenbilder, die ebenfalls animiert und für unterschiedliche Regionen vorhanden sind (Abbildung 2.8(b)).



(a) Temperatur (b) Satellitenbild

Abbildung 2.8: Wettervorhersage auf accuweather.com

Auch bei der Webseite von AccuWeather können die Wetterdaten nicht beliebig kombiniert werden. Eine Vergrößerung einer Region ist ebenso nicht möglich.

2.4 Anforderungen an die Visualisierung einer Wettervorhersage im Internet

Im vorangegangenen Abschnitt wurden einige Webseiten mit Wettervorhersagen vorgestellt. Einen groben Überblick über die kommende Wetterlage bieten alle vorgestellten Webseiten, allerdings werden die multimedialen Möglichkeiten des Internets meist nur sehr spärlich genutzt. Die Darstellung einer Wettervorhersage kann wesentlich interaktiver und dynamischer realisiert werden. Damit ein Betrachter einer Wettervorhersage im Internet die für ihn wichtigen Informationen erhalten kann, sollten folgende Punkte erfüllt sein:

Zoomfähigkeit: Der Überblick über ein ganzes Land liefert einen ersten Eindruck der kommenden Wetterlage. Um aber das Wetter aus einer Region betrachten zu können, muss man einzelne Ausschnitte der Karte verlustfrei vergrößern können.

Genaue Geographie: Damit sich ein Betrachter einer Wetterkarte gut orientieren kann, ist es notwendig, detailreiches Kartenmaterial anzeigen zu können. Dabei sollten je nach Vergrößerung weitere Orte, Flüsse oder andere geographische Merkmale eingeblendet werden.

Freie Kombination der Daten: Damit der Betrachter nicht mit zu vielen Informationen überfordert wird, ist es notwendig, dass er gezielt einzelne Informationen ein- oder ausblenden kann. Wie bei einem Geographischen Informationssystem (GIS) ist es zweckmäßig die unterschiedlichen Informationen auf einzelnen Ebenen, so genannten Layern, anzuordnen.

Zeitverlauf: Der Betrachter sollte in die Lage versetzt werden, die zeitliche Entwicklung des Wetters oder Klimas nachvollziehen zu können. Dazu sollten einzelne

Zeitpunkte gezielt angesteuert werden können oder aber einzelne Zeitpunkte automatisch der Reihe nach angezeigt werden. Nur so ist der Verlauf und die zukünftige Entwicklung der Wetterveränderungen gut zu erkennen.

Isolinien: Die Verwendung von Symbolen in Wetterkarten ist sehr ungenau. Besser ist es, wenn man Isolinien und eine Einfärbung der Flächen zwischen den Isolinien verwendet. So ist für jeden Punkt der Karte klar, welche Wetterausprägung vorhanden ist. Wird ein Ausschnitt vergrößert, sollte sich zudem die Genauigkeit der Isolinien und in Folge dessen auch die der Flächen anpassen, damit feinere Unterschiede in den Wetterausprägungen deutlich werden.

Datenmenge: Trotz der vielen gewünschten Features muss die zu übertragende Datenmenge gering gehalten werden, um eine schnelle Darstellung der Informationen zu gewährleisten.

Automatisierte Erstellung: Die Visualisierung der Wetterkarte muss nach Möglichkeit automatisiert ablaufen. Dadurch werden die Kosten zur Erstellung gering gehalten und es ist leichter, das Kartenmaterial auf dem neuesten Stand zu halten.

Kompatibilität: Eine Wettervorhersage sollte auf den unterschiedlichsten Betriebssystemen mit den verschiedensten Webbrowsern betrachtet werden können. Daher ist auf eine hohe Kompatibilität der Anwendung zu achten.

II Grundlagen

3 Vektorgrafik

Bei Grafikformaten werden grundsätzlich Pixel- und Vektorgrafiken unterschieden. Bei Pixelgrafiken muss die Information eines jeden Bildpunktes angegeben werden. Dadurch wird relativ viel Speicherplatz verbraucht und einzelne Objekte sind nicht als solche klar zu erkennen. Bei Vektorgrafiken werden die geometrischen Objekte über die Angabe ihrer Koordinaten definiert und mit Eigenschaften versehen. So kann Text durch vordefinierte Buchstaben platzsparend in einer Vektorgrafik integriert werden oder ein Rechteck über die Angabe zweier Punkte (linke obere und rechte untere Ecke) definiert werden. Der Rand und der Inhalt des Rechteckes können mit Farbwerten versehen werden. Wenige Informationen beschreiben also größere Bereiche des Bildes. Daher wird bei Vektorgrafiken in der Regel wesentlich weniger Speicherplatz benötigt als bei Pixelgrafiken. Die in der Vektorgrafik definierten Objekte sind als solche zu erkennen und können leicht verschoben, anders eingefärbt oder in der Größe verändert werden, wodurch Vektorgrafiken einfach zu animieren sind. Andererseits ist es aber mit Vektorgrafiken nur schwer möglich einen fotorealistischen Eindruck zu erzeugen. Dazu würden zu viele Objekte benötigt. In diesem Bereich stellen die Pixelgrafiken die einzige Möglichkeit dar, feine Farbnuancen darzustellen und einen fotorealistischen Eindruck zu erzeugen. Daher werden bei Bedarf auch Pixelbilder in eine Vektorgrafik eingebunden.

Pixel- und Vektorgrafiken unterscheiden sich aufgrund ihrer unterschiedlichen Art, eine Grafik abzuspeichern, auch in der Anzeige. Da die Dateigröße von Pixelgrafiken schnell sehr groß werden kann, werden diese Grafiken meist komprimiert. Zum Einsatz kommen verlustfreie und verlustbehaftete Komprimierungsverfahren. Zu nennen wären hier unter anderem das GIF (Graphics Interchange Format), PNG (Portable Network Graphics) und das JPEG (Joint Photographic Experts Group) Format [Miano]. Für die Anzeige auf dem Bildschirm muss also zunächst die Datei dekomprimiert werden, was in der Regel recht einfach ist. Danach werden die Farbinformationen eines jeden Bildpunktes ausgelesen und dargestellt. Bei Vektorgrafiken muss das Bild für die Anzeige erst berechnet werden. So muss für jeden Bildpunkt auf dem Bildschirm erst ermittelt werden, welchen Farbwert er einnimmt. Je mehr Objekte vorhanden sind, um so mehr Rechenarbeit ist notwendig.

Ein weiterer Aspekt ist die Vergrößerung eines Bildausschnittes. Bei einer Pixelgrafik sind bei einer Vergrößerung sofort die einzelnen Bildpunkte zu erkennen (Abbildung 3.1). Bei der Vektorgrafik wird bei einer Vergrößerung der Bildausschnitt neu berechnet. Dadurch sind alle Linien und ggf. auch Farbübergänge ohne Qualitätsverlust darstellbar.

Die beiden Formate sind also recht unterschiedlich und daher ist auch ihr Einsatzgebiet in der Regel klar definiert. Um zu entscheiden, welches Format in dieser Arbeit eingesetzt werden soll, müssen zunächst die Anforderungen betrachtet werden. Für die Anzeige von georeferenzierten Informationen ist es notwendig, einzelne Informationsebenen ein- oder auszublenden, Kartenausschnitte verlustfrei zu vergrößern und ggf. Teile der Anzeige zu animieren, wie bereits in Abschnitt 2.4 gefordert wurde. Außerdem werden in der Regel eingefärbte Flächen, Linien oder Punkte dargestellt, so dass eine fotorealistische Darstellung nicht notwendig ist. Daher ist in dieser Arbeit ganz klar das Vektorgrafikformat vorzuziehen.

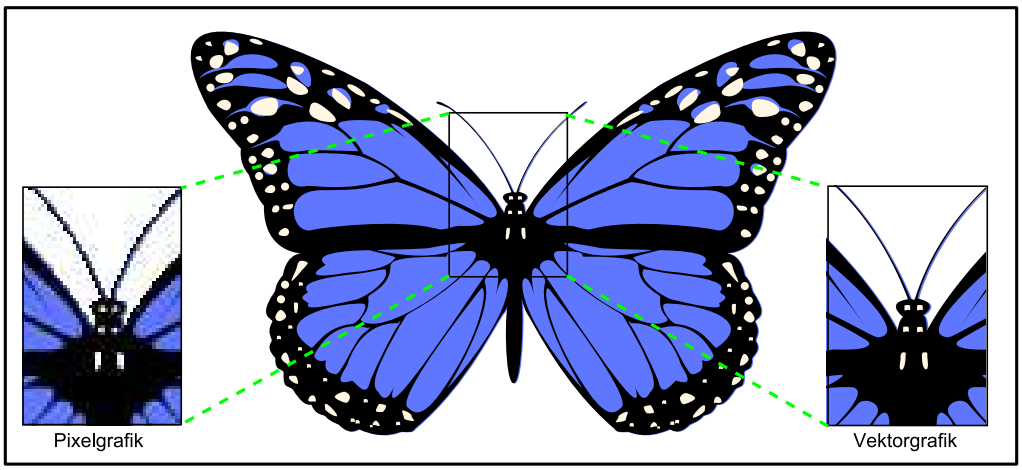


Abbildung 3.1: Pixel- vs Vektorgrafik

In den nächsten Unterkapiteln werden die beiden gängigsten Vektorgrafikformate für das Internet vorgestellt und miteinander verglichen. Dabei handelt es sich um *Macromedia Flash* und *Scalable Vector Graphics* (SVG).

3.1 Macromedia Flash

Macromedia Flash (kurz Flash) ist ein proprietäres Vektorgrafik Format der amerikanischen Firma Adobe [Adobe1], ehemals Macromedia. Im Jahr 1997 wurde von Macromedia die Flash Version 1 veröffentlicht. Ab der Version 4, die im Jahr 1999 auf den Markt kam, wurde die Programmiersprache *ActionScript* eingeführt, die es erlaubte, komplexere Flash Anwendungen zu programmieren. Es konnten Formulare erstellt und der Inhalt der Formulare zur Auswertung an einen Webserver geschickt werden. 2005 wurde Macromedia von der Firma Adobe übernommen. Zu dieser Zeit war Flash MX 2004 (Version 7.2) aktuell. Kurz nach der Übernahme von Macromedia wurde Flash 8 entwickelt, welches eine Videostreaming Unterstützung bietet. Der Markenname *Macromedia* wird nicht weitergeführt. Zunächst wird aber noch von Macromedia Flash gesprochen, was aber bei der nächsten Version in Adobe Flash geändert werden soll.

In den Anfangsjahren von Flash erreichte Macromedia durch geschicktes Marketing eine schnelle Verbreitung des Formates. Macromedia bewegte die Browserhersteller dazu, das Flash Plugin zusammen mit dem Webbrowser auszuliefern. Dadurch wurde Flash bereits bei vielen Internetnutzern gleich mit installiert. Da viele Nutzer das Plugin bereits hatten, war die Hemmschwelle, Flash Anwendungen auf Webseiten zu stellen, sehr gering und Flash konnte sich schnell weltweit verbreiten. Jetzt werden Webbrowser in der Regel nicht mehr mit dem Flash Plugin ausgeliefert, aber das Format wird auf so vielen Webseiten eingesetzt, dass die Internetnutzer sich das Plugin selbst installieren. Der Anteil der Flashinstallationen in Bezug auf die gesamten Internetnutzer ist in Tabelle 3.1 zu sehen (für Asien liegen keine Daten vor). Ermittelt wurden die Daten von dem US Unternehmen NPD Group [Adobe4], die Marktanalysen zu webbasierten Technologien durchführen. Für die Studie wurden 2.000 Personen je Kontinent befragt.

Verfügbarkeit nach Version	Flash 5	Flash 6	Flash 7	Flash 8
USA	97.7%	97.2%	94.8%	69.3%
Kanada	98.0%	96.9%	94.1%	79.0%
Europa	98.3%	97.2%	95.1%	76.1%

Tabelle 3.1: Verfügbarkeit des Macromedia Flash Players (Stand: April 2006) [Adobe3]

Flash wird für unterschiedlichste Zwecke eingesetzt. Oftmals dient es dazu, ein Intro für eine Webseite zu zeigen oder kleine Animationen auf einer Webseite einzubinden. Des Weiteren werden Spiele mit Flash entwickelt. Ernsthafte Anwendungen, wie interaktive Tutorials oder *Rich Internet Applications* sind mit Flash zwar möglich, aber dennoch ist der Haupteinsatz von Flash lediglich das Aufwerten einer Webseite durch Ton und Animation.

Ein neuerer Einsatz von Macromedia Flash ist die Präsentation von Videos im Internet. Durch höhere Bandbreiten ist es möglich, Videos direkt auf einer Webseite per Streaming zu veröffentlichen. Ein Problem waren dabei bisher die verschiedenen Videocodecs, so dass der Anwender ständig neue Codecs installieren musste, um sich ein Video anschauen zu können. Flash hat einen Videocodec in das Flashplugin installiert, so dass das Problem der unterschiedlichen Codecs entfällt, wenn man Videos eingebettet in Flash zur Verfügung stellt. Außerdem bietet Flash die Möglichkeit die Steuerelemente eines Videos frei zu gestalten. Das führte dazu, dass Webseiten wie http://video.google.com, http://www.youtube.com oder http://www.myspace.com Flash zur Präsentation der Videos einsetzen.

3.1.1 Das Flashformat

Beim Flashformat handelt es sich um ein streamingfähiges Binärformat. Jedes Bild des Flashfilmes liegt als so genannter Frame vor. Jeder Frame speichert die Veränderung zum vorangegangenen Frame, so dass nicht alle Informationen in einem Frame vorhanden sein müssen. Neben den Frames gibt es noch Keyframes. Sie beinhalten die gesamte Szene. Durch eine Abfolge von Frames und Keyframes kann ein animierter Film sehr platzsparend abgespeichert werden und es ist durch die Keyframes gewährleistet, dass zwischendurch Einsprungpunkte existieren, die sofort dargestellt werden können, ohne die vorherige Abfolge von Frames zu betrachten.

Die Streamingfähigkeit ist dadurch gewährleistet, dass ein Objekt zunächst definiert werden muss und erst dann verwendet werden kann. Die Shapes werden durch einzelne Tags definiert. Dabei handelt es sich um eine Abfolge von Bits, in der der Typ des Shapes, die Koordinaten und die Farbgebung definiert werden. Die einzelnen Bilder einer Flashanimation sind in Frames abgelegt, die wiederum in einzelne Records gegliedert sind.

3.1.2 Die Erstellung von Flashfilmen

Flashfilme kann man z.B. mit dem Flashstudio von Macromedia erzeugen. Das Flashstudio erlaubt es, so genannte Darsteller zu erzeugen, die auf einer Zeitleiste platziert

werden können. Diese können dann über die Zeit animiert oder mittels Farb- oder Formtweenings (mit Tweening wird bei Flash das Morphen bezeichnet) ineinander überführt werden. Diese Art der Flash Erzeugung richtet sich in erster Linie an Designer, die statische Inhalte interaktiv aufbereiten.

Wenn sich die Inhalte jedoch häufig ändern, müssen sich auch die Filme anpassen. Dazu wurde *Adobe Flex 2* [Adobe5] entwickelt, eine serverseitige Applikation, mit der Daten in einen Flashfilm nachgeladen werden können. Dabei wird der Flashfilm mit Variablen versehen, die erst auf Anfrage an den Flex Server mit Inhalt gefüllt werden. Nachgeladen werden dabei Flashinhalte, Bilder oder Texte, die auf dem Server bereitliegen oder in einer Datenbank gespeichert sind. Zusätzlich ist es möglich, die ganze Flashanimation als XML zu definieren. Diese XML Datei wird dann von dem Flexserver interpretiert und mit Leben gefüllt. Derartige Flashfilme sind so genannte Rich Internet Applications. Der Anwender hat im Webbrowser eine Flashanwendung geladen, mit der er arbeitet. Alle weiteren Daten werden erst vom Server nachgeladen, wenn es notwendig ist. Dabei wird aber nicht eine komplett neue Webseite geladen, sondern nur der wesentliche Bereich ausgetauscht. Dadurch wird der Eindruck erweckt, dass man mit einem Programm auf dem Desktop arbeiten würde und nicht mit einer Webanwendung.

Mit den beiden angeführten Möglichkeiten können Flashfilme recht einfach erstellt werden. Sollen aber sich oftmals ändernde Daten visualisiert werden sind diese Ansätze nicht brauchbar. Bei der Arbeit mit dem Flashstudio hieße es, dass die Daten immer wieder erneut von Hand visualisiert werden müssten. Der zweite Ansatz mit dem Flex Server reicht zur Visualisierung nicht aus, da lediglich Symbole, Schaltflächen oder ähnliche Flash Objekte bzw. einfache Pixelgrafiken nachgeladen werden können. Es fehlt also die Möglichkeit, wirklich zeichnen zu können.

Eine weitere Möglichkeit ist das direkte Schreiben des Flash Binärformates. Dazu existierte von Macromedia ein *Application Programming Interface (API)* für Flash, das jedoch einige Fehler aufweist und seit Jahren nicht mehr aktualisiert wurde. Daher ist das Flash API von Macromedia noch auf dem Stand von Flash in der Version 3. Es gibt aber auch neuere Projekte, die versuchen, Flash direkt zu schreiben. Besonders hervorzuheben ist das Flash API von dem Unternehmen *Flagstone Software* [Flagstone], mit dem es möglich ist, Flash in der Version *MX2004* mit Java zu erzeugen.

3.2 Scalable Vector Graphics

Scalable Vector Graphics ist wie der Name bereits sagt ebenfalls ein Vektorgrafikformat. Bereits seit Februar 1999 wurde beim *World Wide Web Consortium (W3C)* [W3C1] unter Beteiligung verschiedener Firmen (darunter Macromedia, Adobe und Microsoft) an der Spezifikation von SVG gearbeitet, aber erst am fünften September 2001 wurde die erste Empfehlung (engl. recommendation) mit dem Namen *Scalable Vector Graphics (SVG) 1.0* veröffentlicht [W3C5]. Im Januar 2003 wurde die Version 1.1 als Empfehlung veröffentlicht. Zur Zeit ist die Version 1.2 in der Diskussion. Es wird unter anderem überlegt, SVG mit Streaming Mechanismen zu versehen und eine Unterstützung von

multimedialen Inhalten wie Audio oder Video festzuschreiben. Außerdem sollen editierbare Textfelder eingeführt werden [W3C4]. SVG liegt in unterschiedlichen Profilen vor. So gibt es ein umfassendes Profil für Desktopgeräte und ein Profil für mobile Endgeräte, namens *SVG Tiny*.

SVG wurde vom W3C auf der Basis von XML entwickelt. Hierin liegt bereits einer der großen Vorteile von SVG. Da SVG eine XML Ausprägung ist, lässt sich das Format leicht lesen und schreiben. Dies gilt sowohl für die Entwicklung per Hand, als auch für die automatische Generierung von SVG mittels Programmen. Letzteres wird durch standardisierte APIs unterstützt, die dazu dienen, XML zu lesen und zu schreiben. Analog zum Flashstudio von Macromedia gibt es auch für SVG Entwicklertools, sowohl kommerzielle als auch Open Source Produkte. Hervorzuheben ist für SVG das Open Source Programm Inkscape [Inkscape], mit dem es möglich ist, komplexe Grafiken in SVG zu definieren. Allerdings bietet Inkscape nicht die Möglichkeit mittels einer Zeitleiste Filme zu erstellen, sondern lediglich die Möglichkeit, einzelne Grafiken zu erstellen.

Um SVG im Webbrowser betrachten zu können, ist meist ein Plugin notwendig. Das umfassendste Plugin ist von Adobe entwickelt worden und nennt sich *Adobe SVG Viewer (ASV)* [Adobe2]. Dieses Plugin ist für verschiedene Browser unter verschiedenen Betriebssystemen einsetzbar. Seit einiger Zeit gibt es aber auch Webbrowser, die die SVG Unterstützung gleich mitliefern. So kann Opera ab der Version 8 SVG Grafiken, die dem SVG Tiny Profil genügen, anzeigen. Firefox ab der Version 1.5 kann viele Elemente des SVG 1.1 Standards anzeigen und auch das Scripting mittels ECMAScript funktioniert, so dass für den Firefox komplexe SVG Anwendungen entwickelt werden können, ohne dass ein Plugin für den Webbrowser benötigt wird. Eine aktuelle und sehr gute Übersicht über die SVG Fähigkeit unterschiedlicher Webbrowser ist in [Wikipedia1] zu finden.

Neben der Möglichkeit, SVG im Webbrowser via Plugin zu betrachten, kann auch ein Standalone Viewer verwendet werden. Hier hat sich der *Batik-Viewer* [Batik] von der *Apache Foundation* [Apache] durchgesetzt. Es handelt sich in erster Linie um ein Java API, mit dem es möglich ist, Anzeigekomponenten für SVG zu definieren und in eigene Applikationen einzubinden. Es wird aber auch ein SVG Browser mitgeliefert, der als Demonstration dient, wie man mit dem API eine komplexe Anwendung implementiert.

Komplexe SVG Anwendungen arbeiten neben der einfachen Darstellung von grafischen Objekten mit der Manipulation der SVG Grafik mittels ECMAScript. Die *ECMA International* (European association for standardizing information and communication systems) [ECMA1] ist eine private Normungsorganisation im Bereich Computersysteme und hat Netscapes JavaScript und Microsofts JScript vereinheitlicht und daraus den Standard ECMAScript [ECMA2] (im allgemeinen als JavaScript bezeichnet) gebildet. ECMAScript bietet ein eventbasiertes Modell, so dass auf Benutzereingaben reagiert werden kann. Da SVG XML-basiert ist, kann eine SVG Datei als *Document Object Model (DOM)* aufgefasst werden. Hierbei handelt es sich um eine Baumstruktur, in der alle Elemente der SVG Datei enthalten sind (siehe Kapitel 4.4.1). Mittels ECMAScript kann man dieses DOM manipulieren. So können einzelne Knoten des DOM gelöscht, verändert oder hinzugefügt werden, was beliebige Interaktionen und Veränderungen des SVG Dokumentes zur Laufzeit ermöglicht (Kapitel 4.4).

Ebenso wie mit Flash in der Kombination mit Flex ist es in SVG möglich, Rich Internet Applications zu entwickeln. Auch hier kann man eine SVG Anwendung zunächst laden und bei Bedarf werden einzelne Elemente nachgeladen, so dass der Benutzer das Gefühl hat, er arbeite mit einer Desktopapplikation.

3.3 SVG vs. Flash

Die beiden Formate SVG und Flash profitieren von der Vektorgrafik Technologie. Die Grafiken sind frei skalierbar, platzsparend und einfach zu animieren. Dennoch weisen beide Formate starke Unterschiede auf [Winter]. Viele davon beruhen auf der Speicherung der Daten. Bei Flash werden die Daten binär und bei SVG in XML abgespeichert. Dadurch ist SVG wesentlich einfacher zu erzeugen als Flash, für das spezielle APIs nötig sind. SVG kann man sogar von Hand schreiben, um kleinere Beispiele zu erzeugen, bei Flash ist dies nicht möglich. Dies wird sich auch in Zukunft nicht ändern, da früher Macromedia oder jetzt Adobe die Software zur Erzeugung von Flash kommerziell vertreibt und mit Gewinneinbußen zu rechnen hätte, wenn das Format leichter zu erzeugen wäre.

Die einfache Erzeugung von SVG führt auch zu anderen Einsatzgebieten. Flash wird oftmals zur multimedialen Aufbereitung von Webseiten eingesetzt und weniger als wirklich sinnvolle ernstzunehmende Anwendung. SVG hingegen wird gerade in der Geographie und auch allgemein zur Datenvisualisierung eingesetzt. SVG hat dadurch ein seriöseres Image als Flash, welches oft mit grellen und überflüssigen Animationen in Verbindung gebracht wird.

Ein weiterer Unterschied, der auf dem Dateiformat beruht, ist die Indizierbarkeit von Flash und SVG Inhalten. Bei Flash kann eine Suchmaschine den Inhalt nicht indizieren. Folglich sind die Inhalte auch nicht über eine Suchmaschine zu finden. SVG hingegen besteht aus lesbarem XML, was das Indizieren erlaubt. Allerdings muss gesagt werden, dass SVG Dateien zu `.svgz` Dateien mit dem ZIP-Algorithmus komprimiert werden können. Dies stellt aber prinzipiell kein Hindernis dar, da Suchmaschinen ZIP gepackte Dateien leicht entpacken und somit trotzdem indizieren können.

Sowohl Flash als auch SVG können durch Skriptsprachen manipuliert werden. Bei Flash erfolgt dies durch die von Macromedia entwickelte Sprache ActionScript welche aber stark an JavaScript angelehnt ist. Bei SVG können hingegen verschiedene Programmiersprachen zur Manipulation verwendet werden. Clientseitig wird bei SVG JavaScript eingesetzt (siehe Kapitel 4.4). Prinzipiell können SVG Dokumente aber wie alle anderen XML basierenden Dokumente mit jeder Programmiersprache erzeugt und manipuliert werden, für die ein DOM API vorhanden ist. Dadurch ist die serverseitige Manipulation von SVG Dokumenten wesentlich einfacher als bei Flash Dateien.

Barrierefreiheit ist momentan ein sehr aktuelles Thema. So sollen Webseiten für sehbehinderte Menschen zugänglich gemacht werden. Auch mit minimalen technischen Mitteln, wie z.B. einem Textbrowser, soll eine Webseite lesbar bleiben. Wird Flash für eine Webseite eingesetzt ist die Barrierefreiheit nicht gegeben. So ist bei Flash der enthaltene Text nicht mehr als solcher erkennbar, sondern wird nur als Grafik angesehen. Bei SVG

hingegen bleibt der Text enthalten, selbst wenn er mit Filtereffekten versehen oder auf andere Art und Weise verändert wird. Daher kann die textuelle Information weiterhin verarbeitet werden und auch sehbehinderten Menschen zugänglich gemacht werden. Generell wird bei SVG auf den barrierefreien Zugang Wert gelegt und das W3C betont den einfachen Zugang zu SVG Dokumenten [W3C2].

SVG weist gegenüber Flash aber auch einige Nachteile auf. So ist ein SVG Dokument zur Zeit noch nicht streambar. Ein Flashfilm hingegen kann bereits abgespielt werden, ohne dass er komplett geladen wurde. Dies ist ein großer Vorteil von Flash. Allerdings ist es sehr einfach möglich, in SVG Inhalte nachzuladen oder Inhalte bereits vor der Verwendung zu laden (Precaching), so dass bei einer geschickten Aufteilung einer SVG Anwendung die mangelnde Streamingfähigkeit nicht ins Gewicht fällt.

Außerdem unterstützt Flash vielfältige multimediale Inhalte wie Audio und Video. Bei SVG können Audiodateien nur über eine Namespaceerweiterung eingebunden und ausschließlich mit dem Adobe SVG Viewer abgespielt werden. Eine Videounterstützung gibt es bei SVG in der Version 1.1 noch nicht.

Ein schwerwiegender Nachteil von SVG ist die mangelnde Verbreitung von Anzeigekomponenten. Die Plugins und Standaloneviewer implementieren meist nicht den gesamten SVG Umfang und sind generell nur auf relativ wenigen Systemen installiert. Leider gibt es keine offiziellen Studien, die sich mit der Verbreitung von SVG beschäftigen. Da jetzt aber die ersten Webbrowser SVG nativ beherrschen wird sich die Zahl der Nutzer mit SVG Unterstützung schnell erhöhen. Kritischer Faktor ist der Internet Explorer von Microsoft. Falls er SVG nativ darstellen könnte, würde die Verbreitung von SVG und damit auch die Zahl der SVG Anwendungen sprunghaft steigen. Allerdings scheint Microsoft wie auch bei anderen Technologien einen eigenen Weg zu gehen und nur einen SVG Dialekt nativ zu unterstützen. Daher wird es für den Internet Explorer wahrscheinlich weiterhin notwendig sein, ein Plugin wie den Adobe SVG Viewer zu installieren. Aber auch wenn die Verbreitung von SVG nur zögernd vorangeht, ist das Format gerade im GIS Bereich bereits stark vertreten.

4 SVG als interaktives Darstellungsformat

SVG ist XML-basiert und profitiert von den positiven Eigenschaften von XML. So ist SVG einfach per Software und per Hand zu erstellen. Ein einfacher Texteditor reicht aus, um eine einfache SVG Datei zu erzeugen. Selbst etwas komplexere Grafiken können per Hand erstellt werden. Komplexere SVG Anwendungen lassen sich sehr einfach mit Skripten oder Programmen erzeugen, da XML durch standardisierte APIs gelesen und geschrieben werden kann. Ein weiterer Vorteil von SVG ist die klare und strukturierte Definition einzelner Elemente und die einfache Überprüfung mit standardisierten Parsern, ob eine Datei wohlgeformt und valide ist.

4.1 SVG Grundgerüst und Definition von geometrischen Elementen

Wie jede XML Datei enthält auch eine SVG Datei einen XML-Prolog mit der XML Version und einem Verweis auf die *Doctype Definition (DTD)*. Im Wurzelelement `svg` können mit den Attributen `width` und `height` Angaben zur Größe der SVG Grafik gemacht werden (Quellcode 4.1).

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
    "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg height="600" width="800">
</svg>
```

Quellcode 4.1: Grundgerüst eines SVG-Dokumentes

Innerhalb des Wurzelelementes kann in einem `defs`-Tag ECMAScript für die Interaktion mit dem Benutzer angegeben und Prototypen und Patterns definiert werden. Danach folgt der Aufbau der SVG Grafik. Alle graphischen Objekte in SVG bauen auf einfachen grafischen Primitiven auf. Komplexere Objekte werden aus den einfacheren zusammengesetzt. Im Folgenden werden die in SVG vorhandenen Grundtypen kurz erläutert. Nähere Erläuterungen sind in [W3C7], [Eisenberg] und [Watt] zu finden.

Der Pfad

Der Pfad ist das Grundelement in SVG mit dem Kreise, Rechtecke, Polygone usw. aufgebaut werden können. Zur Vereinfachung wurden aber häufig vorkommende Objekte mit eigenen Beschreibungen versehen.

Der Pfad ist eine Sammlung von Koordinatenpaaren, die im Attribut `d` entweder absolut oder relativ zueinander definiert werden können. Dabei gibt es für die Interpretation der Koordinatenpaare verschiedene Anweisungen, die als Buchstabe vorangestellt werden. Großbuchstaben leiten dabei absolute und Kleinbuchstaben relative Koordinatenpaare ein.

- M/m *moveto* = Den Zeichenstift an diese Stelle bewegen
- L/l *lineto* = Mit dem Stift dorthin zeichnen
- Q/q = Quadratische Bézierkurve (zwei Wertepaare: ein Stützpunkt, ein Zielpunkt)
- C/c = Kubische Bézierkurve (drei Wertepaare: zwei Stützpunkte, ein Zielpunkt)

In der Abbildung 4.1 ist die grafische Darstellung für den folgenden Quellcode zu sehen.

```
<path fill="none" stroke="black"
      d="M 10,80 L 160,80 L 160,10
         C 160,10 160,45 80,45
         C 10,45 10,10 6,10
         L 6,80 L 10,80 z "/>
```

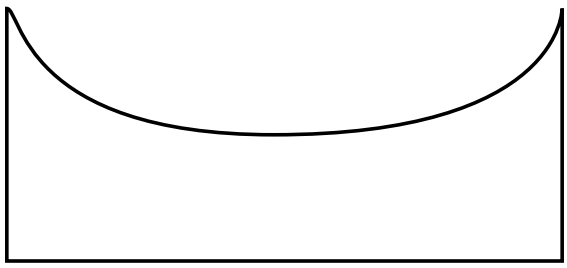


Abbildung 4.1: Pfad in SVG

Neben den oben genannten Anweisungen gibt es noch vereinfachte Anweisungen für waagerechte und senkrechte Linien. Für Ellipsen und Kreise ist eine spezielle Notation mit sieben Werten erforderlich.

- H/h *horizontal line* = horizontale Linie (nur X-Koordinate erforderlich)
- V/v *vertikal line* = vertikale Linie (nur Y-Koordinate erforderlich)
- A/a *elliptical arc* = Bogenkurve (Angabe über zwei Radien, einem X-Achsen Rotationswinkel, dem Zentrum und zwei Parametern zur Zeichenrichtung und Orientierung der Kurve)

Die im folgenden Quellcode definierten Kreissegmente sind in Abbildung 4.2 zu sehen.

```
<path fill="red" stroke="blue" stroke-width="5"
      d="M 300,200 h -150 a 150,150 0 1,0 150,-150 z"/>

<path fill="yellow" stroke="blue" stroke-width="5"
      d="M 275,175 v -150 a 150,150 0 0,0 -150,150 z"/>
```

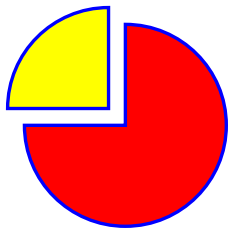


Abbildung 4.2: Pfad mit Kreissegmenten in SVG

Geometrische Grundprimitive

Zur einfacheren Erstellung einer SVG Grafik gibt es die Grundprimitive Kreis, Ellipse, Rechteck und Linie.

Ein Kreis wird durch seinen Radius und seinen Mittelpunkt definiert. Der Mittelpunkt wird in den Attributen `cx` und `cy` festgelegt, der Radius über das Attribut `r`.

```
<circle cx="100" cy="100" r="50"
        fill="blue" stroke="black"/>
```

Eine Ellipse wird durch ihren Mittelpunkt (`cx` und `cy`) und zwei Halbachsradien (`rx` und `ry`) definiert.

```
<ellipse cx="250" cy="100" rx="50" ry="20"
         fill="red" stroke="black"/>
```

Ein Rechteck wird durch die Angabe der linken oberen Ecke, der Breite und der Höhe definiert.

```
<rect x="200" y="150" width="70" height="50"
      fill="lime" stroke="black"/>
```

Eine Linie wird durch ihren Anfangs- und Endpunkt beschrieben.

```
<line x1="100" y1="100" x2="200" y2="200"
      stroke="black"/>
```

Die vier Objekte sind in Abbildung 4.3 dargestellt.

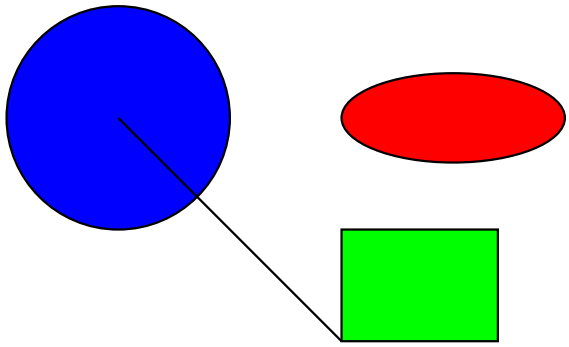


Abbildung 4.3: Geometrische Grundprimitive in SVG

Komplexe Linienzüge

Neben den genannten Grundprimitiven gibt es die Möglichkeit, frei definierte Polygone (Vielecke) und Polylinien zu erzeugen.

Eine Polylinie wird über mehrere Punkte definiert und ist nicht geschlossen. Die Punkte werden als Koordinatenpaare fortlaufend aneinander gehängt (Abbildung 4.4).

```
<polyline fill="none" stroke="blue" stroke-width="10"
          points="50,375 150,375 150,325 250,325 250,375
                  350,375 350,250 450,250 450,375
                  550,375 550,175 650,175 650,375"/>
```

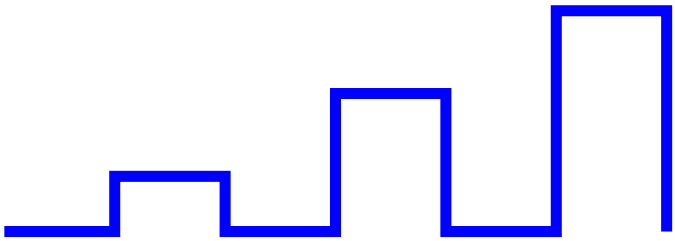


Abbildung 4.4: Eine Polylinie in SVG

Ein Polygon wird wie die Polylinie definiert und wird immer automatisch geschlossen (Abbildung 4.5).

```
<polygon fill="red" stroke="blue" stroke-width="10"
         points="350,75  379,161 469,161 397,215
                 423,301 350,250 277,301 303,215
                 231,161 321,161" />
<polygon fill="lime" stroke="blue" stroke-width="10"
         points="850,75  958,137.5 958,262.5
                 850,325 742,262.6 742,137.5" />
```

Abbildung 4.5: Polygone in SVG

Text

Neben den Elementen zum Zeichnen von geometrischen Objekten kann auch Text angelegt werden. Der Text kann frei positioniert und mit Fonteigenschaften versehen werden. Außerdem kann er anhand eines Pfades ausgerichtet werden.

```
<defs>
  <path d="M 100,200 C 200,100 300,0 400,100
                     C 500,200 600,300 700,200
                     C 800,100 900,100 900,100"
        style="fill:none" id="MyPath" />
</defs>
<use style="stroke:red" x="0" y="0"
     width="1000" height="300" xlink:href="#MyPath"/>

<text style="font-size:42.5px;fill:blue;
             font-family:Verdana">
  <textPath xlink:href="#MyPath">
    Erst nach oben,dann runter und wieder hoch
  </textPath>

</text>
<text x="452.38095" y="83.333328" style="font-size:40px"
      xml:space="default">
  Textpfad in SVG
</text>
```

Das Ergebnis ist in Abbildung 4.6 zu sehen.

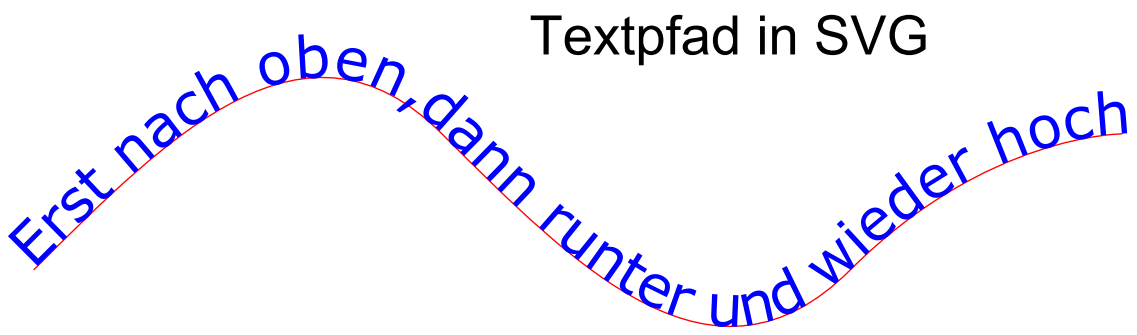


Abbildung 4.6: Text in SVG

4.2 Füllung von Flächen

Eine Fläche kann sehr einfach über das Attribut `fill` mit einer Füllfarbe versehen werden. Dabei kann der Farbwert als RGB Tripel oder über einen Namen bestimmt werden. Ein Wert des RGB Tripels besteht entweder aus einer Dezimalzahl aus dem Wertebereich $[0, 255]$ oder aus einer Hexadezimalzahl aus dem Intervall $[00, ff]$. Die Benennung der Farben stellt eine Erweiterung der bei *Cascading Style Sheets* definierten Namen dar. Eine Übersicht aller erlaubten Farbnamen ist unter [W3C3] zu finden.

Um das Füllverhalten genauer zu spezifizieren, kann im Attribut `fill-rule` eine Füllregel angegeben werden. Wird als Wert `evenodd` angegeben, werden nur Flächen gefüllt die durch einen *Point in Polygon* Algorithmus [Fellner] als innen liegend erkannt wurden. Dazu wird von einem Punkt der Fläche ausgehend ein waagerechter Strahl nach rechts ausgestrahlt und die Schnittpunkte mit dem Polygon gezählt (Abbildung 4.7).

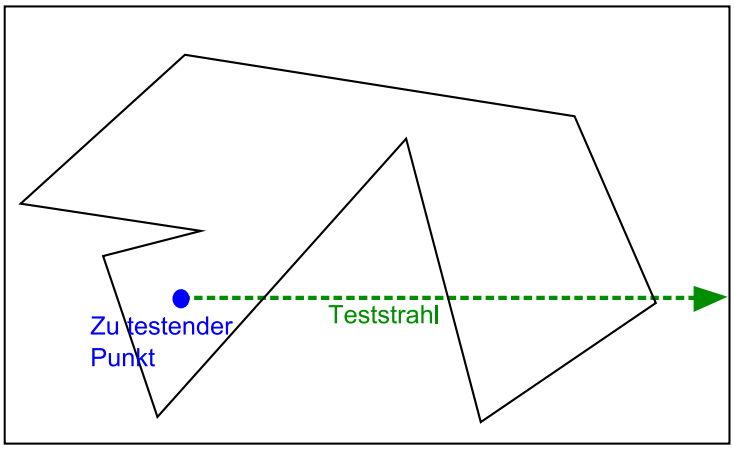
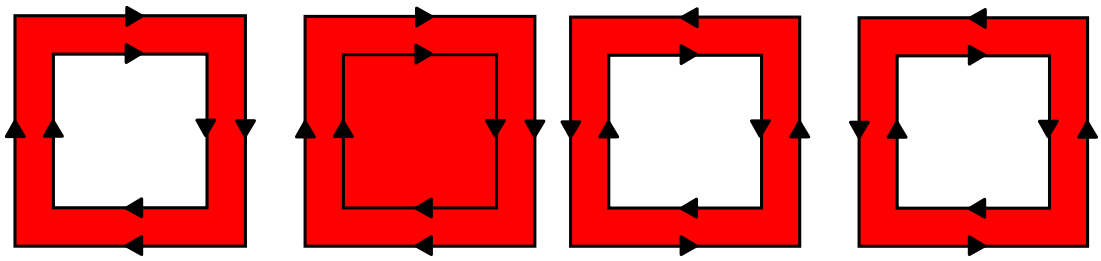


Abbildung 4.7: Point in Polygon

Ist die Anzahl der Schnittpunkte ungerade, wird die Fläche gefüllt, sonst nicht. Die Umlaufrichtung des Polygons spielt also keine Rolle. Bei der Füllregel `nonzero` ist die Umlaufrichtung entscheidend. Auch hier wird ein waagerechter Strahl ausgeschickt und die Schnittkanten gezählt, allerdings werden nur die Kanten einer Umlaufrichtung aufaddiert. Die Schnittkanten mit der anderen Umlaufrichtung werden subtrahiert. Ist der Wert ungleich null, wird an dem Punkt, von dem die Waagerechte ausgeschickt wurde, die Füllfarbe gesetzt (Abbildung 4.8).



(a) Regel: evenodd (b) Regel: nonzero (c) Regel: evenodd (d) Regel: nonzero

Abbildung 4.8: Füllregeln in SVG

Neben der Füllung eines Polygons mit einem Farbwert, gibt es auch die Möglichkeit ein Polygon mit einer Textur, einem so genannten Pattern, zu füllen, wie es im nächsten Abschnitt beschrieben wird.

4.3 Verwendung von Prototypen und Patterns

In einem `defs`-Element können in SVG unter anderem Prototypen und Patterns definiert werden. Bei einem Prototypen handelt es sich um ein geometrisches Objekt, welches einmalig im `defs`-Element definiert wird und beliebig oft wieder verwendet werden kann. So können beispielsweise Symbole definiert werden, die bei Bedarf referenziert werden können. Die Prototypen können bei Verwendung noch um zusätzliche Eigenschaften erweitert werden. So können Attribute, die im der Prototypendefinition keinen Wert erhalten haben, nachträglich mit Werten versehen werden wie in Quellcode 4.2 zu sehen. Dabei wird die Linienfarbe gesetzt und die Prototypen werden mit dem Attribut `transform` positioniert. Dazu werden die Shapes erst rotiert und dann translatiert. Das Ergebnis ist in Abbildung 4.9 dargestellt.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
    "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
  <svg xmlns="http://www.w3.org/2000/svg"
       xmlns:xlink="http://www.w3.org/1999/xlink">
    <defs>
      <g id="wind">
        <path d="M 20,0 L -20,0 L -25,-16" stroke-width="2"/>
        <path d="M -15,0 L -20,-16" stroke-width="2"/>
        <path d="M -10,0 L -15,-16" stroke-width="2"/>
        <path d="M -5,0 L -10,-16" stroke-width="2"/>
      </g>
    </defs>
    <use xlink:href="#wind" stroke="red"
         transform="translate(50,50)"/>
    <use xlink:href="#wind" stroke="green"
         transform="translate(100,50) rotate(45)"/>
    <use xlink:href="#wind" stroke="blue"
         transform="translate(150,50) rotate(90)"/>
  </svg>
```

Quellcode 4.2: Definition und Verwendung eines Prototypen

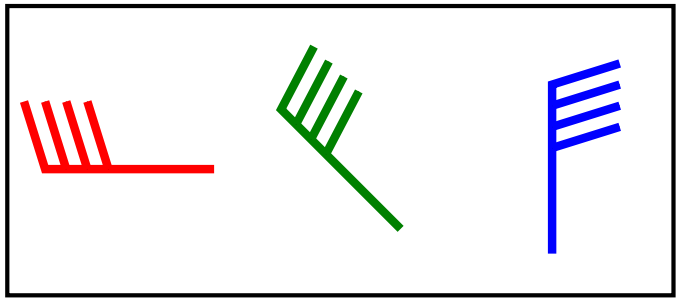


Abbildung 4.9: Verwendung von Prototypen

Ein Pattern wird ebenfalls im `defs`-Tag angegeben. Es wird verwendet, um ein Objekt mit einer Struktur zu füllen, ähnlich wie einer Textur. Dazu werden, wie in Quellcode 4.3 zu sehen, innerhalb des `pattern`-Elementes beliebige Shapes verwendet. Das Pattern selbst besitzt eine eindeutige ID und ein eigenes Koordinatensystem welches im Attribut `viewBox` angegeben wird. Die Attribute `x`, `y`, `width` und `height` geben an, wie groß das Pattern bei der Verwendung wird. Die Angabe `userSpaceOnUse` sorgt dafür, dass die Patterns die definierte Größe beibehalten. Die so definierten Patterns können dann einem Objekt als `fill`-Attribut übergeben werden. Das Ergebnis ist in Abbildung 4.10 zu sehen.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
    "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
  <svg xmlns="http://www.w3.org/2000/svg"
       xmlns:xlink="http://www.w3.org/1999/xlink">
    <defs>
      <pattern viewBox="0 0 100 100" id="muster1"
               patternUnits="userSpaceOnUse"
               x="0" y="0" width="30" height="30">
        <line x1="90" y1="10" x2="10" y2="90"
              stroke="blue" stroke-width="5"/>
      </pattern>
      <pattern viewBox="0 0 100 100" id="muster2"
               patternUnits="userSpaceOnUse"
               x="0" y="0" width="30" height="30">
        <circle cx="50" cy="50" r="30" fill="red"/>
      </pattern>
    </defs>
    <rect x="10"  y="10" width="100" height="100"
          fill="url(#muster1)" stroke="black" stroke-width="1"/>
    <rect x="120" y="10" width="100" height="100"
          fill="url(#muster2)" stroke="black" stroke-width="1"/>
</svg>
```

Quellcode 4.3: Definition und Verwendung eines Patterns

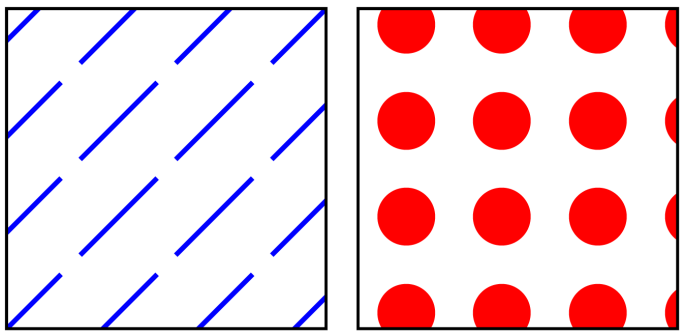


Abbildung 4.10: Verwendung von Patterns

4.4 Scripting in SVG

Interaktionen in SVG zu erstellen, ist mit ECMAScript (auch JavaScript genannt) relativ einfach. Mittels JavaScript ist es möglich, einzelne Attribute zu verändern, neue Elemente zu erschaffen oder mit einem Server zu kommunizieren. Dabei kann auf verschiedene Events reagiert werden. Die Skripte können entweder innerhalb des SVG Dokumentes oder aber außerhalb platziert werden. Im letzteren Fall wird das Skript von der SVG Datei aus referenziert. Auslagern sollte man alle Skripte, sobald sie aus mehreren Funktionen bestehen, da so die Übersichtlichkeit und Wartbarkeit besser gegeben ist. Außerdem sollten sehr komplexe Skripte in logische Einheiten separiert werden und in einzelnen Dateien abgelegt werden (Kapitel 4.6). Dadurch sind bestimmte Programmteile schnell zu finden und das gesamte Projekt wird übersichtlicher, analog zur Objektorientierten Programmierung, bei der auch logisch zusammenhängende Funktionen in einzelnen Klassen abgelegt werden. Um die Vorgehensweise näher zu erläutern, sollen zunächst einige Begriffe erläutert werden und im Weiteren werden Beispiele gezeigt, um das Prinzip der SVG Manipulation zu verdeutlichen.

4.4.1 Document Object Model (DOM)

Um ein SVG Dokument zur Laufzeit verändern zu können, wird das DOM verwendet. Dabei liegt das Dokument in Form einer Baumstruktur, dem so genannten DOM-Tree, im Speicher vor (Abbildung 4.11).

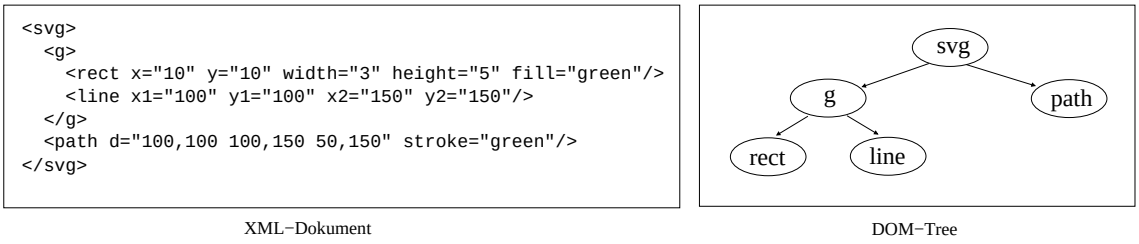


Abbildung 4.11: Baumstruktur eines XML Dokumentes

Jedes Element stellt dabei einen Knoten im DOM-Tree dar. In einem DOM existieren verschiedene Arten von Knoten:

- Dokumentknoten: Referenz für das gesamte Dokument
- Elementknoten: Ein Element in HTML oder XML
- Textknoten: Textueller Inhalt eines Elements oder Attributs

Wird ein Knoten verändert, wird die SVG Grafik entsprechend neu dargestellt. Ebenso können einzelne Knoten neu erzeugt oder auch gelöscht werden. Um diese Manipulationen vornehmen zu können, gibt es das DOM API, also eine Programmierschnittstelle, für XML und HTML Dokumente. Das API wurde vom W3C [W3C1] im Jahre 1998 [W3C6]

definiert und im Laufe der Jahre stets weiterentwickelt. Das DOM API erlaubt es, durch die einzelnen Knoten zu navigieren, Knoten zu erzeugen, zu löschen, zu verschieben oder zu kopieren. Außerdem können Attributwerte gelesen, verändert oder gelöscht werden.

Das API wird von vielen verschiedenen Programmiersprachen implementiert um XML Dokumente zu erzeugen und zu verändern. Im Internet Bereich ist das JavaScript DOM API von besonderer Bedeutung. Fast jeder Webbrowser besitzt eine JavaScript Engine und implementiert das DOM API, so dass beliebige im Webbrowser angezeigte XML Dokumente (XHTML Seiten, SVG Dokumente, etc.) im Webbrowser manipuliert werden können.

4.4.2 DOM Manipulationen

Um das Erscheinungsbild eines SVG Dokumentes zur Laufzeit zu verändern, kann über das DOM API die Struktur des Dokumentes und einzelne Attribute verändert werden. Dazu muss zuerst das zu verändernde Element im DOM-Tree gesucht werden. Der DOM-Tree kann in JavaScript über die Variable `document` angesprochen werden. Mit der Funktion `getElementById` wird im SVG Dokument nach einem Element mit der als Parameter übergebenen ID gesucht. Wurde das Element gefunden, wird der Knoten zurückgegeben. Mit der Funktion `setAttributeNS` kann dann ein Attributwert gesetzt werden.

Im Quellcode 4.4 wird bei einem Klick auf den Kreis die Funktion `changeColor` aufgerufen, die die Füllfarbe des Rechteckes auf `blue` setzt.

```
<svg>
  <script type="text/ecmascript">
  <![CDATA[
    function changeColor() {
      rectangle=document.getElementById('rect');
      rectangle.setAttributeNS(null,'fill',blue);
    }
  \\]]>
  </script>

  <rect x="10" y="10" width="30" height="50" id="rect"/>
  <circle cx="5" cy="5" r="3" fill="red"
          onclick="changeColor()"/>
</svg>
```

Quellcode 4.4: Ändern eines Attributwertes mit JavaScript

Auf diese Art und Weise können alle Attributwerte einfach manipuliert werden. Für die Erzeugung eines neuen Elementes ist die Funktion `createElementNS` zuständig. Um die Vorgehensweise zu erläutern, soll in einem leeren SVG Dokument ein roter Kreis erzeugt werden (Quellcode 4.5). Dazu wird beim Laden des Dokumentes (`onload`, siehe Kapitel 4.4.3) die Funktion `init` aufgerufen. Zunächst wird das Element `circle` mit

dem passenden Namespace erzeugt und wird dann mit der Funktion `setAttributeNS` mit den gewünschten Attributen versehen. Dieses Element gehört zwar logisch zum DOM-Tree, ist aber noch nicht in den Baum eingehängt und wird daher noch nicht dargestellt. Dazu muss es erst mit der Funktion `appendChild` an einen Knoten im SVG Dokument eingehängt werden.

```
<svg onload="init()" id="root">
  <script type="text/ecmascript">
  <![CDATA[
    var xmlns='http://www.w3.org/2000/svg';
    function init() {
      circle=document.createElementNS(xmlns,'circle');
      circle.setAttributeNS(null,'cx','10');
      circle.setAttributeNS(null,'cy','10');
      circle.setAttributeNS(null,'radius','7');
      circle.setAttributeNS(null,'fill','red');
      document.getElementById('root').appendChild(circle);
    }
  \\]]>
  </script>
</svg>
```

Quellcode 4.5: Erzeugen eines Elementes mit JavaScript

Um ein Element zu entfernen wird die Funktion `removeChild` verwendet. Allerdings muss dazu der Elternknoten des zu entfernenden Elementes bekannt sein. Ist die ID des zu entfernenden Elementes bekannt, muss dieser zunächst nach seinem Elternknoten gefragt werden, der über das Datenfeld `parentNode` erreichbar ist. In der Funktion `removeElement` (Quellcode 4.6) ist die Vorgehensweise zu sehen.

```
function removeElement(name){
  var child=document.getElementById(name);
  if(child!=null){
    parent=child.parentNode;
    parent.removeChild(child);
  }
}
```

Quellcode 4.6: Entfernen eines Elementes mit JavaScript

4.4.3 Eventverarbeitung in JavaScript

Für eine interaktive SVG Anwendung ist es notwendig, auf Benutzereingaben zu reagieren. Dazu kann ein SVG Element mit verschiedenen Attributen versehen werden, mit denen ein Eventhandler registriert werden kann. Der Eventhandler stellt die Verbindung zwischen dem SVG Dokument und den JavaScript-Funktionen her. Der Attributname beginnt immer mit `on` und gibt die Art des Events an. Der Wert des Attributes benennt die

aufzurufende JavaScript-Funktion. Folgende Events sind für diese Arbeit von besonderer Bedeutung:

- `onclick`: Klicken der Maus auf das Element
- `onload`: Das Dokument ist geladen
- `onmousedown`: Drücken der linken Maustaste
- `onmouseup`: Loslassen der linken Maustaste
- `onmousemove`: Bewegen der Maus
- `onmouseover`: Überfahren des Elementes mit der Maus

In Quellcode 4.7 ist zu sehen, wie ein Eventlistener definiert wird. Dazu wird im Attribut `onclick` angegeben, was bei einem Klick auf das Rechteck passieren soll. In diesem Fall wird eine Fehlernachricht erstellt und angezeigt.

```
<rect x="100" y="100" width="30" height="50"
      onclick="alert('Klick auf das Rechteck');"/>
```

Quellcode 4.7: Definition eines `onclick`-Events

In der reservierten Variablen `evt` wird das Objekt des auslösenden Eventes gespeichert. Wie in Quellcode 4.8 zu sehen, kann dem `onclick`-Attribut eine JavaScript-Funktion übergeben werden, die mit dem Parameter `evt` versehen ist. Hierbei handelt es sich um das Event-Objekt, welches dann in der aufgerufenen Funktion beispielsweise nach dem auslösenden Element befragt werden kann.

```
<svg id="root">
  <script type="text/ecmascript">
  <![CDATA[
    var xmlns='http://www.w3.org/2000/svg';

    function changeColor(evt) {
      source = evt.getTarget();
      source.setAttributeNS(null,'fill','red');
    }
  \\]]>
  </script>
  <rect x="10" y="10" width="30" height="50"
        fill="blue" onclick="changeColor(evt)"/>
  <circle cx="5" cy="5" r="3"  fill="blue"
        onclick="changeColor(evt)">
</svg>
```

Quellcode 4.8: Verarbeiten des Event Objektes

4.5 Nachladen von SVG

Bei der Visualisierung umfangreicher Daten können nicht alle Daten in einem SVG Dokument enthalten sein, da sonst der Arbeitsspeicher des Clients zu stark belastet werden würde. Daher ist es notwendig, zur Laufzeit Daten nachzuladen. Das Nachladen von Daten wird über JavaScript bewerkstelligt, dabei sind zwei Arten des Nachladens zu unterscheiden. Der Adobe SVG Viewer verwendet die beiden nicht im ECMAScript Standard definierte Funktionen `getUrl` und `parseXML`. In Webbrowsern ohne den Adobe SVG Viewer aber mit nativer SVG Unterstützung stehen diese Funktionen nicht zur Verfügung. Hier muss mit dem `XMLHttpRequest`-Objekt eine Verbindung zu einem Server aufgebaut werden, um Daten zu laden.

Beide Verfahren laden die Daten asynchron nach. Werden Daten von einem Server angefordert, muss der Client nicht warten bis diese Daten geladen sind und bleibt nicht blockiert bis die Daten empfangen wurden. Dadurch kann mit der Anwendung weiter gearbeitet werden, obwohl noch nicht alle Daten geladen wurden. Das Konzept des asynchronen Nachladens von Daten mittels XMLHttpRequest, die Veränderung eines DOM-Trees und die Verwendung von JavaScript werden zusammen als *Ajax-Technologie (Asynchronous JavaScript and XML)* bezeichnet, wie es Jesse James Garret, der den Begriff geprägt hat, in einem der ersten Artikel zum Thema beschreibt [Garret]. Da XMLHttpRequest vom Adobe SVG Viewer nicht unterstützt wird, kann hier genau genommen nicht von einem Ajax Konzept gesprochen werden, aber die grundlegende Vorgehensweise ist gleich.

4.5.1 Die Funktionen `getUrl` und `parseXML`

In der Funktion `datenLaden` in Quellcode 4.9 ist beispielhaft zu sehen, wie Daten von einem Server geladen und in das SVG Dokument eingehängt werden können.

```
function datenLaden(url,parent) {

  getURL(url, callBack);

  function callBack(data) {
    if(data.success) {
      st = data.content;
      var node = parseXML(st, document);
      document.getElementById(parent).appendChild(node);
    }
  }
}
```

Quellcode 4.9: Verwendung von `getURL` und `parseXML`

Die Funktion `getURL` wird mit der URL der Datei und einer Callbackfunktion aufgerufen. Wurden die Daten erfolgreich übertragen, werden die geladenen Daten an die

Callbackfunktion zur weiteren Verarbeitung übergeben. War das Laden erfolgreich wird das Ergebnis in der Variablen `st` abgespeichert. Die in `st` vorhandenen Daten müssen nun als XML interpretiert und dem SVG Dokument verfügbar gemacht werden. Dazu wird die Funktion `parseXML` mit den angelieferten Daten und einer Referenz auf das SVG Dokument aufgerufen. Ergebnis ist ein dem SVG Dokument zugehöriger Knoten, der dann mit der Funktion `appendChild` an das gewünschte Element gehängt wird.

4.5.2 XMLHttpRequest

XMLHttpRequest ist ein API zum Transfer von XML Daten über das HTTP Protokoll. Es können verschiedene Serveranfragen gestellt werden, wie `GET`, `POST`, `HEAD` oder `PUT`. Die empfangenen Daten können entweder als reiner Text oder als DOM-Tree interpretiert werden. XMLHttpRequest kann in JavaScript, JScript und VBScript verwendet werden und stellt einen Grundbestanteil der Ajax-Technik dar. Sie kommt bei allen Webbrowsern zum Einsatz, die SVG nativ unterstützen.

Quellcode 4.10 ist ein Beispiel, wie mit dem `XMLHttpRequest`-Objekt Daten geladen werden können.

```
var xmlRequest=null;

function getdata(url) {
  xmlRequest = new XMLHttpRequest();
  xmlRequest.open("GET",url,true);
  xmlRequest.onreadystatechange = callback;
  xmlRequest.send(null);
}
function callback() {
  if (xmlRequest.readyState == 4) {
    if (xmlRequest.status == 200) {
      var loadedNode = xmlRequest.responseXML.documentElement;
      var importedNode = document.importNode(loadedNode,true);
      document.rootElement.appendChild(importedNode);
    }
  }
}
```

Quellcode 4.10: Serveranfrage mit XMLHttpRequest

Dazu wird zunächst das Objekt erzeugt und mit der Funktion `open` werden die Parameter für den Datentransfer spezifiziert. Dazu werden drei Werte angegeben:

- Die Art der Verbindung (hier `GET`)
- Die Angabe der URL (`url`)
- Synchrones (`false`) oder asynchrones (`true`) Laden der Daten

Werden die Daten synchron geladen, ist die Anwendung im Gegensatz zum asynchronen Laden blockiert, bis die Daten an den Client übermittelt wurden. Der Variablen `onreadystatechange` wird eine Callbackfunktion zugewiesen, die nach Beendigung des Aufrufes ausgeführt wird. Als letztes wird die Funktion `send` aufgerufen, die den Request an den Server schickt. In der Callbackfunktion kann dann überprüft werden, ob die Anfrage komplett ist. Dies ist der Fall, wenn gilt:

```
xmlRequest.readyState == 4
```

Wurden die Daten fehlerfrei übertragen gilt:

```
xmlRequest.status == 200
```

Nach der fehlerfreien Übertragung, kann das Ergebnis über die Variable `responseXML` abgefragt und in das Dokument importiert werden, um den geladenen Knoten nach Belieben zu verarbeiten.

Die Vorgehensweise in Quellcode 4.10 birgt jedoch ein schwerwiegendes Problem. Werden mehrmals hintereinander Serveranfragen gestellt, kann es aufgrund der Nebenläufigkeit dazu kommen, dass einzelne Ladeoperationen nicht komplett ausgeführt werden. Der Grund ist die Verwendung einer globalen Variablen für das `XMLHttpRequest`-Objekt. Wird der erste Ladevorgang initiiert, kann es dazu kommen, dass die Serveranfrage etwas länger dauert und während des Ladens bereits ein zweites `XMLHttpRequest`-Objekt erzeugt wird. In der Callbackfunktion ist dann das erste Objekt nicht mehr sichtbar und die Anfrage kann nicht mehr verarbeitet werden. Die Lösung des Problems ist in Quellcode 4.11 zu sehen.

```
function holeDaten(url,callback) {
  req=new XMLHttpRequest();
  if(req) {
    var handlerFunction = getReadyStateHandler(req,callback);
    req.onreadystatechange=handlerFunction;
    req.open("GET",url,true);
    req.send(null);
  }
}
function getReadyStateHandler(req, responseXmlHandler) {
  return function () { // Liefert eine anonyme Funktion
    if (req.readyState == 4) {
      if (req.status == 200) {
        responseXmlHandler(req.responseXML);
      } else {
        alert("HTTP error: "+req.status);
      }
    }
  }
}
```

Quellcode 4.11: Verbesserte Serveranfrage mit XMLHttpRequest

Es wird die Funktion `holeDaten` aufgerufen, der die URL der zu ladenden Datei und eine Callbackfunktion übergeben wird. Innerhalb der Funktion wird zunächst das `XMLHttpRequest`-Objekt initialisiert. Dabei wird als Callbackfunktion mit der Funktion `getReadyStateHandler` eine anonyme Funktion erzeugt und zurückgeliefert. Die anonyme Funktion erhält das `XMLHttpRequest`-Objekt und die Callbackfunktion `responseXmlHandler` als Parameter, die der Funktion `holeDaten` übergeben wurden. Dadurch besitzt jede Callbackfunktion das passende `XMLHttpRequest`-Objekt und die Abarbeitung ist *threadsafe*. Das Konzept wirkt auf den ersten Blick unnötig kompliziert, allerdings sind diese beiden Funktionen universell einsetzbar und können direkt in anderen JavaScript-Code eingebunden werden. Der `getURL`-Mechanismus des Adobe SVG Viewers wird durch die beiden Funktionen nachgebildet.

4.5.3 Auswahl der Methode zum Nachladen der Daten

Da eine SVG Anwendung sowohl in Webbrowsern mit nativer Unterstützung als auch unter Verwendung des Adobe SVG Viewers in der Lage sein soll, Daten nachzuladen, muss jeweils entschieden werden, welche Methode zu verwenden ist. Dazu kann die Variable `window` befragt werden, ob eine Funktion bekannt ist (Quellcode 4.12).

```
if(window.getURL)
  getURL(url,callback);
else if(window.XMLHttpRequest) {
  req=new XMLHttpRequest();
  if(req) {
    var handlerFunction = getReadyStateHandler(req,callback);
    req.onreadystatechange=handlerFunction;
    req.open("GET",url,true);
    req.send(null);
  }
}
```

Quellcode 4.12: Ermitteln, ob `XMLHttpRequest` oder `getURL` verwendet werden muss

4.6 Programmierkonzepte in SVG

Bei der Softwareentwicklung haben sich standardisierte Programmierschemata durchgesetzt, so genannte Designpatterns [Gamma]. Dabei handelt es sich um Verfahren und Regeln für wieder kehrende Programmierarbeiten. Diese Designpatterns erleichtern oftmals die Softwareentwicklung und erlauben Programmierern, auch fremden Quellcode einfacher zu verstehen.

Bei der Entwicklung von SVG Applikationen wurden solche Techniken bisher kaum berücksichtigt. Dies liegt zum einen an der noch recht jungen Geschichte von SVG, andererseits wird SVG meist nur als Grafikformat verstanden, für das keine Designpatterns nötig wären. Dabei können mit SVG komplexe Oberflächen und Applikationen entwickelt werden. In diesem Bereich ist der Einsatz von Designpatterns durchaus sinnvoll.

Daher wurde ein Designpattern für die Erstellung dynamischer komplexer SVG Anwendungen in dieser Arbeit entworfen (Kapitel 9).

Als grundlegende Vorgehensweise sollte die Trennung von SVG Elementen und JavaScript vorgenommen werden. JavaScript Code in einer SVG Datei ist zum einen unübersichtlich, zum anderen zerstört es die logische XML Struktur und muss als `CDATA`-Element eingebunden werden. Daher sollte JavaScript Code immer in einer oder mehreren separaten Dateien ausgelagert werden und in der SVG Datei referenziert werden. Die externen JavaScript-Dateien können über das Element `script` eingebunden werden: `<script type="text/javascript" xlink:href="datei.js"/>`

Außerdem sollten die JavaScript Funktionen entsprechend ihrer logischen Zugehörigkeit in einzelne Dateien zerlegt werden, wie es analog auch bei objektorientierten Programmiersprachen geschieht, wo zusammengehörige Eigenschaften in separate Klassen eingeteilt werden. Dadurch wird die Programmierung vereinfacht, die Fehlersuche kann strukturierter erfolgen, einzelne Teile können separat getestet werden und bei einer Erweiterung der Applikation müssen nur einzelne Teile ausgetauscht werden.

Zumindest alle wichtigen Elemente sollten eine eindeutige und sinnvolle ID besitzen, damit das Element später schnell wiedergefunden und bearbeitet werden kann. Eine sinnvolle ID sollte erkennen lassen, um welches Element in welchem Kontext es sich handelt, damit die Programmierung später einfacher erfolgen kann, weil man nicht ständig nachschlagen muss, wie ein bestimmtes Objekt bezeichnet wird. Am besten wird vor der Entwicklung einer komplexen SVG Anwendung das Aussehen skizziert, Steuerelemente festgelegt und der Anwendungsverlauf schriftlich festgehalten. Auf dieser Basis können dann die Kernobjekte von Beginn an mit einer ID versehen werden, die dann den Entwicklern als Schnittstelle dienen kann.

Bei der Erstellung einer SVG Anwendung sollten logisch zusammengehörige Komponenten in Gruppen zusammengefasst und ebenfalls mit einer sinnvollen ID versehen werden. Dadurch ist es möglich größere Teile einer Anwendung ein- oder auszublenden, die Menüsteuerung wird vereinfacht und ein strukturierterer Umgang mit den dargestellten Daten wird ermöglicht. Für die Entwicklung von komplexeren SVG Anwendungen, in denen Daten dargestellt werden sollen, sind weitere Aufteilungen der beteiligten SVG Dateien sinnvoll.

5 Ausgangsdaten

Um eine Wettervorhersage mit einer geographischen Karte zu visualisieren sind zwei Datenformate von Bedeutung. Zum einen Shapefiles, in denen geographische Daten abgespeichert sind, zum anderen GRIB Files, in denen Wetter- oder Klimadaten abgespeichert werden. Beide Datenformate werden weltweit eingesetzt und stellen einen so genannten Quasi-Standard dar. Die beiden Formate sollen im Folgenden etwas näher betrachtet werden.

5.1 Shapefiles

Das Unternehmen ESRI (*Environmental Systems Research Institute*) [ESRI1] ist einer der größten Softwarehersteller von Geoinformationssystemen (GIS). Das bekannteste Produkt ist *ArcGIS*, welches unter anderem das Modul *ArcView* beinhaltet. ArcGIS kann als Desktop Lösung, als serverseitiges oder mobiles GIS und als GIS Web Service eingesetzt werden.

Das Dateiformat Shapefile (auch Shapedaten oder Shapes genannt) [ESRI2] wurde von ESRI für ArcView entwickelt und dient dazu, Geodaten abzuspeichern. Das Shapeformat besteht aus mindestens drei Dateien je Shapefile. In der Datei mit der Endung `.shp` wird die Geometrie abgespeichert. Diese Datei ist die Hauptdatei des Shapefiles. Die zur Geometrie gehörigen Sachdaten werden in einer Datei mit der Endung `.dbf` abgelegt. Die Verbindung zwischen den Geometriedaten und den Sachdaten wird in einer Datei mit der Endung `.shx` abgespeichert. Neben diesen drei Dateien gibt es noch folgende weitere, die allerdings optional sind:

- `.atx`: Attributindex
- `.sbx` und `.sbn`: Index für Tabellenverbindungen (Joins)
- `.aih` und `.ain`: Index für Tabellenverknüpfungen (Links)
- `.shp.xml`: Metadaten zum Shapefile
- `.prj`: Projektion der Daten
- `.avl`: Legendendatei

In der Hauptdatei können Geometriedaten eines Typs abgespeichert werden. Die Typen lassen sich in drei Klassen einteilen: Punkte, Linien und Polygone. Diese Typen können wiederum in weitere Shapetypen unterteilt werden. Dabei können zwei- und dreidimensionale Geometrien erfasst werden.

5.1.1 Aufbau eines Shapefiles

Der Inhalt der Hauptdatei kann in zwei Kategorien eingeteilt werden. Zum einen die Daten zum Filemanagement, wie die Angabe der Dateigröße. Zum anderen sind die eigentlichen Daten abgespeichert. Dabei wird die Hauptdatei mit einem 100 Byte langen Header eingeleitet, in dem unter anderem die Dateilänge, die Boundingbox und der Typ der enthaltenen Daten angegeben wird. Danach folgen die geometrischen Daten in einzelnen Records variabler Länge, wobei jeder Record mit einem Recordheader eingeleitet wird. Der Recordheader ist acht Bytes lang. In den ersten vier Bytes wird die Nummer des Records angegeben, in den weiteren vier Bytes wird die Länge des Records definiert. Nach dem Recordheader folgt die Angabe der eigentlichen Geometriedaten. Diese werden je nach Shapetyp mit einer unterschiedlichen Zahl von Bytes je Objekt angegeben, wobei in der Regel nur die Koordinaten und gegebenenfalls eine Z-Komponente angegeben wird. Dazu kann noch ein Messwert als vierte Komponente abgespeichert werden. Dabei können 32 Bit Integer und 64 Bit IEEE Double Werte verwendet werden.

Die Reihenfolge der Grafikobjekte in der Hauptdatei ist mit der Reihenfolge der zugehörigen Sachdaten (Attributen) aus der `.dbf` Datei identisch, was die Verarbeitung vereinfacht (siehe Tabelle 5.1).

ID	city.shp	city.dbf
1	type=1 (point) x=9.9748 y=53.5358	label=Hamburg category=1
2	type=1 (point) x=10.1312 y=54.2962	label=Kiel category=2
3	type=1 (point) x=8,0727 y=52.2697	label=Osnabrück category=4
4	...	...

Tabelle 5.1: Zusammenhang von `.shp` und `.dbf` Datei

5.1.2 Auslesen eines Shapefiles

Um ein Shapefile auszulesen, existieren verschiedene Bibliotheken in unterschiedlichen Programmiersprachen. In dieser Arbeit soll Java verwendet werden, damit die Anwendung plattformunabhängig eingesetzt werden kann. Die am weitesten verbreitete Java API zum Schreiben und Lesen von Shapefiles stammt von *BBN Solutions* [OpenMap]. Mit dem Open Source Projekt *OpenMap* können Karten in einem Javaprogramm dargestellt werden. Dazu sind unter anderem Klassen vorhanden, die Shapefiles lesen und schreiben können. Eine genauere Beschreibung ist im Kapitel 13.3 zu finden.

5.2 GRIB Files

Die Wetterdienste weltweit übertragen ihre Daten in der Regel mittels GRIB (Gridded Binary) Files. In GRIB Files werden Wetterdaten in einem regelmäßigen rechteckigen Gitter gespeichert. Zu jedem Gitterpunkt sind die genauen geographischen Koordinaten bekannt. Auch Klimadaten und andere Daten werden in GRIB Files abgespeichert. Mit

der Zeit sind unterschiedliche Dialekte des Formates entstanden, da sehr unterschiedliche Daten abgespeichert werden sollten. Im Groben werden drei Versionen unterschieden. Die Version 0 wird im allgemeinen nicht mehr verwendet. Version 2 ist noch nicht sehr verbreitet, so dass in erster Linie Version 1 verwendet wird.

5.2.1 Aufbau des GRIB Formates

Die eigentlichen Daten werden innerhalb eines GRIB Files in einzelnen Records strukturiert. Ein GRIB File kann aus beliebig vielen Records bestehen. Ein Record beinhaltet eine bestimmte Kenngröße (Temperatur, Regenmenge, etc.) zu einem festgelegten Zeitpunkt. Weitere Records in einem GRIB File können andere Kenngrößen und/oder andere Zeitpunkte enthalten. Jedes Record ist in sechs Sektionen unterteilt.

Section 0: Indicator Section (IS)

Die *Indicator Section* besteht aus acht Bit. Die ersten vier Bit enthalten die Zeichenfolge *GRIB* um den Start des Records zu markieren. Die nächsten drei Bits geben die Gesamtlänge des Records in Byte an. Das letzte Bit gibt an, um welche GRIB Edition es sich handelt.

Section 1: Product Definition Section (PDS)

In der *Product Definition Section* werden die im GRIB enthaltenen Daten spezifiziert. Die Länge dieser Section ist variabel, daher wird als erstes angegeben, wie viele Bytes in dieser Section enthalten sind. Im Weiteren werden Kennzahlen angegeben, mit denen über vorgegebene Tabellen die Zuordnung zu dem Parametertyp, der Parametereinheit, dem Datenlieferanten oder der Höhenangabe (z.B. zwei Meter über Normalnull) des Parameters vorgenommen werden kann. Neben der genauen Bezeichnung des enthaltenen Parameters werden Angaben über den Zeitpunkt gemacht zu dem dieser Parameter gemessen oder zu wann er vorhergesagt wurde.

Die im GRIB File abgelegten Daten sind alle komprimiert und müssen mittels verschiedener Faktoren zurückgerechnet werden. Die dafür notwendigen Faktoren werden ebenfalls in der Product Definition Section angegeben.

Neben diesen Daten hat jeder Datenlieferant die Möglichkeit weitere Daten am Ende der Section zu speichern. Dies kann für die Datenverarbeitung notwendig sein, oder aber für die zusätzliche Charakterisierung der Daten. Problem ist hierbei, dass die Angaben nicht standardisiert sind. Daher kann man diese Informationen ohne nähere Angaben des Datenlieferanten nicht verwerten.

Section 2: Grid Description Section (GDS)

Neben den enthaltenen Daten müssen auch die Angaben zu der genauen geographischen Lage und dem Aufbau des Gitters definiert werden. Diese Informationen befinden sich in der *Grid Description Section*. Die in GRIB Files abgespeicherten Daten bilden immer ein regelmäßiges Gitter (*Grid*). Daher reicht es aus, die Ausdehnung des Gitters in X als auch Y Richtung und Längen- und Breitenangabe des linken oberen und rechten unteren Punktes anzugeben. Meist wird aber noch der immer gleich bleibende Abstand zwischen zwei Punkten in X und Y Richtung angegeben.

Section 3: Bit Map Section (BMS)

Oftmals sind nicht an jedem Gitterpunkt wirklich Daten vorhanden. Wird beispielsweise die Wassertemperatur betrachtet, liegen für die Landmassen keine Daten vor. Werden nur einige wenige Daten innerhalb des Gitters nicht benötigt, so werden diese mit einem vorher definierten Wert versehen, um zu kennzeichnen, dass keine Daten vorliegen. Sind allerdings viele Gitterpunkte unbesetzt ist es nicht sinnvoll, alle diese Punkte mit einem bestimmten Zahlwert zu belegen. Um den Platz zu sparen, wird die Bit Map Section eingesetzt. Dabei wird für jeden Gitterpunkt mit einem Bit gekennzeichnet wo ein Wert vorliegt und wo nicht.

Section 4: Binary Data Section (BDS)

In der Section 4 werden die eigentlichen Werte abgespeichert. Dazu wird zunächst angegeben, wie viele Bits pro Wert vorgesehen sind und es werden weitere Informationen über die Art der Komprimierung der Daten angegeben. Danach folgen dann fortlaufend die komprimierten Werte.

Section 5: End Section (ES)

Die letzte Section besteht immer aus vier Bits, in denen jeweils eine *7* enthalten ist, um das Ende eines Records anzuzeigen.

5.2.2 Auslesen von Gribdateien

Obwohl das GRIB Format weltweite Verwendung findet und als offener internationaler Standard definiert wurde, sind kaum Programme vorhanden, mit denen das Format ausgelesen werden kann. Dies liegt wahrscheinlich an der Variabilität des Formates. So können fast alle Parameterwerte selbst definiert werden, wodurch viele verschiedene GRIB Dialekte existieren, so dass es fast unmöglich ist, alle in einem Programm zu erfassen. Trotzdem gibt es zwei Programme, mit denen GRIB Files ausgelesen werden können. Zum

einen existiert das Programm *WGRIB* [WGRIB]. Dabei handelt es sich um ein kommandozeilenorientiertes C-Programm. Mit WGRIB können GRIB Dateien ausgelesen und manipuliert werden. Allerdings ist die Ausgabe des Programms sehr kryptisch und kann nur schlecht automatisiert weiterverarbeitet werden. Zum anderen wurde am „Deutschen Klimarechenzentrum“ in der Arbeitsgruppe „Modelle und Daten“ die Programmsammlung *PINGO (Procedural INterface for Grib formatted Objects)* [PINGO] entwickelt. Dabei handelt es sich um eine komplexe Sammlung von kommandozeilenorientierten Programmen, um GRIB Dateien in beliebiger Form zu manipulieren und auszulesen.

Um eine Software zu entwickeln, die möglichst auf jedem Betriebssystem lauffähig ist und ohne die Installation externer Programme auskommt soll die gesamte Software in Java entwickelt werden. WGRIB und PINGO werden daher in dieser Arbeit nicht verwendet. Stattdessen wurde ein eigener `GRIBReader` in Java entwickelt. Näheres ist in Kapitel 13.4 nachzulesen.

5.3 Datenbeschaffung

Die Datenbeschaffung ist insbesondere in Deutschland sehr schwierig. Geographische Daten sind meist sehr teuer und unterliegen dem Datenschutzgesetz. Die deutsche Gesellschaft für Kartografie e.V. [DeKG] und Cartogis [Cartogis] bieten digitale Landkarten in unterschiedlichen Formaten zum Kauf an. ESRI bietet von vielen Ländern der Welt Daten in unterschiedlicher Detailliertheit im Shapefileformat kostenlos an [ESRI3]. Für die Stadt Osnabrück stellt die Intevation GmbH Daten als Open Source kostenlos zur Verfügung [Frida]. Frida ist eines der ersten Open Source Projekte in der Kartendigitalisierung.

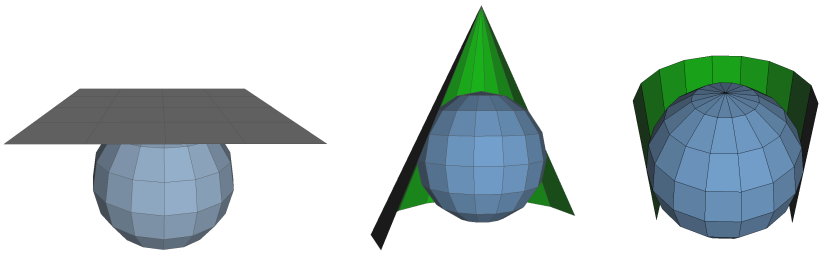
Wetterdaten sind in Deutschland ebenfalls nur schwer zugänglich. Der Deutsche Wetterdienst [DWD2] ist eine staatliche Einrichtung und darf jegliche Wetterdaten nur gegen Gebühr zur Verfügung stellen. Die Gebühr muss die Kosten für die Datenerhebung decken. So kosten beispielsweise die Messwerte einer Wetterstation in Deutschland bereits 60 Euro. Auch sonst werden in Deutschland keine kostenlosen Wettervorhersagedaten zur Verfügung gestellt. Deutschland ist das einzige Land weltweit mit dieser Verfahrensweise. In anderen Ländern werden Wetterdaten frei zur Verfügung gestellt. In der Regel handelt es sich dabei um die Rohdaten als GRIB File. Da insbesondere in den USA Wetterprognosen für die ganze Welt berechnet werden, ist es möglich auch für Europa geeignete Wetterprognosen zu erhalten. Die in dieser Arbeit hauptsächlich verwendeten Wetterdaten stammen von der National Oceanic & Atmospheric Administration [NOAA].

6 Projektionen geographischer Koordinaten

Die Erde stellt keine regelmäßige Kugel dar. Vielmehr besitzt sie eine komplexe Struktur. Dies liegt an Höhenunterschieden durch Berge und Täler und an der Abplattung der Erde an den Polen. Um dennoch ein regelmäßiges Koordinatensystem aufbauen zu können, wird meist von der idealisierten Kugelform ausgegangen. Bei der Vermessung einzelner Regionen werden oftmals Erdmodelle mit regionalem Charakter verwendet. Sie decken speziell die Bedürfnisse eines Landes oder eines Kontinentes ab. Bei diesen regionalen und nationalen Lösungen liegt ein *geodätisches Datum* zugrunde, welches die Lage, Orientierung und Größe der Bezugsfläche zum Erdmittelpunkt und einem darin gelagerten Koordinatensystem beschreibt, welches in der Regel durch die Erdachse, den Äquator und Greenwich (0^o geographische Länge) definiert ist [Voser].

6.1 Projektionseigenschaften

Die Verebnung der gemessenen Lagekoordinaten erfolgt durch eine *Kartenprojektion* als Abbildung des Erdmodells in die Ebene. Dabei ist entscheidend, welche Regionen abgebildet werden und wie groß diese sind. Je nach abzubildendem Gebiet werden verschiedene Projektionen verwendet, wie beispielsweise die Abbildung auf *Tangentialebenen*, *Kegel*, *Zylinder* (Abbildung 6.1) oder andere komplexe mathematische Flächen. Insgesamt sind über 200 verschiedene Projektionsmethoden bekannt [Wikipedia3].



(a) Azimuthal (b) Kegel (c) Zylinder

Abbildung 6.1: Kartenprojektionen mit einem Hilfskörper

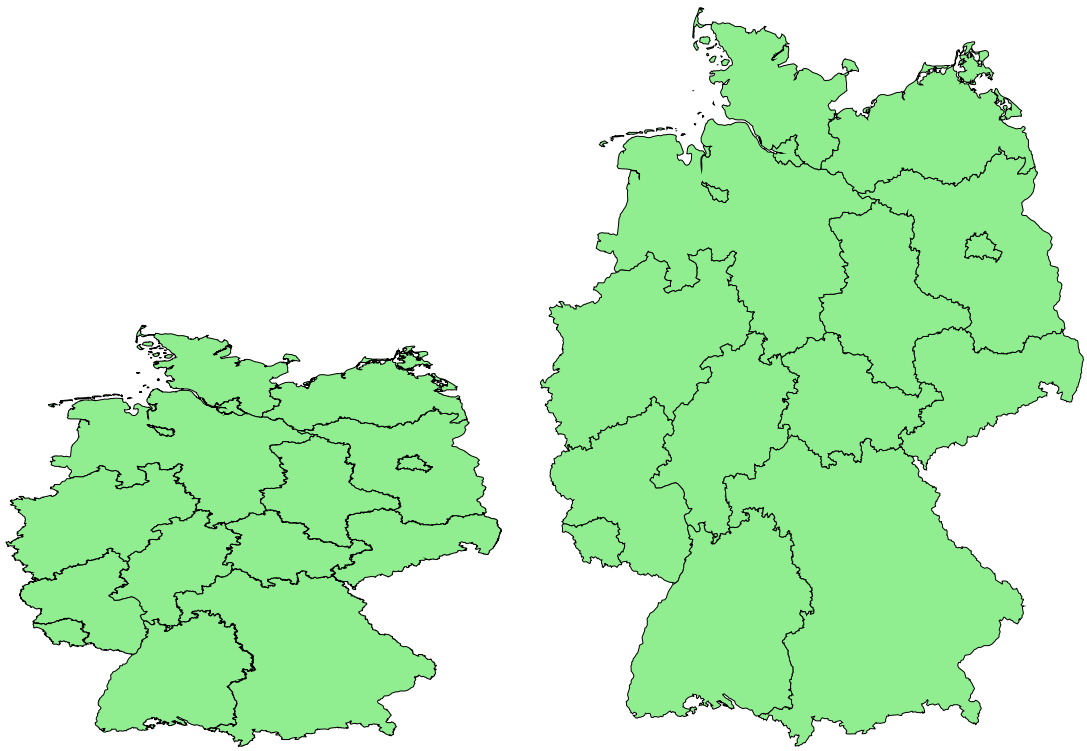
Für eine Kartenprojektion wäre es wünschenswert, dass die folgenden drei Eigenschaften erfüllt sind:

- *Winkeltreue*: Der Winkel zwischen zwei sich scheidenden Linien bleibt bei der Abbildung erhalten. Die Meridiane und Breitenkreise schneiden sich im rechten Winkel.
- *Längentreue*: Der Abstand zweier Punkte bleibt nach der Abbildung bis auf einen festen global gültigen Maßstabsfaktor erhalten.
- *Flächentreue*: Flächengrößen bleiben maßstabsbezogen erhalten und die proportionale Beziehung von Flächengrößen bleiben bestehen.

Verzerrungen treten auf, wenn eine der Anforderungen verletzt wird. Abbildungen, die keine Verzerrungen hinsichtlich der Strecken, Winkel und Flächen aufweisen, werden verzerrungsfrei genannt. Wie Leonhard Euler (1707-1783) gezeigt hat, ist es unmöglich, eine Kugel verzerrungsfrei in eine Ebene abzubilden. Insbesondere existiert keine Abbildung der Kugel in eine Ebene, die abstandstreu ist. Das schließt nicht aus, dass spezielle Linien in wahrer Länge abgebildet werden, bei vielen Abbildungen besteht dann noch die Möglichkeit, diese Linien gezielt auszuwählen. Aus den Überlegungen der Differentialgeometrie folgt weiter, dass sich Winkeltreue und Flächentreue ausschließen [Sosna]. So gibt es jede in Abbildung 6.1 gezeigte Kartenprojektion in einer winkeltreuen, einer längentreuen und einer flächentreuen Variante.

6.2 Lambertsche Azimuthal Projektion

Für regionale Karten, wie die Deutschlandkarte, wird oftmals der flächentreue azimuthale Entwurf von *Lambert* (1728-1777) verwendet. Die Projektion ist winkel- und flächentreu, allerdings werden die Längen verzerrt. Die Verzerrung ist am Zentrum (dem Berührpunkt) gleich Null, steigt aber mit zunehmender Entfernung an. Daher eignet sich die Projektion nicht für globale Kartendarstellungen. Die lambertsche und die geographische Projektion sind in der Abbildung 6.2 gegenübergestellt.



(a) Geographische Projektion (b) Lambertsche Projektion

Abbildung 6.2: Deutschland in geographischer und lambertscher Projektion

Für eine Deutschlandkarte bietet es sich bei der lambertschen Projektion an, die geographische Lage von Eisenach in Thüringen als Berührpunkt der Projektionsebene zu verwenden, da die Stadt ungefähr in der Mitte des Landes liegt und die Verzerrungen gleichmäßig zum Kartenrand zunehmen.

Die Projektion eines grafischen Objektes (beispielsweise einer Isolinie) wird durch das Projizieren seiner einzelnen Definitionspunkte erreicht. Die Formeln zur Berechnung der Bildkoordinaten des azimuthalen Entwurfs von Lambert lauten bei frei wählbarem Berührpunkt (u_0, v_0) [SnyderP]:

$$\begin{aligned} x &= k \cdot \cos v \cdot \sin u - u_0 \\ y &= k \cdot [\cos v_0 - \sin v_0 \cdot \cos v \cdot \cos u - u_0] \\ \text{mit } k &= \sqrt{\frac{2}{1+\sin v_0 \cdot \sin v + \cos v_0 \cdot \cos v \cdot \cos u - u_0}} \end{aligned}$$

Die Werte u und v stellen die geographische Länge und Breite dar. Der Bildbereich der Projektion liegt für x und y im Intervall $[-2, 2]$.

Projektionen für andere geographische Regionen können in [SnyderP] nachgeschlagen werden.

7 Darstellungsformen der Wetterdaten

In dieser Arbeit sollen die wesentlichen Wetterausprägungen visualisiert werden. Hierbei muss berücksichtigt werden, wie die einzelnen Ausprägungen besonders gut dargestellt werden können und wie man diese möglichst gemeinsam anzeigen kann.

7.1 Luftdruck

Der Luftdruck ist einer der entscheidenden Faktoren für das Wetter- und Klimageschehen. Durch die Druckunterschiede entstehen Winde, die letztendlich zur gesamten atmosphärischen Zirkulation führen. Die Einheit, in der der Luftdruck gemessen wird ist Hektopascal (hPa). Der mittlere Luftdruck auf Normalnull beträgt 1013hPa. Gebiete mit höherem Luftdruck werden als Hochdruckgebiete, Gebiete mit niedrigerem Luftdruck werden als Tiefdruckgebiete bezeichnet. Tiefdruckgebiete kennzeichnen in der Regel wechselhaftes und meist schlechtes Wetter. Ein Hochdruckgebiet hingegen steht für Stabilität und meist gutes Wetter. Sind die Druckunterschiede sehr groß, kommt es zu einem starken Druckgefälle. Bei einer solchen Wetterlage kommt es zu starken Stürmen und bei einem zusätzlichen hohen Temperaturgefälle oft auch zu Gewittern.

Trotz der Bedeutung des Luftdruckes wird er nicht immer in der Wettervorhersage angegeben oder wird nur am Rande erwähnt. Dies liegt daran, dass vielen die Bedeutung des Luftdruckes nicht klar ist. Für die Meteorologen ist der Luftdruck aber die entscheidende Größe zur Berechnung einer Wettervorhersage. Dazu wird der gemessene Luftdruck, ein skalarer Wert, in Wetterkarten eingetragen und Punkte gleichen Druckes werden miteinander verbunden. Generell bezeichnet man eine Linie die Punkte gleicher Merkmale oder Werte verbindet als Isolinie. Werden Punkte gleichen Luftdrucks verbunden spricht man von Isobaren (griechisch: *iso* „gleich“ und *baros* „Druck“).

Isobaren sind die gebräuchlichste und geeignetste Darstellungsform für Luftdruck (Abbildung 7.1). Hoch- und Tiefdruckgebiete sind schnell zu erkennen und durch den Abstand der Isobaren ist die Stärke des Druckgefälles ablesbar. Außerdem ist diese Darstellungsform recht einfach, da keine Flächen eingefärbt werden müssen und Isobaren können auch per Hand schnell in eine Karte eingezeichnet werden. Daher wird auch in dieser Arbeit der Luftdruck durch Isobaren visualisiert und auch andere Größen könnten durch Isolinien dargestellt werden.

7.2 Temperatur

Die Temperatur ist neben dem Luftdruck ein weiterer wesentlicher Faktor für das Wettergeschehen. So bewirken Temperaturunterschiede thermische Tiefdruckgebiete durch aufsteigende Warmluft (Bodentief, Hitzetief) oder durch absinkende Kaltluft (Höhentief). Durch diese Tiefs werden die polaren Ostwinde und die äquatorialen Passatwinde erzeugt. Neben diesen Winden treiben die Temperaturunterschiede zwischen den kalten Polen und dem warmen Äquator die planetarische Zirkulation an. Regional betrachtet können thermische Tiefdruckgebiete über warmem Wasser entstehen, indem feuchte

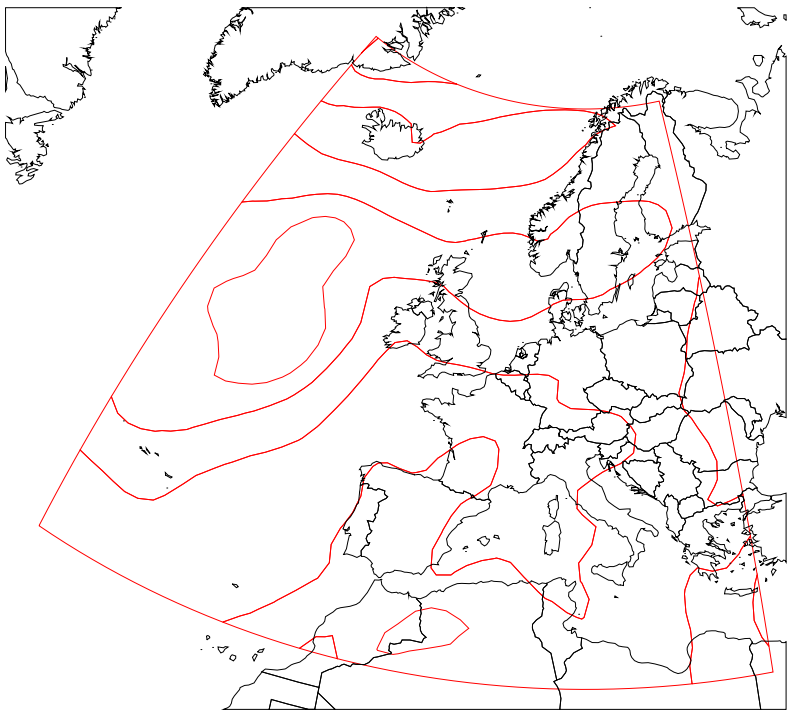


Abbildung 7.1: Isobarendarstellung des Luftdruckes

warme Luft aufsteigt, sich abkühlt und zu starken Regenfällen oder sogar Wirbelstürmen führt. In der Klimatologie ist die Betrachtung der Temperatur wichtig, da sie sich auf die Abschmelzung der Polkappen auswirken könnte und die Temperatur ein Indiz für den klimatischen Wandel einzelner Regionen oder sogar des Weltklimas darstellt.

Die Temperatur wird in unterschiedlichen Einheiten gemessen. Kelvin, Celsius und Fahrenheit sind die gebräuchlichsten, es gibt aber noch viele weitere Einheiten. In der Meteorologie wird Celsius weltweit und Fahrenheit hauptsächlich in den Vereinigten Staaten verwendet.

Wie der Luftdruck ist auch die Temperatur ein skalarer Wert. Zur Darstellung von unterschiedlichen Temperaturen werden in der Regel Farben verwendet, da der Mensch von je her bestimmten Temperaturen eine Farbe zuordnet. So ist eine hohe Temperatur meist rötlich. Dies kommt durch die Rote Farbe einer offenen Flamme. Die kalten Temperaturen werden meist mit Blau assoziiert, wahrscheinlich aufgrund der bläulichen Farbe des Eises. Dies sollte auch bei der Darstellung der Temperaturwerte berücksichtigt werden. Isolinien würden sich prinzipiell dazu eigenen eine Temperatur darzustellen, man spräche dann von Isothermen. Allerdings wären unterschiedlich eingefärbte Isolinien nur schwer zu erkennen. Daher werden die Flächen zwischen zwei Isolinien eingefärbt. Eine solche Fläche stellt einen Wertintervall dar und soll im folgenden *Isofläche* genannt werden. Mit Isoflächen können leicht verständliche Temperaturkarten erstellt werden (Abbildung 7.2).

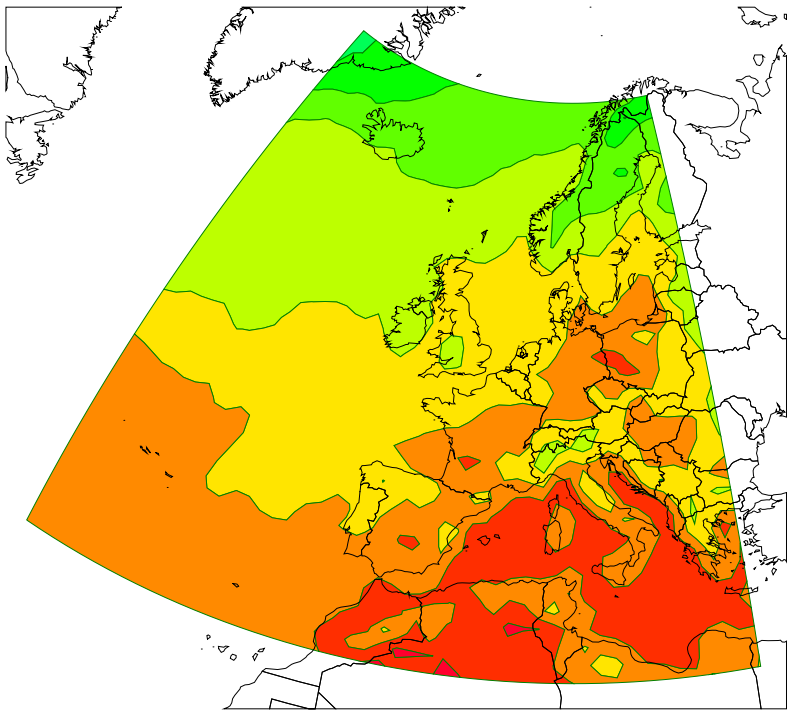


Abbildung 7.2: Isoflächendarstellung der Temperatur

7.3 Niederschlag

Niederschläge sind für den planetaren Wasserkreislauf von Bedeutung. Die durch Verdunstung in die Atmosphäre gelangte Feuchtigkeit wird über große Strecken transportiert und gelangt irgendwann wieder zum Boden zurück. Mit Niederschlag wird Wasser in fester oder flüssiger Form bezeichnet, welches aus der Atmosphäre auf die Erde fällt. Niederschläge bestimmen meist lokale Klimazonen.

Niederschlag wird meist punktuell gemessen. Dazu wird ein Niederschlagsmesser aufgestellt. Dabei handelt es sich um einen genormten Trichter, der das Wasser auffängt. Die Maßeinheit ist mm. Ein mm entspricht einem Liter pro Quadratmeter. Neben dieser punktuellen Messung werden auch Niederschlagsradare eingesetzt, die flächendeckend Niederschläge und deren Intensität messen. Allerdings kann hier nicht die genaue Regenmenge bestimmt werden. Vielmehr wird der Wassergehalt einer Wolke gemessen, was Rückschlüsse auf den Niederschlag zulässt. Daher sind diese Messungen nicht immer fehlerfrei, stellen aber eine gute Unterstützung der punktuellen Messung dar.

Niederschläge sind eine Auswirkung des Wetters und des Klimas. Sie sind weniger für das Wettergeschehen, als für den Wasserkreislauf des Planeten verantwortlich. Trotzdem ist die Angabe von kommenden Niederschlägen von Bedeutung. So spielen die Niederschläge vor allem bei der Landwirtschaft eine große Rolle. Von den Niederschlägen hängt der Ernteerfolg ab. Außerdem ist für einige Bauvorhaben trockenes Wetter notwendig, so dass die Wettervorhersage dieses Bereiches von großer Bedeutung ist, auch wenn die Niederschläge keine grundlegende Bedeutung für das allgemeine Wettergeschehen haben.

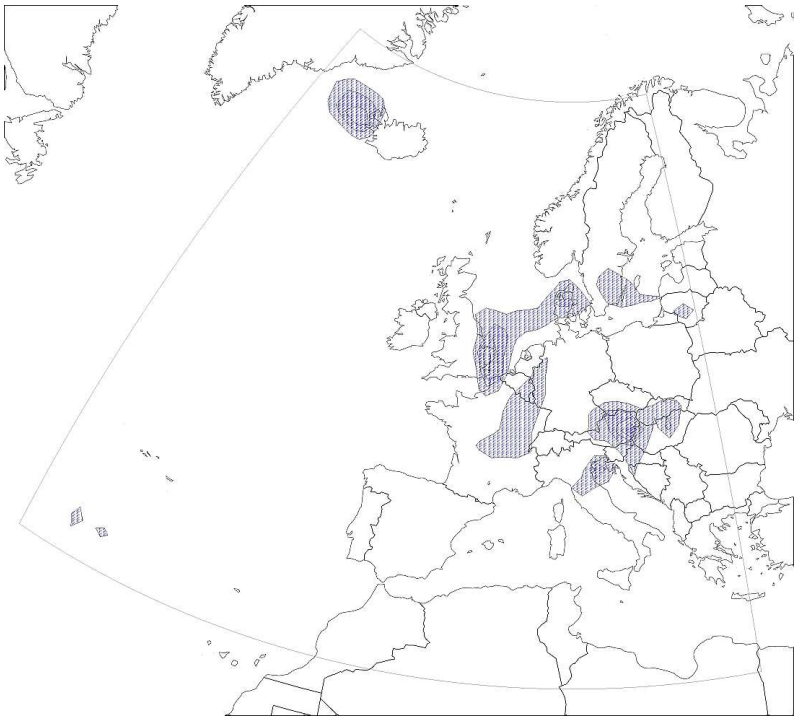


Abbildung 7.3: Darstellung des Regens mit Isoflächen und Patterns

Die gebräuchlichste Art der Visualisierung der Intensität des Niederschlages sind unterschiedlich gefärbte Isoflächen. Allerdings würde es zu Problemen führen, wenn sowohl die Niederschläge als auch die Temperatur mit farbigen Isoflächen dargestellt würden. Sollten beide Datensätze gleichzeitig sichtbar sein, müsste mindestens eine der beiden Komponenten leicht transparent sein. Bei einer Kombination der beiden könnten sich dann Farben überlagern und es würde kein klarer Eindruck über die Wetterlage entstehen können, da Mischfarben entstehen würden. Das ist auch der Grund, warum oftmals Temperatur und Niederschlag in getrennten Grafiken dargestellt werden.

Eine andere Art der Darstellung des Niederschlags sind unterschiedlich große Kreise, je nach Niederschlagsmenge. Auf diese Art und Weise wäre die Menge des Niederschlages klar erkennbar und die Temperatur ist weiterhin gut sichtbar. Aber diese Darstellungsform hat entscheidende Nachteile. Wenn eine Region der Karte vergrößert werden würde, ändert sich auch die Größe der Kreise. Daher wäre nur das Größenverhältnis der Kreise zueinander für die Kennzeichnung der unterschiedlichen Niederschlagsmengen zuständig. Da die Kreise sich nicht überlappen sollten muss ein maximaler Radius gefunden werden, so dass sich die Kreise nicht schneiden. Dies stellt prinzipiell kein Problem dar, da in dem regelmäßigen GRIB Gitter der Abstand eines Punktes zu dem anderen bekannt ist. Daher kann der Radius eines Kreises passend gewählt werden, so dass sie sich nicht überschneiden. Allerdings werden Karten oftmals in unterschiedlichen Projektionen dargestellt, da die Erde eine Kugel ist und deren Oberfläche auf einer zweidimensionalen Fläche dargestellt werden soll. Dabei kommt es zu Verzerrungen, wodurch die Punkte im regelmäßigen Gitter nach der Projektion nicht mehr äquidistant zueinander sind (Kapitel 6). Das macht es schwierig den geeigneten Kreisradius zu finden. Wird in der projizierten Darstellung der kleinste Gitterabstand als Radius gewählt werden die

Kreise zu klein, werden die Radien anhand anderer Gitterabständen festgelegt, kann es dazu kommen, dass sich die Kreise überlappen. Die Darstellung mit Kreisen ist also nur sinnvoll einsetzbar, wenn die Karte entweder in geographischen Koordinaten (also nicht projiziert) dargestellt wird, oder aber wenn nicht der Kreisradius die Intensität angibt, sondern eine unterschiedliche Einfärbung der Kreise.

Für eine interaktive Web Mapping Applikation kommen die beiden Techniken nur bedingt in Frage. Besser ist es, eine Kombination aus unterschiedlichen Techniken zu verwenden. So können die Regengebiete in Isoflächen eingeteilt und mit einem Füllmuster (Pattern) versehen werden (Abbildung 7.3). Dabei sind Pattern, die aus Linien bestehen sehr gut geeignet, da sie genügend Lücken aufweisen, um die darunter liegenden Daten zu erkennen, so dass keine Transparenz notwendig ist und der Farbeindruck nicht verfälscht wird.

7.4 Wind

Wie der Niederschlag ist der Wind eine Auswirkung der Luftdruck- und Temperaturverhältnisse auf der Erde. So entsteht Wind durch den Druckausgleich, was zu erdumspannenden Winden wie dem Jet-Stream oder den Passaten führt, aber auch zu lokaleren Erscheinungen wie Land-See-Wind oder Berg-Tal-Wind. Die Windgeschwindigkeit wird z.B. mit einem Windsack gemessen oder mit einem Schalenkreuzanemometer (Griech. *anemos* „Wind“). Dabei handelt es sich um ein Windrad mit drei oder vier halbkugelartigen Schalen, das durch den Wind angetrieben wird (Abbildung 7.4).

Abbildung 7.4: Anemometer zur Messung der Windstärke

Dabei werden verschiedene Maßsysteme verwendet, wie zum Beispiel *Knoten* (`kn`), *Meter pro Sekunde* (`m/s`) oder *miles per hour* (`mph`). Neben den Angaben der Windgeschwindigkeit kann die Windstärke auch klassifiziert werden. Dies ist mit der Beaufortskala möglich, die die Windstärke in zwölf Kategorien einteilt. Diese Skala wurde im Jahre 1806 von Francis Beaufort während seines Kommandos an Bord der Woolwich eingeführt und orientierte sich am Verhalten des Segels [Lauer]. Die Windvorhersage ist vor allem für die Schifffahrt von großer Bedeutung, da der Wind Auswirkungen auf den Seegang hat. Aber auch sonst sind Windvorhersagen wichtig, um vor stärkeren Stürmen geeignete Maßnahmen zur Absicherung zu treffen.

Da Wind ein vektorieller Wert ist müssen bei der Darstellung zwei Faktoren berücksichtigt werden: Die Windrichtung und die Windstärke. Daher ist die Winddarstellung eine der schwersten und komplexesten Darstellungsformen. In der Regel werden so genannte Fähnchen verwendet, die den Wind charakterisieren sollen (Abbildung 7.5). Dazu wird

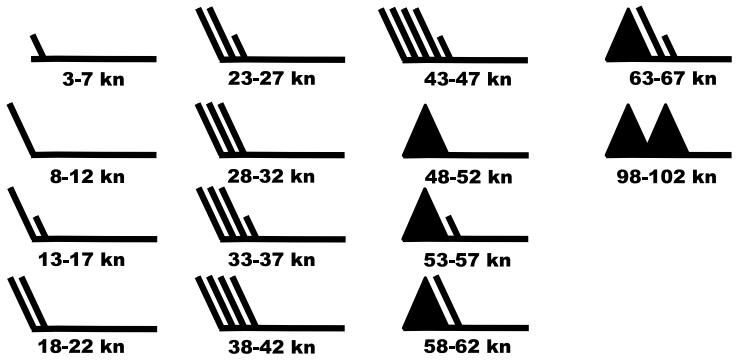


Abbildung 7.5: Windsymbole mit Angabe der Geschwindigkeit in Knoten

der Stab des Fähnchens anhand der Windrichtung ausgerichtet. Die Windstärke selber wird durch „Federn“ gekennzeichnet, die an dem Ende eingezeichnet werden, von wo aus der Wind weht. Für die Angabe der Windstärke wird eine unterschiedliche Anzahl und unterschiedlich lange Federn eingezeichnet. Dabei entspricht ein halber Strich fünf Knoten, ein ganzer zehn und ein Dreieck 50 Knoten. Abbildung 7.6 zeigt eine Windkarte mit den beschriebenen Windsymbolen.

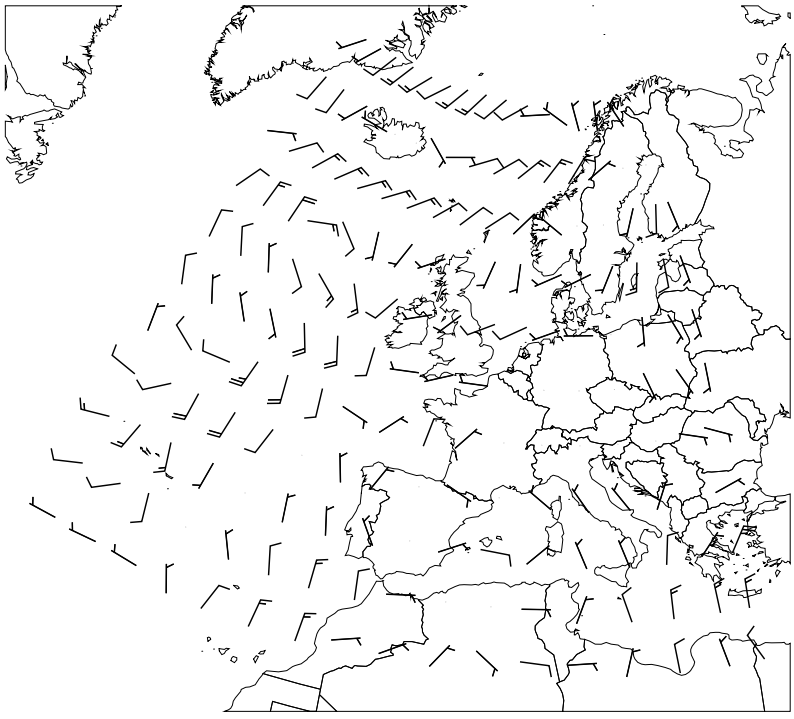


Abbildung 7.6: Darstellung von Winddaten

8 Web Mapping

Wettervorhersagen und die Ergebnisse von Klimasimulationsmodellen haben in der Regel eine zeitliche Dimension und immer eine räumliche Ausdehnung. Um die räumliche Dimension darzustellen, werden die Daten auf einer geographischen Karte angezeigt. Bei Klimadaten werden die Küstenlinien und Ländergrenzen eingezeichnet. Bei lokalen Wettervorhersagen werden zusätzlich größere Städte, Flüsse oder Gemeindegrenzen eingezeichnet. Im Fernsehen sind sogar dreidimensionale Flüge über das Land zu sehen, in denen auch die Höheninformationen des Landes berücksichtigt werden, so dass Berge und Täler zu erkennen sind. Allerdings stellt sich die Frage, ob die Wetterflüge in 3D den Betrachter nicht eher überfordern.

Will man eine interaktive Wettervorhersage im Internet präsentieren, ist es ebenfalls notwendig, eine Karte unter den eigentlichen Wettervorhersagedaten einzublenden. Sollen Ausschnitte vergrößert und hierbei weitere Informationen eingeblendet werden können, reicht eine statische Karte nicht aus. Je nach Zoomstufe sind andere geographische Informationen notwendig, was eine dynamische Karte erfordert.

Für die elektronische Kartendarstellung im Internet gibt es unterschiedliche Begriffe. „Internet Karte“, „Web Karte“, „netzbasierte E-Maps“ oder „Cybermaps“ sind nur eine kleine Auswahl von Namen für die Kartendarstellung im World Wide Web, je nach eingesetzter Technik und Funktionsumfang. Durch die Vielzahl von Begriffen ist eine Abgrenzung der einzelnen Anwendungen nur schwer möglich. Brandon Plewe wählte für die elektronische Kartendarstellungen im Internet den Oberbegriff *WebGIS* [Plewe] und unterteilte die einzelnen Anwendungen in unterschiedliche Typen:

Geodaten-Server Auf einem Server sind lediglich die reinen Geoinformationen zu finden. Der Server bietet Recherchemöglichkeiten, so dass gezielt nach Daten gesucht werden kann. Die Daten werden dann mit einem Desktop-GIS visualisiert. Die Kartendarstellung liegt also weiter bei dem Client, welcher als *thick client* bezeichnet wird.

Map-Server Der Client erhält vom Server eine bereits vorgerechnete Karte, die nur noch angezeigt werden muss. Dabei werden statische und dynamische Server unterschieden. Der statische Server hat vorgefertigte Karten, die auf Anfrage an den Client geliefert werde. Beim dynamischen Map-Server werden die Karten serverseitig zur Zeit der Anfrage generiert und mit den gewünschten Informationen versehen. Bei beiden Ansätzen benötigt der Client nur eine eingeschränkte Funktionalität, da die Karten in der Regel mit einem normalen Webbrowser betrachtet werden können, und wird daher als *thin client* bezeichnet.

Online-GIS Hierbei wird die Funktionalität eines GIS über das Internet verfügbar gemacht, so dass komplexe GIS Operationen durchgeführt werden können.

Neuere Veröffentlichungen verwenden jedoch eine andere Abgrenzung. So teilt Frank Dickmann die elektronischen Kartendarstellungen in zwei Typen ein: *WebGIS* und *Web Mapping* [Dickmann]. Web Mapping werden Kartendarstellungen im Internet genannt,

wenn ein Nutzer nicht nur das Kartenmaterial anschauen kann, sondern auch die Karte zoomen oder verschieben kann und einzelne Informationsebenen (Layer) ein- oder ausblenden kann. Wird der Nutzer durch die Anwendung in die Lage versetzt auf einem Server liegende Sachdaten zu ändern oder andere GIS-Operationen durchzuführen, spricht er von einem WebGIS. Nach [Abreu] wird als Web Mapping der Vorgang der Kartengenerierung durch die Beteiligung verteilter Rechner im Internet bezeichnet. Außerdem müssen verschiedene Datensätze (Informationsebenen) kombinierbar sein. Diese Definitionen des Web Mapping Begriffes sind aktueller und besser nachvollziehbar. Daher wird der Begriff Web Mapping im Folgenden im Sinne von Frank Dickmann und J. Abreu verwendet. Um eine solche Web Mapping Anwendung zu implementieren, stellt sich zunächst die Frage, welche Eigenschaften eine Web Mapping Applikation ausmachen.

8.1 Eigenschaften einer Web Mapping Applikation

Der Nutzer soll durch die Web Mapping Anwendung in die Lage versetzt werden, Regionen vergrößern zu können und die Karte in beliebige Richtungen zu verschieben (*zooming* und *panning*). Dies sollte verlustfrei geschehen, die Darstellungsqualität darf also nicht leiden. Hat man einen Ausschnitt vergrößert, sollte man die Karte auch verschieben können, entweder durch Steuerelemente oder durch ein Klicken und Ziehen mit dem Mauszeiger auf der Karte. Dazu werden in der Regel Lösungen verwendet, die auf Pixelgrafiken basieren. Dabei werden die Grafiken bei einem Verschieben der Karte oder der Vergrößerung einer Region komplett ausgetauscht, was einen höheren Datenverkehr verursacht. Der Datenverkehr kann durch vektorielle Ansätze vermindert werden, wie es beispielsweise bei *Map24* [MAP24] der Fall ist. Hier wird zunächst eine grobe Übersichtskarte dargestellt und bei der Vergrößerung eines Ausschnittes werden lediglich die zusätzlich benötigten Daten geladen. Alle anderen Daten können beibehalten und verlustfrei skaliert werden, da sie als Vektorgrafik vorliegen. So wird ein redundantes Nachladen von Daten vermieden, was zu einem geringeren Datenverkehr führt und die Anwendung wesentlich schneller macht.

Neben dem Einblenden zusätzlicher Informationen sollten beim Zoomen einzelne Layer die Auflösung wechseln können. Beispielsweise könnte bei einer Deutschlandkarte das Pro-Kopf-Einkommen zunächst je Bundesland dargestellt werden. Zoomt der Benutzer in die Karte ein, könnte es je nach Zoomstufe für den Landkreis, die Stadt oder die Gemeinde dargestellt werden. Dadurch können Informationen besser einzelnen Regionen zugeordnet werden.

Eine Web Mapping Applikation sollte vom Benutzer individuell nutzbar sein. Die Informationen sollten möglichst nach Themen gegliedert und in einzelne Ebenen, so genannte Layer, zusammengefasst werden. So ist es möglich, gezielt einzelne Layer ein- oder auszuschalten. Das erlaubt eine Kombination von unterschiedlichen Sachdaten, so dass verschiedene Kenngrößen in Bezug zueinander gestellt werden können, was einen wesentlichen Aspekt bei einem GIS darstellt [Plewe].

Für die Wetterdarstellung ist zudem eine zeitliche Komponente von Bedeutung. Da das Wetter sich mit der Zeit verändert und Wettervorhersagen für mehrere Stunden oder Tage berechnet werden, sollen diese Daten auch Zeitpunkt für Zeitpunkt angezeigt werden

können. Dies ist nicht nur für das Wetter oder das Klima notwendig, auch andere Daten ändern sich über die Zeit. Bisher wurden zur Darstellung von zeitlichen Verläufen die einzelnen Zeitpunkte auf einzelnen Layern angeordnet, wie z.B. beim UMN Mapserver [UMN] oder GoogleEarth [Google2]. So konnte der Benutzer sich einen Zeitpunkt aussuchen, indem der betreffende Layer eingeschaltet und alle anderen abgeschaltet wurden. Für einen anderen Zeitpunkt muss dieser Layer wieder ausgeschaltet und ein andere muss aktiviert werden. Diese Vorgehensweise ist nicht komfortabel, vor allem wenn verschiedene Kenngrößen mit jeweils unterschiedlichen Zeitpunkten visualisiert werden sollen. Daher ist ein Konzept wichtig, bei dem alle Layer mit einer zeitlichen Komponente gleichzeitig auf einen anderen Zeitpunkt umgeschaltet werden können.

8.2 Web Mapping: Pixel vs. Vektorgrafik

Die Unterschiede zwischen Vektor- und Pixelgrafik wurden bereits in Kapitel 3 grundsätzlich erläutert. Es bleibt die Frage, wie sich die beiden Grafikformate auf eine Web Mapping Applikation auswirken.

Bei der Arbeit mit einer Web Mapping Applikationen wird die Karte bewegt, Ausschnitte werden vergrößert und Informationsebenen müssen ein- oder ausgeblendet werden. Werden Pixelgrafiken verwendet, muss bei jeder dieser Aktionen eine komplett neue Grafik geladen werden. Dies verursacht einen hohen Datenverkehr. Besonders ineffizient sind die Pixelgrafiken beim Zoomen in die Karte. Wird ein Ausschnitt vergrößert, wird das gesamte Bild neu geladen, obwohl bereits einige Informationen vorhanden sind. Beispielsweise sei in einer Karte der Umriss von Deutschland zu sehen. Zoomt der Anwender in die Karte hinein kommen je nach Zoomstufe Bundesländer, Flüsse oder kleinere Städte hinzu. Da die Pixelgrafik an sich nicht zoomfähig ist werden alle Informationen, also die komplette Grafik in einer höheren Auflösung neu geladen. Verwendet man ein Vektorgrafikformat, braucht man nur die wirklich neuen Daten laden, da alle anderen bereits vorhandenen Informationen verlustfrei vergrößert werden können.

Ähnlich verhält es sich beim Ein- oder Ausblenden eines Layers. Bei Pixelgrafiken muss die gesamte Grafik ausgetauscht werden, da nicht ohne weiteres zwei Pixelgrafiken übereinander gelegt werden können. Bei der Vektorgrafik jedoch kann man einzelne Teile des Bildes einfach verändern, ohne das ganze Bild neu zu laden. Daher ist hierbei im Idealfall überhaupt keine Datenübertragung notwendig.

Allerdings hat die Vektorgrafik das Problem, dass bei der Anzeige eine höhere Rechenleistung notwendig wird. Insbesondere bei vielen Informationen, die gleichzeitig angezeigt werden sollen ist die Pixelgrafik im Vorteil. Sie wurde beispielsweise auf einem Server vorberechnet und der Client muss lediglich ein einfaches Bild darstellen. Unabhängig davon wie viele Informationen vorhanden sind, bleibt die Performance bei Verwendung von Pixelgrafiken konstant. Bei der Vektorgrafik verschlechtert sich die Performance der Anwendung, je mehr Informationen angezeigt werden. Allerdings kann dieser Nachteil der Vektorgrafik durch geschickte Datenhaltung minimiert werden.

Eine vektorbasierte Web Mapping Anwendung bietet also viele neue Möglichkeiten gegenüber der Verwendung von Pixelgrafiken. Die Frage ist jedoch, welches internetfähige

Vektorgrafikformat in Frage kommt. Grundsätzlich kommen nur zwei Formate in Frage: Flash und SVG. Wie bereits in Kapitel 3.3 gezeigt, bietet SVG bessere Möglichkeiten bei der Entwicklung komplexer Anwendungen und hat auch diverse andere Vorteile. Daher soll die Web Mapping Anwendung mit SVG implementiert werden.

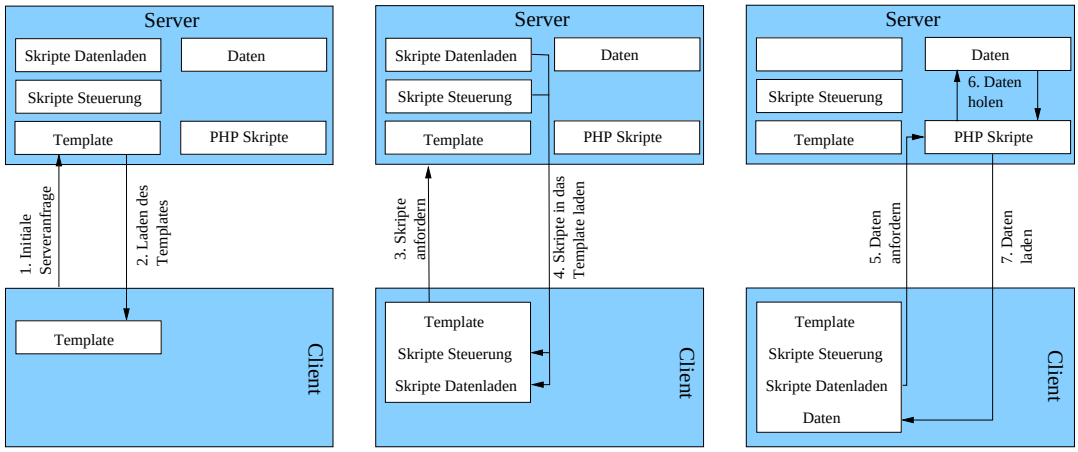
III Realisierung

9 Das SVG Template Konzept

SVG wird insbesondere zur Datenvisualisierung verwendet. Dabei werden Informationen grafisch aufbereitet und in nur einer SVG Datei gespeichert. Eine solche Vorgehensweise hat eine Vielzahl von Nachteilen. So wird die SVG Datei sehr groß, was trotz der breitbandigen Internetzugänge vermieden werden sollte. Durch eine Überfrachtung an Inhalten wird der Client zu stark belastet. Alle Informationen müssen eingelesen und in einen DOM-Tree überführt werden, selbst wenn sie nicht angezeigt werden. Dies belastet unnötig den Clienten und es kann dazu führen, dass die Datenmenge nicht mehr bewältigt werden kann, da der Aufbau eines DOM-Tree sehr speicherintensiv ist. Außerdem wird der Anwender bei größeren Datenmengen ohnehin nicht alles betrachten können, so dass für seine Zwecke zu viele Daten übertragen wurden. Wird eine SVG Anwendung die in einer Datei abgelegt ist von mehreren Entwicklern programmiert, ist die Erstellung ohne eine sinnvolle Aufteilung äußerst schwierig, da so alle an ein und derselben Datei arbeiten müssen, bzw. einzelne Teile nur im ganzen getestet werden können. Ein weiterer Nachteil ist die Aktualisierung der Daten. Ändern sich einzelne Datensätze, muss die gesamte SVG Anwendung neu erstellt werden.

Um die genannten Probleme zu umgehen, ist im Rahmen dieser Arbeit ein Designpattern für SVG Anwendungen entwickelt worden. Das Designpattern soll als *Template Konzept* bezeichnet werden. Dabei wird nicht nur der JavaScript Teil vom SVG Teil getrennt, sondern die Applikation in logische Teilstücke zerlegt.

Eine Anwendung besteht in der Regel aus einer Oberfläche, mit der der Anwender die Applikation steuern kann, den Daten, die angezeigt oder manipuliert werden und der Anwendungslogik (Abbildung 9.1). In diese Teile soll auch eine SVG Anwendung zerlegt werden.



(a) Template laden (b) Skripte laden (c) Daten laden

Abbildung 9.1: Ladevorgang beim SVG Template Konzept

9.1 Anwendungsdaten

Betrachten wir zunächst die darzustellenden Daten. Diese Daten werden separat auf der Serverseite als SVG Fragmente zur Verfügung gestellt, wobei sie in einzelne logische Segmente zerlegt werden sollten. Die Erstellung der Daten kann unabhängig von der Entwicklung der SVG Applikation erfolgen. Dies hat zur Folge, dass man sich voll und ganz auf die Erstellung dieser Daten konzentrieren und dafür Programme oder Skripte entwickeln kann. Die SVG Fragmente können in beliebiger Form auf dem Server gespeichert werden. Vorstellbar ist eine Speicherung direkt im Filesystem oder in einer Datenbank. Um die Daten der SVG Anwendung zugänglich zu machen, sind auf dem Server Skripte notwendig, die die Art der Speicherung kennen und über eine geeignete Schnittstelle verfügen, um clientseitige Anfragen zu bearbeiten.

9.2 Präsentationsebene

Die Präsentationsebene, also die Oberfläche, wird ebenfalls separat entwickelt. Dabei kann das Layout beliebig gewählt werden. Es werden lediglich Platzhalter definiert, an denen später einzelne Komponenten eingefügt werden. Bei diesen Komponenten handelt es sich zum einen um die darzustellenden Daten, zum anderen um Schaltflächen zur Steuerung der Anwendung, die bei der Erstellung des Templates oder erst später mittels JavaScript hinzugefügt werden können. Dabei hat der Entwickler lediglich darauf zu achten, dass die Platzhalter mit eindeutigen und vorab definierten IDs versehen sind. So kann ein grafisch versierter Entwickler für die Anwendung ein ansprechendes Design erstellen.

9.3 Programmlogik

Als dritte und letzte Komponente muss die Programmlogik entwickelt werden. Dazu werden Schaltflächen definiert und die zugehörigen Funktionen in JavaScript implementiert. Dabei bezieht sich der Entwickler auf die vorab definierten IDs der Platzhalter und auf die Dateinamen der darzustellenden Daten, um die Anwendung zu implementieren. Dabei sollte darauf geachtet werden, dass nur die Daten geladen werden, die der Anwender sehen möchte. Dies kann über dynamisches Nachladen realisiert werden, was bereits in Abschnitt 4.5 erläutert wurde. Auch hier kann wiederum eine Aufteilung in zwei Komponenten erfolgen. So stellen die clientseitigen Skripte lediglich eine Anfrage an den Server und senden dabei Angaben mit, aus denen der Server ermitteln kann welche Daten benötigt werden. Auf der Serverseite werden dann die passenden Daten ausgewählt und an den Client zurückgeschickt. So braucht der Client keine Informationen darüber, wie die Daten auf dem Server abgelegt sind und die Datenhaltung kann beliebig geändert werden. Lediglich die serverseitigen Skripte stellen die Verbindung zwischen der Anfrage vom Client und den Daten dar und bieten somit eine Art Kapselung der Datenhaltung.

9.4 Fazit

Der Name Template Konzept leitet sich aus der zu implementierenden Oberfläche mit den Platzhaltern ab. Die Oberfläche übernimmt die Rolle eines Templates, welches erst später mit Leben gefüllt wird.

Durch dieses Konzept wird eine Aufgabenteilung ermöglicht, so dass größere Entwicklerteams eine SVG Anwendung erstellen können. Außerdem können einzelne Teile einfach getestet oder ausgetauscht werden. Durch die Separierung der darzustellenden Daten und ein dynamisches Nachladen wird außerdem der Client wenig belastet und die zu übertragende Datenmenge wird reduziert. Serverseitig können unterschiedliche Systeme zur Datenverwaltung eingesetzt und nach Belieben ausgetauscht werden. So können die Dateien im Filesystem des Servers liegen oder in einer Datenbank gespeichert werden.

10 Vektorisierung der GRIB Rasterdaten

Im Kapitel 5.2 wurde das GRIB Format vorgestellt und in Kapitel 13.4 wird erläutert, wie die GRIB Daten ausgelesen werden können. Diese Daten müssen weiterverarbeitet werden, um daraus Isolinien und -flächen zu erzeugen. Dazu sind Vektorisierungsalgorithmen notwendig und Algorithmen, um die Lage der Isoflächen zueinander zu bestimmen.

Die im GRIB File vorliegenden Werte sind in einem regelmäßigen Gitter abgelegt. Es handelt sich um dreidimensionale Daten, da ein zweidimensionales Gitter vorliegt und an jedem Gitterpunkt ein Wert angegeben ist. Um aus den skalaren Werten Isolinien oder Isoflächen zu gewinnen, ist es notwendig die Rasterdaten zunächst zu vektorisieren. Die Vektorisierung von Rasterdaten ist ein spezielles Gebiet der Computergrafik und wird in der Regel dazu eingesetzt, um aus einer Pixelgrafik eine Vektorgrafik zu erstellen. Je nach Vektorisierungssoftware können zusammenhängende Gebiete, Linien oder Kreisbögen erkannt werden. Dieser Vorgang ist meist sehr rechenintensiv und entsprechende Software kostet bis zu mehreren 10.000 Euro.

GRIB Dateien enthalten die Werte eines Parameters, wie beispielsweise Wind, Luftdruck oder Temperatur, für ein reguläres rechteckiges Gitter, dass einer bestimmten geographischen Region zugeordnet werden kann (Abschnitt 5.2). Die Parameterwerte werden für die Vektorisierung in einem zweidimensionalen Array gespeichert werden. Es müssen nur die Indizes der Werte in X- und Y-Richtung verwendet werden. Die echten Koordinaten der Werte auf der Erdoberfläche können zunächst vernachlässigt werden. Erst nach der Vektorisierung werden die Werte auf Erdkoordinaten umgerechnet. Dadurch wird die Verarbeitung der Gitter-Daten wesentlich einfacher.

Jeder Gitterpunkt hat vier direkte Nachbarpunkte. Zur Vereinfachung wird davon ausgegangen, dass sich der Wert eines Parameters zwischen zwei direkt benachbarten Punkten linear verhält. Daher können für zwei Punkte durch lineare Interpolation Zwischenwerte errechnet werden, wie in Abbildung 10.1 dargestellt.

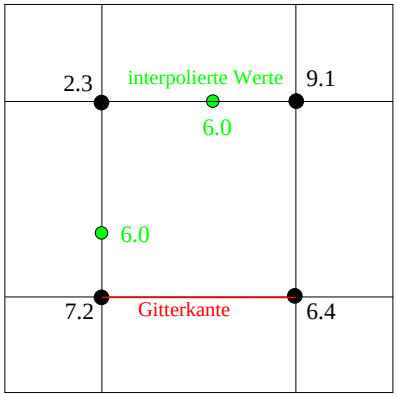


Abbildung 10.1: Interpolation auf Gitterkanten

Um die Daten zu vektorisieren gibt es verschiedene Algorithmen, die im Folgenden näher untersucht werden.

10.1 Contourplot Verfahren

Um besonders schnell Isolinien zeichnen zu können, gibt es so genannte Contourplot Verfahren. Diese Verfahren visualisieren dreidimensionale Daten für eine zweidimensionalen Ausgabe (Papier, Bildschirm). Dabei wird ein Parameter als Funktion zweier Veränderliche, X- und Y- Achse oder Längen- und Breitengrad, gezeichnet. Beispielhaft wird der *CONREC*-Algorithmus [Bourke] von Paul Bourke aus dem Jahr 1987 vorgestellt und anhand dessen die Vor- und Nachteile eines Contourplot Algorithmus dargestellt, da der Algorithmus einfach zu verstehen und dabei sehr effizient ist.

10.1.1 Der CONREC Algorithmus

Der Algorithmus benötigt folgende Eingabeparameter:

- Anzahl der Zeilen und Spalten des Gitters
- Anzahl der Konturwerte
- Eindimensionales Array mit den Konturwerten
- Zweidimensionales Array mit den Gitterwerten
- Zwei Arrays für die Koordinatenwerte in X- und Y-Richtung

Beim CONREC Algorithmus werden jeweils vier Gitterpunkte gleichzeitig betrachtet: (i, j),$(i+1, j)$,$(i, j+1)$ und $(i+1, j+1)$. Durch Bildung des Mittelwertes der Werte dieser vier Gitterpunkte erhält man den interpolierten Wert in der Mitte des Rechteckes. Das Rechteck wird dann in vier Dreiecke unterteilt, indem zwei Diagonalen von Eckpunkt zu Eckpunkt durch den Mittelpunkt gezogen werden, wie in der Abbildung 10.2 zu sehen ist.

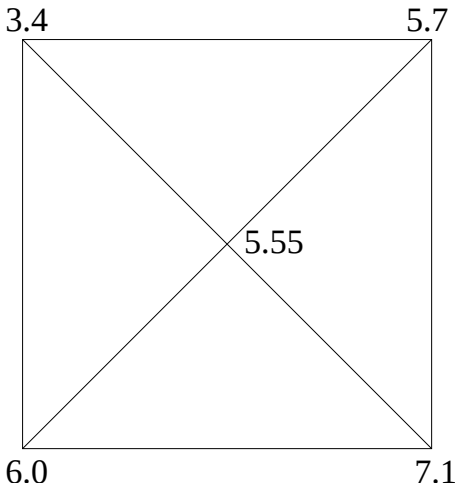


Abbildung 10.2: Triangularisierung im CONREC Algorithmus

Geometrisch interpretiert man die Werte der fünf Punkte als Höheninformation und erhält ein dreidimensionales Gebilde. Ein Konturwert stellt eine Fläche parallel zur XY Ebene dar und schneidet gegebenenfalls eine oder mehrere der Dreiecksflächen. Der Schnitt dieser Flächen ergibt gerade Liniensegmente, welche einen Teil der Konturlinie eines Konturwertes bilden (Abbildung 10.3).

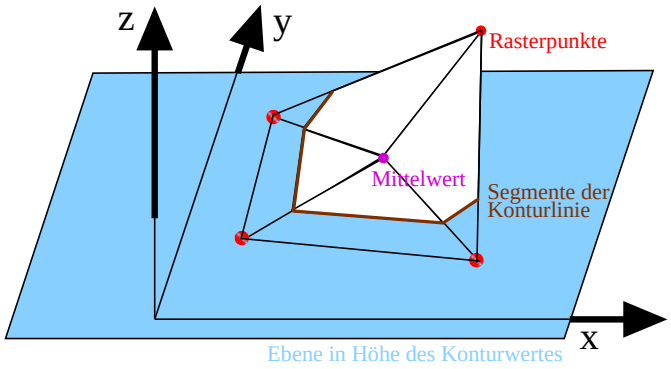


Abbildung 10.3: Geometrische Interpretation des CONREC Algorithmus

Für die Lage der Liniensegmente in Bezug zu den Werten der fünf Punkte und zu der Konturwertebene ergeben sich insgesamt zehn verschiedene Fälle (Abbildung 10.4).

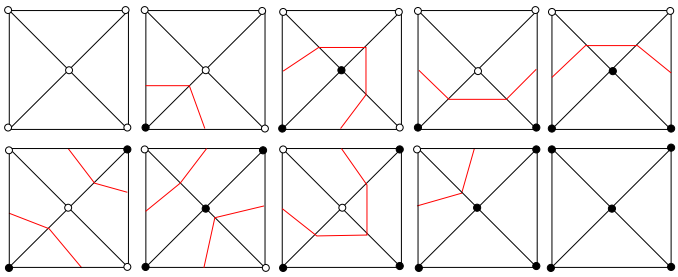


Abbildung 10.4: Die verschiedenen Fälle beim CONREC Algorithmus

Je nach Fall werden die möglichen Konturliniensegmente errechnet. Dabei werden für die vier Punkte des Rechtecks und für den fünften interpolierten Wert der Reihe nach alle Konturwerte betrachtet und alle zugehörigen Liniensegmente bestimmt. Danach wird der Vorgang für das nächste Rechteck fortgeführt. Das Ergebnis wird direkt auf dem Bildschirm oder einem Blatt Papier ausgegeben.

Bewertung

Der CONREC Algorithmus ist wie alle Contourplot Algorithmen sehr effizient, sowohl in der Zeichengeschwindigkeit, als auch in der Speicherbelastung. Der Algorithmus kann so modifiziert werden, dass nur Teile des Gitters eingelesen werden müssen, da immer nur ein Rechteck aus vier Punkten betrachtet wird. Die Konturlinien können mit einer leichten Modifizierung des Algorithmus in Abhängigkeit ihres Konturwertes direkt unterschiedlich eingefärbt werden.

Es entstehen in der Ausgabe geschlossene Konturlinien, die allerdings nur aus Liniensegmenten bestehen und keine geschlossenen Polygone darstellen. Um aus den Teilstücken zusammenhängende Polygone zu bilden müssten alle Teilstücke miteinander verglichen werden, um die einzelnen Teile aneinander zu reihen. Das Laufzeitverhalten wäre außerordentlich schlecht.

Ein weiteres Problem ist die fehlende Randbetrachtung des Gitters. Teilstücke, die an den Rand des Gitters stoßen können nicht zu einem kompletten Polygon verbunden werden. Daher ist ein Füllen einzelner Isoflächen mit dem CONREC Algorithmus nur mit großem Aufwand möglich. Andere Contourplot Verfahren besitzen dieselben Grundprobleme. Daher kommen diese Algorithmen in der Arbeit nicht zum Einsatz.

10.2 Marching Cube Verfahren

Das *Marching Cube* Verfahren [Lorensen] aus dem Jahre 1987 dient zur Visualisierung dreidimensionaler Daten mittels Isoflächen. Die Daten müssen als dreidimensionales skalares Punktfeld vorhanden sein, den so genannten *Voxeln*. Aus jeweils acht Voxeln wird ein Würfel definiert, wobei an den acht Ecken des Würfels die skalaren Werte eingetragen sind. Für einen vom Benutzer angegebenen Isowert wird untersucht, welche der Eckpunkte über und unter dem Isowert liegen. Im nächsten Schritt wird berechnet, wo Schnittpunkte auf den Würfelkanten zu finden sind. Mit diesen Schnittpunkten werden einzelne Flächen gebildet. Insgesamt werden 256 verschiedene Anordnungen unterschieden, die durch Rotation oder Symmetrie auf 15 Grundtypen zurückgeführt werden können (Abbildung 10.5).

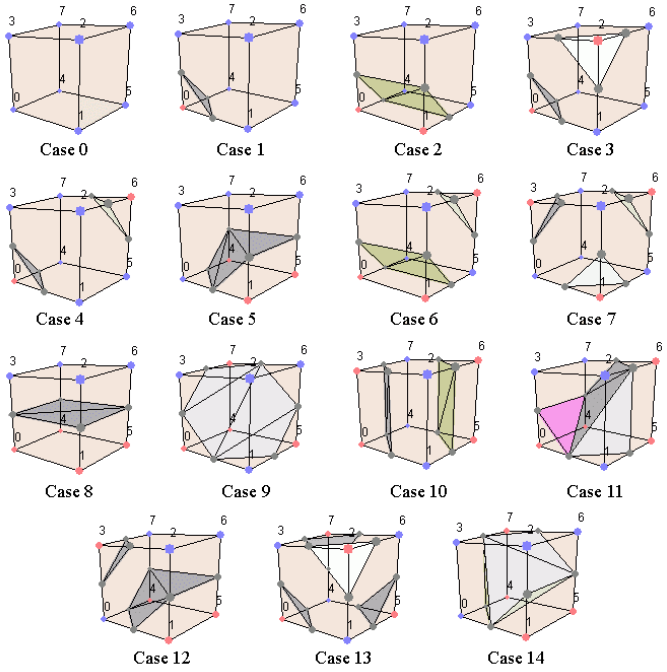


Abbildung 10.5: 15 Grundtypen beim Marching Cube Algorithmus [Lingrand]

Damit die Analyse eines Würfels sehr schnell erfolgen kann, werden die 256 verschiedenen Anordnungen in Tabellen gespeichert und für jeden Würfel kann sofort die Lage der enthaltenen Flächen ermittelt werden. Diese Vorgehensweise wird für jeden einzelnen Würfel wiederholt. Der Algorithmus „marschiert“ durch die einzelnen Würfel. Daher kommt auch der Name des Verfahrens.

Das Marching Cube Verfahren wird unter anderem verwendet, um medizinische Daten zu visualisieren, die beispielsweise bei einer Computertomographie gewonnen werden (Abbildung 10.6), um dreidimensionale Laserscans darzustellen oder um beliebige andere dreidimensionalen Messwerte darzustellen.

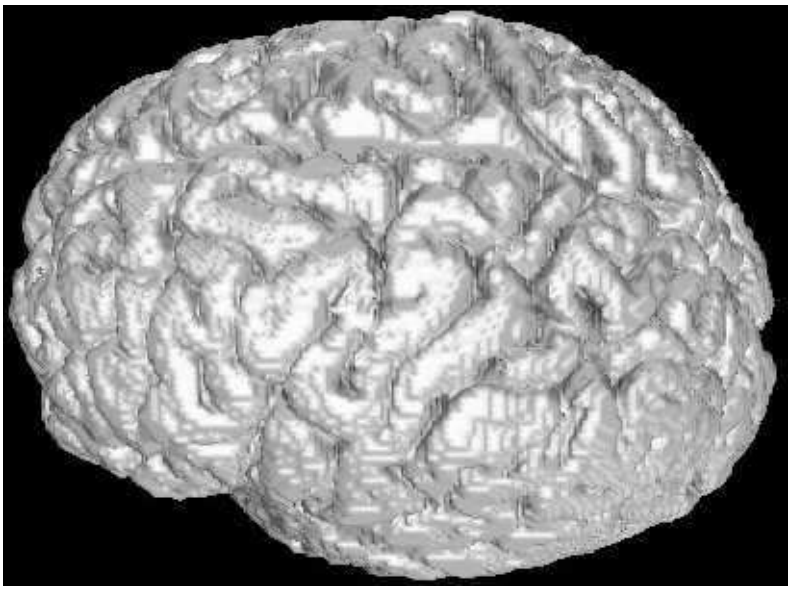


Abbildung 10.6: Visualisierung eines Gehirns aus 128984 Voxeln [Lingrand]

10.2.1 Marching Square Verfahren

Das *Marching Square* Verfahren ist eine Adaption des dreidimensionalen Algorithmus auf den 2D Bereich und wird vor allem für die Visualisierung von Isolinien verwendet, indem zwischen linear interpolierten Schnittpunkten auf den Gitterkanten Liniensegmente gezeichnet werden, ähnlich wie beim CONREC Verfahren (Kapitel 10.1.1).

Als Eingabe erwartet der Algorithmus ein zweidimensionales Gitter mit Werten an den Gitterpunkten. Um eine Isolinie für einen Isowert zu zeichnen, werden die Gitterzellen der Reihe nach betrachtet. Je Gitterzelle wird für die vier Werte an den Ecken untersucht, ob sie größer oder kleiner als der gesuchte Isowert sind. Je nachdem, auf welchen Gitterkanten der Wert gefunden wurde, können insgesamt 16 verschiedene Fälle auftreten. In der Abbildung 10.7 sind die Fälle dargestellt. Ein weißer Kreis bedeutet, dass der Wert unterhalb des Isowertes liegt und ein schwarzer Kreis gibt an, dass sich der Wert oberhalb des Isowertes befindet.

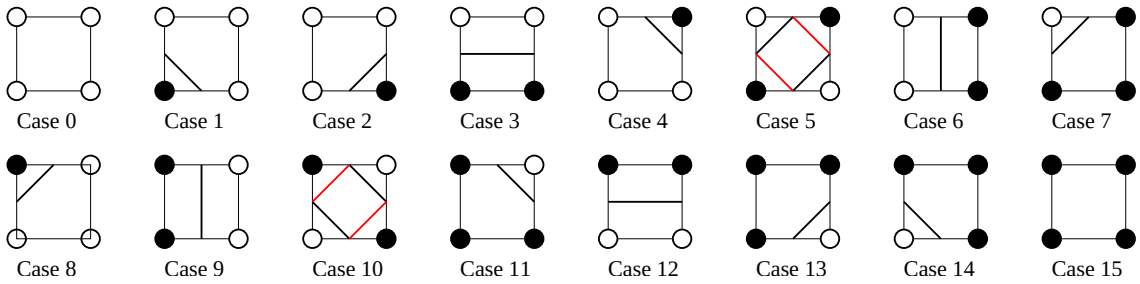


Abbildung 10.7: 16 Fälle beim Marching Square Verfahren

Zwei der sechzehn Fälle sind allerdings nicht eindeutig. Bei Fall 5 und bei Fall 10 ist nicht klar, wie die Linie gezogen werden soll (Abbildung 10.8). Entweder entsteht eine zusammenhängende Isolinie oder zwei einzelne. Da keine eindeutige Zuordnung möglich ist, muss festgelegt werden, wie in einem solchen Fall vorgegangen werden soll.

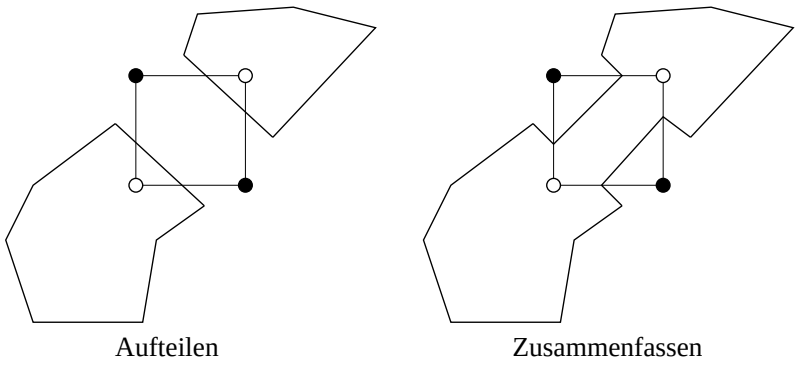


Abbildung 10.8: Nicht eindeutige Fälle beim Marching Square Verfahren

Wenn alle Gitterzellen für einen Isowert der Reihe nach abgearbeitet wurden, erhält man einzelne unzusammenhängende Liniensegmente der zugehörigen Isolinien.

10.2.2 Bewertung

Der Marching Square Algorithmus ist ebenso wie der CONREC Algorithmus sehr effizient in der Zeichengeschwindigkeit und in der Speicherbelastung. Auch hier können sehr große Gitter verarbeitet werden, indem die Gitterzellen einzeln eingelesen werden.

Die Isolinien werden durch einzelne Segmente gezeichnet und liegen nicht als geschlossene Polygone vor. Um ein geschlossenen Linienzug zu bilden, müssten auch hier die einzelnen Segmente aufwändig miteinander verglichen werden.

Die Verfahren Marching Cube und Marching Square sind daher ohne weitere Modifikation nicht für die Erzeugung zusammenhängender Objekte geeignet, sondern stellen effiziente Methoden für das Rendering dar.

10.3 Linefollowing Algorithmen

William V. Snyder beschreibt im Jahr 1978, ausgehend von einem zweidimensionalen Array mit skalaren Werten und der Angabe von Isowerten, allgemein das Zeichnen von Konturlinien bzw. Isolinien [SnyderW]. Er stellt zwei Möglichkeiten vor, um die gesuchten Isolinien zu finden:

- Der Reihe nach wird innerhalb einer Gitterzelle (eingerahmt von vier Punkten) nach Isolinien gesucht. Die gefundenen Teilstücke werden direkt ausgegeben.
- Isolinien werden ausgehend von einem Startpunkt durch das gesamte Raster verfolgt werden, bis sie einen geschlossenen Linienzug bilden oder den Rand des Gitters schneiden.

Die Vorgehensweise im ersten Ansatz unterscheidet sich nicht von Contourplot Verfahren, mit dem keine geschlossenen Polygone und somit auch keine Isoflächen gefunden werden können. Snyder entschied sich für den zweiten Ansatz, um die Anzahl der Plot-Kommandos zu minimieren.

10.3.1 Berechnung der Isolinien

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

1. Das Raster wird zeilenweise durchlaufen. Jede Kante einer Rasterzelle wird auf Schnittpunkten mit Isolinien untersucht. Liegt ein gesuchter Isowert zwischen den Werten zweier Rasterpunkten, existiert ein Schnittpunkt.
2. Falls eine Gitterkante mit einem gesuchten Isowert gefunden wurde, werden die Koordinaten des Schnittpunktes durch lineare Interpolation berechnet. Die Formel lautet für einen Parameterwert $w_{schnitt}$ für die X-Koordinate:
   $x_{schnitt} = x_{start} + (x_{end} - x_{start}) \cdot \frac{w_{schnitt} - w_{start}}{w_{end} - w_{start}}$
   Für Y erfolgt die Berechnung analog. Danach wird vermerkt, dass auf der Gitterkante der Isowert gefunden wurde, damit die Kante nicht erneut untersucht wird.
3. Der Algorithmus untersucht nun die zweite Rasterzelle, die an der Gitterkante mit dem gefundenen Isowert angrenzt, nach einer weiteren Gitterkante mit demselben Isowert. Ist die Kante gefunden wird auch hier der Schnittpunkt wie in Schritt zwei berechnet und die Schnittpunkte werden mit einer geraden Linie verbunden. Dieser Schritt wird wiederholt ausgeführt, bis keine unmarkierte Gitterkante mit dem Isowert gefunden wird (Abbildung 10.9).
4. Die Schritte eins bis drei werden je Isowert solange wiederholt, bis alle Gitterkanten mit einem Schnittpunkt markiert sind.

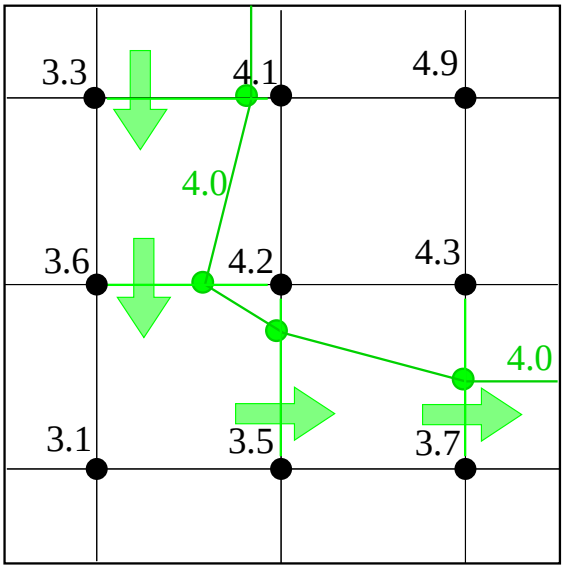


Abbildung 10.9: Linefollowing Algorithmus

Wie bei den Contourplot Algorithmen gibt Snyder die gefundenen Isolinien direkt auf den Bildschirm oder den Plotter aus und erstellt keine programminterne Polygonstruktur, in der die gefundenen Werte vermerkt werden. Dies ist aber einfach zu implementieren, indem die gefundenen Schnittpunkte der Reihe nach abgespeichert werden. Erkennt der Linefollowing Algorithmus, dass das Polygon geschlossen ist, wird ein neues Polygon begonnen. Die einzelnen Polygone werden für jeden Isowert separat gespeichert. Der leicht modifizierte Algorithmus kann somit genutzt werden, um in einem regelmäßigen Gitter zusammenhängende Polygone zu finden. Daher ist der Algorithmus für die vorliegende Arbeit geeignet.

11 Kantenglättung der Isolinien und Isoflächen

Zur Visualisierung von Wetter- und Klimadaten werden hauptsächlich Isolinien und Isoflächen verwendet. Diese werden bei der Vektorisierung in Form von eckigen Polygonen berechnet (Kapitel 10). Eine solche kantige Darstellung ist nicht sehr ästhetisch, vor allem bei einer starken Vergrößerung werden die Ecken sehr deutlich. Für eine harmonischere Darstellung sollen die Polygone geglättet werden. Dabei ist es wichtig die Originalwerte nicht zu sehr zu verändern und dennoch eine ansehnliche Kurve zu erzeugen.

In diesem Kapitel werden zwei Verfahren vorgestellt, mit denen ein Linienzug geglättet werden kann.

11.1 Subdivision Surface Verfahren

Das *Subdivision surface* Verfahren wird in der Computergrafik eingesetzt, um aus beliebigen dreidimensionalen Polyedern glatte Flächen zu bilden. Die Anwendung in der Computergrafik wurde 1978 durch Edwin Catmull und Jim Clark [Catmull], sowie Daniel Doo und Malcolm Sabin [Doo] bekannt. Das Verfahren verwendet eine wiederholte Verfeinerung eines initialen Polyeders, wodurch eine Sequenz mit immer mehr Polygonflächen entsteht, welche gegen eine glatte Oberfläche konvergiert (Abbildung 11.1). Durch das Verfahren wird die Modellierung komplizierter glatter Formen stark vereinfacht.

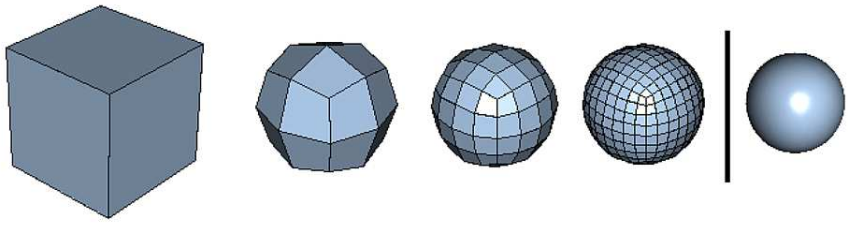


Abbildung 11.1: Ausgehend von einem Würfel, über die ersten drei Schritte der Catmull-Clark Subdivision, hin zum resultierenden Subdivision Surface

Im Wesentlichen werden werden zwei Arten von Subdivision Verfahren unterschieden:

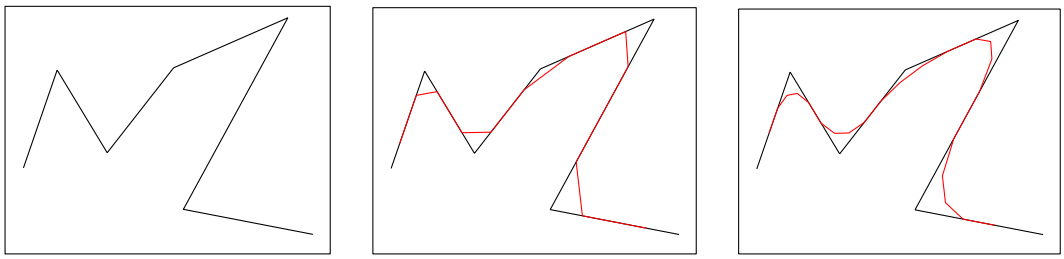
- Interpolative Subdivision: Alte Punkte bleiben erhalten, neue Punkte werden hinzugefügt.
- Approximative Subdivision: Alle Punkte werden neu positioniert.

Aus dem Subdivision Surface Verfahren lässt sich ein Algorithmus für den zweidimensionalen Fall ableiten, der im Folgenden näher betrachtet werden soll.

11.1.1 Subdivision Curve Verfahren

Ein Verfahren zur Glättung von Linienzügen stellt das *Subdivision Curve* Verfahren dar, eine zweidimensionale Vereinfachung des dreidimensionalen Subdivision Surface Verfahrens. Auch hier werden einem Polygonzug durch Unterteilung der Linien immer weitere Punkte hinzugefügt. Die Grundidee zum Subdivision Curve Verfahrens stammt bereits aus dem Jahre 1947 von G. Rahm [Rahm] und führte zum ersten *Corner Cutting* Verfahren von Chaikin im Jahr 1974.

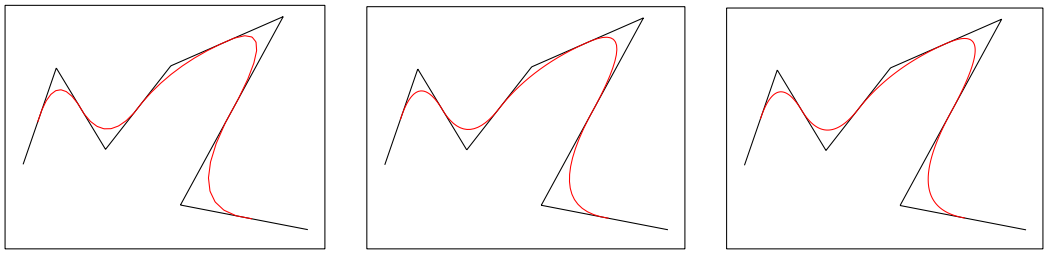
Chaikin verwendet ein stationäres Unterteilungsschema, das heißt in jedem Interationsschritt $k = 1, 2, ...$ wird dieselbe Methode verwendet, um die Strecke zwischen zwei Punkten weiter aufzuteilen (corner cutting). Dazu werden einem Teilstück zwei Zwischenpunkte bei $\frac{1}{4}$ und $\frac{3}{4}$ eingefügt. Der Linienzug wird so bei jedem Iterationsschritt durch weitere Punkte geglättet. Für $k \to \infty$ konvergiert das Verfahren gegen eine quadratische B-Spline Kurve (Abbildung 11.2).



(a) Ungeglätteter Linienzug

(b) 1. Iteration: 12 Eckpunkte

(c) 2. Iteration: 22 Eckpunkte



(d) 3. Iteration: 42 Eckpunkte

(e) 4. Iteration: 82 Eckpunkte

(f) 5. Iteration: 162 Eckpunkte

Abbildung 11.2: Chaikin Verfahren zur Kantenglättung

Das Verfahren von Chaikin gilt als elementares Verfahren. Mit der Zeit wurden weitere Varianten entwickelt, mit denen eine Kurve geglättet werden kann.

Das *4-point-schema* ist ein Interpolationsverfahren welches 1986 von Dubuc [Dubuc] und 1987 von Dyn, Gregory und Levin [Dyn1] präsentiert wurde. Weitere Verfahren sind die Matrix-, Hermit- oder Tensorproduktverfahren. Einen guten Überblick über die unterschiedlichen Vorgehensweisen zur Kantenglättung mittels Subdivision Curve Algorithmen geben Dyn und Levin in [Dyn2].

11.1.2 Bewertung

Da das Subdivision Curve Verfahren die Kurve annähert, indem immer mehr Punkte hinzugefügt werden, ist das Verfahren nur bedingt für den Einsatz mit SVG verwendbar. Es müssen sehr viele Punkte eingefügt werden, um einen guten visuellen Eindruck der Kurve zu erhalten, da vor allem berücksichtigt werden muss, dass die Kurve auch in einer höheren Zoomstufe mit glatten Übergängen dargestellt werden soll. Durch die höhere Anzahl von Punkten wird zum einen die Datenmenge einer SVG Datei wesentlich größer und zum anderen wird der Client mehr belastet, da wesentlich mehr Linienzüge berechnet und gezeichnet werden müssen.

Das Verfahren stellt einen effizienten Algorithmus für die Kantenglättung eines Polygons beim Rendering dar, ist aber aufgrund der angeführten Nachteile nicht ohne weiteres für SVG geeignet. Besser ist es, eine echte Bézierkurve zu berechnen, da man diese direkt in SVG platzsparend codieren kann.

11.2 Berechnung von Bézierkurven

Für die Darstellung eines geglätteten Polygons kommen in SVG lediglich quadratische oder kubische Bézierkurven in Frage, da die Spezifikation des Standards nur diese beiden Arten von Kurven zulässt.

Um die Datenmenge und den Aufwand der Berechnungen auf der Clientseite zu minimieren, sollen lediglich quadratische Bézierkurven Verwendung finden. Daher wurde in einem Projekt an der Universität Osnabrück ein Verfahren zur Kantenglättung mittels Bézierkurven entwickelt [kleine Kruse], welches im Folgenden kurz erläutert werden soll.

11.2.1 Bézierkurven

Eine Bézierkurve wird durch Stützpunkte definiert, wobei die Punkte in Anker- und Kontrollpunkte unterschieden werden. Die Kurve verläuft durch die beiden Endpunkte der Kurve und die Ankerpunkte beeinflussen den Verlauf. Dabei liegt die Kurve in der konvexen Hülle des Kontrollpolygons [Schwarz].

Für $0 \leq t \leq 1, t \in \mathbb{R}; i = 0, ..., n; n \in \mathbb{Z}$ wird eine Bézierkurve n-ter Ordnung mit den Stützpunkten P_i wie folgt definiert:

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

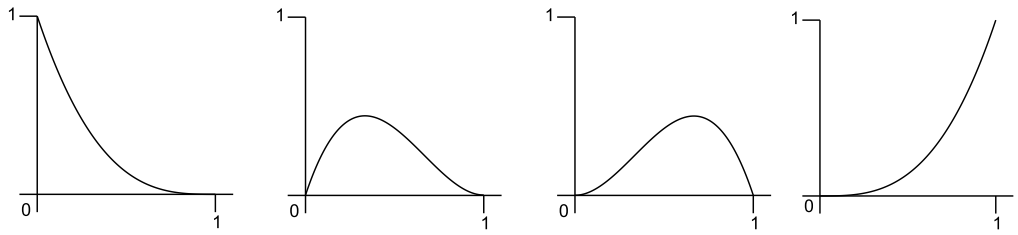
Die B_i entsprechen den Bernsteinpolynomen:

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

Die wichtigsten Eigenschaften der Bernsteinpolynome sind:

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

Die Bernsteinpolynome geben an, wie stark ein Stützpunkt P_i die Kurve beeinflusst (Abbildung 11.3). Dabei verläuft die Kurve durch den Startpunkt P_0 tangential zur Geraden $\overline{P_0P_1}$ und endet im Stützpunkt P_n tangential zur Geraden $\overline{P_{n-1}P_n}$



(a) $B_{0,3}=(1-t)^3$ (b) $B_{1,3}=3t(1-t)^2$ (c) $B_{2,3}=3t^2(1-t)$ (d) $B_{3,3}=t^3$

Abbildung 11.3: Bernsteinpolynome für die Punkte P_0 bis P_3 mit $n = 3$

Zum Zeichnen einer Bézierkurve bietet sich der Algorithmus von de Casteljau [Schwarz] an. Es handelt sich um ein Iterationsverfahren, welches durch wiederholtes Teilen von Linien einen Kurvenpunkt approximieren kann. Es gilt

$$P(t)_i^{n+1} = (1-t) \cdot P(t)_i^n + t \cdot P(t)_{i+1}^n$$

wobei n den Iterationsschritt und i den jeweiligen Index des Punktes angibt. Der Parameter $t \in [0; 1]$ gibt den Teilungsfaktor an. Der Algorithmus ist in der Abbildung 11.4 für $t = 0.5$ dargestellt.

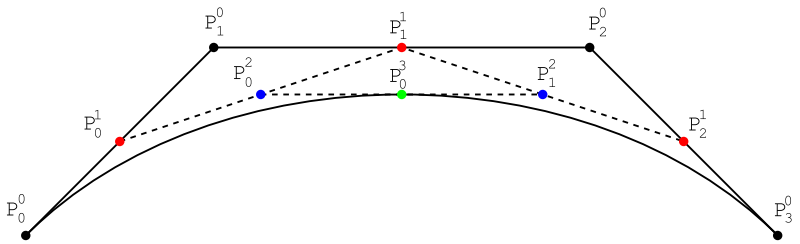


Abbildung 11.4: Algorithmus von de Casteljau für $t = 0.5$

Die Berechnung einer Bézierkurve ist sehr einfach und kann schnell durchgeführt werden. Da die Bézierkurven keine hohen Ansprüche an den Client stellen, sind sie sehr gut für die Kurvendarstellung in SVG geeignet.

11.2.2 Kriterien für quadratische Bézierkurven

Da die Berechnung der Isolinien und -flächen automatisch erfolgen soll und auch die Kantenglättung ohne manuellen Eingriff möglich sein soll, müssen besondere Anforderungen an die Kantenglättung gestellt werden. So dürfen die geglätteten Kurven nicht zu stark vom eigentlichen Polygonzug abweichen, da sonst die Lage der Linien zu stark vom ursprünglichen Polygon entfernt wäre. Außerdem könnten sich eng beieinander liegende Linien bei einer zu großen Abweichung überschneiden, was bei zwei unterschiedlichen Isolinien nicht vorkommen darf. Daher muss die Möglichkeit bestehen, über Parameter den Grad der Glättung zu beeinflussen.

Damit der visuelle Eindruck einer Isofläche gewahrt bleibt, sollte die eingeschlossene Fläche bei der Verwendung einer Bézierkurve oder eines Polygons als Flächenbegrenzung möglichst gleich sein. Wäre dies nicht der Fall, kann es dazu kommen, dass eine Isofläche nach der Glättung wesentlich mehr Fläche einnimmt als erforderlich, wodurch dieser Messwert hervorgehoben werden würde. Ebenso verhält es sich im umgekehrten Fall. Der Kurvenverlauf selbst stellt ein weiteres Kriterium dar. Bei einem konvexen Polygon sollte die Bézierkurve entlang einer Kante nach außen gedrückt werden. Dies vergrößert aber den Flächeninhalt. Daher muss die Kurve an den Ecken des Polygons nach innen gedrückt werden. Würde die Kurve nicht nach außen, sondern nach innen gedrückt werden, würde die Kurve um das Polygon schwingen, um den Flächeninhalt auszugleichen und die Darstellung wäre nicht sehr ästhetisch (Abbildung 11.5).

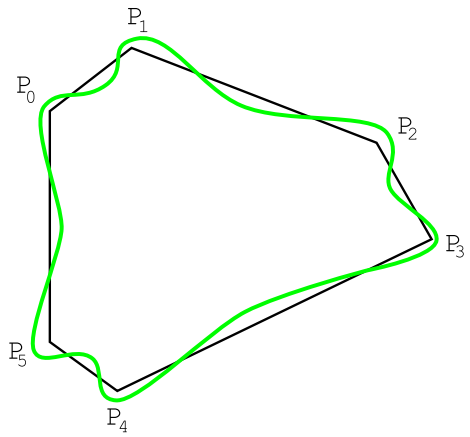


Abbildung 11.5: Schwingende Bézierkurve

Ist das Polygon konkav, werden für die Glättung nur einzelne konvexe Teile des Polygons betrachtet, so dass auch hier das letzte Kriterium angewendet werden kann. Ein weiteres Kriterium stellt die Abweichung der Kurve zum Polygon dar. Diese Abweichung ist erforderlich, da das Polygon sonst nicht geglättet werden kann. Allerdings sollte die Abweichung minimal gehalten werden, um möglichst den ursprünglichen Verlauf beibehalten zu können. Ein Polygon besteht aus einer Menge von Kanten. Um den Kurvenverlauf möglichst genau zu approximieren, wird jede Kante durch eine Bézierkurve angenähert. Die Kombination aller Bézierkurven ergibt dann die geglättete Kurve.

11.2.3 Vorgehensweise

Anhand der aufgestellten Kriterien können Gleichungen aufgestellt werden, die den allgemeinen Verlauf der Bézierkurven im Verhältnis zu den Polygonpunkten beschreiben. Für jede einzelne Polygonkante wird eine quadratische Bézierkurve definiert, so dass diese stetig ineinander übergehen. Die Endpunkte P_{t1} und P_{t2} der Polygonkante werden in das innere des Polygons verschoben und stellen die Ankerpunkte der Bézierkurve dar. Die Eckpunkte werden dabei auf den Winkelhalbierenden V_{t1} und V_{t2} des Winkels verschoben, der von zwei Polygonkanten eingeschlossen wird. Um einen stetigen Übergang zu erhalten müssen die zu einem Ankerpunkt benachbarten Kontrollpunkte auf einer Geraden liegen. Als Gerade wird eine Senkrechte zur Winkelhalbierenden im Ankerpunkt gewählt, die durch die Vektoren V_1 beziehungsweise V_2 definiert ist. Eine schematische Darstellung einer solchen Bézierkurve ist in Abbildung 11.6 zu sehen.

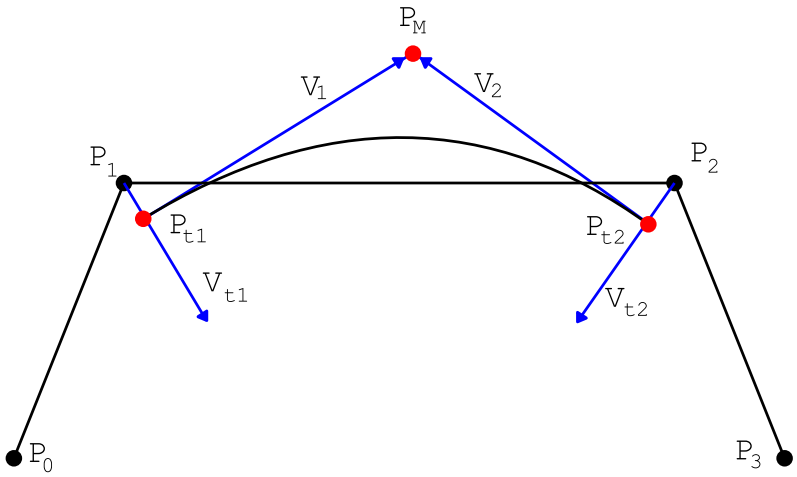


Abbildung 11.6: Schematische Darstellung der Parameter

Die Gleichungen für die Endpunkte der Bézierkurve können in Abhängigkeit der Polygonpunkte und der Winkelhalbierenden wie folgt angegeben werden:

$$P_{t1} = P_1 + t_1 \cdot V_{t1} \tag{1}$$
$$P_{t2} = P_2 + t_2 \cdot V_{t2} \tag{2}$$

Die Variablen t_1 und t_2 geben den Grad der Verschiebung in Richtung der normierten Vektoren V_{t1} und V_{t2} an. Da eine Verschiebung der Eckpunkte nach außen ausgeschlossen ist, dürfen die t_i nicht negativ werden.

Der Punkt P_M stellt den Kontrollpunkt der Bézierkurve dar und kann wie folgt definiert werden:

$$P_M = P_{t1} + s_1 \cdot V_1 \tag{3}$$
$$P_M = P_{t2} + r_2 \cdot V_2 \tag{4}$$

Die Variablen s_1 und r_2 geben den Grad der Verschiebung in Richtung der normierten Vektoren V_1 und V_2 an.

Nach dem Algorithmus von de Casteljau kann so die Gleichung für eine Bézierkurve definiert werden:

$$B_1(x) = P_{t1} + x \cdot s_1 \cdot V_1 \tag{5}$$
$$B_2(x) = P_m + x \cdot r_2 \cdot V_2 \tag{6}$$
$$C(x) = B_1(x) + x \cdot (B_2(x) - B_1(x)), x \in [0, 1] \tag{7}$$

Um den Abstand zwischen dem Polygon und der geglätteten Kurve zu definieren, wird folgende Gleichung verwendet:

$$P_1 + e_1 \cdot (P_2 - P_1) = C(x_1) + d_1 \cdot D \cdot N(P_2 - P_1) \tag{8}$$

Dabei entspricht $N(v)$ der Normierung des Vektors v auf die Länge 1 und D stellt eine Matrix dar, mit der ein Vektor um 90^o im Uhrzeigersinn gedreht wird. Mit d_1 wird der Abstand zwischen approximierten Kurvenpunkt und Polygonkante bezeichnet. Der Parameter e_1 gibt die Entfernung zum Punkt P_1 an, in der d_1 gemessen wird (Abbildung 11.7).

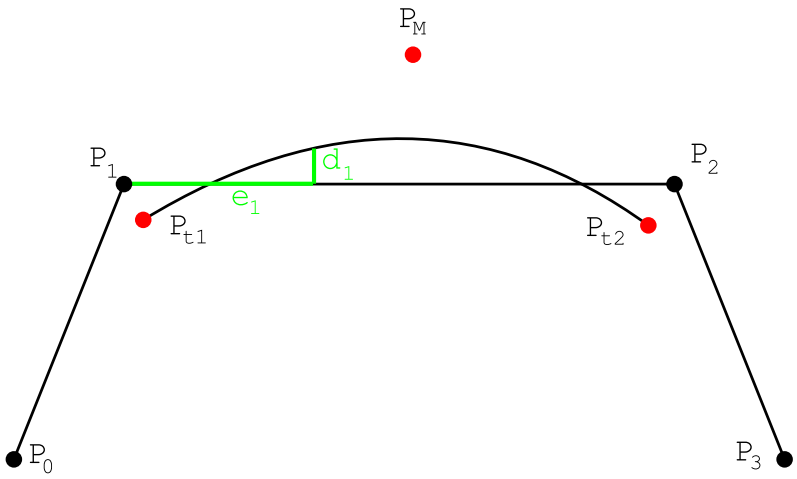
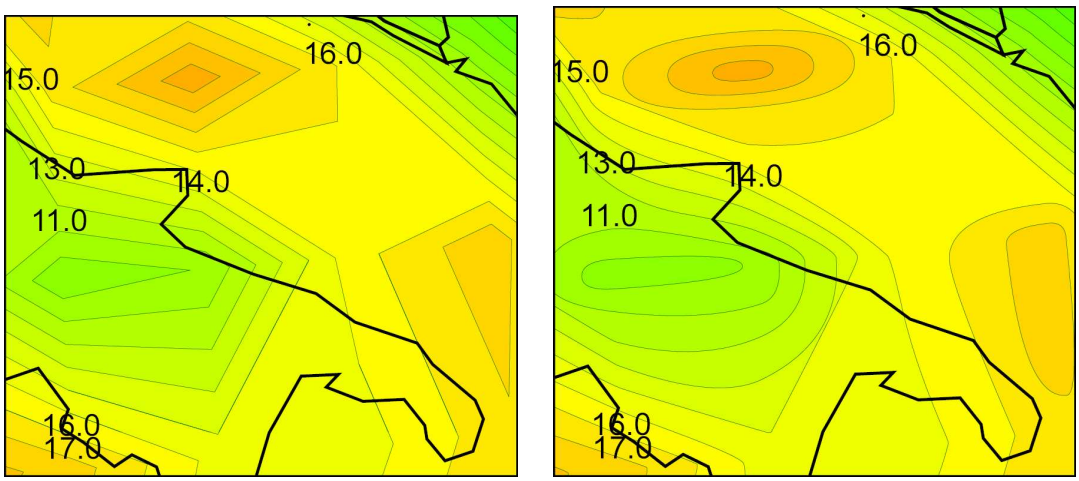


Abbildung 11.7: Parameter für die Abstandsbestimmung

Die Gleichungen werden je Kante aufgestellt und beschreiben die elementare Bézierkurve, die diese Kante approximiert. Die Gleichungen 4 und 8 werden dann in einem linearen Gleichungssystem zusammengefasst, mit dem die Bézierkurve für das ganze Polygon beschrieben wird. Das Gleichungssystem wird nach s_1, e_1, r_2 und d_1 aufgelöst. Ergebnis sind vier Gleichungen in denen die Parameter in Abhängigkeit von t_1, t_2 und x beschrieben werden. Da die Gleichungen sehr lang sind, soll hier auf eine genaue Angabe verzichtet werden. Zu finden sind die Gleichungen in [kleine Kruse]. Die Parameterwerte werden dann über einen Simplex Algorithmus [Teschl] optimiert, so dass eine nach den vorgegebenen Kriterien optimale Bézierkurve entsteht (Abbildung 11.8).



(a) Ungeglättete Polygone (b) Geglättete Polygone

Abbildung 11.8: Temperatur-Isoflächen vor und nach der Kantenglättung

Da die Kantenglättung automatisiert abläuft kann nicht immer garantiert werden, dass die Isolinien und -flächen nach der Glättung überschneidungsfrei bleiben.

Einen Überblick über die Klassen und Methoden zur Kantenglättung ist im Anhang C.5 zu finden.

12 Aufbau der SVG Web Mapping Applikation

Bei der SVG Web Mapping Applikation handelt es sich um eine komplexe Anwendung, die eines strukturierten Aufbaus bedarf. Hierbei sollen die Konzepte, die bereits in Kapitel 4.6 erläutert wurden, zum Einsatz kommen. Vor allem das Template Konzept ist von besonderer Bedeutung, da sich die darzustellenden Daten oft ändern und deshalb nicht in die Anwendung direkt integriert werden sollen. Statt dessen sollen die eigentlichen Daten separat vorliegen, jeweils eingeteilt in einzelne Layer und unterschiedliche Zeitpunkte. Das Template der Anwendung sieht, wie in Abbildung 12.1 zu sehen, aus.

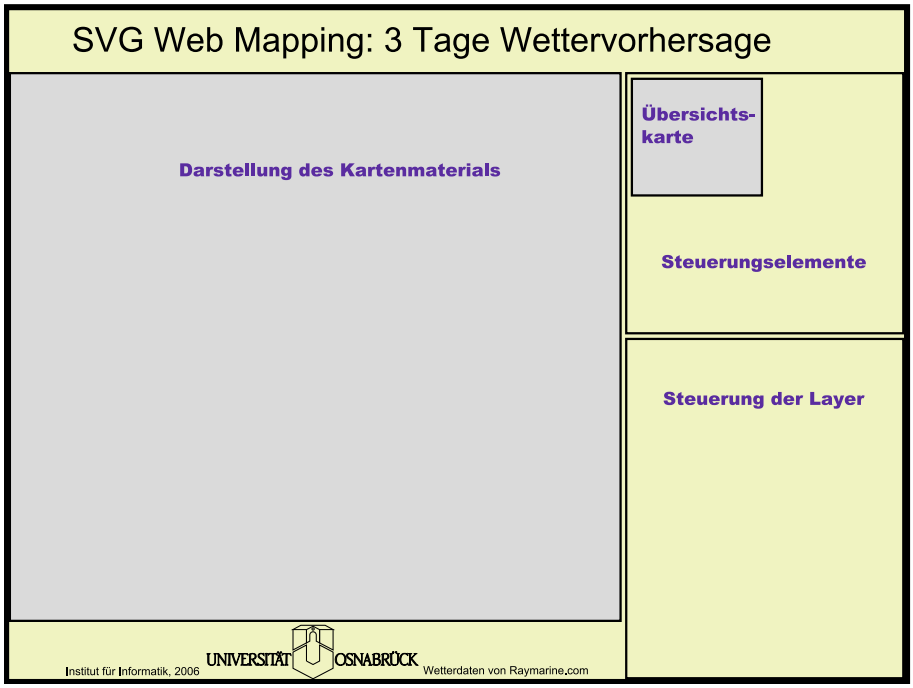


Abbildung 12.1: Template der Web Mapping Applikation

Das Template selbst besitzt keinerlei Programmlogik. Es dient nur als Container für die einzeln nachzuladenden Elemente. Es gibt lediglich die Anordnung der einzelnen Elemente vor und beinhaltet gegebenenfalls zusätzliche textuelle Informationen. Außerdem kann das Design des Templates beliebig verändert werden. Da die Platzhalter mit IDs versehen sind, können die eigentlichen Daten und die Steuerelemente, sowie die in JavaScript implementierte Programmlogik zur Laufzeit nachgeladen werden.

12.1 Steuerung der Anwendung

Durch Schaltelemente soll der Anwender in die Lage versetzt werden, mit dem Kartenmaterial arbeiten zu können. Wichtig hierbei ist, dass der Anwender nur über die angegebenen Steuerelemente die Karte verändern darf und die Anwendung nicht anderweitig beeinflussen kann, was beispielsweise beim Adobe SVG Viewer (ASV) prinzipiell der Fall wäre. Der ASV bietet die Möglichkeit, das gesamte SVG zu vergrößern oder zu

verschieben. Würde der Anwender diese Funktionalität nutzen, ist der Anwendung nicht mehr bekannt, welche Ausschnitte zu sehen sind und sie hätte nicht mehr die Kontrolle über das angezeigte Kartenmaterial. Daher muss die Funktionalität des Zoomens beim Adobe SVG Viewer abgeschaltet werden. Dazu kann im `svg`-Tag der Wert des Attributs `zoomAndPan` auf `disable` gesetzt werden.

Der ASV bietet die Möglichkeit ein eigenes Kontextmenü zu definieren [XML] (Quellcode 12.1). Dies wird genutzt, um eigene Menüpunkte hinzuzufügen, mit denen die gesamte SVG Anwendung vergrößert oder verkleinert oder eine kurze Information zur Anwendung eingeblendet werden kann. In dem `menu`-Tag werden die einzelnen Einträge definiert. Mit dem Tag `separator` wird eine horizontale Linie eingefügt, um das Menü besser strukturieren zu können. Das Tag `item` definiert einen Menüeintrag. Innerhalb des Tags ist der Text angegeben, der im Menü erscheinen soll. Als Attribut wird angegeben, was bei der Auswahl des Menüeintrages passieren soll. Dabei wird das Attribut `action` für die von Adobe vordefinierten Aktionen verwendet. Um eine eigene JavaScript Funktion aufrufen zu können, wird das Attribut `onactivate` mit der Angabe des Funktionsnamens verwendet.

```
<menu id="MyMenu">
  <header>Adobe SVG Viewer</header>

  <separator/>
  <item action="Open">Öffnen</item>
  <item action="OpenNew">Öffnen in neuem Fenster</item>

  <separator/>
  <item action="Quality">Höhere Qualität</item>
  <item onactivate="scale(1.1);">Vergrößern</item>
  <item onactivate="scale(0.9);">Verkleinern</item>

  <separator/>
  <item action="Help">Hilfe</item>
  <item action="About">Über SVGViewer...</item>

  <separator/>
  <item onactivate="aboutInfo();">Info</item>

</menu>
```

Quellcode 12.1: Definition eines eigenen Kontextmenüs

Damit das Menü beim Klick mit der rechten Maustaste erscheint, muss das alte entfernt und das neue eingehängt werden. Je nach Version des ASV muss dabei anders vorgegangen werden. In der JavaScript Funktion `setMenu(menuid)` (Quellcode 12.2) wird dazu zunächst ermittelt, ob die Funktion `getSVGViewerVersion` vorhanden ist. Falls ja, wird mit dieser Funktion die Version des ASV bestimmt und in Abhängigkeit der Versionsnummer das neue Menü gesetzt.

```
function setMenu(menuid){
if(window.getSVGViewerVersion) {
  var ver=getSVGViewerVersion();
   if(ver.indexOf("Adobe")!=-1) {
      var newMenu=parseXML(printNode(getDocument().
                    getElementById(menuid)),getContextMenu());
    if(ver.indexOf("6.0")!=-1) {
      getContextMenu().replaceChild(newMenu.firstChild,
                  getContextMenu().firstChild);
     }
      else {
      getContextMenu().replaceChild(newMenu,
                  getContextMenu().firstChild);
     }
    }
  }
}
```

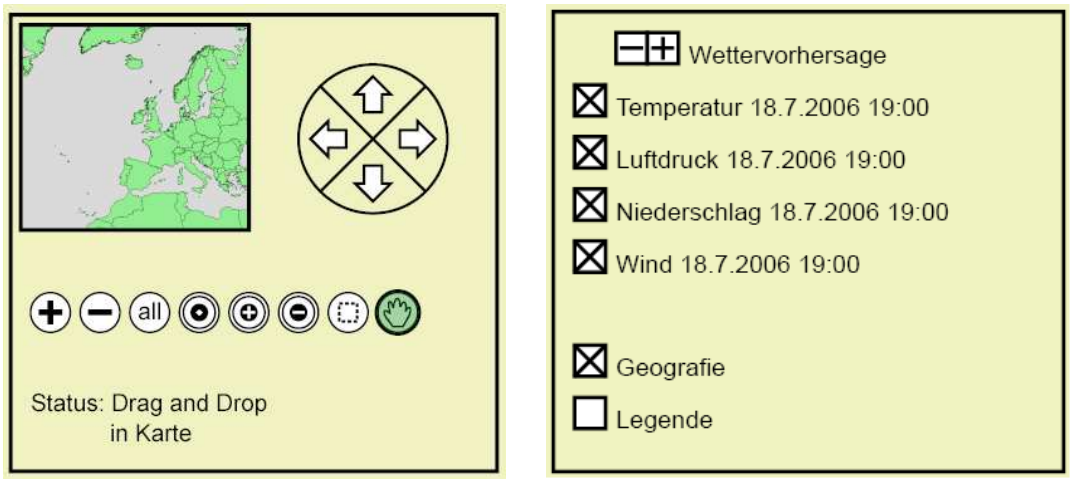
Quellcode 12.2: Setzen des eigenen Kontextmenüs

In der Anwendung müssen dann geeignete Steuerelemente eingefügt werden. Dazu werden Schaltflächen für die folgenden Aktionen eingefügt:

- Verschieben in Nord-, Süd-, Ost- und West-Richtung
- Verschieben der Karte per Mauszeiger (Panning)
- Schrittweises Ein- und Auszoomen
- Ein- und Auszoomen anhand eines neuen Kartenmittelpunktes
- Auswahl eines Zoomrechteckes
- Anzeigen der kompletten Karte
- Wählen eines neuen Kartenzentrums

Die Steuerelemente sind in der Abbildung 12.2(a) zu sehen. Die Schaltflächen werden mit einem `onclick` Attribut versehen, in dem die JavaScript Funktion angegeben ist, welche beim Aktivieren der Schaltfläche ausgeführt wird. Alle Funktionen beruhen auf einer Veränderung der Viewbox der angezeigten Karten. Dazu ist im Template ein `svg`-Tag mit eigener Viewbox vorhanden. Die Viewboxkoordinaten sind global im JavaScript Code vorhanden und werden beim Zooming oder Panning neu berechnet und auf Gültigkeit überprüft. Sind die neu berechneten Koordinaten gültig werden sie gesetzt. Ansonsten werden sie entweder auf gültige Werte gesetzt oder es wird eine Fehlermeldung eingeblendet. Bei einem Neusetzen der Viewbox kann es zudem passieren, dass neue Informationen nachgeladen oder bereits vorhandene Daten entfernt werden müssen. Dazu wird eine Serveranfrage gestellt, die die notwendigen Aktionen zurückliefert. Der Mechanismus wird in Abschnitt 12.3 näher beschrieben.

Neben der Navigation innerhalb des Kartenmaterials muss der Anwender die Möglichkeit haben, einzelne Layer gezielt sichtbar oder unsichtbar zu machen, um beliebige Kombinationen der in den Layern vorhandenen Daten zu erhalten. Dazu wird ein Layer in einem `g`-Tag abgelegt. Das `g` Tag besitzt eine eindeutige ID, über die es angesprochen werden kann und das Attribut `visibility`, das angibt, ob der Layer angezeigt wird oder nicht. Neben den bereits erläuterten Schaltflächen werden zusätzliche Checkboxen je Layer eingefügt (Abbildung 12.2(b)). Wird eine Checkbox aktiviert oder deaktiviert wird das Attribut `visibility` entsprechend geändert (Quellcode 12.3).



(a) Kartennavigation (b) Layerauswahl

Abbildung 12.2: Steuerelemente der Web Mapping Applikation

```
function changeVisibility(name){
  var group=document.getElementById(name);
    if(group.getAttributeNS(null,'visibility')=='hidden')
      group.setAttributeNS(null,'visibility','visible');
    else
      group.setAttributeNS(null,'visibility','hidden');
}

function changeCross(name){
  var cross = document.getElementById(name+'_cross');
  if(cross.getAttributeNS(null,'visibility')=='hidden')
    cross.setAttributeNS(null,'visibility','visible');
  else
    cross.setAttributeNS(null,'visibility','hidden');
}
```

Quellcode 12.3: Ein- und Ausschalten eines Layers

12.2 Effizientes Nachladen von Daten

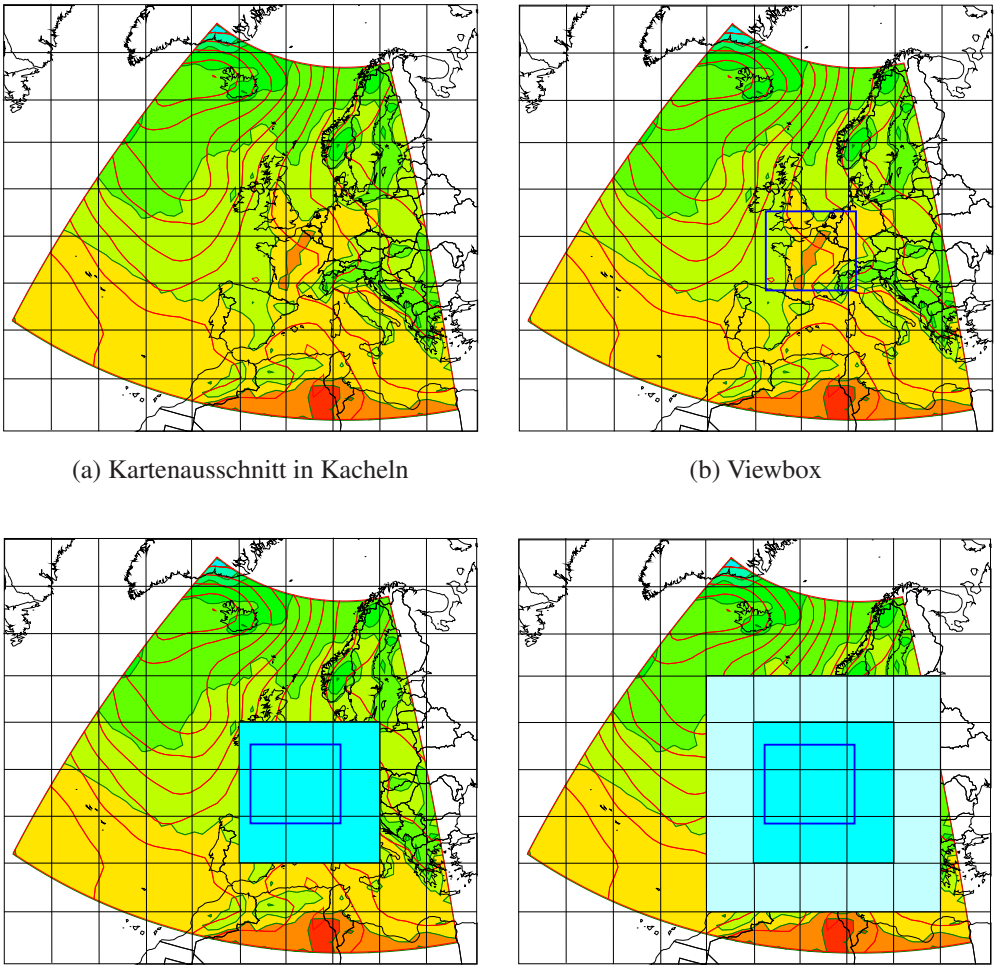
Es ist wichtig, dass die Daten schnell übertragen werden können und dass der Client nur mit den wirklich notwendigen Daten belastet wird. Daher ist die Strukturierung der Daten von besonderer Bedeutung. Um gezielt Daten vom Server laden zu können, sind sie in einzelne Layer und in verschiedene Zeitpunkte eingeteilt. Dadurch kann gezielt ein Datensatz eines bestimmten Zeitpunktes geladen werden. Ein solcher Datensatz kann weiter unterteilt werden. So werden die Karten in einzelne Kacheln mit unterschiedlicher Auflösung der Daten eingeteilt. Dabei existiert ein Datensatz je Layer und Zeitpunkt, der die gesamte Karte umfasst, allerdings nur eine geringe Auflösung der Daten aufweist. Die Karte wird dann in kleinere Kacheln zerlegt, die zusätzliche Informationen enthalten. Diese Kacheln können bei Bedarf wiederum weiter zerlegt werden. Das hat zur Folge, dass der Anwender zunächst einen groben Überblick über die Karte erhält. Wird eine Region vergrößert werden dafür genauere Informationen eingeblendet. Dabei kann unterschieden werden, ob lediglich neue Informationen hinzugeladen werden, oder ob die alten Daten ersetzt (also gelöscht) werden sollen.

Es werden nicht alle Kacheln geladen, sondern nur die tatsächlich benötigten. Hat der Anwender eine Region vergrößert, müssen natürlich alle Kacheln geladen werden, die in dieser Region komplett oder teilweise zu sehen sind. Neben diesen Kacheln werden noch alle Kacheln geladen, die sie umgeben. Dies hat den Vorteil, dass bei einem Verschieben der Karte, die angrenzenden Teile bereits geladen wurden und keine Zeitverzögerung durch das Nachladen spürbar wird. Abbildung 12.3 veranschaulicht das Konzept. Bewegt der Anwender die Karte beispielsweise nach links, sind dort schon geladene Kacheln vorhanden und werden sofort angezeigt. Die daran links angrenzenden Kacheln werden dann bereits im Hintergrund geladen und die Kacheln auf der rechten Seite werden entfernt. Dies erhöht zwar den Datenverkehr, beschleunigt die Anwendung aber wesentlich.

Die Kombination aus Zooming und Kachelung der Daten hat zwei entscheidende Vorteile. Als erstes wird der Anwender nicht mit zu vielen Informationen überfordert. Er kann sich zunächst in Ruhe einen Überblick verschaffen und sich dann entscheiden, für welche Region er tiefergehende Informationen sehen möchte. Der weitere Vorteil ist die geringe Datenmenge bei der Datenübertragung und der Anzeige im Client. Da die höher aufgelösten Informationen nur teilweise geladen werden müssen, wird der Client weniger belastet und durch die geringere Datenmenge ist die Datenübertragung wesentlich schneller. Werden die Daten nur hinzugeladen, also nicht ersetzt, spart man außerdem das redundante Nachladen von Informationen.

12.3 Ermitteln der zu ladenden Daten

Die Daten werden auf dem Server verwaltet, wobei der Client keine Informationen über die Struktur der Daten auf dem Server kennt. Der Client selbst schickt an den Server lediglich eine Anfrage mit den Angaben, welcher Ausschnitt zu sehen ist und der Server ermittelt daraus, welche Daten der Client erhalten muss und welche gegebenenfalls aus dem Client entfernt werden müssen. Dies hat den Vorteil, dass die Datenhaltung auf dem Server beliebig geändert werden kann, ohne dass der Client angepasst werden muss.



(a) Kartenausschnitt in Kacheln

(b) Viewbox

(c) Kacheln, die geladen werden müssen

(d) Kacheln, die geladen werden

Abbildung 12.3: Die zu ladenden Kacheln

12.3.1 Serverseitige Datenhaltung

Auf dem Server werden die Daten in einzelne Dateien abgespeichert. Je Layer und Zeitpunkt existieren dabei einzelne Dateien mit einem eindeutigen Namen. Der Name wird durch die Kodierung der Kachel und der Zoomstufe erweitert und endet mit der Dateiendung `.psvg`. Die Dateiendung soll suggerieren, dass nicht ein komplettes SVG Dokument enthalten ist, sondern nur der Teil (engl. *part*) innerhalb des `svg` Wurzelelementes. Die Datei `country_1_3_4.psvg` enthält also die Daten für die Kachel des Layers `country` in der Zoomstufe `1` aus der dritten Spalte und vierten Zeile.

Neben den Dateien mit den eigentlichen Daten sind weitere XML Dateien vorhanden, die je Layer angeben, ob die Daten ersetzt oder hinzugefügt werden sollen. In diesen Dateien ist lediglich ein `g`-Tag mit einem Attribut `action` enthalten, welches den Wert `add` oder `replace` annehmen kann. Auch diese Dateien besitzen einen klar strukturierten Dateinamen. So enthält die Datei `country_1_action.xml` die Informationen

darüber, wie bei einem Wechsel von Zoomstufe `0` zur Zoomstufe `1` vorgegangen werden muss.

Die Datenhaltung in Dateien scheint auf den ersten Blick etwas unkonventionell, da sich die Verwaltung der Daten mit einer Datenbank anzubieten scheint. Die Verwendung einzelner Dateien hat allerdings gegenüber der Datenbanklösung einige Vorteile. So können die Daten ohne Datenbankzugriff direkt geladen werden. Außerdem muss für die Verwendung der Web Mapping Anwendung nicht erst eine Datenbank installiert werden.

12.3.2 Berechnung der Daten für einen Kartenausschnitt

Welche Daten geladen oder entfernt werden, wird serverseitig mit einem PHP Skript berechnet. Dazu muss der Client die alten und die neuen Viewboxkoordinaten an den Server übermitteln. Dieser kann aus den Koordinaten die alte und die aktuelle Zoomstufe und die sichtbaren Kacheln ermitteln. Der Server generiert die benötigten Dateinamen und ermittelt über die `action`-Datei, ob die Daten hinzugeladen oder andere Daten ersetzt werden sollen. Dazu wird folgendes XML Grundgerüst an den Client übermittelt:

```
<data>
  <delete></delete>
  <append></append>
</data>
```

Im `delete`-Tag werden die IDs der zu löschenden Elemente angegeben. Im `append`-Tag wird angegeben, was neu hinzugefügt werden muss. Dabei sind zwei Varianten möglich. In der ersten werden bereits alle Kartendaten ausgelesen und dem Clienten zurückgeliefert. Dabei wird im `append`-Tag das Attribut `parent` angegeben, damit auf Clientseite klar ist, wo die Daten in den clientseitigen DOM-Tree eingehängt werden müssen. In der zweiten Variante werden lediglich die Dateinamen und die zugehörige ID des Elternknotens übermittelt und der Client lädt die Daten selbst nach.

Beide Varianten haben ihre Vor- und Nachteile. Werden wie im ersten Fall alle Daten auf einmal übertragen, ist die Datenübertragung insgesamt meist schneller als bei den vielen einzelnen Anfragen in der zweiten Variante. Andererseits können die Daten bei der ersten Variante erst in den DOM-Tree eingehängt und somit angezeigt werden, wenn alle Daten komplett übertragen wurden. Bei der zweiten Variante können die einzelnen Datensätze sofort dargestellt werden, was vor allem bei geringerer Bandbreite vorteilhaft ist.

13 Programm zur Erstellung einer SVG Web Mapping Applikation

In der Web Mapping Applikation werden alle vorhergehend beschriebenen Module zusammengeführt. Um einen Einblick in die Struktur der Datenhaltung und den Ablauf beim Erstellen einer Web Mapping Applikation zu erläutern, werden einige Klassen näher vorgestellt. Es werden unter anderem Klassen benötigt, die die ausgelesenen Daten verwalten, die das Konfigurationsfile einlesen und Klassen, die das Schreiben der Daten übernehmen.

Die Darstellung der Geographie war Thema der Diplomarbeit von Dorothee Langfeld. Nähere Informationen zur Visualisierung geographischer Daten sind in [Langfeld] zu finden. Im Folgenden wird das Java Programm zur Erstellung der SVG Web Mapping Applikation näher betrachtet. Dabei wird der Schwerpunkt auf die Visualisierung von Wetterdaten gelegt.

13.1 Die Konfigurationsdatei

Die Erstellung einer Web Mapping Applikation soll ohne händische Nachbearbeitung möglich sein. Dazu muss in maschinenlesbarer Form das äußere Erscheinungsbild der Anwendung festgelegt werden. Diese Angaben sollen in einer Konfigurationsdatei abgespeichert werden, die das Programm einliest und die Anwendung entsprechend den Vorgaben erzeugt. Die Klassen zur Verwaltung des Konfigurationsfiles sind im Anhang C.6 dargestellt.

Für die Konfigurationsdatei wird XML verwendet, da XML einfach zu verarbeiten ist. Mit geeigneten Programmen kann direkt bei der Erstellung der Datei überprüft werden, ob die Angaben formal korrekt sind. Das Format der Konfigurationsdatei könnte mit einer DTD oder mit XML-Schema definiert werden [Harold]. In diesem Falle kommt XML-Schema zum Einsatz, da hier Datentypen definiert werden können, was in einer DTD nicht möglich ist. Dadurch können Attributwerte besser eingegrenzt werden, was die fehlerfreie Erstellung und die spätere Validierung einer solchen Datei erleichtert. Im Folgenden sollen einige wichtige Tags zur Erstellung von Kartenmaterial herausgegriffen und beschrieben werden. Die komplette Schemadefinition ist im Anhang D zu finden.

Das Wurzelelement der Konfigurationsdatei ist das `map`-Tag, in dem als erstes das Element `configuration` enthalten sein muss. In diesem Tag werden einige Grundeinstellungen für die Anwendung und für den Anzeigebereich des Kartenmaterials vorgenommen (Quellcode 13.1).

Mit `zoomAndPan="disable"` wird das Zooming und das Verschieben über den Adobe SVG Viewer abgeschaltet (Kapitel 12.1). Das Verzeichnis, in das die erzeugten Dateien geschrieben werden, wird im Attribut `outputfolder` angegeben. Mit den Attributen `strokecolor`, `symbolfillcolor`, `symbolhighlightcolor` und `textcolor` werden Standardwerte für die Farbgebung definiert.

```
<configuration
  zoomAndPan="disable"
  outputfolder="euro3day"
  strokecolor="black" symbolfillcolor="white"
  symbolhighlightcolor="green" textcolor="black"
  script="v1"
  converterclass="util.LambertConvert"
  title="3 Tage Wettervorhersage"
  legendFile="legends/legende.svg"
  legendX="670" legendY="390"
  layout="layout/eurowetter.xml" simplify="50">
    <viewbox vbX="-35.5" vbY="-70.5"
             vbWidth="61" vbHeight="41" />
</configuration>
```

Quellcode 13.1: `configuration`-Tag der Konfigurationsdatei

Wie in Kapitel 12.3.2 beschrieben, gibt es für den Client zwei verschiedene Möglichkeiten, die erforderlichen Daten zu Laden. Wird im Attribut `script` der Wert `v1` angegeben, werden alle benötigten Daten vom Server zu einer XML Datei kombiniert und an den Client übergeben. Wird der Wert `v2` angegeben, liefert der Server lediglich die Dateinamen an den Client, welcher dann die benötigten Dateien einzeln lädt.

Um geographische Daten in unterschiedliche Projektionen zu überführen (Kapitel 6), kann eine Javaklasse angegeben werden, die die Koordinaten entsprechend umrechnet. Neben dem Titel der Anwendung kann auch eine Legende angegeben und positioniert werden. Das Layout der Oberfläche kann durch eine weitere Konfigurationsdatei angegeben werden, die im Attribut `layout` definiert wird. Mit dem Attribut `simplify` kann angegeben werden, inwieweit Liniensegmente vereinfacht werden sollen. Vereinfachte Polygone spiegeln zwar nicht mehr die genaue geographischen Gegebenheiten wieder, sind dafür aber wesentlich platzsparender. Sinnvoll ist diese Angabe, wenn sehr hochaufgelöste Daten vorliegen, aber nur eine niedrigere Auflösung notwendig ist. Innerhalb des `configuration`-Tags wird dann noch die initiale `viewbox` angegeben. Dies ist der sichtbare Ausschnitt der angezeigten Daten, wobei der linke obere Punkt und die Breite und Höhe als Attribute angegeben werden.

In der Web Mapping Applikation ist eine kleine Kartenübersicht eingefügt, in der die Karte immer als ganzes zu sehen sein sollte und in der der aktuell sichtbare Bereich eingezeichnet ist. Diese Übersichtskarte wird mit dem Tag `overview` näher spezifiziert (Quellcode 13.2).

Auch hier kann ein Wert für die Vereinfachung der Geometrie angegeben werden. Dies ist hierbei besonders wichtig, da die Übersichtskarte nur einen groben Eindruck der Karte liefern muss und dabei möglichst platzsparend sein soll. Im `selection`-Tag wird angegeben, aus welchem Shapefile die Daten für die Übersichtskarte genommen werden sollen und im Tag `layout` wird das Erscheinungsbild der Daten aus dem Shapefile angegeben. In diesem Fall sollen sie als Polygon mit einer schwarzen Linie der Dicke `10` und einer grünen Füllung dargestellt werden.

```
<overview simplify="100">
  <selection sourcefile="input/cntry02">
    <layout xsi:type="Polygon"
            stroke="black"
            stroke-width="10"
            stroke-linecap="round"
            fill="lightgreen"/>
  </selection>
</overview>
```

Quellcode 13.2: `overview`-Tag der Konfigurationsdatei

Im Folgenden werden in der Konfigurationsdatei die einzelnen Layer definiert. Dazu gibt es drei unterschiedliche Typen von `layer`-Tags: `multi`, `group` und `single`. In einem `single`-Layer sind die Daten für eine Informationsebene zu einem bestimmten Zeitpunkt angegeben. In einem `group`-Layer werden alle Ebenen zusammengefasst, die denselben Datentyp besitzen (beispielsweise die Temperatur) und zu unterschiedlichen Zeitpunkten gültig sind. In dem umfassenden `group`-Layer werden dann alle Ebenen zusammengefasst, die sich bei einem Wechsel des Zeitpunktes gleichzeitig ändern sollen. Alle Layer müssen mit einer eindeutigen ID und können mit einem Label zur Beschreibung versehen werden.

Innerhalb des `single`-Layer Tags werden alle notwendigen Daten für diese Informationsebene angegeben. Da sich je nach Zoomstufe die Auflösung eines Layers ändern kann, folgen als erstes `zoomstep`-Tags, in denen das Erscheinungsbild je nach Zoomstufe angegeben werden kann. Dabei besitzt das `zoomstep`-Tag ein Attribut `action` in dem angegeben wird, ob die Daten bei der erreichten Zoomstufe hinzugefügt (`add`) oder ersetzt (`replace`) werden sollen. Um die Datenquelle für den Layer anzugeben wird das `selection`-Tag benötigt. In ihm wird der Pfad zur Datendatei angegeben und es wird der Typ der Datei (`grib` oder `shp`) im Attribut `fileformat` angegeben.

Im `layout`-Tag wird dann das eigentliche Erscheinungsbild des Layers definiert. In der Schema Datei sind dazu sechs verschiedene Untertypen des `layout`-Tags definiert:

Text: Einfache textuelle Beschreibungen, beispielsweise für einen *Point of interest* oder dem Namen eines Staates.

TextPath: Text, der entlang einer Polylinie angegeben wird, um z.B. Fluss- oder Straßennamen wiederholt in die Karte einzuzeichnen.

Point: Definition eines Point of Interest (POI), gegebenenfalls mit einem Symbol zur Darstellung.

Polyline: Angabe von Linien, um z.B. Flüsse oder Straßen zu zeichnen.

Polygon: Geschlossene Flächen, um beispielsweise Länder, Seen, Industriegebiete oder Grünflächen darzustellen.

Grib: Visualisierung unterschiedlicher Wetterausprägungen, wie Wind, Regen, Temperatur, Luftdruck, usw.

Alle Layout-Typen besitzen Attribute für die Farbwerte, Liniendicken, Radien, usw. jeweils speziell für den Layout-Typ zugeschnitten. Bei GRIB Files können zusätzlich Attribute für die Generalisierung angegeben werden. Mit dem Attribut `generalizeX` wird definiert, wieviele Gitterpunkte in X-Richtung zusammengefasst und gemittelt werden sollen, `generalizeY` gibt dies für die Y-Richtung an.

Bei der Visualisierung von Shapefiles kann neben dem `layout`-Tag noch ein `param`-Tag angegeben werden, mit dem die gewünschten Daten besser spezifiziert werden können. So ist es möglich, nur Datensätze zu übernehmen, die eine bestimmte Eigenschaft besitzen. Die Eigenschaften von geometrischen Objekten sind in der `.dbf` Datei (siehe Kapitel 5.1) spaltenweise angegeben. Im `param`-Tag kann mit dem Attribut `index` die Spalte angegeben werden, mit `include` wird angegeben, ob die Daten hinzugefügt oder aber weggelassen werden sollen und im Attribut `valuelist` werden durch Semikoli voneinander getrennt die gewünschten Werte angegeben. Beispielsweise können alle Städte einer bestimmten Größe ausgewählt werden (Quellcode 13.3).

```
<selection sourcefile="input/deu/hires/city">
  <param index="2" include="include" valuelist="1.0;2.0"/>
  <layout xsi:type="Point"
          stroke="none"
          stroke-width="10"
          fill="red"
          radius="18"
          scale="100"/>
</selection>
```

Quellcode 13.3: Auswahl von Städten der Größe `1` und `2`

Am Ende der Konfigurationsdatei wird die Anzahl der unterschiedlichen Zoomstufen angegeben und um welchen Wert die Karte bei einem Zoomschritt vergrößert oder verkleinert wird (Quellcode 13.4). Dabei gibt das Attribut `stepfactor` die Schrittweite bei einer Bewegung in einer der Himmelsrichtungen in Relation zur Viewboxbreite und -höhe an. Das Attribut `steps` definiert die Anzahl der Zoomschritte, die in dieser Zoomstufe gemacht werden können, bevor man in die nächste wechselt, und `zoomfactor` gibt an mit welchem Wert die Viewboxdimensionen je Zoomschritt multipliziert werden.

```
<zoom>
  <zoomstep stepfactor="0.1" steps="3" zoomfactor="0.5"/>
  <zoomstep stepfactor="0.1" steps="3" zoomfactor="0.5"/>
</zoom>
```

Quellcode 13.4: Definition der Zoomstufen

Bei der Angabe des Layouts für GRIB Dateien sind spezielle Parameter notwendig, je nachdem, welche Informationen dargestellt werden sollen. Dazu wird der Layout-Typ `Grib` verwendet in dem unter anderem die Recordnummer (Kapitel 5.2.1) angegeben werden muss, in der der gewünschte Datensatz vorhanden ist. Da in der Arbeit in erster Linie Wetterausprägungen visualisiert werden sollen, soll auf die möglichen Layoutangaben für die GRIB Daten im Folgenden näher eingegangen werden.

13.1.1 Layoutdefinitionen für Isolinien und Isoflächen

Für die Darstellung von Isobaren muss das Attribut `isotype` auf den Wert `line` gesetzt werden. Danach folgen die Isowerte für die einzelnen Isobaren. Des Weiteren kann die Liniendicke und -farbe angegeben werden. Die Isobaren können beschriftet werden. Ist dies erwünscht, wird das Attribut `label` auf den Wert `true` gesetzt. Zusätzlich kann eine Fontgröße ausgewählt werden. Da die Daten in einem GRIB oftmals skaliert sind oder die Einheit der Daten geändert werden soll, kann über das Attribut `unitconverterclass` eine Javaklasse angegeben werden, die die Werte entsprechend umrechnet. Ein Beispiel ist im Quellcode 13.5 angegeben.

```
<layout xsi:type="Grib" recordnumber="1"
     isovaluelist="960;965;970;975;980;985;990;
                   995;1000;1005;1010;1015;1020;
                   1025;1030;1035;1040;"
     stroke="red"
     stroke-width="10"
     labeled="true"
     font-size="350"
     unitconverterclass="util.PRMSLConverter"
     isotype="line"/>
```

Quellcode 13.5: Layoutdefinition für Isobaren

Sollen Isoflächen dargestellt werden, beispielsweise für die Temperatur, muss im Attribut `isotype` der Wert `area` angegeben werden. Außerdem müssen Farbwerte angegeben werden, mit denen die einzelnen Isoflächen eingefärbt werden sollen. Die Farbwerte werden im Attribut `isocolorlist` angegeben und können wie in Quellcode 13.6 über die RGB-Werte oder mit Hexadezimalwerten oder Farbnamen, wie `grey`, `blue`, usw. definiert werden.

```
<layout xsi:type="Grib" recordnumber="34"
        isovaluelist="-25;-20;-15;-10;-5;0;5;10;15;20;25;30"
        isocolorlist="rgb(36,0,255);rgb(0,56,255);
                rgb(0,148,255);rgb(0,240,255);rgb(0,255,178);
                rgb(189,255,0);rgb(255,229,0);rgb(255,138,0)"
        stroke="green"
        stroke-width="3"
        abeled="true"
        font-size="350"
        isotype="area"
        unitconverterclass="util.KelvinCelsiusConverter"/>
```

Quellcode 13.6: Layoutdefinition für Isoflächen

Neben der Angabe von Farbwerten können auch Patterns angegeben werden (Kapitel 4.3). Dabei handelt es sich um vordefinierte Muster, die für die Füllung einer Fläche verwendet werden. Niederschlag kann besonders gut mit Patterns visualisiert werden.

Dazu werden unterschiedlich dicke diagonal verlaufende Linien als Pattern verwendet. Die Namen der Pattern werden dann anstelle der Füllfarbe angegeben (Quellcode 13.7).

```
<layout xsi:type="Grib" recordnumber="45"
            isovaluelist="3;6;9;12;15;18;21;24;27"
            isocolorlist="url(#muster0);url(#muster1);
                          url(#muster2);url(#muster3);
                          url(#muster4);url(#muster5)"
            stroke="none"
            stroke-width="10"
            labeled="true"
            font-size="350"
            unitconverterclass="util.RAINConverter"
            isotype="area"/>
```

Quellcode 13.7: Verwendung von Patterns zur Füllung der Isoflächen

13.1.2 Layoutdefinition für Winddaten

Die Visualisierung von Winddaten soll über Pfeile, an denen die Stärke des Windes eingezeichnet wird, realisiert werden (Kapitel 7.4). Der Benutzer hat hier nur wenig Einflussmöglichkeiten. Dementsprechend ist die Layoutdefinition sehr einfach (Quellcode 13.8). Als `isotype` muss der Wert `wind` angegeben werden. Zusätzlich kann das Grid wie bereits beschrieben generalisiert werden. Damit die Windpfeile in der Größe variiert werden können, kann über das Attribut `scale` ein Skalierungswert angegeben werden, mit dem die Ausmaße des Windsymbols multipliziert werden.

```
<layout xsi:type="Grib"
        recordnumber="12"
        generalizeX="2"
        generalizeY="2"
        scale="20"
        unitconverterclass="util.WINDConverter"
        isotype="wind"/>
```

Quellcode 13.8: Layoutdefinition für Windpfeile

13.2 Die Klasse `Creator`

Die `Creator`-Klasse stellt das Bindeglied zwischen den implementierten Modulen dar. In dieser Klasse werden nach den Angaben im Konfigurationsfile die Daten eingelesen, projiziert und vektorisiert. Die JavaScript und PHP Dateien für die Web Mapping Applikation werden generiert und die SVG Dateien mit der Visualisierung der Daten erzeugt.

Der Programmablauf in der `main`-Methode sieht vereinfacht wie folgt aus:

1. Das Konfigurationsfile wird mit Hilfe der Klasse `ConfigDOM` eingelesen. Dabei wird das Konfigurationsfile geprüft, ob alle Layer Angaben korrekt sind. Die Klasse `ConfigDOM` besitzt Methoden um auf Elemente der Konfigurationsdatei gezielt zugreifen zu können.
2. Die Angaben im Konfigurationsfile werden eingelesen und die Werte werden in der Klasse `Creator` in Feldern gespeichert.
3. Die Daten für die Karte werden eingelesen
4. Die geographischen Koordinaten werden gegebenenfalls projiziert
5. Die Daten werden gegebenenfalls generalisiert
6. Rasterdaten werden vektorisiert
7. Die benötigten Skripte werden erzeugt
8. Es werden Kacheln generiert
9. Die SVG Dateien werden geschrieben

Um alle einzelnen Layer zu erzeugen, werden in einer Schleife innerhalb des Programmes die `layer`-Elemente der Reihe nach abgearbeitet. Je nach `layer`-Typ werden unterschiedliche Module zur Weiterverarbeitung der Daten verwendet.

Ein Überblick über die Methoden der Klasse `Creator` ist im Anhang C.8 dargestellt.

13.3 Einlesen von Shapefiles

Für das Einlesen von Shapefiles werden Java-Klassen des OpenMap Projektes verwendet (Kapitel 5.1.2). Dazu wurden einige Klassen überarbeitet, um die ausgelesenen Daten besser in der eigenen Anwendung verwenden zu können. In erster Linie werden die Klassen `DbfInputStream`, `ShpInputStream` und `ShxInputStream` benötigt. Wie in Quellcode 13.9 zu sehen, wird für die `.shp`, die `.shx` und die `.dbf` Datei ein `URL`-Objekt erzeugt. Mit diesem werden dann die Streams initialisiert. Zunächst wird die `.shx` Datei eingelesen und von der Methode `getIndex` wird ein zweidimensionales `int`-Array zurückgeliefert (`indexData`), mit dem die Beziehung zwischen `.shp` und `.dbf` Datei hergestellt werden kann. Der `ShpInputStream` wird mit einem `DbfInputStream`-Objekt und dem `int`-Array `indexData` initialisiert. Mit der Methode `getGeometry` wird dann eine Liste mit den geometrischen Objekten geliefert.

```
GraphicList list = null;
ShxInputStream xis;
int[][] indexData = null;

try {
  URL shp = new File("source.shp").toURL();
  URL shx = new File("source.shx").toURL();
  URL dbf = new File("source.dbf").toURL();
} catch (MalformedURLException ex) {
  System.err.println("Fehler in URL!");
}
try {
  InputStream is = shx.openStream();
  xis = new ShxInputStream(is);
  indexData = xis.getIndex();
  is.close();
} catch (Exception e) {
  System.err.println("Unable to stream SHX file\n");
}
try {
  InputStream is = shp.openStream();
  ShpInputStream pis = new ShpInputStream(is,
                   new DbfInputStream(dbf.openStream()));
  list = pis.getGeometry(indexData);
  is.close();
} catch (Exception e) {
  System.err.println("Unable to stream SHP file\n");
}
```

Quellcode 13.9: Einlesen eines Shapefiles mit OpenMap

13.4 Einlesen von GRIB Daten

In Kapitel 5.2 wurde das GRIB Format bereits näher erläutert. Es setzt sich aus einzelnen Records zusammen, die jeweils die Daten eines Parameters (Temperatur, Niederschlag, etc.) beinhalten. Jeder Record besteht wiederum aus sechs einzelnen Sektionen, wobei die letzte nicht von Interesse ist, da sie nur die Endmarkierung enthält. Um die Daten einzulesen, wird eine Java Klassenstruktur entworfen, die das GRIB Format widerspiegelt (Abbildung 13.1). Dabei wurde für die fünf wichtigen Sektionen jeweils eine eigene Klasse implementiert.

Jede GRIB Section ist in einer eigenen Javaklasse gekapselt. Alle Sektionen eines Records zusammen werden in einem Objekt der Klasse `GribRecord` verwaltet und die Records werden im Objekt der Klasse `GribFile` in einer Liste abgespeichert. So entsteht eine übersichtliche Struktur eines GRIB Files. Dieses API erlaubt den gezielten Zugriff auf einzelne Records des GRIB Files und auf die einzelnen Sektionen eines Records.

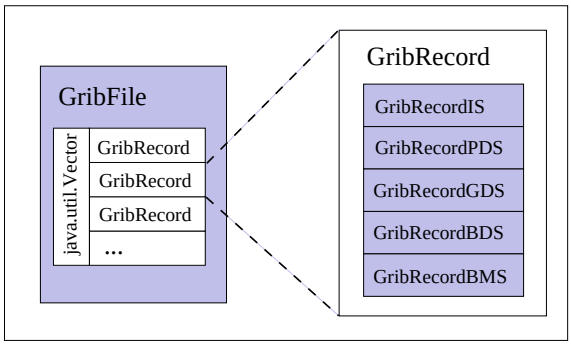


Abbildung 13.1: GRIB Datei in Java

13.4.1 Verarbeitung eines GRIB Files

Die Daten in einem GRIB File sind bitweise abgelegt. Um diese Daten einfach einlesen zu können, wurde eine Klasse `BitInputStream` implementiert, die von der Klasse `java.io.FilterInputStream` erbt. Die Klasse besitzt Methoden, um einzelne Bytes (signed und unsigned), einzelne Bits und Words (zwei Byte) einzulesen. Die eingelesenen Daten können dann mit der Klasse `Bytes2Number` zu Zahlwerten umgewandelt werden. Die Klasse ist notwendig, da beispielsweise in einem GRIB File Gleitkommazahlen sehr kryptisch kodiert sind und diese erst passend umgerechnet werden müssen. Ebenso verhält es sich bei anderen Zahlwerten. Daher wurden die einzelnen Kodierungen in Methoden gefasst, um komfortabler darauf zugreifen zu können.

Das Dekodieren eines Records übernehmen die Klassen, die die einzelnen Sektionen repräsentieren. Jede Klasse erhält das `BitInputStream`-Objekt und liest die notwendigen Daten ein. Jede Klasse kennt den bitweisen Aufbau seiner Sektion und kann so die einzelnen Bits den entsprechenden Werten zuordnen. Die Werte werden dann im entsprechenden Objekt in Instanzvariablen abgespeichert. Dabei handelt es sich um Metadaten, die die Daten des GRIB Files näher spezifizieren, wie die Anzahl der Gridpunkte in X- und Y-Richtung, die Projektion der Daten, Zeitstempel und andere Informationen. Nur die Klasse `GribRecordBDS` enthält die eigentlichen Werte des GRIBs. Da wie bereits erwähnt unterschiedliche Dialekte des GRIB Formates existieren, kann nicht garantiert werden, dass alle Metadaten aus GRIB Files der Version 1 korrekt eingelesen werden können. Allerdings sollten die Gitterwerte in jedem Fall korrekt ausgelesen werden.

13.4.2 Verwendung des GRIB API

Will der Benutzer ein GRIB einlesen, muss er zunächst ein `GribFile`-Objekt mit der Angabe des einzulesenden Dateinamens erzeugen. Das GRIB File wird dann automatisch eingelesen und es kann mit der Methode `getRecord(int i)` ein Record ausgewählt werden. Das Recordobjekt kann dann nach den einzelnen Sektionen befragt werden und auf die zugehörigen Informationen kann über verschiedene Methoden zugegriffen werden. Um die im GRIB enthaltenen Daten als `float`-Array zu erhalten, muss die Methode `getValues` am `GribRecordBDS`-Objekt, welches die Binary Data Section re-

präsentiert, aufgerufen werden. Einen Überblick über die Klassen und Methoden ist im Anhang C.1 zu finden.

Um einen Überblick über den Inhalt eines GRIB Files zu erhalten, kann die Klasse `GribInfo` verwendet werden (Quellcode 13.10).

```
import grib.GribFile;
import grib.GribRecord;
import grib.GribRecordBDS;
import grib.GribRecordGDS;
import grib.GribRecordIS;
import grib.GribRecordPDS;

public class GribInfo {
   public static void main(String[] args) {
      GribFile gf = null;

      try {
         gf = new GribFile(args[0]);
         System.out.println("Info fuer "+args[0]);

         for(int i=1; i<=gf.getRecordCount();i++) {
            GribRecord r = gf.getRecord(i);

            System.out.println("Recordnummer " + i);
            System.out.println("Type: "+r.getType());
            System.out.println("Record: "+r);
            System.out.println("\n********************\n");
         }
      } catch (FileNotFoundException e1) {
       System.err.println("Datei nicht gefunden");
      } catch (NoValidGribException e2) {
       System.err.println("Kein korrektes GRIB File");
      } catch (IOException e3) {
        System.out.println("Lesefehler");
      }
   }
}
```

Quellcode 13.10: Klasse zum Anzeigen des GRIB Inhaltes

13.4.3 Repräsentation der Daten

Die ausgelesenen Daten müssen geeignet im Speicher vorgehalten werden. Dazu werden die Werte des Rasters je Gitterpunkt in einem `GridPoint` Objekt mit den echten geographischen Koordinaten und dem zugehörige Parameterwert, wie beispielsweise die Temperatur oder der Luftdruck, gespeichert. Die `GridPoint`-Objekte werden wiederum in einem Objekt der Klasse `GridShape` verwaltet (Abbildung 13.2).

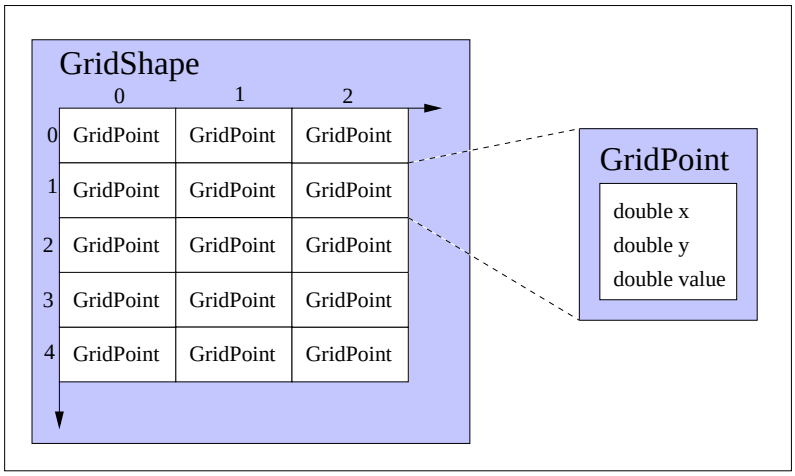


Abbildung 13.2: Verwaltung der Rasterpunkte

Dazu wird in dem `GridShape`-Objekt ein zweidimensionales Array von `GridPoints` angelegt. Die Dimensionen des Arrays entsprechen dabei der Rastergröße. Beide Klassen bieten diverse Methoden für den einfachen Zugriff auf einzelne Komponenten.

Um bei verschiedenen Zoomstufen unterschiedliche Auflösungen der GRIB Daten zu erhalten, kann das `GridShape`-Objekt generalisiert werden. Dazu werden die im Konfigurationsfile angegeben Werte für die Generalisierung ausgelesen. Die Werte sind im `layout`-Element in den Attributen `generalizeX` und `generalizeY` angegeben. Diese Werte werden der Methode `getGeneralized` übergeben. Besteht das Grid des `GridShape`-Objektes aus 100 mal 100 Punkten und das Attribut `generalizeX` hat den Wert zwei und `generalizeY` den Wert zehn, erhält man ein `GridShape`-Objekt der Breite 50 und der Höhe zehn. Dabei werden alle Punkte des zwei mal zehn großen Rechteckes gemittelt und zu einem Wert zusammengefasst. Dabei reduziert sich entsprechend die Dimension (Abbildung 13.3).

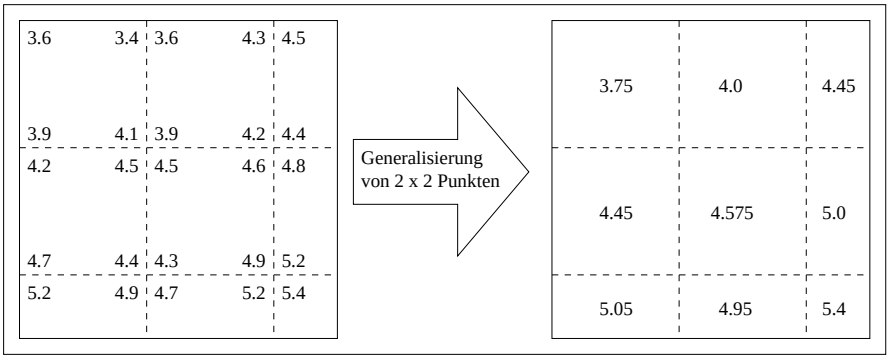


Abbildung 13.3: Generalisierung von Rasterwerten

In der Klasse `VectorizerImpl` wird der modifizierte Algorithmus von Snyder (Kapitel 10.3) zum Finden der Isolinien implementiert. Dazu wird das `GridShape`-Objekt, die Konturwerte und die zugehörigen Farben für die einzelnen Isoflächen übergeben.

Dann werden die Isolinien ermittelt und jeweils in einem `Polygon`-Objekt abgespeichert. Für jeden Isowert existiert eine typsichere `ArrayList` und die `Polygon`-Objekte werden je nach Isowert in einer der Listen gespeichert. Die so entstandenen Listen werden wiederum in einer typsicheren `ArrayList` verwaltet (Abbildung 13.4).

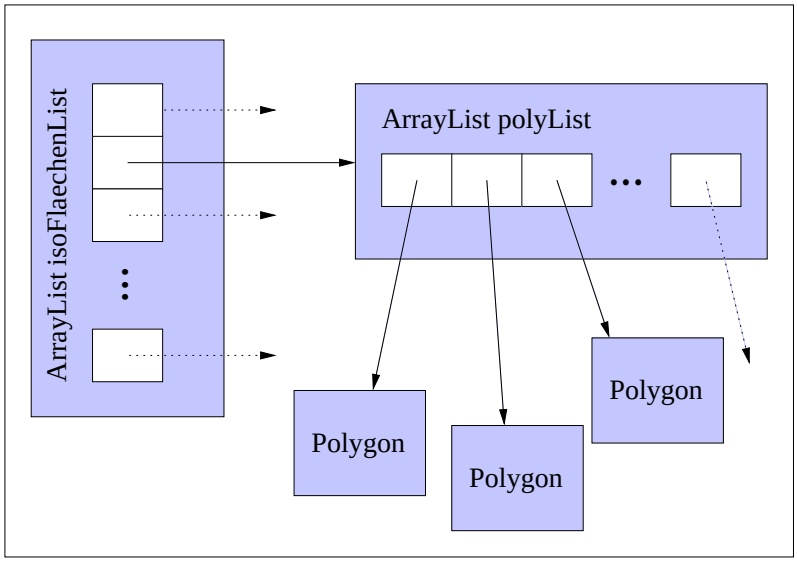


Abbildung 13.4: Verwaltung von Isolinien

Sind alle Isolinien berechnet und in die passenden Listen eingehängt, werden die Nachbarschaften, wie in Abschnitt 13.7 erläutert, ermittelt. Dabei werden jedem `Polygon`-Objekt die direkt darin enthaltenen Polygone zugeordnet und in einer Liste referenziert, damit die innen Liegenden Polygone später ausgeschnitten werden können. So enthält jetzt jedes `Polygon`-Objekt alle benötigten Informationen um die Isofläche zu zeichnen. Das `Polygon`-Objekt besitzt die Fähigkeit, sich selbst als SVG-Element auszugeben, was in Kapitel 13.6 näher beschrieben wird.

13.5 Projektion und Konvertierung der Daten

Sowohl die geographischen Koordinaten der Shapefiles und der GRIB Files müssen gegebenenfalls projiziert werden (Kapitel 6), da je nach Region unterschiedliche Kartenprojektionen verwendet werden sollten [SnyderP]. Dazu kann im Konfigurationsfile im Attribut `converterclass` des Elementes `configuration` eine Java-Klasse angegeben werden, die standardmäßig verwendet wird, um die Koordinaten wie gewünscht von einem Koordinatensystem in ein anderes zu projizieren. Diese Klasse muss das Interface `Converter` implementieren, in dem lediglich eine Signatur angegeben ist:

```
public interface Converter {
  public void convert(MyPoint p);
}
```

Da einige Projektionen zusätzliche Parameter benötigen, können diese in einem Attribut `converterparams` angegeben werden. Dabei dürfen nur Gleitkommazahlen als Parameterwerte verwendet werden.

Eine globale Angabe einer Projektion reicht aber nicht aus, da die Daten der einzelnen Layer in verschiedenen Koordinatensystemen vorliegen können. Daher können für jeden Layer im `selection`-Tag bei Bedarf die Java-Klasse und eventuell benötigte Parameter separat angegeben werden.

Auch die Werte des Rasters selbst müssen gegebenenfalls konvertiert werden, da sie in einer anderen Einheit vorliegen als gewünscht. Im Attribut `unitconverterclass` wird dazu eine Klasse für die Umrechnung der Rasterwerte angegeben, die das Interface `UnitConverter` implementieren muss:

```
public interface UnitConverter {
  public double convert(double value);
}
```

Eine im Konfigurationsfile angegebene Java-Klasse wird über das *Java Reflection* Konzept [Flanagan] geladen und instantiiert (Quellcode 13.11). Dazu werden zunächst die Attribute `converterparams` und `converterclass` aus dem DOM des Konfigurationsfiles gelesen. Wurden keine Parameter angegeben, kann ein Objekt der Klasse direkt instantiiert werden. Dazu wird mit der Methode `forName` die Klasse geladen, der Konstruktor der Klasse wird mit `getConstructor` ermittelt und mit `newInstance` wird ein Objekt der Klasse erzeugt. Wurden zusätzliche Parameter angegeben, müssen diese zunächst eingelesen werden und ein Array mit den Klassen der Datentypen (in diesem Fall nur `Double`) muss angelegt werden. Dann kann mit `getConstructor` der passende Konstruktor gewählt werden und das Objekt wird instanziiert. Da in Quellcode 13.11 viele verschiedene Exceptions auftreten können, aber die Übersichtlichkeit gewahrt bleiben sollte, werden im `catch`-Block alle Exceptions über die Oberklasse `Exception` abgefangen.

Einen Überblick über einige Klassen zur Projektion geographischer Koordinaten und zur Konvertierung von Maßeinheiten ist im Anhang C.3 zu finden.

13.6 Verwaltung der Shapes

Die benötigten Grundprimitive werden in folgenden Klassen gespeichert:

- `Point`
- `Polyline`
- `Polygon`

Die Klassen stammen von der abstrakten Klasse `Graphic` ab, in der verschiedene Felder vorhanden sind, um das Erscheinungsbild der Objekte zu definieren und es sind Methoden implementiert, um diese Werte abzufragen und zu verändern. Jede der drei Klassen

```
try {

  String convParamString = e.getAttribute("converterparams");

  if (convParamString.compareTo("") == 0) {
    converter = (Converter) Class.forName(
            e.getAttribute("converterclass")).getConstructor()
                                             .newInstance();
  }
  else {
    String[] paramStrings = convParamString.split(
             e.getAttribute("delimiter"));

    Double[] convParams = new Double[paramStrings.length];
    Class[] convParamTypes = new Class[paramStrings.length];

    for (int i = 0; i < convParams.length; i++) {
      convParams[i] = Double.parseDouble(paramStrings[i]);
      convParamTypes[i] = Double.TYPE;
    }

    converter = (Converter) Class.forName(converterClassString)
                .getConstructor(convParamTypes)
                .newInstance(convParams);
  }
} catch (Exception e) {
    e1.printStackTrace();
}
```

Quellcode 13.11: Dynamisches Laden und instantiieren einer `Converter`-Klasse

stellt Methoden zur Verfügung, um mit den abgespeicherten Werten ein entsprechendes SVG Element zu erzeugen. Dazu wird ein DOM API verwendet, mit dem es möglich ist einen DOM-Tree zu erzeugen.

Für ein `Point`-Objekt wird, wie in Quellcode 13.12 angegeben, ein Element erzeugt, welches dann an die passende Stelle eingehängt wird. Dazu wird unterschieden, ob für den Punkt ein Symbol oder ein Kreis eingefügt werden soll. Im ersten Fall wird das SVG Element `use` mit passendem Namespace und in dem entsprechenden DOM-Tree erzeugt. Im Folgenden werden die Attribute des Elementes auf die im `Point`-Objekt angegebenen Werte gesetzt. Danach wird das Element an das Element `d` angehängt, welches der Methode als Parameter übergeben wurde. Soll ein Kreis erzeugt werden, wird das SVG Element `circle` passend angelegt und dann mit den entsprechenden Attributwerten versehen. Auch hier wird als letztes das neue Element in den DOM-Tree eingehängt.

Bei dem `Polyline`- und dem `Polygon`-Objekt verläuft das Erzeugen des zugehörigen SVG Elementes analog, nur dass hier Punktlisten verarbeitet und teilweise andere Parameter gesetzt werden müssen.

```
public void append_to(Element d) {
  if(symbolid.compareTo("")!=0){
    Element c = d.getOwnerDocument().
          createElementNS("http://www.w3.org/2000/svg","use");

    c.setAttribute("xlink:href", "#"+symbolid);
    if (rotate.compareTo("0")==0) {
      c.setAttribute("transform", "translate("+getLon()+","+
                     getLat()+ ") scale("+scale+")");
    }
    else {
      c.setAttribute("transform", "translate("+getLon()+","+
           getLat()+ ") scale("+scale+") rotate("+rotate+")");
    }
    d.appendChild(c);
  }
  else{
    Element c = d.getOwnerDocument().
       createElementNS("http://www.w3.org/2000/svg","circle");
    c.setAttribute("cx", getLon() + "");
    c.setAttribute("cy", getLat() + "");
    c.setAttribute("stroke", stroke);
    c.setAttribute("stroke-width", strokewidth);
    c.setAttribute("r", radius);
    c.setAttribute("fill", fill);
    d.appendChild(c);
  }
}
```

Quellcode 13.12: Erzeugen und Einhängen der SVG Darstellung eines `Point`-Objektes

Jedes Shapeobjekt kann an einem Rechteck geclippt werden. Wie in Abschnitt 12.2 beschrieben, ist dies notwendig, damit die Shapes in Kacheln eingeteilt werden können, um das Nachladen von Daten zu optimieren.

Alle drei Shapetypen müssen im Programm verwaltet werden. Dazu existieren für jede Klasse spezielle Listen:

- `PointList`
- `PolyList`
- `PolygonList`
- `IsoareaList`

Alle Listen stammen von der abstrakten Klasse `GraphicList` ab, die wiederum von der Klasse `Graphic` abstammt. Die Listen können Objekte der von `Graphic` abgeleiteten Klassen verwalten. Also können in einer Liste auch wiederum Listen enthalten sein.

Alle Listen bieten Methoden zur Veränderung der enthaltenen Objekte. Dabei werden die Methodenaufrufe an alle enthaltenen Objekte weitergeleitet. Dieses Vorgehen wird als *Composite*-Pattern bezeichnet [Gamma]. Bei diesem Pattern wird eine `Composite`-Klasse definiert. Ein `Composite`-Objekt kann wiederum `Composite`-Objekte enthalten, diese wiederum andere und so weiter. `Leafs` sind ebenfalls `Composite`-Objekte, enthalten aber keine weiteren mehr. Wird nun eine Methode an ein `Composite`-Objekt geschickt, wird sie solange weitergeleitet, bis ein `Leaf` erreicht wurde. Das Konzept bietet den Vorteil, dass beliebig oft verschachtelte Listen vorhanden sein können und das auch unterschiedliche Shapes gemeinsam verwaltet werden können.

Werden in einem `PolygonList`-Objekt mehrere Polygone abgespeichert, können diese auf zwei unterschiedliche Arten interpretiert werden. Grundsätzlich werden die Polygone in einer `PolygonList` als eigenständige Polygone mit identischen Attributen aufgefasst. Wird bei einem `PolygonList`-Objekt jedoch die Methode `treatasone()` aufgerufen, verhält sich die Liste wie ein einziges Polygon. Bei der SVG Erzeugung wird dann für die gesamte Liste ein `path`-Element erzeugt und die Polygone werden in einem Pfad, jeweils durch ein `M` getrennt, aneinandergereiht. Dadurch wird, wie in Abschnitt 4.1 erläutert, die Zeichnung für jedes Polygon erneut begonnen und aus dem äußeren Polygon werden die inneren ausgestanzt. So können Flächen mit Löchern versehen werden.

Die `IsoareaList` verwaltet wie die `PolygonList` `Polygon`-Objekte. Eine separate Klasse wurde implementiert, da an die `IsoareaList` spezielle Anforderungen an die Füllfarben gestellt werden. In den `PolygonList`-Objekten werden nur Polygone mit der gleichen Füll- und Linienfarbe verwaltet. Objekte in einer `IsoareaList` hingegen können mit separaten Füllfarben versehen werden, die dann nicht mehr überschrieben werden. Bei dieser Klasse wird also das Konzept des Composite Pattern nur teilweise umgesetzt.

Die gesamte Klassenstruktur ist in einem vereinfachten Klassendiagramm im Anhang C.4 skizziert.

13.7 Berechnung von Isolinien und Isoflächen

Wird in der Konfigurationsdatei angegeben, dass ein GRIB Record mit Isolinien dargestellt werden soll, müssen die Daten zunächst vektorisiert werden. Dazu wird einem Objekt der Klasse `VectorizerImpl` ein `GridShape` Objekt, die Isowerte und die Angabe, ob die Isolinie beschriftet sein soll oder nicht, übergeben. Das Raster wird mit dem modifizierten Linefollowing Algorithmus 10.3.1 vektorisiert und es wird ein `PolygonList`-Objekt zurückgeliefert, welches weiter verarbeitet werden kann.

Sollen keine Isolinien, sondern Isoflächen dargestellt werden, müssen die Flächen aus den berechneten Isolinien gebildet werden. Eine Isofläche repräsentiert im Gegensatz zu einer Isolinie ein Intervall von Messwerten. Dabei grenzt eine Isolinie zwei benachbarte Isoflächen voneinander ab. Liegt eine Isofläche komplett in einer anderen, so muss diese aus der äußeren ausgestanzt werden, so dass eine Art Doughnut gebildet wird (Abbildung 13.5). Daher muss ermittelt werden, welche Isofläche in welcher anderen enthalten

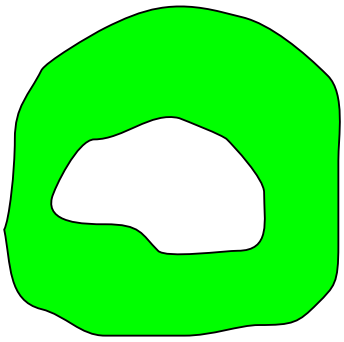


Abbildung 13.5: Eine ausgestanzte Isofläche

ist. Dabei ist von Vorteil, dass nur Isoflächen benachbarter Isowerte betrachtet werden müssen, da andere Isoflächen nicht aneinander grenzen können. Dies verringert die Anzahl der Vergleiche und vermindert so den Rechenaufwand.

Die Isolinien sind in separaten Listen je Isowert gespeichert und je Liste werden alle Isolinien durchlaufen. Für jede zu überprüfende Isofläche wird in der Liste mit dem höheren Isowert und in der Liste mit dem niedrigeren Isowert getestet, ob sie in der zu testenden enthalten ist. Dabei reicht es aus, einen Punkt einer möglichen inneren Isolinien zu betrachten und zu überprüfen, ob er in dem Polygon der zu testenden Isolinie enthalten ist. Der Test wird mit einem *Point in Polygon* Algorithmus (Kapitel 4.2) vorgenommen.

Die Überprüfung, welche Isolinie innerhalb einer anderen liegt, reicht aber nicht aus. Interpretiert man die Isowerte als Höheninformation, besitzen Isolinien eine Art Wellenstruktur. Abbildung 13.6 zeigt eine solche Struktur.

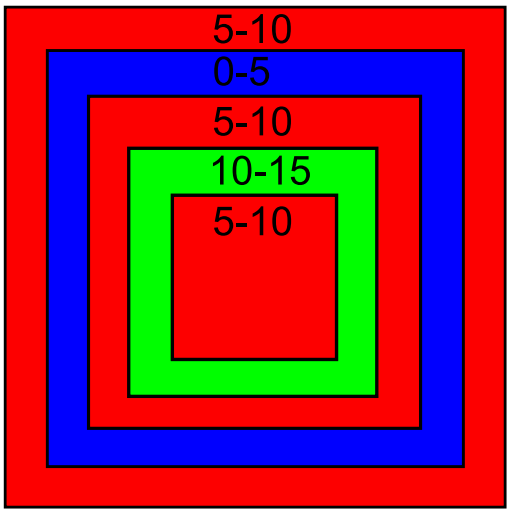


Abbildung 13.6: Ineinander geschachtelte Isoflächen

Von außen nach innen kommt zunächst ein Tal, danach bewegt man sich auf die ursprüngliche Höhe zurück und erreicht dann einen Berg, um in der Mitte der Grafik wieder auf normal Niveau zu gelangen. Relativ zum normalen Niveau stellen hier sowohl Berg als auch Tal eine benachbarte Linie dar und kommen somit beide für die Bildung einer

Isofläche in Frage. Daher muss eine weitere Überprüfung vorgenommen werden. Findet man für eine Isolinie mit dem Isowert t_i eine innen liegende Isolinie mit einem Isowert t_{i+1}, muss getestet werden, ob die innere in einer der anderen Isolinien mit dem Isowert t_{i+1} enthalten ist. Wenn dies nicht der Fall ist, oder aber alle Isolinien durchsucht wurden, bricht die Schleife ab. Der gleiche Vorgang muss auch mit den Isolinien mit Isowert t_{i-1} durchgeführt werden. Das Ergebnis der beiden Suchen kann in drei Fälle eingeteilt werden:

1. Es wurde keine innen liegende Isolinie gefunden. Daraus folgt, dass die überprüfte Isolinie keinen inneren echten Nachbarn besitzt.
2. Es wurde nur in einem der Schleifendurchgänge eine innere Isolinie gefunden. Daraus folgt, dass die überprüfte Isolinie und die gefundene Linie echt benachbart sind.
3. In beiden Schleifen wurde je ein echter Nachbar gefunden. In diesem Fall muss mit dem Point in Polygon Algorithmus getestet werden, welche Isolinie in welcher enthalten ist und den echten Nachbarn darstellt.

Der Algorithmus liefert zwei echt benachbarte Isolinien, die eine Isofläche einschließen.

Eine Übersicht der Klassen für die Vektorisierung ist im Anhang C.2 dargestellt.

13.8 Berechnung der Winddarstellung

Die Winddaten sind als vektorieller Wert abgespeichert. Sie besitzen eine `u`- und eine `v`-Komponente, mit der die Richtung und die Stärke des Windes angegeben wird (Abbildung 13.7). Die Richtung des Windes ist durch die Steigung $\frac{v}{u}$ gegeben. Da die Windsymbole um einen Winkel gedreht werden müssen, muss der Winkel bestimmt werden: $\alpha = \arctan \frac{v}{u}$. Die Intensität des Windes (`s`) ergibt sich aus der Formel $s = \sqrt{u^2 + v^2}$.

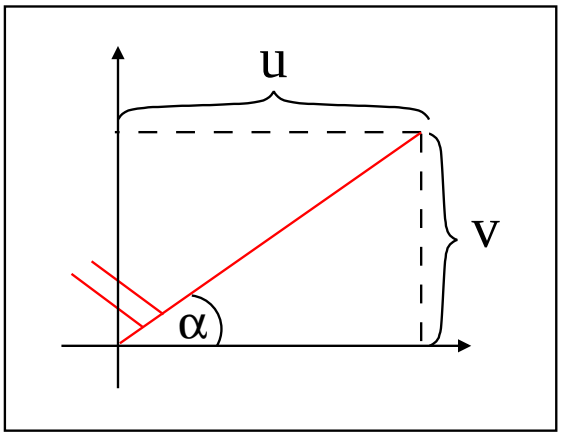


Abbildung 13.7: Schematische Darstellung der Windrichtung und -stärke

Da in einem GRIB-Record nur skalare Werte gespeichert werden können, müssen Winddaten in zwei Records abgespeichert werden. Sind die Werte beider Records geladen und gegebenenfalls projiziert und generalisiert, wird das Raster Punkt für Punkt durchlaufen. Die Windstärke wird an dem Punkt ermittelt und der Winkel berechnet. Mit diesen Daten kann ein Windsymbol (Abbildung 7.5) passend in die Karte eingefügt werden. Die Berechnung der Werte ist in Quellcode 13.13 zu sehen.

```
for(int row=0; row<gp1.length;row++) {
  for(int col=0;col<gp1[row].length;col++) {
    double u = gp1[row][col].getValue();
    double v = gp2[row][col].getValue();

    Point p = new Point(gp1[row][col].getX(),
                        gp1[row][col].getY());

    // Berechnung der Staerke und Umrechnung von m/s in Knoten
    double staerke = Math.sqrt(u*u+v*v)*1.944;

    p.setSymbolid(getSymbolID(staerke));

    double alpha = Math.toDegrees(Math.atan(v/u));

    if(u<0 && v>0) alpha = 180 + alpha;
    else if(u>0 && v<0) alpha = 360 + alpha;
    else if(u<0 && v<0) alpha = 180 + alpha;
    // Drehrichtung aendern
    alpha=-alpha;
    p.setRotate(alpha+"");
    g.add(p);
  }
}
```

Quellcode 13.13: Bestimmung der Windstärke und -richtung

13.9 Schreiben der SVG Dateien

Die einzelnen Listen, in denen die geometrischen Objekte verwaltet werden, werden durchlaufen und entsprechend der Vorgaben der Konfigurationsdatei zu einzelnen Gruppen zusammengefasst. An jedem geometrischem Objekt wird dann eine `append_to` Methode aufgerufen, um anhand seines aktuellen Zustandes ein SVG Element zu erzeugen. Dazu gibt es eine Methode ohne Clipping, die nur das Element, an dem das SVG-Element angehängt werden soll, übergeben bekommt und es gibt eine Methode, die zusätzlich die Clippinggrenzen übergeben bekommt, um die Kacheln erzeugen zu können. Beispielhaft wird die SVG Erzeugung für ein `Polygon`-Objekt gezeigt (Quellcode 13.14).

```
public void append_to(Element parent,MyPoint oben,MyPoint unten)
{
  String ns = "http://www.w3.org/2000/svg"; // Namespace
  String path = getRawPath(oben, unten);
  if (path.compareTo("") != 0) {
    Element poly=d.getOwnerDocument().
                createElementNS(ns,"path");
    poly.setAttribute("d", path);
    poly.setAttribute("stroke-width", "0");
    poly.setAttribute("stroke", "none");
    poly.setAttribute("fill", fill);
    poly.setAttribute("fill-rule", "evenodd");
    poly.setAttribute("stroke-linecap", linecap);
    poly.setAttribute("stroke-linejoin", linejoin);
    d.appendChild(poly);
    Element line=d.getOwnerDocument().
                createElementNS(ns,"path");
    line.setAttribute("d", getStrokePath(oben, unten));
    line.setAttribute("stroke-width", strokewidth);
    line.setAttribute("stroke", stroke);
    line.setAttribute("fill", "none");
    line.setAttribute("stroke-linecap", linecap);
    line.setAttribute("stroke-linejoin", linejoin);
    parent.appendChild(line);
  }
}
```

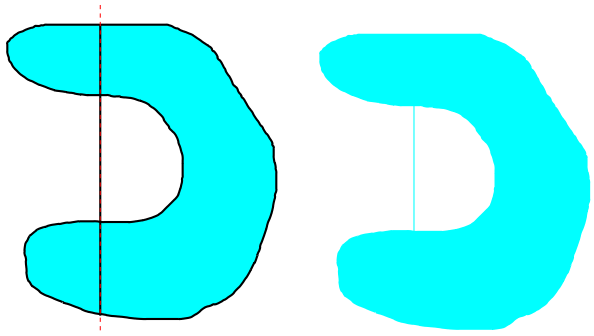
Quellcode 13.14: Erzeugung eines SVG Elementes aus einem `Polygon`-Objekt

Mit der Methode `getRawPath` wird zunächst der Pfad des Polygons bestimmt. Dazu wird das Polygon in der Methode geclippt (Kapitel 13.9.1)und gegebenenfalls geglättet (Kapitel 13.9.2). Wenn das Polygon außerhalb der Clippinggrenzen liegt, liefert die Methode `getRawPath` einen leeren String zurück, und die weitere Verarbeitung wird abgebrochen. Ansonsten wird das Ergebnis der Methode mit `setAttribute` als Wert des `d`-Attributes des `path`-Elementes gesetzt. Die anderen Attribute werden ebenfalls mit der Methode `setAttribute` eingefügt. Das fertige `path`-Element wird dann an das Element `parent` mit der Methode `appendChild` angehängt.

Falls ein Polygon durch das Clipping zerteilt wird, unterscheidet sich der Linienverlauf der äußeren Umrandung vom Linienverlauf der Füllung, da an der Clippinggrenze zwar die Füllgrenze definiert werden muss, aber keine Randlinie gezeichnet werden darf. Daher wird ein zweites `path`-Element mit den entsprechenden Attributen für die Randlinie erzeugt. Der Pfad wird mit der Methode `getStrokePath` erstellt. Auch dieses `path`-Element wird in den DOM eingehängt. Das DOM wird in der Klasse `MySVGDocument` erstellt und ist wie die Klassen zum Schreiben der SVG Dateien im Anhang C.7 dargestellt.

13.9.1 Clippen des Polygons

Prinzipiell lassen sich Polygone sehr leicht mit dem Reentrant Polygon Clipping Algorithmus von Sutherland und Hodgman [Sutherland] clippen. Allerdings führt der Clippingalgorithmus nicht immer zum gewünschten Erfolg. Liegt ein konkaves Polygon vor, kann es durch das Clipping in mehrere Teile zerfallen. Der Algorithmus verbindet die einzelnen Teile mit einer Linie (Abbildung 13.8(a)). Diese Linie darf nicht gezeichnet werden. Daher liegt es nahe, das Füllpolygon ohne Randlinie zu zeichnen, und die Linie getrennt zu behandeln. Allerdings können trotzdem Darstellungsfehler entstehen. So zeichnet der Adobe SVG Viewer eine dünne farbige Linie, die durch die Füllung des Polygons entsteht (Abbildung 13.8(b)).



(a) Mit Randlinie (b) Ohne Randlinie

Abbildung 13.8: Probleme beim Clippen mit Sutherland & Hodgman

Daher müssen zerfallende Polygone getrennt voneinander betrachtet werden. Dazu werden jeweils die inneren Teilstücke und die Randlinien benötigt. In der Abbildung 13.9 sind die entsprechenden Linien in Blau und in Grün markiert.

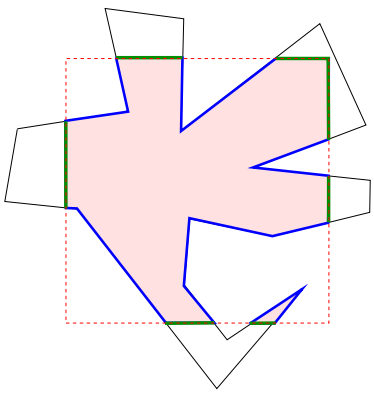


Abbildung 13.9: Durch Clipping in mehrere Teile zerfallendes Polygon

Um die Blauen Liniensegmente zu erhalten, wird das Verfahren von Greiner und Hormann [Greiner] verwendet. Die Punkte des Polygons werden der Reihe nach abgelaufen. Liegt ein Punkt innerhalb des Clippingrechteckes wird er zum Liniensegment hinzugefügt. Folgt auf einen innen liegenden ein Punkt außerhalb des Clippingrechteckes,

wird der Schnittpunkt mit der Clippingkante berechnet. Der interpolierte Punkt wird zum Liniensegment hinzugefügt, welches damit abgeschlossen ist. Gelangt man bei der Verarbeitung der einzelnen Punkte wieder innerhalb des Clippingrechteckes wird ein neues Liniensegment begonnen.

Um die grün markierten Linien zu erhalten, müssen zunächst alle gefundenen Randpunkte nach ihrer X- beziehungsweise Y-Koordinate sortiert werden. Danach werden die Punkte nach folgendem Konzept abgearbeitet:

1. Zwei aufeinanderfolgende Randpunkte werden betrachtet und der Mittelpunkt zwischen diesen berechnet.
2. Es wird überprüft, ob der Mittelpunkt innerhalb des zu clippenden Polygons liegt (Point in Polygon Algorithmus).
3. Liegt der Punkt innen, so gehört die ganze Linie zum geclippten Polygon. Ansonsten kann die Linie verworfen werden. Dabei müssen die Eckpunkte des Clippingrechteckes zu den Randpunkten hinzugefügt werden.
4. Sind alle innen liegenden Linien bekannt, werden anhand der Anfangs und Endpunkte zusammenhängende Linien gebildet, mit denen die Polygone erstellt werden können.

13.9.2 Glätten des Polygons

In der Methode `getRawPath` werden die Polygone gegebenenfalls geglättet. Dabei werden die Füllung und die Randlinie des Polygons unterschiedlich behandelt. Beim zu füllenden Polygon werden lediglich die innen liegenden Teilsegmente geglättet. Die Randlinien am Clippingrechteck bleiben weiterhin gerade Teilstücke. Hierbei stellt sich die Frage, wie ein stetiger Übergang zwischen von der einen geclippten Kachel zu der benachbarten erfolgen kann. Dies stellt aber kein großes Problem dar, da das geclippte Polygon für beide Kacheln denselben interpolierten Randpunkt besitzt. Dieser Punkt stellt jeweils das Ende eines gerundeten Teilsegmentes dar, in dem die Steigung gleich der ursprünglichen Polygonkante ist. In der Abbildung 13.10 ist das geglättete Polygon vor dem Clipping (schwarz) und nach dem Clipping (grün) zu sehen. Die blaue Kante stellt die Steigung im jeweiligen Endpunkt der Teilsegmente dar.

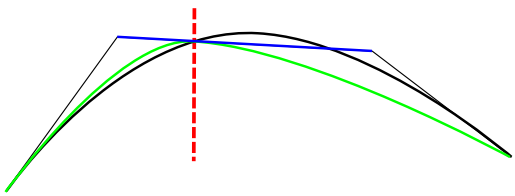


Abbildung 13.10: Kantenglättung eines Polygons vor und nach dem Clipping

Bei der Glättung der Randlinie des Polygons werden lediglich die innen liegenden Teilsegmente geglättet und die auf den Clippinggrenzen liegenden Linien werden nicht berücksichtigt.

IV Weitere Projekte

14 Visualisierung anderer georeferenzierter Daten

Klima- und Wetterdaten sind nur eine Form von Daten, die mit der erstellten Anwendung dargestellt werden können. Generell können alle georeferenzierten Daten visualisiert werden. Unter georeferenzierten Daten versteht man alle Datensätze, die eindeutig einem Längen- und Breitengrad zugeordnet werden können. Mögliche Einsatzgebiete sind die Daten des statistischen Bundesamtes. Diese Daten sind meist auf eine bestimmte Region oder einen Regierungsbezirk bezogen und können mit dem vorliegenden System einfach dargestellt werden. Es ist ebenso möglich, Straßenkarten, Wahlergebnisse, Meeresströmungen und vieles mehr darzustellen.

Die Verwendung der Applikation zur Darstellung von Karten und Daten des statistischen Bundesamtes soll im Folgenden kurz erläutert werden.

14.1 Darstellung von geographischen Karten

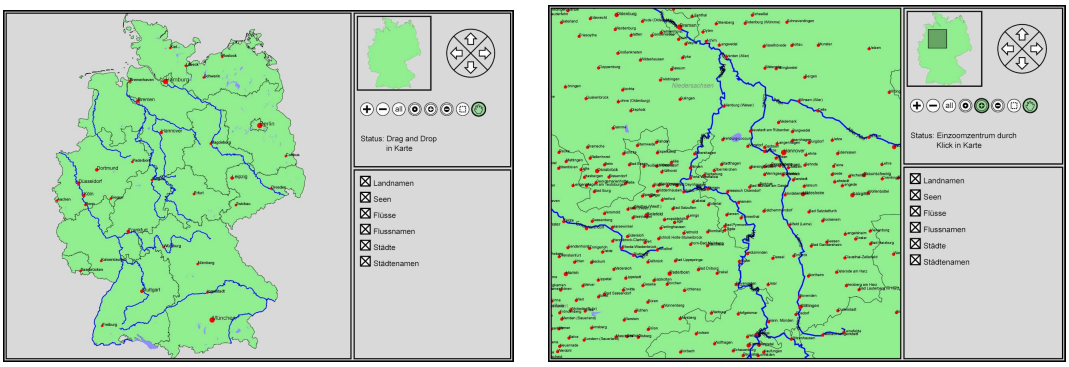
In Shapefiles ist die Geographie einer Region mit zusätzlichen Informationen gespeichert. Diese können ausgelesen (Kapitel 13.3) und visualisiert werden. Liegen die Bundesländer vor, können diese über wenige Angaben in der Konfigurationsdatei dargestellt werden (Quellcode 14.1). Zunächst wird im Element `layer` angegeben, ob die Daten

```
<layer xsi:type="single" changeable="false" id="country"
       visibility="visible" label="Bundesländer">
  <zoomstep action="add">
    <selection sourcefile="input/deu/hires/country">
      <layout xsi:type="Polygon" stroke="black"
              stroke-width="10" stroke-linecap="round"
              fill="lightgreen"/>
    </selection>
  </zoomstep>
</layer>
```

Quellcode 14.1: Angaben zur Visualisierung von Bundesländern

ein- und ausgeschaltet werden können, welche ID verwendet werden soll, ob die Daten initial sichtbar sind und welche Textinformation in der Steuerung erscheinen soll. Danach wird im `selection`-Element die Datenquelle angegeben. Im Element `layout` wird angegeben, wie die Daten dargestellt werden sollen. Werden neben den Bundesländern noch weitere Daten hinzugefügt, kann man eine einfache Deutschlandkarte, wie in Abbildung 14.1 zu sehen, erstellen.

Auch Stadtpläne können mit der Anwendung visualisiert werden, insofern die Daten vorhanden sind. Von der Stadt Osnabrück wurden Geodaten von der Intevation GmbH [Frida] erfasst und im Sinne von Open Source Software zur Verfügung gestellt. Mit diesen Daten kann ein Stadtplan von Osnabrück als SVG Web Mapping Applikation im Internet zur Verfügung gestellt werden (Abbildung 14.2).



(a) Komplette Übersicht (b) Vergrößerter Ausschnitt

Abbildung 14.1: Interaktive Deutschlandkarte

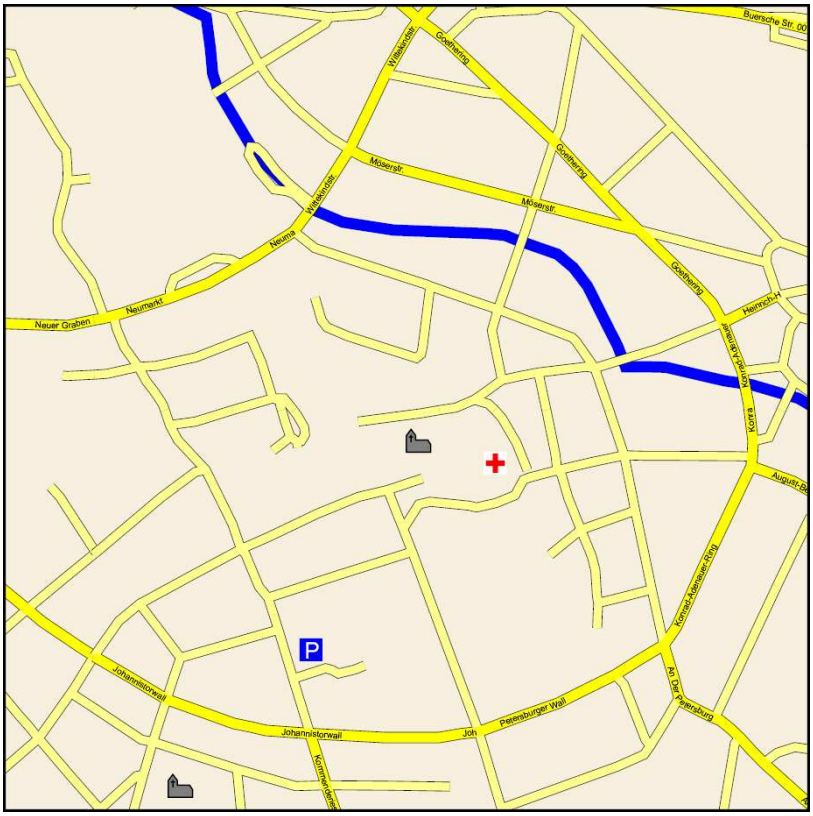


Abbildung 14.2: Ausschnitt aus dem Osnabrücker Stadtplan (Daten von [Frida])

Je nach Zoomstufe können unterschiedliche Informationen eingeblendet werden. So kann zunächst eine grobe Übersicht geladen werden und bei der Vergrößerung werden weitere Details nachgeladen. Da die Daten automatisiert erzeugt werden, bietet ein solcher Stadtplan zwar einen guten Überblick, aber insbesondere bei der Beschriftung von Straßen kann es zu Fehlern kommen.

14.2 Einbinden zusätzlicher Informationen

Die geometrischen Objekte in einer Karte können auch einzeln unterschiedlich eingefärbt werden. So ist es möglich die Bundesländer in einer Karte unterschiedlich farbig darzustellen. Dazu werden genauere Kenntnisse über den Inhalt der `.dbf`-Datei eines Shapefiles benötigt, damit bekannt ist, welche Werte in den enthaltenen Spalten vorhanden sind. Im vorliegenden Shapefile ist beispielsweise in der Spalte `2` der `.dbf`-Datei der Name eines Bundeslandes abgespeichert. Wie in Quellcode 14.2 zu sehen, kann über das `param`-Element mit dem Attribut `index` eine bestimmte Spalte des Shapefiles ausgewählt werden und im Attribut `valuelist` kann angegeben werden, welche Werte von Interesse sind. Mehrere Werte können mit einem frei wählbaren Zeichen voneinander getrennt werden. Mit diesen Angaben werden nur die geometrischen Objekte aus dem Shapefile ausgewählt, für die das angegebene Attribut gilt.

```
<selection sourcefile="input/deu/hires/country">
  <param index="2" include="include"
         valuelist="Berlin;Bremen"/>
  <layout xsi:type="Polygon" stroke="black" stroke-width="10"
          fill="red"/>
</selection>

<selection sourcefile="input/deu/hires/country">
  <param index="2" include="include"
         valuelist="Niedersachsen"/>
  <layout xsi:type="Polygon" stroke="black" stroke-width="10"
          fill="green"/>
</selection>

<selection sourcefile="input/deu/hires/country">
  <param index="2" include="include"
         valuelist="Bayern"/>
  <layout xsi:type="Polygon" stroke="black" stroke-width="10"
          fill="blue"/>
</selection>
```

Quellcode 14.2: Angaben zur Visualisierung von Bundesländern

Mit der unterschiedlichen Färbung einzelner Bundesländer können beispielsweise die Daten des Statistischen Bundesamtes [DESTATIS] visualisiert werden. In der Abbildung 14.3 wurden die Studentenzahlen prozentual zur Bevölkerung des jeweiligen Bundeslandes visualisiert. Mit steigendem Anteil verändert sich die Farbe von Blau über Grün, hin zu Rot. Die Daten stammen aus dem Jahr 2004.

Auf diese Art und Weise können beliebige georeferenzierte Daten in einer SVG Web Mapping Applikation dargestellt werden. Zusätzlich ist es möglich, auch eine zeitliche Änderung der Daten zu visualisieren.

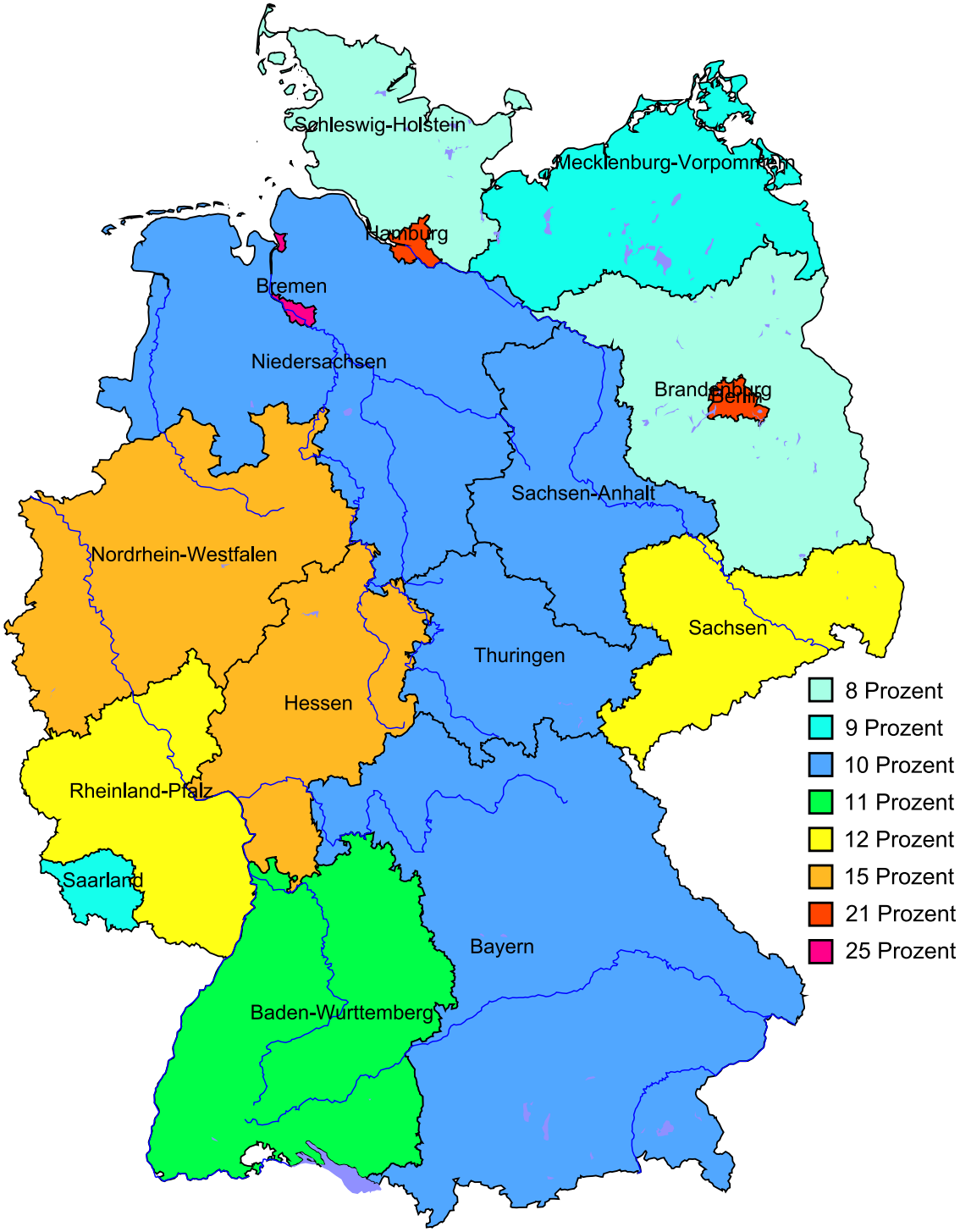


Abbildung 14.3: Prozentualer Anteil an Studenten an der Bevölkerung je Bundesland (Stand 2004, Daten von [DESTATIS])

15 Wettervorhersagedaten in GoogleEarth

Das Unternehmen Google [Google1] ist seit einiger Zeit nicht nur für seine Suchmaschine bekannt, sondern auch für die Desktopanwendung *GoogleEarth* [Google2]. Mit GoogleEarth können Satelliten- und Luftbilder in unterschiedlichen Auflösungen angezeigt werden. Zusätzlich existiert ein Geländemodell, so dass auch Höhenangaben verfügbar sind und es können weitere Geodaten eingeblendet werden. Daher liegt es nahe, auch Wetterdaten in GoogleEarth einzubinden. Dies kann zum einen als Pixelgrafik und zum anderen als vektorielle Darstellung realisiert werden.

Im Folgenden wird GoogleEarth kurz vorgestellt und es wird gezeigt, wie georeferenzierte Daten in GoogleEarth eingefügt werden können.

15.1 Grundlagen

In GoogleEarth wird die Erde als dreidimensionale Kugel angezeigt, auf die die Bilder projiziert werden. Sie kann beliebig gedreht und herangezoomt werden. Dabei werden immer wieder pixelbasierte Satellitenaufnahmen mit weiteren Details nachgeladen, die die alten Bilder ersetzen. In der höchsten Auflösung sind teilweise sogar Personen zu erkennen. Allerdings schwankt die Auflösung der Bilder je nach Region sehr stark. Teilweise werden auch militärische Einrichtungen oder Regierungsgebäude nur in sehr geringer Qualität dargestellt.

Die Software wird von Google kostenlos für die Betriebssysteme Windows (2000, XP), Mac OS X und Linux zur Verfügung gestellt. Ursprünglich wurde die Software von der Firma Keyhole entwickelt, die dann im Jahr 2004 von Google aufgekauft wurde. Seitdem ist die Software unter dem Namen GoogleEarth bekannt. Vor allem die Verwendung von Satellitenbildern hat GoogleEarth schnell bekannt werden lassen, da Satellitenbilder einen faszinierenden Überblick über unsere Erde geben und solch umfangreiches und detailreiches Kartenmaterial bisher nicht frei zugänglich war.

Aufgrund der detailreichen und interessanten Satellitenbilder, die zur Verfügung gestellt werden, wird GoogleEarth oftmals nur als unterhaltsamer Zeitvertreib gesehen. Dabei bietet GoogleEarth vielfältige Möglichkeiten der Anzeige georeferenzierter Daten. So können Ortsmarken (Points of interest), Straßen, Grenzen, Bilder und Vektorgrafikelemente erzeugt und eingeblendet werden. Neben diesen zweidimensionalen Objekten können auch dreidimensionale Objekte (z.B. Gebäude) in GoogleEarth eingebunden werden (Abbildung 15.1). Diese können ab der Version 4 sogar mit Texturen versehen werden, so dass man einzelne Gebäude sehr realistisch nachmodellieren kann. Allerdings wird in dieser Arbeit darauf nicht näher eingegangen, sondern lediglich auf die zweidimensionalen Anzeigeelemente.

15.2 Die Keyhole Markup Language (KML)

Alle in GoogleEarth vorhandenen Elemente werden in der *Keyhole Markup Language* (KML) definiert. KML liegt aktuell in der Version 2.1 vor und ist eine XML Ausprägung

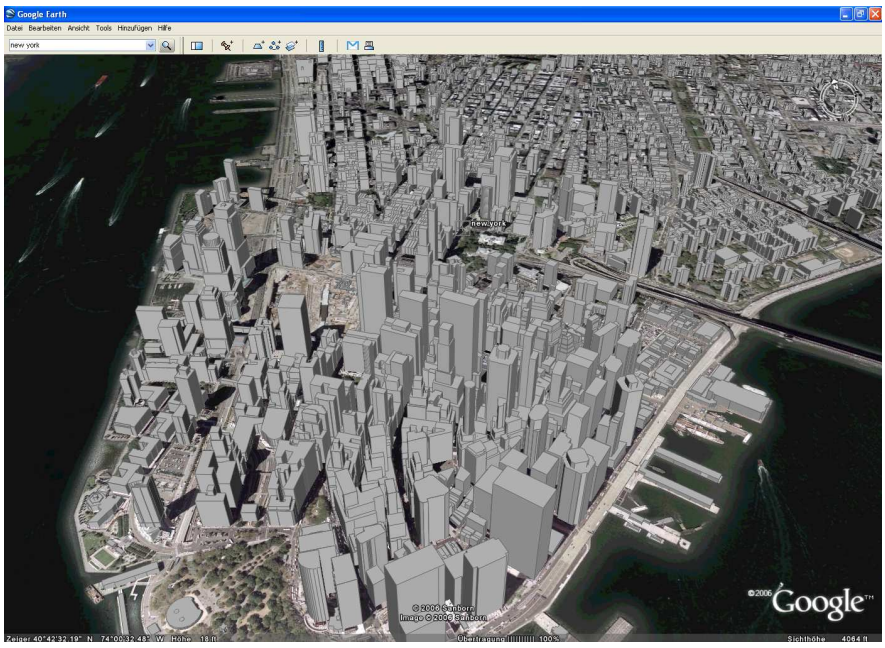


Abbildung 15.1: 3D Gebäude in GoogleEarth

zur Modellierung und Speicherung räumlicher Strukturen, die in dem GoogleEarth Clienten angezeigt werden können. Generell werden KML und KMZ Dateien unterschieden. KMZ Dateien sind mit dem ZIP Algorithmus komprimierte KML Dateien und beinhalten gegebenenfalls noch zusätzlich benötigte Dateien.

Unter anderem kann in KML folgendes definiert werden:

- Ordner zur besseren Strukturierung der Daten
- Icons und Label, um Orte zu lokalisieren
- Verschiedene Kamerapositionen
- Imageoverlays, die auf der Erdkugel positioniert werden können
- Linienzüge und Polygone als Vektorgrafik
- Unterschiedliche Style Angaben der eingefügten Objekte
- Dynamisches Laden von Informationen

KML Dateien können direkt aus der GoogleEarth Anwendung heraus erstellt und abgespeichert werden. Allerdings können so nur recht einfache KML Dateien erzeugt werden. So kann eine Ortsmarke oder ein Imageoverlay leicht erstellt und als KML Datei abgespeichert werden. Komplexere Objekte müssen jedoch per Hand in KML geschrieben oder mit geeigneter Software erzeugt werden, wie z.B. mit Google Sketchup, einer ebenfalls von Google kostenlos bereitgestellten Software zur Erzeugung von 3D Objekten.

Um die Funktionsweise von KML zu verdeutlichen und um zu zeigen, wie georeferenzierte Daten und im Speziellen die Wetterdaten in GoogleEarth eingebunden werden können, soll die Keyhole Markup Language näher vorgestellt werden.

15.2.1 Placemark

Besonderheiten und Sehenswürdigkeiten in den Karten, so genannte *Points of interest (POI)* können markiert und mit einem Text oder einem weiterführenden Link zu einer Webcam, einem Panorama oder erläuternden Webseiten versehen werden. Solche Markierungen werden in GoogleEarth *Placemark* genannt. Dies kann einfach über die GoogleEarth Software realisiert werden, indem man die betreffende Stelle markiert und über das Menü einen Placemark (in der Deutschen Version „Ortsmarke“) hinzufügt. Der Placemark wird in einer Bibliothek abgelegt, die in der GoogleEarth Applikation auf der linken Seite zu finden ist. Ein Placemark enthält die Koordinaten des Punktes, einen Namen und gegebenenfalls eine Beschreibung. Jeder Placemark kann mit einem Icon versehen werden, der dann auf dem Globus dargestellt wird.

Als registrierter Nutzer bei der GoogleEarth Community kann man einen Placemark über die GoogleEarth Software veröffentlichen. Der zu veröffentlichende Placemark wird automatisch in ein Forum eingetragen, wo er zur Diskussion gestellt wird. Bestehen keine Einwände, wird der Placemark veröffentlicht und ist jedermann zugänglich. Der Anwender braucht nur den Layer der Google Community einzuschalten, um diese zusätzlichen Informationen zu sehen.

Außer der GoogleEarth Community gibt es auch andere Projekte, die sich mit der Erfassung von geographischen Daten befassen. Bei Wikipedia gibt es zum Beispiel das „WikiProjekt Georeferenzierung“ [Wikipedia2]. Ziel ist es, die Wikipedia Artikel zu georeferenzieren. In GoogleEarth können dann die Placemarks zu den Artikeln eingebunden werden. Klickt der Betrachter auf ein Placemark öffnet sich eine Webseite mit dem zugehörigen Wikipedia Artikel.

Ein einfaches KML File zur Definition eines Placemarks ist in Quellcode 15.1 angegeben. Das Wurzelelement `kml` enthält eine Ortsmarke, welche mit dem Tag `Placemark` definiert wird. Dazu wird mit `name` ein Name definiert und mit `description` eine Beschreibung angegeben. Der Name wird in der Karte eingeblendet, die Beschreibung wird erst sichtbar, wenn man auf das in der Karte eingeblendete Icon klickt. Im Tag `LookAt` wird der Augenpunkt des Betrachters definiert. Mit dem Breiten- und Längengrad (`longitude` und `latitude`) wird angegeben wohin man schaut, bzw. wohin eine Kamerafahrt stattfindet, wenn man auf die Ortsmarke in der eigenen Bibliothek klickt. Das Element `range` gibt die Entfernung des Augenpunktes zur Erde an. Als letztes folgt die Definition der Koordinaten für die Ortsmarke. Die drei Werte geben den Längen- und Breitengrad, sowie die Höhe des Punktes über normal Null an. Das Ergebnis ist in Abbildung 15.2 dargestellt.

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://earth.google.com/kml/2.1">
  <Placemark>
    <name>Universität Osnabrück, AVZ</name>
    <description>
      Das AVZ beherbergt unter anderem das
      Institut für Informatik
    </description>
    <LookAt>
      <longitude>8.025512518896594</longitude>
      <latitude>52.28418414618795</latitude>
      <range>1002.371881027298</range>
    </LookAt>
    <Point>
      <coordinates>
        8.025512518896594,52.28418414618795,0
      </coordinates>
    </Point>
  </Placemark>
</kml>
```

Quellcode 15.1: Definition eines Placemarks in KML

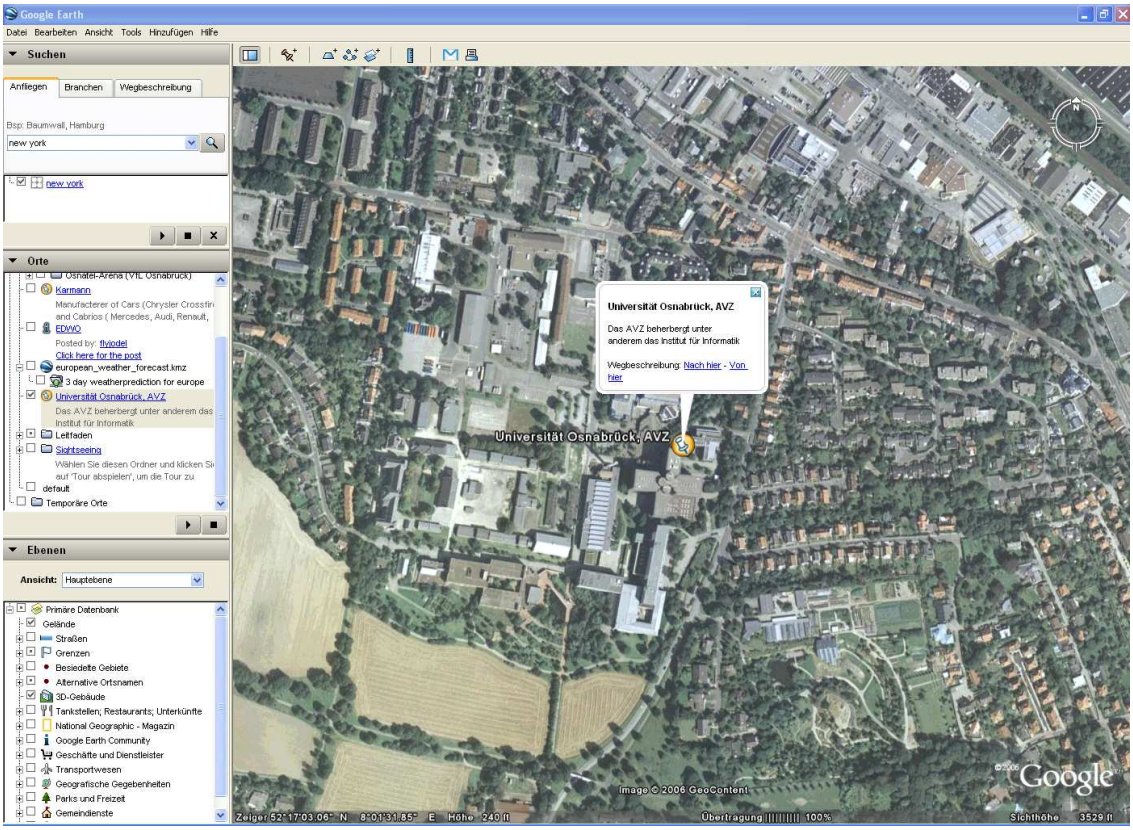


Abbildung 15.2: Anzeige einer Ortsmarke (*Placemark*) in GoogleEarth

15.2.2 ImageOverlay

Neben den Ortsmarken können auch *ImageOverlays* erstellt werden. Dabei wird eine Pixelgrafik über das Satellitenbild von GoogleEarth gelegt. In der GoogleEarth Software ist ein ImageOverlay sehr schnell zu erstellen. Dazu kann man über das Menü ein ImageOverlay hinzufügen. Im sich öffnenden Dialog kann man einen Namen und eine Beschreibung angeben, sowie eine Bilddatei öffnen, die angezeigt werden soll. Das Bild sollte dabei zum Koordinatensystem in GoogleEarth kompatibel sein. GoogleEarth verwendet geographische Koordinaten im WGS84 Format [SnyderP]. Eine senkrecht zur Erde aufgenommene Luftaufnahme kann in der Regel genau passend eingeblendet werden. Schräg aufgenommene Bilder müssen vorher entzerrt werden.

Sind die geographischen Koordinaten des Bildes bekannt, können diese eingetragen werden und das Bild befindet sich an der passenden Position. Falls die Koordinaten nicht bekannt sind, kann der Benutzer das Bild auf der Erdkugel in Breite und Höhe skalieren, das Bild drehen und die Position verändern. Mit etwas Geschick können auch so qualitativ gute ImageOverlays erzeugt werden.

Damit das eingebundene Bild nicht die Satellitenbilder komplett verdeckt, kann ein Alphakanal definiert werden, sodass das Bild leicht transparent dargestellt wird.

In KML wird ein ImageOverlay wie in Quellcode 15.2 angegeben definiert. Das Ergebnis ist in Abbildung 15.3 dargestellt.

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://earth.google.com/kml/2.1">
 <Document>
  <GroundOverlay>
   <name>
     Osnabrueck-Schinkel-Overlay
   </name>
   <color>d8ffffff</color>
   <Icon>
    <href>images/luftbild.jpg</href>
    <viewBoundScale>0.75</viewBoundScale>
   </Icon>
   <LatLonBox id="khLatLonBox583">
    <north>52.28415961337301</north>
    <south>52.27467224758016</south>
    <east>8.076559262389482</east>
    <west>8.06143816509397</west>
    <rotation>0.7772554105059113</rotation>
   </LatLonBox>
  </GroundOverlay>
 </Document>
</kml>
```

Quellcode 15.2: Ein ImageOverlay in KML

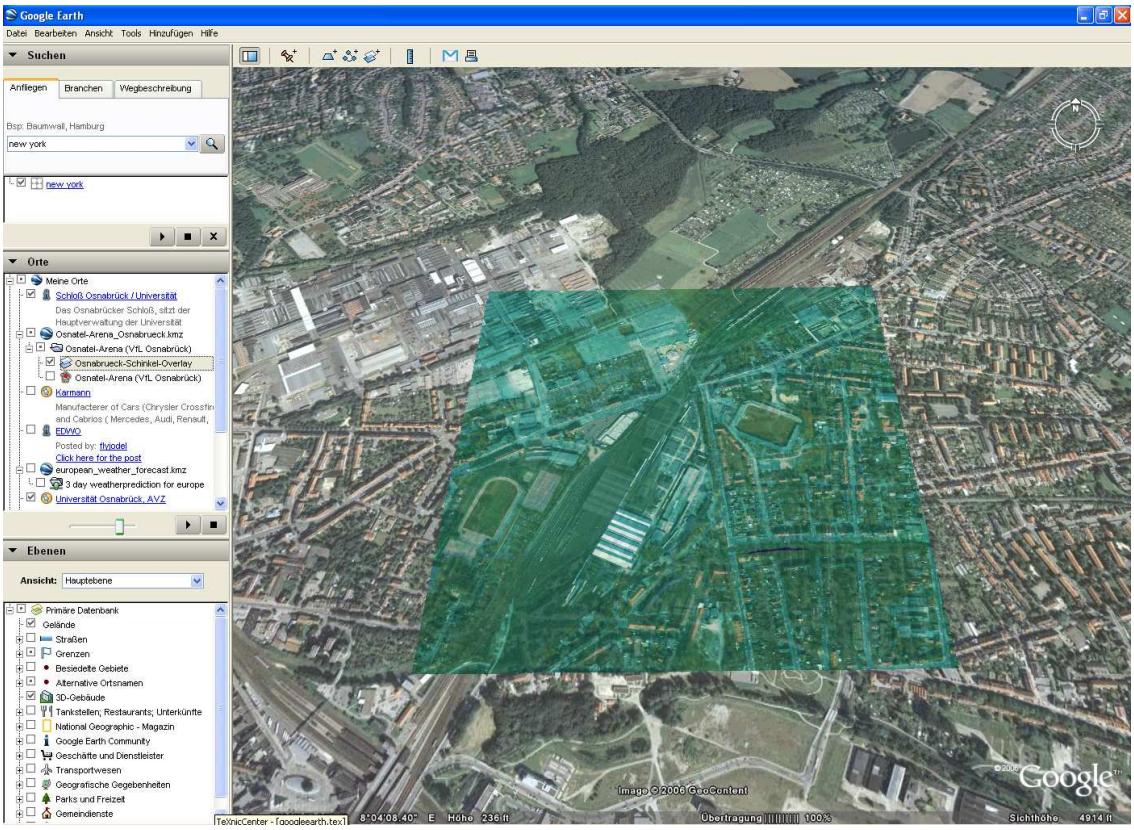


Abbildung 15.3: ImageOverlay in GoogleEarth

Um ein ImageOverlay zu erzeugen wird das Tag `GroundOverlay` angegeben. Mit diesem Tag werden alle Elemente eingeleitet, die auf das Satellitenbild gelegt werden sollen. Zusätzlich wird ein Name definiert. Im `color`-Tag wird der Alphakanal für die Transparenz des Bildes angegeben. Eine Farbe im `color`-Tag wird in hexadezimalen Werten angegeben. Die ersten beiden Stellen geben den Alphakanal an, die jeweils nächsten zwei Stellen den Blau-, Grün- und Rotanteil.

Im Tag `Icon` wird das eigentliche Bild über die Pfadangabe eingebunden. In diesem Fall liegt das Bild in einem Verzeichnis `images`.

Um die Größe des Bildes anzupassen wird mit dem Tag `viewBoundScale` ein Skalierungsfaktor angegeben. Als letztes folgt die `LatLonBox`, die Boundingbox des Bildes. Hier werden die Koordinaten der Eckpunkte und eine Rotation des Bildes angegeben.

Ein ImageOverlay kann in GoogleEarth direkt als KMZ Datei abgespeichert werden. Diese Datei kann man dann an andere GoogleEarth Nutzer z.B. per EMail weitergeben. Es beinhaltet alle notwendigen Informationen und Dateien für dieses ImageOverlay. Die KMZ Datei kann man auch selbst erzeugen, indem man die KML Datei und das Bild im entsprechenden Verzeichnis zu einem ZIP Archiv hinzufügt. Danach muss lediglich die Dateiendung von `zip` auf `kmz` geändert werden.

15.2.3 Polygone

In GoogleEarth können auch Polygone angezeigt werden. Dazu müssen diese als Vektorgrafik in KML vorliegen. In GoogleEarth können Polygone nur in der kostenpflichtigen Version direkt erstellt werden.

Polygone können mit Style Angaben versehen werden, so dass das Erscheinungsbild verändert werden kann. Es kann eine Füllfarbe, eine Linienfarbe (jeweils mit Transparenz) und eine Liniendicke angegeben werden. Der KML Code zur Definition eines Polygon ist im Quellcode 15.3 angegeben.

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://earth.google.com/kml/2.0">
<Document>
    <Placemark>
      <name>Einfaches Polygon</name>
      <Style>
        <PolyStyle>
          <color>b27f00ff</color>
        </PolyStyle>
      </Style>
      <Polygon>
        <outerBoundaryIs>
          <LinearRing>
            <coordinates>
               -35,70,0 25,70,0 25,30,0 -35,30,0 -35,70,0
            </coordinates>
          </LinearRing>
        </outerBoundaryIs>
      </Polygon>
    </Placemark>
</Document>
</kml>
```

Quellcode 15.3: Ein einfaches Polygon in KML

Zunächst wird die Füllfarbe des Polygons im `Style`-Tag angegeben. Auf die Angabe einer Linienfarbe wird hier verzichtet. Danach folgt die Definition des Polygons. Dazu werden innerhalb des `Polygon`-Tags die äußeren Grenzen im Tag `outerBoundaryIs` angegeben. Dazu werden die Koordinaten im `coordinates` Tag als durch ein Leerzeichen getrennte Tripel angegeben. Jedes Tripel besteht aus dem Längen- und Breitengrad sowie der Höhe.

Aus dem Polygon können auch einzelne Flächen „ausgestanzt“ werden, so dass sich ein oder mehrere Löcher im Polygon ergeben. Dazu werden innerhalb des `Polygon`-Tags ein oder mehrere `innerBoundaryIs`-Tags angegeben. Ein Beispiel ist in Quellcode 15.4 zu sehen. Das Ergebnis in GoogleEarth ist in Abbildung 15.4 dargestellt.

```
<Polygon>
  <outerBoundaryIs>
    <LinearRing>
      <coordinates>
        -35,70,0 25,70,0 25,30,0 -35,30,0 -35,70,0
      </coordinates>
    </LinearRing>
  </outerBoundaryIs>
  <innerBoundaryIs>
    <LinearRing>
      <coordinates>
        -30,60,0 20,60,0 20,35,0 -30,35,0 -30,60,0
      </coordinates>
    </LinearRing>
  </innerBoundaryIs>
</Polygon>
```

Quellcode 15.4: Ein Polygon mit Loch in KML

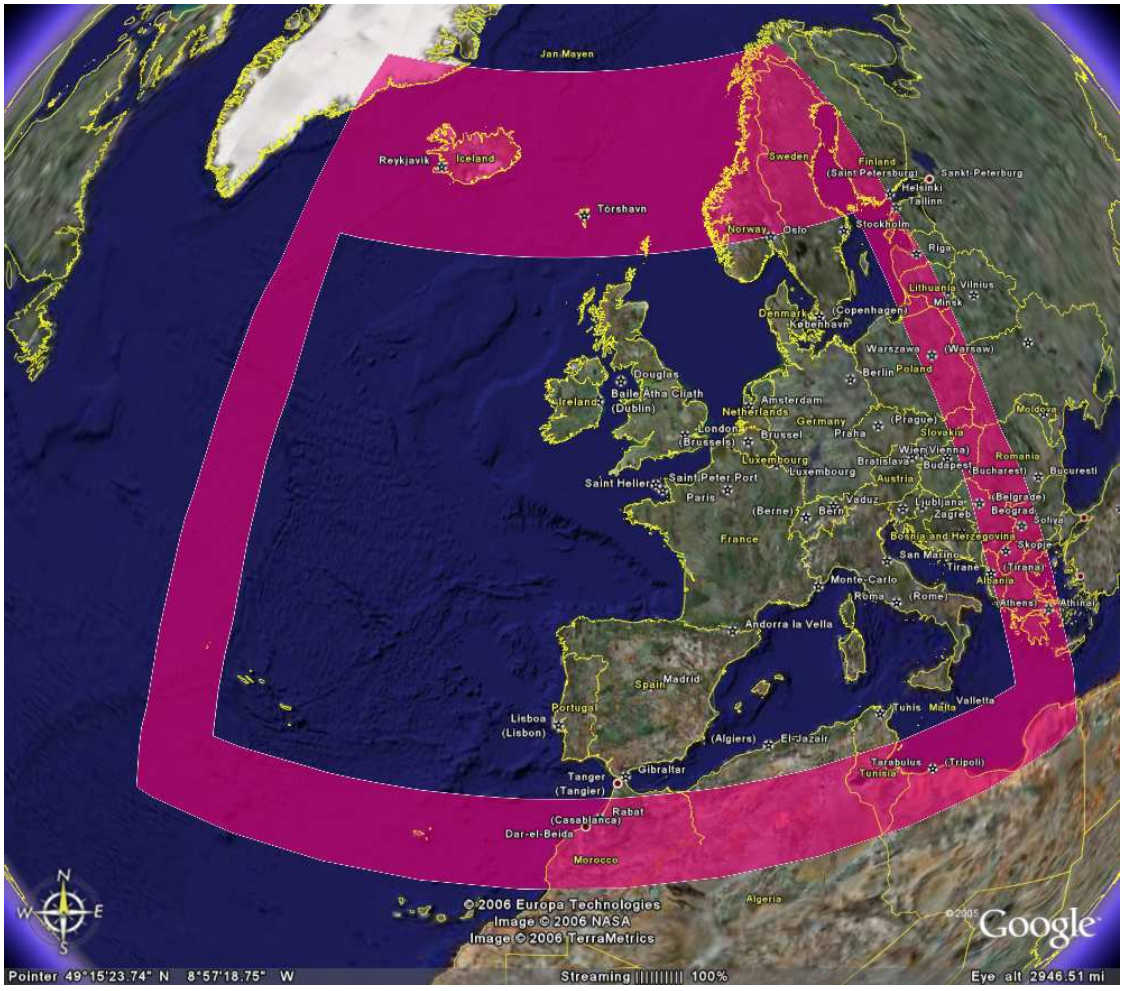


Abbildung 15.4: Polygon mit Loch in GoogleEarth

15.3 Einbinden der Wetterdaten mittels ImageOverlay

Wie in Kapitel 15.2.2 beschrieben, ist es sehr einfach, ein Bild in GoogleEarth einzubinden. Um dies mit den vektorisierten Wetterdaten (Kapitel 10) in SVG zu machen, muss die Vektorgrafik der Wetterdaten erst in eine Pixelgrafik umgewandelt werden.

Durch das Templatekonzept (Kapitel 9) liegen die Ergebnisse der Vektorisierung separat als SVG-Fragment vor. Es fehlen lediglich die `svg`-Tags am Anfang und am Ende der Datei. Jeder einzelne Datensatz zu einem bestimmten Zeitpunkt befindet sich in einer separaten Datei. Die Datei muss um das `svg`-Tag ergänzt werden, damit es eine gültige SVG Datei ist und kann dann mittels Inkscape [Inkscape] in eine Pixelgrafik umgewandelt werden. Inkscape ist ein Programm zum Erstellen von SVG Grafiken und kann SVG Dateien in andere Formate wie z.B. Encapsulated Postscript, JPEG, PNG, usw. konvertieren. Dies ist nicht nur mit der grafischen Oberfläche von Inkscape möglich sondern auch ohne GUI auf der Kommandozeile. Um eine SVG Datei als PNG zu speichern lautet der Befehl:

```
inkscape -b 0 -d 1200 -D -e datei.svg datei.png
```

Der Parameter `-b 0` gibt an, dass der Hintergrund der SVG Grafik als transparent angesehen werden soll. Dies ist notwendig, da aneinandergrenzende Flächen durch Rundungsfehler minimale Lücken aufweisen. Diese Lücken würden bei einem nicht transparenten Hintergrund mit Linienfragmenten versehen, wodurch die Qualität der Pixelgrafik leidet. Mit `-d 1200` wird die Auflösung des Pixelbildes in `DPI` (Dots per Inch) angegeben. Hier wird eine recht hohe Auflösung verwendet, damit die Qualität des Bildes bei einem heranzoomen nicht zu stark leidet. Der Parameter `-e` gibt letztendlich an, dass die SVG Datei in eine PNG Datei umgewandelt werden soll.

Durch diese Umwandlung erhält man eine Pixelgrafik mit den Temperaturwerten. Da in den GRIB Dateien neben den Temperaturwerten auch die geographischen Koordinaten des Rasters angegeben sind, kann man die KML Datei zum Einbinden des Bildes einfach erzeugen.

Wird die Prozesskette per Shellskript durchgeführt, ist es ein Einfaches, die Bilder immer wieder zu aktualisieren und ein KMZ File mit allen Ressourcen zu erstellen. Allerdings muss ein GoogleEarth Anwender diese Datei immer wieder herunterladen, da sich die Bilder nicht automatisch aktualisieren.

Um dieses Problem zu umgehen, werden die Bilder nicht in das KMZ File eingebunden, sondern separat auf einem Webserver gelegt. Dann kann eine KML Datei erzeugt werden, die der Benutzer in GoogleEarth lädt und die dann die auf dem Server liegenden Bilder nachladen kann (Quellcode 15.5).

Im `refreshMode`-Tag wird angegeben, dass sich das Bild intervallweise aktualisieren soll. Mit `refreshInterval` wird die Zeit in Sekunden angegeben, nach der das Bild aktualisiert wird.

Auf diese Art und Weise werden immer aktuell erzeugte Bilder angezeigt. Allerdings muss die Beschreibung im `name`-Tag in der vom Benutzer geladenen KML Datei, die

```
<GroundOverlay>
   <name>Temperatur: from 27.6.2006  1:00</name>
   <color>a3ffffff</color>
   <Icon>
      <refreshMode>onInterval</refreshMode>
      <refreshInterval>3600</refreshInterval>
      <href>http://webserver/pfad/temperatur.png</href>
   </Icon>
   <LatLonBox>
     <north>70</north><south>30</south>
     <east>25</east><west>-35</west>
   </LatLonBox>
</GroundOverlay>
```

Quellcode 15.5: Intervallweises Nachladen eines Bildes in KML

auszugsweise im Quellcode 15.5 zu sehen ist, auch aktualisiert werden. So muss z.B. das Datum immer wieder angepasst werden. Daher muss nicht nur für die Bilder sondern auch für die KML Datei selbst eine Aktualisierung vorgenommen werden. Dazu wird serverseitig die KML Datei erzeugt, in der die Bilder referenziert werden und die eine passende Beschreibung enthält.

Diese Datei wird nicht direkt vom GoogleEarth Anwender geladen, sondern er muss einmalig eine KML Datei laden, in der auf die serverseitige verwiesen wird, damit auch hier eine automatische Aktualisierung stattfindet. Dazu wird das Tag `NetworkLink` verwendet, das eine externe Quelle referenziert (Quellcode 15.6).

```
 <NetworkLink>
   <name>
     3 day weatherprediction for europe
   </name>
   <Url>
     <refreshMode>onInterval</refreshMode>
     <refreshInterval>3600</refreshInterval>
     <refreshVisibility>1</refreshVisibility>
     <href>
       http://webserver/pfad/file.kml
     </href>
   </Url>
 </NetworkLink>
```

Quellcode 15.6: Intervallweises nachladen einer KML Datei

Über das Tag `Url` werden Angaben zu der externen Quelle gemacht. Dazu wird die Adresse im `href` Tag angegeben und mit `refreshMode` und `refreshInterval` wird die Datei stündlich aktualisiert. Mit `refreshVisibility` wird angegeben, ob die neu geladene Datei gleich angezeigt wird (`1`) oder nicht (`0`). Das Ergebnis ist in Abbildung 15.5 zu sehen.

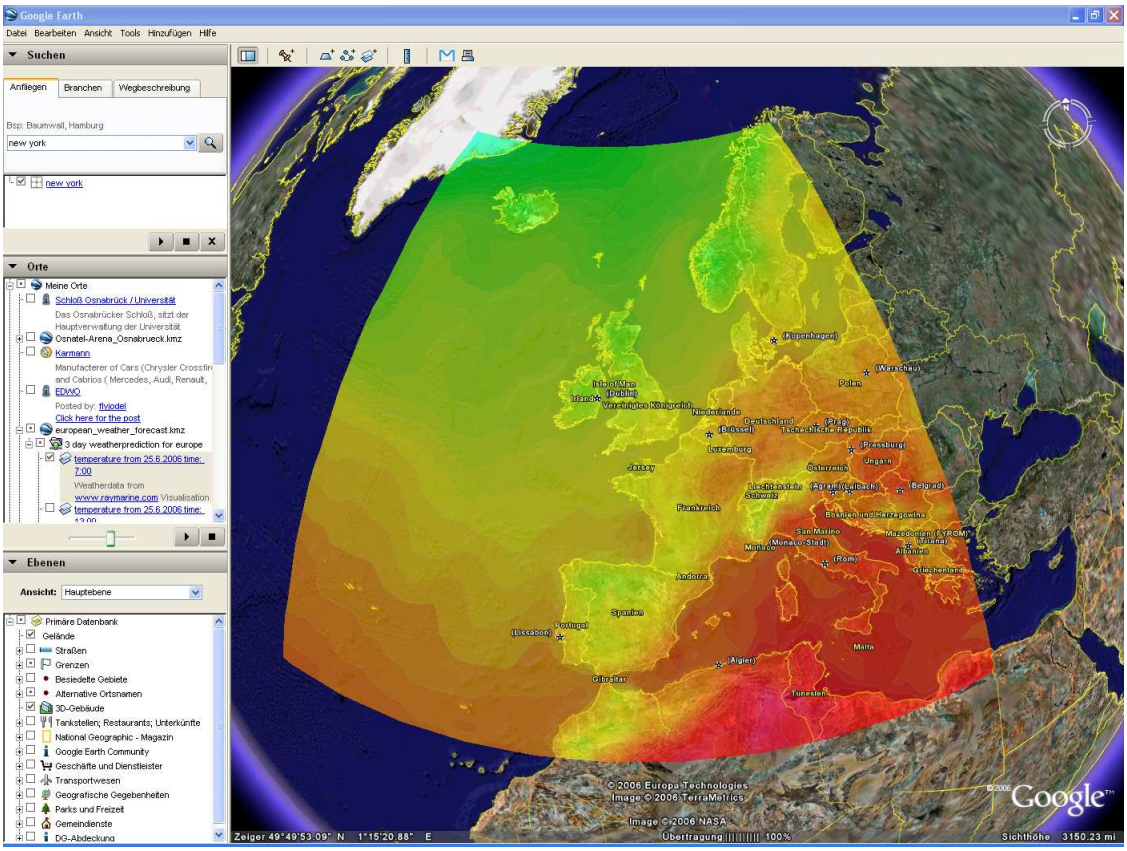


Abbildung 15.5: Temperaturvorhersage in GoogleEarth

Die Methode des ImageOverlay hat jedoch den Nachteil, dass die vorab in SVG berechneten Vektordaten zu einem Pixelbild umgerechnet werden, was bei einem starken Einzoomen zu einer mangelnden Qualität führt. Außerdem können keine feineren Details oder Text eingeblendet werden, da diese beim Einzoomen nicht mehr erkennbar wären.

15.4 Vektorgrafik in GoogleEarth

Wie bereits in Kapitel 15.2.3 gezeigt können auch Polygone in GoogleEarth eingebunden werden. Da die Daten aus den GRIB Files ohnehin vektorisiert werden (Kapitel 10), liegt es nahe, aus diesen vektorisierten Daten Polygone in KML zu erzeugen.

Da bei der Erstellung der SVG Web Mapping Applikation die Vektorisierung integriert ist, wird die Erzeugung der KML Datei mit den Isoflächen in die Applikation mit eingebunden. Dabei muss beachtet werden, dass in SVG das Koordinatensystem gespiegelt ist, was für die Angabe der geographischen Koordinaten wieder rückgängig gemacht werden muss. Das korrekte Ausstanzen der Flächen erfolgt ebenfalls bei der Erstellung der Web Mapping Applikation (Kapitel 13.7).

In KML gibt es die Möglichkeit mit dem Tag `Folder` einzelne Ordner anzulegen. So kann für einen Wetterdatensatz eines bestimmten Zeitpunktes ein Ordner angelegt werden, in dem die Isoflächen nach ihren Werten gruppiert wiederum in einzelnen Ordnern

abgelegt sind. So können ganz gezielt Isoflächen mit einem bestimmten Wert ein- oder ausgeschaltet werden. In der Abbildung 15.6 sind die Temperaturwerte von null bis zehn und von zwanzig bis dreißig Grad Celsius ausgewählt worden.

Wurde dies alles beachtet, kann das KML File einfach erzeugt werden. Die Datei wird dann auf einem Webserver abgelegt und der GoogleEarth Anwender muss lediglich einmal eine KML Datei einbinden, in der die serverseitige Datei mit den entsprechenden Parametern zur Aktualisierung referenziert wird.

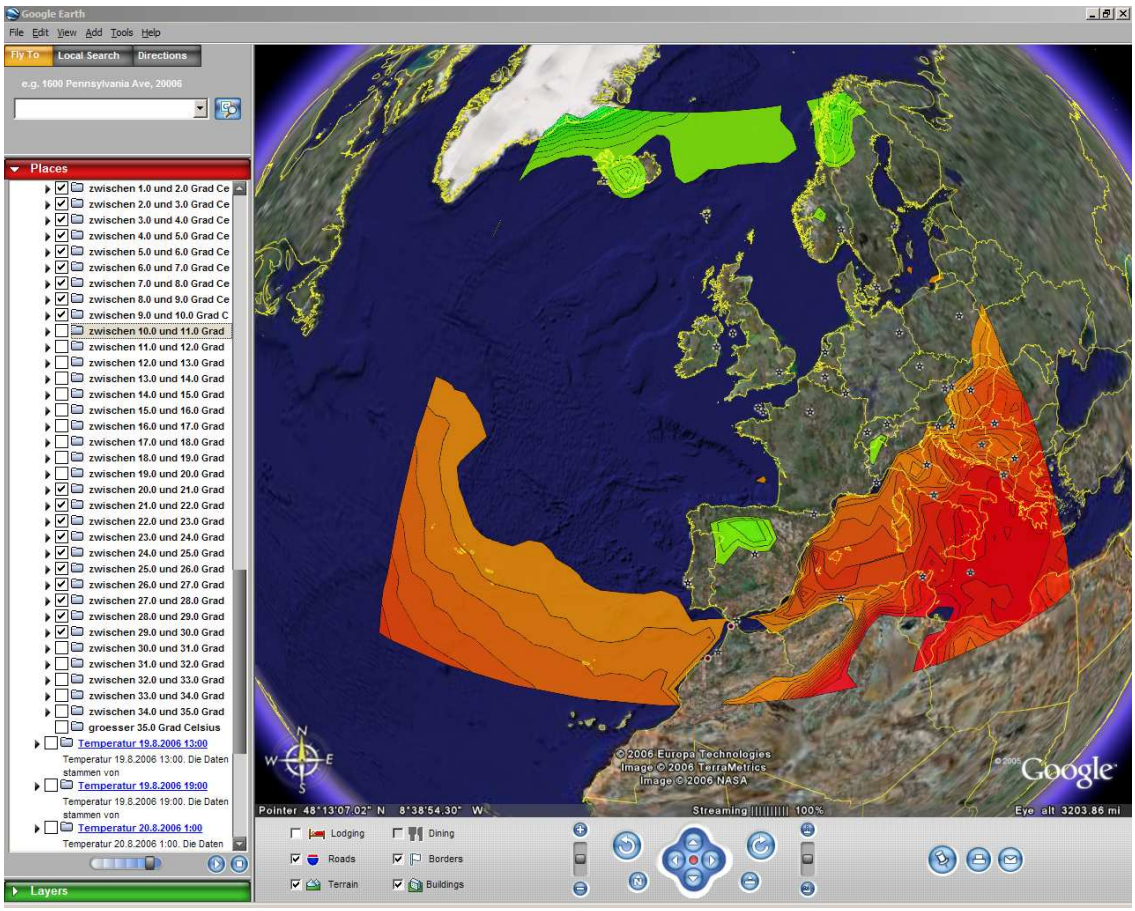


Abbildung 15.6: Ausgewählte Temperaturbereiche als Vektorgrafik in GoogleEarth

16 Verbreitung der Daten mittels Digital Audio Broadcast (DAB)

Neben der Entwicklung einer SVG Applikation im Webbrowser wurden auch andere Verbreitungswege untersucht, um die Visualisierungen auch mobilen Endgeräten zugänglich zu machen. Auf solchen mobilen Endgeräten wird aufgrund der gegenüber einem PC geringeren Hardwareausstattung nicht das komplette SVG Profil umgesetzt. Speziell für mobile Endgeräte wurde daher das SVG Tiny Profil vom W3C definiert [W3C9], welches eine Untermenge des SVG Standards darstellt.

Handys besitzen zwar auch die Möglichkeit eine Internetverbindung aufzubauen und Daten aus dem Internet zu laden und vor allem mit dem neuen UMTS Standard ist die Bandbreite hoch genug, um auch größere Datenmengen aus dem Internet zu laden. Allerdings sind die Internetzugänge via Handy immer noch zu kostenintensiv. Auch mobile Geräte ohne Internetzugang sollten in der Lage sein eine Wetterdarstellung zu erlauben. Vorstellbar ist hier ein vereinfachter PDA, der in einem Wohnzimmer aufgestellt wird und nur dazu dient die Wettervorhersage anzuzeigen (Abbildung 16.1).

Abbildung 16.1: Wetterdarstellung auf einem PDA (Illustration)

Eine Möglichkeit um Daten zu verteilen, bietet *Digital Audio Broadcast* (DAB). An der Universität Osnabrück wird zur Zeit die Machbarkeit des Vorhabens evaluiert [Kunze].

16.1 DAB Verbreitung

DAB ist in Deutschland auch als Digitalradio bekannt. Diese für den mobilen Empfang optimierte Technik soll die analoge *Frequenzmodulation* (FM) auf *Ultrakurzwelle* UKW ablösen. Der erste DAB-Sender Deutschlands wurde im Jahr 2000 in Bayern installiert. Deutschlandweit liegt die Sendeabdeckung bei 80 Prozent und es gibt derzeit etwa 120 DAB-Programme, viele davon sind aber nur lokal zu empfangen. Da die Sendeanstalten aus Kostengründen nur geringe Bitraten bei der Übertragung verwenden ist die Klangqualität zur Zeit kaum besser als bei UKW. Dazu kommt, dass die Sendeleistung noch nicht ausreichend ist. Daher ist der so genannte Inhouse-Empfang relativ schlecht, was eine Außenantenne notwendig macht. Allerdings soll die Sendeleistung im Jahr 2006 verzehnfacht werden, um dieses Problem zu beheben.

Das Verfahren wurde zum europäischen Standard gewählt und es existiert eine EU Richtlinie [EU], nach der der analoge Rundfunk bis zum Jahr 2010 abgeschaltet werden soll und nur noch digitaler Rundfunk existieren soll. Deutschland hat dieser Empfehlung zugestimmt und vertraglich zugesichert, den analogen Rundfunk bis zum Jahr 2010 abzuschalten. Inwieweit dieses Vorhaben umgesetzt werden kann, bleibt abzuwarten. Immerhin würde es bedeuten, dass alle analogen Rundfunkempfänger ausgetauscht werden müssten, was mit immensen Kosten für die Bevölkerung verbunden wäre. Auch wenn bis zum Jahr 2010 der analoge Rundfunk nicht abgeschaltet wird, hat Deutschland jedoch ein klares Signal für DAB gesetzt, so dass davon ausgegangen werden kann, dass sich DAB auf Dauer durchsetzen wird.

16.2 Technische Grundlagen

Grundsätzlich werden mehrere Sender im Paket übertragen. Für die Übertragung werden mehrere Audiodatenströme zusammen mit ebenfalls möglichen reinen Datendiensten zu einem so genannten Ensemble mit hoher Datenrate zusammengeführt.

Der so entstandene Multiplex wird mittels *Coded Orthogonal Frequency Division Multiplex* (COFDM) moduliert [Wikipedia4]. Dieses Verfahren ist im Vergleich zur analogen Ausstrahlung deutlich robuster gegenüber Störungen. Zudem ist es möglich, weite Flächen mit nur einer Frequenz abzudecken (Gleichwellennetz). Dadurch ist die Frequenzökonomie, also der Verbrauch von Spektrum je Programm, bei DAB meist deutlich besser. Ein Nachteil ist jedoch der bereits veraltete *Musicam-Audiocodec* (MPEG 1 Layer 2), der doppelt so viel Frequenzen benötigt, wie der neuere *Advanced Audio Coding-Codec*.

DAB verwendet in Deutschland zur Übertragung Frequenzen im Band III (174-230 MHz), zumeist den ehemaligen Fernsehkanal 12 (223-230 MHz), sowie im L-Band den Bereich von 1452 bis 1492 MHz. Band III findet Verwendung in den landesweit ausgestrahlten Ensembles, das L-Band wird zur Ausstrahlung lokaler Multiplexe genutzt. Die Frequenzen im L-Band erfordern wegen der hohen Frequenz eine große Senderdichte, die nicht immer gewährleistet ist. Dies führt gerade bei lokalen Stationen häufig zu Empfangsproblemen.

Neben den Radioprogrammen als Audiostream können auch andere Daten übertragen werden. Dabei werden grundsätzlich zwei Arten unterschieden: PAD (*Program associated data*) und NPAD (*No program associated data*). Ersteres besteht in der Regel aus Informationen zur aktuellen Sendung, wie z.B. der Name des gespielten Musikstückes oder Informationen zu laufenden Nachrichten. Diese Daten dürfen maximal 128 Zeichen umfassen. NPAD wird mittels des *Multimedia Object Transfer* (MOT) Protokolles übertragen. Hier können im Prinzip beliebig große Dateien versendet werden. Beim MOT ist es zudem vorgesehen, dass ganze Datei- oder HTML Baumstrukturen übertragen werden können. Die zu übertragenden Daten werden in einem Datenkarussel vorgehalten und die Daten werden der Reihe nach ausgestrahlt. Sind alle Daten ausgestrahlt, beginnt die Übertragung wieder von Neuem.

16.3 Machbarkeit

Von der Digitalradio Nord GmbH [DRN] (Sendeverantwortlich für Hamburg, Bremen, Schleswig Holstein, Mecklenburg Vorpommern und Niedersachsen) wurde ein zwei KBit Datenstrom zur Verfügung gestellt, mit dem die Übertragungsraten und die Fehlertoleranz einem ersten Test unterzogen werden konnte. Zu beachten sind verschiedene Aspekte, wie die Dauer der Datenübertragung oder die Fehlertoleranz der Übertragung.

16.3.1 Dauer der Datenübertragung

Für die Visualisierung einer 24 Stunden Wettervorhersage für Deutschland benötigt man für die Temperatur-, Luftdruck-, Bewölkung-, Niederschlags- und Winddaten jeweils 24 SVG-Datensätze, die das Wetter zur jeden vollen Stunde darstellen. Die 24 SVG Dateien für die Temperatur sind sieben Megabyte groß. DAB verringert die Datenmenge automatisch, indem die Dateien zip-komprimiert übertragen werden. Die zu übertragende Datenmenge reduziert sich auf ca. zwei Megabyte. Die Übertragungszeit dieser Datenmenge beträgt bei einem zwei KBit Stream ungefähr vier Stunden und dauert somit viel zu lange. Daher muss die Datenmenge durch eine Verringerung der Auflösung oder durch eine geschickte Kodierung der Daten im SVG File reduziert werden.

16.3.2 Fehlertoleranz

Da DAB nicht über einen Rückkanal verfügt und eine fehlerhafte Datei nicht erneut vom Client angefordert werden kann, muss bei der Übertragung der Daten eine möglichst hohe Fehlertoleranz gegeben sein. DAB verfügt auf Protokollebene über ein Fehlerkorrekturverfahren, die so genannte Vorwärtskorrektur. Vor der Ausstrahlung einer Datei werden ihr in redundanter Form Daten hinzugefügt, so dass die Datei auch bei geringen Fehlern bei der Datenübertragung auf der Clientseite wieder hergestellt werden kann. Allerdings ist bei größeren Fehlern oder Empfangspausen eine Vorwärtskorrektur nicht mehr möglich. Hier kommt das Karussellverfahren zum Einsatz, bei dem die Datenübertragung von vorne beginnt, sobald alle Dateien einmal übertragen wurden. Um so weniger Daten

im Datenkarussell vorhanden sind, um so schneller werden die Daten wiederholt übertragen. Dies ist für die Fehlerkorrektur hilfreich. Wenn ein DAB Empfänger die Daten der Reihe nach empfängt, plötzlich aber beispielsweise in einem U-Bahn Tunnel verschwindet, kann er die folgenden Daten nicht mehr empfangen. Ein Datenpaket ist also fehlerhaft oder es fehlen Datenpakete, um eine Datei zu komplettieren. Der Empfänger muss also warten, bis erneut die fehlenden Daten ausgestrahlt werden.

16.3.3 Weiterführende Entwicklung

Zunächst bleibt abzuwarten, wann DAB in Deutschland in voller Sendeleistung und kompletter Abdeckung vorhanden sein wird. Erst wenn dies absehbar ist, werden mobile Geräte auf den Markt kommen, mit denen DAB empfangen werden kann. Bis dahin müssen die Tests mit einem DAB Empfänger für den Desktop PC vorgenommen werden.

Um die Verbreitung der Wetterdaten über DAB zu realisieren, sind weitere umfangreichere Tests notwendig. Als erstes muss die zu übertragende Datenmenge reduziert werden. Dafür müssen neue Konzepte zur optimierten Datenspeicherung in SVG untersucht werden. Dabei spielt die Verwendung von Prototypen, Patterns und CSS eine wichtige Rolle. Der zweite sehr wichtige Punkt ist die Fehlerminimierung bei der Datenübertragung. Hier müssen neben der Vorwärtskorrektur andere Konzepte ausgearbeitet werden, mit denen eine optimale Datenversorgung gegeben ist.

17 3D Visualisierung von Klima- und Wetterdaten

Die in der Arbeit gewonnen Erkenntnisse sollen genutzt werden, um auch eine dreidimensionale Visualisierung von Klima- und Wetterdaten zu erstellen. So können die Module zum Einlesen eines GRIB Files und zur Vektorisierung genutzt werden. Letzteres wird benötigt, um Isolinien darzustellen. Isoflächen werden im dreidimensionalen nicht dargestellt, da hier effizientere Methoden mit Farbverläufen gewählt werden können. Außerdem profitiert die Darstellung in 3D von den Erkenntnissen im Bereich der Projektion geographischer Daten.

Um die Erde darzustellen, kann sie als zweidimensionale Projektion einer Kugel visualisiert werden. Die Erdkugel ist dann auf einen Blick zu sehen. Hierbei erhält man einen sehr guten Überblick über die Klimadaten. Eine solche Darstellung ist ohne weiteres mit der SVG Web Mapping Applikation möglich. Aber sollen Klimawerte auf der gesamten Erde visualisiert werden, ist die Darstellung als dreidimensionales Objekt ebenso sehr gut nutzbar. Insbesondere wenn man die Erdkugel drehen und interessante Regionen vergrößern kann. Eine solche Visualisierung hat den Vorteil, dass intuitiv mit ihr gearbeitet werden kann. Außerdem bietet eine dreidimensionale Visualisierung noch andere Möglichkeiten der Darstellung der einzelnen klimatischen Kenngrößen. Daher wurde in einem an der Universität Osnabrück im Rahmen einer Bachelorarbeit durchgeführten Projekt eine dreidimensionale Darstellung mittels C++ und OpenGL entwickelt [Wenke].

17.1 OpenGL

OpenGL ist eine Schnittstelle zwischen Anwenderprogrammen und der Grafikkartenhardware. Mit OpenGL können dreidimensionale Objekte modelliert und animiert werden. OpenGL ist in drei Teile gegliedert [Davis]:

OpenGL Library enthält ca. 200 Befehle (beginnend mit `gl`, um elementare geometrische Primitive, wie Punkte, Linien, Polygone, Kurven und deren Attribute wie Farben zu verwalten.

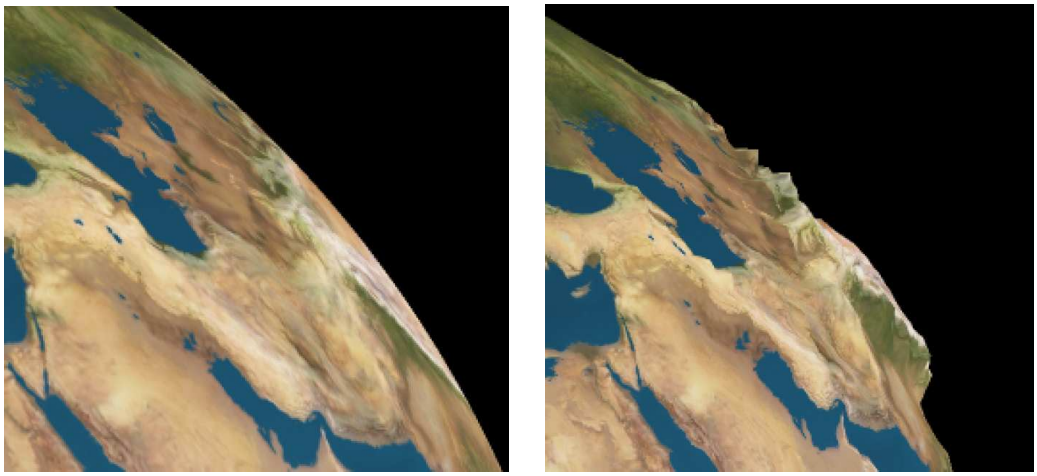
OpenGL Utility Library ca. 50 Befehle (beginnend mit `glu`), um NURBS (non uniform rational b-splines) und runde Körper (Kugeln, Zylinder) zu verwalten.

OpenGL Utility Toolkit ca 30 Befehle (beginnend mit `glut`), um die gerenderte Ausgabe an das jeweilige Fenstersystem auszugeben und komplexe geometrische Objekte zu verwalten.

Generell arbeitet OpenGL zustandsbehaftet. Ein gesetzter Zustand wird beibehalten, bis ein neuer definiert wird, oder aber bis er zurückgesetzt wird. Dadurch müssen nicht alle Farben oder Transformationsmatrizen immer wieder neu definiert werden. Außerdem ist es implementatorisch einfacher, den Zustand immer nur um die benötigten Änderungen zu ergänzen, da so Objekte besser in der Welt relativ zueinander platziert werden können. Soll ein Objekt nur um einige Einheiten zum vorherigen verschoben werden, so ist nur diese Verschiebung anzugeben. Es muss also keine Gesamtmatrix bestimmt werden, sondern nur die relative Änderung.

17.2 Umsetzung der Visualisierung

Um einen möglichst realitätsnahen Eindruck von der Erde zu erhalten, wurde die Kugelgestalt durch Displacementmapping angepasst, um Berge und Täler sichtbar zu machen (Abbildung 17.1).



(a) Ohne Displacementmapping (b) Mit Displacementmapping

Abbildung 17.1: Vergleich der Erdtextur

Auf die aufwändige Berechnung von Isolinien und Isoflächen wurde verzichtet, da in OpenGL Farbübergänge sehr einfach berechnet werden können. So wird auf der Erdkugel ein Gitter aufgebracht, welches dem Raster das darzustellenden GRIB Files entspricht. Darauf werden beispielsweise die Temperaturwerte eines GRIB Files einer Farbe zugeordnet und diese Farbwerte werden in das Raster der Erdkugel eingetragen. Mit OpenGL kann angegeben werden, dass das Raster automatisch triangularisiert wird und die Dreiecke mit einem Farbverlauf gefüllt werden (Abbildung 17.2).

Durch die dreidimensionale Darstellung lassen sich insbesondere Wolken sehr gut darstellen (Abbildung 17.3). Dazu wird eine zweite Kugel definiert, die etwas größer als die Erdkugel ist. Der Radius der Vertices der äußeren Kugel werden entsprechend der Bewölkungsstärke verändert:

$$Radius_i = Normale_i \cdot [(\frac{Bewoelkungsgrad_i}{c} + Kugelradius)]$$

Der Index i identifiziert den betrachteten Vertex. Der Parameter $Normale_i$ enthält die Normale am betrachteten Vertex, c ist ein konstanter Faktor, der passend gewählt werden muss, damit die Wolken nicht zu stark Verschoben werden. Der Bewölkungsgrad für den Vertex an der Stelle i ist in der Variablen $Bewoelkungsgrad_i$ angegeben.

Um auch eine zeitliche Komponente darstellen zu können, werden zwischen jedem Zeitpunkt Zwischenbilder berechnet, die der Reihe nach gezeigt werden, wodurch weiche Übergänge möglich werden.

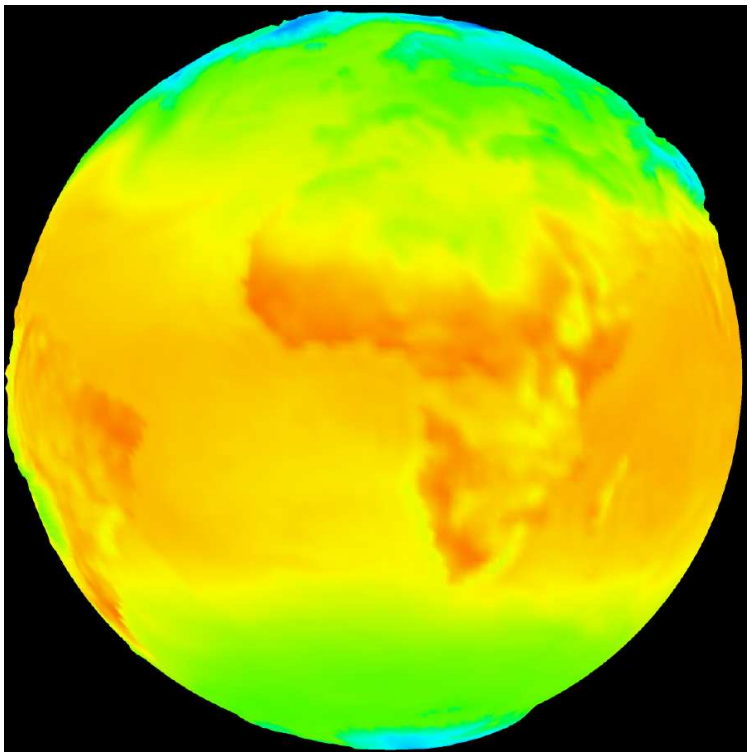


Abbildung 17.2: Visualisierung von Temperaturverläufen mittels OpenGL

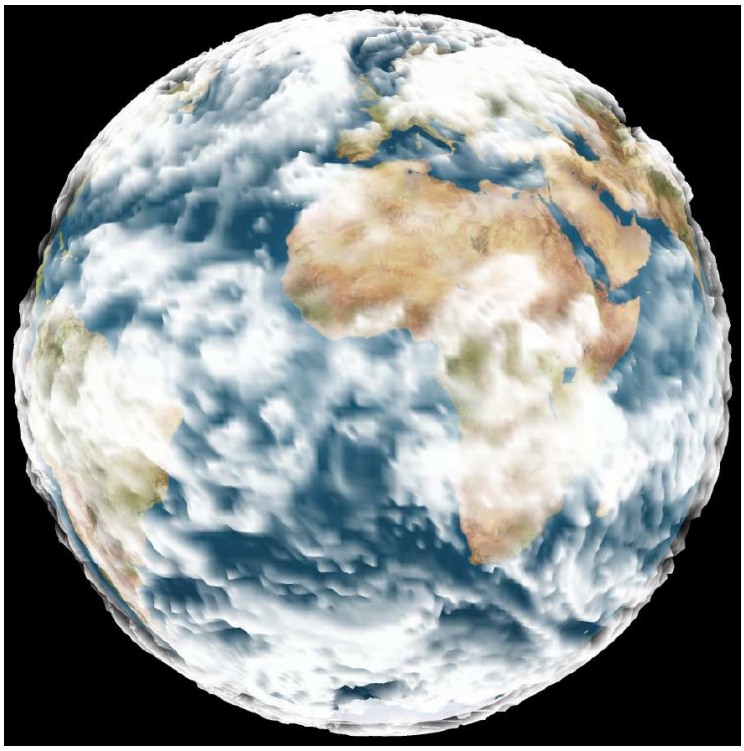


Abbildung 17.3: Visualisierung von dreidimensionalen Wolken mittels OpenGL

17.3 Weiterführende Entwicklung

Durch die Verwendung von C++ als Programmiersprache ist die Anwendung sehr performant, allerdings nicht mehr ohne weiteres plattformunabhängig. Daher soll untersucht werden, inwieweit eine dreidimensionale Visualisierung mit Java implementiert werden kann. Zur Zeit wird in einem Projekt die Möglichkeit getestet, die 3D Visualisierung über ein Java Applet zugänglich zu machen. Dazu wird JOGL [JOGL] verwendet. JOGL ist eine externe OpenGL-Programmbibliothek für die Programmiersprache Java und wurde ursprünglich von Kenneth Russell und Chris Kline entwickelt. Mittlerweile wird sie aber als Open Source, mit Unterstützung von Sun Microsystems, weiterentwickelt und verbessert. JOGL ist für folgende Plattformen erhältlich:

- Windows/x86
- Linux/x86
- Solaris/SPARC 2.8+
- Solaris/x86 2.9+
- Mac OS X 10.3+

Erste Tests ergaben, dass eine dreidimensionale Visualisierung in einem Java-Applet mit JOGL möglich ist (Abbildung 17.4). Allerdings muss die Performance weiter verbessert und verschiedene Visualisierungstechniken erprobt werden.

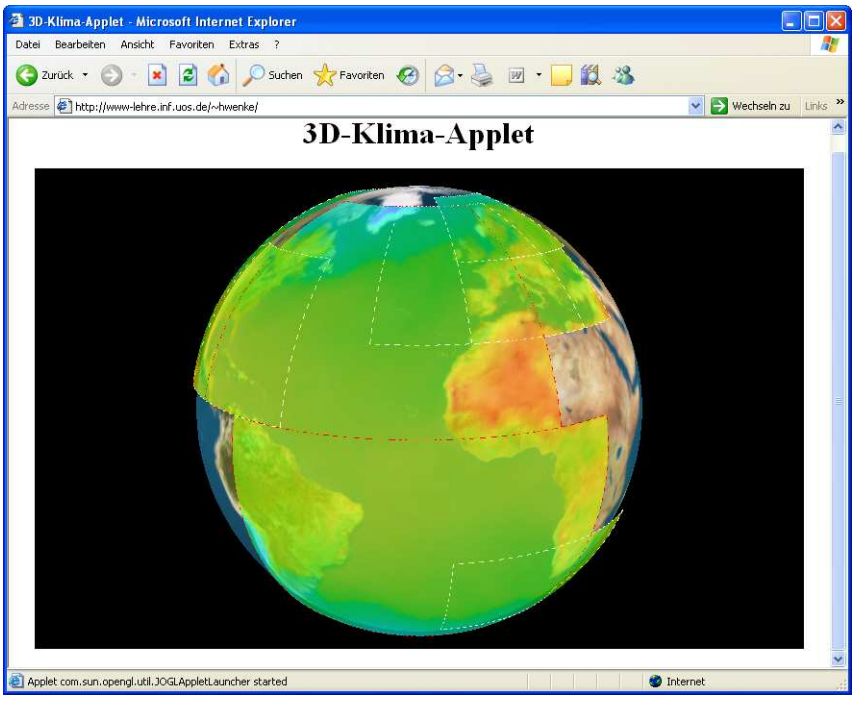


Abbildung 17.4: Dreidimensionale Darstellung von Temperaturwerten in einem Java Applet

V Resümee

18 Ausblick

Mit der in dieser Arbeit entwickelten Anwendung ist es möglich, interaktives Kartenmaterial und interaktive Wettervorhersagen in einer SVG Web Mapping Applikation darzustellen. Durch die automatisierte Prozesskette können sich häufig ändernde Daten einfach auf dem aktuellen Stand gehalten werden. Die Daten können zusätzlich in GoogleEarth dargestellt werden. Experimentell ist eine Visualisierung mit OpenGL implementiert worden. Darüber hinaus wurde die Übertragung der Daten über Digital Audio Broadcast getestet. Im Folgenden sollen weitere Möglichkeiten und Ideen aufgezeigt werden, wie die Anwendung erweitert werden könnte.

18.1 Morphing von SVG Elementen

Beim Morphing zwischen zwei Bildern werden Zwischenbilder berechnet, die nacheinander abgespielt den Eindruck erwecken, dass die beiden Einzelbilder ineinander übergehen. Morphing ist nicht mit der reinen Überblendung von Bildern zu vergleichen, bei der das eine Bild immer transparenter wird, bis nur noch das nächste Bild zu sehen ist.

Bisher können bei einer Wettervorhersage die verschiedenen Zeitpunkte nur durch Einzelbilder dargestellt werden. Es wäre wünschenswert, wenn bei der Visualisierung von Wetterdaten auch die zeitliche Dimension durch Morphing ästhetischer präsentiert werden könnte. Dabei ist zu beachten, dass eine Wetterdatenvisualisierung möglichst ohne manuellen Eingriff erzeugt werden sollte, da sich die Wetterprognosen mehrmals täglich ändern. Nur durch eine automatisierte Erstellung kann ohne großen Aufwand die Aktualität der Wettervorhersage gewahrt werden. Genau in diesem Punkt haben die meisten Morphingalgorithmen Defizite. Sie benötigen einen manuellen Eingriff, da die Morphingalgorithmen meist nicht erkennen können, von wo nach wo ein Objekt wandert. Dadurch können beim Morphing Überschneidungen der einzelnen Objekte entstehen, wodurch der visuelle Eindruck getrübt wird.

Es gibt Bestrebungen, ein überschneidungsfreies Morphing zu realisieren [Efrat]. Dazu muss erkannt werden, welche Punkte zwischen zwei Zeitschritten zusammengehörig sind. Wie man solche Punkte, so genannte Featurepoints, finden kann ist bei [Liu] beschrieben. Diese Ansätze wurden in einer Bachelorarbeit an der Universität Osnabrück näher untersucht [Albrecht].

Prinzipiell sind alle Voraussetzungen gegeben, um ein Morphing realisieren zu können. Allerdings wird das Morphing von Isolinien durch die SVG Struktur praktisch unmöglich. In SVG selbst ist Morphing nicht vorgesehen. Will man zwischen zwei Bildern dennoch morphen, ist es erforderlich, den Vorgang in JavaScript zu implementieren. Dazu muss für jeden Punkt und für jeden Übergang zwischen zwei Bildern eine JavaScript Funktion definiert werden, die die Animation eines Punktes beschreibt. Dies ist problematisch, da die Berechnung der Bewegung der Punkte im Client zu viel Rechenleistung in Anspruch nimmt. Wenn beispielsweise 100 Objekte mit JavaScript auf unterschiedliche Art und Weise animiert werden soll, beginnt die Animation bereits zu ruckeln [Fox]. Ein weiteres Problem stellt die Art der Speicherung der Koordinaten eines Pfades dar. So werden

die Koordinaten in SVG in einem Attribut als String der Reihe nach abgelegt (Abschnitt 4.1), so dass ein bestimmter Punkt nur schwer zu identifizieren ist. Außerdem können die Punkte nicht gleichzeitig mittels JavaScript auf unterschiedlichen Pfaden animiert werden.

In SVG ist echtes Morphing also nicht zufriedenstellend zu realisieren, es sei denn, in Zukunft werden vom W3C entsprechende SVG Spezifikationen entworfen und in den SVG Viewern implementiert.

18.2 Unterstützung mobiler Geräte

Es werden immer mehr PDAs und Handys entwickelt, die Farbbildschirme und eine adäquate Prozessorleistung haben und so in der Lage sind, eine verbesserte Multimedia Unterstützung zu bieten. Derartige Geräte zeichnen sind im Vergleich zu einem PC durch einen geringen Speicher, eine niedrige CPU Leistung und begrenzte Darstellungsmöglichkeiten aus, sind aber dennoch in der Lage standardisierte Webtechnologien wie XHTML, SMIL und SVG darzustellen. Das W3C hat aus der SVG 1.1 Spezifikation zwei Profile bzw. Untergruppen erstellt [W3C8]: SVG Tiny (SVGT), für die Multimediafähigkeit für Mobiltelefone und SVG Basic (SVGB) für Handhelds und Palmtops. Im August 2006 wurde die neueste Candidate Recommendation für SVGT1.2 vorgestellt [W3C9]. Zur Zeit werden schon zahlreiche Handys angeboten, die eine SVGT1.1 Unterstützung bieten. Unter [SVG] findet sich eine laufend aktualisierte Übersicht.

Da die Zahl der SVG Tiny fähigen Endgeräte steigt, liegt es nahe, die SVG Web Mapping Applikation auch diesen Geräten zugänglich zu machen. Da in der vorliegenden Arbeit in erster Linie die Verbreitung im Internet betrachtet werden sollte, wurde die SVG Tiny Unterstützung nicht weiter verfolgt.

18.3 Unterstützung zusätzlicher Datenformate

Zur Zeit werden Shapefiles und GRIB Dateien bei der Erstellung der Web Mapping Applikation unterstützt. Beide stellen in ihrer Anwendungsdomäne einen Quasistandard dar, der weltweit eingesetzt wird. Dennoch gibt es weitere Datenformate, die zur Speicherung georeferenzierter Daten verwendet werden. Dabei handelt es sich unter anderem um:

Geographic Data Format (GDF): Ein europäisches Dateiaustauschformat für nicht binäre vektorisierte Kartendaten, im Speziellen für Straßenkarten. Es wurde von der Telematik Industrie entwickelt und wird unter der Norm ISO/DIS 14825 [ISO] beschrieben.

Geography Markup Language (GML): Ein Datenformat zum Austausch raumbezogener Objekte mit Attributen, Relationen und Geometrien. GML wird vom Open Geospatial Consortium festgelegt und liegt zur Zeit in der Version 3.1.1 [OGC] vor .

Network Common Data Form (NetCDF): Ein binäres Datenformat zur plattformunabhängigen Speicherung raum- und zeitbezogener Daten, ähnlich dem GRIB Format. Es wurde am Unidata Program Center [Unidata] entwickelt und wird unter anderem zur Speicherung von Klima- und Meeresströmungsdaten verwendet.

In der vorliegenden Arbeit wurde auf eine Unterstützung dieser und anderer Formate verzichtet, da Shapefiles und GRIB Dateien die am weitesten verbreiteten Datenformate zur Speicherung von geographischen und klimatischen Daten sind. Die Implementation eines Filters ist aber durch den modularen Aufbau der Java Applikation ohne weiteres möglich.

18.4 Software zum Bearbeiten einer Konfigurationsdatei

Um die Erstellung einer Web Mapping Applikation zu vereinfachen, wäre eine grafische Benutzeroberfläche zur Bearbeitung der Konfigurationsdatei wünschenswert. Für die Bearbeitung von XML Dateien gibt es zwar verschiedene Programme, die die Bearbeitung vereinfachen, aber der Anwender wird immer noch mit der XML Syntax konfrontiert.

Eine Software zur Erstellung einer Konfigurationsdatei sollte folgende Punkte erfüllen:

- Einlesen von Shapefiles: Die `.dbf`-Dateien sollten eingelesen werden können. Der Anwender kann sich dann einen Überblick über die enthaltenen Informationen verschaffen und gezielt die gewünschten Daten selektieren. Zusätzlich kann festgelegt werden, mit welchen geometrischen Objekten oder Symbolen die Daten dargestellt werden sollen. Des weiteren sollten die Objekte mit Style-Eigenschaften, wie Füllfarbe, Liniendicke etc., versehen werden können.
- Einlesen von GRIB Dateien: Wird in der grafischen Oberfläche eine GRIB Datei geladen, sollten die Metadaten ausgelesen werden. So können Informationen über das enthaltene Gitter und die Anzahl und Datentypen der Records angezeigt werden. Auch hier sollte der Benutzer die Möglichkeit haben, gezielt bestimmte Daten auswählen und die Art der Darstellung festlegen zu können.
- Ändern des Layouts: Über die grafische Benutzeroberfläche sollte das SVG Template verändert werden können. So können die einzelnen Elemente genau positioniert werden.

19 Fazit

Den bisherigen Wettervorhersagen mangelte es vor allem an der Möglichkeit individuell eine Region verlustfrei zu vergrößern. Ebenso fehlt die zeitliche Dimension oder ist nur rudimentär vorhanden und die freie Kombination der Daten ist kaum möglich. Einzelne Wetterausprägungen können daher nicht gemeinsam in einer Karte betrachtet werden.

In dieser Arbeit wurde gezeigt, wie diese Mängel behoben werden können und sowohl dem Laien als auch dem Profi eine leicht bedienbare SVG Web Mapping Applikation zur Hand gegeben wird, mit der die zukünftige Wetterentwicklung leicht nachvollzogen werden kann.

Das Java Programm zur Erstellung einer Wettervorhersage ist denkbar einfach zu bedienen. Die Visualisierung kann in einem XML-basierten Konfigurationsfile nach eigenen Wünschen angepasst werden. Dem Java Programm wird beim Start der Pfad des Konfigurationsfiles übergeben. Alle benötigten Dateien werden dann in ein Verzeichnis geschrieben, welches im Konfigurationsfile angegeben wird. Das Verzeichnis braucht nur noch auf einen PHP 5 fähigen Webserver kopiert zu werden. Die Wettervorhersagen können so beispielsweise mittels eines Cronjobs automatisiert und praktisch wartungsfrei auf einem Server erstellt und direkt im Internet veröffentlicht werden. Dadurch fällt die zeitintensive händische Bearbeitung weg und die Wettervorhersagen können sehr komfortabel auf dem neuesten Stand gehalten werden. Nur so ist es einfach und kostengünstig möglich, immer aktuelle Daten im Internet zu präsentieren.

Die Erweiterbarkeit der Java-Anwendung ist durch deren modularen Aufbau gegeben. Es können sehr leicht neue Komponenten implementiert und hinzugefügt werden. Dadurch können weitere Dateiformate eingebunden und neue Visualisierungstechniken hinzugefügt werden. Durch die konsequente Verwendung von Java ist die Software völlig plattformunabhängig. Dies erfordert unter anderem eine Neuimplementation eines GRIB Readers, da Bibliotheken zum Auslesen von GRIB Files nur in anderen Programmiersprachen vorlagen.

In dieser Arbeit wurde gezeigt, dass die Verwendung des eigens entwickelten SVG Template Konzeptes zu einer vereinfachten Entwicklung und Wartbarkeit einer komplexen SVG Anwendung führt. Nur durch die konsequente Aufteilung der einzelnen Komponenten war es bei der Komplexität der SVG Anwendung möglich, den Überblick zu halten. Außerdem konnten so die Datenerzeugung und die Oberflächengestaltung voneinander unabhängig entwickelt werden. Das SVG Template Konzept stellt also keine theoretische Überlegung dar, sondern konnte sich bei der Entwicklung einer umfangreichen SVG Anwendung bewähren.

Bei der SVG Web Mapping Applikation wurde darauf geachtet, dass sie möglichst zu allen Webbrowsern und Plugins kompatibel ist. Um dies zu erreichen wird nur standardisiertes ECMAScript verwendet. Da das Nachladen von serverseitigen Daten im Webbrowsern mit nativer SVG Unterstützung anders verläuft als bei der Verwendung des Adobe SVG Viewers wird zur Laufzeit dynamisch festgestellt, welche Methode angewandt werden muss.

Durch intelligente Mechanismen zum Laden der benötigten Daten, werden nur die wirklich benötigten Informationen übertragen, was den Datenverkehr reduziert und den Speicher des Clients entlastet. Erreicht wird dies durch die Kachelung der darzustellenden Daten. Serverseitig können die Daten auf beliebige Art und Weise verwaltet werden. Lediglich die serverseitigen Skripte müssen auf die Daten zugreifen können.

Neben der Visualisierung einer Wettervorhersage ist es mit der Java Anwendung ebenso möglich, geographische Karten oder andere georeferenzierte Daten zu visualisieren. Außerdem wurde eine Möglichkeit gezeigt, die Wettervorhersage und auch beliebige andere georeferenzierte Daten in GoogleEarth einzubinden. Dadurch wurde sowohl die Variabilität des Java Programmes verdeutlicht, als auch die vielfältigen Möglichkeiten der Datenvisualisierung in GoogleEarth vorgestellt.

In der vorliegenden Arbeit wurde erstmals eine automatisierte Prozesskette entwickelt, mit der eine interaktive und dynamische Wettervorhersage im Internet erstellt werden kann. Die einzelnen Parameter können mit Isolinien und -flächen dargestellt werden. Für die Darstellung von Wind wurde zusätzlich eine Visualisierung mittels Pfeilsymbolen realisiert. Dadurch können alle gängigen Wettererscheinungen dargestellt werden. Zusätzlich können die Wettervorhersagekarten mit umfangreichem und detailreichem Kartenmaterial versehen werden (Abbildung 19.1).

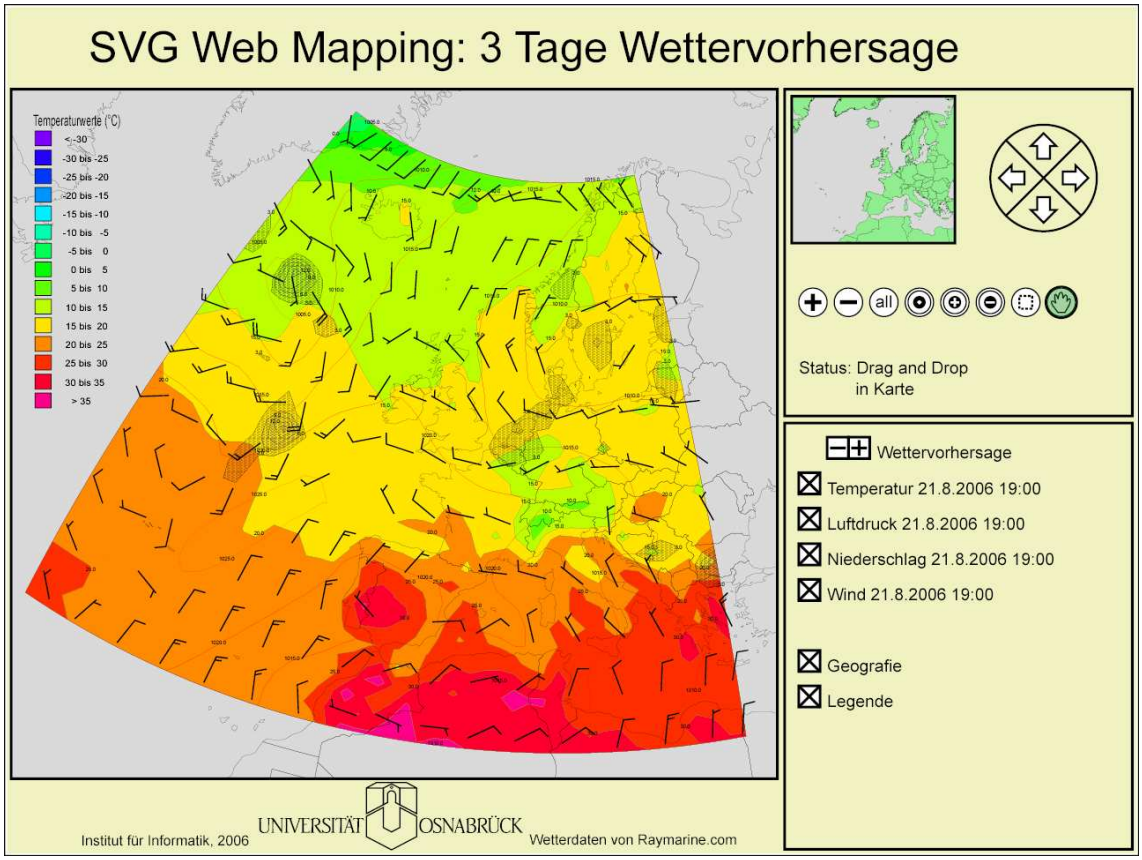


Abbildung 19.1: Wettervorhersage in einer interaktiven SVG Anwendung

VI Anhang

A Screenshots

Mit dem erstellten Javaprogramm kann eine SVG Web Mapping Applikation erzeugt werden. Um einen Überblick über die Darstellungsmöglichkeiten zu erhalten, sind im Folgenden einige Screenshots der SVG Web Mapping Applikation aufgeführt.

A.1 Stadtplan von Osnabrück

Die Intevation GmbH hat Kartenmaterial von Osnabrück digitalisiert und als Open Source Daten [Frida] zur Verfügung gestellt. Mit diesen Daten ist es möglich, einen Stadtplan von Osnabrück in einer SVG Web Mapping Applikation darzustellen.

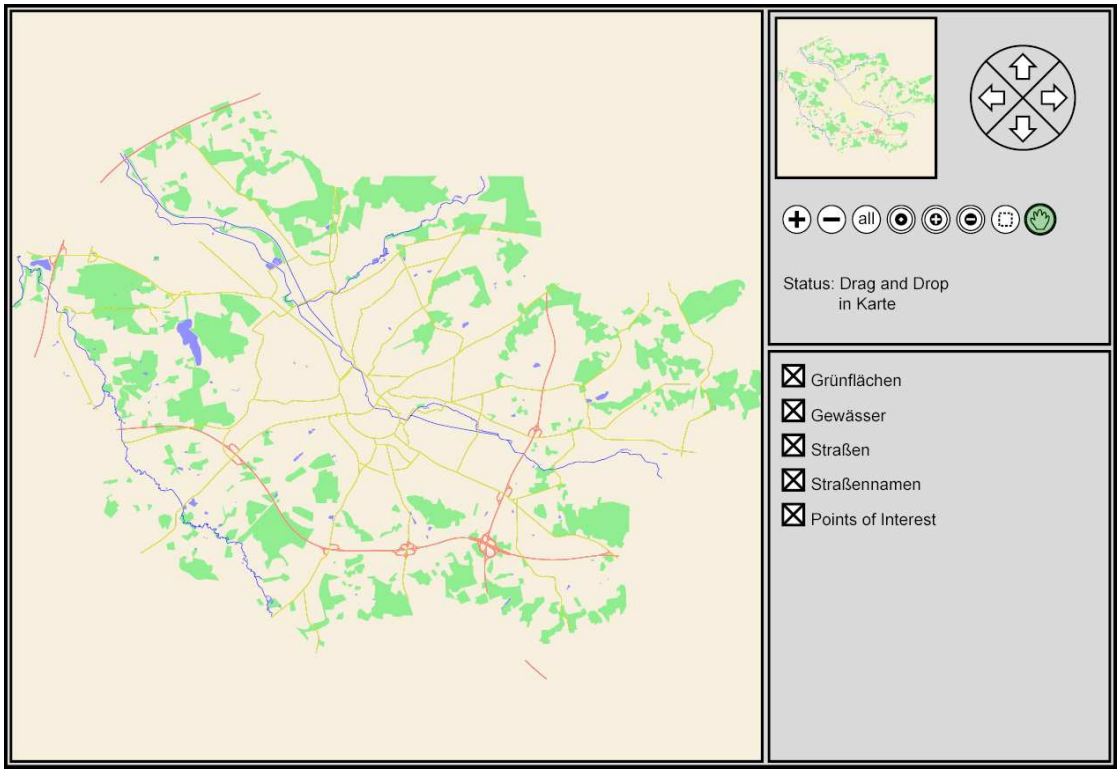


Abbildung A.1: Stadtplan von Osnabrück

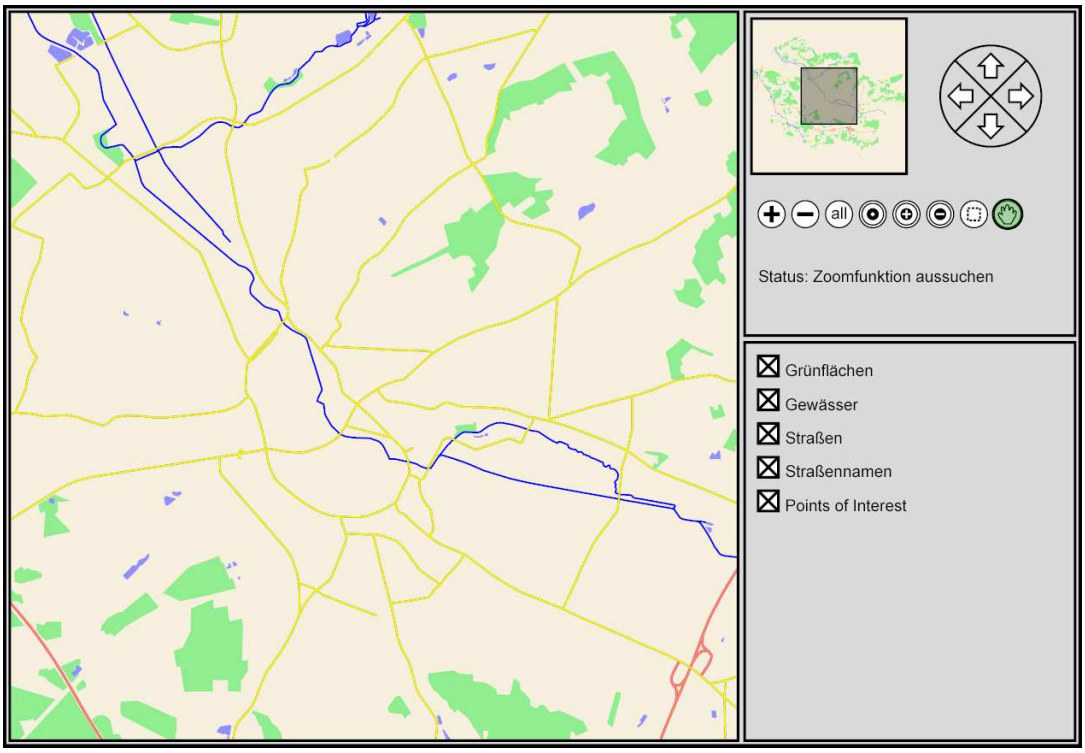


Abbildung A.2: Vergrößerter Stadtplan von Osnabrück

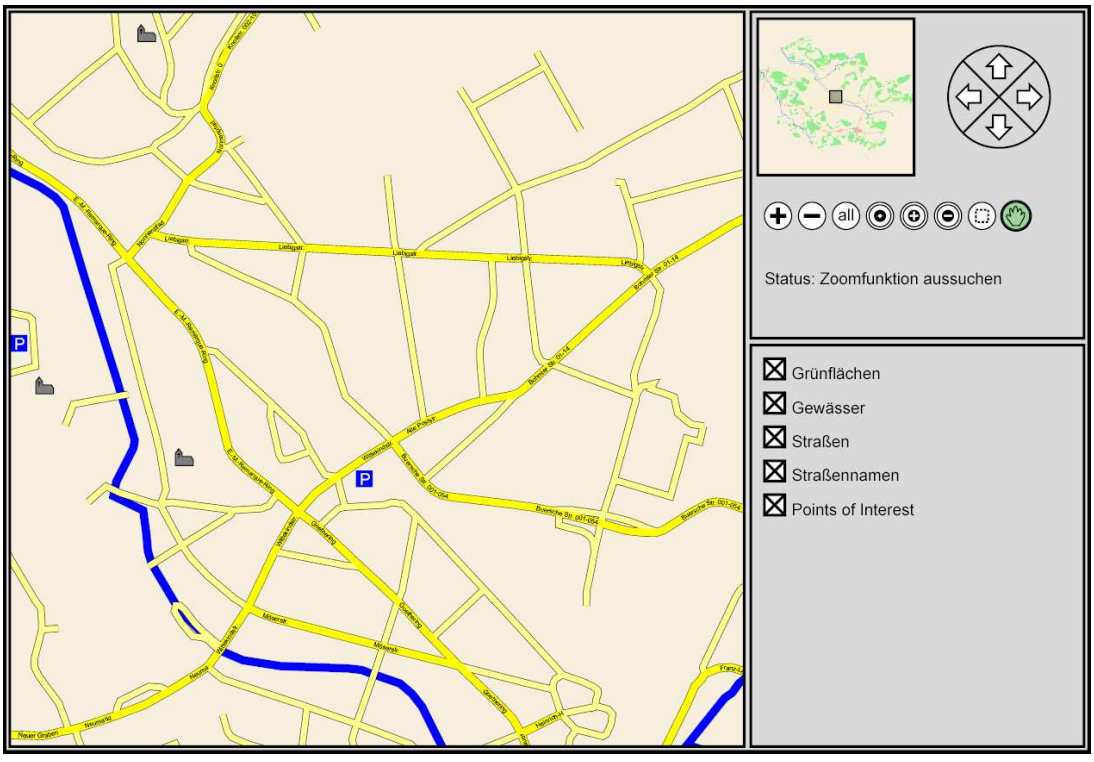


Abbildung A.3: Hinzugeladene Straßen, Straßennamen und Points of Interest

A.2 Wettervorhersage für Europa

In der Wettervorhersage für Europa werden die Temperatur-, Niederschlags-, Luftdruck- und Winddaten [Raymarine] der kommenden drei Tage dargestellt. Die Screenshots zeigen Europa in der Lambertschen Azimuthal Projektion in unterschiedlichen Zoomstufen.

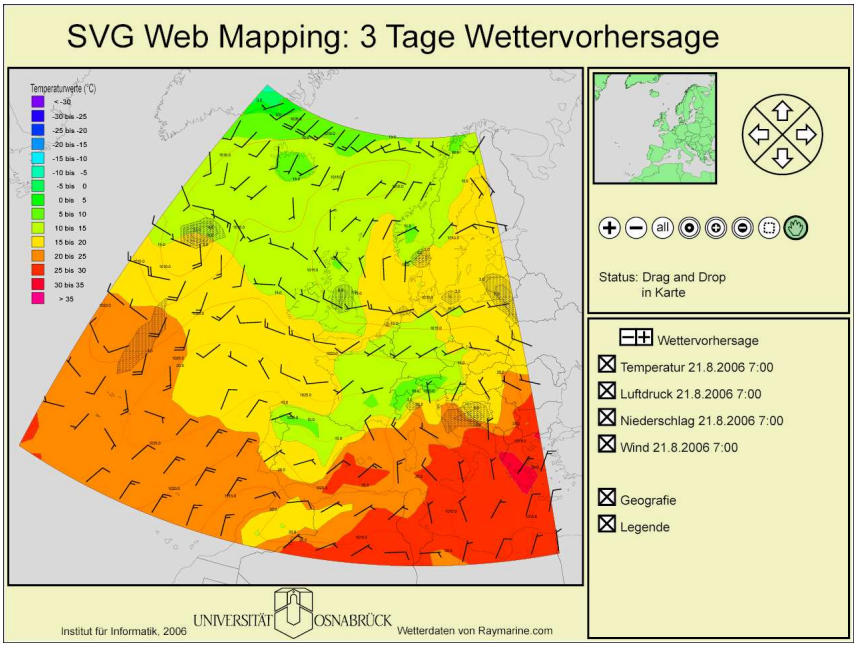


Abbildung A.4: Wettervorhersage für Europa

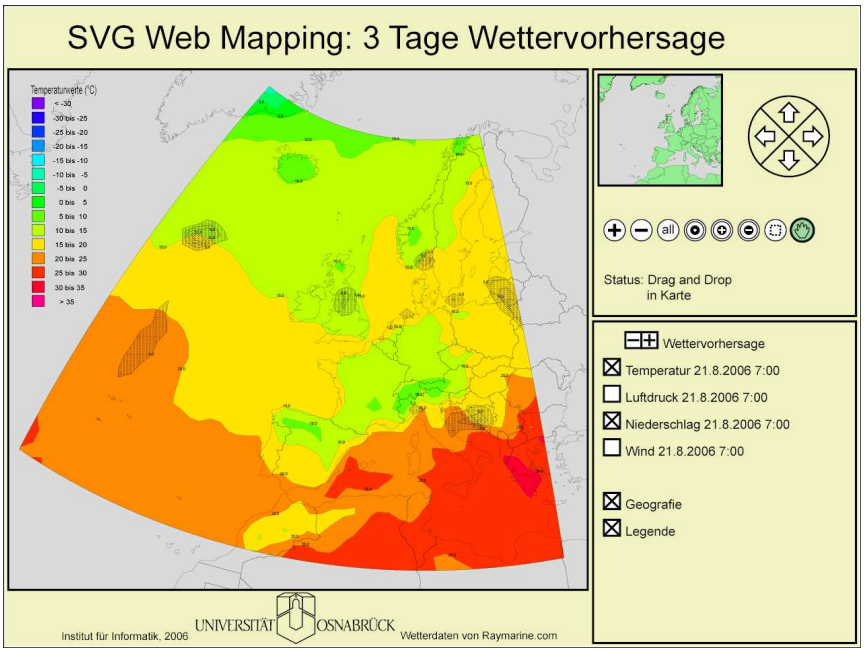


Abbildung A.5: Wettervorhersage mit Temperatur und Niederschlag

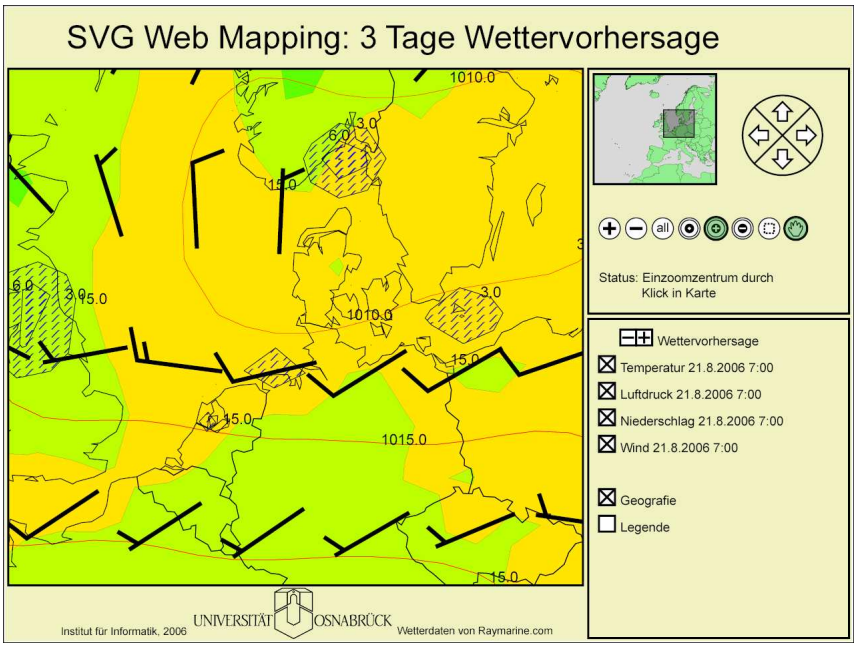


Abbildung A.6: Vergrößerte Darstellung

A.3 Wettervorhersage für verschiedene Gebiete der Welt

Das Unternehmen Raymarine stellt aufbereitete Wettervorhersagedaten [Raymarine] für verschiedene Regionen auf der Welt zur Verfügung, die ursprünglich von der National Oceanic and Atmospheric Administration [NOAA] berechnet wurden. Auf den folgenden Screenshots sind Wettervorhersagegebiete in unterschiedlichen Zoomstufen abgebildet.

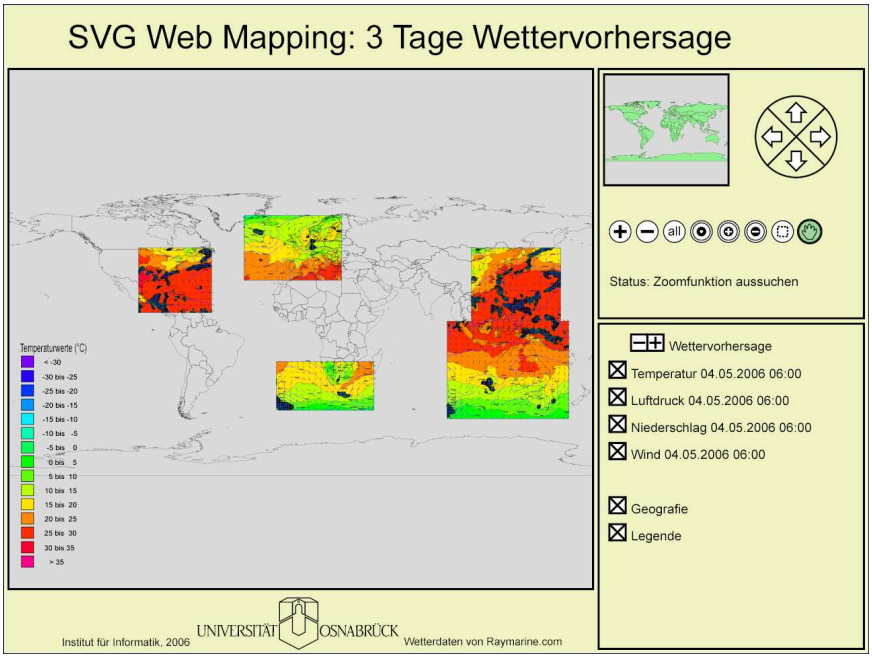


Abbildung A.7: Weltweite Wettervorhersagegebiete

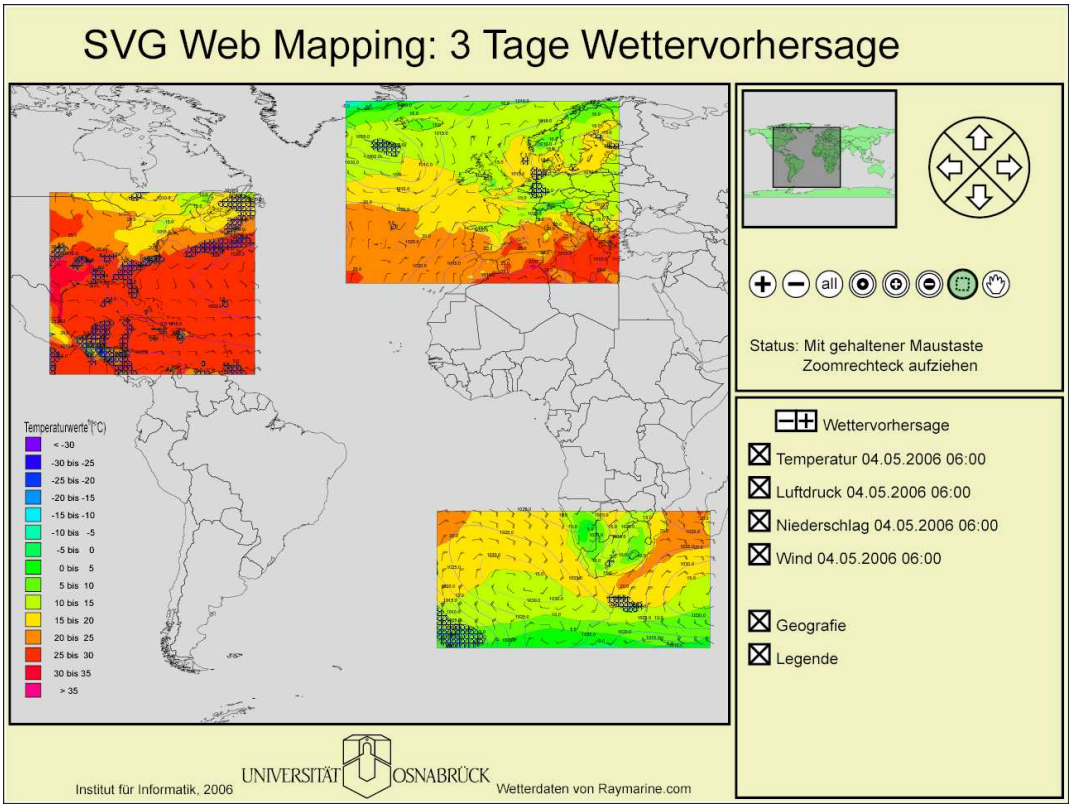


Abbildung A.8: Europa, USA und Südafrika

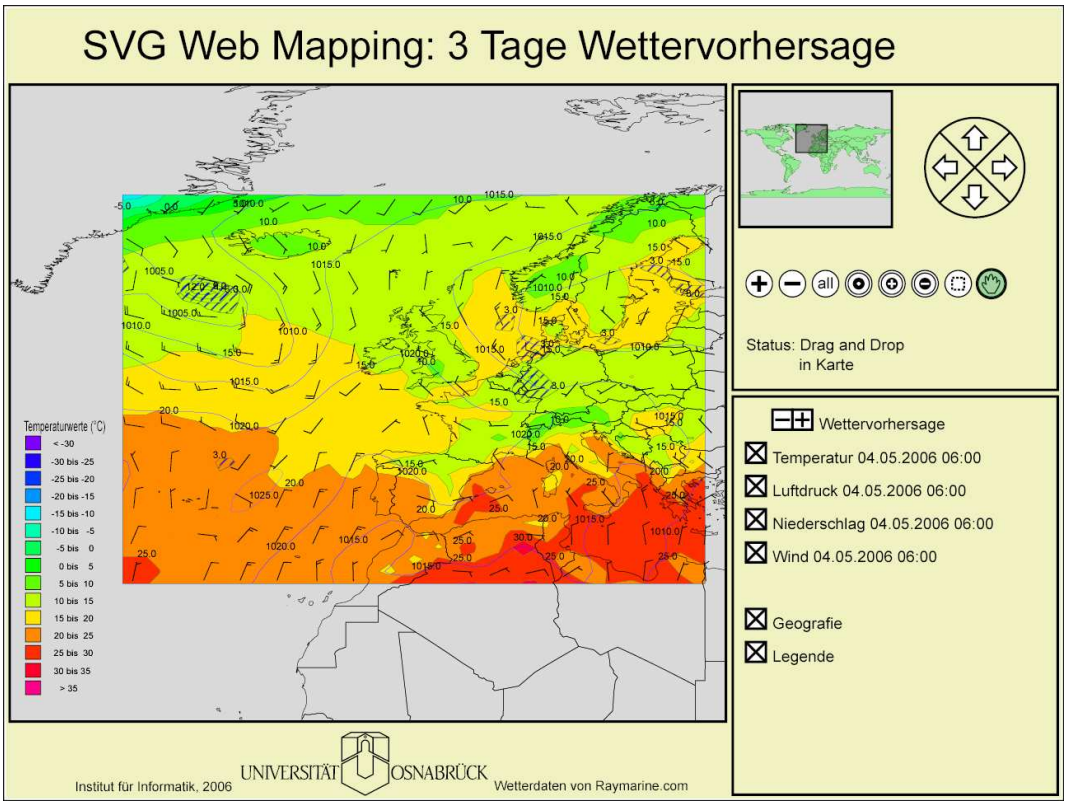


Abbildung A.9: Darstellung für Europa

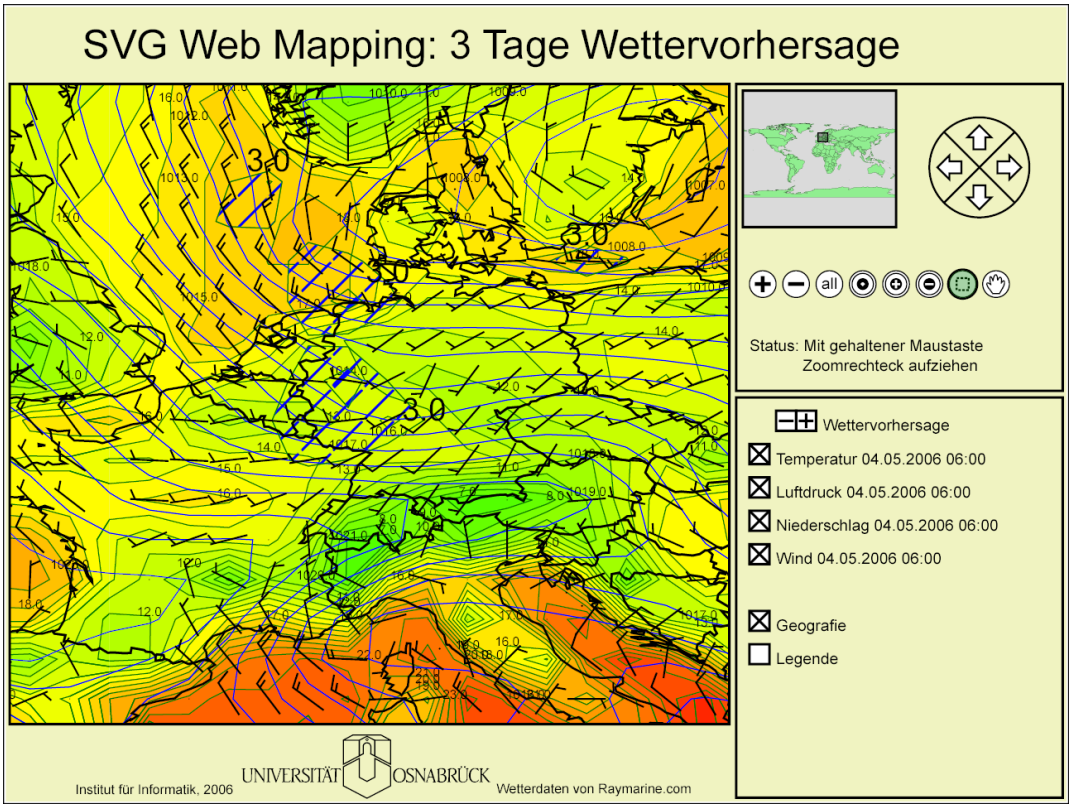


Abbildung A.10: Europa mit höher Aufgelösten Daten

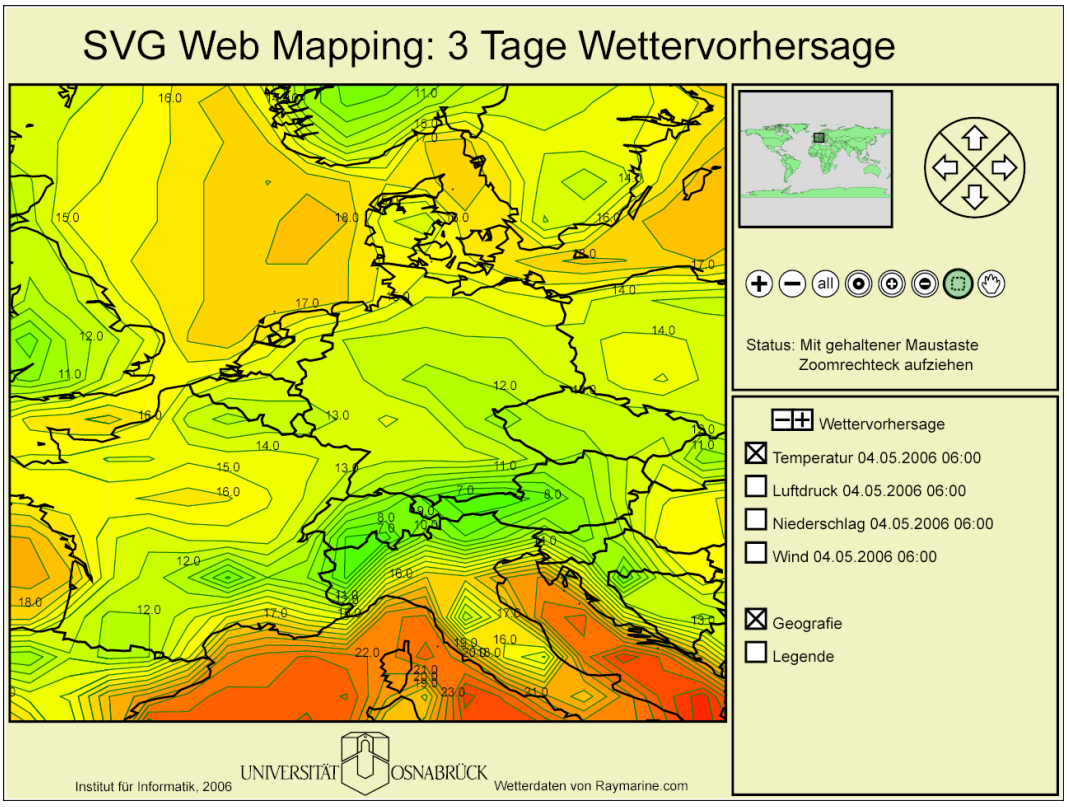


Abbildung A.11: Temperatur über Deutschland

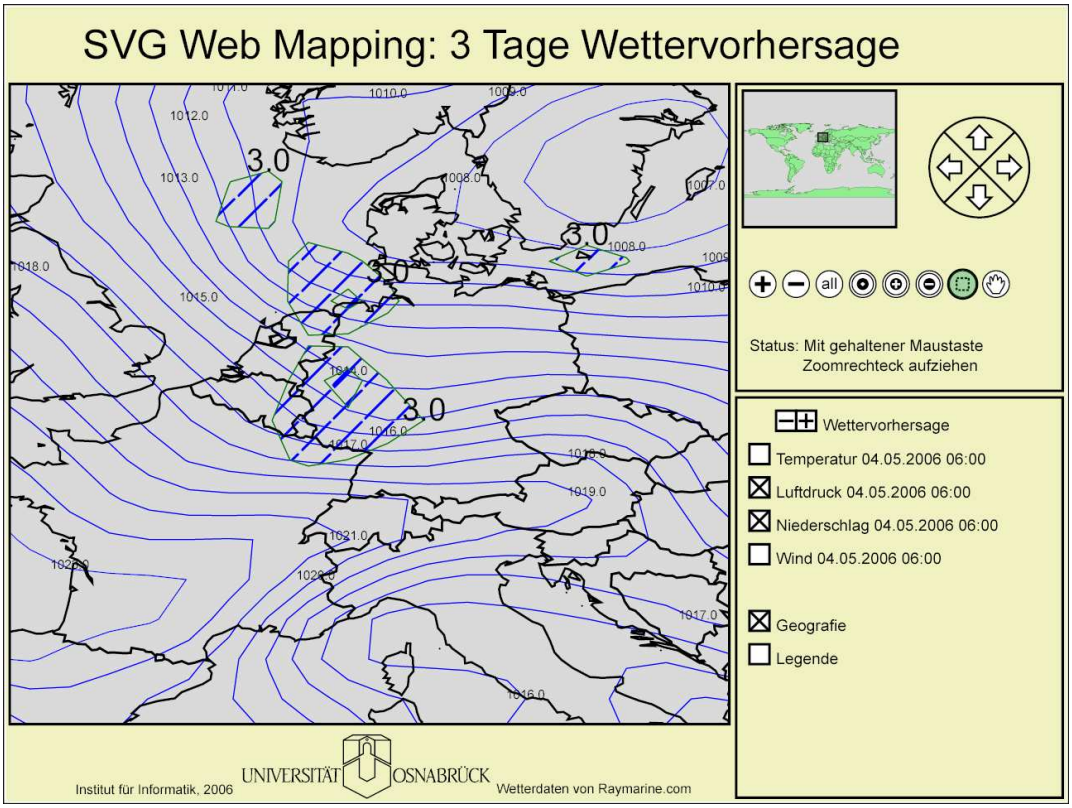


Abbildung A.12: Luftdruck und Niederschlag über Deutschland

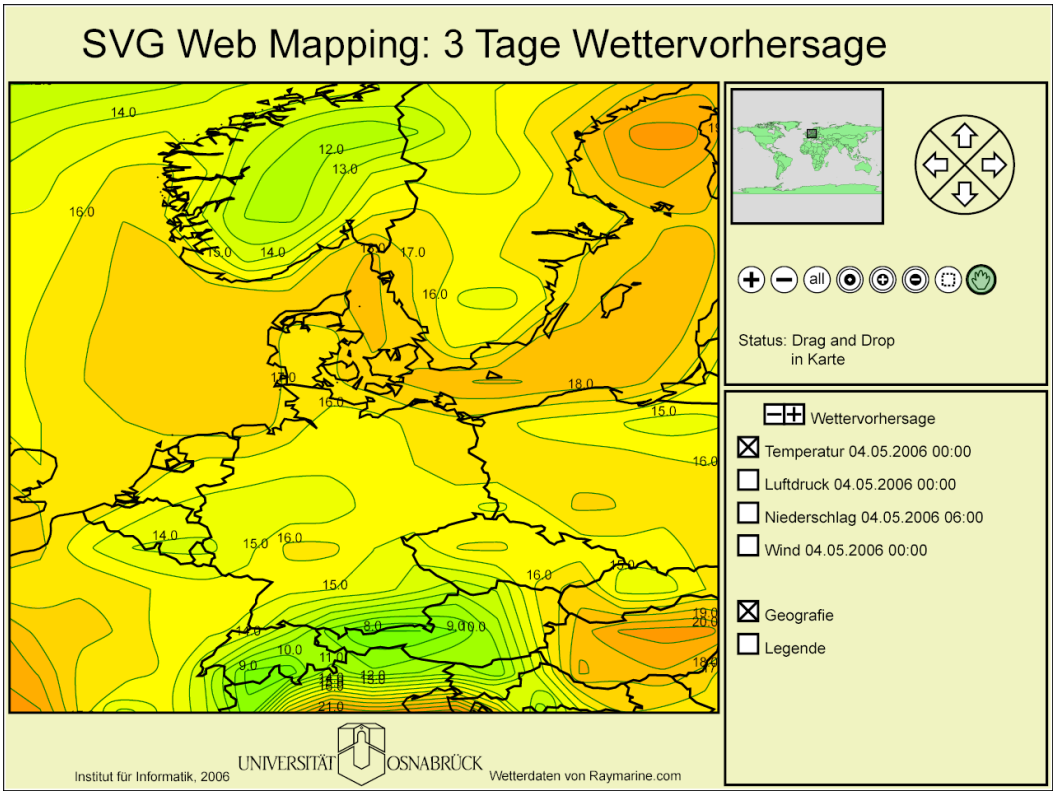


Abbildung A.13: Geglättete Temperatur-Isoflächen

B Erstellung einer Wettervorhersage

Um eine Wettervorhersage zu erstellen müssen zunächst die Daten vorliegen. Für die Geographie werden Shapefiles und für die Wettervorhersage werden GRIB Files benötigt. Um daraus eine SVG Web Mapping Applikation zu erstellen, muss eine Konfigurationsdatei erstellt werden, in der festgelegt wird, wie die Daten visualisiert werden sollen.

Nach der Erstellung der Konfigurationsdatei wird die Klasse `Creator` ausgeführt. Als Parameter muss der Pfad zur Konfigurationsdatei angegeben werden.

```
java -Xmx1024M -Xms1024M -Dfile.encoding=8859_1 \
       export.Creator config.xml
```

Dann werden die benötigten Dateien erstellt und in das im Konfigurationsfile angegebene Verzeichnis geschrieben. Wird das Verzeichnis auf einen PHP5 fähigen Webserver kopiert, kann die Wettervorhersage im Internet betrachtet werden.

Im Folgenden ist eine Konfigurationsdatei angegeben, in der eine Wettervorhersage für zwei Zeitpunkte im sechs Stunden Abstand erstellt wird. Die Wettervorhersage enthält die Parameter Temperatur, Luftdruck, Niederschlag und Wind.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<map xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:noNamespaceSchemaLocation="SVGWebMap.xsd">

<!-- Grundlegende Einstellungen -->
<configuration zoomAndPan="disable" outputfolder="vorhersage"
           strokecolor="black" symbolfillcolor="white"
           symbolhighlightcolor="green" textcolor="black"
           script="v1" converterclass="util.Lambert_Convert"
           title="3 Tage Wettervorhersage"
           legendFile="legends/legendeUntenRechts.svg"
               legendX="690" legendY="370"
               layout="layout/eurowetter.xml" simplify="5000">
<viewbox vbX="-38477"    vbY="-25461"
         vbWidth="53524" vbHeight="48111"/>
</configuration>

<!-- Uebersichtskarte erstellen -->
<overview simplify="1000">
  <selection sourcefile="input/cntry02">
    <layout xsi:type="Polygon" stroke="black" stroke-width="10"
            stroke-linecap="round" fill="lightgreen"/>
  </selection>
</overview>
```

```
<!-- xsi:type="multi" => Alles zusammenfassen, was bei einem
     Wechsel des Zeitpunktes gemeinsam geaendert werden soll -->
<layer xsi:type="multi" id="wetter" label="Wettervorhersage">

<!-- xsi:type="group" => fasst alle gleichartigen Daten
     zusammen, damit sie gleichzeitig sichtbar oder
     unsichtbar gemacht werden können -->
  <layer xsi:type="group" id="tmp" label="Temperatur">
    <layer xsi:type="single"  changeable="true"
           visibility="visible" id="tmp34"
           label="Temperatur 19.8.2006  7:00">
      <!-- erste Zoomstufe: Daten hinzufügen -->
      <zoomstep action="add">
        <!-- Angabe des Pfades zur Datenquelle und des
             Dateiformates -->
        <selection sourcefile="gribinput/Europe.grb"
                   fileformat="grib">
          <!-- Darstellungsart der Daten -->
          <layout xsi:type="Grib" recordnumber="34"
            isovaluelist="-30;-25;-20;-15;-10;-5;0;
                          5;10;15;20;25;30;35"
            isocolorlist="rgb(128,0,255);rgb(36,0,255);
                          rgb(0,56,255);rgb(0,148,255);
                          rgb(0,240,255);rgb(0,255,178);
                          rgb(0,255,87);rgb(5,255,0);
                          rgb(97,255,0);rgb(189,255,0);
                          rgb(255,229,0);rgb(255,138,0);
                          rgb(255,46,0);rgb(255,0,46);
                          rgb(255,0,138)"
            stroke="green" stroke-width="3" labeled="true"
            font-size="350"
            unitconverterclass="util.RaymarineKelvinConverter"/>
        </selection>
      </zoomstep>
      <!-- Zweite Zoomstufe:
           Alte Daten loeschen, neue hinzufügen -->
      <zoomstep action="replace">
        <selection sourcefile="gribinput/Europe.grb"
                   fileformat="grib">
          <layout xsi:type="Grib" recordnumber="34"
            isovaluelist="-30;-29;-28;-27;-26;-25;-24;
                          -23;-22;-21;-20;-19;-18;-17;-16;-15;
                          -14;-13;-12;-11;-10;-9;-8;-7;-6;-5;
                          -4;-3;-2;-1;0;1;2;3;4;5;6;7;8;9;10;
                          11;12;13;14;15;16;17;18;19;20;21;22;
                          23;24;25;26;27;28;29;30;31;32;33;34;35"
            isocolorlist="rgb(128,0,255);rgb(108,0,255);
                          rgb(89,0,255);rgb(69,0,255);rgb(50,0,255);
                          rgb(30,0,255);rgb(11,0,255);rgb(0,9,255);
```

```
                rgb(0,28,255);rgb(0,48,255);rgb(0,67,255);
                rgb(0,87,255);rgb(0,106,255);rgb(0,126,255);
                rgb(0,145,255);rgb(0,165,255);rgb(0,184,255);
                rgb(0,204,255);rgb(0,223,255);rgb(0,242,255);
                rgb(0,255,248);rgb(0,255,229);rgb(0,255,209);
                rgb(0,255,190);rgb(0,255,170);rgb(0,255,151);
                rgb(0,255,131);rgb(0,255,112);rgb(0,255,92);
                rgb(0,255,73);rgb(0,255,53);rgb(0,255,34);
                rgb(0,255,14);rgb(5,255,0);rgb(25,255,0);
                rgb(44,255,0);rgb(64,255,0);rgb(83,255,0);
                rgb(102,255,0);rgb(122,255,0);rgb(141,255,0);
                rgb(161,255,0);rgb(180,255,0);rgb(200,255,0);
                rgb(219,255,0);rgb(239,255,0);rgb(255,252,0);
                rgb(255,232,0);rgb(255,213,0);rgb(255,193,0);
                rgb(255,174,0);rgb(255,154,0);rgb(255,135,0);
                rgb(255,115,0);rgb(255,96,0);rgb(255,77,0);
                rgb(255,57,0);rgb(255,38,0);rgb(255,18,0);
                rgb(255,0,1);rgb(255,0,21);rgb(255,0,40);
                rgb(255,0,60);rgb(255,0,79);rgb(255,0,99);
                rgb(255,0,118);rgb(255,0,138)"
          stroke="green" stroke-width="3" labeled="true"
          font-size="200"
          unitconverterclass="util.RaymarineKelvinConverter"/>
      </selection>
    </zoomstep>
  </layer>
  <layer xsi:type="single"  changeable="true"
         visibility="visible" id="tmp35"
         label="Temperatur 19.8.2006  13:00">
    <zoomstep action="add">
      <selection sourcefile="gribinput/Europe.grb"
                 fileformat="grib">
        <layout xsi:type="Grib" recordnumber="35"
          isovaluelist="-30;-25;-20;-15;-10;-5;0;
                        5;10;15;20;25;30;35"
          isocolorlist="rgb(128,0,255);rgb(36,0,255);
                        rgb(0,56,255);rgb(0,148,255);
                        rgb(0,240,255);rgb(0,255,178);
                        rgb(0,255,87);rgb(5,255,0);
                        rgb(97,255,0);rgb(189,255,0);
                        rgb(255,229,0);rgb(255,138,0);
                        rgb(255,46,0);rgb(255,0,46);
                        rgb(255,0,138)"
          stroke="green" stroke-width="3" labeled="true"
          font-size="350"
          unitconverterclass="util.RaymarineKelvinConverter"/>
      </selection>
    </zoomstep>
    <zoomstep action="replace">
```

```
      <selection sourcefile="gribinput/Europe.grb"
                 fileformat="grib">
        <layout xsi:type="Grib" recordnumber="35"
          isovaluelist="-30;-29;-28;-27;-26;-25;-24;
                       -23;-22;-21;-20;-19;-18;-17;-16;-15;
                       -14;-13;-12;-11;-10;-9;-8;-7;-6;-5;
                       -4;-3;-2;-1;0;1;2;3;4;5;6;7;8;9;10;
                       11;12;13;14;15;16;17;18;19;20;21;22;
                       23;24;25;26;27;28;29;30;31;32;33;34;35"
          isocolorlist="rgb(128,0,255);rgb(108,0,255);
                 rgb(89,0,255);rgb(69,0,255);rgb(50,0,255);
                 rgb(30,0,255);rgb(11,0,255);rgb(0,9,255);
                 rgb(0,28,255);rgb(0,48,255);rgb(0,67,255);
                 rgb(0,87,255);rgb(0,106,255);rgb(0,126,255);
                 rgb(0,145,255);rgb(0,165,255);rgb(0,184,255);
                 rgb(0,204,255);rgb(0,223,255);rgb(0,242,255);
                 rgb(0,255,248);rgb(0,255,229);rgb(0,255,209);
                 rgb(0,255,190);rgb(0,255,170);rgb(0,255,151);
                 rgb(0,255,131);rgb(0,255,112);rgb(0,255,92);
                 rgb(0,255,73);rgb(0,255,53);rgb(0,255,34);
                 rgb(0,255,14);rgb(5,255,0);rgb(25,255,0);
                 rgb(44,255,0);rgb(64,255,0);rgb(83,255,0);
                 rgb(102,255,0);rgb(122,255,0);rgb(141,255,0);
                 rgb(161,255,0);rgb(180,255,0);rgb(200,255,0);
                 rgb(219,255,0);rgb(239,255,0);rgb(255,252,0);
                 rgb(255,232,0);rgb(255,213,0);rgb(255,193,0);
                 rgb(255,174,0);rgb(255,154,0);rgb(255,135,0);
                 rgb(255,115,0);rgb(255,96,0);rgb(255,77,0);
                 rgb(255,57,0);rgb(255,38,0);rgb(255,18,0);
                 rgb(255,0,1);rgb(255,0,21);rgb(255,0,40);
                 rgb(255,0,60);rgb(255,0,79);rgb(255,0,99);
                 rgb(255,0,118);rgb(255,0,138)"
          stroke="green" stroke-width="3" labeled="true"
          font-size="200"
          unitconverterclass="util.RaymarineKelvinConverter"/>
      </selection>
    </zoomstep>
  </layer>
</layer>

<layer xsi:type="group" id="prmsl" label="Luftdruck">
  <layer xsi:type="single"  changeable="true"
         visibility="visible" id="prmsl1"
         label="Luftdruck 19.8.2006  7:00">
    <zoomstep action="add">
      <selection sourcefile="gribinput/Europe.grb"
                 fileformat="grib">
        <layout xsi:type="Grib" recordnumber="1"
            isovaluelist="960;965;970;975;980;985;990;
```

```
                        995;1000;1005;1010;1015;
                        1020;1025;1030;1035;1040;"
          stroke="red" stroke-width="10"
          labeled="true" font-size="350"
          unitconverterclass="util.RaymarinePRMSLConverter"
          isotype="line"/>
    </selection>
  </zoomstep>
  <zoomstep action="replace">
    <selection sourcefile="gribinput/Europe.grb"
                fileformat="grib">
      <layout xsi:type="Grib" recordnumber="1"
           isovaluelist="960;961;962;963;964;965;966;967;
                         968;969;970;971;972;973;974;975;
                         976;977;978;979;980;981;982;983;
                         984;985;986;987;988;989;990;991;
                         992;993;994;995;996;997;998;999;
                         1000;1001;1002;1003;1004;1005;
                         1006;1007;1008;1009;1010;1011;
                         1012;1013;1014;1015;1016;1017;
                         1018;1019;1020;1021;1022;1023;
                         1024;1025;1026;1027;1028;1029;
                         1030;1031;1032;1033;1034;1035;
                         1036;1037;1038;1039;1040"
          stroke="red" stroke-width="10" labeled="true"
          unitconverterclass="util.RaymarinePRMSLConverter"
          isotype="line" font-size="200"/>
    </selection>
  </zoomstep>
</layer>
<layer xsi:type="single"  changeable="true"
       visibility="visible" id="prmsl2"
       label="Luftdruck 19.8.2006  13:00">
  <zoomstep action="add">
    <selection sourcefile="gribinput/Europe.grb"
               fileformat="grib">
      <layout xsi:type="Grib" recordnumber="2"
          isovaluelist="960;965;970;975;980;985;990;
                        995;1000;1005;1010;1015;
                        1020;1025;1030;1035;1040;"
          stroke="red" stroke-width="10"
          labeled="true" font-size="350"
          unitconverterclass="util.RaymarinePRMSLConverter"
          isotype="line"/>
    </selection>
  </zoomstep>
  <zoomstep action="replace">
    <selection sourcefile="gribinput/Europe.grb"
                fileformat="grib">
```

```
        <layout xsi:type="Grib" recordnumber="2"
          isovaluelist="-30;-29;-28;-27;-26;-25;-24;
                        -23;-22;-21;-20;-19;-18;-17;-16;-15;
                        -14;-13;-12;-11;-10;-9;-8;-7;-6;-5;
                        -4;-3;-2;-1;0;1;2;3;4;5;6;7;8;9;10;
                        11;12;13;14;15;16;17;18;19;20;21;22;
                        23;24;25;26;27;28;29;30;31;32;33;34;35"
          isocolorlist="rgb(128,0,255);rgb(108,0,255);
                 rgb(89,0,255);rgb(69,0,255);rgb(50,0,255);
                 rgb(30,0,255);rgb(11,0,255);rgb(0,9,255);
                 rgb(0,28,255);rgb(0,48,255);rgb(0,67,255);
                 rgb(0,87,255);rgb(0,106,255);rgb(0,126,255);
                 rgb(0,145,255);rgb(0,165,255);rgb(0,184,255);
                 rgb(0,204,255);rgb(0,223,255);rgb(0,242,255);
                 rgb(0,255,248);rgb(0,255,229);rgb(0,255,209);
                 rgb(0,255,190);rgb(0,255,170);rgb(0,255,151);
                 rgb(0,255,131);rgb(0,255,112);rgb(0,255,92);
                 rgb(0,255,73);rgb(0,255,53);rgb(0,255,34);
                 rgb(0,255,14);rgb(5,255,0);rgb(25,255,0);
                 rgb(44,255,0);rgb(64,255,0);rgb(83,255,0);
                 rgb(102,255,0);rgb(122,255,0);rgb(141,255,0);
                 rgb(161,255,0);rgb(180,255,0);rgb(200,255,0);
                 rgb(219,255,0);rgb(239,255,0);rgb(255,252,0);
                 rgb(255,232,0);rgb(255,213,0);rgb(255,193,0);
                 rgb(255,174,0);rgb(255,154,0);rgb(255,135,0);
                 rgb(255,115,0);rgb(255,96,0);rgb(255,77,0);
                 rgb(255,57,0);rgb(255,38,0);rgb(255,18,0);
                 rgb(255,0,1);rgb(255,0,21);rgb(255,0,40);
                 rgb(255,0,60);rgb(255,0,79);rgb(255,0,99);
                 rgb(255,0,118);rgb(255,0,138)"
          stroke="green" stroke-width="3" labeled="true"
          font-size="200"
          unitconverterclass="util.RaymarineKelvinConverter"/>
       </selection>
     </zoomstep>
   </layer>
 </layer>
 <layer xsi:type="group" id="rain" label="Niederschlag">
    <layer xsi:type="single"  changeable="true"
           visibility="visible" id="regen0"
           label="Niederschlag: keine Daten">
      <zoomstep/>
      <zoomstep/>
    </layer>
    <layer xsi:type="single"  changeable="true"
           visibility="visible" id="rain45"
           label="Niederschlag 19.8.2006  13:00">
      <zoomstep action="add">
        <selection sourcefile="gribinput/Europe.grb"
```

```
                fileformat="grib">
        <layout xsi:type="Grib" recordnumber="45"
            isovaluelist="3;6;9;12;15;18;21;24;27"
            isocolorlist="url(#muster0);url(#muster10);
                          url(#muster20);url(#muster30);
                          url(#muster40);url(#muster50);
                          url(#muster60);url(#muster70);
                          url(#muster80);url(#muster90);
                          url(#muster100)"
            stroke="black" stroke-width="10"
            labeled="true" font-size="350"
            unitconverterclass="util.RaymarineRAINConverter"
            isotype="area"/>
      </selection>
    </zoomstep>
    <zoomstep/>
  </layer>
</layer>
<layer xsi:type="group" id="wind" label="Wind">
  <layer xsi:type="single"  changeable="true"
         visibility="visible" id="wind12"
         label="Wind 19.8.2006  7:00">
    <zoomstep action="add">
      <selection sourcefile="gribinput/Europe.grb"
                 fileformat="grib">
        <layout xsi:type="Grib" recordnumber="12"
            generalizeX="3"
            generalizeY="3"
            scale="50"
            unitconverterclass="util.RaymarineWINDConverter"
            isotype="wind"/>
      </selection>
    </zoomstep>
    <zoomstep action="replace">
      <selection sourcefile="gribinput/Europe.grb"
                 fileformat="grib">
        <layout xsi:type="Grib" recordnumber="12"
            scale="15"
            unitconverterclass="util.RaymarineWINDConverter"
            isotype="wind"/>
      </selection>
    </zoomstep>
  </layer>
  <layer xsi:type="single"  changeable="true"
         visibility="visible" id="wind14"
         label="Wind 19.8.2006  13:00">
    <zoomstep action="add">
      <selection sourcefile="gribinput/Europe.grb"
                 fileformat="grib">
```

```
          <layout xsi:type="Grib" recordnumber="14"
              generalizeX="3"
              generalizeY="3"
              scale="50"
              unitconverterclass="util.RaymarineWINDConverter"
              isotype="wind"/>
        </selection>
      </zoomstep>
      <zoomstep action="replace">
        <selection sourcefile="gribinput/Europe.grb"
                   fileformat="grib">
          <layout xsi:type="Grib" recordnumber="14"
              scale="15"
              unitconverterclass="util.RaymarineWINDConverter"
              isotype="wind"/>
        </selection>
      </zoomstep>
    </layer>
  </layer>
 </layer>
 <layer xsi:type="single" changeable="true"
        visibility="visible" id="country" label="Geographie">
  <zoomstep action="add">
    <selection sourcefile="input/cntry02">
      <layout xsi:type="Polygon" stroke="black"
              stroke-width="20" stroke-linecap="round"
              fill="none"/>
    </selection>
  </zoomstep>
  <zoomstep/>
</layer>
<zoom>
  <zoomstep stepfactor="0.1" steps="3" zoomfactor="0.5"/>
  <zoomstep stepfactor="0.1" steps="3" zoomfactor="0.5"/>
</zoom>
</map>
```

C UML Diagramme

In dieser Arbeit wurden 83 Java-Klassen mit insgesamt ca. 25.900 Zeilen Code implementiert. Die Teile zur Kantenglättung stammen von Karl kleine Kruse [kleine Kruse]. Die Java Klassen zur Darstellung der Geographie wurden von Dorothee Langfeld implementiert [Langfeld]. Die Klassen sind in sechs Packages eingeteilt. Zusätzlich wurden 2.636 Zeilen ECMAScript und 1080 Zeilen PHP Code geschrieben. Um einen Überblick der Java-Klassen zu erhalten, werden im Folgenden einige vereinfachte UML Klassendiagramme vorgestellt. Um die Übersichtlichkeit zu wahren, werden nicht alle Methoden dargestellt. Die Instanzvariablen der einzelnen Java-Klassen sind in den Klassendiagrammen nicht enthalten.

C.1 Klassen zur Verwaltung eines GRIB Files

Ein GRIB File kann mit Klassen des Packages `grib` ausgelesen und verwaltet werden. Die dazu benötigten Klassen sind in der Abbildung C.1 als vereinfachtes UML Diagramm dargestellt.

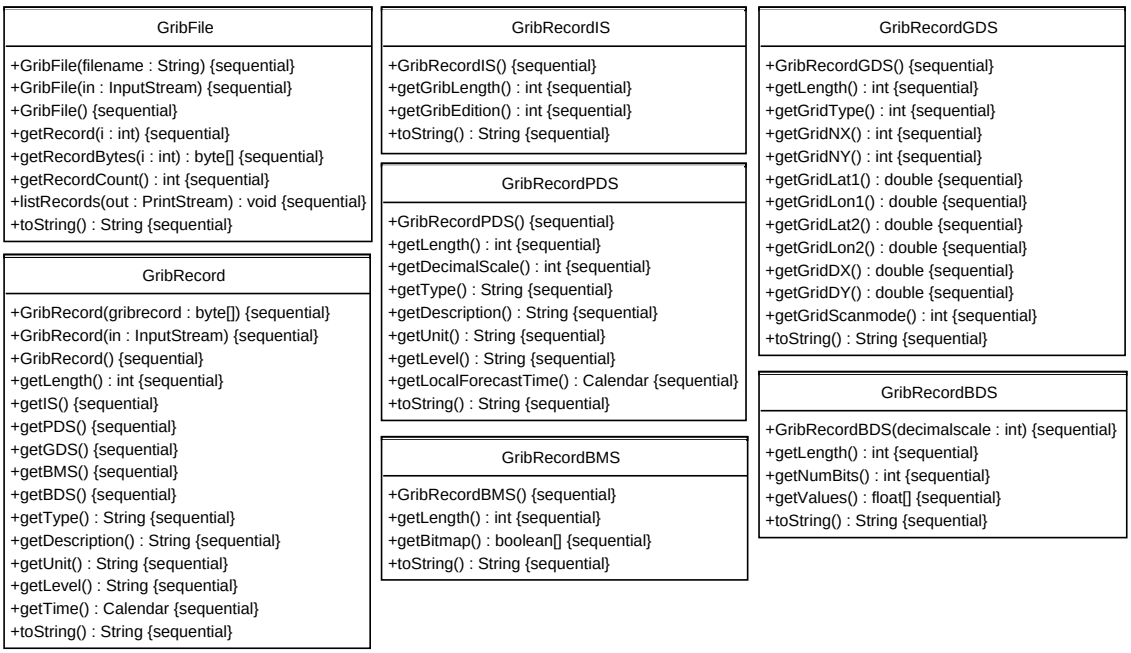


Abbildung C.1: Klassendiagramm der GRIB Verwaltung

Um das Binärformat eines GRIB Files auslesen zu können, wurden Hilfsklassen implementiert (Abbildung C.2), mit denen eine Datei bitweise ausgelesen werden kann und die kodierten Werte leicht umgewandelt werden können. Für die Dekodierung einzelner Parameter wurde eine Klasse `GribTables` implementiert, in der die Parameter mit textuellen Informationen verknüpft sind.

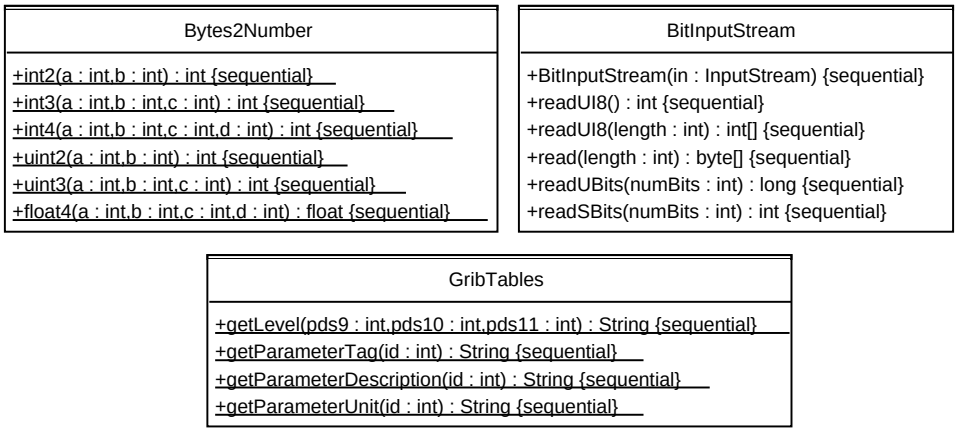


Abbildung C.2: Klassendiagramm der Hilfsklassen zum Auslesen eines GRIB Files

C.2 Klassen zum Vektorisieren eines Rasters

Um ein regelmäßiges Gitter mit Rasterwerten anhand von Isowerten zu vektorisieren, wurden die Klassen im Package `iso` implementiert. Die Klasse `VectorizerImpl` stellt den zentralen Kern der Implementation dar (Abbildung C.3).

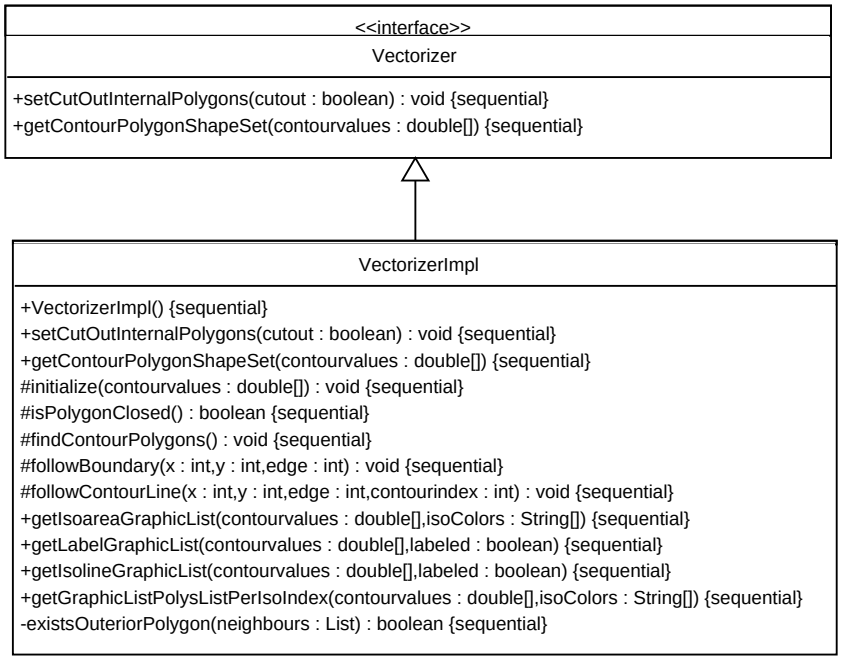


Abbildung C.3: Methoden der Klasse `VectorizerImpl`

Einen Überblick der weiteren Klassen aus dem Package `iso` gibt die Abbildung C.4.

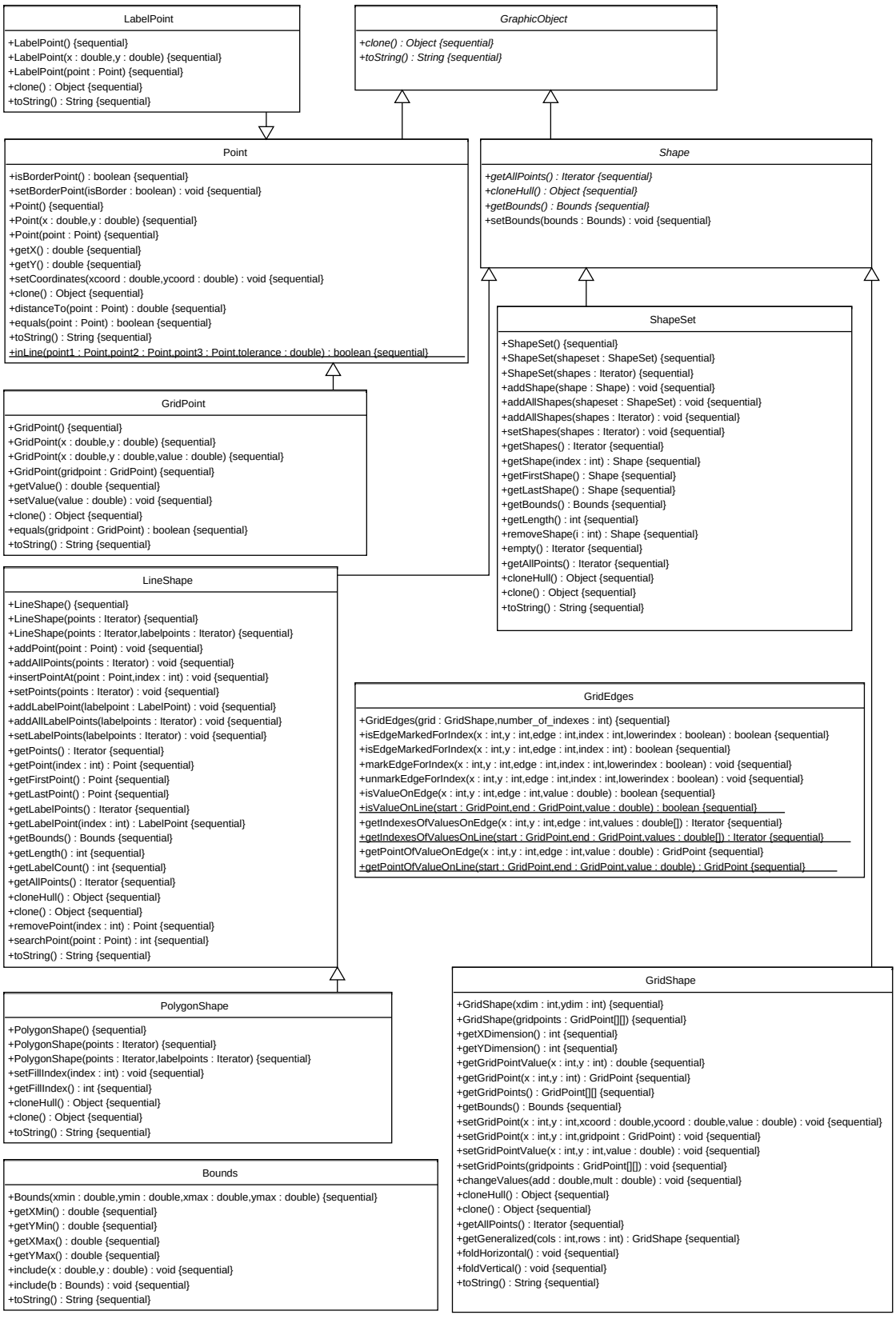


Abbildung C.4: Klassendiagramm des Packages `iso`

C.3 Klassen für Projektionen und Einheitenkonvertierungen

Für die Projektion geographischer Daten und für die Umwandlung der GRIB Daten in andere Maßeinheiten, wurden verschiedene Klassen implementiert. Eine Auswahl ist in der Abbildung C.5 dargestellt.

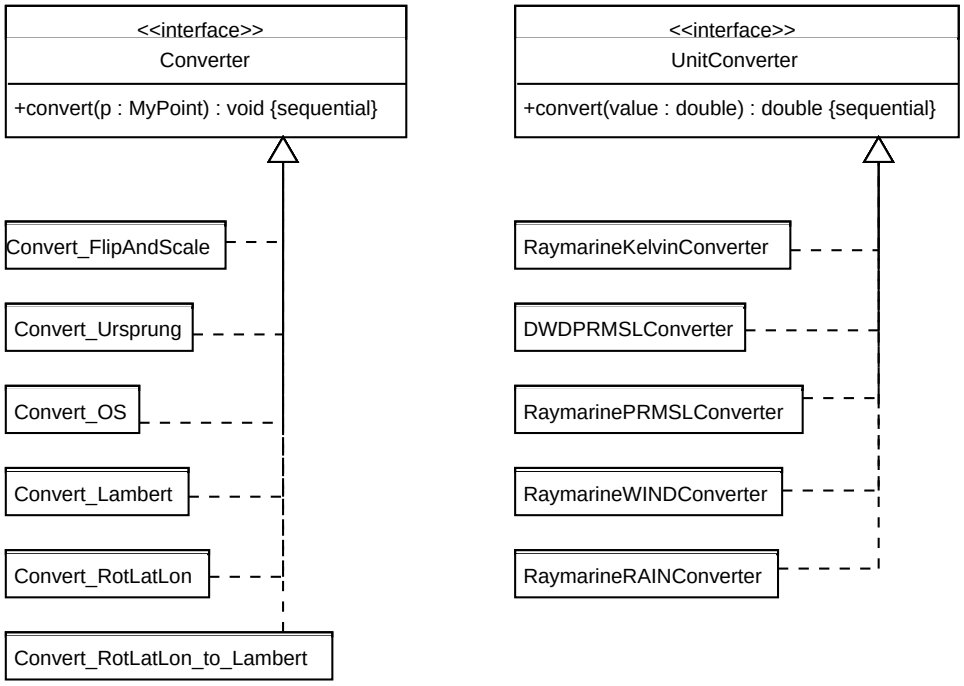


Abbildung C.5: Klassendiagramm einiger Converter Klassen

C.4 Klassen zur Verwaltung geometrischer Elemente

Um die geometrischen Elemente zu verwalten und um sie in SVG Elemente umzuwandeln, wurden verschiedenen Klassen implementiert (Abbildung C.6). Einige Methoden werden zugunsten der Übersichtlichkeit nicht dargestellt.

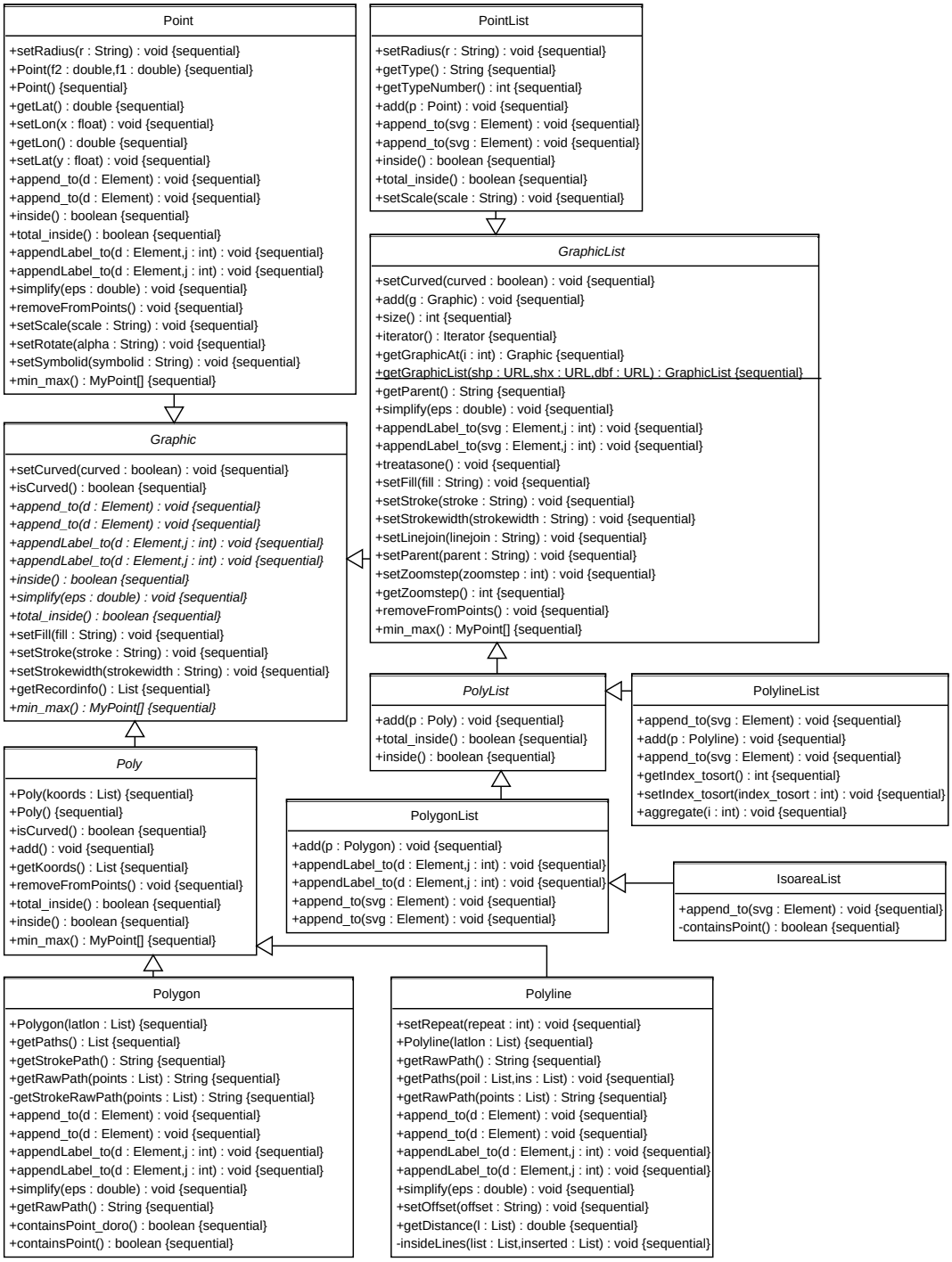


Abbildung C.6: Klassendiagramm der Shape Klassen

C.5 Klassen für die Kantenglättung

Es wurden Java-Klassen implementiert, mit denen ein Linienzug geglättet werden kann. Die Klassen sind in der Abbildung C.7 dargestellt.

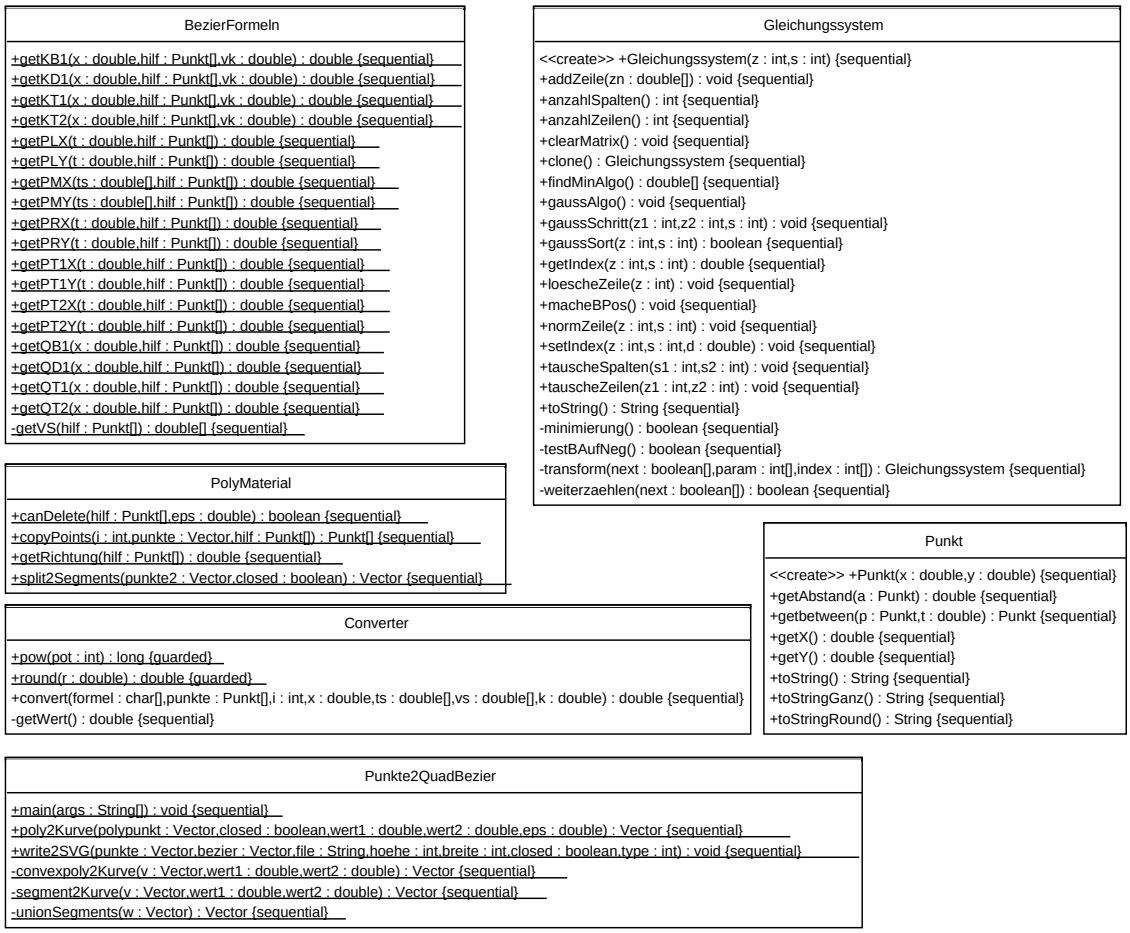


Abbildung C.7: Klassendiagramm der Klassen zur Kantenglättung

C.6 Klassen zur Verwaltung des Konfigurationsfiles

Damit das Konfigurationsfile in der Java Anwendung leicht zugänglich ist, liegt es in einem DOM im Speicher vor. Die Klassen mit denen auf das Konfigurationsfile zugegriffen werden kann, sind in der Abbildung C.8 dargestellt.

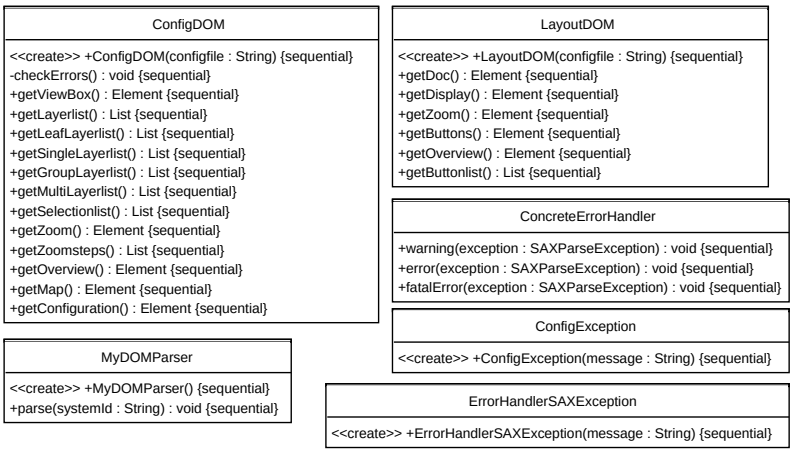


Abbildung C.8: Klassendiagramm für die Verwaltung des Konfigurationsfiles

C.7 Klassen zur DOM Erstellung und das Schreiben der Dateien

In einem Objekt der Klasse `MySVGDocument` wird der DOM der SVG Anwendung erzeugt und in eine Datei geschrieben. Die in SVG umgesetzten georeferenzierten Daten werden mit einem `DOMFileWriter`-Objekt geschrieben. Die XML Dateien mit den Informationen bei welcher Stufe Daten geladen oder gelöscht werden müssen, werden mit Objekten der Klasse `ActionFileWriter` in Dateien geschrieben (Abbildung C.9).

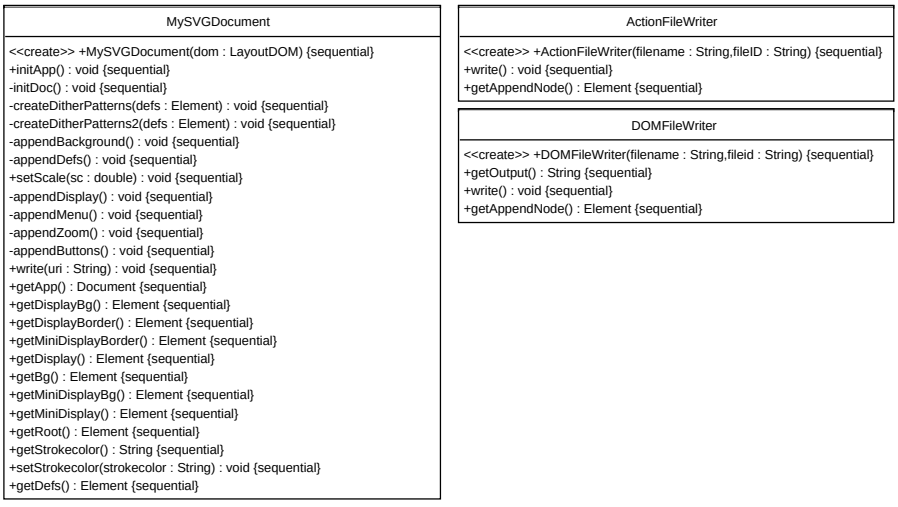


Abbildung C.9: Klassendiagramm der Klassen zur DOM Erzeugung und SVG Export

C.8 Main-Klasse zur Erstellung der SVG Anwendung

In der Klasse `Creator` ist die Main-Klasse der Java Anwendung. Die einzelnen Module werden hier zusammengeführt. Einen Überblick über die Methoden der Klasse gibt die Abbildung C.10.

Abbildung C.10: Klassendiagramm der Main-Klasse

D XML-Schema für die Konfigurationsdatei

In der Konfigurationsdatei werden alle Einstellungen für die Visualisierung georeferenzierter Daten und zur Steuerung der SVG Web Mapping Applikation vorgenommen. Zugunsten des Layouts mussten teilweise Zeilenumbrüche eingefügt werden, die in der Originaldatei nicht enthalten sein dürfen.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="map">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="configuration" type="configType"
          minOccurs="1" maxOccurs="1"/>
        <xs:element name="overview" type="overviewType"
          minOccurs="1" maxOccurs="1"/>
        <xs:element name="layer" type="layerType"
          minOccurs="1" maxOccurs="unbounded"/>
        <xs:element name="zoom" type="zoomType"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:complexType name="configType" mixed="false">
    <xs:sequence>
      <xs:element name="viewbox" type="viewboxType"
        minOccurs="0" maxOccurs="1"/>
    </xs:sequence>
    <xs:attribute name="converterclass" type="xs:string"
                  default="util.Convert"/>
    <xs:attribute name="converterparams" type="xs:anyURI"/>
    <xs:attribute name="delimiter" type="xs:string"
                  default=";"/>
    <xs:attribute name="title" type="xs:string"
                  default="SVG Web Mapping"/>
    <xs:attribute name="zoomAndPan" type="ableType"
                  default="disable"/>
    <xs:attribute name="outputfolder" type="xs:anyURI"
                  use="required"/>
    <xs:attribute name="textcolor" type="colorType"
                  default="black"/>
    <xs:attribute name="strokecolor" type="colorType"
                  default="black"/>
    <xs:attribute name="symbolfillcolor" type="colorType"
                  default="white"/>
    <xs:attribute name="symbolhighlightcolor" type="colorType"
                  default="white"/>
    <xs:attribute name="simplify" type="linewidthType"
                  default="0"/>
    <xs:attribute name="text-simplify" type="linewidthType"
```

```
                default="0"/>
  <xs:attribute name="scale" type="linewidthType" />
  <xs:attribute name="script" type="scriptType"
                default="v1"/>
  <xs:attribute name="legendFile" type="xs:anyURI"/>
  <xs:attribute name="legendX" type="xs:double" default="0"/>
  <xs:attribute name="legendY" type="xs:double" default="0"/>
  <xs:attribute name="layout" type="xs:anyURI"
                default="layout/default.xml"/>
</xs:complexType>
<xs:complexType name="viewboxType" mixed="false">
  <xs:attribute name="vbX" type="xs:double" use="required"/>
  <xs:attribute name="vbY" type="xs:double" use="required"/>
  <xs:attribute name="vbWidth" type="xs:double"
                use="required"/>
  <xs:attribute name="vbHeight" type="xs:double"
                use="required"/>
</xs:complexType>
<xs:complexType name="layoutType" abstract="true"
                mixed="false">
  <xs:attribute name="stroke" type="colorType"
                default="black"/>
  <xs:attribute name="stroke-width" type="linewidthType"
                default="1.0" />
</xs:complexType>
<xs:complexType name="Grib">
  <xs:complexContent>
    <xs:extension base="layoutType">
      <xs:attribute name="generalizeX" type="countType"/>
      <xs:attribute name="generalizeY" type="countType"/>
      <xs:attribute name="isotype" default="area"
                    type="isoType"/>
      <xs:attribute name="isovaluelist" type="xs:string"/>
      <xs:attribute name="max-value" type="countType"/>
      <xs:attribute name="isocolorlist" type="xs:string"/>
      <xs:attribute name="delimiter" type="xs:string"
                    default=";"/>
      <xs:attribute name="stroke-linecap" type="linecapType"
                    default="butt"/>
      <xs:attribute name="stroke-linejoin" type="linejoinType"
                    default="miter"/>
      <xs:attribute name="recordnumber" type="countType"
                    default="1"/>
      <xs:attribute name="labeled" type="xs:boolean"
                    default="false"/>
      <xs:attribute name="font-size" type="countType"
                    default="20"/>
      <xs:attribute name="scale" type="countType"
                    default="1"/>
```

```
      <xs:attribute name="unitconverterclass"
                    type="xs:string"/>
      <xs:attribute name="curved" type="curveType"
                    default="false"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="Polygon">
  <xs:complexContent>
    <xs:extension base="layoutType">
      <xs:attribute name="fill" type="colorType"
                    default="none"/>
      <xs:attribute name="stroke-linecap" type="linecapType"
                    default="butt"/>
      <xs:attribute name="stroke-linejoin" type="linejoinType"
                    default="miter"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="Polyline">
  <xs:complexContent>
    <xs:extension base="Polygon">
      <xs:attribute name="aggregate" type="indexType"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="Point">
  <xs:complexContent>
    <xs:extension base="layoutType">
      <xs:attribute name="fill" type="colorType"
                    default="black"/>
      <xs:attribute name="radius" type="linewidthType"
                    default="1"/>
      <xs:attribute name="scale" type="linewidthType"
                    default="1"/>
      <xs:attribute name="symbolid" type="xs:string"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="Text">
  <xs:complexContent>
    <xs:extension base="layoutType">
      <xs:attribute name="index" type="indexType"
                    use="required"/>
      <xs:attribute name="font" type="xs:string"/>
      <xs:attribute name="font-fill" type="colorType"
                    default="black"/>
      <xs:attribute name="font-size" type="linewidthType"
                    default="1"/>
```

```
      <xs:attribute name="text-anchor" type="anchorType"
        default="start"/>
      <xs:attribute name="font-style" type="styleType"
        default="normal"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="TextPath">
  <xs:complexContent >
    <xs:extension base="Text">
      <xs:attribute name="repeat" type="countType"
                    default="1"/>
      <xs:attribute name="spacing" type="indexType"
                    default="1"/>
      <xs:attribute name="offset" type="offsetType"
                    default="0"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="paramType" mixed="false">
  <xs:attribute name="index" type="indexType"
                use="required"/>
  <xs:attribute name="include" type="includeType"
                use="required"/>
  <xs:attribute name="valuelist" type="xs:string"
                use="required"/>
  <xs:attribute name="delimiter" type="xs:string"
                default=";"/>
</xs:complexType>
<xs:complexType name="zoomstepType" mixed="false">
  <xs:attribute name="zoomfactor" type="faktorType"
                default="0.5"/>
  <xs:attribute name="stepfactor" type="faktorType"
                default="0.1"/>
  <xs:attribute name="steps" type="countType" default="1"/>
</xs:complexType>
<xs:complexType name="selectType" mixed="false">
  <xs:sequence>
    <xs:element name="param" type="paramType" minOccurs="0"
      maxOccurs="unbounded"/>
    <xs:element name="layout" type="layoutType" minOccurs="1"
      maxOccurs="1"/>
  </xs:sequence>
  <xs:attribute name="sourcefile" type="xs:anyURI"
                use="required"/>
  <xs:attribute name="converterclass" type="xs:anyURI"/>
  <xs:attribute name="converterparams" type="xs:string"/>
  <xs:attribute name="delimiter" type="xs:string"
                default=";"/>
```

```
  <xs:attribute name="fileformat" type="fileType"
                default="shp"/>
</xs:complexType>
<xs:complexType name="layerzoomType" mixed="false">
  <xs:sequence>
    <xs:element name="selection" type="selectType"
                minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="action" type="actionType"
                default="add"/>
</xs:complexType>
<xs:complexType name="layerType" abstract="true"
                mixed="false">
  <xs:attribute name="id" type="xs:ID" use="required"/>
  <xs:attribute name="label" type="xs:string"/>
</xs:complexType>
<xs:complexType name="single">
  <xs:complexContent>
    <xs:extension base="layerType">
      <xs:sequence>
        <xs:element name="zoomstep" type="layerzoomType"
                    minOccurs="1" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="changeable" type="xs:boolean"
                    default="true"/>
      <xs:attribute name="visibility" type="visibleType"
                    default="visible"/>
    </xs:extension>
   </xs:complexContent>
</xs:complexType>
<xs:complexType name="group">
  <xs:complexContent>
    <xs:extension base="layerType">
      <xs:sequence>
        <xs:element name="layer" type="single" minOccurs="1"
                    maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="changeable" type="xs:boolean"
                    default="true"/>
      <xs:attribute name="visibility" type="visibleType"
                    default="visible"/>
    </xs:extension>
   </xs:complexContent>
</xs:complexType>
<xs:complexType name="multi">
  <xs:complexContent>
    <xs:extension base="layerType">
      <xs:sequence>
        <xs:element name="layer" type="group" minOccurs="1"
```

```
                    maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:extension>
   </xs:complexContent>
</xs:complexType>
<xs:complexType name="zoomType" mixed="false" >
  <xs:sequence>
    <xs:element name="zoomstep" type="zoomstepType"
                minOccurs="1" maxOccurs="unbounded">
    </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="overviewType" mixed="false" >
  <xs:sequence>
    <xs:element name="selection" type="selectType"
                minOccurs="1" maxOccurs="unbounded">
    </xs:element>
  </xs:sequence>
  <xs:attribute name="simplify" type="linewidthType"
                default="0"/>
</xs:complexType>

<xs:simpleType name="linewidthType">
  <xs:restriction base="xs:double">
    <xs:minInclusive value="0"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="faktorType">
  <xs:restriction base="xs:double">
    <xs:minExclusive value="0"/>
    <xs:maxExclusive value="1"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="indexType">
  <xs:restriction base="xs:integer">
    <xs:minInclusive value="0"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="countType">
  <xs:restriction base="xs:integer">
    <xs:minInclusive value="1"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="offsetType">
  <xs:restriction base="xs:double">
    <xs:minInclusive value="0"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="linecapType">
```

```
    <xs:restriction base="xs:string">
      <xs:enumeration value="round"/>
      <xs:enumeration value="butt"/>
      <xs:enumeration value="square"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="linejoinType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="round"/>
      <xs:enumeration value="bevil"/>
      <xs:enumeration value="miter"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="colorType">
    <xs:restriction base="xs:string">
      <xs:pattern value="#[0-9a-fA-F]{6}|lightblue|lightgreen|
                         brown|grey|lightgrey|pink|magenta|
                         cyan|none|black|maroon|olive|white|
                         silver|red|yellow|green|teal|navy|
                         purple|lime|aqua|blue|fuchsia">
      </xs:pattern>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="anchorType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="start"/>
      <xs:enumeration value="middle"/>
      <xs:enumeration value="end"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="styleType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="normal"/>
      <xs:enumeration value="italic"/>
       <xs:enumeration value="oblique"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="visibleType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="visible"/>
      <xs:enumeration value="hidden"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="fileType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="shp"/>
    <xs:enumeration value="grib"/>
  </xs:restriction>
  </xs:simpleType>
```

```
  <xs:simpleType name="actionType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="add"/>
      <xs:enumeration value="replace"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="ableType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="disable"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="includeType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="include"/>
      <xs:enumeration value="exclude"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="scriptType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="v1"/>
      <xs:enumeration value="v2"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="isoType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="area"/>
      <xs:enumeration value="line"/>
      <xs:enumeration value="point"/>
      <xs:enumeration value="wind"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="curveType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="false"/>
      <xs:enumeration value="true"/>
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
```

E Literaturverzeichnis

Die Literaturquellen beziehen sich nicht nur auf gedruckte Literatur, sondern auch auf Websites. Letztere bieten einen aktuelleren Zugang zu Informationen, vor allem in der recht schnellebigen Informatik, sind aber auch gerade deshalb schnell veraltet und werden gelöscht oder die URL ändert sich durch Umstrukturierungen auf dem Server. Zu dem Zeitpunkt der Abgabe dieser Dissertation waren alle Adressen erreichbar.

Literatur

[Abreu] **Abreu, J., et al**
Combining Spatial Metadata Search-Engines with Webmapping - Exemplified with Emergency Scenarios, GIS (2000) Nr.5, S.8-11

[Adobe1] **Adobe Systems Incorporated**
http://www.adobe.com

[Adobe2] **Adobe Systems Incorporated**
Adobew SVG Viewer download area, 2006
http://www.adobe.com/svg/viewer/install/main.html

[Adobe3] **Adobe Systems Incorporated**
Macromedia Flash Player: Version Penetration, 2006
http://www.adobe.com/products/player_census/
flashplayer/version_penetration.html

[Adobe4] **Adobe Systems Incorporated**
Flash and Shockwave Players: NPD Methodology, 2006
http://www.adobe.com/products/player_census/npd/

[Adobe5] **Adobe Systems Incorporated**
Adobe Flex 2, 2006
http://www.adobe.com/de/products/flex/

[Albrecht] **Albrecht, Sven**
Feature Extraction and vortex correspondence of 2D shapes, Bachelorarbeit an der Universität Osnabrück, 2006
http://www.inf.uos.de/prakt/pers/dipl/svalbrec

[Apache] **The Apache Software Foundation**
http://www.apache.org

[Batik] **The Apache Software Foundation**
Batik SVG Toolkit, 2005
http://xmlgraphics.apache.org/batik/

[Bin] **Bin, Hongzan**
Computer Aided Production Engineering,
John Wiley and Sons, 2002

[Bergmann] **Bergmann, Ludwig; Schaefer, Clemens**
Lehrbuch der Experimentalphysik, Band 7, Erde und Planeten, 2. Auflage
Gruyter, 2001

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

[Cartogis] **Cartogis**
Digitale Landkarten, 2006
http://www.cartogis.de/parallel/karten/digkart.htm

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

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

[Davis] **Davis, Tom; Neider, Jackie; Shreiner, Dave; Woo,Mason**
OpenGL Programming Guide, 5.Auflage
Addison Wesley, 2005

[DeKG] **Deutsche Gesellschaft für Kartographie e.V.**, 2006
http://www.dgfk.net

[DESTATIS] **Statistisches Bundesamt Deutschland**, 2006
http://www.destatis.de

[Dickmann] **Dickmann, Frank**
Das geographische Seminar: web-mapping und web-gis, 1. Auflage
Westermann Schulbuchverlag GmbH, Braunschweig 2001

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

[DRN] **Digital Radio Nord GmbH**
Digital Radio Nord, 2006
http://www.digitalradio-nord.de

[Dubuc] **Dubuc, S.**
Interpolation through an iterative scheme,
J. MATH. ANAL. APPLIC., 1986, Vol. 114, Nr. 1, S. 185-204

[DWD1] **Deutscher Wetterdienst**
Kompetenz für Wetter und Klima, 2006
http://www.dwd.de/de/de.htm

[DWD2] **Deutscher Wetterdienst**
Numerische Wettervorhersage, 2006
http://www.dwd.de/de/FundE/Analyse/Modellierung/Modellierung.htm

[Dyn1] **Dyn, N.; Gregory, J.A.; Levin, D.**
A four-point interpolatory subdivision scheme for curve design,
Computer Aided Geometric Design 4, 1987, S. 257-268

[Dyn2] **Dyn, N.; Levin, D.**
Subdivision Schemes in Geometric Modelling,
Acta Numerica 12, 2002, S. 73-144

[ECMA1] **European Computer Manufacturer's Association**
http://www.ecma-international.org

[ECMA2] **European Computer Manufacturer's Association**
ECMAScript Language Specification, 1999
http://www.ecma-international.org/publications/standards/
Ecma-262.htm

[Efrat] **Efrat, A.; Guibas, L.J.; Har-Peled, S.; Murali, T.M.**
Morphing between Polylines
Proc. 12th ACM-SIAM Sympos. Discrete Algorithms, 2001, 680-689

[Eisenberg] **Eisenberg, David J.**
SVG Essentials, 1. Auflage
O'Reilly, Cambridge 2002

[ESRI1] **ESRI**
ESRI - The GIS Software leader, 2006
http://www.esri.com

[ESRI2] **ESRI**
ESRI Shapefile Technical Description, 1998
http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf

[ESRI3] **ESRI**
ESRI World Basemap Data, 2006
http://www.esri.com/data/download/basemap/index.html

[EU] **EU Press Release**
Commission expects most broadcasting in the EU to be digital by 2010,
Reference IP/05/595, Mai 2005
http://europa.eu.int/radip/pressReleasesAction.de?reference=IP/05/595

[Fellner] **Fellner, Wolf-Dietrich**
Computergrafik, 2. Auflage
Spektrum Akademischer Verlag, Heidelberg 2002

[Flagstone] **Flagstone Software**
Tools for Flash Developers, 2006
http://www.flagstonesoftware.com

[Flanagan] **Flanagan, David**
Java in a Nutshell, 5. Auflage
O'Reilly, Cambridge 2005

[Fox] **Fox, Patrick**
Interaktive Visualisierung von optimierten Zugablaufplänen mit SVG am Beispiel des Bahnhofs Amsterdam-Schipol, Diplomarbeit an der Universität Osnabrück, 2005
http://www.inf.uos.de/prakt/pers/dipl/pfox

[Frida] **Intevation GmbH**
Frida: Freie Vektor-Geodaten Osnabrück, 2004
http://frida.intevation.org

[Gamma] **Gamma, Erich; Helm, Richard; Johnson, Ralph; Vlissides, John**
Design patterns: Elements of Reusable Object-Oriented Software, 32. print.
Addison-Wesley, Boston 2005

[Garret] **Adaptive Path, Jesse James Garret**
Ajax: A New Approach to Web Applications, 2005
http://www.adaptivepath.com/publications/essays/archives/000385.php

[Google1] **Google Inc.**
http://www.google.de

[Google2] **Google Inc.**
GoogleEarth, 2006
http://earth.google.com

[Greiner] **Greiner, Günther; Hormann, Kai**
Efficient clipping of arbitrary polygons,
ACM Transactions on Graphics (TOG), 1998, Vol. 17 ,S. 71-83

[Harold] **Harold, Elliotte R.; Means, W. Scott**
XML in a nutshell, 1. Auflage
O'Reilly, Cambridge 2001

[Inkscape] **Inkscape**
Inkscape - Draw Freely, 2006
http://www.inkscape.org

[ISO] **International Organisation for Standardization**
Intelligent transport systems – Geographic Data Files (GDF) – Overall data specification, 2004
http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=30763&ICS1=3&ICS2=220&ICS3=1

[JOGL] **JOGL**
The JOGL API Project, 2006
https://jogl.dev.java.net

[kleine Kruse] **kleine Kruse, Karl**
Approximation von Bézierkurven durch Polygonzüge, Bachelorarbeit an der Universität Osnabrück, 2005
http://www.inf.uos.de/prakt/pers/dipl/kkleinek

[Kowalski] **Kowalski, Hans-Joachim; Michler, Gerhard O.**
Lineare Algebra, 1. Auflage
Walter de Gruyter, Berlin 2003

[Kunze] **Kunze, R.; Langfeld, D.; Fox, P.; Vornberger,O.**
Klimadatenvisualisierung auf mobilen Endgeräten mit DAB
ITG-Fachtagung 194, VDE Verlag GmbH, Berlin, 2006, S. 145-150

[Langfeld] **Langfeld, Dorothee**
Entwicklung einer SVG Web Mapping Applikation zur Visualisierung von Geoinformationen, Diplomarbeit an der Universität Osnabrück, 2006
http://www.inf.uos.de/prakt/pers/dipl/dlangfel

[Lauer] **Lauer, Wilhelm; Bendix, Jörg**
Das geographische Seminar: Klimatologie, 2. Auflage
Westermann Schulbuchverlag GmbH, Braunschweig 2004

[Lingrand] **Lingrand, Diane**
The Marching Cubes, Université de Nice, 2002
http://www.essi.fr/~lingrand/MarchingCubes/accueil.html

[Liu] **Liu, Ligang; Wang, Guopu; Zhang, Bo; Guo, Baining; Shum, Heung-Yeung**
Perceptually Based Approach for Planar Shape Morphing, Microsoft Research, 2004
http://research.microsoft.com/research/pubs/view.aspx?pubid=1315

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

[MAP24] **Mapsolute GmbH**
Map24 - Routenplaner und Stadtpläne für Deutschland, Europa und USA, 2006
http://www.map24.de

[Miano] **Miano, John**
Compressed Image File Formats: JPEG, PNG, GIF, Xbm, BMP, 2. Auflage
Addison-Wesley, Boston 2000

[NOAA] **National Oceanic & Atmospheric Administration**, 2006
http://www.noaa.org

[OGC] **Open Geospatial Consortium**
Geography Markup Language (GML) Encoding Specification, 2006
http://www.opengeospatial.org/standards/gml/

[OpenMap] **BBN Solutions**
OpenMap, 2005
http://openmap.bbn.com

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

[Plewe] **Plewe, Brandon**
GIS Online - Information retrieval, mapping and the Internet
OnWord Press, 1997

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

[Raymarine] **Raymarine Inc.**
3 Day RayTech Weather Forecast, 2006
http://www.raymarine.com/raymarine/Default.asp?site=1&Section=3&Page=149&Parent=3

[Schwarz] **Schwarz, Hans Rudolf**
Numerische Mathematik, 4. Auflage
B.G. Teubner, Stuttgart 1997

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

[SnyderP] **Snyder, John P.**
Map Projections - A Reference Manual
Taylor & Francis Ltd., London 1995

[Sosna] **Sosna, Dieter**
Von der Erde zur Karte, 1999
http://www.informatik.uni-leipzig.de/~sosna/karten/ek.html

[Sutherland] **Sutherland, Ivan E.; Hodgman, Gary W.**
Reentrant polygon clipping,
Communications of the ACM, 1974, Vol.17, Nr.1, S.32-42

[SVG] **SVG.org**
Shipping and Announced SVG Phones, 2006
http://svg.org/special/svg_phones

[Teschl] **Teschl, Gerald; Teschl, Susanne**
Mathematik für Informatiker: Teil 1: Diskrete Mathematik und lineare Algebra, 1. Auflage
Springer, Berlin Heidelberg 2006

[UMN] **UMN Mapserver**
http://ms.gis.umn.edu

[Unidata] **Unidata Program Center**
NetCDF, 2006
http://www.unidata.ucar.edu/software/netcdf/

[Voser] **Voser, Stefan A.**
Koordinatensysteme, Geodätische Bezugssysteme, Kartenprojektionen, 1998
http://www.giub.uni-bonn.de/gistutor/theorie/grundlag/koordsys/koordina.htm

[W3C1] **World Wide Web Consortium**
http://www.w3.org

[W3C2] **World Wide Web Consortium**
Accessibility Features of SVG, 2000
http://www.w3.org/TR/SVG-access/

[W3C3] **World Wide Web Consortium**
Recognized color keyword names, 2003
http://www.w3.org/TR/SVG11/types.html#ColorKeywords/

[W3C4] **World Wide Web Consortium**
Working Draft: Scalable Vector Graphics (SVG) 1.2, 2004
http://www.w3.org/TR/2004/WD-SVG12-20041027/

[W3C5] **World Wide Web Consortium**
SVG Roadmap, 2005
http://www.w3.org/Graphics/Roadmap/

[W3C6] **World Wide Web Consortium**
Document Object Model (DOM), 2005
http://www.w3.org/DOM/

[W3C7] **World Wide Web Consortium**
Scalable Vector Graphics (SVG) - XML Graphics for the Web, 2006
http://www.w3.org/TR/SVG11/

[W3C8] **World Wide Web Consortium**
Mobile SVG Profiles: SVG Tiny and SVG Basic, 2003
http://www.w3.org/TR/SVGMobile/

[W3C9] **World Wide Web Consortium**
Scalable Vector Graphics (SVG) Tiny 1.2 Specification, 2006
http://www.w3.org/TR/SVGMobile12/

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

[Watt] **Watt, A.; Lilley, C.; Ayers, D.J.; George, R.; Wenz, C.; Hauser, T.; Lindsey, K.; Gustavsson, N.**
SVG Unleashed
Sams Publishing, Indianapolis 2003

[Wenke] **Wenke, Henning**
3D Klimadatenvisualisierung mit OpenGL,Bachelorarbeit an der Universität Osnabrück, 2005
http://www.inf.uos.de/prakt/pers/dipl/hwenke

[WGRIB] **Climate Prediction Center: Ebisuzaki, Wesley**
Climate Prediction Center - wgrib home page, 2006
http://www.cpc.ncep.noaa.gov/products/wesley/wgrib.html

[Wikipedia1] **Wikipedia**
Scalable Vector Graphics, 2006
http://de.wikipedia.org/wiki/SVG

[Wikipedia2] **Wikipedia**
Wikipedia: WikiProjekt Georeferenzierung, 2006
http://de.wikipedia.org/wiki/Wikipedia:WikiProjekt_Georeferenzierung

[Wikipedia3] **Wikipedia**
Kartenprojektionen, 2006
http://de.wikipedia.org/wiki/Kartenprojektion

[Wikipedia4] **Wikipedia**
Digital Audio Broadcasting, 2006
http://de.wikipedia.org/wiki/Digital_Radio

[Winter] **Winter, André M.; Neumann Andreas**
Comparing .SWF (ShockWave Flash) and .SVG (Scalable Vector Graphics) file format specifications, 2004
http://www.carto.net/papers/svg/comparison_flash_svg/

[XML] **O'Reilly Media**
XML.com: SVG Tips and Tricks: Adobe SVG Viewer, 2006
http://www.xml.com/pub/a/2002/07/03/adobesvg.html