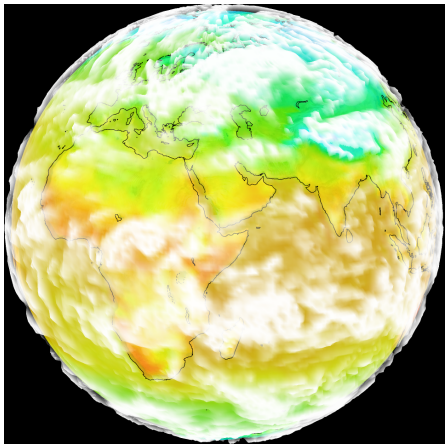
UNIVERSITÄT OSNABRÜCK

3D Klimadatenvisualisierung mit OpenGL

Bachelorarbeit von Henning Wenke

Gutachter:
Prof. Dr. Oliver Vornberger
Prof. Dr. May-Brit Kallenrode

14. März 2005



1 Danksagung

Ich bedanke mich hiermit bei allen Personen, die mich während der Bachelorarbeit unterstützt haben. Besonderer Dank gilt:

- Frau Prof. Dr. May-Brit Kallenrode für die gute Betreuung und für das Korrekturlesen
- Herrn Dipl.-Systemwiss. Ralf Kunze für die gute Betreuung , die konstruktiven Vorschläge und Anmerkungen sowie für das Korrekturlesen
- Herrn Prof. Dr. Oliver Vornberger für die gute Betreuung und für die Stellung des interessanten Themas

Außerdem möchte ich mich noch bei meiner Familie für die Unterstützung und fürs Korrekturlesen bedanken. Weiterer Dank gilt Christoph Abé für das Korrekturlesen, Thomas Wiemann für die Hilfe mit C++ und Patrick Fox für die Tipps im Umgang mit Latex.

Freren, im März 2005

Henning Wenke

Inhaltsverzeichnis

2 Einleitung

In der heutigen Zeit gewinnt die adäquate Präsentation von Produkten, Ideen aber auch von wissenschaftlichen Ergebnissen zusehends an Bedeutung. Auch die Forschung muss ihre Ergebnisse gut darstellen können, um Marktanteile zu sichern bzw. Forschungsgelder zu erhalten. Diese Entwicklung profitiert sowohl vom raschen Fortschritt der Computertechnologie als auch von der steigenden Verbreitung der Rechner, die mittlerweile für jeden erschwinglich geworden sind. Insbesondere im Bereich der 3D-Grafik zeigt sich dieser Trend deutlich: wurden 3D-Grafikkarten bis vor wenigen Jahren nur im professionellen Bereich eingesetzt, gehören sie heute zur Standardausstattung eines jeden PCs. Außerordentlich erfreulich ist dies für Forschungszweige, in denen es große Datenmengen zu überblicken gilt. Hier können durch grafisches Darstellen der Messwerte selbst kleine Besonderheiten oder auch Fehler unmittelbar erkannt werden - *Ein Bild sagt mehr als 1000 Worte* - heißt es nicht umsonst im Volksmund.

Aus diesen Gründen habe ich mich für die Bachelorarbeit *3D Klimadatenvisualisierung mit OpenGL* entschieden. Für diese soll ein Programm erstellt werden, das orts- und zeitabhängige Klimadaten (Temperatur, Bewölkungsdichte und Windrichtung) einliest und diese auf eine Erdkugel projiziert darstellt. Es findet hierzu die frei verfügbare Grafikschnittstelle OpenGL Verwendung, da sie von allen aktuellen Grafikkarten unterstützt wird und ihr Einsatzbereich im Gegensatz zu Microsofts DirectX nicht auf Windows-Rechner beschränkt ist.

3 OpenGL Kurzeinführung

3.1 Allgemeines

OpenGL stellt eine Schnittstelle zwischen Anwenderprogramm und der Grafikhardware dar und dient zum Modellieren und Projizieren von geometrischen Objekten. [Vor04]
Bestandteil der OpenGL-Schnittstelle sind:

- 200 Befehle in der OpenGL Library (beginnen mit `gl`) zum Verwalten von elementaren geometrischen Primitiven wie Punkten, Linien, Polygonen, Bezierkurven und ihrer Attribute wie Farben und Normalenvektoren.
- 50 Befehle in der OpenGL Utility Library (beginnen mit `glu`) zum Verwalten von NURBS (non uniform rational b-splines) und quadrics (Körper mit durch quadratische Gleichungen beschreibbaren Oberflächen, z.B. Kugel, Zylinder) sowie zum vereinfachten Manipulieren von Projektionsmatrizen.
- 30 Befehle im OpenGL Utility Toolkit (beginnen mit `glut`) zur Anbindung der von OpenGL gerenderten Ausgabe an das jeweilige Fenstersystem und zum Verwalten von höheren geometrischen Objekten wie Kugel, Kegel, Torus und Teapot.
  Das OpenGL Utility Toolkit ist nicht im Standard OpenGL Paket enthalten und außerdem plattformabhängig, weshalb es in dieser Arbeit nicht genutzt wird.

Um die Kommunikationsbandbreite zu minimieren, arbeitet OpenGL als Zustandsmaschine. Dies bedeutet, dass einmal gesetzte Zustände (z.B. die Vordergrundfarbe) bis zu ihrem Widerruf beibehalten werden.
Bei manchen OpenGL-Implementationen (z.B. X Window System) kann die Berechnung der Grafik und ihre Ausgabe auf verschiedenen Rechnern stattfinden. Der Klient ruft hierbei ein OpenGL- Kommando auf. Mithilfe eines Protokolls wird es übertragen und der Server setzt es in ein Bild um.

3.2 Elementare Geometrien

3.2.1 Vertices

Vertices (Scheitelpunkte) definieren die Eckpunkte geometrischer Formen. In OpenGL setzt man Vertices folgendermaßen:

```
glVertex{234}  Dimension
        {sifd} Datentyp (short, int, float, double)
        [v]    Vektoriell (optional)
        (Koordinaten);
```

Dabei ist bei zweidimensionaler Darstellung die z-Komponente immer Null. Bei vierdimensionalen Vertices kann die homogene Koordinate, die für die Korrektur der Perspektive benötigt wird, explizit angegeben werden. Bei dreidimensionalen Eckpunkten wird sie implizit auf eins gesetzt. Beispiel:

```
glVertex3d(10.0, 0.0, 1.0);
```

3.2.2 Primitive

Primitive sind elementare grafische Grundelemente wie Punkte, Linien oder Flächen, die mithilfe der GL-Bibliothek erzeugt werden. Auf Basis dieser Elemente können komplexere Gebilde wie Kugeln erzeugt werden, aus denen sich letztendlich die finale Szene zusammensetzt.
Die Primitive sind im Einzelnen:

- `GL_POINTS`

  Ein Punkt wird definiert durch einen Vertex mit räumlicher Ausdehnung.

- `GL_LINES`

  Eine Linie wird definiert durch zwei Vertices und eine gewisse Linienbreite.

- `GL_LINESTRIP`

  Eine Linienfolge wird definiert durch mindestens zwei Vertices, die der Reihe nach mit Linien verbunden werden.

- `GL_LINELOOP`

  Ein geschlossener Linienzug ist eine Linienfolge, bei dem der letzte Punkt implizit mit dem ersten verbunden wird.

- `GL_TRIANGLES`

  Ein Dreieck wird durch drei Vertices definiert.

- `GL_TRIANGLE_FAN`

  Ein Trianglefan besteht aus einer Serie von Dreiecken, die durch mindestens drei Vertices definiert wird, wobei der erste Eckpunkt aller folgenden Dreiecke ist.

- `GL_QUAD`

  Ein Viereck wird durch vier Vertices definiert.

- `GL_QUAD_STRIP`

  erzeugt eine Serie von Vierecken, die durch $4 + 2 \cdot n$ Vertices definiert wird.

Beispiele für einige Primitive sind in Abb. 1 dargestellt.

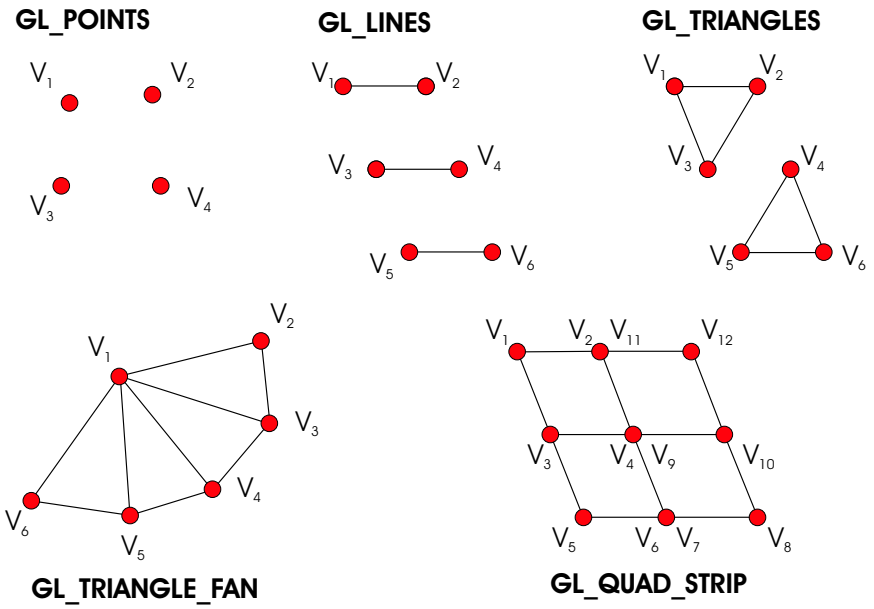


Abbildung 1: Einige elementare Primitive in OpenGL

3.3 Vertex Pointer

Bei der Modellierung eines Körpers mit den bisherigen Mitteln müssen einzelne Vertices im Allgemeinen mehrmals übergeben werden. Hier schafft der Vertex-Pointer, dem alle Vertices nur einmal übergeben werden müssen, Abhilfe [uMB03]. Ein zusätzliches Array enthält Informationen, in welcher Reihenfolge die einzelnen Scheitelpunkte zu der gewünschten Form zusammengesetzt werden sollen. Analog dazu erfolgt die Angabe der Farben, Texturkoordinaten oder Normalen über Colorpointer, TextureCoordinatePointer und NormalPointer.
Ein Vertex-Pointer Aufruf sieht wie folgt aus:

```
glVertexPointer(3, GL_DOUBLE, 0, geometry);
```

Dabei gibt der erste Parameter die gewünschte Dimension an, der zweite den Datentyp und der dritte den Offset zwischen den einzelnen Werten. Der letzte Parameter ist eine Referenz auf das Array mit den Vertices.
Die Ausgabe erfolgt am effizientesten über:

```
glDrawElements(GL_QUAD_STRIP,    // Primitivart
               verticesCnt,      // Vertexzahl
               GL_UNSIGNED_BYTE, // Datentyp der Farben
               topology);        // Topologie-Array
```

Dabei ist der letzte Parameter eine Referenz auf das Array, das die Ordnung der Vertices angibt.

Die Funktionsweise des Vertex-Pointers ist in Abb. 2 ersichtlich.

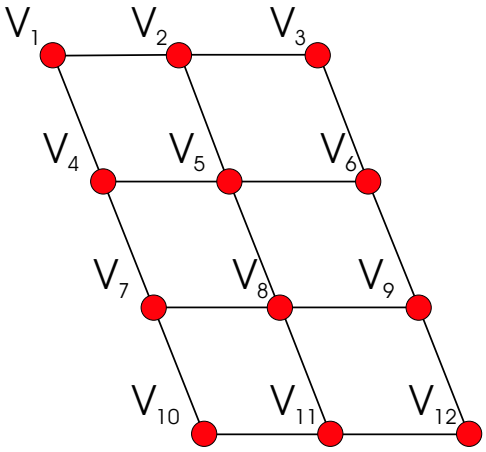


Abbildung 2: Veranschaulichung der Funktionsweise des Vertex Pointers. Das Topologie-Array ist [1,4,2,5,3,6,4,7,5,8,6,9,7,10,8,11,9,12]

3.4 Viewing-Pipeline

Die fotorealistische Darstellung von 3D-Objekten einer Szene auf dem 2D-Bildschirm lässt sich beschreiben durch eine Sequenz von Transformationen, genannt Viewing Pipeline.
Diese setzt sich in OpenGL aus folgenden Schritten zusammen, die der Abb. 3 zu entnehmen sind [Bur03] [JN94]:

- Modeling Transformation
  Die Szene ist anfänglich durch die explizit gesetzten Objektkoordinaten der Scheitelpunkte der Primitive festgelegt. Hierbei werden absolute Koordinaten verwendet, die sich auf den globalen Ursprung der Szene beziehen. Die Ortsvektoren der Vertices werden bei der Modeling-Transformation mit der Modelview-Matrix (`GL_MODELVIEW`) multipliziert, wodurch die Objektkoordinaten in Eyekoordinaten überführt werden. Diese Sichttransformation kann man sich in Form einer virtuellen Kamera vorstellen, die sich irgendwo im Raum befindet und auf die Szene schaut.

- Projection Transformation
  In diesem Schritt wird nur ein bestimmter Bereich für die tatsächliche Darstellung ausgewählt. Dabei werden die Eyekoordinaten durch Multiplikation mit der Projektionsmatrix (`GL_PROJECTION`) in Clipkoordinaten überführt. Anschließend muss nur noch der Teil der Szene weiterverarbeitet werden, der sich auf der sichtbaren Seite aller Clipplanes befindet.

- Perspective Division
  Die Perspective Division bestimmt die räumliche Größenordnung von Szenenelementen in Abhängigkeit von ihrer Entfernung zum Betrachter. Hierbei werden die Clipkoordinaten entweder mit der Matrix `glOrtho` im Falle der Orthogonalprojektion oder mit der Matrix `glFrustum` multipliziert, falls perspektivische Projektion gewünscht ist. Am Ende dieser Transformation liegen in jedem Falle normalisierte Gerätekoordinaten vor.

- Viewport Transformation
  In diesem Schritt wird die in normalisierten Gerätekoordinaten vorliegende Szene noch auf den rechteckigen 2D-Bereich des Grafikfensters projiziert. Dies geschieht mit der Viewport Matrix (`glViewport`). Als finale Koordinaten ergeben sich Window Coordinates.

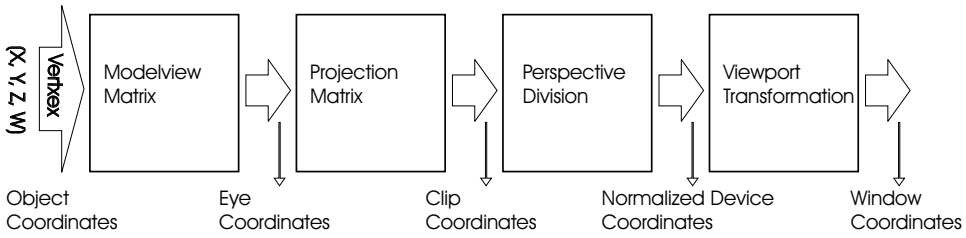


Abbildung 3: Die Viewing Pipeline in OpenGL

3.5 Perspektive

3.5.1 Kamera

Die Kamera legt zusammen mit den Clipplanes den sichtbaren Teil der Szene fest. Sie wird definiert über ihren Standpunkt (PRP), den fokussierten Punkt (VRP) und den View-Up-Vector (VUP), der die Orientierung der Kamera angibt.
Der Befehl zum Setzen der Kamera ist:

```
gluLookAt(PRP.x, PRP.y, PRP.z,
          VRP.x, VRP.y, VRP.z,
          VUP.x, VUP.y, VUP.z);
```

3.5.2 Orthogonalprojektion

Bei der Orthogonalprojektion definieren sechs zueinander orthogonale Clipplanes den sichtbaren Teil der Szene (Vergleich Abb. 4). Die Größe der Ob-

jekte hängt bei dieser Art der Projektion nicht vom Abstand zur Kamera ab. Vorteil dieser Darstellung ist das einfache Zoomen, schließlich muss nur die Kantenlänge des Kastens verkürzt werden, um die Szene zu vergrößern. Der Befehl zum Setzen der Matrix für die Orthogonalprojektion ist:

```
glOrtho(Left,      // linke   Clipplane
        Right,     // rechte  Clipplane
        Bottom,    // untere  Clipplane
        Top,       // obere   Clipplane
        Near,      // vordere Clipplane
        Far);      // hintere Clipplane
```

Dabei ermöglicht die Multiplikation der Abstände `Left`, `Right`, `Bottom`, `Top`, `Near` und `Far` vom VRP mit einem gemeinsamen Faktor auf sehr einfache Art das Zoomen.

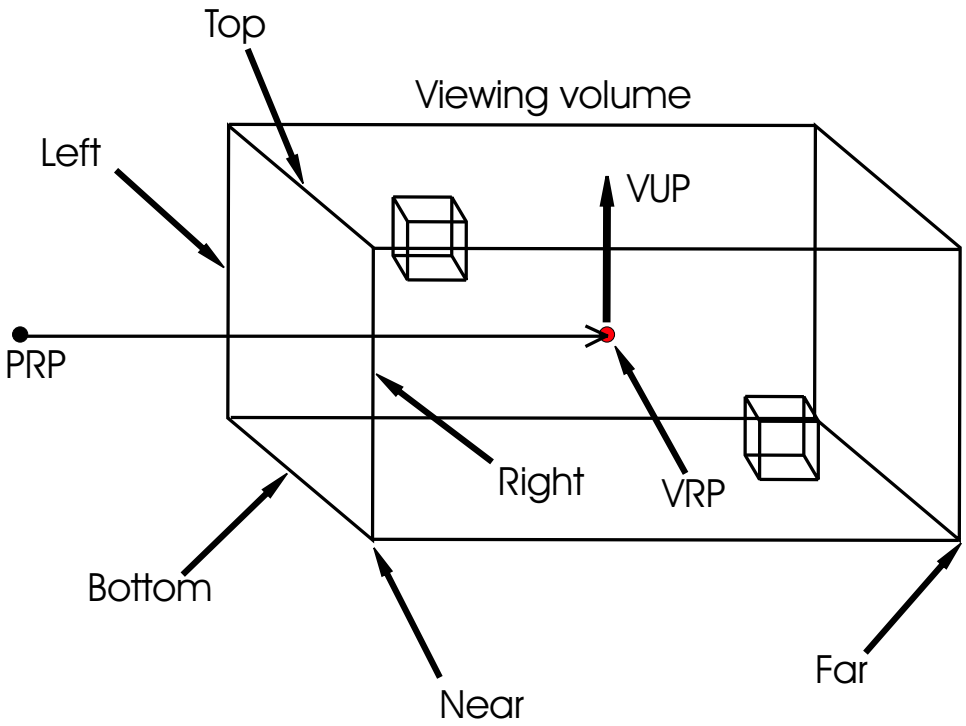


Abbildung 4: Skizze zur Veranschaulichung der Orthogonalprojektion

3.6 Shading

Die Art des Shadings gibt an, nach welchem Schema ein Dreieck in Abhängigkeit von der Farbe seiner Eckpunkte eingefärbt wird.

3.6.1 Farben

Farben können in OpenGL in RGB(A)-Form angegeben werden:

```
     glColor{34}        //  Alpha Wert gewünscht oder nicht
            {bsifd...}  //  Datentyp
            [v]         //  Vektoriell (optional)
            (Farbe);
Beispiel: glColor4b(255, 0, 0, 127)   // halbtransparentes rot
```

Alternativ kann auch der Color-Index-Mode benutzt werden. Da dort aber wichtige Funktionen wie Transparenz nicht vorgesehen sind, ist dieser Modus für diese Arbeit nicht zu gebrauchen.

3.6.2 Flatshading

Beim Flatshading werden im Allgemeinen die Farbwerte gemittelt und das Dreieck mit der resultierenden Farbe eingefärbt (Vergleich Abbildung 5 links).

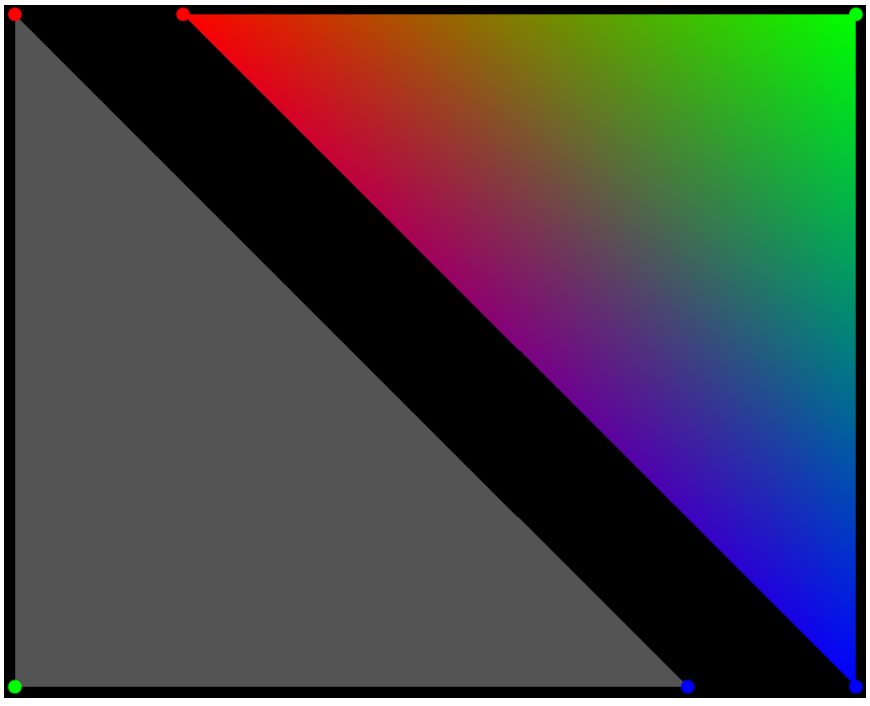


Abbildung 5: Vergleich der Shadingverfahren: Flat (links) und Smooth (rechts)

Offensichtlich stellt das Flatshading nur eine sehr grobe Näherung dar. Schließlich entspricht die Farbe in den Eckpunkten im Allgemeinen nicht der

gesetzten und der Farbübergang zu benachbarten Dreiecken ist unstetig.
Flat-Shading wird aktiviert über:

```
glShadeModel(GL_Flat);
```

In OpenGL nimmt ein mit Flatshading eingefärbtes Polygon einfach die Farbe des letzten es definierenden Vertex an.

3.6.3 Smoothshading

Beim Smoothshading bildet die Farbe einen Verlauf zwischen den Eckpunkten. (Vergleich Abbildung 5 rechts).

Diese Art des Shadings ist weitaus geeigneter als das Flatshading, nimmt doch zum einen der Farbverlauf in den Eckpunkten den exakten Wert an, zum anderen ist der Farbübergang zu benachbarten Dreiecken stetig. Da aber die Ableitung des Farbverlaufs an den Grenzen unstetig ist, können trotzdem noch die einzelnen Dreiecke erahnt werden. Dieses Phänomen wird als Mach-Band-Effekt bezeichnet.
Der zusätzliche Rechenaufwand kann von modernen Grafikkarten ohne nennenswerten Performanceverlust bewerkstelligt werden.
Insgesamt ist somit das Smooth-Shading dem Flat-Shading sowohl in Bezug auf Bildqualität als auch hinsichtlich der Exaktheit der Darstellung überlegen.
Smooth-Shading wird aktiviert über:

```
glShadeModel(GL_SMOOTH);
```

3.7 Listen

Listen können unterschiedlichste Szeneneigenschaften wie geometrische Objekte, Projektionsmatrizen oder gesetzte Optionen kapseln. Dies dient zunächst der Strukturierung. So könnte etwa eine Liste *Haus* aus den Listen *Dach* und *Gemäuer* bestehen, diese wiederum könnten die nötigen Primitive enthalten, mit denen die entsprechenden Gebilde dargestellt werden. Soll nun ein Haus gezeichnet werden, genügt ein Aufruf der Liste *Haus*.

Ein weiterer wesentlicher Vorteil der Listentechnik besteht darin, dass in Listen organisierte Szenenelemente sich fest im Grafikkartenspeicher befinden und entsprechend äußerst effizient bearbeitet und dargestellt werden können.

3.8 Hidden-Surface-Removal

Der Begriff des Hidden-Surface-Removals kapselt verschiedene Mechanismen, mit deren Hilfe nicht sichtbare Flächen entfernt werden können. Dadurch werden verborgene Objekte bzw. verdeckte Teile größerer Gebilde unsichtbar

und die Grafikkarte wird entlastet, da diese nicht gezeichnet werden müssen. Diese Mechanismen sollen im Folgenden genauer erläutert werden.

- **Backface-Culling**
  Abhängig vom Drehsinn werden Vorder- und Hinterseite einer Fläche festgelegt und ggf. ausgeblendet. Das Backface-Culling wird in OpenGL aktiviert über:

```
glEnable(GL_CULL_FACE);   // Culling aktivieren
glFrontFace(GL_CCW);      // Drehsinndefinition
glCullFace(GL_BACK);      // Rückseiten ausblenden
```

- **Clipping**
  Sechs Ebenen, die als Clipplanes bezeichnet werden, legen den sichtbaren Bereich fest. Jede Clipplane teilt den Raum in eine sichtbare und eine unsichtbare Hälfte. Nur der Teil der Szene, der sich in Bezug auf alle Clipplanes auf der sichtbaren Seite befindet, wird zur Darstellung gebracht, der Rest wird abgeschnitten (geclippt). Zusätzlich zu diesen durch die Projektionsmatrix festgelegten Clipplanes können noch weitere manuell gesetzt werden.

- **Z-Buffer**
  Der Z-Buffer ist ein Tiefentest, der in Device Koordinaten arbeitet. In Abhängigkeit von der (noch immer vorhandenen) z-Koordinate werden die Farbwerte von weiter vom Betrachter entfernt liegenden Pixeln durch die des am nächsten liegenden Pixels ersetzt. Der Z-Buffer wird in OpenGL aktiviert über:

```
glEnable(GL_DEPTH_TEST); // Z-Buffer aktivieren
glDepthRange(-1,1);      // Tiefenrichtung definieren
glDepthFunc(GL_LEQUAL);  // Tiefenpuffervergleich
```

3.9 Blending

Durch das Blending werden Transparenzeffekte möglich, d.h. Objekte, die hinter anderen liegen, können durch diese hindurchscheinen. Dazu findet ein Vergleich zwischen verdeckendem (`SRC`) und verdecktem Pixel (`DST`) statt. Dies geschieht, bevor die endgültigen Pixel in den Framebuffer geschrieben werden. Die OpenGL-Funktion

```
glBlendFunc(sfactor, dfactor);
```

regelt das Farbmischungsverhältnis von `SRC` und `DST`. Dabei geben die Parameter `sfaktor` und `dfaktor` an, mit welchen Faktoren die Alpha Werte von

Quellkonstante `sfactor`	**Zielkonstante** `dfactor`
GL_ZERO	GL_ZERO
GL_ONE	GL_ONE
GL_DST_COLOR	GL_SRC_COLOR
GL_SRC_ALPHA	GL_SRC_ALPHA
GL_DST_ALPHA	GL_DST_ALPHA
GL_ONE_MINUS_DST_COLOR	GL_ONE_MINUS_SRC_COLOR
GL_ONE_MINUS_SRC_ALPHA	GL_ONE_MINUS_SRC_ALPHA
GL_ONE_MINUS_DST_ALPHA	GL_ONE_MINUS_DST_ALPHA
GL_SRC_ALPHA_SATURATE	

Tabelle 1: Mögliche Werte, die die Blendigfaktoren annehmen können

`SRC` bzw. `DST` zu multiplizieren sind. Tabelle 1 enthält die möglichen Werte, die die Konstanten `sfaktor` und `dfactor` annehmen können.

Beispiel:

```
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```

Durch diese Einstellung wird bei identischen Alpha Werten der vordere Pixel zum Dominanten, die resultierende Farbe wird demnach an seinen RGB-Werten näher liegen als an denen des hinteren Pixels.

3.10 Texturierung

Bei der Texturierung wird eine zweidimensionale Pixelgrafik auf die Oberfläche eines Primitivs projiziert, um dieses mit Oberflächendetails zu versehen. Mithilfe der Farbpunkte dieser Textur (Texel) kann für jeden Punkt des Monitors (Pixel) die Farbe bestimmt werden, die sich aus einem gewichteten Mittelwert aller überdeckten Texel ergibt. Die Textur muss als 24 Bit Bitmap vorliegen und sollte über Kantenlängen verfügen, die durch Zweierpotenzen darstellbar sind. Diese Länge ist optimal, da aus den geladenen Texturen automatisch mehrere mit geringerer Auflösung erzeugt werden, die jeweils über die halbe Kantenlänge der Nächstgrößeren verfügen. Durch dieses als Mipmapping bezeichnete Verfahren kann in Abhängigkeit von der Entfernung des Betrachters zur Fläche die optimal aufgelöste Textur bereitgestellt werden, sodass nicht zur Laufzeit für jedes Pixel die Summe über alle überdeckten Texel gebildet werden muss. Stattdessen werden die Pixel in die Version der Textur projiziert, in der sie etwa ein bis zwei Texel überdeckt.

Soll nun beispielsweise ein Viereck texturiert werden, muss zunächst die Textur in den Speicher geladen werden. Dann werden die normierten Texturkoordinaten den einzelnen Vertices zugewiesen. Ein wesentlicher Nachteil dieser Technik liegt darin, dass bei stärkeren Vergrößerungen die Darstellung aufgrund der begrenzten Texelzahl zunehmend unschärfer wird, Texturen sind folglich nur begrenzt skalierbar. [Vor04] [Bur03]

4 Daten

4.1 Grib-Files

Die verwendeten Klimadaten stammen vom Deutschen Klima Rechenzentrum (DKRZ), der Arbeitsgruppe Modelle und Daten des Max Planck Instituts (MPI) und sind in Grib (Gridded Binary)-Files enthalten.
In den Grib-Files sind wiederum Records enthalten, die die Menge der benötigten Klimadaten zu einem festen Zeitpunkt darstellen. Die Grib-Files können sowohl Klimasimulationsdaten als auch Referenzwerte enthalten, die wiederum von verschiedener Art sein können (Temperatur, Wind, Bewölkungsgrad,...). Des Weiteren sind Informationen über den jeweiligen Erdausschnitt, die Auflösung, den Zeitpunkt und die Extrema enthalten. Darüber hinaus beinhalten die Grib-Files natürlich noch die darzustellenden Werte, die durch das Programm eingelesen werden können:

- **Temperaturen**
  Enthalten die Oberflächentemperaturwerte in Kelvin
- **Bewölkungsgrad**
  Die möglichen Werte liegen zwischen null (keine Bewölkung) und eins (vollkommen bedeckter Himmel)
- **Wind**
  Der Wind ist eine zweidimensionale vektorielle Größe. Es sind hierzu zwei Records einzulesen, das Eine für die u-Komponente, die zum Nordpol zeigt, und ein Weiteres für die nach Osten zeigende v-Komponente. Die resultierenden Vektoren geben durch ihre Richtung die Windrichtung an diesem Ort an und durch ihren Betrag die Windstärke.

Die weiteren Klimadaten, wie z.B. Niederschlag und Luftdruck, können in dieser Programmversion noch nicht dargestellt werden, da der Schwerpunkt dieser Bachelor-Arbeit auf der Entwicklung und Optimierung der Darstellungsmethode liegt, nicht jedoch in der möglichst umfassenden Darstellung.

4.2 Durch das Programm einlesbare Daten

Das Programm liest Textdateien ein, die auf den Grib-Files basieren. Diese müssen mehrere Konventionen einhalten, um eingelesen und richtig interpretiert werden zu können.
Da die Daten in jedem Fall für den Einsatz in diesem Programm aufbereitet werden müssen, wird davon ausgegangen, dass sie die Konventionen exakt einhalten und somit zugunsten der Performance auf eine Fehlerkontrolle verzichtet werden kann.
Im Dateinamen muss die Art des Records enthalten sein, *clouds* für Wolken, *temp* für Temperaturen und *wind* für eine Windkomponente.

Die erste Zeile muss in der folgenden Reihenfolge diese Angaben enthalten:

1. **Anzahl der Intervalle in geographischer Breite und Länge**
   Die Angaben über die Gitterauflösung werden zum einen für den Aufbau der Geometrie benötigt, zum anderen steht damit fest, nach wie vielen Werten das Record zu Ende ist.

2. **Erster/Letzter Breitengrad, Erster/Letzter Längengrad**
   Die Angabe der ersten und letzten Längen- und Breitengrade legt fest, ob die gesamte Erdkugel oder nur ein bestimmter Ausschnitt daraus darzustellen ist.

3. **Maximalwert, Minimalwert**
   Die Extrema werden benötigt um das Farbspektrum für die Temperatur bzw. die maximale Windstärke festzulegen.

4. **Anzahl der Records**
   Die Recordzahl wird nur benötigt um die Animation nach ihrer Beendigung wieder neu zu starten.

5. **Der Zeitraum ddmmyyyyhhdt, der durch die Records abgedeckt wird**
   Er setzt sich zusammen aus:

   - Zwei Stellen dd für den Tag
   - Zwei Stellen mm für den Monat
   - Vier Stellen yyyy für das Jahr
   - Zwei Stellen hh für die Stunde
   - Mindestens zwei Stellen dt für die Zeitdifferenz zum nächsten Record in Stunden

   Nur wenn Zeitpunkt und Zeitdifferenz zwischen zwei Records bekannt sind, ist es möglich für jedes dargestellte Bild, insbesondere bei zeitlich Interpolierten, das korrekte Datum anzugeben.

Soll beispielsweise die gesamte Erdkugel in der Auflösung $320 \cdot 160$ mit hundert Records, Werten zwischen -57 und 50 im Abstand von sechs Stunden und dem Startzeitpunkt 22.11.2049 sechs Uhr dargestellt werden, so müsste die Datei die folgende Kopfzeile haben:

```
320 160 0.0 360.0 0.0 180.0 50.0 -57.0 100 221120490606
```

Das Einlesen der Werte erfolgt zeilenweise, wobei jede Zeile in einem Character-Array abgespeichert wird, welches zunächst in einen Zahlentyp umgewandelt werden muss. Dies ist mit der C-Funktion `atof` allgemeingültig möglich. Es hat sich jedoch gezeigt, dass diese Methode bei der großen

Zahl der umzuwandelnden Arrays zu großen Performanceverlusten führt. Daher enthält das Programm Methoden, die Character-Arrays in Zahlen umwandeln und davon profitieren, dass von den zu erwartenden Zahlen einiges bekannt ist:

- Bewölkung
  Alle möglichen Werte sind im Intervall $[0, 1]$ enthalten

- Temperatur
  Die in Kelvin gegebenen Werte sind auf jeden Fall positiv und dreistellig, d.h. im Intervall $[100.0K, 1000.0K[$ enthalten.
  Zum Vergleich: Der Negativrekord liegt bei 184 K in der Antarktis (Vostsol, 21. Juni 1983) und der Positivrekord bei 330 K (gemessen im Death Valley)

- Wind
  Die in $\frac{m}{sec}$ vorliegenden Werte sind höchstens dreistellig.
  Dies wird im Vergleich mit der Schallgeschwindigkeit deutlich, die in Luft bei 273 K etwa 332 $\frac{m}{sec}$beträgt.

Der sehr einfache Code zum Umrechnen eines Character-Arrays, welches eine Temperatur in Kelvin repräsentiert, in einen Celsiuswert, der in einem `double` codiert ist, sieht folgendermaßen aus:

```
double tempCharArray2double(char *kelvinArray){
       return((kelvinArray[0]-48)  *100.0
             +(kelvinArray[1]-48)  *10.0
             + kelvinArray[2]-48
             +(kelvinArray[4]-48)  /10.0
             +(kelvinArray[5]-48)  /100.0
             +(kelvinArray[6]-48)  /1000.0
             + ...
             -273.0);
}
```

Die Umrechnung profitiert offensichtlich davon, dass sich das Komma auf jeden Fall an der dritten Stelle befindet. Außerdem werden nicht beliebig viele Nachkommastellen berücksichtigt, da die sich durch derart minimale Unterschiede ergebende Abweichung in der Visualisierung durch das menschliche Auge nicht erkennbar ist.

4.3 Lesen aus Grib-Files

Die Ideallösung wäre natürlich, die Daten direkt aus den Grib-Files zu lesen, da auf diese Weise die Performance deutlich erhöht würde und echte Daten ohne Vorbehandlung eingelesen werden könnten. Dies ist in der aktuellen Programmversion noch nicht implementiert.

5 Berechnung der Geometrie

5.1 Kugelkoordinaten

Das Programm soll Klimadaten der Erde, oder eines Erdausschnitts graphisch darstellen können, daher ist die Verwendung von Kugelkoordinaten naheliegend. Für deren Normalen gilt [Kal05]:

$$\begin{pmatrix} n_x \\ n_y \\ n_z \end{pmatrix} = \begin{pmatrix} cos(\varphi) \cdot sin(\theta) \\ sin(\varphi) \cdot sin(\theta) \\ cos(\theta) \end{pmatrix}$$

Die Ortsvektoren der Scheitelpunkte ergeben sich dann aus dem Produkt von Kugelradius und der zugehörigen Normalen:

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

Dabei ist r_{Kugel} der Kugelradius, $\theta \in [0, \pi]$ der (Nord)Polabstand und $\varphi \in [0, 2\pi[$ gibt den Breitengrad an.

5.2 Displacement Mapping

Displacement Mapping [Vor04] bezeichnet allgemein eine Technik, bei der eine Oberfläche verändert werden kann, indem eine größere ebene Fläche in mehrere kleinere unterteilt wird, deren Eckpunkte entlang der Flächennormalen verschoben werden. Die Stärke der Verschiebung ist in der Displacement-Map als Grauwert hinterlegt.

In dem Programm ist ein ähnliches Verfahren implementiert, bei dem allerdings nur die bereits vorhandenen Vertices entlang ihrer Normalen verschoben werden, um die Oberflächenbeschaffenheit der Erde anzudeuten. Diese Information wird in den Normalen abgespeichert, damit die Verschiebung bei den 3D-Wolken nicht für jedes Bild neu berechnet werden muss.
Die Oberflächenstruktur der texturierten Erdkugel ist links auf der Abbildung 6 zu sehen. Im Vergleich dazu zeigt Abbildung 6 rechts die texturierte Erdkugel ohne Oberflächenstrukturen.

Aufgrund der Implementation des Displacementmappings müssen Temperatur- und Texturkugel exakt die gleiche Geometrieauflösung haben, da Erhebungen sonst nicht zwangsläufig bei beiden identisch sind. Folglich würde

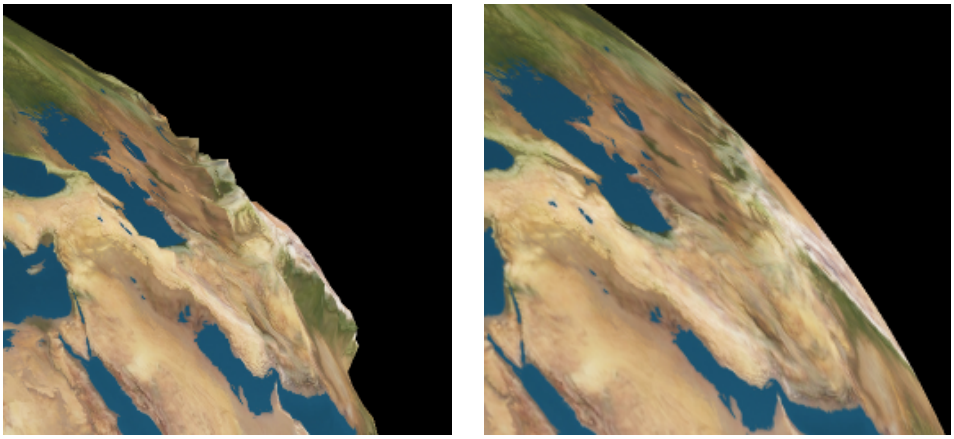


Abbildung 6: Vergleich der Erdtextur mit Displacementmapping (links) und ohne (rechts)

in diesem Fall die Texturkugel nicht unbedingt vollständig unter der Temperaturkugel liegen, was zu unschönen Grafikfehlern führen kann. Des Weiteren musste das Displacement Mapping auch bei den Wolken implementiert werden, da sonst größere Gebirgszüge (Himalaja) die Wolkenkugel durchdringen könnten. Die Küstenlinien dagegen befinden sich naturgemäß in Höhe des Meeresspiegels, weshalb hier theoretisch keine Anpassungen nötig sind. Aufgrund von Rechenungenauigkeiten, bzw. der begrenzten Auflösung der verwendeten Displacementmap kann es jedoch trotzdem vorkommen, dass kleinere Abschnitte der Küstenlinien in der Erdkugel versinken. Da dieser Graphikfehler nur selten auftritt, wird er in der weiteren Optimierung der Darstellung nicht gesondert korrigiert, sondern ignoriert.

6 Darstellung

6.1 Allgemeines

Der Schwerpunkt dieser Arbeit liegt bei der dreidimensionalen Echtzeitdarstellung von Daten. Dabei ist im Wesentlichen der optimale Kompromiss zu finden aus:

- Bildqualität
  Die Darstellung sollte möglichst ansprechend sein.
- Performance
  Die Animation soll möglichst flüssig und mit gleichmäßiger Geschwindigkeit ablaufen.
- Genauigkeit
  Die Daten müssen möglichst exakt wiedergegeben werden.

6.2 Geographie

Um dem Betrachter eine Orientierungsmöglichkeit zu geben, wird die Geographie der Erde bzw. des gewünschten Erdausschnitts angezeigt. Dabei sind zwei verschiedene Ansätze realisiert worden.

- Der erste ist, auf eine minimal kleinere Kugel eine Erdtextur zu projizieren (vgl. Abbildung 7). Dadurch kommt es insbesondere in Verbindung mit der Wolkendarstellung zu einer sehr guten Visualisierung, da der Bewölkungsgrad noch nahezu unvermindert erkennbar ist.
  Zusammen mit Temperaturen kommt es aufgrund der Farbvermischung durch das Blending zu einer Verfälschung der Temperaturwerte, werden diese doch über die Farbe visualisiert.
  Ein weiterer Nachteil ist natürlich die begrenzte Skalierbarkeit der Texturen, gerade wenn nur kleine Kartenausschnitte, etwa Deutschland, angezeigt werden, bleibt ein Verschwimmen der Textur nicht aus.
  Der Hardwaremehraufwand mit zugeschalteter Texturkugel ist relativ gering, die Framerate bleibt auf aktuellen Rechnern nahezu konstant, lediglich der Speicherbedarf steigt im Falle einer sehr hochauflösenden Textur erheblich an. Dieser liegt zwischen 250MB bei der höchsten Texturauflösung (10000*5000 Pixel) und 5 MB bei der geringsten (1000*500 Pixel).

- Der zweite Ansatz ist, die Geographie der Erde über einfarbige (schwarze) Küstenlinien anzudeuten (vgl. Abbildung. 8). Die Temperaturdarstellung wird nur minimal verfälscht, bei Wind oder Wolken dagegen

ist diese Orientierungshilfe allein nicht ausreichend, da sie gerade im Landesinneren nicht genügend Anhaltspunkte liefert.
Im Gegensatz zu den Texturen sind die aus Linestrips bestehenden Küstenlinien beliebig skalierbar. In Bezug auf den Hardwareaufwand lässt sich festhalten, dass die Animationsgeschwindigkeit bei zugeschalteten Küstenlinien ebenfalls nahezu konstant bleibt und der zusätzliche Speicherbedarf in Höhe von knapp 2MB bei heutigen Rechnern vernachlässigbar ist.

Insgesamt kann auf keinen der beiden Ansätze verzichtet werden, mit Temperaturen sind die Küstenlinien optimal, bei Wind und Wolken dagegen die Erdtextur.

Abbildung 7: Die texturierte Erdkugel

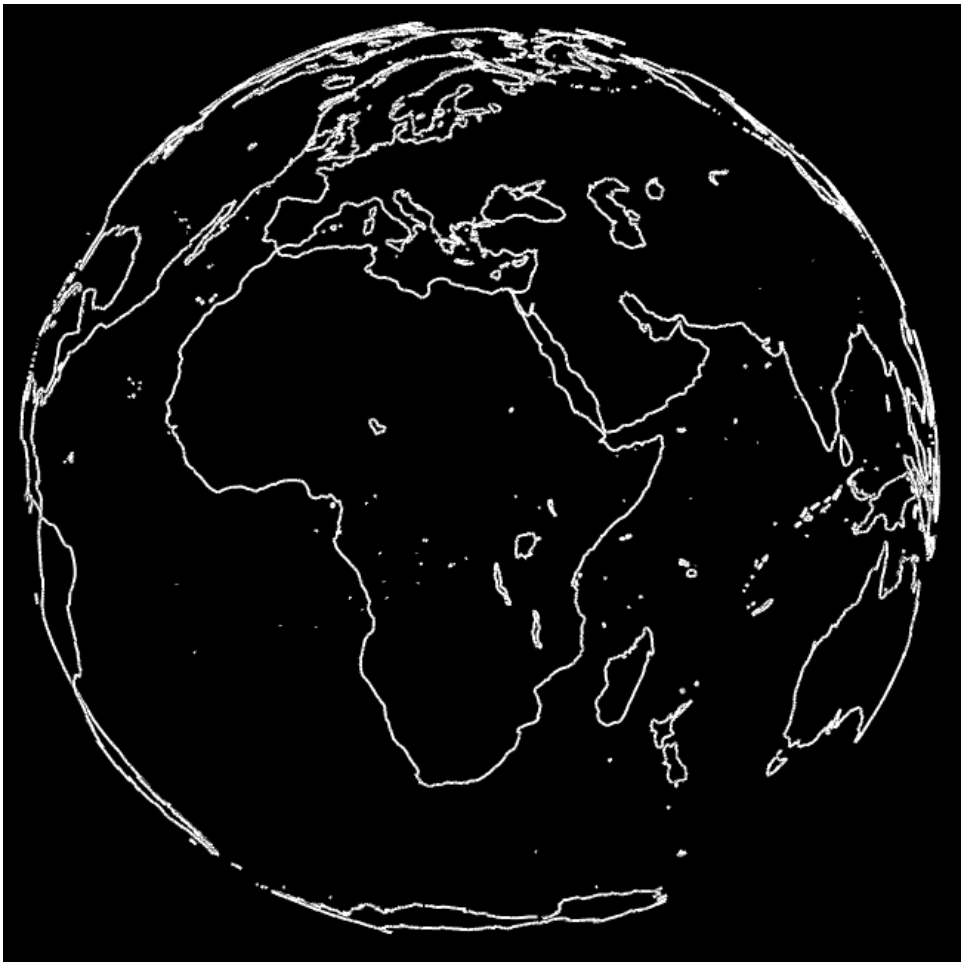


Abbildung 8: Darstellung der Küstenlinien (hier in weiß)

6.3 Temperaturen

Bei der Visualisierung der Temperatur müssen zunächst die in Kelvin gegebenen Werte auf Farben abgebildet werden. Die Temperaturen können aus den Grib-Files als double Werte ausgelesen werden, die im Intervall [MinTemp, MaxTemp] liegen, dessen Grenzen ebenfalls den Grib-Files zu entnehmen sind. Diese sollen nun auf ein Farbspektrum abgebildet werden, welches sich über Blau (sehr kalt), Cyan, Grün, Gelb und Rot (sehr warm) erstreckt. Dies kann erreicht werden, indem das gesamte Temperaturintervall in vier gleichgroße Intervalle unterteilt wird, von denen das Erste linear auf die Kante Blau-Cyan des RGB Würfels projiziert wird, das Zweite auf die Kante Cyan-Grün, das Dritte auf die Kante Grün-Gelb und das Letzte auf die Kante Gelb-Rot (vgl. Abb. 9).

Die resultierende Darstellung ist äquivalent zu einer, die sich ergäbe, wenn

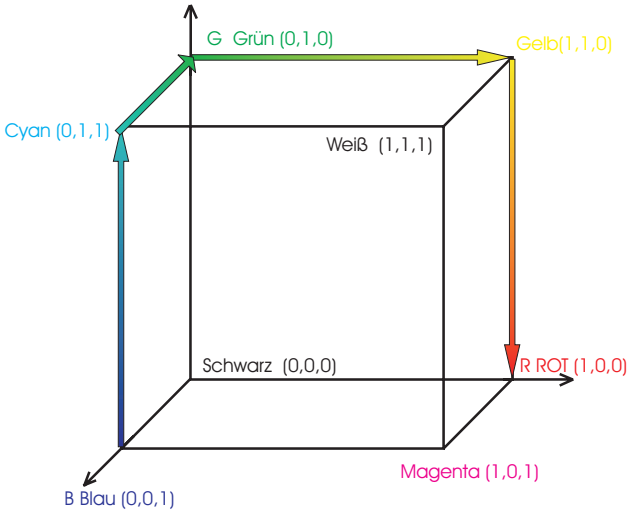


Abbildung 9: Das ursprünglich verwandte Farbspektrum im RGB-Würfel. Es erstreckt sich über Blau , Cyan, Grün, Gelb und Rot

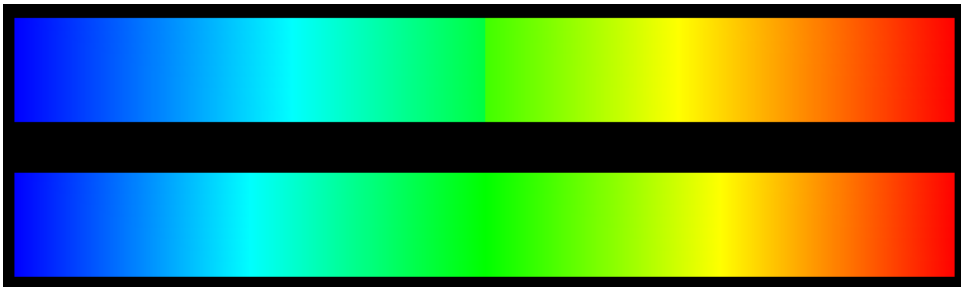


Abbildung 10: Vergleich der Spektren: Unten: Spektrum über Blau, Cyan, Grün, Gelb, Rot. Oben: Verbessertes Spektrum mit verringertem Grünteil.

man im HSV-Model über die entsprechenden Kanten der Pyramide ginge, allerdings mit dem Vorteil, dass der zusätzliche Rechenaufwand für das Umrechnen der HSV-Tripel in RGB-Werte entfällt. Das sich ergebende Farbspektrum ist im unteren Teil von Abb. 10 dargestellt.

Auffallend ist dabei der sehr dominante Grünbereich, der einen Großteil des Spektrums für sich beansprucht. Außerdem sind vor allem in der Mitte dieses Bereichs einzelne Grüntöne durch das Auge nicht zu unterscheiden. Die Ursache dafür ist zum Einen, dass die Farbe Grün mehr Leuchtkraft besitzt als Blau und Rot, zum Anderen kann das menschliche Auge bei Vorhandensein der grünen Farbkomponente andere Farbkomponeten nur begrenzt wahrnehmen. Als Folge dieser Probleme ergibt sich eine über weite Teile grün gefärbte Erdkugel, bei der außerdem in diesen Bereichen kaum Temperaturdifferenzen erkennbar sind.
Eine leichte Verbesserung ergibt sich durch *herausschneiden* eines Stückes

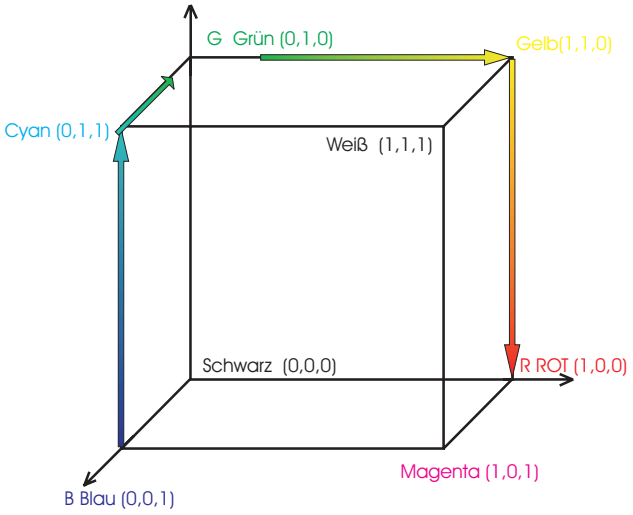


Abbildung 11: Das verbesserte Farbspektrum im RGB-Würfel mit verringertem Grünanteil

aus dem Farbverlauf, indem das zweite Temperaturintervall auf RGB-Tripel zwischen (0,1,1) und (0,1,0.25) und das Dritte auf Werte zwischen (0.25,1,0) und (1,1,0) abgebildet wird (vgl. Abb. 11).
Das sich ergebende Farbspektrum ist im oberen Teil von Abb. 10 dargestellt. Offensichtlich ist der Grünbereich weniger dominant und die Unstetigkeitsstelle im Farbverlauf kaum erkennbar. Folglich stellt dieses Spektrum zumindest eine kleine Verbesserung gegenüber dem Ersten dar.

Unabhängig davon ergibt sich noch ein anderes Problem: Soll nur ein Teil der dargestellten Werte genauer betrachtet werden, so wird für diesen nur ein kleiner Teil des gesamten Temperaturintervalls benötigt, folglich sind Temperaturdifferenzen wiederum nur schwer erkennbar. Deshalb sind zwei Regler implementiert worden, mit denen sich lokale Temperaturextrema einstellen lassen. Dann wird nur das zwischen den lokalen Extrema liegende Temperaturintervall auf die beschriebene Art auf RGB-Tripel abgebildet, während nicht enthaltene Werte einfach durch Dunkelblau (kälter) bzw. Dunkelrot (wärmer) dargestellt werden. Abbildung 12 zeigt Europa mit der unveränderten Darstellung, Abbildung 13 dagegen mit angepasstem Farbspektrum.

Wenn alle für das aktuelle Bild benötigten Werte als RGB-Tripel vorliegen, wird jedem Vertex ein Farbwert zugewiesen, diese werden mithilfe der Vertex-Pointer zu Flächen zusammengefasst und durch Smoothshading eingefärbt. Das Ergebnis ist in Abb. 14 zu sehen.

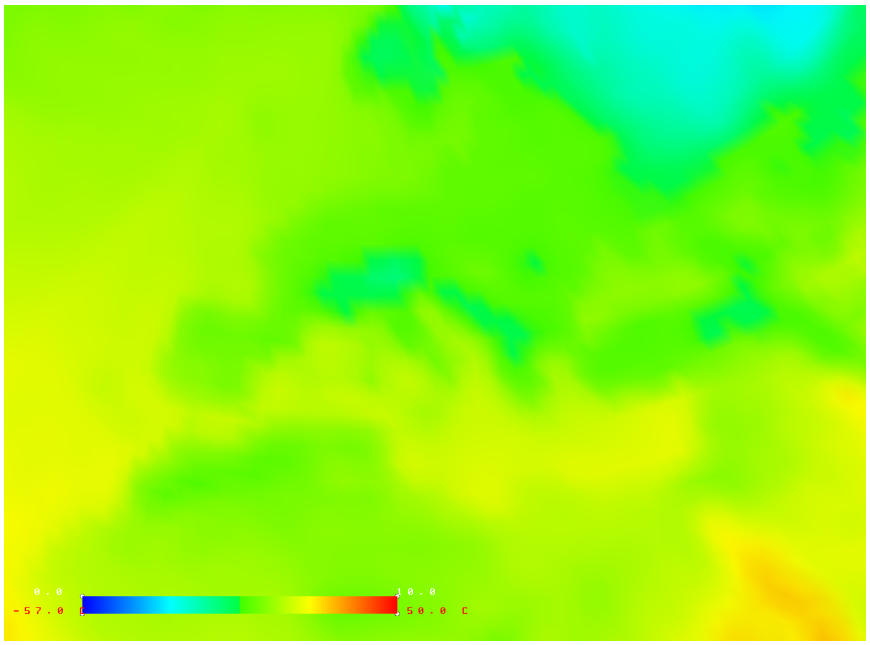


Abbildung 12: Dieses Bild zeigt Europa ohne Anpassungen des Farbspektrums. Die Regler befinden sich an den ursprünglichen Positionen

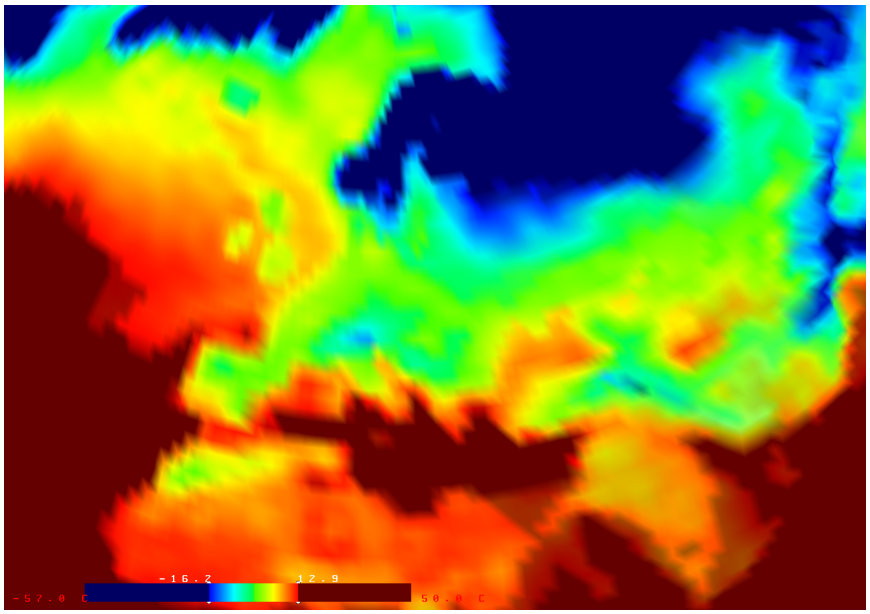


Abbildung 13: Dieses Bild zeigt Europa mit angepasstem Farbspektrum. Die Regler befinden sich deutlich innerhalb des Spektrums

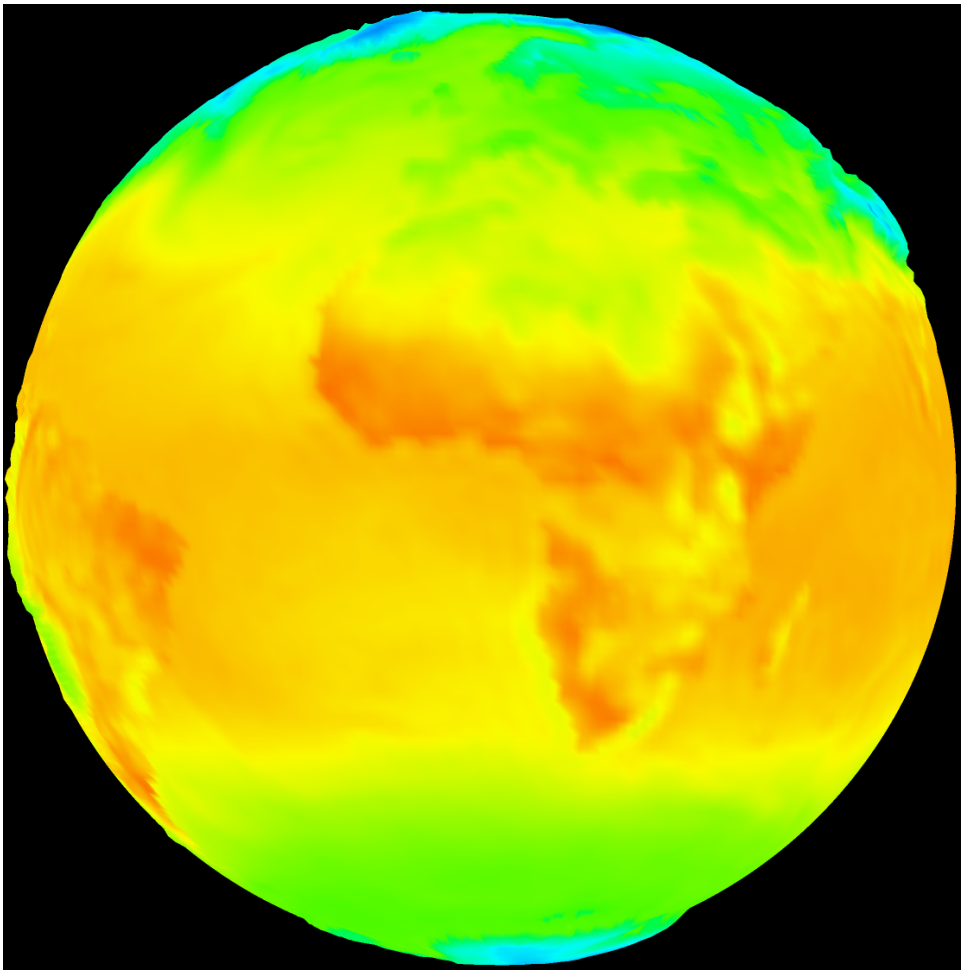


Abbildung 14: Darstellung der Temperaturen der Erdkugel in der hohen Auflösung

6.4 Bewölkung

Der Bewölkungsgrad wird durch eine Kugel visualisiert, deren Radius etwas größer als der der anderen Kugeln ist. Die zunächst naheliegendste Wolkendarstellungsart ist, den Vertices Grautöne zuzuordnen, deren Opacity proportional zu dem in Prozent gegebenen Bewölkungsgrad ist. Anschließend werden die zugehörigen Flächen analog zur Temperaturdarstellung durch Smoothshading eingefärbt. Das Resultat ist in Abb. 15 zu sehen.

Ein Vorteil dieser Methode ist der relativ geringe Hardwareaufwand, da keine zusätzlichen Berechnungen durchgeführt werden müssen. Außerdem stimmt die Darstellung bestmöglich mit den Daten überein, da diese durch nichts verfälscht werden und nichts hineininterpretiert wird, was die Daten

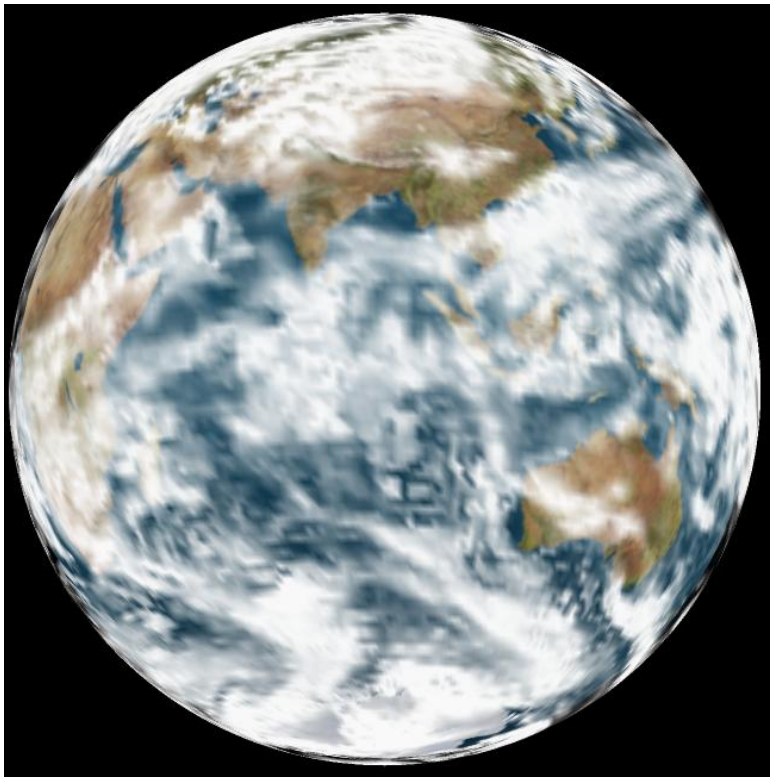


Abbildung 15: Darstellung der *einfachen* 2D-Wolken

nicht hergeben.
Trotzdem sollte nach Möglichkeiten gesucht werden, die Darstellung optisch ansprechender zu gestalten. Diese Versuche sollen in den folgenden Kapiteln aufgezeigt werden.

6.4.1 Texturierte 2D Wolken

Ein erster möglicher Ansatz, den optischen Eindruck der Bewölkung realistischer erscheinen zu lassen, ist der Folgende:
Es gibt eine unsichtbare Kugel, deren Vertices zum Bewölkungsgrad proportionale Opacitywerte erhalten. Diese Kugel fungiert als Alphamaske für drei weitere mit Wolkentexturen überzogene Kugeln. Das Resultat ist in Abb. 16 zu sehen.
Auch wenn dieser Weg die Bewölkung *wolkenähnlicher* erscheinen lässt, so können die teilweise gravierenden Nachteile nicht verleugnet werden:

- Die Daten werden stark verfälscht, da der Bewölkungseindruck nicht nur von den Opacitywerten, sondern auch von der örtlichen Texturbeschaffenheit abhängt

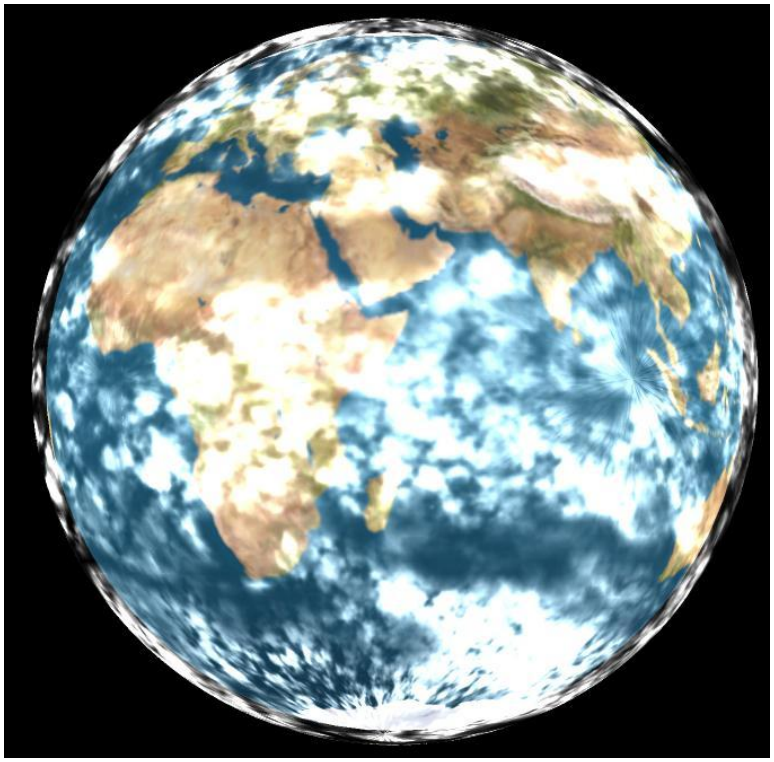


Abbildung 16: Darstellung der Wolken mit zusätzlichen Texturen

- Bei gleichzeitiger Darstellung mit Temperaturen kam es zu ungewollten Farbvermischungen (bunte Wolken)
- Die Pole der Texturkugeln sind deutlich erkennbar, da die Textur nicht dafür geeignet ist, auf eine Kugel projiziert zu werden. Dieser Effekt ist in Abb. 16 z.B. im Indischen Ozean westlich von Indonesien deutlich erkennbar.
- Etwas höherer Hardwareaufwand als bei den einfachen Wolken, da zusätzlicher Speicher für die Texturen benötigt wird und drei zusätzliche Kugeln gesetzt werden müssen. Diese sind allerdings statisch und tragen somit wenig zur Animationsgeschwindigkeit bei.

6.4.2 Bitmap Wölkchen

Ein weiterer Versuch, die Darstellung anprechender zu gestalten, war mittels kleiner Wölkchen, die zu größeren Formationen vereint dreidimensional wirken sollten. Diese wurden durch kleine Wölkchentexturen repräsentiert, die mithilfe einer Alphamaske in Randnähe transparent waren und somit *rund*

wirkten. Diese wurden anstelle der einzelnen Vertices der normalen Kugelgeometrie platziert und mit der Flächennormalen zum Betrachter gedreht. Gesetzt wurden sie nur bei von Null verschiedenen Bewölkungsgraden und dann mit einer zu diesem proportionalen Opacity.
Das Ziel, räumlich wirkende Wolken ohne die bei Polyedern unvermeidlichen *Ecken* zu erzeugen konnte zwar erreicht werden, jedoch ist ihr Einsatz aus den folgenden Gründen indiskutabel:

- Die Daten werden stark verfälscht, da der Bewölkungseindruck nicht nur von den Werten, sondern auch von der Anzahl der übereinanderliegenden Texturen abhängt. Diese ist am Rand der Kugel und insbesondere in Polnähe größer.
- Da im Schnitt deutlich mehr Vertices gesetzt werden, als bei jeder anderen Darstellungsart, leidet die Animationsgeschwindigkeit nicht unerheblich.
- Bei gleichzeitiger Darstellung mit Temperaturen kam es zu ungewollten Farbvermischungen (bunte Wolken)

Das Resultat ist in Abb. 17 zu sehen.

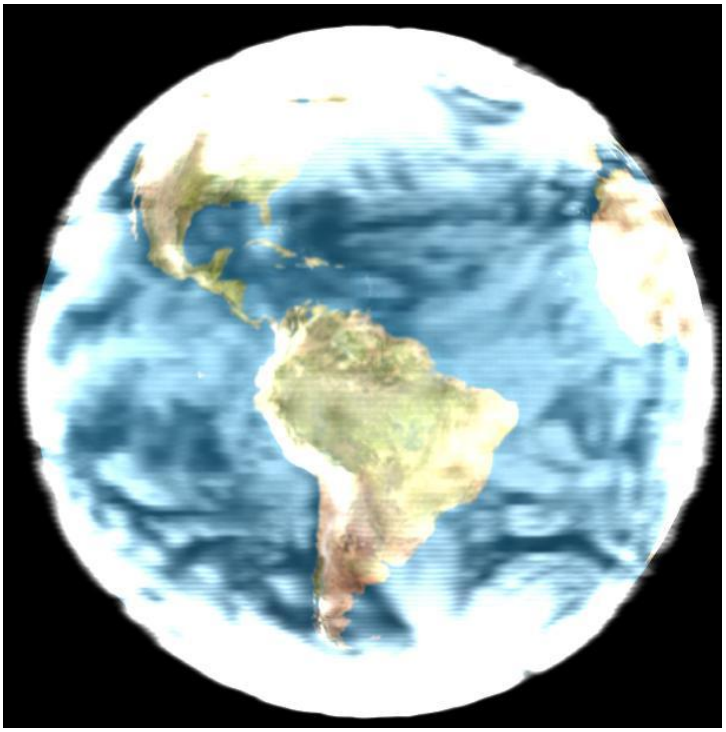


Abbildung 17: Bewölkungsdarstellung durch kleine Bitmap-Wölkchen

6.4.3 3D Wolken

Da sich die bisherigen Ansätze als unbefriedigend erwiesen, sind echte 3D-Wolken implementiert worden.
Wie bereits beschrieben ergeben sich die Ortsvektoren der Vertices in Kugelkoordinaten als Produkt aus dem Kugelradius und der zugehörigen Normalen. Dazu wird der aktuelle Radius eines Vertex als Funktion des ursprünglichen Kugelradius und seines Opacitywertes definiert. Für die Radien $Radius_i$ der Vertices $Vertex_i$ mit Opacity $Opacity_i$ und den Normalen $Normale_i$ gilt dann:

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

Dabei ist c eine Konstante, die so zu wählen ist, dass die Verschiebung klein gegenüber dem Kugelradius ist. Abb. 18 zeigt die 3D Wolkendarstellung.
Das Ergebnis ist deutlich ansehnlicher als jede andere der bisher getesteten Darstellungsarten. Außerdem kommt es aufgrund des einfachen Blendings (konstante Anzahl und Reihenfolge der Schichten) nicht zu unansehnlichen Farbvermischungen mit den Temperaturen. Des Weiteren ist der Hardwareaufwand nur vernachlässigbar größer als bei den einfachen 2D Wolken, da die Koordinaten jedes einzelnen Vertex zwar für jedes Bild neu zu berechnen sind, diese aber in jedem Fall neu gesetzt werden müssen. Und genau das Neusetzen der Eckpunkte ist, wie im Kapitel Performance noch genauer erläutert wird, der für die Animationsgeschwindigkeit entscheidende Faktor. Ein Nachteil im Vergleich zu den einfachen 2D-Wolken ist die dagegen ungenauere Darstellung der Daten. Schließlich ist in den Grib-Files nur der Bewölkungsgrad, nicht jedoch die Dicke bzw. Höhe der Wolkenschichten enthalten. Die Daten werden an dieser Stelle demnach etwas überinterpretiert. Da in dieser Arbeit jedoch schwerpunktmäßig die 3D-Darstellung behandelt wird, ist diese Wolkenart den anderen vorzuziehen.

6.5 Wind

Der Wind soll mithilfe von Pfeilen aus `GL_LINES` visualisiert werden, die in Windrichtung zeigen und deren Länge proportional zur Windstärke ist. Die Daten sind als 2D-Vektor gegeben, bestehend aus einer u- und einer getrennt einzulesenden v-Komponente. Da die Auflösung der Winddaten genauso hoch ist wie die der anderen Daten, wird sie zunächst auf ein sechzehntel reduziert, um die Pfeile nicht zu klein und die Animation nicht zu langsam werden zu lassen. Dabei werden jeweils sechzehn quadratisch angeordnete Werte zu einem neuen gemittelt, der sich in der Mitte dieses Quadrats befindet. Die resultierenden Windvektoren werden nun mit ihren Schwerpunkten auf diesen Punkten platziert und so skaliert, dass die Länge des längsten Pfeils genau

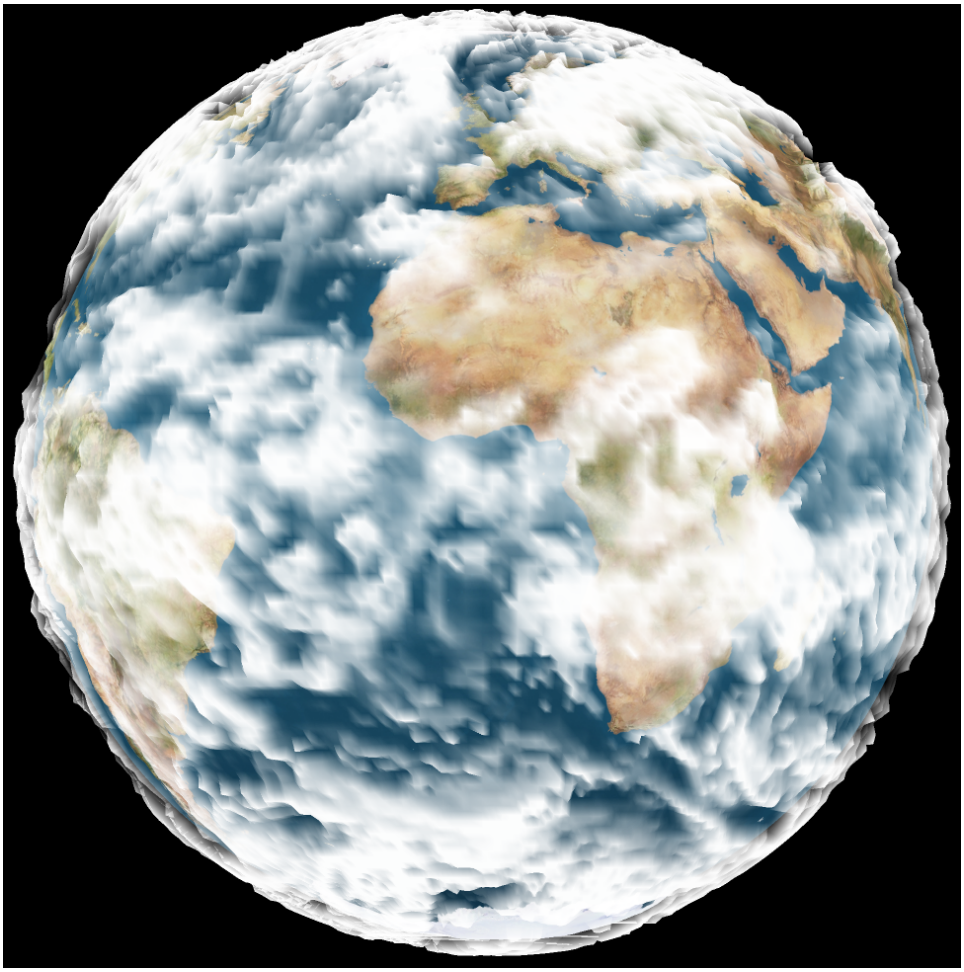


Abbildung 18: Darstellung der *echten* 3D-Wolken

der kleinsten Seitenlänge des jeweiligen Vierecks entspricht. Auf diese Art ist sichergestellt, dass kein Pfeil sein Quadrat verlässt. Abb. 19 veranschaulicht die Bestimmung eines Windvektors und Abb. 20 zeigt die resultierende Darstellung.

Diese Darstellung ist nur dann wirklich exakt, wenn die Oberfläche des Körpers in gleichgroße Quadrate unterteilbar ist. Da dies bei einer Kugel unmöglich ist, kann es nur eine Näherungslösung geben. Die beschriebene Lösung ist besonders ungeeignet, weil nur in Äquatornähe Quadrate entstehen, in Polnähe werden diese aufgrund der steigenden Konzentration der Vertices Rechtecken immer ähnlicher. Die Folge ist zum einen eine Verfälschung der Windstärke, zum anderen wird die Windrichtung ungenügend wiedergegeben, da die Höhe der Rechtecke zwar korrekt bleibt, die Breite jedoch in Polnähe immer weiter abnimmt. Entsprechend entsteht auf diese

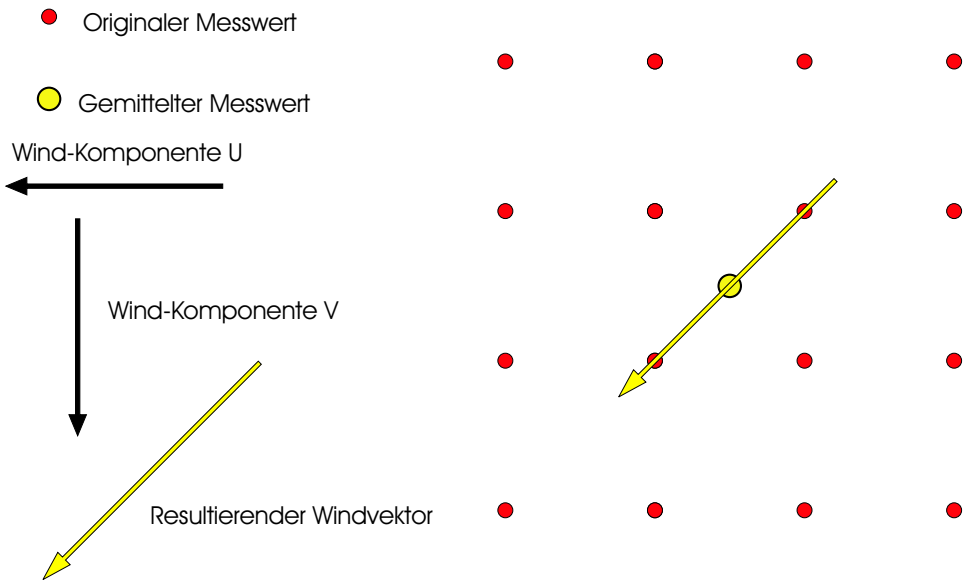


Abbildung 19: Skizze zur Veranschaulichung der Berechnung der Windrichtung

Weise der Eindruck, als würde der Wind im Polbereich fast immer zu diesem hin oder davon weg zeigen, wie in Abb. 20 erkennbar ist.

Ziel der Versuche die Winddarstellung zu verbessern muss demnach sein, die Erdkugel mit *Quadraten* zu überziehen, deren Breite in Abhängigkeit des jeweiligen Längengrads etwa konstant ist. Der *Radius* eines Längengrades in Abhängigkeit des Winkels Theta ist gegeben als:

$$RadiusLG\,(Theta) = 2 \cdot \pi \cdot Radius_{Kugel} \cdot \sin\,(Theta)$$

Daher liegt nahe, die Vertexzahl in Abhängigkeit des Winkels Theta zu definieren als:

$$VertexZahl\,(Theta) = VertexZahl_{max} \cdot \sin\,(Theta)$$

Leider muss die Vertexzahl natürlich noch auf einen ganzzahligen Wert gerundet werden, wodurch es aufgrund deren geringer Anzahl in Polnähe zu Ungenauigkeiten kommt. Außerdem muss für je zwei Längengrade die gleiche Vertexzahl erzwungen werden, da sich sonst alles verschieben würde und somit keine geeigneten Vierecke erzeugbar wären. Abb. 21 zeigt oben in der zuerst beschriebenen Darstellung die Vierecke (blau), in denen sich die Pfeile befinden und Windstärken, deren u- und v-Komponente überall knapp unter der maximalen Windgeschwindigkeit liegen.

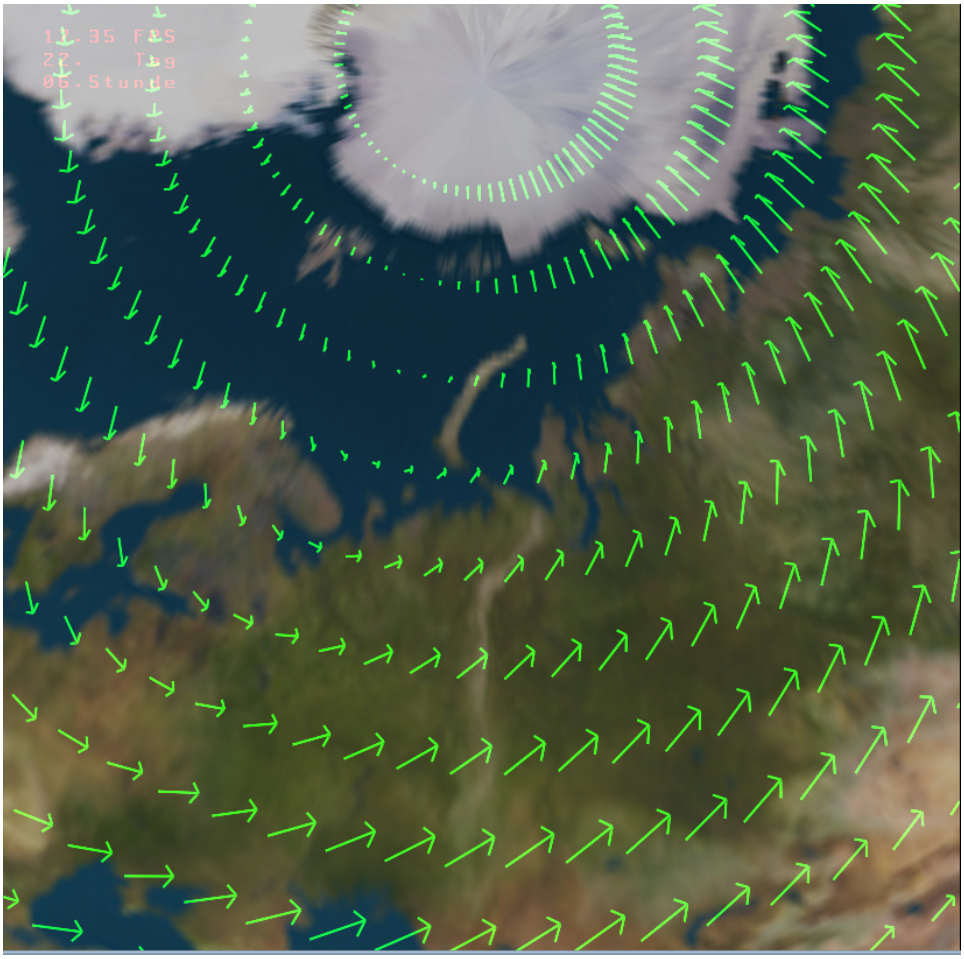


Abbildung 20: Die ursprüngliche Darstellung der Windrichtung

Im Vergleich dazu zeigt Abb. 21 unten den gleichen Zustand mit der verbesserten Darstellungsmethode. Es zeigt sich deutlich, dass die Längenunterschiede der Pfeile hier weitaus geringer ausfallen und auch ihre Ausrichtung nach Nordwesten überall gut getroffen ist. Die Darstellung kommt dem Idealergebniss recht nahe, lediglich in direkter Polnähe sind die Pfeile etwas länger als die Übrigen. Dies ist auch an den Vierecken gut erkennbar, die in Polnähe wieder Rechtecken ähnlich sind und somit Windrichtung und Windstärke nicht optimal, aber um Welten besser als das erste Verfahren wiedergeben können.
Unter Performancegesichtspunkten kann außerdem festgehalten werden, dass mit dieser Methode aufgrund der verringerten Vertexzahl in Polnähe die Animationsgeschwindigkeit deutlich höher ist. Abb. 22 zeigt das nun erheblich exaktere, ansehnlichere und performantere Ergebnis.

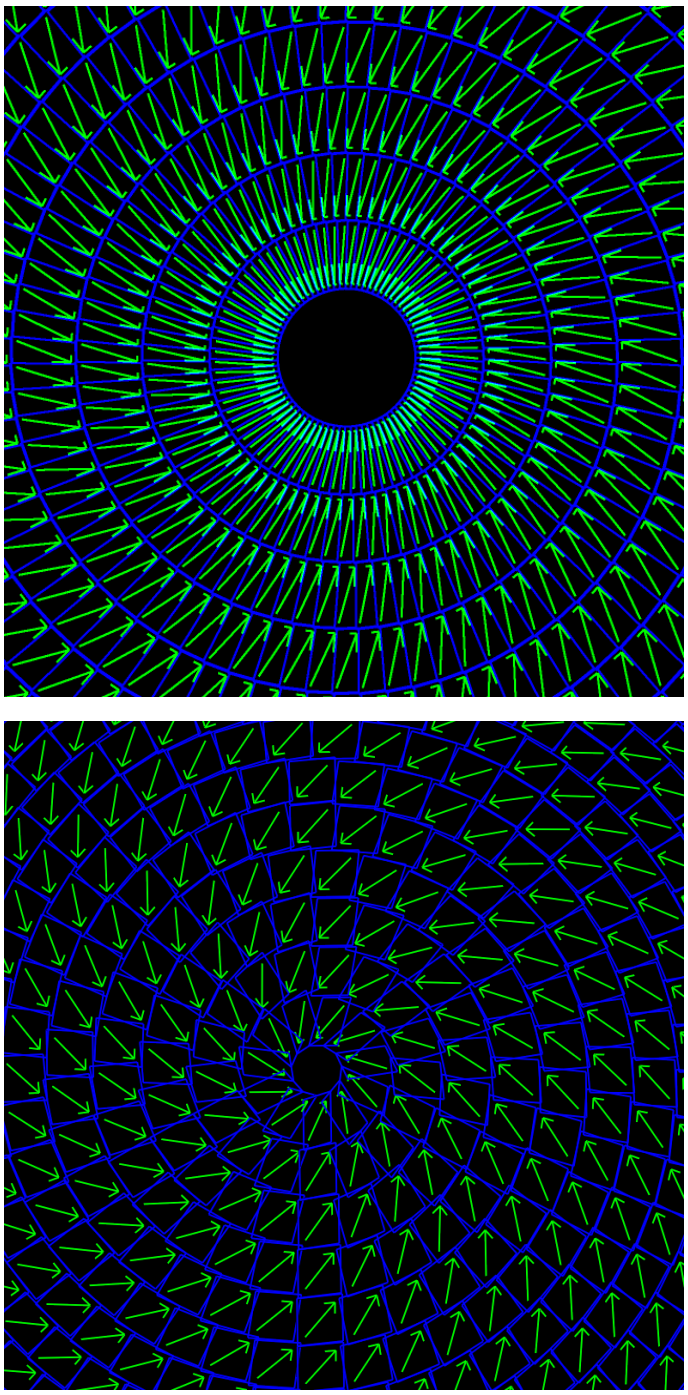


Abbildung 21: Skizze zur alten Einteilung der Kugel in Vierecke (oben) im Vergleich mit der verbesserten Methode (unten)

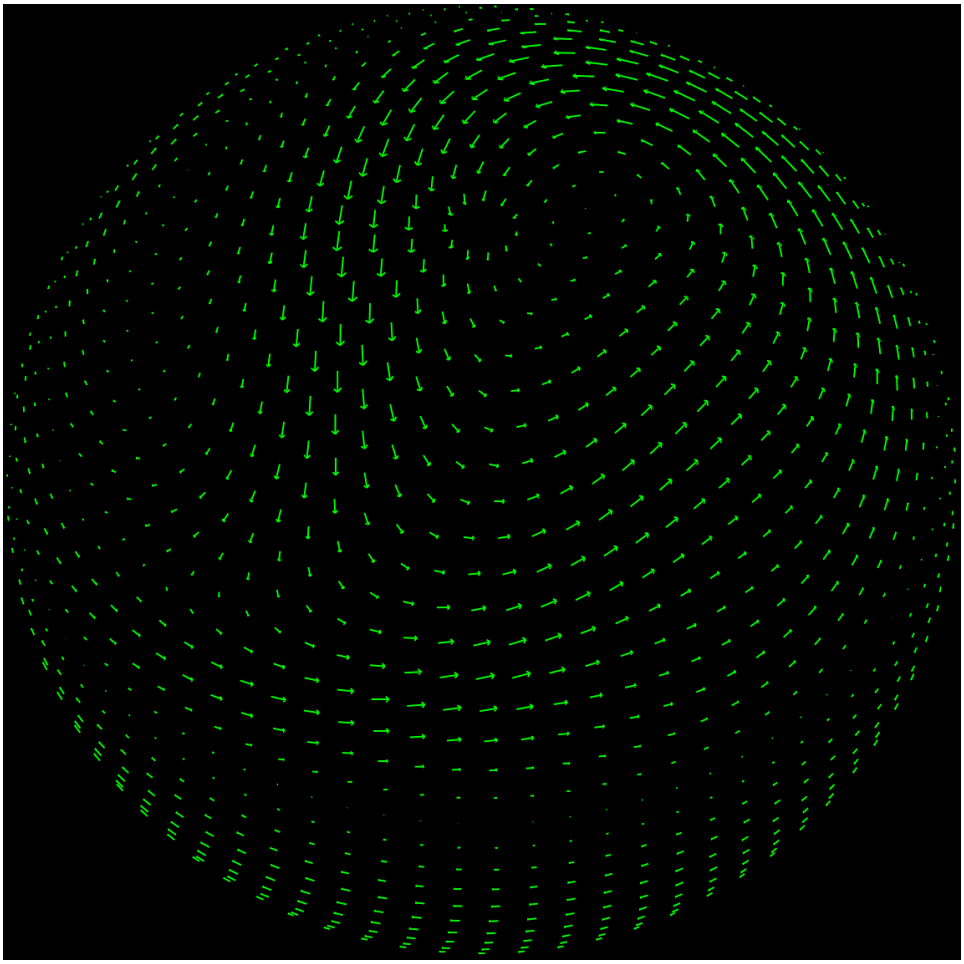


Abbildung 22: Die finale Winddarstellung

6.6 Visualisierung von Erdausschnitten

Das Programm kann wie bereits erwähnt nicht nur die gesamte Erdkugel, sondern bei entsprechenden Daten auch Ausschnitte daraus darstellen. Hierbei wird auch nur der benötigte Teil der Texturkugel dargestellt. Abb. 23 zeigt Europa in einer Auflösung von $325 \cdot 325$ Messpunkten. Da für jeden dieser Messwerte ein Vertex gesetzt wird, läuft die Animation nur sehr langsam ab.

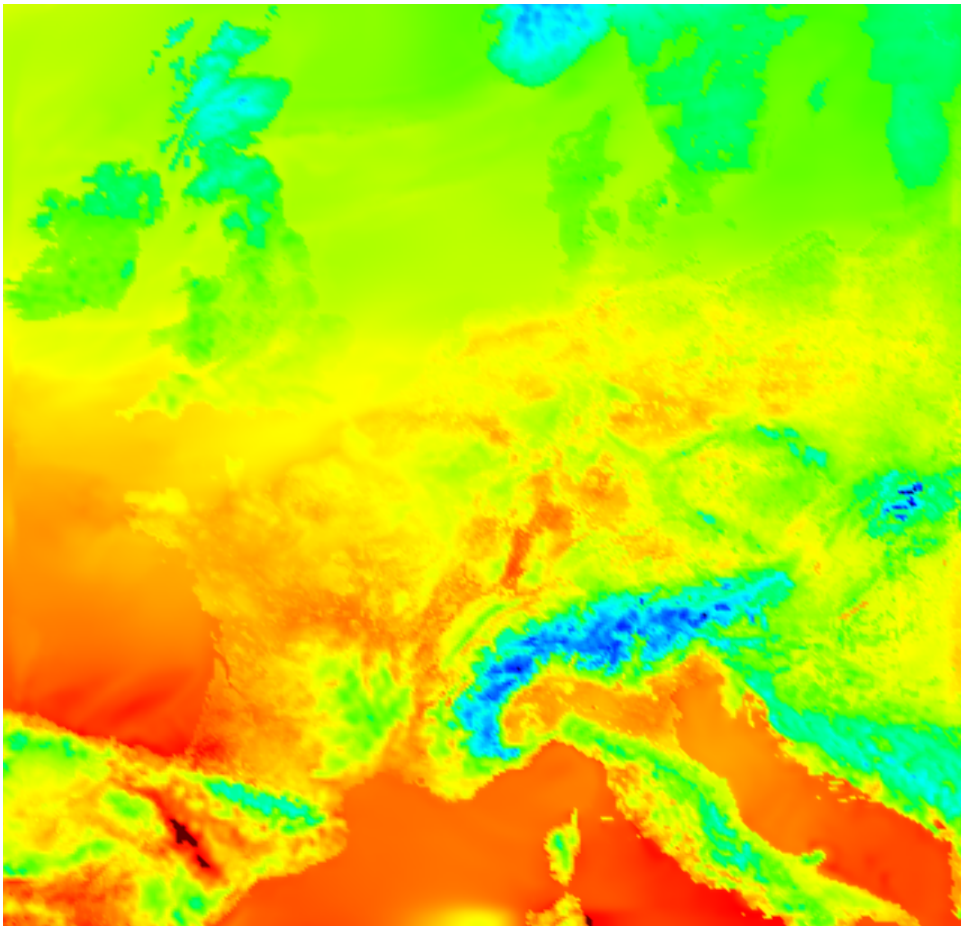


Abbildung 23: Darstellung der Temperaturen in Europa

6.7 On-Screen-Display

Das On-Screen-Display wird benötigt, um den Benutzer mit Informationen über die aktuell dargestellten Daten zu versorgen. Diese sind:

- Referenzwerte zum Einschätzen der dargestellten Daten:
  - Pfeile, die die maximale Windstärke angeben, sowie die zugehörige Windgeschwindigkeit
  - Das Temperaturspektrum und die Werte zwischen denen es verläuft
  - Das Bewölkungsspektrum und die Werte zwischen denen es verläuft
- Das Datum und die Tageszeit des dargestellten Bildes

- Die Animationsgeschwindigkeit in Bildern pro Sekunde (FPS)
- Die Regler zum Setzen der lokalen Temperaturextrema

Es ist nicht möglich zweidimensionale Objekte derart in ein OpenGL-Fenster einzufügen, dass sie unabhängig von der Kameraposition und der Zoomstufe mit konstanter Größe, Ausrichtung und Anordnung auf dem Bildschirm erscheinen (vgl. Abb, 24). Deshalb müssen diese Elemente so in der Szene platziert, gedreht und skaliert werden, dass sie zweidimensional wirken. Dazu wird eine zum Betrachter senkrechte Ebene benötigt, die von einem horizontalen und einem vertikalen Vektor aufgespannt wird. Für den vertikalen Vektor kann der View-Up-Vektor verwendet werden, der Horizontale ergibt sich aus dem Kreuzprodukt dieses Vektors mit dem Ortsvektor der Kamera, der zugleich die Blickrichtung angibt. Für den die Ebene festlegenden Punkt kann ein Punkt der Strecke zwischen Kamera und Erdmitte gewählt werden, der jedoch außerhalb des Erdradius liegen muss.
Ändert die Kamera ihre Position, muss die Ebene neu bestimmt werden, ändert sich der Zoomfaktor, müssen die die Ebene aufspannenden Vektoren ihre Länge anpassen, da das On-Screen-Display unabhängig vom Zoomfaktor sein soll.

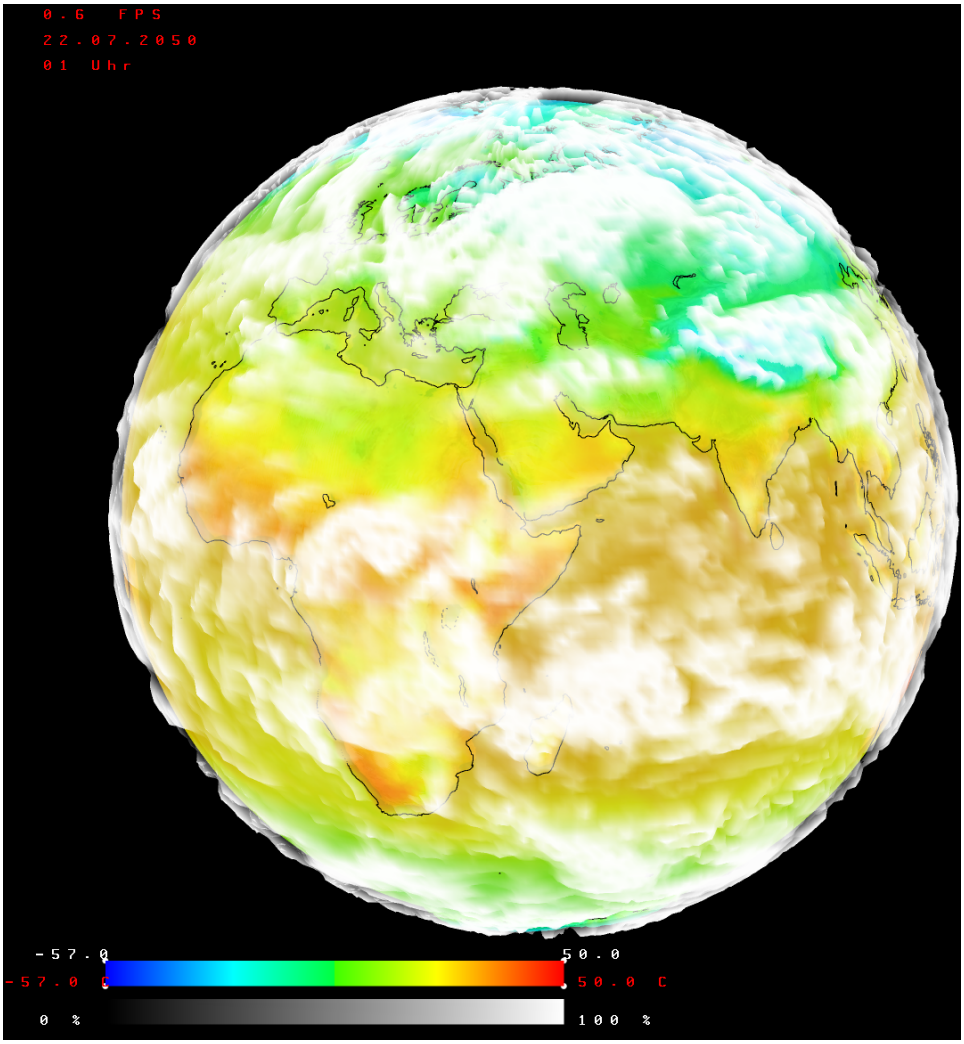


Abbildung 24: Darstellung des On-Screen-Displays, welches den Benutzer mit Informationen über die dargestellten Daten versorgt.

7 Animation

7.1 Laden aller Records beim Programmstart

Der erste Ansatz war, zunächst alle darzustellenden Records einzulesen und für die Animation zusätzliche Zwischenbilder zu berechnen.
Dazu wird jeweils aus dem aktuellen und dem nächsten Record mittels linearer Interpolation ein gewichteter Mittelwert gebildet:

$$\frac{(schrittZahl - aktuellerSchritt) \cdot aktuelleFarbe + aktuellerSchritt \cdot naechsteFarbe}{schrittZahl}$$

Vorteil des Ladens der Records am Anfang ist die hohe und gleichmäßige Animationsgeschwindigkeit. Nachteil ist neben der langen Ladezeit die beschränkte anzeigbare Recordmenge.

7.2 Dynamisches Nachladen der Records zur Laufzeit

Dieser Ansatz löst die Speicherprobleme des ersten Ansatzes, bewirkt jedoch eine geringere Animationsgeschwindigkeit. Es sind zwei verschiedene Lademechanismen ausprobiert worden:

7.2.1 Doublebuffer

Hier werden jeweils die *aktuellen* und die *nächsten Records* geladen. Dann berechnet das Programm auf die beschriebene Art Zwischenbilder, bis die *nächsten Records* angezeigt werden. Jetzt müssen neue *nächste Records* geladen werden und die bisherigen *nächsten Records* werden zu den *Aktuellen.*
Nachteil: da alle *nächsten Records* (WindU, WindV, Wolken, Temperaturen, ...) gleichzeitig geladen werden müssen, kommt es regelmäßig zu Aussetzern und die Animation läuft ungleichmäßig schnell ab.

7.2.2 Triplebuffer

Mithilfe des Triplebuffers kann auch beim dynamischen Nachladen eine gleichmäßige Animationsgeschwindigkeit erzielt werden, indem diese wieder von den *Aktuellen* zu den *nächsten Records* läuft und während dieser Zeit die *übernächsten Records* vorgeladen werden.
Dazu kommen zwei Mechanismen in Frage:

- **Laden kompletter Records**
  Bei diesem Lademechanismus wird das Einlesen der einzelnen Records in Abhängigkeit der Zwischenbilderzahl möglichst gleichmäßig verteilt. So könnte beispielsweise bei vier Zwischenbildern vor dem ersten die u-Komponente des Windes, vor dem zweiten die v-Komponente des

Windes, vor dem dritten die Temperatur und vor dem letzten Zwischenbild die Bewölkung geladen werden. In diesem Idealfall (Anzahl der Zwischenbilder = Recordzahl) läuft die Animation beinahe gleichmäßig ab, da vor jedem Bild ein Record geladen wird. Aber nur fast, da die Ladezeit einzelner Records nicht zwangsläufig identisch ist, etwa aufgrund unterschiedlicher Auflösung, oder der Geschwindigkeit der Umrechnung der Character-Arrays in Zahlentypen (vgl. Daten).
Ist dagegen die Zwischenbilderzahl nicht mit der Recordzahl identisch, läuft die Animation im Allgemeinen zwar gleichmäßiger ab als beim Doublebuffer, jedoch nicht vollständig flüssig. Ein weiteres Problem ist, ein allgemeines Verfahren zu finden, welches bei beliebiger Record- und Zwischenbilderzahl die jeweils ideale Ladeverteilung bestimmt. Der vermutlich beste Weg wäre die explizite Angabe aller denkbarer Kombinationen, da eine allgemeine Lösung wahrscheinlich nicht möglich ist.
Des Weiteren bewirkt dieser Triplebuffermechanismus im Spezialfall der Darstellung eines einzelnen Recordtyps keine Verbesserung gegenüber dem Doublebuffer, da ein Aufteilen des Ladevorgangs nicht möglich ist. Weil in der Regel nicht mehrere Recordtypen zur gleichen Zeit dargestellt werden, kann dieser Spezialfall als Normalfall angesehen werden und stellt somit den Hauptnachteil dieser Methode dar.

- **Teilweises Laden der Records**
  Bei diesem Lademechanismus wird von jedem Record bei n Zwischenbildern vor jedem einzelnen Schritt ein n-tel geladen, sodass alle benötigten Werte rechtzeitig verfügbar sind.
  Da die Zahl der Zwischenbilder sehr klein gegenüber der Zahl der in den Records enthaltenen Messwerte ist, sind diese nahezu ganzzahlig durch die Zahl der Schritte teilbar, sodass der Ladevorgang praktisch vollständig gleichmäßig ablaufen kann, unabhängig von:

  – Zahl der unterschiedlichen Records
  – Zahl der Zwischenbilder
  – Verhältnis von Recordzahl und Zwischenschrittzahl
  – Art der kombinierten Records:
    etwa hochaufgelöster Wind mit gering aufgelösten Wolken

  Da dieser Triplebuffer immer eine maximal flüssige Animation ermöglicht, ist er dem zuvor Beschriebenen vorzuziehen und wurde konsequenterweise im Programm implementiert.
  Falls die Daten jedoch durch ein externes Programm (GribReader) eingelesen werden sollen, so muss dieser Triplebuffer dort implementiert werden. Wenn das nicht möglich ist, bleibt als Alternative die Verwendung des ersten Triplebuffers mit den zuvor beschriebenen Nachteilen.

8 Bedienung

Die Steuerung des Programms erfolgt über eine Maus und Tastatur Kombination.

- **Maussteuerung**
  Mit der Maus kann die Szene gedreht werden, indem die linke Maustaste gedrückt, die Maus mit gedrückter Taste bewegt und die Taste schließlich wieder losgelassen wird. Außerdem können einige Elemente des On-Screen-Displays angeklickt werden. Die weißen Zahlen in der Abb. 25 markieren diese. Die Erklärungen dazu im einzelnen:

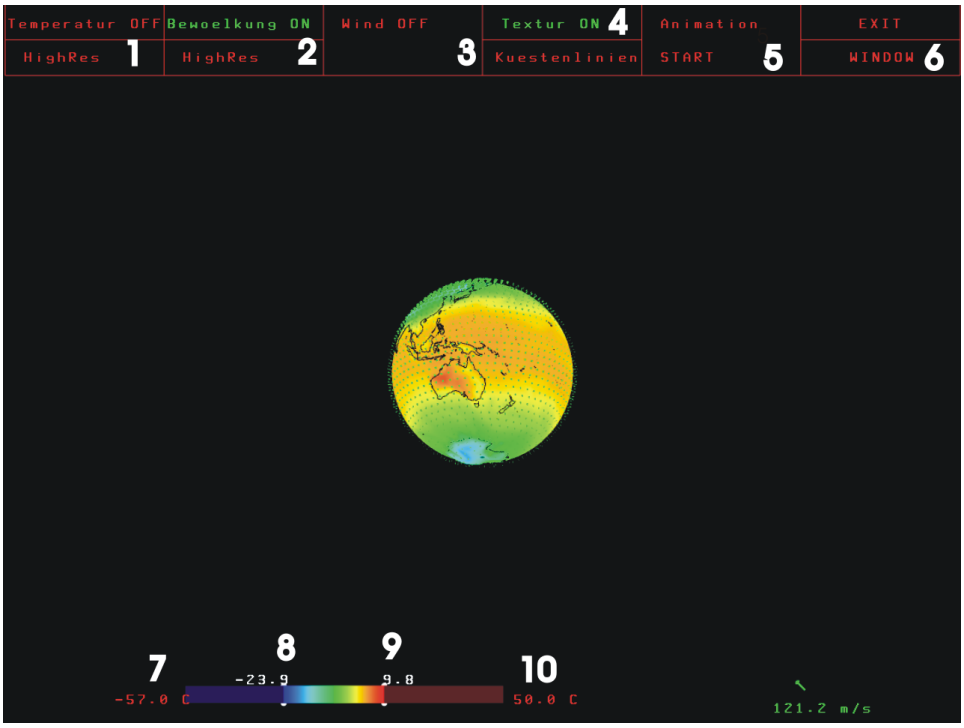


Abbildung 25: Das Menü mit den Bedienelementen des Programms

1. Temperatur
   Der obere Button dient zum Aktivieren/Deaktivieren der Temperaturanzeige und der untere ändert die Auflösung der Temperaturen
2. Bewölkung
   Der obere Button dient zum Aktivieren/Deaktivieren der Bewölkungsanzeige und der untere ändert die Auflösung der Wolkendarstellung
3. Wind
   Hier kann die Windanzeige aktiviert/deaktiviert werden

4. Geometrie
   Der obere Button dient zum aktivieren/deaktivieren der Erdtextur und der untere aktiviert/deaktiviert die Küstenlinien
5. Animation
   Hier kann die Animation gestartet oder angehalten werden
6. Beenden
   Die Schaltfläche `Exit` beendet das Programm.
7. Die minimale Temperatur
8. Die aktuelle minimale Temperatur
   Der Marker für das aktuelle Minimum kann mit gedrückter linker Maustaste verschoben werden. Die Zahl gibt Auskunft über den aktuellen Stand und das Spektrum passt sich an.
9. Die aktuelle maximale Temperatur
   Der Marker für das aktuelle Maximum kann mit gedrückter rechter Maustaste verschoben werden.
10. Die maximale Temperatur

Die Bildelemente sind nur verfügbar, wenn die entsprechenden Daten auch geladen wurden. Die Informationen über die Daten werden nur angezeigt, wenn die Daten dargestellt werden. Aus diesem Grund fehlt in Abb. 25 das Wolkenspektrum.

- **Tastatursteuerung**
  Zusätzlich können die folgenden Operationen auf Tastendruck ausgeführt werden:

  - Kamera nach links drehen: `a`
  - Kamera nach rechts drehen: `d`
  - Hineinzoomen: `w`
  - Herauszoomen: `s`
  - Kamera nach unten bewegen: `LShift`
  - Kamera nach oben bewegen: `Space`
  - Animationsgeschwindigkeit erhöhen: `+`
  - Animationsgeschwindigkeit verringern: `-`

9 Performance

9.1 Optimieren des Programms

Die Animationsgeschwindigkeit nimmt neben der Bildqualität den höchsten Stellenwert ein. Dem Umstand Rechnung tragend, dass zur Laufzeit eine größere Menge interner Berechnungen durchgeführt werden muss, wurde das Programm in C++ geschrieben. Um die Performance maximieren zu können, ist zunächst zu bestimmen, wovon diese in erster Linie abhängt. Mögliche Performancefaktoren sind:

1. Zeichenleistung der Grafikkarte, beansprucht von:
   - Art des Shadings (Smooth vs. Flat)
   - Auflösung des Fensters
   - Anti-Aliasing (AA)
   - Anisotropischer Filter (AF)
2. Geometrie
   - Anzahl der Vertices
   - Anzahl der neu gesetzten Vertices pro Bild
   - Art der Flächen zu denen die Vertices zusammengesetzt werden (Triangle- oder Quadstrips)
3. Interne Berechnungen des Programms
   - Einlesen der Daten von der Festplatte
   - Umrechnen der in Character-Arrays kodierten Werte in Zahlentypen
   - Zeitliches Interpolieren der Messwerte (Zwischenbilder)
   - Räumliches Interpolieren der Messwerte (verringerte Auflösung)
   - Berechnen der variablen Wolkengeometrie

Diese Punkte sollen im Folgenden genauer untersucht werden, indem zunächst Beobachtungen gesammelt werden, wie sich die Animationsgeschwindigkeit bei Veränderungen der obigen Performancefaktoren verhält.
In Bezug auf die Abhängigkeit der Animationsgeschwindigkeit von der Zeichenleistung der Grafikkarte ist der Tabelle 2 zu entnehmen, dass diese erst bei geringer Szenenkomplexität eine Rolle spielt.

Ein weiterer möglicher performancekritischer Punkt ist die Höhe der Geometrieauflösung der Szene. Tabelle 3 kann entnommen werden, dass das Hinzuschalten der Erdtextur und der Küstenlinien, die beide über eine größere

	hohe Bildqualität	geringe Bildqualität
Wolken und Temperaturen in 320 x 160, mit Erdtextur und Küstenlinien	7,0 FPS	7,1 FPS
Wolken in 320 x 160	18.5 FPS	19.2 FPS
Wolken in 160 x 80	35,0 FPS	54,0 FPS

Tabelle 2: Vergleich der Zeichenleistungsabhängigkeit der Animationsgeschwindigkeit.
Hohe Qualität: Auflösung 1600x1200, Smooth-Shading, 4xAA, 8xAF
Geringe Qualität: Auflösung 800x600, Flat-Shading, kein AA, kein AF

Anzahl Vertices definiert werden als die dort dargestellte Szene, die Animationsgeschwindigkeit nicht merklich beeinflusst. Wird die Szene durch eine explizit gesetzte Clipplane so geclippt, dass nur ca. ein Viertel sichtbar ist, so hat dies keinen Einfluss auf die Framerate. Wenn stattdessen nur ein Viertel der Vertices gesetzt wird, steigt die Animationsgeschwindigkeit erheblich an. Dabei werden weiterhin alle Daten eingelesen und alle internen Berechnungen des Programms durchgeführt, es fehlt lediglich der Aufruf zum Setzen der fehlenden Vertices. Unwesentlich ist dagegen, ob die Vertices zu Quadstrips oder Trianglestrips zusammengesetzt werden, da diese intern höchstwahrscheinlich in Trianglestrips umgewandelt werden. Dadurch ergibt sich in beiden Fällen exakt die gleiche Performance und mit Smooth-Shading auch ein identischer optischer Eindruck.

Einstellung	FPS	Relativer Unterschied
Wolken in 320 x 160	19,3	0 %
Erdtextur zugeschaltet	19,0	-1,6 %
Küstenlinien zugeschaltet	19,1	-1 %
Erdtextur und Küstenlinien zugeschaltet	18,7	-3.1 %
Wolken in 320 x 40 (Rest nicht gesetzt)	44,5	+131 %
Wolken in 160 x 80 (runtergerechnet)	44,0	+128 %
Manuelles Clipping deaktiviert	11,7	-39,4 %

Tabelle 3: Vergleich zur Abhängigkeit der Animationsgeschwindigkeit von der Geometrie. Die relativen Unterschiede beziehen sich auf die erste Einstellung.
Auflösung: 1152x768, 4xAA. System: Athlon XP 2400 , GeForce4 Ti 4400

In Bezug auf das dynamische Nachladen der Daten zur Laufzeit kann der Tabelle 4 entnommen werden, dass die Animationsgeschwindigkeit sich zwar bei im Voraus geladenen Daten verbessern würde, die Unterschiede

sich aber im Allgemeinen in Grenzen halten. An dieser Stelle kann außerdem festgehalten werden, dass die speziellen Methoden zum Umrechnen der Character-Arrays deutlich schneller arbeiten als die allgemeingültige Methode `atof`.

Einstellung	FPS	Relativer Unterschied
Dynamisches Nachladen der Daten eingeschaltet	19,3	0 %
Dynamisches Nachladen der Daten ausgeschaltet	21,4	+10,9 %
Verwendung von atof zum Umwandeln der Characterarrays	15,3	-20,7 %

Tabelle 4: Abhängigkeit der Animationsgeschwindigkeit vom Nachladen der Daten: Dargestellt werden jeweils Wolken der Auflösung 320x160. Die relativen Unterschiede beziehen sich auf diese Einstellung. System: Athlon XP 2400 , GeForce4 Ti 4400

Die Beobachtungen zeigen, dass die Animationsgeschwindigkeit in erster Linie von der Zahl der für jedes Bild neu zu setzenden Eckpunkte abhängt. Unwesentlich ist dagegen die Geometrieauflösung der statischen Objekte, da diese sich fest im Speicher der Grafikkarte befinden, sobald sie einmal als Listen definiert wurden. Gerade bei großen Zahlen neu zu setzender Vertices spielt die Zeichenleistung aktueller Grafikkarten keine Rolle, da sie aufgrund der lange dauernden Geometrieberechnungen genügend Zeit zum Rendern haben.
Die Animationsgeschwindigkeit des Programms hängt in erster Linie von der Anzahl der neu gesetzten Vertices pro Bild ab. Da sich die meisten Werte von einem Bild zum nächsten ändern und es aufgrund der Vertexpointertechnik schwer möglich ist, einzelne Dreiecke neu zu setzen, während andere erhalten bleiben, werden alle Vertices, die Messwerte repräsentieren, neu gesetzt.
Ziel ist daher, die Zahl der zu setzenden Vertices zu verringern, ohne die Darstellungsqualität zu sehr in Mitleidenschaft zu ziehen.

9.1.1 Verringerung der Geometrieauflösung

Um die Zahl der Vertices zu verringern, ist es möglich, je zwei Werte einer Zeile und Spalte zu mitteln. Die nur noch durch ein Viertel der Eckpunkte repräsentierte Darstellung der Temperaturen ist oben in Abb. 26 zu sehen. Im Vergleich zur Originalauflösung (Abb. 26 unten) ist sie unschärfer und die Wiedergabe der Messwerte folglich ungenauer.

Bei der Wolkendarstellung ist der Unterschied zwischen der auf ein Viertel reduzierten Auflösung (vgl. Abb. 27 links) und dem Original (vgl. Abb. 27 rechts) noch größer, da der Bewölkungsgrad selbst über kleinste räumliche Abstände stark variieren kann und somit mehr Details verloren gehen.

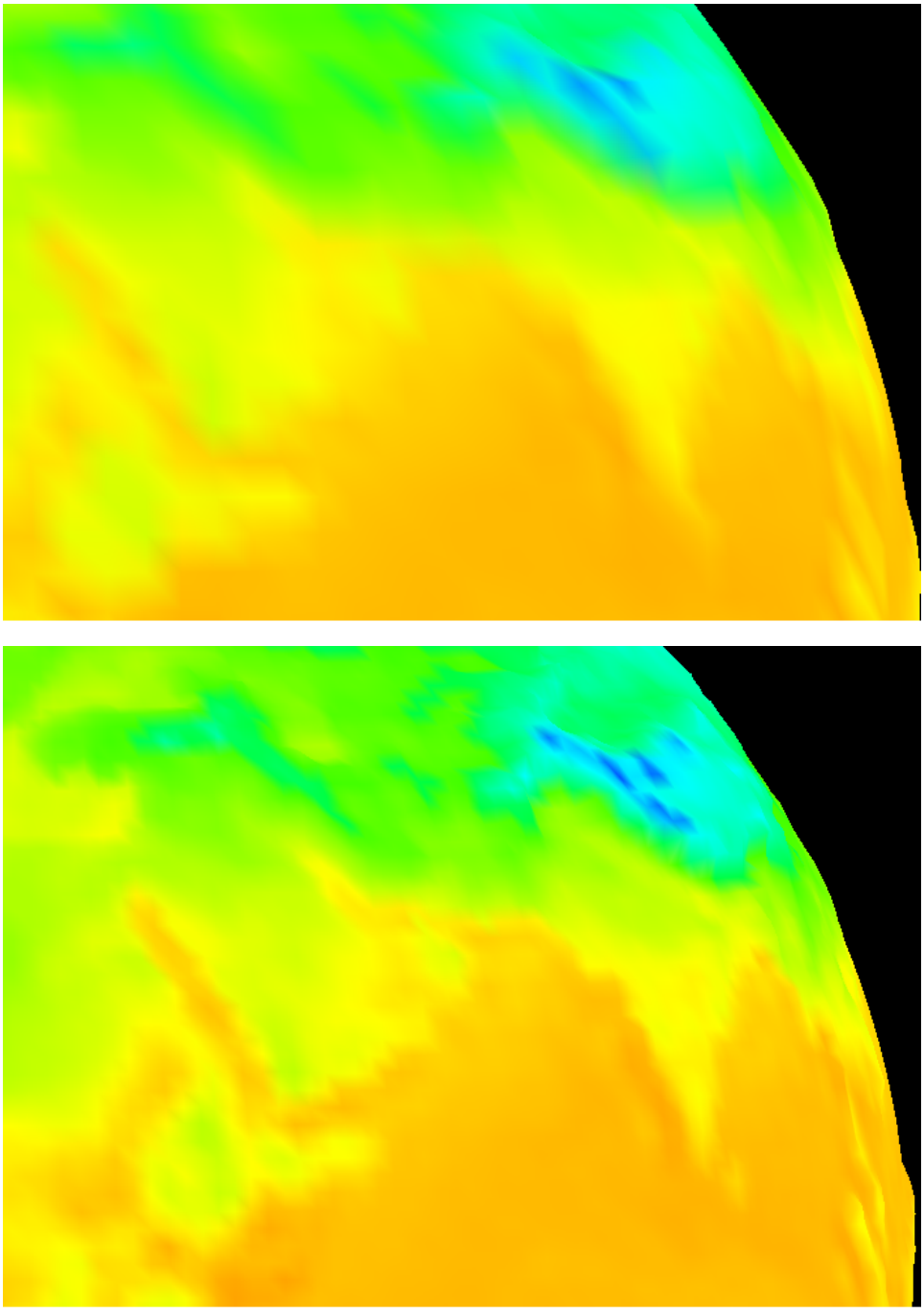


Abbildung 26: Darstellung der gering aufgelösten Temperaturen (oben) und der hoch aufgelösten (unten)

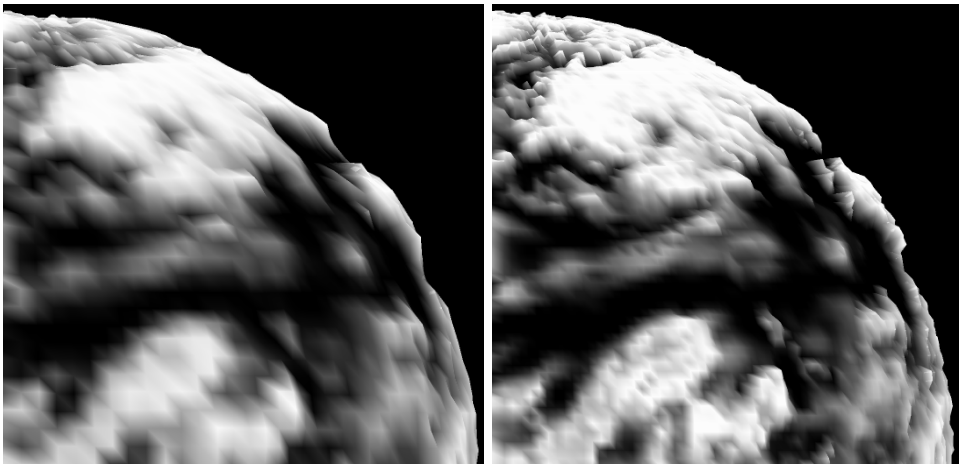


Abbildung 27: Darstellung der gering aufgelösten Wolken (links) und der hoch aufgelösten (rechts)

Tabelle 3 kann entnommen werden, dass die verringerte Auflösung einen erheblichen Performancegewinn gegenüber der Originalauflösung ermöglicht. Außerdem hält sich auch der zusätzliche Aufwand des Herunterrechnens der Auflösung in Grenzen, da die Animation beinahe so schnell abläuft, als wenn nur ein Viertel der Vertices ohne Mitteln gesetzt würde (vgl. Tabelle 3). Trotz der verringerten Bildqualität ist die Verringerung der Vertexzahl bei sehr hohen Auflösungen oder älteren Rechnern empfehlenswert, um eine flüssige Animation zu ermöglichen. Dem Benutzer ist deshalb die Möglichkeit gegeben, dieses Verfahren bei Bedarf manuell zu aktivieren / deaktivieren.

9.1.2 Frühes Clipping

Um die Zahl der zu setzenden Vertices zu verringern, ohne die Darstellungsqualität zu beeinträchtigen, ist eine Art *frühes Clipping* eingebaut worden, bei dem die auf der vom Betrachter abgewandten Erdhälfte liegenden Eckpunkte nicht in den Vertex Pointern berücksichtigt werden.
Der Sichtbarkeitstest für einen Scheitelpunkt funktioniert wie folgt:
Es gibt eine gedachte Ebene, die durch den Erdmittelpunkt verläuft und senkrecht zum Betrachter liegt. Mit der Hesseschen Normalenform kann dann sehr einfach bestimmt werden, ob ein Vertex sichtbar ist. Die zugehörige Gleichung lautet: [Vor04]

$$\vec{v} \cdot \vec{n} - \vec{a} \cdot \vec{n} = e$$

Dabei ist $\vec{a}$ ein Punkt der Ebene, $\vec{n}$ die Flächennormale und $\vec{v}$ der zu überprüfende Vertex. Ergibt sich ein e > 0, so befindet sich der Punkt vor dieser Ebene.

Da die Ebene immer durch den Kugelmittelpunkt und somit den Ursprung des Koordinatensystems verläuft, ist ein Vertex sichtbar, wenn gilt:

$$\begin{pmatrix} Vertex_x \\ Vertex_y \\ Vertex_z \end{pmatrix} \cdot \begin{pmatrix} ClipplaneNormale_x \\ ClipplaneNormale_y \\ ClipplaneNormale_z \end{pmatrix} > 0$$

Mit dieser Methode verbessert sich die Performance im Idealfall (vollständige Erdkugel und hohe Geometrieauflösung) außerordentlich stark, da dann nur jeder zweite Scheitelpunkt gesetzt werden muss. In der in Tabelle 3 dargestellten Szene bewirkt der Verzicht auf diese Technik einen erheblichen Performanceeinbruch.
Einziger Nachteil sind leichte Grafikfehler mit 3D-Wolken in Polnähe, die vermutlich durch die Flächen hervorgerufen werden, die die Halbkugel auf der Rückseite abschließen. Diese sind zwar eigentlich vom Betrachter abgewandt und aufgrund des Backface-Cullings ausgeblendet, in Polnähe könnten sie jedoch wegen der unebenen Oberfläche trotzdem zum Betrachter gedreht sein. Dadurch kommt es zu unschönen Streifen, die in Abb. 28 zu sehen sind. Dieser Grafikfehler kann als unwesentlich eingestuft werden, da er zum Großteil durch die Erdtextur überdeckt und somit durch den Z-Buffer eliminiert wird. Insgesamt stellt diese Methode eine große Verbesserung der Animationsgeschwindigkeit bei der Darstellung der gesamten Erdkugel dar. Das zuvor beschriebene Herunterrechnen der Auflösung wird dadurch nicht ersetzt, sondern ergänzt, da es auch bei kleinen Erdausschnitten einsetzbar ist und bei sehr hohen Auflösungen einen zusätzlichen Performancegewinn ermöglicht.

9.2 Hardwareabhängigkeit der Performance

Gegenstand der Untersuchung ist in diesem Kapitel, von welchen Hardwarekomponenten die Performance maßgeblich abhängt und inwieweit das Programm von Fähigkeiten aktueller Hardware profitieren kann.

Die Geometrieberechnungen, von denen die Animationsgeschwindigkeit hauptsächlich abhängt, teilen sich der Prozessor und (falls vorhanden) die Geometrieeinheiten der Grafikkarte (Vertex-Shader bzw. T&L-Einheiten). Dadurch erklärt sich die starke Prozessorabhängigkeit des Programms (vgl. Tabelle 5). Ein nicht unbedingt zu erwartender Effekt ist der in Tabelle

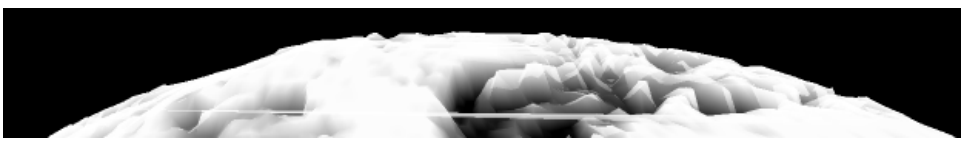


Abbildung 28: Die Wolkenfehler, die sich durch das Clipping an der Rückseite der 3D-Wolken ergeben

2 ersichtliche, dass bei verdoppelter Geometrieauflösung die Framerate von 19.2 auf 7.1 FPS sinkt und damit mehr als halbiert wird. Dies ist möglich, da Prozessor und Geometrieeinheiten parallel arbeiten, die Geometrieeinheiten die Geometrieberechnungen aber schneller durchführen können. Sind sie aber vollständig ausgelastet, muss der Prozessor den Rest berechnen, wodurch der überproportionale Performanceeinbruch zu erklären ist. Besonders deutlich zeigt sich der durch die Geometrieeinheiten mögliche Performancegewinn beim Einsatz einer Grafikkarte mit mehr als den 2 Vertex-Shadern, über die die hier verwendete Grafikkarte verfügt. Äußerst unerwartet dagegen ist der in Tabelle 5 ersichtliche geringe Geschwindigkeitseinbruch beim Runtertakten der Grafikkarte (GPU und Speicher). Die daraus resultierende Verringerung der Zeichenleistung ist, wie bereits gezeigt, vernachlässigbar. Aber es wäre zu erwarten, dass mit sinkendem GPU Takt auch die Vertex-Shader langsamer arbeiten, was einen größeren Performanceeinbruch erwarten ließe. Ein denkbarer Erklärungsversuch wäre, dass die Vertex-Shader zumindest bei der verwendeten Grafikkarte unabhängig vom GPU-Takt arbeiten. Unabhängig davon kann Tabelle 5 noch entnommen werden, dass diese nicht von der Bandbreite des AGP-Busses abhängt, da eine Veränderung der Einstellungen von 4X auf 1X die Framerate nicht messbar beeinflusst.

Einstellung	FPS	Relativer Unterschied
Wolken in 320x160	19,3	0 %
Senken der Bandbreite auf AGP 1X	19,3	0 %
Verringern des Prozessortakts auf 1GHz	13,2	-31,6 %
Senken des Grafikkartentakts auf 148/288MHz	17,8	-7,8 %

Tabelle 5: Vergleich der Abhängigkeit der Animationsgeschwindigkeit vom der Hardware: Dargestellt werden jeweils Wolken der Auflösung 320x160. Die relativen Unterschiede beziehen sich auf diese Einstellung.
System: Athlon XP 2400 (@2000 MHz), GeForce4 Ti 4400 @295/575MHz(GPU/Speicher), AGP 4X

Insgesamt kann festgehalten werden, dass das Programm stark von den Errungenschaften moderner Grafikkarten wie z.B. dem Füllen mit Farbverläufen ohne Performanceverlust oder dem Antialiasing profitiert. Auch die in modernen Grafikkarten enthaltenen Geometrieeinheiten wirken sich positiv auf die Flüssigkeit der Animation aus. Trotzdem ist die Prozessorauslastung durch das ständige Neusetzen der Vertices noch sehr hoch. Es muss an dieser Stelle die Frage gestellt werden, ob in OpenGL eine optimale Lösung für dieses Problem überhaupt möglich ist. So kann z.B. bei den Temperaturen nicht davon profitiert werden, dass die Position der Vertices sich niemals ändert.

Positiv ausgedrückt verursachen die sich ständig ändernden Positionen der Eckpunkte bei den 3D-Wolken dagegen auch keinen Mehraufwand. Wäre es in Direct3D ähnlich wie in SVG möglich, jedem Vertex mehrere Farben und zusätzlich eine Information mitzugeben, wie über diese Farben ein zeitlicher Verlauf zu bilden ist, müssten nicht mehr alle Vertices für jedes Bild neu gesetzt werden. Dann wäre theoretisch eine weit bessere Performance als in diesem Programm möglich.

10 Ausblick

Dieses Kapitel soll Anregungen geben, inwieweit das Programm noch weiterentwickelt werden könnte.

- Es wäre sehr sinnvoll einen Mechanismus zum direkten Einlesen aus den Grib-Files zu implementieren.
- Es müsste dringend überprüft werden, ob es in Direct3D effizienter möglich ist, die Farben einzelner Vertices bei gleichbleibender Geometrie zu verändern.
- In Polnähe ist die Vertexdichte weit höher als in anderen Gebieten und damit wesentlich größer als benötigt. Es wäre überlegenswert, ob nicht auch bei Temperaturen und Wolken analog zur Winddarstellung die zu setzende Vertexzahl in Abhängigkeit von der Polnähe definiert werden könnte. Auch wenn hier das Problem, dass die Vierecke nicht hundertprozentig zusammenpassen, wesentlich deutlicher auftreten würde, wäre der Performancegewinn im Falle einer Lösung dieses Problems nicht unerheblich.
- Es sollte das Springen zu einer bestimmten Stelle der Animation ermöglicht werden, außerdem könnte ein Balken die zeitliche Position relativ zur Gesamtanimationszeit angeben.
- Visualisierung bislang noch nicht berücksichtigter Daten:
  - Luftdruck
  - Niederschlag
  - mehrere Wolkenschichten in verschieden Höhen
  - ...
- Es könnte eine angepasste Linuxversion erstellt werden.

11 Zusammenfassung

Die Zielsetzung, orts- und zeitabhängige Klimadaten mithilfe moderner Technik ansprechend zu präsentieren, ist erfolgreich umgesetzt worden. Das entwickelte Programm ist in der Lage, Daten dynamisch zur Laufzeit zu laden und somit theoretisch beliebig lange Animationen zu erzeugen. Diese laufen mithilfe des verwandten Triplebuffers nahezu völlig gleichmäßig und außer bei extremen Geometrieauflösungen auf heutigen Rechnern auch sehr flüssig ab. Von modernen Grafikkarten profitiert das Programm in doppelter Hinsicht: Zum einen kann durch das Füllen von Flächen mit Farbverläufen und durch Techniken wie das Antialiasing ein in einer reinen Softwareanimation nicht vorstellbarer Detailgrad erreicht werden. An dieser Stelle wird deutlich, dass die Verwendung der 3D-Hardware nicht nur optische Spielerei ist, da das verwandte Smooth-Shading die Darstellung nicht nur ansprechender, sondern gegenüber dem in Software aus Performancegründen zu bevorzugenden Flatshading auch exakter werden lässt. Zum anderen erhöht sich die Animationsgeschwindigkeit unter anderem durch die Geometrieeinheiten moderner Grafikkarten, die den Prozessor bei der Geometrieberechnung entlasten. Trotzdem bleibt gerade bei hohen Geometrieauflösungen die Prozessorauslastung sehr hoch. Mithilfe des manuellen Clippings konnte das Problem zwar in erheblichem Maße abgemildert aber nicht vollständig gelöst werden, da immer noch sehr viele Eckpunkte für jedes Bild neu gesetzt werden müssen. Das Programm könnte noch in vielerlei Hinsicht erweitert werden: Die Implementation des Gribreaders etwa wäre ebenso sinnvoll wie die Überprüfung neuer Strategien zur weiteren Verbesserung der Performance, um noch höher aufgelöste Daten flüssig darstellen zu können - aber bekanntlich gilt: *Der Weg ist das Ziel.*
Abschließend kann festgehalten werden, dass das entwickelte Programm auch in seinem jetzigen Zustand optisch beeindruckende Echtzeitanimationen aus nüchternen Zahlen zaubern kann.

Abbildungsverzeichnis

12 Anhang

12.1 Lieferumfang

Im Anhang befindet sich eine CD mit den folgenden Ordnern:

- 3DKlimadatenvisualisierungMitOpenGL
  Enthält das in Windows ausführbare Programm
- Data
  Enthält die benötigten Daten
- Code
  Dieser Ordner enthält den Quelltext
- Text
  Der Ordner Text enthält diese Bachelorarbeit als Pdf-Datei

12.2 Starten des Programms

Das Programm kann mithilfe der Datei OGLProjekt im Ordner 3DKlimadatenvisualisierungMitOpenGL gestartet werden. Dann muss zunächst die Schaltfläche `Applikation vorbereiten` angeklickt werden, bevor mit `add Data` die benötigten Daten ausgewählt werden können. Es müssen mindestens die Küstenlinien, eine Erdtextur, die Displacement-Datei und ein Recordtyp ausgewählt werden. Ein Druck auf die Schaltfläche `Fertig` betätigt die Auswahl und wechselt in den Vollbildmodus.

Literatur

[Bur03] Lorentz Burggraf. *Jetzt lerne ich OpenGL.* Markt + Technik, 2003.

[JN94] Mason Woo Jackie Neider, Tom Davis. *OpenGL Programming Guide (The Red Book).* Silicon Graphics, Inc, 1994.

[Kal05] May-Britt Kallenrode. *Rechenmethoden der Physik.* Springer, 2005.

[uMB03] Michael Bender und Manfred Brill. *Computergrafik.* Hanser, 2003.

[Vor04] Oliver Vornberger. *Computergrafik.* Universität Osnabrück, 2004.

Erklärung

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

Freren, den .....

.....
(Unterschrift)