Fachhochschule Aachen
Abteilung Jülich

Fachbereich: Physikalische Technik
Studienrichtung: Technomathematik

Plattformunabhängige Visualisierung von Neutronenspektren auf der Basis von OpenGL

Diplomarbeit von Robert Neßelrath
Jülich, 6. Mai 2005

Die vorliegende Diplomarbeit wurde in Zusammenarbeit mit dem Forschungszentrum Jülich GmbH, Institut für Festkörperforschung (IFF), angefertigt.

Diese Diplomarbeit wurde betreut von:

Referent: Prof. Dr. M. Reißel

Koreferent: Privatdozent Dr. H. Lustfeld

Betreuer: J. Heinen

Diese Arbeit ist selbständig angefertigt und verfasst. Es sind keine anderen als die angegebenen Quellen und Hilfsmittel benutzt worden.

Jülich, 6. Mai 2005

Inhaltsverzeichnis

Kapitel 1

Einleitung

Der GLI [1] ist ein Grafiksystem, das eine vielfältige Umgebung zur grafischen Darstellung von komplexen Datensätzen und Bildern zur Verfügung stellt. GLI kombiniert mächtige Schnittstellen, flexible Grafik-Werkzeuge und einen umfangreichen Satz von Programmierwerkzeugen, um Ingenieuren, Wissenschaftlern und Analysten eine komplette Lösung zur Datenanalyse und wissenschaftlichen Visualierung zu bieten.

Ein Anwendungsbereich der GLI-Werkzeuge ermöglicht die dreidimensionalen Visualisierung von Datensätzen, die im Institut für Festkörperforschung (IFF) des Forschungszentrums Jülich insbesondere zur Darstellung von Neutronenspektren aus den Instrumenten KWS I-II und DNS zum Einsatz kommen. Speziell auf die Experimente angepaßte Programme greifen auf Schnittstellen des GLI zurück, um ohne großen Programmieraufwand in den Messungen anfallende Daten im gewünschten Format live darzustellen. Der Leistungsumfang umfaßt hierbei zum Beispiel das Zeichnen dreidimensionaler Oberflächen in Form von Gittern, als Oberflächen mit Farbinformationen oder als Kontur mit Höhenlinien. Es besteht die Möglichkeit, individuell vom Benutzer angepasste Achsen und Beschriftungen anzuzeigen, die Darstellung bis zu einem Winkel von 90 Grad zu kippen bzw. zu rotieren oder an einer beliebigen Achse zu spiegeln. Jede Achse kann logarithmisch skaliert werden.

Selbstverständlich ist die dreidimensionale Visualisierung nicht auf Neutronenspektren beschränkt, sondern kann auch für andere Anwendungsbereiche verwendet werden, z.B. zum Plot von Funktionen mehrerer Veränderlicher oder in geographischen Informationssystemen. Voraussetzung ist lediglich, dass man Funktionswerte zu den Knotenpunkten eines äquidistant verteilten Gitters hat.

Gegenüber dem in der Wissenschaft häufig verwendeten Programm *gnuplot* können

[1] Graphics Language Interpreter

Josef Heinen and Gunnar Grimm: Institut für Festerkörperforschung, Forschungszentrum Jülich

Erweiterungen des Funktionsumfangs im GLI vergleichsweise einfach realisiert werden: Aufgrund des schichtförmigen und modularen Aufbaus, den *gnuplot* nicht hat, bietet es den entscheidenden Vorteil, dass neue Funktionalitäten einfach implementiert bzw. bestehende optimiert werden können, ohne an anderer Stelle unerwünschte Seiteneffekte zu verursachen.

Die gesamte grafische Ausgabe des bisherigen GLI basiert auf den Funktionen eines grafischen Kernsystems(GKS) eine Hardwarebeschleunigung wird nicht verwendet. Dies hat bei großen Datensätzen den Nachteil, dass die Darstellung bei höheren Auflösungen merkbar langsam wird.

Aufgabe dieser Diplomarbeit ist es, die GLI-Werkzeuge, die für die dreidimensionale Darstellung zuständig sind, durch ein Plugin zu erweitern, das den gleichen Funktionsumfang bietet, jedoch unter Ausnutzung der Hardwarebeschleunigung. Hierbei ist zu beachten, dass die Software in die bestehende GLI-Struktur integriert wird und weiterhin plattformunabhängig, d.h. auf Linux-, Macintosh- und Windowssystemen, lauffähig ist. Zusätzlich soll der Funktionsumfang erweitert werden, z.B. um die Möglichkeit, die Grafik bis zu 360 Grad zu kippen und zu drehen. Ein besonderes Augenmerk liegt auf der Textdarstellung, wobei die Qualität der Schriften trotz Scherung und freier Skalierbarkeit der von Outline-Fonts entsprechen soll. Besonders für die Betrachtung finiter Elemente ist von Interesse, zusätzliche Informationen visualisieren zu können. Deswegen sollen die GLI-Werkzeuge um die Funktion erweitert werden, Informationen wie z.B. die Temperaturen eines Objektes durch Farbcodierung der Oberfläche sichtbar zu machen.

Kapitel 2

Neutronenspektren

Das Institut für Neutronenstreuung betreibt Forschung mit Neutronen mit dem Schwerpunkt 'Weiche Materie'. Weich sind Materialien, die stark auf schwache Kräfte reagieren. Neben dem Aufbau interessiert man sich für die Dynamik solcher Systeme.

Das Institut für Neutronenstreuung betreibt mehrere Streuinstrumente am Reaktor DIDO (FRJ-2) und im angegliederten Neutronenleiterlabor ELLA, darunter Messplätze für Kleinwinkelstreuung (SANS), Neutronenspinechospektroskopie (NSE), Reflektometrie, Rückstreuspektroskopie (BSS1). Ein weiterer Schwerpunkt ist die Entwicklung der Neutroneninstrumentierung für Forschungsreaktoren und zukünftige Spallationsquellen. Hierzu zählen der neuen Münchner Reaktor FRM-II oder die Megawatt-Spallationsquelle SNS in Oak Ridge (USA).

Neutronen sind elektrisch neutrale Teilchen mit einem Spin von 1/2, die zusammen mit den Protonen die Bausteine der Atomkerne sind. Ihre Masse ist geringfügig größer als die des positiv geladenen Protons. Der von der Quantenmechanik beschriebene Welle-Teilchen-Dualismus besagt, dass sich Neutronen auch wie Materiewellen verhalten. Die Wellenlänge thermischer Neutronen liegt im Bereich von Atomabständen in kondensierter Materie (Kristalle, Flüssigkeiten).

Da Neutronen elektrisch neutral sind, dringen sie tief in Materie ein, sie werden hauptsächlich von den Atomkernen gestreut. Somit lassen sich Anordnung und Bewegungsformen der Atom(kern)e in einem geeigneten Streuexperiment bestimmen.

Die Experimente KWS-1 und KWS-2, die sich mit der Kleinwinkelstreuung beschäftigen, sind für Wellenlängen im Bereich zwischen 10^{-3} bis 0.2 $Å^{-1}$ konstruiert und bieten die Möglichkeit, heterogene Strukturen zwischen 10 und 1000Å zu untersuchen. Dazu gehören Polymere in Schmelzen und Lösungen wie Gele und gummiartige Netzwerke in Form von Aggregaten sowie die Glasdynamik an amor-

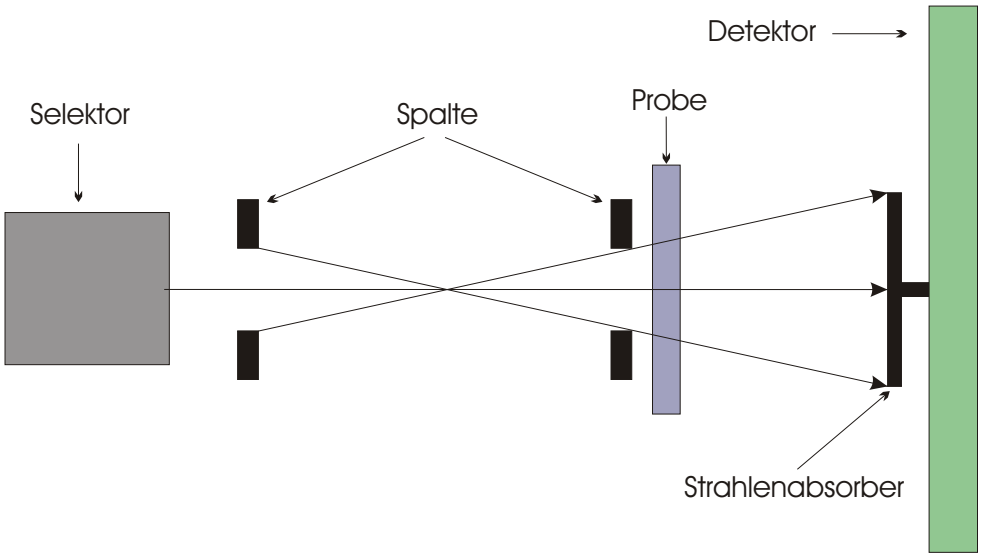


Abbildung 2.1: Aufbau des KWS-1 Streuexperiments

phen Polymeren. Eine weitere Klasse bilden komplexe Flüssigkeiten wie Mikroemulsionen oder Kolloidsysteme.

Als Neutronenquelle dient der DIDO Reaktor des Forschungszentrums. Aus dem Neutronenvorrat werden im Selektor Neutronen einer bestimmten Wellenlänge gefiltert, parallelisiert und anschliessend auf eine Probe geschossen, die die Neutronen streut. Um die Streuung quantitativ zu erfassen, ist hinter der Probe eine Detektorplatte platziert. Für die Messungen sind nur die Neutronen von Interesse, die tatsächlich gestreut wurden. Daher ist vor dem Detektor ein Neutronenabsorber angebracht, der Neutronen, die nicht gebrochen werden und eventuell die Meßergebnisse verfälschen würden, auffängt. Der Detektor im KWS-1 Experiment ist ein Szintillator aus ^{6}Li-Glas. Innerhalb des Szintillators werden die Neutronen absorbiert und dadurch Photonen emittiert, welche eine Anordnung aus 8 x 8 Photonenmultipliern erfasst und anschliessend auf 128x128 Kanäle verteilt.

Die folgende 3D-Darstellung zeigt ein typisches Messbild des KWS-1 Experiments.

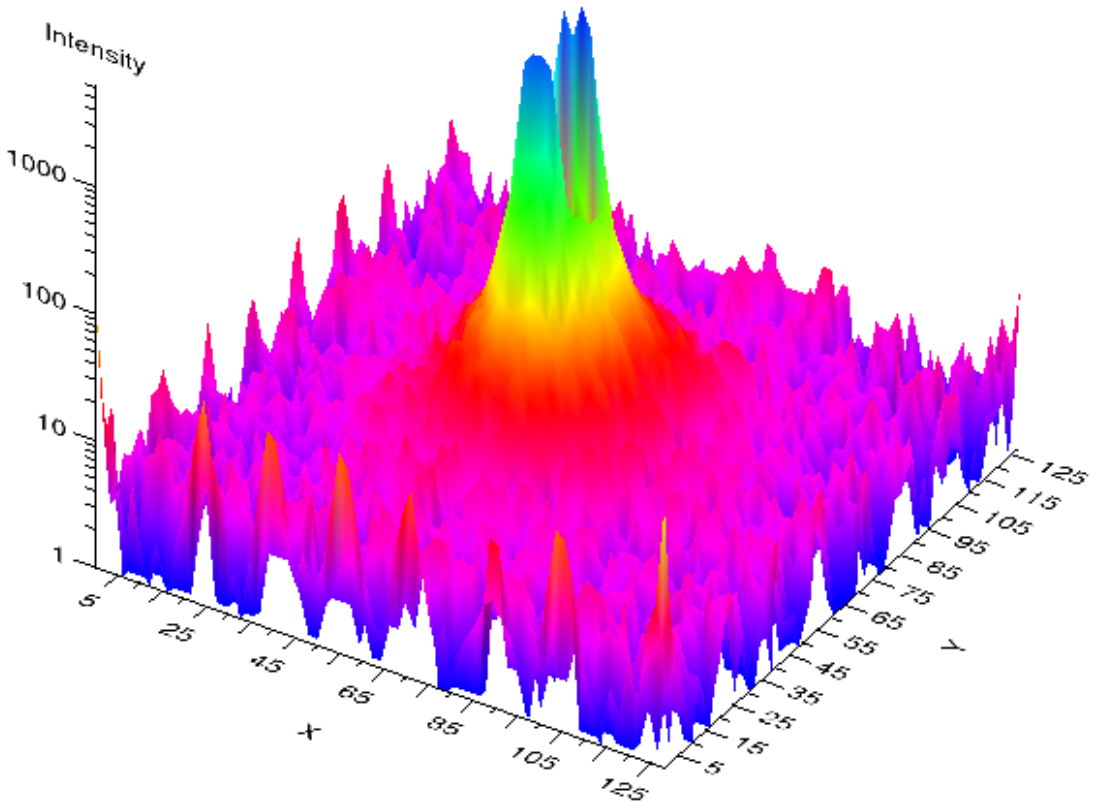


Abbildung 2.2: Beispiel für eine typische Messung des KWS1-Streuexperiments

Kapitel 3

Hilfsmittel

3.1 OpenGL

Zur Realisierung der 3D-Computergrafik musste zunächst entschieden werden, welche Schnittstelle zur Darstellung verwendet wird. Voraussetzung ist, dass die Bibliothek auf allen GLI unterstützenden Betriebssystemen läuft und die Hardwarebeschleunigung von Grafikkarten ausnutzt.

OpenGL ist eine Spezifikation für ein plattform- und programmiersprachenunabhängiges API (Application Programming Interface) zur Entwicklung von 3D-Computergrafiken. OpenGL ist nur ein Standard, keine Implementierung. Wie das Betriebssystem die Befehle verarbeitet, ist Sache des Subsystem-Treibers, der die Befehle an die Grafikkarte weitergibt (Hardware- oder auch Direct Rendering) oder auf der CPU ausführt, wenn die Grafikkarte den entsprechenden Befehl nicht bearbeiten kann (Software Rendering).

Aufgrund seiner Plattformunabhängigkeit gehört OpenGL im 3D-Grafik-Bereich mittlerweile zum Industriestandard. Somit eignet es sich ideal zur Implementierung der Visualisierung, da die Lauffähigkeit auf den unterschiedlichsten Plattformen gewährleistet bleibt. Die Qt-Library (GUI-Library, die ich für meine Arbeit verwende, näheres s.u.) enthält ein Modul, das OpenGL direkt unterstützt.

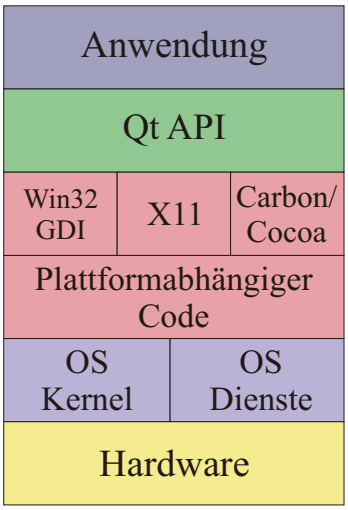
3.2 Qt

Qt ist ein Multiplattform C++ GUI Toolkit, das von Trolltech entwickelt wurde. Es bietet Anwendungsentwicklern alle Funktionalitäten, um moderne grafische Benut-

zerschnittstellen zu erstellen. Qt ist vollkommen objektorientiert und leicht erweiterbar. Seit seiner kommerziellen Einführung im Jahr 1996 bildet Qt die Basis für tausende erfolgreiche Anwendungen weltweit. Auf Qt basiert zum Beispiel die berühmte KDE Linux Desktop Umgebung, eine Standardkomponente aller wichtigen Linux Distributionen.

Qt wird auf folgenen Plattformen unterstützt:

- *MS/Windows*
  95, 98, NT 4.0, ME, 2000, and XP
- *Unix/X11*
  Linux, Sun Solaris, HP-UX, Compaq Tru64 UNIX, IBM AIX, SGI IRIX und viele andere
- *Macintosh*
  Mac OS X



Durch die Plattformunabhängigkeit bietet Qt genau die Voraussetzung, die zur Implementierung der Arbeit benötigt wird. Erwähnenswert ist auch ein OpenGL-Modul innerhalb von Qt, das es dem Anwender ermöglicht, OpenGL Grafiken direkt in eine Qt-Applikation zu zeichnen. Außerdem wird von folgenden Funktionalitäten Gebrauch gemacht:

- Erstellen eines Socket Servers (Ausnutzung der Portabilität)
- Mausabtastung zum Kippen und Drehen der Grafik
- Druckerunterstützung und Zeichen-Funktionen zum Erstellen von Schriftzügen

Kapitel 4

Die GR-Struktur

4.1 GR-Schnittstelle

Die GR(Graphics)-Schnittstelle, ein Bestandteil des GLI, beinhaltet Routinen zur Darstellung von zwei- und dreidimensionalen Grafiken. Sie kann aus den unterschiedlichsten Programmiersprachen angesprochen werden, dazu gehören unter anderen C, C++, Fortran und Python. Zur Realisierung nutzt das GR GKS-Funktionen, die es über ein C-Binding aufruft.

Das GKS (Graphical Kernel System) ist ein ISO-Standard zur Darstellung von zweidimensionalen Linear- und Vektorgrafiken, der unabhängig von Plattformen und Programmiersprachen entwickelt wurde. Das GKS ermöglicht, Grafiken auf viele verschiedene Logical Device Driver abzubilden. Hierzu gehören die grafischen Subsysteme verschiedenster Plattformen, aber auch Industriestandards wie PS oder PDF.

Auch zur Darstellung dreidimensionaler Oberflächen wird in der bisherigen Versi-

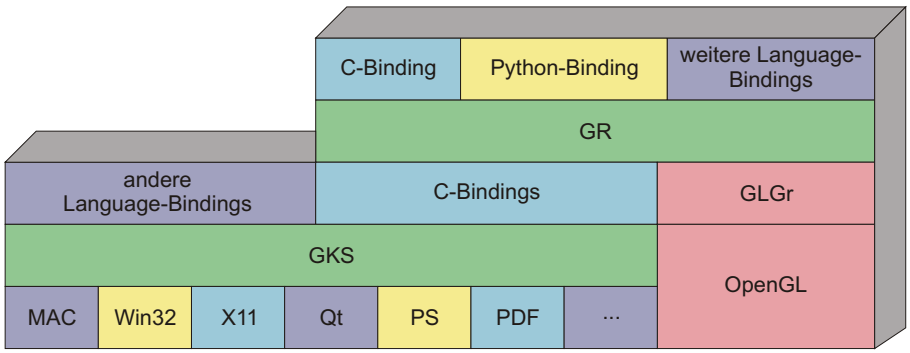


Abbildung 4.1: GR-Struktur

on das GKS verwendet. Transformationen, Rotationen etc. übernehmen Funktionen des GR, haben jedoch, wie in der Einleitung bereits erwähnt, einige Schwächen, wie z.B. unzureichende Performance sowie weitere Beschränkungen in der Darstellung. Daher wird zukünftig parallel oder alternativ zu der bisherigen Umsetzung mit GKS das Modul GLGr verwendet, welches im Rahmen der Diplomarbeit neu entwickelt wurde. Dieses wird lose in die bestehende Struktur eingebunden und kann über eine Message-basierte Schnittstelle, die über einen TCP-Socket realisiert wurde, angesprochen werden (siehe Kapitel ’Kommunkation des GR mit dem Modul GLGr’, Seite 29).

Das GLGr Modul nutzt mit der OpenGL-Engine eine mächtige Umgebung, um dreidimensionale Darstellungen zu ermöglichen, wodurch der Bildaufbau merkbar beschleunigt wird.

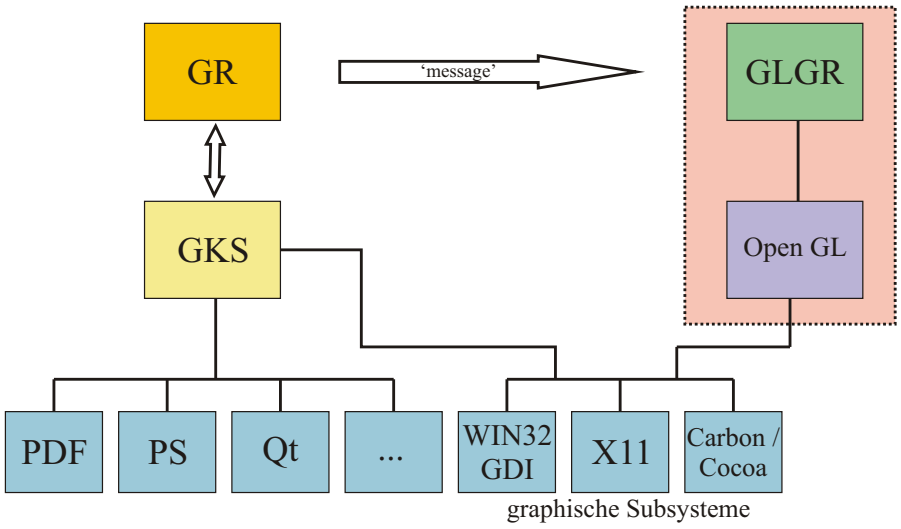


Abbildung 4.2: Flussdiagramm der GR-Struktur

4.2 GLGr-Modul

4.2.1 GLGr

Das GLGr-Modul ist in C++ implementiert und nutzt die Qt-Library. Kernstück ist eine Klasse, die vom QGLWidget abgeleitet ist, einem Qt-Widget, das dem Programmierer eine einfache Möglichkeit bietet, OpenGL-Grafiken in Qt-Anwendungen einzubauen.

Die Klasse beinhaltet alle Attribute, die zur Visualisierung benötigt werden. Dazu gehören die in Oberflächen umzusetzenden Daten und Parameter, die angeben, wie

die Werte visualiert werden sollen.
Anhand dieser Informationen werden sogenannte Vertex-Arrays berechnet, die die Punkte der Oberfläche im Raum beschreiben. Hinzu kommen weitere Daten wie Indexvektoren, um festzulegen, in welcher Reihenfolge die Punkte im Raum miteinander verbunden werden sollen, Schriftzug-Texturen, Farbpaletten oder Farbinformationen für die Oberfläche.
Sobald die Grafik neu gezeichnet wird, ruft das GLGr mit Hilfe der erzeugten Datenstrukturen entsprechende Funktionen der OpenGL-Engine auf, um das finale Bild zu erstellen.

Eine weitere Funktion der GLGr-Klasse ist, die vom GR empfangenen Kommandos zu interpretieren und dementsprechende Einstellungen vorzunehmen.

4.2.2 Klasse `glgrserver`

Die Klasse `glgrserver` stellt einen Server zur Verfügung, der die Kommunikation mit dem GR regelt. Die Verbindung wird über ein Socket aufgebaut und empfangene Daten per `SIGNAL` an die GLGr-Klasse zur Weiterverarbeitung übergeben.

4.2.3 Klasse `vec`

Die `vec`-Klasse beschreibt einen Punkt im dreidimensionalen Raum und wird von der GLGr-Klasse als Hilfsmittel verwendet. Neben normalen Vektoroperationen wie Addition und Multiplikation mit Skalaren kann sie Multiplikationen mit Transformationsmatrizen durchführen oder Beträge der Vektoren bilden.

4.2.4 Klasse `rect`

Die Klasse ist ebenfalls eine Hilfsklasse der GLGr-Klasse. Sie beschreibt die Rechtecke, auf die die Texturen der Schriftzüge projiziert werden. Mit Hilfe der Funktion `intersects` kann geprüft werden, ob sich zwei Rechtecke überlappen. Dies ist für das Zeichnen von Achsen nötig. Wird die Beschriftung der Achse erstellt, überprüft GLGr mit dieser Funktion, ob sich einzelne Texturen überschneiden und erhöht in dem Fall die Abstände der zu beschriftenden Teilstriche.

Kapitel 5

Programmbeschreibung

Zum besseren Verständnis der Beschreibungen ist es von Vorteil, sich mit der Qt-Bibliothek auszukennen. Grundlegende Dinge wie die Funktionsweise von Events werden als bekannt vorausgesetzt und es wird nicht näher auf sie eingegangen. Ebenfalls sollten Vorkenntnisse in Open-GL vorhanden sein.

5.1 Koordinatensystem

Die Messungen der Spektraldaten liegen als Datensatz vor, der sich auf einem äquidistanten Gitter befindet. Es liegt also nahe, dieses Gitter auf eine Grundebene zu legen und die Werte zu den einzelnen Gitterpunkten als Höhenwert im Raum abzubilden.

Open-GL bietet die Möglichkeit, in einem beinah unendlich großen, dreidimensionalen Raum zu arbeiten. Im unrotierten Zustand ist das Koordinatensystem ein Rechtssystem, dessen z-Achse aus dem Bildschirm herausragt.
In eine kleine Teilmenge hiervon werden die Daten der Spektren transformiert. Die drei Einheitsvektoren in Richtung der Achsen spannen hierbei den Raum auf, in dem die Darstellung stattfindet. Die x-Achse der Spektraldaten wird weiterhin durch die x-Achse dargestellt, die y-Achse durch die negative z-Achse und die z-Achse durch die y-Achse. Die Umrechnung für jede Achse erfolgt über die Formel:

$$w = \frac{q - qmin}{qmax - qmin}$$

Im Prinzip wird hierbei der Bereich jeder Achse auf das Intervall [0,1] abgebildet. q ist der zu transformierende Wert, w der transformierte Wert. $qmax$ und $qmin$ geben die Grenzen des Ursprungsintervalls an.

5.2 Datenstrukturen

Für die Visualisierung der Oberflächen werden folgende Datenstrukturen benötigt:

- `source_data3D` und `source_dataCC`

  Die beiden Datenstrukturen sind dynamisch angelegte `float`-Vektoren. In ihnen werden die ursprünglichen Daten, die das GLGr-Modul vom Anwender erhält, abgelegt. Die Höheninformationen werden in `source_data3D` gespeichert, in `source_dataCC` die Informationen für die Werte der Farbcodierung beziehungsweise ein NULL-Pointer, falls solche nicht vorhanden sind.

- `data3D` und `dataCC`

  Beide Strukturen sind `float`-Vektoren, die dynamisch angelegt werden. In ihnen werden die Daten gespeichert, die tatsächlich zur Darstellung der Oberflächen genutzt werden. Unterschied zu den Quelldaten ist, dass hier die Datenmenge durch bilineare Interpolation verringert sein kann.

- `surfaceVertices`

  Dieser dynamisch angelegte `GLfloat`-Vektor beinhaltet die Positionen aller Punkte, durch die die Oberfläche dargestellt wird. Er wird per Pointer an die OpenGL-Schnittstelle übergeben. Drei aufeinanderfolgende Werte beschreiben die X,Y und Z Position eines Punktes. Das Gitter wird zeilenweise auf dem Vektor abgelegt, d.h. zu y=1 bis y=n werden nacheinander alle x-Werte gespeichert.

- `surfaceIndices`

  Im Vektor `surfaceVertices` sind alle Punkte der Oberfläche gespeichert. Anhand der Indizes wird beschrieben, in welcher Reihenfolge die Punkte zur Erzeugung der Oberfläche verwendet werden. Jede Darstellungsart der Oberfläche benötigt eine eigene Indexmenge. Die Oberfläche wird reihenweise aufgebaut, weshalb die einzelnen Elemente von `surfaceIndices` auf einen Vektor mit Indizes vom Typ `GLuint` zeigen.

- `gridIndices`

  dient dem gleichen Zweck wie surfaceIndices, nur werden hier die Indizes zum Erzeugen eines Gitters gespeichert

- `surfaceColors`

  Zu jedem Punkt der Oberfläche existiert eine Farbinformation, die im Vektor `surfaceColors` gespeichert wird. Je drei aufeinanderfolgende Werte beschreiben den rot-,grün- und Blauanteil der Farbe eines Punktes.

- `colorTable`

  `colorTable` ist ein Vektor vom Typ `QColor`, der die 72 Farben der gerade aktiven Farbpalette enthält.

- `contourVertices` und `contourIndices`

  Die beiden Datenstrukturen dienen einem gleichen Zweck wie `surfaceVertices` und `surfaceIndices`, stellen jedoch die Oberfläche der Konturebene dar. Als Farbinformationen werden die Werte von `surfaceColors` verwendet.

- `contourLineVertices`

  `contourLineVertices` ist ein Vektor, der die Punkte der zu zeichnenden Höhenlinien enthält. Zusammenhängende Linien liegen auch als Punkte hintereinander auf diesem Vektor. Eine Trennung zweier solcher Linien wird durch einen Punkt mit den Koordinaten (0,1000,0) gekennzeichnet. Indizes werden nicht benötigt, da die Punkte der Reihe nach verarbeitet werden.

5.3 Bilineare Interpolation der Daten

Um sehr große Datenmengen zu zeichnen kann es Sinn machen, die Größe des Gitters zu reduzieren. Hierdurch wird die Darstellung merkbar beschleunigt, der Verlust an Oberflächeninformationen sollte jedoch möglichst gering bleiben. Das Interpolieren zur Vergrößerung von Datenmengen wird im GLGr nicht unterstützt.

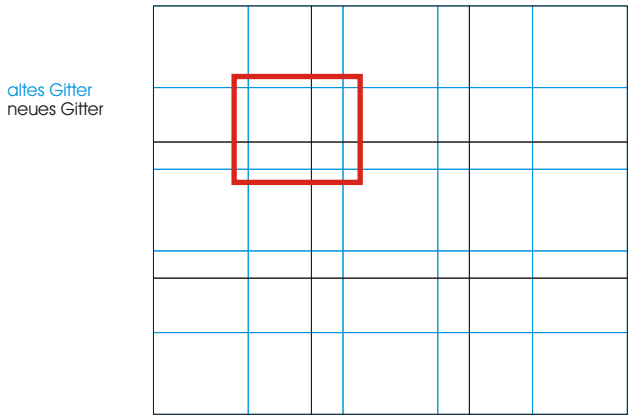
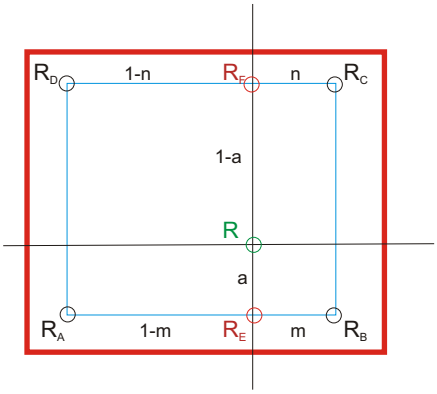


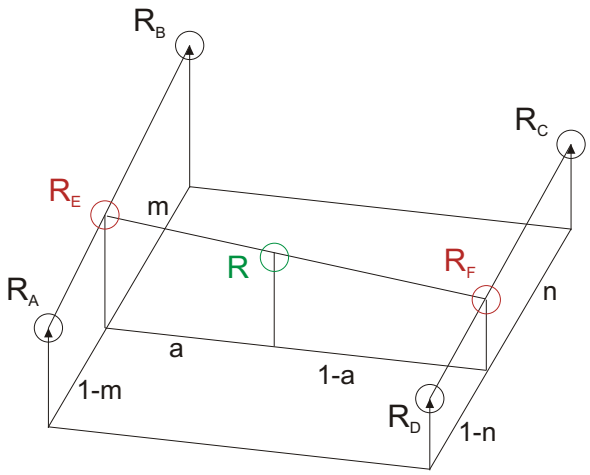
Abbildung 5.1: bilineare Interpolation: Festlegung eines neuen Gitters

Zur Reduzierung wird über das ursprüngliche Gitter ein neues Gitter mit einer geringeren Anzahl an Punkten gelegt. Zu den Knotenpunkten des neuen Gitters müssen nun neue Werte bestimmt werden. Da diese im seltensten Fall genau auf die Punkte

des ursprünglichen Gitters fallen, benötigt man ein Verfahren, das die Werte mit Hilfe der vier benachbarten Punkte abschätzt. GLGr macht hierbei von der bilinearen Interpolation, einer Standardmethode in der Bildverarbeitung, Gebrauch. Die bilineare Interpolation besteht aus zwei Schritten:

- lineare Interpolation zweier gegenüberliegender Seiten
- lineare Interpolation der Zwischenergebnisse





$$R_E \quad m\ R_A \quad (1 \quad m)\ R_B$$
$$R_F \quad m\ R_D \quad (1 \quad n)\ R_C$$
$$R \quad (1 \quad a)\ R_E \quad a\ R_F$$

Abbildung 5.2: bilineare Interpolation: Interpolationsschritte

R	-	interpolierter Wert
R_A, R_B, R_C, R_D	-	Werte der vier benachbarten Punkte
R_E, R_F	-	Zwischenergebnisse

5.4 Erzeugen von Oberflächen

Oberflächen und Gitter im GLGr werden anhand der Punkteinformation im Vektor `surfaceVertices` und der Indexinformationen `surfaceIndices` und `gridIndices` aufgebaut. Es gibt vier unterschiedliche Darstellungsweisen:

- Oberfläche und Gitter

  Die Punkte des Datensatzes werden zu einer Oberfläche verbunden. Hiefür werden Dreiecke verwendet, die mit Hilfe der Option `GL_TRIANGLE_STRIP` die Fläche zeilenweise auf dem Gitter aufbauen. Auf `surfaceVertices` sind die Punkte im Raum abgespeichert. Anhand der Indizes in `surfaceIndices` ist nun festgelegt, in welcher Reihenfolge die Punkte verbunden werden. Dies geschieht Zeile für Zeile, weshalb es zu jeder Reihe einen Vektor gibt, auf den ein Element von surfaceIndices zeigt.
  Ähnlich werden die Punkte zum Erzeugen des Gitters verwendet. Die Indizes sind die gleichen, nur werden sie auf `gridIndices` abgespeichert und die Punkte mit der Option `GL_QUAD_STRIP` verbunden, wodurch Vierecke entstehen. Die Reihenfolge der Indizierung wird auf Abb. 5.3 deutlich:

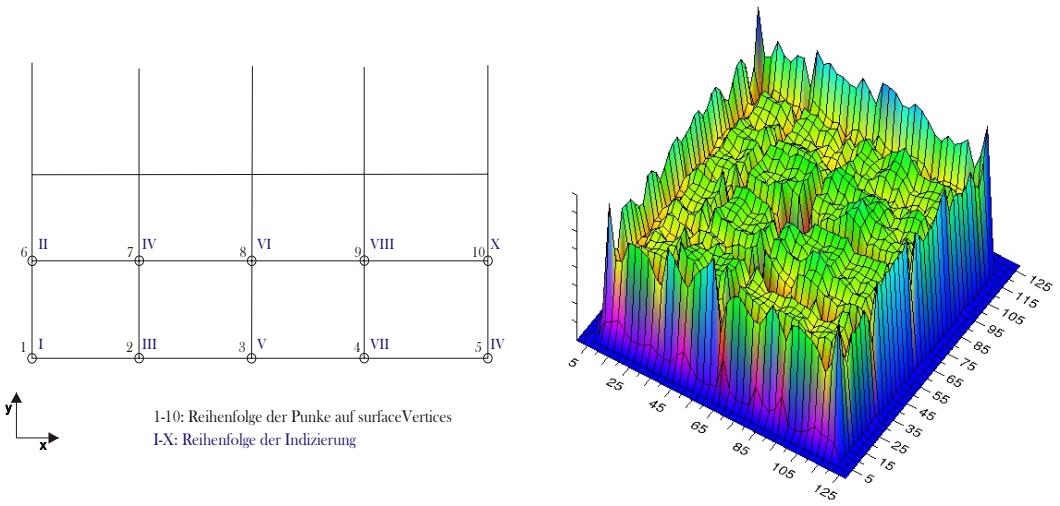


Abbildung 5.3: Aufbau: Oberfläche und Gitter

- Schichtmodell

  Beim Schichtmodell werden die Daten wie eine Zwiebel in einzelne Schichten aufgeteilt, wobei es die Möglichkeit gibt, die Schichten in x- oder in y- Richtung zu zeichnen. Da in dem Fall nicht nur Linien gezeichnet werden, sondern auch weisse Flächen, die jeweils eine Schicht darstellen, müssen die Punkte des `surfaceVectors` um die Gitterpunkte bei z=0 erweitert werden. Es

werden zeilenweise bzw. reihenweise (je nach Richtung) immer abwechselnd der Punkt am Boden des Gitters und der Punkt im Raum abgespeichert. Die Indizes sind so bei beiden Richtungen gleich. Für die Linien werden jeweils nur die oberen Punkte benötigt, zum Aufbauen der Fläche verwendet man wiederum einen `GL_TRIANGLE_STRIP`.

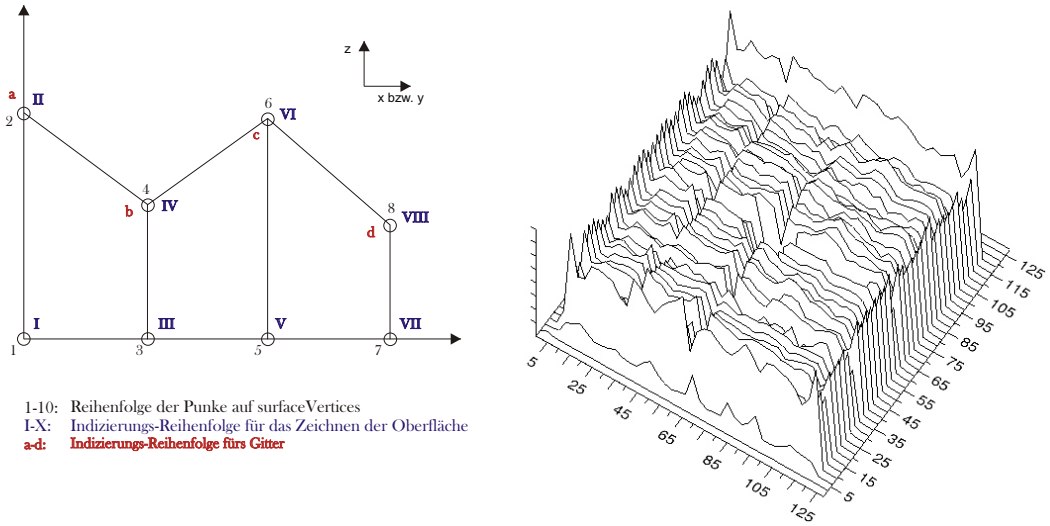


Abbildung 5.4: Aufbau: Schichtmodell

5.5 Farbcodierung

Neben einer räumlichen Darstellung der Daten(Oberflächenmodus) besteht die Möglichkeit, Farben zu verwenden(Konturmodus). In diesem Fall wird die xy-Ebene werteabhängig eingefärbt, wobei die Werte entweder linear oder logarithmisch über eines von elf Farbspektren verteilt sind. Die Spektren sind aus der bisherigen GLI-Version übernommen worden. Zusätzlich kann man beide Darstellungsmodi kombinieren, d.h. es werden nicht die Punkte der xy-Ebene sondern die entsprechenden Punkte der Oberfläche eingefärbt (als Beispiel s. Abb. 2.2 auf Seite 11).

In Anwendungen wie der Betrachtung finiter Elemente macht es Sinn, die Farbcodierung der Oberfläche unabhängig vom Datensatz zu gestalten, der zur räumlichen Darstellung verwendet wird. Das ist zum Beispiel der Fall, wenn man die Wärmeverteilung auf einem Körper mit Hilfe von Farben verdeutlichen möchte.

Der GLI bietet in der neuen Version unter Ausnutzung des GLGr-Moduls die Möglichkeit, zwei unterschiedliche Datensätze gleicher Größe einzulesen, von denen einer die Oberfläche beschreibt und der zweite die Farbcodierung.

5.6 Zeichnen von Höhenlinien

Höhenlinien werden benutzt, um in der Konturdarstellung Höheninformationen abzubilden. Hierzu werden in regelmäßigen Intervallen (Äquidistanz) alle Punkte gleicher Höhe durch eine Kurve verbunden.

In GLGr besteht die Möglichkeit, die Anzahl der gewünschten Linien anzugeben. Aus dem Wert werden über dem Werteintervall äquidistant verteilte Höhen berechnet und die Linien entsprechend hierzu gezeichnet. Ein Problem dabei ist, dass nur die Punkte auf den Gitterknoten bekannt sind, die Punkte zu der entsprechenden Höhe also jeweils interpoliert werden müssen. Für die Interpolation wird ein lineares Verfahren verwendet. Im folgenden wird beschrieben, wie das Zeichnen der Höhenlinien programmiertechnisch umgesetzt wurde.

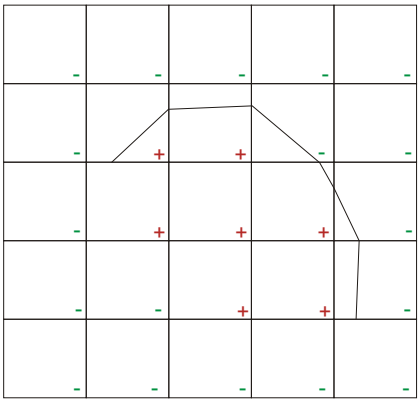


Abbildung 5.5: Erzeugen von Höhenlinien

Zunächst wird eine Matrix erzeugt, die die gleiche Größe wie das Gitter besitzt. Zu jedem Punkt wird dort vermerkt, ob er grösser-gleich oder kleiner als der gewünschte Höhenwert ist. Anschliessend durchläuft der Algorithmus die Matrix von unten links nach oben rechts und prüft bei jedem Punkt, der über dem Höhenwert liegt, ob in eine der vier Richtungen (sofern kein Rand erreicht wurde) ein Übergang unter den Höhenwert stattfindet. Ist dies der Fall, wird der wahre Punkt zwischen den beiden Gitterpunkten linear interpoliert und die Funktion `find_way` aufgerufen, die gegen den Uhrzeigersinn die Höhenlinie weiterzeichnet. Die Linie wird in dem Fall soweit gezeichnet, bis entweder der Rand erreicht wird oder ein Punkt, der in entsprechender Richtung bereits bearbeitet wurde. In diesem Fall ist die Höhenlinie entweder wieder am Ausgangspunkt angekommen oder an einer Stelle, die zuvor schon in entsprechender Richtung besucht wurde. Die Funktionsweise von `find_way` wird im Struktrogramm (Abb. 5.6) verdeutlicht:

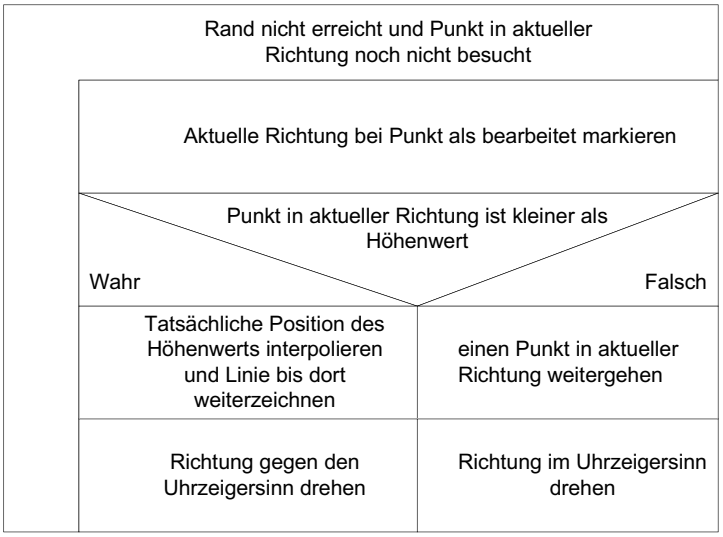


Abbildung 5.6: Nassi: find_way Funktion

5.7 Achsen

Durch Rotation des Bildes kann es passieren, dass einzelne Achsen hinter der Oberfläche verschwinden. Deswegen sollte immer sichergestellt sein, dass die Achse an der Kante gezeichnet wird, die aktuell nicht verdeckt wird. Hierfür muss zunächst festgestellt werden, ob das Bild auf dem Kopf steht.
Die Vektoren (0.5, 0, 0) und (0.5, 1, 0) sind hierbei entscheidend. Beide werden mit der aktuellen Transformationsmatrix multipliziert und verglichen. Ist der y-Wert des ersten Punktes größer als der des zweiten Punktes, steht die Grafik auf dem Kopf, ansonsten nicht. Mit Hilfe dieser Information wird nun die Entscheidung gefällt, an welcher Kante man die Achse zeichnet. Steht das Bild nicht auf dem Kopf, so ist es sinnvoll sie an die Kante zu zeichnen, die transformiert weiter unten auf dem Bildschirm liegt, ansonsten wird die weiter oben liegende gewählt. Welche der Kanten weiter oben bzw. unten liegt, findet man heraus, indem die Mitten beider Kanten transformiert und anschliessend beide y-Werte miteinander verglichen werden.

Ein weiteres Problem ist, die Richtung der Teilstriche einer Achse festzulegen. Je nach Winkel kann es passieren, dass die Striche nach der Transformation in Richtung der z-Achse gezeichnet werden, also entweder gar nicht oder kaum sichtbar sind. Da ein Strich in eine der beiden Dimensionen gezeichnet wird, in die die Achse nicht zeigt, muss entschieden werden, in welcher Richtung mehr vom Teilstrich sichtbar ist.
Hierzu werden die Einheitsvektoren in beide Richtungen betrachtet. Nach Transformation ist von Interesse, welcher der beiden Vektoren betragsmäßig auf dem Bildschirm größer ist. Zum Vergleich sind also nur die x- und y-Werte der transformierten

Vektoren von Interesse, deren Beträge miteinander verglichen werden.

5.8 Erstellen und Einfügen von Zeichensätzen

Das Einfügen von Zeichensätzen wird von OpenGL nicht direkt unterstützt. Im GL-Gr werden diese jedoch benötigt, um z.B. Titelzeilen zu erstellen oder Achsen zu beschriften. Es gibt zwar allerlei Hilfsmittel, um Schriften zu erzeugen (z.B. die GLUT-Library), jedoch ist das Ergebnis nicht besonders zufriedenstellend, da es bei hohen Auflösungen zu Treppeneffekten kommt. Eine Alternative hierzu ist, Bitmaps aus entsprechenden Zeichensätzen zu erzeugen und sie als Texturen in die Open-GL Umgebung einzufügen. Diese Bitmaps werden auf einfache Geometrien gemappt und können so bei Rotationen mittransformiert werden. Die Berechnung der Scherungen übernimmt hierbei OpenGL. Durch Erstellen von Bitmaps genügend großer Auflösung ist es so möglich, selbst bei Ausdrucken auf DIN A0-Größe ein gutes Ergebnis zu erzielen. Im folgenden Abschnitt wird beschrieben, wie im GLGrModul die Texturen erstellt und eingebunden werden.

Es wird wieder die QT-Library als Hilfsmittel verwendet. Mit QFontMetrics ermittelt man die benötigte Höhe und Breite des Schriftzugs. Da OpenGL nur Texturen unterstützt, deren Abmessungen jeweils einer Zweierpotenz entsprechen, werden diese mit Hilfe der Funktion `make_pot(int)` ermittelt. Hierbei sucht man die nächst höhere Zweierpotenz. Es wird ein Pixmap der errechneten Größe erzeugt und hierauf der Schriftzug gezeichnet. Das Pixmap konvertiert man zu einem Bitmap und speichert es in einem OpenGL-kompatiblem Format ab. Um Transparenz zu erreichen, sucht man das Bitmap nach weißen Pixmaps ab und markiert diese. In OpenGL geschieht dies durch den Alpha-Wert einer Farbe, den man auf 0 (also volle Transparenz) setzt.

Mit dem erzeugten Bitmap erstellt man eine OpenGL Textur und bindet diese an eine Identifikationsnummer, um sie später erneut ansprechen zu können. Da das Bitmap aufgrund der benötigten Zweierpotenzen größer ist als der Schriftzug, wird eine Struktur mit dem Namen `Textur` verwendet, die die ID der Textur und die Ausmaße des Bitmaps sowie die des Schriftzugs enthält. Anhand dieser Daten ist es möglich, die Textur anschließend allein auf den Zeichensatz zurecht zu schneiden.

Als nächstes muss eine geometrische Figur erstellt werden, auf die man die Textur projiziert. In diesem Fall genügt ein Rechteck. Da das Rechteck durch Drehen und Kippen in der Darstellung auf dem Kopf stehen kann, ist es wichtig zu ermitteln, in welche Richtung die gemappte Textur in Bezug auf das Rechteck im untransformierten Raum zeigen muss. Hierdurch wird sichergestellt, dass der Text für den Betrachter nicht spiegelverkehrt oder kopfüber erscheint. Dies geschieht mit Hilfe der Funktion `provideCorrectDirections`. Man übergibt die beiden Richtungs-

vektoren, durch die die Projektionsfläche aufgespannt wird. Zunächst betrachtet die Funktion den Vektor, der die Textrichtung bestimmt. Durch Multiplikation mit der aktuellen Transformationsmatrix kann herausgefunden werden, ob der Text auf dem Bildschirm in positive oder negative x-Richtung zeigen würde. Ist die Richtung negativ, wird der Vektor mit -1 multipliziert. Anschließend erfolgt die Betrachtung des zweiten Vektors, der ebenfalls mit der Transformationsmatrix multipliziert wird. Man vernachlässigt den z-Wert und bildet das Kreuzprodukt der x- und y-Werte beider transformierter Vektoren. Ist das Kreuzprodukt kleiner Null, liegt kein Rechtssystem vor mit der Folge, dass der Text auf dem Kopf stehen würde. In diesem Fall wird der zweite Vektor mit -1 multipliziert.

Zuletzt wird die Projektionsfläche gezeichnet und die Textur mit Angabe der korrekten Schnittpunkte eingebunden.

5.9 Kommunikation des GR mit dem Modul GLGr

Da mit Plugin-Technik gearbeitet wird, benötigt man eine Schnittstelle, um GLGr in die bestehende GR-Struktur einzubauen. Hierüber können die Gr-Funktionen die benötigten Kommandos an das GLGr-Modul weitergeben, sofern dieses Plugin aktiv ist.

Die Kommandos werden zunächst vom GR gesammelt und als Ressourcen in Zeichenketten abgelegt. Anschließend werden die Informationen über eine TCP-Socket-Verbindung an den GLGr-Client übertragen. Die Syntax sieht wie folgt aus:

```
<Kommando1>: <Attribut1> <Attribut2> .... <Attribut n>;
<Kommando2>: <Attribut1> <Attribut2> .... <Attribut m>
```

Nach der Kommando-ID folgt ein Doppelpunkt und anschließend die zum Kommando gehörenden Attribute. Diese können entweder durch Leerstellen oder Kommata voneinander getrennt werden. Zusammenhängende Whitespaces werden als einfache Leerstelle gedeutet. Werden mehrere Kommandos übergeben, trennt man sie durch Semikolons voneinander. Argumente, die einen Boolean-Wert verlangen, nehmen entweder 1 für TRUE oder 0 für FALSE an. Wird ein String verlangt, muss dieser von doppelten Anführungszeichen umgeben sein.

Das Kommando `flush` signalisiert, dass der Kommando-String beendet ist und er zur Verarbeitung an das GLGr-Modul weitergegeben werden kann. Dies ist nötig, weil die Attributliste bei der Übermittlung von Datensätzen so lang sein kann, dass der TCP-Buffer hierfür nicht ausreicht. Das Kommando muss daher aufgeteilt und die einzelnen Segmente nacheinander verschickt werden.

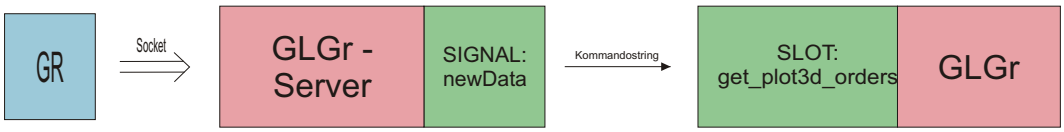


Abbildung 5.7: Kommunikation GR mit GLGr

Das GLGr-Modul hat seine eigene ’event loop’, in der alle Ereignisse vom Windowsystem und anderen Quellen behandelt werden. Neue Befehle vom GR werden asynchron hierzu gesendet, es muss also eine Möglichkeit gefunden werden, diese Daten zu empfangen.
Hierzu wird ein `QSocketNotifier` verwendet, der den TCP-Socket auf eingehende Daten überwacht. Er läuft asynchron zum Hauptprogramm und ist Bestandteil der `QServerSocket` Klasse, von der die Klasse `glgrserver` abgeleitet wurde. Erkennt der `QSocketNotifier`, dass neue Daten vorliegen, werden diese im `glgrserver` empfangen und zu einer Zeichenkette zusammengefügt. Gleichzeitig

wird überprüft, ob das `flush`-Kommando gesendet wird. Ist dies der Fall, emittiert der glgrserver das SIGNAL `newData`, mit dem gleichzeitig die empfangene Zeichenkette übergeben wird. Diese wird vom GLGr-Modul empfangen und dort verarbeitet.

Auf den beiden folgenden Seiten sind die verfügbaren Kommandos aufgelistet.

`data3d: <int> <int>` `<float> ... <float>`	Setzen der Oberflächendaten. Die beiden Integer-Werte geben die Dimension des Gitters an. Es folgen die Daten.
`dataCC: <int> <int>` `<float> ... <float>`	Setzen der Farbcodierung. Die beiden Integer-Werte geben die Dimension des Gitters an. Es folgen die Farbdaten.
`useCCdata: <bool>`	Oberfläche mit Farbcodierung einfärben.

`flipx: <bool>`
`flipy: <bool>`
`flipz: <bool>`
	Umdrehen der x-,y- bzw. z-Achse

`title: <string>`	Titel. Wird über der Grafik eingeblendet.
`subtitle: <string>`	Untertitel. Wird über der Grafik eingeblendet.
`xmin: <int>`	Untere Grenze der X-Achse.
`xmax: <int>`	Obere Grenze der X-Achse.
`ymin: <int>`	Untere Grenze der Y-Achse.
`ymax: <int>`	Obere Grenze der Y-Achse.
`zmin: <int>`	Untere Grenze der Z-Achse.
`zmax: <int>`	Obere Grenze der Z-Achse.
`cmin: <int>`	Untere Grenze der Farbcodierung.
`cmax: <int>`	Obere Grenze der Farbcodierung.

`xtick: <float>`
`ytick: <float>`
`ztick: <float>`
	Abstand der Teilstriche auf der entsprechenden Achse. Es gibt keine Änderung, wenn der Wert 0 ist.

`majorx: <int>`
`majory: <int>`
`majorz: <int>`
	Gibt das Intervall auf der entsprechenden Achse an, in dem die Teilstriche beschriftet werden. Überlappen Texturen, wird das Intervall automatisch reduziert.

`ticksize: <float>`	Größe der Achsen-Teilstriche in Prozent der Displaygröße
`xlabel: <string>`	Beschriftung der X-Achse
`ylabel: <string>`	Beschriftung der Y-Achse
`zlabel: <string>`	Beschriftung der Z-Achse
`rotation: <int>`	Rotationswinkel der Grafik
`tilt: <int>`	Kippwinkel der Grafik

`surface: <bool>`	Oberfläche zeichnen
`contour: <bool>`	Kontur zeichnen
`clines: <bool>`	Höhenlinien zeichnen
`numclines: <int>`	Anzahl der Höhenlinien
`axes: <bool>`	Achsen zeichnen
`orientation: <bool>`	Orientierungspfeile anzeigen
`legend: <bool>`	Farblegende anzeigen
`logx: <bool>`	X-Achse logarithmisch darstellen
`logy: <bool>`	Y-Achse logarithmisch darstellen
`logz: <bool>`	Z-Achse logarithmisch darstellen
`xytype: <int>`	Art der Oberflächendarstellung: 0: Kein Gitter 1: Schichtmodell in y-Richtung 2: Schichtmodell in x-Richtung 3: Gitter
`gridcolor: <int>`	Farbe des Gitters ändern: 0: schwarz, 1: rot 2: grün, 3: blau 4: hellblau, 5: gelb 6: violett, 7: grau
`reduction_rate: <int>`	Erlaubter Wert liegt zwischen 1 und 100 und ist der Prozentsatz, um den die Gittergröße reduziert werden soll. Die Daten werden auf ein neues Gitter interpoliert.(s. Seite 21)
`customized: <axis> <float> <string>`	individuelle Beschriftung der Achse: `<axis>` kann `x`, `y` oder `z` sein. Die Position wird als `<float>` übermittelt. `<string>` ist der alternative Text.
`colormap: <int>`	Ändern der Farbpalette: 0: Uniform, 1: Temperature 2: Grayscale, 3: Glowing 4: Rainbow, 5: Geologic 6: Greenscale, 7: Cyanscale 8: Bluescale, 9: Magentascale 10: Redscale, 11: Flame
`print:`	aktuelle Darstellung drucken
`flush`	Kommandoende

Kapitel 6

Anwendungsbeispiele

6.1 Daten aus der Kleinwinkelstreuung

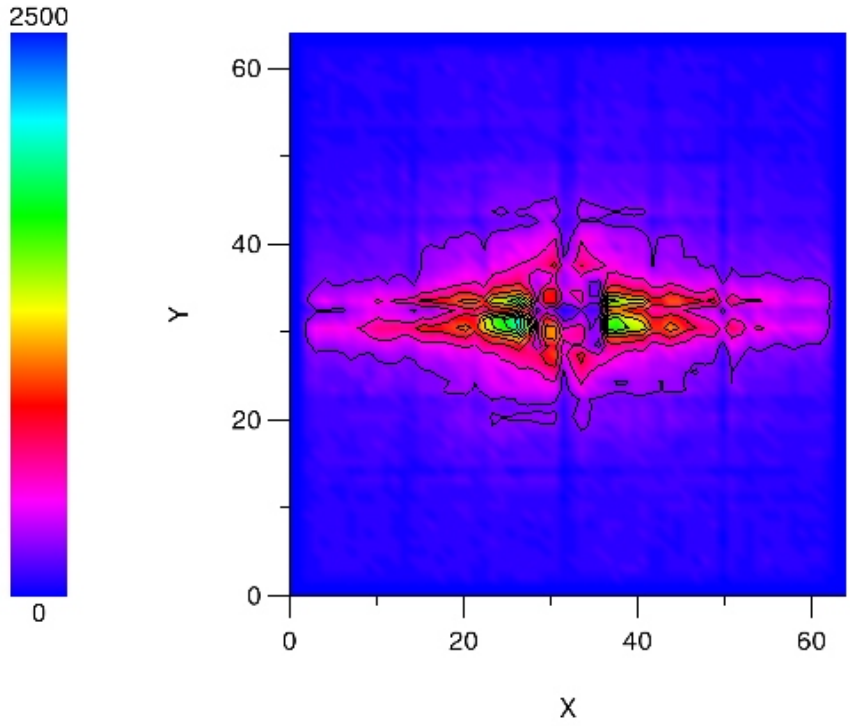


Abbildung 6.1: Konturdarstellung KWS-Daten, Intensität als Funktion von x und y. Die Intensität ist farblich dargestellt und die Stärke an der beigefügten Farbskala abzulesen.

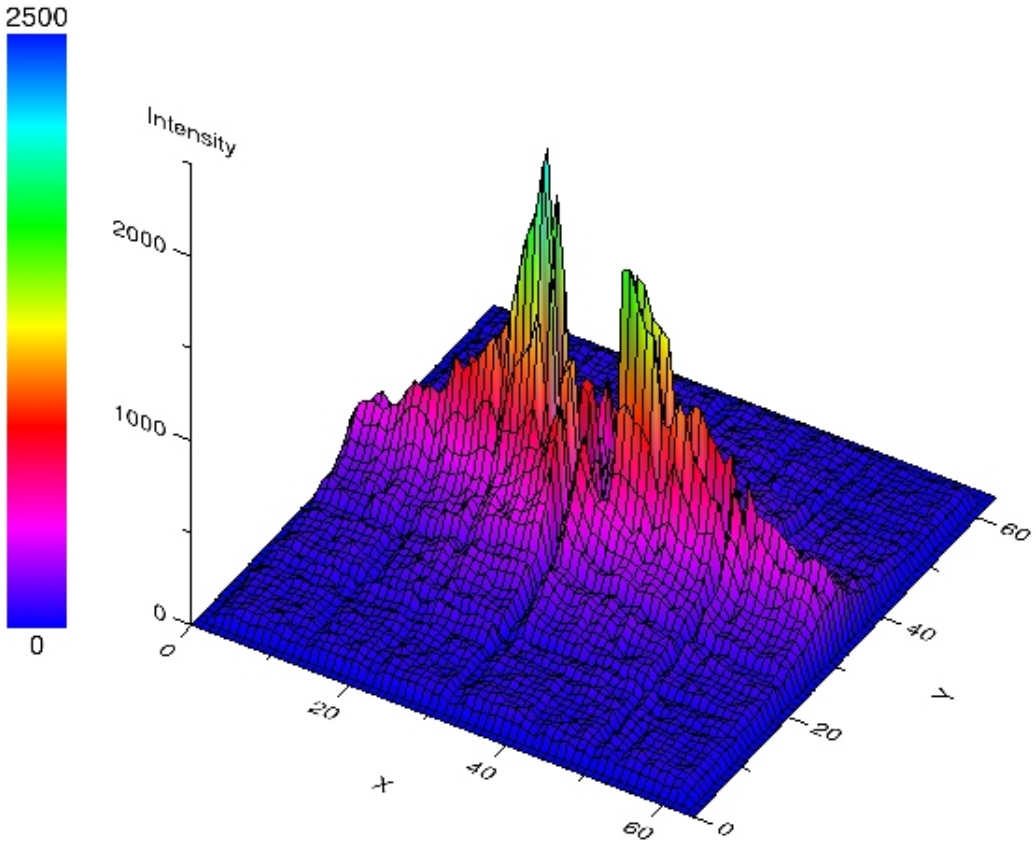


Abbildung 6.2: Oberflächen-Gitterdarstellung von KWS-Daten, Intensität als Funktion von x und y. Die Intensität ist dargestellt: a) räumlich und gleichzeitig b) farblich, wobei die Intensitätsfarbe auf der Oberfläche direkt eingetragen ist. Die Stärke ist an der beigefügten Farbskala abzulesen. Durch die Einfärbung wird die wegen der räumlichen Perspektive entstehende Unsicherheit, welche Intensität tatsächlich vorliegt, deutlich reduziert.

6.2 Diffuse Neutronenstreuung

Die Software DNS-Live nutzt das Schichtmodell des GR, um gemessene Daten zu visualisieren.

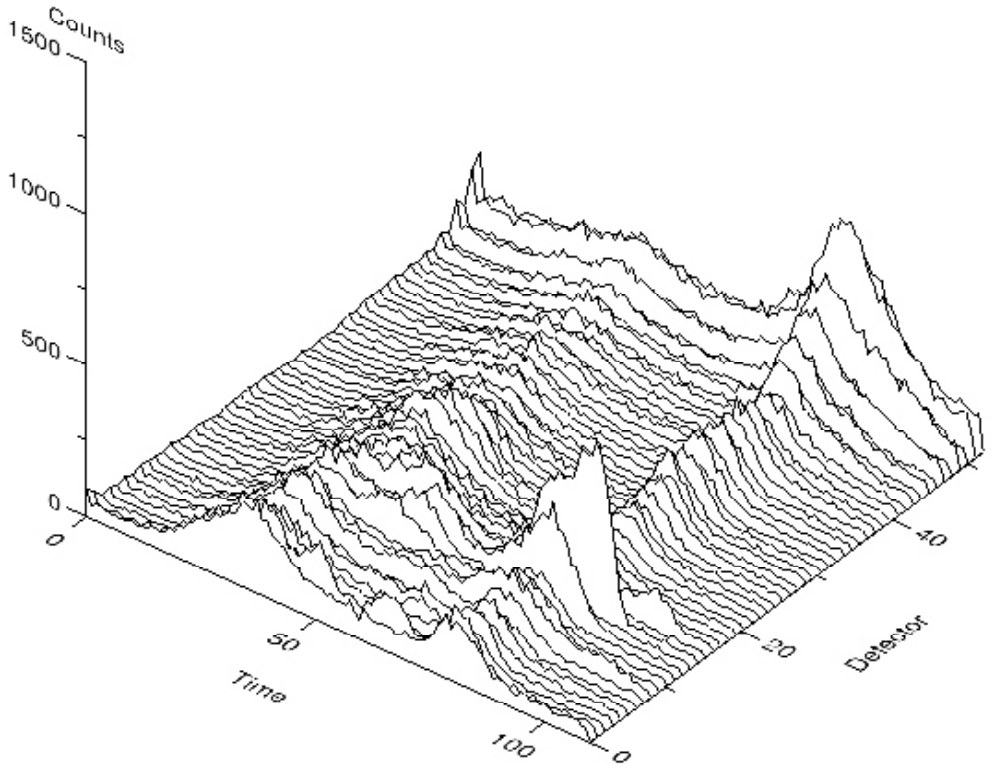


Abbildung 6.3: Schichtmodell DNS-Daten. Zu den einzelnen Sensoren des Experiments werden auf einer zeitlichen Achse die dort gemessenen Neutronen abgebildet. Das Schichtmodell ermöglicht es, jeden Sensor getrennt zu betrachten.

6.3 Funktionsplot mehrerer Veränderlicher

Plot der Funktion $f(x, y) = x \cdot y \cdot sin(x) \cdot cos(y)$.

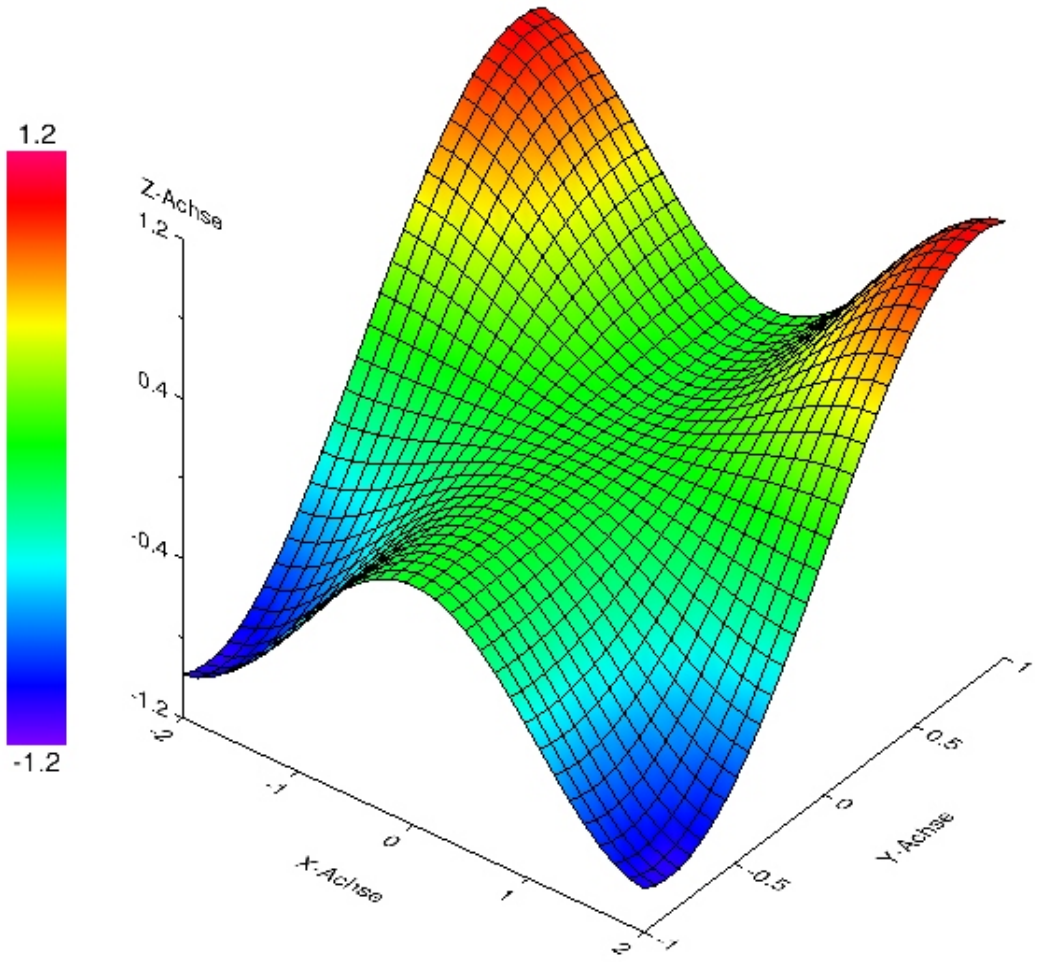


Abbildung 6.4: Die Funktion f(x,y) wird sowohl räumlich als auch farblich dargestellt. Die abgebildete Farbskala beschreibt die der Färbung zugehörigen Werte.

6.4 Geographie

Das Bild zeigt einen Gebirgszug im Colorado:

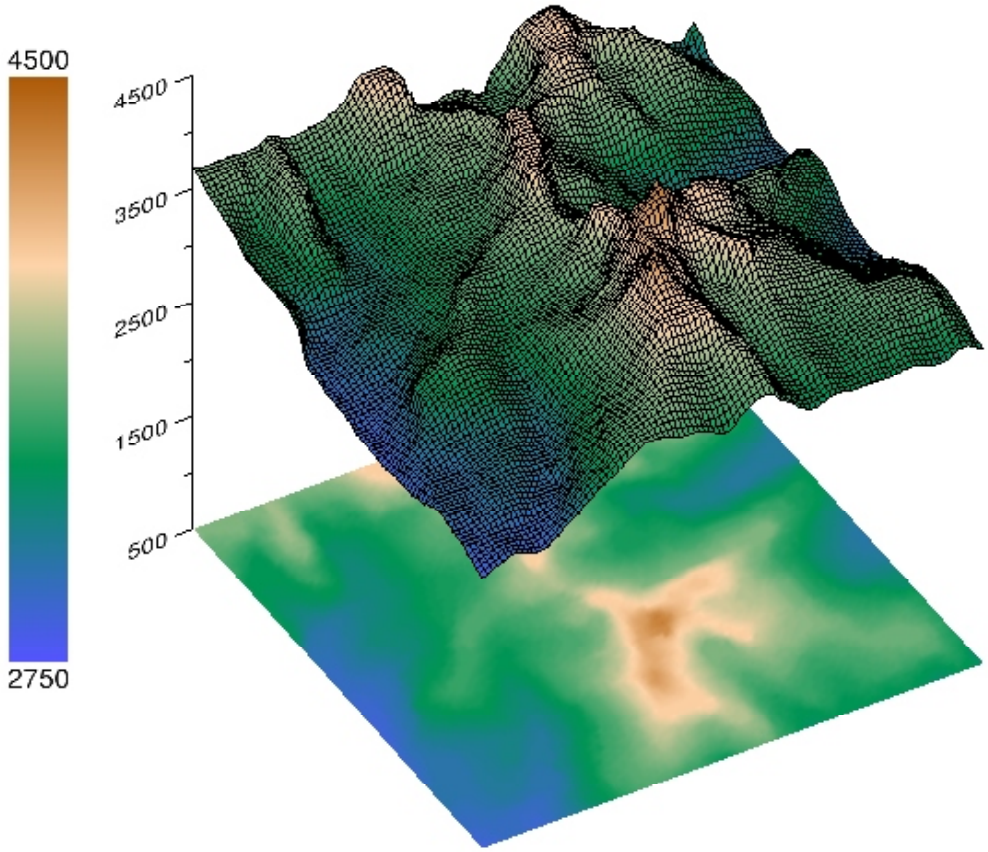


Abbildung 6.5: Mit GLGr lassen sich auch Landschaften zeichnen. Die Höhenwerte werden sowohl räumlich als auch farblich dargestellt. Die Farben sind einmal direkt auf der Oberfläche eingetragen, das andere Mal auf der xy-Ebene als Konturdarstellung.

Kapitel 7

Zusammenfassung

Das neue GLGr-Modul konnte im IFF bereits in die bestehende GLI-Struktur integriert werden und wurde erfolgreich getestet. Die bisherige Funktionalität wird weiterhin gewährleistet, zusätzlich sind einige Verbesserungen hinzugekommen:

- Die Betrachtung der Daten im Raum kann nun in beliebigen Winkeln gekippt und gedreht werden, eine Beschränkung auf lediglich 90° ist nicht mehr vorhanden.
- Mit Hilfe eines weiteren Datensatzes kann die Oberfläche nun unabhängig von den Daten für die räumliche Darstellung gefärbt werden.
- Schriften im Raum sind frei skalierbar und entsprechen der Qualität von Outline-Fonts.
- Die Datenmenge kann zur Beschleunigung der Darstellung durch Interpolation reduziert werden.

Ein besonderes Augenmerk liegt auf dem Geschwindigkeitsvorteil des neuen Moduls. Dieser wurde in einem dadurch erreicht, dass mit OpenGL eine hocheffiziente Bibliothek zur 3D-Visualisierung verwendet wurde, die die Ausnutzung einer vorhandener Hardwarebeschleunigung ermöglich. Andererseits arbeitet GLGr effizienter mit den darzustellenden Datensätzen. Während in der alten Version vor jeder Darstellung die Daten neu übermittelt werden mussten, benötigt GLGr dies nur bei Änderungen der Daten. Besonders im Netzwerk bringt dies eine erhebliche Beschleunigung.

Es wurden einige Tests durchgeführt, um die Unterschiede in der Geschwindigkeit zu verdeutlichen. Inhalt war, Beispieldatensätze zu visualisieren und die hierfür benötigte Zeit zu ermitteln. Als Datensatz dienten ein 128x128 und ein 1000x1000 Gitter.

Die Bilder wurden hierbei in Einerschritten um jeweils 90° Grad rotiert, insgesamt mussten folglich 90 Bilder dargestellt werden.

Als Testsystem diente ein Athlon XP 2600+ Prozessor mit 1GB RAM und einer NVIDIA GeForce FX5200 Grafikkarte. Untersucht wurde die Laufzeit im bisherigen GLI und im Vergleich hierzu im GLI mit dem neuen GLGr-Modul. Hierbei wurden die Tests einmal mit und einmal ohne aktivierter Hardwarebeschleunigung durchgeführt. Folgendes Diagramm zeigt die Ergebnisse der Messungen:

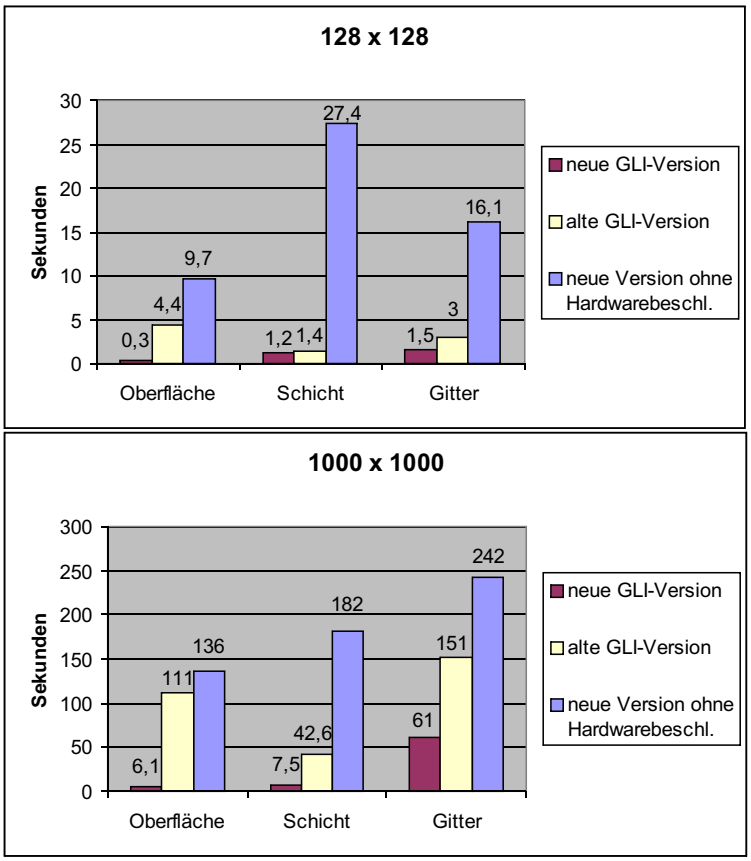


Abbildung 7.1: Ergebnisse des Geschwindigkeitstests in Sekunden

Man sieht, dass die neue Versionen der alten in allen Tests überlegen ist. Besonders bei der Darstellung von Oberflächen kann die Geschwindigkeit um mehr als das Zehnfache gesteigert werden.
Um zu vergleichen, welchen Vorteil das Auslagern der Rechenarbeit auf die Grafikkarte bringt, wurde der Test des neuen GLI bei deaktivierter Hardwarebeschleunigung durchgeführt. Hier ist zu beobachten, dass die Beschleunigung teilweise eine Steigerung der Effizienz auf über das Zwanzigfache bringt.

Anhang A

Funktionsübersicht

Existierende Anwendungen, die den GLI verwenden, können aufgrund der Plugin-Technik ohne Änderungen transparent weitergenutzt werden. Alle in diesem Kapitel beschriebenen Funktionen sind Bestandteil des GLGr-Modul und wurden für die Arbeit neu entwickelt. Sie werden ausschliesslich vom GLGr Modul selbst verwendet, um die geforderten Funktionalitäten zur Verfügung zu stellen (es besteht die Möglichkeit sie auch als Objektmethoden zu verwenden, was jedoch im primären Anwendungsfall nicht vorgesehen ist).

Zufgriffsfunktionen:

- `void addCustomizedLabel(int axis, float value, const QString &text)`

  Der Achse `axis` wird an der Position `value` die Beschriftung `text` zugeordnet. Alle individuellen Beschriftungen werden auf einen Vektor abgelegt und können mit `resetCustomizedLabel` wieder entfernt werden.

- `void calc_autoscale()`

  bestimmt die Minimal- und Maximalwerte der darzustellenden Daten

- `float colorData(int i, int j)`

  gibt den Farbwert des Datensatzes in Reihe `i`, Spalte `j` unter Berücksichtigung von Spiegelungen zurück

- `void createColorTable(int ct)`

  Es wird die dem Wert `ct` zugeordnete Farbpalette von 72 Farben erzeugt und auf den Vektor `colorTable` gespeichert.

- `void createContourIndices()`

  Es werden die Index-Vektoren erzeugt, die angeben, in welcher Reihenfolge die Punkte in `contourVertices` zum Aufbau der Konturoberfläche verwendet werden.

- `void createContourLineVertices()`

  Es werden mit Hilfe der Funktion find_way Höhenlinien erzeugt und auf den Vektor `contourLineVertices` gespeichert.

- `void createContourVertices()`

  Die Funktion berechnet die Punkte der Konturoberfläche und speichert diese auf den Vektor `contourVertices`.

- `void createGridIndices()`

  Es werden die Index-Vektoren erzeugt, die angeben, in welcher Reihenfolge die Punkte in `surfaceVertices` zum Aufbau des Gitters verwendet werden.

- `void createSurfaceColors()`

  Zu den Punkten der Oberfläche werden Farbinformationen erstellt. Diese richten sich entweder nach den Höhenwerten der Oberfläche oder nach einem separaten Datensatz. Auf eine Palette von insgesamt 72 Farben werden die Werte je nach Einstellung entweder linear oder logarithmisch verteilt und auf den Vektor `surfaceColors` gespeichert.

- `void createSurfaceIndices()`

  Es werden die Index-Vektoren erzeugt, die angeben, in welcher Reihenfolge die Punkte in `surfaceVertices` zum Aufbau der Oberfläche verwendet werden.

- `void createSurfaceVertices()`

  Die Funktion berechnet die Punkte der Oberfläche im Raum und speichert diese auf den Vektor `surfaceVertices`.

- `void createTextureArray(float min,float max,float tick,int major, vector<TickTexture> & textures, vector<CustomizedLabel> *indiv,bool is_log)`

  Mit Hilfe dieser Funktion wird ein Vektor aus Texturen erzeugt, die man zum Beschriften einer Achse benötigt. `min` und `max` geben das Intervall an, das durch die Achse dargestellt wird. `tick` ist der Abstand zwischen den Teilstrichen, `major` gibt an, wie weit beschriftete Teilstriche voneinander entfernt sein sollen. Ist `is_log` = true, wird die Achse logarithmisch dargestellt. `indiv` zeigt auf einen Vektor, der die individuelle Beschriftung der Achse

enhält, sofern sie verlangt wird. Anhand dieser Informationen werden entsprechende Texturen erzeugt und auf den Vektor `textures` abgelegt.

- `float data(int i, int j)`

  gibt den Wert des Datensatzes in Reihe `i`, Spalte `j` unter Berücksichtigung von Spiegelungen zurück

- `void drawTexturedRect(const vek& pos, vek xdir, vek ydir, float width, float height, float xcut, float ycut)`

  Funktion, die das Zeichnen einer Textur übernimmt. Die Textur wird auf eine Projektionsfläche gezeichnet, deren Mittelpunkt sich in `pos` befindet und die sich in die Richtungen `xdir` und `ydir` erstreckt. Die Werte `height`, `width`, `xcut` und `ycut` geben die Größe der Textur und die Schnittpunkte an, auf die die Textur zurechtgeschnitten werden soll.

- `void find_way(int x, int y, int direction, float value)`

  Funktion zum Zeichnen einer Konturlinie (siehe Kapitel 'Zeichnen von Höhenlinien', Seite 25)

- `size_t getNextTag(const QString &txt,QString &attribute, int pos)`

  Aufteilen der Syntax des Kommandostrings, den GR an GLGr schickt. `txt` ist der zu entschlüsselnde Text. In `attribute` wird das nächste Element abgelegt, das ausgehende von der Position `pos` in `txt` gefunden wird. Rückgabewert ist die Position hinter dem neu gefunden Element im QString `txt`.

- `void get_plot3d_orders()`

  In dieser Funktion wird der Kommandostring, den GR an GLGr schickt, interpretiert und den Kommandos entsprechende Funktionen aufgerufen.

- `Texture load_texture(QString str, const QColor& color)`

  erzeugt eine Textur mit dem Text `str` in der Farbe `color`, schickt diese an OpenGL und gibt eine entsprechendes Objekt vom Typ `Texture` zurück. Standardfarbe ist schwarz.

- `float logXValue(float x)`

  berechnet zu `x` den Wert, den es unter Berücksichtigung von Spiegelungen und logarithmischer Darstellung in x-Richtung auf der Zeichnung einnimmt

- `float logYValue(float y)`

  berechnet zu `y` den Wert, den es unter Berücksichtigung von Spiegelungen und logarithmischer Darstellung in y-Richtung auf der Zeichnung einnimmt

- `void makeImage(QString str, GLubyte **texData, Texture *tex,const QColor& color)`

  Hilfsfunktion zum Erzeugen einer Textur und zum Konvertieren in ein OpenGL kompatibles Bitmap

- `int make_pot(int x)`

  ermittelt die zu x nächsthöhere Zweierpotenz

- `void paintArrow()`

  Funktion zum Zeichnen eines Pfeils

- `void paintAxes()`

  Zeichnen des Koordinatensystems

- `void paintAxis(const vek& start, const vek& direction, const vek& tick_direction, float tick, int major, float tick_size, float min, float max, const vector<TickTexture> & textures, bool flip, bool,is_log)`

  Zeichnen einer einzelnen Achse mit Startpunkt `start`, Richtung `direction` und Richtung der Teilstriche in `tick_direction`. `min` und `max` geben das Intervall an, das durch die Achse dargestellt wird. `tick` ist der Abstand zwischen den Teilstrichen, `major` gibt an, wie weit beschriftete Teilstriche voneinander entfernt sein sollen. Ist `is_log` = true, wird die Achse logarithmisch dargestellt. Bei `flip` = true wird die Achse in umgekehrter Richtung beschriftet. Zur Beschriftung werden die Texturen auf dem Vektor `textures` verwendet.

- `void paintContour()`

  Zeichnen der Kontur

- `void paintContourLines()`

  Zeichnen der Höhenlinien

- `void paintGrid()`

  Zeichnen des Gitter

- `void paintLegend()`

  Zeichnen der Farblegende

- `void paintOrientation()`

  Zeichnen der Orientierungspfeile

- `void paintSubTitle()`

  Zeichnen des Beschriftungsuntertitels

- `void paintSurface()`

  Zeichnen der Oberfläche

- `void paintTitle()`

  Zeichnen der Titel-Beschriftung

- `void provideCorrectDirections(const vek& r1, const vek& r2, vek& nr1, vek& nr2,GLfloat camMat[4][4])`

  Ausgehend von den Vektoren `r1` und `r2` werden die beiden Vektoren `nr1` und `nr2` so ausgerichtet , dass nach Transformation mit der Matrix `camMat` eine Textur, die in Richtung der Vektoren aufgespannt ist, weder spiegelverkehrt noch auf dem Kopf ist.

- `void rebin(float *a1, int m, int n, float *a2, int a, int b)`

  lineare Interpolation des Feldes in Vektor `a1` mit der Dimension `m x n` auf den Vektor `a2` mit der Dimension `a x b`

- `void reduceData(bool resample3D=true)`

  In dieser Funktion werden die Quelldaten auf einen kleineren Datensatz interpoliert. In `reduction_rate` steht der Prozentsatz, den die neue Größe im Verhältnis zu dem Quelldatensatz haben soll. Es wird sowohl der normale Datensatz als auch der `dataCC` - Datensatz verkleinert. Ist `resample3D` = false wird ausschliesslich `dataCC` verringert.

- `void resetCustomizedLabel(int axis)`

  Der Vektor, der die individuelle Beschriftung der Achse `axis` enthält, wird geleert.

- `int round_f_to_i(float x)`

  Runden von x auf eine ganze Zahl

- `void setAutoScale(bool b)`

  Ändern der Variable `autoscale` auf `b`. Ist `autoscale`=true, werden die Werte `zmin`, `zmax`, `cmin` und `cmax` automatisch ermittelt. Die min-Werte werden in dem Fall auf den kleinsten, die max-Werte auf den größten Wert des jeweiligen Datensatzes gesetzt.

- `void setCMax(float value); void setCMax(int value)`

  Setzen der Variable `cmax` auf `value`. Mit `cmax` wird der größte Wert des Farb-Datensatzes festgelegt.

- `void setCMin(float value); void setCMin(int value)`

  Setzen der Variable `cmin` auf `value`. Mit `cmin` wird der kleinste Wert des Farb-Datensatzes festgelegt.

- `void setColormap(int c)`

  Es wird Palette `c` als aktuelle Farbpalette ausgewählt. Zur Auswahl stehen folgende Paletten:
  0: Uniform, 1: Temperature, 2: Grayscale, 3: Glowing, 4: Rainbow, 5: Geologic, 6: Greenscale, 7: Cyanscale, 8: Bluescale, 9: Magentascale, 10: Redscale, 11: Flame

- `void setData(int xdim_, int ydim_, float *data_, float *dataCC_=NULL)`

  Es werden die darzustellenden Daten übergeben. `dataCC_` kann ein NULL-Pointer sein, falls keine Farbcodierungs-Daten angezeigt werden sollen. Mit `xdim_` und `ydim_` wird die Größe des Feldes angegeben.

- `void setFlipX(bool b)`

  Ändern der Variable `flipx` auf `b`. Ist `b`=true wird die Grafik in Richtung der X-Achse gespiegelt

- `void setFlipY(bool b)`

  Ändern der Variable `flipy` auf `b`. Ist `b`=true wird die Grafik in Richtung der Y-Achse gespiegelt

- `void setFlipZ(bool b)`

  Ändern der Variable `flipz` auf `b`. Ist `b`=true wird die Grafik in Richtung der Z-Achse gespiegelt

- `void setGridColor(int nr)`

  Die Farbe des Gitters wird auf die dem Wert der Variable `nr` entsprechende Farbe gesetzt. Folgende Farben stehen zur Verfügung:
  0: schwarz, 1: rot, 2: grün, 3: blau, 4: hellblau, 5: gelb, 6: violett, 7: grau

- `void setLogX(bool b)`

  Ändern der Variable `logX` auf `b`. Ist `logX`=true, wird die X-Achse logarithmisch skaliert.

- `void setLogY(bool b)`

  Ändern der Variable `logY` auf `b`. Ist `logY`=true, wird die Y-Achse logarithmisch skaliert.

- `void setLogZ(bool b)`

  Ändern der Variable `logZ` auf `b`. Ist `logZ`=true, wird die Z-Achse logarithmisch skaliert.

- `void setMajorX(int major)`

  Setzen der Variable `x_major_tick` auf `major`, soweit `major` ungleich 0 ist. `x_major_tick` gibt an, in welchen Abständen die Teilstriche der X-Achse beschriftet werden.

- `void setMajorY(int major)`

  Setzen der Variable `y_major_tick` auf `major`, soweit `major` ungleich 0 ist. `y_major_tick` gibt an, in welchen Abständen die Teilstriche der Y-Achse beschriftet werden.

- `void setMajorZ(int major)`

  Setzen der Variable `z_major_tick` auf `major`, soweit `major` ungleich 0 ist. `z_major_tick` gibt an, in welchen Abständen die Teilstriche der Z-Achse beschriftet werden.

- `void setPaintAxes(bool b)`

  Ändern der Variable `paint_axes` auf `b`. Ist `paint_axes`=true, wird ein Koordinatensystem eingezeichnet.

- `void setPaintContour(bool b)`

  Ändern der Variable `paint_contour` auf `value`. Ist `paint_contour`=true, wird eine Kontur gezeichnet.

- `void setPaintContourLineNumber(int value)`

  Ändern der Variable `contourLineNumber` auf `value`. Hierdurch wird die Anzahl der zu zeichnenden Höhenlinien angegeben.

- `void setPaintCustomizedLabel(int axis, bool b)`

  Ist `b`=true wird die Achse `axis` mit der eingestellten individuellen Beschriftung dargestellt.

- `void setPaintLegend(bool b)`

  Ändern der Variable `paint_legend` auf `b`. Ist `paint_legend`=true, wird eine Farblegende eingeblendet.

- `void setPaintOrientation(bool b)`

  Ändern der Variable `paintOrientation` auf `b`. Ist `paintOrientation`=true, wird in die linke untere Ecke ein Koordinatensystem eingezeichnet, das die aktuellen Rotationen veranschaulicht.

- `void setPaintSurface(bool b)`

  Ändern der Variable `paint_surface` auf `value`. Ist paint_surface=true, wird eine Oberfläche gezeichnet.

- `void setPaintXAxis(bool b)`

  Ändern der Variable `paintXAxis` auf `b`. Ist `paintXAxis`=true, wird die X-Achse eingezeichnet.

- `void setPaintYAxis(bool b)`

  Ändern der Variable `paintYAxis` auf `b`. Ist `paintYAxis`=true, wird die Y-Achse eingezeichnet.

- `void setPaintZAxis(bool b)`

  Ändern der Variable `paintZAxis` auf `b`. Ist `paintZAxis`=true, wird die Z-Achse eingezeichnet.

- `void setReductionRate(int rate)`

  Setzen der Variable `reduction_rate` auf `rate`. Mit `rate` wird angegeben, auf wieviel Prozent des ursprünglichen Datensatzes die Daten durch Interpolation reduziert werden.

- `void setRotation(int value)`

  Ändern der Variable `rotation` auf `value`. `value` gibt den Winkel an, um den die Grafik gedreht wird.

- ```
  void setSubTitle(const char *t);
  void setSubTitle(const QString &t)
  ```

  Mit `t` wird der Untertitel der Beschriftung im Kopf der Grafik festgelegt.

- `void setTickSize(float value)`

  Setzen der Variable `tick_size` auf `value`. Mit `tick_size` legt man die Länge der Teilstriche im Verhältnis zur Zeichenfläche fest. Ist `value` z.B. 0.05 haben die Teilstriche eine Länge von 5% der Seitenlänge der Zeichenfläche.

- `void setTilt(int value)`

  Ändern der Variable `tilt` auf `value`. `tilt` gibt den Winkel an, um den die Grafik gekippt wird.
  ```

- `void setTitle(const char *t);`
  `void setTitle(const QString &t)`

  Mit `t` wird die Beschriftung im Kopf der Grafik festgelegt.

- `void setUseCCdata(bool b)`

  Ändern der Variable `useCCdata` auf `b`. Ist `useCCdata`=true wird der Farbcodierungs-Datensatz zum Einfärben der Oberfläche verwendet, sofern er vorhanden ist.

- `void setXLabel(const char *t);`
  `void setXLabel(const QString &t)`

  Die X-Achse wird mit dem Text der Variable `t` beschriftet.

- `void setXMax(float value); void setXMax(int value)`

  Setzen der Variable `xmax` auf `value`. Mit `xmax` legt man den obersten Wert der X-Achse fest.

- `void setXMin(float value); void setXMin(int value)`

  Setzen der Variable `xmin` auf `value`. Mit `xmin` legt man den untersten Wert der X-Achse fest.

- `void setXTick(float tick); void setXTick(int tick)`

  Setzen der Variable `x_tick` auf `tick`, soweit `tick` ungleich 0 ist. `x_tick` gibt die Abstände der Teilstriche auf der X-Achse an.

- `void setXYType(int v)`

  Umstellen der Darstellung des Gitters. `v` steht für folgende Darstellungstypen:

  0 Es wird kein Gitter gezeichnet

  1 Zeichnen eines Schichtmodells in X-Richtung

  2 Zeichnen eines Schichtmodells in Y-Richtung

  3 Zeichnen eines rechteckigen Gitters

- `void setYLabel(const char *t);`
  `void setYLabel(const QString &t)`

  Die Y-Achse wird mit dem Text der Variable `t` beschriftet.

- `void setYMax(float value); void setYMax(int value)`

  Setzen der Variable `ymax` auf `value`. Mit `ymax` legt man den obersten Wert der Y-Achse fest.

- `void setYMin(float value); void setYMin(int value)`

  Setzen der Variable `ymin` auf `value`. Mit `ymin` legt man den untersten Wert der Y-Achse fest.

- `void setYTick(float tick); void setYTick(float tick)`

  Setzen der Variable `y_tick` auf `tick`, soweit `tick` ungleich 0 ist. `y_tick` gibt die Abstände der Teilstriche auf der Y-Achse an.

- `void setZLabel(const char *t);`
  `void setZLabel(const QString &t)`

  Die Z-Achse wird mit dem Text der Variable `t` beschriftet.

- `void setZMax(float value); void setZMax(int value)`

  Setzen der Variable `zmax` auf `value`. Mit `zmax` legt man den obersten Wert der Z-Achse fest.

- `void setZMin(float value); void setZMin(int value)`

  Setzen der Variable `zmin` auf `value`. Mit `zmin` legt man den untersten Wert der Z-Achse fest.

- `void setZTick(float tick); void setZTick(float tick)`

  Setzen der Variable `z_tick` auf `tick`, soweit `tick` ungleich 0 ist. `z_tick` gibt die Abstände der Teilstriche auf der Z-Achse an.

Literaturverzeichnis

[1] http://www.wikipedia.org

[2] Trolltech Homepage, Qt-Referenz
http://www.trolltech.com

[3] Jackie Neider, Tom Davis, Mason Woo
Opengl Programming Guide: The Official Guide to Learning Opengl, Release 1
Addison Wesley, 1993

[4] IFF-Institut für Festkörperforschung, Neutronenstreuung
http://www.fz-juelich.de/iff/e_ins/