Entwicklung einer Visualisierung von Messdaten mittels OpenGL

Michael Jung

26. Mai 2008

Zusammenfassung:

Im Rahmen dieses Praktikums wurde eine Visualisierungs-Anwendung für Messdaten entwickelt, die aus Experimenten an der windgescherten, freien Luft-Wasser-Grenzschicht gewonnen wurden. Auf der Wasseroberfläche können Skalar- und zweidimensionale Vektorfelder in Falschfarben dargestellt werden.

Inhaltsverzeichnis

1. Einleitung

Ziel des Projektpraktikums war es, eine interaktive Visualisierung von Messdaten mittels OpenGL zu entwickeln. Bei den Messdaten handelt es sich um Ergebnisse, die aus Experimenten an der windgescherten, freien Luft-Wasser-Grenzschicht gewonnen wurden. Auf der visualisierten Wasseroberfläche sollten beliebige Skalarfelder sowie zweidimensionale Vektorfelder in Falschfarben dargestellt werden können. Außerdem sollte es möglich sein, Oberflächenströmungen mit Hilfe von Streaklines zu veranschaulichen. Da die gemessenen Sequenzen mitunter auch recht lang werden können war des Weiteren ein Buffering zu implementieren, sodass die Sequenzen dynamisch nachgeladen werden können und nicht als Ganzes in den Speicher passen müssen. Das Programm war objektorientiert und mit Schnittstellen zur einfachen Erweiterung zu implementieren und sollte außerdem plattformunabhängig sein.

2. Programmfeatures

- Frei dreh- und zoombare Visualisierung von animierten Wellenoberflächen aus Höhen- und Neigungsdaten.
- Darstellung von Skalarfeldern auf der Wasseroberfläche in Falschfarben. Die Colormap ist frei wählbar.
- Darstellung von 2D-Vektorfeldern über das HSV-Farbmodell
- Veranschaulichung von 2D-Vektorfeldern durch Streamlines
- Kameraflüge über eine beliebige Anzahl von frei definierbaren Positionen hinweg
- Gepuffertes Nachladen längerer Sequenzen bei frei wählbarer Puffergröße

3. Übersetzen des Quellcodes

Zum Übersetzen des Quellcodes wird, neben OpenGL sowie der OpenGL Utility Library (GLU), die beide bei modernen Betriebssystemen mitgeliefert werden, das Free OpenGL Utility Toolkit (http://freeglut.sourceforge.net) benötigt. Das Übersetzen sollte allerdings auch mit der closed-source-Variante glut möglich sein.

3.1.Linux

Unter Linux muss gelinkt werden gegen:

- GL
- GLU
- glut

3.2.Windows

Unter Windows muss gelinkt werden gegen:

- opengl32.lib
- glu32.lib
- freeglut.lib

Das Programm muss als Konsolen-Applikation kompiliert werden.

4. Benutzung des Programms

4.1.Datenformat

In der jetzigen Version verlangt das Programm raw-Dateien. Die einzelnen Werte müssen vom Typ float sein. Es werden eine Datei mit Höhendaten, sowie zwei Dateien mit den x- und y-Neigungsdaten benötigt. Optional sind beliebig viele weiteren Dateien mit Skalarfeldern oder Datei-Paare mit den x- bzw. y-Werten eines zweidimensionalen Vektorfelds.

Colormaps für die Skalarfelder müssen im Ascii-Format sein. In einer Zeile müssen sich der Rot-, der Grün- und der Blauwert der Farbe befinden, durch Leerzeichen oder Tabulator getrennt. Die einzelnen Farbwerte müssen zwischen 0 und 1 liegen, die nächste Farbe muss sich in einer neuen Zeile befinden. Die Anzahl der Farbeinträge, also die Auflösung der Lookup-Tabelle, wird automatisch bestimmt.

4.2.Konfigurationsdatei

Um das Programm verwenden zu können muss als Parameter der Name einer Konfigurationsdatei angegeben werden. Ein Beispiel für eine solche Datei befindet sich im Anhang und liegt auch dem Programm bei. Die Werte in der Datei müssen immer durch das Tabulator-Zeichen von ihren Bezeichnern getrennt sein. Einzelne Zeilen können durch Voranstellen von '#' auskommentiert werden.

Am Anfang der Datei müssen stets Auflösung und Anzahl der Frames angegeben werden. Außerdem wird benötigt: Die Anzahl der Bilder pro Sekunde, die Größe des Pufferspeichers falls die Daten dynamisch geladen werden sollen, die Skalierung in z-Richtung sowie der z-Wert des Bodens der „Wasser-Box“. Die Standard-Einstellungen sollten hier jedoch in den meisten Fällen gute Ergebnisse liefern. Außerdem kann noch die Hintergrundfarbe verändert werden, sowie festgelegt werden ob die Animation der Daten zu Beginn bereits laufen soll.

Als nächstes müssen die Namen der Dateien mit den Höhen- sowie Neigungsdaten angegeben werden.

Danach können jetzt beliebig viele Skalar- bzw. Vektorfelder zur Überlagerung der Wasseroberfläche angegeben werden. Ein solches Feld muss immer umgeben sein von *OVERLAY_BEGIN* und *OVERLAY_END* (gefolgt von einem Tabulatorzeichen). Als *OVERLAY_TYPE* muss angegeben werden ob es sich um ein Skalarfeld (1) oder um ein 2D-Vektorfeld (2) handelt. Als nächstes müssen der Dateiname bzw. die Dateinamen angegeben

werden. Bei einem Skalarfeld wird hierzu *OVERLAY_FILE* verwendet, bei einem Vektorfeld bezeichnet *OVERLAY_FILE1* die Datei mit den x-Werten, *OVERLAY_FILE2* die Datei mit den y-Werten.

Bei einem Skalarfeld können nun der Maximal- und Minimalwert (*OVERLAY_VALUE_MAX*, *OVERLAY_VALUE_MIN*) angegeben werden, ansonsten wird die Datei einfach nach dem größten und kleinsten Wert durchsucht. Falls Buffering aktiviert ist, müssen hier Maximal- und Minimalwert angegeben werden. Außerdem können die Bezeichnungen für die Skala, die später eingeblendet wird, verändert werden (*OVERLAY_TAG_MAX*, *OVERLAY_TAG_MIN*). Sind hier keine eigenen Bezeichnungen angegeben werden Maximal- und Minimalwert verwendet. Außerdem muss mittels *OVERLAY_COLORMAP* die Datei mit der Loopup-Tabelle für die Farben angegeben werden.

Bei einem Vektorfeld kann ebenfalls die maximale Vektorlänge (*OVERLAY_VALUE_MAX*) angegeben werden. Ansonsten wird der längste Vektor aus den Daten ermittelt. Falls Buffering aktiviert ist, ist die Angabe hier ebenfalls notwendig.

Am Ende der Konfigurationsdatei können nun noch Kameraflüge erstellt werden. Hierzu muss *FLIGHT_ENABLED* auf 1 gesetzt werden. Danach muss mindestens ein Kamerapunkt festgelegt werden. Ein einzelner Kamerapunkt muss umgeben sein von *CAMERA_BEGIN* und *CAMERA_END* (gefolgt von einem Tabulatorzeichen). Es müssen die Position der Kamera, die x- und y-Koordinaten des Ziels sowie die Flugdauer zum nächsten Kamerapunkt angegeben werden.

4.3.Bedienung

Die Bedienung über Tastatur und Maus kann bei laufendem Programm durch Drücken von F1 eingeblendet werden.

Bei gedrückter linker Maustaste ist es möglich, die Darstellung zu rotieren, mit mittlerer Maustaste kann gezoomt werden. Durch Drücken der rechten Taste kann die Kamera auf ein neues Ziel zentriert werden. Die Leertaste startet und stoppt die Animation. Mit *'o'* können die Overlay-Felder an- und abgeschaltet werden; mit hoch und runter auf den Pfeiltasten werden die einzelnen Felder durchgeschaltet. Mit rechts und links kann man manuell durch die einzelnen Frames laufen.

Streaklines werden aktiviert, indem man bei gedrückter Strg-Taste und aktiviertem 2D-Vektorfeld an die gewünschte Stelle klickt. Zum Deaktivieren muss lediglich an eine Stelle außerhalb der Wasseroberfläche geklickt werden.

5. Funktionsweise des Programms

Die einzelnen Programmfunktionen, wie Darstellung, Einlesen der Daten, Parsen der Konfigurationsdatei, usw., sind auf verschiedene Klassen aufgeteilt.

Die Klasse *wave* ist für die graphische Darstellung zuständig. Bei der Instanzierung muss ihr eine Instanz der Klasse *waveConfig* übergeben werden, in der alle benötigten Daten gespeichert sind. In diesem Konfigurations-Objekt befindet sich, neben einigen Variablen wie Auflösung und Zahl der Frames, eine Liste mit Objekten vom Typ *DataSource*, die die Klasse *wave* mit den Höhen-, Neigungs- und Überlagerungsdaten versorgen. Des Weiteren befinden sich in dieser Liste auch Objekte vom Typ *colorMap*, falls Skalarfelder visualisiert werden sollen. Deren Routinen werden von *wave* aufgerufen, wenn festgestellt werden soll, welche Farbe ein bestimmter Wert des Skalarfeldes erhalten soll. Im *waveConfig*-Objekt befindet sich außerdem noch eine Liste mit den Punkten die bei einem eventuellen Kameraflug angeflogen werden sollen.

Bei *DataSource* und *colorMap* handelt es sich um Basisklassen, die nicht selbst instanziert werden können. Mit der Kindklasse *DataSourceRaw* können raw-Dateien als Ganzes eingelesen werden. Die Klasse *DataSourceRawRingBuffer* ermöglicht das gepufferte Einlesen von raw-Dateien. Mit der Kindklasse *colorMapAsc* ist es möglich Ascii-Colormaps einzulesen.

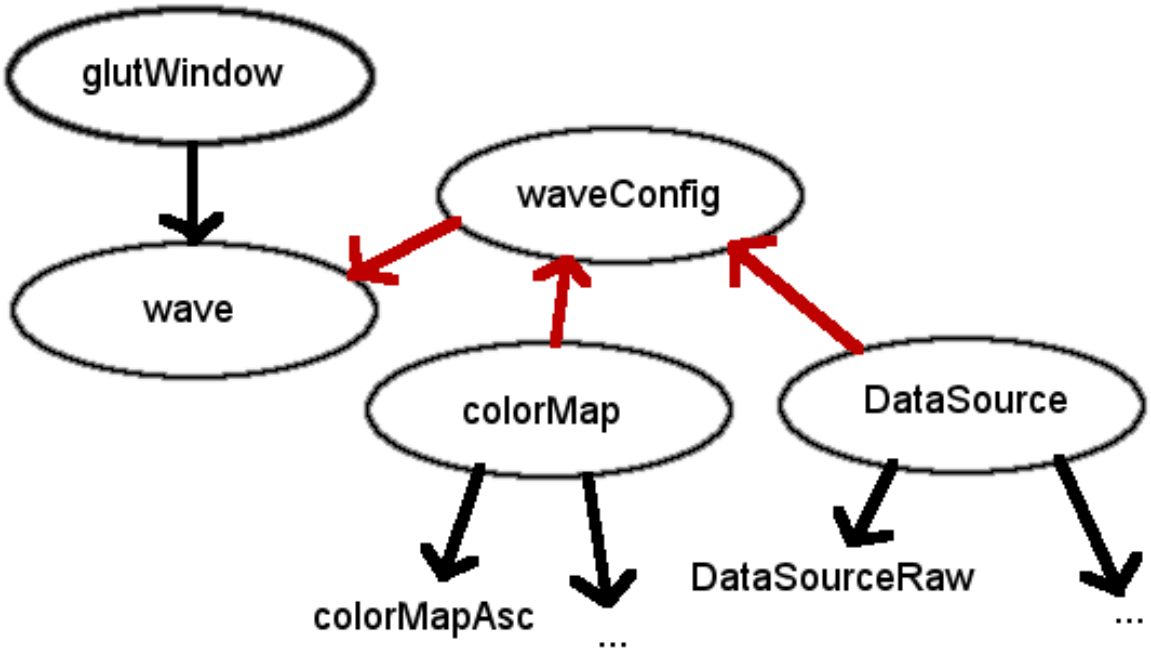

Bild 1: Funktionsweise des Programms. Schwarze Pfeile: Vererbung; rote Pfeile: Objekt wird als Pointer übergeben.

6. Beschreibung der einzelnen Klassen und Routinen

6.1.glutWindow

Hierbei handelt es sich um die Elternklasse von *wave*. Sie ist dafür zuständig, dass die Callback-Aufrufe von glut auf die richtigen Member-Funktionen umgelegt werden. In der map *windowIDToWindow* werden hierzu alle Instanzen mit der zugehörigen WindowID gespeichert. Wenn nun in der *wave*-Klasse ein Callback registriert wird, muss ein Zeiger auf die zugehörige Dispatcher-Funktion übergeben werden, da die glut-Routinen mit Member-Function-Pointern nicht umgehen können.

6.2.wave

Diese Klasse ist für die graphische Darstellung sowie die Interaktion mit dem Benutzer zuständig.

Die Routinen, die im Zusammenhang mit der graphischen Darstellung stehen befinden sich allesamt in der Datei *wavedisplay.cpp*. Die wichtigste Methode dort ist *wave::display()*, die immer dann aufgerufen wird, wenn das Bild neu gezeichnet werden muss.

In der Datei *waveinteraction.cpp* befinden sich sämtliche Methoden, die mit Maus- oder Tastatur-Ereignissen zu tun haben.

Alle übrigen Funktionen, befinden sich in *waveother.cpp*. Hier befinden sich Konstruktor und Destruktor, sowie die Funktionen die zum Frame-Wechsel benötigt werden. Außerdem werden hier die Streaklines und die Kameraflüge berechnet.

Dem Konstruktor muss ein Objekt vom Typ *waveConfig* übergeben werden, in dem alle zur Darstellung benötigten Informationen stehen.

Die Daten des aktuellen Frames sind immer in den Arrays *height*, *slopeX*, *slopeY*, *overlayDim1* und *overlayDim2* gespeichert. Beim Frame-Wechsel werden diese von den Datenquellen neu gefüllt.

In den Vektoren *streakline_forward* und *streakline_backward* wird die zur Zeit dargestellte Streakline gespeichert.

6.3.waveConfig

Diese Klasse beinhaltet alle Variablen, die zur Darstellung wichtig sind. Dem *wave*-Konstruktor muss eine Instanz dieser Klasse übergeben werden. Sie beinhaltet neben der Auflösung und der Anzahl der Frames auch Werte für die Hintergrundfarbe, die Skalierung, usw. In der Liste *dataList*

sind sämtliche Datenquellen und evtl. zugehörige Colormaps gespeichert, in der Liste *cameraFlight* befinden sich alle Punkte des Kameraflugs.

Außerdem beinhaltet diese Klasse einen function pointer, der mit der Methode *idleFunc* gesetzt bzw. gelesen werden kann. Diese Funktion wird immer dann von *wave* aufgerufen, wenn gerade nichts zu tun ist. Ihr wird das Konfigurations-Objekt der gerade aktiven *wave*-Instanz übergeben.

6.4.DataSource

Dies ist die Basisklasse für Datenquellen, die von *wave* benötigt werden um an die darzustellenden Daten zu gelangen. In den abgeleiteten Klassen müssen die Methoden *nextFrame*, *currentFrame* und *lastFrame* in verschiedenen Ausführungen implementiert werden. *NextFrame* und *lastFrame* werden immer dann aufgerufen, wenn ein anderer Frame angezeigt werden soll. Bei den Datenquellen, die gerade benötigt werden, wird immer die Variante aufgerufen, bei der ein Zeiger, bzw. zwei Zeiger bei zweidimensionalen Vektorfeldern, übergeben werden. Ist die Datenquelle gerade nicht in Verwendung, weil beispielsweise gerade ein anderes Skalar- oder Vektorfeld angezeigt wird, wird lediglich die Variante ohne Parameter aufgerufen. Die übergebenen Zeiger zeigen auf ein Array von Floats, das genau die Größe eines einzelnen Frames hat. Die Funktion *nextFrame(int n)*, sowie die beiden Varianten bei denen zusätzlich noch ein bzw. zwei Zeiger übergeben werden, werden immer dann aufgerufen, wenn einzelne Frames beim Abspielen übersprungen werden. Dies tritt auf, wenn die Anzahl der Frames pro Sekunde relativ hoch ist. Diese Funktionen können in der Kindklasse auch selbst implementiert werden, falls zum Beispiel eine Art von Interpolation gewünscht ist.

Falls es sich bei der Datenquelle um ein Skalarfeld handelt, muss dafür gesorgt werden, dass die Werte so skaliert werden, dass sich der Wertebereich, der später farblich dargestellt werden soll, zwischen 0 und 1 befindet. Bei einem zweidimensionalen Vektorfeld müssen die Daten so skaliert sein, dass der längste Vektor die Länge 1 hat; längere Vektoren werden bei der Darstellung genauso aussehen wie Vektoren der Länge 1.

Zur korrekten Darstellung von Streaklines werden bei deren Berechnung interpolierte Werte zwischen den einzelnen Bildpunkten benötigt. Dazu muss die Funktion *interpolate* implementiert werden, wobei darauf zu achten ist, dass hier die Werte wieder unskaliert ausgegeben werden müssen. *Interpolate* wird nur bei zweidimensionalen Vektorfeldern aufgerufen.

Die Funktion *idle* kann verwendet werden, falls diese Datenquelle im Leerlauf bestimmte Operationen durchführen soll. Sie wird nicht automatisch aufgerufen. Der Aufruf kann aber beispielsweise von der Leerlauf-Funktion durchgeführt werden, die man im *waveConfig*-Objekt gespeichert hat.

InUse wird aufgerufen, falls eine unbenutze Datenquelle nun in Verwendung kommt, bzw. eine benutze Datenquelle jetzt nicht mehr benötigt wird.

6.5.DataSourceRaw

Dies ist die Implementation einer Datenquelle, die Daten im raw-Format als Ganzes in den Speicher lädt.

6.6.DataSourceRawRingBuffer

Hiermit können Daten im raw-Format gepuffert geladen werden. Die Puffergröße ist variabel und wird dem Konstruktor übergeben. Hiermit können auch große Sequenzen eingelesen werden.

6.7.colorMap

Dies ist die Basisklasse für Colormaps. Diese werden benötigt für die Falschfarben-Darstellung von Skalarfeldern. Es müssen die Methoden *lookUp*, *minTag* und *maxTag* implementiert werden. *LookUp* wird von *wave* aufgerufen, wenn die zu verwendende Farbe ermittelt werden soll. Als Parameter wird ein Wert zwischen 0 und 1, sowie ein Zeiger auf ein Float-Array mit drei Einträgen übergeben. *LookUp* muss nun den passenden Farbwert bestimmen und in das übergebene Array eintragen.

6.8.colorMapAsc

Hierbei handelt es sich um eine Implementation einer colorMap, die eine Lookup-Tabelle aus einer Ascii-Datei ausliest.

6.9.vector

Vector ist die simple Implementation eines dreidimensionalen Vektors, die für einige vektoralgebraische Operationen in den anderen Klassen benötigt wird.

7. Ausblick

Als nächsten Schritt könnte man aus den bereits vorhandenen Routinen eine Shared Library bzw. Dynamically Linked Library erstellen. Dazu müsste eine Initialisierungsfunktion geschrieben werden, die die Init-Befehle aus der bisherigen *main*-Funktion aufruft. In jener Funktion hätte außerdem dafür gesorgt zu werden, dass *glutMainLoop* als eigener Thread läuft, da diese Funktion nicht wieder verlassen wird. Das aufrufende Programm müsste dann selbst das benötigte *waveConfig*-Objekt erstellen und an diese Initialisierungsroutine übergeben.

Des Weiteren könnte man die Konfigurationsmöglichkeiten über die Config-Datei erweitern. Dazu müsste die *waveConfig*-Klasse um die gewünschten Einträge erweitert werden, die dann wiederum in *wave* verwendet werden, falls die Änderungen die Darstellung selbst betreffen. Falls die Änderungen nur die Daten betreffen, müssten entsprechende Anpassungen an den Datenquellen vorgenommen werden bzw. eine neue Datenquelle entwickelt werden. In beiden Fällen wäre der Parser in *main.cpp* um die entsprechenden Einträge zu erweitern.

Anhang A: Beispiel einer Konfigurationsdatei

```
#Wave visualization config file
#use tab button to seperate tag from value
#if you don't do so, you will get strange results or segmentation
#faults

#resolution and framenumber must not be placed after the file
#entries

#resolution of data
RES_X     320
RES_Y     149

#number of frames
FRAMES    1000
FRAMES_PER_SEC 30

#set this to 1 if the data should be loaded into RAM dynamically
#if activated be sure to set the max / min values for scalar
#overlay fields
BUFFERED  1
#size of the buffer in frames
BUFFER_SIZE    50

#scaling in z direction
SCALING   1.0

#z value for the water box's ground
GROUND_Z  -30.0

#set to 1 if animation shall be enabled at startup, 0 otherwise
ANIMATION 0

#color of the background in numbers between 0.0 and 1.0
#default is black, uncomment the following lines to change
BG_RED    1.0
BG_GREEN  1.0
BG_BLUE   1.0

#data files
FILE_HEIGHT    Run234_Height.raw
FILE_SLOPE_X   Run234_Sx.raw
FILE_SLOPE_Y   Run234_Sy.raw

#the begin and end lines also have to end with \t
OVERLAY_BEGIN
OVERLAY_TYPE   1
OVERLAY_FILE   Run234_IR.raw
OVERLAY_COLORMAP    jet_colormap.txt
#uncomment the following lines to set the colormap scale manually
#or if you use buffering
```

```
#OVERLAY_VALUE_MIN  22.5
#OVERLAY_VALUE_MAX  25.0
#uncomment the following lines if you want to change the
#colormap tags
#OVERLAY_TAG_MIN    273K
#OVERLAY_TAG_MAX    300K
OVERLAY_END

OVERLAY_BEGIN
OVERLAY_TYPE   2
OVERLAY_FILE1  velx.raw
OVERLAY_FILE2  vely.raw
#the following value is needed if buffering is used
#OVERLAY_VALUE_MAX  5.0
OVERLAY_END

#you can add other overlays here


####camera flights

#set this to 1 if camera flights should be enabled,
#set to 0 otherwise
FLIGHT_ENABLED 0

CAMERA_BEGIN
CAMERA_X  -100
CAMERA_Y  -100
CAMERA_Z  100
CAMERA_TARGET_X     160
CAMERA_TARGET_Y     75
#duration of the flight to the next camera position in ms
FLIGHT_DURATION     2000
CAMERA_END

CAMERA_BEGIN
CAMERA_X  250
CAMERA_Y  -100
CAMERA_Z  200
CAMERA_TARGET_X     160
CAMERA_TARGET_Y     75
FLIGHT_DURATION     2000
CAMERA_END

CAMERA_BEGIN
CAMERA_X  250
CAMERA_Y  420
CAMERA_Z  100
CAMERA_TARGET_X     160
CAMERA_TARGET_Y     75
FLIGHT_DURATION     2000
CAMERA_END
```

```
CAMERA_BEGIN
CAMERA_X  -100
CAMERA_Y  420
CAMERA_Z  200
CAMERA_TARGET_X     160
CAMERA_TARGET_Y     75
FLIGHT_DURATION     2000
CAMERA_END
```