Implementierung eines Tutorials für
OpenGL-Objekte (glut) mit Interpreter

Detaildokumentation

Andreas Volz, 157063

14. Dezember 2002

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Hinweise zur C-Programmierrichtline

Das Programm *glInterpret* ist grundsätzlich vom Design her **völlig** unabhängig von *glDesigner* entworfen worden. Diese Eigenschaft ermöglicht es *glInterpret* auch auf Systemen einzusetzten die nicht über das *gtk2-Toolkit* verfügen. Im Interpreter ist soweit möglich nur mit den **ANSI-** oder **ISO-C** definierten Sprachmitteln programmiert worden. Die Abhängigkeit von *OpenGL* ergibt sich natürlich Zwangsläufig aufgrund der Aufgabenstellung.

Anders sieht das bei den Modulen von *glDesigner* aus. Durch die Abhängigkeit vom *gtk2-Toolkit* ergibt sich automatisch auch die Abhängigkeit von *glibc*. Daduch sind in den Modulen von *glDesigner* meist die erweiterten *glibc*-Funktionen den normalen C-Funktionen vorgezogen worden. Da einige Module sowohl von *glDesigner* als auch von *glInterpret* verwendet wird, kommt es an diesen Stellen teilweise zu unschönen Vermischungen von *ANSI-C-*, *glibc-* und *OpenGL*-Datentypen. Durch geschicktes *casten* und die Tatsache, dass meist die Datentypen nur ein *typedef* auf den entspechenden *ANSI-C*-Datentyp sind, sollte dies kein Problem darstellen.

In fast allen Fällen ist versucht worden die C-Richtlinien so gut wie möglich umzusetzen. Eine Ausnahme stellen die C-Dateien *interface.c* und *support.c*, sowie deren *Headerdateien* dar. Das ist jedoch in der Modulbeschreibung noch einmal genau erläutert. Die einzige direkte Nichteinhaltung der Richtlinie bezieht sich auf das Semikolon nach jeder schließenden geschweiften Klammer. Da ich den Quellcode auch im weiteren Verlauf nach dem Ende der Lehrveranstaltung weiter entwickeln möchte schien mir die Einhaltung dieser Richtlinie nicht sehr sinnvoll.

1.2 Zusammenspiel der Programme

Da es sich beim Designer und Interpreter um zwei getrennte Programme handelt, wird beim berechnen der Szene aus *glDesigner* ein zweiter Prozess

mit *glInterpret* gestartet. Der zweite Prozess befindet sich in einem komplett getrennten Adressraum und besteht auch nach Beendigung von *glDesigner* fort. Es können auf *glDesigner* heraus geliebig viele *OpenGL*-Fenster gestartet werden. Alldings ist zu bedenken, dass die offenen Fenster beim Beenden von *glDesigner* weiterhin bestehen und entweder manuell oder duch Verwendung von *kill*, bzw. *killall* unter *UNIX* beendet werden sollten.

Kapitel 2

Kompilieren des Programms

Aufgrund der Tatsache, dass die *Makefiles* momentan noch nicht mit automake bzw. autoconf erzeugt werden ist der Kompiliervorgang recht unflexibel. In der nächsten Version des Programm wird sich das wesentlich verbessern. Die Pfade sind momentan für die CAE-Rechner der FH-Fulda angepasst. Die *Makefiles* und Quellen befinden sich alle im Unterverzeichnis `src`. Die jeweils zum kompilieren und binden genutzten Module können im Kapitel Modulbeschreibung genau nachgelesen werden.

Voraussetzung zum erfolgreichen kompilieren des Interpreters ist nur eine OpenGL-Implementierung, das OpenGL-Toolkit *GLUT* und ein geeignetes Entwicklungssystem. Für die *gtk2*-Oberfläche ist allerdings das *gtk2-Toolkit* mit den Entwicklerpaketen notwendig.

2.1 Kompilieren unter Linux

Das *Makefile* für Linux hat den Namen `Makefile`. Der Befehl um den Interpreter zu kompilieren ist `make glInterpret`. Dadurch beginnt der Kompilier- und Bindevorgang des Interpreters. Falls die Kompilation nicht funktioniert ist als erstes die Korrektheit der System-Pfade im *Makefile* zu überprüfen. Falls auf dem System nur Mesa OpenGL vorhanden ist, müssen auch Änderungen vorgenommen werden. Möchte man zusätzlich noch die *gtk2*-Oberfläche übersetzen, dann geschieht das durch den Aufruf von `make glDesigner`.

Als *Compiler* unter Linux ist *gcc 3.2* geeignet. Es spricht aber nichts dagegen die Programme auch mit einem *Compiler* kleinerer Versionsnummer zu kompilieren. Andere *Compiler* sind nicht getestet worden.

2.2 Kompilieren unter Solaris x86

Das *Makefile* für Solaris x86 hat den Namen Makefile.SunOS_x86. Der Befehl um den Interpreter zu kompilieren ist make -f Makefile.SunOS_x86 glInter-

pret. Dadurch beginnt der Kompilier- und Bindevorgang des Interpreters. Falls die Kompilation nicht funktioniert ist als erstes die Korrektheit der System-Pfade im *Makefile* zu überprüfen. Falls auf dem System nur Mesa OpenGL vorhanden ist, müssen auch Änderungen vorgenommen werden. Möchte man zusätzlich noch die *gtk2*-Oberfläche übersetzen, dann geschieht das durch den Aufruf von `make -f Makefile.SunOS_x86 glDesigner`.

Als *Compiler* unter Solaris x86 ist *gcc 2.95* geeignet. Es spricht aber nichts dagegen die Programme auch mit einem *Compiler* kleinerer Versionsnummer zu kompilieren. Andere *Compiler* sind nicht getestet worden.

2.3 Kompilieren unter Solaris Sparc

Das *Makefile* für Solaris Sparc hat den Namen `Makefile.SunOS_sparc`. Der Befehl um den Interpreter zu kompilieren ist `make -f Makefile.SunOS_sparc glInterpret`. Dadurch beginnt der Kompilier- und Bindevorgang des Interpreters. Falls die Kompilation nicht funktioniert ist als erstes die Korrektheit der System-Pfade im *Makefile* zu überprüfen. Falls auf dem System echtes OpenGL vorhanden ist, müssen auch Änderungen vorgenommen werden. Möchte man zusätzlich noch die *gtk2*-Oberfläche übersetzen, dann geschieht das durch den Aufruf von `make -f Makefile.SunOS_sparc glDesigner`. Der Kompiliervorgang von `glDesigner` konnte nicht getestet werden, da auf den Rechnern kein *gtk2* zur Verfügung stand.

Als *Compiler* unter Solaris x86 ist *gcc 2.95* geeignet. Es spricht aber nichts dagegen die Programme auch mit einem *Compiler* kleinerer Versionsnummer zu kompilieren. Andere *Compiler* sind nicht getestet worden.

2.4 Kompilieren unter IRIX

Das *Makefile* für SGI IRIX hat den Namen `Makefile.IRIX`. Der Befehl um den Interpreter zu kompilieren ist `make -f Makefile.IRIX glInterpret`. Dadurch beginnt der Kompilier- und Bindevorgang des Interpreters. Falls die Kompilation nicht funktioniert ist als erstes die Korrektheit der System-Pfade im *Makefile* zu überprüfen. Falls auf dem System echtes OpenGL vorhanden ist, müssen auch Änderungen vorgenommen werden. Das *Makefile* von IRIX ist nicht auf die Kompilation von `glDesigner` vorbereitet, da auf den Rechnern kein *gtk2* zur Verfügung stand. Durch die zukünftige Einführung von automake und autoconf wird das allerding möglich sein.

Als *Compiler* unter IRIX ist *gcc 2.95* geeignet. Es spricht aber nichts dagegen die Programme auch mit einem *Compiler* kleinerer Versionsnummer zu kompilieren. Andere *Compiler* sind nicht getestet worden.

2.5 Kompilieren unter Windows

Das *Makefile* für Microsoft Windows hat den Namen `Makefile.win32`. Der Befehl um den Interpreter zu kompilieren ist `make -f Makefile.win32 glInterpret`. Dadurch beginnt der Kompilier- und Bindevorgang des Interpreters. Falls die Kompilation nicht funktioniert ist als erstes die Korrektheit der System-Pfade im *Makefile* zu überprüfen. Falls auf dem System nur Mesa OpenGL vorhanden ist, müssen auch Änderungen vorgenommen werden. Möchte man zusätzlich noch die gtk2-Oberfläche übersetzen, dann geschieht das durch den Aufruf von `make -f Makefile.win32 glDesigner`.

Als *Compilerumgebung* unter Windows ist *DevC++* geeignet. Es spricht aber nichts dagegen die Programme auch mit einem anderen *Compiler* wie Borland C++ oder Microsoft Visual C++ zu kompilieren. Für diese *Compilier* sind allerdings weder *Makefiles* noch Projektdateien vorhanden.

2.6 Kompilieren unter anderen Betriebsystemen

Es sollte kein großes Problem darstellen die Programme für hier nicht genannte Betriebsysteme zu kompilieren. Die bereits erwähnten Abhängigkeiten von *OpenGL* und *gtk2* sind natürlich auch hier zu beachten. Desweiteres ist für *glDesigner* entweder eine Windows Prozessimplementation oder eine POSIX-Prozessimplementation erforderlich. Falls keine der beiden Prozessimplementationen zur Verfügung steht ist es möglich den Quelltext auf eine andere Implementation umzuändern. Weiteres dazu kann unter der Modulbeschreibung von `sysdepend.c` nachgelesen werden.

Kapitel 3

Modulbeschreibung

3.1 glInterpret.c

In dieser C-Datei befindet sich die *main()-Funktion* von `glInterpret`. Die *main()-Funktion* ist hauptsächlich dafür zuständig das *OpenGL-GLUT* Fenster zu öffnen und zu initialisieren. In der zugehörigen *Header-Datei* `glInterpret.h` ist die primäre Datenstruktur enthalten. Diese Datenstruktur ist in der Lage sämtliche unterstützten Objekte mit allen Eigenschaften zu speichern. Zusätzlich kann die Datenstruktur Informationen über Licht- und Szenen Verhältnisse aufnehmen. An dieser Stelle ist durch die Konstante `MAX_LIGHT` die maximale Anzahl an verfügbaren Lichterquellen festgelegt. Der *OpenGL*-Standard garantiert nur eine Anzahl von **mindenstens** acht Lichtquellen. Werden mehr Lichtquellen benötigt, so kann man das **lokal** durch eine Erhöhung dieses Wertes erreichen. Es ist zu beachten, dass diese Lösung unter Umständen nicht kompatibel zu anderen Programmversionen ist.

In der Quelldatei selbst befinden sich einige für *OpenGL* typische Funktionen. Die Funktion *display()* stellt die Objekt im Fenster dar, indem sie die Datenstruktur durchläuft. Die Datenstruktur ist in Form einer einfach verketteten Liste realisiert. Die übrigen Funktionen von *glInterpret.c* sind Initalisierungs und *Callback*- Funktionen und im Quelltext hinreichend dokumentiert.

3.2 glParser.c

Die C-Datei `glParser.c` wird sowohl von `glInterpret`, als auch von `glDesigner` genutzt. Die Funktionen *load_renderlist()* und *save_renderlist()* sind dabei einzig für das Laden und Speichern der Datenstruktur auf der Platte zuständig. Die übrigen Funktionen sind entweder dafür zuständig die Datenstruktur mit *OpenGL*-Standardwerten für Material und Licht zu füllen oder stellen Hilfsfunktionen zur Konvertierung von *Strings* zu internen Konstanten zur Verfügung.

In der zugehörigen *Header-Datei* `glParser.h` sind alle bekannten *OpenGL*-Objekte definiert. Die darin definierten *Tags* zum analysieren der Szenen Daten sollten nicht leichtfertig verändert werden. Eine Veränderung an diesem Punkt führt unweigerlich zu einer Unkompatibilität mit allen bisher erzeugten Szenendaten.

3.3 glList.c

Die C-Datei `glList.c` wird sowohl von `glInterpret`, als auch von `glDesigner` genutzt. Bedingt durch den Einsatz der Listenstruktur ergibt sich die Notwendigkeit für einige Funktionen die das Arbeiten mit der Liste erheblich vereinfachen. Die Datei `glList.c` hat eine Funktion *initList()* um alle Listen einmal beim Start des Programms zu initalisieren. Falls man eine neue Datei anlegt und alle Listen vorher leeren möchte steht die Funktion *clearList()* zur Verfügung. Die restlichen Funktionen sind für das Löschen, Suchen und Entfernen von Elementen der Objekt- und Lichtliste zuständig.

3.4 main.c

In dieser C-Datei befindet sich die *main()-Funktion* von `glDesigner`. An dieser Stelle wird das gtk2 Toolkit initialisiert und die benötigten Fenster gezeichnet.

3.5 interface.c

Die C-Datei `interface.c` ist von dem Oberflächen-Generator *glade* erzeugt worden. Die Funktionen darin sind für das Zeichnen der einzelnen Fenster und Benutzerelemente zuständig. Diese C-Datei ist nicht mit Kommentaren versehen und auch nicht an die vorgegebenen Richtlinen zur C-Programmierung angepasst. Die Fähigkeit von *glade*, Code automatisch zu erzeugen, macht es unmöglich die C-Richtlinen einzuhalten.

3.6 support.c

Ebenso wie `interface.c` wird die Datei `support.c` automatisch von *glade* erzeugt. Es sind hauptsächlich interne Funktionen zum Finden von Referenzen auf gtk2-Elemente darin enthalten. Es gelten, im Bezug auf die C-Richtlinen, die gleichen Einschränkungen wie bei `interface.c`

3.7 callbacks.c

In der C-Datei `callbacks.c` sind die *Callback*-Aufrufe der Benutzerelemente der gtk2-Oberfläche enthalten. Hier werden die *Callbacks* nur abgefangen und zur weiteren Bearbeitung an `glInterface.c` weitergeleitet. Das Programm wird so flexibler. Zu einem späteren Zeitpunkt lassen sich Aufrufe wie das Hinzufügen eines Objektes nicht nur aus der *Toolbox* durchführen, sondern wären auch aus einem Menüpunkt denkbar.

3.8 glInterface.c

Die in der C-Datei angefangenen *Callbacks* werden an dieser Stelle ausgeführt. Es gibt eine Vielzahl an Funktionen, welche hauptsächlich für das "Füllen" und das darauffolgende "Auslesen" der Eingabemasken zuständig sind. Darüber hinaus gibt es noch einige Funktionen die für die Verwaltung der gtk2-Listenelemente und einige weitere Hilfsfunktionen.

3.9 sysdepend.c

Alle Funktionen die nicht Systemunabhängig sind stehen in der C-Datei `sysdepend.c`. Im Moment ist das nur ein Funktion mit dem Namen *start_glInterpret()*. Die Funktion ist für das Starten von glInterpret in einem zweiten Prozess verantwortlich. Solange das verwendete System mit POSIX- oder Windows-Prozessen arbeiten kann, müssen hier keine Veränderungen vorgenommen werden. Im Bedarfsfall kann hier aber eine Systemabhängige Lösung eingefügt werden. In diesem Fall sind entweder die aktuellen *Makefiles* zu ändern, oder Modifikationen an der *automake/autoconf* Prozedur durchzuführen.

Kapitel 4

Beschreibung von glInterpret

4.1 Erklärung der Datenstrukturen

Die Objekt-Datenstruktur von *glInterpret* besteht aus eine einfach verketteten Liste. Diese Liste ist eine Struktur, welche weitere Unterstrukturen enthält. Objekt-Atribute die nur jedes Objekt anders vergeben werden, sind in einer *union* gespeichert. Dadurch wird nur der Speicherplatz verbraucht, der auch wirklich für das Objekt genutzt wird. Die Datenstruktur in der sich die Lichtinformationen befinden ist ebenfalls eine einfach verketette Liste. Beide Listen haben sowohl einen Kopf als auch eine Ende-Element um die Verwaltung und das Bewegen in der Liste zu vereinfachen. Einen genaueren Einblick in die Datenstruktur bekommt man durch aus der *Headerdatei glInterpret.h*.

4.2 Hinzufügen von neuen Objekten und Eigenschaften

Im momentanen Entwicklungsstadium sind lediglich alle *GLUT*-Objekte verfügbar. Andere *GLU*-Objekte wie Beispielsweise *gluDisk* oder *gluCylinder* sind momentan noch nicht ins Programm integriert. Eine Integration dieser *GLU*-Objekte ist eine rein zeitabhängige Frage und erst für spätere Versionen geplant.

Für eventuelle zusätzliche Entwickler sind hier, in Form eine Skizze, die Schritte zur Erweiterung der Objekte dargestellt:

- Als wichtigster Schritt müssen die Objekte erst mal eingelesen werden. Deshalb ist am Anfang der Systaxanalysator *glParser.c* zu modifizieren. In der zugehörigen *Headerdatei glParser.h* befindet sich eine *enum*-Liste der bekannten Objekte. Hier sind die neuen Objekte einzutragen. Anschließend müssen in der C-Datei noch die entsprechenden

switch- und *if*-Anweisungen zum analysieren der Szenen-Daten erweitert werden.

- Die Datenstruktur muss natürlich Erweitert werden um die neuen Objekte aufzunehmen. Dazu reicht es in die Objekt-*union* eine neue Struktur mit Namen des Objektes einzufügen. Diese Struktur muss dann in der Lage sein, alle für das Objekt relevanten Optionen aufzunehmen.

- An diesem Punkt würden die Objekte schon richtig eingelesen und sich im Speicher befinden, jedoch nicht dargestellt werden. Damit dies geschieht sind Änderungen an *glInterpret.c* notwendig. In der *display()*-Funktion müssen die zuvor in der Liste eingelesenen Werte natürlich auch dargestellt werden. Die Änderungen sind durch den Quelltext relativ einfach ersichtlich.

- Wenn alles richtig funktioniert ist es natürlich unabdingbar alle neuen Objekte und deren Eigenschaften auch für spätere Verwendung zu dokumentieren.

Kapitel 5

Beschreibung von glDesigner

5.1 Hinzufügen von neuen Objekten und Eigenschaften

Im Gegensatz zur relativ einfachen Objekt-Erweiterung des Interpreters, wird sich die Erweiterung von *glDesigner* als wesentlich Zeitintensiver zeigen. Durch die gemeinsame Verwendung des Syntaxanalysators ergibt sich hier ein großer Vorteil. Jedoch ist an recht vielen Stellen die Programmlogik zu ändern. Auch hier die Änderungen in Form einer Skizze:

- Im ersten Schritt ist er erforderlich mit Hilfe des Oberflächen-Generators *glade* die benötigten Einstellungs-Dialog für neue Objekte zu entwerfen. Die Dialoge sollten in keinem Fall direkt durch Änderungen im C-Code erzeugt werden, weil dann die Möglichkeit der späteren Verwendung in *glade* nicht mehr existiert.

- Die dadurch generierten *Callback*-Funktionen in *callbacks.c* sind danach zu bearbeiten. Soweit möglich sollten keine Aktionen in den *Callback*-Funktionen ausgeführt werden. Lediglich ein Aufruf der entsprechenden *fill_** oder *apply_**-Funktionen mit weiterreichen der Pointer zu Oberflächenelementen sollte hier passieren. Die eigentliche Funktion ist dann in *glInterface.c*. Das bringt die Flexibilität mit sich, diese Funktionen später z.B. auch von einem optionalen Menüpunkt aus aufzurufen.

Kapitel 6

Detailierter Fehlerreport

Zusätzlich zu den in der Benutzer-Dokumentation aufgezeigten Problemen ist hier noch eine Liste aller momentanen Fehler und Probleme dargestellt. Auch kleinere Unschönheiten sind aufgeführt.

- Aufgrund der Tatsache, dass Programmintern nur mit einer einfach verketteten Liste gearbeitet wird, kann es bei extrem großen Objektlisten zu Nachteilen kommen. Im Moment muss beim einfügen eines Objektes am Ende der Liste zuvor die gesamte Liste durchlaufen werden. Eine doppelt verkettete Liste würde das einfügen von Objekten am Ende wesentlich erleichtern.

- Aufgrund der komplexen Materie der UTF-8 *Strings* in *glDesigner* bzw. in *gtk2* sollten in dieser Programm-Version **keine** deutschen Umlaute oder Sonderzeichen im Bezeichner oder der Beschreibung verwendet werden. *glInterpret* sollte prinzipiell damit zurechtkommen. Allerdings könnten die so erzeugten Szenen-Dateien zu der aktuellen Programm-Version von *glDesigner* inkompatibel sein.

- Das Format der Szenen-Datei ist strikt einzuhalten. Der Syntaxanalysator versucht zwar auch von der Beschreibung leicht abweichende Formatierungen zu erkennen, ist aber noch nicht perfekt was Einrückungen betrifft.

- Die unter *Microsoft Windows* erzeugten Szenen-Dateien machen Aufgrund des Zeilen-Endezeichens Probleme beim Einlesen unter *UNIX*. Duch einen Editor oder verschiedene verfügbare Programme wie *dos2unix* kann man diese Daten aber "reparieren". Ein Problem das in der nächsten Programm-Version hoffentlich beseitigt ist.

- Momentan ergeben sich aufgrund der Verwendung von relativ unflexiblem *Makefiles* Probleme beim kompilieren auf anderen als den vorgesehenen Rechnern. Durch die Verwendung von *automake* und *auto-*

conf in der nächsten Programm-Version wird die Kompilierung erheblich flexibler.

- Kleinere Unschönheiten wie das Schliessen von Dialog-Fenstern mit `<ESC>` oder die momentan noch nicht benannten Titel bei Eigenschafts-Dialog werden auch noch bereinigt.

Anhang A

Quellcodes