# Abschlussbericht über das Praktikum

von Marcus Timm

Rostock, 14.07.2006

# Inhaltsverzeichnis

# 1. Einleitung

## 1.1 Gegenstand des Prakikums

Das Praktikum wurde durchgeführt am Institut für graphische Datenverarbeitung in Rostock (kurz „IGD Rostock“), einer Zweigstelle des IGD Darmstadt und zur Fraunhofer Gesellschaft gehörend, sowie Teil des INI-GraphicsNet, einem Verbund von Instituten und Hochschulen, die sich mit der praktischen Anwendung von Computergraphik beschäftigen.

Genau wie das Institut in Darmstadt verfügt auch das IGD Rostock über eine Reihe von Hardware und Software für Virtual Reality (VR) und Augmented Reality (AR). Das Praktikum wurde im sogenannten „EnterAction Lab“ des IGD Rostock absolviert, in der Abteilung für „Human centered interaction“. In diesem speziellen Fall ging es um die Arbeit mit Hardware im Bereich VR.

Ein Kernproblem, welches im Praktikum eine zentrale Rolle spielte, war folgendes: Die Hardware, auf der VR-Programme laufen, veraltet mit der Zeit und ganze Linien von Spezialhardware starben in der Vergangenheit aus (wenn man z.B. die Grafik-Workstations von SGI betrachtet). Um die vorhandene Software weiter benutzen zu können, muss diese auf zeitgemäße Hardware übertragen werden, es muss eine sogenannte „Portierung“ durchgeführt werden. In diesem Fall musste eine Reihe von Programmen sowohl auf ein neues Betriebssystem, als auch auf neue Hardware übertragen werden, die sich vom bisherigen stark unterschieden.

Ein weiteres Problem, das im Praktikum relevant war, ist die Abhängigkeit von Spezialisten, die im Rahmen von Projekten ein gewisses Know-How erwerben und dann zu einem späteren Zeitpunkt nicht mehr zur Verfügung stehen. Es ist daher notwendig, eine aussagekräftige Dokumentation anzufertigen, damit erworbenes Wissen auch nach Beendigung eines Projektes und nach Verlust von Mitarbeitern nicht abhanden kommt. Im vorliegenden Fall war die Software so schlecht dokumentiert, dass man sich nur mit großer Mühe einarbeiten konnte.

## 1.2 Aufgabe des Praktikanten

Hauptziel war die Durchführung einer Portierung für möglichst viele Demo-Programme von einem veralteten VR-System auf ein moderneres VR-System. Bisher liefen diese Programme auf einer alten Workstation von SGI, mit dem Betriebssystem IRIX. Zielplattform sollte ein PC mit dem Windows-Betriebssystem sein.

Daneben war es auch wichtig, die gesammelten Erfahrungen so zu dokumentieren, dass nachfolgende Programmierer die Software erweitern und warten können.

## 1.3 Weitere Informationen

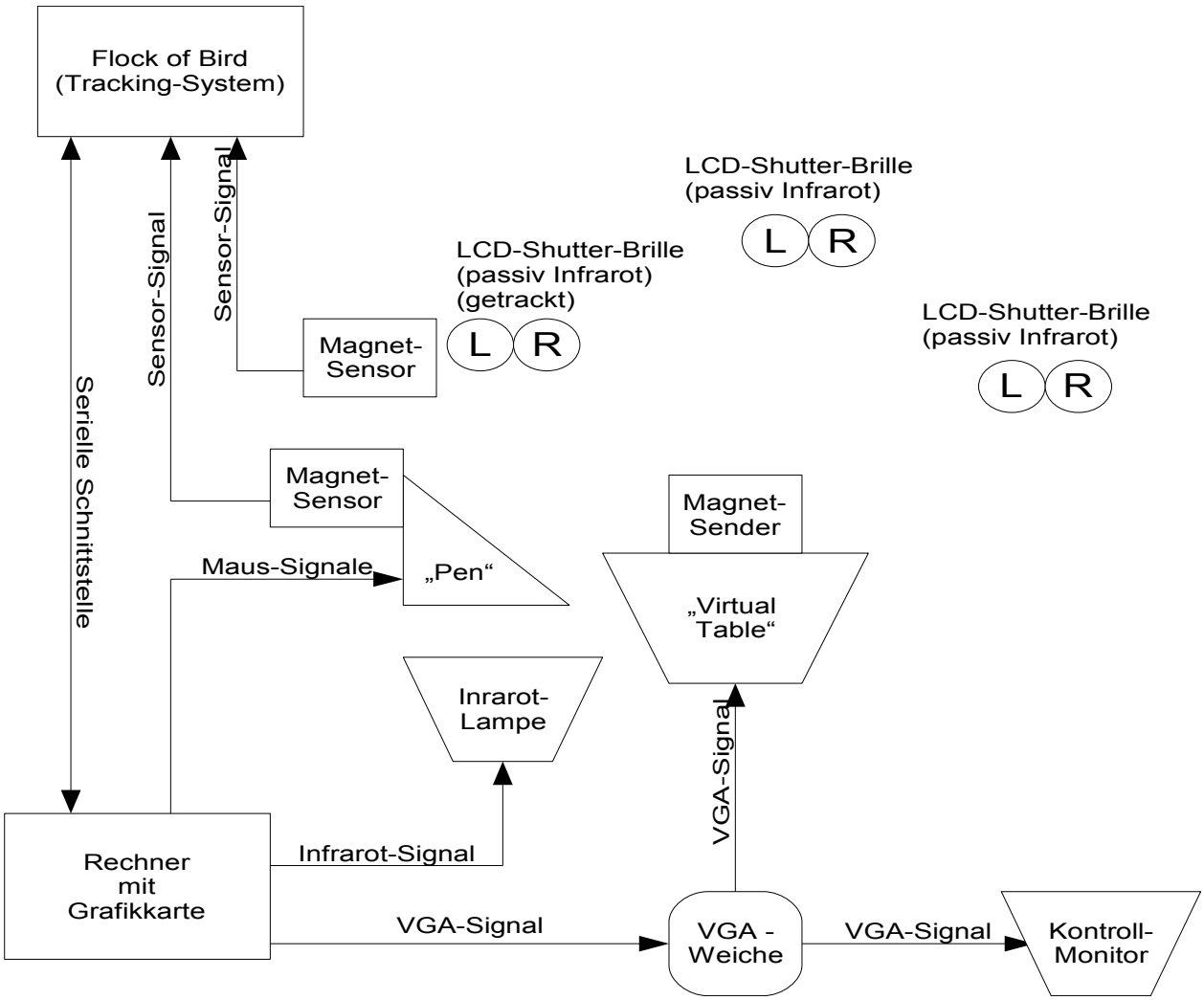
Betreuer während des Praktikums war Steffen Mader. Abteilungsleiter der Abteilung AR2 (Human centered Interaction) war zum Zeitpunkt des Praktikums Jörg Voskamp. Es gab zum Zeitpunkt des Praktikums kein Projekt mit der Hardware, die der Praktikant programmieren sollte. Die Arbeit wurde erschwert durch das Fehlen eines Teams, jedoch erleichtert durch das Vorhandensein einer Software, die nur noch für ein neues System angepasst werden musste.

# 2. Verwendete Hardware

## 2.1 Aufzählung der verwendeten Hardware

- SGI-Workstation „Indigo²“, Betriebssystem IRIX, Video-Ausgang, Infrarot-Signalgeber
- PC-Workstation, Betriebssystem Windows XP, FireGL-Grafikkarte, Infrarot-Steckkarte
- VGA-Weiche und mehrere Röhren-Monitore für PC und SGI-Workstation
- magnetisches Trackingsystem „Flock of Birds“ (Firma Ascension), mit 4 Sensoren
- 2 Stifte („Pen“) und 1 transparentes Zeichentablett („Pad“), mit Sensoren gekoppelt
- 2 Infrarot-Lampen für Infrarot-Signale, passende Zuleitung für beide Rechner
- Infrarot-Gesteuerte Stereo-Brillen (LCD-Shutter-Prinzip)
- Stationärer Spezialmonitor („Virtual Table“ von BARCO) mit Fernbedienung

## 2.2 Aufbau im Labor (Skizze)



## 2.3 Probleme mit der Hardware

Wie man der Bedienungsanleitung des magnetischen Trackingsystems entnehmen kann, ist es recht empfindlich für den Einfluss elektromagnetischer Felder in der Umgebung, sowie für größere Ansammlung von metallischem Material. Gerade am oberen Ende des „Virtual Table“, wo sich der Kathodenstrahler dieses übergroßen Monitores befindet, ist ein solcher „hot spot“, an dem sehr ungünstige Bedingungen herrschen, zudem ist hier auch die maximale Sende-Reichweite für die Sensoren erreicht. Daher ist in der Nähe der Oberkante des „Virtual Table“ mit vermehrtem Zittern („jitters“) zu rechnen, was zu sehr ungenauen Sensordaten führt.

Ein weiteres Problem bildet die Infrarot-Verbindungslinie zu den Stereo-Brillen. Sobald die optische Strecke zwischen der Brille und der Infrarot-Lampe unterbrochen wird, erlischt der Stereo-Effekt und man sieht beide Bilder als ein überlagertes Mono-Bild.

Eine genaue Kalibrierung der 3D-Koordinaten für die Sensoren und den „Virtual Table“ konnte noch nicht erfolgreich durchgeführt werden. Daher stimmen momentan die Koordinaten der realen Welt nicht exakt mit den Koordinaten der virtuellen Welt überein. Man kann dies nur durch sorgfältige Kalibrierung und softwareseitige Nachbearbeitung beheben.

## 2.4 Weiterführende Dokumentation

Zum Trackingsystem „Flock of Birds“ von Ascension gibt es im Institut und im Internet eine Bedienungsanleitung, welche man für weitere Informationen zum System studieren kann. Die erwähnte Anleitung ist sehr sinnvoll, wenn man die Erfassung der Daten (siehe auch Kapitel 4) genauer verstehen will.

Eine Dokumentation zur Spezial-Grafikkarte „FireGL III“, sowie aktuelle Treiber sind leider nicht erhältlich. Man kann sich lediglich einen älteren Treiber von den Seiten der Firma ATI herunterladen.

# 3. Das VR-Framework (Software)

## 3.1 Verwendungszweck

Das VR-Framework für den Virtual Table kann benutzt werden, um mittels einer magnetischen Tracking-Einheit, einer Shutterbrille, einer Quad-Buffer-Grafikkarte und eines Virtual Table ein interaktives 3-Stereogram zu erzeugen. Es enthält außerdem ein Modul für Erkennung von Gesten mit beliebig vielen Freiheitsgraden, z.B. 2D, 3D, 6D (Ort in 3D + 3 Euler-Winkel).

## 3.2 Komponenten der Software

Es gibt 4 Basis-Module (Bibliotheken):

- **gtl** – die mathematische Basis (Vektoren, Matrizen, Quaternionen, Methoden für Rechnung, ...)
- **imgr** – das „**i**nterface for **m**otion-based **g**esture **r**ecognition“ (Gesten-Erkennungs-Modul)
- **QtGL** – eine auf Qt basierte C++-Bibliothek für die graphische Ausgabe mittels OpenGL
- **QTCS** – eine auf Qt basierte C++-Bibliothek für die Ansteuerung von Tracking-Geräten

## 3.3 Allgemeine Anmerkungen

Die Module **gtl** und **imgr** liegen als Sammlung von C++ Templates vor. Um sie zu benutzen zu können, muss man die Kompiler-Option „ALL_INLINE“ gesetzt sein. Außerdem sollte man nur die Header-Dateien einbinden (mit #include). Außerdem entsprechen die beiden Module momentan nicht dem ANSI C++ Standard, können daher nicht mit Visual Studio 2003 und höher verwendet werden ohne Anpassungen an den Standard vorzunehmen.

Die Module **QtGL** und **QTCS** basieren auf dem Qt Framework von Trolltech. Man muss daher Qt benutzen. Eine Kurzbeschreibung zur Programmierung mit Qt findet sich im nächsten Abschnitt dieser Dokumentation.

## 3.4 Verwendung von Qt (Kurz-Übersicht)

Es hat sich folgende Methode bewährt:

Es muss Qt vorhanden sein (Umgebungsvariable QTDIR muss richtig gesetzt werden). Eine Installation von Qt ist im Netz des Hauses verfügbar.

Zuerst eine .pro-Datei erstellen (siehe test.pro in den Beispiel-Demos).

Dann qmake starten (siehe test.bat in den Beispiel-Demos).

Die entstandene .dsp Datei mit Visual Studio 6.0 öffnen.

Es wird automatisch ein Arbeitsbereich (.dsw Datei) erzeugt.

Für weitere Informationen zur Programmierung mit Qt ist folgendes Buch zu empfehlen:

„Das Qt Buch“ (von Helmut Herold)

## 3.5 Benutzung des VR-Framework

Es existieren zwei Demos, die unter Windows (Win32 API) auf einem Standard-PC lauffähig sind.

Diese enthalten Projekt-Dateien für Visual Studio 6.0, welche man als Ausgangspunkt für eine eigene Anwendung verwenden kann. Diese beiden Demos befinden sich auf der Abgabe-CD.

Konzeptionell muss man 3 Software-Bausteine erstellen, wenn man eine Anwendung für das Framework erstellen möchte:

1.

Eine **Content-Klasse**, die von QtGLContent abgeleitet ist und die Definition der Szene enthält. Genaueres dazu kann man der Dokumentation zu QtGL und den Beispiel-Demos entnehmen. In der Demo „3dsketching“ sollte man die Datei „constraint_content.h“ / „constraint_content.cpp“ ansehen.

In der Demo „basketball“ ist die Datei „test_content.h“ / „test_content.cpp“ wichtig.

Diese Content-Klasse wird mit einem QTCSWidget verbunden, indem die Methode QTCSWidget::setContent( ) aufgerufen wird.

2.

Eine **Interaction-Klasse**, die dazu benutzt wird, ein Objekt der Klasse QTCSCursor3D mit einem Objekt der Content-Klasse (siehe 1.) zu verbinden, um Interaktion zu erreichen (z.B. Eingabe mittels Stift). Das Cursor-Objekt liefert dabei die Positions-Daten für das Eingabegerät (z.B. Stift) und das Content-Objekt manipuliert die Szene entsprechend. Das Interaction-Objekt reagiert auch auf die Gesten, ist also z.B. dafür zuständig ein Objekt zu drehen, nachdem der Stift einen Halbkreis gemalt hat. Beispiele für die Implementierung einer Interaction-Klasse finden sich in den beiden Demos, jeweils in den Dateien „interaction.h“ und „interaction.cpp“.

3.

Eine **main.cpp**, wo alle notwendigen Header eingebunden werden und neben dem Objekt der Conten- und der Interaction- Klasse auch noch die Tracker initialisiert und mit einem Objekt der Klasse QTCSWidget verbunden werden, damit das Sichtgerät (z.B. Shutterbrille + Table) mit den Positionsdaten des Trackers verbunden werden kann.

Die main.cpp enthält außerdem noch ein Objekt der Klasse „ConfigEnvironment“ (siehe nächster Abschnitt) und ein Objekt der Klasse QApplication (für Qt, siehe letzter Abschnitt).

## 3.6 Der Config-Loader

Ebenfalls Bestandteil des VR-Framework ist ein neuer Config-Loader, der bestimmte Variablen aus einer Konfigurationsdatei einliest. Die Deklaration und Implementierung des Config-Loader findet man im Ordner „configloader“ in den Dateien „configenvironment.cpp“ und „configenvironment.h“.

Die Schnittstelle zum Framework wird über ein globales Objekt hergestellt.

Dieses Objekt sollte bequemerweise in der main.cpp angelegt werden. Dazu reicht es aus, folgende Zeile zu übernehmen und in die main.cpp zu setzen:

```
//global configuration object:
ConfigEnvironment configuration;
```

Man kann jedoch auch an Stelle der Datei „config.txt“ (Standard) eine andere Datei benutzen, indem man folgende Zeile benutzt:

```
//global configuration object:
```

ConfigEnvironment configuration(„Dateiname“);

Der Syntax der Config-Datei ist relativ simpel:

Zeilen mit # (Raute) am Anfang gelten als Kommentare.

Leere Zeilen (nur Zeilenumbruch) werden nicht gewertet.

Zeilen, die mindestens zwei Zeichen enthalten, werden als Variablen gewertet: Alles bis zum ersten Leerzeichen ist der Name der Variable, alles nach dem ersten Leerzeichen ist der Inhalt der Variable.

Im Programm kann man die Methoden der Klasse ConfigEnvironment (siehe „configenvironment.h“) benutzen, um Werte abzufragen, zu manipulieren oder eine Config-Datei zu speichern.

Achtung:

Boolean-Variablen können folgende Werte enthalten:

Als TRUE werden bewertet:

Yes, yes, True, true, ON, 1 und auch Y..., y..., T..., t..., 1...

Als FALSE werden bewertet:

No, no, False, false, OFF, 0 und auch N..., n..., F..., f..., 0...

## 3.7 Weiterführende Dokumentation

Es existiert eine Dokumentation zum Modul „IMGR“, welches (wie alle Teile des Framework) von Oliver Bimber erstellt wurde. Es erleichtert das Verständnis der Gesten-Erkennungs-Software erheblich. Die Dokumentation liegt im Fraunhofer Institut vor.

Zum Framework Qt gibt es auf den Seiten der Firma Trolltech eine umfangreiche Online-Dokumentation, welche die komplette API der Version 2, 3 und 4 beschreibt. Zum Zeitpunkt, als das Praktikum beendet wurde, war die Version 4.0 gerade aktuell. Das VR-Framework nutzt Qt 3.1.

# 4. Datenfluss im System

## 4.1 Grobe Übersicht

Wenn man das EVA-Prinzip aus der Informatik anwendet, so ergibt sich für das vorliegende System folgende Klassifikation:

**Eingabe:**

Koordinaten und Orientierung von mehreren VR-Eingabegeräten (z.B. Stift oder Tablett) im 3D-Raum werden verfolgt.

**Verarbeitung:**

Erkennung von bereits trainierten Bewegungsmustern, nutzt eine Bibliothek zur Gestenerkennung in Echtzeit. Anwendungsabhängige Reaktion auf Eingabedaten und die erkannten Gesten.

**Ausgabe:**

Visualisierung einer 3D-Szene mit VR-Hardware, dabei korrekte Perspektive für einen Betrachter, weitere Betrachter möglich.

## 4.2 Detaillierte Beschreibung

**Eingabe:**

Auf der Eingabeseite spielen die Sensoren des magnetischen Trackingsystems eine zentrale Rolle. Wie man der Bedienungsanleitung des „Flock of Bird“ entnehmen kann, wird vom Sender ein magnetischer Impuls ausgesendet, welcher von den Sensoren registriert wird. Die Sensoren senden ihre Messdaten an die Verarbeitungsstelle im Gerät. Der Controler im Gerät rechnet diese Daten je nach eingestelltem Programm des Controlers in räumliche Koordinaten relativ zum Sender um. Die Daten liegen nun als Winkel oder Quaternionen vor, je nach Zustand des Controlers. Nun übernimmt ein proprietärer Bus (der „Flock of Birds Bus“, kurz „FBB“) die Synchronisation und die Weiterleitung der Daten innerhalb des Verbunds der Tracking-Module. Schließlich werden die Datenpakete in einem genau definierten Format über die serielle Schnittstelle (RS-232) aus dem Tracker, hin zum Rechner geschickt, welcher diese entgegennimmt.

Eine weitere Form der Eingabe sind bestimmte Tasten-Drücke (die Zeige-Geräte, „Pens“, verfügen über ein paar Knöpfe, die mit Maustasten vergleichbar sind). Diese Tastendrücke werden unterschiedlich an den Rechner übertragen: Entweder über eine Funk-Maus, deren Signalleitungen mit dem Prototypen des Zeigestiftes verbunden sind. Oder über die Serielle Schnittstelle selbst. Die erste Variante wurde im Praktikum verwendet.

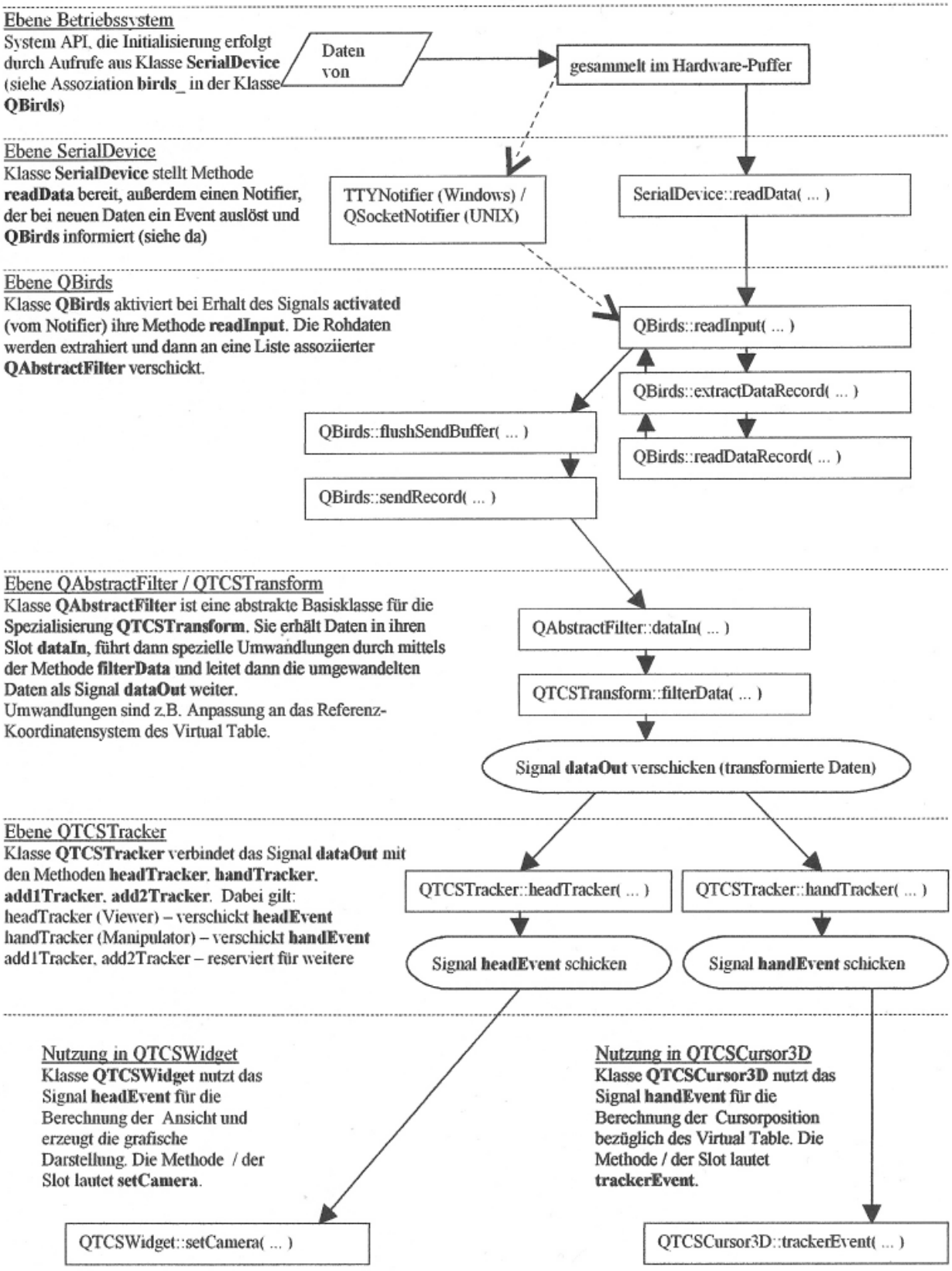
**Verarbeitung:**

Das Programm (VR-Framework, siehe Kapitel 3) nimmt die Raumdaten entgegen und benutzt diese zur Berechnung einer 3D-Szene, die nun auf dem Ausgabegerät (Monitor oder „Virtual Table“) angezeigt werden soll.

**Ausgabe:**

Es werden zwei Signale ausgegeben: Ein VGA-Stereo-Signal und ein Infrarot-Signal. Beide Signale sind miteinander gekoppelt (innerhalb des Rechners kommunizieren der Infrarot-Signalgeber und die Grafik-Steckkarte über Treiber). Das Infrarot-Signal wird verwendet, um mittels einer Lampe der Stereo-Brille des Betrachters mitzuteilen, wann ein neues Bild für eines der beiden Augen übertragen wird. Das VGA-Signal leitet das berechnete 3D-Bild von der Grafikkarte an den „Virtual Table“ weiter.

## 4.3 Flussdiagramm



# 5. Probleme und Verbesserungsmöglichkeiten

## 5.1 Behobene und offene Probleme

- ~~Config-Datei statt Umgebungsvariablen~~ (Lösung implementiert)
  Problem: wichtige Konfigurationen des Programms stehen in Umgebungsvariablen

- ~~Gestenerkennung läuft noch nicht~~ (Lösung implementiert, NEU: jetzt „G"-Taste zum Laden)
  Problem: Programm reagiert nicht auf Druck der „L"-Taste und macht keine Gestenerkennung

- ~~Kalibrierung der Stereografik-Synchronisation für die PC Workstation notwendig~~ (manuell per Taste „X")
  Problem: das Infrarot-Signal vom PC ist nicht immer mit dem Stereo-Buffer synchronisiert. Nach einer Änderung der Auflösung sind (je nach Zufall) linkes und rechtes Auge vertauscht. Bisher wird das durch einen Trick in der main.cpp ausgeglichen und nach Änderung der Auflösung einfach neu kompiliert. Lösung wäre eine manuelle bzw. automatische Kalibrierung (Idee für beides ist da).

- ~~Korrekte Datenverwaltung in QBirds / QSerialDevice~~ (alles außer extrem niedrige Frameraten)
  Problem: bei hoher Auslastung des Viewing-Moduls kommt es zu Desynchronisation oder sogar zu Datenstau / Pufferüberlauf. Ein Pufferüberlauf wird automatisch verhindert. Das Problem ist jedoch die Aktualität der gepufferten Daten – die Desynchronisation führt zu einer Zeitverschiebung, die im schlimmsten Fall nicht mehr automatisch resynchronisiert werden kann. Das Problem tritt in einem der drei folgenden Fälle auf:
  1. Hohe Komplexität der Szene, also niedrige Framerate bei hoher Auslastung
  2. Zu kleine Paketgröße* in QBirds <-> Verhältnis Overhead* zu Nutzdaten
  3. In Verbindung mit dem nächsten Problem (fehlender Reset des Flock of Birds)

  * = kleine Pakete -> höhere Datenrate und weniger Latenz (Implementierung verbesserungswürdig ...)
  * = Overhead durch Listener-Event und Auslesen des Puffers

  Ein einfacher Lösungsansatz wäre, einen speziellen Viewing-Modus zur erstellen, der in Anwendungen mit Visualisierungs-Aufgaben benutzt wird und jeweils nur das letzte Datum aus dem Datenpuffer liest und den Rest wegwirft.
  Ein schwierigerer Ansatz wäre, gemischte Visualisierung und Interaktion durch unterschiedliche Behandlung der Datenpuffer nach Eingabegerät und Sichtgerät zu erreichen.
  Wenn auch die Zeitmessung wichtig ist, muss man den schwierigen Ansatz noch mit Threads kombinieren, da das Framework größtenteils Single-Threaded ist und die Daten vom Tracker keine Zeitmarken enthalten. Für exakte Zeitmessung muss man experimentell Latenzzeiten ermitteln.

- Beenden der Verbindung zum Flock of Birds beim Beenden des Programms
  Problem: Nach erneutem Start des Programms sind die Birds noch im Stream-Modus. Das kann zu einem Datenstau führen (siehe letztes Problem) und zu weiteren Fehlern. Ein

Reset über Schalter am Bird ist zu viel Arbeit, ein Software-Reset wäre angenehmer. Wichtig ist hierbei, wodurch das Programm beendet wurde:
Wenn das Beenden durch eine Nutzereingabe (ESC-Taste) erfolgte, kann man vorher noch einen Software-Reset der Birds durchführen.
Wenn das Programm abgestürzt ist, kann eventuell Ausnahmebehandlung helfen, Ausnahmefehler während der Laufzeit abzufangen (Exceptions aus der Runtime Library).
Eine andere Lösung wäre, dem Programm zu verbieten, Trackerdaten zu akzeptieren, bis es bereit ist für die ersten Daten. Man müsste den Notifier auf stumm schalten und nach dem ersten Aufwecken den Datenpuffer löschen. Das Problem ist hierbei, die Stelle im Framework zu finden, an der man den Notifier problemlos aufwecken kann. Die bisherige Struktur des Frameworks ist generell inkompatibel zu dieser Vorgehensweise (sie geht aus von einem Flock of Birds der sich im Reset befand -> siehe Programmcode und Fehlverhalten im Testbetrieb unter Irix / Windows).

- Problem der unterschiedlichen Newline-Zeichen unter Windows und UNIX
  Problem: Konfigurationsdatei und Referenzdateien werden unter UNIX und Windows unterschiedlich gespeichert, weil sie im Textmodus abgelegt werden zwecks einfacher Veränderbarkeit. Das kann bei einem Wechsel des Betriebssystems Probleme bereiten. Als Ausweg kann man entweder die Daten im Binärmodus speichern und einen Editor für Binärdateien nutzen oder bei einem Wechsel des Betriebssystems die Dateien konvertieren.

- Nicht-standardkonformer C++ Code im VR-Framework
  Problem: Die gtl (mathematische Basis des Framework) benutzt noch alte Headerdateien, die aus der Zeit vor dem ANSI C++ Standard stammen. Wenn man dies ändert, kriegt man Haufenweise Fehlermeldungen (Namensraum std nicht vorhanden), weil irgendwelche Header diesen Namensraum nicht definieren (Internet-Recherche). Außerdem wird std::string in <iostream> statt in <string> definiert (durch Microsoft Compiler).
  !
  Es ist extrem nervig, weil man so den Namensraum „std“ nicht direkt benutzen kann, der wertvolle Deklarationen enthält – außerdem gibt es Widersprüche zwischen <iostream.h> und <iostream>. Eine versuchte Bereinigung des Quellcodes hatte keinen Erfolg, sondern stundenlangen Frust zur Folge. Mit einem anderen Compiler oder Umstrukturierung der gtl und anderer Teile könnte es klappen.

- Anpassung der Tracker-Schnittstelle an OpenTracker
  Problem: Der Einbau ist möglich, hat Vor- und Nachteile.
  Vorteile: OpenTracker bietet eine gute Konfigurationsmöglichkeit bei hoher Abstraktion.
  Nachteile: Für Windows nur verteiltes Tracking gut implementiert. Kein Support mehr.
  Ein Versuch, OpenTracker selber umzubauen wäre zu schwer (kein Multithreading für die serielle Schnittstelle unter Windows, da OpenTracker singlethreaded arbeitet). Auch die Anbindung selbst ist ein hartes Stück Arbeit, ist aber in begrenzter Zeit machbar.

- ~~Speicherleck (mehrere KB/s) in der Demo „3dsketching“~~ (fehlendes gluDeleteQuadric im Programm)
  Problem: Nach nicht erkannten Gesten (vorderer Knopf). Vermutlich bei Verwaltung von 3D-Objekten.

- Speicherleck in der Demo „3dsketching“ (2 KB pro Sekunde)
  Problem: während der Laufzeit im Taskmanager sichtbar. Vermutlich bei Verwaltung von 3D-Objekten.

- ~~Fast keine Dokumentation vorhanden~~ (QtGL dokumentiert, Allgemeine Beschreibungen gemacht)
  Problem: Es ist nur das IMGR selbst dokumentiert. Das eigentliche Framework jedoch nicht.

- ~~Kalibrierung des Virtual Table~~ (Einstellung für 1024x768@120Hz vorgenommen)
  Problem: Beim Zeichnen mit Stift werden Linien nicht direkt an der Spitze gezeichnet. Das kann daran liegen, dass der Tisch nicht ordnungsgemäß kalibriert ist. Es gibt zwei Faktoren, die relevant sind:
  - Pixelgröße muss richtig eingestellt werden (Viewfrustum muss übereinstimmen)
  - Die Ausrichtung des Bildes muss perfekt sein (Monitorbild richtig justieren / kalibrieren)

  Eigentlich müsste es dann klappen.

- Programm lässt sich nicht unter Visual Studio 2003.NET compilieren.
  Problem: Visual Studio 2003 (wahrscheinlich auch Visual Studio 2005) haben keine alten Header mehr.

- Bessere Modularisierung (dlls und libraries oder eine Makefile-Hierarchie)
  Problem: Bisher sind die einzelnen Bestandteile des VR-Framework nur durch Visual Studio 6 Projektdateien und durch übergeordnete Ordner mit Quellcode und Headerdateien modularisiert. So muss jedes Mal alles neu kompiliert werden (für jede Demo), wenn es eine kleine Änderung im VR-Framework gibt und wenn man einen sauberen Build haben will.
  Eine Lösung wären Makefile-Hierarchien (unter IRIX so gemacht)
  Eine andere Lösung wäre, die Template-Dateien so zu bearbeiten, dass man eine lib oder dll bauen kann. Momentan sind Templates aber nur inline definiert, durch Einbinden aller .cpp-Dateien und werden dann von den Modulen QtGL und QTCS beim Kompilieren gegenseitig inkludiert.

## 5.2 Der 64-Bit-Shift-Trick

Während der Arbeit ist ein ganz besonderer Fehler aufgetaucht, der seinen Ursprung in der Entwicklungsumgebung zu haben scheint.

Bei der Verwendung des Qt-Konzeptes der Signale und Slots unterläuft dem Compiler ein Fehler bei der Auswertung.
Den Fehler verursacht dabei entweder MOC (der Meta-Objects-Compiler) von Qt oder aber Visual Studio 6. Hinweise aus externen Quellen wie dem Internet konnte ich leider zu

diesem speziellen Bug nicht finden.

Der Fehler besteht darin, dass Daten falsch übergeben werden. Beim Aufruf einer Slot-Funktion erhält diese zwar einen Zeiger auf die Daten, die von der Signal-Funktion übergeben werden sollen, jedoch sind diese Daten um 64 Bit verschoben.

Durch einen speziellen Trick konnte ich dieses Verhalten korrigieren, indem ich einfach temporäre Zeiger erstellt habe, die dann „manuell“ in die entgegengesetzte Richtung verschoben wurden und somit wieder auf die korrekten Daten zeigen.
Jedoch ist dieser Trick keine gute Lösung. Eine bessere Lösung wäre es, eine neue Qt Version zu installieren (Lizenz-Probleme beachten) oder das Framework so umzuschreiben, dass man es auch mit Visual Studio 7 kompilieren kann oder einfach eine ganz andere Umgebung (MinGW oder Cygwin) zu benutzen.

Im folgenden sind alle Stellen im Quellcode benannt, an denen ich den erwähnten Trick eingesetzt habe:

Im VR-Framework:

- QTCSTracker::headTracker( )
- QTCSTracker::handTracker( )
- QTCSTracker::add1Tracker( )
- QTCSTracker::add2Tracker( )
- QTCSWidget::setCamera( )
- QTCSCursor3D::trackerEvent( )
- QTCSCursor3D::trackerEventAngles( )
- QTCSAngleTracker::headTracker( )
- QTCSAngleTracker::handTracker( )
- QTCSAngleTracker::add1Tracker( )
- QTCSAngleTracker::add2Tracker( )
- QTCSAngleTracker::headTrackerAngles( )
- QTCSAngleTracker::handTrackerAngles ( )
- QTCSAngleTracker::add1TrackerAngles ( )
- QTCSAngleTracker::add2TrackerAngles ( )

In der VR-Demo „3dsketching“:

- Interaction::UserMove( )

In der VR-Demo „basketball“:

- Interaction::UserMove( )

## 5.3 Verbesserungsmöglichkeiten

Während der Arbeit mit dem Gestenerkennungs-Modul IMGR von Oliver Bimber wurde festgestellt, dass das VR-Framework noch keine Zeit-Messung berücksichtigt und somit Geschwindigkeit von Bewegungen nicht direkt möglich sind, sondern immer von der eingestellten

Taktrate der Sensoren abhängt. Diesen Mangel müsste man beseitigen, indem man für den Raum der Gesten eine Dimension hinzufügt und eine Zeitmessung in das VR-Framework einbaut oder indem man das IMGR-Modul so umschreibt, dass es immer Zeit-Daten erwartet für die Auswertung. Dies würde auf jeden Fall auch ein neues Eintrainieren von neuen Gesten erfordern, die noch nicht in der Datenbank des IMGR enthalten sind. Somit ist eine eingehende Beschäftigung mit der von Herr Bimber hinterlassenen Software nötig.

Eine weitere wichtige Verbesserung wäre die Anbindung eines weiter verbreiteten Standard-Protokolls für VR-Hardware, wie zum Beispiel das Virtual-Reality Peripheral Network (VRPN) oder OpenTracker. Dazu müsste das momentan sehr stark auf den Flock of Birds fixierte System aufgebohrt und eine Schnittstelle einprogrammiert werden, um so eine Abstraktion für das Trackingsystem zu schaffen.