Ludwigs Maximilians Universität

Institut für Informatik

Fortgeschrittenenpraktikum Diplom Informatik

Modellierung und Entwicklung einer virtuellen Welt mit vernetzten Benutzern

Autor:
Robin Promesberger

Aufgabensteller:
Professor Dr. Wirsing
Betreuer:
Dr. Matthias Hölzl

12. Mai 2009

Inhaltsverzeichnis

1 Grundlagen

1.1 Einführung

In der heutigen immer mehr vernetzen Welt wäre ein Leben ohne digitale Kommunikation nicht mehr vorstellbar. Heute gehören Handy, Computer und Breitbandanschluss genauso zum täglichen Leben, wie vor einigen Jahren ein Telefon. Die Technik hat in den letzten Jahren unglaublich Fortschritte gemacht, so dass die Entwickler ihren Fokus nun auf komplett neue Arten der Kommunikation legen können. Eines der ersten Mittel der Kommunikation zweier Menschen war das Morsegerät, dass aber in der Anwendung sehr unpersönlich und umständlich war. Das später entwickelte Telefon bot den Menschen zum ersten Mal die Möglichkeit außerhalb ihrer Hörreichweite über die Stimme miteinander zu kommunizieren. Dann kamen vernetzte Systeme und Programme wie der *Internet Relay Chat*, der es zum ersten mal vielen Benutzern gleichzeitig ermöglichte, sich mit Textnachrichten zu unterhalten. Heutzutage ist es mit Programmen wie Skypetm möglich, über Video und Sprachübertragung mit jedem Teil der Welt zu kommunizieren. Dies wirft die Frage auf, was bei steigender Leistung und Schnelligkeit der Systeme als nächste Form der Kommunikation kommen kann.
Eines der Hauptprobleme von Kommunikationssoftware ist es, eine feine Trennung zu ziehen zwischen Privatleben und Öffentlichkeit. Während die User es vorziehen, mit Familie und Freunden über Videokonferenzen zu sprechen, wollen sie gegenüber Unbekannten ihre eigene Privatsphäre bewahren und benutzen deshalb Text-Chat Programme. Eine Lösung, um diesen Raum zwischen einem Chat in absoluter Anonymität und einem persönlichen Gespräch auszufüllen, ist erstrebenswert. Ein weiteres Problem ist, dass das Gefühl, das die meisten Benutzer von Kommunikationsprogrammen haben, wenig mit einer realen Kommunikation zu tun hat.
Um diese Probleme zu lösen, muss man die Gegebenheiten der jetzigen Software und die einer realen Kommunikation vergleichen. Daraus kann man Aspekte ableiten, die sich in zukünftiger Software verwirklichen lassen.
Um der Realität möglichst nahe zu kommen, müssen diese Aspekte nach und nach erschlossen werden.

1.2 Vorhandene Software

Die in diesem Sinne wohl bekannteste Software ist das Spiel *Second Life*.

Second Life (von Teilnehmern kurz „SL“ genannt) ist eine Online-3D-Infrastruktur für von Benutzern gestaltete virtuelle Welten, in der Menschen durch Avatare[1] intera-

[1]Ein Avatar ist eine künstliche Person oder ein grafischer Stellvertreter einer echten Person in der

gieren, spielen, Handel betreiben und anderweitig kommunizieren können. Das seit 2003 verfügbare System hat 15 Millionen registrierte Benutzerkonten, über die rund um die Uhr meist 60.000 Nutzer gleichzeitig in das System eingeloggt sind.

Second Life wurde ab 1999 von Linden Lab in San Francisco entwickelt. Das erklärte Ziel von Linden Lab ist es, eine Welt wie das „Metaversum“ zu schaffen, das in dem Roman *Snow Crash* beschrieben wird: eine vom Benutzer bestimmte Parallelwelt von allgemeinem Nutzen, in der Menschen interagieren, spielen, Handel betreiben und anderweitig kommunizieren können. Die erste Präsentation von Second Life fand im Sommer 2002 statt, im Herbst begann eine Betatestphase und seit dem 24. Juni 2003 ist Second Life online.

Die Second-Life-„Welt“ existiert in einer großen Serverfarm, die von Linden Lab betrieben und allgemein als das Grid (Gitter) bezeichnet wird. Die Welt wird von der Client-Software als kontinuierliche 3D-Animation dargestellt, die ein Raumgefühl verleiht und in die zusätzliche Audio- und Videostreams eingebunden werden können.

Die Client-Software stellt ihren Nutzern, die als Bewohner bezeichnet werden, Werkzeuge zur Verfügung, um ihren Avatar zu gestalten, Objekte zu erschaffen, durch die Second-Life-Welt zu navigieren, die Welt durch eine erweiterte Kamerasteuerung in komfortabler Weise zu betrachten und mit anderen zu kommunizieren. Die Navigation wird durch eine interne Suchmaschine und die Möglichkeit erleichtert, Landmarken zu setzen, über die man sich durch die Welt teleportieren kann.

Verschiedene Personen und Unternehmen können auf neue Weise miteinander in Kontakt treten und sich gegenseitig virtuelle Waren oder Dienstleistungen anbieten. Die Kommunikation erfolgt per öffentlichem oder privatem Chat, wobei es zahlreiche Darstellungsoptionen für den Chatverlauf gibt. Neuerdings kann auch mündlich über das optionale interne Second Talk kommuniziert werden.

Second Life fungiert auch als Plattform zur sozialen Interaktionen für verschiedenste Communities. Gleichgesinnte können Gruppen bilden und über den integrierten Instant Messenger nicht nur mit Einzelpersonen, sondern auch mit allen Mitgliedern der jeweiligen Gruppe kommunizieren. Das Programm wurde bereits für Schulungen und virtuelle universitäre Vorlesungen genutzt, sogar Livekonzerte lassen sich virtuell durchführen. Die grafische Anlehnung an Computerspiele erlaubt den Teilnehmern, Second Life auch als Onlinespiel zu nutzen und zu begreifen. Viele Teilnehmer investieren ihre Zeit und ihre Fähigkeiten, die virtuelle 3D-Welt durch neue Gegenstände (Kleidung, Accessoires, Wohnungen, Häuser, Landgestaltung etc.) permanent zu erweitern.

Gegenwärtig gilt Second Life als bedeutendste Plattform dieser Art in der westlichen Welt. Im asiatischen Raum soll es größere und leistungsstärkere Plattformen geben, wie beispielsweise das chinesische hipihi.com. Zu den Mitbewerbern von Second Life in der westlichen Hemisphäre zählen Active Worlds, von einigen als Gründerunternehmen des 3-D-Internetkonzepts 1997 betrachtet, Entropia Universe, There und Newcomer wie der Dotsoul Cyberpark.[6, SecondLife]

---

virtuellen Welt, beispielsweise in einem Computerspiel. [6, Avatar]

Abbildung 1.1: Second Life Screenshots

1.3 Ziele

Ziel dieses Fortgeschrittenenpraktikums ist es, ein System für eine Online 3D Welt zu schaffen, in der sich mehrere Benutzer einloggen und interagieren können. Das System sollte auf einer Server-Client Architektur basieren und dem User möglichst einfach zugänglich sein. Im Programm soll man sich von einem beliebigen System aus auf einem Server einloggen können. Über das grafische Interface soll man sich in einer virtuellen Welt bewegen, andere Teilnehmer treffen und mit ihnen kommunizieren können.

1.3.1 Erforderliche Use-Case Ziele

- Eine 3D Welt, die komplett aus in Echtzeit generierter Grafik besteht. Diese Welt besteht aus einem Untergrund und anderen Objekten, die sich darauf befinden, wie bewegliche Figuren, Gebäude oder Gebrauchsgegenstände.
- Der Spieler kontrolliert genau eine Spielfigur. Die Kamera ist dabei immer auf diese Figur gerichtet, d.h. der Spieler sieht die Welt aus der dritten Person.
- Mehrere Spieler können sich über Internet gleichzeitig in dieser Welt aufhalten und miteinander kommunizieren und interagieren.
- Ein Werkzeug zum userfreundlichen Erstellen neuer Welten.
- Eine angemessene Spielegrafik und -physik.

1.3.2 Optionale Use-Case Ziele

- Mehrere Welten, zu denen die Spieler jederzeit reisen können (Serverfarm).
- Ein Charakter-Bau-Werkzeug, mit dem sich die Spieler individuelle Charaktere zustammenstellen können.

- Ein Spiel-Objekt System, mit dem auf einfache Weise neue Gebrauchsgegenstände, mit denen die Charaktere interagieren können, erstellt werden können. Ausserdem ein System zur langfristigen Speicherung des Spielzustands mit all dessen Objekten in einer Datenbank.

2 Programm VIR - Virtual Interactive Reality

Der Praktische Teil dieses Praktikums ist ein Java Programm mit dem Namen *"Virtual Interactive Reality"*, kurz *VIR*.

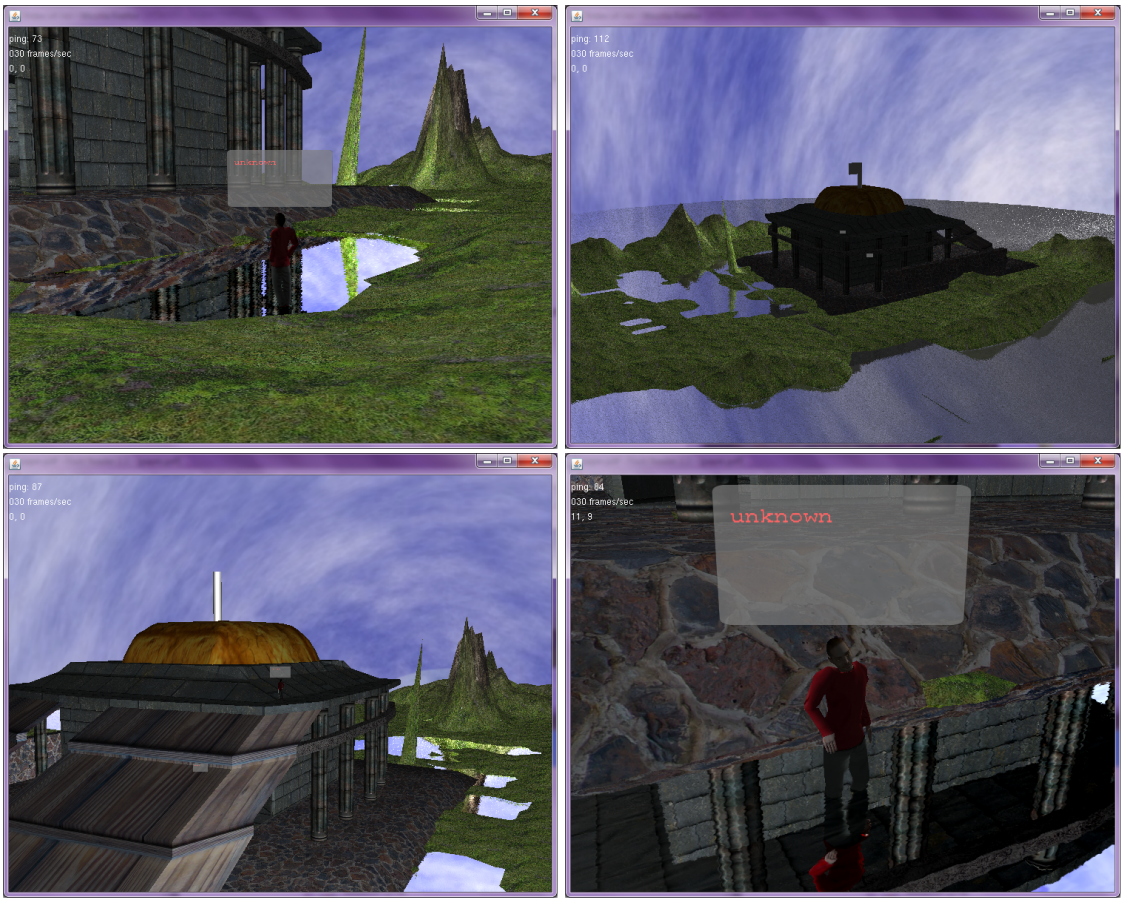

Abbildung 2.1: VIR Screenshots

2.1 Modellierung

In diesem Abschnitt wird die Modellierung des Programms VIR erläutert.

2.1.1 Grundkonzept

VIR ist in der Programmiersprache Java objektorientiert geschrieben. Die Wahl fiel auf Java, da es neben der modularen und objektorientierten Grundstruktur bereits viele Klassen zu Netzwerk, Grafik und Deployment zur Verfügung stellt. Da Java durch seine Natur als virtuelle Maschine nicht von sich aus in der Lage ist eine 3D-Grafik effizient über die Hardware zu beschleunigen, benutze ich die Bibliothek JOGL[1] um über native Bibliotheken die Schnittstelle zur Grafikkarte herzustellen. JOGL ist eine direkte Abbildung von OpenGL[2], und bietet die Möglichkeit, eine OpenGL-Leinwand in einen Java Component zu laden. Vorraussetzung für die Benutzung des Programms ist also eine 3D-fähige Grafikkarte, die die OpenGL Version 2.0 unterstützt.
Das Programm ist als Client-Server System aufgebaut, das sowohl *thick*- als auch *thin*-Clients unterstützt. Allgemein lässt sich das Programm in drei Hauptsysteme aufteilen:

1. Spielsystem
2. Userinterface
3. Netzwerksystem

Zum Starten des Spiels kann das komfortable Java Webstart System verwendet werden, das neben dem Hauptprogramm automatisch die JOGL Bibliothek herunterlädt.

2.1.2 Spielsystem

Das Spielsystem ist rein serverseitig und beinhaltet sämtliche Datenstrukturen und Kontrollmethoden für den logischen Ablauf des Spiels. Alle relevanten Daten sind in einer Datenstruktur GameState gesammelt, die auch gleichzeitig als Fassadenklasse für das Netzwerkinterface dient. Außerdem führt Gamestate in periodischen Abständen ein Update aus, bei dem alle möglichen computergesteuerten Aktionen (z.B. NPC[3]) ausgeführt werden. Als Spielumgebung gibt es eine 2-dimensionale Datenstruktur *GameMap*, die in feste Größen eingeteilte Felder enthält, welche wiederum Spielobjekte (Architektur, Spielcharaktere, Interaktionsobjekte, etc.) enthalten könnten.

2.1.3 Userinterface

Das Userinterface läuft rein clientseitig und stellt die Daten dar, die vom Server aus gesendet werden. Außerdem sendet es den Input des Users über das Netzwerkinterface an den Server. Als Interface zur Grafik dient ein JComponent, das eine OpenGL Leinwand beinhaltet. Das OpenGL Interface ruft hier immer wieder abwechselnd eine *draw* Methode auf, in der mit OpenGL Anweisungen das nächste Bild gezeichnet werden kann und eine *update* Methode, in der die zu zeichnenden Objekte manipuliert werden können

[1] Java Open Graphics Library
[2] www.opengl.org
[3] non-playable-character. ein computergesteuerter Charakter

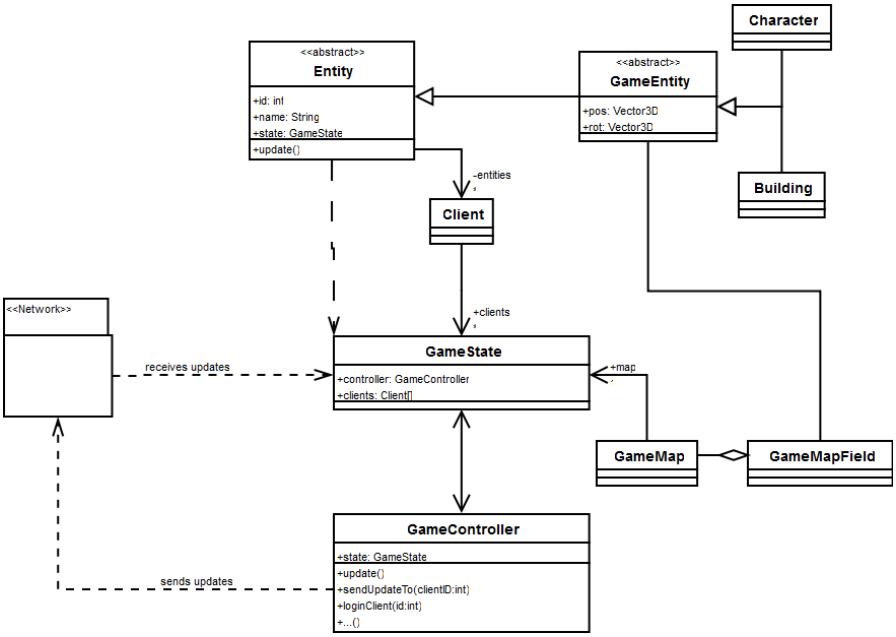

Abbildung 2.2: UML-Diagramm des Spiel Systems

und andere Berechnungen durchgeführt werden können (z.B. Kollisionserkennung und Reaktion).

2.1.4 Netzwerksystem

Das Netzwerksystem stellt eine Verbindung zu einem Server her und sorgt dafür, dass der Client Kommandos zum Server senden kann und bestimmt, wie auf die Antworten oder auf Kommandos vom Server reagiert wird. Als System dient hier eine normale Socketverbindung. Die Möglichkeit des Java-RMI [4] kam hier nicht in Frage, da es eine relativ schlechte Performance bei zeitkritischen Operationen aufweist. Deshalb habe ich ein eigenes ähnliches Interface gebaut, dass entfernt Methoden und Funktionen ausführen und beantworten kann, weiterhin aber einfach erweiterbar ist. Ebenso ist es in der Lage, ganze Objekte zu versenden, wobei hier auf die geringe Größe der zu sendenden Daten wert gelegt wurde.

2.1.5 Deployment

Java Web Start ist eine Technik von Sun Microsystems, die es ermöglicht, Java-Anwendungen über das Internet mit nur einem Klick zu starten. Im Unterschied zu Java-Applets benötigen Java-Web-Start-Anwendungen jedoch keinen Browser, um ablaufen zu können. Bei jedem Start einer Java-Web-Start-Anwendung kann überprüft werden, ob neuere Komponenten vorliegen. So kann der Anwender stets mit der aktuellen vom Autor des

[4] Remote Method Invocation

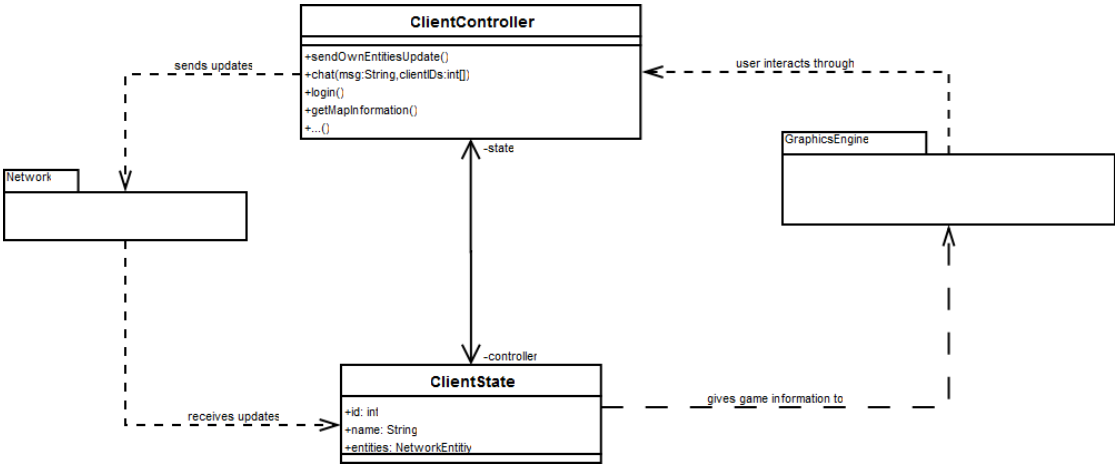

Abbildung 2.3: UML-Diagramm des Client Systems

Programms zur Verfügung gestellten Version arbeiten. Eine einmal heruntergeladene Version einer Anwendung bleibt solange in einem Cache auf der Festplatte des Clients, bis bei der Prüfung festgestellt wird, dass eine neue Version vorliegt und diese geladen werden muss. Somit werden unnötige Downloads verhindert und trotzdem sichergestellt, dass immer die aktuelle Programmversion läuft.

Voraussetzung für Java Web Start ist, dass

- der Entwickler das Programm auf einem Server zum Download anbietet und eine spezielle XML-Datei bereitstellt (mit der Endung „.jnlp“), in der die einzelnen Komponenten der Anwendung beschrieben sind,
- der Webserver des Servers, auf der die Java-Anwendung bereitgestellt wird, den MIME-Type application/x-java-jnlp-file kennt,
- der Anwender sowohl eine JRE als auch Java Web Start installiert hat, da die Java-Anwendung mit Hilfe der Java VM des Anwenders ausgeführt wird. Java Web Start wird seit dem JRE 1.4.2 automatisch installiert.

Das Java Network Launching Protocol (JNLP) ist ein XML-Format, das festlegt, wie Anwendungen per Java Web Start aufgerufen werden. JNLP-Dateien enthalten Informationen wie den Ablageort von JAR-Dateien, den Namen der Hauptklasse einer Anwendung und zusätzliche Parameter für das aufzurufende Programm. Ein korrekt konfigurierter Webbrowser übergibt JNLP-Dateien an die Java-Laufzeitumgebung, die dann ihrerseits die Anwendung auf den PC des Anwenders herunterlädt und startet. JNLP wurde im Java Community Process als JSR 56 entwickelt, und liegt mittlerweile in den drei Versionen 1.0, 1.5 und 1.6 vor.[6, Java Webstart] VIR unterstützt nur die Version 1.6. Der Grund dafür ist eine Java interne Umstellung der Funktionsweise des Caching der jar-Dateien.

3 Implementierung

Im Folgenden wird detaillierter auf wichtige Aspekte der Implementierung eingegangen.

3.1 Netzwerk

Das Netzwerksystem gliedert sich in ein Server- und ein Client-System auf, die gemeinsame Klassen gebrauchen.

Die abstrakte Klasse *Command* gibt eine Vorlage für jede Art der Kommunikation sowohl für den Server als auch für den Client. Die Klasse enthält zwei abstrakte Methoden, eine für die serverseitige Ausführung und eine für die clientseitige Antwortverarbeitung. Für Client und Server ist die jeweils andere Methode ein Stub, der nicht implementiert werden muss. Diese Kommandos garantieren die Einheitlichkeit der Befehlsverarbeitung. Kommandos können entweder Methoden oder Funktionen mit Ergebnis sein. Bei einer Funktion merkt sich der Client das Kommando und wartet, bis der Server das Kommando beantwortet hat. Es können auch ganze Objekte (allerdings nur mit deren primitiven Feldern) verschickt werden. Hierzu dient eine abstrakte Klasse *NetworkObject*, die Kodierungsmethoden von ganzen Objekten oder Veränderungen von Objekten zur Verfügung stellt. Diese lesen automatisch alle primitiven Felder jeder erbenden Klasse und fügen diese in die Felder der Klasse am anderen Ende ein. Existieren die entfernten Objekte bereits, werden nur die Änderungen gesendet und übernommen um die zu sendenden Daten möglichst klein zu halten.

Der Server registriert alle eingehenden Verbindungen und startet den Prozessor für diesen Client, wenn er sich über den offenen Socket per Login anmeldet. Sowohl Client als auch Server halten zwei Sockets über zwei Ports für ausgehende und eingehende Verbindungen. Ein Öffnen der Firewall beim Client ist deshalb nicht nötig. Die Clients werden durch eine ID eindeutig bestimmt und die Listen der Client-IDs liegen synchronisiert sowohl beim Server als auch bei den Clients. Das Routing der Nachrichten kann vom Kommando und vom Server durch Parameter beim Senden der Kommandos bestimmt werden (z.B. ist es möglich eine Textnachricht sowohl als Broadcast an alle anderen Clients zu schicken, aber auch als Single- oder Multicast).

3.2 Grafik

Detailliertere Einblicke in die Funktion von 3D-Grafik: Shader, Kollision, Kamera, Szenegraph, Datenstrukturen und Optimierungen.

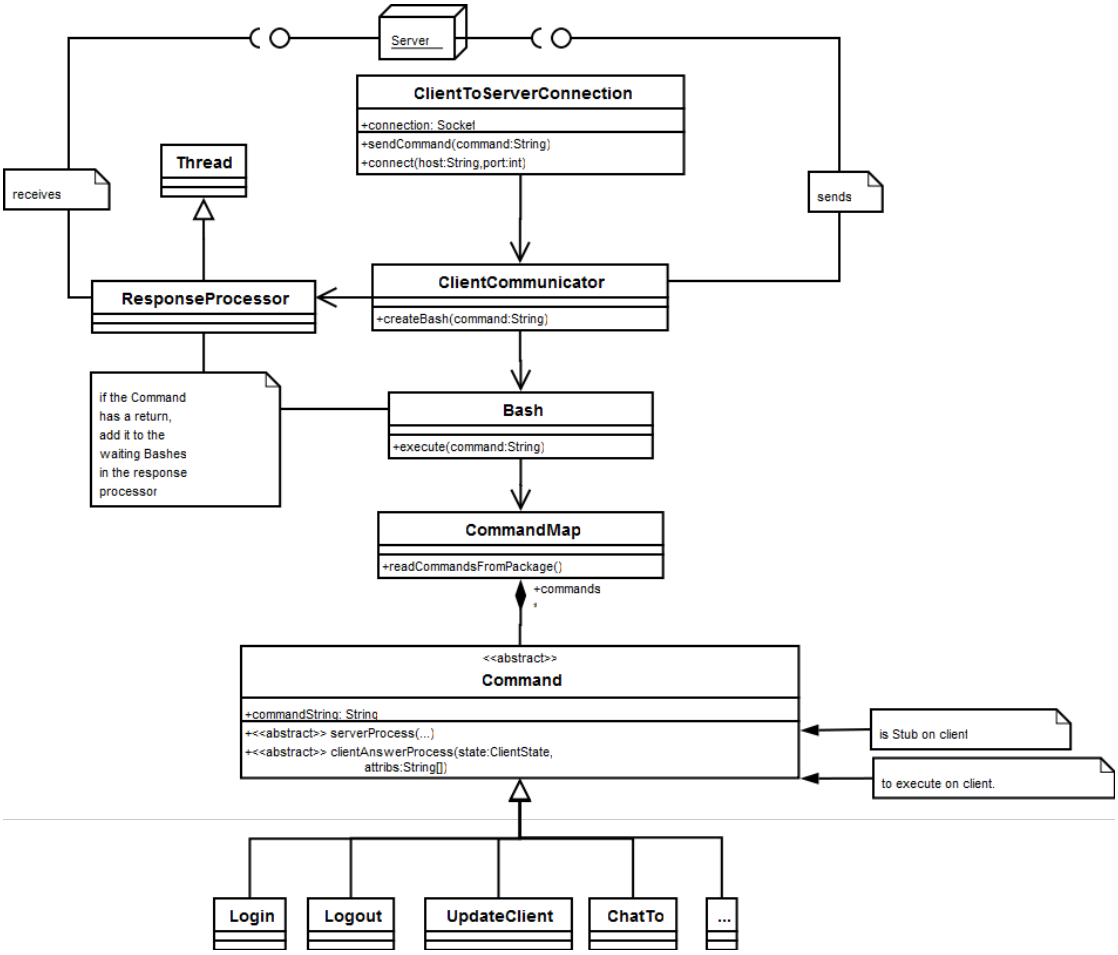

Abbildung 3.1: UML-Diagramm des Netzwerk Systems

3.2.1 OpenGL

OpenGL (Open Graphics Library) ist eine Spezifikation für eine Plattform- und Programmiersprachen-unabhängige Programmierschnittstelle zur Entwicklung von 3D-Computergrafik. Der OpenGL-Standard beschreibt etwa 250 Befehle, die die Darstellung komplexer 3D-Szenen in Echtzeit erlauben. Zudem können andere Organisationen (zumeist Hersteller von Grafikkarten) proprietäre Erweiterungen definieren.

3.2.2 Grundlagen

Nach der Einbindung von openGL wird so oft wie möglich eine Draw-Funktion aufgerufen, die den aktuellen Frame (das Bild) zeichnet. Um Flackern zu vermeiden werden 2 Buffer benutzt, damit immer ein fertiges Bild angezeigt werden kann, während das andere off-screen gezeichnet wird. Die Grundzeichenfunktionen von OpenGL beschränken sich

auf Punkte, Linien, Dreiecke, Vierecke und Polygone, wobei bei Oberflächen Dreiecke natürlich die beste Performance haben, sich dadurch allerdings die Komplexität erhöht, da bei größeren Objekten trianguliert werden muss. Auch Licht kann in OpenGL mit wenigen Parametern erzeugt werden. Dazu kommen noch eine Menge von OpenGL Befehlen zur Optimierung und Qualitätssteuerung. Um Objekte auf den Bildschirm zu bringen, muss erst ein Frustum bestimmt werden: Ein Trapez, das angibt, welcher Bereich des 3-dimensionalen Raums überhaupt und in welcher Perspektive der Inhalt gezeichnet werden muss. Für jedes gezeichnete Dreieck (Face) werden 3 Punkte (Vertex), eine Normale[1]

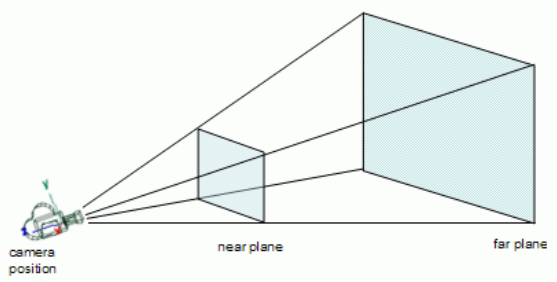

Abbildung 3.2: Frustum

und bei texturierten Objekten drei 2-dimensionale Texturkoordinaten, die die Position und Skalierung der Textur auf dem Face angeben[2], benötigt.

3.2.3 Transformationen

Die Position, Rotation und Skalierung eines Objekts wird durch zwei 4x4 Matrizen bestimmt: Die Projektionsmatrix, die dem oben erwähnten Frustum entspricht und der Modelview Matrix, die den Raum entsprechend transformiert.

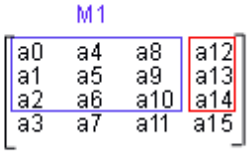

Abbildung 3.3: Transformationsmatrix

Die blau gekennzeichnete Untermatrix stellt hierbei eine Rotationsmatrix dar, der rote Bereich einen Transformationsvektor. Die Skalierung wird durch die Diagonale in der Rotationsmatrix (a0,a5,a10) gespeichert. Die untere Reihe ist immer (0,0,0,1), um

[1] die angibt, in welche Richtung das Face zeigt und deshalb nur für die Lichtberechnung erforderlich ist
[2] wobei 0.0, 0.0 der untere linke und 1.0, 1.0 der obere rechte Rand der Textur sind. Bei Werten über 1.0 wird die Textur entsprechend verdoppelt und gestaucht

die Matrix quadratisch zu halten, was vorteilhafte Eigenschaften hat. Einer der Vorteile ist neben der Invertierbarkeit die einfache Multiplizierbarkeit der Matrix, denn anstatt 3 Matrix-Multiplikationen muss nur eine einzige durchgeführt werden und das ist mit einem modifizierten Algorithmus von Strassen recht effizient. In OpenGL gibt es für diese Modelview Matrizen einen Stack, auf den sie außerhalb von Zeichenoperationen gepusht bzw. von dem sie gepoppt werden können.

3.2.4 Szenengraph

Ein Szenengraph ist eine Datenstruktur, die häufig bei der Entwicklung grafischer Anwendungen eingesetzt wird. Es handelt sich um eine objektorientierte Datenstruktur, mit der die logische, in vielen Fällen auch die räumliche Anordnung der darzustellenden zwei- oder dreidimensionalen Szene beschrieben wird.

Der Begriff Szenengraph ist nur unscharf definiert. Dies liegt daran, dass konkrete Szenengraphen in der Regel anwendungsgetrieben entwickelt werden. Die Programmierer nutzen also die Grundidee, passen sie aber für die spezifischen Erfordernisse der Anwendung an. Feste Regeln, welche Funktionen ein Szenengraph erfüllen muss, gibt es daher nicht.

Aus graphentheoretischer Sicht ist ein Szenengraph ein zusammenhängender einseitig gerichteter Graph, der keine Kreise aufweist (oftmals auch ein Baum), dessen Wurzelknoten die Gesamtszene (das „Universum“) enthält. Dieser Wurzel untergeordnet sind Kindknoten, die einzelne Objekte der Szene enthalten. Diese Knoten können wiederum Wurzel eines weiteren Baumes, also einer weiteren Hierarchie von Objekten sein.

Dieser Ansatz ermöglicht die hierarchische Modellierung der Objekte in einer Szene. Jeder Knoten des Szenengraphen hat üblicherweise eine Transformationsmatrix. Bei Manipulation dieser Matrix wird das zugehörige Objekt selbst, aber auch die Objekte aller untergeordneten Knoten transformiert. Man unterscheidet in diesem Fall zwischen Objektkoordinaten (Koordinaten eines Objektes bezüglich des übergeordneten Objektes) und Weltkoordinaten (Koordinaten eines Objektes bezüglich des Ursprungs des Universums - der Wurzel des Szenegraphen). Durch diese hierarchische Sicht wird der Aufbau und das Manipulieren einer Szene deutlich vereinfacht. Man muss nicht jedes Einzelteil eines Objektes einzeln transformieren, sondern transformiert einfach die Gesamtheit aller Einzelteile.

Als Beispiel mag die Modellierung eines Autos mit vier Rädern dienen. Ein Knoten im Szenengraph repräsentiert das Objekt Auto. Dieser Knoten hat vier Kindknoten, die jeweils ein Objekt vom Typ Rad enthalten. Wird die Position oder die Lage des Auto-Knotens verändert, so wirkt sich die Veränderung auch auf alle Kindknoten aus, in diesem Fall also die Räder. Eine manuelle Neuberechnung der Position der Räder ist deshalb nicht erforderlich. [6, Szenen Graph]

Es gibt einige bereits existierende Szenengraphen, sogar für Java (Java3D), doch schien mir die Modellierung und Implementierung eines eigenen Szenengraphen deshalb sinnvoll, um eigene Features wie Shader, Heightmaps, 3D-Model Loading, Keyframe- und Skeleton Animation hinzuzufügen.

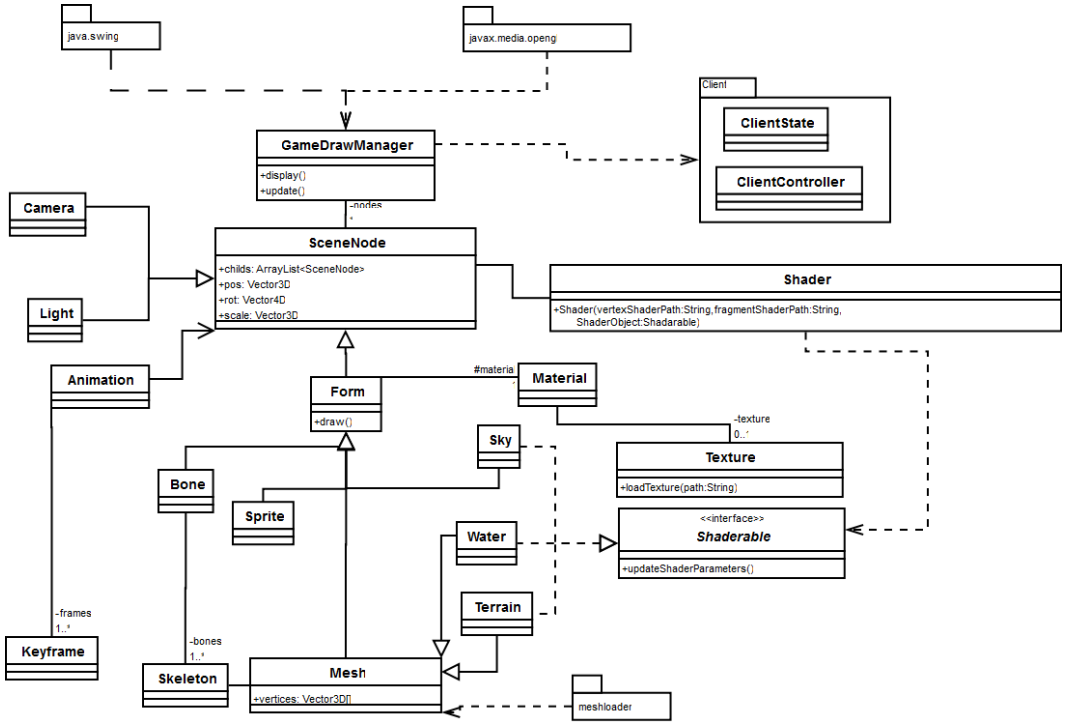

Abbildung 3.4: der Szenengraph von VIR

3.2.5 Shader

Um die mittlerweile sehr schnellen Speicher und Prozessoren der Grafikkarten voll auszunutzen, werden Shader benutzt. Shader, auch als Schattierer bezeichnet, sind Hardware- oder Softwaremodule, die bestimmte Renderingeffekte bei der 3D-Computergrafik implementieren. Aus technischer Sicht bezeichnet „Shader“ denjenigen Teil eines Renderers, der für die Ermittlung der Farbe eines Objektes zuständig ist – im Gegensatz zu dem Teil, der die Sichtbarkeit des Objektes ermittelt. Shader wurden ursprünglich für das Shading, also die Beleuchtungs-Berechnung, entwickelt, werden aber mittlerweile auch für andere Dinge verwendet.

Der Begriff „Shader“ wird sowohl für die Hardware-Shader als auch für die darauf laufenden Programme selbst verwendet. Hardware-Shader (auch: Shadereinheiten, Shader Units) sind kleine Recheneinheiten in aktuellen Grafikchips (unter Windows seit DirectX-Version 8, plattformunabhängig seit OpenGL 2.0 ansprechbar). Traditionell wird zwischen zwei Typen unterschieden, den Pixel- und den Vertex-Shadern. Shader können zur Erzeugung von 3D-Effekten programmiert werden. Während Pixel-Shader die Bildpunkte verändern und auch die Pixelfarbe berechnen können, dienen Vertex-Shader geometrischen Berechnungen und dynamischen Veränderungen von Objekten. So erzeugen z.B. beide Shader kombiniert den Wassereffekt im Computerspiel Far Cry. Sie können auch zur Berechnung von Lava, Lack, Fell usw. eingesetzt werden. Seit DirectX 10 ist als dritter Shader-Typ der Geometry-Shader hinzugekommen, der die vom

Vertex-Shader ausgegebenen Polygondaten erhält und diese noch weit flexibler bearbeiten kann, sogar weitere Geometrien zur Szene hinzufügen können (der Vertex-Shader kann nur bestehende Geometrien manipulieren).

Die Shader in VIR sind mit der von OpenGL ausführbaren Sprache HLSL (High Level Shader Language) geschrieben und kompiliert, die mit ihrer C-ähnlichen Syntax recht einfach zu erlernen ist. Das Debugging allerdings gestaltet sich relativ aufwändig, da das Programm komplett auf die Grafikkarte geladen wird und somit von herkömmlichen Debuggern nicht erreichbar ist.[6, Shader]

Terrainshader setzt mehrere Texturen (z.B. Schmutz, Gras, Stein, Schnee) zu einer Geländetextur zusammen. Je nach y-Wert (Höhe) der Punkte blendet der Fragmentshader die entsprechenden Texturen zusammen. Damit wird ein fließender Übergang der Texturen erzielt.

Himmelshader überblendet zwei Texturen und bewegt diese.

Wassershader der komplizierteste der Shader. Wenn man sich echtes Wasser ansieht, wird man als erstes die reflektierte Umwelt bemerken, die von den Wellen verzerrt worden ist. Man wird ebenso bemerken, dass auch die Umwelt unter dem Wasser verzerrt ist (wegen den Wellen und der Brechung des Lichts). Die Verzerrung kann mit einer sog. du/dv Map dargestellt werden. Der Fresnel Faktor gibt an, wie viel Licht von der Wasseroberfläche reflektiert wird. Man approximiert den Fresnel Faktor mit 1.0 minus dem Punktprodukt der Normale des Pixels und dem normalisierten Vektor zur Kameraposition. In der Natur gibt es außerdem viele Schmutzpartikel im Wasser, die das Wasser trüben. Um diesen Effekt zu simulieren, rendert man das Geländes je nach Tiefe mehr oder weniger transparent auf die Wassertextur. Als Letztes dürfen Lichtspiegelungen bei keinem Wasser fehlen [7]. Abbildung 3.5 zeigt einen Flowchart der Zusammensetzung des Effekts.

3.2.6 Heightmaps

Heightmaps (Höhenfelder) sind zweidimensionale skalare Felder, die ein Höhenrelief beschreiben. Jedem Ort ist hier also ein Wert zugeordnet, der eine Position in der dritten räumlichen Dimension angibt, eine Höhe. So können z.B. die Höhenreliefs von Landschaften und anderen Oberflächen als Höhenfeld beschrieben werden. Ein Höhenfeld kann mit einem unregelmäßigen Netz aus Punkten beziehungsweise Dreiecken in einer Höhenkarte (engl. heightmap) beschrieben werden. Häufig wird es jedoch auch mit einem darüber gelegten (regelmäßigen) Raster von Abtastwerten beschrieben [6, Heightmap]. Abbildung 3.6 zeigt eine Heightmap und das dazugehörige gerenderte Polygonnetz.

3.2.7 Kollisionserkennung

Als Kollisionserkennung oder Kollisionsabfrage (engl. Collision Detection) wird in der Algorithmischen Geometrie, Computer Aided Design, Computersimulation, -animation und -grafik sowie in der Robotik das Erkennen des Berührens oder Überlappens zweier

oder mehrerer geometrischer (starrer oder deformierbarer) Objekte im zwei- oder dreidimensionalen Raum verstanden. Einer Kollisionserkennung folgt die Kollisionsantwort oder Kollisionsbehandlung, wodurch eine Reaktion auf die Kollision eingeleitet wird.[6, Kollisionserkennung] In VIR wird die Kollisionserkennung mit hierarchischen Bounding Boxes und Dreieck-Linien Schnitt-Tests verwirklicht. Um die Kollision eines Objektes zu testen, wird es mit jedem Kollisions-Objekt in der Nähe getestet, ob sich deren umfassende Bounding Boxes schneiden. Ist dies der Fall, wird das gleiche mit allen Bounding Boxes der Unterobjekte getestet. Bei allen positiven Tests werden Schnitt-Tests von allen Dreiecken des Objekts mit bestimmten Strecken in Bewegungsrichtung getestet und die Anzahl der Schnitte gezählt. Ist diese Anzahl ungerade, befindet sich der Anfangspunkt der Strecke in einem Kollisionsobjekt und muss behandelt werden.

3.2.8 Meshloader

Ein Meshloader ist eine Struktur um vorgefertige Polygonobjekte inklusive Normalen, Texturkoordinaten und Materialien in und aus einer Datei zu laden. Als Dateiformat habe ich hier das Ogre3D Format[5] gewählt, das ermöglicht, aus 3D Programmen wie 3D Studio Max beliebige Szenen, die u.a. 3D-Objekten, Lichtern, Kamera, Knochen und Animationen beinhalten können, im übersichtlichen XML-Format zu exportieren. Diese Dateien werden von einem selbst entwickelten Parser gelesen und in einen VIR Szenen-Graph umgewandelt.

3.3 Übersicht verwendeter Libraries und Tutorials

3.3.1 Libraries

- JOGL (Java Open Graphics Library) [1]
- PDFView: PDF Renderer. [8]
- Kamera [3]

3.3.2 Tutorials

- NeHe Productions [4]
- Lighthouse 3D [2]
- Bonzai Software [7]

3.4 Appendix

3.4.1 Mapeditor

Um eine Virtuelle Welt zu erstellen, bietet VIR einen Gelände Editor, mit dem das Gelände geformt und Objekte gesetzt werden können. Diese Karten können gespeichert

und vom Hauptprogramm geladen werden. Der Editor benutzt die gleiche Grafik Engine wie VIR und ermöglicht ein Formen des Geländes in Echtzeit.

3.4.2 Skin- und Bones-System

Die Skin- und Bones Technik ist eine der fortgeschritteneren Techniken in der Entwicklung von Computergrafik und kommt seit mehr als 10 Jahren in Computerspiel-Grafik zum Einsatz.

Die Motivation Beim Versuch, komplexere Bewegungen zu animieren, kommt man mit den 3 Grundformen der Animation (Translation, Rotation und Skalierung) sehr schnell an die Grenzen des Durchführbaren. Natürlich ist es möglich Objekte zu zerlegen, und davon einzelne Teile zu animieren (wie z.B. das Bein einer menschlichen Figur), aber dadurch treten hässliche Fehler auf (wie das Abreissen des Beins vom Torso beim rotieren), die man zwar mit einigen Tricks überdecken kann, aber irgendwann nicht mehr zu verheimlichen sind. Ein anderer Ansatz ist die sog. Vertexanimation, in der die Positionsinformation jedes einzelnen Punktes in einem Mesh getspeichert, exportiert und eingelesen wird. Dies hat den Vorteil, das jede beliebige Art der Animation eins zu eins exportierbar ist. Der Nachteil ist allerdings, dass mit wachsender Anzahl von Punkten die Menge der zu Lesenden Daten viel zu groß wird. Deshalb hat man das Bone/Skin System entwickelt, um einen Mittelweg zwischen der Freiheit und Einfachheit beim Animieren und der Performance beim Abspielen zu schaffen.

Begriffe Ein Bone (Knochen) ist eine simple Datenstruktur, die aus einer Position und einer Achse besteht. Deweiteren haben Bones eine Hierarchische Struktur, in der sie ein sog. Skeleton (Skelett) bilden. Jeder Bone hat also beliebig viele Kinderknochen und bis auf den Wurzelknoten einen Elternknochen. Das Skin beschreibt nun das Verhältniss zwischen Punkten und Knochen. Dabei sind zwei Informationen von Bedeutung: Welche Punkte gehören zu welchen Knochen und wie stark wird der Punkt von der jeweiligen Bewegung der Knochen beeinflusst.

Daten In Programmen wie 3D Studio Max können Skeletons erstellt und einem Mesh zugewiesen werden. Das Programm wird automatisch als Startkonfiguration die Punkte, die einem Knochen am nächsten sind, diesem zuweisen. Ist ein Punkt mehreren Knochen sehr nahe, wird das Programm (oder der Benutzer) diesen Punkt allen diesen Knochen zuweisen, aber dessen Beeinflussung (Weight) unter ihnen verteilen. Die Summe dieser sog. Weights eines Puktes ergibt als gewichtete Summe immer 1. Als Implementierung benutze ich ebenfalls Bone Objekte, die belibig viele Kinderknoten und einen Elternknoten haben können. Zur späteren Berechnung hat jeder Bone eine Position, eine Rotation als 4-dimensionalen Vektor (winkel, achse.x, achse.y, achse.z), eine 4x4 Bone-Matrix und eine ebenfalls 4x4 relative Bone-Matrix. Ein Mesh, auf dass Bones angewendet werden soll hat ein Skeleton und eine HashMap, die für jedes beeinflusste Vertex ein Array des Types

BoneWeight besitzt, welches ein Container für einen Bone und ein Weight ist, das angibt, wie sehr der Bone dieses Vertex beeinflusst.

Funktion folgender Algorithmus aktualisiert die Vertices eines Meshes:

Zur Vorbereitung werden 2 Hilfsfunktionen in der Klasse Bone zum Updaten der Bone-Matrizen benötigt:

Listing 3.1: applySkeletonToMesh

```
1
public void updateBoneMatrix() {


 pushCurrentModeviewMatrixOnOpenGLStack();
  if(this.isRootBone()){
   loadIdentityMatrix();
   //1 0 0 0
   //0 1 0 0
   //0 0 1 0
   //0 0 0 1
  }

  //translates the modelviewmatrix by bonePosition
  translate(bonePosition.x, bonePosition.y, bonePosition.z);

  //rotates the modelviewMatrix by
  //boneRotation.a around the axis bonePosition.xyz
  rotate(boneRotation.a, boneRotation.x,
      boneRotation.y, boneRotation.z);

  this.boneMatrix = loadCurrentModelViewMatrixIntoArray();

  //invert bonematrix
  this.boneMatrix = this.boneMatrix.inverse();

  //recursive call method for all childs
  for (Bone b : getChilds()) {
   pushCurrentModeviewMatrixOnOpenGLStack();
   b.updateBoneMatrix();
    popCurrentModeviewMatrixFromOpenGLStack();
  }
  popCurrentModeviewMatrixFromOpenGLStack();
 }

 public void updateRelativeBoneMatrix() {
  //does essentially the same thing as updateBoneMatrix
  //but also adds any translation or rotation from any animation
  //to the modelview matrix
  //also this matrix IS NOT intertved
 }
```

BonePosition und BoneRotation geben die ursprüngliche Position und Achse des Bones beim Laden an. Durch das Ausführen dieser beiden Funktionen am Wur-

zelknochen hat jeder Knochen nun 2 Matrizen. Eine Matrix boneMatrix, die die ursprüngliche Transformation des Knochens in Abhängigkeit zu dessen Eltern beschreibt und eine Matrix relativeBoneMatrix, die den zustand des Bones nach jeglicher Tansformation in Abhängigkeit zu ihren Eltern beschreibt, d.h. wenn sich die Matrix eines Bones ändert, ändern sich auch die Matrizen aller Kinderknochen unter diesem Bone. Als invertierte Matrix habe ich deshalb die boneMatrix gewählt, da diese sich nicht ändert und dadurch nur ein einziges mal invertiert werden muss. Da die Matrix boneMatrix invertiert ist, ergibt sich durch eine Matrix-Multiplikation genau der *Unterschied* zwischen den Beiden Matrizen und somit die Information für jedes Vertex, das vor diesem diesem Knochen beeinflusst wird, wie es transformiert werden muss. Diese Information muss nun auf jedes beeinflusste Vertex gerechnet werden. Folgende Methode wird beim Update eines Objekts *mesh* zwischen jedem Zeichnen ausgeführt:

Listing 3.2: applySkeletonToMesh

```
1
1
2  public void applySkeletonToMesh(){
3   doOnce{
4    //we only have to do this once, since the
5    //initial boneMatrix never changes
6    skeleton.getRootBone().updateBoneMatrix();
7   }
8   //updates the other matrix according to the
9   //tranformation of the bones
10  skeleton.getRootBone().updateRelativeBoneMatrix();
11
12  //iterate over all influenced Vertices
13  for (VertexBoneWeight[] weightList : vertexWeightMap.values()) {
14
15   //get the Vertex which is influenced
16   int vertIndex = weightList[0].vertexIndex;
17   Vector3D old = vertices[vertIndex].clone();
18
19   //create a new Vertex for this
20   Vector3D newVert = new Vector3D();
21
22   //iterate over the bones, that influence the current vertex
23   for (VertexBoneWeight vbw : weightList) {
24    float weight = vbw.weight;
25    Bone b = skeleton.getBones().get(vbw.boneIndex);
26    //the inverted static basic
27    Matrix16f bMat = b.boneMatrix;
28    Matrix16f rbMat = b.relBoneMatrix;
29
30    Matrix16f bone = rbMat.clone();
31
32    //multiply the bone with it's inverted
33    //static inverted basic matrix
34    bone.multiply(bMat);
```

```
35      //this is only for demonstration
36      //(it's better if each bone would
37      //calculate their difference matrix beforehand)
38
39
40      float[] boneMat = bone.m;
41
42      //add the resulting bone matrix to the vector
43      //according to its weight
44      //the resoult is the weighted sum of all bones
45      //that influence this vertex
46      newVert.x += (old.x * boneMat[0] + old.y * boneMat[4]
47        + old.z * boneMat[8] + boneMat[12])
48        * weight;
49      newVert.y += (old.x * boneMat[1] + old.y * boneMat[5]
50        + old.z * boneMat[9] + boneMat[13])
51        * weight;
52      newVert.z += (old.x * boneMat[2] + old.y * boneMat[6]
53        + old.z * boneMat[10] + boneMat[14])
54        * weight;
55    }
56
57   //apply the new vertex
58   skinnedVertices[vertIndex] = newVert;
59   }
60
61 }
```

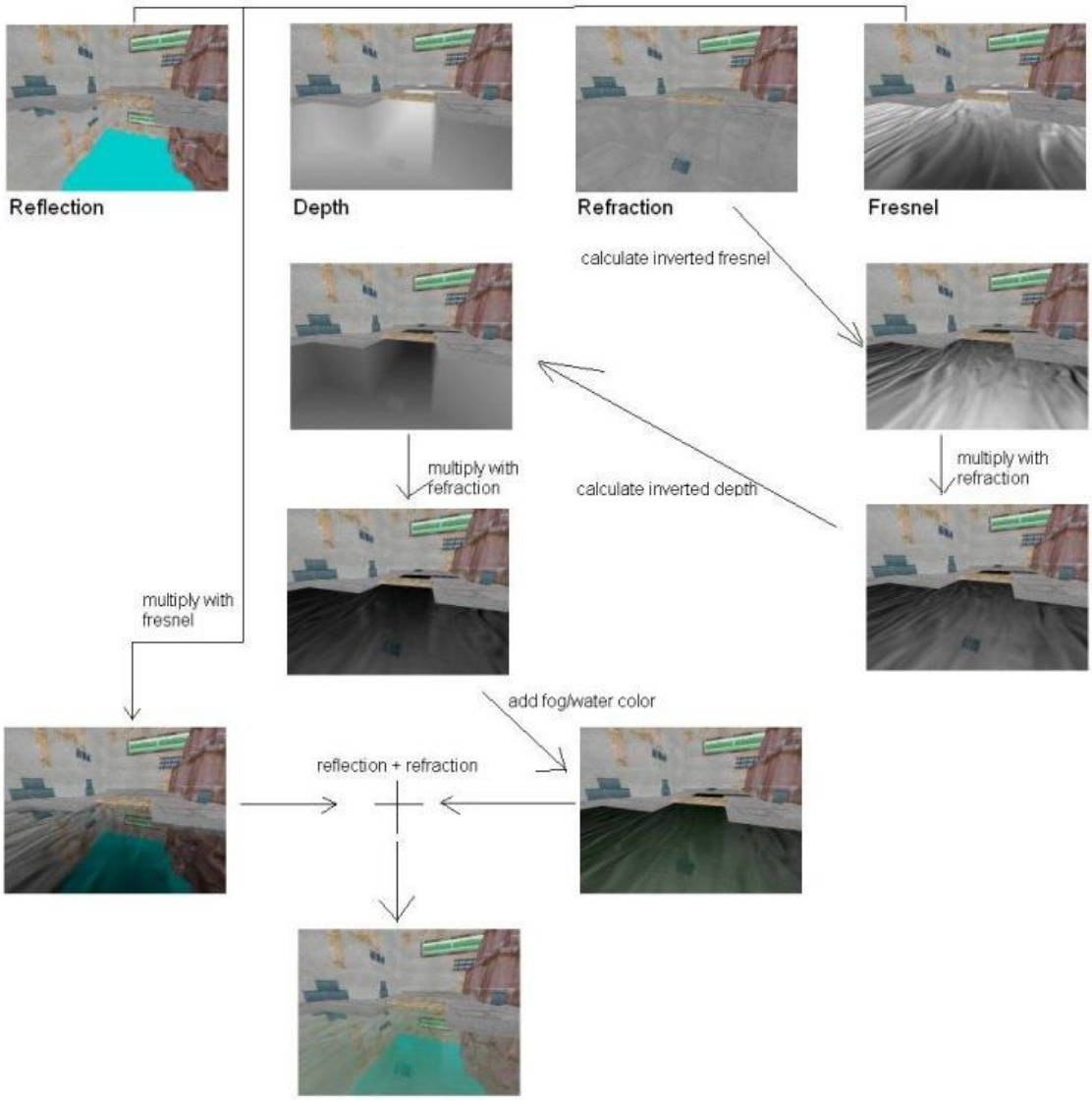

Abbildung 3.5: Flowchart eines Wassereffekts

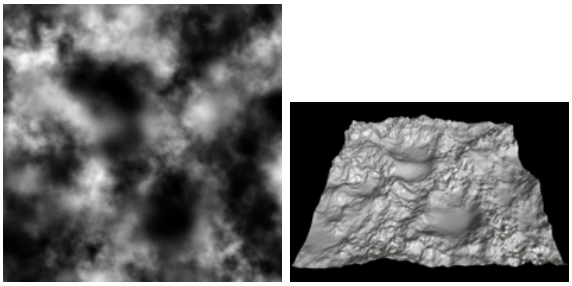

Abbildung 3.6: Heightmap als Bild und gerendert

4 Ausblick

Mit dem Programm VIR ist die Grundstruktur gegeben, um größere virtuelle Welten zu erstellen. Die nächsten Schritte in dieser Entwicklung wären die Einbindung einer Datenbank zur Charakter- und Objekt-Zustandssicherung und die Möglichkeit der Interaktion mit Objekten. Dies würde den Spielern beispielsweise ermöglichen auf eine Leinwand PDF Dokumente zu projizieren oder Musik abzuspielen. Des Weiteren muss der Spieler die Möglichkeit haben, selbst Objekte erstellen und diese mit Funktionalität versehen zu können. VIR bietet bereits gute Möglichkeiten, um 3D-Objekte, Animation und Texturen zu laden, doch am Inhalt fehlt es noch sehr. Ziel wäre es, eine Art Campuslandschaft auf verschiedenen vernetzten Inseln zu erstellen, auf denen jede Universität ihren eigenen virtuellen Campus besitzt, aber auch neutrale Plätze existieren, die für jeden Nutzer öffentlich zugänglich sind. Dort könnten dann virtuell Vorlesungen und Seminare gehalten werden. Natürlich bräuchte man dazu eine Art Sprachübertragung und die Möglichkeit, möglichst unkompliziert Medien (Bild, Ton, Film) zu zeigen.

Mit all diesen Features wäre VIR eine sehr gutes Mittel Universitäten noch enger zu vernetzen. Ziel ist es, jedem Studenten bzw. Dozenten die Möglichkeit zu geben, jederzeit jede beliebige Vorlesung zu hören bzw. halten.

Literaturverzeichnis

[1] Jogl. https://jogl.dev.java.net/, 2009.

[2] Lighthouse 3d. http://www.lighthouse3d.com, 2009.

[3] Nehe productions. http://nehe.gamedev.net/, 2009.

[4] Nehe productions. http://nehe.gamedev.net/, 2009.

[5] Ogre 3d. http://www.ogre3d.org/, 2009.

[6] Wikipedia. http://www.wikipedia.com, 2009.

[7] Michael Horsch. Bonzai software. http://www.bonsaisoftware.com, 2007.

[8] Mike Wessler. Pdf view. http://www.sun.java.com/pdfView, 2009.