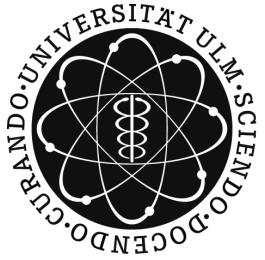

ulm university universität
uulm

Fakultät für
Ingenieurwissenschaften
und Informatik
Institut für Verteilte Systeme

Virtuelle Präsenz mittels transaktionaler Konsistenz

Dissertation

zur Erlangung des Doktorgrades Dr. rer. nat.
der Fakultät für Ingenieurwissenschaften und Informatik
der Universität Ulm

vorgelegt von

Markus Fakler

aus Memmingen

Dezember 2008

Amtierender Dekan:	Prof. Dr. Michael Weber
Gutachter:	Prof. Dr. Peter Schulthess
Gutachter:	Prof. Dr. Michael Weber
Tag der Promotion:	10.12.2008

Danksagung

Ich möchte mich sehr bei meinem Doktorvater Prof. Dr. Peter Schulthess für die vielen Gespräche, die ergiebigen Diskussionen und Anregungen während der Zeit meiner Arbeit in den letzten Jahren bedanken, die wesentlich zum Gelingen dieser Doktorarbeit beigetragen haben. Ebenso möchte ich Prof. Dr. Michael Weber für die Begutachtung dieser Arbeit danken.

Ich danke auch all meinen Arbeitskollegen, die mir bei fachlichen Fragen unterstützend halfen und mir darüber hinaus immer eine freundschaftliche und angenehme Arbeitsatmosphäre boten.

Insbesondere gilt meinen Eltern großen Dank dafür, dass sie mir das Studium und das Einschlagen dieser Laufbahn ermöglicht haben und mich auch im Hintergrund immer wieder auf diesem Weg bestärkten und mir zur Seite standen.

Schließlich möchte ich auch meiner Freundin Nina herzlich danken, die mich während dieser Arbeit immer verständnisvoll begleitet hat und mir dadurch eine große Stütze war.

Für Rosa und Albert.

Inhaltsverzeichnis

1. Einleitung

1.1 Motivation

Programme und Anwendungen, die sich auf computergenerierte virtuelle Welten oder 3D-Umgebungen stützen und eine Kommunikation zwischen den Benutzern ermöglichen, halten heutzutage in zunehmendem Maße in vielfältigen Bereichen des täglichen Lebens Einzug. Primär werden sie bislang mit Online-Spielen für die Freizeitgestaltung in Verbindung gebracht, aber auch im geschäftlichen Einsatz für Unternehmen sind sie in zunehmenden Maße nicht mehr wegzudenken, wie z.B. verschiedene Konferenzsysteme.

Dabei bewegen sich diese virtuellen 3D-Welten mittlerweile oft auf qualitativ hohem Niveau, so dass sie nicht mehr länger nur mit einfachen 3D-Umgebungen oder rudimentären Kommunikationsmöglichkeiten zwischen Benutzern wie noch in der Anfangsphase vor einigen Jahren gleichzusetzen sind. Sie haben sich vielmehr bereits zu vollständigen Präsenzsystemen entwickelt, welche für die Teilnehmer neue und vielfältige Begegnungsmöglichkeiten und fortgeschrittene Kommunikationsformen bereitstellen und dadurch auch mit zu ihrer beständigen Verbreitung beitragen.

Mit dem weiteren Zuwachs an verfügbaren Programmen und Anwendungen aus diesem Bereich und der fortschreitenden Verbesserung ihrer Qualität geht auch der Trend einher, dass immer mehr Menschen in unserer Gesellschaft viel Zeit in virtuellen Umgebungen verbringen. Dieser Trend wird sich noch weiter verstärken – sei es, dass die Menschen aus beruflichen Gründen oder zum Zeitvertreib, zum Verdienen des Lebensunterhalts oder auch zum Einkaufen von Sach- und ebenso virtuellen Gütern sich in der Virtualität aufhalten.

Betrachtet man dabei den Aufbau und die Strukturen verschiedener virtueller Welten aus technischer Sicht genauer, so fällt allerdings schnell auf, dass nahezu ausnahmslos alle Arten von virtuellen Realitäten nach der gleichen, historisch gewachsenen Grundstruktur innerhalb der zugehörigen Anwendung agieren, obwohl diese nahezu einheitliche Vorgehensweise für viele der Einsatzzwecke dieser Programme nicht immer zum Vorteil für die jeweilige Anwendung gereicht. Eine alternative oder zumindest teilweise individuellere Arbeitsweise würde in vielen Fällen zu einer Vereinfachung der Umsetzung und der Arbeitsweise des damit aufgebauten Präsenzsystems oder auch zu einer Erhöhung ihrer Leistungsfähigkeit verhelfen und dadurch kürzere Entwicklungszeiten ermöglichen oder in einer einfacheren Bedien- und Wartbarkeit resultieren.

Die von einem Präsenzsystem verwendeten Szenarien von virtuellen Umgebungen stützen sich dabei in der Regel auf einen sogenannten Szenengraphen – eine Datenstruktur, welche den Aufbau dieser Umgebungen beschreibt. Die üblicherweise eingesetzten Szenengraphen sind dabei fast immer vom ursprünglichen Design her auf einen Einzelplatz ausgerichtet. Da aktuelle virtuelle Welten aber zu einem wesentlichen Teil verteilte Komponenten beinhalten – so sollen sich in diesen virtuellen Realitäten viele Personen und nicht nur der Benutzer selbst aufhalten können –, werden die Verteilungsaspekte über nachträglich eingebrachte Schichten in die Anwendung eingefügt, welches einen großen Aufwand für die dabei ebenfalls benötigte Synchronisierung und Konsistenzhaltung der verteilten Daten nach sich zieht. Ein vom grundlegenden Design bereits auf Verteilung ausgelegter Szenengraph und ein dazugehöriges Konsistenzmodell würde an dieser Stelle bedeutende Vorteile gegenüber dem traditionellen Verfahren mit sich bringen und eine aufwändige und umständliche Anpassung im Nachhinein überflüssig machen.

Im Rahmen dieser Arbeit wurde daher ein alternativer und neuartiger Ansatz für einen solchen Szenengraphen entwickelt, der diese angesprochenen Eigenschaften erfüllt.

1.2 Überblick über diese Arbeit

In dieser Dissertation wird erstmalig ein Ansatz zum Aufbau und zur Realisierung von virtuellen Präsenzsystemen mittels der Verwendung eines transaktionalen Szenengraphens vorgestellt. Dieser Ansatz bietet etliche Vorzüge gegenüber bislang verwendeten Szenengraphenansätzen, welche im weiteren Verlauf detailliert betrachtet werden.

Im vorliegenden Kapitel werden grundlegende Konzepte zur Verteilung und zur (transaktionalen) Konsistenz beschrieben, welche als Ausgangsbasis für die darauf aufbauenden Ansätze dieser Arbeit dienen.

Im Anschluss daran wird in Kapitel zwei auf das zentrale Forschungsgebiet der virtuellen Präsenz eingegangen, da sich die zugehörige Terminologie und deren Inhalte in diesem relativ jungen Bereich noch nicht allgemein etabliert haben. Es werden Beispiele zu bestehenden Systemen aufgeführt und die von ihnen eingesetzten Konzepte aufgezeigt. An dieser Stelle wird ebenfalls der im Rahmen dieser Arbeit entwickelte Prototyp eines eigenen virtuellen Präsenzsystems vorgestellt, welcher mit den nachfolgend in dieser Arbeit vorgestellten, neuartigen Konzepten arbeitet.

Im dritten Kapitel wird dann als eigentlicher Kern dieser Arbeit das neuartige Konzept eines transaktionalen Szenengraphens vorgestellt und anschließend detailliert erläutert, welche Vorzüge sich durch diesen alternativen Ansatz ergeben. Dies umfasst dabei die genaue Analyse bestehender und vergleichbarer Arbeiten aus diesem Umfeld, sowohl für Einzelplatzsysteme als auch für verteilte Ansätze, um die Unterschiede (aber auch die Gemeinsamkeiten) zum hier vorgestellten, eigenen Ansatz klarer zu verdeutlichen.

Kapitel vier verfolgt eine hauptsächlich theoretische Betrachtung, in welcher die Hardwarestrukturen heutiger Rechnerkomponenten – und im Speziellen die von Grafikkarten – dahingehend untersucht werden, ob sich deren Aufbau und Arbeitsweise eignen, um die Strukturen und Anforderungen eines transaktionalen Szenengraphens auch direkt auf der Hardwareebene zu unterstützen.

Breitgefächerte Messungen am realisierten Prototypen belegen im fünften Kapitel die Aussagen dieser Arbeit und zeigen die dabei gewonnenen Erkenntnisse auf.

Mit der Zusammenfassung und einem Ausblick auf weiterführende Entwicklungen und Arbeiten, auch in Verbindung mit dem erstellten Prototypen, schließt diese Arbeit ab.

1.3 Verteilter virtueller Speicher

Um Daten zwischen vernetzten Rechnern auszutauschen, haben sich unterschiedliche Vorgehensweisen etabliert. Im Wesentlichen lässt sich dies auf zwei gegenläufige Ansätze beschränken, welche auch zugleich als Unterscheidungskriterien für verteilte Systeme herangezogen werden können.

Der traditionelle und bislang auch verbreitetere Ansatz ist ein expliziter Nachrichtenaustausch zwischen den kommunizierenden Rechnern ('Message Passing'). Die auszutauschenden Daten werden hierbei über Nachrichten an die anderen Rechner gesendet. Da bei verteilten Aufgabenstellungen sehr bald Synchronisierungsaufgaben und Konsistenzbetrachtungen zum reinen

Datenaustausch hinzukommen, wird dieser Ansatz in den meisten Fällen durch eine klassische Client-Server-Architektur umgesetzt. Ein einzelner, spezialisierter Server stellt dabei die zur Verteilung vorgesehenen Daten für mehrere Klienten zur Verfügung. Ein Zugriff auf diese gemeinsamen Daten geschieht dann immer über den Server, welcher als zentraler Ansprechpartner fungiert. Eine direkte Kommunikation der einzelnen Klienten untereinander wird nicht vorgenommen (siehe Abbildung 1, linker Teil). Durch die Bereitstellung der Daten von einer zentralen Stelle aus wird in solchen Systemen die Wahrung der Datenkonsistenz deutlich erleichtert. Alle auftretenden Zugriffe auf die verteilten Daten werden von der zentralen Kontrollinstanz koordiniert und serialisiert, wodurch simultane – und dadurch konsistenzgefährdende – Zugriffe mehrerer Klienten vermieden werden.

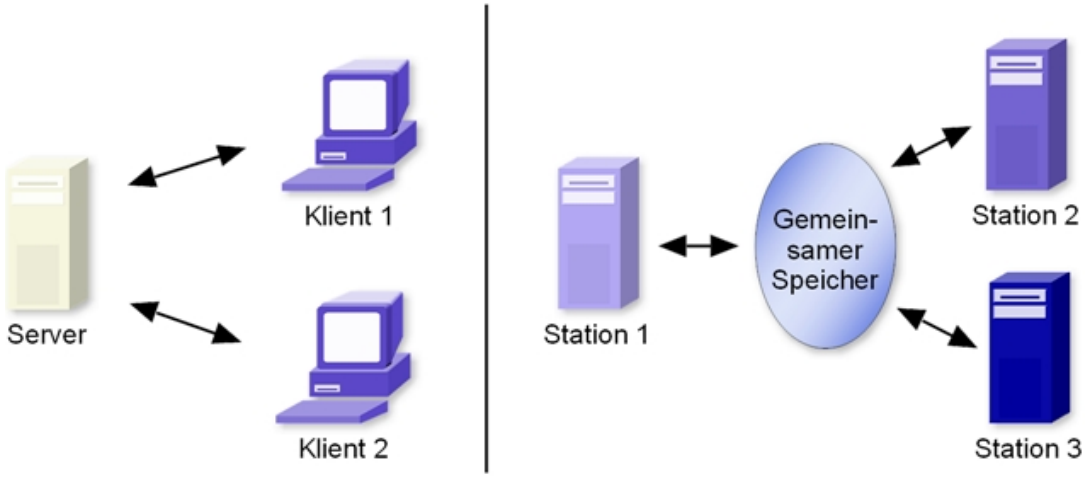

Abbildung 1: Arten der Rechnerkommunikation

Allerdings stellt die zentrale Instanz eine potentielle Engstelle innerhalb eines verteilten Systems dar, welche insbesondere die Skalierbarkeit einschränkt. Um die potentielle Last eines Servers zu mindern, wird deshalb oftmals eine Aufteilung der von ihm verwalteten Daten über mehrere verschiedene Server hinweg vorgenommen, wobei im klassischen Szenario jeweils immer nur ein Datum von einem einzelnen, hierfür zuständigen Server verwaltet wird (d.h. es wird keine weitergehende Replikation der Daten zwischen den Servern vorgenommen).

Der zweite Ansatz für eine Verteilung von Daten nutzt einen gemeinsamen Speicher als Grundlage für den Datenaustausch und zur Kommunikation zwischen den vernetzten Rechnern. Dieser verteilte gemeinsame Speicher ('Distributed Shared Memory', DSM; als deutsche Übersetzung geläufiger: verteilter virtueller Speicher, VVS) wird von allen teilnehmenden Rechnern gleich gesehen und ist von ihnen auch durch eine einheitliche (virtuelle) Adressierung ansprechbar. Ein zentraler Server zur Verwaltung der darin befindlichen Daten ist daher nicht mehr notwendig und die teilnehmenden Rechner entsprechen gleichberechtigten Stationen im Sinne eines Peer-to-Peer-Systems (siehe Abbildung 1, rechter Teil). Systeme, welche einen VVS verwenden, lassen sich daher mit steigender Anzahl an Stationen leichter skalieren. Für den Datenaustausch selbst legt eine Station hierbei die zu verteilenden Daten innerhalb des gemeinsamen Speichers ab, auf welchen die anderen Stationen anschließend zugreifen und diese Daten auslesen können. Der Zugriff auf die verteilten Daten erfolgt dabei wie ein gewöhnlicher Speicherzugriff; eine Generierung von expliziten Nachrichten durch das Anwendungsprogramm ist hierzu nicht erforderlich.

Der VVS selbst entspricht dabei nicht dem physikalisch vorhandenen Speicher der einzelnen

Stationen – im Gegensatz zu der tatsächlichen Sichtweise eines Servers auf seinen angesprochenen Speicher –, sondern einer logischen bzw. virtuellen Sicht auf einen übergreifenden Speicher aller teilnehmenden Stationen. Die Umsetzung zwischen dem virtuellen Adressraum innerhalb des VVSs und den tatsächlichen physikalischen Speicheradressen wird von einer Speicherverwaltung vorgenommen, welche entweder als Zwischenschicht bzw. Middleware einer Anwendung zur Verfügung gestellt wird oder bereits im Betriebssystem integriert ist und dadurch den höheren Schichten transparent verborgen bleibt. Anhand der Art und Weise der Realisierung solcher Systeme lassen sich weitere Unterscheidungen und Kategorisierungen von VVS-Systemen vornehmen; eine detaillierte Übersicht hierzu findet sich u.a. in [1], [2] und [3].

Der genaue Speicherort eines Datums innerhalb eines VVSs ist dabei nicht festgelegt, lediglich die zugehörige virtuelle Adresse ist eindeutig. Ob benötigte Daten in einem Rechner lokal vorhanden sind oder zuerst von außerhalb, d.h. von einer anderen Station, übermittelt und eingelagert werden müssen, wird von der Speicherverwaltung während eines Speicherzugriffs überprüft und erkannt. Für die Anwendung bleibt dieser Vorgang vollständig verborgen, d.h. es muss keine Unterscheidung für diese Fälle im Programmablauf vorgenommen werden. Dieser Ansatz realisiert dadurch eine Sichtweise aller teilnehmenden Stationen entsprechend einem einzelnen Gesamtsystem ('Single System Image', SSI; [4]) – zumindest in Bezug auf den Speicher.

Die Umsetzung der zugehörigen Kommunikationsart auf System- und Netzwerkebene zur Übermittlung von Daten kann dabei je nach System unterschiedlich realisiert werden und ist nicht zwingend mit dem VVS-Konzept festgelegt, sondern vielmehr orthogonal hierzu zu betrachten. So kann ein VVS sowohl tatsächlich direkte Speicherzugriffe auf einen gemeinsamen, physikalisch vorliegenden Speicher vornehmen (beispielsweise beim Zugriff von Multiprozessoren auf einen gemeinsamen Hauptspeicher), oder diese Speicherzugriffe können sogar in einer Realisierung als Nachrichten in tieferen Schichten resultieren (beispielsweise bei einer Bus-basierten Vernetzung wie bei Ethernet). Diese Umsetzung wird von der jeweils eingesetzten Speicherverwaltung vorgenommen; nur für die darauf aufsetzenden Anwendungen bzw. höheren Schichten wird ein regulärer VVS mit einem direkten Speicherzugriffsmodell angeboten.

Ein weiterer Unterschied zwischen diesen beiden Ansätzen für einen verteilten Speicher stellt die Benachrichtigung einer Anwendung beim Auftreten von Änderungen an den zugehörigen Daten dar. So verwenden auf Nachrichtenaustausch aufbauende Ansätze bislang nahezu ausnahmslos ereignisbasierte Mechanismen, um einer Anwendung die Änderung von Daten mitzuteilen, zumeist sogar direkt gekoppelt mit dem Empfang dieser Aktualisierungsnachrichten. VVS-basierte Ansätze stellen dahingegen in der Regel die geänderten Daten nach einer Aktualisierung lediglich innerhalb des gemeinsamen Speichers zur Verfügung; eine Anwendung wird hierbei nicht direkt informiert. Soll auf Änderungen eines bestimmten Datums reagiert werden, so muss sie selbst durch periodisches Abfragen überprüfen, ob es in der Zwischenzeit modifiziert wurde.

Sowohl der nachrichtenbasierte Austausch als auch die Nutzung eines VVSs für verteilte Systeme gewährleisten aber von sich aus noch nicht die Konsistenz der damit ausgetauschten Daten. Dies wird erst durch die jeweils vorgenommene Realisierung festgelegt (eine Umsetzung als Client-Server-Architektur ermöglicht beispielsweise die automatische Wahrung der Konsistenz für die gemeinsamen Daten durch den alleinigen Zugriff des Servers) oder muss ansonsten durch separate Mechanismen berücksichtigt werden.

1.4 Transaktionale Konsistenz

Es existieren verschiedene Konsistenzmodelle, um bei vernetzten Rechnern eine einheitliche Sichtweise von durchgeführten Änderungen auf den verteilten, gemeinsamen Daten zu gewährleisten (einen umfassenden Überblick hierzu bietet u.a. [2] und [3]). Abhängig vom gewählten Konsistenzmodell[1] ergeben sich dadurch unterschiedliche Anforderungen bezüglich der Sichtbarkeit und des Abgleichs der geänderten Daten und deren zeitlichen Abfolge sowohl an das System, welches dieses Modell umsetzt, als auch an die damit arbeitende Anwendung.

Ein spezielles Konsistenzmodell, welches an der Universität Ulm entwickelt wurde, stellt die transaktionale Konsistenz dar, welche Transaktionen zur Programmausführung verwendet. Die transaktionale Konsistenz lässt sich zwischen der strikten und sequentiellen Konsistenz einordnen. Standardmäßig realisiert sie eine sequentielle Konsistenz für verteilte Daten; zum Zeitpunkt des Abschlusses einer Transaktion (d.h. am Ende der darin vorkommenden Validierungsphase) ist sogar kurzfristig strikte Konsistenz vorhanden [2].

Transaktionen stammen ursprünglich aus dem Umfeld von Datenbanken und bezeichnen eine Abfolge von Operationen und Berechnungen, welche entweder als Ganzes einem System gegenüber sichtbar werden oder gar nicht. Sie erfüllen dabei die sogenannten 'ACID'-Bedingungen:

- Atomarität ('Atomicity'): Eine Transaktion wird entweder als Ganzes oder gar nicht ausgeführt. Wird eine Transaktion nicht erfolgreich beendet („abgebrochen“), so hat sie auch keine Auswirkungen auf das Gesamtsystem.
- Konsistenz ('Consistency'): Ein konsistentes System wird durch eine Transaktion wiederum in einen konsistenten Zustand überführt, d.h. es treten nach der Ausführung einer Transaktion keine Inkonsistenzen durch die vorgenommen Operationen auf.
- Isolierung ('Isolation'): Die Operationen und Abläufe einer Transaktion sind vor deren Abschluss für andere Transaktionen nicht sichtbar und beeinflussen sich somit gegenseitig auch nicht.
- Dauerhaftigkeit ('Durability'): Nach dem Abschluss einer Transaktion sind die von ihr vorgenommenen Änderungen von Dauer, d.h. sie sind vom gesamten System wahrnehmbar und stehen auch nach eventuellen Fehlersituationen weiterhin korrekt zur Verfügung.

Bei der transaktionalen Konsistenz entsprechen die Transaktionen einer Kapselung von Programmabschnitten, welche ebenfalls diese ACID-Bedingungen erfüllen.

Zusätzlich zu den ACID-Bedingungen und ähnlich wie bei Datenbanksystemen wird bei der transaktionalen Konsistenz im Abbruchfall, d.h. bei einem erzwungenen Abbruch einer Transaktion infolge eines durch eine andere, schreibende Transaktion ausgelösten Konflikts auf gemeinsame Daten, eine Rücksetzung der abgebrochenen Transaktion ermöglicht. Diese Rücksetzung bedeutet ein Zurücksetzen des Zustandes der abgebrochenen Transaktion auf den Ursprungszustand vor

1 Die stärkste dabei möglich auftretende Konsistenzform ist die strikte Konsistenz. Bei ihr sehen alle Teilnehmer bei einem Datenzugriff immer den Wert der global betrachtet letzten Schreiboperation auf diese Daten. Dies ist in der Regel nur bei Einzelplatzmaschinen (mit auch nur einem Prozessorkern) der Fall, da bei mehreren Rechnern bzw. Prozessorkernen allein schon durch Nachrichtenlaufzeiten eine global gültige und gleichzeitige Sichtbarkeit bei simultan vorgenommen Änderungen nicht mehr gewährleistet werden kann.
Die bislang verwendete, stärkste Konsistenzform bei mehreren Rechnern bzw. Prozessen stellt die sequentielle Konsistenz dar, in der alle Teilnehmer vorgenommene Änderungen in der jeweils von ihnen durchgeführten Reihenfolge wahrnehmen.
Schwächere Konsistenzmodelle erlauben weitere Einschränkungen was die Reihenfolge und Sichtbarkeit einzelner Berechnungen für ein Gesamtsystem angeht bis hin zu unvorhersagbaren Sichtweisen auf Dateninhalte für nicht konsistente Fälle.

Beginn der Programmausführung. Bei einem Neustart dieser Transaktion entsprechen die zuvor geänderten Daten, welche zum Abbruch geführt haben, den nun aktuellen Daten, so dass die erneute Ausführung auf wieder gültigen und nicht den mittlerweile veralteten Daten aus der vorherigen Abarbeitung vorgenommen wird.

1.5 Plurix – Ein verteiltes Betriebssystem

Ein Betriebssystem, welches die zuvor beschriebenen Eigenschaften eines VVS-Konzeptes mit der transaktionalen Konsistenz kombiniert, stell 'Plurix' dar. Plurix ([2], [5], [w1]) ist ein neuartiges Clusterbetriebssystem, welches im Institut für Verteilte Systeme der Universität Ulm entwickelt wurde[2]. Es ist als verteiltes Betriebssystem von Grund auf für einen Verbund von Rechnern konzipiert, kann aber auch auf einer Einzelplatzmaschine ohne angeschlossenes Netzwerk verwendet werden. Die eigentlichen Stärken von Plurix kommen allerdings erst durch das Zusammenspiel mehrerer Rechner zum Ausdruck.

Plurix selbst ist ein vollständig in der Hochsprache Java geschriebenes, objektorientiertes Betriebssystem (inklusive der Gerätetreiber), welches sich mittels eines eigenen Plurix-Java-Compilers direkt in nativen x86-kompatiblen Maschinencode übersetzt lässt. Infolgedessen wird zur Ausführung von Plurix keine Java Virtual Machine zur Interpretierung eines Bytecodes als Zwischeninstanz benötigt, wie es bei normalen Java-Applikationen der Fall ist. Dadurch wird auch die von der Hardware zur Verfügung gestellte Leistung vollständig für das Plurix-System nutzbar gemacht.

Durch eine transparente Bereitstellung der Ressourcen aller beteiligter Rechner im Cluster für einen Benutzer realisiert Plurix das SSI-Konzept. D.h. jeder Benutzer hat eine einheitliche Sichtweise auf den gesamten Rechnerverbund und der dabei zur Verfügung stehenden Ressourcen, so als wäre anstelle des Clusters nur eine große Einheit vorhanden, auf die dann auch wie auf einen einzelnen Rechner zugegriffen werden kann, welche allerdings die aggregierte Leistung aller enthaltenen Clusterrechner besitzt. Die tatsächliche Lokalisierung von Systemvorgängen, wie die eigentliche Verteilung von Daten oder Prozessen über Rechnergrenzen hinweg, bleibt dem Benutzer dabei transparent durch das Betriebssystem verborgen.

Die Kommunikation und der Datenaustausch innerhalb eines Plurixclusters mit den anderen Rechnern wird über ein weiterentwickeltes Konzeptes eines VVSs geregelt: einem verteilten Heap ('Distributed Heap Storage', DHS; [5]). Innerhalb dieses DHSs werden alle von Plurix verwendeten Objekte abgelegt. Der DHS wird dabei von der Hardware über die Speicherverwaltungseinheit ('Memory Management Unit', MMU) unterstützt und arbeitet somit seitenbasiert[3].

Die im DHS abgelegten Objekte sind über einen Namensdienst von jeder Station aus referenzierbar. Bei einem Zugriff einer Station auf ein fremdes, d.h. lokal nicht präsentes Objekt, wird automatisch ein Seitenfehler durch die MMU für die dem Objekt zugeordnete(n) Speicherseite(n) ausgelöst, welcher das Betriebssystem infolgedessen dazu veranlasst, die fehlende(n) Seite(n) von demjenigen Rechner anzufordern, der das zugehörige Objekt tatsächlich vorhält. Nachdem diese lokal

2 Inzwischen wird bereits ein Nachfolgesystem zu Plurix mit dem Namen 'Rainbow-OS' entwickelt, welches u.a. eine Erweiterung des ursprünglichen 32-Bit-Betriebssystems auf eine Unterstützung von aktuellen 64-Bit-Architekturen beinhaltet und in Kürze fertiggestellt sein wird. Während des Zeitraums dieser Arbeit stand es allerdings noch nicht zur Verfügung, weshalb im Folgenden weiterhin Plurix und noch nicht Rainbow-OS referenziert wird. Die konzeptuellen Eigenschaften von Plurix sind aber ebenfalls in Rainbow-OS enthalten und somit auch auf dort angesiedelte und weiterführende Arbeiten übertragbar.

3 Die von der Hardware unterstütze Seitengröße kann dabei entweder 4 KiB oder 2 MiB betragen. Standardmäßig werden bei Plurix 4 KiB-Seiten verwendet.

eingelagert und dadurch verfügbar gemacht wurden, läuft die Anwendung an derjenigen Stelle weiter, wo der Zugriff auf das fehlende Objekt ursprünglich statt fand. Der Anwendung und somit auch dem Benutzer selbst bleibt dieses Verhalten von Plurix dabei verborgen.

Alle für diesen Mechanismus benötigten Strukturen zum Datenaustausch sind direkt im Betriebssystemkern von Plurix verankert. Es muss somit weder für einen Benutzer noch für die restlichen Teile des Betriebssystems eine Unterscheidung zwischen lokalen und entfernten Objektzugriffen stattfinden, der Zugriffsmechanismus bleibt in beiden Fällen identisch.

Eine Programmausführung geschieht in Plurix mit Hilfe von Transaktionen, über welche sich auch die zuvor beschriebene transaktionale Konsistenz realisieren lässt. Die Transaktionen werden dabei im Betrieb von Plurix optimistisch synchronisiert ([6], [7]), d.h. es wird davon ausgegangen, dass in der Regel keine oder nur selten Konflikte mit anderen Transaktionen beim Modifizieren von Daten auftreten. Für den Fall einer erforderlichen Rücksetzung von modifizierten Daten wegen eines doch aufgetretenen Abbruchs werden vor Schreibzugriffen automatisch sogenannte „Schattenkopien" der zu modifizierenden Daten angelegt, mit welchen der ursprüngliche Zustand wieder hergestellt werden kann. Abgebrochene Transaktionen werden nach der Rücksetzung automatisch von Plurix neu gestartet, so dass eine Anwendung in ihrem Ablauf von einem aufgetretenen Konflikt für gewöhnlich nichts wahrnimmt.

Persistenz innerhalb von Plurix wird mit Hilfe eines optionalen 'Page-Servers' realisiert [8]. Dieser Page-Server läuft parallel zum normalen Plurix-System und fertigt periodisch konsistente Abbilder des DHSs auf einer Festplatte an. Beim Neustart eines Plurix-Clusters, aber auch bei einem eventuellen Fehlerfall, wenn beispielsweise durch den Ausfall einer Maschine inkonsistente oder nicht mehr abrufbare Daten auftreten, liefert der Page-Server das zuletzt gültige Speicherabbild an den Cluster zurück und dieser kann die Arbeit ab der gespeicherten Position wieder aufnehmen. Die für die Sicherung erforderliche Zeit beträgt dabei einige Millisekunden bis Sekunden, im Gegensatz zu automatischen Sicherungsmaßnahmen vergleichbarer kommerzieller Checkpointing-Systeme, welche hierfür im höheren Minutenbereich liegen (siehe [8]).

2. Virtuelle Präsenz

2.1 Einführung

Virtuelle Präsenz stellt ein relativ junges Forschungsgebiet dar. Dies lässt sich unter anderem daran erkennen, dass der Begriff „Virtuelle Präsenz" selbst in der Literatur noch nicht einheitlich geprägt ist. Demgegenüber ist der Präsenzbegriff an sich weit verbreitet, wird aber in erster Linie im Bereich der Psychologie verwendet und beschrieben, worauf hier aber nicht weiter eingegangen wird. Im Rahmen dieser Arbeit wird Präsenz (und virtuelle Präsenz im Speziellen) in einem informatikspezifischen Zusammenhang betrachtet. In der Informatik ist der Präsenzbegriff selbst zwar auch bereits vorhanden, jedoch gibt es hier keine eindeutige und klare Definition, sondern vielmehr eine Fülle unterschiedlicher Ansätze und Interpretationsmöglichkeiten (vgl. [9], [10], [w3], [w4]). Eine mögliche Einteilung und Beschreibung des Präsenzbegriffs nach Riva, Davide und IJsselsteijn ([11], [w5]), welche sich auch für den in dieser Arbeit relevanten virtuellen Präsenzbegriff übertragen lässt, sei hier nachfolgend vorgestellt und wird im Folgenden weiterhin in diesem Sinne verwendet.

Riva, Davide und IJsselsteijn beschreiben den Präsenzbegriff als das komplexe, subjektive Wahrnehmungsempfinden einer Person, die Gegenwart an einem Ort, eines Gegenstandes oder anderer Personen zu fühlen. Verdeutlicht wird dies von den Autoren, indem sie dieses Wahrnehmungsempfinden und somit den damit einhergehenden Präsenzbegriff in drei Kategorien unterteilen: eine physische, eine soziale und eine sogenannte Ko-Präsenz (siehe Abbildung 2).

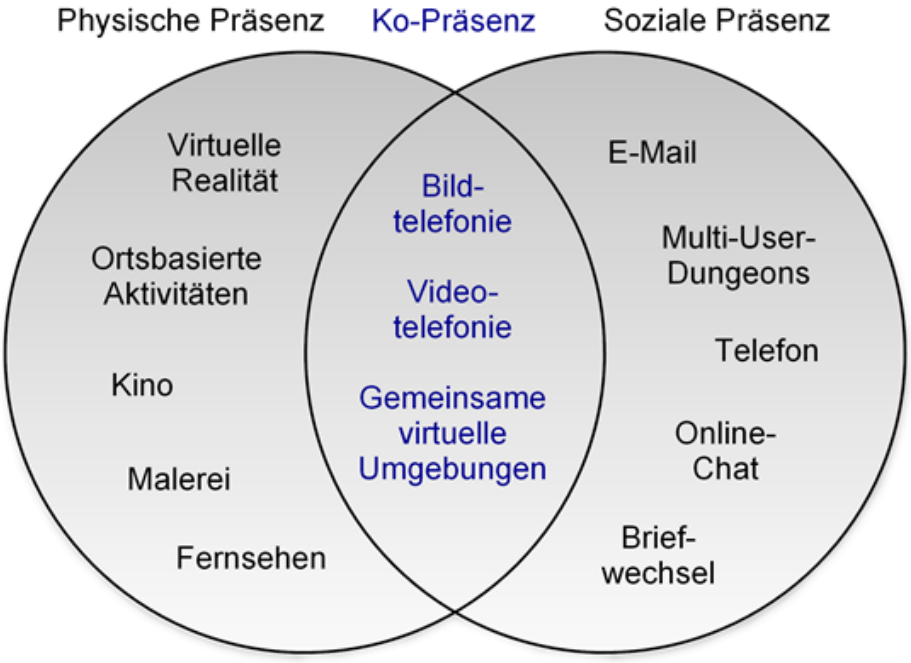

Abbildung 2: Präsenzkategorisierung nach Riva, Davide und IJsselsteijn

2.1.1 Physische Präsenz

Die physische Präsenz beschreibt das subjektive Gefühl einer Person, sich selbst an einem bestimmten Ort anwesend zu fühlen. Es ist hierbei nicht entscheidend, ob dieser Ort real oder nur virtuell existent ist, oder ob man sich für eine bestimmte Tätigkeit oder zu einem bestimmten Zweck an diesem Ort befindet bzw. einfach nur so dort ist; einzig allein der vermittelte Eindruck, sich an diesem Ort zu befinden ist für die physische Präsenz entscheidend.

Beispiele für physische Präsenzen in vermittelten Umgebungen sind alle Arten von immersiven Szenarien. Triviale Beispiele sind hierbei Fernsehen und Kino, wenn z.B. ein Zuschauer in einen Film „eintaucht" und die Realität um sich herum vergisst. Weitergehende Varianten sind virtuelle Realitäten ('Virtual Realities', VRs), wobei diese wiederum unabhängig von der dabei verwendeten Technik betrachtet werden. Es ist insgesamt somit unerheblich, ob über einen einfachen Monitor die vermittelte Umgebung erlebbar gemacht wird oder ob mit hohem technischen Aufwand zu einem bedeutend stärkeren immersiven Eindruck verholfen wird (VR-Helme, 3D-Displays, CAVEs, ...).

Zu der Kategorie der physischen Präsenz zählt ebenfalls der Bereich der erweiterten Realität ('Augmented Reality', AR), bei der eine Mischung zwischen realer und virtueller Welt vorgenommen wird, indem virtuelle Objekte (2D-, 3D-Gegenstände, Texte, ...) in die reale Wahrnehmung eingeblendet und dadurch mit einbezogen werden (zumeist über spezielle Bildschirme, VR-Helme oder Spiegelsysteme). Als ein besonderes Beispiel unter einer Fülle von Arbeiten im Bereich der AR sei hier ein Verweis zum Projekt „Studierstube" der Technischen Universität Graz erwähnt ([12], [w6]).

2.1.2 Soziale Präsenz

Die zweite, zur physischen Präsenz orthogonale Kategorie ist die soziale Präsenz. Bei der sozialen Präsenz steht das Gefühl der Zusammengehörigkeit bzw. des Zusammenseins mehrerer Personen im Vordergrund. Soziale Präsenz entspricht somit einem vermittelten Gegenwartsgefühl zwischen Personen und der freien Kommunikationsmöglichkeit dieser untereinander. Ob diese Personen sich dabei wirklich am gleichen Ort befinden ist für die soziale Präsenz nicht relevant.

Das wohl bekannteste Beispiel hierzu ist das Telefon bzw. Telefonie an sich. Hier kommunizieren zwei oder mehr Teilnehmer, welche sich dabei physisch an verschiedenen Orten befinden können, derart miteinander, als ob sie direkt nebeneinander stehen und ebenso direkt miteinander sprechen würden. Die räumliche Trennung und das Benutzen technischer Hilfsmittel, welche diese Kommunikation überhaupt erst ermöglichen, treten dabei in der Wahrnehmung bei den Gesprächspartnern während der Kommunikation in den Hintergrund und werden heutzutage auch nicht (mehr) als störend wahrgenommen bzw. beeinflussen den Gesprächsablauf nicht wesentlich. Weitergehende Beispiele sind generell Kommunikationsmedien aller Art (Briefe, E-Mails oder Chats), aber auch „klassische" Mehrbenutzercomputerspiele wie Multi-User-Dungeons, in denen die Teilnehmer hauptsächlich textuell miteinander kommunizieren.

2.1.3 Ko-Präsenz

Als Ko-Präsenz wird von Riva, Davide und IJsselsteijn die Kombination zwischen den beiden vorherigen Präsenzkategorien bezeichnet. Sie verbindet sowohl die Kommunikation zwischen verschiedenen Teilnehmern als auch deren Empfindung, sich physisch am selben Ort zu befinden. Im realen Leben entspricht dies einer direkten Begegnung und einem Gespräch von Personen. Werden technische Hilfsmittel hinzugenommen, kann der Begegnungsradius deutlich erhöht werden. Anschaulichstes Beispiel hierfür stellt die Bildtelefonie bzw. im Allgemeinen

Videokonferenzsysteme dar (siehe [13]). Des Weiteren gehören insbesondere auch alle avatarbasierten, virtuellen Welten zu dieser Kategorie (siehe Kapitel 2.3).

In den nachfolgenden Unterkapiteln werden zwei spezielle Varianten von Präsenzformen aus der sozialen und der Ko-Präsenz genauer betrachtet, da sie im informatikspezifischen Bereich und im Rahmen dieser Arbeit von besonderem Interesse sind. Dies ist zum einen die Web-Präsenz und zum anderen die virtuelle Präsenz. Vom Fachbegriff selbst tauchen sie nicht in den Beispielen der Kategorisierung von Riva, Davide und IJsselsteijn auf, da sie sich erst in den letzten Jahren neu geprägt haben und somit erst in jüngster Zeit Beispiele und Anwendungen hierzu entstanden sind. Sie lassen sich aber inhaltlich nahtlos in die bestehende Kategorisierung mit einfügen.

2.2 Web-Präsenz

2.2.1 Beschreibung

Die Web-Präsenz ist nach der oben beschriebenen Kategorisierung zur sozialen Präsenz zugehörig. Sie setzt auf die klassische Internetstruktur auf, welche ursprünglich nur anonymes Abrufen von Informationen und Webseiten in einer 2D-Browserumgebung erlaubte, ohne dabei einen direkten Kontakt bzw. eine Kommunikation mit anderen Personen aufbauen zu können (explizite Chatsysteme seien hiervon ausgenommen, da sie an sich bereits eine Weiterentwicklung zum klassischen Internet darstellen). Bei der Web-Präsenz wird dahingegen versucht, dieses „einsame" und anonyme Surfverhalten im Internet zu durchbrechen und andere Internetnutzer, die ähnliche Interessen besitzen oder sich bei ähnlichen Inhalten bzw. Webseiten aufhalten, für sich selbst sichtbar und ansprechbar zu machen.

Um dies zu ermöglichen, werden anhand zusätzlicher Dienste und Metainformationen zu den bestehenden Internetseiten sogenannte Nachbarschaftsbeziehungen aufgebaut. Dabei entsprechen diese Nachbarschaftsbeziehungen allerdings nicht den klassischen Hyperlinks, welche auf den Webseiten direkt zu finden sind und für die bisherige Internetnavigation verwendet werden, sondern es wird ein zweites, davon unabhängiges Verbindungsnetz über das Internet bzw. die Internetseiten gelegt. Diese zweite Verknüpfungsstruktur wird anhand inhaltlicher Kriterien zwischen den Webseiten erstellt und muss über separate Server bzw. Serverdienste, den sogenannten Nachbarschaftsdiensten, zur Verfügung gestellt werden. Nach welchen Kriterien diese Verknüpfungen gebildet werden variiert dabei teilweise sehr stark, je nach gesetztem Schwerpunkt den ein realisierter Nachbarschaftsdienst hierbei besonders hervorhebt (vgl. [14], [15]). Zumeist werden bestimmte Metriken zur Bewertung herangezogen, die beispielsweise Kriterien nach der inhaltlichen Übereinstimmung von Seiten, dem Verlinkungsgrad auf diese Seite oder der Besucheranzahl der Webseite heranziehen. Hieraus werden dann Gewichtungen erstellt, anhand derer das neue Verknüpfungsgeflecht bestimmt wird. Mit Hilfe eines eigenständigen Klientenprogramms oder – zumeist einfacher – über eine im Browser eingebundene Erweiterung werden diese neuen Nachbarschaftsbeziehungen dann abrufbar gemacht. Um nun andere Nutzer oder Gleichgesinnte sichtbar und ansprechbar zu machen, sobald sie sich auf der gleichen oder thematisch sehr ähnlichen Webseiten (die nicht unbedingt über Hyperlinks verbunden sein müssen) befinden, wird von den Programmen eine Art Sichtbarkeitsradius für die eigene Person berechnet, welcher beschreibt, wie „weit" man selbst in dem Nachbarschaftsgeflecht für andere sichtbar und somit auffindbar ist. Sobald dann die Sichtbarkeitsradien zweier Benutzer überlappen, können sie voneinander Notiz nehmen und werden dadurch automatisch füreinander sichtbar. Dies wird von den Klientenprogrammen mittels der Einblendung von (simplen) Figuren anderer Personen auf den jeweiligen Webseiten umgesetzt. Über diese sogenannten Avatare kann dann – sofern gewünscht –

Kontakt zu anderen Benutzern aufgenommen werden.

Dabei müssen diese Avatare in ihrer Erscheinungsform nicht unbedingt auch dem tatsächlichen Aussehen der zugehörigen Person entsprechen. Vielmehr wird einem Benutzer hierdurch die Möglichkeit geboten, beliebige andere Identitäten auszuprobieren und damit gegenüber anderen aufzutreten. Oftmals steht von einer Anwendung, welche Avatare zur Repräsentation ihrer Benutzer verwendet, sogar nur eine begrenzte Anzahl an fest vorgegebenen Figuren zur Auswahl bereit, wodurch sich alle möglichen Benutzer gar nicht individuell abbilden lassen bzw. zur Auswahl eines vom eigenen Aussehen abweichenden Avatar gezwungen sind. Nur die wenigsten avatarbasierten Systeme bieten selbst heutzutage überhaupt die Möglichkeit zur freien Konfiguration der Erscheinungsform des eigenen Avatars an.

2.2.2 Bestehende Arbeiten

Für den Bereich der Web-Präsenz gibt es nur wenige freie oder kommerzielle Anwendungen, die dieses Konzept verfolgen. Zu den bekanntesten Vertretern gehören 'CoBrow' ([16], [w7]), 'Lluna' [w8], 'Zweitgeist' [w9] und deren Nachfolger 'Weblin' [w10] aus Deutschland, sowie weitere konkurrierende Produkte wie das amerikanische 'me.dium' [w11] und das französische 'Weezu' [w12].

Abbildung 3: Der Webclient von 'Lluna' – Sichtbarkeit anderer Webuser beim Browsen

Sie alle funktionieren nach dem beschriebenen Prinzip der Einblendung von Avataren bzw. kleinen (teilweise animierten) Bildern, welche die Avatare repräsentieren, in die Internetwebseite, auf der man sich momentan beim Surfen befindet (siehe Abbildung 3 und 4). Die hauptsächlichen Unterschiede zwischen den Programmen finden sich in der Art der aufgespannten Nachbarschaftsbeziehungen (CoBrow nutzt u.a. inhaltliche Gesichtspunkte für die Nachbarschaftskriterien; die restlichen stützen sich rein auf die Webseite, auf der man sich befindet, und der darin enthaltenen Hyperlinks und lassen die eigentlich behandelten Themen großteils außen vor) und in der verwendeten Grafik.

Abbildung 4: 'Weblin'-Browsererweiterung mit Beispielavataren

Der größte Vorteil all dieser Programme gegenüber herkömmlichen Chat-Systemen besteht darin, schneller als bei traditionellen Chat-Systemen Gleichgesinnte zum Diskutieren zu treffen. Bei den klassischen Chat-Systemen muss ein Benutzer zuerst einen Sprachkanal ('Channel') zu einem bestimmten Thema auswählen, dem er sich anschließen will. Bei den Web-Präsenz-basierten Programmen reicht es aus, eine Webseite zum gewünschten Thema anzusteuern, um hierzu Gleichgesinnte zu treffen. Die vorhergehende Auswahl eines speziellen Channels entfällt dadurch. Auch kann durch einfaches Surfen auf eine andere Internetwebseite ein neues Thema ausgewählt und erneut hierfür interessierte Benutzer entdeckt werden. Dies würde einem automatischen Umschalten der Channels bei den klassischen Chat-Systemen entsprechen, welches diese aber nicht bieten (können).

Einen weiteren, interessanten Ansatz, Surfer im Web untereinander präsent zu machen, verfolgt die Firma Three-B International Limited mit ihrem '3B-Browser' [w13]. Hier wird allerdings der oben beschriebene Ansatz umgekehrt. Anstelle der Einblendung von Avataren auf eine zweidimensionale Webseite innerhalb des Webbrowsers wird beim 3B-Browser ein dreidimensionaler, virtueller 3B-Raum erstellt, in dem als Wände zweidimensionale Webseiten angezeigt werden. Dies können dabei beliebige Internetwebseiten sein, die eigenen Bookmark-Seiten oder speziell unterstützte 'MySpace'-Einträge eines Benutzers. All diese gewählten Webseiten werden gleichzeitig als eine Art Galerie angezeigt, und der Avatar kann sich der gerade gewünschten Seite einfach nähern, um sie genauer betrachten und lesen zu können (siehe Abbildung 5).

Abbildung 5: Ein begehbarer 3B-Raum mit Internetseiten als Wände beim '3B-Browser'

Ursprünglich war es mit dem 3B-Browser nur möglich, sich selbst in seinem 3B-Raum aufzuhalten, was nur einem neuen, etwas ungewöhnlichem Browserkonzept entsprach, aber noch nicht der Web-Präsenz zuordnenbar war. Inzwischen können aber auch andere Benutzer einbezogen und sichtbar gemacht werden, indem befreundete Avatare in den eigenen 3B-Raum eingeladen werden können und dann bei einem Besuch für Diskussionen bereitstehen.

Insgesamt entspricht dieser Ansatz inzwischen sogar teilweise demjenigen einer Online-Welt aus dem Bereich der virtuellen Präsenz (siehe Kapitel 2.3.2.2) oder zumindest einer Mischform aus Web-Präsenz und virtueller Präsenz; er wird hier trotzdem bei der Web-Präsenz aufgeführt, weil dies die ursprüngliche Ausrichtung dieses Projektes war. Es lässt sich an diesem Beispiel aber bereits erkennen, dass sich eine eindeutige Einteilung vieler Anwendungsprogramme zu einem einzelnen Bereich teilweise nur sehr schwer vornehmen lässt, da die Grenzen zunehmend verschwimmen.

2.3 Virtuelle Präsenz

2.3.1 Beschreibung

Der im Rahmen dieser Arbeit wichtigste Vertreter von Präsenzkategorien ist die virtuelle Präsenz. Nach der Einordnung von Riva, Davide und IJsselsteijn ist sie der Ko-Präsenz zugehörig.

Mit virtueller Präsenz sind im Folgenden ganze Präsenzsysteme bzw. -plattformen gemeint, welche ein (virtuelles) Präsenzempfinden vermitteln wollen. Es soll mit diesen Systemen erreicht werden, dass zwischen entfernten Kommunikationspartnern ein Präsenzgefühl aufgebaut werden kann, welches den Eindruck einer Vor-Ort-Begegnung entstehen und die tatsächliche Entfernung zwischen den Teilnehmern in den Hintergrund treten lassen soll. Erzielt wird dies über simulierte, virtuelle Umgebungen, die als Begegnungsstätte für die Kommunikation dienen. Hierzu gehören insbesondere alle avatarbasierten, virtuellen Mehrbenutzerwelten ('Virtual Realtiy Environments', VREs). Dabei werden erneut Avatare als virtuelle Repräsentanten eines Teilnehmers in der virtuellen Umgebung verwendet, welche entweder vom Benutzer selbst gesteuert werden oder in seinem Auftrag entsprechend handeln.

VREs sind mittlerweile im Begriff, bestehende Kommunikationsmedien wie z.B. die bisherige Telefonie und vorhandene Internet-Chatsysteme zu erweitern und teilweise auch zu ersetzen. Erste

Entwicklungen, bei denen dies erkennbar wird, sind die Fortschritte bei den bereits erwähnten Bildtelefonen und Videokonferenzen. Ein wesentliches Merkmal, welches dabei mit zu ihrer zunehmenden Verbreitung beiträgt, ist die Eigenschaft, dass bei einer stattfindenden Kommunikation zusätzlich zur gesprochenen Sprache auch das Bild des Kommunikationspartners für einen sichtbar ist und somit die in einer direkten Konversation ebenfalls üblichen und wichtigen sekundären Ausdrucksmöglichkeiten mittels Gestik und Mimik übermittelt werden und dadurch mit in die Kommunikation eingebunden sind.

Die langfristige Zielsetzung, welche bei der Entwicklung avatarbasierter VREs verfolgt wird, lässt sich im Wesentlichen dadurch beschreiben, dass bei einer Kommunikation mit einem anderen Avatar zukünftig nicht mehr unterscheidbar sein soll, ob es sich dabei um einen von einem anderen Benutzer direkt gesteuerten Avatar handelt oder um einen Avatar, der „nur“ von einem Computer gesteuert autonom agiert. Insbesondere im visuellen Bereich wird die Technik so weit voranschreiten, dass keine sichtbare Unterscheidung mehr möglich sein wird, ob es sich beim optischen Erscheinungsbild des Avatars um eine von einer Kamera übertragene Aufnahme des Kommunikationspartners handelt oder um eine computergenerierte Darstellung. Der momentane Stand der Technik macht diesen Unterschied noch leicht erkennbar. Mit dem weiteren Voranschreiten der technologischen Möglichkeiten – insbesondere im Bereich der Grafikkartenentwicklung – wird in einigen Jahren dieses Ziel, die virtuelle Repräsentation eines Kommunikationsteilnehmers durch einen generierten, fotorealistischen Avatar ununterscheidbar zu machen, verwirklicht sein. Abbildung 6 stellt zwei Beispiele hierfür dar.

Abbildung 6: Beispiele für den momentanen Realismus beim Avatardesign

Die parallele Weiterentwicklung von autonom agierenden Agenten, die im Sinne eines Benutzers eigenständig Handlungen und Gespräche erledigen – beispielsweise ein Avatar, der verschiedene Botengänge für Behörden erledigt oder gar die Meinung eines Benutzer in einer Konferenz vertritt (z.B. im Krankheitsfall oder bei überlappenden Terminen) – ist ebenso vorauszusehen. Auch dies wird noch geraume Zeit bis zum Einzug in das Alltagsleben benötigen; die Ausrichtung unterschiedlicher Forschungsbereiche in der Informatik hierauf – speziell bei der künstlichen Intelligenz – lassen aber auch hier erkennen, dass dieselbe Zielsetzung verfolgt wird [17].

Im gleichen Maße, wie sich die Kommunikationsformen ändern und erweitern werden, wird der Erfolg dieser virtueller Präsenzsysteme aber auch von der Art der Bedienung abhängen. Die klassischen Benutzerschnittstellen mit Maus und Tastatur werden zukünftig ebenfalls durch neue

Formen wie Spracheingaben und automatischer Gestenerkennung ergänzt, wenn nicht sogar ersetzt werden. VREs werden sich erst durch einfache und intuitiv zu bedienende (Komplett-)Systeme ohne großen Einarbeitungsaufwand – vergleichbar dem heutigen Umgang mit dem Telefon bzw. Handy – für den Großteil der Bevölkerung nahebringen lassen und für eine weite Verbreitung sorgen.

2.3.2 Bestehende Arbeiten

Im Gegensatz zu Programmen aus dem Bereich der Web-Präsenz gibt es innerhalb der virtuellen Präsenz inzwischen eine nahezu unüberschaubare Vielzahl an freien und kommerziell vertriebenen Programmen. Ihnen allen ist dabei gemeinsam, dass der Benutzer durch einen virtuellen Avatar repräsentiert wird und über ihn Handlungen und Aktionen in der Virtualität ausführen und dadurch Einfluss auf seine Umgebung nehmen kann.

Insgesamt lassen sich die existierenden Programme in zwei große Kategorien unterteilen, welche sich in ihrer Zielsetzung unterscheiden: den Online-Spielen und den Online-Welten. Nachfolgend werden die grundlegenden Konzepte und einige verbreitete Vertreter beider Kategorien näher erläutert.

2.3.2.1 Online-Spiele

Online-Spiele zeichnen sich dadurch aus, dass sie eine feste, vorgegebene Handlung und ein definiertes (Spiel-)Ziel besitzen, auf das die benutzergesteuerten Avatare hinarbeiten bzw. sich hinentwickeln (sollen). Dies kann je nach Spielgenre über sogenannte Charakterwerte geschehen, die es zu erhöhen und zu verbessern gilt, oder dadurch dass bestimmte vorgegebene Missionen erfüllt werden müssen. Ob ein Benutzer sich dabei als Einzelspieler durch eine Online-Spielwelt bewegt und andere Spieler als Feinde oder Freunde ansieht oder ob er sich mit anderen Benutzern zu ganzen Gruppen zusammenschließen kann, um gemeinsam das Spielziel zu erreichen, hängt jeweils von den Möglichkeiten des gewählten Online-Spiels selbst ab und trägt mit zum Facettenreichtum dieser Spiele bei.

Die meisten Online-Spiele sind erst als Nachfolger zu einem bestehenden Einzelplatzspiel entstanden. Nur durch den Erfolg eines vorhergehenden „Offline“-Spieles lohnt sich für die meisten Herstellerfirmen der Aufwand, die benötigte Infrastruktur für Online-Welten zur Verfügung zu stellen. Ein wichtiger Punkt, der die Online-Spiele dabei von ihren oftmals im Vorfeld bereits schon netzwerk- und somit mehrspielerfähigen Offline-Pendants unterscheidet, ist der Umstand, dass im Gegensatz zu den Offline-Spielen alle (spiel-)relevanten Eigenschaften des gesteuerten Avatars auf einem vom Hersteller bereitgestellten Spielserver gespeichert werden und erst beim Anmelden an diesen Server wieder abrufbar sind. Diese Server garantieren dafür eine (zumindest über bestimmte Zeitbereiche hinweg) persistente Spielwelt, in der Handlungen innerhalb dieser Spielwelt auch während der nicht teilnehmenden Zeiten des Avatars voranschreiten und die Welt sich dabei gegebenenfalls weiterentwickeln bzw. verändern kann. Im Vergleich hierzu wird bei normalen Spielen in der Regel beim Beenden der aktuelle Spielstand gesichert und bei einem späteren Weiterspielen an genau derselben Position mit genau demselben Zustand der zugehörigen Spielwelt wie vor dem Beenden wieder fortgesetzt. Für die Online-Spiele bedeutet dies andererseits aber auch, dass zum Spielen zwingend eine (Internet-)Verbindung zum jeweiligen Spielserver erforderlich ist, ohne die das zugehörige Online-Spiel nicht mehr gespielt werden kann. In Folge dessen lassen sich neue Konzepte für einen Spielablauf realisieren, beispielsweise kann so eine Manipulation der Spieldaten durch einzelne Benutzer, die sich einen Vorteil verschaffen wollen, unterbunden werden, da die Kontrolle über diese Daten und die Datenhaltung nicht mehr auf dem eigenen Spielrechner sondern auf dem Spielserver stattfindet. Es erlaubt den Firmen aber gleichzeitig, den Zugang zu

diesen Daten zu beschränken und Gebühren zu deren Nutzung zu verlangen. Mittlerweile ist dies sogar die gängige (Spiele-)Praxis, d.h. Online-Spiele finanzieren sich zumeist dauerhaft über monatliche Abonnementsgebühren, ohne die auf die „eigenen" Spieldaten nicht mehr zugegriffen werden kann; nur selten reichen Einmalzahlungen[4] aus oder sind gänzlich kostenfreie Versionen erhältlich.

Da hinter (Online-)Computerspielen ein enormes finanzielles Potential steht, sind die Spielehersteller sehr darauf bedacht, ihre Konkurrenz durch ständig bessere und realistischer werdende Systeme in Bezug auf ihre verwendete Grafik, der enthaltenen künstlichen Intelligenz für die Computergegner und der Spielhandlung voranzubringen. Der Spielesektor ist daher eine treibende Kraft für die virtuelle Präsenz, insbesondere was die Weiterentwicklung von Fotorealismus und autonomen Avataren anbelangt, und hat dadurch auch großen Einfluss auf die parallel entwickelten und eingesetzten Möglichkeiten von virtuellen Präsenzsystemen.

Bei den Online-Spielen steht allerdings der Präsenzgedanke nicht zentral im Vordergrund. Zwar wird von den Herstellern großen Wert auf eine immer realistischere Spielumgebung gelegt, was sich allein schon anhand der Grafikentwicklung bei einzelnen Spielen über verschiedene Versionen hinweg aufzeigt und somit der physischen Präsenz zuträglich ist, aber andere mögliche Präsenzaspekte, vor allem die soziale Präsenz, sind eher untergeordnet. Wichtig ist in Online-Spielen vor allem die Spielhandlung und das Vorankommen zum vorgegebenen Spielziel bzw. damit einhergehend die Verbesserung der Fähigkeiten des eigenen Avatars. Mögliche Kommunikationen zwischen Avataren sind daher oftmals thematisch auf Taktiken oder Tipps zur besseren Bewältigung der Spielaufgaben ausgerichtet, andere Gesprächsthemen ergeben sich zumeist nur spärlich oder gar nicht.

Online-Spiele lassen sich wie normale Computerspiele in verschiedene Genres einteilen (siehe [w14]); es seien im Folgenden deshalb nur exemplarisch einige aktuelle Beispieltitel genannt.

Online-Rollenspiele

Das am häufigsten mit dem Begriff Online-Spiele assoziierte Genre sind die sogenannten Mehrspieler-Online-Rollenspiele ('Massively Multiplayer Online Role-Playing Games', MMORPGs). Die Handlung spielt sich für gewöhnlich in eigenen Universen, wie z.B. Mittelalter-, Fantasie- oder Science-Fiction-Welten ab. Die Spieleravatare besitzen dabei jeweils bestimmte Attribute oder Charakterwerte (z.B. Stärke, Intelligenz, Beweglichkeit, ...), die Einfluss auf die Handlungsmöglichkeiten des Avatars in ihrer Spielwelt haben und die es zu pflegen und zu erhöhen gilt. Oftmals einhergehend mit den Charakterwerten gibt es auch erreichbare Stufen bzw. Level für die Avatare, welche den allgemeinen Entwicklungsstand innerhalb des Spieles beschreiben. Anhand dieser Levels wird eine Vergleichsmöglichkeit zum Entwicklungsstand anderer Avatare geschaffen. Es lassen sich so Ranglisten der Avatare untereinander erstellen und der damit geförderte Wettstreit soll die Spieler zum Weitermachen animieren. Um bei diesen Online-Spielen Erfolg zu haben und zu den Besten zu gehören, sind dann ausschließlich die vorgegebenen (auch teils fiktiven) Charakterwerte des jeweiligen Online-Spiels entscheidend; die menschlichen Charakterwerte des den Avatar steuernden Spielers sind hingegen nicht von Belang.

Da sich dieses Genre einer beständig wachsenden Spielergemeinde erfreut und eine erfolgreiche Online-Spielwelt einen beträchtlichen finanziellen Faktor für ein Unternehmen bedeuten kann,

4 Ein Beispiel hierfür ist das Online-Spiel 'GuildWars' von AreaNet, wo nur das Spiel einmalig bezahlt werden muss und keine laufenden Serverkosten anfallen. Dafür müssen aber, um auf dem aktuellen Stand bleiben zu können, die regelmäßig erscheinenden Spielerweiterungen erneut gekauft werden, wodurch sich letztlich ähnliche Ausgaben wie bei monatlichen Abonnementsgebühren ergeben.

steigt die Zahl der verfügbaren Online-Rollenspiele kontinuierlich stark an. Eine Übersicht über aktuelle MMORPGs liefert hierzu [w15]. Der heutzutage inzwischen wohl bekannteste Vertreter dabei ist 'World of Warcraft' von Blizzard Entertainment [w16], welches inzwischen in mehreren Sprachen vertrieben wird und weltweit über 10 Millionen registrierte Benutzer zählt[5].

Abbildung 7: Screenshots aus 'World of Warcraft': Auswahl eines Avatars und eine Abenteurergruppe im Spiel

Die Grafik des Spiels ist dabei in einem comichaften, zeichentrickartigen Stil gehalten, welcher aber teils erstaunlich detailliert ausgestaltet ist (vgl. Abbildung 7). Hierdurch sind die Anforderungen des Spiels an die zum Spielen benötigte Hardwareausstattung relativ moderat, was ebenfalls zu der großen Verbreitung von World of Warcraft beiträgt. Ein Spieler kann in World of Warcraft durch das Erledigen von Aufgaben – sogenannten 'Quests' – und das Töten von Monstern seinen Spielavatar verbessern und weitere Erfahrungslevel hinzugewinnen. Durch dieses jederzeit mögliche Voranschreiten des Spielcharakters und dem schnellen Erreichen von Teilzielen (eben diesen Quests) wird in World of Warcraft auch ein nicht zu unterschätzender Suchtfaktor aufgebaut, der von der Herstellerfirma (wie aber auch in anderen vergleichbaren Spielen von anderen Herstellern ebenfalls praktiziert) bewusst gefördert wird, um die Spieler zum Weiterspielen und somit an langfristigen Weiterzahlungen der Abonnementsgebühren zu animieren [w18].

Um den für die immense Anzahl an registrierten Benutzern benötigten Serveraufwand überhaupt bewältigen zu können, werden alle Benutzer auf verschiedene Einzelserver weltweit aufgeteilt, welche wiederum jeweils auf ca. 2500 Spieler beschränkt sind. Ein Spieler muss sich daher bei Spielbeginn einen freien Server aussuchen. Eine Kontaktaufnahme zu Spielern auf anderen Servern als dem eigenen ist nicht möglich. Es gibt lediglich die Möglichkeit, den eigenen Spielavatar komplett – und kostenpflichtig – auf einen anderen Server zu transferieren; ein Spieler kann dann aber wiederum nur mit den Personen auf dem neuen Server in Verbindung treten, der Kontakt zum alten Server ist damit abgebrochen.

World of Warcraft selbst kann, obwohl es ein MMORPG ist, zu großen Teilen wie ein Einzelbenutzerspiel gespielt werden, ohne Kontakt zu anderen Spielern aufnehmen zu müssen (eine Internetverbindung ist aber trotzdem dauerhaft erforderlich). Allerdings gibt es auch – vor allem im späteren Verlauf des Spieles – immer wieder Teilaufgaben, die derart angelegt sind, dass sie nicht mehr allein zu bewältigen sind. Auf diese Weise wird zumindest im Ansatz eine gewisse soziale Komponente zwangsweise in das Spiel integriert, da man sich für diese Gruppen-Quests mit

5 Stand Januar 2008, lt. Herstellerangaben, [w17].

anderen, auch fremden Avataren zusammenfinden und absprechen muss. Die Kommunikation findet dabei im Spiel nur über einen eingebauten Textchat statt, für den verschiedene Chat-Kanäle bereitstehen. Auch eine gewisse Grundauswahl an Gesten – sogenannte 'Emotes' –, wie lachen, tanzen, usw. lassen sich per Knopfdruck vom eigenen Avatar ausführen. Um gerade die zuvor genannten Gruppen-Quests einigermaßen koordiniert absolvieren zu können, sind diese Kommunikationsmöglichkeiten innerhalb des Spiels aber zu träge und nicht ausreichend. Daher gibt es einige Zusatzprogramme von anderen Herstellern, die sich diesem Manko gewidmet haben, beispielsweise die Sprachkonferenzsoftware 'TeamSpeak' von TeamSpeak Systems [w19] oder 'Ventrilo' von Flagship Industries [w20], welche als separate und eigenständige Programme einen zum Spielen parallelen Audiochat der Nutzer ermöglichen.

Andere MMORPGs wie 'EVE Online' von CCP Games [w21] versuchen ihre Position auf dem Online-Spielemarkt durch Berücksichtigung dieser Kritikpunkte zu stärken, indem sie bereits bessere Kommunikationsmöglichkeiten wie den Sprachchat innerhalb des Grundspieles von Beginn an integriert haben und auch keine explizite Serveraufteilung für die Benutzer vornehmen. Allerdings bewegen sich die Benutzerzahlen hierbei noch in bedeutend kleineren Dimensionen[6], so dass abzuwarten bleibt, ob dieses Vorgehen zukünftig bei steigenden Spielerzahlen beibehalten werden kann.

Ein weiteres Beispiel, wie in diesem Genre um Benutzer und somit um Kunden geworben wird, stellt 'Der Herr der Ringe Online' von Turbine.Inc [w22] dar. Hier wird ein hoher Stellenwert auf die eingesetzte Grafik gelegt, um dadurch einen deutlich höheren Grad an Realismus für die teilnehmenden Spieler zu erreichen (siehe Abbildung 8). Allerdings sind für einen flüssigen Spielablauf hierfür auch (noch) Hardwareanforderungen im gehobenen Bereich der heutigen Heimrechnerausstattung zu erfüllen. Vom Grundprinzip des Spiels her ist es World of Warcraft allerdings sehr ähnlich; die hauptsächlichen Unterschiede finden sich – neben der Grafik – in der zugehörigen Hintergrundgeschichte und einigen, sich daraus ergebenden und hierfür angepassten Spielelementen und Charakterwerten.

Abbildung 8: Aktuelle Spielgrafik von 'Der Herr der Ringe Online'

6 Der bisherige Rekord für Eve Online an gleichzeitigen Teilnehmern im gemeinsamen Spieluniversum liegt laut Herstellerangaben bei knapp 35.000 am 24.06.2007.

Online-Shooter

Ein weiteres Genre, welches häufig mit dem Begriff der Online-Spiele verknüpft wird, stellen Online-Shooter dar, die in den Medien durch ihre teils fragwürdigen Spielziele auch schon zu trauriger Berühmtheit gekommen sind. Bei Online-Shootern wird zumeist in zwei Teams gegeneinander gekämpft, wobei sich das Spielprinzip grob auf das Abschießen der gegnerischen Spieler reduzieren lässt. Wer am wenigsten oft selbst abgeschossen wird bzw. am meisten Gegner tötet führt die Ranglisten an.

Soziale Aspekte, wie sie für ein vollwertiges Präsenzsystem erforderlich sind, sind bei Online-Shootern nur schwach ausgeprägt. Erfahrene Spieler gehen nach erprobten Taktiken vor, ohne dass noch Absprachen mit anderen Spielern nötig sind; für abweichende Vorgehensweisen kann ein rudimentärer Textchat verwendet werden. Audiochat für eine direkte und bessere Kommunikation wird von den Spielen in der Regel nicht angeboten und muss wie bei den Online-Rollenspielen über zusätzliche Sprachkonferenzsysteme vorgenommen werden ([w19], [w20]).

Online-Shooter legen dafür umso mehr Wert auf eine möglichst realistische und detailgetreue Darstellung des Spielszenarios. Je nach Spiel beläuft sich die Umgebung dabei dann auf eine fiktive Stadt mit einem Häuserkampf oder es handelt sich um die Nachbildung real existierender Orte bzw. an historische Ereignisse angelehnte Szenarien wie z.B. der Zweite Weltkrieg, die tropische Umgebung Vietnams oder die Wüstengebiete der Golfregion.

Einer der bekanntesten Vertreter von Online-Shootern stellt 'Half-Life: Counter-Strike' von Valve [w23] dar, bei dem sich ein Spieler aussuchen kann, ob er für Terroristen Aufträge erledigt, welche vom Bomben legen bis zum Geiseln nehmen reichen, oder ob in einer Anti-Terror-Einheit gegen die Terroristen gekämpft wird.

Abbildung 9: „Spielkämpfe" im Online-Shooter 'Counter-Strike'

Die Bezeichnung „Online" für dieses Genre ist im Gegensatz zu anderen Online-Spielen nicht sonderlich ausgeprägt und erfüllt zumeist nur die Minimalanforderung, um diesen Begriff zu rechtfertigen – eine Verbindung über das Internet zu anderen Rechnern. Online-Shooter entsprechen daher prinzipiell lediglich einer anderen Variante eines Mehrspielermodus, wie es von den Einzelspielen bekannt ist. Es gibt bei den Online-Shootern auch keine oder nur kaum Daten, die vom Hersteller gespeichert werden müssen; einzig die Ranglisten werden typischerweise persistent fortgeführt. Von den Herstellern werden lediglich die Spielserver bereitgestellt, die es ermöglichen, sich jederzeit mit anderen Spielern weltweit messen zu können und die den Spielfluss steuern, d.h.

die Aktionen der Spieler auf allen teilnehmenden Rechnern synchronisieren. Für ein lokales Netz können aber auch eigene Server aufgestellt und eingerichtet werden, so dass eine Internetverbindung in diesem Fall zum Spielen nicht notwendig ist. Auch sind die Avatare von Online-Shootern nicht selbst gestalt- oder erweiterbar; nach Auswahl für welches Team man „spielen" möchte, werden die Figuren fest vergeben. Die einzige Unterscheidung der Spieler beläuft sich dann auf die Wahl der mitgeführten und verwendeten Waffen.

Insgesamt lassen sich Online-Shooter daher nur sehr eingeschränkt mit anderen Präsenzsystemen vergleichen oder in Relation dazu einstufen, da das Spielprinzip zu fest auf die eigentliche Spielhandlung fixiert ist und keinen Raum für weitergehende Aktionen zulässt. Dennoch besitzen sie eine große Beliebtheit im Bereich der Online-Spiele und werden deshalb auch hier mit aufgeführt; immerhin stellen sie nach den Online-Rollenspielen die zweitgrößte Gruppe bei den Online-Spielgenren dar.

Online-Adventures

Nicht ganz so bekannt und verbreitet wie die vorhergehenden beiden Genres sind Online-Adventures, wie beispielsweise 'Uru' von Cyan Worlds [w24]. Sie sind bis zu einem gewissen Grad mit den Online-Rollenspielen vergleichbar; im Gegensatz dazu haben die Avatare aber keine Charakterwerte, die gesteigert und erhöht werden müssen, sondern sie besitzen die gesamte Zeit über dieselben Fähigkeiten wie von Beginn an, welche sich hauptsächlich auf die Manipulation von Objekten in und mit der virtuellen Umgebung beziehen. Die Spielwelt von Online-Adventures ist für gewöhnlich – analog zu ihren „Offline"-Pendants – mit einer tiefgehenden Hintergrundgeschichte verwoben. Der Hauptaugenmerk von Online-Adventures richtet sich daher vornehmlich auf die Erkundung dieser virtuellen Spielwelten mit deren Rätseln und Geheimnissen und der Kommunikation mit anderen Teilnehmern hierüber. Auf diese Kommunikation wird in diesem Genre großen Wert gelegt, so dass ein komfortabler Textchat die Regel ist. Die Unterstützung eines Audiochats ist nicht durchgängig verbreitet, aber teilweise vorhanden.

Abbildung 10: Ein Avatar erkundet die fremdartige Welt von 'Uru'

Durch das gemeinsame Lösen von Rätseln oder Diskutieren über verschiedene Spielwege lässt sich schnell eine Unterhaltung mit anderen Avataren beginnen. Der soziale Aspekt ist bei diesem Genre dadurch besonders stark ausgeprägt, was sich auch teilweise über das eigentliche Spiel hinaus durch fortgeführte Unterhaltungen in Forenbeiträgen oder einer zwar kleineren aber dafür umso

engagierteren Fangemeinde dieser Spiele zeigt. Online-Adventures kommen daher den im nachfolgen Unterkapitel 2.3.2.2 beschriebenen Online-Welten noch am nächsten.

Für nahezu alle existierenden weiteren Spielgenres gibt es inzwischen Online-Varianten, vom klassischen Brettspiel bis hin zu Browser-Spielen. Da diese Online-Versionen allerdings prinzipiell nur den Originalspielen mit hinzugefügter bzw. verbesserter Netzwerk- und Mehrspielerfähigkeit entsprechen, aber keine eigenen Präsenzaspekte verfolgen, wird auf diese hier nicht weiter eingegangen.

2.3.2.2 Online-Welten

Online-Welten unterscheiden sich in einigen wesentlichen Punkten von Online-Spielen. Dies wird auch von den jeweiligen Herstellern bzw. Vertreibern der Online-Welten immer wieder stark betont, um sich vom anderen Genre abzugrenzen. Der größte Unterschied liegt darin, gleich von Beginn an einen eigenen, möglichst frei gestaltbaren Avatar zu kreieren, mit dem sich der Benutzer identifizieren kann – egal ob er dabei das reale Aussehen widerspiegelt oder nur die Verkörperung dessen darstellt, was der Benutzer gerne sein möchte. Nachdem der Avatar erstellt wurde, bleibt er in dieser Form normalerweise bestehen. Es gibt in Online-Welten keine Werte oder Stufen, die, im Gegensatz zu den Charakterwerten vieler Online-Spiele, geschult werden müssen, um bestimmte Tätigkeiten verrichten oder bestimmte Gebiete betreten zu können. Es entfällt somit der in Online-Spielen vorhandene Zwang des ständigen Weiterspielens, um konkurrenzfähig zu bleiben. Die zumeist einzige (aber oftmals wichtigste) Änderungsmöglichkeit am eigenen Avatar besteht in der äußeren Erscheinung, d.h. hauptsächlich in der Wahl und Gestaltung der Kleidung.

Dafür ist allen Online-Welten ein hoher Stellenwert der Kommunikation und der Kommunikationsmöglichkeiten zwischen den Avataren gemein, im Gegensatz zu Online-Spielen oftmals noch vor der Grafik bzw. dem dargestellten Realismus der virtuellen Welt. Dies ergibt sich schon aus dem Ursprung vieler Online-Welten; viele von ihnen haben sich als neue Formen aus ehemaligen Chat-Programmen gebildet, denen eine 3D-Welt der vorhergehenden Kommunikationsmöglichkeit erst nachgeschoben bzw. „übergestülpt" wurde.

Trotzdem wird die eigentliche Kommunikation innerhalb der Online-Welten zum Großteil über die klassischen Chat-Mechanismen vorgenommen: zumeist per Textchat, aber auch (teilweise erst nach kostenpflichtiger Freischaltung) via Sprach- und Audiochat. Die Einbeziehung weiterer Kommunikationsmöglichkeiten wie Videoaufnahmen oder Webkameras, um beispielsweise Videokonferenzen abhalten zu können, ist bislang in den bestehenden Online-Welten nicht gebräuchlich bzw. gar nicht erst möglich; Nutzer in Unternehmen oder Gruppen, die beispielsweise Konferenzen in Online-Welten abhalten möchten, sind dort rein auf ihre avatarbasierte Erscheinungsform beschränkt (vgl. auch Kapitel 2.4.3).

Dafür sind Gesten und Gesichtsausdrücke ein beliebtes und bei den Online-Welten weit verbreitetes Mittel, Gefühle und Emotionen erkennbar werden zu lassen und sind folglich zumeist vielfältig vorhanden. Aufgerufen werden sie explizit über spezielle Befehle oder Tastenkommandos oder, nicht ganz so häufig verbreitet, auch durch automatisches Aktivieren beim Erkennen bestimmter vorhandener Schlüsselworte im Chat-Text selbst.

Abbildung 11: Mögliche Gesten eines Avatars zum Ausdruck der empfundenen Gefühle

Ein wesentliches Merkmal, welches den Benutzern von Online-Welten zugesprochen wird, ist die Freiheit, selbst zu entscheiden, welche Ziele sie darin verfolgen möchten. Online-Welten legen mehr Wert auf die Bildung sozialer Strukturen zwischen den Benutzern, als auf die Bereitstellung von Inhalten. Die meisten Anbieter verstehen ihre Online-Welt daher lediglich als eine Plattform, die durch die Kreativität und Aktionen von Benutzern mit Leben gefüllt werden und dabei soziale Kontakte der Benutzer untereinander aufbauen und fördern soll.

Online-Welten sind daher verstärkt darauf ausgelegt, schnell andere Avatare mit ähnlichen Interessen finden zu können und mit ihnen in Kontakt zu treten. Umfangreiche Suchfunktionen hierfür gehören mittlerweile zum Standard. Es lassen sich häufig auch auf einfache und schnelle Art und Weise Gruppen mit den neu geschlossenen Kontakten anlegen und für zukünftige Gespräche sichern. Oftmals organisieren sich diese Gruppen dann auch weiterhin und halten regelmäßige Treffen ab, was einer Art Vereinsleben in Online-Welten schon sehr nahe kommt. Viele Teilnehmer nutzen so die Möglichkeit, soziale Kontakte auch weltweit über die besuchten Online-Welten zu knüpfen und zu pflegen.

Genau diese Freiheit, sich die eigenen Ziele zu setzen und zu verfolgen, ist es aber auch, die viele Benutzer einer Online-Welt bereits nach kurzer Zeit die Lust an der virtuellen Welt verlieren lässt. Da sie oftmals gerade zu Beginn nicht wissen, welche Möglichkeiten sie mit ihrem Avatar überhaupt haben, selbst aktiv etwas zu unternehmen oder etwas aufbauen zu können, langweilen sie sich schlicht. Viele Anbieter kommerzieller Online-Welten sind deshalb inzwischen dazu übergegangen, für ihre (zahlenden) Mitglieder organisierte Freizeit- und Unterhaltungsangebote in ihre Welten einzubauen, um gewisse Anreize für die Benutzer zu schaffen, dort länger präsent zu sein bzw. häufig wiederzukommen und damit auch insgesamt die Beliebtheit ihrer Welt zu steigern.

Eine weitere Gemeinsamkeit zwischen den verschiedenen Online-Welten besteht in der Art der Fortbewegung innerhalb der virtuellen Umgebung. Üblicherweise gehen die Avatare zu Fuß oder nutzen einen im Großteil der Online-Welten verfügbaren Flugmodus, mit dem sich der Avatar frei überall hin bewegen lässt. Für weite Strecken oder aber um schnell zu einem bestimmten Treffpunkt mit anderen Avataren oder Freunden zu gelangen, gehört Teleportation inzwischen ebenfalls zum Standard. Entweder geschieht dies über spezielle, vorhandene Teleport-Plattformen, welche an beliebten oder stark frequentierten Punkten in der virtuellen Welten zugänglich sind, oder – je nach Umsetzung des Anbieters – auch ganz frei per Tastendruck von jedem beliebigen Punkt aus. Somit gibt es kaum Beschränkungen, wo sich ein Avatar aufhalten bzw. wie schnell er reisen kann. Dadurch können schnell spontane Treffen mit Freunden realisiert werden, auch wenn man nur kurze Zeit in der Online-Welt unterwegs ist.

Andere Arten der Fortbewegung, beispielsweise mittels (virtueller) Motorräder, Autos oder Schiffe, existieren zwar in Online-Welten sehr zahlreich, stehen üblicherweise aber nicht von Beginn an zur Verfügung, oder falls doch, dann nur in sehr einfacher und rudimentärer Ausführung. Bessere (und damit eigentlich nur optisch ansprechendere) Fahrzeuge sind erst durch den Erwerb beim Hersteller der Online-Welt oder einem Avatar-Händler gegen virtuelle oder reale Bezahlung erhältlich und stellen somit – wie im echten Leben – Prestige- und Statussymbole für einen Avatar dar, mit denen er sich von den anderen Avataren abgrenzen kann.

Abbildung 12: Statussymbole in Online-Welten

Dieser Statuskult und der damit einhergehende Warenhandel gilt auch fast ausnahmslos für alle anderen virtuellen Güter innerhalb der Online-Welt, insbesondere für Kleidung. Besonders kreative Benutzer können mit ihrem und für ihren Avatar auch selbst neue Kleidung und Waren entwerfen und mit diesen dann in der Online-Welt Handel treiben. Dies führte bereits in einigen Online-Welten zur Ausprägung ganzer Wirtschaftsbereiche, was die Erzeugung und den Vertrieb von Waren, aber auch den Handel mit virtuellem Bauland anbelangt, einschließlich der zugehörigen ökonomischen Prozesse und den sich daraus ergebenden ersten virtuellen Online-Welt-Millionären [w25]. Insbesondere die Industrie hat diesen Online-Markt bereits für sich entdeckt und viele große Firmen wie z.B. Coca-Cola, Nike oder Adidas sind bereits in virtuellen Welten vertreten und bieten Teile ihrer Produktpalette als (virtuelle) Güter an. So kann ein Benutzer beispielsweise in der Online-Welt 'Second Life' sich gegen reales Geld eine virtuelle Markenjeans kaufen und mit dieser seinen Avatar ausstatten. Es laufen auch Versuche von Firmen, virtuelle Produkte, die in Online-Welten gekauft werden, auch im realen Leben zustellen zu lassen, so dass es prinzipiell gar nicht mehr nötig wäre, das eigene Haus (und somit – von manchen Firmen angedacht – auch die Online-Welt) zu verlassen. Ob dies das zukünftige Ziel von Online-Welten darstellen soll, bleibt dahingestellt.

Abbildung 13: Firmenauftritte in Online-Welten (hier: Adidas und T-Online in 'Second Life')

Die Finanzierung der Unterhaltskosten von Online-Welten für ihre Betreiber funktioniert dabei im Wesentlichen genauso wie bei Online-Spielen: entweder über monatliche Abonnementsgebühren – teilweise auch kombiniert mit einem kostenlosen Basiszugang, um Neueinsteiger anzulocken, und kostenpflichtigen Premiumdiensten[7].

Ähnlich wie bei den Online-Spielen gibt es heutzutage inzwischen eine Fülle sowohl kommerzieller als auch kostenloser, virtueller Online-Welten. Nachfolgend werden hier die bedeutendsten Vertreter ihrer Art vorgestellt; einen weitergehenden Überblick auch über kleinere oder spezialisierte Online-Welten verschafft Virtual Worlds Review [w26].

Zu einer der ersten Online-Welten und somit zu den Gründern von dynamischen, virtuellen 3D-Welten zählt das 1997 erstmals für die Öffentlichkeit freigegebene 'Active Worlds' von Activeworlds Inc. [w27]. Es war ursprünglich, ähnlich zu dem in Kapitel 2.2.2 beschriebenen 3B-Browser, als eine neue Art von Internetbrowser gedacht, in dem es zusätzlich virtuelle Büros für Firmen und Geschäftsleute geben sollte, wo sie ihre Kunden treffen und Verkaufsgespräche abhalten konnten. Schnell hat sich hieraus eine eigenständige Online-Welt entwickelt; lediglich die Namensgebung des benötigten Klientenprogramms deutet noch auf diese Wurzeln hin: der 'Active Worlds Browser'. In Active Worlds war es erstmals möglich, sich mit einem eigenen ausgewählten Avatar frei in der 'Alpha World' zu bewegen, der ersten damals verfügbaren Welt, welche noch heute – bei inzwischen mehreren hundert weiteren Welten – die größte der vorhandenen virtuellen Welten im Active Worlds-Universum darstellt.

Die Herstellerfirma Activeworlds Inc. hat auch schon früh die Möglichkeit angeboten, gegen Bezahlung die Infrastruktur für eine komplett eigene und bei Bedarf geschlossene Welt zur Verfügung zu stellen, um beispielsweise Firmen ihre eigene virtuelle Welt nur für ihre Mitarbeiter zu ermöglichen; sei es zum Zeitvertreib während der Arbeitspausen oder für die interne, weltweite Firmenkommunikation.

7 Die Definition von Premiumdiensten variiert dabei stark von Anbieter zu Anbieter: es kann von inhaltlichen Belangen, wie dem erst gegen Gebühr ermöglichten Handeln in der Online-Welt oder Pachten von (virtuellem) Bauland, bis hin zu technischen Aspekten, wie dem Freischalten von Audio- und Sprachchat reichen.

Abbildung 14: Impressionen von 'Active Worlds'

Ansonsten besitzt Active Worlds die zuvor beschriebenen Merkmale von Online-Welten. Den Chat gibt es darüber hinaus in verschiedenen Ausprägungen. Neben einem Privatchat zwischen zwei Avataren oder einem Gruppenchat gibt es auch (wie mittlerweile nahezu Standard in allen Online-Welten) einen allgemeinen Umgebungschat, der nur die Nachrichten der Personen in einem bestimmten Umkreis um den eigenen Avatar herum erscheinen lässt; bewegt sich dieser weiter, verstummen die Personen in der Umgebung (bei Audioübertragung) bzw. sind deren Texte nicht mehr lesbar. Durch bestimmte Aktionen („schreien", „flüstern", ...) lassen sich diese Umgebungsradien bei der Wahrnehmung auch beeinflussen.

Durch die lange Bestehenszeit von Active Worlds im Vergleich zu anderen, neueren Online-Welten, wirkt allerdings insbesondere die Grafik aus heutiger Sicht etwas klobig und simpel und dadurch nicht mehr ganz zeitgemäß. Auch das beständige, sichtbare Nachladen von Objekten während des Reisens durch die virtuelle Active Worlds-Umgebung wurde inzwischen von anderen Mitstreitern deutlich flüssiger gelöst.

Zu diesen weiteren großen Mitbewerbern[8] bei den Online-Welten zählen speziell die in einem relativ kurzen Zeitraum zueinander veröffentlichten Online-Welten 'Second Life' von Linden Labs (Juni 2003) [w28] und 'There' von Makena Technologies Inc. (Oktober 2003) [w29]. Die mittlerweile mit Abstand größte und wohl bekannteste Online-Welt ist 'Second Life' (oftmals auch nur kurz 'SL' genannt).

Second Life war zum Zeitpunkt seines Erscheinens grafisch mit am weitesten fortgeschritten (das kurz danach erschienene There mit eingeschlossen), was sicherlich einen wesentlichen Beitrag zu der inzwischen enormen Verbreitung beigetragen hat. Aktuell besitzt Second Life circa 9 Millionen registrierte Benutzer weltweit, von denen aber nur ca. 1,6 Millionen aktiv sind[9]. Aktiv bedeutet hierbei, dass diese Nutzer sich mindestens einmal innerhalb der letzten zwei Monate eingeloggt haben. Im Umkehrschluss bedeutet dies aber auch, dass es viele „passive" Benutzer von Second Life gibt, die es anscheinend nur kurz ausprobiert haben, aber nicht dauerhaft dabei geblieben sind.

8 Diese Aussage gilt zumindest für die westliche Welt. Im asiatischen Raum gibt es andere, hier eher unbekannte Online-Welten, die von der Größe und Bedeutung die hierzulande verbreiteten Online-Welten überbieten. Hierzu gehört u.a. das chinesische HiPiHi [w30].

9 Stand August 2007, lt. Herstellerangaben.

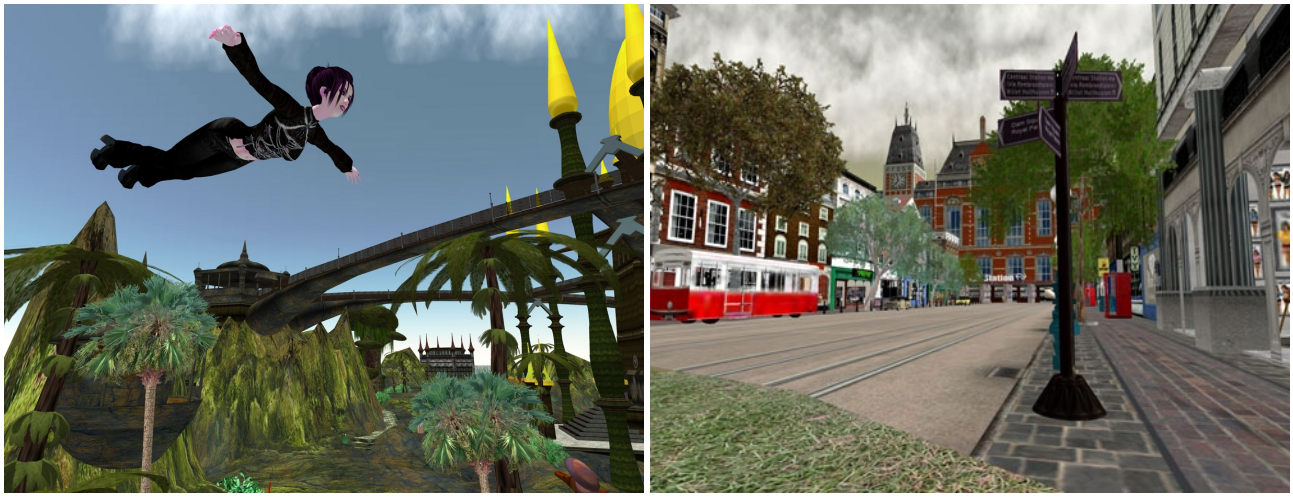

Abbildung 15: Szenarien aus 'Second Life'

Second Life entwickelte sich in den letzten Jahren durch die bereits angesprochenen Handelsmechanismen verstärkt zu einer vollwertigen Handelsplattform, in der wirtschaftliche Interessen dominieren. Insbesondere Firmen nutzen Second Life als neue Art einer Werbeplattform für ihre Produkte. Aber es suggeriert auch für den einzelnen Avatar den Traum vom (virtuellen) Tellerwäscher zum Millionär, indem es den Gedanken vermittelt, dass jeder durch Kreativität und Handel in Second Life reich werden kann. In einigen Fällen ist dies bereits gelungen [w25]. Betrachtet man jedoch die hierzu veröffentlichten Daten von Linden Labs, erkennt man, dass es sich aber nur um einige wenige Nutzer handelt, die tatsächlich davon leben können. Allerdings nimmt der Trend, dass immer mehr Personen in der Virtualität ihr Geld oder zumindest Nebeneinkünfte für das reale Leben verdienen, kontinuierlich zu [w31].

Als virtuelle Währung für den Handel innerhalb von Second Life dient der Linden-Dollar ('L$'), mit dem das komplette Wirtschaftssystem innerhalb Second Life aufgebaut wurde und der auch von und zu realem Geld umgetauscht werden kann. Den Umrechnungskurs bestimmt der LindeX, ein von Linden Labs dem realen Geldmarkt nachempfundenes, börsenähnliches System mit verschiedenen zugehörigen Geld-Tauschbörsen.

Second Life bietet ebenfalls die Möglichkeit, virtuelles (Bau-)Land zu erwerben, auf dem dann eigene Gebäude errichtet werden können, sowohl für Privatpersonen als auch für Gewerbetreibende. Etliche Firmen, Organisationen und auch Behörden weltweit haben inzwischen schon „virtuelle Adressen“, die als Anlaufstelle oder Verkaufsfläche dienen. Dabei werden die angebotenen Dienste immer vielfältiger. Seit Ende 2006 gibt es sogar eine eigene virtuelle Wochenzeitung innerhalb und für Second Life, angeboten vom Axel-Springer-Verlag, den 'AvaStar' [w32], mit Themen und Nachrichten aus Second Life oder Second Life betreffend.

Der Besitz von Landflächen wird benötigt, um darauf Handel mit eigenen Waren und den Besitz größerer, persistenter Objekte für einen Avatar erst zu ermöglichen und entsprechen somit einer Art Einheit für „Speicherplatz”. Nur auf dem eigenen Land können Gebäude errichtet werden; der Maximalgrad der Ausgestaltung des Hauses wiederum hängt von der Größe des Grundstückes ab. Denn jede Grundstückseinheit ('Sims' bzw. 'Plots' genannt) kann nur eine bestimmte Menge an Einzelobjekten ('Prims') aufnehmen. Erst größere Flächen erlauben daher filigranere (und somit schönere) Objekte mit mehr Details. Aber auch der Besitz bestimmter Gegenstände (vor allem von Statussymbolen) setzt eine gewisse Mindestgröße an eigenem Land voraus. So kann ein Avatar beispielsweise nur dann eine Yacht erwerben, wenn er entsprechend der Größe der Yacht auch eine

gewisse Landfläche besitzt, die wiederum der Prim-Anzahl der Yacht entspricht bzw. „aufnimmt“. Umgekehrt entspricht damit eine Yacht nur einem (sehr speziellen) „mobilen” Gebäudetyp.

Zum Handel mit anderen Avataren gehören aber auch nicht nur vom realen Leben her bekannte und ins Virtuelle übertragene Dinge wie Gegenstände, Kleidung oder Gebäude, sondern auch neue, rein virtuell existierende Güter wie Animationen. Ein Benutzer kann so für seinen Avatar selbst in einer vom Hersteller Linden Labs bereitgestellten Skriptsprache, der Linden Scripting Language (LSL), neue Bewegungen, Gesten oder ganze Animationsabläufe erstellen und dadurch seinen Avatar beispielsweise besonders individuell tanzen oder flirten lassen. Diese Animationen können dann ebenfalls als Handelsgut dienen, mit denen wiederum Geld zu verdienen ist.

Eine andere, weitere Besonderheit von Second Life – zumindest unter den hier erwähnten großen Online-Welten – besteht darin, als Avatar auch ungewöhnlichere Erscheinungsformen als die standardisierte männliche oder weibliche menschliche Gestalt zuzulassen. So ist es möglich, auch als Fantasiewesen oder in gänzlich unhumanoider Form die virtuelle Welt zu erkunden.

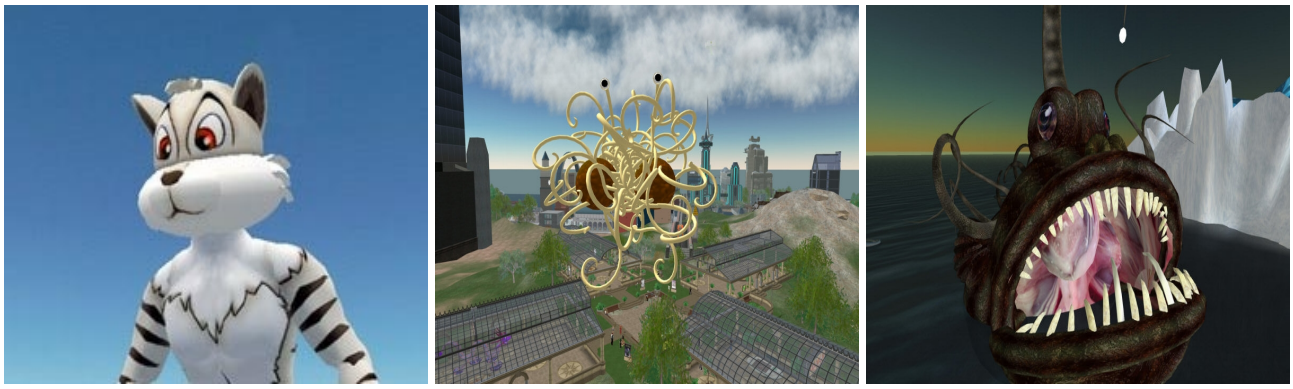

Abbildung 16: Mögliche nicht-menschliche Erscheinungsformen für Avatare in 'Second Life'

Technisch funktioniert Second Life – wie auch die anderen hier aufgeführten und vergleichbaren Online-Welten – nach dem von Online-Spielen her bekannten Client-Server-Prinzip. Ein Server bzw. eine Serverinstanz ist hierbei für die Verwaltung einer Region zuständig. Dies entspricht einer Fläche eines 256 x 256 m² großen Abschnittes auf der virtuellen Landkarte inklusive ihrem „Inhalt” (darauf befindliche Gegenstände bzw. Prims) und den momentan präsenten Avataren. Die Server sind dabei für ca. 70 Avatare gleichzeitig innerhalb einer Region ausgelegt; bei Überschreitung dieses Grenzwertes wird weiteren Avataren ein Betreten der überfüllten Region verwehrt.

Eine Ausnahme zu der Zielsetzung der bisherigen kommerziellen Vertreter von Online-Welten stellt die seit 2002 verfügbare Online-Welt 'Moove Online' der deutschen Firma moove [w33] dar. Anders als bei den zuvor angesprochenen Online-Welten geht es in Moove Online nicht um die ansonsten stark vorhandenen wirtschaftliche Interessen und Verdienstmöglichkeiten, sondern rein um soziale Aspekte und soziale Erfolge der Nutzer. Das Ziel von Moove Online ist somit, neue Freunde zu finden, mit denen man sich gut unterhalten kann, sich eventuell zu verlieben und vielleicht sogar (virtuell) zu heiraten. Um entstandene Beziehungen über die Virtualität hinaus entstehen zu lassen, bietet Moove Online die Möglichkeit, ein reales Bild des Nutzers in sein Avatarprofil mit einzubinden (allerdings nicht auf die Avatarfigur selbst). Moove Online hat daher bereits gewisse Grundtendenzen zu einer virtuellen Partnerbörse, wohingegen die Herstellerfirma selbst nur von einer 3D-Chat-Welt spricht.

Abbildung 17: Die 3D-Chat-Welt 'Moove Online'

Da keine wirtschaftlichen Ziele verfolgt werden, gibt es in Moove Online auch kein Geld, mit dem Handel betrieben werden könnte. Gewünschte Gegenstände zum Ausstatten seines Avatars bzw. persönlicher Räume können selbst erstellt werden oder bereits verfügbare Objekte können kostenlos „eingekauft" und aufgestellt werden. So will Moove Online es jedem Avatar ermöglichen, sich ein eigenes virtuelles Heim mit individuell ausgestatteten Räumen einzurichten. Zu diesem Zweck steht beliebig viel Platz zur Verfügung; explizites, virtuelles Bauland wird nicht benötigt. Für die eigenen Räumlichkeiten können nach den Vorstellungen des Erstellers Zugangsbeschränkungen für andere Avatare eingerichtet werden, um so die eigene Privatsphäre zu schützen. Andere Avatare können aber zu Besuchen eingeladen werden, bei denen dann die eigenen Räume vorgeführt werden können oder die zur Kontaktknüpfung dienen.

Eine gewisse Sonderstellung bei der Einordnung zu Online-Welten nimmt die Online-Welt 'Entropia Universe' von MindArk [w34] ein. Die Herstellerfirma MindArk versucht mit Entropia Universe eine Mischung zwischen einer Online-Welt mit den Elementen von Online-Spielen zu erreichen. So kann sich ein Avatar nicht nur in Entropia Universe frei bewegen und mit anderen Avataren kommunizieren, sondern es gibt auch zusätzlich Aufgaben im Spiel, über deren Erledigung Belohnungen verfügbar sind (zumeist Gegenstände), die sich dann wiederum für virtuelles Geld verkaufen lassen und so den Kontostand aufbessern. Davon lässt sich dann wiederum bessere Ausrüstung für den eigenen Avatar erwerben. Auch gibt es Kampfplätze, wo ähnlich zu Online-Shootern Avatare gegeneinander antreten können. Allerdings gibt es im Gegensatz zu reinen Online-Spielen keine klare Levelstruktur oder ein konkretes bzw. gemeinsames (Spiel-)Ziel für die Erfüllung all dieser Aufgaben bzw. eine Notwendigkeit dies zu tun. Dahingegen ist die Eigenschaft von Avataren, rollenspieltypische Charakterwerte zu besitzen, die verbessert werden können, wieder klar einem Online-Spiel zuzuschreiben. So kann ein Avatar in Entropia Universe u.a. einen (Handwerks-)Beruf erlernen und diesen trainieren, um die Erzeugnisse seines Berufs anschließend gewinnbringend – ähnlich den durch Aufgaben erhaltenen Gegenständen – verkaufen zu können. Im Gegensatz zu den meisten Online-Welten ist allerdings ein beliebiges Erstellen bzw. Modellieren von 3D-Objekten und Gegenständen in und für Entropia Universe nicht möglich.

Eine genaue Einteilung von Entropia Universe in eine Online-Welt oder in ein Online-Spiel fällt daher schwer. Die Hersteller sprechen zwar von einer Online-Welt, typische Spielelemente von

Online-Spielen sind aber ausgeprägt vorhanden. Hier lässt sich – ähnlich wie beim 3B-Webbrowser der Web-Präsenzsysteme (siehe Kapitel 2.2.2) – bereits ein fließender Übergang zwischen den verschiedenen Genres und Kategorien erkennen, was sich zukünftig weiter verstärken wird.

Ein abschließendes Beispiel einer Online-Welt, welches aus einem wissenschaftlichen Umfeld entstammt und mittlerweile auch als Open-Source-Projekt den Nutzern zur beliebigen Erweiterung zu Verfügung steht, ist 'Croquet' [w35].

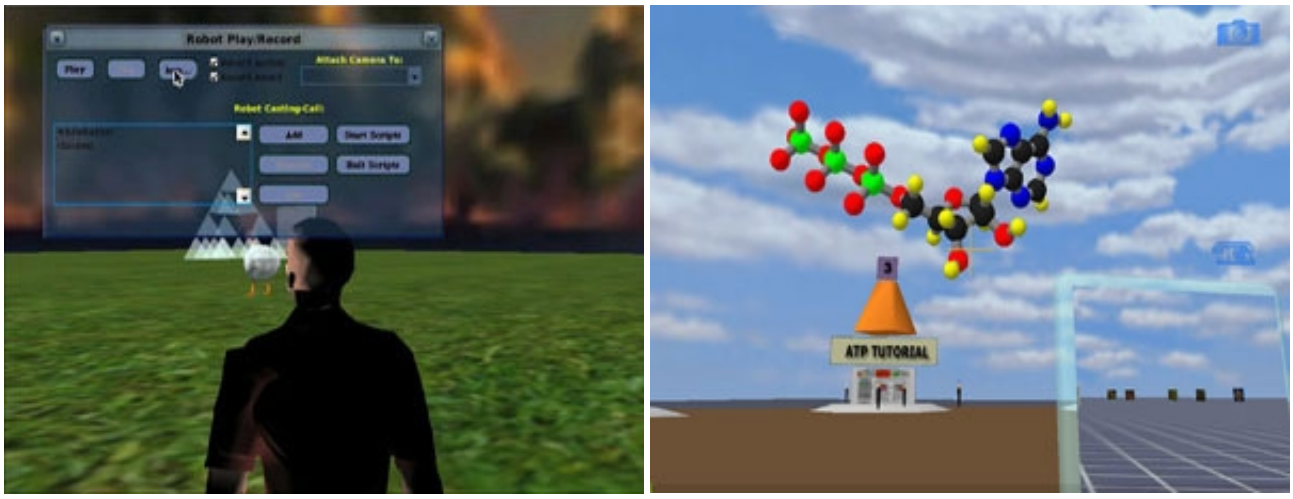

Abbildung 18: Die Open-Source-Online-Welt 'Croquet'

Das Ziel von Croquet ist es, eine Plattform bzw. Entwicklungsumgebung zum freien Aufbau und Gestalten eigener, über das Internet vernetzbarer und mehrbenutzerfähiger Welten – „Inseln" genannt – zur Verfügung zu stellen. Die eigenen, erstellten Inseln können anschließend mit bereits bestehenden Inseln über Portale „verlinkt" werden, so dass sich alle Croquet-Inseln untereinander besuchen lassen. Hierdurch ist eine nahezu beliebige Erweiterbarkeit eines von Croquet-Inseln erstellten Universums möglich und jeder Nutzer kann in diesem seine private und nach eigenen Vorstellungen entworfene Insel hinzufügen.

Die Ausrichtung von Croquet ist im Gegensatz zu den bisher besprochenen Online-Welten allerdings nicht primär auf die Knüpfung von Kontakten zwischen den Avataren untereinander ausgelegt, obwohl auch verschiedene Chatvarianten in das System integriert sind, sondern mehr auf die Visualisierung von vorhandenem Datenmaterial, wie der Integration von damit selbst erstellten Simulationen, der Darstellung von Datenreihen, aber auch der Einbindung eigener 2D- oder 3D-Anwendungen.

2.4 World of Wissenheim – Eine virtuelle Präsenzplattform

2.4.1 Motivation und Zielsetzung

Im Rahmen dieser Arbeit wurde ein Prototyp einer virtuellen Präsenzplattform namens 'World of Wissenheim' [w2] (im Folgenden weiter 'Wissenheim' genannt) entwickelt, mit welchem mehrere Forschungsaspekte zugleich angesprochen werden. Einerseits zeigt Wissenheim durch seine Umsetzung die Lauffähigkeit und die Einsatzmöglichkeiten eines neuartigen und alternativen technischen Konzeptes auf, welches der Realisierung zugrundeliegt. Andererseits stellt Wissenheim eine experimentelle Plattform mit neuartigen Möglichkeiten für weitergehende Formen der Kommunikation und der Inhaltsaufbereitung innerhalb eines virtuellen Präsenzsystems zur Verfügung. Diese kann dann wiederum als Basis für weitergehende Arbeiten und Forschungen in und zu diesem Bereich dienen.

Abbildung 19: Verschiedene Erscheinungsformen von Wissenheim

Eine Zielsetzung von Wissenheim ist, einem Benutzer die gewohnten Dienste und Merkmale zu bieten, wie sie von bereits bestehenden Präsenzplattformen – insbesondere von Online-Welten – her bekannt sind, aber darüber hinaus auch noch zusätzliche Erweiterungen zu enthalten, die neue und innovative Aspekte für ein Präsenzsystem mit sich bringen.

Im Folgenden werden die wichtigsten Eigenschaften und Merkmale von Wissenheim näher beschrieben. Messungen mit dem erstellten Wissenheim-Prototypen belegen zudem die Funktionsfähigkeit des neuartigen, verwendeten Ansatzes und dokumentieren das gewonnene Ergebnis (siehe Kapitel 5).

2.4.2 Umsetzungsaspekte

Betrachtet man die technische Umsetzung bei den zuvor in Kapitel 2.3 beschriebenen Arten von Online-Spielen und Online-Welten, so lässt sich feststellen, dass die vorherrschende Systemarchitektur, auf die heutige Präsenzsysteme aufbauen, für gewöhnlich eine Variante eines Client-Server-basierten Ansatzes darstellt. Bei Peer-to-Peer-basierten Ansätzen gehen in der Regel größere Einbußen in verschiedensten Bereichen einher, um diese Herangehensweise überhaupt lauffähig zu bekommen, beispielsweise in einer Abschwächung des Konsistenzmodells für die gemeinsamen Daten (siehe Kapitel 3.2.2). Infolgedessen sind hierauf aufsetzende Systeme kaum erforscht und auch nicht kommerziell weiterentwickelt worden. Dabei bieten solche Systeme, bei

Verwendung eines geeigneten Konzeptes, etliche Vorteile, welche in traditionellen Client-Server-basierten Systemen entweder gar nicht zur Verfügung stehen oder ansonsten aufwändig hinzuprogrammiert werden müssen (siehe Kapitel 3.2.3). Wissenheim stellt eine derartige Realisierung dar und macht sich die Vorzüge eines solchen Ansatzes in seiner Arbeitsweise zunutze.

Wissenheim entspricht als virtuelle Präsenzplattform im Kern einer verteilten Anwendung, die einen hohen immanenten Kommunikationsaufwand beinhaltet. Der auftretende, hohe Grad an Kommunikation zwischen den einzelnen teilnehmenden Stationen ist für den Datenaustausch und -abgleich erforderlich, um die von der verteilten Anwendung verwendeten Daten synchron und konsistent zu halten. Um dabei die Anwendung soweit wie möglich von dem damit verbundenen (Programmier-)Aufwand für diesen Abgleich zu entlasten, und damit einhergehende, potentielle Fehlerquellen von vorne herein bei der Entwicklung der verteilten Anwendung zu vermeiden, ist es vorteilhaft, wenn bereits von Betriebssystemseite her so viel wie möglich von dieser benötigten Funktionalität übernommen oder zumindest hierfür Unterstützung geboten wird. Ein verteiltes Betriebssystem als Grundlage für eine darauf aufbauende, verteilte Anwendung stellt daher eine ideale Ausgangsbasis für Wissenheim dar. Aus diesem Grunde wird hierzu das Clusterbetriebssystem Plurix (siehe Kapitel 1.5) eingesetzt; Wissenheim selbst wurde als verteilte Anwendung auf diesem System konzipiert.

Bei der Realisierung verwendet Wissenheim dabei als Ausgangspunkt kein Client-Server-Konzept, sondern setzt auf einem alternativen Ansatz auf, in welchem ein transaktionaler Szenengraph als Grundgerüst für die verteilte Anwendung dient. Dies bietet etliche Vorzüge im Vergleich zu traditionellen Konzepten; beispielsweise ermöglicht es die Verwendung einer damit verbundenen, dauerhaft starken Konsistenzform für die Datenhaltung zwischen den teilnehmenden Stationen, was bislang in dieser Art und Weise nicht ohne explizit programmierte und aufwändige Mechanismen für die Wahrung der Konsistenz zur Verfügung stand. Die Verteilung der Daten wird über einen gemeinsamen, verteilten Speicher vorgenommen, bei dem die Stationen gleichberechtigten Zugriff auf die geteilten Daten besitzen. Dies entspricht dadurch prinzipiell einem Peer-to-Peer-basierten Verteilungsansatz. Eine detaillierte Betrachtung des zugrundeliegenden, transaktionalen Szenengraphens und der sich daraus ergebenden Eigenschaften wird in Kapitel 3.3 und 3.4 vorgenommen.

Insgesamt implementiert Wissenheim durch diesen Ansatz ein effizientes und schlankes Präsenzsystem, welches durch seine Strukturen sowohl dem Anwender als auch dem Entwickler eine einfache und intuitive Arbeitsweise mit diesem System ermöglicht.

2.4.3 Merkmale und Besonderheiten

Wissenheim bietet die von bestehenden Präsenzsystemen bekannten Merkmale. Ein virtueller Avatar dient zur Repräsentation eines Benutzers in der virtuellen Welt, mit dem sich diese Welt weiter erkunden und über welchen darin interagiert werden kann. Ebenso lässt sich über den eigenen Avatar Kontakt mit anderen Avataren aufnehmen. Die Kommunikation kann dabei wie gewohnt mittels eines Chats ablaufen. Es stehen hierfür verschiedene Chat-Kanäle zur Auswahl bereit[10]; standardmäßig wird ein allgemeiner Chat verwendet, bei welchem alle Avatare in der Umgebung die diskutierten Inhalte mitverfolgen und sich beteiligen können. Es können aber auch private Gespräche innerhalb einer Gruppe oder zwischen zwei Avataren geschaltet werden.

10 Diese Chat-Kanäle sind im Plurix-spezifischen Namensdienst registriert. Durch Abrufen und Auswahl des gewünschten Kanals kann diesem automatisch beigetreten werden.

2.4.3.1 Videokonferenzen

Über den normalen Chat hinaus bietet Wissenheim auch die Möglichkeit, weitere Formen der Kommunikation direkt innerhalb der virtuellen Welt miteinzubeziehen bzw. bekannte, bislang aber separate Kommunikationsformen hierüber zu verbinden. So bietet Wissenheim bereits die Option, zusätzlich zum normalen (virtuellen) Erscheinungsbild des Avatars die Einblendung einer Framegrabber-Karte bzw. einer Webkamera innerhalb der virtuellen Welt zu aktivieren (siehe Abbildung 20). Dadurch ist es möglich, direkt in Wissenheim einen Videochat zu realisieren, ohne dass ein separates Viedeokonferenzprogramm notwendig ist. Man begibt sich einfach direkt in die Nähe eines anderen Avatars, und kann so dessen übertragenen Videostrom betrachten, wie auch den von weiteren Avataren in der Umgebung. Auf diese einfache und intuitive Art und Weise lassen sich die Mimik und Gestik der beteiligten Personen bei einer Diskussion mit in die virtuelle Welt integrieren, ohne dass über spezielle Tastaturkommandos wie in vergleichbaren Systemen Gestenanimationen für den eigenen Avatar ausgesucht und explizit angewählt werden müssen, um auftretende Emotionen ausdrücken zu können.

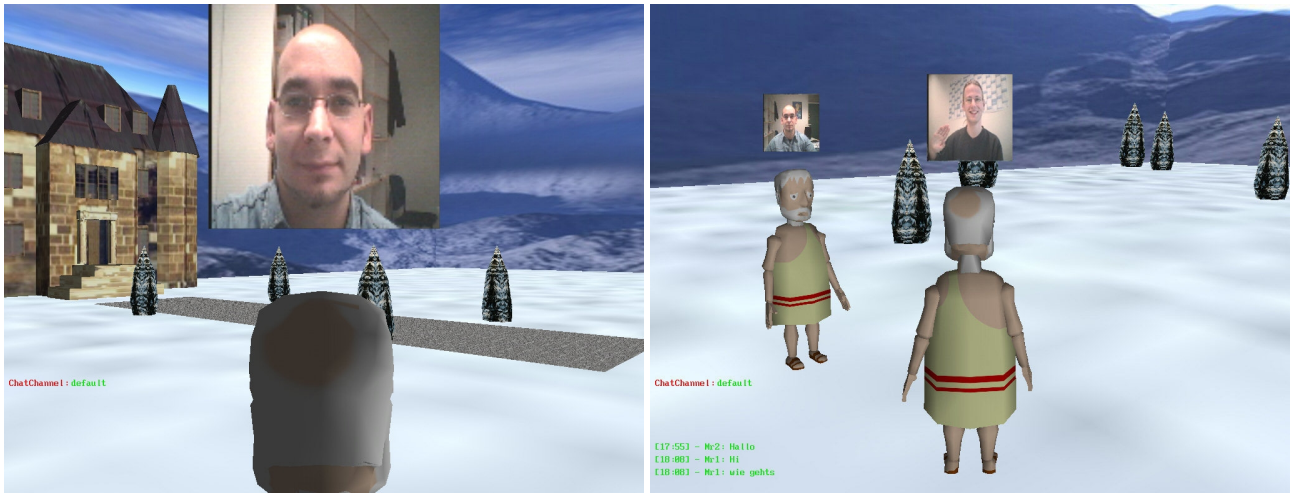

Abbildung 20: Videoeinblendungen ermöglichen Konferenzen direkt in Wissenheim

Um die Zuordnung der einzelnen Viedeoströme zu den beteiligten Personen zu erleichtern – vor allem wenn sich mehrere Avatare bzw. ganze Gruppen miteinander unterhalten –, werden die Videoeinblendungen direkt über den zugehörigen Avataren angezeigt und sind mit ihren Positionen gekoppelt. Insbesondere bei identisch gewählten Erscheinungsformen bei den verwendeten Avataren verhilft dies zu einem intuitiven Verständnis, welcher Benutzer nun welchen Avatar steuert oder wer beispielsweise bei einer Diskussion für eine bestimmte Frage angesprochenen werden muss. Dabei hat ein weiter entfernter Avatare dementsprechend ein kleineres (Video-)Bild als ein Avatar, der einem direkt gegenüber steht, wie es auch den normalen Gegebenheiten in der Realität entspricht.

Zusätzlich werden die übermittelten, zweidimensionalen Videoströme einer Webkamera zwar mit den Positionen der zugeordneten Avatare gekoppelt; um jedoch die 3D-Drehung innerhalb der virtuellen Welt immer zum eigenen Avatar hin ausgerichtet zu halten, werden die Rotationsdaten von der Position entkoppelt und mit der Ausrichtung des eigenen Avatars korreliert, anstatt diejenige des zugeordneten Avatars zu benutzen. Dies wird lokal für jeden Benutzer auf seiner eigenen Station vorgenommen und sorgt dafür, dass die übertragenen Videoströme immer vollständig sichtbar sind, egal ob sich der eigene Avatar um den anderen Avatar bewegt oder dieser sich von einem abwendet. Andernfalls würde dabei der Effekt auftreten, dass gerade eine solche

Bewegung bzw. Drehung des anderen Avatars das Videobild aus dem Sichtfeld des eigenen Avatars verschwinden lässt – vergleichbar einem Blatt Papier, das entlang der Hochachse gedreht wird –, obwohl man sich z.B. gerade mit diesem Avatar unterhält und sich dieser vielleicht nur kurz bewegt hat. Dieses Verfahren wird ansonsten bislang lediglich zum Ausrichten von Texten eingesetzt, beispielsweise um ein Erklärungsfenster, das einem feststehenden Objekt zugeordnet ist, unabhängig von der Position und Ausrichtung eines Avatars immer lesbar zu halten. Man spricht hierbei vom sogenannten 'Billboarding'. Die Anwendung eines daran angelehnten Verfahrens auf Videoströme wurde bislang von anderen Anwendungen noch nicht umgesetzt.

2.4.3.2 Kooperatives Arbeiten

Wissenheim eignet sich insbesondere auch für den Einsatz bei kooperativen Arbeiten mehrerer Avatare simultan an gemeinsamen Objekten ('Collaborative Work'). Gemeinsames Entwerfen, Erstellen und Editieren von 3D-Objekten (vergleichbar mit Mehrbenutzer-CAD-Systemen) oder Texten (sogenannte 'Whiteboard'-Funktionen) kann so leicht von mehreren Avataren miteinander vorgenommen werden. Durch den eingesetzten transaktionalen Szenengraphen sind simultane Manipulationen und Erweiterungen an bestehenden Objekten oder Texten möglich und werden zeitgleich für die anderen mitwirkenden Avatare sichtbar. Über die vorhandenen Chat-Mechanismen können diese dann auch gleich besprochen werden. Dies entspricht Konzepten, wie sie ansonsten hauptsächlich nur von speziellen und eigenständigen CAW-Anwendungen ('Computer Aided Work') her bekannt sind.

In Wissenheim stehen hierzu für die gebräuchlichsten Transformationsmanipulationen Helferobjekte zur Verfügung (siehe Abbildung 21), mittels denen durch einfaches Anklicken die gewünschte Manipulation am ausgewählten Objekt vorgenommen werden kann. Durch farbliche Kennzeichnung ist dabei für die anderen Avatare ersichtlich, welches Helferobjekt gerade von einem Avatar benutzt wird. So kann besser nachvollzogen werden, was gerade verändert wird, oder es wird möglich, durch Eingreifen dieser Veränderung entgegenzuwirken bzw. gleichzeitig mit an diesem Manipulator zu interagieren.

Abbildung 21: Zwei Avatare bearbeiten gemeinsam ein Objekt in Wissenheim

Eine umfassende Beschreibung der Arbeitsweise von Wissenheim im Zusammenhang mit der kooperativen Manipulation von 3D-Objekten findet sich in [18].

2.4.3.3 Interaktive Inhalte

Ein weiterer Anwendungsbereich von Wissenheim stellt der Einsatz als Plattform einer virtuellen Spaß- und Lernumgebung für Studenten dar, in dessen Rahmen Wissenheim auch kontinuierlich weiterentwickelt und ausgebaut wird.

Das Hauptaugenmerk liegt hier neben einer allgemeinen Bereitstellung einer interessanten, zur Erkundung einladenden virtuellen Welt und deren Objekten, auf der Einbeziehung von interaktiven Lerninhalten innerhalb von Wissenheim. Hierdurch soll die angebotene virtuelle Welt nicht nur ausschließlich der Freizeitgestaltung dienen, sondern es sollen auch fachspezifische Themen anschaulich vermittelt werden.

Bislang werden hierzu in Wissenheim verschiedene Themen aus dem Lehrstoff von (Informatik-)Studenten visuell umgesetzt und interaktiv zum Erkunden und Erforschen aufbereitet. Die umgesetzten Themen stellen dabei vorzugsweise stark abstrakte, theoretische oder generell schwierig zu vermittelnde Lerninhalte dar, welche auch über klassische Lehrformen und -medien wie Vorlesungen oder Bücher nicht immer in ausreichender Art und Weise veranschaulicht werden können. Durch die Einbeziehung in Wissenheim soll es ermöglicht werden, diese Themen „erlebbar“ und dadurch deutlich leichter verständlich zu machen.

Zu diesem Zweck werden für eine Einbindung in Wissenheim einzelne Themen oder Themenbereiche mittels dazu geeignet gewählter Metaphern als 3D-Objekte modelliert und innerhalb der virtuellen Welt platziert. So werden sie für einen Avatar buchstäblich „begreiflich“ gemacht. Um dabei auch in der jeweiligen Thematik enthaltene Abläufe, Algorithmen oder spezielle Funktionsweisen nachvollziehbar und verständlich präsentieren zu können, werden die modellierten 3D-Objekte nicht einfach als starre Skulpturen präsentiert, sondern ihnen werden zum jeweiligen Sachverhalt passende Animationen hinzugefügt. Mittels diesen Animationen können so direkt die enthaltenen Zusammenhänge visualisiert und vorgeführt werden.

Im Unterschied zu reinen Animationen bzw. animierten Objekten in anderen Präsenzsystemen (z.B. [w36]) werden innerhalb von Wissenheim diese Animationen allerdings zusätzlich interaktiv aufbereitet. Dies bedeutet, dass es einem Avatar bei modellierten Objekten möglich ist, die zu dem Objekt gehörende(n) Animation(en) nicht nur wie bislang üblich entsprechend einer Filmsequenz einfach nur zu starten und zu stoppen, sondern darüber hinaus Einfluss auf den eigentlichen Ablauf und die dadurch möglichen Verhaltensweisen des dargestellten Sachverhaltes nehmen zu können. So lassen sich resultierende Unterschiede bei Änderung der Vorbedingungen oder Variationen von Parametern von visualisierten Sachverhalten direkt im Ablauf erkennen.

Abbildung 22 zeigt die Bildschirmfotos zweier Beispiele in Wissenheim. Das linke Bild gehört zu einer Umsetzung des in der Betriebssystemtechnik bekannten „Philosophenproblems” [w37]. Ein Benutzer kann bei der interaktiven Variante dabei mittels seines Avatars selbst in die Rolle eines beliebigen, speisenden Philosophens schlüpfen und an dessen Stelle mit den zur Verfügung stehenden Besteckteilen essen oder auch diese für die anderen Philosophen blockieren.

Auf dem rechten Bild wird eine interaktive Animation zum sicheren Nachrichten- oder Schlüsselaustauschs nach dem Shamir-Verfahren aus dem Bereich der Kryptographie gezeigt [w38]. Hierbei können verschiedene Möglichkeiten zur Verschlüsselung ausprobiert werden und anschließend die Auswirkungen auf die Sicherheit bei der Übertragung beobachtet werden.

Abbildung 22: Interaktive Animationen in Wissenheim

Erreicht wird dies durch die Verwendung eines Entscheidungsbaumes, welcher jeder interaktiven Animation schon während der Erstellungsphase zugrundeliegt. In ihm sind die möglichen Einflussnahmen und die daraus resultierenden Ablaufsänderungen enthalten. Umgesetzt wird dieser Entscheidungsbaum dann mittels einer Mischung aus einzelnen, abspielbaren Animationen für feste, nichtteilbare Aktionen, und programmierter Logik zur weiteren Ablaufsteuerung.

Der Einsatz solcher interaktiver Animationen erlaubt es einem Benutzer – im Gegensatz zu reinen und dadurch nur passiven (Film-)Animationen – aktiv am präsentierten Geschehen teilnehmen zu können. Dies mindert die Tendenz von passiven Animationen, das Präsentierte nicht mehr weiter zu hinterfragen, sondern nur zu konsumieren. Es wird im Gegenteil vielmehr der Benutzer ermuntert, gerade die noch nicht bekannten Verhaltensweisen der interaktiven Animationen zu erkunden und auszuprobieren und dadurch über den gezeigten Sachverhalt hinaus Einsatzmöglichkeiten zu überlegen und zu erkennen, aber auch eventuelle Grenzen oder Einschränkungen dieser Sachverhalte zu verstehen.

Dabei ist die in Wissenheim mögliche Umsetzung von Themen nicht nur auf Lerninhalte von Studenten beschränkt, sondern diese interaktiven Animationen eignen sich insbesondere für alle Arten von Lehre: von einfachen Alltagsvorgängen angefangen, über Schulinhalte bis hin zu hochspezialisierten (Studien-)Themen. Um die durch die Virtualität ermöglichten Konzepte noch weiter auszuschöpfen, können diese interaktiven (Einzel-)Animationen auch miteinander verknüpft werden, so dass ganze virtuelle Welten zu einer großen Themeneinheit, vergleichbar einem „Erlebnispark“, für spezielle Themenbereiche entstehen können.

Bislang sind solche interaktiven Animationen hauptsächlich aus eigenständigen Computerspielen und expliziter Lernsoftware – zumeist mit nur Schülern als Zielgruppe (siehe [19]) – her bekannt. Durch die Einbeziehung in Wissenheim lassen sich auf diese Weise bestehende Grenzen zwischen reinen Spielen – die mittlerweile als Freizeitbeschäftigung einen festen Platz in der Gesellschaft gefunden haben – und explizit vorgeführten Lerninhalten – welche oftmals im Gegensatz zu den Spielen nicht freiwillig genutzt werden, sondern nur, wenn die behandelte Thematik beispielsweise prüfungsrelevant ist – weiter abbauen und die Benutzer werden so motiviert, sich auch ohne (Prüfungs-)Zwang neuen Themen (spielerisch) zuzuwenden bzw. interessante (Lehr-)Themen werden so einem breiteren Publikum zugänglich gemacht.

2.5 Zusammenfassung

Die virtuelle Präsenz ist ein junges Forschungsgebiet, in dem sich die zugehörige Terminologie und Inhalte noch nicht gänzlich etabliert haben. Sie zählt dabei nach einer Kategorisierung von Riva, Davide und IJsselsteijn zur Ko-Präsenz. Die eng zu ihr verwandte Web-Präsenz, welche als Vorstufe zur virtuellen Präsenz betrachtet werden kann, ist dahingegen der sozialen Präsenz zuzuordnen und deckt durch ihre etwas unterschiedliche Ausrichtung auch andere, von der virtuellen Präsenz nicht umfasste Bereiche ab, wie beispielsweise Präsenzsysteme ohne zugehörige virtuelle Umgebungen.

Innerhalb der virtuellen Präsenz haben sich zwei weitere, in ihrer Zielsetzung unterschiedliche Gebiete herausgebildet – die Online-Spiele und die Online-Welten –, wobei die Grenzen zwischen ihnen teilweise fließend sind. Sie alle verwenden Avatare zur Repräsentation eines Benutzers innerhalb der zugehörigen virtuellen Welt und bieten verschieden stark ausgeprägte Möglichkeiten zur Kommunikation zwischen den Avataren bzw. den sie steuernden Benutzern an. Online-Spiele besitzen dabei Charakterwerte für einen Avatar – allerdings je nach Spiel-Genre unterschiedlich stark ausgeprägt –, die sich erhöhen und verbessern lassen und dadurch die Fähigkeiten des Avatars mit der Zeit erweitern. Über Spielstufen bzw. Levels können die verschiedenen Avatare untereinander vergleichbar gehalten werden (zumindest innerhalb des jeweiligen Spiels), und es lassen sich Ranglisten bilden, die zur Förderung der Konkurrenz und des Wettbewerbs zwischen den Spielern dienen. Allen Online-Spielen ist dabei eine jeweils vorgegebene Handlung und ein festes Spielziel gemein, welches teilweise nur in Zusammenarbeit mit anderen Benutzern erreicht werden kann. Nach der Erfüllung des Spielziels nimmt allerdings der Reiz solcher Spiele oftmals rapide ab, weshalb die Spielehersteller bemüht sind, beständig neue Erweiterungen oder Wettkämpfe anzubieten, um die Spieler weiter und langfristig an sich zu binden.

Die Online-Welten bieten dahingegen in der Regel einen von seinen Fähigkeiten bereits von Beginn an vollständig ausgeprägten Avatar, der sich somit auch nicht verbessern kann bzw. muss. Dafür sind die individuellen Ausstattungsmöglichkeiten zur Abgrenzung von anderen Avataren in Online-Welten deutlich stärker ausgeprägt – nicht zuletzt auch durch die Möglichkeit, diese als Statussymbole zu verwenden. Online-Welten zeichnen sich darüber hinaus durch eine (nahezu) völlig freie Gestaltungsmöglichkeit der ausführbaren Aktionen innerhalb der virtuellen Welt aus, durch die ein Benutzer selbst entscheiden kann, was er tun und welche Ziele er verfolgen möchte und über die im Ganzen die Online-Welt von den teilnehmenden Avataren auch mitgestaltet und beeinflusst werden kann. Ebenso besitzt der Kommunikationsaspekt der Avatare untereinander einen deutlich höheren Stellenwert als bei Online-Spielen.

Anhand einer Auswahl an bestehenden Arbeiten zu sowohl der Web-Präsenz als auch der virtuellen Präsenz wurden die beschriebenen Konzepte aus den jeweiligen Bereichen auch in ihren Umsetzungen aufgezeigt. Ebenfalls wurde der im Rahmen dieser Arbeit entwickelte Prototyp eines eigenen virtuellen Präsenzsystems namens 'World of Wissenheim' vorgestellt. Dieser basiert auf einem alternativen Grundkonzept für Präsenzsysteme – einem transaktionalen Szenengraphen –, welches im nachfolgenden Kapitel genauer betrachtet wird. Wissenheim bietet zahlreiche Merkmale, wie sie aus bestehenden virtuellen Präsenzsystemen oder auch teilweise nur von eigenständigen Programmen her bekannt sind. So enthält es u.a. einen integrierten Video-Chat, über den sich Videokonferenzen in der virtuellen Welt realisieren lassen. Ebenso ermöglicht es kooperatives Arbeiten der Benutzer an gemeinsamen Objekten. Ein wesentlicher Punkt innerhalb von Wissenheim bilden zusätzlich interaktive Inhalte, mit welchen sich für einen Benutzer komplexe Themen und Sachverhalte aktiv und verständlich präsentieren lassen, so dass sich Wissenheim auch als Lernumgebung für verschiedenste Bereiche einsetzen lässt.

3. Grundstrukturen interaktiver virtueller Umgebungen

3.1 Szenengraphenkonzepte

Um die eingebundenen Objekte einer virtuellen Welt bzw. den grafischen Aufbau eines virtuellen Präsenzsystems auf einem Rechner darstellen zu können, werden grundlegende Strukturen benötigt, welche die vorhandenen Daten für die jeweilige Anwendung geeignet vorhalten und nutzbar machen. Bei grafischen, virtuellen Umgebungen spricht man hierbei von sogenannten Szenengraphen. Sie entsprechen einer Abbildungsvorschrift welche beschreibt, wie eine darzustellende Szene gegliedert und aufgebaut ist. Ein Szenengraph stellt somit das „Rückgrat" von virtuellen 3D-Welten dar. Der interne Aufbau des Szenengraphens wird dabei in der Regel hierarchisch organisiert.

Für gewöhnlich sind Szenengraphen mit einer auf den Szenengraphen abgestimmten Darstellungseinheit, der 'Renderengine', gekoppelt, die die vorgehaltenen Daten interpretieren und in geeigneter Form an die Grafikkarte zur Visualisierung weiterleiten kann. Man spricht dann von einem Szenengraphensystem. Das Vorhandensein einer solchen Renderengine ist aber für die eigentliche Funktion des Szenengraphens nicht zwingend notwendig; es erleichtert lediglich dessen Einsatz, da keine zusätzliche bzw. separate Anwendung benötigt wird, welche die Darstellung übernimmt.

Der folgende Überblick verdeutlicht den Werdegang und Einsatz von Szenengraphen.

3.1.1 Entstehung von Szenengraphen

Um dreidimensionale Echtzeitgrafik auf Rechnern darstellen zu können, war in den Anfängen der Computergrafik (1970er – 1980er Jahre) lange Zeit Spezialhardware für die eigentliche Bildgenerierung erforderlich. Als gebräuchlicher Mechanismus wurden hierbei die Modelldaten einer darzustellenden 3D-Szene in – vom jeweiligen Hersteller proprietäre – Datenformate exportiert, die direkt an die dazugehörige Spezialhardware gekoppelt und strukturell auch eng damit verwoben waren.

Ein Grafiker oder Modellierer generierte dazu die benötigten Datensätze aus dem von ihm konstruierten Modell und transferierte diese anschließend in die Grafikhardware. Veränderungen an den Modellen (zu Beginn sogar nur an dem einzig möglichen, darzustellenden Objekt) mussten direkt an den originalen Modelldaten selbst vorgenommen werden; der darstellende Rechner bzw. die Grafikhardware besaß nur sehr begrenzte Manipulationsmöglichkeiten, welche sich zumeist auf Änderungen an den Grundtransformationen wie Verschiebung, Rotation und Skalierung des Objektes innerhalb einer einfachen Umgebung beschränkten.

Die Firma SGI führte als eine der ersten flexiblere Grafikhardware ein[11]. Mit der damit verbundenen Zunahme der Leistungsfähigkeit bei den damals verwendeten Grafikkarten wurden auch die gebotenen Rendermöglichkeiten immer ausgereifter und flexibler. Zusammen mit dem technischen Voranschreiten wurden so Grafiksprachen entwickelt, um diese neu gewonnene Flexibilität auch für die Systemanwender nutzbar machen zu können. Dies führte mit zum Entstehen des Grafikschnittstellenstandards OpenGL, welcher durch ein Konsortium von Hard- und Softwareherstellern definiert wird.

11 Bekannt wurde SGI mit ihrer damaligen IRIS-Reihe für Hochleistungsgrafik-Terminals, beginnend mit der IRIS 1000-Serie von 1983.

OpenGL ([w39], [w40]) setzt direkt auf den Treibern der Grafikhardware auf und fungiert dadurch als einheitliche und hardwareunabhängige Schnittstelle hin zu höheren Anwendungsebenen. Es beinhaltet zusätzlich eine softwareseitige Emulation von standardisierten, aber von der Hardware nicht unterstützen Grafikfunktionen (abgesehen von heutigen Shader-Funktionen). Letztlich werden aber alle vorhandenen Grafikdaten an die Grafikkarte weitergeleitet und dort auf ihre Sichtbarkeit geprüft. Da dies ebenso auf nur teilweise oder gänzlich nicht sichtbare Objekte zutrifft, werden bis zur endgültig dargestellten Szene auch im Nachhinein unnötige Berechnungen durchgeführt, d.h. es werden u.a. nicht benötigte Texturen geladen oder nicht sichtbare Eckpunkte berechnet. Diese Berechnungen kosten wertvolle Rechenzeit und mindern dadurch die erreichbare (Grafik-)Leistung. Um diese weiter zu erhöhen, versucht daher bereits OpenGL in (Software-)Tests noch vor Übermittlung der Grafikdaten an die Grafikkarte, einen Teil der Dreiecke und Flächen, welche sich außerhalb des Sichtbarkeitsbereiches (siehe Abbildung 23) befinden, von der Darstellung und somit von der Übertragung herauszunehmen. Dies geschieht durch sogenannte 'Clipping'-Tests. Mit diesen wird geprüft, ob sich ein darzustellendes Dreieck innerhalb eines aufgespannten Sichtbarkeitsvolumens befindet und somit weiterverarbeitet werden muss oder ob es bei der endgültigen Darstellung nicht sichtbar ist und es an dieser Stelle bereits verworfen und zur nächsten Dreiecksberechnung übergegangen werden kann. Allerdings arbeitet OpenGL sehr hardwarenah, weshalb anhand der Tests auf dieser Ebene nur für einzelne Dreiecke, aus denen ein darzustellendes Objekt oder eine ganze Szene aus zu Tausenden aufgebaut ist, entschieden werden kann, ob sie weiterverwendet werden müssen oder nicht.

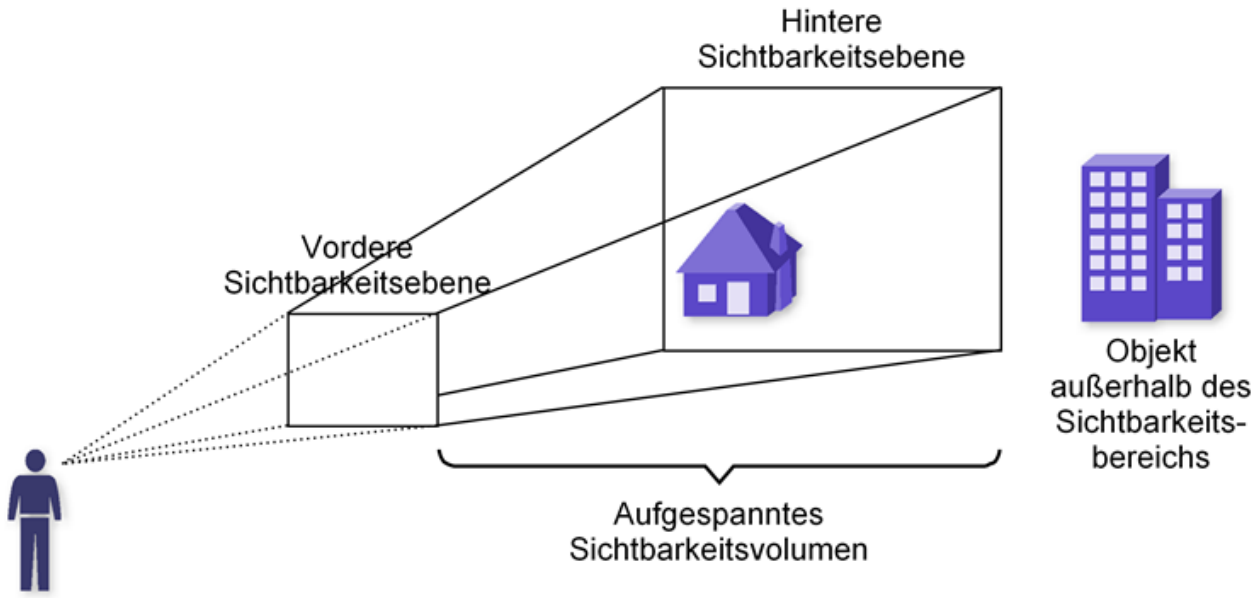

Abbildung 23: Sichtbarkeitsvolumen für Clipping-Tests

Bedeutend effektiver ist es, wenn bereits für komplette Objekte auf einer höheren Abstraktionsschicht entschieden werden kann, ob diese dargestellt werden müssen oder nicht, anstelle der Entscheidung bei jedem einzelnen Dreieck. Für diese Fälle lässt sich dann für ganze Objekte die Berechnung in der Grafikhardware vermeiden bzw. zusätzlich sogar die Weiterleitung der Dreiecksdaten an die OpenGL-Schicht selbst übergehen. Aus diesen Überlegungen und Randbedingungen heraus haben sich die ersten Szenengraphen entwickelt [w41].

3.1.2 Grundlegender Aufbau und Einsatz von Szenengraphen

Szenengraphen entsprechen einer Abbildungsvorschrift mit welcher beschrieben wird, wie (hauptsächlich graphische) Objekte und Elemente in einer virtuellen Umgebung zueinander in Verbindung stehen und spezifizieren dadurch insgesamt, wie daraus eine 3D-Szene aufgebaut ist. Sie arbeiten – im Gegensatz zur oben angesprochenen OpenGL-Schnittstelle – nicht auf der Ebene einzelner Dreiecksdaten, sondern auf der höheren Ebene von kompletten Objekten oder zumindest auf nicht weiter unterteilbaren Einzelobjekten, aus denen wiederum ein komplexes Objekt zusammengesetzt sein kann (dies hängt von der Art und Weise ab, wie ein Modellierer ein Gesamtobjekt ursprünglich erstellt hat).

Für den traditionellen Einsatz eines Szenengraphens wird eine hierarchisch aufgebaute Struktur verwendet, um eine intuitive und verständliche Zusammensetzung der darzustellenden Szene zu beschreiben. Diese basiert dabei normalerweise auf der räumlichen Anordnung der darin enthaltenen Objekte, ist aber nicht zwingend auf diese Einteilung festgelegt.

Ein Szenengraph ist als ein gerichteter, azyklischer Graph ('Directed Acyclic Graph', DAG) aufgebaut und lässt sich daher mittels einer Baumstrukur umsetzen und visualisieren. Die eigentlichen Objektdaten liegen dabei innerhalb der Knoten des Graphen. Die Granularität der Einzelobjekte und somit deren Anzahl kann dabei vom Modellierer individuell festgelegt werden und die erstellten (Einzel-)Objekte werden dann in den Szenengraphen eingefügt. Der Szenengraph selbst nimmt keine weitere Unterteilung dieser Daten vor (spezielle Optimierungen seinen hiervon ausgenommen). Als Beispiel kann ein Auto mit vier Rädern als ein einzelnes Objekt in einem Knoten dargestellt werden (siehe Abbildung 24). Die Räder sind in diesem Fall allerdings fest mit dem Auto verbunden und können sich nicht einzeln drehen. Bewegt sich das Auto, so werden auch alle vier Räder starr mitbewegt, da es sich bei diesem Aufbau des Autos gewissermaßen um ein Objekt aus „einem Guss“ handelt.

Abbildung 24: Ein Auto als Einzelobjekt

Sollen sich dahingegen die Reifen auch unabhängig von der Karosserie drehen und bewegen lassen, müssen diese durch den Modellierer von der Karosserie separiert und als einzelne Objekte erstellt werden. Sie werden anschließend als zusätzliche Knoten in den Szenengraphen eingefügt (siehe Abbildung 25).

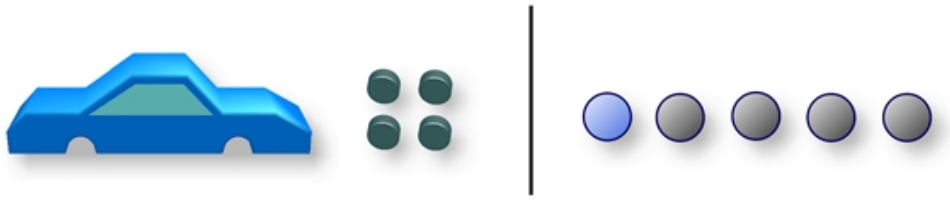

Abbildung 25: Ein Auto aus gleichgestellten Einzelobjekten

Werden die Räder auf der gleichen Ebene wie das Auto in den Szenengraphen eingetragen, so werden bei einer Bewegung des Autos die vier Reifen allerdings nicht automatisch mit bewegt, wie es vermutlich gewünscht wäre. Die Bewegung der vier Räder muss im Gegenteil separat zum Auto für jedes Rad einzeln berechnet und dann auf diese übertragen werden. Zusätzlich müssen die jeweiligen Veränderungen noch synchronisiert werden, um eine einheitliche Bewegung des Autos zu gewährleisten. Besonders bei komplexeren Vorgängen, z.B. einer Kurvenfahrt des Autos, sind aufwändige Berechnungen nötig, um eine korrekte Positionierung der einzelnen Reifen zu gewährleisten. Allerdings bietet dieses Vorgehen den Vorteil, dass jedes Einzelobjekt komplett eigenständig ist und somit beliebig und ohne weitere Restriktion durch die anderen Objekte manipuliert werden kann.

In vielen Fällen ist diese vollkommene Unabhängigkeit der Objekte bzw. Szenengraphenknoten zueinander aber gar nicht nötig bzw. gewünscht und entspricht auch nicht immer einem intuitiven Verständnis für die Arbeit mit zusammengehörenden Objekten. Deshalb wird üblicherweise eine Verbindung der zugehörigen Szenengraphenknoten vorgenommen, bei der die Objekte über unterschiedliche Hierarchiestufen miteinander verknüpft sind. Die untergeordneten sogenannten Kindknoten beziehen sich dann in ihrer Ausrichtung und Positionierung relativ zu ihrem Vaterknoten und nicht mehr auf absolute Werte innerhalb der virtuellen Welt, in der sie sich befinden. Im Beispiel werden so die einzelnen Reifen im Szenengraphen unterhalb der Hierarchiestufe der zugehörigen Autokarosserie eingeordnet (siehe Abbildung 26).

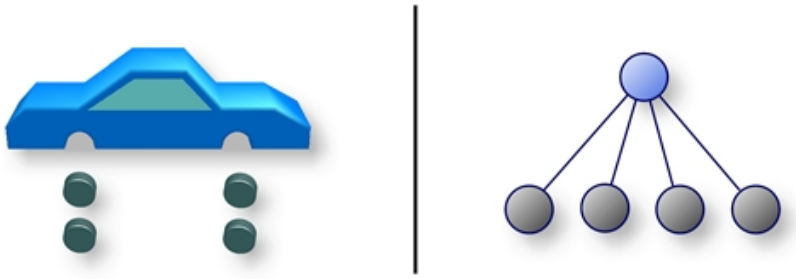

Abbildung 26: Eine hierarchische Aufteilung der Autoobjekte

Dies bewirkt dadurch, dass sich eine Bewegung des Autos automatisch und ohne eine zusätzlich notwendige Synchronisierung direkt auch auf die Reifen mit überträgt. Bei Bedarf können die Reifen aber weiterhin einzeln manipuliert werden, um so für eine realistischere Autofahrt die Drehbewegung der Räder mit zu simulieren, ohne dass die Positionsänderung für alle Räder nochmals zusätzlich berechnet und auf diese übertragen werden muss.

Szenengraphen sind in den meisten Fällen nach diesem Prinzip aufgebaut. Man spricht hierbei von einem Transformationsgraphen, da der Szenengraph primär die Funktion übernimmt, die vererbbaren Eltern-Kind-Beziehungen zwischen den Knotenhierarchien auszudrücken, bei denen die übergeordneten Knoten ihre Eigenschaften auch auf alle ihre Kindknoten übertragen.

Ein Szenengraph kann darüber hinaus aber auch für andere Zwecke benutzt werden. Hauptsächlich ist dies bei verschiedenen Arten von Optimierungen der Fall, welche eine schnellere Verarbeitung je nach benötigtem Verwendungszweck bewirken. Für die Grafikkartenhardware ist es beispielsweise wichtig, Sortierungen nach Gemeinsamkeiten bei der Darstellung vornehmen zu können (gleiche Texturen bei mehreren Objekten, eine Sortierung nach unbeleuchteten und beleuchteten Objekten, ...), um eine Minimierung der in der Grafikhardware hierzu benötigten und teils aufwändigen Zustandsänderungen zu erwirken. Vor allem OpenGL kann von solch einer

zustandsorientierten Sortierung profitieren, da häufige Zustandsänderungen allein schon durch die Übertragung dieser Änderungen an die Grafikhardware schnell sehr zeitintensiv werden können und sich somit negativ auf die Leistungsfähigkeit bei der Grafikdarstellung auswirkt. Eine Reduzierung dieser Übertragungen wirkt sich somit insgesamt günstig auf die Gesamtleistung aus. Auch lassen sich einige Spezialeffekte ohne eine vorherige Sortierung nach bestimmten Kriterien nicht realisieren[12].

Die in einer Szene vorkommenden Objekte lassen sich dabei in zwei grundsätzliche Kategorien unterteilen, welche wiederum hauptsächlich für weitere Verarbeitungs- bzw. Darstellungsoptimierungen herangezogen werden. So wird zwischen statischen und dynamischen Objekten unterschieden. Statische Objekte entsprechen in der Regel unbeweglichen und nicht manipulierbaren Objekten einer Szene, beispielsweise feststehende Gebäude oder Bäume in einer Landschaft. Dynamische Objekte entsprechen allen übrigen Fällen. Der Vorteil dieser Aufteilung liegt darin, dass bei statischen Objekten alle Eigenschaften dieser Objekte bereits zu Beginn an fest stehen und somit nur einmalig im Vorfeld des Szenenaufbaus – etwa zur Übersetzungszeit oder während des Imports der Objektdaten in einen Szenengraphen – vollständig berechnet und gegebenenfalls optimiert werden können, da sich diese Daten anschließend nicht mehr ändern. Die Bearbeitung zur Laufzeit entfällt dann für diese Objekte komplett, was Rechenkapazitäten für andere Aufgaben frei hält. So können mehrere statische Objekte im günstigsten Fall sogar mit nur einmaligem Aufwand in ein einzelnes statisches Objekt umgewandelt werden, bei dem alle enthaltenen Kindknoten direkt in die Dreiecksdaten des Vaterknotens einberechnet werden, wodurch sich später der komplette Traversierungs- und Berechnungsaufwand zwischen den ursprünglichen Hierarchiestufen vermeiden lässt. Dies gilt ebenfalls für benötigte Sortierungen – seien es beispielsweise die Zustandsänderungen für OpenGL-Anweisungen oder die Berechnung von Umgebungshüllen für eine darauf aufbauende Kollisionserkennung. Besonders effektiv wirkt sich die Verwendung von statischen Objekten bei Tests auf die generelle Sichtbarkeit dieser Objekte aus, da sich in diesen Fällen eine effiziente Graphenstruktur mit räumlicher Aufteilung vornehmen lässt (sogenanntes 'Space Partitioning'[13]). Auch für verteilten Szenarien (siehe Kapitel 3.2) können statische Objekte viele Vorteile bringen, da sie im Normalfall nur lesbar sind und somit eine Verteilung der Daten mittels simpler Kopien oder Replikaten ermöglichen.

Bei dynamischen Objekten gilt dies alles nur sehr eingeschränkt oder gar nicht, da ständige Positionsbewegungen keine feste Raumzuteilung erlauben oder Veränderungen am Objekte jeweils neue Zustandssortierungen erfordern, wodurch etwaige Berechnungen im Vorfeld unbrauchbar werden und zur Laufzeit wiederholt werden müssen. Je nach Komplexität und Anzahl der hierfür nötigen Verarbeitungsschritte lässt sich dies aber nicht mehr in Echtzeit erledigen. Für diese Fälle ist nach wie vor der ursprüngliche Transformationsgraph die optimale und flexibelste Szenengraphenart.

12 Ein Beispiel hierfür sind Transparenzeffekte. Es ist hierbei notwendig, von der Grafikkarte zuerst alle opaken Objekte zeichnen zu lassen und danach erst die transparenten Objekte, um so sicherzustellen, dass die jeweils korrekten Hintergründe für die durchscheinenden Objekte bereits vorhanden sind.

13 Beim Space Partitioning werden spezielle Szenengraphen, sogenannte 'Space Partition Trees', erstellt, bei welchen sich zur Laufzeit mit einfachen Vergleichen effizient abfragen und auswerten lässt, ob sich ein bestimmtes Objekt innerhalb oder außerhalb einer festgelegten Region befindet. Sie benötigten dafür allerdings einen verhältnismäßig großen Aufwand zu ihrer Erstellung. Insbesondere wenn das ursprünglichen Objekte auf den Regionsgrenzen liegt, muss dieses Objekte hierzu „zerlegt“, d.h. entlang den einzelnen Regionsgrenzen aufgeteilt werden. Der hierfür benötigte Rechenaufwand ist zumeist so groß, dass es sich selbst für kleine Szenen mit wenig Objekten nicht lohnt, dies zur Laufzeit vorzunehmen.

3.1.3 Moderne Szenengraphen

Heutige Szenengraphen bzw. komplette, darauf aufbauende Graphiksysteme (siehe Kapitel 3.1.4) sind inzwischen allgemein verfügbar und in vielen Fällen sogar als Open-Source-Software erhältlich. Die grundlegenden Aufgaben eines Szenengraphens erfüllen dabei erwartungsgemäß alle Systeme und werden daher oftmals zur Abgrenzung von ihren Mitbewerbern mit einen breiten Fundus an zusätzlichen Erweiterungen ihrer Grundfunktionalität ausgestattet. Die Einsatzbereiche und Zielgruppen dieser Komplettsysteme sind dabei zwar verschieden, je nach dem ursprünglichen Umfeld, für welches sie konzipiert wurden; mit ihrer Weiterentwicklung und dem damit verbundenen Zuwachs an enthaltenen Funktionen verschwimmen diese ursprünglich spezialisierten Ausrichtungen zwischen den einzelnen Systemen aber zunehmend wieder.

Viele Komplettsysteme erheben inzwischen sogar den Anspruch, zur schnellen Erstellung[14] von Grafikanwendungen für Benutzer geeignet zu sein, die selbst keine eigenen Grafikroutinen oder gar ein ganzes, darauf aufbauendes Rendersystem erstellen möchten, indem sie diese Komponenten selbst bereits alle enthalten. Durch die gebotene Funktionsfülle versuchen diese Systeme dabei so viele Einsatzbereiche wie möglich abzudecken. Allerdings gilt es hierbei zu beachten, dass der große Funktionsumfang und die Fülle an angebotenen Möglichkeiten sich auch in einem immensen Zuwachs der Größe und Komplexität dieser Systeme niederschlägt und sich im Extremfall durch teilweise nicht benötigte aber (standardmäßig) eingebundene Funktionen sogar negativ in ihrer Leistung durch überflüssige, interne Berechnungen auswirken kann. Ebenfalls steigt dadurch der Einarbeitungsaufwand beim Einsatz dieser Systeme beträchtlich; die oftmals suggerierte einfache Handhabung entspricht daher in vielen Fällen nicht der Wirklichkeit.

Da je nach Anwendung und Einsatzbereich ein einzelner Szenengraph nicht alle gewünschten oder benötigten Anforderungen direkt umsetzen kann (so bildet ein nach Grafikzuständen sortierter Szenengraph nicht mehr direkt zugreifbar die räumliche Anordnung der Objekte zueinander ab, wie sie für eine eventuelle, parallel dazu arbeitende Kollisionserkennung benötigt wird), sind aufwändige Extraktionen oder gar Umsortierungen des Szenengraphens für die einzelnen Verwendungszwecke zur Laufzeit nötig. Dies erfüllen die heutigen Szenengraphensysteme in der Regel zwar tadellos, benötigen dafür aber auch eine gewisses Mindestmaß an Rechenleistung des zugrundeliegenden Systems.

Darüber hinaus sind fast alle aktuellen Szenengraphensysteme vom Grundkonzept her nach wie vor als Szenengraphen für Einzelplatzsysteme ausgelegt; eine Verteilung wird erst durch Zusatzbibliotheken oder separate Schichten zur Verfügung gestellt (siehe Kapitel 3.2.3).

Als mögliche, zukünftige Weiterentwicklung heutiger Szenengraphensysteme gelten Multi-Szenengraphensysteme. Hier werden innerhalb des Szenengraphensystems intern mehrere Szenengraphen, optimiert für unterschiedliche Einsatzbereiche, parallel vorgehalten, so dass unterschiedliche Verarbeitungseinheiten (Renderengine, Kollisionserkennung, Animationen, ...) direkt auf die von ihnen benötigte und jeweils darauf optimierte Graphenstruktur zugreifen kann. Diese Multi-Szenengraphensysteme müssen dabei aber mit den Eigenheiten dieser mehrfach vorhandenen Graphen zurecht kommen, welche vom Grundkonzept teilweise verteilten Szenengraphen sehr ähnlich sind (siehe Kapitel 3.2). So ist es für sie ebenfalls zwingend notwendig, vorgenommene Änderungen an einem Szenengraphen auch auf die anderen, parallel vorgehaltenen Szenengraphen zu transferieren, was einen erhöhten internen Verwaltungsaufwand durch eine doppelte bzw. mehrfache „Buchführung“ der darauf vorgenommenen Operationen bedeutet. Insbesondere die hierfür nötige Synchronisierung und Konsistenzhaltung zwischen den einzelnen Szenengraphen wird dabei zu einem immer größer werdenden Problem. Nutzbare Vertreter von

14 Insbesondere sind hier Projekte im industriellen Umfeld auf dem Gebiet des 'Rapid Prototypings' gemeint.

Multi-Szenengraphensystemen existieren daher in dieser angesprochenen Form bislang noch nicht.

3.1.4 Bestehende Arbeiten

Die ersten entstandenen Szenengraphen, welche die Grundlage für weitere, verteilte Systeme bilden, sind Szenengraphen für Einzelplatzanwendungen. Sie sind gemeinsam mit der Entwicklung der Computergrafik entstanden und haben sich parallel zu deren Weiterentwicklung im Laufe der Zeit ebenfalls weiter verbessert und spezialisiert. Daher gibt es mittlerweile im Umfeld von Szenengraphen eine Reihe von existierenden Konzepten und Systemen. Nachfolgend werden die zugrundeliegenden Konzepte der wichtigsten Vertreter genauer beschrieben.

3.1.4.1 Inventor bzw. Open Inventor

'Inventor' ([20], [21]) (oder ursprünglich 'IRIS Inventor') stellt eines der ersten Szenengraphensysteme überhaupt dar und wurde von Silicon Graphics im Jahre 1988 entwickelt, um die aufwändige Programmierung von Grafikszenen auf ihren Grafikstationen mit OpenGL komfortabler und einfacher zu machen. Später wurde es dann im Quellcode unter dem Namen 'Open Inventor' freigegeben und wird auch heute noch (zumeist im wissenschaftlichen Bereich) eingesetzt.

Open Inventor stellt eine objektorientierte Bibliothek zur Erstellung und Verwaltung von 3D-Grafiken auf einzelnen Rechnern dar. Es setzt direkt auf der OpenGL-Schnittstelle auf, verwaltet einen internen Szenengraphen und bietet Möglichkeiten zur Manipulation der darzustellenden Szene. Der Szenengraph setzt sich dabei aus einzelnen Knoten (den 'Nodes') zusammen, welche einen Transformationsgraphen bilden und ihrerseits wiederum zu größeren Gruppen (den 'Groups') zusammengefasst werden können, die dann gemeinsam manipulierbar sind. Die eigentlichen (Geometrie-)Daten und Attribute eines Knotens werden in zugehörigen Feldern abgespeichert, wobei sich diese Felder teilweise mit den Feldern anderer Szenengraphenknoten verknüpfen lassen (sofern sie vom gleichen Typ sind), um durch Ereignisse bzw. getätigte Eingaben gemeinsame Änderungen an mehreren, logisch zusammengehörenden Objekten vornehmen zu können. Alternativ können Felder ihre Werte auch von verschiedenen sogenannten 'Engines' bekommen – beispielsweise der Zeitwert eines Timers –, um so automatische Ablaufsteuerungen oder Animationen mit den Objekten zu ermöglichen.

Die Traversierung und Manipulation des Szenengraphens zum Erledigen bestimmter Aufgaben nehmen sogenannte 'Actions' vor. Es gibt verschiedene Actions für unterschiedlichste Aufgabenbereiche, so z.B. für das Rendern des Szenengraphens oder auch nur eines Teils davon, aber auch für das Suchen bzw. Ändern von Objekten und deren Transformationsdaten oder deren Eigenschaften im Szenengraphen, bis hin zu relativ komplexen Vorgängen wie das komplette automatisierte Erstellen und Berechnen von Umgebungshüllen für ganze Objekte inklusive ihrer Unterobjekte (z.B. für Kollisionsberechnungen). Weiter können über spezielle Actions auch Ereignisse (z.B. Mausklicks auf ein Objekt, ...) an den Szenengraphen selbst übergeben werden, was einer rudimentären Art einer Benutzerschnittstelle gleichkommt; diese Actions sorgen dann dafür, dass alle Knoten innerhalb des Szenengraphens, die auf bestimmte Ereignisse reagieren können, diese auch übermittelt bekommen. Über einstellbare Traversierungsarten zu den Unterknoten (Pre-, In-, Post-Order) kann dabei auf die Reihenfolge der Abarbeitung der aufgetretenen Ereignisse bzw. allgemein der Abarbeitung der Actions Einfluss genommen werden.

3.1.4.2 OpenGL Performer

Das 'OpenGL Performer Toolkit' [22] (anfangs auch 'IRIS Performer') wurde wie Inventor ebenfalls von Silicon Graphics entwickelt. Ein Teil der ursprünglichen Entwicklergruppe von Inventor legte 1991 ihren Fokus mehr auf die Erzielung einer maximalen Leistungsfähigkeit für das 3D-Grafiksystem als auf eine einfache Bedien- und Programmierbarkeit, wie sie bei Inventor im Vordergrund stand, und kreierte so in einem eigenen Projekt schließlich OpenGL Performer.

Die wesentlichen Szenengraphenkonzepte entsprechen daher denen von Open Inventor; Unterschiede ergeben sich hauptsächlich durch eingearbeitete Optimierungen und einer modulareren Implementierung. So wurde der interne Szenengraph angepasst, um eine Sortierung nach den Grafikzuständen, wie sie OpenGL kennt, vorzunehmen. Ebenfalls werden Transformationen bei statischen Objekten durch eine direkte Einberechnung in die Geometriedaten wegoptimiert. Allerdings bietet Performer im Unterschied zu Open Inventor keine eigenen Mechanismen an, um beispielsweise Ein- und Ausgabegeräte anzusprechen oder Animationen abzuarbeiten. Dafür sind die Knoten des Performer-Szenengraphens relativ einfach gehaltene C++-Objekte und werden daher auch über Standard-C/C++-Funktionen erzeugt, manipuliert und gelöscht, ohne dass spezielle Methoden oder Actions hierfür benötigt werden.

Performer bietet weiter die Möglichkeit, eine parallele Verarbeitung der Szenengraphendaten zu bewerkstelligen (beispielsweise für Multi-Threading oder auch um mehrere Bildschirme zu unterstützen). Hierfür wurden 'Pipes' eingeführt, welche jeweils die Softwareabstraktion einer Renderpipeline von Grafikkarten darstellen und daher einem Renderdurchlauf der Szene entsprechen. Eine Pipe muss aber nicht notwendigerweise die gesamte Szene rendern, sondern kann sich auch auf einzelne Szenengraphenabschnitte beschränken. Performer selbst kann prinzipiell mit beliebig vielen Pipes umgehen. Empfohlen wird aber, nur so viele Pipes einzusetzen, wie sie auch hardwareseitig zur Verfügung stehen[15], da ansonsten die Gesamtleistung durch die dann erforderliche Mehrfachnutzung der tatsächlich vorhandenen Renderpipelines pro darzustellendem Bild drastisch sinkt.

Durch die mögliche Parallelisierbarkeit der Datenzugriffe auf den Szenengraphen mit Hilfe der Pipes werden andererseits aber zusätzliche Synchronisierungen in Performer benötigt, welche das System intern über Nachrichten in gemeinsam genutzten Speicherbereichen umsetzt. Allerdings ergeben sich hieraus an manchen Stellen Engpässe im System, da nun jede Pipe für Operationen im Szenengraphen immer alle Knoten traversieren muss, selbst wenn diese in der jeweiligen Pipe eigentlich nicht sichtbaren (und dadurch prinzipiell vernachlässigbaren) Unterobjekten bzw. Teilgraphen entsprechen; die Sichtbarkeit könnte inzwischen durch andere Pipes beeinflusst bzw. verändert worden sein. Ebenfalls kann hierdurch bei unvorsichtigen Aufrufen der recht komplexen Performer-Schnittstelle ein undeterministisches Verhalten nicht immer gänzlich ausgeschlossen werden. Trotzdem bietet OpenGL Performer eine hohe Leistungsfähigkeit für 3D-Anwendungen, insbesondere auch durch eine mögliche, individuelle Anpassung des Szenengraphensystems an die im Rechnersystem vorhandene Grafikhardware.

Ein interessanter Punkt bei Performer stellt die Möglichkeit dar, die Bildwiederholraten der zugehörigen Renderengine für die Darstellung fest vorzugeben, wie es beispielsweise bei Anwendungen für Simulationen gewünscht ist bzw. teilweise auch benötigt wird. Performer kann über eigene interne Messroutinen erkennen, ob durch die Renderengine diese vorgegebenen Frameraten erreicht werden oder nicht. Im negativen Fall versucht Performer dies über verschiedene Mechanismen zu kompensieren, beispielsweise durch Verwenden einer weniger komplexen

15 Eine normale Grafikkarte besitzt eine Renderpipeline. Spezialhardware, besonders im (Grafik-)Workstation-Bereich, können aber auch mehrere Renderpipelines besitzen.

Geometrie mit geringerer darzustellender Dreiecksanzahl (Verminderung des 'Level of Detail') oder durch Rendern eines kleineren Bildes mit somit geringerer Auflösung und einhergehend niedrigerer Pixelanzahl, welches dann anschließend automatisch wieder auf die normale Bildgröße hochskaliert wird ('Dynamic Video Resolution').

3.1.4.3 VRML

'VRML' ('Virtual Reality Modeling Language', [w42]) an sich stellt kein eigenständiges Szenengraphensystem dar, sondern lediglich eine textuelle Beschreibungssprache für 3D-Objekte und modellierte Szenen und wurde als eine Austauschsprache für solche Daten via das Internet konzipiert. Allerdings basiert die Struktur der Objektdaten auf einem impliziten Szenengraphen, der von einer Anwendung, die diese VRML-Daten interpretieren und darstellen will – dem VRML-Viewer –, nachgebildet werden muss. Üblicherweise ist dieser VRML-Viewer direkt in den gängigen Internetbrowsern integriert oder kann zumindest über Browser-Plugins die VRML-Daten darstellen.

VRML hat einen gemeinsamen Ursprung mit Open Inventor und wurde 1994 von Silicon Graphics als Erweiterung für das von Inventor bereits vorhandene Dateiformat entwickelt. 1997 wurde es von der ISO als VRML97-Standard spezifiziert und sollte als der Standard für 3D-Beschreibungen im Internet dienen. Inzwischen gibt es bereits den VRML-Nachfolger X3D, eine 3D-Modellierungssprache welche auf XML und VRML aufbaut und auch ein gegenüber der rein textuellen Beschreibung von VRML kompakteres Binärformat ermöglicht, was einer der Hauptkritikpunkte an VRML darstellte. Bis heute konnten sich aber weder VRML noch X3D weitreichend durchsetzen.

Die Grundstruktur im Aufbau von VRML-Dateien ist durch den gemeinsamen Ursprung auch sehr ähnlich zu Open Inventor. Es gibt verschiedene Knotentypen, welche den Knoten im Szenengraphen entsprechen, und Felder für ihre Attribute. Es können auch Materialien zugewiesen werden (Farbe, Texturen, ...) und es gibt zusätzliche Angaben zum 'Level of Detail' bei den Geometriedaten. Auch gibt es die Möglichkeit, System- oder Benutzerereignisse miteinbeziehen zu lassen und hierüber Auswirkungen auf den Szenengraphen zu ermöglichen. Dies wird über verschiedene „Sensoren” ermöglicht, welche vom VRML-Viewer zur Verfügung gestellt werden müssen und die Ereignisse entgegennehmen (beispielsweise ein 'TouchSensor', der durch ein Mausklick auf ein Objekt aktiviert wird) und über sogenannte 'Route'-Anweisungen mit einem Zielobjekt (d.h. dem betroffenen Szenengraphenknoten) verknüpft werden können.

Der VRML97-Standard bietet darüber hinaus sogar die Möglichkeit, Multimediadaten wie Audio oder MPEG-Filme mit in den Szenengraphen einzubinden. Sie entsprechen dann Multimediaobjekten, die mittels Referenzen auf die eigentlichen Daten, welche separat zur Verfügung gestellt werden müssen, verknüpft sind, und können mittels Koordinaten und Zusatzinformationen genauso in der 3D-Welt wie normale Grafikobjekte positioniert und (mit Einschränkungen) manipuliert werden.

Ein wesentlicher Unterschied von VRML zu anderen Formaten bzw. Systemen ist jedoch, dass die Traversierungsreihenfolge der von VRML beschriebenen Objektstruktur nicht festgelegt ist. Es können so unterschiedliche Interpretationen bei der Darstellung durch verschiedene VRML-Viewer entstehen, die beim Einsatz von VRML berücksichtigt werden müssen.

3.1.4.4 Java3D

'Java3D' [w43] wurde 1997 von Sun Microsystems entwickelt, um eine plattformübergreifende Schnittstelle für 3D-Anwendungen unter Java zu erhalten, welche ursprünglich für grafische Internet-Anwendungen gedacht war. Es handelt sich dabei um eine Bibliothek von Java-Klassen mit einer eigenen Schnittstelle, welche inzwischen auch als Open Source verfügbar ist.

Der Szenengraph von Java3D setzt sich aus bestimmten (Java3D-)Objekten für Knoten und Kanten zusammen, wobei – im Gegensatz zu den bisher beschriebenen Szenengraphensystemen – nur auf Blattebene die eigentlichen Renderdaten auftreten dürfen (Dreieckspunkte, Texturen, ...) und die vorgeschalteten Knoten ausschließlich Transformations- oder Gruppierungsinformationen in sich tragen. Spezielle Knotentypen ('Switch Nodes') erlauben es, Abfragen beim Traversieren des Szenengraphens einzubauen, um so eine sich daraus ergebende Darstellung an bestimmte Bedingungen knüpfen zu können. Mittels weiterer Knotentypen kann auf verschiedene Ereignisse reagiert werden (sogenannte 'Behaviours'). Eine zusätzliche Knotenkategorie (die 'Interpolatoren') erlaubt die zeitliche Steuerung von Untergruppen, womit sich Animationen realisieren lassen.

Java3D bietet außerdem die Möglichkeit (erneut mittels spezieller Knotentypen), ganze Teilbäume eines Szenengraphens mehrfach von unterschiedlichen Stellen aus zu referenzieren, um Speicherplatz zu sparen. Eine Änderung innerhalb dieses gemeinsamen Teilbaums wirkt sich dementsprechend auch an allen referenzierten Stellen aus. Ebenso ist es aber auch möglich, die Struktur der Teilbäume (automatisch) kopieren zu lassen und nur die Blattdaten gemeinsam zu verwenden, so dass sich die Objekte unabhängig voneinander bewegen lassen, aber nach wie vor eine gemeinsame (und somit speicherplatzsparende, da nur einmal vorhandene) Geometrie besitzen.

Ein Szenengraphen(teil)baum wird in Java3D über ein spezielles „Schauplatzobjekt“ ('Locale Object'), welches als Einstiegspunkt für eine Traversierung dient, in das virtuelle Java3D-Universum eingebunden und wird mittels spezieller, hochauflösender Koordinaten[16] darin positioniert. Alle eingebunden Subgraphen werden dann relativ zu diesen speziellen Koordinaten berechnet.

Das Kamerasystem in Java3D wurde so ausgelegt, dass es zwischen einer virtuellen und einer physikalischen Sichtweise auf eine modellierte 3D-Welt unterscheidet. Dadurch ist es möglich, dass eine Szene auf verschiedenen Ausgabegeräten dargestellt werden kann (vom einfachen Monitor bis zum 6-seitigen CAVE; [w47]), ohne dass an der eigentlichen Szene oder der von einer Anwendung gewünschten Sicht darauf etwas geändert werden muss. Die tatsächliche Sichtweise für die eigentliche Darstellung auf dem jeweiligen Anzeigegerät errechnet Java3D dann automatisch aus der Verknüpfung der virtuellen Sicht mit den lokalen Eigenschaften bzw. Gegebenheiten der physikalisch vorhandenen und angesteuerten Hardware.

3.1.4.5 OpenSceneGraph

'OpenSceneGraph' (OSG) [w44] ist ein aktuelles[17] Szenengraphen-System ('Toolkit') für die Erstellung von grafischen (3D-)Anwendungen. Es wurde von OpenGL-Performer inspiriert, wodurch sich die internen Strukturen dieser Szenengraphen stark ähneln, und hat sich ebenfalls auf die Bereitstellung einer größtmöglichen Leistung für die Darstellung einer Szene, d.h. auf möglichst hohe Bildwiederholraten, spezialisiert. Somit zielt OSG hauptsächlich auf Spiele oder VR-Anwendungen als Einsatzgebiet ab. Zu diesem Zweck werden verschiedenste Optimierungen

16 Bei diesen hochauflösenden Koordinaten handelt es sich um 256-Bit-Fließkommawerte, davon 128 Bit Vorkommastellen. Mit der dadurch erreichbaren Auflösung ist es theoretisch möglich, Positionierungen im Java3D-Universum in der Größenordnung von Protonen bis hin zu mehreren Millionen Lichtjahren vorzunehmen.

17 Die bereits mehrfach überarbeitete Version 1.0 erschien erst Ende 2005; inzwischen ist Version 2.4 aktuell.

eingesetzt und unterstützt, die dann beispielsweise eine schnelle Überprüfung auf die Sichtbarkeit der Szenengraphenobjekte erlauben, den Szenengraphen nach den OpenGL-spezifischen Zuständen umsortieren oder zusammengefasste und vorkompilierte Display- und Geometrielisten für ganze (statische) Unterobjektstrukturen oder Subgraphen verwendet.

OSG ist ein in C++ geschriebenes, objektorientiertes Framework und ist als Open Source verfügbar. Es liegt für verschiedene Plattformen vor und erleichtert so eine Portierung einzelner, darauf aufbauender Anwendungen. Zusätzlich zu seinem großen, standardmäßig mitgeliefertem Funktionsumfang bietet es die Möglichkeit, über hinzufügbare Bibliotheken eine Vielzahl weiterer Zusatzfunktionen für nahezu alle möglichen Anforderungen einzubinden. So gibt es Importfilter für alle gängigen 2D- und 3D-Grafikformate und Spezialfunktionen für die Nutzung ganzer Partikelsysteme, komplette Animationsengines wie z.B. für Knochenskelettanimationen oder zur Shader-Ansteuerung. Durch eben diesen Versuch, alle möglichen Einsatzfelder abzudecken – insbesondere durch die hinzufügbaren Bibliotheken – ist OSG relativ komplex in der Handhabung und sehr groß vom Datenumfang.

Ein Anwendungsfeld, auf das OSG ebenfalls abzielt, ist, die schnelle Erstellung von visuellen Projekten aller Art zu ermöglichen, ohne dass sich ein Entwickler explizit mit Grafikprogrammierung auskennen muss. Daher wird es insbesondere bei kleineren Programmen verstärkt eingesetzt – zumeist kurze oder schnelllebige Projekte bzw. Projektarbeiten, bei denen keine Zeit für die Erstellung eines eigenen Szenengraphens bzw. der zugehörigen Renderengine bleibt. Dies trägt so zu einer beständig wachsenden Verbreitung dieses Szenengraphensystems bei.

3.1.4.6 OpenSG

'OpenSG' ([23], [24], [w45]) entstand nahezu zeitgleich zu OpenSceneGraph (Ende 1999). Es stellt aber trotz der Namensähnlichkeit ein komplett eigenständiges System dar und besitzt keinerlei Verwandtschaft mit OSG. Das Einsatzgebiet von OpenSG bewegt sich in erster Linie im industriellen Umfeld als Szenengraphensystem für CAD-Entwicklungsumgebungen oder auch für Augmented-Reality-Anwendungen.

Der interne Aufbau des Szenengraphens ähnelt dem Open-Inventor-Konzept. Knoten mit Feldern erstellen den von dort bekannten Aufbau des Szenengraphens, Actions übernehmen die Aufgabe, Teile des Szenengraphens zu manipulieren oder zu rendern. OpenSG bietet aber auch ähnlich zu OpenGL Performer die Möglichkeit, den Zugriff auf den Szenengraphen zu parallelisieren, um verschiedene Tasks oder Threads gleichzeitig darauf arbeiten zu lassen. Potentielle Zugriffskonflikte werden – im Gegensatz zu OpenGL Performer – durch eine Aufteilung in nicht geteilte Daten und replizierte Daten vermieden. Jeder Task/Thread erstellt zu diesem Zweck bei den gemeinsamen Abschnitten innerhalb des Szenengraphens ein privates Replikat zu dem von ihm bearbeiteten Subgraphen und führt nur auf diesem eigene Berechnungen durch. Anstehende Änderungen werden ab dann in eine globale Änderungsliste eingetragen. Beim Traversieren des Szenengraphens durch eine beliebige Action werden diese Änderungen wiederum in das zugehörige Replikat des jeweiligen Tasks/Threads übernommen und eingebaut.

OpenSG wurde als Einzelplatzszenengraph entwickelt, unterstützt mittlerweile aber auch mehrere Rechner, wobei der Einsatz dieser zusätzlichen Rechner allerdings nur auf die Realisierung eines großen Rendersystems beschränkt bleibt, um z.B. mittels mehrerer Projektoren, denen jeweils ein eigener Rechner mit eigener Grafikkarte zugrunde liegt, unterschiedliche Blickwinkel einer Szene bzw. eines Objektes gleichzeitig darstellen zu können. Eine weitere Interaktion bzw. Manipulationsmöglichkeit auf den zusätzlichen Rechnern ist nicht vorgesehen. Die Verteilung beinhaltet somit lediglich ein Replizieren des gesamten Szenengraphens auf den weiteren

Darstellungsrechnern ohne Einfluss- oder Manipulationsmöglichkeit durch diese, so dass sie alle nur eine gemeinsame Datensicht auf den Szenengraphen haben und aus dieser lediglich ihren lokalen Blickwinkel ableiten können. Die Darstellung wird über Nachrichten synchron gehalten (wobei eine ausreichend geringe Netzwerklatenz vorausgesetzt wird bzw. diese sich auf die möglichen Frameraten auswirkt) und muss von der Anforderung daher lediglich für einen interaktiven Betrieb ausreichen, aber nicht wirkliche Echtzeitfähigkeit liefern. Insgesamt verbleibt OpenSG daher bei der Einordnung als Einzelplatzszenengraph.

Im Rahmen eines vom Bundesministerium für Bildung und Forschung geförderten Projektes wurden bis 2003 etliche zusätzliche Leistungsmerkmale hinzugefügt. Hierzu gehört die Handhabung komplexer bzw. riesiger Szenen[18], was unter anderem durch eine Aufteilung des Szenengraphens in dynamische und statische Elemente und die Umwandlung letzterer in effizientere Space Partition Trees erreicht wird. Um trotzdem eine schnelle Darstellung zu ermöglichen, kann OpenSG weit entfernte Objekte als fertige Bilder vorberechnen und dann in die momentane Szene einfügen. Auch sind Schnittstellen zur erweiterten Geometriemanipulation erstellt worden, die aus mathematischen Beschreibungen und Funktionen Geometriedaten erstellen und die hieraus generierten Dreiecksdaten bzw. Objekte direkt in den Szenengraphen einbauen (was bislang ansonsten nur von Modellierungsprogrammen bewerkstelligt wird).

3.2 Verteilung von Szenengraphen

Sobald mehrere Benutzer gemeinsam in einer virtuellen Umgebung teilnehmen und auf sie Einfluss nehmen wollen, kommen Verteilungs- und Synchronisierungsaspekte zu den bisherigen Aufgaben der eingesetzten Szenengraphen hinzu. Im Idealfall sollte ein verteilter Szenengraph dieselben Möglichkeiten und Merkmale bieten, wie ein lokaler Einzelplatzszenengraph, ohne aber dem Benutzer bzw. Entwickler weitere und eventuell unnötige Komplexität angesichts dieser Verteilungsaspekte aufzubürden. Zwischen den bislang bestehenden Systemen gibt es aber große Unterschiede bei der Annäherung an dieses Ideal. Zu den wichtigsten Punkten, die es bei der Verteilung eines Szenengraphens zu beachten gilt, gehören die Synchronisierung der Abläufe und Operationen auf dem Szenengraphen zwischen allen beteiligten Rechnern bei Änderungen, sowie eng damit verbunden die Konsistenzhaltung der Daten bzw. die Sichtweisen der einzelnen Rechner auf die gemeinsamen Teile des Szenengraphens.

Für gewöhnlich wird die Verteilung eines Szenengraphens nachträglich zu einem bereits bestehenden Einzelplatzszenengraphen mittels zusätzlicher Schichten oder Bibliotheken hinzugefügt. Über die Art der damit verbundenen Verteilungsstrategie lassen sich die zugehörigen Szenengraphensysteme grob in zwei Kategorien aufteilen: den Client-Server-basierten Varianten und den Peer-to-Peer-basierten Ansätzen.

3.2.1 Client-Server-basierter Ansatz

Bei den Client-Server-basierten Varianten besitzt ein zentraler Server bzw. eine Serverinstanz (welche sich auch auf dem gleichen Rechner wie ein Klient befinden kann), die vollständige Kontrolle über den global gültigen Szenengraphen (siehe Abbildung 27). Änderungen am Szenengraphen kann der Server jederzeit direkt und selbst vornehmen, wodurch er auch eine ständig aktuelle Sichtweise auf die zugehörigen Daten hat.

18 Hiermit sind Szenen mit Millionen bis Milliarden von Eckpunkten gemeint bzw. Szenen, deren Datenmengen zu groß sind, als dass sie im Grafikkarten- oder Hauptspeicher komplett vorgehalten werden können.

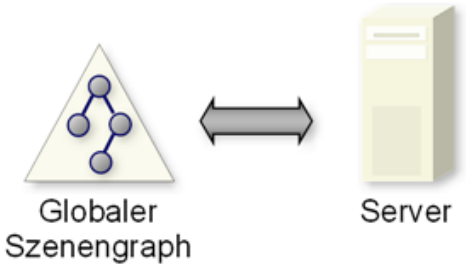

Abbildung 27: Ein Server übt die Kontrolle über den global gültigen Szenengraphen aus

Die Klienten, welche nun ihre lokalen Sichtweisen mittels der Vorgabe des globalen Szenengraphens darstellen wollen, können auf diesen dahingegen nicht direkt bzw. selbständig zugreifen, sondern müssen erst die benötigten Informationen vom Server geliefert bekommen. Damit die Klienten aber nicht bei jedem einzelnen Zugriff auf die Datenstruktur den kompletten Szenengraphen übermittelt bekommen müssen, halten sie lokale Kopien des globalen Szenengraphens vor (siehe Abbildung 28). Dieser muss somit nur noch einmalig vom Server vollständig übertragen werden; danach reicht es aus, wenn der Server inkrementell auftretende Änderungen an die Klienten weiterleitet, welche diese dann wiederum in ihre lokalen Szenengraphen einpflegen.

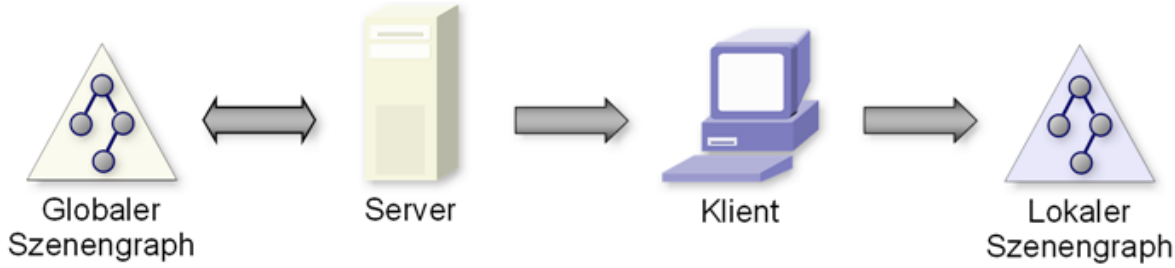

Abbildung 28: Der Server übermittelt die Szenengraphendaten an den Klienten

Will nun ein Klient eine Aktion in der virtuellen Umgebung vornehmen, welche dadurch eine Änderung im Szenengraphen zur Folge hat, so müssen diese Änderungen auch allen anderen Teilnehmern mitgeteilt werden (beispielsweise falls sich ein Avatar einen Schritt vorwärts bewegt). Da der Klient aber keinen direkten Zugriff auf den global gültigen Szenengraph besitzt, kann er diese Änderung nicht selbst durchführen, sondern muss stattdessen den Server darum bitten, dies für ihn zu erledigen (siehe Abbildung 29). Er stellt daher zuerst eine Anfrage an den Server mit seinem Änderungswunsch (1). Der Server überprüft nun die Zulässigkeit und Ausführbarkeit der Anfrage und nimmt dann die dazu benötigten Berechnungen vor. Danach fügt er die resultierenden Änderungen in den globalen Szenengraphen ein (2). Um nun alle teilnehmenden Klienten von der Änderung in Kenntnis zu setzen, muss der Server anschließend Aktualisierungsnachrichten nicht nur an den ursprünglichen Klienten mit der gestellten Anfrage, sondern auch an alle restlichen teilnehmenden Klienten versenden (3)[19]. Die Klienten übernehmen schließlich diese Aktualisierungen und gleichen ihre lokalen Szenengraphenkopien mit der übermittelten Nachricht ab (4).

19 Aus Effizienzgründen bietet sich hierfür zumeist der Einsatz von gruppenbasierten Kommunikationsverfahren bzw. Multicast-Nachrichten an.

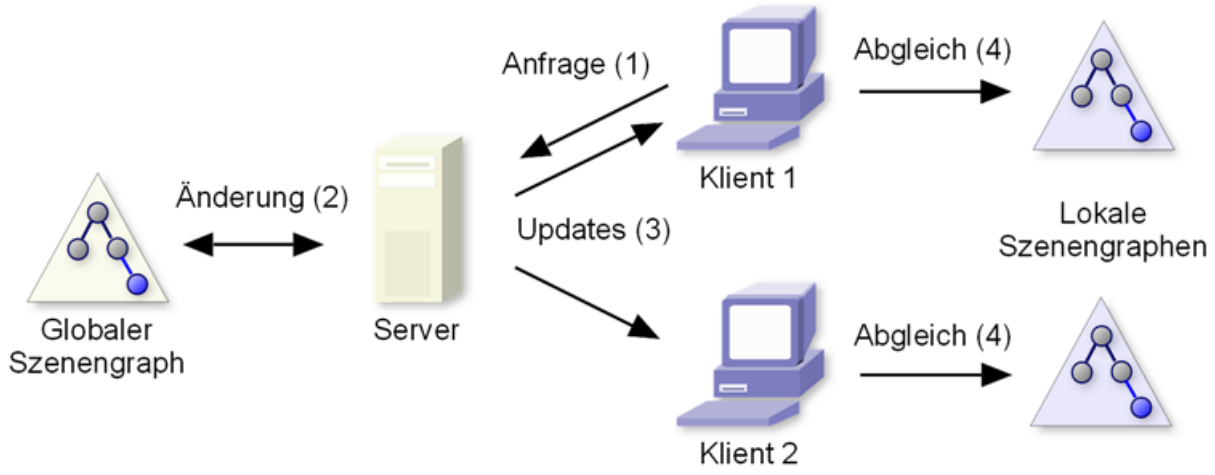

Abbildung 29: Der Aktualisierungssvorgang zwischen Server und Klienten

Über diese Vorgehensweise kann relativ einfach die Konsistenz des globalen Szenengraphens gewährleistet werden, da nur der Server die alleinige Befugnis besitzt, diese Daten zu manipulieren, und gleichzeitige Änderungswünsche mehrerer Klienten mittels Serialisierung durch ihn aufgelöst werden. Auf diese Weise lassen sich Nichtdeterminismen bei Manipulationen des Szenengraphens durch eventuell simultan auftretende und gegensätzliche Anfragen der Klienten vermeiden.

Allerdings wird der Server in diesem Szenario auch schnell zu einem Engpass des Gesamtsystems, da von ihm alle benötigten Berechnungen, ausgelöst durch die Klientenanfragen, vorgenommen werden müssen. Außerdem hat er für den nachfolgenden Update-Mechanismus einen zusätzlichen Verwaltungsaufwand zu bewältigen, da er sich ständig die aktuell teilnehmenden Rechner merken muss; einerseits, um prüfen zu können, ob die Änderungsanfragen zulässig sind und nicht von fremden Rechnern stammen, und andererseits um zu wissen, an welche Klienten er nachfolgend die Aktualisierungsnachrichten zu senden hat (sofern hierfür keine generelle Gruppenkommunikation verwendet wird). Dies setzt gewisse Mindestanforderungen an die Leistungsfähigkeit des Servers voraus, die beim Einsatz eines Client-Server-basierten Ansatzes zu beachten sind bzw. umgekehrt, die je nach Serverausstattung den Einsatz solcher Systeme auf eine bestimmte maximale Anzahl von Klienten beschränkt.

3.2.2 Peer-to-Peer-basierter Ansatz

Ein weiterer Ansatz stellt die Verteilung der Szenengraphendaten mittels eines Peer-to-Peer-Netzwerkes dar. Die einzelnen Peers sind dabei alle gleichberechtigt und können selbst den vorhandenen Szenengraphen manipulieren (siehe Abbildung 30). Um dabei eine gleichmäßige Sicht aller teilnehmenden Peers auf einen einheitlichen Szenengraphen zu gewährleisten, müssen sich die Peers bei auftretenden Änderungen entweder darüber abstimmen oder diese zumindest an die anderen Peers weiterpropagieren und ihre lokalen Varianten daraufhin abgleichen. Dies geht im Vergleich zu einer zentralen Serverinstanz zumeist nur mit einer abgeschwächten Form der Konsistenz in Bezug auf die gemeinsamen Szenengraphendaten einher oder erfordert sogar bestimmte Grade an Inkonsistenzen zwischen den Peers. Ein Echtzeitverhalten ist oftmals nur durch diese Einschränkungen erreichbar, da beispielsweise durch eventuelle Abstimmungsverfahren große zeitliche Latenzen auftreten, die einem Echtzeitverhalten bei starker Konsistenz zuwiderlaufen, oder es nicht immer gewährleistet ist, dass alle Änderungen auch zu einem bestimmten Zeitpunkt bei allen Peers im Netzwerk ankommen.

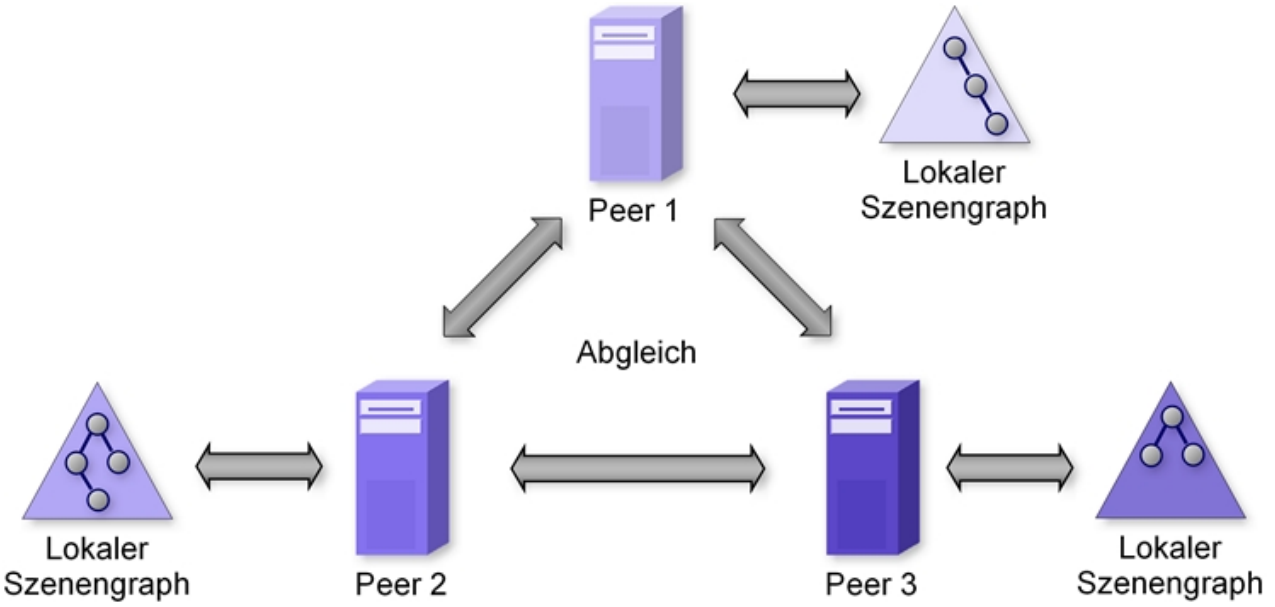

Abbildung 30: Eine Verteilung mit gleichberechtigten Peers

Da diese Punkte für viele Einsatzgebiete nicht gewünscht oder gar tolerierbar sind, sind reine Peer-to-Peer-basierte Szenengraphensysteme nur in wenigen Anwendungsfällen sinnvoll einsetzbar und werden deshalb auch nur selten umgesetzt. Etwas häufiger existieren daher Hybridsysteme, die durch unterschiedliche Mischformen aus den Peer-to-Peer- und Client-Server-basierten Ansätzen lauffähige Systeme realisieren. Teilweise sind Peer-to-Peer-Systeme allerdings auch nur Spezialfälle eines ursprünglich rein Client-Server-basierten Systems, welche durch den Peer-to-Peer-Ansatz auf ein größeres Rechnernetz als dem Ursprünglichen ausgedehnt werden und damit über die bisherigen Serverleistungsgrenzen hinausreichen sollen. Hier tauschen sich dann nur die Server untereinander als gleichberechtigte Peers aus, die Klienten sind nach wie vor einem einzelnen Server fest zugeordnet.

3.2.3 Bestehende Arbeiten

In den nachfolgenden Beschreibungen wird auf die Konzepte einiger der wichtigsten und bekanntesten Vertreter von verteilten Szenengraphensystemen näher eingegangen. Die meisten hiervon sind dabei der ungleich häufiger anzutreffenden Kategorie der Client-Server-basierten Szenengraphensysteme zuzuordnen als den Peer-to-Peer-Systemen.

3.2.3.1 Distributed Open Inventor

'Distributed Open Inventor' [25] ist die Erweiterung des Open Inventor Toolkits, die eine Verteilung des Szenengraphens über Rechnergrenzen hinweg realisiert. Sie bietet für die darauf aufsetzende Anwendung eine gemeinsame Sicht auf den verteilten Szenengraphen, ähnlich wie bei einem DSM-Ansatz. Der verteilte Szenengraph (oder zumindest Teile davon) wird dabei von allen teilnehmenden Rechnern repliziert. Gemeinsame Zugriffe auf verteilte Objekte regeln dabei eigenständige 'Sequencer', die jedem Objekt individuell zugeordnet sind und welche als eine Art Serverinstanz für diese Objekte erscheinen. Diese Sequencer fangen alle Zugriffe auf das zugehörige Objekt ab, serialisieren diese und veranlassen nach vorgenommenen Änderungen anschließend die Übermittlung davon an die anderen Rechner.

Die Synchronisierung der von den Teilnehmern möglichen Manipulationsaktionen auf den

Szenengraphendaten zwischen den Rechnern geschieht über explizite Update-Nachrichten. Um diese möglichst kurz und den Kommunikationsaufwand dabei gering zu halten, sind verschiedene Nachrichtentypen für unterschiedliche Operationen möglich (vom Aktualisieren von einzelnen Wertfeldern bis hin zum Löschen oder Einfügen ganzer Subgraphen). Diese Nachrichten werden über einen eingebundenen, zuverlässigen Multicast-Dienst zugestellt.

Tritt ein neuer Knoten dem Distributed-Open-Inventor-Netz bei, so bekommt er als Ausgangszustand den momentanen Szenengraphenzustand eines bestehenden Knotens übermittelt; an diesem nimmt er dann seine lokalen Anpassungen und Änderungen vor. Bis dieser Startzustand über das Netzwerk übermittelt ist, wird die Kommunikation mit anderen Knoten angehalten, um weitere Abweichungen des bis dahin übertragenen Graphens zu vermeiden.

Distributed Open Inventor unterscheidet bei der Verteilung von Szenengraphendaten auch, ob diese Daten lokal oder global zu behandeln sind. Jeder Rechner hat dabei seine „lokale Variation", was den unterschiedlichen Sichtweisen der einzelnen Benutzer auf die gemeinsame Szene entspricht. Der Szenengraph jedes teilnehmenden Rechners ist zuerst einmal lokal; die verteilten Szenengraphenobjekte werden nur als Subgraph in den lokalen Szenengraphen eingehängt. Diese Unterscheidung ist allerdings nicht transparent für einen Anwendungsentwickler. D.h., dieser muss selbst dauerhaft darauf achten, ob er mit „privaten" oder mit globalen Einträgen arbeitet, da für den verteilten Teil auch syntaktisch unterschiedliche Anweisungen für eigentlich semantisch gleiche Operationen aufgerufen werden müssen.

Wegen dieser Unterscheidung gibt es auch noch eine zusätzliche Trennung zwischen 'Input Streams' und 'Input Events'. Input Streams sind nur für Audio- und Videodaten zuständig und kommen nur im lokalen Teil des Szenengraphens vor bzw. müssen zu deren Verwendung lokal auf dem Rechner vorliegen. Dies bewirkt, dass Input Streams nur bereits abgespeicherte Daten wie Videos oder Musik abrufen können, aber keinen Austausch dieser Daten zwischen Rechnern ermöglichen, um beispielsweise eine Sprachübertragung zu etablieren. Diese Input Streams beeinflussen somit auch nur den lokalen Teil des Szenengraphens und sind daher nicht mit den Input Streams anderer Teilnehmer gekoppelt bzw. synchronisiert. Dahingegen werden über die Input Events, welche global gültig sind und bei ihrem Auftreten per Nachricht an alle teilnehmenden Rechner im Netzwerk gesendet werden, die eigentlichen Manipulationen am Szenengraphen vorgenommen (Bewegungen von Avataren und Objekten, ...), aber auch generelle Ereignisse innerhalb der Szene ausgelöst und propagiert, welche u.a. wiederum zur Steuerung der lokalen Input Streams verwendet werden können (gemeinsames Ein-/Ausschalten eines Videos in der virtuellen Welt auf allen Rechnern, ...).

3.2.3.2 Repo-3D

Bei 'Repo-3D' [26] handelt es sich um eine objektorientierte, hochsprachliche Grafikbibliothek, welche entwickelt wurde, um in erster Linie eine Entwicklungsumgebung zum Rapid Prototyping für verteilte, interaktive 3D-Anwendungen über heterogene Rechnerausstattungen hinweg zu ermöglichen. Sie setzt auf Repo [27] auf, einer interpretierten, verteilten Programmiersprache, welche selbst wiederum in Modula-3 implementiert ist und einen um verteilte Replikate erweiterten Nachfolger von Obliq (ebenfalls einer Interpretersprache für objektorientierte, verteilte Berechnungen; [28]) darstellt. Repo-3D (wie auch Repo) baut auf eine Vielzahl bereits bestehender Modula-3-Pakete auf und fügt diese in mehreren Schichten zusammen, um die von Repo-3D gewünschten Anforderungen zu realisieren. So wird beispielsweise die Verteilung von Objekten nicht von Repo-3D selbst, sondern von dem eingebundenen Modula-3-Paket 'Network Objects' übernommen; hierauf setzt wiederum das Paket 'Replicated Objects' auf, welches die Replikateverwaltung und -verteilung für die verteilten Szenengraphdaten übernimmt, die dann

schließlich auch in Repo-3D nutzbar sind. Verschiedene weitere eingebundene Pakete kümmern sich beispielsweise um die Ereignisverarbeitung oder die 3D-Grafikdarstellung ('Anim-3D' bzw. 'DistAnim-3D').

Das Ziel, welches Repo-3D als oberste Schicht dieser Paketsammlung verfolgt, ist, dass verteilte Objekte wie in einem DSM-System zugreifbar sein sollen bzw. dass DSM-ähnliche Verteilungsaspekte einer Anwendung zur Verfügung gestellt werden, indem es ihr die „Illusion eines gemeinsamen verteilten Speichers" [26] vermittelt.

Bei der Verwendung von Repo-3D-Objekten muss ein Entwickler explizit zwischen drei verschiedenen Verteilungsmechanismen und somit implizit zugeordneten Konsistenzmodellen unterscheiden. So gibt es die 'Simple Objects', welche kompletten Kopien der verteilten Objekte entsprechen und die keinerlei Abhängigkeiten untereinander haben. Somit sehen die Simple Objects auch keinerlei vorgenommene Änderungen untereinander. Weiter gibt es die 'Remote Objects'. Diese nutzen ein Client-Server-ähnliches Prinzip zur Aktualisierung der Objekte untereinander. Ein Remote Object exisitert dabei nur auf demjenigen Rechner, welches dieses Objekt als erstes erstellt hat. Alle weiteren Kopien hiervon entsprechen nur Proxy-Objekten, welche alle Änderungen an das Ursprungsobjekt weiterleiten, auf dem dann diese Änderungen ausgeführt werden. Ein zugehöriges „Überwacher-Objekt" generiert anschließend Update-Nachrichten, mit denen diese Änderungen wieder an alle Proxies zurückpropagiert werden. Die Datenübertragung in diesem Fall geschieht dabei ähnlich zu einem Java-RMI-Aufruf, einschließlich dem dazugehörigen Marshalling bzw. Unmarshalling der Objektdaten. Schließlich gibt es noch 'Replicated Objects', bei denen das verteilte Objekt als wirkliches Replikat auf den verschiedenen Rechnern vorgehalten wird und Änderungen am eigenen Replikat auch auf allen anderen Replikaten sichtbar werden. Dabei muss ein Programmierer beim Erstellen von Methoden für Replicated Objects explizit unterscheiden, ob dieser Methodenaufruf jeweils nur einem Lesezugriff entspricht (d.h. es wird keine Veränderung am Objekt vorgenommen) oder ob der Zugriff eine Aktualisierung der Daten nach sich zieht. Dies bedeutet somit, dass die Konsistenzhaltung der replizierten Objekte zumindest teilweise Aufgabe des Programmierers ist. Eine Zwischenschicht bei Repo-3D führt für die Abarbeitung der Updates pro Replicated Object zusätzlich einen 'Sequencer' ein (ähnlich denjenigen bei Distibuted Open Inventor), der sich um die Abfolge der Befehlsausführung bei simultanen Zugriffen auf das Objekt kümmert und somit festlegt, in welcher Reihenfolge diese Updates bei allen Replikaten sichtbar werden. Auftretende Änderungen werden dabei über blockierende Updates behandelt, d.h. der Initiator einer Änderung wartet mit der eigentlichen Ausführung der Änderung, bis er – wie auch alle anderen Replikate – eine Nachricht vom Sequencer mit der Erlaubnis zur Ausführung erhält. Sollen mehrere Änderungen zusammenhängend durchgeführt werden, ist es wiederum Aufgabe des Programmierers, sich um die Atomarität der Befehlsausführung auf den replizierten Objekten zu kümmern.

Um dabei eventuelle Netzwerklatenzen oder Verzögerungen bei interaktiven Ereignissen zu vermeiden, können Änderungen bei Repo-3D auch zuerst lokal durchgeführt und angesammelt und erst später an die anderen Replikate übermittelt werden, um daraufhin die globale Konsistenz der einzelnen Replicated Objects wieder herzustellen bzw. zu „reparieren". Auch können „lokale Variationen" verwendet werden, um diesen Punkten entgegenzuwirken.

Im Szenengraphen-Design von Repo-3D selbst gibt es eine Fülle an verwendbaren Knotentypen. Diese werden 'Graphical Objects' (GO) genannt und sind in 21 unterschiedlichen Ausprägungen vorhanden. Die GOs lassen sich über zugehörige Eigenschaften ('Properties') beeinflussen und teilweise auch darüber steuern. Hierzu gibt es 101 verschiedene Properties in 17 unterschiedlichen Kategorien, welche alle mit jedem GO kombiniert werden können.

Im Gegensatz zu anderen Systemen, beispielsweise Java3D, ist es bei Repo-3D nicht möglich, Verweise auf bestehende Objekte zu erstellen, um bei mehrfach verwendeten Objekten Speicherplatz zu sparen. Es müssen in diesem Fall immer komplette, eigenständige Kopien angefertigt werden. Die GOs selbst sind mit dem nicht verteilten 'RootGO' verknüpft, welcher als lokaler Wurzelknoten den Einstiegspunkt für die Traversierung auf den jeweiligen Rechnern darstellt.

Repo-3D benötigt zum synchronisierten Betrieb auf allen beteiligten Rechnern außerdem eine globale Zeit, um z.B. Animationen gleichmäßig ablaufen zu lassen. Dies wird von Repo-3D aber nicht selbst bereitgestellt. Es setzt daher den Einsatz eines Zeitsynchronisierungsprotokolls wie z.B. NTP voraus, um welches sich der Anwender bzw. Entwickler zusätzlich kümmern muss.

3.2.3.3 blue-c Distributed Scene Graph

Der 'blue-c Distributed Scene Graph' [29] basiert auf dem OpenGL Performer Toolkit. Er erweitert den ursprünglichen, lokalen Szenengraphen um einen verteilten Teilgraphen mit den dazugehörigen Netzwerkanbindungen. Ausgerichtet ist der blue-c-Szenengraph vor allem für gemeinschaftliches Arbeiten aller Arten.

Der verteilte Abschnitt des blue-c-Szenengraphens setzt sich hierfür aus speziellen 'Shared Nodes' zusammen, welche von Performer-Objekten abgeleitet sind und ein zusätzliches Synchronisierungsinterface enthalten. Die Shared Nodes sind auf den teilnehmenden Rechnern repliziert und werden komplett mit allen Daten (einschließlich der Geometrie- und Texturdaten) synchron gehalten. Ein eigener Synchronisierungsservice traversiert dazu einmal pro darzustellendem Bild den verteilten Abschnitt des Szenengraphens und wertet dabei eventuell durch Änderungen gesetzte 'Dirty Flags' aus. Diese Änderungen der betroffenen Knoten werden dann über Update-Nachrichten als Broadcast im Netz propagiert. Dabei dient eine spezielle Knoten-ID zum schnellen, direkten Auffinden der jeweils betroffenen Knoten bei den informierten Rechnern. Änderungen dürfen aber nur vom jeweiligen Besitzer eines Objektes durchgeführt und propagiert werden. Will ein Teilnehmer Änderungen auf einem fremden Knoten ausführen, so muss er erst selbst Eigentümer des betroffenen Objektes werden, indem er einen Wechsel der Eigentümerrechte beim ursprünglichen Besitzer beantragt. Hierfür gibt es in blue-c verschiedene Anfrage- aber auch Ablehnungsmeldungen. Ein abgeschwächter Sperrmechanismus bei blue-c erlaubt es einer Anwendung aber auch, ein Szenengraphenobjekt jederzeit zu ändern, selbst wenn sie nicht der Eigentümer des Objektes ist, um schnelle Reaktionszeiten des Systems auf Eingaben und somit ein zeitnahes Arbeiten auf den Szenendaten zu ermöglichen, indem lokale Änderungen möglich sind. Falls das zu bearbeitende Objekt nicht dem momentanen Rechner bzw. dessen Anwendung gehört, wird das Eigentürmerrecht automatisch im Hintergrund vom eigentlichen Besitzer angefordert, um diese Änderungen anschließend propagieren zu können. Ankommende Update-Meldungen für dieses Objekt haben aber vor eigenen Änderungen Vorrang und setzen das Objekt zurück bzw. es werden nur die eingetroffenen Aktualisierungsinformationen ausgeführt; eigene Änderungen werden verworfen. Falls somit mehrere Teilnehmer gleichzeitig auf dem selben Objekt arbeiten, kann es zu „Sprüngen” des manipulierten Objekts durch diesen Mechanismus kommen; bei der Abarbeitung der Änderungen gewinnt dann immer der momentane Besitzer des Objektes.

Die Netzwerkkommunikation in blue-c geschieht über ein CORBA-basiertes Grundgerüst, um eine plattformunabhängige Netzwerkschicht zu ermöglichen. Insbesondere die Video- und Audio-Streaming-Dienste von CORBA werden für die blue-c-Kommunikation verwendet, was auch eine Einbindung von Video- und Audiodaten innerhalb von blue-c-Anwendungen ermöglicht.

3.2.3.4 Avango / Avocado

'Avango' – teils auch als 'Avocado' bezeichnet (Avocado war der Vorläufer von Avango; [30], [31]) – stellt ebenfalls ein Framework eines Szenengraphens für verteilte VR-Anwendungen basierend auf OpenGL Performer dar.

Die einem Benutzer zur Verfügung stehenden Avango-Objekte lassen sich dabei in zwei Kategorien einteilen: einerseits in die Szenengraphenknoten und andererseits in die Sensoren. Die normalen Szenengraphenknoten entsprechen dabei der übernommenen Funktionsweise der ursprünglichen Szenengraphenknoten von OpenGL Performer. Die Sensoren hingegen fungieren als Schnittstelle für verschiedene Eingabegeräte und dienen zur Manipulation der Szenengraphenknoten. Da sie nicht im ursprünglichen OpenGL-Performer-System vorgesehen waren, lassen sie sich nicht direkt in den Szenengraphen selbst als eigene Knoten integrieren. Die von den Eingabegeräten empfangenen Ereignisse werden aber von den Sensoren verarbeitet und anschließend an die entsprechenden Stellen zu davon betroffenen Szenengraphenknoten weitergeleitet.

Die Avango-Objekte selbst speichern wiederum ähnlich zu Open Inventor ihre Attribute in Feldern, welche vom zugehörigen Avango-Objekt (dem 'Feldcontainer') überwacht werden. Die von OpenGL Performer abstammenden Szenengraphenknoten sind dabei über (Mehrfach-)Vererbungen in das Avango-Feldcontainer-Konzept eingebunden, erlauben so aber auch nur die Nutzung der in den Performer-Objekten enthaltenen Fähigkeiten im Avango-System und nicht aller vom OpenGL-Performer-System insgesamt möglichen Fähigkeiten. Durch diese vereinheitlichte Zusammenführung aller Performer-Objekte wird eine gemeinsame Schnittstelle für den Zugriff auf alle darin enthaltenen Felder geschaffen, welche – sofern sie zueinander Typ-kompatibel sind – auch beliebig miteinander verknüpft werden können (sogar über Feldcontainergrenzen hinweg) und so ein dichtes Geflecht für weitreichende Manipulationsmöglichkeiten erlauben.

Änderungen an Knoten bzw. den zugehörigen Felddaten erkennt das Avango-System dabei durch eine im Feldcontainer automatische gerufene 'Notify'-Methode bei einem Zugriff darauf. Hierdurch kann zusätzlich auf Änderungen reagiert werden, wobei sich über die Verknüpfungen auch Zustandsänderungen in weiteren Feldern von anderen Feldcontainern realisieren lassen. Die Weiterleitung dieser Änderungen zu den anderen Feldcontainern geschieht dabei über eigene Nachrichten zu den jeweiligen Feldcontainern. Wurden Felder verändert, wird von jedem Feldcontainer anschließend einmal vor dem Rendern des nächsten Frames eine 'Evaluate'-Methode gerufen, welche diese Änderungen verarbeitet und sichtbar werden lässt.

Das Avango-System synchronisiert insgesamt transparent für den Benutzer den Szenengraphen auf allen beteiligten Rechnern. Allerdings wird aus Performance-Gründen hierzu bei jedem Rechner eine vollständige, lokale Kopie des Szenengraphens vorgehalten, weshalb somit auch alle auftretenden Änderungen von jedem Rechner eingepflegt werden müssen. Die Änderungsnachrichten werden deshalb nicht nur lokal an die betroffenen Feldcontainer gesendet, sondern auch über das Netzwerk an die jeweiligen Feldcontainer aller beteiligter Rechner. Insgesamt betrachtet realisiert diese Vorgehensweise dadurch ein verteiltes Ereignisbehandlungsmodell.

3.2.3.5 Syzygy

Das 'Syzygy'-System [32] wurde zum Erstellen von eng gekoppelten bzw. stark synchronisierten VR-Anwendungen auf PC-Clustern entworfen, wie sie bei verteiltem Rendern z.B. für ein CAVE-System zum Einsatz kommen. Es realisiert eine mehrschichtige Bibliothek mit Werkzeugen, die die Steuerung, Verwaltung und Kommunikation einer solchen VR-Anwendung in dem Cluster übernehmen.

Die unterste Schicht im Syzygy-System stellt dabei die Kommunikationsschicht für Syzygy-Nachrichten dar, welche konzeptuelle Mechanismen verwendet, wie sie bereits von CORBA oder RPCs her bekannt sind. Somit geschieht die Kommunikation für die jeweiligen Nachrichten nach einer Client-Server-artigen Struktur. Für eine bestimmte Aufgabe übernehmen eine oder mehrere Serverkomponenten die Verwaltung und Kommunikation mit den Klienten, welche wiederum aber jeweils nur mit einem dieser Server verbunden sein dürfen. Fällt für einen Klienten die zugehörige Serverkomponente aus, sucht er einen anderen Server für die gleiche Aufgabe und verbindet sich neu mit diesem, um danach wie zuvor weiterarbeiten zu können.

Syzygy selbst baut auf das verteilte Betriebssystem 'Phleet' [33] auf bzw. bettet es in sein Schichtenkonzept mit ein, da Phleet selbst auch auf die gerade angesprochene Kommunikationsschicht zugreift. Phleet bietet Mechanismen zum Starten, Anhalten und Überprüfen von Prozessen auf einzelnen Clusterrechnern, denen es auch im laufenden Zustand Nachrichten zukommen lassen kann. Änderungen in der Clusterverwaltung kann Phleet über globale Sperren automatisch vornehmen.

Das Syzygy-System bietet für die Entwicklung von darauf aufsetzenden Anwendungen zwei unterschiedliche Ausprägungen seines Frameworks als Betriebsart zur Auswahl an: zum Einen ein Framework mit einem verteilten Szenengraphen, welches insbesondere für VR-Anwendungen gedacht ist, und zum Anderen ein Master-Slave-Konzept für allgemeine, verteilte Clusteranwendungen, die synchronisiert werden können bzw. müssen.

Das von Syzygy für VR-Anwendungen bereitgestellte, verteilte Szenengraphen-Framework entspricht dabei einer verteilten Datenbank mit benannten Knoten als Datenbankeinträgen. Diese Datenbankknoten werden jeweils als Syzygy-Nachrichten an andere Clusterrechner versendet und dort wieder ausgepackt. Die zugehörige Datenbankverwaltung erlaubt die Replikation dieser Datenbank auf unterschiedlichen Clusterrechnern, indem mittels Sperren die Daten synchronisiert an eine (auch leere) Datenbank eines anderen Clusterrechners weitergereicht werden. Die Datenbankknoten für den Szenengraphen bieten dabei eine Vielzahl unterschiedlicher Knotentypen für speziell zur Grafikverarbeitung benötigter Daten an, so z.B. Punkt-, Linien-, und Dreiecksknoten, sowie Knoten für Transformationen, Texturen oder Materialien. Diese Knotentypen beinhalten Aufrufe an OpenGL-Kommandos (bzw. ganze Sequenzen dieser Kommandos), um die eigentlichen Grafikdaten zu speichern oder das Rendering vorzunehmen. Über spezielle Syzygy-Nachrichten an einen (oder auch mehrere) Clusterrechner, welcher als Server für den Szenengraphen fungiert, können diese Grafikknoten manipuliert werden. Für die Anwendung selbst bleibt dieser interne Client-Server-Ansatz hinter der Syzygy-Schnittstelle verborgen. Somit bietet diese Framework-Variante eine konsistente Sicht aller teilnehmenden Rechner für das Rendern der vorgehaltenen Daten (bezeichnet als 'Visual Consistency'), allerdings zu Lasten der Darstellungsperformance.

Ebenso wie mit Grafikdaten kann Syzygy auch mit Audio und Sound umgehen. Prinzipiell wird hierfür die gleiche Infrastruktur wie beim Szenengraphen für Grafikdaten genutzt, indem ein paralleler Soundgraph eingesetzt wird. Ein Clusterrechner übernimmt erneut die Rolle eines Servers und erstellt den Soundgraphen. Als Server stellt er dann auch die Audiodaten zur Verfügung, welche mit der Verteilung des Graphens an die Klienten propagiert und von diesen über Sound-Renderer dargestellt bzw. hörbar gemacht werden. Über Transformationen können die Audioquellen analog zur Grafik positioniert und dadurch in ihrer Lautstärke oder anderen Eigenschaften beeinflusst werden.

Auch können Benutzer- und Geräteeingaben bei Syzygy über das Netz im Cluster verteilt und verarbeitet werden. Dies geschieht durch 'Virtual Devices', nach deren Konzept sogenannte

„Eingabequellen“ die zugehörigen Daten liefern (z.B. von einem Joystick). Diese Daten können anschließend von „Eingabeknoten“ modifiziert werden (beispielsweise das Kalibrieren der rohen Joystick-Daten), wobei dies bereits auf anderen Rechnern geschehen kann als auf demjenigen, von dem die Eingabedaten stammen, um schließlich von „Eingabesenken“ verarbeitet zu werden (d.h. die Anwendung, die die Joystick-Daten als Bewegung interpretiert).

Für diejenigen Fälle, in denen das Szenengraphen-Framework die gewünschten Anforderungen einer zu erstellenden (VR-)Anwendung nicht erfüllt, kann das alternative Master-Slave-Framework von Syzygy verwendet werden. Dieses bietet Programmierern die Möglichkeit, Anwendungen zu erstellen, die mehrere allgemeine, aber synchron gehaltene Instanzen auf verschiedenen Clusterrechnern benötigen. Eine Instanz übernimmt dabei die Rolle des Masters, welcher sämtliche Eingaben geliefert bekommt, diese verarbeitet und an die anderen Instanzen – die Slaves – zu bestimmten Zeitpunkten (auch verzögert) weiterleitet. Der Datenaustausch geschieht dabei mittels spezieller, gemeinsamer Speicherblöcke, um ein möglichst flexibles Austauschformat für beliebige Daten zu erreichen. Insgesamt können von Syzygy in dieser Betriebsart zwar die einzelnen Instanzen einer Anwendung in ihrer Arbeit synchron gehalten werden (etwa die Renderengine einer damit realisierten VR-Anwendung), die Konsistenz der verwendeten Daten kann durch die unterschiedlichen Zeitpunkte ihrer jeweils lokalen Erstellung (insbesondere begründet durch die verzögerte Weiterleitung von Eingaben) auf den einzelnen Rechnern dabei aber nicht garantiert werden.

3.2.3.6 DIVE

Bei 'DIVE' [34] handelt es sich um eine Abwandlung einer verteilter Datenbank, mittels der ein Szenengraph umgesetzt und verteilt wird. Diese Datenbank kann sogenannte DIVE-Welten beherbergen, deren Datensätze die Knoten und Daten der virtuellen Welt beherbergen. Jede (verteilte) Anwendung hat ein eigenständiges und von anderen Anwendungen unabhängiges Replikat dieser Datenbank, wodurch prinzipiell beliebig viele unterschiedliche VR-Anwendungen innerhalb des DIVE-Systems ermöglicht werden. Umgekehrt bedeutet dies aber auch, dass sobald keine Anwendung mehr auf eine virtuelle Welt zugreift, diese „stirbt”. Beim Wiedereinstieg in diese Welt werden dann nur die ursprünglichen Standardwerte geladen, alle Änderungen und Entwicklungen innerhalb der Welt sind verloren. Über einen expliziten „Persistenz-Prozesses” pro Welt kann dies aber vermieden werden, indem dieser periodische Speicherungen veranlasst (vergleichbar mit dem Pageserver-Konzept innerhalb des Plurix-Systems; siehe [8]).

Der zugrundeliegende Verteilungsmechansimus von DIVE (basierend auf ISIS, einem fehlertoleranten Kommunikationssystem der Cornell Universität; [35]) geschieht über einen Nachrichtenaustausch mittels eines Peer-to-Peer-Multicast-Systems. Dabei werden für den Betrieb zwei unterschiedliche Nachrichtentypen zur Verfügung gestellt: einen zuverlässigen Multicast, der für Änderungen am Szenengraphen und somit an der beteiligten Datenbank verwendet wird, um für die DIVE-Welten eine Konsistenz der Datenbankdaten gewährleisten zu können; aber auch einen unzuverlässigen Multicast für Datenströme, z.B. Audio- und Videodaten für die virtuelle Welt.

Ein neuer Teilnehmer, der einer virtuellen Welt beitreten möchte, findet diese über einen DIVE-Server, der als eine Art von Namensdienst für die DIVE-Welten fungiert. Dieser Server ermöglicht letztlich nur den Beitritt in die zugehörige Multicast-Gruppe der gewünschten Welt, mit der die versendeten Updates anschließend mitverfolgt werden können. Alle Teilnehmer einer Welt müssen sich initial beim DIVE-Server melden, wodurch dieser bei vielen gleichzeitigen Anmeldungen zu einer Engstelle werden kann, allerdings auch nur einmalig zum Erhalt der Multicast-Channel-Daten benötigt wird.

Durch die Verwendung von DIVE-Proxyservern können auch unterschiedliche Netzwerke verbunden bzw. eine Kommunikation über das Internet ermöglicht werden. Diese Proxyserver tunneln dazu die benötigten Multicastverbindungen über die Fremdnetze; die Verbindung zwischen den Proxyservern stellt daher wiederum eine Engstelle dar und benötigt eine entsprechend große Bandbreite.

Im Betrieb werden Änderungen der Szenengraphendaten zuerst auf der lokalen Datenbankkopie ausgeführt und erst in einem späteren Schritt an die zugehörigen Replikate auf anderen Rechnern weitergemeldet. Um einen hohen Skalierungsgrad zu ermöglichen, werden im DIVE-System auch kleinere Abweichungen der Weltkopien auf den einzelnen Rechnern toleriert, welche durch automatische Hintergrunddienste über eine gewisse Zeitspanne[20] hinweg wieder ausgeglichen werden. Um dies vor dem Anwender zu verbergen, werden Dead-Reckoning-Strategien auf den Rechnern eingesetzt. Des Weiteren werden die DIVE-Welten in Subhierachien mit eigenen Multicast-Gruppen aufgeteilt, für die sich zumeist nur eine kleinere Anzahl an Rechnern bzw. Anwendungen interessieren (sollen), so dass nicht immer von allen die gesamten Updates für die komplette, gemeinsame DIVE-Welt erfasst werden müssen.

3.2.3.7 Myriad

Das 'Myriad'-System [36] realisiert einen verteilten Szenengraphen mittels Peer-to-Peer-Mechanismen. Es baut auf dem bereits beschriebenen Syzygy-System auf (siehe Kapitel 3.2.3.5), wandelt den ursprünglich Client-Server-basierten Verteilungsaspekt zwischen den einzelnen Rechnern aber in eine dezentrale Variante ab. Jeder teilnehmende Rechner im Myriad-Netzwerk entspricht dabei einem sogenannten 'Reality Peer', wobei es sich letztlich nur um ein C++-Objekt handelt. Diese Reality Peers besitzen intern einen eigenständigen, individuellen Syzygy-Szenengraphen, allerdings ohne dessen Verteilungsfunktionalität. Die Knotentypen des Syzygy-Szenengraphens korrespondieren dabei aber weiterhin mit den bestehenden OpenGL-Kommandos, um die Grafikdaten verarbeiten zu können.

Um dem Myriad-Netzwerk beizutreten (siehe Abbildung 31), muss ein neuer Reality Peer sich zuerst bei einem zentralen Vermittler, dem 'Syzygy-Broker' (ähnlich der Funktionsweise eines CORBA-ORBs), registrieren (1). Von diesem erhält der Reality Peer im Gegenzug auf seine Anfrage eine Liste mit weiteren, bereits bekannten Reality Peers (2), zu denen er dann eine Verbindung aufbauen kann (3). Über anschließende Update-Nachrichten zwischen den verbundenen Peers (4) werden Aktualisierungen vorgenommen und die Änderungen der individuellen Syzygy-Szenengraphen ausgetauscht, wobei jeder Reality Peer aus den erhaltenen Updates nur die für ihn relevanten Updates herausfiltert. Zusätzlich können die einzelnen Peers über spezielle Feedback-Nachrichten die gewünschte Aktualisierungsrate und somit das generelle Nachrichtenaufkommen abhängig von ihrer eigenen möglichen Verarbeitungsgeschwindigkeit beeinflussen und gegebenenfalls bei den sendenden Peers verlangsamen. Diese merken sich pro verbundenem Peer die jeweils mögliche bzw. gewünschte Aktualisierungsrate und reduzieren dementsprechend die Anzahl ihrer Update-Nachrichten für die betroffenen Peers.

Bei dem vom Myriad-Netzwerk aufgebauten Szenario werden Inkonsistenzen in den Daten der einzelnen Szenengraphen zwischen den Peers toleriert. Diese Inkonsistenzen werden im Laufe der Zeit über einen auf jedem Peer laufenden Hintergrundprozess mittels der Update-Nachrichten wieder abgeglichen und behoben. Die Entwickler des Myriad-Systems sprechen hierbei von einer flüchtigen Inkonsistenz ('Transient Inconsistency') als realisierter Konsistenzform.

20 Die Zeitspanne ist abhängig von der Größe und Fehleranfälligkeit des eingesetzten Netzes, d.h. der möglichen Umlaufzeit einer Nachricht und kann im WAN-Fall bis in den Sekundenbereich reichen.

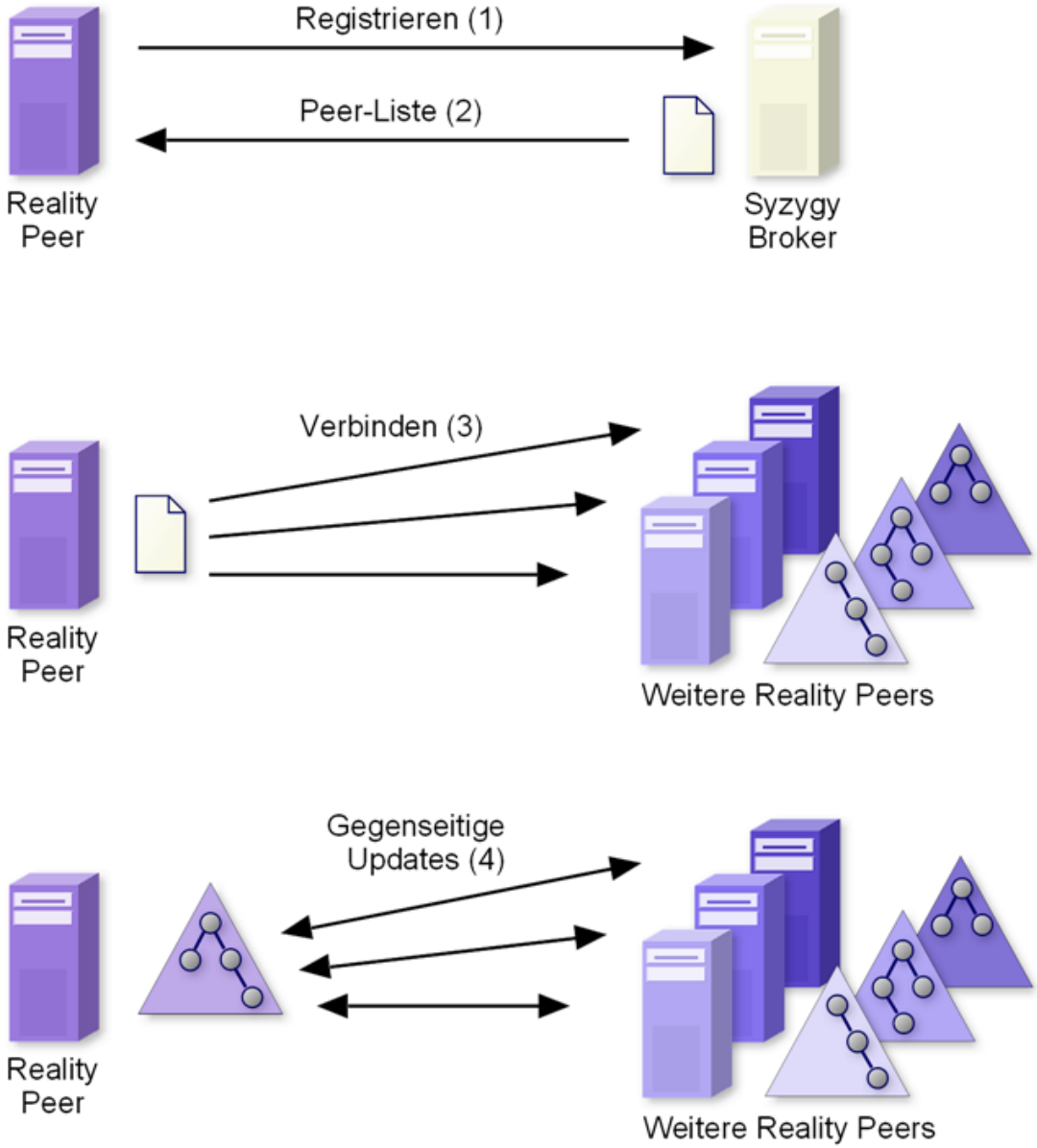

Abbildung 31: Verbindungsaufbau und Arbeitsweise des Myriad-Peer-to-Peer-Netzwerkes

Um diese Angleichungen effizienter vornehmen zu können, werden im Myriad-System die Reality Peers in vier verschiedene Peer-Unterarten eingeteilt. 'Pull-Peers' verbinden sich zu einem anderen Peer und holen sich Änderungen des Szenengraphens von diesem ab, können selbst aber keine Änderungen am Szenengraphen vornehmen oder propagieren. Zu diesem Zweck gibt es im Gegensatz dazu die 'Push-Peers'. Ihnen ist es erlaubt, Änderungen in ihrem Szenengraphen aktiv vorzunehmen und diese auch an andere Peers weiterzuleiten, allerdings nehmen sie selbst keine Änderungen von anderen Peers an. Ein 'Feedback-Peer' wiederum verbindet sich nur mit einem einzelnen anderen Peer, kann mit diesem dafür aber einen gegenseitigen Austausch von Änderungen der beiden individuellen Szenengraphen vornehmen, um diese zu synchronisieren. Schließlich gibt es noch die 'Shadow-Peers', eine spezielle Variante des Feedback-Peers. Ein Shadow-Peer stellt nur die eigentliche Szenengraphenstruktur für andere Peers zur Verfügung, hält aber die zugehörigen Inhalte der Szenengraphenknoten (Geometrie, Texturen, Materialien, ...) nicht vor. Daher ist sein Einsatz nur zur schnellen Synchronisierung der Graphenstruktur mit einem anderen Peer gedacht,

z.B. bei schmalbandigen Verbindungen oder im WAN-Fall.

Über sogenannte 'Reality Maps' werden die Verknüpfungspunkte bzw. Überlappungen der individuellen Szenengraphen der einzelnen Reality Peers durch eindeutige Szenengraphenknoten-IDs beschrieben. So können die einzelnen Peers ihre Szenengraphen erweitern oder auch erkennen, ob bei Änderungen ein Knoten lokal oder auf einem fremden Rechner geändert werden muss. Über einen zusätzlichen Sperrmechanismus für Szenengraphenknoten lassen sich Änderungen an einem Knoten sogar auf einen bestimmten Peer beschränken.

Um verteilte VR-Anwendungen im Myriad-Netzwerk zu steuern (z.B. um Geometriedaten zu laden und zu speichern oder die Verbindungs- bzw. die Peerarten einzuteilen), aber auch um die Szenengraphendaten selbst durch Benutzerinteraktion ändern zu können, werden (Python-)Skripte aufgerufen, die mittels eines Interpreters innerhalb des Myriad-Systems abgearbeitet werden. Über Proxyobjekte lassen sich dabei Methodenaufrufe überall ins Netz verteilen, auch wenn die ausführenden Objekte auf anderen Peers angesiedelt sind; der hierzu eventuell nötige Datenaustausch geschieht über RPC-ähnliche Mechanismen.

Zusätzliche, spezielle Synchronisierungs-Peers erlauben im Myriad-System auch die bildgenaue Darstellungssynchronisierung einzelner Peers durch Bereitstellen einer zentralen Synchronisierungsinstanz. Diese sorgt für einen identischen und synchronisierten Rendervorgang auf den zugeordneten Peers, indem es die von ihnen zu verarbeiteten Update-Nachrichten zentral festlegt und die Bildschirmspeicherumschaltung für die jeweiligen Grafikkarten regelt. Allerdings bedeutet das in diesem Fall auch, dass die langsamste Grafikkarte die Bildwiederholrate aller synchronisierten Peers bestimmt (plus einer zusätzlichen Netzwerklatenz für die Umschaltnachrichten) und der Synchronisierungs-Peer dabei zu einer Engstelle im Myriad-System wird.

3.3 Transaktionaler Szenengraph

Im Rahmen dieser Arbeit wurde ein alternativer und neuartiger Ansatz entwickelt, um ein verteiltes Szenengraphensystem zu realisieren. Es handelt sich hierbei um einen gemeinsamen, transaktional konsistent gehaltenen Szenengraphen (im Folgenden weiter als transaktionaler Szenengraph bezeichnet). Hierfür wurden die Eigenschaften eines verteilten virtuellen Speicherkonzeptes (siehe Kapitel 1.3) mit einem transaktionalen Konsistenzmodell für die Datenhaltung (siehe Kapitel 1.4) kombiniert, wodurch neue Arten der Nutzungsmöglichkeit und des Einsatzes für einen Szenengraphen geboten werden.

Der für eine virtuelle 3D-Welt benötigte Szenengraph wird bei diesem Ansatz innerhalb eines verteilten virtuellen Speichers abgelegt. Dadurch, dass dieser Speicher von allen teilnehmenden Stationen gemeinsam genutzt werden kann, bewirkt dies eine einheitliche Sichtweise aller Rechner auf den eigentlichen Szenengraphenaufbau und die darin enthaltenen Szenengraphendaten. Der im VVS abgelegte Szenengraph erhält dadurch die Rolle eines gemeinsam nutzbaren und global gültigen Szenengraphens (siehe Abbildung 32).

Da die beteiligten Stationen aber nicht nur während des Lesens, sondern insbesondere auch beim Schreiben einen direkten Zugriff auf die gesamte Szenengraphenstruktur besitzen, ergibt sich ein gleichberechtigtes Nutzungsszenario für alle teilnehmenden Rechner. Es lässt sich so ein Client-Server-basierter Ansatz vermeiden, in dem eine servergestützte Verwaltung des Szenengraphens eine potentielle Engstelle im System darstellt. Die teilnehmenden Stationen innerhalb der Anwendung eines transaktionalen Szenengraphensystems entsprechen dadurch vielmehr gleichberechtigten Peers.

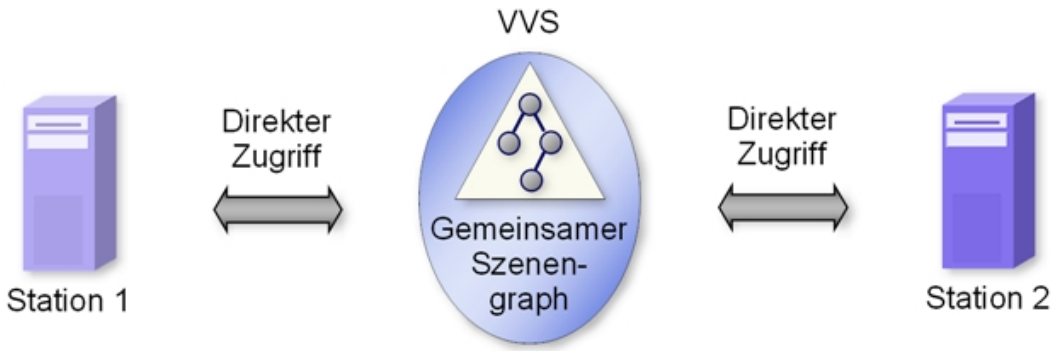

Abbildung 32: Ein gemeinsamer Szenengraph innerhalb eines verteilten, virtuellen Speichers

Durch das zugrundeliegende transaktionale Konsistenzmodell wird darüber hinaus sichergestellt, dass die von einer Station auf den Szenengraphendaten vorgenommenen Änderungen auch zugleich für alle anderen Stationen unmittelbar sichtbar werden. Ein normaler Lesezugriff auf diese Daten genügt, um automatisch die zu diesem Zeitpunkt jeweils aktuellste Version der Daten zu erhalten. Eine darauf zugreifende Anwendung muss somit keine zusätzlichen Schichten oder expliziten Mechanismen zur Synchronisierung und zum Datenabgleich zwischenschalten und aufrufen.

Im Gegensatz zu bestehenden Systemen, welche mit dem Konzept eines verteilten gemeinsamen Speichers arbeiten, ist es bei einem transaktional konsistent gehaltenen Szenengraphen nicht notwendig, ein relaxiertes Konsistenzmodell zu verwenden, um die Lauffähigkeit des Systems überhaupt erst zu ermöglichen bzw. Annahmen für die Datenhaltung treffen zu können. Durch die transaktionale Konsistenz wird einer verteilten Anwendung dauerhaft eine starke Konsistenzform als Standard zur Verfügung gestellt.

Diese angebotene, transaktionale Konsistenz kommt in der Form zum Ausdruck, dass Konflikte bei mehreren simultan ausgeführten Schreibvorgängen auf gemeinsame Daten transparent für die darauf zugreifende Anwendung aufgelöst werden. Eine Anwendung benötigt somit, um einen exklusiven Zugriff auf zu modifizierende Daten zu erhalten, keine zusätzlichen Mechanismen mehr, wie z.B. Sperren oder Semaphore oder gar ein Eingreifen des Benutzers. Dies sorgt im Endeffekt für ein verklemmungsfreies Zugriffsverhalten und führt zu einer deutlichen Vereinfachung der Arbeitsweise einer darauf aufbauenden Anwendung.

Der Einsatz eines transaktionalen Szenengraphens ermöglicht es des Weiteren allen teilnehmenden Stationen, gänzlich auf lokale Kopien des Szenengraphens zu verzichten. Dies ergibt sich aus der direkten Zugriffsmöglichkeit aller Stationen auf den gemeinsamen Szenengraphen. Da es jeder Station unmittelbar möglich ist, auf die Daten innerhalb des globalen Szenengraphens zuzugreifen, kann eine darauf laufende Anwendung jeweils auch erst beim eigentlichen Bedarf dieser Szenengraphendaten – beispielsweise während einer Renderroutine oder dem Methodenaufruf zu einer Kollisionsberechnung während einer Bewegung – auf diese zugreifen und anschließend weiterverwenden. Der wesentliche Unterschied zu den klassischen, bislang vorgestellten Verfahren liegt dabei im Zeitpunkt des Zugriffs auf die Szenengraphendaten. Dieser kann nun genau in dem Moment erfolgen, in welchem die jeweilige Anwendung diese Daten auch wirklich benötigt.

In bestehenden Client-Server-basierten Systemen hingegen ist in der Regel durch die Nachrichten- bzw. Update-basierte Art der Übermittlung von Aktualisierungsmeldungen der Gültigkeitszeitpunkt der empfangenen Daten für alle Klienten vom Server aus gegeben. In dem Moment, in dem die Klienten eine Aktualisierungsnachricht empfangen, müssen sie diese in einen lokalen Szenengraphen zur Zwischenspeicherung einpflegen, da der Zeitpunkt des Erhalts dieser Nachrichten normalerweise nicht mit dem eigentlichen Zeitpunkt des Bedarfs und der

Weiterverarbeitung bei den Klienten synchronisiert ist. Erst wenn von diesen wieder ein Prozess ausgeführt wird, der diese zwischengespeicherten Daten verwenden möchte – beispielsweise die zuvor erwähnte Renderroutine –, kann die eigentliche Verarbeitung mit diesen Daten stattfinden. Da sich die erhaltenen Updates aber normalerweise in ihrer Gültigkeit auf den Sendezeitpunkt des Servers beziehen, tritt beim Zugriff der Klienten auf diese Daten der Fall ein, dass dieser Zeitpunkt bereits vergangen ist bzw. beim Empfang der Daten schon vergangen war. Dies verstärkt sich insbesondere noch weiter, wenn bei einem Klienten gerade andere Prozesse ausgeführt werden, die zuerst abgearbeitet oder zumindest verdrängt werden müssen und bis zum endgültigen Zugriff die entstehenden Zeitdifferenzen weiter vergrößern. Somit bedeutet dies, dass die zuvor zwischengespeicherten Nachrichten nahezu ausnahmslos zum eigentlichen Zeitpunkt der Weiterverwendung von einer Anwendung entweder bereits veraltet sind oder zumindest eine große zu erwartende Abweichung zu den eigentlich aktuellen, aber nur beim Server vorgehaltenen, global gültigen Szenengraphenendaten (und somit indirekt auch zu den Zeitpunkten aller anderen Klienten) aufweisen, als dass dies für eine Anwendung ohne geeignete Gegenmaßnahmen stets tolerierbar wäre. Es muss daher vor dem eigentlichen Zugriff zusätzlicher Berechnungsaufwand investiert werden, um diesen zeitlichen Versatz zu korrigieren und damit den zwischengespeicherten Daten erneut Gültigkeit für ihre Weiterverarbeitung zu verleihen.

Um dies auszugleichen, wird daher in bestehenden Systemen über teilweise sehr aufwändige Verfahren wie der Koppelnavigation ('Dead Reckoning'; [37], [38], [39]) oder durch Verzögerungsausgleiche ('Lag Compensation'; [40]) versucht, diese vorhandenen Zeitdifferenzen zu beheben und die Synchronisierung der Daten effizienter umzusetzen. Hierfür werden von den Klienten hauptsächlich Extrapolationen anhand der vorliegenden, lokalen Daten über mehrere Zeitschritte aus den letzten Aktualisierungsnachrichten hinweg vorgenommen, um so Voraussagen über die wahrscheinlich momentan aktuellen Werte dieser Daten treffen zu können. Mittlerweile gibt es daher zu diesem Gebiet bereits eine Fülle an unterschiedlichen, eingesetzten Konzepten und Varianten (z.B. 'Time Warp' [41], [42]; 'Bucket Synchronization' [43]; 'Trailing State Synchronization' [44], ...).

Durch den Einsatz eines transaktionalen Szenengraphens kann auf diesen gesamten Aufwand sowohl auf Serverseite (Verwalten von Klientenlisten, Zusammenstellen der inkrementellen Updates, Versenden dieser Updates an die Klienten, ...) als auch auf Klientenseite (Empfang der Updates, Einpflegen in den lokalen Szenengraphen, Extrapolation der Daten beim Zugriff, Driftkorrektur, ...) verzichtet werden. Da die Klienten den oben beschriebenen direkten Zugriff auf die gemeinsamen Szenengraphendaten besitzen, können sie diesen Zugriff auch erst zu genau demjenigen Zeitpunkt durchführen, in dem sie diese Daten benötigen. In Folge dessen ergibt sich im Gegensatz zur bisherigen, klassischen Update-Semantik eine neue, „bedarfsgesteuerte“ Semantik für die Arbeit mit einem Szenengraphensystem. Zusätzlich kann ein Entwickler bzw. eine Anwendung hierbei die gemeinsamen Szenengraphendaten derart nutzen, als ob sie auf jeder Station lokal vorhanden wären, benötigt dafür selbst aber keine lokalen und expliziten Zwischenspeicherungen mehr.

Allerdings schließt die Verwendung eines transaktionalen Szenengraphens umgekehrt den Einsatz eines privaten bzw. lokalen Szenengraphens auf den einzelnen Stationen nicht per se aus. Falls es für bestimmte Einsatzzwecke für sinnvoll erachtet wird, eine solche lokale Kopie einzusetzen – beispielsweise für spezielle Optimierungen –, so können diese auch nach wie vor verwendet werden. Zwar ist dies in der Regel bei der Nutzung eines transaktionalen Szenengraphens nicht nötig; für einen Entwickler bleibt dadurch aber ein Maximum an Entscheidungsfreiheit für die Gestaltung seiner Anwendung bestehen. Insbesondere kann über diese Möglichkeit auch eine Portierung bestehender Szenengraphensysteme hin zu einem transaktionalen Szenengraphensystem

einfacher vorgenommen werden, da eine Kompatibilität zu den ursprünglichen Strukturen und Vorgehensweisen durch Nachbildung der bisherigen Strukturen im neuen, transaktionalen Kontext (einschließlich der gewohnten, lokalen Kopien) gewahrt werden kann.

Für den Einsatz eines transaktionalen Szenengraphensystems kann prinzipiell jedes beliebige Betriebssystem verwendet werden. Bei der Nutzung traditioneller (Einzelplatz-)Betriebssysteme hierfür muss allerdings berücksichtigt werden, dass diese für gewöhnlich keine Verteilung von Daten anbieten bzw. kein verteiltes Speicherkonzept verwenden. Folglich ist es erforderlich, durch den Einsatz von zusätzlicher Middleware oder durch eingezogene Zwischenschichten mittels spezieller Bibliotheken (wie z.B. bei Repo-3D; siehe Kapitel 3.2.3.2) eine für die Anwendung einheitliche Sichtweise eines verteilten virtuellen Speichers zu erreichen. Besonderes beim Zusammenspiel von unterschiedlichen Komponenten oder Bibliotheken lassen sich die (verteilten) Zugriffsmechanismen dieser zwischengeschalteten Schichten aber zumeist nicht komplett und transparent vor den höheren Schichten verbergen, was mit einer Anpassung des Programmcodes bei einer Portierung bestehender Anwendungen berücksichtigt werden muss. Daher empfiehlt sich für den Einsatz eines transaktionalen Szenengraphensystems generell auch die Verwendung eines verteilten Betriebssystems, welches auf Systemebene bereits ein verteiltes Speicherkonzept zur Verfügung stellt (beispielsweise IVY [45], TreadMarks [46], Kerrighed [47], ...). Für die zusätzlich benötigte transaktionale Konsistenz gilt dies analog. Eine Realisierung mittels einer erneut eingezogenen Zwischenschicht hebt die Komplexität eines dadurch erweiterten Systems noch weiter an. Dahingegen stellt ein von Grund auf transaktional arbeitendes, verteiltes Betriebssystem – wie beispielsweise Plurix (siehe Kapitel 1.5) – eine ideale Ausgangsbasis dar, um ein solches Szenengraphensystem mit nur geringem Zusatzaufwand darauf aufsetzen zu können.

3.4 Anwendungsgebiete transaktionaler Szenengraphen

Durch den Einsatz eines transaktionalen Szenengraphens ergeben sich aus Anwendungssicht etliche Vorzüge im Vergleich zu den traditionellen Möglichkeiten bestehender Szenengraphensysteme, die dort bislang nur durch teilweise aufwändige Lösungen und Mechanismen bereitgestellt werden können. Auf diese Merkmale wird nachfolgend genauer eingegangen.

3.4.1 Intuitive Entwicklung verteilter Anwendungen

Als einer der ersten erkennbaren Vorzüge bei der Verwendung eines transaktionalen Szenengraphens zeigt sich eine wesentliche Vereinfachung sowohl im Entwicklungsaufwand von verteilten, darauf aufsetzenden Anwendungen bereits während ihrer Erstellung, als auch bei deren Arbeitsweisen im weiteren Einsatz. Durch die Verwendung eines transaktionalen Szenengraphens wird ein intuitiver Umgang mit den zugrundeliegenden Daten ermöglicht. Indem die Daten im Szenengraphen zur Verfügung gestellt werden und alle Stationen direkten Zugriff darauf besitzen, wird einer Anwendung ein Nutzungsverhalten mit diesen Daten wie in einem rein lokalen Fall ermöglicht. Gleichzeitig wird eine automatische Verteilung dieser Daten vorgenommen, ohne dass von der Anwendung bzw. dem Entwickler explizite Aufrufe an das Betriebssystem oder andere Mechanismen hierfür eingesetzt werden müssen. Die Übermittlung der Daten beim Zugriff auf entfernte Stationen wird transparent für die Anwendung vom Betriebssystem bzw. durch den verteilten, virtuellen Speicher vorgenommen und die Korrektheit bzw. Gültigkeit dieser Daten wird durch die transaktionale Konsistenz sichergestellt. Hierdurch lassen sich sowohl die Entwicklung von aufwändigen Kommunikationsmechanismen zur Verteilung als auch expliziter Synchronisierungsalgorithmen zur Gänze einsparen. Ein Entwickler kann sich stattdessen auf die Erstellung der eigentlichen Anwendung und deren Arbeitsweise konzentrieren, ohne durch

fehleranfällige Nebenaufgaben zusätzlich belastet zu werden. Dies vereinfacht und beschleunigt eine auf einen transaktionalen Szenengraphen aufsetzende Anwendungsentwicklung erheblich.

3.4.2 Dediziertes Rechnen

Ein weiterer Vorteil, welcher sich durch die Nutzung eines transaktionalen Szenengraphens ergibt, ist die Möglichkeit, einzelne Aufgaben von ausgewählten Stationen vornehmen zu lassen, ohne dass hierfür eine klassische und starre Rollenaufteilung in Klienten und Server wie in vergleichbaren Systemen vorgenommen werden muss.

3.4.2.1 Ergebnisbereitstellung

In bestehenden Systemen mit einem traditionellen Client-Server-basierten Szenario wird üblicherweise eine Arbeitsaufteilung durch einen zentralen Server vorgenommen (oder statisch durch einen Entwickler festgelegt), mit der bestimmt wird, welche Berechnungen von ihm selbst durchgeführt werden bzw. welche Aufgaben er weiter an die Klienten delegieren kann[21]. Im günstigsten Fall kann der Server alle Berechnungen selbst übernehmen und leitet nur die Ergebnisse an die Klienten weiter. So werden Inkonsistenzen vermieden und Berechnungen nicht unnötig mehrfach durchgeführt. Typischerweise scheitert dieses Vorgehen aber schnell an der Leistungsgrenze des Servers, so dass dieser gezwungen ist, zumindest einen Teil der anstehenden Berechnungen an die Klienten abzugeben. Oftmals nimmt er selbst sogar keine Berechnungen mehr vor, sondern stellt nur noch Daten bereit und lässt alle Aufgaben von den Klienten bewältigen.

Die Klienten erhalten in diesen Fällen nur noch Aufforderungsnachrichten vom Server und verwenden ihre eigene vorhandene Rechenleistung dazu, die von ihnen benötigten Berechnungen – wie z.B. eine Kollisionserkennung des eigenen Avatars mit anderen Avataren oder Objekten in der virtuellen Umgebung – selbständig vorzunehmen. Diese Vorgehensweise ist prinzipiell zwar für die von einem Rechner lokal benötigten Berechnungen sinnvoll, allerdings müssen die erhaltenen Ergebnisse trotzdem mit den Zuständen anderer Klienten abgeglichen werden, da neue Situationen entstanden sein könnten, die auch Einfluss auf die anderen Klienten haben. In dem genannten Beispiel kann sich der eigene Avatar weiterbewegt haben, was für die anderen Klienten wiederum wichtig zu wissen ist, um ihrerseits eine korrekte Kollisionserkennung durchführen zu können[22]. Dieser Abgleich oder zumindest die Bereitstellung der berechneten Ergebnisse muss wiederum vom Server durchgeführt werden, da in der Regel keine Kommunikation zwischen den Klienten

21 Wo hierbei die Grenze bei der Aufteilung zwischen Server und Klient gezogen wird, hängt von einer Vielzahl unterschiedlicher Faktoren und von der jeweiligen Situation ab, wie z.B. von der Leistungsfähigkeit der zur Verfügung stehenden Hardware, von der Netzwerkanbindung, aber auch von der Arbeitsweise und vom Kommunikationsaufwand der Anwendung. Ein Entwickler muss daher individuell für die jeweilige Anwendung eine optimale Aufteilung zwischen den Arbeiten, welche vom Server übernommen werden können bzw. übernommen werden müssen und den Aufgaben, welche auch von den Klienten selbst durchgeführt werden können, finden.
In Online-Welten und -Spielen gibt es daher beispielsweise eine Obergrenze in der Anzahl der maximalen Nutzer, welche von einem Server bedient werden können. Dies lässt umgekehrt Rückschlüsse über den von einem Server zu erledigenden Aufwand bzw. die zugrundeliegende Serverkapazität zu. Die genaue Zahl bleibt aber zumeist ein Firmengeheimnis. So können bei kommerziellen Anbietern einige 10.000 Nutzer zusammengeschlossen sein ('Eve Online') oder es wird schon deutlich früher eine Aufteilung vorgenommen ('World of Warcraft' macht dies ab ca. 2500 Nutzer; 'Second Life' bereits ab ca. 70 Nutzer; siehe Kapitel 2.3.2.2).
Überschreiten die Klientenzahlen diese Maximalwerte, so werden potentielle Engpässe durch verschiedene Methoden vorgebeugt oder behoben: sei es durch simples Beschränken des Zutritts weiterer Teilnehmer für das betroffene Gebiet oder ein Aufteilen vorhandener Gebiete auf verschiedene, kleinere „Welten" oder „Regionen", mit jeweils eigenen Servern. Die eingesetzten Server sollten dabei für die jeweilige „Welt" und deren Objekte, die sie steuern, weitere Kapazitätsreserven vorhalten, um auch bei Volllast flüssige Abläufe zu garantieren (wobei bei einigen Programmen durch erkennbare Ruckler oder Aussetzer bis hin zur Unbenutzbarkeit während Stoßzeiten ein gegenteiliges Verhalten erkennbar wird).

stattfindet und somit auch kein gegenseitiger Ergebnisaustausch vorgenommen wird[23].

Aus Serversicht betrachtet entspricht dieses Vorgehen einer Verschiebung von benötigten (Server-)CPU-Ressourcen hin zu einem erhöhten, stattdessen von ihm zu bewältigenden Kommunikationsaufwand für die anfallende Ergebnissynchronisierung. Die hierfür benötigte Kommunikationsbandbreite ist aber heutzutage typischerweise leichter zu handhaben und zu erweitern, als eine Erhöhung der Rechenkapazität. Auch lässt sich die Zielsetzung des Servers, mehr Klienten bei gleichen oder niedrigeren (Betriebs-)Kosten bedienen zu können, dadurch einfacher erreichen. Insbesondere falls der Anbieter einer Serverdienstleistung nicht selbst die zugehörigen Klienten und deren Hardwareausstattung betreut (wie es bei allen kommerziellen Online-Spielen und -Welten der Fall ist), wird dieses Vorgehen standardmäßig eingesetzt, um so die eigene (Server-)Infrastruktur klein und die Kosten dafür niedrig halten zu können, da eventuelle Mehr- bzw. Aufrüstungskosten auf Klientenseite nicht von der Anbieterfirma aufgewendet werden müssen.

Für die Klienten bedeutet dies hingegen, dass nun jeder Klient zusätzliche Berechnungen vorzunehmen hat. Dies betrifft dabei nicht nur die erwähnten, lokal von einzelnen Stationen benötigten Berechnungen, sondern auch sämtliche global gültigen Operationen, wie beispielsweise die Berechnung aller sichtbaren, darzustellenden Animationen. Somit nehmen dann alle Klienten dieselben Berechnungen parallel zueinander vor. Sollte ein Klient nicht (mehr) in der Lage sein, dieses Arbeitspensum zu bewältigen, so liegt es im Verantwortungsbereich des Benutzers, die hierfür benötigten Ressourcen aufzurüsten. Andernfalls kann er nur mit Einschränkung bzw. im ungünstigsten Fall gar nicht mehr an der virtuellen Welt teilnehmen.

Ein Server wird durch dieses Vorgehen entlastet; global gesehen steigt aber der verursachte Rechenaufwand stark an, da nun alle Klienten die benötigten (z.B. Animations-) Berechnungen zusammen n-fach redundant ausführen. Eine Serverkommunikation zur Synchronisierung und zum Ausgleich möglicher auftretender Abweichungen, Drifts oder Inkonsistenzen zwischen den einzelnen Klientenberechnungen durch unterschiedliche Berechnungsgeschwindigkeiten oder -genauigkeiten erfordert zusätzlichen Aufwand, da nur wenige Anwendungsfälle abweichendes Verhalten der Klienten untereinander tolerieren. Gerade in solchen Situationen wäre aber ein zentraler Server geeigneter, um die notwendigen Daten einmalig selbst zu berechnen und anschließend den Klienten lediglich die Ergebnisse zu liefern.

Bei der Verwendung eines transaktionalen Szenengraphens hingegen kann nun eine Station selbst diejenigen Berechnungen durchführen, die sie benötigt; kann deren Ergebnisse aber anschließend direkt für alle anderen Stationen sichtbar werden lassen, indem sie diese in den gemeinsamen Szenengraphen einfügt (siehe Abbildung 33). Hierdurch lassen sich auf einfache Art und Weise Mehrfachberechnungen vermeiden, da eine anstehende Berechnung jeweils nur von einer Station ausgeführt werden muss und alle anderen Stationen anschließend direkt von den bereitgestellten Ergebnissen profitieren können. Zusätzlich wird keine zentrale Instanz mehr benötigt, um eine Synchronisierung bzw. Koordinierung dieser Ergebnisse durchzuführen.

22 Einige Programme und (Online-)Spiele vereinfachen diesen Sachverhalt erheblich zugunsten einer größeren Leistungsfähigkeit und geringeren Latenzanforderungen. 'World of Warcraft' beispielsweise verzichtet vollkommen auf die gegenseitige Kollisionserkennung zwischen Avataren oder Monstern und preist dies sogar als Vorteil an, dass sich Avatare gegenseitig nicht stören oder behindern können.

23 Dies entspräche dann auch nicht mehr der für gewöhnlich von Entwicklern standardmäßig eingesetzten, klassischen Client-Server-Architektur, sondern einem Hybridsystem mit Peer-to-Peer-Techniken, welches eigene Anpassungen und Erweiterungen des Entwicklers nach sich ziehen würde, da hierzu kaum bestehende Bibliotheken existieren.

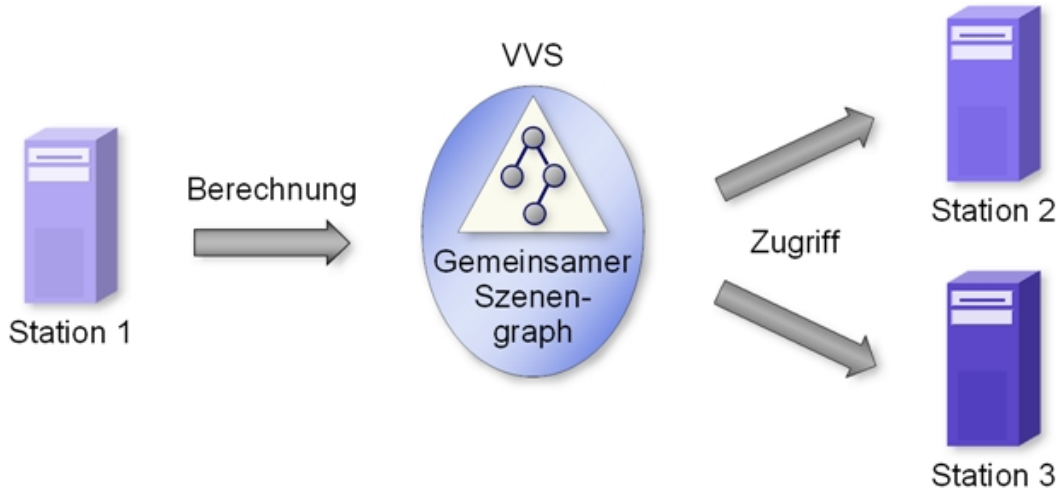

Abbildung 33: Ergebnisbereitstellung mittels eines VVS

So reicht es völlig aus, wenn in dem erwähnten Beispiel einer Animationsberechnung eine einzige, beliebige Station diese Berechnung vornimmt und als Ergebnis die neuen Positionen der animierten Gegenstände und Objekte im transaktionalen Szenengraphen anpasst. Alle anderen Stationen können dann diese aktualisierten Daten unmittelbar auslesen und weiterverwenden, ohne eigene Berechnungen vornehmen zu müssen, da sie ebenfalls einen direkten Zugriff auf das Animationsobjekt besitzen. Dies spart global Rechenzeit und Ressourcen ein und dadurch freigewordene Ressourcen können für andere Vorgänge genutzt werden. So lässt sich etwa die Anzahl der insgesamt vorhandenen Animationen erhöhen, um so ein vielfältigeres Erscheinungsbild der virtuellen Welt zu erreichen, da nun von den lediglich abrufenden Stationen ihrerseits weitere Objekte animiert werden können. Im besten Fall kann so durch einen transaktionalen Szenengraphen eine mögliche Skalierung des Gesamtsystems vom klassischen Client-Server-Fall mit nur einer Animationsberechnung auf allen n Rechnern hin zur Berechnung von n Animationen auf n Stationen ermöglicht werden[24].

Ein weiterer Unterschied zur klassischen Client-Server-Architektur besteht in der Art der Vermittlung dieser Ergebnisse. Da dies automatisch und transparent durch den transaktionalen Szenengraphen vorgenommen wird, werden auch keine zusätzlichen Synchronisierungsmaßnahmen mehr benötigt; weder mittels eines zentralen Servers noch mittels eines den (animierten) Objekten zugeordneten Sequencers (wie u.a. bei Distributed Open Inventor, siehe Kapitel 3.2.3).

3.4.2.2 Dedizierte Stationen

Standardmäßig bleibt es einem Anwender bei der Nutzung eines transaktionalen Szenengraphens verborgen, auf welcher teilnehmenden Station eine bestimmte Berechnung ausgeführt wird; in der Regel wird dies allerdings von derjenigen Station vorgenommen, welche eine Berechnung anstößt. Beispielsweise wird eine aufgerufene Kollisionserkennung während der Bewegung des eigenen Avatars auch lokal von der eigenen Station durchgeführt. Diese Berechnung wird dabei für gewöhnlich durch den Aufruf von Subroutinen oder eines anderen Prozesses erledigt. Die eigentliche Szenengraphenanwendung ist dabei aber letztlich nur am Ergebnis dieser Berechnung interessiert; von welcher Stelle aus dieses Ergebnis geliefert wird bzw. wer die Subroutine ruft oder den zugehörigen Prozess abarbeitet, ist nur von untergeordneter Bedeutung. Für die Anwendung

24 Bei vergleichbaren Animationen bzw. Rechenaufwand für die Animationsberechnung und unter Vernachlässigung des jeweils benötigten Kommunikations- oder Synchronisierungsaufwandes.

macht es somit auch keinen Unterschied, falls diese Berechnung nicht von der lokalen Station selbst, sondern eventuell von einer zweiten, separaten Station durchgeführt wird. Dies kann dann sinnvoll sein, wenn diese zweite Station gerade weniger ausgelastet ist und daher freie Rechenzeit zur Verfügung hat oder durch vorhandene Spezialhardware auf bestimmte Berechnungen spezialisiert ist. Die Anwendung benötigt nur einen Zugriff auf das Resultat der Berechnung, um damit weiterarbeiten zu können, und dies wird über den gemeinsamen Szenengraphen gewährleistet. Somit bedeutet dies, dass Berechnungen prinzipiell von jeder teilnehmenden Station aus geschehen können[25].

Dieses Konzept lässt sich weiter derart verallgemeinern, dass es hierdurch möglich wird, alle Arten von Berechnungen von beliebigen anderen Stationen vornehmen zu lassen, welche dann anschließend ihre Ergebnisse wiederum über den transaktionalen Szenengraphen allen zugänglich machen können. Die rechnenden Stationen müssen dabei nicht einmal selbst an der virtuellen Welt bzw. dem Präsenzsystem teilnehmen; es reicht völlig aus, wenn sie Zugriff auf den transaktionalen Szenengraphen besitzen, über den sie ihre Berechnungsergebnisse einbringen können (siehe Abbildung 34).

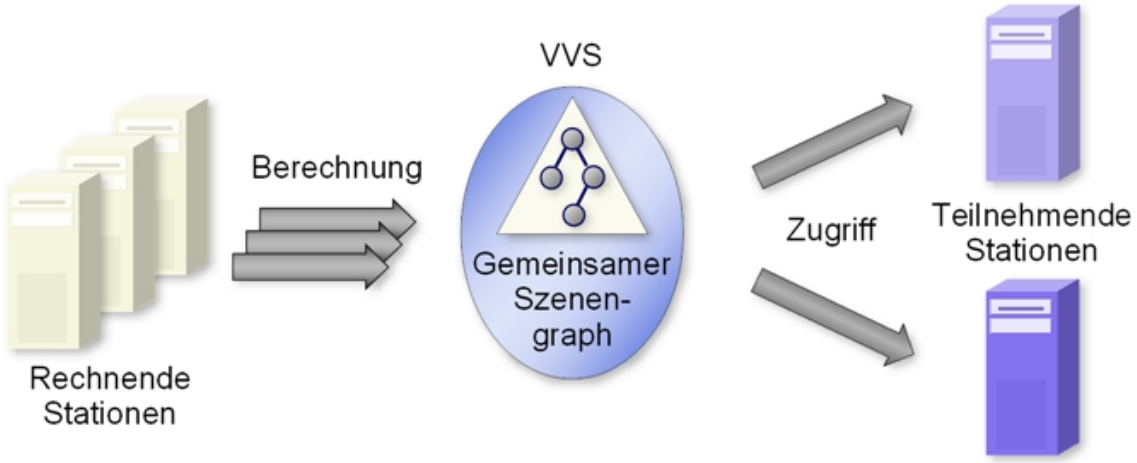

Abbildung 34: Dedizierte Berechnungen von separaten Stationen

Die eigentlich teilnehmenden Stationen können wie zuvor durch ihren Zugriff auf den gemeinsamen Szenengraphen die Ergebnisse aufgreifen und damit weiterarbeiten, als ob sie die hierfür nötigen Berechnungen selbst durchgeführt hätten. So lassen sich auch Mischformen zwischen teilnehmenden und dediziert rechnenden Stationen einfach realisieren oder sogar eine komplette Entkopplung zwischen rechnenden Stationen und darstellenden Stationen vornehmen.

Durch diesen Ansatz wird einem Szenengraphensystem insbesondere eine einfache Skalierbarkeit ermöglicht. So lässt sich die angebotene Vielfalt innerhalb einer virtuellen Welt weiter vergrößern, als es von den einzelnen Stationen bei eigener Berechnungen aller notwendigen Daten realisierbar wäre. Falls die rechnenden Stationen nicht mehr ausreichend schnell genug die anfallenden Aufgaben bewältigen können, so können dem Gesamtsystem einfach ein oder mehrere zusätzliche (dedizierte) Stationen hinzugefügt werden, um diese zusätzliche Last abzufangen.

Auf diese Art und Weise lassen sich selbst aufwändige Berechnungen, wie komplette Simulationen komplexer Sachverhalte oder besonders rechenintensive Arbeiten, wie z.B. künstliche, durch neuronale Netze gesteuerte Verhaltensweisen von Avataren (sogenannte 'Soft-Bots'), problemlos in

25 Wie dies vorgenommen werden kann, wird im nachfolgenden Unterkapitel 3.4.2.3 dargelegt.

ein Präsenzsystem miteinbeziehen [48]. Analog kann so auch der eigentliche Darstellungsvorgang der virtuellen Umgebung durch eine Verknüpfung mit weiteren Verfahren ([49], [w46]) auf aufwändigere Varianten wie z.B. Echtzeit-Raytracing ausgedehnt werden.

Ein weiteres Beispiel stellt ähnlich zu den Animationsberechnungen die Arbeit einer Physik-Engine dar, welche sich um physikalisch korrektes Verhalten der Objekte innerhalb der virtuellen Welt kümmert (Beschleunigen von Objekten durch Gravitation, Impulsverhalten beim Zusammenstoß von Objekten, ...). Auch hier reicht es aus, wenn diese Aufgabe einmalig von einer Station berechnet wird, und nur die Resultate im Szenengraphen Eingang finden, um allen teilnehmenden Stationen ein physikalisches korrektes Verhalten ihrer Objekte zu ermöglichen, anstatt dass jeder Klient eine eigene Physik-Engine benötigt. Durch einen modularen Aufbau der Physik-Engine können die vorzunehmenden Einzelberechnungen sogar auf mehrere teilnehmende Stationen oder dedizierte Rechner aufgeteilt werden, um so den Aufwand einer einzelnen Station noch weiter zu reduzieren [50].

3.4.2.3 Load Balancing

Bei einer Client-Server-basierten Architektur sind in der Regel die vom Server durchgeführten Berechnungen auch fest an den Server gebunden bzw. bei mehreren Servern fest einem einzelnen Server zugewiesen. Sollte der Server durch viele Anfragen oder zusätzliche Berechnungen ausgelastet sein, so kann es zu Verzögerungen bei der Abarbeitung seiner Berechnungen kommen, obwohl anderen Servern eventuell noch freie Ressourcen zur Verfügung stehen. Ohne den Einsatz von weitergehenden, speziellen Mechanismen zur Lastverteilung wird ein entstehendes Ungleichgewicht bei der Arbeitsaufteilung zwischen den Servern nicht automatisch ausgeglichen.

Im Falle eines transaktionalen Szenengraphens ist hingegen durch die damit verbundene Peer-to-Peer-basierte Sichtweise die Berechnung einer bestimmten Aufgabe nicht zwangsläufig fest einer Station zugeordnet. Sobald eine Aufgabe zur Bewältigung ansteht, kann vielmehr eine beliebige Station, welche gerade freie Ressourcen besitzt, diese Aufgabe aufgreifen und übernehmen. Es kann beispielsweise ein eigener Abschnitt im transaktionalen Szenengraphen als „Bearbeitungspool“ für anstehenden Berechnungen dienen, aus der eine Station automatisch die nächste auszuführende Berechnung auswählen kann. Durch das Transaktionskonzept wird hierbei auch eine mehrfache Abarbeitung auf unterschiedlichen Stationen bei simultanem Zugriff auf diesen Bearbeitungspool vermieden, da eine einmal entnommene Aufgabe ab diesem Zeitpunkt für die anderen Stationen nicht mehr sichtbar ist. Diese Arbeitsaufteilung geschieht somit vollständig dynamisch während der Laufzeit und ohne dass spezielle Lastverteilungsalgorithmen oder steuernde Überwachungsinstanzen verwendet werden müssen.

Je nach Auslastungsverhalten der einzelnen Stationen können diese Berechnungen dabei auch automatisch zwischen den Stationen migrieren. Sobald eine Station durch zu hohe Last all ihre Aufgaben nicht mehr schnell genug erledigen kann, so fügt sie ein oder mehrere davon wieder in den Bearbeitungspool zurück, aus dem sie erneut von einer anderen Station aufgegriffen werden. Durch den direkten Zugriff auf den gemeinsamen Szenengraphen von allen Stationen aus können diese auch die darin enthaltenen Daten so ansprechen, als ob sie bereits lokal vorliegen. Hierdurch entfällt beim Migrieren einer Transaktion ein explizites „ein-” und „auspacken” der zugehörigen Daten ('Marshalling/Unmarshalling') komplett, wie es ansonsten als klassische Vorgehensweise von bestehenden Systemen durchgeführt wird (beispielsweise in Verbindung mit entfernten Methodenaufrufen wie 'RPC' oder 'RMI').

3.4.2.4 Heterogene Hardware

Die vorgestellten Konzepte eines transaktionalen Szenengraphen helfen aber auch umgekehrt stark heterogenen Hardwareumgebungen und Geräten mit wenig leistungsfähiger Ausstattung, an einem Präsenzsystem teilzunehmen. Dies können sowohl ältere Rechnersysteme sein, als auch mobile Endgeräte, wie PDAs und Handys, welche ansonsten durch ihre geringe Hardwareausstattung in einem klassischen Szenario nicht in einer virtuellen Umgebung partizipieren können. Insbesondere ist es durch diesen Ansatz nicht länger notwendig, die Berechnungslogik auf allen Stationen vorhalten zu müssen. Durch den Zugriff auf den gemeinsamen Szenengraphen können sie gleichermaßen alle für sie relevanten Daten erhalten. Die Fähigkeit, selbst (lokale) Berechnungen anhand der Szenengraphendaten vornehmen zu müssen, ist dadurch von diesen Geräten nicht mehr zwingend erforderlich; dies kann analog wie zuvor beschrieben von anderen, leistungsfähigeren Stationen übernommen und über den transaktionalen Szenengraph wiederum als Ergebnis zur Verfügung gestellt werden (siehe Abbildung 35).

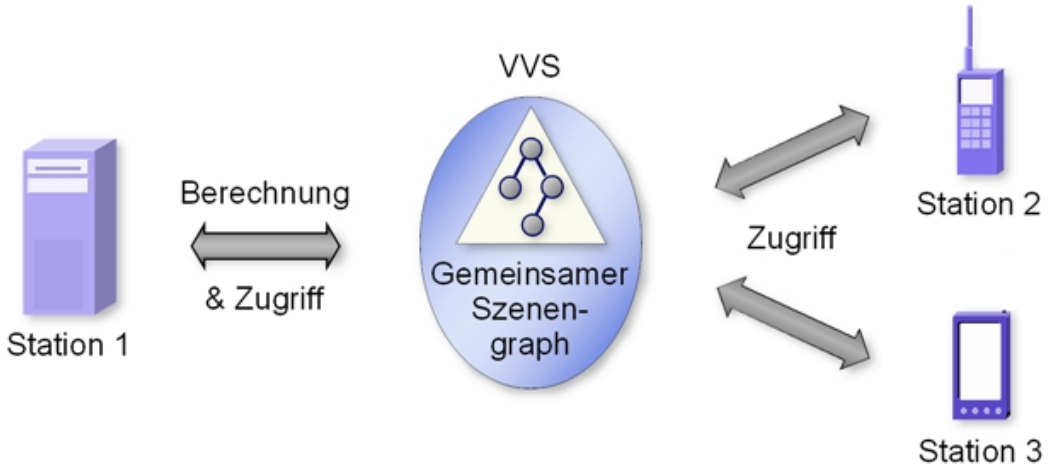

Abbildung 35: Mögliche Teilnahme verschiedenster, auch mobiler Endgeräte

Solche „kleinen“ oder älteren Geräte müssen lediglich eine Mindestanforderung bezüglich ihrer Kommunikationsfähigkeit und der grafischen Leistung erfüllen, um auf die vom Szenengraphensystem vorgehaltenen Daten zugreifen und diese auch auf dem jeweiligen Gerät darstellen zu können[26].

3.4.3 Multi-View-Systeme

Ein transaktionales Szenengraphensystem kann aber nicht nur wie zuvor beschrieben die Bereitstellung von Ergebnissen durchgeführter Berechnungen auf verschiedene Stationen aufteilen, sondern auch deren Darstellung, d.h. den Rendervorgang der virtuellen Welt anhand dieser Szenengraphendaten.

Im Normalfall wird die Perspektive auf eine virtuelle 3D-Welt individuell von jeder einzelnen Station durch den Benutzer bzw. über dessen zugehörigen Avatar gesteuert und dann nur lokal auf dieser Station dargestellt. Es ist aber auch umgekehrt möglich, die lokale Perspektive eines fremden Avatars gleichermaßen mittels des transaktionalen Szenengraphens anderen Stationen mitzuteilen.

26 Mittlerweile besitzen selbst aktuelle Handys integrierte 3D-Grafikchips, so dass auch deren Leistungsfähigkeit inzwischen ausreicht, um sie in ein transaktionales, virtuelles Präsenzsystem integrieren zu können.

Dies ermöglicht, dass die ursprünglich rein lokale Sichtweise eines Avatars (siehe Abbildung 36, links) auch von mehreren Stationen gemeinsam oder ergänzend dargestellt werden kann. Die ursprüngliche Station kann so die Renderengines der anderen Stationen (fern-)steuern (siehe Abbildung 36, Mitte und rechts), ohne dass hierfür explizite Steuerbefehle an die anderen Stationen gesendet werden müssen. Die Renderengines der zusätzlichen Stationen werden hierfür lediglich alle mit dem Avatar der steuernden Station verbunden und übernehmen dann dessen Perspektive genauso, als ob sie von einem eigenen Avatar gesteuert würden.

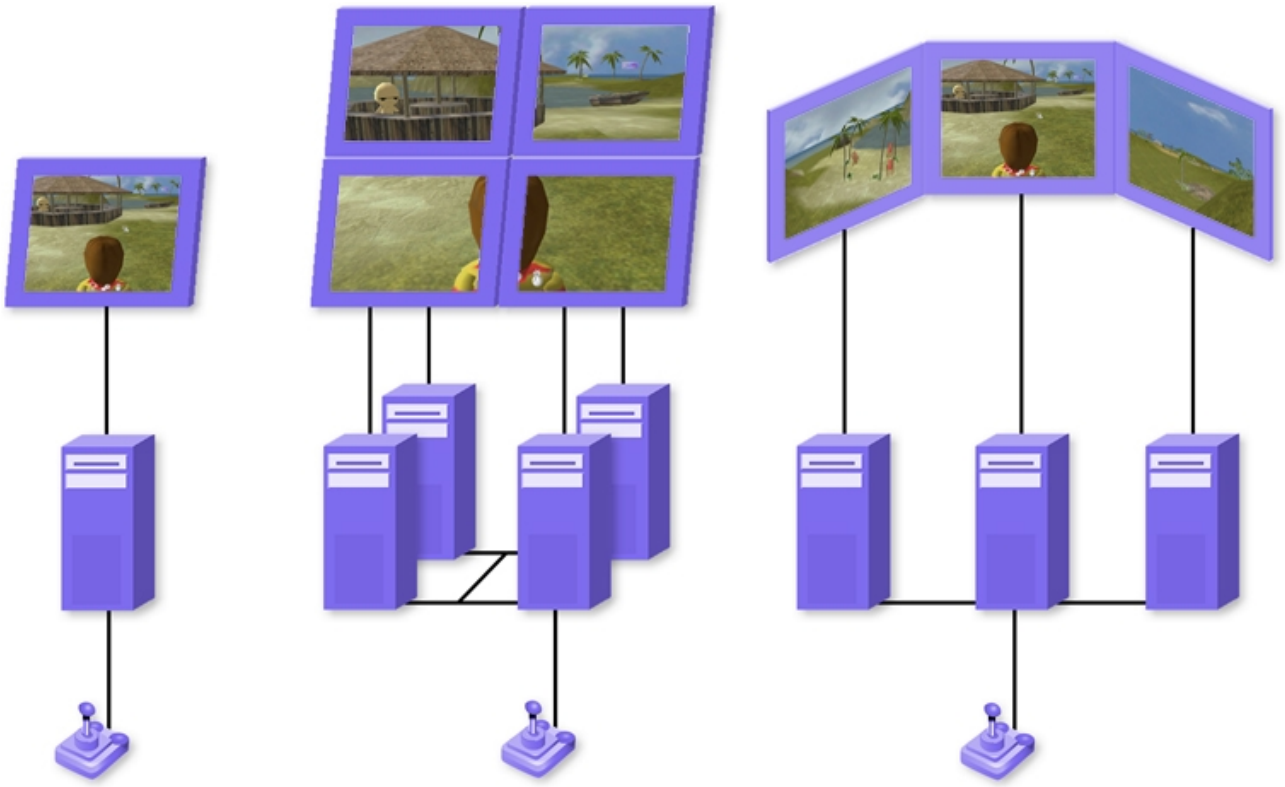

Abbildung 36: Einfacher Aufbau von Mehrschirm-Systemen ohne Spezialhardware

Insbesondere für Arbeiten im Mehrschirmbetrieb ist diese Vorgehensweise von Interesse. Zwar bieten moderne Grafikkarten bereits die Ansteuerung von zwei (vereinzelt, wie die Matrox 'Parhelia', auch drei) Monitoren parallel an, beliebige Erweiterungen darüber hinaus lassen sich so aber nicht vornehmen. Wo für einen solchen Fall bislang teure oder aufwändige Spezialhardware benötigt wird – wie z.B. beim Einsatz eines CAVE-Systems [w47] oder einer Powerwall [w48] –, bietet sich hier die Möglichkeit, stattdessen doch normale Standardhardware zu verwenden. Pro gewünschtem oder benötigtem, zusätzlichem Monitor wird eine weitere Station hinzugefügt oder sogar nur eine bislang nicht am Präsenzsystem teilnehmende Station mit zum Rendern einbezogen. Auf diese Art und Weise lassen sich elegant und kostengünstig größere Projektionen oder ein beliebiger Mehrschirmbetrieb mit Standarkomponenten aufbauen, ohne dass hierfür eine aufwändige Anpassung der bestehenden Software wie in vergleichbaren Arbeiten nötig ist [51].

3.4.4 Explizite Synchronisierung

Der Einsatz eines transaktionalen Szenengraphensystems benötigt konzeptuell keine zusätzlichen Mechanismen, um den Abgleich von Daten oder Abläufen zu synchronisieren. Dies geschieht transparent durch das integrierte, transaktionale Konsistenzmodell. Sollten von einem Entwickler oder einer Anwendung dennoch explizite Synchronisierungsmechanismen gewünscht oder benötigt werden, so ist deren Einsatz aber auch bei einem transaktionalen Szenengraphensystem ohne weiteres möglich.

Speziell für den Rendervorgang einer virtuellen Welt kann dies vorteilhaft oder eventuell sogar erforderlich sein. In der Regel arbeiten zwar alle Stationen mit gemeinsamen und einheitlichen Daten, die Darstellung geschieht aber typischerweise autonom und lokal durch jede Station selbst (vgl. Kapitel 3.4.3). Dies bedeutet im Speziellen, dass jede Station die virtuelle Umgebung individuell mit einer von der jeweilig vorhandenen Hardware bzw. Grafikkarte abhängigen Geschwindigkeit darstellt. Hierdurch können sich unterschiedliche Bildwiederholraten zwischen den einzelnen Stationen ergeben. Gerade bei der Nutzung von speziellen Multi-View-Systemen kann dies aber eventuell nicht mehr genügen. In der Regel werden unterschiedliche Bildwiederholraten zwar vom Nutzer eines Mehrschirmbetriebs nicht bemerkt, solange alle Stationen flüssige Bildwiederholraten liefern; wird aber Spezialhardware hierfür eingesetzt, welche bildsynchrone Eingangssignale voraussetzen – beispielsweise ein klassisches CAVE-System –, so kann dies nur durch explizite Synchronisierungsmechanismen realisiert werden.

Die Verwendung solcher Synchronisierungsmechanismen ist auch auf andere Einsatzgebiete übertragbar; nachfolgend werden diese Mechanismen aber anhand des erwähnten Rendervorgangs aufgezeigt, da dies eine der häufigsten Einsatzzwecke hierfür darstellt und sich die genutzten Konzepte dadurch anschaulich beschreiben lassen.

3.4.4.1 Direktes Rendering

Der Darstellungs- oder Rendervorgang (kurz: das Rendering) ist der eigentliche Ablauf zum Visualisieren einer virtuellen Umgebung. Er wird von einer sogenannten Renderengine vorgenommen und in der Regel lokal auf demjenigen Rechner ausgeführt, der an der virtuellen Welt teilnimmt und diese auf seinem Bildschirm darstellen möchte. Für das Rendering einer Szene sind hierfür mehrere Schritte notwendig. Zu Beginn steht dabei eine Traversierung des Szenengraphens. Hierbei wird der Szenengraph beginnend mit einem gegebenen Startpunkt (im einfachsten Fall ist dies der Wurzelknoten des Szenengraphens) vollständig bis zu den Blättern oder bis zum Erreichen einer bestimmten Abbruchbedingung durchlaufen (siehe Kapitel 3.1.2). Während dieser Traversierung werden die zum Darstellen der aktuellen Szene benötigten Szenengraphendaten extrahiert. In einem zweiten Schritt werden diese gesammelten Informationen mit der aktuellen, lokalen Sichtweise des jeweiligen Benutzers auf die virtuelle Welt verknüpft, um das Ergebnis anschließend im letzten Schritt zur eigentlichen Darstellung an die Grafikkarte weiterzuleiten.

Die Länge der einzelnen Schritte korreliert dabei direkt mit der Menge der zu verarbeitenden (Grafik-)Daten. D.h., je komplexer ein Szenengraph aufgebaut ist bzw. je mehr Daten darin enthalten sind, desto länger dauert der zugehörige Rendervorgang für jedes einzelne Bild, da in jedem Schritt mehr Daten zu verarbeiten sind und diese dadurch auch mehr Laufzeit benötigen. Ein Benutzer kann dieses Verhalten durch sinkende Bildwiederholraten der Anwendung bei steigender

Komplexität der virtuellen Umgebung bemerken[27].

Beim direkten Rendering werden nun diese Schritte alle gemeinsam vorgenommen, wobei die beschriebenen Abläufe ineinander verzahnt sind. Die gesammelten Szenengraphenobjekte werden so bereits parallel zur Traversierung nachfolgender Objekte mit der lokalen Ansicht des Avatars abgeglichen und weiter an die Grafikkarte gesendet, um so einen schnelleren Rendervorgang zu ermöglichen. Durch diese enge Verzahnung bzw. die teilweise Überlagerung der einzelnen Schritte stellt das direkte Rendering die effektivste Art eines Rendervorgangs dar.

Dieses Vorgehen entspricht dabei der Standardvorgehensweise bei nahezu allen Rendervorgängen in Einzelbenutzerszenarien und benötigt an sich auch keine weitergehende Synchronisierung innerhalb dieses Prozesses.

3.4.4.2 Mehrstufiges Rendern

In Mehrbenutzerszenarien[28] ist das direkte Rendering allerdings nicht mehr in allen Fällen effizient anwendbar. Da bei gleichzeitigem Zugriff mehrerer Stationen auf die gemeinsamen Szenengraphendaten während des Rendervorgangs auch potentiell Schreibvorgänge anderer Station auftreten können, die zwar im Kollisionsfall durch die transaktionale Konsistenz transparent im Hintergrund aufgelöst werden, kann es dabei aber in ungünstigsten Fällen gerade im Rahmen dieser Kollisionsauflösung zu einem Abbruch und Wiederanlaufen einer für das Rendern zuständigen Transaktion führen. Dies resultiert in einer kurzen Verzögerung des eigentlichen Rendervorgangs der Station, welches sich bei mehrfachem Auftreten auch bis hin zum Benutzer durch sinkende Bildwiederholraten bemerkbar machen kann.

Daher ist das direkte Rendering in Mehrbenutzerszenarien nur für sehr kleine Szenarien geeignet, wenn insgesamt nur wenig Szenengraphendaten zu traversieren sind, oder auch falls nur wenig bis selten Änderungen auf den zu rendernden Szenengraphendaten durchgeführt werden, so dass nur selten Kollisionen auftreten.

Um etwaige Einbrüche der Bildwiederholraten durch eventuelle Verzögerungen bei größeren bzw. häufig geänderten Szenen entgegenzuwirken, können spezielle Mechanismen zur Optimierung verwendet werden. Eine elegante und einfach zu realisierende Möglichkeit ist die Umstellung des direkten Renderings hin zu einer mehrstufigen Variante.

Beim mehrstufigen Rendern werden die zuvor erwähnten Schritte in einzelne, voneinander getrennte Stufen oder Phasen unterteilt (siehe Abbildung 37). In der ersten Stufe wird dann einzig der Szenengraph traversiert und dabei die zum Rendern benötigten Szenengraphendaten extrahiert. Die zweite Stufe bereitet dann diese entnommenen Daten auf und nimmt notwendige Berechnungen für die lokale Sichtweise daran vor, um diese im letzten Schritt zur eigentlichen Darstellung an die Grafikkarte senden zu können. Insofern unterscheiden sich diese Schritte, abgesehen von der vollständigen Trennung der Stufen voneinander, inhaltlich nicht weiter vom direkten Rendering, weshalb sich eine Anpassung des direkten Rendering eines Einzelplatzszenariums auf ein Mehrbenutzerumfeld sehr leicht vornehmen lässt. In einem transaktionsbasierten Umfeld können diese einzelnen Stufen sogar nahezu unmittelbar durch einfaches Aufspalten des ursprünglich

27 Spezielle Vorgehensweisen können zwar auch feste Bildwiederholraten oder das Absinken unter bestimmte Grenzwerte vermeiden, indem beispielsweise dieselben Bilder mehrfach angezeigt oder Teile der zu verarbeitenden Daten weggelassen werden; diese Mechanismen verschleiern aber nur die sichtbaren Auswirkungen für den Benutzer, die eigentliche Ursache für das Absinken der Bildwiederholraten wird hierdurch nicht behoben.

28 Dies gilt dabei nicht nur für mehrere, physikalisch getrennte Rechner, sondern ebenfalls bei Einzelplatzszenarien, bei denen ein Mehrprozessor-System Verwendung findet, in welchem unterschiedliche Threads bzw. Prozesse simultan auf den gemeinsamen Szenengraphen zugreifen können.

direkten Renderings in drei einzelne Transaktionen voneinander entkoppelt werden.

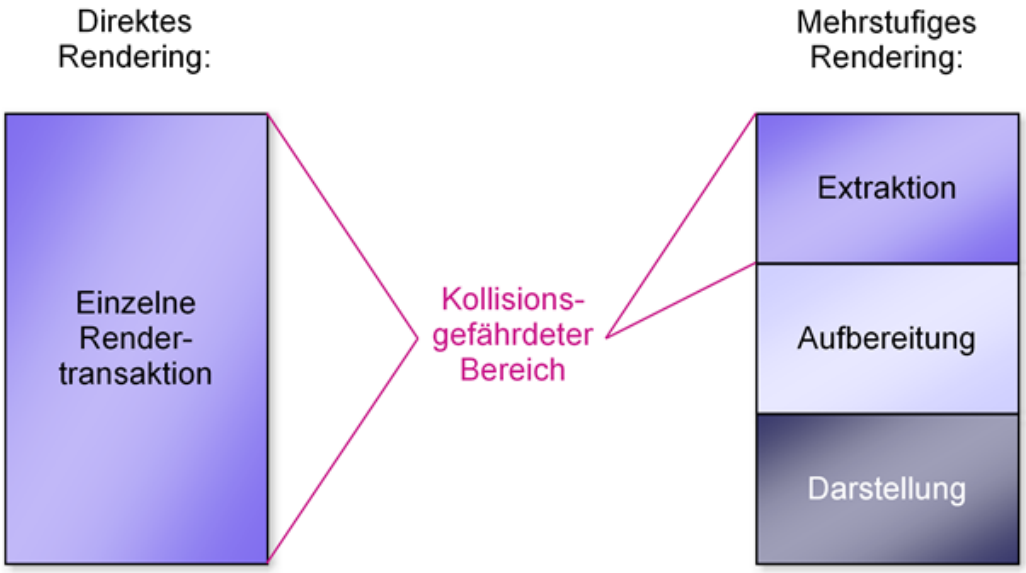

Abbildung 37: Direktes vs. mehrstufiges Rendern

Der Vorteil dieser Trennung liegt in der damit verbundenen deutlichen Absenkung des potentiellen Kollisionsrisikos bei simultan auftretenden Schreibzugriffen auf gemeinsame Daten. Beim direkten Rendering übernimmt eine einzelne Transaktion den kompletten Rendervorgang. Während der gesamten Laufzeit dieser Transaktion kann ein Schreibzugriff einer weiteren Station – beispielsweise die Positionsänderung eines zweiten Avatars im Sichtfeld – einen Abbruch der eigenen Rendertransaktion auslösen. Diese wird dann zwar sofort wieder gestartet, verliert aber durch den Abbruch die bisherige Laufzeit, was zu kurzzeitigen Verzögerungen führt.

Wird nun der Rendervorgang auf mehrere einzelne, hintereinandergeschaltete Stufen aufgeteilt, so kann der Zugriff auf die gemeinsamen Daten auf die erste Stufe allein, d.h. der Traversierung und Extraktion der Szenengraphendaten, beschränkt werden. Diese Phase besitzt zwar immer noch ein gewisses Kollisionsrisiko mit den Schreibzugriffen anderer Transaktionen, doch ist dieses durch die wesentlich kürzere Laufzeit deutlich abgesenkt. Die beiden nachfolgenden Stufen können anschließend komplett lokal weiterarbeiten und sind daher durch nicht konkurrierende Zugriffe auf ihre lokalen Daten auch nicht mehr abbruchgefährdet.

Abbildung 38 verdeutlicht diesen Sachverhalt anhand eines Beispiels mit drei Stationen. Station 1 ändert hierbei periodisch Daten des gemeinsamen Szenengraphens ab (z.B. durch eine automatisierte, zeitgesteuerte Animation). Währenddessen führt Station 2 ebenfalls periodisch ein direktes Rendering durch. Infolge der einzelnen Rendertransaktion von Station 2 ist diese jeweils während ihrer gesamten Laufzeit über anfällig gegenüber den potentiellen Schreibvorgängen von Station 1. Tritt hierbei der Kollisionsfall ein, so führt dies zu einem Abbruch und anschließendem Neustart der Rendertransaktion von Station 2, wodurch sich eine Verzögerung im Darstellungsablauf bei dieser Station ergibt.

Station 3 führt nun ebenfalls ein periodisches Rendering durch, verwendet aber anstelle des direkten Renderings ein mehrstufiges Renderverfahren wie zuvor in Abbildung 37 dargestellt. Somit ist nun nur noch die erste Phase des Rendervorgangs anfällig für eventuelle Kollisionen mit den Schreibvorgängen der ersten Station. Ein Konflikt zum Zeitpunkt einer nachfolgenden Stufe ist für den restlichen Renderprozess dann nicht mehr von Belang bzw. gar nicht mehr möglich, da diese

Phasen auf rein lokalen Daten ausgeführt werden und folglich nicht mehr abgebrochen werden können. Hierdurch verkürzt sich insgesamt die kollisionsgefährdete Zeitspanne und in Folge dessen reduziert sich auch wesentlich das verbundene Kollisionsrisiko und damit einhergehende Verzögerungen. Analysen an einem Testsystem haben zudem gezeigt, dass das Verhältnis der kollisionsgefährdeten Extraktionsphase zu den restlichen Phasen noch wesentlich kürzer ausfällt, als in Abbildung 37 dargestellt (siehe Kapitel 5.2.2).

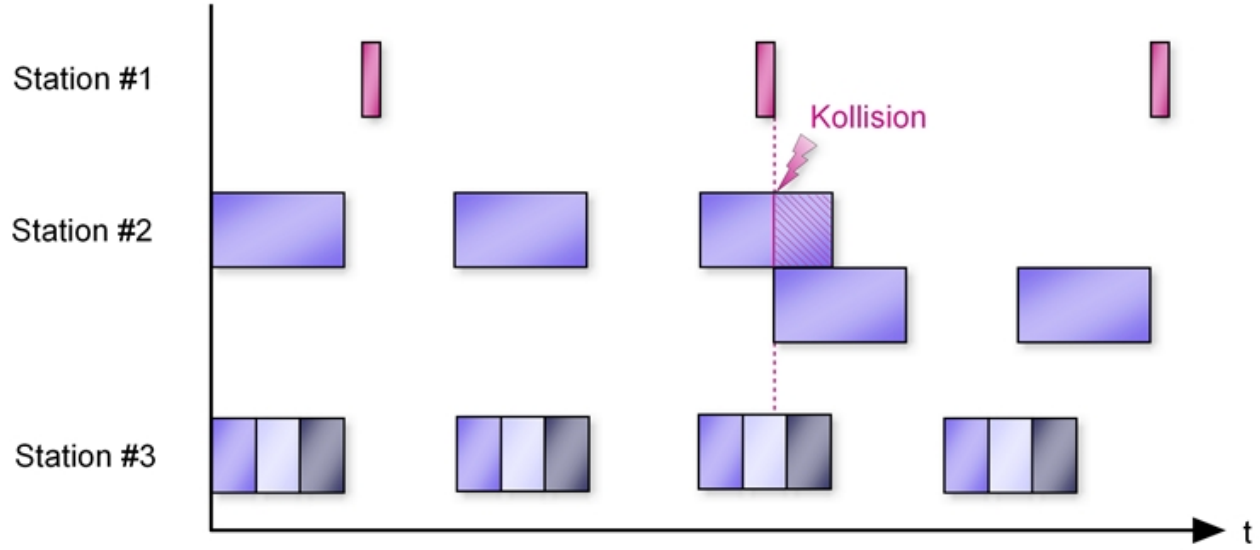

Abbildung 38: Absenkung des Kollisionsrisikos durch mehrstufiges Rendern

Da beim mehrstufigen Rendern die einzelnen, hintereinandergeschalteten Stufen auf den Ergebnissen ihrer Vorgänger aufbauen, ist jedoch eine zusätzliche Zwischenspeicherung der jeweils zuvor angesammelten Daten notwendig. Dies wird mittels lokal vorgehaltener Strukturen vorgenommen, welche dann zur Übergabe für die nachfolgenden Stufen verwendet werden und dabei je nach Art der verwendeten Daten noch weitere Optimierungsmöglichkeiten bieten. Im einfachsten Fall wird von der beginnenden Extraktionsphase eine vollständige, lokale Kopie des verteilten Szenengraphens bzw. des relevanten Subgraphens angelegt, auf dem dann die nachfolgenden Stufen direkt weiterarbeiten. Eine bereits optimierte Vorgehensweise kopiert dahingegen nur die geänderten bzw. mit einer gewissen Wahrscheinlichkeit geänderten Daten ('Hot Data') und erstellt bei statischen oder vergleichsweise selten geänderten Daten ('Cold Data') nur Verweise auf den originalen Szenengraphen bzw. übernimmt sogar die Strukturen vom vorherigen Renderdurchlauf[29].

Analog können auch mehrstufige Varianten beliebiger anderer (z.B. Animations-)Berechnungen auf diese Art gebildet werden, insbesondere bei ansonsten aufwändigen bzw. langlaufenden Transaktionen. Hierdurch lässt sich das Kollisionsrisiko dieser Transaktion bei zusätzlichen Schreibzugriffen auf die gemeinsamen Daten wesentlich verringern und potentiell auftretenden Verzögerungen weiter entgegenwirken oder ganz vorbeugen.

29 Der im Rahmen dieser Arbeit entstandene Prototyp eines virtuellen Präsenzsystems, 'World of Wissenheim' (siehe Kapitel 2.4), arbeitet ebenfalls nach diesem optimierten Prinzip. Zusätzlich werden die hier vorgestellte zweite und dritte Stufe zu einer gemeinsamen Renderphase zusammengefasst, um hierbei wieder eine stärkere Überlagerung dieser beiden lokalen und nicht abbrechbaren Phasen zu erreichen und eine Datenübergabe zwischen diesen beiden Phasen zusätzlich einzusparen (siehe auch [52]).

Durch einen mehrstufigen Rendervorgang lässt sich somit verhältnismäßig einfach und intuitiv eine (zumindest schwache) Art der Synchronisierung erreichen. Insbesondere wenn die zu entkoppelnden Aufgaben bereits modular strukturiert sind bzw. modular programmiert wurden, lässt sich eine (auch nachträgliche) Aufteilung in einzelne, auf Transaktionen abbildbare Phasen unproblematisch vornehmen.

3.4.4.3 Synchronisiertes Rendern

Üblicherweise arbeiten die einzelnen Stationen autonom voneinander die von ihnen durchgeführten Rendervorgänge ab. Sollte aber von einer Anwendung ein exakter Abgleich zwischen den Stationen erforderlich sein, bei der auftretende Kollisionen wie eben beschrieben nicht nur vermindert sondern gänzlich vermieden werden müssen, so ist dies durch eine starke Form der Synchronisierung möglich. Zu diesem Zweck können unterschiedliche Mechanismen genutzt werden, wie sie aus dem Bereich der Softwaretechnik für Synchronisierungsaufgaben bei verteilten Systemen bekannt sind [53]. Eine gebräuchliche Form stellt hierbei der Einsatz von expliziten Barrieren dar, welche im Folgenden beispielhaft als eine mögliche Art der Synchronisierung weiter verwendet werden.

Beim Einsatz von Barrieren arbeiten alle Prozesse oder Rechner so lange ihre eigenen Berechnungen ab, bis sie eine Barriere erreichen. An dieser Stelle stoppt die Verarbeitung des aktuellen Prozesses bzw. der zugehörigen Transaktion (entweder durch Blockieren oder aktives Warten an dieser Stelle), bis alle anderen teilnehmenden Prozesse oder Rechner ebenfalls diese Barriere erreicht haben. Erst dann können alle Beteiligten wieder gemeinsam mit den nächsten Berechnungsschritten fortfahren. Die Überwachung und Freigabe an der Barriere erfolgt dabei entweder über einen zentralen Koordinator oder dezentral, beispielsweise über gegenseitige Rundsprüche der teilnehmenden Stationen.

Integriert in den (mehrstufigen) Rendervorgang bedeutet dies nun, dass nach Erreichen einer eingefügten Barriere zu Beginn des Renderdurchlaufs dafür gesorgt wird, dass alle Stationen gemeinsam mit dem Rendervorgang fortfahren können. Dies bewirkt, dass alle Stationen, unabhängig von ihrer individuellen Arbeitsgeschwindigkeit, zur gleichen Zeit mit dem Rendervorgang beginnen und so eventuelle Geschwindigkeitsunterschiede zwischen den einzelnen Stationen synchronisiert werden.

Der Einsatz von Barrieren bedeutet dabei allerdings auch, dass sich die Arbeitsgeschwindigkeit aller an den Barrieren teilnehmenden Stationen – sofern keine Aufweichung des Barrierenkonzepts verwendet wird – anhand der Geschwindigkeit der langsamsten Station orientieren, wodurch schnellere Stationen ausgebremst werden können[30]. Bezogen auf einen Rendervorgang kann dies auf schnellen Rechnern zu geringeren Bildwiederholraten führen als in vergleichbaren Fällen, wenn keine Barrieren verwendet werden. Die beiden schnellen Stationen 2 und 3 aus Abbildung 39 zeigen den Unterschied bei einem kontinuierlichen Rendervorgang auf, in dem Station 2 nach Erledigung seines Rendervorgangs vor dem nächsten Durchlauf immer kurz auf die langsamere, aber durch die gemeinsame Barriere verknüpfte Station 1 wartet, Station 3 hingegen ohne Synchronisierung unverzüglich weiterarbeiten kann.

30 Dies bezieht sich auf das originale Konzept von Barrieren. Zur Verbesserung des angesprochenen Sachverhaltes lassen sich auch optimierte Varianten von Barrieren einsetzen. So kann beispielsweise bei geringen zu erwartenden Geschwindigkeitsabweichungen die Barrierensynchronisierung in größeren Abständen erfolgen, d.h. es wird nicht zwingend bei jedem Renderdurchlauf an der Barriere angehalten. Ebenso können Timeouts eingeführt werden, nach denen mit der Programmausführung in jedem Fall fortgefahren wird, um eine gewisse Mindestwiederholrate zu gewährleisten und insbesondere bei einer Fehlersituation kein dauerhaftes Blockieren aller beteiligter Rechner bzw. Prozesse hervorzurufen.

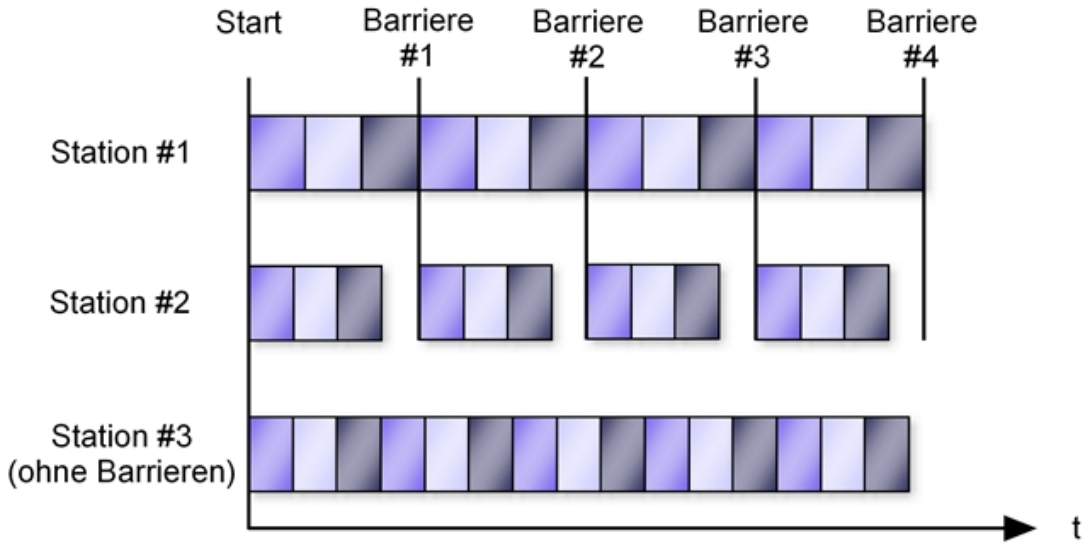

Abbildung 39: Explizite (Render-)Synchronisierung mit Barrieren

Um sogar eine exakte, bildgenaue Synchronisierung zu erreichen, wie es z.B. der Einsatz von Spezialhardware erfordert, können Barrieren auch noch feingranularer direkt mit dem Umschaltvorgang des dargestellten Bildes verknüpft werden. Hierbei wird der Rendervorgang von jeder Station ganz normal (auch gegebenenfalls abweichend bezüglich der Darstellungsgeschwindigkeit anderer Stationen) durchgeführt; der tatsächliche Umschaltvorgang des dargestellten Bildes auf das nächste, bereits fertig gerenderte Bild innerhalb der Grafikkarte[31] wird dann aber erst gemeinsam über die Barriere initiiert (so z.B. auch im Myriad-System; siehe Kapitel 3.2.3.7).

Auch weitere, periodisch durchgeführt Abläufe können einfach mit in dieses Konzept integriert werden, wie beispielsweise das gemeinsame Animieren von Objekten in einer eigenen Animationsphase, welche mit über die vorhandene Barriere synchronisiert werden. Werden darin alle Schreibvorgänge auf den Szenengraphenobjekten simultan ausgeführt, wobei hier jede Station nur ihren „eigenen" Avatar bzw. nur die von ihr animierten Objekte bewegt, so treten in diesem Moment zwar viele Schreibzugriffe gleichzeitig innerhalb des Szenengraphens auf, allerdings werden diese jeweils auf disjunkten Daten ausgeführt. In Folge dessen verursachen diese Schreibzugriffe keine Konflikte untereinander und durch die Synchronisierung auch nicht mit anderen Zugriffen weiterer Transaktionen – insbesondere nicht innerhalb der Extraktionsphase anderer Stationen –, wodurch Kollisionen nicht nur vermindert sondern gänzlich vermieden werden können.

Umgekehrt ist eine Mischung von barrierensynchronisierten Rendervorgängen einer Teilmenge der teilnehmenden Stationen mit einzelnen unsynchronisierten Stationen nicht zu empfehlen. Speziell die nicht synchronisierten Stationen sind zwar theoretisch ohne den Aufwand und eventuelle Wartezeiten an den Barrieren etwas schneller als die synchronisierten Stationen, allerdings kann

31 Das Rendern eines Bildes vor dem eigentlichen Anzeigen in einen (noch) nicht sichtbaren Bereich der Grafikkarte und ein Austauschen mit dem aktuellen Bild nach dessen Fertigstellung entspricht dem sogenannten 'Back -' bzw. 'Double Buffering' und ist heutzutage die Standardvorgehensweise von Grafikkarten, um Bilder darzustellen. Hierdurch lässt sich ein ungewünschter Effekt unterdrücken, bei dem ein noch nicht fertig gerendertes Bild bereits angezeigt wird, während die Grafikkarte noch weitere Objekte oder Texturen darin einfügt oder ein Flackern durch das Löschen des alten Bildinhaltes vor dem nächsten Renderdurchlauf sichtbar wird.
Partiell wird heutzutage auch schon mit Triple-Buffering oder noch mehr Zwischenpuffern gearbeitet, um noch weichere Übergänge zwischen den Umschaltvorgängen zu erreichen.

dies in einigen Fällen auch zu deutlichen Leistungseinbußen auf diesen Stationen führen.

Abbildung 40 verdeutlicht diese Situation. Es herrscht dieselbe Ausgangslage wie zuvor (Station 1 ist etwas langsamer als Station 2 und 3 und die ersten beiden Stationen sind über eine Barriere synchronisiert). Lediglich ein Schreibvorgang wird von allen Stationen zusätzlich nach dem Rendern vorgenommen, in dem sie beispielsweise ein Objekt animieren. Station 1 und 2 bleiben aufgrund der Synchronisierung durch die Barriere konstant in ihrem Verhalten. Station 3 dahingegen beginnt bereits früher mit dem nächsten Rendervorgang, da sie nicht an der Barriere ansteht, wird allerdings ständig während ihrer kollisionsgefährdeten Extraktionsphase durch den späteren Schreibvorgang von Station 1 abgebrochen und muss danach die zugehörige Transaktion neu starten. Hierdurch verliert sie dauerhaft ihren eigentlichen Geschwindigkeitsvorteil und ist letztendlich nicht schneller als die anderen, synchronisierten Stationen. Im ungünstigsten Fall können bei Station 3 sogar noch stärkere Verzögerungen durch den Aufwand einer Rücksetzung der abgebrochenen Transaktion auftreten, welche in der Abbildung aber nicht berücksichtigt werden.

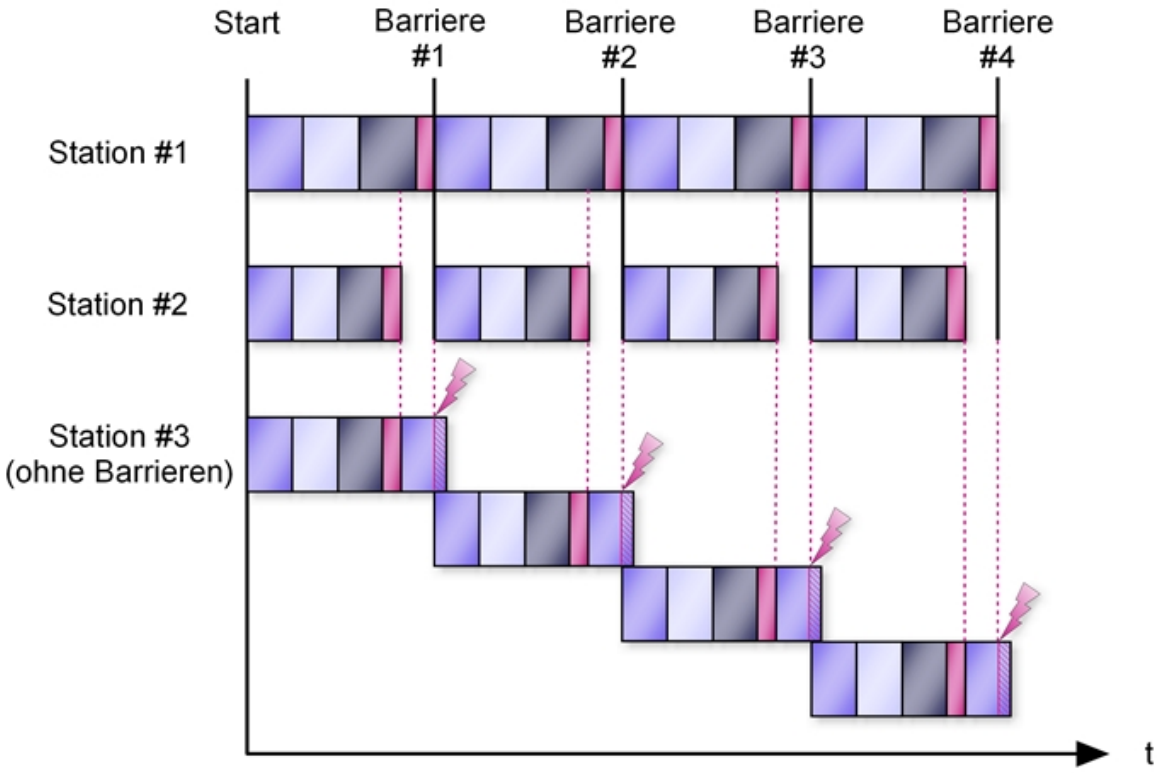

Abbildung 40: Auswirkungen auf die Rendervorgänge durch eine zusätzliche Schreibphase

Ein Vergleich der Arbeitsweisen eines transaktionalen Szenengraphensystems mit und ohne Barrieren wird in Kapitel 5.5 vorgenommen.

3.4.5 Kooperatives Arbeiten

Die Vorteile eines gemeinsamen, transaktionalen Szenengraphensystems lassen sich insbesondere auch bei der kooperativen Arbeit mehrerer Avatare bzw. Benutzer an gemeinsamen Objekten mit einbringen. Ohne dass sich hierbei ein Entwickler um spezielle Synchronisierungsmechanismen kümmern muss, lassen sich Modifikationen an Objekten innerhalb des Szenengraphens von mehreren Benutzern unmittelbar bei allen anderen Stationen sichtbar machen.

Dies kann sowohl für die 3D-Objekte aus denen die virtuelle Umgebung aufgebaut ist verwendet werden, um beispielsweise die Umgebung und Landschaft anzupassen, oder auch für gemeinsame Arbeitsprojekte, wie z.B. im CAD-Bereich. Auch in der Lehre kann ein solches Vorgehen eingesetzt werden, um Lerninhalte komplexer Sachverhalte über die gemeinsame Betrachtung und Erforschung mehreren Avataren zugänglich zu machen, sowie auch darüber hinaus für weitere Objekte, welche sich in einem transaktionalen Szenengraphen einfügen lassen, beispielsweise Texte, die dann ebenfalls gemeinsam editiert werden können ('Whiteboard'-Funktionalität).

Um die Auflösung eventueller Konflikte bei simultanen Modifikationen auf den gemeinsamen Objekten kümmert sich dabei die transaktionale Konsistenz, so dass automatisch gewährleistet ist, dass sichtbare Modifikationen auch von wirklich allen teilnehmenden Avataren einheitlich gesehen werden, und keine lokalen Abweichungen entstehen.

Ein Beispiel für die Umsetzung dieser Einsatzmöglichkeit stellt der in dieser Arbeit vorgestellte Wissenheim-Prototyp dar (siehe Kapitel 2.4.3.2).

3.5 Zusammenfassung

Szenengraphen stellen die Grundstruktur aller virtueller 3D-Umgebungen dar, indem sie den Aufbau und die Anordnung der darin enthaltenen Objekte beschreiben. Es gibt verschiedene mögliche Arten der Organisation von darin enthaltenen Szenengraphendaten; die gebräuchlichste ist eine Umsetzung als Transformationsgraph, in der die übergeordneten Elternknoten ihre Eigenschaften auf die darunterliegenden Kindknoten vererben. Oftmals ist mit einem Szenengraphen auch eine Renderengine zur Darstellung der darin enthaltenen Daten verknüpft, wodurch sich ein ganzes Szenengraphensystem daraus ergibt.

Die ursprünglichen Szenengraphen waren nur für Einzelplatzsysteme konzipiert. Mit der Vernetzung von Rechnern und virtuellen Welten wurde auch der Einsatz von Szenengraphen über Rechnergrenzen hinweg erforderlich und es entwickelten sich verteilte Szenengraphen. Die häufigste Realisierungsart hierzu stellt eine klassische Client-Server-Architektur dar. Mit ihr lassen sich in bisherigen Systemen Synchronisierungsaufgaben und die Konsistenzhaltung von verteilten Szenengraphendaten zwischen mehreren Teilnehmern relativ leicht vornehmen. Allerdings stellt die zentrale Serverkomponente eine potentielle Engstelle im System dar und es wird in der Regel viel Berechnungsaufwand auf die Klienten verlagert, um den Server zu entlasten, was dadurch aber zu mehrfachen, parallelen Berechnungen von allen Klienten führt. Seltener sind Peer-to-Peer-basierte Szenengraphen oder Hybrid-Systeme anzutreffen, da bei ihnen in bestehenden Systemen die Konsistenzhaltung der Daten bislang deutlich umständlicher ausfällt.

Aus dem Gebiet der Szenengraphen wurden verschiedene Arbeiten von Einzelplatzszenengraphen als auch von verteilten Varianten vorgestellt und ihre Merkmale bzw. Vor- und Nachteile eingehend erläutert. Im Anschluss daran wurde der im Rahmen dieser Arbeit entwickelte alternative und neuartige Ansatz eines transaktionalen Szenengraphens präsentiert und dessen Vorzüge bzw. Eigenschaften dargelegt. Konkrete Messungen, welche diese Aussagen untermauern, werden im

späteren Kapitel 5 noch aufgeführt.

Das Konzept eines transaktionalen Szenengraphens entspricht dabei einem Peer-to-Peer-ähnlichen-Szenario der teilnehmenden Rechner mit einem starken Konsistenzmodell ohne die Notwendigkeit zentraler Komponenten. Es basiert auf der Nutzung eines verteilten Speichers zur Kommunikation mit anderen Stationen und einer Konsistenzhaltung der Daten mittels der transaktionalen Konsistenz. Anders als bei bestehenden Szenengraphensystemen mit verteilten Speichermodellen wird vom transaktionalen Szenengraphen dauerhaft eine starke Konsistenzform zur Verfügung gestellt, welche den Einsatz zusätzlicher Synchronisierungsaktionen im Gegensatz zu bestehenden Systemen optional werden lässt. Die Verteilung der gemeinsamen Daten durch den verteilten Speicher wird dabei ebenfalls automatisch und für die Anwendung transparent vorgenommen. Somit wird durch die Struktur und Eigenschaften eines transaktionalen Szenengraphens eine einfache und intuitive Art der Entwicklung und Nutzung von verteilten, graphischen Anwendungen realisiert, die auch Zeit und Aufwand für zusätzliche Synchronisierungsmaßnahmen und Konsistenzbetrachtungen einspart.

Durch den direkten und bedarfsgesteuerten Zugriff auf die Daten innerhalb des transaktionalen Szenengraphens entfällt zusätzlich die Notwendigkeit, die von einer Station benötigten Daten lokal zwischenzuspeichern und vor der Benutzung weiter aufbereiten zu müssen, um aus empfangenen aber eventuell veralteten Daten wieder aktuell gültige Daten zu generieren. Des Weiteren entfällt durch die von einem transaktionalen Szenengraphen ermöglichte gleichberechtigte Rolle aller teilnehmenden Stationen in einer virtuellen Welt auch die Unterscheidung und getrennte Entwicklung zwischen dem Aufgabenbereich eines Servers und dem von den Benutzern bedienten Klienten. Vielmehr können Berechnungen von beliebigen Stationen auch stellvertretend für andere Stationen ausgeführt und über den transaktionalen Szenengraphen direkt bereitgestellt werden. So lassen sich auf einfache Art und Weise Skalierungen des Gesamtsystems erreichen, Lastausgleiche zwischen den Stationen vornehmen und unterschiedlich leistungsfähige Hardware bis hin zu mobilen Endgeräten in ein solches System integrieren. Genauso leicht lässt sich die Darstellung der Szenengraphendaten variieren, um einen beliebigen Mehrschirmbetrieb ohne separate Spezialhardware zu ermöglichen. Abschließend ermöglicht ein transaktionaler Szenengraph bei Bedarf von speziellen Anwendungen oder Einsatzgebieten auch die problemlose Integration expliziter Synchronisierungen, wobei diese aber für den Normalbetrieb konzeptuell nicht zwingend erforderlich sind.

4. Hardwareseitige Unterstützung transaktionaler Szenengraphen

Im bisherigen Teil dieser Dissertation wurden die Fähigkeiten und Einsatzmöglichkeiten eines transaktionalen Szenengraphens als Basis für virtuelle Präsenzsysteme beschrieben. Das dabei verwendete und zugrundeliegende Konzept des gemeinsamen verteilten Speichers und der transaktionalen Konsistenz wird hierbei bislang als spezielle Form eines Programmiermodells auf Softwareseite umgesetzt. In diesem Kapitel wird nun untersucht, ob diese Strukturen für ein transaktionales Präsenzsystem auch darüber hinaus direkt von der Hardware unterstützt werden können bzw. ob und wie sich existierende Hardware in einem transaktionalen Szenengraphen verwenden lässt.

Eine grundlegende Analyse wie normale Hardwarekomponenten (Tastatur, Netzwerkkarte, Drucker, ...) in ein transaktionales Umfeld integriert werden können und wie die hierfür nötige Ansteuerung und Treiberstruktur für diese Hardwarekomponenten aussehen muss, wird bereits in [1] beschrieben. Es wird dort primär auf die Kommunikation von nicht-transaktionaler Hardware mit einem transaktionalen Heap eingegangen. Diese Vorgehensweise trifft prinzipiell auf alle Standardhardwarekomponenten zu, da diese nicht transaktional arbeiten und zumeist auch von ihrem Aufbau nicht den Anforderungen eines Transaktionskonzeptes genügen. Insbesondere für die Zurücksetzung von Transaktionen im Kollisionsfall und einem damit verbundenen Wiederherstellen der ursprünglichen Arbeitsdaten wird mit Hilfe der Einführung des Konzeptes von 'SmartBuffern' eine sichere Vorgehensweise zur Übermittlung von Gerätedaten von der nicht-transaktionalen Hardware in den transaktionalen Heap und umgekehrt aufgezeigt.

Auf die Kommunikation zwischen den Hardwarekomponenten und dem transaktionalen Heap wird an dieser Stelle daher nicht weiter eingegangen. Es wird vielmehr betrachtet, ob und wie diese bisherige strikte Trennung der Hardware mit ihrer nicht-transaktionalen Arbeitsweise einerseits und dem transaktionalen Heap andererseits überwunden werden kann und ob hier eine direkte Einbeziehung von Hardwarekomponenten unter Umgehung der SmartBuffer möglich ist.

Ein Vorteil der sich hierdurch ergeben würde, wäre eine deutliche Vereinfachung in der Ansteuerung der zugehörigen Geräte. Bislang werden alle Komponenten innerhalb eines Rechners über ein festes Master-Slave-Schema angesprochen, wobei die CPU dem Master entspricht und die Kontrolle über die restlichen Systemkomponeneten – den Slaves – besitzt und für die Zuteilung der von ihnen zum Arbeiten nötigen Daten zuständig ist. An diese Vorgehensweise ist auch eine mehrfache Speicherung der verwendeten Daten geknüpft, da sie vom Master kontrolliert und erst bei Verwendung an den Slave weitergeleitet, d.h. für gewöhnlich umkopiert, werden. Durch eine direkte Einbeziehung von Hardwarekomponenten innerhalb eines transaktionalen Systems könnte für diese Geräte die indirekte Arbeitszuteilung und die mehrfache Datenhaltung durch die CPU verringert werden oder im besten Fall sogar wegfallen.

Eine solche Betrachtung ist allerdings nur für diejenigen Hardwarekomponenten sinnvoll, welche über eine gewisse Mindestausstattung an eigenen Ressourcen verfügen, die sich in ein transaktionales Umfeld integrieren lassen und die auch von außerhalb des Gerätes ansprechbar sind. Hardwarekomponenten, die ihre Ressourcen nur geräteintern und abgeschottet verwalten – beispielsweise der Zwischenspeicher eines Druckers, in welchem dieser zu druckende Daten empfängt und selbständig ablegt, welcher aber selbst nicht wieder auslesbar ist –, sind daher nicht geeignet, um in einem transaktionalen Heap integriert zu werden. Deshalb ist zur Nutzung einer

Hardwarekomponente insbesondere das Vorhandensein von lokalem Gerätespeicher nötig, welcher sich über Speicherzugriffe des Hauptprozessors ansprechen lässt.

Derzeit wird diese Anforderung hauptsächlich nur von modernen Grafikkarten erfüllt, weswegen die weitere Analyse innerhalb dieses Kapitels im Folgenden auf diese Gerätekategorie beschränkt wird. Auf andere (auch zukünftige) Geräte können die nachfolgenden Aussagen allerdings analog übertragen werden, sofern die zuvor erwähnten Voraussetzungen zutreffen. Grafikkarten qualifizieren sich für eine solche Betrachtung auch zusätzlich durch den im Verhältnis zu anderen Gerätekategorien deutlich größeren lokalen (Grafikkarten-)Speicher[32], wodurch sich auch ein eventuell höherer Aufwand im Vergleich zu normalem Hauptspeicher lohnt, diesen direkt in einen transaktionalen Szenengraphen zu integrieren.

4.1 Grafikkartenaufbau

Bevor genauer auf das Zusammenspiel zwischen den transaktionalen Strukturen und der Grafikkartenhardware eingegangen werden kann, ist eine Betrachtung des Aufbaus und der Arbeitsweise heutiger Grafikkarten und speziell deren Speichern vonnöten, um die Besonderheiten bei der Verknüpfung dieser Strukturen und Konzepte zu verdeutlichen.

4.1.1 Allgemeiner Aufbau

Auch wenn heutige Grafikkarten bezüglich ihrer Leistung diejenigen Modelle der ersten Stunde inzwischen um mehrere Größenordnungen übertreffen, sind die wesentlichen Komponenten und Aufgaben immer noch gleich geblieben. Abbildung 41 zeigt das prinzipielle Blockschaltbild einer Grafikkarte mit den aufs Wesentliche reduzierten Komponenten. Die beständig zunehmenden Leistungssteigerungen bei Grafikkarten werden dabei nahezu ausschließlich im Bereich der einzelnen Verarbeitungseinheiten erreicht, wozu auch die (mittlerweile zahlreich vorhandenen) Grafikpipelines gezählt werden. Beispielsweise geschieht dies, indem anstelle von ursprünglich hart verdrahteten und somit festen Hardwarestrukturen inzwischen frei programmierbare Vertex- und Pixelshader bis hin zu noch generischer einsetzbaren Uniform-Shadern Einzug gefunden haben. Die restlichen Komponenten der Grafikkarte werden dabei hauptsächlich nur zur Unterstützung dieser Verarbeitungseinheiten weiterentwickelt, um deren Leistung auch nach außen hin verfügbar zu machen. Die Arbeitsweisen dieser Komponenten selbst ändern sich dabei aber nur wenig.

32 Aktuelle Grafikkartenmodelle besitzen bereits bis zu 2 GiB an eigenem, lokalem Grafikkartenspeicher, welcher dadurch teilweise sogar mit der normalen Hauptspeicherausstattung des restlichen Rechnersystems gleichzieht oder, falls mehrere Grafikkarten innerhalb eines Systems eingebaut sind, diese sogar übertrifft.

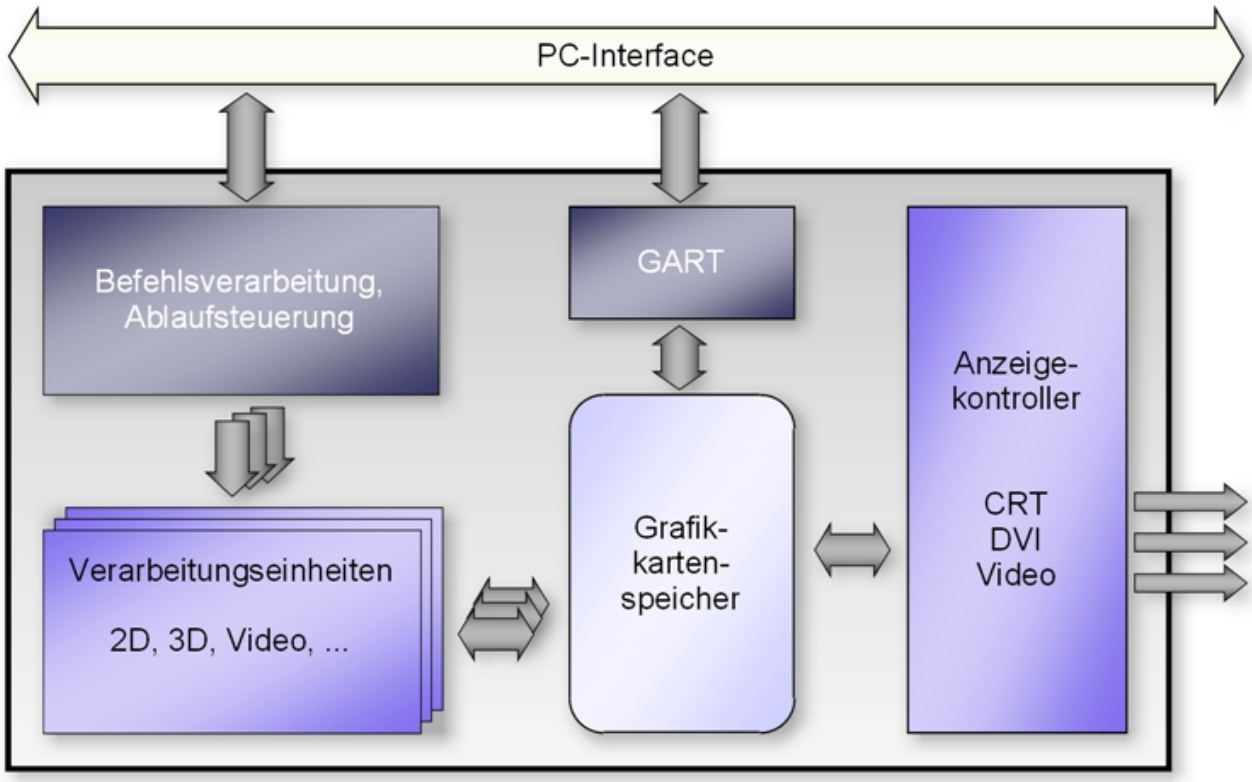

Abbildung 41: Prinzipieller Aufbau einer Grafikkarte

Während des Betriebs einer Grafikkarte werden vom Hauptprozessor (Zeichen-)Befehle an die Grafikkarte geschickt, welche von ihr aufgegriffen werden und aus denen anschließend über ihre eigene Steuereinheit Ablaufsequenzen für die einzelnen Verarbeitungseinheiten erstellt werden. Die Verarbeitungseinheiten berechnen daraufhin die gewünschten Vorgänge (Rendern eines 3D-Bildes, 2D-Operationen einer grafischen GUI, Skalieren von Videoströmen, ...) anhand vorhandener Grafikdaten aus dem Grafikkartenspeicher und legen die Resultate ebenfalls wieder im Grafikkartenspeicher ab. Die Grafikdaten selbst müssen daher entweder im Vorfeld über explizite Speicheroperationen in den Grafikkartenspeicher kopiert werden oder sie können auch teilweise selbständig von der Grafikkarte generiert werden, wie z.B. Texturen anhand prozeduraler Funktionen. Am Ende der Verarbeitungskette nimmt dann der Anzeigekontroller die fertig berechneten Bildinhalte aus dem Grafikkartenspeicher auf und wandelt sie je nach angeschlossener Bildschirmart in die dafür benötigten analogen oder digitalen Signale um.

Eine detailliertere Betrachtung des Aufbaus und der Arbeitsweise von Grafikkarten und der verschiedenen Untereinheiten wird u.a. in [54], [55] und [56] aufgezeigt.

4.1.2 Moderne Ansteuerung von Grafikkarten

Ursprünglich arbeiteten Grafikkarten rein registerbezogen, d.h. Befehle an die Grafikkarte oder auch nur Änderungen von einzelnen Zuständen wurden durch die Programmierung spezieller Grafikkartenregister vorgenommen. Dabei konnte die Umsetzung eines einzelnen Grafikbefehls für eine Anwendung durchaus die Ansteuerung mehrerer Dutzend verschiedener Grafikkartenregister bedeuten, welche teilweise sogar nur in bestimmten Reihenfolgen beschrieben werden durften oder bezüglich des Timing-Verhaltens der Grafikkarte Wartezeiten zwischen einzelnen Zugriffen erforderten. Bei fehlerhaften oder zu frühen Registerzugriffen blockierte die Grafikkarte, was nicht

selten zu einem kompletten Stillstand des Gesamtsystems führte[33]. Mit der zunehmenden Komplexität und Leistungsfähigkeit der entwickelten Grafikkartengenerationen stieg dabei die Zahl der vorhandenen Register bei den nun aktuellen Grafikkarten bis in die mehreren tausend.

Um diese fehleranfällige und aufwändige Programmierung einzelner Grafikkartenregister zu vereinfachen, unterstützen heutige Grafikkarten daher zusätzlich alternative Möglichkeiten zur Grafikkartenansteuerung, welche auch komplexe Aufgaben über einfache Anweisungen zur Verfügung stellen und dadurch deutlich leistungsfähiger sind. Hierzu gehört insbesondere die Einführung von speziellen Kommandopaketen, welche vom Hauptprozessor an die Grafikkarte gesendet werden. Die Grafikkarten arbeiten dadurch nahezu vollständig befehlsorientiert. Die hierfür verwendeten Kommandopakete können unterschiedlichste Mitteilungen an die Grafikkarte beinhalten: von der einfachen Änderung eines einzelnen Grafikkartenzustandes bis hin zu komplexen Zeichenbefehlen oder gar ganzen Befehlssequenzen. Lediglich erste Initialisierungen direkt nach dem Booten werden noch registerbezogen vorgenommen (z.B. das Setzen des Pixeltaktes oder der Bildschirmauflösung), bevor die Grafikkarten ausschließlich über Kommandopakete angesprochen werden.

Zum Verarbeiten dieser Kommandopakete benötigen die Grafikkarten je nach Modell einen eigenen, individuellen Mikrocode, der während der Initialisierungsphase in die Grafikkarte geladen wird. Dieser sorgt für eine korrekte Interpretation der späteren Kommandopakete und kann auch weitergehende Funktionen innerhalb der Grafikkarte beeinflussen. Die Grafikkartenhersteller können hierüber auch nach dem Vertrieb der Grafikkarten noch weitere Funktionen und Bugfixes für ihre Grafikkarten nachträglich ausliefern, indem sie modifizierte Mikrocodes in ihren aktuellen Treiberversionen integrieren und bereitstellen. Ein fehlerhafter Mikrocode kann allerdings genauso auch die Grafikkarte vollständig stilllegen.

Der Mikrocode stellt daher einen interessanten und vielversprechenden Ansatzpunkt zur Beeinflussung der Arbeitsweise einer Grafikkarte dar, insbesondere wie es im Rahmen dieser Arbeit für eine eventuelle Umstellung ihrer Arbeitsweise auf ein transaktionales Verhalten hin potentiell hilfreich gewesen wäre. Da der Mikrocode selbst aber von den Herstellern weder dokumentiert wird noch anderweitig genauer nachvollziehbar ist und ausschließlich im Binärformat vorliegt, kann dieser leider nicht als Ausgangspunkt für eigene Adaptierungen herangezogen werden[34].

Nach dem erfolgreichen Laden des Mikrocodes in die Grafikkarte wird diese dann in den Kommandomodus umgeschaltet, d.h. ab diesem Zeitpunkt können bzw. müssen nun alle zur weiteren Ansteuerung der Grafikkarte verwendeten Befehle mittels der Kommandopakete an die Grafikkarte gesendet werden. Zur weiteren Optimierung wird die Übertragung dieser Kommandopakete für gewöhnlich nebenläufig über den DMA-Modus ('Direct Memory Access') vorgenommen[35]. Hierzu werden die vom Treiber oder auch von der Anwendung generierten Kommandopakete in einen zuvor reservierten Bereich des Hauptspeichers abgelegt, von dem aus

33 Dies kommt dadurch zustande, dass die Grafikkarte in diesen Fällen ebenfalls das PC-Interface blockiert und somit die weitere Kommunikation innerhalb eines Rechnersystems verhindert wird, wodurch dieser „einfriert“.

34 Analysen des bei ATI 2 KiB großen Mikrocodes – auch zwischen einzelnen Versionen – ließen beispielsweise keine nachvollziehbaren Strukturen erkennen; jegliche Art der Manipulation führte beständig zum Stillstand der Grafikkarte. Vermutungen des Autors lassen daher auf eine zusätzliche Sicherung des Mikrocodes mittels einer Prüfsumme oder eines Hash-Wertes schließen, welches sich aber auch durch weitere Untersuchungen nicht explizit bestätigen oder falsifizieren ließ.

35 Alternativ hierzu können die Kommandopakete teilweise auch über den PIO-Modus ('Programmed Input/Output') übermittelt werden. In diesem Fall werden die generierten Kommandopakete dann allerdings aktiv vom Hauptprozessor an die Grafikkarte gesendet und es wird darauf gewartet, dass diese von ihr entgegengenommen werden ('Busy Waiting'). Dadurch ist dieser Vorgang deutlich langsamer als bei DMA und es können keine nebenläufigen Prozesse von der CPU abgearbeitet werden, weshalb dieser Modus in der Praxis kaum genutzt wird.

die Grafikkarte sie dann selbständig auslesen kann und daraufhin mit deren Abarbeitung beginnt. Dieses Verfahren hat den Vorteil, dass die CPU mit anderen Prozessen fortfahren kann, während die Grafikkarte nebenläufig die gewünschten Zeichenoperationen ausführt. Ebenso kann die Grafikkarte die vorhandenen Kommandopakete in ihrer eigenen Geschwindigkeit abarbeiten; eventuelle Timing-Probleme werden so vermieden. Eine Anwendung kann auch sich wiederholende Kommandopakete vorbereitend im Hauptspeicher ablegen und diesen Bereich dann mehrfach zur Abarbeitung an die Grafikkarte verweisen, um hierdurch eine Wiederverwendung einmal erstellter Kommandopakete zu ermöglichen.

4.1.3 Analyse von Grafikkartenfunktionen

Um Grafikkarten ansteuern und ihre Fähigkeiten vollständig nutzen zu können, sind hardwareabhängige Treiber erforderlich, die sich nach den Spezifikationen der Grafikkartenhersteller richten. Der Einsatz von Grafikkarten in einem Standardbetriebssystem wie Windows oder Linux stellt somit auch kein Problem dar, da von den Herstellern in der Regel leistungsfähige Treiber zur Verfügung gestellt und vertrieben werden. Allerdings wurden diese Treiber von den Herstellern in der Vergangenheit bislang ausschließlich im Binärformat ausgeliefert und die Spezifikationen für den Betrieb ihrer Grafikkarten zum Schutz vor Nachahmungen als streng gehütetes Betriebsgeheimnis behandelt[36]. Somit war anhand dieser Treiber bislang die Art und Weise der Ansteuerung der Grafikkarten nicht oder nur sehr schwer nachvollziehbar. Für die unterstützten Betriebssysteme stellt dies keine Einschränkung dar; bei der Entwicklung eines neuen Betriebssystems ist man allerdings darauf angewiesen, eigene Hardwaretreiber für die zu unterstützenden Geräte zu erstellen, was insbesondere bei den Grafikkartentreibern durch die restriktive Haltung der Hersteller und der Komplexität heutiger Grafikkarten erheblich erschwert wird. Durch die Problematik der nicht veröffentlichten Spezifikationen war man bislang dazu gezwungen, entweder im Rahmen von Vertraulichkeitsvereinbarungen mit den Grafikkartenherstellen Einsicht in diese Spezifikationen zu erhalten, was von den Herstellern aber nur selten ermöglicht bzw. für nichtkommerzielle Anbieter zumeist gänzlich verweigert wird. Alternativ können Analyseverfahren herangezogen werden, um selbst durch Reverse Engineering die Funktionsweisen und Ansteuerungsarten der Grafikkarten herauszufinden, zu entschlüsseln und danach aufbauend auf diesen Ergebnissen eigene Treiber zu entwickeln.

Vom zu betreibenden Aufwand ist Reverse Engineering allerdings nicht zu unterschätzen, speziell der zeitliche Aspekt lässt sich hierbei oft nur schwer kalkulieren. Dennoch ist dies in einigen Fällen die einzige Möglichkeit, um an die Informationen zum Erstellen und Nutzen von bestimmten Treiberfunktionen zu gelangen. Zur Unterstützung hierfür bieten sich verschiedene Möglichkeiten für eine gezielte Analyse der nachzuvollziehenden Arbeitsweisen oder der übertragenen Kommandopakte und Grafikdaten an die Grafikkarte an. Im Folgenden werden daher die wichtigsten Reverse-Engineering-Konzepte etwas genauer erläutert und deren Einsatzbereiche beschrieben. Die vorgestellten Konzepte sind dabei nicht nur auf Grafikkarten beschränkt, sondern können generell zur Analyse beliebiger Hardwarekomponenten herangezogen werden.

36 Erst in jüngster Zeit werden durch verschiedene Firmenaufkäufe und Umstrukturierungen erste Bemühungen unternommen, die Spezifikationen zur Ansteuerung von Grafikkarten zu veröffentlichen. So gibt es inzwischen Dokumentationen von Intel zur Ansteuerung ihrer Grafikchips [w49] und auch ATI (mittlerweile aufgekauft von AMD) hat zumindest erste Teile ihrer Grafikkartenspezifikationen veröffentlicht [w50]. nVidia, wie auch kleinere Grafikkartenfirmen (Matrox, ...), verhalten sich diesbezüglich aber immer noch komplett verschlossen.

4.1.3.1 Sniffer

Sniffer sind Programme oder Geräte, welche zwischen die Grafikkarte und dem restlichen System eingebracht werden, um den Datenverkehr zwischen dem Rechner und der Grafikkarte zu belauschen und zur anschließenden Auswertung mitzuprotokollieren.

Dies kann durch entsprechende Softwareprogramme vorgenommen werden, die systemnah alle ausgeführten Zugriffe des Busses von und zur Grafikkarte abfangen und so an die übermittelten Daten gelangen. Allerdings kann diese softwaremäßige Überwachung nicht in allen Fällen zum gewünschten Erfolg führen, da sich eventuell nicht alle Buszugriffe abfangen lassen oder da von bestimmten Schichten bereits Filterungen und Umsetzungen bei den übermittelten Daten automatisch vorgenommen werden.

Alternativ können daher auch Hardwareadapter zwischen die Grafikkarte und dem angeschlossenen Bus eingebaut werden (z.B. AGP-Extender, ...), welche auf diese Art und Weise direkt die (elektrischen) Signale mitverfolgen und aufzeichnen können.

Eine Kommunikation mit der Grafikkarte und die dabei übermittelten Daten können so von einem Sniffer komplett gesammelt und aufgezeichnet werden. Es ist hierbei allerdings zu beachten, dass durch die für gewöhnlich hohen Übertragungsraten auf den Grafikkartenbussen (insbesondere bei den heutzutage aktuellen, auf Übertragungsleistung optimierten Bussystemen wie 8x-, 16x-AGP oder PCI-Express) schon in sehr kurzer Zeit beträchtliche Datenvolumen anfallen. Sowohl die Speicherung als auch die nachfolgende Auswertung dieser Daten gestalten sich daher bereits sehr aufwändig.

Komfortabler ist hierbei eine Protokollierung, welche auf bestimmte Schlüsselsequenzen oder -worte zur Auslösung und Beendigung der Protokollierung reagiert. Dies reduziert das anfallende Datenvolumen beträchtlich und schränkt die Suche nach Strukturen auf relevante Abschnitte ein. Allerdings setzt dieses Vorgehen voraus, dass hierfür zumindest Teile der Übertragung (mindestens die Schlüsselworte) bereits im Vorfeld bekannt sind.

4.1.3.2 Provokateur

Ein „Provokateur” arbeitet ähnlich zu einem (Software-)Sniffer, kann aber nicht nur passiv den auftretenden Datenverkehr abhorchen, sondern aktiv und gezielt einzelne Befehle an die Grafikkarte absetzen und die Reaktion darauf bzw. die dadurch entstehende Kommunikation mitverfolgen. So können Vergleiche zwischen dem abgesetzten Befehl und den sich daraus ergebenden Datenübertragungen angestellt werden, um so Rückschlüsse auf die Bedeutung dieser Daten ziehen zu können. Ein Provokateur eignet sich daher besonders für die Analyse von Arbeitsweisen und eventueller Umsetzungen, welche innerhalb eines bestimmten Programmabschnittes vorgenommen werden – beispielsweise innerhalb eines Binärtreibers –, indem er die Eingaben vor dem Programmabschnitt und die Resultate danach extrahieren und miteinander in Verbindung setzen kann.

Ein Provokateur hilft daher sowohl beim Analysieren unbekannter Befehle, als auch insbesondere bei der Untersuchung der Auswirkungen unterschiedlicher Parameter von nur teilweise bekannten Befehlen, indem eine automatisierte Erkennung von Unterschieden zur Funktionsweise dieser Befehle, ausgelöst durch Variationen bei den Parametern, miteinander verglichen werden können.

4.1.3.3 Analyser / Profiler

Analyser bzw. Profiler können auf höheren Schichten eingesetzt werden, wenn eine vollständige oder hardwarenahe Protokollierung aller anfallender Daten oder die genaue Auswirkung von Befehlen bei der Kommunikation mit der Grafikkarte nicht vonnöten ist, sondern es um die Bestimmung verschiedener (Einzel-)Aspekte zur Laufzeit geht.

So kann mit einem Analyser bzw. Profiler beispielsweise der zeitliche Ablauf von einzelnen Zugriffen auf bestimmte Bereiche ermittelt werden, wie sie für das Erstellen oder Nachvollziehen eines Timing-Verhaltens eventuell erforderlich sind. Hierzu werden die betroffenen Bereiche überwacht, indem alle Zugriffe darauf abgefangen und protokolliert werden.

Ebenso können von diesen Analyse-Programmen durch die Überwachung ganzer Speicherbereiche darin befindliche Strukturen aufgespürt werden. Über die Art der Zugriffe, der zugreifenden Geräte und der Adressbereiche, in dem diese geschehen, können so z.B. vorhandene Puffer gefunden und deren Organisation ermittelt werden (lineare Puffer, Ringpuffer, ...).

Werden diese Analysen auch auf die Inhalte der Daten in Pufferbereichen ausgedehnt, so können zusätzlich Erkenntnisse über deren Aufbau gewonnen werden, wie z.B. die Zusammensetzung von Kommandopaketen im Befehlspuffer einer Grafikkarte oder die Paketabfolge von mehreren Kommandos bei einer ganzen Befehlssequenz.

4.1.3.4 Wrapper

Falls sich bestimmte Funktionen nicht entschlüsseln lassen oder eine Analyse zu aufwändig wird kann es sinnvoll sein, doch die bestehenden (Binär-)Treiber des Herstellers zu verwenden, indem diese in das eigene System eingebettet werden. Hierzu muss eine Nachbildung aller der von dem Treibermodul genutzten Systemaufrufe und Kommunikationsarten über Wrapper-Methoden und -Strukturen vorgenommen werden. Für Treiber, welche nur wenige unterschiedliche Systemaufrufe tätigen, ist diese Vorgehensweise durchaus praktikabel.

Speziell für aktuelle Grafikkartentreiber ist dieses Vorgehen aber nur mit sehr viel Aufwand zu bewerkstelligen, da die Grafikkartentreiber sehr komplex aufgebaut sind und bis zu mehreren Dutzend verschiedener Systemaufrufe und spezifische Speicherstrukturen benötigen, welche sich schon allein aufgrund der Vielzahl an benötigten Aufrufen nur schwer nachbilden lassen. Ein Versuch, dies zu unternehmen, kann im Extremfall bis zur kompletten Einbindung bzw. Nachbildung des ursprünglichen Wirtsystems führen, um die herstellerspezifischen Treiber überhaupt verwenden zu können.

Einem Einsatz von Wrappern sollte in jedem Fall eine Abschätzung vorausgehen, ob die Nachbildung der nötigen Strukturen mit einem vertretbaren Aufwand durchführbar ist bzw. ob dies wegen eventuell inkompatibler Laufzeitstrukturen überhaupt möglich ist, oder ob die zuvor genannten Reverse-Engineering-Techniken nicht schneller zum gewünschten Erfolg führen.

Eine tiefergehende Betrachtung dieser Reverse-Engineering-Techniken und empfohlener Vorgehensweisen sowie eine konkrete Beschreibung zu einzelnen Werkzeugen und Programmen, insbesondere in Bezug auf die Analyse von Grafikkartenfunktionen, findet sich in [56].

4.2 Verteilung von Grafikkartenspeicher

Ein erster Schritt zur direkten Integration von Grafikkartenspeicher innerhalb eines transaktionalen Szenengraphens ist die Einbeziehung des lokal vorhandenen Grafikkartenspeichers in die Verteilung des bislang verwendeten VVSs. Hierbei soll allerdings nicht nur ein gemeinsamer aber zum bestehenden verteilten (Haupt-)Speicher separater Speicher nur für die Verteilung des Grafikkartenspeichers über mehrere Grafikkarten hinweg aufgebaut werden, über welchen von diesen Grafikkarten die Grafikdaten untereinander ausgetauscht und dadurch eine Art von verteiltem gemeinsamem Pixelspeicher ('Distributed Shared Pixel Memory', DSPM) umgesetzt würden. Vielmehr ist weitreichender die vollständige Einbeziehung des Grafikkartenspeichers innerhalb des (Hauptspeicher-)VVSs gemeint. Der Grafikkartenspeicher stellt in diesem Zusammenhang somit eine Erweiterung bzw. Vergrößerung des Hauptspeichers dar, in dem theoretisch auch beliebige andere Objekte abgelegt werden können (d.h. insbesondere auch Nicht-Grafikobjekte).

Ein wesentlicher Vorteil, den eine direkte Einbeziehung des Grafikkartenspeichers in einen verteilten Speicher bieten würde, wäre, die von einer Grafikkarte zur Darstellung benötigten (Grafik-)Objekte eines Szenengraphens nicht mehr über den Hauptspeicher der jeweiligen Station zwischenzuspeichern und vorhalten zu müssen. Diese könnten folglich direkt im Grafikkartenspeicher angesiedelt werden, von wo aus sie dann ohne Umwege von der Grafikkarte angesprochen und verarbeitet werden könnten. Darüber hinaus könnten diese Daten aber ebenfalls durch die bereitgestellte Verteilung unmittelbar anderen Stationen bzw. deren Grafikkarten(-speichern) zum direkten Zugriff zur Verfügung stehen.

Als Anwendungsbeispiel könnte so etwa bei einem Videochat auf die eingehenden Videodaten im Grafikkartenspeicher einer an die Grafikkarte angeschlossenen Webkamera direkt von anderen Stationen aus zugegriffen und diese an sie weitergeleitet werden, ohne dass wie in bisherigen Verfahren zuerst in einem zusätzlichen Schritt die Videodaten von der CPU in den Hauptspeicher umkopiert werden müssen, damit sie erst für die anderen Stationen abrufbar werden und wiederum auf einem Zielrechner zuerst in dessen Hauptspeicher landen, bevor sie zur eigentlichen Darstellung erneut in die dort vorhandene Grafikkarte bzw. deren Speicher umkopiert werden müssen.

Die heutzutage vorhandene Hardware ist für eine solche Nutzung allerdings nicht unmittelbar ausgelegt. Bei der Arbeitsweise innerhalb eines Rechnersystems werden die von der Grafikkarte zur Darstellung benötigten Grafikdaten standardmäßig immer im Hauptspeicher vorgehalten, um diese dann erst bei Bedarf aktiv, d.h. von der CPU aus veranlasst, in den Grafikkartenspeicher kopieren zu können. Um eventuelle Verzögerungen, die sich durch diesen Transfer ergeben können, zu vermeiden, werden daher in einzelnen Ansätzen auch bereits im Vorfeld spekulativ zusätzliche Grafikdaten transferiert, welche später mit einer gewissen Wahrscheinlichkeit benötigt werden. Dies kann aber auch dazu führen, dass diese vorsorglich übertragenen Daten oder Teile davon doch nicht gebraucht werden und dadurch einerseits unnötig übertragen wurden und damit Laufzeit beansprucht haben, als auch andererseits unnötig Platz belegen, der für neue Daten eventuell erst wieder freizugeben ist.

4.2.1 Arbeitsweise des Grafikkartenspeichers

Um die erforderlichen Aspekte, die sich durch die Verteilung des Grafikkartenspeichers ergeben, und um die darauf möglichen bzw. notwendigen Zugriffsarten besser nachvollziehen zu können, ist zu Beginn dieser Betrachtung eine Beschreibung des Grafikkartenspeichers und seiner Ansteuerung erforderlich.

Die Grafikdaten, mit denen eine Grafikkarte arbeitet, werden in der Regel separat zu den Kommandopaketen an die Grafikkarte übertragen. Normalerweise geschieht dies durch einfaches Kopieren von Speicherinhalten aus dem Hauptspeicher in den Grafikkartenspeicher und wird von der CPU aus im Vorfeld ihrer Verwendung initiiert. Die Grafikkarte erhält dann zur weiteren Verarbeitung dieser Daten nur noch einen Verweis auf die kopierten Stellen in ihrem Grafikkartenspeicher und entnimmt von dort die benötigten Daten.

Der Grafikkartenspeicher ist hierzu üblicherweise in einen Bereich des normalen (Speicher-)Adressraumes eines Rechners eingeblendet ('Memory Mapping'), um ihn für den Prozessor wie bei einem gewöhnlichen (Haupt-)Speicherzugriff ansprechbar zu machen. Auf diese Weise macht es für eine CPU keinen Unterschied, ob Daten in den Hauptspeicher oder in den Grafikkartenspeicher abgelegt werden. Der Bereich, in dem der Grafikkartenspeicher eingeblendet wird, befindet sich für gewöhnlich am Ende des adressierbaren Bereichs, um nicht in Konflikt mit dem tatsächlich installierten Speicher zu kommen, kann aber auch an frühere Stellen eingeblendet werden[37]. Es können hierdurch Lücken im (physikalischen) Adressraum zwischen dem Bereich des tatsächlich installierten Speichers und des eingeblendeten Speichers entstehen. Die CPU muss in diesen Fällen selbst dafür sorgen, dass keine Zugriffe darauf stattfinden bzw. es wird von der Speicher- und Seitenverwaltung darauf geachtet, dass für die CPU ein linearer bzw. virtueller Adressraum zur Verfügung steht und gegebenenfalls automatisch eine entsprechende Abbildung auf den physikalischen (Haupt-)Speicher oder auf eingeblendete Gerätespeicher – je nach angesprochener Adresse – stattfindet.

Beim Zugriff auf den Grafikkartenspeicher muss nun weiter eine Unterscheidung vorgenommen werden, ob dieser – wie eben beschreiben – von außerhalb der Grafikkarte durch die CPU vorgenommen wird, oder von innerhalb der Grafikkarte vom Grafikprozessor (GPU) stammt. Die GPU kennt im Gegensatz zur CPU für den Grafikkartenspeicher nur einen einzigen Adressbereich mit physikalischer Adressierung und kann auch nur diesen ansprechen. Dieser GPU-Adressraum unterscheidet sich dabei von dem physikalischen Adressraum aus Sicht der CPU. Für die GPU müssen die physikalischen Adressen einen zusammenhängenden Bereich bilden; „Löcher" im Adressbereich duldet bzw. kennt ein Grafikprozessor nicht. Es gibt umgekehrt auch bislang keine eigene Grafikkartenspeicherverwaltung, welche ein (dynamisches) Abbilden von Grafikspeicheradressen zu existierenden Speicherstellen vornehmen könnte, wodurch sich logische oder virtuelle Grafikkartenadressen bilden ließen; der Grafikkartenspeicher ist direkt auf den GPU-Adressraum abgebildet.

Eine Technik, die eingeführt wurde, um eine Erhöhung des für eine Grafikkarte vorhandenen lokalen Grafikspeichers zu ermöglichen, ohne dass dieser tatsächlich vorhanden sein muss – um beispielsweise mehr bzw. größere Textur- oder Vertexdaten ansprechen und zugreifbar machen zu können –, ist eine Erweiterung des Speicherzugriffs der Grafikkarte über die 'Graphics Address Remapping Table' (GART)[38]. Mit Hilfe der GART bzw. des hierfür zuständigen speziellen Speicherkontrollers innerhalb der Grafikkarte, welcher die GART beinhaltet, ist es für eine Grafikkarte nun möglich, ähnlich zu einem Speichereinblendungsverfahren innerhalb des Adressraums der CPU, (physikalischen) Hauptspeicher als Erweiterung ihres lokalen

37 Diese Einblendung wird zumindest in einem 32-Bit-Adressraum so für gewöhnlich vorgenommen, welcher maximal 4 GiB an Speicher adressieren kann. Sollte ein Rechner mit einem 32-Bit-Adressraum tatsächlich 4 GiB an Hauptspeicher installiert haben, so können Teile davon durch diese Einblendungen nicht mehr adressiert werden. Mit dem 64-Bit-Adressraum moderner Standardrechner und Betriebssysteme kann hingegen der volle Speicherumfang durch den größeren Adressierungsbereich nutzbar gemacht werden.

38 Insbesondere zu Zeiten, in denen der Grafikkartenspeicher relativ teuer war, aber auch bei Laptops mit mobilen Grafikkartenversionen ohne eigenen oder mit nur sehr geringem vorhandenen Grafikkartenspeicher, wurde dieses Vorgehen intensiv genutzt, um die Kosten für solche Systeme bzw. deren Grafikkarten gering zu halten.

Grafikkartenspeichers zu verwenden. Die Größe dieses zusätzlichen Bereichs kann dabei über die GART eingestellt werden; sie bezieht sich aber immer auf einzelne Kacheln in der vom Rechner verwendeten Seitengröße (in der Regel 4 KiB-Kacheln). Die verwendeten Hauptspeicherkacheln müssen dabei nicht einmal zusammenhängend sein, sondern können beliebig fragmentiert vorliegen. Die Umsetzung zu einem zusammenhängenden Speicherbereich für die Sicht der GPU nimmt der GART-Kontroller transparent vor (siehe Abbildung 42).

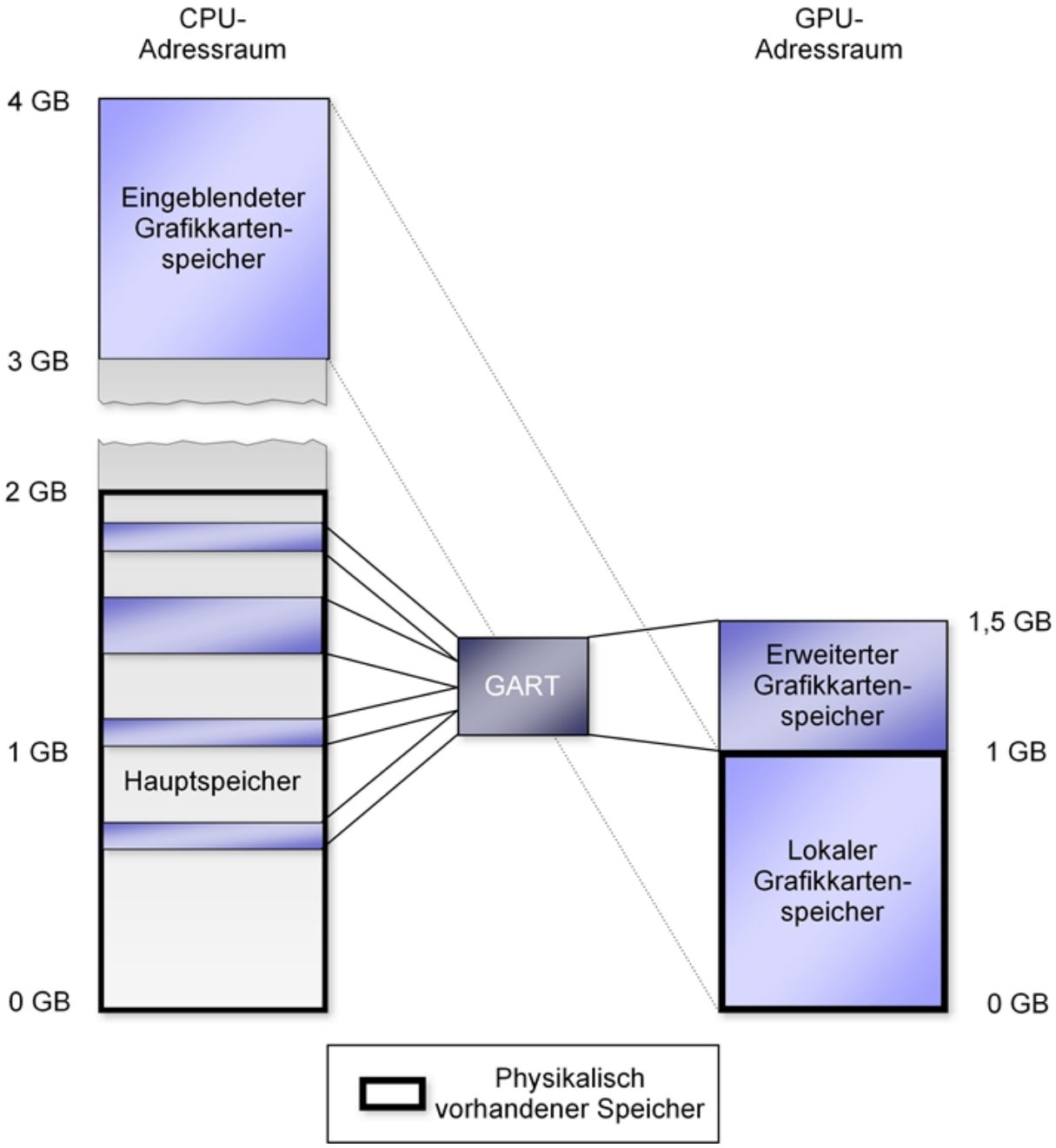

Abbildung 42: Speicheraufbau aus Sicht der CPU und GPU

Dieser hinzugefügte Hauptspeicherbereich wird dabei aus Sicht der GPU vom GART-Kontroller direkt an das Ende des vorhandenen lokalen Grafikspeichers angefügt. Für den Grafikprozessor ist

dieser erweiterte Bereich nicht vom lokalen Grafikkartenspeicher unterscheidbar und daher genauso ansprechbar, als ob tatsächlich mehr lokaler Grafikkartenspeicher installiert wäre und ohne dass Unterscheidungen während des Zugriffs hierauf nötig wären. Allerdings ist die Zugriffszeit der GPU auf Daten innerhalb dieses erweiterten Bereichs wesentlich höher als innerhalb des lokalen Grafikkartenspeichers, weshalb diese Technik in der Regel nur für zusätzliche Texturen eingesetzt wird, auf die nicht ständig zugegriffen wird, um die Leistung einer Anwendung nicht unnötig auszubremsen.

4.2.2 Verteilter Grafikkartenspeicher

Damit (Grafik-)Objekte über einzelne Grafikkarten und deren Speicher hinweg überhaupt verteilt und genutzt werden können, sind mehrere Aspekte hierzu relevant, welche nun genauer betrachtet werden.

4.2.2.1 Adressraum

Um verteilte (Grafik-)Objekte in einem VVS einheitlich ansprechen zu können, sind virtuelle Adressen vonnöten (siehe Kapitel 1.3). Der virtuelle Adressraum sorgt dafür, dass verteilte Objekte, die von mehreren Stationen gemeinsam verwendet werden, auch von jeder Station aus über dieselben einheitlichen, virtuellen Adressen referenzierbar sind. Die lokalen Speicherverwaltungen jeder Station sorgen ihrerseits dafür, dass diese virtuellen Adressen auf gültige, lokale Arbeitskopien der gemeinsamen Objekte verweisen, die sich dabei auf beliebigen physikalischen Seiten bzw. Kacheln im jeweiligen Rechner befinden können, je nachdem wo bei einer Station zum Zeitpunkt der Speicherzuordnung innerhalb des physikalischen Speichers gerade freie Bereiche für die Benutzung zur Verfügung standen.

Umgekehrt sorgen auch die Speicherverwaltungen bei einem Zugriff auf ein nicht lokal vorhandenes Objekt dafür, dass dieses automatisch von einer anderen Station bzw. dessen Besitzer angefordert und verfügbar gemacht wird. Dies geschieht unter Zuhilfenahme eines Seitenfehlermechanismus, mit dem eine Speicherverwaltung einen Zugriff der CPU auf eine virtuelle Adresse eines nicht lokal vorhandenen Objekts abfängt, in einer Unterbrechung das fehlende Objekt von anderen Rechnern anfordert, nach dessen Erhalt bei sich einlagert und eine entsprechende, nun gültige Abbildung zwischen der physikalischen Adresse des tatsächlichen Speicherortes und der abgefangenen virtuellen Adresse der CPU vornimmt. Die CPU kann anschließend normal mit ihrem Zugriff fortfahren; der gesamte vorherige Ablauf bleibt für sie transparent durch die Speicherverwaltung verborgen. Um dabei eine größtmögliche Systemleistung zu erzielen, wird die Adressumsetzung von virtuellen zu physikalischen Adressen und der Seitenfehlermechanismus in der Regel direkt in der Hardware von der MMU ausgeführt. Mit Hilfe dieses Mechanismus lässt sich auch eine weitergehende Verteilung von Objekten mittels eines VVS leicht und effektiv umsetzen (siehe Kapitel 1.5).

Soll nun zusätzlich lokaler Grafikkartenspeicher genauso wie bislang nur normaler Hauptspeicher in den VVS integriert werden, so sind hierbei einige Besonderheiten zu beachten. Um beispielsweise Szenengraphenobjekte direkt in den Grafikkartenspeicher ablegen zu können, ohne ein vorausgehendes, lokales Umkopieren aus dem Hauptspeicher in den Grafikkartenspeicher zu benötigten, muss für einzulagernde Objekte von der Speicherverwaltung eine Unterscheidung des Zielortes vorgenommen werden. Da bisherige Speicherverwaltungen allerdings nur für eine Nutzung innerhalb des bzw. mit dem Hauptspeicher ausgelegt sind, aber nicht für die Verwaltung von zusätzlich vorhandenem, lokalem Gerätespeicher, stellt der Grafikkartenspeicher nun neben dem bisherigen Hauptspeicher einen zweiten, separaten Speicherort dar, der auch separat behandelt

und angesprochen werden muss. Der Grafikkartenspeicher selbst ist zwar für gewöhnlich in den Hauptspeicheradressraum an definierbarer Stelle eingeblendet (siehe Kapitel 4.2.1), so dass von der Speicherverwaltung keine zusätzliche Adressraumumschaltung für einen Zugriff darauf vorgenommen werden muss, allerdings ist die Art der Zuordnung zwischen virtuellen und physikalischen Adressen für die beiden Speicherorte verschieden. Bei der normalen Einlagerung eines Objektes in den Hauptspeicher wird von der Speicherverwaltung eine freie Kachel des Hauptspeichers verwendet und anschließend mit der virtuellen Adresse des Objektes verknüpft. Für den Grafikkartenspeicher kann dieses Vorgehen so allerdings nicht direkt übernommen werden, sondern es dürfen nur diejenigen Kacheln ausgewählt werden, die auch mit dem eingeblendeten Adressbereich des Grafikkartenspeichers korrelieren, damit die dort abgelegten Daten letztlich auch von der Grafikkarte zugreifbar sind. Für zu verteilende Grafikobjekte steht also im Vergleich zu normalen Objekten nur eine Untermenge des Hauptspeichers mit fest vorgegebenen Speicherorten zur Verfügung. Eine Speicherverwaltung, welche dies umsetzen soll, muss daher die einzelnen Gerätespeicherbereiche klar vom restlichen Speicher abgegrenzt behandeln und verwalten.

Die Unterscheidung des Speicherortes für die Speicherverwaltung wird dabei durch eine parallele Verwendung der GART durch die Grafikkarte weiter erschwert. Die GART erweitert zwar die Sicht einer Grafikkarte bzw. der GPU auf zusätzliche Bereiche des Hauptspeichers, um so einen größeren lokalen Grafikkartenspeicher zu simulieren. Aus der Sicht eines VVS bzw. der Speicherverwaltung des Systems hingegen ist dieser von der GART zusätzlich angesprochene Hauptspeicher allerdings bereits im bislang verfügbaren Speicherbereich enthalten; nur der lokale Grafikkartenspeicher selbst kommt tatsächlich neu hinzu. Eine GART bringt somit für die Speicherverwaltung keinen zusätzlichen Speicher in das System bzw. den VVS mit ein, sie verändert lediglich die Sichtweise der Grafikkarte auf den von ihr nutzbaren Speicherbereich. Eine Speicherverwaltung muss folglich bei gleichzeitig aktivierter GART von der Grafikkarte wiederum weitere Hauptspeicherblöcke unterschiedlich behandeln, die sogar beliebig innerhalb des normalen Hauptspeichers verstreut sein können, da eine GART genau diese Nutzung für die Grafikkarte ermöglicht. Prinzipiell ist es daher nicht sinnvoll, eine GART parallel zu einer Verteilung des Grafikkartenspeichers zu verwenden, da dies kein Zugewinn an Speicher für eine Speicherverwaltung bedeutet, die Arbeitsweise von ihr dadurch aber erschwert wird.

Damit nun eine Speicherverwaltung eine solche Unterscheidung der Speicherorte innerhalb eines Rechners vornehmen kann, müssen Modifizierungen vorgenommen werden. Beispielsweise kann dies in der Art erfolgen, dass verschiedene Speicherpools von der Speicherverwaltung verwendet werden, welche für den normalen Hauptspeicher, den Grafikkartenspeicher und eventuell auch für die von einer GART angesprochen Speicherbereiche stehen. Die Speicherverwaltung entscheidet dann bei jeder Einlagerung eines Objektes in den lokalen Speicher einer Station, in welchem Pool dieses Objekt angesiedelt wird, und kann dementsprechende Arten der Zuweisung von physikalischen Kacheln dafür vornehmen.

Zu diesem Zweck müsste aber auch bei den Objekten eine von der Speicherverwaltung erkennbare Unterscheidung auf die Art des Objektes eingeführt werden, um überhaupt eine korrekte Zuordnung zu den jeweiligen Speicherpoolen zu ermöglichen. So könnten von der Anwendung explizite Typen wie Grafikobjekte kreiert werden, welche nur für die Einlagerung innerhalb eines Grafikkartenspeichers bestimmt sind; alle normalen und restlichen Objekttypen werden dahingegen wie gewohnt innerhalb des Hauptspeichers abgelegt.

Eine weitere Modifikation betrifft die Freigabe von belegten Kacheln. Normalerweise kümmert sich eine Standardspeicher- und -seitenverwaltung eines Rechners selbst nicht aktiv um das Freigeben von nicht mehr benötigten Speicherseiten. Dies ist entweder die Aufgabe einer Garbage Collection oder eines Programmierers bzw. der Anwendung selbst, welche nicht mehr benötigte Objekte bzw.

Seiten explizit wieder freigibt. Sollte kein (physikalischer) Speicher für die Einlagerung von Objekten mehr verfügbar sein, so wird im besten Fall von der Speicherverwaltung die Garbage Collection gestartet oder mit anderen Mechanismen versucht, belegten Platz wieder freizugeben oder Relozierungen vornehmen zu lassen. Im ungünstigsten Fall gibt es Zugriffsfehler, welche zu unvorhersehbaren Ausführungen der CPU bis hin zum Anhalten des Rechner führen, weshalb dieser Zustand typischerweise für eine Speicherverwaltung bereits von Anwendungs- bzw. Betriebssystemseite her vermieden wird. Bei Grafikkartenspeichern hat diese Situation dahingegen andere und weit weniger schwerwiegende Auswirkungen. Da die verwendeten Grafik- und Texturdaten einer Anwendung oftmals deutlich umfangreicher als der in einer Grafikkarte zur Verfügung stehende Grafikkartenspeicher sind, kommt die eben beschriebene Situation, dass der Grafikkartenspeicher komplett gefüllt ist und somit weitere Daten keinen Platz mehr finden, vergleichsweise häufig vor. In der Regel arbeitet eine Grafikkarte auch mit nicht vorhandenen Texturdaten weiter; die berechneten Bilder haben in diesen Fällen lediglich falsche Farben bzw. Texturen oder es werden nicht alle eigentlich sichtbaren Objekte angezeigt, wenn der GPU nicht alle Vertexdaten zur Berechnung zur Verfügung stehen. Die Grafikengine der Anwendungssoftware oder der Grafikkartentreiber – welche beide von der CPU ausgeführt werden – kümmert sich dabei üblicherweise darum, dass alle von der GPU benötigten Grafikdaten auch rechtzeitig auf der Grafikkarte vorhanden sind. Bei zu wenig Platz wird dann auf geringere Detailstufen mit weniger Platzbedarf für die Grafikobjekte und Texturen umgeschaltet, es werden ältere Daten überschrieben oder die benötigten Grafikdaten zwischen dem Hauptspeicher und dem Grafikkartenspeicher ständig umgelagert, was sich dann allerdings auch in deutlich niedrigeren Bildwiederholraten niederschlägt. Ein Fehlverhalten bezüglich nicht vorhandener Texturdaten der Grafikkarte wegen vollem Speicher beeinflusst die Rechnerstabilität im Gegensatz zur normalen Hauptspeicherverwaltung aber üblicherweise nicht nachhaltig und ist zumeist mit dem nächsten dargestellten Bild bereits wieder behoben. Bei einer Übernahme der Verwaltung des Grafikkartenspeichers durch die Systemspeicherverwaltung (und nicht mehr länger durch die Anwendung, d.h. deren Grafikengine, oder einem Treiber) müssen dann diese unterschiedlichen Verhaltensweisen innerhalb der jeweiligen Speicherpools ebenfalls berücksichtigt werden.

Besondere Beachtung muss dabei dem Umstand gewidmet werden, dass Grafikobjekte für die GPU von einer Speicherverwaltung – im Gegensatz zur Vorgehensweise bei normalen Hauptspeicherkacheln – nicht beliebig eingelagert bzw. wieder freigegeben werden können. Die GPU selbst kennt ausschließlich ihren physikalischen und insbesondere kontinuierlichen Adressraum. Sie erwartet daher, dass vorhandene Grafikdaten auch immer kontinuierlich in ihrem Speicher eingelagert wurden und nicht fragmentiert vorliegen. Da die GPU zusätzlich auch immer von vollständig verfügbaren Grafikdaten ausgeht – gleichgültig ob sie diese zur Gänze benötigt oder nicht –, kann (und braucht) sie selbst nicht unterscheiden, ob einzelne Kacheln innerhalb des von diesen Grafikdaten belegten Bereichs etwa inzwischen anderweitig durch ein Aus- bzw. Umlagern von weiteren Grafikobjekten genutzt wird. Dies hat zur Folge, dass von einer Speicherverwaltung die in die Grafikkarte einzulagernden Grafikobjekte nicht beliebig und über freie Kacheln verteilt in den Grafikkartenspeicher eingelagert werden dürfen, sondern dass dabei für eine lineare und kontinuierliche Anordnung gesorgt werden muss. Auch können so umgekehrt bei zu wenig Platz nicht einfach nur einzelne Kacheln überschrieben und neu genutzt werden, wenn die GPU beispielsweise nur kleine Teile einer großen Textur verwendet. Da in diesen Fällen die Speicherverwaltung üblicherweise keine Informationen darüber besitzt, auf welche Teile der Grafik- bzw. Texturobjekte eine GPU letztlich zugreifen wird, erfordert eine eventuelle Anpassung der Speicherverwaltung hierzu eine vorhergehende Analyse bezüglich des Nutzungsverhaltens der Grafikdaten, um so eventuell nicht doch alle Daten immer vollständig einlagern zu müssen.

Bei (Haupt-)Speicherbereichen, die über eine GART einer Grafikkarte zugänglich gemacht werden,

muss zusätzlich die von der GART erzeugte weitere Indirektion für die von ihr angesprochenen Adressbereiche berücksichtigt werden. Die GART erlaubt eine fragmentierte Zuteilung der zugehörigen Hauptspeicherkacheln bei der insbesondere auch die Reihenfolge der Kacheleinblendungen relevant ist, da über sie erst ein für die Grafikkarte kontinuierlich adressierbarer Bereich geschaffen wird. Bei der Übernahme dieser Aufgabe von einer Speicherverwaltung wird dadurch die von ihr zu bewältigende Komplexität weiter erhöht.

Insgesamt kann somit die Arbeitsweise von bestehenden Systemspeicherverwaltungen nicht direkt auf lokale Gerätespeicher übernommen werden, da durch die Unterscheidung von normalen und Grafikobjekten auch unterschiedliche Mechanismen zu deren Adressierung und für zugehörige Ein- bzw. Auslagerungsstrategien verwendet werden müssen, wie sie für eine Nutzung im Rahmen eines transaktionalen Szenengraphens benötigt werden und daher mit zu berücksichtigen sind.

4.2.2.2 Anforderung von Daten

Heutige Grafikkarten sind von ihrem Design und ihrer Arbeitsweise her darauf ausgelegt und optimiert, Befehle und Grafikdaten lediglich zu empfangen, darauf Berechnungen auszuführen und die Ergebnisse an ein Anzeigegerät zu liefern bzw. in ihren Grafikkartenspeicher abzulegen. Dies äußert sich erwartungsgemäß auch im Hardwaredesign von Grafikkarten. Eine GPU kennt beispielsweise nur den integrierten Grafikkartenspeicher, in dem alle für sie relevanten Daten vorhanden sein müssen und greift auf diesen über ihren eigenen Adressraum mit ausschließlich physikalischen Adressen und kontinuierlichen, d.h. hierbei unfragmentierten, Grafikdaten zu. Da sie ferner keine eigene Speicherverwaltung oder MMU besitzt, ist für die GPU auch kein Mechanismus vorgesehen – und vom Hardwaredesign her auch nicht vorhanden –, um mit dem restlichen System bzw. der CPU oder der Systemspeicherverwaltung Informationen direkt auszutauschen. Eine Rückmeldung an den Rechner oder die CPU und somit eine bidirektionale Kommunikation ist für die eigentliche Arbeitsweise der Grafikkarte nicht erforderlich und somit auch nicht ausgeprägt vorhanden[39]. Aus diesem Grund kann eine GPU auch von sich aus keine selbsttätigen Anfragen nach benötigten, aber momentan eventuell noch nicht vorhandenen Grafik- bzw. Seitendaten außerhalb ihres Adressbereiches vornehmen, um so beispielsweise autonom ein neues Bild oder Szenario mit geänderten Daten rendern zu können. Diese geänderten Grafikdaten müssen daher bislang immer jeweils explizit von außerhalb, d.h. über die CPU, an die Grafikkarte geliefert werden.

Infolgedessen muss auch die Verwaltung der von einer Grafikkarte benötigten Daten explizit vorgenommen werden. In der Regel wird dies vom Anwendungsprogramm bzw. deren Grafikengine durchgeführt. Diese überwacht die für eine Szene benötigten Grafik- und Texturdaten und transferiert sie in den Grafikkartenspeicher bzw. weist die bestehende Speicherverwaltung hierzu an. Da allerdings auch die Grafikengine nicht immer im Voraus exakt weiß, welche Grafikdaten genau von einer GPU jeweils als nächstes benötigt werden, werden im Zweifelsfall immer alle eventuell benötigten Daten in den Grafikkartenspeicher transferiert, auch auf die Gefahr hin, dass einige dieser Kopieroperationen womöglich unnötig waren. Durch den Einsatz von Szenengraphen werden die hierfür ausgewählten Daten zwar bereits wesentlich eingeschränkt und dadurch dieses Vorgehen optimiert; erst die Grafikkarte selbst erkennt aber letztlich durch die von ihr vorgenommenen Sichtbarkeitsprüfungen, ob sich ein bestimmtes Objekt tatsächlich vollständig im

39 Grafikkarten können in der Regel zwar oftmals über Interrupts Rückmeldungen an die CPU liefern (dies stammt noch von der Ansteuerung früherer Grafikkartengenerationen innerhalb eines Rechnersystems); eine Ansteuerung der Grafikkarte hierüber ist aber im Vergleich zum Kommandopaket-Modus von der erreichbaren Leistung her deutlich schlechter und es tritt insbesondere eine weitere Beeinträchtigung des restlichen Systems auf, da von der CPU zusätzliche Zeit durch die Interrupt-Behandlung verwendet werden muss. Typischerweise werden daher Interrupts bei heutigen Grafikkarten nicht mehr genutzt.

Sichtbarkeitsbereich des Anwenders befindet und somit auch weitere Texturen zum Rendern benötigt werden, oder ob Teile davon nicht dargestellt werden müssen und somit verworfen werden können und dadurch auch keine diesen Teilen zugehörigen Texturen in der Grafikkarte vorgehalten werden müssen.

Durch den Einsatz verschiedener Heuristiken von der Grafikdatenverwaltung, um mit einer gewissen Wahrscheinlichkeit benötigte Grafikdaten mittels 'Prefetching' bereits vorzeitig in den Grafikkartenspeicher einzulagern, lässt sich zwar die Effizienz einer grafischen Anwendung weiter erhöhen; der eigentliche Punkt, dass typischerweise immer noch zu viel bzw. unnötige Daten umkopiert werden und dadurch auch weitere Ressourcen des Gesamtsystems beanspruchen, wird dadurch aber nicht gelöst.

Erst wenn eine Grafikkarte selbständig benötigte Grafikdaten anfordern kann, lässt sich eine deutliche Vereinfachung dieser Vorgehensweise erreichen. Hierzu ist ein beliebiger Rückkanal von der Grafikkarte an das System erforderlich, beispielsweise durch Nachrichten von der GPU an das System oder gar einem eigenen Speicherkontroller der Grafikkarte. Dies würde dann eine Arbeitsweise analog zu der System-MMU innerhalb der Grafikkarte ermöglichen, welche dann autonom fehlende Seiten anfordern könnte und darüber hinaus auch die Prefetching-Operationen einsparen ließe. Solange allerdings kein geeignetes und effektives Verfahren für eine Rückmeldung der GPU an die CPU oder der Speicherverwaltung existiert, wird die Aufgabe der Grafikkartenspeicherverwaltung weiterhin extern unter Verwendung der CPU vorgenommen werden müssen, indem z.B. die CPU periodisch bei der Grafikkarte überprüft, ob neue Grafikdaten benötigt und eingelagert werden müssen.

4.2.2.3 Verteilung

Bei den vorhergehenden Aspekten innerhalb eines Rechnersystems kann eine eventuelle Anpassung einer vorhandenen Speicherverwaltung eventuell teilweise durch die System-MMU unterstützt werden. Sobald aber darüber hinaus eine Verteilung der in den Grafikkarten eingelagerten Grafikobjekte über mehrere Stationen hinweg vorgenommen wird, reicht der hiervon angebotene Mechanismus nicht mehr aus, die zusätzlichen Verteilungsaspekte zu berücksichtigen. Die Verteilung muss daher explizit von der Anwendungssoftware oder einer eingefügten Schicht bzw. Middleware bewerkstelligt werden. Dies ist insbesondere dann erforderlich, wenn die Grafikdaten nicht wie von traditionellen Ansätzen gewohnt zuerst im Hauptspeicher jedes Rechners vorgehalten werden sollen, um sie dann erst beim eigentlichen Bedarf in die Grafikkarte lokal umzukopieren, sondern sie sich bereits direkt in einem Grafikkartenspeicher befinden, wodurch die dadurch realisierte Erweiterung zum vorhandenen Hauptspeicher einem NUMA[40]-Ansatz entspricht.

Hierzu muss die Speicherverwaltung eine Unterscheidung vornehmen, ob eine angesprochene virtuelle Adresse ein normales Objekt referenziert, welches dann bei einer lokalen Einlagerung wie gewohnt einer freien Speicherkachel im Hauptspeicher zugeordnet werden kann, oder ob es sich um ein Grafikobjekt des Szenengraphens handelt. Dieses darf dann nur im Grafikkartenspeicher oder den mittels einer GART der Grafikkarte eingeblendeten, physikalischen Kacheln abgelegt werden, wenn gerade diese zusätzlichen Umkopieroperationen vom Hauptspeicher aus in den Grafikkartenspeicher vermieden werden sollen.

Eine Schwierigkeit, die es hierbei zu beachten gilt, tritt beispielsweise in der Situation auf, wenn

40 'Non-Uniform Memory Architecture'. Dies beschreibt eine (Speicher-)Architektur, bei der unterschiedliche Zugriffszeiten auf einzelne Speicherinhalte auftreten, je nachdem an welchem Ort sich die zugegriffenen Daten befinden, trotz einer einheitlichen Art des Speicherzugriffs für eine Anwendung. Die Latenz hängt in diesem Fall davon ab, ob auf einen lokalen oder einen entfernten Speicherort – d.h. auf einen Gerätespeicher – zugegriffen wird.

dabei mehr Grafikobjekte in einem Szenengraphen vorkommen, als gemeinsamer Grafikkartenspeicher insgesamt zur Verfügung steht. Es muss dann die Frage behandelt werden, wie mit den restlichen Grafikobjekten, für welche kein Platz mehr innerhalb der einzelnen Grafikkartenspeicher existiert, verfahren werden muss. Noch weiter verschärfen lässt sich diese Situation, wenn beispielsweise alle Nutzer eines transaktionalen Szenengraphens genau dieselbe (umfangreiche) Szene anzeigen wollen, welche jeweils den kompletten Grafikkartenspeicher für sich beansprucht bzw. benötigt. Hier sind dann alle restlichen, momentan nicht angezeigten Szenengraphenobjekte von dieser Problematik betroffen.

Eine strikte Trennung durch eine Speicherverwaltung, Grafikobjekte nur innerhalb der existierenden Grafikkartenspeicher und normale Objekte nur in den Hauptspeichern abzulegen, kann in diesen Fällen nicht mehr vorgenommen werden. Andernfalls würde es in den genannten Beispielen zu einem Verlust derjenigen Grafikdaten kommen, welche momentan nicht sichtbar sind bzw. für welche gerade kein Platz mehr in den Grafikkartenspeichern zur Verfügung steht. Es müssen folglich bestimmte Ausnahmeregelungen greifen, welche möglicherweise ein Ablegen solcher Grafikobjekte in anderen, nicht benutzten Grafikkartenspeichern – falls vorhanden – erfordern oder doch eine Zwischenspeicherung im regulären Hauptspeicher erlauben würde. Dies widerspricht dann allerdings der Zielsetzung des ursprünglichen Ansatzes, zusätzliche Transferoperationen zwischen dem Hauptspeicher und dem Grafikkartenspeicher durch eine direkte Verteilung dieser Grafikobjekte innerhalb der Grafikkartenspeicher zu vermeiden.

Eine weiterer Aspekt, welcher in einer Speicherverwaltung für einen verteilten Grafikkartenspeicher berücksichtigt werden muss, wird durch die Vorgehensweise vieler verteilter Betriebssysteme bestimmt, bei verteilten Objekten den Replikaten einen einzelnen Eigentümer zuzuweisen. Bei simultanem Zugriff von unterschiedlichen (insbesondere entfernten) Prozessen auf diese Objekte, werden dann im lesenden Fall vom Betriebssystem reine Arbeitskopien erstellt und an die entfernten Stationen verteilt, mit denen diese dann jeweils lokal weiterarbeiten können. Für einen schreibenden Zugriff muss dann aber immer zuerst der Eigentümer um die Erlaubnis für die Änderung gefragt oder eine Übernahme der Eigentümerschaft des betroffenen Objektes vorgenommen werden, um eventuelle Konsistenzprobleme bei mehrfachen Schreibzugriffen zu vermeiden. Der Eigentümer benachrichtigt vor oder nach dem Schreibzugriff dann die anderen Stationen, welche ihrerseits ihre nun veralteten Arbeitskopien verwerfen und bei weiterem Bedarf neu anfordern. Für normale Objekte ist dies eine übliche Verteilungsstrategie (so beispielsweise auch bei Plurix; siehe [1] und [2]).

Für Grafikobjekte, welche direkt innerhalb der Grafikkartenspeicher angesiedelt werden, ist diese Vorgehensweise jedoch nur bedingt einsetzbar. Einerseits verursacht dies einen beständigen Kommunikationsaufwand durch die Befragung des Eigentümers und die (Neu-)Anforderungen der Daten bei Schreibzugriffen. Der hierfür benötigte Zeitaufwand belastet ebenfalls die CPU, welche sich um die Abarbeitung dieser Anfragen kümmern muss, und bremst dadurch indirekt andere Prozesse aus. Andererseits sind aber auch die von der GPU vorgenommenen Schreibzugriffe auf Grafikobjekte innerhalb des Grafikkartenspeichers nicht direkt ersichtlich oder erkenn- und abfangbar, was gerade für die Benachrichtigung der anderen Stationen zur Invalidierung ihrer nun veralteten Grafikobjekte notwendig wäre. Um allein diese Schreibzugriffe feststellen zu können, wäre eine ständige Überwachung der Grafikobjekte[41] oder ein expliziter Mechanismus der GPU erforderlich (beispielsweise das Führen von Änderungslisten o.ä.), da Grafikkarten im Gegensatz zu

41 Dies müsste wiederum von der CPU vorgenommen werden, welche hierfür eigene, zusätzliche Kopien der Grafikobjekte benötigen würde, um die Grafikobjekte im Grafikkartenspeicher damit vergleichen zu können. Eine zugehörige Speicherverwaltung müsste diese weiteren Kopien ebenfalls berücksichtigen, um eventuelle Invalidierungen oder Änderungen auch mit diesen abgleichen zu können.

normalem Hauptspeicher keine Schutzmechanismen innerhalb ihres Speichers für die dort befindlichen Daten besitzen.

Falls nun wie in einem der oben genannten Beispiele mehrere oder gar alle teilnehmenden Stationen auf dasselbe Szenario zugreifen und dieses darstellen (oder gar modifizieren) wollen, müssen die dabei verwendeten Grafikobjekte auf allen zugehörigen Grafikkarten präsent sein, da die Grafikprozessoren jeweils alle zur Darstellung benötigten Daten lokal benötigen und erwarten. Bereits der Schreibzugriff einer einzelnen GPU würde aber die Grafikdarstellung auf sämtlichen anderen teilnehmenden Stationen negativ beeinflussen. Allein durch die hierdurch entstehende Kommunikationslast wären flüssige Bildwiederholraten in Echtzeit mit Standardhardware bei bereits wenigen Stationen nicht mehr zu bewerkstelligen. Innerhalb eines virtuellen Präsenzsystems kommt diese Situation jedoch relativ häufig vor, da sich in der Regel immer mehrere Avatare am gleichen Ort aufhalten und auch miteinander in (Sicht-)Kontakt stehen.

Für Grafikobjekte würde sich aus den genannten Gründen daher eine Umsetzung der Verteilung mittels autonomer Replikate wesentlich besser eignen. Hierüber ließe sich der nötige Kommunikationsaufwand reduzieren oder zumindest zeitlich entkoppeln. Dies würde allerdings auch eine erneute Unterscheidung der Objekte durch die Speicherverwaltung zu der Art ihrer Verteilung erfordern, um nun einerseits die Grafikobjekte mittels Replikaten zu verteilen und andererseits die restlichen Objekte wie zuvor mit Eigentümern zu verwalten.

4.2.2.4 Leistungsfähigkeit

Die erreichbare Leistung eines verteilten Grafikkartenspeichers ist nicht zuletzt auch ein wesentlicher Aspekt, welcher bei einer Realisierung der hier aufgezeigten Speicherverwaltung berücksichtigt werden muss. Eine Vielzahl der zuvor angesprochenen Punkte lässt sich bislang aber mit bestehender Standardhardware nicht direkt unterstützen bzw. ist auch für einen solchen Einsatzzweck nicht ausgelegt und muss daher in Software umgesetzt werden.

Insbesondere die Hardwarebeschleunigung einzelner Teilaspekte durch die System-MMU kann in der bestehenden Form nicht mehr durchgängig eingesetzt werden. Die MMU reagiert beispielsweise nur auf Seitenfehler der CPU, nicht aber auf etwaige Anforderungen einer GPU. Ebenso kennt sie von sich aus keinen verteilten Speicher und kann auch keine eigenständige Unterscheidung bei der virtuellen Adressauflösung zwischen normalen und Grafikobjekten vornehmen.

Des Weiteren ist zu beachten, dass heutige Grafikkarten nicht autonom zu der CPU arbeiten können. Für jedes einzelne zu rendernde Bild müssen sie einen expliziten Befehl von „außen“, d.h. von der CPU, bekommen, welcher diesen Rendervorgang für das nächste Bild startet und Kommandos für die Abarbeitung der dabei vorkommenden Shader-Programme absetzt. Eine Art von „kontinuierlichem Modus“, in dem die Grafikkarte periodisch und selbständig aus den vorhandenen Daten ein neues Bild generiert, ohne explizite Befehle oder Kommandopakete von der CPU zu benötigen, existiert bei Grafikkarten bislang nicht. Dies erzwingt bereits deshalb schon eine beständige Kommunikation bzw. Befehlsübermittlung zwischen der CPU und der Grafikkarte.

Ein weiterer Punkt stellt die Leistungsfähigkeit eines solchen Systems bezüglich der vorhandenen Übertragungswege für die Grafikobjekte und -daten dar. Die System-MMU arbeitet direkt mit dem Hauptspeicher zusammen, mit welchem sie über sehr schnelle und speziell dafür ausgelegte Anbindungen direkt über die Northbridge kommuniziert kann. Alle Zugriffe auf den Grafikkartenspeicher müssen in einem Standardsystem dahingegen als deutlich langsamere Buszugriffe (üblicherweise über PCI, AGP oder PCI-Express) umgesetzt werden. Zwar sind auch diese verwendeten Bussysteme einhergehend mit der Weiterentwicklung der Grafikkarten immer schneller geworden – der heutzutage aktuelle PCI-Express-Bus erreicht Datentransferraten von ca.

4 GiB/s (PCIe x16) –, sie sind damit aber immer noch ca. 2,5-fach langsamer, als Transfers zum Hauptspeicher, welche mit bis zu 10 GiB/s stattfinden können.

4.2.3 Bestehende Arbeiten

Die Einbeziehung von Grafikkartenspeicher innerhalb verteilter Speicherkonzepte ist ein insgesamt nur wenig erforschtes Gebiet, obwohl es schon seit einigen Jahren erste Ansätze hierzu gibt.

Bereits 1999 wurde von 3Dlabs [w51] ein Entwurf vorgestellt [57], in welchem Texturen für eine Grafikkarte mittels einer virtuellen Adressierung nutzbar gemacht wurden und als „Virtuelle Texturen" bezeichnet wurde. Dieser Ansatz resultierte im Jahr 2000 sogar im Beginn der Oxygen-Grafikkartenreihe von 3Dlabs, welche für damalige Workstations entwickelt und vertrieben wurde. Der Einsatzzweck dieser Grafikkarten war hauptsächlich für CAD-Anwendungen im Profi-Bereich ausgelegt; insgesamt konnten sich die Grafikkarten aber nicht dauerhaft durchsetzen und verschwanden somit auch bald wieder vom Markt[42].

In diesen Grafikkarten (und somit in diesem Ansatz) war eine grafikkarteneigene MMU für die virtuellen Texturen integriert. Mit ihr konnte auf bis zu 256 MiB an virtuellen Texturen, welche sich im normalen Hauptspeicher befanden, zugegriffen werden (bei damals durchschnittlich nur 32 MiB an tatsächlichem Grafikkartenspeicher). Die 'Virtual Texture MMU' arbeitete dabei sehr ähnlich zu der normalen virtuellen Speicherverwaltung einer CPU. Die GPU (damals 'Graphics Engine') verwendete hierbei zur Adressierung ihrer Grafikdaten eigene (d.h. grafikkartenspezifische), virtuelle Adressen, welche von der Virtual Texture MMU transparent auf die physikalischen Grafikkartenspeicherstellen umgesetzt wurden. Gab es dabei von der GPU einen Zugriff auf eine nicht lokal vorhandene Textur, so wurde dies von der Virtual Texture MMU erkannt und ein Seitenfehlermechanismus (innerhalb der Grafikkarte) ausgelöst, welcher anschließend per DMA, und somit ohne Eingreifen der normalen CPU, die fehlenden Texturdaten aus einem für die virtuellen Texturen reservierten Bereich des Hauptspeichers in den lokalen Grafikkartenspeicher einlagerte und dabei auch gegebenenfalls ältere Texturen bei Platzmangel zuvor entfernte. Die Virtual Texture MMU der Grafikkarte arbeitete dabei – analog zu ihrem MMU-Pendant der CPU – auf einzelnen 4 KiB-Speicherseiten, war allerdings – im Gegensatz zu ihrem Pendant – auch fest auf diese Größe beschränkt.

Ein Novum für damalige Grafikkarten – welche mit der Einführung der virtuellen Adressierung einherging – war, dass Texturen nun nicht mehr als Ganzes angefasst und somit auch nicht mehr immer vollständig in die Grafikkarte übertragen werden mussten, wie es dahingegen selbst heute noch üblich ist, auch wenn nur kleine Bereiche einer Textur benötigt werden. Es genügte, wenn nur noch die einzelnen, angefassten Teile bzw. Seiten einer Textur eingelagert wurden. Auch konnten die eingelagerten Texturdaten nun ebenfalls beliebig fragmentiert abgelegt werden (zumindest auf Seitengröße) und mussten nicht mehr komplett linear innerhalb des lokalen Grafikkartenspeichers vorliegen.

Insgesamt kamen diese (damals) innovativen Neuerungen aber einzig aus der Einführung einer Grafikkartenspeicherverwaltung; die GPU selbst arbeitete dabei nicht wesentlich anders als zuvor. D.h. sie kannte weiterhin nur kontinuierliche, aber anstelle physikalischer eben virtuelle Adressen (was für die GPU selbst keinen Unterschied bedeutete; lediglich von der Virtual Texture MMU wurden die Adressen nun anders interpretiert) und griff aus ihrer Sicht nach wie vor immer nur auf vollständig geladene, lineare Texturen zu.

Mit den Oxygen-Grafikkarten erschienen so die ersten Grafikkarten, welche eine virtuelle

42 Mittlerweile (seit Anfang 2006) hat sich 3Dlabs sogar gänzlich aus dem Markt für Grafikkarten zurückgezogen und sich nur noch auf mobile und eingebettete Prozessoren spezialisiert [w52].

Speicherverwaltung besaßen, wie es für eine Nutzung bei einer weitergehenden Verteilung des Grafikkartenspeichers erforderlich bzw. zumindest wünschenswert ist. Allerdings war ein Verteilungsaspekt bei diesen Grafikkarten nicht vorgesehen und infolgedessen auch nicht weiter verwirklicht oder angedacht.

Seit dieser Arbeit hat sich dieses Gebiet aber kaum weiterentwickelt. Erst in jüngster Zeit, einhergehend mit der voranschreitenden Leistungsfähigkeit und universelleren Einsetzbarkeit von heutigen Grafikkarten, aber auch mit dem beständigen Zuwachs an vorhandenem lokalem Grafikkartenspeicher, rückt das Thema eines verteilten Grafikkartenspeichers wieder stärker in den Bereich gegenwärtiger Forschung[43].

Eine aktuelle Arbeit der University of California von 2008 [58] beschreibt beispielsweise eine mögliche Realisierung, wie unter Verwendung moderner, Shader-basierter Grafikkarten eine Verteilung von Texturdaten – und somit Teile des Grafikkartenspeichers – über mehrere Grafikkarten hinweg erreicht werden kann.

Bei dem dort vorgeschlagenen Ansatz wird mittels einer Zwischenschicht eine separate Speicherverwaltung für den Grafikkartenspeicher eingefügt und über diese dann weitergehend eine Verteilung von Texturdaten, ähnlich einem DSM-basierten Ansatz, vorgenommen. Die entwickelte Zwischenschicht setzt hierzu auf bestehende Grafikkartentreiber der jeweiligen Hersteller auf. Hiermit wird die Schwierigkeit umgangen, selbst für die unterstützten Grafikkarten Treiber entwickeln und zur Verfügung stellen zu müssen, und es lassen sich dadurch prinzipiell alle Shader-basierten Grafikkarten von beliebigen Herstellern verwenden. Andererseits ist es bei diesem Entwurf dadurch aber nicht mehr oder nur schwer möglich, selbst Einfluss auf die Arbeitsweise der verwendeten Grafikkartenhardware zu nehmen, wodurch die von der Hardware potentiell mögliche Leistungsfähigkeit durch diese Zwischenschicht bei weitem nicht erreicht werden kann und dies in der beschriebenen Arbeit auch sehr deutlich zum Ausdruck kommt. Dennoch handelt es sich um ein innovatives Konzept, welches viele der bislang beschriebenen Punkte aufgreift und zu lösen versucht, und welches deswegen hier nachfolgend genauer vorgestellt wird.

Die erwähnte Zwischenschicht bietet nach oben, d.h. zur Anwendungsseite hin, eine spezielle Schnittstelle an, über welche die Verteilung der Grafikdaten auf die verschiedenen GPUs transparent verborgen wird. Innerhalb dieser Zwischenschicht wird nun eine eigenständige Speicherverwaltung für die Grafikkarte bzw. deren Grafikkartenspeicher eingeführt, um die Verteilung des Speichers vornehmen zu können. Zu diesem Zweck wird von der Speicherverwaltung ein großer, einheitlicher Adressraum aufgebaut, welcher ebenfalls nach unten für die Grafikkarten nutzbar gemacht wird und der ähnlich dem Konzept eines DSMs für Grafikkartenspeicher arbeitet. Es wird dadurch erreicht, dass alle Grafikdaten einheitliche (virtuelle) Adressen erhalten, über die alle GPUs gleichartig auf diese zugreifen können.

Der von der Speicherverwaltung und den GPUs gemeinsam angesprochene (physikalische) Grafikkartenspeicher wird dazu in einzelne Bereiche, den sogenannten Texturseiten, aufgeteilt, welche dann auch die kleinste Einheit zum weiteren Datentransfer darstellen. Die Größe einer solchen Texturseite – und somit die Granularität der Speicherverwaltung – ist im Vorfeld konfigurierbar, entspricht dabei aber immer rechteckigen Texturblöcken in Zweierpotenz-Schritten (z.B. 512 * 512 Texel).

43 Insbesondere von den großen Grafikkartenherstellern werden Teilaspekte hiervon weiter verfolgt, so beispielsweise nVidia mit ihrer SLI-Technik oder AMD/ATI mit ihrer korrespondierenden CrossFire-Technik, welche beide eine – wenn auch eingeschränkte – Art einer Verteilung bzw. gemeinsamen Nutzung ihres Grafikkarenspeichers innerhalb eines Rechnersystems mit mehreren Grafikkarten ermöglicht.

Für die Verwaltung dieser Texturseiten wird von der Speicherverwaltung ein verzeichnisbasiertes Verfahren eingesetzt, welches ähnlich zu einer traditionellen Speicherverwaltung mit Seitentabellen arbeitet. Das eingesetzte Verzeichnis enthält hierfür alle relevanten (Zusatz-)Informationen zu jeder Texturseite, d.h. Einträge, von welcher GPU bzw. welchen GPUs eine Texturseite gerade verwendet wird und weiteren Statusbits wie für das Vorhandensein einer Texturseite oder zur Erkennung von durchgeführten Schreibzugriffen. Zusätzlich existiert auch immer eine Referenz, wo sich die zugehörigen Grafikdaten (ursprünglich) im Hauptspeicher befinden, denn erst bei Bedarf von einer GPU wird von der Speicherverwaltung eine Kopie der betroffenen Texturseite(n) aus dem Hauptspeicher in den Grafikkartenspeicher der jeweiligen Grafikkarte transferiert.

Die Arbeit der Speicherverwaltung, wie auch die einhergehende, beständige Aktualisierung des Verzeichnisses, werden von einer CPU übernommen. Damit das Verzeichnis allerdings nicht zu einer zentralen Komponente wird und einer eventuellen, späteren Skalierung der Speicherverwaltung entgegensteht, wurde eine spezielle Variante eines verzeichnisbasierten Verfahrens ausgewählt, welches sich prinzipiell auch verteilt betreiben lässt (aber in der dort vorgestellten Arbeit bislang noch nicht verteilt verwendet wird).

Damit nun die GPU innerhalb dieses Konzeptes auch selbst Anforderungen nach bestimmten Daten dem restlichen Rechnersystem bzw. der verwaltenden CPU zukommen lassen kann – was von der aktuellen Grafikkartenhardware selbst bislang nicht unterstützt wird bzw. gar nicht möglich ist –, wird ein spezieller Mechanismus eingeführt, um dies – zumindest indirekt – doch zu ermöglichen.

(Nahezu) Alle Grafikoperationen einer GPU werden bei Shader-basierten Grafikkarten heutzutage mittels darin ablaufender Shader-Programme ausgeführt. Hierzu müssen im Vorfeld für alle von einer Grafikkarte durchzuführenden Berechnungen von einem Programmierer zugehörige Shader-Programme erstellt und anschließend für die jeweilige Shader-Hardware kompiliert werden. Erst danach sind sie für den Einsatz innerhalb einer GPU bzw. eines Shaders nutzbar und können dann in die Grafikkarte geladen und dort ausgeführt werden. In dem vorgeschlagenen Ansatz wird nun dieser Kompilierungsvorgang derart modifiziert, dass von jedem Shader-Programm zwei separate, kompilierte Versionen erzeugt werden, die dann nacheinander von der GPU abgearbeitet werden[44]. Die erste Version fängt dabei alle Lese- und Schreibvorgänge auf Texturdaten ab und berechnet anstelle der eigentlich darauf auszuführenden Grafikoperationen die Adressen der Texturseiten, auf welche an dieser Stelle jeweils zugegriffen werden soll. Dabei werden insbesondere die Schreibzugriffe als exklusive Seitenzugriffe für die betroffenen Daten markiert. Das Ergebnis dieser Adressberechnungen wird anschließend in einen speziellen Bereich innerhalb des Grafikkartenspeichers „gerendert“, d.h. abgelegt, welcher als Anforderungspuffer bezeichnet wird[45]. In diesem Anforderungspuffer kann nun die Speicherverwaltung der Zwischenschicht bzw. die zuständige CPU aktiv nachschauen (mittels Polling), ob hier neue Einträge vorhanden sind und somit von der GPU neue Texturseiten benötigt werden und diese daraufhin anhand der dort befindlichen Adressen, sofern sie nicht schon vorhanden sind, einlagern bzw. umkopieren. Sobald alle benötigten Grafikdaten dann auf der Grafikkarte verfügbar sind, kann die GPU die zweite Version des Shader-Programmes ausführen, welches nun die eigentlichen Grafikoperationen wie gewünscht mit den jetzt vorhandenen Daten vornehmen kann.

Bei simultanem Zugriff zweier GPUs auf dieselben Texturdaten können dabei allerdings

44 Dieser Eingriff in den Kompilierungsvorgang wird momentan noch manuell durchgeführt, ist aber von den Autoren dieser Arbeit in späteren Versionen als dynamisch vorgenommene Modifikation geplant, welche dann automatisch von einem Treiber zur Laufzeit durchgeführt werden soll.

45 Genau genommen ist es für die Grafikkarte kein spezieller Bereich, sondern ein gewöhnliches Renderziel im Grafikkartenspeicher. Lediglich die CPU muss wissen, dass alle Daten, die innerhalb dieses bestimmten Bereichs im Grafikkartenspeicher abgelegt sind, als Adressen für benötigte Texturseiten der Grafikkarte zu interpretieren sind und sieht daher diesen Bereich als speziellen Anforderungspuffer.

konkurrierende Zugriffe ('Race Conditions') auftreten, wobei sich zwar die Speicherverwaltung um eine Sequentialisierung dieser Zugriffe kümmert (mittels Sperren), dabei aber keine Reihenfolge der Schreibzugriffe regelt. Für solche Fälle muss der Programmierer selbst wissen, wie seine Berechnungen und die dabei auftretenden Zugriffe verteilt werden und wie diese konkurrierenden Zugriffe von ihm durch die Verwendung geeigneter Zugriffsmuster vermieden oder mittels eingefügter, expliziter Synchronisierungen aufgelöst werden.

Bislang wurde dieser Ansatz in der aufgeführten Arbeit erst für einen einzelnen PC mit mehreren Grafikkarten realisiert. Eine Ausdehnung auf mehrere Rechner mit mehreren Grafikkarten ist allerdings konzeptuell möglich und von den Autoren in künftigen Arbeiten angedacht. Allerdings sind bereits in der bestehenden Umsetzung deutliche Leistungseinbußen durch das gänzlich in Software ausgeführte Verfahren und die Verzögerungen durch die eingesetzten Mechanismen (Polling bzw. Locking) erkennbar. Die Autoren schlagen deshalb auch mögliche Änderungen am Hardwaredesign von Grafikkarten vor, mit welchen sich ihre vorgeschlagenen Konzepte effizienter umsetzen ließen.

Eine weitere Arbeit von der irischen 'Trinity College'-Universität [59] untersucht einen ähnlichen Ansatz zu der gerade beschriebenen Arbeit, in welcher der lokale Speicher von Grafikkarten mittels eines DSMs verknüpft wird und sich hierüber eine Datenverteilung durchführen lässt. Es wird dabei allerdings ein anderer Weg beschritten, indem ein Hardware-DSM vorgeschlagen wird, welcher über eine spezielle, selbstentwickelte Hardware auch prototypisch bereits realisiert wurde. Es wird hierbei mittels programmierbarer FPGAs ('Field Programmable Gate Arrays') ein Interface bzw. Adapter geschaffen, welcher einerseits mit dem Rechner über den Grafikkartenbus verbunden wird und andererseits selbst eine Grafikkarte aufnehmen kann. Durch separate Verbindung zu anderen FPGA-Adaptern können diese direkt miteinander kommunizieren und Daten austauschen. Mit diesem entwickelten FPGA-Board wird dieser Hardware-DSM ebenfalls als Zwischenschicht in das Gesamtsystem eingeführt und soll für Anwendungen transparent sein. Die Grafikkarten werden dabei zwar hardwaremäßig nicht verändert, aber für die Kommunikation mit dem FPGA-Board müssen jeweils Anpassungen in der Ansteuerung der Grafikkarten vorgenommen werden, was wiederum die Entwicklung eigener Grafikkartentreiber vonnöten macht. Der vorgeschlagene Ansatz entspricht somit einer Hybrid-Lösung aus Hardware- und Softwaremodifikationen.

Jüngste Pläne von Microsoft gehen in diesem Bereich sogar noch einen Schritt weiter [60]. Microsoft kündigt hier neue Vorgaben und Standards für ein zukünftiges Grafiktreibermodell unter Windows Vista namens 'Window Display Driver Model' (WDDM) an. Verschiedene Versionen von WDDM definieren dabei die von einer hierfür eingesetzten Grafikkarte die in Hardware geforderten Fähigkeiten und Merkmale. So ist WDDM 1.0 für die aktuell existierenden, Shader-basierten Grafikkarten gedacht, deren Anforderungen sich mit den in dieser Dissertation bisher beschriebenen Fähigkeiten moderner Grafikkarten decken. Speziell wird hierbei die bislang verwendete physikalische Adressierung des Grafikkartenspeichers von der GPU und eine sequentielle bzw. kooperative Abarbeitung der Shader-Programme innerhalb der GPU beschrieben.

Mit der Einführung von WDDM 2.0, wird von den Grafikkarten ein deutlicher Zuwachs an Funktionalität bezüglich ihrer Arbeitsweisen verlangt. Die dieser Spezifikation entsprechenden Grafikkarten müssen dann bereits eigenständig mit virtuellen Adressen und somit mit Seitentabellen umgehen können. Ein zugehöriger Seitenfehlermechanismus muss ebenfalls unterstützt werden, mit dem sich Daten des Grafikkartenspeichers in das RAM auslagern bzw. hierüber wieder anfordern und einlagern lassen. Die Spezifikation erlaubt einer GPU dabei allerdings, dass sie bis zum

Eintreffen der angeforderten Daten warten bzw. blockieren kann. Für die Abarbeitung der Shader-Programme selbst wird ein präemptives Multitasking innerhalb der GPU vorgeschrieben. Diese Anforderungen werden derzeit noch nicht von den Grafikkartenherstellern unterstützt, da sie eine wesentliche Überarbeitung bzw. Erweiterung der bisherigen Grafikkartenhardware erfordern[46].

In der nachfolgenden Version WDDM 2.1 werden die Anforderungen für die Grafikkartenhersteller noch weiter verschärft; insbesondere das präemptive Multitasking aus Version 2.0 wird hierbei als echtes präemptives Multitasking vorgeschrieben, d.h. ein Blockieren der GPU aufgrund von Wartezeiten ist nicht länger zulässig.

Eine von Microsoft bislang erst vage angekündigte Version WDDM 3.0 soll dann zusätzlich noch Virtualisierung in den Grafikkarten ermöglichen. Zu welchem Zeitpunkt und in welcher Spezifikation WDDM 3.0 vorgeschrieben und enthalten sein soll, steht bislang allerdings noch nicht fest.

4.3 Transaktionierung von Grafikkartenspeicher

Der zweite Schritt zur direkten Nutzung von Grafikkartenhardware bzw. dessen Speicher in einem transaktionalen Szenengraphen beinhaltet die Einbeziehung der transaktionalen Konsistenz innerhalb der verwendeten Hardware. Hierdurch lassen sich die Eigenschaften eines transaktionalen Szenengraphens direkt und effizient übertragen und eine Entlastung der CPU erreichen, welche ansonsten die für die transaktionale Konsistenz benötigten Aspekte umzusetzen hat.

4.3.1 Transaktionaler Grafikkartenspeicher

Um eine transaktionale Konsistenz bei der Arbeitsweise von Grafikkartenspeichern ermöglichen und gewährleisten zu können, muss in die bisherige Arbeitsweise von Grafikkarten eingegriffen werden. Da diese hauptsächlich dafür ausgelegt sind, Grafikdaten lediglich zu empfangen, um anschließend unbeeinflusst von anderen Systemkomponenten oder Geräten diese verarbeiten und ausgeben zu können, wurde von den Grafikkartenherstellern bisher eine separate Betrachtung zur Konsistenz dieser Grafikdaten nicht vorgenommen und daher auch im Hardwareaufbau und im internen Ablauf nicht weiter berücksichtigt. Um trotzdem eine Einbeziehung der transaktionalen Konsistenz in die Arbeitsweise mit einem Grafikkartenspeicher zu ermöglichen, müssen somit entweder von der Grafikkarte selbst oder von einer hier vorgeschlagenen Speicherverwaltung zusätzliche Eigenschaften berücksichtigt und eingebracht werden.

4.3.1.1 Transaktionsbedingungen

Die Sicht heutiger Grafikprozessoren auf den ihnen zugeordneten Grafikkartenspeicher entspricht im Wesentlichen derjenigen einer reinen Datenquelle, von welcher alle während des Rendervorgangs benötigte Daten geliefert werden, und demjenigen eines Zwischenspeichers, welcher als Übergabeort der Renderresultate für den nebenläufig arbeitenden Anzeigekontroller dient. Die Möglichkeit des Auslesens von Daten aus dem Grafikkartenspeicher von anderen Komponenten als der GPU selbst und des Anzeigekontrollers ist prinzipiell zwar gegeben, wird bislang aber nur in Ausnahmefällen von einer Anwendung bei expliziter Programmierung vorgenommen und ist daher auch nicht weiter im Hardwaredesign von Grafikkarten berücksichtigt. Besonders die ersten Grafikkartengenerationen waren auf den reinen Empfang von Daten durch die

46 Somit könnte dadurch zumindest einem Teil der Konzepte, welche bereits vor knapp 10 Jahren in den Oxygen-Grafikkarten von 3Dlabs enthalten waren, zu neuer Renaissance verholfen werden.

CPU, d.h. auf von ihr vorgenommene Schreibvorgänge zum Einlagern von Daten in den Grafikkartenspeicher optimiert; ein erneutes Auslesen war demgegenüber wesentlich langsamer. Mittlerweile besteht dieser Unterschied zwar nicht mehr, allerdings ist die Arbeitsweise einer GPU mit „ihrem“ Speicher nach wie vor auf eine aus ihrer Sicht alleinige Nutzung ausgelegt, welches dann von ihr auch mit einem jederzeit exklusiven Zugriff darauf gleichgesetzt wird. Eine Beeinflussung durch andere Komponenten tritt in der Sichtweise der GPU nicht weiter auf und es wird daher auch keine weitergehende Konsistenzbetrachtung zu den vorliegenden Grafikdaten vorgenommen.

Aus der Sicht der GPU greift keine andere Instanz außer sie selbst modifizierend auf die Daten im Grafikkartenspeicher zu[47], weshalb von ihr sogar eine strikte Konsistenz der vorliegenden Daten impliziert wird (eventuell vorhandene Grafikpipelines und Shader-Einheiten arbeiten in der Regel auf eigenen, getrennten Bildabschnitten und somit verschiedenen Speicherbereichen und beeinflussen sich daher gegenseitig ebenfalls nicht). Erst wenn ein beliebiger Zugriff von außerhalb der Grafikkarte jederzeit mit zu berücksichtigen ist oder eine eingesetzte Speicherverwaltung Grafikobjekte nicht nur im Grafikkartenspeicher ablegt, sondern auch wieder daraus abruft, kann die strikte Konsistenz nicht länger impliziert werden.

Um nun eine andere Konsistenzform auf den Grafikdaten zu unterstützen bzw. überhaupt erst zu ermöglichen – im Speziellen die transaktionale Konsistenz –, wäre ein Schutz der Grafikdaten nötig, welcher einen Zugriff abfangen und entsprechend darauf reagieren könnte. Dies würde eine eigenständige Grafikkartenspeicherverwaltung oder eine spezielle MMU erfordern, wie sie in Kapitel 4.2.2 bereits vorgeschlagen wurde. Alternativ ließe sich dies auch durch eine transaktionale Arbeitsweise bei den darauf ausgeführten Berechnungen erreichen. D.h., dass die vorgenommenen Arbeitsschritte den in Kapitel 1.4 beschriebenen ACID-Bedingungen für Transaktionen genügen müssten[48]. Hierfür wären Eingriffe in die Verhaltensweise der GPU bei einem Speicherzugriff nötig.

Die Grafikkartenhardware ist von den Grafikkartenherstellern allerdings nicht auf eine transaktionale Arbeitsweise ausgelegt. Besonders ältere, „fest verdrahtete” ('Fixed Function') Grafikkarten sind daher für einen solchen Einsatz nur sehr bedingt geeignet, da die Transaktionsbedingungen nicht von den Grafikkarten selbst erfüllt werden können und somit eine ständige Kontrolle von außen, d.h. durch die CPU, notwendig wäre, was einer vollwertigen Integration dieser Grafikkarten bzw. deren -speicher in ein transaktionales Umfeld deutlich erschwert oder gar verhindert.

Aktuellere Grafikkarten sind durch ihren flexibleren Hardwareaufbau bereits wesentlich besser hierfür geeignet. Eine direkte Anpassung ihrer internen Arbeitsweisen auf ein transaktionales Vorgehen durch beispielsweise eine Modifikation ihrer Firmware oder des benötigten Mikrocodes ist allerdings durch die nicht veröffentliche Dokumentation hierzu auch noch kein realistisches Vorgehen (siehe Kapitel 4.1.2).

47 Die Übertragung und Einlagerung der Grafikdaten durch die CPU wird von der GPU hierbei nicht weiter beachtet. Die GPU wird über diese (neuen) Daten erst zu Beginn des nächsten Renderdurchlaufs informiert, so dass hier prinzipiell nur sequentielle Zugriffe auftreten, die sich gegenseitig nicht beeinflussen und die Konsistenz gefährden können. Ebenso ist das Auslesen des fertigen Bildes durch den Anzeigekontroller unproblematisch. Hierbei treten keine konsistenzgefährdenden Schreibzugriffe auf und der Zugriff erfolgt komplett nebenläufig und unbemerkt von der GPU.
Sollte von der CPU doch während eines Rendervorgangs modifizierend auf die Grafikdaten zugegriffen werden, so liegt es in ihrer eigenen Verantwortung bzw. bei der zugehörigen Anwendung, die Konsistenz der Grafikdaten hierbei zu wahren; die GPU nimmt während ihrer Berechnungen keine weiteren Überprüfungen hierzu vor.

48 Einige Betrachtungen zu Transaktionen und transaktional genutztem Speicher gehen dabei sogar soweit zu sagen, dass Haupt- bzw. Gerätespeicher in der Regel flüchtige Speicherarten darstellen, welche nach einem Reset oder Fehlverhalten sowieso explizit wieder zu befüllen sind, weshalb diese dann nur noch von den ACI-Bedingungen sprechen und die Dauerhaftikgkeit in ihren Überlegungen weggelassen (siehe [61]).

Ein möglicher Ansatz wäre daher die Realisierung der benötigten Transaktionsbedingungen mittels der von der GPU abgearbeiteten Shader-Programme. Ähnlich zu dem bestehenden Programmiermodell für Transaktionen in Software wäre eine Kapselung der Shader-Programme innerhalb eines hierbei umschließenden 'Begin of Transaction' (BOT) und 'End of Transaction' (EOT) denkbar. Dabei könnten alle Berechnungen auf speziellen, „lokalen" Bereichen der Grafikkarte ausgeführt werden (gleichsam der Nachbildung eines privaten Stacks bzw. von „Schattenkopien") und erst beim EOT sichtbar gemacht werden.

Die Befehlsverarbeitung und Ablaufsteuerung von aktuellen Grafikkarten enthalten bereits Scheduler-ähnliche Vorgehensweisen, um eine Aufteilung der anstehenden Berechnungen und Shader-Programme auf die einzelnen Grafikpipelines und Verarbeitungseinheiten vorzunehmen. Eine Erweiterung dieser Scheduler-ähnlichen Vorgehensweisen könnte beispielsweise die Berücksichtigung des Transaktionskonzeptes anhand einer von ihr vorgenommenen Erkennung der BOT- und EOT-Instruktionen beinhalten.

4.3.1.2 Rücksetzbarkeit

Die transaktionale Konsistenz beinhaltet neben den Transaktionsbedingungen auch die Rücksetzbarkeit von abgebrochenen Transaktionen im Konfliktfall. Dieses Verhalten müsste dann ebenfalls von der Grafikkarte berücksichtigt werden, indem die GPU abgebrochene Shader-Programme zur erneuten Berechnung anstößt.

Um eine Rücksetzung der eventuell bereits ausgeführten Modifikationen zu erreichen, müssen diese vorläufigen Shader-Berechnungen auf lokalen Kopien angefertigt werden (beispielsweise dem erwähnten privaten Stack). So könnte ein BOT die GPU dazu veranlassen, alle folgenden Instruktionen eines Shader-Programms bis zum EOT zuerst auf den erstellten Kopien in einem separaten Speicherbereich auszuführen und erst in der Commit-Phase nach dem EOT die Resultate in den eigentlichen Bildschirmspeicher oder die jeweilig angedachte Zielregion (Texturen, Grafikobjekte, ...) zu transferieren. Im Falle eines Abbruchs kann dahingegen einfach die Kopie verworfen werden.

Dies entspricht dabei einer pessimistischen Vorgehensweise, welche von häufigen Konflikten und Abbrüchen ausgeht. Anzunehmen ist bei der hohen Verarbeitungsgeschwindigkeit von Grafikkarten allerdings viel mehr, dass ein solches Verhalten relativ selten auftritt und daher eine optimistische Variante besser geeignet wäre (ähnlich zu dem im Plurix-System verwendeten Verfahren, siehe Kapitel 1.5), bei der die einzelnen Shader-Programme direkt auf den Originaldaten bzw. -zielorten arbeiten und nur ein Abbild der ursprünglichen Daten zum Zeitpunkt des BOTs anfertigen. Im Falle eines Abbruchs können die Originaldaten dann anhand dieses Abbildes wieder rekonstruieren werden, im Erfolgsfall sind aber die Resultate bereits an den korrekten Stellen vorhanden und das Abbild kann verworfen bzw. freigegeben werden. Die geeignete Vorgehensweise könnte anhand von Messungen bei einer konkreten Realisierung verifiziert werden.

Falls eine Anpassung der GPU-Steuerung für automatische Rücksetzungen nicht weiter möglich ist, können alternativ hierzu auch die Shader-Programme selbst (zumindest teilweise) diese Aufgaben übernehmen. Ein Compiler für die Shader-Programme könnte alle durch BOT-und EOT-Anweisungen gekapselte Instruktionen derart abwandeln, dass von den Shader-Programmen in dieser Zeit nur auf für sie privaten Speicherbereichen gearbeitet und die Resultate dort abgelegt werden. Ein jeweils nachgeschaltetes, zweites Shader-Programm könnte anschließend die Aufgabe der Validierung vornehmen und die Daten dann entweder freigeben (d.h. in die gemeinsam sichtbaren Bereiche umkopieren) oder gegebenenfalls eine Rücksetzung durchführen, abhängig davon, ob die Berechnungen erfolgreich durchgeführt wurden oder ein Konflikt vorliegt.

4.3.1.3 Invalidierungen

Die zuvor erwähnten Aspekte betreffen die Arbeitsweise der Grafikkarte selbst. Für eine Einbeziehung in einen übergreifenden, transaktionalen Szenengraphen ist allerdings auch eine Benachrichtigung anderer Stationen und deren Speicherverwaltungen bzw. Grafikkarten notwendig, sollten diese ebenfalls Arbeitskopien von Grafikobjekten besitzen und sich diese ändern.

Eine Schwierigkeit, die hierbei auftritt, besteht darin, Schreibzugriffe einer Grafikkarte auf Objekte in ihrem Speicher überhaupt erst zu bemerken. Da der Grafikkartenspeicher bisher nicht direkt schützbar ist, kann infolgedessen auch nicht direkt erkannt werden, wann die GPU einen (Schreib-)Zugriff darauf vornimmt. Somit wäre auch in diesem Fall eine beständige, aktive Kontrolle der Inhalte durch Vergleiche mit älteren Versionen vonnöten, um auftretende Änderungen überhaupt erkennen und darauf reagieren zu können.

Geeigneter wäre daher auch hier eine Verschiebung dieser Aufgabe in die Validierungsphase der zugehörigen Transaktion. Diese würde dann entweder von einem separaten, einer Transaktion nachgeschalteten Shader-Programm oder von der GPU selbst während des EOTs vorgenommen werden. Dabei könnten dann die geänderten Speicherbereiche oder eventuell sogar die zugehörigen (Grafik-)Objekte beim Commit ermittelt werden, welche dann durch eine Speicherverwaltung aufgegriffen werden und gegebenenfalls auch zu einer Benachrichtigung oder Aktualisierung von anderen Stationen führt.

Dieses Verfahren wird allerdings umso anspruchsvoller, je mehr Autonomie einer Grafikkarte zugestanden wird. Insbesondere ist dies erkennbar, wenn eine Grafikkarte selbständig neue Grafikobjekte erzeugen kann (beispielsweise wenn von ihr eine neue Textur erstellt wird), die gemeinsam nutzbar sein sollen. Dem System bzw. der Speicherverwaltung muss diese Objekterstellung explizit mitgeteilt werden (für sie könnte es sich ansonsten auch nur um temporäre, lokale Daten handeln) und es sind zusätzliche Informationen nötig, um aus den neuen generierten Daten ein systemkompatibles Objekt zu machen (d.h. die vom System verwendete Objektstruktur muss um die generierten Daten herum angebracht werden, was z.B. Längenangaben, Zeigerstrukturen u.ä. beinhaltet). Dies müsste dann entweder durch nachträgliches Hinzufügen der Objektinformationen an die Daten innerhalb des Grafikkartenspeichers erfolgen (eingefügt durch die GPU oder die Speicherverwaltung) oder es könnte eine separate Verwaltungsstruktur hierzu vom System administriert werden, welche parallel zu den eigentlichen, erzeugten Daten innerhalb der Grafikkarte geführt wird und eine Referenz auf diese besitzt.

4.3.2 Bestehende Arbeiten

Vergleichbare Arbeiten, welche ein transaktionales Konsistenzmodell auf einen Gerätespeicher und im Speziellen auf Grafikkartenspeicher anwenden bzw. dort einsetzen, existieren in dieser Form bislang nicht.

Es gibt dahingegen eine wachsende Anzahl von Arbeiten welche sich darauf beziehen, aktuelle Grafikkarten bzw. deren Grafikprozessoren zur Berechnung von allgemeinen, d.h. nicht ausschließlich grafik- oder renderbezogenen, aber abgeschlossenen Aufgaben heranzuziehen, wie z.B. die Lösung beliebiger numerischer Probleme, oder zur Simulation von physikalischen Sachverhalten, beispielsweise einem Strömungsverhalten ([62], [63], [64]).

In [65] wird darüber hinaus eine Sammlung von Arbeiten vorgestellt, welche sich mit allgemeinen Berechnungen auf Grafikprozessoren beschäftigten. In diesem Zusammenhang hat sich mittlerweile in der Fachliteratur auch anstelle des normalen Grafikprozessors der Begriff des Mehrzweckgrafikprozessors ('General Purpose GPU', GPGPU) etabliert.

Vom Grafikkartenhersteller nVidia [w54] gibt es inzwischen sogar komplette Rechnersysteme – die Tesla-Systeme [w55] –, welche Hochleistungsrechnen durch die freie Nutzung der GPU einer oder auch mehrerer Grafikkarten ermöglichen. Diese Systeme können dabei mit einer unterschiedlicher Anzahl an GPUs erworben werden, wobei pro vorhandener GPU dabei laut Herstellerangaben eine Leistung von über 500 GFlops für Berechnungen erreicht wird.

Andererseits gibt es mittlerweile bereits einige Arbeiten zur Nutzung von transaktionalem Hauptspeicher. U.a. zeigt [61] einen Überblick über verschiedenste Systeme auf, welche mit transaktionalem (Haupt-)Speicher arbeiten, sowohl mittels reiner Softwarelösungen als auch mit (vorgeschlagenen) Hardwareansätzen. Hauptsächlich wird dabei aber der Schwerpunkt auf die Einhaltung der Transaktionsbedingungen gelegt und immer das Vorliegen einer Kombination aus (normalem) Hauptspeicher und einer CPU impliziert, welche nur gemeinsam (eventuell noch mit weiteren Komponenten) die gestellten Anforderungen erfüllen. Gerätespeicher wird in diesen Ansätzen allerdings nicht berücksichtigt. Da die an dieser Stelle vorgestellten, einzelnen Arbeiten mit teilweise sehr restriktiven Einschränkungen auf ihren jeweiligen konzipierten Einsatzbereich ausgerichtet sind, können sie daher auch kaum bis gar nicht auf eine transaktionale Arbeitsweise für einen Grafikkartenspeicher oder einen allgemeinen Gerätespeicher übertragen werden.

Dahingegen existieren inzwischen bereits schon erste Ansätze zu Prozessoren, welche bereits intern mit Transaktionen arbeiten bzw. transaktionalen Speicher direkt durch die Hardware ansprechen können [66]. Ausgelegt sind diese (theoretischen) Arbeiten hauptsächlich für einen Einsatz im Multiprozessorbereich, um weitergehende oder explizite Synchronisierungsmaßnahmen zwischen einzelnen Prozessoren bzw. deren Prozessorkernen zu vermeiden. Sun Microsystems hat inzwischen sogar eine erste Realisierung eines solchen Prozessors bewerkstelligt – den 'Rock'-Prozessor ([67], [w56]) –, welcher bis 2009 auf den Mark kommen soll.

4.4 Bewertung

Eine direkte Einbeziehung von beliebigen Gerätespeichern in einen transaktionalen Szenengraphen ist durch die mangelnde Ausstattung und Flexibilität von bestehender Standardhardware momentan nicht realistisch. Einzig moderne Grafikkarten kommen durch ihre Leistungsfähigkeit und ihren relativ hohen Grad an Autonomie durch den (zumindest teilweise) frei programmierbaren Grafikprozessor in Frage.

Mit ihnen ist eine solche Integration möglich, allerdings ist damit ein nicht unbeträchtlicher Aufwand zur Verwaltung und Verteilung des Grafikkartenspeichers verbunden. Dieser Aufwand lässt sich im Wesentlichen auf die unzureichende Kommunikation zwischen der Grafikkarte und dem restlichen Rechnersystem zurückführen, welche hauptsächlich auf eine unidirektionale Übertragung von Daten zur Grafikkarte hin ausgelegt ist, und somit die Erstellung einer (Hardware-)Speicherverwaltung innerhalb der Grafikkarte und eines dazugehörigen Mechanismus zum selbständigen Anfordern neuer Grafikdaten erschwert bzw. verhindert.

Die aktuellen Forschungen auf diesem Gebiet und die Weiterentwicklungen der Grafikkartenhardware lassen jedoch erkennen, dass die hierfür eingesetzten Hardwarestrukturen zunehmend universeller und leistungsfähiger werden – nicht nur bezüglich ihrer Rechenleistung sondern auch bezüglich ihrer Kommunikationsmöglichkeiten mit dem restlichen Rechnersystem, was die Grundvoraussetzung für eine Realisierung einer grafikkarteneigenen Speicherverwaltung darstellt. Insbesondere die Spezifikationen von Microsoft zum neuen WDDM-Treibermodell für Windows Vista ab Version 2.0 und 2.1 erfordern eine Anpassung und Erweiterung der

Grafikkartenhardware von den Herstellern, welche genau die beschriebenen Defizite betreffen (siehe Kapitel 4.2.3). Zukünftige Generationen von Grafikkarten werden daher durch ihre verbesserte Hardwareunterstützung und der selbständigen Anforderung von benötigten Daten deutlich geeignetere Voraussetzungen bieten, um darauf aufbauend eine direkte und vor allem effiziente Verteilung des Grafikkartenspeichers vornehmen zu können.

Demgegenüber steht die transaktionale Konsistenz durch ihre schwierigere Umsetzbarkeit innerhalb von (Grafikkarten-)Hardware noch deutlich weiter entfernt von einer Integration in zukünftige Standardhardwarekomponenten. Diese kann deshalb, vergleichbar zu den Konzepten von bestehenden transaktional arbeitenden Betriebssystemen (wie z.B. Plurix), in Software durch ein geeignetes Programmiermodell zur Verfügung gestellt werden und durch Unterstützung eines eingesetzten Compilers weiter vereinfacht werden. Ein Blick auf den zunehmenden Einsatz von transaktionaler Konsistenz in den Forschungsarbeiten anderer Systeme lässt aber sogar vermuten, dass dieses Konzept zukünftig auch Einzug in die Arbeitsweise von weiterentwickelten (Grafik-)Prozessoren finden und somit langfristig sogar eine direkte und vollständige Hardwareunterstützung eines transaktionalen Szenengraphens möglich sein wird.

Insgesamt würde allerdings bereits durch die Kombination einer in dieser Arbeit vorgeschlagen Speicherverwaltung zur Verteilung der Grafikdaten mit Hilfe der Hardwareunterstützung kommender Grafikkarten und einem transaktional konsistenten Programmiermodell eine ausreichende Basis für eine effiziente und hardwarenahe Unterstützung von transaktionalen Szenengraphen geschaffen, über die sich darauf aufbauende Präsenzsysteme intuitiv und leistungsfähig unterstützen ließen.

4.5 Zusammenfassung

Die hardwareseitige Unterstützung eines transaktionalen Szenengraphens erfordert gewisse Mindestvoraussetzungen an die hierfür verwendbare Hardware, d.h. sie muss ein bestimmtes Mindestmaß an eigenem Speicher und einen gewissen Grad an Autonomie innerhalb des eingesetzten Rechnersystems besitzen. Diesen Voraussetzungen kommen aktuelle Grafikkarten durch ihre mittlerweile flexiblen Grafikprozessoren und einem in der Regel relativ großen Grafikkartenspeicher am nächsten, weshalb sie in diesem Zusammenhang stellvertretend für einen allgemeinen Ansatz zu diesen Betrachtungen herangezogen wurden.

Um dabei die Unterschiede einer hardwareseitigen Unterstützung eines transaktionalen Szenengraphens besser zu der bisherigen Arbeitsweise von Grafikkarten aufzeigen zu können, wurden zuerst deren allgemeiner Aufbau und deren Strukturen näher erläutert. Da durch mangelnde Dokumentationen der Grafikkartenhersteller die hierzu benötigten Informationen allerdings nicht immer direkt verfügbar sind, wurden auch mögliche alternative Analyseverfahren mittels verschiedener Reverse-Engineering-Techniken beschrieben, um aufzuzeigen, wie bei einem solchen Vorgehen die erforderlichen Informationen dennoch erhalten werden können.

Es wurde anschließend detailliert auf die beiden wesentlichen Punkte für die hardwarenahe Unterstützung eines transaktionalen Szenengraphens eingegangen: die Einbeziehung des zugehörigen Geräte- bzw. Grafikkartenspeichers in eine allgemeine Verteilung des Speichers und die Unterstützung der transaktionalen Konsistenz. Eine Speicherverwaltung, die diese Punkte umzusetzen versucht, muss somit die hierbei auftretenden Besonderheiten berücksichtigen.

Für die Einbeziehung des Grafikkartenspeichers in ein allgemeines verteiltes Speichermodell wurde daher der Aufbau des Grafikkartenspeichers selbst und seine Ansteuerung durch die Grafikkarte sowie durch die CPU betrachtet. Hierdurch wird die Komplexität der unterschiedlichen

Adressräume erkennbar, bei der nur die CPU mit virtuellen Adressen umgehen kann. Eine GPU kennt dahingegen einzig rein physikalische Adressen, die sich in dieser Form nicht direkt für eine Verteilung von Grafikdaten aus dem Grafikkartenspeicher in andere Grafikkarten oder Rechner eignen. Hierfür wäre eine separate Speicherverwaltung für den Grafikkartenspeicher zuständig, welche hierzu aber einen Rückkanal für Anforderungen des Grafikprozessors nach neuen bzw. benötigten Daten an das restliche Rechnersystem benötigten würde, welcher in dieser Art und Weise bislang aber nicht in effizienter Form bei Grafikkarten existiert. Zusätzlich müsste auch eine Unterscheidung bei den Speicherorten von zu verteilenden Objekten vorgenommen werden, damit Grafikobjekte auch nur im Grafikkartenspeicher vorgehalten werden und nicht wie in bestehenden Systemen im Hauptspeicher zwischengespeichert werden, um dann vor der Benutzung durch die Grafikkarte in diese umkopiert werden zu müssen, und dadurch auch mehrfach im Speicher vorliegen. Ebenso muss von einer derart arbeitenden Speicherverwaltung die hierbei mögliche erreichbare Leistungsfähigkeit bzw. deren Beeinträchtigung durch die Kommunikation mit der Grafikhardware mit berücksichtigt werden.

Die hardwareseitige Unterstützung der transaktionalen Konsistenz für einen transaktionalen Szenengraphen erhöht zusätzlich die Komplexität. Bisherige Standardhardware – und somit auch Grafikkarten – arbeiten generell nicht transaktional bzw. nach den hierfür notwendigen Transaktionsbedingungen, sondern lassen diese durch die Software umsetzen. Dasselbe gilt infolgedessen auch für eine eventuelle Rücksetzung von Transaktionen oder eine Benachrichtigung anderer Stationen zu geänderten Daten. Ein mögliche Realisierung könnte durch Anpassungen oder Modifikationen der Arbeitsweise von Shader-Programmen innerhalb der Grafikkarte erfolgen, indem an dieser Stelle die benötigten transaktionalen Strukturen und Abläufe umgesetzt und von dort einer Anwendung bzw. Speicherverwaltung als eine Art von Programmiermodell oder Schnittstelle zur Verfügung gestellt wird.

Die bestehenden Arbeiten zu beiden Bereichen, welche auch in einer geeigneten Auswahl hier vorgestellt wurden, haben alle bislang experimentellen Charakter. Sie besitzen zwar realisierbare Ansätze für eine Umsetzung, erfordern aber in allen Fällen eine Anpassung der Hardware oder einen beträchtlichen Aufwand, um die von ihnen benötigten Strukturen durch die Software nachzubilden, und beinhalten daher auch Empfehlungen für eine Anpassung des ursprünglichen Hardwaredesigns, um eine effizientere Nutzung zu ermöglichen.

Die zukünftige Entwicklung der Grafikkartenhardware – insbesondere durch die aktuellen Ankündigungen von Microsoft bezüglich neuer Spezifikationen zu den zukünftigen Arbeitsweisen von Grafikkarten, welche als Voraussetzung für den kommenden Einsatz in Betriebssystemen ab Windows Vista bzw. kommenden WDDM-Versionen getätigt wurden – lässt aber bereits zahlreiche Umsetzungen zu den vorgestellten und vorgeschlagenen Ansätzen und besprochenen Anforderungen in naher Zukunft erkennen. So ist bei kommenden Grafikkartengenerationen und deren Arbeitsweisen eine deutlich bessere Unterstützung von Seiten der Hardware für die hier beschriebenen Konzepte zu erwarten, welche dadurch die (zumindest teilweise) hardwareseitige Unterstützung eines transaktionalen Szenengraphens deutlich leichter realisierbar werden lässt.

5. Messungen

Der im Rahmen dieser Arbeit erstellte und bereits vorgestellte Prototyp „World of Wissenheim“ (siehe Kapitel 2.4) belegt die generelle Machbarkeit des neuartigen Ansatzes, ein virtuelles Präsenzsystem mit einem transaktionalen Szenengraphen als Basis zu betreiben, und die Funktionsfähigkeit des zugrundeliegenden Konzeptes. Um auch die Praxistauglichkeit von Wissenheim aufzuzeigen, wurden zusätzlich Messungen mit diesem Prototypen durchgeführt.

Nachfolgend in diesem Kapitel werden die relevantesten der hierdurch gewonnenen Ergebnisse zum Betrieb von Wissenheim präsentiert. Weitere Messungen und detailliertere Ergebnisse sind zusätzlich auch in [68] aufgeführt.

5.1 Messaufbau

Für die Messungen wurde ein Rechnercluster bestehend aus homogenen Einzelplatzrechnern verwendet. Jeder dieser Rechner besteht aus Standard-PC-Komponenten und ist mit einem Mainboard von MSI, Modell K8T Neo2-F V2.0, ausgestattet. Dieses ist jeweils mit einem 64-Bit AMD-Prozessor des Typs Athlon 64 X2 Dual Core Pro 4200+ mit 2,2 GHz Taktfrequenz bestückt, von welchem von Plurix allerdings nur jeweils ein Kern verwendet wird[49]. Weiter verfügen alle Rechner über 1 GiB Hauptspeicher und eine Intel Netzwerkkarte des Modells 82540EM, welche sich sowohl mit Fast Ethernet als auch mit Gigabit Ethernet betreiben lässt.

Bei den durchgeführten Messungen wurde als Standardkonfiguration für die Datenrate des genutzten Netzwerks Fast Ethernet angenommen. Zur Vernetzung der Rechner wurde deshalb ein 100 MBit/s-Switch von Allied Telesyn des Typs AT-8024 verwendet. Für Vergleichsmessungen mit Gigabit Ethernet stand ein 1000 MBit/s-Switch, Modell GS924i, ebenfalls von der Firma Allied Telesyn, zur Verfügung.

Als Grafikkarte ist in allen Rechnern eine ATI Radeon 9250 mit 128 MiB Grafikkartenspeicher vorhanden. Diese zählt zwar nicht mehr zur aktuellen Grafikkartengeneration, Treiber für neuere Modelle standen unter Plurix zum Zeitpunkt der Erstellung dieser Arbeit aber (noch) nicht zur Verfügung (siehe Kapitel 4.1.3). Für das Nachfolgesystem von Plurix, Rainbow-OS, ist jedoch bereits ein neuer Grafikkartentreiber für die nachfolgende Grafikkartengeneration in Entwicklung (siehe [56]). Da die verwendeten Grafikkarten ebenfalls einen Einfluss auf die Messwerte haben – ein Großteil der Renderphase besteht aus der Darstellung der Grafikobjekte durch die Grafikkarte, wobei die hierfür benötigte Zeitdauer direkt von ihrer Leistungsfähigkeit abhängt – ist davon auszugehen, dass bei zukünftigen Messungen von Wissenheim mit den neueren Grafikkartentreibern noch weitere Verbesserungen zu den momentan ermittelten Messwerten auftreten werden.

49 Im Nachfolgesystem Rainbow-OS werden Multikernprozessoren vollständig unterstützt.

5.2 World of Wissenheim

Wissenheim verwendet zur Darstellung ein mehrstufiges Renderverfahren (siehe Kapitel 3.4.4.2). Der Rendervorgang wird dabei von der Wissenheim-Renderengine in zwei separate Phasen unterteilt: einer ersten Extraktionsphase, welche den Szenengraphen traversiert und die entnommenen Daten (genauer: nur die 'Hot Data'; siehe ebenfalls Kapitel 3.4.4.2) lokal zwischenspeichert, und einer zweiten, nachgeschalteten Renderphase, welche den eigentlichen Darstellungsvorgang anhand der gesammelten, stationslokalen Daten durchführt. Durch die Nutzung dieser ausschließlich lokalen Daten in der Renderphase kann ein Abbruch der zugehörigen Transaktion aufgrund von Kollisionen, verursacht durch parallel durchgeführte Schreibzugriffe von anderen (Render-)Transaktionen, gänzlich vermieden werden und somit auch das Kollisionsrisiko für den gesamten Rendervorgang gesenkt werden. Kollisionen und damit verbundene Abbrüche können somit nur noch in der Extraktionsphase auftreten.

Weiter ist die Laufzeit der Renderphase direkt proportional zur Anzahl der darzustellenden Objekte bzw. zu der Anzahl der den Objekten zugehörigen Eckpunkte (siehe [68]). Um daher vergleichbare Messungen zu realisieren, wurde für alle Messungen eine einheitliche Standardszene gewählt, welche aus rund 50.000 Eckpunkten besteht.

Werden während des Betriebs von Wissenheim keinerlei Modifikationen innerhalb des transaktionalen Szenengraphens vorgenommen, d.h. alle an Wissenheim teilnehmenden Stationen stellen lediglich die Standardszene dar, ohne sich darin zu bewegen oder Objekte zu animieren, so führt dies dazu, dass alle Renderengines der einzelnen Stationen ihren Rendervorgang auf vom Betriebssystem vorgehaltenen, lokalen Arbeitskopien der Szenengraphendaten durchführen. Dies gilt im Speziellen für die Extraktionsphase der einzelnen Stationen. Da von diesen jeweils nur lesend auf den Szenengraphen zugegriffen wird und ansonsten in einem unbewegten Szenario keine weiteren Schreibzugriffe auftreten, ist für den Rendervorgang keine weitere Kommunikation mit den anderen Stationen über das Netzwerk erforderlich. Folglich können alle Stationen die Standardszene mit ihrer maximalen, durch die Hardware bestimmten Geschwindigkeit und ungestört durch andere Stationen bzw. deren Transaktionen darstellen. Insbesondere bedeutet dies auch, dass keine Konflikte und somit Abbrüche während der Extraktionsphase auftreten. Diese Situation entspricht in diesem Fall einer rein statischen Welt, in welcher sich höchstens die (rein kamerabezogene) Blickrichtung einer lokalen Station ändert – vergleichbar mit der Kopfbewegung eines stehenden Avatars –, ohne dass eine Standortveränderung oder Drehung vorgenommen wird (dies würde ansonsten zu einem Schreibzugriff auf den Positions- bzw. Rotationskoordinaten des eigenen Avatars führen).

Da ein solches Vorgehen nicht dem realistischen Einsatz bzw. dem angenommenen typischen Verhalten eines virtuellen Präsenzsystems entspricht, wird aus diesem Grund für die durchgeführten Messungen eine (künstliche) Last erzeugt, welche kontinuierlich Modifikationen am gemeinsamen Szenengraphen vornimmt. Dadurch lässt sich eine Kommunikation der Stationen untereinander erzwingen und somit ein realistischeres Verhalten simulieren. Für die Messungen wird daher von jeweils einer der teilnehmenden Stationen pro durchgeführtem Renderdurchlauf ein Schreibzugriff im Szenengraphen vorgenommen. Diese Änderungen werden dann mit jedem Zugriff von anderen Stationen auf die betroffenen Szenengraphendaten automatisch vom Betriebssystem an diese weitergeleitet. Die eingefügte Last von einem Schreibzugriff pro Einzelbild simuliert dabei die aufsummierten Bewegungen und Animationen von mehreren Teilnehmern, da in der Regel davon auszugehen ist, dass Schreibzugriffe von einzelnen Stationen nicht mit jedem dargestelltem Bild

auftreten[50].

Für gewöhnlich ist aber auch im normalen Betrieb von Wissenheim nicht permanent mit einer (aufsummierten) Last von einer Modifikation pro Bild zu rechnen. Dies bedeutet, dass in aufeinanderfolgenden Renderdurchläufen, in denen dazwischen keine Änderungen am Szenengraphen durchgeführt werden, wiederum auf den vom Betriebssystem vorgehaltenen, lokalen Arbeitskopien des vorhergehenden Durchlaufs gearbeitet werden kann, da keine neueren Daten von einer schreibenden Station übertragen werden müssen. Dadurch erhöht sich in diesen Fällen die Bildwiederholrate bis zum eingangs erwähnten, durch die (Grafikkarten-)Hardware bestimmten Maximum; bei einer erneut auftretenden Last sinkt diese aber sofort wieder ab. Im Durchschnitt ergeben sich daher im Normalbetrieb höhere Bildwiederholraten als bei den nachfolgend aufgezeigten Messungen mit einer kontinuierlichen Last. Allerdings sind diese höheren, durchschnittlichen Bildwiederholraten auch mit größeren Schwankungen verbunden, je nach Auftreten der durchgeführten Schreibzugriffe. Daher wird für die hier vorgenommenen Messungen eine kontinuierliche Last verwendet, welche dadurch zwar insgesamt niedrigere Bildwiederholraten liefert, diese sind dafür aber gleichmäßiger und mit weniger Schwankungen behaftet und dadurch ihrerseits geeigneter, um die Messungen untereinander vergleichbar zu halten.

5.2.1 Bildwiederholraten

In Abbildung 43 werden die durchschnittlichen Bildwiederholraten von Wissenheim dargestellt – aufgetrennt in die Extraktions- und die Renderphase[51] –, welche bei einer konstanten Last von einem Schreibzugriff pro Rendervorgang auf den Szenengraphendaten ermittelt wurden. D.h. alle Stationen greifen pro Renderdurchlauf auf mindestens ein zur Darstellung benötigtes Objekt zu, an welchem sich Daten geändert haben. Somit liegen diese Daten nicht mehr gültig vom vorherigen Durchlauf lokal in der jeweiligen Station vor, sondern müssen über das Netzwerk (transparent für Wissenheim vom Betriebssystem) neu von der Station, die diese Änderungen vorgenommen hat, angefordert werden.

Es lässt sich hierbei erkennen, dass die Renderphase mit zunehmender Clustergröße nur gering in ihrer Leistung abnimmt, wohingegen die Extraktionsphase stärkeren Schwankungen unterworfen ist. Dieses Verhalten lässt sich anhand der auftretenden Kollisionen erklären.

Da die Renderphase aufgrund ihrer Arbeitsweise nicht durch Schreibkonflikte von anderen (Render-)Transaktionen abgebrochen werden kann, nimmt bei ihr die Bildwiederholrate nur geringfügig ab. Diese Verringerung liegt an einem mit zunehmender Clustergröße gesteigerten Kommunikationsaufwand zwischen den einzelnen Stationen und einem Anstieg der Dauer der Extraktionsphase, welche dadurch auch weniger Rechenzeit pro Sekunde für die Renderphase übrig lässt (vgl. Abbildung 44). Dennoch bleiben die Bildwiederholraten der Rendertransaktion hoch genug, um eine flüssige[52] Darstellung dauerhaft zu ermöglichen.

50 Ein Avatar wird in 3D-Welten typischerweise immer nur kurzzeitig und für kleine Stücke bewegt und verändert mit deutlich weniger Schritten pro Sekunde seine Position als es der Bildwiederholrate des Rendervorgangs entspricht. Ähnliches gilt für animierte Objekte, welche normalerweise nur so oft pro Sekunde animiert werden, dass ihre Bewegungen fließend aussehen und keine Ruckler beinhalten. Für sich langsam bewegende Objekte können hierfür beispielsweise schon 5-10 Wiederholungen pro Sekunde ausreichen. Selbst wenn die eigentliche Bildwiederholrate des Rendevorgangs deutlich darüber liegt, werden üblicherweise die animierten Objekte nicht proportional hierzu häufiger bewegt, sondern verbleiben auf dem minimal nötigen Niveau, um Rechenzeit einzusparen.

51 Im Diagramm wird die Wiederholrate der Extraktionsphase ebenfalls wie für die Renderphase mit FPS bezeichnet, auch wenn diese Bezeichnung nur sinngemäß für die erfolgreichen Wiederholungen pro Sekunde gedacht ist und nicht mit der tatsächlichen Bildwiederholrate der Renderphase verwechselt werden darf.

52 Für gewöhnlich bedeuten flüssige Wiederholraten Werte ab ca. 18-25 FPS, da ab diesen Werten das menschliche Auge die Einzelbilder nicht mehr unterscheiden kann.

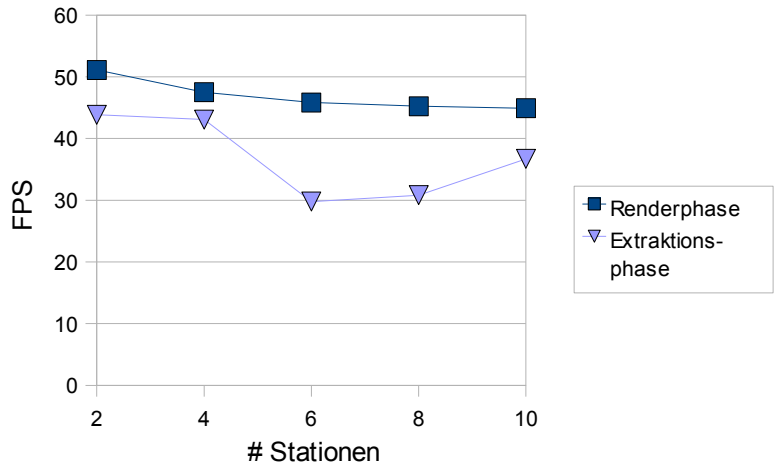

Abbildung 43: Bildwiederholraten von Wissenheim

Die Extraktionsphase dahingegen ist durch mögliche Schreibkonflikte, ausgelöst durch die vorhandene Last, kollisions- und dadurch auch abbruchgefährdet. Mit dem Auftreten einer Kollision und dem daraus folgenden Abbruch der Extraktionsphase wird diese erst mit dem nächsten Renderdurchlauf wieder neu aufgerufen. Hierdurch fehlt durch den Abbruch eine Aktualisierung der von der Renderphase nachfolgend dargestellten Daten und führt somit auch zu einem Absinken der Wiederholrate. Dennoch liegen die hier durchgeführten Aktualisierungen insgesamt in einem Bereich, bei dem diese Schwankungen bzw. das Absinken der Wiederholraten von einem Benutzer nicht zu bemerken sind.

5.2.2 Laufzeit

Abbildung 44 stellt dieselbe Situation wie zuvor nochmals anhand der Laufzeiten der beiden einzelnen Phasen dar. Anhand dieser dargestellten Messwerte ist ersichtlich, dass zu dem zuvor beschriebenen Kollisionsrisiko noch ein weiter Aspekt hinzukommt, welcher sich in der Laufzeit der einzelnen Phasen widerspiegelt.

Mit jeder hinzukommenden Station für den Betrieb von Wissenheim steigt mit dem für sie benötigten Kommunikationsaufwand die Anzahl an Paketen auf dem eingesetzten Netzwerk, über welches die geänderten Szenengraphendaten angefordert und zurückgeliefert werden. Demzufolge dauert nun einerseits das durchschnittliche Versenden eines Pakets mit jeder hinzukommenden Station länger. Da es durch das gesteigerte Paketaufkommen zu häufigeren Paketkollisionen auf dem Netzwerk selbst kommt, vergeht durchschnittlich mehr Zeit bis zu einer erfolgreichen Übertragung. Andererseits fordern mit zunehmender Clustergröße auch mehr Stationen die geänderten Daten von der schreibenden Station an, weshalb auch die Abarbeitung der Anfragen insgesamt mehr Zeit in Anspruch nimmt, da diese von der modifizierenden Station in der Reihenfolge des Eintreffens sequentiell beantwortet werden. Aufgrund dieser beiden Faktoren steigt die Laufzeit der Extraktionsphase an, was umgekehrt eine Verringerung der möglichen Wiederholraten dieser Phase pro Sekunde bedeutet (siehe zuvor in Abbildung 43).

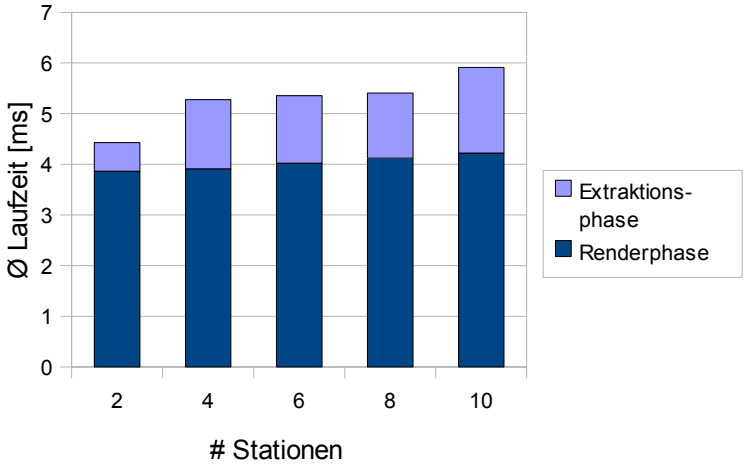

Abbildung 44: Verhältnis der Extraktionsphase zur Renderphase

Dahingegen bleibt die Laufzeit der Renderphase nahezu gleich bzw. nimmt nur unwesentlich zu. Der geringfügige Anstieg lässt sich ebenfalls auf das gesteigerte Paketaufkommen zurückführen. Da die Pakete auf dem Netzwerk (ausgelöst durch die Kommunikation der jeweiligen Extraktionsphasen) teilweise als Rundspruch versandt werden, müssen im Moment des Erhalts alle Stationen kurzfristig überprüfen, ob sie der Inhalt des empfangenen Pakets betrifft. Auch wenn hierbei nur festgestellt wird, dass diese Nachricht für eine andere Station bestimmt war und sie daher verworfen werden kann, wurde hierdurch etwas an Rechenzeit verbraucht, welche die Ausführung der gerade laufenden Transaktion verlängert – so auch im Fall der für die Renderphase zuständigen Transaktion.

Eine Optimierung hierzu, welche sowohl dem Anstieg der Laufzeiten der einzelnen Phasen entgegenwirkt, als auch das Paketaufkommen auf der Netzwerkschicht insgesamt vermindert, stellt die Einführung einer Kommunikation via Multicast zur gemeinsamen Anforderung bzw. Beantwortung der geänderten Szenengraphendaten dar. Da sich die Anfragen der teilnehmenden Stationen auf dieselben Änderungen beziehen, muss dann insbesondere die Beantwortung der Anfragen nicht mehr an alle anfragenden Stationen einzeln vorgenommen werden, sondern kann in diesem Fall einmalig als Mitteilung an die Multicast-Gruppe erledigt werden und alle anfragenden Rechner erhalten somit gemeinsam eine Antwort. Zusätzlich werden beim Einsatz einer Multicast-Kommunikation „außenstehende“ Rechner, d.h. Clusterrechner, welche nicht an Wissenheim teilnehmen, nicht mehr von normalen Rundsprüchen oder Paketen, welche für diese Stationen nicht relevant sind, davon gestört.

Als weitere Optimierung kann durch eine Umstellung der bislang Invalidierung-basierten Benachrichtigung anderer Stationen über die durchgeführten Schreibzugriffe hin zu einem Update-basierten Übermitteln der geänderten Daten an alle interessierten Stationen das Paketaufkommen nochmals reduziert werden. Im günstigsten Fall lassen sich dadurch alle Anfragen nach geänderten Szenengraphendaten gänzlich einsparen, da die einzelnen teilnehmenden Stationen direkt mit der Benachrichtigung einer Änderung diese geänderten Daten auch gleich mitgeliefert bekommen.

Eine Umsetzung dieser Optimierungen ist für kommende Versionen von Wissenheim unter dem Plurix-Nachfolger Rainbow-OS geplant.

5.2.3 Kollisionsaufkommen

Wird bei den durchgeführten Messreihen das Kollisionsaufkommen der Extraktionsphase genauer betrachtet, lässt sich eine weitere Facette im Verhalten von Wissenheim erkennen, welche im nachfolgenden Diagramm dargestellt wird.

Während des kontinuierlichen Rendervorgangs der einzelnen Stationen kommt es nicht in jedem Durchlauf zu einem Abbruch der Extraktionsphase, sondern nur in denjenigen Fällen, wenn eine Kollision durch einen gleichzeitigen Schreibzugriff einer modifizierenden Station auf den ausgelesenen Szenengraphendaten stattfindet. Sobald allerdings ein Abbruch der Extraktionsphase auftritt kann es vorkommen, dass durch die Rücksetzung und dem Neustart der zugehörigen Transaktion die dabei benötigten Szenengraphendaten bereits erneut wieder von einer anderen, schnelleren Station modifiziert werden, bevor die Extraktionsphase abgeschlossen ist, was zu einem nochmaligen Abbruch führt.

Abbildung 45 zeigt die durchschnittliche Anzahl an zusammenhängenden, d.h. direkt aufeinanderfolgenden Abbrüchen der Extraktionsphase, bevor diese wieder erfolgreich durchlief für denjenigen Fälle, in denen doch ein Abbruch bei der Extraktionsphase auftrat. Da die Renderphase im Gegensatz dazu selbst nicht kollisionsgefährdet ist, wurde diese hier nicht weiter berücksichtigt.

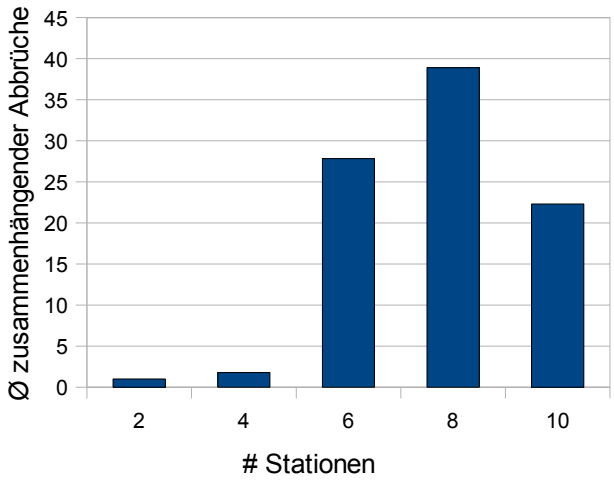

Abbildung 45: Anzahl zusammenhängender Abbrüche der Extraktionsphase beim Auftreten von Abbrüchen

Es lässt sich nun erkennen, dass bei wenigen teilnehmenden Stationen an Wissenheim kaum mehrfache Abbrüche bei der Extraktionsphase hintereinander auftreten. Mit zunehmender Anzahl steigt dieses Verhalten aber sprunghaft an und verbleibt dann auf hohem Niveau.

Anhand der vorherigen Diagramme war ersichtlich, dass die Wiederholraten der Extraktionsphase im akzeptablen Bereich auch bei ansteigender Clustergröße verbleiben. Kombiniert man nun diese Aussagen mit dem in diesem Diagramm ersichtlichen Sachverhalt, so lässt sich nachvollziehen, dass es trotz ausreichender Wiederholraten kurzfristig auch zu erkennbaren Verzögerungen bei der Aktualisierung der von der Renderphase benötigten lokalen Grafikdaten aufgrund des steigenden Paketaufkommens und ungünstigem Kollisionsverhalten bei den Stationen kommt.

Die Auswirkung dieser Situation ist partiell sogar im laufenden Betrieb von einem Benutzer bemerkbar. Da die Renderphase selbst in ihrer Ausführung nicht beeinträchtigt wird, bleiben die Bildwiederholraten für die Darstellung gleich und sind somit ausreichend, um ein „Ruckeln“ der Anzeige zu verhindern. Die kurzzeitig hohe Anzahl an zusammenhängenden Abbrüchen von der

Extraktionsphase bewirken allerdings in diesem Moment ein Ausbleiben der Aktualisierung der darzustellenden Szenengraphendaten, weshalb – trotz flüssiger Bildwiederholraten der Renderphase – sich bewegende Objekte durch ein teilweise „springendes“ Verhalten in Wissenheim auffallen, d.h. sie verbleiben kurzzeitig an ihrer alten Position und tauchen danach plötzlich an anderer Stelle wieder auf, ohne dass eine nahtlose Bewegung dazwischen liegt.

Untersucht man die gemessenen Werte genauer, so lässt sich weiter aufdecken, dass diese teilweise hohen zusammenhängenden Abbruchwerte zumeist nur bei einer einzelnen Station auftreten und nicht gleichmäßig über alle Stationen verteilt sind. Aufgrund ungünstiger Bedingungen wird die Extraktionsphase dieser Station bzw. die zugehörige Transaktion gegenüber den Transaktionen anderer Stationen benachteiligt. Leichte Abweichungen auf verschiedensten Ebenen können hierfür bereits die Ursache sein (z.B. ein versetzter Startzeitpunkt des Rendervorgangs, auftretende Netzwerkkollisionen, eine geringfügig langsamer arbeitende Hardware, ...). Letztlich resultiert dies in einem unfairen Verhalten des Gesamtsystems dieser Extraktionsphase bzw. der zugehörigen Transaktion gegenüber. Da von dem für die Messungen eingesetzten Plurix-System bislang allerdings noch keine Betrachtung zu vom System ungleich behandelten Transaktionen vorgenommen wird, lässt sich das hier beschriebene Verhalten nicht gänzlich ausschließen. Die kommende Einbeziehung von Fairness-Mechanismen in Rainbow-OS wird zukünftig für solche Fälle allerdings eine gleichmäßigere Verteilung und dadurch auch insgesamt niedrigere Abbruchwerte aller Extraktionsphasen liefern (siehe [69]).

5.3 Wissenheim-Lasttest

Mit der folgenden Messreihe wurden dieselben Auswirkungen auf den Betrieb von Wissenheim untersucht wie zuvor, allerdings wurde eine weitere, signifikante Erhöhung der darin auftretenden Last vorgenommen („Stresstest“), wie sie im normalen Betrieb nur selten auftreten wird. Die Last wurde dabei nicht mehr auf einen einzelnen, kontinuierlichen Schreibzugriff pro Renderdurchlauf beschränkt, sondern abhängig von der Clustergröße durch kontinuierliche Schreibzugriffe von mindestens einem Drittel der teilnehmenden Stationen vorgenommen[53]. Dies beinhaltet dadurch eine Verteilung der erzeugten Last auf mehrere Stationen, welche Modifikationen vornehmen, so dass es den normalen Betrieb von Wissenheim auch unter diesen extremeren Bedingungen geeignet nachstellt.

5.3.1 Bildwiederholraten

Die Auswirkung der Lasterhöhung auf die Wiederholraten der Extraktions- und Renderphase wird in Abbildung 46 wiedergegeben.

Durch den von der erhöhten Last verursachten Anstieg des Kommunikationsaufkommens zwischen den teilnehmenden Stationen sinken erwartungsgemäß die Wiederholraten der beiden Phasen stärker ab als zuvor. Im Besonderen trifft dies auf die Extraktionsphase zu, was sich auf den mit steigender Anzahl an Schreibvorgängen verbundenen Anstieg an auftretenden Kollisionen zurückführen lässt. Die durchschnittliche Wiederholrate der Extraktionsphase vermindert sich dabei zwar bis an die Grenze der flüssigen Wiederholraten, verbleibt aber noch im akzeptablen Bereich.

53 Im Falle von 2 bzw. 4 Stationen nehmen die Hälfte der Rechner schreibende Zugriffe vor, bei den restlichen Messungen gerundet ein Drittel der vorhandenen Stationen.

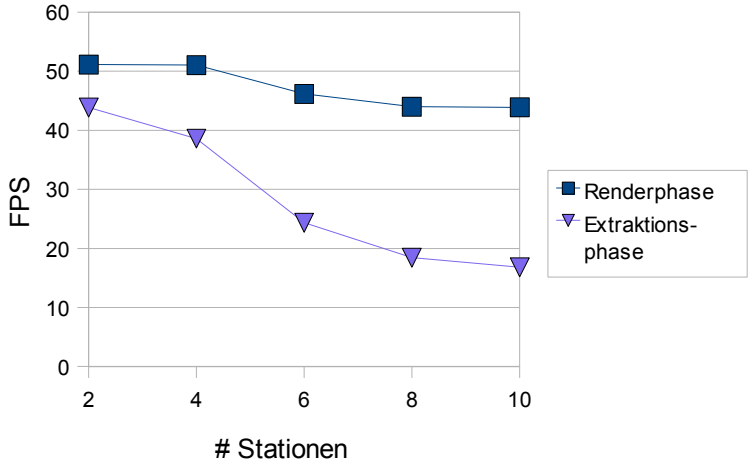

Abbildung 46: Bildwiederholraten von Wissenheim bei hoher Last

Dahingegen bleibt auch in diesem Fall die Renderphase verhältnismäßig stabil und liefert weiterhin relativ gleichbleibende Bildwiederholraten.

5.3.2 Laufzeit

Das eben gezeigte Verhalten lässt sich anhand der Laufzeiten der einzelnen Phasen noch detaillierter darstellen, wie in Abbildung 47 zu sehen ist.

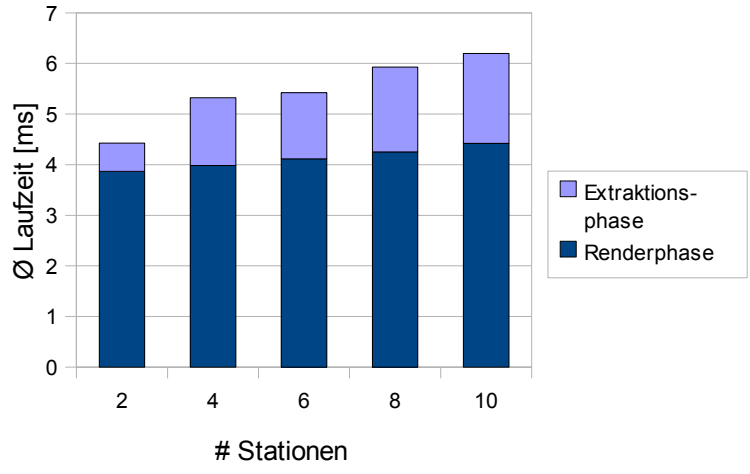

Abbildung 47: Verhältnis der Extraktionsphase zur Renderphase bei hoher Last

Die Verlängerung der Laufzeiten der Extraktions- und Renderphase lässt sich erneut auf den stärkeren Anstieg des durch mehrere schreibende Stationen ebenfalls stärker zunehmenden Paketaufkommens auf der Netzwerkebene zurückführen. Zusätzlich wirkt sich auf die Extraktionsphase noch das insgesamt höhere Kollisionsrisiko durch die vermehrt auftretenden Schreibzugriffe verstärkt aus. Dennoch bleiben die Werte im tolerierbaren Bereich.

Für diese Fälle bieten sich daher auch die gleichen Optimierungsstrategien wie zuvor an, um auftretende Kollisionen zu minimieren bzw. die Paketlast auf der Netzwerkebene zu senken, indem beispielsweise Multicast-Kommunikation oder auch eine Umstellung auf ein Update-basiertes Aktualisierungsverfahren eingesetzt wird.

5.3.3 Kollisionsaufkommen

Die Betrachtung, im Falle des Auftretens von Abbrüchen, der durchschnittlich zusammenhängenden Abbrüche durch Kollisionen während der Extraktionsphase unter hoher Last wird in der nachfolgenden Abbildung aufgezeigt.

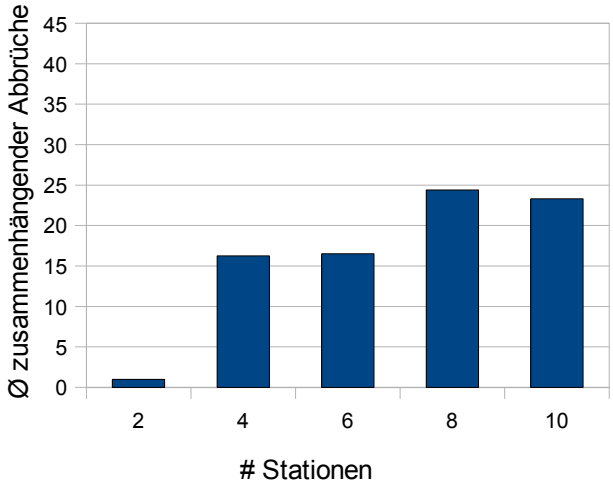

Abbildung 48: Anzahl an zusammenhängenden Abbrüchen der Extraktionsphase bei hoher Last

Es ist klar zu erkennen, dass es bei hoher Last zu einem früheren Anstieg an zusammenhängenden Abbrüchen der Extraktionsphase kommt als zuvor mit nur einer einzelnen schreibenden Station (vgl. Abbildung 45). Allerdings bewegen sich die Werte in diesem Fall nicht in einem so hohen Bereich wie zuvor, da die auftretende Last nun stärker zwischen mehreren Stationen aufgeteilt ist. Dadurch werden eventuelle Benachteiligungen einzelner Stationen durch ungünstige Bedingungen oder unfaires Abbruchverhalten im Cluster oder auf Netzwerkebene besser ausgeglichen.

Die aufeinanderfolgenden Abbrüche treten somit durch das gesteigerte Kollisionsrisiko insgesamt zwar häufiger auf (was sich in den sinkenden Wiederholraten bei Abbildung 46 widerspiegelt), bleiben aber in ihren Maximalwerten niedriger, als bei einem einzelnen modifizierenden Teilnehmer.

5.4 Netzwerkeinfluss

Bei den vorgenommenen Messungen wurde auch der Einfluss untersucht, den das verwendete Netzwerk auf den Betrieb von Wissenheim hat. Im Speziellen wurde hierbei die Datenrate des Netzwerks erhöht. Das Ergebnis wird in der folgenden Messreihe aufgezeigt. Die Auswirkungen des Netzwerkes betreffen dabei erwartungsgemäß nur diejenigen Transaktionen, welche eine Kommunikation mit anderen Stationen beinhalten, d.h. speziell bei Wissenheim die Extraktionsphase des Rendervorgangs. Auf die Verarbeitungsgeschwindigkeit der Renderphase hat eine Veränderung des eingesetzten Netzwerkes folglich keine relevante Auswirkung, da diese jeweils gänzlich lokal arbeitet und keine Kommunikation mit anderen Stationen benötigt. Sie wurden daher im aufgezeigten Diagramm nicht weiter berücksichtigt.

Der Einfluss des Netzwerks wird mit jeder Zunahme der Kommunikations- bzw. Paketlast darauf umso deutlicher sichtbar. Für die durchgeführte Messreihe wurde daher zum Vergleich der unterschiedlich verwendeten Netzwerkgeschwindigkeiten eine hohe auftretende Last für den Betrieb

von Wissenheim herangezogen, wie sie bereits in Kapitel 5.3 verwendet wurde.

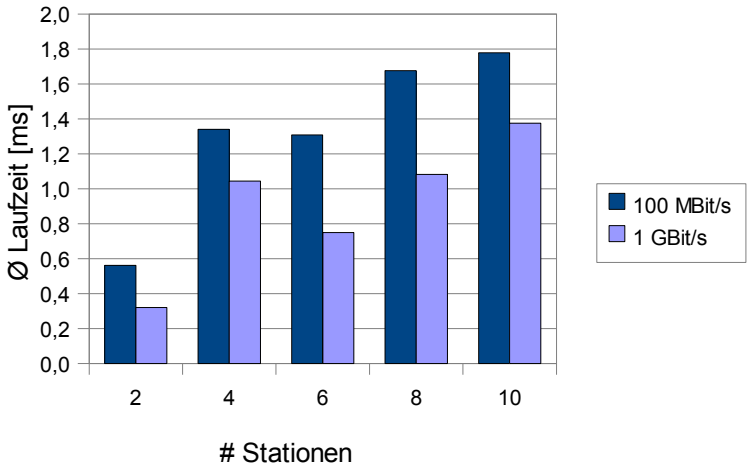

Abbildung 49: Auswirkung des Netzwerks auf die Laufzeit der Extraktionsphase bei hoher Last

Anhand des Diagramms lässt sich deutlich ablesen, dass die Erhöhung der Datenrate des eingesetzten Netzwerkes wie erwartet zu einem Absinken der Laufzeiten der Extraktionsphasen auf den einzelnen Stationen führt. Ein schnelleres Netzwerk lässt die Anfragen und Antworten nach den geänderten Daten schneller übertragen, wodurch gleichzeitig die auftretenden Netzwerkkollisionen vermindert werden. Da auch die Anzahl der Anforderungen insgesamt gleich bleibt, nehmen diese wegen des höheren Durchsatzes eine kürzere Zeitspanne auf dem Netzwerk ein. Folglich reduziert sich auch zusätzlich das Kollisionsrisiko der Netzwerkpakete untereinander.

Die Latenzen der Datenanforderungen und deren Beantwortung hierbei werden durch ein schnelleres Netzwerk allerdings nur wenig verbessert, da zwar die Übertragung schneller vonstatten geht, ein wesentlicher Teil der Abarbeitung davon aber erst von den beteiligten Stationen erledigt wird.

Zusammenfassend kann somit zum Einsatz eines schnelleren Netzwerkes ausgesagt werden, dass sich die Laufzeiten der Extraktionsphasen hierdurch zwar reduzieren lassen, womit sich einhergehend auch die Wiederholraten der zugehörigen Transaktion verbessern; generell sind die Auswirkungen aber nicht so groß, als dass das verwendete Netzwerk den wichtigsten Punkt für eine Leistungsoptimierung darstellt. Umgekehrt lässt sich somit aber auch feststellen, dass der vorgestellte Prototyp von Wissenheim nicht zwingend auf ein schnelles Netzwerk zum flüssigen Betrieb angewiesen ist.

5.5 Synchronisierter Betrieb von Wissenheim

Im Gegensatz zu den bisherigen Messungen, bei welchen die Rendervorgänge der einzelnen Stationen unsynchronisiert voneinander arbeiteten, wurden auch Messreihen mit einer explizit synchronisierten Version von Wissenheim durchgeführt (siehe Kapitel 3.4.4.3).

Zu diesem Zweck wurde zu Beginn des mehrstufigen Rendervorgangs (d.h. somit an den Anfang der Extraktionsphase) eine Barriere eingefügt, welche sich darum kümmert, dass alle Stationen jeweils gemeinsam mit der weiteren Verarbeitung beginnen. Diese Synchronisierung sorgt dann dafür, dass die jeweiligen einzelnen Phasen gleichzeitig bzw. parallel von allen Stationen

abgearbeitet werden. Insbesondere werden dadurch die jeweiligen Extraktionsphasen gleichzeitig ausgeführt, was deren Kollsisionsrisiko wesentlich vermindert bzw. sogar vollständig reduziert, da Schreibzugriffe auf die von der Extraktionsphase gelesenen Szenengraphendaten zu diesem Zeitpunkt nicht mehr stattfinden können, sondern frühestens nach Abschluss der nachfolgenden Renderphasen auftreten. Dadurch bewirkt die Barriere ebenfalls eine Parallelisierung der Schreibvorgänge von sich bewegenden bzw. animierenden Stationen nach dem Rendervorgang.

Der Einsatz von Barrieren führt somit einerseits zu einer Trennung der ansonsten verzahnten bzw. unsynchronisierten Schreib- und Lesezugriffe durch die einzelnen Phasen bzw. den zugehörigen Transaktionen der einzelnen Stationen, andererseits wird allerdings auch durch die Barrieren selbst ein zusätzlicher Aufwand zur Synchronisierung der Barrieren notwendig. Pro Barriere sind im günstigsten Fall zwei zusätzliche Nachrichten pro Station erforderlich: eine Meldung von jeder Station an die Barriere, dass sie alle Berechnungen erledigt haben und nun an der Barriere warten, und eine zweite Meldung von der Barriere zurück an die Stationen, dass sie nun mit dem nächsten Rendervorgang fortfahren können. Als eine Optimierung kann auch weitergehend die Rückmeldung von der Barriere an die Stationen zum Weiterarbeiten per Multicast-Nachricht oder zumindest mittels Rundspruch effizienter vorgenommen werden. Die für die Messungen implementierte Barrierensynchronisierung arbeitet allerdings nicht mit dem optimierten Prinzip, da sich dies nicht ohne Änderungen am zugrundeliegenden Betriebssystem umsetzen ließe. Prinzipiell können auch andere, teilweise effizientere Synchronisierungsmechanismen anstelle von Barrieren verwendet werden; Barrieren stellen aber eine intuitive und einfach zu integrierende Möglichkeit zur Synchronisierung dar, ohne tiefergehende bzw. überhaupt Änderungen an einem System zu deren Einsatz vorauszusetzen.

Für die durchgeführten Messreihen wurde dabei eine „normale” Last von einer schreibenden Station pro Rendervorgang herangezogen (siehe Kapitel 5.2).

5.5.1 Bildwiederholraten

Die sich im synchronisierten Betrieb von Wissenheim ergebenden (Bild-)Wiederholraten der Extraktions- und Renderphase sind in Abbildung 50 dargestellt.

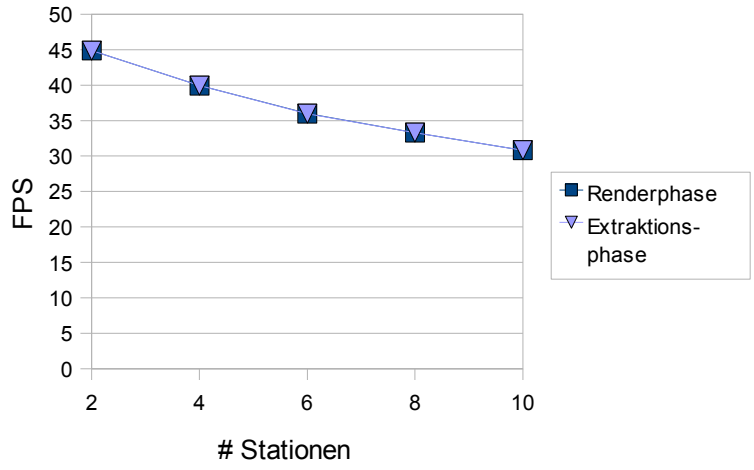

Abbildung 50: Bildwiederholraten bei Synchronisierung mit einer Barriere

Durch die Kopplung der Extraktions- und der Renderphase mittels einer Barriere ist auch der

identische Verlauf dieser beiden Phasen eindeutig ersichtlich. Speziell die Extraktionsphase zeigt im Vergleich zum unsynchronisierten Fall (vgl. Abbildung 43) in ihrem Verlauf ein deutlich gleichmäßigeres und stabileres Verhalten auf.

Es lässt sich allerdings auch erkennen, dass sich die Gesamtleistung der beiden Phasen im Gegensatz zum unsynchronisierten Betrieb etwas vermindert hat. Dies lässt sich auf den zusätzlichen Aufwand für den Betrieb der Barriere zurückführen.

Eine weitere Reduktion der Wiederholraten ist bei der Verwendung unterschiedlicher Hardware zu erwarten, da sich hierbei die von Barrieren typische Eigenschaft, dass der langsamste Teilnehmer letztlich das Tempo des gesamten Rendervorgangs bestimmt, stärker auswirken wird. Dem ließe sich aber auch in einer heterogenen Hardwareumgebung mit weitergehenden und geeigneten Optimierungsvarianten teilweise entgegenwirken, was im vorliegenden Messaufbau aber nicht vonnöten war. Umgekehrt lässt sich so aber auch feststellen, dass in einer homogenen Hardwareumgebung der Einsatz von Barrieren ein geeignetes Synchronisierungsverfahren darstellt, da ein Ausbremsen durch langsame Teilnehmer nicht in ausgeprägtem Maße auftritt.

5.5.2 Kollisionsaufkommen

Besonders deutlich wird der Unterschied zwischen einer synchronisierten und einer unsynchronisierten Arbeitsweise von Wissenheim, wenn das bereits zuvor erwähnte Kollisionsaufkommen der Extraktionsphase (siehe Kapitel 5.2.3 bzw. 5.3.3) während eines Kollisionsfalles anhand der dann möglich auftretenden, zusammenhängenden Abbrüche betrachtet wird. Abbildung 51 stellt hierzu die gewonnenen Messdaten dar.

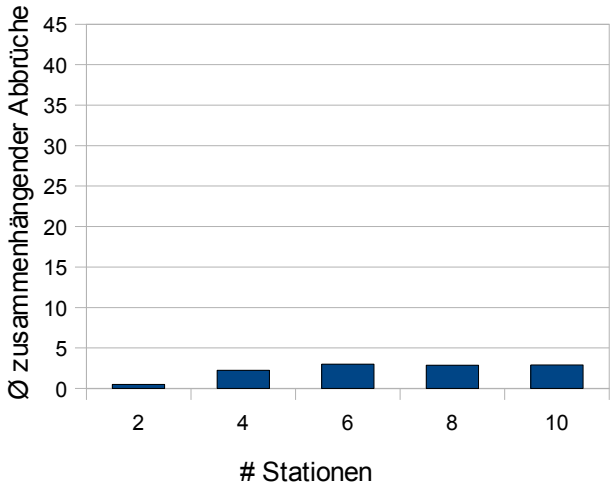

Abbildung 51: Anzahl an zusammenhängenden Abbrüchen der Extraktionsphase bei Einsatz einer Synchronisierung

Im Gegensatz zu den unsynchronisierten Fällen (vgl. Abbildung 45 und Abbildung 48) lässt sich mit dem Einsatz einer Synchronisierung erkennen, dass es nahezu keine aufeinanderfolgenden Abbrüche mehr gibt[54]. Die Verbesserung dabei liegt im besten Fall sogar über Faktor zehn. Dieses Verhalten bedeutet, dass selbst mit zunehmender Clustergröße nicht nur von der Renderphase wie

54 Die verbleibenden Abbrüche der Extraktionsphase hängen mit Abbrüchen der Barriere selbst durch die gesteigerte Netzwerkkommunikation zusammen, da die Barriere keine eigenständige Transaktion darstellt sondern an die Extraktionsphase gekoppelt ist und sich dadurch die Abbrüche an den einzelnen Stellen nicht voneinander trennen lassen. Genau genommen handelt es sich somit um Abbrüche der Barriere; die Anzahl an zusammenhängenden Abbrüchen der Extraktionsphase entspräche dabei dem konstanten Wert von Null.

im unsynchronisierten Fall hohe Bildwiederholraten geliefert werden, sondern auch die Extraktionsphase nun genauso hohe Aktualisierungen der darzustellenden Szene ermöglicht. Dadurch tritt im synchronisierten Betrieb von Wissenheim das ansonsten teilweise erkennbare „Springen" von Objekten und Avataren nicht mehr auf. Einem Benutzer verhilft dieses Verhalten dadurch auch zu einem subjektiv flüssigeren Bewegungsablauf innerhalb von Wissenheim.

Insgesamt lässt sich dadurch feststellen, dass mit zunehmender Anzahl an teilnehmenden Stationen für einen dauerhaft flüssigen Betrieb von Wissenheim der Einsatz einer Synchronisierung empfohlen bzw. sogar erforderlich wird. Im unsynchronisierten Betrieb bleibt zwar die nötige Kommunikation der Stationen untereinander deutlich geringer, da der zusätzliche Kommunikationsaufwand durch die Synchronisierung entfällt (was sich auch in einem niedrigeren Netzwerkpaketaufkommen widerspiegelt), durch die zeitliche Entkopplung der Arbeitsphasen von Wissenheim – insbesondere während des Rendervorgangs – treten aber verstärkt Kollisionen bei den zugehörigen (Extraktions-)Transaktionen durch zeitgleiche Schreibzugriffe auf den gemeinsamen Szenengraphendaten auf. Diese wirken sich dabei erheblich ungünstiger auf den Betrieb von Wissenheim aus, als das gesteigerte Netzwerkpaketaufkommen im (barrieren-)synchronisierten Fall, welcher sich sogar durch weitere Optimierungen bei der Umsetzung als bei den für diese Messungen eingesetzten Standardbarrieren noch weiter minimieren lässt (beispielsweise durch den Einsatz von Multicast-Nachrichten oder an bestehende Netzwerkpakete angehängte Huckepack-Informationen).

6. Zusammenfassung und Ausblick

6.1 Zusammenfassung der Arbeit

In dieser Dissertation wurde erstmals der neuartige Ansatz eines transaktionalen Szenengraphens als Grundlage für virtuelle Präsenzsysteme vorgestellt. Es wird hierdurch ein alternatives Konzept geschaffen, mit welchem sich intuitive und vielfältig einsetzbare, verteilte 3D-Umgebungen realisieren lassen, ohne dass hierzu ein Entwickler oder eine Anwendung viel Zeit und Aufwand für Synchronisierungsaufgaben und Konsistenzbetrachtungen aufbringen muss.

Es wurde hierfür einleitend auf das relativ junge Forschungsgebiet der virtuellen Präsenz eingegangen, um die zugehörigen virtuellen Präsenzsysteme genauer zu beschreiben und einzuteilen, aber auch um deren Eigenschaften und Fähigkeiten besser herauszustellen, da sich die zugehörige Terminologie und Inhalte in diesem Bereich noch nicht allgemein etabliert haben. Anhand einer geeigneten Kategorisierung wurde eine Einteilung der für diese Arbeit relevanten virtuellen Präsenz und der eng mit ihr verwandten Web-Präsenz vorgenommen und diese in Verbindung mit einem Überblick zu bestehenden Arbeiten aus beiden Bereichen gesetzt. Dabei wurden die wesentlichen Konzepte und Eigenschaften dieser avatarbasierten virtuellen Umgebungen aufgezeigt und umfassend dargestellt.

Im Anschluss wurde auf die Grundstrukturen von virtuellen Umgebungen eingegangen. Hierzu wurden eingangs die Eigenschaften bestehender Szenengraphensysteme und deren Konzepte sowohl für Einzelplatzsysteme als auch für den verteilten Einsatz detailliert in ihrem Aufbau und ihrer Funktionalität beschrieben und deren bekannteste Vertreter vorgestellt. Die bestehenden Arbeiten aus diesem Bereich basieren dabei nahezu ausschließlich auf nachrichtenbasierten Systemen, welche für gewöhnlich mittels traditioneller Client-Server-Architekturen umgesetzt werden. Diese bieten zwar den Vorteil, dass die gemeinsamen Daten auf einfache Art und Weise konsistent gehalten werden können, da nur die zentrale Serverkomponente direkten Zugriff auf sie hat und auch die Verteilung der Daten nur über sie geregelt wird. Allerdings stellt die zentrale Komponente gleichzeitig eine potentielle Engstelle in den verwendeten Systemen dar, welche deren Skalierbarkeit einschränkt. Um dies auszugleichen werden vorzunehmende Berechnungen auf den gemeinsamen Szenengraphendaten in der Regel vom Server an die Klienten verlagert, wodurch diese Berechnungen dann aber mehrfach bzw. parallel ausgeführt werden und die Mindesthardwareanforderungen der Klienten erhöhen. Zusätzlich ist hierbei ein großer Aufwand erforderlich, um die Daten zwischen den Klienten synchron zu halten, beispielsweise über den Einsatz von Dead-Reckoning-Mechanismen.

Mit der Nutzung eines transaktionalen Szenengraphens wird dahingegen ein gemeinsamer verteilter Speicher zur Kommunikation und zum Datenaustausch zwischen den einzelnen Stationen verwendet und mit diesem ein starkes Konsistenzmodell zur Datenhaltung verknüpft. Durch diese Art der Verteilung besitzen alle teilnehmenden Stationen einen gleichberechtigten Zugriff auf die gemeinsamen Daten, wodurch sich eine Peer-to-Peer-ähnliche Struktur ohne zentrale Komponenten ergibt. Bestehende Arbeiten, welche das Konzept eines gemeinsamen Speichers verfolgen, nutzen stattdessen bislang ausschließlich schwache Konsistenzmodelle, um ein Echtzeitverhalten oder gar die Lauffähigkeit ihrer Systeme zu ermöglichen. Ein transaktionaler Szenengraph bietet darüber hinaus die Vorteile, dass Berechnungen generell nur einmalig vorgenommen werden müssen; alle Stationen können anschließend direkt auf die im Szenengraphen eingebrachten Ergebnisse zurückgreifen und diese bei sich weiter verwenden. Des Weiteren können die von einer Station

benötigten Berechnungen auch stellvertretend von beliebigen anderen Stationen ausgeführt werden, wodurch sich ein dynamischer Lastausgleich realisieren lässt oder sich auch ressourcenbeschränkte oder mobile Geräte einfach in einen transaktionalen Szenengraphen integrieren lassen und somit eine gute Skalierbarkeit für das Gesamtsystem gegeben ist.

Ein weiterer Teil dieser Dissertation beschäftigt sich mit einer möglichen direkten Unterstützung der Strukturen eines transaktionalen Szenengraphens durch die Hardware für einen effizienteren Einsatz. Im Speziellen wurden hierbei Grafikkarten stellvertretend zu allgemeinen Hardwarekomponenten als flexibelste und leistungsstärkste Vertreter betrachtet. Um dabei eine deutlichere Unterscheidung zu den momentanen Strukturen von Grafikkarten zu erreichen, wurden deren Arbeitsweisen und Abläufe genauer untersucht und dargestellt. Da die hierfür nötigen Informationen innerhalb der Grafikkarten aber nicht immer frei verfügbar sind, wurden darüber hinaus auch weitergehende Analyseverfahren vorgestellt, wie diese alternativ zu ermitteln sind.

Hierauf aufbauend wurden dann die für einen transaktionalen Szenengraphen benötigten Strukturen – die Verteilung der Szenengraphendaten einerseits und die transaktionale Konsistenz andererseits – detailliert betrachtet. Es wurden hierzu verschiedene Aspekte aufgezeigt, welche von einer theoretischen Speicherverwaltung, die diese Aufgabe übernehmen soll, berücksichtigt werden müssen. Hierzu gehört die Einbeziehung des unterschiedlichen Adressraums durch den Grafikprozessor im Vergleich zur normalen CPU, welcher sich vom bisherigen Aufbau her nicht direkt für eine Verteilung von in Grafikkarten abgelegten Daten eignet. Weiter existiert für eine Kommunikation mit der Grafikkarte zur Verteilung bzw. zur Anforderung benötigter Grafikdaten bislang kein effizienter Mechanismus, wie er speziell für einen Rückkanal von der Grafikkarte zum restlichen System bzw. zur Speicherverwaltung erforderlich wäre. Auch können die von einer Grafikkarte vorgenommenen Modifikationen an Grafikdaten nicht unmittelbar festgestellt werden, so dass sich Schreibzugriffe kaum überwachen oder abfangen lassen, um dies in einer Verteilung zu berücksichtigen. Nicht zuletzt unterscheidet sich auch die erreichbare Leistungsfähigkeit durch die Einlagerung von Objekten in den Grafikkartenspeicher anstelle des Hauptspeichers deutlich.

Die Bereitstellung der transaktionalen Konsistenz erhöht dabei nochmals die auftretende Komplexität, da bestehende Hardwarekomponenten bislang nicht transaktional arbeiten und infolgedessen auch die hierfür benötigen Strukturen bzw. Transaktionen nicht unterstützen. Dies gilt gleichermaßen für die darauf aufbauenden Abläufe wie Rücksetzungen von Transaktionen oder die Benachrichtigung anderer Stationen über aufgetretene Änderungen.

Bestehende Arbeiten aus diesen Bereichen sind bislang lediglich von experimenteller Natur; die Weiterentwicklung insbesondere von Grafikkartenhardware lässt aber zumindest in kommenden Grafikkartengenerationen ein Einfließen von etlichen der zuvor angesprochenen und benötigten Aspekte in die Hardware erkennen, wodurch sich zukünftig eine hardwareseitige Unterstützung eines transaktionalen Szenengraphens deutlich besser und effizient realisieren lassen wird.

Im Rahmen dieser Arbeit wurde ebenfalls ein Prototyp eines virtuellen Präsenzsystems namens 'World of Wissenheim' entwickelt, welcher auf dem vorgestellten, neuartigen Ansatz basiert. Er bietet dabei die von bestehenden virtuellen Präsenzsystemen typischen Merkmale sowie auch zusätzlich den Einsatz interaktiver Inhalte, mit welchen sich komplexe Sachverhalte aus beliebigen Themenbereichen anschaulich und verständlich präsentieren lassen und so den Aufbau von Lernumgebungen für verschiedenste Zielgruppen und Einsatzzwecke ermöglichen. Darüber hinaus wurde er auch um eigene Fähigkeiten erweitert wie z.B. die Möglichkeit der Integration von Videokonferenzen, für die bislang eigenständige Programme erforderlich waren. Messungen, welche mit dem entwickelten Prototypen durchgeführt wurden, runden diese Dissertation ab und die präsentierten Ergebnisse belegen die Realisierbarkeit des vorgestellten neuartigen Ansatzes.

6.2 Ausblick

Der entwickelte und vorgestellte Ansatz eines transaktionalen Szenengraphens wurde im Rahmen dieser Arbeit bislang nur in der von Szenengraphen gebräuchlichsten Variante eines hierarchischen Transformationsgraphens umgesetzt. Es bieten sich daher verschiedene weitergehende Optimierungsvarianten zur Leistungssteigerung an.

Einerseits kann der transaktionale Szenengraph selbst für eine darauf aufbauende Grafikengine weiter angepasst und verbessert werden (z.B. anhand von Zustandssortierungen), aber auch der weitere Ausbau zu einem modernen Multi-Szenengraphensystem ist mit dem vorgestellten Konzept problemlos möglich. Da insbesondere anhand der eingesetzten transaktionalen Konsistenz die Synchronisierung verteilter Daten ohne zusätzlichen Aufwand von einer Anwendung aus möglich ist, lässt sich dieser Ansatz infolgedessen vergleichsweise einfach zu mehreren spezialisierten und parallel vorgehaltenen Szenengraphen erweitern, die miteinander gekoppelt sind und dadurch für verschiedenste Anwendungsfälle eines darauf aufbauenden virtuellen Präsenzsystems (Rendering, Kollisionserkennung, ...) eine optimale Leistung bieten können.

Andererseits kann der transaktionale Szenengraph auch als Grundlage für weitergehende und differenziertere Arbeitsweisen bei unterschiedlichen Anforderungen innerhalb einer Anwendung dienen. So stellt beispielsweise zwar die transaktionale Konsistenz der Anwendung standardmäßig eine starke Form der Konsistenz dauerhaft zur Verfügung, allerdings wird diese nicht in allen Bereichen einer Anwendung auch immer tatsächlich benötigt. Hier könnte eine Aufteilung in Unterabschnitte mit verschiedenen, nebeneinander koexistierenden Konsistenzmodellen für unterschiedliche Arten der (Grafik-)Daten noch weitere Erkenntnisse und Leistungssteigerungen bieten. Beispielsweise würde es ausreichen, die Aktualisierung von Bildern einer Webkamera oder von Videoströmen innerhalb des transaktionalen Szenengraphens mit einer abgeschwächten Konsistenz durchzuführen, da ein einzelnes verpasstes Bild hierbei kaum wahrgenommen wird bzw. auch keine Auswirkungen auf die Konsistenz des Gesamtsystems hat. Die Struktur des Szenengraphen selbst bliebe dabei aber stark, d.h. transaktional konsistent, so dass z.B. eine korrekte und einheitliche Bewegung von Objekten und Avataren weiterhin gewährleistet wird. Die Einbeziehung schwächerer Konsistenzmodelle in einen von einer starken Konsistenz ausgehenden Szenengraphen ist dabei vergleichsweise einfach möglich. Umgekehrt erfordert die Realisierung eines stärkeren Konsistenzmodells einen beträchtlichen Aufwand, weshalb dies in bestehenden Szenengraphensystemen bislang noch nicht vorgenommen wurde.

Der ebenfalls im Rahmen dieser Arbeit entwickelte und auf einen transaktionalen Szenengraphen aufbauende Prototyp 'World of Wissenheim' wird mittlerweile auch in vielen über diese Arbeit hinausgehenden Bereichen eingesetzt. Inzwischen ist er beispielsweise bereits in zahlreichen Lehrveranstaltungen des Institutes für Verteilte Systeme vertreten. Des Weiteren wird Wissenheim auch zwischenzeitlich auf Grid-Ebene eingesetzt. So dient es innerhalb eines aktuellen EU-Forschungsprojekts als Demonstrator-Anwendung für das dort entwickelte Grid-Computing-Betriebssystem namens 'Xtreem OS' [w57]. Bei XtreemOS handelt es sich um ein Linux-basiertes Betriebssystem, welches die Ressourcen eines Grids ebenso für einen Benutzer bereitstellt und verwaltet, wie ein normales Betriebssystem diejenigen eines einzelnen Rechners. Dieses Forschungsprojekt wird dabei von einem Konsortium aus 19 verschiedenen EU-weiten Partnern durchgeführt, darunter neben der Universität Ulm auch u.a. SAP und EADS. Für diese unterschiedlichen Einsatzgebiete wird Wissenheim kontinuierlich verbessert und weiterentwickelt, so dass auch eine zunehmende und weitergehende Verbreitung dieser Arbeit zu erwarten ist.

Anhang A: Literaturverzeichnis

[1] R. Goeckelmann: "Speicherverwaltung und Bootstrategien für ein Betriebssystem mit transaktionalem verteilten Heap", Dissertation, Universität Ulm, 2006

[2] M. Wende: "Kommunikationsmodell eines verteilten virtuellen Speichers", Dissertation, Universität Ulm, 2003

[3] M. Weber: "Verteilte Systeme", Spektrum Akademischer Verlag GmbH, 1998, ISBN 3-8274-0221-2

[4] R. Buyya, T. Cortes, H. Jin: "Single System Image (SSI)", International Journal of High Performance Computing ApplicationsVolume 15 (2), p. 124-135, Sage Publications Inc., 2001

[5] S. Traub: "Speicherverwaltung und Kollissionsbehandlung in transaktionsbasierten verteilten Betriebssystemen", Dissertation, Universität Ulm, 1996

[6] M. Schöttner: "Persistente Type und Laufzeitstrukturen in einem Betriebssystem mit verteiltem virtuellen Speicher", Dissertation, Universität Ulm, 2002

[7] H. Kung, J. Robinson: "On Optimistic Methods for Concurrency Control", in Journal: ACM Transactions on Database Systems (TODS), Volume 6 (2), p. 213-226, ACM Press, 1981

[8] S. Frenz: "Zuverlässiger verteilter Speicher mit transaktionaler Konsistenz", Dissertation, Universität Ulm, 2006

[9] M. Slater, M. Usoh: "Depth of Presence in Virtual Environments", in Journal: Presence: Teleoperators and Virtual Environments, p.130-144, MIT Press volume 3.2, 1994

[10] M. Lombard, T. Ditton: "At the Heart of It All: The Concept of Presence", in: Journal of Computer Mediated Communication, Volume 3, No 2, 1997

[11] G. Riva, F. Davide, W. IJsselsteijn: "Being There: The experience of presence in mediated environments", Ios Press, Amsterdam, Niederlande, 2003

[12] D. Schmalstieg, A. Fuhrmann, G. Hesina, Z. Szalavari, L. Encarnação, M. Gervautz, W. Purgathofer: "The Studierstube Augmented Reality Project", in: PRESENCE - Teleoperators and Virtual Environments, Vol. 11, No. 1, p. 33-54, MIT Press, 2002

[13] I. Barakonyi, W. Frieb, D. Schmalstieg: "Augmented Reality Videoconferencing for Collaborative Work", in: Proceedings of the 2nd Hungarian Conference on Computer Graphics and Geomtery, Budapest, Hungary, 2003

[14] M. Merz, K. Froitzheim, H. Wolf: "Dynamic Neighborhoods - Browsing the World Wide Web Together", in: Proceedings of SPIE/SYBEN Conference on Interactive Multimedia Services and Equipment, p. 423-432, Zurich, Switzerland, 1998

[15] K. Froitzheim, P. Schubert: "Presence in Communication Spaces", in: Proceedings of the 19th International CODATA Conference, Berlin, Germany, 2004

[16] G. Sidler, A. Scott, H. Wolf: "Collaborative Browsing in the World Wide Web", in: Proceedings of the 8th Joint European Networking Conference, Edinburgh, Scotland, 1997

[17] R. Schroeder: "The Social Life of Avatars - Presence and Interaction in Shared Virtual Environments", Springer Verlag, 2002, ISBN 1-85233-461-4

[18] F. Boos: "Kooperative Manipulation von 3D-Objekten in Wissenheim", Diplomarbeit, Universität Ulm, 2007

[19] E. Akpinar, O. Ergin: "The Effect of Interactive Computer Animations Accompanied with Experiments on Grade 6th Students' Achievements and Attitudes toward Science", in: Proceedings of the 2nd International Conference on Interactive Mobile and Computer Aided Learning IMCL07, Amman, Jordan, 2007

[20] P. Strauss: "IRIS Inventor, A 3D Graphics Toolkit", in: Proceedings of the 8th annual conference on Object-Oriented Programming Systems, Languages, and Applications, p. 192-200, Washington D.C., United States, 1993

[21] J. Wernecke: "The Inventor Mentor: Programming Object Oriented 3D Graphics with Open Inventor, Release 2", Addison Wesley, 1994, ISBN 0-20162-495-8

[22] J. Rohlf, J. Helman: "IRIS Performer: A High Performance Multiprocessing Toolkit for Real-Time 3D Graphics", in: Proceedings of the 21st annual conference on Computer graphics and interactive techniques, p. 381-395, Orlando, Florida, 1994

[23] D. Reiners: "OpenSG: A Scene Graph System for Flexible and Efficient Realtime Rendering for Virtual and Augmented Reality Applications", Dissertation, Technische Universität Darmstadt, 2002

[24] D. Reiners, G. Voß, J. Behr: "OpenSG: Basic Concepts", in: 1. OpenSG Symposium OpenSG 2002, Darmstadt, Deutschland, 2002

[25] G. Hesina, D. Schmalstieg, A. Fuhrmann, W. Purgathofer: "Distributed Open Inventor: A Practical Approach to Distributed 3D Graphics", in: Proceedings of the ACM Symposium on Virtual Reality Software and Technology, p. 74-81, London, UK, 1999

[26] B. MacIntyre, S. Feiner: "Repo-3D - A Distributed 3D Graphics Library", in: Proceedings of the ACM SIGGRAPH '98, p. 361-370 Orlando, Florida, 1998

[27] B. MacIntyre: "Repo: Obliq with Replicated Objects. Programmers Guide and Reference Manual.", Columbia University Computer Science Department Research Report CUCS-023-97, 1997

[28] L. Cardelli: "Obliq - A Language with Distributed Scope", Technical report, Digital Equipment Corporation, Systems Research Center, 1994

[29] M. Naef, E. Lamboray, O. Staadt, M. Gross: "The blue-c Distributed Scene Graph", in: Proceedings of the IPT/EGVE Workshop, p. 125-133, Zurich, Switzerland, 2003

[30] H. Tramberend: "Avocado: A Distributed Virtual Reality Framework", in: Proceedings of IEEE Virtual Reality '99, p. 14-21, Houston, USA, 1999

[31] H. Tramberend: "Avocado: A Distributed Virtual Environment Framework", Dissertation, Universität Bielefeld, 2003

[32] B. Schaeffer, C. Goudeseune: "Syzygy: Native PC Cluster VR", in: Proceedings of the IEEE Virtual Reality 2003, p. 15-22, Los Angeles, USA, 2003

[33] B. Schaeffer: "Networking and Management Frameworks for Cluster-Based Graphics", in: 'Virtual Environment on a PC Cluster Workshop', Protvino, Russia, 2002

[34] E. Frécon, M. Stenius: "DIVE: a scaleable network architecture for distributed virtuel environments", special issue on Distributed Virtual Environments5(3), 91-100 Distributed Systems Engineering Journal, 1998

[35] K. Birman, R. Cooper: "The ISIS Project: Real experience with a fault tolerant programming system", in: Proceedings of the ACM SIGOPS Operating Systems Review, Volume 25 (2), p. 103-107, Bologna, Italy, 1991

[36] B. Schaeffer, P. Brinkmann, G. Francis, C. Goudeseune, J. Crowell, H. Kaczmarski: "Myriad: Scalable VR via Peer-to-Peer Connectivity, PC Clustering, and Transient Inconsistency", in: Proceedings of the ACM symposium on Virtual reality software and technology, p. 68-77, Monterey, USA, 2005

[37] K. Lin: "Dead reckoning and distributed interactive simulation", In Proceedings of the SPIE Conference (AeroSense'95), p. 16-36, Orlando, USA, 1995

[38] W. Cai, F. Lee, L. Chen: "An auto-adaptive dead reckoning algorithm for distributed interactive simulation", In Proceedings of the 13th Workshop on Parallel and Distributed Simulation, p. 82-89, Atlanta, USA, 1999

[39] L. Pantel, L. Wolf: "On the suitability of dead reckoning schemes for games", in Proceedings of the 1st workshop on Network and system support for games, p. 79-84, Braunschweig, Germany, 2002

[40] M. Wloka: "Lag in Multiprocessor Virtual Reality", PRESENCE: Teleoperators and Virtual Environments4(1), p. 50-63, MIT Press, 1995

[41] D. Ball, S. Hoyt: "The Adaptive Time-Warp Concurrency Control Algorithm", In Proceedings of the SCS Multiconference on Distributed Simulation, p. 174-177, Volume 22, 1990

[42] J. Steinmann: "Breathing Time Warp", in: Proceedings of the ACM SIGSIM Simulation Digest, Volume 23, Issue 1, p. 109-118, New York, USA, 1993

[43] L. Gautier, C. Diot: "Design and Evaluation of MiMaze,a Multi-Player Game on the Internet", In Proceedings of the IEEE International Conference on Multimedia Computing and Systems, p. 233-236, Austin, USA, 1998

[44] E. Cronin, B. Filstrup, A. Kurc: "A distributed multiplayer game server system", Technical Report UM EECS589, University of Michigan, 2001

[45] K. Li: "IVY: A shared virtual memory system for parallel computing", in: Proceedings of the 1988 International Conference on Parallel Processing (ICPP'88), volume 2, p. 94-101, St. Charles, USA, 1988

[46] P. Keleher, A. Cox, S. Dwarkadas, W. Zwaenepoel: "TreadMarks: Distributed Shared Memory on Standard Workstations and Operating Systems", in: Proceedings of the Winter 1994 USENIX Conference, San Francisco, USA, 1994

[47] D. Margery, G. Vallée, R. Lottiaux, C. Morin, J. Berthou: "Kerrighed: A SSI Cluster OS Running OpenMP", Research Report RR-4947, 2003

[48] P. Pichler: "Populating virtual worlds: an evolutionary approach", Diplomarbeit, Universität Ulm, 2005

[49] I. Wald, A. Dietrich, C. Benthin, A. Efremov, T. Dahmen, J. Günther, V. Havran, H. Seidel, P. Slusallek: "Applying Ray Tracing for Virtual Reality and Industrial Design", in: Proceedings of the 2006 IEEE Symposium on Interactive Ray Tracing, p. 177-185, Salt Lake City, USA, 2006

[50] S. Nourian, X. Shen, N. Georganas: "XPHEVE: an extensible physics engine for virtual environments", in: Proceedings of the IEEE Canadian Conference on Electrical and Computer Engineering, Ottawa, Canada, 2006

[51] E. Penner, R. Schmidt, S. Carpendale: "A GPU Cluster Without the Clutter: A Drop-in Scalable Programmable-Pipeline with Several GPUs and Only One PC", in Symposium on Interactive 3D Graphics and Games (I3D 2006), Redwood City, USA, 2006

[52] A. Altmayer: "Aufbau und Visualisierung virtueller Welten mit Mehrbenutzerszenarien in einem gemeinsamen virtuellen Speicher", Diplomarbeit, Universität Ulm, 2004

[53] A. Tanenbaum: "Verteilte Betriebssysteme", Prentice-Hall, 1995, ISBN 3-930436-23-X

[54] R. Göckelmann: "2D-/3D-Grafik unter Plurix", Diplomarbeit, Universität Ulm, 2001

[55] M. Fakler: "Direkte Ansteuerung von beschleunigten Grafikfunktionen einer Hochleistungsgrafikkarte am Beispiel der ATI Radeon", Diplomarbeit, Universität Ulm, 2003

[56] A. Weggerle: "Hardwarenahe Analyse und Umsetzung von 3D-Grafikfunktionen insbesondere von Vertex- und Pixelshadern", Diplomarbeit, Universität Ulm, 2007

[57] C. Hall: "Virtual Textures - a true demand-paged texture memory management system in silicon", in: 'Hot3D Session' of the Siggraph Eurographics Workshop, Los Angeles, USA, 1999

[58] A. Moerschell, J. Owens: "Distributed Texture Memory in a Multi-GPU Environment", Computer Graphics Forum, The International Journal of the Eurographics Association, Volume 27 (2), Eurographics Digital Library, 2008

[59] R. Brennan, M. Manzke, K. O'Conor, J. Dingliana, C. O'Sullivan: "A Scalable and Reconfigurable Shared-Memory Graphics Cluster Architecture", in: Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA'07), Las Vegas, USA, 2006

[60] S. Pronovost, H. Moreton, T. Kelley: "Windows Display Driver Model (WDDM) v2 and Beyond", in: Windows Hardware Engineering Conference (WinHEC 2006), Seattle, USA, 2006

[61] J. Larus, R. Rajwar: "Transactional Memory", Morgan & Claypool Publishers, 2007, ISBN 1-59829-124-6

[62] S. Lohrmann: "Programmierung von Hochleistungs-Grafikprozessoren zur Lösung numerischer Problemstellungen", Diplomarbeit, Fachhochschule Münster, 2006

[63] Z. Fan, F. Qiu, A. Kaufman, S. Yoakum-Stover: "GPU Cluster for High Performance Computing", in: Proceedings of the 2004 ACM/IEEE conference on Supercomputing, p. 47-58, Pittsbourgh, USA, 2004

[64] J. Krüger: "A GPU Framework for Interactive Simulation and Rendering of Fluid Effects", Dissertation, Technische Universität München, 2006

[65] D. Luebke, M. Harris, et al.: "GPGPU: General Purpose Computation On Graphics Hardware", in: ACM SIGGRAPH 2004 Course Notes, Los Angeles, USA, 2004

[66] B. Saha, A. Adl-Tabatabai, Q. Jacobson: "Architectural Support for Software Transactional Memory", in: Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture, p. 185-196, Orlando, USA, 2006

[67] D. Dice, M. Herlihy, D. Lea, Y. Lev, V. Luchangco, W. Mesard, M. Moir, K. Moore, D. Nussbaum: "Applications of the Adaptive Transactional Memory Test Platform", in: 3rd ACM SIGPLAN Workshop on Transactional Computing, Salt Lake City, USA, 2008

[68] G. Martinek: "Test und Leistungsanalyse des Clusterbetriebssystems Plurix am Beispiel von Wissenheim", Diplomarbeit, Universität Ulm, 2007

[69] N. Kämmer: "Fairness für verteilte und kollidierende Transaktionen", Diplomarbeit, Universität Ulm, 2006

Anhang B: Webverzeichnis

Alle Links wurden zum Zeitpunkt der Abgabe dieser Dissertation geprüft und waren funktionsfähig.

[w1] Das verteilte Betriebssystem Plurix
http://www.plurix.de/

[w2] Wissenheim-Projektseite
http://www.wissenheim.de/

[w3] At the Heart of It All: The Concept of Presence
http://jcmc.indiana.edu/vol3/issue2/lombard.html

[w4] Presence-Research
http://www.presence-research.org/

[w5] Being There: Concepts, effects and measurement of user presence in synthetic environments
http://www.vepsy.com/communication//volume5.html

[w6] „Studierstube“ – Ein Augmented-Realty-Projekt der Technischen Universität Graz
http://studierstube.icg.tu-graz.ac.at/

[w7] CoBrow – Pleased to meet you on the Web
http://virtual-presence.org/external/www.cobrow.com/pages/components/plainui/

[w8] Lluna, the real web life – The Lluna Virtual Presence Project
http://community.lluna.de/

[w9] Zweitgeist
http://zweitgeist.com/

[w10] Weblin – Triff deine Freunde und neue Leute auf jeder Website!
http://www.weblin.com/

[w11] me.dium – Reveal the hidden worldof people and activity behind your browser
http://me.dium.com/

[w12] Weezu – Chat on any website with the people visiting it
http://weezu.com/

[w13] 3B Browser, Three-B International Limited
http://3b.net/

[w14] Arten von Online-Spielen
http://en.wikipedia.org/wiki/MMO#Types_of_MMOGs

[w15] Übersicht über aktuelle MMORPG-Spiele
http://www.mmorpg-planet.de/

[w16] World of Warcraft, Blizzard Entertainment
http://www.worldofwarcraft.com/

[w17] Pressenotiz Blizzard Entertainment: „World of Warcraft® reaches new milestone: 10 million subscribers“
http://www.blizzard.com/press/080122.shtml

[w18] Informationsportal über und zur Hilfe von Online-Spielesüchtigen
http://www.onlinesucht.de/site2/?page_id=25

[w19] TeamSpeak, TeamSpeak Systems
http://www.teamspeak.com/

[w20] Ventrilo, Flagship Industries
http://www.ventrilo.com/

[w21] Eve Online, CCP Games
http://www.eve-online.com/

[w22] Der Herr der Ringe Online, Turbine Inc.
http://www.lotro.com/

[w23] Half-Life: Counter-Strike, Valve
http://www.counter-strike.net/

[w24] Uru, Cyan Worlds
http://uru.ubi.com/

[w25] Anshe Chung – Erste virtuelle Millionärin in Second Life
http://en.wikipedia.org/wiki/Anshe_Chung

[w26] Virtual Worlds Review
http://www.virtualworldsreview.com/

[w27] Active Worlds, Activeworlds Inc.
http://www.activeworlds.com/

[w28] Second Life, Linden Lab
http://www.secondlife.com/

[w29] There, Makena Technologies Inc.
http://www.there.com/

[w30] Die chinesische Online-Welt HiPiHi
http://www.hipihi.com/index_english.html

[w31] Second Life-Wirtschaftsdaten
http://secondlife.com/whatis/economy_stats.php

[w32] Das Second Life Wochenmagazin AvaStar des Axel-Springer-Verlags
http://www.the-avastar.com/

[w33] Moove Online, moove
http://www.moove.de/
[w34] Entropia Universe, MindArk
http://www.entropiauniverse.com/
[w35] The Croquet Consortium
www.opencroquet.org
[w36] Animationen in Second Life
http://wiki.secondlife.com/wiki/Animation
[w37] Beschreibung des Philosophenproblems auf Wikipedia
http://de.wikipedia.org/wiki/Philosophenproblem
[w38] Beschreibung des geheimen Schlüsselaustauschs nach Shamir auf Wikipedia
http://de.wikipedia.org/wiki/Shamirs_Secret_Sharing
[w39] OpenGL-Spezifikation und Dokumentation
http://www.opengl.org/
[w40] OpenGL-RedBook
http://fly.cc.fer.hr/~unreal/theredbook/
[w41] Entwicklungsverlauf von Szenengraphen
http://www.realityprime.com/scenegraph.php
[w42] VRML97- und X3D-Spezifikation
http://www.web3d.org/x3d/specifications/#vrml97
[w43] The Java 3D Parent Project
https://java3d.dev.java.net/
[w44] OpenSceneGraph
http://www.openscenegraph.org/
[w45] OpenSG
http://www.opensg.org/
[w46] The OpenRT Real-Time Ray-Tracing Project
http://www.openrt.de
[w47] Beschreibung eines CAVE-Systems auf Wikipedia
http://de.wikipedia.org/wiki/Cave_Automatic_Virtual_Environment
[w48] Beschreibung einer Powerwall auf Wikipedia
http://en.wikipedia.org/wiki/Powerwall
[w49] Grafikchipspezifikationen von Intel
http://intellinuxgraphics.com/documentation.html

[w50] Grafikchipspezifikationen von AMD/ATI
http://ati.amd.com/developer/open_gpu_documentation.html
[w51] 3Dlabs Semiconductor Inc.
http://www.3dlabs.com/
[w52] Wikipediaeintrag zu 3Dlabs Semiconductor Inc.
http://en.wikipedia.org/wiki/3Dlabs
[w53] ATI Technologies Inc.
http://www.ati.com/
[w54] nVidia Corporation
http://www.nvidia.com
[w55] Das Tesla-Computersystem von nVidia
http://www.nvidia.de/page/tesla_computing_solutions.html
[w56] Wikipediaeintrag zum Rock-Prozessor von Sun Microsystems
http://en.wikipedia.org/wiki/Rock_processor
[w57] EU-Forschungsprojekt zum Grid-Computing-Betriebssystem XtreemOS
https://www.xtreemos.org

Anhang C: Abbildungsverzeichnis

Anhang D: Bildreferenzen

Alle Abbildungen, Zeichnungen und Bilder sind selbst erstellt, sofern nicht anders angegeben.

Abbildung 3: http://community.lluna.de/
© 2004, bluehands GmbH & Co.mmunication KG. All rights reserved.

Abbildung 4: Beide Bilder:
http://www.weblin.com/
© 2007, Zweitgeist GmbH. All rights reserved.

Abbildung 5: http://3b.net/browser/index.html
© 2005, Three-B International Limited. All rights reserved.

Abbildung 6: Bild links:
http://www.secondlife.com/
© 2007, Linden Research, Inc. All rights reserved.
Bild rechts:
http://www.darwindimensions.com/
© 2006, Darwin Dimensions, Inc. All rights reserved.

Abbildung 7: Beide Bilder:
http://www.blizzard.com/
© 2004-2006 Blizzard Entertainment, Inc. All rights reserved.

Abbildung 8: Beide Bilder:
http://www.lotro-europe.com/
© 2007 Turbine, Inc. All rights reserved.

Abbildung 9: Beide Bilder:
http://www.steamgames.com/
© 2007 Valve Corporation. All rights reserved.

Abbildung 10: Beide Bilder:
http://uru.ubi.com/
© 2004, Cyan Worlds, Inc. Published by Ubisoft Entertainment, S.A. All rights reserved.

Abbildung 12: Bild links:
http://mysocalledsecondlife.com/fun.html
© 2006, Copyright Nan Schultz. All rights reserved.
Bild Mitte:
http://money.cnn.com/popups/2006/autos/second_life/6.html
© Copyright Francis Chung. All rights reserved.

Bild rechts:

http://anya.blogsome.com/category/second-life/

Abbildung 13: Bild links:

http://www.flickr.com/photos/52327305@N00/243950606/

Bild rechts:

http://de.wikipedia.org/wiki/Bild:SL_T_Online_Island_%2828%2C130%2C21%29.jpg

Abbildung 15: Beide Bilder:

http://www.secondlife.com/

Abbildung 16: Alle Bilder:

http://www.secondlife.com/

Abbildung 17: Beide Bilder:

http://www.moove.de/

Abbildung 18: Beide Bilder:

http://www.opencroquet.org/

Anhang E: Veröffentlichungen

M. Fakler, S. Frenz, R. Göckelmann, M. Schöttner, P. Schulthess: „An interactive 3D world built on a transactional operating system“; in: Proceedings of the IEEE Canadian Conference on Electrical and Computer Engineering, Saskatoon, Canada, 2005

M. Fakler, S. Frenz, R. Göckelmann, M. Schöttner, P. Schulthess: „Project Tetropolis - Application of Grid Computing to Interactive Virtual 3D Worlds”; in: Proceedings of the MIPRO Conference on Hypermedia and Grid Systems, Opatija, Croatia, 2005

M. Fakler, S. Frenz, M. Schöttner, P. Schulthess: „A demand-driven approach for a distributed virtual environment“; in: Proceedings of the IEEE Canadian Conference on Electrical and Computer Engineering, Ottawa, Canada, 2006

M. Schöttner, R. Göckelmann, S. Frenz, M. Fakler, P. Schulthess: „Incremental Distributed Garbage Collection using Reverse Reference Tracking”; in: Proceedings of the Euro-Par 2006, Dresden, Germany, 2006

S. Gerhold, M. Schöttner, M. Fakler, M. Sonnenfroh, P. Schulthess: „Smart Restartable Snapshots on top of a Distributed Transactional Memory”; in: Proceedings of the Eurosys 2007, Lisbon, Portugal, 2007

P. Schulthess, M. Fakler, S. Gerhold, M. Sonnenfroh: „'World of Wissenheim' - A Virtual 3D-World for Avatar-based Leisure and Learning”; in: Proceedings of the 2nd International Conference on Interactive Mobile and Computer Aided Learning IMCL'07, Amman, Jordan, 2007

Anhang F: Lebenslauf

Vor- und Zuname: Markus Fakler

Geburtsdatum: 18. Mai 1976

Nationalität: deutsch

Familienstand: ledig

Privatadresse: Lindenstr. 34
D - 89077 Ulm
Tel.: +49 731 56279

Büroadresse: James-Frank-Ring, O-27 / 3209
D - 89069 Ulm
Tel.: +49 731 50 24138
Fax: +49 731 50 24142

E-Mail: markus.fakler@uni-ulm.de

WWW: http://www-vs.informatik.uni-ulm.de/~fakler

Ausbildung:

1982 – 1986:	Grundschule Haslach-Rot, Baden-Württemberg
1986 – 1995:	Gymnasium Ochsenhausen, Baden-Württemberg
27.06.1995:	Abitur
1995 – 1996:	Zivildienst im Jugendhaus Rot a. d. Rot, Baden-Württemberg
1996 – 2003:	Studium der Informatik an der Universität Ulm
30.09.2003:	Diplom Informatik
seit August 2003:	Wissenschaftlicher Mitarbeiter im Institut für Verteilte Systeme, Universität Ulm