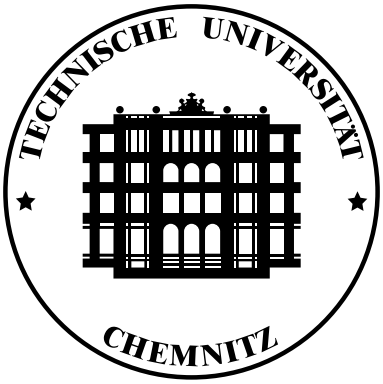
Technische Universität Chemnitz



Fakultät für Informatik

Professur für Graphische Datenverarbeitung und Visualisierung

Studienarbeit

Paralleles, adaptives Level-of-Detail für VR-Simulationen

Verfasser:

Tino Schwarze

16. September 2002

Betreuer:

Dipl.-Inf. M. Lorenz,
Dipl.-Math. H. Wagner

Schwarze, Tino:
Paralleles, adaptives Level-of-Detail für VR-Simulationen
Studienarbeit, Technische Universität Chemnitz, 2002.

Inhaltsverzeichnis

1. Einleitung

Mit zunehmender Leistungsfähigkeit von Standard-Computern und damit einhergehendem Preisverfall wurde in letzter Zeit immer wieder die Idee aufgegriffen, durch Verbindung einer größeren Anzahl von preiswerten Standard-Computern Rechenleistungen zu erzielen, die bisher verhältnismäßig teuren Supercomputern vorbehalten waren. Das sogenannte Cluster-Computing erfreut sich steigender Beliebtheit, was unter anderem durch die Anschaffung eines Clusters an der TU Chemnitz deutlich wurde (Chemnitzer Linux Cluster, kurz CLiC.

In der Computergrafik werden in allen Teildisziplinen nach wie vor hohe Rechenleistungen benötigt. Echtzeit-Rendering wird nur deshalb von anderen Darstellungsverfahren (wie z.B. Raytracing) unterschieden, weil für komplexere Verfahren bisher die entsprechend leistungsfähige Hardware fehlt. Ein großer Teil der Forschung widmet sich der Einsparung von Rechenleistung, um komplexe Anwendungen überhaupt realisierbar zu machen. Viele Algorithmen, die im Kontext von Echtzeit-Rendering und Virtual Reality (VR) angesiedelt sind funktionieren auf der Basis von geschickt vereinfachten Modellen physikalischer Vorgänge.

Aufgrund dieser Umstände liegt es nahe, parallele Ansätze zu verfolgen, da diese eine deutliche Steigerung der verfügbaren Rechenleistung bei vergleichsweise geringem finanziellen und materiellen Aufwand versprechen (man vergleiche etwa die Anschaffungskosten für 500 identische Rechner mit den Entwicklungskosten für ein System mit der 500fachen Leistung eines herkömmlichen Systems[1]).

Ist mehr Rechenleistung nutzbar, macht sich dies sofort in der Reaktionsgeschwindigkeit einer VR-Simulation bemerkbar. Falls die Simulationslatenz

[1] In der Praxis werden 500 identische Rechner nicht die 500fache Leistung erreichen, da parallele Programme im Allgemeinen serielle Abschnitte enthalten und der Aufwand für die Kommunikation im parallelen Programm höher ist.

bereits ausreichend ist, läßt sich die hinzugewonnene Leistung sofort in eine höhere Komplexität der simulierten Szenen investieren, um einen höheren Realismusgrad zu erreichen. Auch für die Erweiterung der Simulation um intuitive Mensch-Maschine-Schnittstellen (z.B. Gesten- oder Spracherkennung) kann zusätzlich nutzbare Rechenleistung verwendet werden. Es ist derzeit noch nicht vorstellbar, wann „ausreichend“ Rechenleistung zur Verfügung stehen wird, um auch höchste Ansprüche befriedigen zu können (Stichworte: Echtzeit-Raytracing, Simulation hochkomplexer physikalischer Prozesse in Echtzeit).

Ziel dieser Studienarbeit ist es, einen neuen Ansatz zur Parallelisierung von Echtzeit-Rendering zu untersuchen. Dabei werden zwei Hauptziele verfolgt:

1. Feststellung, ob das Verfahren prinzipiell auf dem CLiC realisierbar ist und Implementierung eines Prototyps
2. Theoretische Betrachtungen des neuen Verfahrens:
   - Grundlegende Zusammenhänge und Abläufe
   - Falls nicht realisierbar: Warum?
   - Falls realisierbar: Verbesserungsmöglichkeiten, Stärken und Schwächen

Im folgenden soll zunächst der offensichtlichste Ansatz für paralleles Rendering betrachtet werden um die Motivation für ein neues Verfahren zu klären.

1.1. Einfacher Ansatz für paralleles Rendering

Ein einfacher Ansatz für die Parallelisierung von Rendering ist die Aufteilung des Bildes in Teilbilder, die jeweils von einem einzelnen Knoten gerendert werden. Für Raytracing und die Animation von Filmen ist diese Methode prädestiniert, im günstigsten Fall können bereits vorhandene Arbeitsplatzcomputer eingesetzt werden.

Die Ergebnisse des Rendering müssen jedoch zum Ausgabegerät übertragen werden. Besonders für Großbildprojektionen, wie sie z.B. in CAVEs benötigt werden, sind hohe Auflösungen notwendig. Bei einer Auflösung von $1280 * 1024$ Pixeln, einer Farbtiefe von 24 Bit und einer Bildwiederholrate von 25 Hz (im Allgemeinen als die untere Grenze für eine flüssige Simulation angesehen) sind

$1280 * 1024 * 3 * 25 = 98304000$ Byte (ca. 93 Megabyte[2]) pro Sekunde an das Ausgabegerät zu übertragen. Diese Datenmenge ist von gängigen PCs kaum zu bewältigen (der PCI-Bus transportiert maximal 133 MB/s, in der Praxis bleiben davon nur ca. 80-100 MB/s übrig). Auch die Netzwerkhardware wäre entsprechend kostspielig – derzeit können nur Gigabit-Ethernet oder Spezialnetzwerke wie Myrinet diese Bandbreiten bereitstellen.

Mit dem Wunsch nach stereoskopischer Darstellung verdoppeln sich die Bandbreitenanforderungen und sprengen damit sofort die Möglichkeiten von Standard-Komponenten. Der Einsatz eines „Standard-Clusters“ wird unmöglich und Speziallösungen müssen verwendet werden, die entsprechend teuer sind. Weiterhin werden 25 Hz Bildwiederholrate als untere Grenze betrachtet – von einer flüssigen Simulation spricht man erst ab ca. 50 Hz, was mit einer weiteren Verdoppelung der notwendigen Bandbreiten auf 375 MB/s einhergehen würde. Durch Cluster sollen aber gerade teure Speziallösungen vermieden werden.

Ein weiteres Argument gegen diesen Ansatz ergibt sich aus der Beobachtung, daß dank leistungsfähiger Rendering-Hardware der Zeitaufwand für die Darstellung einer Szene stärker von der Menge der Geometriedaten (Punkte, Polygone) als von der Bildschirmauflösung abhängt. Bei einer einfachen Unterteilung des Bildes wird jedoch die Szenenkomplexität nicht verringert.

Das Verfahren der Unterteilung des Bildes mit verteiltem Rendering und Zusammenführen zum Anzeigegerät ist also für „Standard-Cluster“ ungeeignet.

1.2. Ein neuer Ansatz: Verteilte Reduktion der Geometriedaten

Es muß also eine andere Möglichkeit gefunden werden, um die Last des Rendering zu verteilen. Man kann beobachten, daß der Rechenaufwand vorwiegend proportional zur Anzahl der Polygone bzw. Punkte in der darzustellenden Szene ist. In den letzten Jahren wurden viele Verfahren entwickelt um die Anzahl der zu verarbeitenden Punkte und Polygone zu reduzieren. Level-of-Detail (LOD) ist vermutlich am bekanntesten und am weitesten verbreitet.

Hier wird die Idee verfolgt, daß das gesamte Rendering nach wie vor auf einem einzelnen Knoten (im Folgenden Renderknoten genannt) stattfindet, dieser jedoch vereinfachte Geometrien verwendet und somit Rechenzeit spart.

[2]Die Präfixe „Kilo“, „Mega“ usw. werden in dieser Arbeit mit der Basis 2 verwendet, d. h. 1 Kilobyte = 1024 Byte etc.

Die Vereinfachung der Geometrien soll Gegenstand der Parallelisierung sein wie schematisch in Abbildung 1.1 dargestellt.

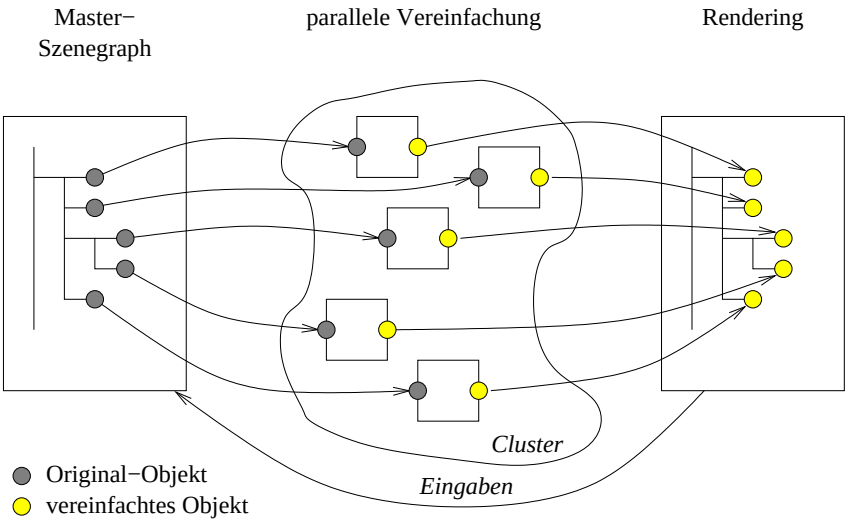


Abbildung 1.1.: Grobentwurf der Systemstruktur

Welche Verfahren zur Vereinfachung der Geometrien verwendet werden ist dabei zunächst nicht von primärem Interesse, muß aber beim Entwurf der Schnittstellen zur Vereinfachungskomponente berücksichtigt werden. Bei entsprechend weitsichtigem Entwurf kann man die Vereinfachung als Black Box betrachten, die Geometriedaten als Eingabe erhält und wieder Geometriedaten ausgibt.

2. Rahmenbedingungen

Im Folgenden sollen zunächst die praktischen Voraussetzungen erörtert und danach einige theoretische Betrachtungen zum zu entwerfenden System durchgeführt werden.

2.1. Praktische Voraussetzungen

Im Rahmen der praktischen Umsetzung eines Prototyps sind die Parameter der vorhandenen Hardware maßgebend. Der CLiC verfügt über 528 Knoten mit jeweils einem Pentium-III / 800 MHz und 512 MB Arbeitsspeicher. Jeder Knoten ist mit jeweils einer Fast-Ethernet-Netzwerkkarte an das Kommunikationsnetzwerk und das Servicenetzwerk angeschlossen.

Das Kommunikationsnetzwerk dient ausschließlich der Kommunikation unter den Knoten. Es ist kompressionsfrei, d.h. alle Knoten können mit voller Bandbreite (100 MBit/s) gleichzeitig senden und empfangen ohne das es zu Kollisionen im Ethernet kommt.

Das Servicenetzwerk dient dem Zugang und der Überwachung der Knoten. Weiterhin wird das verteilte Dateisystem AFS über dieses Netzwerk zur Verfügung gestellt.

Softwareseitig kommt als Betriebssystem RedHat Linux zum Einsatz. Für die Kommunikation im Netzwerk stehen explizit diverse MPI-Implementationen[1] sowie PVM (Parallel Virtual Machine) zur Verfügung. Implizit, da standardmäßig auf jedem Linux-System vorhanden, sind TCP, UDP und RPC nutzbar. Weitere Kommunikationmechanismen wie z.B. CORBA können jederzeit nachgerüstet werden (problematisch sind höchstens Systeme, die Kernelmodifikationen oder lokale Installation auf jedem Knoten erfordern).

[1] zur Zeit sind dies LAM-MPI (http://www.lam-mpi.org/) und MPIch (http://www-unix.mcs.anl.gov/mpi/mpich/)

Das Rendering selbst wird außerhalb des CLiC stattfinden, da dieser im Serverraum des Rechenzentrums installiert und somit nicht physisch zugänglich ist. Außerdem ist weder leistungsfähige 3D-Hardware noch VR-Peripherie vorhanden. Die Powerwall-Installation im VR-Labor der Professur für graphische Datenverarbeitung und Visualisierung scheint prädestiniert für das Rendering der reduzierten Szene.

2.2. Theoretische Betrachtungen

Wie in Abschnitt 1.1 bereits dargelegt wurde, ist die verfügbare Netzwerkbandbreite sowohl ein Grund für diese Arbeit als auch ein beschränkendes Kriterium. Die Beschleunigung von VR-Simulationen steht im Mittelpunkt der Arbeit, deshalb wird außerdem die Latenz im Netzwerk eine entscheidende Rolle spielen, da eine möglichst hohe Bildwiederholrate und somit eine möglichst geringe Latenz im Datenverkehr wünschenswert ist.

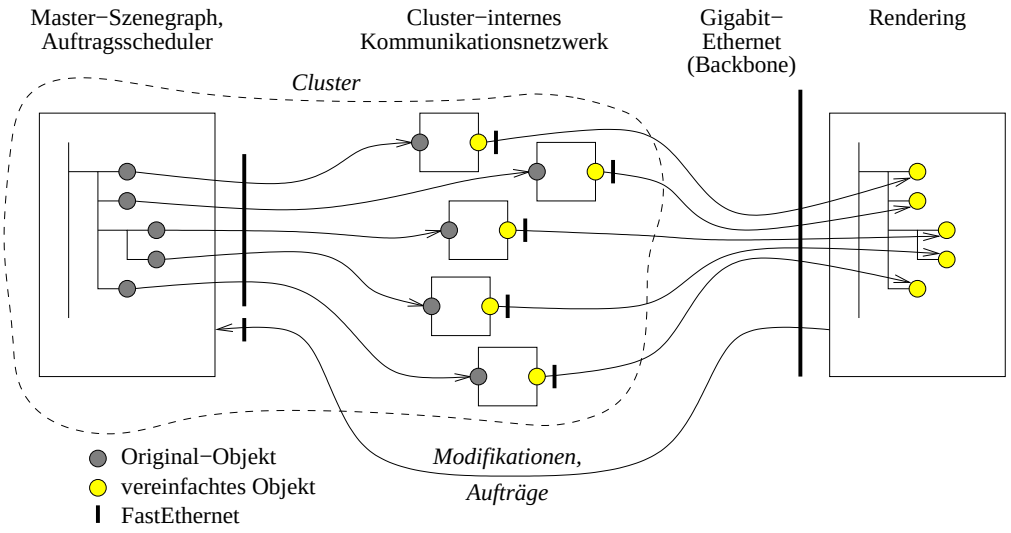


Abbildung 2.1.: Flaschenhälse im Netzwerk des Prototypen

Um eine Abschätzung für die Gesamtleistung des Systems zu erhalten, betrachten wir zum einen die maximal nutzbare Bandbreite und zum anderen die minimale Latenzzeit für die Bearbeitung eines Reduktionsauftrages. Mit diesen beiden Größen ist dann eine optimistische Abschätzung des vertretbaren Kommunikationsaufkommens möglich. Die Maximalbandbreite wird bestimmt

durch den „engsten Flaschenhals“, d.h. die Kommunikationsverbindung mit der geringsten Bandbreite. Wie aus Abbildung 2.1 ersichtlich, ist dies der Fast-Ethernet-Anschluß des Renderknotens:

$$B_{\max} = 100 \text{ MBit/s}$$

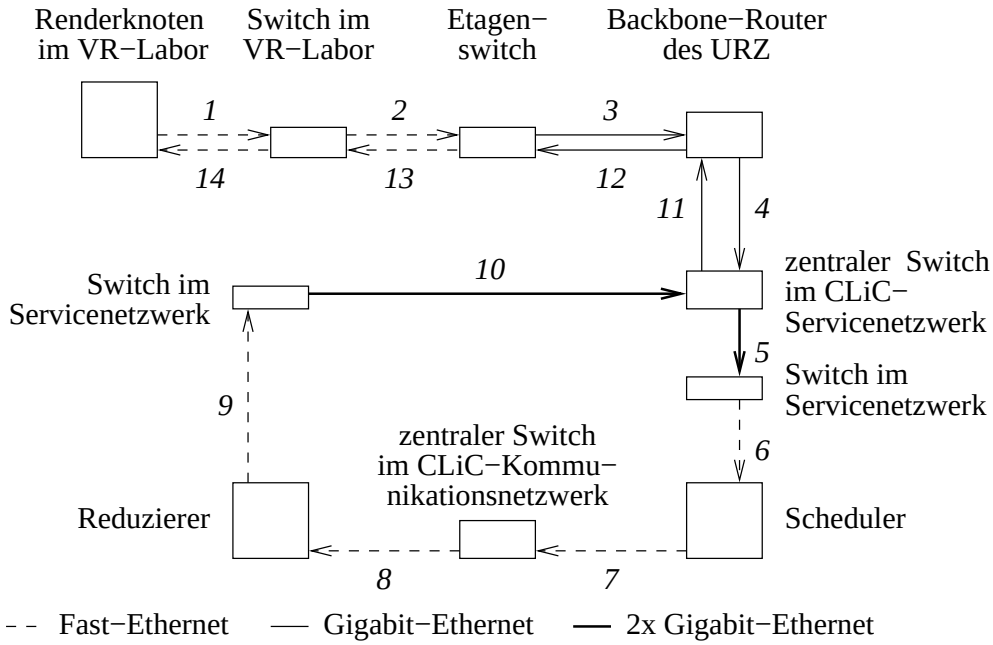


Abbildung 2.2.: Der Weg eines Reduktionsauftrages im Netzwerk

Für die Minimallatenz ist die Netzwerkstruktur und insbesondere die Anzahl der an der Kommunikation beteiligten aktiven Komponenten ausschlaggebend, wie in Abbildung 2.2 dargestellt. Die Round-Trip-Time (ermittelt mit `ping`) vom Renderknoten zum Scheduler (Strecken „1-2-3-4-5-6“ und zurück) beträgt ca. 180 μs. Die Round-Trip-Time zwischen zwei Knoten im Kommunikationsnetzwerk des CLiC (Strecken „7-8“ und zurück) beträgt ca. 80 μs. Die Latenzen $l_{1,6}$, $l_{7,8}$ und $l_{9,14}$ betragen also 90, 40 und 90 μs. Daraus ergibt sich für die Minimallatenz eines Auftrages:

$$l_{\min} = l_{1,6} + l_{7,8} + l_{9,14} = 260\mu\text{s}$$

Leider läuft die Kommunikation teilweise über „öffentliche“ Netzsegmente, so daß sowohl die Latenzzeiten als auch die Bandbreite je nach Auslastung

des Universitätsbackbones bzw. des CLiC-Servicenetzwerkes stark schwanken können. Solche Einflüße sollen an dieser Stelle vernachlässigt werden – das Ziel ist eine Abschätzung des maximal vertretbaren Kommunikationsaufkommens für das verteilte Reduzieren von VR-Objekten.

Vorerst werden außerdem die für die Berechnung der Objektreduktion sowie für das Rendering notwendigen Zeiten vernachlässigt. Für die Verarbeitung eines Reduktionsauftrages vom Zeitpunkt des Versands bis zum Eintreffen des Ergebnisses sind also mindestens 260 μs anzusetzen.

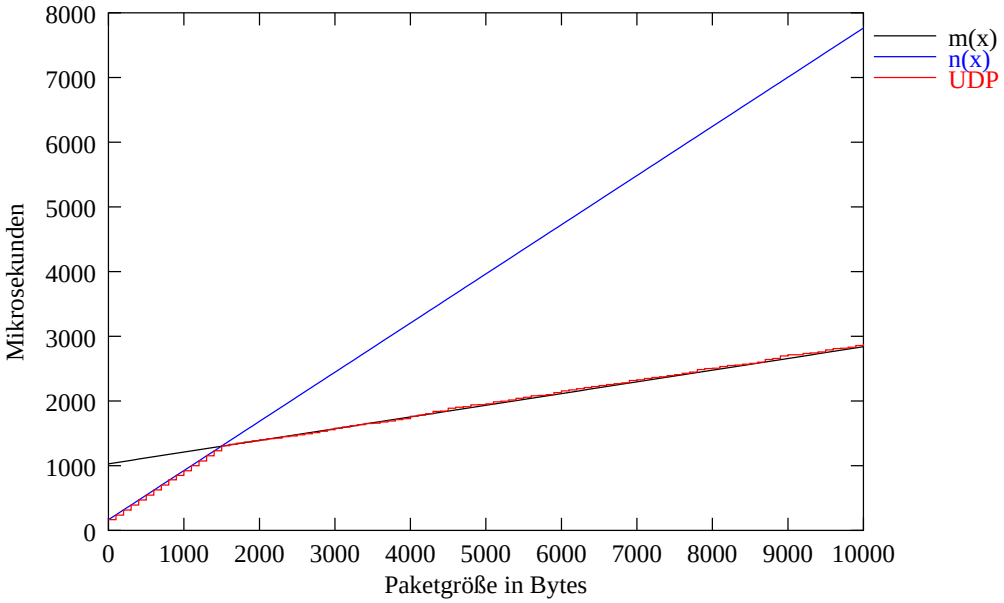


Abbildung 2.3.: Round-Trip-Time zwischen VR-Labor und CLiC-Knoten in Abhängigkeit von der Paketgröße. $m(x) = a_m x + b_m$ beschreibt den Zusammenhang zwischen Paketgröße und Latenz von Paketen > 1500 Byte. $n(x) = a_n x + b_n$ beschreibt die Abhängigkeit für Pakete ≤ 1500 Byte. Mittels Fitting wurde $a_m = 0,180876$, $b_m = 1029,43$, $a_n = 0,759981$ und $b_n = 164,737$ bestimmt.

Um die maximal vertretbare Menge von Daten abzuschätzen, die für ein Bild transportiert werden können, kann man sich ebenfalls auf die Betrachtung der Latenz beschränken, da Latenz und Bandbreite invers proportional zueinander sind. In Abbildung 2.3 ist zu sehen, daß die Latenz ab einem bestimmten Punkt

(der kleinsten MTU[2] auf dem Transportweg) linear mit der Paketgröße steigt. Nach dieser Beobachtung wurde mit `gnuplot` eine Kurvenanpassung vorgenommen. `gnuplot` verwendet dazu den „nonlinear least-squares Marquardt-Levenberg“ Algorithmus[10], der Iterationsabbruch erfolgte, wenn die Summe der Fehler um nicht mehr als 10^{-5} verändert wurde. Basis für die Kurvenanpassung waren die Mittelwerte der Latenzen von 10.000 Tests pro Paketgröße gewichtet nach der Standardabweichung.

Damit ergibt sich:

$$l_{1,6}(x) = l_{9,14}(x) = \begin{cases} a_n x + b_n & , x \leq 1500 \\ a_m x + b_m & , x > 1500 \end{cases} \quad \left[\frac{\mu\text{s}}{\text{Byte}}\right] \tag{2.1}$$

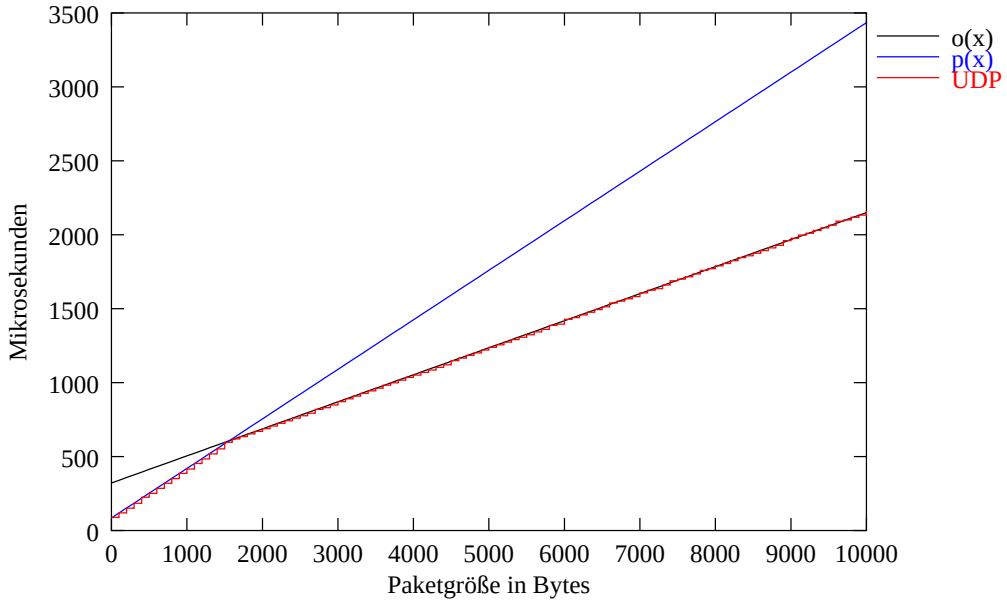


Abbildung 2.4.: Round-Trip-Time zwischen zwei CLiC-Knoten in Abhängigkeit von der Paketgröße. $o(x) = a_o x + b_o$ beschreibt den Zusammenhang zwischen Paketgröße und Latenz von Paketen > 1500 Byte, $p(x) = a_p x + b_p$ beschreibt die Abhängigkeit für Pakete ≤ 1500 Byte. Mittels Fitting wurde $a_o = 0,182849$, $b_o = 321,685$, $a_p = 0,334969$ und $b_p = 84,8851$ bestimmt.

[2]Maximum Transfer Unit, d. h. maximale Paketgröße, hier 1500 Byte. Größere Datenmengen werden automatisch in mehrere Pakete aufgeteilt.

In Abbildung 2.4 sind die entsprechenden Messwerte und angepaßten Kurven für die Kommunikation zwischen zwei Knoten im CLiC dargestellt. Man erhält somit:

$$l_{7,8}(x) = \begin{cases} a_p x + b_p & , x \leq 1500 \\ a_o x + b_o & , x > 1500 \end{cases} \quad \left[\frac{\mu s}{\text{Byte}}\right] \tag{2.2}$$

Aus 2.1 und 2.2 erhält man für die Gesamtlatenz eines Paketes (ohne Beachtung von etwaigen Verzögerungen für Berechnungen oder Modellierung der Tatsache dass durch die Reduktion größere oder kleinere Pakete entstehen können):

$$\begin{aligned} l_{1,14}(x) &= \frac{l_{1,6}(x) + l_{7,8}(x) + l_{9,14}(x)}{2} \\ &= \begin{cases} \left(a_n + \frac{1}{2}a_p\right) x + b_n + \frac{1}{2}b_p & , x \leq 1500 \\ \left(a_m + \frac{1}{2}a_o\right) x + b_m + \frac{1}{2}b_o & , x > 1500 \end{cases} \\ &\approx \begin{cases} 0,93x + 207,18 & , x \leq 1500 \\ 0,27x + 1190,27 & , x > 1500 \end{cases} \quad \left[\frac{\mu s}{\text{Byte}}\right] \end{aligned} \tag{2.3}$$

Diese Abschätzung basiert allerdings auf dem verwendeten Ping-Pong-Benchmark und modelliert die realen Gegebenheiten nur schlecht. So werden zum Beispiel die Reduktionsaufträge sehr kleine und die reduzierten Objekte verhältnismäßig große Pakete produzieren. Auch werden die einzelnen Strecken im Netzwerk unterschiedlich stark belastet.

Das Datenaufkommen für das Rendering eines Bildes läßt sich wie folgt unterteilen und abschätzen (bei s Reduzierern):

1. Versand von t Transformationsmatrizen mit zugehöriger Objekt-ID von Render-Knoten zu Scheduler:

   $n_1 \geq t * (4 * 16 + 4) \Rightarrow n_1 \geq t * 68$, da eine Transformationsmatrix aus 16 Einträgen zu 4 Byte (IEEE 754 Fließkommazahlen einfacher Genauigkeit, entsprechend dem C++-Typ float) besteht und 4 Byte Objekt-ID hinzukommen.

2. Versand von r Reduktionsaufträgen bestehend aus Objekt-ID von Render-Knoten zu Scheduler:

   $n_2 \geq r * 4$

3. Versand von $t * s$ Transformationsmatrizen von Scheduler zu Reduzierer

   $n_3 \geq s * n_1$

4. Versand von r Reduktionsaufträgen von Scheduler zu Reduzierer:

   $n_4 = n_2$

5. Versand von r Reduktionsergebnissen von Reduzierern zu Render-Knoten:

   n_5 läßt sich nur sehr schwer abschätzen, da über die reduzierten Objekte keine Details bekannt sind. Im Falle der Reduktion auf die Bounding-Box kann man $n_4 \geq r * 400$ ansetzen (Objekt bestehend aus 8 Punkten, 6 Polygonen, Transformation und ID sowie expliziter Kodierung von Punkt-, Polygon- und Kindanzahl)

Letztendlich dürfte das begrenzende Kriterium der Umfang der reduzierten Objekte sein. Gerade dies läßt sich an dieser Stelle aber kaum abschätzen, da die Art der Objektreduktion nach wie vor offen gelassen wird.

3. Entwurf und Realisierung

3.1. Programmiersprache

Der Prototyp könnte potentiell mit verschiedensten Sprachen wie z. B. Python, C oder C++ entwickelt werden. Es wird C++ verwendet, da

- es eine Programmiersprache mit hoher Typsicherheit ist
- es eine hohe Effizienz erreichen kann (z.B. durch die Verwendung von Templates)
- bereits Module für das Laden von VR-Szenen existieren.

3.2. Kommunikation im Netzwerk

Die Kommunikation im Netzwerk ist grundlegender Bestandteil des zu entwickelnden Systems, deshalb kommt der Wahl der Kommunikationstechnik große Bedeutung zu.

Um den Entwicklungsaufwand zu reduzieren und nicht „das Rad neu zu erfinden“ (und anderer Leute Fehler zu wiederholen), soll für die Kommunikation ein bestehendes und erprobtes Verfahren verwendet werden. Low-Level-Mechanismen wie TCP, UDP etc. kommen deshalb von Anfang an nicht in Betracht.

Die Anforderungen an das Kommunikationssystem sind:

- hohe Effizienz, d.h. bestmögliche Nutzung der verfügbaren Bandbreite und geringe Latenzzeiten

- High-Level-Interface, d.h. Typsicherheit (keine Typecasts im Programm[1])
- geringer Aufwand, um Datenstrukturen versandfähig zu machen, d.h. optimalerweise können Datenstrukturen sicher versandt werden ohne spezielle Funktionen zu schreiben, die die Struktur serialisieren und deserialisieren
- Unterstützung asynchrone Kommunikation

Um die Effizienz eines Kommunikationssystems beurteilen zu können, wurde ein Benchmark durchgeführt. Der Benchmark bestand im Versand eines Arrays von Zeichen an den Server, welcher eine Kopie des Arrays als Rückgabewert lieferte. Gemessen wurde die Zeit, die vom Absetzen des Aufrufs bis zum vollständigen Erhalt des Resultats verging. Dieser Vorgang wurde insgesamt 10 000 mal wiederholt; außerdem wurden verschiedene Arraygrößen zwischen 1 und 10000 Zeichen vermessen. Danach wurden die mininmale, maximale und durchschnittliche Latenzzeit ermittelt.

Aus der durchschnittlichen Latenzzeit läßt sich mit Hilfe der Paketgröße die tatsächlich erreichte Bandbreite errechnen. Zu bedenken ist dabei, daß die Kommunikation synchron stattfand, somit also entsprechend HalfDuplex maximal 100 MBit/s Durchsatz zu erzielen war.

3.2.1. RPC

RPC ist ein verhältnismäßig alter Standard[2] für die Kommunikation von verteilten Anwendungen. RPC ist weit verbreitet, denn es ist unter anderem die Basis für das NFS-Dateisystem und NIS.

Aufgrund seines Alters ist es jedoch rein prozedural und nicht objektorientiert. Die Daten werden mittels External Data Representation (XDR)[3] kodiert und dann im Netzwerk übertragen. Man kann für eigene Datenstrukturen recht einfach eigene Kodier- und Dekodierfunktionen schreiben. Alternativ kann das Programm `rpcgen`, welches für die Generierung der Standardkomponenten eines RPC-fähigen Programms benutzt wird, die (De-)Kodierroutinen erzeugen.

[1] Natürlich sind Typecasts notwendig insofern die Programmiersprache keine Mechanismen zur (De-)Serialisierung bietet. Typecasts sollen aber innerhalb des Kommunikationssystems, z.B. in automatisch generierten Deserialisierungsfunktionen, erfolgen und nicht im eigentlichen Programm.

[2] RPC Version 2 wird in RFC 1057, Juni 1988[4] spezifiziert.

[3] XDR wird spezifiziert in RFC 1014, Juni 1987[3].

In jedem Fall bedeutet die Verwendung von RPC einen Wechsel von objektorientierter zu prozeduraler Programmierung. Man kann nicht ohne Weiteres eine Methode einer Klasse auf einem anderen Rechner aufrufen.

RPC unterstützt asynchrone Kommunikation (im RPC Kontext Batching genannt.) RPC fiel bei den Benchmarks durch drastische Performanzeinbrüche auf. Mehr dazu später.

3.2.2. MPI

MPI ist ein Standard zur Kommunikation in verteilten Applikationen[4]. Es existieren mehrere freie verfügbare Implementationen. MPI wurde gezielt für den Einsatz in massiv parallelen Systemen entwickelt und optimiert und ist *der* Standard für verteiltes wissenschaftliches Rechnen.

MPI spezifiziert vielfältige Kommunikationsoperationen (Senden, Empfangen, Broadcasts, Scatter, Gather; sowohl synchron als auch asynchron). Weiterhin können mit MPI virtuelle Topologien erzeugt werden.

MPI erreicht eine hervorragende Performanz im Netzwerk - bis zu 70 MBit/s im Benchmark. Es liegt damit knapp unter UDP welches nur zu Referenzzwecken aufgeführt ist.

Der erste Nachteil von MPI bis Version 1.2 ist, daß man beim Start der Applikation mittels implementationsspezifischer Mechanismen die Verteilung der Prozesse auf die Knoten festlegt. Eine Rekonfiguration zur Laufzeit (z.B. Hinzufügen oder Entfernen von Knoten) ist nicht möglich. Diesen Nachteil beseitigt MPI 2.0, welches jedoch noch nicht von allen Implementationen komplett unterstützt wird.

Der zweite Nachteil von MPI ist, daß man nicht direkt Funktionen auf anderen Knoten aufrufen kann. Man muß also den RPC-Mechanismus nachbauen und auf jedem Knoten muß ein Dispatcher laufen, der eine spezielle Nachricht auswertet und entsprechend verzweigt. Der Aufrufer wiederum muß explizit diese Nachricht absenden bevor die Daten für die zu rufende Funktion versendet werden.

Weiterhin muß man für jeden zu versendenden Datentyp manuell einen MPI-Datentyp erzeugen, will man MPI nicht als Transportmedium für Binärdaten mißbrauchen, die vom Empfänger deserialisiert werden. Eine hochsprachliche Verwendung im Sinne von

[4]Die Standardisierung begann im Januar 1993, am 5. Mai 1994 wurde die Version 1.0 verabschiedet, im Juni 1994 die Version 1.1, aktuell ist Version 2.0.

```
remote_object->PerformAction (complex_object);
```

ist somit nicht ohne beträchtlichen Mehraufwand realisierbar.

3.2.3. CORBA

Auch CORBA ist ein Standard zur Kommunikation von verteilten Anwendungen. Im Unterschied zu MPI und RPC wird hier das objektorientierte Programmiermodell vorausgesetzt. Deshalb spricht man auch weniger von verteilten Anwendungen als vielmehr (und korrekter) von verteilten Objekten.

Urheber von CORBA ist die OMG (Object Management Group). Der CORBA-Standard ist sehr umfangreich. Es wird nicht nur das Kommunikationsprotokoll sondern bis ins Detail das API spezifiziert. Dazu kommen verschiedene Dienste (z.B. Namensauflösung). Mein persönlicher Eindruck ist, daß die Spezifikationen gut durchdacht und sehr realitätsbezogen sind. Versucht man, selbst eine verteilte Applikation zu entwickeln, kommt man früher oder später nahezu zwangsläufig zu Funktionalitäten, die CORBA spezifiziert.

Es existieren mehrere, unterschiedlich vollständige, freie Implementationen von CORBA[5]. Die Implementation „TAO“ ist davon vermutlich am vollständigsten.

CORBA wird oft als langsam angesehen. In meinen Messungen kann ich dies nicht bestätigen. Es erreicht über 80 % der Geschwindigkeit von MPI, wobei letzteres explizit auf schnelle Kommunikation optimiert ist, während CORBA Geschwindigkeit nicht als primäres Ziel verfolgt.

3.2.4. Benchmarkergebnisse

Zu Referenzzwecken wurde zu o.g. Transportverfahren UDP hinzugenommen, da es potentiell den geringsten Overhead hat. Grundsätzlich kommt UDP nicht in Frage, da es einerseits keine Integritätssicherung beinhaltet und man andererseits sämtliche Aspekte der verteilten Applikation selbst implementieren müßte – UDP ist ein reines Transportprotokoll für Binärdaten.

In Abbildung 3.1 sind die Ergebnisse der Messungen für Kommunikation innerhalb eines Rechners dargestellt. Die extremen Einbrüche bei RPC lassen sich auf Eigenheiten des TCP-Protokolls zurückführen, den sogenannten „Nagle Algorithmus“ im Zusammenspiel mit „delayed ACK“. Der Zweck der genannten Algorithmen ist es, die Anzahl kleiner Pakete im Netzwerk zu reduzieren. In

[5] siehe auch http://patriot.net/~tvalesky/freecorba.html

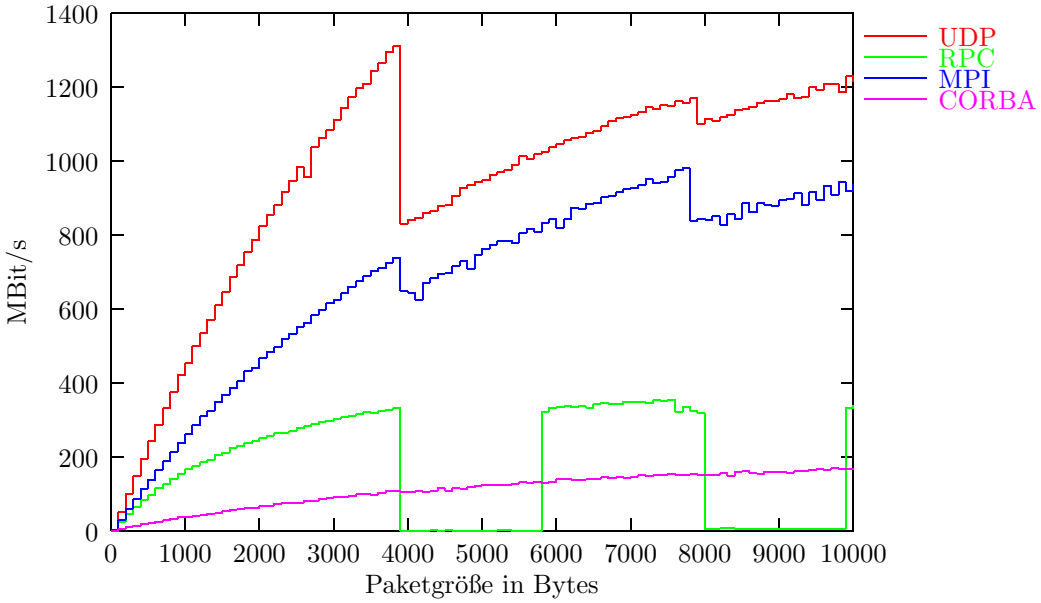


Abbildung 3.1.: Lokal erreichter Durchsatz in Abhängigkeit von der Paketgröße und dem verwendeten Transportverfahren.

diesem Fall wird die Datenübertragung massiv ausgebremst, da der Sender „noch ein wenig“ wartet, bevor er ein kleines Paket mit einem ACK versendet, in der Hoffnung, dass von der Anwendung noch ein paar Daten kommen. Der Einbruch wird durch den verwendeten Ping-Pong-Benchmark besonders deutlich. Prinzipiell könnte der „Nagle Algorithmus“ deaktiviert werden, dazu müßte man sich allerdings auf eine sehr niedrige Ebene der RPC-Bibliotheken begeben und diese unter Umständen sogar neu kompilieren.

Das schlechte Abschneiden von CORBA beim lokalen Durchsatz (über das Loopback-Interface) ist auf den höheren Aufwand fuer die (De-)Serialisierung zurückzuführen. MPI hat hier einen Vorteil, da es explizit Pakete unterstützt, die eine Vielzahl von Werten des gleichen Datentyps enthalten. RPC wiederum ist nicht so komplex wie eine CORBA-Implementierung.

Der niedrigere Durchsatz von CORBA ist allerdings tolerierbar, da er knapp unter der Maximalbandbreite von Fast-Ethernet liegt. Somit stellt er praktisch keine all zu große Beschränkung dar.

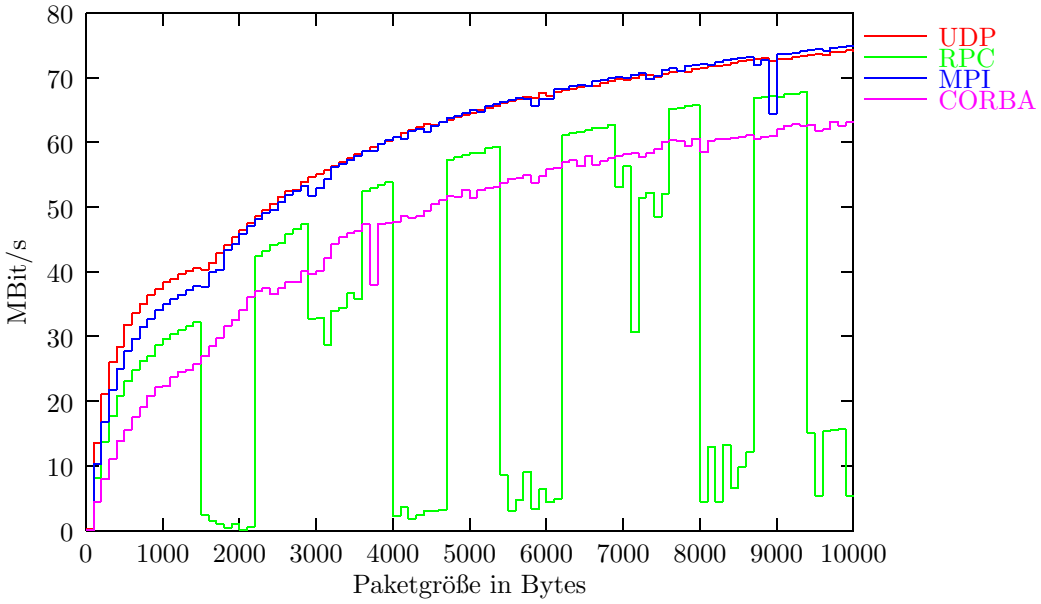


Abbildung 3.2.: Im Netz erreichter Durchsatz in Abhängigkeit von der Paketgröße und dem verwendeten Transportverfahren.

3.2.5. Entscheidung

Ich wähle für die Prototyp-Implementation CORBA, weil es meinen Anforderungen am besten genügt: Es bietet ein objektorientiertes High-Level-Interface, integriert sich nahtlos in C++ und erreicht dazu noch einen guten Durchsatz. RPC disqualifiziert sich durch die extremen Performanzeinbrüche. MPI als schnellstes Transportverfahren wird allein deshalb nicht gewählt, weil sich der Programmieraufwand stark erhöhen würde (eigene (De-)Serialisierungsinfrastruktur sowie eigene Implementierung von RPC-ähnlichen Funktionalitäten wären notwendig).

3.3. Systemarchitektur

Nach der Auswertung der Benchmarks wurde ein Prototyp implementiert. Der Prototyp fungiert als Plugin für den im Rahmen meiner Tätigkeit als studentische Hilfskraft entwickelten CADaVR-Viewer[6]. Die Integration in den CADaVR-

[6] CADaVR ist eine Schnittstelle zwischen CAD-Daten und VR-Systemen, die am Lehrstuhl Computergrafik der TU Chemnitz entwickelt wurde. Siehe auch [11].

Viewer ermöglicht den Zugriff auf eine umfangreiche Auswahl an VR-Szenen. Die Prototyp-Implementation diente gleichzeitig als Motivation und Testfall für die Entwicklung eines erweiterbaren Viewer-Kerns (SceneGraph genannt) für den Viewer.

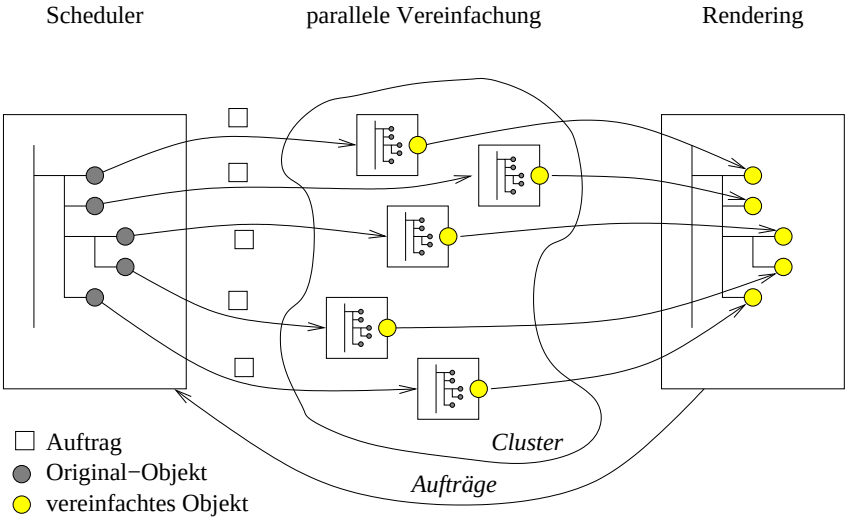


Abbildung 3.3.: Verbesserte Systemarchitektur

Während der Implementation des Prototypen ergab sich eine prinzipielle Änderung der vorgesehenen Systemarchitektur. Wie in Abbildung 3.3 zu sehen, wird auf einen expliziten Master-Szenegraph verzichtet und eine Kopie des Szenegraph auf jedem Reduzierer gespeichert. Diese Architektur begründet sich in der Verwendung von CADaVR-Szenen, die keine Veränderung der Geometrie zulassen, sondern nur Änderungen der Objekttransformationen. Es wird beträchtlicher Kommunikationsaufwand eingespart, indem vor Beginn des Renderings die geänderten Transformationsmatrizen übertragen werden und die Aufträge nur noch aus Objektnummern bestehen. Der Szenegraph wird nur einmalig nach Aktivierung des Plugins versendet.

In Abbildung 3.4 sind die Hauptkomponenten des Viewers und ihr Zusammenwirken skizziert. Die verteilte Komponente des Prototyps wird als Plugin nachgeladen. Der Hauptteil des Programms ist also vollständig unabhängig von CORBA und der Prototyp tritt „nur“ als Plugin für den CADaVR-Viewer auf.

Die Kopplung der einzelnen Komponenten erfolgt über den vom SceneGraph zur Verfügung gestellen Trigger-Mechanismus. So wird z. B. die Instanz von

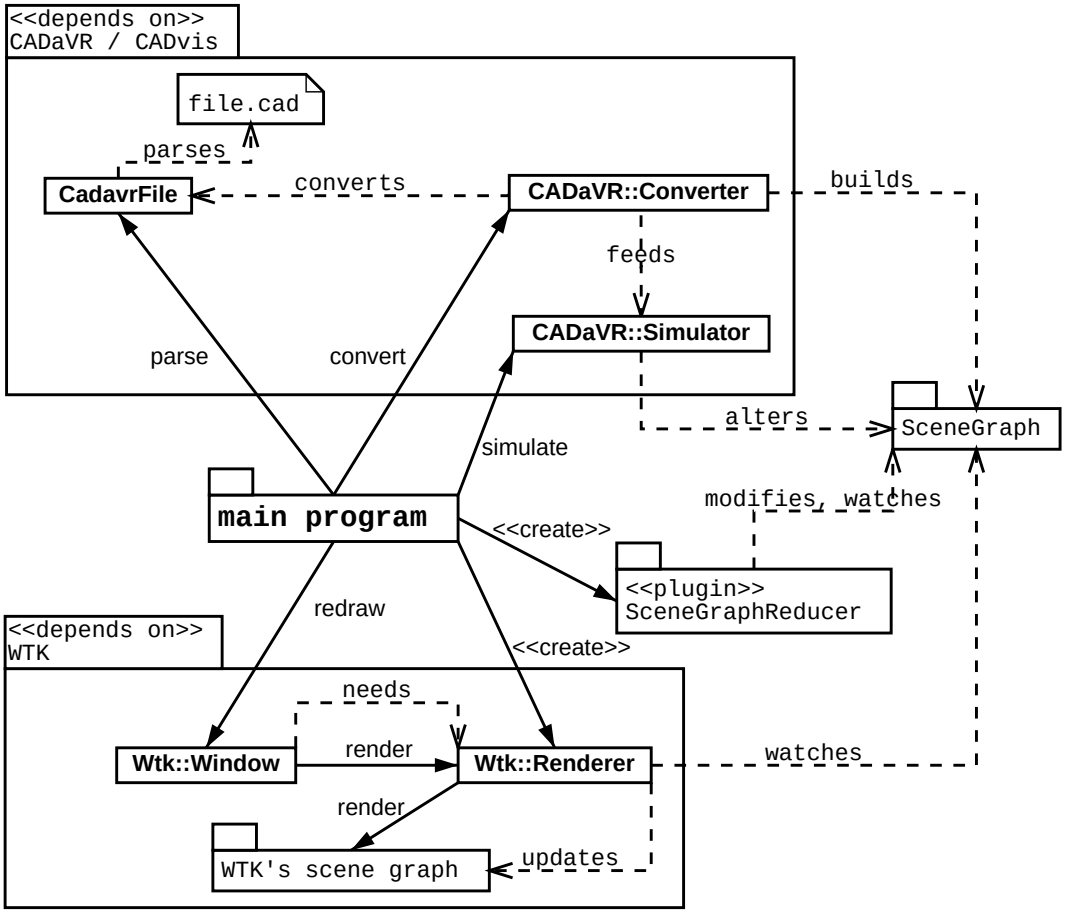


Abbildung 3.4.: Der Prototyp als Plugin im Kontext des CADaVR-Viewers. Die verteilten CORBA-Komponenten sind nicht abgebildet.

Wtk::Renderer über Änderungen in der Szene benachrichtigt und aktualisiert dann die entsprechenden WTK-Datenstrukturen.

Wie in Abbildung 3.5 zu sehen, wird die CORBA-Komponente über einen Trigger vom Renderer benachrichtigt, bevor ein neues Bild gerendert wird. Beim ersten Aufruf wird eine Repräsentation des Szenegraph generiert, die den Interfaces der CORBA-Applikation entspricht. Diese Repräsentation wird an das CORBA-Objekt ReducerScheduler verschickt, welches wiederum alle Reduktionsobjekte damit versorgt.

Vor dem Rendern eines Bildes wird jetzt eine Menge von Reduktionsanforderungen an den ReducerScheduler versendet, welcher diese an die Reduktionsob-

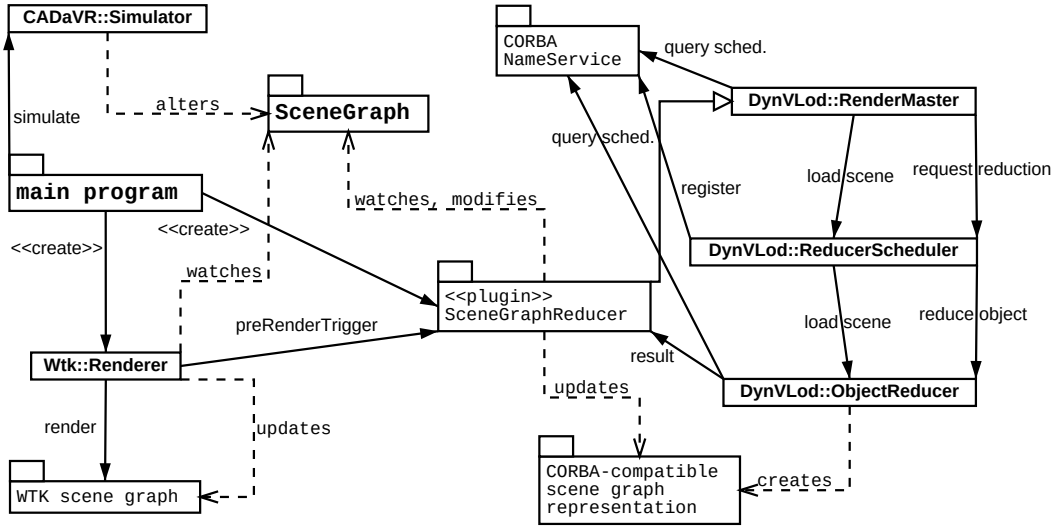


Abbildung 3.5.: Für den CORBA-Teil des Prototyps relevante Komponenten. Gut zu erkennen: Das Plugin SceneGraphReducer ist als einzige Komponente sowohl vom SceneGraph als auch vom CORBA-Teil abhängig.

jekte weiterleitet. Die Reduktionsobjekte generieren eine vereinfachte Darstellung (im Falle des Prototypen eine Bounding Box des Objektes) und senden diese an den RenderMaster, in diesem Falle den SceneGraphReducer, zurück.

Der SceneGraphReducer wiederum konvertiert die CORBA-spezifische Repräsentation des reduzierten Objektes in eine SceneGraph-spezifische und ersetzt das originale, nicht-reduzierte Objekt damit. Die Ersetzung des Objektes löst wiederum einen Trigger für den Renderer aus, welcher das Objekt im WTK-Szenegraph aktualisiert.

3.4. Leistung des Prototyps

Der vorliegende Prototyp besteht aus vier Komponenten, davon sind die letzten drei verteilt: CADaVR-Viewer, Plugin für CADaVR-Viewer, Scheduler für Reduktionsaufträge, theoretisch beliebige Anzahl von Reduzierern. Eine Szene aus SceneGraph kann in die CORBA-gerechte Form (definiert durch die Interfacebeschreibung in IDL) überführt und damit die geometrische und hierarchische Beschreibung der Szenedaten über Aufrufe verteilter Objekte versendet

werden. Der Prototyp generiert Reduktionsaufträgen vor dem Rendering. Es erfolgt eine „Reduktion“ der Objekte auf ihre Bounding Box durch einen Beispielreduzierer, BoundingBoxObjectReducer. Die reduzierten Objekte werden in SceneGraph reintegriert und damit automatisch gerendert. Das Rendering beginnt erst, wenn alle Reduktionsergebnisse eingetroffen sind.

Es können dynamisch Reduzierer hinzugefügt und entfernt werden. Das Entfernen ist allerdings nicht sehr robust, d. h. es könnte ein Auftrag verloren gehen und somit die Applikation zum Stillstand kommen.

3.5. Einschränkungen des Prototyps

Der Prototyp läßt – leider – noch einige Wünsche offen. Die folgende Liste stellt die meines Erachtens wichtigsten Unzulänglichkeiten des Prototypen dar. Denkbare zukünftige Erweiterungen werden später diskutiert.

- Die CORBA-Repräsentation des Szenegraph unterstützt bisher keinerlei Materialien.
- Die Reduktionsaufträge sind betrachter- und auflösungsunabhängig. Ein Reduzierer hat somit keine Möglichkeit, die Ausmaße oder relative Position eines Objektes auf dem Bildschirm zu bestimmen. ⇒ Kameratransformation und Fenstergröße müssen dem Reduzierer bekannt gemacht werden, da sie wahrscheinlich für fortgeschrittene Reduktionsverfahren notwendig sind.
- Die Geometriedaten werden nur einmal in die CORBA-Repräsentation überführt. Danach sind nur Änderungen der Transformationen möglich.
- Keine Ausnutzung möglicher Parallelität zwischen Rendering und Reduktion. Optimal wäre ein spezieller Renderer, der zuerst den Szenegraph traversiert, Reduktionsaufträge für alle sichtbaren Objekte generiert und dann bereits parallel zur Reduktion mit dem Rendering verfügbarer Objekte beginnt (nicht alle Objekte müssen reduziert werden). Es sollte nach Möglichkeit also nicht auf das Eintreffen von reduzierten Objekten gewartet werden. An dieser Stelle ist auch ein spezieller Rendering-Modus denkbar, der vorhandene Reduktionen wiederverwendet, wenn die benötigten noch nicht eingetroffen sind.

- Die Reduzierer erhalten für jedes Bild eine Kopie aller seit dem letzten Bild veränderten Transformationsmatrizen. Es werden also auch die Transformationen für Objekte übertragen, die gar nicht reduziert werden können.

- Die Applikation ist nicht gegen den Ausfall eines Reduzierers gesichert. Da die Reduktion optimalerweise asynchron abläuft, wird die Applikation nicht automatisch vom Ausfall eines Reduzierers benachrichtigt. Der Scheduler sollte die Verfügbarkeit der Reduzierer überwachen und eventuell Aufträge erneut vergeben.

4. Zusammenfassung

Der Prototyp demonstriert die prinzipielle Realisierbarkeit der verteilten Reduktion von Geometriedaten.

Es ergeben sich eine Reihe von weiterführenden Fragestellungen, die im Rahmen von Projekt-, Studien- oder Diplomarbeiten untersucht werden können.

5. Problemkomplexe für weiterführende Arbeiten

Diese Arbeit wirft eine Reihe von Fragen und Problemen auf. Es gibt eine Menge von Problemkomplexen im Zusammenhang mit der verteilten Reduktion von Objekten, die ich im Folgenden aufführe und näher erläutere.

5.1. Sinnvolle Objektreduktionen

Aufgrund der Komplexität dieses Themas wurde im Prototyp eine für praktische Anwendungen wenig sinnvolle Objektreduktion implementiert. Es ist zu untersuchen, welche Art von Objektreduktionen in Frage kommen und welche davon Echtzeitanforderungen genügen können.

An dieser Stelle ist auch der Inhalt eines Reduktionsauftrages genauer zu bestimmen. Für eine sinnvolle Objektreduktion wird wahrscheinlich eine Projektions- und Kameramatrix für das aktuelle Bild sowie Informationen über die Bildgeometrie benötigt. Denkbar ist auch die Vorgabe von Maximalkomplexitäten (Anzahl der Punkte und Polygone) durch den Renderer.

5.2. Vervollständigung des CORBA-Szenegraph-Interface

Die CORBA-Repräsentation des Szenegraph muß vervollständigt werden. Sie unterstützt noch keine Materialien. Der Reduzierer hat somit nicht einmal die Möglichkeit, eine Flächenfarbe zu ermitteln oder festzulegen.

5.3. Aktualisierung des verteilten Szenegraph

Der Prototyp unterstützt keine Modifikation der Szenegeometrie. Der Szenegraph wird genau einmal in die CORBA-Repräsentation überführt. Wenn die Szenegeometrie zur Laufzeit geändert werden soll, wird ein Protokoll (= CORBA-Interface) zum Scheduler und den Reduzierern benötigt, welches dies ermöglicht. Zu dieser Problematik wurde anderweitig bereits umfassende Grundlagenforschung betrieben. Möglicherweise ist hier OpenSG (`http://www.opensg.org/`) oder zumindest Ansätze davon verwendbar.

5.4. Effizientere Verteilung globaler Informationen

Informationen, die alle Reduzierer erhalten müssen (Transformationsupdates, Kopie des Szenegraph), werden zur Zeit seriell an jeden einzelnen Reduzierer versendet. Mit Hilfe von Multicasting könnte ein beträchtlicher Kommunikationsaufwand eingespart werden. ACE beinhaltet zum Beispiel eine Bibliothek für zuverlässige Multicasts („Reliable Multicasting“), welche in diesem Kontext eingesetzt werden könnte.

5.5. Bessere Integration reduzierter Objekte in den Szenegraph

Der Prototyp ersetzt die nicht reduzierten Objekte im Szenegraph einfach durch die reduzierten (das Original-Objekt geht damit verloren und ist nur noch innerhalb der CORBA-Repräsentation des Szenegraph vorhanden). Optimal wäre eine Implementation einer Subklasse von SwitchObject, die die reduzierbaren Objekte besser verwaltet und z. B. die vorgegebenen Level-of-Detail-Stufen beibehält.

5.6. Automatische Klassifikation reduzierbarer Objekte

Zur Zeit werden alle Objekte zur Reduktion herangezogen, die als Level-of-Detail-Objekte modelliert sind. Wünschenswert ist allerdings, daß die Appli-

kation automatisch den Szenegraph durchmustert und heuristisch Objekte für die Reduktion auswählt.

5.7. Wiederverwendung reduzierter Objekte

Je nach Art der Objektreduktion sind vermutlich einige Reduktionen innerhalb gewisser Grenzen wiederverwendbar und müssen deshalb nicht erneut berechnet werden. Dies ist im Protoyp bereits vorgesehen: Ein reduziertes Objekt wird mit einem Satz von Validitätskriterien an den Renderknoten zurückgeliefert. Es ist zu untersuchen, welche Kriterien für die Wiederverwendbarkeit von Reduktionen relevant sind. Im Prototyp sind z. B. eine Bounding Box (innerhalb derer sich das Objekt befinden muss) und ein Raumwinkel (in dem das Objekt erscheinen muß) vorgesehen, werden jedoch nicht verwendet, da die Objektreduktion für jedes Bild erneut vorgenommen wird.

5.8. Erweiterung des Szenegraph

Derzeit ist der Szenegraph kein Graph im eigentlichen Sinne, sondern ein Baum. Die SceneGraph-Bibliothek, die den Kern des CADaVR-Viewers darstellt, könnte derart erweitert werden, daß der Szenegraph tatsächlich ein azyklischer gerichteter Graph ist. Somit könnte bei vielfach instantiierten Objekten (z. B. Büsche) ein beträchtlicher Rechenaufwand eingespart werden. Im Idealfall müssen bei n Instanzen nicht n Reduktionen generiert werden, da bereits vorhandene Reduktionen wiederverwendet werden können.

5.9. Höhere Parallelisierung innerhalb der Komponenten

Um die Komplexität der Implementation von Anfang an in Grenzen zu halten, wurde auf den Einsatz von Multithreading komplett verzichtet. Die verwendete CORBA-Implementation unterstützt Multithreading jedoch in vielfältiger Art und es sollte aus Effizienzgründen auch davon Gebrauch gemacht werden.

Mindestens der Scheduler und die Empfangskomponente des Renderknotens sollten parallelisiert werden, da sonst unnötige Wartezeiten in Kauf zu nehmen sind. In diesem Rahmen ist evtl. auch die Verwending von Asynchronous Method Invocation (AMI) für die CORBA-Kommunikation von Vorteil.

A. Beschreibung der CORBA-Interfaces

Es folgt ein Listing der CORBA-Interfaces beschrieben in OMG IDL (Interface Description Language). Die Interface-Beschreibung läßt sich in zwei Teile gliedern:

1. Beschreibung der CORBA-Repräsentation des Szenegraph
2. Interfaces der verteilten Applikation

```
module DynVLod
{
    // used for name service look up
    const string ReducerScheduler_name = "ReducerScheduler";

// first part: scene graph represented in CORBA

    typedef float Vertex[3];
    typedef float Matrix[4][4];

    struct Quaternion
    {
        float   s;
        Vertex  v;
    };

    // we reference scene objects by their ID - using CORBA references
    // would be overkill (the decision here is to threat scene objects like
    // data rather than like real objects)
    typedef unsigned long SceneObjectId;

    struct BoundingBox
    {
        Vertex min;
        Vertex max;
```

```
};

union OptionalBoundingBox switch (boolean)
{
    case TRUE:
        BoundingBox value;
};

struct SolidAngle    // Raumwinkel
{
    double  value;
    Vertex  center;
    Vertex  direction;
};

union OptionalSolidAngle switch (boolean)
{
    case TRUE:
        SolidAngle  value;
};

/* better approach?:
struct Transformation
{
    Vertex      position;
    Quaternion  rotation;
    Vertex      scaling;
};
*/
struct Transformation
{
    Matrix      frame;
};

struct Polygon
{
    sequence<unsigned long> point_indices;
    // a lot of possible attributes are left out here
};

struct Faceset
{
    sequence<Vertex>    points;
    sequence<Polygon>   polygons;
    // a lot of possible attributes are left out here
};

struct SceneObject
```

```
    {
        string              name;
        SceneObjectId       id;
        Transformation      transform;
        sequence<Faceset>   facesets;
        sequence<SceneObject>   childs;
    };

// second part: description of distributed application interfaces

    struct TransformationUpdate
    {
        SceneObjectId   scene_object_id;
        Transformation  transform;
    };

    typedef sequence<TransformationUpdate> TransformationUpdateSequence;

    interface ReducedObjectReceiver; // forward declaration

    struct ReductionRequest
    {
        SceneObjectId object_id;
        ReducedObjectReceiver result_receiver;
    };

    interface ObjectReducer
    {
        // hand this reducer instance over a copy of the scene we're working on
        boolean LoadMasterScene (
            in SceneObject scene
            );

        // update global per-frame information
        void StartNextFrame (
            in unsigned long frame_no,
            in TransformationUpdateSequence updates
            );

        // TODO: decide whether new transformations should be saved
        oneway void ReduceObject (
            in ReductionRequest reduction_request
            );

        struct ReductionValidityConstraints
        {
            // bounding box where this object is valid, not object's bounding box
            OptionalBoundingBox  bounding_box;
```

```
            OptionalSolidAngle    solid_angle;
        };

        void Shutdown ();
    };

    interface ReducedObjectReceiver
    {
        void ReceiveReducedObject (                                          130
            in SceneObject reduced_object,
            in ObjectReducer::ReductionValidityConstraints validity_constraints
            );

    };

    interface ReducerScheduler
    {
        // set global parameters for the next frame
        // (avoids redundant transmission)                                   140
        // TODO: add window size, view point, model view matrix etc.
        void StartNextFrame (
            in unsigned long frame_no,
            in TransformationUpdateSequence updates
            );

        void EnqueueReductionRequest (
            in ReductionRequest reduction_request
            );
        // assign this reducer an ID which it uses to send back heartbeats   150
        // after work completion
        long RegisterObjectReducer (
            in ObjectReducer reducer
            );
        boolean LoadMasterScene (
            in SceneObject scene
            );
        oneway void ReceiveHeartbeat (
            in long id
            );                                                               160
        void UnregisterObjectReducer (
            in long id
            );
        void Shutdown ();
    };

};

// vim:ts=4:expandtab
```

B. Bedienungsanleitung

Der Prototyp besteht aus mehreren Teilen:

- Renderer
- Scheduler
- Reduzierer

Die Inbetriebnahme des Prototypen ist leider verhältnismäßig kompliziert, da als Renderer der CADaVR-Viewer für Unix verwendet wird und zusätzlich noch die CORBA-Implementierung vorbereitet werden muß.

Alle im folgenden angegebenen Shell-Kommandos sind für eine Bourne-Shell (also z.B. `bash` ausgelegt).

Keine der beschriebenen Komponenten muß (oder kann) in irgend einer Weise auf einem System installiert werden. Alle Programme werden direkt aus den Sourcecode-Bäumen heraus verwendet.

B.1. CADvis-Portierung

Eine mit dem Prototypen funktionsfähige Version des CADaVR-Viewers ist im CVS-Repository mit dem CVS-Tag `Studienarbeit_Prototyp` versehen. Den Sourcecode erhält man aus dem Repository mit:

```
export CVS_RSH=ssh
export CVS_SERVER=/usr/local/bin/cvs
cvs -d rad:/home/CVSROOT/Repository checkout\
   -r Studienarbeit_Prototyp -d CADvis-src CADvis/src
```

Für die Kompilierung benötigt man den GNU C Compiler (Version $\geq$ 2.95.3), bison++ (Version $\geq$ 1.28), flexx++ (Version $\geq$ 1.28-7) und WTK (Version $\geq$ 8). Falls bison++ nicht vorhanden ist, findet sich eine Version im CADvis-Sourcebaum im Archiv `src/Tools/bison++-1.21-8.tar.bz2`.

Eventuell sind noch systemspezifische Einstellungen in der Datei `src/Common/Make.defs` notwendig, (siehe z.B. die Variablen `WTKDIR`, `X11DIR`, `LEX`, `BISONXX`).

Der Sourcecode ist unter Linux, HP-UX und IRIX lauffähig, die Kompilierumgebung ist außerdem multiplattformfähig, d. h. im selben Sourcebaum können gleichzeitig Kompilate für verschiedene Plattformen existieren (die Objekt-Dateien werden jeweils im Verzeichnis `.obj.Plattform`, die Bibliotheken in `lib.Plattform` und die Programme in `bin.Plattform` abgelegt).

Die Kompilierung aller notwendigen Bibliotheken wird nun durch den Aufruf von GNU make (unter HP-UX und IRIX als `gmake` verfügbar) initiiert:

```
cd CADvis-src
gmake
```

B.2. CORBA-Implementation

Wie bereits erwähnt, wird als CORBA-Implementation TAO verwendet. Die Kompilierung von TAO ist recht umfangreich und benötigt viel Plattenplatz (siehe dazu die TAO Dokumentation). Fertig kompilierte Versionen von TAO (und der benötigten ACE-Bibliothek) finden sich im Verzeichnis `/afs/tu-chemnitz.de/project/dynvlod`. Die notwendigen Umgebungsvariablen setzt man durch Einbinden der Datei `setup.ace` im genannten Verzeichnis:

```
. /afs/tu-chemnitz.de/project/dynvlod/setup.ace
```

Danach zeigen die Umgebungsvariablen `ACE_ROOT` und `TAO_ROOT` auf die Verzeichnisse mit den Kompilaten für die aktuelle Plattform. Außerdem wird `LD_LIBRARY_PATH` entsprechend angepaßt.

B.3. Protoyp

Ein funktionierender Stand des Prototypen ist ebenfalls mit dem CVS-Tag `Studienarbeit_Prototyp` versehen. Man erhält die markierte Kopie mit:

```
export CVS_RSH=ssh
export CVS_SERVER=/usr/local/bin/cvs
cvs -d rad:/home/CVSROOT/Repository checkout\
   -r Studienarbeit_Prototyp -d tisc-SA\
   tisc/Studienarbeit/Prototyp
```

Auch dieser Sourcebaum ist multiplattformfähig. Für das Plugin wird der Pfad zum CADaVR-Viewer-Sourcebaum benötigt, sowie die Umgebungsvariablen für TAO:

```
export CADVIS_ROOT=/pfad/zu/CADvis-src
. /afs/tu-chemnitz.de/project/dynvlod/setup.ace
```

Die Kompilierung wird wiederum mit GNU make initiiert:

```
cd tisc-SA
gmake
```

B.4. Starten der Applikation

Bevor der eigentliche Viewer gestartet werden kann, müssen zunächst die Komponenten für die verteilte Reduktion gestartet werden[1]. Die Abhängigkeiten zwischen den Komponenten sind in Abbildung B.1 dargestellt.

Aus den Abhängigkeiten ergibt sich die Startreihenfolge für die Komponenten:

1. CORBA NameService
2. Scheduler
3. Reduzierer
4. Viewer

[1] Für den Viewer ohne das Reduktionsplugin ist dies nicht notwendig, er benötigt nur die WTK-relevanten Umgebungsvariablen. Eine vollständigere Implementation der CADaVR-Spezifikation findet sich außerdem im Programm `CADvis/src/bin.‘uname‘/viewer`.

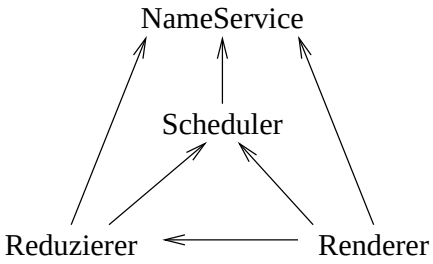


Abbildung B.1.: Abhängigkeit zwischen den Komponenten des Protoyps

B.4.1. CORBA NameService

Eine Implementation des CORBA NameService ist Teil von TAO. TAO bietet zwei Wege, um den NameService aufzufinden: Multicasting und die explizite Übergabe der Adresse in einer IOR[2]. Die Verwendung von Multicasting wurde nicht getestet, deshalb ist es notwendig, die IOR des NameService den anderen Komponenten mitzuteilen.

Der NameService wird gestartet mit:

```
. /afs/tu-chemnitz.de/project/dynvlod/setup.ace
$TAO_ROOT/orbsvcs/Naming_Service/Naming_Service\
   -ORBEndPoint iiop://voller.hostname.domain\
   -o nameservice.ior &
```

Jetzt steht in der Datei `nameservice.ior` die IOR des NameService[3]. Um den Aufruf der weiteren Komponenten zu vereinfachen (und lange Kommandozeilenargumente zu vermeiden), bietet es sich an, die Umgebungsvariable `NameServiceIOR` zu setzen:

```
export NameServiceIOR=file:///pfad/zu/nameservice.ior
```

(Zu beachten sind die drei Slashes nach `file:`!)

[2] Interoperable Object Reference. Innerhalb des CORBA-Standards definierte Schnittstelle zur Beschreibung einer Objektreferenz. In einer IOR sind sowohl die Adresse als auch das Protokoll für den Zugriff auf ein Objekt kodiert. Dabei kann ein Objekt über verschiedene Protokolle und Adressen gleichzeitig erreichbar sein.

[3] Mit dem Tool catior kann man die IOR dekodieren und anzeigen:

```
$TAO_ROOT/utils/catior/catior -f nameservice.ior
```

B.4.2. Scheduler

Als nächstes wird der Scheduler für den Protoypen gestartet. Prinzipiell können alle Komponenten auf verschiedenen Rechnern laufen – diese müssen nur per Netzwerk einander erreichen können.

Der Scheduler wird wie folgt mit Hilfe des Shellskriptes `run` gestartet:

```
export NameServiceIOR=file:///pfad/zu/nameservice.ior
cd tisc-SA/ReducerScheduler
./run
```

`run` dient dabei lediglich der Auswahl des plattformspezifischen Executables. Bei Erfolg sollte in etwa folgende Ausgabe erscheinen:

```
POA activated.
Registering with NameService.
Activating 2 ORB threads.
thread 1026: starting ORB.
Ready.
Waiting for a signal to arrive.
thread 2051: starting ORB.
```

B.4.3. Reduzierer

Die Reduzierer werden analog zum Scheduler gestartet. Im Unterschied zu diesem wird allerdings ein Plugin benötigt, welches die eigentliche Reduktion durchführt. Das Shellskript `run` lädt standardmäßig den einzig verfügbaren Reduzierer, `BoundingBoxObjectReducer`.

Reduzierer starten:

```
export NameServiceIOR=file:///pfad/zu/nameservice.ior
cd tisc-SA/ObjectReducer
./run
```

Bei erfolgreichem Start erfolgt in etwa folgende Ausgabe:

```
POA activated.
Loading plug-in ../bin.Linux/BoundingBoxObjectReducer.so
1034170281.730018 DynVLod_ObjectReducer_i: constructing
BoundingBoxObjectReducer: constructed.
ObjectReducerServer: registering 0x808f09c
1034170281.744797 ObjectReducer registered with ID 1
Creating 2 ORB threads.
thread 1026: starting ORB.
Ready.
Waiting for shutdown.
thread 2051: starting ORB.
```

B.4.4. Viewer

Zuletzt wird der eigentliche CADaVR-Viewer gestartet. Die von WTK benötigten Umgebungsvariablen müssen gesetzt sein[4]. Um Problemen mit fehlenden Materialdateien vorzubeugen, bietet es sich an, den Viewer aus dem Verzeichnis mit den `.cad`-Dateien zu starten:

```
VIEWER=/pfad/zu/CADvis/src/bin.‘uname‘/wtkrendertest
PLUGIN=/pfad/zu/tisc-SA/bin.‘uname‘/SceneGraphReducer.so
export NameServiceIOR=file:///pfad/zu/nameservice.ior
cd /pfad/zu/Modellen
$VIEWER Modell.cad $PLUGIN
```

Der Viewer unterstützt derzeit nur Maus- und Tastaturbedienung. Die Navigation erfolgt im Standard-WTK-Modus. Folgende Tastaturkürzel werden unterstützt:

a	Antialiasing ein/aus
w	Wireframe ein/aus
l	Beleuchtung ein/aus
s	Flat / Smooth Shading
g	Screenshot in Datei screen####.bmp schreiben
>	Navigationsgeschwindigkeit erhöhen
<	Navigationsgeschwindigkeit verringern
R	verteilte Reduktion ein/aus

[4] Auf den HP-UX-Maschinen kann man dies schnell und einfach sicherstellen mit:

```
. /opt/wtk/README.setup
```

Je nach Szenekomplexität kommt es nach der ersten Aktivierung der verteilten Reduktion zu einer größeren Wartezeit, da aus dem Szenegraph eine für CORBA verwendbare Repräsentation erzeugt wird. Nach dem Erzeugen der Repräsentation wird diese an den Scheduler verschickt, der diese wiederum an alle Reduzierer verteilt (beide Komponenten geben entsprechende Meldungen aus).

In Abbildung B.2 ff. sind die Auswirkungen der Reduktion mittels `BoundingBoxObjectReducer` beispielhaft dokumentiert.

Im vorliegenden Entwicklungsstand des Prototypen werden die Original-Objekte durch die Reduzierten ersetzt. Nach Deaktivierung der Reduktion bleibt also trotzdem die reduzierte Szene bestehen.

Abbildung B.2.: Screenshot einer Szene vor der Reduktion

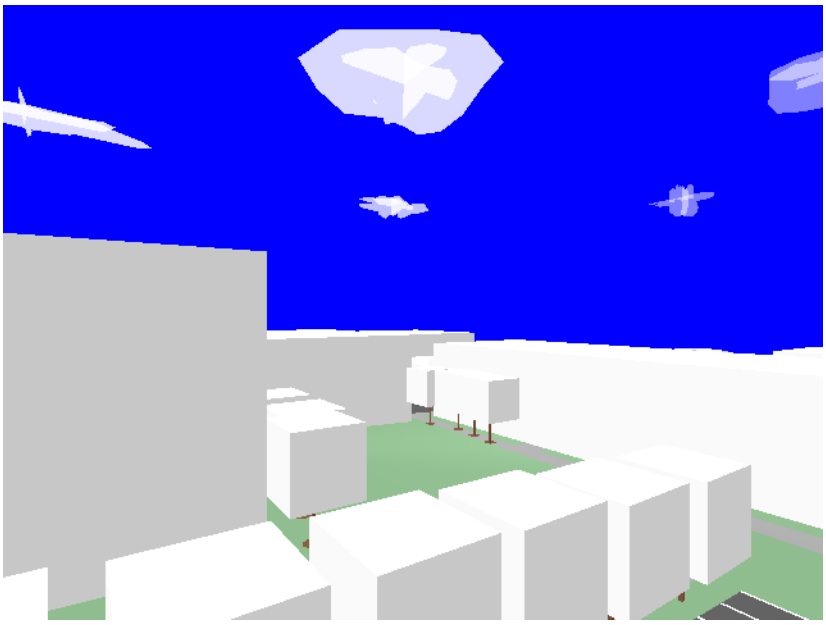


Abbildung B.3.: Screenshot einer Szene nach der Reduktion

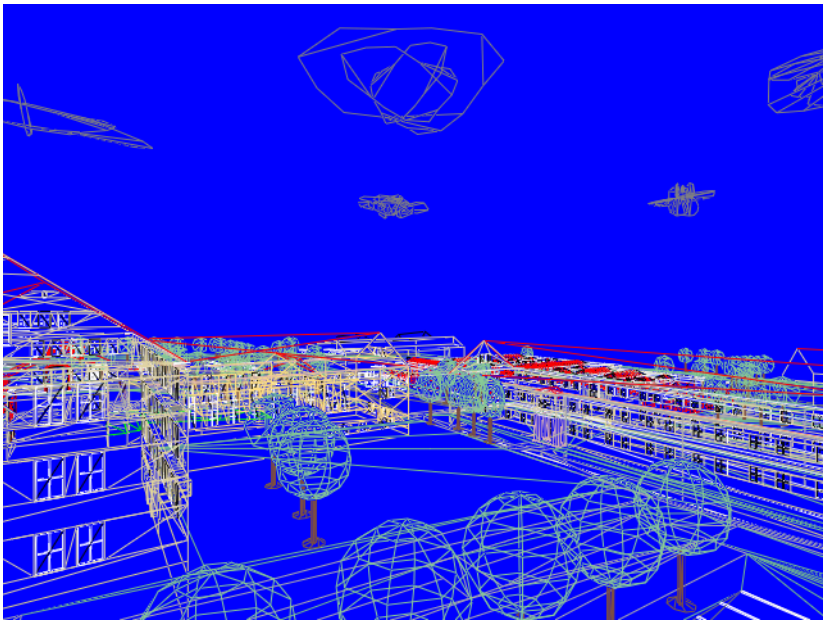


Abbildung B.4.: Screenshot einer Szene vor der Reduktion, Wireframe

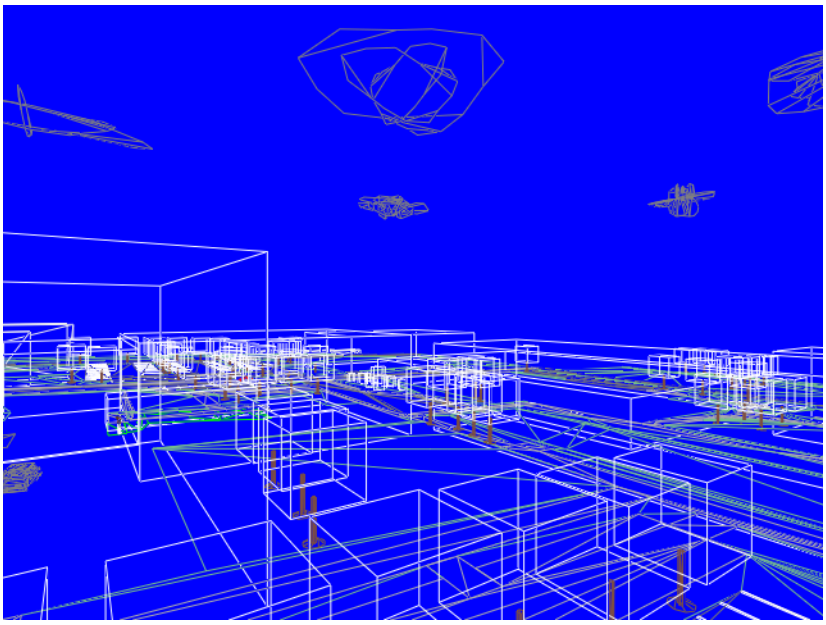


Abbildung B.5.: Screenshot einer Szene nach der Reduktion, Wireframe

Abbildungsverzeichnis

Glossar

ACE Adaptive Communication Environment, ein objektorientiertes, plattformübergreifendes C++ Toolkit für die Netzwerkprogrammierung, wie TAO am „Center for Distributed Object Computing“ entwickelt. Siehe [5].

CADaVR CADaVR ist eine Schnittstelle zwischen CAD-Daten und VR-Systemen, die am Lehrstuhl Computergrafik der TU Chemnitz entwickelt wurde. Siehe auch [11].

CAVE Abkürzung für „Cave Automatic Virtual Environment“, erstmals 1992 von der University of Illinois at Chicago vorgestellte VR-Umgebung. Allgemein als Oberbegriff für VR-Installationen verwendet, die aus einem Raum bestehen, an dessen Wände Bilder projiziert werden. Ein CAVE ist von mehreren Personen gleichzeitig begehbar und erzeugt einen sehr guten immersiven Eindruck, da sich die Projektionen über das gesamte Gesichtsfeld erstrecken.

CLiC Chemnitzer Linux Cluster, siehe auch [7].

Cluster Vernetzung einer größeren Menge von Standard-Computern zu einem Rechnerverbund für parallele Programme

CORBA Common Object Request Broker Architecture, Standard der OMG (`http://www.omg.org/`) für verteilte, objektorientierte Anwendungen. Zum CORBA-Standard zählen unter anderem die IDL (Interface Description Language) und das Protokoll IIOP. Siehe [9].

IDL Interface Description Language. Teil der CORBA-Spezifikation. IDL dient zur Beschreibung von Schnittstellen von verteilten Objekten. Aus der IDL-Beschreibung generiert ein ORB-spezifischer IDL-Compiler für die Zielsprache die entsprechende Infrastruktur, für die Verwendung dieser Interfaces. Eine IDL-Beschreibung kann man als eine Art plattform- und programmiersprachenunabhängiges `.h`-File ansehen.

IIOP Interoperable Inter ORB Protocol. Ein im CORBA-Standard spezifiziertes Protokoll zur Kommunikation zwischen ORBs. Zweck des Protokolls ist es, die Interoperabilität von ORBs verschiedener Hersteller zu gewährleisten. IIOP trägt dazu bei, daß in den Komponenten einer verteilten Applikation gleichzeitig verschiedene CORBA-Implementierungen eingesetzt werden können. Siehe [9].

IOR Interoperable Object Reference. Innerhalb des CORBA-Standards definierte Schnittstelle zur Beschreibung einer Objektreferenz. In einer IOR sind sowohl die Adresse als auch das Protokoll für den Zugriff auf ein Objekt kodiert. Dabei kann ein Objekt über verschiedene Protokolle und Adressen gleichzeitig erreichbar sein.

LOD Level-of-Detail. Eine Technik zur Beschleunigung der Darstellung virtueller Welten. Ein Objekt wird in verschieden komplexen Varianten vorbereitet und je nach Entfernung des Betrachters zum Objekt wird eine davon zum Rendering ausgewählt. Ein Haus kann aus großer Entfernung z.B. durch einen Quader dargestellt werden, aus der Nähe aber komplett ausmodelliert sein.

MPI Message Passing Interface, ein Standard für die Kommunikation von verteilten Programmen. Es existieren mehrere hochoptimierte, freie MPI-Implementationen.

OMG Object Management Group, ein internationales Konsortium aus über 600 Firmen, daß sich ausschließlich der Schaffung von herstellerunabhängigen und plattformübergreifenden Standards widmet. Erklärtes Ziel ist maximale Interoperabilität von Produkten verschiedenster Hersteller. Die Modellierungssprache UML stammt auch von der OMG.

ORB Object Request Broker, Teil der CORBA-Implementierung, die Anfragen an andere Objekte weiterleitet. Kann je nach Implementation Teil der Applikation, ein eigenständiger Prozeß oder eigener Server sein.

PVM Parallel Virtual Machine, ein zu MPI „konkurrierender“ Ansatz für Kommunikation in verteilten Systemen. Die erste Version wurde 1989 am „Oak Ridge National Laboratory“ entwickelt aber nur intern benutzt, im März 1991 wurde Version 2, im März 1993 Version 3 veröffentlicht. Im Gegensatz zu MPI ist PVM kein expliziter Standard sondern nur eine Bibliothek.

RFC Request For Comments, eine Serie von Dokumenten, die technische und organisatorische Aspekte des Internet beschreiben. Einige davon spezifizieren Internet Standards.

RPC Remote Procedure Call, ein Standard zum Aufruf von Prozeduren auf entfernten Rechnern. Ist u.a. Basis für das NFS-Dateisystem.

TAO „The ACE ORB“, eine CORBA-Implementierung, die am „Center for Distributed Object Computing“ der Washington University in St. Louis' begonnen wurde. Siehe [6].

UML Unified Modelling Language. Von der OMG ausgearbeiteter Standard für den objektorientierten Softwareentwurf. UML spezifiziert verschiedene Arten von Diagrammen für die Spezifikation und Dokumentation von Softwarearchitekturen. Siehe [8].

VR Virtual Reality. Sammelbegriff für die Simulation virtueller Welten. Dem Betrachter soll eine möglichst realistische Umgebung vorgespielt werden, in der er sich möglichst frei bewegen kann. Einsatzgebiete sind zum Beispiel Unterhaltung, Architektur, Konstruktion und Design.

WTK World Toolkit. Ursprünglich von der Firma Sense8 entwickelt (derzeitiger Eigentümer unklar aufgrund häufiger Übernahmen). WTK ist eine Bibliothek von C-Funktionen zur Erzeugung virtueller Welten. Enthält u.a. umfangreiche Funktionen zur Verwaltung von Szenegraphen und unterstützt eine Reihe von 3D-Peripherie.

Literaturverzeichnis

[1] Cruz-Neira, C. / Sandin, D. / DeFanti, T. 1992 *The CAVE : A Virtual Reality Theater*. Homepage. http://www.evl.uic.edu/pape/CAVE/oldCAVE/CAVE.html. 14.08.2002.

[2] Hübner, U. (2001). *Untersuchungen zur Kommunikation am CLiC*. Internetseite. http://www-user.tu-chemnitz.de/~huebner/cliccom/. 14.08.2002.

[3] Network Working Group, Sun Microsystems, Inc. (1987). *XDR: External Data Representation Standard*. RFC 1014.

[4] Network Working Group, Sun Microsystems, Inc. (1988). *RPC: Remote Procedure Call*. Protocol Specification Version 2. RFC 1057.

[5] Schmidt, Douglas C. *The Adaptive Communication Environment*. Homepage. http://www.cs.wustl.edu/~schmidt/ACE.html.

[6] Schmidt, Douglas C. *TAO - The ACE ORB*. Homepage. http://www.cs.wustl.edu/~schmidt/TAO.html.

[7] Universitätsrechenzentrum der Technischen Universität Chemnitz. (2001). *Chemnitzer Linux Cluster (CLiC)*. http://www.tu-chemnitz.de/urz/anwendungen/CLIC/.

[8] Object Management Group. (2000). *OMG Unified Modeling Language Specification*. Version 1.3. March 2000.

[9] Object Management Group. (2002). *The Common Object Request Broker: Architecture and Specification*. Revision 2.6.1. May 2002.

[10] Williams, T. / Kelley, C. (1998). *gnuplot*. An Interactive Plotting Program. Handbuch.

[11] Wenisch, M. / Langer, T. / Wagner, H. (1998). *CADaVR-Schnittstelle und Dateiformat.* Version 0.70 vom 25.08.1998. Technische Universität Chemnitz.