Integration paralleler Rendering-Verfahren für lose gekoppelte Systeme in OpenSG

Marcus Roth

Fraunhofer Institut für graphische Datenverarbeitung
Abteilung Visualisierung und Virtuelle Realität
Rundeturmstr. 6, D-64283 Darmstadt

Inhalt

Dieser Bericht beschreibt Entwurf und Implementierung einer Softwareumgebung, mit deren Hilfe es möglich ist, auf der Basis von OpenSG parallele Rendering-Verfahren für PC-Cluster zu realisieren. Aufbauend auf dieser Umgebung wird die Umsetzung einer Stereoprojektion, einer Multiprojektor Projektion und eines parallelen Sort-First Rendering-Algorithmus beschrieben. Abschließend werden Beispiele und Vorschläge für weitere parallele Rendering-Verfahren erläutert.

Einleitung

Applikationen aus den Bereichen CAD oder VR stellen sehr hohe Anforderungen an Graphiksysteme. In der Regel liegt die gewünschte Leistung jenseits des momentan technisch Machbaren. Um hier eine Abhilfe zu schaffen, wurden Verfahren entwickelt die den Rendering-Prozeß parallelisieren und somit auf der Grundlage bestehender Technologien eine Leistungssteigerung ermöglichen.
In OpenSG wurde dieser Tatsache Rechnung getragen, indem man bereits in einem frühen Designstadium die Unterstützung von Multiprozessorsystemen vorgesehen hat. Die hierbei entwickelten Mechanismen können in Mehrprozessorsystemen mit gemeinsam genutztem Hauptspeicher verwendet werden. Diese eng gekoppelten Systeme stellen jedoch nur eine Variante paralleler Architekturen dar. Mit dem Aufkommen preisgünstiger leistungsfähiger PCs, wurde es möglich, hohe Rechenleistung zu günstigen Preisen durch den Aufbau von PC-Clustern zu erreichen. Diese lose gekoppelten Systeme verfügen jedoch nicht mehr über einen gemeinsam genutzten Hauptspeicher. Die in OpenSG vorhandenen Multithreading-Mechanismen sind in PC-Clustern nicht verwendbar. Um dieses Problem zu beheben, muß OpenSG um die Möglichkeit der Prozeßkommunikation und Synchronisation über Rechnergrenzen hinweg erweitert werden.
Obwohl schon viele parallele Rendering-Verfahren entwickelt wurden, ist dieser Bereich weiterhin ein aktives Forschungsgebiet. Um bestehende aber auch zukünftige Verfahren einfach in OpenSG integrieren zu können, sind flexible und erweiterbare Schnittstellen erforderlich. Eine Applikation, die auf OpenSG aufsetzt, sollte mit einem möglichst geringen Aufwand von neu implementierten Verfahren profitieren können.
Die mittlerweile sehr hohen Graphikleistung, die von handelsüblichen PC erreicht wird, führen zu dem Wunsch, auch komplexere VR-Projektionen wie z.B. eine CAVE [CF93] oder eine Power-Wall aus günstigen Einzelkomponenten aufzubauen. Neben der Anforderung, aus einer Applikation heraus mehrere Projektionen zu betreiben, können mehrere PC auch dazu verwendet werden, gemeinsam Bilder für eine Projektion zu berechnen. Hierbei müssen die spezifischen Einschränkungen lose gekoppelter Rechnerarchitekturen berücksichtigt werden.

Bisherige Arbeiten

Es existieren viele Ansätze, das Rendering zu paralellisieren. Um die Vielzahl der Ansätze einordnen und vergleichen zu können, wird von Steven Molnart u.a. [MCE94] eine Klassifikation in „Sort-First“, „Sort-Middle“ und „Sort-Last“ vorgeschlagen. Hierbei wird der Prozeß des Renderns als Sortierproblem betrachtet. Vor dem Rendern liegt die Geometrie in einem ungeordneten Zustand vor. Ziel des Renderns ist es, die Geometrie in eine Ordnung zu bringen, so daß als Ergebnis ein korrektes Bild entsteht. Es wird hierbei davon ausgegangen, daß das Rendering in zwei aufeinanderfolgende Teilaufgaben zerlegt werden kann. Dies sind die Geometriebearbeitung und die Rasterisierung. Während der Geometriebearbeitung werden Objektpositionen berechnet, der Einfluß von Materialien und Lichtquellen bestimmt und graphische Primitive, die außerhalb des sichtbaren Bereichs liegen, verworfen. Während der Rasterisierung werden Bildschirmpunkte zugewiesen und deren Sichtbarkeit und Farbe berechnet. In parallelen Rendering-Verfahren kann die Sortierung während der Geometriebearbeitung, zwischen Geometriebearbeitung und Rasterisierung und während der Rasterisierung erfolgen. Abhängig davon spricht man von einem „Sort-First“, „Sort-Middle“ oder „Sort-Last“ Verfahren.

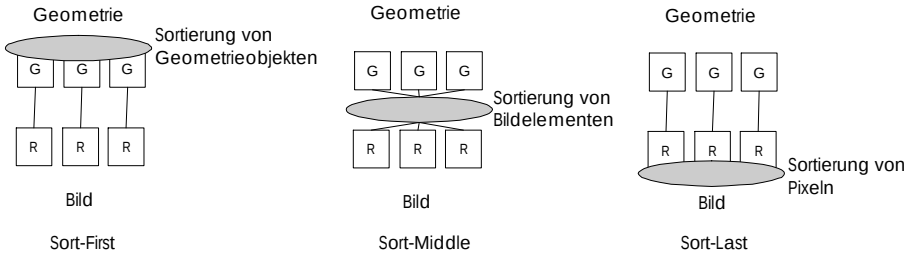

Abbildung 1 Klassifikation paralleler Rendering-Verfahren

Für lose gekoppelte Systeme bedeutet ein Sort-First Verfahren [MUE95], daß zu Beginn der Geometriebearbeitung jedes graphische Objekt einem Bildbereich zugeordnet wird. Jeder Rechner ist für einen Bildbereich zuständig und rendert nur die Objekte, die innerhalb seines Bereiches liegen. WireGL [HEB01] ist ein Beispiel für eine Sort-First Implementierung. In WireGL werden OpenGL-Befehle auf mehrere Rechner verteilt. Der Hauptvorteil dieses Ansatzes besteht darin, daß das Rendering bestehender OpenGL-Applikationen ohne Anpassung parallelisiert werden kann.
Sort-Middle Verfahren lasen sich in lose gekoppelten Systemen nur schwer realisieren, da moderne Graphik-Hardware sowohl die Geometriebearbeitung als auch die Rasterisierung durchführt. Die Zwischenergebnisse stehen für eine Sortierung nicht zur Verfügung. In [EIH00] wird hierfür eine spezielle Hardware-Architektur vorgeschlagen.
In Sort-Last Verfahren berechnet jedes Einzelsystem mit einem Teil der Geometrie ein komplettes Bild. Am Ende der Rasterisierung werden die Einzelbilder zu einem Bild zusammengefügt. Hierbei sind neben den Pixeln auch Tiefeninformationen erforderlich. In [NZ99] wird ein Sort-Last Verfahren beschrieben, das durch eine geschickte Objektverteilung die Übertragung der Tiefeninformation umgeht.
Neben Verfahren, die exakt in die von Molnart vorgeschlagenen Kategorien einzuordnen sind, existieren auch solche aus Kombinationen der drei Kategorien. In [SFL00] wird eine Kombination aus Sort-First und Sort-Last für PC-Cluster beschrieben.
Ein spezifisches Problem lose gekoppelter Systeme gegenüber eng gekoppelten Systemen besteht in der, im Vergleich zu Hauptspeicherzugriffen, geringen Geschwindigkeit der Datenübertragung. PC-Cluster sind üblicherweise über ein Ethernet oder Myrinet miteinander Verbunden. Hierbei werden maximale Datenraten von 1Gbit erreicht. Bei einer Frame-Rate von 25 Bildern/s können pro Bild maximal ca. 5Mbyte übertragen werden. Je nach Art des parallelen Rendering-Verfahrens kann dies die Skalierbarkeit des Gesamtsystems beschränken.
In [CCF01] werden verschiedene Ebenen in einer Applikation untersucht, in denen die Trennung zwischen Client und Server erfolgen kann, sowie den daraus resultierenden Datenvolumen und Verzögerungszeiten für ein Multi-Projektor Display.

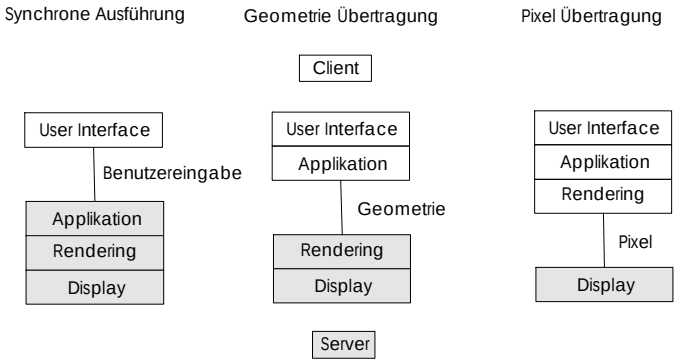

Abbildung 2 Aufgabenverteilung zwischen Client und Server

Die Synchrone Ausführung liefert in Bezug auf Latenz und Datenvolumen die besten Ergebnisse. Es wird vorausgesetzt, daß identische Server-Applikationen auf verschiedenen Rechnern ausgeführt werden. Jeder Server erhält die gleichen Eingabewerte und somit sind alle Server in einem identischen Zustand. Dies kann jedoch nur für Applikationen gewährleistet werden, die lediglich in einem Thread laufen. Der Zustand von multithreaded Applikationen ist nicht mehr ausschließlich durch die Eingabedaten bestimmt, sondern auch durch die zeitliche Abfolge der Thread-Bearbeitung. Für VR-Applikationen kommt dieser Ansatz daher nur bedingt in Frage. Die Übertragung von Pixeln bietet dagegen wenig Spielraum für eine effiziente Parallelisierung. Die Übertragung von Geometriedaten an parallel rendernde PC hat zum einen den Vorteil, daß das komplette Rendering parallelisiert werden kann, und zum anderen eine Applikation nur geringe Anpassungen erfordert um paralleles

Rendering zu ermöglichen. Der im folgenden beschriebenen Ansatz beruht daher auf der Übertragung von Objekten des Szenegraphen.

Systementwurf

In OpenSG wird Multithreading durch das Konzept der Aspects unterstützt. Hierbei kann jedem Thread seine persönliche Sicht der Datenstrukturen zur Verfügung gestellt werden. Modifikationen eines Threads haben zunächst keine Auswirkung auf andere Threads. Erst zu definierten Zeitpunkten werden die Änderungen synchronisiert. Um die Änderungen synchronisieren zu können, wird jede Feldmanipulation in einer Änderungsliste festgehalten. Diese Liste dient als Basis für die Datenverteilung. An einem Synchronisationspunkt werden die lokalen Threads synchronisiert und alle Änderungen die in der Änderungsliste vermerkt sind, über eine Netzwerkverbindung mit einem Remote-Aspect synchronisiert. Dieser Ansatz hat den Vorteil, daß nur geänderte Daten übertragen werden. Da Aspects von parallel laufenden Threads bearbeitet werden, kann auch der Synchronisations-Thread parallel zum Anwendungs-Thread ausgeführt werden.

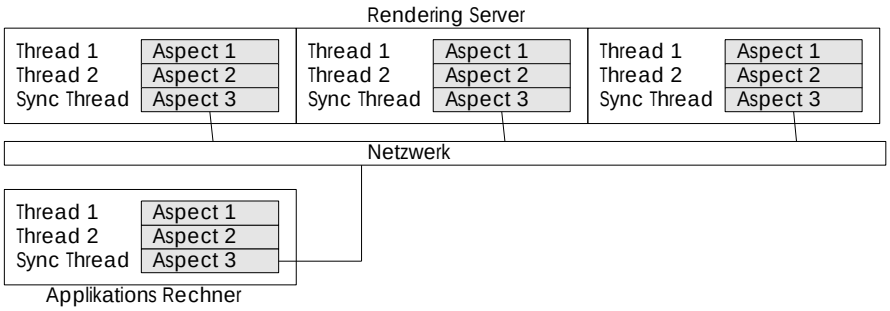

Abbildung 3 OpenSG Aspect Erweiterung um Remote-Aspects

Der Mechanismus, mit dem die Daten verteilt werden, soll weitestgehend unabhängig von der eingesetzten Netzwerk-Hardware sein. Mechanismen wie z.B. Multicast oder Zerocopy müssen durch eine abstrakte Softwareschicht abgebildet werden. Um dies zu erreichen, wird OpenSG um ein binäres Dateninterface erweitert. Der „BinaryDataHandler“ ist dafür zuständig, binäre Daten wie z.B. Inhalte von Szenegraph-Knoten effizient zu lesen und zu schreiben. Er unterstützt gepufferte und ungepufferte Datenströme sowie Zerocopy-Read und Zerocopy-Write. Hierauf setzen mehrere Datahandler wie beispielsweise der BinFileDataHandler und die Netzwerk Connection auf.
Die Connection bildet ein abstraktes Interface für eine 1 zu N Verbindung. Alle Connections bieten ein einheitliches Interface für den Verbindungsaufbau, Datentransfer und Synchronisation. Aus Sicht des Interfaces ist es unerheblich, ob die 1 zu N Verbindung durch Punkt zu Punkt Verbindungen oder durch ein Multicast Protokoll realisiert sind. Konkrete Connection-Implementierungen sind „MulticastConnection“, „StreamSockConnection“ und „MyrinetConnection“.

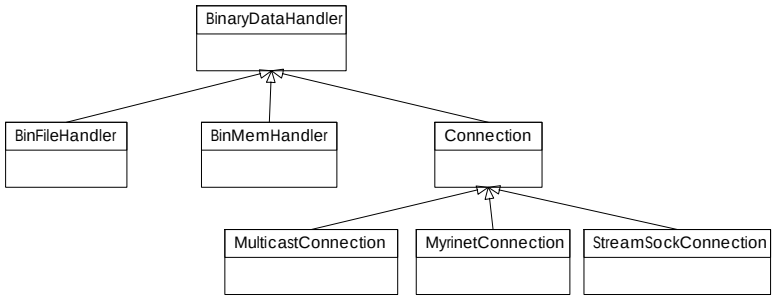

Abbildung 4 Klassenhirachie der Netzwerk-Connection

Da bisher kein optimales Verfahren für paralleles Rendering in Rechner-Clustern existiert und dieser Bereich noch ein aktives Forschungsgebiet ist, muß die Art und Weise, wie solche Mechanismen in OpenSG integriert werden, sehr flexibel für zukünftige Erweiterungen sein. Um dies zu erreichen, muß das Clustering für eine Applikation weitestgehend transparent sein. Es muß eine Stelle in OpenSG gefunden werden, an der die Konfiguration der Clustering-Verfahren erfolgen kann. Obwohl hierfür viele Varianten denkbar sind, wurde das OpenSG-Window als zentrales Konfigurationselement für das parallele Rendern im Cluster ausgewählt. In

OpenSG beschreibt ein Window, was in welcher Form gerendert wird. Dies läßt sich leicht erweitern, indem man definiert, daß ein Cluster-Window beschreibt, was in welcher Form wo gerendert wird. Ein Applikations-entwickler kann mit diesem Ansatz weiterhin wie gewohnt mit OpenSG Windows arbeiten. Er muß lediglich ein Cluster-Window anstatt eines GLUT- oder QT-Windows verwenden. Die Client-Applikation ersetzt damit eine Verbindung des graphischen Contexts mit einem GUI durch eine Netzwerkverbindung. Das Cluster-Window hat hierbei die volle Kontrolle über den parallelen Rendering-Algorithmus und verbindet den Client über ein Netzwerk mit GLUT- oder QT-Windows der Rendeirng-Server. Da ein Window als OpenSG Field-Container implementiert ist, werden alle Feldänderungen automatisch über das Netzwerk synchronisiert. Für die Parameterübertragung ist daher kein besonderer Mechanismus erforderlich.

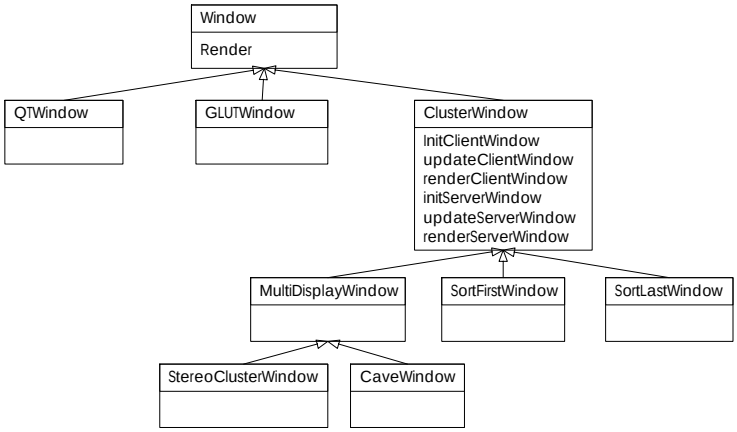

Abbildung 5 Cluster-Konfiguration als OpenSG Window

Multi-Display Cluster-Rendering

Bei einer Multi-Display Projektion mit Hilfe eines PC-Clusters wird jede Einzelprojektion durch einen PC gesteuert. Für eine Stereo-Projektion sind beispielsweise zwei PC erforderlich. Ein PC rendert das Bild für das rechte, der andere für das linke Auge. Mit Hilfe der oben beschriebenen Netzwerk-Connection wird der Szenegraph auf jeden Display-PC repliziert. Das Clustering-Verfahren muß lediglich die Kamera-Parameter für die einzelnen Projektionen anpassen.
Als Beispiel für diesen Konfigurationstyp wurde ein MultiDisplayWindow implementiert. Mit diesem Window werden Großbildprojektionen beschrieben, die aus Einzelprojektionen zusammengesetzt sind. In [LCC00] wird der Aufbau einer solchen Projektion an der Princeton University beschrieben. Als Konfigurationsparameter sind lediglich die Rendering-Server und die Anzahl der Projektoren in horizontaler und vertikaler Richtung anzugeben.

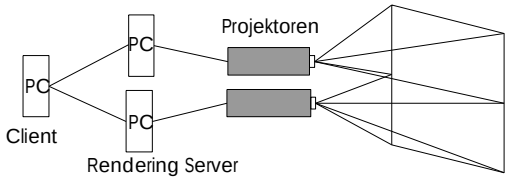

Abbildung 6 Multi-Projektor Display

Parallele Image-Composition Algorithmen

Rendering-Verfahren für PC-Cluster, bei denen viele PC gemeinsam ein Bild berechnen, erfordern einen höheren Aufwand als Multi-Projektor Installationen. Die Synchronisation des Szenegraphen ist bei beiden Ansätzen identisch. Um allerdings ein gemeinsames Bild zu erhalten, müssen die Teilbilder, die von den Rendering-Servern berechnet wurden, über ein Netzwerk zurück an den Client übertragen werden. Je nach Verfahren muß der Client, bei einer Bildauflösung von 1024x768 Pixeln, pro Bild einen Datenstrom von ca. 2MB für RGB-Werte oder bis zu 5.5MB für RGB mit Tiefeninformation bewältigen. Je nach Netzwerkverbindung empfiehlt sich hier die Verwendung von Kompressionsverfahren. Da dieses Problem allen Image-Composition Algorithmen gemein ist, wurden verschiedene Bildübertragungsverfahren implementiert. Es können Inhalte des RGBA Buffers, des Z Buffers und des Stenci Buffers komprimiert oder unkomprimiert übertragen werden. Bei ersten Testläufen liefert beispielsweise eine JPEG-Komprimierung bei einer 100Mbit

Ethernet Verbindungen eine spürbare Performance-Steigerung. Bei einer 1Gbit Verbindung ist jedoch der Overhead der Kompression größer als die Zeitersparnis bei der Datenübertragung. Um auch bei schnellen Verbindungen eine Kompression sinnvoll einsetzen zu können, wurde eine schnelle Runlength-Kompression implementiert. Durch die getrennte Kompression der Farbkanäle kann mit diesem Verfahren häufig auch in texturierten Bildern eine leichte Kompression erreicht werden. Es werden sehr gute Kompressionsraten in Szenen mit einfarbigen Flächen oder bei einer nur teilweisen Überdeckung des Bildraums und einem monochromen Hintergrund erreicht.
Die eigentliche Image-Komposition besteht aus mehreren Teilaufgaben. Bei diesen Teilaufgaben werden verschiedene PC-Komponenten unterschiedlich stark ausgelastet. Die Netzwerkkommunikation belastet die CPU nur gering, während Lese- und Schreiboperationen der Graphik-Hardware nur einen geringen Einfluß auf die Netzwerkkommunikation haben. Folgende Teilaufgaben müssen ausgeführt werden.

- Bilddaten aus den Graphikkarten der Render-Server auslesen
- Bilddaten komprimieren
- Daten versenden
- Bilddaten entpacken
- Pixel in die Graphikkarte des Clients schreiben

Um eine optimale Systemauslastung zu erreichen, werden die zu übertragenden Images in Subimages zerlegt. Jedes Subimage wird getrennt ausgelesen, komprimiert, versendet, entpackt und in den Bildspeicher des Clients geschrieben.

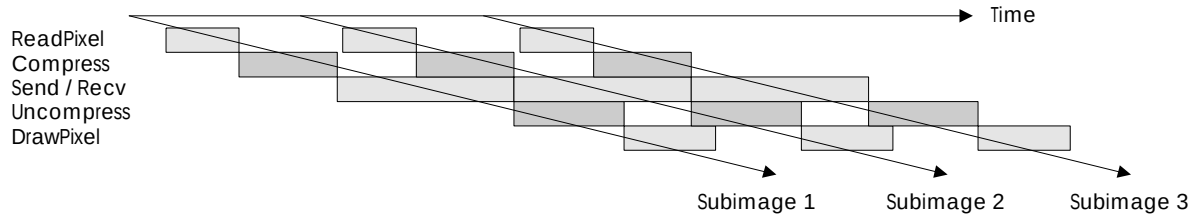

Abbildung 7 Image-Composition mit Subimages

Erste Versuche haben gezeigt, daß mit dieser Methode bei unkomprimierten Image-Daten in einem 100Mbit Ethernet mit einer Geschwindigkeit übertragen werden kann, die nur durch die Netzwerkbandbreite bestimmt wird. Lese- und Schreiboperationen in und aus der Graphik-Hardware werden also parallel zur Netzwerkübertragung ausgeführt.

Komplexe parallele Rendering-Verfahren

Im folgenden soll gezeigt werden, daß der gewählte Ansatz zur Integration paralleler Rendering-Verfahren für PC-Cluster in OpenSG auch geeignet ist, komplexere Konfigurationen abzubilden.
Es soll eine Pipeline aus Rendering-Servern aufgebaut werden, wobei jeder Server einen begrenzten Near-Far Bereich rendert. Das Ergebnis-Image soll an den Client gesendet werden. Der nachfolgende Server der Pipeline erhält eine Occlusion-Map seines Vorgängers. Eine Performance-Steigerung wird durch eine Verlängerung der Pipeline erreicht. Hierbei ist zu berücksichtigen, daß sich bei einer längeren Pipeline auch höhere Verzögerungszeiten ergeben. Anwendungen, die eine hohe Bildwiederholrate benötigen und gleichzeitig keine sehr kurzen Latenzen erfordern, sind für diesem Ansatz gut geeignet.
Dieses Verfahren wurde bisher noch nicht umgesetzt. Es soll an diesem Beispiel lediglich gezeigt werden, wie eine Integration in OpenSG erfolgen könnte. Zuerst muß ein neues Cluster-Window implementiert werden. Die Parameterisierung erfolgt über Felder in diesem neuen Window.

- MFString pipeServer Liste der Server in der Rendering-Pipeline
- SFReal32 near Vordere Clipping-Ebene
- SFInt32 occlusionGridSize Auflösung der Occlusion-Map

Der Client initialisiert das Fenster und führt eine Synchronisation mit dem ersten Server durch. Dann beginnt der erste Server mit seiner Verarbeitung.

- Berechne Far, so daß die folgenden Pipeline-Stufen gleichmäßig ausgelastet werden
- Rendere das Bild von Near bis Far
- Überschreibe Near mit Far
- Entferne den ersten Eintrag aus der Server-Liste
- Führe eine Synchronisation mit dem nächsten Server in der Server-Liste durch
- Übertrage die Occlusion-Map an den Folge-Server
- Übertrage das berechnete Bild an den Client

Der nachfolgende Server führt nun seinerseits die beschriebenen Verarbeitungsschritte aus.

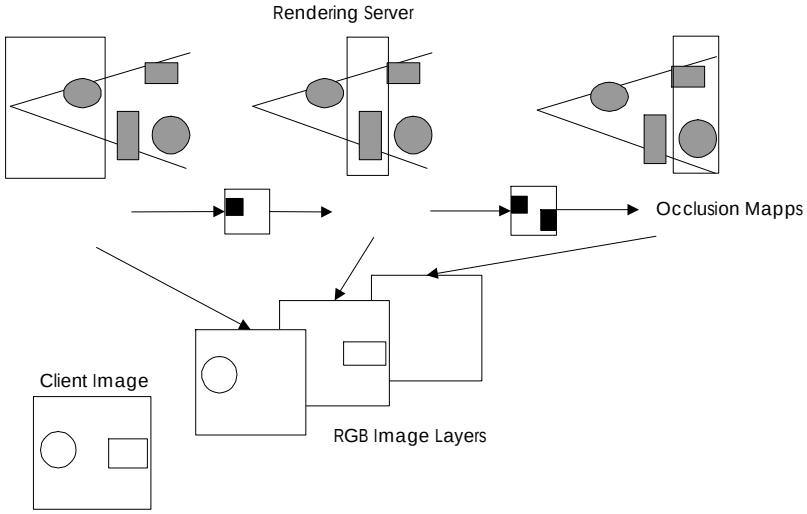

Abbildung 8 Occlusion-Pipe Rendering

Zusammenfassung

Es wurde die Integration von Synchronisationsmechanismen für lose gekoppelte Systeme in OpenSG präsentiert. OpenSG kann auf diese Weise eingesetzt werden, um in PC-Clustern parallel zu rendern. Die Integration in OpenSG erfolgte so, daß zukünftige Netzwerkhardware und neue parallele Rendering-Verfahren mit einem geringen Aufwand integriert werden können.
Bisher sind noch keine Mechanismen zur Lastverteilung implementiert. Die implementierten Verfahren zeigen lediglich die Verwendbarkeit des gewählten Ansatzes. Optimierte Verfahren und exakte Performance-Messungen sind Ziel zukünftiger Arbeiten.

Literaturverzeichnis

CCF01 H. Chen; Y. Chen; A. Finkelsetin; T. Funkhauser, K. Li, Z. Liu; R. Samanta; G. Wallace, Data Distribution Strategies for High-Resolution Displays, Computers & Graphics, Special Issue on Mixed Realities Beyond Conventions, October 2001

CF93 C. Cruz-Nera; D. Sandin;T. DeFanti, Surround-Screen Projection-Based Virtual Reality; The Design and Implementation of the CAVE, Computer Graphics SIGGRAPH Proceedings 1993

EIH00 M. Eldrige; H. Igeby; P. Hanrahan, Pomegranate: A Fully Scalable Graphics Architecture, Proceedings of SIGGRAPH 2000, page 443-454, Juli 2000

HEB01 G. Humphreys; M. Eldridge; I. Buck; G. Stoll; M. Everett; P. Hanrahen, WireGL: A Scalable Graphics System for Clusters, Computer Graphics (Proceedings of SIGGRAPH01) , August 2001

LCC00 Kai Li; Han Chen; Yuqun Chen; Duglas W Clark, Perry Cook; u.a., Building and Using A Scalable Display Wall System, IEEE Computer Graphics and Applications, Juli 2000

MCE94 S. Molnar; M.Cox D. Ellsworth; H. Fuchs, A Sorting Classification of Parallel Rendering, IEEE Computer Graphics and Applications, Vol 14, No 4, 4. Juli 1994, 23-32

MUE95 Carl Mueller, The Sort-First Rendering Architecture for High-Performance Graphics, Symposium on Interactive 3D Graphics 1995

NZ99 Thu D. Nguyen; John Zahorjan, Image Layer Decomposition for Distributed Rendering on NOWs, 1990

SFL00 R. Samanta; T. Funkhouser; K. Li; J. Sing, Hybrid Sort-First and Sort-Last parallel rendering with a Cluster of PCs, SIGGRAPH workshop on Graphics hardware, pages 99-108, ACM Press August 2000