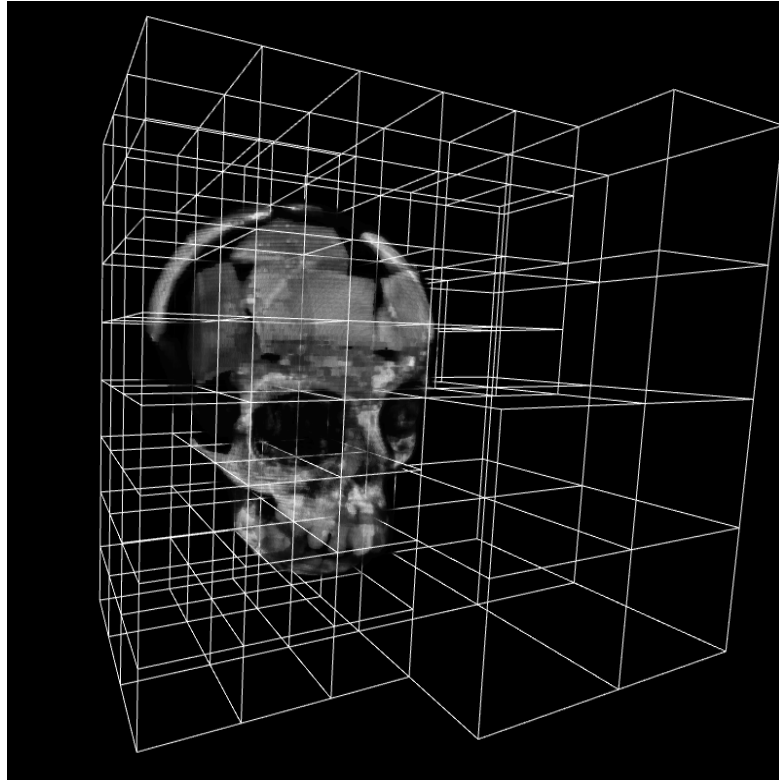


Fachhochschule Köln, Abteilung Gummersbach
Fachbereich Informatik, Studiengang Technische Informatik



**Ein hierarchischer Algorithmus zur Visualisierung von
sehr großen volumetrischen Datensätzen in Echtzeit**

Diplomarbeit von Michael Tirtasana
31. Mai 2000

- 1. Prüfer: Prof. Dr. Horst Stenzel**
- 2. Prüfer: Dr. Bernd Fröhlich**

Inhalt:

1	Einleitung.....	3
2	Andere und ähnliche Arbeiten.....	5
2.1	OpenGL Volumizer API	5
2.2	Ein anderes Octree-Darstellungsverfahren.....	7
3	Algorithmische Vorgehensweise.....	8
3.1	Volumendaten und Datenformate.....	8
3.2	Grundlagen zu 3D-Texturen.....	9
3.3	Hierarchische Datenrepräsentation	10
3.4	Darstellung des Algorithmus	11
3.4.1	Erzeugen des Octrees aus den Volumendaten	11
3.4.2	Einsortieren der Geometrie und Erzeugen einer Wunschliste	13
3.4.3	Optimieren der Wunschliste unter der Berücksichtigung des Zuladelimits.....	15
3.4.4	Zeichnen der Bricks.....	17
4	Die Implementierung des Algorithmus.....	18
4.1	Verarbeitung des Volumens.....	18
4.1.1	Laden des Originalvolumens	18
4.1.2	Zerlegung des Originalvolumens in Bricks	19
4.2	Erzeugen einer optimierten Wunschliste.....	21
4.2.1	Transformation der zu zeichnenden Dreiecke in das Volumenkoordinatensystem.	22
4.2.2	Erzeugen einer Wunschliste	22
4.2.3	Optimieren der Wunschliste	24
4.3	Dreieck Clipping	25
4.4	Zeichnen der optimierten Wunschliste	27
4.5	Implementierung der Software auf einem Multiprozessorsystem	29
4.6	Texturspeicherverwaltung von GigaVol.....	31
4.7	Kurze Beschreibung der Programmschnittstelle von GigaVol	32
5	Verschiedene Entwicklungstufen von GigaVol.....	33
6	Zeit und Geschwindigkeitsmessungen für das Volumenrendern.....	37
6.1	TexBench für OpenGL.....	37
6.2	Texturspeicherdurchsatz auf der ONYX2 mit InfiniteReality2 und der Wildcat 4110.	38
6.3	Textur-Fillrate auf der ONYX2 InfiniteReality2 und der Wildcat 4110.....	39
7	GigaVol Benchmarks.....	41
7.1	OpenGL Version von GigaVol	41
7.2	Avango-Version von GigaVol	44
8	Praktischer Einsatz von GigaVol im VRGeo Projekt	49
9	Zusammenfassung und Ausblick.....	50
9.1	Ergebnis der Arbeit	50
9.2	Mögliche Weiterentwicklungen von GigaVol.....	50
10	Literaturverzeichnis	51

1 Einleitung

In der traditionellen Computergrafik werden zur Beschreibung dreidimensionaler Modelle überwiegend Polygone verwendet. Diese polygonalen Modelle lassen sich auf verschiedene Weise erzeugen, wie zum Beispiel durch Modellierprogramme, CAD-Systeme oder dreidimensionale Scan-Techniken. Um ein realistisches Aussehen der Modelle zu erreichen, werden oft zweidimensionale Texturen eingesetzt. Texturen sind digitale Bilder, die sich wie Abziehbilder auf polygonale Modelle aufkleben lassen. Heutige Grafikhardware ist auf die Bearbeitung solcher polygonaler Szenen mit zweidimensionalen Texturen optimiert und erlaubt die Darstellung von mehreren Millionen Polygonen pro Sekunde.

Bildgebende Verfahren wie die Computer-Tomographie (CT) im Bereich der Medizin, Reflexionsseismikmessungen im Bereich der Erdölindustrie oder auch Computersimulationen von z.B. Strömungssimulationen liefern oft hochaufgelöste, volumetrische Daten, die im Bereich mehrerer hundert Megabyte bis in die Terabyte liegen können. Die gelieferten Datenmengen sind im allgemeinen als dreidimensionale Matrix strukturiert, in der die gemessenen Werte eingetragen sind. Jeder Wert stellt einen Abtastwert in einem kleinen Raumbereich dar. Diese Einträge in die Matrix werden auch als Voxels bezeichnet.

Die Visualisierung dieser Daten kann durch verschiedene Verfahren erfolgen. Eine weit verbreitete Methode sind Isoflächen, die typischerweise durch Polygonnetze repräsentiert werden. Diese Polygonnetze sind für hochaufgelöste Volumendaten oft sehr umfangreich und lassen sich dann nicht mehr in Echtzeit darstellen. Eine einfache und oft verwendete Visualisierungstechnik sind Schnittebenen durch das Volumen, die allerdings nur einen zweidimensionalen Ausschnitt des Volumens darstellen. Wesentlich aufwendiger sind Volumenrendering-Verfahren, die eine halbtransparente, dreidimensionale Darstellung der Daten erlauben. Besonders interessant ist die Echtzeit-Visualisierung von Volumendaten, die mehrere Schnitt- oder Volumenrendering-Bilder pro Sekunde generiert und somit ein interaktives Untersuchen der Datensätze erlaubt. Diese Echtzeitdarstellung erfolgt heute meist mit Unterstützung spezieller Grafikhardware, die über einen Speicher für dreidimensionale Volumendaten, sogenannte 3D-Texturen, verfügen. Die Speichergröße für 3D-Texturen liegt typischerweise im Bereich zwischen 1MB im Low-End- und 64MB im High-End-Bereich. Bei einer Auflösung von 8 Bit pro Voxel ist man damit auf eine max. Auflösung von $512 \times 512 \times 256$ Voxels bei 64MB Texturspeicher beschränkt. Größere Volumen lassen sich damit nicht oder nur mit sehr langsamen Bildwiederholraten darstellen.

Diese Diplomarbeit beschreibt die Entwicklung und Realisierung eines Verfahrens, welches auf existierenden Grafikkarten mit 3D-Texturunterstützung eine Darstellung von mehreren Gigabyte großen Volumen in Echtzeit ermöglicht. Interaktive Anwendungen erfordern eine solche Echtzeitdarstellung, bei der mit hohen Bildwiederholraten von möglichst mehr als 8Hz gearbeitet werden sollte. Für die Berechnung eines Bildes stehen somit nur maximal 125 Millisekunden zur Verfügung. Das hier vorgestellte Verfahren unterteilt den Texturspeicher in eine Menge gleich großer Blöcke, sogenannte Bricks, die ein Teilvolumen einer bestimmten Größe (z.B. $64 \times 64 \times 64$ Voxel) aufnehmen können. Die darzustellenden Volumendaten werden in einem Präprozeß in ebensolche Bricks unterteilt. Bei einem Volumen von $1024 \times 1024 \times 1024$ Voxel erhält man somit zum Beispiel $16 \times 16 \times 16$ Bricks, die jeder $64 \times 64 \times 64$ Voxels enthalten. Zusätzlich werden immer 8 benachbarte Bricks zusammengefaßt und daraus ein größerer Brick berechnet. Dieses Verfahren wird dann hierarchisch fortgesetzt und die Bricks in den verschiedenen Auflösungsstufen in einem Octree, einer Baumstruktur mit 8 Kindern pro Knoten, gespeichert. Das Darstellungsverfahren arbeitet nach ähnlichen Prinzipien wie ein virtuelles Speichersystem. Zur Laufzeit wird entschieden, welche Bricks für die Darstellung

benötigt werden. Dabei muß berücksichtigt werden, daß nur eine gewisse Anzahl Bricks in den Texturspeicher paßt und außerdem pro Bild nur eine gewisse Anzahl Bricks nachgeladen werden sollte, da sonst die Echtzeit nicht mehr garantiert werden kann. Das hier entwickelte Verfahren erlaubt es nun den Speicher- und Zeitbedarf pro Bild gegen die Auflösung der Daten zu handeln, da diese in verschiedenen Auflösungen in dem Octree vorhanden sind. So führt zum Beispiel die schnelle Bewegung einer Schnittebene durch einen hochaufgelösten Datensatz dazu, daß während der Bewegung die Volumendaten nur in relativ grober Auflösung zu sehen sind, und sobald man die Schnittebene anhält, die nur durch den Texturspeicher begrenzte maximale Auflösung zu sehen ist. Diese Vorgehensweise ist gut vergleichbar mit der allgemein bekannten Bewegungsunschärfe und ist somit nicht sehr störend.

Das hier vorgestellte Verfahren erlaubt die Darstellung eines 1GB großes Volumen mit einer Auflösung von 1024x1024x1024 auf einer High-End-Grafikworkstation mit 2GB Hauptspeicher und 64MB Texturspeicher mit einer Bildwiederholrate von 8Hz bis 15Hz. Damit lassen sich diese großen Volumendatensätze sehr gut interaktiv untersuchen und bearbeiten.

Die Diplomarbeit wurde in der GMD – Forschungszentrum Informationstechnik GmbH erstellt.

2 Andere und ähnliche Arbeiten

Bis zum Beginn dieser Diplomarbeit gab es folgende Möglichkeiten für eine computer-gestützte Volumendarstellung.

2.1 OpenGL Volumizer API

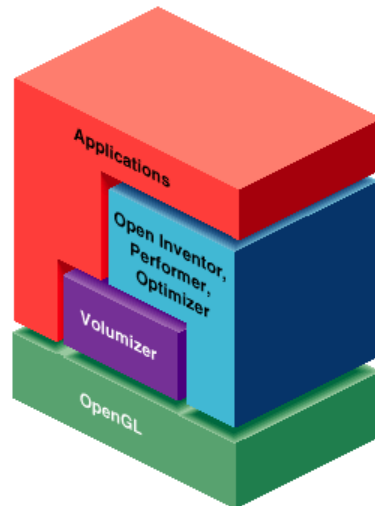


Abb. 2-1 Volumizer Struktur.

Die OpenGL Volumizer API [5] ist eine Bibliothek von C++ Klassen, die eine Darstellung und Manipulation von volumetrischen Daten erlaubt.

Volumizer verspricht folgende Vorteile für diese Aufgabe:

- hohe Geschwindigkeit
- einfach zu handhabende API
- Multipipe Funktionalität
- Darstellung großer Daten

Volumizer zerlegt Volumendaten immer in eine Menge von Tetraedern (Abb. 2-1).

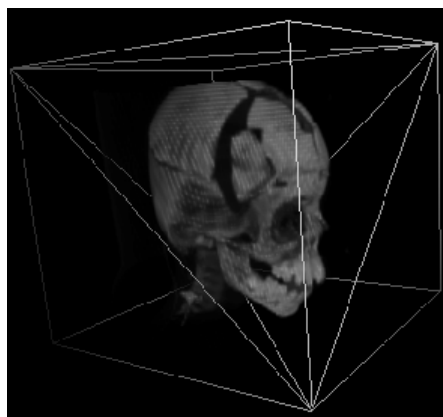


Abb. 2-2 Tetraederzerlegung.

Volumizer nutzt auch die Möglichkeit, das Volumen bei der Repräsentation in mehrere Bricks zu unterteilen. Dies bietet Volumizer die Möglichkeit durch Paging (Aus- und Einlagern von Texturen) mehr Volumen darzustellen als in den Texturspeicher paßt. Allerdings werden die Bricks nur in einer Auflösungsstufe gespeichert.

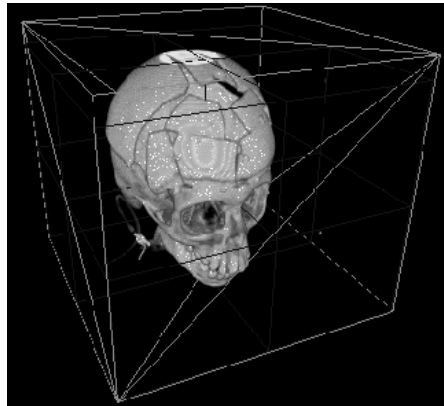


Abb. 2-3 Eine CT Volumendarstellung.

Volumizer übernimmt unter anderem folgende Aufgaben:

- die Verwaltung der Bricks und des Texturspeichers.
- die Berechnung und Erzeugung der Dreiecke.
- die Ausrichtung und Sortierung der Dreiecke.
- die Verwendung der besten technischen Darstellungstechnik auf der verwendeten Grafikhardware.

Gründe, die gegen die Verwendung von Volumizer sprachen:

- Tests haben ergeben, daß die Darstellung von Volumen, die größer als der Texturspeicher sind, nicht mehr in Echtzeit durch Volumizer möglich ist. Auf einer Grafikhardware mit 64MB Texturspeicher war es nicht möglich mit hoher Geschwindigkeit ein 128MB großes Volumen darzustellen. Die Hardware muß für jede einzelne Bildberechnung das Volumen in den Texturspeicher ein- und auslagern, was sehr viel Zeit benötigt.
- Wurde ein Volumen durch Volumizer in kleinere Bricks unterteilt, führte das zu einem hohen Geschwindigkeitsverlust, obwohl die Texturen in den Texturspeicher paßten.
- Die Software ist nicht als Open Source verfügbar, so daß spezielle Anpassungen an Multiprozessor-Systeme nur begrenzt möglich ist. So war es sehr schwierig, eine Anpassung an Performer durchzuführen, welches für den Multipipe-Modus eine Verwendung von Shared Memory benötigt.
- Eine gute Ausnutzung von Multiprozessor-Systemen ist nur schwierig mit einer vorgegebenen API realisierbar.
- Volumizer bietet von sich aus keine Möglichkeit, ein Volumen in mehreren Auflösungsstufen zu speichern. Dies würde man für eine hierarchische Datenrepräsentation benötigen.

2.2 Ein anderes Octree-Darstellungsverfahren.

Parallel zu den hier durchgeführten Entwicklungen wurde von Eric LaMar, Bernd Hamann und Kenneth I. Joy an der Universität Davis, USA [8], auch ein Octree-Darstellungsverfahren entwickelt. In dem dort entwickelten Verfahren wird der Volumendatensatz ebenfalls in mehreren Auflösungsstufen repräsentiert. Der Unterschied zwischen den beiden Octree-Darstellungen ist, daß das Verfahren aus den USA nicht das Durchwandern großer Volumendatensätze in Echtzeit unterstützt. Statt dessen wird versucht, durch neue Methoden die Bildqualität des Volumenrenderings zu verbessern. Texturbasiertes Volumenrendering benötigt immer eine Geometrie, auf die die Textur projiziert werden kann. So werden dort statt der üblichen Schnittebenen als Geometrie sogenannte „spherical shells“ (Abb. 2-4) benutzt. Die „spherical shells“ sind in jedem Punkt zum Betrachter ausgerichtet. Sie vermeiden visuelle Artefakte, die bei der Darstellung durch Ebenen auftreten können. Außerdem bietet das Verfahren die Möglichkeit, die für den Anwender interessanten Bereiche innerhalb des verwendeten Volumens festzulegen (region-of-interest). Diese Bereiche werden bei der Darstellung mit höherer Auflösung gezeichnet. Ein Nachteil bei dem Verfahren ist die hohe Bildberechnungszeit, die bei der Verwendung dieser Techniken entsteht. Die Zeit betrug bis zu mehreren Sekunden, so daß das Verfahren die Echtzeitanforderungen nicht erfüllen kann.

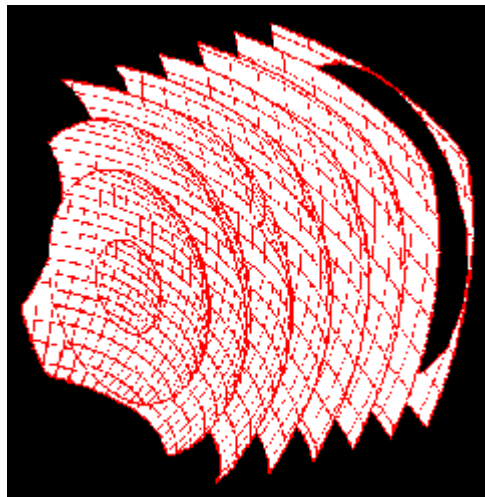


Abb. 2-4 „Spherical shells“ als Geometrie

3 Algorithmische Vorgehensweise

In diesem Kapitel werden wichtige Begriffe erklärt und die Grundlagen der Algorithmen dargestellt.

3.1 Volumendaten und Datenformate

Volumendaten sind Datenmengen, die in einer dreidimensionalen Matrix strukturiert sind, deren Einträge die Dateninformation repräsentieren. Jeder Eintrag stellt einen Abtastwert in einem kleinen Raumbereich dar. Diese Werte können Farbinformationen, Materialdichte, Reflexionskoeffizienten oder andere Eigenschaften repräsentieren. Diese Einträge in der Matrix werden auch als Voxels bezeichnet.

Dateiformate, die häufig für Volumendaten verwendet werden:

- *Bilderserien*
Oft werden zweidimensionale Bilder als Volumendatenspeicher verwendet. Diese werden meistens durchnummeriert und nacheinander in den Speicher geladen. Im Speicher werden diese Bilder in einem dreidimensionalen Datensatz zusammengeführt.
- *Das dreidimensionale TIFF Datenformat*
Das TIFF (Tag Image File Format) speichert normalerweise zweidimensionale Bilder. Dieses Format wurde von SGI so erweitert, daß auch eine Serie von Bildern gespeichert werden kann. Diese Serie kann auch ein Volumen repräsentieren.
- *Das Voxelgeo Volumen Format*
Voxelgeo ist ein Programm, welches von der Erdölindustrie für die Analyse von geologischen Daten verwendet wird. Voxelgeo verarbeitet das SEG Y Format und speichert die Ergebnisse als Polygonnetze oder als Volumendaten in einem eigenen Format (Extension .vol). Dieses Format besteht aus Interpretationen von seismischen Daten, die in 8-Bit-Auflösung in einer dreidimensionalen Matrix gespeichert sind.
- *Das SEG Y Format*
Dieses Format ist als „standard magnetic tape“ von der „Society of Exploration Geophysicists“ definiert. In diesem Format werden Reflexionsmessungen von geologischen Erdschichten in 32-Bit-Auflösung gespeichert. Solche Daten werden zum Beispiel von Schiffen aus gesammelt. Druckwellen, die im Wasser von einer Explosion ausgelöst werden, werden im Meeresgrund unterschiedlich reflektiert. Diese Reflexionen werden gemessen, bearbeitet und dann oft im SEG Y Format gespeichert (Abb. 3-1).

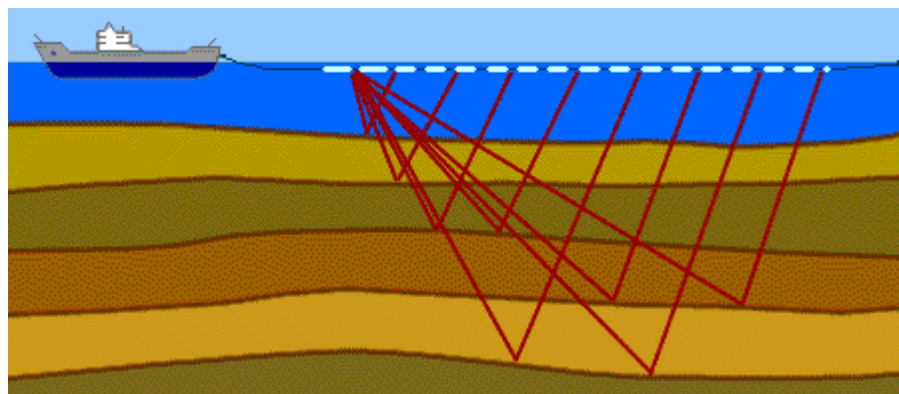


Abb. 3-1 Druckwellen werden von den Erdschichten unterschiedlich reflektiert [7]

3.2 Grundlagen zu 3D-Texturen

Eine der wichtigsten Techniken für Echtzeit-Visualisierung von Volumen sind 3D-Texturen [6]. 3D-Texturen sind den normalen 2D-Texturen sehr ähnlich. Der Unterschied besteht darin, daß als Texturquelle nicht eine 2D-Matrix (Bild), sondern eine 3D-Matrix (Volumen) dient. Entsprechend benötigt man zur Definition einer 3D-Textur nicht nur 2 Texturkoordinaten wie bei den 2D-Texturen, sondern 3 Texturkoordinaten.

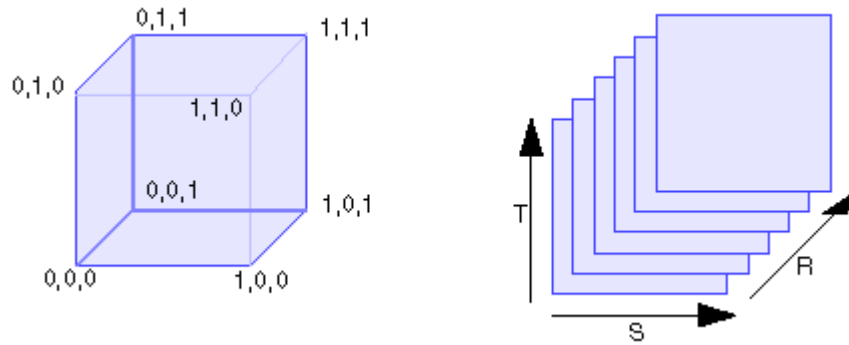


Abb. 3-2 Eine 3D-Textur kann als ein Stapel aus 2D-Texturen betrachtet werden.

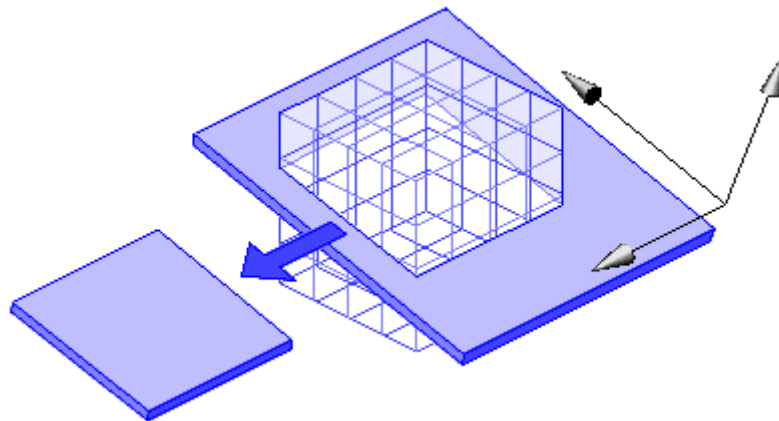


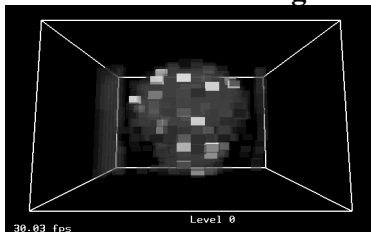
Abb. 3-3 Beliebige Schnitte durch ein 3D Volumen sind mit 3D-Texturen möglich.

3.3 Hierarchische Datenrepräsentation

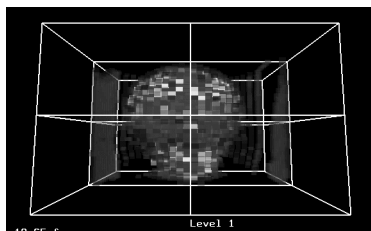
Die Grundlage des Algorithmus ist ein Octree, in dem die Volumendaten in verschiedenen Auflösungen hierarchisch gespeichert werden. Die Knoten des Octree entsprechen einem quaderförmigen Ausschnitt des Volumens, einem sogenannten Brick. Jeder Brick enthält eine vom Benutzer festgelegte Anzahl Voxel. Alle Bricks einer Ebene haben dieselbe Größe und repräsentieren gemeinsam das gesamte Volumen. Die vom Benutzer festgelegte Anzahl der Ebenen im Octree beeinflusst dessen Größe und bestimmt die Anzahl der Bricks, die aus dem Volumen erzeugt werden. Der Octree wird von der Ebene 0 bis zur festgelegten niedrigsten Ebene erzeugt. Die höchste Ebene, die Ebene 0, besteht aus nur einem Brick. Durch die nachfolgenden Ebenen kommen immer acht mal so viele Bricks hinzu, wie die jeweils übergeordnete Ebene enthält. Die Gesamtanzahl der Bricks läßt sich folgendermaßen berechnen:

$$\text{Gesamtanzahl der Bricks} = \sum_{n=0}^{\text{niedrigste Ebene}} 8^n$$

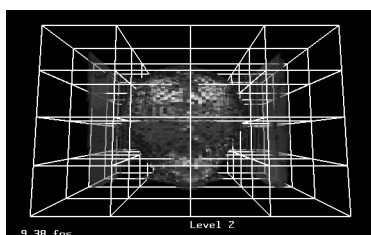
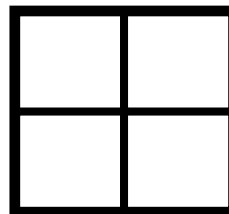
Dreidimensionale Darstellung des Octrees: Zweidimensionale Darstellung des Octrees:



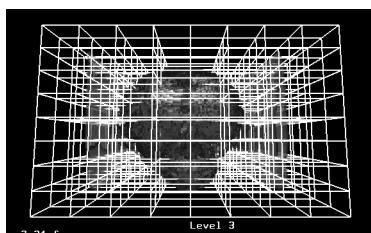
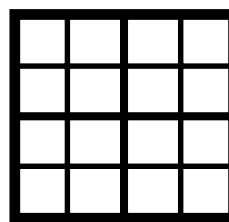
Ebene 0: Dieser Brick hat die Dimensionen 16x16x16 und stellt die Wurzel des Octrees dar.



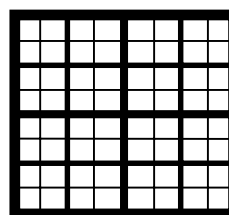
Ebene 1: Enthält 8 Bricks und hat die Gesamtauflösung 32x32x32



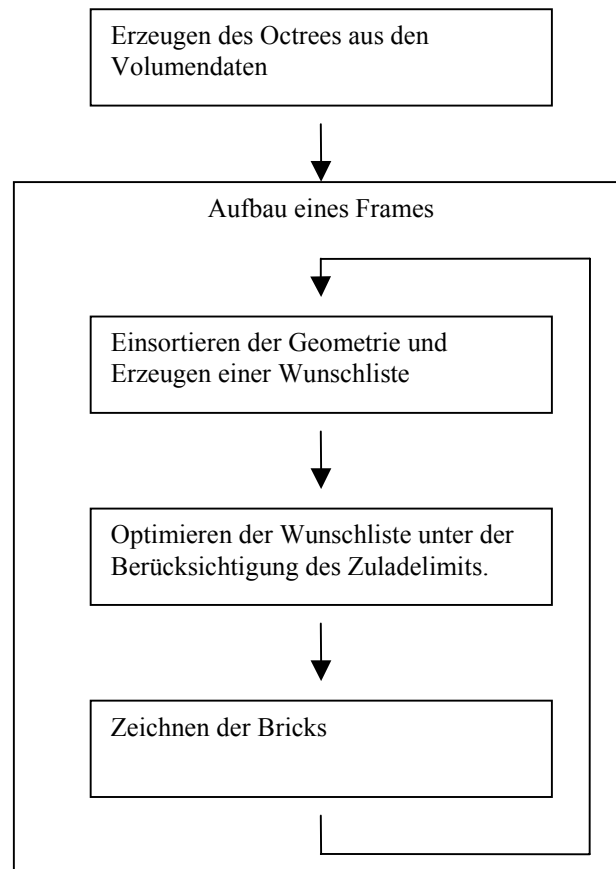
Ebene 2: Enthält 64 Bricks und hat die Gesamtauflösung 64x64x64



Ebene 3: Enthält 512 Bricks und hat die Gesamtauflösung 128x128x128



3.4 Darstellung des Algorithmus



In diesem Unterkapitel wird der grundsätzliche Algorithmus dargestellt. Er besteht aus der einmaligen Erzeugung des Octrees und den verschiedenen Berechnungen, die für jedes Bild nötig sind.

3.4.1 Erzeugen des Octrees aus den Volumendaten

Die vom Nutzer ausgewählten Volumendaten werden in den Hauptspeicher geladen und in einem internen Format gespeichert. Die Volumendaten können als Grauwertinformationen (8bit) oder Farbinformationen (RGBA 32bit) abgelegt werden. Der Brick in der höchsten Ebene erstreckt sich über die gesamte räumliche Ausdehnung des originalen Volumendatensatzes. Die Auflösung des Bricks entspricht den festgelegten Dimensionen, die für alle Bricks gleich sind. Die Bricks der nachfolgenden Ebenen erstrecken sich jeweils über einen entsprechenden Teil des Originalvolumens. Für die Erzeugung der Voxeldaten für die einzelnen Bricks wird ein Filteralgorithmus eingesetzt. Dieser verwendet zur Berechnung eines neuen Wertes einen oder mehrere Einträge aus dem originalen Datensatz. Da jede Ebene achtmal mehr Daten enthält als die Darüberliegende, erhöht sich jeweils die maximale verfügbare Gesamtauflösung mit jeder neuen Ebene um diesen Faktor. Somit verbessert sich die Annäherung an den originalen Volumendatensatz mit jeder weiteren Ebene im Octree. Insgesamt bestimmen also die Anzahl der Ebenen im Octree und die Auflösung der einzelnen Bricks, welche Auflösungsstufen verfügbar sind und inwieweit das Originalvolumen angenähert werden kann.

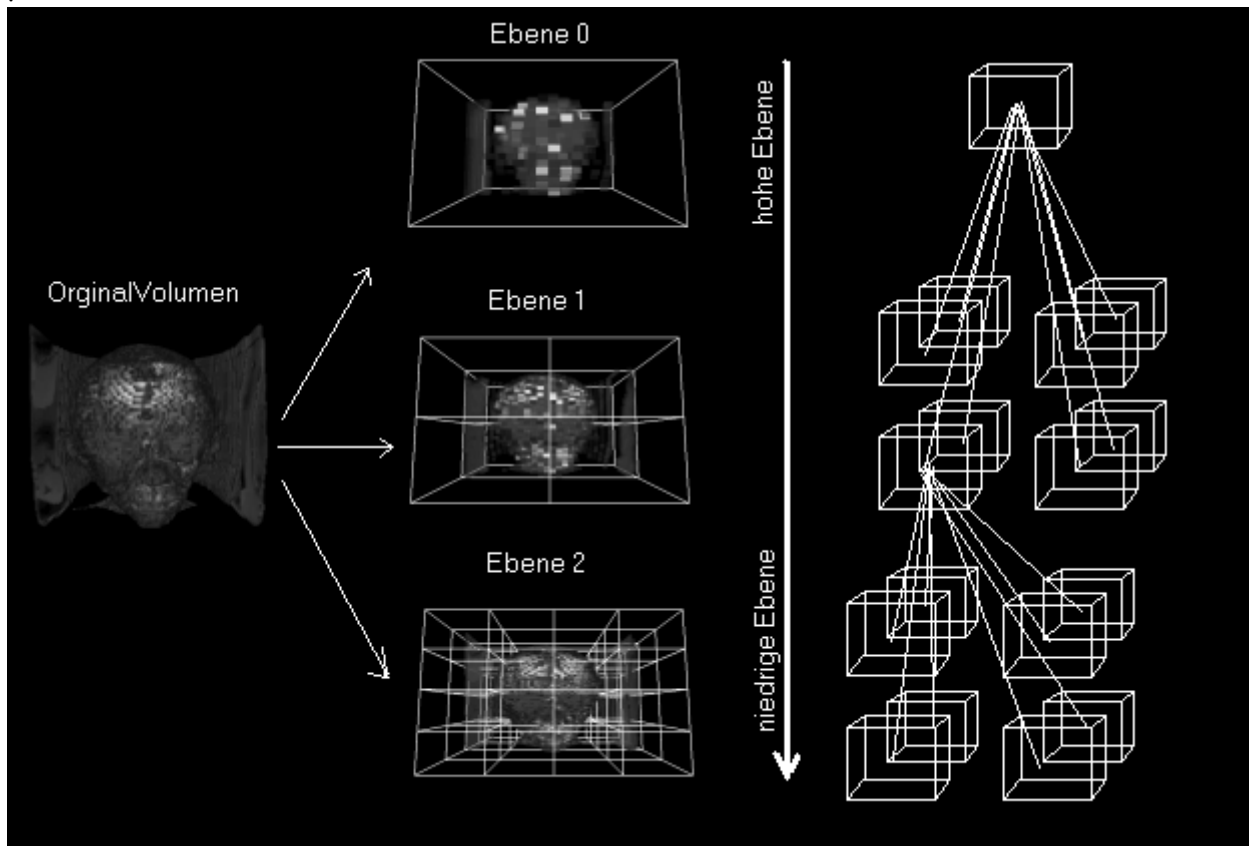


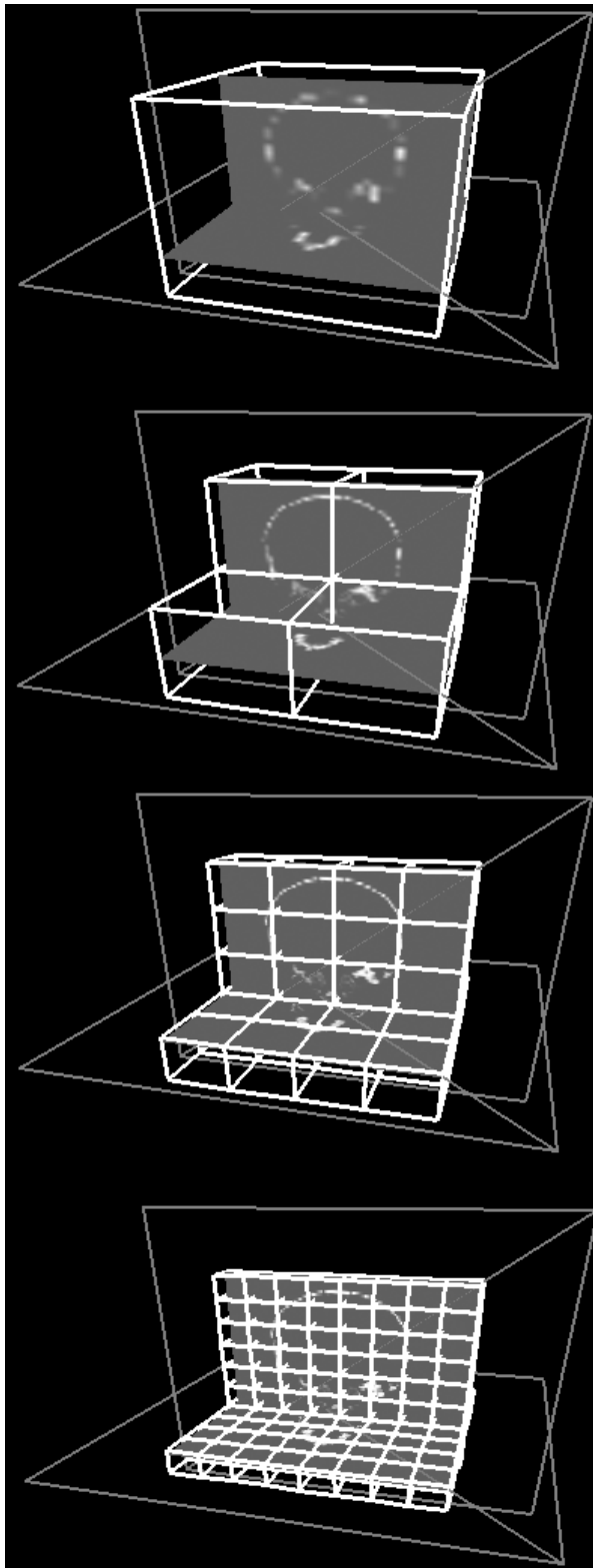
Abb. 3-4 Hier wurde ein Octree mit 3 Ebenen erzeugt, wobei die Anzahl der benötigten Bricks 73 beträgt.

3.4.2 Einsortieren der Geometrie und Erzeugen einer Wunschliste

Texturbasierte Verfahren zur Darstellung von Volumen verwenden dreidimensionale Geometrien, auch als Proxy-Geometrien bezeichnet, auf die die Textur aufgebracht wird. Für das hier entwickelte Verfahren wird angenommen, daß sich die Proxy-Geometrien aus Dreiecken zusammensetzen. Im einfachsten Fall handelt es sich dabei um eine rechteckige Schnittebene, die aus zwei Dreiecken besteht. Zur semi-transparenten Darstellung von Volumen werden Stapel hintereinanderliegender Rechtecke als Proxy-Geometrien eingesetzt, die dann von hinten nach vorne dargestellt werden.

Die Volumendaten sind als 3D-Textur in den Bricks des Octree gespeichert. Die Proxy-Geometrien werden hierarchisch in den Octree einsortiert und eine Liste der Bricks erstellt, die von den Proxy-Geometrien geschnitten und somit für die Darstellung benötigt werden. Diese Liste wird hier als Wunschliste bezeichnet, da sie eine optimale Auflösung der Volumendaten auf den Proxy-Geometrien unter Berücksichtigung der Texturspeichergröße garantiert.

Zum Erstellen der Wunschliste wird für die Dreiecke der Proxy-Geometrien festgestellt, ob sie mit einem Brick eine räumliche Überlappung haben. Zuerst wird jedes Dreieck daraufhin getestet, ob es den „Wurzel-Brick“ in der Ebene 0 schneidet. Die den Brick schneidenden Dreiecke werden als Liste innerhalb des Bricks gespeichert und dieser dann in die Wunschliste eingefügt. In weiteren Verfahrensschritten wird die Wunschliste „entfaltet“. Das heißt, daß in der Wunschliste ein Brick durch alle seine „Kinder-Bricks“ ersetzt wird, die mindestens ein Dreieck schneiden. Dabei läßt sich ausnutzen, daß ein Kinder-Brick nur die Dreiecke schneiden kann, die auch seinen Vater schneiden. Dabei werden die Kinder-Bricks abhängig von der Entfernung zum Betrachter sortiert und in dieser Reihenfolge in die Wunschliste eingefügt. Somit ist die Wunschliste immer in Abhängigkeit der Entfernung zum Betrachter sortiert. Die Bricks werden gemäß ihrer Reihenfolge entfaltet, damit die Bricks mit der geringsten Entfernung zum Betrachter zuerst verfeinert werden. Die höchste Auflösung und Qualität haben dadurch die Bricks, die dem Betrachter am nächsten sind und am größten dargestellt werden. Die Wunschliste wird bis zur niedrigsten vorhandenen Ebene entfaltet, es sei denn, die durch den Texturspeicher limitierte Anzahl der Bricks ist vorher erreicht. Die Wunschliste muß für jeden Frame neu erzeugt werden, da die Positionen der Dreiecke und des Betrachters sich typischerweise kontinuierlich ändern und somit die Struktur der Wunschliste beeinflussen.



Vier Dreiecke, die zu einer vertikalen und einer horizontalen Schnittebene gehören, schneiden den Brick in der Ebene 0. Diese vier Dreiecke werden innerhalb dieses Bricks gespeichert. Der Brick wird in der Wunschliste gespeichert.

Die Dreiecke des Vater-Bricks werden gegen seine acht Kinder geprüft. Zwei der acht Bricks in Ebene 1 werden nicht mehr von Dreiecken geschnitten. Die jeweiligen Kinder-Bricks müssen nicht mehr im Octree weiter verarbeitet werden. Die Wunschliste besteht jetzt aus sechs Bricks.

In der Ebene 2 besteht die Wunschliste aus 28 Bricks. 36 Bricks in dieser Ebene müssen nicht verarbeitet werden.

In der Ebene 3 werden 120 Bricks von Dreiecken geschnitten. Es werden nur 24% der Bricks für die Darstellung der Dreiecke benötigt. Die endgültige Wunschliste besteht aus 120 Bricks.

Abb. 3-5 Erzeugung einer Wunschliste bis zur 3. Ebene

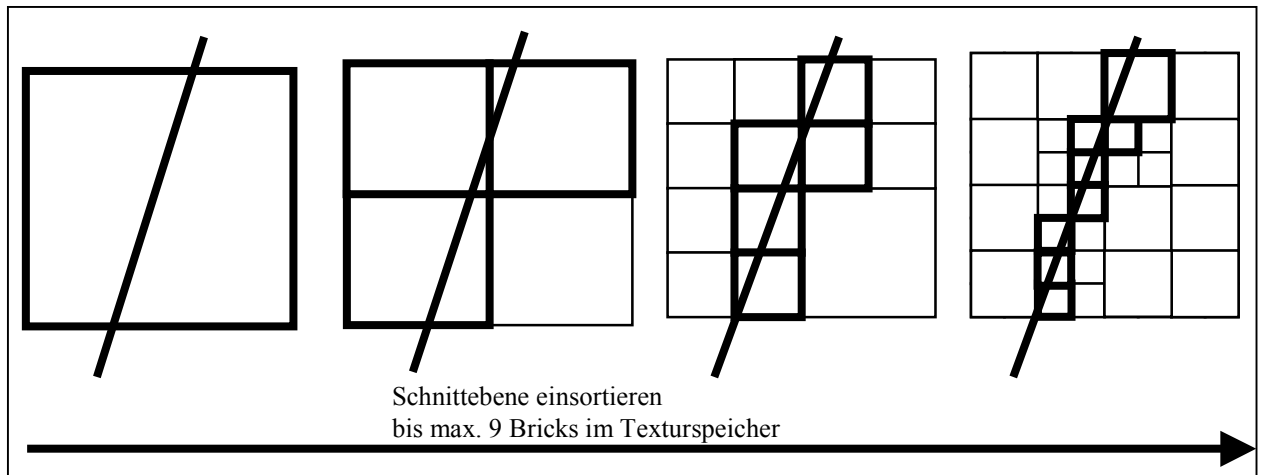


Abb. 3-6 Zweidimensionale Betrachtung der Erzeugung einer Wunschliste mit maximal 9 Bricks

3.4.3 Optimieren der Wunschliste unter der Berücksichtigung des Zuladelimits.

Die Wunschliste nutzt den Texturspeicher ideal aus. Die enthaltenen Bricks können aber aus Geschwindigkeitsgründen nicht immer in der geforderten Zeit in den Texturspeicher geladen werden, auch wenn man die bereits im Texturspeicher vorhandenen Bricks berücksichtigt. Dieser Fall tritt insbesondere dann auf, wenn man die Proxy-Geometrie innerhalb des Volumens schnell bewegt.

Daher wird nur eine begrenzte Anzahl neuer Bricks zugelassen, die pro Frame in den Texturspeicher geladen werden. Solch ein Zuladelimit begrenzt die Zeit, die pro Frame für das Nachladen der Textur benötigt wird und erlaubt konstante Frameraten. Es kann aber verursachen, daß die der Wunschliste entsprechende Auflösung erst nach einer bestimmten Anzahl gezeichneter Frames dargestellt werden kann, auch wenn sie sich nicht verändert. Das Zuladelimit muß mindestens acht Bricks betragen, da die Möglichkeit besteht, daß sich ein Vater-Brick im Octree zu acht Kinder-Bricks entfaltet. Ein Zuladelimit von weniger als acht Bricks hätte in einem ungünstigen Fall zur Folge, daß kein Brick mehr entfaltet und die Wunschliste nie erreicht werden kann.

Das Verfahren geht nun so vor, daß es die bestehende Wunschliste so lange bezüglich der Anzahl der enthaltenen Bricks verkleinert, bis das Zuladelimit unterschritten wird. Dies geschieht, indem die vorhandenen Kinder-Bricks eines gemeinsamen Vaters wieder zu dem Vater-Brick zusammengefaßt werden. Die Kinder-Bricks, die am weitesten entfernt sind, werden als erstes zusammengefaßt, da eine reduzierte Auflösung in größerer Entfernung weniger auffällt.

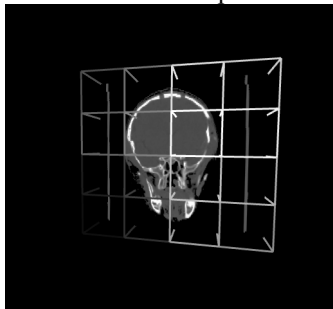
Beispiel:

Optimieren der Wunschliste bei gleichbleibender Schnittebene zweidimensional betrachtet.

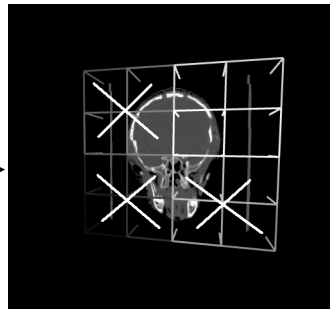
- Die Octreestruktur der Wunschliste besteht aus 16 Bricks, die in den Texturspeicher passen.
- Das Zuladelimit beträgt acht Bricks.
- Es sind noch keine Bricks im Texturspeicher.
- Die heller gekennzeichneten Bricks sind näher am Betrachter.

Frame 1:

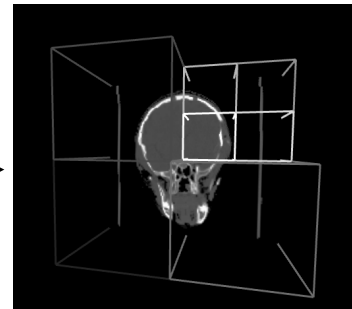
0 Bricks im Texturspeicher



Wunschstruktur



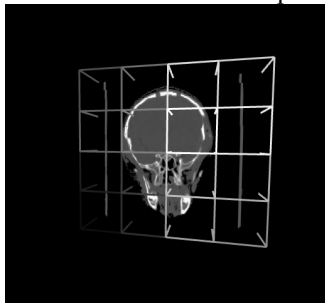
Die 3x4 am weitesten entfernten Bricks werden zusammengefaßt.



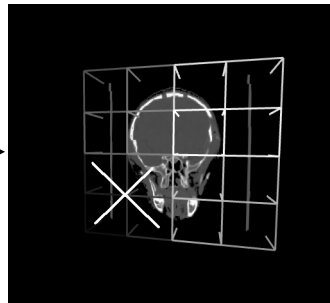
**Sieben neue Bricks im Texturspeicher.
Zuladelimit erfüllt.**

Frame 2:

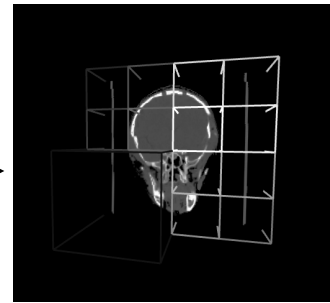
Sieben Bricks im Texturspeicher



Wunschstruktur



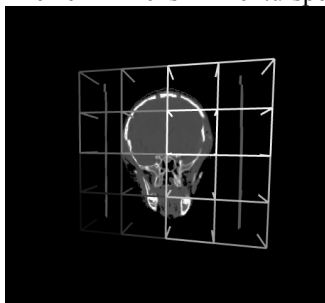
Die 1x4 am weitesten entfernten Bricks werden zusammengefaßt



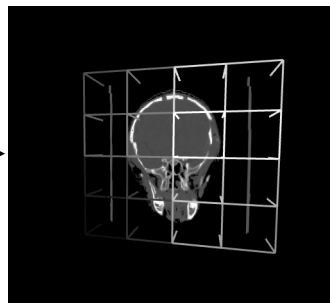
**Acht neue Bricks im Texturspeicher.
Zuladelimit erfüllt.**

Frame 3:

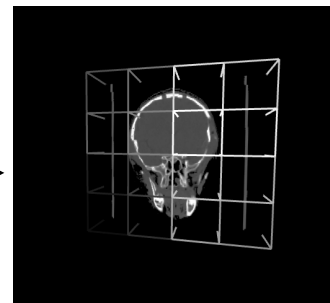
Dreizehn Bricks im Texturspeicher



Wunschstruktur



Es müssen keine Bricks zusammengefaßt werden.



**Vier neue Bricks im Texturspeicher.
Zuladelimit erfüllt.**

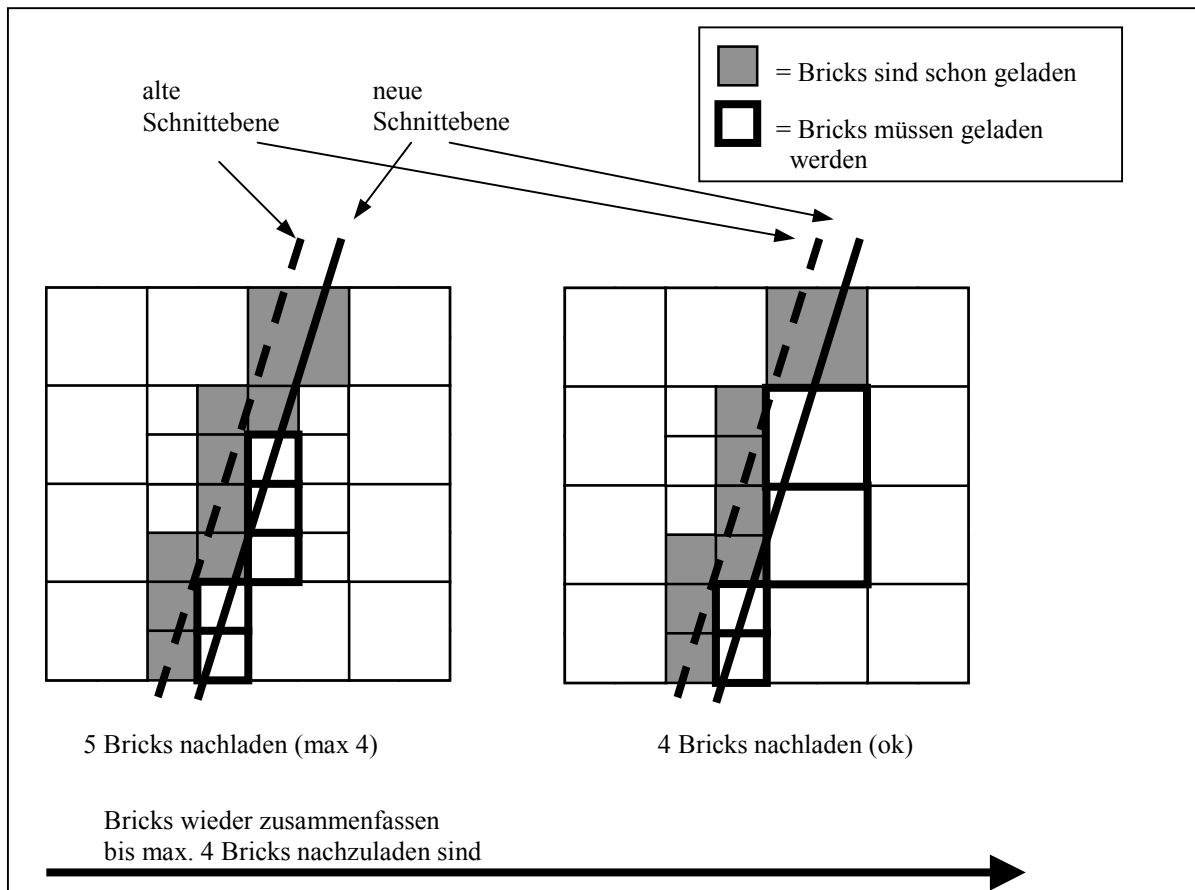


Abb. 3-7 Zweidimensionale Betrachtung der Optimierung der Wunschliste bei verschobener Schnittebene

3.4.4 Zeichnen der Bricks

Die Bricks müssen in der Reihenfolge von hinten nach vorne zum Betrachter gezeichnet werden. Dies ist notwendig, da sich in den Bricks auch transparente Dreiecke befinden können, die immer von hinten nach vorne gezeichnet werden müssen um eine korrekte Darstellung zu gewährleisten. Ein Sortierung der Bricks ist hier unnötig, da sie schon sortiert in der Wunschliste vorliegen. Da auch die Dreiecke innerhalb eines Bricks von hinten nach vorne gezeichnet werden müssen, verlangt der Algorithmus, daß die Dreiecke in der richtigen Reihenfolge übergeben werden.

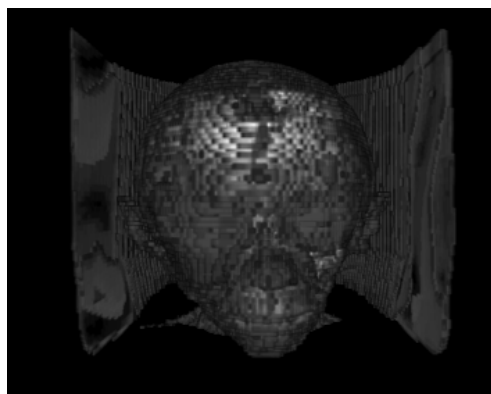
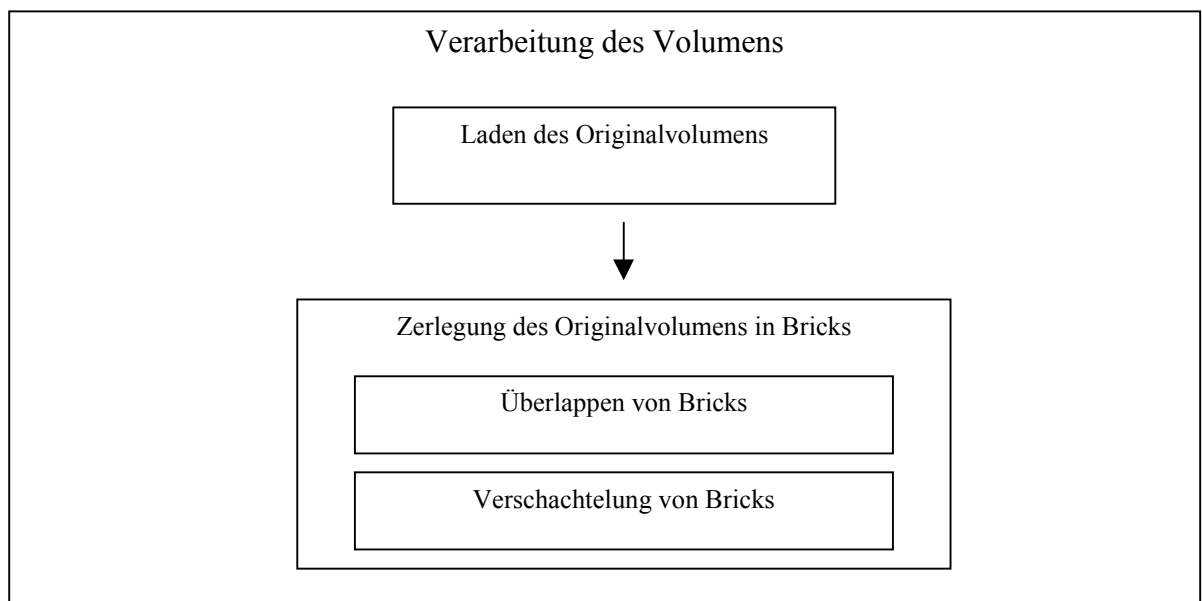


Abb. 3-8 Eine Volumendarstellung einer Computertomographie

4 Die Implementierung des Algorithmus

Der Algorithmus wurde hauptsächlich auf einer SGI ONYX2 [9] entwickelt, da die ONYX 3D-Texturen in der benötigten Leistung darstellen kann. Die Software wurde GigaVol genannt, da sie Volumen, die größer als ein Gigabyte sind, darstellen kann. GigaVol wurde möglichst plattformunabhängig entwickelt und kann auf Rechnern eingesetzt werden, die 3D-Texturen unter OpenGL [1] darstellen können. Neben einer OpenGL Version für IRIX und einer für Windows PC's, gibt es noch eine Version für das VR System Avango [2], welches Multiprozessor-Systeme unterstützt. Avango ist eine Software, die darauf spezialisiert ist, virtuelle Welten auf verschiedenen VR-Projektionssystemen darzustellen. Dies können stereographische Systeme wie eine Cave oder eine Workbench sein. Die Verwaltung der 3D-Objekte geschieht in einem Szenengraphen.

4.1 Verarbeitung des Volumens



4.1.1 Laden des Originalvolumens

Um die Bricks erzeugen zu können, wird das Originalvolumen vollständig in den Speicher geladen. GigaVol bietet die Möglichkeit, folgende Formate zu laden:

- Voxelgeo *.vol Format
- Dreidimensionales TIFF
- Bilder oder Bilderserien im SGI RGBA Format
- Das medizinische *.lat Format

Des weiteren besteht auch die Möglichkeit, vorgerechnete Bricks im eigenen bst-Format zu laden. Damit entfällt die aufwendige Zerlegung des Originalvolumens in Bricks.

4.1.2 Zerlegung des Originalvolumens in Bricks

Bei der Erzeugung des Octrees müssen einige Besonderheiten von OpenGL und der verwendeten Grafikkarte berücksichtigt werden. So muß berücksichtigt werden, ob OpenGL die 3D-Textur mit dem Texturfilter „GL_NEAREST“ oder „GL_LINEAR“ zeichnen soll. Diese Texturfilter verarbeiten die gespeicherten Datenelemente einer Textur, die man bei einer 3D-Textur „Voxel“, bei einer 2D-Textur „Texel“ und bei einem Bild „Pixel“ nennt. Dabei verwendet der Texturfilter „GL_NEAREST“ nur einen Voxel bei der Darstellung eines Pixels. Der Texturfilter „GL_LINEAR“ verwendet jedoch mehrere benachbarte Voxel bei der Darstellung eines Pixels. Durch die Zerlegung des Originalvolumens in Bricks müssen unterschiedliche große 3D-Texturen erzeugt werden. Diese werden im Gesamtbild wieder zusammengesetzt und sollten visuell einer großen 3D-Textur gleichen. Wird eine Textur in zwei Texturen zerlegt und mit dem Texturfilter „GL_NEAREST“ gezeichnet, so entsteht visuell kein Unterschied. Wird der Texturfilter „GL_LINEAR“ angewendet, so entstehen an den Brickgrenzen visuelle Fehler (Abb. 4-1). Die visuellen Fehler bestehen aus sichtbaren Kanten an den Brickgrenzen. Die Fehler entstehen, da der Filteralgorithmus die Nachbarvoxel an den Brickgrenzen nicht berücksichtigt. Um diesen Fehler zu vermeiden, werden die 3D-Texturen an den Brickgrenzen um ein Voxel überlappend errechnet (Abb. 4-2). Dadurch haben die Voxel an den Brickgrenzen denselben Farbwert. Zusätzlich werden die Texturkoordinaten angepaßt, um die veränderte Texturgröße auszugleichen. Durch diese Techniken entstehen bei der Darstellung der Bricks mit Texturfilter keine visuellen Fehler.

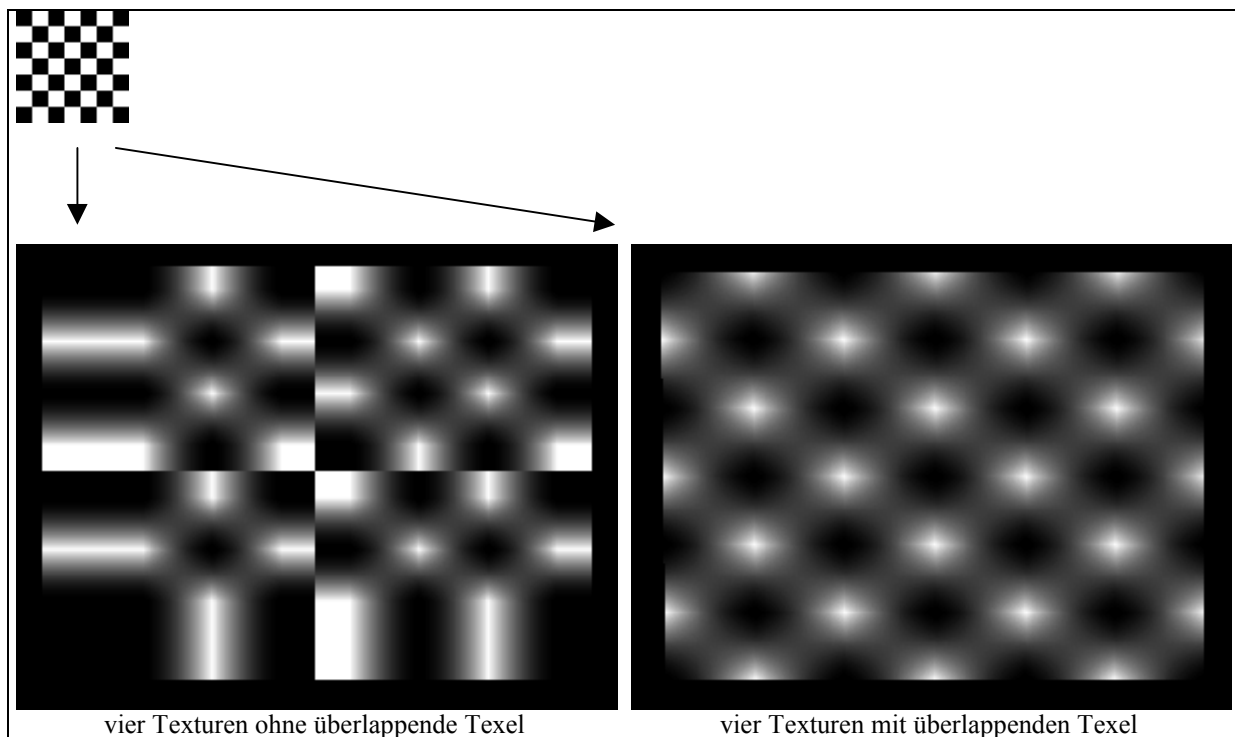


Abb. 4-1 Eine Textur mit der Auflösung 7x7 wird in vier Texturen zerlegt.

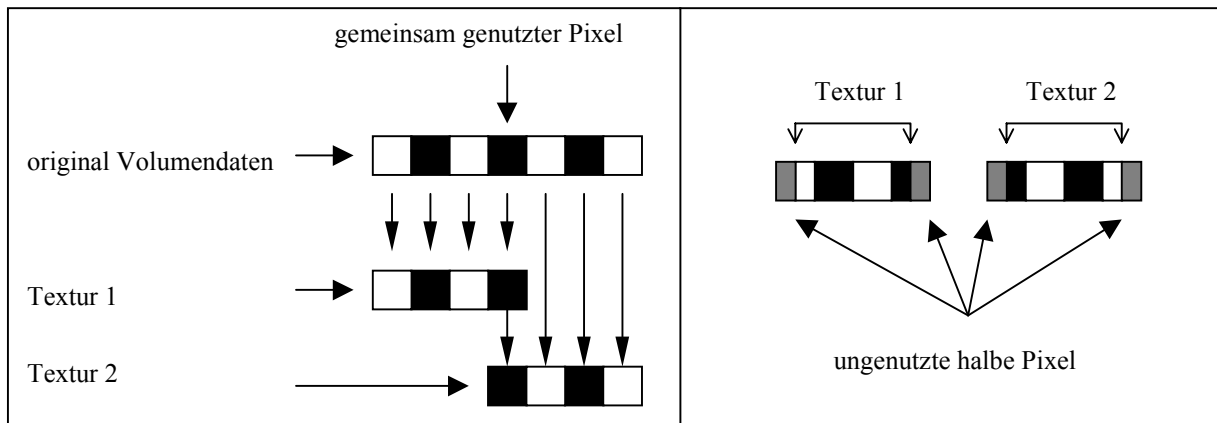


Abb. 4-2 Erzeugung einer eindimensionalen, überlappenden Textur und Darstellung der Texturen mit linearer Filterung

Eine weitere Maßnahme, die aufgrund der Verwendung der ONYX2 Hardware durchgeführt wurde, war die Verschachtelung der Bricks. Die Farbinformationen der 3D-Texturen kann aus Farbinformationen oder Grauwertinformationen bestehen. Ein Voxel bestand also aus 4x8 Bit Farbinformationen, oder aus 1x8 Bit Grauwertinformationen. Die ONYX2 Grafikhardware kann im Texturspeicher als kleinste Voxelinformation 16Bit verwalten. Damit würde normalerweise bei Grauwertinformationen die Hälfte des Texturspeichers ungenutzt bleiben. Die ONYX2 bietet aber die Möglichkeit, bei der Berechnung eines 8Bit Voxels die höherwertigen oder niederwertigen 8Bit eines 16Bit Voxels zu verwenden. Um den Texturspeicher bei 8Bit Grauwertinformationen vollständig auszunutzen, werden immer zwei 3D-Texturen miteinander verschachtelt (Abb. 4-3). In einem solchen Fall haben zwei Bricks die gleiche gemeinsame 3D-Textur. Bei der Darstellung der Bricks werden jeweils die dem Brick zugehörigen höherwertigen oder niederwertigen 8Bit eines 16Bit Voxels verwendet.

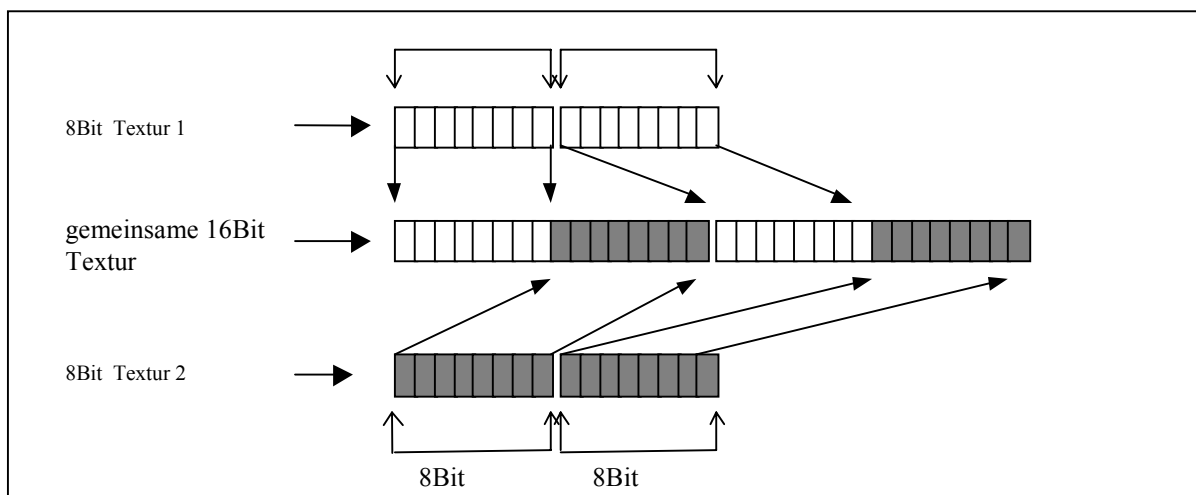
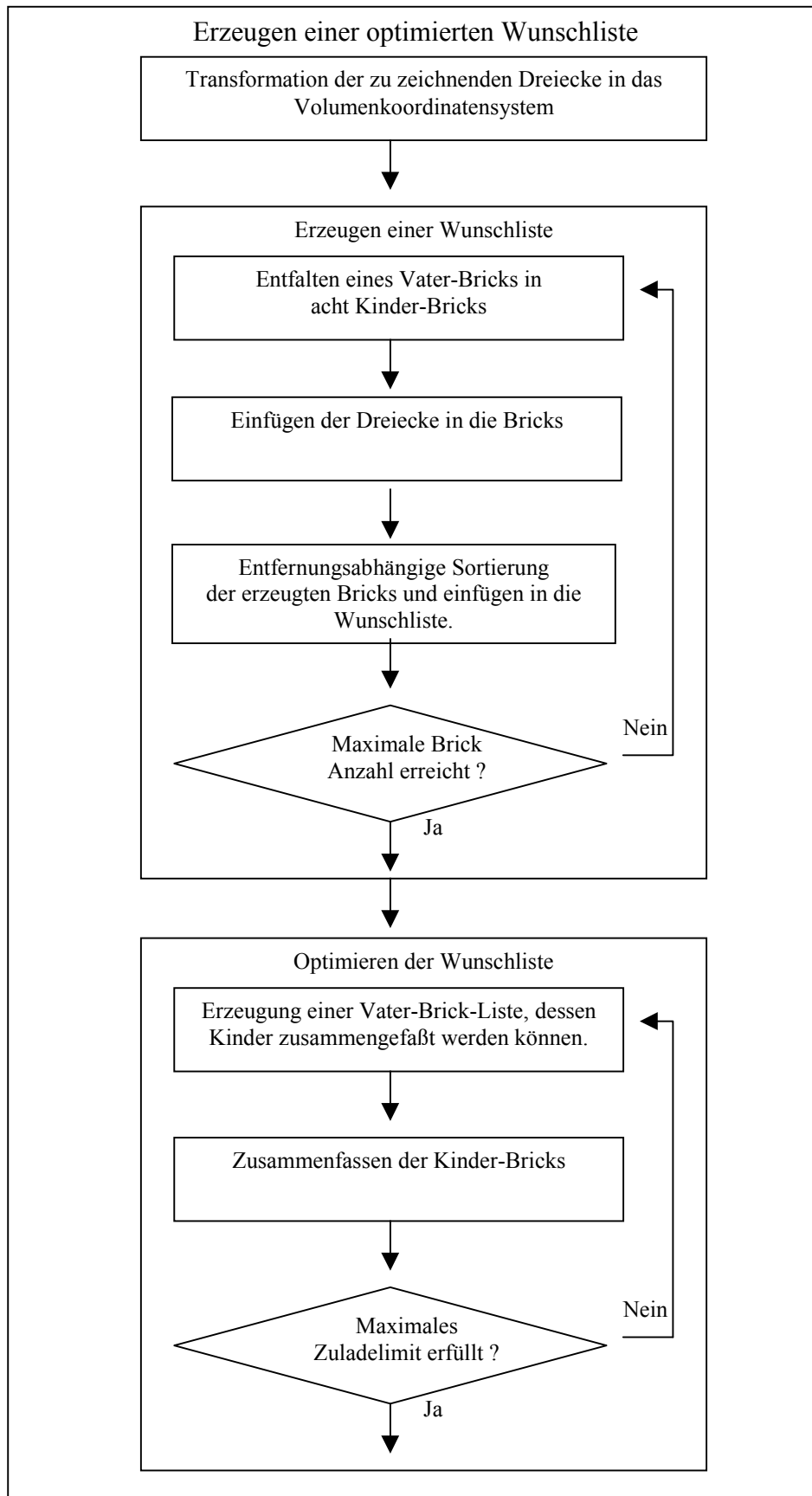


Abb. 4-3 Das Verschachteln zweier 8Bit Texturen

4.2 Erzeugen einer optimierten Wunschliste



4.2.1 Transformation der zu zeichnenden Dreiecke in das Volumenkoordinatensystem.

Avango verwendet zur Verwaltung seiner Matrizen eine Baumstruktur (Abb. 4-4). Daher besitzt das Volumen und die zu zeichnenden Objekte jeweils eine eigene Matrix. Die Dreiecke der Objekte werden innerhalb des Volumenkoordinatensystems gezeichnet und müssen deshalb in dieses transformiert werden. Dazu werden die Dreiecke mit dem Produkt aus der inversen Volumenmatrix und der Objektmatrix transformiert.

$$M_{Transformation} = \prod_{n..1} (MV)^{-1} \cdot \prod_{1..m} (MO)$$

MV = Volumenmatrix
 MO = Objektmatrix
 n = Anzahl der Volumenknoten
 m = Anzahl der Objektknoten

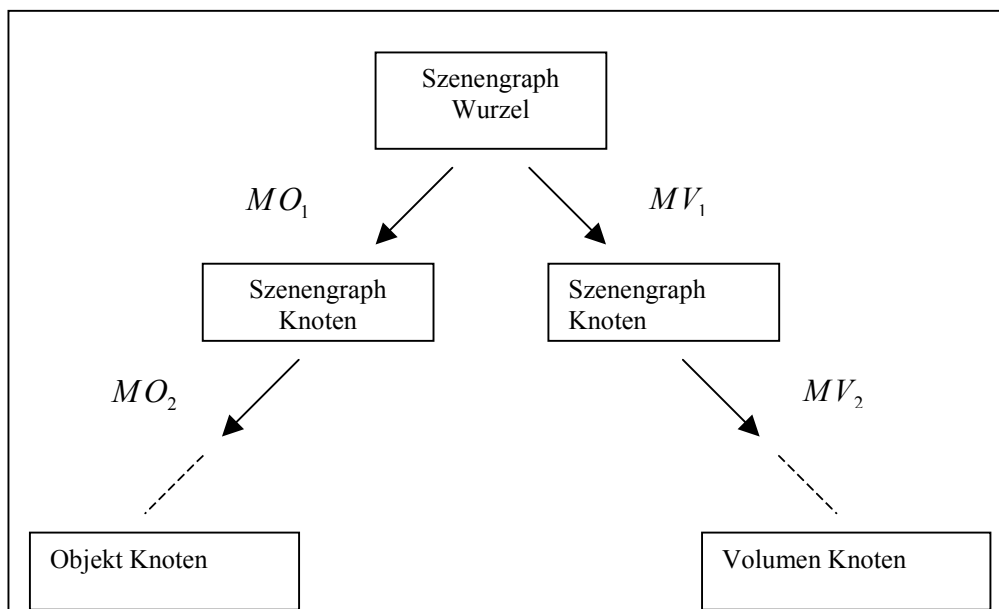


Abb. 4-4 Ein Beispiel eines Szenengraphen

4.2.2 Erzeugen einer Wunschliste

Das Erzeugen einer Wunschliste geschieht wie im Abschnitt 3.4.2 beschrieben unter Ausnutzung der Octreestruktur. Dabei wird überprüft, ob sich die zu zeichnenden Dreiecke ganz oder teilweise innerhalb der Bricks befinden. Befindet sich keines der Dreiecke ganz oder teilweise innerhalb der Bricks, so muß keine Wunschliste erstellt und nichts gezeichnet werden. Befinden sich Dreiecke innerhalb der Bricks, so werden diese in dem Brick gespeichert. Dies dient zur Optimierung des Tests der Dreiecke auf die Kinder-Bricks, denn es können sich nur Dreiecke des Vater-Bricks innerhalb der Kinder-Bricks befinden. Innerhalb des Octrees werden in der nächsten Ebene nur noch die Dreiecke des jeweiligen Vater-Bricks verarbeitet.

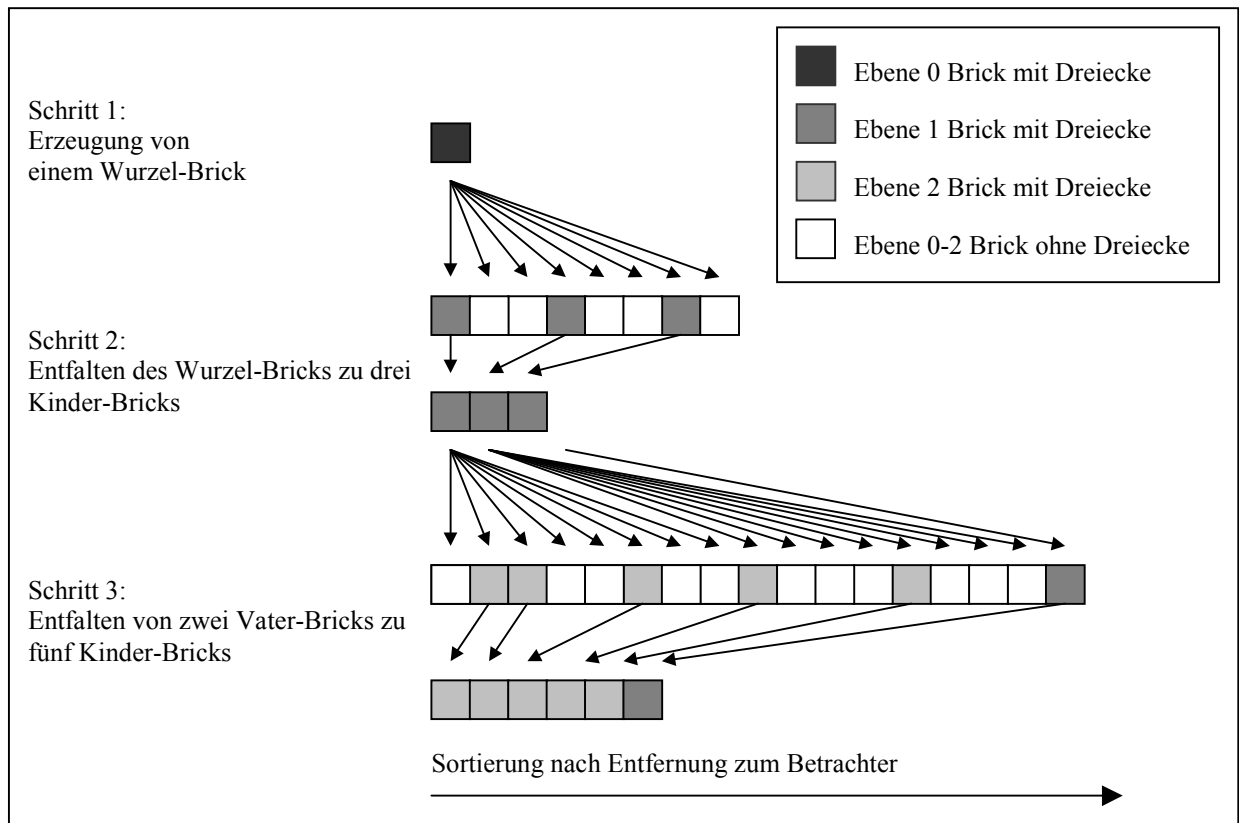


Abb. 4-5 Erzeugung einer Wunschliste mit maximal sechs Bricks

Um festzustellen, ob sich ein Dreieck ganz oder teilweise innerhalb eines Bricks befindet, werden zwei Verfahren angewendet. Danach wird die Tatsache ausgenutzt, daß sich die Dreiecke bereits in dem Volumenkoordinatensystem befinden, und die Brickgrenzen rechtwinklig dazu sind. Zuerst werden die sechs Ebenen definiert, die den Brick begrenzen. Dann wird für jedes Dreieck überprüft, ob sich die drei Eckpunkte auf der dem Brick abgewandten Seite einer Ebene befinden. Ist dies der Fall, ist das jeweilige Dreieck nicht innerhalb des Bricks, und es müssen keine weiteren Tests durchgeführt werden. Der nächste Test der durchgeführt wird, ist ein Ebenentest mit einer Kugel. Das Dreieck definiert die Ebene, und die Kugel umfaßt völlig den jeweiligen Brick. Eine Ebene wird durch einen Abstand und einen Richtungsvektor definiert. Den Richtungsvektor erhält man, indem zwei Vektoren des Dreiecks multipliziert werden, und das Ergebnis normiert wird. Den Ebenenabstand erhält man durch das Skalarprodukt des Richtungsvektors mit einem Dreieckspunkt. Die Kugel hat den gleichen Mittelpunkt wie der Brick. Der Radius der Kugel wird durch einen Briceckpunkt bestimmt. Durch ein Skalarprodukt des Kugelmittelpunktes mit dem Ebenenrichtungsvektor erhält man einen Kugelabstand. Mit dem Kugelabstand und dem Ebenenabstand kann festgestellt werden, ob ein Dreieck einen Brick nicht schneidet. Unterscheidet sich der Ebenenabstand um mehr als der Kugelradius zum Kugelabstand, so steht fest, daß das Dreieck die Kugel nicht schneidet. Mit diesen beiden Tests kann nur nachgewiesen werden, ob sich ein Dreieck nicht innerhalb eines Bricks befindet (Abb. 4-6).

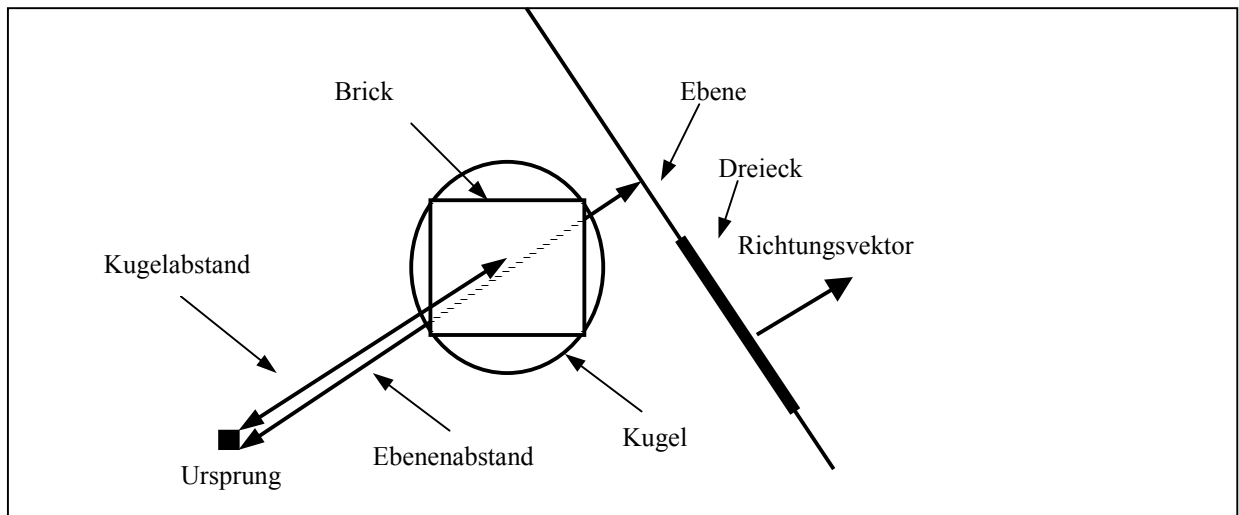


Abb. 4-6 Das Dreieck befindet sich außerhalb des Bricks, da der Kugelabstand plus Radius kleiner ist als der Ebenenabstand

4.2.3 Optimieren der Wunschliste

Um die Echtzeitfähigkeit des Algorithmus zu gewährleisten, wurde ein Zuladelimit definiert. Dieses Zuladelimit begrenzt die Anzahl der Bricks, die pro Frame neu in den Texturspeicher geladen werden dürfen. Dazu wird die Wunschliste mit den schon im Texturspeicher vorhandenen Bricks verglichen. Wird das Zuladelimit überschritten, muß die Wunschliste verkleinert werden. Hierzu wird eine Liste von Vater-Bricks erstellt, dessen Kinder-Bricks zu einem Vater-Brick zusammengefaßt werden. Es ist nicht möglich, Bricks einfach aus der Wunschliste zu entfernen, da hierdurch Löcher im Volumen entstehen würden. Welche Kinder-Bricks zuerst zusammengefaßt werden, entscheiden mehrere Bedingungen. Zuerst werden die Kinder-Bricks ermittelt, die nicht im Texturspeicher sind. Anschließend werden die Bricks die nicht im Texturspeicher sind zusammengefaßt. Dabei werden Bricks höherer Ebenen zuerst zusammengefaßt. Dies ist notwendig, damit das Gesamtbild aus Bricks besteht, die nur aus zwei benachbarten Ebenen stammen. Ein Gesamtbild, welches nur aus zwei hochauflösten, benachbarten Ebenen besteht, sah optisch besser aus als eines mit vielen Auflösungsstufen. Innerhalb der gleichen Ebene werden die Bricks bevorzugt zusammengefaßt, die vom Betrachter weiter entfernt sind. Geringere Auflösungen fallen bei weiter entfernten Bricks nicht so stark auf.

Ein besseres Darstellungsverfahren wäre es, eine Auswahl der Auflösungsstufen ähnlich dem „mip-mapping“ bei 2D-Texturen zu verwenden. Dabei würde die verwendete Auflösung in Abhängigkeit von der Größe und Darstellungsauflösung der verwendeten Bricks ausgesucht werden.

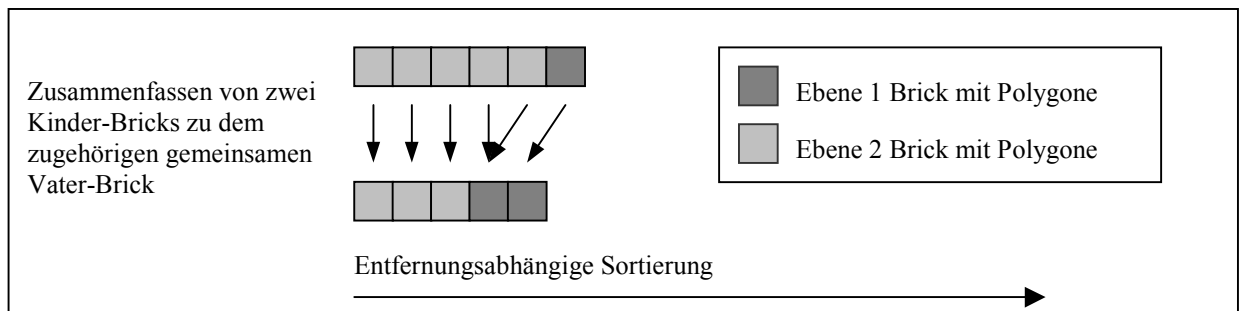
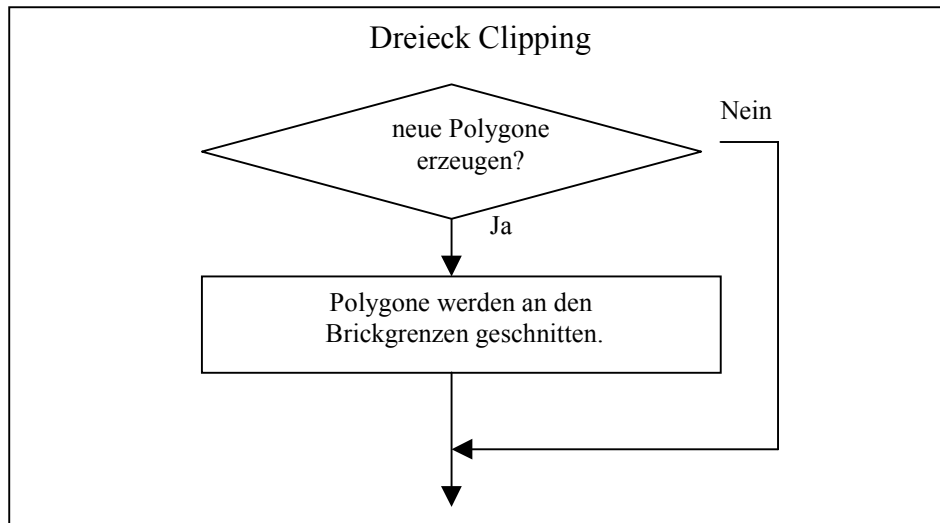


Abb. 4-7 Optimieren einer Wunschliste

4.3 Dreieck Clipping



Die Dreiecke dürfen nicht über die Brickgrenzen hinaus gezeichnet werden. Um dies zu Gewährleisten, gibt es zwei Möglichkeiten. Die Dreiecke lassen sich mit OpenGL Clippingplanes begrenzen. Hierbei wird der Brick durch sechs Clippingplanes begrenzt.

Die zweite Möglichkeit besteht darin, aus den vorhandenen Dreiecken neue auf die Brickgrenzen zugeschnittene Dreiecke zu erzeugen. Es werden nur die Dreiecke aus den Bricks der optimierten Wunschliste zugeschnitten. Für jedes im Brick gespeicherte Dreieck müssen neue Dreiecke erzeugt werden, deren Grenzen an den Dreiecksgrenzen anschließen. Dreiecke im dreidimensionalen Raum auf Brickgrenzen zu schneiden erfordert viel Rechenarbeit, da sehr viele Fallunterscheidungen behandelt werden müssen.

GigaVol verwendet daher ein vereinfachtes Verfahren, welches man „Zweifachschnitt“ nennen könnte (Abb. 4-8). Hierbei wird nicht versucht, in einem Schritt das Dreieck an allen sechs Ebenen zu begrenzen, sondern es werden nur jeweils zwei parallele Ebenen gleichzeitig behandelt. Ein Dreieck wird erst an den zwei X-Ebenen geschnitten, und das Resultat wird danach an den beiden Y-Ebenen geschnitten. Dieses Resultat wird wiederum an den Z-Ebenen geschnitten. Hierdurch wird ein Dreieck durch drei vereinfachte Verfahrensschritte auf die Brickgrenzen zurechtgeschnitten. Aus einem Dreieck erzeugt dieses Verfahren im günstigsten Fall ein neues Dreieck, im schlechtesten Fall neun neue Dreiecke. Dieses Verfahren ist schnell, was für die Echtzeitfähigkeit des Systems wichtig war, aber nicht optimal.

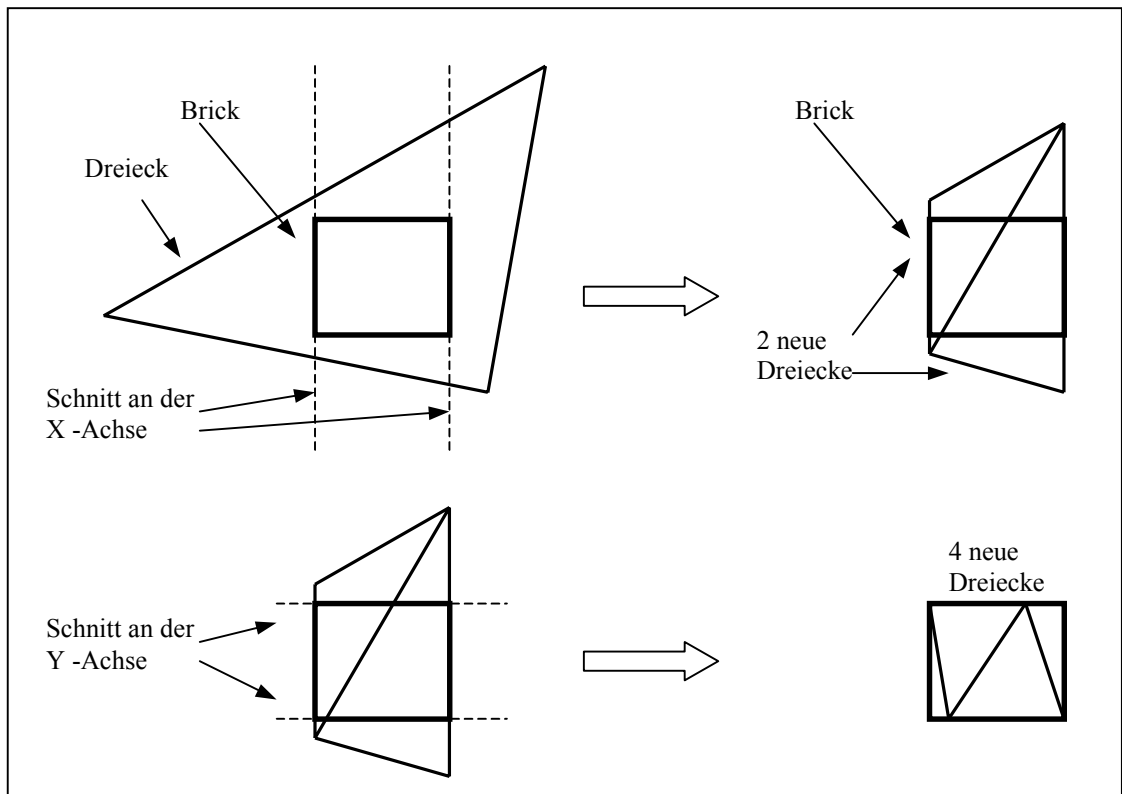
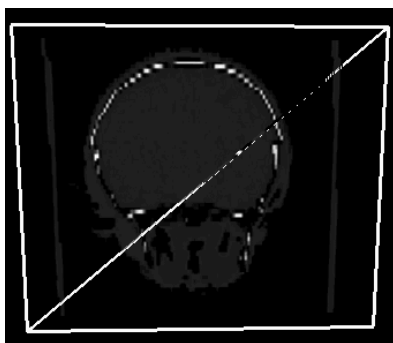
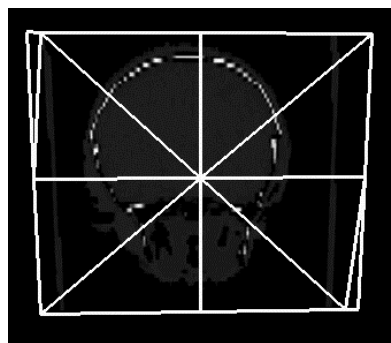


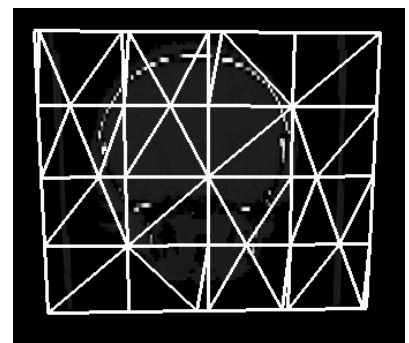
Abb. 4-8 „Zweifachschnitt“ mit zwei Achsen



Zwei ungeschnittene Dreiecke

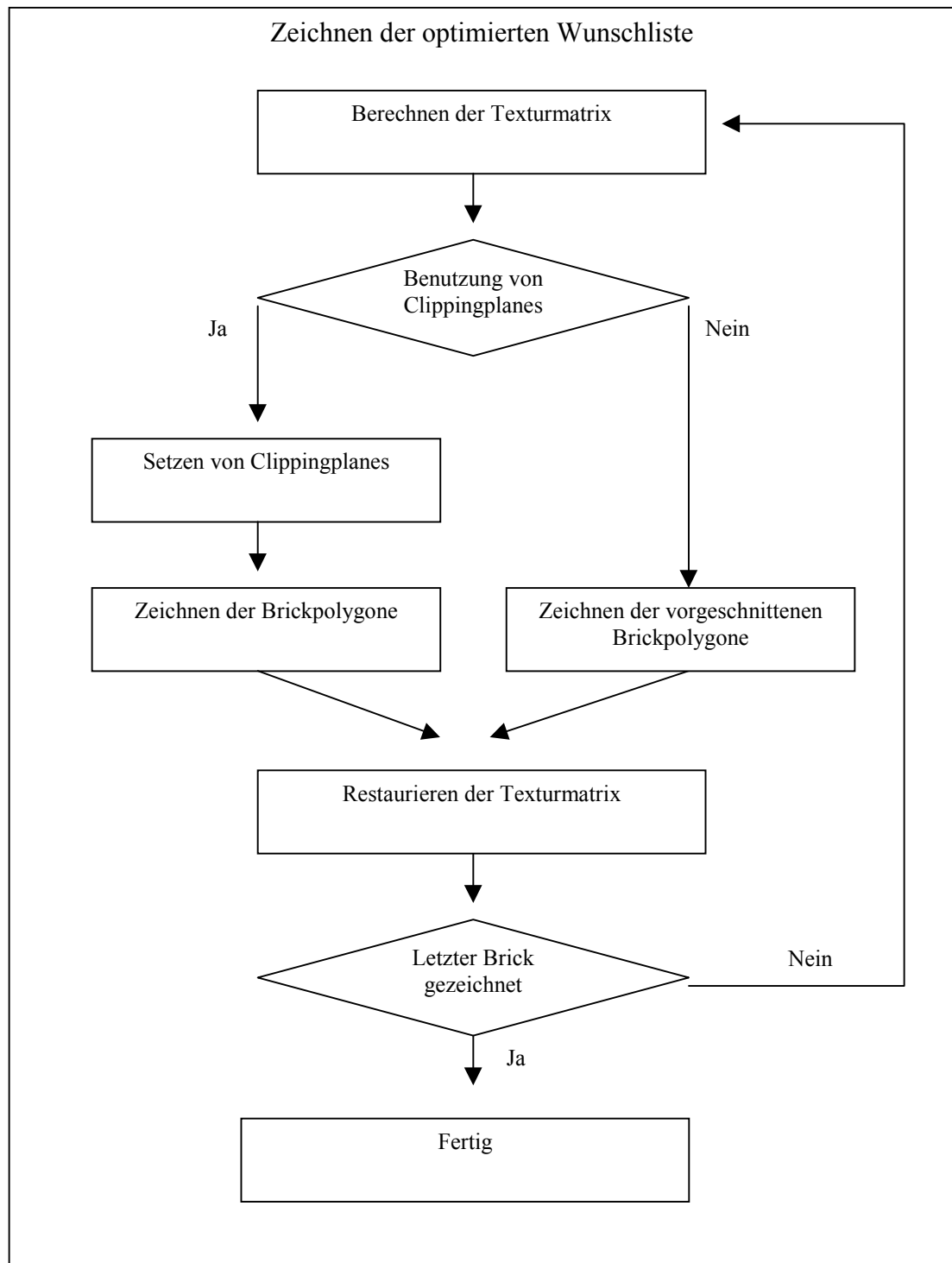


Auf vier Bricks geschnittene Dreiecke



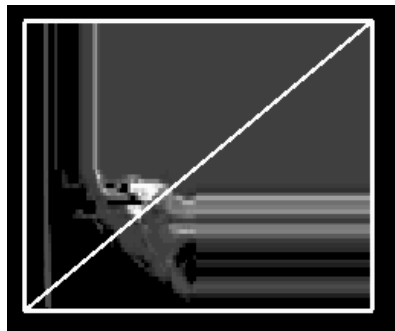
Auf acht Bricks geschnittene Dreiecke

4.4 Zeichnen der optimierten Wunschliste

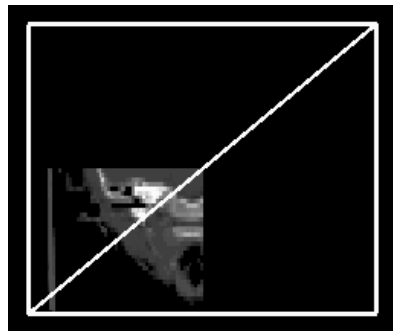


Nachdem die Wunschliste erstellt wurde, müssen die in der Liste gespeicherten Bricks gezeichnet werden. Da die Bricks in der Wunschliste bereits in der Betrachterentfernung sortiert sind, können die Bricks in der gespeicherten Reihenfolge gezeichnet werden. Um die Dreiecke in den Bricks zeichnen zu können, werden noch die dreidimensionalen Texturkoordinaten benötigt. Hierzu wird die automatische Texturkoordinaten-Generierung (glTexGen) von OpenGL verwendet. OpenGL berechnet die Koordinaten aufgrund einer Texturmatrix und der Koordinaten der Dreiecke. Für jeden Brick muß diese Texturmatrix neu berechnet und gesetzt werden. Dies ist notwendig, da das Volumen in Bricks zerlegt wurde,

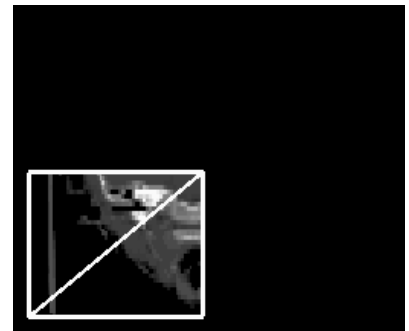
und deshalb die Koordinaten der Dreiecke sich nicht mehr im richtigen Koordinatensystem befinden. Sind die Dreiecke in dem Brick nicht auf die Brickgrenzen zurechtgeschnitten, müssen sechs Clippingplanes gesetzt werden. Clippingplanes sind Ebenen, die das Ausfüllen von Dreiecke mit Pixeln auf nur eine Ebenenhälfte beschränken. Die sechs Clippingplanes werden so berechnet, daß sie die Grenzen des aktuell zu zeichnenden Bricks genau umschließen. Beim Zeichnen werden nur die Pixel gesetzt, die sich innerhalb der Brickgrenzen befinden.



Zwei Dreiecke ohne Clipping



Zwei Dreiecke mit Clipping



Zwei auf Brickgrenzen vorgerechnete Dreiecke

Alle im jeweiligen Brick gespeicherten Dreiecke werden mit oder ohne Clippingplanes gezeichnet. Anschließend wird die Texturmatrix wieder restauriert. Dieses Verfahren wird mit jedem Brick durchgeführt, bis der letzte Brick in der Liste gezeichnet ist.

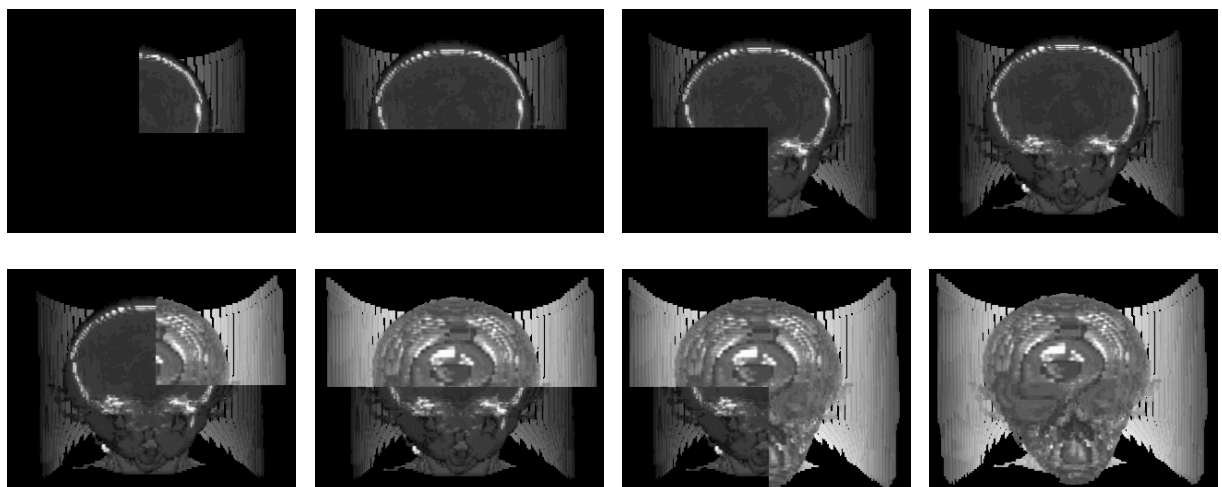


Abb. 4-9 Ein Volumen wird aus acht Bricks zusammengesetzt

4.5 Implementierung der Software auf einem Multiprozessorsystem

Eine große Herausforderung bei der Implementierung der GigaVol-Software stellte die Integration in ein Multiprozessorsystem wie der ONYX2 dar. Mithilfe von Avango konnte die Software die Multiprozeß- und Multipipefähigkeiten einer ONYX2 sehr gut ausnutzen.

Um den Bildaufbau zu beschleunigen, teilt Avango die Bildberechnung in drei Prozesse auf:

- „application process“: Behandelt sämtliche Aufgaben, die vor der eigentlichen Zeichnung des Bildes erledigt werden müssen.
- „cull process“: Alle Bildelemente, die nicht im sichtbaren Bereich liegen, werden in diesem Prozeß entfernt (view-frustum-culling).
- „draw process“: Alle nötigen OpenGL-Anweisungen werden in diesem Prozeß ausgeführt.

Je nach Verfügbarkeit von Prozessoren können diese drei Prozesse auf einen, zwei oder drei Prozessoren aufgeteilt werden.

	Prozessor 1: „application process“	Prozessor 2: „cull process“	Prozessor 3 „draw process“ und BildschirmAusgabe:
Bildberechnung Zeit 1	Bild 1	-	-
Bildberechnung Zeit 2	Bild 2	Bild 1	-
Bildberechnung Zeit 3	Bild 3	Bild 2	Bild 1
Bildberechnung Zeit 4	Bild 4	Bild 3	Bild 2
Bildberechnung Zeit 5	Bild 5	Bild 4	Bild 3

Abb. 4-10 Avango Bilderzeugung mit drei Prozessoren

Verfügt jeder Prozeß über einen eigenen Prozessor, so können praktisch drei Bilder gleichzeitig berechnet werden. Dies ermöglicht Geschwindigkeitsvorteile, führt aber auch dazu, daß eine Veränderung des Bildes im „application process“ immer mit einer Verzögerung von zwei Bildern erfolgt (Latenz).

GigaVol bietet die Möglichkeit, die aufwendige Berechnung der optimierten Wunschliste und das Dreieck-Clipping vom „draw process“ in den „cull process“ auszulagern.

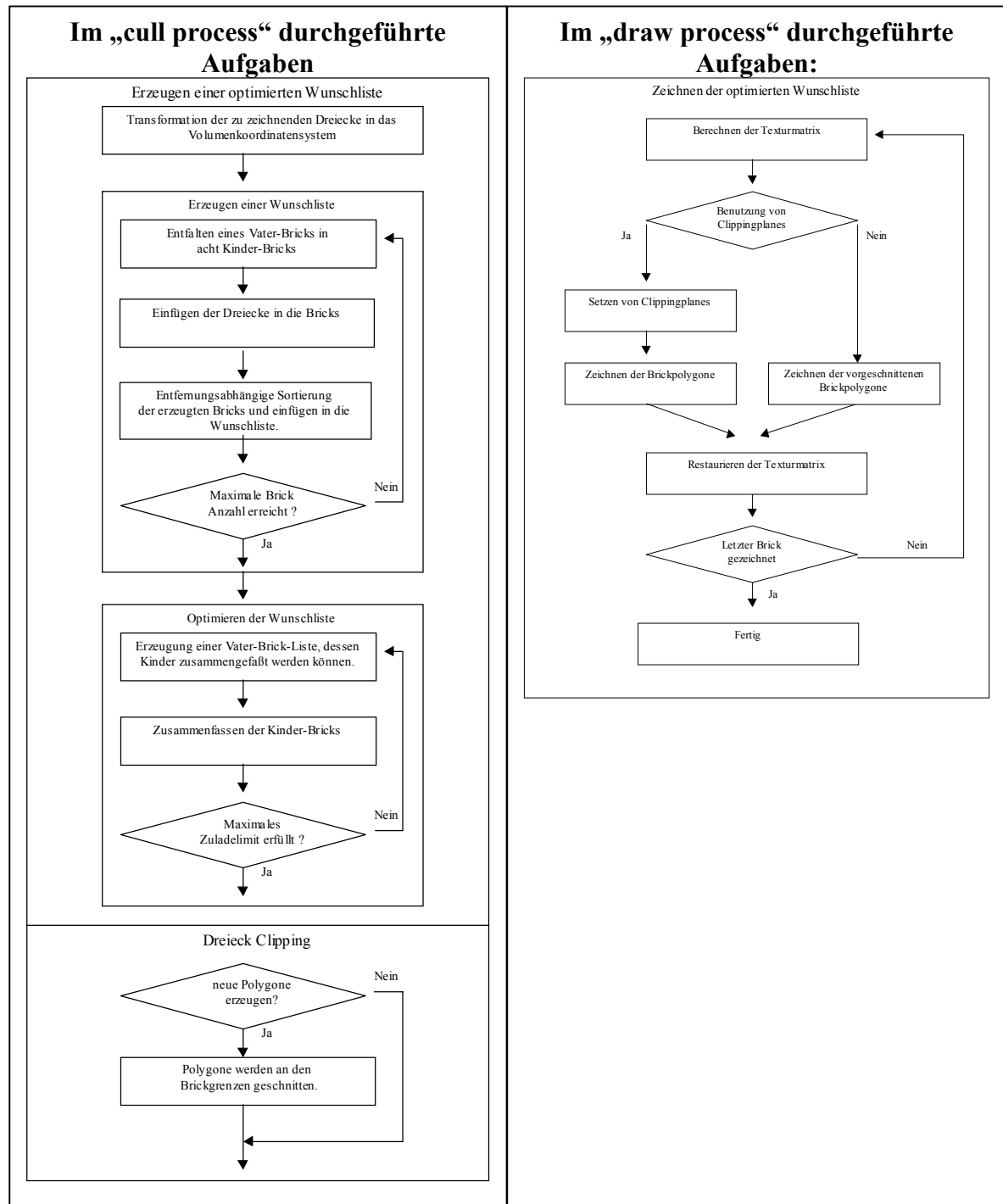
	Prozessor 1: „application process“	Prozessor 2: „cull process“	Prozessor 3 „draw process“ und BildschirmAusgabe:
Bildberechnung Zeit 1	Bild 1	-	-
Bildberechnung Zeit 2	Bild 2	Erzeugung optimierte Wunschliste Bild 1	-
Bildberechnung Zeit 3	Bild 3	Erzeugung optimierte Wunschliste Bild 2	Zeichnen Bild 1
Bildberechnung Zeit 4	Bild 4	Erzeugung optimierte Wunschliste Bild 3	Zeichnen Bild 2
Bildberechnung Zeit 5	Bild 5	Erzeugung optimierte Wunschliste Bild 4	Zeichnen Bild 3

Abb. 4-11 GigaVol nutzt den „cull process“ und den „draw process“

Es entstehen somit mehrere voneinander unabhängige Berechnungsprozesse. Die Anzahl der Prozesse läßt sich folgendermaßen berechnen:

Anzahl der Prozesse = 2 x Anzahl der verwendeten Pipes

Um Dateninkonsistenzen zu vermeiden, werden für jeden Prozeß einige Variablen mehrmals verwaltet. Für die gemeinsam benutzten Variablen wird der Zugriff auf jeweils einen Prozeß eingeschränkt.



4.6 Texturspeicherverwaltung von GigaVol

Um eine Darstellung von besonders großen Volumendaten zu ermöglichen, wurde die Verwaltung des Texturspeichers von GigaVol selbst übernommen. Normalerweise verwaltet OpenGL den Texturspeicher selbst. Eine normale Anwendung lädt vor dem Zeichnen alle Texturen in den Texturspeicher und OpenGL lagert die Texturen bei Bedarf ein und aus. Ein großer Nachteil wäre, wenn die Texturen von OpenGL noch in den Hauptspeicher kopiert würden und deshalb den doppelten Speicherbedarf erforderten. Außerdem lassen sich bei der Verwaltung keine Prioritäten bestimmen, welche Texturen noch im Hauptspeicher bleiben müssen, und welche ausgelagert werden können. Es kommt somit zu erheblichen Verzögerungen, wenn die angeforderten Texturen nicht im Texturspeicher sind.

Um die Verwaltung der Texturen mit GigaVol zu ermöglichen, reserviert GigaVol bei OpenGL eine bestimmte Menge Texturspeicher. Diese entspricht höchstens der Menge von Bricks, die maximal in den Texturspeicher passen. Es ist auch möglich, einen Teil des Texturspeichers für andere Anwendungen zu reservieren. Müssen neue Bricks in den Texturspeicher geladen werden, so werden die alten Bricks nicht ausgelagert, sondern durch neue Bricks ersetzt. Derzeit ersetzt GigaVol die nicht benötigten Bricks, die es als erstes findet. Eine mögliche Verbesserung des Systems wäre es, die Bricks zu ersetzen, die eine längere Zeit nicht verwendet worden sind.

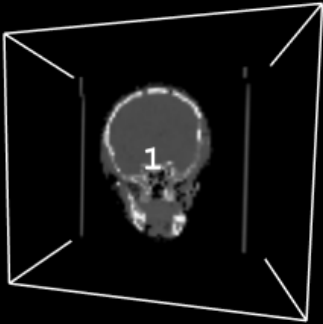
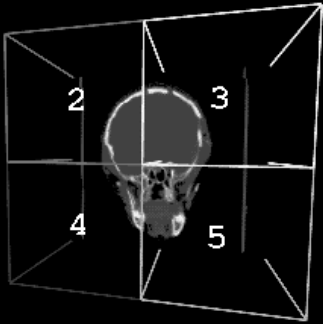
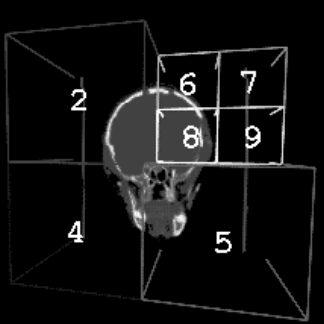
		
<p>Im Texturspeicher: 1 </p> <p>Brick 1 wird in den Texturspeicher geladen.</p>	<p>Im Texturspeicher: 1 2 3 4 5 </p> <p>Vier neue Bricks im Texturspeicher. Brick 1 wird nicht ersetzt, da er gebraucht werden könnte.</p>	<p>Im Texturspeicher: 6 2 7 4 5 8 9 </p> <p>Vier neue Bricks im Texturspeicher. Die Bricks 1 und 3 mußten ersetzt werden, da sonst kein Texturspeicher frei war.</p>

Abb. 4-12 Texturspeicherverwaltung mit sieben Texturbrickreservierungen

4.7 Kurze Beschreibung der Programmschnittstelle von GigaVol

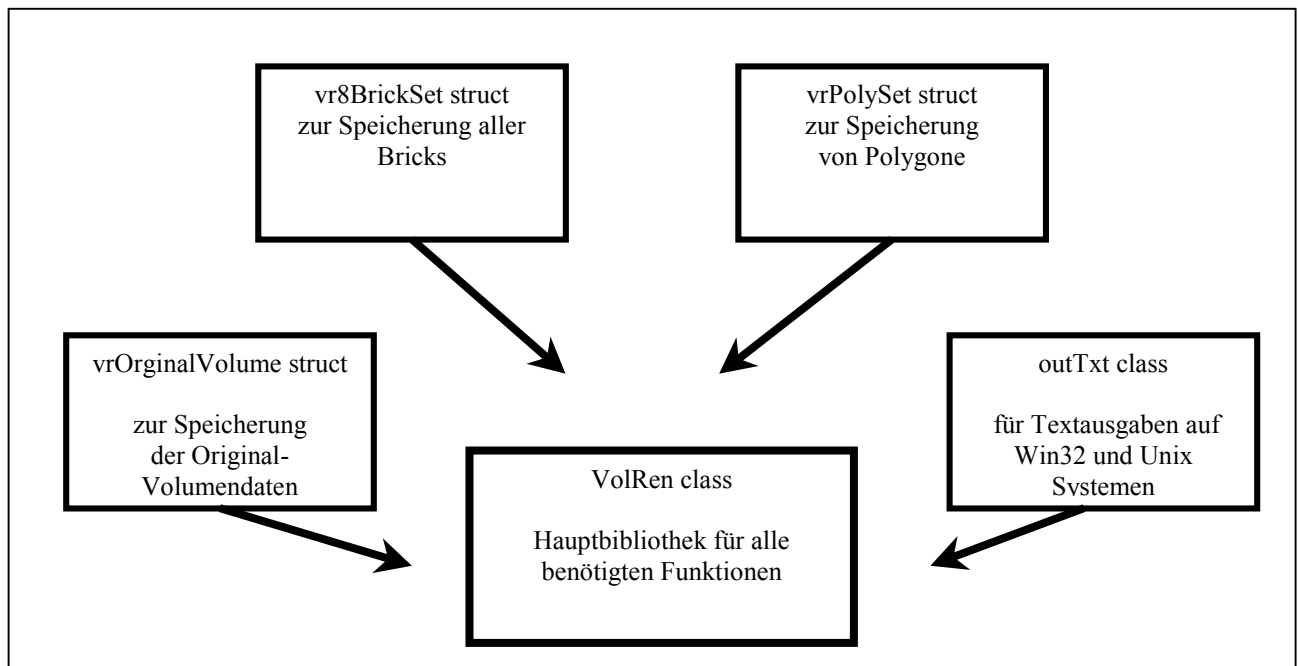


Abb. 4-13 Strukturaufbau von GigaVol

Die Programmierung von GigaVol erfolgt durch ein C++ Hauptklasse (*VolRen*), die alle Funktionen enthält, die für das Zeichnen von Volumendaten nötig sind. Die Hauptklasse erhält alle Daten von mehreren Strukturen (*vrOriginalVolume*, *vr8BrickSet*, *vrPolySet*), die vorher initialisiert werden müssen.

Die wichtigsten Programmierschritte die zum Zeichnen eines Volumens nötig sind:

1. Anlegen einer Instanz von *VolRen* und Strukturen:

```
volRen *volumenRenderer = new volRen (out);  
vrOriginalVolume orgVl;  
vr8BrickSet brickSet;  
vrPolySet polySet;
```
2. Polgone die für das Zeichnen des Volumens benötigt werden eingefügt:

```
polySet.pv1[polySet.numOfPolys] = vec_4 (-1.0, -1.0, 0.0 , 1.0);  
.....
```
3. Laden eines Volumens

```
volumenRenderer ->loadVolume3DTIFF (&orgVl,"..\..\VolData\CThead.tiff")
```
4. Eine optimale Brickliste wird errechnet und in *brickSet* gespeichert

```
volumenRenderer ->makeOptimizedWishList (&brickSet, &polySet.....);
```
5. Die Brickliste wird gezeichnet:

```
volumenRenderer ->drawPolys (&brickSet, &polySet.....);
```


5 Verschiedene Entwicklungsstufen von GigaVol

Die beiden Hauptziele bei der Entwicklung des Algorithmus waren, die Echtzeitfähigkeit des Systems und die Möglichkeit, sehr große Volumendatensätze darstellen zu können. Diese Ziele wurden durch mehrere Faktoren der modernen Grafikhardware limitiert. Diese Faktoren werden anhand der SGI ONYX2 erläutert, der besten und teuersten Grafikhardware, die zur jetzigen Zeit (Jahr 2000) zu erwerben ist. Ein leicht zu überwindender limitierender Faktor ist der vorhandene Hauptspeicher. Er läßt sich auf einer ONYX2 auf mehrere Gigabyte ausbauen, und durch die vorhandenen 64Bit-Prozessoren leicht ansprechen. Zwei wichtige limitierende Faktoren des Systems sind die gegebene Rasterleistung und die Größe des vorhandenen Texturspeichers. Die Rasterleistung ist die Geschwindigkeit, die das System benötigt, um die gegebenen Dreiecke im Bildschirmspeicher mit Pixeln auszufüllen. Sie läßt sich auf der ONYX2 in drei Ausbaustufen variieren. Der vorhandene Texturspeicher ist mit 64MB fest vorgegeben. Eine Verbesserung dieser beiden Faktoren wäre nur durch eine neue Generation der Grafikhardware gegeben, welche immer ein paar Jahre Entwicklungszeit in Anspruch nimmt. Die Zeit, die man für die Berechnung eines Frames benötigt, setzt sich somit zusammen aus den Zeiten für das Zeichnen der Pixel und dem Ein- und Auslagern von Texturen.

Zeit für ein Frame = Zeit für Rastern der Pixel + Zeit für Ein- und Auslagern von Texturen

Die Zeit, die man für das Ein- und Auslagern von Texturen (englisch „paging“) benötigt, ist maßgeblich von der Größe des Volumens abhängig. Paßt das Volumen komplett in den Texturspeicher, so kann diese Zeit vernachlässigt werden, da das Volumen nur einmal in den Texturspeicher einlagert werden muß. Ist das Volumen größer als der Texturspeicher, so muß innerhalb der Berechnung eines Bildes der Texturspeicher ganz oder teilweise ausgetauscht werden. Unter günstigen Bedingungen kann die ONYX2 ca. 250 MB/s 3D-Texturen in den Texturspeicher einlagern. Ein Volumen, welches doppelt so groß ist wie der Texturspeicher, müßte in zwei 64MB-Texturen aufgeteilt werden, damit es gezeichnet werden kann. Die Zeit, die die ONYX2 zum Einlagern der zwei 64MB-Texturen braucht, kann folgendermaßen berechnet werden:

$$\frac{2 \times 64MB}{250 \frac{MB}{s}} = 512ms$$

Um das System in Echtzeit zu verwenden (unter 125 ms/Frame), ist diese zum Einlagern der Texturen benötigte Zeit viel zu hoch.

Der gewählte Ansatz war, das Volumen in viele kleine Bricks zu unterteilen, die eine feste, vorgegebene Größe haben. Der Nutzer konnte sich das Volumen mit einer vorgegebenen „Betrachtungsebene“ ansehen. Dabei wurden nur die benötigten Bricks in den Texturspeicher geladen.

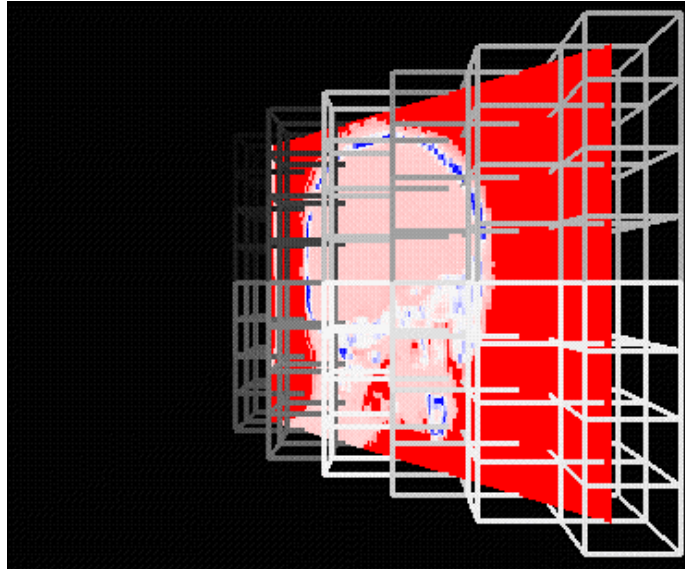


Abb. 5-1 Eine Schnittsebene wird durch ein Volumen geschoben

Dieser Ansatz hatte einige Nachteile, die den praktischen Nutzen ausschlossen:

- Das Volumen konnte nicht komplett betrachtet werden.
- War die Texturmenge der benötigten Bricks größer, so kam es wieder zu Verzögerungen durch das Ein- und Auslagern der Textur.
- Wurde die Ebene schnell durch das Volumen bewegt, so kam es auch wieder zu langen Ein- und Auslagerungszeiten.

Ein weiterer Ansatz war es, die Bricks in verschiedenen Auflösungsstufen zu speichern. Es wurden erst grob aufgelöste Bricks in den Texturspeicher eingelagert und später durch die feinen Auflösungen ersetzt.

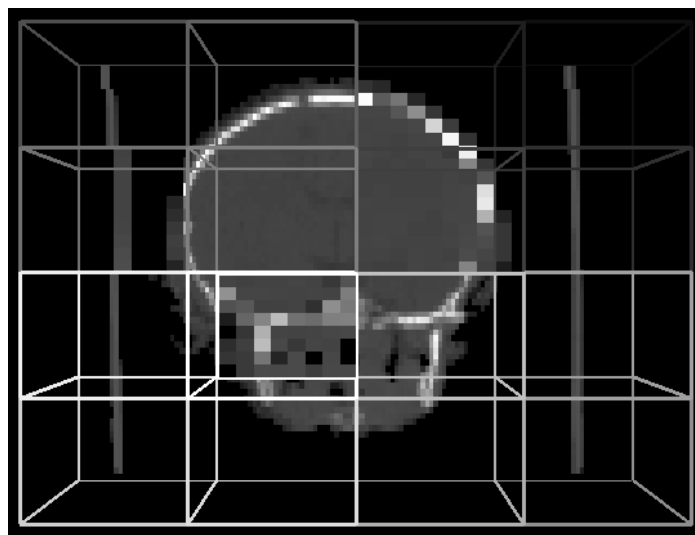


Abb. 5-2 Ein Bild wird aus Bricks mit mehreren Auflösungen zusammengesetzt

Um Texturladezeit zu sparen, wurden verschiedene Auflösungsstufen verwendet. Dadurch ergaben sich viele kleine Texturen. Dabei wurde festgestellt, daß die Geschwindigkeit der Textureinlagerungszeiten der ONYX2 sehr stark von der Größe der Textur abhängt. Viele kleine Texturen werden von der ONYX2 erheblich langsamer in den Texturspeicher geladen als wenige große Texturen mit derselben Gesamtgröße.

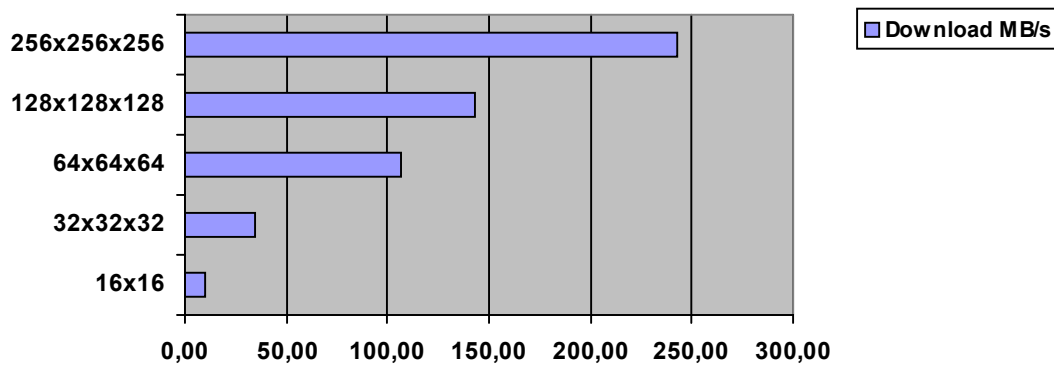


Abb. 5-3 Durchsatz der Textureinlagerung bei der ONYX2

Diese verschiedenen Werte des Durchsatzes der Textureinlagerung entstehen durch das Pipelinesystem der ONYX2, bei der eine hohe Leistung nur bei großen Texturen erreicht werden kann.

Diese Nachteile wurden durch den Octreeansatz beseitigt. In dem jetzt verwendeten Octree haben alle Texturen die gleiche Größe. Der Vorteil ist, daß die grob aufgelösten Texturen in höherer Ebenen nicht sehr klein sind, sondern einen großen Bereich des Volumens speichern. Zusätzlich wurde ein Zuladelimit eingeführt, welches die Bricks limitiert, die pro Frame in den Texturspeicher geladen werden.

Vorteile der Verwaltung der Bricks in einem Octree:

- Es entstehen keine kleinen Bricks bei groben Auflösungen.
- Die Anzahl der Bricks, die pro Frame eingelagert werden, kann limitiert werden.
- Der vorhandene Texturspeicher kann optimal ausgenutzt werden.
- Da keine langen Pausen beim Bildaufbau entstehen, ist das System echtzeitfähig.
- Das komplette Volumen ist darstellbar.

Nachteile der Verwaltung der Bricks in einem Octree:

- Die Verwaltung der Bricks benötigt Rechenleistung.
- Die feinste Auflösung des Volumens ist nicht sofort sichtbar.

Bei der Verwaltung der Texturen von OpenGL kam es öfters zu Verzögerungen, weil die benötigten Texturen ausgelagert wurden. Daher wurde die Verwaltung der Texturen von GigaVol selbst übernommen (siehe Seite 31).

Bei einer Berechnung der Wunschliste muß getestet werden, ob sich die Dreiecke in den Bricks befinden. Außerdem müssen die Dreiecke beim Zeichnen durch Clippingplanes begrenzt werden.

Es wurde versucht, diese Aufgaben durch ein Zuschneiden der Dreiecke bei der Octree-Erzeugung einzusparen. Die in den Vater-Bricks befindlichen Dreiecke wurden auf die Kinder-Bricks zugeschnitten. Danach wurde überprüft, ob sich in den Kinder-Bricks Dreiecke befinden. Die Überprüfung, ob sich Dreiecke innerhalb der Bricks befinden, wurde damit überflüssig (siehe Seite 25). Es wurde jedoch festgestellt, daß häufiges Zuschneiden der Dreiecke einen erheblichen Rechenaufwand erfordert, und die Anzahl der resultierenden Dreiecke sehr groß und verschwenderisch war. Daher wird das Zuschneiden der Dreiecke optional an der optimierten Wunschliste durchgeführt.

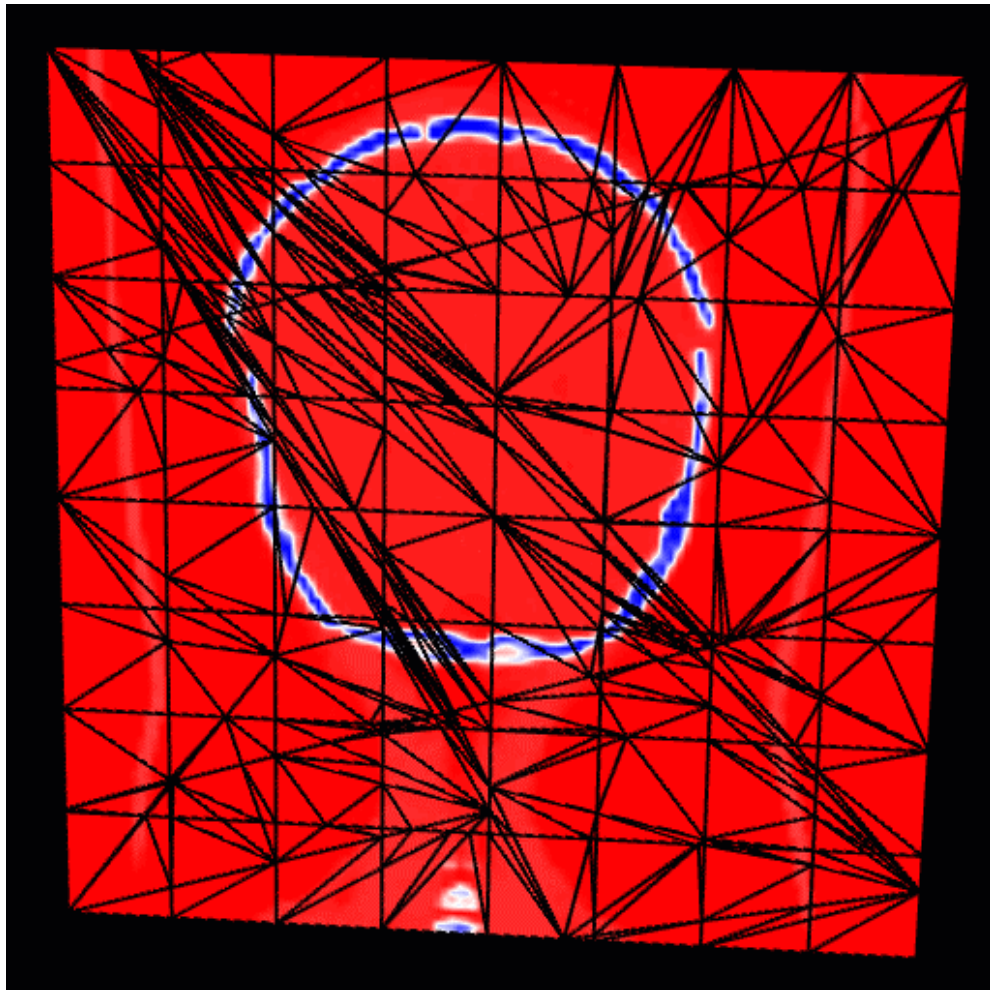


Abb. 5-4 Häufiges Zuschneiden der Dreiecke erzeugt einen „Dreiecks- Scherbenhaufen“

Durch diese Maßnahmen können sehr große Volumen dargestellt werden, und die Zeit für das Ein- und Auslagern von Texturen konnte minimiert werden. Die Zeit, die für das Rastern der Pixel benötigt wird, hängt von der verwendenden Hardware ab.

**Zeit für ein Frame = Zeit für Rastern der Pixel +
minimierte Zeit für das limitierte Einlagern von Texturen**

6 Zeit und Geschwindigkeitsmessungen für das Volumenrendern

In diesem Kapitel wird die Geschwindigkeit der für das Volumenrendern nötigen Faktoren gemessen.

6.1 TexBench für OpenGL

Die Geschwindigkeit, die beim Volumenrendering erzielt wird, hängt primär von der Textureinlagerungszeit und der Rasterleistung des verwendeten Systems ab. Um diese beiden Faktoren, die man auch „download time“ und „fillrate“ nennt, zu messen, wurde ein eigenes OpenGL-Programm entwickelt. Dieses TexBench genannte Programm ist in der Lage, die Textureinlagerungszeit und die maximale Rasterleistung für verschiedene Texturgrößen und Texturarten zu ermitteln.

Um die Texturladezeit zu ermitteln, werden die Texturen mehrmals in den Texturspeicher geladen und die verwendete Zeit gemessen. Die Rasterleistung wird ermittelt, indem zwei große Dreiecke mit der gewünschten Textur mehrmals gezeichnet werden.

Das Programm wurde für verschiedene Systeme veröffentlicht und ist kostenlos mit Sourcecode unter [10] erhältlich.

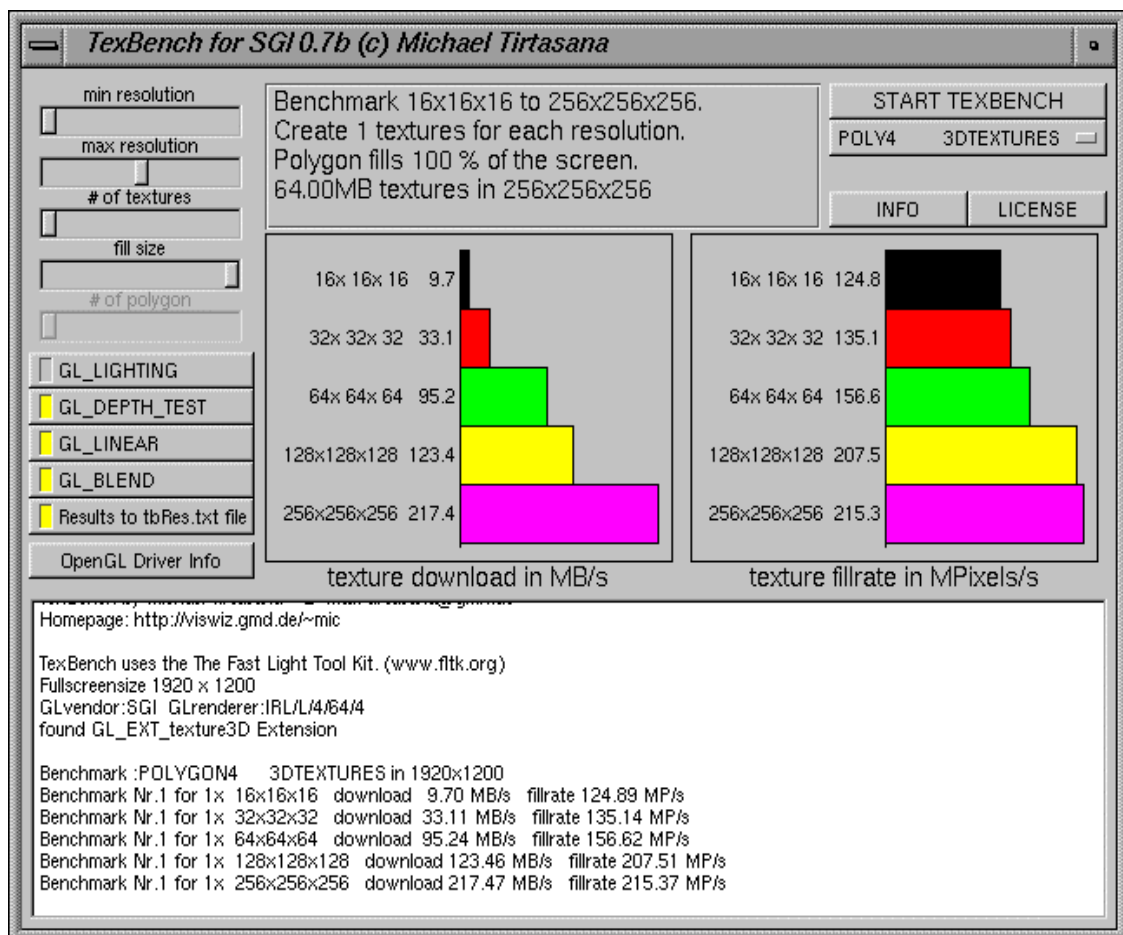


Abb. 6-1 Die Oberfläche von TexBench nach einer Messung von 3D-Texturen

Das Programm bietet folgende Möglichkeiten:

- Messung von 2D-, 3D- und Multi-Texturen
- Messung von verschiedenen Texturen mit Größen von 16x16x16 bis 4096x4096x4096
- Messung unter verschiedenen OpenGL-Zuständen.

6.2 Texturspeicherdurchsatz auf der ONYX2 mit InfiniteReality2 und der Wildcat 4110

Die Messungen wurden mit TexBench auf einer SGI ONYX2 mit InfiniteReality2 und einer PC-Grafikkarte, der Intense3D Wildcat 4110, durchgeführt. Die Wildcat wurde einbezogen, weil sie die im PC-Bereich selten vorzufindenden 3D-Texturen in Hardware beschleunigt.

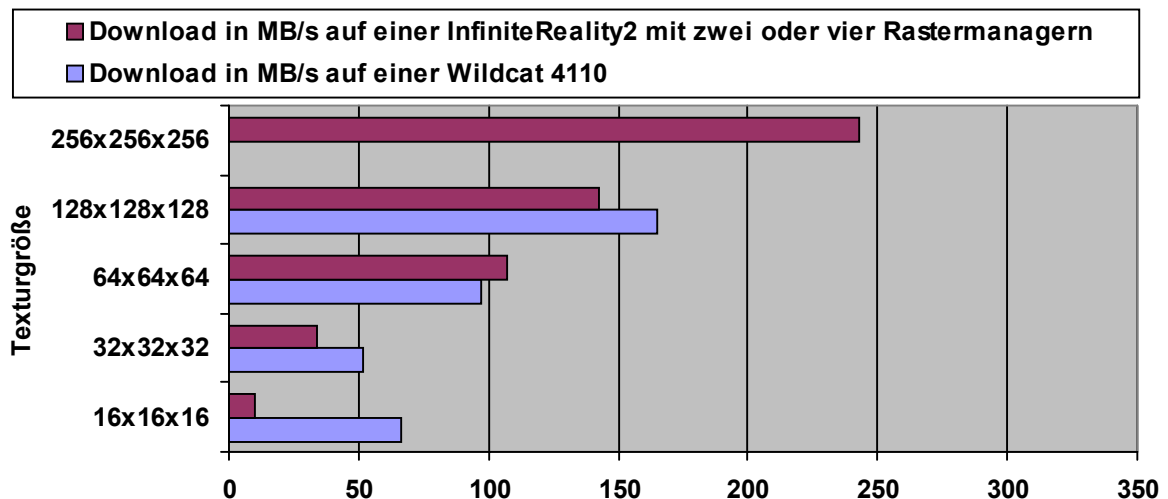


Abb. 6-2 3D-Texturdurchsatz auf einer ONYX2 und auf einer Intense3D Wildcat

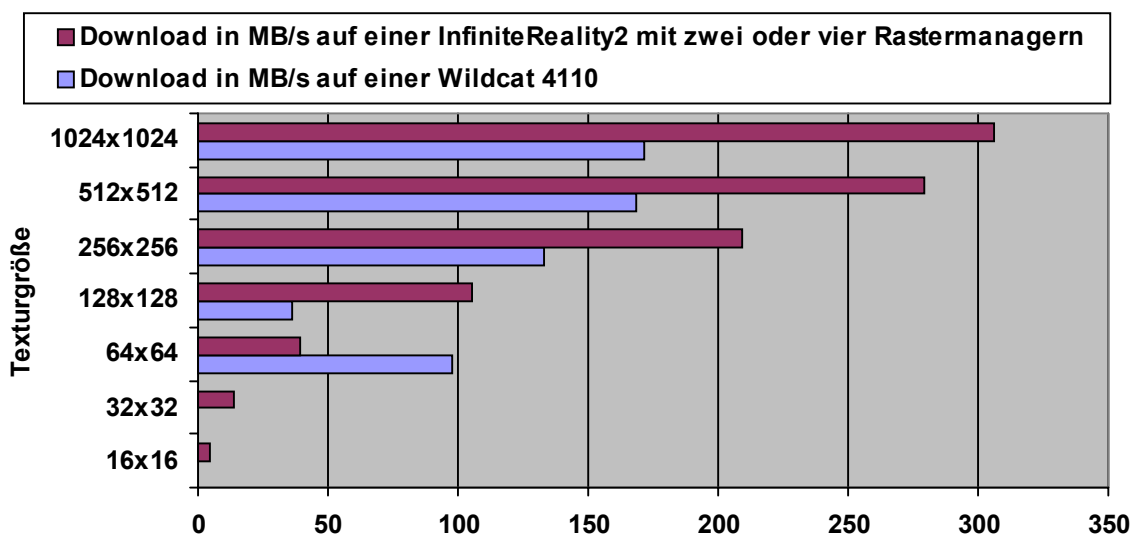


Abb. 6-3 2D-Texturdurchsatz auf einer ONYX2 und auf einer Intense3D Wildcat

Die Datenrate, die der Hersteller für die Grafikpipeline der ONYX2 angibt, beträgt 320MB/s. Dieser Leistung der Grafikpipeline, die auch für das Laden in den Texturspeicher zuständig ist, wird nur selten erreicht. Die erzielte Leistung wird maßgeblich durch die Texturgröße bestimmt. Bei den Messungen der 2D- und 3D-Texturen ist eine lineare Abhängigkeit zwischen der Texturgröße und dem Texturdurchsatz deutlich erkennbar. Bei sehr kleinen Texturen wird nur ein Bruchteil der maximalen Leistung erzielt. Dieses Verhalten liegt an dem Pipelinesystem der ONYX2, die für jede Textur einen DMA-Transfer benötigt (Direct Memory Access). Das Pipelinesystem erhält direkten Zugriff auf den Arbeitsspeicher der

ONYX2). Jeder DMA-Transfer erfordert eine Anlaufzeit, daher wird die volle Leistung nur bei großen Texturen erreicht.

Auf der Intense3D Wildcat 4110 läßt sich ein ähnliches Verhalten feststellen, welches aber nicht so deutlich ausgeprägt ist wie auf der ONYX2.

6.3 Textur-Fillrate auf der ONYX2 InfiniteReality2 und der Wildcat 4110

Die Messung wurden mit aktiviertem bilinearen oder trilinearen Filter durchgeführt.

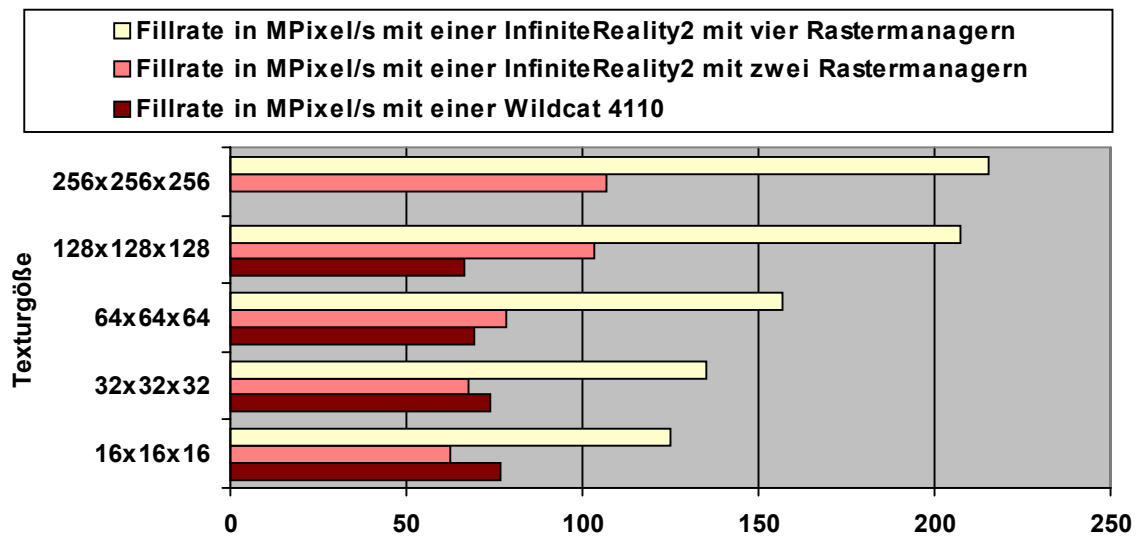


Abb. 6-4 3D-Textur-Fillrate auf einer ONYX2 und einer Wildcat 4110

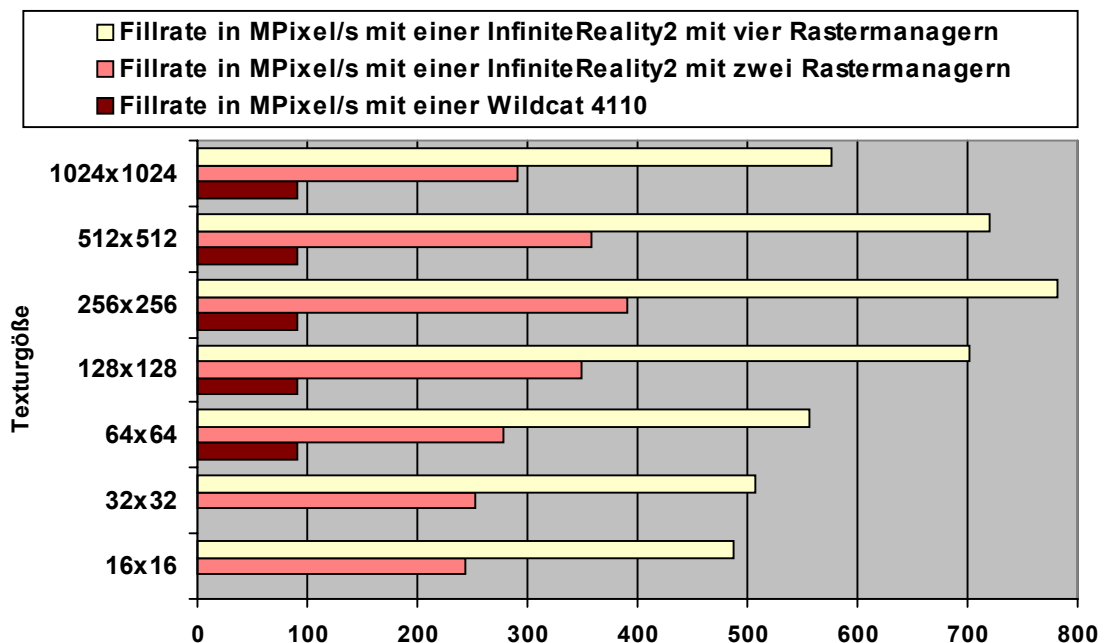


Abb. 6-5 2D-Textur-Fillrate auf einer ONYX2 und einer Wildcat 4110

Auch bei der Fillrate läßt sich ein Zusammenhang zwischen der Texturgröße und Geschwindigkeit ermitteln. Die Abhängigkeit ist nicht so deutlich ausgeprägt wie bei dem Texturdurchsatz, ist jedoch erkennbar vorhanden. Es ist Festzustellen, daß auf der ONYX2

höher aufgelöste Texturen schneller gezeichnet werden als geringer aufgelöste Texturen. Die Messungen wurden mit zwei und vier Rastermanagern durchgeführt. Eine ONYX2 kann mit einem, zwei oder vier Rastermanagern ausgerüstet werden. Diese Rastermanager setzen die Pixel beim Zeichnen. Werden mehrere Rastermanager eingesetzt, werden diese parallel verwendet. Bei den Messungen ist bei der Verwendung von vier Rastermanagern eine Verdoppelung der Leistung gegenüber zwei Rastermanagern zu erkennen. Sehr deutlich ist auch ein Leistungsabfall bei der Verwendung von 3D-Texturen gegenüber 2D-Texturen zu erkennen. Dies läßt sich mit einem erhöhten Rechenaufwand bei der Texturfilterung erklären. Trotzdem ist der Leistungsabfall sehr hoch.

Die Intense3D Grafikkarte wird durch verschiedene Texturgrößen nur wenig in der Leistung beeinflusst. Auch ist der Leistungsverlust bei der Verwendung von 3D-Texturen gegenüber 2D-Texturen nicht so hoch wie bei der ONYX2.

Die Fillrate beim Zeichnen von 3D-Texturen ist nur geringfügig schlechter als bei der wesentlich teureren ONYX2.

7 GigaVol Benchmarks

In diesem Kapitel werden Geschwindigkeitsmessungen mit der OpenGL- und der Avango-Version von GigaVol durchgeführt.

7.1 OpenGL Version von GigaVol

Um die OpenGL-Version der Octree-Volumenrenderingsoftware leicht anwenden und testen zu können, wurde sie in eine graphische Benutzeroberfläche integriert. Für die Benutzeroberfläche wurde das frei verfügbare FLTK [4] verwendet, welches eine gute OpenGL-Anbindung enthält. FLTK ist sowohl für die UNIX- und die Windows-Plattform verfügbar. Die OpenGL-Version von GigaVol ist für IRIX, Linux und Windows verfügbar. GigaVol bietet die Möglichkeit, Volumendatensätze in fünf verschiedenen Formaten zu laden. Die erzeugten Bricks lassen sich im eigenen „Brickset“-Format speichern. Das „Brickset“-Format wurde entwickelt, um die für das Erzeugen der Bricks benötigte Zeit zu sparen. Mit der OpenGL-Version ist eine einfache Darstellung von Volumen oder Schnittebenen möglich. Neben der 32Bit-IRIX-Version gibt es auch eine 64Bit-Version, um Volumen verwalten zu können, die größer als 2GB sind.

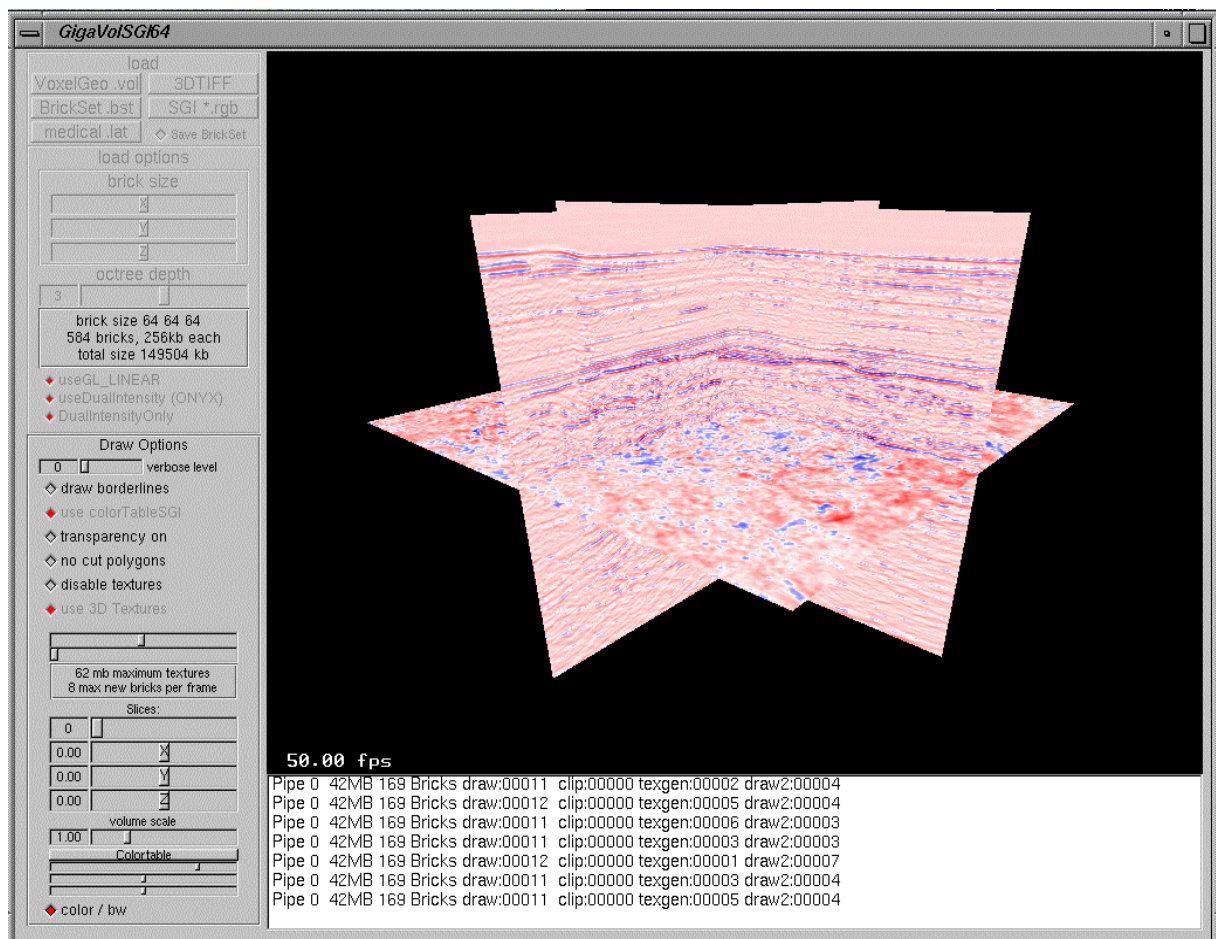
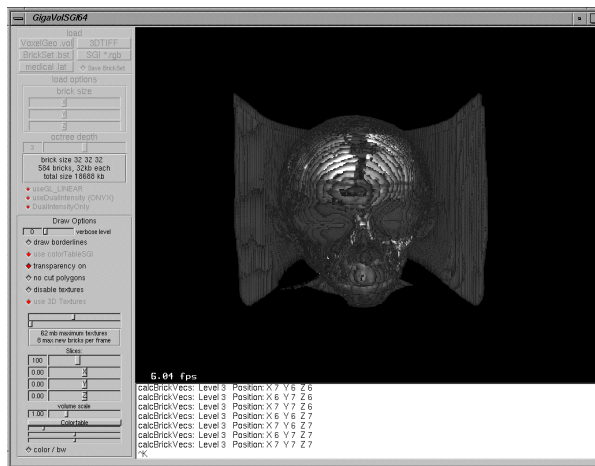


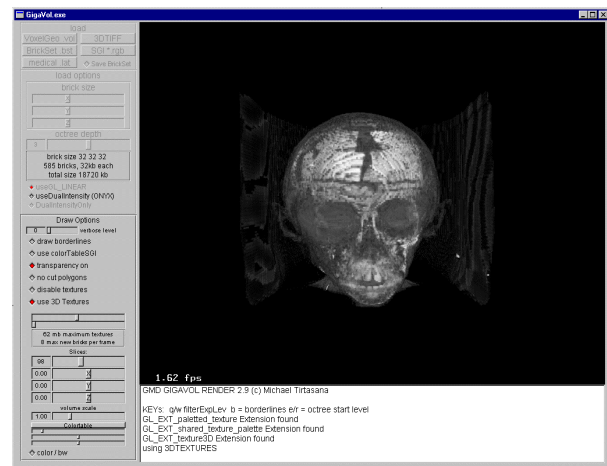
Abb. 7-1 Darstellung von geologischen Daten mit drei Schnittebenen unter IRIX

Darstellung eines Volumens mit 100 Ebenen von GigaVol unter IRIX auf einer ONYX2 und unter Windows auf einer Intense3D Wildcat 4110.

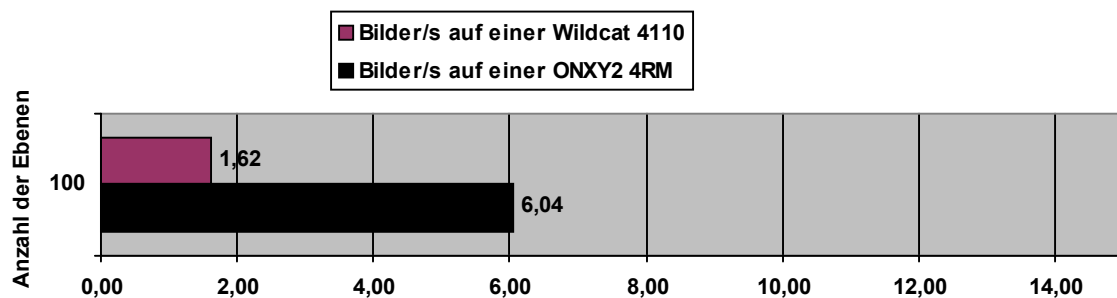
Das Volumen hat die Dimensionen 128x128x128. Das Volumen belegt im höchsten Octree-Level 16MB und paßt daher vollständig in beide Texturspeicher.



100 Ebenen unter IRIX auf einer ONYX2

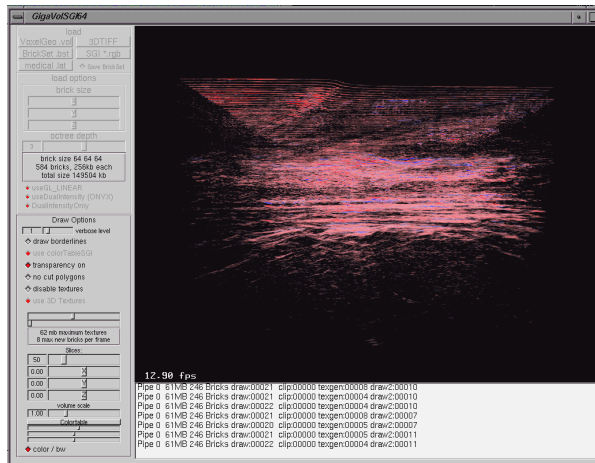


100 Ebenen unter Windows auf einer Intense3D Wildcat 4110

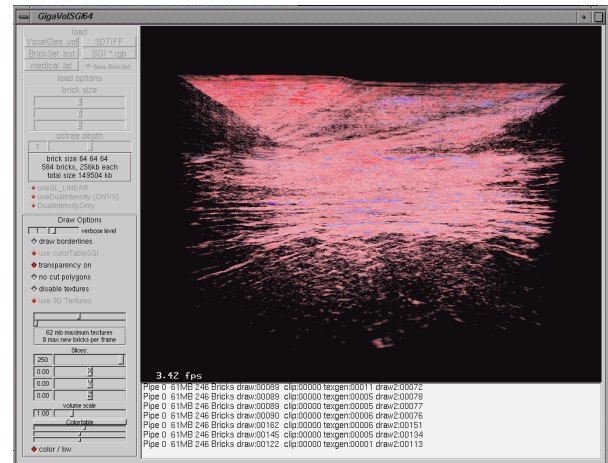


Erwartungsgemäß erzielt eine ONYX2 mit dem Maximalausbau von vier Rastermanagern eine höhere Leistung als die Wildcat. Das Verhältnis der Geschwindigkeit entspricht dem, was bei den Messungen der Textur-Fillrate ermittelt wurde.

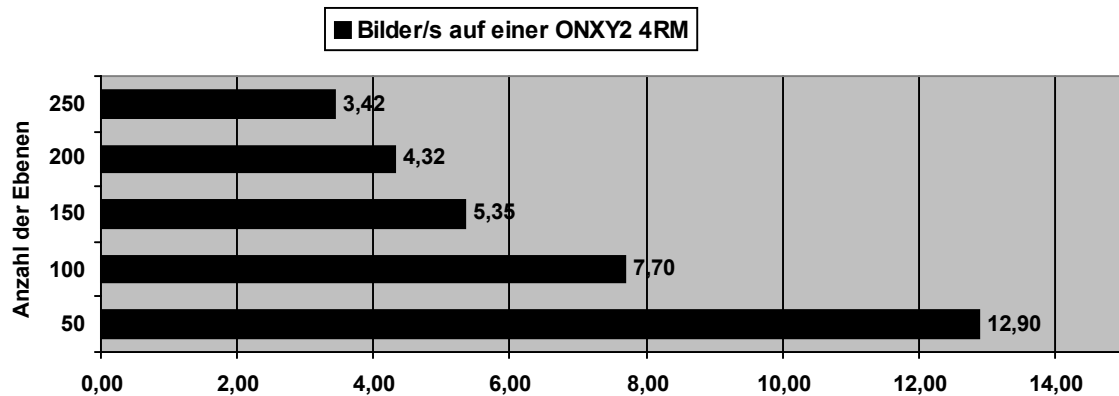
Darstellung eines Volumens mit 50 bis 250 Ebenen auf einer ONYX2.



Darstellung mit 50 Ebenen



Darstellung mit 250 Ebenen



Die Anzahl der Ebenen bestimmt erwartungsgemäß maßgeblich die Geschwindigkeit. Da man in Abhängigkeit der Bildwiederholrate unterschiedlich lange auf den Monitorzeilenrücklauf warten muß, ist das Verhältnis der Geschwindigkeit zur Anzahl der Ebenen nicht linear.

7.2 Avango-Version von GigaVol

Die Avango-Version von GigaVol ist für den praktischen Einsatz entwickelt worden. Diese Version verwendet das von GigaVol angebotene Multiprocessing und besitzt unter anderem folgende Darstellungsoptionen:

- Verwendung des „cull process“ für die Berechnung der optimierten Wunschlisten
- Benutzung von OpenGL-Clippingplanes oder Erzeugung neuer Polygone
- Begrenzung des verwendeten Texturspeichers
- Setzen des Zuladelimits

Avango kann eine genaue Geschwindigkeitsstatistik ausgeben, die für die Tests ausgewertet wurde.

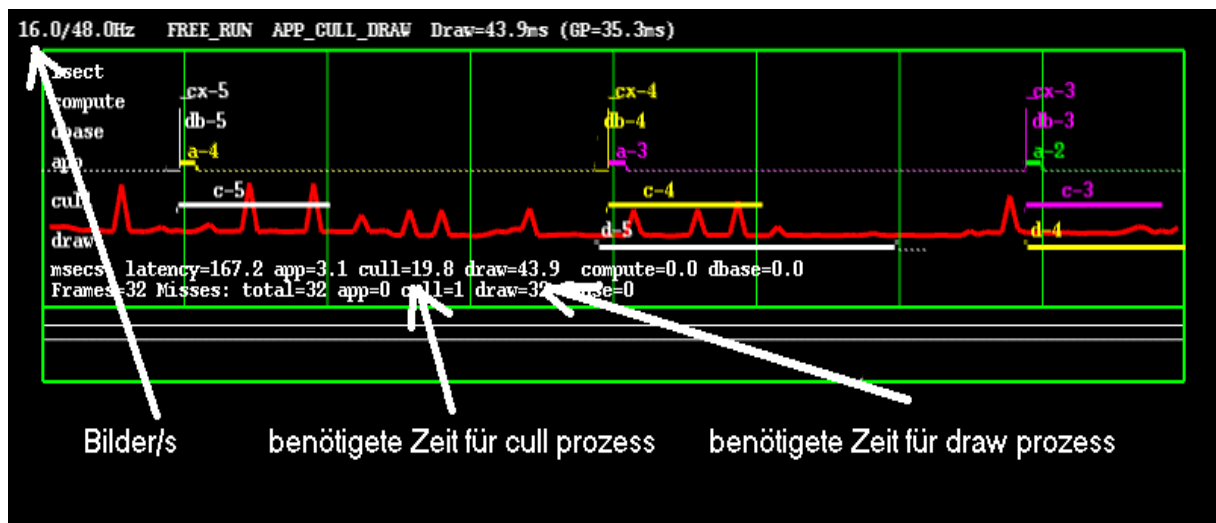


Abb. 7-2 Beispiel für eine Geschwindigkeitsstatistik von Avango

Bei den Tests werden vier unterschiedliche, geschwindigkeitsbeeinflussende Einstellungen von GigaVol verwendet:

- Verwendung des „cull process“ und Erzeugung neuer Polygone (**CP an PC an**).
- Verwendung des „cull process“ und keine Erzeugung neuer Polygone (**CP an PC aus**).
- Keine Verwendung des „cull process“ und Erzeugung neuer Polygone (**CP aus PC an**).
- Keine Verwendung des „cull process“ und keine Erzeugung neuer Polygone (**CP aus PC aus**).

Visualisierung von 128MB geologischen Volumendaten durch

- drei zueinander orthogonalen Schnittebenen
- Darstellung des ganzen Volumens mit 64 transparenten Schnittebenen
- vorberechneten, polygonalen Interpretationsergebnissen

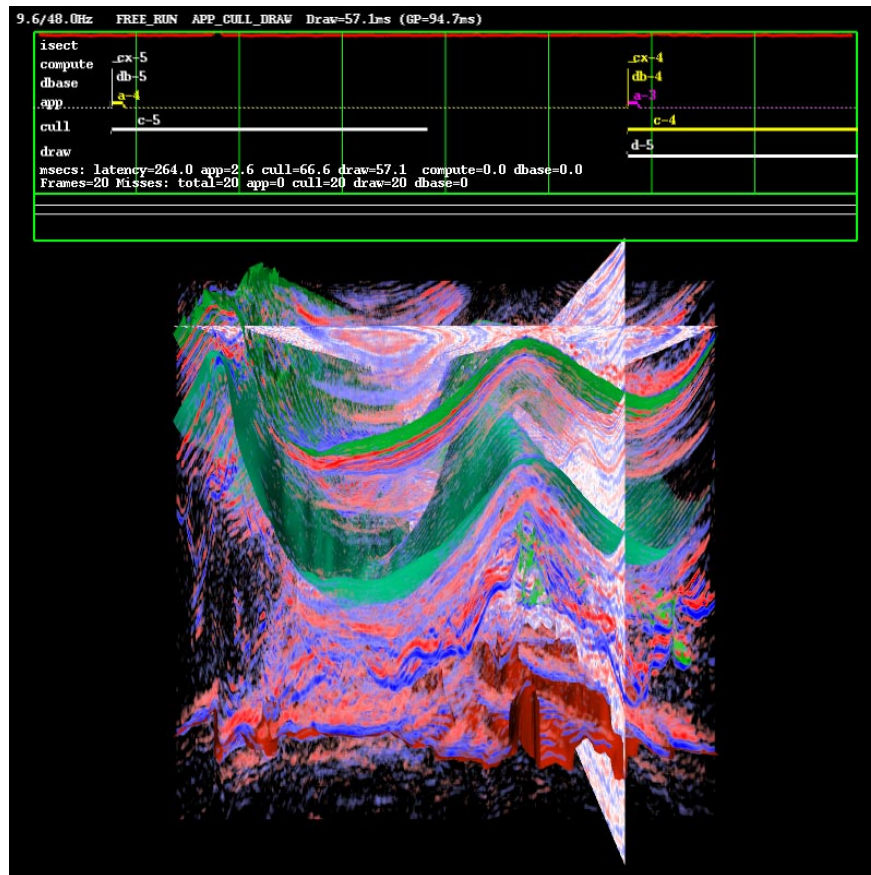


Abb. 7-3 Eine Avango Darstellung mit 800x800 Pixel.

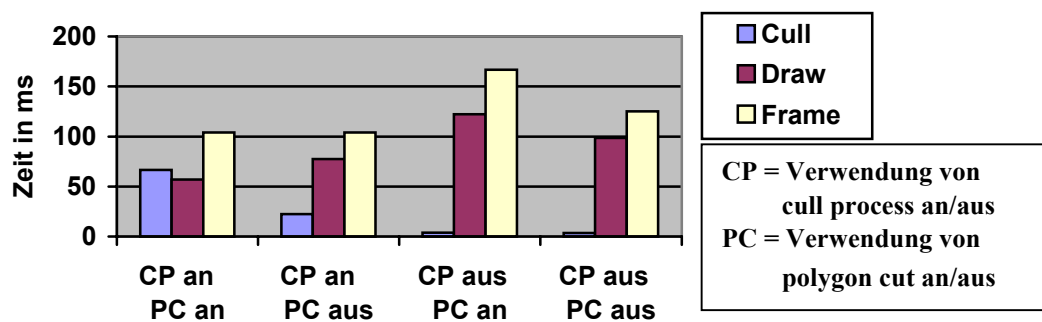


Abb. 7-4 Die verschiedenen Berechnungszeiten einer ONYX2 mit 2RM in Millisekunden.

Die Verwendung des „cull process“ bringt bei dieser Darstellung große Geschwindigkeitsvorteile. Auch das Zuschneiden der Polygone bringt Geschwindigkeitsvorteile im „draw process“.

Visualisierung von 128MB geologischen Volumendaten durch

- drei zueinander orthogonalen Schnittebenen
- vorberechneten, polygonalen Interpretationsergebnissen

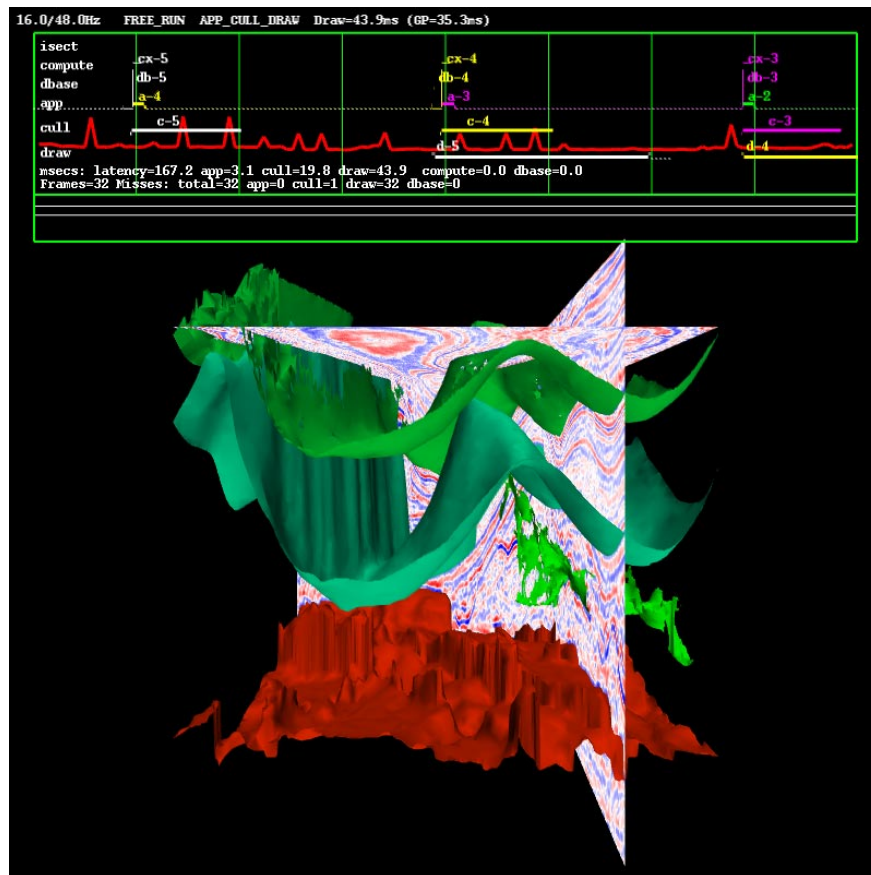


Abb. 7-5 Eine Avango Darstellung mit 800x800 Pixel.

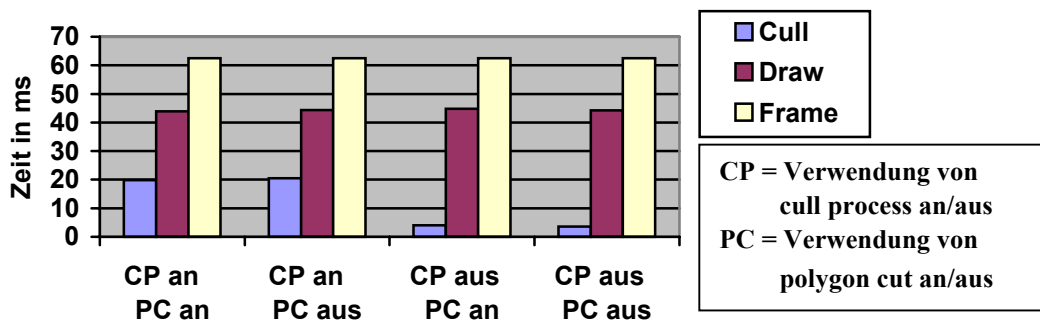


Abb. 7-6 Die verschiedenen Berechnungszeiten ONYX2 mit 2RM in Millisekunden.

Die Verwendung des „cull process“ bringt bei dieser Darstellung keine Geschwindigkeitsvorteile, da der GigaVol-Anteil bei der Bildberechnung nicht so hoch ist.

Visualisierung von 128MB geologischen Volumendaten durch

- drei zueinander orthogonalen Schnittebenen
- Darstellung des ganzen Volumens mit 64 transparenten Schnittebenen

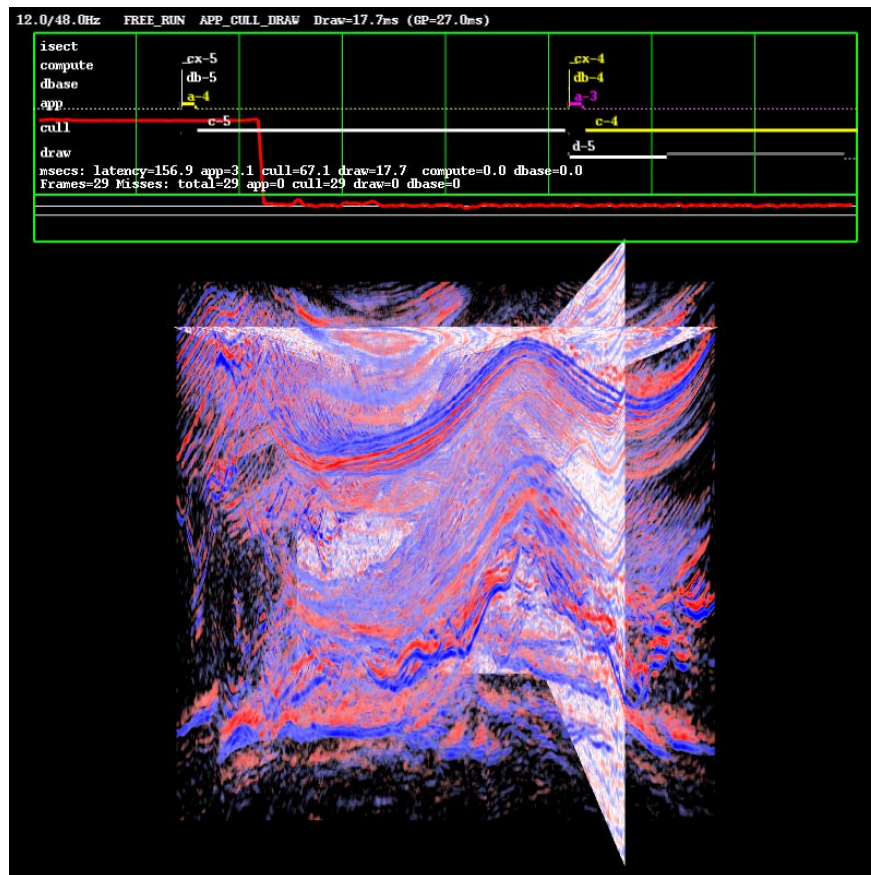


Abb. 7-7 Eine Avango Darstellung mit 800x800 Pixel.

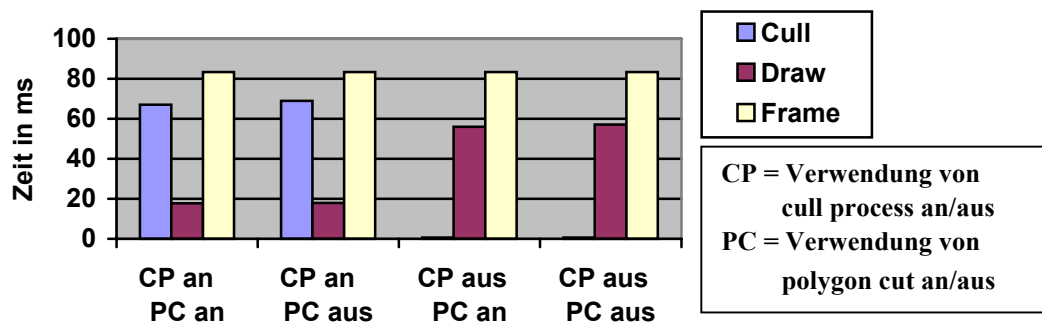


Abb. 7-8 Die verschiedenen Berechnungszeiten ONYX2 mit 2RM in Millisekunden.

Bei dieser großen Volumendarstellung ohne weitere Bildelemente arbeitet die ONYX2 am „fill limit“ und verschiedene Darstellungstechniken bringen keine Geschwindigkeitsvorteile.

Visualisierung von 128MB geologischen Volumendaten, durch Darstellung des ganzen Volumens mit 64 transparenten Schnittebenen, bei verschiedenen Bildauflösungen.

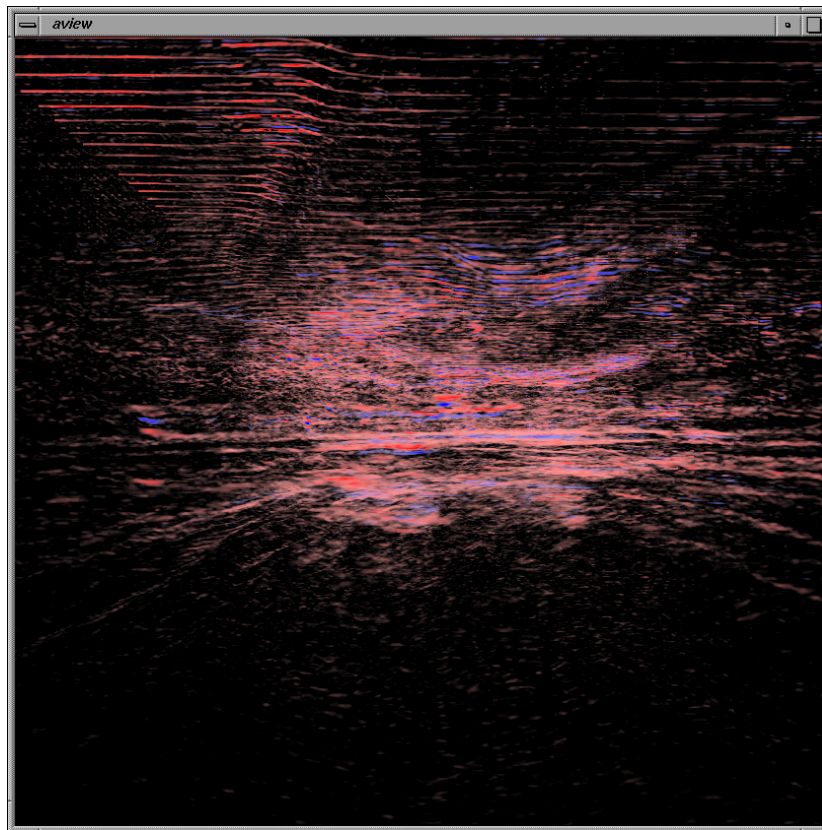


Abb. 7-9 Eine Avango Darstellung

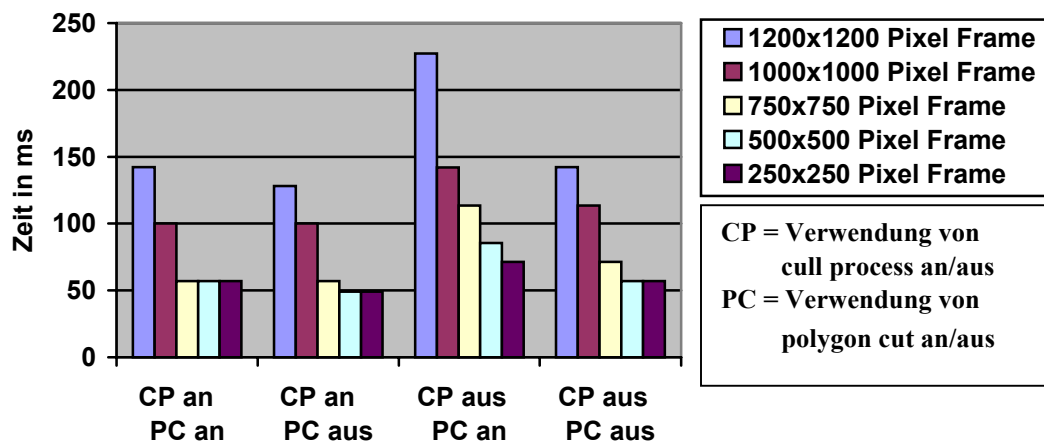


Abb. 7-10 Die verschiedenen Berechnungszeiten für einen Frame auf einer ONYX2 mit 4RM in verschiedenen Auflösungen.

Bei Auflösungen über 500x500 werden die Berechnungszeiten für ein Frame durch das „fill limit“ bestimmt. Bei Auflösungen unter 500x500 wird mehr Zeit für die Verwaltung der Bricks benötigt als für das Zeichnen.

8 Praktischer Einsatz von GigaVol im VRGeo Projekt

Das VRGeo-Projekt [11] der GMD wird von einem Konsortium aus Firmen aus der Öl- und Gas-Industrie unterstützt. Mitglieder des Konsortium waren Arco, Amoco, BHP, EXXON, Landmark, Mobil, Saga, Schlumberger, Shell, Smedvig und Statoil. Ziel war es, die Entwicklung und Anwendung von Virtual Reality (VR) in der Öl- und Gas-Industrie zu erforschen.

Bei der Darstellung von Öl- und Gas-Datensätzen müssen sehr große Volumendaten in Echtzeit dargestellt werden. Diese Datensätze mußten bisher auf die Größe des Texturspeichers verkleinert werden, um dargestellt zu werden.

GigaVol ermöglicht erstmals eine Echtzeitdarstellung, ohne daß die Datensätze reduziert werden müssen.

Hierzu wurde die Avango-Version von GigaVol in die VRGeo-Applikation integriert. Diese wird derzeit bei Shell, Statoil und der GMD angewendet. Des weiteren wurde die Applikation auf diversen Konferenzen wie der SIGGRAPH 99 und der SEG99 präsentiert. Die Softwarefirmen Landmark und Schlumberger arbeiteten an einer möglichen Integration der OpenGL-Version in ihre Anwendungen.

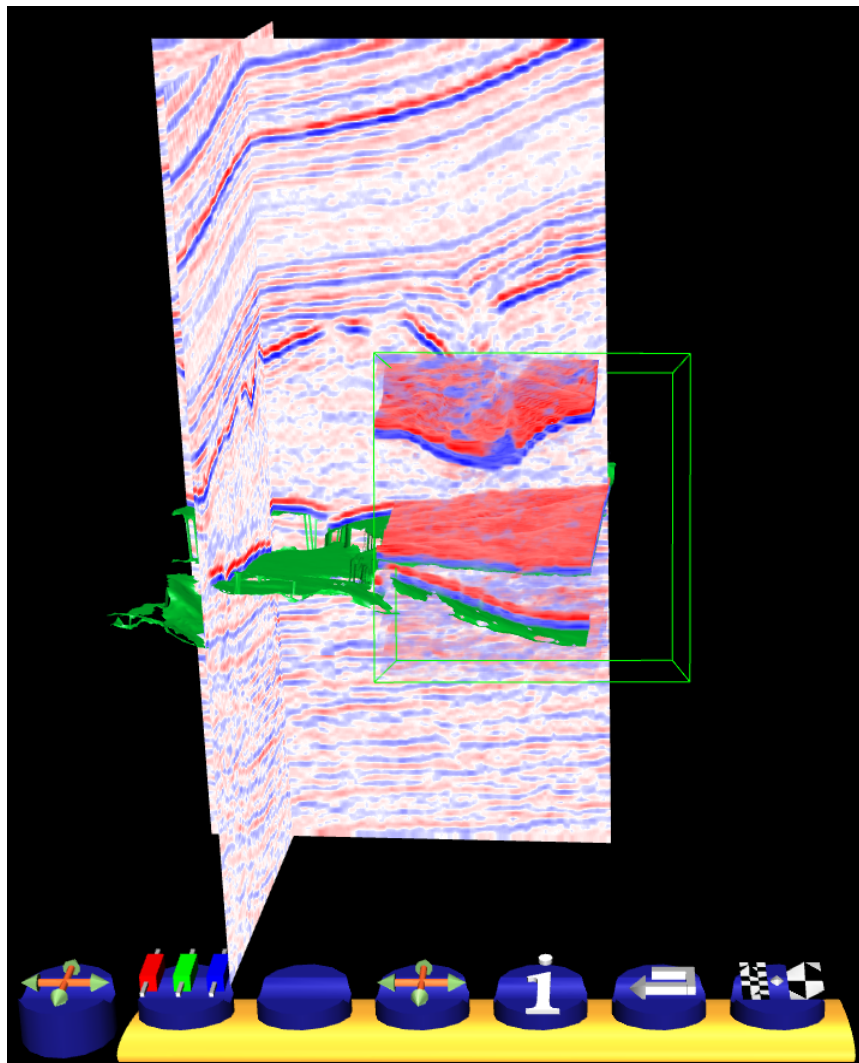


Abb. 8-1 Bildschirmfoto der VRGeo-Applikation

9 Zusammenfassung und Ausblick

Die Aufgabe war es, ein Verfahren zu entwickeln, das auf den existierenden Grafikworkstations eine Darstellung von mehreren Gigabyte großen Volumen in Echtzeit ermöglicht.

9.1 Ergebnis der Arbeit

Der Teilaspekt der Darstellung von sehr großen Datensätzen konnte vollständig erfüllt werden. Die Datensätze konnten ca. 70% der Größe des Hauptspeichers betragen, welchen man auf einer ONYX2 auf mehrere Gigabyte ausbauen kann. Auch der Aspekt der Echtzeitfähigkeit konnte bis zur Leistungsgrenze auf den existierenden Grafikworkstations erfüllt werden. Leider ist die Leistungsfähigkeit der existierenden Hardware für das Volumenrendern nicht immer zufriedenstellend. Einer ONYX2 war es zwar möglich auf einer monoskopischen Projektion gute Ergebnisse zu liefern. Die Darstellung auf stereographischen VR-Projektionssystemen, wo die Anforderungen an die Geschwindigkeit mehr als doppelt so hoch sind, konnte aber teilweise nur durchschnittliche Ergebnisse liefern. Das Problem liegt an der unzureichenden Raster-Leistung der ONYX2 und anderen Grafikworkstations. Wird bei normalen 3D-Darstellungen ein Pixel des Bildes ca. drei bis acht mal neu bearbeitet und gezeichnet, so muß bei der Volumendarstellung ein Pixel ca. 32 bis 100 mal bearbeitet und gezeichnet werden. Dafür ist die Fillrate einer ONYX2 mit effektiv 100 – 200 MPixel nicht ausreichend. Die Größe eines Texturspeichers war durch das verwendete Octree-Verfahren kein limitierender Faktor mehr. So waren die 64MB einer ONYX2 ausreichend für eine Bildauflösung von 1024x768 Pixel. Auch die Zeit, die für das Einlagern der Texturen benötigt wird, konnte subjektiv für den Anwender eliminiert werden.

Das Volumenrendern stellte auch maximale Anforderungen an die ONYX2 Hardware. So waren häufige Abstürze, die im alltäglichen Betrieb selten vorkamen, auf einen Defekt in der Hardware zurückzuführen.

Erst zukünftige Hardware-Generationen versprechen eine bessere Leistung für das Volumenrendern. Neben der Leistungssteigerung können auch technische Innovationen eine Verbesserung bringen. So wäre es möglich, mit Texturkompression ein größeres Volumen zu bearbeiten.

Eine große Herausforderung bei der Implementierung war die Integration der Software in ein Multiprozessorsystem wie die ONYX2. Verschiedene Prozesse mußten verwaltet und die Software deswegen oft angepaßt werden.

9.2 Mögliche Weiterentwicklungen von GigaVol.

Eine Verbesserung des Systems wäre eine intelligentere Texturspeicherverwaltung. Bisher werden die Bricks in den Texturspeicher geladen, wenn sie in dem aktuellen Frame benötigt werden. Die Bricks könnten im voraus in den Texturspeicher geladen werden, wenn vermutet wird, das sie vielleicht in einem der folgenden Frames gebraucht werden („prepaging“).

Um diese Methode zu realisieren, muß eine Technik entwickelt werden, die vorhersagt, welche Bricks wahrscheinlich als nächstes benötigt werden. Beim Volumenrendering verwendet man meistens eine oder mehrere Ebenen, die durch das Volumen bewegt werden. Die Bewegung und Geschwindigkeit dieser Ebenen kann als Berechnungsgrundlage dienen, welche Bricks in den Texturspeicher geladen werden.

Eine weitere Verbesserung würde ein dreistufiges Speicherkonzept darstellen. Dabei werden die Bricks für sehr große Datensätze vorberechnet und auf der Festplatte gespeichert. Diese Bricks werden dann nur bei Bedarf in den Hauptspeicher geladen, der nur eine begrenzte Anzahl von Bricks speichert. Der Hauptspeicher würde hierbei als eine Art „cache“ dienen. Für diese Speicherverwaltung würde ein „prepaging“-Mechanismus eine wichtige Rolle spielen.

10 Literaturverzeichnis

[1] OpenGL

OpenGL Datasheet The Industry's Foundation for High-Performance Graphics

<http://www.opengl.org/About/About.html>

[2] Avango(ehemals Avocado)

Tramberend, Henrik (März 1999): Avocado: A Distributed Virtual Reality Framework. IEEE Virtual Reality, S. 14-21.

<http://www.avango.org>

[3] Intense3D Wildcat 4110

Intense3D Wildcat 4110 Datasheet

<http://www.intense3d.com/Wildcat4110.asp>

[4] Fast Lightning Tool Kit

Bill Spitzak, FLTK a LGPL'd C++ graphical user interface toolkit for X, OpenGL, and WIN32.

www.fltk.org

[5]Volumizer API

George Eckel OpenGL ® Volumizer Programmer's Guide

Document Number 007-3720-001

<http://techpubs.sgi.com/library/manuals/3000/007-3720-001/pdf/>

[6] 3D-Texturen

Silicon Graphics DEVELOPER'S TOOLBOX

EXT_texture3D--The 3D Texture Extension

<http://toolbox.sgi.com/TasteOfDT/documents/OpenGL/OpenGLonSGS/OpenGLonSGI-46.html>

[7] Elf Frankreich Exploration

Toward New Horizons

<http://www.elf.fr/exploprod/reportag/us/grfonds/explo>

[8] Multiresolution Techniques for InteractiveTexture-Based Volume Visualization

Eric LaMar, Bernd Hamann, Kenneth I. Joy University of California, Davis 95616-8562

IEEE Visualization, S. 355.

[9] ONYX2

Onyx2 Technical Specifications

http://www.sgi.com/onyx2/tech_specs.html

[10] Michael Tirtasana Homepage 3D Programs

<http://viswiz.gmd.de/~mic/prog3d.html>

[11] Fröhlich, Bernd, Stephen Barrass, Björn Zehner, John Plate, Martin Göbel (Oktober

1999): Exploring Geo-Scientific Data in Virtual Environments. IEEE Visualization, S. 169-173.