

Strategien und Algorithmen zur interaktiven Volumenvisualisierung in Digitalen Dokumenten

Von der Fakultät Informatik der Universität Stuttgart
zur Erlangung der Würde eines Doktors der
Naturwissenschaften (Dr. rer. nat.) genehmigte Abhandlung

Vorgelegt von

Klaus Dieter Engel

aus Nördlingen

Hauptberichter: Prof. Dr. T. Ertl
Mitberichter: Prof. Dr. R. Westermann

Tag der mündlichen Prüfung: 31.5.2002

Institut für Informatik der Universität Stuttgart
2002

Jedes Wort ein wahres Juwel ...
was mich beunruhigt, ist nur die Reihenfolge,
in der sie hier aneinandergesetzt wurden.

Terry Jones, Monty Python

Kurzfassung

In Forschung und Lehre hat sich in den letzten Jahren das Internet als wichtigstes Kommunikationsmedium etabliert. Neben der Mensch-zu-Mensch-Kommunikation und der Rolle des Internets als Publikationsmedium erlaubt das Internet auch den Zugriff auf eine enorme Menge wissenschaftlicher Daten und Algorithmen. Hauptziel der wissenschaftlichen Visualisierung ist es, die in den Daten vorhandenen wesentlichen Informationen durch fortgeschrittene Darstellungstechniken den Menschen zugänglich zu machen. Dabei spielt eine interaktive Darstellung, also die unmittelbare Reaktion bei Modifikation von Visualisierungsparametern, eine Schlüsselrolle. Vor allem im Bereich eines der wichtigsten wissenschaftlichen Datentypen, den skalaren Volumendaten, sind in den letzten Jahren durch Fortschritte in der Mess- und Simulationstechnik die produzierten Datenmengen explosionsartig gewachsen, so dass zur Verarbeitung und Visualisierung gewaltige Rechen- und Speicherressourcen notwendig sind. Zwar kann über das Internet auch auf eine große Menge von Ressourcen in Form von Hochleistungsrechnern, Rechner-Clustern, Workstations und PCs zugegriffen werden. Allerdings verlangen limitierte Übertragungsbandbreiten im Internet neue Verfahren zur optimalen Nutzung entfernter und lokaler Infrastruktur.

Diese Arbeit zeigt neue Strategien und Algorithmen auf, die eine interaktive Visualisierung umfangreicher wissenschaftlicher Volumendaten auf einer Vielzahl unterschiedlicher Client-Architekturen und Netzwerkverbindungen in digitalen Netzwerken ermöglichen. Dies wird auf unterschiedliche Weise erreicht, wobei oftmals eine Kombination verschiedener Algorithmen und Strategien zum Erfolg führt. Bei der Implementierung der vorgestellten Verfahren wird dabei großer Wert auf Architekturneutralität und die Verwendung von Internet-Standards gelegt.

Zur Erreichung des Ziels der Interaktivität ist zunächst die Suche einer geeigneten Aufteilung der Teilaufgaben der Visualisierungspipeline auf Client- und Server-Rechner erforderlich. Dabei ist die Schnittstelle so zu wählen, dass sowohl die zu übertragende Datenmenge möglichst klein gehalten werden kann als auch vorhandene Rechen- und Graphikkapazitäten der beteiligten Systeme optimal genutzt werden.

Eine entscheidende Rolle spielt dabei in dieser Arbeit die Nutzung schneller, optimierter Graphikhardware. Dabei wird sowohl auf die Verwendung Server-seitig vorhandener Spezialhardware wie auch auf die Ausschöpfung Client-seitiger PC-Graphikhardware eingegangen. Gerade der gewaltige Leistungszuwachs hinsichtlich Verarbeitungsgeschwindigkeit, Qualität und Flexibilität moderner PC-Graphikhardware erweist sich als Kristallisationspunkt neuer Algorithmen und Methoden. Neben der Optimierung der Rendering-Verfahren lassen sich auch andere Aufgaben der Visualisierungspipeline effektiv in die Graphikhardware abbilden. Demonstriert wird dies anhand direkter Volumenvisualisierungsalgorithmen, die ausgiebig Nutzen von Mehrstufen-

rasterisierungseinheiten moderner Graphikhardware machen. Ein wesentlicher Beitrag ist in einem völlig neuen Volume-Rendering-Verfahren zu sehen, das hohe Abstraten bei nicht-linearen Transferfunktionen durch eine Vorintegration von Strahlsegmenten vermeidet. Eine Hardwarebeschleunigte Implementierung dieses Algorithmus ist in neuester PC-Graphikhardware durch abhängige Texturzugriffe effizient möglich. Der Ansatz basiert auf einem neuen Klassifikationschema für Volumendaten, für das sich der Begriff Vorintegrierte Klassifikation etabliert hat.

Einen weiteren Schwerpunkt bilden adaptive und progressive Methoden, die durch eine Repräsentation der umfangreichen Volumendaten in einer Hierarchie von Auflösungsstufen die gewaltigen Datenmengen handhabbar machen. Dabei dünnen diese Verfahren die Daten bereits im Vorfeld durch „intelligente“ Algorithmen so weit aus, dass die Übertragung und Darstellung der Daten später problemlos vonstatten gehen kann. Dies wird anhand einer adaptiven, auf dem lokalen Approximationsfehler basierenden Finiten-Elemente-Methode demonstriert, die eine globale, progressive Extraktion, Übertragung und Darstellung komplexer Isoflächen erlaubt. Im Gegensatz dazu steht eine Methode, mittels derer selektiv einzelne Teilbereiche einer Isofläche verfeinert werden können. In diesem Zusammenhang wird auch auf die Verringerung der Datenmengen durch eine geschickte Modularisierung und eine verbesserte Isoflächenrepräsentation eingegangen.

Mit Hilfe der vorgestellten Methoden können Volumendaten in digitalen Dokumenten mit bisher unerreichter Qualität und Geschwindigkeit in interaktiven Bilderwiederholraten dargestellt werden. Diese Ergebnisse werden deutliche Wirkung auf weitere Forschungsaktivitäten haben und eröffnen gleichzeitig ein breites Anwendungsfeld in den Ingenieurwissenschaften, der Medizin, der Physik, der Chemie sowie Bildung und Lehre.

Every little word a true jewel ...
what troubles me is just the sequence
in which they were arranged here.

Terry Jones, Monty Python

Abstract

Over the past years the Internet has evolved to the most important medium of communication for science and teaching. Besides the man-to-man communication and the role of the Internet as a publication medium, the Internet provides access to huge amounts of scientific data and algorithms. The main focus of scientific visualization is, to provide people insights into the main information contained within scientific data through advanced visualization techniques. Withal interactivity, generally speaking the immediate response to changes of visualization parameters, plays a key role. Particularly in the field of one of the most important scientific data types, the scalar volume data sets, latest progress in measurements and simulations lead to explosion-like increases in data sizes. This induces the need for enormous computation and storage resources for processing and visualization. In fact, the Internet provides access to huge amounts of resources in terms of high-performance computers, computation clusters, workstations and PCs. However, limited bandwidths, storage and processing capacities are in many cases a limiting factor.

This dissertation reveals new strategies and algorithms that enable the interactive visualization of substantial volumetric data sets on a wide variety of client-architectures and over different internet-connections. This is accomplished in various ways, where the combination of different algorithms and strategies often leads to success. Regarding the implementation, special attention has been paid to platform-independence and the use of Internet standards.

To accomplish the goals of interactivity, initially, an appropriate partitioning of the subtask of the visualization pipeline onto clients and servers has to be found. Here the interface has to be chosen with respect to minimal data transfer and to optimal utilization of the computation and graphics resources of the participating systems at the same time.

First, the new term „client-server strategies“ is introduced, which provides a fundamental classification scheme for the distinction of the different visualization task distribution approaches in digital networks. In substantial three different strategies are worked out:

Client-side approaches shift the modules of the visualization pipeline onto a local personal computer, i.e. both the data to be visualized and the necessary visualization algorithm are transferred to a client. The essential advantages of this strategy are that locally available memory, graphics and computation resources can be employed, which have developed rapidly over the past few years. This involves unloading of server-side resources, an important matter for visualization services used by a larger number of users at the same time. Furthermore, because of the local interaction with the data, no further latencies due to data transfer occur. However, the transfer of the original visualization data might not be possible or disirable due to limited network bandwidth, limited memory resources on the client side or security reasons. Additionally, locally available computation and graphics resources might not be capable to process the data.

The antipode to client-side strategies are server-side strategies, which relocate the modules of the visualization pipeline onto a dedicated high-performance server. The produced 3D visualizations are streamed to clients and displayed using simple 2D graphics hardware. For interaction purposes with the remote visualization, locally generated mouse and keyboard events are sent back to the server and processed. This approach provides the possibility to use remote specialized hardware, like supercomputers or graphics workstations, with a simple client. There is no need to transfer or duplicate the original data. On the other hand, the server-side must provide sufficient capacities. Consequently, a shortage of server resources for a larger number of users might occur. Additionally, due to the transfer of interaction events to the server and the streaming of images back from the server, latencies are inevitable.

Finally, the goal of hybrid strategies is to find a well-balanced partition of the modules and submodules of the visualization pipeline onto client and server, to optimally utilize the available resource. It is preferable to use a data minimum in the visualization pipeline for the necessary data transfer in this scenario. Similar to server-side approaches, this strategy does not require the transfer of original data. Unfortunately, latencies cannot be avoided for certain visualization parameter modifications. The choice for an appropriate strategy for a specific visualization task depends on the availability of resources and sensitivity of the data to be visualized. In the case of a possible transfer of the original data and the availability of sufficient resources on the client side, a client-side strategy should be employed. Whenever a simple display device should be used in combination with specialized server hardware, a server-side strategy is applicable. In all other cases a hybrid strategy is the preferable choice, which utilizes client-side hardware to process and render data from the server, that has already been prepared. To make a visualization service available to a wide audience, with different network connections and client resources, a dynamic choice of the client-server strategy at the start of a visualization session is desirable. After the determination of network bandwidth, latencies and client architecture an appropriate algorithm is chosen and the required visualization pipeline modules are transferred to the client. In this context, the platform-independent implementation of the algorithms is another important aspect.

The question, whether and to which extent newest, flexible consumer graphics hardware can be employed for the visualization of volume data is one of the most relevant questions of this thesis. Because of the tight coupling of the developed algorithms to the programmable units of the consumer graphics hardware, the visualization data must be transferred to client. All processing is performed on the client side, thus with respect to the above nomenclature, the presented algorithms fall into the category of client-side strategies. First of all, the first generation of consumer graphics hardware was employed to embed texture-based visualizations of volume data into digital documents. A platform-independent implementation based on the Virtual Reality Modeling Language (VRML) and a Java applet was realized. The Java applet receives volume data from a data server and employs a VRML Plugin to visualize the data by means of 2D textures. For that, the data is sent to the corresponding nodes in the VRML scenegraph using the External Authoring Interface (EAI). These first encouraging results show the potential of PC-based volume graphics solutions.

Just now, the immense gain of capacity, regarding performance, quality and flexibility, of PC graphics hardware, is the crystallization point for new algorithms and methods. Besides the optimization of rendering techniques, other parts of the visualization pipeline can be mapped to gra-

phics hardware very efficiently. This is demonstrated by direct volume visualization algorithms, that make explicit use of multi-stage rasterization capabilities of modern graphics hardware. An algorithm for trilinear interpolation in 2D texture-based volume rendering approaches, employs multi-textures and programmable per-pixel operations to prevent undersampling artifacts by allowing higher sampling rates without memory overhead. Additionally techniques for fast shaded isosurfaces, speedup of rendering, volume shading, tagged volume rendering, clipping and temporal interpolation are presented. It is shown, that for volume data sets of moderate size, PC graphics hardware is significantly faster than high-end systems. Since only low-cost hardware is required, the presented approaches significantly contribute to the availability of interactive direct volume rendering in practice.

High accuracy in direct volume rendering is usually achieved by very high sampling rates resulting in heavy performance losses. A main contribution is a new texture-based volume rendering approach, that avoids additional slices by integrating non-linear transfer functions in a pre-processing step. A hardware-accelerated implementation is possible by means of programmable texture fetch operations, which are another significant enhancement of newest consumer graphics hardware. By employing these operations, pre-calculated values can be looked up during rendering. The method of pre-integrated volume rendering is based on the new pre-integrated classification scheme. Pre-integrated classification has the potential to improve the accuracy (less undersampling) and the performance (fewer samples) of a volume renderer at the same time. Besides semi-transparent renderings, the approach also allows rendering shaded volumes and isosurfaces with complex lighting conditions. The performance neither depends on the number of isosurfaces nor the definition of the transfer functions, and is therefore well suited for interactive high-quality volume graphics.

In contrast to direct volume approaches, isosurfaces still play an important role in volume visualization. The extraction of isosurfaces has positioned itself as a fundamental visualization technique and a lot of effort has been made during the last years to come up with optimized algorithms. Most importantly, the motivation in all these approaches is to develop algorithms that react to changes of mapping parameters (e.g. varying the iso-value) by almost immediately regenerating the corresponding geometrical representation which then ought to be rendered with several frames per second. Particularly in Web-based applications, where in general there's no access to large-scale computing environments, new techniques have to be developed that allow for an effective reduction of the computational expense of the reconstruction process and of the number of generated primitives to be viewed on affordable low-end computers. Adaptive and progressive methods make extensive volume data sets manageable by an advanced hierarchical multi-resolution representation. Preliminary to data transfer and rendering, these methods reduce the data size using „intelligent“ algorithms for better handling. This is demonstrated by means of an error-controlled finite element method, which allows for a global progressive extraction, transfer, and rendering of the isosurface. A user is able to interactively adjust the iso-value and the resolution of extracted iso-surfaces. As soon as an interesting iso-value is found, the user can refine the iso-surface depending on the performance of the client machine, hardware support and network load. In order to be platform independent on the client side, a Java applet running in a web-browser is used to reconstruct the hierarchy progressively. The visualization of one level of resolution is done by a VRML web-browser plug-in or the Java3D API, which permits the

employment of 3D graphics hardware for interactive updates.

The above method is ideally suited for getting a fast, global impression of the basic shape of an isosurface for large scale volume data sets. However, the main disadvantage is, that if a user is searching for a certain detail in a particular volume data set, the isosurface must be reconstructed, transferred and rendered with the highest level-of-detail. For that reason, the advantages of the progressive approach completely vanish. To solve this problem, a second algorithm allows for the selective refinement of user specified subareas of the volume data. In this context, the reduction of the amount of data through a smart modularization of the marching cubes visualization pipeline and an advanced isosurfaces representation is accentuated. Three methods for the direct reconstruction of stripped surface representations out of volume data are introduced, which reduce network load and increase render performance at the same time. Adaptive and hierarchical concepts minimize the number of vertices that have to be reconstructed, transmitted and rendered by means of an advanced octree representation of the volume data. The major contribution here is an algorithm that reduces the number of geometric primitives reconstructed in order to minimize computational load and network traffic. Remote interactive exploration of large scale volume datasets is achieved in this way. The proposed algorithms have been merged together in order to build a platform-independent Web-based application. Extensive use of VRML and Java OpenGL-bindings allows for the exploration of large-scale volume data quite efficiently.

Supplementary to the above methods, a server-side remote visualization approach, in terms of the framework, was developed to allow access high-end graphics servers with a thin-client. Images from the framebuffer of the graphics server, rendered with special 3D graphics hardware, are streamed to the client and displayed within the local 2D graphics display system. To allow interaction with the remote visualization, local user interface events are sent back to the visualization server. Applying these events to the visualization application on the server-side leads to new image sequences, which are streamed to the client in turn. The framework allows collaborative visualization sessions by allowing multiple connections to the server that share the same image stream. The value of the approach is demonstrated by two application scenarios. The first one is an application that uses 3D texture mapping hardware of high-end graphics workstations to visualize medical volume data. Data volumes from 3D medical imaging like CT are approaching sizes of 512^3 which amounts to more than 100 million voxel cells. Often, a local visualization of that kind of data is neither feasible nor possible due to secrecy of patient data. By applying the remote visualization framework medical volume data, stored on a central data server, can be accessed and interactively visualized using a Java-enabled web-client. The second example is an application that is used for visualization of the car development process for crash-worthiness simulations. The car bodies are represented by about 500.000, mainly four-sided, finite elements. During simulation the first 120 ms are computed and the coordinates of the deforming mesh are stored in 60 time steps together with the tracked parameters into a result file. Those result files often contain more than 1.5 GB of data. A local visualization, which implies transfer and processing of the data on a client PC, is not a reasonable option. By applying the remote visualization framework, collaborative working groups are able to discuss simulation results in distributed heterogeneous environments. There are two main advantage of this approach. First, the system requirements of the participating client systems are much lower. Second, with respect to security aspects, for example if third party engineers of subcontractors are involved, the pure data will

stay in-house; instead, just image data is transferred.

By way of the introduced methods, embedding of volumetric data into digital documents is now possible with yet unmatched quality and interactive performance. These results will have immediate impact to further scientific work and, at the same time, open up wide application areas for engineering science, medicine, physics, chemistry as well as education and teaching.

I don't care to belong to a club
that accepts people like me as members. ;-)

Groucho Marx, 1890-1977

Danksagung

Die Arbeiten zu dieser Dissertation entstanden zur Hälfte am Lehrstuhl für graphische Datenverarbeitung der Universität Erlangen-Nürnberg und in der Abteilung für Visualisierung und Interaktive Systeme der Universität Stuttgart. Dank des hervorragenden Arbeitsklimas und den netten Kolleginnen und Kollegen beider Gruppen habe ich mich zu jeder Zeit wohl gefühlt.

Diese Arbeit wurde im Rahmen des Schwerpunktprogramms „Verteilte Verarbeitung und Vermittlung digitaler Dokumente“ der Deutschen Forschungsgemeinschaft (DFG) im Teilprojekt „Chemische Visualisierung im Internet“ gefördert. Mein Dank gilt sowohl allen involvierten Personen bei der DFG als auch dem Initiator des Schwerpunktprogramms Prof. Dr. Dieter Fellner. Für die sehr erfolgreiche Zusammenarbeit möchte ich mich bei meinen Projektpartnern, Dr. Wolf-Dietrich Ihlenfeldt und Frank Oellien, herzlich bedanken.

Mein besonderer Dank gilt meinem Doktorvater und wissenschaftlichen Betreuer, Prof. Dr. Thomas Ertl, der mir zu jeder Zeit mit Rat und Tat zur Seite stand und wichtige Impulse, Ratschläge und Ideen für diese Arbeit lieferte. Danke sowohl für all die Freiheiten als auch für die manchmal notwendigen Spürkorrekturen.

Eine besondere Betonung verdienen natürlich alle Kollegen, mit denen ich in den letzten Jahren gemeinsam wissenschaftlich arbeiten und veröffentlichen durfte. Intensive Diskussionen brachten wesentliche Ideen zu Tage, die mit in diese Arbeit einfließen. Dies sind vor allem Dr. Roberto Grosso, Prof. Dr. Rüdiger Westermann, Ove Sommer, Dr. Martin Schulz, Martin Kraus, Dr. Bernd F. Tomandl, Dr. Daniel Weiskopf, Dr. Peter Hastreiter, Prof. Dr. Günther Greiner und Dr. Christof Rezk-Salama.

Bei der Fehlersuche und der Korrektur dieser Dissertation waren Sabine Iserhardt-Bauer, Simon Stegmaier und Manfred Weiler eine unentbehrliche Hilfe. Auch ihnen möchte ich an dieser Stelle herzlich danken.

Schließlich möchte ich allen Studenten, deren Arbeit mit in diese Dissertation eingeflossen ist, für die gute Zusammenarbeit danken. Zu nennen sind: Heike Pohl, Christian Ernst, Jürgen Beckmann, Michael Bauer und Guido Reina.

Danke an alle die ich hier vergessen habe - es ist nichts Persönliches.

Klaus Engel

Inhaltsverzeichnis

1	Einführung	23
1.1	Fragestellung	25
1.2	Gliederung der Arbeit	26
1.3	Beiträge der Arbeit	27
1.4	Betreute Studien- und Diplomarbeiten	30
2	Grundlagen	31
2.1	Visualisierung	31
2.1.1	Visualisierungspipeline	32
2.1.2	Gitter und Daten	33
2.2	Interaktive Computergraphik	34
2.2.1	Graphikhardware	34
2.2.2	3D-Graphikschnittstellen	37
2.2.3	Multi-Texturen	37
2.2.4	Programmierbare Rasterisierungseinheiten	38
2.3	Volumenvisualisierung	41
2.3.1	Indirekte Volumenvisualisierung	42
2.3.2	Direkte Volumenvisualisierung	43
2.3.3	Prä- und Post-Klassifikation	47
2.3.4	Numerische Integration	49
2.3.5	Texturbasiertes Volume-Rendering	50
2.3.6	Rasterisierungsisoflächen	53
2.4	Internet-Techniken	54
2.4.1	TCP/IP und Sockets	54
2.4.2	Objektbusse	55
2.4.3	Unicasting, Multicasting und Streaming	55
2.4.4	VRML/X3D	55
2.4.5	Java	56
2.4.6	Java3D	57
2.4.7	Java OpenGL Anbindungen	57
2.5	Wissenschaftliche Visualisierung im Internet	57
2.5.1	Anfänge	58
2.5.2	Visualisierungsdatentransfer	58

2.5.3	Softwaretransfer	59
2.5.4	Graphiktransfer	60
3	Client-Server-Strategien	61
3.1	Client-Server-Visualisierungspipeline	61
3.2	Client-seitige Strategien	61
3.3	Server-seitige Strategien	63
3.4	Hybride Strategien	64
3.5	Zusammenfassung	66
4	Volumenvisualisierung mit flexibler Rasterisierungshardware	67
4.1	Architekturneutrale Texturbasierte Visualisierung	68
4.1.1	Implementierung	68
4.1.2	Transfer-Funktion	72
4.1.3	Ergebnisse	72
4.2	Multi-Textur Volume-Rendering	73
4.2.1	Multi-Textur-Interpolation	74
4.2.2	Geschwindigkeitsoptimierung	77
4.2.3	Beleuchtete Isoflächen	79
4.2.4	Beleuchtung semi-transparenter Volumina	80
4.2.5	Weitere Anwendungen	81
4.2.6	Ergebnisse	82
4.3	Volume-Rendering mit Vorintegration	87
4.3.1	Vorintegrierte Klassifikation	87
4.3.2	Beschleunigung der Vorintegration	89
4.3.3	Texturbasiertes Vorintegriertes Volume-Rendering	90
4.3.4	Projektion	92
4.3.5	Texel-Fetch	93
4.3.6	Gradienteninterpolation	94
4.3.7	Beleuchtung	98
4.3.8	Probleme	99
4.3.9	Post-Klassifikation mit trilinearere Interpolation	99
4.3.10	Ergebnisse	101
4.4	Ergebnisse	104
5	Extraktionstechniken für Isoflächen	109
5.1	Lokale Isoflächenrekonstruktion	110
5.1.1	Algorithmus	110
5.1.2	Anwendungen in der Chemie	111
5.1.3	Anbindung an ein chemisches Datenmanagementsystem	111
5.1.4	Orbitalvisualisierung	113
5.1.5	Visualisierung Cryo-Elektronenmikroskopischer Dichten	119
5.1.6	Ergebnisse	120

5.2	Progressive Isoflächen	122
5.2.1	Einleitung	122
5.2.2	Multi-Resolution Volumenrepräsentation	123
5.2.3	Progressive Isoflächenextraktion	124
5.2.4	Progressive Isoflächenvisualisierung im WWW	124
5.2.5	Ergebnisse	127
5.3	Adaptive Echtzeit-Isoflächen	130
5.3.1	Einleitung	131
5.3.2	Rekonstruktion von Isoflächenstreifen	133
5.3.3	Octree-basierte Isoflächen	136
5.3.4	Verteilte Isoflächenrekonstruktion	137
5.3.5	Implementierung	140
5.3.6	Ergebnisse	141
5.4	Ergebnisse	144
6	Fern-Visualisierung	145
6.1	Grundlagen	146
6.2	Das Rahmenwerk	147
6.2.1	Server-Module	148
6.2.2	Client-Module	150
6.2.3	Datentransfer	151
6.3	Anwendungen	153
6.4	Hybride texturbasierte Visualisierung	154
6.5	Ergebnisse	155
7	Ergebnisse	159
8	Zusammenfassung und Ausblick	163
9	Farbseiten	171

Abbildungsverzeichnis

1.1	Visualisierung der MBone-Topologie	24
2.1	Minard's Karte des Russlandfeldzuges von Napoleon	32
2.2	Visualisierungspipeline	33
2.3	Rendering-Pipeline	35
2.4	NVIDIA Register-Combiner	38
2.5	RGB-Teil einer General-Combiner-Stufe	39
2.6	Final-Combiner-Stufe	40
2.7	Marching-Cubes-Konfigurationen	42
2.8	Ray-Casting mit Parallelprojektion	44
2.9	Splatting.	45
2.10	Fourier-Domain-Volume-Rendering	46
2.11	Parameter des Volume-Rendering-Integrals	46
2.12	Prä- und Post-Klassifikation	48
2.13	Parameter eines Strahlsegments	49
2.14	Viewport-parallele Schichten	51
2.15	Objekt-parallele Schichten	52
3.1	Client-Server-Visualisierungspipeline.	62
3.2	Client-seitige Strategie.	62
3.3	Server-seitige Strategie.	64
3.4	Hybride Strategie.	65
4.1	Texturbasierte Volumenvisualisierung mit VRML	69
4.2	VRML Szenengraph	70
4.3	Szenengraph des Java3D Volume-Renderers	71
4.4	Artefakte bei objekt-parallelen Schichten	74
4.5	Combiner-Einstellungen zur Interpolation von Zwischenschichten	75
4.6	Verfahren zur Rekonstruktion beliebig ausgerichteter Schichtbilder	76
4.7	Projektion angrenzender Texturschichten auf ein Schnittpolygon	77
4.8	Combiner-Einstellungen zum korrekten Überblenden zweier Schichten	78
4.9	Combiner-Einstellungen zum Darstellen beleuchteter Isoflächen	79
4.10	Combiner-Konfiguration zur Darstellung semi-transparenter Volumina	81
4.11	Multi-Textur-Clipping	83

4.12	Visuelle Artefakte durch das Fehlen trilinearer Interpolation	84
4.13	Beschleunigte Bildwiederholraten durch zweifach texturierte Schichtpolygone	85
4.14	Bildwiederholraten bei einer Datensatz-Größe von $128 \times 128 \times 64$ Voxeln.	85
4.15	Bildrate bei einer Datensatz-Größe von $256 \times 256 \times 128$ Voxeln	86
4.16	Bildraten bei einer Datensatz-Größe $256 \times 256 \times 256$	86
4.17	Abbildung eines <i>Slabs</i> des Volumens	91
4.18	Texture-Shader-Einstellungen für Vorintegration	94
4.19	Register-Combiner-Konfiguration zur Gradienteninterpolation	96
4.20	Alternative Register-Combiner-Konfiguration zur Gradienteninterpolation	97
4.21	Entfernung von Artefakten bei semi-transparenten Isoflächen	100
4.22	Dependent-Texture für trilineare Interpolation	101
4.23	Bildraten für Vorintegriertes Volume-Rendering	103
4.24	Bildwiederholraten beim Rendering eines Motorblock-Datensatzes	106
5.1	Schematische Darstellung des CACTVS-Systems	112
5.2	Wellenfunktionen der 1s-, 2s-, 2p- und 3d-Wasserstoff-Orbitale	114
5.3	Struktureditor und Energiestufen	115
5.4	Schematische Darstellung einer <i>OrbVis</i> -Visualisierungssitzung	117
5.5	Visualisierung eines Molekülorbitals mit <i>OrbVis</i>	118
5.6	Visualisierung einer Ribosomoberfläche	121
5.7	Reguläre und irreguläre Dreiecke	124
5.8	Progressive Isoflächenvisualisierung	125
5.9	VRML-Szenengraph für progressive Isoflächen	127
5.10	Java3D-Szenengraph für progressive Isoflächen	128
5.11	Progressive Druckisofläche	129
5.12	Verteilungsszenarios für den Marching-Cubes-Algorithmus	131
5.13	Dreieckstreifenrekonstruktion	133
5.14	Erweiterte MC-Tabelle	134
5.15	Zusätzlich erweiterte MC-Tabelle	135
5.16	Behebung von Diskontinuitäten	138
5.17	Adaptive Level-of-Detail Isoflächenrekonstruktion	139
5.18	Vergleich der Isoflächentechniken	140
5.19	Messung der Transfer- and Renderzeiten	143
6.1	Client-Server-Szenario des Fernvisualisierungs-Framework	147
6.2	Latenzzeiten	151
6.3	Kompressionsraten	157
7.1	Entscheidungsdiagramm für die vorgestellten Strategien und Algorithmen	160
7.2	Gesamtbetrachtung der vorgestellten Strategien und Algorithmen	161
9.1	Konvektionsströmung in der Erdkruste	171
9.2	Multi-Textur-Volumenvisualisierung eines CTA Aneurysma-Datensatzes	172
9.3	Multi-Textur-Volumenvisualisierung eines Motorblocks	172

9.4	Alpha-Test-basierte Isoflächen	172
9.5	Konvektionsströmungen in der Erdkruste	173
9.6	Vorintegrierte Isoflächenvisualisierung, Transparenz	174
9.7	Vorintegrierte Isoflächenvisualisierung, multiple Isoflächen	174
9.8	Mischung von semi-transparenter Darstellung und Isoflächen	175
9.9	Vorintegriertes Volume-Rendering eines Zahn-Datensatzes	175
9.10	Vorintegriertes Volume-Rendering einer sphärischen, harmonischen Funktion	176
9.11	Vorintegriertes Volume-Rendering mit randomisierter Transferfunktion	176
9.12	Vergleich der Klassifikationsschemata mit randomisierter Transferfunktion	177
9.13	Vergleich der Klassifikationsverfahren bei der Visualisierung kleiner Strukturen des Innenohrs	178
9.14	Vorintegriertes Volume-Rendering eines Feuerballs	179
9.15	Ergebnisse der Rekonstruktion von Dreiecksstreifen aus Volumendaten	180
9.16	Hierarchische Level-of-Detail Rekonstruktion	181
9.17	Client-Applikation zur Fokus-basierten Visualisierung	181
9.18	Client- und Server-Applikation für entfernte Visualisierung	182
9.19	Visualisierungsapplikation aus dem Fahrzeugentwicklungsprozess	182

Tabellenverzeichnis

2.1	Charakteristische Leistungsdaten für die Produktreihe von Silicon Graphics	35
2.2	Charakteristische Leistungsdaten für die Produktreihe von NVIDIA	36
5.1	Datenmengen einer progressiven Isofläche	130
5.2	Modelcharakteristik und Rekonstruktionszeiten	142
6.1	Bildgröße und Bildwiederholraten für Fernvisualisierung.	156

Alles, was sich zu lange hinschleppt,
ehe es zu etwas nur irgend Sichtbarem wird,
verliert an Interesse.

Wilhelm von Humboldt, 1767-1835

Kapitel 1

Einführung

Im Umfeld des Austauschs und der Visualisierung wissenschaftlicher Daten profitieren die Naturwissenschaften in steigendem Maße von der Entwicklung und Verbreitung digitaler Kommunikationsnetze. Vor allem das Internet hat sich in diesem Zusammenhang als entscheidendes Kommunikationsmedium für Forschung und Lehre etabliert. Neben dem Zugriff auf entfernte Datenbestände und dem Austausch von Informationen und Publikationen, eröffnet das Internet ein breites Feld an Möglichkeiten der Einbettung interaktiven, multimedialen Inhalts in Digitale Dokumente. Zwar existieren hier inzwischen zahlreiche Lösungen für statische 3D-Szenen oder Videos, erheblicher Forschungsbedarf besteht aber noch im Bereich der dynamischen, interaktiven Visualisierung wissenschaftlicher Daten. Es genügt hier nicht allein, neue Standards zur Einbettung der Daten in Digitale Dokumente zu schaffen. Vielmehr müssen die Daten mit zusätzlicher „Intelligenz“ in Form effizienter Algorithmen zur Visualisierung versehen werden. Ziel dieser Bemühungen muss es sein, zu jeder Zeit und an jedem Ort auf wissenschaftliche Daten zugreifen zu können und dabei eine interaktive Exploration (Erkundung), Kognition (Erkennung) und Explanatation (Erklärung) von den in wissenschaftlichen Daten vorhandenen Strukturen und Prozessen zu ermöglichen.

In der Vergangenheit wurde die Visualisierung wissenschaftlicher Daten und die Forschung im Bereich der digitalen Kommunikationsnetze meist als zwei disjunkte Disziplinen angesehen. Hier hat sich in den letzten Jahren ein erheblicher Wandel vollzogen. Es existieren bereits eine Reihe von Netzwerk-basierten Visualisierungslösungen, die eine Konvergenz dieser beiden Disziplinen zur Folge haben. Die Visualisierung wissenschaftlicher Daten mit jedem Rastergraphik-fähigen Gerät im Internet, jederzeit und an jedem Ort, ist in greifbare Nähe gerückt. 3D-Web-Technologie-Standards, wie die Virtual Reality Modeling Language (VRML), Java3D oder X3D, sind dabei wichtige Werkzeuge, um die Vielzahl unterschiedlicher Rechnerarchitekturen, Netzwerktopologien und Bandbreiten im Internet in Einklang zu bringen. Über Breitband-Kommunikationsdienste wie Mbone (Multicast Backbone, siehe Abbildung 1.1) sind schon seit einiger Zeit Videokonferenzen in Echtzeit, sowie Streaming interaktiver 3D-Graphik über weite Distanzen möglich. Aber auch im privaten Bereich wachsen Infrastrukturen mit höheren Kapazitäten, wie beispielsweise DSL (Digital Subscriber Line) im Heimbereich oder UMTS (Universal Mobile Telecommunications System) im Bereich mobiler Kommunikationsdienste.

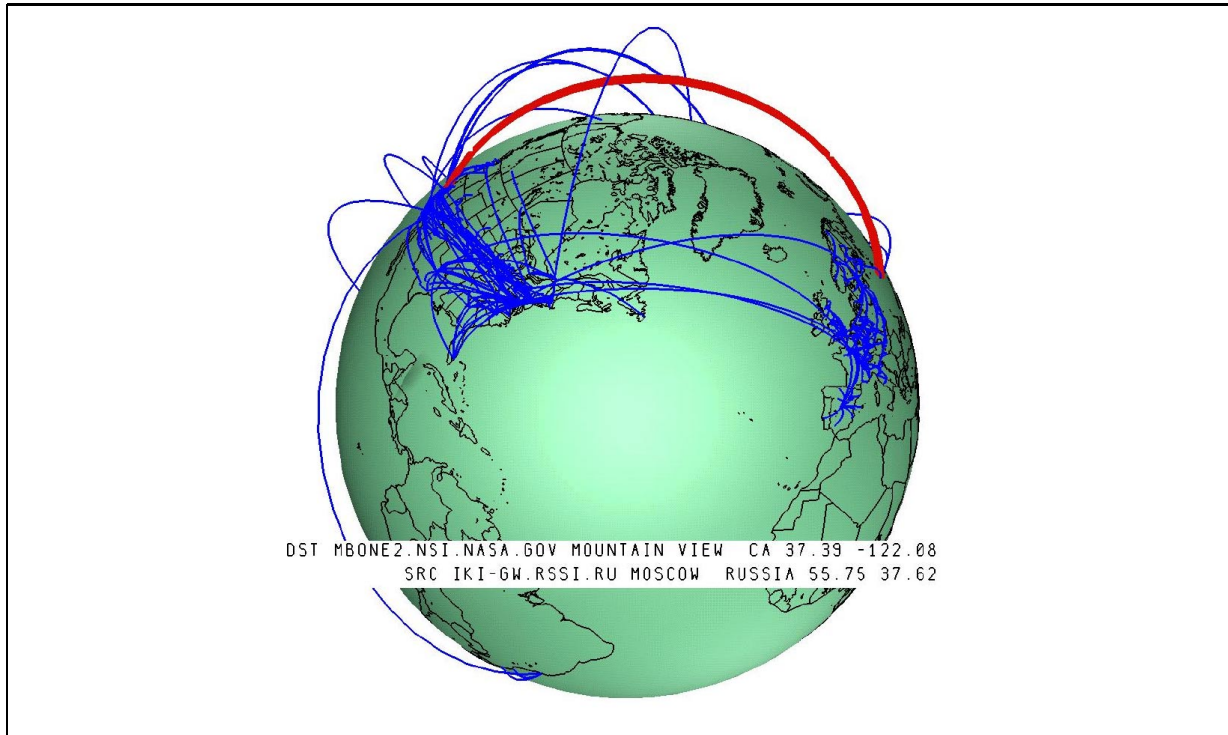


Abbildung 1.1: Visualisierung der Mbone-Topologie durch Abbildung auf einen 3D-Globus, modelliert in der Virtual Reality Modeling Language (VRML)[101].

Zwar existiert im Web-Umfeld bereits eine Vielzahl unterschiedlichster Visualisierungsdienste, allerdings sind diese in den meisten Fällen ausschließlich im Batch-Betrieb nutzbar. Darunter ist zu verstehen, dass die zu visualisierenden Daten zunächst an einen Server gesandt werden, der nach einer gewissen Bearbeitungszeit eine fertige Visualisierung, beispielsweise in Form einer 3D-VRML-Szene, zurückliefert. Eine interaktive Manipulation der Visualisierungsparameter ist somit nicht möglich. Dagegen bietet die verteilte, interaktive Visualisierung wissenschaftlicher Daten ein enormes Potenzial für Forschung und Lehre. Der Trend geht hier eindeutig weg von statischen Bildern hin zu interaktiven drei-dimensionalen Visualisierungen. Solche Darstellungen, eingebettet in Digitale Dokumente, erlauben ein breiteres Verständnis komplexer Zusammenhänge.

Dokumente dieser Art bilden die Grundlage einer *Digitalen Bibliothek*. Laut [31] ist eine *Digitale Bibliothek* eine Einrichtung,

- die Texte, Bilder, Animationen, Ton- und Videoaufnahmen auf elektronischen Datenträgern vorhält (elektronische Bibliothek),
- die eine Vielzahl von Bibliotheksdiensten in einem ortsübergreifenden Verbund anbietet (virtuelle Bibliothek) und

- deren Bestände und Dienste integriert sind, die einen effizienten Zugriff darauf über eine einheitliche Systemoberfläche gestattet, und deren „Systemintelligenz“ über die der Teile hinausgeht.

Für die Einbettung und Visualisierung wissenschaftlicher Daten innerhalb *Digitaler Dokumente* ist aber eine reine Integration von Tabellen, Bildern, Videos oder 3D-Modellen nicht ausreichend. Vielmehr sollte eine interaktive Visualisierung der Daten durch eine direkte Manipulation möglichst vieler Visualisierungsparameter unterstützt werden, um so zu den unterschiedlichsten Darstellungen zu gelangen. Die Erfüllung dieser Voraussetzungen führt zu dem Begriff der *Interaktiven Digitalen Bibliothek*, die neben dem Abruf von Wissen auch die eigenständige Gewinnung neuer Erkenntnisse unterstützt.

Interaktion im Zusammenhang mit der wissenschaftlichen Visualisierung ist dabei die unmittelbare Reaktion (Feedback) einer Darstellung auf die Änderung eines oder mehrerer Visualisierungsparameter. Dieses unmittelbare Feedback ist eine zwingende Voraussetzung für die Erkundung unbekannter Daten. Gegen eine interaktive Darstellung spricht allerdings die enorme Größe, durch die sich typische wissenschaftliche Volumendatensätze heute auszeichnen. Dadurch und wegen der Vielfalt der unterschiedlichen Kommunikationskanäle und Client-Architekturen im Internet kann eine interaktive Darstellung nicht ohne weiteres in jedem Falle gewährleistet werden. Es müssen deshalb für jede Visualisierungsaufgabe Algorithmen und Strategien entwickelt werden, um optimale Voraussetzungen für die jeweils gegebenen Anforderungen und Anwendungen zu schaffen.

1.1 Fragestellung

Ausgehend von den obigen Überlegungen stellt sich nun die Frage, wie eine interaktive Visualisierung großer bzw. komplexer wissenschaftlicher Daten im Internet erreicht werden kann. Unter interaktiver Visualisierung wird dabei im Folgenden eine Darstellung mit einer Bildwiederholrate von mindestens 5 Hertz angesehen (im Idealfall größer 25 Hertz), bei der die Latenz zwischen Benutzeraktion und Aktualisierung der Darstellung weniger als 100 Millisekunden beträgt.

Jedoch stehen dem Ziel der Interaktivität vor allem die enormen Datenmengen entgegen, die heute bei typischen Messungen und Simulationen gewonnen werden. Die Heterogenität der im Internet verfügbaren Netzverbindungen und Client-Systeme stellt dabei eine zusätzliche Herausforderung dar. Limitierte Bandbreiten und Rechnerkapazitäten, hohe Latenzzeiten sowie die limitierte Verfügbarkeit von Server-Ressourcen stehen im krassen Gegensatz zu den geforderten hohen Interaktionsraten. Auch können Daten im Internet nicht beliebig auf Client- und Server-Rechner verteilt werden, da sich oftmals ein Transfer der Daten aus Sicherheits-, Geheimhaltungs- oder Datenschutzgründen verbietet.

Daraus ergibt sich eine der Kernfragen, mit der sich diese Dissertation beschäftigt: Wie kann die Vielzahl der möglichen Client-Server-Konfigurationen im Internet optimal unterstützt werden? Dazu ist eine Aufspaltung der klassischen Visualisierungspipeline in Client- und Server-Module notwendig, zwischen denen ein zusätzliches Datentransfer-Modul eingefügt wird. Die Optimierung der so entstandenen Pipeline im Hinblick auf größtmögliche Interaktionsraten ist

eines der wichtigsten Ziele dieser Arbeit. Dabei stellt sich auch die Frage nach der Eignung adaptiver Verfahren zur Optimierung der Pipeline hinsichtlich Datenaufkommen und Antwortzeit.

Ein zusätzliches Ziel ist es, dem Benutzer bei der Exploration der Daten größtmögliche Freiheit zu lassen, also die Manipulation der Visualisierungsparameter aller Stufen der Visualisierungspipeline zuzulassen. Doch müssen die unterschiedlichen Anforderungen und Fähigkeiten der Vielzahl an Nutzern eines Visualisierungsdienstes berücksichtigt werden. Es stellt sich somit die Frage, wie man bestimmten Nutzern größtmögliche Freiheit bei der Manipulation der Parameter lässt, ohne dabei andere durch zu hohe Komplexität abzuschrecken.

Eine wesentliche Rolle in der interaktiven Visualisierung spielt die Nutzung leistungsfähiger 3D-Graphikhardware. Das Internet bietet hier prinzipiell Zugriff auf eine große Menge entfernter Rechenkapazitäten. Im Batch-Computing-Bereich gibt es hier seit Jahren Lösungen, die entfernte numerische Ressourcen nutzbar machen. Dagegen waren im Bereich interaktiver Nutzung entfernter 3D-Graphikhardware bisher kaum Lösungen vorhanden. Es werden zwar häufig schon entfernt erzeugte Darstellungen zu einem Arbeitsplatzrechner übertragen, eine interaktive Rückkopplung ist aber kaum vorhanden. Aus diesem Grunde war die interaktive Nutzung entfernter 3D-Graphikhardware eines der wesentlichen Ziele dieser Arbeit.

Im Gegensatz dazu stehen die gestiegenen Fähigkeiten heutiger Arbeitsplatzrechner im 3D-Bereich. Sowohl bei der numerischen Leistung als auch bei den graphischen Darstellungsfähigkeiten wurden in den letzten Jahren enorme Fortschritte erzielt. Somit stellt sich die Frage, ob die Graphikhardware heutiger Client-Rechner zur Visualisierung herangezogen werden kann. Diese Graphikhardware zeichnet sich durch eine Reihe von Erweiterungen aus, die hauptsächlich durch die Nachfrage nach neuen Effekten und realistischeren Szenen in Computerspielen vorangetrieben werden. Es war zu Beginn der Forschungen zu dieser Arbeit nicht klar, ob diese Erweiterungen auch für die wissenschaftliche Visualisierung herangezogen werden können. Die Nutzung herstellerspezifischer Erweiterungen wirft eine weitere Frage auf: Wie kann bei Nutzung neuester Funktionalität moderner Graphikhardware die Portabilität der Algorithmen gewährleistet werden und können dabei Internet-Standards zur Implementierung herangezogen werden ?

1.2 Gliederung der Arbeit

Anschließend an dieses Kapitel gliedert sich die vorliegende Arbeit folgendermaßen.

Kapitel 2 gibt eine Einführung in die grundlegenden Begriffe der Volumenvisualisierung, der interaktiven Computergraphik und digitaler Netzwerke. Daran schließt sich ein Überblick zur Historie der Visualisierung im Internet an.

Kapitel 3 führt den neuen Begriff der Client-Server-Strategien ein, der im weiteren Verlauf der Arbeit zur Abgrenzung und Klassifikation der verschiedenen Visualisierungsszenarien eingesetzt wird. Im wesentlichen werden hier drei verschiedene Methoden der Aufgabenverteilung zwischen Client und Server näher beleuchtet.

Kapitel 4 beantwortet die Frage, ob und wie flexible Rasterisierungshardware moderner Arbeitsplatzrechner zur Volumenvisualisierung herangezogen werden kann. Das Kapitel spiegelt den großen Wandel bezüglich der Leistungsfähigkeit dieser 3D-Graphikhardware wider, der sich während der Forschungsarbeiten zu dieser Arbeit vollzog. Zunächst werden architekturneutrale

Algorithmen zur texturbasierten Volumenvisualisierung betrachtet. Anschließend wird gezeigt, wie durch den Einsatz von Multi-Texturen und programmierbaren Fragment-Operationen moderner PC-Graphikhardware erhebliche Verbesserungen hinsichtlich Qualität und Geschwindigkeit der Ergebnisse erzielt werden können. Ein komplett neuer, auf aktuellen Entwicklungen in der Graphikhardware basierender Algorithmus zur vorintegrierten Volumenvisualisierung zeigt abschließend weitere, erhebliche Verbesserungen.

Kapitel 5 demonstriert, wie ein auf der Extraktion geometrischer Primitive basierender Algorithmus zur Visualisierung von Isoflächen aus Volumendaten in einem Client-Server-Umfeld verteilt und optimiert werden kann. Dazu wird zunächst ein Algorithmus zur lokalen Rekonstruktion und Darstellung betrachtet, der unter anderem in einem Web-basierten Orbitalvisualisierungsservice Einsatz findet. Anschließend wird ein Fehler-gesteuerter Multi-Level-Ansatz zur progressiven Übertragung und Darstellung komplexer Isoflächen vorgestellt. Im Gegensatz zu diesem global progressiven Ansatz steht ein Verfahren, welches mit Hilfe eines Octrees eine Fokus-gesteuerte Verfeinerung interessanter Bereiche der Volumendaten erlaubt. Es werden hier sowohl Fragen nach der effizienten Übertragung, als auch nach der Balancierung der Teilaufgaben zwischen Client und Server beantwortet.

Kapitel 6 befasst sich schließlich mit einem Rahmenwerk, das die Nutzung entfernter hochwertiger 3D-Graphikhardware mit einem einfachen Rastergraphik-fähigen Client über weite Entfernungen hinweg erlaubt. Die Tauglichkeit des Ansatzes wird an konkreten Anwendungen aus der Medizin und den Ingenieurwissenschaften demonstriert. In Kombination mit den in Kapitel 4 vorgestellten Algorithmen können die Vorteile der entfernten Visualisierung mit der Nutzung lokaler Graphikhardware in einem hybriden Verfahren vereint werden.

In **Kapitel 7** werden die in dieser Dissertation behandelten Algorithmen und Strategien in einem Gesamtbild betrachtet, bevor schließlich in **Kapitel 8** die vorliegende Arbeit zusammengefasst und ein Ausblick auf zukünftige Entwicklungen gegeben wird.

1.3 Beiträge der Arbeit

Diese Arbeit zeigt Strategien und Algorithmen auf, welche die interaktive Visualisierung entfernter wissenschaftlicher Daten an jedem Ort und auf jedem Rastergraphik-fähigen Client, angefangen von einem PDA (Personal Digital Assistant) bis hin zu einem Arbeitsplatzrechner, erlauben. Dabei steht primär die Visualisierung drei-dimensionaler Skalarfelder im Vordergrund, auch wenn einige der vorgestellten Konzepte unabhängig vom konkreten Daten- und Gittertyp sind.

Als Server wird im Folgenden ein Rechner vorausgesetzt, der Daten aus einer beliebigen Stufe der Visualisierungspipeline über eine Netzwerkverbindung zur Verfügung stellt. Dies fängt an bei Datenbankservern und geht über High-End-Graphik-Server bis hin zu Supercomputern- oder Cluster-Servern. Dabei stehen weniger Server-seitige Parallelisierungsverfahren als vielmehr die Aufspaltung der Visualisierungspipeline zwischen Client und Server unter optimaler Nutzung der verfügbaren Rechen- und Graphikressourcen im Vordergrund. Ebenfalls keine Rolle spielen Batch-Rendering-Lösungen, wie sie beim *Virtual-Supercomputing*-Konzept zum Einsatz kommen. Diese liefern zwar die Möglichkeit, riesige Datenmengen verteilt zu bearbeiten, zur

interaktiven Darstellung wissenschaftlicher Daten sind sie aber gänzlich ungeeignet.

Da zur interaktiven Visualisierung hochaufgelöster Daten in den meisten Fällen zwingend 3D-Graphikhardware genutzt werden muss, stellte sich zu Beginn dieser Arbeit die Frage, ob und in wie weit sich aktuelle PC-Graphikhardware, die in den meisten Client-Rechnern heutzutage vorhanden ist, nutzen lässt. Während der Forschungsarbeiten zu dieser Arbeit vollzog sich ein enormer Wandel in den Fähigkeiten dieser Graphikhardware. Dies spiegelt sich auch in dieser Arbeit wider. Wurden anfangs nur die Rasterisierungseinheiten früher PC-Graphikhardware zur Darstellung von Dreiecken genutzt (siehe Kapitel 5), so wurden später neueste Erweiterungen dieser Karten zur bis *dato* auf Spezialgraphikhardware beschränkten direkten Volumenvisualisierung eingesetzt (siehe Kapitel 4). Die Arbeit zeigt nicht nur, dass solche Hardware zur Visualisierung eingesetzt werden kann. Vielmehr wird demonstriert, dass die durch den großen Konkurrenzdruck in der Unterhaltungsbranche entstandenen Erweiterungen eine Reihe von Algorithmen möglich werden, die sogar denen in Spezialhardware implementierbaren Algorithmen überlegen sind. Dies trifft sowohl auf die Darstellungsqualität als auch auf die Darstellungsgeschwindigkeit der resultierenden Anwendungen zu.

Neben diesen erweiterten 3D-Darstellungskapazitäten neuer Arbeitsplatzrechner war auch die Erkundung der Möglichkeiten einer mobilen Verfügbarkeit eingebetteter wissenschaftlicher Visualisierungen in Digitalen Dokumenten ein bedeutendes Ziel. Mittels Funknetzwerken oder Mobilfunk können über Notebooks, PDAs oder Mobiltelefone jederzeit solche Digitalen Dokumente eingesehen werden. Standort-abhängige Dienste (Location-aware Services) profitieren dabei von der Einbettung komplexer Visualisierungen außerordentlich, so dass deren Wert erheblich gesteigert werden kann. Leider verfügen mobile Geräte im Allgemeinen über nur eingeschränkte Darstellungskapazitäten, weswegen eine lokale Bildgenerierung nicht in Frage kommt. In dieser Arbeit wird gezeigt, wie eine interaktive Darstellung hochaufgelöster wissenschaftlicher Daten auf jedem Rastergraphik-fähigen Gerät durch Zugriff auf entfernte Graphikhardware erreicht werden kann (siehe Kapitel 6).

Desweiteren wird ein neues Schema zur optimalen Verteilung der Teilaufgaben einer Visualisierung auf Client und Server eingeführt, die so genannten *Client-Server-Strategien* (siehe Kapitel 3). Ziel dieser Strategien ist es, je nach vorhandenen Ressourcen die Visualisierungspipeline aufzuspalten und die sich daraus ergebenden Teilaufgaben auf Client und Server so zu verteilen, dass optimale Interaktionsraten durch Minimierung des Datenaufkommens und der Latenzen erreicht werden können. Mittels einer dynamischen Wahl der Client-Server-Strategie kann auch auf variable Verfügbarkeit von Ressourcen zu Beginn oder sogar während einer Visualisierungssitzung eingegangen werden. Client-Server-Strategien werden im weiteren Verlauf der Arbeit dann auf konkrete Probleme angewandt. Für die Visualisierung von Isoflächen durch Dreiecksextraktion werden in Kapitel 5 beispielsweise sechs verschiedene Szenarien ausgearbeitet.

Trotz optimaler Nutzung von Client- und Server-Hardware sind häufig keine befriedigenden Bildwiederholraten erreichbar, da die Menge der zu verarbeitenden Daten typische Verarbeitungskapazitäten heutiger Rechner bei weitem übertrifft. Beispielsweise ist die große Anzahl geometrischer Primitive, die mit einem Standard-Visualisierungsverfahren wie dem Marching-Cubes-Algorithmus[91] für mittlere und hochaufgelöste Volumendatensätze generiert werden, weder mit einem Arbeitsplatzrechner, noch mit einer Graphik-Workstation in interaktiven Raten

darstellbar. Im Rahmen dieser Arbeit werden deshalb auch Verfahren behandelt, die in der Lage sind, auf mehreren Auflösungsstufen der zu visualisierenden Daten zu arbeiten. Diese Verfahren erlauben einerseits eine globale progressive Darstellung von Isoflächen (Kapitel 5.2), als auch eine lokale Verfeinerung einer Isofläche mit einem Fokuspunkt (Kapitel 5.3). In diesem Zusammenhang wird auch die effiziente Übertragung der Daten behandelt.

Ferner spielt die Portierbarkeit und die Orientierung der im Rahmen dieser Arbeit entwickelten Systeme an Internet-Standards eine wichtige Rolle. Viele der vorgestellten Lösungen wurden unter Zuhilfenahme wirklicher oder *de-facto* Internet-Standards entwickelt. Die Nutzung spezieller, häufig proprietärer Graphikhardware-Erweiterungen, steht hier aber auf den ersten Blick im genauen Gegensatz zu den obigen Forderungen. Diese Arbeit zeigt auf, wie durch eine dynamische Wahl der Client-Server-Strategie und durch Anbindung plattformunabhängiger Programmiersprachen an *low-level* Graphikhardware Forderungen dieser Art trotzdem erfüllt werden können.

Diese Arbeit wurde von der Deutsche Forschungsgemeinschaft im Rahmen des Schwerpunktprogramms V3D2 (Verteilte Vermittlung und Verarbeitung Digitaler Dokumente) unterstützt. Das Teilprojekt **Chemische Visualisierung im Internet** beschäftigt sich mit der Entwicklung eines portablen, sich an Internet-Standards orientierenden Systems zur Einbettung von dynamischen Visualisierungen aus dem Bereich der Chemie in digitale Dokumente. Das Ziel ist es, anhand konkret implementierter Beispiele den Nachweis zu führen, dass mit einem solchen System gegenüber statischen 3D-Szenen oder Videos das Verständnis komplexer Phänomene, die aktuelle Forschungsthemen der Chemie sind, wesentlich erleichtert oder sogar erst ermöglicht werden. Wichtige Effekte sind zum Beispiel von der räumlichen Geometrie, der Zeitachse und Feldeinflüssen abhängig und somit kaum durch einzelne statische Punktansichten zu vermitteln. Die Kopplung und Echtzeitmanipulation von 3D-Visualisierungen, gesteuert von durch den Benutzer geänderten Darstellungsattributen, Raum/Zeitparameter oder andere Einflussfaktoren, erlaubt es, eine neue Qualität und Dichte der Information zu realisieren, die mit der Komplexität der Forschungsgegenstände Schritt hält. Verschiedene Modelle der Szenenmanipulation sollen einen breiten Rahmen der Einsatzmöglichkeiten und -voraussetzungen solcher dynamischer Szenarios ausleuchten. Einige der in dieser Arbeit vorgestellten Visualisierungslösungen kommen daher aus dem Umfeld der Chemie.

1.4 Betreute Studien- und Diplomarbeiten

Im Rahmen dieser Arbeit wurden eine Reihe von Studien- und Diplomarbeiten betreut, die nachfolgend zusammengestellt sind:

Heike Pohl: *Hardwareunterstützte Visualisierung medizinischer Daten mittels plattformunabhängiger Graphikschnittstellen für den Einsatz im World Wide Web*, 1999

Christian Ernst: *Medizinische Visualisierung im WWW mittels 3D-Texturen*, 1999

Jürgen Beckmann: *3D Interaktionselemente als Benutzerschnittstelle in virtuellen Umgebungen*, 1999

Michael Bauer: *Optimierung der Volumenvisualisierung auf PC Hardware*, 2000

Guido Reina: *Visualisierung und Manipulation großer Graphen in einem Graphical User Interface am Beispiel der zSeries I/O-Topologie*, 2001

Alles Gescheite ist schon gedacht worden.
Man muß nur versuchen, es noch einmal zu denken.

Johann Wolfgang von Goethe, 1749-1832 (Maximen und Reflexionen)

Kapitel 2

Grundlagen

In diesem Kapitel sollen die wesentlichen Grundlagen und Vorarbeiten erläutert werden, die zum Verständnis der nachfolgenden Kapitel notwendig sind. Zunächst steht der Begriff der wissenschaftlichen Visualisierung mit seiner typischen Interpretation als rückgekoppelte lineare Pipeline im Vordergrund. In diesem Zusammenhang werden auch charakteristische Gitter- und Datentypen aus unterschiedlichen Problemfeldern kurz angesprochen.

Es schließt sich eine Einführung in den Bereich der interaktiven Computergraphik und deren Realisierung durch eine effektive Nutzung neuester Graphikhardwarearchitekturen an. Eine interaktive Visualisierung ist zum Verständnis drei-dimensionaler Daten eine wesentliche Grundvoraussetzung.

Ein weiterer Abschnitt behandelt Volumendaten, den für diese Arbeit wesentlichen Datentyp. Er führt in die grundlegenden physikalischen und algorithmischen Fundamente der Visualisierung solcher Daten ein, wobei auch auf Vorarbeiten zur interaktiven Volumenvisualisierung unter Nutzung von Graphikhardware eingegangen wird.

Nach einer kurzen Vorstellung der wichtigsten relevanten Techniken, die im Zusammenhang mit der Visualisierung in digitalen Dokumenten im Internet eine Rolle spielen, werden abschließend noch entscheidende Ansätze und Vorarbeiten im Bereich der Visualisierung im Internet besprochen.

2.1 Visualisierung

Im Mittelpunkt der wissenschaftlichen Visualisierung steht die Gewinnung von Erkenntnissen aus abstrakten Daten durch die Transformation von Unsichtbarem in Sichtbares. Das Sehen eignet sich zur Gewinnung von Erkenntnissen gerade deshalb so gut, weil unter allen Wahrnehmungssystemen des Menschen der Sehapparat, in modernen Begriffen gehalten, die größte „Bandbreite“ bei der Aufnahme von Informationen aller Sinnesorgane des Menschen besitzt.

Dabei ist bekanntermaßen die Visualisierung nicht erst eine Erfindung des Computerzeitalters. Beispielsweise werden die verheerenden Verluste an Menschenleben bei Napoleons Russlandfeldzug in den Jahren 1812-1813 in Joseph Minards „Carte figurative des pertes successives

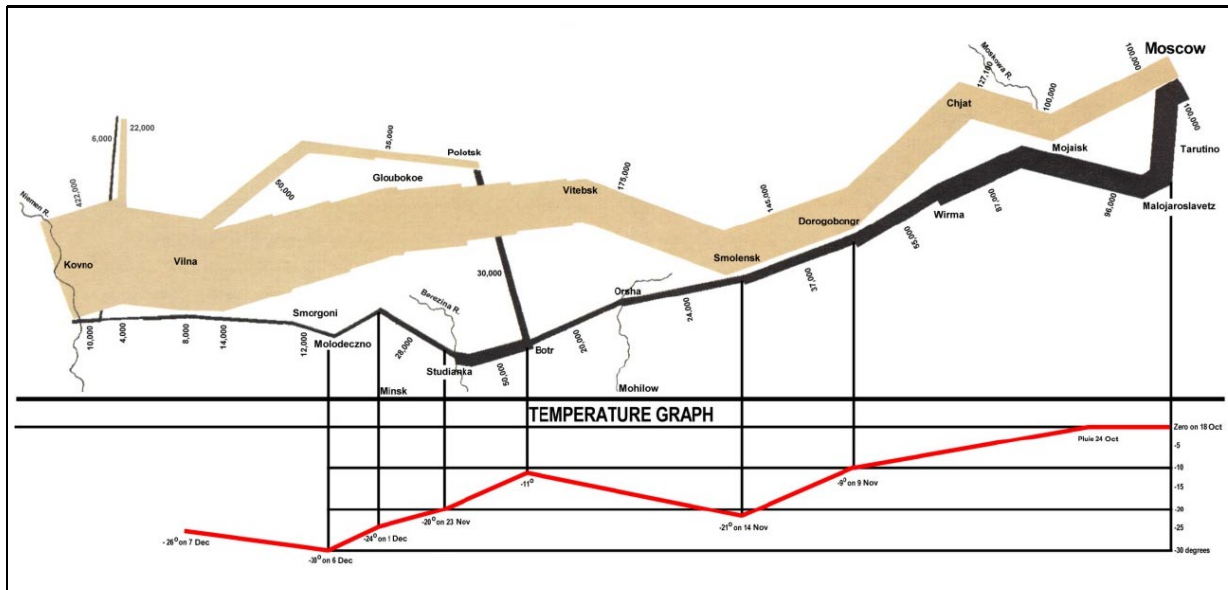


Abbildung 2.1: Reproduktion von Minard's Karte des Russlandfeldzuges von Napoleon in den Jahren 1812-1813.

en hommes de l' Armee Française dans la campagne de Russie 1812-1813“ deutlich (siehe Abbildung 2.1), die unumstritten als eine der besten Visualisierungen angesehen wird, die je geschaffen wurden. Diese exzellente Visualisierung vereinigt eine Vielzahl unterschiedlicher Parameter in einer einzigen Darstellung: die Größe der Armee (visualisiert durch die Breite des Graphen), der Aufenthaltsort der Truppen zu einem bestimmten Zeitpunkt, die Bewegungsrichtung der Armee und die Temperatur auf dem Rückzug der Armee nach der Belagerung von Moskau. Dieses Beispiel beweist eindrucksvoll, dass eine bildliche Darstellung in besonderer Weise zur effizienten Vermittlung und schnellen Auffassung einer derartigen Informationsfülle geeignet ist.

Heute werden Visualisierungen üblicherweise mit Rechnerunterstützung erzeugt. Dadurch ist ein äußerst wichtiger Freiheitsgrad hinzu gekommen: die Interaktion. Durch Manipulation von Visualisierungsparametern und einer darauf folgenden unmittelbaren Anpassung der Darstellung mit den aktualisierten Parametern sind weitere Einsichten in die Daten möglich. Technisch wird die Visualisierung heute als die Transformation von Ausgangsdaten aus Messungen, Datenbanken oder Simulationen mittels verschiedener linear angeordneter Verarbeitungsstufen in ein möglichst aussagekräftiges graphisches Abbild verstanden. Eine realistische Darstellung spielt dabei meist eine untergeordnete Rolle. Vielmehr steht eine korrekte und leicht verständliche Repräsentation im Vordergrund, um so eine rasche Analyse der Daten zu ermöglichen.

2.1.1 Visualisierungspipeline

Der Prozess der Visualisierung wissenschaftlicher Daten wird im Allgemeinen als mehrstufige Pipeline (Bild 2.2) dargestellt, deren erste Stufe die Ausgangsdaten ausdünn (Filter), um

beispielsweise für die Visualisierung irrelevante Datenpunkte oder Parameter zu entfernen oder Daten zu konvertieren. Die auf diese Weise gefilterten Daten werden dann durch ein Abbildungsmodul (Mapper) in eine darstellbare Repräsentation gebracht. Dies sind im Allgemeinen geometrische Objekte oder Primitive, denen zusätzliche Attribute wie Position, Farbe, Textur oder Transparenz zugeordnet werden. Anschließend wird die so erzeugte geometrische Repräsentation der Daten von einem Darstellungsmodul (Renderer) durch Projektion auf eine Bildebene in ein Rasterbild abgebildet.

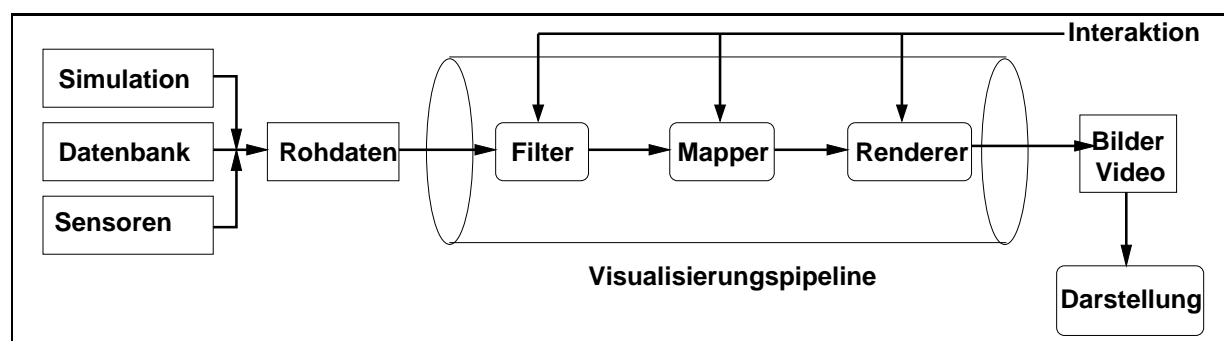


Abbildung 2.2: Die Visualisierungspipeline.

Ein wesentlicher Aspekt ist dabei die Interaktion des Benutzers mit dem System durch eine Rückkopplung mit allen Stufen der Pipeline. Häufig ist, vor allem bei unbekanntem Daten, erst durch die interaktive Manipulation der Visualisierungsparameter aller Stufen ein Erkenntnis über die in den Daten vorhandenen Informationen möglich. Voraussetzung für hohe Interaktionsraten sind effiziente Algorithmen und Datenstrukturen sowie entsprechende Verarbeitungskapazitäten der verwendeten Hardware. Es ist daher nicht verwunderlich, dass frühe Visualisierungssysteme keine oder nur eingeschränkte Interaktionsmöglichkeiten boten.

2.1.2 Gitter und Daten

Je nach Art der Messung oder Simulation liegen die gewonnenen Daten mit unterschiedlichem Typ und auf verschiedenen Gittern vor. Bezüglich der Datentypen wird im Wesentlichen zwischen Skalar-, Vektor- und Tensorfeldern sowie multivariaten Daten unterschieden. Die vorliegende Arbeit befasst sich ausschließlich mit Skalarfeldern.

Bei den Gittern werden kartesische, reguläre, rektilineare, strukturierte oder kurvilineare, blockstrukturierte, unstrukturierte und hybride Gitter unterschieden. In dieser Arbeit werden überwiegend kartesische Gitter behandelt. Diese Art der Daten sind für eine Reihe von Messverfahren und Simulationen charakteristisch und eignen sich in besonderer Weise für eine Darstellung mittels schneller Graphikhardware.

2.2 Interaktive Computergraphik

Eine zwingende Grundvoraussetzung zur Exploration unbekannter wissenschaftlicher Datensätze ist, wie bereits betont wurde, die Interaktivität der Visualisierung, also die unmittelbare Reaktion der Darstellung auf Änderungen von Parametern der Visualisierungspipeline. Das enorme Wachstum der Datenmengen durch neue oder verbesserte Simulations- und Messverfahren erfordert die Optimierung aller Stufen der Visualisierungspipeline. Neben der Entwicklung neuer und optimierter Filter- und Mapping-Verfahren spielt in diesem Zusammenhang die Nutzung leistungsfähiger Graphikhardware eine immer größere Rolle. Das rapide Wachstum der Leistungsfähigkeit und die zunehmende Flexibilisierung der Graphikhardware erlaubt sowohl das effiziente Rendering der Voxel bzw. Polygone der Mapping-Stufe als auch die Verlagerung aufwändiger Mapping-Algorithmen in den Aufgabenbereich der Graphikhardware (siehe Kapitel 4).

2.2.1 Graphikhardware

In dem nun folgenden kurzen Abriss über die Historie der Graphikhardware wird allein auf Objektraumverfahren-basierte Hardwarearchitekturen eingegangen, also auf Architekturen die GL-ähnliche Eigenschaften[102] (siehe Abschnitt 2.2.2) besitzen. Die erste Generation dieser Hardware (vor 1987) unterstützte ausschließlich Drahtgitter-Darstellungen, die durch Transformation, Projektion und Clipping der Start- und Endpunkte von Linien erzeugt wurden. Diese Hardware war nur in der Lage, Linien und Punkte zu rasterisieren, wobei im Bildspeicher (*Framebuffer*) bereits gesetzte Pixel überschrieben wurden. Ausgefüllte Polygone und eine Beleuchtungsberechnung pro Eckpunkt (*Vertex*) wurde erst in der zweiten Generation (1987-1992) der Rasterisierungshardware eingeführt. Deren Framebuffer verfügte bereits über Tiefenpufferung (*Depth Buffer*) und Überblendfunktionalität (*Blending*). Die Möglichkeiten, Polygone mit Texturen zu versehen (*Texture Mapping*) und eine Bildkantenglättung vorzunehmen (*Anti-Aliasing*) wurden in der dritten Generation (1992-2000) eingeführt. Schließlich ist die heute verfügbare Generation der Graphikhardware durch eine flexible Programmierbarkeit auf Vertex und Fragment-Ebene charakterisiert.

Für die Überlegenheit spezialisierter Graphikhardwarechips (GPUs) gegenüber CPUs gibt es unterschiedliche Gründe. Diese resultieren primär aus der für diese Architektur typische, logische Separation der Rendering-Teilaufgaben in Module einer Pipeline (siehe Abbildung 2.3). Diese Aufteilung bietet Platz für vielerlei Optimierungen: Frühere Stufen der Pipeline können bereits mit neuen Daten arbeiten, während nachfolgende Stufen noch ältere Daten verarbeiten. Dadurch können einzelne Module effizient parallelisiert werden, wobei jede dieser Stufen nur über lokales Wissen der Szene verfügt. Zusätzlich gibt es in dieser Architektur feste und damit effiziente Datenpfade, in denen Schleifen vermieden werden, um Wartezustände zu vermeiden. Da die Datenmenge beim Übergang von Vertices zu Fragmenten in typischen Szenen sprunghaft ansteigt, wird auf Vertex-Ebene, also in den frühen Stufen der Pipeline, noch mit Fließkomma-Arithmetik gearbeitet, während in späteren Stufen, also auf auf Fragment-Ebene, Festpunkt-Arithmetik zum Einsatz kommt. Wegen der Verwendung von Festpunkt-Arithmetik auf Fragment-Ebene sind inzwischen Füllraten von mehr als einer Milliarde Pixel pro Sekunde

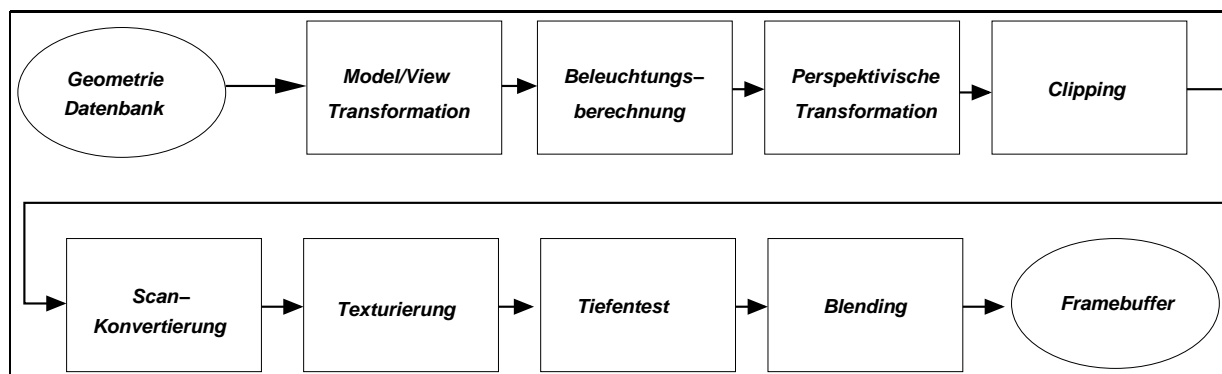


Abbildung 2.3: Rendering-Pipeline.

Jahr	Produkt	Füllrate	Dreiecksrate
1984	Iris 2000	46 M	10 K
1988	GTX	80 M	135 K
1992	RealityEngine	380 M	2 M
1996	InfiniteReality	1000 M	12 M

Tabelle 2.1: Charakteristische Leistungsdaten für die Produktreihe von Silicon Graphics.

möglich geworden. Daneben sind durch effiziente Caching-Strategien und optimierte Speicherzugriffe weitere wesentliche Geschwindigkeitsoptimierungen möglich.

Ein wichtiger Nachteil der Anordnung in einer festen Pipeline mit lokalem Wissen sind eine mangelnde Flexibilität und die schwere Realisierbarkeit von globalen Beleuchtungseffekten. In aktueller Graphikhardware wird versucht, die erste dieser Einschränkungen durch flexible Programmierbarkeit einzelner Module der Pipeline zu beheben. Aktuell können sowohl die Geometrieinheit (*Geometry oder auch Vertex Engine*) als auch Teile der Rasterisierungseinheit (*Raster Manager*, siehe Abschnitt 2.2.4) flexibel programmiert werden. In dieser Arbeit wird primär die Programmierbarkeit aktueller Graphikhardware auf Fragmentebene eine Rolle spielen, da sich die Menge der Vertices bei direkten Volumenvisualisierungsverfahren im Gegensatz zu der Zahl der zu bearbeitenden Fragmente in Grenzen hält. Die Verwendung von Vertexoperationen für indirekte Volumenvisualisierungsverfahren ist zwar denkbar, wegen fehlender Konnektivitätsinformationen und Vertex-Erzeugungsoperationen in dieser Einheit, sind allerdings im Augenblick solche Einheiten nur eingeschränkt zur Implementierung von Visualisierungsalgorithmen nutzbar.

Bei der Verfügbarkeit schneller Graphikhardware hat sich in den letzten Jahren ein enormer Wandel vollzogen. Bis vor wenigen Jahren war effiziente 3D-Graphik ausschließlich auf teure Spezialhardware[3, 100] beschränkt. Die Entwicklung dieser Hardware ist in Tabelle 2.1 für die Produktreihe von Silicon Graphics (SGI) angegeben. Im Bereich der PC-Graphikhardware sind aber deutlich höhere Leistungszuwächse zu beobachten (siehe Tabelle 2.2). Durch den wesentlich breiteren Absatzmarkt und den hohen Konkurrenzdruck haben aktuelle PC-basierte Graphik-

Saison	Produkt	Prozess	#Transistoren	Bandbreite	Füllrate (AA)	Dreiecksrate
2H97	Riva 128	.35 μ	3M	1.2 GB/s	20M	3M
1H98	Riva ZX	.25 μ	5M	1.2 GB/s	31M	3M
2H98	Riva TNT	.25 μ	7M	1.2 GB/s	50M	6M
1H99	TNT2	.22 μ	9M	2.4 GB/s	75M	9M
2H99	GeForce	.22 μ	23M	5.0 GB/s	120M	15M
1H00	GeForce2	.18 μ	25M	5.3 GB/s	200M	25M
2H00	NV16	.18 μ	25M	6.4 GB/s	250M	31M
1H01	NV20	.15 μ	55M	8.8 GB/s	500M	30M
1H02	NV25	.15 μ	63M	10.4 GB/s	600M	75M

Tabelle 2.2: Charakteristische Leistungsdaten für die Produktreihe von NVIDIA. In chronologischer Reihenfolge sind für jedes Produkt, der Herstellungsprozess, die Anzahl der Transistoren, die lokale Speicherbandbreite, die Füllrate (mit Anti-Aliasing) und die maximale Anzahl der pro Sekunde verarbeitbaren Dreiecke angegeben.

lösungen professionelle Graphikhardware teilweise in der Leistungsfähigkeit bereits überholt. Allenfalls bei der Speicherkapazität, den Busbandbreiten und der Qualität der erzeugten Rasterbilder können sich professionelle Lösungen noch entscheidend absetzen. Wesentlicher Grund hierfür sind bessere Samplingverfahren und Puffertiefen dieser Hardware.

Entscheidende Faktoren für die Geschwindigkeit einer Graphikhardwarelösung sind Geometrieverarbeitungsraten, Speicherbandbreiten und Füllraten. Die Geometrieverarbeitungsrate gibt die Leistungsfähigkeit der Geometrieinheit an und wird durch die Anzahl der pro Sekunde transformierten und beleuchteten Vertices oder Dreiecke bestimmt. Der Begriff der Füllrate gibt die Zahl der pro Sekunde verarbeiteten Fragmente an. Beide Faktoren werden wesentlich von der verfügbaren Speicherbandbreite beeinflusst. Dabei spielt sowohl die Bandbreite des Hauptspeicherbusses zum lokalen Speicher der Graphikhardware über den AGP-Bus, als auch die Speicherbandbreite des lokalen Speichers zum Graphikchip eine wichtige Rolle.

Durch die sehr kurzen Produktzyklen bei aktueller Graphikhardware ist in den nächsten Jahren mit weiteren Fortschritten zu rechnen. Diese sind sowohl in den Bereichen Geschwindigkeit, Funktionalität als auch Qualität des Renderings zu erwarten. Durch verbesserte Speichertechnologie, weitere Parallelisierung und Reduktion der Datenmengen, beispielsweise durch Culling, ZBuffer- und Textur-Kompression oder prozedurale Erzeugung von Geometrie und Texturen, sind noch wesentliche Geschwindigkeitssteigerungen möglich. Erweiterungen im Bereich der Funktionalität sind durch weitere Flexibilisierung der Programmierung verschiedener Einheiten der Rendering-Pipeline zu erreichen. Auch hinsichtlich der Qualität des Renderings sind durch eine gesteigerte numerische Genauigkeit sowie bessere Filter- und Multi-Sampling-Verfahren weitere Fortschritte zu erwarten.

2.2.2 3D-Graphikschnittstellen

Zur Programmierung interaktiver, durch 3D-Graphikhardware unterstützter Computergraphik haben sich im Augenblick zwei *immediate mode* Graphikschnittstellen etabliert. Seit OpenGL[153, 12] von *Silican Graphics (SGI)* 1992 als offener Standard eingeführt wurde, hat es sich als branchenweit meist verbreitetes und unterstütztes Application Programming Interface (API) für 2D- und 3D-Graphik etabliert. OpenGL ist heute auf jeder bedeutenden Computer-Plattform verfügbar - u.a. für die Betriebssysteme *IRIX*, *Solaris*, *HP/UX*, *Digital Unix*, *AIX*, *BeOS*, *Microsoft Windows*, *Linux* sowie *MacOS*. Die Weiterentwicklung von OpenGL wird von dem *Architecture Review Board (ARB)* gesteuert, einem Gremium, das sich aus führenden Vertretern der Hard- und Software-Industrie zusammensetzt. OpenGL ist ein anbieterneutraler, plattformübergreifender Graphikstandard, der branchenweite Unterstützung genießt. Dabei wird OpenGL ständig weiterentwickelt, um die kontinuierlichen evolutionären Verbesserungen der Graphikhardware auszuschöpfen. Jeder Hersteller kann eigene Erweiterungen der OpenGL Spezifikation einbringen, die zu einem späteren Zeitpunkt in den OpenGL Standard aufgenommen werden können.

Dagegen ist *Direct3D*[63] eine auf die Windows-Plattform beschränkte API zur Programmierung interaktiver, 3D-Computergraphik, welches in die *DirectX* API eingebettet ist. Die Weiterentwicklung von *Direct3D* wird vor allem durch Microsoft und einigen Hardware-Herstellern vorangetrieben.

Daneben existieren eine Reihe von *high-level* Programmierschnittstellen, die eine hierarchische Szene in Form eines gerichteten, azyklischen Szenengraphen (DAG) verwalten. Dazu zählen beispielsweise *Java3D*[135], *OpenInventor*[53, 143, 142], *Cosmo3D/Optimizer*[71, 72], *OpenSG*[64] und *Open Scenegraph*[46].

Da im Rahmen dieser Arbeit eine plattform-übergreifende Implementierung der vorgeschlagenen Algorithmen im Vordergrund stand, kamen ausschließlich plattform-neutrale Schnittstellen zum Einsatz. Zur Programmierung spezieller Erweiterungen von PC-Graphikhardware eignet sich vor allem OpenGL. Diese Erweiterungen sollen in den nun folgenden Abschnitten zur Sprache kommen.

2.2.3 Multi-Texturen

Unter Multi-Texturen versteht man die Möglichkeit, innerhalb der Rasterisierungsstufe der Rendering Pipeline mehrere interpolierte Texel für ein Fragment in einem einzigen Schritt nachzuschlagen und diese zu einem Farb- und Opazitätswert zu kombinieren. Multi-Texturen wurden primär wegen der Notwendigkeit eingeführt, ein einzelnes Polygon sowohl mit einer Struktur-Textur als auch mit einer Schatten-Textur zu versehen. Diese Technik wird häufig in 3D-Spielen eingesetzt. Komplexe Beleuchtungsverhältnisse werden in einem Vorverarbeitungsschritt mit Hilfe eines globalen Beleuchtungsverfahrens berechnet. Daraus resultieren so genannte *Shadow-Maps*, die während des Renderings einer 3D-Szene zur Modulation der Helligkeit der einzelnen Texel eines Polygons eingesetzt werden. Inzwischen haben durch die Nachfrage nach weiteren Effekten, wie beispielsweise rauen Oberflächen (*Bump-Maps*) und Spiegelungen (*Environment-Maps*), bis zu acht Multi-Texturen in günstige PC-Graphikhardware Einzug gefunden.

OpenGL 1.2 spezifiziert Multi-Texturierung als eine strenge Sequenz von Texturierungsstufen, in der das Resultat einer vorherigen Texturstufe in der nachfolgenden Stufe verwendet werden kann. Die Kombination der einzelnen Resultate wird dabei durch fest vorgegebene Operationen in der Texturumgebung bestimmt. Diese begrenzte Zahl an Operationen wurde zunächst in den OpenGL Erweiterungen `ARB_texture_env_combine` und `NV_texture_env_combine4` um zusätzliche fest „verdrahtete“ Operationen erweitert[115, 44].

2.2.4 Programmierbare Rasterisierungseinheiten

Die Kombination der nachgeschlagenen Texel, durch in der Hardware fest vorgegebenen Operationen erwies sich jedoch sehr bald als zu restriktiv. Aus diesem Grunde wurden programmierbare Rasterisierungseinheiten eingeführt, die in mehreren Stufen eine Verknüpfung der einzelnen Texel über eine flexible Fragment-Arithmetik zulassen. Die NVIDIA OpenGL Erweiterung `NV_register_combiners` umgeht die gewohnte Texturumgebung und ersetzt diese durch eine registerbasierte Rasterisierungseinheit[136, 78] (siehe Abbildung 2.4). Diese Einheit besteht aus zwei, bzw. in neueren Chip-Generationen aus acht, flexiblen *General-Combiner*-Stufen und einer *Final-Combiner*-Stufe. Jede der *General-Combiner*-Stufen ist in einen RGB-Teil (siehe

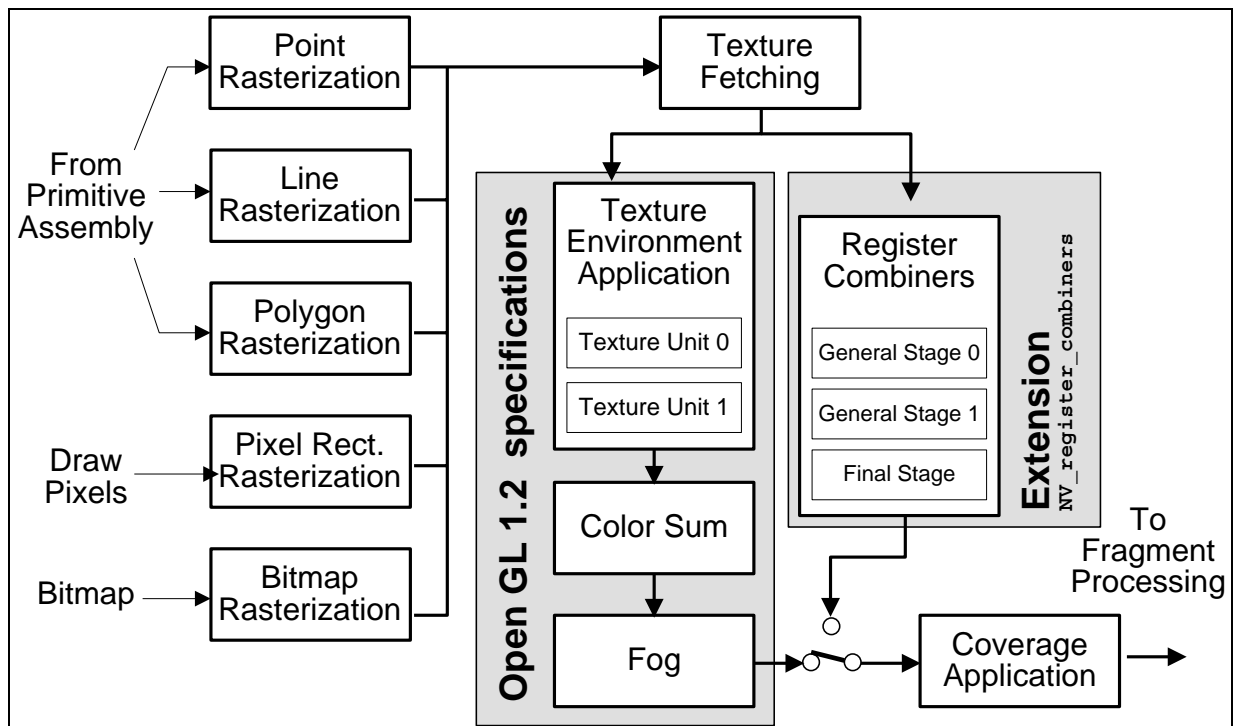


Abbildung 2.4: Da sich das Multi-Textur-Modell von OpenGL 1.2 als zu restriktiv erwies, wurde mit NVIDIAs *GeForce256* Prozessor das Modell der Register-Combiner eingeführt, das die gewohnte Texturumgebung komplett umgeht.

Abbildung 2.5) und einen Alpha-Teil unterteilt. Letztere ist ähnlich wie der RGB-Teil aufgebaut und kann von diesem unabhängig programmiert werden.

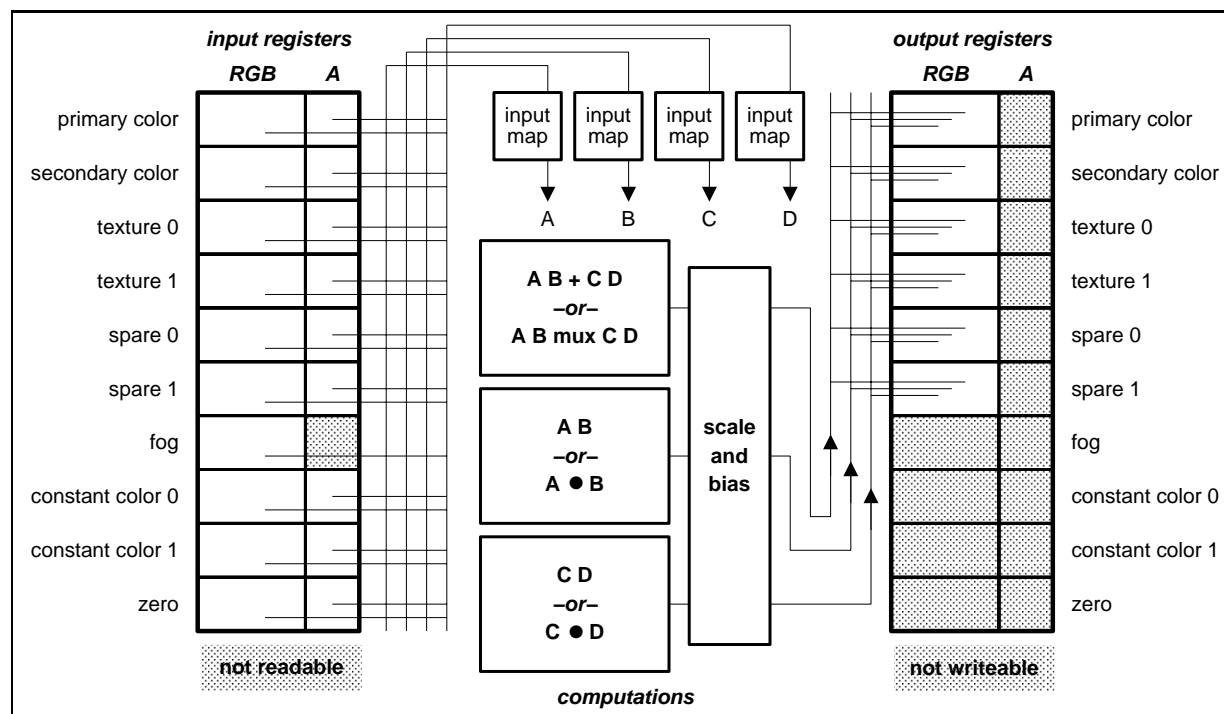


Abbildung 2.5: Der RGB-Teil einer *General-Combiner*-Stufe unterstützt beliebige Eingabevariablenbelegung und komplexe Berechnungen wie Skalarprodukte und komponentenweise gewichtete Summen.

Innerhalb dieser Hardwarearchitektur werden Fragment-Informationen in einem Satz von Eingaberegistern gespeichert, deren Inhalt beliebig auf freie Variablen A , B , C und D gelegt werden kann. Nachdem eine Berechnung beispielsweise in Form eines Skalarprodukts oder einer gewichteten Summe in einer Einheit durchgeführt wurde, kann das Ergebnis noch mit Skalierungs- (*scale*) und Verschiebe-Operationen (*bias*) beeinflusst werden und in eines der Ausgaberegister überführt werden. Diese Ausgaberegister können nun wiederum Eingaberegister für nachgeschaltete Stufen sein. Intern arbeiten diese flexiblen Rasterisierungseinheiten mit höherer Genauigkeit in einem vorzeichenbehafteten Festkomma-Zahlenbereich. Deshalb kann jede Variable mit einer Eingabemaske versehen werden, welche die vorzeichenlosen Eingabe-Farbwerte beispielsweise in den Zahlenbereich $[-1, 1]$ abbildet. Dadurch ergibt sich die Möglichkeit der Speicherung von Vektoren in Farbregistern, ohne diese intern umwandeln zu müssen.

Am Ende dieser Fragment-Pipeline steht zwingend die *Final-Combiner*-Stufe (siehe Abbildung 2.6). Diese Stufe bietet nur die zwei Ausgaberegister RGB und Alpha, die den entgeltigen Fragmentausgaben entsprechen. Während im Alpha-Teil dieser Stufe keine weitere Berechnung durchgeführt werden kann, wird im RGB-Teil fest der Term $AB + (1 - A)C + D$ ausgewertet. Darüber hinaus kann eine der Variablen A bis D mit einem zusätzlichen komponenten-

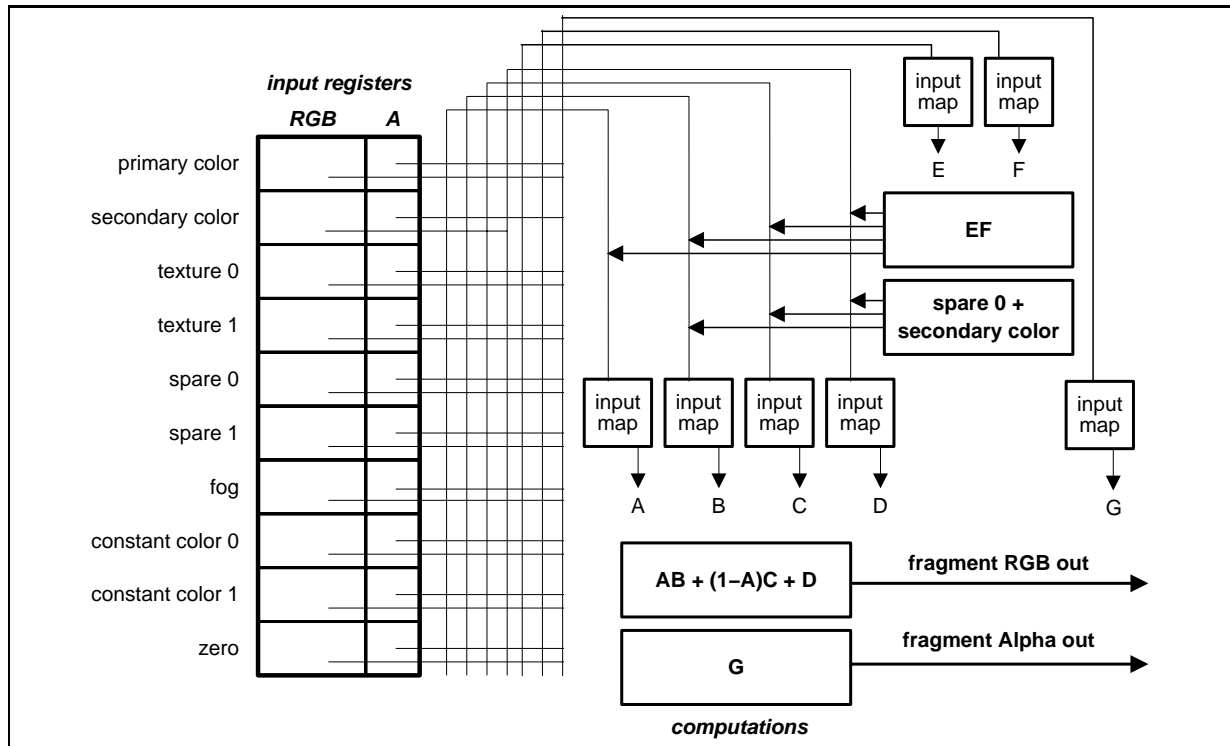


Abbildung 2.6: Die *Final-Combiner*-Stufe berechnet die entgültigen Fragmentausgaben für RGB und Alpha.

weisen Produkt EF belegt werden. Dies dient im Allgemeinen der Erhöhung des Exponenten für den spekularen Anteil im Phong-Beleuchtungsmodell. Anschließend an diese Mehrstufen-Rasterisierungseinheit werden zusätzlich Fragment-Operationen, wie beispielsweise Alpha- und Stencil-Test, auf Basis der Ausgaben der Final-Combiner-Stufe durchgeführt.

Auf Konstanten konnte in den Combinern anfangs nur global über die Belegung zweier konstanter Farbreister zugegriffen werden. Wegen der Erweiterung der Anzahl der General-Combiner-Stufen von zwei auf acht beim NV20 Chip von NVIDIA und der daraus resultierenden Notwendigkeit für zusätzliche Konstanten, wurden weitere Konstanten für jede Combiner-Stufe über die OpenGL Erweiterung `NV_register_combiners2` eingeführt[77, 79].

In einem weiteren Schritt der Flexibilisierung der OpenGL Pipeline wurde beim NV20 Chip die OpenGL Erweiterung `NV_texture_shader` realisiert[27, 80], welche den bis dato festen Mechanismus der Abbildung von Texturkoordinaten in gefilterte Farben durch eine programmierbare Einheit ersetzt. Diese Einheit ist in der Graphikpipeline vor den Fragment-Operationen der Register-Combiner-Einheit angeordnet und für das Nachschlagen gefilterter Texel aus Texturen zuständig (*Texel-Fetch*). Sie besteht aus vier Stufen, die mit jeweils einem von 21 möglichen Texture Shader Programmen belegt werden können. Die Programme unterstützen sowohl konventionelles Texture Mapping als auch abhängige Texturzugrif-

fe (*Dependent-Textures*), Skalarprodukt-Texturprogramme und Spezial-Modi. Die OpenGL-Erweiterung `NV_texture_shader2`[81] erlaubt zusätzlich die Definition von 3D-Textur-Programmen. Jedes der Programme benutzt die interpolierten Texturkoordinaten der jeweiligen Texturstufe (s,t,r,q) und erzeugt zwei Ergebnisse: Ein Resultat der Shader-Stufe, das als Eingabe für ein Textur-Programm einer nachfolgenden Texturstufe genutzt werden kann und ein RGBA-Resultat der Texturinheit, das in den Register-Combinern für weitere Fragment-Operationen benutzt werden kann.

Auch im R200 Chip von ATI sind flexibel programmierbare Fragment-Operationen verfügbar, die über die OpenGL Erweiterung `ATI_fragment_shader` angesprochen werden[56, 42]. Im Gegensatz zur NVIDIA-Architektur werden mit dieser Erweiterung sowohl Texel-Fetch als auch Fragmentoperationen programmiert, wobei die Berechnung der entgültigen Fragmentfarbe in zwei Phasen verlaufen kann. In jeder der beiden Phasen sind 6 Texturzugriffe und nachfolgend 8 Fragmentoperationen möglich. In der zweiten Phase können abhängige Texturzugriffe unter Verwendung der Ergebnisse der ersten Phase realisiert werden. Mehrfache rekursiv-abhängige Texturzugriffe sind durch die zweistufige Architektur im Gegensatz zur NVIDIA-Erweiterung nicht möglich. Dafür ist durch die größere Zahl der möglichen Operationen und das Mischen von Fetch- und Fragment-Operationen ein breiteres Anwendungsspektrum denkbar.

Wesentlicher Nachteil all dieser proprietärer OpenGL Erweiterungen ist natürlich, dass die Idee der architekturneutralen Graphikchnittstelle OpenGL unterminiert wird, und die Programmierung von Anwendungen durch die Anpassung an jede dieser unterschiedlichen Erweiterungen erheblich erschwert wird. Deshalb werden zunehmend Shading-Sprachen, ähnlich Renderman[5], vorgeschlagen, die den direkten Zugriff auf die jeweilige Hardware über eine höhere Abstraktionsschicht erlauben[94, 93]. Auch für die kommende OpenGL Version 2.0 existiert inzwischen ein Entwurf für eine Shading-Sprache[70].

Direct3D erlaubt den Zugriff auf Texel-Fetch- und Fragment-Operationen über die *Pixel-Shader* API. Hier wurde von Beginn an eine sehr einfache Shading-Sprache realisiert, die aber durch zu kurzfristige Spezifikationen ebenfalls herstellerepezifisch fragmentiert ist. So unterstützt NVIDIA augenblicklich die *Pixel-Shader* API nur bis zur Version 1.1 (DirectX 8.0), während ATI auf der Version 1.4 der *Pixel-Shader* aufsetzt (DirectX 8.1). Diese Fragmentierung soll in DirectX9 durch die Einführung einer mächtigeren Shading-Sprache behoben werden.

2.3 Volumenvisualisierung

Der Begriff Volumenvisualisierung[28] bezeichnet die Repräsentation, Manipulation und Darstellung von Volumendaten. Er umfasst alle Stufen der Visualisierungspipeline, angefangen vom *Filtering* und *Mapping*, bis zur algorithmischen Bildgenerierung, die oft auch als *Rendering* bezeichnet wird. Im Zusammenhang mit der Erzeugung von Bildern aus Volumendaten wird deshalb häufig von *Volume-Visualization* und *Volume-Rendering* gesprochen.

3D-Skalarfelder sind mathematisch durch die Abbildung $f : R^3 \rightarrow R$ beschrieben. Da Volumendaten aus Messungen und Simulationen üblicherweise nur an diskreten Gitterpositionen im Raum gegeben sind, werden zur Bestimmung skalarer Werte an beliebigen Stellen in Raum

benachbarte Datenpunkte interpoliert. Volumendaten werden in einer Reihe wissenschaftlicher Anwendungen in unterschiedlicher Weise ermittelt: In Simulationen werden aufgrund eines mathematischen Modells physikalische Vorgänge simuliert und damit beispielsweise die Dichtewerte eines Gases an verschiedenen Raumpunkten während einer Strömungssimulation ermittelt. Messungen bestimmen physikalische Größen im Raum mit entsprechender Sensorik, wie zum Beispiel die Abschwächung der Röntgenstrahlung durch einen Computertomographen in der medizinischen Bildverarbeitung. Oft lassen sich Volumendaten auch direkt berechnen, so im Falle der Aufenthaltswahrscheinlichkeit von Elektronen um einen Atomkern oder in einem Molekül, die zur Visualisierung von Molekülorbitalen benötigt wird.

Im Wesentlichen lassen sich Visualisierungsverfahren für Volumendaten durch die Einteilung in indirekte und direkte Methoden klassifizieren.

2.3.1 Indirekte Volumenvisualisierung

Indirekte Volumenvisualisierungsverfahren bilden die Volumendaten zunächst in eine intermediäre Repräsentation ab, die dann oftmals mit geringerem Aufwand dargestellt werden kann. Diese Abbildung beinhaltet meistens einen Informationsverlust, der aber bewusst hingenommen wird, da eventuell gar kein Interesse an einer „ganzheitlichen“ Darstellung der Daten besteht.

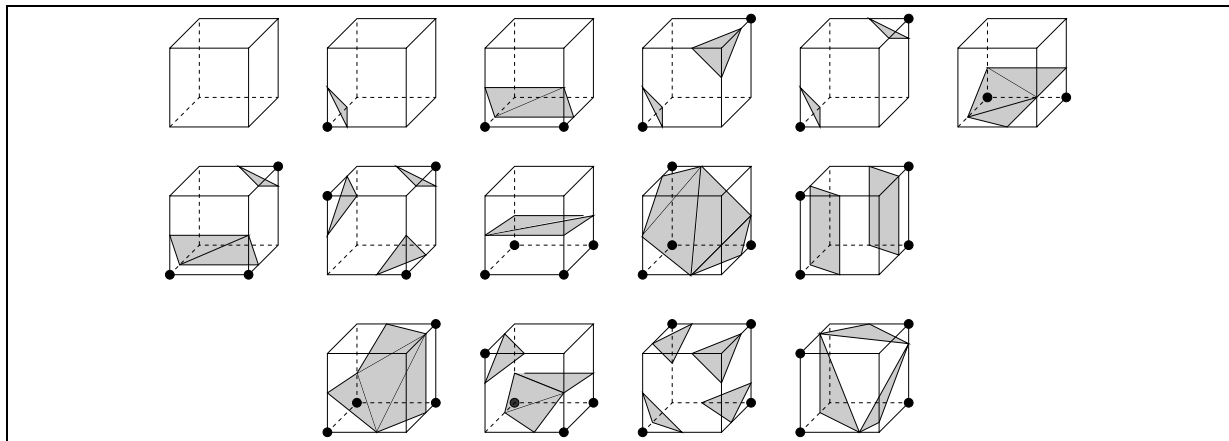


Abbildung 2.7: Marching-Cubes-Konfigurationen.

Einer der bekanntesten Vertreter dieser Klasse von Algorithmen ist das *Marching-Cubes*-Verfahren[91] von Lorensen und Cline. *Marching-Cubes* rekonstruiert für einen gegebenen Schwellwert (*Isowert*) eine Grenzfläche (*Isosfläche*) bestehend aus Dreiecken mit Normalen an den Eckpunkten der Dreiecke. Dazu werden die Eckpunkte jedes Subwürfels des Volumens bezüglich des Isowertes als größer oder kleiner klassifiziert und daraus für jeden Subwürfel ein Index erstellt. Mit diesem Index wird aus einer Tabelle eine von 256 möglichen Dreiecks-Konfigurationen nachgeschlagen. Aus Symmetriegründen lassen sich 15 Basiskonfigurationen ableiten (siehe Abbildung 2.7). Die Ecken der Dreiecke ergeben sich durch lineare Interpolation

entlang der Subwürfelkanten. Die Normalen an den Vertizes ergeben sich ebenfalls durch lineare Interpolation der Normalen an den Ecken der Subwürfel. An den Ecken der Subwürfel werden die Normalen beispielsweise durch die zentrale Differenzenmethode ermittelt. So erzeugte Dreiecksgitter können mit konventioneller Graphikhardware dargestellt werden. Mehrdeutigkeiten bei den Marching-Cubes-Konfigurationen wurden von Nielson und Hamann behandelt[104].

Da die Zahl der durch einen Marching-Cubes-Algorithmus generierten geometrischen Primitive für realistisch dimensionierte Volumendatensätze im Bereich von mehreren Millionen liegen kann, wurden von verschiedenen Autoren Optimierungen des Algorithmus vorgeschlagen. Das Ziel dieser Optimierungen ist mannigfaltig: Gitter-Dezimierungs- und Reorganisierungsverfahren[23, 41, 66, 108, 127, 129] versuchen die sich ergebende Flächenrepräsentation in der Weise zu konvertieren, dass die Zahl der zu verarbeitenden Primitive reduziert wird, um so die Fläche mit erhöhter Geschwindigkeit darstellen zu können. Diese Ansätze verlagern die Optimierung der Gitter in einen Nachbearbeitungsschritt, während auch schon Ansätze existieren, welche die Optimierung bereits während der Rekonstruktion erledigen[118]. Andere Techniken adressieren dagegen direkt das Problem einer schnellen Zelltraversierung. Dadurch kann vermieden werden, Bereiche des Volumens zu bearbeiten, welche die Isofläche nicht enthalten. Die Vorteile von Octree-Repräsentationen zur schnelleren Rekonstruktion von Isoflächen aus regulären Volumendaten wurde als erstes in [150] beschrieben. In den letzten Jahren wurden verschiedene verwandte Techniken vorgeschlagen, wie eine verbesserte Partitionierung, inkrementelle Aktualisierungstechniken[7, 130, 131], effiziente Zellsuche mit Intervalldatenstrukturen[15, 16, 90], und zuletzt Verfahren, welche die hierarchische Natur von Multi-Skalen Repräsentation nutzen[109, 147]. Hybride Ansätze, wie sie in [89] vorgestellt wurden, versuchen die Rekonstruktion von Primitiven zu vermeiden, die nicht zur aktuellen Ansicht beitragen.

Diese Vielzahl der Techniken macht den großen Bedarf für Optimierungen zur Visualisierung von Isoflächen an einem Arbeitsplatzrechner deutlich. Eine interaktive Visualisierung hoch aufgelöster Volumendaten in einem Client-Server-Umfeld stellt eine zusätzliche Herausforderung dar. In dieser Arbeit werden grundsätzlich drei verschiedene Wege zur indirekten Volumenvisualisierung besprochen. Sie werden in Kapitel 5 ausführlich erläutert.

2.3.2 Direkte Volumenvisualisierung

Während indirekte Verfahren der Volumenvisualisierung auf einer Zwischenrepräsentation der Originaldaten arbeiten, werden bei direkten Methoden die Volumendaten selbst zur Darstellung herangezogen. Im Gegensatz zur indirekten Volumenvisualisierung, die häufig mit Informationsverlust behaftet ist, ergibt sich durch direkte Verfahren eine „ganzheitliche“ Visualisierung der Daten.

Seit ihrer Einführung in den späten 1980er Jahren wurden im Umfeld der direkten Volumenvisualisierung einige mehr oder weniger unterschiedliche Techniken entwickelt, die sich grob in zwei Klassen einteilen lassen:

Bildraumverfahren (*image order*): Ausgehend von den Bildpunkten des Zielbildes werden Strahlen in die Szene geschickt, die den Beitrag des Volumens zu einem Punkt der Bildebe-

ne ermitteln. Werden nur primäre Sichtstrahlen betrachtet so wird vom *Ray-Casting*-Verfahren gesprochen (siehe Abbildung 2.8).

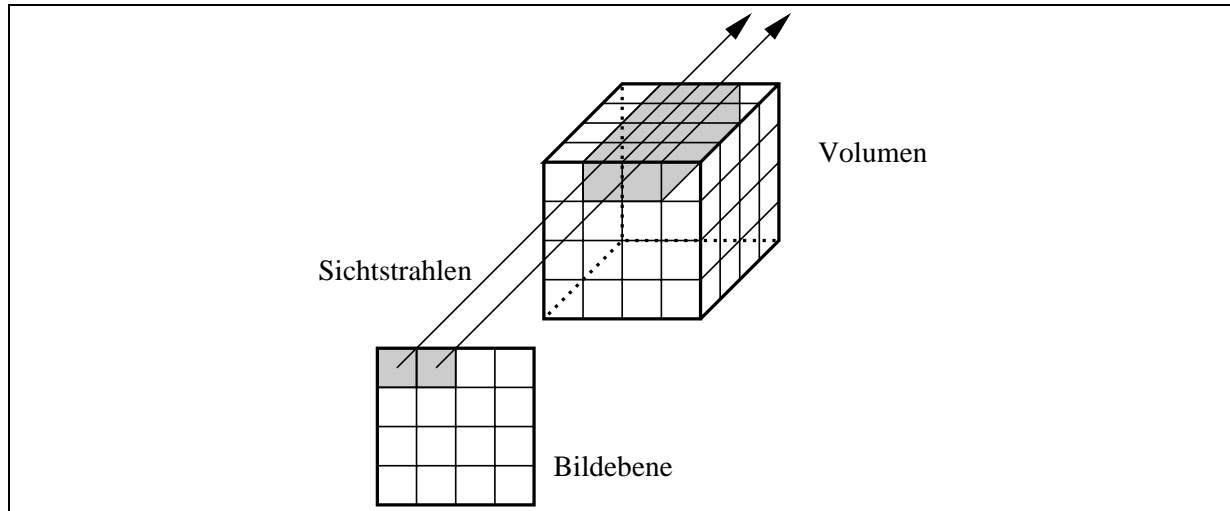


Abbildung 2.8: Ray-Casting mit Parallelprojektion.

Objektraumverfahren (*object order*): Ausgehend von den Voxeln des Volumens werden deren Beiträge zu jedem Bildpunkt des Zielbildes bestimmt und zur Farbe des Punktes integriert. Beim *Splattting*-Verfahren[148] (Abbildung 2.9) erfolgt beispielsweise zunächst eine Projektion des Zentrums der Voxel in den Bildraum. In der Umgebung dieser Punkte wird dann ein so genannter *Footprint* in das Ergebnisbild eingefügt. Der Footprint entsteht aus einer Faltung des Voxelwerts mit einem Filter, der den Beitrag eines Voxels auf benachbarte Pixel verteilt.

Daneben existieren zwei orthogonale Konzepte:

Frequenzraumverfahren (*frequency domain*): Durch die Transformation der Volumendaten in den Frequenzraum kann das Zielbild einfach berechnet werden. Daraus wird durch eine inverse Transformation das Ergebnisbild im Ortsraum erzeugt. Allerdings muss dazu die Absorption vernachlässigt werden, was zu Röntgenbild-artigen Darstellungen führt. Zu der Klasse der Frequenzraumverfahren zählt das Fourier-Domain-Volume-Rendering[92, 140] (siehe Abbildung 2.10).

Kompressionsraumverfahren (*compression domain*): Bei diesen Verfahren werden die Volumendaten zunächst mittels hierarchischer Basisfunktionen, also beispielsweise mit einer Wavelet-Basis[73] komprimiert. Der große Vorteil des Verfahrens liegt darin, dass das Verfahren ohne eine explizite Dekomprimierung der gesamten Daten auskommt. Das Rendering der Volumendaten kann direkt auf der komprimierten Fassung der Volumendaten mit lokaler Dekomprimierung erfolgen[145, 144, 49]. Dies führt zu drastisch reduziertem Speicherbedarf und verbesserten Rendering-Zeiten. Fraktale Volumenkompression wurde in [19] untersucht. Eine Kombination

von Frequenzraumverfahren und Kompressionsraumverfahren für das verteilte Rendering in einem Client-Server-Umfeld wurde von Gross et al. in [47] vorgestellt.

Gemeinsam ist diesen Verfahren jedoch die (approximative) Auswertung des Volume-Rendering-Integrals[84] für jeden Bildpunkt des Zielbilds, also die Integration von abgeschwächten Farb- und Absorptionskoeffizienten entlang eines Sichtstrahls. Im Folgenden wird davon ausgegangen, dass der Sichtstrahl $\mathbf{x}(\lambda)$ in Abhängigkeit vom Abstand zur Betrachterposition parametrisiert ist und dass die Farbdichten $\text{color}(\mathbf{x})$ zusammen mit den Absorptionsdichten $\text{extinction}(\mathbf{x})$ für jeden Punkt im Raum \mathbf{x} gegeben sind. Die Einheiten der Farb- und Absorptionsdichten sind Farbintensität und Absorptionsstärke pro Längeneinheit. Im weiteren Verlauf wird aber von Farb- und Absorptionskoeffizienten die Rede sein, wenn die genaue Bedeutung aus dem Kontext klar wird. Für jeden Sichtstrahl ergibt sich die resultierende Intensität I aus der Volume-Rendering-Gleichung

$$I = \int_0^D \text{color}(\mathbf{x}(\lambda)) \exp\left(-\int_0^\lambda \text{extinction}(\mathbf{x}(\lambda')) d\lambda'\right) d\lambda$$

mit der maximalen Distanz D , d.h. es existiert keine Farbdichte $\text{color}(\mathbf{x}(\lambda))$ für λ größer als D und kleiner als 0 (siehe Abbildung 2.11). Die Gleichung drückt aus, dass Farbe an jedem Punkt \mathbf{x} bezüglich der Funktion $\text{color}(\mathbf{x})$ emittiert wird und durch die integrierten Absorptionskoeffizienten $\text{extinction}(\mathbf{x})$ zwischen dem Blickpunkt und dem Emissionspunkt abgeschwächt wird.

Leider ist diese Form der Volume-Rendering-Gleichung nicht zur Beschreibung der Visualisierung eines kontinuierlichen Skalarfeldes $s(\mathbf{x})$ geeignet, da die Berechnung der Farb- und Absorptionskoeffizienten nicht spezifiziert ist. Zwei Schritte werden bei der Ermittlung dieser Farb- und Absorptionskoeffizienten unterschieden: Die *Klassifikation* ist die Zuweisung einer primären Farbe und des Absorptionskoeffizienten zu einem Skalarwert, wobei der Begriff der

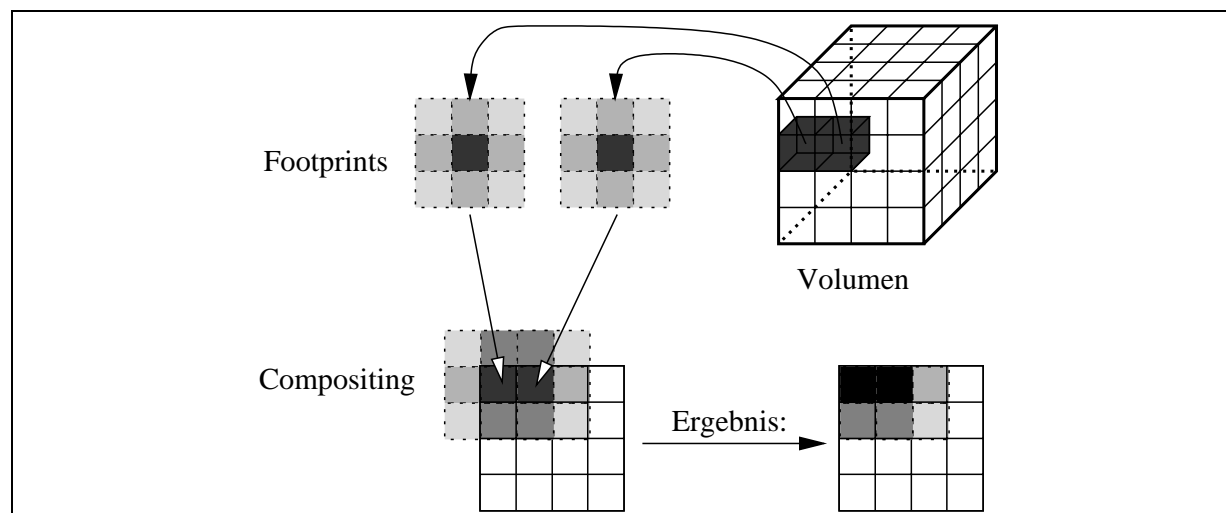


Abbildung 2.9: Splatting.

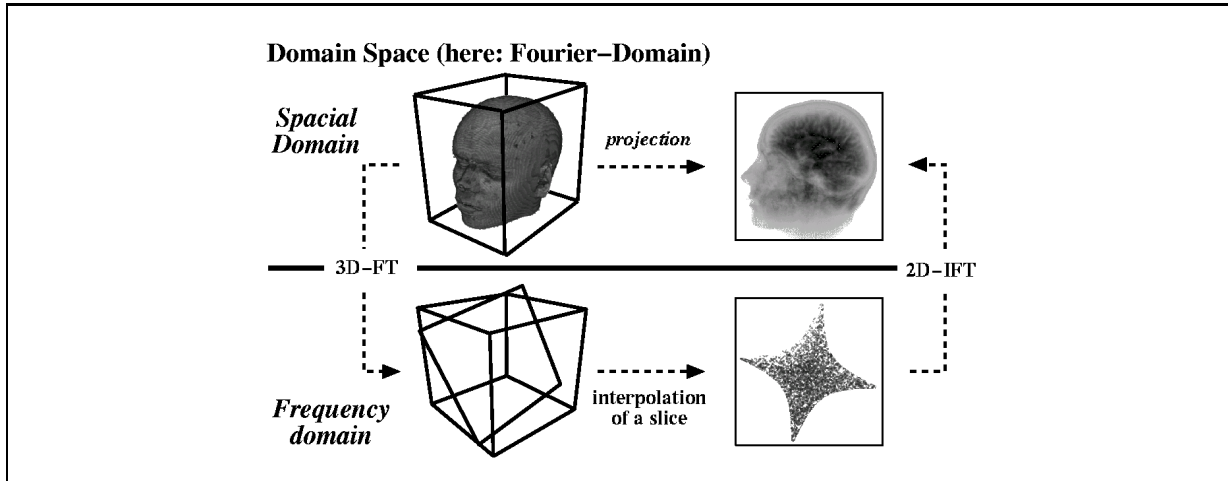


Abbildung 2.10: Fourier-Domain-Volume-Rendering.

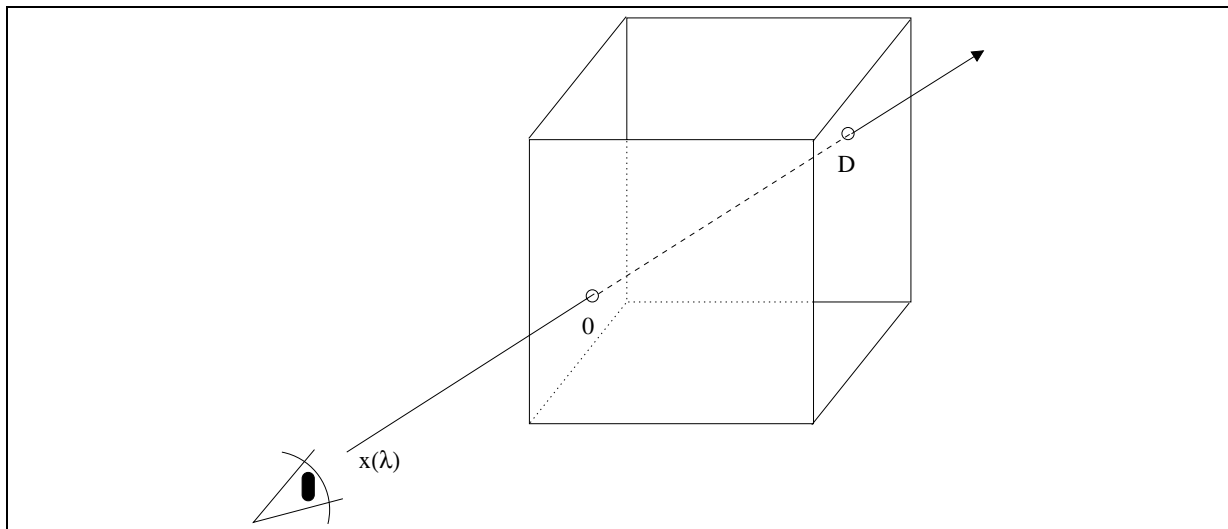


Abbildung 2.11: Parameter des Volume-Rendering-Integrals.

primären Farbe von OpenGL übernommen wurde, um anzudeuten, dass es sich um die Farbe vor der Beleuchtungsberechnung (*Shading*) handelt. Eine Klassifikation wird durch die Einführung von Transferfunktionen für die Farbdichten $\tilde{c}(s)$ und Absorptionsdichten $\tau(s)$ erreicht, welche die Skalarwerte aus dem Volumen $s = s(\mathbf{x})$ auf Farb- und Absorptionskoeffizienten abbilden. Allgemein ist dabei \tilde{c} ein Vektor, der die Farbe in einem beliebigen Farbraum angibt, während es sich bei τ um einen skalaren Absorptionskoeffizienten handelt.

Der zweite Schritt ist das Shading, das den Beitrag eines Punktes im Raum, also die Funkti-

on $\text{color}(\mathbf{x})$, berechnet. Das Shading hängt natürlich von der Primärfarbe ab, kann aber auch von anderen Parametern, wie beispielsweise dem Gradienten $\nabla s(\mathbf{x})$ des Skalarfelds, ambienten und diffusen Lichtparametern etc. beeinflusst werden. Im weiteren Verlauf dieses Abschnitts wird allein die Klassifikation eine Rolle spielen. Das Shading wird an verschiedenen Stellen dieser Arbeit noch behandelt werden. Aus diesem Grunde wird zunächst von einem trivialen Shading ausgegangen, d.h. $\text{color}(\mathbf{x})$ wird durch die in der Klassifikation zugewiesene Primärfarbe $\tilde{c}(s(\mathbf{x}))$ ersetzt. Analog wird $\text{extinction}(\mathbf{x})$ durch den in der Klassifikation zugewiesenen Absorptionskoeffizienten $\tau(s(\mathbf{x}))$ ersetzt. Mit diesen Vereinbarungen kann die Volume-Rendering-Gleichung in die folgende Form gebracht werden:

$$I = \int_0^D \tilde{c}(s(\mathbf{x}(\lambda))) \exp\left(-\int_0^\lambda \tau(s(\mathbf{x}(\lambda'))) d\lambda'\right) d\lambda \quad (2.1)$$

2.3.3 Prä- und Post-Klassifikation

Direkte Volumenvisualisierungstechniken unterscheiden sich wesentlich in der Auswertung von Gleichung (2.1). Ein wichtiger und wesentlicher Unterschied ist die Berechnung von $\tilde{c}(s(\mathbf{x}))$ und $\tau(s(\mathbf{x}))$. Tatsächlich ist das kontinuierliche Skalarfeld $s(\mathbf{x})$ üblicherweise durch ein Gitter mit skalaren Werten s_i an jedem Gitterpunkt \mathbf{v}_i beschrieben, aus dem mit einer Interpolationsvorschrift alle weiteren Punkte berechnet werden können.

Die Reihenfolge der Interpolation und der Anwendung der Transferfunktionen definiert den Unterschied zwischen *Prä-* und *Post-Klassifikation*. Post-Klassifikation ist durch die Anwendung der Transferfunktionen *nach* der Interpolation von $s(\mathbf{x})$ aus den umgebenden Skalarwerten charakterisiert. Im Gegensatz dazu wird bei der Prä-Klassifikation die Transferfunktion *vor* dem Interpolationsschritt angewandt, d.h. Farben $\tilde{c}(s_i)$ und Absorptionskoeffizienten $\tau(s_i)$ werden in einem ersten Schritt für jeden Gitterpunkt \mathbf{v}_i nachgeschlagen und dann für die Berechnung des Volume-Rendering-Integrals $\tilde{c}(s(\mathbf{x}))$ und $\tau(s(\mathbf{x}))$ interpoliert.

Offensichtlich produzieren Prä- und Post-Klassifikation unterschiedliche Ergebnisse, sobald die Interpolation zweier Skalarwerte des Volumens nicht mit der Anwendung der Transferfunktionen vertauschbar ist. Da die Interpolation üblicherweise nichtlinear ist (beispielsweise tri-linear in kartesischen Gittern), kann diese nur dann mit der Transferfunktionen vertauscht werden, wenn die Transferfunktion eine lineare Funktion durch den Ursprung oder die Identitätsfunktion ist. In *allen* anderen Fällen führt Prä-Klassifikation zu Abweichungen von der Post-Klassifikation (siehe Abbildung 2.12). Letztere kann in dem Sinne als „korrekt“ angesehen werden, als sie die Anwendung einer Transferfunktion auf ein kontinuierliches Skalarfeld annimmt, das durch ein Gitter zusammen mit einer Interpolationsvorschrift gegeben ist. Nichtsdestoweniger ist Prä-Klassifikation unter bestimmten Umständen durchaus nützlich, da sie als eine grundlegende Segmentierungstechnik eingesetzt werden kann.

In der Literatur gibt es einige Verwirrung über die Korrektheit der Prä-Klassifikation mit linearen Transferfunktionen. Wie die obige Diskussion zeigt, ist dazu eine lineare Interpolation notwendig, wie sie bei Tetraeder-Gittern nicht aber bei kartesischen Gittern üblich ist. Allerdings korrespondiert $\tilde{c}(s)$ üblicherweise mit einem Produkt von Farbe $c(s)$ und einem Absorptionskoeffizienten $\tau(s)$, in diesem Zusammenhang „Dichte“ und in der diskretisierten Form „Opazität“

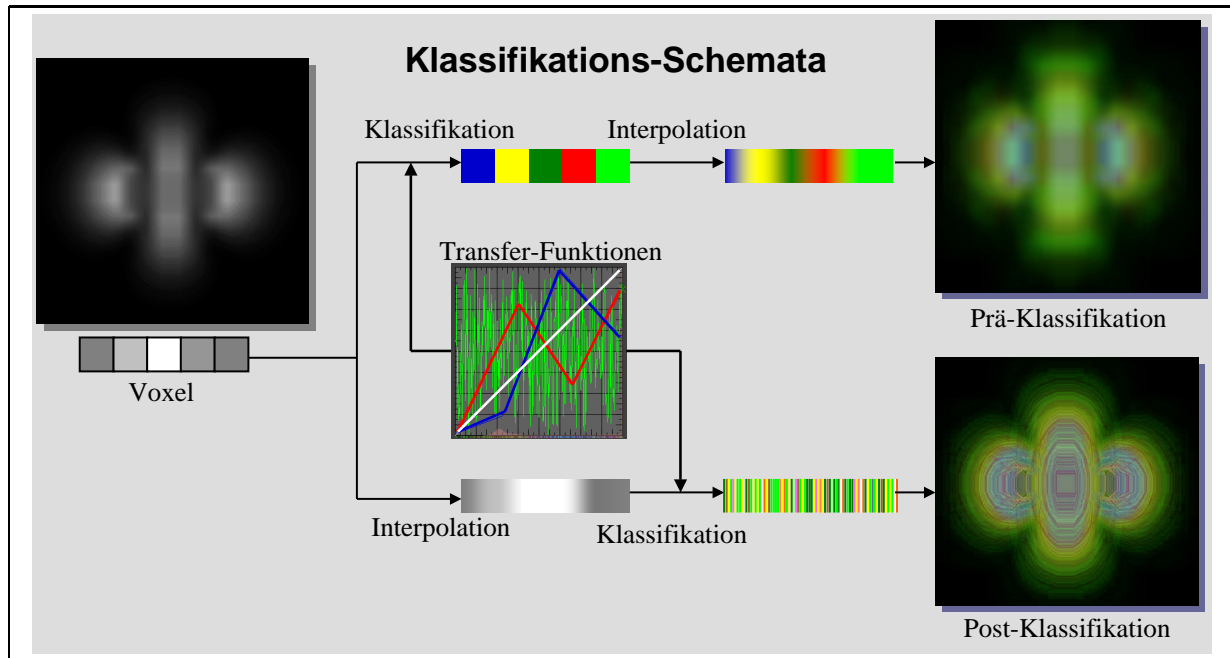


Abbildung 2.12: Unterschiede der Klassifikations-Schemata am Beispiel einer sphärischen harmonischen Funktion mit nur 16^3 Voxeln Auflösung (links). Zur Verdeutlichung der Unterschiede zwischen Prä- (rechts, oben) und Post-Klassifikation (rechts, unten) wird eine hochfrequente Transferfunktion für den Grün-Kanal eingesetzt, während alle anderen Transferfunktionen durch stückweise lineare Funktionen gegeben sind. Die deutlichen Unterschiede in den Ergebnissen sind durch die Reihenfolge der Anwendung von Interpolation und Klassifikation zu erklären.

genannt:

$$\tilde{c}(s) = \tau(s)c(s)$$

(Für einen Vergleich zwischen der direkten Spezifikation der Transferfunktion $\tilde{c}(s)$ und der Spezifikation von $c(s)$ sei auf [95] verwiesen.)

Deshalb ist $\tilde{c}(s)$ nur dann linear, wenn entweder die Transferfunktion für die Farbe $c(s)$ linear ist und $\tau(s)$ konstant oder die Transferfunktion des Absorptionskoeffizienten $\tau(s)$ linear und $c(s)$ konstant ist. Diese Restriktion kann durch die Definition einer Transferfunktion für $\tilde{c}(s)$ [149] oder durch die Verwendung vormultiplizierter Farben[152] abgeschwächt werden. Allerdings sind auch in diesem Falle die Transferfunktionen für (vormultiplizierte) Farben und Absorptionskoeffizienten auf lineare Funktionen beschränkt.

Für nichtlineare Transferfunktionen, eingeschlossen stückweise lineare Funktionen und/oder nicht-lineare Interpolation wird nur Post-Klassifikation das kontinuierliche Skalarfeld korrekt visualisieren, vorausgesetzt, dass das Volume-Rendering-Integral mit ausreichender Genauigkeit ausgewertet wird.

2.3.4 Numerische Integration

Da eine analytische Auswertung des Volume-Rendering-Integrals nur in bestimmten Fällen, speziell bei linearer Interpolation und bei stückweise linearer Transferfunktion[151] möglich ist, wird eine numerische Integration benötigt. Die verbreitetste numerische Approximation des Volume-Rendering-Integrals ist die Berechnung von Riemann-Summen für n gleiche Strahlsegmente der Länge $d = D/n$ (siehe auch Bild 2.13 und Sektion IV.A in [95]). Die folgenden Überlegungen lassen sich leicht für Strahlsegmente mit unterschiedlicher Länge generalisieren.

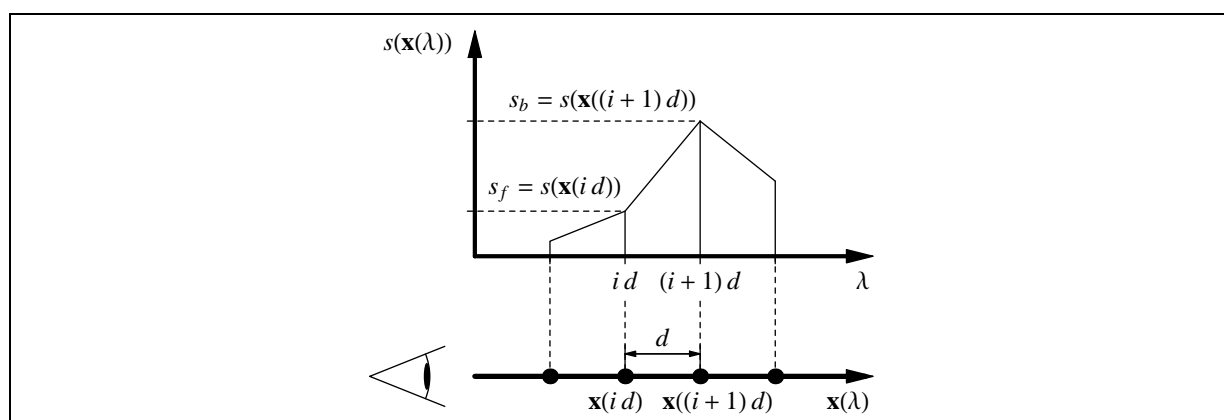


Abbildung 2.13: Schematische Darstellung der Parameter zur Ermittlung der Farben und Opazitäten des i -ten Strahlsegments.

Der Faktor

$$\exp\left(-\int_0^\lambda \tau(s(\mathbf{x}(\lambda'))) d\lambda'\right)$$

in Gleichung (2.1) wird durch

$$\exp\left(-\sum_{i=0}^{\lambda/d} \tau(s(\mathbf{x}(i d))) d\right) = \prod_{i=0}^{\lambda/d} \exp\left(-\tau(s(\mathbf{x}(i d))) d\right) = \prod_{i=0}^{\lambda/d} (1 - \alpha_i)$$

approximiert, wobei die *Opazität* α_i des i -ten Strahlstücks durch

$$\alpha_i \approx 1 - \exp\left(-\tau(s(\mathbf{x}(i d))) d\right)$$

definiert wird. Für kleine d wird dies häufig weiter durch $\alpha_i \approx \tau(s(\mathbf{x}(i d))) d$ abgeschätzt. Der Term $1 - \alpha_i$ wird im Folgenden *Transparenz* des i -ten Strahlsegments genannt. Die Farbe \tilde{C}_i , die im i -ten Strahlstück emittiert wird, kann durch $\tilde{C}_i \approx \tilde{c}(s(\mathbf{x}(i d))) d$ approximiert werden. Daraus folgt die Approximation des Volume-Rendering-Integrals in Gleichung (2.1) zu

$$I \approx \sum_{i=0}^n \tilde{C}_i \prod_{j=0}^{i-1} (1 - \alpha_j) \quad (2.2)$$

In iterativer Form ergibt sich daraus folgender *Back-to-Front-Compositing*-Algorithmus:

$$\tilde{C}'_i = \tilde{C}_i + (1 - \alpha_i) \tilde{C}'_{i+1}. \quad (2.3)$$

Dabei wird über die Strahlstücke i iteriert und die akkumulierte Farbe im i -ten Strahlstück ergibt sich zu \tilde{C}'_i .

$\tilde{c}(s)$ wird oft durch $\tau(s)c(s)$ ersetzt[95]. In diesem Falle ergibt sich aus der Approximation

$$C_i \approx \tau(s(\mathbf{x}(i d))) c(s(\mathbf{x}(i d))) d$$

die geläufigere Approximation

$$I \approx \sum_{i=0}^n \alpha_i C_i \prod_{j=0}^{i-1} (1 - \alpha_j)$$

mit der entsprechenden *Back-to-Front-Compositing*-Gleichung

$$C'_i = \alpha_i C_i + (1 - \alpha_i) C'_{i+1}. \quad (2.4)$$

Diese Compositing-Gleichung ist auch als *Over-Operator* bekannt und wurde erstmals von Porter und Duff[117] zum Mischen von Bildern vorgeschlagen. Die Compositing-Gleichung zeigt, dass \tilde{C} mit einer *vormultiplizierten Farbe* αC korrespondiert. Diese wird auch *opazitäts-gewichtete Farbe* oder *assoziierte Farbe* genannt. Gemäß Blinn[11] sind assoziierten Farben ihrer Opazität zugeordnet, d.h. es sind reguläre Farben, die auf Schwarz gemischt werden. Blinn vermerkt auch, dass aus einigen Intensitätsberechnungen assoziierte Farben resultieren, obwohl diese nicht explizit mit ihrer Opazität multipliziert werden. So gesehen ist die Transferfunktion $\tilde{c}(s)$ eigentlich die Transferfunktion für eine assoziierte Farbdichte.

Eine kohärente Diskretisierung des Volume-Rendering-Integrals kann als Diskretisierung des Volumens in ausgedehnte Schichten, sogenannte *Slabs*, angesehen werden. Ein Slab emittiert Licht und schwächt gleichzeitig das Licht der dahinter liegenden Slabs ab. Allerdings wird das Licht, das in einem Slab emittiert wird, nicht durch den Slab selbst abgeschwächt. Die diskrete Approximation des Volume-Rendering-Integrals wird für $d \rightarrow 0$ gegen das korrekte Resultat konvergieren, d.h. für hohe Samplingraten $n/D = 1/d$.

2.3.5 Texturbasiertes Volume-Rendering

Um die typischerweise großen Datensätze aus heutigen wissenschaftlichen Anwendungen auf einem Einprozessor-Arbeitsplatzrechner mit interaktiven Bildwiederholraten darstellen zu können, ist die Nutzung von Graphikhardware[113, 112, 3, 100, 128, 98, 114] zwingend notwendig. Die Verwendung von Texturen für das Volume-Rendering wurde erstmals von Cabral[13] vorgeschlagen. Bei der Definition der Volumendaten als 3D-Textur (OpenGL 1.2) werden die Volumendaten im Texturspeicher der Graphikhardware abgelegt. Für das Rendering der 3D-Textur werden,

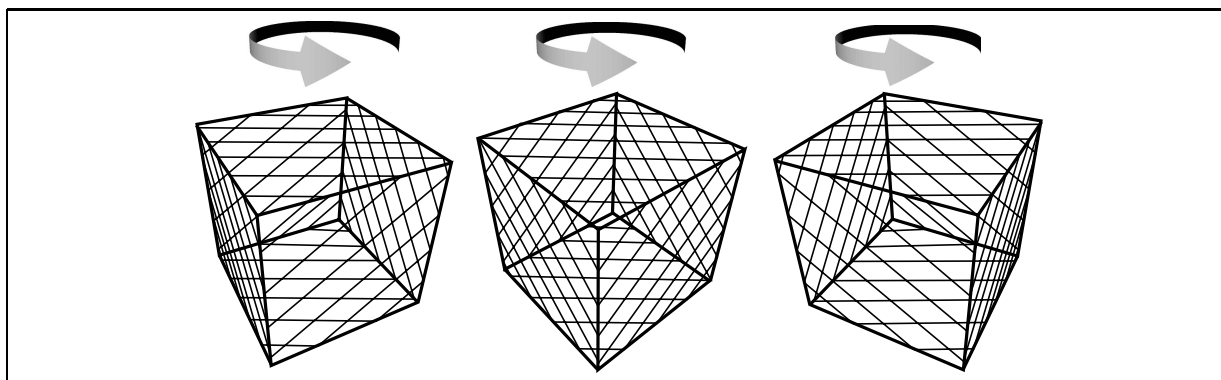


Abbildung 2.14: Viewport-parallele Schichten bei Verwendung von 3D-Texturen. Die Schichtpolygone werden für jede Ansicht neu ermittelt.

unter Berücksichtigung der Blickrichtung, Polygone durch den Schnitt äquidistanter, viewport-paralleler Ebenen mit der Bounding-Box des Volumens erzeugt (siehe Abbildung 2.14). Dabei ist es notwendig, die Polygone für jede neue Betrachterposition neu zu berechnen. Jedem Eckpunkt der Polygone wird eine Textur-Koordinate zugeordnet. Die Polygone werden aus Sicht des Betrachters von hinten nach vorne unter Verwendung des *Over-Operators* im Framebuffer überblendet, wobei während der Rasterisierung sowohl eine trilineare Interpolation der Volumendaten auf den Schichtpolygonen durchgeführt wird, als auch eine Klassifikation der Skalarwerte in RGBA-Werte über eine Indextabelle erfolgt. Die Reihenfolge von Klassifikation und Interpolation und damit das Ergebnis hängt dabei wesentlich von der verwendeten Hardware mit entsprechenden OpenGL Erweiterungen ab (siehe auch Abschnitt 2.3.3). Die OpenGL Erweiterung `SGI_texture_color_table` implementiert Post-Klassifikation, während die Erweiterung `EXT_paletted_texture` Prä-Klassifikation implementiert.

Durch das Rendern zusätzlicher Schichtpolygone kann die Bildqualität dieses Verfahrens erhöht werden. Dies entspricht dem Erhöhen der Samplingrate durch Verringerung des Schichtabstands. Dabei ist eine Skalierung der Opazität der Schichtpolygone notwendig.

Entsprechend Abschnitt 2.3.4 ist bei diesem Verfahren ein Slab der Raum zwischen zwei aneinandergrenzenden Schichten. Durch den Schnitt eines Sehstrahls mit den Slabs werden Strahlsegmente definiert. Für orthogonale Projektion ergeben sich durch die Wahl äquidistanter Schichtpolygone Strahlsegmente mit konstanter Länge. Da dies bei perspektivischer Projektion nicht mehr zutrifft, wurde in [88] ein Verfahren vorgeschlagen, bei dem anstelle von Schichtpolygonen, um den Betrachter zentrierte Kugelschalenelemente[54, 88] (*Shells*) aus dem Schnitt der Kugelschalen mit der Bounding-Box des Volumens erzeugt werden. Daraus ergibt sich auch für eine perspektivische Projektion eine konstante Abtastrate des Volumens entlang jedes Strahls.

Eine Variante dieses Algorithmus stellt das 2D-Textur-basierte Volume-Rendering dar. Anstelle der Definition des Volumens als eine 3D-Textur wird das Volumen als Stapel von 2D-Texturen definiert. Es werden drei Schicht-Stapel verwendet, deren Schichten jeweils parallel zur xy -, xz - und yz -Ebene liegen. Dabei wird derjenige Schichtstapel für die Darstellung ausgewählt, bei dem

der Winkel zwischen der Schicht-Normale und dem Viewing-Vektor minimal ist. Der Wechsel der Schicht-Normale, der einer Auswahl des entsprechenden Schicht-Stapels entspricht, ist in Abbildung 2.15 dargestellt. Die Schichten eines Stapels werden wieder aus Sicht des Betrachters von hinten nach vorne unter Verwendung des *Over-Operators* im Framebuffer überblendet, wobei während der Rasterisierung anstelle der trilinearen Interpolation nur eine bilineare Interpolation tritt. Die Klassifikation erfolgt dagegen wie gewohnt. Dieser Ansatz ist in gewisser Weise ähnlich zum Shear-Warp-Algorithmus[87, 85, 86]. Allerdings wird dabei keine explizite Faktorisierung der Viewing-Transformation vorgenommen.

Vorteile dieses Verfahrens sind neben der besseren Verfügbarkeit von 2D-Texturen die im Allgemeinen höheren Bildwiederholraten, die sich im Wesentlichen aus der verbesserten Ausnutzung schneller Caches auf dem Graphikchip ergeben. Daneben sind aber auch einige wesentliche Nachteile, wie dreifacher Speicherbedarf, fehlende trilineare Interpolation und Stapel-Umschalt-Effekte bei Stapelwechseln in Kauf zu nehmen. Möglichkeiten, diese Probleme zu beseitigen, werden in Abschnitt 4.2.1 behandelt.

Ein diesen Verfahren immanentes Problem ist, dass eine korrekte Rekonstruktion des Signals nach dem Abtast-Theorem von Nyquist zwar für Samplingraten über der Nyquist Frequenz möglich ist, die notwendige Samplingrate aber bei nicht-linearen Features in den Transferfunktionen erheblich ansteigen kann. Dies folgt daraus, dass die Nyquist-Frequenz der Felder $\tilde{c}(s(\mathbf{x}))$ und $\tau(s(\mathbf{x}))$ für das Abtasten entlang eines Strahls approximativ das Produkt aus der Nyquist-Frequenz des Skalarfelds $s(\mathbf{x})$ und dem Maximum der Nyquist-Frequenzen der Transferfunktionen $\tilde{c}(s)$ und $\tau(s)$ bzw. der des Produkts $c(s)\tau(s)$ ist. Deshalb ist es unter keinen Umständen ausreichend, das Volumen mit seiner Nyquist-Frequenz abzutasten, wenn nichtlineare Transferfunktionen erlaubt sind. Daraus resultierende Unterabtastungsartefakte sind in vielen Fällen zu beobachten, solange diese nicht durch sehr „weiche“ Transferfunktionen unterdrückt werden. In Abschnitt 4.3 wird zur Behebung dieser Problematik ein völlig neues Verfahren vorgestellt, das durch eine Trennung der Abtastung des Volumens und der Integration über die Transferfunktion hohe Abtastraten vermeidet.

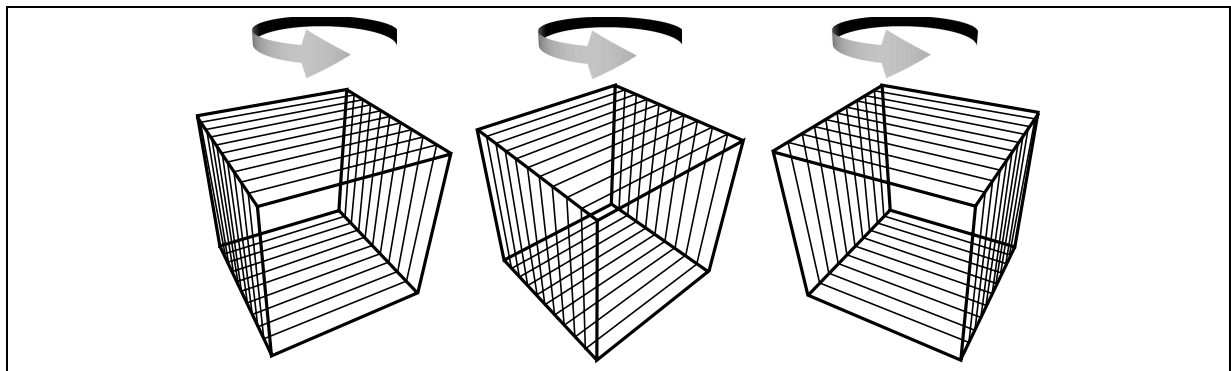


Abbildung 2.15: Objekt-parallele Schichten bei der Verwendung von 2D-Texturen. Bei der Überschreitung eines Grenzwinkels wird auf einen alternativen Schichtenstapel gewechselt.

2.3.6 Rasterisierungsflächen

Ein effizientes Verfahren zum Rendering beleuchteter Isoflächen unter Nutzung von Rasterisierungshardware wurde von Westermann und Ertl in [146] entwickelt. Im Gegensatz zu den in Abschnitt 2.3.1 vorgestellten indirekten Visualisierungsmethoden zur Isoflächenextraktion wird bei diesem Verfahren keine polygonale Zwischenrepräsentation erzeugt. Stattdessen wird ein texturbasierter Ansatz wie in Abschnitt 2.3.5 verwendet.

Der Volumendatensatz wird in eine RGBA 3D-Textur abgebildet, die in den RGB-Komponenten für jeden Voxel den zuvor berechneten Gradienten enthält. In der Alpha-Komponente werden die Intensitäten der Voxel gespeichert. Beim Rendern des Volumens in den Framebuffer kann nun der Alpha-Test von OpenGL benutzt werden, um ausschließlich solche Fragmente passieren zu lassen, die in der Alpha-Komponente den angegebenen Isowert enthalten oder darüber bzw. darunter liegen.

Zur lokalen Beleuchtungsberechnung wird der Term

$$I = I_a + I_d \cdot \langle \vec{n}, \vec{l} \rangle$$

ausgewertet, wobei \vec{l} die Licht-Richtung und \vec{n} die Normale der Isofläche ist. Zur Bestimmung des Normalenvektors kann der Volumen-Gradient verwendet werden, welcher sich z.B. mit zentralen Differenzen bestimmen lässt. I_a und I_d sind die Intensitäten der ambienten bzw. diffusen Beleuchtung.

Der Framebuffer enthält nach dem ersten Render-Schritt das Bild einer Isofläche, bei der die Voxel-Gradienten im RGB-Kanal enthalten sind. Der Framebuffer-Inhalt wird nun bei aktivierter Color-Matrix erneut in die Rasterisierungs-Pipeline eingespeist, was eine Multiplikation der RGBA-Werte im Framebuffer mit der Color-Matrix bewirkt.

Zur Berechnung des Skalarproduktes, welches für die lokale Beleuchtungsberechnung benutzt wird, erfolgt folgende Parametrisierung der OpenGL-Color-Matrix:

$$\begin{pmatrix} R \\ G \\ B \\ \alpha \end{pmatrix} = \begin{pmatrix} L_x & L_y & L_z & 0 \\ L_x & L_y & L_z & 0 \\ L_x & L_y & L_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \vec{n}_x \\ \vec{n}_y \\ \vec{n}_z \\ I \end{pmatrix} = \begin{pmatrix} \langle \vec{n}, \vec{L} \rangle \\ \langle \vec{n}, \vec{L} \rangle \\ \langle \vec{n}, \vec{L} \rangle \\ I \end{pmatrix} \quad (2.5)$$

Die obige Matrix muss mit der aktuellen Rotationsmatrix M_{rot} multipliziert werden, um die Gradienten in den Raum der Lichtquelle zu transformieren. Da die Gradienten-Komponenten im Framebuffer im Intervall $[0, 1]$ kodiert sind, muss zusätzlich eine Multiplikation mit folgender Matrix vorgenommen werden, die eine Abbildung $[0, 1] \mapsto [-1, 1]$ realisiert:

$$\begin{pmatrix} 2 & 0 & 0 & -1 \\ 0 & 2 & 0 & -1 \\ 0 & 0 & 2 & -1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (2.6)$$

Damit ergibt sich als zu verwendende Color-Matrix

$$CM = \begin{pmatrix} k_{dR} & 0 & 0 & 0 \\ 0 & k_{dG} & 0 & 0 \\ 0 & 0 & k_{dB} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} L_x & L_y & L_z & 0 \\ L_x & L_y & L_z & 0 \\ L_x & L_y & L_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} M_{\text{rot}} \begin{pmatrix} 2 & 0 & 0 & -1 \\ 0 & 2 & 0 & -1 \\ 0 & 0 & 2 & -1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

bei einer farbigen diffusen Lichtquelle mit Farbe (k_{dR}, k_{dG}, k_{dB}) . Die Addition von I_a wird durch eine *post-color-matrix-scale-and-bias*-Operation durchgeführt.

Ein wesentlicher Vorteil dieses Verfahrens gegenüber der Isoflächenextraktion ist die Möglichkeit einer interaktiven Manipulation des Isowertes, da lediglich der Referenzwert für den Alpha-Test geändert und ein erneuter Rendering-Vorgang gestartet werden muss. Bei Volumendaten entsprechender Größe ist ein interaktiver Wechsel des Isowerts bei Verfahren, die eine Flächenextraktion durchführen, wegen der enormen Zahl geometrischer Primitive nicht möglich. Allerdings sind Rasterisierungsisoflächen nach diesem Verfahren nur einseitig beleuchtet, da nur ein OpenGL Alpha-Test nutzbar ist, der ausschließlich Fragmente passieren lässt die entweder über (und gleich) oder unter (und gleich) dem Isowert sind. Bei Vergleich auf Gleichheit mit dem Isowert ergeben sich Löcher in den sich ergebenden Isoflächen. Dieser Nachteil kann zwar durch Definition eines Alpha-Test-Intervalls umgangen werden[8], allerdings ist hierfür ein Mehrstufenverfahren notwendig. Ein weiterer Nachteil des Verfahrens ist, dass die resultierenden Isoflächen nicht in anderen Programmen weiter verwendet werden können.

Optimierungen und Verbesserungen dieses Ansatzes mittels flexibler Rasterisierungshardware werden in Abschnitt 4.2.3 präsentiert, während in Abschnitt 4.3 ein völlig neuer Ansatz zur Darstellung von beleuchteten Isoflächen mit Graphikhardware vorgestellt wird, der eine Isoflächendarstellung durch die Klassifikation von Strahlsegmenten erlaubt.

Nach dieser kurzen Einführung in die Volumenvisualisierung sollen im nun folgenden Abschnitt für diese Arbeit grundlegende Internet-Techniken zur Sprache kommen.

2.4 Internet-Techniken

Dieser Abschnitt beschreibt kurz ausgewählte Internet-Techniken, d.h. Protokolle, 3D-Formate und die Programmiersprache Java. Diese werden im weiteren Verlauf dieser Arbeit für die Implementierung der verschiedenen Algorithmen und Strategien zur verteilten Visualisierung noch eine Rollen spielen.

2.4.1 TCP/IP und Sockets

Auf der untersten Ebene ist das Internet auf dem *Internet Protokoll* (IP) aufgebaut[20]. Dieses Protokoll garantiert die Zustellung eines bestimmten Paketes an den Empfänger nicht, da Datagramme nicht explizit vor Verlust oder Übertragungsfehlern geschützt werden. Im OSI-Referenzmodell ist IP auf der Ebene der Vermittlungsschicht einzuordnen. Auf der darüberliegenden Transportschicht findet sich das *Transmission Control Protokol* (TCP), das einen verbindungsorientierten Dienst erlaubt, wobei Übertragungsfehler, Verluste oder Sequenzfehler des darunter liegenden IP-Dienstes entdeckt und korrigiert werden. Dagegen handelt es sich beim

User Datagram Protocol (UDP) um einen verbindungslosen Dienst, der nur einige wenige zusätzliche Funktionen gegenüber IP aufweist. Im Wesentlichen handelt es sich dabei um eine *Port*-Nummer zur Adressierung von Anwendungsprozessen und eine optionale Prüfsumme. Die *Port*-Nummer einer TCP- oder UDP-Verbindung bildet zusammen mit der IP-Adresse einen *Socket*, über den beliebige Daten ausgetauscht werden können. Der Datenaustausch über TCP-Sockets wird vor allem in Kapitel 5 von Bedeutung sein.

2.4.2 Objektbusse

Neben der Kommunikation verschiedener Prozesse auf unterschiedlichen Rechnern über entfernte Prozeduraufrufe, sogenannten *Remote Procedure Calls* (RPCs), haben sich heute vor allem objekt-orientierte Paradigmen wie der Industriestandard CORBA (*Common Object Request Broker Architecture*) der *Object Management Group* (OMG) durchgesetzt[132]. Bei CORBA handelt es sich um eine Objektbus-Architektur, die das Aufrufen entfernter Objekt-Methoden erlaubt. Dabei können diese Methoden in unterschiedlichen Programmiersprachen implementiert sein und auf unterschiedlichen Hardwarearchitekturen ausgeführt werden. Zur Bekanntmachung der Schnittstellen entfernter Objekte wird eine Schnittstellenbeschreibungssprache (CORBA-IDL) verwendet. Aus dieser werden Proxy-Objekte für statische Objektaufrufe erzeugt, die auf Client-Seite *Stubs* und auf Server-Seite *Skeletons* genannt werden. Zusätzlich bietet CORBA mit dem *Dynamic Invocation Interface* (DII) und dem *Dynamic Skeleton Interface* (DSI) auch die Möglichkeit, Methoden von Objekten aufzurufen, deren Interfaces zur Compilierzeit noch nicht bekannt sind. CORBA wird in Kapitel 6 zur Kommunikation einer replizierten Java-Applikation mit einem Visualisierungsserver eingesetzt.

2.4.3 Unicasting, Multicasting und Streaming

Unicasting- oder Punkt-zu-Punkt-Kommunikation erlaubt den Datenaustausch zwischen zwei Kommunikationspartnern. Diese Art des Datenaustauschs spielt in dieser Arbeit die wesentliche Rolle, da im Allgemeinen von der Kommunikation von einem Client mit einem Server ausgegangen wird. Das im Kapitel 6 erläuterte Verfahren zur Fernvisualisierung erlaubt allerdings implizit die Darstellung einer Visualisierungssitzung auf beliebig vielen verbundenen Clients. Zur Entlastung des Übertragungskanal auf der Seite des Servers ist in diesem Fall ein *Multicasting*-Dienst[24] sinnvoll, der das Senden von Nachrichten an eine Gruppe von Empfängern in einer einzigen Sendeoperation zulässt. Das *Streaming* von Videodaten durch Video-Server oder im Rahmen von Multimedia-Konferenzen ist bei größerer Zahl von Nutzern nur unter Verwendung von *Multicasting*-Diensten effizient möglich.

2.4.4 VRML/X3D

Die *Virtual Reality Modeling Language* (VRML97 ISO/IEC 14772-1:1997)[1] ist ein Dateiformat für die Beschreibung und den Austausch von 3D-Szenen im Internet, das von der ISO (International Organization for Standardization) und der IEC (International Electrotechnical Commission) standardisiert wurde. Zur Beschreibung der Szene wird in VRML97 ein Szenengraph

eingesetzt, der als gerichteter, azyklischer Graph (DAG) modelliert ist. Dieser Szenengraph wird unabhängig von der spezifischen Implementierung eines Graphiksystems in einer ASCII-Syntax beschrieben. Neben der Möglichkeit, statische Objekte beschreiben zu können, erlaubt VRML97 auch ereignisbasierte Dynamik in Szenen. Ein Ereignis ist eine Nachricht, die an einem Eigenschaftsfeld eines Knoten ausgelöst und entlang festgelegter Routen an Felder anderer Knoten weitergegeben werden kann. Dort können weitere Ereignisse ausgelöst werden. Durch Annäherungssensoren, Berührungssensoren oder zyklischen Zeitsensoren ergibt sich somit die Möglichkeit einfacher Benutzerinteraktionen und Animationen. Zusätzlich ergibt sich durch Definition von Skript-Knoten eine beschränkte Programmierbarkeit.

Im Gegensatz zu anderen Szenengraphschnittstellen ist VRML ein inhaltsorientiertes Format, weshalb primär die Verbreitung von 3D-Inhalt im Internet und nicht die Programmierbarkeit im Vordergrund steht. Trotzdem kann auf den Szenengraphen mittels Skript-Knoten und dem *External Authoring Interface* (EAI ISO/IEC 14772-2, 1997)[2] zugegriffen werden. Im Rahmen des *External Authoring Interface* geschieht dies durch Empfang und Versenden von Ereignissen aus bzw. in Felder der Knoten des Szenengraphen. So kann beispielsweise ein Java-Applet, das mit dem VRML-Plugin in eine HTML-Seite eingebettet ist, direkt die VRML-Szene beobachten und dort Geometrie entfernen oder einfügen. Diese Technik wird in Kapitel 5 intensiv eingesetzt.

Das Web3D-Konsortium, ein Industrieverband der aus dem VRML-Konsortium hervorging, startete im August 2001 die nächste Generation der Web-basierten 3D-Szenenbeschreibungssprachen. Der Nachfolger von VRML97 ist X3D (Extensible 3D)[21] und vereinigt dessen Beschreibungsmöglichkeiten mit der Erweiterbarkeit der *Extensible Markup Language* (XML).

Neben diesen Standards existieren im Bereich des WWW eine ganze Reihe weiterer, proprietärer Web3D-Formate. Diese spielen in dieser Arbeit allerdings keine Rolle, da hier eine breite Verfügbarkeit der Dienste über plattformneutrale und standardisierte Formate und Schnittstellen im Vordergrund stand.

2.4.5 Java

Java[45, 6] ist eine objektorientierte, in starkem Maße an C++ angelehnte betriebssystem- und plattformunabhängige Programmiersprache, die 1995 von der Firma *Sun Microsystems* entwickelt wurde. Die Sprache ist zur Implementierung von in Netzen verteilbaren Anwendungen geeignet, bei denen Daten mit zugehöriger „Intelligenz“ in einen entfernten Rechner geladen und dort ausgeführt werden sollen. Damit wird die in der Informatik, speziell im Internet, äußerst wichtige Forderung nach Systemunabhängigkeit erfüllt. Java-Programme können in Form von Java-Applets auf beliebigen Rechnern ausgeführt werden, die über einen Java-fähigen WWW-Browser verfügen. Zusätzlich können Java Programme - wie herkömmliche Programme - auch ohne Browser ausgeführt werden, sofern ein Java-Laufzeitsystem in Form einer virtuellen Maschine vorhanden ist.

Nach einer Definition von *Sun Microsystems* ist Java eine einfache und kleine, objektorientierte, dezentrale, interpretierte, stabil laufende, architekturneutrale, portierbare und dynamische Sprache, die Hochgeschwindigkeitsanwendungen und Multithreading unterstützt. Java hat sich heute als de-facto Standard für die Programmierung von Applikationen im Internet durchgesetzt.

Ein Großteil der in dieser Arbeit realisierten Algorithmen wurden mit Java implementiert.

2.4.6 Java3D

Java3D[135] ist eine Erweiterung von Java zur Entwicklung von 3D-Graphik-Anwendungen und -Applets. Im Gegensatz zu VRML definiert Java3D kein Format für den Austausch von Szenen im Internet, vielmehr steht die Programmierung von Applikationen zum Darstellen von dreidimensionalen Szenen im Vordergrund. Java3D realisiert einen Szenengraphen basierend auf den *low-level* Schnittstellen OpenGL bzw. DirectX/Direct3D. Der Szenengraph kann über das API komfortabel programmiert werden, allerdings bietet Java3D keinen direkten Zugriff auf spezielle Graphikhardware-Erweiterungen bestimmter Hersteller.

2.4.7 Java OpenGL Anbindungen

Die direkte Anbindungen von OpenGL and Java (OpenGL Bindings) bieten im Gegensatz zu VRML und Java3D vollständige Kontrolle über den Rendering-Prozess. Erreicht wird dies durch die Kapselung aller OpenGL Funktionsaufrufe in Java Methodenaufrufe mittels des *Java Native Interface* (JNI). Es existieren eine Reihe von Anbindungen, beispielsweise Jogl[119], Magician[25] oder GL4Java[43]. Allerdings ist keine dieser Anbindungen bisher standardisiert worden. Die Hauptproblematik bei der Nutzung von OpenGL-Anbindungen stellt die Weitergabe großer Datenmenge von Java an OpenGL dar, da dies bisher mit einem Kopieren der Daten verbunden war. Diese Problematik wird sich im Java Developer Kit (JDK) 1.4 durch die Einführung des Konzepts der direkten Puffer im neuen Paket `java.nio` lösen. Durch den direkten Zugriff auf OpenGL sind Java OpenGL-Anbindungen sicher die flexibelste Lösung zur architekturneutralen Programmierung von hardwarebeschleunigter 3D-Graphik.

Die in diesem Abschnitt vorgestellten Techniken bilden die Grundlage der wissenschaftlichen Visualisierung im Internet. Der folgende Abschnitt beschreibt wesentliche Vorarbeiten in diesem Bereich.

2.5 Wissenschaftliche Visualisierung im Internet

Durch den raschen Erfolg des World Wide Web zu Beginn der 1990er Jahre wurde es möglich, wissenschaftliche Visualisierungen in digitale Dokumente einzubetten und damit einer breiten Öffentlichkeit zugänglich zu machen. Allerdings war zu dieser Zeit die wissenschaftliche Visualisierung im WWW eine rein passive Angelegenheit. Ein Wissenschaftler erstellte eine Visualisierung mit einem konventionellen Visualisierungssystem und publizierte diese auf einer Web-Seite, so dass ein Leser diese später betrachten konnte. Das Web war also ein reines Publikationsmedium, das keinerlei Interaktion mit den Darstellungen erlaubte. Heute dagegen stellt das Web eine große verteilte Rechnerumgebung dar, in der Visualisierungen nun unter aktiver Teilnahme des Publikums *live* erstellt werden können. Daraus ist eine neue Disziplin entstanden, das Feld der Web-basierten Visualisierung.

2.5.1 Anfänge

Pionierarbeit in diesem Feld wurde 1994 von Ang et al.[4] geleistet, die das *Multipurpose Internet Mail Extensions* (MIME) Konzept nutzten, um Visualisierungsdaten über das Web zur Verarbeitung in einem Web-Browser zu verschicken. Dazu verknüpften sie das von ihnen entwickelte medizinische Visualisierungssystem VIS mit dem Mosaic Browser über dessen Schnittstelle *Common Client Interface* (CCI). Sobald der Browser Daten mit dem MIME-Typ `hdf/volume` (NCSA Hierarchical Data Format - HDF) empfängt, gibt er diese an das VIS-System zu Verarbeitung weiter.

Mit dem Aufkommen von Java wurde ein anderer Ansatz möglich. Nun wurden sowohl Daten als auch Software über das Web transferiert. Ein frühes Beispiel dafür ist der *Visible Human Viewer* des Northeast Parallel Architectures Center (NPAC), der im Jahre 1995 vorgestellt wurde und auch heute noch breite Nutzung findet[111].

Wood et al.[154] setzten auf einen anderen Ansatz. Anstelle des Versendens von Visualisierungsdaten oder von Software über das Web, schlugen sie ein System vor, das die Visualisierung auf einem entfernten Rechner vornahm und dann die resultierenden Graphiken über das Web zu einem Web-Browser übertrug. Sie entwickelten einen Visualisierungsservice zur Überwachung der Luftqualität, bei dem ein Benutzer Daten und Visualisierungsparameter in ein Web-Formular einträgt. Das Formular wird mit einem *Common Gateway Interface* (CGI) Script auf der Seite des Servers ausgewertet, welches dann das Visualisierungssystem *IRIS Explorer* zur Visualisierung aufruft. Das Ergebnis wird als eine VRML-Szene zum Web-Browser des Clients zurückgesandt.

Diese Beispiele legen bereits eine mögliche Klassifizierung der verschiedenen Ansätze entsprechend der transferierten Daten nahe. Es werden also entweder Visualisierungsdaten, Software oder Graphiken übertragen.

2.5.2 Visualisierungsdatentransfer

Dieser Ansatz benutzt das Web eigentlich nur als Kommunikationsmedium für wissenschaftliche Datensätze. Diese Idee wurde auch von Hibbard im *Vis5D* System genutzt[62], das meteorologische Daten mit dem MIME-Typ `application/vis5d` versieht. Der Browser wird so konfiguriert, dass er die Daten an *Vis5D* weiterleitet. Ein Benutzer kann dort alle Parameter zur Generierung von Visualisierungen beeinflussen.

Dieser Ansatz wurde im bereits genannten System von Wood et al. erweitert[154], das auf dem *Modular Visualization Environment* (MVE) *IRIS Explorer* basiert. MVEs sind programmierbare Systeme, in denen der Benutzer Module in einer Datenfluss-Pipeline verbindet. Dies kann entweder in einem visuellen Editor oder mit einer Skripting-Sprache geschehen. Wood et al. verwenden nun *IRIS Explorer*, um ein System zu realisieren, das nicht allein die Visualisierungsdaten über das Netz an einen Client schickt, sondern zusätzlich die Spezifikation der Datenfluss-Pipeline. Auf der Client-Seite wird *IRIS Explorer* gestartet, die Pipeline entsprechend konfiguriert und der Datensatz geladen. Interessante Möglichkeiten eröffnen sich durch den Transfer weiterer Kommandofolgen, mit denen beispielsweise nacheinander verschiedene Visualisierungsparameter der einzelnen Module geändert werden können. Für einen weiteren Nutzer im Web entsteht so eine Art Präsentation. Dies kann beispielsweise zu Konsultations-

zwecken genutzt werden. Der Ansatz ähnelt dem *FASTexpeditions* System[18] des *NASA Ames Research Center*.

Ein System, das komprimierte Volumendaten progressiv zu einem Client transferiert, wurde in [47] vorgestellt. Die Daten werden auf dem Client direkt, ohne Dekomprimierung, von einem Java-Applet in der Wavelet-Basis gerendert. Da das Rendering im Frequenzraum durchgeführt wird, sind allerdings nur Röntgenbild-ähnliche Darstellungen möglich.

2.5.3 Softwaretransfer

Die Ursprünge des Transfers von Software können zu frühen Beispielen wie dem NPAC *Visible Human Viewer*[111] zurückverfolgt werden. Die Daten sind in diesem Fall eng mit dem Algorithmus verknüpft, mit dem sie dargestellt werden können. Daten und Software sind auf dem gleichen Server vorhanden. In vielen Fällen ist es jedoch notwendig, dass der Benutzer selbst Daten in das System importieren kann. Eines der ersten Applets, das Benutzer-Daten importieren konnte, war *VizWiz*[14]. Dieses Applet erlaubt die Visualisierung von Isoflächen. Um die Sicherheitsmechanismen von Java zu umgehen, die es einem Applet nur erlaubt Daten von dem Server zu laden von dem es stammt, werden die Daten zunächst mit dem Browser in ein temporäres Verzeichnis des Servers geladen und dann wieder zurück transferiert. Natürlich kann dieser zweifache Transfer der Daten für große Datensätze Probleme bereiten.

Diese Konzept wurde von Kee[76] in der Hinsicht erweitert, dass das von ihm entwickelte System das Laden von Daten von jeder beliebigen Stelle aus dem Web ermöglicht. Dies wird dadurch erreicht, dass auf dem Server eine Applikation läuft, die Daten von einer gegebenen URL zunächst auf den Server lädt, um diese dann an einen Client weitergeben zu können. Dabei ist wieder ein mehrfacher Datentransfer nötig. Durch die Möglichkeit signierter Java Applets, lokaler Dateisysteme lesen zu können, ist inzwischen dieser mehrfache Datentransfer nicht mehr nötig.

Die Ausnutzung Client-seitiger Rechen- und Graphikressourcen wurde von Bender et al.[10, 9] demonstriert. Das dort vorgestellte System erlaubt interaktive Molekülvisualisierung in einem Web-Browser, wobei alle notwendigen Berechnungen von einem Java-Applet durchgeführt werden und das Rendering unter Ausnutzung lokaler Graphikhardware in einem VRML-Plugin vorgenommen wird. Ein weiteres Beispiel für eine Visualisierungsapplikation, die ausschließlich Client-seitige Rechen- und Graphikfähigkeiten ausnutzt, wurde von Hendin, John und Shochet[59] vorgestellt. In diesem System werden die 2D-Textur-Fähigkeiten eines VRML-Plugins zur direkten Volumenvisualisierung mit drei orthogonalen Schichtenstapeln ausgenutzt. Ein Java-Applet lädt medizinische Daten von einem Server und gibt diese in Form von 2D-Texturen an das VRML-Plugin weiter. Erweiterte Interaktionsmöglichkeiten mit einer VRML-Szene durch eine Anbindung eines Java-Applets mit dem EAI wurden auch für Chirurgie-Trainingssysteme in [30, 75] vorgeschlagen. Das *VisAD* System schließlich erlaubt die Entwicklung von Web-basierten Visualisierungssystemen aus Java-Komponenten[61, 60].

2.5.4 Graphiktransfer

Ursprung der auf dem Transfer von Graphiken beruhenden Ansätze ist ein von Wood et al. realisiertes System[154]. Da der hier gewählte Ansatz sehr generell ist, wurden eine Reihe ähnlicher Systeme vorgeschlagen. Beispielsweise beschreiben die Autoren des *Visualization Toolkits* (VTK) [125], wie ein VTK-basierter Visualisierungsservice auf ähnlicher Basis realisiert werden kann.

Die auf Web-Formularen basierende Eingabe der Daten ist zwar sehr einfach zu realisieren, aber auch recht unflexibel. Trapp und Pagendam[74] ersetzen in ihrem Vis-a-Web System das Web-Formular durch ein flexibleres Java-Applet. Der Service liefert eine VRML-Szene der eingereichten Daten zurück. Eine Erweiterung dieses Ansatzes stellt die Aufspaltung eines MVEs in zwei Teile dar: Eine Java-Benutzeroberfläche auf der Client-Seite, die automatisch von der Pipeline einer MVE-Applikation erzeugt wird, die auf der Server-Seite läuft. Dieser Ansatz wurde von Wood et al. auf der Basis von *IRIS Explorer*[154] und von Treinish auf der Basis von *IBM Data Explorer*[141] realisiert.

Als Alternative zum Transfer von 3D-Szenen ist es auch möglich, die 3D-nach-2D Projektion auf einem Server durchzuführen und die sich ergebenden Bilder zu einem Client zu übertragen. Im System *DiVision* von Ma und Patten[110] werden Folgen von Bildern an den Client übertragen, wo sie als Knoten eines Graphen abgelegt werden. Die Kanten des Graphen stehen dabei für verschiedene Interaktionen, wodurch semantische Zusammenhänge erkennbar gemacht werden.

Ähnlich verhält es sich mit dem *WAVES* (Web-Accessed Visualization for Engineering and Sciences) System[99], das einen entfernten *GNUplot* Service realisiert. Ein Benutzer gibt Daten mit einem Web-Formular ein und das System liefert ein GIF-Bild zurück. Ein weiteres Beispiel ist das *CurVis* System von Theisel et al.[138], das im Rahmen des DFG-Projekts *Mobile Visualization* (MoVi) entwickelt wurde. Das System produziert Server-seitig Videosequenzen von 2D-Vektorfeldern, deren Visualisierungsparameter auf dem Client über ein Web-Formular beeinflusst werden können. Dabei wird besonderer Wert auf die effiziente Komprimierung der Videodaten gelegt.

Alle in diesem Abschnitt erläuterten Systeme erlauben aber keine wirklich interaktive Darstellung, da bei Änderung von Visualisierungsparametern keine unmittelbare Reaktion auf die geänderten Parameter erfolgt. Ein Ziel des in dieser Arbeit verfolgten Fernvisualisierungsdienstes (siehe Kapitel 6) ist es, den Visualisierungsdienst wie eine lokale Applikation bedienbar zu machen. Inzwischen existieren weitere Systeme, die sich das Ziel einer interaktiven Bedienung einer entfernten Applikation gesetzt haben. Das *Virtual Network Computing* (VNC) System von Richardson et al.[121] und das *OpenGL Vizserver* Produkt von SGI [103] werden in Kapitel 6 genauer beleuchtet.

Nach dieser Einführung in einige für diese Arbeit grundlegenden Themen, wird im folgenden Abschnitt nun der Begriff der Client-Server-Strategien eingeführt. Im Gegensatz zu der in diesem Abschnitt verwendeten Unterscheidung der Ansätze anhand der transferierten Daten steht bei den Client-Server-Strategien die Aufteilung der Teilaufgaben der Visualisierungspipeline auf die beteiligten Rechner im Vordergrund.

However beautiful the strategy,
you should occasionally look at the results.

Sir Winston Churchill, 1874-1965

Kapitel 3

Client-Server-Strategien

Die im vorausgehenden Kapitel erläuterte Visualisierungspipeline beschreibt die Reihenfolge der Verarbeitungsstufen, welche die Ausgangsdaten bis zur graphischen Repräsentation durchlaufen. Dabei wurde bisher angenommen, dass die Verarbeitung der Daten auf einem einzigen System durchgeführt wird. In einem Client-Server-System ist eine Aufteilung der Module der Visualisierungspipeline notwendig. Diese Visualisierungspipeline und die sich durch die Aufteilung der Module der Pipeline ergebenden Client-Server-Strategien[37] sollen nun im Vordergrund stehen.

3.1 Client-Server-Visualisierungspipeline

Der fundamentale Unterschied zu der traditionellen Visualisierungspipeline aus Abschnitt 2.1.1 ist in einem Client-Server-Umfeld durch die Notwendigkeit einer Übertragung von Daten zwischen dem Client und dem Server gegeben (siehe Abbildung 3.1), d.h. die Visualisierungspipeline muss an einer bestimmten Stelle aufgetrennt und die Module entsprechend aufgeteilt werden. Dabei ist auch eine Aufspaltung einzelner Module in verschiedene Submodule denkbar, wie dies später in Abschnitt 5.3.1 noch genauer zur Sprache kommen wird.

Die Wahl der Aufteilung dieser Aufgaben für Client und Server wird im Folgenden als Client-Server-Strategie bezeichnet. Die Wahl einer geeigneten Strategie hängt dabei von einer Reihe von Faktoren ab: Art und Größe der Daten, Bandbreite und Latenz des Netzwerks, graphische und numerische Fähigkeiten der vorhandenen Client- und Server-Rechner. Grundsätzlich können drei verschiedene Strategien unterschieden werden, die in den nun folgenden Abschnitten erläutert werden.

3.2 Client-seitige Strategien

Durch die enorm gestiegenen Rechen-, Speicher- und Graphikkapazitäten heutiger Arbeitsplatzrechner sind in den letzten Jahren Client-seitige Ansätze (Bild 3.2) stark in den Vordergrund getreten. Bei Client-seitigen Ansätzen werden die zu visualisierenden Daten vollständig auf den

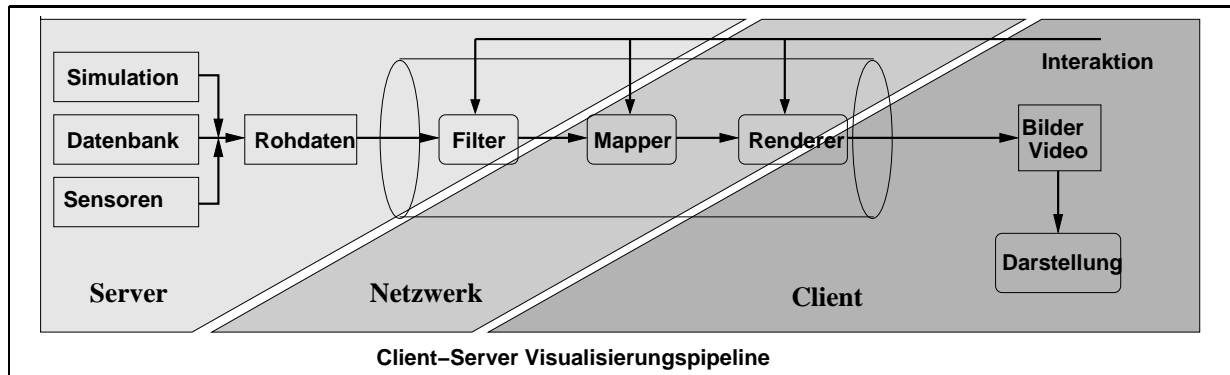


Abbildung 3.1: In der Client-Server-Visualisierungspipeline ist ein Datentransfer an einer bestimmten Stelle innerhalb der Pipeline notwendig.

Client übertragen, der dann alle weiteren Berechnungen der verbleibenden Module der Visualisierungspipeline ausgeführt. Da während der Interaktion keine weitere Datenübertragung notwendig ist, also keine weiteren Verzögerungen durch einen Datentransfer auftreten, können bei entsprechender Leistungsfähigkeit des Clients hohe Interaktionsraten erzielt werden.

Für den Web-Service *OrbVis* (siehe Abschnitt 5.1.4) wurde dieser Ansatz gewählt, da die Zahl der Dreiecke bei Orbitalvisualisierungen generell in einer Größenordnung bleibt, bei der jeder Standard-PC die volle Szene in ihrer maximalen Auflösung bewältigen kann. Außerdem ist der Transfer der Volumendaten, die zur Erzeugung der Orbitaloberflächen benötigt werden, hierbei nicht notwendig, da ein vom Server zum Client übertragener Algorithmus das Volumen lokal rekonstruiert.

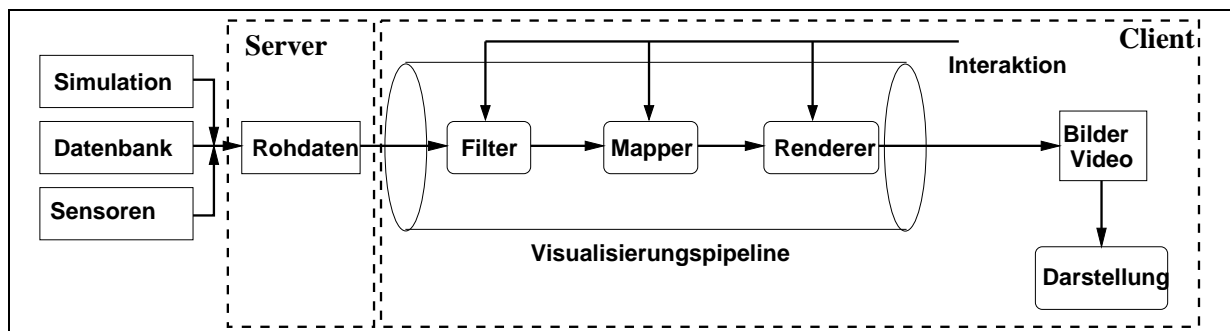


Abbildung 3.2: Client-seitige Strategie.

OrbVis ist ein Online-Service zur Berechnung und Visualisierung von Molekülorbitalen (MOs)[107]. MOs beschreiben die Aufenthaltswahrscheinlichkeit von Elektronen im Raum und sind unter anderem wichtig zum Verständnis chemischer Reaktionen. *OrbVis* ist in der Lage, MOs von Molekülen, die in Form von Konnektivitätslisten an den Server geschickt werden, zu

berechnen und zu visualisieren. Die Berechnung der 3D-Koordinaten und der Orbitalvektoren wird von kommerziell verfügbaren Programmen zur chemischen Informationsverarbeitung auf dem Server bewerkstelligt. Das Ergebnis der Berechnungen wird in stark komprimierter Form als Funktionsparameter an den Client übermittelt. Der Client startet daraufhin ein Java-Applet, das Funktionen zur Berechnung des Elektronendichtevolumens aus den Funktionsparametern enthält. Aus diesem berechneten Volumen wird nun eine Isofläche der Elektronendichte zu einem gegebenen Isowert generiert und mittels eines VRML-Plugins dargestellt. Der Vorteil dieser Methode ist, dass anstelle großer Volumendatensätze nur eine HTML-Seite mit Funktionsparametern an den Client gesandt werden muss. Diese Vorgehensweise erlaubt eine interaktive Analyse der Strukturen und der Volumendaten bei gleichzeitiger Balancierung von Rendering-Qualität und Echtzeit-Performance[107]. Desweiteren wurde eine Prototyp-Applikation zur Animation von Volumendaten, wie z.B. Molekülorbitalen, während einer chemischen Reaktion entwickelt.

Wegen der fehlenden Möglichkeiten der schnellen trilinearen Interpolation, die zur Erzeugung qualitativ hochwertiger Visualisierungen von Volumendaten notwendig ist, war eine interaktive direkte Volumenvisualisierung bisher auf Hochleistungsgraphikrechner und Spezialhardware beschränkt. Moderne Low-Cost-Graphikkarten, die vor allem für Spiele- und Multimediaanwendungen konzipiert wurden, verfügen aber zunehmend über leistungsfähige mehrstufige Textur-Rasterisierungseinheiten, die eine Vielzahl von Optimierungen der direkten Volumenvisualisierung zulassen. Da diese Karten aber im allgemeinen noch nicht über 3D-Texturen verfügen, können hier auch 2D-Texturen zum Einsatz kommen, die heute auf jeder dieser Karten unterstützt werden. Auf der Basis des in Abschnitt 2.3.6 beschriebenen Verfahrens zur interaktiven Visualisierung von beleuchteten Isoflächen wurde ein optimiertes Verfahren mit Hilfe von Mehrstufenrasterisierung auf Standard-PC-Graphikhardware entwickelt[120]. Das ursprüngliche Zwei-Schritt-Verfahren kann mittels Mehrstufenrasterisierung in nur einem Rendering-Durchgang durchgeführt werden. Die Beleuchtungsberechnung wird dabei direkt von der Rasterisierungshardware realisiert. Die Bildwiederholungsraten liegen durch dieses Ein-Schritt-Verfahren teilweise deutlich über den von Hochleistungsworkstations erzielten Raten.

Mit Hilfe dieser Techniken kann nun ein Client-seitiger Ansatz zur Visualisierung hochauflösender Datensätze in digitalen Dokumenten konzipiert werden. Dazu wird der Server nur mehr für Anfragen nach in der Datenbank der gespeicherten Volumendatensätzen benötigt. Ein solcher Datensatz wird dann komprimiert zu dem jeweiligen Arbeitsplatzrechner übertragen. Dort wird die lokal vorhandene leistungsfähige Rasterisierungshardware verwendet, um interaktive Bildwiederholraten zu erreichen (siehe Kapitel 4). Dies wird durch die Ausnutzung der Mehrstufenrasterisierung für trilineare Interpolation und Beleuchtungsberechnung erreicht. Vorteil dabei ist natürlich auch, dass keine Verzögerungen durch das Übertragen von Daten während der Darstellung auftreten, da nach der anfänglichen Datenübertragung kein weiterer Netztransfer notwendig ist.

3.3 Server-seitige Strategien

Server-seitige Ansätze (Bild 3.3) verlagern die Module der Visualisierungspipeline auf einen oder mehrere Hochleistungsrechner, welche die zur Visualisierung notwendigen Berechnungen

unter Ausnutzung leistungsfähiger Spezialhardware durchführen. Die von den Servern berechneten Bilder werden zu Arbeitsplatzrechnern übertragen, die als einfache Anzeigeräte fungieren. Durch Rückübertragung der Benutzeroberflächen-Ereignisse (hauptsächlich Maus- und Tastatureingaben) der Clients kann die Visualisierung beeinflusst werden.

Server-seitige Strategien werden vor allem dann eingesetzt, wenn der Client nicht die zur Visualisierung der Daten notwendige Ausstattung besitzt, ein Transfer der Visualisierungsdaten wegen ihrer Größe unmöglich ist oder der Transfer der Daten, beispielsweise aus Datenschutzgründen, unerwünscht ist.

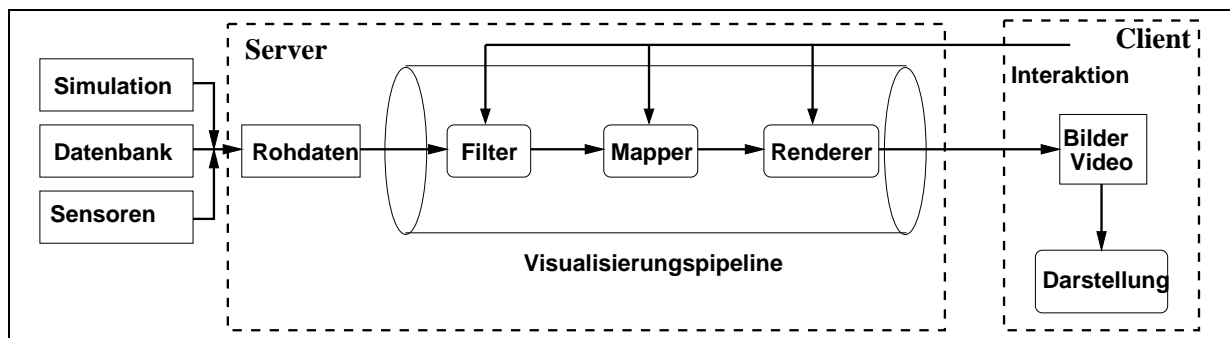


Abbildung 3.3: Server-seitige Strategie.

Im Abschnitt 6 wird ein Framework vorgestellt, das die Entwicklung von Applikationen unterstützt, die auf solchen Strategien basieren[39]. Das Framework ermöglicht es, bestehende Visualisierungsapplikationen basierend auf den Szenegraphenarchitekturen OpenInventor oder Cosmo3D in Visualisierungsserver-Applikationen zu konvertieren. Dazu werden in den jeweiligen Szenegraphen neue Knoten eingefügt, die die Komprimierung und Übertragung der auf dem Server generierten Darstellungen zu den verbundenen Clients ermöglichen.

Die Bilder werden von Java-basierten Clients entgegengenommen, dekomprimiert und dargestellt. Ereignisse der Benutzeroberfläche werden über CORBA-Aufrufe zurück zum Visualisierungsserver geleitet und dort wie lokale Ereignisse behandelt. Diese Architektur des Frameworks erlaubt es, von beliebigen Endnutzer-Geräten auf die numerischen und graphischen Fähigkeiten von Hochleistungsgraphikrechnern zuzugreifen. Dazu muss der Client lediglich über 2D-Darstellungsmöglichkeiten verfügen. Dies erlaubt den Einsatz einfacher mobiler Anzeigeräte wie Taschen-PCs, Web-Pads oder Mobiltelefonen, die im Allgemeinen nicht über die nötigen Speicher- und Verarbeitungskapazitäten verfügen, um die Originaldaten selbst zu verarbeiten.

3.4 Hybride Strategien

Das Ziel hybrider Ansätze (Bild 3.4) ist eine möglichst günstige Verteilung der Visualisierungsaufgaben zwischen Client- und Server-Rechner. Durch diese Aufteilung der Aufgaben sollen sowohl Server- als auch Client-seitige Rechen- und Speicherkapazitäten optimal genutzt werden,

wobei auch eine Netzwerklastminimierung angestrebt wird.

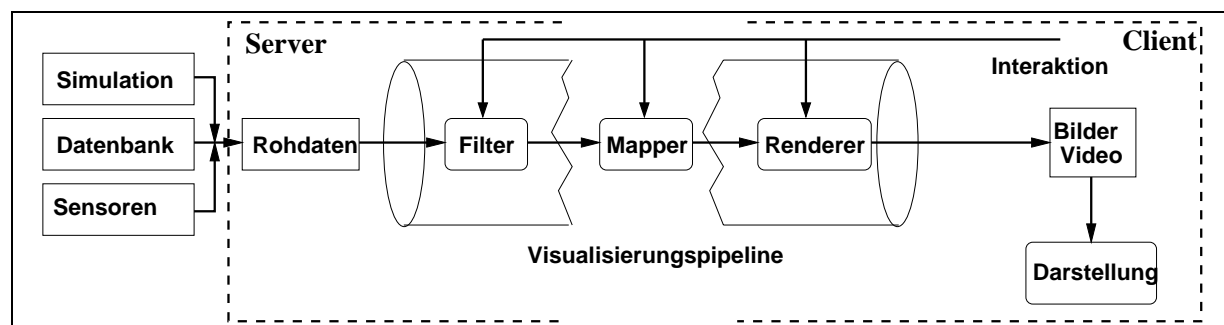


Abbildung 3.4: Hybride Strategie.

Solche hybriden Ansätze wurden im Zusammenhang mit der Extraktion von Isoflächen aus Volumendaten untersucht. Dazu werden auf der Server-Seite mittels eines optimierten Marching-Cubes-Algorithmus[34] geometrische Primitive aus den Volumendaten extrahiert und zu Clients übertragen, die mit lokaler 3D-Hardware die Darstellung übernehmen. Die Filter- und Mapper-Module der Visualisierungspipeline verbleiben also auf dem Server, während das Render-Modul auf den Client verlagert wird. Vorteile dieser Strategie sind, dass die meiste sehr großen Volumendaten auf dem Server verbleiben können und sowohl numerische Kapazitäten des Servers als auch die 3D-Hardware des Clients genutzt werden können. Durch die Darstellung der transferierten Dreiecksgitter mittels lokaler 3D-Graphikhardware sind hohe Interaktionsraten möglich, da zur Exploration der Netze kein weiterer Datentransfer notwendig ist.

Allerdings erfordert dies Techniken, die einerseits keine zu hohe Netzlast erzeugen und andererseits die Graphikhardware des Clients nicht überlasten. Es ist dazu notwendig, die enorme Anzahl der geometrischen Primitive, die bei einer Oberflächenrekonstruktion anfallen, zu begrenzen. Die progressive Übertragung von Isoflächen[34] gibt dem Benutzer zur Begrenzung der Zahl der geometrischen Primitive eine globale Level-of-Detail (LOD) Kontrolle an die Hand. Mittels eines Multi-Resolution-Ansatzes, basierend auf einer fehlergesteuerten Unterteilung des Volumens in Tetraeder und Oktaeder, werden hier Isoflächen mit unterschiedlichem Detaillierungsgrad rekonstruiert (siehe Abschnitt 5.2.3). Der Benutzer legt zunächst die Detailstufe der Isofläche fest, die dann auf Server-Seite erzeugt und zum Client transferiert wird. Dort kann die Fläche mittels 3D-Graphikhardware untersucht werden. Nachteil dieses Verfahrens ist allerdings, dass die Fläche immer als Ganzes verfeinert wird. Sucht ein Nutzer anstelle der globalen Form der Isofläche nach bestimmten Details, ist eine Fokus-basierte Isoflächenübertragung[40] besser geeignet. Der Fokus kann im Volumendatensatz auf dem Server frei platziert werden und durch die Definition eines Abstandes um diesen Punkt ein sphärischer Bereich definiert werden. Um diesen Fokus wird nun mittels eines Octree-Ansatzes eine Isofläche mit hoher Auflösung erzeugt, deren Detaillierungsgrad mit steigendem Abstand abnimmt (siehe Abschnitt 5.3). Die so erzeugten geometrischen Primitive werden zum Client transferiert und dort von einem Java-Applet entgegengenommen. Dieses nutzt ein VRML-Plugin für die im Allgemeinen Hardware-

beschleunigte Darstellung. Der Fokuspunkt kann frei auf der Isofläche oder im Raum platziert werden. Der Radius der höchsten Auflösung lässt sich im Benutzerinterface des Java-Clients manipulieren und wird in der lokalen Darstellung durch eine transparente Kugel visualisiert. Diese Vorgehensweise ähnelt einer Suche mit einer Lupe, wobei anstelle einer Vergrößerung eine verfeinerte Darstellung erzeugt wird. In [40] wurde zudem gezeigt, wie durch eine feinkörnigere Unterteilung des Marching-Cubes-Algorithmus in Module einer Pipeline und deren Verteilung auf Client- und Server-Rechner eine effiziente Lastverteilung und Datenreduktion erreicht werden kann.

3.5 Zusammenfassung

Der Zugriff auf digitale Dokumente ist heute über eine Vielzahl unterschiedlicher Rechnerplattformen und Netzwerkinfrastrukturen möglich. Um jeder dieser möglichen Zugriffskonfigurationen eine optimale Interaktion mit den eingebetteten Daten zu ermöglichen, ist eine adaptive Anpassung der Client-Server-Strategie an die jeweils gegebenen Verhältnisse denkbar. So würde entweder einmal zu Beginn oder ständig während der Sitzung die Kapazität auf Client- und Server-Seite sowie die Bandbreite und Latenz des verbindenden Netzwerks überprüft werden, um daraus die jeweilige optimale Client-Server-Strategie zu ermitteln.

Zu Beginn einer Sitzung können beispielsweise die Graphikfähigkeiten eines Client-Rechners ermittelt und bei Vorhandensein der entsprechenden Hardware kann eine Client-seitige Strategie gewählt werden. Andernfalls wird je nach Auslastung des Servers eine Server-seitige oder hybride Strategie gewählt. Die Strategie kann natürlich auch ständig adaptiert werden. Sind zu Beginn einer Sitzung genügend Server-Kapazitäten vorhanden, so könnte zunächst eine Server-seitige Strategie zur Anwendung kommen. Steigt dann während der Sitzung die Last des Servers und damit die Antwortzeit, so könnte dynamisch eine andere Strategie gewählt werden.

Der in diesem Abschnitt definierte Begriff der Client-Server-Strategien gibt uns ein Werkzeug zur Klassifikation der in den folgenden Abschnitten erläuterten Verfahren in die Hand. Im folgenden Kapitel sollen zunächst Strategien im Vordergrund stehen, die eine optimale Nutzung der Fähigkeiten moderner Arbeitsplatzrechner zum Ziel haben.

Der Irrtum ist viel leichter zu erkennen,
als die Wahrheit zu finden;
jener liegt auf der Oberfläche,
damit läßt sich wohl fertig werden;
diese ruht in der Tiefe,
danach zu forschen ist nicht jedermannes Sache.

Johann Wolfgang von Goethe, 1749-1832

Kapitel 4

Volumenvisualisierung mit flexibler Rasterisierungshardware

Eine der wesentlichen Fragen, die sich zu Beginn dieser Arbeit stellte war, ob und inwieweit sich moderne Arbeitsplatzrechner zur interaktiven Visualisierung skalarer Volumendatensätze eignen. Diese Frage resultierte nicht zuletzt aus dem enormen Leistungszuwachs, der in den letzten Jahren in diesem Bereich stattgefunden hat. Sowohl hinsichtlich Speicherkapazität als auch bezüglich der Verarbeitungsgeschwindigkeit haben heutige Client-Rechner zu professionellen Workstations aufgeschlossen und mittlerweile diese sogar teilweise überholt. Einer der wenigen verbleibenden Vorteile professioneller Graphikhardware ist in dem Vorhandensein spezieller I/O-Architekturen zu sehen, die mittels optimierter Datenpfade hohe Busbandbreiten und Datentransferraten erreichen. Hinzu kommt die erzielbare Darstellungsqualität als ein weiterer wesentlicher Aspekt. Diese spielt im Konsumentenbereich häufig eine untergeordnete Rolle, für wissenschaftliche Visualisierungszwecke ist sie allerdings von entscheidender Bedeutung. Professionelle Graphikhardware bietet hier oftmals höhere Genauigkeit durch breitere Datenpfade und größere Puffertiefen. Allerdings ist auch hier eine Annäherung von PC-Graphikhardware und professioneller Graphikhardware zu beobachten, die im weiteren Verlauf dieses Kapitels noch zur Sprache kommen wird.

Texturbasierte Ansätze zum Volume-Rendering, die in diesem Kapitel vorrangig behandelt werden, erfordern es, weitere Module der Visualisierungspipeline auf den Client zu verlagern, sobald lokale Rasterisierungshardware zum Rendering genutzt werden soll. Dies ergibt sich im Grunde daraus, dass der eigentliche Mapping-Schritt der Visualisierungspipeline bei texturbasierten Verfahren der Definition von Texturen und dem Anwenden einer Transferfunktion entspricht. Die Definition von Texturen bedeutet bei moderner Graphikhardware aber, dass die Texturdaten in den lokalen Speicher der Graphikhardware geladen werden. Das Anwenden der Transferfunktion wird durch entsprechende Funktionalität der Graphikhardware bewerkstelligt. Beide Schritte sind somit ebenfalls auf der Client-Seite durchzuführen. Einzig der Filter-Schritt

kann auf der Server-Seite verbleiben, allerdings wird eine räumliche Filterung der Daten häufig mit Clipping-Ebenen erzielt, was wieder auf der lokalen Graphikhardware erfolgt.

In diesem Kapitel kommen deshalb im Wesentlichen Client-seitige Ansätze zur Volumenvisualisierung zur Sprache. Dabei spielt die optimale Ausnutzung lokaler Graphikhardware eine wichtige Rolle. Dazu ist es oft notwendig, die Graphikhardware auf sehr niedriger Abstraktionsebene anzusprechen. Dies steht auf den ersten Blick im Widerspruch zu anderen Zielen dieser Arbeit, wie Architekturneutralität und Nutzung von Internet-Standards. Es werden deshalb in diesem Kapitel auch Techniken vorgestellt, die es erlauben, lokale Graphikhardware unter Nutzung von Internet-Standards optimal zu nutzen und dabei die Plattformunabhängigkeit zu wahren. Sie werden im nun folgenden Abschnitt behandelt, bevor fortgeschrittene Ansätze in den zwei darauf folgenden Abschnitten zur Sprache kommen.

4.1 Architekturneutrale Texturbasierte Visualisierung

Durch die Einbettung von Java Programmen in HTML-Dokumente können Programme auf einem Arbeitsplatzrechner ausgeführt werden, welche die Darstellung von verlinkten Daten erst zur Laufzeit berechnen. Die eigentlichen Daten werden also mit einem Programm, das die Visualisierung der Daten übernimmt, in eine HTML-Seite eingebettet. Dadurch eröffnet sich die Möglichkeit auf Darstellungsparameter Einfluss zu nehmen und so sehr viele unterschiedliche Visualisierungen zu erzeugen. Dies ermöglicht eine Reihe von Anwendungen, angefangen von jederzeit abrufbaren Patientenakten mit dynamischen Visualisierungen der dazugehörigen Patientendaten (Abbildung 4.1) im Intranet eines Krankenhauses bis hin zu wissenschaftlichen Publikationen oder Lehrmitteln mit eingebetteten Visualisierungen komplexer Daten.

Natürlich können auch skalare Volumendaten mit geeigneten Visualisierungsalgorithmen in digitale Dokumente eingebracht werden. Zur interaktiven Visualisierung dieser Daten ist aber, wie bereits betont wurde, dringend Unterstützung durch geeignete 3D-Graphikhardware nötig. Es stellt sich deshalb die Frage, welche plattformunabhängigen Graphikschnittstellen sich zur Visualisierung eignen und wie sie eingesetzt werden können, um hohe Bildwiederholraten zu erreichen. Im Folgenden werden Implementierungen von Volume-Rendering-Algorithmen mit den Graphikschnittstellen Java3D, der Virtual Reality Modeling Language (VRML) und Java OpenGL-Anbindungen untersucht.

4.1.1 Implementierung

Da zum Zeitpunkt der hier vorgestellten Implementierungen trilinear interpolierte Texturen sowohl von der Graphikhardware als auch teilweise von den genutzten Graphikschnittstellen nicht unterstützt wurden, kamen 2D-Texturen mit bilinearer Interpolation zum Einsatz. Anstelle der Ausrichtung der Schnittpolygone an der Bildebene werden diese am Objekt selbst ausgerichtet (siehe Abbildung 2.15). Sobald eine Abweichung der Blickrichtung von der Schichtnormale um mehr als 45 Grad eintritt, wird auf einen alternativen Schichtenstapel umgeschaltet. Demzufolge muss das Volumen dreifach im Speicher gehalten werden, jeweils ein Stapel für jede Raumachse. Der Abtastfehler durch den je nach Blickrichtung zwischen 1 und $\sqrt{3}$ variierenden Schichtab-

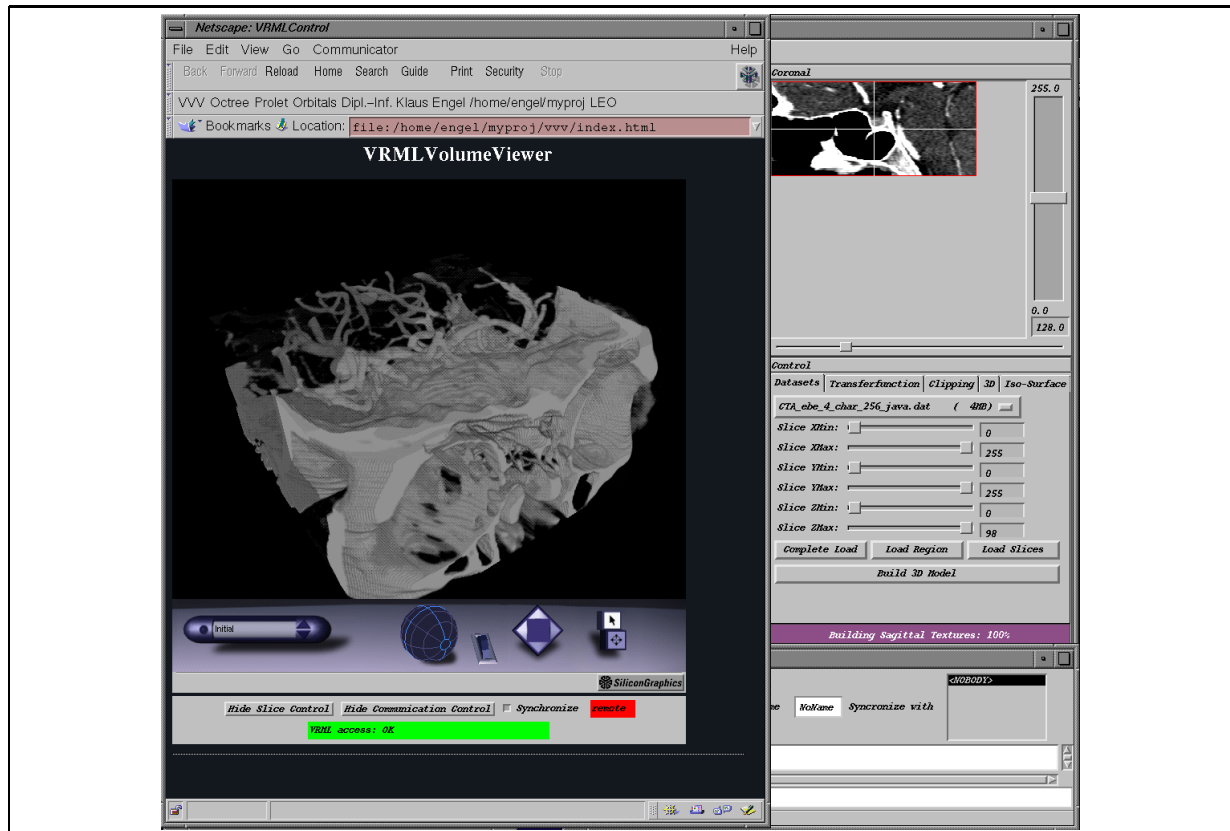


Abbildung 4.1: Texturbasiertes Volume-Rending mit einem VRML-Plugin. Die Daten werden von einem Java-Applet empfangen, aufbereitet und über die EAI-Schnittstelle an die entsprechenden Schichtpolygone in der VRML-Basisszene als Texturdaten weitergeleitet.

stand muss dabei vernachlässigt werden. Die Schichten werden von hinten nach vorne in den Framebuffer gezeichnet, wobei Compositing mit dem *Over-Operator* für das korrekte Überblenden der Texturen sorgt. Die Umsetzung dieses elementaren Algorithmus geschieht in den hier abgehandelten Schnittstellen zum Teil sehr unterschiedlich.

VRML wurde zum ersten Mal von Hendin[59] zur Visualisierung von medizinischen Volumendatensätzen eingesetzt. Der hier vorgestellte Ansatz[32] stellt eine Erweiterung dieses Verfahrens dar. Wie bereits im Abschnitt 2.4.4 erläutert, erlaubt das External Authoring Interface (EAI) unter Nutzung des VRML-Eventmodells den Zugriff auf Knoten des VRML-Szenengraphen. Zunächst wird in das VRML-Plugin, das zusammen mit dem Java-Applet in eine HTML-Seite eingebettet ist, nur eine Basis-VRML-Szene vom Server geladen. Diese besteht unter anderem aus einem *Switch*-Knoten, der für die Umschaltung der drei Schichtenstapel zuständig ist. Unter diesem *Switch*-Knoten sind die zugehörigen Gruppenknoten für die Schichtenstapel (siehe Abb. 4.2) angeordnet. Nachdem das Java-Applet über eine URL oder eine Socket-Verbindung den Volumendatensatz geladen hat, wird die VRML-Szene über das EAI mit

texturierten Slice-Polygonen aufgefüllt. Dazu werden über einen Aufruf von *createVrmlFromString* zunächst einzelne Slice-Polygone erzeugt (in Abb. 4.2 Teilszenengraphen unterhalb und einschließlich der *Toggle*-Knoten) und über ein Event dem jeweiligen Gruppenknoten hinzugefügt. Zu diesem Teilszenengraph gehört jeweils ein *Switch*- und ein *Shape*-Knoten mit den dazugehörigen *IndexedFaceSet*-, *Appearance*- und *PixelTexture*-Knoten. Die *PixelTexture*-Knoten werden mit RGBA-Texturdaten der jeweiligen Volumenschicht über ein entsprechendes *Event* gefüllt.

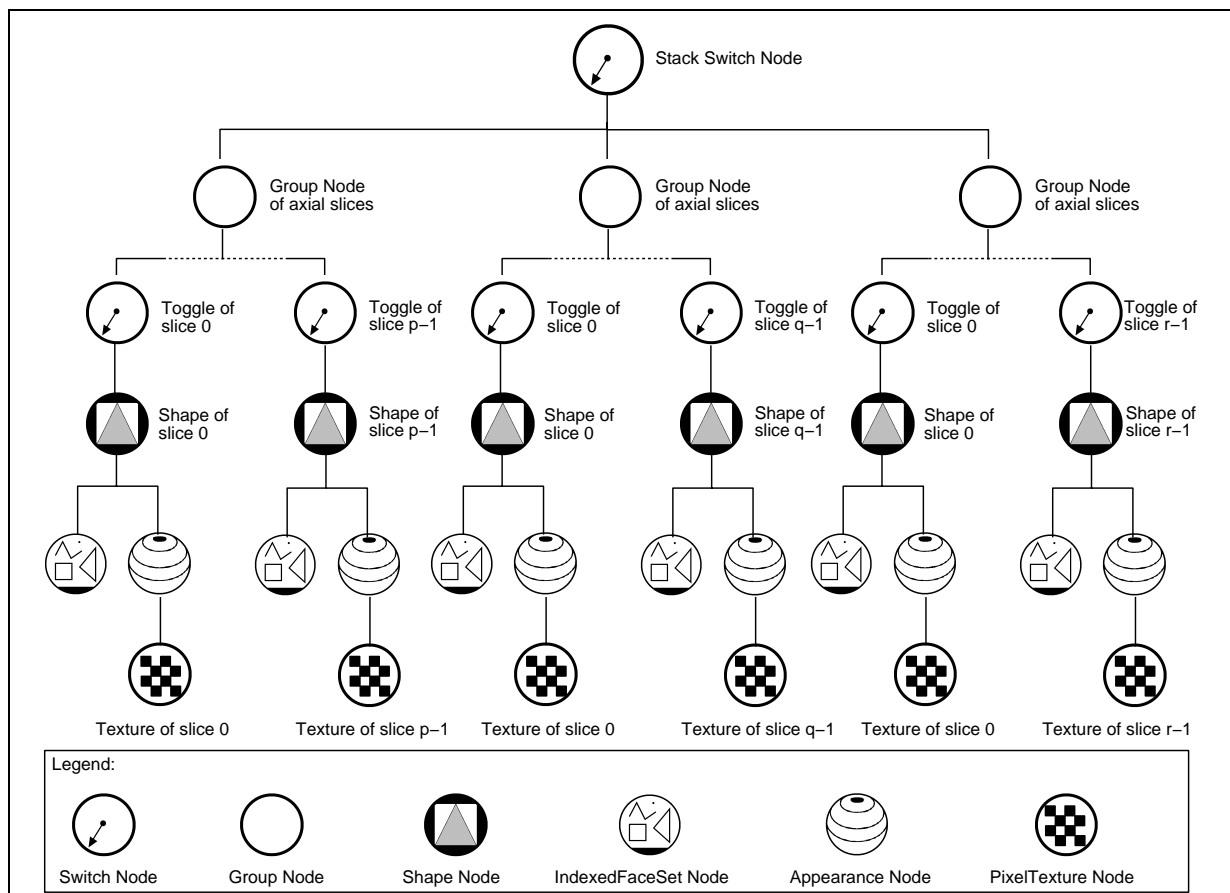


Abbildung 4.2: VRML - Szenengraph.

Ein *ProximitySensor* mit ausreichend großem Radius sorgt bei jeder Änderung des Blickwinkels auf die Szene durch Auslösen eines *Events*, das mit dem obersten *Switch*-Knoten verknüpft ist, für die Auswahl des jeweils benötigten Schichtenstapels.

Im Gegensatz zu VRML ist Java3D eine Szenengraphenprogrammierschnittstelle im eigentlichen Sinne, d.h. Java3D ermöglicht einen komfortablen Zugriff auf den Szenengraphen über Objekte und Methodenaufrufe. Java3D stellt sowohl Klassen für 2D- als auch für 3D-Texturen zur Verfügung, so dass sowohl eine Implementierung mit objekt-parallelen als auch viewport-parallelen Schnittpolygonen möglich ist. Wegen mangelnder Unterstützung der 3D-Texturen

wurde allerdings auf deren Nutzung verzichtet. Inzwischen wird für Java3D auch ein Treiber angeboten, der Java3D um hardwarebeschleunigtes Volume-Rendering mit einem Mitsubishi VolumePro-Board[114] unter Solaris erweitert.

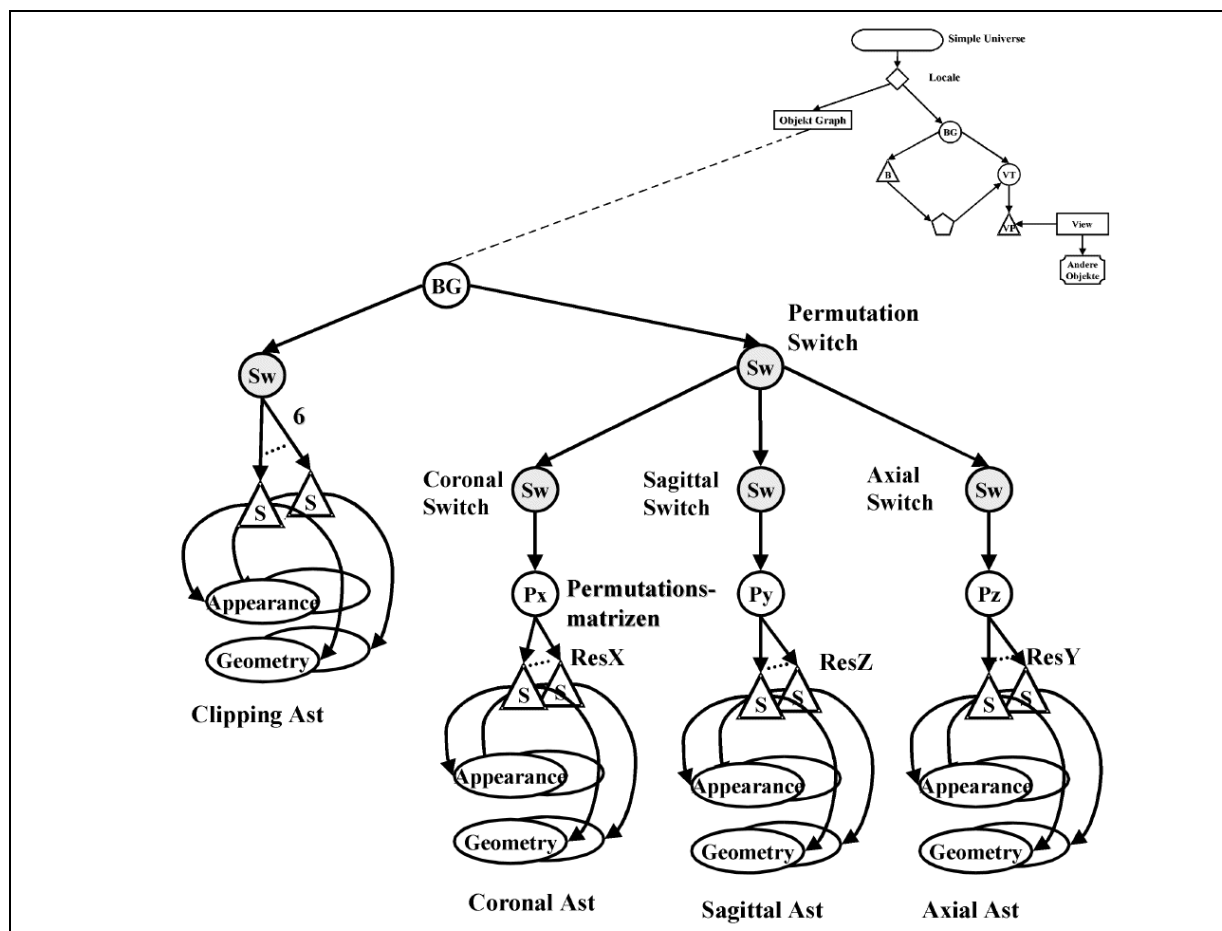


Abbildung 4.3: Szenengraph des Java3D Volume-Renderers.

Abbildung 4.3 zeigt eine Ausschnittvergrößerung des Java3D-Szenengraphen. Dieser besteht aus einem Clipping-Ast, der für die Darstellung der Clipping-Ebenen zuständig ist, und aus dem Volumen-Ast, der das Rendering des Volumens übernimmt. Der Volumen-Ast definiert die einzelnen Schichten des Volumens, wobei für jede ein eigener Shape3D-Knoten benutzt wird. Die Wahl des aktuellen Schichtenstapels wird über einen Auswahlknoten (PermutationSwitch) übernommen.

Java OpenGL-Anbindungen definieren im Gegensatz zu VRML und Java3D keine Datenhaltungsfunktionalität über einen Szenengraphen zur Verfügung. Vielmehr sind sie nur eine minimale Schicht zwischen Java und der *low-level*-Graphikschnittstelle OpenGL, die OpenGL-Funktionsaufrufe mittels Java Native Interface (JNI) Aufrufen an OpenGL weiterleiten. Die tex-

turierten Schnittpolygone werden für jedes Einzelbild direkt über die entsprechenden OpenGL-Funktionen von hinten nach vorne in den Framebuffer überblendet.

4.1.2 Transfer-Funktion

Da VRML und Java3D keine Möglichkeit zur Anwendung einer Farbtabelle während des Renderings bieten und OpenGL dies nur über herstellerspezifische Erweiterungen erlaubt, werden die skalaren Volumendaten in einem Vorverarbeitungsschritt in RGBA-Werte abgebildet. Somit lässt sich a priori nur vorklassifiziertes Shading erreichen. Die Transferfunktion kann im Java-Applet über ein entsprechendes Panel manipuliert werden, für jede Änderung der Transferfunktion ist allerdings ein erneuter Transfer der Texturen zur Graphikhardware notwendig, was eine interaktive Manipulation der Transferfunktion verhindert. Eine Vorschau der zu erwartenden Ergebnisse ist allerdings, zumindest was die RGB-Werte angeht, über ein 2D-Slicing-Modul möglich.

4.1.3 Ergebnisse

Die realisierten Prototypen zeigen, dass eine Implementierung interaktiver, texturbasierter Volume-Renderer auf der Basis jeder der in diesem Abschnitt vorgestellten APIs prinzipiell möglich ist. Diese ersten Ansätze zeigen schon deutlich das Potenzial PC-basierter Volumenvisualisierungslösungen. Bezüglich der Performance können keine signifikanten Unterschiede der Implementierungen festgestellt werden. Allenfalls bei einer Implementierung mit VRML hängt die Performance stark vom jeweils eingesetzten Plugin ab. Bei Datensätzen mit kleiner bis mittlerer Größe zeichnete sich schon mit PC-Graphikhardware der zweiten Generation (NVIDIA TNT Chip) eine weitgehende Annäherung an die Möglichkeiten professioneller Graphikhardware ab.

Die VRML-Implementierung zeigt je nach verwendetem Plugin gute Verarbeitungsgeschwindigkeit. VRML-Plugins sind für nahezu jede Plattform verfügbar und einfach einzurichten. Sowohl VRML als auch das EAI sind internationale Standards. Die Programmierung über das Event-Modell der EAI-Schnittstelle erweist sich allerdings als recht kompliziert und unkomfortabel. Zudem ist hier zu beobachten, dass die Plugins bei steigender Texturmenge überfordert sind, da diese für derlei Anforderungen in keiner Weise optimiert wurden. Hinzu kommt, dass die Plugins auch hinsichtlich der Interaktion mit der Szene eher auf das Begehen virtueller Welten als auf die Interaktion mit wissenschaftlichen Visualisierungen optimiert sind. Der VRML-Standard definiert zwar den Sprachumfang und die Anbindung an Java, die Ausgestaltung der Bedienelemente bleibt aber vollkommen den Herstellern der Plugins überlassen. Eine weiterer Nachteil von VRML ist die mangelnde Erweiterbarkeit. Für die Visualisierung wichtige, elementare Funktionalität wie Clipping-Ebenen, Farbtabelle oder auch der Zugriff auf neue Möglichkeiten der Graphikhardware bleiben somit verweigert. Dieses Problem wird sich zwar durch den kommenden Web-Standard X3D lösen, doch letztlich wird die Erweiterung von X3D auf die Definition neuer Knoten und das Erstellen von X3D-Plugin-Erweiterungen hinauslaufen.

Java3D bietet inzwischen relativ gute Performance und eine große Klassenbibliothek mit einer komfortablen Programmierschnittstelle. Java ist heute ein *de facto* Standard und deshalb in jedem Browser verfügbar. Java3D gehört aber als Erweiterung nicht zum Standardumfang der

Java2 Plattform. Der eigentliche Hauptnachteil von Java3D ist aber die fehlende Ansprechbarkeit herstellerspezifischer Erweiterungen. Java3D hinkt daher aktuellen Entwicklungen im Feld der Graphikhardware immer hinterher. Neueste Erweiterungen können erst nach der Aufnahme in den Java3D Standard genutzt werden. Bis zur Version 1.2 bot Java3D beispielsweise keine frei definierbaren Clipping-Ebenen und bis heute sind Farbtabelle für Texturen nicht vorgesehen.

Anbindungen von Java an die OpenGL-Graphikbibliothek erlauben dagegen prinzipiell, alle Funktionalität der Hardware zu nutzen, da die Hersteller von Graphikhardware üblicherweise diese über OpenGL-Erweiterungen zugänglich machen. OpenGL ist zudem als *de facto*-Industriestandard weit verfügbar. Die Architekturneutralität kann durch Abfrage herstellerspezifischer Erweiterungen vor deren Nutzung gewährleistet werden. Durch das Absetzen jedes OpenGL-Aufrufes über die JNI-Schnittstelle können allerdings vor allem beim Transfer großer Datenmengen Probleme bei der Darstellungsgeschwindigkeit entstehen. Dieses Problem wird allerdings im Java Developer Kit 1.4 durch das Konzept der direkten Puffer im neuen Paket `java.nio` zumindest teilweise behoben. Im Augenblick existiert leider kein offizieller Standard für die Anbindung von Java an OpenGL.

Letztlich erweist sich aber doch eine *low-level*-Anbindung von Java an OpenGL als die günstigste Alternative. Gerade bei der rapiden Entwicklung neuer Fähigkeiten der neuen Graphikhardware ist eine solche Anbindung unumgänglich. Im weiteren Verlauf dieses Kapitels wird wegen der hier beschriebenen Probleme von VRML- oder Java3D-basierten Implementierungen eine Verwendung von OpenGL unter Zuhilfenahme von Java-Anbindungen vorausgesetzt.

4.2 Multi-Textur Volume-Rendering

Wie bereits in Kapitel 2.2 beschrieben ist es bei der Visualisierung in digitalen Netzwerken wünschenswert, die immens gestiegene Leistung günstiger PC-Graphikhardware verfügbar zu machen und dabei Spezialfunktionalität der Hardware optimal zu nutzen. Zur Funktionalität von PC-Graphikhardware gehören vor allem Multi-Texturen. Die Möglichkeiten, die sich für die wissenschaftliche Visualisierung durch Multi-Texturen ergeben, wurden vor allem von den Herstellern professioneller Graphikhardware lange unterschätzt und auch heute noch gibt es keine breite Unterstützung im Workstation-Bereich. In Verbindung mit flexiblen, programmierbaren Rasterisierungseinheiten sind Multi-Texturen bei der texturbasierten Volumenvisualisierung von größter Bedeutung. Dies ergibt sich vor allem dadurch, dass pro Fragment auf mehr Informationen aus dem Volumen zugegriffen werden kann, also anstelle eines gefilterten Wertes mehrere Werte in die Berechnungen einbezogen werden können.

Die Nützlichkeit von Multi-Texturen und flexibler Rasterisierung für das texturbasierte Volume-Rendering bezüglich Qualität, Geschwindigkeit und Effektivität wurde in [120] demonstriert. Um eine interaktive Anpassung der Transferfunktion bei den folgenden Ansätzen zu ermöglichen, werden prä-interpolative Farbtabelle (vorklassifiziertes Shading), wie sie etwa in NVIDIAs *GeForce256* Prozessor unterstützt werden, eingesetzt. Die Implementierung der hier vorgestellten Verfahren erfolgt exemplarisch anhand der bereits in Kapitel 2.2.4 vorgestellten Register-Combiner OpenGL-Erweiterung von NVIDIA. Inzwischen werden derartige Operationen auch von weiteren Herstellern, wie beispielsweise ATI, mit ähnlichen OpenGL-

Erweiterungen unterstützt und sind ab Version 8.0 der Microsoft DirectX-API standardisiert.

4.2.1 Multi-Textur-Interpolation

In heutige PC-Graphikhardware halten zwar 3D-Texturen kontinuierlich Einzug, dennoch sind diese noch nicht breit verfügbar. Dagegen ist Graphikhardware, die 2D-Texturen mit Multi-Textur-Funktionalität unterstützt, inzwischen in nahezu jedem Arbeitsplatzrechner vorhanden. Neben den Nachteilen, die bei der Nutzung von 2D-Texturen für das Volume-Rending entstehen, wie dreifachem Speicherbedarf und Stapel-Umschalt-Effekten, haben 2D-Texturen auch einen entscheidenden Vorteil: Volume-Rending-Verfahren basierend auf 2D-Texturen sind im Allgemeinen performanter als ihre Pendanten basierend auf 3D-Texturen. Dies lässt sich primär durch die bessere Cache-Kohärenz aufgrund des geringeren Speicherbedarfs einzelner 2D-Texturen erklären.

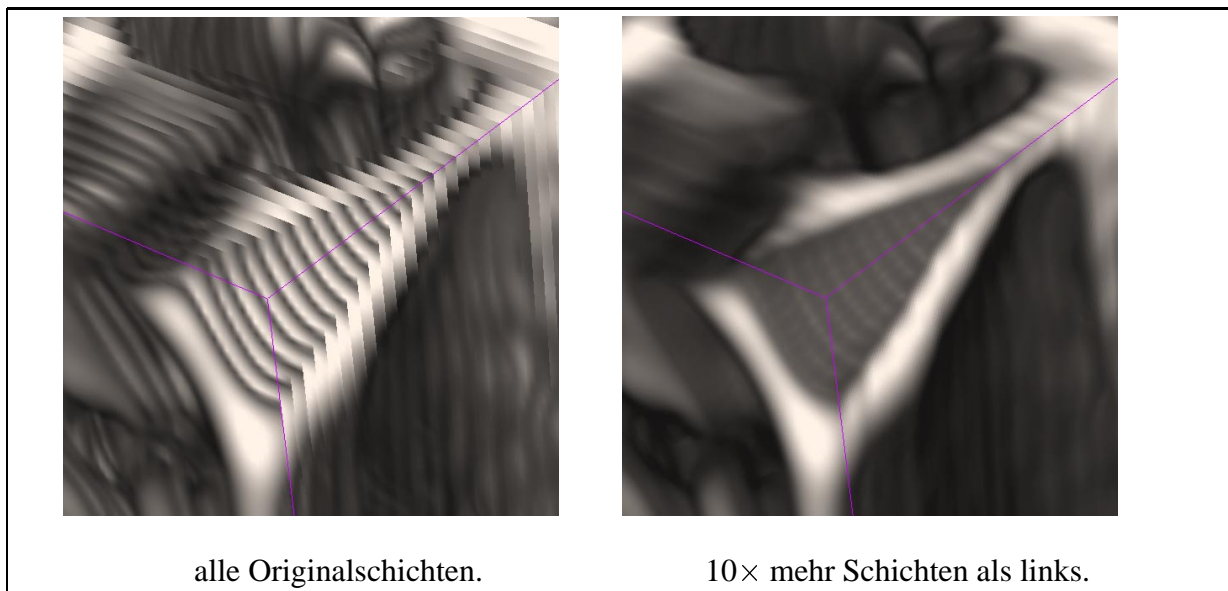


Abbildung 4.4: Artefakte bei objekt-parallelen Schichten und fehlender trilinearere Interpolation (links) im Vergleich mit trilinearere Interpolation (rechts).

Allerdings verursacht die bilineare Filterung der Daten anstelle einer trilinearen Filterung störende Artefakte in den resultierenden Bildern. Abbildung 4.4 (links) veranschaulicht diese Unterabtastungsartefakte an den Volumengrenzen bei semi-transparenter Darstellung.

Unter Zuhilfenahme von Multi-Texturen ist aber eine trilineare Interpolation auch mit 2D-Texturen zu erreichen (siehe Abbildung 4.4, rechts). Dazu werden Zwischenschichten eingefügt, die mit linearen Kombinationen angrenzender Schichten texturiert werden. Die trilineare Interpolation einer Texturschicht $S_{i+\alpha}$ kann beschrieben werden als Überblenden zweier aneinandergrenzender bilinear interpolierter Schichten S_i und S_{i+1} :

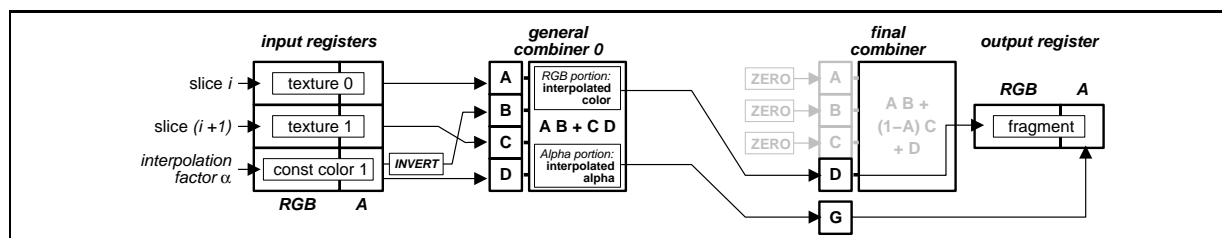


Abbildung 4.5: Combiner-Einstellungen zur Interpolation von Zwischenschichten.

$$S_{i+\alpha} = (1 - \alpha) \cdot S_i + \alpha \cdot S_{i+1}. \quad (4.1)$$

Es wird zunächst eine bilineare Interpolation durch die Textureinheit mittels der entsprechenden Texturumgebung erzielt, anschließend erfolgt die fehlende dritte Interpolation in der Rasterisierungseinheit. Die zwei angrenzenden Schichten S_i und S_{i+1} werden als Multi-Texturen an die Zwischenschicht gebunden und jeweils zwei korrespondierende Texel der Schichten in der Rasterisierungseinheit werden miteinander kombiniert. Solche Berechnungen werden auf PC-Graphikkarten mittlerweile standardmäßig unterstützt, beispielsweise in DirectX 8.0 oder unter OpenGL durch die OpenGL-Erweiterungen `GL_ARB_texture_env_combine`, `GL_NV_texture_env_combine4` und NVIDIAs *Register-Combiners*-Erweiterung. Wie in Abbildung 4.5 demonstriert, kann die Interpolation mit einer einzigen *General-Combiner*-Stufe realisiert werden. Dazu wird der Interpolationsfaktor α im Register für die konstante Farbe abgelegt. Durch eine entsprechende Eingabeabbildung (*Input Mapping*) wird daraus $(1 - \alpha)$ für die Berechnung der gewichteten Summe $AB + CD$ gewonnen. Der Alpha-Teil der Stufen wird analog verschaltet, um interpolierte Alpha-Werte zu gewinnen.

Da die Multi-Texturierung und die Kombination der Texel in einem Taktzyklus durch zwei parallele Rasterisierungseinheiten erfolgen, wird eine interpolierte Schicht mit nahezu der gleichen Geschwindigkeit dargestellt wie ein einfach texturiertes Polygon. Somit können beliebig viele Zwischenschichten, ohne zusätzlichen Speicherbedarf eingefügt werden. Abbildung 4.12 (rechts) demonstriert die Erhöhung der Bildqualität, die durch diesen Ansatz erreicht wird. Analog zu einem Ansatz basierend auf 3D-Texturen müssen die Opazitätswerte bei direktem Volume-Rendering für den neuen Schichtabstand angepasst werden. Dies kann beispielsweise durch Anwendung eines konstanten linearen Skalierungsfaktor approximiert werden. Da der Schichtabstand bei Verwendung dreier achsenorthogonaler Schichtenstapel bei orthogonaler Projektion je nach Kameraposition zwischen 1 und $\sqrt{3}$ variiert (bei perspektivischer Projektion sind die Variationen noch größer), sollten ausschließlich interpolierte Schichten dargestellt werden, deren Position dynamisch für jede Betrachterposition neu gewählt wird, um einen konstanten Schichtabstand beizubehalten.

Neben der Möglichkeit Zwischenschichten zu interpolieren, können mit diesem Ansatz auch beliebig ausgerichtete Schichtbilder aus den Volumendaten rekonstruiert werden. In der Medizin wird dieser Vorgang als multi-planare Rekonstruktion (MPR) bezeichnet.

Erstmals wurde die folgende Technik, die leicht auf Multi-Textur-adaptiert werden kann, in [29] vorgestellt. Das Grundprinzip des Algorithmus ist in Abbildung 4.6 dargestellt. Zunächst

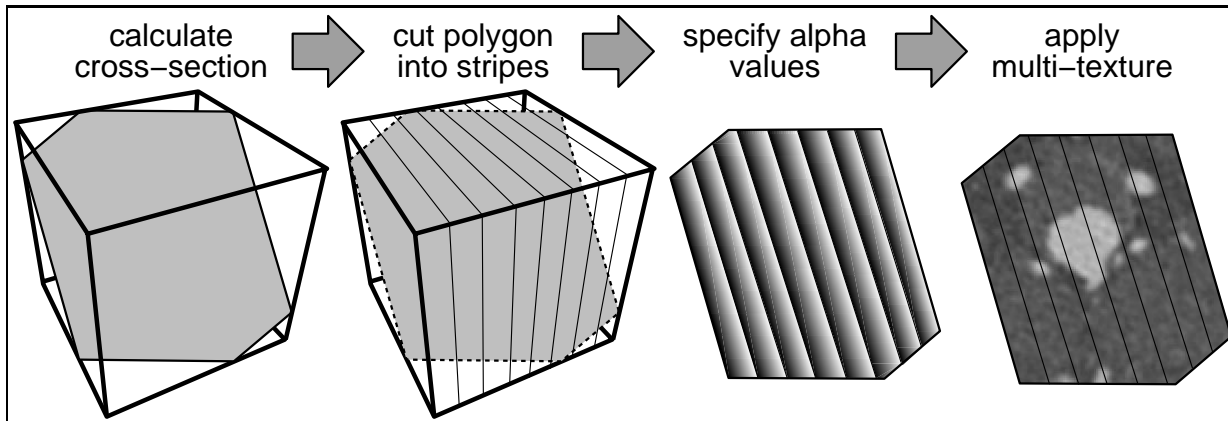


Abbildung 4.6: Verfahren zur Rekonstruktion beliebig ausgerichteter Schichtbilder aus Volumendaten.

wird ein Schnittpolygon der Schnittebene mit der Bounding Box des Volumens berechnet. Das Schnittpolygon wird dann in eine Reihe von Polygonstreifen an den Schnittlinien mit den am Objekt ausgerichteten Texturschichten unterteilt. Nachfolgend wird auf jedem dieser Polygonstreifen die Bildinformation trilinear gewonnen, indem die zwei angrenzenden Texturschichten interpoliert werden. Dazu wird wiederum die in Abbildung 4.5 vorgestellte Combiner-Einstellung herangezogen, wobei der Interpolationsfaktor in diesem Falle aber auf die primäre Farbe des Polygons gesetzt wird. Im Alpha-Kanal der primären Farbe werden nun, an den Vertices, die durch den Schnitt mit der ersten Textur entstanden sind, Alpha-Werte von 0 gesetzt, an den Vertices, die durch den Schnitt mit der zweiten Textur entstanden sind, Alpha-Werte von 1 gesetzt. Durch Gouraud Shading werden diese Alpha-Werte, und damit der Interpolationsfaktor, linear über die Polygonstreifen interpoliert. Dies bewirkt unter Anwendung der Register-Combiner, dass die Bildinformation trilinear auf den Polygonstreifen interpoliert wird.

Da die Interpolation im Gegensatz zu der Technik in *OpenGL Volumizer*[29] nicht im Framebuffer, sondern während der Rasterisierung erzielt wird, können auf diese Weise statt eines einzelnen Schichtbilds beliebig viele Schichtbilder rekonstruiert und im Framebuffer gemischt werden. Es kann also ein Volume-Rendering-Ansatz analog dem 3D-texturbasierten Verfahren realisiert werden. Da das Volumen dafür nur noch einfach gespeichert werden muss, fällt somit einer der Hauptnachteile des 2D-texturbasierter Verfahren weg. Allerdings sind durch den zusätzlichen Aufwand der Schnittpunktberechnungen und die größere Anzahl von Polygonen nicht die Bildwiederholraten eines 2D-Ansatzes zu erzielen. Durch Ausrichten der Schnittpolygone an den Volumendaten, analog zum Rendern mit objekt-parallel Schichten, wird die Schnittpunktberechnung stark vereinfacht, dabei ist aber wieder das Rendern dreier alternativer Schichtenstapel notwendig. Allerdings werden die Volumendaten weiterhin nur einfach im Speicher gehalten. Eine weitere Geschwindigkeitsoptimierung ist dadurch zu erreichen, dass anstelle des Schicht-für-Schicht-Überblendens von hinten nach vorne in den Framebuffer, die Polygonstreifen eines Schichtzwischenraums (*Slabs*) von hinten nach vorne überblendet werden, bevor der nächste Slab betrachtet wird. Dies hat eine bessere Cache Kohärenz und weniger Texturbindungen zur

Folge. Dabei ist zu beachten, dass für ein korrektes Blending eine Sortierung der Slabs notwendig ist.

4.2.2 Geschwindigkeitsoptimierung

Neben der Verbesserung der Bildqualität bei Verwendung von 2D-Texturen können Multi-Texturen auch das Rendering von Volumendaten beschleunigen. Die zugrunde liegende Idee ist, durch gleichzeitige Überlagerung mehrerer Schichtbilder auf einem einzigen Schnittpolygon die Anzahl der Schnittpolygone und damit die Anzahl der Überblendungsoperationen im Framebuffer zu reduzieren.

Sind n Multi-Textur-Einheiten verfügbar, so wird nur jedes n -te Schnittpolygon gezeichnet, allerdings mit den Texturen von n aufeinander folgenden Schichten. Da das Compositing der Volumenschichten assoziativ ist, werden die Texturen zuerst in der Multi-Textur-Einheit kombiniert, bevor das resultierende Fragment in den Framebuffer geblendet wird. Durch perspektivische Verschiebung der Schichten gegeneinander wird natürlich so nur diejenige Textur an der korrekten Stelle gezeichnet, die zu der an der Stelle liegenden Schicht gehört. Deshalb ist für alle anderen Schichten eine Korrektur der Texturposition notwendig, die aber durch Anpassung der Texturkoordinaten entsprechend der Projektion der Texturen auf das Schnittpolygon erreicht werden kann (siehe Abbildung 4.7). Dies kann gegebenenfalls durch automatische Texturkoordinaten

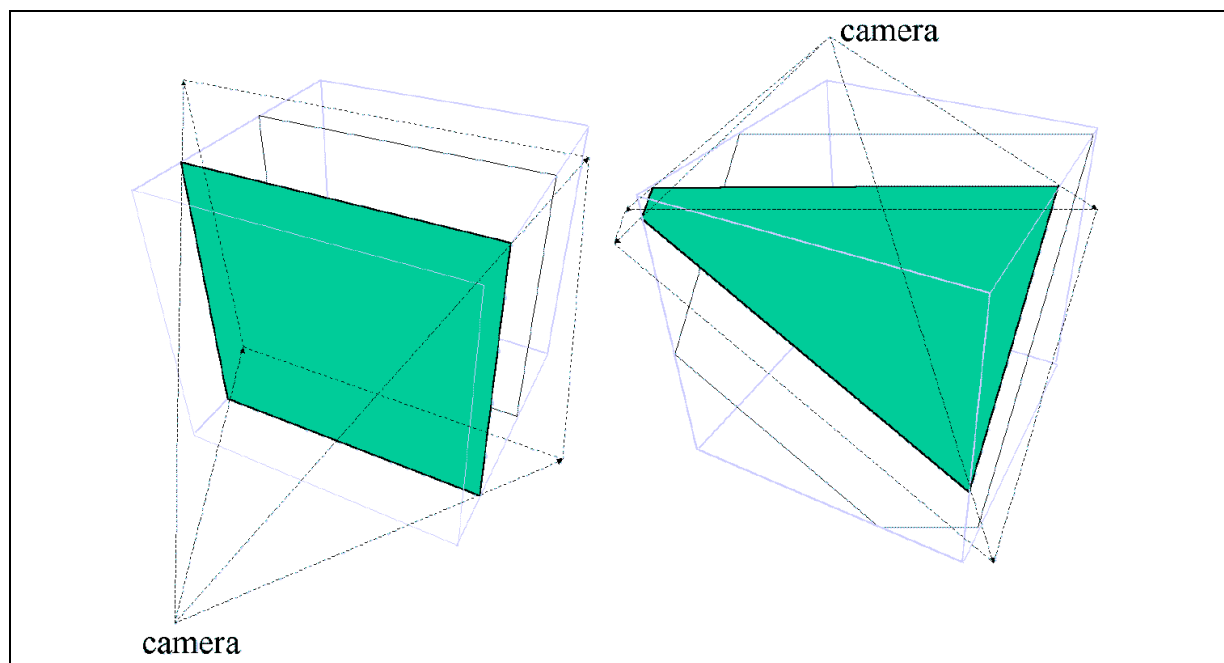


Abbildung 4.7: Projektion angrenzender Texturschichten auf ein Schnittpolygon für objekt-parallele und viewport-parallele Schichten.

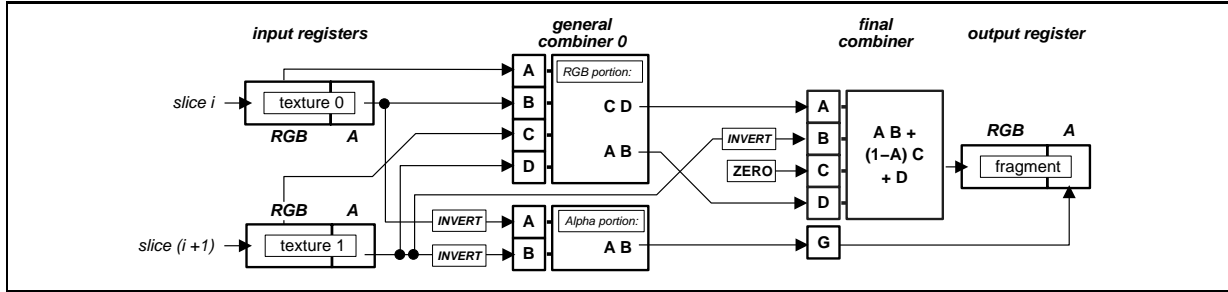


Abbildung 4.8: Combiner-Einstellungen zum korrekten Überblenden zweier Schichten.

datengenerierung mit dem Parameter `GL_EYE_LINEAR` realisiert werden.

Für das Compositing der angrenzenden Schichten ist wieder programmierbare Rasterisierungshardware notwendig. Die Konfiguration der Rasterisierungshardware wird im Folgenden am Beispiel der Register Combiner von NVIDIA erläutert. Dabei wird der Fall zweier Texturschichten pro Schnittpolygon betrachtet. In den Überlegungen bezeichnet der Buchstabe *C* eine Farbkomponente *R*, *G* oder *B*, während *A* auf einen Alpha-Wert verweist. Indizes t_0 und t_1 deuten auf die Zugehörigkeit der Werte zur ersten bzw. zweiten Textur hin. Ein Index d bedeutet, dass es sich um einen Ziel-Wert des Fragments handelt, während ein Index s sich auf einen Quell-Wert des Fragments bezieht.

Das Blending eines texturierten Polygons in den Framebuffer mit dem *Over-Operator* ergibt

$$C'_d = C_{t_0}A_{t_0} + C_d(1 - A_{t_0}). \quad (4.2)$$

Dieser Farbwert wird nun als Ziel-Wert beim Blending des zweiten texturierten Polygons verwendet. Es ergibt sich

$$\begin{aligned} C''_d &= C_{t_1}A_{t_1} + C'_d(1 - A_{t_1}) \\ &= C_{t_1}A_{t_1} + (C_{t_0}A_{t_0} + C_d(1 - A_{t_0}))(1 - A_{t_1}) \\ &= C_{t_1}A_{t_1} + C_{t_0}A_{t_0}(1 - A_{t_1}) + C_d(1 - A_{t_0})(1 - A_{t_1}). \end{aligned}$$

Um diese Blending-Operation in den Register-Combinern nachzubilden, müssen diese wie in Abbildung 4.8 gezeigt programmiert werden. Der RGB-Teil des General-Combiners 0 berechnet $C_{t_0}A_{t_0}$ und $C_{t_1}A_{t_1}$. Der Alpha-Teil wird zur Berechnung von $(1 - A_{t_0})(1 - A_{t_1})$ benutzt. Die Ausgabe des RGB-Teils wird in den Final-Combiner geleitet, der den finalen Farbwert des Fragments berechnet:

$$C_s = C_{t_0}A_{t_0}(1 - A_{t_1}) + C_{t_1}A_{t_1}. \quad (4.3)$$

Das Ergebnis des Alpha-Teils des Combiners wird direkt als Fragment-Alpha durch den Final-Combiner weitergeleitet. Dieses Fragment wird mittels der Blending-Funktion `glBlendFunc(GL_ONE, GL_SRC_ALPHA)` in den Framebuffer geschrieben. Daraus ergibt sich das gewünschte Ergebnis:

$$\begin{aligned}
 C_d'' &= C_s \cdot 1 + C_d \cdot A_s \\
 &= C_{t_1} A_{t_1} + C_{t_0} A_{t_0} (1 - A_{t_1}) + C_d (1 - A_{t_0}) (1 - A_{t_1}).
 \end{aligned}$$

Durch Einsatz dieses Verfahrens können also mehrere Schichten des Volumens gleichzeitig bearbeitet und damit die Anzahl der Blending-Operationen im Framebuffer verringert werden. Die sich daraus ergebende Geschwindigkeitsoptimierung wird anhand der in Abschnitt 4.2.6 aufgezeigten Ergebnisse deutlich. Eine weitere Verbesserung ist hinsichtlich der erzielten Rendering-Qualität zu erwarten, da Quantisierungsartefakte durch die niedrigere Zahl der Blending-Operationen im Framebuffer verringert werden. Diese Blending-Operationen erfolgen bei heutigen Graphikchips mit nur 8 Bit Genauigkeit. Dagegen arbeitet der R200 Graphikchip von ATI bei Fragmentoperationen in der sogenannten *Fragment-Shader*-Einheit mit 16 Bit Genauigkeit.

4.2.3 Beleuchtete Isoflächen

Moderne Multi-Textur-Hardware unterstützt weitaus komplexere Möglichkeiten der Texturkombination. So kann durch den Einsatz mehrstufiger Rasterisierungshardware das in Abschnitt 2.3.6 vorgestellte Two-Pass-Verfahren zum Rendering beleuchteter Isoflächen durch ein One-Pass-Verfahren ersetzt werden.

Die Basis dieses Ansatzes ist wiederum die Kodierung vorberechneter Gradienten in den RGB-Kanälen und der Volumendaten im Alpha-Kanal einer RGBA-Textur. Durch Einsatz des OpenGL Alpha-Tests passieren ausschließlich Fragmente den Alpha-Test, die den Isowert je nach gesetztem Alpha-Test entweder über- oder unterschreiten. Es ergeben sich einseitige Isoflächen, deren Beleuchtung während der Rasterisierung mit dem Phong-Modell ermittelt werden kann. Dazu wird in der Rasterisierung das für die Beleuchtung benötigte Skalarprodukt zwischen Gradient und Lichtrichtung (bzw. Halfway-Vektor) bestimmt. Je nach Flexibilität der Rasterisierungshardware und Anzahl der Combiner-Stufen lassen sich so unterschiedlich komplexe Beleuchtungsverhältnisse realisieren.

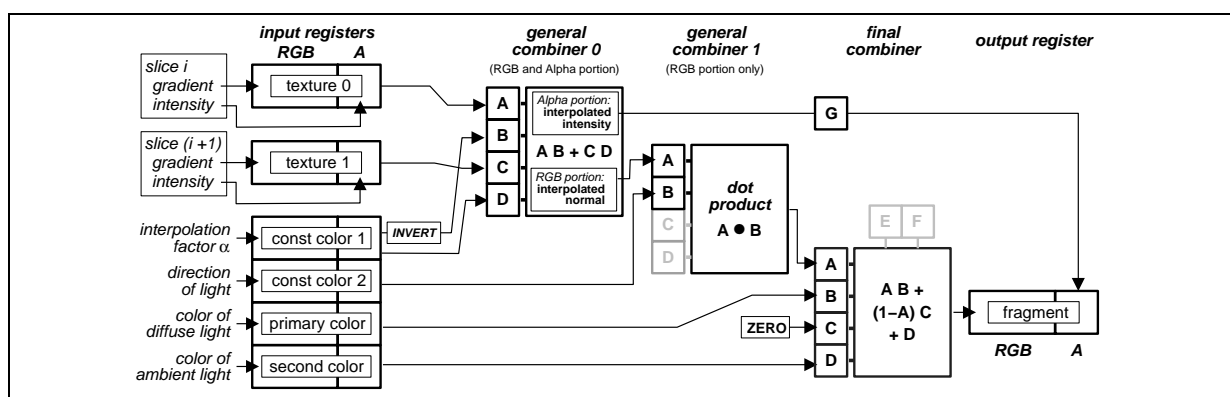


Abbildung 4.9: Combiner-Einstellungen zum Darstellen beleuchteter Isoflächen.

Abbildung 4.9 demonstriert anhand der Register-Combiner OpenGL-Erweiterung die Kombination der im vorigen Abschnitt demonstrierten trilinearen Interpolation mit einer einfachen Beleuchtung mit ambientem und diffusem Anteil. Der erste Combiner realisiert die trilineare Interpolation der Gradienten, während der zweite Combiner über ein Skalarprodukt den Koeffizienten für den diffusen Beleuchtungsterm berechnet. Im Final-Combiner werden schließlich die ambienten und diffusen Terme aufaddiert und die so ermittelte Fragmentfarbe in der Rendering-Pipeline weitergereicht.

Durch eine andere Verschaltung der Register-Combiner und weitere Combiner-Stufen sind andere Beleuchtungseffekte realisierbar, wie beispielsweise spekulare Beleuchtung und farbige Lichtquellen. In [8] wird zusätzlich die Möglichkeit beidseitig beleuchteter und semi-transparenter Isoflächen demonstriert.

Alternativ zum Einsatz speicherintensiver RGBA-Texturen können bei statischer Beleuchtung mit einfarbigen Lichtquellen auch vorberechnete Skalarprodukte im Luminanz-Kanal und Volumendaten im Alpha-Kanal einer Luminanz-Alpha-Textur gespeichert werden. Dies vermeidet nebenbei auch Artefakte durch die fehlende Renormalisierung interpolierter Gradienten.

Abschließend ist zu betonen, dass dieser Ansatz der Visualisierung von Isoflächen einen interaktiven Wechsel des Isowertes ermöglicht. Die dabei erzielte Bildwiederholrate entspricht der bei einer beliebigen Transformation des Volumens erreichten Rate. Dies ist deshalb möglich, weil dazu nur der Alpha-Test-Referenzwert neu zu setzen und ein erneuter Rendering-Vorgang zu starten ist. Im Gegensatz zu indirekten Volumenvisualisierungsmethoden ist keine aufwändige Rekonstruktion von Flächen und das Rendering von enormen Mengen geometrischer Primitive nötig.

4.2.4 Beleuchtung semi-transparenter Volumina

Eine weitere Anwendung flexibler Rasterisierungshardware findet sich in der Darstellung beleuchteter semi-transparenter Volumina[97]. Die zugrunde liegende Idee ist im Wesentlichen eine Erweiterung des in Abschnitt 4.2.3 vorgestellten Algorithmus, wobei an die Stelle des Alpha-Tests nun Alpha-Blending tritt. Vorberechnete Volumen-Gradienten werden wieder in den RGB-Komponenten einer Textur abgelegt. Da die in diesem Abschnitt eingesetzte prä-interpolative Farbtabelle über die `EXT_paletted_texture` OpenGL-Erweiterung keinen Tabellennachschlag einer einzelnen Komponente der Textur zulässt, muss die Intensität in einer weiteren Textur abgelegt werden.

Die Register-Combiner werden wie in Abbildung 4.10 konfiguriert. Im ersten Combiner wird das Skalarprodukt für diffuse Reflexion aus dem Gradienten und dem Lichtvektor ermittelt. Der Final-Combiner berechnet die Farbe des Fragments aus der Summe des über die Transferfunktion ermittelten Farbwertes und der sich aus der Beleuchtungsberechnung ergebenden Farbe. Als Alpha-Kanal des Fragments kommt der nachgeschlagene Alpha-Wert aus der Transferfunktion zum Einsatz.

Eine trilineare Interpolation ist bei diesem Ansatz entweder durch Einsatz von 3D-Texturen oder durch Einsatz weiterer Multi-Texturen unter Verwendung von 2D-Texturen möglich. Da die in der Implementierung für diesen Abschnitt verwendete *GeForce256*-Hardware aber leider über diese Möglichkeiten nicht verfügt, ist entlang einer Achse nur eine *Nearest-Neighbor*-

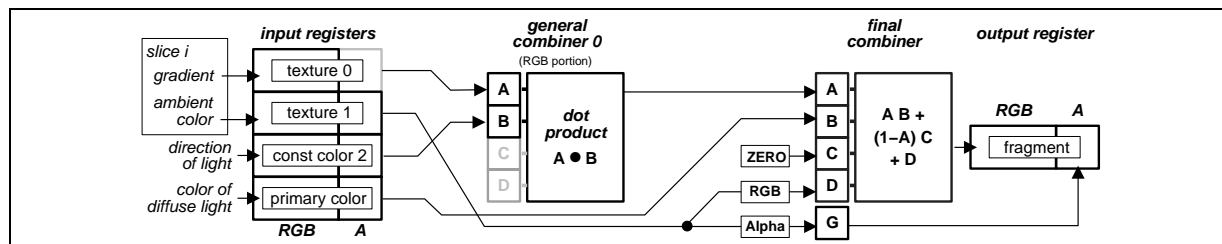


Abbildung 4.10: Combiner-Konfiguration zur Darstellung semi-transparenter Volumina mit diffuser lokaler Beleuchtung.

Interpolation durch Einfügen weiterer Schichten realisierbar. Diese Einschränkung ist in aktueller Graphikhardware nicht mehr gegeben.

Auch bei diesem Ansatz können durch den Einsatz weiterer Combiner-Stufen komplexere Beleuchtungsverhältnisse realisiert werden. In [8] wird zusätzlich eine Gewichtung der Opazität mit dem Betrag des lokalen Gradienten demonstriert.

4.2.5 Weitere Anwendungen

Neben den oben aufgeführten Anwendungen lassen sich Multi-Texturen in weiterer, vielfältiger Weise zur Volumenvisualisierung nutzen. In der 3D-Visualisierung tomographischer Volumendaten in der medizinischen Bildgebung sind einzelne Voxel häufig durch eine explizite Segmentierung mit weiteren Informationen versehen [139], so dass beispielsweise verschiedene Gewebearten unterschieden werden können. Für die Visualisierung dieser Daten ist es wünschenswert, einzelne markierte Voxel des Volumens anhand dieser Zusatzinformation einfärben oder entfernen zu können. Genau dies ist mit Hilfe von Multi-Texturen sehr effizient möglich.

Dazu wird für jedes Voxel des Volumens neben dem eigentlichen Skalarwert in einer ersten Textur Zusatzinformation in einer zweiten Textur abgelegt. Diese Zusatzinformation ist meist in Form einer Marke (Tag), also als einer ganzen Zahl zwischen 1 und der maximalen Anzahl an Marken, gegeben (0 steht für unmarkierte Voxel). Nachdem für Volumendaten und Zusatzinformation diesselbe Farbtabelle verwendet wird, erfolgt eine Partitionierung der Lookup-Tabelle, so dass durch den Einsatz der Lookup-Tabelle für die Voxel sowie für die Marken jeweils ein RGBA-Wert nachgeschlagen wird. Der Bereich der Marken wird dazu reserviert. Da diese ohnehin in einem Wertebereich liegen, wo in tomographischen Volumendaten Rauschen auftritt, können die Volumendaten dort abgeschnitten werden.

Die für die Voxel in der Lookup-Tabelle nachgeschlagenen RGBA-Werte können nun in den Multi-Textur-Combinern verarbeitet werden. Dazu kann ein gewichtetes Produkt $AB + CD$ verwendet werden, mit dem eine Gewichtung oder eine Verschiebung der Farbkanäle der RGBA-Werte der Voxel erzielt werden kann. Dadurch werden Bereiche der Volumendaten mit unterschiedlichen Marken verschieden eingefärbt oder mit Hilfe des Alpha-Kanals ein- oder ausgeblendet. Die Gewichtung kann interaktiv manipuliert werden, indem in der Transferfunktion die Einträge für die Marken geändert werden.

Eine Spezialanwendung des Verfahrens stellt das Clippen von Volumendaten mit beliebigen Clip-Geometrien dar. Dazu wird eine einzelne Marke definiert, die angibt, ob ein Voxel ausgeblendet wird. Diese Marke wird in einem zweiten Volumen (*Stencil-Volumen*) für alle Voxel gesetzt, die nicht dargestellt werden sollen. Die Auflösung des Stencil-Volumens kann je nach Komplexität der Clip-Geometrie unabhängig von der des Datenvolumens gewählt werden. In Abbildung 4.11 wird dies anhand dreier Clip-Geometrien demonstriert. Im Fall der sphärischen Clip-Geometrie wird ein Stencil-Volumen mit der Auflösung des Datensatzes gewählt, während im Fall der kubischen und zylindrischen Clip-Geometrie eine einzige, wiederholte 2D-Textur eingesetzt wurde. Die Stencil-Textur kann durch die Veränderung ihrer Textur-Koordinaten interaktiv bewegt werden. Bei Einsatz von 2D-Texturen ist nur eine Skalierung und Translation möglich, während 3D-Texturen zusätzlich auch die Rotation des Stencil-Volumens erlauben.

Zusätzlich zu der in Abschnitt 4.2.1 vorgestellten räumlichen Interpolation ergibt sich für zeitabhängige Volumendaten auch die Möglichkeit einer zeitlichen Interpolation mittels Multi-Texturen. Dazu werden statt zweier aneinander grenzender Schichten eines Volumens zwei oder mehr sich räumlich entsprechender Schichten aus verschiedenen Zeitschritten miteinander gewichtet. Es wird dieselbe Combiner-Konfiguration wie in Abschnitt 4.2.1 eingesetzt, der Interpolations-Faktor gibt aber nun statt der räumlichen Nähe zu einer der Schichten die zeitliche Nähe zu einem der Zeitschritte an. Auf diese Weise können die verschiedenen Zeitschritte weich ineinander übergeblendet werden, wodurch auch zeitabhängige Volumendaten mit wenigen Einzelschritten flüssig animiert werden können. Während des Überblendens zweier zeitlich aufeinander folgender Volumen kann bereits ein Volumen eines anschließenden Zeitschritts in den Texturspeicher nachgeladen werden. Dazu sind allerdings asynchrone Mechanismen des Transfers von Texturdaten in den lokalen Speicher der Graphikhardware nötig. Ein Beispiel eines solchen zeitabhängigen Volumendatensatzes ist auf den Farbseiten in Abbildung 9.5 dargestellt.

4.2.6 Ergebnisse

Während sich der erste Abschnitt dieses Kapitels noch der Frage widmete, ob Graphikhardware heutiger Arbeitsplatzrechner zur interaktiven Visualisierung wissenschaftlicher Volumendaten eingesetzt werden kann, wurde in diesem Abschnitt demonstriert, dass flexible Mehrstufen-Rasterisierungseinheiten inzwischen die Fähigkeiten professioneller Graphikhardware erreichen, zum Teil sogar übertreffen. Es wurde demonstriert, dass sich durch die Möglichkeit der Bearbeitung zeitlich und räumlich kohärenter Daten auf Fragment-Basis und die flexiblen Kombinationsmöglichkeiten der PC-Graphikhardware völlig neue Perspektiven eröffnen.

Für die folgenden Messungen der Geschwindigkeit und Qualität der auf Multi-Texturen basierenden Volume-Rendering-Algorithmen wurde ein PC eingesetzt, der mit einer 500 MHz Pentium III CPU, 512 MB Speicher und einer NVIDIA GeForce256 Graphikkarte mit 32 MB DDR SDRAM und 2x AGP ausgestattet war.

Zunächst kann die Darstellungsqualität 2D-texturbasierter Volume-Rendering-Verfahren durch das Einfügen trilinear interpolierter Zwischenschichten deutlich verbessert werden. Die fehlende dritte lineare Interpolation bei Verwendung von 2D-Texturen wird dabei während der Rasterisierung durch die Kombination zweier aneinandergrenzender Texturschichten erzielt, ohne dass dazu zusätzlicher Speicher benötigt wird. Abbildung 4.12 verdeutlicht den Qualitätsge-

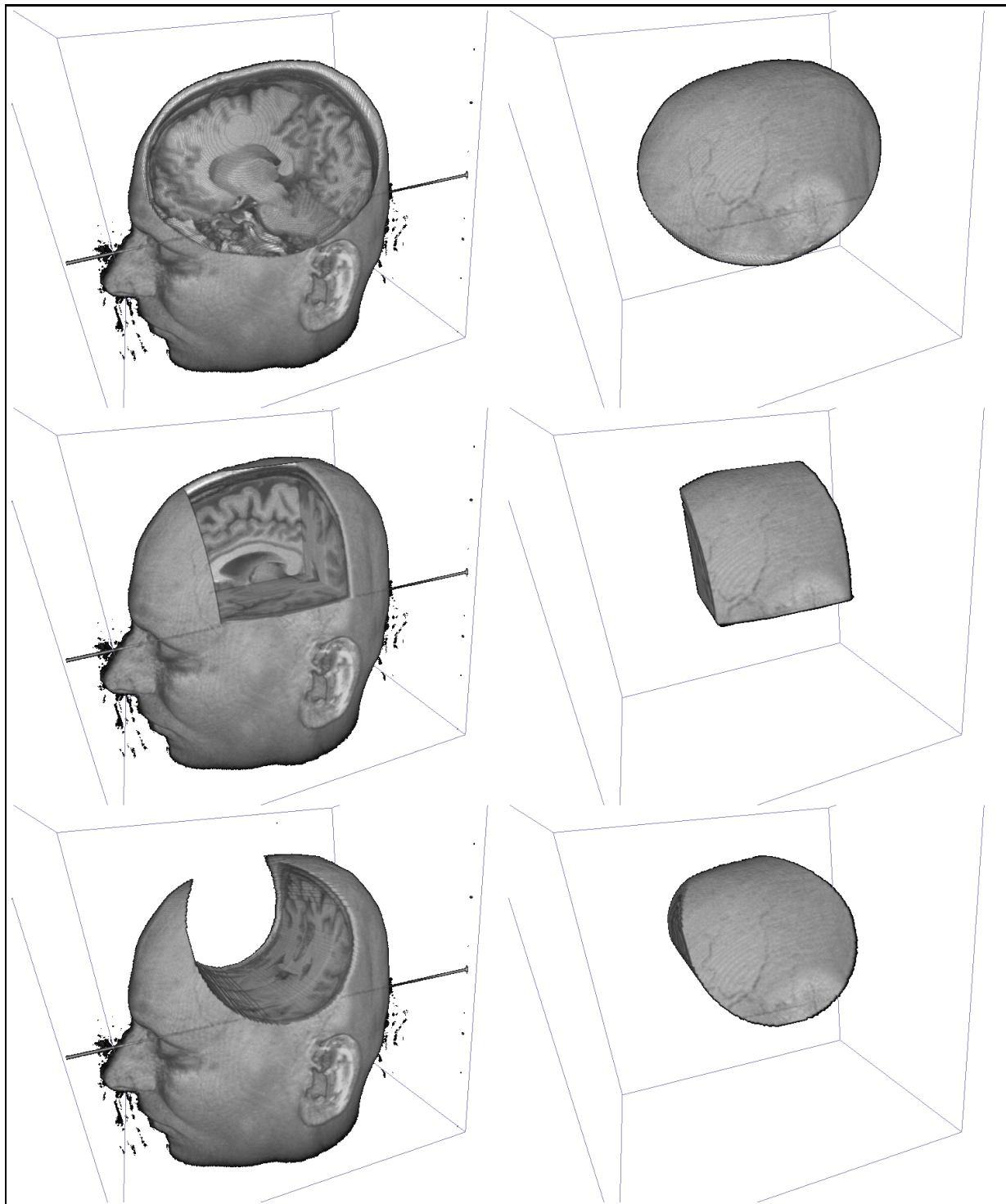


Abbildung 4.11: Multi-Textur-Clipping mit einer sphärischen, kubischen und zylindrischen Clip-Geometrie (links). Invertierung der Clip-Geometrie durch Zuweisen einer anderen Eingabeabbildung in den Combinern (rechts).

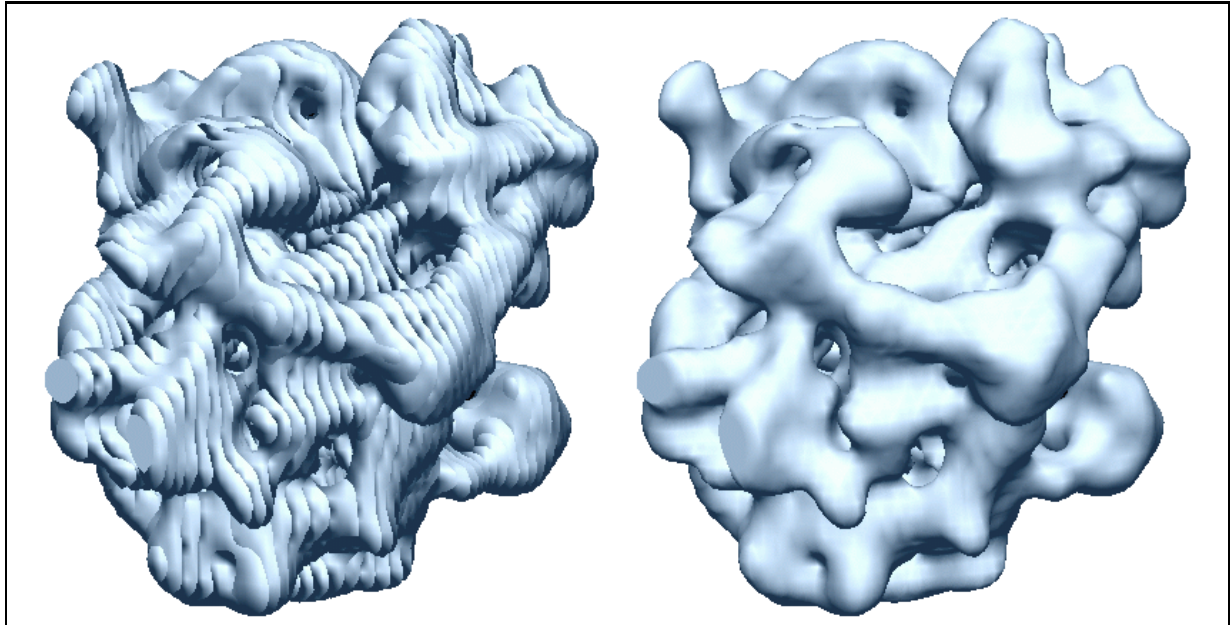


Abbildung 4.12: Visuelle Artefakte, die durch das Fehlen einer trilinearen Interpolation verursacht werden (links), aber durch das Einfügen mehrerer trilinear interpolierter Zwischenschichten entfernt werden können (rechts). Die Abbildung zeigt eine Isofläche des *Escherichia coli* Ribosoms bei einer Auflösung von 18 Å.

winn durch eine höhere Samplingrate anhand einer beleuchteten Isofläche. Weitere Bilder, die die Qualität des Volume-Renderings mit flexibler Rasterisierungshardware demonstrieren, sind auf den Farbseiten in den Abbildungen 9.2, 9.3 und 9.4 dargestellt.

Natürlich sinken durch das Einfügen trilinear interpolierter Schichten die Bildraten, wie aus den Abbildungen 4.14, 4.15 und 4.16 zu ersehen ist. Dort werden die Bildraten der Multi-Textur-Methoden auf der NVIDIA GeForce256 Hardware mit einem ähnlichen, 3D-texturbasierten Algorithmus auf einem SGI Onyx2-Rechner mit BaseReality Graphik verglichen. Bei einer Volumen-Größe von $128 \times 128 \times 64$ und $256 \times 256 \times 128$ erreicht die GeForce-Hardware sowohl bei direktem Volume-Rendering als auch bei Isoflächendarstellung eine höhere Bildrate als die BaseReality Graphik. Bemerkenswert ist außerdem, dass die Isoflächendarstellung bei kleinen Datensätzen eine höhere Bildrate im Vergleich zu semi-transparenter Darstellung erreicht. Dies lässt sich durch das beim Blending notwendige Lesen des Framebuffers erklären, der bei der Isoflächendarstellung nicht notwendig ist. Bei größeren Datensätzen ergeben sich bei der GeForce Hardware niedrigere Bildraten als bei dem High-End-Rechner, da offensichtlich die GeForce256 Hardware eine höhere Rasterisierungsleistung aber eine geringere Speicherbandbreite als die BaseReality Graphik besitzt.

Die parallele Bearbeitung zweier Schichten zur gleichen Zeit bringt zwei entscheidende Vorteile: Einerseits wird die Rasterisierungsleistung der Graphikhardware optimal ausgenutzt, da neueste PC-Graphikhardware zwei Texel pro Rendering-Pipeline parallel bearbeiten kann.

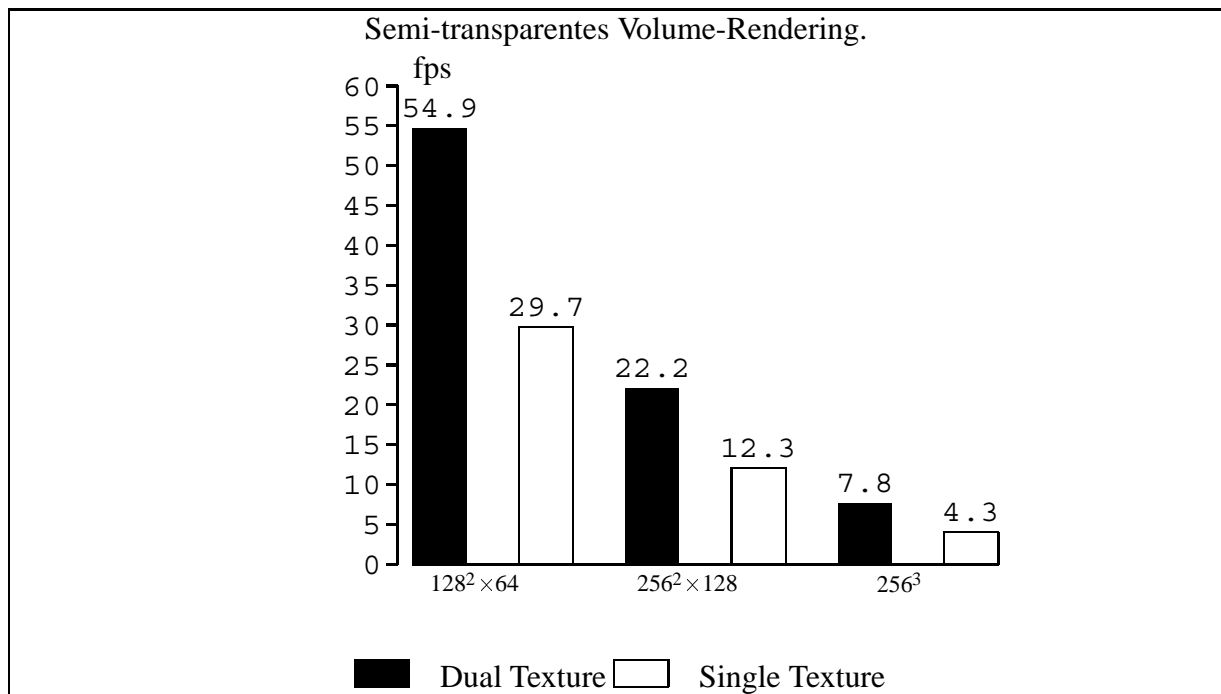


Abbildung 4.13: Vergleich der Bildwiederholraten bei zweifach texturierten Schichtpolygenen (Dual Texture) im Vergleich zu einfach texturierten Schichtpolygenen (Single Texture) für verschieden dimensionierte Volumendatensätze.

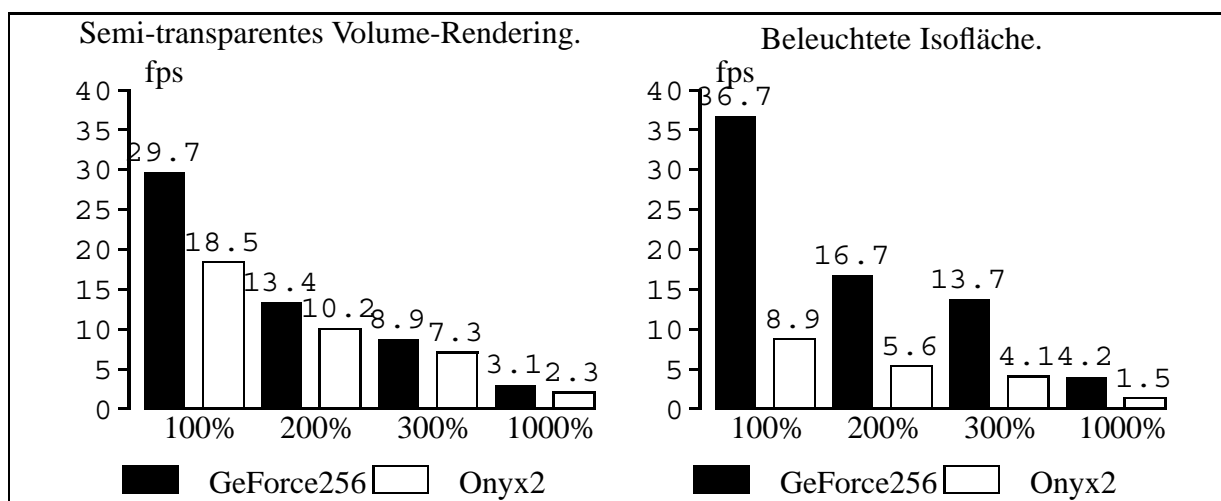
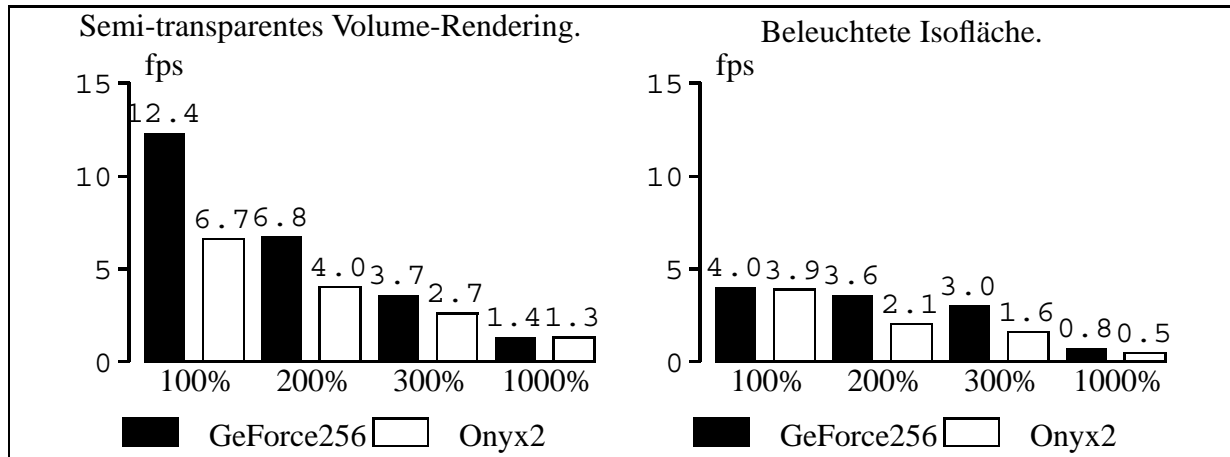
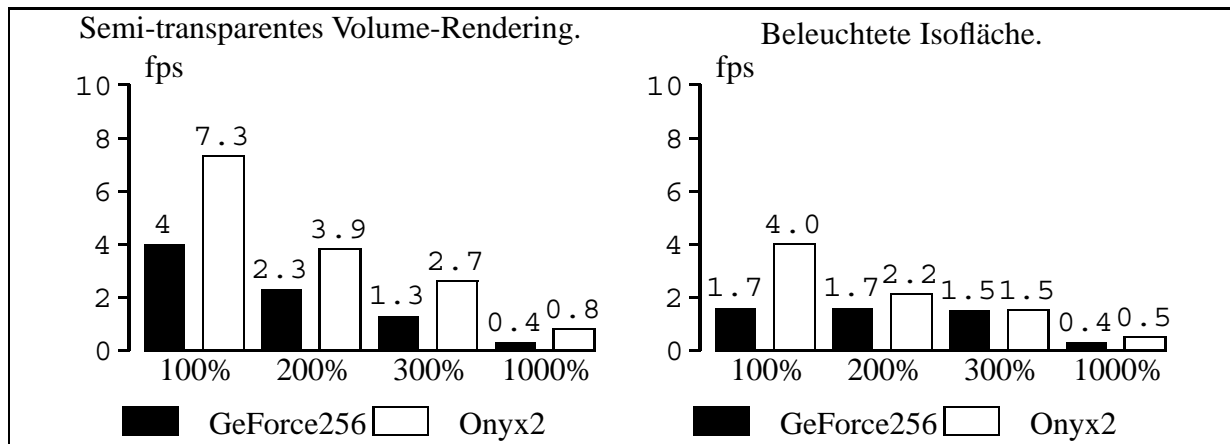


Abbildung 4.14: Bildwiederholraten bei einer Datensatz-Größe von $128 \times 128 \times 64$ Voxeln.

Dies hat eine Verdopplung der Bildwiederholrate zur Folge (siehe Abbildung 4.13), solange der Durchsatz der Pipelines nicht durch die Speicherbandbreite limitiert wird. Der zweite wesentliche Vorteil ergibt sich durch die höhere numerische Genauigkeit der Berechnungen in der

Abbildung 4.15: Bildraten bei einer Datensatz-Größe von $256 \times 256 \times 128$ Voxeln.Abbildung 4.16: Bildraten bei einer Datensatz-Größe $256 \times 256 \times 256$.

Rasterisierungseinheit. Framebuffer-Operationen werden üblicherweise mit 8 Bit Genauigkeit durchgeführt. Dies hat zur Folge, dass sich numerische Fehler bei vielen Blending-Operationen anhäufen. Durch die Verringerung der Anzahl der Blending-Operationen wird aber mit dem hier vorgestellten Ansatz die Bildqualität erhöht, da die Rasterisierungseinheiten intern üblicherweise mit höherer Genauigkeit arbeiten.

Die Kombinierbarkeit der in diesem Abschnitt vorgestellten Verfahren hängt wesentlich von der Anzahl der Multi-Texturen und Combiner-Stufen ab. Zwar lassen sich trilinear interpolierte Schichten für beleuchtete Isoflächen darstellen, andere Kombinationen lassen sich jedoch nicht realisieren. Es ist daher absehbar, dass sich mit zusätzlichen Combiner-Stufen und Multi-Texturen in Zukunft weitere Verbesserungen erzielen lassen.

4.3 Volume-Rendering mit Vorintegration

Die im letzten Abschnitt vorgestellten Algorithmen bringen bereits wesentliche Verbesserungen für die Visualisierung von Volumendaten auf Arbeitsplatzrechnern. Es wurde allerdings auch herausgearbeitet, dass bei weitem noch nicht alle Probleme durch diese Techniken gelöst werden. Wie bereits im Abschnitt 2.3.3 betont wurde, ist eine Post-Klassifikation der Prä-Klassifikation in den meisten Fällen vorzuziehen. Mit der Einführung programmierbarer *Texel-Fetch*-Operationen in aktueller PC-Graphikhardware (NVIDIA GeForce3, ATI Radeon8500) kann bereits eine Post-Klassifikation an die Stelle der Prä-Klassifikation mit der bisher verwendeten `EXT_paletted_texture` OpenGL-Erweiterung treten. Dies wird dadurch erreicht, dass die interpolierten Skalarwerte des Volumens als Texturkoordinaten für weitere, so genannte abhängige *Texel-Fetch*-Operationen (Dependent-Textures), benutzt werden. Die Transferfunktion wird dazu in einer abhängigen, ein-dimensionalen Textur definiert, aus der mit einer `DEPENDENT_AR_TEXTURE_2D_NV` Texture-Fetch-Operation die RGBA-Werte nachgeschlagen werden. Dadurch erfolgt die Klassifikation nach der Filterung der Skalarwerte. In der Medizin werden Messdaten häufig mit einer Quantisierung von 12-Bit ermittelt. Um eine Post-Klassifikation für diese Art der Daten zu erreichen, können die Daten in `LUMINANCE16`-Texturen (16-Bit) abgelegt werden. Zur Klassifikation kann dann eine zweidimensionale *Dependent-Texture* zum Einsatz kommen, die Spalte-für-Spalte und Zeile-für-Zeile mit den Werten aus den Transferfunktionen gefüllt wird. Die Interpolation erfolgt somit in 16 Bit, während für den Nachschlag der Transferfunktionen in der *Dependent-Texture* das niedrigerwertige Byte als x- und das höherwertige Byte in y-Koordinate herangezogen wird.

Post-Klassifikation ist in der Lage, hohe Frequenzen der Transferfunktion auf den Schnittpolygonen abzubilden (siehe Abbildungen 2.12 und 9.11). Allerdings ist zur Repräsentation eines hochfrequenten Funktionsverlaufs zwischen den Schichtpolygonen das Einfügen einer sehr hohen Zahl interpolierter Schichten notwendig. Jedes dieser Schichtpolygone muss in den Framebuffer geblendet werden, wodurch einerseits die Geschwindigkeit deutlich verringert wird und andererseits mit höherer Schichtenzahl die Qualität durch akkumulierte Quantisierungsartefakte leidet. Aus diesem Grund wird im Folgenden ein völlig neues Klassifikationschema vorgestellt, für das der Begriff *Vorintegrierte Klassifikation* geprägt wurde.

4.3.1 Vorintegrierte Klassifikation

Um die oben beschriebenen Einschränkungen zu überwinden, ist es notwendig, die Approximation des Volume-Rendering-Integrals zu verbessern. Tatsächlich wurden bereits viele Verbesserungen vorgeschlagen. Dazu zählen beispielsweise Interpolationsschemata höherer Ordnung[22, 55] und adaptives Supersampling[83, 105].

Allerdings adressieren diese Methoden nicht das Problem hoher Nyquist-Frequenzen von $\tilde{c}(s(\mathbf{x}))$ und $\tau(s(\mathbf{x}))$ durch nicht-lineare Transferfunktionen und bleiben somit an der Oberfläche des Kernproblems. Dagegen ist das Ziel der *Vorintegrierten Klassifikation*[35, 124], die numerische Integration in zwei Integrationen zu unterteilen: jeweils eine für das Skalarfeld $s(\mathbf{x})$ und eine weitere für die Transferfunktionen $\tilde{c}(s)$ und $\tau(s)$, so dass problematische Produkte der Nyquist Frequenzen vermieden werden.

Im ersten Schritt wird das kontinuierliche Skalarfeld $s(\mathbf{x})$ entlang eines Sichtstrahls abgetastet. Zu beachten ist dabei, dass die Nyquist-Frequenz für diese Abtastung nicht durch die Transferfunktion beeinflusst wird. Für die vorintegrierte Klassifikation definieren die abgetasteten Werte ein eindimensionales, stückweise lineares Skalarfeld. Das Volume-Rendering-Integral für dieses Skalarfeld kann sehr einfach durch einen Tabellennachschlag für jedes Strahlsegment ermittelt werden. Die drei Argumente für diesen Tabellennachschlag sind der Skalarwert am Start (front) des Segments $s_f := s(\mathbf{x}(id))$, der Skalarwert am Ende (back) des Segments $s_b := s(\mathbf{x}((i+1)d))$ und die Länge des Segments d (siehe Abbildung 2.13). Genauer gesagt wird die Opazität α_i des i -ten Strahlsegments folgendermaßen approximiert:

$$\begin{aligned}\alpha_i &= 1 - \exp\left(-\int_{id}^{(i+1)d} \tau(s(\mathbf{x}(\lambda))) d\lambda\right) \\ &\approx 1 - \exp\left(-\int_0^1 \tau((1-\omega)s_f + \omega s_b) d\omega\right) d.\end{aligned}\quad (4.4)$$

α_i ist also eine Funktion von s_f , s_b , und d , bzw. von s_f und s_b , wenn die Längen der Segmente gleich sind. Die (assoziierten) Farben \tilde{C}_i werden entsprechend angenähert:

$$\tilde{C}_i \approx \int_0^1 \tilde{c}((1-\omega)s_f + \omega s_b) \exp\left(-\int_0^\omega \tau((1-\omega')s_f + \omega' s_b) d\omega'\right) d\omega. \quad (4.5)$$

Analog zu α_i ist \tilde{C}_i eine Funktion von s_f , s_b , und d . Vorintegrierte Klassifikation approximiert also das Volume-Rendering-Integral durch die Auswertung der folgenden Gleichung (2.2):

$$I \approx \sum_{i=0}^n \tilde{C}_i \prod_{j=0}^{i-1} (1 - \alpha_j)$$

Die Farben \tilde{C}_i und Opazitäten α_i werden entsprechend Gleichung (4.5) und Gleichung (4.4) vorberechnet. Werden nicht-assoziierte Farbtransferfunktionen verwendet, also wenn $\tilde{c}(s)$ durch $\tau(s)c(s)$ ersetzt wird, so wird ebenfalls Gleichung (4.4) für die Abschätzung von α_i benutzt und die folgende Approximation für die assoziierte Farbe \tilde{C}_i^τ :

$$\begin{aligned}\tilde{C}_i^\tau &\approx \int_0^1 \tau((1-\omega)s_f + \omega s_b) c((1-\omega)s_f + \omega s_b) \\ &\quad \times \exp\left(-\int_0^\omega \tau((1-\omega')s_f + \omega' s_b) d\omega'\right) d\omega.\end{aligned}\quad (4.6)$$

In jedem Falle werden aber durch die Vorintegrierte Klassifikation assoziierte Farben berechnet, unabhängig davon, ob eine Transferfunktion für assoziierte Farben $\tilde{c}(s)$ oder für nicht-assoziierte Farben $c(s)$ eingesetzt wird.

Der wesentlich Vorteil, der sich aus der Vorintegrierten Klassifikation ergibt, ist, dass ein kontinuierliches Skalarfeld $s(\mathbf{x})$ unabhängig von jeder nicht-linearen Transferfunktion abgetastet werden kann. Aus diesem Grund hat die Vorintegrierte Klassifikation das Potenzial, sowohl die Genauigkeit (weniger Unterabtastung) als auch die Geschwindigkeit (weniger Abtastungen) eines Volume-Renderers zur gleichen Zeit zu verbessern.

4.3.2 Beschleunigung der Vorintegration

Der primäre Nachteil des oben vorgeschlagenen Klassifikationsschemas ist die zum Nachschlagen der Werte nötige Vorintegration der Tabelle, die die drei Integrationsparameter (Skalarwert vorne s_f , Skalarwert hinten s_b und die Länge des Strahlsegments d) in die vorintegrierten Farben $\tilde{C} = \tilde{C}(s_f, s_b, d)$ und Opazitäten $\alpha = \alpha(s_f, s_b, d)$ abbildet. Da diese Tabelle von der Transferfunktion abhängig ist, müssen bei jeder Änderung der Transferfunktion die Tabellen neu berechnet werden. Dies mag in Spielen oder ähnlichen Unterhaltungsapplikationen keine Rolle spielen, limitiert aber die Interaktivität von Anwendungen im Umfeld der wissenschaftlichen Visualisierung, da diese oft von der interaktiven Manipulation benutzerspezifischer Transferfunktionen abhängen. Deshalb werden im Folgenden drei Optimierungen zur Beschleunigung der Vorintegration vorgeschlagen.

Zunächst ist es unter bestimmten Umständen möglich die Dimensionalität der Tabellen von drei auf zwei zu reduzieren, indem eine konstante Länge des Strahlsegments d angenommen wird. Dies trifft offensichtlich für den Raycasting-Algorithmus mit äquidistanten Abtastabständen entlang der Strahlen zu. Ebenfalls zutreffend ist dies für 3D-texturbasierte Volumenvisualisierungsalgorithmen mit orthographischer Projektion; und zumindest für die meisten perspektivischen Projektionen ist es eine gute Approximation. Zur Gewährleistung einer konstanten Strahlsegmentlänge können in jedem Fall Kugelschalen (Shells) eingesetzt werden[54, 88]. Die konstante Strahlsegmentlänge ist weniger zutreffend für 2D-texturbasierte Algorithmen mit objekt-parallelen Schichten, wie sie in Abschnitt 2.3.5 diskutiert wurden. Obwohl dabei sehr unterschiedliche Längen auftreten, kann die komplizierte Abhängigkeit von der Strahlsegmentlänge mit einer linearen Abhängigkeit, wie in [124] vorgeschlagen, approximiert werden. Aus diesen Überlegungen folgt, dass es in den meisten Fällen genügt, die vorintegrierten Tabellen nur für eine Strahlsegmentlänge zu berechnen.

Die zweite Optimierung folgt daraus, dass eine lokale Modifikation der Transferfunktion keineswegs eine Neuberechnung der ganzen Tabelle erfordert. Vielmehr müssen bei einer Änderung eines Eintrags in der Transferfunktion $\tilde{c}(s)$ bzw. $\tau(s)$ nur die Werte $\tilde{C}(s_f, s_b, d)$ und $\alpha(s_f, s_b, d)$ mit $s_f \leq s \leq s_b$ oder $s_f \geq s \geq s_b$ erneut ermittelt werden. Also ist im schlechtesten Falle eine Neuberechnung ungefähr die Hälfte der Tabelle notwendig.

Schließlich ergibt sich eine außerordentliche Beschleunigung der Auswertung des Integrals in den Gleichungen (4.4), (4.5) und (4.6), indem Integralfunktionen für $\tau(s)$, $\tilde{c}(s)$ und $\tau(s)c(s)$, eingesetzt werden. Genauer gesagt kann Gleichung (4.4) für $\alpha_i = \alpha(s_f, s_b, d)$ als

$$\alpha(s_f, s_b, d) \approx 1 - \exp\left(-\int_0^1 \tau((1-\omega)s_f + \omega s_b) d \omega\right) \quad (4.7)$$

$$= 1 - \exp\left(-\frac{d}{s_b - s_f} \int_{s_b}^{s_f} \tau(s) ds\right) \quad (4.8)$$

$$\alpha(s_f, s_b, d) \approx 1 - \exp\left(-\frac{d}{s_b - s_f} (T(s_b) - T(s_f))\right) \quad (4.9)$$

mit der Integralfunktion $T(s) := \int_0^s \tau(s) ds$ umgeschrieben werden, die in der Praxis einfach berechnet werden kann, da die Skalarwerte üblicherweise quantisiert sind.

Gleichung (4.5) kann für $\tilde{C}_i = \tilde{C}(s_f, s_b, d)$ mit der Integralfunktion $K(s) := \int_0^s \tilde{c}(s) ds$ analog approximiert werden:

$$\tilde{C}(s_f, s_b, d) \approx \int_0^1 \tilde{c}((1-\omega)s_f + \omega s_b) d \omega \quad (4.10)$$

$$= \frac{d}{s_b - s_f} \int_{s_f}^{s_b} \tilde{c}(s) ds \quad (4.11)$$

$$\tilde{C}(s_f, s_b, d) \approx \frac{d}{s_b - s_f} (K(s_b) - K(s_f)). \quad (4.12)$$

Allerdings muss dazu die Selbstabschwächung innerhalb eines Strahlsegments vernachlässigt werden. Wie bereits oben erwähnt ist dies eine übliche Approximation bei der Post-Klassifikation und für kleine Produkte $\tau(s)d$ durchaus gerechtfertigt.

Für nicht-assoziierte Farbtransferfunktionen $c(s)$ wird die Gleichung (4.6) durch

$$\tilde{C}^\tau(s_f, s_b, d) \approx \int_0^1 \tau((1-\omega)s_f + \omega s_b) \quad (4.13)$$

$$\times c((1-\omega)s_f + \omega s_b) d \omega \quad (4.14)$$

$$= \frac{d}{s_b - s_f} \int_{s_f}^{s_b} \tau(s) c(s) ds \quad (4.15)$$

$$\tilde{C}^\tau(s_f, s_b, d) \approx \frac{d}{s_b - s_f} (K^\tau(s_b) - K^\tau(s_f)) \quad (4.16)$$

approximiert, mit $K^\tau(s) := \int_0^s \tau(s) c(s) ds$.

Anstelle der numerischen Integration der Gleichungen (4.4), (4.5) und (4.6) können deshalb nunmehr für jede Kombination von s_f , s_b und d , die Integralfunktionen $T(s)$, $K(s)$ oder $K^\tau(s)$ berechnet und zur Auswertung der Farben und Opazitäten mit den Gleichungen (4.9), (4.12), oder (4.16) ohne weitere Integration eingesetzt werden.

Alle diese Maßnahmen zur Beschleunigung der Vorintegration erlauben letztlich eine interaktive Manipulation der Transferfunktion für eine beschränkte Menge quantisierter Skalarwerte. Ergebnisse, die diese Behauptung stützen, werden in Abschnitt 4.3.10 behandelt.

4.3.3 Texturbasiertes Vorintegriertes Volume-Rendering

Basierend auf der Beschreibung der Vorintegrierten Klassifikation in Abschnitt 4.3.1 wird nun ein neuer texturbasierter Algorithmus unter Verwendung der Vorintegration vorgestellt. Dieser benutzt abhängige Texturzugriffe, also die Möglichkeit, Fragmentwerte als Texturkoordination zu verwenden, die dann für weitere Texturzugriffe eingesetzt werden (siehe Abschnitt 2.2.4). Wie bereits erwähnt kann mit abhängigen Texturzugriffen Post-Klassifikation mittels einer ein-dimensionalen Lookup-Textur realisiert werden. Im Folgenden werden aber die abhängigen Texturzugriffe zum Nachschlag vorintegrierter Strahlsegment-Werte eingesetzt.

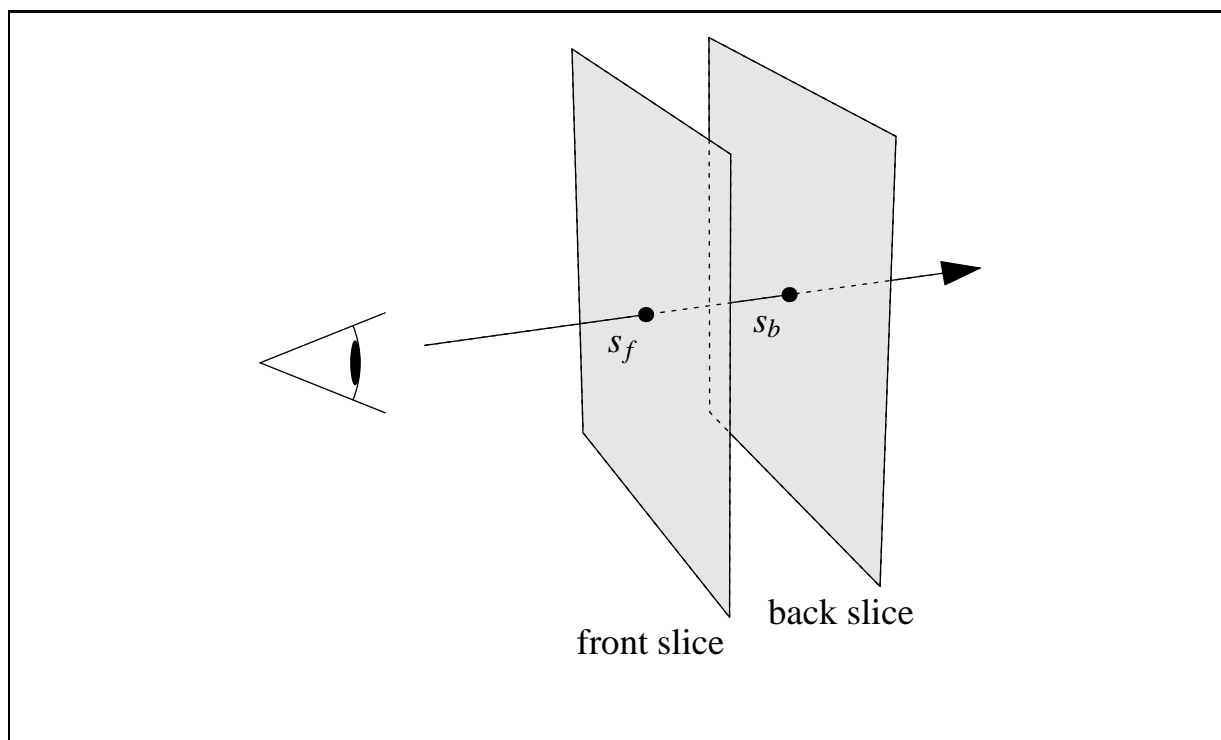


Abbildung 4.17: Abbildung eines *Slabs* des Volumens. Dieser wird durch zwei aufeinanderfolgende Schichten des Volumens begrenzt. Der Skalarwert der, in Bezug auf einen Sichtstrahl, vorderen bzw. hinteren Schicht, wird als s_f bzw. s_b bezeichnet.

Dieser Hardware-beschleunigte Algorithmus kann sowohl in Kombination mit 3D-Texturen und viewport-parallelen Schnittebenen als auch mit 2D-Texturen und drei objekt-parallelen Schichtenstapeln realisiert werden. Die Volumentexturen enthalten wie bei der Post-Klassifikation die Skalarwerte des Volumens. Da jedes Paar aneinandergrenzender Schichten einen Slab des Volumen repräsentiert (siehe Abbildung 4.17), werden nun anhand von Multi-Texturen die Texturen zweier aneinandergrenzender Schichten auf ein Schnittpolygon projiziert. Dazu kann entweder die, bezüglich des Augpunktes, vordere auf die hintere Schicht projiziert werden, oder vice versa. Dadurch kommen entlang jedes Sehstrahls die Skalarwerte der vorderen und hinteren Schicht während der Rasterisierung eines Polygons aufeinander zu liegen. Somit kann auf diese Skalarwerte mit Texturoperationen einfach zugegriffen werden. Die beiden Skalarwerte werden als Texturkoordinaten für einen dritten Texturzugriff in eine zweidimensionale Textur verwendet, die vorintegrierte Farben und Opazitäten enthält. Da dieser Zugriff von zwei vorausgegangenen Texturoperationen abhängt, wird diese Operation auch abhängiger Texturzugriff (*Dependent-Texture-Fetch*) genannt.

Die Opazitäten der *Dependent-Texture* werden nach Gleichung (4.4) berechnet, während die Farben nach Gleichung (4.5) ermittelt werden, wenn die Transferfunktion assoziierte Farben $\tilde{c}(s)$ enthält. Entsprechend werden die Farben nach Gleichung (4.6) für nicht-assozierte Farben

$c(s)$ berechnet. In beiden Fällen wird aber die Compositing-Gleichung (2.3) für das Blending eingesetzt, da die Dependent-Texture immer assoziierte Farben enthält.

Offensichtlich ist eine hardwarebeschleunigte Implementierung dieser Algorithmen von indirekten, relativ komplizierten Texturzugriffen abhängig. Aber genau solche Texturzugriffe haben in neueste flexibler Graphikhardware Einzug gefunden, so dass eine effiziente Implementierung möglich ist. Im Folgenden wird die Umsetzung des Algorithmus auf den NVIDIA GeForce3 Chip auf Basis der OpenGL-Erweiterungen `NV_register_combiners` und `NV_texture_shader` bzw. `NV_texture_shader2` beschrieben (siehe Abschnitt 2.2.4). Eine entsprechende Implementierung wurde auch auf einer alternativen Hardware-Architektur des Herstellers ATI, der Radeon8500, realisiert. Dazu wurde die OpenGL-Erweiterung `ATI_fragment_shader` eingesetzt.

Vorintegriertes, texturbasiertes Volume-Rendering lässt sich in drei Hauptschritte unterteilen: Zunächst werden zwei aneinandergrenzende Schichten aus Sicht des Augpunktes aufeinander projiziert, um diese Schichten auf ein einziges Schnittpolygon in Form von Multi-Texturen aufzubringen. Die Projektion wird, ähnlich wie in Abschnitt 4.2.2, durch Adaption von Texturkoordinaten erreicht (eventuell durch projektive Texturkoordinaten-Generierung). Dadurch kommen immer zwei Texel entlang jedes Sichtstrahls, eines der vorderen und eines der hinteren Schicht, aufeinander zu liegen. Der zweite Hauptschritt besteht darin, die so aufeinander projizierten Texel mittels der Erweiterung `NV_texture_shader` bzw. `NV_texture_shader2` während der Rasterisierung nachzuschlagen und dann als Texturkoordinaten für einen abhängigen Texturzugriff auf eine 2D-Texture zu benutzen. Diese enthält vorintegrierte Werte für jede Kombination von vorderem Texelwert s_f und hinterem Texelwert s_b . Alternativ kann diese 2D-Texture auch Farbe, Transparenz und Interpolationswerte für eine Isoflächendarstellung enthalten. Dazu müssen diese Werte gesetzt sein, wenn der Isowert zwischen dem vorderen und hinteren Texelwert liegt.

Für direktes Volume-Rendering ohne Beleuchtung werden die Texturen des Volumens im OpenGL Texturformat `GL_LUMINANCE8` definiert, während für direktes Volume-Rendering beleuchteter Volumina und die Isoflächendarstellung `GL_RGBA`-Texturen eingesetzt werden, die vorberechnete Gradienten und die Skalarwerte des Volumens enthalten. Bei statischer Beleuchtung ist es auch möglich durch den Einsatz von `LUMINANCE_ALPHA`-Texturen mit vorberechneten Skalarprodukten und Skalarwerten die Hälfte des Texturspeichers gegenüber `RGBA`-Texturen einzusparen.

Im letzten Schritt werden schließlich falls nötig die Gradienten in den Register-Combinern interpoliert und die Beleuchtungsberechnung durchgeführt. Die folgenden Abschnitte betrachten die Einzelschritte nochmals im Detail.

4.3.4 Projektion

Anstelle eines Schicht-für-Schicht-Ansatzes, wie er sonst bei texturbasierten Algorithmen üblich ist, wird beim vorintegrierten Volume-Rendering ein Schichtzwischenraum (Slab) nach dem anderen von hinten nach vorne mit Hilfe des *Over-Operators* im Framebuffer überblendet. Im Grunde stellt dies eine Verallgemeinerung des Ansatzes zum beschleunigten Rendering von Volumendaten mit Multi-Texturen dar. Anstelle eines einfachen Überblendens von \tilde{C}_i und \tilde{C}_{i+1} wird hier aber für jedes Sichtstrahlsegment zwischen den beiden Schichten ein vorintegriertes

Strahlsegment gerendert. Dazu wird für jedes Paar aneinandergrenzender Schichten ein einziges Schnittpolygon mit den dazugehörigen Texturschichten als Multi-Texturen gebunden. Damit während der Rasterisierung auf Texel entlang jedes Sichtstrahls von der vorderen und hinteren Texturschicht zugegriffen werden kann, müssen diese aufeinander projiziert werden. Dies erfolgt durch Anpassung von Texturkoordinaten analog zu Abschnitt 4.2.2. Dabei spielt es keine Rolle, ob die vordere Schicht auf die hintere Schicht projiziert wird oder umgekehrt. Eventuell kann auch die Größe des gezeichneten Schnittpolygons an die Ausmaße der projizierten und der nicht-projizierten Schicht angepasst werden. Die Projektion erfolgt wieder analog zu der bereits in Abschnitt 4.2.2 vorgestellten Vorgehensweise. Für die möglichen Projektionen bei objekt-parallelen und viewport-parallelen Schichten sei erneut auf Abbildung 4.7 verwiesen.

4.3.5 Texel-Fetch

Für jedes Fragment kommen durch die obige Projektion aneinandergrenzender Schichten Texel der vorderen und hinteren Schicht aufeinander zu liegen. Auf diese wird mit ihren gegebenen Texturkoordinaten zugegriffen, so dass die Werte s_f und s_b pro Fragment vorliegen. In einem abhängigen Texturzugriff werden diese nun als Texturkoordinaten (s, t) genutzt. Da die Texturkoordinaten aus zwei unterschiedlichen Multi-Texturen kommen, kann leider die in den Texture-Shadern dazu vorgesehene Operation nicht benutzt werden. Diese setzt nämlich die Grün- oder Blau-Komponenten (oder alternativ Rot- und Alpha-Komponenten) eines vorausgegangenen Texturnachschlags als (s, t) Koordinaten für einen weiteren Nachschlag in einer 2D-Textur voraus. Zur Umgehung dieser Problematik kann stattdessen aber eine Skalarprodukt-Operation verwendet werden, die das Skalarprodukt zwischen den Texturkoordinaten der aktuellen Shader-Stufe (s, t, r) und dem RGB-Vektor einer vorausgegangenen Textur-Operation berechnet. Die Ergebnisse zweier dieser Operationen werden als Texturkoordinaten für den abhängigen Texturzugriff auf die vorintegrierten Strahlsegment-Werte eingesetzt (siehe Abbildung 4.18).

Das Skalarprodukt ist dabei ausschließlich nötig, um die Werte s_f und s_b aus den RGB-Komponenten der Texturen zu extrahieren. Da die Texture-Shader-Einheit ausschließlich eine DOT3-Skalarprodukt definiert, können die Skalarwerte nicht wie üblich in der Alpha-Komponente der Textur abgelegt werden. Stattdessen werden die Skalarwerte beispielsweise in der Rot-Komponente der Textur gespeichert und durch das Skalarprodukt mit der konstanten Texturkoordinate $\mathbf{v} = (1, 0, 0)^T$ extrahiert. Die Texture-Shader-Erweiterung erlaubt für jede der Skalarproduktoperationen die Definition einer vorhergehenden Shader-Operation über die Texturumgebung `GL_PREVIOUS_TEXTURE_INPUT_NV`, mit deren Ergebnis das Skalarprodukt ermittelt wird. In der Texture-Shader-Konfiguration aus Abbildung 4.18 benutzt die erste Skalarprodukt-Operation das RGB-Ergebnis der Shader-Stufe 1 und die zugehörigen Texturkoordinaten der Stufe zur Berechnung des Skalarprodukts, während die zweite Skalarprodukt-Operation das RGB-Ergebnis der Shader-Stufe 0 und die zugehörigen Texturkoordinaten der Stufe als Parameter nutzt. Das Ergebnis der beiden Skalarproduktoperation wird als Texturkoordinaten für den abhängigen Texturzugriff in die 2D-Textur mit den vorintegrierten Strahlsegmenten benutzt und als Ausgabe der Shader-Stufe weitergeleitet.

Für das direkte Volume-Rendering ohne Beleuchtung wird das Ergebnis der letzten Texture-Shader-Stufe ohne weitere Bearbeitung durch die Register-Combiner geleitet, die sich in der

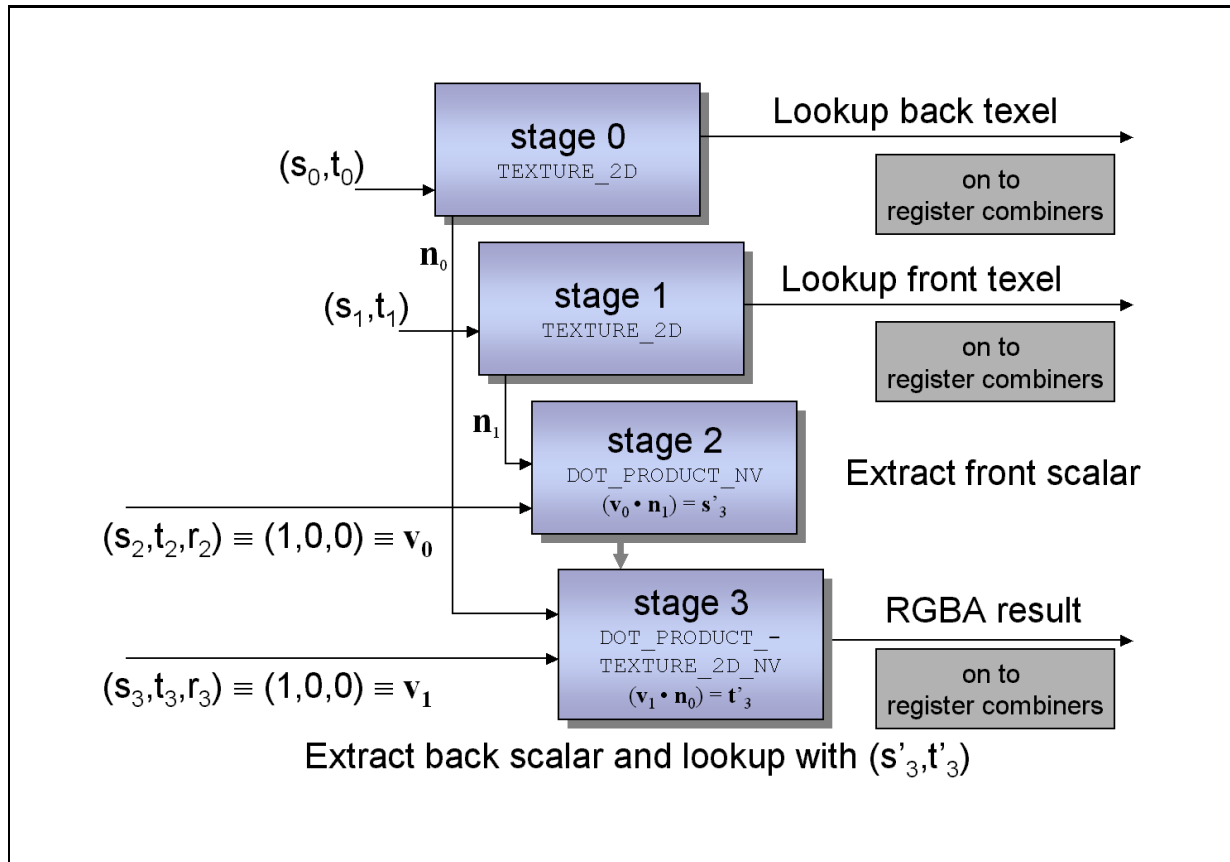


Abbildung 4.18: Texture-Shader-Einstellungen für einen abhängigen Texturzugriff mit Texturkoordinaten aus zwei vorherigen Texturoperationen. Stage 0 und 1 sind für den Nachschlag der Skalarwerte aus der hinteren bzw. vorderen Schicht zuständig (alternativ können an diesen Stellen auch Nachschläge aus 3D-Texturen verwendet werden). Die im Rot-Kanal gespeicherten Skalarwerte werden mittels Skalarprodukten in Stage 2 und 3 extrahiert und schließlich in Stage 3 als Texturkoordinaten für einen abhängigen Texturzugriff eingesetzt.

Rendering-Pipeline an die Texture-Shader-Einheit anschließen. Das Ergebnis wird mit Hilfe der OpenGL-Blending-Funktion `glBlendFunc(GL_ONE, GL_ONE_MINUS_SRC_ALPHA)`, die der Blending-Funktion aus Gleichung (2.3) für assoziierte Farben entspricht, in den Framebuffer geblendet.

4.3.6 Gradienteninterpolation

In [124] wurde ein Verfahren vorgestellt, das mittels Vorintegrierter Klassifikation eine beliebige Anzahl von Isoflächen darstellt. Die Grundidee ist, jedes Strahlsegment mit der Farbe derjenigen Isofläche einzufärben, die das Strahlstück als erstes schneidet. Dazu kann der gleiche Ansatz wie oben genutzt werden, wobei eine spezielle *Dependent-Texture* zum Einsatz kommt. Diese hat

an denjenigen Stellen nicht-transparente Einträge, an denen ein oder mehrere Isowerte zwischen s_f und s_b liegen. Für mehrere semi-transparente Isoflächen innerhalb eines Strahlsegments wird eine Farbe und Opazität gespeichert, die sich aus dem Compositing der Isoflächenfarben innerhalb des Strahlsegments von hinten nach vorne ergibt. Im Gegensatz zu dem in Abschnitt 4.2.3 beschriebenen Verfahren zur Darstellung von beleuchteten Isoflächen unter Zuhilfenahme des OpenGL Alpha-Tests, lassen sich mit diesem Verfahren beidseitig beleuchtete Isoflächen darstellen, wobei auch die Blickrichtung durch die Isofläche klassifiziert werden kann. Die Anzahl der Isoflächen, deren Farbe und Transparenz wird somit rein durch die Wahl der Dependent-Texture beeinflusst.

In den bisher vorgestellten Verfahren zu Beleuchtungsberechnung wurden üblicherweise RGBA-Texturen eingesetzt, die den vorberechneten Gradienten in den RGB-Komponenten und den Voxelwert in der Alpha-Komponente enthalten. Da wegen des verwendeten Skalarprodukts der Voxelwert in der Rot-Komponente der Textur gespeichert werden muss, wird im Gegenzug die erste Komponente des Gradienten im Alpha-Kanal abgelegt. In den Register-Combinern wird der Gradient zur Beleuchtungsberechnung wieder zusammengesetzt, d.h. der Alpha-Kanal wird zurück in den Rot-Kanal geleitet. Dies wird in den ersten beiden Combiner-Stufen der Register-Combiner für den Gradienten des vorderen und hinteren Voxels durch jeweils eine $(AB + CD)$ -Operation erreicht (siehe Abbildung 4.19). Diese beiden Gradienten werden danach an der Stelle der Isofläche linear interpoliert. Der Gewichtungsfaktor IP des Gradienten der hinteren Schicht ergibt sich zu $IP = (s_{iso} - s_f)/(s_b - s_f)$; entsprechend ergibt sich der Gewichtungsfaktor des Gradienten der vorderen Schicht zu $1 - IP$. Theoretisch ließen sich diese Gewichtungsfaktoren aus s_{iso} , s_f und s_b während der Rasterisierung errechnen. Leider wäre dazu eine Division notwendig, die in den Register-Combinern nicht unterstützt wird, da Divisionen in Hardware „teuer“ zu realisieren sind. Deshalb muss der Gewichtungsfaktor IP für jede Kombination von s_f und s_b vorberechnet werden und während der Rasterisierung mit einem abhängigen Texturzugriff in einer Dependent-Texture nachgeschlagen werden. Leider erlaubt die Texture-Shader-Erweiterung nur vier Operationen, die bereits durch die vorigen Operationen verbraucht sind. Aus diesem Grund wird der Gewichtungsfaktor in der ersten (und einzigen) Dependent-Texture abgelegt.

Dazu kann als erste Möglichkeit der Alpha-Kanal der Dependent-Texture zur Speicherung des Gewichtungsfaktors genutzt werden (R, G, B, IP) (siehe Abbildung 4.19), wodurch zwangsläufig der Freiheitsgrad verloren geht, für jede Seite der Isofläche die Opazität frei definieren zu können. Es ist so durch eine entsprechende Konfiguration nur noch ein globaler Transparenzwert für alle Isoflächen realisierbar. Für Strahlsegmente, die eine Isofläche schneiden, wird der Gewichtungsfaktor im Bereich von 128 bis 255 (7 Bit) gespeichert, während Strahlsegmente, die keine Isofläche schneiden, ein Gewichtungsfaktor von 0 zugewiesen wird. Dies erlaubt es zunächst, die Gewichtungsfaktoren für die Interpolation der Gradienten zu nutzen und anschließend durch eine Skalierung der Gewichtungsfaktoren opake Isoflächenstrahlsegmente zu gewinnen. Danach kann nochmals eine Multiplikation mit einem konstanten Opazitätswert realisiert werden, der die globale Transparenz aller Isoflächen bestimmt.

Eine weitere Möglichkeit ergibt sich durch die Nutzung des Blau-Kanals der Dependent-Texture (R, G, IP, A) für den Gewichtungsfaktor (siehe Abbildung 4.20). Dadurch kann nun die Transparenz jeder Isofläche frei gewählt werden, allerdings wird der Blau-Kanal mit einem konstanten Wert für alle Isoflächen in den Combinern gefüllt. Bei diesem Verfahren kann der

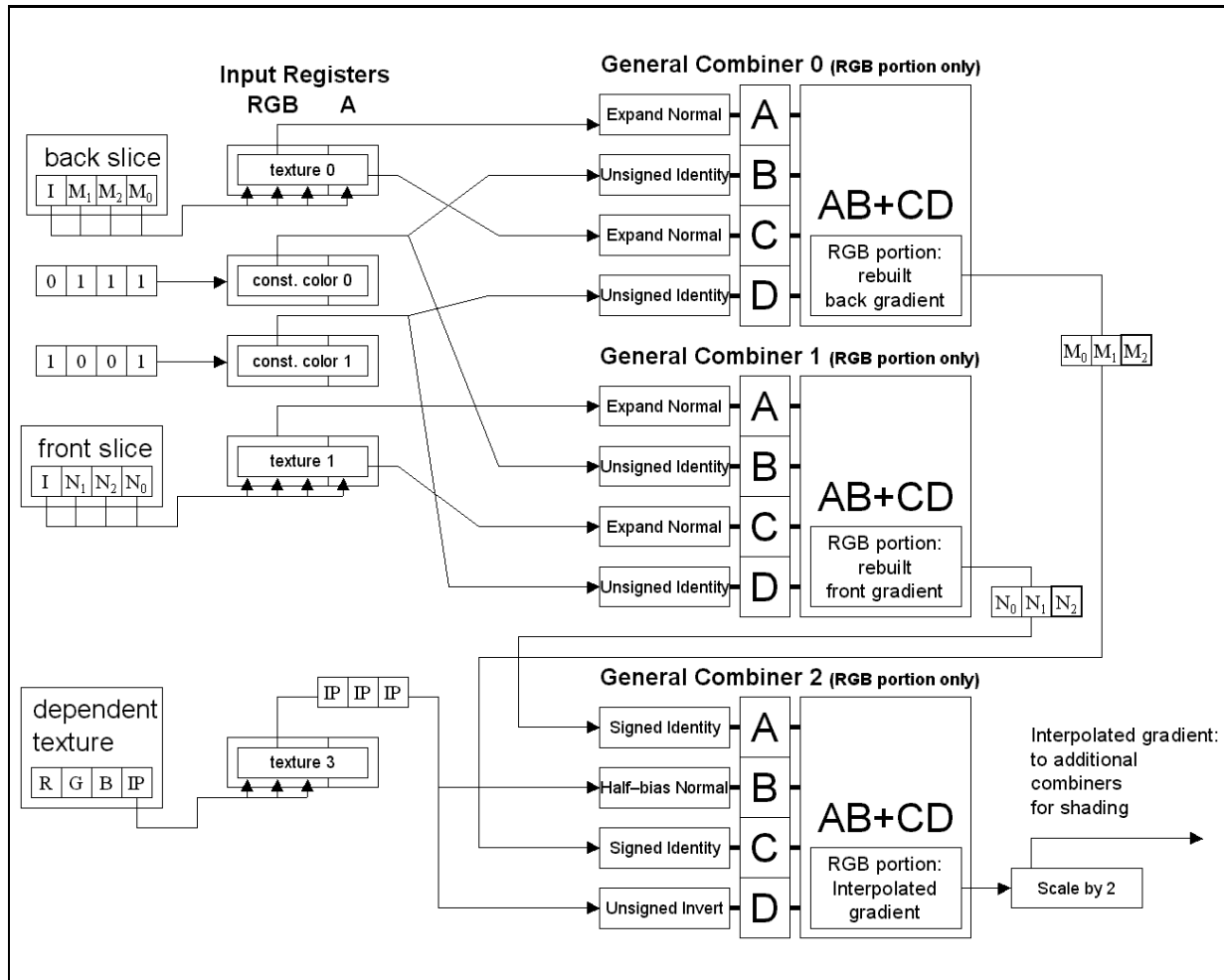


Abbildung 4.19: Register-Combiner-Konfiguration zur Gradienteninterpolation mit Gewichtungsfaktoren im Alpha-Kanal. General-Combiner 0 und General-Combiner 1 setzen den Gradienten für die vordere und hintere Schicht wieder zusammen, während General-Combiner 2 die Interpolation durchführt. Dabei steht M für den Gradienten der hinteren Schicht und N für den Gradienten der vorderen Schicht. Da der Gewichtungsfaktor aus der Dependent-Texture für die Interpolation im Bereich 0.5 bis 1.0 liegt, wird durch entsprechende Eingabeabbildungen mittels `GL_HALF_BIAS_NORMAL_NV` und `GL_UNSIGNED_INVERT_NV` der Faktor für die Interpolation auf 0 bis 0.5 und 0.5 bis 0 abgebildet. Nach der Interpolation wird zur Kompensation eine Skalierung des Ergebnisses mit 2 durchgeführt.

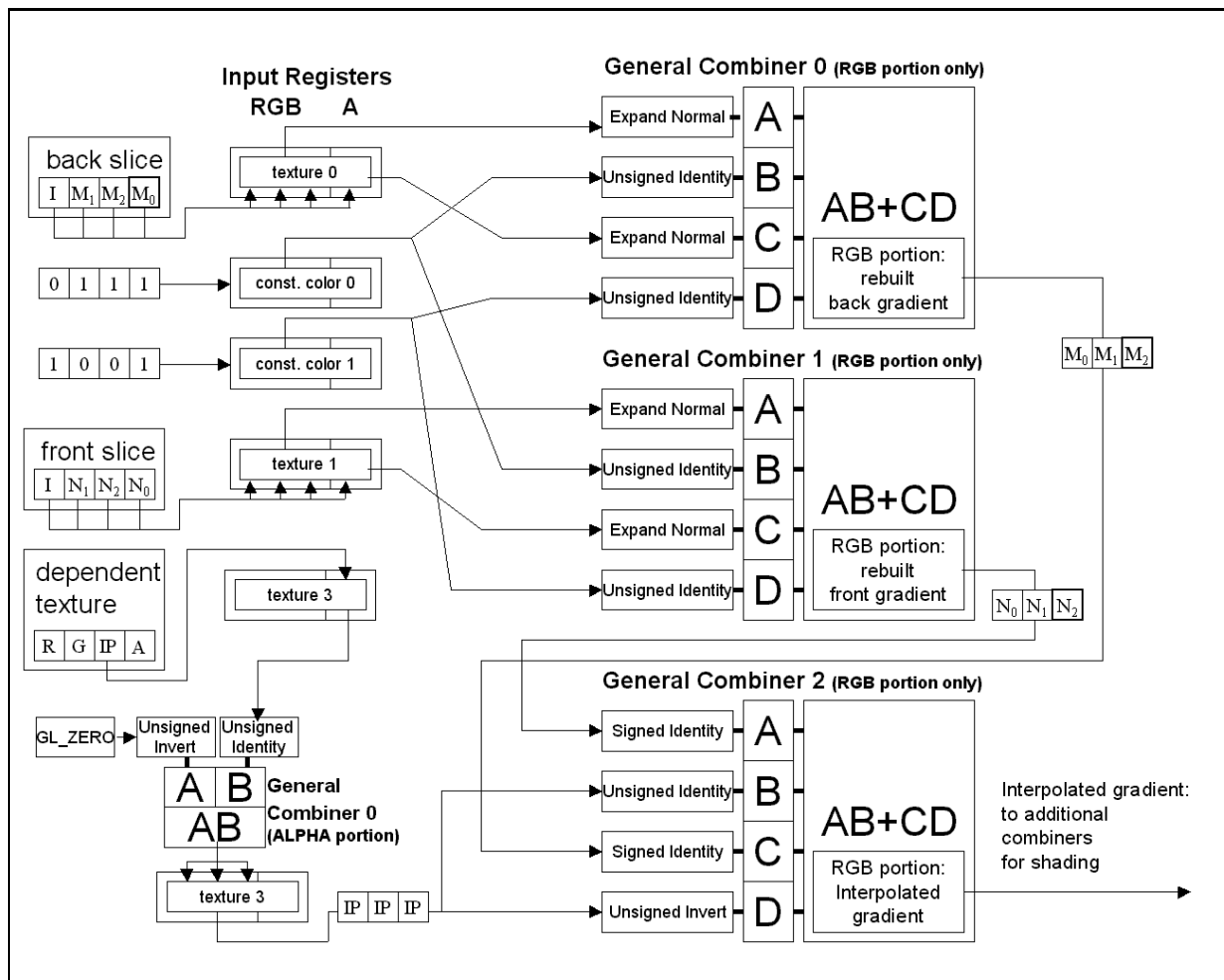


Abbildung 4.20: Register-Combiner-Konfiguration zur Gradienteninterpolation mit Gewichtungsfaktoren im Blau-Kanal. General-Combiner 0 und General-Combiner 1 setzen den Gradienten für die vordere und hintere Schicht wieder zusammen, während General-Combiner 2 die Interpolation durchführt. Dabei steht M für den Gradienten der hinteren Schicht und N für den Gradienten der vorderen Schicht. Der Gewichtungsfaktor aus der Dependent-Texture für die Interpolation wird im Alpha-Teil von General-Combiner 0 zunächst auf die RGB-Farbkanäle verteilt und anschließend in General-Combiner 2 zur Interpolation der Gradienten eingesetzt.

Gewichtungsfaktor kann mit 8 Bit Genauigkeit gespeichert werden.

In jedem der beiden hier vorgestellten Fälle gehen Freiheitsgrade verloren. Dies ist aber nur auf die Beschränkung auf vier Texturnachschräge zurückzuführen. Neueste Graphikhardware erlaubt eine größere Zahl von Texturnachschräge, so dass zur Speicherung der Gewichtungsfaktoren eine getrennte Dependent-Texture eingesetzt werden kann.

4.3.7 Beleuchtung

Nach der Interpolation des Gradienten in den ersten drei Combiner-Stufen können die restlichen fünf Combiner-Stufen für die Beleuchtungsberechnung eingesetzt werden. Dazu wird das Phong-Beleuchtungsmodell verwendet, wobei ein Halfway-Vektor zur Ermittlung des spekularen Anteils benutzt wird:

$$I = I_a + I_d C(\mathbf{n} \cdot \mathbf{l}) + I_s (\mathbf{n} \cdot \mathbf{h})^{16}. \quad (4.17)$$

In der Gleichung steht \mathbf{n} für den interpolierten Gradienten aus den ersten drei Combiner-Stufen und C für die Farbe der Isofläche aus dem RGB-Eintrag in der abhängigen Textur. Die Werte \mathbf{l} für die Lichtquellenrichtung, \mathbf{h} für den Halfway-Vektor und I_a , I_d und I_s für die Koeffizienten des ambienten, diffusen und spekularen Beleuchtungsanteils werden als Konstante in die Register-Combiner-Einheit geleitet. Die Ergebnisse dieser Beleuchtungsberechnung für Isoflächen sind auf den Farbseiten in den Abbildungen 9.6 und 9.7 zu sehen.

Für die Beleuchtungsberechnung innerhalb eines semi-transparenten Volumens (*Volume-Shading*) kommt wieder eine Dependent-Texture mit vorintegrierten Strahlsegmenten zum Einsatz. Für die Beleuchtungsberechnung werden wie bei der Isoflächendarstellung vorberechnete Gradienten in der Volumentextur gespeichert, die während des Renderings zwischen der vorderen und hinteren Schicht gemittelt werden. Es ist also nicht notwendig, Gewichtungsfaktoren in der Dependent-Texture zu speichern. Diese enthält ausschließlich vorintegrierte Opazitäten α_i und Farben \tilde{C}_i . Letztere kommen beim diffusen Beleuchtungsterm zum Einsatz:

$$I = I_d \tilde{C}_i (\mathbf{n} \cdot \mathbf{l}) + I_s (\mathbf{n} \cdot \mathbf{h})^{16} \quad (4.18)$$

Dabei steht wiederum \mathbf{n} für den gemittelten Gradienten, \mathbf{l} für die Lichtquellenrichtung, \mathbf{h} für den Halfway-Vektor und I_d und I_s für die Koeffizienten des diffusen und spekularen Beleuchtungsanteils.

Der Volume-Shading-Ansatz erlaubt auch das Mischen von beleuchteten Isoflächen mit semi-transparenter Volumendarstellung. Dazu wird für jede Isofläche in der Transferfunktion eine Spitze (*Peak*) im Alpha-Kanal eingetragen. Durch die Vorintegration der Strahlsegmente im Volume-Shading-Modus ergibt sich eine Mischung aus beleuchteten Isoflächen und semi-transparenter Darstellung. Beispiele beleuchteter semi-transparenter Volumen sind auf den Farbseiten in Abbildung 9.8 zu finden. In Abschnitt 4.3.10 und 4.4 werden zufällige Transferfunktionen zur Ermittlung der Qualität der verschiedenen Klassifikationsschemata eingesetzt. Dies entspricht prinzipiell einer Erweiterung der Isoflächenerzeugung durch Peaks in der Transferfunktion. Durch die Wahl einer zufälligen Transferfunktion besteht diese sozusagen ausschließlich aus Peaks und visualisiert damit sehr viele Isoflächen des Datensatzes zur gleichen Zeit. Dies stellt somit einen völlig neuen Ansatz zum Rendering von Volumendaten dar. Ergebnisse der Anwendung von randomisierten Transferfunktionen sind auf den Farbseiten in Abbildung 9.11 zu sehen.

Anstelle der dynamischen Beleuchtungsberechnung, die den Einsatz von speicherintensiven RGBA-Texturen erfordert, kann alternativ eine statische Beleuchtungsberechnung mit vorberechneten Skalarprodukten zwischen dem Lichtvektor und dem Gradienten oder dem Lichtvektor und dem Halfway-Vektor eingesetzt werden. Dies ermöglicht den Einsatz von Luminanz-Alpha-Texturen, die das vorberechnete Skalarprodukt und den Voxelwert enthalten. Diese verbrauchen

nur die Hälfte des Texturspeichers von RGBA-Texturen, allerdings kann dann nicht zur gleichen Zeit die diffuse und spekulare Beleuchtung ermittelt werden.

4.3.8 Probleme

Obwohl die Dicke der Slabs konstant gehalten werden kann, bedeutet dies üblicherweise nicht, dass auch konstante Strahlsegmentlängen während des Renderings vorliegen. Für viewport-parallele Schichten resultieren nur aus perspektivischen Projektionen Strahlsegmente mit unterschiedlicher Länge. Diese Abweichungen sind häufig vernachlässigbar, da in der Praxis extreme Perspektiven üblicherweise vermieden werden. Bei Perspektiven mit größerem Blickwinkel kann durch das Rendern von Kugelschalen (Shells) eine konstante Strahlsegmentlänge erreicht werden[54, 88]. Deshalb ist es in jedem Falle ausreichend, bei Verwendung von 3D-Texturen eine zweidimensionale Lookup-Tabelle einzusetzen, die nur vom Skalarwert am Start und am Ende des Strahlsegments abhängt.

Für objekt-parallele Schichten variiert der Schichtabstand zusätzlich bei der Rotation des Volumens. Jedoch gibt es für jede Rotation nur eine Strahlsegmentlänge, zumindest für orthogonale Projektionen. Der maximale Faktor der Variation dieser Länge ist $\sqrt{3}$. Um zu starke Artefakte zu vermeiden, kann ein Satz zweidimensionaler Lookup-Tabellen für unterschiedliche Längen eingesetzt werden. Dies ist ebenfalls notwendig bei Volumen, die in den Hauptachsenrichtungen unterschiedliche Abstände der Schichten vorweisen.

Neben dem Problem der unterschiedlichen Strahlsegmentlängen treten bei semi-transparenten Isoflächen Artefakte durch zweifach gezeichnete Pixel auf. Dies geschieht dann, wenn die Isofläche den Sichtstrahl genau an der Position der vorderen Schicht schneidet. Wird nun der nächste Slab gezeichnet, so wird das Pixel erneut in den Framebuffer geblendet, da die Fläche den Slab nun genau an der Position der hinteren Schicht schneidet. Um dieses Problem zu umgehen, kann der OpenGL Stencil-Test eingesetzt werden. Jedes Mal, wenn ein Slab in den Framebuffer gezeichnet wird, wird dieser Slab auch in den Stencil-Buffer mit einer speziellen Dependent-Texture gerendert, die nur dort Pixel zeichnet, wo die Isofläche den Strahl an der Position der vorderen Schicht schneidet. Wenn nun der nächste Slab gezeichnet wird, so vermeidet der Stencil-Test, dass genau diese Pixel erneut gezeichnet werden. So werden Fehler durch mehrfach gerenderte Isoflächen vermieden (siehe Abbildung 4.21). Leider funktioniert die Methode augenblicklich nur für eine einzige semi-transparente Isofläche.

Anstelle der Beleuchtungsberechnung durch ein Skalarprodukt der interpolierten Gradienten mit der Richtung der Lichtquellen, kann alternativ die Beleuchtungsinformation auch in einer *Lightmap* nachgeschlagen werden[96]. Dazu werden kubische *Environment-Maps* genutzt. Dies erlaubt beliebig komplexe Beleuchtungsverhältnisse und vermeidet zusätzlich Beleuchtungsartefakte durch die Nutzung nicht-normalisierter, interpolierter Gradienten.

4.3.9 Post-Klassifikation mit trilinearärer Interpolation

Auf die Vorteile von 2D-Texturen zum Rendering von Volumendaten wurde bereits in Abschnitt 4.2.1 eingegangen. Leider lässt sich der dort vorgestellte Ansatz zur trilinearen Interpolation mittels Multi-Texturen nicht auf Post-Klassifikation mit 1D-Lookup-Texturen übertragen.

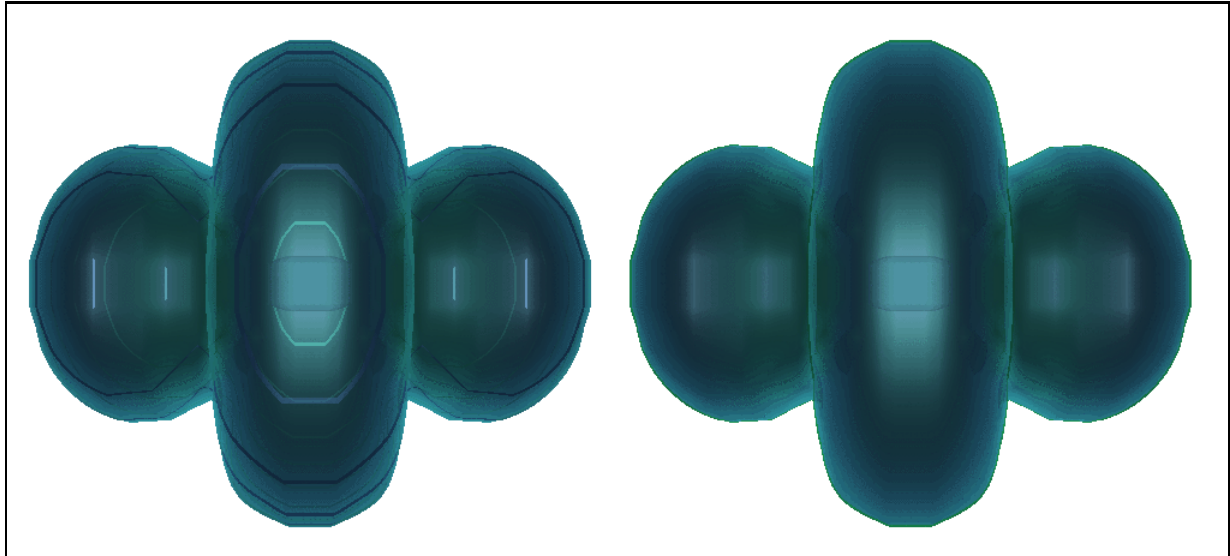


Abbildung 4.21: Semi-transparente Visualisierung einer Isofläche einer sphärischen, harmonischen Funktion ohne (links) und mit Korrektur (rechts). Die ringförmigen Artefakte sind im rechten Bild erfolgreich entfernt worden.

Die Interpolation in der dritten Dimension erfolgt nämlich relativ spät in der Pipeline, genauer gesagt in den Register-Combinern, während die Interpolation in den anderen Dimensionen früher in der Pipeline, in den Texture-Shadern, erfolgt. Eine Interpolation wie in Abschnitt 4.2.1 würde also eine Mischung aus Post- und Prä-Klassifikation zur Folge haben.

Die in diesem Abschnitt vorgeschlagene Konfiguration der Texture-Shader Einheit lässt sich auch für die Kombination von Post-Klassifikation Shading und trilinearer Interpolation bei Verwendung von objekt-parallelen Schichten einsetzen. Dies ist unter anderem auch deshalb sinnvoll, da der vorintegrierte Volume-Rendering-Ansatz einen linearen Verlauf der Daten des Volumens vom vorderen Skalarwert bis zum hinteren Skalarwert annimmt. Unter Umständen mag aber eine trilineare Interpolation vonnöten sein, weshalb eine trilineare Interpolation in Kombination mit Post-Klassifikation in bestimmten Situationen der Vorintegrierten Klassifikation vorzuziehen ist.

Abbildung 4.22 zeigt eine notwendige Dependent-Texture für die trilineare Interpolation mit 2D-Texturen in Kombination mit Post-Klassifikation. Wie in Abschnitt 4.2.1 werden Zwischenschichten gerendert, auf die jeweils angrenzende Schichten orthogonal als Multi-Texturen projiziert werden. Dabei werden in der Texture-Shader-Einheit gewichtete Skalare der vorderen und hinteren Schicht für den abhängigen Texturzugriff eingesetzt. Der Skalarwert der hinteren Schicht wird mit α und der der vorderen Schicht mit $1 - \alpha$ gewichtet, wobei α den Gewichtungsfaktor der Position der Zwischenschicht bezüglich der hinteren Schicht darstellt. Dies wird durch die Verwendung von $((1 - \alpha), 0, 0)^T$ and $(\alpha, 0, 0)^T$ als Texturkoordinaten für die dritte und vierte Texturstufe erreicht. Durch die Texture-Shader-Konfiguration werden die Skalarwerte vor dem abhängigen Texturzugriff mit Hilfe des Skalarprodukts dieser Stufen gewichtet. Dadurch werden

die Skalarwerte vor dem Lookup in die Transferfunktion interpoliert.

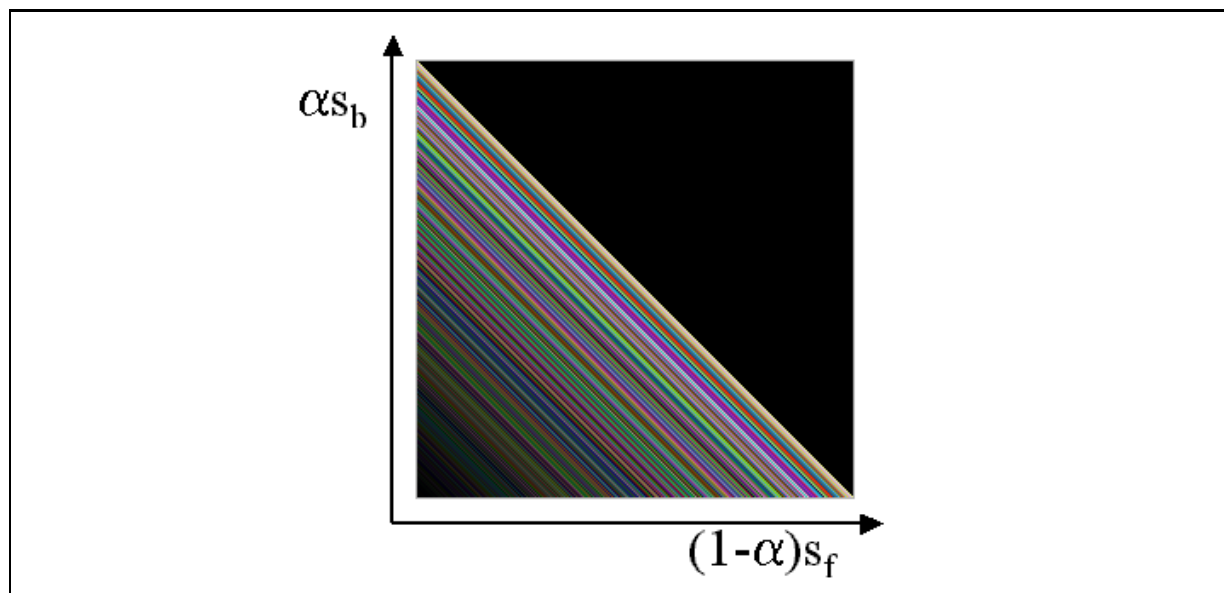


Abbildung 4.22: Dependent-Texture für trilineare Interpolation unter Verwendung von Post-Klassifikation und 2D-Texturen. Diese Textur ist das Ergebnis einer randomisierten Transferfunktion, die in der Textur auf der Diagonalen und den Nebendiagonalen zu sehen ist. α bezeichnet den Interpolationsfaktor bezüglich der Position der Zwischenschicht relativ zur hinteren Schicht.

Wie in Abschnitt 4.2.1 wird eine bilineare Interpolation der Skalarwerte durch die Texturumgebung vorgenommen. Die dritte Interpolation wird nun aber früher in der Rendering-Pipeline erzeugt, genauer gesagt in der Texture-Shader-Stufe, anstelle der Register-Combiner-Stufe. Man erreicht also zur gleichen Zeit die dritte Interpolation und eine Klassifikation in der Texture-Shader-Einheit.

4.3.10 Ergebnisse

Für die folgenden Messungen der Geschwindigkeit und Qualität des vorintegrierten Volume-Rendering-Verfahrens wurde ein PC eingesetzt, der mit einer 650 MHz Athlon CPU, 512 MB Speicher und einer NVIDIA GeForce3 Graphikkarte mit 64 MB DDR SDRAM (3.8 ns), 200 MHz internem Takt, 460 MHz Speichertakt und 4x AGP ausgestattet war. Die verwendeten Linux Treiber hatten die Versionsnummer 15.12.

In der wissenschaftlichen Visualisierung werden häufig unbekannte Volumendaten nach bestimmten Details durchsucht. Dafür ist die Möglichkeit der interaktiven Manipulation der Transferfunktion und der Isowerte ein wichtiges Hilfsmittel. Dies erfordert im Fall des vorintegrierten Volume-Rendering-Ansatzes eine Neuberechnung der Lookup-Textur im Hauptspeicher, die daraufhin in den lokalen Speicher der Graphikhardware transferiert werden muss. Für semi-

transparentes Volume-Rendering ist dazu bei einer globalen Änderung der Transferfunktion das Lösen des Volume-Rendering-Integrals für alle Kombinationen von vorderem und hinterem Skalarwert notwendig. Bei lokaler Manipulation ist wie bereits oben erläutert im schlechtesten Falle eine Aktualisierung der Hälfte der Lookup-Textur notwendig. Im Falle der Isoflächendarstellung ist die Berechnung der Lookup-Textur sehr viel einfacher, da hierbei ein reines Setzen oder Überblenden von Werten in der Lookup-Textur an die Stelle der numerischen Integration tritt. Im Folgenden werden nur die Zeiten zur Aktualisierung der Lookup-Textur betrachtet, da die Zeit zum Transfer der Lookup-Textur zum Speicher der Graphikhardware bei den hier verwendeten, auf 8 Bit quantisierten Daten vernachlässigt werden kann (64KB Daten der Dependent-Texture). Die komplette Neuberechnung der Lookup-Textur für eine semi-transparente Darstellung unter Berücksichtigung der Selbstabschwächung innerhalb eines Slabs nach den Gleichungen (4.5) und (4.4) benötigt ungefähr 20 Sekunden. Dagegen ist die Integration nach den Gleichungen (4.12) und (4.9) ohne die Berücksichtigung der Selbstabschwächung im Mittel in 43 Millisekunden möglich. Daraus folgt, dass die Transferfunktion in einer Sekunde also ca. 25 mal aktualisiert werden kann, und damit die geforderte Interaktivität gewährleistet ist. Eine Manipulation einzelner Einträge in der Transferfunktion benötigte im schlechtesten Fall die Hälfte der oben angegebenen Zeiten. Die Aktualisierung einer Lookup-Textur zur Darstellung von Isoflächen, die sowohl Farbe und Transparenz der Isoflächen als auch Interpolationsgewichte enthält, dauerte für eine Isofläche im schlechtesten Fall 13 ms, entsprechend für n Isowerte $n \times 13ms$. Eine interaktive Manipulation der Isowerte ist also auch für viele Isoflächen garantiert.

Sowohl bei der Manipulation der Transferfunktion, beim Wechsel des Isowertes, als auch bei der Änderung von Kameraparametern wird ein Rendering-Vorgang angestoßen. In Abbildung 4.23 sind die Bildwiederholraten für verschieden dimensionierte Datensätze bei einer Viewport-Größe von 600×600 Pixeln unter Verwendung von 2D- und 3D-Texturen angegeben, die den Nachteil bezüglich der Geschwindigkeit von 3D-Texturen gegenüber 2D-Texturen sehr deutlich machen. Dieser Nachteil lässt sich, wie bereits vorher angedeutet, durch die schlechtere Kohärenz bei der Nutzung Chip-interner Caches erklären. Diese sind üblicherweise relativ klein dimensioniert, so dass zwar eine einzelne 2D-Textur dort abgelegt werden kann. Bei der Nutzung von 3D-Texturen muss der Cache dagegen ständig neu gefüllt werden. Dies hat erheblich höhere Speicherbandbreitenanforderungen zur Folge. Bei größeren Volumendatensätzen fällt der Unterschied zwischen 2D- und 3D-Texturen allerdings durch den dreifachen Speicherbedarf von 2D-Texturen und dem damit verbundenen Transfer der Texturdaten über den AGP-Bus kleiner aus.

Für Datensätze mit numerisch höherer Auflösung ergibt sich ein wesentlicher Nachteil durch die größere Zahl von benötigten Vorintegrationsschritten und den hohen Speicherbedarf für die Nachschlagtabellen. Bei einer Datenauflösung von 12 Bit, die in der medizinischen Bildverarbeitung heute üblich sind, benötigt jede Transferfunktion $2^{12} \times 2^{12} = 2^{24} \approx 16$ Millionen Vorintegrationen und damit für RGBA-Werte 64 MB Texturspeicher.

Sicherlich ist die mit einer relativ geringen Anzahl an Schichten erzielbare Rendering-Qualität des vorintegrierten Volume-Rendering ein wesentlicher Vorteil gegenüber bisherigen Ansätzen. Dies ist auf zwei wesentliche Gründe zurückzuführen: Erstens können die später nachgeschlagenen Strahlsegmente mit voller numerischer Genauigkeit der CPU integriert werden. Zweitens können numerische Fehler durch die verringerte Zahl der Framebuffer-Blending Ope-

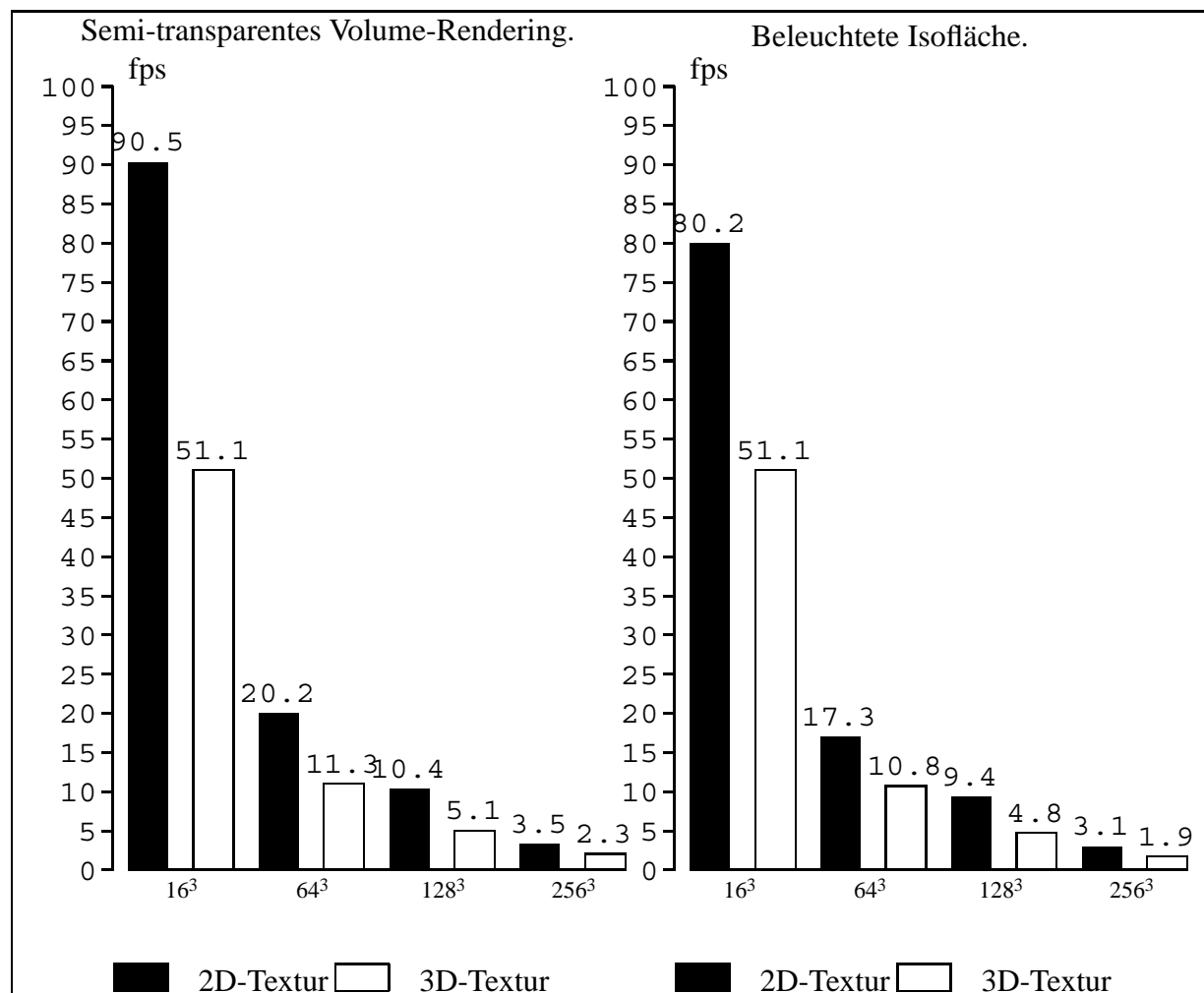


Abbildung 4.23: Bildwiederholraten bei verschiedenen dimensionierten Volumendatensätzen für Vorintegriertes Volume-Rendering mit 2D- und 3D-Texturen.

rationen verringert werden, da das Blending im Framebuffer in aktueller Hardware mit einer Genauigkeit von nur 8 Bit erfolgt.

Die Qualität des Volume-Rendering mit Vorintegration unter Nutzung flexibler Rasterisierungshardware wird auf den Farbseiten in den Abbildungen 9.9 und 9.10 demonstriert. Abbildung 9.9 zeigt verschiedene Darstellungsvarianten einer CT-Aufnahme eines menschlichen Zahnes. Die verschiedenen Darstellungen werden allein durch die Anwendung unterschiedlicher abhängiger Texturen, unterschiedlicher Texturformate und unterschiedlicher Register-Combiner Verschaltung erreicht. Beispiele für die Visualisierung von Isoflächen sind auch in den Abbildungen 9.6 und 9.7 zu finden. Dabei werden die Vorteile dieses Ansatzes deutlich: Beliebige viele Isoflächen können in einem einzigen Rendering-Vorgang dargestellt werden. Über deren Menge, Farbe und Transparenz entscheidet allein der Aufbau der abhängigen Textur. Farbe und Transparenz der verschiedenen Isoflächen können individuell jeweils für Vorder- und Rückseite getrennt

zugewiesen werden.

Eine Variante der Darstellung von Isoflächen ergibt sich durch die Verwendung vorintegrierter abhängiger Texturen, berechnet mit einer Transferfunktion, in die für jede Isofläche eine Spitze (*Peak*) gesetzt wird. Dadurch wird auch das Mischen semi-transparenter Bereiche mit Isoflächen möglich. Ein Beispiel einer solchen Visualisierung ist in Abbildung 9.8 zu sehen. Zur Beleuchtungsberechnung kann dabei wieder Volume-Shading zum Einsatz kommen.

Eine interessante Erweiterung ergibt sich durch die Verwendung randomisierter Transferfunktionen. Diese entspricht einer Transferfunktion mit sehr vielen Peaks, wodurch sich sozusagen alle Isoflächen des Volumendatensatzes gleichzeitig darstellen lassen (siehe Abbildung 9.11). Die Stärke des vorintegrierten Volume-Rendering-Ansatzes zeigt sich hier deutlich, da trotz der sich ergebenden hohen Frequenzen in der Transferfunktion keine Details verloren gehen. Transferfunktionen dieser Art werden im folgenden Kapitel auch zur Evaluation der Qualität der verschiedenen Klassifikation-Schemata eingesetzt.

Ausblickend soll hier nochmals auf die Leistungsfähigkeit des hier vorgestellten Verfahrens zum vorintegrierten Volume-Rendering hingewiesen werden. Durch die Verwendung eines Skalarprodukts in den Texture-Shadern können verschiedene Volumina während des Renderings gemischt werden, wodurch sich volumetrische Effekte realisieren lassen (siehe Abbildung 9.14). Dieser Ansatz ähnelt der zeitlichen Interpolation zwischen Volumina verschiedener Zeitschritte aus Abschnitt 4.2.5. Allerdings werden hier bereits während des Nachschlags der Texturen verschiedene Volumina, die in den einzelnen Farbkanälen einer Textur gespeichert sind, durch eine Skalarproduktoperation der Texture-Shader-Einheit miteinander gewichtet.

Im folgenden Abschnitt wird nun ein qualitativer Vergleich des Volume-Renderings mit Prä-, Post- und Vorintegrierter Klassifikation durchgeführt.

4.4 Ergebnisse

Der stetige Fortschritt der Qualität und Leistungsfähigkeit im Bereich der PC-Graphikhardware spiegelt sich deutlich in diesem Kapitel wider. Während im ersten Abschnitt die Nutzung von Standard-Texturen im Vordergrund steht, beschäftigen sich die darauffolgenden Abschnitte mit der Nutzung von speziellen Erweiterungen zur Verbesserung der Qualität und Darstellungsgeschwindigkeit texturbasierter Verfahren. Das Vorintegrierte Volume-Rendering kann dabei auch als Erweiterung des in Abschnitt 4.2.2 vorgestellten Ansatzes zur Beschleunigung des Renderings von Volumen angesehen werden. Während zur Beschleunigung zwei Texels in einem Schritt übereinander geblendet wurden, dienen diese beim Vorintegrierten Volume-Rendering als Textur-Koordinaten zum Nachschlag der vorintegrierten Strahlsegmente.

Die erzielbare Qualität der vorgestellten Klassifikationsschemata wird in den Abbildung 9.11, 9.12 und 9.13 auf den Farbseiten deutlich. Abbildung 9.11 zeigt einen Vergleich der Ergebnisse von Prä-Klassifikation, Post-Klassifikation und Vorintegration am Beispiel einer sphärischen harmonischen Funktion mit 32^3 Voxeln Auflösung, die mit 32 Schichten und einer randomisierten Transferfunktion dargestellt wird. Das „verwaschene“ Ergebnis der Prä-Klassifikation ist auf die Interpolation der Farben zurückzuführen. Dagegen bilden sowohl die Post-Klassifikation als auch die Vorintegration die hohen Frequenzen der Transferfunktion gut

ab. Im Falle der Post-Klassifikation werden die hohen Frequenzen aber nur auf den Schichtpolygonen deutlich. Der Funktionsverlauf zwischen den Schichtpolygonen kann nur durch das Einfügen trilinear interpolierter Schichten abgebildet werden. Vorintegration bildet den Funktionsverlauf sowohl auf den Schichtpolygonen als auch zwischen den Schichtpolygonen ab, was zu einem sehr viel harmonischeren Gesamteindruck führt.

Bei der Darstellung von Isoflächen mit Hilfe von Rasterisierungshardware haben sowohl die durch Vorintegration gewonnenen Isoflächen als auch die Alpha-Test-Isoflächen wesentliche Vorteile gegenüber der Rekonstruktion von Dreiecksflächen: Der Isowert kann durch Änderung des Alpha-Test-Referenzwertes oder durch die Anpassung der abhängigen Textur interaktiv geändert werden. Zudem ist kein zusätzlicher Speicher für eine Zwischenrepräsentation nötig. Vorintegrierte Isoflächen sind aber den Alpha-Test-Isoflächen in zweierlei Hinsicht überlegen: Einerseits sind Alpha-Test-Isoflächen nur einseitig darstellbar. Andererseits kann mit Hilfe des Alpha-Tests pro Rendering-Vorgang nur eine einzige Isofläche dargestellt werden. Dagegen können bei der vorintegrierten Isoflächendarstellung beliebig viele Isoflächen in einem einzigen Rendering-Durchgang dargestellt werden, wobei für Vorder- und Rückseite unterschiedliche Farben und Transparenzen gewählt werden können. Es lassen sich zwar mit dem Alpha-Test auch beidseitig beleuchtete Isoflächen realisieren[8], dies ist aber nur durch ein mehrstufiges Verfahren zu erreichen. Beispiele vorintegrierter Isoflächen sind auf den Farbseiten in den Abbildungen 9.6, 9.7 und 9.9 zu finden.

2D-Texturen haben speziell beim vorintegriertem Ansatz einen wesentlichen Vorteil gegenüber 3D-Texturen, da auf den Originaldaten gearbeitet wird, das Volumen also im Gegensatz zum 3D-Textur-Ansatz nicht mit der Nyquist-Frequenz abgetastet werden muss. Ohne die Vorintegration verschwindet dieser Vorteil, da, wie in den Abschnitten 4.2.1 und 4.3.9 demonstriert, das Einfügen von trilinear interpolierten Zwischenschichten zur Bildqualitätssteigerung auch bei 2D-Texturen möglich ist. Ein bedeutender Nachteil von 2D-Texturen ist außerdem immer noch der höhere Speicherbedarf und die Umschalteffekte beim Wechsel des Schichtenstapels. Dieser Nachteil kann zwar durch das Interpolieren beliebig ausgerichteter Schichten aus einem Schichtenstapel umgangen werden (siehe Abschnitt 4.2.1). Dies hat aber wiederum eine Verminderung der Geschwindigkeit zur Folge.

Im Abbildung 4.24 sind Post-Klassifikation und Vorintegration hinsichtlich ihrer Geschwindigkeit auf dem NV20 Chip von NVIDIA verglichen. Dazu wurden für einen Bildschirmabschnitt mit 640×640 Pixeln direktes Volume-Rendering (8-Bit-Texturen) und Isoflächenrendering (32-Bit-Texturen) miteinander verglichen. Dabei wird deutlich, dass sowohl die Verwendung von 3D-Texturen, als auch die Nutzung von mehr als zwei Texturoperationen deutliche Geschwindigkeitsnachteile bringt. Die Nachteile der 3D-Texturen lassen wieder durch schlechtere Cache-Kohärenz erklären. Die Implementierung der Vorintegration benötigt auf aktueller Graphikhardware von NVIDIA vier Texturoperationen, während Post-Klassifikation mit zwei Texturoperationen auskommt. Im NV20 Chip von NVIDIA werden die letzten beiden Texturoperationen durch einen *Loopback* in der Hardware realisiert. Dies erklärt die Nachteile der Vorintegration bezüglich Geschwindigkeit, die aber durch weitere Verbesserungen in künftiger Hardware beseitigt werden können. Zudem wird, durch die geringere Zahl an Schichten, dieser Nachteil kompensiert. Schließlich sind die geringeren Bildwiederholraten bei der Darstellung von Isoflächen auf den erhöhten Speicherbandbreitenbedarf bei der Nutzung von 32-Bit-Texturen

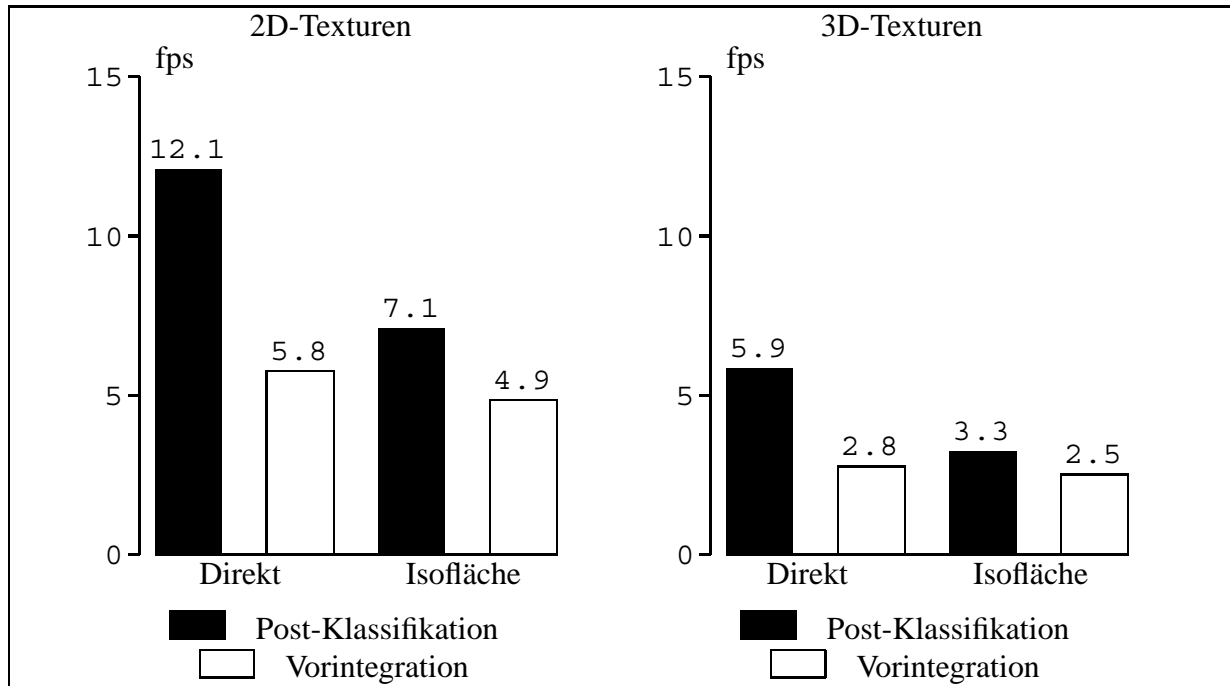


Abbildung 4.24: Bildwiederholraten beim Rendering des Engine-Datensatzes ($256 \times 256 \times 110$) unter Verwendung von Post-Klassifikation (2 Texturoperationen) und Vorintegration (4 Texturoperationen) mit 2D- und 3D-Texturen.

zurückzuführen. Die Verwendung speicherintensiverer Texturen führt auch dazu, dass die Unterschiede bei der Darstellung von Isoflächen geringer ausfallen, da die Speicherbandbreite hier zum limitierenden Faktor wird.

Dieses Kapitel macht die hervorragende Eignung moderner PC-Graphikhardware zur Visualisierung von Volumendaten deutlich. Wegen der zu erwartenden breiten Verfügbarkeit dieser Hardware auf einem Großteil der Arbeitsplatzrechner, sollte diese Hardware wenn möglich im Client-Server-Umfeld eingesetzt werden. Multi-Texturen in Verbindung mit flexibel programmierbaren Rasterisierungseinheiten bieten dabei ganz neue Möglichkeiten und erlauben vielerlei Verbesserungen hinsichtlich Darstellungsgeschwindigkeit, Qualität und Funktionalität. Eine Reihe von Forschungsarbeiten gehen inzwischen auf diese gesteigerte Flexibilität ein. Dazu zählt die Verwendung von PC-Graphikhardware zur Implementierung von hochqualitativen Filtern[55], als auch die Nutzung abhängiger Texturzugriffe für multi-dimensionale Transferfunktionen[82].

In den nächsten Jahren ist mit weiteren Verbesserungen in diesem Bereich zu rechnen. Dabei dürfte eine weitere Flexibilisierung und die Erhöhung der Qualität im Vordergrund stehen. Letzteres lässt sich beispielsweise durch höhere Genauigkeit der Farbkanäle der Texturen oder des Framebuffers erreichen. Dies würde nochmals deutliche Qualitätsgewinne für das Rendering von Volumendaten bringen. Auch einer der Hauptnachteile heutiger PC-Graphikhardware, nämlich die limitierte Verfügbarkeit von Speicher, könnte in nächster Zeit durch die bereits beginnende Einführung von *Shared-Memory*-Architekturen aufgehoben werden. Eine weitere interessante

Richtung ist sicherlich auch die Nutzung von programmierbaren Einheiten zur Realisierung von Kompressions- und Multi-Resolution-Ansätzen. Gerade solche Ansätze werden durch die neu hinzugekommenen indirekten Speicherzugriffe in steigenden Maße möglich werden.

Beauty, like ice, our footing does betray;
Who can tread sure on the smooth, slippery way:
Pleased with the surface, we glide swiftly on,
And see the dangers that we cannot shun.

John Dryden, 1631-1700

Kapitel 5

Extraktionstechniken für Isoflächen

Neben den im letzten Kapitel besprochenen direkten Methoden der Volumenvisualisierung spielen indirekte Methoden in der wissenschaftlichen Volumenvisualisierung heute immer noch eine große Rolle. In Abschnitt 2.3.1 wurde bereits der fundamentale und gleichzeitig prominenteste Vertreter dieser Algorithmen, der Marching-Cubes-Algorithmus, erläutert. Den unbestrittenen Qualitäten und Vorteilen dieses Algorithmus steht die enorme Zahl der generierten geometrischen Primitive gegenüber, die dieser Ansatz aus Volumendatensätzen heute üblicher Komplexität rekonstruiert. Die Neugenerierung der Isofläche, die bei jedem Wechsel des Isowertes notwendig wird, ist dabei mit erheblichem Zeitaufwand und Speicherbedarf verbunden. Somit stellt eine interaktive Darstellung selbst leistungsfähige Graphikworkstations vor erhebliche Probleme und ist im Allgemeinen ohne die Zuhilfenahme „intelligenter“ Algorithmen sogar unmöglich. Eine Einbettung dieser Visualisierungen in digitale Dokumente im Client-Server-Umfeld führt zu weiteren Problemen, die im Wesentlichen auf limitierte Bandbreiten und Rechenressourcen zurückzuführen sind. Zur Lösung dieser Probleme sind speziell auf die Infrastruktur und den Anwendungsfall zugeschnittene Algorithmen notwendig, die in diesem Kapitel zur Sprache kommen sollen.

Im folgenden Abschnitt wird zunächst eine architekturneutrale Methode zur lokalen Rekonstruktion und Visualisierung von Isoflächen vorgestellt, die sich hauptsächlich für Einsatzgebiete eignet, in denen kleiner dimensionierte Volumendaten anfallen. Die Tauglichkeit des Ansatzes wird anhand des Web-Dienstes *OrbVis* demonstriert. Die Grenzen dieses Ansatzes werden allerdings anschließend bereits bei der im Anschluß beschriebenen Applikation zur Visualisierung elektronenkryomikroskopischer Dichten deutlich. Der darauffolgende Abschnitt befasst sich mit einem progressiven Verfahren zur Extraktion, Übertragung und Darstellung von Isoflächen aus einer Fehler-basierten Multi-Level-Approximation der Volumendaten.

Während das im folgenden Abschnitt vorgestellte Verfahren noch in die Kategorie der Client-seitigen Methoden fällt, sind alle anderen in diesem Kapitel vorgestellten Verfahren als hybrid anzusehen. Durch die geschickte Aufteilung der Visualisierungsaufgaben auf Client und Server sind diese Verfahren in der Lage, auch große Volumendatensätze handhabbar zu machen. Unter diese Kategorie fallen also auch die im letzten Abschnitt dieses Kapitels vorgestellten

Methoden. Auf Basis einer Octree-Datenrepräsentation werden hier verschiedene Aufteilungen des Marching-Cubes-Algorithmus auf Client und Server, eine Fokus-basierte Detailgenauigkeitssteuerung (*Level-of-Detail*) und Komprimierungsverfahren auf der Grundlage von Dreiecksstreifen (*Triangle-Strips*) erläutert.

5.1 Lokale Isoflächenrekonstruktion

Isoflächen aus einer Marching-Cubes-Rekonstruktion sind polygonale Gitter, die problemlos mit heute vorhandenen proprietären oder auf Standards basierenden Plugins (siehe Abschnitt 2.4.4) in digitale Dokumente eingebettet werden können. Solch eine Einbettung statischer Modelle mag für gewisse Anwendungsfälle ausreichend sein, oftmals ist jedoch die direkte Einbettung der Volumendaten mit einer dynamischen Generierung verschiedener Visualisierungen wünschenswert. Dazu können die Volumendaten zusammen mit geeigneten Algorithmen zur Extraktion und Darstellung von Isoflächen in ein digitales Dokument eingebracht werden, so dass diese auf einen Arbeitsplatzrechner übertragen werden und erst dort dynamisch Darstellungen generiert werden. Folglich ergibt sich eine wesentlich flexiblere Visualisierung, die dem Benutzer bedeutend mehr Freiheiten lässt, die Daten zu erkunden.

5.1.1 Algorithmus

Die Grundidee des in diesem Abschnitt verfolgten Ansatzes zur Visualisierung dynamischer Isoflächen im Internet ist es, Volumendaten zusammen mit einem Java-Applet in ein digitales Dokument einzubetten. Nach Übertragung auf einen Arbeitsplatzrechner, wird dort das Java-Applet ausgeführt und rekonstruiert lokal aus den Daten Flächen, die unter Nutzung der vorhandenen Graphikhardware dargestellt werden. Dazu wurde eine Java-Klasse zur Extraktion von Isoflächen unter Verwendung des Marching-Cubes-Algorithmus erstellt. Weiterhin besteht das Java-Applet aus einer Benutzeroberfläche, die das Anpassen von Visualisierungsparametern, wie beispielsweise das Ändern des Isowertes oder das Setzen der Opazität der Isofläche, erlaubt.

Die von der Marching-Cubes-Klasse erzeugten Dreiecke werden über eine abstrakte Schnittstelle an eines der Darstellungsmodulare weitergeleitet. Im Moment stehen Module für die hardwarebeschleunigte Darstellung der Daten in VRML und Java3D zur Verfügung. Auf eine Implementierung mittels Java OpenGL-Bindings wurde verzichtet, da die Funktionalität von VRML und Java3D zur Darstellung von Dreiecksmodellen völlig ausreicht und keine Hardware-spezifischen Erweiterungen benötigt werden.

Java3D erlaubt über seine Szenengraphschnittstelle das komfortable Einfügen der generierten geometrischen Primitive in den Szenengraphen. Dazu wird ein neu erzeugter `IndexedTriangleArray`-Knoten mit den Vertex-, Konnektivitäts- und Gradientendaten gefüllt und in den Szenengraphen eingehängt oder bei Änderung des Isowerts ein vorhandener Knoten mit neuen Daten gefüllt.

Die VRML-Implementierung basiert auf der Einbettung eines VRML-Plugins zusammen mit dem Java-Applet in eine HTML-Seite. Das EAI erlaubt auch das Erzeugen neuer Knoten über die `createVRMLFromString`-Methode. Da diese aber aus Zeichenfolgen in VRML-Syntax

Knoten erzeugt, ist die Generierung neuer Knoten mit umfangreicher Geometrie sehr ineffizient. Deshalb wurde ein anderer Ansatz gewählt: Am Beginn einer Visualisierungssitzung wird eine Basis-VRML-Szene mit leerem `IndexedFaceSet`-Knoten durch das VRML-Plugin vom HTTP-Server geladen. Nachdem vom Java-Applet aus den Volumendaten eine Isofläche extrahiert wurde, wird der `IndexedFaceSet`-Knoten über das External Authoring Interface mittels eines Events mit den Dreiecksdaten gefüllt. Bei Änderung des Isowertes werden mit den neu erzeugten Dreiecksdaten die Daten ebenfalls über ein Event des EAI im `IndexedFaceSet`-Knoten überschrieben. Auf diese Weise kann eine VRML-Szene relativ effizient mit dynamischer Geometrie gefüllt werden. Es ergeben sich eine Reihe von Anwendungsmöglichkeiten, die am Beispiel chemischer Volumendaten in den folgenden Abschnitten erläutert werden sollen.

5.1.2 Anwendungen in der Chemie

Innerhalb der Naturwissenschaften hat in den letzten Jahren vor allem die Chemie im Bereich der digitalen Dokumente sowie der Web-basierten chemischen Dienste erhebliche Fortschritte erzielt. Allerdings wurden vor allem Internet-Dienste entwickelt, die zur Lösung von einzelnen Frage- und Problemstellungen bestimmt waren. Der große Vorteil des WWW, nämlich die einfache Verknüpfung von digitalen Dokumenten und der damit verbundene Informationsfluss wird dadurch unterbrochen.

Ein weiteres Problem, das im Kontext des Internet immer wieder auftaucht, ist das Fehlen von geeigneten Mechanismen zur Darstellung von chemischen Informationen. Zwar können Strukturen oder Volumendaten auch als Bilder oder MPEG-Videos dargestellt werden, allerdings erlauben diese Datenformate keinerlei Interaktion mit den Daten. Auch statische 3D-Szenen kommen schnell an ihre Grenzen, wenn es um das Verständnis von komplexeren Sachverhalten geht. Erst durch den Einsatz dynamischer Methoden zur Visualisierung ist eine verständliche Dokumentation und Erklärung komplexer Phänomene möglich.

Die hier geschilderte Situation erfordert deshalb den Einsatz eines Web-basierten Systems zur Beschaffung, Analyse und Visualisierung von chemischen Daten. Im Folgenden wird die Anbindung eines Dienstes zur Visualisierung chemischer Volumendaten an solch ein Datenmanagementsystem beschrieben. Diese Anbindung ermöglicht einen lückenlosen Informationsfluss zwischen digitalen Dokumenten von anderen Modulen bis hin zum Volumenvisualisierungsmodul.

5.1.3 Anbindung an ein chemisches Datenmanagementsystem

Das CACTVS-System[68, 67] ist solch ein offenes, chemisches Datenmanagementsystem (siehe Abbildung 5.1). Die Applikation besteht dabei aus einem Benutzer-Teil, der sich in Eingabe und Ausgabe gliedert, einer dünnen CGI-Schicht und der CACTVS-Kern-Bibliothek. Zudem können beliebige externe Programme eingebunden werden. Die benutzerdefinierte Eingabe von chemischen Informationen erfolgt auf zwei Wegen. Zum einen kann der Benutzer direkt chemische Daten an den Dienst übermitteln, indem er über ein HTML-Formular chemische Datenformate einliest. Der Online-Dienst ist dabei nicht auf eine limitierte Anzahl von bestimmten Datenobjekten und -formaten beschränkt, sondern kann in dieser Hinsicht kontinuierlich erweitert wer-

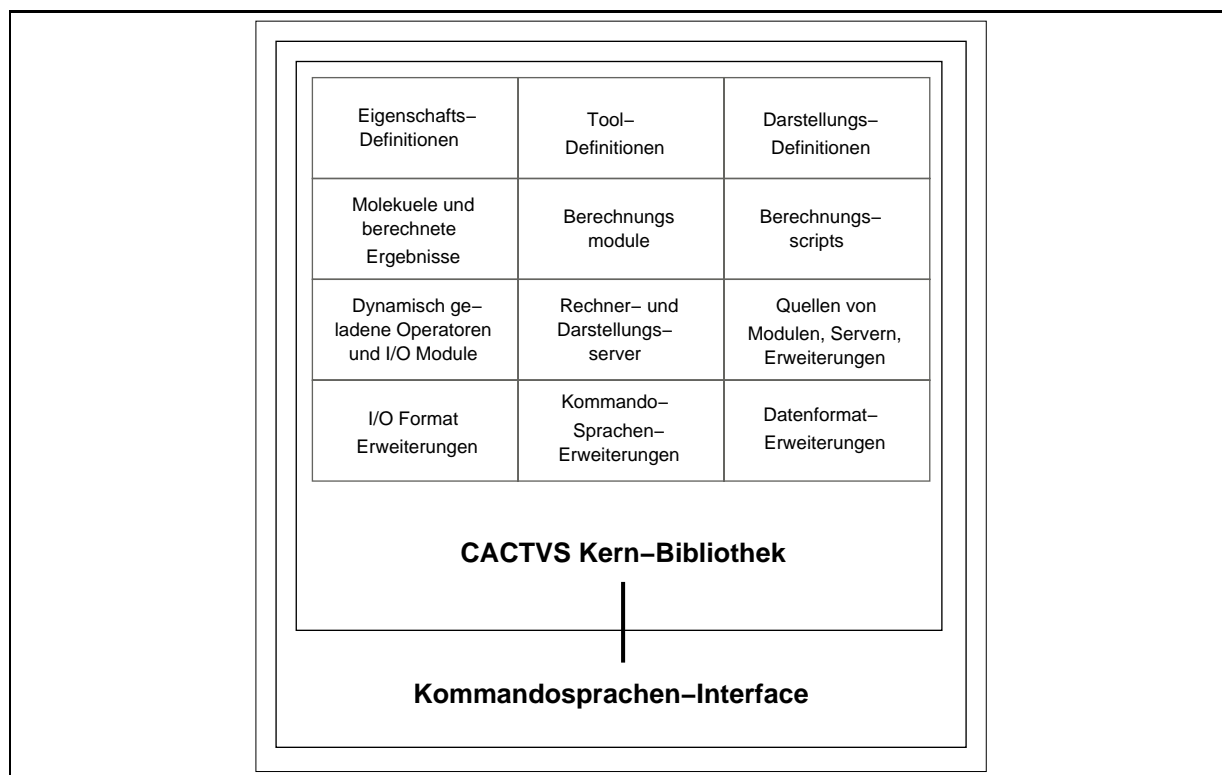


Abbildung 5.1: Schematische Darstellung des CACTVS-Systems.

den. Außerdem kann der Benutzer mit Hilfe von Java-Applets chemische Strukturen zeichnen und diese Informationen an das HTML-Formular übergeben. Diese Art der Eingaben finden sich beispielsweise in den chemischen Diensten *OrbVis*[36] und *ComSpec3D*[69] wieder. Auf der anderen Seite kann die Applikation auch direkt chemische Information aus anderen digitalen Dokumenten übernehmen. Diese Web-typische Art der Verknüpfung von digitalen Dokumenten ermöglicht somit den lückenlosen, kohärenten und flexiblen Transfer von strukturellen und anderen chemischen Daten im Internet. Das CGI-Script ist im Allgemeinen nur einige zehn Zeilen lang. Es kontrolliert lediglich die Kommunikation zwischen Client und Server bzw. Benutzer und Datenmanagementsystem. Die Verwaltung, Analyse und Beschaffung der chemischen Informationen wird alleine durch das offene Datenmanagementsystem bewältigt. Das CACTVS-System verfügt ebenfalls über Module zur Generierung von 2D- und 3D-Daten (z.B. Molekülzeichnungen als GIF-Dateien, 3D-Molekülmodelle als VRML-Dateien, etc.). Mit Hilfe von Plugins und Java-Applets lassen sich diese Dateiformate anschließend betrachten. In Hinblick auf dynamische Visualisierungsmethoden und der chemischen Weiterverarbeitung bzw. Analyse der visualisierten chemischen Daten wurden Client-seitige Visualisierungsmethoden entwickelt, die sich einfach in das Gesamtgefüge einbinden lassen.

5.1.4 Orbitalvisualisierung

Der Internetdienst *OrbVis* wurde vor allem wegen der Notwendigkeit einer didaktischen Erschließung eines tendenziell sehr theoretischen Gebietes der Chemie, der Quantenchemie, entwickelt. Aufgrund des abstrakten Charakters und dem großen Anteil an höherer Mathematik empfinden angehende und auf dieses Gebiet nicht spezialisierte Chemiker diesen Teil der Chemie als schwer zugänglich. Dennoch sollte jeder Chemiker ein solides Grundwissen auf diesem Gebiet besitzen, da die Quantenchemie die Basis für das Verständnis vieler wichtiger chemischer Phänomene darstellt. Um die grundsätzlichen Prinzipien von *OrbVis* darzustellen, wird im Folgenden ein kurzer Exkurs in die allgemeine Quantenchemie und die approximative Berechnung von Molekülorbitalen[116] gegeben. Anschließend wird die detaillierte Implementierung geschildert.

Die Quantenmechanik basiert auf der allgemeinen Schrödinger-Gleichung. Diese Gleichung besteht im Grunde aus der Wellenfunktion Ψ , welche die Elektronenverteilung für ein Atomorbital beschreibt, der Energie E und dem Hamilton-Operator \hat{H} .

$$\hat{H}\Psi = E\Psi \quad (5.1)$$

Der Hamilton-Operator enthält wiederum die Summe der kinetischen und potentiellen Energien von Elektronen und Atomkernen, sowie ihre Wechselwirkung. Um die Schrödinger-Gleichung zu lösen, sind eine Reihe von Näherungen nötig. Die erste Näherung ist die so genannte Born-Oppenheimer-Näherung, deren Grundlage die unterschiedlich schnelle Bewegung von Elektronen und Atomkernen ist. Die Masse der Atomkerne ist einige tausendmal größer als die der Elektronen. Aufgrund der daraus resultierenden wesentlich höheren Geschwindigkeit der Elektronen, können sich diese fast instantan auf eine Änderung der Kernposition einstellen. Aus der Sicht der Elektronen bleiben die Kerne somit quasi statisch, so dass bei der Betrachtung der Wechselwirkungen zwischen Elektronen und Kernen die Bewegungen der Kerne vernachlässigt werden können. Die Schrödinger-Gleichung vereinfacht sich somit auf eine elektronische Schrödinger-Gleichung:

$$\hat{H}_{el}\Psi_{el} = E_{el}\Psi_{el}. \quad (5.2)$$

Die im elektronischen Hamilton-Operator enthaltenen Wechselwirkungsanteile zwischen den Elektronen verhindern allerdings immer noch eine einfache Lösung, so dass eine weitere Näherung, die Ein-Elektronen- bzw. Hartree-Fock-Näherung durchgeführt werden muss. Durch Vorgabe einer antisymmetrischen Wellenfunktion erhält man für jedes Elektron ein gemitteltes Elektronenfeld. Das heißt, jedes Elektron erfährt das effektive Potential aller anderen Elektronen. Auf diese Art und Weise bekommt man einen Satz von n Ein-Teilchen-Eigenwertgleichungen, die Hartree-Fock-Gleichungen (HF-Gleichungen):

$$F\varphi_i = \varepsilon_i\varphi_i. \quad (5.3)$$

Da der Fock-Operator F von den Parametern φ_i abhängt, können die HF-Gleichungen nur iterativ gelöst werden. Zur weiteren Lösung ist daher eine letzte Näherung nötig. φ_i ist dabei die Summe über eine endliche Zahl von Basisfunktionen χ .

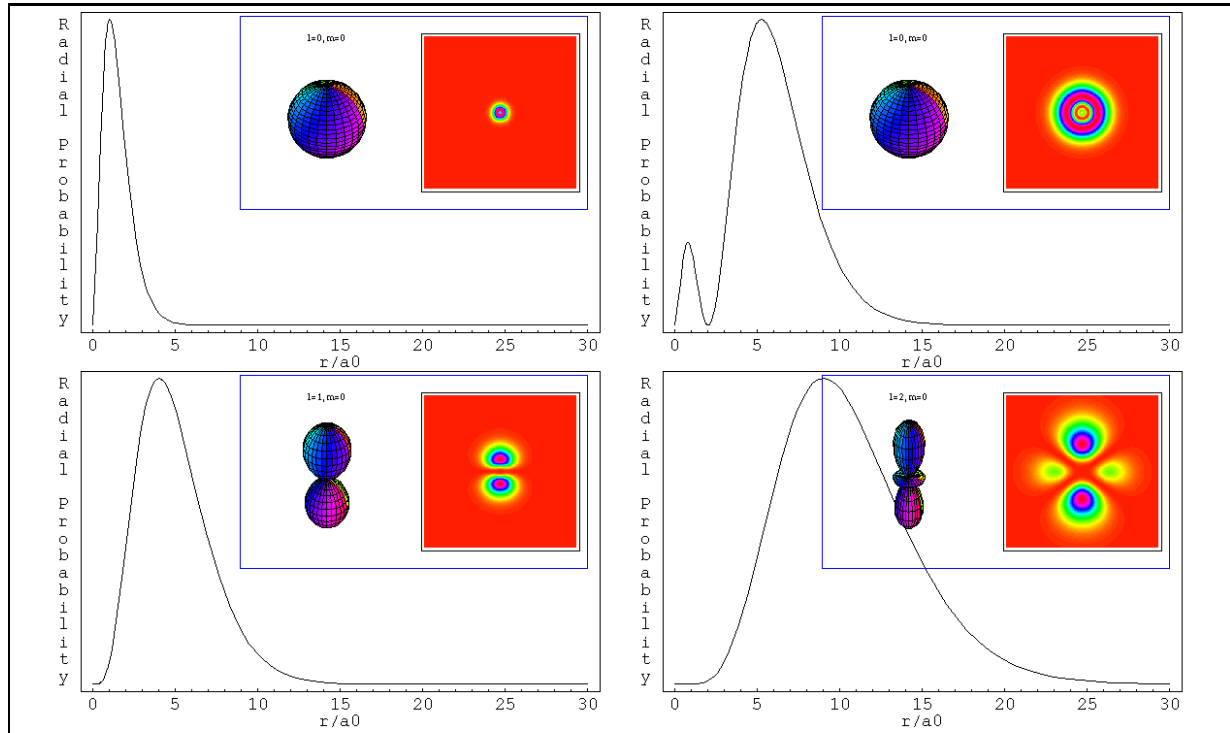


Abbildung 5.2: Wellenfunktionen der 1s-, 2s-, 2p- und 3d-Wasserstoff-Orbitale als radiale Aufenthaltswahrscheinlichkeitsfunktion, jeweils mit 3D-Visualisierung und 2D-Dichteplot.

$$\varphi_i = \sum c_i \chi_i \quad (5.4)$$

Im Allgemeinen handelt es sich bei diesen Basisfunktionen um die Wellenfunktionen der Atomorbitale der Valenzschale (äußerste Schicht in der Elektronenhülle). Die Näherung wird demnach auch als **Linear Combination of Atomic Orbitals (LCAO)** bezeichnet. Die Gestalt der Atomorbitale ist dabei abhängig vom jeweiligen Atomtyp und der Nebenquantenzahl (siehe Abbildung 5.2). In *OrbVis* werden nur die s- und p-Orbitale der Atome berücksichtigt, weshalb man auch von einer minimalen s,p-Basis spricht. Grundsätzlich kann bei den Basisfunktionen zwischen den so genannten **Gauß Type Orbitals (GTOs)** und den **Slater Type Orbitals (STOs)** unterschieden werden. Beide Typen beschreiben die Abnahme der Elektronendichte mit dem Abstand vom Kern. Da in *OrbVis* nur Slater Type Orbitale verwendet werden, soll hier auch nur die Wellenfunktionen der Letzteren dargestellt werden:

$$\chi_i = a * e^{-\zeta r}. \quad (5.5)$$

In obiger Gleichung entspricht a dem Normierungsfaktor, ζ dem Orbitalkoeffizienten und r dem Abstand vom Atomkern. Letztendlich erhält man nun eine lösbare Matrixgleichung

$$FC = SC, \quad (5.6)$$

wobei F die Fockmatrix mit den Wechselwirkungen der Elektronen, S die Überlappungsmatrix und C die Matrix der Orbitalkoeffizienten ist.

Da bei einem Molekülorbital immer nach dem energetisch stabilsten Zustand gesucht wird, kann man nun in einem iterativen Prozess solange die Elemente in der Orbitalkoeffizientenmatrix C verändern, bis ein Energieminimum erreicht wird (Lösung des Eigenwertproblems der Matrix). Die korrespondierenden Molekülorbitalkoeffizienten beschreiben dann die Gestalt des Orbitals.

Semiempirische Verfahren, wie die in *OrbVis* verwendete Austin Method 1 (AM1)[26] basieren ebenfalls auf diesem allgemeinen Ansatz der Quantenchemie. Die Wellenfunktionen werden hier aber noch mit zusätzlichen Normierungsfaktoren ergänzt, damit die berechneten Energiewerte aus dem oben beschriebenen Verfahren möglichst nahe an den experimentell gemessenen Ergebnissen liegen. Diese Normierungsfaktoren fließen zusätzlich in den Lösungsansatz ein.

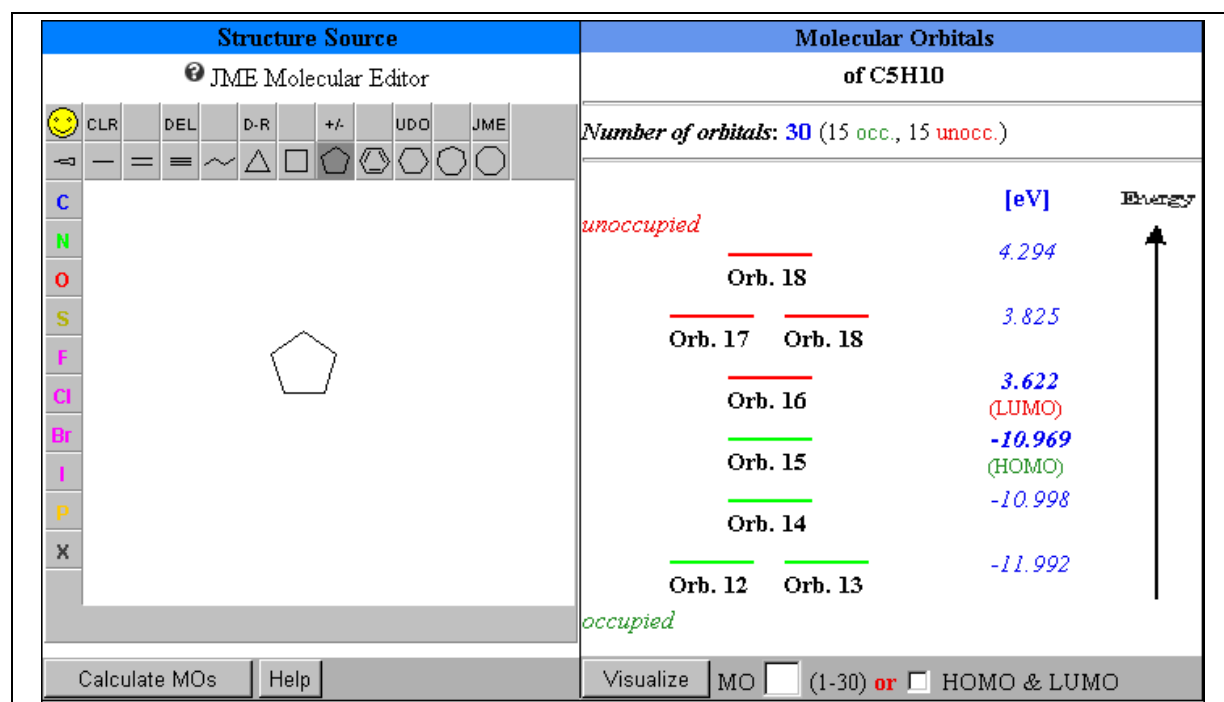


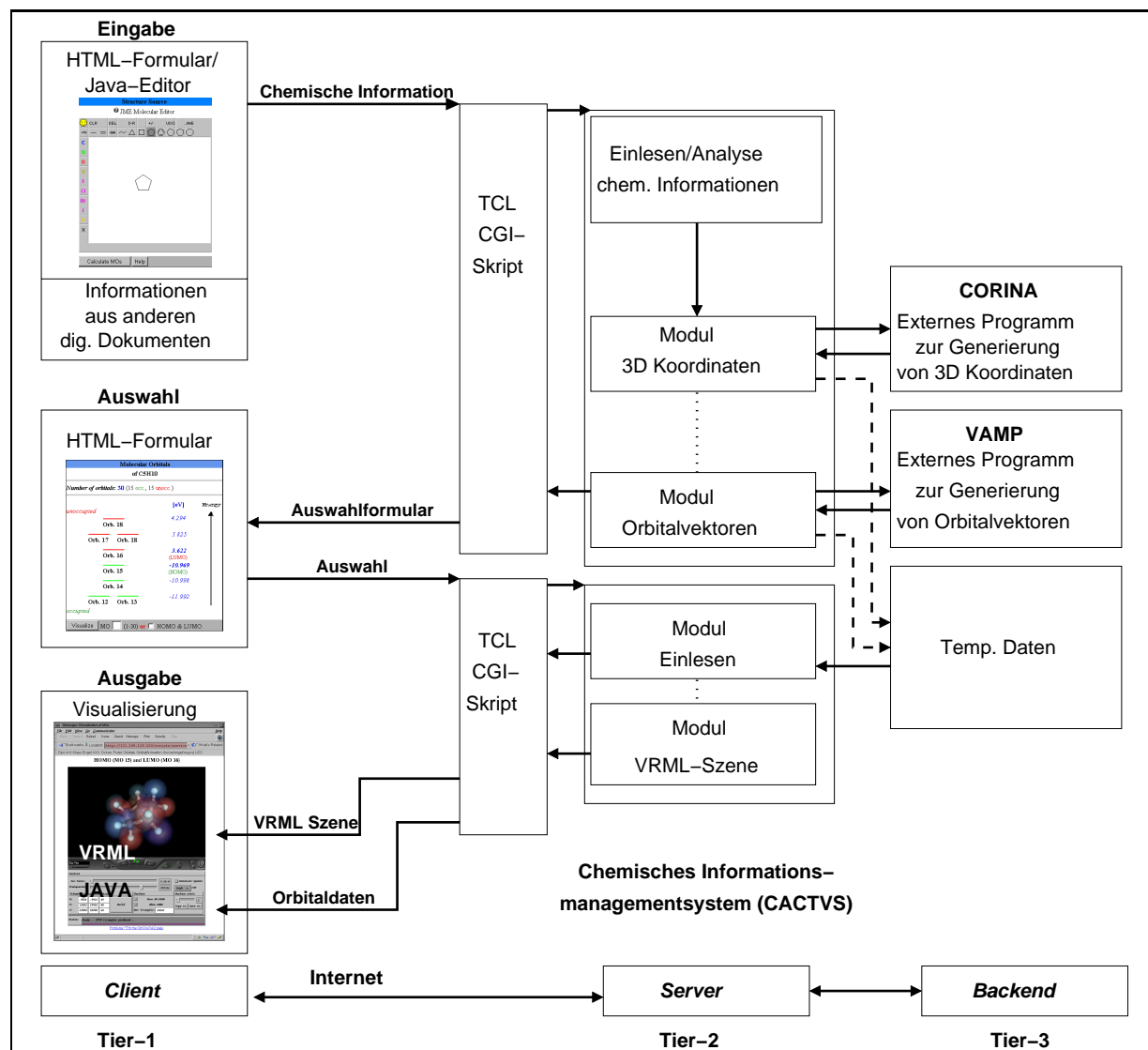
Abbildung 5.3: Nach Eingabe eines Moleküls mit dem Struktureditor (links), kann in der folgenden HTML-Seite das Orbital einer bestimmten Energiestufe zur Visualisierung gewählt werden (rechts).

Zur lokalen Berechnung des Volumens kommt bei *OrbVis* folgender Algorithmus zum Einsatz:

```

float snorm, pnorm, f;
Für alle Atome a des Moleküls {
  // Typ des Atoms als Ganzzahl
  int typ = a.typ;
  // Position des Atoms
  float ax=a.x, ay=a.y, az=a.z;
  // schlage semi-empirische Gewichtungsfaktoren nach (aus
  Tabellen)
  float zs = zs_am1[typ-1];
  float zp = zp_am1[typ-1];
  // schlage Orbitalkoeffizienten nach (vom Server)
  float koef[] = a.koef;
  Wenn typ==1 { // 1s Orbital (Helium und Wasserstoff)
    snorm =  $\sqrt{(zs)^3/\pi}$ ;
    pnorm = 0;
  }
  sonst { // 2s und 2p Orbital
    snorm =  $\sqrt{(zs)^5/(3*\pi)}$ ;
    pnorm =  $\sqrt{(zp)^5/\pi}$ ;
  }
  Für alle Raumpositionen x,y,z {
    // Berechne Abstand des Raumpunkts vom Atomzentrum
    float distanz =  $\sqrt{(ax-x)^2 + (ay-y)^2 + (az-z)^2}$ ;
    // Umrechnung auf atomare Einheit (Bohr)
    float abstand = distanz/0.53;
    // Berechne Wellenfunktion (siehe Gleichung 5.5)
    Wenn typ==1 { // 1s Orbital (Helium und Wasserstoff)
      f = koef[0] * snorm *  $e^{-zs*abstand}$ ;
    }
    sonst { // 2s und 2p Orbital
      f = koef[0] * abstand * snorm *  $e^{-zs*abstand}$  + (
        koef[1] * (ax-x) + koef[2] * (ay-y) + koef[3] * (az-z)
      ) / 0.57 * pnorm *  $e^{-zp*abstand}$ ;
    }
    // aktualisiere Skalarfeld
    Wenn f < 0
      volume[x][y][z] = volume[x][y][z] - f2;
    sonst
      volume[x][y][z] = volume[x][y][z] + f2;
  }
}

```

Abbildung 5.4: Schematische Darstellung einer *OrbVis* Visualisierungssitzung.

Eine typische Visualisierungssitzung mit *OrbVis* läuft nach folgendem Schema ab (siehe Abbildung 5.4): Der Benutzer übergibt entweder direkt mit einem Java-Editor chemische Strukturen an den Dienst (Abbildung 5.3, links) oder übernimmt diese aus einem anderen Web-Dienst. Alle weiteren chemischen Informationen, die für die Berechnung notwendig sind, müssen nicht vom Benutzer eingegeben werden, sondern werden von der CACTVS-Bibliothek automatisch erzeugt bzw. angefordert. So werden beispielsweise 3D-Koordinaten und alle wichtigen quantenchemischen Informationen durch Aufruf von entsprechenden kommerziellen, externen, Server-seitigen

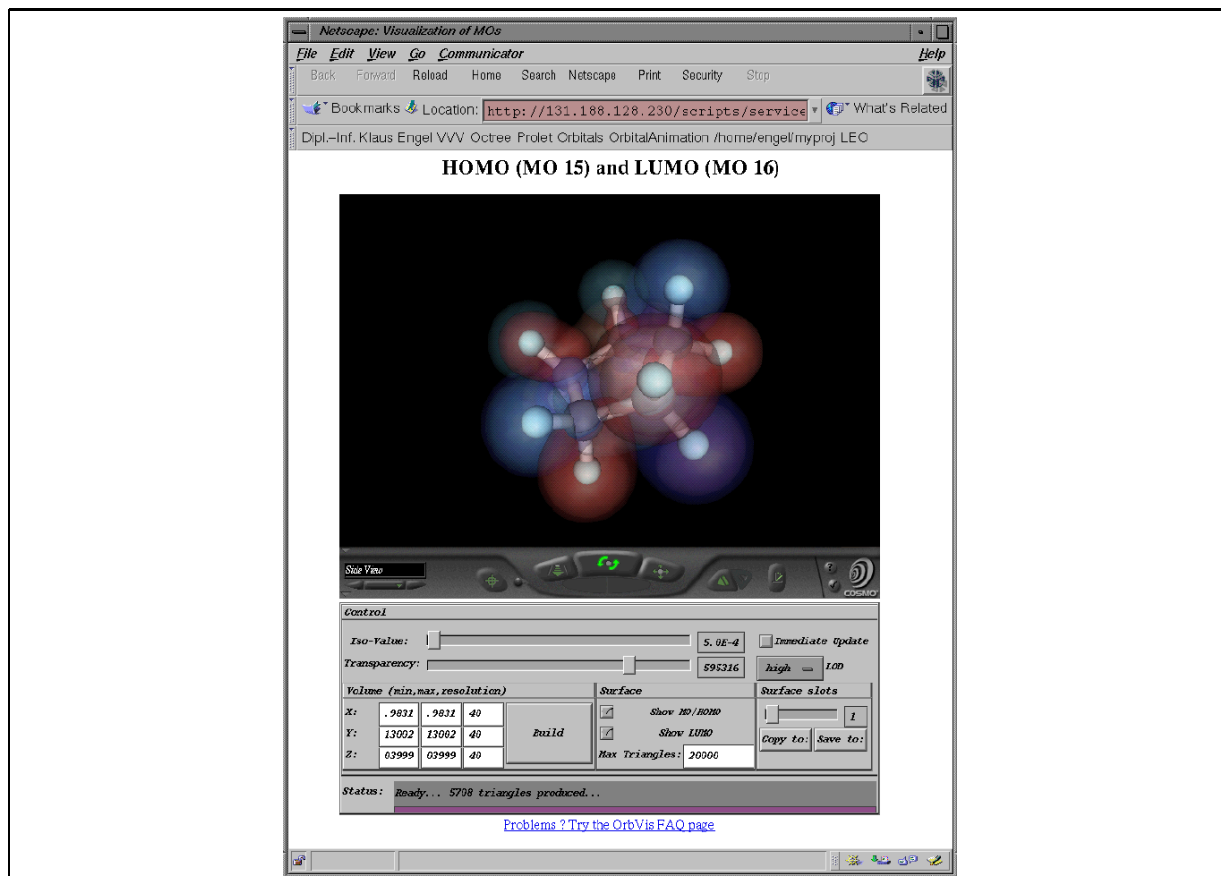


Abbildung 5.5: Visualisierung eines Molekülorbitals mit *OrbVis*. Das Java-Applet (unten) kontrolliert das VRML-Plugin (oben) über das External Authoring Interface. Dies ermöglicht eine dynamische Aktualisierung der Visualisierung.

Programmen[17] generiert. Nach der Datenanalyse wird dem Benutzer eine Übersicht der berechneten Daten im HTML-Format präsentiert (Abbildung 5.3, rechts).

Nachdem der Benutzer eine Auswahl getroffen hat, wird durch das CACTVS-System eine *Ball-Stick*-VRML-Szene des Moleküls generiert. Das mit dieser VRML-Szene in eine HTML-Seite integrierte Java-Applet nimmt dabei sowohl VRML-Szenen als auch Informationen über die molekularen Orbitalkoeffizienten entgegen. Dem Applet liegen damit alle chemischen Informationen zur Darstellung der Molekülorbitale vor, was eine komplette lokale Interaktion auf der Client-Seite ermöglicht. Die Berechnung der Volumendaten, die Generierung der Isoflächen durch den Marching-Cubes-Algorithmus, sowie die Kombination aller Daten in der Basisszene werden vollständig vom Applet übernommen.

Das Applet beinhaltet zwei Minimalbasisfunktionen zur Berechnung von STO-Molekülorbitalen. Eine der Wellenfunktion ist nur auf die Atome Wasserstoff und Helium begrenzt, da diese nur über s-Orbitalanteile verfügen. Alle anderen Atome werden durch die zweite Wellenfunktion

berechnet, in der auch die p-Orbitalanteile berücksichtigt werden. In diese Funktionen fließen zum einen die Normierungsfaktoren der AM1 Methode und zum anderen die vom Server berechneten Orbitalkoeffizienten für das entsprechende Molekülorbital ein. Aus diesen Informationen wird nun lokal ein skalarer Volumendatensatz auf einem kartesischen Gitter berechnet, wobei jeder Skalarwert die Elektronendichte an der entsprechenden Raumposition angibt. Aus diesem Volumen werden mit dem bereits vorgestellten Marching-Cubes-Modul Isoflächen extrahiert, die mit dem EAI in die Basis-VRML-Szene des VRML-Plugins eingebaut werden.

Diese Architektur beinhaltet mehrere Vorteile. Zum einen ist die Menge der über das Internet transportierten Daten klein, da die Berechnung der verhältnismäßig großen Volumendaten vollständig auf der Seite des Client durchgeführt werden kann. Diese Client-seitige Berechnung unterstützt dabei auch die gute 3D- und hohe Prozessor-Leistung heutiger Personalcomputer. Der maßgebende Vorteil der Applikation liegt jedoch in der Gewährleistung einer schnellen Reaktion auf Benutzereingaben und der damit einhergehenden bestmöglichen Interaktion mit den darzustellenden Daten. Der Benutzer kann z.B. den Detailierungsgrad und die Transparenz der Orbitale einstellen, einzelne Orbitale voneinander getrennt an- oder ausschalten oder aber die Grenzwerte für die Orbital-Isoflächen dynamisch ändern, so dass die Elektronendichteverteilung der Orbitale interaktiv verfolgt werden kann. Jeder Effekt, der durch eine benutzerabhängige Wertänderung hervorgerufen wird, wird sofort in der Basisszene angezeigt. Dies wird auch durch die relativ geringe Auflösung der Volumendaten und der damit einhergehenden geringen Zahl an extrahierten Polygonen erreicht. Da in der Elektronendichteverteilung typischer molekularer Strukturen keine hohen Frequenzen auftreten, ist eine hochqualitative Rekonstruktion und Darstellung der Orbitalflächen auch mit Volumendaten dieser Größe möglich.

5.1.5 Visualisierung Cryo-Elektronenmikroskopischer Dichten

Cryo-Elektronenmikroskopie-Daten spielen bei der Visualisierung von Proteinen eine große Rolle. Bei Proteinen handelt es sich um so genannte biologische Makromoleküle, die aus mehreren tausend bis zehntausend Atomen bestehen. Diese Makromoleküle können sich wieder zu komplexen Funktionseinheiten gruppieren. Eine solche Funktionseinheit ist beispielsweise das Ribosom, in dem der letzte Schritt bei der Übersetzung des genetischen Codes in Biomoleküle vollzogen wird - die Proteinsynthese. Es gibt bereits eine Vielzahl von Programmen, die solche Strukturen ausgehend von den Konnektivitätstabellen (atomare Informationen, 3D-Koordinaten der Atome) visualisieren. Die Information dafür erhält man aus Kristallen, die durch Röntgenstrahlen vermessen werden. Neu ist aber die Visualisierung der Proteinkomplexe mittels der Cryo-Elektronenmikroskopie. Diese Technologie ist recht jung und hat erst in den letzten Jahren solche Fortschritte gemacht, dass heute atomare Auflösungen erreicht werden.

Hierbei liegt die Information nicht in Form der atomaren Strukturen vor, sondern in Form eines Volumens, das durch das Verfahren reproduziert wird. Mit anderen Worten erhält man ein echtes Abbild des Makrokomplexes im nativen Zustand. Unter dem nativem Zustand versteht man, dass das Protein in der Konfiguration gewonnen wird, wie es auch in der Zelle (also seiner natürlichen Umgebung) vorliegt. Diesem nativen Zustand kommt deshalb eine so hohe Bedeutung zu, da so die Funktion des Proteins effektiver erforscht werden kann. Bisherige Ansätze zeigten zwar die atomare Information an, waren aber eher modellhaft, da in einem Kristall

natürlich andere Kräfte herrschen als in der natürlichen, wässrigen Umgebung der Zelle. Man erhält also keine naturgetreue Anordnung der einzelnen Segmente. Genau diese naturgetreue Anordnung der einzelnen Segmente macht aber die Cryo-Elektronenmikroskopie so interessant. Bei der Technik der Cryo-Elektronenmikroskopie wird eine Flüssigkeit mit den zu beobachtenden Proteinen schockgefroren. Der Prozess vollzieht dermaßen rapide, dass sich die Proteine nicht mehr strukturell zu verändern können. Sie werden also quasi in exakt derjenigen Form eingefroren, in der sie auch vorher im flüssigem Medium vorlagen. Nun wird dieses schockgefrorene Medium, in dem eine Vielzahl zufällig orientierter Proteine vorliegen, einer Elektronenmikroskopie unterzogen. Aus den verschiedenen Ansichten der Proteine lässt sich so die 3D-Struktur rekonstruieren.

Vorteilhaft bei dieser Gewinnung der Struktur von Makromolekülen ist nun, dass man auf diesen Volumen die wichtigen Elemente des Proteins wiedererkennt und unter Zuhilfenahme der atomaren Information eine exakte Rekonstruktion des realen Zustandes erhält.

Solche Volumendaten aus der Cryo-Elektronenmikroskopie können ebenfalls mit dem in Abschnitt 5.1.1 vorgestellten Ansatz in digitale Dokumente eingebettet werden. Im Gegensatz zu dem Visualisierungsdienst *OrbVis* werden dazu die Volumendaten von einem Datenbankserver zum Client transferiert. Daraufhin wird wiederum lokal aus den Daten eine Isofläche mit Hilfe des Java-Applets rekonstruiert, die unter Ausnutzung der lokalen Graphikhardware dargestellt wird (siehe Abbildung 5.6).

Die ersten verfügbaren Volumendaten aus der Cryo-Elektronenmikroskopie hatten noch eine relativ geringe Auflösung, so dass die Anzahl der rekonstruierten geometrischen Primitive noch handhabbar war. Neueste Fortschritte in der Cryo-Elektronenmikroskopie ermöglichen aber zunehmend höhere Abstraten, so dass der hier vorgestellte Ansatz schnell an seine Grenzen stößt. Im weiteren Verlauf dieses Kapitels werden deshalb Techniken zur Sprache kommen, die mittels adaptiver und progressiver Techniken die Anzahl der geometrischen Primitive begrenzen und so auch Volumendaten mit höherer Auflösung handhabbar machen.

5.1.6 Ergebnisse

Standardmäßig berechnet der Web-Dienst *OrbVis* aus den vom Server empfangenen Orbitalkoeffizienten Volumendaten mit einer Auflösung von 40^3 Voxeln. Diese Auflösung hat sich für üblich dimensionierte Moleküle als ausreichend herausgestellt, da der Funktionsverlauf des 3D-Skalarfelds im Allgemeinen relativ „glatt“ ist. Natürlich kann der Benutzer jederzeit die Abstrategie erhöhen. Die Zahl der aus diesem Skalarfeld rekonstruierten geometrischen Primitive liegt im Bereich mehrerer tausend bis mehrerer zehntausend. Unter diesen Voraussetzungen ist sowohl ein Wechsel des Isowertes als auch die Untersuchung der generierten 3D-Szene mit heute üblichen Arbeitsplatzrechnern in interaktiven Raten möglich. Je nach Komplexität der generierten Isofläche sind Bildwiederholraten von über 20 Bildern pro Sekunde auf nahezu jeder heute üblichen Plattform zu erreichen. Bei Wechsel des Isowerts wird die Szene innerhalb von weniger als einer halben Sekunde aktualisiert. Die Akzeptanz des Web-Dienstes zeigt sich auch an den hohen Nutzungszahlen. Innerhalb von ungefähr zwei Jahren wurde *OrbVis* bereits über 8000 mal aufgerufen.

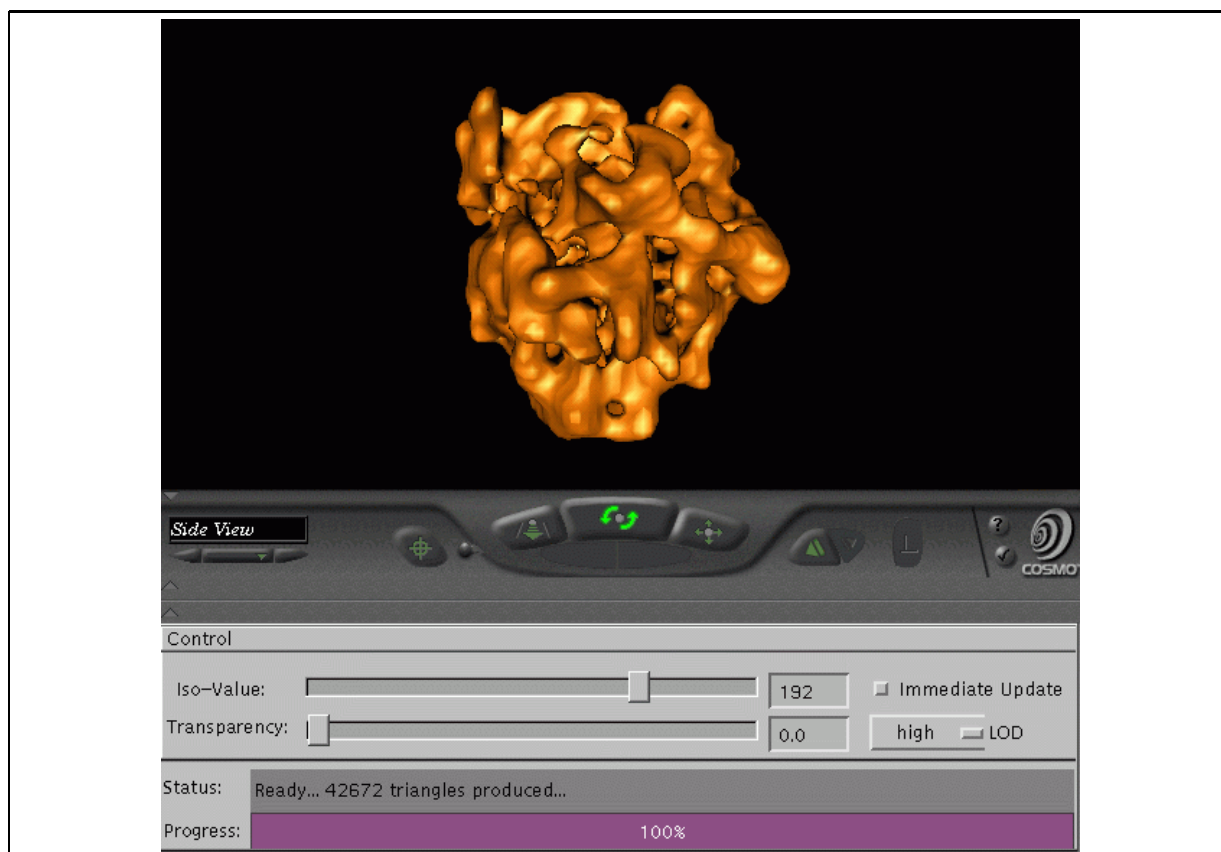


Abbildung 5.6: Visualisierung einer Ribosomoberfläche. Die Isofläche wird aus den von einem Server übertragenen Volumendaten durch das Java-Applet (unten) erzeugt. Dieses führt lokal einen Marching-Cubes-Algorithmus aus und übergibt die so gewonnene Geometrie an das VRML-Plugin (oben) weiter.

Auch für cryo-elektronenmikroskopische Datensätze wurden anfangs mit diesem Ansatz befriedigende Ergebnisse erzielt. Die Auflösung dieser Daten und die daraus rekonstruierte Zahl der geometrischen Primitive war mit 72^3 Voxeln bzw. ≈ 40000 Dreiecken relativ begrenzt, so dass sich wieder interaktive Bildwiederhol- und Szenenaktualisierungsraten ergaben. Neuere Datensätze, die aus weiteren Fortschritten in der Cryo-Elektronenmikroskopie resultieren, haben bereits eine Auflösung von 160^3 Voxeln. Die sich daraus ergebende Zahl von ≈ 300000 Dreiecken zeigt die Grenzen dieses Ansatzes. Sowohl die Darstellungsgeschwindigkeit des VRML-Plugins als auch die Verarbeitungsgeschwindigkeit des Java-Applets sind mit dieser Datenmenge deutlich überlastet. Auch zeigt sich, dass das EAI für das Versenden einfacher Nachrichten und nicht für die Übergabe derartig großer Datenmengen konzipiert wurde.

In den nächsten Jahren sind im Bereich der Cryo-Elektronenmikroskopie noch weitere Fortschritte zu erwarten, so dass die Auflösung der Daten nochmals erheblich anwachsen dürfte. Diese letzten Ergebnisse machen die Notwendigkeit „intelligenter“ Algorithmen deutlich, welche die

Zahl der erzeugten geometrischen Primitive schon im Vorfeld reduzieren. Solche Algorithmen sollen im Folgenden erläutert werden.

5.2 Progressive Isoflächen

Der im letzten Abschnitt vorgestellte Ansatz zur lokalen, plattform-unabhängigen Extraktion und Darstellung von Isoflächen aus Volumendaten stößt mit dem Fortschritt in der Sensor- und Simulationstechnik schnell an seine Grenzen, da mit der steigenden Auflösung der resultierenden Volumendaten auch moderne Arbeitsplatzrechner an ihre Speicher- und Verarbeitungskapazität stoßen. Dies ist vor allem auf die Eigenschaft des Marching-Cubes-Algorithmus zurückzuführen, pro Volumenzelle bis zu fünf Dreiecke zu rekonstruieren. Bei mehreren hundert Millionen Zellen, die bei heutigen Volumendaten durchaus üblich sind, ergibt sich eine enorme Zahl von Dreiecken, die weder gespeichert noch mit heutiger Graphikhardware interaktiv dargestellt werden kann. Deshalb werden sowohl in diesem als auch im folgenden Abschnitt Algorithmen betrachtet, die bereits im Vorfeld die Anzahl der geometrischen Primitive begrenzen.

5.2.1 Einleitung

Im Gegensatz zu dem im letzten Abschnitt beschriebenen Verfahren zur Extraktion und Darstellung einer Isofläche in einer festen Auflösungsstufe stehen Algorithmen, die durch eine hierarchische Zerlegung des Definitionsbereiches einer Funktion eine Darstellung mit unterschiedlichen Auflösungsstufen (Multi-Resolution-Repräsentationen) ermöglichen. Dadurch werden hohe Kompressionsraten, progressive Übertragung und Darstellung sowie *level-of-detail* (LOD) Repräsentationen möglich, die sowohl die Vorverarbeitung (Pre-Processing) als auch die Visualisierung beschleunigen können.

Im Folgenden wird ein Web-basierter Dienst zur progressiven Extraktion, Transmission und Darstellung von Isoflächen aus einer Multi-Resolution-Repräsentation eines Volumendatensatzes vorgestellt[33]. Der Dienst besteht aus einem Visualisierungsserver, der auf vorverarbeitete Datensätze Zugriff hat, die in einer Multi-Resolution-Repräsentation vorliegen. Mittels eines Java-Applets kontaktiert ein Client diesen Dienst und veranlasst das Laden eines bestimmten Datensatzes. Aus diesem Datensatz extrahiert der Server auf Anforderung des Client eine Isofläche in der gewünschten Auflösungsstufe und transferiert die erzeugten Polygone zum Client. Das Java-Applet auf Client-Seite gibt die empfangenen Polygone an eine Darstellungskomponente weiter, die entweder mittels eines VRML-Plugins oder mit Java3D die Flächen unter Ausnutzung lokaler Rasterisierungshardware darstellt. Ein Benutzer kann mit einer LOD-Kontrolle weitere Details der Isofläche anfordern, woraufhin zusätzliche Polygone vom Server extrahiert und transferiert werden. Durch die LOD-Kontrolle sowie den progressiven Eigenschaften dieses Ansatzes kann sowohl die Menge der übertragenen Daten minimiert als auch die Zahl der auf Client-Seite darzustellenden Polygone in Grenzen gehalten werden, so dass eine interaktive Visualisierung hochaufgelöster Volumendaten auf heutigen Web-Clients möglich wird.

5.2.2 Multi-Resolution Volumenrepräsentation

In [52, 48, 51, 50] wurden Methoden basierend auf Multi-Level Finite-Elemente Approximationen entwickelt, die die Erzeugung von Repräsentationen eines gegebenen Datensatzes mit multiplen Auflösungsstufen durch lokale Gitterverfeinerung ermöglichen. Im Falle von Dreiecksgittern basiert die Erzeugung von LODs auf der Vereinfachung des Gitters durch die Elimination von geometrischen Primitiven wie Vertices und Kanten [123, 126, 65], wobei geometrische Aspekte wie die Erhaltung der allgemeinen Form oder von Details im Vordergrund stehen. Bei dreidimensionalen Skalarfeldern steht dagegen eine möglichst genaue Approximation einer Funktion im Zentrum des Interesses. Deshalb wird hier von einer groben Unterteilung des Definitionsbereichs des Volumens ausgegangen, mit deren Hilfe die beste Approximation der Eingabedaten auf der Basis einer gegebenen Norm berechnet wird. Typische Normen, die in der Computergraphik benutzt werden, sind die L_p Normen mit $1 \leq p \leq \infty$ Normen. Es können allerdings auch generellere Normen zum Einsatz kommen. Im nächsten Schritt wird der lokale Beitrag jedes Elements zum globalen Approximationsfehler ermittelt und diejenigen Elemente für eine Verfeinerung markiert, deren lokaler Approximationsfehler über einem gewissen Grenzwert liegt. Das Gitter wird also adaptiv durch die lokale Unterteilung einzelner Elemente verfeinert, was in jedem Schritt zu einer besseren Approximation der Funktion führt. Durch die Iteration dieser Schritte ergeben sich ineinander geschachtelte Räume, welche die Funktion im Bezug auf das Fehlermaß optimal für eine Analyse oder die Visualisierung approximieren.

Der Algorithmus zur adaptiven Gitterverfeinerung basiert auf der Unterteilung des Definitionsbereichs des Volumens in zwei reguläre Polyeder, genauer gesagt in Tetraeder und Oktaeder. Die regulären Verfeinerungsregeln erzeugen translierte und skalierte Nachkommen der initialen Polyeder. Die reguläre Unterteilungsregel für Tetraeder unterteilt ein Element in vier Tetraeder und einen Oktaeder, indem auf jeder Fläche die Mittelpunkte der Kanten miteinander verbunden werden. Die reguläre Unterteilungsregel für Oktaeder unterteilt ein Element in sechs Oktaeder, indem die Mittelpunkte der Kanten jeder Fläche und alle Kantenmittelpunkte mit dem Schwerpunkt des Vatelements verbunden werden. Zusätzlich werden acht Tetraeder erzeugt, indem die Ecken der Dreiecke in der Mitte der Flächen des Vatelements mit dessen Schwerpunkt verbunden werden. Die regulären Verfeinerungsregeln können auf benachbarte Elemente ohne Konsistenzprobleme angewendet werden, d.h. es werden keine hängenden Knoten auftreten. Im Falle einer adaptiven Unterteilung wird allerdings nur ein Teil der Elemente regulär unterteilt, weshalb die Partition durch irreguläre Unterteilung der Transitionselemente geschlossen werden muss. Um nur Tetraeder irregulär unterteilen zu müssen, werden Oktaeder, die eigentlich irregulär unterteilt werden müssen, zunächst in acht Tetraeder unterteilt. Die irreguläre Unterteilung wird daraufhin nur auf die resultierenden Tetraeder angewendet. Da ein Tetraeder sechs Kanten besitzt gibt es $2^6 = 64$ mögliche Verfeinerungsmuster. Aus Symmetriegründen können 11 verschiedene Verfeinerungsregeln klassifiziert werden, von denen zwei der vollen und leeren Verfeinerung entsprechen. In bestimmten Konfigurationen müssen auch benachbarte Elemente in Betracht gezogen werden.

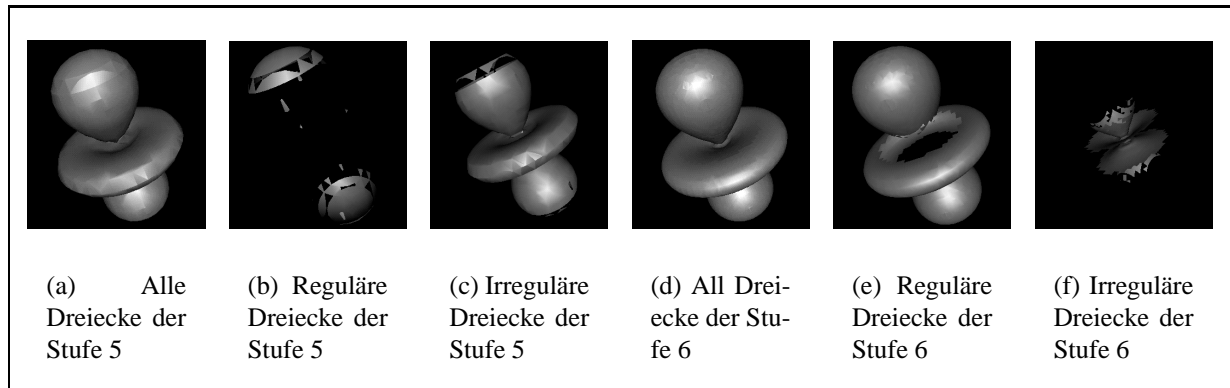


Abbildung 5.7: Reguliäre und irreguläre Dreiecke zweier Stufen der Isofläche der sphärischen harmonischen Funktion.

5.2.3 Progressive Isoflächenextraktion

Mittels der im vorigen Abschnitt vorgestellten Regeln kann nun eine Multi-Resolution-Repräsentation des Volumens erzeugt werden. Aus dieser Darstellung können progressiv, also mit fortschreitendem Detaillierungsgrad, Isoflächen generiert werden. Ebenso kann eine bestehende Isofläche weiter verfeinert werden. Es ergeben sich zwei Algorithmen, die beide stufenweise arbeiten und die sich ergebenden Dreiecke in einer Verfeinerungsstufe der Fläche speichern. Wird der Isowert geändert, so ist es notwendig, eine komplett neue Isofläche zu rekonstruieren. Wenn die Isofläche bis zu einer gegebenen Stufe verfeinert werden soll, so werden allen Stufen kleiner oder gleich der gewünschten Stufe abgearbeitet, wobei mit der kleinsten Stufe begonnen wird. In jeder der Stufen untersucht der Algorithmus die Elemente und speichert die Dreiecke in einer Verfeinerungsstufe ab. Dreiecke, die aus Elementen resultieren, für die weitere Verfeinerungen existieren, werden als *irregulär* markiert, da sie in feineren Stufen der Isofläche ersetzt werden. Dreiecke aus nicht weiter verfeinerten Elementen werden als *regulär* markiert, da sie in feineren Stufen der Isofläche erhalten bleiben (siehe Abbildung 5.7).

5.2.4 Progressive Isoflächenvisualisierung im WWW

Ein progressiver Algorithmus zur Extraktion von Isoflächen aus hochauflösten Volumendaten, wie im vorausgegangenen Abschnitt beschrieben, bietet sich natürlich direkt für einen Web-basierten Visualisierungsservice an, da in diesem Umfeld sowohl begrenzte Bandbreiten als auch limitierte Verarbeitungskapazitäten lokaler Arbeitsplatzrechner beachtet werden müssen. Ein Benutzer kann sich zunächst einen groben Überblick über die globale Form der Isofläche verschaffen und durch die geringe Zahl der zu übertragenden und darzustellenden geometrischen Primitive relativ interaktiv nach einem geeigneten Isowert suchen. Bei Bedarf kann der Benutzer mittels der LOD-Kontrolle weitere Details der Fläche anfordern.

Der Visualisierungsservice ist über eine HTML-Seite nutzbar, in die ein Java-Applet ein-

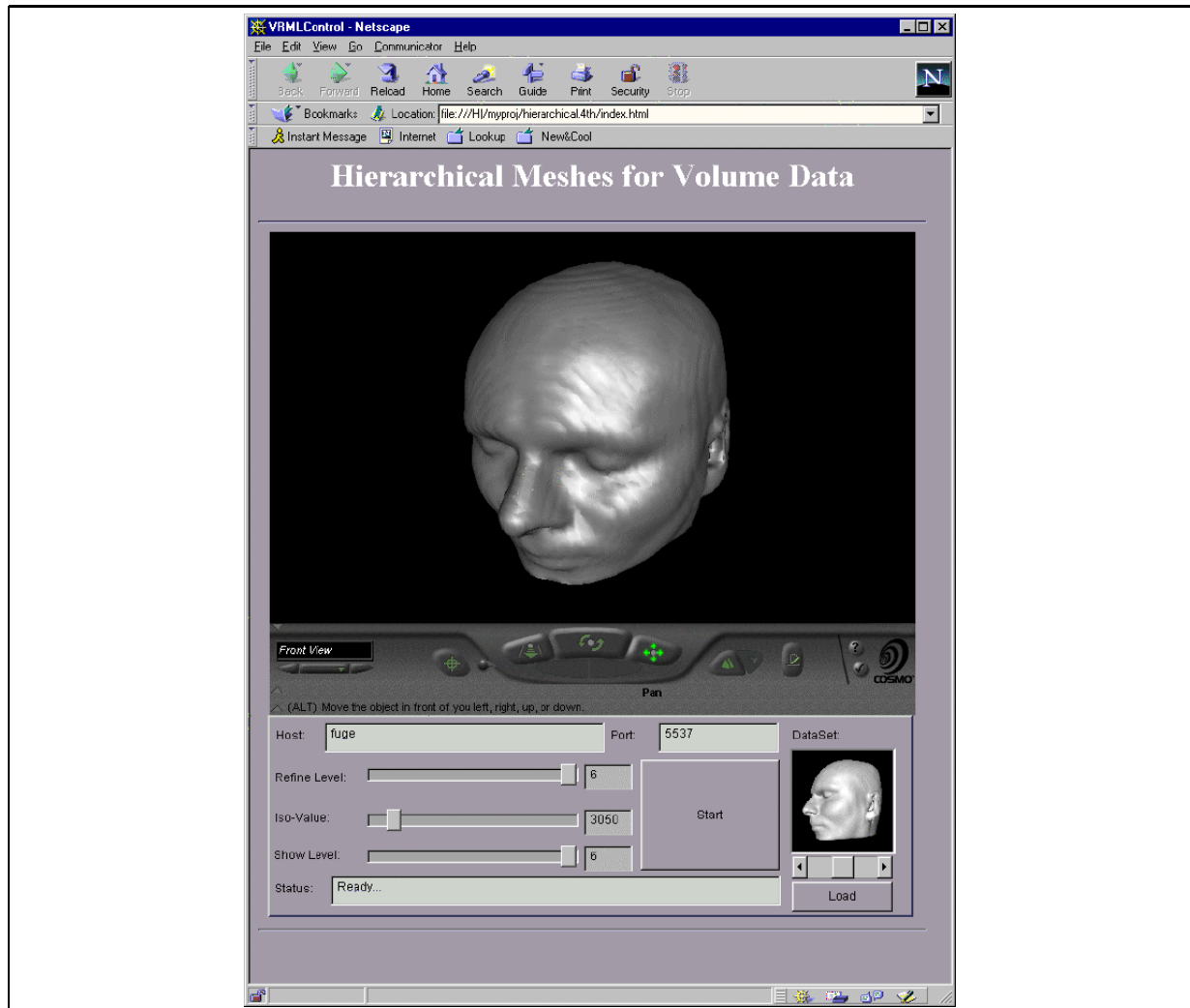


Abbildung 5.8: Progressive Isoflächenvisualisierung in einem Web-Browser. Das Java-Applet(unten) empfängt die progressiven Flächendaten und reicht diese mittels des External Authoring Interfaces an das VRML-Plugin(oben) weiter.

gebettet ist, das die Benutzeroberfläche mit der LOD-Kontrolle als wichtigstes Element zur Verfügung stellt. Weiterhin wird zur Ausnutzung lokal vorhandener Graphikhardware entweder ein VRML-Plugin oder alternativ ein Java3D Rendering-Kontext genutzt, die ebenfalls in die Web-Seite integriert sind (siehe Abbildung 5.8). Das Java-Applet stellt eine TCP-Socket-basierte Netzwerkverbindung zum Visualisierungsserver her, über die mit einem speziellen Protokoll die Anforderungen des Clients und Antworten des Visualisierungsservers ausgetauscht werden. Dieses einfache Protokoll besteht im Wesentlichen aus vier Client-Server- und drei Server-Client-Befehlen:

- Client-Server-Befehle

- `Load_Dataset (int i)`: Veranlasst den Server, einen bestimmten Datensatz `i` zu laden.
- `Compute_Isosurface (int isovalue, int level)`: Der Server extrahiert eine neue Isofläche mit Isowert `i` von der Anfangsstufe 0 bis zu einer gegebenen Stufe `k`.
- `Refine_Isosurface (int k)`: Verfeinert die Isofläche bis zu einer gegebenen Stufe `k`.
- `Close_Connection`: Schließt die Verbindung.

- Server-Client-Befehle

- `Sending_Maximum_Level (int i)`: Teilt dem Client die maximale Verfeinerungsstufe der Isofläche mit.
- `Sending_Isosurface (isosurface s)`: Sendet eine neue Isofläche bis zur gewünschten Stufe.
- `Refining_Isosurface (isosurface s)`: Verfeinert eine Isofläche bis zu einer gewünschten Stufe.

Zunächst veranlasst der Benutzer den Server, einen bestimmten Datensatz zu laden, indem er den entsprechenden Datensatz in der Benutzeroberfläche des Java-Applets wählt (`Load_Dataset`). Der Server lädt daraufhin den gewünschten Datensatz und schickt die maximale Verfeinerungsstufe des Datensatzes zurück an den Client (`Sending_Maximum_Level`). Der LOD-Slider in der Java-Benutzeroberfläche wird angepasst. Der Benutzer wählt eine Detailstufe und einen Isowert und fordert die Isofläche an (`Compute_Isosurface`). Der Server startet daraufhin die Extraktion der Isofläche bis zu der gewünschten Stufe und schickt die generierten Dreiecke zum Client zurück (`Sending_Isosurface`).

Wird eine höhere Auflösungsstufe vom Benutzer gewünscht, so wird eine Anforderung mit der neuen Stufe vom Client an den Server gesendet (`Refine_Isosurface`). Der Server verfeinert die Isofläche bis zu der gegebenen Stufe und schickt die zusätzlichen Dreiecke zurück zum Client (`Refining_Isosurface`). Werden bei einer Verfeinerung nur die neu hinzugekommenen Details übertragen, so ist jedoch bei einer Änderung des Isowerts eine Übertragung von der größten Stufe an notwendig.

Auf Client-Seite werden die empfangenen Dreiecksdaten in eine VRML-Basiszene oder einen Java3D Szenengraph eingefügt. Die VRML-Basiszene wird zu Beginn der Sitzung in das VRML-Plugin geladen. Die Szenengraphen speichern alle Verfeinerungsstufen der Isofläche (siehe Abbildungen 5.9 und 5.10), d.h. nachdem eine Isofläche bis zu einer gewünschten Stufe übertragen wurde, sind weiterhin alle Verfeinerungsstufen bis zur höchsten übertragenen Stufe ohne erneuten Datentransfer abrufbar. Dadurch kann eine bestimmte Bildwiederholrate durch Zurückschalten auf eine gröbere Detailstufe gewährleistet werden. Dies wird durch einen Umschaltknoten (`levelSwitch`) erreicht, der als Kindknoten die Gruppenknoten der einzelnen

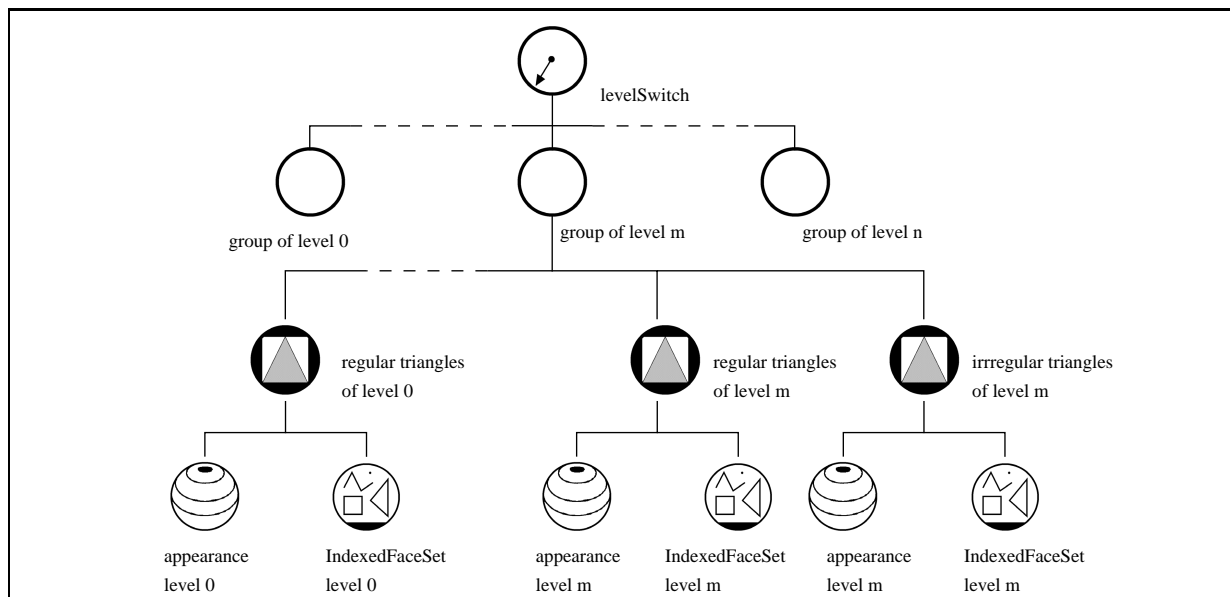


Abbildung 5.9: VRML-Szenengraph zur Darstellung progressiver Isoflächen. Die `IndexedFaceSet`-Knoten werden mit den vom Server empfangenen Dreiecksdaten gefüllt.

Stufen des Isofläche aufnimmt. Unter dem Gruppenknoten der Stufe m sind sowohl die irregulären Dreiecke der Stufe m als auch die regulären Dreiecke der Stufen $0 \dots m$ in Shape-Knoten untergebracht, die jeweils eine Liste von Dreiecken in `IndexedFaceSet`-(VRML) oder `TriangleArray`-(Java3D) Knoten aufnehmen können. In Java3D können diese Knoten komfortabel über die Szenengraphprogrammierschnittstelle mit den empfangenen Dreiecken gefüllt werden. Bei Nutzung von VRML als Darstellungskomponente werden, nachdem die maximale Verfeinerungsstufe der Isofläche bekannt ist, Shape-Knoten mit leeren `IndexedFaceSet`-Knoten mit der EAI-Methode `createVRMLFromstring` erzeugt, die später reguläre und irreguläre Dreiecke aufnehmen. Bei der Übertragung von Dreiecken der Isofläche vom Server sind Gruppen von Dreiecken immer als regulär oder irregulär markiert. Diese werden mittels EAI-Events an die entsprechenden `IndexedFaceSet`-Knoten gesendet und dort eingefügt. Sowohl Java3D als auch VRML ermöglicht somit eine effiziente Aktualisierung des Szenengraphen mit den vom Server empfangenen Dreiecksdaten.

5.2.5 Ergebnisse

Der in diesem Abschnitt vorgestellte progressive Ansatz zur Rekonstruktion, Übertragung und Darstellung von Isoflächen ermöglicht nun auch die Visualisierung höher aufgelöster Volumendaten. Dies wird durch die Bereitstellung einer Level-of-Detail-Kontrolle erreicht, mittels derer ein Benutzer jederzeit den Verfeinerungsgrad der Isofläche bestimmen kann. Bei Anforderung

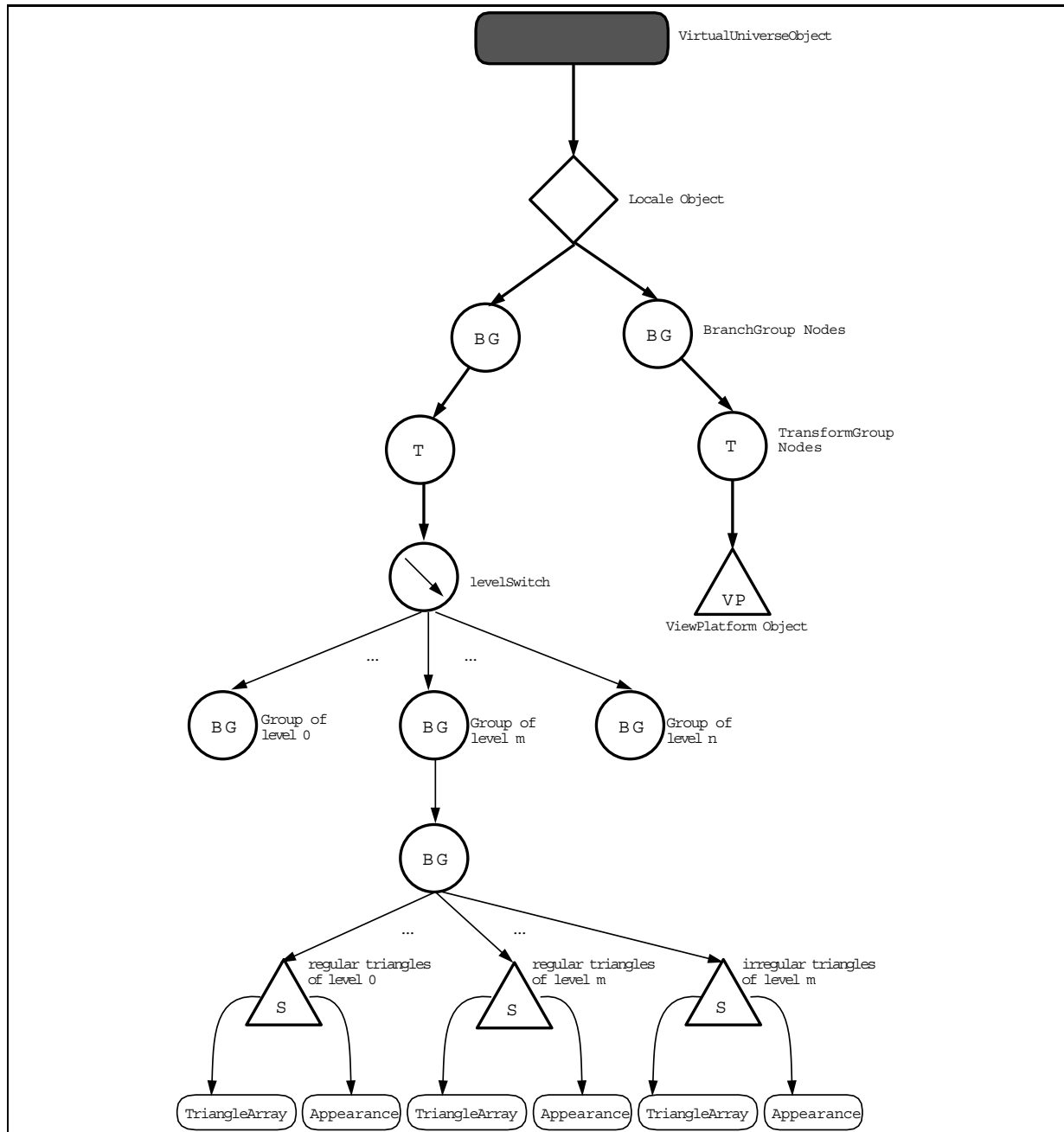


Abbildung 5.10: Java3D-Szenengraph zur Darstellung progressiver Isoflächen. Die `TriangleArray`-Knoten werden mit den vom Server empfangenen Dreiecksdaten gefüllt. Die progressiven Eigenschaften des Dreiecksnetzes bleiben auch auf der Client-Seite erhalten, d.h. alle Detailstufen der Isofläche sind bis zur übertragenen Maximalstufe über einen *Switch*-Knoten abrufbar.

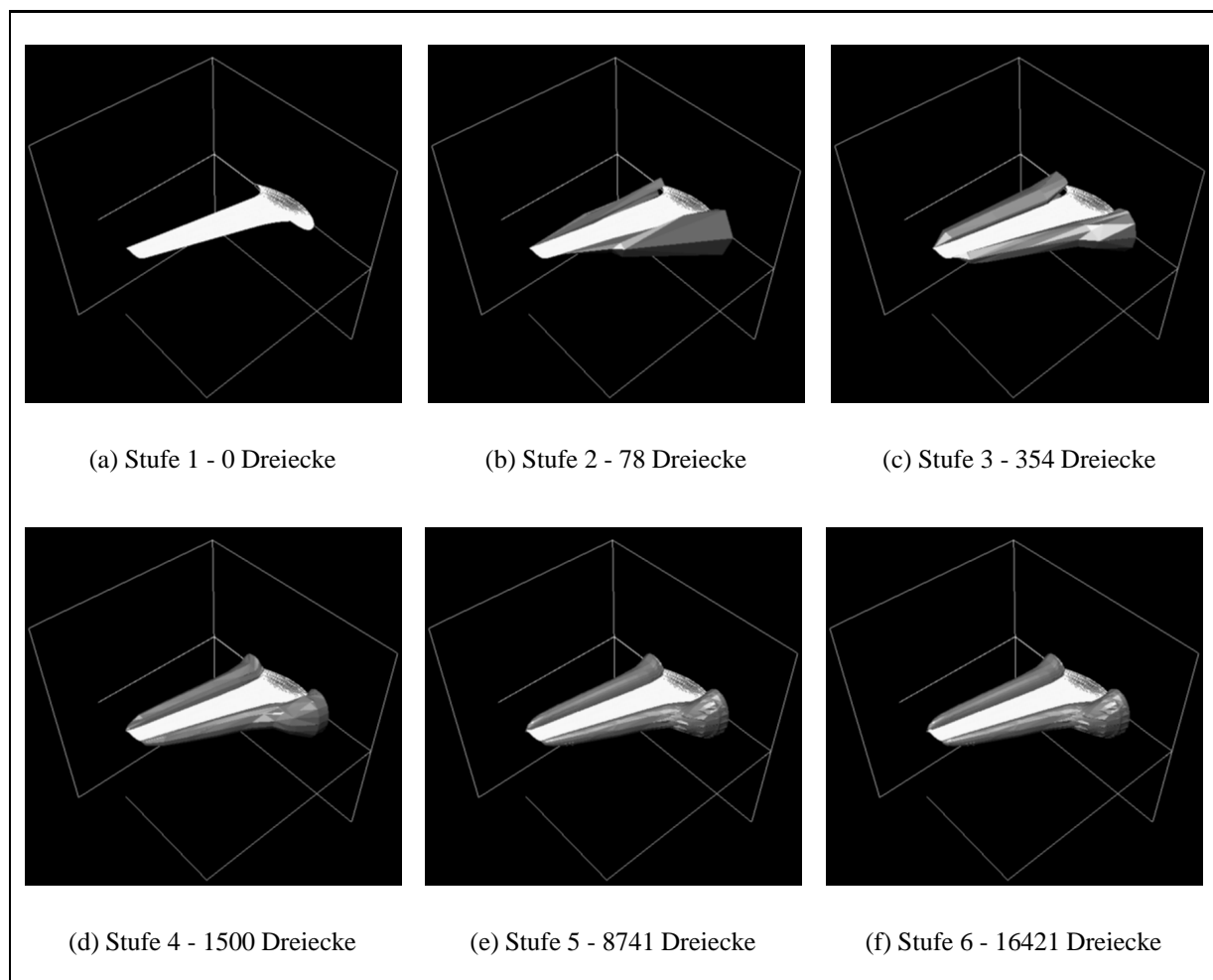


Abbildung 5.11: Darstellung des Druckes um einen Flugzeugflügel mittels einer progressiven Isofläche (Isowert 1.1).

einer höheren Verfeinerungsstufe werden ausschließlich Dreiecksdaten aus weiter verfeinerten Bereichen der Isofläche übertragen. Die folgenden Ergebnisse basieren auf der progressiven Visualisierung der Strömung um einen Flugzeugflügel (siehe Abbildung 5.11).

In Stufe 4 ist die globale Form der Isofläche bereits gut zu erkennen, obwohl weniger als einem Zehntel der Dreiecke der höchsten Stufe dargestellt und übertragen werden müssen. Die Menge der dargestellten Dreiecke und die übertragene Datenmenge ist in Tabelle 5.1 aufgeführt. Daraus wird ersichtlich, dass auch die Datenübertragungsmenge in der Stufe 4 annähernd ein Zehntel derer der feinsten Stufe entspricht.

Aus dieser Reduktion von Dreieckszahl und Datenmenge resultieren sowohl ein rascher Transfer der Daten auch über Netzwerkverbindungen mit niedriger Bandbreite als auch hohe Bildwiederholraten für die lokale Untersuchung der Fläche. Ein Benutzer ist also in der Lage, sich schnell einen globalen Überblick über die Form und Struktur der Isofläche zu verschaffen,

Stufe	Δ Dreiecke ^a	Indizes ^b	Dreiecke ^c	Daten ^d	Summe ^e
0	2	2	2	40	40
1	6	6	6	120	160
2	78	78	78	1560	1720
3	354	354	354	7080	8800
4	1500	1076	1500	29152	37952
5	8317	1171	8741	151836	189788
6	8745	1881	16421	157410	347198

^aAnzahl Dreiecke dieser Stufe

^bAnzahl Indizes dieser Stufe (Konnektivität)

^cSumme der Dreiecke in dieser Stufe

^dDatenmenge in dieser Stufe (Bytes)

^eDatenmenge insgesamt transferiert bis zu dieser Stufe
(Bytes)

Tabelle 5.1: Übertragene Datenmenge für eine progressive Darstellung des Flugzeugflügel-Datensatzes mit einem Isowert von 1.1.

was sich vor allem zur Suche nach einem interessanten Isowert eignet. Sobald dieser Isowert gefunden wurde, können weitere Details der Fläche angefordert werden. Dieser Ansatz eignet sich somit vor allem für hochaufgelöste Datensätze, deren Größe sowohl den Transfer des Datensatzes zu einem Arbeitsplatzrechner als auch eine lokale Rekonstruktion der Isofläche unmöglich macht.

5.3 Adaptive Echtzeit-Isoflächen

Eine progressive Isoflächenvisualisierung, wie sie im letzten Abschnitt besprochen wurde, bietet wesentliche Vorteile gegenüber der einfachen Methode aus dem ersten Abschnitt dieses Kapitels. Ein wesentlicher Nachteil des progressiven Ansatzes ist allerdings, dass die Isofläche immer als Ganzes verfeinert wird. Damit ist für eine Detailsuche der Transfer der kompletten Fläche bis zur höchsten Auflösungsstufe notwendig, wodurch der wesentliche Vorteil, nämlich die progressive Rekonstruktion, Übertragung und Darstellung der Isofläche, verloren geht.

Aus diesem Grunde wird im Folgenden ein Algorithmus vorgestellt, der eine selektive Level-of-Detail Darstellung von Isoflächen über eine Fokussierung einzelner Bereiche des Definitionsbereichs des Volumens erlaubt. Desweiteren stehen Verteilungsstrategien für die Isoflächenvisualisierung und Kompressionsmethoden durch die direkte Extraktion von Dreiecksstreifen (*Triangle Strips*) im Vordergrund.

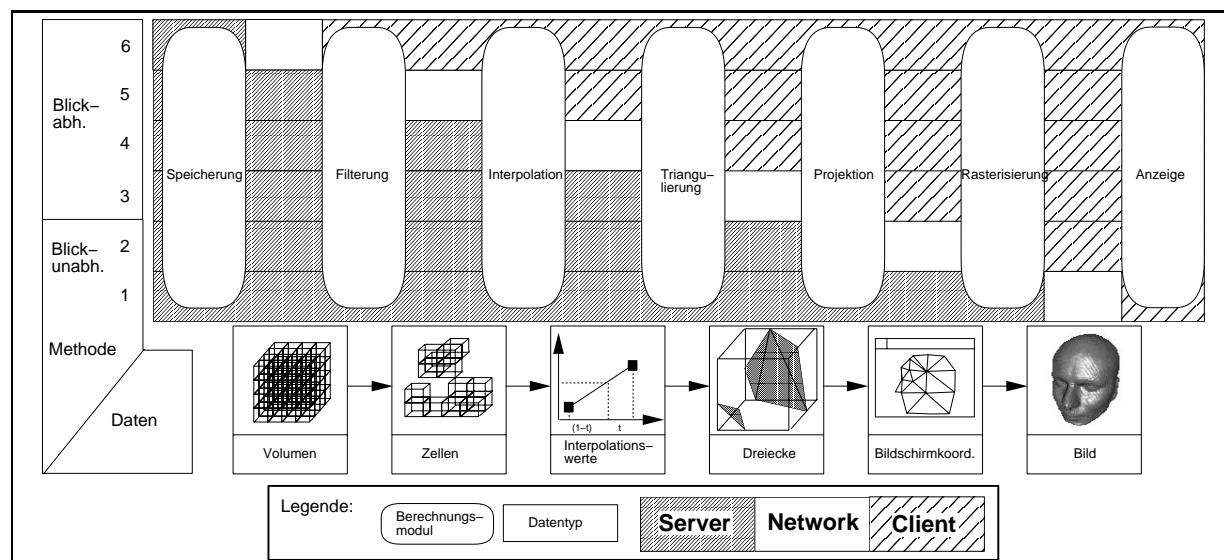


Abbildung 5.12: Verteilungsszenarios für den Marching-Cubes-Algorithmus.

5.3.1 Einleitung

Zur Formalisierung der möglichen Verteilungsvarianten in einem Client-Server-Umfeld wird zunächst eine modifizierte Version der Visualisierungspipeline betrachtet, welche diese um spezielle Module des Marching-Cubes-Algorithmus erweitert. Diese wandeln die Volumendaten in verschiedene intermediäre Datentypen um, bevor die rekonstruierten geometrischen Primitive dargestellt werden (siehe Abbildung 5.12). Das erste dieser Module ist für die Speicherung und Bereitstellung der Volumendaten zuständig, es kann sich somit sowohl um einen Datenbankserver für eine Volumendatenbibliothek als auch um einen einfachen Web-Server handeln. Aus diesen Daten werden zunächst durch ein Filtermodul diejenigen Zellen extrahiert, welche die Isofläche zu einem gegebenen Isowert enthalten. In den so gefilterten Zellen werden mittels eines Interpolationsmoduls die Interpolationsgewichte für die Schnittpunkte der Isofläche mit den Zellkanten bestimmt. Diese werden dann zur Bestimmung der Schnittpunkte herangezogen, aus denen mit einem Triangulierungsmodul Dreiecke gebildet werden. Die Dreiecke werden nach verschiedenen Transformationen auf die Bildebene projiziert, so dass sich für jeden der Eckpunkte Bildschirmkoordinaten ergeben. Schließlich ist ein Rasterisierungsmodul für die Erzeugung des finalen Bildes zuständig, das von einem Darstellungsmodul angezeigt wird.

In einer Client-Server-Umgebung kann nun an einer beliebigen Stelle innerhalb dieser Pipeline die Schnittstelle zwischen Client und Server gelegt werden, womit sich unterschiedliche Aufteilungen der Module der Pipeline ergeben:

1. Der Server ist für alle Module mit Ausnahme des Darstellungsmoduls zuständig und führt damit sowohl die Rekonstruktion der Isoflächen als auch das Rendering der geometrischen Primitive aus. In diesem Szenario ist der Client allein für das Anzeigen der vom Server

empfangenen Bilddaten verantwortlich.

2. In diesem Szenario werden Bildschirmkoordinaten der Ecken der Dreiecke mit den dazugehörigen Farben zum Client transferiert. Die Rasterisierung wird unter Zuhilfenahme lokaler Rasterisierungshardware erledigt.
3. Dieses Szenario kam bereits in Abschnitt 5.2.3 zum Einsatz. Vom Server aus den Volumendaten extrahierte Dreiecke werden zu einem Client (progressiv) übertragen und dort mit lokaler Graphikhardware gerendert.
4. Durch den Transfer der Interpolationswerte entlang der Zellkanten und der Marching-Cubes-Konfiguration ergibt sich dieses Szenario, das die Bestimmung der Vertex-Positionen und die Triangulierung der Dreiecke erst auf Client-Seite erlaubt.
5. Durch die zusätzliche Verlagerung des Interpolationsmoduls auf den Client wird der Transfer von Zellen vom Server zum Client notwendig, die eine Isofläche enthalten. Letzterer ist dann für die restlichen Berechnungen und das Rendering der Daten zuständig.
6. In diesem Fall ist der Server allein für die Speicherung und den Transfer der Volumendaten zu einem Client zuständig. Die Volumendaten werden komplett auf den Client übertragen, der alle restlichen Berechnungen durchführt.

Der Server-seitige Ansatz aus Szenario 1 kann vor allem dann zum Einsatz kommen, wenn ein einfaches Anzeigegerät ohne 3D-Graphikhardware als Client eingesetzt werden soll. In Kapitel 6 wird ein Rahmenwerk zur Fern-Visualisierung vorgestellt[38, 39, 34], mit dessen Hilfe ein Isoflächen-Visualisierungsdienst auf Basis von Szenario 1 realisiert werden kann. Szenario 2 ist in einem Umfeld denkbar, in dem auf der Seite des Clients frühe PC-Graphikhardware benutzt werden soll, die keine Geometrieinheit zur Transformation von Vertices aber sehr schnelle Rasterisierungseinheiten besitzt. Der Client-seitige Ansatz aus dem Szenario 6 wurde bereits in Abschnitt 5.1 behandelt.

Während es sich bei den Szenarios 1 und 2 um blickpunktabhängige Methoden handelt, die für jede neue Betrachterposition oder Ausrichtung des Volumens den Transfer zusätzlicher Daten erfordern, erlauben die Szenarios 3, 4, 5 und 6 eine lokale Interaktion mit den übertragenen Daten. Da in diesem Abschnitt große Volumendatensätze unter Ausnutzung lokaler Graphikhardware über verschiedene Netzwerkanbindungen dargestellt werden sollen, werden im Folgenden die hybriden Szenarios 3, 4 und 5 näher betrachtet.

Der nun folgende Ansatz widmet sich dem Problem, die Isofläche so zu rekonstruieren, dass einerseits die Netzwerklast für die zu übertragende Geometrie durch eine effizientere Darstellung reduziert wird, und andererseits die Darstellung auf der Client-Seite beschleunigt werden kann. Anschließend wird eine Methode besprochen, die es erlaubt, die Anzahl der zu rekonstruierenden geometrischen Primitive schon im Vorfeld einzuschränken und dabei dem Benutzer die Suche nach Details über eine Level-of-Detail-Kontrolle ermöglicht. Schließlich wird eine balancierte Verteilung der Aufgaben der Pipeline auf Client und Server erläutert.

5.3.2 Rekonstruktion von Isoflächenstreifen

Bei der Organisation mehrerer Dreiecke in einem Dreiecksstreifen (*Triangle Strip*) handelt es sich um eine verbreitete Möglichkeit der Kompression von Dreiecksgittern. Dabei definiert nach den ersten zwei Vertices jeder weitere Vertex mit den zwei vorherigen Vertices ein Dreieck. Offensichtlicherweise wird durch die Organisation eines Dreiecksgitters in Dreiecksstreifen die Anzahl der Vertices, die zur Repräsentation des Gitters notwendig sind, verringert. Dies führt einerseits zu einer Reduzierung der Netzwerklast bei der Übertragung des Gitters und andererseits zu einer Optimierung der Rendering-Geschwindigkeit. In Abhängigkeit von der Graphikhardwarearchitektur wird bei einer bestimmten Länge der Streifen eine optimale Verarbeitungsgeschwindigkeit erreicht. Zur Kompression des Dreiecksgitters liefern längere Streifen aber offensichtlich bessere Ergebnisse. Diese können auf der Client-Seite wieder für das Rendering zu Streifen optimaler Länge zerlegt werden.

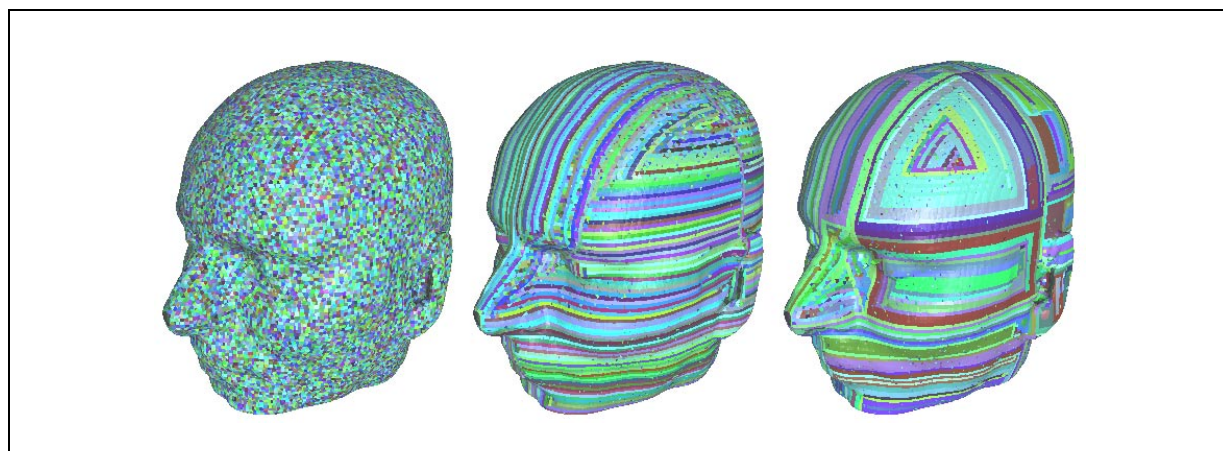


Abbildung 5.13: Alle drei Bilder demonstrieren die Ergebnisse verschiedener Methoden, Dreiecksstreifen direkt aus einem Skalarfeld zu gewinnen. Zur Unterscheidung der Streifen sind die Dreiecke jedes Streifen einheitlich eingefärbt. Links: die Streifen sind nicht über Zellgrenzen hinaus verbunden. Mitte: Streifen sind über Zellgrenzen hinaus verbunden, allerdings nur in eine Vorzugsrichtung. Rechts: Analog zur mittleren Abbildung, nun sind aber verschiedene Richtungen und Richtungswechsel möglich. Die durchschnittliche Streifenlänge beträgt 2, 7 und 14, von links nach rechts.

In einem ersten Schritt wird die Marching-Cubes-Tabelle in der Weise modifiziert, dass für jede Konfiguration die dazugehörigen Dreiecke als Dreiecksstreifen kodiert werden. Dann wird der MC-Algorithmus in der gewohnten Weise ausgeführt, wobei die rekonstruierten Dreiecksstreifen zum Client geschickt werden (siehe Abbildung 5.13, links). Wie zu erwarten, sind die so gewonnenen Dreiecksstreifen sehr kurz, da keine Anstrengungen unternommen wurden, die Streifen aneinanderliegender Zellen zu verbinden.

Um nun die durchschnittliche Streifenlänge zu erhöhen, wird die MC-Tabelle, wie in Abbildung 5.14 angedeutet erweitert. Für jede Konfiguration werden neben der Anzahl der

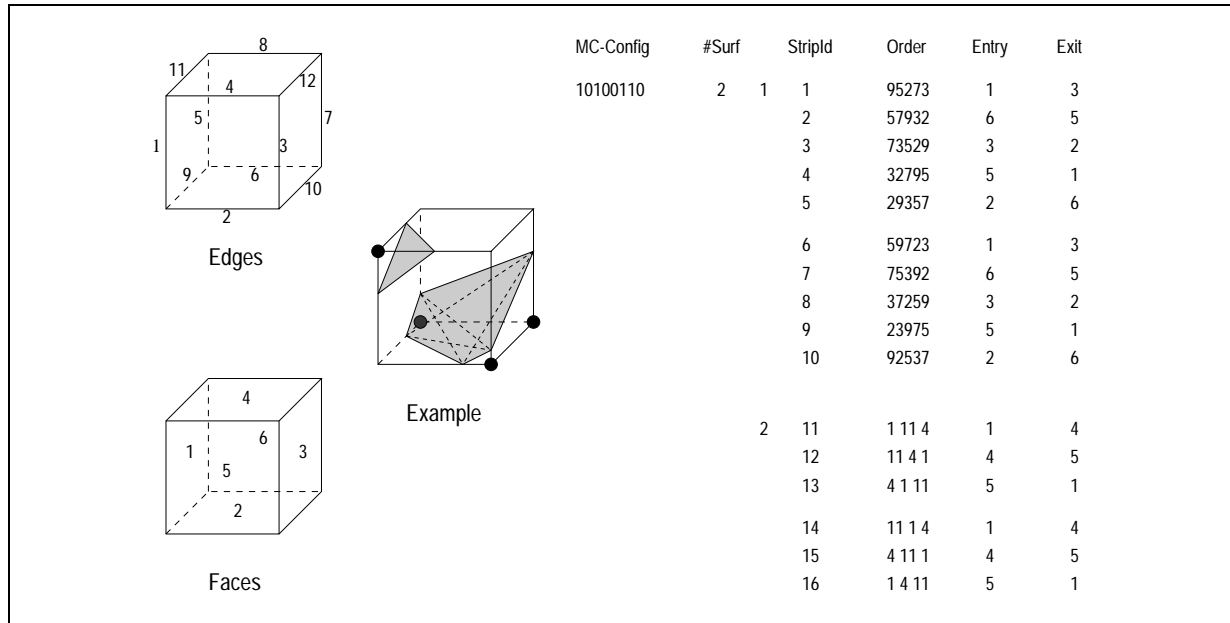


Abbildung 5.14: Die erweiterte MC-Tabelle wird benutzt, um Streifen über Zellgrenzen hinaus zu verbinden. Zu beachten ist, dass Flächenstücke in absteigender Anzahl ihrer Vertizes sortiert sind, da längere Streifen zuerst gewählt werden sollen.

Flächenstücke, die durch die Zelle verlaufen, auch alle möglichen regulären Streifen mit ihren Anfangs- und End-Zellflächen in der Tabelle gespeichert. Zusätzlich wird zu jedem Streifen auch dessen Zwilling gespeichert, der an der selben Zellfläche beginnt, allerdings eine unterschiedliche Reihenfolge der Vertizes aufweist. An dieser Stelle sei darauf hingewiesen, dass unterschiedliche Streifen offensichtlich zu verschiedenen Triangulierungen der Fläche führen. Da aber keine allgemein gültige Regel für die Triangulierung spezifizierbar ist, kann diese beliebig gewählt werden.

Nach der Selektion einer Startzelle, wird der erste Streifen über den zugehörigen Tabelleneintrag gewählt. Es wird versucht, diesen in den benachbarten Zellen fortzusetzen. Die Nachbarzelle ist eindeutig durch die End-Zellfläche und die letzten beiden Vertizes des aktuellen Streifens bestimmt. Alle Vertizes der Fortsetzung des Streifens in der Nachbarzelle, bis auf die beiden ersten, werden daraufhin rekonstruiert.

Dieser Vorgang wird so lange fortgesetzt, bis die folgende Zelle entweder außerhalb des Volumens liegt oder schon bearbeitet wurde. Zur Überprüfung, ob noch eine unbearbeitete Fläche in einer Zelle vorliegt, wird ein weiteres Byte pro Zelle gespeichert. Nach dem erstmaligen Bearbeiten einer Zelle, wird jedes Flächenstück $n \in [1, \#Surf]$ durch Maskieren des entsprechenden Bits n kodiert. Wird ein Flächenstück für einen Dreiecksstreifen benutzt, so wird das entsprechende Bit auf Null gesetzt.

Da Streifen, die auf diese Weise zusammengesetzt wurden, nur mit Nachbarzellen verbunden werden können, die durch die End-Zellfläche des vorherigen Flächenstücks bestimmt werden, ist

es unmöglich, den Streifen in andere, noch nicht bearbeiteten Nachbarzellen, fortzusetzen. Diese Einschränkung führt zu langen und dünnen Streifen (siehe Abbildung 5.14, mitte).

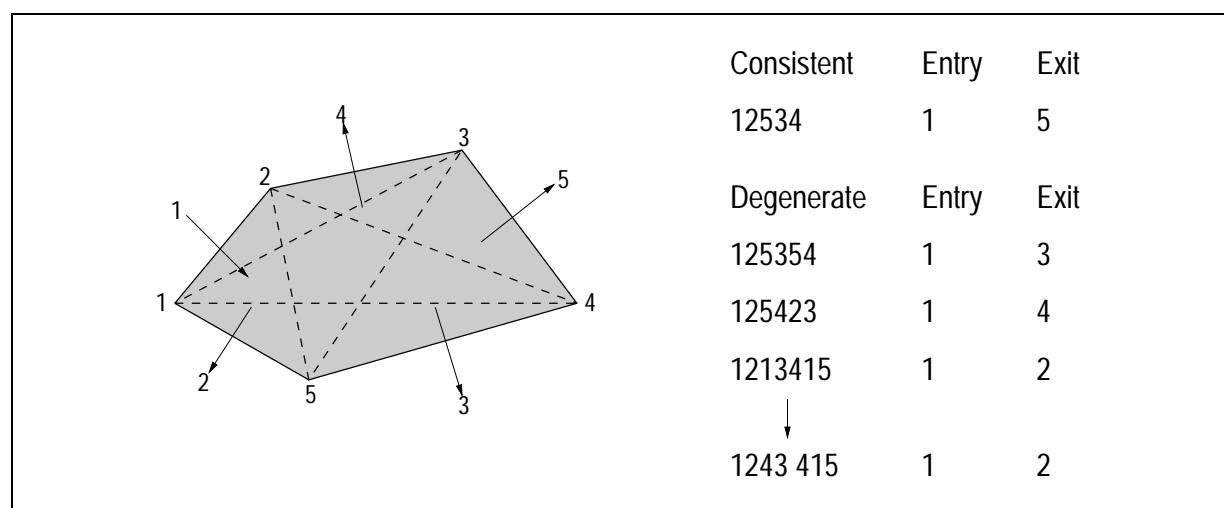


Abbildung 5.15: Die zusätzlich erweiterte MC-Tabelle enthält für jeden Streifen alle alternativen Repräsentationen, um diesen zu beliebigen Nachbarzellen fortzusetzen. Diese müssen für alle möglichen Start-Zellflächen gespeichert werden. Nun werden die Streifen in ansteigender Zahl ihrer Vertizes geordnet, da reguläre Streifen zuerst gewählt werden sollen.

Um nun noch längere Streifen zuzulassen, wird zusätzliche Information in der MC-Tabelle eingebracht. Nun werden auch nicht-reguläre, oder auch degenerierte Streifen zugelassen, die durch wiederholte Vertizes charakterisiert sind (siehe Abbildung 5.15).

Da für verschiedene Kombinationen von Start- und End-Zellflächen unter Umständen keine reguläre Fortsetzung des Streifen möglich ist, werden nun irreguläre Streifen eingesetzt, um die Zellen richtig zu verbinden. Auf den ersten Blick mag es sonderbar erscheinen, redundante Information in den Strom der Vertizes einzubringen. Da aber anstelle des Neubeginns eines Streifens der alte Streifen fortgesetzt werden kann, erweist sich diese Vorgehensweise als vorteilhaft. Wird nur ein redundanter Vertex in den Streifen eingefügt, so ist dies in jedem Falle dem Beginn eines neuen Streifens mit zwei Start-Vertizes vorzuziehen. Bei zwei redundanten Vertizes ist es eigentlich besser einen neuen Streifen zu beginnen. Hinsichtlich der Datentransfermenge sind beide Strategien gleich gut, wenn aber irreguläre Streifen dargestellt werden, so müssen unter Umständen zusätzliche degenerierte Dreiecke gezeichnet werden. In den Ergebnissen dieses Abschnitts wird allerdings demonstriert, dass diese Art der Streifengenerierung die Zahl der Vertizes, die rekonstruiert und gezeichnet werden, signifikant verringert.

Wiederholte Vertizes können sehr effizient kodiert werden, ohne dabei die Datentransfermenge zu erhöhen. Sobald ein Vertex in einer Zelle wiederholt wird, ist die genaue Position des Vertex im aktuellen Streifen bekannt. Die Distanz zu dieser Position wird in den Vorzeichen der (xyz)-Komponenten der Position des aktuellen Vertex gespeichert. Das Bit mit der niedrigsten Wertigkeit wird in der z-Position des Vertex gespeichert, etc.. Da die Distanz immer kleiner als

7 ist, genügen drei Bits zur Kodierung der Distanzinformation. Die *Bounding-Box* des Volumens muss dazu allerdings in den positiven Oktanten verlagert werden, damit das Kodierungsverfahren nicht in Konflikt mit der Position der Vertizes gerät. Sobald der Client eine negative Komponente bei der Position des Vertex empfängt, werden die Vorzeichen aller Komponenten zur Berechnung der Distanz herangezogen, und sowohl der bereits im Streifen vorhandene Vertex, als auch der neue Vertex in den Streifen eingefügt.

5.3.3 Octree-basierte Isoflächen

Obwohl durch den im letzten Abschnitt vorgestellten Ansatz die Anzahl der Vertizes deutlich verringert werden kann, sind interaktive Raten für Datensätze mit hoher Auflösung nicht zu erreichen. Es ist vielmehr notwendig, die hierarchische Struktur von Multi-Skalen Repräsentationen heranzuziehen, um die Zahl der die Fläche repräsentierenden Primitive zu verringern.

In [147] wurde ein adaptiver, Octree-basierter Ansatz zur Rekonstruktion von Isoflächen mit beliebigem Level-of-Detail aus regulären Volumendaten vorgestellt. Der Algorithmus erlaubt eine Realzeit-Interaktion mit komplexen Strukturen durch einstellbare Auflösungsstufen. Dabei werden aus einem hierarchischen Octree *on-the-fly* adaptive Isoflächen rekonstruiert und so einem Nutzer die Möglichkeit gegeben, alle verfügbaren Isoflächen interaktiv zu untersuchen. Der Behebung von Diskontinuitäten an den Stellen aneinandergrenzender Stufen des Octrees wurde dabei spezielle Aufmerksamkeit gewidmet.

Um auch Untersuchungen an Datensätzen mit höchsten Auflösungen zuzulassen, kann ein Benutzer manuell eine interessante Region selektieren, in deren Bereich die Isofläche mit hoher Auflösung rekonstruiert wird. Außerhalb dieser Region und mit steigendem Abstand wird die Fläche mit zunehmend gröberer Auflösungsstufen rekonstruiert. Folglich ist es möglich, durch adaptive Wahl der Größe und Auflösung dieser Region, ausreichend hohe Bildwiederholraten für eine interaktive Navigation zu erreichen.

Um nun diesen Ansatz für ein verteiltes Web-basiertes Szenario zu optimieren, wurden verschiedene Verbesserungen hinzugefügt:

Rekursive Rekonstruktion: Der Rekonstruktionsprozess wird am Fokuspunkt gestartet und dann rekursiv um diesen Punkt herum fortgesetzt. Dadurch kann ein Benutzer die Rekonstruktion anhalten, sobald die gewünschten Details angezeigt wurden. Immer wenn ein bestimmter innerer Knoten des Octrees besucht wird, werden dessen Kinder aufgrund ihres Abstands zum Zentrum des Fokus sortiert und in dieser Reihenfolge abgearbeitet.

Stufen-weises Stripping: Der Benutzer ist in der Lage, eine beliebige Stufe zu wählen, aus der die Isofläche mittels des im letzten Abschnitt besprochenen Verfahrens für Isoflächenstreifen rekonstruiert wird. Obwohl eine Rekonstruktion von Streifen aus einer Mehrstufen-Repräsentation sehr einfach realisiert werden kann, ist eine Verbindung der Streifen verschiedener Stufen im Allgemeinen nicht ohne Weiteres möglich.

Objektraum-Clipping: Auf der Seite des Clients können beliebige Clipping-Ebenen definiert werden, deren Parameter zum Server übermittelt werden. Diese werden daraufhin beim Traversieren des Octrees mit einbezogen, so dass Strukturen innerhalb eines Objekts betrachtet wer-

den können, die normalerweise durch andere Strukturen verdeckt wären. Zusätzlich können eine große Menge von Operationen durch die alleinige Betrachtung eines Halbraums eingespart werden.

Optimierte Triangulierung: Die Behandlung von Übergängen zwischen verschiedenen Stufen des Octrees wurde mit dem Ziel der Minimierung der sich ergebenden geometrischen Primitive optimiert. In [147] wurde vorgeschlagen, Dreiecke der gröberen Stufe in so genannte *Triangle-Fans* zu unterteilen, um so T-Vertizes zu vermeiden (siehe Abbildung 5.16 A). Die neue Methode erlaubt ein kontinuierliches Skalarfeld, auch wenn die Extraktionsstufe gewechselt wird, indem die Skalarwerte an den Flächen eines Stufenübergangs angepasst werden (siehe Abbildung 5.16 B). Diese Kontinuität wird dadurch sichergestellt, dass die Zellen der gröberen Stufe das Skalarfeld der feineren Stufen an den geradzahigen Gitterpositionen benutzen. An den ungeraden Gitterpositionen werden im Gegenzug die Skalarwerte auf der feineren Stufe durch lineare Interpolation aus den gröberen Werten berechnet. Der Skalarwert in der Mitte der Fläche einer Zelle muss derart berechnet werden, dass dieselben Vertizes auf beiden Seiten des Stufenübergangs rekonstruiert werden. Er kann einfach durch Einbeziehung der Geradengleichung der gröberen Stufe mit den bereits interpolierten Skalarwerten berechnet werden. Nun können die Dreiecke auf der gröberen Stufe ohne weitere Modifikation benutzt werden. Offensichtlich führt dies zu T-Vertizes, aber auf der anderen Seite wird dadurch die Anzahl der erzeugten Dreiecke reduziert. Desweiteren kann eine Standard-MC-Tabelle ohne weitere Modifikationen zum Einsatz kommen.

Abbildung 5.17 demonstriert eine adaptive Level-of-Detail Isoflächenrekonstruktion aus einem Octree, deren Kontinuität an den Übergängen der Auflösungsstufen gewahrt wurde.

Bisher wurden verschiedene alternative Ansätze vorgeschlagen, um die Zahl der geometrischen Primitive, die rekonstruiert, übertragen und dargestellt werden müssen, zu reduzieren. Eine weitere Kompression des Datenstroms und eine Einbeziehung der numerischen Fähigkeiten des Clients wurden nicht betrachtet. Allerdings scheint es sinnvoll, die Ressourcen des Clients einzubeziehen, da Clients üblicherweise mit immer stärkeren CPUs ausgestattet sind. Im Folgenden wird demonstriert, wie diese Fähigkeiten genutzt werden können, um die Last auf dem Übertragungskanal zu reduzieren und gleichzeitig die Geschwindigkeit des gesamten Rekonstruktionsprozesses zu verbessern.

5.3.4 Verteilte Isoflächenrekonstruktion

Eine sorgfältige Evaluierung der bisherigen Ansätze erbrachte, dass ein großer Teil der Visualisierungszeit mit der Traversierung der hierarchischen Datenstruktur, dem Testen der Zellen auf mögliche Verfeinerungen und dem Transfer der Primitive über den Kommunikationskanal verbracht wird. Dabei tritt häufig die Situation ein, dass der Client auf neue Daten warten muss. Auf der anderen Seite ist aber ein direkter Zugriff auf die Daten auf der Seite des Clients in vielen Fällen nicht möglich, da Datensätze mit realistischer Größe weder übertragen noch auf einem Client gespeichert werden können. Deshalb wird nun eine verteilte Isoflächenrekonstruktion angestrebt.

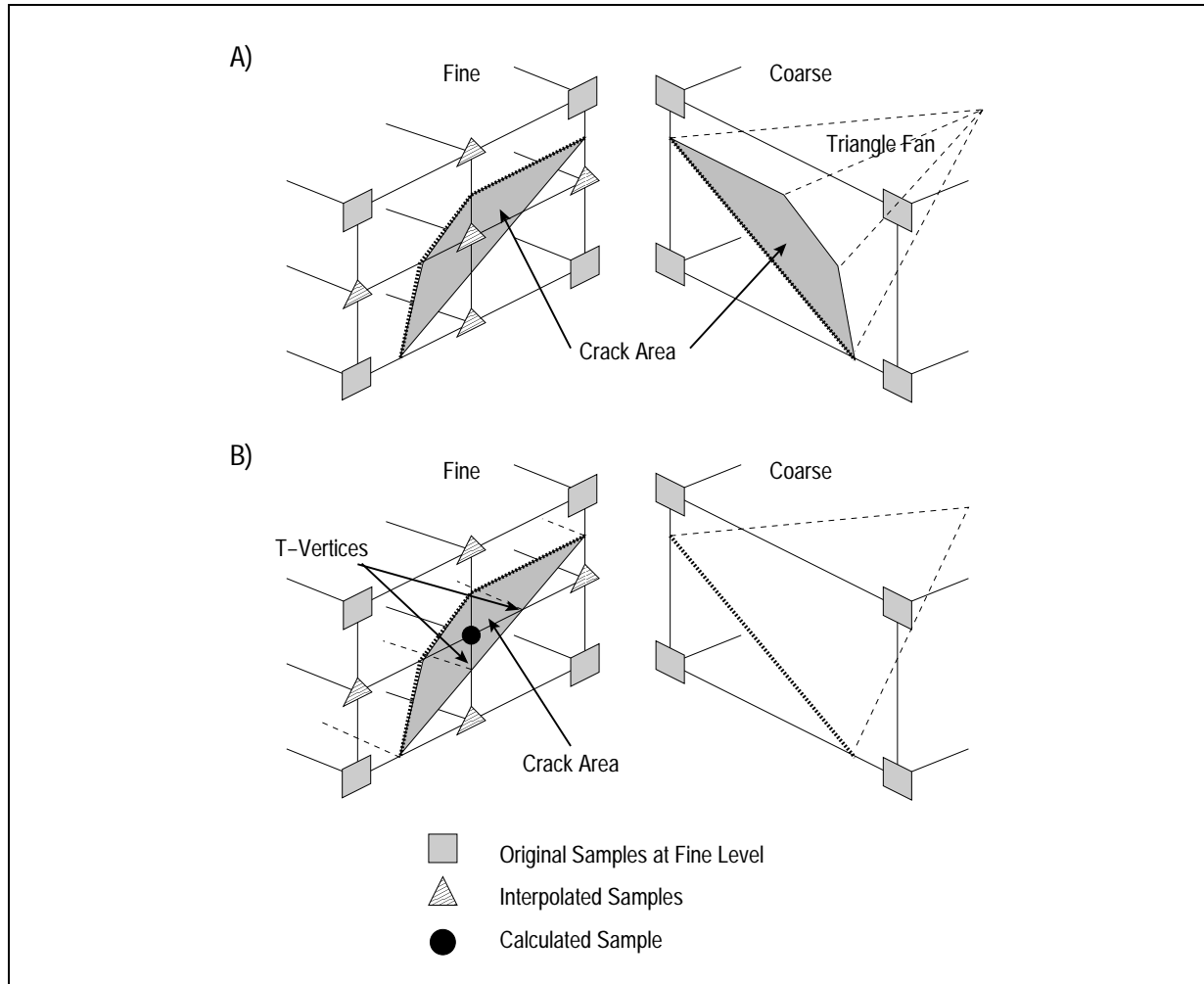


Abbildung 5.16: Rekonstruktion einer kontinuierlichen Isofläche durch Behebung von Diskontinuitäten an den Stellen aneinandergrenzender Stufen des Octrees.

Auch wenn viele Datensätze nicht auf einem Arbeitsplatzrechner verarbeitbar sind, so können doch lokal vorhandene numerische Fähigkeiten genutzt werden, um Wartezeiten zu verringern und die Belastung der involvierten Systeme besser zu balancieren. Hierfür wurde ein verteiltes Berechnungsverfahren entwickelt, das die Berechnung der Vertexpositionen auf der Seite des Clients erlaubt. Es werden nun zwei verschiedene Szenarien vorgestellt, die den Server von einem Teil der Berechnungen entlasten.

Im ersten dieser Szenarien führt der Server in gewohnter Manier einen Standard-(oder auch Octree-basierten) MC-Algorithmus aus. Bevor die Vertices aber aus den bereits interpolierten Gewichten an den Zellkanten ermittelt werden, wird die Rekonstruktion abgebrochen. Die Gewichte werden zusammen mit der MC-Konfiguration, der Octree-Stufe und der Position der Zelle zum Client geschickt (siehe Abbildung 5.18 A). Die Position der Zelle wird dabei als Index mit 4 Byte

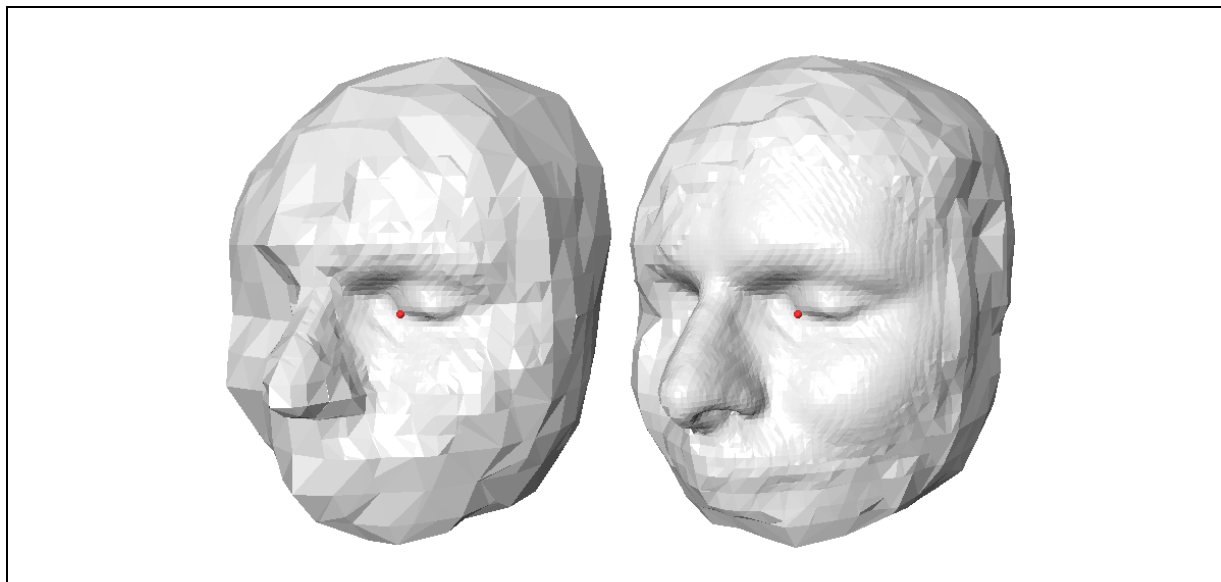


Abbildung 5.17: Adaptive Level-of-Detail Isoflächenrekonstruktion aus einer Mehrstufen-Hierarchie. Der Radius der Fokusregion, in der eine Rekonstruktion der Isofläche aus den Originaldaten stattfindet, wurde für das rechte Bild im Vergleich zum linken Bild vergrößert.

kodiert, wodurch die Auflösung der Daten auf $2048^2 \cdot 1024$ Voxel beschränkt ist. Dies entspricht Szenario 4 aus Abbildung 5.12.

Aus dem in Abbildung 5.18 A gegebenen Beispiel wird ersichtlich, dass im Vergleich zu einem Standard-MC-Algorithmus und der zellweisen Kodierung von Dreiecken in Streifen ein erheblicher Teil der transferierten Daten eingespart werden kann. Auch wenn der Streifen an einen bestehenden Streifen angehängt werden kann, ist diese Vorgehensweise immer noch im Vorteil. Um allerdings einen fairen Vergleich zu erlauben, muss auch bedacht werden, dass die mehrfache Übertragung von Interpolationsgewichten verschiedener Zellen mit ein und derselben Zellkante vermieden werden kann. Außerdem ist es fraglich, ob diese Interpolationsgewichte mit vier Byte kodiert werden müssen. In den von uns getesteten Fällen erwies sich eine Kodierung der Interpolationsgewichte mit einem Byte als ausreichend, ohne dass dabei visuelle Artefakte zu erkennen waren.

Das zweite Szenario lässt den Server als Datenmanager agieren, der die Octree-Repräsentation der Daten im Speicher hält und traversiert, und die zur Rekonstruktion benötigten Zellen zum Client transferiert. Der Server beendet die Rekonstruktion sobald die Daten der aktuellen Zelle und deren MC-Konfiguration ermittelt wurde. Die Konfiguration, die Datenwerte, die notwendig sind um die Isofläche zu rekonstruieren, und die Position der Zelle in der Stufe des Octrees werden zum Client übermittelt (siehe Abbildung 5.18 B).

Offensichtlicherweise wird in dem gewählten Beispiel die Datentransfermenge nicht weiter reduziert. Da anstelle eines Interpolationsgewichtes nun zwei Datenwerte übermittelt werden müssen. Dennoch ist dieser Ansatz in vielen Fällen vorteilhaft, da Volumendaten häufig mit 8

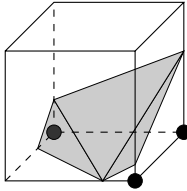
MC-Config	Transmission (Bytes)	
00100110	Local Strips	A) Interpolation Weights
	$5 * 3 * 4 = 60$	$1 + 4 +$ (Level + Index[xyz])
	Optimal Strips	$1 + 5 * 4 = 26$ (Config + 5*Weight)
	$3 * 3 * 4 = 36$	B) Data Samples
		$1 + 4 +$ (Level + Index[xyz])
		$1 + 7 * 4 = 34$ (Config + 7*Sample)

Abbildung 5.18: Übertragung von Vertizes im Vergleich zum Transfer der MC-Konfiguration und Interpolationsgewichten oder Datenpunkten. Es wird angenommen, dass alle Daten bis auf die MC-Konfiguration und die Stufe des Octrees mit vier Byte Genauigkeit kodiert sind. Von der hier als Beispiel gewählten MC-Konfiguration wird angenommen, dass sie in realen Daten am häufigsten auftritt[104].

oder 16 Bit Genauigkeit vorliegen. Eine Quantisierung ist dadurch implizit gegeben und muss nicht wie bei der oben besprochenen Methode unter Einbeziehung von Verlusten erreicht werden.

In beiden Fällen wird die MC-Tabelle auf dem Client gespeichert. Die Position einer Zelle aus dem übertragenen Index wird mittels der Bounding-Box des Volumens ermittelt, deren Position und Größe anfangs, d.h. beim Laden eines Datensatzes, übermittelt wird.

5.3.5 Implementierung

Bei der Implementierung der hier vorgestellten Algorithmen stand optimale Verarbeitungsrates des Servers, Client-seitige Plattformunabhängigkeit, sowie die Nutzung lokal vorhandener Graphikhardware im Vordergrund. Der Server wurde in C++ und OpenInventor implementiert. Der Client basiert auf Java, das wegen seiner breiten Verfügbarkeit auf verschiedenen Plattformen und Web-Browsern gewählt wurde. Um auf der Seite des Clients lokale Graphikhardware nutzen zu können, wurde das Rendering mit VRML und OpenGL-Anbindungen implementiert. Sowohl die *Retained-Mode* Schnittstelle VRML als auch die *Immediate-Mode* Schnittstelle OpenGL stellen dabei die notwendigen Interaktionsmechanismen zur Verfügung, die es einem Benutzer erlauben, mit der Darstellung durch Platzieren des Fokuspunktes und Definition eines Radius um diesen zu interagieren. An der Stelle des Fokuspunktes wird eine Kugel mit einem Koordinatenkreuz dargestellt, um die der Radius des Fokus durch eine transparente Kugel visualisiert wird. Der Fokus kann sowohl frei auf der Isofläche per *Click-and-Drag*, als auch durch die Pfeile des an ihm angebrachten Koordinatenkreuzes entlang der Hauptachsen bewegt werden (siehe Abbildung 9.17). Durch das ständige Senden der Position des Fokus und des Radius über eine Socket-Verbindung zum Server, wird die Fläche dabei permanent aktualisiert.

Der effizienten Dekodierung des Datenstroms und der Aktualisierung der 3D Szene galt ein weiteres Hauptaugenmerk. Um dies zu gewährleisten, wird eine dynamische Speicheranforderung vermieden, einer der wesentlichen Flaschenhälse in Java. Dazu wird zu Beginn ein fester Speicherbereich für die empfangenden Daten belegt, der dann permanent mit empfangenen Daten gefüllt wird. Sobald dieser Speicherbereich vollständig belegt ist, werden die Daten an VRML und OpenGL weitergegeben und der Speicherbereich erneut benutzt. Dieses Kopieren bei der Übergabe der Daten an die Graphikschnittstellen ist leider weder bei Nutzung von VRML, noch bei Nutzung von OpenGL-Anbindungen zu vermeiden.

Für die VRML-Implementierung kommt, wie dies bereits in dieser Arbeit öfter der Fall war, wieder eine VRML-Basiszene mit leeren `IndexedFaceSet`-Knoten zum Einsatz, die mit dem EAI effizient mit Daten gefüllt wird. Um den Fokuspunkt auf der Isofläche bewegen zu können, wird ein `TouchSensor`-Knoten vor den `IndexedFaceSet`-Knoten im Szenengraphen eingefügt. Sobald dieser ein Event auslöst, also der Benutzer auf die Isofläche klickt, wird dieses Event über das EAI an das Java-Applet weitergeleitet. Dieses aktualisiert daraufhin die Position des Fokuspunktes im VRML-Szenengraphen und sendet die Position des Fokuses an den Server. Dieser aktualisiert die Isofläche und transferiert neue Dreiecksdaten zum Client in einem kontinuierlichen Datenstrom. Die Daten werden Block-weise in den Szenengraphen eingefügt und dabei jedes Mal ein erneutes Rendern der kompletten Szene angestoßen. Da die Isofläche, wie oben bereits erwähnt, beginnend mit der Position des Fokuspunktes aktualisiert wird, kann der Benutzer zu jeder Zeit die Übertragung abbrechen.

Bei der *Immediate-Mode*-Implementierung mit OpenGL-Anbindungen können die empfangenen Dreiecke direkt in den Framebuffer gerendert werden. Es ist also kein komplettes Neuzeichnen der Isofläche nötig, solange die Betrachterposition nicht verändert wird. Der Vorteil einer *Immediate-Mode*-Implementierung gegenüber der *Retained-Mode-Implementierung* mit VRML wird in den nun folgenden Ergebnissen deutlich.

5.3.6 Ergebnisse

Für die folgenden Ergebnisse wurde als Server eine SGI Octane MXE mit einem 250 MHz R10000 Prozessor und 1024 MB Hauptspeicher eingesetzt. Als Client kam eine SGI O2 Workstation mit 195 MHz R10000 Prozessor und 128 MB Hauptspeicher zum Einsatz. Beide Rechner waren durch ein 100 MBit Fast-Ethernet Netzwerk verbunden. Auf der Client-Seite wurde der Netscape Communicator 4.07 mit CosmoPlayer 2.1 und JDK 1.1.6 mit Magician OpenGL-Bindings 1.1[25] benutzt.

Zunächst werden die drei verschiedenen Ansätze zur Rekonstruktion von Isoflächenstreifen analysiert und mit einem Standard-MC-Algorithmus mit und ohne komprimierten Datentransfer verglichen. Tabelle 5.2 zeigt detaillierte Ergebnisse der vorgestellten Techniken. Die entsprechenden Visualisierungen sind in der Abbildung 9.15 illustriert.

Die Methode *Org* bezeichnet einen Standard-MC-Algorithmus, bei dem die Dreiecke unabhängig voneinander übertragen werden. *Cmp* und *Smp* stehen für die Zell-weise Rekonstruktion von Flächenstreifen, ohne Streifen aneinandergrenzender Zellen zu verbinden. Die Methode *Cmp* stoppt allerdings die Rekonstruktion genau nach der Berechnung der Interpolationswerte und übermittelt diese Werte zusammen mit der MC-Konfiguration zum Client. Die Methoden

	Org	Cmp	Smp	Red	Opt
Reconstruct	100%	97%	99%	43%	37%
Strip Length	1	-	1.9	7.7	13.2
#Vertizes (Send)	100%	-	69%	43%	38%
#Bytes .	100%	13%	70%	46%	40%
#Vertizes (Rend)	100%	69%	69%	43%	39%

Tabelle 5.2: Modelcharakteristik und Rekonstruktionszeiten für die verschiedenen Isoflächen-Rekonstruktionsszenarios für einen Datensatz mit einer Auflösung von $512^2 \times 256$ Voxeln.

Red und *Opt* zeigen die Ergebnisse der Rekonstruktion von verbundenen Isoflächenstreifen mit und ohne Vorzugsrichtung.

Die benötigte Zeit zur Rekonstruktion der Isofläche auf dem Server variiert nur leicht zwischen den Methoden *Org*, *Cmp* und *Smp*, da die Berechnung der Vertexpositionen aus den bereits bestimmten Interpolationsgewichten und die Organisation in separate Dreiecke nur unwesentlich zur Gesamtzeit beiträgt. Dagegen sind beide Ansätze, die Isoflächenstreifen über Zellgrenzen hinweg zu berechnen, durch die verringerte Anzahl berechneter Vertizes deutlich schneller. Obwohl ein gewisser zusätzlicher Aufwand zum Traversieren der erweiterten MC-Tabelle und zur Überprüfung der Fortsetzbarkeit eines Streifens in angrenzenden Zellen entsteht, dominiert der Nutzen durch die Vermeidung mehrfacher Berechnung desselben Vertizes deutlich die insgesamt Verarbeitungsgeschwindigkeit. Dies wird ebenfalls aus den Zeiten der Ansätze *Red* and *Opt* deutlich, deren Unterschiede im Wesentlichen aus der Länge der Streifen resultieren.

Die Zahl der rekonstruierten Vertizes beeinflusst nachhaltig die Datentransferlast und Rendering-Geschwindigkeit. An dieser Stelle sei darauf hingewiesen, dass die Methode *Opt* im Bezug auf die Zahl der rekonstruierten, übertragenen und dargestellten Vertizes trotz des Einfügens mehrfacher Vertizes in einen Streifen im Vergleich der hier vorgestellten Techniken optimal ist.

Hinsichtlich der Datentransferlast ist offensichtlich der Ansatz *Cmp* gegenüber anderen Methoden im Vorteil, da quantifizierte Interpolationswerte anstelle von Vertizes übertragen werden. Dieser Vorteil wird durch die zusätzlichen Berechnungen, die auf Client-Seite zur Bestimmung der Vertex-Positionen notwendig ist, leicht reduziert. Allerdings ist aus der ersten Zeile der Tabelle 5.2 zu ersehen, dass dies nur einen Bruchteil der Gesamtzeit ausmacht. In diesem Zusammenhang ist auch zu erwähnen, dass bei der Übertragung von Isoflächenstreifen durch das Hinzufügen der Streifenlängen zum Datenstrom ein geringer Zusatzaufwand entsteht.

Zur Verdeutlichung des Vorteils des vorgestellten Multi-Skalen-Ansatzes zeigt Abbildung 9.16 das Ergebnis einer Standard-Isoflächenrekonstruktion mit dem MC-Algorithmus im Vergleich zur Level-of-Detail Rekonstruktion desselben Datensatzes. Die originale Fläche besteht aus ungefähr 6 Millionen Vertizes. In der Web-basierten Testumgebung benötigte die Rekonstruktion der Isofläche auf dem Server ungefähr 41 Sekunden und 72 MB Daten mussten zum Client transferiert werden. In der mittleren Abbildung wurde der Fokuspunkt nahe dem Bauchnabel positioniert. Dadurch wurde die Gesamtzahl der Vertizes auf grob 157 000 reduziert, die in 1.3 Sekunden berechnet werden konnten. Durch den Transfer der quantifizierten Interpolati-

onswerte konnten ungefähr 97% der Daten eingespart werden. Die Verarbeitungsgeschwindigkeit wurde durch die Verwendung zweier Clipping-Ebenen weiter verbessert. Die verbleibende Fläche ist im rechten Bild der Abbildung 9.16 zu erkennen, die in 0.3 Sekunden rekonstruiert wurde. Dies demonstriert die Möglichkeit einer Realzeit-Exploration auch höchst aufgelöster Datensätze.

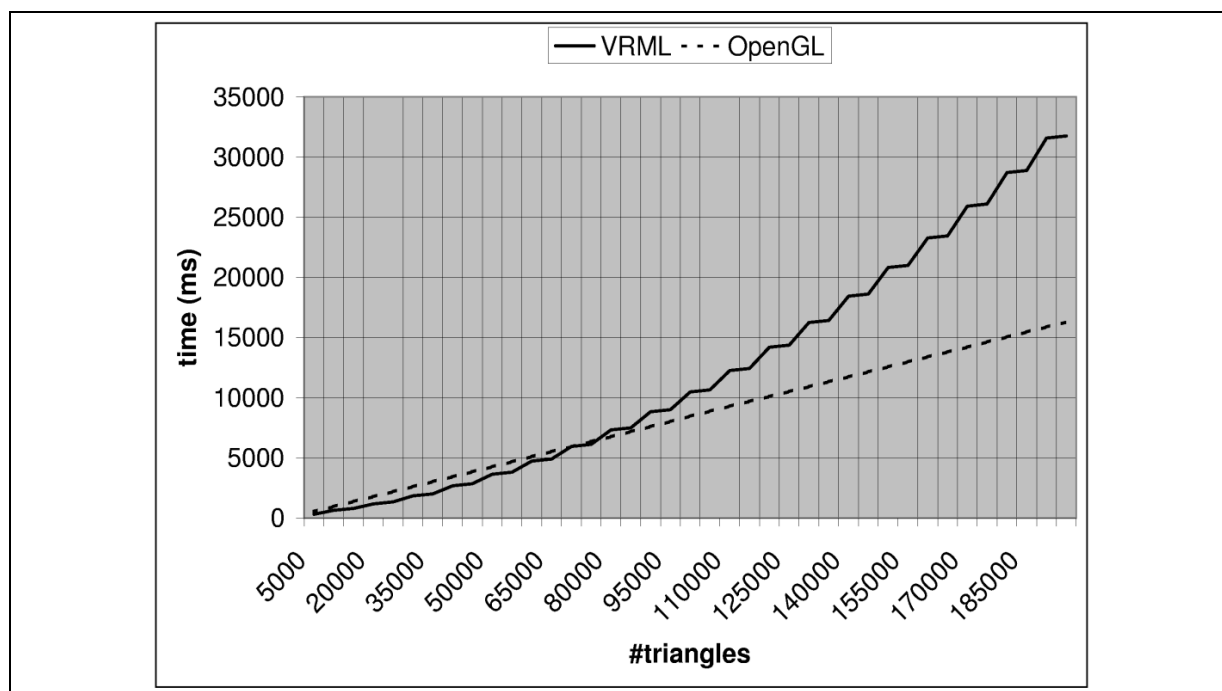


Abbildung 5.19: Graphische Auswertung der Messung von Transfer- and Renderzeiten für die VRML- und OpenGL-Implementierung.

Abbildung 5.19 zeigt eine detaillierte Geschwindigkeitsevaluation der OpenGL- und VRML-basierten Implementierungen des Systems. Nach dem Empfang von jeweils 5000 Dreiecken werden diese an die Render-Module weitergegeben und dargestellt. Für eine niedrige Zahl von Dreiecken ist die VRML-Implementierung leicht im Vorteil, da diese offensichtlich die Daten effizienter vom Java-Applet entgegennehmen kann. Das Kopieren der Daten durch das Java Native Interface (JNI) bei der Nutzung der OpenGL-Anbindung erweist sich hier als der wesentlicher Flaschenhals. Dagegen ist beim Empfang zusätzlicher Dreiecke die VRML-Implementierung im Nachteil, da das Rendern der gesamten Geometrie, dem direkten Rendern der nur neu empfangenen geometrischen Primitive unterlegen ist. Für das Streaming der Geometrie bei ständiger Aktualisierung der Darstellung ist also eine Immediate-Mode Graphikschnittstelle vorteilhaft, da VRML keinerlei Einfluss auf den Rendering-Vorgang erlaubt.

5.4 Ergebnisse

In diesem Kapitel wurden Algorithmen und Methoden zur interaktiven Visualisierung von polygonalen Isoflächen in einem Web-basierten Umfeld besprochen. Einfache Methoden der lokalen Erzeugung und Darstellung von Isoflächen für Datensätze mit limitierter Auflösung sind für bestimmte Anwendungsfälle sehr gut geeignet. Dies wurde am Beispiel des Web-Dienstes *OrbVis* demonstriert, der sich durch seine einfache Bedienbarkeit und gute Verarbeitungsgeschwindigkeit auszeichnet. Die Grenzen dieses Ansatzes wurden aber schon bei der Visualisierung von Ribosomen-Volumendatensätzen deutlich, deren Auflösung durch jüngste Fortschritte in der Messtechnik die Möglichkeiten dieses Ansatzes sprengt. Die progressive Extraktion, Übermittlung und Darstellung von Isoflächen aus einer Fehler-basierten Mehrstufen-Repräsentation der Volumendaten erlaubt es, durch eine explizite Level-of-Detail-Kontrolle, die Komplexität der Fläche zu reduzieren. Auf diese Weise wird sowohl die Menge der zu transferierenden Daten als auch die Rendering-Geschwindigkeit verbessert, so dass auch Datensätze mit grosser Auflösung handhabbar werden. Zunächst kann sich ein Benutzer einen groben Eindruck über die Form der Isofläche verschaffen, bevor zusätzliche Details angefordert werden. Der Datensatz kann dabei vollständig auf dem Server verbleiben. Obwohl dieser Ansatz bereits einen wesentlichen Vorteil gegenüber einfachen Methoden bietet, ist doch die globale Verfeinerung der Fläche ein erheblicher Nachteil in bestimmten Anwendungsfällen. Sucht ein Benutzer nach bestimmten Details der Fläche, so ist der Transfer der Daten in der höchsten Auflösungsstufe notwendig. Der Vorteil der progressiven Extraktion und Darstellung geht dabei natürlich verloren. Eine Lösung dieser Problematik liefern Level-of-Detail-Darstellungen von Isoflächen aus einer Octree-Hierarchie mittels eines Fokuspunktes. Hier ist ein Benutzer in der Lage, einen Fokuspunkt interaktiv über die Isofläche zu bewegen. Um dessen Zentrum wird die Fläche in hoher Detailtreue dargestellt. Zusätzlich wurden in diesem Abschnitt verschiedene Verteilungsstrategien entwickelt, um sowohl die Netzwerklast zu minimieren als auch eine wohlbalancierte Verteilung der Teilaufgaben einer Isoflächenvisualisierung zu erreichen.

Letztlich ist die Frage, welcher der hier vorgestellten Algorithmen für einen speziellen Anwendungsfall geeignet ist, von der Art und Komplexität der jeweils gegebenen Daten abhängig. Eine komplett lokale Extraktion und Darstellung von Isoflächen ist sicherlich nur bei begrenzter Auflösung der Volumendaten zu empfehlen. Bei Volumendaten mit höherer Auflösung müssen zwingend progressive oder adaptive Methoden zum Einsatz kommen, um die Komplexität der Flächen schon im Vorfeld zu reduzieren. Spielt die Suche nach der globalen Form einer Isofläche über die Manipulation des Isowertes eine wesentliche Rolle, so dürfte ein progressiver Ansatz die besten Ergebnisse erzielen. Ist aber die Hauptaufgabe eines Benutzers die Suche nach bestimmten Details in den Daten, so ist ein Fokus-basierter Algorithmus mit effizientem Datentransfer zu empfehlen.

Time is the longest distance between two places.

Tennessee Williams, 1911-1983

Kapitel 6

Fern-Visualisierung

Nachdem in den vorhergehenden zwei Kapiteln Verfahren zur direkten und indirekten Visualisierung von Volumendaten in digitalen Dokumenten besprochen wurden, soll im Folgenden eine Methode zur Fernvisualisierung von Volumendaten entwickelt werden, die unabhängig von der Art des Visualisierungsalgorithmus arbeitet. Es handelt sich dabei um einen Server-seitigen Ansatz zur Darstellung jeglicher Art von Visualisierungen auf einem Client, der auf dem Streaming von Server-seitig generierten Bildern und der Rückübertragung von Benutzerinteraktionen beruht. Ziel dieses Verfahrens ist es, eine interaktive Visualisierung komplexer wissenschaftlicher Daten auf einem beliebigen Rastergraphik-fähigen Client zu ermöglichen. Dieser „dünne“ Client wird dabei ausschließlich zur Anzeige der auf einem Hochleistungsrechner generierten Graphiken sowie zur Entgegennahme von Benutzeraktionen genutzt. Dies hat zur Folge, dass äußerst einfache Client-Plattformen zum Einsatz kommen.

Die Fernvisualisierung eröffnet enorme Möglichkeiten in zweierlei Hinsicht: Einerseits kann mit diesem Ansatz auf entfernte Daten sowie Rechen- und Graphikressourcen mit einem einfachen Gerät zugegriffen werden. Beispielsweise sind im Bereich des *Mobile Computing* so genannte *Location-Based-Services* realisierbar, die dem Nutzer eines mobilen Geräts drei-dimensionale Darstellungen komplexer Szenen in Abhängigkeit von dessen Standort zur Verfügung stellen. Denkbare Szenarien in diesem Zusammenhang sind beispielsweise drei-dimensionale Städtetouren für Touristen-Informationssysteme oder die mobile Verfügbarkeit radiologischer Patientendaten in einem Krankenhaus. Andererseits sind kollaborative Anwendungen mit einem solchen Ansatz realisierbar, beispielsweise eine gemeinsame Visualisierung komplexer Crash-Simulationen durch Mitarbeiter eines Automobilkonzerns, die sich an unterschiedlichen Standorten befinden und dabei verschiedene Anzeigegeräte einsetzen.

Neben den Nutzern eines Fernvisualisierungsdienstes können auch die Anbieter solcher Services enorm profitieren. Durch die Bereitstellung der Ressourcen teurer Hochleistungsrechner für ein breites „Publikum“ können Leerlauf-Zeiten minimiert werden und sich auf diese Weise Anschaffungskosten schneller amortisieren.

6.1 Grundlagen

Die Nutzung entfernter Graphikhardware für Visualisierungszwecke erfordert das Ausführen einer Visualisierungsapplikation auf einem Server-Rechner, der über entsprechende Fähigkeiten verfügt. Im Umfeld der UNIX-Betriebssysteme kann die graphische Ausgabe einer, auf einem entfernten Rechner laufenden Applikation über das X-Windows System auf den lokalen Bildschirm umgeleitet werden. Die Darstellung erfolgt dabei aber leider unter Ausnutzung lokaler Graphikhardware. Bei OpenGL-basierten Applikationen ist für die Weitergabe der OpenGL-Kommandos das GLX-Protokoll zuständig, das eine X11-Protokoll-Erweiterung für die Verbindung von OpenGL und dem X-Windows System darstellt.

Im Virtual Network Computing (VNC) System von Richardson et al.[121] stellen Server-Maschinen Applikationen, Daten und die Benutzeroberflächen zur Verfügung. Darauf kann mit einer großen Zahl von Clients mit unterschiedlicher Architektur zugegriffen werden. Der Zugriff der Clients erfolgt dabei über TCP/IP. Somit hat ein Nutzer Zugriff auf eine Benutzeroberfläche eines entfernten Rechners. Allerdings erlaubt VNC nicht die Nutzung der auf dem Server vorhandenen 3D-Graphikhardware. Diese Einschränkung kann aber durch das *Pre-Loading* einer GLX-Bibliothek, die GLX-Kommandos an die Graphikhardware des Servers weiterleitet und die erzeugten Bilder als X-Bitmaps per X-Protokoll an den Client verschickt umgangen werden[137].

Das Produkt *OpenGL Vizserver* von SGI[103] erlaubt es einer einzelnen Onyx2 Deskside Workstation, Bilder aus dem Framebuffer an verschiedene X-Windows-basierte Arbeitsplatzrechner zu verteilen. Dazu kommuniziert die Applikation mit dem X-Server über eine spezielle Schicht, die alle OpenGL-Kommandos auf die Graphikhardware des Host-Systems umleitet. Sobald ein Bild fertiggestellt ist, was üblicherweise durch einen Aufruf von `glXSwapbuffers()` oder `glFinish()` erkennbar ist, wird dieses aus dem Framebuffer ausgelesen, komprimiert und an den Client verschickt. Dieser dekomprimiert das Bild und zeigt es im entsprechenden Fenster an. Allerdings ist dieses System auf eine Onyx2 Workstation als Server, ein lokales Netzwerk und X-Windows basierte Arbeitsplatzrechner beschränkt.

Die Grundidee des in diesem Kapitel vorgestellten Ansatzes ist es, den Zugriff auf entfernt vorhandene Graphikhardware mit jedem Java-fähigen Client mit 2D-Rastergraphik über eine Vielzahl von Internet-Verbindungsbandbreiten zu ermöglichen. Die Benutzeroberfläche soll dabei lokal erzeugt werden, während die aufwändigen 3D-Darstellungen durch den Transfer von Bildern des Framebufferinhalts eines Hochleistungsgraphikrechners realisiert werden.

Die Basisarchitektur des Systems ist in Abbildung 6.1 abgebildet. Auf der Server-Seite werden die erzeugten Bilder einer OpenInventor- oder Cosmo3D-basierten Visualisierungsapplikation entweder in einen sichtbaren Bildschirmbereich oder einen unsichtbaren Puffer (PBuffer) unter Verwendung der Graphikhardware geschrieben. Fertige Bilder werden daraufhin komprimiert und über einen TCP/IP Socket an die verbundenen Arbeitsplatzrechner weitergeleitet.

Der auf den Arbeitsplatzrechnern vorhandene Java-Client dekomprimiert die empfangenen Bilddaten, wandelt sie in gepufferte Java2D-Bilder um und stellt sie im entsprechenden Fenster dar. Auf dem Arbeitsplatzrechner generierte Maus- und Tastatureingaben werden über einen *Common Object Request Broker Architecture* (CORBA) Objektbus an den Server weitergeleitet, wo die lokal generierten Eingaben verarbeitet werden. Eine weitere CORBA-Schnittstelle nimmt

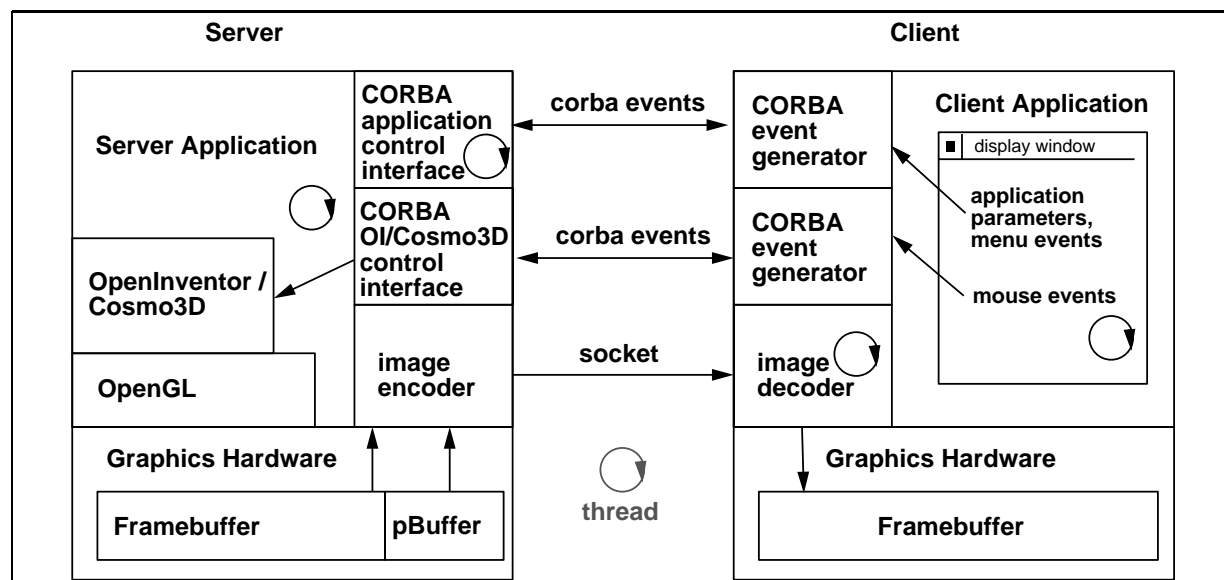


Abbildung 6.1: Das Client-Server-Szenario für das Fernvisualisierungs-Framework. Einer oder mehrere Clients sind mit dem Visualisierungsserver über zwei CORBA-Schnittstellen und einen Socket verbunden.

Applikations-spezifische Parameter entgegen. Bei Bedarf wird ein neuer Rendering-Vorgang gestartet und aktualisierte Bilddaten zum Client geschickt.

Der wesentliche Vorteil dieser Architektur besteht darin, dass das Benutzer-Interface lokal generiert und verarbeitet wird und so auch bei Übertragungskanälen mit niedriger Bandbreite keine Latenzen auftreten. Der Client übernimmt also die Aufgaben, zu der heute jeder beliebige Rechner in der Lage ist, während die aufwändige Verarbeitung der Daten und das Rendering dem Server überlassen wird. Allerdings erfordert dieses Design, dass eine bestehende Visualisierungsapplikation angepasst werden muss, um diese in einen Server für die Fernvisualisierung zu konvertieren. Zusätzlich ist die Erstellung einer Java Client-Applikation notwendig, die das Benutzerinterface der Visualisierungsapplikation nachbildet und die Kommunikation mit dem Server übernimmt. Da dies unter Umständen sehr aufwändig sein kann, wurde ein Rahmenwerk entwickelt, das beide Aufgaben, also sowohl die Konvertierung der Visualisierungsapplikation in einen Visualisierungsdienst als auch die Erstellung einer Client-Applikation unterstützt. Mit Hilfe dieses Rahmenwerks lässt sich der Aufwand zur Erstellung eines Fernvisualisierungsdienstes wesentlich reduzieren. Die Komponenten des Rahmenwerks werden im folgenden Abschnitt besprochen, bevor im anschließenden Abschnitt Visualisierungsdienste erläutert werden, die auf dieser Architektur und dem dazugehörigen Rahmenwerk basieren.

6.2 Das Rahmenwerk

Wie bereits oben erläutert, erfordert die Konvertierung einer bestehenden Visualisierungsapplikation in einen entfernt nutzbaren Visualisierungsservice eine Erweiterung der Anwendung um ent-

sprechende Funktionalität. Es müssen sowohl die generierten Bilder zu den verbundenen Clients verschickt, als auch Benutzerinteraktionen von diesen entgegen genommen werden, um diese zu verarbeiten. Zusätzlich ist die Erstellung einer Java Client-Applikation erforderlich, welche die Benutzerschnittstelle der originalen Visualisierungsapplikation nachbildet und in der Lage ist, lokale generierte Benutzerereignisse an den Server weiterzugeben und die empfangenen Bilder darzustellen. Diese Aufgaben werden durch das Rahmenwerk, das aus einer Reihe von C++ und Java Klassen besteht, wesentlich erleichtert. Durch den modularen Aufbau können sehr einfach weitere Module, wie beispielsweise Bildkomprimierungscodecs, hinzugefügt werden.

6.2.1 Server-Module

Da für das Rahmenwerk von Szenengraph-basierten Visualisierungsapplikationen ausgegangen wird, kann eine solche Applikation sehr einfach durch Einfügen zusätzlicher Knoten in den Szenengraph in einen Server konvertiert werden. Dazu werden durch das Rahmenwerk die entsprechenden Knoten bereitgestellt, die mit nur einigen zusätzlichen Quellcodezeilen in den bestehenden Szenengraph eingebaut werden können. Momentan werden die Szenengraphschnittstellen OpenInventor und Cosmo3D unterstützt.

Die Traversierung des Szenengraphen erfolgt bei OpenInventor in einer festen Reihenfolge von oben nach unten und von links nach rechts. Dabei werden die Zustände der OpenGL-Maschine auch horizontal im Szenengraphen vererbt. Wegen dieser Traversierungsregel kann für das Rendern in einen unsichtbaren Framebuffer-Bereich ein neuer Szenengraphknoten links in den Szenengraph eingefügt werden. Der so genannte `PBufferNode` wechselt vom Standard-Rendering-Kontext in den `PBuffer` Rendering Kontext. Der `PBuffer` ist ein spezieller, unsichtbarer und geschützter Framebufferbereich, der für Hardware-unterstütztes Rendering eingesetzt werden kann. Ein weiterer Knoten, der `SocketNode`, wird ganz rechts in den Szenengraphen eingefügt. Dieser ist für die Entgegennahme von Socket-Verbindungen der Clients zuständig und liest den Inhalt des Framebuffers in den Hauptspeicher aus. Zusätzlich hat dieser Knoten die Aufgabe, den Framebufferinhalt zu komprimieren und an die verbundenen Clients zu verschicken.

Zum Rendern des Szenengraphen wird bei OpenInventor eine *RenderAction* auf den Wurzelknoten des Szenengraphen angewendet, die daraufhin den Szenengraphen traversiert. Durch die eingefügten Knoten werden dabei automatisch die notwendigen Schritte durchgeführt, um den `PBuffer` zu benutzen und die neu generierten Bilder auszulesen, zu komprimieren und zu den Clients zu verschicken. Konkret werden dabei die folgenden Schritte ausgeführt:

1. Die *RenderAction* traversiert den Szenengraphen von oben nach unten und von links nach rechts.
2. Sobald der `PBufferNode` erreicht wird, schaltet dessen Render-Methode in den Rendering-Kontext des `PBuffers` um. Wurde kein `PBufferNode` in den Szenengraphen eingefügt, so verbleibt der sichtbare Rendering-Kontext.
3. Die *RenderAction* fährt mit der Traversierung des Szenengraphen fort. Die Render-Methoden der entsprechenden Szenengraph-Knoten rendern dabei Geometrie in den Rendering-Kontext.

4. Sobald der `SocketNode` am Ende des Szenengraphen besucht wird, liest dieser den Framebufferinhalt des augenblicklichen Rendering-Kontexts in den Hauptspeicher. Die Bild-daten werden mit einem der verfügbaren Algorithmen komprimiert und an die verbundenen Clients über die Socket-Verbindungen verschickt.
5. Die Render-Aktion traversiert den restlichen Szenengraphen.

OpenInventor stellt komfortable Mechanismen zur Verfügung, um die 2D-Mausereignisse, die von einem Client empfangen wurden, in 3D-Ereignisse umzuwandeln. Die dazu benötigten Schritte lassen sich folgendermaßen zusammenfassen:

1. Die Client-Applikation registriert sich für bestimmte Benutzerschnittstellenereignisse bei ihrem Fenstersystem.
2. Die Client-Applikation empfängt ein Ereignis von seinem Fenstersystem.
3. Der Client ruft die entsprechende Server-Methode zur Verarbeitung des Ereignisses über die CORBA-Schnittstelle auf und übergibt die entsprechenden Parameter, wie beispielsweise Mausposition oder gedrückte Maustaste.
4. Die Server-Methode übersetzt das Ereignis in ein von `SoEvent` abgeleitetes Ereignis, beispielsweise ein `SoLocation2Event` für Maus-Ereignisse.
5. Dieses Ereignis wird an eine Instanz der Klasse `SoHandleEventAction` übergeben, die für die Behandlung des Ereignisses zuständig ist.
6. Die Aktion wird auf den Wurzelknoten des Szenengraphen angewendet und traversiert diesen. Jeder der Knoten im Szenengraphen implementiert sein eigenes Verhalten für bestimmte Ereignisse. Wenn ein Knoten des Szenengraphen an dem Ereignis interessiert ist (typischerweise ein Manipulator-Knoten), behandelt dieser das Ereignis.
7. Falls notwendig wird eine neue *RenderAction* gestartet.

Die Hauptschleifen von CORBA und OpenInventor laufen in zwei getrennten Threads ab. Da OpenInventor nicht Thread-sicher ist, können die vom Client empfangenen Ereignisse nicht unmittelbar auf den Szenengraphen angewendet werden. Stattdessen werden die Ereignisse in der `Ticker`-Klasse gepuffert, die als OpenInventor Engine (abgeleitet von `SoEngine`) in regelmäßigen Abständen von der OpenInventor Hauptschleife aufgerufen wird. Dabei werden alle gepufferten Ereignisse auf den Szenengraphen angewendet.

Die Szenengraphstruktur von `Cosmo3D` unterscheidet sich wesentlich von der in OpenInventor. Im Gegensatz zu OpenInventor wird keine Informationen horizontal im Szenengraphen vererbt, sondern ausschließlich vertikal in den einzelnen Teilbäumen. Für die Optimierung komplexer `Cosmo3D`-Szenengraphen existiert das OpenGL Optimizer Toolkit. OpenGL Optimizer bietet verschiedene Aktionen zum Traversieren des Szenengraphen: Auf der einen Seite existiert analog zu OpenInventor eine Tiefensuche. Auf der anderen Seite kann auch Breitensuche für die parallele Traversierung mit mehreren Prozessoren angewendet werden. Aus diesem Grunde wird

ein neuer Cosmo3D Szenengraphknoten abgeleitet von der Cosmo3D-Klasse `csGroup` eingesetzt, der die Funktionalität des entsprechenden `PBufferNode` und `SocketNode` implementiert. Dessen Methode `drawVisit()` enthält Prä- und Post-Traversierungssektionen. Erstere wechselt den Rendering-Kontext in den `PBuffer`, während die letztere das Auslesen, Komprimieren und Übertragen der Bilddaten übernimmt.

Konträr zur Implementierung in `OpenInventor` wird dieser Knoten als neuer Wurzelknoten in den Szenengraphen eingehängt. Er enthält den originalen Szenengraphen als Untergraphen.

Alternativ kann auch der Szenengraph unverändert bleiben und dafür ein Funktionsaufruf an den Beginn der Methode `opXmViewer::swapBuffers()` eingefügt werden. Diese Funktion bestimmt dann den augenblicklichen Rendering-Kontext, ruft `glReadPixels()` auf und initiiert die Komprimierung und Übertragung der Bilddaten.

Eingehende Maus- oder Tastaturereignisse werden nach einem Vorverarbeitungsschritt durch die entsprechenden Methoden der Klasse `opXmViewer` wie lokale Ereignisse verarbeitet. Mausereignisse werden beispielsweise an eine Methode weitergeleitet, welche die eingehenden Daten konvertiert und die Methode `processPendingEvents()` von `opXmViewer` emuliert.

6.2.2 Client-Module

Auf der Client-Seite werden Module für die Java2-Plattform angeboten, welche die Entwicklung einer Client-Applikation wesentlich vereinfachen. Die wesentlichen Klassen sind `RenderArea`, `FullViewer`, `Decoder` und `SimpleViewer`.

Die Basis-Klasse für das Darstellen der vom Server empfangenen Bilder, `vis.inventor.RenderArea`, ist von der Java-Klasse `java.awt.Panel` abgeleitet und kann somit in jeden Java-Container eingebaut werden. Sie benutzt die Java2D-Klasse `java.awt.image.BufferedImage` zum effizienten Darstellen der Bilddaten. Die von der Klasse `vis.inventor.RenderArea` abgeleitete Klasse `vis.inventor.FullViewer` stellt das gleiche *Look-and-Feel* des `SoXtFullViewer` von `OpenInventor` zur Verfügung, die um den eigentlichen Render-Bereich eine Reihe von Drehreglern und Buttons bereitstellt. Diese Klasse unterstützt auch ein Kontext-Menü, welches das Ändern des `OpenInventor` Render-Zustands ermöglicht.

Die Klasse `vis.imagedecoder.Decoder` ist die abstrakte Basisklasse aller Dekodierer des Server-Bildstroms. Neue Dekodierer können durch das Hinzufügen einer abgeleiteten Klasse einfach in das Rahmenwerk integriert werden. Momentan werden die Dekodierer `vis.imagedecoder.ZLIBDecoder`, `vis.imagedecoder.LZODecoder`, `vis.imagedecoder.RLEDecoder` und `vis.imagedecoder.RAWDecoder` zur Verfügung gestellt.

Schließlich stellt die Klasse `vis.viewers.SimpleViewer` ein einfaches Betrachter-Applet zur Beobachtung einer entfernten Visualisierungssitzung dar. Bei Verwendung dieses Applets ist keinerlei Interaktion mit der entfernten Darstellung möglich. Um eine Visualisierungsapplikation für dieses Applet zugänglich zu machen, muss lediglich deren Szenengraph durch Einbringen der neuen Knoten erweitert werden. Die CORBA-Schnittstelle kann in diesem Falle weggelassen werden.

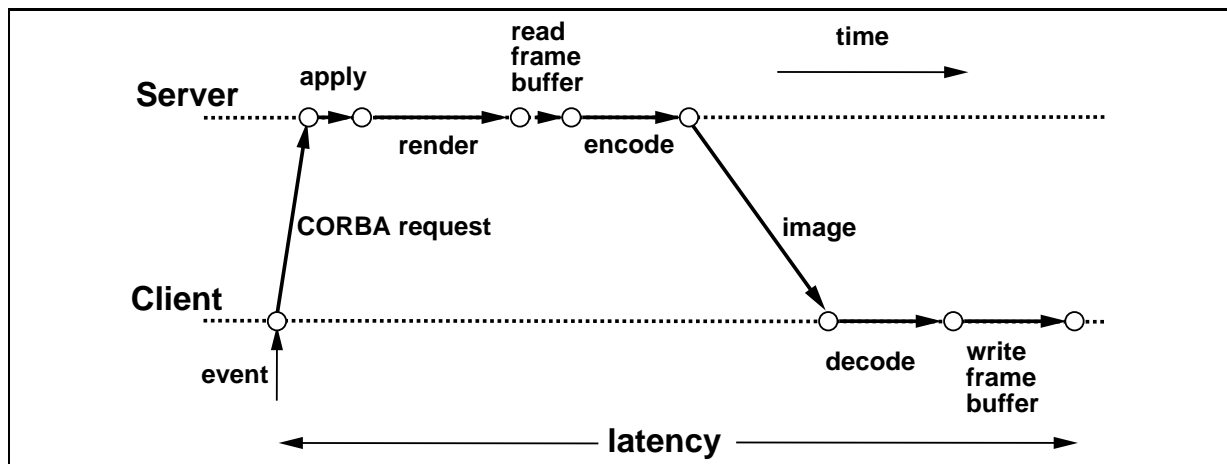


Abbildung 6.2: Die Latenzzeit zwischen der Benutzerinteraktion und der Aktualisierung der Darstellung auf der Seite des Clients setzt sich aus den Einzelzeiten der Übertragung des Steuerkommandos über den Objektbus, dem Übersetzen und Anwenden der Kommandos, dem Rendering, dem Auslesen des Framebuffers, der Komprimierung der Bilddaten, der Übertragung der Bilddaten zum Client, der Dekomprimierung und dem Anzeigen der Daten zusammen.

6.2.3 Datentransfer

Die Übertragung der Bilder und Steuerdaten sind in der hier verwendeten Architektur streng getrennt. Dies ermöglicht eine Verschlüsselung der eventuell sensitiven Bilddaten, während die Steuerdaten zur Gewährleistung einer minimalen Latenz unverschlüsselt bleiben können. Zudem kann für eine einfache Überwachung einer entfernten Visualisierungssitzung auf die Verwendung von CORBA verzichtet werden. Die Bilddaten werden über eine TCP-Socket-Verbindung als kontinuierlicher Datenstrom an den Client weitergeleitet, während die Steuerdaten als CORBA-Methodenaufrufe übertragen werden. Bei mehreren verbundenen Clients kann für die Übertragung der Bilddaten optional Multicasting eingesetzt werden, um die Duplizierung von Paketen auf der Server-Seite zu vermeiden. Sobald der Server neue Rendering- oder Mapping-Parameter erhalten hat, wird die Visualisierungspipeline angestoßen und ein neuer Rendering-Vorgang gestartet. Das auf diese Weise erzeugte Bild wird aus dem Framebuffer gelesen, kodiert, transferiert, dekodiert und in den Framebuffer des Clients geschrieben. Dabei ergibt sich zwangsweise eine gewisse Latenz zwischen der Benutzeraktion und der Aktualisierung der Darstellung (siehe Abbildung 6.2).

Zum Transfer der Steuerdaten vom Client zum Server werden zwei unterschiedliche CORBA-Schnittstellen benutzt. Diese Aufteilung ergibt sich durch die Unterscheidung in Applikations-spezifische und allgemeine Steuerdaten:

- Steuerdaten, die an die `SoXtRenderArea`- oder `SoXtFullViewer`-Klassen des Server weitergeleitet werden sind im Allgemeinen für jede Applikation gleich. Aus diesem Grunde steht die CORBA-Schnittstelle `RenderArea` zur Verfügung, die einfache Maus- und Tastaturereignisse entgegennehmen kann. Diese wird durch die Schnittstelle `FullViewer` um die Behandlung von Ereignissen der `SoXtFullViewer`-Klasse erweitert.

- Applikations-spezifische Steuerdaten, beispielsweise Mapping Parameter einer Visualisierungsapplikation, werden mittels der Schnittstelle `Application` weitergeleitet. Diese Schnittstelle muss für jede Applikation um deren individuelle Funktionalität erweitert werden. Soll zum Beispiel mittels eines Menüpunkts eine Clipping-Ebene eingefügt werden, so könnte eine Methode `addClippingPlane` in diese Schnittstelle eingebracht werden. Die zugehörige Methode muss auf der Seite des Servers mit der entsprechenden Funktionalität gefüllt werden.

Bei der Übertragung der Bilddaten vom Server zu den Clients ist eine Komprimierung erforderlich, um die Datenmenge zu reduzieren und damit eine Vielzahl unterschiedlicher Übertragungskanäle zu unterstützen. Zur Komprimierung werden momentan folgende Kodierer/Dekodierer unterstützt:

- **RAW:** Dieser Codec belässt die Bilddaten unverändert, es wird keine Kompression vorgenommen. Bei Übertragungskanälen mit hoher Bandbreite kann eine unkomprimierte Übertragung schneller sein als eine Komprimierung, Übertragung und Dekomprimierung. In diesem Falle sollte dieses Verfahren eingesetzt werden.
- **ZLIB:** Führt eine verlustlose Kompression mittels der ZLIB Bibliothek durch[122]. Dieser Algorithmus findet duplizierte Zeichenfolgen in den Eingabedaten. Bei mehrmaligem Auftauchen einer Zeichenfolge wird diese in der Form eines Zahlenpaars (Distanz, Länge) gespeichert. Die ZLIB Kompression ist eine Standardfunktionalität von Java seit der Version 1.1 und wird in schnellem nativem Code ausgeführt.
- **LZO:** Führt eine verlustlose Kompression mittels der LZO Bibliothek durch[106]. LZO ist ein Blockkompressionsalgorithmus, der mehr auf eine hohe Komprimierungs- und Dekomprimierungsgeschwindigkeit als eine hohe Kompressionsrate setzt.
- **RLE:** Führt eine verlustlose Kompression mittels einer Lauflängenkodierung durch.

Eine verlustbehaftete Kompression, beispielsweise mit einer JPEG-Kompression, wurde nicht in Betracht gezogen, da eine Reduktion der Bildqualität für das angestrebte Ziel, die Visualisierung medizinischer Volumendaten, nicht akzeptabel ist. Da auch nach Anwendung einer dieser Kompressionsalgorithmen die Datenrate des Bildstroms die Kapazität vieler Übertragungskanäle sprengt, kann zusätzlich vor der Kompression eine Datenreduktion durch Verringerung der Bildauflösung vorgenommen werden. Dazu werden die Bilder vom Server während der Interaktion eines Benutzers mit der Hälfte oder einem Viertel der Originalauflösung berechnet und auf der Seite des Clients wieder auf die Originalgröße skaliert. Dies erhöht die Bildwiederholrate sowohl durch verringerte Rasterisierungsanforderungen als auch durch verringerte Datenmengen. Beendet ein Benutzer die Interaktion, so wird ein Bild mit voller Auflösung übermittelt. Diese Kombination von Datenkompression und Reduktion der Bildauflösung ermöglicht ausreichende Bildwiederholraten, im Extremfall selbst mit ISDN-Verbindungen (siehe Abschnitt 6.5).

6.3 Anwendungen

Die Anwendungsgebiete des in diesem Kapitels vorgestellten Ansatzes zur Fernvisualisierung sind vielfältig. Einerseits eröffnet die Nutzung entfernter 3D-Graphikhardware neue Möglichkeiten im Bereich des *Mobile Computing*. Einfache mobile, Rastergraphik-fähige Geräte sind zur Visualisierung komplexer wissenschaftlicher Daten nutzbar. Zudem ist eine effektive Auslastung vorhandener Rechen- und Graphikressourcen möglich, so dass sich die Anschaffungskosten teurer Hardware schneller amortisieren. Auf der anderen Seite enthält der Ansatz implizit Kollaborationsmöglichkeiten, da alle mit dem Fernvisualisierungsserver verbundenen Clients dieselbe Ansicht teilen und gemeinsam auf die Darstellung Einfluss nehmen können.

Somit eignet sich dieser Ansatz auch für Bereiche, wo Spezialisten an unterschiedlichen Standorten kollaborieren, Experten zur Diskussion herangezogen werden oder Schulungen bzw. Trainings über größere räumliche Distanzen durchgeführt werden. Die Nutzung entfernter 3D-Graphikhardware für kollaboratives Arbeiten soll an zwei Anwendungsgebieten, der medizinischen Bildverarbeitung und den Ingenieurwissenschaften, verdeutlicht werden.

In der Medizin spielen moderne dreidimensionale Bildgebungsverfahren wie Computertomographie (CT), Magnetresonanztomographie (MRT) oder nuklearmedizinische Verfahren (SPECT, PET) in der Diagnostik eine immer größere Rolle. Die erzielte Genauigkeit dieser Verfahren ist in den letzten Jahren so enorm gestiegen, dass heute bereits Volumendaten mit mehr als 100 Millionen Voxeln keine Seltenheit mehr darstellen. Eine interaktive, dreidimensionale Visualisierung dieser Daten ist mithin ohne die Unterstützung moderner Graphikhochleistungsrechner nicht möglich. Wegen den hohen Anschaffungskosten solcher Spezialhardware, ist eine gemeinsame Nutzung dieser Geräte im klinischen Umfeld unverzichtbar. Aus diesem Grund wurde eine bestehende Applikation zur texturbasierten Visualisierung radiologischer Volumendaten mit dem in diesem Kapitel vorgestellten Ansatz in einen Fernvisualisierungsservice konvertiert (siehe Abbildung 9.18, links). Diese Applikation, genannt TIVOR (**T**exture-**B**ased **I**nteractive **V**olume **R**enderer), basiert auf OpenInventor, in dessen Klassenhierarchie ein neuer Szenengraphknoten zur Darstellung von Volumendaten eingebracht wurde. Die Einbettung in OpenInventor erlaubt der Applikation die komfortable Nutzung von 3D-Widgets und anderer OpenInventor Funktionalität[57, 133].

Allein durch das Einbringen des `SocketNode` in den Szenengraphen ist es bereits möglich, eine entfernte Visualisierungssitzung mit TIVOR zu beobachten. Dazu kann die `SimpleViewer`-Klasse als Java-Applikation oder Java-Applet in einem Web-Browser verwendet werden. Zur Fernkontrolle der Applikation sind einige weitere Modifikationen nötig. Dazu wurde eine Client-Applikation erstellt, welche die Klasse `vis.inventor.FullViewer` als Anzeigebereich für die auf dem Server generierten Bilder einsetzt. Es wurden einige zusätzliche Knöpfe und eine Menüleiste hinzugefügt, so dass diese Client-Applikation das Aussehen der Originalapplikation vollständig nachbildet (siehe Abbildung 9.18, rechts). Die CORBA-Schnittstelle wurde um diese zusätzliche Funktionalität von TIVOR erweitert und die entsprechenden *Stub*- und *Skeleton*-Methoden implementiert. Durch konsistente Menu-IDs auf Client- und Server-Seite können Menü-Ereignisse des Clients durch eine einfache CORBA-Schnittstelle auf der Server-Seite ausgewertet werden.

Die Server-Applikation kann entweder sichtbar als Desktop-Applikation oder unsichtbar

im Hintergrund betrieben werden. Dadurch ergeben sich zwei Szenarien: Wird die Server-Applikation sichtbar auf dem Desktop des Server-Rechners ausgeführt, so kann ein Hauptbenutzer dort weitere Nutzer zu einer gemeinsamen Visualisierungssitzung einladen. Dazu macht der Hauptbenutzer eine bestimmte URL bekannt, zu der sich die anderen Nutzer mit einem Java-fähigen Web-Browser verbinden. Von dort wird automatisch die Client-Applikation bezogen, die sich mit dem Fernvisualisierungsserver verbindet. Die Kontrolle der Visualisierungapplikation kann explizit vom Hauptbenutzer an einzelne Teilnehmer weitergegeben werden, so dass allein dieser Benutzer mit der Darstellung interagieren kann. Dagegen nehmen alle Teilnehmer gleichberechtigt an einer Visualisierungssitzung teil, wenn der Fernvisualisierungsserver unsichtbar im Hintergrund betrieben wird. Eine Steuerung der Visualisierungssitzung am Server durch einen Hauptbenutzer ist in diesem Szenario nicht vorgesehen. Vielmehr sind die Benutzer hier für die Koordination der Kontrolle selbst zuständig. Alternativ kann hier auch ein *Token*-Mechanismus zum Einsatz kommen. Alle Teilnehmer verbinden sich zum Visualisierungsserver wieder mit einer vorher bekannt gegeben eindeutigen Adresse.

Da bei dieser Art der Visualisierung keine sensitiven Patientendaten transferiert werden, ist sie vor allem im medizinischen Bereich äußerst interessant. Der Transfer des Bildstroms der Visualisierungapplikation ist sicherlich weniger kritisch als die Übertragung von Originaldaten. Eine weitere Absicherung ist mittels Tunnelung der Datenstroms über einen sicheren Kanal denkbar, also beispielsweise mittels eines SSH-Socket-Kanals[155].

Ein weiteres Anwendungsbeispiel, in dem sich das hier vorgestellte Rahmenwerk als sehr nützlich herausgestellt hat, ist die Visualisierung im Fahrzeugentwicklungsprozess. Speziell bei zeitabhängigen numerischen Berechnungen von Crash-Simulationen ist in der letzten Jahren die dabei erzeugte Datenmenge in den Gigabyte-Bereich gewachsen. Eine fließende Echtzeit-Animation dieser Daten ist zur Zeit nur auf sehr leistungsfähigen Workstations mit spezieller Graphik- und IO-Architektur möglich.

Die auf Cosmo3D/OpenGL Optimizer-basierende Applikation *crashViewer*[134] (siehe Abbildung 9.19, links) wurde speziell für die Visualisierung dieser Daten auf einer Hochleistungsgraphikworkstation erstellt. Damit mehrere Ingenieure einer Visualisierungssitzung mit einfachen Arbeitsplatzrechnern von beliebiger Stelle aus teilnehmen können, wurde *crashViewer* mit dem hier vorgestellten Rahmenwerk erweitert. Zur Diskussion einer aktuellen Modellvariante startet ein Ingenieur die *crashViewer*-Applikation, die in der Lage ist, die erzeugten Bilder als kodierte Datenstrom zur Verfügung zu stellen. Er übermittelt den anderen Teilnehmern eine URL, die daraufhin einen Web-Browser starten und die angegebene HTML-Seite mit dem eingebetteten Java-Applet laden (siehe Abbildung 9.19, rechts). Damit können sie an der Visualisierungssitzung teilhaben und Ergebnisse diskutieren. Vorteil ist auch hier, dass die eigentlichen Daten wieder im Unternehmen verbleiben, was vor allem dann wichtig ist, wenn Ingenieure anderer Partnerunternehmen oder Zulieferer konsultiert werden sollen.

6.4 Hybride texturbasierte Visualisierung

Ein wesentlicher Nachteil des Fernvisualisierungsansatzes ist sicherlich, dass durch den Transfer der Steuerkommandos und den Rücktransfer der Bilddaten unvermeidliche Latenzen auftreten.

Dies führt vor allem bei Verbindungen über größere Entfernungen hinweg zu äußerst störenden Effekten. Wie aber bereits in Kapitel 4 betont wurde, sind heutige moderne Arbeitsplatzrechner durchaus in der Lage, Volumendaten mit eingeschränkter Auflösung flüssig darzustellen.

Daraus ergibt sich die Idee einer Kombination entfernter und lokaler Graphikressourcen in einem hybriden Rendering-System. Dazu wird eine mit einem geeigneten *Resampling*-Filter reduzierte Version des Datensatzes zu Beginn der Visualisierungssitzung auf einen Client übertragen, während das Original des Datensatzes auf dem Server verbleibt. Die Auflösung des reduzierten Datensatzes richtet sich dabei nach der Speicher- und Verarbeitungskapazität des Clients. Während ein Benutzer mit der Visualisierung interagiert, also eine Rotation, Translation, Manipulation der Transferfunktion oder Ähnliches durchführt, kommt der reduzierte Datensatz mit der lokal vorhandenen Graphikhardware zum Einsatz. Sobald der Benutzer diese Interaktion beendet, werden die neuen Visualisierungsparameter zum Server geschickt und ein Bild des vollen Datensatzes angefordert. Die lokal generierten Darstellungen werden mit den endgültigen Bildern des Servers überlagert.

Folglich sind während der Interaktion keinerlei Latenzen durch einen Datentransfer vom Server zu erwarten. Allenfalls nach Beendigung der Interaktion kann sich eine gewisse Verzögerung ergeben, bis das hochqualitative Bild des Servers angezeigt wird. Dies ist aber als relativ unkritisch anzusehen, da Fehlbedienungen durch Latenzen vor allem bei der Interaktion auftreten. Zu beachten ist, dass auf Client und Server konsistente Klassifikations- und Rendering-Verfahren zu Einsatz kommen müssen, um übereinstimmende Darstellungen zu gewinnen. Weiterhin muss bei der Visualisierung von Volumendaten die aus der Transferfunktion gewonnene Opazität bei der Darstellung auf dem Client für die verringerte Schichtenzahl angepasst werden.

6.5 Ergebnisse

Für die folgenden Ergebnisse des vorgestellten Rahmenwerks zur Fernvisualisierung wurde eine SGI Onyx2 InfiniteReality Workstation mit zwei 195 MHz R10000 Prozessoren und 1024 MB Hauptspeicher als Server eingesetzt. Als Client kamen sowohl eine SGI O2 Workstation mit R10000 Prozessor und 128 MB Hauptspeicher, als auch ein 333 MHz Celeron PC mit 64 MB Hauptspeicher zum Einsatz. Die O2 Workstation hatte einen 100 MBit Zugang über eine lokale Fast Ethernet-Verbindung zum Server, während der PC über eine 64 KBit Internet-Verbindung Zugriff hatte. Als Testumgebung kam der medizinische Volumenvisualisierungsdienst zum Einsatz, der einen CT Datensatz mit einer Auflösung von $512 \times 512 \times 106$ Voxeln zu verarbeiten hatte. Dabei wurde eine typische Bildfolge mit einer Auflösung von 704×576 Pixeln generiert.

Die resultierenden Bildwiederholraten auf Server und Client für die verschiedenen Kompressionsverfahren sowie die durchschnittliche Größe der zu übertragenen Bilder sind in Tabelle 6.1 aufgeführt. Während der Interaktion werden die Bilder mit einem Viertel der Originalauflösung (176×144 Pixel) berechnet und übertragen, was schnellere Render-, Kodierungs-, Transfer-, Dekodierungs- und Anzeigeweiten zur Folge hat. Bei der Benutzung der Visualisierungsapplikation auf dem Server selbst wird eine durchschnittliche Bildwiederholrate von 4.1 Bildern pro Sekunde für die volle Auflösung und 25.4 Bildern pro Sekunde für ein Viertel der Auflösung erreicht. Wie nicht anders zu erwarten, sind bei der Nutzung einer ISDN Netzwerkverbindung

Methode	Bildgröße		LOKAL		LAN		ISDN	
	voll	viertel	voll	viertel	voll	viertel	voll	viertel
RAW	1.2 MB	76 KB			1.9	19	-	-
ZLIB	106 KB	6 KB	4.1	25	2.2	14	0.09	1.5
LZO	150 KB	9 KB			2.4	16	0.06	1.3
RLE	160 KB	10 KB			2.5	16	0.06	1.2

Tabelle 6.1: Durchschnittliche Größe der Bilder und Bildwiederholraten bei der Visualisierung eines CT Datensatzes mit einer Auflösung von $512 \times 512 \times 106$ Voxeln. Dabei wurde die Bildwiederholrate lokal, über eine ISDN Internet-Verbindung, sowie eine LAN Fast Ethernet-Verbindung gemessen. Die Auflösung wurde während der Interaktion auf ein Viertel reduziert (176×144 Pixel, 24 Bit), während nach der Interaktion ein Bild in voller Auflösung übertragen wurde (704×576 Pixels, 24 Bit).

für die volle Auflösung keine interaktiven Raten zu erreichen. Während der Interaktion werden Bilder mit nur einem Viertel der Auflösung übertragen, wobei die Bildwiederholraten von ≈ 1.3 gerade noch im Bereich des Zumutbaren liegen. Nachdem ein Benutzer die Interaktion beendet, beispielsweise durch Loslassen eines Manipulators, vergehen aber einige Sekunden bis ein Bild in voller Auflösung von Server übertragen und dargestellt wird. Die besten Bildwiederholraten werden dabei mit dem ZLIB Komprimierungsverfahren erzielt, da dieses wie später deutlich wird, die besten Komprimierungsraten erzielt. Der Transfer der Daten ist also im Falle der niedrigen Bandbreite von ISDN der limitierende Faktor. Bei der entfernten Visualisierung im lokalen Netz über die 100 MBit Ethernet Verbindung erzielte die RLE Komprimierung die besten Werte. Dies lässt sich dadurch erklären dass in diesem Fall die Komprimierung und Dekomprimierung der Daten eine wesentliche Rolle spielt. RLE ist die einfachste und damit schnellste der verwendeten Komprimierungsmethoden. Eine unkomprimierte Übertragung ist erst in einem lokalen Netzwerk in Kombination mit der Reduzierung der Auflösung der Bilder gegenüber den anderen Methoden vorteilhaft.

Kompressionsraten bei der Verwendung von ZLIB, LZO und RLE für eine typische Bildfolge sind in Abbildung 6.3 zu ersehen. Daraus ist zu erkennen, dass ZLIB in allen Fällen die besten Kompressionsraten erzielt. Zwischen LZO und RLE bestehen nur geringe Unterschiede. Die durchschnittliche Komprimierungszeit eines Bildes mit einer Auflösung von 704×576 Pixeln beträgt 180 Millisekunden für ZLIB, 50 Millisekunden für LZO und 30 Millisekunden für RLE. Die durchschnittliche Dekomprimierungszeit beträgt 93 Millisekunden bei ZLIB, 29 Millisekunden bei LZO und 13 Millisekunden bei RLE. RLE ist also das schnellste der implementierten Komprimierungsverfahren, während ZLIB die besten Komprimierungsraten erzielt. Es ist deshalb zu empfehlen, ZLIB auf Übertragungskanälen mit niedriger Bandbreite zu verwenden, während RLE bei Übertragungskanälen mit hoher Bandbreite eingesetzt werden sollte. Erste Experimente mit Codecs, die auch zeitlicher Kohärenz ausnutzen, brachten noch nicht den gewünschten Erfolg[38]. Zwar konnte die Datenmenge nochmals erheblich reduziert werden, der erhöhte Zeitaufwand bei der Kodierung und Dekodierung machte diesen Vorteil allerdings wie-

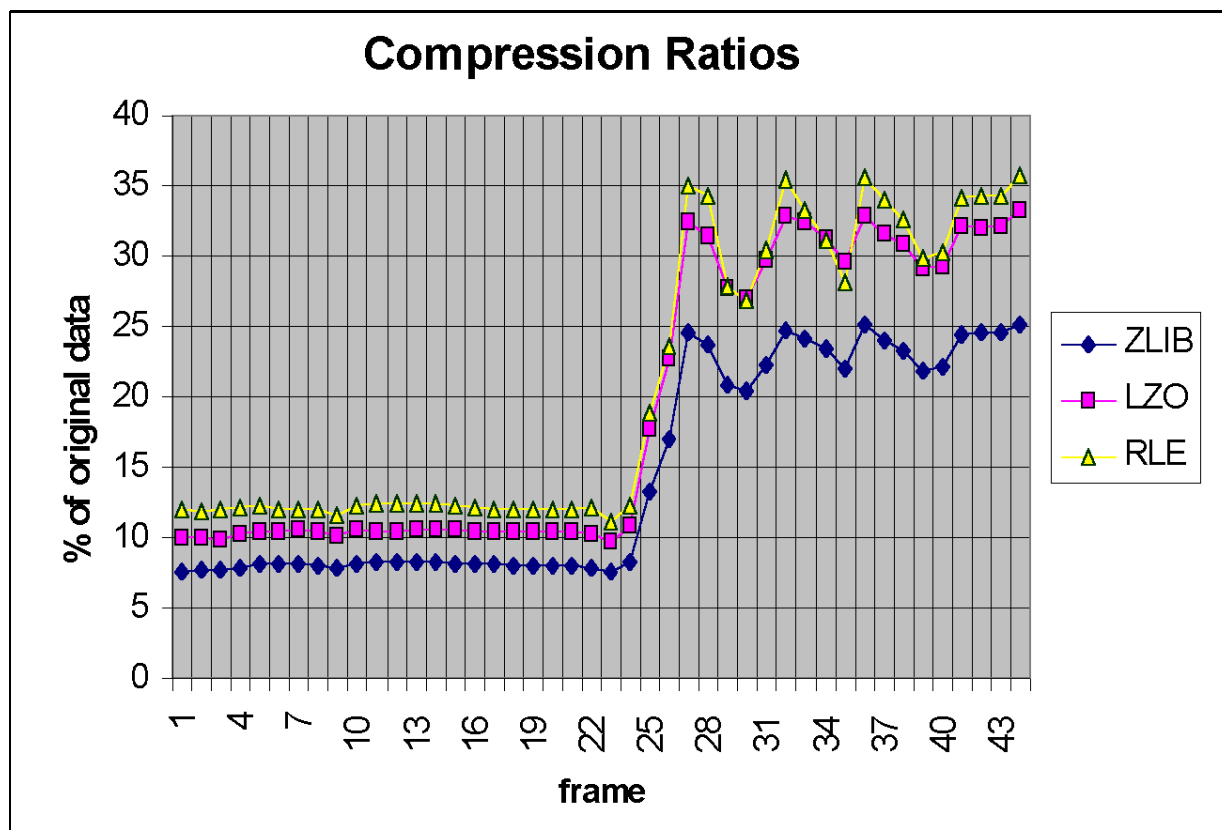


Abbildung 6.3: Kompressionsraten bei Verwendung von ZLIB, LZO und RLE für eine typische Bildfolge. Diese besteht aus einer Rotation des Volumens, gefolgt von einer Vergrößerung.

der zunichte. Im Bereich der Echtzeit-Kompression von Computer-generierten Bildern bedarf es somit noch weiterer Forschungsaktivitäten.

Zusammenfassend kann gesagt werden, dass die Fernvisualisierung eine Reihe neuer Möglichkeiten der Visualisierung komplexer wissenschaftlicher Daten eröffnet. Als Client kann ein einfaches Rastergraphik-fähiges Gerät zum Einsatz kommen, wodurch auch eine Reihe mobiler Applikationen denkbar werden. Zusätzlich kann teure Spezialhardware effektiver genutzt werden und die Kollaboration verschiedener Nutzer einfach realisiert werden. Zum Transfer des Bildstroms auf die Anzeigegeräte ist eine relativ hohe Bandbreite nötig, während der Transfer der Steuerdaten zum Server nur eine sehr begrenzte Netzwerklast erzeugt. Genau dieses Verhältnis liegt bei günstigen Breitband-Internet-Zugängen wie Kabel-Modems, DSL oder Satelliten-Verbindungen vor. Auch im Bereich mobiler Geräte sind die notwendigen Bandbreiten durch GPRS und UMTS gegeben.

Those are my principles,
and if you don't like them...
well, I have others.
Groucho Marx, 1890-1977

Kapitel 7

Ergebnisse

In den vorausgegangenen Kapiteln sind eine Reihe von Strategien und Algorithmen vorgestellt worden, die eine interaktive Visualisierung hochaufgelöster Volumendaten in digitalen Dokumenten zum Ziel haben. Erreicht wurde dieses Ziel durch eine geeignete Verteilungsstrategie zur Aufspaltung der Teilaufgaben der Visualisierungspipeline auf Client und Server. Dabei wird möglichst ein Datenminimum in der Pipeline als Schnittstelle genutzt, um die Netzlast in erträglichen Grenzen zu halten. Daneben sind adaptive Verfahren geeignet, die Datenflut durch das Arbeiten auf unterschiedlichen Auflösungsstufen der Daten schon im Vorfeld einzudämmen. Progressive Methoden erlauben einen schnellen ersten Eindruck der finalen Geometrie bei weit reduziertem Datenaufkommen. Desweiteren stand die effektive Nutzung lokaler oder entfernter Graphikhardware im Mittelpunkt.

Alle diese Aspekte spielen zur Realisierung einer interaktiven Visualisierung eine wesentliche Rolle, ein Universalrezept ist allerdings nicht zu finden. Vielmehr hängt die zu wählende Strategie und der anzuwendende Algorithmus von der konkreten Problemstellung ab (siehe Abbildung 7.1). Wichtige Aspekte sind dabei Fragen betreffend der Bandbreite und der Latenz des Datenkanals, den Graphikfähigkeiten von Client und Server, der Verfügbarkeit von Ressourcen sowie Sicherheitsbelangen. Hinsichtlich der Datenmenge stellen sich Fragen nach Speicher- und Verarbeitungskapazitäten von Client und Server und Bandbreite des Netzwerks. Bei einer Interaktionsrückkopplung zwischen Client und Server stellt sich zusätzlich die Frage, welche Latenzen für einen Nutzer noch akzeptabel sind. Ein weiterer Aspekt ist die Eignung Client- und Server-seitig vorhandener Graphikhardware. Zudem stellt die Verfügbarkeit der Ressourcen bei Nutzung Server-seitiger Hardware vor allem bei größerer Nutzerzahl ein weiteres Problem dar – ein Aspekt, der bei Client-seitigen Strategien kaum eine Rolle spielt. Der Transfer einer Kopie der Daten zum Client wirft zusätzlich Sicherheitsfragen auf, da der Transfer sensibler oder rechtlich geschützter Daten häufig nicht erlaubt oder erwünscht ist.

Es stellt sich nun die Frage, wie sich all diese Aspekte in ein Gesamtbild einfügen lassen. In Abbildung 7.2 sind alle in dieser Arbeit betrachteten Strategien und Algorithmen nochmals zusammengefasst. Insgesamt wurden sechs verschiedene Strategien für indirektes und drei Strategien für direktes Volume-Rendering diskutiert.

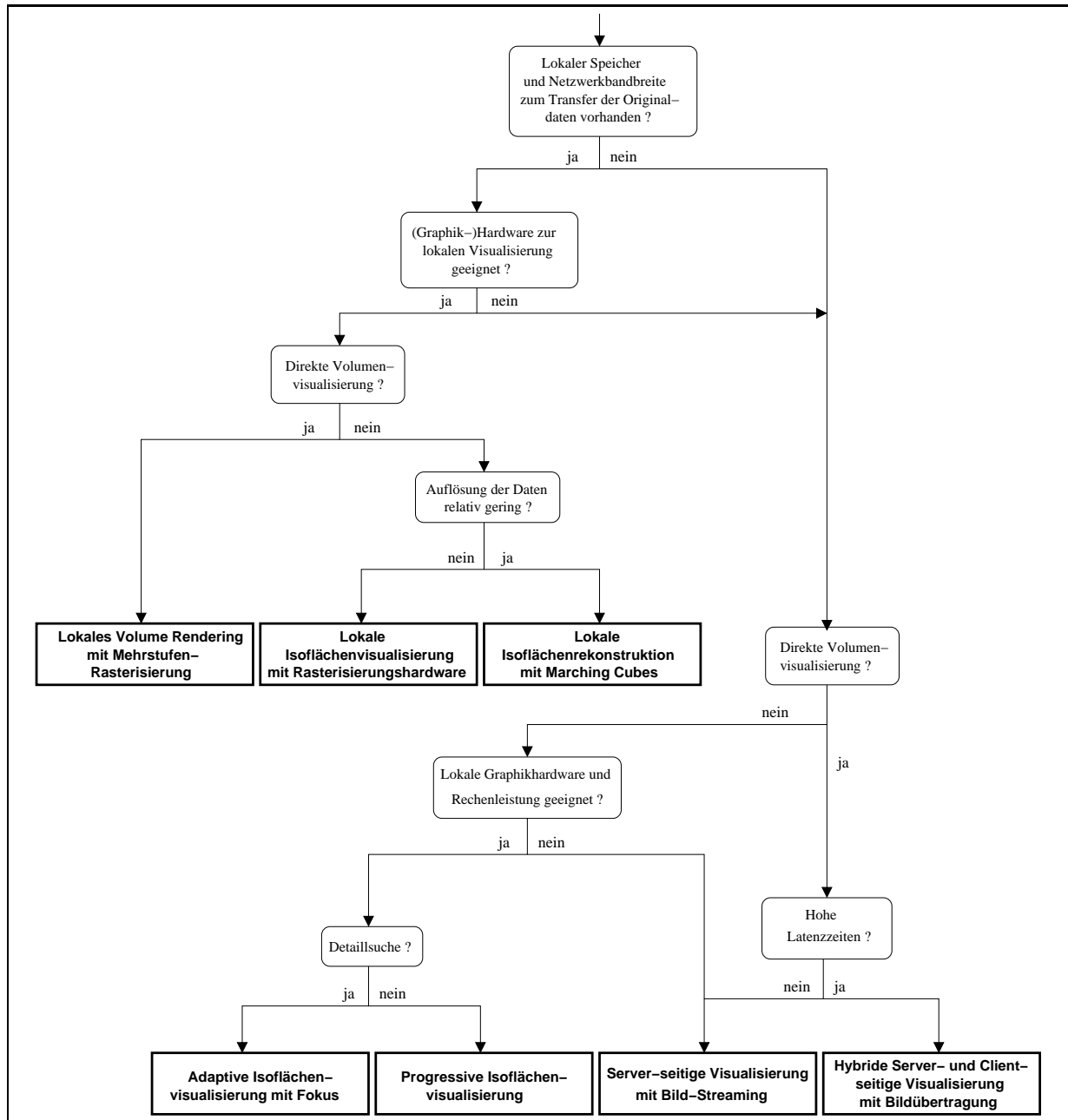


Abbildung 7.1: Entscheidungsdiagramm für die vorgestellten Strategien und Algorithmen.

Es zeigt sich, dass eine Aufspaltung der Visualisierungspipeline bei indirekten Volumenvisualisierungsmethoden in einer größeren Variation als bei direkten Methoden möglich ist. Zurückzuführen ist dies auf die beschränkte Aufspaltbarkeit der Mapping-Stufe bei texturbasierten Verfahren aufgrund der engen Kopplung an die Rendering-Stufe. Es ist im Allgemeinen

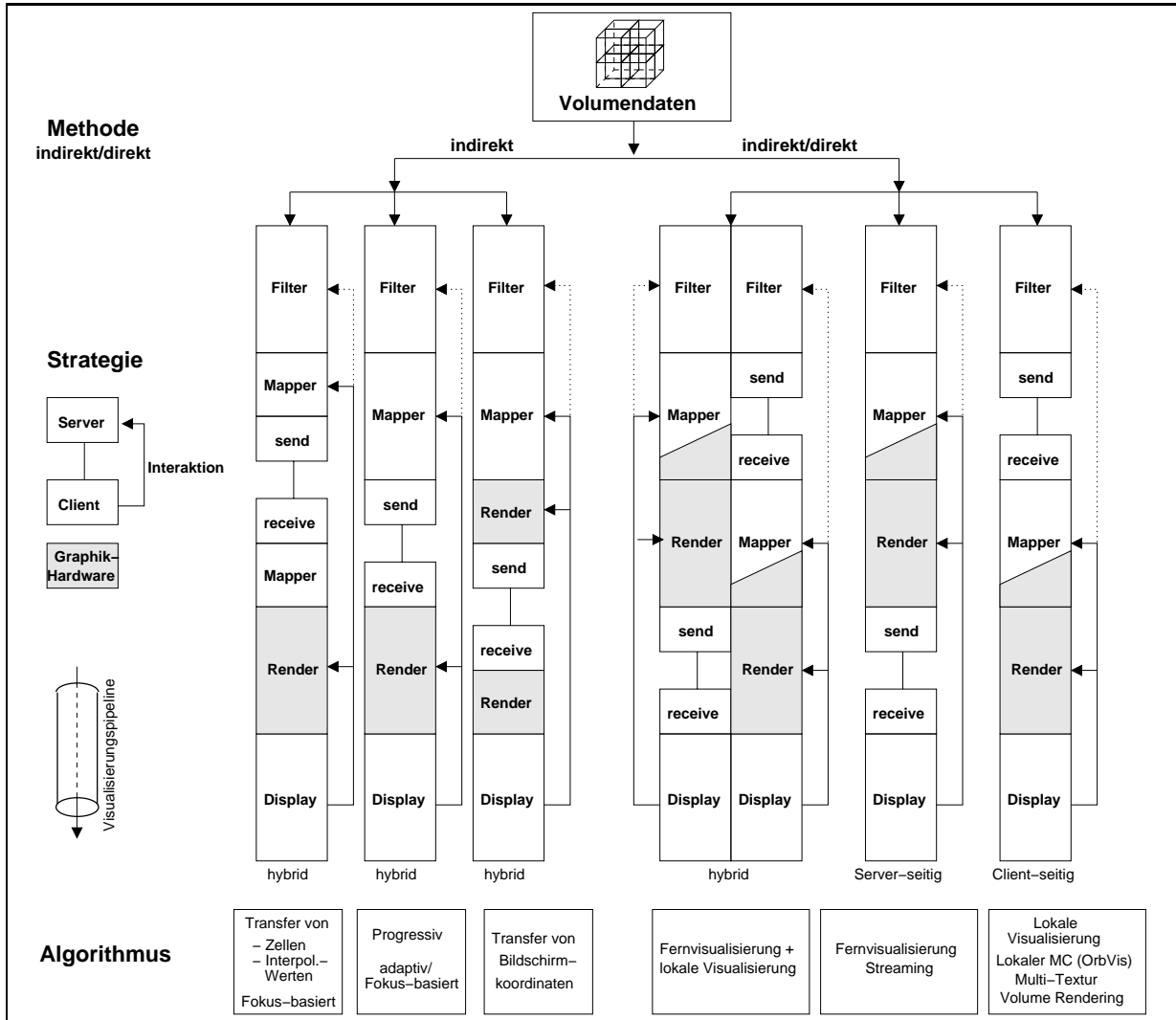


Abbildung 7.2: Gesamtbetrachtung der vorgestellten Strategien und Algorithmen.

sogar keine klare Trennung dieser Stufen möglich, da bei texturbasierten Verfahren das Mapping der Definition von Texturen aus den Volumendaten und der Anwendung einer Transferfunktion entspricht. Diese Arbeitsschritte sind fest mit der Graphikhardware und damit mit dem Rendering-Modul verknüpft.

Natürlich hat diese enge Kopplung der Mapping- und Rendering-Stufe wesentliche Vorteile. So ist bei der Manipulation von Mapping-Parametern kein, im Allgemeinen „teurer“, Datentransfer über ein Netzwerk oder einen Systembus notwendig. In Zukunft wird durch die zunehmende Flexibilisierung der Graphikhardware eine Abarbeitung zusätzlicher Teile der Visualisierungspipeline in Graphikhardware möglich. Schon heute lassen sich Teile der Mapping-Stufe eines Marching-Cubes Algorithmus in Graphikhardware realisieren. So könnten die Vertexpositio-

nen innerhalb einer Zelle durch ein Vertex-Programm aus den Interpolationsgewichten berechnet werden und so Bandbreite des AGP-Bus zum lokalen Speicher der Graphikhardware eingespart werden. Graphikhardware wird also zukünftig in der Lage sein, mehr als nur reine Rendering-Aufgaben zu übernehmen.

Resümierend ist die Verwendung einer Client-seitigen Strategie für viele Fälle zu empfehlen. Auf der einen Seite werden dadurch Server-seitige Ressourcen entlastet, was speziell dann ein wichtiger Faktor ist, wenn viele Nutzer gleichzeitig Zugriff auf einen Visualisierungsdienst haben sollen. Auf der anderen Seite kann lokale Rechenleistung und Graphikhardware genutzt werden, die in vielen Fällen schon heute Hochleistungsgraphikservern überlegen ist. Zudem treten durch Client-seitige Strategien keine Latenzen während der Interaktion durch den Transfer von Parametern zum Server und durch den Rücktransfer von Daten vom Server auf.

Selbstverständlich ist eine solche Strategie in vielen Fällen nicht anwendbar. So kann aus unterschiedlichen Gründen ein Transfer der Daten vom Server zum Client nicht möglich oder nicht erwünscht sein. Bei Nutzung eines einfachen Anzeigegerätes oder wenn die Visualisierung der Daten nur mit spezieller Server-seitiger Infrastruktur möglich ist, so bietet sich eine Fernvisualisierungslösung an.

Oft ist aber ein verteilter Ansatz mit gleichzeitiger Nutzung von Client- und Server-Ressourcen vorteilhaft. Dabei sollte ein Datenminimum in der Visualisierungspipeline als Schnittstelle zwischen Client und Server gewählt werden. Zudem sind progressive und adaptive Algorithmen ideal geeignet, um die von einem Server produzierte Datenflut schon im Vorfeld einzudämmen.

Kirk: Mr. Spock, alle neuen Fakten scheinen unsere Situation noch komplexer zu gestalten.
Spock: Das ist ein unvermeidliches Risiko bei allen wissenschaftlichen Bemühungen, Captain.
Carmen Carter: McCoys Träume, S.93

Kapitel 8

Zusammenfassung und Ausblick

Die vorliegende Dissertation befasst sich mit der interaktiven Visualisierung von 3D-Skalarfeldern in digitalen Dokumenten. Im Folgenden werden kurz die in dieser Arbeit diskutierten Themen wiederholt und die erzielten Ergebnisse zusammengefasst. Nach einer Einführung in die Grundlagen der interaktiven Visualisierung in digitalen Dokumenten wurde zunächst in Kapitel 3 die Client-Server-Visualisierungspipeline eingeführt und der neue Begriff der *Client-Server-Strategien* geprägt. Dieser Begriff dient zur Abgrenzung unterschiedlicher Ansätze zur Visualisierung komplexer wissenschaftlicher Daten in einem Client-Server-Umfeld. Im Wesentlichen wurden drei verschiedene Strategien herausgearbeitet:

Client-seitige Strategien verlagern die Module der Visualisierungspipeline auf einen lokalen Arbeitsplatzrechner, d.h. es werden sowohl die zu visualisierenden Daten als auch der eigentliche, zur Visualisierung notwendige Algorithmus zum Client transferiert. Vorteil dieser Strategie ist im Wesentlichen die Nutzung lokal vorhandener, in den letzten Jahren enorm gestiegener Rechen- und Graphikressourcen, die mit einer entsprechenden Entlastung auf Server-Seite einhergeht. Außerdem treten hier, abgesehen vom Beginn einer Sitzung, wegen der lokalen Interaktion mit den Daten keine Latenzen durch zusätzlichen Datentransfer auf. Ein Nachteil ist, dass die eventuell sehr großen Originaldaten über das Netz transferiert werden und lokal verarbeitbar sein müssen. Dies ist häufig wegen limitierter Bandbreiten und Speicherkapazitäten nicht möglich. Aus Gründen des Schutzes der Originaldaten verbietet sich auch häufig ein solcher Ansatz.

Den Antipod zu diesen Strategien stellen Server-seitige Ansätze dar. Hier wird die Verarbeitung der Daten durch die Module der Visualisierungspipeline auf der Server-Seite vorgenommen. Die so erzeugten Einzelbilder werden in einem kontinuierlichen Datenstrom zum Client-Rechner übertragen und dort mit einfacher 2D-Graphikhardware angezeigt. Zur Interaktion mit dem entfernten Visualisierungsdienst werden lokal generierte Events von Eingabegeräten des Clients zum Server zurückübertragen und dort verarbeitet. Dies eröffnet die Möglichkeit, entfernte Spezialhardware wie Supercomputer oder Graphikworkstations über einen einfachen Client zu nutzen. Dabei werden keine Originaldaten dupliziert und transferiert. Andererseits müssen auf der Server-Seite entsprechende Kapazitäten zur Verfügung stehen. Folglich kann dies bei einer

zeitgleichen Nutzung des Servers durch viele Clients zur Verknappung von Server-Ressourcen führen. Zusätzlich entstehen durch den Transfer der Interaktionsereignisse und des Bilderstroms Latenzen, die bei größeren räumlichen Entfernungen zwischen Client und Server inakzeptabel werden können.

Hybride Strategien haben schließlich zum Ziel, durch eine balancierte Verteilung der Module und Submodule der Visualisierungspipeline sowohl Client- als auch Server-Hardware optimal auszunutzen. Als Schnittstelle zwischen Client und Server bietet sich ein Minimum in der Visualisierungspipeline an. Ähnlich wie bei den Server-seitigen Strategien ist es auch hier nicht notwendig, Originaldaten zu übertragen. Allerdings sind bei diesen Ansätzen Latenzen bei Änderung bestimmter Visualisierungsparameter nicht zu vermeiden.

Die Wahl einer für eine konkrete Visualisierungsaufgabe angemessenen Strategie sollte sich an den vorhandenen Ressourcen orientieren. Für den Fall, dass der Transfer der zu visualisierenden Daten zum Client möglich ist und zusätzlich eine interaktive Visualisierung der Daten auf dem Client realisierbar ist, sollte eine Client-seitige Strategie zum Einsatz kommen. Soll indes ein einfaches Rastergraphik-fähiges Gerät auf Client-Seite genutzt werden oder Spezialhardware auf der Server-Seite zum Einsatz kommen, ist eine Server-seitige Strategie anwendbar. In allen anderen Fällen ist eine hybride Strategie empfehlenswert, die lokale Graphikhardware zur Darstellung nutzt, aber bereits aufbereitete Daten vom Server bezieht.

Um Visualisierungsdienste im Internet einem breiten Publikum mit unterschiedlichsten Netzwerkverbindungen und Client-Architekturen zugänglich zu machen, ist neben einem architekturneutralen Aufbau des Dienstes vor allem die flexible Wahl einer Visualisierungsstrategie zu Beginn einer Sitzung wünschenswert. Nach der Ermittlung der Verbindungsbandbreite, Latenz und Client-Architektur wird dazu ein geeigneter Algorithmus gewählt und die lokal benötigten Module des Visualisierungsalgorithmus werden an den Client zur Ausführung übertragen. Auch eine dynamische Wahl der Client-Server-Strategie, also die ständige Anpassung der Strategie an die Verfügbarkeit von Ressourcen, ist denkbar, wurde allerdings im Rahmen dieser Arbeit nicht realisiert.

Der Frage, in wieweit neuste, flexible PC-Graphikhardware zur Visualisierung von Volumendaten eingesetzt werden kann, wurde in Kapitel 4 nachgegangen. Durch die enge Kopplung der in diesem Kapitel erläuterten Visualisierungsalgorithmen an die programmierbaren Einheiten der PC-Graphikhardware müssen die zu visualisierenden Daten zum Client-Rechner transferiert werden. Da die Daten daraufhin ausschließlich lokal verarbeitet werden, lassen sich die hier vorgestellten Ansätze nach obiger Nomenklatur in die Reihe der Client-seitigen Strategien einordnen. Es wurden im Verlauf der Forschungsarbeiten zu dieser Dissertation drei Phasen durchlaufen, die mit der rapiden Entwicklungen im Bereich der PC-Graphikhardware einhergingen:

Zunächst stand die Nutzung erster Rasterisierungshardware mit 2D-Textur Unterstützung mittels architekturneutraler Programmierschnittstellen im Vordergrund. Dazu wurden die Schnittstellen VRML/EAI, OpenGL-Bindings und Java3D hinsichtlich ihrer Eignung zur interaktiven Visualisierung von 3D-Skalarfeldern untersucht und schließlich anhand Java-basierter Volume-Renderer die Geschwindigkeit dieser APIs verglichen. Es wurde gezeigt, dass eine Implementierung interaktiver texturbasierter Volume-Renderer auf Basis jeder dieser APIs prinzipiell möglich ist. Allerdings zeigten sich bei der Implementierung mit VRML/EAI Probleme durch Unterschiede in den Implementierungen der jeweiligen VRML-Plugins. Sie bestanden in

der Anbindung von Java an VRML über den Event-Mechanismus des External Authoring Interface (EAI) und in fehlender Funktionalität, wie beispielsweise Clipping-Ebenen. OpenGL-Bindings hingegen bieten den vollen Zugriff auf die darunterliegende Graphikhardware, sind aber im Gegensatz zu VRML nicht ISO/IEC standardisiert. Durch das Absetzen jedes einzelnen OpenGL-Kommandos über das Java Native Interface (JNI), lag die Performance der OpenGL-Binding Implementierung unter der mit VRML und Java3D erzielten Framerate. Java3D erreichte die höchsten Frameraten, bietet aber keinen vollen Zugriff auf die Funktionalität der darunter liegenden Hardware, so dass fehlende Funktionen in der API das Einsatzgebiet einschränken.

Diese ersten Ansätze zeigten jedoch schon deutlich das Potenzial PC-basierter Volumenvisualisierungslösungen. Mit der nächsten Generation der PC-Graphikhardware, die mit Mehrstufen-Rasterisierungseinheiten und programmierbaren Pixeloperationen ausgestattet war, wurden eine Reihe neuer Algorithmen erforscht, die teilweise über die Möglichkeiten professioneller 3D-Graphikhardware hinausgingen. Dazu zählen beispielsweise die Verbesserung der Darstellungsqualität durch das Einfügen trilinear interpolierter Schichten ohne Verwendung von 3D-Texturhardware, die Beschleunigung der Darstellung durch die Nutzung mehrerer paralleler Rasterisierungseinheiten, die Darstellung von beleuchteten Isoflächen sowie beleuchteter semi-transparenter Volumina und das Clippen mit beliebigen Clip-Objekten. Schließlich wurde mit einer weiteren Flexibilisierung der Graphikhardware die Implementierung eines völlig neuen Ansatzes zur Visualisierung skalarer Volumendaten möglich.

Dieser Ansatz basiert nicht mehr auf dem traditionellen Verfahren, das trilinear interpolierte Schichten im Bildspeicher überblendet. Vielmehr werden vorintegrierte Strahlsegmente unter Nutzung abhängiger Texturzugriffe in den Bildspeicher gezeichnet. Es wurde gezeigt, dass das Verfahren bei nichtlinearen Transferfunktionen eine sehr viel höhere Bildqualität erreicht als bisherige Ansätze. Desweiteren kann mit dem gleichen Algorithmus eine beliebige Anzahl beleuchteter Isoflächen effizient dargestellt werden. Zusammenfassend wurde demonstriert, dass sich flexible PC-Graphikhardware zur interaktiven, Client-seitigen Visualisierung wissenschaftlicher Volumendaten eignet und dabei in vielerlei Hinsicht professioneller Graphikhardware überlegen ist. Die Vorteile professioneller Graphikhardware sind heute nur noch im Bereich höherer Speicherkapazität, Genauigkeit und Bustransferraten zu sehen. Es ist zu erwarten, dass in der Zukunft noch eine weitere Nivellierung stattfinden wird.

Die Hauptproblematik bei der Nutzung günstiger PC-Graphikhardware liegt in deren beschränkter Speicherkapazität. Dies hat zur Folge, dass Volumendaten mit hoher Auflösung nicht oder nur eingeschränkt visualisiert werden können. Zwar wird die Speicherkapazität von Generation zu Generation gesteigert, doch die bei Weitem höheren Steigerungsraten sind aktuell im Bereich des Hauptspeichers von PCs zu erkennen. Hier können eventuell gerade aufkommende *Shared-Memory*-Architekturen Abhilfe schaffen. Eine andere Möglichkeit ist die effektivere Nutzung des vorhandenen Speichers durch Kompressions- und Multi-Resolution-Ansätze. Durch das Vorhandensein flexibler Einheiten, die Speicherzugriffe mit Indirektionen erlauben, ist man dabei nicht allein auf bereits in heutiger Hardware vorhandene Texturkompressionsverfahren beschränkt. Vielmehr werden hier bereits effektive Algorithmen diskutiert, die den Wert von PC-Graphikhardware weiter steigern könnten.

Fazit dieses Abschnitts ist, dass zumindest für kleine und mittelgroße Volumendatensätze, eine Client-seitige Strategie zu empfehlen ist, falls die entsprechende Hardware lokal vorhanden

ist. Client-seitige Graphikhardware kann hinsichtlich Qualität und Darstellungsgeschwindigkeit hervorragende Ergebnisse erzielen. Dabei muss das Ziel der Client-seitigen Architekturneutralität keineswegs aus den Augen verloren werden, wie dies durch den Einsatz spezifischer Erweiterungen einzelner Hersteller zu vermuten wäre. Vielmehr ermöglichen Java-Anbindungen an OpenGL Zugriff auf die gesamte Funktionalität der auf Client-Seite vorhandenen Graphikhardware. Falls die notwendigen Anforderungen zur Sicherstellung einer interaktiven Visualisierung allerdings nicht gegeben sind, kann eine der alternativen Strategien zum Einsatz kommen, die in Kapitel 6 diskutiert wurden.

Kapitel 5 untersucht am Beispiel der Visualisierung von Isoflächen, welche Verfahren sich zur Optimierung von indirekten Volumenvisualisierungsalgorithmen in einem Client-Server-Umfeld eignen und wie dabei eine balancierte Aufteilung der Aufgaben auf Client und Server erreicht werden kann. Auch in diesem Abschnitt wurde Wert auf die Nutzung architekturneutraler Graphikschnittstellen gelegt. Zunächst wurde ähnlich wie im vorigen Abschnitt ein rein Client-seitiger Ansatz verfolgt. Dazu wurde in der Programmiersprache Java ein *Marching-Cubes*-Modul erstellt, das die Rekonstruktion von Isoflächen aus Skalarfeldern auf einem Client erlaubt. Die Darstellung der Isoflächen erfolgt über eine abstrakte Schnittstelle, von der Module zum Rendering mit VRML, Java3D und OpenGL-Bindings abgeleitet wurden. Die Volumendaten werden zu Beginn einer Visualisierungssitzung zu einem in eine Web-Seite eingebetteten Java-Applet transferiert oder, wie beim Web-Service *OrbVis*, erst lokal aus Parametern, die vom Server übertragen wurden, erstellt.

Um Datensätze mittlerer Größe handhabbar zu machen, wurden einfache Optimierungen über einen Subsamplingmechanismus implementiert. Diese Technik hat sich beim Web-Dienst *OrbVis* durchaus bewährt, wie an der großen Zahl der Nutzer dieses Services zur Visualisierung von Molekülorbitalen deutlich wird. Allerdings ist dieser Ansatz nur für kleine bis mittelgroße Volumendatensätze anwendbar, da bei großen Datensätzen einerseits ein Transfer der Daten auf den Client unmöglich ist und andererseits die Zahl der rekonstruierten geometrischen Primitive die Kapazität heutiger Arbeitsplatzrechner deutlich sprengt. Im weiteren Verlauf wurden deshalb progressive und adaptive Multi-Resolution-Verfahren entwickelt.

Das erste dieser Verfahren basiert auf einer fehlergesteuerten Zerlegung des Skalarfelds in Tetraeder, Oktaeder oder Hexaeder. Daraus resultiert eine adaptive Mehrstufen-Approximation des Volumens, aus der nun Isoflächen mit unterschiedlichem Detailgrad extrahiert werden können. Dieser Ansatz wurde für eine Web-basierte Applikation zur progressiven Visualisierung von Isoflächen genutzt. Aus der auf einem Server gespeicherten Mehrstufen-Repräsentation des Volumens werden mit steigender Genauigkeit Isoflächen extrahiert und zu einem in eine Web-Seite eingebetteten Java-Applet übertragen. Dort werden die Dreiecksdaten mit einem VRML-Plugin unter Einsatz lokaler Graphikhardware dargestellt. Durch eine in das Java-Applet integrierte Level-of-Detail (LOD) Kontrolle kann der Benutzer den Detailgrad der Isofläche festlegen, sich zunächst einen groben Einblick über die Form der Isofläche verschaffen, um dann im weiteren Verlauf zusätzliche Details anzufordern. Dabei werden bei Erhöhung der Detailstufe jeweils nur die Differenzen zur vorigen Stufe übertragen. Somit können auch umfangreiche Volumendaten mit einem Arbeitsplatzrechner untersucht werden.

Obwohl dieser Ansatz schon wesentliche Vorteile brachte, um sich einen Eindruck der Form der Isofläche zu verschaffen, so hatte er doch einen entscheidenden Nachteil: Die Isofläche

wird immer als Ganzes verfeinert. Sucht ein Benutzer nach bestimmten Details, so muss die Isofläche bis zur höchsten Genauigkeitsstufe übertragen werden, wodurch der Vorteil des progressiven Ansatzes verloren geht. Aus diesem Grunde wurde noch ein weiterer Ansatz verfolgt, mit dessen Hilfe über einen Fokussierungsmechanismus einzelne Teile der Isofläche gezielt verfeinert werden können. Dazu wird zunächst eine Octree-Hierarchie des Volumens erstellt, auf deren einzelnen Stufen Isoflächen mit unterschiedlicher Güte rekonstruiert werden können. Diese Octree-Repräsentation des Volumens ist wiederum auf einem Server gespeichert, während ein Java-Applet auf einem Arbeitsplatzrechner die Darstellung der Dreiecke unter Zuhilfenahme eines VRML-Plugins oder einer Java OpenGL-Anbindung unter Einsatz lokaler Graphikhardware übernimmt.

Ein Benutzer kann nun lokal innerhalb der 3D-Szene einen Fokuspunkt interaktiv platzieren und einen Radius um diesen Punkt bestimmen. Diese Parameter werden vom Applet an den Server weitergegeben, der daraufhin innerhalb des Radius um den Fokuspunkt Dreiecke mit hoher Genauigkeit extrahiert, während diese Genauigkeit nach außen hin abfällt. Somit können von einem Benutzer selektiv bestimmte Bereiche einer Isofläche verfeinert und Details durch das Bewegen des Fokuspunktes auf der Isofläche verfolgt werden. Im Zusammenhang mit diesem Verfahren wurde eine Reihe von Strategien erarbeitet, die Teilaufgaben auf unterschiedliche Weise auf Client und Server verteilen. Zusätzlich wurde zur effizienteren Übertragung von Isoflächen ein Verfahren entwickelt, das Isoflächen als Dreiecksstreifen direkt aus den Volumendaten extrahiert. Zusammenfassend ermöglichen es die in diesem Abschnitt vorgestellten Verfahren, auch größere Volumendatensätze mit einem normalen Arbeitsplatzrechner zu untersuchen. Die zwei hybriden Ansätze zur progressiven und zur selektiv-verfeinerbaren Visualisierung von Isoflächen ergänzen sich dabei in perfekter Weise. Während der erste Ansatz durch Bereitstellung eines schnellen, groben Überblicks über die Form der Isofläche die Suche nach einem bestimmten Isowert erleichtert, gibt der zweite Ansatz die Möglichkeit der Suche und des Verfolgens bestimmter Details in einer Isofläche.

Ergänzend zu den oben vorgestellten Ansätzen wurden in Kapitel 6 rein Server-seitige Strategien zur Visualisierung von wissenschaftlichen Daten verfolgt. Wesentliche Grundidee dabei ist, die Visualisierungspipeline komplett auf einen speziellen Server zu verlagern, der dedizierte numerische oder graphische Kapazitäten besitzt, um Visualisierungen der Daten in schneller Bildfolge zu erzeugen. Der Server selbst, oder alternativ ein im Hintergrund arbeitender Rechnercluster, führt alle Teilschritte der Visualisierungspipeline aus und transferiert schließlich die erzeugten Bilder zu einem Client, der ausschließlich für das Anzeigen der Bilder und für das Entgegennehmen von Benutzerinteraktionen zuständig ist. Diese Benutzerinteraktionen werden zum Server gesendet und dort wie lokal generierte Ereignisse verarbeitet. Daraufhin werden neue Bilder generiert, so dass der Interaktionskreislauf geschlossen ist.

Die resultierende entfernte Visualisierung kann bei unterschiedlichsten Voraussetzungen zum Einsatz kommen. So sind beispielsweise wissenschaftliche Datensätze wegen ihrer enormen Größe häufig weder in annehmbarer Zeit zum Client transferierbar, noch dort verarbeitbar. Andererseits sind die heute oder zumindest in naher Zukunft vorhandenen Bandbreiten im Internet durchaus geeignet, einen kontinuierlichen Bilddatenstrom zu übertragen. Die Vielzahl an Geräten, die heute einen Zugriff auf Daten im Internet ermöglichen, stellen ein weiteres Einsatzgebiet für diesen Ansatz dar. Häufig verfügen diese Geräte nicht über ausreichend Speicher- und

Rechenkapazität, um Daten lokal interaktiv zu visualisieren. Andererseits sind sie als einfache Anzeige- und Interaktionsgeräte durchaus geeignet und die benötigten Rechenressourcen im Internet mit ihnen leicht erreichbar. So kann teure Spezialhardware, die an einer beliebigen Stelle auf der Welt steht, mit einem einfachen Gerät wie einem PDA oder Web-Pad, verfügbar gemacht werden. Aus Sicht der Anbieter solch eines Visualisierungsservices bietet sich die Möglichkeit, teure Spezialhardware mit anderen Nutzern zu teilen und damit effektiver und kostendeckender zu nutzen.

Ziel des hier entwickelten Systems war es, Spezial-Graphikhardware des Herstellers *Silicon Graphics* (SGI) über einen beliebigen Java-fähigen Client zu nutzen. Die graphische Benutzerschnittstelle sollte dabei lokal generiert und verarbeitet werden, während das Rendering der Daten entfernt vonstatten gehen sollte. Da dazu eine beliebige bestehende, auf *Open Inventor* oder *Cosmo3D* basierende Visualisierungsapplikation in eine entfernt steuerbare Visualisierungsserver-Applikation konvertiert werden sollte, wurde ein Rahmenwerk geschaffen, das diese Konvertierung vereinfacht. Außerdem unterstützt das Rahmenwerk durch Bereitstellung einer geeigneten Klassenbibliothek die Entwicklung einer entsprechenden Java-basierten Client-Applikation. Basierend auf diesem Rahmenwerk wurden sowohl eine auf *OpenInventor* basierende Applikation zur Visualisierung radiologischer Volumendaten als auch eine auf *Cosmo3D* basierende Anwendung zur Visualisierung von Crash-Simulationen aus dem Automobilbereich in Visualisierungsserver konvertiert. In verschiedenen Testszenarien zeigte sich die hervorragende Eignung der Visualisierungsdienste in der Praxis. Das Problem hoher Latenzzeiten bei weiten Entfernungen zwischen Client und Server wurde zusätzlich durch die Kombination lokaler mit entfernter Visualisierung gelöst. Dazu dient eine hybride Strategie, die während der Interaktion eine Visualisierung in verminderter Qualität lokal vornimmt und nach der Interaktion ein Bild in hoher Qualität vom Server anfordert. Es ist dabei ausreichend, lokal nur eine Kopie der Daten in niedrigerer Auflösung zu speichern.

Die Ergebnisse der vorliegenden Arbeit zeigen neuartige Strategien und Ansätze zur Visualisierung skalarer Volumendaten in digitalen Dokumenten auf. Wesentlich dabei ist, dass die vorgestellten Verfahren sich einfach in digitale Dokumente einbetten lassen und die interaktive Visualisierung der Daten in nahezu jedem Client-Server-Umfeld gewährleistet ist. Durch die Heterogenität der Ressourcen im Internet stellt dies eine besondere Herausforderung dar.

Ein wesentliches Ergebnis dieser Arbeit ist sicherlich die Bedeutung der enorm gestiegenen Fähigkeiten und Kapazitäten moderner Arbeitsplatzrechner zur interaktiven Visualisierung wissenschaftlicher Daten. Diese sollten soweit möglich in einem Client-Server-Umfeld genutzt werden, da eine lokale Verarbeitung der Daten Netzwerklatenzen vermeidet und außerdem Bandbreiten und Serverressourcen schont. Trotzdem ist davon auszugehen, dass es ungeachtet des riesigen Leistungszuwachses preisgünstiger Graphikhardware, in Zukunft eine Klasse von Hochleistungsrechnern geben wird, die die Fähigkeiten typischer Arbeitsplatzrechner hinsichtlich Speicherkapazität, Busbandbreiten und Verarbeitungsgeschwindigkeit übersteigen. Deshalb haben auch hybride oder Server-seitige Ansätze weiterhin ihre Berechtigung.

Zukünftig ist zu hoffen, dass sich erweiterbare, standardisierte Formate für die Einbettung von 3D-Szenen in digitale Dokumente durchsetzen werden. Dabei ist sicherlich die Abstraktion der darunter liegenden Graphikhardware ein wesentlicher Aspekt. Es sollte aber auch nicht außer Acht gelassen werden, dass zur Nutzung von Spezialfunktionalität ein Zugriff auf Low-

Level-Funktionen der Graphikhardware eminent wichtig ist. So kann sowohl eine große Zahl unterschiedlicher Clients unterstützt werden, als auch die Fähigkeiten eines modernen Clients optimal genutzt werden. Einen Schritt in die richtige Richtung gehen prozedurale, echtzeitfähige Shading-Sprachen[94], wie sie auch in DirectX9 und OpenGL 2.0 geplant sind.

Die Einbettung hochaufgelöster Volumendaten in digitale Dokumente in Verbindung mit einer interaktiven Visualisierung ist mit den in dieser Arbeit vorgestellten Strategien und Algorithmen bereits heute möglich. Um mit dem ständigen Fortschritt der Mess- und Simulationstechnik und der damit einhergehenden Steigerung der Auflösung der Volumendatensätze Schritt halten zu können, sind „intelligente“ Verteilungsstrategien, die Optimierung aller Stufen der Visualisierungspipeline und die Nutzung neuester, hochgradig optimierter Hardware notwendig. Im Bereich der Hardware ist die höchste Leistungssteigerung augenblicklich im Bereich der Graphikhardware, insbesondere bei der PC-Graphikhardware im Spiele-Bereich, zu beobachten. Diese Entwicklung beeinflusst nachhaltig die Entwicklung moderner Visualisierungsalgorithmen und lässt zukünftig auf neue Einsichten in komplexe wissenschaftliche Daten hoffen.

For a moment, nothing happened.
Then, after a second or so,
nothing continued to happen.
Douglas Adams, 1952-2001

Kapitel 9

Farbseiten

Auf den folgenden Farbseiten befindet sich eine Sammlung verschiedener Farbbilder, die aus drucktechnischen Gründen von den zugehörigen Kapiteln getrennt wurde.

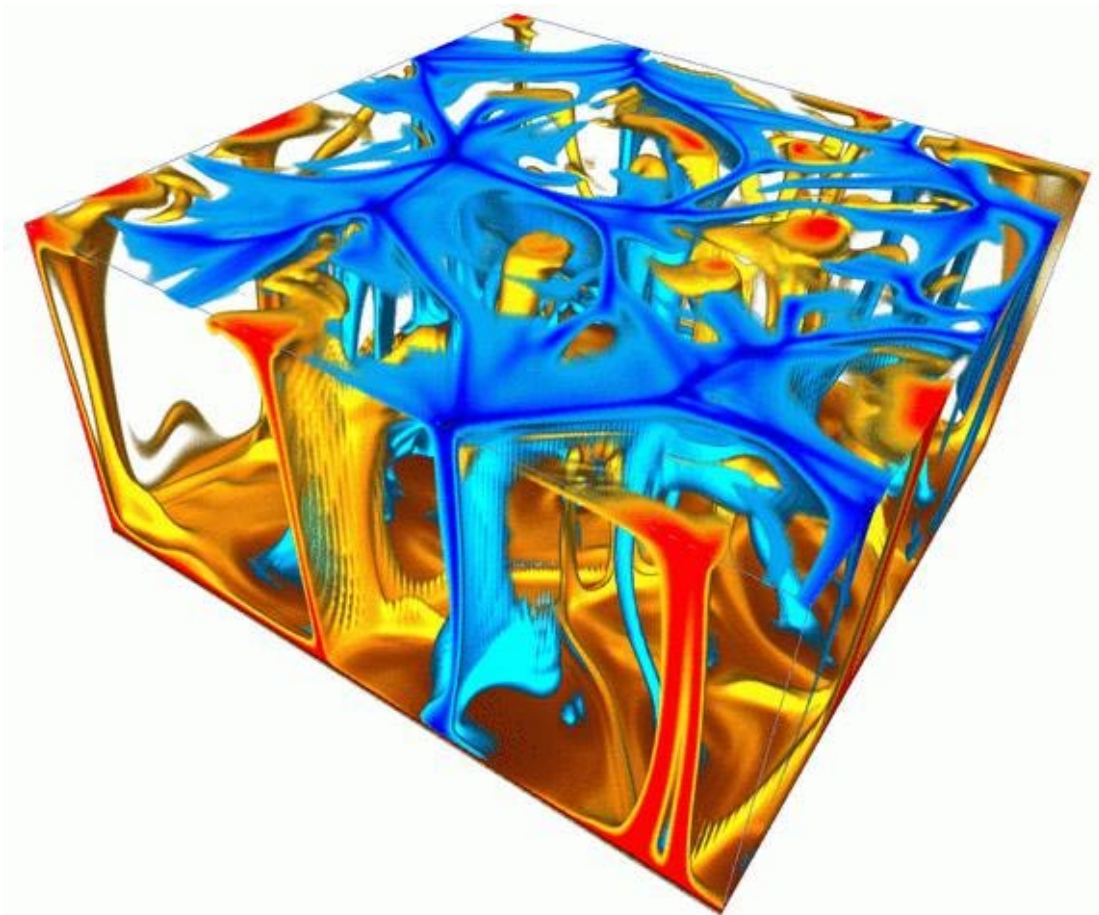


Abbildung 9.1: Textur-basierte Visualisierung einer Simulation von Konvektionsströmungen in der Erdkruste.

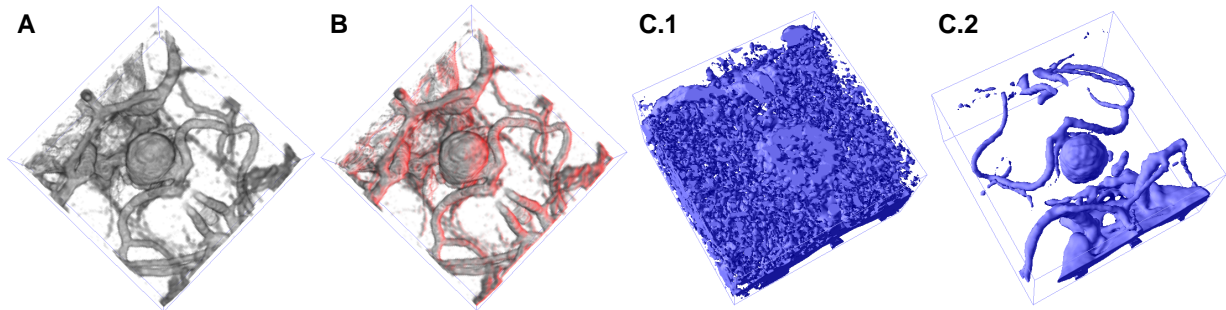
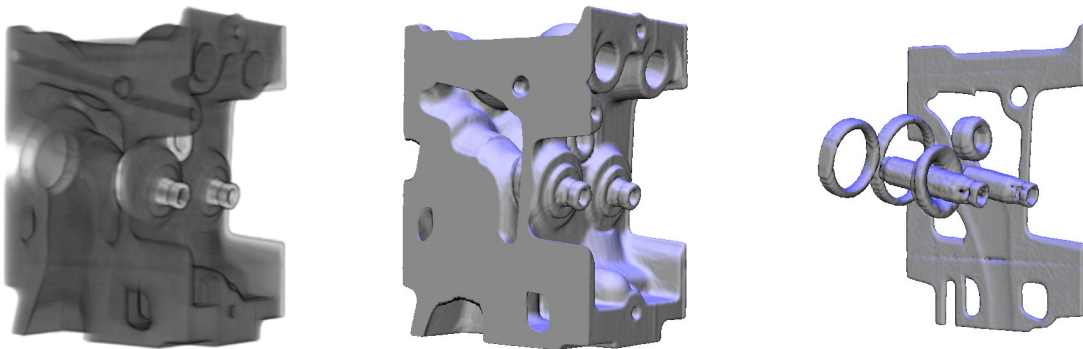


Abbildung 9.2: Multi-Textur-Volumenvisualisierung eines CTA Aneurysma-Datensatzes ($128^2 \times 64$): (A) Direktes Volume-Rending ohne Beleuchtung, (B) Direktes Volume-Rending mit roter diffuser Lichtquelle, (C) beleuchtete Isoflächen für verschiedene Isowerte.

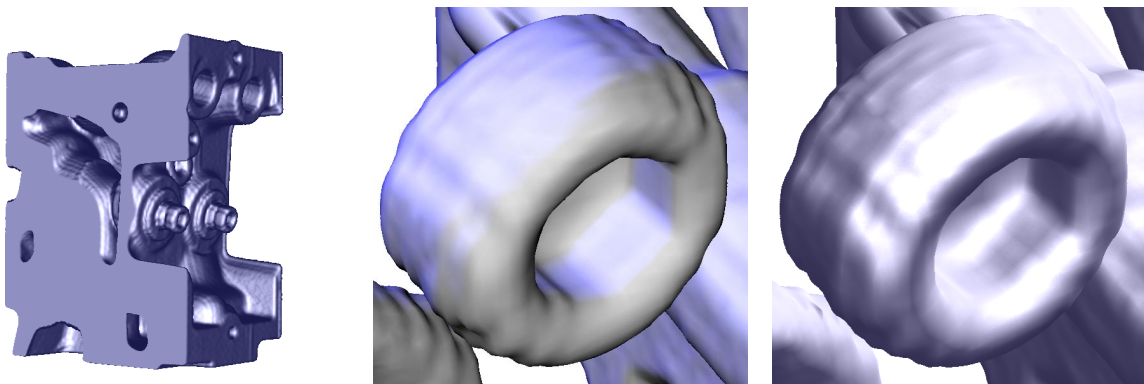


Direkte Darstellung.

Diffus beleuchtete Isofläche
(niedriger Isowert).

diffus beleuchtete Isofläche
(hoher Isowert).

Abbildung 9.3: Multi-Textur-Volumenvisualisierung eines Motorblocks (Datensatz-Größe: $256 \times 256 \times 110$)



Spekulare Beleuchtung.

Detail, diff. Beleuchtung.

Detail, spek. Beleuchtung.

Abbildung 9.4: Alpha-Test-basierte Isoflächen-Darstellungen mit Beleuchtungsberechnung in den Register-Combinern.

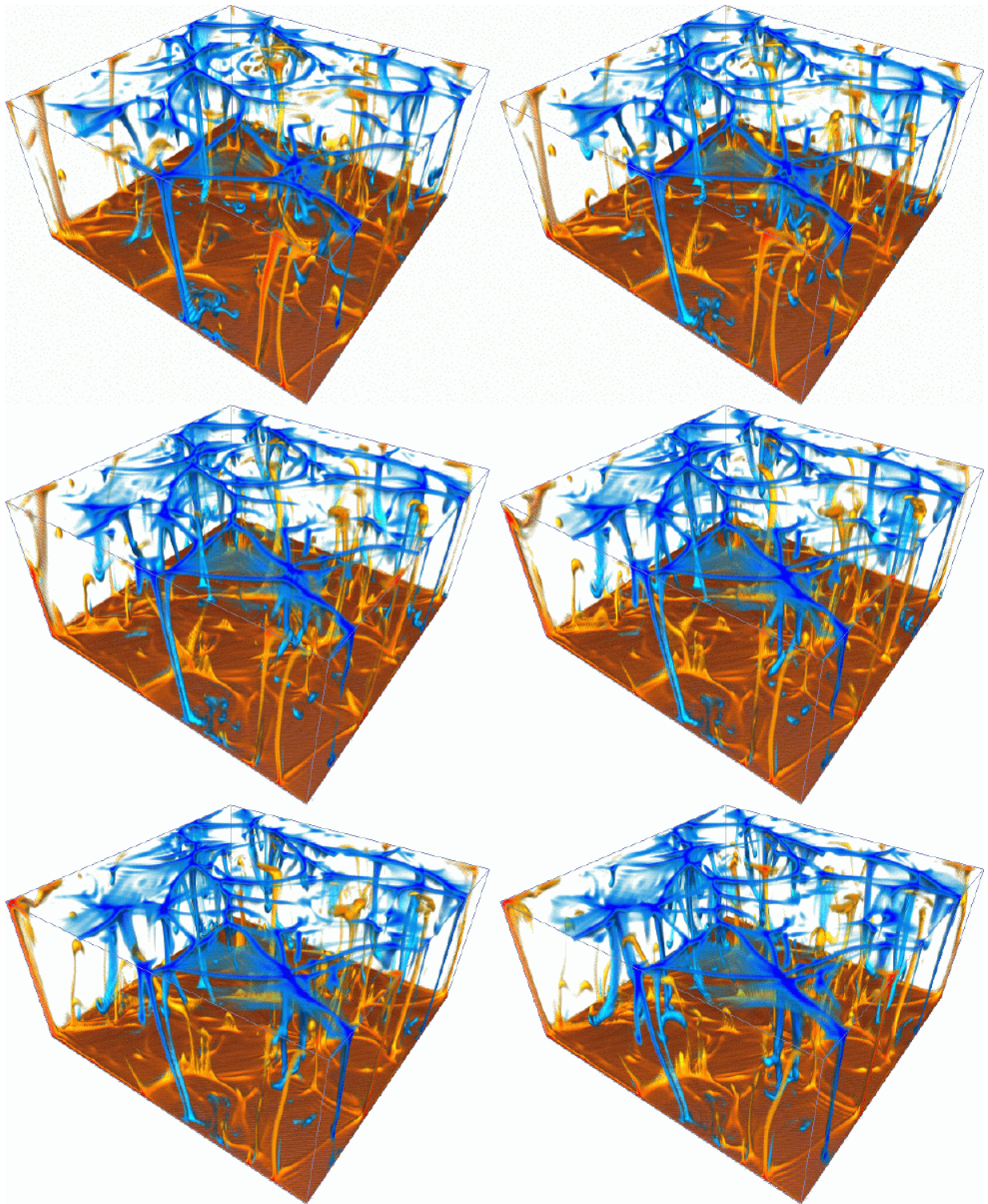


Abbildung 9.5: 6 von insgesamt 30 Zeitschritten einer Simulation von Konvektionsströmungen in der Erdkruste. Durch die Interpolation der Volumen aufeinanderfolgender Zeitschritte ist eine ruckfreie Animation möglich.

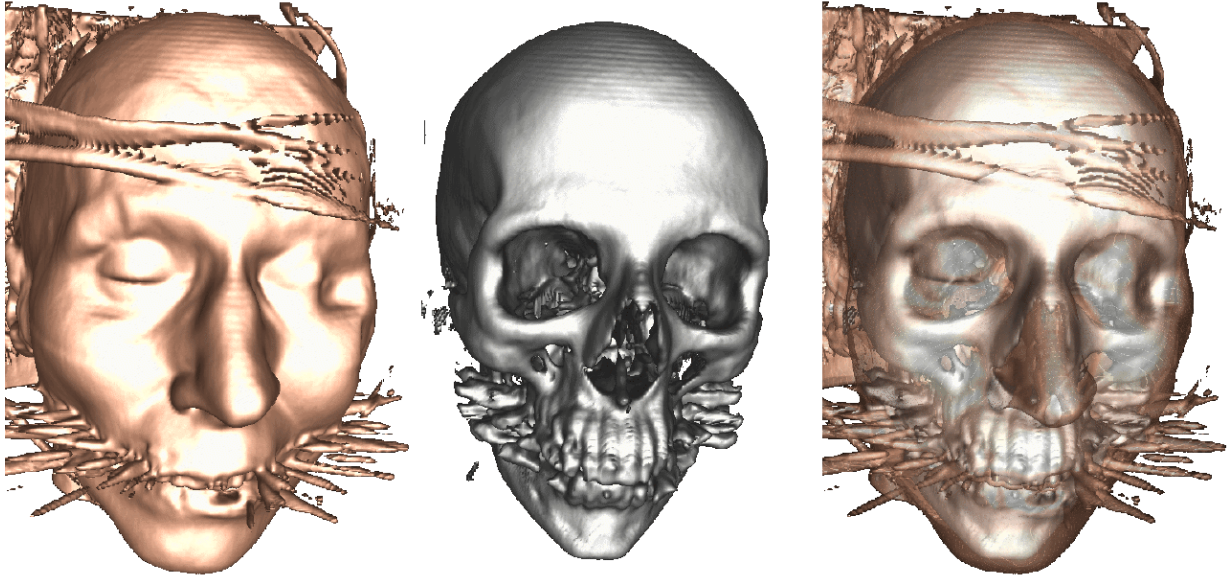


Abbildung 9.6: Vorintegrierte Isoflächenvisualisierung: CT Aufnahme eines menschlichen Kopfes (256^3 Voxel Auflösung): Haut, Knochen, halbtransparente Haut mit opakem Knochen.

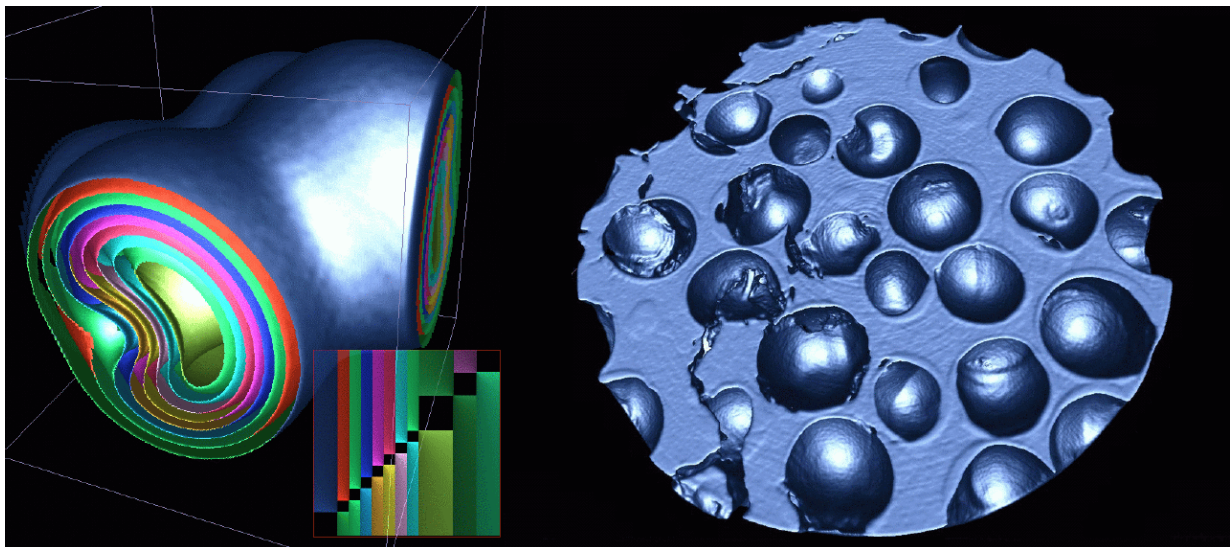


Abbildung 9.7: Vorintegrierte Isoflächenvisualisierung: Mehrere unterschiedlich eingefärbte Isoflächen eines synthetischen Datensatzes und die dazugehörige abhängige Textur (links). Isofläche einer Mikro-CT Aufnahme eines Metallschaums (rechts).

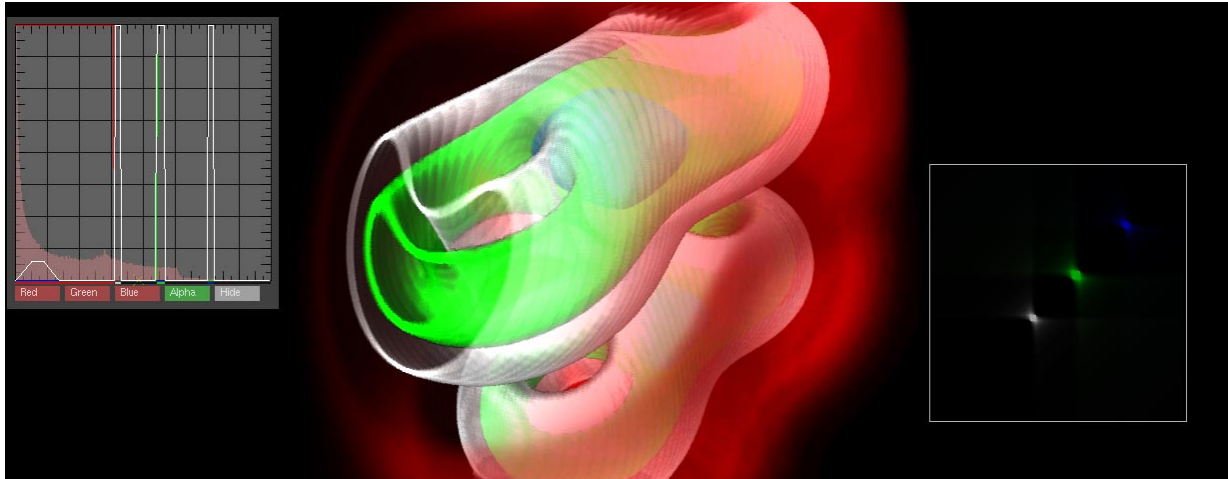


Abbildung 9.8: Mischung von semi-transparenter Darstellung und Isoflächen beim Vorintegrierten Volume-Rending eines synthetischen Datensatzes. Die Isoflächen wurden durch Spitzen in der Transferfunktion erzeugt (links). Die daraus resultierende abhängige Textur ist rechts abgebildet.

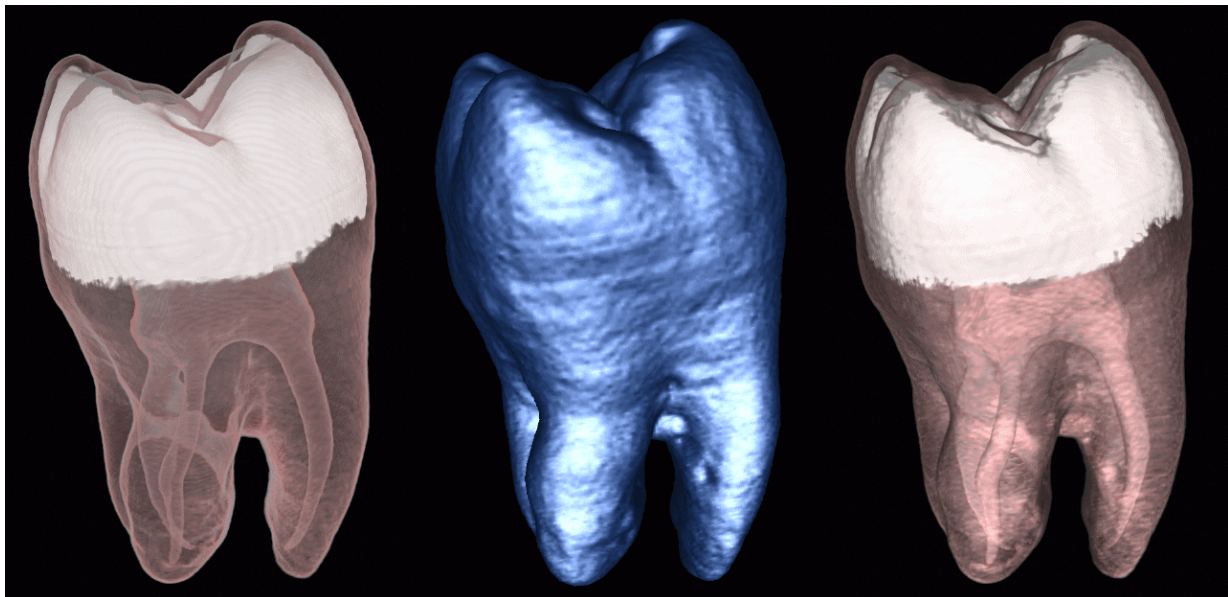


Abbildung 9.9: Vorintegriertes Volume-Rending eines Zahn-Datensatzes: direktes Volume-Rending (links), Isofläche (mitte), Volume-Shading (rechts).

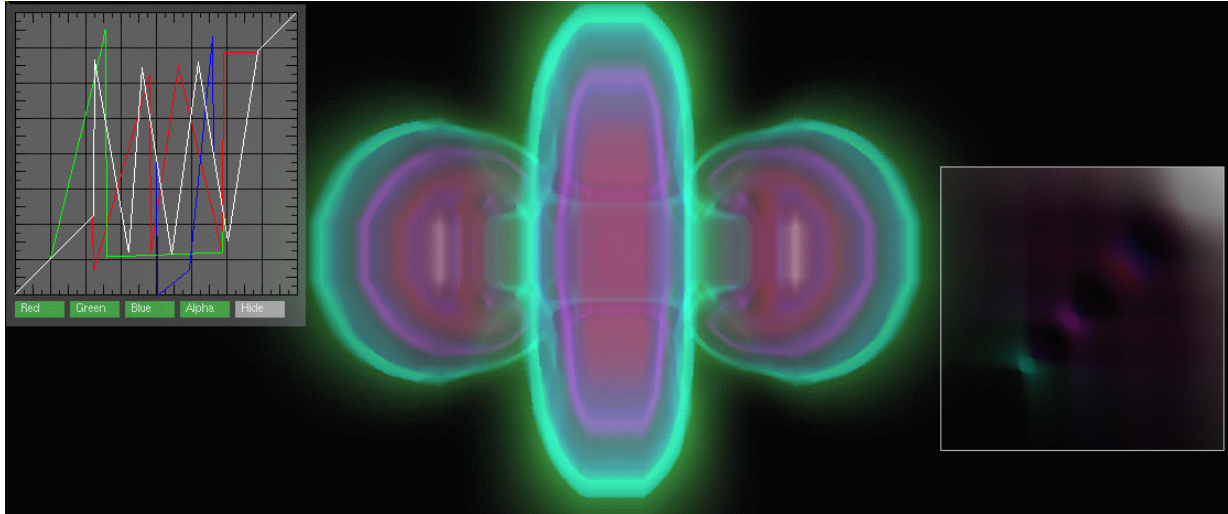


Abbildung 9.10: Vorintegriertes Volume-Rending einer sphärischen, harmonischen Funktion mit nicht-linearer Transferfunktion (links) und zugehöriger abhängiger Textur (rechts). 16 Schichtenpolygone, 16^3 Voxel Auflösung.

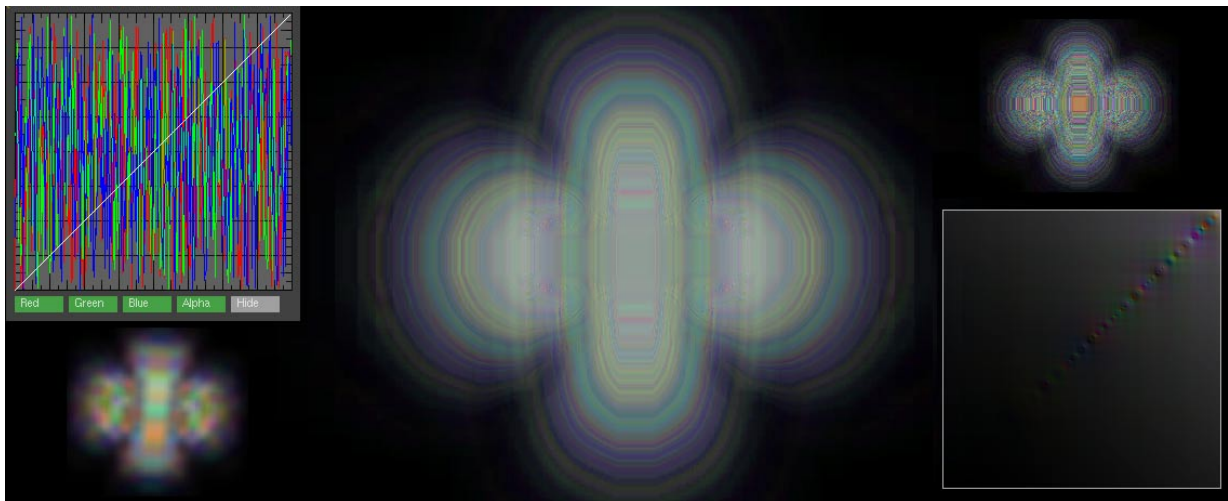


Abbildung 9.11: Vorintegriertes Volume-Rending einer sphärischen, harmonischen Funktion mit hochfrequenter (randomisierter) Transferfunktion (oben, links) und zugehöriger abhängiger Textur (unten, rechts). 16 Schichten wurden jeweils gezeichnet (Datensatz-Auflösung: 16^3 Voxel). Die Ergebnisse von Präklassifikation (unten, links) und Postklassifikation (unten, rechts) sind zum Vergleich abgebildet.

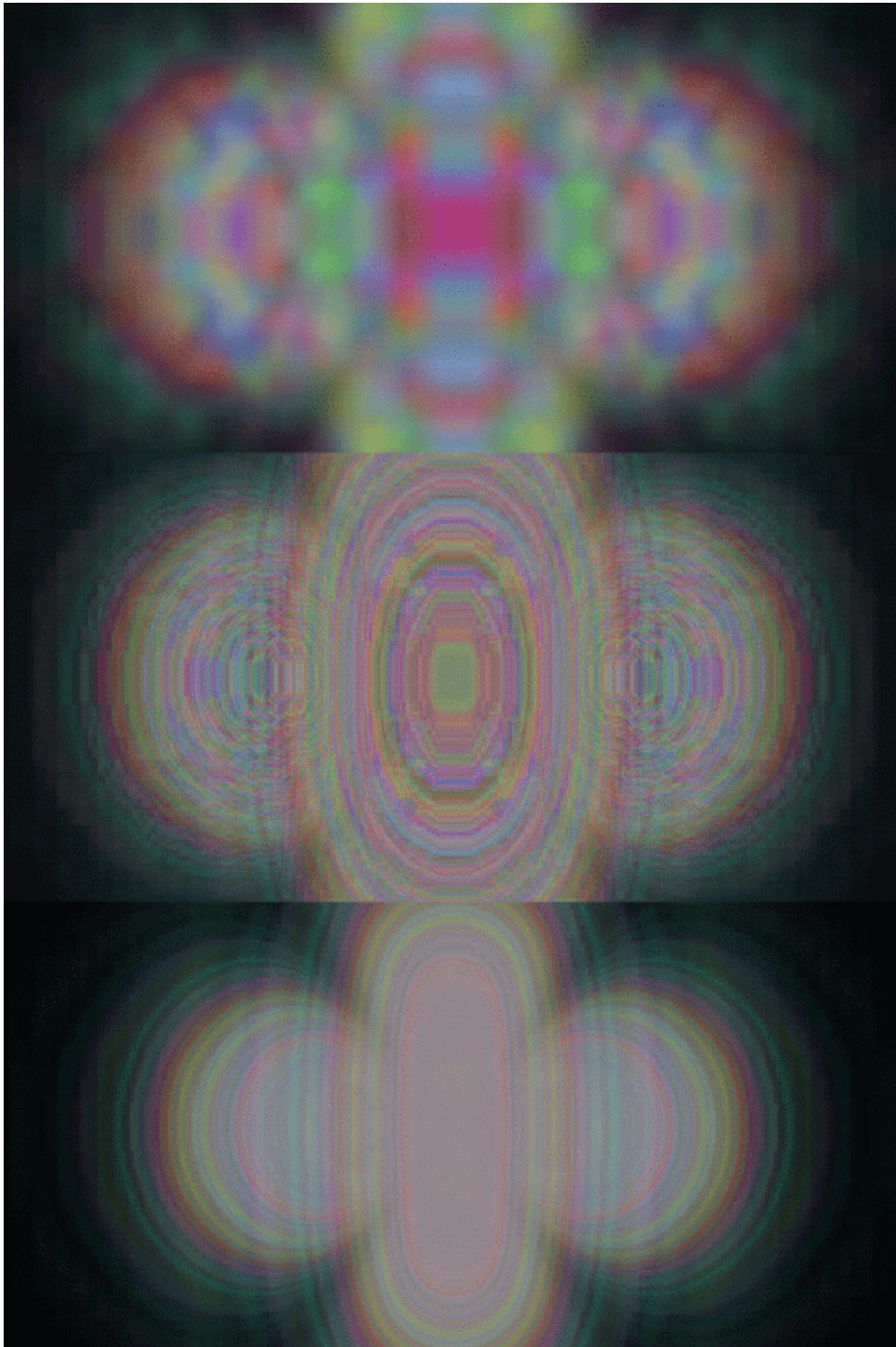


Abbildung 9.12: Visualisierung einer sphärischen, harmonischen Funktion mit randomisierter Transferfunktion (32 Schichtpolygone, 32^3 Voxel). Vergleich der Ergebnisse von Präklassifikation (oben), Postklassifikation (mitte) und Vorintegration (unten).

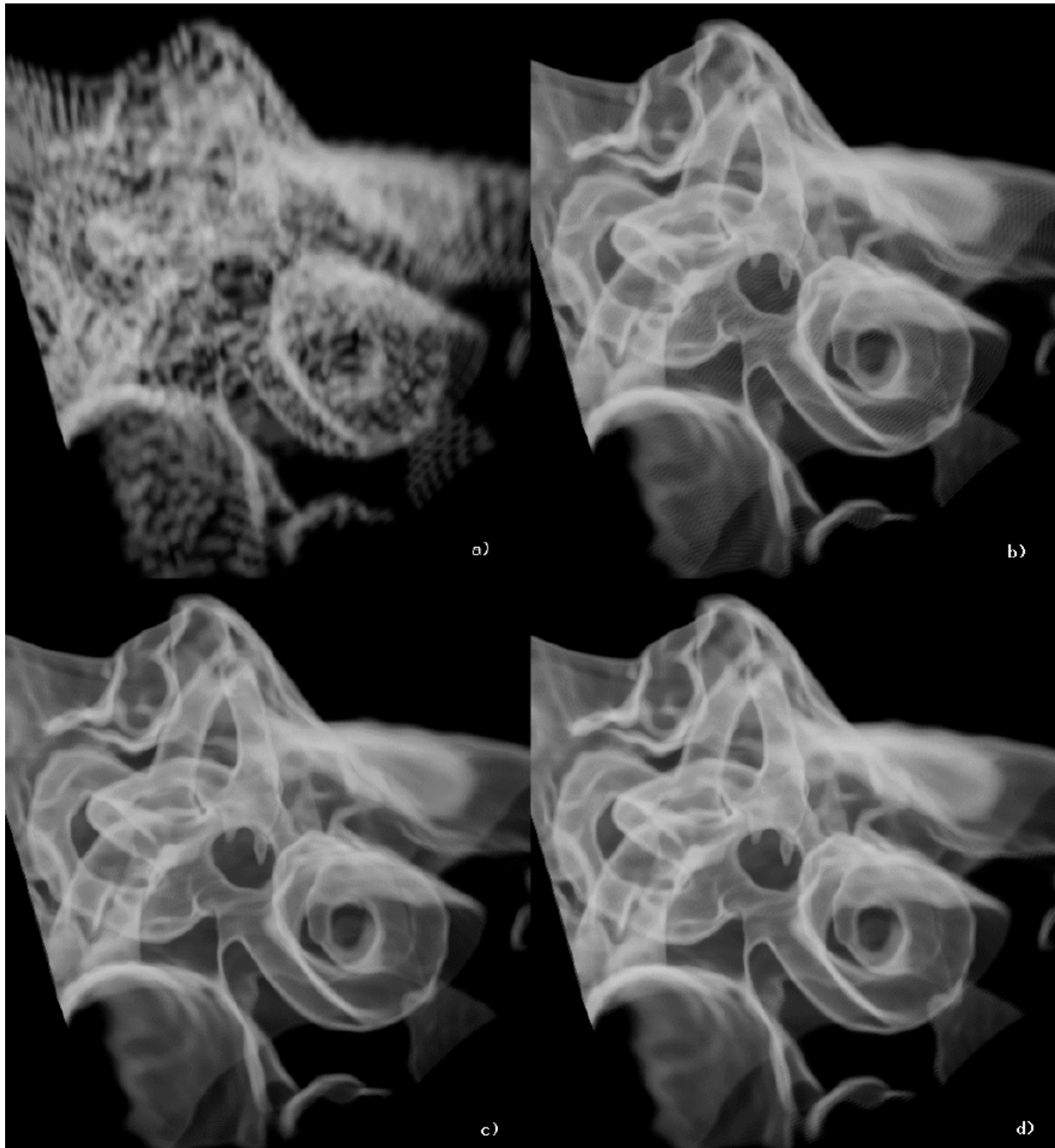
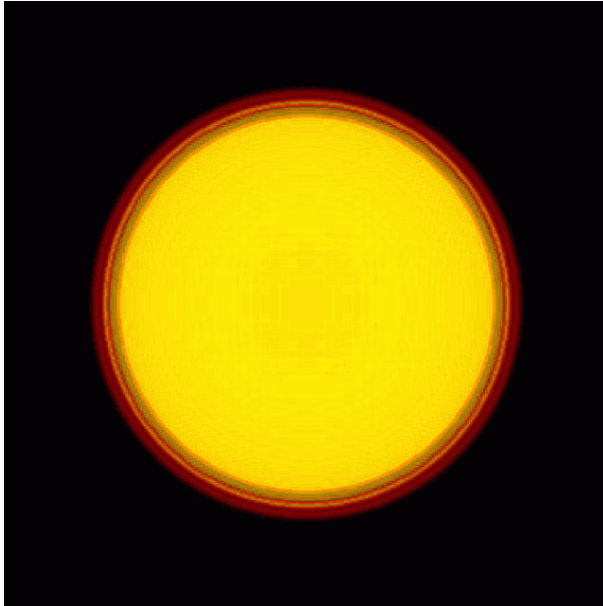
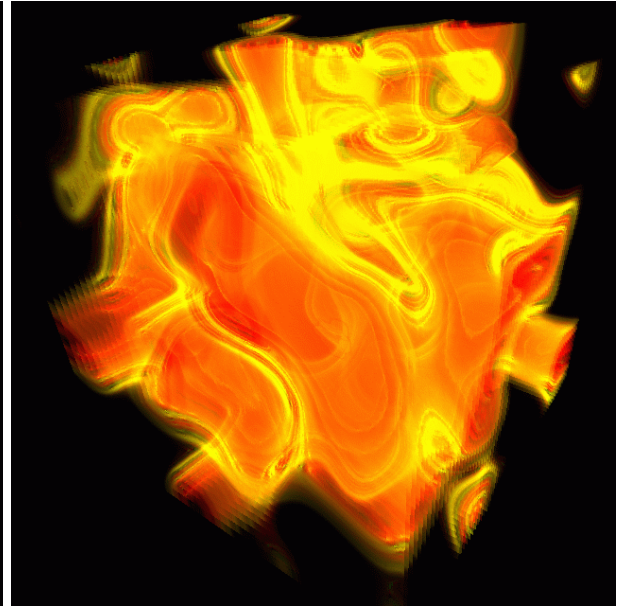


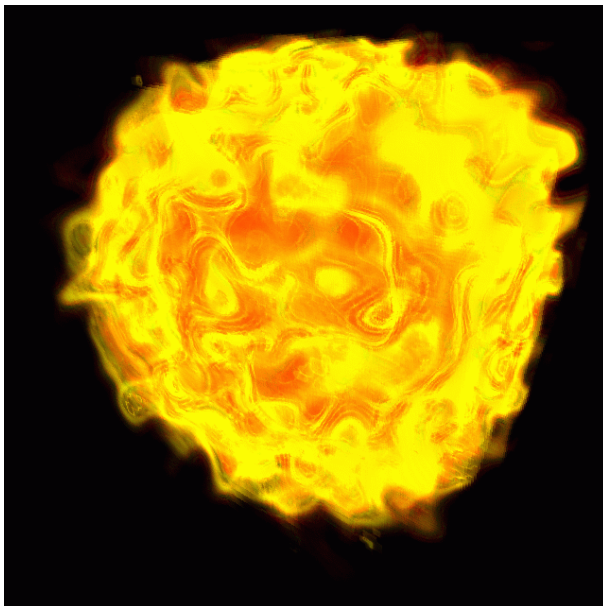
Abbildung 9.13: Vergleich verschiedener Klassifikationsverfahren bei der Visualisierung kleiner Strukturen des Innenohrs[58] (Auflösung des Datensatzes: $128 \times 128 \times 30$ Voxel): a) Präklassifikation (128 Schichten), b) Postklassifikation (128 Schichten), c) Postklassifikation (284 Schichten) und d) Vorintegration (128 Schichten).



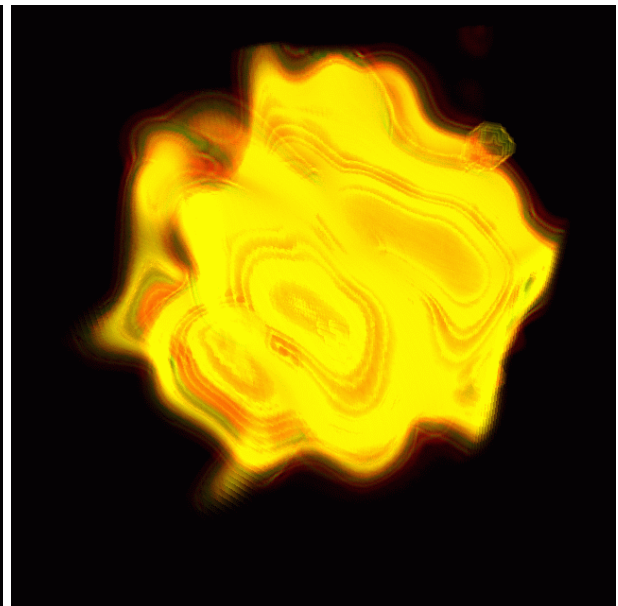
(a) Radiales Distanzvolumen mit hochfrequenter Feuer-Transferfunktion



(b) Perlin-Noise-Volumen mit Feuer-Transferfunktion



(c) Gewichtete Kombination des Distanzvolumens und zweier Perlin-Noise-Volumina



(d) Wie (c), aber mit erhöhter Gewichtung der Perlin-Noise-Volumina

Abbildung 9.14: Vorintegriertes Volume-Rendering eines Feuerballs. Verschiedene Volumina können während des Renderings gemischt werden.

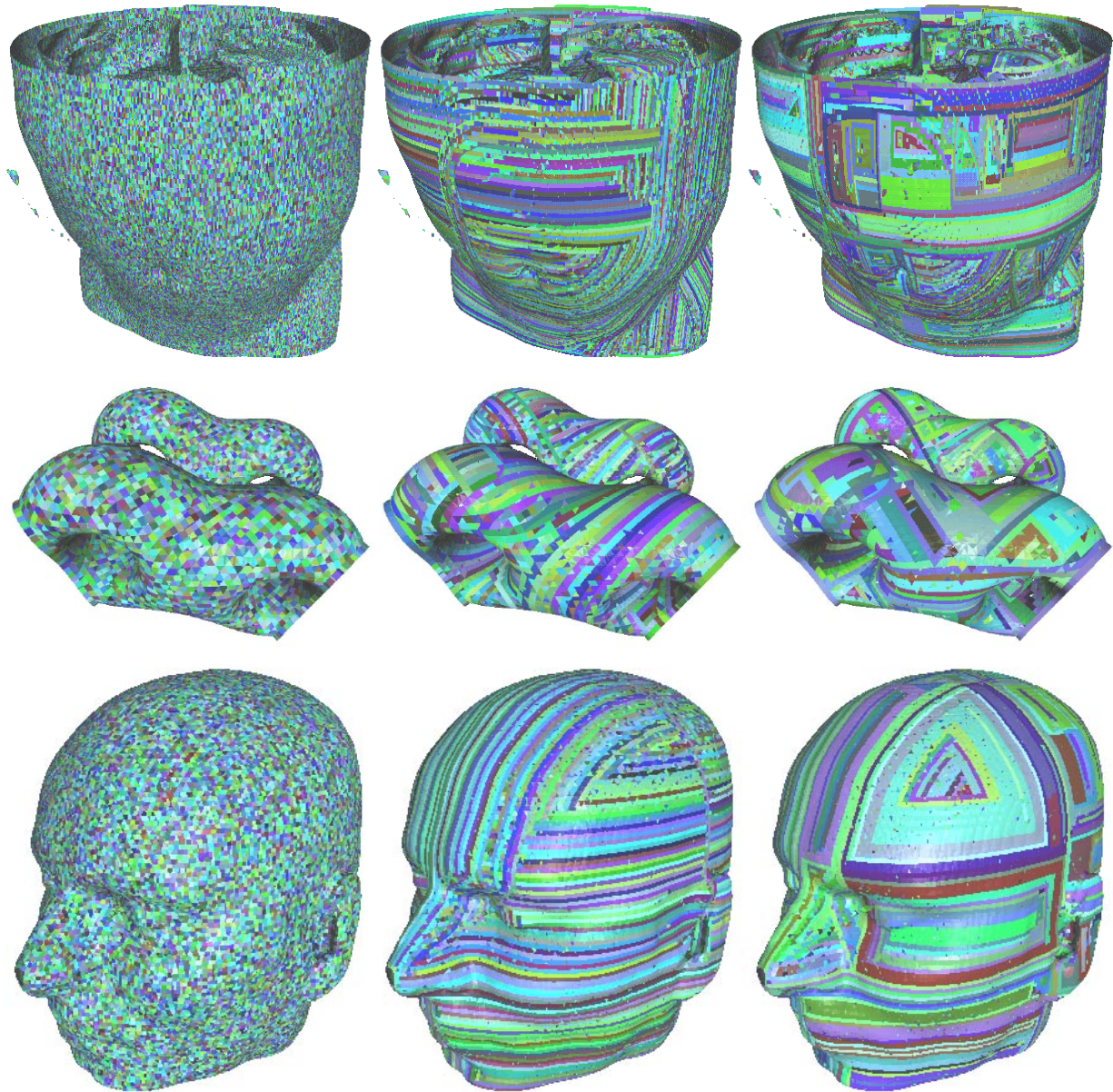


Abbildung 9.15: Ergebnisse der verschiedenen Ansätze zur direkten Rekonstruktion von Dreieckstreifen aus Volumendaten (Streifen sind farblich gekennzeichnet). Links: Streifen sind nicht über Zellgrenzen hinaus verbunden. Mitte: Streifen sind über Zellgrenzen hinaus in eine Vorzugsrichtung verbunden. Rechts: Streifen sind über Zellgrenzen hinaus verbunden, verschiedene Richtungen sind dabei möglich. Die durchschnittliche Streifenlänge beträgt 1,9, 7,7 und 13,4, von links nach rechts.

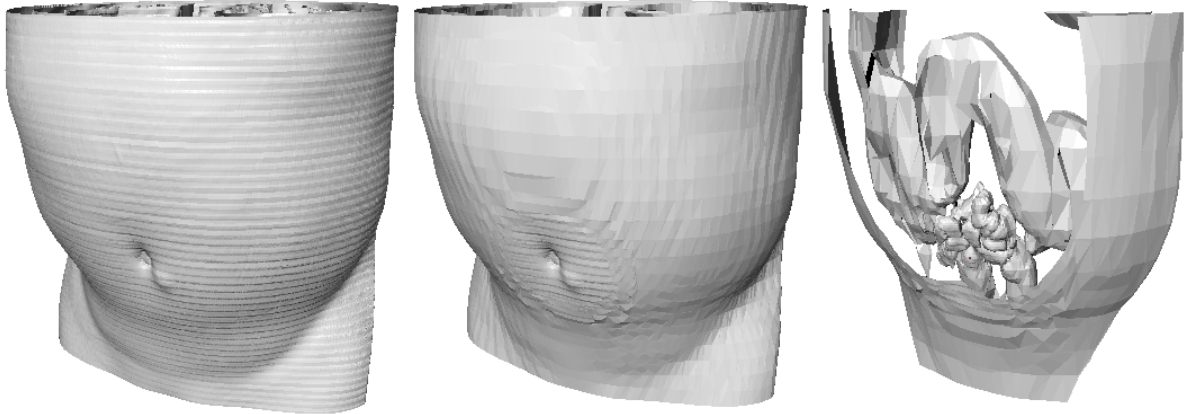


Abbildung 9.16: Hierarchische Level-of-Detail-Rekonstruktion mit einer Octree-Repräsentation des Volumens. Links: Isofläche aus den originalen Abtastpunkten (2.3 Millionen Dreiecke). Mitte: LOD Repräsentation mit dem Fokus um den Bauchnabel (135 Tausend Dreiecke). Rechts: Zusätzliche Nutzung zweier Objektraum-Clipping Ebenen.

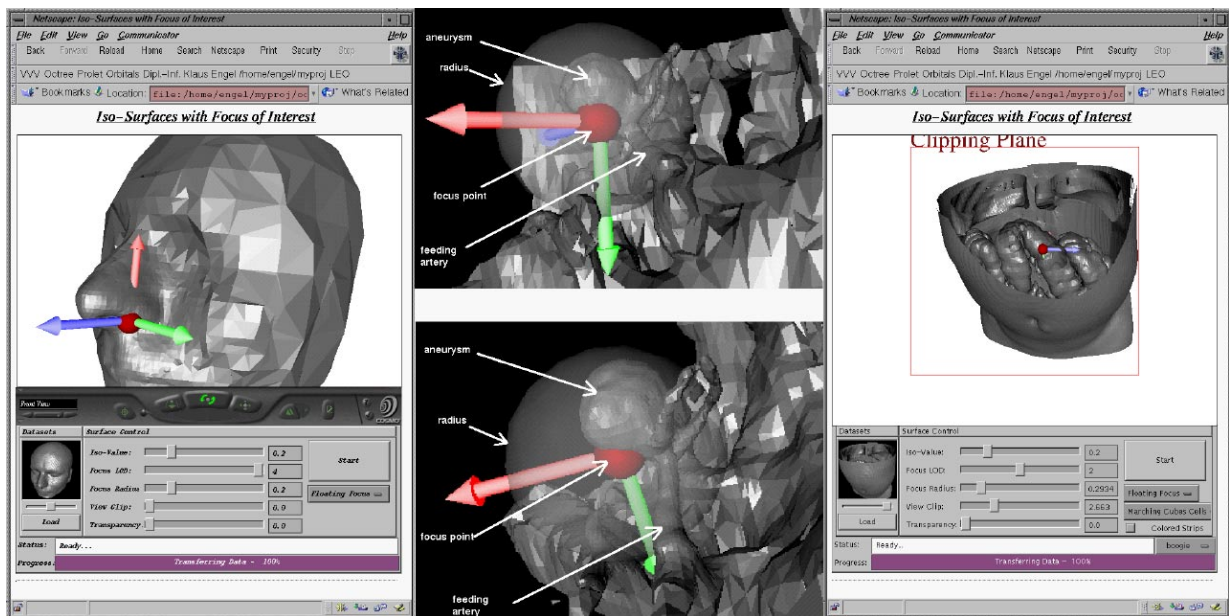


Abbildung 9.17: Client-Applikation zur Fokus-basierten Visualisierung von Isoflächen, die auf einem entfernten Rechner rekonstruiert werden. Links: VRML-Visualisierung mit Fokuspunkt. Mitte: Interaktives Bewegen des Fokuspunktes entlang einer zerebralen Arterie zu einem Aneurysma (Beeren-förmige Blase der zerebralen Arterie). Rechts: VRML-Visualisierung mit Fokuspunkt und Clipping-Ebene.

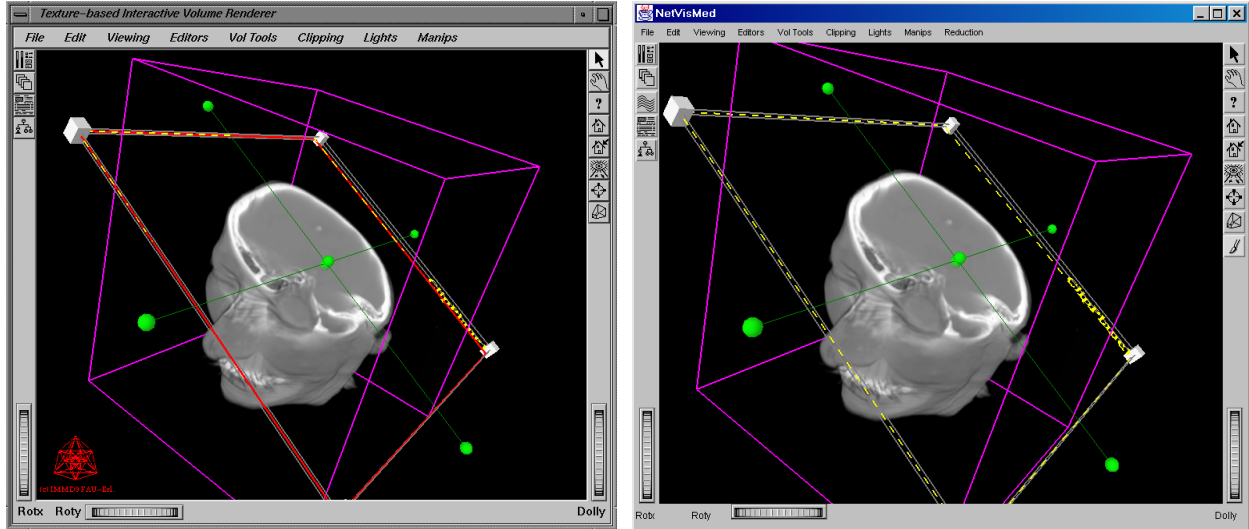


Abbildung 9.18: Transparenter Zugriff auf eine C++ Applikation, die 3D-Texturhardware eines Graphik-Hochleistungservers nutzt (links), durch eine Java-Applikation auf einem Client (rechts). Dabei bietet die Client-Applikation die gleiche Benutzeroberfläche wie die Server-Applikation.

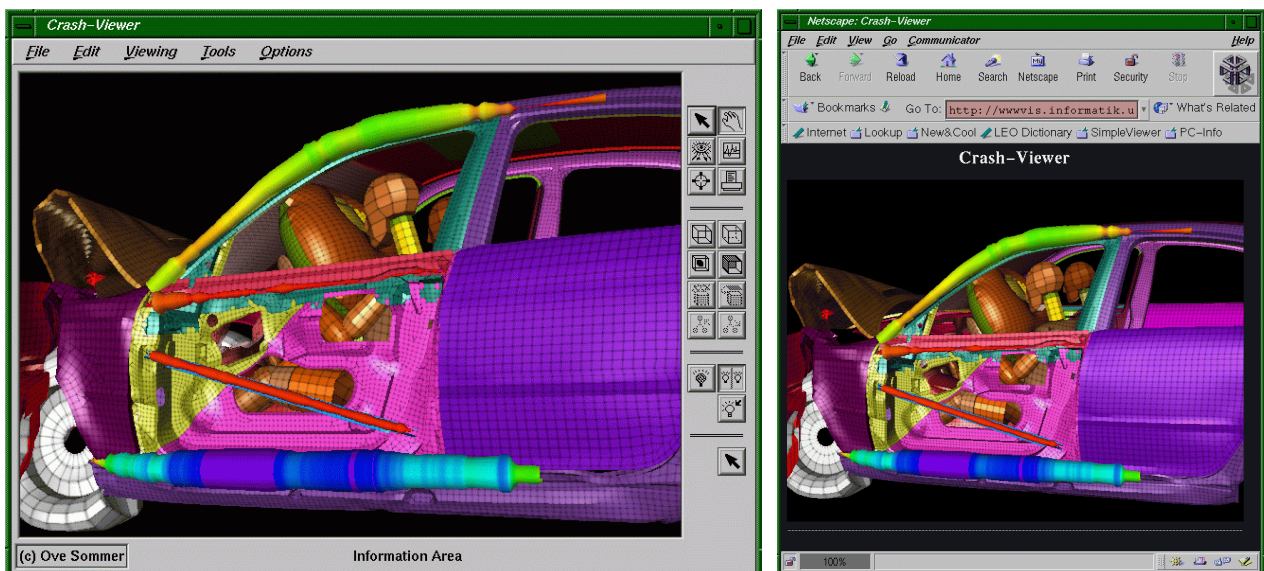


Abbildung 9.19: Transfer von Bilddaten einer Visualisierungsapplikation aus dem Fahrzeugentwicklungsprozess (links) via einer Socket-Verbindung zu einem Web-Browser (rechts).

Literaturverzeichnis

- [1] ISO/IEC 14772-1:1997. The Virtual Reality Modeling Language. <http://www.web3d.org/technicalinfo/specifications/vrml97/>, 1997.
- [2] ISO/IEC 14772-2:1997. External Authoring Interface. <http://www.vrml.org/WorkingGroups/vrml-eai/Specification/>, 1997.
- [3] K. Akeley. RealityEngine Graphics . In *Proceedings of SIGGRAPH*, Computer Graphics Conference Series, pages 109–116, 1993.
- [4] Ang, Martin, and Doyle. Integrated Control of Distributed Volume Visualization through the World Wide Web. In *Proceedings of IEEE Visualization*, pages 13–20. IEEE Computer Society Press, 1994.
- [5] A. Apodaca and L. Gritz. *Advanced RenderMan, Creating CGI for Motion Pictures*. Morgan Kaufmann, San Francisco, CA, 1999.
- [6] Ken Arnold and James Gosling. *The Java Programming Language*. Addison-Wesley, Reading, MA, 1998.
- [7] C.L. Bajaj, V. Pascucci, and D.R. Schikore. Fast Isocontouring for Improved Interactivity. In *IEEE Symposium on Volume Visualization*, pages 39–46, 1996.
- [8] Michael Bauer. Optimierung der Volumenvisualisierung auf PC-Hardware. Master's thesis, Graphische Datenverarbeitung IMMD9, Universität Erlangen, 2000.
- [9] M. Bender, A. Seck, and H. Hagen. A Client-side Approach towards Platform Independent Molecular Visualization over the World Wide Web. In *Proceedings of VisSym '99, Vienna*, 1999.
- [10] M. Bender, A. Seck, and H. Hagen. Using the Web as a Basis for an Efficient, Platform Independent Client-driven Molecular Visualization System. In *Proceedings of WWW8, Toronto*, 1999.
- [11] J. Blinn. Jim Blinn's Corner – Compositing, Part I: Theory. *IEEE Computer Graphics and Applications*, 14(5):83–87, 1994.

- [12] OpenGL Architecture Review Board. *OpenGL Reference Manual, Release 1*. Addison-Wesley, 1992.
- [13] B. Cabral, N. Cam, and J. Foran. Accelerated Volume Rendering and Tomographic Reconstruction Using Texture Mapping Hardware. *ACM Symp. on Vol. Vis.*, 1994.
- [14] Michael Bailey Cherylyn Michaels. VizWiz: A Java Applet for Interactive 3D Scientific Visualization on the Web. In *Proceedings of IEEE Visualization*, pages 261–267. IEEE Computer Society Press, 1997.
- [15] P. Cignoni, P. Marino, C. Montani, E. Puppo, and R. Scopigno. Speeding Up Isosurface Extraction Using Interval Trees. *IEEE Transactions on Visualization and Computer Graphics*, 3(2):158–170, 1997.
- [16] P. Cignoni, C. Montani, E. Puppo, and R. Scopigno. Optimal Isosurface Extraction from Irregular Volume Data. In *IEEE Symposium on Volume Visualization*, pages 31–39, 1996.
- [17] T. Clark, A. Alex, B. Beck, P. Chandrasekhar, P. Gedeck, A. Horn, M. Hutter, G. Martin, G. Rauhut, W. Sauer, T. Schnidler, and T. Steinke. Version VAMP 7.5A (Build 18). Erlangen, 2001.
- [18] J. Clucas. Interactive Visualization of Computational Fluid Dynamics Using Mosaic. In *Proceedings Second International WWW Conference*, 1994.
- [19] W.O. Cochran, J.C. Hart, and P.J. Flynn. Fractal Volume Compression. *IEEE Transactions on Visualization and Computer Graphics*, 2(4):313–322, 1996.
- [20] D. E. Comer and D. L. Stevens. *Internetworking with TCP/IP, Volume III: Client-Server Programming and Applications, BSD Socket Version*. Prentice Hall, Englewood Cliffs, NJ, 1993.
- [21] Web3D Consortium. The X3D Specification.
<http://www.web3d.org/TaskGroups/x3d/specification-2001november/main.html>, 2001.
- [22] M. de Boer, A. Gröpl, J. Hesser, and R. Männer. Reducing Artifacts in Volume Rendering by Higher Order Integration. In *Late Breaking Hot Topics, Proceedings of IEEE Visualization*. IEEE Computer Society Press, 1997.
- [23] M. Deering. Geometry Compression. In *Proceedings of SIGGRAPH*, Computer Graphics Conference Series, pages 13–20, 1995.
- [24] Steve Deering. Host Extensions for IP Multicasting. *RFC 1112, Network Working Group*, August, 1989.
- [25] Alligator Descartes. Magician Programmer’s Guide for Java.
<http://www.arcana.co.uk/products/magician>, 1998.

- [26] M.J.S. Dewar, E. G. Zoebisch, E.F. Healy, and J. J. P. Stewart. AM1: A New General Purpose Quantum Mechanical Molecular Model. *J. Am. Chem. Soc.*, 107, 3902, 1985.
- [27] Sébastien Dominé. Texture Shaders. <http://www.nvidia.com/Developer>.
- [28] B. Drebin, L. Carpenter, and P. Hanrahan. Volume Rendering. In *Proceedings of SIGGRAPH*, Computer Graphics Conference Series, pages 65–74, 1988.
- [29] G. Eckel. *OpenGL Volumizer Programmer's Guide*. SGI Developer Bookshelf, 1998.
- [30] N.H. El-Khalili and K.W. Brodlie. Architectural Design Issues for Web-based Virtual Reality Training Systems. In *Proceedings of 1998 International Conference on Web-based Modelling and Simulation*, pages 153–158. Society for Computer Simulation, 1998.
- [31] A. Endres and D. W. Fellner. *Digitale Bibliotheken - Informatik-Lösungen für die globalen Wissensmärkte*. dpunkt-verlag, Heidelberg, 2000.
- [32] K. Engel and T. Ertl. Texture-based Volume Visualization for Multiple Users on the World Wide Web. In Gervautz, M. and Hildebrand, A. and Schmalstieg, D., editor, *Virtual Environments '99*, pages 115–124. Eurographics, Springer, 1999.
- [33] K. Engel, R. Grosso, and Th. Ertl. Progressive Iso-surfaces on the Web. In *Proceedings of IEEE Visualization*. IEEE Computer Society Press, 1998.
- [34] K. Engel, P. Hastreiter, B. Tomandl, K. Eberhardt, and T. Ertl. Combining Local and Remote Visualization Techniques for Interactive Volume Rendering in Medical Applications. In *Proceedings of IEEE Visualization*, pages 449–452, 2000.
- [35] K. Engel, M. Kraus, and T. Ertl. High-Quality Pre-Integrated Volume Rendering Using Hardware-Accelerated Pixel Shading. In *Eurographics / SIGGRAPH Workshop on Graphics Hardware '01*, Annual Conference Series, page 9. Addison-Wesley Publishing Company, Inc., 2001.
- [36] K. Engel and F. Oellien. OrbVis – Orbital-Visualisierungs-Service. <http://www2.chemie.uni-erlangen.de/services/orbvis/>, 1999.
- [37] K. Engel, F. Oellien, W.D. Ihlenfeldt, and T. Ertl. Client-Server-Strategien zur Visualisierung komplexer Struktureigenschaften in digitalen Dokumenten der Chemie. In *IT+TI 6/2000 Informationstechnik und Technische Informatik*, pages 17–23, 2000.
- [38] K. Engel, O. Sommer, C. Ernst, and T. Ertl. Remote 3D Visualization using Image-Streaming Techniques. In *Advances in Intelligent Computing and Multimedia Systems (ISIMADE '99)*, pages 91–96, 1999.
- [39] K. Engel, O. Sommer, and T. Ertl. A Framework for Interactive Hardware Accelerated Remote 3D-Visualization. In *Proceedings of EG/IEEE TCVG Symposium on Visualization VisSym '00*, pages 167–177, 291, May 2000.

- [40] K. Engel, R. Westermann, and T. Ertl. Isosurface Extraction Techniques for Web-based Volume Visualization. In *Proceedings of IEEE Visualization*, pages 139–146. IEEE Computer Society Press, 1999.
- [41] F. Evans, S. Skiens, and A. Varshney. Optimizing Triangle Strips for Fast Rendering. In *Proceedings of IEEE Visualization*, pages 319–326. IEEE Computer Society Press, 1996.
- [42] D. Ginsburg, E. Hart, and J. Mitchell. ATI_fragment_shader. OpenGL Extension Registry: http://oss.sgi.com/projects/ogl-sample/registry/ATI/fragment_shader.txt.
- [43] Sven Goethel. GL4Java. <http://www.jausoft.com/gl4java.html>, 2001.
- [44] Michael Gold. NV_texture_env_combine4. OpenGL Extension Registry: http://oss.sgi.com/projects/ogl-sample/registry/NV/texture_env_combine4.txt.
- [45] James Gosling and Henry McGilton. The Java Language Environment: A White Paper. Technical report, SUN Microsystems, 2550 Garcia Avenue, Mountain View, CA 94043, USA, 1996.
- [46] Open Scene Graph. <http://www.openscenegraph.org/>, 2002.
- [47] M. Gross, L. Lippert, and C. Kurmann. Compression Domain Volume Rendering for Distributed Environments. *Computers Graphics Forum (EUROGRAPHICS '97)*, 16(3):96–107, 1997.
- [48] R. Grosso and T. Ertl. Mesh Optimization and Multilevel Finite Element Approximations. In *Proceedings VisMath '97*, 1997.
- [49] R. Grosso, T. Ertl, and J. Aschoff. Efficient Data Structures for Volume Rendering of Wavelet-Compressed Data. In N.M. Thalmann and V. Skala, editors, *WSCG '96 - The Fourth International Conference in Central Europe on Computer Graphics and Visualization*, volume I, pages 103–112, University of West Bohemia, Plzen, 1996.
- [50] R. Grosso and Th. Ertl. Progressive Iso-Surface Extraction from Hierarchical 3D Meshes. In *Proceedings Eurographics*, Computer Graphics Forum, 1998.
- [51] R. Grosso and G. Greiner. Multilevel Sobolev Approximations and Adaptive Mesh Reduction. Technical Report 10, Universität Erlangen-Nürnberg, 1997.
- [52] R. Grosso, C. Lürig, and T. Ertl. The Multilevel Finite Element Method for Adaptive Mesh Optimization and Visualization of Volume Data. In *Proceedings of IEEE Visualization*. IEEE Computer Society Press, 1997.
- [53] Open Inventor Architecture Group. *Open Inventor C++ Reference Manual, Release 2*. Addison-Wesley, 1994.
- [54] R. Grzeszczuk, C. Henn, and R. Yagel. Advanced Geometric Techniques for Ray Casting Volumes. In SIGGRAPH 98 Course Nbr. 4, Orlando, FL, 1998., 1998.

- [55] M. Hadwiger, T. Theußl, H. Hauser, and E. Gröller. Hardware-Accelerated High-Quality Filtering on PC Graphics Hardware. In *in Proceedings of Vision, Modeling, and Visualization*, 2001.
- [56] E. Hart and J.L. Mitchell. Hardware Shading with EXT_vertex_shader and ATI_fragment_shader. http://www.ati.com/na/pages/resource_centre/dev_rel/devrel.html.
- [57] P. Hastreiter, H.K. Çakmak, and Th. Ertl. Intuitive and Interactive Manipulation of 3D Data Sets by Integrating Texture Mapping Based Volume Rendering into the OpenInventor Class Hierarchy. In K. Spitzer Th. Lehman, I. Scholl, editor, *Bildverarbeitung für die Medizin: Algorithmen, Systeme, Anwendungen*, pages 149–154. Inst. f. Med. Inf. u. Biom. d. RWTH, Aachen, Verlag der Augustinus Buchhandlung, 1996.
- [58] P. Hastreiter, C. Rezk-Salama, B. Tomandl, K. Eberhard, and T. Ertl. Interactive Direct Volume Rendering of the Inner Ear for the Planning of Neurosurgery. In *Proceedings Workshop Bildverarbeitung für die Medizin (BVM)*, pages 192–196. Springer, 1999.
- [59] O. Hendin and O. John. Medical volume rendering over the WWW using VRML and Java. In *Westwood J. et al. (eds) Medicine Meets Virtual Reality:6. IOS Press and Ohmsha, Amsterdam, 1998; 34-40.*, 1998.
- [60] W. Hibbard. VisAD: connecting people to computations and people to people. In *Computer Graphics*, 32(3):10–12, 1998., 1998.
- [61] W. Hibbard, J. Anderson, and B. Paul. A Java and world wideweb implementation of VisAD. In *AMS IIPS International Conference, (1997).*, 1997.
- [62] W. Hibbard and D. Santek. The VIS-5D System for Easy Interactive Visualization. In *Proceedings of IEEE Visualization*, pages 28–35. IEEE Computer Society Press, 1990.
- [63] DirectX Home. <http://www.microsoft.com/directx/default.asp>, 2002.
- [64] OpenSG Home. <http://www.opensg.org/>, 2002.
- [65] H. Hoppe. Progressive Meshes. In *Proceedings of SIGGRAPH*, Computer Graphics Conference Series, pages 99–108, 1996.
- [66] Hugues Hoppe. Progressive Meshes. In *Proceedings of SIGGRAPH*, Computer Graphics Conference Series, pages 99–108, 1996.
- [67] W. D. Ihlenfeldt, Y. Takahashi, and H. Abe. Data Flow Processing for Computational Chemistry Problems. In *Proceedings ECCO Nancy*, 1994.
- [68] W. D. Ihlenfeldt, Y. Takahashi, H. Abe, and S. Sasaki. Computation and Management of Chemical Properties in CACTVS: An extensible Networked Approach toward Modularity and Flexibility. In *J. Chem. Inf. Comp. Sci.*34, pages 109–116, 1994.

- [69] W.D. Ihlenfeldt, F. Oellien, and K. Engel. ComSpec3D. <http://www2.ccc.uni-erlangen.de/services/vrmlvib/>, 1999.
- [70] 3Dlabs Inc. OpenGL 2.0 Shading Language White Paper. Technical report, 3Dlabs Inc., 2001.
- [71] Silicon Graphics Inc. Cosmo3D™ Programmer's Guide. Silicon Graphics Inc., IRIS Insight Library, 1998. <http://techpubs.sgi.com/>.
- [72] Silicon Graphics Inc. OpenGL Optimizer™ Programmer's Guide: An Open API for Large-Model Visualization. Silicon Graphics Inc., IRIS Insight Library, 1998. <http://techpubs.sgi.com/>.
- [73] S. Jaffarth. *Wavelets: Mathematics and Applications*, chapter Wavelets and Nonlinear Analysis, pages 467–505. ACM, 1993.
- [74] Hans-Georg Pagendarm Jens Trapp. A Prototype for a WWW-based Visualization Service. In *Proceedings Eurographics '97*, pages 23–30, 1997.
- [75] N. John, N. Phillips, R. Vawda, and J. Perrin. A VRML Simulator for Ventricular Catheterisation. In *In Eurographics UK'99, Cambridge, UK, April 1999.*, 1999.
- [76] A. Kee. Visualization over WWW using Java. Master's thesis, University of Leeds, 1996.
- [77] Mark J. Kilgard. NVIDIA OpenGL Extension Specifications. <http://www.nvidia.com/Developer>.
- [78] Mark J. Kilgard. NV_register_combiners. OpenGL Extension Registry: http://oss.sgi.com/projects/ogl-sample/registry/NV/register_combiners.txt.
- [79] Mark J. Kilgard. NV_register_combiners2. OpenGL Extension Registry: http://oss.sgi.com/projects/ogl-sample/registry/NV/register_combiners2.txt.
- [80] Mark J. Kilgard. NV_texture_shader. OpenGL Extension Registry: http://oss.sgi.com/projects/ogl-sample/registry/NV/texture_shader.txt.
- [81] Mark J. Kilgard. NV_texture_shader2. OpenGL Extension Registry: http://oss.sgi.com/projects/ogl-sample/registry/NV/texture_shader2.txt.
- [82] J. Kniss, G. Kindlmann, and C. Hansen. Interactive Volume Rendering Using Multi-Dimensional Transfer Functions and Direct Manipulation Widgets. In *Proceedings of IEEE Visualization*. IEEE Computer Society Press, 2001.
- [83] Kevin Kreeger, Ingmar Bitter, Frank Dacheille, Baoquan Chen, and Arie Kaufman. Adaptive Perspective Ray Casting. In *IEEE Symposium on Volume Visualization*, pages 55–62, 1998.

- [84] W. Krüger. The Application of Transport Theory to the Visualization of 3D Scalar Data Fields. In A. Kaufman, editor, *Proceedings of IEEE Visualization*, pages 273–280. IEEE Computer Society Press, 1990.
- [85] P. Lacroute. Real-Time Volume Rendering on Shared Memory Multiprocessors Using the Shear-Warp Factorization. In *1995 Parallel Rendering Symposium*, pages 15–22. ACM SIGGRAPH, 1995.
- [86] P. Lacroute. Analysis of a Parallel Volume Rendering System Based on the Shear-Warp-Factorization. *IEEE Transactions on Visualization and Computer Graphics*, 2(3):218–231, 1996.
- [87] P. Lacroute and M. Levoy. Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transform. *Comp. Graphics*, 28(4), 1994.
- [88] Eric C. LaMar, Bernd Hamann, and Kenneth I. Joy. Multiresolution Techniques for Interactive Texture-based Volume Visualization. In David Ebert, Markus Gross, and Bernd Hamann, editors, *Proceedings of IEEE Visualization*, pages 355–362, San Francisco, 1999. IEEE Computer Society Press.
- [89] Y. Livnat and C. Hansen. View Dependent Isosurface Extraction. In *Proceedings of IEEE Visualization*, pages 175–181. IEEE Computer Society Press, 1998.
- [90] Y. Livnat, H.-W. Shen, and C.R. Johnson. A Near Optimal Isosurface Extraction Algorithm using Span Space. *IEEE Transactions on Visualization and Computer Graphics*, 2(1):73–84, 1996.
- [91] W.E. Lorensen and H.E. Cline. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. In *Proceedings of SIGGRAPH*, Computer Graphics Conference Series, pages 163–169, 1987.
- [92] T. Malzbender. Fourier Volume Rendering. *ACM Transactions on Graphics*, 12(3), 1993.
- [93] William R. Mark and Kekoa Proudfoot. Compiling to a VLIW Fragment Pipeline. In *Eurographics / SIGGRAPH Workshop on Graphics Hardware '01*, Annual Conference Series. Addison-Wesley Publishing Company, Inc., 2001.
- [94] William R. Mark, Svetoslav Tzvetkov, and Pat Hanrahan. A Real-Time Procedural Shading System for Programmable Graphics Hardware. In *Proceedings of SIGGRAPH*, Computer Graphics Conference Series, 2001.
- [95] N. Max. Optical Models For Direct Volume Rendering. *IEEE Transactions on Visualization and Computer Graphics*, pages 99–108, 1995.
- [96] M. Meissner, S. Guthe, and W. Strasser. Interactive lighting models and pre-integration for volume rendering on pc graphics accelerators. In *In Proceedings of Graphics Interface, Conference on Human Computer Interaction and Computer Graphics*, 2002.

- [97] M. Meißner, U. Hoffmann, and W. Straßer. Enabling Classification and Shading for 3D Texture Based Volume Rendering Using OpenGL and Extensions. In *Proceedings of IEEE Visualization*. IEEE Computer Society Press, 1999.
- [98] M. Meißner, U. Kanus, and W. Straßer. VIZARD II, A PCI-Card for Real-Time Volume Rendering. In *Proceedings Eurographics/Siggraph Workshop on Graphics Hardware*, pages 61–68. IEEE Comp. Soc. Press, 1998.
- [99] Tom Misley. Web-Accessed Visualization Expert System (WAVES). Technical report, NACSE at Portland State University, 1997.
- [100] John S. Montrym, Daniel R. Baum, David L. Dignam, and Christopher J. Migdal. InfiniteReality: A real-time graphics system. *Computer Graphics*, 31(Annual Conference Series):293–302, 1997.
- [101] T. Munzner, E. Hoffman, K. Claffy, and B. Fenner. Visualizing the Global Topology of the MBone, 1996.
- [102] Jackie Neider, Tom Davis, and Mason Woo. *OpenGL Programming Guide*. Addison-Wesley, Reading MA, 1993.
- [103] SGI Newsroom. SGI Brings Advanced Visualization to the Desktop with OpenGL Viz-server. <http://www.sgi.com/software/vizserver/>.
- [104] G. Nielson and B. Hamann. The Asymptotic Decider: Removing the Ambiguity in Marching Cubes. In G. Nielson and Rosenblum. L., editors, *Proceedings of IEEE Visualization*, pages 83–91. IEEE Computer Society Press, 1991.
- [105] Kevin L. Novins. Towards Accurate and Efficient Volume Rendering. Technical Report TR93-1395, Cornell University, 1993.
- [106] M.F.X.J. Oberhumer. LZO. <http://wildsau.idv.uni-linz.ac.at/mfx/lzo.html>.
- [107] F. Oellien, W.D. Ihlenfeldt, K. Engel, and T. T.Ertl. Chemische Visualisierung und Datenintegration im Internet. In *Neue Medien in Forschung und Lehre, Ein Workshop im Rahmen der 29. Jahrestagung der Gesellschaft für Informatik (Informatik 99)*. Gesellschaft für Informatik, 1999.
- [108] K.M. Oh and K.H. Park. A Type-Merging Algorithm for Extracting an Isosurface from Volumetric Data. *The Visual Computer*, 12:406–419, 1996.
- [109] M. Ohlberger and M. Rumpf. Hierarchical and Adaptive Visualization on Nested Grids. *Computing*, 59 (4):269–285, 1997.
- [110] J. Patten and K. Ma. A Graph Based Approach for Visualizing Volume Rendering Results. In *In Proceedings of Graphics Interface, Conference on Computer Graphics and Interactive Techniques*, 1998.

- [111] Yuh-Jye Chang Paul Coddington and Karlie Hutchens. The NPAC/OLDA Visible Human Viewer. <http://www.dhpc.adelaide.edu.au/projects/vishuman>.
- [112] H. Pfister and A. Kaufman. Cube-4 - A Scalable Architecture for Real-Time Volume Rendering. In R. Crawfis and C. Hansen, editors, *IEEE Symposium on Volume Visualization*, pages 47–54. ACM SIGGRAPH, 1996.
- [113] H. Pfister and T. Kaufmann, A. amd Chiueh. Cube-3: A Real-Time Architecture for High-Resolution Volume Visualization. In A. Kaufman and W. Krüger, editors, *IEEE Symposium on Volume Visualization*, pages 75–82. ACM SIGGRAPH, 1994.
- [114] Hanspeter Pfister, Jan Hardenbergh, Jim Knittel, Hugh Lauer, and Larry Seiler. The VolumePro Real-Time Ray-casting System. In Alyn Rockwood, editor, *Proceedings of SIGGRAPH*, Computer Graphics Conference Series, pages 251–260, Los Angeles, 1999. Addison Wesley Longman.
- [115] B. Poddar, M. Gold, FrisingerT., and R. Hammerstone. ARB_texture_env_combine. OpenGL Extension Registry: http://oss.sgi.com/projects/ogl-sample/registry/ARB/texture_env_combine.txt.
- [116] J.A. Pople and Beveridge D.L. *Approximate Molecular Orbital Theory*. McGraw-Hill Verlag, New York, 1970.
- [117] Thomas Porter and Tom Duff. Compositing Digital Images. In *Proceedings of SIGGRAPH*, volume 18/3 of *Computer Graphics Conference Series*, pages 253–259, 1984.
- [118] T. Poston, H.T. Nguyen, P.A. Heng, and T.T. Wong. Skeleton-Climbing:Fast Isosurfaces With Fewer Triangles. In *Pacific Graphics*, pages 117–126, 1997.
- [119] Tommy Reilly. Jogl. <http://copa.pajato.com/jogl/>, 1997.
- [120] C. Rezk-Salama, K. Engel, M. Bauer, G. Greiner, and T. Ertl. Interactive Volume Rendering on Standard PC Graphics Hardware Using Multi-Textures and Multi-Stage-Rasterization. In *Eurographics / SIGGRAPH Workshop on Graphics Hardware '00*, pages 109–118,147. Addison-Wesley Publishing Company, Inc., 2000.
- [121] T. Richardson, Q. Stafford-Fraser, K. R. Wood, and A. Hopper. Virtual Network Computing. *IEEE Internet Computing*, 2(1), January 1998.
- [122] G. Roelofs. ZLIB. <http://www.cdrom.com/pub/infozip/zlib/>.
- [123] J. Rossignac and P. Borrel. *Modeling in Computer Graphics*. Springer-Verlag, 1993.
- [124] S. Röttger., M. Kraus, and T. Ertl. Hardware-accelerated volume and isosurface rendering. In *Proceedings of IEEE Visualization*, pages 109–116. IEEE Computer Society Press, 2000.

- [125] W. Schroeder, K. Martin, B. Lorensen, and V. Toolkit. *The Visualization Toolkit, An Object-Oriented Approach To 3D Graphics*. Prentice Hall, 1996.
- [126] W. Schroeder, J. Zarge, and W. Lorensen. Decimation of Triangular Meshes. In *Proceedings of SIGGRAPH*, Computer Graphics Conference Series, pages 65–70, 1992.
- [127] W. Schroeder, J. A. Zarge, and W. E. Lorensen. Decimation of Triangle Meshes. In *Proceedings of SIGGRAPH*, Computer Graphics Conference Series, pages 65–70, 1992.
- [128] SGI. Volume Rendering using RE2 Technology. Technical report, SGI, 1994.
- [129] R. Shekhar, E. Fayyad, R. Yagel, and J. Cornhill. Octree-based Decimation of Marching Cubes Surfaces. In *Proceedings of IEEE Visualization*, pages 335–342. IEEE Computer Society Press, 1996.
- [130] H. Shen and C. Johnson. Sweeping Simplices: A Fast Iso-Surface Extraction Algorithm for Unstructured Grids. In *Proceedings of IEEE Visualization*, pages 143–150. IEEE Computer Society Press, 1995.
- [131] H.-W. Shen, C. Hansen, Y. Livnat, and C. R. Johnson. Isosurfacing in Span Space with Utmost Efficiency (ISSUE). In *Proceedings of IEEE Visualization*, pages 287–294. IEEE Computer Society Press, 1996.
- [132] J. Siegal. *CORBA: Fundamentals and Programming*. Wiley, 1996.
- [133] O. Sommer, A. Dietz, R. Westermann, and T. Ertl. An Interactive Visualization and Navigation Tool for Medical Volume Data. In V. Skala, editor, *Proceedings 6th International Conference in Central Europe on Computer Graphics and Visualization '98*, pages 362–371, 1998.
- [134] Ove Sommer and Thomas Ertl. Geometry and rendering optimizations for the interactive visualization of crash-worthiness simulations. In *Proceedings of SPIE, Visual Data Exploration and Analysis VII*, volume 3960, January 2000.
- [135] H. Sowizral, D. Nadeau, M. Bailey, and M. Deering. Introduction to Programming with Java3D. ACM SIGGRAPH 98 Course Notes, July, 1998., 1998.
- [136] J. Spitzer. GeForce 256 and RIVA TNT Combiners. <http://www.nvidia.com/Developer>.
- [137] S. Stegmaier, M. Magallon, and T. Ertl. A Generic Solution for Hardware-Accelerated Remote Visualization. In *IEEE TCVG Symposium on Visualization*, 2002.
- [138] H. Theisel, U. Rauschenbach, C. Perez Risquet, and H. Schumann. Vector Field Visualization on the Internet. In *The 11th Conference on Simulation and Visualization, Sim-Vis'2000, in Proceedings*, Magdeburg, Germany, 2000. Otto-von-Guericke-University of Magdeburg.

- [139] U. Tiede, T. Schiemann, and K.-H. Höhne. High Quality Rendering of Attributed Volume Data. In *Proceedings of IEEE Visualization*. IEEE Computer Society Press, 1998.
- [140] Takashi Totsuka and Marc Levoy. Frequency Domain Volume Rendering. *Computer Graphics*, 27(Annual Conference Series):271–278, 1993.
- [141] L. Treinish. Enablement of a Fluid Client-Server Visualization Environment. In *Workshop on PC-Based Visualization and Computer Graphics, IEEE Visualization*, 1997.
- [142] J. Wernecke. *The Inventor Toolmaker, Release 2*. Addison–Wesley, 1994.
- [143] J. Wernecke. *The Inventor Mentor, Programming Object-Oriented 3D Graphics with OpenInventor*. Addison-Wesley, release 2 edition, 1994.
- [144] R. Westermann. A Multiresolution Framework for Volume Rendering. In A. Kaufman and W. Krüger, editors, *IEEE Symposium on Volume Visualization*, pages 51–58. ACM SIGGRAPH, 1994.
- [145] R. Westermann and T. Ertl. A Multiscale Approach to Integrated Volume Segmentation and Rendering. *Computers Graphics Forum (EUROGRAPHICS '97)*, 16(3):96–107, 1997.
- [146] R. Westermann and T. Ertl. Efficiently Using Graphics Hardware in Volume Rendering Applications. In *Proceedings of SIGGRAPH*, Computer Graphics Conference Series, 1998.
- [147] R. Westermann, L. Kobbelt, and T. Ertl. Real-time Exploration of Regular Volume Data by Adaptive Reconstruction of Iso-Surfaces. *The Visual Computer*, 1999.
- [148] L. Westover. Footprint Evaluation for Volume Rendering. *Computer Graphics*, 24(4), 1990.
- [149] J. Wilhelms and A. Van Gelder. A Coherent Projection Approach For Direct Volume Rendering. *Computer Graphics*, 25(4):275–284, 1991.
- [150] J. Wilhelms and A. Van Gelder. Octrees for Faster Iso-surface Generation. In *ACM Transactions on Graphics*, pages 201–227, 1992.
- [151] P. Williams, N. Max, and C. Stein. A High Accuracy Volume Renderer For Unstructured Data. *IEEE Transactions on Visualization and Computer Graphics*, 4(1):37–54, 1998.
- [152] C. Wittenbrink, T. Malzbender, and M. Goss. Opacity-Weighted Color Interpolation For Volume Visualization. In *IEEE Symposium on Volume Visualization*, pages 135–142, 1998.
- [153] M. Woo, J. Neider, and T. Davis. *OpenGL Programming Guide, Version 1.1*. Addison–Wesley, 2 edition, 1997.

- [154] J. Wood, K. Brodlie, and H. Wright. Visualization over the World Wide Web and its Application to Environmental Data. In *Proceedings of IEEE Visualization*, pages 81–86. IEEE Computer Society Press, 1996.
- [155] Tatu Ylonen. SSH – Secure Login Connections Over the Internet. In *Proceedings of the Sixth USENIX Security Symposium*, pages 37–42, 1996.

Lebenslauf

Name:

Klaus Engel

Geburtsort, -datum:

Nördlingen, 13. Oktober 1969

Familienstand:

ledig

Schulbildung:

1976-1980: Grundschule Löpsingen

1980-1989: Theodor-Heuss Gymnasium Nördlingen

1989: Abitur, Theodor-Heuss Gymnasium Nördlingen

Studium:

1990-1993: Grundstudium am Institut für Informatik der Universität Würzburg

1993: Vordiplom vom Institut für Informatik der Universität Würzburg

1993-1997: Hauptstudium am Institut für Informatik der Universität Erlangen

1997: Dipl.-Inf.(Univ.) vom Institut für Informatik der Universität Erlangen

Beruf:

1997: Informatiker bei der 3SOFT GmbH, Erlangen

Forschung:

1998-1999: Wissenschaftlicher Mitarbeiter am Lehrstuhl für Graphische Datenverarbeitung der Universität Erlangen

2000-2002: Wissenschaftlicher Mitarbeiter in der Abteilung für Visualisierung und Interaktive Systeme (VIS) der Universität Stuttgart