Prototypische Realisierung eines 3D Modelling Systems nach dem Sculpturing Ansatz.

Diplomarbeit

zur Erlangung des akademischen Grades
Diplom-Informatiker

an der
Fachhochschule für Technik und Wirtschaft Berlin
Fachbereich Wirtschaftswissenschaften VI
Studiengang Angewandte Informatik

1. Betreuer: Prof. Dr.-Ing. Thomas Jung
2. Betreuer: Prof. Dr. Hermann Hessling

Eingereicht von Frank Otto
Berlin, 10. November 2006

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Vorwort

Es war immer ein Traum von mir, Informatik zu studieren. Träume können Realität werden. Im Folgenden möchte ich mich für die Unterstützung innerhalb des Studiums und bei der Erstellung dieser Arbeit Bedanken.
Mein besonderer Dank gilt Prof. Dr. Thomas Jung, der sich stets Zeit für mich genommen hat und mir wertvolle Ratschläge gab. Danken möchte ich meinem Vater, bei dem ich mentale und finanzielle Unterstützung fand. Weiterhin danke ich meiner Freundin Frauke, welche sich während der Erstellung dieser Arbeit aufopferungsvoll um mich gekümmert hat.

1 Einleitung

1.1 Motivation

Der Grundgedanke zu dieser Arbeit wurde durch die Praktische Projekt Arbeit "Immersive Visualisierung medizinischer Daten" der FHTW (2005/2006) angeregt. Da medizinische Daten in Schicht-Bildern vorliegen, musste das Projekt sich mit Volumendaten auseinandersetzen. Warum nicht ein System erstellen in dem der Benutzer ein Volumen editiert bzw. modelliert, wobei modellieren das entfernen von Volumenelementen bedeutet. Da der Benutzer in erster Linie nur Teile des Ganzen entfernen soll, gibt ihm das Verhalten des Systems ein Gefühl von Formgebung im Sinne des erstellen einer Skulptur (Sculpturing). Das Gebiet der Formgebung von dreidimensionalen Objekten innerhalb der 3D-Computergrafik ist nach wie vor Teil der Forschung. Das Ziel dieser Forschung liegt im Auffinden immer effizienterer Methoden zur Erstellung von Modellen. Dabei liegt das Augenmerk im besonderen im Bereich der Benutzerschnittstelle.

1.2 Aufgabenstellung und Ziele

Die Aufgabe eines 3D Modelling Systems liegt Grundsätzlich in der Erstellung eines dreidimensionalen Modells durch den Benutzer. In den letzten Jahren wurde immer deutlicher dass die 3D-Computergrafik neben Ihrer Nutzung für Wissensschaft und Unterhaltungsbranche vermehrt als Form der Kunst angesehen wird. Gerade in der Entertainment Industrie wird in diesem Zusammenhang oft von 3D-Artists (Künstler) gesprochen. Ziel dieser Arbeit ist die prototypische Realisierung eines dreidimensionalen Modelling Systems nach dem Sculpturing Ansatz. Sculpturing bedeutet in diesem Zusammenhang das bearbeiten eines "festen" Ausgangsmaterials. Durch Wegnahme von Elementen soll die gewünschte Form aus einem Würfel herausgearbeitet werden können. Ein anschließender Export in ein gängiges 3D Dateiformat soll als Schnittstelle zu Kommerziellen Lösungen dienen. Die Bedienung wird über die Standard Eingabe Geräte Maus und Tastatur erfolgen. Diese System soll einfachen und intuitiv Zugang zur Erstellung von 3D Objekten bieten.

1.3 Aufbau der Arbeit

Das Kapitel Grundlagen beschäftigt sich mit den theoretischen Grundlagen, welche zum weiteren Verständnis dieser Arbeit notwendig sind. Zunächst werden die beiden verwendeten Formen der Beschreibung von 3D-Objekten vorgestellt. Danach wird die Idee der Partitionierung des Raumes erläutert. Der dritte Abschnitt der Grundlagen beschäftigt sich mit der Mensch-Maschine Schnittstelle. Dazu wird zuerst die klassische Polygonmodellierung erörtert. Abschließend wird ein Überblick über den Stand der Wissenschaft im Bereich der Volumenmodellierung gegeben.

Im Kapitel Analyse findet eine Bewertung von theoretischen Ansätzen und Verfahren statt. Es werden zunächst Voxel- gegenüber Polygonaldarstellungen bewertet. Nachfolgend wird eine Entscheidung über die zu verwendende Grafik API getroffen. Abschließend wird mittels eines von den Standardverfahren der Visualisierung von Voxeldaten abweichendes Verfahren evaluiert.

Die Anforderungen an ein 3D-Modelling System nach dem Sculpturing Ansatz werden im Kapitel Anforderungsdefinition wiedergegeben. Dabei werden funktionelle und nicht funktionelle Anforderungen unterschieden.

Das Kapitel Design beschäftigt sich im Wesentlichen mit dem Aufbau des Gesamtsystems. Es werden Lösungsansätze, ebenso wie zentrale Entwurfsentscheidungen getroffen. Ein besonderes Augenmerk liegt auf der Modellierung der Benutzerschnittstelle. Dazu gehört die Interaktion mit dem Sculpturing-Modell, genauso wie die Gestaltung der grafischen Benutzer Schnittstelle.

Eine Erläuterung der Vorgehensweise während der Realisierung bildet den Anfang des Kapitels Implementierung. Nachfolgend wird die Architektur der Anwendung beschrieben, um abschließend die Feinheiten während der Realisierung näher zu betrachten.

Tests, welche das Gesamtsystem betreffen, wurden in diesem Kapitel durchgeführt. Dazu gehören die Modellierung und die Ergebnisse der Triangulation.

Im letzen Kapitel werden zunächst alle Ergebnisse zusammengefasst. Anschließend werden die Ergebnisse mit der Zielstellung verglichen. Danach erfolgt eine Bewertung der Ergebnisse mit nachfolgender Problembeschreibung. Als Letzes wird noch ein Ausblick auf Möglichkeiten der Erweiterung des Prototypen, sowie die anderen Einsatzgebiete der verwendeten Technologie gegeben.

2 Grundlagen

2.1 Sculpturing Ansatz

Mit Sculpturing[1] wird im Allgemeinen die Formgebung eines Werkstoffes von Hand auf Ton oder mit einem Werkzeug wie z.B. einem Meissel auf Stein bezeichnet. Der Begriff wird im Bereich der bildenden Künste verwendet. In der dreidimensionalen Computergrafik werden unter dem Sculpturing Ansatz die Anwendungen und Strategien zusammengefasst, welche sich *direkt* mit der Formgebung von dreidimensionalen Objekten beschäftigen. Dieser Zusammenhang von direkter Formgebung impliziert eine Interaktion eines Benutzers mit einem Sculpturing System.

2.2 Beschreibung von dreidimensionalen Objekten

In der dreidimensionalen Computergrafik zählen neben bikubischen parametrischen Teilflächen, CSG und impliziten Darstellungen die Polygonnetze und Volumenelemente (Voxel) zu den gebräuchlichen Modellen. Zum besseren Verständnis ist es notwendig, genauer auf die Polygonnetze sowie die Volumenelemente einzugehen. Dabei wird der Aufbau und auch die Darstellung näher betrachtet.

2.2.1 Polygon Netze

Die bekannteste Erscheinungsform aus der Gruppe der dreidimensionalen Objekte stellen die Polygonnetze dar. Die beschriebenen "Objekte werden durch ein Netz planarer Polygon-Flächen angenähert" (Watt 2002, S. 45 2. Absatz). polygonale Netze erlauben es, das Objekt mit beliebiger, nicht stetiger Genauigkeit abzubilden. Da die einzelnen Polygonflächen planar sind, ist die Darstellung gekrümmter Objekte stets eine Annäherung. Dieses Objektmodell ist eine Oberflächenrepräsentation, d.h. es wird die Oberfläche, nicht aber das Volumen des Objektes modelliert. Die

[1] Anfänglich wurde der Begriff Sculpting geprägt.

polygonale Darstellung von dreidimensionalen Objekten ist die klassische Art der Abbildung. Die einzelnen Vertices werden mit Hilfe einer Indexliste zu einem Objekt zusammengefasst. Zusätzlich können weitere Informationen wie Texturkoordinaten, Vertexfarbe oder die so genannten Normalen gespeichert werden. Bei den Normalen handelt es sich um einen Vektor, welcher orthogonal zur jeweiligen polygonalen Fläche steht und zur Lichtberechnung verwendet wird. Polygonnetze sind prinzipiell eine Maschinendarstellung, die häufig als Benutzerdarstellung verwendet wird.

Hinsichtlich des Sculpturing wird in modernen 3D-Modellierungssystemen das Objekt in seine Bestandteile (Unterobjekte) zergliedert. Diese Unterbobjekte bilden die Basis der Manipulation aller Teile des Objektes. Die Tabelle 2.1 erläutert den Aufbau eines dreidimensionalen Objektes und seiner Unterobjekte am Beispiel von "editable Poly" Objekten (*Autodesk´s 3D-Studio Max7*).

Tabelle 2.1: Logischer Aufbau des 3D-Studio Max editable Polyobjekt

Element	**Bedeutung**
Vertex	Der Punkt im kartesischen Koordinatensystem.
Edge	Eine Kante als Verbindung zwischen zwei Punkten.
Border	Die Grenze des Polygonobjektes als Selektion von Kanten.
Polygon	Die polygonale Fläche kann mehr als drei Punkte umfassen.
Element	Elemente können Mengen aller anderen Unterobjekte darstellen.

Im dem Kapitel 2.4.1 auf Seite 10 werden gängige Operationen während des Modellierens eines Polygonobjektes erläutert.

Rendern von Polygonnetzen

Die Erzeugung eines digitalen Bildes - der Meta Beschreibung eines oder mehrerer Objekte - wird im Bereich der 3D-Computergrafik als *"Rendern"* bezeichnet. Computergestützte Grafiken werden im Allgemeinen in der Form einer Rastergrafik auf einem Monitor ausgegeben. Diese Rastergrafiken bestehen aus einer Matrix von Farbwerten. Die einzelnen Elemente werden als Pixel (Picture Element) bezeichnet.

Für die Erstellung des Prototypen sind in erster Linie Techniken zur Echtzeitdarstellung von Interesse. Deshalb wird im Folgenden die Leistungfähigkeit im Bezug auf das Polygonrendering näher betrachtet.

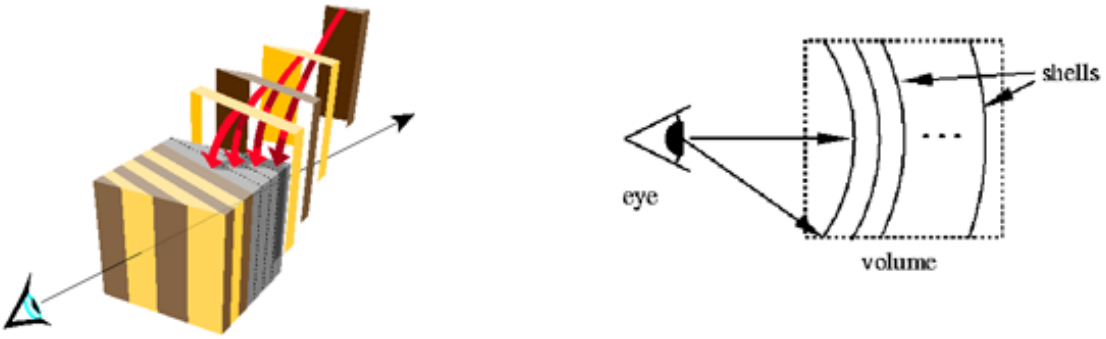
Die Wiedergabe von Polygonnetzen ist die in Hardware "gegossene" Standardoperation von modernen 3D-Beschleunigern. Die Leistungsfähigkeit ist einer stetigen Entwicklung unterworfen, dabei ist sowohl der Polygondurchsatz als auch die Füllrate gemessen in *Texel* (Textur Element) von Interesse. Der Polygondurchsatz bezeichnet die Anzahl von Dreiecken, welche pro Sekunde transformiert werden können, während die Füllrate die Geschwindigkeit der Rasterisierung der Dreiecke bezeichnet. In einigen Fällen wird die Füllrate in Abhängigkeit der verwendenten Technologie angeben. Dabei wird zwischen den Hardwaretextureinheiten und den so genannten Pixelshadern unterschieden. Durch die parallele Architektur der Vertex- sowie Pixel-Shaderpipeline wurden die Füllraten immens gesteigert.

2.2.2 Das volumetrische Element (Voxel)

Der Begriff Voxel wird in der 3D-Computergrafik verwendet und setzt sich aus den Wörtern "Volumen" und "Element" zusammen. Ein Voxel ist die dreidimensionale Analogie zu einem Pixel. Elementare Würfel kennzeichnen die Raumbelegung des zu beschreibenden Objektes. Objekte werden gemeinhin mittels eines Voxelfeldes in kartesischen Koordinaten dargestellt. In den 90er Jahren fand diese Form der Objektdarstellung in der Praxis wenig Verwendung, da der Speicherbedarf dieser Objektform relativ hoch ist. W. Lorenson und H. Cline stellten auf der SIGGRAPH 1987 den Marching Cubes Algorithmus vor, welcher eine Triangulation eines Voxelmodelles ermöglicht (Lorenson u. Cline 1987, Paper abstract). Ihr Verfahren überführt eine Volumendarstellung in eine polygonale Oberflächenrepräsentation. 1996 wurde ein Verfahren zur Reduzierung der vom Marching Cubes Algorithmus erzeugten Flächen vorgestellt. Dieses Verfahren basiert auf der Verwendung eines Oktonärbaums und reduziert die Anzahl erzeugter Polygone erheblich (Shekhar u. a. 1996, Paper abstract). Heutzutage werden Voxelrepräsentationen u.a. innerhalb von industriellen oder medizinischen Anwendungen wie der Computer-Tomographie (CT) oder Magnet-Resonanz-Tomographie (MRT) verwendet. Volumenrendering-Systeme vereinfachen dabei eine Analyse der gewonnenen Daten. Es existieren unterschiedliche Verfahren anhand derer Volumendaten in Echtzeit visualisiert werden können. Zum Einen verwendet man polygonale Schnittflächen, ein anderer Ansatz besteht im Ray-Casting.

Echtzeitvisualisierung mittels Schnittflächen

Moderne Desktop 3D-Beschleuniger verfügen über so genannte *dreidimensionale Texturen*, welche die Speicherung eines Voxelfeldes in eine Textur ermöglichen. Die meisten Systeme verfügen über >128 MB an Videospeicher. Bei einer Datensatzgröße von 256^3 Voxel und einem RGBA (4 Byte) Farbwert pro Voxel werden rund 64 MB an Texturspeicher benötigt. Eine Darstellung bestehend aus 512^3 Elementen (512 MB) würde die Mehrheit der verfügbaren Desktopsysteme überfordern. Ein

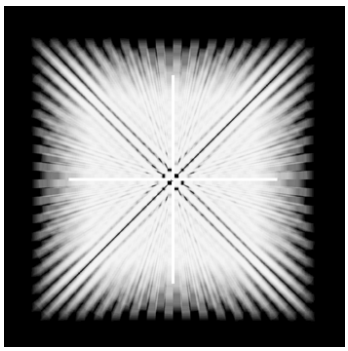


(a) Zur Darstellung des Volumen wird eine 3D-Textur geschnitten

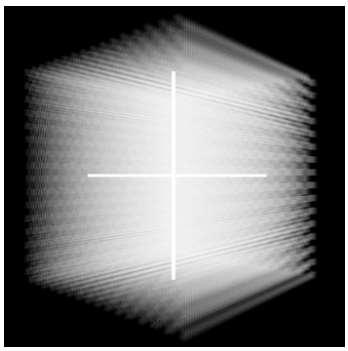
(b) Um ein Aliasing der Voxel zu verhindern werden Kugelabschnitte gezeichnet

Abbildung 2.1: Volumenrendering mittels 3D-Texturen und Schnittflächen(Quelle: OpenGL 1997, Webresource)

sehr effizienter Weg um eine dreidimensionale Textur zu visualisieren wurde von der Opengl Group auf der SIGGRAPH 1997 vorgestellt (OpenGL 1997, Webresource). Das Verfahren schneidet das Sichtvolumen in Blickrichtung mit einzelnen Flächen (vgl. Abbildung 2.1a). Mittels generierter Texturkoordinaten wird die Textur auf den Flächen positioniert. Die Flächen werden miteinander verblendet. Dabei wird als Quell-Bildelement der Bildschirmspeicher und als Ziel-Bildelement der Alphawert des jeweiligen Voxel verwendet. Damit diese Form einer Alpha-Blendoperation korrekt arbeitet, müssen die Flächen in sortierter Reihenfolge gezeichnet werden(von hinten nach vorne). Das Ergebnis gleicht einem halb-transparenten Gel. Um ein Aliasing der Voxel an den Rändern des Bildes zu verhindern, werden Kugelabschnitte anstatt planarer Flächen vorgeschlagen (vgl. Abbildung 2.1b). Damals schnitten die Flächen nur die Blickrichtung, heute verwenden die meisten Implementierungen Flächen, welche den Raum in drei Achsen schneiden. Hierbei wird in der Regel maximal eine Fläche pro Voxelebene genutzt. Daraus resultiert eine sehr exakte Wiedergabe der Texturdaten (vgl. Abbildung 2.2 auf der nächsten Seite).



(a) Testdatensatz bestehend aus einer dreidimesionalen "Karo" Textur



(b) Seitenansicht der Testdaten

(c) DICOM Datensatz eines Kopfes bei Anwendung eines Schwellwertes

Abbildung 2.2: Volumenrendering mittels Schnittflächen (Quelle: 2005/2006 FHTW Fachbereich 4 Semesterprojekt Immersive Visualisierung medizinischer Daten: Volumenrenderer Prototyp)

Echtzeitvisualisierung mittels Ray-Casting

Bei diesem Verfahren wird der Volumendatensatz in die gewünschte Ansichtsrichtung gedreht und mit einer Menge von parallelen Strahlen geschnitten. Von der Bildebene ausgehend wird ein Strahl pro Bildelement in den Raum geworfen. Trifft ein Strahl auf einen Voxel, wird die Farbe/Intensität als Pixel verwendet. Dabei kann ein Pixel aus mehreren "getroffenen" Voxeln berechnet werden. Der Unterschied zum Ray-Tracing Verfahren besteht in der Tatsache, dass die Strahlen ihren Weg durch das Objekt fortsetzten, statt sich beim Auftreffen zu trennen. In der Praxis existieren zahlreiche Ansätze dieses Konzept umzusetzen (Watt 2002, S. 417). Das Verfahren ist relativ aufwendig in der Berechnung. Man stelle sich eine Bildebene mit 800*600 Bildpunkten vor. Bei einer Transformation der gesamten Voxelmenge müssten 480.000 Strahlen unter Echtzeitbedingungen gesendet werden.

Wie J. A. Bærentzen gezeigt hat, besteht die Möglichkeit einer inkrementellen Aktualisierung des Bildinhaltes (Bærentzen 1998, Paper abstract). Dies begründet sich auf der Annahme dass während der Gestaltung eines Objektes meist nur Teile des Modelles verändert werden.

2.3 Partitionierung des Raumes

Dieser Abschnitt der Grundlagen beschäftigt sich mit den Verfahren zur strukturierten Raumaufteilung. Ein großer Teil moderner 3D-Anwendungen verwendet eine

gewisse Form der Raumaufteilung. In diesem Zusammenhang wird auch von Partitionierung des Raumes gesprochen. Diese Form der Strukturierung findet oft bei der Optimierung der Sichtbarkeit bzw. Verringerung von Zugriffen Verwendung.

2.3.1 Quaternärbaum (Quadtree)

Mit Hilfe von Quaternärbäumen lassen sich zweidimensionale Bereiche partitionieren. Ein Knoten des Baums hat vier Kinder, welche den Knoten im Normalfall in vier gleiche Teile partitionieren (vgl. Abbildung **??** auf Seite ??). In der dreidimensionalen Computergrafik wird ein Quaternärbaum oft eingesetzt um z.B. Landschaften oder Räume innerhalb von Spielen zu partitionieren. Dabei ist in jedem Knoten eine Liste von Zeigern auf die sich in dem Knoten befindenden Polygone gespeichert. Der Vorteil dieser Herangehensweise besteht in dem schnellen Auffinden bestimmter Bereiche, ohne dabei immer das gesamte Objekt zu durchsuchen. Es wird rekursiv eine Intersektion bis zu den Blättern des Baum berechnet und dann den entsprechenden Zeigern gefolgt, um an die gewünschten Polygone zu kommen. Da eine Landschaft im Prinzip nur eine zweidimensionale Fläche darstellt, eignet sich ein Quaternärbaum hervorragend zur Partitionierung von 3D-Landschaften.

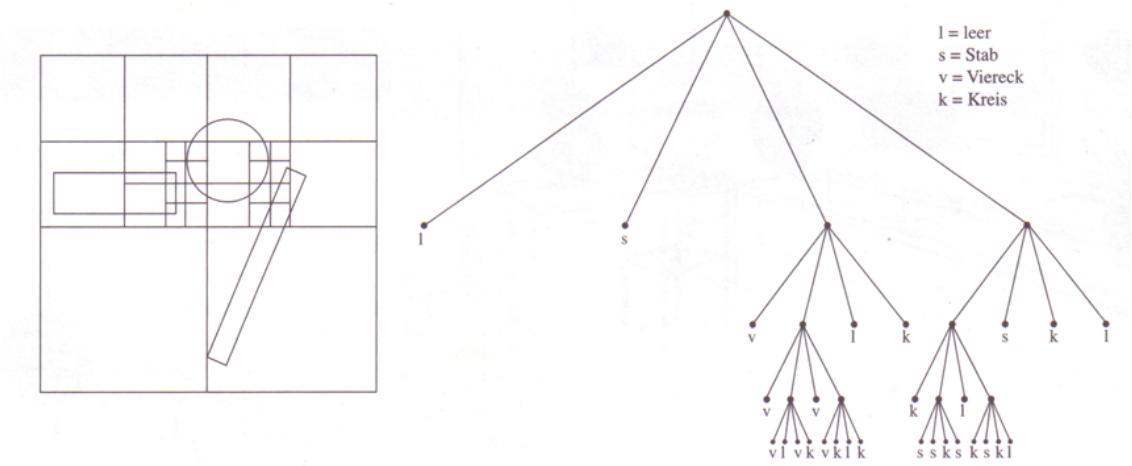


Abbildung 2.3: Schematische Darstellung eines Quaternärbaum mit Kennzeichnung der Raumbelegung (Quelle: Watt 2002, S.69 Abbildung 2.16)

2.3.2 Oktonärbaum (Octree)

Ein Oktonärbaum ist die dreidimensionale Analogie zum Quaternärbaum. Er wird zur Partitionierung des Raumes insbesondere beim so genannten Ray-Tracing ein-

gesetzt. Der Baum erzeugt eine Struktur räumlicher Abhängigkeiten (vgl. Abbildung 2.4). Wobei jeder Knoten seine Position, die Ausdehnung im Raum und die Adressen seiner Kindknoten, sowie Nachbarknoten speichert. Diese Struktur wird häufig zur Organisation einer dreidimensionalen Szene, welche aus vielen Objekten besteht eingesetzt. Dabei kann ein Oktonärbaum zur Lösung des Problemes der Sichbarkeit von Objekten (Occlusion Culling) verwendet werden. In der Regel sind bei dieser Anwendungsform die Adressen der einzelnen Polygone in den Blättern des Baums gespeichert. Dazu wird ein Schwellwert verwendet, welcher angibt wie viele Polygone maximal in einem Blatt enthalten sein sollen. Ein Weg zur Verdeckung nicht sichtbarer Knoten führt dann über die Berechnung der Intersektion des Baums mit dem Sichtvolumen. Die Knoten, welche vom Sichtvolumen geschnitten werden sind sichtbar. C. Saona-Vazquez, I. Navazo und P. Brunet stellten 1999 ein Verfahren zur Vorberechnung eines Oktonärbaums in ihrer Arbeit *The visibility octree: A data structure for 3d navigation* vor (Saona-Vazquez u. a. 1999, Paper abstract). Dieses Verfahren findet gegenwärtig fast ausnahmslos in dreidimensionalen Spielen Verwendung.

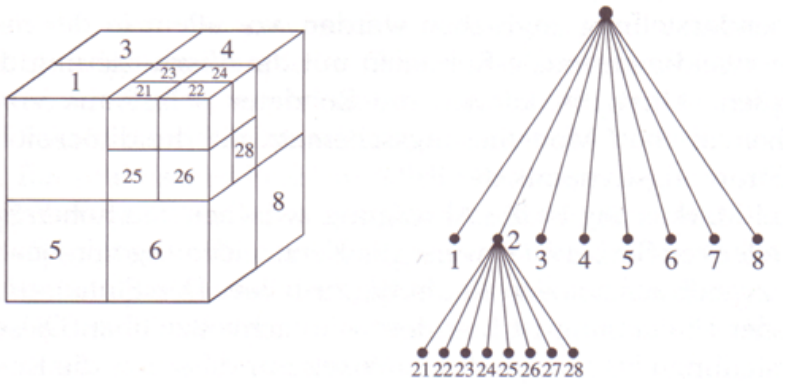


Abbildung 2.4: Schematische Darstellung eines Oktonärbaum mit Kennzeichnung der Knoten (Quelle: Watt 2002, S.68 Abbildung 2.15)

2.4 Mensch-Maschine Schnittstelle

Die Modellierung von dreidimensionalen Objekten setzt ein hohes Maß an räumlicher Vorstellungskraft voraus. Systeme zur Volumenmodellierung sind nach wie vor Teil der wissenschaftlichen Forschung im Bereich der 3D-Computergrafik. Im Folgenden werden zuerst die klassische Polygonmodellierung und ihre Eigenarten näher erläutert. Dies ist notwendig, um den Grad der Abstraktion, welche vom Designer

verlangt wird, zu verstehen. Anschließend werden die Meilensteine der Volumenmodellierung vorgestellt. Dabei wird deutlich, dass diese Systeme alle etwas gemeinsam haben. Die Metapher der Modellierung ist stets eine Form des Hinzufügens bzw. Entfernens von Teilen eines Werkstückes.

2.4.1 Klassische Polygonmodellierung

Neben dem Digitalisieren von Objekten werden Objekte in der Regel manuell modelliert. Dabei kommen Werkzeuge wie *3D Studio Max*, *Maya* oder die freie Software *Blender*, aber auch CAD/CAM Systeme wie *AutoCAD 4D* zum Einsatz. Die Ausgangsbasis des Modellierungsprozesses bilden Primärobjekte wie Dreiecke, Quader, Kugeln oder Flächen.

Abgesehen vom Erstellen der Primärobjekte und ihrer Untermengen verfügen diese Systeme über eine Reihe von Funktionen zur weiteren Bearbeitung des Netzobjektes. Diese lassen sich in zwei Teilebereiche gliedern. Einesteils Funktionen, welche auf das gesamte Objekt, anderenteils Funktionen, welche in erster Linie auf Teilmengen des Objektes angewendet werden. Die letzte Gruppe bildet die Grundlage der Polygonmodellierung.

Polygon Modellierung am Beispiel von 3ds Max

Neben der Transformationen und dem Hinzufügen bzw. Löschen von Vertices gibt es eine Reihe von Funktionen, mit denen sich die Topologie des Gitternetzobjektes verändern lässt. Einige Grundfunktionalitäten zur Manipulation werden im Folgenden anhand des *editable Poly Object*, der Anwendung 3D Studio Max Version 7.0 (3ds Max) kurz erläutert.

Extrudieren (engl. *extrude*) Das Extrudieren von Untermengen des Polygonnetzes wird innerhalb der Abbildung 2.5a verdeutlicht. Diese Funktion ist auf alle Unterobjekte des Poly-Objektes anwendbar.

abgeflachte Kante (engl. *bevel*) Die Funktionalität des schrägen Extrudieren wird als Bevel(abgeflachte Kante) bezeichnet. Dabei wird die extrudierte Grundfläche skaliert (vgl. Abbildung 2.5b).

Abschrägung (engl. *chamfer*) Mittels der Abschrägung lassen sich im Nachhinein Fasen an einer Kante oder einem Punkt einfügen (vgl. Abbildung 2.5c).

Einfügung (engl. inset) *Inset* ermöglicht das Einfügen von Polygonkanten. Dabei wird ein skaliertes Abbild der Originalkante des ausgewählten Polygons in das Polygon verschoben. Die Originalkante bleibt dabei erhalten (vgl. Abbildung 2.5d).

Brücke (engl. *bridge*) Diese Funktion erlaubt das einfache Verbinden zweier Polygone. Es können geschlossenen Kanten sowie Polygonselektionen mit einer Brücke aus Polygonen versehen werden. Diese Funktion wurde mit Version 7 von 3ds Max eingeführt (vgl. Abbildung 2.6).

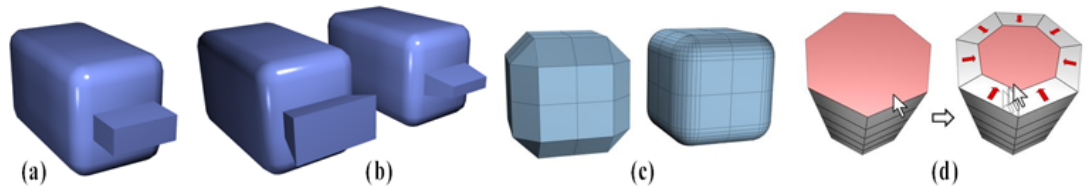


Abbildung 2.5: Basisfunktionen der Polygonmodellierung in *3ds Max editable Poly Objekt* (a) extrude,(b) bevel,(c) chamfer,(d) inset (Quelle: Autodesk 2004b, Seite 1-30)

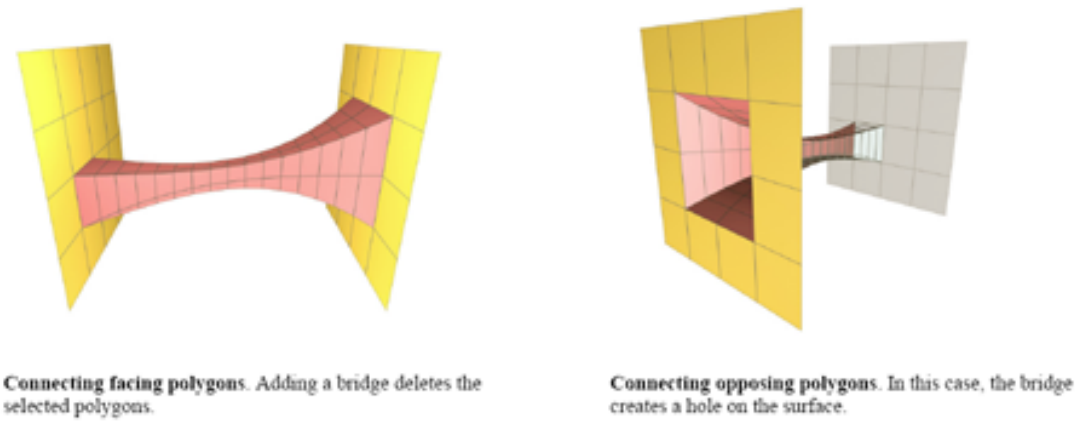


Abbildung 2.6: Eine Brücke zwischen Polygonen ist außerhalb und innerhalb eines Objektes anwendbar. (Quelle: Autodesk 2004a, Seite 27)

Gemalte Deformation (function: *Paint Deformation*) Der Name dieser Funktionalität impliziert, dass es sich bei dieser Funktion um eine gemalte Verformung handelt. Unter zur Hilfenahme eines skalierbaren Pinsels (Grösse, Verlauf etc.) kann an dem Gitterobjekt gezogen oder gedrückt werden. Dabei hat der Benutzer die Wahl, die Kraft entlang der ursprünglichen Normalen oder der bereits verformten

Normalen wirken zu lassen. Der Pinsel gleitet auf der Oberfläche des Objektes entlang, so dass sehr intuitiv gearbeitet werden kann.

Da die reine Deformation zu kantigen Objekten führt, wurde an dieser Stelle noch ein Gitterentspannugsalgorithmus integriert. Diese *Relax* genannte Funktion lässt sich ebenfalls per Pinsel anwenden. Alle Deformationen können "malend" rückgängig gemacht werden. Zur Erzeugung einer Grundform ist dieses Werkzeug eher ungeeignet. Seine Stärken liegen im Verfeinern der Details eines Objektes (vgl. Abbildung 2.7). Da die zu deformierende Oberfläche nicht weiter tesseliert wird, hat diese Form der Modellierung ihre Grenzen.

Diese relativ neuen Funktionen, wurden erst mit Erscheinen der Version 7 eingeführt. Das hat nicht zuletzt mit der signifikanten Steigerung der Leistungsfähigkeit heutiger Computersysteme zu tun. Paint Deformation wird gerne eingesetzt, um so genannte Normalmaps zu erzeugen. Bei dieser Technik wird aus einem hochaufgelösten Gitterobjekt eine Textur berechnet, die die Normalvektoren des Objektes in die RGB-Kanäle der Textur kodiert. Normalmaps werden häufig in der Echtzeit 3D-Computergrafik zur Licht- und Tiefensimulation verwandt. Der Vorteil der Verwendung einer Normalmap liegt darin, dem Betrachter den Eindruck zu vermitteln, ein hochgradig detailliertes Objektes präsentiert zu bekommen.

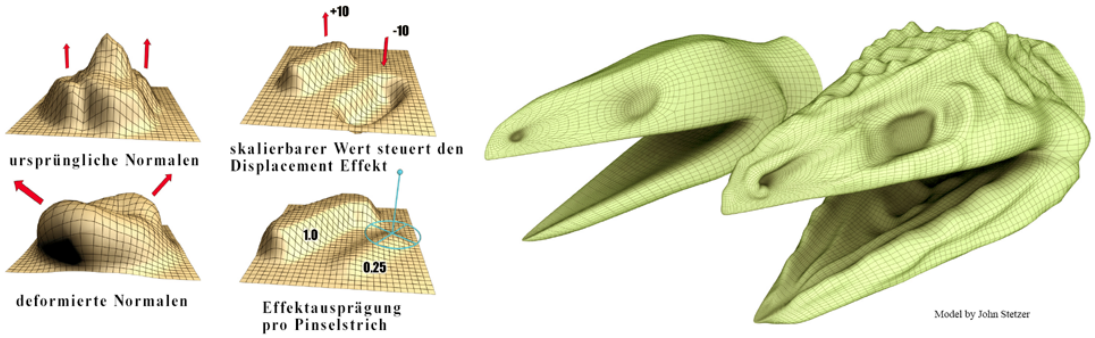


Abbildung 2.7: 3ds Max erweiterte Basisfunktion *Gemalte Deformation* (Quelle: Autodesk 2004a, Seite 30-31)

Modifikatoren-Paradigma am Beispiel 3ds Max

Neben den Grundfunktionen zur Manipulation von Objekten finden in 3D Studio Max auch die Modifikatoren Verwendung. Diese verfolgen einen anderen Ansatz. Modifikatoren werden ebenfalls bei Teilbereichen bzw. ganze Objekte nutzbar gemacht und ermöglichen diverse Funktionalitäten. Ein sehr einfaches Beispiel ist der *noise* Modifikator. Dieser ermöglicht es ein Rauschen (engl. noise) auf eine bestehende

Netz-Topologie anzuwenden. Dabei wird eine Translation mittels skalierbarer Zufallswerte auf die Punktmenge des Objektes ausgeführt. 3D Studio Max organisiert die Modifikatoren in einem Stapel, welcher von unten nach oben abgearbeitet wird. Dieser Modifikator-Stapel erlaubt es dabei die einzelnen Modifikatoren miteinander zu kombinieren. *Subdivision-Surface Algorithmen* werden im Allgemeinen auf diese Weise angewendet. Da der Modifikator nicht das Objekt verändert sondern nur die Sicht auf dieses, lassen sich im Zusammenspiel von Skalierung des Modifikators und Veränderung des eigentlichen Objektes sehr schnell komplexe Modelle entwerfen.

Beispiel: Modellieren einer Tischdecke durch physikalische Simulation und anschließende Verfeinerung (vgl. Abbildung 2.8). Dabei kommen folgenden Schritte zur Anwendung:

1. Erstellen des Polygon-Netzes für die physikalische Simulation, die Unterteilung sollte so gewählt werden das eine Simulation in Echtzeit noch möglich ist (z.B. $20 * 20 * 2 Dreiecke = 800$ Dreiecke).
2. Den Modifikator "Kleidung" auf den Stapel des Modells anwenden.
3. Werteanpassungen entsprechend dem gewünschten Verhalten bezüglich Dichte, Faltung, Kollision des Stoffes vornehmen.
4. Die Simulation starten und z.B. die Tischdecke auf einen Tisch fallen lassen.
5. Um nun das Ergebnismodell entscheidend aufzuwerten einen *Subdivision-Surface* Modifikator z.B *Meshsmooth* dem Modifikator Stapel hinzufügen und über seine Parameter die Genauigkeit des Ausgabeobjektes steuern.

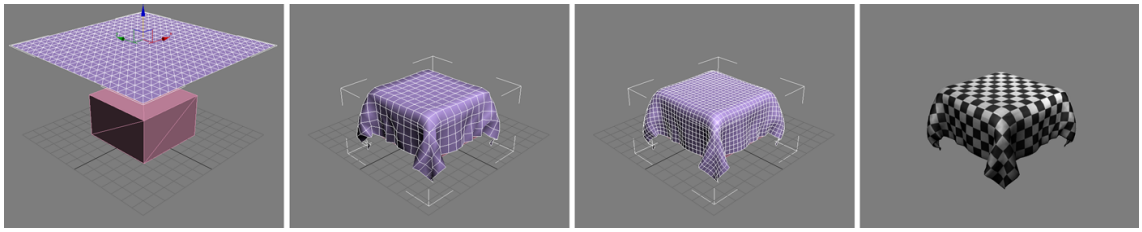


Abbildung 2.8: Modifikator Paradigma in 3ds Max anhand der Modellierung einer Tischdecke (v.l.) Ausgangsmodell, nach Simulation, Meshsmooth 1. Iteration, finales Rendering

In dieser Beispielszene ist es möglich, die physikalische Simulation zu verändern und dabei lediglich ein niedrig aufgelöstes Gitterobjekt zu simulieren und als Ausgabe ein sehr genaues Objekt zu erhalten.

Modifikatoren finden oft Verwendung, um einem gegebenen Objekt zu einer bestimmten Form bzw. einem bestimmten Verhalten zu verhelfen. Dabei verfügt 3D-Studio Max über eine sehr große Ansammlung von Modifikatoren. Einige sind auf

die Verarbeitung von Partikelsystemen und andere auf den Einsatz mit Splines spezialisiert. Viele von Ihnen jedoch lassen sich auf nahezu jeden Objekttyp anwenden z.B. die Texturkoordinaten-Generierung (UVW Mapping). Häufig verwandt wird auch das Verbiegen, Torsieren oder Rotationsobjekte. Zu den höher entwickelten Modifikatoren gehören Haut sowie Haare/Fell oder Kleidung.

Low-Poly-Modellierung

Low-Poly-Modellierung ist eine besondere Ausprägung der Polygonmodellierung. Ihr Ziel ist es, ein Model mit möglichst wenigen Polygonen zu erzeugen. Low-Poly-Modellierung wurde in der Vergangenheit häufig genutzt um Modelle für 3D-Spiele zu erzeugen, da die Leistungsfähigkeit der damaligen Systeme sehr eingeschränkt war. Heute findet sie ihre Hauptaufgabe in der *Subdivision-Surface-Modellierung.*

Subdivision-Surface Modellierung

Innerhalb dieser Form der Modellierung werden *"Subdivision-Surface-Algorithmen"* (Olmos 2004, S. 92) zur iterativen Verfeinerung der Topologie des Basisnetzes eingesetzt. Dabei wird ein sehr einfaches Polygonobjekt herangezogen um ein hoch aufgelöstes Objekt zu erzeugen bzw. zu manipulieren. Die einzelnen Vertices des niedrig aufgelösten Objektes fungieren als Kontrollpunkte für einen Verfeinerungsalgorithmus.

Mit der Unterteilungsflächen-Modellierung wurde eine einfache Methode zur Erstellung komplexer und zugleich glatter, biomorpher Formen geschaffen. Subdivision-Surface Modellierung bedarf einigen Wissens um die Zusammenhänge. Ein gezieltes Sculpturing ist überhaupt nur möglich, wenn Erfahrung im Bereich der klassischen geometrischen Polygon-Modellierung vorhanden ist. Die Abbildung 2.9 auf der nächsten Seite verdeutlicht die Leitungsfähigkeit dieses Verfahrens.

Zu den bekanntesten Vertretern von Subdivision-Surface-Algorithmen gehören die Kantenhalbierung, das Doo-Sabin Schema (Doo u. Sabin 1978, Paper abstract) und das Catmull-Clark Schema (Catmull u. Clark 1978, Paper abstract). Die Grundideen der Unterteilung sind in den 40er Jahren entstanden. G. Rahm verwendete die Kantenhalbierung zum Beschreiben von glatten Kurven (de Rahm 1956, Paper abstract). Das Doo-Sabin sowie das Catmull-Clark Schema arbeiten auf der Basis von Polygonen mit 4 Seiten bzw. 4 Vertices pro Masche, so genannten Viereckssnetzen (*Quads*). Die Verwendung mit Dreiecksnetzen führt zu einem ungewünschten Ergebnis. Das Doo-Sabin Schema erzeugt dreidimensionale Gitterobjekte, welche

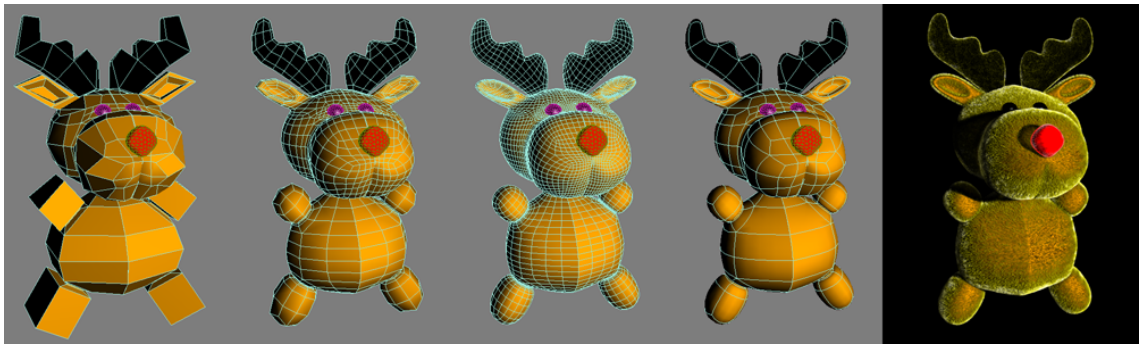


Abbildung 2.9: Subdivision-Surface-Modellierung (v.l.) ohne Subdivision, 1.Iter.; 2.Iter.; 2.Iter. mit Kontrollgitter Isolininen; finales Rendering

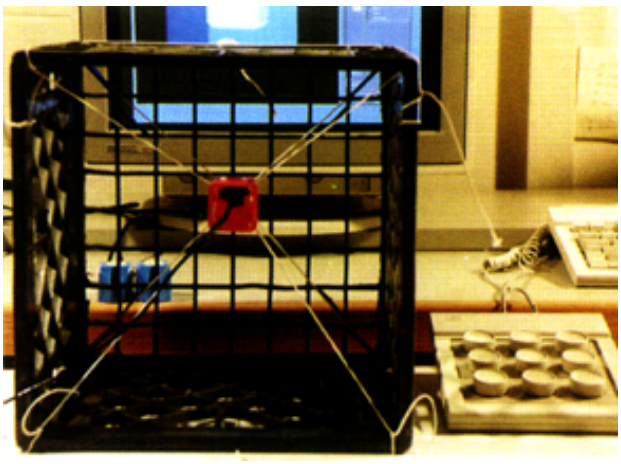
gegen bi-quadratische B-Spline Flächen konvergieren. Das Objekt nähert sich abgerundet der Form des Ausgansnetzes an. Während dessen konvergiert das Catmull-Clark Schema gegen bi-kubische B-Spline Flächen. Diese Objekte integrieren sich annähernd spannungsfrei in das Ausgangsnetz.

2.4.2 Volumenmodellierung

Im Jahre 1991 nutzten J.F. Hughes, A. Galyean und A. Tinsley das erste Mal eine Voxeldarstellung um ein *Sculpting* System vorzustellen (An interactive volumetric modeling technique). Voxeldaten wurden mittels eines dreidimensionalen Feldes gespeichert. Dieses System verfügte über ein dreidimensionales Eingabegerät und verwendete den *Marching Cubes Algorithmus* zur Überführung der Volumendaten in eine polygonale Darstellung. Es wurden neben dem Abschneiden und Hinzufügen von Volumen zwei Werkzeuge vorgestellt; zum Einen Schmirgelpapier zum Anderen eine Art Hitzekanone. Dabei werden die Materialeigenschaften von Ton beziehungsweise Wachs nachgebildet (Galyean u. Hughes 1991, Paper abstract). Das System verfügt sogar über eine primitive Kraftrückmeldung (vgl. Abbildung 2.10).

Sidney Wang and Arie E.Kaufman stellten 1995 mit der Arbeit *Volume sculpting* ein Modellierungssystem vor, welches ebenfalls auf der Basis eines Voxelfelds arbeitet. Im Unterschied zu der Arbeit von Hughes, Galyean und Tinsley wurden die Voxeldaten mittels des Ray-Casting Verfahrens visualisiert. Dabei wird nur der veränderte Bereich der Darstellung aktualisiert. Zusätzlich speichert das System für jeden Voxel Materialeigenschaften wie Farbe und Textur. Auch dieses System benötigt ein dreidimensionales Eingabegerät. Die Werkzeuge basieren auf Formen, die durch Voxel repräsentiert werden. Diese Herangehensweise erlaubt es, mit einfachen auf Voxelmengen basierenden Operationen das Objekt zu verändern. Das Verhalten

(a) Teekanne vor und nach dem Bearbeiten mit dem "Schmirgelpapier" Werkzeug



(b) poor man´s force feedback unit

Abbildung 2.10: Sculpting: An interactive Volumentric Modeling Technique (Quelle: Galyean u. Hughes 1991, S.269-270)

der Werkzeuge ähnelt dem Schnitzen respektive dem Sägen eines Objektes (Wang u. Kaufman 1995a, Paper abstract).

Octree based Volume Sculpting

Octree based Volume Sculpting wurde 1998 von Jakob Andreas Bærentzen an der Technischen Universität von Dänemark entwickelt. Ziel der Arbeit war eine Zusammmenführung von Werkzeugeigenschaften der beiden vorangegangenen Arbeiten (Hughes, Galyean und Tinsley bzw. Wang und Kaufman). Als Datenstruktur zur Speicherung der Voxel findet ein Oktonärbaum Verwendung. Grundsätzlich werden zwei Arten der Formgebung unterschieden. Einerseits das Hinzufügen bzw. Entfernen von vordefinierten Formen nach der CSG[2] Metapher, andererseits die "Sprühdosen-Metapher". Letztgenannte erlaubt das Auftragen und Entfernen von Voxeln mittels einer Art Sprühdose. Die Volumendaten werden mit Hilfe des Ray-Casting Verfahrens visualisiert. Um den Aufwand dieses Verfahrens zu reduzieren, werden nur die Knoten des Oktonärbaums, welche "veränderte" Voxel enthalten, neu gezeichnet. Als Ausblick diskutiert Bærentzen die Möglichkeit der Erweiterung zum Multiresolution-Sculpting. Dabei sollen Voxel mit unterschiedlicher Ausdehnung zum Einsatz kommen (Bærentzen 1998, Paper abstract).

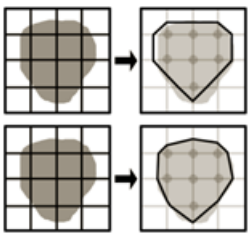
[2]Constructive Solid Geometry verwendet implizite Beschreibungen von Grundkörpern oder elementaren Formen, die mit Hilfe von boolescher Algebra kombiniert werden können.

Abbildung 2.11: Octree based Volume Sculpting, l. eine Troll-ähnliche Kreatur, r. ein Kopf mittels Ray-Tracing visualisiert (Quelle: Bærentzen 1998, Abbildung 2 u. 3)

Bei meiner Recherche bin ich auf eine Reihe anderer Arbeiten gestoßen, welche ähnlich Ansätze verfolgen, dabei jedoch vermehrt eine Form von dreidimensionalen Eingabe- und Ausgabegeräten verwenden. Im folgenden werde ich die Unterschiede zu dem von Bærentzen entwickelten System erläutern.

A Real-time 3D Virtual Sculpting Tool Based on Modified Marching Cubes
Dieses Verfahren wurde 2001 von K.Perng, W.Wang, M.Flanagan und M.Ouhyoung an der Universität von Oregon entwickelt. Es nutzt im Gegensatz zu Bærentzen einen modifizierten Marching Cubes Algorithmus zur Visualisierung eines Polygonmodells. Die Modifikation führt zu einer genaueren Annäherung der Triangulation an die Form des Voxelmodells und verhindert ein Aliasing am Schnittpunkt zweier Kanten (vgl. Abbildung 2.12a). Die Benutzerschnittstelle bildet ein 3D-Eingabererät (3D-Tacker). Als Ausgabegerät wird ein *Head Mounted Display* (HMD) oder eine *Vision Station*[3] vorgeschlagen. Damit ist der Gestaltungsprozess immersiv wahrnehmbar, lediglich eine Kraftrückmeldung existiert nicht. Wie in Bærentzens System kann Material aufgetragen bzw. entfernt werden. Zusätzlich kann die polygonale Darstellung mittels Vertex Kolorierung bemalt werden. Dieses System wurde intensiven Speicheroptimierungen unterzogen, um die Echtzeitfähigkeit zu ermöglichen. Auch diese Anwendung nutzt einen Oktonärbaum zur Reduzierung des Datenaufkommens. Der eigentliche Bereich in dem das Sculpting stattfindet, umfasst $16cm^3$. Zusätzlich verfügt das System über die Möglichkeit, das Objekt rotieren zu lassen, welches ein schnelles Prototyping von Modellen fördert (vgl. 2.12 auf der nächsten Seite). Ein Pentium III Prozessor mit 500 Mhz und eine Geforce2 MX entspricht den Mindestanforderungen der Systemleistung (Perng u. a. 2001, Paper abstract).

[3]Dabei handelt es sich um einen Kugelabschnitt auf dessen Innenseite die Ausgabe projiziert wird.



(a) Oben: Ursprünglicher Marching Cubes Algorithmus. Unten: Ergebnis nach der Modifikation.

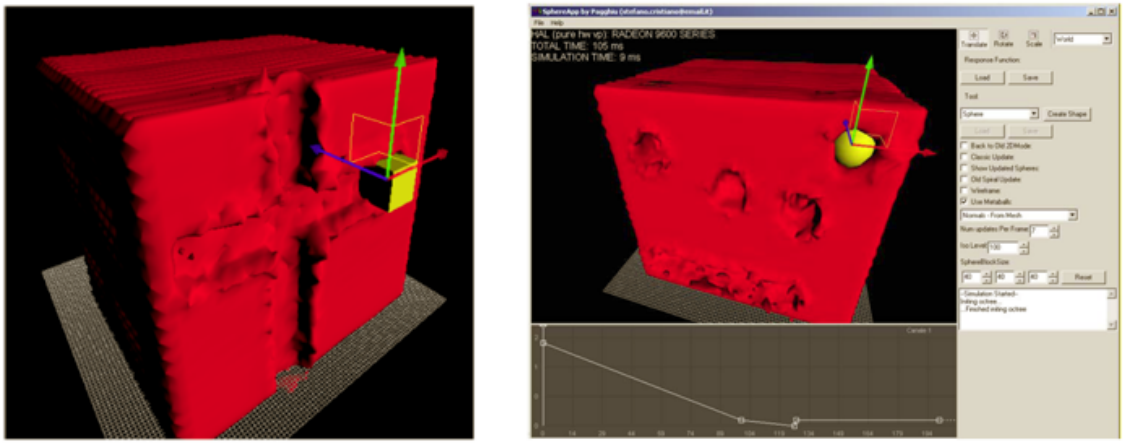
(b) Das Werkstück kann während der Formgebung gedreht werden.

(c) "An apple pierced ba an arrow" wurde von Hagihara Tadanori in weniger als 15 Minuten erstellt

Abbildung 2.12: A Real-time 3D Virtual Sculpting Tool Based on Modified Marching Cubes (Quelle: Perng u. a. 2001, Abbildung 3,6,10)

REAL-TIME PARTICLE BASED VIRTUAL SCULPTURING Eine Forschungsgruppe der Politecnico di Bari (Italien) veröffentlichte 2004 dieses System, dass das Manipulieren eines verformbaren Materials nachbildet (Cristiano u. a. 2004, Paper abstract). Im Gegensatz zu Bærentzen simuliert dieses System physikalisches Modellieren basierend auf einem Partikelsystem. Als Partikel wurden Kugeln mit einem festen ganzzahligen Radius eingesetzt. Durch diese Vorgehensweise war es lediglich nötig den Mittelpunkt einer Kugel zu speichern. Dieses ist prinzipiell eine Voxeldarstellung. Dem eigentlichen Modellieren liegt ein Kollisionstest der Partikel mit dem Sculpting Werkzeug zugrunde. Die Partikel werden entsprechend ihrer Kollision mit dem Werkzeug verschoben. Dabei wird ebenso die Kollision der Partikel untereinander berücksichtigt. Die Partikel, zusammen mit ihrem impliziten Modell einer Kugel, werden in eine Voxeldartellung überführt. Eine optimierte, echtzeitfähige Implementierung des Marching Cubes Algorithmus extrahiert aus dieser Darstellung ein Polygonmodell. Anders als Bærentzen verwendet dieses System einen *Loose Octree* (Deloura 2002, S. 434) zur Speicherung der Partikel. Diese Ausprägung eines Oktonärbaums macht sich überlappende Grenzen zur Beschreibung der einzelnen Knoten zu Nutze. Da die Partikel als Kugeln abgebildet werden, wird somit erreicht, dass einzelne Partikel lediglich zu einem Knoten des Baums gehören. Der Prototyp ist mittels der DirectX 9 API realisiert worden und nutzt Vertexshader-Programme zur Beschleunigung der Anwendung. Wie Bærentzens Arbeit setzt dieses System keine Form eines dreidimensionalen Eingabegerätes voraus. Das Verhalten des Systems kommt den Materialeigenschaften von Ton sehr nahe (vgl. Abbildung 2.13 auf der nächsten Seite).

***SensAble´s Technologies* Claytools** SensAble´s Claytools basiert auf der bereits 2001 von K.T. McDonnell, H. Qin und A. Wlodarczyk vorgestellten Arbeit



(a) Würfel als Werkzeug (b) Kugel als Werkzeug

Abbildung 2.13: REAL-TIME PARTICLE BASED VIRTUAL SCULPTURING (Quelle: Cristiano u. a. 2004, S.1 ,S.7)

Virtual Clay: A Real-time Sculpting System with Haptic Toolkits. Dabei wurden verschiedenste Ansätze der Modellierung innerhalb eines Framework zusammengefasst. Neben dem Volumenmodell werden Oberflächenrepräsentationen unterstützt. Hierbei finden so genannte *subdivision-solid* Verwendung. Diese repräsentieren sowohl die Oberfläche als auch das Innere eines Objektes (McDonnell u. a. 2001, Paper abstract). Im folgenden wird die kommerzielle Ausprägung namens Claytools näher betrachtet.

Dieses kommerzielle Modellierungssystem besteht aus einer Hardware-Software Kombination. Claytools bildet die Softwareebene. Das dreidimensionale Objekt,"eine Art virtueller Ton, besteht aus einer Voxelwolke" (Friebe u. Kuhn 2006, S. 62 2.Absatz). Die Grösse der Voxel, welche die Wolke bilden ist skalierbar. Aus ihr resultiert die Körnung des virtuellen Ton. Zusätzlich werden NURBS und Polygonmodelle unterstützt. Eine Kombination der einzelnen Darstellungen ist möglich. Das *Phantom Omni* als haptisches Gerät kann seinerseits mit Hilfe von Kraftrückmeldung verschiedene Widerstände gegen das Eindringen in die Oberfläche diese Objektes simulieren. Das erlebte Verhalten ähnelt dabei dem Eindringen eines Lötkolbens in Styropor. Neben dem Eindringen sind diverse Funktionen implementiert, u.a Maskieren, Ziehen, Drücken sowie das Verschmieren der Oberfläche. Es stehen Importschnittstellen für die bekanntesten 3D-Fileformate zur Verfügung. Mit diesem System bildet *Sensable* der Zeit die Königsklasse der Modellierwerkzeuge, was sich nicht zuletzt durch den Preis äußert. Das Einstiegsmodell *FreeFrom Modeling Omni* (mit Phantom Omni) kostet derzeit 6.000,- EUR (zzgl. MWST). Über den kompletten Funktionsumfang kann man für 25.000,- EUR (zzgl. MWST) verfügen (*FreeForm Modeling Plus* mit

Phantom Desktop). Die Ergebnisse sprechen für sich (vgl. Abbildung 2.14).

Abbildung 2.14: SensAble Technologies Phantom Omni (Quelle: FreeForm Model Gallery http://www.sensable.com)

Am Rande ist zu erwähnen das dieses Werkzeug wiederum nicht ausschließlich zur Modellgewinnung im eigentlichen Sinne verwendet wird. In der professionellen *digital content creation (dcc)* werden oftmals nur Occlusion-, Height- und Normalmaps exportiert. Diese dienen anschließend dem Visualisieren innerhalb eines anderen Systems. Diese Vorgehensweise erlaubt die Erstellung von z.B. Animationen sehr komplexer Modelle, ohne wirklich das komplexe Objekt zu animieren. Gearbeitet wird mit einer geringeren Detailstufe, die dann mittels Displacementmapping, Nomalmapping und Occlusionmapping während des Renderingprozesses aufgewertet wird.

3 Analyse

3.1 Voxel und Polygone

3.1.1 Bewertung von Voxel- gegenüber polygonalen Darstellungen hinsichtlich des Sculpturing Prozesses

Polygonale Darstellungen sind im allgemeinen Oberflächenrepräsentationen. Dies impliziert, dass keine Informationen über das "Innere" eines Objektes gespeichert werden. Handelsübliche Grafikbeschleuniger sind auf das Rendering von Polygonen optimiert, dies macht diese Darstellungsform sehr effizient auch bei großen Polygonmengen. Der wesentliche Nachteil dieser Darstellungsform äußert sich in einem hohe Grad der Komplexität des Modellierungsprozesses. Gerade die klassische Polygonmodellierung verdeutlicht diese Problematik (vgl. Kapitel 2.4.1 auf Seite 10). Die Interaktion mit dem 3D-Modell durch den formgebenden Designer ist ein abstrakter Prozess. Die Realität besteht nicht nur aus Oberflächen, die Werkzeuge zur Formgebung im Gegensatz dazu basieren auf polygonalen Funktionen. Die Formgebung findet in einer abstrakten nur begrenzt kausalen Software Umgebung statt. In diesem Widerspruch begründet liegt die Notwendigkeit eines Umdenkens während des Sculpturing Prozesses.

Volumendaten hingegen erschließen sich dem menschlichen Verstand um ein Vielfaches leichter. Der Mensch verfügt über eine räumliches Wahrnehmungssystem seiner Umwelt. "Zum einen wird die Entfernung eines Objektes von den Augen wahrgenommen, zum anderen erfolgt über die Kenntnis der Welt und der darin vorkommenden Objekte eine Interpretation der räumlichen Tiefe" (Wikipedia d, Webresource). Da der Benutzer lediglich über ein zweidimensionales Ausgabegerät verfügt, kann er sich nur virtuell um das Objekt bewegen um seine Ausdehnung wahrzunehmen. Projektionen dreidimensionaler Inhalte auf zweidimensionalen Ausgabegeräten werden demnach besser wahrgenommen, wenn eine hohe Bewegungsfreiheit innerhalb der Anwendung besteht.

Ein Nachteil von Volumendaten liegt in der Visualisierung, da sich die Darstellung der Voxel aufwändig gestaltet. Heutige Desktopsysteme verfügen zwar meist über

dreidimensionale Texturen, allerdings liegt der Speicherbedarf von Volumendaten generell sehr hoch. Bei $512^3 Voxel * 1 - 4Byte$ fallen zwischen 128-512 Megabyte (MB) an benötigtem Speicher an. Dabei sind die geometrischen Daten des Volumenrenderings noch unberücksichtigt. Mehr als 512 MB Speicher besitzen derzeit nur Grafikkarten im professionellen Bereich. In der Medizin und im Maschinenbau kommen zu diesem Zwecke dedizierte Volumenrendering-Systeme zum Einsatz. Diese Systeme verfügen über wesentlich mehr Speicher als ein Desktopgrafikbeschleuniger. Somit ist es ihnen möglich, größere Voxelmodelle zu visualisieren. Triangulationsverfahren wie der Marching Cubes Algorithmus und seine Variationen erlauben das Extrahieren einer Polygonoberfläche aus einer gegebenen Voxelmenge und schlagen damit die Brücke zum polygonalen Darstellungsmodell.

Prinzipiell gestaltet sich die Interaktion mit Volumendaten, ohne die Verwendung von dreidimensionalen Eingabe- bzw. Ausgabegeräten, schwierig. Da die Benutzerschnittstelle lediglich eine zweidimensionale Projektion der Daten auf den Bildschirm ausgibt. Es wäre zwar denkbar ein System zu entwerfen, welches dreidimensionale Texturdaten manipuliert um ein Voxelmodel zu erzeugen. Werden die Daten über Schnittflächen im Raum visualisiert, so wäre es allerdings notwendig, die Flächen, welche vor dem zu verändernden Elementen liegen, auszublenden, um den Zielvoxel durch die Benutzerschnittstelle interaktiv selektieren zu können. Die Interaktion dieses Systems würde hinsichtlich des Sculpturing Prozesses alles andere als einfach zu handhaben sein.

3.1.2 Synthese im Sinne des Sculpturing Ansatzes

J. A. Bærentzen hat gezeigt, dass mittels des Ray-Casting Verfahrens prinzipiell eine interaktive Visualisierung möglich ist (vgl. Kapitel 2.4.2 auf Seite 16). Wobei er von der Tatsache ausgeht das nur der veränderte Bereich des Modelles neu visualisiert werden muss. Das Ray-Casting Verfahren ist sicherlich eine Möglichkeit Volumen in Echtzeit zu visualisieren. Es impliziert jedoch, dass bei einer Drehung des Objektes alle Daten neu abgetastet werden müssten. Bei der Verwendung eines kleinen Objektes mag dies keine Problem darstellen, allerdings wird von einem Modellierungssystem erwartet, dass der Benutzer über eine gewisse Bewegungsfreiheit innerhalb des Systems verfügt, da der Designer ohne eine Form der Stereoprojektion auskommen sollte. Bezüglich dieser Bewegungsfreiheit ist jedoch davon auszugehen, dass bei einer Bildschirmauflösung von 800x600 Pixel im schlechtesten Falle 480.000 Strahlen in den Raum ausgesendet werden müssten, um beispielsweise ein herangezoomtes Objekt zu drehen. Da sich in diesem Falle alle Pixel des Bildes ändern würden. Deshalb wird das Ray-Casting Verfahren ohne weitere Tests als zu aufwendig eingeschätzt.

Wenn nun die räumliche Besetzung so besser mit unserer Wahrnehmung vereinbar ist, Oberflächen sich aber effizienter darstellen lassen, dann wäre mein Vorschlag eine Erweiterung des von J. A. Bærentzen (siehe Kapitel 2.4.2 auf Seite 16) vorgestellten Ansatzes, um ein polygonales Rendering von Voxeldaten. Anders als bei Volumenrendering Algorithmen soll dabei prinzipiell ein Polgonwürfel pro Voxel gezeichnet werden. Damit erhält man theoretisch wieder eine Oberfläche und kein Volumen als visualisiertes Abbild, dieses ist für den Sculpturing Prozess auch nicht notwendig. Wie schon Hughes und Galyean es 1991 getan haben, soll dieses System den Marching Cubes Algorithmus verwenden, um die gespeicherten Voxeldaten in eine polygonale Oberfläche zu überführen (vgl. Kapitel 2.4.2 auf Seite 15).

Im Folgenden wird anhand eines Fallbeispiels analysiert, inwiefern sich eine Form des polygonalen Voxelrendering auf einem modernen Desktopsystem realisieren lässt. Dazu werden zunächst die zwei bekanntesten Programmierschnittstellen zur Visualisierung näher betrachtet.

3.2 OpenGL oder Direct3D

OpenGL und Direct3D sind Programierschnittstellen (API)[1] zur Visualisierung dreidimensionaler Inhalte. Diese ermöglichen dem Programmierer den Zugriff auf Teile der Grafikkarte des Rechners. Der offene Standard OpenGL steht auf allen gängigen Plattformen zur Verfügung. Die GLut (GL Utilities) erleichtert dabei die Verwendung dieser Schnittstelle. OpenGL verwendet keine Software-Objekte, Funktionen werden in strukturierter Form aufgerufen.

Die Direct3D Schnittstelle wird nur von Microsoft Software unterstützt (Windows bzw. XBox). Direct3D wurde mit einer Reihe anderer APIs unter dem Namen DirectX 9 zusammengefasst. Dazu gehören unter anderem die Audio-Schnittstellen Direct Sound und Direct Music, die Schnittstelle für Eingabegeräte Direct Input, sowie Direct Play, welches die Netzwerkkommunikation unterstützt. Diese Programmierschnittstellen verfolgen teilweise unterschiedliche Strategien innerhalb Ihrer Architektur. Prinzipiell allerdings verfügen beide über einen ähnlichen Funktionsumfang bezüglich der Visualisierung. Dabei werden Vertex- und Pixelshaderprogramme von beiden[2] Schnittstellen unterstützt.

Aus Erfahrung mit beiden Schnittstellen wird Direct3D zur Visualisierung präferiert. Dies hat im wesentlichen zwei Gründe. Zum einen bietet diese API ein hohes

[1] API (engl. application programming interface, deutsch: Schnittstelle zur Anwendungsprogrammierung)

[2] OpenGL Shading Language (GLSL) bzw. DirectX 9 High Level Shading Language (HLSL)

Maß an Abstraktion. OpenGL ist eine Zustandsengine, das bedeutet jeder Funktionsaufruf bleibt solange aktiv bis ein anderer Aufruf diesen Zustand verändert. Direct3D arbeitet prinzipiell nach dem selbem Schema mit dem Unterschied, dass wesentliche Teile dieser API objektorientiert programmiert werden. Beispielsweise sind die Vertices der dreidimensionalen Objekte (der Vertexbuffer und Indexbuffer) in sogenannten *Container* Objekten gekapselt (Microsoft, Webresource). Im Gegensatz zu OpenGL, gestaltet sich gerade das Handling von einzelnen 3D-Objekten durch die Kapselung in Containerobjekte leichter. Zusätzlich existieren eine Reihe von Hilfsfunktionen zur Weiterverarbeitung dieser Objekte. Andererseits wird das Debugging der einzelnen Funktionsaufrufe von Direct3D sehr gut unterstützt. Der Programmierer hat jederzeit die Möglichkeit eine Blick in den Speicher der Grafikkarte zu werfen, um eventuell auftretende Probleme zu erkennen. Diese Funktionalität gibt zusätzlich Auskunft über verworfenen Direct3D-Funktionsaufrufe, welche von der 3D-Grafikkarte nicht bearbeitet werden, weil sie entweder nicht unterstützt werden oder sinnlos bzw. fehlerhaft sind. Letzteres kommt häufiger vor. Der Debug Modus der Direct3D-Schnittstelle macht es dabei überflüssig, das Ergebnisse eines Funktionsaufrufs zur Problembeseitigung von Hand auszuwerten.

3.3 Machbarkeitsstudie: polygonales Voxelrendering

Ein dreidimensionales Modellierungssystem impliziert ein Verhalten von Aktion des Benutzers und eine zeitnahe Reaktion des Systems. Dieser Zusammenhang wird im Allgemeinen als Echtzeit bezeichnet. Der Begriff ist dehnbar und wir deswegen unterschiedlich ausgelegt. Ein System, welches mit einer Bildwiederholrate von 10 Bildern pro Sekunde arbeitet, würde man in jedem Falle als ein Echtzeitsystem bezeichnen. Grundsätzlich sind Bildwiederholraten von mehr als 3 Bildern pro Sekunde (FPS)[3] innerhalb der 3D Computergrafik als Echtzeit anzusehen. Damit das Verhalten auch zugänglich wird, wäre jedoch ein Ergebnis von mindestens 10 FPS erstrebenswert.

3.3.1 Leistungsfähigkeit moderner Desktop Grafikbeschleuniger

Die Leistungsanforderungen moderner 3D-Anwendungen sind einem stetigen Wachstum unterworfen. Vor allem die Videospielindustrie treibt diesen Prozess voran.

[3] engl. frames per second (FPS)

Tabelle 3.1: Leistungsvergleich technischer Daten der Grafikbeschleuniger Nvidia GeForce 7800 GTX und ATI Radeon X1900 XTX (Quelle: computerbase 2006, Webresource)

	GeForce 7800 GTX	**Radeon X1900 XTX**
Chipsatz	G70	R580
Transistoren	ca. 303 Mio	ca. 384 Mio
Fertigungprozess	110 nm	90 nm
Chiptakt	550 MHz	650 MHz
Texelfüllrate	13200 MTex/s	10400 MPix/s
Dreiecksdurchsatz	1100 MV/s	1300 MV/s
Speicher	512 MB DDR3	512 MB DDR3
Speichertakt	850 MHz	775 MHz
Speicherinterface	256 Bit	256 Bit
Speicherbandbreite	54400 MB/s	49600 MB/s

Der Markt reagiert auf die stetig steigenden Anforderungen mit immer neuen Produkten im Bereich der Grafikbeschleuniger. Im wesentlichen besteht der Endkonsumentenmarkt aus der *Nvidia Corporation* und der *ATI Technologies Incorporated*. Beide Firmen liefern sich einen harten Konkurenzkampf, was unter anderem zu ungenauen Informationen über die Tatsächliche Leistungsfähigkeit ihrer Produkte führt. Die Tabelle 3.1 ist aus diesem Grunde einem Vergleichstest der Webseite www.computerbase.de entnommen (computerbase 2006, Webresource). Als Grafikbeschleuniger im Workstation-Segment leistet "die FX 4500[4] < 181 Mio Dreiecke pro Sekunde und erreicht dabei eine Füllrate von ca. 10,8 Mrd Texel/Sekunde" (Production 2005, S.16). Diese Angabe ist von Mitte 2005, deshalb ist davon auszugehen, dass die Leistung bereits im Bereich um die 500 Mio Dreiecke pro Sekunde pro GPU operiert. Nvidia hat auf der *SIGGRAPH* 2006 eine Clusterlösung vorgestellt. In einem externen Gehäuse finden bis zu 8 Quadro Grafikbeschleuniger Platz, die per Scalable Link Interface (SLI) zusammengeschaltet werden und in etwa die 8fache Leistung einer einzelnen GPU zu Verfügung stellen.

3.3.2 Ausgangssituation

Der Benutzer soll möglichst viele Voxel als Polygonwürfel auf dem Bildschirm ausgeben und dabei mit diesen interagieren können. Diese relativ einfach gefasste Zielstellung wirft hinsichtlich der technischen Basis zwei Fragen auf:

1. Wie viele Dreiecke pro Sekunde sind auf dem Zielsystem darstellbar?

[4]Nvidia Corp. Quadro FX 4500 Workstation Grafikkarte

Tabelle 3.2: Anzahl der benötigten Polygone pro Sekunde für polygonales Voxelrendering bei 10 Bildern pro Sekunde (Anzahl Voxel * 12 Polygone/Voxel * 10 FPS)/2 wegen des Backfacecullings.

Voxel Anzahl	Polygondurchsatz pro Sekunde
512^3	8.053.063.680
256^3	1.006.632.960
128^3	125.829.120

2. Wie viele einzelne dreidimensionale Objekte lassen sich mit dem Zielsystem verwalten, so dass eine Selektion eines 3D-Objekts unter Echtzeitanforderung möglich ist?

Zur Illustration der Leistungsanforderungen wurden in der Tabelle 3.2 die benötigten Polygone für ein polygonales Voxelrendering bei gängigen Größen des Voxelfeldes zusammengefasst. Es wird davon ausgegangen, dass pro Voxel ein Polygonwürfel dargestellt werden muss. Die anvisierte Bildwiederholrate liegt bei 10 Bildern pro Sekunde. Die Verdeckung rückseitiger Polygone (Backfaceculling) würde zwar die Anzahl der darzustellenden Dreiecke halbieren. Prinzipiell müssten diese allerdings vorhanden sein, respektive während der Selektion mit betrachtet werden. Weitere Verfahren zur Reduktion der darzustellenden Elemente werden zunächst nicht angewendet.

Rund 8 Milliarden Polygone übersteigen die Leistungsfähigkeit eines modernen Grafikbeschleunigers um ein Vielfaches. Selbst $\backsim$ 1 Milliarde Polygone sind nicht in Echtzeit darstellbar. Ungefähr 125 Millionen Polygone pro Sekunde wären mit der neusten Hardware bezüglich der Polygonleistung realisierbar. Dies entspricht einem Modell mit Rund 25 Millionen Dreiecken[5].

Fazit zum Polygondurchsatz

Es existieren Hardwarekomponenten, welche über einen hinreichend großen Polygondurchsatz verfügen, um ein Voxelfeld mit 128^3 Elementen mittels polygonaler Darstellung abzubilden. Würde man nun ein einziges 3D-Objekt mit einer solch großen Polygonanzahl verwenden, um ein Voxelmodell zu erzeugen, stellt sich immer noch die Frage der Interaktion mit der Topologie dieses Gitterobjektes.

Das 3D-Objekt läge in diesem Falle in dem Speicher der Grafikkarte. Jegliche Interaktion, zum Beispiel das Treffen einer Auswahl, das sogenannte *Picking* von Polygonen, würde die Berechnung einer Intersektion mit diesem Objekt zur Folge ha-

[5] (Rund 125 Millionen multipliziert mit 2, wegen des Backfaceculling, und dividiert durch 10 Bilder pro Sekunde ergibt rund 25 Millionen Dreiecke)

ben. Die DirectX 9 API führt eine Intersektion auf der Ebene der Grafikkarte aus. Dies impliziert eine sehr hohe Geschwindigkeit der notwendigen Vektorberechnungen. Moderne GPU's verfügen über mehr als die 10fache Leistung der eigentlichen System-CPU im Bereich der Vektorrechnung. Eine Intersektion mit $\backsim$ 25 Millionen Polygonen stellt allerdings einen Grenzfall dar, da schließlich die eigentliche Darstellung der Polygonmenge die Grafikkarte an ihrer Leistungsgrenze betreibt.

Es besteht die Möglichkeit das 3D-Objekt mittels eines Oktonärbaums[6] zu partitionieren. Da allerdings dieser Ansatz der Partitionierung des Objektes nicht "kostenlos" zu haben ist, sondern automatisch zu einer Verlagerung des Problems auf die CPU-Seite des Systems führt, wird im Folgenden die Leistungsfähigkeit des zur Verfügung stehenden Systems im Bezug auf Objektanzahl unter Echtzeitanforderungen ermittelt.

3.3.3 Versuch: Bildwiederholrate im Bezug zu steigender 3D-Objekt-Anzahl bei variierender Polygonanzahl pro Objekt

Zur Erinnerung hier noch mal die in Abschnitt 3.3.2 aufgeworfene Frage: Wie viele Objekte lassen sich in dem Zielsystem effizient verwalten, so dass eine gezielte Selektion unter Echtzeitanforderung möglich ist?

Der folgende Versuch soll einerseits ermitteln, wie viele einzelne 3D-Objekte sich nacheinander zeichnen lassen, ohne den Bereich der Echtzeit zu verlassen. Andererseits ist von Interesse, wie sich die Grafikkarte bei steigenden Polygonmengen verhält. Dabei ist zu beachten, dass ein 3D-Objekt in diesem Versuch im wesentlichen aus zwei Teilen besteht. Zum einen die 3D-Daten, die in der Grafikkarte gehalten werden. Auf der anderen Seite das Softwareobjekt, das zur Laufzeit verwaltet wird. Letzteres stellt Methoden zur Positionierung, Kolorierung und Darstellung des 3D-Objekts zur Verfügung.

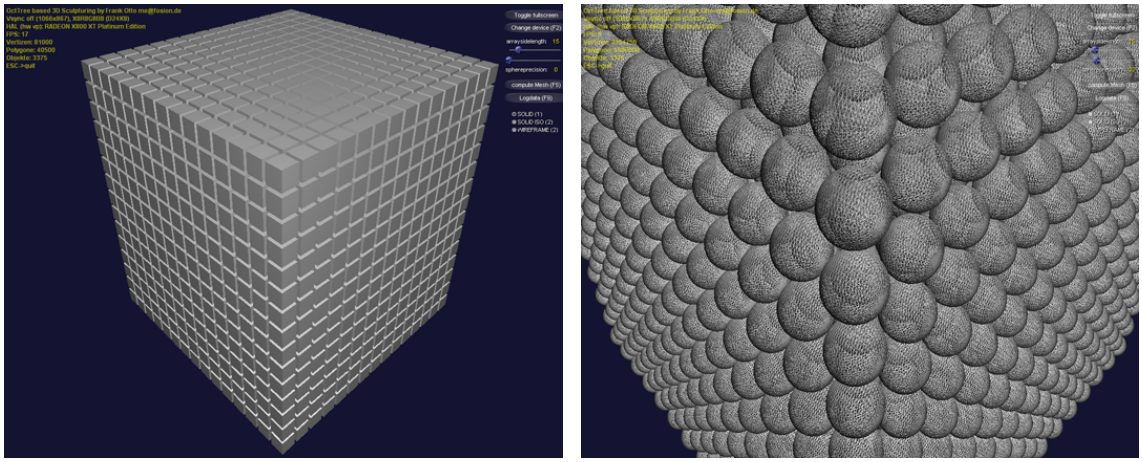
Versuchsaufbau

Hardware: Die System-CPU ist ein *AMD XP 3200* mit 1,5 GB DDR Speicher bei 400 Mhz. Als Grafikbeschleuniger wird die *ATI Radeon X800 XT* mit 256 MB VRAM (256 Bit Busbreite) benutzt.

[6] vgl. Kapitel 2.3.2 auf Seite 8

Software Prototyp: Der Prototyp, aufgesetzt auf das Sample Framework der DirectX 9 API von Microsoft, verfügt über eine hardwaregestützte Transformation (Matrixtransformation), mittels derer sich Objekte innerhalb der Szene positionieren lassen. Der Anwendungsfall des Voxelfeldes wurde durch ein dreidimensionales Feld bestehend aus Szenen-Objekten[7] erstellt. Die Seitenlänge des Feldes ist frei skalierbar. Die Objekte werden in einem statischen, eindimensionalen Feld gehalten und mittels einer "for" Schleife gezeichnet. Als eigentliche Gitterobjekte werden Würfel und Kugeln eingesetzt. Die Genauigkeit der Kugeln ist parametrisierbar, wobei der maximale Testwert, 150 Unterteilungen in X- und Y-Richtung, zu einer Menge von 22352 Dreiecken pro Kugel führt.

Versuchsablauf: Schrittweise werden die Seitenlängen des Objektfeldes erweitert. Die 3D-Objekte werden mit variierenden Polygonanzahlen von 12-22352 Dreiecken pro 3D-Objekt gezeichnet. Die Kugel-Objekte skalieren die Anzahl der Polygone pro 3D-Objekt (vgl. Abbildung 3.1b). Die Ausgabeauflösung beträgt 1280*1024 Bildpunkte Vollbild. Zur anschließenden Analyse wurden die Messwerte in eine *CSV*[8] Datei ausgegeben.



(a) Rendering mit Würfeln als Voxelelemente (Gouraud-Shading)

(b) Rendering mit Kugeln zur Steigerung der Polygonanzahl (Wireframe)

Abbildung 3.1: Versuch: Polygondurchsatz

Anmerkung: Es zeichnete sich während des Versuchs ab, dass bei einer Seitenlänge von > 6 und hohen Werten für die Anzahl der Dreiecke pro Objekt die Bildwiederholrate unterhalb von 10 FPS fiel. Deshalb wurde an diesen Stellen davon abgesehen,

[7] vgl. Abbildung A.1 auf Seite 68 (VxM_SceneObject.h)

[8] CSV engl. für Character Separated Values, Comma Separated Values

bis 22352 Dreiecke pro Objekt zu testen. Bei einer Seitenlänge größer 16 erzielte der Prototyp Bildwiederholraten unterhalb von 1 FPS[9]. An dieser Stelle wurde der Versuch abgebrochen.

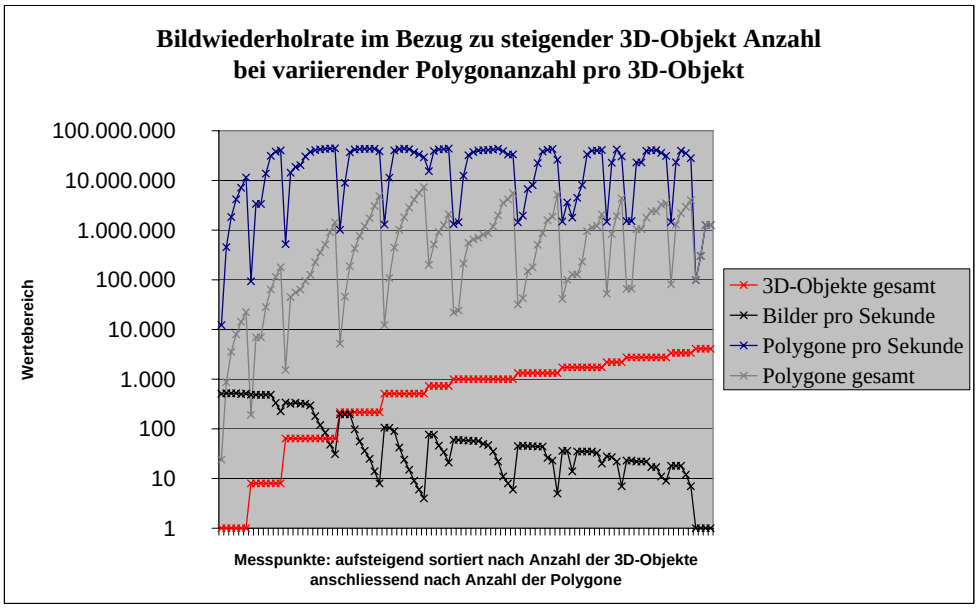


Abbildung 3.2: Messung der Leistung des Arbeitssystems

Analyse der Ergebnisse

Grundsätzlich ist davon auszugehen, dass die Messwerte lediglich ein Gefühl für die Geschwindigkeit des Zielsystems vermitteln, da die Implementierung prototypisch realisiert wurde und somit ihr Optimum mit Sicherheit nicht erreicht hat. Das Diagramm Abbildung 3.2 fasst alle Messergebnisse zusammen. Die logarithmische Einteilung der Ordinate erlaubt die Darstellung aller Messwerte innerhalb eines Diagrammes. Auf der Abzisse wurden die einzelnen Messpunkte, aufsteigend nach Anzahl der Objekte sortiert bzw. Anzahl der Polygone pro Objekt, aufgetragen. Die beiden oberen Kurven geben die Menge der insgesamt dargestellten Polygone/Vertices wieder.

[9]Eine Seitenlänge 16 ergibt eine Gesamtanzahl von 4096 Objekten

Tabelle 3.3: Auszug der Messwerte des Versuches

3D-Objekte	Polygone gesamt	FPS	Polygone/3D-Objekt	Polygone/Sekunde
216	5184	196	24	1.016.064
216	45792	195	212	8.929.440
216	188352	196	872	36.916.992
216	428112	98	1982	41.954.976
...	...	...	...	...
3375	81000	18	24	1.458.000
3375	1289250	18	382	23.206.500
3375	2200500	18	652	39.609.000
3375	2943000	12	872	35.316.000

Bei steigender Objektanzahl zeigt sich die Bildwiederholrate in ihrer Tendenz rückläufig. Ein sehr interessanter Effekt wird gerade bei geringeren Objektmengen deutlich. Die Bildwiederholrate scheint bei gleichbleibender Objektanzahl und steigenden Polyonanzahlen bis zu einem gewissen Grade annähernd konstant zu bleiben. In der Tabelle 3.3 wurden auszugsweise Werte zusammengestellt, bei denen dieser Effekt besonders ausgeprägt ist.

Der Prototyp scheint bei größeren Polygonmengen seinen Wirkungsgrad zu steigern. Bei 216 Objekten und 188.352 Polygonen leistet das System 196 FPS. Bei gleicher Objektanzahl und mehr als doppelt so vielen Polygonen (428.112) wird genau die Hälfte der Bildwiederholrate von 98 FPS erreicht. Man würde an dieser Stelle tendenziell eine geringere Bildwiederholrate erwarten[10]. Vermutlich ist dieses Verhalten auf die wesentlich höhere Speicherbandbreite der Grafikkarte zurückzuführen. Da dieses Phänomen nicht Teil der Untersuchung ist wird an dieser Stelle davon abgesehen dieses Verhalten näher zu ergründen.

Bei 64 3D-Objekten mit 22.352 Dreiecken pro 3D-Objekt erzielt der Prototyp seinen maximalen Polygondurchsatz von 44.346.368 Dreiecken pro Sekunde (bei 31 FPS). Weitere Tests haben ergeben, dass sich auf dem Zielsystem $\backsim$ 3800 3D-Objekte bei einer Bildwiederholrate von $\backsim$ 17 FPS wiedergeben lassen. Bei größeren Objektmengen bricht die Bildwiederholrate schnell auf unterhalb von einem Bild pro Sekunde ein.

[10] Eine Menge von 188.352 Polygone entspricht 196 FPS, 98 FPS würde man bereits bei 376704 Polygonen vermuten.

Fazit:

Der Polygondurchsatz der Grafikkarte steht in keinem Verhältnis zu der tatsächlich möglichen Anzahl zu zeichnender 3D-Objekte. Da jeder Zeichenvorgang von der CPU an den Grafikbeschleuniger gesendet werden muss, bildet die CPU-Leistung im Zusammenspiel mit der Speicherbandbreite des Systems das Nadelöhr innerhalb einer 3D Anwendung. Anzumerken ist, dass die vorhandene Geometrieleistung der Grafikkarte prinzipiell erlauben würde, anstatt einzelner Würfel, Kugeln zu zeichnen.

Weiterhin wird ersichtlich, dass ohne eine Reduktion der darzustellenden Elemente kein Feld aus einzelnen 3D-Objekten zur Darstellung eines Volumen verwendet werden kann. Die Größe des Feldes erreichte eine Kantenbreite von 16 Elementen. Dies reichte nicht an die Anforderungen eines Volumensculpturing heran. Das enstehende Modell wäre einfach zu ungenau. Aus diesem Grunde werden im folgenden Abschnitt die Möglichkeiten der Reduktion darzustellender Elemente im Bezug auf den Anwendungsfall analysiert.

3.3.4 Reduktion der darzustellenden Elemente

Würfel oder einzelne Flächen?

Ein Würfel hat grundsätzlich die Eigenart, dass maximal drei Seiten gleichzeitig sichtbar sind. Man könnte auf Basis dieses Zusammenhanges sicherlich die Menge der benötigten dreidimensionalen Punkte verringern. Allerdings hat der vorangegangene Versuch gezeigt, dass eine Reduktion der dreidimensionalen Daten keinen Gewinn bezüglich der möglichen Menge der einzeln referenzierbaren 3D-Objekte bringt.

Daraus folgte lediglich, dass es grundsätzlich machbar ist, Kugeln anstelle von Würfeln darzustellen, ohne dabei an Leistung zu verlieren. Das Verdecken der rückseitigen Polygone bringt eine bis zu 15-prozentige Leistungssteigerung mit sich. Dieses Verhalten haben Versuche am Prototypen ergeben. Allerdings wirkt sich diese Leistungssteigerung nicht auf die Anzahl möglicher 3D-Objekte aus, da das sogenannte *Backfaceculling* von der Grafikkarte ausgeführt wird.

Logische Sichtbarkeit und der Oktonärbaum

Die Analyse des Versuchs hat gezeigt, dass nur eine sehr begrenzte Menge an Objekten zeichenbar ist. Es waren ca. 3800 Objekte möglich. Der Polygondurchsatz des Zielsystems erlaubt hingegen ein ausreichend komplexes Objekt zur polygonalen Voxeldarstellung. Angenommen man partitioniert ein einzelnes Polygonobjekt

mittels eines Oktonärbaums[11] und reduziert damit die Anzahl der notwendig zu berechnenden Intersektionen mit dem Modell. Die direkte Folge wäre eine mögliche Interaktion mit dem Objekt unter Echtzeitbedingungen. Des Weiteren könnte eine große Anzahl von Würfeln abgebildet werden.

Dieses Verfahren brächte allerdings eine Reihe von Problemen mit sich. Ein Oktonärbaum zur Optimierung der Displayliste wird lediglich einmal beim Start der Anwendung generiert und nicht ständig reorganisiert. Dies ist mit "Kosten" in Form von Rechenzeit verbunden. Da alle Voxel innerhalb eines Objektes gehalten werden, müssten diese ihrerseits in einer Struktur organisiert sein. Um letztlich ein korrektes Gitterobjekt aus dieser polygonalen Voxeldarstellung zu gewinnen, bedarf es zusätzlich des Marching Cubes Algorithmus. Diese Herangehensweise setzt wiederum voraus, dass man über ein Voxelmodell verfügt.

Zusammenfassend ist zusagen, dass einerseits Würfel/Kugeln innerhalb eines Objektes der Grafikkarte verwaltet werden müssten. Auf der anderen Seite wäre es notwendig einen Oktonärbaum unter Echtzeitbedingungen zu reorganisieren, um die Interaktion mit dem System zu gewährleisten. Bei einem so umfangreichen dreidimensionalen Objekt (bis zu 25.165.824 Polygone) ist mit einem erheblichen Aufwand für die Reorganisation der Zeiger auf die einzelnen Dreiecke zurechnen. Da dieser Prozess Zugriffe zwischen System-CPU und Grafikkarte erfordert, würde dieser Vorgang zu Lasten der Geschwindigkeít des System gehen. Des Weiteren wäre ein einzelnes Polygonobjekt in eine Voxeldarstellung zu überführen, um es an den Marching Cubes zu übergeben. Aus dieser Betrachtung wird geschlossen, der Ansatzpunkt muss das Voxelmodell selbst sein. Deshalb wird im nachfolgenden Abschnitt der Vorschlag von J. A. Bærentzen, einem Oktonärbaum zur Speicherung der Voxeldaten zu verwenden, diskutiert.

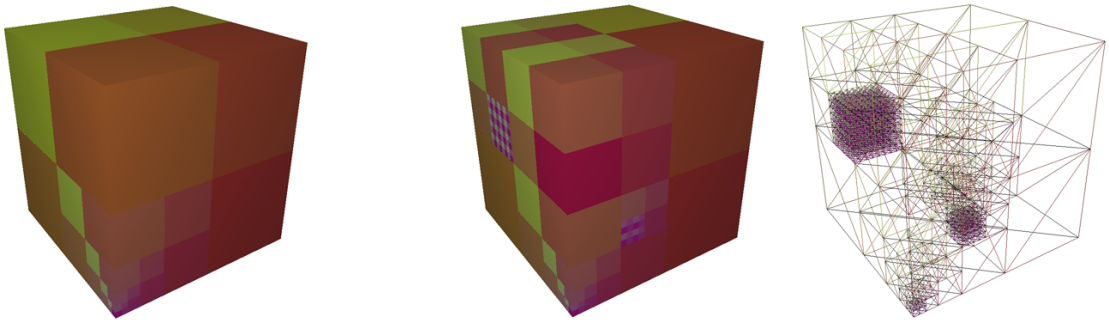
3.3.5 Der Oktonärbaum als Voxelmodell

Gewöhnlich wird ein Oktonärbaum "nicht als Datenstruktur für Voxel Daten verwendet"(Watt 2002, Seite 68). Bærentzen hat gezeigt, dass die Organisation der Voxedaten in einer Oktonärbaum-Struktur von Vorteil bezüglich der Leistungsfähigkeit des Gesamtsystems sein kann. Da nur dort, wo Voxel entfernt werden, auch eine entsprechend feine Unterteilung des Oktonärbaums existieren muss, reduziert sich die Anzahl der darzustellenden Elemente erheblich. Die Tiefe des Baumes steht dabei in Relation zur Anzahl der logisch abgebildeten Voxel. Bei einer Tiefe von 8 Knoten würden $2^8 * 2^8 * 2^8$ Voxel abgebildet. Auf diese Weise sollte es möglich sein, anstatt alle Voxel in einem 3D-Objekt zu halten, prinzipiell einen Würfel pro Voxel zu verwenden. Zusätzlich erlaubt die Partitionierung des Raumes einen Test

[11] siehe Kapitel 2.3.2 auf Seite 8

auf Verdeckung der einzelnen Knoten. Wodurch im Maximum erreicht würde, dass ein Voxelmodel aus wesentlich mehr, als der theoretisch darstellbaren Anzahl von Objekten, bestehen könnte.

Bei der Verwendung eines Oktonärbaum zur Optimierung der darzustellenden Elemente speichert jedes Blatt eine Liste von Zeigern auf einzelne Polygone, die meist zu ein und demselben Objektes gehören. Der Baum wird in seiner Funktionsweise dahingehend modifiziert, dass jeder Knoten die Adresse eines separaten 3D-Objekts speichert. Dieses Objekt wird in der Grafikkarte lokalisiert gehalten. Diese Vorgehensweise ermöglicht es, einen sehr schnellen Test auf Intersektion innerhalb der Grafikkarte auszuführen und somit eine Interaktion mit dem Modell zu gewährleisten. Des Weiteren wird jeder Knoten um eine Methode die das Objekt zeichnet erweitert. Der Prototyp des Versuches[12] wurde hinsichtlich dieser Vorgaben erweitert (vgl. Abbildung 3.3a). Zu diesem Zeitpunkt der Entwicklung, als die Selektion eines



(a) Rendering eines Oktonärbaum

(b) Rekursive Verfeinerung einiger Knoten des Baums

Abbildung 3.3: Rendering eines Oktonärbaums in einer frühen Phase des Prototypen

der Würfel noch nicht implementiert war, wurden einige Versuche im Zusammenhang mit der Verfeinerung der Knoten des Baums unternommen (vgl. Abbildung 3.3b). Der Prototyp lieferte dabei 38 Bilder pro Sekunde, folglich kann mit Hilfe dieser Datenstruktur prinzipiell ein Voxelmodell polygonal dargestellt werden.

Ein Problem innerhalb dieses Ansatzes stellt die Überführung dieses Oktonärbaums in ein Voxelfeld dar. Ein Lösungsansatz besteht in der Ermittlung einer Raumbelegung durch gerichtete Traversierung des Baums. Der von J. A. Bærentzen diskutierte Ansatz des *multiresolution sculpting* sollte sich bei Verwendung einer Raumbelegung realisieren lassen, da es für diesen Algorithmus prinzipiell eine untergeordnete Rolle spielt, ob ein Voxel nun diese oder jene Ausdehnung hat. Dieses Problem gilt es noch zu lösen.

[12] vgl. Kapitel 3.3.3 auf Seite 27

3.4 Fazit

Die vorangegangene Analyse hat gezeigt, dass ein polygonales Voxelrendering auf dem Zielsystem im Bereich des Möglichen liegt. Der Prototyp verfügt bereits über einige Grundfunktionalitäten zur Selektion bzw. Interaktion mit seinem Oktonärbaum. Im Folgenden Kapitel werden die Anforderungen an das Sculpturing System definiert.

4 Anforderungsdefinition

4.1 Zielstellung

Als Primärziel wird die Erstellung einer polygonalen Visualisierung eines Voxelmodells, welche zur Modellierung eines 3D-Objektes Verwendung finden soll. Eine anschließende Triangulation soll das Voxelmodell in ein Polygonmodell überführen. Das System soll die Modellierung nach dem Sculpturing Ansatz ermöglichen. Unter Sculpturing wird das Bearbeiten eines "festen" Ausgangsmaterials verstanden. Die gewünschte Form wird durch Wegnahme von Elementen aus einem Würfel herausgearbeitet. Als Datenmodell findet ein Oktonärbaum Verwendung, welcher in der Form von einzelnen Würfeln polygonal auf dem Bildschirm dargestellt wird. Dieses Modell soll in eine Polygondarstellung überführbar sein. Der Benutzer sollte über ausreichend Bewegungsfreiheit innerhalb der Benutzeroberfläche verfügen. Natürlich muss das System die grundlegenden Funktionen eines Modellierungswerkzeuges beherrschen. Dazu gehört die persistente Speicherung des Systemzustands und die Möglichkeit ein Objekt in ein verbreitetes 3D-Fileformat zu exportieren. Dabei findet das *Wavefront OBJ* Format Verwendung, da es sich dabei um ein sehr einfach aufzubauendes Fileformat handelt (vgl. Tabelle 4.1 auf Seite 38). Nicht zuletzt wird ein skalierbares Werkzeug benötigt welches zumindest verschieden große "Voxel" entfernen kann.

4.2 Funktionelle Anforderungen

Die funktionellen Anforderungen an den Protypen ergeben sich aus der Zielstellung wie folgt: Ein Oktonärbaum soll als Volumenmodell eingesetzt werden. Der Benutzer soll unterschiedlich große Elemente aus dem Modell entfernen können. Das Volumenmodell soll mittels des Marching Cubes Algorithmus in ein Polygonmodell überführbar sein. Das erzeugte Polygonmodell soll als OBJ-File exportiert werden können. Die Struktur des Oktonärbaum soll persitent sein, um eine weiterführende Bearbeitung des Modells zu ermöglichen.

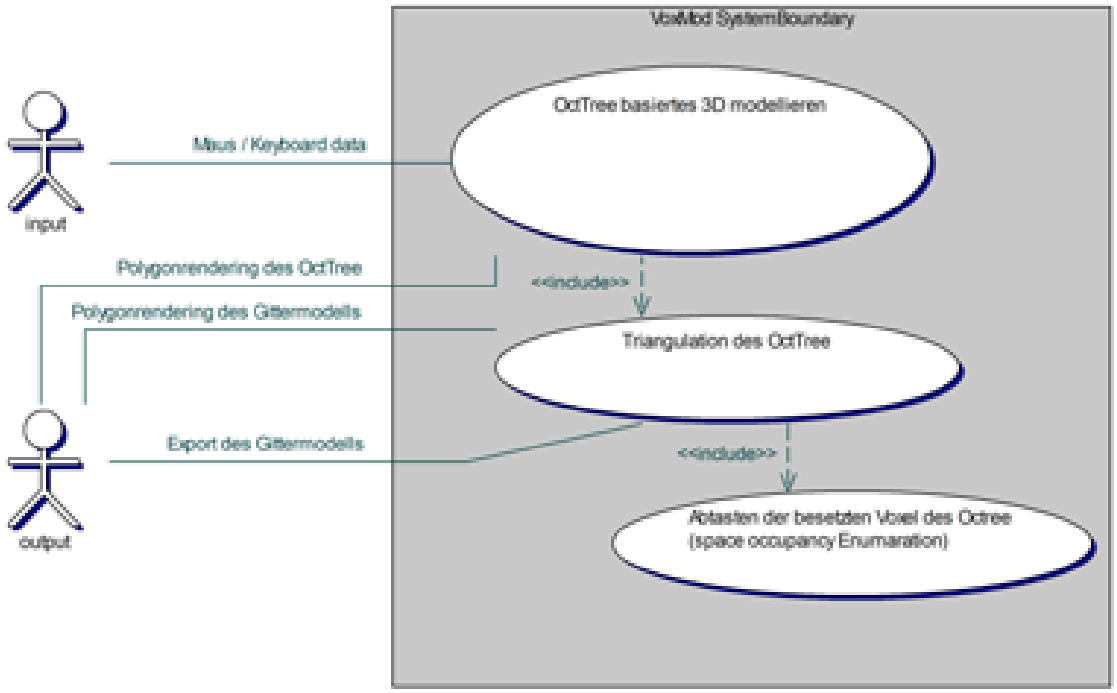


Abbildung 4.1: Der Anwendungsfall des Sculpturing Systems

Wunschkriterien Die Entwicklung des Sculpturing System erfordert die Kombination einer Reihe verschiedener Techniken, welche jede für sich betrachtet einen fortgeschrittenen Grad der Komplexität besitzt. Da es bei der Implementierung noch einige Probleme zulösen gilt, werden die folgenden Kriterien zunächst als Wunschkriterien eingeordnet. Dazu gehört das Bemalen des Voxelmodell bzw. Gitterobjektes, sowie die Abbildung verschiedener Werkstoffe bzw. Werkzeugformen. Zusätzlich wäre die direkte Gestaltung des Polygonmodells (ohne Voxelmodell), innerhalb derer der Marching Cubes die Visualisierung des Sculpturing Prozesses übernimmt, wünschenswert.

4.3 Nicht funktionelle Anforderungen

Das System soll dem Benutzer genügend Bewegungsfreiheit innerhalb der dreidimensionalen Szene einräumen. Damit ist in erster Linie die Rotation/Translation der Kameraposition aber auch das so genannte *strafing* gemeint. Bei letzterem handelt es sich um die Verschiebung der Kameraposition entlang ihrer X- bzw. Y-Achse.

4.4 Abgrenzungskriterien

Dieses Sculpturing System wird lediglich prototypisch realisiert. Es werden ausgewählte Prinzipien, der in den Grundlagen vorgestellten Arbeiten, zur Volumenmodellierung verwendet. Diese Arbeit dient dem Verständnis der technischen Zusammenhänge von Volumensculpturing. Die gewonnenen Erkenntnisse sollen in die zukünftige Entwicklung eines Volumensculpturing-Systems einfließen.

4.5 Hardware Anforderungen

Abgeleitet aus der vorangegangenen Analyse (Kapitel 3.3) sollte das Zielsystem über folgenden Eigenschaften verfügen:

Ein Prozessor der X86er Familie mit einer Taktfrequenz von 2 GHz, sowie mindestens ein Gigabyte (1GB) Speicher sollten vorhanden sein. Der Grafikbeschleuniger muss uneingeschränkt die DirectX 9 Schnittstelle unterstützen und sollte mindestens über 64 Megabyte Speicher verfügen. Als Betriebsystem wird WindowsXP mit DirectX 9.0 (mind. August2006 update) vorausgesetzt. Der Prototyp wurde auch auf Systemen mit geringerer Leistung getestet, dabei hat sich gezeigt dass die *DXUT* als Mindestanforderung ein DirectX 9 Gerät beansprucht.

4.6 Daten

Als Format für den 3D-Daten Export wird das OBJ-Format verwendet. Die Tabelle 4.1 auf der nächsten Seite verdeutlicht den Aufbau dieses 3D-Formats.

Tabelle 4.1: Aufbau des OBJ Fileformats (Joswig u. Polthier 2000, Webresource)

Eintrag	**Bedeutung**
v float float float	Ein einzelner Vertex verkörpert durch einen dreidimesionalen Vektor bestehend aus Float-Werten. Der erste Vertex innerhalb der Datei hat den Index 1 (bspw. v1 1.0 1.0 1.0); alle darauf folgenden werden sequentiell nummeriert.
vn float float float	Der Normalvektor. Sein Index beginnt ebenfalls mit 1 und alle darauf folgenden werden sequentiell nummeriert. Die Zugehörigkeit zu den einzelnen Vertices ergibt sich aus der Indizierung.
vt float float	Eine Textur Koordinate. Die Indizierung gleicht den Vorangegangenen. Auch hier ergibt sich die Vertexzugehörigkeit aus dem Index.
f int int int	Mehrere Punkte werden zu einer polygonalen Fläche zusammengefasst. Die Integerwerte stellen die Indices der verwendeten Vertices dar. Bei der Anwendung von Normalen bzw. Texturkoordinaten werden die Indices dieser jeweils zusammengefasst(f int/int/int int/int/int int/int/int). Jeder Block verkörpert einen Punkt eines Dreiecks (Vertex-Index/Normalen-Index/Texturkoodinaten-Index). Nicht benötigte Elemente wie z.B. Texturkoordinaten können weggelassen werden. Es existiert keine Beschränkung der maximal verwendbaren Vertices pro Polygon, allerdings muss jede Fläche flach und konvex sein.

5 Design

5.1 Aufbau des Gesamtsystems

Als Ergebnis der Analyse wird der Prototyp nachstehenden Aufbau besitzen. Ein Oktonärbaum wird als Datenstruktur zur Abbildung der Voxel verwendet. Diese Struktur wird mittels einzelner Polygonwürfel visualisiert. Der Benutzer kann durch Maus und Tastatur mit diesem Volumenmodell interagieren. Innerhalb dieses Sculpturing-Prozesses werden im wesentlichen Würfel selektiert und entfernt. Das polygonal visualisierte Volumenmodell muss in eine Voxelrepräsentation überführt werden. Zu diesem Zweck wird die Raumbelegung des Oktonärbaums berechnet. Als Datenstruktur zur Speicherung der Voxel wird ein dreidimensionales Feld aus Ganzzahlen benutzt. Dieses Feld wird anschließend an den Marching Cubes Algorithmus zur Triangulation übergeben. Als Ergebnis wird eine Oberflächenrepräsentation, des durch den Benutzer erstellten Volumens, visualisiert. Diese Darstellung soll in das OBJ-Format exportiert werden können.

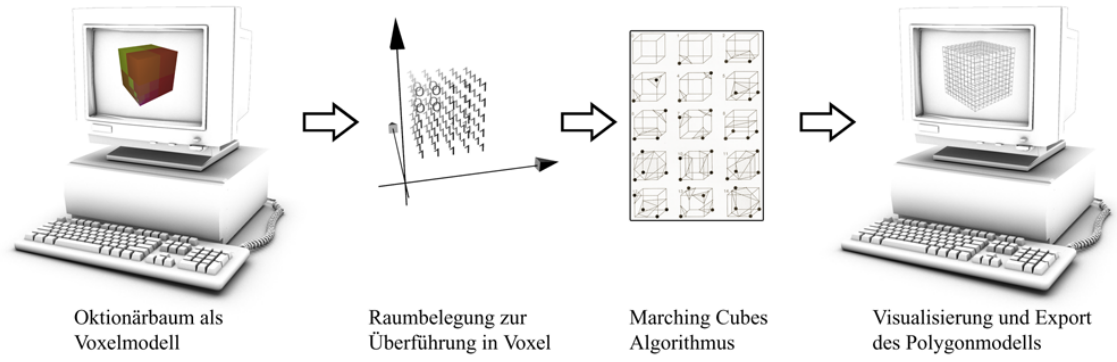


Abbildung 5.1: Aufbau des Gesamtsystems

5.1.1 Überführung des Modells in Voxeldaten

Ein erster Ansatz zur Überführung der Daten besteht in einer gerichteten Traversierung des Oktonärbaums. Dies erfordert das Durchlaufen des Raumes mit gleich-

zeitiger Speicherung eines Voxelfelds welche die Raumbelegung des Oktonärbaums wiederspiegelt.

Raumbelegung (*space occupancy enumaration*)

Die Raumbelegung stellt im Prinzip eine Datenstruktur dar, welche räumlich lokalisiert die Belegung bestimmter Positionen speichert. Als Verfahren wird folgende Herangehensweise vorgeschlagen. Als erstes werden alle Knoten des Baums, welche sichtbar sind in einer Liste gespeichert. Die Ausdehnung und Position des Baums und seiner Kinder ist bekannt. Anschließend wird ein Feld mit der gewünschten Ausdehnung mit Nullen gefüllt. Diese Ausdehnung entspricht später der Genauigkeit mit der der Baum abgetastet werden sollen. Die Position der einzelnen Knoten wird der Position innerhalb des Raumbelegungsfelds zugeordnet. Die Ausdehnung bestimmt anschließend den mit Einsen zu füllenden Bereich. Auf diese Weise ist es möglich den Oktonärbaum in eine Voxeldarstellung zu überführen (vgl. Abbildung 5.2). Problematisch zeigt sich einzig und allein die Überprüfung der Richtigkeit

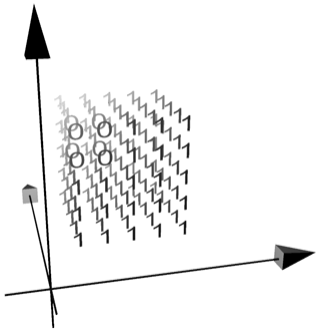


Abbildung 5.2: Raumbelegung: *space occupancy enumaration*

einer Implementierung dieses Algorithmus. Als Ergebnis wird ein dreidimensionales Feld mit Werten aufgebaut. Diese Datenstrukturen lassen sich nur schwer auf ihre Richtigkeit überprüfen ohne sie zu visualisieren. Auf dieses Problem wird im Kapitel 6.3.6 eingegangen.

5.1.2 DirectX API Sample Framework (DXUT)

Das Sample Framework DXUT (sample Framework DXUT 2006, Webresource) ermöglicht unter anderem den Zugriff auf Elemente einer grafischen Benutzerschnittstelle (Slider, Button, Textfield etc.). Dies führte zu der Entscheidung dieses Framework zu verwenden. Die DXUT stellt innerhalb des zu entwickelnden Portotypen

die View-Ebene einer MVC-Struktur[1] dar. Das Komponentendiagramm in Abbildung 5.3 verdeutlicht die Integration der DXUT in den Aufbau des Prototypen. Das Sample Framework wird dabei im wesentlichen als View-Komponente auf die Anwendung gesetzt. Durch diese Vorgehensweise wird gewährleistet, dass die wesentlichen Teile der Anwendung auch ohne die DXUT funktionsfähig sind. Es wird angestrebt, die Mehrzahl der Software-Objekte weitgehend generisch zu implementieren, um eine spätere Wiederverwendung dieser zu unterstützen.

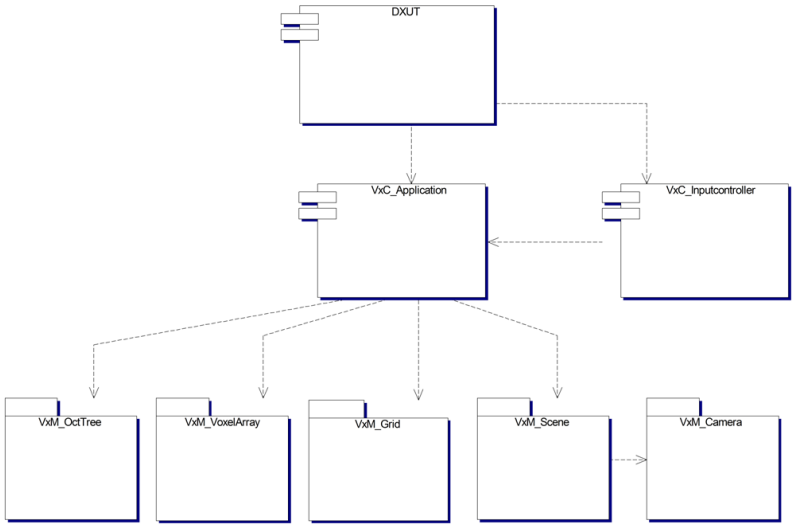


Abbildung 5.3: Komponenten des Gesamtsystems

5.2 Designmuster Einzelstück (Singleton)

Dieses Entwurfsmuster der Softwareentwicklung gehört zu den Erzeugungsmustern. Das Einzelstück (engl. Singleton) erzeugt genau eine Instanz zu einer Klasse und stellt einen globalen Zugriff auf dieses Objekt sicher (Wikipedia a, Webresource). Die Verwendung diese Designmusters ermöglicht einen schnellen unkomplizierten Zugriff auf alle Objekte, von denen nur eine Instanz benötigt wird. Beispielsweise ist es nicht notwendig, die Adressen steuernder Objekte innerhalb jeder Komponente als Membervariable zu speichern. Die Verwendung des Singleton Designmusters wird für den Oktonärbaum, die 3D-Scene als Model-Komponenten, sowie die eigentliche Applikation und den Inputcontroller als Control-Komponenten vorgesehen.

[1] engl. Model-View-Control (MVC) Software Architekturmuster zur Aufteilung von Softwaresystemen in Datenmodelle , Präsentation und Programmsteuerung.

5.3 Modellierung der Benutzerschnittstelle

In der Regel erhöht die Benutzerschnittstelle mit steigendem Funktionsumfang ihren Grad an Komplexität. Modellieren innerhalb dieses Systems bedeutet das entfernen von vorhandenen Knoten des Baums. Diese werden nicht wirklich entfernt, sondern lediglich nicht mehr gezeichnet. Im folgenden Abschnitt wird zunächst die Interaktion mit dem Voxelmodell erstellt. Danach ist die Entfernung von Knoten verschiedener Ausdehnung durch den Benutzer Teil der näheren Betrachtung. Abschließend werden die verschiedenen Aktivitäten der Visualisierungmodi mittels eines Diagramms verdeutlicht.

5.3.1 Interaktion mit dem Voxelmodell

Mit dem Begriff *Picking* wird in der 3D-Computergrafik die Auswahl eines Objektes oder seiner Untermengen bezeichnet. Um eine Interaktion mit dem polygonalen Voxelmodell herzustellen, wird eine Selektion von einzelnen Knoten des Oktonärbaums implementiert. Da es sich bei diesem Objektmodell um einen Baum handelt, muss diese Funktion rekursiv bis zu den Blättern arbeiten. Es wird ein Strahl senkrecht ausgehend von den Bildschirmkoordinaten der Maus in den Raum der 3D-Projektion ausgesendet. Mittels der Berechnung einer Intersektion des Oktonärbaums mit dem Strahl wird festgestellt, ob ein Objekt getroffen wurde. Dabei ermittelt die Intersektionsfunktion, der DirectX 9 API, die Entfernung des Treffers. Nun ist es möglich den Knoten zu ermitteln, dessen Entfernung zur Kamera das Minimum darstellt. Dieser Knoten wird als aktiver Knoten gespeichert bzw. grafisch kenntlich gemacht. Nach Erstellung der Funktionalität des Picking zeigte sich ein sehr flexibles Ver-

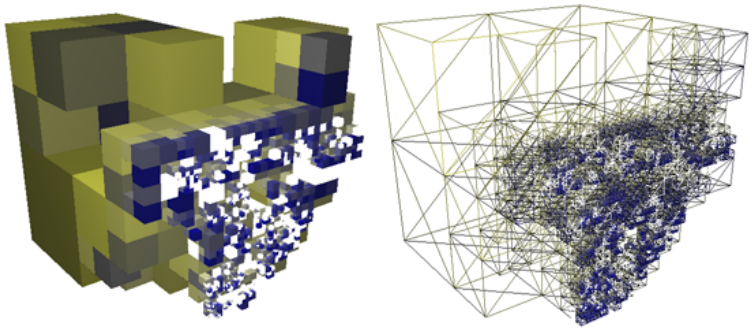


Abbildung 5.4: Interaktion mit der polygonalen Darstellung des Voxelmodells.

halten des Prototypen. Die Abbildung 5.4 verdeutlicht die Leistungsfähigkeit ohne Verwendung von Strategien zur Verdeckung einzelner Elemente. Damit der Benutzer das Voxelmodell einfach gestalten kann, sollte das System zumindest Knoten

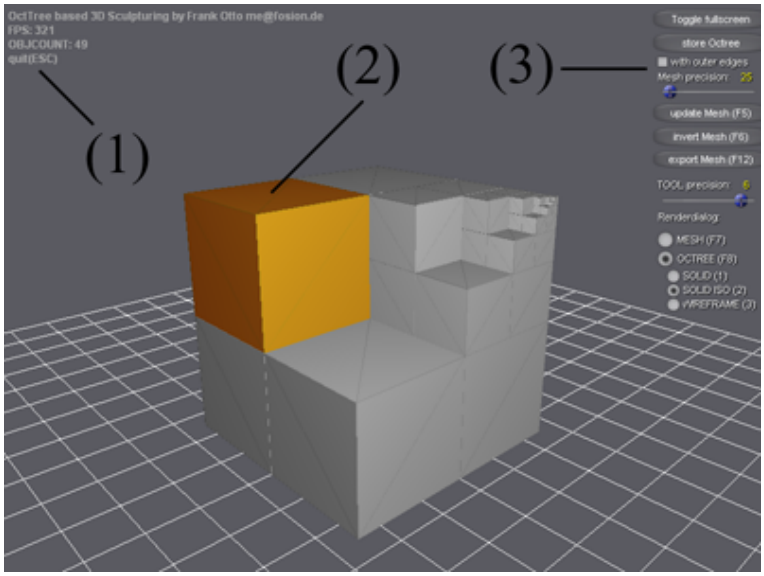
von verschiedener Ausdehnung aus dem Baum entfernen können. Im Kapitel 6.3.3 auf Seite 50 werden die Besonderheiten sowie die einzelnen Stufen der Entwicklung während der Implementierung dargestellt.

5.3.2 Gestaltung des GUI

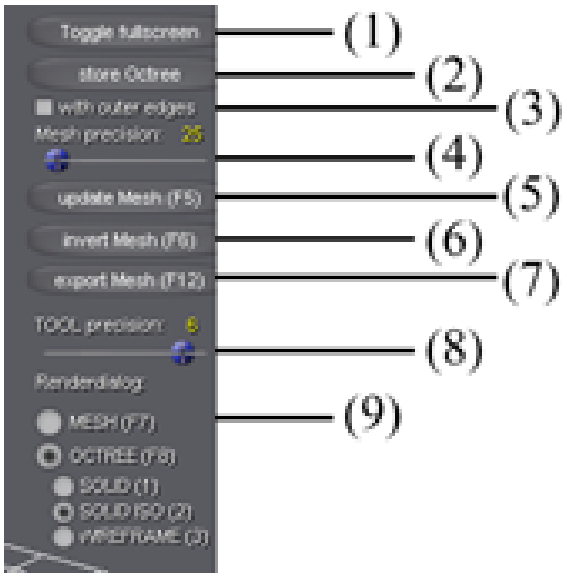
Die Grafische Benutzer Schnittstelle (GUI) ist in drei Bereiche untergliedert. Die Abbildung 5.5a verdeutlicht diese Aufteilung. Der erste Bereich wird zur Ausgabe von Status Meldungen des Prototypen angewandt. Der zweite Bereich ist, bildlich ausgedrückt, die Bühne auf der das Sculpturing stattfindet. Das Gitter symbolisiert dabei den Boden auf dem der Benutzer sich bewegt. Die Selektion eines Knoten wird durch die Mausbewegung ausgelöst, wenn sich die Maus über dem Wurzelknoten des Voxelmodells befindet. Ein selektierter Knoten wird orangenfarbig hervorgehoben. Die Kameraposition rotiert unter Zuhilfenahme der rechten Maustaste um einen Punkt vor der Kamera. Die mittlere Maustaste (Mausrad) aktiviert die Translation der Kameraposition entlang ihrer X- bzw. Y-Achse (*Strafing*). Die Drehung des Mausrads bewegt die Kamera vor oder zurück. Die Linke Maustaste dient dem Entfernen von Knoten aus dem Voxelmodell.

5.3.3 Aktivität der einzelnen Visualisierungsformen

Das Aktivitätsdiagramm in Abbildung 5.6 auf Seite 46 beschreibt die Aktivität der Visualisierung des Oktonärbaums bzw. des triangulierten Gittermodells. Dabei wird zwischen DXUT GUI Ereignissen und Modellierungs- bzw. Kamera-Ereignissen unterschieden.



(a) Die Gesamtansicht ist in drei Breiche geteilt. (1) Anzahl der Objekte im Voxelmodell sowie die Bildwiederholrate; (2) Farbliche Kennzeichnung des Aktiven Knoten des Voxelmodells; (3) Benutzerschnittstelle



(b) Die Positionen (1)-(9) der Benutzerschnittstelle werden in der Tabelle 5.1 auf der nächsten Seite erläutert.

Abbildung 5.5: Gestaltung der Grafischen Benutzer Schnittstelle

Tabelle 5.1: Beschreibung der Benutzerschnittstelle des Prototypen (vgl. Abbildung 5.5b)

Nr.	**Funktion**
(1)	Umschalten zwischen Vollbildmodus und Fenstermodus
(2)	Abspeichern des Voxelmodells; der Aufbau der Datei wird in Kapitel 6.3.2 auf Seite 48 beschrieben.
(3)	Wenn der Modus *with outer edges* aktiviert ist, wird vom Marching Cubes an jede 90 Grad Kante eine 45 Grad Fase interpoliert.
(4)	Mithilfe dieses Schiebers lässt sich die Genauigkeit, mit der der Oktonärbaum vom Marching Cubes Algorithmus abgetastet werden soll, einstellen. Der Wert 25 entspricht 25^3 Voxel.
(5)	Der Button *update Mesh* löst die Abtastung und anschließende Triangulation des Octree aus. Nach erfolgter Berechnung schaltet der *Renderdialog (9)* auf *MESH*.
(6)	Der Button *invert Mesh* invertiert den Octree. Abhängig vom Renderdialog wird entweder der Octree invertiert oder zusätzlich ein neues Gitter generiert.
(7)	Der Button *export Mesh* speichert eine OBJ-Datei des Gittermodells.
(8)	Mit diesem Slider wird die Größe der zu entfernenden Voxel eingestellt. Der Wert entspricht der Tiefe der Knoten des Octree.
(9)	Der Renderdialog legt fest welche Form der Visualisierung aktiv sein soll. Die beiden grösseren Radio-Button schalten zwischen der Darstellung des Octree und des Gittermodells hin und her. Mit den darunter liegenden Button wird zwischen den einzelnen Schattierungsmethoden gewechselt. Der Modus *SOLID* entspricht Gauraud Shading. Bei *SOLID ISO* wird zusätzlich das Wireframe-Model über dem *SOLID* Model gezeichnet.

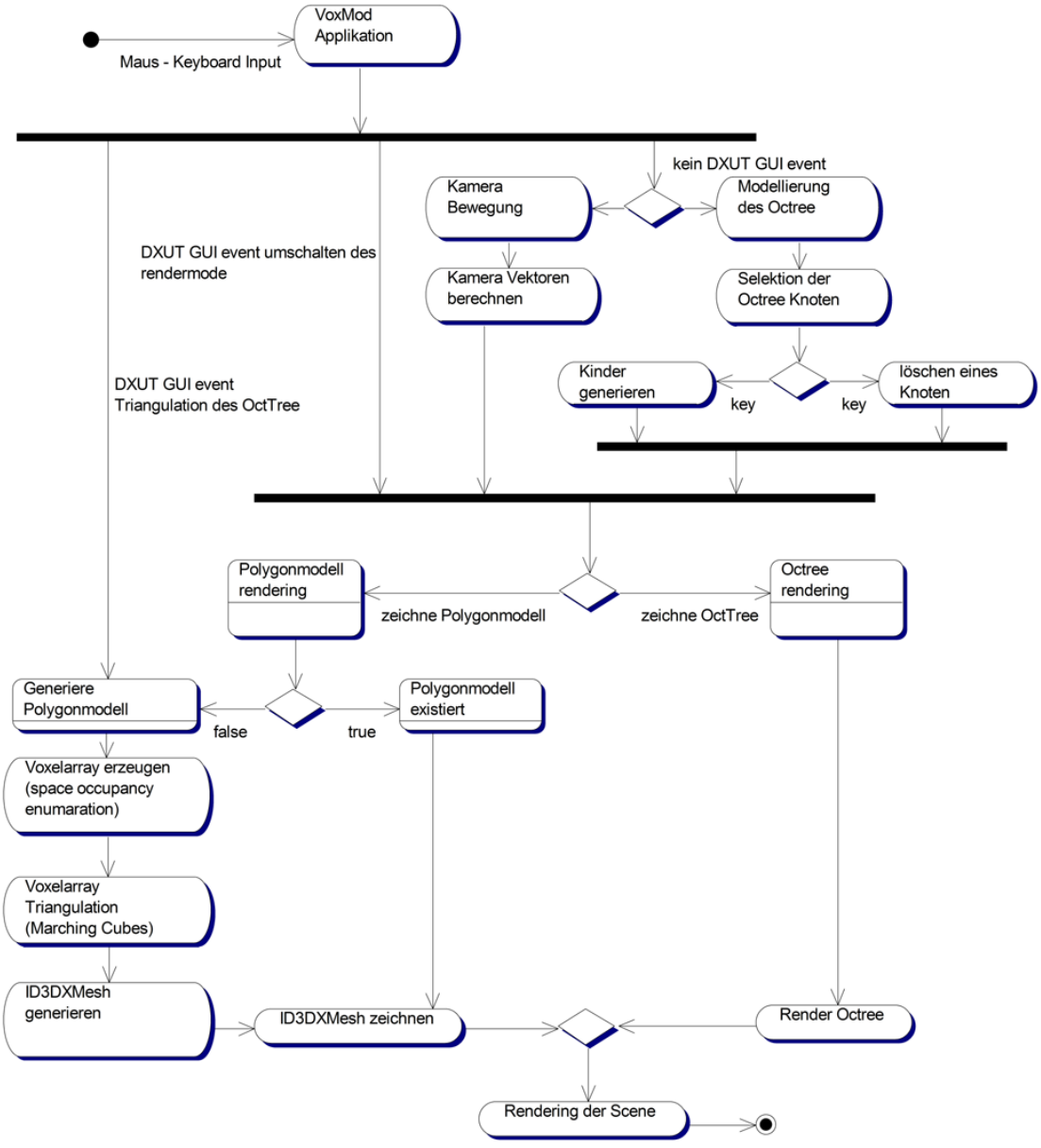


Abbildung 5.6: Aktivitätsdiagramm der Modi der Visualisierung

6 Implementierung

6.1 Vorgehensweise

Die Entwicklung des 3D-Modelling-Systems erfolgte schrittweise. Die Vorgehensweise ist dem explorativen Prototyping zuzuordnen. Dabei wurde auf die Verwendung klassischer Vorgehensmodelle wie beispielsweise dem V-Modell, Wasserfallmodell oder RUP (Rational Unified Prozess) verzichtet. Diese Form des Prototyping wird im Allgemeinen zur Beurteilung der angestrebten Problemlösungen eingesetzt. Dabei liegt der Schwerpunkt auf der Erstellung der Funktionalitäten des Systems. Das Ziel ist die Bestätigung der Tauglichkeit der Spezifikation (Wikipedia c, Webresource).

6.2 Architektur der Anwendung

Während der Entwicklung des Prototypen wurde grundsätzlich eine Model-View-Control Architektur (MVC) angestrebt. Zum Zeitpunkt der Implementierung zeigte sich jedoch, dass durch die Verwendung des DirectX 9 API Sample Framework (DXUT) diese Struktur nicht stringent eingehalten werden kann. Das Framework wird ähnlich der GLut (OpenGL) von einem C-Programm ausgeführt. Zunächst werden Callback-Methoden wie z.B. Messageprocessing, Rendering und Deviceevents bei der DXUT registriert. Anschließend wird durch den Aufruf der Funktion *DXUTMainLoop()* in den *mainloop* eingetreten.

Die einzelnen Modellkomponenten, welche Teile zur Visualisierung beisteuerten, wurden jeweils um eine *draw()* Methode ergänzt. Die Klasse VxC_Application steuert die Anwendung. Hierbei werden die Kernfunktionalitäten, wie die einzelnen Visualisierungsmodi, das Picking, die Triangulation, sowie der Export des OBJ files, von dieser Klasse ausgelösst. Der VxC_InputController übernimmt die Abarbeitung der Maus- und Keyboardeingaben. Durch die Verwendung des Singleton-Designmusters war es möglich, alle von der DXUT ausgelösten GUI-EVENTS durch statische Objektaufrufe direkt an die beteiligten Klassen zu übergeben. Diese Herangehensweise erbrachte eine Erleichterung des Prototyping. Es konnte zu Testzwecken schnell programmweit auf die wesentlichen Elemente zugegriffen werden.

6.3 Realisierung/Umsetzung

6.3.1 Refactoring

Mittels der Refactoringmöglichkeiten, welche das *Borland Together Control Center* bietet, war es auf einfache Weise möglich, eine Namensgebung auf Klassenebene einzuführen. Alle Controlkomponenten wurden mit dem Präfix *VxC* versehen. Modelkomponenten erhielten das Kürzel *VxM* und alle Hilfsfunktionen (utilities) *VxU*. Da moderne, integrierte Entwicklungsumgebungen (IDE)[1] Quellcodevervollständigung unterstützen, hatte die Einführung dieser Präfixnotation eine Erleichterung der Programmierung zur Folge.

Um das Verständnis für den Quellcode und die Aufrufstrukturen der Anwendung zu unterstützen, wurden mit Hilfe der Re-Engineering-Fähigkeiten von Together Klassendiagramme erzeugt. Zusätzlich wurden ausgewählte Sequenzen in der Form von Sequenzdiagrammen aufbereitet. Dazu gehören die Kernfunktionalitäten des Prototypen, wie die Initialisierung der Anwendung, die Auswahl eines Knoten und die Triangulation des Oktonärbaums. Die Sequenzendiagramme und die zugehörigen Klassendiagramme befinden sich im Anhang A.1 - A.6 auf den Seiten 68 bis 73.

6.3.2 Implementierung des Oktonärbaums

Die Basis Klasse des Oktonärbaums bildet die Datei "VxM_OctTree.h". Innerhalb der Klasse VxM_OctTreeNode, welche die einzelnen Knoten des Oktonärbaums verkörpert, wurde u.a. die Position, Kantenbreite und die Tiefe des jeweiligen Knoten gespeichert. Als dreidimensionale Objekte wurde ein DirectX ID3DXMESH Objekt verwendet. Das ID3DXMESH Objekt verfügt über einen Vertex- und Index-Buffer, sowie einen Adjazenz-Graphen, der die Nachtbarschaft der einzelnen Polygone abbilden kann. Zur einfacheren Verwendung dieser Objekte wurde die Klasse VxM_SceneObject implementiert. Sie kapselt das ID3DXMESH innerhalb eines Objekts. Zusätzlich wird eine Matrixtransformation, welche die Transformation der Modellkoordinaten des ID3DXMESH erlaubt, unterstützt. Des Weiteren werden Materialeigenschaften, die das Erscheinungsbild eines 3D-Objektes bestimmen gespeichert (vgl. Klassendiagramm Abbildung A.1 auf Seite 68). Die Bibliothek *d3dx9shape.h* stellt Basisfunktionen zur Arbeit mit DirectX MESH Objekten zur Verfügung. Diese Container werden nicht mit dem New-Operator erzeugt, stattdessen kommen geeignete Erzeugungsmethoden wie z.B. D3DXCreateBox(...) oder D3DXCreateMeshFVF(...) zum Einsatz. Letztere erzeugt einen Container mit einem vom Programmierer bestimmten Vertexformat (Flexible Vertex Format FVF).

[1] engl. integrated development environment (IDE)

Als Ergebnis dieses Prozedere verfügte jeder Knoten des Baums über ein 3D-Objekt, welches die Abmessung des Knoten verkörpert und im Speicher der Grafikkarte lokalisiert ist. Dieser Aufbau ermöglichte die Verwendung der Funktion D3DXIntersect(), die eine Intersektion mit ID3DXMESH Objekten berechnet.

Zur Indizierung der einzelnen Knoten wurde die von Alan Watt vorgeschlagene Struktur eingesetzt (vgl. Abbildung 2.4 auf Seite 9). Um einen Identifikator (ID) dieser Struktur zu erstellen, gibt es sicherlich eine sehr große Anzahl von Lösungen. Vorteilhaft erschien diese ID in einer Integervariablen zu speichern. Wenn man jedem Knoten in Abhängigkeit seiner Position innerhalb seines Vaterknotens eine Zahl zwischen 1 und 8 zuweist, dann lässt sich folgende Vorschrift zur Indizierung ableiten:

```
ObjektID = ID des Vaterknoten * 10 + Nummer des Kindes
```

Zur persistenten Speicherung in einer Datei, wird der Baum traversiert und alle Identifikatoren in einer Liste gespeichert. Zusätzlich wird gespeichert, ob der jeweilige Knoten Kinder hat oder sichtbar ist. Um dieses zu realisieren, wird während der Traversierung abwechselnd eine ObjektID und eine Kennzeichnung des Zustandes dieses Objektes in eine Liste geschrieben. Zur Speicherung des gesamten Baums werden alle Identifikatoren in eine Textdatei gespeichert. Der Aufbau einer solchen Datei ist am Folgenden Beispiel erläutert.

```
1 #
11 #
111 -
112 -
113 -
```

Die 1 entspricht dem Wurzelknoten des Baums. Das Zeichen "#" sagt aus, dass der Knoten Kinder hat. Daraus folgt wiederum, dass er selbst nicht sichtbar ist. Das Zeichen "-" bedeutet, dass ein Knoten nicht sichtbar ist. Insgesamt enthält diese Datei demnach folgende Anweisungen:

1. Verfeinere den Wurzelknoten (1)
2. Verfeinere das 1. Kind des Wurzelknotens (11)
3. Die Kinder 111 112 und 113 des Knoten 11 sind nicht sichtbar

Bei der Speicherung des Voxelmodells entsteht eine Datei mit dem Namen VXMOD_octreedata.txt. Zur Verhinderung des versehentlichen Überschreibens wurde dem Dateinamen das Datum und die Uhrzeit der Erzeugung hinzugefügt.

6.3.3 Benutzerinteraktion

Die Implementierung einer rekursiven Picking-Funktion stellte eine unerwartete Herausforderung dar. Es galt mehrere Bedingungen innerhalb einer rekursiven Funktion abzufragen. Gesucht wurde der Knoten, welcher sichtbar ist, vom Picking Strahl getroffen wurde und dabei von allen getroffenen Objekten die geringste Entfernung zum Viewport besitzt. Sobald für einen gegebenen Knoten Kinder generiert werden, ist dieser nicht mehr sichtbar und darf nicht mehr selektiert werden.

Die Software 3D-Objekte VxM_SceneObject werden mittels Matrixtransformation in den Weltkoordinatenraum transformiert. Um diese Objekte nun korrekt auswählen zu können, muss der Strahl, der zur Auswahl in den Raum geschossen wird, zunächst in die Modellkoordinaten übersetzt werden. Eine sehr gute Quelle zur Erläuterung der Vorgehensweise stellt Robert Dunlop´s Webseite X-Zone dar. Er gehört zu dem Microsoft Most Valuable Professionals(MVP) Programm (Dunlop 2002a, Webresource).

Stufen der Entwicklung

Nachstehend werden die einzelnen Schritte der Entwicklung der Benutzerinteraktion bis hin zu ihrer jetzigen Form beschrieben. Zu Allererst wurde die Kamera durch die Klasse VxM_Camera implementiert. Eine generell sehr aufschlussreiche Quelle im Bereich der DirectX-Programmierung stellt die Webseite Toymaker.info dar (Ditchburn 2006b, Webresource). Es werden eine große Anzahl von Tutorien angeboten, welche sich nicht ausschließlich mit der Grafikprogrammierung auseinandersetzten. Dazu gehört auch eine Sektion, die sich mit den Grundlagen der Programmierung einer Kamera beschäftigt. Teile dieser Anleitung wurden mit kleinen Änderungen übernommen (Ditchburn 2006a, Webresource). Die rechte Maustaste löst die Rotation um einen Punkt vor der Kamera aus (3rd Person). Mit dem mittleren Mausbutton (Mausrad) wurden das sogenannte *Strafing* ausgelösst und das Mausrad kontrolliert die Vorwärts bzw. Rückwärtsbewegung der Kamera.

Die Verfeinerung und das Entfernen[2] von Knoten waren anfänglich noch entkoppelt. Es erforderte jeweils eine Aktion des Benutzers, entweder zu verfeinern oder ein Element zu entfernen. Die eigentliche Auswahl eines Knoten wurde anfänglich mit der linken Maustaste ausgelöst. Die Tasten "R" und "Z" dienten zum Verfeinern bzw. Entfernen des selektierten Knotens. Die Anforderung des Entfernens von Knoten mit verschiedener Ausdehnung wurde durch diese Entwicklungsstufe allerdings noch nicht erreicht. Es war zwar möglich erst zu verfeinern und dann den gewünschten

[2]Entfernen in diesem Zusammenhang bedeutet lediglich, dass der betreffende Knoten nicht mehr gezeichnet wird.

Knoten zu entfernen, nur wird sich dieses Konzept mit Sicherheit dem Benutzer nicht intuitiv erschließen.

Der Benutzer sollte per GUI die Größe der Elemente, welche entfernt werden, auswählen können. Zu diesem Zwecke wurde der Oktonärbaum um eine Variable, welche die Eigenschaft der maximalen, zugelassenen Tiefe des Baums verkörpert, erweitert. Dieser Wert wurde mit einem Schieber innerhalb der grafischen Benutzerschnittstelle verbunden (vgl. Abbildung 5.5b (8)). Dabei sind ganzzahlige Werte zwischen 1 und 8 einstellbar. Der Wert ergibt die maximale Tiefe, die innerhalb des Baums verfeinert werden soll, wobei der Wert 8 einer Anzahl 256^3 Voxel entspricht (vgl. Kapitel 3.3.5 auf Seite 32).

Da bisher die linke Maustaste die Selektion auslöst, musste zunächst die Selektion eines Knotens modifiziert werden. Sinnvoll erschien die automatische Selektion, sobald die Mausposition innerhalb des Bereichs des Voxelmodells liegt. Um dieses Verhalten zu realisieren wurde die Picking-Routine modifiziert. In der Methode VxC_Applikation::pick() wird nun zunächst nur einmal gepickt, um festzustellen, ob der Mauszeiger über dem Voxelmodell liegt. Nur wenn dies der Fall ist, wird rekursiv der Baum zur Ermittlung des genauen Treffers prozessiert. Gekoppelt mit dem Systemereignis der Mausbewegung (Mousemove) ergab sich eine sehr flexibles Verhalten. Die Knoten wurden automatisch selektiert.

Als Folge der automatischen Selektion traten Probleme mit der Kamerasteuerung auf. Problematisch stellte sich folgender Fall dar. Der Benutzer führt den Mauszeiger, während einer Kamerabewegung über das Voxelmodell. Durch die Auslösung des Picking verruckelte die Kamerabewegung. Deshalb wurde diese und die Selektion eines Knotens entkoppelt. Da Kamerabewegungen ebenso "Mousemove" Ereignisse erfordern, konnte nicht einfach mit dem Ereignis "MouseDownRight" gearbeitet werden. Alle Mausereignisse wurden stattdessen an die Methode processMouseInput() der Klasse VxC_InputController übergeben. Den Zustand des Mausbuttons ermittelt diese Klasse über die DirectInput-Schnittstelle. Zunächst werden alle Kameraereignisse abgearbeitet und nur, wenn es sich um kein Kameraereignis handelt, wird die Selektion ausgelöst.

Elemente mit verschiedener Ausdehnung zu entfernen, bedeutet logisch betrachtet, abwechselnd die Auswahl eines Knotens des Baums und die anschließende Verfeinerung dieses Knotens. Wenn die gewünschte Ausdehnung erreicht ist, wird der Knoten entfernt. Dieses Verhalten wurde durch ein while() Konstrukt realisiert.

```
Pseudocode:

while(Tiefe des selektierten Knoten < gewünschte Tiefe )
{
```

```
verfeinere den selektierten Knoten;
starte neue Selektion;
}
entferne den selektierten Knoten;
```

Nun können Elemente verschiedener Ausdehnung einfach durch Anklicken entfernt werden. Bei gleichzeitigem Bewegen der Maus zeigte der Prototyp ein überraschend flexibles Verhalten. Die Interaktion mit dem Voxelmodell vermittelt ein Gefühl von Sculpturing im Sinne von malerischem Gestalten eines Voxelobjektes. Da diese Form der Objektselektion in Abhängigkeit von der Mausbewegung ausgeführt wird, sollte an dieser Stelle ein Timer, der die Anzahl der Selektionen begrenzt, eingebaut werden. Auf diese Weise würde verhindert, dass ein wesentlich schnellerer Computer das Werkzeug zu einer "Laserkanone" macht, die sofort das Voxelmodell durchdringt.

Der Prototyp verfügte zu diesem Zeitpunkt nicht über die Möglichkeit, Voxel hinzuzufügen. Während der Implementierung der Speicherung des Voxelmodells entwickelte sich der Gedanke, dass die Umkehrung des Baums mit einer simplen Rückgängig-Funktion gleichzusetzen ist. Da ein invertierter Baum das genaue Gegenteil des Ausgangsmodells darstellt, erlaubt diese Funktionalität auf einem Umweg das Hinzufügen bereits entfernter Voxel. Man invertiert den Baum und entfernt den Teil, der später wieder da sein soll. Nach nochmaligem Invertieren sind die zuvor entfernten Elemente wieder vorhanden. Nach Erstellung dieser Funktionalität traten allerdings Probleme mit dem Speichern des Voxelmodells auf (siehe Kapitel 8.3.2 auf Seite 61).

6.3.4 Marching Cubes Algorithmus

Das Projekt *immersive Visualisierung medizinischer Daten*, welches im Projektsemester (ws2005) des Studienganges Angewandte Informatik (FB4/FHTW) von Herrn Prof. Dr. Thomas Jung betreut wurde, beschäftigte sich mit dem Einlesen und immersiven Visualisieren von DICOM[3] Datensätzen. Ich hatte das Glück, Teil der Projektgruppe zu sein. Meine Aufgabe umfasste die Programmierung eines Volumenrenderers in C++ OpenGL, sowie die Zusammenführung aller von den Gruppenmitgliedern erstellter Programmkomponenten. Unter anderem verwendete dieses Projekt eine Implementierung des Marching Cubes Algorithmus von Paul Bourke (Bourke 1994, Webresource). Bourke´s Quellcode wurde dabei lediglich in eine Klasse umgeschrieben, der eigentliche Algorithmus blieb unberührt. Aus diesem Grunde wird diese Klasse zur Erstellung des Prototypen wiederverwendet. Eine eigene Implementierung des Marching Cubes Algorithmus wird nicht vorgenommen. Das Sequenzdiagramm in Abbildung A.5 auf Seite 72 verdeutlicht die

[3]Digital Imaging and Communications in Medicine

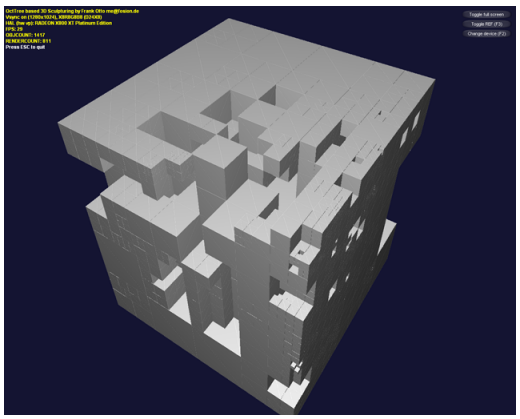
Verwendung dieser Klasse. Zunächst wird das Objekt, welches die Raumbelegung (VxM_VoxelArray) verkörpert, erzeugt. Anschließend wird dieses an den Konstruktor der Klasse VxU_Grid (Marching Cubes) zur Triangulation übergeben. Nach erfolgter Triangulation verfügt die Instanz der Klasse VxU_Grid über ein dreidimensionales Objekt (VxM_SceneObject).

6.3.5 Triangulation des Voxelmodells

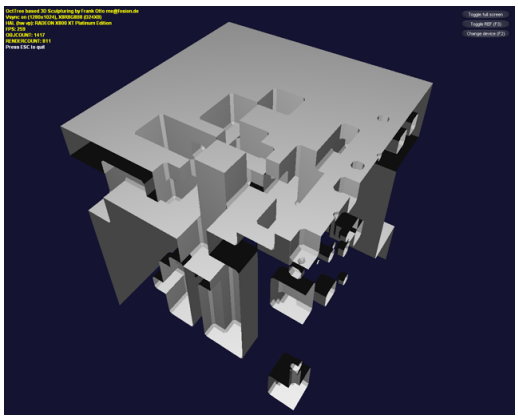
Die Triangulation arbeitete zunächst nicht wie erwartet. Bei Verwendung eines einzigen Elementes, dem Wurzel Knoten des Baums, hätte die Triangulation als Ergebnis ein Würfel aus Polygonen ergeben müssen. Stattdessen wurden überhaupt keine Dreiecke generiert. Anscheinend werden nur Dreiecke erzeugt, wenn das Voxelfeld nicht voll besetzt ist (je mit dem Wert '1.0'). Um dieses Problem zu umgehen, wurde kurzer Hand die Übergabe der Werte an den Marching Cubes umgebaut. Die Abbildung 6.1 auf der nächsten Seite verdeutlicht das Ergebnis bei Erweiterung des Voxelfeldes um den Wert 1 pro Achse. Die erweiterten Bereiche wurden mit Nullen aufgefüllt. Als Folge der Erweiterung besitzen drei Seiten des Gittermodells Polygone an den Außenkanten. Es ist deutlich zu erkennen, dass im vorderen Bereich des Modells keine Dreiecke generiert wurden. Zur Lösung des Problems wurde der Index des Feldes, welches intern vom Marching Cubes abgelaufen wird, um den Wert 2 erweitert. Die äußeren Grenzen des Feldes werden mit Nullen aufgefüllt. Durch diese Verfahrensweise ist das entstehende Gittermodell etwas kleiner als das Voxelmodell. Um diesen Effekt möglichst klein zuhalten, wird das generierte Gittermodell während der Ausgabe etwas nach oben skaliert.

Nach Integration der Triangulation in die grafische Benutzerschnittstelle zeigte sich schnell, dass diese bei einer Breite von 30 Voxel pro Achse nahezu unmittelbar erfolgt. Die Interaktion mit dem Voxelmodell kann prinzipiell ohne eine Darstellung dieses Modells erfolgen. Schließlich sind alle 3D-Objekte des Oktonärbaums innerhalb der Grafikkarte gespeichert und werden deshalb auch selektiert vorausgesetzt. Es wurde ein Objekt getroffen. Aus dieser Erkenntnis heraus, wurde innerhalb der Ansicht des Polygonmodells ein automatisches Triangulieren eingeführt. Sobald der Benutzer den linken Mausbutton loslässt, wird eine erneute Triangulation ausgeführt. Erst, wenn mehr als 100 Voxel pro Achse eingestellt sind, muss die Trangualtion explizit vom Benutzer ausgelöst werden. Das Gefühl, welches von diesem Modus vermittelt wird, gleicht dem Meißeln in Stein. Es bedarf jedoch ein wenig Übung, da die Triangulation nur nach dem Loslassen der Maustaste aktiviert wird. Dabei wurde bereits mit mehr als 20.000 Objekten innerhalb des Oktonärbaums gearbeitet[4].

[4]Innerhalb der Ansicht des Voxelmodells (OCTREE) werden auf dem Zielsystem lediglich bis zu 3800 3D-Objekte in Echtzeit visualisiert (vgl. Kapitel 3.3.3 auf Seite 29).



(a) Voxelmodell, welches zum Test des Marching Cubes verwendet wurde.



(b) Ergebnis der Triangulation des Marching Cubes Algorithmus.

Abbildung 6.1: Probleme mit dem Marching Cubes Algorithmus: Abbildung (b) verdeutlicht, dass im vorderen Bereich des Modells keine Dreiecke generiert wurden.

Im Kapitel 8.4.2 auf Seite 63 wird ein Ausblick auf Ansätze zur Echtzeitintegration des Marching Cubes gegeben.

6.3.6 Überprüfung der *Space occupancy enumaration*

Zur Speicherung der Voxeldaten wurde ein eindimensionales Feld aus Integerwerten verwendet. Die drei Dimensionen des Voxeldatensatzes wurden mittels nachstehender Vorschrift in diesem Feld abgebildet.

```
x + ( y * dimXYZ ) + ( z * dimXYZ * dimXYZ )
```

Das Voxelfeld soll eine gleiche Ausdehnung in X, Y und Z Achse besitzen. Deshalb wurde als Offset der Wert dimXYZ verwendet, welcher der Ausdehnung des Voxelfeldes entspricht. Dieses Mapping von drei Dimensionen auf eine birgt in sich einige Probleme, was die Überprüfung der Richtigkeit der enthaltenen Werte betrifft. Zum Zwecke der Überprüfung wurden die abgetasteten Werte an den Marching Cubes übergeben. Dabei wurde ein Oktonärbaum, welcher aus acht Elementen bestand, verwendet. Eines dieser Elemente war nicht sichtbar. Dieses Modell wurde mit einer Genauigkeit von 10 Voxel pro Achse abgetastet. Die verwendete Implementierung traversiert die übergebenen Voxeldaten in allen drei Achsen. An dieser Stelle des Programms wurde mithilfe von Debugging zur Laufzeit ein kleiner Datensatz überprüft.

7 Test

7.1 Modellierung

Im Folgenden Abschnitt werden mit dem Prototypen durchgeführte Tests erläutert. Anhand von Abbildungen wird die Leistungsfähigkeit des 3D Modelling Systems dargestellt. Die Nachstehenden Abbildungen wurden zum Zwecke der besseren Illustration freigestellt [1]. Sonstige Veränderungen, wenn nicht explizit angegeben, sind nicht vorgenommen worden. Die Abbildungen 7.1a-h auf der Seite 56 zeigen zwei Testmodelle. Diese Modelle wurden mit verschiedenen Genauigkeiten abgetastet und trianguliert. Mit steigender Präzision nähert sich das Polygonmodell der Form des Voxelmodells an.

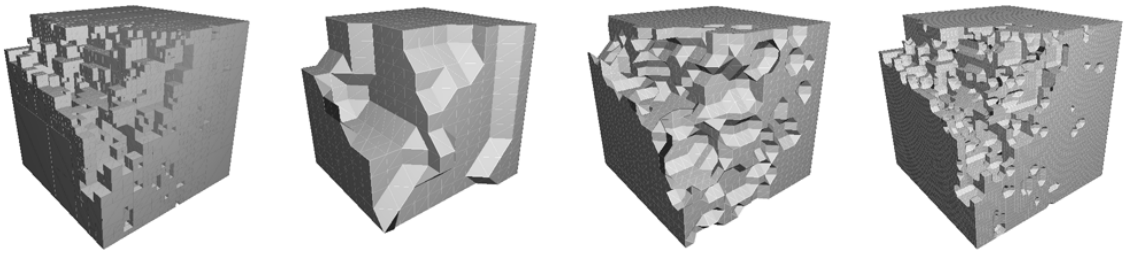
7.1.1 Neue Materialeigenschaft

Während der Arbeit am Marching Cubes wurde die Kanteninterpolation um einen zweiten Modus ergänzt. Die Checkbox 'with outer edges' innerhalb der Benutzeroberfläche aktiviert diesen Modus (vgl. Kapitel 5.3.2 auf Seite 43). Damit verfügt die Triangulation über eine Art zweite Materialeigenschaft. Diese hat zum Ergebnis, dass alle Kanten durch 45 Grad-Fasen gebrochen werden. Außerdem sind alle Vertiefungen innerhalb des Modells zwei Voxel kleiner. Die Abbildungen 7.2a-h auf Seite 57 zeigen beide Modi bei verschiedener Genauigkeit der Abtastung des Voxelmodells einer einfachen Teekanne. Die oberen Bilder der Abbildungen 7.2b-h zeigen jeweils die ursprüngliche Variante, die unteren die Modifikation.

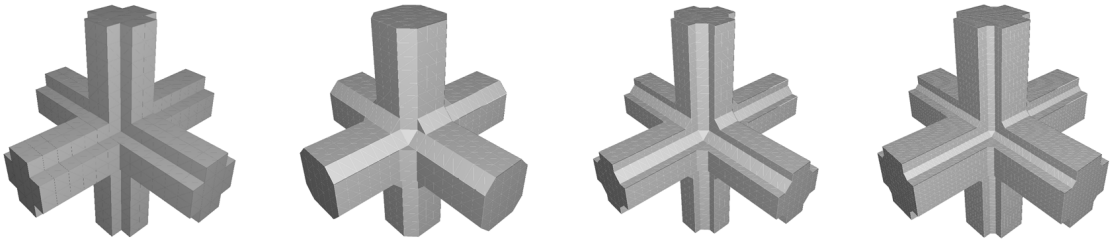
7.1.2 Invertierung führt zu neuer Idee

Die Erstellung der Grafiken zur Demonstration der Invertierungsfunktion regte die Idee zum Bruch von Modellen an. Im Kapitel 8.4.3 auf Seite 63 wird der Bruch eines 3D-Modells mittels der vom Prototypen implementierten Funktionalitäten erläutert. Die Abbildungen 7.3a-d auf der Seite 56 verdeutlicht die Möglichkeiten dieser Idee.

[1] Die original Screenshots sind auf dem der Arbeit beigefügten Datenträger enthalten.



(a) Voxelmodell (b) Triangulation von 10^3 Voxel (c) Triangulation von 20^3 Voxel (d) Triangulation von 40^3 Voxel

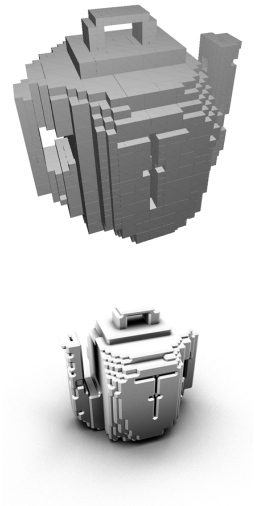


(e) Voxelmodell (f) Triangulation von 20^3 Voxel (g) Triangulation von 30^3 Voxel (h) Triangulation von 50^3 Voxel

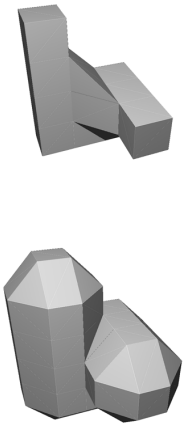
Abbildung 7.1: Test der Triangulation: Abtastung des Voxelmodells mit variierender Genauigkeit

7.2 OBJ-Export

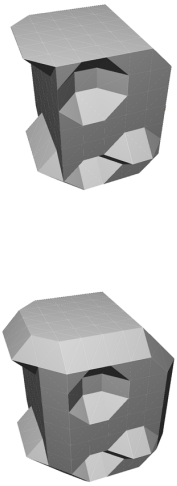
Während des Imports in 3D Studio Max wurden einige Fehler entdeckt. Die Normalen des importierten Objektes schienen nach innen zu zeigen. Logisch betrachtet konnte dies eigentlich nicht möglich sein, da der Prototyp das Polygonobjekt sauber darstellt. Bei Invertierung der Normalen brachte der Import das selbe Ergebnis. Die Abbildung 7.2a auf Seite 57 zeigt im unteren Bereich ein Rendering des exportierten OBJ Files. Während der Zusammenstellung dieser Grafiken wurde schnell klar, dass die Teekanne spiegelverkehrt ist. Dies würde auch die fehlerhaften Normalen erklären. Wenn alle Koordinaten spiegelverkehrt sind, dann stimmen auch die generierten Normalen nicht mehr. Zur Lösung des Problems wurden die Vorzeichen der Koordinaten des Modells umgekehrt. Der Import in 3D Studio Max zeigte nun ein kopfstehendes, aber dafür korrektes, Polygonmodell. Sicherlich sollte dieses Problem besser an seiner Wurzel bekämpft werden, jedoch ist die zur Verfügung stehende Zeit begrenzt. Aus diesem Grunde muss für den Prototypen dieser Workaround genügen. Ein weiteres Problem wurde bei der Überprüfung der Anzahl der generierten Dreiecke aufgedeckt. Es werden anscheinend zu viele Dreiecke generiert bzw. doppelte Vertices generiert (siehe Kapitel 8.3.1 auf Seite 61).



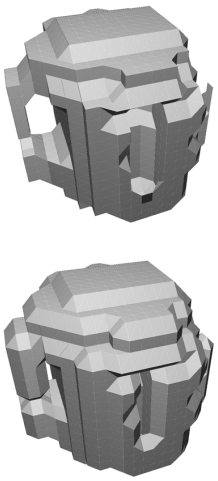
(a) Voxelmodell einer einfachen Teekanne(oben) Rendering des OBJ Exports(unten)



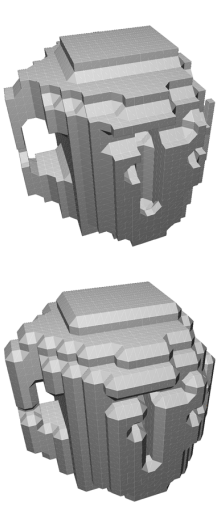
(b) Triangulation der Teekanne bei 5^3 Voxel



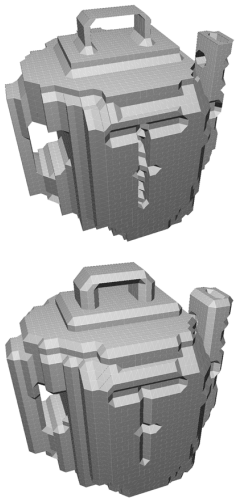
(c) Triangulation der Teekanne bei 10^3 Voxel



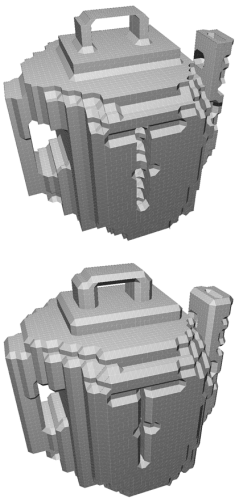
(d) Triangulation der Teekanne bei 20^3 Voxel



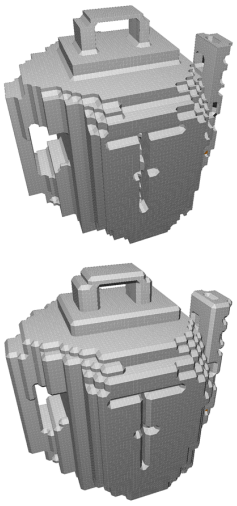
(e) Triangulation der Teekanne bei 30^3 Voxel



(f) Triangulation der Teekanne bei 40^3 Voxel

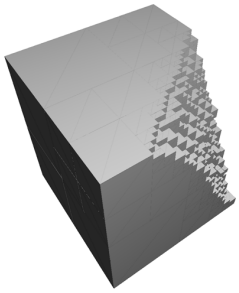


(g) Triangulation der Teekanne bei 50^3 Voxel

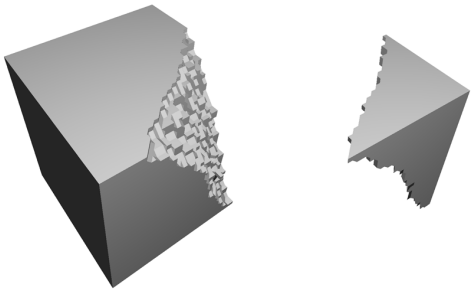


(h) Triangulation der Teekanne bei 70^3 Voxel

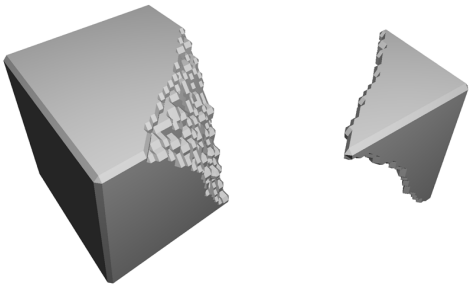
Abbildung 7.2: Voxelmodell einer einfachen Teekanne mit variierender Genauigkeit trianguliert. Die Abbildungen b-h zeigen oben jeweils die ursprüngliche Triangulation, die unteren eine Modifikation.



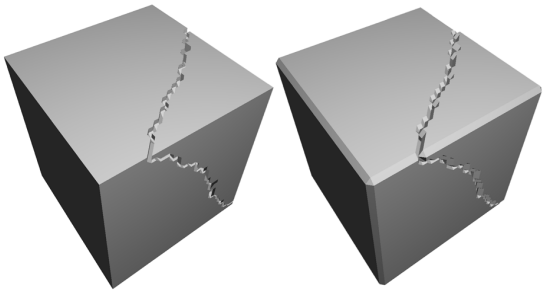
(a) Voxelmodell links; invertiertes Voxelmodell rechts



(b) Polygonmodell links; invertiertes Polygonmodell rechts; bei einer Genauigkeit von 50^3 Voxel



(c) Polygonmodell links; invertiertes Polygonmodell rechts; bei einer Genauigkeit von 50^3 Voxel (mit 'with outer edges')



(d) Mittels Photoshop zusammengesetzte Grafiken der Bruchstücke; (keine Funktionalität des Prototypen)

Abbildung 7.3: Die Invertierung des Voxelmodells führte zu einem Ansatz zur Realisierung des Bruchs eines 3D-Modells.

8 Ergebnis

8.1 Vergleich mit der Zielstellung

Die verfolgten Ziele wurden weitestgehend erreicht. Der Prototyp vermittelt auf eindrucksvolle Weise das Potenzial des Sculpturing von 3D-Modellen nach dem verfolgten Ansatz (vgl. Abbildung 8.1 auf der nächsten Seite). Es ist möglich, mittels der Gestaltung eines polygonal dargestellten Volumenmodells, ein 3D-Objekt zu entwerfen. Die Formgebung basiert auf dem Entfernen von Elementen des Voxelmodells. Während der Modellierung können Knoten verschiedener Ausdehnung weggenommen werden. Das triangulierte Objekt ist in ein gängiges 3D-Fileformat exportierbar. Das Volumenmodell, welches zum Sculpturing verwendet wird, ist persistent speicherbar. Damit ist eine Fortführung bereits begonnener Modellierung prinzipiell gewährleistet. Lediglich die verwendete Implementierung des Marching Cubes Algorithmus brachte einige Probleme mit sich. Die Kernfunktionalitäten des Prototypen werden davon nur eingeschränkt berührt.

8.2 Bewertung der Ergebnisse

Es war ein recht aufwendiger Entwickelungsprozess notwendig, um diesen Prototypen zu realisieren. Die gewonnenen Erkenntnisse sind breitgefächerter Natur. So wurde das allgemeine Verständnis bezüglich der Idee des Gestaltens mittels Volumenmodellen (Volumensculpturing) geschärft. Zusätzlich wurden eine Reihe von Erfahrungen im Bereich der Partitionierung des Raumes und der Anwendungsbereiche dieser Technologie errungen.

Im Besonderen ist die Erkenntnis zu bemerken, dass die stetig steigende Leistung, der zur Verfügung stehenden Rechnersysteme, völlig neue Anwendungsbereiche innerhalb der 3D-Computergrafik eröffnet. Die Leistungssteigerung, gerade im Bereich der Grafikbeschleuniger, lässt Strategien, welche bis dato notwendig waren um eine hohe Performanz zu gewährleisten, teilweise überflüssig werden. Der Polygondurchsatz moderner Grafikkarten ist in einem so hohen Maße gestiegen, das sich der Gewinn an Leistung durch Optimierung oftmals nur in noch höheren Bildwiederholraten ausdrückt. Ob ein System nun 100 oder 500 Bilder pro Sekunde leistet,

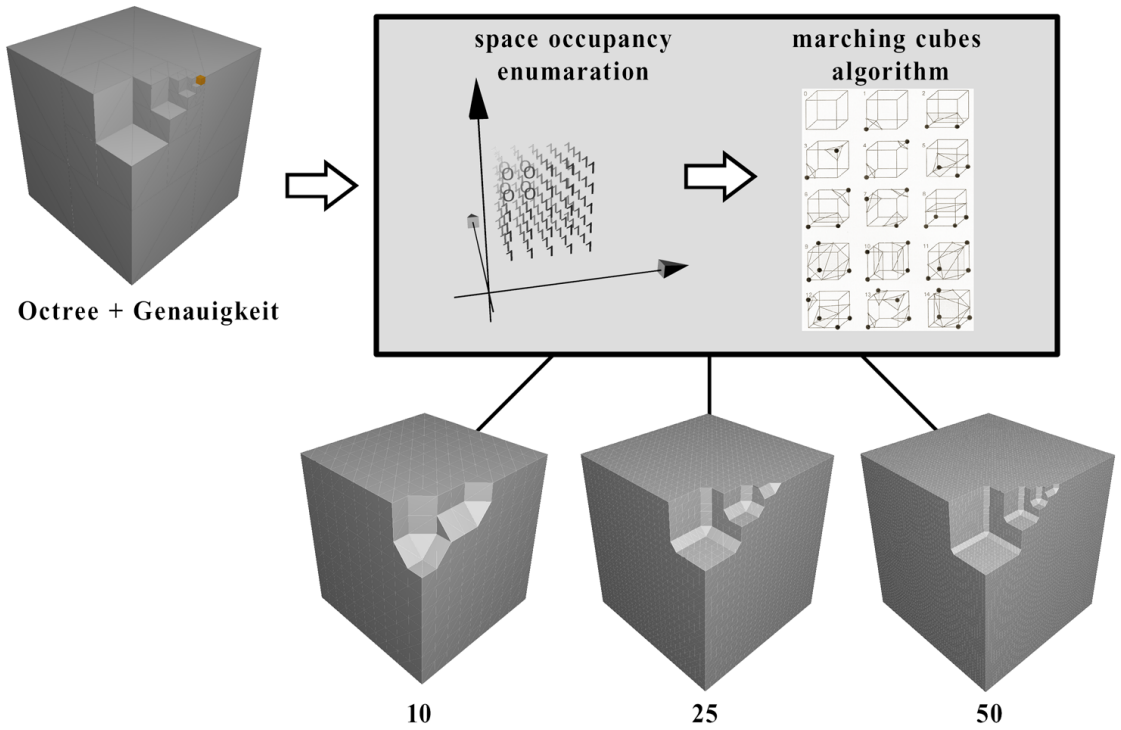


Abbildung 8.1: Volumenmodell, Abtastung und Triangulation ergeben Polygonmodelle, welche sich mit steigender Genauigkeit dem Volumenmodell annähern.

entzieht sich dabei dem Auge des Betrachters, da ein Monitor meist nicht mehr als 100Hz Bildwiederholfrequenz unterstützt. Problematisch stellt sich jedoch das ungleiche Wachstum der CPU- und GPU-Leistung dar. Den "Flaschenhals" innerhalb einer 3D-Anwendung bildet heutzutage oftmals die CPU und ihre Anbindung an den Systemspeicher, sowie den Grafikbeschleuniger. Der erstellte Prototyp zeigt eine Möglichkeit auf, dieses hohe Leistungsniveau nutzbar zumachen. Die Interaktion mit dem Voxelmodell beruht auf der Idee das Modell prinzipiell als Polygone innerhalb der Grafikkarte zu halten. Die Selektion eines Elementes ist dadurch sehr effizient ausführbar.

Allgemein werden die Ergebnisse als sehr positiv im Bezug auf die verfolgten Ziele angesehen. Zusätzlich zu den gewonnenen Erkenntnissen bereitete die prototypische Realisierung dieses Sculptutring Systems den Weg zur Erschließung innovativer Anwendungsbereiche der 3D-Computergrafik (siehe Kapitel 8.4.3 auf Seite 63 Bruch von 3D-Modellen).

8.3 Problembeschreibung

Es traten eine Reihe von Problemen während der Implementierung des Prototypen auf. Einige ließen sich ohne großen Aufwand bewältigen, andere sind bisher nicht gelöst worden. So existiert ein Problem mit dem Vollbildmodus. Der Prototyp stürzt ab, wenn nach erfolgter Triangulation auf Vollbild umgeschaltet oder die Fenstergröße geändert wird. Dieses Problem scheint sich auf die Verwendung der DXUT zurückführen zulassen. Die registrierte Callback-Funktion onlostDevice() wird bei Verlust der Referenz zum D3D-Device aufgerufen um alle Objekte wieder herzustellen. Das an die Methode übergebene D3D-Device ist allerdings ein NULL-Pointer. Es wird vermutet, dass dies mit den verwendeten ID3DXMESH Objekten zusammenhängt.

8.3.1 Probleme mit dem Marching Cubes

Zunächst zeigte sich, dass die Marching Cubes Implementierung von Paul Bourke (Bourke 1994, Webresource) nur einzelne Dreiecke generiert. Das entstehende Gittermodell besteht demnach aus nicht indizierten Dreiecken. Damit erhöhte sich die Anzahl der verwendeten Vertices zwar beträchtlich, allerdings traten keinerlei Performaceprobleme während der Visualisierung auf. Selbst sehr feine Gittermodelle werden flüssig dargestellt.

Ein wirkliches Problem äußerte sich wie folgt. Aus einem nicht ersichtlichen Grund wird eine große Anzahl von überschüssigen Dreiecken generiert. Dieses Verhalten wurde während des Imports eines OBJ-Files nach 3D Studio Max bemerkt. Bei Verwendung eines einzigen Würfels als Voxelmodell, welcher mit einer Genauigkeit von 10 Voxel pro Achse (10^3 Voxel) abgestastet wird, sollten 1200 einzelne Dreiecke entstehen[1]. Stattdessen werden 1448 Dreiecke generiert. Es ensteht ein zusätzlicher Überschuss an Geometriedaten von rund zwanzig Prozent. Dies wirkt sich zwar ebenfalls nicht auf die Visualisierung aus, allerdings enthält das exportierte Objekt doppelte Dreiecke.

8.3.2 Speichern des Voxelmodells

Um ein Modell einzuladen, muss die zuladende Datei im Verzeichnis des Prototypen liegen. Der Dateiname wird über einen Kommandozeilen-Parameter (voxmod.exe dateiname) übergeben, um beim Start der Anwendung geladen zu werden. Nach

[1] Zwei Dreiecke ergeben ein Viereck, 10 Vierecke pro Achse entstehen. Daraus folgt, dass ein Würfel aus 1200 einzelnen Dreiecken bestehen müsste ($10^2 * 6 Seiten * 2 Dreieckepro Viereck = 1200 Dreiecke$).

dem Einbau der Invertierungsfunktion funktioniert das Speichern des Objektmodells nur noch eingeschränkt. Dies äußert sich auf folgende Weise. Wenn das Voxelmodell einmal invertiert und danach gespeichert wurde, lässt sich dieses nicht mehr mit dem System laden[2].

8.4 Ausblick

Der Entwicklungsstand des Prototypen hat sein Optimum noch nicht erreicht. Er genügt zwar den Anforderungen eines Prototypen. Um allerdings ein ausgereiftes 3D Modelling Werkzeug zu verkörpern, bedarf es noch einiger Ergänzungen im Bereich der Funktionalität der Anwendung. Im nachfolgenden Abschnitt werden Erweiterungsmöglichkeiten und Optimierungen des Prototypen vorgestellt.

Bisher wird nur die Wegnahmen von Elementen des Voxelmodells unterstützt. Zunächst sollte also die Möglichkeit des Hinzufügens von zuvor entfernten Elementen implementiert werden. Nachfolgend wären verschiedene Sculpturing Werkzeuge bzw. Materialeigenschaften des entstehenden Gittermodells von Interesse. Einerseits könnten immer gleich mehrere Elemente selektiert bzw. entfernt/hinzugefügt werden. Auf diese Weise ließe sich die Eindringtiefe des Werkzeuges skalieren. Andererseits könnten die Würfel während der Abtastung, als eine andere primitive Form (z.B. Kugel oder Tetraeder) innerhalb des, von der Raumbelegung erzeugten Voxelfeldes, abgebildet werden. Dies würde der Einführung neuer Materialeigenschaften gleichkommen. Zur Erweiterung der Gestaltungsmöglichkeiten wäre selbstverständlich die Kolorierung des Voxelmodells von Interesse. Das triangulierte Ergebnis könnte die Voxelfarben als Vertex Farbe einsetzten. Auch das Bemalen von fertigen 3D-Modellen mittels Verwendung von Texturen wäre zu nennen. Die Generierung eines Oktonärbaums zur weiteren Bearbeitung von Teilbereichen eines bereits bestehenden 3D-Modells wäre wünschenswert. Die Realisierung dieser Funktionalität bedarf allerdings der Lösung diverser Probleme. Im Nachstehenden werden Strategien zur Optimierung des Prototypen, sowie die Idee zum Bruch von 3D-Modellen erläutert.

8.4.1 Optimierung der Voxeldarstellung

Die polygonale Darstellung des Voxelmodell ist bisher nicht optimiert. Die Einführung des sogenannten "Object-Occlusion-Culling" (Deloura 2002, S.410-413) würde

[2]Da die zur Verfügung stehende Zeit sich dem Ende neigt, nimmt dieses Problem eine untergeordnete Rolle innerhalb der Erstellung dieser Arbeit ein. Es wird versucht, dieses Problem bis zur Abgabe zu beseitigen.

die Darstellung um einiges effizienter gestallten. Zur Zeit kann nur die maximal vom Zielsystem unterstützte Anzahl von einzelnen 3D-Objekten gezeichnet werden (ca. 3800 3D-Objekte innerhalb des Oktonärbaums). Darüber hinaus bricht die Leistung der Visualisierung auf unterhalb von einem Bild pro Sekunde ein (vgl. Kapitel 3.3.3 auf Seite 29). Eine weitere Gestaltung muss in diesem Falle in der Ansicht des triangulierten Gittermodells erfolgen. Dabei wurde schon mit mehr als 20.000 3-D Objekten innerhalb des Baumes gearbeitet. Bei Einsatz von Object-Occlusion-Culling würden alle 3D-Objekte, welche aus Sicht des Betrachters verdeckt sind, nicht gezeichnet. Damit wäre eine wesentlich grössere Anzahl von Elementen des Oktonärbaums darstellbar.

8.4.2 Optimierung der Triangulation

Nachfolgend werden Strategien zur Optimierung des Triangulations Prozesses erklärt. Während der Triangulation werden derzeit zunächst alle sichtbaren Knoten des Baumes in einer Liste gespeichert. Diese werden anschließend von der Klasse VxM_VoxelArray in ein Voxelfeld überführt. Dieses wiederum wird vom Marching Cubes zur Triangulation abgelaufen. Als Letztes werden alle erzeugten Dreiecke in 3D-Objekte umgeschichtet (vgl. Abbildung A.5 auf Seite 72). Bei Veränderung der Architektur der Anwendung könnte dieser Vorgang zu einem einzigen Prozess zusammengefasst werden. Dabei würde sicherlich CPU-Zeit eingespart.

Die Verwendung eines Oktonärbaum als Voxelmodell könnte zusätzlich von Vorteil zur Optimierung der Anwendung sein. Wenn in den Blättern des Baums die zugehörigen Dreiecke des triangulierten Objektes gespeichert würden, so könnte die Triangulation inkrementell ausgeführt werden. Nur die jeweils veränderten Knoten würden trianguliert und in das bestehende Gitter integriert. Außerdem wäre eine Optimierung des Marching Cubes, nach dem von Raj Shekhar u.a. 1996 vorgeschlagenen Verfahren, interessant. Ihnen ist es gelungen, bei Erhalt der Form, die Anzahl der generierten Flächen zu minimieren (Shekhar u. a. 1996, Paper abstract).

8.4.3 Bruch von 3D-Modellen

Die Idee beruht im Wesentlichen auf dem Gedanken, dass sich ein Voxelmodell auf einfache Weise invertieren lässt. Durch die Möglichkeit, dieses Modell zu triangulieren, ergibt sich die Grundlage zur Erzeugung eines weggenommen Teils. Eine Realisierung wäre durch die Modifikation der Abtastung des Baums möglich. Während der Abtastung würden zwei Felder mit Voxldaten aufgebaut. Eines für die sichtbaren Teile des Voxelmodells und eines für die nicht sichtbaren. Damit wäre es möglich,

unabhängige Gittermodelle zu erzeugen. Die Abbildungen 7.3 auf Seite 58 erläutern das Prinzip, welches dieser Idee zugrunde liegt.

Um eine wirkungsvolle Interaktion zu gewährleisten, sollten allerdings neue Werkzeuge erstellt werden. Die Auswahl eines Elements hat je nach Einstellung z.Z. eine Verfeinerung des Baums zur Folge. Wenn vom getroffenen Element ausgehend, die dahinterliegenden Würfel verfeinert und anschließend selektiert würden, könnte man abgeschrägte Brüche erzeugen. Es wäre auch möglich, eine gerichtete Selektion auszuführen, bei der ähnlich einem L-System (Wikipedia b, Webresource), eine Handlungsanweisung die Art und Weise der Selektion bestimmt.

Eine anderer Anwendungsbereich läge im Bereich der Videospielindustrie. Man könnte die verwendete Technologie dahingehend modifizieren, dass bereits existierende 3D Modelle interaktiv verändert (z.B. zerstört) werden könnten. Dazu würde der Oktonärbaum an der Stelle der Interaktion erzeugt, um Teile des bestehenden Gittermodells zu partitionieren. Die Interaktion des Benutzers würde zusätzlich innerhalb des Baums abgebildet. Dadurch könnten die veränderten Bereiche trianguliert und in das bestehende 3D-Modell eingefügt werden. Die herausgelösten Teile könnten ihrerseits erzeugt bzw. animiert werden. Das Ergebnis wäre eine 3D-Welt, deren Topologie sich interaktiv verändern lässt.

A Anhang

Die folgende Tabelle enthält alle im Kapitel 3.3.3 ermittelten Messwerte.

Arraysize	3D-Objekte	Polygonanzahl	FPS	Poly/3D-Obj	Poly/Sekunde
1	1	24	509	24	12216
1	1	872	521	872	454312
1	1	3542	519	3542	1838298
1	1	8012	521	8012	4174252
1	1	14282	500	14282	7141000
1	1	22352	514	22352	11488928
2	8	192	487	24	93504
2	8	6976	482	872	3362432
2	8	6976	485	872	3383360
2	8	28336	484	3542	13714624
2	8	64096	488	8012	31278848
2	8	114256	333	14282	38047248
2	8	178816	226	22352	40412416
4	64	1536	339	24	520704
4	64	45056	318	704	14327808
4	64	55808	336	872	18751488
4	64	63616	320	994	20357120
4	64	94976	320	1484	30392320
4	64	126848	297	1982	37673856
4	64	226688	180	3542	40803840
4	64	355328	119	5552	42284032
4	64	512768	84	8012	43072512
4	64	914048	48	14282	43874304
4	64	1430528	31	22352	44346368
6	216	5184	196	24	1016064
6	216	45792	195	212	8929440
6	216	188352	196	872	36916992
6	216	428112	98	1982	41954976
6	216	765072	56	3542	42844032
6	216	1199232	36	5552	43172352

6	216	1730592	25	8012	43264800
6	216	3084912	14	14282	43188768
6	216	4828032	8	22352	38624256
8	512	12288	106	24	1302528
8	512	108544	105	212	11397120
8	512	446464	90	872	40181760
8	512	1014784	42	1982	42620928
8	512	1813504	24	3542	43524096
8	512	2842624	15	5552	42639360
8	512	4102144	9	8012	36919296
8	512	5592064	6	10922	33552384
8	512	7312384	4	14282	29249536
9	729	199746	77	274	15380442
9	729	513216	76	704	39004416
9	729	919998	46	1262	42319908
9	729	1256796	34	1724	42731064
9	729	2087856	21	2864	43844976
10	1000	22000	60	22	1320000
10	1000	24000	60	24	1440000
10	1000	212000	59	212	12508000
10	1000	554000	58	554	32132000
10	1000	652000	57	652	37164000
10	1000	704000	57	704	40128000
10	1000	814000	50	814	40700000
10	1000	872000	47	872	40984000
10	1000	1192000	35	1192	41720000
10	1000	1982000	22	1982	43604000
10	1000	3542000	11	3542	38962000
10	1000	4162000	8	4162	33296000
10	1000	5552000	6	5552	33312000
11	1331	31944	45	24	1437480
11	1331	42592	46	32	1959232
11	1331	149072	45	112	6708240
11	1331	178354	45	134	8025930
11	1331	508442	44	382	22371448
11	1331	867812	44	652	38183728
11	1331	1586552	26	1192	41250352
11	1331	1874048	23	1408	43103104
11	1331	5201548	5	3908	26007740

12	1728	41472	36	24	1492992
12	1728	100224	36	58	3608064
12	1728	127872	14	74	1790208
12	1728	127872	35	74	4475520
12	1728	231552	35	134	8104320
12	1728	957312	35	554	33505920
12	1728	1126656	35	652	39432960
12	1728	1216512	33	704	40144896
12	1728	2059776	20	1192	41195520
13	2197	52728	28	24	1476384
13	2197	839254	27	382	22659858
13	2197	1915784	22	872	42147248
13	2197	4354454	7	1982	30481178
14	2744	65856	23	24	1514688
14	2744	65856	23	24	1514688
14	2744	1048208	22	382	23060576
14	2744	1048208	22	382	23060576
14	2744	1789088	22	652	39359936
14	2744	2392768	17	872	40677056
14	2744	2392768	17	872	40677056
14	2744	3270848	11	1192	35979328
14	2744	3462928	9	1262	31166352
15	3375	81000	18	24	1458000
15	3375	1289250	18	382	23206500
15	3375	2200500	18	652	39609000
15	3375	2943000	12	872	35316000
15	3375	4023000	7	1192	28161000
16	4096	98304	1	24	98304
16	4096	303104	1	74	303104
16	4096	1261568	1	308	1261568
16	4096	1261568	1	308	1261568

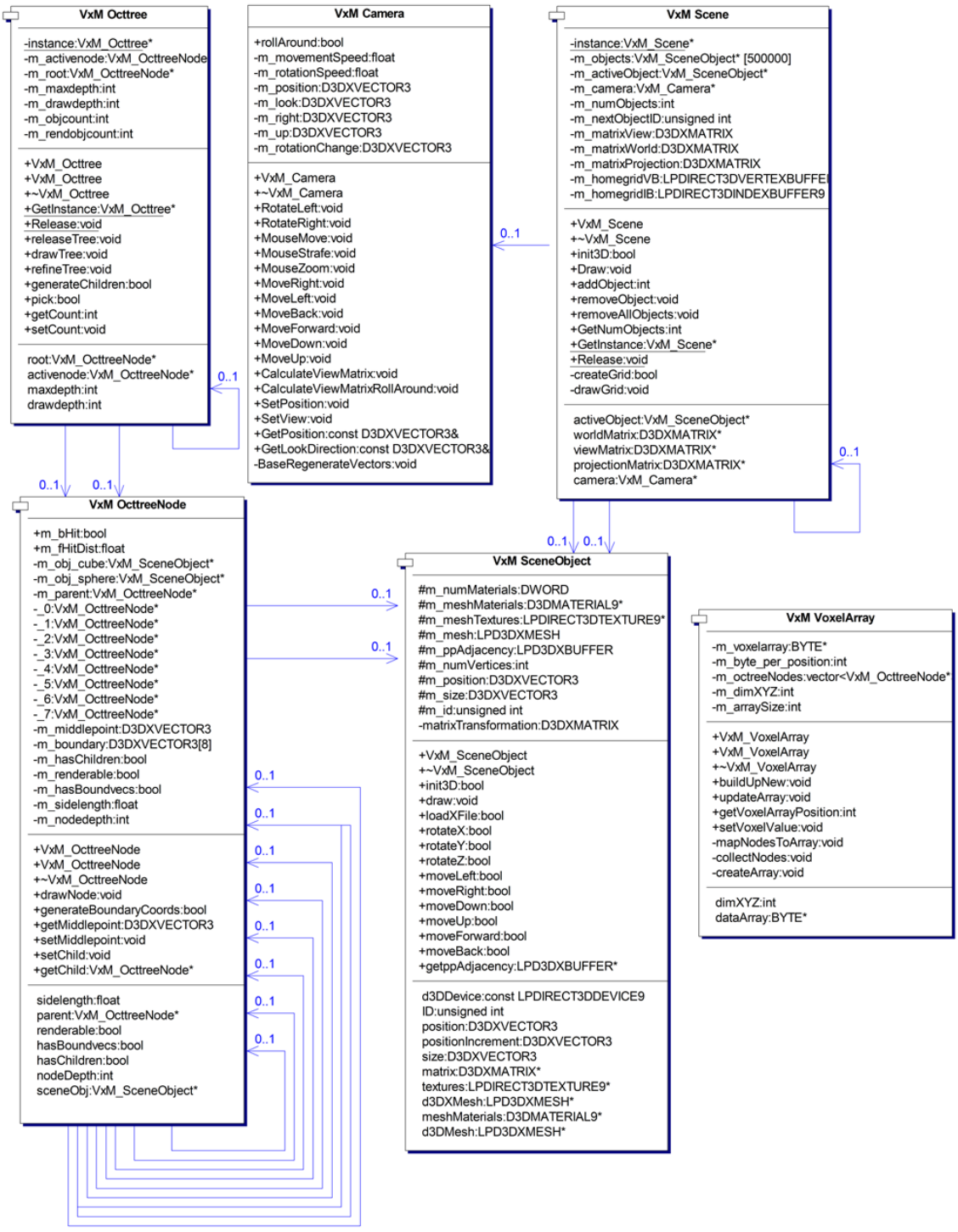


Abbildung A.1: Klassendiagramm: Model Komponenten

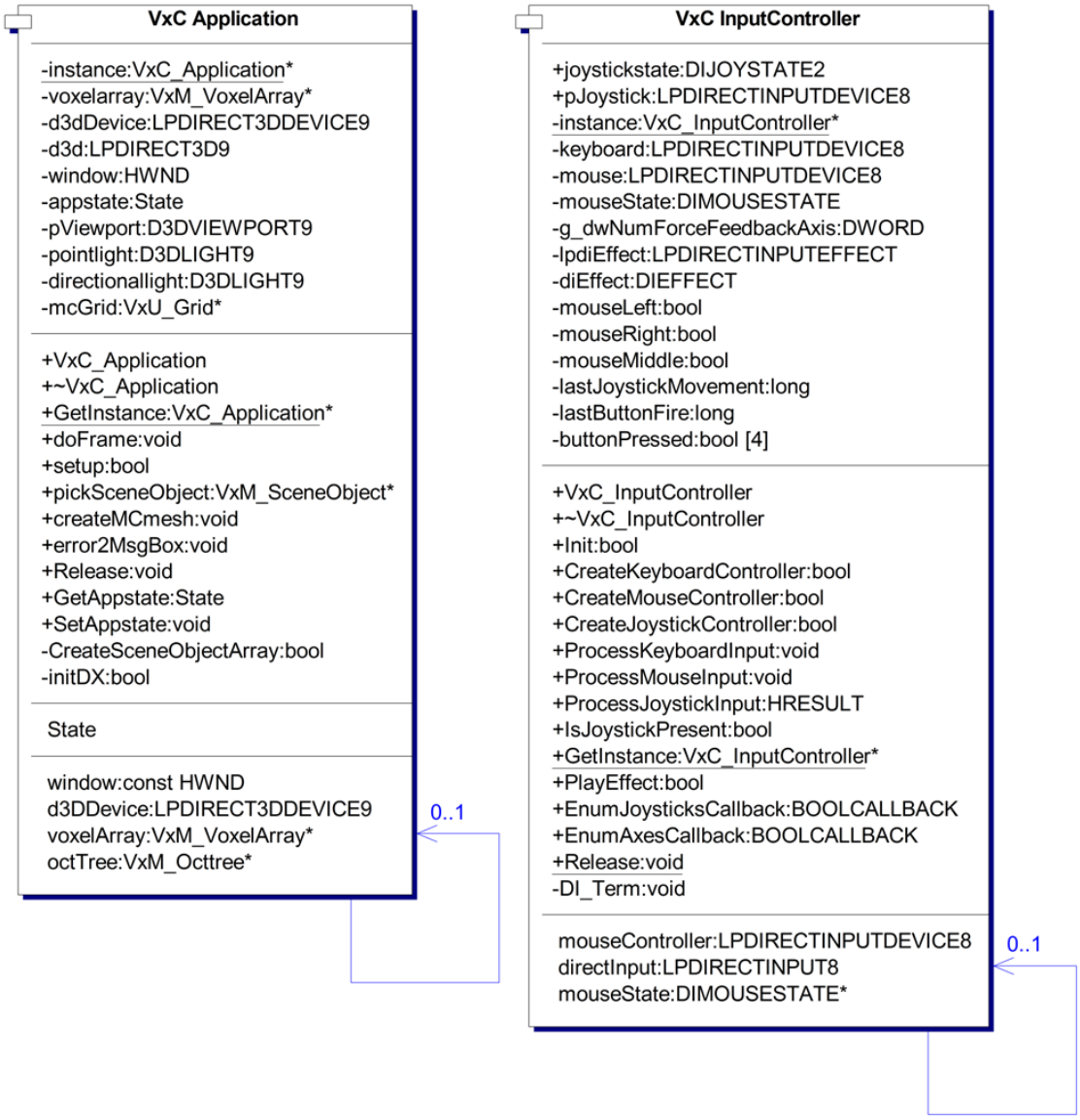


Abbildung A.2: Klassendiagramm: Control Komponenten

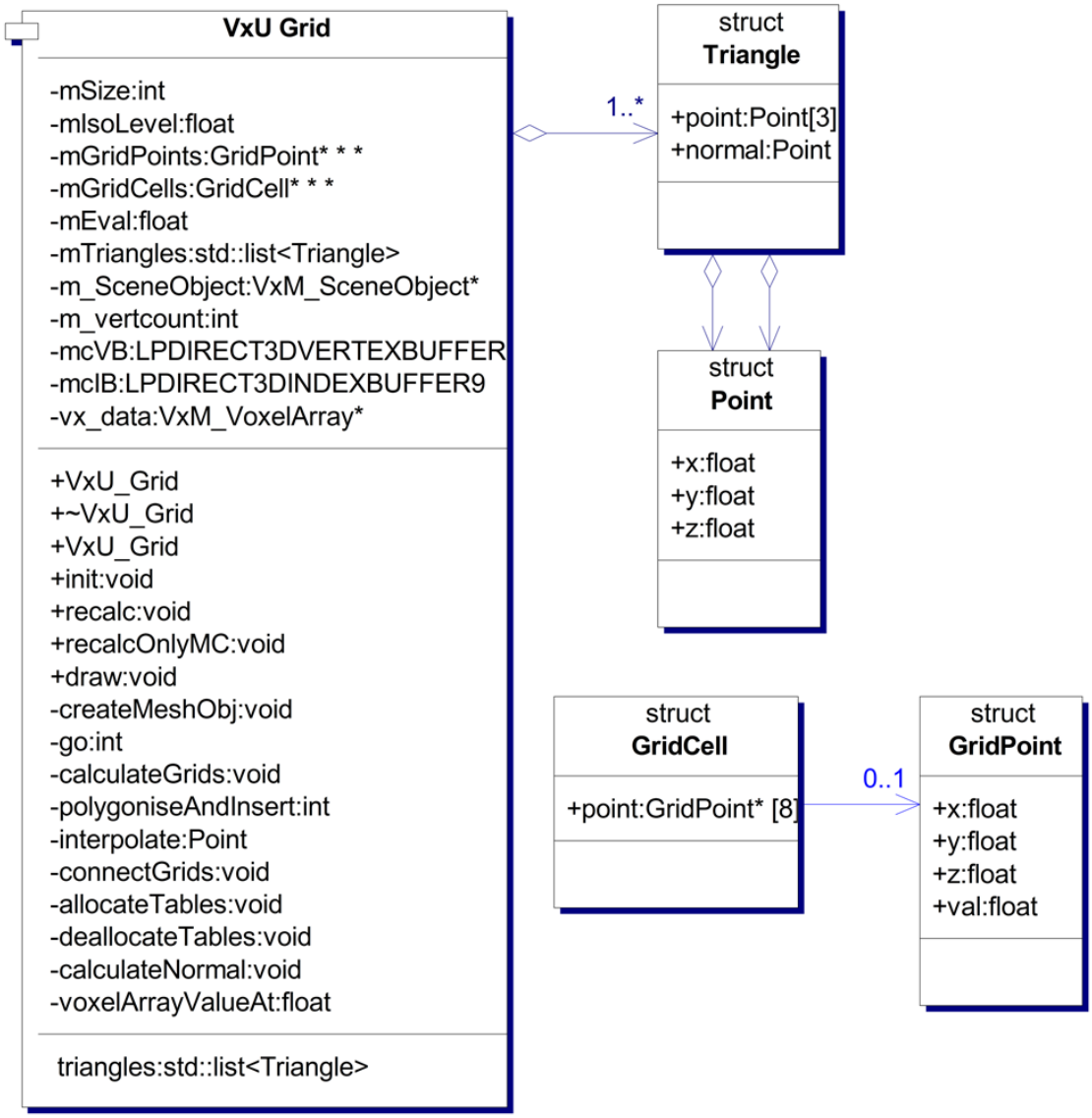


Abbildung A.3: Klassendiagramm: Utility Komponenten

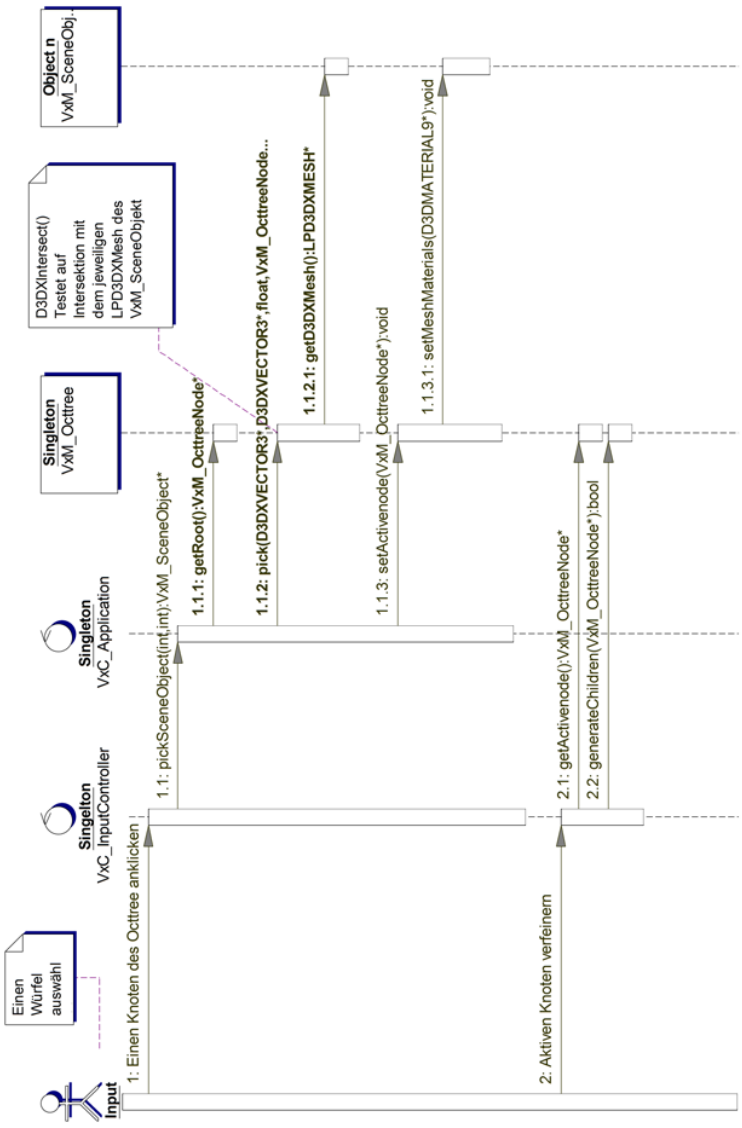


Abbildung A.4: Sequenzdiagramm der Selektion eines Knoten mit anschließender Verfeinerung.

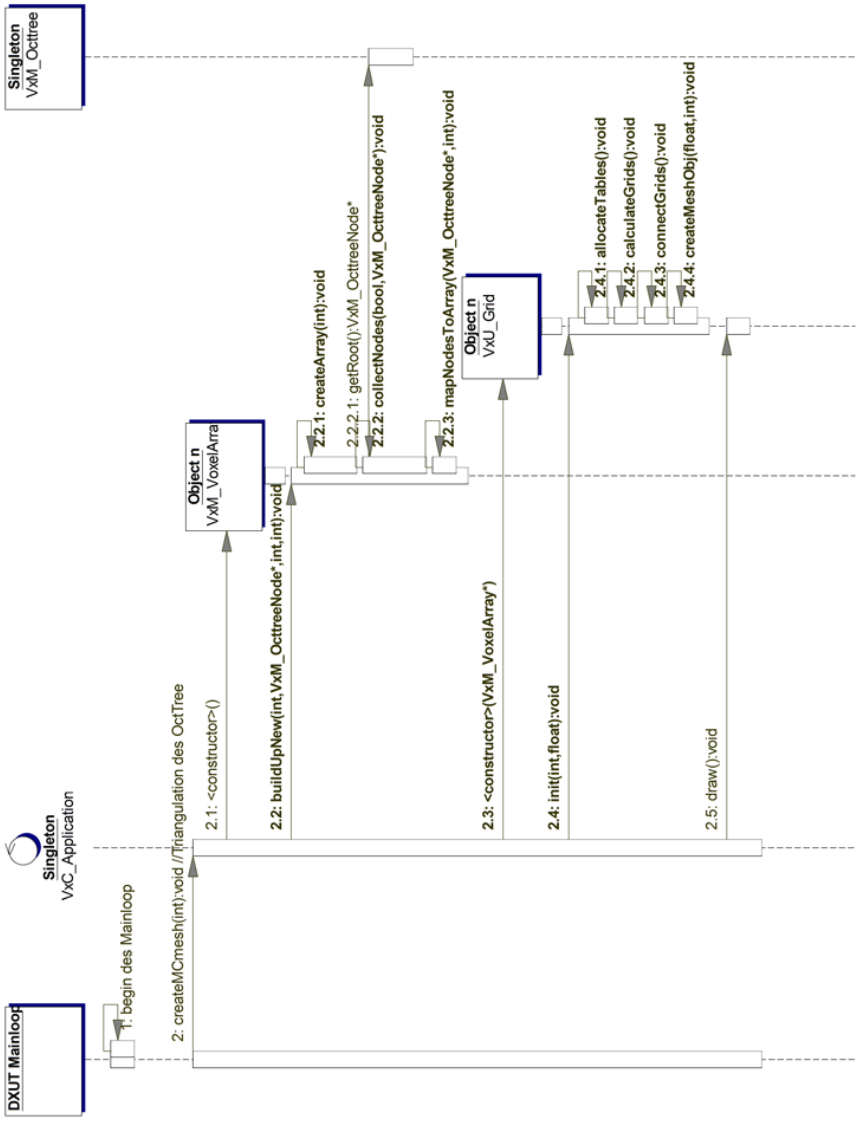


Abbildung A.5: Sequenzdiagramm: Triangulation des Oktonärbaums

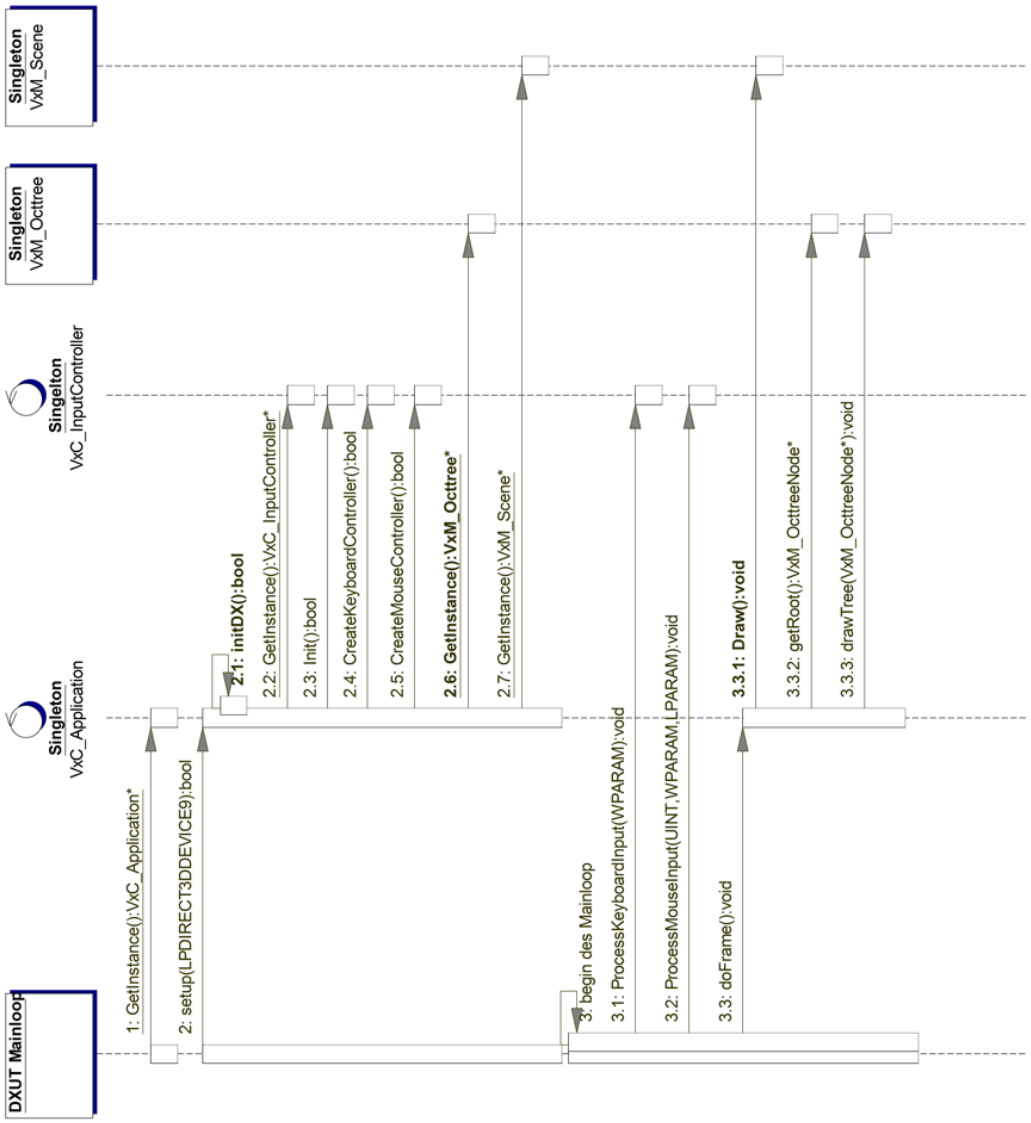
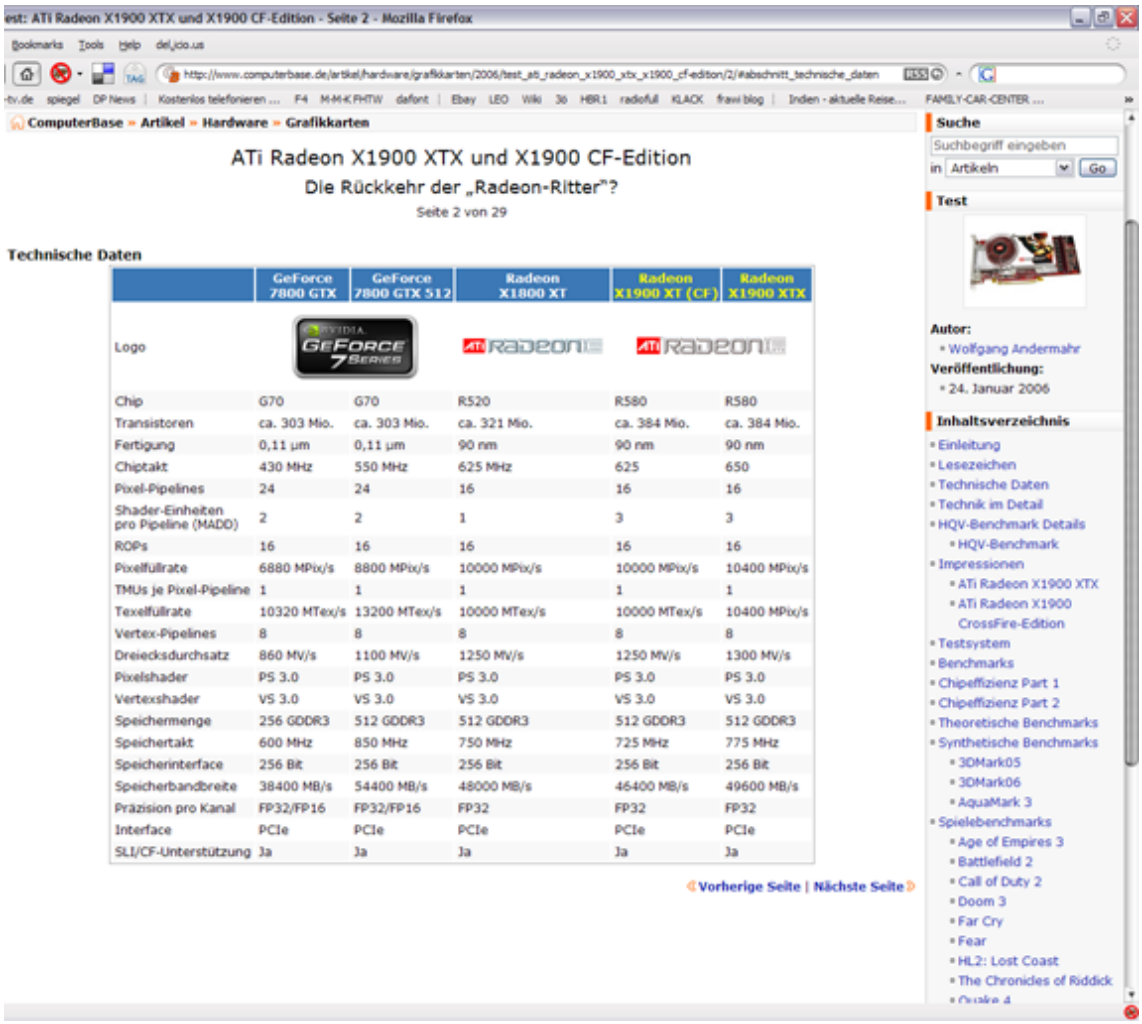


Abbildung A.6: Sequenzdiagramm: Initialisieren der Applikation



	GeForce 7800 GTX	GeForce 7800 GTX 512	Radeon X1800 XT	Radeon X1900 XT (CF)	Radeon X1900 XTX
Logo					
Chip	G70	G70	R520	R580	R580
Transistoren	ca. 303 Mio.	ca. 303 Mio.	ca. 321 Mio.	ca. 384 Mio.	ca. 384 Mio.
Fertigung	0,11 µm	0,11 µm	90 nm	90 nm	90 nm
Chiptakt	430 MHz	550 MHz	625 MHz	625	650
Pixel-Pipelines	24	24	16	16	16
Shader-Einheiten pro Pipeline (MADD)	2	2	1	3	3
ROPs	16	16	16	16	16
Pixelfüllrate	6880 MPix/s	8800 MPix/s	10000 MPix/s	10000 MPix/s	10400 MPix/s
TMUs je Pixel-Pipeline	1	1	1	1	1
Texelfüllrate	10320 MTex/s	13200 MTex/s	10000 MTex/s	10000 MTex/s	10400 MPix/s
Vertex-Pipelines	8	8	8	8	8
Dreiecksdurchsatz	860 MV/s	1100 MV/s	1250 MV/s	1250 MV/s	1300 MV/s
Pixelshader	PS 3.0	PS 3.0	PS 3.0	PS 3.0	PS 3.0
Vertexshader	VS 3.0	VS 3.0	VS 3.0	VS 3.0	VS 3.0
Speichermenge	256 GDDR3	512 GDDR3	512 GDDR3	512 GDDR3	512 GDDR3
Speichertakt	600 MHz	850 MHz	750 MHz	725 MHz	775 MHz
Speicherinterface	256 Bit	256 Bit	256 Bit	256 Bit	256 Bit
Speicherbandbreite	38400 MB/s	54400 MB/s	48000 MB/s	46400 MB/s	49600 MB/s
Präzision pro Kanal	FP32/FP16	FP32/FP16	FP32	FP32	FP32
Interface	PCIe	PCIe	PCIe	PCIe	PCIe
SLI/CF-Unterstützung	Ja	Ja	Ja	Ja	Ja

Abbildung A.7: Grafikkarten Leistungsvergleich der Webseite Computerbase

Literaturverzeichnis

Autodesk 2004a
AUTODESK: 3D Studio Max 7 new features guide. (2004), Oktober

Autodesk 2004b
AUTODESK: 3D Studio Max 7 Users Reference. (2004)

Bærentzen 1998
BÆRENTZEN, Jakob A.: Octree-based Volume Sculpting. In: *Journal of Visualization and Computer Animation* (1998), 1-4. `http://citeseer.ist.psu.edu/boerentzen98octreebased.html`. ISBN 1–58113–106–2

Catmull u. Clark 1978
CATMULL, E. ; CLARK, J.: Recursively generated B-spline surfaces on arbitrary topological surfaces. In: *Computer-Aided Design* 10(6) (1978), November, S. 350–355

Cristiano u. a. 2004
CRISTIANO, S. ; FIORENTINO, M. ; MONNO, G. ; UVA, A. E.: Politecnico di Bari: REAL-TIME PARTICLE BASED VIRTUAL SCULPTURING. (2004), 31.August - 2.September, 1-9. `http://climeg.poliba.it/~disegno/vr3lab/`

Deloura 2002
DELOURA, Marc: *Spieleprogrammierungs Gems 1.* 1. Bonn : mitp-Verlag, 2002. – ISBN 3–8266–0923–9

Doo u. Sabin 1978
DOO, D. ; SABIN, M.: Behaviour of recursive division surfaces near extraordinary points. In: *Computer-Aided Design* 10(6) (1978), November, S. 356–360

Friebe u. Kuhn 2006
FRIEBE, Dr. J. ; KUHN, Michael: Virtueller Bildhauer. In: *Digital Production* 03 (2006)

Galyean u. Hughes 1991
GALYEAN, Tinsley A. ; HUGHES, John F.: An interactive volumetric modeling technique. In: *ACM Computer Graphics* 25 (1991), Nr. 4, S. 267–274

Lorenson u. Cline 1987
LORENSON, W.E. ; CLINE, H.E.: Marching Cubes: a high resolution 3D Surface construction algorithm. (1987), July 27.-31., S. 7

McDonnell u. a. 2001
MCDONNELL, Kevin T. ; QIN, Hong ; WLODARCZYK, Robert A.: Virtual clay: a real-time sculpting system with haptic toolkits. In: *Symposium on Interactive 3D Graphics*, 2001, 179-190

Olmos 2004
OLMOS, Pablo: *Virtuelle Charaktere mit 3ds Max Modeling Charakter-Setup Animation*. 1. Bonn : Galileo Design, 2004. – ISBN 3–89842–376

Perng u. a. 2001
PERNG, Kuo-Luen ; WANG, Wei-Teh ; FLANAGAN, Mary ; OUHYOUNG, Ming: A Real-time 3D Virtual Sculpting Tool Based on Modified Marching Cubes. Virtual Sculpting System was created by Perng Kuo-Luen(Peter Pon) as my master thesis. (2001), Mai, 1-9. `http://www.cmlab.csie.ntu.edu.tw/%7Eperng/projects/sculpture/index_e.html`

Production 2005
PRODUCTION, Digital: Nvidia: Neue SLI-Grafikkarten und 3D-Software. In: *Digital Production* 05 (2005)

de Rahm 1956
RAHM, G. de: Sur une courbe plane. In: *J. de Math. pures et Appl.* 35 (1956), November, S. 25–42

Saona-Vazquez u. a. 1999
SAONA-VAZQUEZ, C. ; NAVAZO, I. ; BRUNET, P.: The visibility octree: A data structure for 3d navigation. In: *Computer and Graphics* 23(5) (1999), 635-644. `http://citeseer.ist.psu.edu/274871.html`

Shekhar u. a. 1996
SHEKHAR, Raj ; FAYYAD, Elias ; YAGEL, Roni ; CORNHILL, J.Fredrick: Octree-Based Decimation of Marching Cubes Surfaces. In: *(IEEE) Visualization*, 1996, 335-342

Wang u. Kaufman 1995a
WANG, Sidney ; KAUFMAN, Arie E.: Volume sculpting. In: *Symposium on Interactive Graphics* (1995)

Watt 2002
WATT, Alan: *3D-Computergrafik*. 3. München : Addison-Wesley, 2002. – ISBN 3–8273–7014–0

Internetverzeichnis

Bourke 1994
BOURKE, Paul ; BOURKE, Paul (Hrsg.): *Polygonising A Scalar Field*. Version: Mai 1994. http://local.wasp.uwa.edu.au/~pbourke/modelling/polygonise/l, Abruf: 15. Okt. 2006. – Marching Cubes Implementierung

computerbase 2006
ATI Radeon X1900 XTX und X1900 CF-Edition. Version: 2006. http://www.computerbase.de/artikel/hardware/grafikkarten/2006/test_ati_radeon_x1900_xtx_x1900_cf-edition/2/#abschnitt_technische_daten, Abruf: 15. Okt. 2006. – Vergleichstest von Grafikkarten der Hersteller NVIDIA und ATI

Ditchburn 2006a
DITCHBURN, Keith: *Camera in Direct3D*. Version: 2006. http://www.toymaker.info/Games/html/camera.html, Abruf: 29. Okt. 2006. – Erstellen einer Kamera unter Direct3D

Ditchburn 2006b
DITCHBURN, Keith: *Games Programming*. Version: 2006. http://www.toymaker.info/Games/index.html, Abruf: 29. Okt. 2006. – Spiele Programmierung mit DirectX

Dunlop 2002a
DUNLOP, Robert: *Improved Ray Picking*. Version: 2002. http://www.mvps.org/directx/articles/improved_ray_picking.htm, Abruf: 29. Okt. 2006. – Microsoft DirecX MVP(Most Valuable Professional)

sample Framework DXUT 2006
FRAMEWORK DXUT, Microsoft sample ; MICROSOFT (Hrsg.): *Programming Guide*. Version: 2006. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/directx9_c/dx9_graphics_programming_guide.asp, Abruf: 15. Okt. 2006. – Onlinetutorial

Joswig u. Polthier 2000
JOSWIG, Michael ; POLTHIER, Konrad ; BERLIN, Freie U. (Hrsg.): *OBJ-Fileformat*. Version: 2000-2005. http://www.eg-models.de/formats/

Format_Obj.html, Abruf: 15. Okt. 2004. – Beschreibung des OBJ Fileformats

Microsoft
MICROSOFT: *D3DX Reference: Mesh Functions*. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/directx9_c/dx9_graphics_reference_d3dx_functions_mesh.asp, Abruf: 29. Okt. 2006. – Direct3D Online Referenz

OpenGL 1997
advanced97. Version: 1997. http://www.opengl.org//resources/code/samples/advanced/advanced97/notes/node180.html, Abruf: 15. Okt. 2006. – Volumenrendering mit OpenGL

Wikipedia a
WIKIPEDIA: *Einzelstück(Entwurfsmuster)*. http://de.wikipedia.org/wiki/Einzelst%C3%BCck_%28Entwurfsmuster%29, Abruf: 20. Okt. 2006

Wikipedia b
WIKIPEDIA: *Lindenmayer System (L-System)*. http://de.wikipedia.org/wiki/L-System, Abruf: 1. Nov. 2006. letzte Änderung erfolgte am 7.9.2006

Wikipedia c
WIKIPEDIA: *Prototyping (Softwareentwicklung)*. http://de.wikipedia.org/wiki/Prototyping_(Softwareentwicklung), Abruf: 27. Okt. 2006. letzte Änderung erfolgte am 21.9.2006

Wikipedia d
WIKIPEDIA: *Raumwahrnehmung*. http://de.wikipedia.org/wiki/Raumwahrnehmung, Abruf: 6. Nov. 2006. letzte Änderung erfolgte am 20.7.2006

Erklärung

Hiermit versichere ich, dass ich die vorliegende Diplomarbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel verfasst habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

Berlin, den 26. Januar 2007 Frank Otto