3D Modellierung von Oberflächen mittels Marching Cubes Algorithmus und generische Darstellung mittels OpenGL

Peter P. Ortner

BACHELORARBEIT

Nr. S1210307080-A

eingereicht am
Fachhochschul-Bachelorstudiengang

Software Engineering

in Hagenberg

im August 2015

Diese Arbeit entstand im Rahmen des Gegenstands

Digitale Bildverarbeitung und Graphik

im

Sommersemester 2015

Betreuer:

Werner Backfrieder, FH-Prof. DI Dr.

Erklärung

Ich erkläre eidesstattlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benutzt und die den benutzten Quellen entnommenen Stellen als solche gekennzeichnet habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

Hagenberg, am 1. August 2015

Peter P. Ortner

Inhaltsverzeichnis

Kurzfassung

In der computergestützten Bildverarbeitung gibt es diverse Möglichkeiten für die Darstellung von dreidimensionalen Objekten. Die wohl am weitesten verbreitete Darstellungsform ist die polygonale Darstellung. Diese Form der Aufbereitung zerlegt ein gegebenes Objekte in Dreiecke.

Ein weiteres Verfahren ist die Methode der Modellierung via einer so genanten Voxel-Datenmenge. Jedoch birgt diese, im Bezug auf die digitale Verarbeitung, einige Nachteile gegenüber der polygonalen Darstellungsform. Vordergründige Probleme hierbei sind der vergleichsweise hohe Speicherverbrauch der Modelle, die Visualisierung benötigt länger und Objektmanipulationen erweisen sich als schwieriger.

Da in der Medizin im Bereich der bildgebenden Systeme wie der Computertomografie von Natur aus solche Voxel-Modelle erzeugt werden, besteht die Anforderung auch diese nach Möglichkeit schnell und Aussagekräftig darzustellen.

Um diesen Anforderungen an die Darstellung gerecht zu werden bietet sich der so genante Marching-Cubes Algorithmus an. Dieser ermöglicht es eine Voxel-Datenmenge in eine polygonale Darstellung zu überführen.

Abstract

Considering the computer based image processing there are multiple possibilities to represent three-dimensional objects. The most common way to illustrate these objects is the polygonal approach. This approach fragments an object into triangles.

An other possible procedure to model objects is to represent them as a voxel grid. But if we consider the ability to process this kind of representation we have to face some disadvantages. The main problems are: the model needs a comparatively high amount of disk space, it takes much longer to show the image and it is difficult to perform manipulations on the object.

In the field of Medical imaging such as computed tomography creates such voxel models, these should be presented quickly and meaningfully.

To implement these requirements on the presentation we have to transform voxel grids into polygon objects. The so called marching cubes algorithm can achieve this goal.

Kapitel 1

Einleitung

1.1 Aufgabenstellung

In der medizinischen Diagnostik wird im Gegensatz zu CAD-Konstruktionen die dreidimensionale Gestalt anatomischer Details aus Volumsbildern abgeleitet. Durch vorangegangene Segmentierung werden binäre Objekte erzeugt, d.h. das Objekt ist wie eine Lego-Figur aufgebaut. Mit dem Marching Cubes Algorithmus wird aus diesem binären Volumen eine Oberfläche, die aus Dreiecken besteht aufgebaut. Diese Oberfläche wird in einem binären STL-Format persistiert und anschließend mit einem generischen Rendering als 3D-Objekt dargestellt.

Anforderungen: C/C++ Implementierung des MC-Algorithmus (Matlab-Version vorhanden), Konversion in STL-Format, OpenGl Visualisierung.

1.2 Motivation

Da moderne Grafikchips für die Darstellung von polygonalen Modellen ausgelegt sind, ist es sinnvoll die aus der medizinischen Diagnostik erhaltenen Voxel-Modelle, für spätere Verarbeitung, in diese Form zu überführen. Ein weiterer Vorteile neben der vereinfachten Verarbeitung und Darstellungsform von Polygonen liegt in dem vergleichsweise geringen Speicherbedarf eines solchen Objektes.

1.3 Zielsetzung

Ziel dieser Arbeit ist es, die aus bildgebenden Verfahren der Medizin erhaltenden Voxel-Mengen mithilfe des Marching Cubes Algorithmus in eine Polygonale Darstellung zu überführen. Als Input werden die Daten welche

von dem Programm Analyze (7.5)[1] erzeugten werden (Image und Header File) verwendet. Die Voxel-Menge welche in der Image-Datei abgelegt ist wird ausgelesen und mithilfe des Marching Cubes Algorithmus in Polygone zerlegt. Nach erfolgreicher Umwandlung wird die erhaltene Datenmenge via OpenGL dargestellt. Des Weiteren soll das Modell als STL-Datei exportiert werden können. Die gesamte Umsetzung erfolgt in der Programmiersprache C++.

[1] https://rportal.mayo.edu/bir/

Kapitel 2

Allgemeine Einführung

2.1 Bildgebende Verfahren

> "Die Medizinische Bildverarbeitung hat das Ziel, medizinische Bilder und Bildfolgen zur Unterstützung der medizinischen Diagnostik und Therapie aufzubereiten, zu analysieren und zu visualisieren." - [Hadels, 2000]

Die verschiedenen medizinischen Verfahren können in die Art der erzeugten Bilddaten eingeteilt werden:

- **Schnittbilder** z.B. mittels Computertomografie, Magnetresonanztomografie oder Röntgentomografie.
- **Projektionsbilder** z.B. durch "klassisches" Röntgen.
- **Oberflächenabbildungen** z.B. durch Rastertunnelmikroskop.

Da das Hauptaugenmerk dieser Arbeit liegt auf der aus den tomographischen Verfahren erhaltenden Schnittbildern, welche als sogenannte Voxel-Daten gespeichert werden. Ein vollständiges dreidimensionales Bild besteht aus mehreren solcher übereinandergelegten Schnittbildern.

2.2 Volumengrafik

Unter Volumengrafik versteht man in der Computergrafik die Darstellung von Objekten durch eine Menge von Voxeln.

Ein solches Voxel ist als ein einzelner Punkt in einem dreidimensionalen Objekt zu verstehen, welcher einen gewissen Dichtewert aufweist. Dieser Wert ist essentiell um z.B. bei den tomographischen Verfahren in der Medizin die festeren von den weicheren teilen eines Körpers zu unterscheiden (z.B. Knochen und Gewebe). In Abbildung 2.1 ist eine solche Voxel-Menge zusehen die verschiedenen Grauwerte der einzelnen Bildpunkte stellen dabei die unterschiedlichen Dichtewert dar.

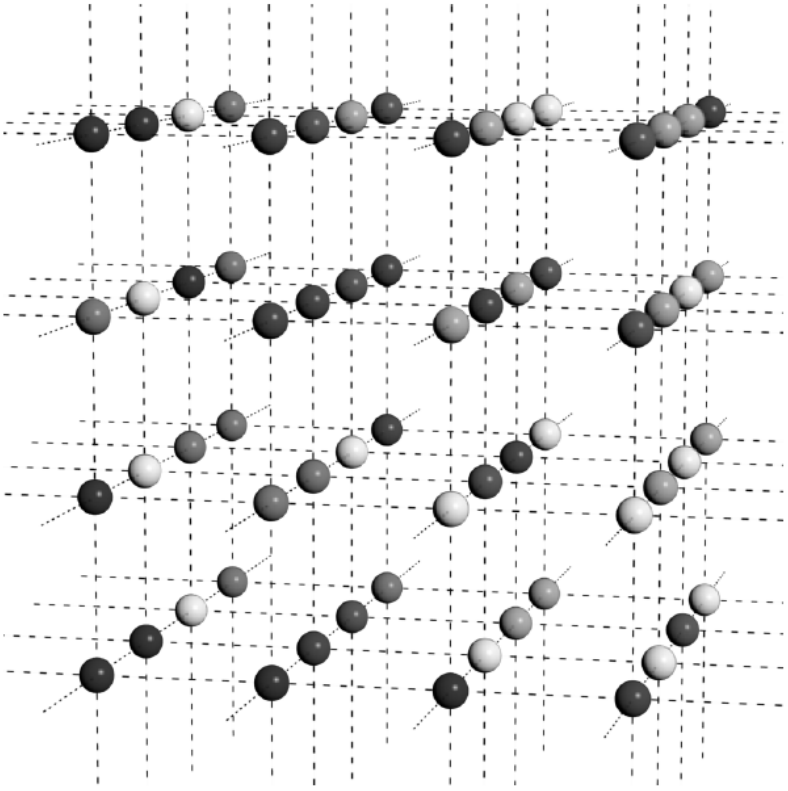

Abbildung 2.1: Ein Voxelgitter [Seibt, 2014].

2.3 Marching Cubes

Der Marching Cubes Algorithmus wurde erstmals 1988 vorgestellt ([Lorensen u. Cline, 1988]). Ziel dieses Algorithmus ist die Extraktion von Isoflächen aus Volumendaten.

2.3.1 Formale Definition

Die Extraktion von Isoflächen ist wie folgt definiert:

Aus einer Funktion $\varphi : \mathbb{R}^n \rightarrow \mathbb{R}$ wird, gegeben ein Dichtewert c ϵ $\mathbb{R}$, eine Isofläche S_c extrahiert, für die gilt:

$$S_c := \{\vartheta \; \epsilon \; \mathbb{R}^n \mid \varphi(\vartheta) = c\} \tag{2.1}$$

(vgl. [Seibt, 2014])

2.3.2 Funktionsweise

Wie der Name des Algorithmus bereits sagt wird durch die Voxel-Menge "marschiert". Die Input-Menge des Algorithmus umfasst 8 aneinander grenzende Punkte der Voxel-Menge welche zusammen einen Würfel bilden sowie eines Schwellwert für die Dichte. Nach erfolgreicher Verarbeitung wird zum nächste Würfel gewandert ("marschiert") bis die gesamte Datenmenge abgearbeitet wurde.

Vorbereitung

Wie bereits erwähnt wird der Algorithmus für jeden einzelnen Würfel der gesamten Menge angewendet. Für ein besseres Verständnis zeigt Abbildung 2.2 einen solchen Würfel (rot) in einem Voxelgitter.

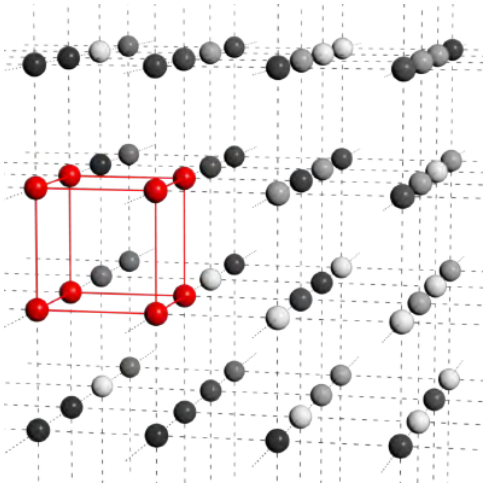

Abbildung 2.2: Ein Wüfel im Voxelgitter [Seibt, 2014].

Als erster Schritt im Algorithmus werden nun die Ecken und Kanten des Würfels für die spätere Verarbeitung indiziert (s. Abbildung 2.3).

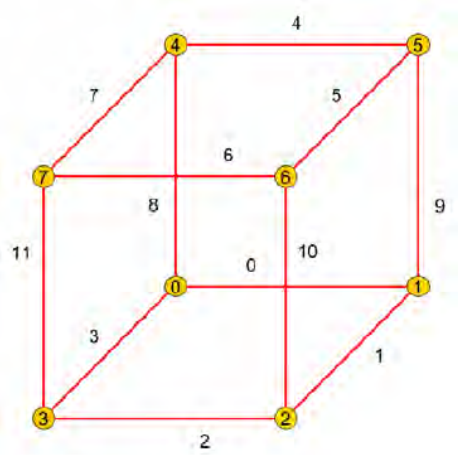

Abbildung 2.3: Indizierung eines Würfels [Seibt, 2014].

Jede Ecke des Würfels kann aufgrund seines Dichtewertes als Solide bzw. Transparent klassifiziert werden. Folglich sind aufgrund der Zwei möglichen Werte jeder Ecke $2^8 = 256$ unterschiedliche Konfigurationen der Eingabemenge möglich. Jede dieser Konfigurationen kann als ein 8 Bit Muster dargestellt werden wobei gilt, dass jedes Bit i bei welchem der Dichtewert d_i des dazugehörigen Voxel einen gewissen Schwellwert c überschreitet als binäre 0 interpretiert wird. Betrachtet man nun dieses Bitmuster als natürliche Zahl erhält man einen sogenannten Würfelindex zwischen 0 und 255 welcher für die weitere Verarbeitung essentiell ist.

Verarbeitung

2.4 Dateiformate

Die zu dieser Arbeit herangezogenen Dateiformate sind einerseits die von dem Softwarepaket Analyze[1] verwendeten Image und Header Files sowie die sogenannte STereoLithography-Schnittstelle.

2.4.1 Image File (.img)

Diese Datei ist vergleichsweise einfach aufgebaut und enthält ein Objekt bestehend aus (normalerweise) unkomprimierten Pixel Daten (vgl. [Mayo, 2015]). Jedes Pixel repräsentiert eine Voxel mit dem dazugehörigen Dichtewert. Das gesamte Objekt kann somit in eine 3x3 Matrix eingelesen und verarbeitet werde.

[1] https://rportal.mayo.edu/bir/

Programm 2.1: Header key als C-Struktur [Mayo, 2015]

```
1 struct header_key /* header key */
2 { /* off + size */
3   int sizeof_hdr /* 0 + 4 */
4   char data_type[10]; /* 4 + 10 */
5   char db_name[18]; /* 14 + 18 */
6   int extents; /* 32 + 4 */
7   short int session_error; /* 36 + 2 */
8   char regular; /* 38 + 1 */
9   char hkey_un0; /* 39 + 1 */
10 }; /* total=40 bytes */
11
```

Programm 2.2: Data history als C-Struktur [Mayo, 2015]

```
1 struct data_history
2 { /* off + size */
3   char descrip[80]; /* 0 + 80 */
4   char aux_file[24]; /* 80 + 24 */
5   char orient; /* 104 + 1 */
6   char originator[10]; /* 105 + 10 */
7   char generated[10]; /* 115 + 10 */
8   char scannum[10]; /* 125 + 10 */
9   char patient_id[10]; /* 135 + 10 */
10  char exp_date[10]; /* 145 + 10 */
11  char exp_time[10]; /* 155 + 10 */
12  char hist_un0[3]; /* 165 + 3 */
13  int views /* 168 + 4 */
14  int vols_added; /* 172 + 4 */
15  int start_field; /* 176 + 4 */
16  int field_skip; /* 180 + 4 */
17  int omax, omin; /* 184 + 8 */
18  int smax, smin; /* 192 + 8 */
19 };
20
```

2.4.2 Header File (.hdr)

Diese Datei beschreibt die Ausmaße der Pixel-Datei sowie ihre Historie. (vgl. [Mayo, 2015]).

Die genaue Struktur nach [Mayo, 2015] ist in drei Teilbereiche aufgeteilt. Der erste Teil ist der sogenannte ”header key” und beinhaltet allgemeine Informationen bezüglich der Datei (s. 2.1). Der zweite Teil beinhaltet Informationen bezüglich der Dimension der Image-Datei(s. 2.3. Der dritte und letzte Abschnitt hält Informationen bezüglich der Historie (s. 2.2).

Programm 2.3: Image Dimension als C-Struktur [Mayo, 2015]

```
1 struct image_dimension
2 { /* off + size */
3   short int dim[8]; /* 0 + 16 */
4   short int unused8; /* 16 + 2 */
5   short int unused9; /* 18 + 2 */
6   short int unused10; /* 20 + 2 */
7   short int unused11; /* 22 + 2 */
8   short int unused12; /* 24 + 2 */
9   short int unused13; /* 26 + 2 */
10  short int unused14; /* 28 + 2 */
11  short int datatype; /* 30 + 2 */
12  short int bitpix; /* 32 + 2 */
13  short int dim_un0; /* 34 + 2 */
14  float pixdim[8]; /* 36 + 32 */
15  /*
16  pixdim[]  specifies  the  voxel  dimensitons:
17  pixdim[1] - voxel width
18  pixdim[2] - voxel height
19  pixdim[3] -  interslice  distance
20   ... etc
21  */
22  float vox_offset; /* 68 + 4 */
23  float funused1; /* 72 + 4 */
24  float funused2; /* 76 + 4 */
25  float funused3; /* 80 + 4 */
26  float cal_max; /* 84 + 4 */
27  float cal_min; /* 88 + 4 */
28  float compressed; /* 92 + 4 */
29  float verified; /* 96 + 4 */
30  int glmax,glmin; /* 100 + 8 */
31 }; /* total=108 bytes */
32
```

2.4.3 STereoLithography (.stl)

> "The STL (STereoLithography) file format, as developed by 3D Systems, has been widely used by most Rapid Prototyping (RP) systems and is supported by all major computer-aided design (CAD) systems." - [Chua u. a., 1997]

Eine STL-Datei besteht im Prinzip aus einer liste von Dreiecken. Jedes Dreieck wird durch seine drei Eckpunkte im Raum (jeweils x, y und z Position) sowie durch seinen Normalvektor beschrieben. Dies führt folglich zu einer Summe von 12 Werten pro Dreieck.

Zum besseren Verständnis kann in Abbildung 2.4 der Aufbau einer solchen Datei als ASCII Darstellung betrachtet werden. Für ein besseres Verständ-

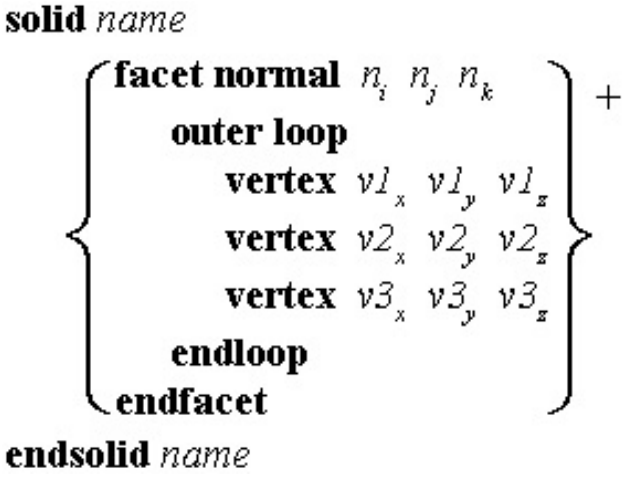

Abbildung 2.4: ASCII Darstellung des STL-Format [Fabbers, 2015].

Bytes	Data type	Description
80	ASCII	Header. No data significance.
4	unsigned long integer	Number of facets in file
4	float	*i* for normal
4	float	*j*
4	float	*k*
4	float	*x* for vertex 1
4	float	*y*
4	float	*z*
4	float	*x* for vertex 2
4	float	*y*
4	float	*z*
4	float	*x* for vertex 3
4	float	*y*
4	float	*z*
2	unsigned integer	Attribute byte count

+

Abbildung 2.5: Binäre Darstellung des STL-Format [Fabbers, 2015].

nis hinsichtlich der Implementierung ist in Abbildung 2.5 der Binäre Aufbau des STL-Formates zu dargestellt.

2.5 Computergrafik

Computergrafik beschreibt das computergestützte Erstellen und Verarbeiten von Grafiken (vgl. [Shirley u. Marschner, 2009]). In dieser Arbeit wird auf die Verarbeitung und insbesondere auf die Darstellung von dreidimensiona-

len Objekten als Polygon-Menge zurückgegriffen. Zu diesem Zweck bieten sich diverse Programmierschnittstellen wie OpenGL[2], Direct3D[3] oder AMD Mantle[4] an, welche für Grafikausgaben genutzt werden können. Aufgrund der Aufgabenstellung wird in dieser Arbeit OpenGL verwendet.

2.5.1 OpenGL

> "OpenGL (for "Open Graphics Library") is a software interface to graphics hardware. The interface consists of a set of several hundred procedures and functions that allow a programmer to specify the objects and operations involved in producing high-quality graphical images, specifically color images of three-dimensional objects." - [Segal u. Akeley, 2009]

[2] https://www.opengl.org/

[3] https://msdn.microsoft.com/en-us/library/windows/desktop/bb153256(v=vs.85).aspx

[4] http://www.amd.com/de-de/innovations/software-technologies/technologies-gaming/mantle

Kapitel 3

Umsetzung

Im Rahmen dieser Arbeit entstand ein in C++ geschriebenes Programm welches den Marching Cubes Algorithmus auf eine Voxelmenge anwendet und das Resultat via OpgenGL visualisiert. Des Weiteren ist es möglich das verarbeitete Model als STereoLithography (.stl) Datei zu exportieren.

3.1 Marching Cubes

Der Marching Cubes Algorithmus ist das Herzstück der entstandenen Applikation er ermöglicht die Umrechnung der gegebenen Voxel Datenmenge in eine polygonale Darstellung welche sich im später vergleichsweise einfach darstellen lässt.

3.1.1 Allgemein

Die Implementierung ist eine angepasst Version der von [ref] bereitgestellten Umsetzung. Die wesentlichen Änderungen sind die Auslagerung der Funktionen in eine eigenen Klasse und das verwenden anderer Datenstrukturen. Durch die Umstellung auf STL-Behälter und der daraus folgende Verzicht auf C-Strukturen welche zur Laufzeit immer neuen Speicher anfordern konnte die Geschwindigkeit enorm erhöht werde.

3.1.2 Klasse

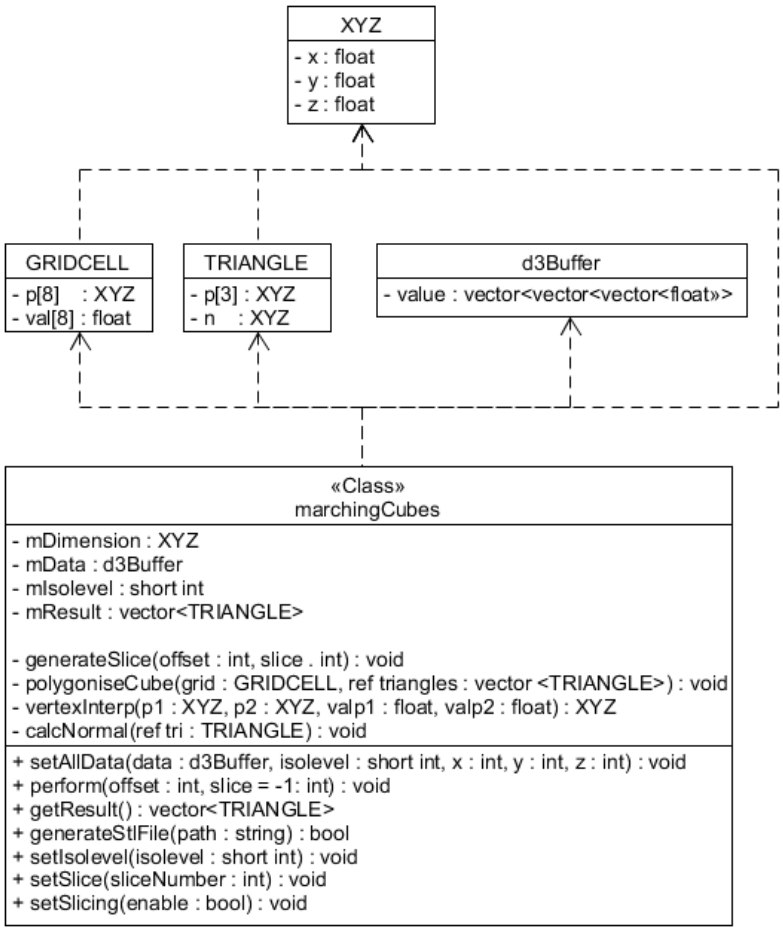

Abbildung 3.1: UML-Diagramm der marchingCubes Klasse

3.1.3 Auszüge Implementierung

3.2 File Formate

3.2.1 Allgemein

3.2.2 Image File (.img)

3.2.3 Header File (.hdr)

3.2.4 STereoLithography (.stl)

3.1

3.2.5 Klasse

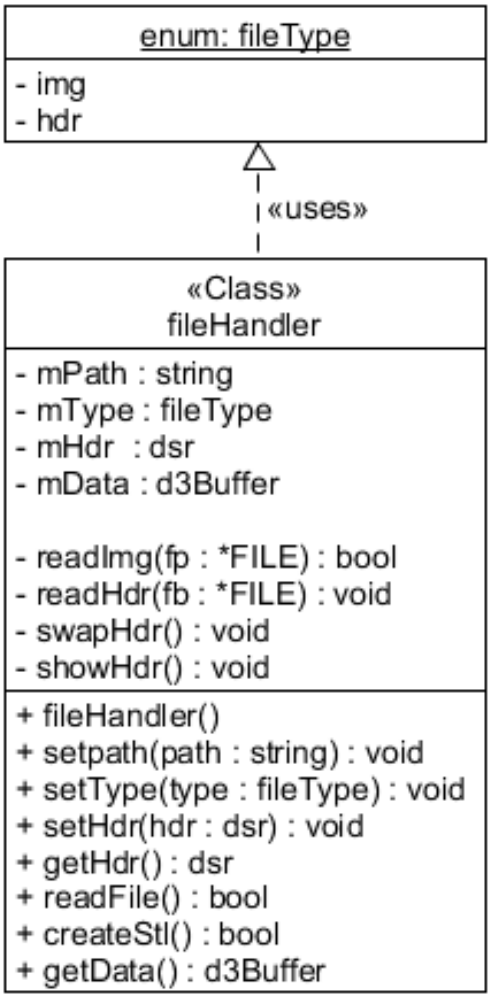

Abbildung 3.2: UML-Diagramm der fileHandler Klasse

Programm 3.1: Generierung einer STL-Datei

```
1  bool marchingCubes::GenerateStlFile(std::string path){
2    FILE *fptr = NULL;
3    fprintf(stderr, "Writing triangles ...\n");
4    if ((fptr = fopen(path.c_str(), "a+b")) == NULL) {
5      fprintf(stderr, "Failed to open output file\n");
6      return false;
7    }
8    char fileHeader[81] = "solid Test Head";
9    char bytes[3] = { 0x00, 0x00 };
10   fwrite(&fileHeader, sizeof(fileHeader)-1, 1, fptr);
11   fwrite(&ntri, sizeof(int), 1, fptr);
12   for (int i = 0; i < ntri; i++) {
13     fwrite(&tri[i].n[0], sizeof(float), 3, fptr);
14     for (int k = 0; k < 3; k++)  {
15       fwrite(&tri[i].p[k], sizeof(float), 3, fptr);
16     }
17     fwrite(bytes, 2, 1, fptr);
18   }
19   fclose(fptr);
20   return true;
21 }
22
```

3.3 OpenGL

3.3.1 Allgmein

3.3.2 Schnittstelle (Klasse)

3.3.3 Klasse

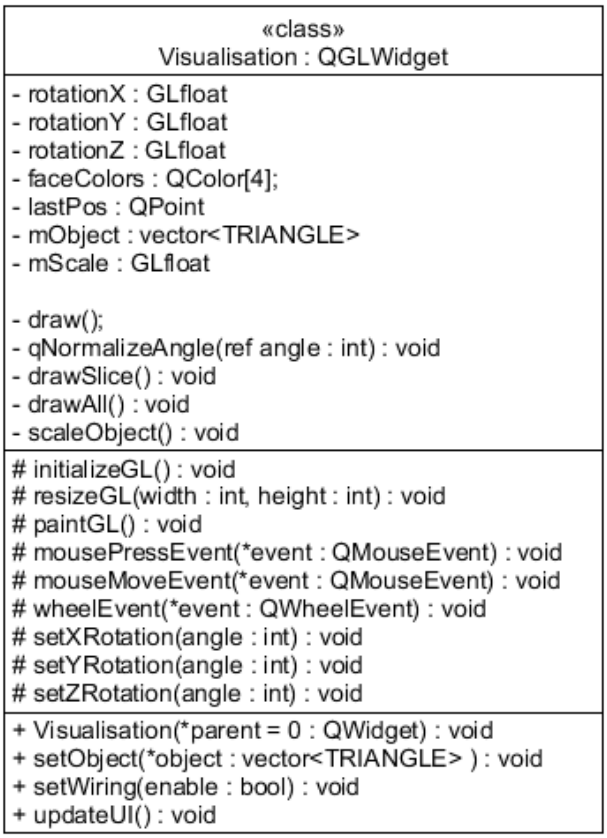

Abbildung 3.3: UML-Diagramm der Visualisation Klasse

3.3.4 Auszüge Implementierung

3.4 Benutzeroberfläche

Programm 3.2: Berechnung der Normalen eines Dreiecks

```
1  void marchingCubes::CalcNormal(TRIANGLE &tri){
2    XYZ U;
3    XYZ V;
4    U.x = tri.p[1].x - tri.p[0].x;
5    U.y = tri.p[1].y - tri.p[0].y;
6    U.z = tri.p[1].z - tri.p[0].z;
7
8    V.x = tri.p[2].x - tri.p[0].x;
9    V.y = tri.p[2].y - tri.p[0].y;
10   V.z = tri.p[2].z - tri.p[0].z;
11
12   tri.n[0].x = (U.y * V.z) - (U.z * V.y);
13   tri.n[0].y = (U.z * V.x) - (U.x * V.z);
14   tri.n[0].z = (U.x * V.y) - (U.y * V.x);
15 }
```

Kapitel 4

Zusammenfassung

Literaturverzeichnis

[Chua u. a. 1997] Chua, Kai Chee ; Gan, K. J. G. ; Tong, Mei: *Interface between CAD and Rapid Prototyping systems. Part 2: LMI — An improved interface*. 13. Springer-Verlag, 1997

[Fabbers 2015] Fabbers: *The StL Format*. http://www.fabbers.com/tech/STL_Format, 2015. – Besucht: 2015-09.08

[Hadels 2000] Hadels, Heinz: *Medizinische Bildverarbeitung*. 2. Springer-Verlag, 2000

[Lorensen u. Cline 1988] Lorensen, William E. ; Cline, Harvey E.: Marching Cubes: A High Resolution 3D Surface Construction Algorithm. In: *IEEE Conference Publications* (1988)

[Mayo 2015] Mayo: *ANALYZE 7.5 File Format*. https://rportal.mayo.edu/bir/ANALYZE75.pdf, 2015. – Besucht: 2015-09.08

[Segal u. Akeley 2009] Segal, Mark ; Akeley, Kurt: *The OpenGL Graphics System: A Specification*. 3.1. https://www.opengl.org/registry/doc/glspec31.20090324.pdf, 2009

[Seibt 2014] Seibt, Georg: *Oberflächenextraktion mittels des Marching Cubes Algorithmus*. Passau, 10 2014

[Shirley u. Marschner 2009] Shirley, Peter ; Marschner, Steve: *Fundamentals of Computer Graphics*. 3. Taylor & Francis Ltd, 2009

Anhang A

Inhalt der CD-ROM

Format: CD-ROM, Single Layer, ISO9660-Format

A.1 PDF-Dateien

Pfad: /

_DaBa.pdf	Bachelorarbeit mit Instruktionen (Gesamtdokument)

A.2 LaTeX-Dateien

Pfad: /

_DaBa.tex	Diplom-/Bachelorarbeit (Hauptdokument)
kurzfassung.tex	Kurzfassung
abstract.tex	Abstract
einleitung.tex	Kapitel 1
einfuehrung.tex	Kapitel 2
umsetzung.tex	Kapitel 3
zusammenfassung.tex .	Kapitel 4
anhang_a.tex	Anhang A ((Inhalt CD-ROM)
literatur.bib	Literatur-Datenbank (BibTeX-File)

A.3 Style/Class-Dateien

Pfad: /

hgbthesis.cls	LaTeX Class-Datei für Master- und Bachelorarbeiten

hgb.sty	LaTeX Style-Datei für alle Hagenberg-Dokumente

A.4 Implementierung

Pfad: /MarchingCubesVisualisation

/Release	Beispielprogramm und benötigte .dll Dateien
marchingCubes.h	
marchingCubes.cpp . .	
fileHandler.h	
fileHandler.cpp	
visualisation.h	
visualisation.cpp	
mainwindow.h	
mainwindow.cpp	
mainwindow.ui	
main.cpp	
dbh.h	
tabels.h	
MCV.pro	QT5 Projekt

A.5 Sonstiges

Pfad: /images

*.jpg, *.png	Original Rasterbilder

Pfad: /umlDiagrams

*.uxf	Original UML-Diagramme