Institut für Parallele und Verteilte Systeme
Abteilung Parallele Systeme
Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Studienarbeit Nr. 2307

Implementierung von Algorithmen zur CAD-Modell Rekonstruktion von CT-Bilddaten

Alexander Schuck

Studiengang:	Informatik
Prüfer:	Prof. Dr. S. Simon
Betreuer:	Dipl.-Ing. Jürgen Hillebrand
begonnen am:	18. Oktober 2010
beendet am:	19. April 2011
CR-Klassifikation:	I.3.3, I.3.5, I.4.9, J.6

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Algorithmen

Abstract

Sowohl bei der Durchführung elektromagnetischer Feldsimulationen an CAD-Modellen von passiven Baugruppen, als auch bei der Qualitätskontrolle bei der ein produziertes Bauteil mit einem vorgegebenen CAD-Referenzmodell verglichen werden soll, ist es notwendig, ein CAD-Modell so nah wie möglich den realen Gegebenheiten des Objektes anzunähern. Um bei beiden Anwendungsszenarien ein genaues Bild vom Inneren des Objektes, wie beispielsweise die genauen Positionen der Bauteile oder den exakten Verlauf von Leiterbahnen, zu bekommen, ist die Computertomographie (CT) geeignet. Im Folgenden wird die Entwicklung eines Prototyps beschrieben, der die in der CT produzierten Daten in einem zweidimensionalen Schnittebenenmodell anzeigen kann und welcher aus diesem zweidimensionalen Schnittbild mit Hilfe von Kantenerkennungsverfahren und einer CAD-Entwicklungsumgebung ein dreidimensionales CAD-Modell erstellt. Dieses so erstellte CAD-Modell ist dann der Ausgangspunkt für weitere ekletromagnetische Feldsimulationen.

Both in the process of electromagnetic field simulations on CAD models of passive microwave circuits and in the quality inspection where a produced unit is checked to a given CAD model there is a need of getting a CAD model as close to the real object as possible. Computed tomography (CT) is appropriate to visualize the inside of an object like the positions of an electronic component or the exact course of conductor paths. In the following the construction of a prototype which can present the data made by CT in a two-dimensional model of a sectional plane and produce a three-dimensional CAD model out of a two-dimensional sectional image by edge detection methods and by a CAD development environment is shown. This so produced CAD model then is a basis for further electromagnetic field simulations.

1 Einleitung

Elektromagnetische Feldmessungen aufgrund von Berechnungen an virtuellen Objekten haben gegenüber Messungen an realen Objekten den Vorteil, dass sie kostengünstig durchführbar sind. Während für die Messungen des elektromagnetischen Feldes am physischen Objekt sehr feine und kostspielige Messvorrichtungen notwendig sind, werden für die Berechnungen an CAD Modellen nur entsprechende Rechenleistung sowie Rechenzeit benötigt. Desweiteren sind für Hochfrequenzmessungen spezielle Messtechniken nötig. Bei entsprechend hohen Bandbreiten bis in den GHz-Bereich können gängige Messvorrichtungen an ihre Grenzen stoßen. Um den Vorteil der Simulationen ausnutzen zu können, muss aber zunächst ein genaues Abbild des Testobjekts geschaffen werden. Dabei reicht eine eventuelle CAD-Vorlage eines Objekts für die Simulationen an dem Objekt nicht immer aus. Jedes Bauteil eines Produktionsprozesses weist Fehler auf, die sich innerhalb einer gewissen Fehlertoleranz bewegen. Für den Vergleich der Messergebnisse zwischen dem simulierten Objekt und dem realen Objekt ist es wichtig, das nachgebildete Modell so nah wie möglich dem realen Objekt anzunähern. In [HWS] und [HWS11] wurde ein Verfahren gezeigt, wie mit Hilfe von Kantenerkennungsverfahren ein CAD-Modell aus einem Datensatz der Computertomographie erstellt werden kann. Es wurde dort auch durch Messungen gezeigt, dass die so erstellten Modelle nah an die realen Ausmaße herankommen [HWS].

1.1 Motivation

Ziel dieser Studienarbeit ist es nun, ein Programm zu entwerfen, das dem Benutzer die Rekonstruktion von Objekten aus CT-Daten in ein CAD-Modell vereinfacht. Bisher wurden mit einem CAD-Viewer einzelne Bilder wie in Abbildung 1.1 mit Hilfe von CAD-Viewern ausgewählt und als Einzelbild abgespeichert. Mit Hilfe von MATLAB wurde eine Kantenerkennung am Bild durchgeführt. Die so entstandenen Binärbilder wurden durch Skripte in eine Simulationsumgebung geladen. Diese Studienarbeit soll diese einzelnen Prozesse in einem Programm vereinen und das entwickelte Programm soll so gestaltet werden, dass dieses für spätere Anwendungen erweiterbar sein soll.

Die Anforderungen an das zu erstellende Programm sind:

- Daten einlesen
- Bild anzeigen und ausrichten, Ausrichtinformationen abspeichern und laden
- 2D-Schnitte aus Datensatz erstellen
- Kantenerkennung auf einzelne 2D-Schnitte anwenden

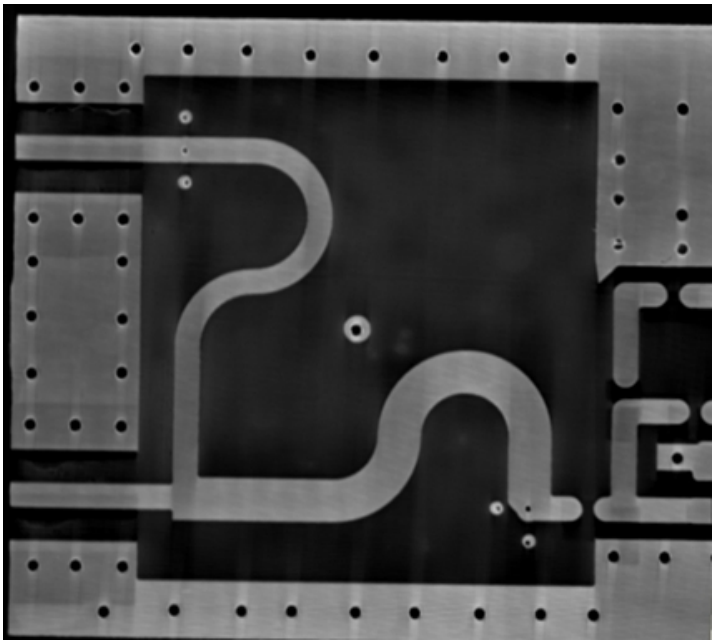


Abbildung 1.1: Schnittebene durch eine Leiterplatine.

- Einzelne Punkte zu Polygonen zusammenführen
- Polygone in CAD-Datensatz ablegen
- CAD-Datensatz anzeigen, bearbeiten
- Polygone zu 3D-Objekten transformieren
- Objekte aus einzelnen Schnittebenen zu 2.5D-Aufbau zusammenfügen
- Hinzufügen von Definitionen zu einzelnen Objekten des Datensatzes
- Materialdefinitionen und Materialeigenschaften definieren
- Datensatz der Definitionen soll erweiterbar sein, durch Anhängen von weiteren Definitionsobjekten
- Datensatz anzeigen
- Datensatz abspeichern und laden in gängigen Datenaustauschformaten
- Einzelne Aufgabenschritte aus GUI heraus aufrufbar gestalten

Ziel dieser Studienarbeit ist es, ein Programm zu schreiben mit dem es möglich ist, einen RAW-Datensatz[1] einzulesen, den Voxelsatz anzuzeigen und Schnittebenen zu definieren. Anhand der 2D-Schnittebenen sollen mit Hilfe von Kantenerkennungsverfahren die Kanten

[1]Der RAW-Datensatz beinhaltet die Werte der CT-Messung.

eines Objektes erkannt werden. Die gefundenen Kanten eines Objektes sollen miteinander zu einem Polygonzug verbunden werden, um eine zweidimensionale Darstellung des gesuchten Objekts zu erhalten. Ein Polygonzug besteht aus mehreren Linien, die die einzelnen ermittelten Kantenpunkte miteinander verbinden. Um aus dem Polygonzug eine dreidimensionale Darstellung zu erhalten, soll das Objekt entlang der z-Achse mit einer vorgegebenen Höhe h extrudiert werden können.

Gliederung

Die Arbeit ist wie folgt gegliedert:

Kapitel 2 – CAD-Modellierungssoftware: Hier wird eine passende CAD-Entwicklungsumgebung gesucht, die den oben angegebenen Anforderungen genügt. Dabei werden die zwei Entwicklungsumgebungen von Open CASCADE und SPATIAL vorgestellt und miteinander verglichen.

Kapitel 3 – Voxelsatz und Visualisierung: Es werden die Details der Computertomographie und der daraus resultierenden 3D-Voxeldaten erläutert. Die verschiedenen Klassen zur Anzeige von Voxeldaten werden vorgestellt und daraufhin untersucht, ob sie für die Kantenerkennung geeignet sind. Desweiteren wird die durch Schnittebenen realisierte Anzeige vorgestellt.

Kapitel 4 – CAD-Modell Erzeugung durch Kantenerkennungsverfahren: In diesem Kapitel werden die Grundlagen der Kantenerkennung sowie eine Einführung in die verschiedenen Klassen der Kantenerkennung gezeigt. Das in dieser Arbeit verwendete Kantenerkennungsverfahren von Canny wird vorgestellt und die Morphing- und Edge-Link-Operationen, welche zur CAD-Modell-Erstellung notwendig sind, werden erklärt.

Kapitel 5 – Entwickeltes System: Das entwickelte System wird in diesem Kapitel vorgestellt.

Kapitel 6 – Zusammenfassung: In diesem Kapitel sind die Ergebnisse der Arbeit zusammengefasst und es wird ein Ausblick auf die zukünftige Weiterentwicklung der Software gegeben.

2 CAD-Modellierungssoftware

Um die in Kapitel 1.1 angegebenen Anforderungen an die Erzeugung von 3D-Modellen aus den Schnittbildern zu erfüllen, wurden zwei Entwicklungsumgebungen für die Erstellung von CAD-Software untersucht. Dabei wurde untersucht, inwieweit sich die folgenden drei Kernanforderungen realisieren lassen:

1. Lässt sich anhand der bei der Kantenerkennung gefundenen Punktmenge ein dreidimensionales Objekt erzeugen?
2. Kann das erzeugte Objekt mit Attributen, welche die Eigenschaften des Materials aufnehmen können, versehen werden und können für spätere Zwecke noch weitere Attribute angehängt werden?
3. Werden wichtige Standardformate zum Datenaustausch mit anderen CAD-Anwendungen und elektromagnetischen Feldsimulatoren unterstützt?

Darüber hinaus wurde noch untersucht, wie gut die Dokumentation und der Support für die Entwicklungsumgebung sind und an welche Programmiersprachen diese Umgebungen gebunden sind. Die Wahl der Programmiersprache ist wichtig, um die weiteren Anforderungen wie die der Bilderkennungsverfahren sowie der Anzeige und Auswahl der Schnittebenen realisieren zu können.

Zunächst wird in Kapitel 2.1 das Open Source Projekt OPENCASCADE vorgestellt. Kapitel 2.2 stellt eine der marktführenden CAD-Entwicklungsumgebungen von SPATIAL vor. Kapitel 2.3 fasst die für diese Aufgabe relevanten Informationen beider untersuchter Entwicklungsumgebungen zusammen, vergleicht diese miteinander und liefert eine Begründung für die für diese Studienarbeit benutzte CAD-Entwicklungsumgebung.

2.1 Open CASCADE

Open CASCADE (Computer Aided Software for Computer Aided Design and Engineering) [OPE11] ist eine Open Source Entwicklungsumgebung für die Erstellung von Software für CAD, CAM und CAE. Es besteht aus einer Vielzahl von C++ Programmbibliotheken. Die Benutzung der Software ist kostenlos, der Support hingegen ist kostenpflichtig. Es werden mehrere Beispielprogramme für die Microsoft Foundation Class (MFC), Java, C♯ und Qt angeboten. In Open CASCADE gibt es eine Vielzahl von geometrischen Standardobjekten wie Punkte, Linien, Oberflächen usw. Darüber hinaus existieren mathematische Operationen

wie die Schnittbildung zwischen geometrischen Objekten oder Projektionsoperationen an Objekten.

Zur Erfüllung der ersten Kernanforderung sind schon die Standardfunktionen ausreichend. Mit Hilfe der angebotenen Standardfunktionen lassen sich Polygone in der zweidimensionalen Ebene erstellen, welche durch eine Sweep-Operation in ein dreidimensionales Objekt umgewandelt werden können.

OpenCASCADE bietet für die schnelle Erstellung einer 3D-Anwendung ein Application Framework an, das sogenannte Open CASCADE Application Framework (OCAF). Alle für die Anwendung notwendigen Daten werden in einem oder mehreren Dokumenten gespeichert. Dabei werden dem Anwendungsprogrammierer eine Vielzahl von Aufgaben wie beispielsweise das Speichern und Laden eines Dokumentes oder UNDO/REDO Funktionen für die Erstellung eines Objektes durch OCAF abgenommen. Bei vielen 3D- Entwicklungsumgebungen werden Attribute eines Objektes direkt mit der Repräsentation des Objektes abgespeichert. Als Repräsentation des Objektes dient normalerweise die Begrenzung eines Objektes, dargestellt durch beispielsweise die Koordinaten der Begrenzungslinien. Diese sind wiederum durch die Kanten eines Objektes dargestellt. OCAF geht einen anderen Weg der Zuordnung der Attribute eines Objektes. OCAF führt eine weitere Ebene der Zuordnungshierarchie durch eine schlüsselbasierte Indirektion ein. Dadurch lassen sich die Attribute eines Objektes immer noch eindeutig einem Objekt zuordnen, auch wenn sich die Topologie des Objektes ändert, ohne dass die durch die Änderung der Topologie hervorgerufene Änderung der Begrenzung des Objektes eine Neuzuordnung der Attribute notwendig macht. Die Topologie eines Objektes ist in OCAF somit auch ein Attribut. All diese Daten werden in einem OCAF-Dokument verwaltet.

Die Daten werden intern in einer gerichteten Baumstruktur verwaltet. Der Referenzschlüssel wird durch Labels realisiert. Jedes Label erhält intern einen Tag, welcher einer Integerzahl entspricht. Abbildung 2.1 zeigt die Baumstruktur für ein Dokument, das aus mehreren Lampen besteht. Jede Lampe besteht aus den Bauteilen Stiel, Lampenschirm und Glühbirne. Die Zahl innerhalb des Kreises ist der Tag. Die Wurzel hat immer den Tag 0. Labels mit demselben Vaterknoten müssen unterschiedliche Tags besitzen. Um die Labels eindeutig zu machen, wird eine Liste von Tags definiert. Die Liste setzt sich aus dem Weg von der Wurzel zum Label zusammen, wobei von jedem Knoten der Tag angehängt wird. Als Trennzeichen dient hier ":". Der Lampenschirm der Lampe Nr. 1 aus Abbildung 2.1 hätte z.B. die Tagliste $0 : 1 : 1$.

Den Labeln können nun beliebig viele Attribute angehängt werden (siehe rechteckige Ken in Abbildung 2.1). Neben den geometrischen Attributen gibt es noch die Standardattribute wie Integer-, Real- oder Stringattribute. Somit lässt sich auch die zweite Kernanforderung in Open CASCADE mit Hilfe von OCAF implementieren.

Auch die dritte Kernanforderung wird unterstützt. In Open CASCADE werden unter anderem die STEP- und IGES-Formate unterstützt. Weitere Datenaustauschformate für die Interoperabilität mit anderen Systemen finden sich in der Tabelle 2.1 der Zusammenfassung in Kapitel 2.3.

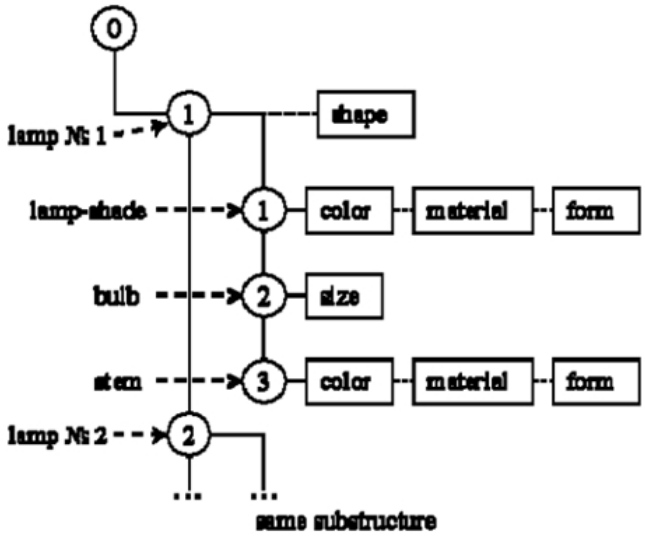


Abbildung 2.1: OCAF-Baumstruktur für eine Lampe (Quelle: [OPE11]).

2.2 SPATIAL

SPATIAL [Spa11] bietet für die Erstellung von CAD-, CAE- und CAM-Software eine Palette von Softwarekomponenten an, die es Herstellern entsprechender Software ermöglicht, in kurzer Zeit hochwertige 3D-Applikationen zu erstellen ohne diese von Grund auf selbst zu implementieren. SPATIAL ist eine der führenden CAD-Entwicklungsumgebungen und wird auch schon in vielen Simulationswerkzeugen benutzt [DGSH03]. Eine dieser Komponenten ist SPATIALs 3D ACIS Modeler. Mit Hilfe des ACIS Modelers lassen sich 3D-Modelle erstellen. Er wird in Verbindung mit anderen Komponenten wie HOOPS, welches eine grafische Anzeigenkomponente bietet, von vielen Firmen in den unterschiedlichsten Bereichen verwendet. Mit Hilfe einer eingerichteten Bridge lassen sich in ACIS modellierte Objekte in HOOPS anzeigen. Der ACIS Modeler ist vollständig in C++ Klassen geschrieben und enthält eine Vielzahl von Standardfunktionen zum Erstellen von geometrischen Modellen und stellt mathematische Hilfsfunktionen bereit.

SPATIALs ACIS Modeler und andere Produkte aus ihrem Hause sind im Gegensatz zu Open CASCADE nicht kostenlos.

Die erste der drei Kernanforderungen lässt sich mit Hilfe des ACIS Modelers einfach realisieren. Mit der WIRE-Topologieklasse lässt sich aus einer Punktmenge eine Fläche

gestalten und diese dann entlang einer Richtungsachse oder einem vorgegebenen Vektor und einer vorgegebenen Länge in eine Richtung ziehen, um einen soliden Körper zu erstellen.

Auch die zweite Kernanforderung, nach der es möglich sein soll, Attribute zu den Objekten hinzufügen zu können, um diese später dann für weitere Berechnungen am Objekt auslesen zu können, lassen sich in SPATIAL implementieren. Es wird eine eigene Klasse ATTRIB angeboten. Mit dieser Klasse lassen sich dann Attribute für die Klasse ENTITY definieren. ENTITY ist eine abstrakte Klasse aus der sich sowohl die topologischen Klassen wie z.B. die WIRE, FACE oder EDGE als auch die geometrischen Klassen wie APOINT oder CURVE ableiten. Dadurch ist es beispielsweise möglich, nicht nur einem fertigen Objekt ein Attribut zuzuordnen, sondern auch einzelnen Bestandteilen des Objektes wie Kanten oder Oberflächen. ACIS bietet eigene Datenattribute an. Der Benutzer kann seine eigenen Attributklassen von ATTRIB ableiten und nach seinen Wünschen ergänzen. Es können auch Pointer zu anderen Attributen oder anwendungsspezifischen Daten als Attribut definiert werden.

Um die letzte Kernanforderung zu erfllen, wird eine Klasse Interop realisiert, welche die durch ACIS modellierten Objekte in eine Vielzahl von Standarddatenaustauschformate exportiert und auch importiert. Eine Auflistung der unterstützten Formate findet sich in Kapitel 2.3 in Tabelle 2.1 zum Vergleich der beiden untersuchten Systeme.

Mit den Produkten von SPATIAL lassen sich die gestellten Anforderungen realisieren. Desweiteren lässt sich bei SPATIAL feststellen, dass der SUPPORT und auch die Dokumentation sehr gut sind. Es gibt zudem ein Einstiegstutorial mit vielen Codebeispielen.

2.3 Vergleich beider Entwicklungsumgebungen

Fasst man die Anforderungen einer CAD-Entwicklungsumgebung für die zu erstellende Software zusammen und vergleicht die in den obigen Kapiteln beschriebenen Fähigkeiten der beiden Systeme, so lässt sich der Schluss ziehen, dass beide Systeme für die vorliegende Aufgabe geeignet sind. Tabelle 2.1 fasst die Anforderungen und die Eigenschaften beider Systeme zusammen.

Beide Systeme besitzen eine Vielzahl von geometrischen Objekten. Bei beiden Systemen ist es möglich, aus einer Menge von Punkten im dreidimensionalen Raum die Punkte miteinander zu verbinden und daraus eine Fläche zu konstruieren. Aus dieser Fläche ist es dann bei beiden Systemen möglich, die Fläche entlang einer vorgegebenen Richtung zu einem dreidimensionalen Objekt zu ziehen. Die Anforderung zur Angabe einer Materialeigenschaft durch das Anhängen von einem oder mehreren Attributen erfüllen sowohl SPATIAL als auch Open CASCADE. Beide erlauben es dem Entwickler, Attribute für ein geometrisches Objekt zu definieren. Beide Systeme bieten die Möglichkeit, das erstellte Objekt in ein gängiges CAD-Format zu exportieren und zu importieren.

Die Untersuchung beider Entwicklungsumgebungen ergab, dass beide Systeme fr die Aufgabe gut geeignet sind. Die Entscheidung fiel letztlich auf SPATIAL. Ausschlaggebend hierfür war der sehr gute Support mit vielen Codebeispielen und Tutorials. Zudem ist SPATIAL für eine parallel abgehandelte Studienarbeit [Bra11] die bessere Wahl, da dort die schon in

Anforderung	OPENCASCADE	Spatial
3D-Objekt aus 2D-Punktmenge	ja	ja
Import Export	STEP, IGES, ACIS, Parasolid, DXF	ACIS, CATIA V4, CATIA V5, HSF, IGES, Inventor, Parasolid, Pro/E, SolidWorks, STEP, Unigraphics, VDA-FS, XML E-BOM
Attribute zu Objekten hinzufügen	ja	ja
Support	kostenpflichtig	kostenpflichtig
Beispiele	wenig	viel
Tutorials	nein	ja
Kosten	OpenSource	Lizenzgebühren

Tabelle 2.1: CAD-Entwicklungsumgebungen im Vergleich

SPATIAL vorhandenen Funktionen für die Mesh-Erzeugung genutzt werden können und es bei dem ACIS Modeler einfach ist, die geometrischen Unterobjekte aus einem Objekt zu extrahieren. Somit ist es möglich, dass die Ergebnisse dieser und der genannten Studienarbeit zu einem späteren Zeitpunkt leichter in ein Programm integriert und in Zukunft noch für andere Zwecke erweitert werden.

3 Voxelsatz und Visualisierung

In diesem Kapitel werden die Details für die Entstehung und Anzeige der durch die CT abgetasteten Voxeldaten erklärt. Es wird untersucht, inwieweit die gängigen Volumenvisualisierungsverfahren für die Auswahl einer zweidimensionalen Ebene, welche für die spätere zweidimensionale Kantenerkennung benötigt wird, geeignet sind. Kapitel 3.1 erklärt das CT-Verfahren und die Struktur der dadurch produzierten dreidimensionalen Bilddaten. Kapitel 3.2 gibt einen Überblick über die gängigen Volumenvisualisierungsverfahren und untersucht, ob diese für die gewünschte CAD-Modell-Rekonstruktion durch Kantenerkennungsverfahren geeignet sind. Die Realisierung der Anzeige und der Schnittebene für das Kantenerkennungverfahren wird in Kapitel 3.3 dargestellt.

3.1 Erzeugung der Voxeldaten

Ursprung der dreidimensionalen Bilddaten ist die Computertomographie (CT). Bei der CT wird das zu untersuchende Objekt mit Röntgenstrahlen durchdrungen, um Bilder aus dem Inneren des Objektes zu erhalten. Vorwiegend wird die CT heute in der Medizin verwendet, um beispielsweise Krebszellen zu identifizieren. In letzter Zeit werden die CT-Scanner aber auch immer häufiger in der Industrie für die Materialforschung eingesetzt. Dort können die produzierten Bauteile nach Materialfehlern im Inneren durchsucht werden, welche mit bloßem Auge von außen nicht entdeckt werden können. Die Computertomographie wird auch zur Rekonstruktion von passiven Baugruppen oder printed circuit boards (PCB) genutzt [HWS] [HWS11].

Abbildung 3.1 zeigt den Aufbau eines Computertomographen. Dieser besteht aus einer Röntgenröhre (X-Ray Source), dem zu untersuchenden Objekt (Device under Test DUT), welches sich auf einem Drehtisch befindet und einem digitalen Röntgendetektor gegenüber der Röntgenröhre. Die Röntgenröhre sendet Röntgenstrahlen in einem Kegelstrahl auf das Objekt aus. Die ausgesendeten Strahlen durchdringen das Objekt und werden innerhalb der Atome der verschiedenen Materialien des Objekts je nach Materialeigenschaft unterschiedlich stark absorbiert. Die Absorption der Röntgenstrahlen ist für jedes Atom unterschiedlich und proportional zur Kernladungszahl Z^4 [Buzo8]. Nachdem die Strahlen das Objekt durchdrungen haben, erreichen sie den Detektor. Dort wird an einer Vielzahl von Messpunkten die durch die Absorptionen gedämpfte Strahlungsintensität gemessen. Die in den Detektoren gesammelten Dämpfungswerte durch die CT können als Grauwertstufen interpretiert werden. Eine hohe Absorption in Materialen mit einer hohen Dichte wird durch einen hohen Weißanteil repräsentiert. Somit lassen sich, sofern die Materialen des untersuchten Objektes

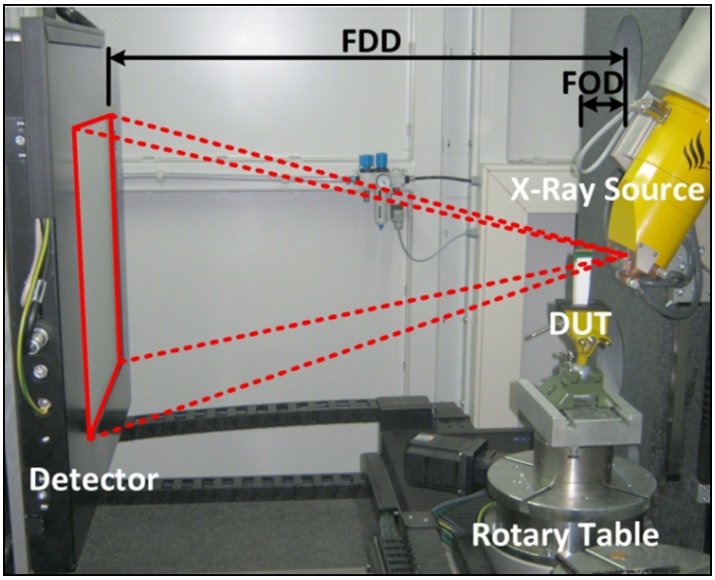


Abbildung 3.1: Aufbau eines Computertomographen [HWS11].

bekannt sind, die Grauwertstufen den einzelnen Materialien zuordnen. Der Computertomograph der in diesem Kontext verwendet wurde, hat eine Auflösung von 16 Bit, d.h. es können $2^{16} = 65.536$ verschiedene Abstufungen unterschieden werden. Gespeichert werden die Werte als 16 Bit unsigned Integer Variablen und haben somit nur positive Werte von 0 bis 65.535.

Eine einzelne Messung liefert ein zweidimensionales Dämpfungsbild. Um ein dreidimensionales Abbild des Testobjektes zu erstellen, wird das Objekt mehrfach gescannt, wobei es mit Hilfe des Drehtisches um bestimmte Winkel verschoben wird. Dies wird solange durchgeführt bis das Objekt eine vollständige Umdrehung um 360 Grad hinter sich gebracht hat. Das Ergebnis ist eine Sammlung von zweidimensionalen Bildern die am Detektor aufgezeichnet wurden.

Mit Hilfe der Radontransformation [Buzo8] lässt sich nun anhand der gesammelten zweidimensionalen Bilddaten ein dreidimensionales Abbild ermitteln. Die so gesammelten Werte lassen sich in einem dreidimensionalen Modell darstellen, dem sogenannten Voxelmodell. Ein Voxel (Volume Pixel) im dreidimensionalen Raum ist das Gegenstück zum Pixel im zweidimensionalen Raum. Das Modell besteht aus vielen Voxeln die sich räumlich aneinanderreihen. Ein Voxel hat die Form eines Quaders und kann in jeder Dimension eine andere Kantenlänge annehmen. Für die weitere Arbeit wird hier von einer einheitlichen Kantenlänge ausgegangen und die Voxel haben somit eine Würfelform. Abbildung 3.2 zeigt ein Modell eines Voxelvolumens. Jedem Voxel wird dabei ein Wert zugeordnet, welcher für die Anzeige als Grauwert interpretiert wird. Der Grauwert in diesem Kontext entspricht der Absorption in den Atomen der verschiedenen Materialen.

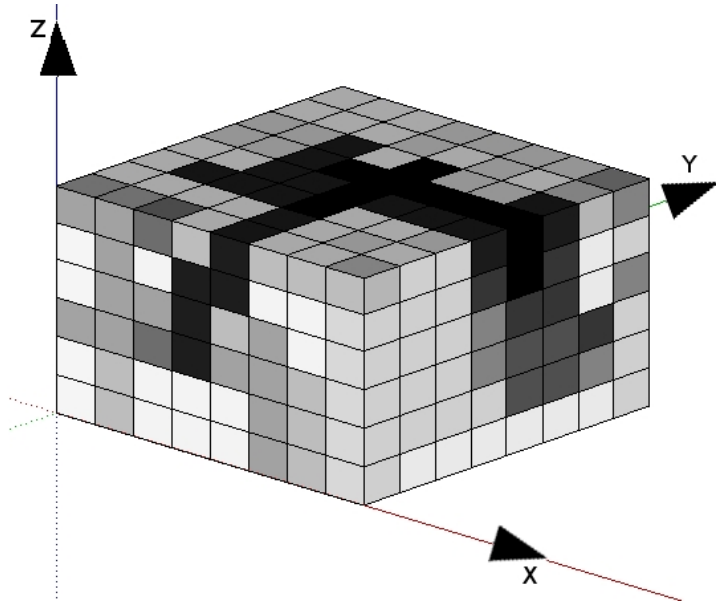


Abbildung 3.2: Schematischer Aufbau eines Voxelvolumens.

3.2 Anzeige der Voxeldaten

Die in den vorherigen Kapiteln beschriebenen Voxeldaten sollen nun für die geplante CAD-Modell-Rekonstruktion auf eine geeignete Weise am Bildschirm dargestellt werden, und in eine für die spätere Kantenerkennung geeignete Darstellung auf die Bildebene gebracht werden. In der Literatur gibt es verschiedene Verfahren wie voxelbasierte Volumen dargestellt werden können. Eine Übersicht über die Verfahren findet sich beispielsweise in [Owe99] [Elv92]. Abbildung 3.3 zeigt die Klassifizierung der allgemeinen Verfahren für die Volumenvisualisierung mit ein paar Beispielalgorithmen.

Grundsätzlich lassen sich die Verfahren in ein oberflächenbasiertes Rendering (Surface Fitting) und ein voxelorientiertes direktes Volumenrendering unterteilen [Elv92]. Die oberflächenbasierten Visualisierungsverfahren erstellen durch einen vorgegebenen Wert Oberflächen innerhalb des Voxelvolumens, indem sie das Voxelvolumen in innen und außen aufteilen. Voxel die den vorgegebenen Wert haben gehören zur Oberfläche und werden miteinander verbunden. Die voxelbasierten direkten Volumenrendering-Verfahren lassen sich in zwei Kategorien einteilen. Sie unterscheiden sich in der Richtung in welcher die Projektion durchlaufen wird. Projektionen in der Bildordnung durchlaufen die Bildebene in Scanline Ordnung und senden Sehstrahlen auf das Objekt. In der anderen Kategorie werden die Daten im Objekt durchlaufen und dann auf die Bildebene projiziert.

Das Prinzip des Konturverfahrens (siehe Abbildung 3.4) legt für ein zu renderndes Bild des dreidimensionalen Modells zuerst einen Schwellwert fest. Das Volumen ist in feste Datenscheiben (Data Slices) unterteilt. Anhand des festgelegten Schwellwertes werden für jede Scheibe die zu dem Schwellwert dazugehörigen Kanten miteinander verbunden. Für

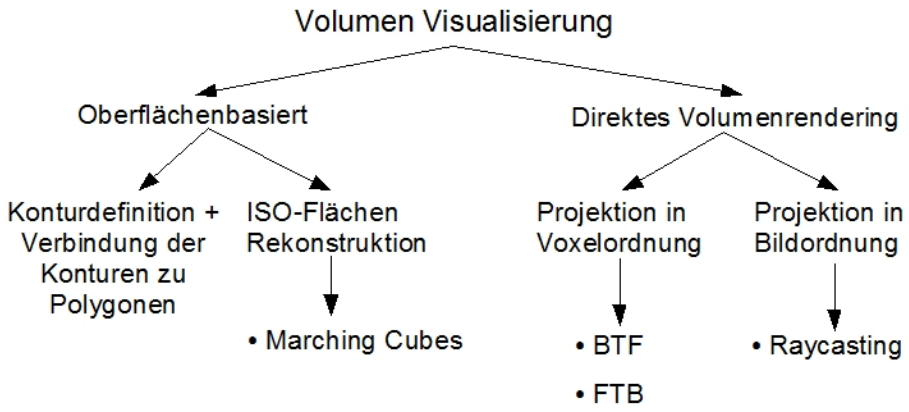


Abbildung 3.3: Klassifizierung der Visualisierungsverfahren.

die dreidimensionale Anzeige werden die zusammengehörigen Konturen der Scheiben mit Hilfe einer Triangulation verbunden und ergeben auf diese Weise das gerenderte Bild.

Eine andere Klasse der oberflächenbasierten Verfahren ist das Rendern aus Isoflächen. Isoflächen sind Flächen im dreidimensionalen Raum, die sich aus Nachbarpunkten mit gleichen Werten zusammensetzen. Das bekannteste dieser Verfahren ist Marching Cubes [LC87]. Der Marching Cube Algorithmus trennt mit Hilfe eines vorgegebenen Schwellwertes Innen- und Außenvoxel und erstellt dadurch Isoflächen aus den Voxeldaten. Das Voxelvolumen ist beim Marching Cube Verfahren in aneinanderliegende Würfel aufgeteilt. Für jeden dieser Würfel wird entschieden, wie die Oberfläche den Würfel schneidet, oder anders gesagt, wie die acht Ecken des Würfels voneinander getrennt werden können. Dabei gibt es $2^8 = 256$ verschiedene Möglichkeiten, um die acht Ecken eines Würfels voneinander zu trennen. Man kann aber zeigen, dass sich durch Ähnlichkeiten und durch Rotationen der Würfel die verschiedenen Möglichkeiten auf 15 reduzieren lassen (siehe Abbildung 3.5).

Der Vorteil der oberflächenbasierten Volumenvisualisierungsverfahren liegt in der einfachen Umsetzung der Algorithmen und das Verfahren ermöglicht eine schnelle Visualisierung des Voxelsatzes. Der Nachteil dieser Verfahren ist die Notwendigkeit eines Schwellwertes (Threshold). Um verschiedene Oberflächen zu erhalten, werden jeweils verschiedene Schwellwerte benötigt. Ein großer Nachteil dieses Verfahrens für die Anforderungen dieser Studienarbeit ist die strikte Einteilung zwischen innen und außen. Dadurch werden innere Strukturen verdeckt.

Die zweite Klasse der Volumenvisualisierungsverfahren ist das direkte Volumenrendering. Dabei fließen für das Rendern alle Voxelpunkte in die Berechnung mit ein. Diese Verfahren berechnen die Bildpunkte mit Hilfe von Projektionen. Das direkte Volumenrendering lässt sich in zwei Kategorien unterteilen. Dabei unterscheiden sich die beiden Kategorien in der Richtung der Projektion. Die Projektion kann von der Bildebene oder von dem Voxelsatz hervorgehen. Bei beiden Kategorien werden Transferfunktionen eingesetzt, die die Farb-

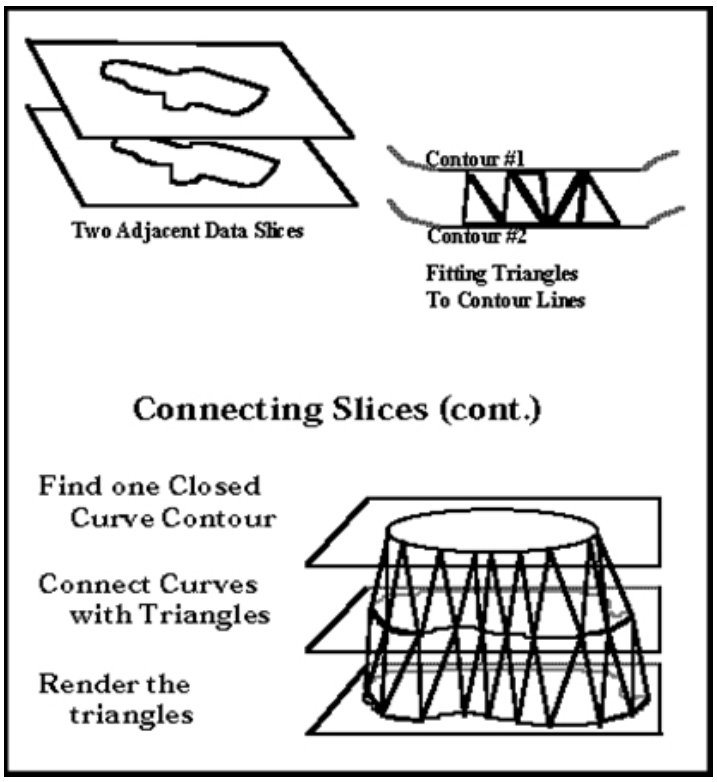


Abbildung 3.4: Prinzip der Konturverfahren [Owe99].

und Opazitätswerte von mehreren Voxeln aufsummieren um den später als Pixel in der zeidimensionalen Bildebene angezeigten Wert zu erhalten.

Das Raycasting ist ein Beispiel für die Projektion in der Bildebene. Beim Raycasting wird für jeden zu berechnenden Pixelwert ein Strahl von der Bildebene auf das Volumen ausgesendet. Der Strahl schneidet die einzelnen Voxel und kumuliert für jeden Schnitt mit einem Voxel den entsprechenden Farb- und Opazitätswert. Der Strahl läuft solange, bis der Strahl das Voxelvolumen verlässt oder bis ein vorher festgelegter Grenzwert für einen der beiden Werte erreicht wurde.

Der Vorteil des Raycastings ist es, dass auch innere Strukturen angezeigt werden können, je nachdem wie die Transparenzwerte durch die Transferfunktion eingestellt wurden . Ein Nachteil des Raycastings ist der hohe Berechnungsaufwand, da für jedes Pixel ein Strahl berechnet werden muss, welcher wiederum eine Vielzahl von Voxel schneidet.

Bei der Projektion in der Voxelordnung ist die Richtung umgekehrt. Die Voxel werden durchlaufen und ein Strahl wird senkrecht zur Bildebene auf die Bildebene projiziert. Beispiele für die Projektion in der Bildebene sind das Back to Front (BTF) (siehe Abbildung 3.6) und das Front to Back (FTB) Verfahren.

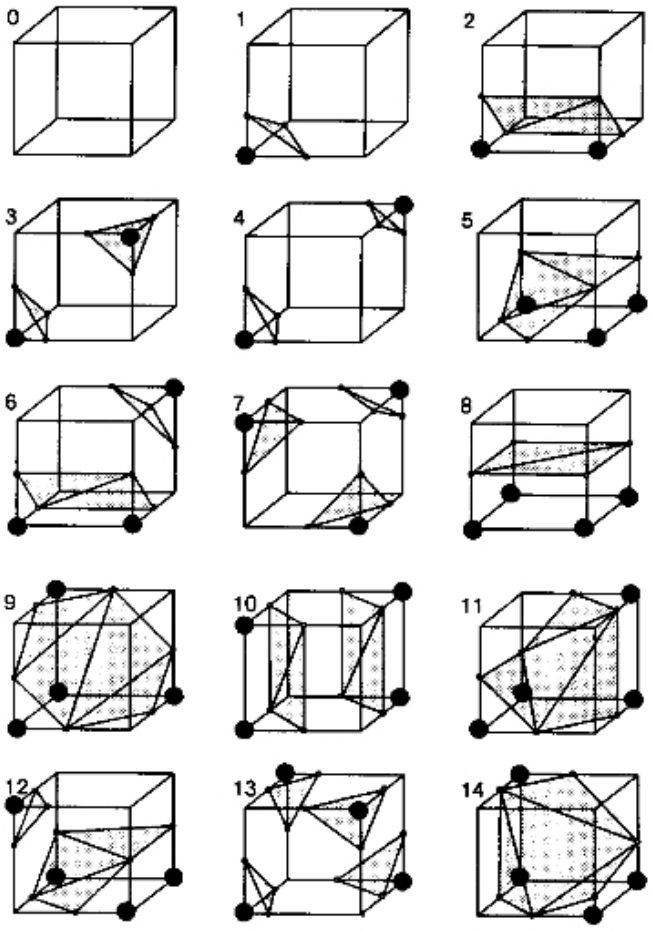


Abbildung 3.5: 15 Möglichkeiten, um die Eckpunkte der Würfel bei Marching Cubes zwischen innen und außen zu trennen [LC87].

Beim BTF Verfahren wird, wie der Name sagt, das Volumen von hinten nach vorne durchlaufen. Dabei wird mit dem der Bildebene entferntesten Voxel begonnen. Beim FTB Verfahren wird das Volumen von vorne nach hinten durchlaufen. Das hat den Vorteil, dass womöglich nicht alle Voxel bis zum Ende durchlaufen werden müssen, falls der Opazitätswert den Grenzwert schon erreicht hat. FTB Verfahren haben den Vorteil, dass der Benutzer das gewünschte Bild sukzessive aufbauen und schauen kann, welche Strukturen verdeckt werden könnten.

Eine weitere Möglichkeit für die Volumenvisualisierung ist das texturbasierte Rendering [MFW09]. Das texturbasierte Renderingverfahren wird durch die schnelle Weiterentwicklung moderner GPU für die Visualisierung immer interessanter. Es werden hierbei 3D-Volumendaten als 3D-Texturen innerhalb der GPU gespeichert. Die 3D-Texturen werden als aneinandergereihte parallele Ebenen realisiert. Diese Ebenen werden durch Polygone mit Texturkoordinaten versehen. Den Texturen werden anhand der Texturkoordinaten Datenwer-

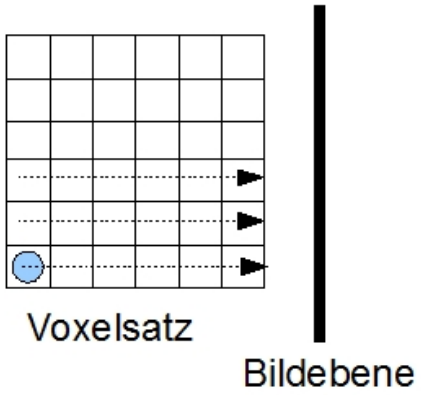


Abbildung 3.6: Back to Front Verfahren.

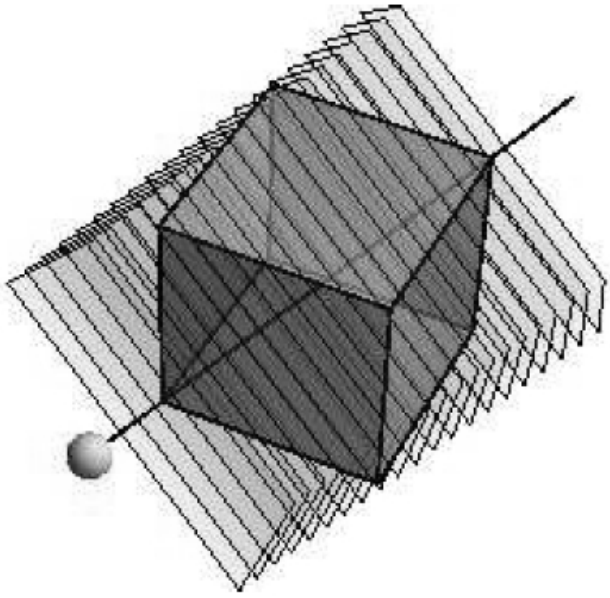


Abbildung 3.7: Texturbasiertes Renderingverfahren mit parallelen Schnittebenen. Der Sehstrahl schneidet die zu ihm senkrechten Ebenen und berechnet den Farb- und Opazitätswert [MFW09].

te des 3D-Volumens zugeordnet. Von der Bildebene werden dann wie bei den Raycasting Verfahren Sehstrahlen ausgesandt welche die Ebenen schneiden. Auch hier werden Farb- und Opazitätswert kumuliert. Abbildung 3.9 zeigt das Prinzip der texturbasierten Renderingverfahren mit den zu dem aktuellen Viewpoint senkrechten Ebenen. Die texturbasierten Renderingverfahren sind sehr speicherlastig. Das gesamte 3D-Modell muss im Speicher gehalten werden. Bei den hier verwendeten Voxeldaten ist dies ein nicht zu unterschätzender Aspekt.

Welche der vorgestellten Verfahren sind nun für die 2D-Kantenerkennung geeignet? Die oberflächenbasierten Verfahren sind gut geeignet für die schnelle Anzeige eines Voxelsatzes. Für die Kantenerkennung ist jedoch ein zweidimensionales Abbild notwendig. Möchte man eine Kantenerkennung auf der Grundlage von oberflächenbasierten Verfahren durchführen, so muss eine Projektion in die Bildebene stattfinden. Der Nachteil der oberflächenbasierten Verfahren ist, dass sie die Oberfläche nur aufgrund eines vorgegebenen Wertes aufbauen. Dadurch lassen sich nur Kantenzüge finden, die alle denselben Wert haben. Es wäre aber ein Bereichsintervall statt eines festen Wertes notwendig, da die Auflösung bei dem CT-Verfahren wie in Kapitel 3.1 geschildert bei 65.535 möglichen Werten liegt. Somit kommen diese Verfahren nicht in Frage.

Die direkten Volumenrendering Verfahren wie Raycasting, BTF oder FTB sind besser für die CAD-Modell-Erstellung geeignet. Eine Projektion in die Bildebene ist bei ihnen schon integriert. Durch die Transferfunktion lässt sich ein Bereichsintervall festlegen. Mit diesem Verfahren wäre es möglich, gute Ausgangsbilder für die spätere Kantenerkennung zu erhalten. Das Problem der Überdeckung von wichtigen Details lässt sich mit einer gut ausgewählten Transferfunktion lösen. Nachteil an diesen Verfahren ist der große Berechnungsaufwand für das Renderingverfahren, da alle Voxel in die Berechnung miteinfließen. Bei den vorliegenden Daten von mehreren hundert Millionen Voxel könnte die Berechnung somit sehr aufwändig werden.

Bei den texturbasierten Renderingverfahren werden ebenfalls keine inneren Strukturen versteckt, je nachdem wie geschickt die Transferfunktion gewählt wurde. Der Nachteil dieser Verfahren ist der hohe Speicherbedarf.

Die Entscheidung für ein für die Kantenerkennung geeignetes Visualisierungsverfahren fiel auf das texturbasierte Verfahren mit seinen Schnittebenen, jedoch mit einigen Veränderungen. Die Implementierungsdetails der entwickelten Schnittebenen werden in Kapitel 3.3 gezeigt. Der Nachteil des hohen Speicherbedarfs wurde dadurch gelöst, dass es bei den hier geforderten Anforderungen nicht auf eine korrekte dreidimensionale Anzeige der CT-Voxeldaten ankommt, sondern vielmehr auf eine für die Kantenerkennung geeignete Projektion auf eine zweidimensionale Ebene. Dadurch ist im Prinzip nur eine einzige Schnittebene notwendig und der Speicherbedarf wird dadurch gesenkt. Es können auch die aufwändigen Schnittberechnungen von Strahlen durch den gesamten Voxelsatz entfallen. Die benötigten Daten können anhand der Koordinaten der Schnittebene berechnet werden. Es wird davon ausgegangen, dass das für das CAD-Modell notwendige Schnittbild sich mit Hilfe von drehbaren Ebenen innerhalb des Voxelvolumens ausrichten lässt. Im günstigsten Fall ist bei einer guten Ausrichtung der Schnittebene und bei einer geeigneten Geometrie des Objektes, welche es erlaubt aus der Fläche durch Strecken der Fläche in die dritte Dimension

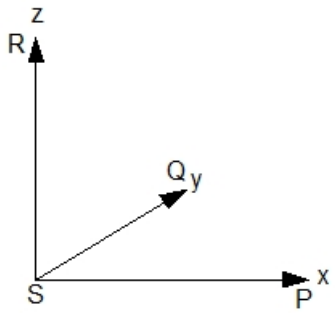


Abbildung 3.8: Endpunkte für die drei entworfenen Schnittebenen.

das gesuchte Objekt zu erhalten, nur ein einziges Schnittbild für die CAD-Rekonstruktion notwendig. Sollte dies nicht möglich sein, da das Objekt eine komplexere Struktur vorweist, so kann dennoch ein Modell erstellt werden. Es muss sich dann aus mehreren Schichten mit jeweiliger Kantenerkennung zusammensetzen.

3.3 Realisierung der Anzeige

Wie im vorherigen Kapitel schon erwähnt, wurde die Anzeige des Voxelvolumens mit Hilfe von Schnittebenen realisiert. Als Grundlage für die Anzeige wurde der Voxelsatz in ein dreidimensionales Modell gebracht. Dabei stimmen die Außenkanten des Rechteckes mit den Koordinatenachsen überein. Zusätzlich zu dem dreidimensionalen Voxelobjekt wurden drei Schnittebenen definiert. Jeweils eine Schnittebene für die XY-, XZ- und YZ-Ebene. Realisiert wurden die Schnittebenen durch Definition von drei Ebenen im dreidimensionalen Raum.

Eine Ebene wird durch zwei Vektoren aufgespannt. In Abbildung 3.8 sind die End- oder Eckpunkte des Voxelvolumens dargestellt. Der Punkt P ist der Endpunkt des Voxelvolumens auf der x-Achse, Q der Endpunkt auf der y-Achse und R der Endpunkt auf der z-Achse. Mit Hilfe dieser Endpunkte und dem Punkt S im Ursprung lassen sich die drei Schnittebenen definieren. Die XY-Ebene wird durch die Vektoren $\overrightarrow{SP}$ und $\overrightarrow{SQ}$ bestimmt. Die XZ-Ebene wird durch die Vektoren $\overrightarrow{SP}$ und $\overrightarrow{SR}$ dargestellt und die YZ-Ebene durch die Vektoren $\overrightarrow{SQ}$ und $\overrightarrow{SR}$.

Um das Ziel dieser Studienarbeit, eine genaue Rekonstruktion der Bauteile oder den genauen Verlauf von Leiterbahnen zu erreichen, müssen die entsprechenden Bauteile zuerst aus dem dreidimensionalen Modell mit Hilfe einer zweidimensionalen Schnittebene ausgewählt werden können. Daher ist es notwendig, dass eine Möglichkeit zur Navigation durch das 3D-Modell zur Verfügung steht. Die Navigation durch das 3D-Modell beinhaltet neben dem Durchlaufen der verschiedenen Schichten auch ein genaues Ausrichten der Ebenen durch Drehung der Ebenen, da es vorkommen kann, dass die gesuchten Leitungen oder Bauteile bei der CT-Erfassung nicht genau in der Aufnahmelinie lagen.

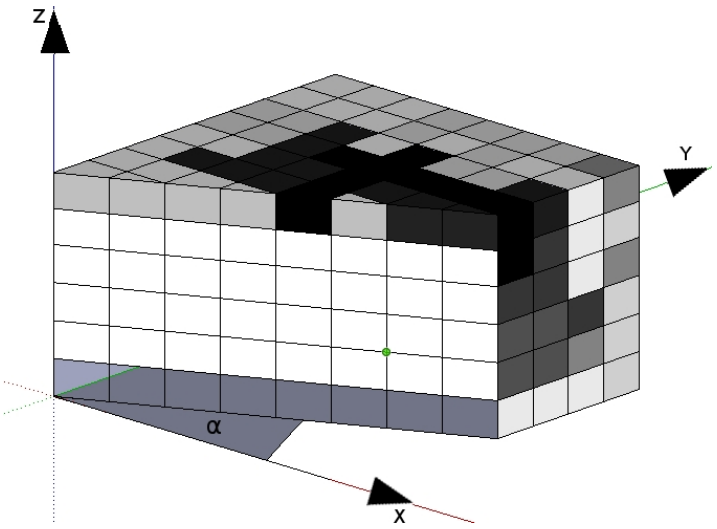


Abbildung 3.9: Schnittebene im Voxelvolumen.

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{bmatrix} \qquad R_y(\alpha) = \begin{bmatrix} \cos(\alpha) & 0 & \sin(\alpha) \\ 0 & 1 & 0 \\ -\sin(\alpha) & 0 & \cos(\alpha) \end{bmatrix}$$

$$R_z(\alpha) = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Abbildung 3.10: Rotationsoperatoren im dreidimensionalen Raum [Ant98].

Drehungen finden hier an den Ebenen und nicht am Objekt statt. Eine Drehung eines Punktes oder Vektors im dreidimensionalen Raum wird durch Drehungen um die Koordinatenachsen durchgeführt. Diese Drehungen oder Rotationen werden durch sogenannte Rotationsoperatoren durchgeführt, welche sich als Matrizen darstellen lassen [Ant98]. Eine Multiplikation eines Punktes P in der Koordinatendarstellung mit der jeweiligen Rotationsmatrix um eine Achse ergibt die neuen Koordinaten des Punktes P nach der Rotation. Abbildung 3.10 zeigt die verschiedenen Rotationsmatrizen für die Rotation um den Winkel α um die X-, Y- oder Z-Achse. Die Richtung der Drehung ist nach dem mathematischen Rechtssystem positiv, d.h. eine Drehung um ein positives α ist eine Drehung gegen den Uhrzeigersinn.

Für jede Ebene sind je ein Aufpunkt und zwei Vektoren gegeben. Soll die Ebene um den Winkel α gedreht werden, so werden die zwei Vektoren um die vorgegebene Achse durch eine Multiplikation mit der Rotationsmatrix in 3.10 gedreht. Abbildung 3.9 zeigt eine um einen Winkel α gedrehte XZ-Schnittebene welche den Voxelsatz schneidet.

Beim Drehen der Ebenen stellte sich folgendes Problem heraus. Durch mehrere Drehungen einer Ebene von einem Anfangszustand zum selben Anfangszustand zurück unterscheiden sich die Vektoren der Ebene gegenüber den Vektoren im Anfangszustand. Die Ursache liegt an Rundungsfehlern, die sich mit jeder Drehung summieren. Dies kann zu dem Problem führen, dass nach mehreren Rotationen die Vektoren durch die Rundungsfehler derart vom tatsächlichen Vektor abweichen, dass ein falscher Voxel ausgewählt wird.

Dieses Problem wurde dadurch gelöst, dass der neue Zustand nicht relativ zum alten Vektor berechnet wird, sondern durch den absoluten Winkel zum Initialvektor. Dadurch wird immer nur eine Rotationsoperation durchgeführt anstatt mehrerer relativer Rotationen.

Die Anzeige des Voxelvolumens geschieht durch eine Abbildungsfunktion, die für jeden Punkt im dreidimensionalen Raum entscheidet, ob dieser innerhalb des Voxelvolumens ist oder nicht. Ist der Punkt innerhalb des Volumens, so wird der entsprechende Voxel berechnet, der den Punkt umschließt und sein Wert wird für die Anzeige als Pixel ermittelt.

4 CAD-Modell Erzeugung durch Kantenerkennungsverfahren

In diesem Kapitel werden die Grundlagen der Kantenerkennung erläutert, welche die Grundlage für die CAD-Modell-Rekonstruktion liefern. Kapitel 4.1 gibt eine Einführung in die allgemeinen Problemstellungen, die beim Erkennen von Kanten in einem digitalen Bild beachtet werden müssen. Kapitel 4.2 beschreibt das Verfahren der Bildfilterung. In Kapitel 4.3 folgt eine kurze Übersicht über die verschiedenen Kantenerkennungsverfahren. In Kapitel 4.4 wird das in dieser Studienarbeit verwendete Kantenerkennungsverfahren von J. Canny vorgestellt. Kapitel 4.5 stellt ein Verfahren für die maximale Ausdünnung der in der Kantenerkennung produzierten Kanten vor. Als letzter Schritt wird in Kapitel 4.6 das Verknüpfen der Kanten zu geschlossenen Randobjekten gezeigt.

4.1 Allgemeine Problemstellungen in der Kantenerkennung

Eine wichtige Aufgabe der Bildverarbeitung ist es, Objekte in einem Bild vom Bildhintergrund oder von anderen Objekten im Bild zu extrahieren [MT00]. Die Begrenzungen der Objekte mit dem Hintergrund oder mit anderen Objekten werden Kanten genannt. Das menschliche Auge bewältigt diese Aufgabe im Bruchteil einer Sekunde und fehlerfrei. In der maschinenunterstützten Bildverarbeitung gestaltet sich diese Aufgabe jedoch etwas problematischer und fehleranfälliger.

Abbildung 4.1 links zeigt ein Beispielbild, in dem drei Objekte dargestellt werden. Aus diesem Bild sollen nun für die Objekterkennung die wichtigen Details (siehe 4.1 rechts), die Begrenzungslinien oder Kanten der drei Objekte, extrahiert werden. Was sind nun Kanten

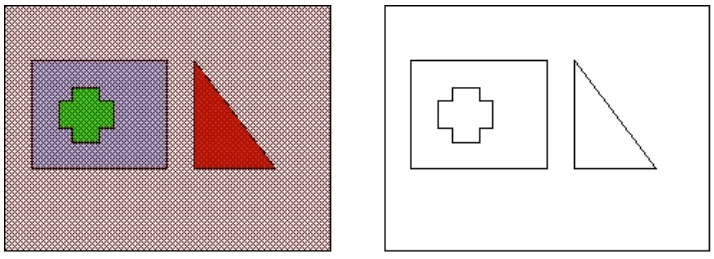


Abbildung 4.1: Aufgabe der Kantenerkennung ist es, aus einem Bild die wichtigsten Details zu extrahieren.

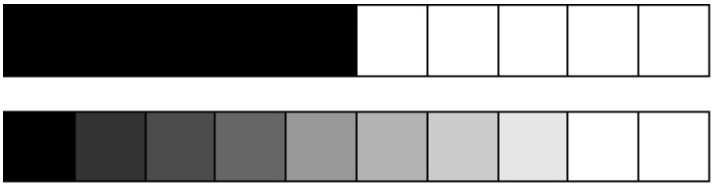


Abbildung 4.2: oben: ideale Stufenkante: Kante am Helligkeitsübergang klar erkennbar. unten: abgestufte Helligkeitsübergänge: Kante nicht klar erkennbar.

Abbildung 4.3: Problem durch Rauschen im Bild: Schwierigkeit zu entscheiden, ob es sich hier um eine Kantenline oder um zwei Kantenlinien handelt.

in der Bildverarbeitung? Kanten in einem Bild zeichnen sich durch Intensitätswechsel aus. Ein Übergang von einem hellen auf einen dunklen Wert oder umgekehrt ist ein Indiz für eine Kante. Abbildung 4.2 oben zeigt das theoretische Modell einer idealen Stufenkante. Der Übergang von schwarz auf weiß oder umgekehrt ist klar definiert. Es gibt eine klare Grenzlinie. In der Realität existiert so ein klar definierter Übergang meistens nicht, sondern der Übergang ist mehr abgestuft wie in Abbildung 4.2 unten. Hier eine klare Trennlinie zu finden ist die Aufgabe der Kantenerkennungsalgorithmen.

Eine weitere Schwierigkeit für die digitale Kantenerkennung ist Bildrauschen. In Abbildung 4.3 ist ein Beispiel für ein durch Rauschen fehlerhaftes Bild des idealen Kantenübergangs aus Abbildung 4.2 oben gezeigt. Das Problem hierbei besteht darin, zu entscheiden, ob dieses durch das Rauschen gestörte Pixel einen Kantenübergang darstellt oder noch zu der Kantenline gehört. Kanten sind Farbübergänge im Farbbild oder Helligkeitsübergänge im Graustufenbild. Ein zu hohes Rauschen in einem Bild verringert die Wahrscheinlichkeit für die korrekte Kantenerkennung und erhöht die Wahrscheinlichkeit für eine falsche Kantenerkennung, bei der eine Kante erkannt wird wo keine Kante ist. Durch Filterverfahren kann das Rauschen durch spezielle Glättungsfilter verringert werden. Bei den Glättungsfiltern ist dabei genau abzuwägen, inwieweit das Bild geglättet werden soll. Eine starke Glättung verringert auf der einen Seite die Wahrscheinlichkeit eine falsche Kante wegen Rauscheffekten zu erkennen auf der anderen Seite führt eine starke Glättung dazu dass vorherige starke Kantenübergänge sich abschwächen und dadurch eine Kante womöglich nicht erkannt wird. Im folgenden Kapitel werden die Filterverfahren für die Kantenerkennung genauer erläutert.

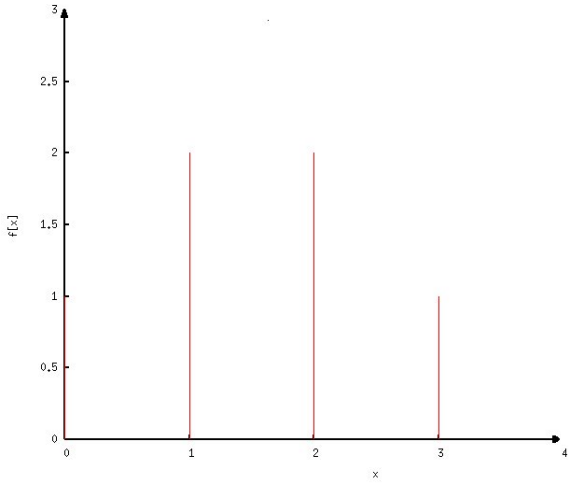


Abbildung 4.4: Beispiel für ein diskretes Zeitsignal $f = \{f[0] = 1 f[1] = 2 f[2] = 2 f[3] = 1\}$.

4.2 Filter

Filter werden dazu benutzt, um wie oben schon erwähnt, Bilder zu glätten. Sie werden aber auch bei den Kantenerkennungsverfahren als Kantendetektoren benutzt wie z.B. beim Sobel Filter[ZT98].

Um das Filterverfahren in der Bildverarbeitung zu verstehen, erfolgt eine kurze Einführung in die diskreten Zeitsignale und diskreten Systeme [Ahno6].

Ein diskretes Zeitsignal f ist eine Sequenz von Zahlen, wobei das n-te Element als x[n] bezeichnet wird. Abbildung 4.4 zeigt ein Beispiel für ein diskretes Zeitsignal mit vier Werten zu den Zeitpunkten $n = 0$ bis $n = 3$.

Die Impulsfunktion δ ist wie folgt definiert:

$$\delta[x] = \begin{cases} 1, & \text{wenn } x = 0 \\ 0, & \text{wenn } x \neq 0 \end{cases}$$

Das diskrete Zeitsignal soll nun in additive Komponenten zerlegt werden. Ähnlich der Zerlegung eines Signals bei der Fouriertransformation, bei der eine Funktion in Sinus- und Cosinusanteile aufgeteilt wird, wird das diskrete Zeitsignal mit Hilfe der Impulsfunktion in eine Summe von gewichteten und verschobenen Einheitsimpulsen dargestellt. Das Beispiel in Abbildung 4.4 lässt sich dann folgendermaßen darstellen:

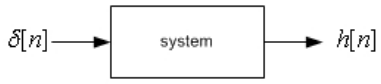
$$x[0] = x[0] \cdot \delta[n] = 1 \cdot \delta[n-0] \qquad x[1] = x[1] \cdot \delta[n-1] = 2 \cdot \delta[n-1]$$

$$x[2] = x[2] \cdot \delta[n-2] = 2 \cdot \delta[n-2] \qquad x[3] = x[3] \cdot \delta[n-3] = 1 \cdot \delta[n-3]$$

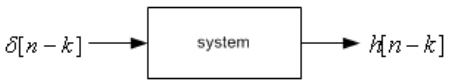
Zusammenfassend ergibt sich dann das diskrete Zeitsignal:

$$x[n] = \sum_{k=-\infty}^{\infty} x[n] \cdot \delta[x-k]$$

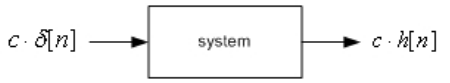
Eine Impulsantwort h ist die Ausgabe eines Systems, welches einen Impuls δ als Eingabe erhält.



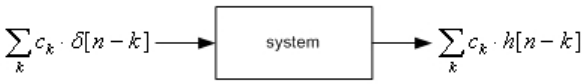
Ist das Eingangssignal zeitverschoben und handelt es sich um ein zeitinvariantes System, so ist auch die Impulsantwort zeitverschoben.



Handelt es sich desweiteren um ein lineares System, so bewirkt eine Multiplikation des Eingangssignals mit einem Skalar c, dass auch das Ausgangssignal um den Faktor c multipliziert wird.



Kombiniert man die bisherigen Ergebnisse mit der Impulszerlegung wie oben beschrieben,



so ergibt sich die Definition der eindimensionalen Faltung in der diskreten Zeitdomäne:

$$y[n] = x[n] * h[n] = \sum_{k=-\infty}^{\infty} x[k] \cdot h[n-k]$$

wobei $y[n]$ das Ausgabe- und $x[n]$ das Eingangssignal ist. Das Symbol $*$ kennzeichnet den Faltungsoperator zwischen dem Eingangssignal und der Impulsantwort h[n].

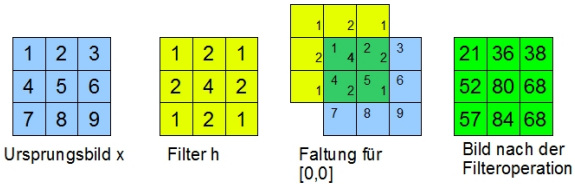


Abbildung 4.5: Beispiel für eine Filteroperation zwischen dem Ursprungsbild x und dem Filter h.

Die Faltung lässt sich auch auf den zweidimensionalen Fall anwenden. Die Gleichung lautet dann

$$y[n,m] = x[n,m] * h[n,m] = \sum_{j=-\infty}^{\infty} \sum_{k=-\infty}^{\infty} x[j,k] \cdot h[n-j, m-k]$$

Die Impulsantwort eines Systems im zweidimensionalen Raum wird Filter oder auch Kernel genannt. Ein Filter in der Bildverarbeitung ist ein Faltungsverfahren, das aus dem Eingangssignal (dem Ursprungsbild) mit Hilfe der Faltungsfunktion (Filter) ein neues Bild erzeugt.

Ein Beispiel für einen Bildfilter wird in Abbildung 4.5 dargestellt. Ein 3 x 3 Bild wird mit einem 3 x 3 Filter gefaltet. Der Filter wandert zeilenweise über das Ursprungsbild und führt für das neu zu berechnende Pixel die Faltungsfunktion aus. Dabei ist zu beachten, dass der Filter in der Filtermitte angesetzt wird. Die einzelnen Werte des Ausgangbildes lassen sich mit der oben angegebenen zweidimensionalen Faltungsformel berechnen.

Bei der Faltung muss beachtet werden, dass eine Faltungsoperation an den Randpunkten eigentlich nicht definiert ist, da für Teile der Filtermaske keine dazugehörigen Bildwerte verfügbar sind. Um dennoch den neuen Wert berechnen zu können, gibt es mehrere Möglichkeiten. Die naheliegendste und einfachste Möglichkeit ist es, die nicht vorhandenen Werte auf Null zu setzen, wie in dem Beispiel in Abbildung 4.5 geschehen. Diese Methode hat aber den Nachteil, dass dadurch an den Randpunkten ein niedriger Wert berechnet wird als bei Nichtrandpunkten. Dies könnte später bei der Kantenerkennung dazu führen, dass am Rand falsche Kanten erkannt werden. Als Lösung für dieses Problem bietet sich eine Duplizierung der Randpunkte an, so dass alle Berechnungswerte außerhalb des Bildes den Wert ihres nächsten Randpunktes in der jeweiligen Zeile oder Spalte annehmen. Dadurch werden die später eventuell falsch erkannten Kanten am Bildrand vermieden.

In der Praxis verwendet man wenn möglich separierbare Filter [BB05]. Separierbare Filter sind solche Filter, bei denen es möglich ist, die zweidimensionale Filterung in zwei eindimensionale Filter aufzuspalten.

Angenommen man hat folgenden zweidimensionalen 5 x 5 Filter H:

$$H = \begin{bmatrix} 1 & 2 & 4 & 2 & 1 \\ 2 & 4 & 8 & 4 & 2 \\ 4 & 8 & 16 & 8 & 4 \\ 2 & 4 & 8 & 4 & 2 \\ 1 & 2 & 4 & 2 & 1 \end{bmatrix}$$

Bei der Faltungsoperation eines Bildes mit dem Filter H werden für jedes Pixel im Bild $5 \cdot 5 = 25$ Multiplikationen durchgeführt. Der obige Filter kann aber auch folgendermaßen dargestellt werden:

Seien $H_x = \begin{bmatrix} 1 & 2 & 4 & 2 & 1 \end{bmatrix}$ und $H_y = \begin{bmatrix} 1 \\ 2 \\ 4 \\ 2 \\ 1 \end{bmatrix}$ jeweils eindimensionale Filter für die x- bzw. y-Richtung. Dann lässt sich H durch H_x und H_y darstellen:

$$H_y \cdot H_x = \begin{bmatrix} 1 \\ 2 \\ 4 \\ 2 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 2 & 4 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 4 & 2 & 1 \\ 2 & 4 & 8 & 4 & 2 \\ 4 & 8 & 16 & 8 & 4 \\ 2 & 4 & 8 & 4 & 2 \\ 1 & 2 & 4 & 2 & 1 \end{bmatrix} = \text{H}$$

Die Faltung mit einem zweidimensionalen Filter kann somit bei einem separierbaren Filter durch eine Faltung in x-Richtung und eine Faltung in y-Richtung dargestellt werden. Der Vorteil, der durch die Aufspaltung in zwei Richtungsfaltungen entsteht, ist eine deutliche Einsparung an Rechenoperationen. Sowohl beim Filter H_x als auch beim Filter H_y sind jeweils nur fünf Multiplikationen je Pixel nötig. Verglichen mit den 25 Operationen bei Filter H werden durch das Aufspalten in zwei Faltungen je Pixel zehn Rechenoperationen eingespart. Dadurch wird die Rechenzeit bei der Filterung verringert. Für den allgemeinen Fall eines zweidimensionalen Filters mit n Zeilen und m Spalten benötigt eine Filterung n*m Multiplikationen. Separiert man diesen Filter, so sind nur noch n+m Multiplikationen nötig. Diese Verbesserung der Performance macht sich umso deutlicher bemerkbar, je größer das Bild ist oder je größer der Filter bei der Faltung gewählt wird.

Eine Funktion, die diese gewünschte Eigenschaft der Separierbarkeit besitzt und in vielen Kantenerkennungsverfahren eingesetzt wird, ist die Gaußfunktion [Bas02].

$G(x,y) = \frac{1}{2 \cdot \pi \cdot \sigma^2} \cdot e^{-}(\frac{x^2+y^2}{2 \cdot \sigma^2})$

4.3 Kantenerkennungsverfahren

Wie schon in Kapitel 4.1 erwähnt, zeichnen sich Kanten in einem Bild durch einen Wechsel in der Intensität aus. In die Kantenerkennung ist viel Forschungsarbeit investiert worden [Hil85] [ZT98]. In der Literatur haben sich zwei verschiedene Wege durchgesetzt um Kanten zu

erkennen. Zum Einen gibt es die Gradientenverfahren. Diese berechnen die Steigungen der Kanten anhand von Ableitungsfiltern der ersten Ableitung. Sie werden daher auch als Verfahren erster Ordnung bezeichnet. Ein hoher Anstieg in der ersten Ableitung entspricht einer hohen Änderung der Grauwerte und ist dadurch ein gutes Indiz für eine Kante. Ein bekannter Vertreter dieser Verfahren ist der Canny Edge Algorithmus [Can86].

Die zweite Gruppe der Kantenerkennungsverfahren sucht Kanten anhand von Änderungen an den Kantensteigungen. Es wurde gezeigt, dass sich mit Hilfe der zweiten Ableitung Kanten als Nulldurchgänge in der zweiten Ableitung identifizieren lassen [Har84]. Durch die Anwendung der zweiten Ableitung werden diese Verfahren auch Verfahren zweiter Ordnung genannt. Ein Vertreter dieser Verfahren ist ist der Marr-Hildreth Operator [MH80].

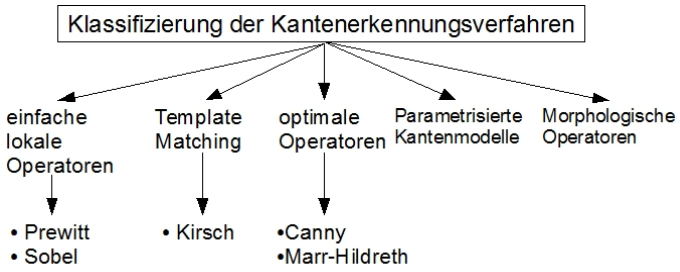


Abbildung 4.6: Klassifizierung der Kantenerkennungsverfahren nach [Ste93].

Die Verfahren der Kantenerkennung lassen sich nach [Ste93] wie in Abbildung 4.6 gezeigt in verschiedene Kategorien einteilen.

Einfache lokale Operatoren approximieren die erste oder zweite Ableitung. Meist ist noch eine Glättung integriert. Zu den einfachen lokalen Operatoren gehören beispielsweise der Prewitt- und der Sobel-Operator. Beide sind einfache Gradientenfilter.

$$\text{Prewitt Operator } P_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \; P_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Der Sobel-Operator ist ein Glättungsfilter quer zur Differenzierungsrichtung. Seine Maske wird durch eine Binomialverteilung der Gaußverteilung angenähert. Die Filter S_x für die x-Richtung und S_y für die y-Richtung lauten:

$$S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \; S_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Mit dem beschriebenen Sobel- bzw. Prewitt-Operator erhält man nach der Faltung ein Gradientenbild. Das Gradientenbild gibt die Steigung der Kanten an. Die Gradientenstärke kann mit folgender Formel berechnet werden:

$$G = \sqrt{G_x^2 + G_y^2}$$

G_x ist das Bild nach der Faltung des Sobel- bzw. Prewitt-Operators in x-Richtung und G_y das Bild nach der Faltung in y-Richtung. Die Kantenrichtung wird durch $\arctan\left(\frac{G_y}{G_x}\right)$ berechnet.

Bei den Template Matching-Verfahren werden Kantenmuster anhand von vorgegebenen Masken (Templates) gesucht. Diejenige Maske, welche am Besten zu dem untersuchten Bildausschnitt passt, wird ausgewählt und repräsentiert dann die Kante. Der Kirsch-Operator ist ein Vertreter dieser Verfahren. Er besteht aus acht verschiedenen Filtern die die acht verschiedenen Kantenrichtungen repräsentieren. Die Kantenrichtungen laufen dabei in 45°-Schritten von 0° bis 360°. Die Masken lauten beginnend von 0° :

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & -2 & 1 \\ -1 & -1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ -1 & -2 & 1 \\ -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} -1 & 1 & 1 \\ -1 & -2 & 1 \\ -1 & 1 & 1 \end{bmatrix} \begin{bmatrix} -1 & -1 & 1 \\ -1 & -2 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

$$\begin{bmatrix} -1 & -1 & -1 \\ 1 & -2 & 1 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & -1 & -1 \\ 1 & 2 & -1 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & -1 \\ 1 & -2 & -1 \\ 1 & 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ 1 & -2 & -1 \\ 1 & -1 & -1 \end{bmatrix}$$

Die optimalen Operatoren zeichnen sich dadurch aus, dass sie gegenüber den bisherigen Verfahren eine Vorstellung über die Bilddaten mitbringen, z.B. wie das Kantenmodell aussieht.

Der Canny Edge Detektor [Can86] ist einer der bekanntesten Vertreter der Gradientenverfahren und wird im folgenden Kapitel 4.4 genauer vorgestellt.

Die Kantenerkennung nach Marr und Hildtreth [MH80] ist eine Vertretung der Kantenerkennung durch Nulldurchgänge in der zweiten Ableitung. Im ersten Schritt der Kantenerkennung nach Marr-Hildreth wird das Bild durch einen zweidimensionalen Gaußfilter geglättet.

$$G(x,y) = \frac{1}{2 \cdot \pi \cdot \sigma^2} \cdot e^{-}\left(\frac{x^2+y^2}{2 \cdot \sigma^2}\right)$$

Im zweiten Schritt wird die zweite Ableitung mit Hilfe des La Place Operators ∇^2 berechnet.

$$\nabla^2 G(x,y) = \frac{1}{\pi\sigma^4}\left(\frac{x^2+y^2}{2\sigma^2} - 1\right)e^{-\left(\frac{x^2+y^2}{2\sigma^2}\right)}$$

Im letzten Schritt werden die Nulldurchgänge durch 3x3 Masken gesucht. Ein Nulldurchgang entspricht dann einer Kante.

Nach Steinbrecher gibt es noch zwei weitere Kategorien: Parametrisierte Kantenmodelle und Morphologische Operatoren. Diese werden hier nicht weiter vorgestellt und können unter [Ste93] nachgeschlagen werden.

4.4 Canny Edge Detection

J. Canny [Can86] entwickelte einen Kantenerkennungsalgorithmus, indem er die Anforderungen an einen seiner Ansicht nach optimalen Algorithmus zur Kantenerkennung auf eine mathematische Form gebracht hat. Nach seiner Auffassung muss ein optimaler Kantenerkennungsalgorithmus folgende drei Anforderungen erfüllen.

1. Geringe Fehlerrate: Die Wahrscheinlichkeit eine vorhandene Kante nicht zu erkennen soll gering sein. Ebenso soll die Wahrscheinlichkeit einer Kantenerkennung einer nicht vorhandenen Kante gering sein.

2. Gute Lokalisierung: Die durch den Operator als Kanten markierten Punkte sollen sich so nah wie möglich an den tatsächlichen Kanten befinden.

3. Nur eine Impulsantwort für eine einzelne Kante: Wenn sich im Bild nur eine Kante befindet, sollte der Kantendetektor nicht mehrere Kanten identifizieren.

Der Canny Edge Algorithmus besteht aus fünf Schritten:

1. Bild glätten
2. Kanten finden
3. Steigung finden
4. Non Maximum suppression
5. Hysterese

Die einzelnen Schritte werden im Folgenden genauer erklärt.

Im ersten Schritt des Canny Edge Algorithmus wird wie bei den meisten Kantenerkennungsverfahren das Bild zuerst geglättet um Störungen, welche die weitere Kantenerkennung negativ beeinflussen würden, zu entfernen. Als Glättungsfilter wird bei der Canny Kantenerkennung ein zweidimensionaler Gaußfilter verwendet.

$G(x,y) = \frac{1}{2\cdot\pi\cdot\sigma^2} \cdot e^{-}(\frac{x^2+y^2}{2\cdot\sigma^2})$

Die verstellbaren Parameter bei diesem Filter sind zum Einen die Standardabweichung σ und zum Anderen die Dimension des Filters. Je größer der Filter, desto unempfindlicher ist der Algorithmus bei der späteren Kantenerkennung gegenüber einzelnen Pixelfehlern aufgrund von Bildrauschen. Jedoch verschlechtert ein zu großer Filter die Lokalisierung der Kanten.

Ein zweidimensionaler Gaußfilter mit der Standardabweichung $\sigma = 1$ kann wie folgt angenähert werden:

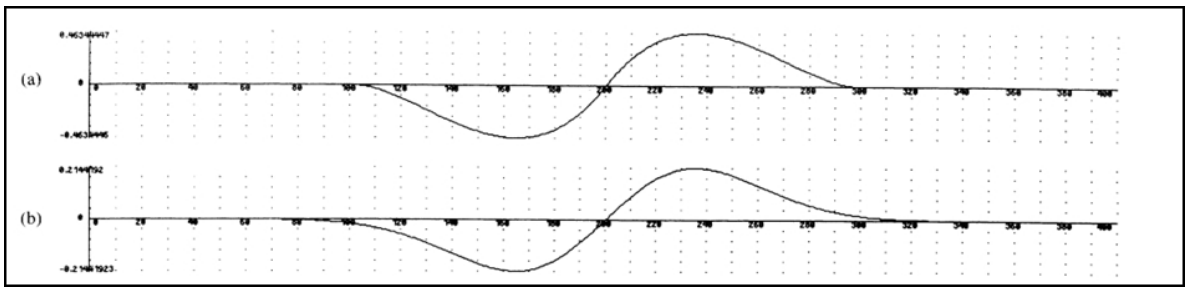


Abbildung 4.7: Vergleich zwischen hergeleitetem optimalen Operator a) und erster Ableitung der Gaußfunktion b) aus [Can86].

$$G = \frac{1}{159} \cdot \begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

Da der Gaußfilter separierbar ist, wird das Bild zuerst in die x- und dann in die y-Richtung durch einen eindimensionalen Gaußfilter geglättet.

Im zweiten Schritt des Algorithmus werden die Kanten gesucht. Die Kanten können entweder mit Sobel-Operatoren für die x- und y-Richtungen gefunden werden oder wie von Canny hergeleitet mit der ersten Ableitung der Gaußfunktion. Er hat gezeigt, dass die erste Ableitung des Gaußfilters eine sehr gute Annäherung an den von ihm definierten optimalen Operator für die Kantenerkennung liefert (siehe Abbildung 4.7). Die erste Ableitung der Gaußfunktion in x-Richtung lautet: $G' = \frac{-x}{2 \cdot \pi^2} \cdot e^{-}(\frac{x^2+y^2}{2 \cdot \sigma^2})$

Das Ergebnis der Faltung zwischen der ersten Ableitung der Gaußfunktion und dem Bild ist ein Gradientenbild. Der Gradient $\vec{D}$(x,y)=$D_x \cdot \vec{e_x}$+$D_y \cdot \vec{e_y}$ zeigt in die Richtung des größten Anstiegs. Der Gradient steht dabei immer senkrecht zur Kantenrichtung. Der Betrag des Gradienten ist ein gutes Maß für die Kantenstärke:

$$|\vec{D}| = \sqrt{D_x^2 + D_y^2}$$

Die Richtung der Kanten lässt sich durch den Winkel θ angeben.

$$\theta = \arctan(\frac{D_y}{D_x})$$

Es gibt nur vier Richtungen, die eine Kante durch den Punkt p haben kann (siehe Abbildung 4.8). Die Kante kann entweder horizontal oder vertikal verlaufen oder sie hat eine Steigung von 45° oder 90°. Der berechnete Winkel θ muss nun um die Kantenrichtung zu bestimmen auf eine dieser vier Möglichkeiten abgebildet werden. Abbildung 4.9 zeigt anhand eines Halbkreises, wie der Winkel θ auf eine der vier Kantenrichtungen abgebildet wird.

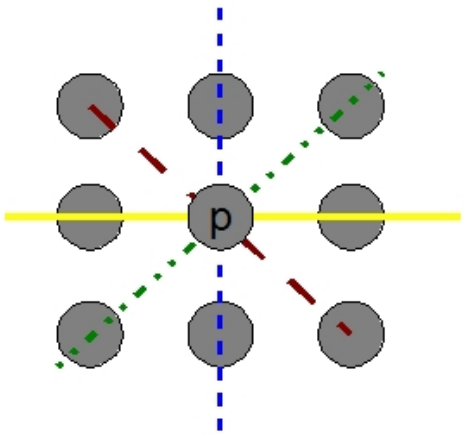


Abbildung 4.8: Eine Kante durch Punkt p kann vier mögliche Richtungen haben.

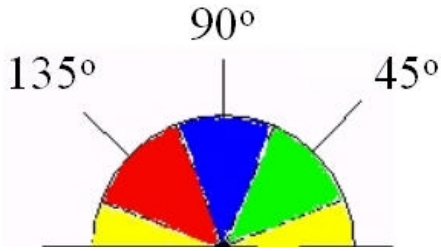


Abbildung 4.9: Abbildung der Winkel zu den vier möglichen Kantenrichtungen (aus [Greo2]).

$$Kante = \begin{cases} 0°, & \text{falls } (0 \leq \theta < 22,5) \vee (157,5 \leq \theta \leq 180) \\ 45°, & \text{falls } (22,5 \leq \theta < 67,5) \\ 90°, & \text{falls } (67,5 \leq \theta < 112,5) \\ 135°, & \text{falls } (112,5 \leq \theta < 157,5) \end{cases}$$

Der vierte Schritt im Canny Algorithmus ist das Non Maximum Suppression. Um den Verlauf einer Kante bestimmen zu können, müssen diejenigen Pixel ausgewählt werden, welche sich nahe an der maximalen Kantensteigung befinden, oder anders gesagt, diejenigen Pixel, welche sich nicht nahe an der maximalen Kantensteigung befinden, müssen unterdrückt werden. Dabei wird für jedes Pixel im Gradientenbild der Wert des Pixels mit den Werten zweier seiner Nachbarn in der Gradientenrichtung verglichen. Ausgehend von der berechneten Kantenrichtung wird das zu untersuchende Pixel p mit den beiden Nachbarpixeln, welche sich mit dem Punkt p in der zur berechneten Kantenrichtung orthogonalen Gradientenrichtung befinden, verglichen. Ist das Pixel p größer als seine beiden Nachbarn,

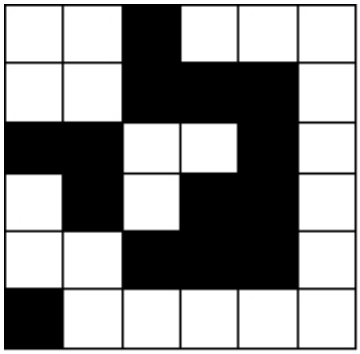


Abbildung 4.10: Beispiel für ein mögliches Binärbild nach der Kantenerkennung durch Canny. Einzelne Kantenstücke haben nicht die Breite eins. Auch ist eine einzelne, ein Pixel große Struktur vorhanden.

so ist dieses Pixel der maximalen Kantensteigung näher und bildet somit die Kante. Hat einer der beiden Nachbarn einen höheren Gradientenwert, so kann p kein Kantenpixel sein und wird unterdrückt.

Der letzte Schritt der Kantenerkennung ist die Hysterese. Bei den meisten Kantenerkennungsalgorithmen wird ein einziger Schwellwert für die Unterscheidung zwischen Kante und Nichtkante eingesetzt. In diesem Algorithmus gibt es eine untere und eine obere Schranke. Ist ein Pixel größer als die obere Schranke, so gehört das Pixel auf jeden Fall zur Kante. Solche Pixel werden als starke Kanten bezeichnet. Ist das Pixel jedoch kleiner als die untere Schranke, so gehört es definitiv nicht zur Kante. Liegt der Wert eines Pixels zwischen den beiden Schwellwerten, so wird das Pixel der Kante nur dann zugeordnet, falls einer seiner Nachbarn eine starke Kante ist. Der Algorithmus bei der Hysterese läuft durch das Bild, bis ein Pixel gefunden wird, welches größer als der hohe Schwellwert ist. Von diesem Pixel aus werden alle Nachbarpixel, welche zwischen den beiden Schwellwerten liegen, verbunden.

4.5 Morphologische Operationen

Das Ergebnis der Kantenerkennung ist ein Binärbild. Um aus diesem Binärbild nun ein Objekt zu extrahieren, muss in einem weiteren Verarbeitungsschritt dafür gesorgt werden, dass der Rand des Objektes maximal ein Pixel breit ist. Es kann nämlich nach der Canny Kantenerkennung Folgendes (wie in Abbildung 4.10 gezeigt) passieren. Einzelne Kantenstücke haben eine Breite von zwei Pixeln. Es ist damit nicht ganz klar, wie die Begrenzung des Objektes verläuft. Es muss ein Verfahren gesucht werden, mit welchem die Kanten auf eine maximale Kantenbreite von eins ausgedünnt werden können. Desweiteren ist in Abbildung 4.10 zu sehen, dass einzelne Kanten der Pixelgröße eins als Kante erkannt werden. Ein einzelnes Pixel kann aber niemals die Begrenzungslinien eines Objekts darstellen.

P8	P1	P2
P7	P	P3
P6	P5	P4

Abbildung 4.11: Namensdefinition der Nachbarschaft von Pixel P.

Um diese Probleme lösen zu können, wurde nach Problemstellungen in ähnlichen Themengebieten gesucht, um die dort vorhandenen Problemlösungen auf dieses Problem anwenden zu können. Die Thinning- oder Skeletonization-Algorithmen bieten eine Lösung für diese Probleme.

Thinning- oder Skeletonization-Algorithmen werden beispielsweise in der Zeichenerkennung benutzt, um ein Objekt auf sein Skelett zu reduzieren und dann anhand des Skeletts das Zeichen durch Pattern Matching zu identifizieren. Sie werden aber auch in ganz unterschiedlichen Bereichen wie z.B. in der Medizin bei der Bestimmung der Anzahl weißer Blutkörper eingesetzt. Thinning Algorithmen finden auch Anwendung bei der Klassifizierung von Fingerabdrücken oder bei printes circuit boards (PCB) [LLS92].

In dieser Studienarbeit sollen die Thinning-Algorithmen anstatt eine Skelettierung eines Objektes zu produzieren, dazu verwendet werden, die gefundenen Kanten auszudünnen, bis eine maximale Randbreite von einem Pixel erreicht wird. Dabei ist es wichtig, dass nach dem Ausdünnen immer noch geschlossene Kantenzüge dargestellt werden können.

Bei den Thinning-Algorithmen wird für ein Pixel entschieden, ob es gelöscht wird oder nicht. Dabei wird für das zu untersuchende Pixel die lokale Nachbarschaft betrachtet. Je nach Algorithmus wird eine Vierer- oder eine Achter-Nachbarschaft betrachtet. In Abbildung 4.11 wird die Vierer-Nachbarschaft eines Pixels P, bestehend aus den Pixeln P1, P3, P5 und P7, und die Achter-Nachbarschaft, bestehend aus den Pixeln P1-P8, gezeigt.

Die Pixel werden in mehreren Iterationsschritten solange gelöscht, bis es keine Änderungen mehr am Bild gibt. Dabei ist der aktuelle Iterationsschritt je nach Art des Algorithmus von den vorherigen Iterationsschritten mehr oder weniger abhängig. Die Thinning Algorithmen lassen sich in sequentielle Algorithmen und parallele Algorithmen unterteilen. Bei den sequentiellen Algorithmen hängt der n-te Iterationsschritt von allen vorherigen n-1 Iterationen ab. Für die parallelen Algorithmen gilt die Abhängigkeit nur jeweils dem vorherigen Schritt.

Eine wichtige Anforderung an die Thinning Algorithmen ist, dass Endpunkte und zusammenhängende Pixel nicht gelöscht werden. Endpunkte sind Punkte, die nur einen Nachbarn mit demselben Wert haben. Der Zusammenhang der Pixel wird bei den Thinning Algorithmen durch eine Konnektivitätszahl ausgedrückt. Sie gibt den Grad an, wie das zu untersuchende Pixel p mit den anderen Pixeln des Objekts verbunden ist.

In der Literatur haben sich zwei Definitionen der Konnektivität durchgesetzt. Die erste Definition von Rutovitz zählt die Übergänge von schwarz auf weiß oder von weiß auf schwarz wenn man die Nachbarn des zu untersuchenden Pixels, wie in Abbildung 4.11 gezeigt, in der Reihenfolge P1, P2, ..., P8, P1 durchläuft. Ein Pixel kann in einer Vierer-Nachbarschaft gelöscht werden, falls $X_{Rutovitz}(p) = 2$.

$$X_{Rutovitz}(p) = \sum_{i=1}^{8} |x_{i+1} - x_i|$$

Die zweite Definition von Hilditch zählt für die Konnektivität die Anzahl der 01 Muster für das Durchlaufen der geordneten Nachbarpixel P1, P2, ..., P8, P1.

$$X_{Hilditch}(p) = \sum_{i=1}^{4} b_i$$

wobei

$$b_i = \begin{cases} 1, & \text{wenn } x_{2i-1} = 0 \wedge ((x_{2i} = 1) \vee (x_{2i+1} = 1)) \\ 0, & \text{sonst} \end{cases}$$

Ein Pixel kann gelöscht werden falls $X_{Hilditch}(p) = 1$.

Im Folgenden wird der hier in der Studienarbeit benutzte parallele Thinning Algorithmus vorgestellt. Bei dem Algorithmus von Zhang und Suen [ZS84] wird jeder Iterationsschritt in zwei Teiliterationsschritte aufgeteilt. Im ersten Iterationsschritt wird ein Pixel p gelöscht, wenn die folgenden Bedingungen alle zutreffen.

(1) $2 \leq B(p) \leq 6$

(2) $A(p) = 1$

(3) $P1 * P3 * P5 = 0$

(4) $P3 * P5 * P7 = 0$

Im zweiten Iterationsschritt werden die Regeln drei und vier aus dem ersten Iterationsschritt durch folgende Regeln ausgetauscht:

(3') $P1 * P3 * P7 = 0$

(4') $P1 * P5 * P7 = 0$

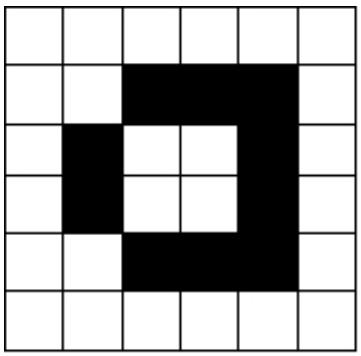


Abbildung 4.12: Das Binärbild aus Abbildung 4.10 nach Anwendung des Thinning Algorithmus.

B(p) ist die Anzahl der von 0 verschiedenen Nachbarn von Pixel p. A(p) ist die Anzahl der 01 Muster wenn man die Nachbarn von Pixel p in der Reihenfolge P1, P2, $\cdots$, P8, P1 durchläuft. Die erste Bedingung besagt, dass die Anzahl der Nachbarn mit dem Wert 1 mindestens zwei und höchstens sechs sein darf. Ist B(p)=0, so handelt es sich um ein isoliertes Pixel. Bei diesem Algorithmus wird davon ausgegangen, dass isolierte Pixel schon vorher durch Glättungsverfahren oder ähnliches entfernt wurden. In dieser Studienarbeit werden isolierte Pixel bei der Thinning-Prozedur ignoriert, da sie nach dem folgenden Edge Link Verfahren entfernt werden. Der Grund dafür ist, dass ein einzelnes Pixel kein räumliches Objekt darstellen kann. Ist B(p)=1, so handelt es sich bei dem Pixel p um einen Endpunkt. Dieser darf nicht entfernt werden. Ist B(p) größer als sechs, so würde durch das Entfernen von p ein Loch im Muster entstehen.

Die zweite Funktion A(p) gibt die Konnektivität an. Das Pixel darf nur entfernt werden, wenn die Konnektivität einen Grad von 1 hat.

Abbildung 4.12 zeigt das in Abbildung 4.10 gezeigte Beispiel nach Ausführung des Thinning Algoritmus von Zhang und Suen.

4.6 Edge Link

Nach der Kantenerkennung und nach den Thinning Operationen bleibt ein binäres Bild der Konturen mit einer maximal ein Pixel breiten Kontur der Objekte übrig. Im letzten Verarbeitungsschritt müssen nun die Punkte der Kanten miteinander zu einem Objekt verbunden werden. Dabei sollen die Objekte durch einen geschlossenen Pfad von einem beliebigen Kantenpunkt entlang der Kanten zurück zum Startpunkt dargestellt werden.

Im Folgenden wird nun der in dieser Studienarbeit verwendete Edge Link Algorithmus vorgestellt. Er wurde von Peter Kovesi [Kov] für die Edge Link Funktion von MATLAB entwickelt. Algorithmus 4.1 auf Seite 47 zeigt den Edgelink Algorithmus. Innerhalb dieser Prozedur wird die Prozedur Trackedge aufgerufen, die für die Kantenverfolgung zuständig

ist und die in Algorithmus 4.2 auf Seite 48 beschrieben wird. Abbildung 4.13 zeigt anhand eines Beispiels die Funktionsweise des Algorithmus.

Der Algorithmus durchläuft das Bild zeilenweise bis das erste Pixel einer Kante (eine 1) gefunden wird. Danach startet die Suche nach den Nachbarknoten des gefundenen Pixels. Diese werden in eine Liste aufgenommen und einer davon als nächster Punkt ausgewählt. Das aktuelle Pixel wird in eine Kantenliste aufgenommen und mit einer fortlaufenden Nummer markiert, um erstens zu verhindern, dass der Edge Link Algorithmus zu einem schon besuchten Pixel zurückkehrt und dadurch Schleifen entstehen, und zweitens werden die Pixel nummeriert um die Zugehörigkeit der Pixel zu einem Kantenzug zu kennzeichnen. Als Markierung wird eine fortlaufende Nummerierung beginnend mit 1 gewählt. Diese wird bei der Markierung negiert um keine Probleme mit den Kantenpixeln des Ursprungbildes zu bekommen. Mit dem als nächsten gewählten Punkt wird nun fortgefahren und weitere noch nicht besuchte Nachbarpixel gesucht. Das Durchlaufen entlang der Kante in eine Richtung wird solange durchgeführt, bis der aktuelle Punkt ein Verzweigungspunkt ist, oder bis keine weiteren Kanten mehr gefunden werden können. Ein Verzweigungspunkt zeichnet sich dadurch aus, dass es mehr als eine Möglichkeit gibt, wie die Kante fortgesetzt werden kann. Erkennen lässt sich ein Verzweigungspunkt mit der Konnektivitätszahl aus den Thinning-Methoden aus Kapitel 4.5. Hat ein Punkt eine größere Konnektivitätszahl als 2, so handelt es sich um einen Verzweigungspunkt. Dabei wurden die 01 Übergänge gezählt. So hat z.B. das Pixel $(3,4)$ im Beispiel in Abbildung 4.13 eine Konnektivitätszahl von 3.

Wird bei der Tracking Prozedur ein Verzweigungspunkt entdeckt, so wird die Suche nach weiteren Kantenpunkten in dieser Richtung abgebrochen. Die Suche wird in der anderen Richtung vom Ausgangspunkt aus fortgesetzt unter der Bedingung, dass der Ausgangspunkt selbst kein Verzweigungspunkt ist. Die in der einen Richtung gesammelten Kantenpunkte in der Kantenliste werden umgedreht, so dass die in der neuen Richtung gesammelten Punkte in der richtigen Reihenfolge angehängt werden können. Die Suche in der neuen Richtung läuft wieder so lange bis entweder ein weiterer Verzweigungspunkt gefunden wird oder kein weiterer Punkt mehr vorhanden ist.

Am Ende entsteht so eine Liste mit Kantenlisten. Die Listen für das Beispiel aus Abbildung 4.13 sieht wie folgt aus:

$EdgeList[1] = \{(3,4);(2,4);(1,3);(2,2);(3,2);(3,3)\}$
$EdgeList[2] = \{(1,6)\}$
$EdgeList[3] = \{(4,4);(5,5);(6,5);(6,4);(6,3);(5,3)\}$

Kantenlisten mit weniger als vier Kanten werden ignoriert, da mindestens vier Punkte für die Gestaltung eines Objektes notwendig sind.

Die Endpunkte in den Kantenlisten werden mit den Startpunkten der Kantenliste verbunden, sofern die beiden Punkte benachbart sind. Falls dies nicht zutrifft, so entstehen nicht geschlossene Kantenzüge. Solche Kantenzüge wurden in dieser Studienarbeit nicht bei der Erstellung als CAD-Modell beachtet. Eine Lösung von nicht geschlossenen Kantenlisten ist Teil einer zukünftigen Weiterentwicklung des Systems.

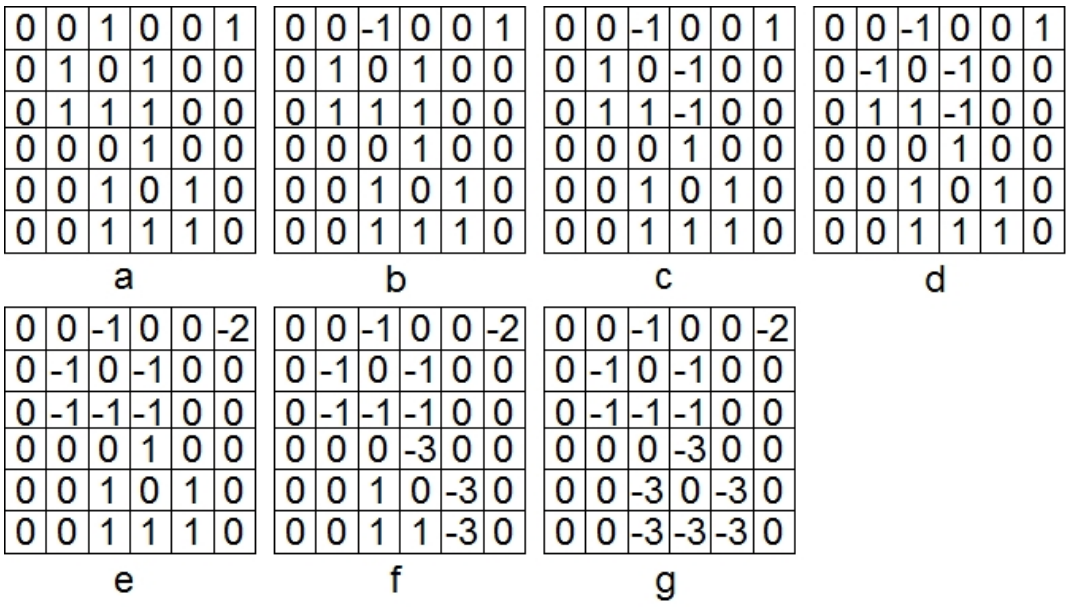


Abbildung 4.13: Edge Link Beispiel in den verschiedenen Phasen des Algorithmus a) Anfangszustand b) erstes Kantenpixel wurde gefunden c) Verzweigungspunkt wurde gefunden d) Zurück zum Ausgangspunkt und Suche in anderer Richtung e) erster Kantenzug vollständig erkannt, zweiter Kantenzug besteht nur aus einem Pixel f) Zwischenzustand beim Suchen des dritten Polygonzuges g) Endzustand.

Algorithmus 4.1 Edge link

```
procedure EdgeLink
    EdgeList[] = ∅                                  // Speichert Liste der Polygone
    EdgeNo = 1                                      // gibt die aktuelle Kantennummer an
    EdgePoints = ∅
    for all rows do
        for all cols do
            if image[row][col] = 1 then             // Noch nicht besuchte Kante
                EdgePoints ← TrackEdge(row,col,edgeNo)
                if EdgePoints ≠ ∅ then
                    EdgeNo = EdgeNo + 1
                    EdgeList[EdgeNo] ← EdgePoints
                end if
            end if
        end for
    end for
end procedure
```

Algorithmus 4.2 TrackEdge

```
procedure TRACKEDGE(StartRow,StartCol,EdgeNo)
    nextpoint = [StartRow, StartCol]
    //Finde den nächsten Kantenpunkt der mit dem aktuellen Punkt verbunden ist
    nextpoint = NEXTPOINT(StartRow, StartCol, EdgeNo)
    while nextpoint ≠ ∅ do
        EdgePoints ← nextpoint
        image[nextpoint] = −EdgeNo                    // markiere besuchte Kante
        if (nextpoint is a Junction Point) then   // ist aktueller Punkt eine Verzweigung?
            break
        else
            EdgePoints ← nextpoint
        end if
    end while
    //Nun wird vom Ausgangspunkt in die andere Richtung gesucht falls dieser
    //kein Verzweigungspunkt war
    if [StartRow, StartCol] ≠ JunctionPoint then
        nextpoint = NEXTPOINT(StartRow, StartCol, EdgeNo)
        while nextpoint ≠ ∅ do
            EdgePoints ← nextpoint
            image[nextpoint] = −EdgeNo                // markiere besuchte Kante
            if (nextpoint is a Junction Point) then // ist aktueller Punkt eine Verzweigung?
                break
            else
                EdgePoints ← nextpoint
            end if
        end while
    end if
end procedure
```

5 Entwickeltes System

In diesem Kapitel wird das entwickelte System vorgestellt. Es werden die wichtigsten entwickelten objektorientierten Klassen gezeigt, die grafische Benutzeroberfläche vorgestellt und der Ablauf für die Erstellung eines CAD-Modells aus einer CT-Datei erläutert.

Bei der Wahl der Programmiersprache fiel die Entscheidung auf C++. Durch die objektorientierten Eigenschaften von C++ ist es möglich, die benötigten Funktionen für die Anzeige und Kantenerkennung auf Klassen zu verteilen und die Vorteile der objektorientierten Programmierung wie Vererbung und Geheimhaltung voll auszuschöpfen. Die Unterteilung in Klassen hat auch einen modalen Aspekt, so dass einzelne Funktionen wie die Wahl des Kantenerkennungverfahrens oder das Thinningverfahren jederzeit schnell ausgetauscht oder ergänzt werden können.

Für die Anzeige der Schnittebenen im Voxelsatz wurde nach einer Lösung gesucht, welche es erlaubt, in einem späteren womöglich größeren Programm eingebunden zu werden. Die Anzeige und Auswahl der Schnittebenen bis hin zu Erstellung der CAD-Modelle wäre in so einem Fall nur ein Aufruf in einem komplexeren Verarbeitungsprozess. Um dies zu ermöglichen, wurden die Anzeige und die CAD-Rekonstruktion als eigener Dialog realisiert. Als Entwicklungsumgebung wurde Visual Studio verwendet. Darin integriert ist die Microsoft Foundation Class (MFC). Mit Hilfe der MFC ist es möglich, fensterbasierte Anwendungen für Windowssysteme zu erstellen.

Es folgt eine kurze Erläuterung über die wichtigsten entwickelten Klassen.

NewBoxDlg: verwaltet die für den Dialog notwendigen Daten.

Vector und Plane: Klassen zum Erstellen und Drehen der drei Schnittebenen.

CannyEdge: liefert Methoden für die Canny Edge Detection.

Thinning: implementiert den Thinning Algorithmus.

Im Folgenden wird nun der Ablauf der CAD-Modell-Rekonstruktion aus den CT-Datensätzen beschrieben. Eine der Kernanforderungen war, dass die einzelnen Schritte aus einer GUI aus aufrufbar sind. Die Schritte lassen sich folgendermaßen unterteilen:

1. Datensatz laden
2. Gewünschte Schnittebene suchen
3. Kantenerkennung starten
4. Kanten ausdünnen

5. Kanten verbinden
6. CAD-Modell erstellen

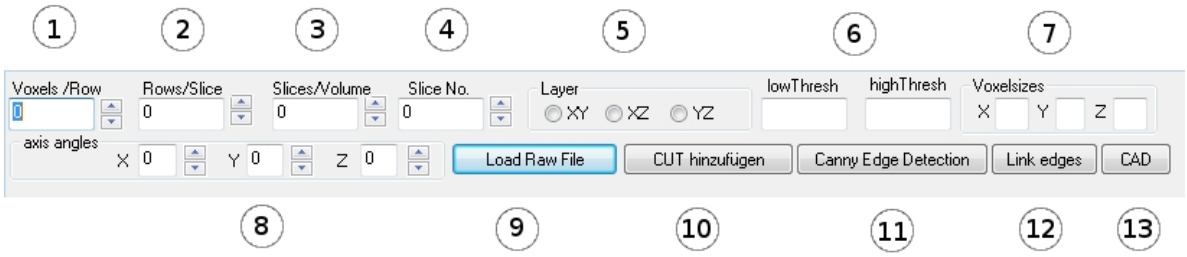


Abbildung 5.1: Schaltflächen des entwickelten Systems
1 = Anzahl der Voxel in x-Dimension 2 = Anzahl der Voxel in y-Dimension 3 = Anzahl der Voxel in z-Dimension 4 = aktuelles Schnittbild für das Durchlaufen der Schnittebenen 5 = Radiobuttons für das Umschalten der aktuellen Schnittebene 6 = obere und untere Schranke für den Canny Edge Algorithmus 7 = Voxelbreite Δx Δy Δz 8 = Drehwinkel der Ebene um die x-, y-, z-Achse 9 = Datensatz laden 10 = Auswahl der aktuellen Anzeige als CUT-Ebene 11 = Durchführung des Canny Edge Algorithmus 12 = gefundene Kanten zu Kantenlisten hinzufügen 13 = CAD-Modell aus Kantenlisten erstellen.

Die Schaltflächen des entwickelten Systems werden in Abbildung 5.1 dargestellt. Für die Darstellung eines Datensatzes ist es notwendig, die Länge, Breite und Höhe des Objekts anzugeben. Um für die Anzeige von verschiedenen Voxeldatensätzen flexibel zu sein, da oft in den Datensätzen keine Informationen über die Dimensionsgrößen des Voxelsatzes vorliegen, können mit den unter den Nummern 1 bis 3 angegebenen Editfeldern die jeweilige Voxelanzahl in den drei Dimensionen eingegeben werden. Das Voxelvolumen wird dabei so in ein dreidimensionales Modell eingepasst, dass es es im Ursprung beginnt und die Kanten des Voxelvolumens entlang der x-, y- und z-Achsen entlanglaufen. Die Endpunkte der Kanten sind wie in Abbildung 3.8 in Kapitel 3.3 gezeigt. Der Betrag des Vektors $\overrightarrow{SP}$ entspricht dabei der Anzahl der Voxel in x-Richtung mal der Voxelbreite Δx. Analoges gilt für die anderen Vektoren $\overrightarrow{SQ}$ und $\overrightarrow{SR}$. Die Voxelbreiten Δx, Δy und Δz lassen sich in den Editfeldern unter Nummer 7 eingeben. Bei dem bisherigen implementierten System wird angenommen, dass die Voxelbreiten die Länge 1 haben.

Geladen wird ein Datensatz durch den Button 9. Nachdem ein RAW-Datensatz ausgewählt wurde und die passenden Voxeldimensionen eingegeben sind, kann (wie in Abbildung 5.2 gezeigt) der Datensatz betrachtet werden. Durch die Radiobuttons der Nummer 5 aus Abbildung 5.1 kann zwischen den drei verschiedenen Ebenen umgeschaltet

werden. Abbildung 5.3 zeigt die verschiedenen Ansichten eines Objekts aus den drei unterschiedlichen Ebenen.

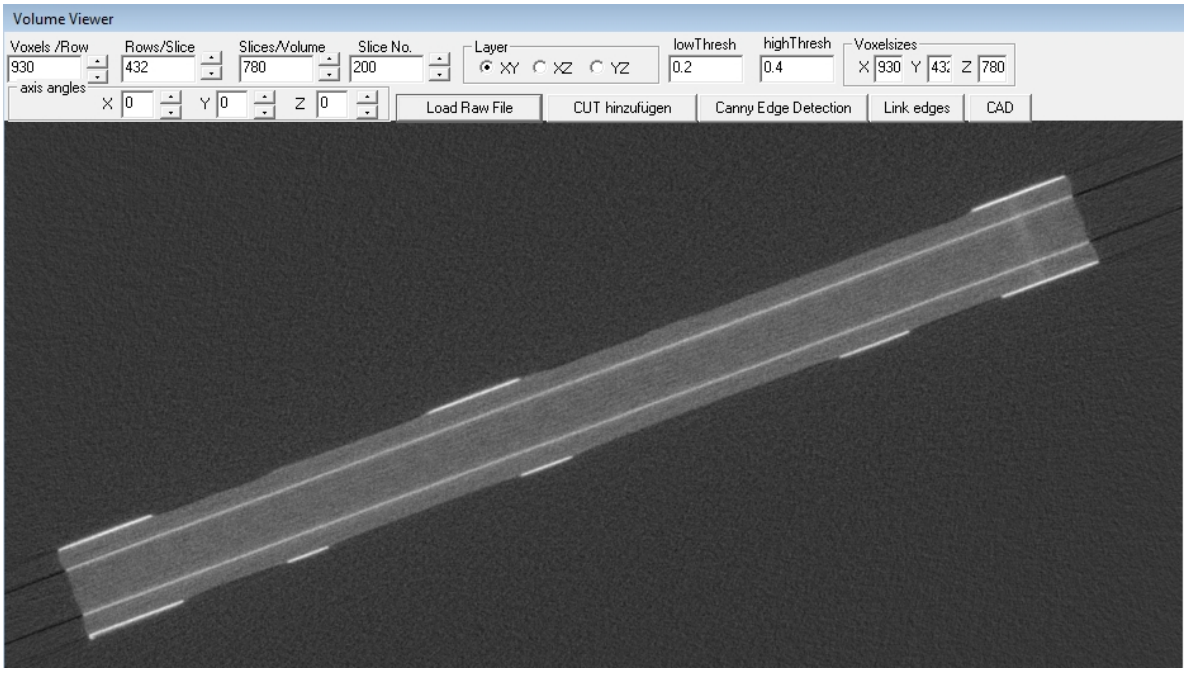


Abbildung 5.2: Oberfläche des entwickelten Systems.

Für das Navigieren durch die verschiedenen Ebenen des Voxelsatzes ist das Feld Nr. 4 zuständig. Durch die Angabe der Nummer der Schnittebene kann der Voxelsatz scheibchenweise in der dritten Dimension durchlaufen werden. Für die Kantenerkennung ist es manchmal notwendig das Objekt richtig auszurichten. Um das Objekt wie gewünscht auszurichten, wurde (wie in Kapitel 3.3 gezeigt) eine Drehung der Ebenen implementiert. Eine Drehung der Ebenen wird durch die drei Editfelder unter Nummer 8 durchgeführt. Die Werte der drei Felder entsprechen einer Drehung der aktuellen Ebene um den angegebenen Drehwinkel α um die x-, y- oder z-Achse. Die jeweiligen Winkel werden in Grad angegeben und intern in das Bogenmaß umgerechnet. Eine positive Drehung entspricht dabei einer Drehung gegen den Uhrzeigersinn gemäß dem Rechtssystem.

Hat man die gewünschte Schnittebene durch das Durchlaufen der Voxel und durch Drehungen der Ebenen ausgerichtet, so ist der nächste Schritt zur Erstellung eines CAD-Modells die Kantenerkennung. Dafür wird die aktuelle Schnittebene durch den Button Nummer 10 dem CUT hinzugefügt. Der CUT dient als Ausgangspunkt für die Kantenerkennung. Im aktuellen Programm besteht der CUT nur aus einem Bild. Für eine zukünftige Weiterentwicklung des Programms wäre es denkbar, den CUT als eine Liste zu implementieren, in der mehr als ein Schnittbild gespeichert wird.

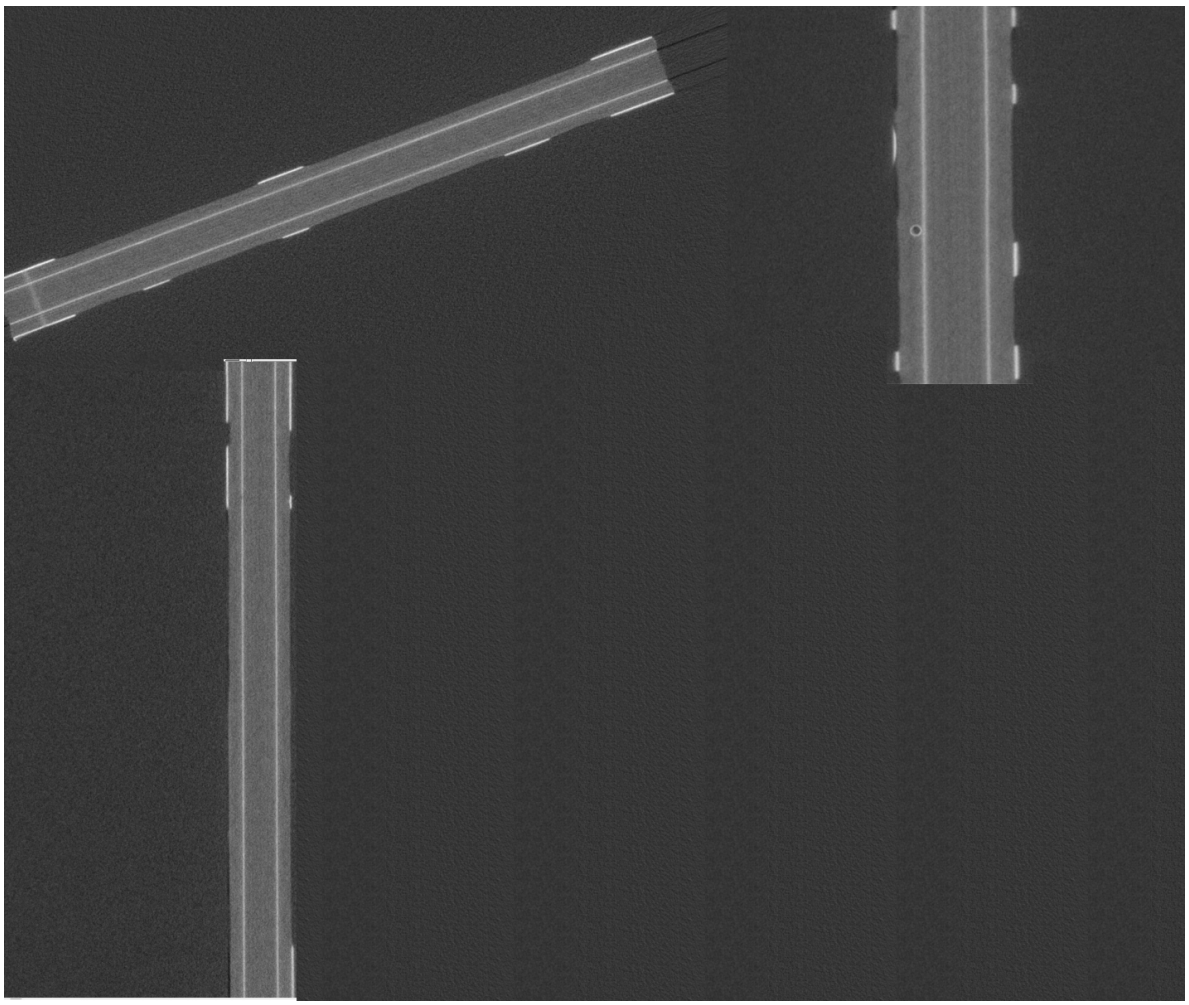


Abbildung 5.3: Drei verschiedene Ansichten eines Objekts aus der XY-Ebene oben links, der XZ-Ebene oben rechts und der YZ-Ebene unten links.

Aus dem definierten CUT wird durch das Drücken des Buttons Canny Edge Detection (Nummer 11 in Abbildung 5.1) die Kantenerkennung des CUTs durchgeführt. Implementiert wurde die in Kapitel 4.4 geschilderte Kantenerkennung von Canny. Die Editfelder unter der Nummer 6 geben die jeweils obere und untere Schranke für den Canny Edge Detection Algorithmus an. Anschließend an die Kantenerkennung wird das Thinningverfahren von Zhang und Suen, wie in Kapitel 4.5 beschrieben, durchgeführt. Abbildung 5.5 zeigt einen CUT vor und Abbildung 5.6 zeigt den CUT nach der Canny Edge Detection. Abbildung 5.7 zeigt ein Problem, das nach der Kantenerkennung auftreten kann: Die gefundenen Kanten ergeben keinen geschlossenen Linienzug. Um ein CAD-Modell zu erstellen müssen die Linien geschlossen sein. Es gibt nun zwei grundsätzlich Verfahren wie mit solchen nicht geschlossenen Linienzügen umgegangen wird. Die erste Möglichkeit besteht darin, solche Objekte zu ignorieren und nur geschlossene Objekte in das spätere CAD-Modell zu über-

```
...
edgeno= 221 edge size= 38
(399 109), (399 110), (399 111), (399 112), (399 113), (400 114), (400 115), (401 116),
(402 116), (403 117), (404 117), (405 117), (406 117), (407 117), (408 117), (409 116),
(410 115), (411 114), (412 113), (412 112), (413 111), (413 110), (413 109), (412 108),
(412 107), (411 106), (410 105), (409 105), (408 104), (407 104), (406 104), (405 104),
(404 104), (403 104), (402 105), (401 106), (400 107), (399 108),
edgeno= 222 edge size= 2
(401 105), (400 106),
...
```

Abbildung 5.4: Auszug aus der Edge Liste.

nehmen. Die zweite Möglichkeit besteht darin, den Anfangspunkt mit dem Endpunkt zu verbinden. Jedoch entsteht dadurch das Problem, dass das Objekt verfälscht wird. Gerade bei runden Objekten wird das Objekt durch das strikte Verbinden von Anfang und Ende die Eigenschaft eines runden Objekts verlieren, je weiter der Anfangs- und der Endpunkt auseinanderliegen.

Nach der Kantenerkennung und dem Thinningverfahren müssen die gefundenen Kanten den jeweiligen Objekten zugeordnet werden. Dies geschieht durch den in Kapitel 4.6 angegebenen Edge Link Algorithmus. Durch den Button mit der Nummer 12 wird der Edge Link Algorithmus ausgeführt. Das Ergebnis dieses Edge Link Verfahrens ist eine Liste von zusammengehörigen Kantenpunkten. Abbildung 5.4 zeigt einen Ausschnitt aus der Liste des Objekts PolyObjects welches die Poly-Objekte verwaltet. Man beachte, dass der Anfangs- und der Endpunkt des Kantenzugs Nummer 221 nur ein Feld auseinanderliegen. Sie können somit problemlos miteinander verbunden werden. Der Kantenzug Nummer 222 besteht nur aus zwei Kantenpunkten und kann für die weiteren Schritte ignoriert werden, da mindestens vier Punkte für die Objekterstellung nötig sind.

Der letzte Schritt der Exportierung als CAD-Modell in eine SAT-Datei konnte aus Zeitgründen leider nicht mehr realisiert werden. Gründe für den Zeitmangel waren zum Einen die lange Einarbeitungszeit in Open CASCADE, die länger als geplant dauerte und die anschließende neue Einarbeitung in den ACIS Modeler, wobei die bis dahin erstellte Software in eine neuere Version der Visual-Studio Entwicklungsumgebung eingebunden werden musste. Desweiteren war es für die Möglichkeit der Drehung der Ebenen notwendig, das bis dahin entwickelte Konzept der Datenanzeige zu überarbeiten.

Aber es wurde durch ein Beispiel gezeigt, dass es möglich ist, aus einer Polygonliste ein SAT-Objekt zu exportieren. Abbildung 5.8 zeigt ein ACIS-Testobjekt, das zu Testzwecken für den SAT-Export durch den ACIS Modeler erstellt wurde. Das Testobjekt besteht aus den fünf dreidimensionalen Punkten $P_1 = (0,0,0), P_2 = (10,0,0), P_3 = (10,10,0), P_4 = (0,2,0), P_5 = (0,0,0)$. Für die Vollendung des Programms ist es nur noch notwendig, die ACIS-Umgebung in die entwickelte Software einzubinden. Durch ein paar ACIS-Methodenaufrufe werden die in den Polygonlisten gesammelten Koordinaten als SAT-Datei exportiert.

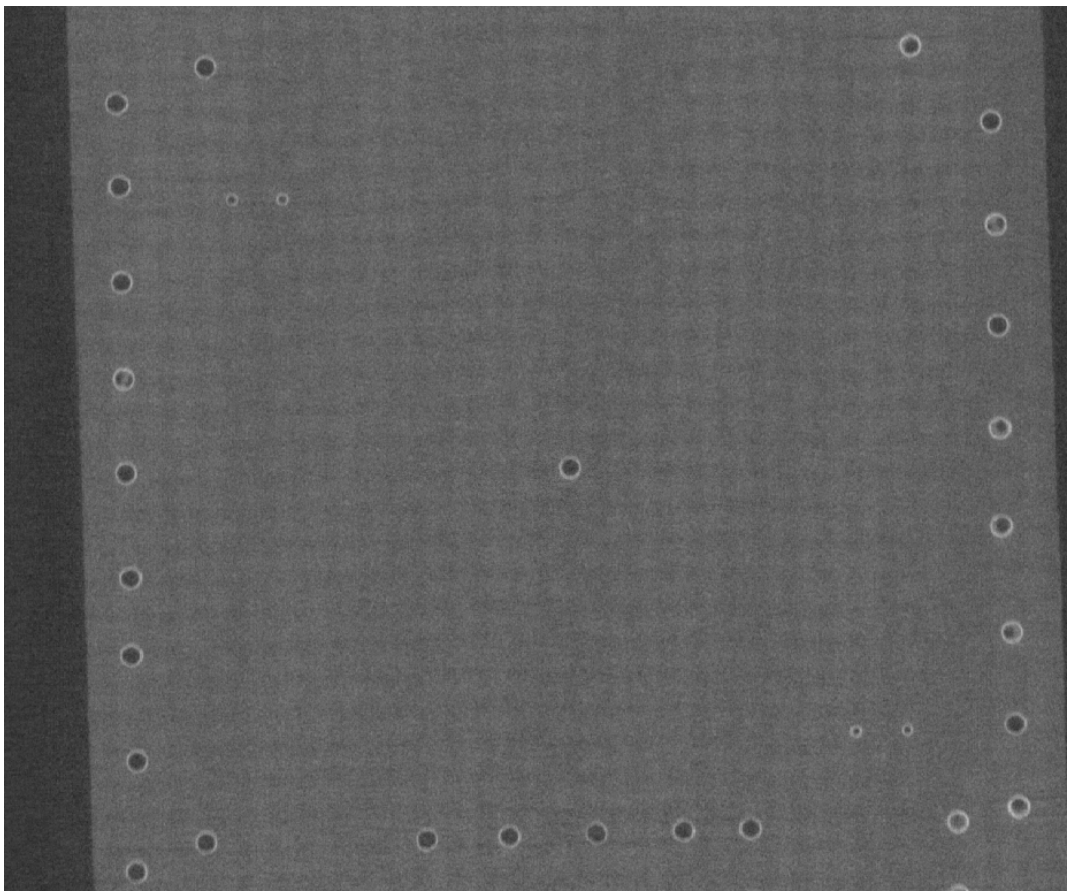


Abbildung 5.5: CUT-Beispiel vor der Canny Edge Detection.

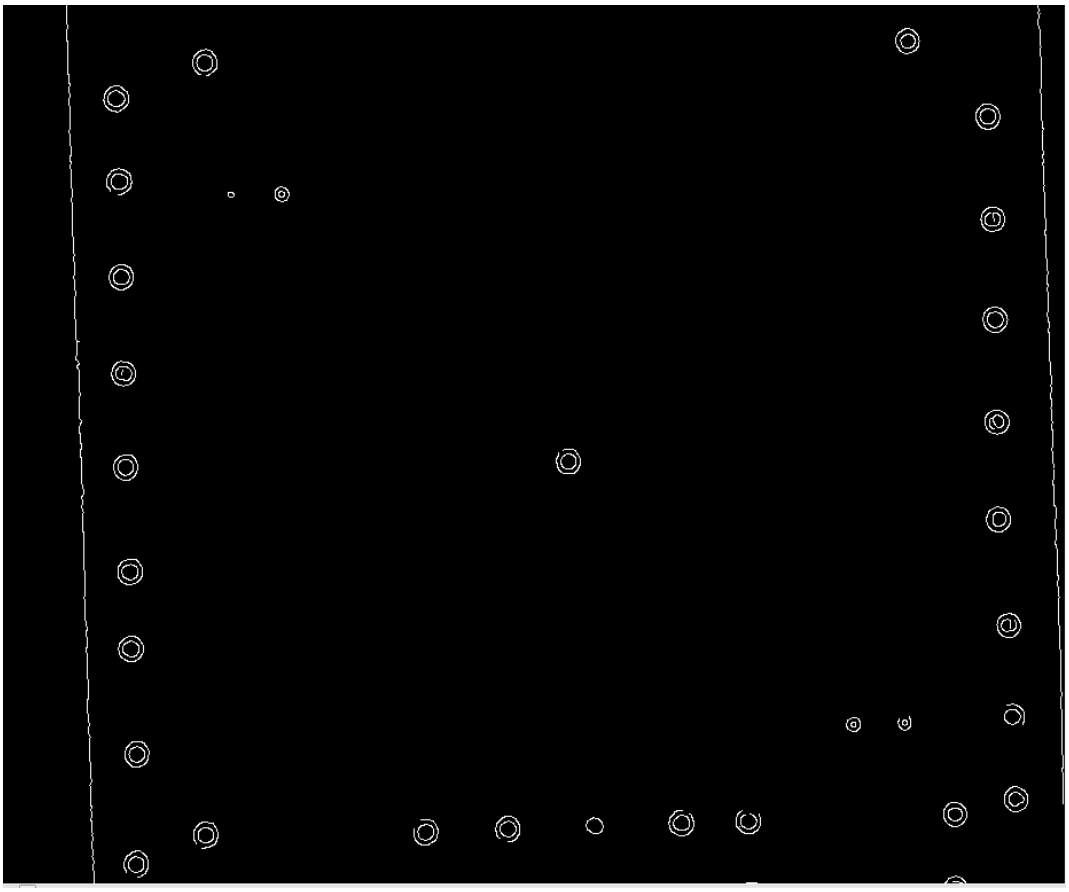


Abbildung 5.6: CUT-Beispiel nach der Canny Edge Detection.

Abbildung 5.7: Nicht geschlossener Polygonzug.

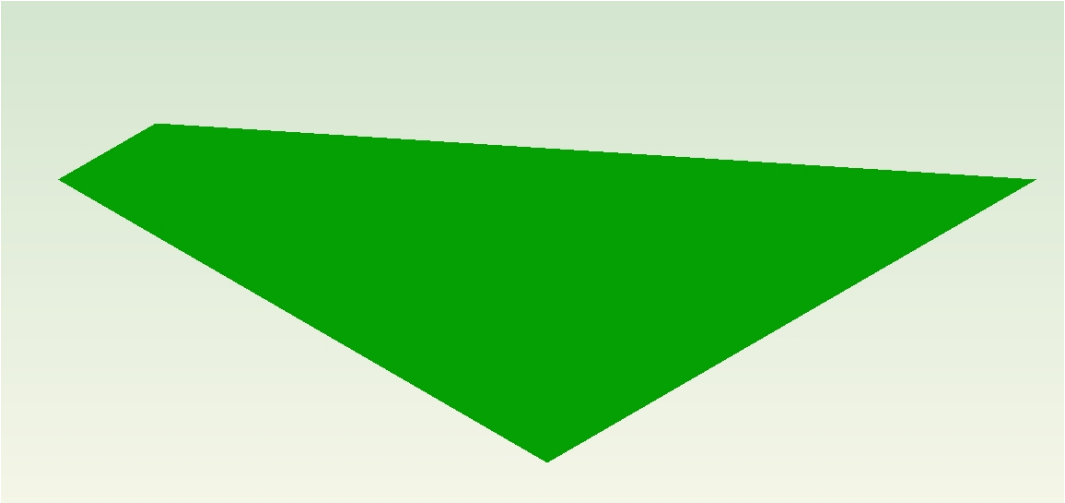


Abbildung 5.8: In ACIS Modeler erstelltes Testobjekt aus den Punkten $P_1 = (0,0,0); P_2 = (10,0,0); P_3 = (10,10,0); P_4 = (0,2,0);$ $P_5 = (0,0,0)$ angezeigt in einem SAT Viewer.

6 Zusammenfassung

Es wurde die Entwicklung eines Prototyps zur Erstellung eines CAD-Modells aus CT-Schnittbildern vorgestellt. Mit dem entwickelten System ist es möglich, auf einfache Weise durch eine grafische Oberfläche einen Voxelsatz zu laden und den Datensatz mit Hilfe von Schnittebenen anzuzeigen und auszurichten. Anhand der ausgerichteten Schnittebenen werden mittels Kantenerkennungsverfahren die Polyline-Koordinaten bestimmt, aus denen sich ein CAD-Modell erstellen lässt. Ausgehend von den Anforderungen an das System wurden zwei CAD-Entwicklungsumgebungen auf die Anforderungen hin untersucht und miteinander verglichen. SPATIALs ACIS Modeler, einer der beiden untersuchten CAD-Entwicklungsumgebungen, wurde ausgewählt. Diese Modellierungsumgebung kann aus zweidimensionalen Kantenlisten, die bei einer Kantenerkennung entstehen, ein dreidimensionales CAD-Objekt erstellen. Die gewählte CAD-Entwicklungsumgebung ist auch in der Lage, das CAD-Objekt in ein gängiges Datenaustauschformat zu exportieren und Attributeigenschaften an Objekte anzuhängen. Das Prinzip der Computertomographie und der daraus gewonnenen dreidimensionalen Voxeldaten wurden ebenso erklärt, wie auch die allgemeinen Verfahren für das Anzeigen von dreidimensionalen Voxeldaten. Es wurde gezeigt, dass es mit Hilfe von drei Schnittebenen möglich ist, Daten aus dem Voxelsatz anzuzeigen und dass diese Schnittebenen als Ausgangspunkt für eine zweidimensionale Kantenerkennung dienen. Mit Hilfe von Rotationsoperationen wurde eine Möglichkeit implementiert die Ebenen so auszurichten, dass mit ihnen die Kantenerkennung leichter durchführbar ist. Es wurden die allgemeinen Problemstellungen der Kantenerkennungsverfahren untersucht und Filterverfahren vorgestellt. Unter den untersuchten allgemeinen Kantenerkennungsverfahren wurde der Canny Edge Detection Algorithmus implementiert. Durch die Implementierung eines Thinning Algorithmus wurde das Kantenbild auf eine minimale Kantenbreite ausgedünnt, so dass sich die Kontur der Objekte mit Hilfe eines Kantenverbindungsverfahrens durch eine Liste von Polygonen darstellen lt. Es wurde gezeigt, dass sich aus dieser Liste ein CAD-Modell erzeugen lässt.

Ausblick

Für die zukünftige Weiterentwicklung des entwickelten Systems könnte man untersuchen, inwieweit sich eine vor der Kantenerkennung durchgeführte Filterung auf die durch die Kantenerkennung gefundenen Objekte auswirken kann. Interessant wäre auch ein Vergleich weiterer Kantenerkennungsverfahren mit dem in dieser Arbeit verwendeten Canny Algorithmus. Ein Problem das festgestellt wurde, sind nicht geschlossene Polygonzüge. Eine Möglichkeit dies zu lösen wäre beispielsweise dem Benutzer in Zukunft die Möglichkeit zu

geben, die Polygonzüge manuell zu schließen. Dies könnte durch eine interaktive Oberfläche realisiert werden. Ebenso könnte man nach anderen Lösungen zum Schließen der nicht geschlossenen Polygonzüge weiterforschen.

Literaturverzeichnis

[Ahn06] S. H. Ahn. Digital Signal Processing. http://www.songho.ca/dsp/, 2006. (Zitiert auf Seite 33)

[Ant98] H. Anton. *Lineare Algebra*. Spektrum Akademischer Verlag, Heidelberg, Berlin, 1998. (Zitiert auf Seite 28)

[Bas02] M. Basu. Gaussian-Based Edge-Detection MethodsA Survey. *IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICSPART C: APPLICATIONS AND REVIEWS*, 2002. (Zitiert auf Seite 36)

[BB05] W. Burger, M. J. Burge. *Digitale Bildverarbeitung : Eine Einführung mit Java und ImageJ*. Springer-Verlag, Berlin, Heidelberg, 2., berarbeitete auflage edition, 2005. (Zitiert auf Seite 35)

[Bra11] A. Braunstein. Implementierung von Algorithmen zur MESH-Erzeugung und numerischen Impedanzberechnung, 2011. (Zitiert auf Seite 16)

[Buz08] T. M. Buzug. *Computed Tomography: From Photon Statistics to Modern Cone-Beam CT*. Springer-Verlag, Berlin/Heidelberg, 2008. URL http://www.springer.com/medicine/radiology/book/978-3-540-39407-5. (Zitiert auf den Seiten 19 und 20)

[Can86] J. F. Canny. A computational approach to edge detection. *IEEE Trans.Pattern Anal. Machine Intell.*, PAMI-8:679698, 1986. (Zitiert auf den Seiten 37, 38, 39 und 40)

[DGSH03] J. Daniel G. Swanson, W. J. R. Hoefer. *Microwave Circuit Modeling,Using Electromagnetic Field Simulation*. ARTECH HOUSE, INC., 2003. (Zitiert auf Seite 15)

[Elv92] T. T. Elvins. A survey of algorithms for volume visualization. *SIGGRAPH Comput. Graph.*, 26:194–201, 1992. URL http://doi.acm.org/10.1145/142413.142427. (Zitiert auf Seite 21)

[Gre02] B. Green. Canny Edge Detection Tutorial. http://www.pages.drexel.edu/~weg22/can_tut.html, 2002. (Zitiert auf Seite 41)

[Har84] R. M. Haralick. Digital Step Edges from Zero Crossing of Second Directional Derivatives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(1):58–68, 1984. (Zitiert auf Seite 37)

[Hil85] E. C. Hildreth. Edge Detection. 207:187–217, 1985. (Zitiert auf Seite 36)

[HWS] J. Hillebrand, M. Wróblewski, S. Simon. Contactless Characterization of a 90° Hybrid Coupler by Means of Computed Tomography. Unveröffentlichter Bericht der Abteilung Parallele Systeme der Universität Stuttgart. (Zitiert auf den Seiten 9 und 19)

[HWS11] J. Hillebrand, M. Wróblewski, S. Simon. 3D Computed Tomography for High-Speed Interconnect Characterization. *Design Con 2011*, 2011. (Zitiert auf den Seiten 9, 19 und 20)

[Kov] P. D. Kovesi. MATLAB and Octave Functions for Computer Vision and Image Processing. Centre for Exploration Targeting, School of Earth and Environment, The University of Western Australia http://www.csse.uwa.edu.au/~pk/research/matlabfns/. (Zitiert auf Seite 45)

[LC87] W. E. Lorensen, H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. *SIGGRAPH Comput. Graph.*, 21:163–169, 1987. doi:http://doi.acm.org/10.1145/37402.37422. URL http://doi.acm.org/10.1145/37402.37422. (Zitiert auf den Seiten 22 und 24)

[LLS92] L. Lam, S.-W. Lee, C. Y. Suen. Thinning Methodologies-A Comprehensive Survey. *IEEE Trans. Pattern Anal. Mach. Intell.*, 14:869–885, 1992. doi:10.1109/34.161346. URL http://portal.acm.org/citation.cfm?id=138791.138793. (Zitiert auf Seite 43)

[MFW09] T. McPhail, P. Feng, J. Warren. Fast Cube Cutting for Interactive Volume Visualization. In *Advances in Visual Computing*, volume 5875 of *Lecture Notes in Computer Science*, pp. 620–631. Springer Berlin / Heidelberg, 2009. URL http://dx.doi.org/10.1007/978-3-642-10331-5_58. (Zitiert auf den Seiten 24 und 25)

[MH80] D. Marr, E. Hildreth. Theory of Edge Detection. *Proceedings of the Royal Society of London. Series B. Biological Sciences*, 207(1167):187–217, 1980. (Zitiert auf den Seiten 37 und 38)

[MT00] A. Martin, S. Tosunoglu. Image Processing Techniques for Machine Vision. In *Conference on Recent Advances in Robotics*, pp. 1–9. 2000. (Zitiert auf Seite 31)

[OPE11] OPENCASCADE. 3D modeling and numerical simulation. http://www.opencascade.org, 2011. [Online; accessed 18-Januar-2011]. (Zitiert auf den Seiten 13 und 15)

[Owe99] G. S. Owen. Volume Visualization and Rendering. http://www.siggraph.org/education/materials/HyperVis/vistech/volume/volume.htm, 1999. [Online; accessed 25-Januar-2011]. (Zitiert auf den Seiten 21 und 23)

[Spa11] Spatial. 3D Software Components for 3D Modeling, CAD Translation and 3D Visualization. http://www.spatial.com, 2011. [Online; accessed 25-Januar-2011]. (Zitiert auf Seite 15)

[Ste93] R. Steinbrecher. *Bildverarbeitung in der Praxis*. R. Oldenbourg Verlag, 1993. (Zitiert auf den Seiten 37 und 38)

[ZS84] T. Y. Zhang, C. Y. Suen. A fast parallel algorithm for thinning digital patterns. *Commun. ACM*, 27:236–239, 1984. doi:http://doi.acm.org/10.1145/357994.358023. URL http://doi.acm.org/10.1145/357994.358023. (Zitiert auf Seite 44)

[ZT98] D. Ziou, S. Tabbone. Edge Detection Techniques - An Overview. *International Journal of Pattern Recognition and Image Analysis*, 8:537–559, 1998. (Zitiert auf den Seiten 33 und 36)

Alle URLs wurden zuletzt am 12.04.2011 geprüft.

Erklärung

Hiermit versichere ich, diese Arbeit selbständig verfasst und nur die angegebenen Quellen benutzt zu haben.

(Alexander Schuck)