Texturbasierte dreidimensionale Rekonstruktion von Gebäudemodellen aus Rundumansichten

Diplomarbeit

zur Erlangung des akademischen Grades
Diplominformatiker

Humboldt-Universität zu Berlin
Mathematisch-Naturwissenschaftliche Fakultät II
Institut für Informatik

eingereicht von:	Aleksander Gudalo
geboren am:	11.02.1973
in:	Mönchengladbach
Gutachter(innen):	Prof. Dr. Peter Eisert
	Prof. Dr. Beate Meffert
eingereicht am:	.....
verteidigt am:	.....

Inhaltsverzeichnis

1 Einleitung **5**

1.1 Aufgabenstellung 6
1.2 Aufbau der Arbeit 7
1.3 Motivation 8

2 Theoretische Grundlagen **10**

2.1 Dreidimensionale Objektrepräsentation 10
2.2 Volumenmodell 11
2.3 Flächenmodell 12
2.3.1 Half-Edge-Datenstruktur 13
2.4 Perspektivische Projektion und Kameramodell 14
2.4.1 Geometrische Transformation 14
2.4.2 Kameramodell 16
2.5 Rekonstruktion der visuellen Hülle 18
2.5.1 Silhouettenschnitttechnik 19
2.5.2 Marching-Cubes-Algorithmus 20
2.6 Rekonstruktion aus den Bilddaten 21
2.6.1 Kongruenzanalyse 22
2.6.2 Stereoverfahren 24
2.6.3 Multiview-Verfahren 28

3 Datenerfassung **33**

3.1 Drehtellersteuerung und Versuchsaufbau 33
3.1.1 Fotoaufnahmen 34
3.2 Fehler auf der Bildebene 35
3.3 Fehler bei der Volumenrekonstruktion 36

4 Rekonstruktion der Oberfläche **39**

4.1 Polygon-Sichtbarkeitstest 40
4.1.1 Partitionierung 42
4.1.2 Erweiterung durch Strahlverfolgung 44
4.2 Verdeckungsanalyse 45

4.3 Lokale Optimierung mit Simulated Annealing 47
4.3.1 Kostenfunktion 48
4.3.2 Farbkorrespondenz 49
4.3.3 Glattheitseigenschaften 51
4.3.4 Polygonqualität 52
4.3.5 Distanzkriterien 52
4.3.6 Optimierungsfunktion 53
4.4 Globale Optimierung mit Graph Cuts 56
4.4.1 Kostenfunktion 56
4.4.2 Farbkorrespondenz 57
4.4.3 Glattheitsbedingungen 57
4.4.4 Graph Cut Verfahren 59
4.4.5 Konstruktion des Graphen 60
4.4.6 Maximaler Fluss in einem Graphen 63

5 Auswertung **66**
5.1 Ergebnisse des Sichtbarkeitstests 66
5.2 Verdeckungstest 70
5.3 Auswertung des Simulated Annealing Verfahrens 70
5.3.1 Parameter der Kostenfunktion 70
5.3.2 Analyse des Algorithmus 77
5.4 Auswertung Graph Cut 78
5.4.1 Parameter der Kostenfunktion 78
5.5 Laufzeit und Speicherverbrauch 85
5.6 Ergebnisse 88

6 Zusammenfassung und Ausblick **92**

A Programmbeschreibung **I**

B Abkürzungsverzeichnis **IV**

C Abbildungsverzeichnis **V**

D Tabellenverzeichnis **VI**

E Quelltextverzeichnis **VI**

F Literaturverzeichnis **VII**

G Danksagung **XX**

H Selbständigkeitserklärung **XXI**

1 Einleitung

> Es kann der beste Künstler nichts erdenken,
> Was nicht der Marmor schon in sich enthielt,
> Und der allein erreicht worauf er zielt,
> Dem Geist und Sinne seine Hände lenken.[1]
>
> Michelangelo Buonarroti

In vielen Bereichen der Informationsverarbeitung werden präzise dreidimensionale digitale Abbildungen von realen Objekten benötigt. Sowohl in der Industrie als auch in der Wirtschaft und in der Forschung braucht man diese Modelle, um Simulationen zu berechnen, Konstruktionen zu planen oder Entwürfe zu visualisieren. Aktuell etablieren sich 3D-Drucker, die auf hochwertige Modelle zur Ausgabe angewiesen sind. Viele weitere Anwendungen könnten aufgeführt werden.

Es gibt verschiedene Möglichkeiten diese digitalen Abbilder zu erzeugen. Die naheliegendste Herangehensweise, um Modelle zu erstellen ist es diese von Grund auf selbst nachzubilden. Dafür gibt eine große Anzahl an 3D-Grafik-Programmen und CAD-Anwendungen, die es erlauben nahezu jedes Objekt zu modellieren. Allerdings ist die Einarbeitung in die oft sehr komplexe Software aufwendig und somit nur durch Spezialisten bedienbar. Bei vielschichtigen, aufwendig konstruierten Modellen wird es zunehmend schwieriger diese selbst zu gestalten. Darüber hinaus bergen diese aufwendigen Konstruktionen das Risiko Abbildungsfehler zu forcieren. Die selektive Wahrnehmung des Gestalters kann ebenfalls Einfluss auf die Güte des erzeugten Modells haben. Deshalb ist es sinnvoll, einen realen Gegenstand automatisiert in ein virtuelles dreidimensionales Objekt zu überführen.

Um dies zu ermöglichen, gibt es eine Vielzahl an Verfahren, die sich aufgrund ihrer technischen Hilfsmittel unterscheiden. Neben den Laserscannern sind vor allem bildbasierte Verfahren weit verbreitet. Die Rekonstruktion der dreidimensionalen Oberfläche über zweidimensionale Bild-

[1] Engelhard, M.: Michelangelo. Sämtliche Gedichte. Insel Verlag, Frankfurt, 1992

aufnahmen wird seit über drei Jahrzehnten wissenschaftlich erforscht. Neben der einfachen und flexiblen Anwendung sind diese Verfahren besonders preisgünstig, da man mit einfachen handelsüblichen digitalen Kameras arbeiten kann. Eine Reihe von Verfahren, die nahezu exakte Modelle erstellen können, hat sich dafür etabliert.

Trotz verschiedener Lösungsansätze bei der Überführung in den virtuellen dreidimensionalen Raum muss bei allen Verfahren die Tiefenschätzung über eine Korrespondenzanalyse gelöst werden. Dafür benötigt man eine, meistens jedoch mehrere Fotografien des realen Objektes, die in aufwendige Berechnungen einfließen. Darüber wird der Tiefenwert für die Bildpunkte berechnet und ein komplettes Modell rekonstruiert.

In dieser Arbeit wird ein dreidimensionales Computermodell anhand von Fotografien eines auf einem Drehteller rotierenden Gebäudemodells konstruiert. Aus diesen Rundum-Aufnahmen wird zuerst ein Voxelmodell erstellt, welches in ein polygonales Modell überführt werden kann. Diese visuelle Hülle ist allerdings nur eine grobe Annäherung an das reale Objekt, da konkave Flächen und Aushöhlungen über das benutzte Silhouettenschnittverfahren nicht berücksichtigt werden.

Ausgehend von diesem einfachen rohen Polygonmodell werden verschiedene Ansätze erarbeitet, um eine präzise Annäherung an das originale Objekt zu erzielen. Durch die Auswertung aller relevanten Informationen aus den Bilddaten erhält man ein genaueres Verständnis des fotografierten Objektes und kann dieses dann subtraktiv verfeinern.

1.1 Aufgabenstellung

In dieser Diplomarbeit wird elaboriert, wie aus einem realen Objekt ein nahezu exaktes, polygonales, dreidimensionales Modell erzeugt werden kann. Dabei liegt der Schwerpunkt dieser Betrachtung bei der Verbesserung eines zuvor mit dem Silhouettenschnittverfahren erzeugten Polygonmodells.

Das virtuelle Modell wird dabei schrittweise rekonstruiert und verbessert. Dieser Ablauf kann in folgende Abschnitte untergliedert werden: Es werden Fotoaufnahmen von einem Gebäudeobjekt erzeugt, die dieses

allseitig rundum erfassen. Dafür wird ein Drehteller, auf dem das Objekt positioniert wird, vor einer Fotokamera rotiert. Die erzeugten Aufnahmen werden nun maskiert, um das Objekt vom Hintergrund abzugrenzen. Aus diesen hochaufgelösten Fotodaten wird dann mit dem Silhouettenschnittverfahren ein Voxelmodell erzeugt. Die Auflösung des Voxelmodells bestimmt dabei auch die Qualität des fertigen virtuellen Modells. Über eine Triangulation wird dieses Voxelobjekt folgend in ein polygonales Dreiecksnetz überführt. Um die Qualität des virtuellen Objektes zu verbessern, werden die hochaufgelösten Fotodaten erneut betrachtet. Über mehrere Aufnahmen aus verschiedenen Perspektiven werden die Tiefenschätzungen für eine Referenzposition errechnet und das Modell wird an die neuen Werte angepasst.

In dieser Arbeit werden verschiedene bekannte Verfahren betrachtet und mit den konkreten Bedürfnisse verglichen. Es wird schließlich ein lokales und ein globales Optimierungsverfahren in *C++* implementiert und diese werden miteinander verglichen.

1.2 Aufbau der Arbeit

Im Grundlagenteil der Arbeit wird die theoretische Basis herausgearbeitet, die nötig ist, um die Verfahren und Algorithmen zu verstehen, welche Bestandteil dieser Arbeit sind. Außerdem werden diverse Möglichkeiten der Repräsentation eines Modells in der Computergrafik beschrieben. Darauf aufbauend wird erläutert, wie diese Modelle mathematisch behandelt werden können und welchen Einfluss die Kamera- und Projektionsparameter darauf haben. Die fundamentalen Verfahren zur dreidimensionalen Rekonstruktion werden sukzessiv besprochen. Den Schwerpunkt bildet, ausgehend von den Verfahren der Korrespondenzanalyse, die Betrachtung der Rekonstruktionsoptimierung. Hier wird die Entwicklung der bekannten Verfahren kategorisiert und es wird exemplarisch auf die Vorzüge einiger für diese Arbeit relevanten Verfahren eingegangen.

Das Kapitel *Datenerfassung* beschreibt alle erforderlichen Schritte, die benötigt werden um die Daten für die Rekonstruktion zusammenzutragen. Ein wichtiger Teil dieses Abschnittes ist die Fehlerkategorisierung,

mit deren Hilfe die Güte des Modells direkt bestimmt wird.

Im Kapitel *Datenverarbeitung* werden dann alle Algorithmen beschrieben, die zur Rekonstruktion und zur Optimierung benötigt werden. Ausgehend vom Sichtbarkeitstest wird explizit auf die Kostenfunktionen der Optimierung und auf die Optimierungsstrategien selbst eingegangen. Hier werden sowohl lokale als auch globale Verfahren der Optimierung charakterisiert.

Im folgenden Kapitel *Resultate* werden die Laufzeit und der Speicherverbrauch der verschiedenen Algorithmen betrachtet, und es wird eine ausführliche Bewertung der Verfahren besprochen. Verschiedene Ansätze zur Verbesserung der Algorithmen werden einander gegenübergestellt und bewertet.

Am Ende dieser Arbeit wird in einer beurteilenden Zusammenfassung auf die Vor- und Nachteile der Versuche eingegangen.

Den Abschluss der Diplomarbeit bildet die Funktionsbeschreibung des neu entwickelten Optimierungswerzeuges *shaper*, welches für diese Arbeit in C++ programmiert wurde.

1.3 Motivation

Für das *Augmented Reality* (AR) Forschungsprojekt *VENTURI*[1], welches für mobile Endgeräte entwickelt wird, sollen digitale Modelle von Häusern erzeugt werden. Unter AR versteht man die kontextabhängige Einbettung zusätzlicher virtueller Informationen in die reale Welt. Eine maßstabsgetreue Miniaturmodellstadt ist die Vorgabe, die digitalisiert werden soll, um als Basis für die weitere Forschung zu dienen.

Es wird ein geeignetes Verfahren gesucht, um die Häuser der Stadt naturgetreu in virtuelle Computermodelle zu überführen. Untersucht wird dabei eine Vielzahl von bekannten Verfahren, um zwei grundlegend verschiedene Ansätze zu implementieren. Dabei müssen die Eigenheiten der architektonischen Objekte berücksichtigt und die Algorithmen an diese angepasst werden. Trotzdem sind die entwickelten Verfahren auch auf an-

[1] VENTURI ist ein Akronym und steht für: immersiVe ENhancemenT of User-woRld Interactions.

dere Objekte anwendbar, da sie über externe Parameter frei beeinflussbar sind.

Ausgehend von der einfachen visuellen Hülle der Häusermodelle werden diese Optimierungsverfahren angewandt, um die Modelle zu verbessern. Ein Kompromiss aus der Qualität der Ergebnisse und der Laufzeit der Optimierung muss bestimmt werden, um die Fassade des Hauses so genau wie möglich dem originalen Modell anzupassen. Das Ziel dieser Arbeit ist, ein automatisches Verfahren der Rekonstruktion zu entwickeln, in welches der Anwender nicht eingreifen muss.

2 Theoretische Grundlagen

2.1 Dreidimensionale Objektrepräsentation

Es gibt verschiedene mögliche Darstellungsformen der dreidimensionalen Modellrepräsentation. Diese werden in unterschiedlichen Datenstrukturen gespeichert und gezielt für spezielle Anwendungsgebiete verwendet. Sie werden genutzt, um vorgegebene räumliche geometrische Eigenschaften eines Gegenstandes abzubilden. Bei der Modellierung eines virtuellen Objektes gibt es laut Bender und Brill einige grundlegende Kriterien, die man beachten sollte [BB06]: Gibt es ein reales Pendant des Computermodells, das als Vorbild dient, kann man über dieses die Qualität der Abbildung beurteilen. Dabei sollte sich die Genauigkeit des Modells an der geplanten Anwendung orientieren. Die Rechenleistung der Computer schränken allerdings eine beliebig gute Annäherung an das originale Vorbild ein.

Es gibt vier wichtige Repräsentationsarten für Computermodelle, die je nach Anwendungsbereich eingesetzt werden können. Es handelt sich dabei um die Punktwolke, das Drahtmodell, das Flächenmodell und das Volumenmodell. Diese können hierarchisch einsortiert werden, da sie jeweils das nächste Modell um weitere Eigenschaften ergänzen. Auf der untersten Ebene steht die Punktwolke, aus der durch Hinzunehmen von Kanten, die jeweils zwei Punkte verbindet, ein Drahtgittermodell erzeugt werden kann [Del34, Cav74]. Aus dem Drahtgittermodell kann wiederum durch Hinzunahme von Flächen, die aus drei oder mehr Punkten bestehen können, ein Flächenmodell aufgespannt werden. Das Volumenobjekt kann ebenfalls aus der Punktwolke erzeugt werden und erweitert diese um Volumeneigenschaften. Aus diesem Volumenmodell kann wiederum ein Flächenmodell abgeleitet werden [LC87].

In dieser Arbeit wird sowohl das Volumenmodell als auch das Flächenmodell genutzt und deshalb in den folgenden Abschnitten ausführlich erläutert.

2.2 Volumenmodell

In dieser Arbeit wird für die Volumenrepräsentation das Voxelmodell verwendet. In der Computergrafik bezeichnet ein *Voxel* einen Volumenpunkt im Raum. Der Begriff des Voxels leitet sich aus den Begriffen *Volumetric* und *Pixel* ab [Fas12]. Ein Voxel wird über seine Koordinaten definiert und beschreibt eine dreidimensionale Rastergrafik. Dabei sind die Seitenlängen des Würfels in der Regel für alle Kanten und Elemente gleich lang. Die Genauigkeit des Modells kann über die Anzahl der Voxel und somit über die Kantenlänge bestimmt werden. Voxel werden häufig in effektiven hierarchischen Listenstrukturen, wie etwa dem *Oktalbaum*, gespeichert.

Diese Form der Datenrepräsentation wird vor allem in den bildgebenden Verfahren der Medizin zur Wiedergabe verwendet [LSL+05]. In der Computertomographie (CT) oder Magnetresonanztomographie (MRT) wird diese Darstellung beispielsweise eingesetzt, um Befunde zu visualisieren. Voxel werden jedoch häufig nur als Zwischendarstellungsstufen benutzt und zur weiteren Verarbeitung in Polygonmodelle umgewandelt.

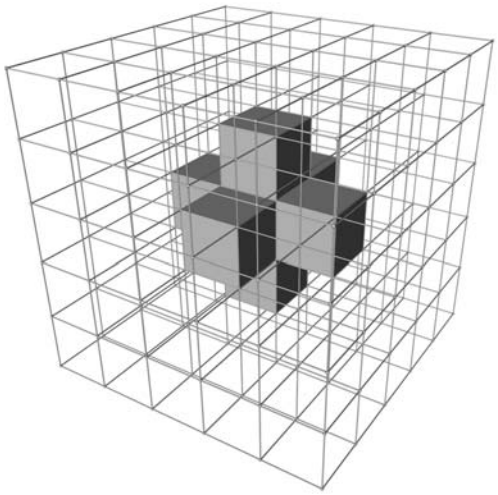


Abbildung 1: Beispiel eines einfachen Voxelmodells

2.3 Flächenmodell

Das Flächenmodell, oder auch Polygonmodell genannt, ist in der Computergrafik das am häufigsten eingesetzte Repräsentationsmodell. Die Oberfläche wird durch Polygonflächen mit eindeutigen Eigenschaften beschrieben. Zusammengesetzte Polygone spannen dabei ein Polygonnetz auf, welches die Oberfläche des hohlen Modells bestimmt.

Die einfachste Form des Polygons ist das Dreieck, welches immer planar und konvex ist und sich deshalb besonders gut zur Modellierung eignet. Es wird direkt von der modernen Grafikhardware (GPU) unterstützt und ist deshalb effizient in Anwendungen zu integrieren. Vierecke werden ebenfalls verwendet und sind vor allem in Kombination mit funktionaler Geometrie und bei Modellierungswerkzeugen beliebt. Andere komplexere Polygonformen sind ebenfalls möglich, jedoch selten in praktischen Anwendungen zu finden. Alle Polygonformen können aber letztendlich aus elementaren Dreiecken zusammengesetzt werden.

Die Eckpunkte werden mit Kanten verbunden und diese Vertices $v_0 \dots v_n$ spannen das Polygon auf, wobei $v_0 = v_n$ gilt. Dieses muss aus mindestens drei Eckpunkten bestehen und bildet somit im einfachsten Fall ein Dreieck. Die Polygone eines Modells werden mit einer festen Orientierung beschrieben und legen durch diese die Vorder- und Rückseite des Objektes fest.

Es existieren viele verschiedene Datenstrukturen, die sich zur Speicherung von Polygonnetzen eignen. In dieser Arbeit werden drei Arten von Datenstrukturen verwendet, die auch in vielen anderen Anwendungen benutzt werden.

Es gibt die polygonbasierte Datenstruktur, bei der die Koordinaten der Vertices aller Polygone in einer ungeordneten Liste gespeichert werden. Dabei kann es häufig zu vielen redundant abgelegten Vertices kommen.

Eine speichersparende Alternative ist die Indexierung der Vertices. Die Vertices werden dabei nur einmal gespeichert und die Polygone werden durch Zeiger indexiert. Dadurch werden Redundanzen bei der Speicherung der Koordinaten verhindert und die Koordinaten werden durch einfache Indizes ersetzt.

Alternativ kann auch eine kantenbasierte Datenstruktur eingesetzt werden, bei der neben einer Vertexliste auch eine indexierte Kantenliste gespeichert wird.

2.3.1 Half-Edge-Datenstruktur

Eine kantenbasierte Datenstruktur ist die *Half-Edge*-Struktur. In dieser werden nicht die einfachen Kanten des Polygons hinterlegt, sondern für jeder Verbindung zwischen zwei Vertices werden zwei gerichtete, gegenläufige *Halbkanten* gespeichert.

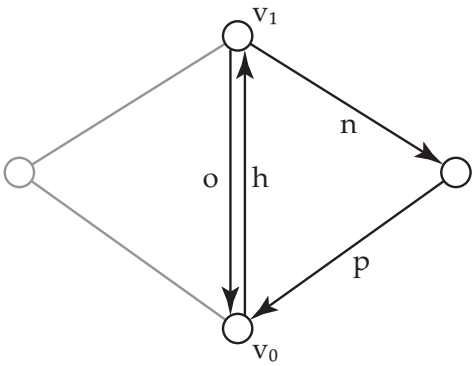


Abbildung 2: *Half-Edge*-Struktur

In Abbildung 2 verbinden die beiden Halbkanten h und o die beiden Vertices v_0 und v_1. Die Vorgängerhalbkante von h ist p und die nachfolgende Halbkante ist n. Diese spannen das Polygon auf und erleichtern die Navigation durch die umliegende Datenstruktur.

Neben einer erweiterten Vertex- und Polygonliste muss auch eine Halbkantenliste gespeichert werden. Dabei wird die Vertexliste, die die Koordinaten beinhaltet, um eine indexierte ausgehende Halbkante erweitert. In der Polygonliste wird zusätzlich ein Index auf eine angrenzende Halbkante des Polygons hinterlegt. In der *Half-Edge*-Liste ist das Vertice auf das diese Kante verweist, das angrenzende Polygon, die nächste Halbkante innerhalb dieses Polygons, die gegenläufige Halbkante und optional die Vorgänger-Halbkante des Polygons gespeichert [Dok13, BSBK02].

Die *Half-Edge*-Struktur erleichtert durch das Polygonnetz zu navigie-

ren, um etwa Nachbarn zu finden, da die Vorgängerkante, die Nachfolgerkante und die angrenzenden Flächen eindeutig bestimmt sind. Diese Operationen können in konstanter Zeit unabhängig von der Größe des Polygonnetzes implementiert werden. Eine solche kantenbasierte Datenstruktur hat allerdings im Vergleich zu den einfachen Strukturen einen sehr hohen Speicherverbrauch.

2.4 Perspektivische Projektion und Kameramodell

2.4.1 Geometrische Transformation

Um ein virtuelles Objekt im kartesischen Raum $\mathbb{R}^3$ zu positionieren, gibt es verschiedene lineare Transformationen $T : R^n \rightarrow R^m$ mit $m = n$ [AW98]. Man unterscheidet die Translation, die Rotation und die Skalierung [BB82]. Die Scherung gehört auch zu dieser Gruppe, wird jedoch in dieser Arbeit nicht verwendet. Diese linearen Transformationen werden auf jeden Vertex des Objektes angewandt. In der linearen Algebra für Computergrafik verwendet man oft homogene Koordinaten, die die drei Koordinaten im $\mathbb{R}^3$ um eine vierte Koordinate ergänzen. Die homogene Erweiterung erleichtert dabei die Berechnung der Matrixmultiplikationen.

Bei der Translation werden die Vertices eines Objektes um einen Richtungsvektor t verschoben. Für die Translation gilt also: $x = x + t$. Die Translationsmatrix ist wie folgt aufgebaut:

$$T = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Bei der Rotation wird für jede Achse eine eigene Rotationsmatrix benötigt. Diese Matrizen können jedoch zusammengefasst werden, um die Rotation in einem Schritt auszuführen. Für die Rotation gilt: $x = R \cdot x$.

$$R_x(\varphi) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\varphi & sin\varphi & 0 \\ 0 & -sin\varphi & cos\varphi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$R_y(\varphi) = \begin{pmatrix} cos\varphi & 0 & -sin\varphi & 0 \\ 0 & 1 & 0 & 0 \\ sin\varphi & 0 & cos\varphi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

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

Bei der Skalierung werden die Vertices mit der Skalierungsmatrix multipliziert. Für die Skalierung gilt: $x = S \cdot x$.

$$S = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Alle Transformationsmatrizen können miteinander kombiniert werden. Dies ist sehr effizient, da die Matrix nur einmal für mehrere Transformationsoperationen berechnet werden muss und alle Vertices nur mit dieser einen resultierenden Matrix multipliziert werden. Die Reihenfolge der Transformationen muss jedoch beibehalten werden, da die Matrixmultiplikation nicht kommutativ ist.

2.4.2 Kameramodell

Eine Kamera erzeugt ein planares Abbild einer dreidimensionalen Umgebung. Die geometrischen Abhängigkeiten wurden erstmalig von den Malern der Renaissance verstanden und angewandt [FLP01]. Die mathematisch perspektivische Darstellung wurde demnach um etwa 1413 von Filippo Brunelleschi entdeckt. Wie die Autoren Faugeras et al. weiter ausführen, wurden weitere Forschungen von Fotografen im 19. Jahrhundert vorgenommen, die schließlich dazu führten, dass sich Mathematiker mit der projektiven Geometrie befassten [FLP01].

Das einfache Modell einer Lochkamera, die eine zweidimensionale Projektion abbildet, kann über die Bildebene S, das Projektionszentrum C und über die Brennweite f, die den Abstand angibt, vollständig beschrieben werden (Abbildung 3). Die Projektion eines Punktes ist somit der Schnittpunkt dieser Bildebene mit dem Sehstrahl, der durch den Szenepunkt und das Projektionszentrum führt. Dieses einfache Projektionsmodell approximiert eine Kamera für viele Anwendungen ausreichend genau. Sie kann über weitere Parameter präzisiert werden, die man als *extrinsische Parameter*, die die Position und Orientierung der Kamera beschreiben, und als *intrinsische Parameter*, die die physikalischen Eigenschaften der Kamera beschreiben, bezeichnet.

Um einen Punkt mathematisch mehreren Kameras zur Verfügung zu stellen, muss man die Kamerakoordinaten in Weltkoordinaten überführen. Diese Transformation erfolgt mit einer Translationsmatrix T und einer Rotationsmatrix R [CS09].

Die extrinsischen Parameter der Kamera werden über das Zentrum der Kamera C und die Einheitsvektoren x_C, y_C und z_C beschrieben. Für einen Punkt P ergibt sich dann die Relation $P_c = R(P_w - T)$, wobei P_c den Punkt im Kamerakoordinatensystem abbildet und P_w im Weltkoordinatensystem. Die extrinsische Transformation kann auch in einer homogenen Matrix E dargestellt werden.

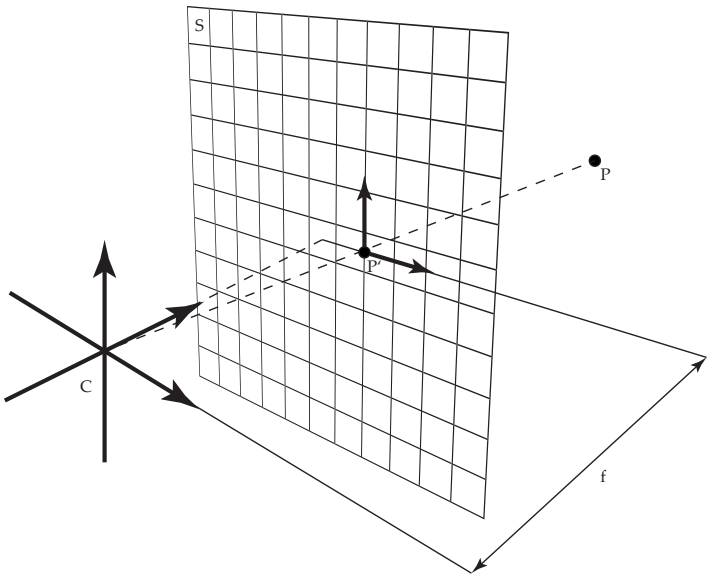


Abbildung 3: Projektion eines 3D-Punktes P auf die Bildebene S nach dem Lochkamera-Modell

$$R = \begin{pmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, T = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$E = TR = \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Die intrinsischen Parameter beschreiben die Ausrichtung der Kamera über die Brennweite und die Abbildungsparameter der beiden Koordinatensysteme [CS09]. Sie sind invariant bei Veränderung der Kameraposition. Verzerrungen durch die optischen Elemente der Kamera können ebenfalls in diese Berechnungen einfließen. Die Gleichung $P' = MP$ beschreibt das Abbild P' des realen Punktes P, der mit der Projektionsmatrix multipliziert wird. Die Projektionsmatrix M kann in $M = M_i M_e$ zerlegt werden [CS09].

$$M_i = \begin{pmatrix} f/h_x & 0 & o_x \\ 0 & f/h_y & o_y \\ 0 & 0 & 1 \end{pmatrix}_{3\times3}, M_e = \begin{pmatrix} R_1 & -R_1T \\ R_2 & -R_2T \\ R_3 & -R_3T \end{pmatrix}_{3\times4}$$

Die Matrix M_e besteht aus der Rotationsmatrix R und der Translationsmatrix T und beschreibt die Transformation des Punktes über die extrinsischen Parameter. M_i ist der intrinsische Parameter, der die Brennweite und die Position des Projektionszentrums o beschreibt. f/h_x und f/h_y geben die physikalischen Dimensionen eines Bildpunktes an.

2.5 Rekonstruktion der visuellen Hülle

Nach Bungartz et al. ergeben sich bei der grafischen Datenverarbeitung grundsätzlich drei wesentliche Arbeitsschritte [BGZ02]. Zuerst muss ein dreidimensionales Modell, das in der Regel von einem realen Vorbild ausgeht, erzeugt werden. Die Abbildung dieses Modells findet in einem zweidimensionalen Bildraum statt. Dieses Abbild kann auf einem Ausgabegerät wiedergeben werden. Durch den Einsatz von 3D-Druckern kann die Ausgabe wiederum in einem dreidimensionalen Abbild enden. Meistens findet die Ausgabe jedoch auf einem zweidimensionalen Medium statt.

Die virtuelle geometrische Modellierung des realen Gegenstandes wird in dieser Arbeit in zwei Schritten ausgeführt. Es wird aus einer Folge von Rundum-Fotografien eines realen Objektes ein Voxelmodell mit der Silhouettenschnitttechnik erzeugt. Aus diesem Voxelmodell wird dann mit dem *Marching-Cube*-Algorithmus ein Polygonmodell geschaffen, das eine einfache visuelle Hülle des realen Objektes wiedergibt. Die visuelle Hülle ist die maximale äußere Form, die für alle Bilder eines Objektes die gleiche Silhouette mit allen Punkten des Objektes abbildet [Lau94].

2.5.1 Silhouettenschnitttechnik

Bei der Silhouettenschnitttechnik, dem *Shape-From-Silhouette*-Verfahren (SFS) wird die Form eines realen Objektes schrittweise verfeinert. Dafür werden Fotografien aus verschiedenen Perspektiven mit bekannten extrinsischen und intrinsischen Parametern der Kamera aufgenommen (Abbildung 4). Die Qualität des Objektes ist dabei stark abhängig von der genauen Kalibrierung der Kamera [Eis04]. Die Anzahl der Ansichten sollte ausreichend groß sein und die Voxelauflösung sollte vernünftig ausgewählt werden, um die Qualität zu verbessern [Nie97]. Aus den Aufnahmen wird dann sukzessiv ein Voxelmodell konstruiert [Sze93]. Dabei sollte sich das Objekt stark vom Hintergrund abheben, um die sichtbaren Formen gut extrahieren zu können. Ein homogener Hintergrund und ein gut ausgeleuchtetes Objekt verbessern dabei die Segmentierung deutlich [MM05].

Aus jeder Kameraperspektive $c_n \in C$ wird eine Silhouette Δc_n extrahiert. Ein farblicher Schwellenwert bestimmt dabei, ob ein Bildpunkt p_m Teil des Objektes ist oder zum Hintergrund gehört. Ist der Bildpunkt innerhalb der Silhouette $p_m \bigcap_{c_n}$, wird der entsprechende Voxel gesetzt. Diese rekonstruierte visuelle Hülle h_p des Voxelmodells approximiert das reale Objekt.

Nicht alle Objekte sind exakt rekonstruierbar, da nicht alle Punkte Teil der Silhouettenhülle sind [Sze93]. Laurentini unterscheidet etwa zwischen *hard*- und *soft*-Punkten [Lau94, Lau95]. Die *hard*-Punkte werden, im Gegensatz zu den *soft*-Punkten, von mindestens einem Sehstrahl direkt getroffen und sind Teil der Silhouettenhülle. Objekte mit konkaven Einhöhlungen, bei denen nicht alle Punkte einer Silhouette zugeordnet werden können, sind nicht korrekt zu rekonstruieren. Außerdem können Objekte, bei denen Teile andere Teile des Objektes verdecken, nicht richtig dargestellt werden.

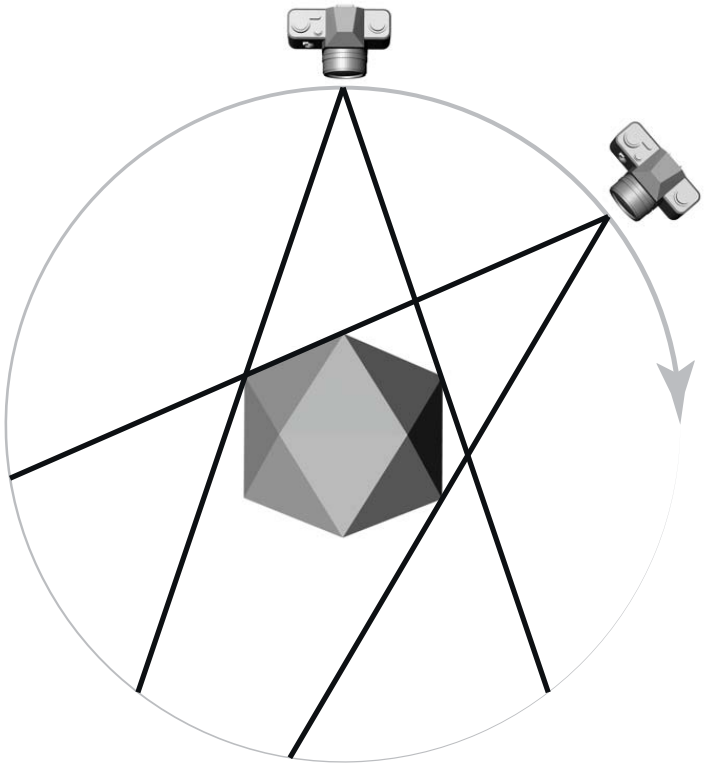


Abbildung 4: Fotografien aus verschiedenen Perspektiven

2.5.2 Marching-Cubes-Algorithmus

Um das Modell weiter verwenden zu können, muss es in ein eindeutiges Polygonnetz überführt werden. Dafür wird das *Marching-Cubes*-Verfahren eingesetzt, welches eine zusammenhängende geschlossene Oberfläche erzeugt [LC87]. Es werden dafür alle Voxel durchlaufen und je nach Nachbarschaftstatus markiert. Dabei werden an den Übergängen die Oberflächen in Form von Polygonen generiert. Aus den acht Eckpunkten des Würfels ergibt sich eine Auswahl von $2^8 = 256$ möglichen Kombinationen der Oberfläche für einen markierten Voxel [NY06]. Durch Rotationen und Spiegelungen können diese auf 15 verschiedene Äquivalenzklassen reduziert werden [Che95]. In einer vordefinierten *Lookup*-Tabelle (Abbildung 5) werden diese Polygonmuster hinterlegt. Beim Durchlauf wird über Schwellenwerte bestimmt, welche Teile innerhalb und welche außerhalb des Objektes liegen.

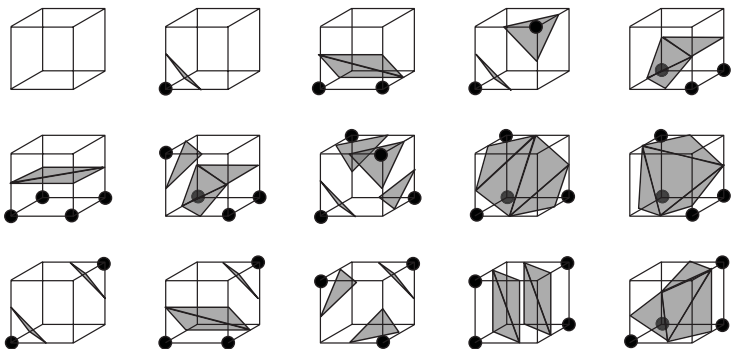


Abbildung 5: *Lookup*-Tabelle der 15 verschiedenen Kombinationen

Da es aufgrund von Mehrdeutigkeiten bei der Konstruktion zu Löchern im degenerierten Polygonnetz kommen kann, wird häufig ein erweitertes Verfahren eingesetzt [LC87, Che95]. Dazu wird die *Lookup*-Tabelle ergänzt, um diese Mehrdeutigkeiten zu verhindern und ein abgeschlossenes Polygonnetz zu erzeugen.

2.6 Rekonstruktion aus den Bilddaten

Seit über 30 Jahren wird die Rekonstruktion von dreidimensionalen Objekten aus zweidimensionalen Bilddaten erforscht [BBH03]. Dabei sind eine Vielzahl von Methoden und Variationen von Verfahren entstanden, die zum Teil sehr präzise Ergebnisse liefern [Pro13]. Frühe Anwendungen der ersten Verfahren waren vor allem in der Robotertechnik angesiedelt [SCMS01]. Später kamen andere wissenschaftliche Forschungsgebiete, wie etwa die Medizintechnik, hinzu [LSL+05]. Multimediale Anwendungen im Unterhaltungsbereich forcierten die Forschung. Aktuell sind es 3D-Druckverfahren und *Augmented Reality* Anwendungen, die präzise geometrische Informationen über die dreidimensionale Umwelt benötigen und die Forschung vorantreiben.

Man kann zwischen aktiven Verfahren, die beispielsweise mit einem Laserscanner oder kodiertem Licht arbeiten, und passiven Methoden, die mit Fotodaten umgehen, unterscheiden. Einige Verfahren kombinieren auch beide Ansätze, um dreidimensionale Modelle zu rekonstruieren [MS09]. In dieser Arbeit werden jedoch nur die komplexeren passiven

Verfahren behandelt. Diese Verfahren benötigen eine kalibrierte Kamera oder zusätzliche Informationen über die Geometrie der Szene. Oft benötigt man auch eine Segmentierung der Szene in Vorder- und Hintergrund. Andere Prämissen, wie etwa die Glattheitsbedingungen, werden aus den gegebenen Umweltbedingungen abgeleitet.

Die meisten Verfahren arbeiten mit dem *Lambertschen*-Lichtmodell, bei dem der Sichtwinkel keinen Einfluss auf die Lichtreflektion hat. Dadurch sollte ein Punkt des Objektes, in jedem Bild indem er sichtbar ist, die gleiche Farbe aufweisen. Es gibt aber auch Verfahren, die das Lichtmodell durch eine bidirektionale Reflektanzverteilungsfunktion (BRDF) erweitern oder Schattenwürfe in das Berechnungsmodell integrieren [FL95, Wei97, BMR01]. Bernardini beispielsweise erzeugt darüber eine einfache Normalenkarte, die bei der Darstellung bessere Ergebnisse liefert. Vogiatzis et al. arbeiten mit variierenden Lichtbedingungen, um Reflexionen herauszufiltern und feinere geometrische Details herauszuarbeiten [VHC06]. Ein neuer Ansatz von Abrams et al. wertet den realen Sonnenstand aus, um die Lichtrichtung festzulegen und reale Gebäude aus Serienaufnahmen zu rekonstruieren [AHP12].

Für die Ausgabe der Ergebnisse einer Rekonstruktion sind verschiedene Möglichkeiten vorhanden. Die *Stereo*-Verfahren speichern die Werte häufig in *Disparity Maps* oder *Level Sets*. Bei *Multview*-Verfahren werden die Ergebnisse in Voxelmodelle oder Polygonnetze überführt. Bei den Voxelmodellen können dabei durchaus auch rechteckige, nicht würfelförmige Voxel für die Speicherung genutzt werden [Nie94].

2.6.1 Kongruenzanalyse

Der wohl wichtigste Bestandteil eines Algorithmus zur Rekonstruktion eines dreidimensionalen Objektes ist die Übereinstimmungsanalyse. Dabei wird die Kongruenz zwischen zwei Bildeinheiten untersucht und der Grad der Übereinstimmung festgestellt. Durch die Kalibrierung der Kameras kann der Tiefenwert für einen Bereich, bei einer Übereinstimmung in verschiedenen Ansichten, festgelegt werden. Dafür können Farbintensitäten oder bestimmte Merkmale, wie Linien oder Kanten aus den ver-

schiedenen Bildern, miteinander verglichen werden [SS01, CS09]. Die eigenschaftsbasierten Verfahren werden in dieser Arbeit jedoch nicht weiter betrachtet.

Bei den Intensitätsvergleichsverfahren gibt es eine Vielzahl von Ansätzen, die benutzt werden, um einzelne Bildpunkte oder Regionen miteinander zu vergleichen. Die wichtigsten und für diese Arbeit relevanten Verfahren sind die *Summe der absoluten Differenzen* (SAD), die *Summe der quadratischen Differenzen* (SSD) und die *Summe des Kreuzproduktes* (SCP) [CS09].

$$D_{SAD} = \sum_{(x,y)\in I} |I_1(x,y) - I_2(x+d_x, y+d_y)|$$

$$D_{SSD} = \sum_{(x,y)\in I} (I_1(x,y) - I_2(x+d_x, y+d_y))^2$$

$$D_{SCP} = \sum_{(x,y)\in I} I_1(x,y) I_2(x+d_x, y+d_y)$$

Da der Rechenaufwand gering und die Implementierung einfach ist, sind viele weitere Variationen mit Beschränkungen nach oben und unten oder in normalisierter Form möglich [BBH03]. Diese Verfahren sind allerdings sensibel für Ausreißer [BN97, CS09]. Cyganek und Siebert schlagen deshalb beispielsweise das *Kovarianz-Varianz*-Verfahren (CV) vor, um Ausreißer zu berücksichtigen.

Ein anderer Vorschlag, der jedoch nicht besonders verbreitet ist, ist der *bitweise* Vergleich, der etwa über die *Hamming-Distanz* berechnet werden kann [SS02]. Für mehrdimensionale Vektoren bietet sich die *Minkowski-Metrik* (MM) an.

$$D_{MM}(p,q) = \left(\sum_{k=1}^{N} |p_k - q_k|^{\alpha} \right)^{1/\alpha}$$

Statistische Ähnlichkeitsmaße können ebenfalls für die Korrespondenzanalyse in Bildfeldern verwendet werden. So sind die *Gaußsche-Distanz* (GD) und die *Cauchy-Distanz* (CD) robuste und beliebte Metriken [BN97, HZ04, CS09].

$$D_{GD}(p,q) = e^{-\left(\frac{D_{SSD}(p,q)}{\sigma}\right)^2}$$

$$D_{CD}(p,q) = \tau^2 log\left[1 + \left(\frac{D_{SSD}(p,q)}{\tau}\right)^2\right]$$

Andere Ähnlichkeitsmaße verwenden entropische Bildeigenschaften, um den Bereich eines Bildpunktes zu vergleichen oder um die Bereichsgröße anzupassen. Auch Bildhistogramme können zur Auswertung der Fotokonsistenz genutzt werden oder als Vorstufe für andere Verfahren dienen. Nichtparametrische Verfahren messen dagegen Nachbarschaftsrelationen, wie die Permutation oder die Reihenfolge der Differenzen [CS09].

2.6.2 Stereoverfahren

Bei den Stereoverfahren, die mit zwei Kameraansichten arbeiten, wird in der Literatur grundsätzlich zwischen *dense*- und *spare*-Verfahren unterschieden. Diese Ausdrücke beziehen sich auf die Verarbeitung der Bilddaten. Bei den sogenannten dichten Verfahren werden alle Bildpunkte ausgewertet und für jeden Pixel wird ein Tiefenwert ermittelt. In der anderen Gruppe der Verfahren wird nach bestimmten Merkmalen gesucht und es wird nur für diese Merkmale eine Tiefeninformationen erzeugt.

Bei den Stereoverfahren wird die Ausgabe der Ergebnisse für die Optimierung häufig in einer Tiefenkarte oder *Disparity Map* abgelegt. Dafür kann eine einfache Tiefenkarte mit einem Referenzbild oder eine Tiefenkarte, in der beide Bilder vereint sind, benutzt werden. Aus diesem Tiefenbild können schließlich triangulierte 3D-Modelle, Voxelmodelle oder Level-Sets erzeugt werden.

Die Stereoverfahren lassen sich in globale und lokale Optimierungsverfahren aufteilen [CS09]. Globale Verfahren berechnen die Kosten der Optimierung für alle Bildpunkte quasi gleichzeitig. Lokale Verfahren optimieren dagegen punktuell und sind daher häufig schneller als globale Verfahren. Allerdings gibt es einige Überschneidungen bei diesen Verfahren, da viele globale Methoden lokale Ansätze für die Optimierung

nutzen. Globale Verfahren haben den Vorteil, dass sie in Bildregionen mit wenig Farbvarianz bessere Ergebnisse liefern.

Der Aufbau der Verfahren kann in drei grundlegende Schritte aufgeteilt werden. Im ersten Schritt wird für jeden Pixel eine Korrespondenzanalyse pro Tiefenverschiebung ausgeführt, wie sie bereits im Abschnitt 2.6.1 beschrieben wurde. Da durch das Rauschen und durch ungenaue Daten Fehler von einzelnen Bildpunkten starke Schwankungen erzeugen können, werden diese häufig zu Gruppen zusammengefasst und berechnet. Die Tiefeninformation kann entweder über einen Bereich der Tiefenkarte auf einer Ebene errechnet werden oder der Fehler wird zusätzlich über mehrere Ebenen der Tiefenkarte eines Bildpunktes akkumuliert [SS98, MMHM02]. Dafür sind verschiedene Methoden geeignet, wie einfache *Box*-Filter, *Binomial*-Filter oder *Gauß*-Filter [CS09]. Einige Verfahren arbeiten dabei auch mit einer variablen Fenstergröße, die sich an der Varianz des Bildes ausrichtet [SS98, Vek03, Cyg05]. Es kann auch eine variable Suchregion für die Kongruenzanalyse, das sogenannten *Shiftable-Window*-Verfahren, implementiert werden [SS02].

Aus den errechneten Tiefeninformationen wird im zweiten Schritt entweder global oder lokal die Disparitätenkarte errechnet. Bei den lokalen Verfahren wird dafür einfach der beste Wert für jeden Pixel ausgewählt. Problematisch ist beim lokalen Ansatz, dass ein Bildpunkt in mehreren Positionen gleich oder ähnlich gute Korrespondenzwerte aufweisen kann und für eine korrekte Entscheidung keine weiteren Mittel zur Verfügung stehen. Um dies zu umgehen, kann man allerdings mehrere Werte zwischenspeichern und den endgültigen Wert an den Nachbarbildpunkten ausrichten. Globale Verfahren sind im Allgemeinen effektiver und erzeugen bessere Ergebnisse, da alle lokalen Berechnungen gleichzeitig in die Optimierung einfließen. Es handelt sich bei dieser Optimierung allerdings um ein *NP-schweres* Problem. Für die Optimierung wird üblicherweise eine Energiefunktion eingesetzt, die lokale Pixeldisparitäten mit Glattheitswerten, die sich meistens nur auf die direkten Nachbarn beziehen, kombiniert. Das Optimierungsproblem kann mit einer Vielzahl von verschiedenen Algorithmen gelöst werden. Viele Algorithmen enden allerdings in lokalen Minima und finden kein exaktes globales Minimum.

Im dritten Schritt können letztendlich Verbesserungen an den Ergebnissen vorgenommen werden. So kann die Qualität eines zuvor erzeugten Polygonnetzes ausgeglichen oder Löcher in diesem repariert werden. Cyganek und Siebert unterscheiden bei den lokalen Verfahren einfach zwischen *Feature-Matching*-Verfahren und den *Area-Matching*-Verfahren [CS09]. Bei Brown et al. werden auch lokale Gradienten-Verfahren als eigenständige Gruppe genannt [BBH03]. Beim *Feature-Matching* werden bestimmte Merkmale der Bilddaten, wie etwa Kanten oder Ecken, ausgewertet, die dann zur Berechnung der Tiefeninformationen dienen. Alle unbekannten Zwischenräume werden aus den errechneten Tiefenwerten der Merkmale interpoliert. Bei dem *Area-Matching*-Verfahren oder *Block-Matching*-Verfahren werden nicht mehr einzelne Bildpunkte betrachtet, sondern Bildregionen miteinander verglichen [MMHM02, BBH03]. Dabei fließt die Position der Bildpunkte in dieser Region mit in die Berechnung ein. Es wird eine Region um einen Punkt mit einer entsprechenden Serie an Punkten im zweiten Bild verglichen. Die Epipolargeometrie hilft dabei den Suchraum einzuschränken, da ein bekannter Bildpunkt des ersten Bildes im zweiten Bild auf einer Linie zwischen Epipol und dem Fluchtpunkt liegen muss [Cor05, CS09]. Zur Vereinfachung der Vergleiche können die Eingabebilder außerdem *rektifiziert* werden. Dabei werden die Bildpunkte an der horizontalen Achse ausgerichtet und der Suchbereich wird auf eine horizontale Linie reduziert. Es sind noch viele weitere Optimierungen für die Korrespodenzanalyse von Stereobildern bekannt [CS09, MMHM02, SS02]. Beispielsweise können schon verglichene Bildbereiche beim *Area-Matching*-Verfahren zwischengespeichert und im nächsten Schritt wiederverwendet werden.

Bei den Gradientenverfahren wird über Differentialgleichungen nach Farb- und Helligkeitsunterschieden in den Bildpaaren gesucht [BBH03].

Für die Optimierung des globalen Optimierungsproblems wurden verschiedene Techniken entwickelt. So wird das *Downhill-Simplex*-Verfahren von Kück et al. für die Optimierung eines Polygonnetzes eingesetzt [KHV04, FL95]. Dabei werden schrittweise alle Vertices aus einer *Priority-Queue* durchlaufen und verschoben. Der Algorithmus wurde von Nelder und Mead entwickelt und basiert auf einem Simplex, bei dem für

jeden Punkt des Volumens ein Funktionswert für eine Reihe unabhängiger Variablen berechnet wird [NM65, PTVF07]. In jeder Iterationsstufe wird dann der beste Wert durch Expansion, Reduktion oder Reflektion über dem Simplex ermittelt und beibehalten, um das Minimum der Funktion zu finden. Der robuste Algorithmus konvergiert zwar langsam, kann aber über festgelegte Abbruchbedingungen und Schwellenwerte beendet werden.

Beim *Simulated-Annealing*-Verfahren wird ein Abkühlungsprozess nachgebildet um ein globales Optimum zu finden [CS09]. Lokale Minima können dabei unter einem vorgegebenen Schwellenwert verlassen werden um bessere Ergebnisse zu ermitteln. Bei unendlicher Abkühlungszeit wird ein globales Minimum sicher gefunden. Dies muss bei einer endlichen Zeitvorgabe nicht zutreffen, da es sich hierbei um ein heuristisches Verfahren handelt.

Das auf *Markov Random Field* (MRF) aufsetzende *Belief-Propagation*-Verfahren arbeitet mit statistischen Modellen, die die Knoten in Vertrauensklassen einsortieren. Über verschiedene Ketten können die Glattheitsbedingungen, die Diskontinuität und die Verdeckungsprämissen geregelt werden [BVZ98, CS09].

Bei der *Dynamischen Programmierung* wird über epipolare *Scanlines* optimiert. Dabei wird das globale Problem durch Aufteilung in Teilprobleme, die separat gelöst und dann zusammengesetzt werden, verbessert. Die Berechnung der lokalen Kosten kann nach den vorgestellten Methoden in Abschnitt 2.6.1 entweder für jeden Bildpunkt einzeln oder für bestimmte Merkmale berechnet werden. Es werden dabei Ordnungs- und Glattheitsbedingungen in jeder *Scanline* berücksichtigt. Die Minimierung der globalen Funktion ist dann ein minimaler Kostenpfad durch die Disparitätenkarte [BBH03]. Durch die Ordnungs- und Eindeutigkeitsbedingungen der Bildpunkte, die nicht immer gegeben sind, kann es zu fehlerhaften Streifenbildungen entlang der *Scanlines* kommen [CS09]. Eine Erweiterung stellen die *Intrinsischen-Kurven* dar, die Bildpunkte jeder einzelnen *Scanline* vektorbasiert abbilden und miteinander vergleichen [BBH03].

Bei dem *Graph-Cut*-Verfahren wird die Optimierung einer Energie-

funktion über den maximalen Fluss in einem Graphen berechnet [FF56, BVZ01, KZ02a, Kol04, CLL08]. Dabei wird im Gegensatz zur *Dynamischen Programmierung* in alle Richtungen optimiert. Hierbei können in einem einzigen Schritt mehrere Bildpunkte gleichzeitig angepasst werden [Kol04]. Dafür wurden zwei Strategien von Boykov et al. eingeführt [BVZ01]. Der *α-β-Swap* und die *α-Expansion*. Bei dem *α-β-Swap* können die Tiefenwerte aus zwei Regionen untereinander ausgetauscht werden. Bei der *α-Expansion* wandert eine Gruppe in die andere, wenn es dadurch zu einer Verbesserung der Werte einer Energiefunktion kommt. Der Abbruch kann nach beliebig vielen Durchläufen festgelegt werden [Kol04].

2.6.3 Multiview-Verfahren

Ausgehend von den Stereoverfahren wurden als Erweiterung *Multiview*-Verfahren entwickelt, um mit mehr als zwei Kameraansichten Objekte zu rekonstruieren. Viele Techniken der Stereoverfahren können bei den Multiview-Verfahren weiter verwendet werden [GCS06]. Die Qualität der Multiview-Verfahren ist stark abhängig von der Qualität der Kameradaten und der Anzahl an Kameraperspektiven. Will man ein abgeschlossenes Objekt erzeugen, benötigt man eine ausreichende Anzahl an Perspektiven, die das Objekt rundum erfassen. Die ersten Verfahren versuchten einfach die visuelle Hülle zu approximieren [SCMS01]. Laut Slabaugh et al. sind die ersten Versuche, mit einer visuellen Hülle zu arbeiten, bei Martin und Aggarwal zu finden [SCMS01, MA83]. Dabei wurde eine aufwendige Segmentierung eingesetzt, um den Hintergrund vom tatsächlichen Objekt zu trennen. Zur Speicherung wurden die Volumendaten dann in einer Oktalbaum-Struktur abgelegt.

Man kann die Rekonstruktionsalgorithmen in die gleichen Gruppen einteilen wie bei den Stereoverfahren. Auch hier unterscheidet man grundsätzlich zwischen dichten Verfahren, bei denen alle Bildpunkte in die Berechnung einbezogen werden, und *Feature-Based*-Verfahren, bei denen Merkmale aus den Bilddaten ausgewertet und zur Rekonstruktion herangezogen werden.

In der Literatur wird häufig zwischen direkten und indirekten Verfah-

ren unterschieden. Bei den direkten Verfahren wird eine Kostenfunktion für ein dreidimensionales Objekt berechnet, um unmittelbar daraus die Oberfläche zu extrahieren [Dye01]. Dazu gehört beispielsweise das *Voxel-Coloring*-Verfahren oder das *Shape-From-Silhouette*-Verfahren, welches im Abschnitt 2.5.1 besprochen wurde [Nie94, SD97, PD98, Eck00]. Bei dem *Voxel-Coloring*-Verfahren wird jedem Voxel eine Farbe zugeordnet, die mit den Bilddaten am besten übereinstimmt und dadurch eine maximale Übereinstimmung der Farbkonsistenz abbildet. Dabei werden Überdeckungen über Schwellenwerte aussortiert und speziell behandelt. Verbesserungen mit variablen Voxel-Auflösungen lassen die *Voxel-Coloring*-Verfahren in Echtzeit arbeiten [PD98]. Andere Verfahren, die mit verschiedenen Sichtbarkeitshypothesen für jeden Voxel arbeiten, verbessern die Ergebnisse ebenfalls [KS98, ESG99, SGEB00, YSB+02]. Hierbei wird zuerst für jeden Voxel eine Hypothese über die Sichtbarkeit aufgestellt. Es wird danach jedoch mehrmals über alle Ansichten iteriert, um über Rückprojektionen alle inkonsistenten Voxel zu entfernen, bis eine korrekte Repräsentation vorliegt und eine ausreichend gute Darstellung erzeugt wurde. Eine freie, uneingeschränkte Positionierung der Kameraansichten ist bei diesen Verfahren möglich [KS00]. Für die Speicherung der Voxel werden neben skalaren Werten auch gerichtete Distanzwerte benutzt [KBSS01]. Kobbelt et al. erweitern das *Marching-Cubes*-Verfahren um Merkmalsextraktion, die dann in die Rekonstruktion einfließt [KBSS01]. Variationen des *Space-Carving*-Verfahrens erlauben neben dem Entfernen von Voxeln auch die Hinzunahme weiterer Voxel, um eine Kostenfunktion zu minimieren [SCD+06]. Zheng analysiert und kategorisiert in seiner Arbeit die Konturen des Objektes, um ein besseres Verständnis der konkaven Flächen zu bekommen [Zhe94].

Andere Verfahren, die eine direkte Rekonstruktion ermöglichen, sind die volumetrischen *Markov-Random-Field*-Algorithmen oder auch die globalen *Graph-Cut*-Verfahren, die im letzten Abschnitt besprochen wurden.

Iterative Verfahren, bei denen eine Kostenfunktion sukzessiv verbessert wird, können beispielsweise mit fertigen Polygonnetzen arbeiten und diese schrittweise verbessern [FL94, FL95, Pul97, EWN01, EHS02, IS03, EWN04]. Bei Eckert et al. wird etwa ein zweistufiges Verfahren ange-

wandt [EWN01, EWN04]. Im ersten Schritt wird ein grobes Polygonmodell mit dem *Shape-From-Silhouette*-Verfahren erzeugt, welches dann im Folgenden iterativ lokal verbessert wird. Dabei wird in jedem Schritt eine Vertexposition über einen lokalen Gradienten optimiert, bis eine vorgegebene Energiefunktion minimal ist. Die Richtung der Verschiebung kann auch über die Normale der visuellen Hülle oder über die Kameraposition festgelegt werden [EHS02]. Die Funktion setzt sich aus einem Korrelationsterm, einem Qualitätsterm für die umgebenden Polygone und mehreren Glattheitstermen zusammen. Esteban et al. benutzen für den Korrelationsterm mehrere Bildpunkte, um die Robustheit zu verbessern [EHS02]. Ebenso arbeiten Matsumoto et al. mit Blöcken von Bildpunkten, um das Rauschen zu verringern [MFK99]. Bei ihrem *Shape-From-Silhouette*-Abstimmungsverfahren werden vorgegebene Fehlertoleranzen akzeptiert um die Robustheit zu erhöhen. Bei Eckert et al. werden die Dreieckspolygone schrittweise über die Vertices entlang der Normalen in sieben verschiedene Positionen gesetzt und dann in alle Kameraperspektiven zurückprojiziert, um die Korrelation zu überprüfen [EWN01]. Kück et al. benutzen für die orthogonale Rückprojektion einer Polygonfläche die Grafikhardware (GPU) und berechnen dabei auch die Schatten und Verdeckungen [KHV04]. Das Verfahren arbeitet bei dieser Optimierung mit einem hierarchischen, schrittweise detailreicher werdenden Modell. Bei der Verschiebung wird streng geprüft, ob die Vertices innerhalb der Silhouette bleiben und ob es Überschneidungen innerhalb des Dreiecksnetzes gibt. Um den Berechnungsaufwand weiter zu senken, kann auch auf der Bildebene mit verschiedenen Detailstufen gearbeitet werden [ES04]. Auclair et al. benutzen ebenfalls Dreiecksflächen, die durch eine Funktion, die mit inneren und äußeren Kräften arbeitet, Merkmale eines Modells formt [AVC08, SCD$^+$06]. Die Kräfte werden von der äußeren Silhouette, die vorberechnet wird und über die Distanz einwirkt, über die Fotokonsistenz und einen Glattheitsterm bestimmt. Durch den Einsatz eines Polygonnetzes lassen sich dabei glattere Übergänge erzeugen, da besser zwischen den Tiefenwerten interpoliert werden kann. Fua modelliert eine globale Fläche aus lokalen gerichteten Teilflächen und optimiert diese dann über eine Energiefunktion [Fua97]. Diese Schei-

ben, die auch zur Darstellung dienen, werden aus den Bilddaten *geclustert*. Auch Habbecke und Kobbelt arbeiten mit Scheiben, die schrittweise nach der Initialisierung erweitert werden [HK07]. Neue Scheiben werden hinzugenommen, um unbekannte Bereiche des Objektes zu erschließen. Diese werden auf der Tangentialfläche der bekannten Vorgänger bis zur Konvergenz verschoben. Fua und Leclerc nutzen neben dem Korrelationsterm auch die Albedo, um die Optimierungsfunktion zu verbessern [FL95]. Sie können dabei auch fehlende Kameraparameter aus den Bilddaten errechnen [FL94]. Die Autoren Furukawa und Ponce arbeiten mit kleinen rechteckigen Teilflächen [FP10]. Dabei wird in einem dreistufigen Verfahren zuerst nach Merkmalen gesucht, die in allen Ansichten genau übereinstimmen. Danach wird die Region um die Merkmale erweitert, wie bei Lhuillier und Quan, um eine dichtere Oberfläche zu bekommen [LQ02]. Im dritten Schritt werden dann alle Ausreißer herausgefiltert. Iterative Verfahren, die in Echtzeit Objekte rekonstruieren, sind ebenfalls bekannt [SLMHP02]. Hierbei wird das polyedrische Initialvolumen aus mehreren Stereokamerapaaren errechnet. Mit Stereokamerapaaren arbeiten auch Okutomi und Kanade oder Collins beim *Space-Sweep*-Verfahren [OK93, Col96].

Level-Sets und Voxelobjekte können auch sukzessiv bearbeitet und verbessert werden [SCD$^+$06, Dye01]. *Level-Set*-Verfahren arbeiten beispielsweise mit Differentialgleichungen, die das Initialvolumen verbessern. Dabei werden keine Voxel, sondern Punkte in einem diskreten dreidimensionalen Gitter arrangiert [Sla00].

Andere Verfahren arbeiten mit mehreren Tiefenkarten, wie sie im Stereo-Bereich benutzt werden oder speichern die Tiefenwerte im Alphakanal der Textur [KHV04, GCS06]. Dafür können sowohl *dense*-Verfahren als auch *spare*-Verfahren benutzt werden. Diese werden für jede Position separat berechnet und zusammengeführt, um ein Objekt daraus zu erzeugen. Goesele et al. benutzen eine robuste Korrespondenzanalyse über einen Bereichsschätzer und setzen nur eine geringe Anzahl von Ansichten ein, um aus den erzeugten Tiefenkarten für jede Ansicht ein Objekt zu formen [GCS06]. Dabei werden nur die Bereiche rekonstruiert, die mit Gewissheit zugeordnet werden können. Fehlende Informationen führen

dadurch allerdings zwangsläufig zu Löchern im Objekt.

Schneider und Eisert erlauben dem Anwender interaktiv einzugreifen, um die Qualität der Rekonstruktion zu verbessern [SE12]. Dabei kann neben der Form auch die Glattheit des Objektes beeinflusst werden.

Einige Verfahren wurden speziell für die Rekonstruktion von Straßenansichten und Häusern entwickelt [XFZ+09]. Xiao et al. verwenden ein mehrstufiges Verfahren, in dem zuerst eine Segmentierung des Bildinhalts in rechteckige Blöcke durchgeführt wird. Danach werden einzelne Häuserblöcke separiert. Diese einzelnen Häuserfassaden werden daraufhin aus dreidimensionalen Flächen zusammengesetzt. Ein ähnliches Verfahren wird bei Zebedin et al. für Luftbildaufnahmen genutzt [ZKGGK06]. Dabei wird aus der errechneten Tiefenkarte und den Segmentationsdaten ein aus geometrischen Primitiven bestehendes Basismodell erzeugt. Eine Kombination aus Luftbildaufnahmen und Straßenansichten ist ebenfalls möglich [Frü02]. Furukawa et al. haben ein mehrstufiges Verfahren entwickelt [FCSS09]. Dabei wird zuerst in Bereichen mit starker Farbvarianz der Textur das Objekt rekonstruiert, um daraus eine Richtung für die dominanten Flächen zu extrahieren. Diesen Flächenhypothesen werden dann die Bildpunkte zugeordnet. Aus diesen Hypothesen wird danach mit dem *Markov-Random-Field*-Verfahren eine Tiefenkarte errechnet. Dick et al. arbeiten mit Primitiven, die in den Bildern erkannt werden, und fügen diese automatisiert in ein Modell ein [DTC04]. Werner und Zisserman benutzen antrainierte Algorithmen, die aus den erlernten Informationen elementare Gebäudeteile in das Modell einfügen [WZ02]. Andere Verfahren benötigen zusätzliche Interaktion mit dem Benutzer und erleichtern einfach die Modellierung von komplexen Gebäuden [DTM96, XFT+08, SSS+08]. Auch bei der Gebäuderekonstruktion gibt es Verfahren, die in Echtzeit befriedigende Ergebnisse liefern können [PNF+08, CLCG08].

3 Datenerfassung

3.1 Drehtellersteuerung und Versuchsaufbau

Da es sich bei den zu rekonstruierenden Objekten um Häusermodelle handelt, die auf einem Drehteller positioniert werden können, wurde eine Kamera auf einem Stativ befestigt, um die rotierenden Objekte allseitig um 360° aufzunehmen. Wie bereits in Abschnitt 2.6 beschrieben, kommen in vielen Arbeiten der *Computer Vision* Drehteller mit kalibrierten Kamerasystemen zum Einsatz. Dabei sind die intrinsischen und extrinsischen Parameter der Kamera, der Rotationswinkel und die Achsenabstände bekannt.

Abbildung 6: Drehteller der Firma Kaidan (Quelle:www.kaidan.com)

Für diese Arbeit wurde ein einachsiger Drehteller der Firma *Kaidan* benutzt, der um die vertikale Achse rotiert und eine etwa 30 *cm* große Auflagefläche für das Modell bietet (Abbildung 6). Der Antrieb erfolgt über einen Schrittmotor, der um einen vorgegebenen Winkel gedreht werden kann. Für die Steuerung wurde eine in *C++* entwickelte Bibliothek verwendet, die mit der Steuereinheit des Drehtellers über die serielle Schnittstelle kommuniziert. Dabei kann eine absolute oder eine relative Position angesteuert werden, weiterhin kann man einen vorgegebenen Winkel anfahren. Die Kommandos können automatisiert an den Drehteller gesendet und mit der Kamera synchronisiert werden.

3.1.1 Fotoaufnahmen

Die Fotos des Hausmodells wurden mit einer handelsüblichen Kamera, die auf einem Kamerastativ ausgerichtet wurde, vor schwarz abgedecktem Hintergrund aufgenommen. Auf einem gleichmäßig ausgeleuchteten Drehteller wurde das Motiv dann schrittweise rundum die vertikale Achse gedreht. Durch die diffuse Beleuchtung konnten Schattenwürfe und Glanzlichter reduziert werden. Die Fotografien (Abbildungen 7 bis 9) wurden dabei unter gleichbleibenden Bedingungen in einer hohen Auflösung aufgenommen. Je nach Objekt wurden 50 bis 72 Bilder für die Rekonstruktion fotografiert und für die weitere Bearbeitung im verlustfrei komprimierten PNG-Format gespeichert. Die Aufnahmen haben eine Auflösung von 18 Megapixel bei 5184×3456 Bildpunkten.

Abbildung 7: 6 von 50 Aufnahmen des Fachwerkhauses

Abbildung 8: 4 von 72 Aufnahmen des *Burger King* Modells

Abbildung 9: 8 von 72 Aufnahmen des Bankhauses

Je mehr Bilder von einem Objekt aufgenommen werden, desto besser wird das rekonstruierte Modell. Allerdings erhöht sich dadurch auch die Aufnahmedauer und die Rekonstruktionszeit für das Objekt. Schon mit wenigen Bildern und bei geringer Auflösung erhält man jedoch verwertbare Resultate.

Um das Hausmodell vom Hintergrund und dem Drehteller abzugrenzen muss das Bild segmentiert werden (Abbildung 10). Es wird für jedes Bild eine Schnittmaske erzeugt, um den Hintergrund zu entfernen.

Abbildung 10: (links) Originale Aufnahme des Hausmodells, (mitte) Schnittmaske für die Aufnahme, (rechts) Ausgeschnittenes Hausmodell

3.2 Fehler auf der Bildebene

Die Qualität des rekonstruierten Objektes ist abhängig von der Güte des Rekonstruktionsalgorithmus und der Anzahl der Rekonstruktionsfehler. Die Rekonstruktionsfehler können schon bei der Bilderfassung entstehen. Diese können durch die begrenzte Auflösung der Fotografie oder durch fehlerhafte Kameraparameter verursacht werden [Nie97, Nie98].

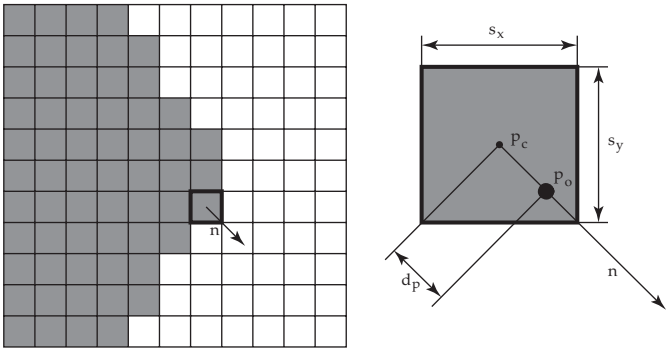


Abbildung 11: (links) Bildpunkte der Oberflächenkontur, (rechts) Ausgewählter Bildpunkt der Oberflächenkontur mit tatsächlicher Silhouettenposition p_o

Durch die begrenzte Auflösung kann die tatsächliche Oberfläche eines Objektes nicht eindeutig innerhalb eines Bildpunktes liegen. Der reale

Oberflächenpunkt p_o befindet sich also auf der Achse der Oberflächennormalen n zwischen dem Mittelpunkt des Pixels p_c in Richtung der Normalen (Abbildung 11) [Nie97]. Der Pixel hat eine geometrischen Auflösung von $s_x \times s_y$. Daraus kann dann der Positionsfehler Δd_p errechnet werden:

$$\Delta d_p = \begin{cases} \frac{1}{2}\sqrt{s_x^2 + (\frac{n_y}{n_x} s_y)^2} & , |n_x| \geq |n_y| \\ \frac{1}{2}\sqrt{s_y^2 + (\frac{n_x}{n_y} s_x)^2} & , |n_y| \geq |n_x| \end{cases}$$

Für jede Komponente der Kameraparameter kann eine Analyse der Abweichung ausgewertet werden. Der resultierende Bildfehler ergibt sich aus allen Kamerapositionen und beschreibt den Abstand des rekonstruierten Silhouettenpunktes von der realen Objektkontur. Dieser muss größer sein als der Formfehler Δd_p, der aus der begrenzten Auflösung resultiert [Nie98].

Andere Fehler, die durch den Kamerasensor oder das Kameraobjektiv entstehen, können ebenfalls das Ergebnis der Rekonstruktion verschlechtern.

3.3 Fehler bei der Volumenrekonstruktion

Auf Fehler, die bei der Rekonstruktion des Volumenmodells entstehen, hat der Benutzer einen größeren Einfluss. Man kann hier zwischen Fehlern der begrenzten Formauflösung und Fehlern, die durch die Anzahl der Kameraperspektiven entstehen, unterscheiden [Nie98]. Beide Werte können durch den Benutzer frei verändert werden.

Der Fehler, der durch die begrenzte Formauflösung entsteht, ist abhängig von der Größe der Voxelkanten w. Die kleinste Einheit des Volumens ist der Voxel, der einen Teil des Konturverlaufs abbildet. Durch die begrenzte Dimension kann die tatsächliche Oberfläche innerhalb eines Voxels nicht eindeutig bestimmt werden. Der reale Oberflächenpunkt v_o befindet sich also irgendwo auf der Achse der Oberflächennormalen n zwischen dem Mittelpunkt des Voxels v_c in Richtung der Normalen (Abbildung 12) [Nie97]. Daraus kann dann der Positionsfehler Δd_v analog zu dem Bildfehler errechnet werden:

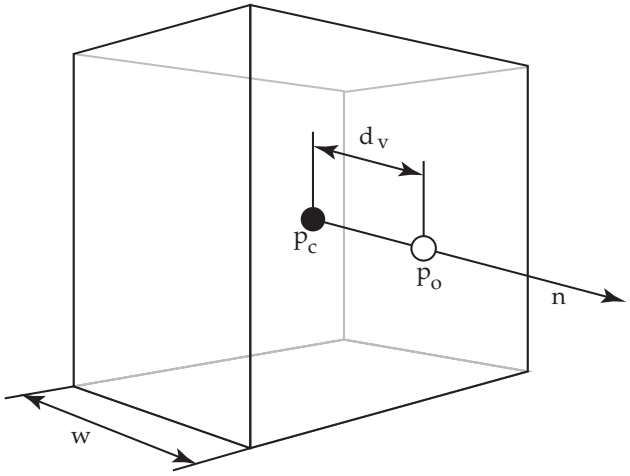


Abbildung 12: Voxel der Oberflächenkontur mit realer Silhouettenposition p_o

$$\Delta d_v = \begin{cases} \frac{w}{2}\sqrt{(\frac{n_y}{n_x})^2 + (\frac{n_z}{n_x})^2 + 1} & , |n_x| \geq |n_y|, |n_x| \geq |n_z| \\ \frac{w}{2}\sqrt{(\frac{n_x}{n_y})^2 + (\frac{n_z}{n_y})^2 + 1} & , |n_y| \geq |n_x|, |n_y| \geq |n_z| \\ \frac{w}{2}\sqrt{(\frac{n_x}{n_z})^2 + (\frac{n_y}{n_z})^2 + 1} & , |n_z| \geq |n_x|, |n_z| \geq |n_y| \end{cases}$$

Die beschränkte Anzahl der Kamerabilder hat offensichtlich Einfluss auf die Qualität des rekonstruierten Objektes. Die visuelle Hülle der Rekonstruktion ist dabei immer größer oder gleich der tatsächlichen Objektform. Betrachtet man ein Objekt mit glatter konvexer Oberfläche und einem konstanten Krümmungsradius r, so ist der Formfehler Δd_v abhängig von dem Winkel $\Delta\phi$ zwischen zwei aufeinander folgenden Ansichten und dem Krümmungsradius selbst (Abbildung 13) [Nie97].

Andere Fehler der visuellen Hülle, wie etwa Löcher in einem Objekt, können dadurch entstehen, dass diese Partien des Objektes in keiner Ansicht wiedergegeben werden oder durch andere Teile des Objektes verdeckt werden. Durch eine vernünftige Auswahl der Perspektiven und eine angemessene Anzahl der Ansichten kann die Qualität jedoch verbessert werden.

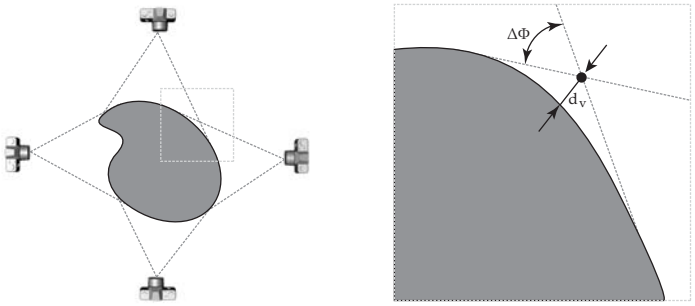


Abbildung 13: Fehler durch begrenzte Anzahl der Kamerabilder

Die systembedingten konkaven Einhöhlungsfehler, die in keiner Silhouette abgebildet werden können, und die Verdeckungsfehler werden bei der Rekonstruktion in Kapitel 4 genauer analysiert.

4 Rekonstruktion der Oberfläche

Die Rekonstruktion und Anpassung der Oberflächenstruktur, ausgehend von einem einfachen dreidimensionalen Modell, welches durch das Silhouettenschnittverfahren erzeugt wurde, ist der zentrale Bestandteil dieser Arbeit. Das erzeugte Polygonnetz wurde nur aus den Silhouettendaten des Objektes erzeugt, ohne weitere vorhandene Bildinformationen zu verwenden. Im Optimierungsschritt werden nun die Bilddaten genutzt, um die Qualität des Modells zu verbessern und die Tiefeninformationen, die durch die Dimensionsreduktion der Fotografie verloren gegangen sind, zu berechnen. In Abschnitt 2.6 wurden verschiedene Verfahren vorgestellt, die sich dazu eignen. Die besonderen Eigenschaften der Gebäudearchitektur schränkten die Auswahl zwar ein, jedoch wurden die verwendeten Verfahren so ausgewählt, dass durch Anpassung der Parameter auch andere Objektformen eingescannt werden können. Das Ziel ist, die qualitativ bestmögliche Rekonstruktion zu erreichen.

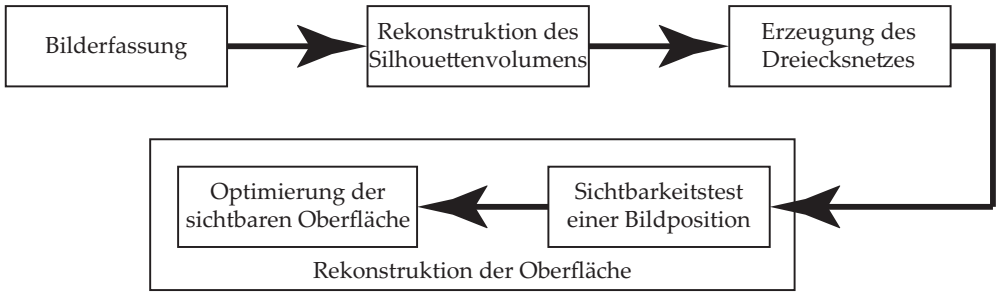


Abbildung 14: Ablaufplan der automatischen Objektrekonstruktion

Wegen des enormen Aufwands bei der Implementierung der verschiedenen *Multiview*-Verfahren, musste eine vernünftige Auswahl getroffen werden. Nach tief reichender Recherche wurde ein lokales Optimierungsverfahren, welches das *Simulated Annealing* Verfahren implementiert, und ein globaler Ansatz, der den *Graph Cut* einsetzt, ausgewählt. Beide Verfahren wurden in *C++* implementiert und an die vorgegebenen Bedürfnisse angepasst.

Hauptbestandteil des Optimierungsalgorithmus ist es, ein gegebenes Polygonnetz so anzupassen, dass für möglichst alle Bildpunkte des Objektes in allen vorgegebenen Ansichten, in dem der jeweilige Bildpunkt sichtbar ist, eine Farbübereinstimmung vorliegt. Dabei darf die Kontur des Objektes nicht vom originalem Objekt abweichen und Überdeckungen von Teilbereichen müssen berücksichtigt werden. Diese Bedingungen werden als Minimierungsproblem einer Kostenfunktion betrachtet und in endlicher Zeit gelöst.

4.1 Polygon-Sichtbarkeitstest

Bevor man den Optimierungsalgorithmus ausführt, müssen die Vertices des *Meshes* bestimmt werden, die verschoben werden sollen. Dafür müssen alle in der Referenzansicht nicht sichtbaren Polygone ausgeschlossen werden. Neben dem *Backface-Culling*, bei dem alle Polygone entfernt werden, deren Normale nicht zur Kamera gerichtet ist und die somit nur von der Rückseite zu sehen sind, bietet sich vor allem ein Tiefentest, das sogenannte *Hidden Surface Removal* an, um diese Polygone zu ermitteln.

Nach Versuchen mit beiden Verfahren wurde deutlich, dass sich trotz einiger Schwächen, der Tiefentest besser eignet. Dafür wird die von *OpenGL* zur Verfügung gestellte *Rendering Pipeline* genutzt. Dabei wird eine Textur im *Framebuffer* erstellt, in die alle Polygone des Objektes gerendert werden. Jedem Polygon wird dafür eine eigene Farbe, die den Index des Polygons repräsentiert, zugeteilt. Alle Lichteinstellungen, die Kantenglättungseffekte, die Transparenz und das *Blending* werden beim *Rasterizer* deaktiviert und das Objekt wird in die entsprechende Referenzposition transformiert, um die eingefärbten Polygone mit eingeschaltetem Tiefentest in die Textur zu rendern. Dadurch werden alle Polygone, die auf einer der Kamera zugewandten Seite des Objektes sind, in der Textur dargestellt (Abbildung 15).

Nun kann die Textur über alle Bildpunkte durchlaufen werden, um das darunterliegende Polygon zu ermitteln und in einer Liste zu speichern. Ob ein Polygon oder Vertex sichtbar ist, kann über die Liste abgefragt werden. Außerdem lässt sich leicht über das Auslesen eines Bild-

punktes aus der Textur ermitteln, welches Polygon des Objektes an einer bestimmten Bildposition zu sehen ist.

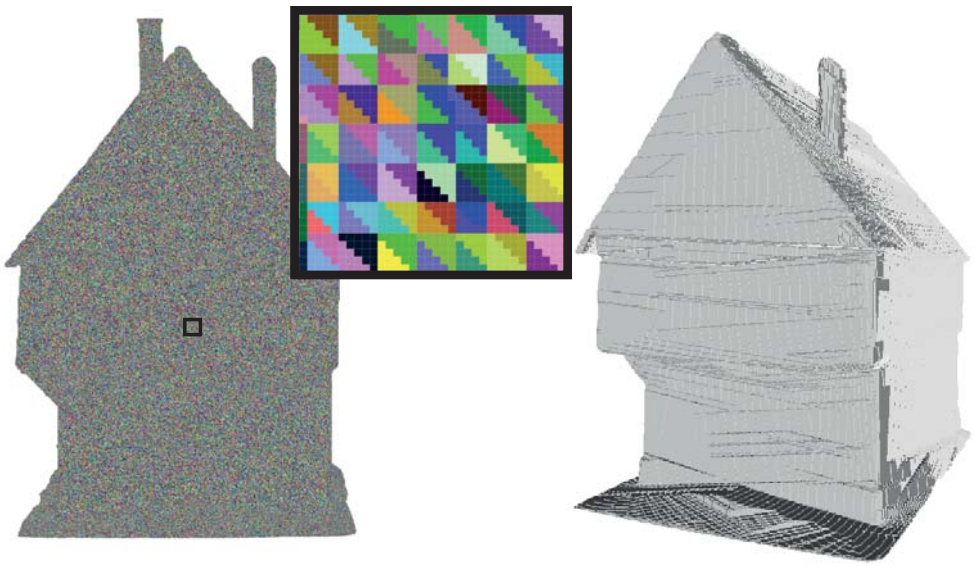


Abbildung 15: Sichtbarkeitstiefentest mit Polygonen in Falschfarben (links) und errechnetes sichtbares Polygonmodell (rechts)

Da die Polygongröße bei der Objekterzeugung an der Bildauflösung ausgerichtet wird, ist die Textur so groß wie das Bild selbst. Allerdings kann es vorkommen, dass einige sichtbare Polygone nicht in der Textur abgebildet werden, da andere Polygone auf denselben Bildpunkten liegen. Deshalb läßt sich die Auflösung der Rendertextur über einen Parameter mit einem festgelegten Faktor vergrößern. Die maximale Texturgröße, die durch die *OpenGL*-Schnittstelle festgesetzt wird, darf jedoch nicht überschritten werden. Aus diesem Grund wird das Verfahren erweitert und man kann in mehrere gekachelte Texturen rendern.

Bei dem *OpenGL-Rasterizer* stellte sich heraus, das bei bestimmten Grafiktreibern und Konfigurationen der Kantenglättungseffekt nicht abgestellt werden kann. Dies führt an den Übergängen der Polygone zu verfälschten Farbresultaten, so dass Polygone in die Sichtbarkeitsliste aufgenommen werden, die eigentlich nicht sichtbar sind (Abbildung 16). Um ein sicheres Ergebnis zu bekommen wird deshalb ein zweites Verfahren, das einen Strahlverfolgungsalgorithmus benutzt, implementiert.

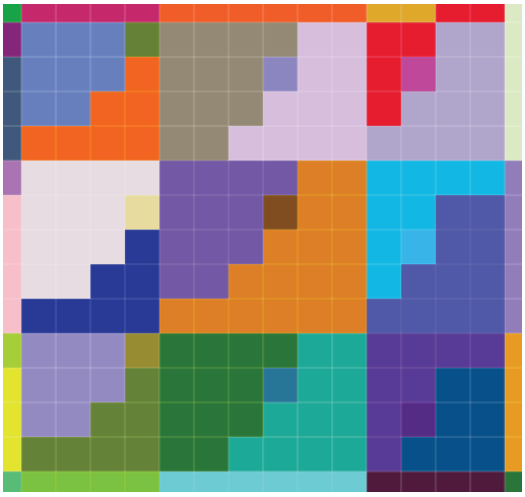


Abbildung 16: Fehler an den Polygonübergängen im Sichtbarkeitstiefentest

4.1.1 Partitionierung

Um auf die einzelnen Komponenten eines so komplexen Polygonobjektes einfacher und schneller zugreifen zu können, muss eine übergeordnete Struktur das Objekt aufteilen. Hierarchische Strukturen sind hierfür nicht geeignet, da die Vertices über die Zellen hinweg verschoben werden und die aufwendige Konstruktion der Struktur in jedem Schritt neu angelegt und verändert werden muss. Eine einfache und für diese Arbeit sinnvolle Datenstruktur ist das gleichförmige *Hashgrid*, das den Raum in würfelförmige Zellen aufteilt (Abbildung 17). Alle Zellen haben bei gleicher Kantenlänge die gleiche Größe und die Anzahl der Zellen s kann in jede Richtung frei gewählt werden.

Bei der Initialisierung wird jedem Vertex des Objektes eine Zelle, in der er sich befindet, zugeordnet. Die Zelle wird über die Indizes c_x, c_y und c_z eindeutig im dreidimensionalen Raster bestimmt. Der lineare Hashwert h kann dann eindeutig berechnet werden:

$$h = c_x + c_y \cdot s_x + c_z \cdot s_x \cdot s_y$$

Außerdem lässt sich optional für jede Zelle eine Liste mit allen darin enthaltenen Vertices anlegen, um schnelleren Zugriff auf die Zelleninhalte zu bekommen. Um die Zelle eines Vertex v zu ermitteln, muss der Raum

einheitlich unterteilt werden. Dafür wird die zugehörige Zellkoordinate c in jeder Dimension i berechnet:

$$c_i = \frac{v_i - m_i}{d_i}$$

Hierfür werden lediglich der unterste Zellpunkt m und die Kantenlänge d der Zelle benötigt.

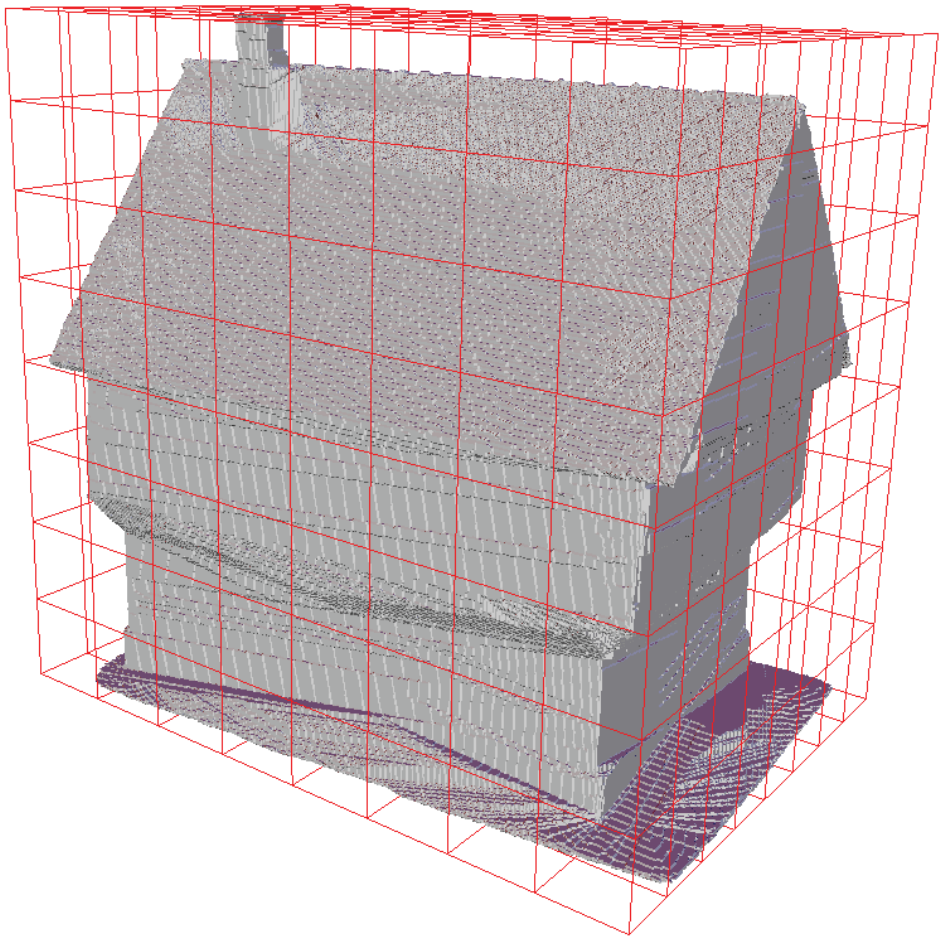


Abbildung 17: Hausmodell partitioniert in einem Hashgrid

4.1.2 Erweiterung durch Strahlverfolgung

Ähnlich wie bei einem *Raytracer*, der in der Computergrafik zur Darstellung einer 3D-Szene genutzt wird, wird bei dem erweiterten Sichtbarkeitstest ein Strahlenverfolgungsalgorithmus eingesetzt [App68]. Zur Ermittlung der Sichtbarkeit eines dreidimensionalen Objektes wird vom Kamerazentrum c ausgehend ein Strahl durch jedem Bildpunkt der Bildebene geschickt. Der Polygonpunkt des Objektes mit der geringsten Distanz zum optischen Zentrum, welcher vom Strahl getroffen wird, ist somit sichtbar. Der Pixel durch den der Strahl geht enthält den codierten Polygonindex. Wird kein Objekt vom Strahl getroffen, so ist der Bildpunkt an dieser Stelle Teil des Hintergrundes.

Ein Strahl t kann als Geradengleichung in Richtung d des Sehstrahls definiert werden:

$$t = c + \lambda d$$

Das kleinste λ, bei dem der Strahl in der entsprechenden Ansicht ein Objekt trifft, ist somit sichtbar. Um den Schnittpunkt eines Strahls mit einem Objekt zu bestimmen, wurde der effiziente Algorithmus von Möller und Trumbore verwendet [MT97]. Ein Punkt p in einem Dreieck mit den Eckpunkten a,b und c ist als baryzentrische Koordinate definiert durch:

$$p(r,s) = (1-u-v)a + ub + vc$$

Gilt dabei $r \geq 0$, $s \geq 0$ und $r + s \leq 1$, dann befindet sich der Punkt innerhalb des Polygons. Ist $p = t$, so schneidet der Strahl das Dreieck und somit gilt:

$$c + \lambda d = (1-u-v)a + ub + vc$$

Über die *Cramersche Regel* erhält man mit $w = c - a$ die Gleichung:

$$\begin{bmatrix} q \\ r \\ s \end{bmatrix} = \frac{1}{(d \times v) \cdot u} \cdot \begin{bmatrix} (w \times u) \cdot v \\ (d \times v) \cdot w \\ (w \times u) \cdot d \end{bmatrix}$$

Bei der Implementierung stellte sich heraus, dass geschnittene Punkte, die genau auf der Kante oder auf dem Eckpunkt liegen, als nicht getroffen erkannt werden. Deshalb muss beim Vergleichstest ein sehr geringes ϵ aufaddiert werden.

Die geschnittenen sichtbaren Polygone können in einer Liste abgelegt und abgefragt werden, wie es auch bei dem Rasterisierungsverfahren in Abschnitt 4.1 besprochen wurde.

Zur Beschleunigung wird oft eine hierarchische Struktur eingesetzt, wie etwa die Kd-Bäume oder Oktalbäume, die die Polygone räumlich aufteilen. Ansonsten muss jedes Polygon mit jedem Strahl getestet werden, so dass der Aufwand exponentiell mit der Anzahl der Polygone wächst. In dieser Arbeit kann die Partitionierung von einem *Hashgrid*, wie er im vorherigen Absatz 4.1.1 besprochen wurde, übernommen werden. Bei dem Durchlauf des Algorithmus muss zuerst getestet werden, ob eine der Zellen getroffen wurde. Alle Polygone, die dann innerhalb der geschnittenen Zellen liegen, werden wie oben beschrieben weiter behandelt.

4.2 Verdeckungsanalyse

Probleme bei der Kongruenzanalyse entstehen hauptsächlich durch Rauschen, sowie durch Konfigurations- und Kalibrierungsfehler, wie sie in Abschnitt 3 beschrieben wurden, oder weil korrespondierende Bildpunkte in einer oder mehreren Ansichten verdeckt sind [NMSO96]. Andere Probleme treten durch periodische Strukturen oder durch homogene Bereiche auf, die zu mehrdeutigen Resultaten führen können. Nach den Autoren Nakamura et al. ist es bei den Stereoverfahren mit zwei Ansichten kaum möglich, diese Verdeckungsfehler zu finden und ohne globale Informationen auszubessern [NMSO96]. Stehen jedoch mehr als zwei Kameraansichten zur Verfügung, können diese Fehler korrigiert werden. Auch wenn durch mehr Ansichten vermehrt Verdeckungen entstehen, erreicht man letztendlich doch bessere Ergebnisse.

Man kann bei der Korrespondenzanalyse zwei Arten von Verdeckungsfehlern unterscheiden (Abbildung 18). Es gibt Fehler, die nur

auf einer Seite der Referenzansicht auftauchen können und die anhand der anderen Ansichten erkannt und ausgeglichen werden können. Der zweite Fehlertyp kann dagegen nicht korrigiert werden, da das Objekt im Vordergrund nur für eine Kamera sichtbar ist und die anderen Kameras auf das dahinter liegende Objekt schauen und den Referenzpunkt nicht sehen.

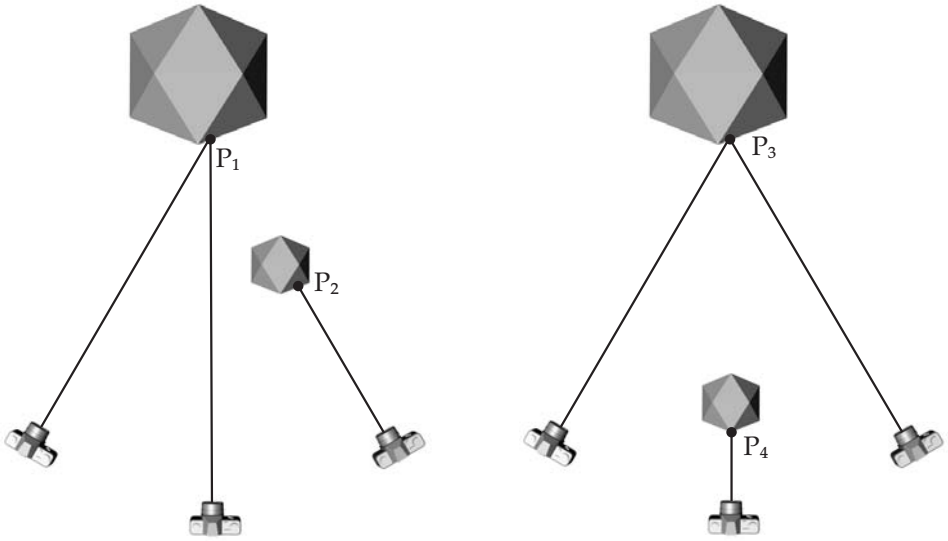


Abbildung 18: (links) Verdeckungsfehler, der für den Punkt P_1 erkannt und korrigiert werden kann, (rechts) Verdeckungsfehler, bei dem nur eine Ansicht den Punkt P_4 sieht

Viele Arbeiten ignorieren das Verdeckungsproblem und verschieben solche Fehler in die Nachbearbeitung. Bei den *Stereo*-Verfahren haben sich laut Cyganek und Siebert einige effektive Methoden der Verdeckungsanalyse etabliert [CS09]. Eine beliebte Technik ist das *Cross Checking* Verfahren (LRC), bei dem zuerst mit dem linken Bild als Referenz die Tiefenkarte berechnet wird und dann anschließend mit dem rechten Bild, um daraufhin beide miteinander zu vergleichen [MMHM02]. Bei dem *Occlusion Constraint* (OCC) werden Unstetigkeiten in der Tiefenkarte gesucht, um sie dann, ähnlich wie bei dem LRC-Verfahren, in beiden Ansichten zu prüfen. Beide Techniken haben in Versuchen gute Ergebnisse gezeigt [EW02, CS09]. Das *Point Ordering Constraint* (ORD) prüft die Reihenfol-

ge der Übereinstimmungen in beiden Ansichten. Bei der *Bimodality* Methode (BMD) wird untersucht, ob sich im Histogramm Extrema an den Rändern zur Verdeckung in der Tiefenkarte abzeichnen. Mit den *Match Goodness Jumps* (MGJ) wird geprüft, ob der Fehler bei der Kongruenzanalyse durch eine eventuelle Übereinstimmung größer wird.

Kang et al. schlagen bei den *Multiview*-Verfahren zwei verschiedene Ansätze vor [KSC01]. Eine weit verbreitete Methode ist das *Shiftable Window*, bei dem für den Vergleichstest eine vorgegebene verschiebbare Region um einen Pixel herum benutzt wird. Punkte, die verdeckt sind, können dadurch ignoriert werden und man vergleicht nur die sichtbaren Bildpunkte der Region. Beim zweiten Verfahren werden verdeckte Punkte markiert und nur die wirklich sichtbaren Punkte werden bei der Minimierung berücksichtigt. Dies kann über die einseitige Auswahl der Besseren von beiden Seiten einer Referenzansicht geschehen oder man verwendet einfach die besten 50% der Ansichten für die Fehlerberechnung. Mit der zweiten Methode erreichten Kang et al. bessere Ergebnisse [KSC01].

Nach den Versuchen mit den verschiedenen Verfahren wird deshalb auch der zweite Ansatz für diese Arbeit verwendet. Man kann über Parameter die Verdeckungsanalyse ein- und ausschalten und zwischen einseitiger Auswahl und der Auswahl der besten Ansichten wählen.

4.3 Lokale Optimierung mit Simulated Annealing

Für die bildbasierte Optimierung des Polygonnetzes werden zwei verschiedene Verfahren implementiert und miteinander verglichen. Das lokale Verfahren wird mittels *Simulated Annealing* optimiert und benutzt eine komplexe Optimierungsfunktion zur Anpassung des Objektes. Der globale Ansatz verwendet den *Graph Cut* für die Optimierung. Um den Informationsgehalt der Bilddaten vollständig zu nutzen, verwenden beide Verfahren einen flächenbasierten Ansatz für die Kongruenzanalyse. Dadurch erhält man dichte Tiefeninformationen für das gesamte Objekt.

Bei der iterativen lokalen Optimierung wird die initialisierende visuelle Hülle an das tatsächliche Objekt schrittweise angenähert. Dafür werden die Bilddaten vollständig ausgewertet, um neben der Silhouetteninforma-

tion ein erweitertes Verständnis der Objektgeometrie zu bekommen. Das Verfahren kann in zwei unabhängige Schritte eingeteilt werden. Im ersten Schritt werden die Vertices des Objektes ausgewählt, die in einer Referenzansicht sichtbar sind (Abschnitt 4.2). Eine Kostenfunktion wird dann im zweiten Schritt für jeden sichtbaren Vertex erstellt und optimiert.

Dabei wird das Modell iterativ so angepasst, dass eine maximale Übereinstimmung der projizierten Bildpunkte in den verschiedenen Ansichten erreicht wird. Die Iteration erfolgt lokal über alle Vertices und alle Tiefeneinstellungen einer Referenzansicht. Daraus erhält man viele gleich gute Ergebnisse und somit viele verschiedene optimale Objekte. Weitere Kriterien wie die Silhouettenkonsistenz, verschiedene Glattheitseigenschaften und die Polygongüte verbessern die Qualität der Kostenfunktion und führen zu besseren und eindeutigen Resultaten.

Bei der Iteration wird immer nur ein Vertex um eine fest vorgegebene Schrittweite verschoben. Die Richtung der Verschiebung ist die verlängerte Sichtachse zwischen dem Vertex und der aktuellen Kameraposition. Nach jedem Schritt wird die Kostenfunktion aufgestellt und mit den Kosten der letzten Vertexposition verglichen.

4.3.1 Kostenfunktion

Pro Iterationschritt wird jeweils ein Vertex verschoben und dabei wird die Qualität des Polygonnetzes mit der Qualität der letzten Position verglichen. Diese wird über eine Kostenfunktion für den Vertex v wie folgt bestimmt:

$$E(v) = \lambda_C E_C + \lambda_{S1} E_{S1} + \lambda_{S2} E_{S2} + \lambda_Q E_Q + \lambda_D E_D + \lambda_B E_B + E_H$$

Der wichtigste Faktor der Kostenfunktion ist die Farbübereinstimmung E_C einer Vertexposition. Die beiden Glattheitsterme E_{S1} und E_{S2} sollen eine stetige Oberfläche garantieren. E_Q steht für die Qualität der um einen Vertex liegenden Polygone. Der Tiefenterm E_D bestraft eine zu große Abweichung von der visuellen Hülle mit einem hohen Fehlerwert. Über den Term E_B sollen Tiefenausreißer getilgt werden. Um einen Ausbruch aus der visuellen Hülle zu verhindern wird der Parameter E_H ge-

setzt. Diese Terme können über die benutzerdefinierten λs an die Objekte angepasst werden.

4.3.2 Farbkorrespondenz

Der wichtigste Term ist die Farbübereinstimmung E_C eines Vertex v für alle gegebenen Ansichten. Der Vertex soll dabei so verschoben werden, dass die Farbabweichungen zu einem Referenzbild in den verschiedenen Perspektiven minimiert wird. Die Richtung der Verschiebung ergibt sich aus der Kameraposition der Referenzansicht und dem Bildpunkt selbst. Dafür wird der Vertex auf dem Strahl, ausgehend von der Kamera c und durch den Bildpunkt p_i gehend, in Richtung des Objektes verschoben. Da die Verschiebung nur auf den Sichtstrahl der Referenzansicht reduziert wird, schränkt man den Suchraum auf eine Gerade ein. Der neu verschobene Punkt wird in alle ausgewählten Ansichten rückprojiziert, um die Farbübereinstimmung zu messen.

Um dichtere Tiefeninformationen zu erhalten, wird jeder Bildpunkt der Referenzansicht virtuell verschoben. Daraus wird eine dichte Tiefenkarte erzeugt, die als Grundlage für die Verschiebung der Vertices dient.

Dafür muss der Bildpunkt in den Raum auf das Objekt projiziert werden. Durch den verwendeten Sichtbarkeitstest (Abschnitt 4.1) kann aus der aktuellen Bildposition der Vertexindex aus dem Indexbild ausgelesen werden. Über die Eckpunkte a, b und c des indexierten Dreiecks und die baryzentrische Koordinaten u und v des Schnittpunktes im Polygon lässt sich dann die Weltkoordinate p_w errechnen.

$$p_w = a + u(c - a) + v(b - a)$$

Erst danach kann der Objektpunkt zurück in eine der Bildansichten abgebildet werden. Das Skalarprodukt des Punktes als homogene Koordinate und der Projektionsmatrix P der jeweiligen Ansicht ergibt die neue Bildposition p_i.

$$p_i = P \cdot p_w = \begin{bmatrix} P_{11} & P_{12} & P_{13} & P_{14} \\ P_{21} & P_{22} & P_{23} & P_{24} \\ P_{31} & P_{32} & P_{33} & P_{34} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

Für die Kongruenzanalyse wurden in Abschnitt 2.6.1 verschiedene Verfahren vorgestellt. Das *Lambertsche* Lichtmodell, das unabhängig vom Sichtwinkel der Ansichten ist, wird als Berechnungsgrundlage vorrausgesetzt. Die Intensität des Lichts ist damit nur abhängig vom Einstrahlwinkel der Lichtquelle zur Oberflächennormalen. Werden die Punkte mit einer korrekten Position des Raumpunktes in die verschiedenen Ansichten zurückprojiziert, hat jeder Bildpunkt die gleiche Farbhelligkeit. Eine Farbabweichung in einer Ansicht deutet auf eine Fehlstellung des Raumpunktes hin. Allerdings müssen die in Abschnitt 4.2 besprochenen Verdeckungsprobleme und etwaige homogene Bildregionen berücksichtigt werden.

Die Kongruenzanalyse kann in zwei Phasen aufgeteilt werden. In der ersten Phase wird die Korrespondenz der einzelnen Bildpunkte bestimmt. Die Aggregationskosten werden in der zweiten Phase aus den Korrespondenzkosten aufsummiert.

Es wird die Übereinstimmung vom Referenzbild I_r zu allen ausgewählten Ansichten I_i gemessen. Diese Farbdifferenzen werden über die *Summe der absoluten Differenzen* D_{SAD} oder über die *Summe der quadratischen Differenzen* D_{SSD} ermittelt. Dafür wird der Fehler für jeden der drei Farbkanäle des *RGB*-Farbraums eines Bildpunktes aufaddiert. Diese Fehler werden wiederum aufsummiert und durch die Anzahl der Ansichten dividiert.

$$E'_C = \frac{1}{|I|} \sum_{i \in I} D_{SAD/SSD}(I_r, I_i)$$

Die Korrespondenzkosten E'_C werden in einer Tiefenkarte gespeichert. Da nur einzelne Pixel lokal betrachtet werden, können große Schwankungen in der Tiefenkarte entstehen. Deshalb werden die Kosten in der zweiten Phase für jeden Vertex aus der Tiefenkarte über einen vorgegebenen

symmetrischen Fensterbereich errechnet. Dafür wurden mehrere vorgestellte Methoden getestet (Abschnitt 2.6.2) und neben dem einfachen *Box*-Filter, also der Summe der Tiefenfehler (SSAD/SSSD), wurde auch der *Gauß*-Filter implementiert.

$$E_C = \frac{1}{i} \sum E'_C(x,y) \quad ,(x,y) \in I$$

Durch die Verwendung der Aggregation wird das Rauschen reduziert und eine errechnete Vertexposition ist dadurch von mehreren Bildpunkten abhängig. Die Fensterbreite sollte nicht zu klein sein, um eine Wirkung zu erkennen aber auch nicht zu groß, um eine zu starke Glättung der Tiefenkarte zu vermeiden.

Die in Abschnitt 4.2 vorgestellten Verdeckungstests können hier direkt angewendet werden. Dafür werden die Korrelationskosten für die linke und rechte Seite einer Referenzansicht separat berechnet und auf diskrete Gleichverteilung geprüft. Falls die Fehlerdifferenz der beiden Seiten einen vorgegebenen Schwellenwert überschreitet, wird von einer Verdeckung des Bildpunktes ausgegangen. Wie bereits vorgeschlagen, können die Fehler der verdeckten Seite nun verworfen werden und die restlichen Fehler werden in der Tiefenkarte aufsummiert. Die Aggregationskosten für die Vertices werden wieder aus der Tiefenkarte errechnet.

4.3.3 Glattheitseigenschaften

Die erweiterte Farbkorrespondenz reicht nicht aus um alle Ausreißer und Fehler zu erkennen und zu eliminieren. Deshalb müssen weitere Eigenschaften des Objektes an die Optimierungsfunktion gebunden werden, um die Kostenfunktion sinnvoll zu erweitern. Eine wichtige Objekteigenschaft von Gebäudemodellen und vielen anderen erzeugten Gegenständen, ist die Ebenheit.

Um die Glattheit zu forcieren, wird ein lokaler und ein erweiterter lokaler Glattheitsparameter berechnet. Für den lokalen Parameter E_{S1} wird die Normale des Vertex mit den Normalen der anliegenden Nachbarpolygone verglichen. Die Polygonnormalen werden aufaddiert, um ein Skalarprodukt mit der Vertexnormalen zu bilden. Das Skalarprodukt der

normalisierten Vektoren definiert den Winkel zwischen den Vektoren. Ist der Winkel groß, so ist die Oberfläche an dieser Stelle uneben.

Da dieser Glattheitsparameter beschränkt lokal wirkt wurde ein zweiter Wert E_{S2} eingeführt, bei dem der zweite Ring an Nachbarpolygonen für die Berechnung genutzt wird. Hierbei wird das Skalarprodukt der Normalen der erweiterten Nachbarpolygone und der Vertexnormalen gebildet, um den Winkel zu vergleichen.

Beide Parameter können unabhängig voneinander gewichtet werden, um verschiedene Objekteigenschaften zu bevorzugen.

4.3.4 Polygonqualität

Das Qualitätskriterium für Dreiecke wurde von Cavendish vorgeschlagen [Cav74]. Es kann als normierte Summe der Seitenquadrate und der Dreiecksfläche A berechnet werden [KHV04, EWN04].

$$E'_Q = \frac{4 \cdot \sqrt{3} \cdot A}{a^2 + b^2 + c^2}$$

Für ein gleichseitiges Dreieck gilt $E'_Q = 1$ ist maximal. Je kleiner der Wert wird, umso schlechter ist somit die Qualität eines Dreiecks. E_Q ist die Summe aller $1 - E'_Q$s der adjazenten Polygone.

4.3.5 Distanzkriterien

Mit dem Tiefenterm E_D kann verhindert werden, dass Vertices zu weit verschoben werden. Die Kosten dieses Terms steigen mit der Tiefe der Verschiebung s bis zur maximalen Tiefe t. Es gilt also:

$$E_D = s/t$$

Da die Kostenfunktion tiefere Verschiebungen bevorzugt, kann mit diesem Parameter gegengesteuert werden.

Die Terme E_{S2} und E_Q sollen auch Tiefenausreißer innerhalb eines Polygonnetzes verhindern. Allerdings können diese Parameter oft nicht hoch genug gewichtet werden, da sie sonst das Objekt zu stark glätten.

Deshalb wurde der Parameter E_B eingeführt. Dabei werden die Tiefeneinstellungen der Nachbarvertices mit dem aktuellen Vertex verglichen. Je größer die Tiefenunterschiede sind, desto höher fällt der Fehler aus. Der Tiefenunterschied wird über die Farbinformation der Bildpunkte an den Vertexpositionen gewichtet, um dadurch Kanten an Übergängen zuzulassen. Bei gleicher Farbintensität ist die Gewichtung hoch.

Um zu verhindern, dass Vertices aus der visuellen Hülle geschoben werden, wurde der Parameter E_H eingeführt. Dabei wird für alle vorhandenen Ansichten geprüft, ob die neue Position des aktuellen Vertex noch innerhalb der visuellen Hülle liegt. Der Vertex wird über die Projektionsmatrix P, ähnlich wie in Abschnitt 4.3.2 besprochen, in die verschiedenen Ansichten projiziert. Falls der neue Bildpunkt in einer Ansicht außerhalb des Objektes liegt, so wird E_H auf den maximalen Fehler gesetzt. Es gilt also für alle Ansichten:

$$E_H = \begin{cases} 0 & \text{falls} \quad p \in h_p \\ \max & \text{sonst} \end{cases}$$

4.3.6 Optimierungsfunktion

Bei der lokalen Optimierung wird jeder Vertex v separat verschoben, um eine optimale Position für diesen zu finden. Diese optimale Position ist durch die minimalen Kosten in der Kostenfunktion $E(v)$ bestimmt. Dafür wird der Vertex auf dem Sichtstrahl $s = v - c_i$ zwischen dem aktuellen Vertex und der Kamera der Referenzansicht, wie in Abschnitt 4.3.2 besprochen, verschoben. Die Schrittweite d und Tiefe t bestimmen die neue Position $v' = v - s \cdot d \cdot t$, wenn die Kosten durch die Verschiebung verringert werden können

Die Vertices werden strikt hintereinander abgearbeitet, wenn eine Verbesserung erreicht wird. Falls ein Vertex nicht verschoben wird, weil die Kosten nicht verringert werden können, werden zuerst alle Nachbarvertices betrachtet, da ein Vertex nicht unabhängig von den adjazenten Vertices ist. So können sich etwa Vertices über die Glattheitsbedingungen E_{S1} und E_{S2} gegenseitig auf einer Ebene blockieren, wenn die Farbkorrespondenz E_C nicht stark verbessert wird. Deshalb werden die Nachbarn

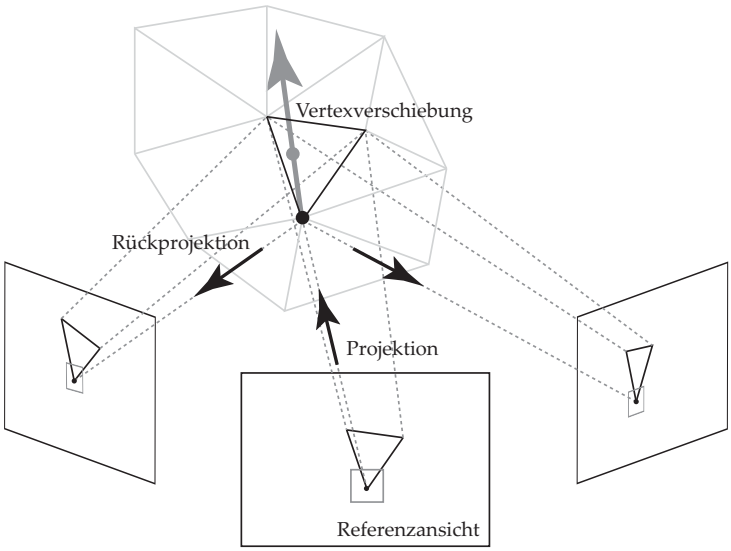


Abbildung 19: Lokale Optimierung mit einer Projektion der Bildinformationen auf das Objekt und anschließender Rückprojektion in die anderen Ansichten

ebenfalls schrittweise von ihrer aktuellen Position verschoben, bis sie die Tiefe des aktuellen Vertex erreichen. Falls sich die Kosten in einem Schritt positiv verändern, wird die neue Position beibehalten.

Um diesen aufwendigen Vorgang nicht bei jeder Vertexverschiebung ausführen zu müssen, wurde das *Simulated Annealing* Verfahren implementiert. Dabei handelt es sich um ein heuristisches Optimierungsverfahren für diskrete Probleme, welches einen Abkühlungsprozess nachbildet. Die *Fitness*-Funktion $f : D \rightarrow \mathbb{R}$ soll dabei von einem festgelegten Startwert $x \in D$ in einer vorgegebenen Anzahl an Schritten $i \in \mathbb{N}$ auf 0 reduziert werden. Der Kurvenverlauf der Funktion kann dabei frei gewählt werden und ist in dieser Arbeit linear. In jedem Schritt wird geprüft, ob die Kosten $E(v)$ durch eine Vertexverschiebung gesenkt werden können. Ist dies nicht möglich, so wird trotzdem mit einer vorgegebenen Wahrscheinlichkeit, die dem Kurvenverlauf folgt, der neue Zustand übernommen. Die Nachbarvertices werden nun verschoben, um die neuen Kosten abzuschätzen. Anfangs werden dabei fast alle Einstellungen akzeptiert. Mit dem fallenden Kurvenverlauf werden jedoch nur noch starke Schwankungen zugelassen. Werden die Kosten gesenkt, so wird

die neue Position beibehalten. Damit können lokale Minima verlassen werden, um einen eventuell besseren Zustand einzunehmen.

```
FOR t ← 1 TO N DO
  FOR x,y ∈ I DO
    berechne Tiefenfehler für x,y
  END FOR
  FOR v ∈ M DO
    verschiebe v und berechne Fehler E(v)
    IF E'(v) > E(v) THEN
      setze neuen Tiefenwert für v
      E'(v) := E(v)
    ELSE
      berechne Wahrscheinlichkeit P*
      IF P* > E(v) THEN
        wähle alle Nachbarn V von v
        FOR n ∈ V DO
          verschiebe n und berechne Fehler E(n)
          IF E'(n) > E(n) THEN
            setze neuen Tiefenwert für n
            E'(n) := E(n)
            IF E'(v) > E(v) THEN
              setze neuen Tiefenwert für v
              E'(v) := E(v)
            END IF
          END IF
        END FOR
      END IF
    END IF ELSE
  END FOR
END FOR
```

Quelltext 1: Pseudocode der *Simulated Annealing* Optimierung

4.4 Globale Optimierung mit Graph Cuts

Eine globale Optimierung des Polygonnetzes mit einem flächenbasierten Verfahren kann über den *Graph Cut* Algorithmus gelöst werden. Für die Konstruktion des Graphen und für den Aufbau der Kostenfunktion haben sich verschiedene Ansätze etabliert [RC98, BVZ01, KZ01, Kol04, CS09]. Diese Verfahren arbeiten allerdings direkt mit den Bilddaten und erzeugen daraus Tiefenkarten, aus denen ein dreidimensionales Modell konstruiert wird. In dieser Arbeit wird dagegen ein bereits vorhandenes simples Modell der visuellen Hülle eines Ojektes optimiert, so dass das erzeugte Polygonnetz als zu optimierender Graph betrachtet werden kann. Die Optimierung erfolgt hier, wie schon bei dem lokalen Verfahren, schrittweise für jede Tiefenstufe.

Wie bei der lokalen Optimierung kann auch der globale Algorithmus in zwei unabhängige Schritte eingeteilt werden. Im ersten Schritt wird die Sichtbarkeit der Vertices geprüft (Abschnitt 4.2). Der *Graph Cut* Algorithmus optimiert, folgend im zweiten Schritt, alle sichtbaren Vertices gleichzeitig pro Tiefenstufe.

Dafür muss eine Kostenberechnung für jeden Vertex in jeder Tiefenstufe erfolgen. Diese Kosten werden global zusammengefasst und sukzessiv global über den minimalen Schnitt in dem gerichteten Graphen optimiert. Der Algorithmus kann zur Verbesserung der Ergebnisse mehrmals hintereinander ausgeführt werden.

4.4.1 Kostenfunktion

Die Kostenfunktion des *Graph Cut* Algorithmus ist einfacher aufgebaut als bei der lokalen Optimierung. Die Kosten werden dabei nur über zwei gewichtete Parameter global aufsummiert:

$$E = \sum \lambda_C E_C(v) + \sum \lambda_S E_S(v, v_n)$$

Der Korrespondenzterm E_C gibt die Kosten der Farbübereinstimmung der Bildpunkte wieder. Um die Ebenheit der Oberfläche zu bestimmen, wird der Glattheitsterm E_S hinzugezogen. Beide Terme können über die benutzerdefinierten λs an die gegebenen Objekte angepasst werden. Bei

der Optimierung wird nun das Minimum für die Gesamtkosten aller Vertices gesucht.

4.4.2 Farbkorrespondenz

Analog zur Korrespondenzanalyse beim lokalen Verfahren wird beim *Graph Cut* Algorithmus die Farbübereinstimmung E_C eines Vertex v berechnet (Abschnitt 4.3.2). Dazu wird eine Tiefenkarte erstellt, in der jedem Bildpunkt ein Tiefenfehler, der sich aus den Farbabweichungen aller relevanten Ansichten I_i ergibt, zugeordnet wird. Die Bildpunkte werden dafür auf das Objekt projiziert und jeweils um einen Tiefenschritt in Richtung des Sehstrahls einer Kameraansicht verschoben.

Die Farbübereinstimmung wird über die *Summe der absoluten Differenzen* D_{SAD} oder über die *Summe der quadratischen Differenzen* D_{SSD}, vergleichbar zu dem lokalen Verfahren, ermittelt. Anschließend werden die Fehler über den *Gauß*-Filter für jeden Vertex aus einer vorgegebenen Fenstergröße akkumuliert.

$$E_C = \frac{1}{i} \sum_{(x,y)\in I} \frac{1}{|I|} \sum_{i\in I} D_{SAD/SSD}(I_r, I_i)$$

Die vorgestellten Verdeckungstests (Abschnitt 4.2 und Abschnitt 4.3.2) können hier ebenfalls direkt angewendet werden.

4.4.3 Glattheitsbedingungen

Bei dem vorgestellten *Graph Cut* Verfahren werden die Glattheitswerte aus den Bildpunkten der Tiefenkarte gewonnen [RC98, BVZ01, KZ02a, KZ02b]. Die vier Nachbarn sind die direkten Pixelnachbarn und diese sollten eine ähnliche Disparität aufweisen. Der Fehler steigt also mit der Differenz der Abstände in der Tiefenkarte.

Diese Methode ist einfach und effektiv, aber sie kann nicht auf diese Arbeit übertragen werden, da hier ein vorhandenes Polygonnetz optimiert wird. Dadurch hat man bereits verschiedene Tiefenstufen für die Vertices, die nicht in der Tiefenkarte abgebildet werden. Eine Rückprojektion der Oberfläche in die Tiefenkarte wäre zwar möglich, aber wegen

des Berechnungsaufwands wenig sinnvoll. Deshalb werden alternative Methoden benutzt, die für das *Mesh* genauere Ergebnisse liefern können.

Die einfachste Methode ist, die Tiefenstufen $s \in \mathbb{N}$ der Verschiebungen für jeden Vertex zu speichern, um diese miteinander zu vergleichen. Die quadratische Differenz der Tiefenstufen zweier Vertices entspricht dem Glattheitsfehler $E_S(v_1, v_2) = (s_{v_1} - s_{v_2})^2$. Da jedoch die Ausgangstiefe der Vertices, wie oben angedeutet, nicht berücksichtigt wird, führt diese Methode nicht immer zu guten Ergebnissen.

Um die Tiefe zu berücksichtigen, wird die Distanz $d \in \mathbb{R}$ der Vertices zum Kamerazentrum c einer Ansicht gemessen. Aus der Distanz $d = v - c$ kann nun die quadratische Differenz berechnet werden um die Glattheit zu bestimmen $E_S(v_1, v_2) = (d_{v_1} - d_{v_2})^2$. Wenn man die Distanz durch die Schritttiefe einer Verschiebungsstufe teilt, erhält man ein Maß, das dem oben beschriebenen ersten Verfahren entspricht.

Eine andere Methode, die dem Glattheitsterm des lokalen Verfahrens ähnlich ist, kann über die Normalen der Vertices berechnet werden. Dafür errechnet man aus den adjazenten Polygonen der beiden Vertices die Vertexnormalen. Das Skalarprodukt der Normalen n definiert den Winkel, der den Glattheitsfehler $E_S(v_1, v_2) = (n_{v_1} \cdot n_{v_2})^2$ bestimmt.

Da jedoch an vielen Stellen des Objektes Winkelabweichungen erwünscht sind, wurden die Bildinformationen für eine zusätzliche Gewichtung hinzugezogen. Geht man davon aus, dass glatte Flächen in der Regel eine ähnliche Farbe aufweisen sollten, so kann man die Farbdifferenz l der Bildpunkte unter den projizierten Vertices für eine zusätzliche Gewichtung nutzen. In dieser Arbeit wird die maximale absolute Differenz der drei Farbkanäle dieser beiden Bildpunkte berechnet und mit einem Schwellenwert verglichen. Liegt der Wert unterhalb der Schwelle, so wird ein quadratischer Glattheitsfehler berechnet. Liegt der Wert darüber, so wird ein niedriger Konstantenwert an die Kostenfunktion weitergereicht.

$$E_S(v_1, v_2) = \begin{cases} s^2 & \text{falls} \quad l < \tau \\ \lambda & \text{sonst} \end{cases}$$

Eine Erweiterung dieser Methode wird über eine angepasste *Cauchy*-Funktion realisiert.

$$E_S(v_1, v_2) = \lambda \cdot \left(l \cdot log \left(1 + \frac{s^2}{l} \right) \right)$$

Zieht man die Farbdifferenz von einem Maximalwert ab, so wächst die Funktion bei hoher Farbübereinstimmung wie eine quadratische Funktion und entwickelt sich bei hoher Differenz zu einer flach abfallenden Kurve. Dadurch werden Werte mit großer Farbdifferenz geringer bestraft als Werte mit niedriger Differenz, die im schlechtesten Fall quadratisch gewichtet werden.

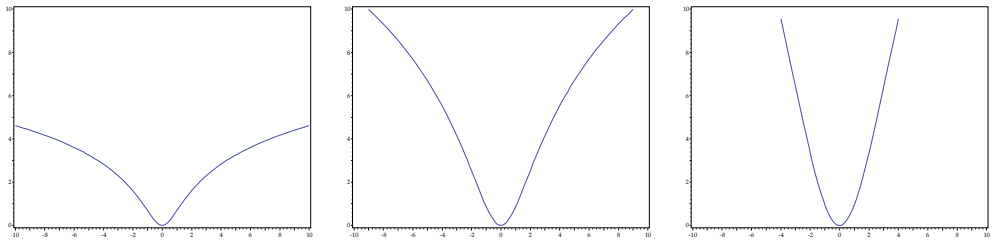


Abbildung 20: Angepasste *Cauchy-Funktion* mit der Farbdifferenz $l = \{1, 3, 10\}$

4.4.4 Graph Cut Verfahren

Zur Optimierung der Kostenfunktion wird ein effizientes *Graph Cut* Verfahren verwendet. Dabei wird die Optimierung als *labeling*-Problem verstanden und als solches gelöst [Kol04]. Diese Labels sind bei der Tiefenrekonstruktion die Fehler der Tiefeneinstellung der Vertices.

Zur Lösung wird ein gewichteter Graph $G = (V, E)$ aus den Knoten V, die hier aus den sichtbaren Vertices des *Meshes* M bestehen, und den gerichteten Kanten E, denen die Kosten c zugeordnet werden, zusammengesetzt. Die Kanten werden aus den Nachbarschaftsbedingungen abgeleitet und die Kosten werden durch die vorgestellte Kostenfunktion $E(v)$ zugeordnet. Der Graph wird um zwei weitere Knoten, die als Terminalknoten bezeichnet werden, erweitert. Die sogenannte Quelle s und die

Senke t werden mit jedem sichtbaren, zu optimierenden Vertex verbunden. Man unterscheidet n-Kanten, die aus der Nachbarschaftsbeziehung abgeleitet werden und denen die Farbkorrespondenzkosten $E_C(v)$ zugeordnet werden, und den t-Kanten, die Vertices mit den Terminalknoten verbinden, und die Kosten des Glattheitsterms $E_S(v_1, v_2)$ repräsentieren (Abbildung 21).

Auf diesem Graphen wird ein minimaler s-t-Schnitt ausgeführt, der über den maximalen Fluss ermittelt wird [Sed92]. Dabei wird der Graph so geteilt, dass jeder Vertex mit einem Terminalknoten verbunden bleibt. Es gilt also $s \in S$ und $t \in T$ mit $S = V \setminus T$. Die Gesamtkosten $C \subset E$ des s-t-Schnittes sind die Summe der Kantenkosten zwischen S und T:

$$C(s,t) = \sum_{v_i \in S, v_j \in T} c(s,t)$$

Bei einem minimalen Schnitt handelt es sich also, um den Schnitt durch einen Graphen mit den geringsten Kosten. Dieser Schnitt wird über den maximalen Fluss zwischen den Knoten im Graphen nach einer erweiterten Methode von Ford und Fulkerson gefunden [FF56, FF62, Din70, BK04, Kol04]. Der maximale Fluss von s nach t sättigt die Knoten in dem Graphen, so dass der Schnitt durch die disjunkten Mengen S und T den minimalen Kosten entspricht [Kol04].

4.4.5 Konstruktion des Graphen

Boykov et al. haben zwei Algorithmen entwickelt, um das NP-schwere Minimierungsproblem der globalen Kostenfunktion in polynomialer Zeit zu approximieren [BVZ01]. Bei allen etablierten Verfahren zur Optimierung wird pro Schritt immer ein Vertex oder Bildpunkt behandelt [CS09]. Mit den Methoden von Boykov et al. können dagegen ganze Regionen gleichzeitig bearbeitet werden, so dass die Methode schnell konvergiert. Sie stellen dafür zwei Verfahren, den *α-β-Swap* und die *α-Expansion* vor, die gleich gute Ergebnisse liefern (Abbildung 22) [BVZ01].

Beim *α-β-Swap* werden Bereiche eines Labels mit Bereichen des anderen Labels getauscht. Werden die globalen Kosten durch den Tausch

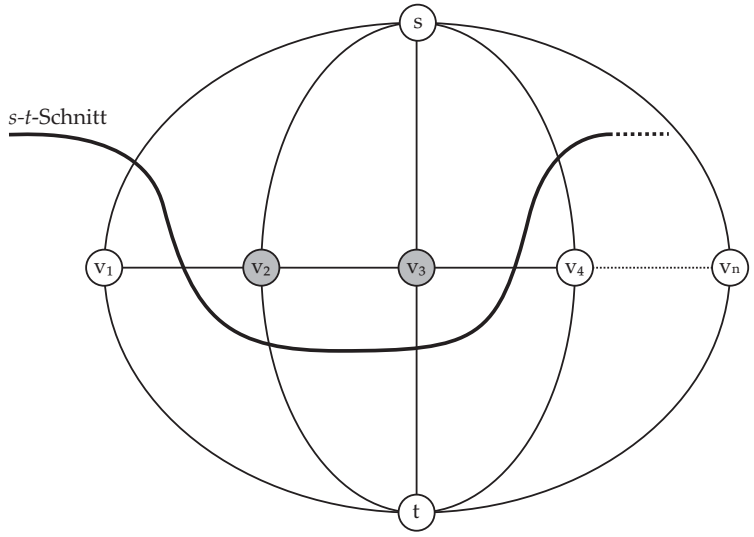


Abbildung 21: Ein s-t-Schnitt teilt den Graphen G in zwei Teilgraphen S und T

verbessert, so wird die neue Konfiguration angenommen. Bei der α-*Expansion* wird eine Region in die andere expandiert, um eine Minimierung der Kosten zu erzielen. Bei einem erfolglosen Durchlauf wird der Algorithmus garantiert beendet.

```
start mit beliebigem Labeling f ∈ L
DO
  setze erfolgreich := FALSE
  FOR EACH α ∈ L DO
    finde g = arg minE(f') in einer α Erweiterung
    IF (E(g) < E(f)) THEN
      setze f := g
      setzte erfolgreich := TRUE
    END IF
  END FOR
WHILE (erfolgreich = TRUE)
```

Quelltext 2: Pseudocode der *Graph Cut* α-Entfaltung

In dieser Arbeit wird die α-Entfaltung benutzt, um den Schnitt im Graphen zu minimieren, der angibt, welche Vertices verschoben werden sollen und welche ihre Position behalten. Der Algorithmus iteriert über

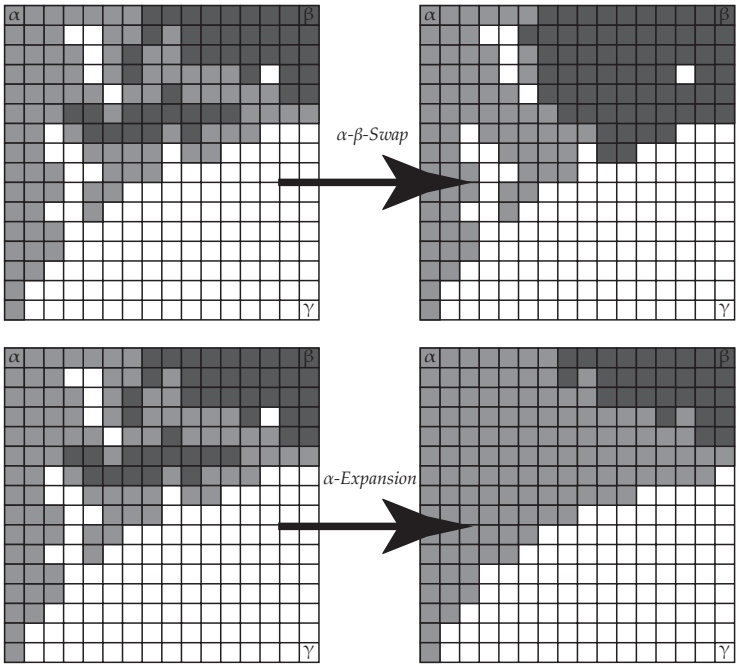


Abbildung 22: (oben) α-β-*Swap* bei dem Bereiche eines Labels mit Bereichen des anderen Labels getauscht werden, (unten) α-*Expansion* bei dem ein Label in die anderen Label erweitert wird

alle Tiefenstufen und führt in jedem Schritt eine α-Entfaltung durch, um eine neue Konfiguration mit geringeren Kosten zu finden. Dabei wird ein Label α von einer Partition P in eine neue Partition P' expandiert. Das Verfahren kann mehrmals hintereinander durchlaufen werden und bricht nach einem erfolglosen Durchlauf ab. Das errechnete lokale Minimum nähert sich dem globalen Minimum an [BVZ01].

Der aktuelle Graph $G_\alpha = (V_\alpha, E_\alpha)$ ist durch die aktuelle Partition P und das Label α definiert. Wie beschrieben besteht der Graph aus den sichtbaren Vertices und den zusätzlichen Terminalknoten $V = \{v_1, \ldots, v_n, s, t\}$. Laut Definition von Kolmogorov ist eine Funktion E von n binären Variablen graphenrepräsentierbar, wenn es einen Graphen $G = (V, E)$ gibt mit den Terminalknoten s und t und einer Knotenmenge $V_0 = \{v_1, \ldots, v_k\} \subset V - \{s, t\}$, so dass für jede Konfiguration $x_1, \ldots, x_n$ die Kosten $E(x_1, \ldots, x_n)$ gleich einer Konstanten und den Kosten c des

minimalen s-t-Schnittes unter allen $C = S, T$ ist, bei dem $v_i \in S$, wenn $x_i = 0$ und $v_i \in T$, wenn $x_i = 1$ $(1 \leq i \leq n)$. Die Funktion E ist exakt repräsentierbar in G von V_0, wenn die Konstante 0 ist [Kol04, KZ02b]. Der Schnitt S, T kann dabei in binären Variablen $x_1, \ldots, x_n$ repräsentiert werden mit $x_i = 0$, wenn $v_i \in S$ und $x_i = 1$, wenn $v_i \in T$. Die Energiefunktion E von n binären Variablen ist dann $E(x_1, \ldots, x_n)$ gleich der Konfiguration von $x_1, \ldots, x_n$ mit $x_1 = \{0, 1\}$.

Wenn eine Kostenfunktion E wie beschrieben graphenrepräsentierbar ist, kann ein gutes Minimum von E in polynomialer Zeit mit einem s-t-Schnitt ermittelt werden [Kol04]. Die Kostenfunktion E kann mit binären Variablen dargestellt werden:

$$E(x_1, \ldots, x_n) = \sum_i E^i(x_i) + \sum_{i<j} E^{i,j}(x_i, x_j)$$

Sie ist graphenrepräsentierbar, wenn

$$E^{i,j}(0,0) + E^{i,j}(1,1) \leq E^{i,j}(0,1) + E^{i,j}(1,0)$$

gilt. Dann handelt es sich um eine reguläre Funktion [KZ02b].

Der Graph G wird für jeden Term separat angelegt, um dann alle Teilgraphen zusammenzufügen. Es gibt $n + 2$ Knoten $V = \{v_1, \ldots, v_n, s, t\}$, wobei die nichtterminalen Knoten v_i die binären Variablen x_i repräsentieren. Jeder Term von E bekommt eine oder mehrere Kanten im Graphen zugeordnet, die die Knoten miteinander verbinden [Kol04]. .

4.4.6 Maximaler Fluss in einem Graphen

Algorithmen, die das Minimaler-Schnitt-Maximaler-Fluss Problem (Max-Flow-Min-Cut-Theorem) lösen, basieren entweder auf der *push-relabel* Methode oder auf dem Verfahren von Ford und Fulkerson [FF56, FF62, Din70, Kol04]. Bei den augmentierenden Pfad Verfahren nach Ford und Fulkerson, die meistens auf der Verbesserung durch Dinics Algorithmus basieren, wird der Fluss über nicht gesättigte Pfade von der Quelle zur Senke erzeugt bis ein maximaler Fluss im Graphen G erreicht ist [Din70, Kol04]. Um den kürzesten Pfad von s zu t im Residualgraphen zu finden wird ein Breitensuchverfahren benutzt [Din70].

Kolmogorov erweitert das Verfahren, in dem zwei Suchbäume, von der Quelle und der Senke aus, den augmentierenden Pfad ermitteln und aufeinander zuwachsen [Kol04]. Die Suchbäume werden dabei wiederverwendet und nicht nach jedem Schritt neu aufgebaut.

Im Suchbaum S, der von der Quelle s ausgeht, sind alle Kanten, die von den Elternknoten zu den Kindknoten verlaufen, nicht gesättigt, so wie auch alle Kanten des Suchbaums T von den Kindknoten zu den Elternknoten nicht gesättigt sind. Die Knoten können entweder den Status *aktiv* oder *passiv* besitzen, wobei die aktiven Knoten den äußeren Rand repräsentieren. Kolmogorov schlägt vor, dass aktive Knoten den Suchbaum erweitern und nicht gesättigte Kanten diesem freie Kindknoten zuordnen. Ein augmentierender Pfad ist gefunden, wenn ein aktiver Knoten eines Suchbaums auf einen Nachbarknoten des anderen Suchbaums trifft.

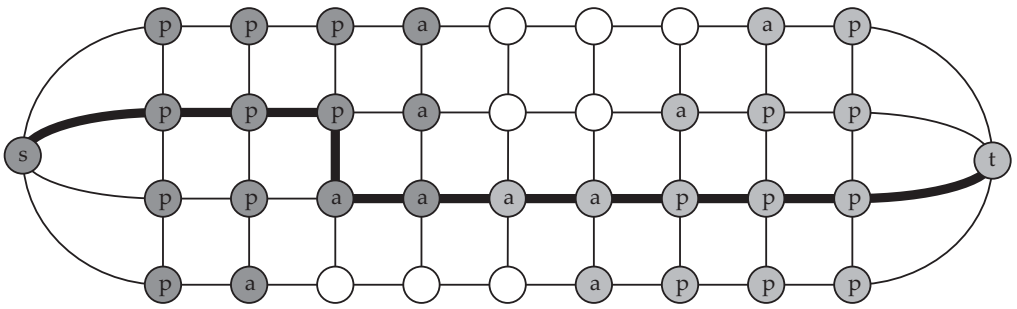


Abbildung 23: Suchbäume S und T mit aktiven und passiven Knoten nach der *Wachstumphase* mit einem Pfad von s nach t

Der Algorithmus von Kolmogorov wiederholt iterativ drei Schritte, bis die Abbruchbedingung erreicht wird. Dabei werden die *Wachstumsphase*, die *Vermehrungsphase* und die *Adoptionsphase* durchlaufen.

In der ersten Phase werden die Suchbäume S und T von beiden Seiten erweitert, bis sie sich treffen und somit einen Pfad von s nach t aufspannen (Abbildung 23). Über die aktiven Knoten werden die ungesättigten Kanten untersucht, um diese mit den freien Kindknoten zu verbinden und den Baum zu erweitern. Die neuen Knoten werden aktiviert und sobald alle Nachbarkanten untersucht wurden, wird der Status des aktu-

ellen Knotens auf passiv gesetzt. Die *Wachstumsphase* ist abgeschlossen, sobald die Suchbäume aufeinander treffen.

In der zweiten Phase, der *Vermehrungsphase*, wird der augmentierende Pfad erweitert. Einige Pfade des Residualgraphen sind nach der ersten Phase gesättigt, wodurch Waisenknoten entstehen, bei denen die Kanten mit den Elternknoten gesättigt sind. In dieser Phase können die Suchbäume in Wälder zerfallen, wobei die Waisenknoten zu neuen Wurzelknoten werden.

Die *Adoptionsphase* stellt die durchgehende Suchbaumstruktur zwischen S und T wieder her. Hier soll für jeden Waisenknoten ein gültiger Elternknoten gefunden werden, der zum selben Suchbaum gehört und über eine nicht gesättigt Kante verbunden ist. Sollte kein gültiger Knoten gefunden werden, so wird der Waisenknoten aus dem Suchbaum entfernt und als frei deklariert während die Kindknoten als Waisenknoten markiert werden. Diese Phase ist beendet, wenn keine Waisenknoten mehr vorhanden sind und der komplette Suchbaum wieder hergestellt ist.

Alle Schritte werden solange durchlaufen, bis keine aktiven Knoten mehr vorhanden sind. Dann ist der maximale Fluss und somit ein minimaler Schnitt gefunden.

5 Auswertung

Ziel der Arbeit ist es, ein möglichst präzises Modell eines Gebäudes aus Bilddaten dreidimensional zu rekonstruieren. Dafür wurden zwei Verfahren implementiert, die in den folgenden Abschnitten miteinander verglichen werden. Eine experimentelle Auswertung der Verfahren soll die Qualität und die Möglichkeiten der beiden Methoden wiedergeben. Dafür werden verschiedene Aspekte der Algorithmen betrachtet und gegenübergestellt. Beide Verfahren werden mit unterschiedlichen Parametern ausgeführt, um die bestmöglichen Einstellungen für jedes Verfahren zu finden.

Neben der Qualität der Ergebnisse ist auch die Laufzeit und der Speicherverbrauch für einen guten Algorithmus entscheidend. Deshalb werden auch diese Werte empirisch erfasst und geprüft.

5.1 Ergebnisse des Sichtbarkeitstests

Für die Optimierung müssen, wie in Abschnitt 4.1 beschrieben, die sichtbaren Polygone und Vertices herausgefiltert werden. Dafür wird das sogenannte *Hidden Surface Removal* Verfahren verwendet, welches als *Rasterizer* über die *OpenGL Rendering Pipeline* implementiert wurde, und ein *Raytracer* der neu programmiert wurde. Beide Verfahren wurden mit den vorgestellten Modellen aus Abschnitt 3 getestet.

Abbildung 24: Referenzansichten der Gebäudefassaden für den Sichtbarkeitstest

Das nach dem Verfahren aus Absatz 2.5 erzeugte Hausmodell hat 2397960 Polygone und 1198994 Vertices. In der Referenzansicht 55 filtert das einfache Verfahren das Modell auf 343513 Polygone, die optimiert

werden sollen. Wie beschrieben werden viele Polygone nicht herausgefiltert, weil sich einige Polygone gegenseitig überdecken, obwohl sie in der eigentlich sichtbaren Fassade liegen. Um dies zu umgehen, wurden wie in Abschnitt 4.1 beschrieben, verschiedene Methoden getestet. Allerdings brachte, nach mehreren Tests, nur die Erhöhung der Auflösung der Rendertextur Verbesserungen. So konnte im Bankenmodell fast eine Verdoppelung der sichtbaren Polygone erreicht werden.

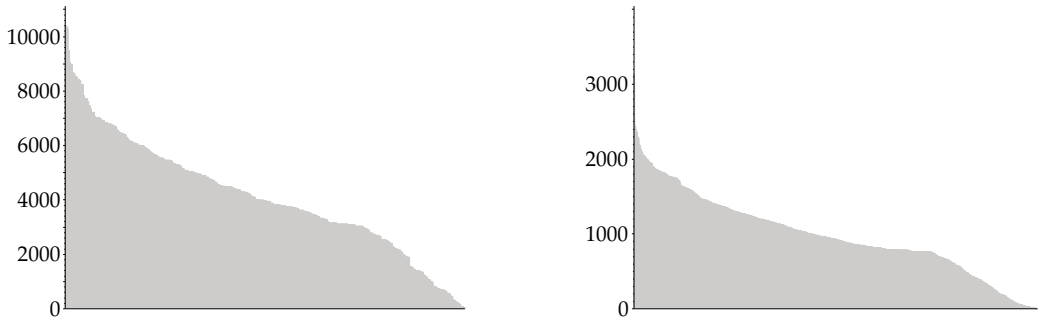
Der *Raytracer*, der die Probleme der Grafikkartentreiber umgehen soll, wurde wegen der schlechten Laufzeit nur für das Hausmodell getestet. Zur Beschleunigung wurde die in Absatz 4.1.1 besprochene Partitionierung verwendet. Die Qualität der Ergebnisse entspricht dem Test mit dem *Rasterizer*. Die Versuche mit verschiedenen Rastergrößen zeigen, dass die Verteilungen ähnlich ausfallen (Abbildung 25). Einige wenige Zellen haben besonders viele oder besonders wenige bis keine Polygone. Der mittlere Bereich der Verteilung fällt jedoch ähnlich aus.

Das *Burger King* Modell hat 5561728 Polygone und 2781066 Vertices. Als Referenzansicht für die Rekonstruktion der Fassade wurde die Position 0 gewählt. Die Rendertextur wird bei den Tests von 5184×3456 Bildpunkten bis auf 82944×55296 vergrößert.

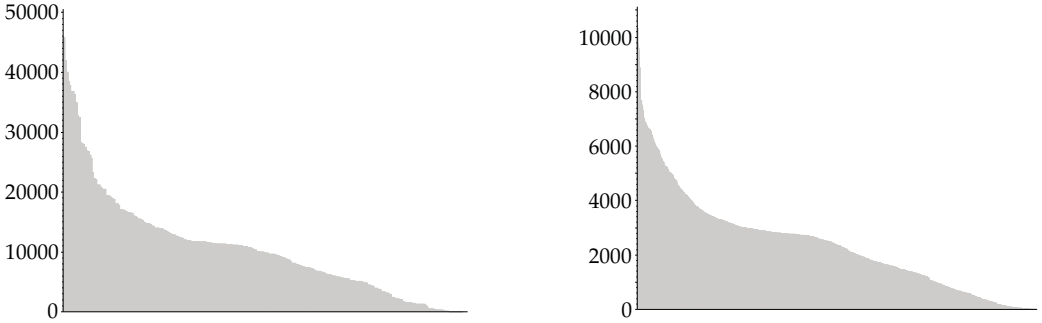
Das Bankenmodell besteht aus 4801604 Polygone und 2404266 Vertices. Die Referenzansicht für den Test ist die Position 37.

Zu erkennen ist, dass durch die Vergrößerung der Textur mehr Polygone abgebildet werden können und man dadurch eine geschlossenere Polygonfassade bekommt. Allerdings steigt dadurch auch die Laufzeit, da eine größere Textur ausgelesen werden muss. Die ideale Texturgröße ist, wie aus den Tests zu erkennen ist, abhängig von dem Modellcharakter und muss deshalb vom Benutzer eingestellt werden.

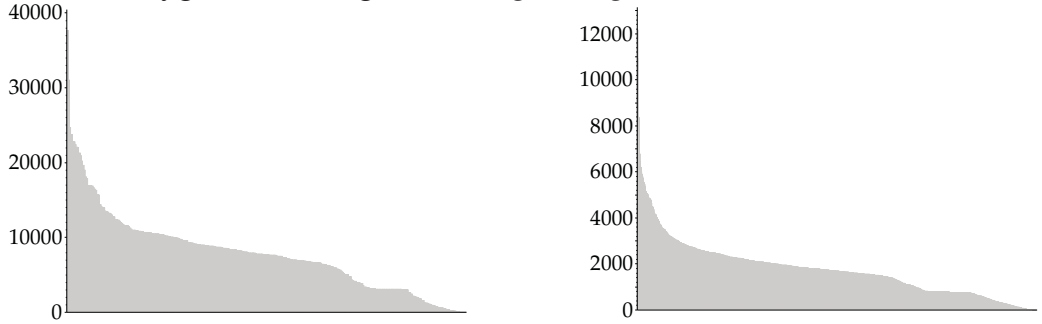
Trotz der Vergrößerung kann jedoch keine vollkommen dichte, geschlossene Polygonfassade erzeugt werden, da viele Polygone einfach aus bestimmten Winkel nicht sichtbar sind. Hier kann aktuell nur ein manueller Eingriff helfen, da Versuche die offenen Polygonbereiche automatisch zu schließen, an den Rändern der offenen Fassade scheitern. Für viele Ansichten reicht jedoch auch der einfache Sichtbarkeitstest mit dem lückenhaften Polygonmodell.



Polygonverteilung beim Haumsodell in Position 55



Polygonverteilung beim *Burger King* Modell in Position 0



Polygonverteilung beim Bankenmodell in Position 37

Abbildung 25: Histogramm der Polygonverteilung beim Sichtbarkeitstest für die Gebäudemodelle mit $8 \times 8 \times 8$ Zellen (links) und $16 \times 16 \times 16$ Zellen (rechts)

Größenfaktor	Polygone	Laufzeit (s)
1	343513	2,7
4	396280	30,8
8	402449	116,3
16	406056	456,9
32	408223	1782,6

Tabelle 1: Sichtbarkeitstest für das Hausmodell in Ansicht 55

Größenfaktor	Polygone	Laufzeit (s)
1	644594	2,7
4	823174	38,3
8	857600	148,1
16	881697	582,2
32	910744	2320,7

Tabelle 2: Sichtbarkeitstest für das *Burger King* Modell in Ansicht 0

Größenfaktor	Polygone	Laufzeit (s)
1	1243169	3,5
4	1998441	50,5
8	2168700	175,0
16	2208550	676,5
32	2222595	2645,4

Tabelle 3: Sichtbarkeitstest für das Bankmodell in Ansicht 37

5.2 Verdeckungstest

Wie in Abschnitt 4.2 beschrieben, wird für die Verdeckungsanalyse auf diskrete Gleichverteilung für die linke und rechte Seite einer Referenzansicht getestet. Wird ein vorgegebener Schwellenwert überschritten, liegt wahrscheinlich eine Verdeckung vor und die schlechteren Werte können verworfen werden. Der Schwellenwert muss so gewählt werden, dass nicht zu viele Bereiche aussortiert werden (Abbildung 26).

Bei dem Hausmodell ist eine starke Ausprägung der linksseitigen Fehler zu erkennen, die in den anderen Modellen so nicht vorkommt. Mit fallendem Schwellenwert sinkt die Anzahl der Fehler linear ab. Weitere Zusammenhänge manifestieren sich in der Verteilung nicht. In der Referenzansicht 0 wurden insgesamt 4382966 Vergleichstests für die erste Tiefenstufe mit 3, 5 und 7 Vor- und Rückschaubildern ausgeführt. Die Anzahl der Vor- und Rückschauansichten hat direkten Einfluss auf die Qualität der Ergebnisse. Je mehr Bilder verwendet werden, desto besser wird das Resultat und desto weniger Gleichverteilungsfehler treten auf. Allerdings sind hier natürliche Schranken durch den Winkel zur Referenzansicht gegeben.

5.3 Auswertung des Simulated Annealing Verfahrens

5.3.1 Parameter der Kostenfunktion

Die Verteilung der Farbfehler in den einzelnen Tiefenstufen kann einen Aufschluss darüber geben, bis zu welcher Tiefe eine Verschiebung der Vertices sinnvoll ist. Wie man in den Histogrammen in Abbildung 31 erkennen kann, steigt mit zunehmender Distanz zum Ausgangsmesh tendenziell auch der Anteil der höheren Fehlerabweichungen D_{SSD} ab einer bestimmten Tiefe. Parallel dazu sinkt der Anteil der geringen Farbfehler. Die Fehler sind auf der horizontalen Achse abgebildet und der Prozentsatz der Fehler wird in der vertikalen Achse gemessen. Kann man den Übergang der ansteigenden Fehlerabweichungen finden, so lässt sich die zu erwartende maximale Tiefe genau festlegen. Wird dies nicht beachtet, werden viele unnötige Tiefenstufen durchlaufen, wobei weder eine

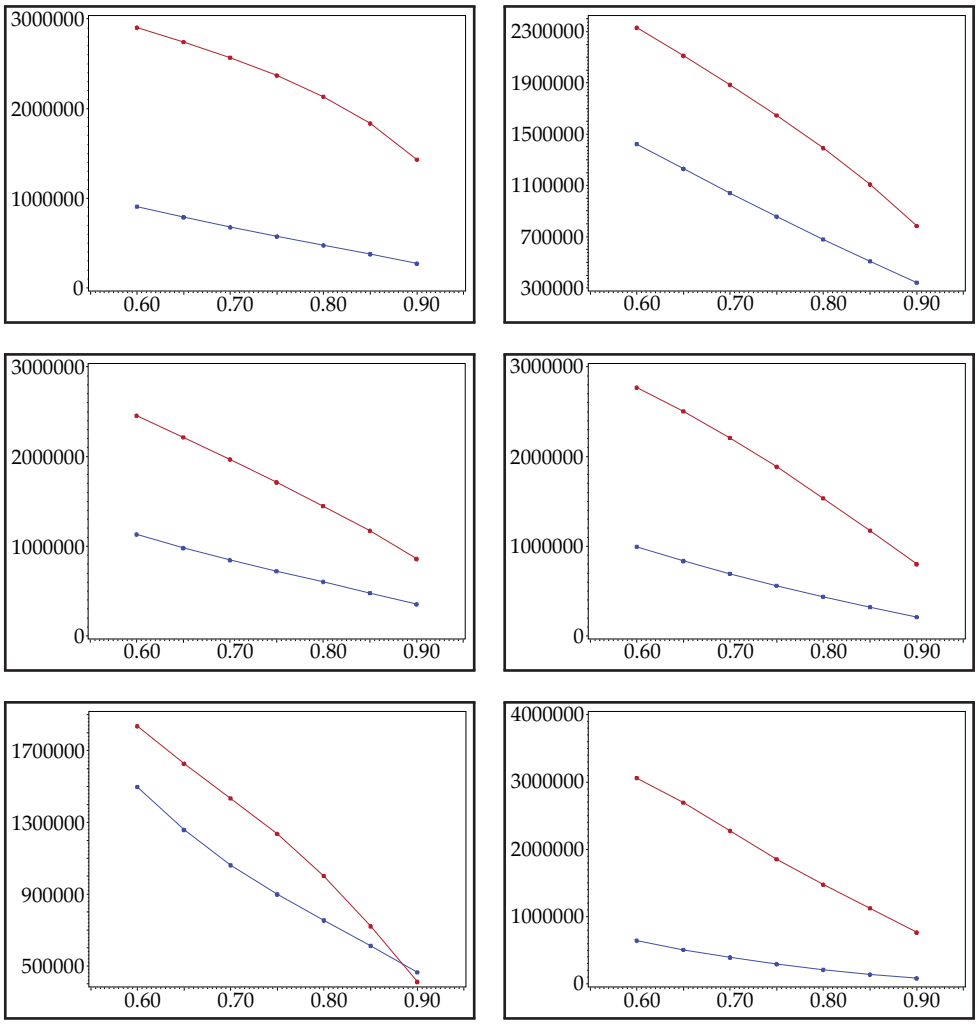


Abbildung 26: Verteilung der Verdeckungsfehler beim Hausmodell für die Referenzansicht 0 (links) und 17 (rechts) mit linksseitigen Fehlern (rot) und rechtsseitigen Fehlern (blau) und 3 (oben), 5 (mitte) und 7 (unten) Vor- und Rückschaubildern; die horizontale Achse gibt den Prozentsatz der Gleichverteilung wieder ab dem ein Fehler erkannt wird und die vertikale Achse gibt die Anzahl der Gesamtfehler an

erkennbare Verbesserung des Modells erzielt wird, noch kann eine ausreichende Tiefe für alle Vertices erreicht werden, welche zur Verbesserung des Modell erforderlich ist.

Die Gewichtungsparameter der Kostenfunktion sind bei verschieden-

artigen Modellen unterschiedlich einzustellen. Da hier nur Gebäudemodelle mit ähnlichen Eigenschaften behandelt werden, kann neben der experimentellen Einstellung auch eine einmalige fundierte Analyse der Parameter ausgeführt werden. Dafür werden die einzelnen Parameter schrittweise in die Kostenfunktion aufgenommen, um die Auswirkungen der einzelnen Werte auf das Ergebnis zu messen. Alle Fehlerwerte der Kostenfunktion werden auf das Intervall $[0, 1]$ abgebildet, um sie zu gewichten und dann aufzusummieren.

Legt man die Farbkorrespondenzkosten fest, so kann nun die Auswirkung des Glattheitsterms E_{S1} auf die Gesamtkosten gemessen werden. Die Farbkorrespondenz ist der wichtigste Term der Funktion und wird deshalb sehr hoch gewichtet. In den Versuchen wurde dafür ein Wert zwischen 50 und 100 gewählt.

Wie zu erwarten war, steigt mit zunehmender Gewichtung von E_{S1} auch die Ebenheit der Oberfläche. Betrachtet man die Abbildungen 27, so kann hier der Einfluss des Parameters gut erkannt werden. Je stärker rot der Bereich ist, umso glatter ist die Oberfläche. Bei zunehmender Unebenheit geht der Farbwert in Richtung Blau. Setzt man den Wert zu hoch an, so wird keine gute Rekonstruktion möglich, da der Glattheitsterm die Verschiebung der Vertices für eine ungünstige Farbkorrespondenz blockiert. Ist der Wert zu niedrig, wird die Oberfläche uneben und das Rauschen der Farbkorrespondenzanalyse überwiegt.

Der zweite Glattheitsterm E_{S2} hat stärkeren Einfluss auf die Oberflächenebenheit, da er einen weiteren Bereich abdeckt. In Abbildung 28 ist gut zu erkennen, wie die Qualität der Oberfläche mit einer höheren Gewichtung des Parameters zunimmt. Auch hier muss der Wert so angepasst werden, dass die Glattheit nicht die Verschiebung der Vertices verhindert. Es ist wichtig eine gute Kombination zwischen E_{S1} und E_{S2} zu finden, um die großen Flächen der Gebäudemodelle zu erhalten.

Der Qualitätsparameter für die Polygone E_Q hat Einfluss auf die Ausgeglichenheit der Polygone. Durch eine passende Gewichtung können degenerierte Polygone verhindert werden, die eventuell aus Fehlern der Farbkorrespondenz entstehen. Dieser Wert kann allerdings nicht besonders hoch eingestellt werden, da an Flächenübergängen langgezogene

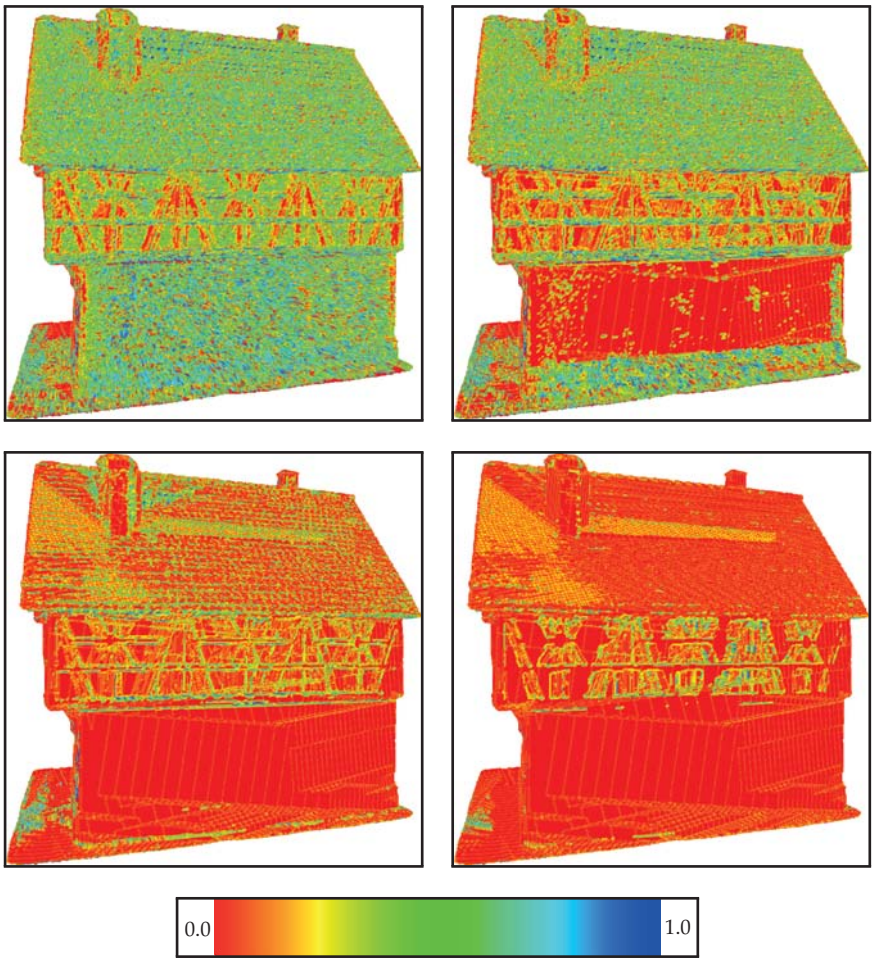


Abbildung 27: Visualisierung der Modellebenheit für verschiedene Gewichtungen des Glattheitsterms E_{S1} mit $\lambda_{S1} = \{0.0, 0.1, 0.7, 3.0\}$

Polygone durchaus erwünscht sind, um eine Kante zu bilden. In den Versuchen haben Werte zwischen 0.1 und 0.5 bessere Ergebnisse bei der Rekonstruktion gebracht und viele Ausreißer wurden dadurch blockiert (Abbildung 30).

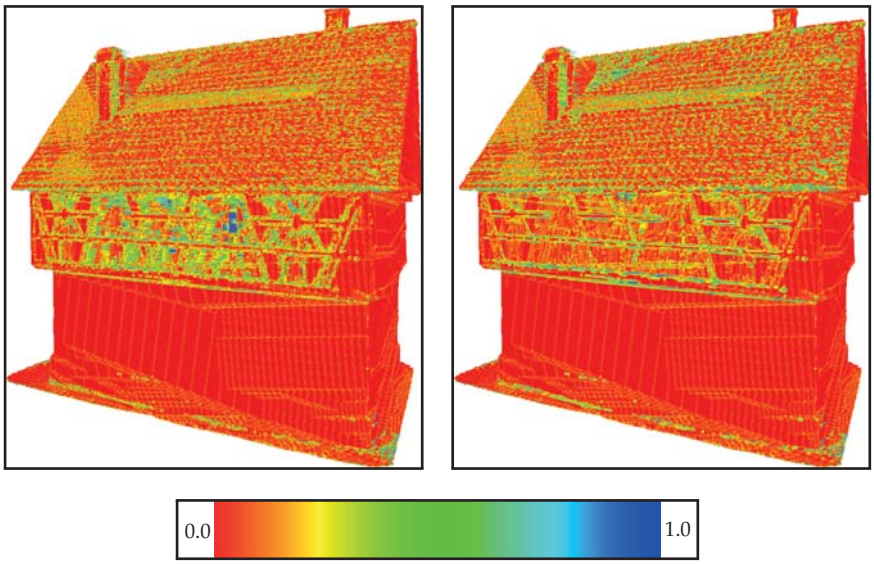


Abbildung 28: Visualisierung der Modellebenheit für verschiedene Gewichtungen des Glattheitsterms E_{S2} mit $\lambda_{S2} = \{0.1, 1.0\}$ und konstantem $\lambda_{S1} = 0.7$

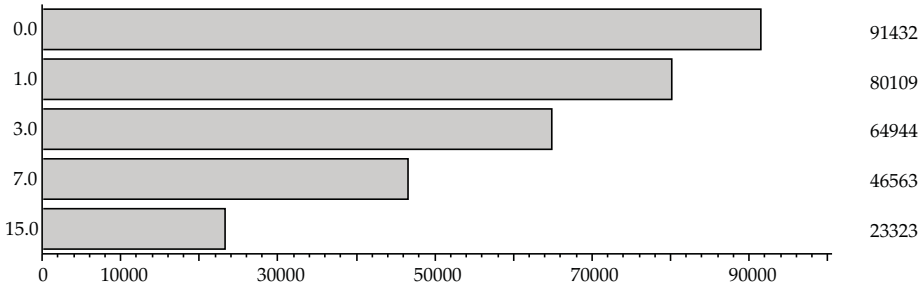


Abbildung 29: Die vertikale Achse gibt den gewichteten Tiefenterm $\lambda_D E_D$ und die horizontale Achse gibt die Anzahl der verschobenen Vertices in Position 55 des Hausmodells mit $\lambda_{S1} = 0.7$, $\lambda_{S2} = 0.5$ und $\lambda_Q = 0.1$ wieder.

Um eine zu starke Schrumpfung des Objektes zu verhindern, kann der Tiefenterm E_D höher gewichtet werden. Gerade wenn viele Ausreißer auf einer Ebene vorhanden sind, kann es in diesem Bereich zu einer Schrumpfung führen. Um dem entgegenzuwirken kann der Tiefenparameter λ_D erhöht werden. Wird der Wert jedoch zu hoch angesetzt, dann werden kaum noch Vertices verschoben (Abbildung 29).

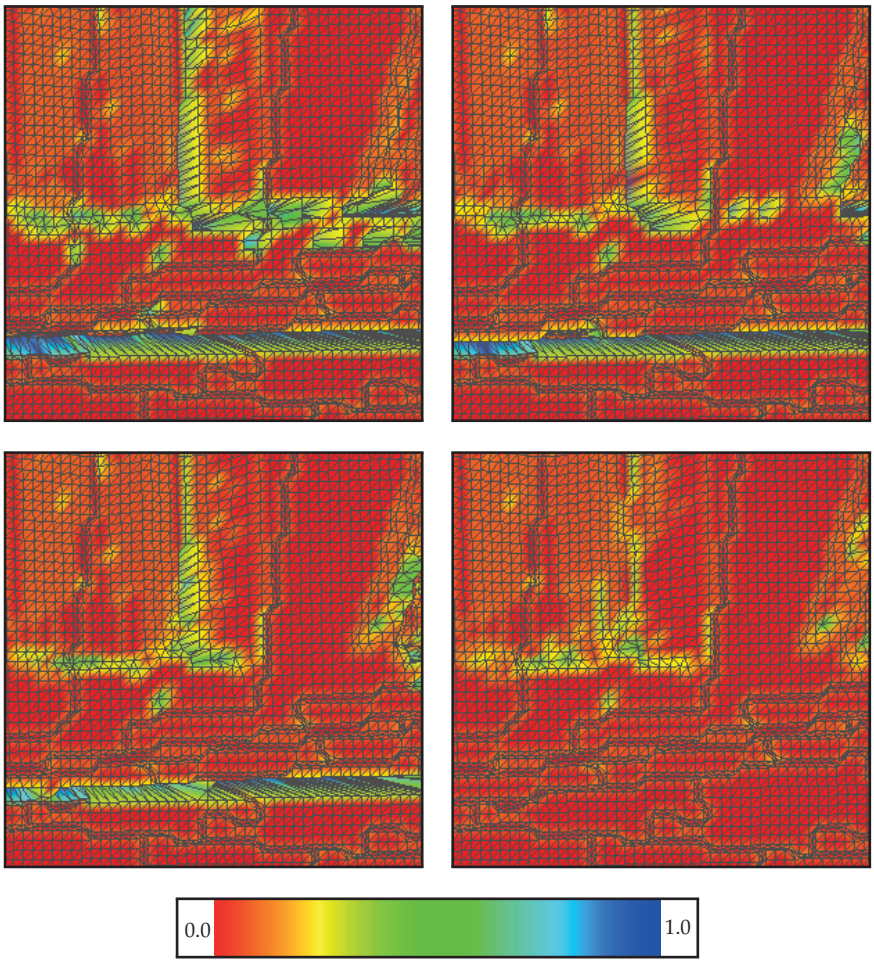


Abbildung 30: Visualisierung der Polygonqualität bei verschiedener Gewichtungen des Qualitätsterms E_Q mit $\lambda_Q = \{0.1, 0.5, 1.0, 3.0\}$ und konstantem $\lambda_{S1} = 0.7$ und $\lambda_{S2} = 0.5$

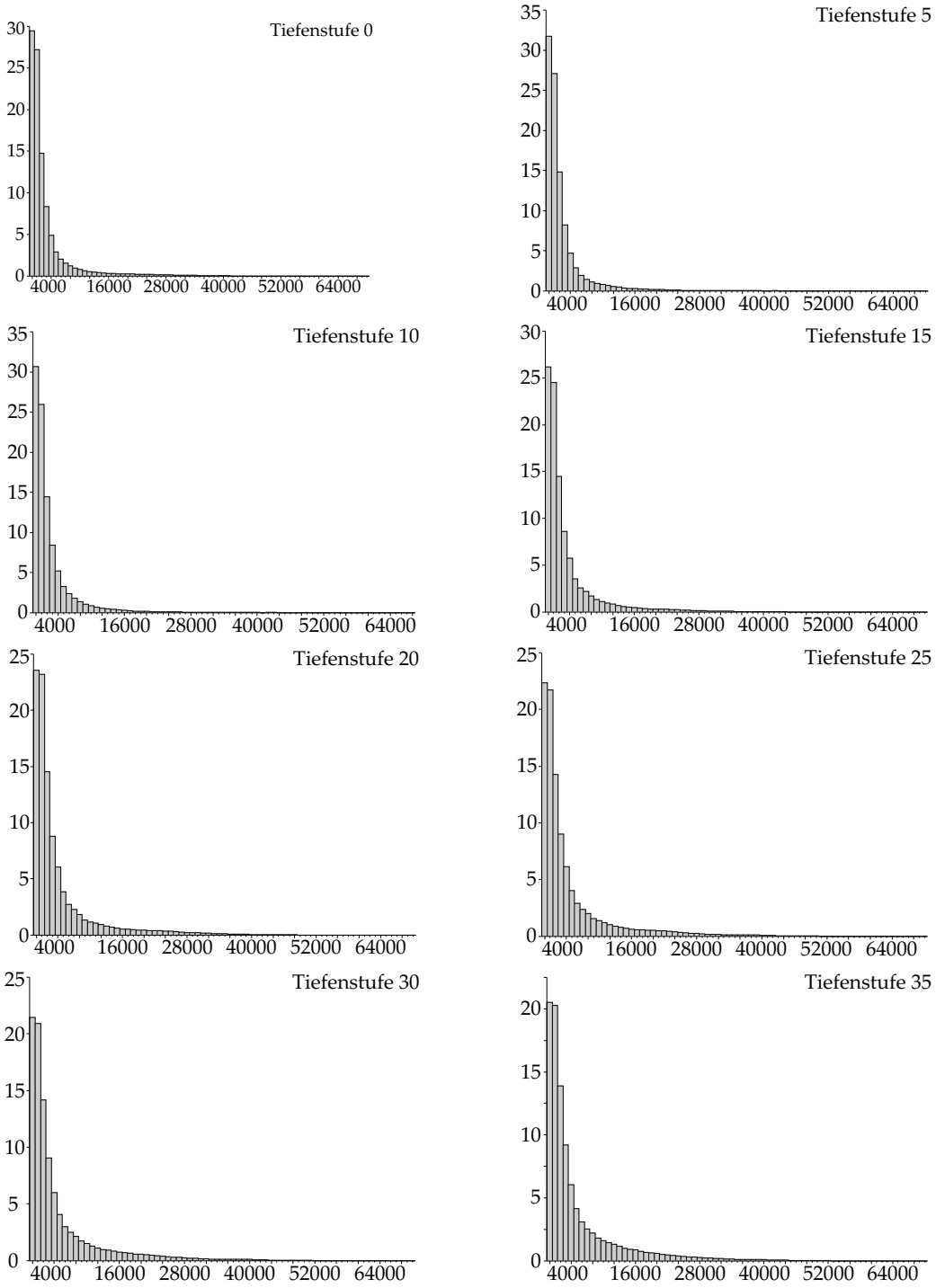


Abbildung 31: Verteilung der Fehlerabweichungen des Hausmodells in der Referenzansicht 55 für verschiedene Tiefenstufen

5.3.2 Analyse des Algorithmus

Das iterative Verfahren verschiebt schrittweise alle sichtbaren Vertices, bis eine optimale Position für jedes Objekt gefunden wurde. Besonders aufwendig ist dabei die Berechnung der Kostenfunktion, die für jede Position und jeden Vertex berechnet werden muss. Berechnet man zusätzlich noch die Positionen der Nachbarvertices ein, wie es in Abschnitt 4.3 besprochen wurde, so kommt man schnell in Berechnungsumfänge, die aktuelle Rechnerkapazitäten weit überschreiten. Deshalb muss die Anzahl der Fehlerberechnungen gesenkt werden.

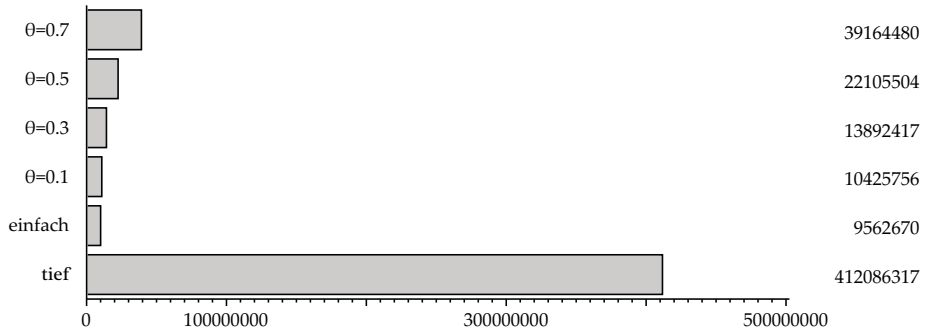


Abbildung 32: Anzahl der Fehlerberechnungen mit verschiedenen Startwerten θ beim *Simulated Annealing* und bei zwei einfachen Verfahren für das Hausmodell

In der Abbildung 32 werden drei Verfahren nach der Abfragehäufigkeit nebeneinander gestellt. Bei dem einfachen Verfahren wird die Kostenfunktion für jede Stufe und alle Vertices nur einmal berechnet. Dabei werden, wie besprochen häufig lokale Minima erreicht und nicht mehr verlassen. Um dies zu umgehen, wird das Verfahren erweitert und jeder Nachbarvertex des aktuellen Vertex wird stufenweise an die aktuelle Position herangeführt um zu testen ob sich der Fehler allgemein verbessert. Das *Simulated Annealing* Verfahren wird mit verschiedenen Startwerten θ ausgeführt. Dabei zeichnet sich deutlich ab, wie hoch das Potential der Einsparung gegenüber der Exhaustionsmethode ist.

Vergleicht man verschiedene Startwerte des *Simulated Annealing*, so sinkt erwartungsgemäß bei fallender Schwelle auch die Anzahl der Fehlertests (Abbildung 32). Ein sinnvoller Wert kann einfach über die zur

Verfügung stehende Rechnerkapazität aus der Tabelle und dem Berechnungsaufwand der Kostenfunktion abgeleitet werden.

5.4 Auswertung Graph Cut

5.4.1 Parameter der Kostenfunktion

Im Gegensatz zur lokalen Optimierung mit dem *Simulated Annealing* Verfahren ist die Kostenfunktion des *Graph Cut* Verfahrens einfach aufgebaut. Hier sind lediglich zwei Terme miteinander verknüpft, so dass die Gewichtungseinstellung einfacher zu handhaben ist. Die Farbkorrespondenzkosten $E_C(v)$ und der Glattheitsterm $E_S(v, v_n)$ werden für jede Tiefenstufe global aufsummiert, um die Oberfläche zu optimieren.

Die Farbkorrespondenz wird dabei in einer Tiefenkarte für jede Tiefenstufe gespeichert, bevor sie für jeden sichtbaren Vertex akkumuliert wird. Betrachtet man die Tiefenkarten der einzelnen Stufen (Abbildung 33), so sieht man wie stark das Rauschen bei den Tiefeninformationen aus der Kongruenzanalyse ist. Deshalb werden die Tiefenfehler in einem Ring um den projizierten Vertex aufsummiert (Abbildung 34). Dafür wird der *Gauß*-Filter mit einer an die Dimensionen des Bildes angepassten Breite von 3 bis 5 Bildpunkten verwendet. Dies entspricht einem einfachen und zweifachen Nachbarschaftsring um einen Bildpunkt. Größere Filterbereiche führen zu vielen Überschneidungen der Regionen und glätten dadurch vorhandene Kanten ungewollt ab. In Abbildung 35 ist für vier Referenzansichten der Fassaden des Hausmodells die mehrfache Überlagerung von Tiefeninformationen zusammengefasst mit verschiedenen Filterbreiten aufgeschlüsselt worden. Mit steigender Boxgröße erhöht sich erwartungsgemäß auch die Anzahl der Überlagerungen (Abbildung 35).

Für den Glattheitsterm $E_S(v, v_n)$ wurden verschiedene Verfahren implementiert und miteinander verglichen. Das einfachste Verfahren wertet die Tiefenstufe eines Vertex aus und vergleicht diese mit der Tiefenstufe des Nachbarvertex (Abbildung 36). Je stärker rot der Bereich ist, umso glatter ist die Oberfläche des Modells, bei zunehmender Unebenheit wandert der Farbwert in Richtung blau. Diese Werte werden wie in Abschnitt 5.3 über die gleiche Glattheitsfunktion auf einem Intervall zwischn

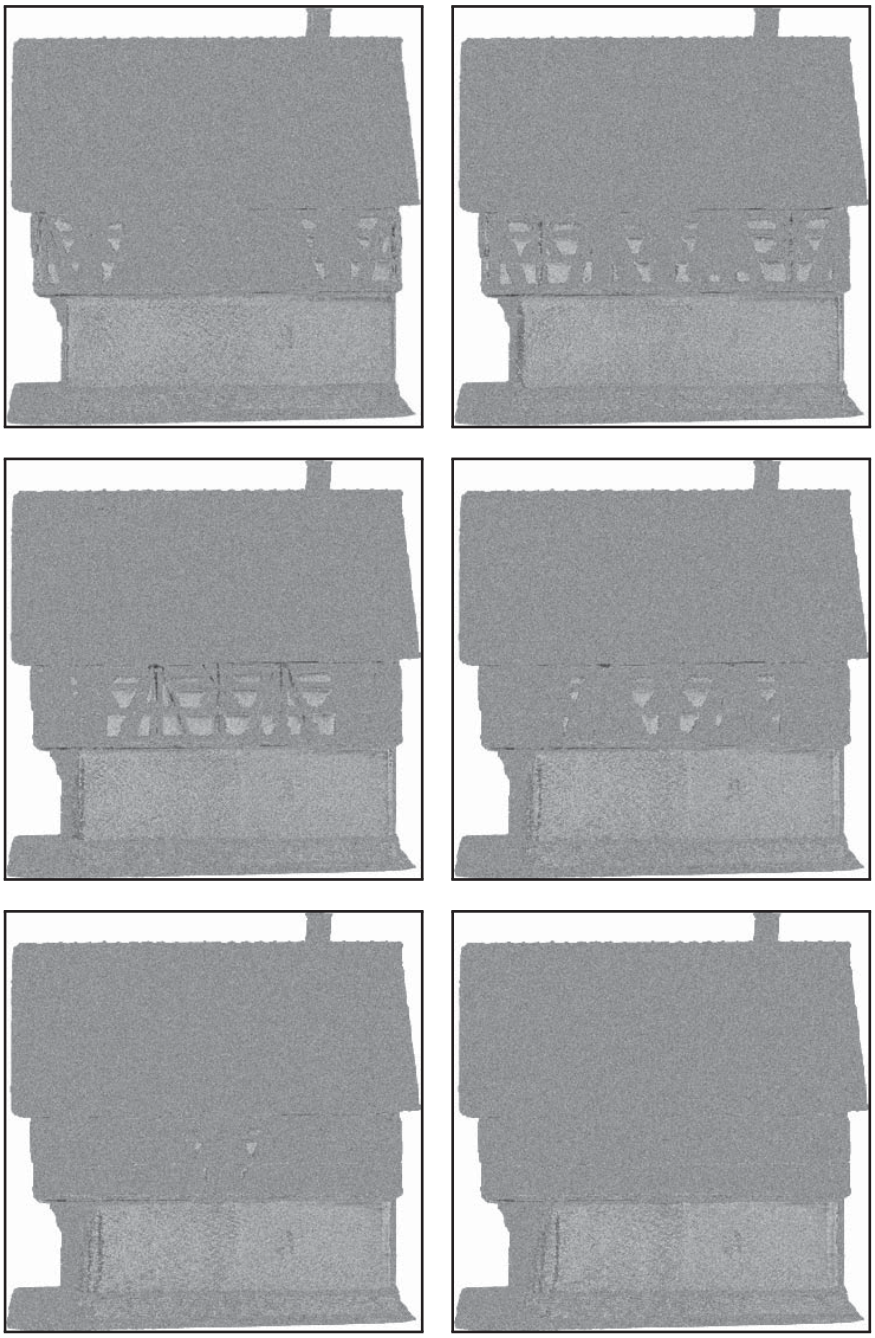


Abbildung 33: Tiefenfehler der Farbkorrespondenzanalyse D_{SSD} für die Tiefenstufen $t = \{0, 5, 10, 15, 20, 25\}$ für das Hausmodell in der Referenzansicht 0

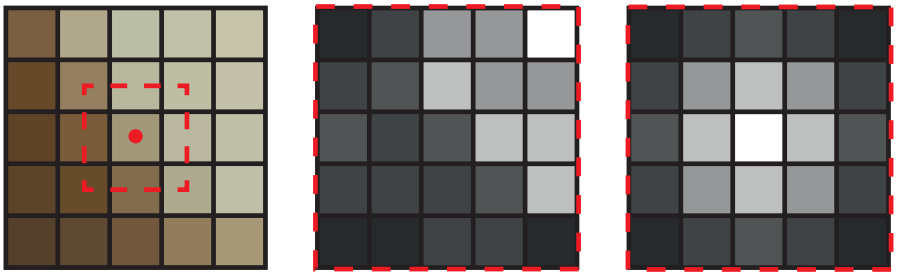


Abbildung 34: (links) Bilddaten mit ausgewähltem Pixel, auf den ein Vertex projiziert wird; (mitte) errechnete Tiefenkarte; (rechts) 5×5 *Gauß*-Filter, der auf die Tiefenkarte angewendet wird und die Tiefe für den Vertex des Bildpunktes bestimmt

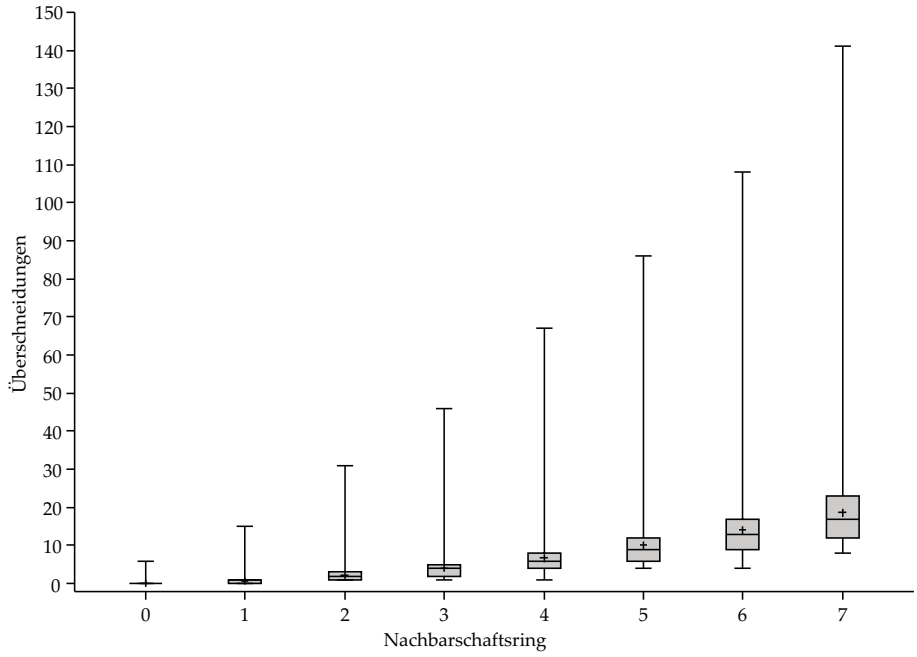


Abbildung 35: Filtergröße und Überschneidungen der Bildpunkte für das Hausmodell

0 und 1 gemessen, um miteinander vergleichbare Ergebnisse zu bekommen. Mit einer höheren Gewichtung λ_S steigt auch der Anteil der ebenen Bereiche, jedoch verschwinden dadurch auch Details des Objektes.

Um die tatsächliche Distanz der Vertices zu messen wird das Kame-

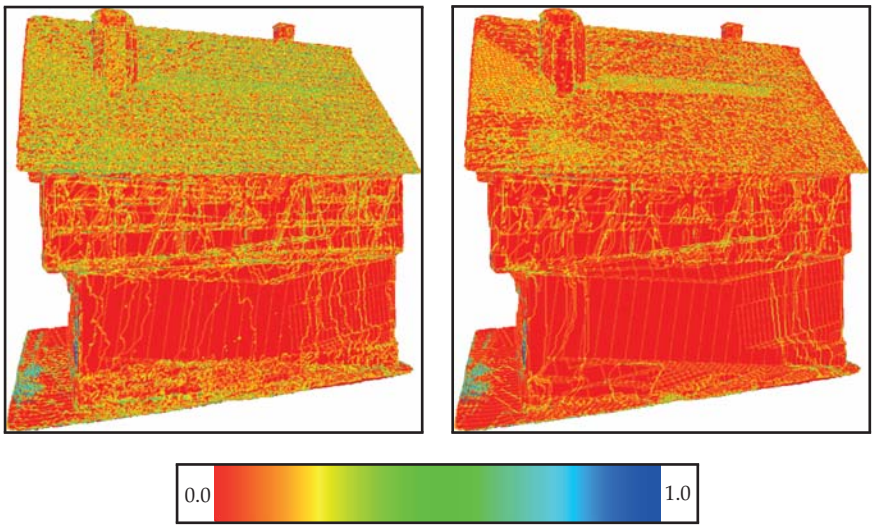


Abbildung 36: Einfluss des Gewichtungsfaktors $\lambda_S = \{10, 50\}$ auf die Objekt-Ebenheit

razentrum als Referenzpunkt benutzt. Durch diese Distanz werden vorhandene Modellstrukturen der Rekonstruktion berücksichtigt, um die Ergebnisse zu verbessern. Die Ergebnisse sind zwar vergleichbar mit den Werten aus dem vorherigen Verfahren, jedoch werden Details wie etwa das Fachwerk des Hauses besser modelliert (Abbildung 37). Trotz höherem Rechenaufwand bietet sich das Verfahren gerade bei mehrfacher Anwendung für verschiedene Referenzansichten an, da eine vorherige Verschiebung der Vertices berücksichtigt wird.

Berechnet man die Normalen der Vertices und nimmt die Winkeldifferenz von zwei Nachbarn als Maß der Ebenheit, so erhält man ein vergleichbares Verfahren zu den vorgestellten lokalen Energiekosten in Abschnitt 4.3.3. Die Gewichtung von λ_S muss hier jedoch feiner eingestellt werden. Vergleicht man dieses Verfahren mit den ersten beiden Verfahren, so ist eine deutliche Verschlechterung sichtbar, da durch Erhöhung der Gewichtung nahezu alle Details des Modells verloren gehen (Abbildung 38). Das Verfahren erreicht auch nicht die Qualität der lokalen Glattheitsfunktion, da immer nur ein Nachbarvertex in die Berechnung einfließt. Eine Erweiterung ist nicht möglich, da nur ein adjazenter Vertex für den

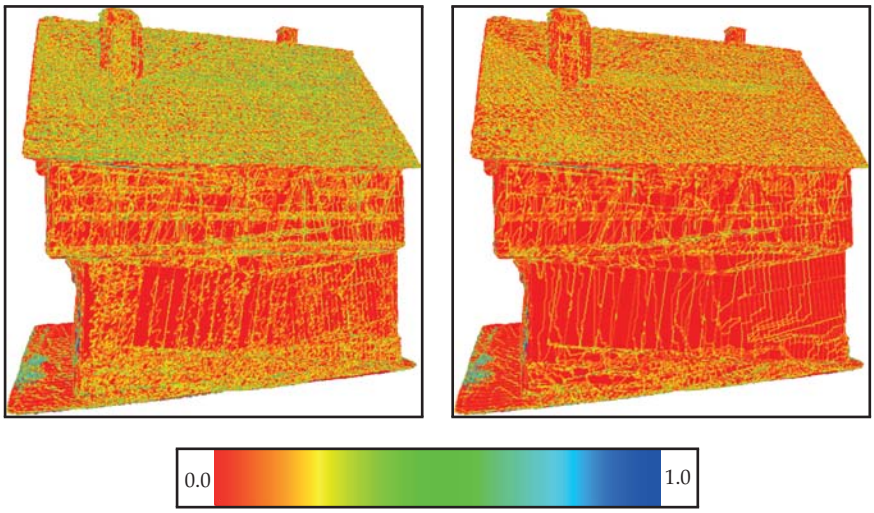


Abbildung 37: Einfluss des Gewichtungsfaktors $\lambda_S = \{10, 50\}$ auf die Objekt-Ebenheit mit Berücksichtigung der Tiefendistanz

Graphenaufbau vorgesehen ist.

Da höhere Winkeldifferenzen an einigen Stellen des Modells erwünscht sind, können die ersten beiden Verfahren erweitert werden, um weitere Bildinformationen zu berücksichtigen. Die Farbdifferenz, die bei glatten Flächen kleiner ausfallen sollte, wird als zusätzliche Gewichtung in die Berechnung einbezogen. Je geringer die Differenz zwischen den Farben unter den benachbarten projizierten Vertices ist, desto höher wird die Distanzabweichung gewichtet. Die einfache Methode nutzt eine Stufenfunktion, wie sie in Abschnitt 4.3.3 vorgestellt wurde. Da dabei aber drei Parameter gesetzt werden müssen und man diese für jedes Modell neu justieren müsste, wurde eine erweiterte *Cauchy*-Funktion entwickelt. So muss nur ein Gewichtungsfaktor eingestellt werden um ähnlich gute Ergebnisse zu erhalten (Abbildung 39).

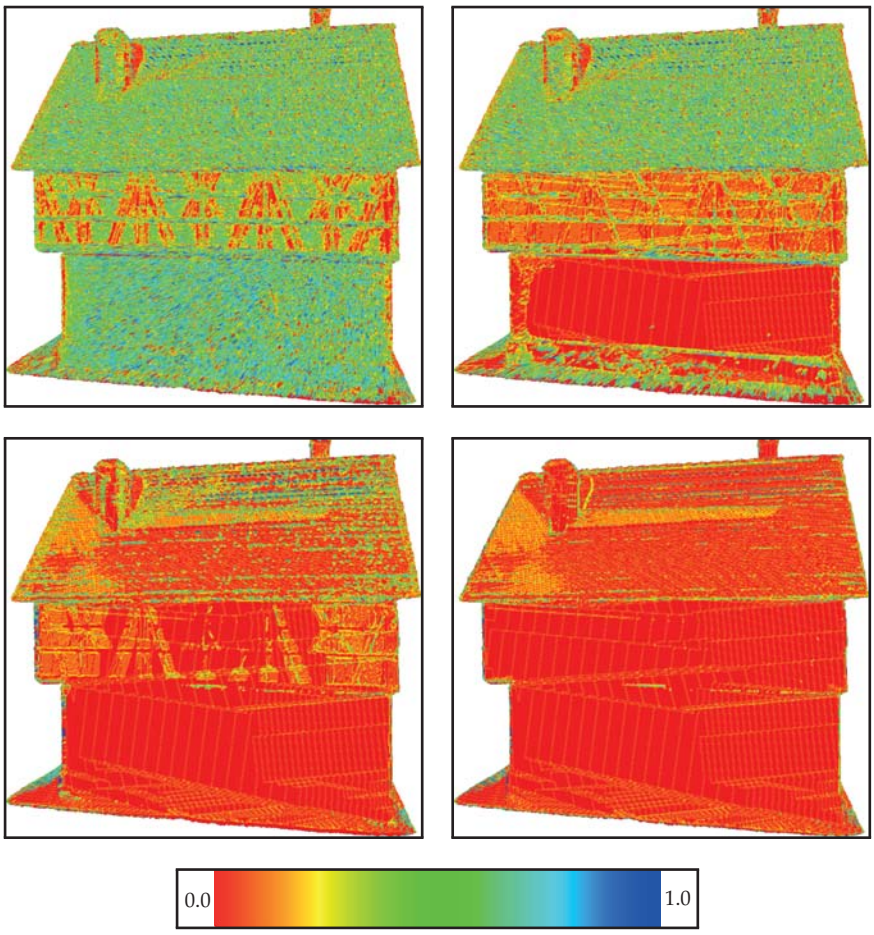


Abbildung 38: Einfluss des Gewichtungsfaktors $\lambda_S = \{1, 3, 5, 10\}$ auf die Objekt-Ebenheit bei dem Normalenverfahren

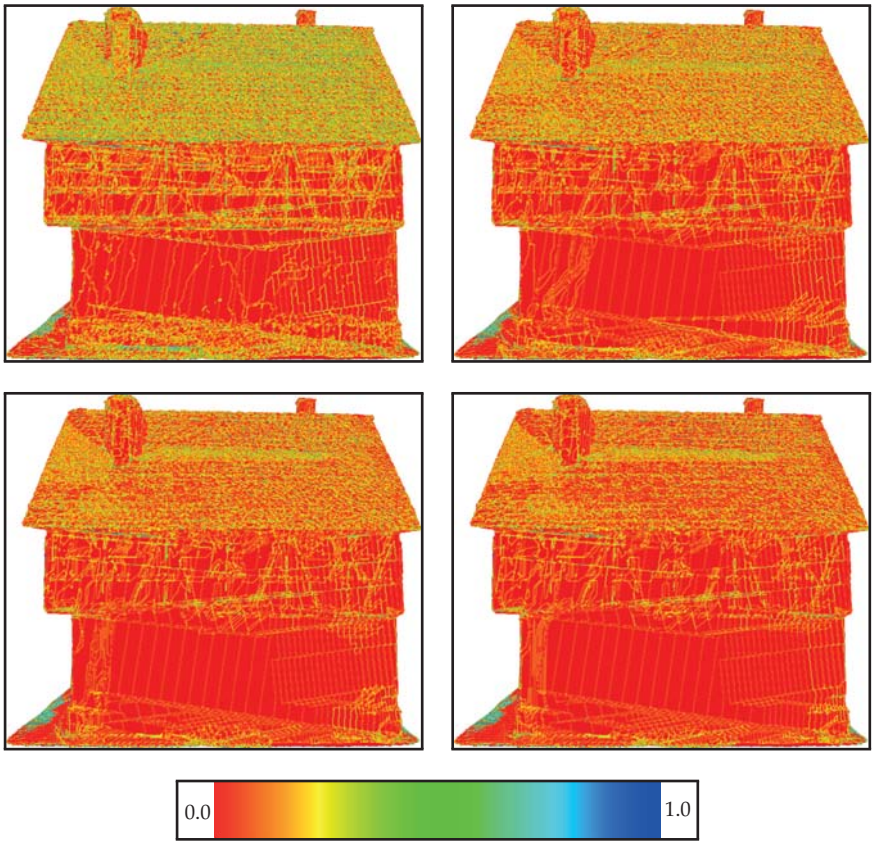


Abbildung 39: Einfluss des Gewichtungsfaktors $\lambda_S = \{10, 30, 50, 70\}$ auf die *Cauchy*-Funktion

5.5 Laufzeit und Speicherverbrauch

Ausgehend von den Daten aus Kapitel 3 werden verschiedene Tests durchgeführt um die Laufzeiten und den Speicherverbrauch der beiden Methoden zu vergleichen. Im ersten Test werden die beiden Verfahren für die Referenzansicht 0 des Hausmodells parallel ausgeführt. Für die wichtigen Einzelschritte werden die Ausführungszeiten gemessen. Dafür wurde zu Beginn des Routinenaufrufs und zum Ende der Ausführung die Systemzeit gemessen.

Obwohl das *Graph Cut* Verfahren global optimiert, ist es im direkten Vergleich mit dem lokalen *Simulated Annealing* Verfahren viel schneller (Tabelle 4). Für den Aufbau des Polygonnetzes und die Vorbereitung der Bilddaten werden fast 4 Minuten benötigt. Durch eine einfachere Datenstruktur und geringere Bildauflösungen, könnte an dieser Stelle viel Zeit eingespart werden. Auch die enorme Speicherauslastung von 3,5 bis 4 GB könnte dadurch eingeschränkt werden.

Routine	Laufzeit (s)
Laden des Polygonnetzes	75.6
Laden der Bilddaten	157.2
Sichtbarkeitstest	2.8
Simulated Annealing Algorithmus	3345.4
Graph Cut Algorithmus	2426.9

Tabelle 4: Laufzeit der einzelnen Routinen

Bei der Evaluierung des Einflusses der Anzahl der Vertices auf die Laufzeit der Verfahren wurde eine repräsentative Ansicht des Hausmodell ausgewählt und immer weiter vereinfacht (Abbildung 40). Für die Rekonstruktion wurden 5 Tiefenstufen angesetzt und jeweils 7 Ansichten für die Vor- und Rückschau. Das *Graph Cut* Verfahren ist bei einer höheren Anzahl an Vertices überlegen. Sinkt die Anzahl der Vertices, so nähern sich die Laufzeiten der beiden Verfahren aneinander an.

Untersucht man den Einfluss der Tiefenstufen auf die Laufzeit der beiden Verfahren, so kommt man zu ähnlichen Ergebnissen. Je mehr Vertices

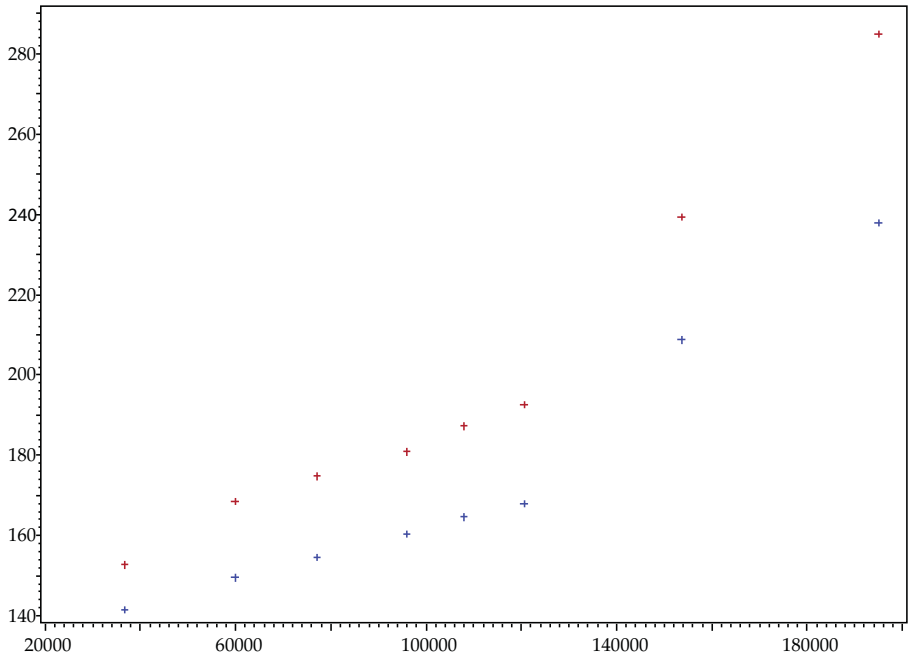


Abbildung 40: Einflusses der Anzahl der Vertices auf die Laufzeit (s) beim *Graph Cut* Verfahren (blau) und beim *Simulated Annealing* Verfahren (rot); die horizontale Achse gibt Anzahl der Polygone wieder und die vertikale Achse gibt die Laufzeit an

vorhanden sind, desto effizienter arbeitet der *Graph Cut* Algorithmus. Für die Rekonstruktion wurde eine repräsentative Ansicht des Hausmodells mit etwa 100000 sichtbaren Vertices ausgewählt.

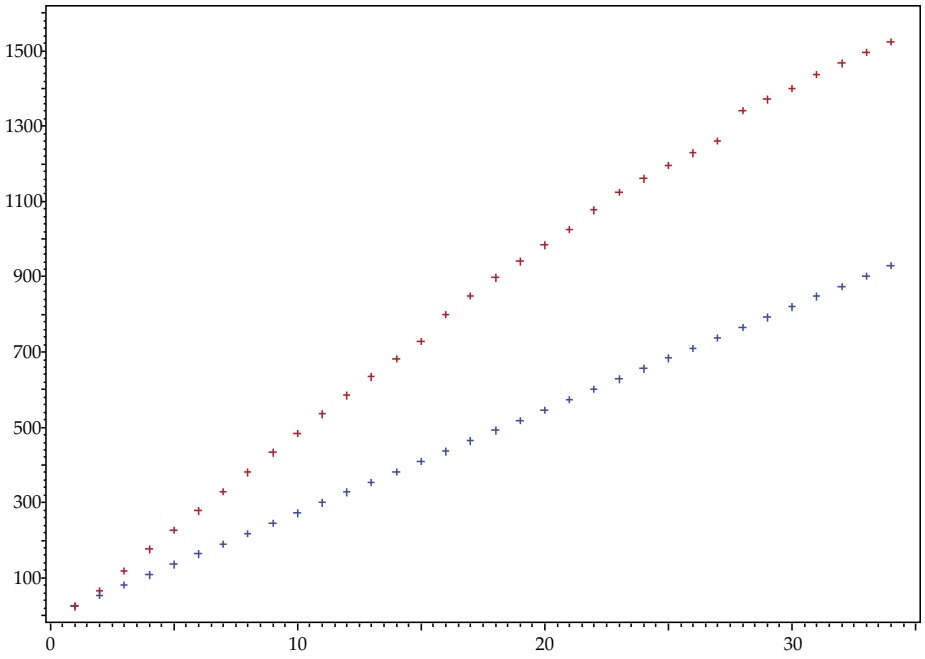


Abbildung 41: Einflusses der Anzahl der Tiefenstufen auf die Laufzeit beim *Graph Cut* Verfahren (blau) und beim *Simulated Annealing* Verfahren (rot); die horizontale Achse gibt Anzahl der Tiefenstufen wieder und die vertikale Achse gibt die Laufzeit an

5.6 Ergebnisse

Betrachtet man die Ergebnisse der beiden Verfahren nebeneinander, so wird deutlich, dass beim *Grah Cut* Verfahren wesentlich glattere Flächen erzeugt werden. Trotz der Ebenheit der Flächen werden die Details der Gebäudemodelle erhalten. Das Gesamtbild wirkt beim *Grah Cut* Verfahren deutlich genauer.

Dargestellt sind jeweils das Ausgangsbild und das Polygonmodell nach dem Silhouettenschnittverfahren. Dazu werden die Optimierungsmethoden nebeneinander abgebildet, um einen direkten Vergleich zu haben. Das texturierte Ergebnis wird aus dem *Grah Cut* Modell erzeugt.

Abbildung 42: Ergebnis beim Fachwerkhaus; (oben) Ausgangsbild und einfaches Polygonmodell nach dem Silhouettenschnittverfahren; (mitte) Optimierung mit *Simulated Annealing* und dem *Graph Cut* Verfahren; (unten) texturiertes Modell

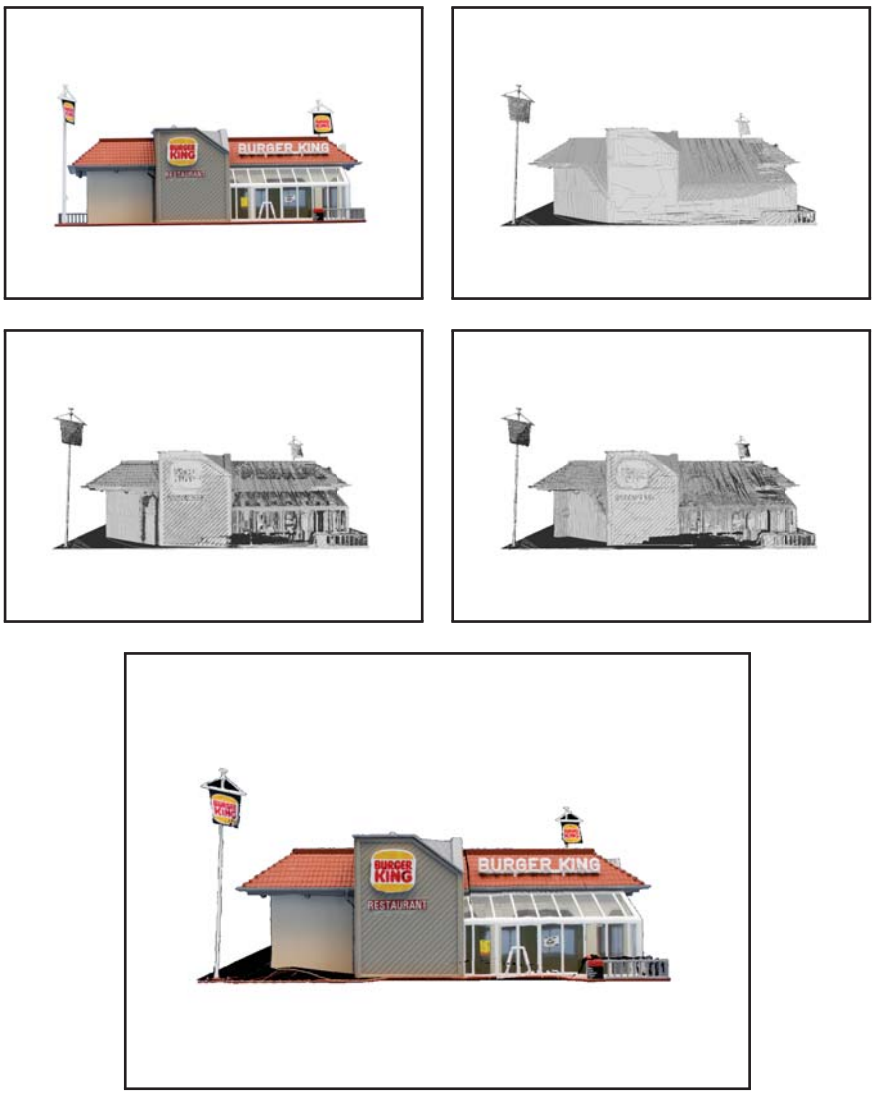


Abbildung 43: Ergebnis beim *Burger King* Modell; (oben) Ausgangsbild und einfaches Polygonmodell nach dem Silhouettenschnittverfahren; (mitte) Optimierung mit *Simulated Annealing* und dem *Graph Cut* Verfahren; (unten) texturiertes Modell

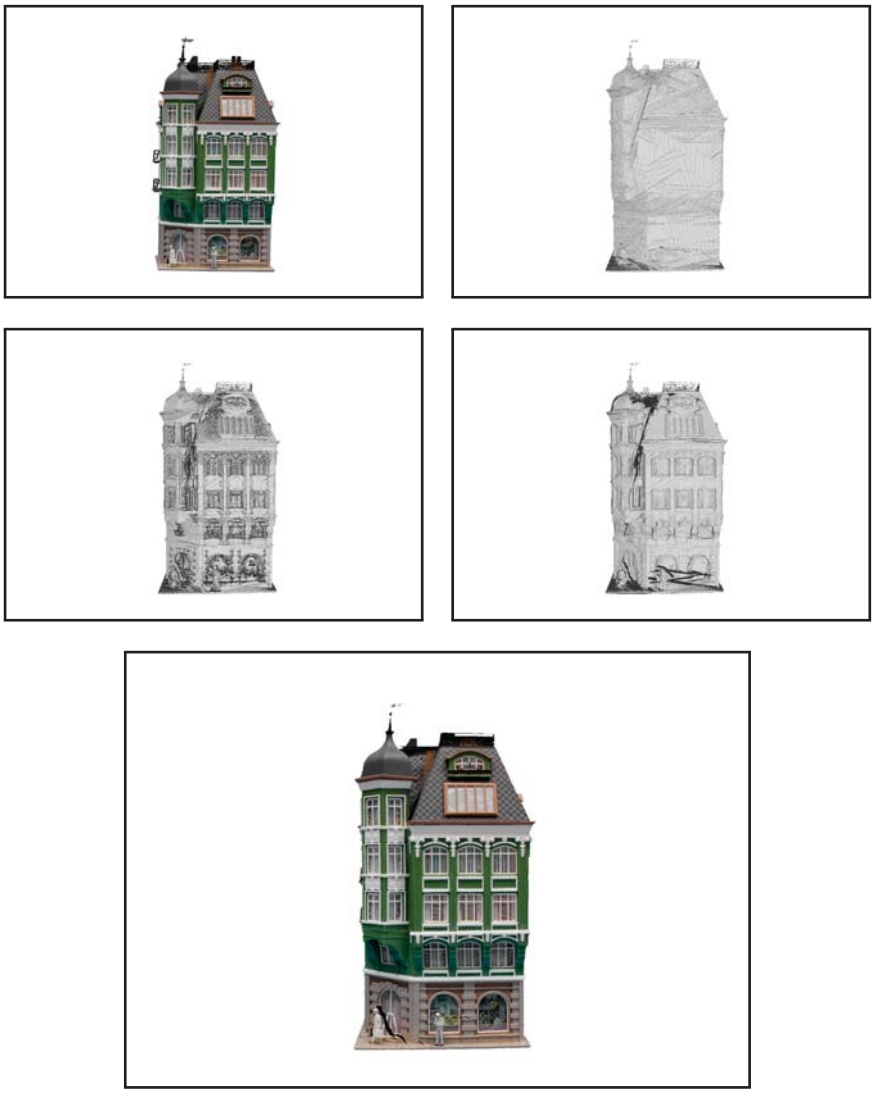


Abbildung 44: Ergebnis beim Bank Modell; (oben) Ausgangsbild und einfaches Polygonmodell nach dem Silhouettenschnittverfahren; (mitte) Optimierung mit *Simulated Annealing* und dem *Graph Cut* Verfahren; (unten) texturiertes Modell

6 Zusammenfassung und Ausblick

Die Aufgabenstellung, ein Hausmodell einzuscannen und es als dreidimensionales Objekt zu rekonstruieren, wurde mit zwei unterschiedlichen Verfahren gelöst. Dabei lag der Schwerpunkt bei der Optimierung der visuellen Hülle eines zuvor erzeugten dreidimensionalen Polygonnetzes. Es wurde ein lokaler und ein globaler Algorithmus in *C++* für das *shaper* Programm implementiert. Dafür wurden bekannte Verfahren analysiert, neu implementiert und für diese Arbeit angepasst und erweitert. Die Vor- und Nachteile der Verfahren wurden in den vorhergehenden Kapiteln ausführlich erläutert und mit den Ergebnissen bestätigt. Dabei stellte sich deutlich heraus, dass ein globaler Ansatz mit *Graph Cuts* die besten Ergebnisse bei den flächigen Gebäudemodellen liefert. Ein Nachteil der Implementierung ist die lange Laufzeit, die jedoch mit der Nutzung der GPU bei Bedarf leicht verbessert werden kann.

Die Güte der Rekonstruktion ist stark abhängig von der Qualität des Bildmaterials und der Kamerakalibrierung. Da die Bilder unter Laborbedingungen aufgenommen wurden, konnten sehr präzise Modelle erzeugt werden. Eine weitere Evaluierung des Verfahrens muss deshalb mit Außenaufnahmen *in situ* erfolgen.

Beide Verfahren können durch die freie Wahl der Parameter leicht an neue Bedürfnisse angepasst werden, um auch für andersartige Objekte qualitativ hochwertige Modelle zu erzeugen. Diese Freiheit wurde nicht durch eine automatisierte Parameterwahl eingeschränkt, um auch weitere Möglichkeiten der Nutzung offen zu lassen. Um bei der Parameterwahl eine Hilfestellung zu bieten, wurden mathematische Auswertungen erarbeitet.

Es gibt allerdings einige Einschränkungen die für ein vollständiges Rekonstruktionswerkzeug gelöst werden müssen. So ist etwa die ringförmige Anordnung der Bilder eine Vorgabe, die im nächsten Schritt der Entwicklung aufgelöst werden sollte, so dass eine freie uneingeschränkte Positionierung der Kamera auch ohne Drehteller möglich ist. Bilder in verschiedenen Auflösungen nutzen zu können sollte ebenfalls ermöglicht werden. Trotz dieser Einschränkungen können viele weitere Testdaten be-

arbeitet werden.

Durch die Nutzung der Polygonstruktur ist eine Weiterverarbeitung des Modells in anderen Anwendungen möglich. Für die Weiterverarbeitung wäre es allerdings sinnvoll das erzeugte Modell vorzubereiten. So kann etwa die Qualität der Polygone erhöht werden oder eine anschließende Polygonreduktion erzeugt ein handhabbares Modell für den weiteren Gebrauch.

Der Fokus der Arbeit lag jedoch bei der möglichst genauen und qualitativ hochwertigen Rekonstruktion eines Objektes als dreidimensionales Modell.

A Programmbeschreibung

Das Werkzeug *shaper* ist ein Kommandozeilenprogramm, welches speziell für die Rekonstruktion von Gebäudemodellen entwickelt wurde. Es bietet zur Einstellung der Parameter eine Verarbeitung von Konfigurationsdateien über die alle relevanten Werte gesetzt werden können.

Die Software wurde in objektorientierten *C++* entwickelt und verwendet die offenen Bibliotheken *OpenMesh*, von der *Computer Graphics Group* der RWTH Aachen[1] zur effizienten Verarbeitung der *Half-Edge*-Datenstruktur, die Referenzbibliothek *libpng* zur Verwaltung der PNG Bilddaten, und die Bibliotheken *GLUT* und *GLUI* der OpenGL Schnittstelle. Durch die offene Klassenstruktur ist es leicht neue Verfahren in das Programm zu integrieren und den Algorithmus anzupassen.

Über die Konfigurationsdatei werden verschiedene Parameter in Zeilenform angegeben. Der genaue Aufbau einer Kommandozeile der Konfiguration ist in *EBNF*-Form im Quelltext 3 dargestellt.

Ein Beispiel der Konfigurationsdatei ist in der Quelle 4 zu finden. Hier werden alle benötigten Parameter für die Optimierung und die geometrische Beschreibung angegeben. Die Mesh-Dateien können in verschiedenen Formaten, die durch die *OpenMesh*-Bibliothek zur Verfügung gestellt werden, eingelesen und ausgegeben werden.

[1] Rheinisch-Westfaelische Technische Hochschule Aachen

```
NUM    ::= "0" | "1" | "2" | "3" | "4" | "5" |
           "6" | "7" | "8" | "9" ;
INT    ::= ["+"|"-"] NUM* ;
FLOAT  ::= ["+"|"-"] NUM* ". " NUM* ;
CHAR   ::= "A" | "B" | "C" | "D" | ... | "Z" |
           "a" | "b" | ... | "z" ;
BOOL   ::= "true" | "false" ;
SPACE  ::= " " ;
END    ::= "\n" ;
VAR    ::= CHAR* ;

EXPRESSION     ::= INT | FLOAT | CHAR* | BOOL ;
_COMMENT       ::= "#" [NUM|CHAR|SPACE]* END ;
_ECHO          ::= "ECHO" SPACE CHAR* ;
_VALUE         ::= "VALUE" SPACE VAR SPACE
                   EXPRESSION ;
_MESH          ::= "MESH" SPACE CHAR* ;
_OUTPUT        ::= "OUTPUT" SPACE CHAR* ;
_IMAGE         ::= "IMAGE" SPACE INT SPACE CHAR* ;
_ORIENTATION   ::= "ORIENTATION" SPACE INT SPACE
                   7*FLOAT ;
_PROJECTION    ::= "PROJECTION" SPACE INT SPACE
                   12*FLOAT ;
COMMAND        ::= _COMMENT | _VALUE | _IMAGE |
                   _MESH | _OUTPUT | _ORIENTATION |
                   _PROJECTION ;
```

Quelltext 3: EBNF der Konfigurationsdatei

```
ECHO "loading and executing configuration file"

VALUE fNear 0.1
VALUE fFar 100.0
VALUE fHA 0.2877

VALUE shapeDepthSteps 25
VALUE shapeDepth 0.0002
VALUE forwardPreviewImages 7
VALUE backwardPreviewImages 7
VALUE shapePosition 0
VALUE useOcclusionTest true
VALUE occlusionLimit 0.9
VALUE hiddenSurface render
VALUE algorithm gc

VALUE maxflowIterationsGC 3
VALUE weightSmoothGC 30

MESH "house.obj"
OUTPUT "output.obj"

IMAGE 00 "1933164922_0000.png"
IMAGE 04 "1933164922_0004.png"

ORIENTATION 00 0.988925 0.0050851 0.148332
   0.0693042 -0.00615527 0 -0.5
ORIENTATION 04 0.0365212 -0.998896 -0.0295423
   1.65881 -0.00615527 0 -0.5
PROJECTION 00 -11928.910061 299.903416
   2577.512929 -1222.570455 122.681913
   12019.243550 907.008183 -864.000000 0.000000
   0.068484 0.997652 -0.500000
PROJECTION 04 3607.377731 299.903416
   11658.871903 -1222.570455 892.864310
   12019.243550 -201.266044 -864.000000 0.993856
   0.068484 -0.086951 -0.500000
```

Quelltext 4: Beispiel einer Konfigurationsdatei

B Abkürzungsverzeichnis

2D	zweidimensional
3D	dreidimensional
AR	Augmented Reality
BMD	Bimodality
BRDF	Bidirectional Reflectance Distribution Function
CAD	Computer Aided Design
CCD	Charge-coupled Device Sensor
CD	Cauchy-Distanz
cm	Zentimeter
CT	Computertomographie
CV	Kovarianz-Varianz-Verfahren
GB	Gigabyte
GD	Gaußsche-Distanz
GPU	Graphics Processing Unit
JPEG	Joint Photographic Experts Group
KB	Kilobyte
LRC	Left Right Checking oder Cross Checking
m	Minute
MB	Megabyte
MGJ	Match Goodness Jumps
MM	Minkowski-Metrik
MRT	Magnetresonanztomographie
MRF	Markov Random Field
OCC	Occlusion Constraint
ORD	Point Ordering Constraint
PNG	Portable Network Graphics
RGB	Rot, Grün und Blau Farbraum
s	Sekunde
SAD	Sum of Absolute Differences
SCP	Sum of Cross Products
SFS	Shape From Silhouette
SSD	Sum of Squared Differences

C Abbildungsverzeichnis

1 Beispiel eines einfachen Voxelmodells 11
2 *Half-Edge* Datenstruktur 13
3 Lochkamera Modell 17
4 Fotografien aus verschiedenen Perspektiven 20
5 *Marching-Cubes Lookup*-Tabelle 21
6 Drehteller der Firma Kaidan 33
7 6 von 50 Aufnahmen des Fachwerkhauses 34
8 4 von 72 Aufnahmen des *Burger King* Modells 34
9 8 von 72 Aufnahmen des Bankhauses 34
10 Schnittmaske für das Hausmodell 35
11 Positionsfehler eines Oberflächenpunktes 35
12 Positionsfehler in einem Voxel 37
13 Fehler durch begrenzte Anzahl der Kamerabilder 38
14 Ablaufplan der automatischen Objektrekonstruktion 39
15 Sichtbarkeitstiefentest mit Polygonen in Falschfarben 41
16 Fehler an den Polygonübergängen im Sichtbarkeitstiefentest 42
17 Hausmodell partitioniert in einem Hashgrid 43
18 Zwei Arten von Verdeckungsfehlern 46
19 Lokale Optimierung mit *Simulated Annealing* 54
20 Angepasste *Cauchy-Funktion* 59
21 *s*-*t*-Schnitt eines Graphen 61
22 *α*-*β*-*Swap* und *α*-*Expansion* 62
23 Suchbäume des Minimale-Schnitt-Maximale-Fluss Algorithmus 64
24 Referenzansichten der Gebäudefassaden für den Sichtbarkeitstest 66
25 Polygonverteilung beim Sichtbarkeitstest 68
26 Verteilung der Verdeckungfehler 71
27 Visualisierung der Modellebenheit 73

28 Visualisierung der Modellebenheit 74
29 Auswirkungen des Tiefenterms 74
30 Visualisierung der Polygonqualität 75
31 Verteilung der Fehlerabweichungen 76
32 Anzahl der Fehlerberechnungen im Vergleich 77
33 Visualisierung der Tiefenfehler 79
34 Akkumulation der Tiefenwerte 80
35 Filtergröße und Überschneidungen 80
36 Visualisierung der Glattheitseigenschaften 81
37 Visualisierung der Glattheitseigenschaften 82
38 Visualisierung der Glattheitseigenschaften 83
39 Visualisierung der Glattheitseigenschaften 84
40 Einfluss der Anzahl der Vertices auf die Laufzeit 86
41 Einfluss der Anzahl der Tiefenstufen auf die Laufzeit 87
42 Ergebnis beim Fachwerkhaus 89
43 Ergebnis beim *Burger King* Modell 90
44 Ergebnis beim Bank Modell 91

D Tabellenverzeichnis

1 Sichtbarkeitstest für das Hausmodell 69
2 Sichtbarkeitstest für das *Burger King* Modell 69
3 Sichtbarkeitstest für das Bankmodell 69
4 Laufzeit der einzelnen Routinen 85

E Quelltextverzeichnis

1 Pseudocode der *Simulated Annealing* Optimierung 55
2 Pseudocode der *Graph Cut* α-Entfaltung 61
3 EBNF der Konfigurationsdatei II
4 Beispiel einer Konfigurationsdatei III

F Literaturverzeichnis

[AHP12] Abrams, Austin ; Hawley, Christopher ; Pless, Robert: Heliometric Stereo: Shape From Sun Position. In: *European Conference on Computer Vision (ECCV)*, 2012

[App68] Appel, Arthur: Some techniques for shading machine renderings of solids. In: *Proceedings of the April 30–May 2, 1968, spring joint computer conference*. New York, USA : ACM, 1968, S. 37–45

[AVC08] Auclair, Adrien ; Vincent, Nicole ; Cohen, Laurent D.: Using point correspondences without projective deformation for multi-view stereo reconstruction. In: *Image Processing ICIP*, 2008, S. 193–196

[AW98] Anton, Howard ; Walz, Anke: *Lineare Algebra*. 1. Auflage. Heidelberg : Spektrum, Akademischer Verlag, 1998. – ISBN 978–3–827–40324–7

[BB82] Ballard, Dana H. ; Brown, Christopher M.: *Computer vision*. New York : Prentice-Hall, 1982. – ISBN 978–0–131–65316–0

[BB06] Bender, Michael ; Brill, Manfred: *Computergrafik: ein anwendungsorientiertes Lehrbuch*. 2. Auflage. München : Hanser Verlag, 2006. – ISBN 3–446–40434–1

[BBH03] Brown, Myron Z. ; Burschka, Darius ; Hager, Gregory D.: Advances in Computational Stereo. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 25 (2003), August, Nr. 8, S. 993–1008. – ISSN 0162–8828

[BGZ02] Bungartz, Hans-Joachim ; Griebel, Michael ; Zenger, Christoph: *Einführung in die Computergraphik - Grundlagen, Geometrische Modellierung, Algorithmen*. 2. Auflage. Wiesbaden : Vieweg+Teubner Verlag, 2002. – ISBN 978–3–528–16769–1

[BK04] Boykov, Yuri ; Kolmogorov, Vladimir: An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 26 (2004), September, Nr. 9, S. 1124–1137. – ISSN 0162–8828

[BMR01] Bernardini, Fausto ; Martin, Ioana M. ; Rushmeier, Holly: High-quality texture reconstruction from multiple scans. In: *IEEE Transactions on Visualization and Computer Graphics* 7 (2001), S. 318–332

[BN97] Bhat, Dinkar N. ; Nayar, Shree K.: Ordinal Measures for Visual Correspondence. In: *In IEEE Conference on Computer Vision and Pattern Recognition*, 1997, S. 351–357

[BSBK02] Botsch, M. ; Steinberg, S. ; Bischoff, S. ; Kobbelt, L.: OpenMesh - a generic and efficient polygon mesh data structure. In: *In OpenSG Symposium*, 2002

[BVZ98] Boykov, Y. ; Veksler, O. ; Zabih, R.: Markov Random Fields with Efficient Approximations. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Washington, DC, USA : IEEE Computer Society, 1998 (CVPR '98). – ISBN 0–8186–8497–6, S. 648–

[BVZ01] Boykov, Yuri ; Veksler, Olga ; Zabih, Ramin: Fast Approximate Energy Minimization via Graph Cuts. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 23 (2001), November, Nr. 11, S. 1222–1239. – ISSN 0162–8828

[Cav74] Cavendish, J. C.: Automatic triangulation of arbitrary planar domains for the finite element method. In: *International Journal for Numerical Methods in Engineering* 8 (1974), S. 679–696

[Che95] Chernyaev, Evgeni V.: Marching Cubes 33: Construction of Topologically Correct Isosurfaces. In: *Technical Report CN/95-17* (1995)

[CLCG08] Cornelis, Nico ; Leibe, Bastian ; Cornelis, Kurt ; Gool, Luc: 3D Urban Scene Modeling Integrating Recognition and Reconstruction. In: *International Journal of Computer Vision* 78 (2008), Juli, Nr. 2-3, S. 121–141. – ISSN 0920–5691

[CLL08] Chang, Ju Y. ; Lee, Kyoung M. ; Lee, Sang U.: Shape from shading using graph cuts. In: *Pattern Recognition* 41 (2008), Nr. 12, S. 3749–3757

[Col96] Collins, Robert T.: A Space-Sweep Approach to True Multi-Image Matching. In: *Conference on Computer Vision and Pattern Recognition*, 1996, S. 358–363

[Cor05] Corrochano, E.B.: *Handbook of Geometric Computing: Applications in Pattern Recognition, Computer Vision, Neuralcomputing, and Robotics*. Springer, 2005. – ISBN 9783540205951

[CS09] Cyganek, Boguslaw ; Siebert, J. P.: *An Introduction to 3D Computer Vision Techniques and Algorithms*. West Sussex : John Wiley & Sons, 2009. – ISBN 978–1–119–96447–6

[Cyg05] Cyganek, Boguslaw: Adaptive Window Growing Technique for Efficient Image Matching. In: *IbPRIA*, Springer-Verlag, 2005, S. 308–315

[Del34] Delaunay, Boris N.: Sur la sphère vide. In: *Bulletin of Academy of Sciences of the USSR* (1934), Nr. 6, S. 793–800

[Din70] Dinic, E. A.: Algorithm for Solution of a Problem of Maximum Flow in a Network with Power Estimation. In: *Soviet Math Doklady* 11 (1970), S. 1277–1280

[Dok13] Dokumentation, OpenMesh: *The Halfedge Data Structure*. `http://openmesh.org/Documentation/OpenMesh-Doc-Latest/mesh_hds.html`, Konsultationsdatum: 23. März 2013

[DTC04] Dick, A. R. ; Torr, P. H. S. ; Cipolla, R.: Modelling and Interpretation of Architecture from Several Images. In: *International Journal of Computer Vision* 60 (2004), November, Nr. 2, S. 111–134. – ISSN 0920–5691

[DTM96] Debevec, Paul E. ; Taylor, Camillo J. ; Malik, Jitendra: Modeling and rendering architecture from photographs: a hybrid geometry- and image-based approach. In: *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. New York, USA : ACM, 1996 (SIGGRAPH '96). – ISBN 0–89791–746–4, S. 11–20

[Dye01] Dyer, Charles R.: Volumetric Scene Reconstruction From Multiple Views. In: *Foundations of Image Understanding*, Kluwer, 2001, S. 469–489

[Eck00] Eckert, Gerald: Automatic Shape Reconstruction of Rigid 3-D Objects from Multiple Calibrated Images. In: *Eusipco 2000 Proceedings, September 4-8, 2000, Tampere, Finland* (2000). – Paper

[EHS02] Esteban, Carlos H. ; Hernandez, Carlos ; Schmitt, Esteban F.: Multi-Stereo 3D Object Reconstruction. In: *3DPVT '02*, 2002, S. 159–166

[Eis04] Eisert, Peter: 3-D Geometry Enhancement by Contour Optimization in Turntable Sequences. In: *International Conference on Image Processing*, 2004, S. 3021–3024

[ES04] Esteban, Carlos H. ; Schmitt, Francis: Silhouette and stereo fusion for 3D object modeling. In: *Computer Vision and Image Understanding* 96 (2004), Dezember, Nr. 3, S. 367–392. – ISSN 1077–3142

[ESG99] Eisert, Peter ; Steinbach, Eckehard ; Girod, Bernd: Multihypothesis volumetric reconstruction of 3-D objects from multiple calibrated camera views. In: *International Conference on Acoustics, Speech, and Signal Processing*, 1999, S. 3509–3512

[EW02] Egnal, Geoffrey ; Wildes, Richard P.: Detecting Binocular Half-Occlusions: Empirical Comparisons of Five Approaches. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24 (2002), August, Nr. 8, S. 1127–1133. – ISSN 0162–8828

[EWN01] Eckert, G. ; Wingbermuehle, J. ; Niem, W.: Shape Refinement for Reconstructing 3D-Objects Using an Analysis-Synthesis Approach. In: *International Conference on Image Processing* (2001)

[EWN04] Eckert, G. ; Wingbermühle, J. ; Niem, W.: Mesh Based Shape Refinement for Reconstructing 3D-Objects from Multiple Images. In: *1st European Conference on Visual Media Production*, 2004, S. 103–110

[Fas12] Fastermann, Petra: *3D-Druck Rapid Prototyping*. 2012. Berlin : Springer, 2012. – ISBN 978–3–642–29225–5

[FCSS09] Furukawa, Yasutaka ; Curless, Brian ; Seitz, Steven M. ; Szeliski, Richard: Manhattan-world stereo. In: *Computer Vision and Pattern Recognition*, 2009, S. 1422–1429

[FF56] Ford, L. R. ; Fulkerson, D. R.: Maximal Flow through a Network. In: *Canadian Journal of Mathematics* 8 (1956), S. 399–404

[FF62] Ford, L. R. ; Fulkerson, D. R.: *Flows in Networks*. Princeton University Press, 1962

[FL94] Fua, P. ; Leclerc, Y. G.: Registration Without Correspondences. In: *Conference on Computer Vision and Pattern Recognition*. Seattle, WA, Jun 1994, S. 121–128

[FL95] Fua, P. ; Leclerc, Y.G.: Object-Centered Surface Reconstruction: Combining Multi-Image Stereo and Shading. In: *International Journal of Computer Vision* 16 (1995), S. 35–56

[FLP01] Faugeras, Olivier D. ; Luong, Quang-Tuan ; Papadopoulo, Théodore: *The geometry of multiple images - the laws that govern the formation of multiple images of a scene and some of their applications*. MIT Press, 2001. – ISBN 978–0–262–06220–6

[FP10] Furukawa, Yasutaka ; Ponce, Jean: Accurate, Dense, and Robust Multiview Stereopsis. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 32 (2010), Nr. 8, S. 1362–1376

[Frü02] Früh, Christian: *Automated 3D model generation for urban environments*, Diss., 2002

[Fua97] Fua, P.: From Multiple Stereo Views to Multiple 3-D Surfaces. In: *International Journal of Computer Vision* 24 (1997), August, Nr. 1, S. 19–35. – ISSN 0920–5691

[GCS06] Goesele, Michael ; Curless, Brian ; Seitz, Steven M.: Multi-View Stereo Revisited. In: *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2*. Washington, DC, USA : IEEE Computer Society, 2006 (CVPR '06). – ISBN 0–7695–2597–0, S. 2402–2409

[HK07] Habbecke, Martin ; Kobbelt, Leif: A Surface-Growing Approach to Multi-View Stereo Reconstruction. In: *Computer Vision and Pattern Recognition*, 2007

[HZ04] Hartley, R.I. ; Zisserman, A.: *Multiple View Geometry in Computer Vision*. Second. Cambridge University Press, 2004

[IS03] Isidoro, John ; Sclaroff, Stan: Stochastic Refinement of the Visual Hull to Satisfy Photometric and Silhouette Consistency Constraints. In: *Proceedings of the Ninth IEEE International Conference on Computer Vision - Volume 2*. Washington, DC, USA : IEEE Computer Society, 2003 (ICCV '03). – ISBN 0–7695–1950–4, S. 1335–1342

[KBSS01] Kobbelt, Leif P. ; Botsch, Mario ; Schwanecke, Ulrich ; Seidel, Hans-Peter: Feature sensitive surface extraction from volume data. In: *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. New York, NY, USA : ACM, 2001 (SIGGRAPH '01). – ISBN 1–58113–374–X, S. 57–66

[KHV04] Kück, Hendrik ; Heidrich, Wolfgang ; Vogelgsang, Christian: Shape from Contours and Multiple Stereo - A Hierarchical, Mesh-Based Approach. In: *1st Canadian Conference on Computer and Robot Vision*, 2004, S. 76–83

[Kol04] Kolmogorov, Vladimir: *Graph based algorithms for scene reconstruction from two or more views*. NY, USA, Diss., 2004

[KS98] Kutulakos, Kiriakos N. ; Seitz, Steven M.: What Do N Photographs Tell Us about 3D Shape? Rochester, NY, USA : University of Rochester, 1998. – Forschungsbericht

[KS00] Kutulakos, Kiriakos N. ; Seitz, Steven M.: A Theory of Shape by Space Carving. In: *International Journal of Computer Vision* 38 (2000), Juli, Nr. 3, S. 199–218. – ISSN 0920–5691

[KSC01] Kang, Sing B. ; Szeliski, R. ; Chai, Jinxiang: Handling occlusions in dense multi-view stereo. In: *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on* Bd. 1, IEEE, 2001. – ISBN 0–7695–1272–0, S. I–103–I–110 vol.1

[KZ01] Kolmogorov, Vladimir ; Zabih, Ramin: Computing Visual Correspondence with Occlusions using Graph Cuts. In: *International Conference on Computer Vision*, 2001, S. 508–515

[KZ02a] Kolmogorov, Vladimir ; Zabih, Ramin: Multi-camera Scene Reconstruction via Graph Cuts. In: *Proceedings of the 7th European Conference on Computer Vision-Part III*. London, UK, UK : Springer-Verlag, 2002 (ECCV '02). – ISBN 3–540–43746–0, S. 82–96

[KZ02b] Kolmogorov, Vladimir ; Zabih, Ramin: What Energy Functions Can Be Minimized via Graph Cuts? In: *Proceedings of the 7th European Conference on Computer Vision-Part III*. London, UK, UK : Springer-Verlag, 2002 (ECCV '02). – ISBN 3–540–43746–0, S. 65–81

[Lau94] Laurentini, A.: The Visual Hull Concept for Silhouette-Based Image Understanding. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 16 (1994), Februar, Nr. 2, S. 150–162. – ISSN 0162–8828

[Lau95] Laurentini, Aldo: How Far 3D Shapes Can Be Understood from 2D Silhouettes. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 17 (1995), Nr. 2, S. 188–195

[LC87] Lorensen, William E. ; Cline, Harvey E.: Marching cubes: A high resolution 3D surface construction algorithm. In: *COMPUTER GRAPHICS* 21 (1987), Nr. 4, S. 163–169

[LQ02] Lhuillier, Maxime ; Quan, Long: Match Propogation for Image-Based Modeling and Rendering. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24 (2002), August, Nr. 8, S. 1140–1146. – ISSN 0162–8828

[LSL+05] Lehmann, Thomas M. ; Stock, Stephanie ; Lehmacher, Walter ; Köhler, Claus O. ; Spreckelsen, Cord ; Spitzer, Klaus ; Meinzer, Hans-Peter ; Lauterbach, Karl W. ; Langowski, Jörg: *Handbuch der medizinischen Informatik*. 2. vollständig überarbeitete Auflage. München : Hanser Verlag, 2005. – ISBN 978–3–446–22701–9

[MA83] Martin, Worthy N. ; Aggarwal, J. K.: Volumetric Descriptions of Objects from Multiple Views. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 5 (1983), Februar, Nr. 2, S. 150–158. – ISSN 0162–8828

[MFK99] Matsumoto, Y. ; Fujimura, K. ; Kitamura, T.: Shape-from-Silhouette/Stereo and Its Application to 3-D Digitizer. In:

Proceedings of the 8th International Conference on Discrete Geometry for Computer Imagery. London, UK, UK : Springer-Verlag, 1999 (DCGI '99). – ISBN 3–540–65685–5, S. 177–190

[MM05] Mercier, Bruno ; Meneveaux, Daniel: Shape from Silhouette: Image Pixels for Marching Cubes. In: *WSCG (Journal Papers)'05*, 2005, S. 112–118

[MMHM02] Mühlmann, Karsten ; Maier, Dennis ; Hesser, Jürgen ; Männer, Reinhard: Calculating Dense Disparity Maps from Color Stereo Images, an Efficient Implementation. In: *International Journal of Computer Vision* 47 (2002), April, Nr. 1-3, S. 79–88. – ISSN 0920–5691

[MS09] Meierhold, N. ; Schmich, A: Referencing of images to laser scanner data using linear features extracted from digital images and range images. In: *International Archives of Photogrammetry, Remote Sensing and Spatial Information Science* Bd. Volume XXXVIII-3/W8., 2009

[MT97] Möller, Tomas ; Trumbore, Ben: Fast, minimum storage ray-triangle intersection. In: *Journal of Graphics Tools* 2 (1997), Oktober, Nr. 1, S. 21–28. – ISSN 1086–7651

[Nie94] Niem, Wolfgang: Robust and fast modelling of 3D natural objects from multiple views. In: *SPIE Proceedings Image and Video Processing II*, 1994, S. 388–397

[Nie97] Niem, Wolfgang: Error Analysis for Silhouette-Based 3D Shape Estimation from Multiple Views. In: *International Workshop on Synthetic-Natural Hybrid Coding and Three. Dimensional Imaging* (1997), September

[Nie98] Niem, Wolfgang: *Automatische Rekonstruktion starrer dreidimensionaler Objekte aus Kamerabildern*, Universität Hannover, Diss., 1998

[NM65] Nelder, J. A. ; Mead, R.: A Simplex Method for Function Minimization. In: *The Computer Journal* 7 (1965), Januar, Nr. 4, S. 308–313. – ISSN 1460–2067

[NMSO96] Nakamura, Yuichi ; Matsuura, Tomohiko ; Satoh, Kiyohide ; Ohta, Yuichi: Occlusion Detectable Stereo - Occlusion Patterns in Camera Matrix. In: *Conference on Computer Vision and Pattern Recognition*, 1996, S. 371–378

[NY06] Newman, Timothy S. ; Yi, Hong: A survey of the marching cubes algorithm. In: *Computers & Graphics* 30 (2006), Nr. 5, S. 854–879

[OK93] Okutomi, M. ; Kanade, T.: A Multiple-Baseline Stereo. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 15 (1993), April, Nr. 4, S. 353–363. – ISSN 0162–8828

[PD98] Prock, Andrew ; Dyer, Charles R.: Towards Real-Time Voxel Coloring. In: *Image understanding workshop*, 1998, S. 315–321

[PNF+08] Pollefeys, M. ; Nistér, D. ; Frahm, J. M. ; Akbarzadeh, A. ; Mordohai, P. ; Clipp, B. ; Engels, C. ; Gallup, D. ; Kim, S. J. ; Merrell, P. ; Salmi, C. ; Sinha, S. ; Talton, B. ; Wang, L. ; Yang, Q. ; Stewénius, H. ; Yang, R. ; Welch, G. ; Towles, H.: Detailed Real-Time Urban 3D Reconstruction from Video. In: *International Journal of Computer Vision* 78 (2008), Juli, Nr. 2-3, S. 143–167. – ISSN 0920–5691

[Pro13] Project, Middlebury: *A Comparison and Evaluation of Multi-View Stereo Reconstruction Algorithms*. `http://vision.middlebury.edu/mview/eval/`, Konsultationsdatum: 12. April 2013

[PTVF07] Press, William H. ; Teukolsky, Saul A. ; Vetterling, William T. ; Flannery, Brian P.: *Numerical Recipes: The Art of Scientific Computing*. 3. New York, NY, USA : Cambridge University Press, 2007. – ISBN 0521880688, 9780521880688

[Pul97] Pulli, Kari A.: *Surface reconstruction and display from range and color data*, Diss., 1997. – AAI9819292

[RC98] Roy, Sebastien ; Cox, Ingemar J.: A Maximum-Flow Formulation of the N -camera Stereo Correspondence Problem. In: *IEEE International Conference on Computer Vision*. Bombai, 1998

[SCD+06] Seitz, Steven M. ; Curless, Brian ; Diebel, James ; Scharstein, Daniel ; Szeliski, Richard: A Comparison and Evaluation of Multi-View Stereo Reconstruction Algorithms. In: *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 1*. Washington, DC, USA : IEEE Computer Society, 2006 (CVPR '06). – ISBN 0–7695–2597–0, S. 519–528

[SCMS01] Slabaugh, Greg ; Culbertson, Bruce ; Malzbender, Tom ; Schafer, Ron: A survey of methods for volumetric scene reconstruction from photographs. In: *Proceedings of the 2001 Eurographics conference on Volume Graphics*. Aire-la-Ville, Switzerland, Switzerland : Eurographics Association, 2001 (VG'01). – ISBN 3–211–83737–X, S. 81–101

[SD97] Seitz, Steven M. ; Dyer, Charles R.: Photorealistic Scene Reconstruction by Voxel Coloring. In: *International Journal of Computer Vision*, 1997, S. 1067–1073

[SE12] Schneider, David C. ; Eisert, Peter: On User-Interaction in 3D Reconstruction. In: *Eurographics* Bd. EG 2012, 2012, S. 13–14

[Sed92] Sedgewick, Robert: *Algorithmen in C*. Bonn : Addison Wesley Verlag, 1992. – ISBN 978–3–8273–7182–9

[SGEB00] Steinbach, Eckehard ; Girod, Bernd ; Eisert, Peter ; Betz, Arnulf: 3-D Reconstruction Of Real-World Objects Using Extended Voxels. In: *International Conference on Image Processing*, 2000, S. 823–826

[Sla00] Slabaugh, Gregory G.: *Multi-View 3D Scene Reconstruction Using Arbitrarily Positioned Cameras*, Georgia Institute of Technology, Diss., 2000

[SLMHP02] Schirmacher, Hartmut ; Li, Ming ; Magnor, Marcus A. ; Hans-PeterSeidel: Combining stereo and visual hull information for on-line reconstruction and rendering of dynamic scenes. In: *IEEE Workshop on Multimedia Signal Processing*, 2002, S. 9–12

[SS98] Scharstein, Daniel ; Szeliski, Richard: Stereo Matching with Non-Linear Diffusion. In: *International Journal of Computer Vision* 28 (1998), S. 155–174

[SS01] Stockman, George ; Shapiro, Linda G.: *Computer Vision*. 1st. Upper Saddle River, NJ, USA : Prentice Hall PTR, 2001. – ISBN 0130307963

[SS02] Scharstein, Daniel ; Szeliski, Richard: A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms. In: *International Journal of Computer Vision* 47 (2002), Nr. 1-3, S. 7–42. – ISSN 0920–5691

[SSS+08] Sinha, Sudipta N. ; Steedly, Drew ; Szeliski, Richard ; Agrawala, Maneesh ; Pollefeys, Marc: Interactive 3D architectural modeling from unordered photo collections. In: *ACM Transactions on Graphics* 27 (2008), Dezember, Nr. 5, S. 159:1–159:10. – ISSN 0730–0301

[Sze93] Szeliski, Richard: Rapid octree construction from image sequences. In: *CVGIP: Image Understanding* 58 (1993), Juli, Nr. 1, S. 23–32. – ISSN 1049–9660

[Vek03] Veksler, Olga: Fast variable window for stereo correspondence using integral images. In: *Proceedings of the 2003 IEEE computer society conference on Computer vision and pattern recognition*, 2003. – ISBN 0–7695–1900–8, 978–0–7695–1900–5, S. 556–561

[VHC06] Vogiatzis, G. ; Hernández, C. ; Cipolla, R.: Lighting-up geometry: accurate 3D modelling of museum artifacts with a torch and a camera. In: *Eurographics*. Vienna, 2006, S. 85–88

[Wei97] Weik, S.: Registration of 3-D Partial Surface Models Using Luminance and Depth Information. In: *International Conference on Recent Advances in 3-D Digital Imaging and Modeling*, 1997, S. 93–100

[WZ02] Werner, T. ; Zisserman, A.: Model Selection for Automated Architectural Reconstruction from Multiple Views. In: *British Machine Vision Conference*, 2002, S. 53–62

[XFT+08] Xiao, Jianxiong ; Fang, Tian ; Tan, Ping ; Zhao, Peng ; Ofek, Eyal ; Quan, Long: Image-based facade modeling. In: *ACM Transactions on Graphics* 27 (2008), Dezember, Nr. 5, S. 161:1–161:10. – ISSN 0730–0301

[XFZ+09] Xiao, Jianxiong ; Fang, Tian ; Zhao, Peng ; Lhuillier, Maxime ; Quan, Long: Image-based street-side city modeling. In: *ACM Transactions on Graphics* 28 (2009), Dezember, Nr. 5, S. 114:1–114:12. – ISSN 0730–0301

[YSB+02] Yezzi, Anthony J. ; Slabaugh, Gregory G. ; Broadhurst, Adrian ; Cipolla, Roberto ; Schafer, Ronald W.: A Surface Evolution Approach to Probabilistic Space Carving. In: *3DPVT*, IEEE Computer Society, 2002. – ISBN 0–7695–1521–5, S. 618–621

[Zhe94] Zheng, Jiang Y.: Acquiring 3-D Models from Sequences of Contours. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 16 (1994), Nr. 2, S. 163–178

[ZKGGK06] Zebedin, Lukas ; Klaus, Andreas ; Gruber-Geymayer, Barbara ; Karner, Konrad: Towards 3D map generation from digital aerial images. In: *ISPRS Journal of Photogrammetry and Remote Sensing* 60 (2006), Nr. 6, S. 413 – 427

G Danksagung

Für die Unterstützung in allen Phasen dieser Arbeit und für die Überlassung dieses interessanten Themas, mit der Bereitschaft alle offenen Fragen gemeinsam zu besprechen, möchte ich mich bei Prof. Dr. Peter Eisert besonders bedanken. Bei Prof. Dr. Beate Meffert bedanke ich mich vielmals für die Übernahme des Zweitgutachtens.
Ferner möchte ich mich bei meinen Korrekturleser/innen Nora, Ursula, Martin und meiner Schwester Sandra herzlich bedanken.
Schließlich danke ich von ganzem Herzen meiner Partnerin Ulrike für den verständnisvollen Beistand und die geduldig aufopfernde Unterstützung bei dieser Arbeit und in allen Lebenslagen.

H Selbständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe. Weiterhin erkläre ich, eine Diplomarbeit in diesem Studiengebiet erstmalig einzureichen.

Berlin, den 10. Juni 2013 .....