HAW HAMBURG

Masterarbeit

Thien Schlodinski

Bildbasierte Rekonstruktion von Gebäuden durch Fitting volumetrischer Körper

Fakultät Technik und Informatik
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

Thien Schlodinski

Bildbasierte Rekonstruktion von Gebäuden durch Fitting volumetrischer Körper

Mastertarbeit eingereicht im Rahmen der Masterprüfung
im Studiengang Master of Science Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Philipp Jenke
Zweitgutachter: Prof. Dr. Ulrike Steffens

Eingereicht am: 18. April 2019

Thien Schlodinski

Thema der Arbeit

Bildbasierte Rekonstruktion von Gebäuden durch Fitting volumetrischer Körper

Stichworte

Bildbasiert, Rekonstruktion, Gebäuderekonstruktion, Gradientenabstiegsverfahren, Simulated Annealing, Textur Mapping

Kurzzusammenfassung

In dieser Arbeit wird ein Annäherungsalgorithmus entwickelt, mithilfe dessen verschiedene Gebäuden anhand eines zweidimensionalen Bildes zu einem dreidimensionalen Objekt rekonstruiert werden. Eine automatische Objekterkennung im Eingabebild ist nicht Teil der Arbeit, da der Fokus auf der genauen Rekonstruktion liegt. Stattdessen sollen die Eckpunkte des Gebäudes durch den Anwender eingegeben werden. Um den Prozess zu vereinfachen, werden die Gebäude in mehrere Einzelteile unterteilt, zum Beispiel wird ein Einfamilienhaus aus einem Quader und einem Prisma zusammengesetzt. Diese Einzelobjekte gibt es in einer Datenbank als Modellobjekte. Aus dieser Datenbank wird anhand der Benutzereingabe die passenden Objektteile für die Rekonstruktion entnommen. Bei der Rekonstruktion werden sowohl der Ort und die Dimension des Modellobjekts rekonstruiert, als auch auch der Ort und Blickwinkel der Kamera, mit der das Objekt des Eingabebildes aufgenommen wurde. Für die Annäherung wird das Gradientenabstiegsverfahren verwendet. Dieses findet, abhängig von den Eigenschaften des Modellobjekts, nicht immer ein optimales Ergebnis. Daher wird es mit einem Simulated Annealing Algorithmus kombiniert, um die Wahrscheinlichkeit eines sehr guten Rekonstruktionsergebnisses zu erhöhen. Außerdem wird in dieser Arbeit ein Algorithmus mit dem Namen „Sticky Objects“ entwickelt, mit der rekonstruierte Einzelobjekte, die nicht exakt auf- oder aneinander liegen, zum Beispiel wenn sie aus einer anderen Perspektive betrachtet im Raum schweben, aneinander haftend gemacht. Es wird ein Prototyp für die Anwendung entwickelt, der bestätigt, dass die hier verwendeten Algorithmen sich für bestimmte Beispiele sehr gut eignen. Ob ein Objekt mit einem sehr guten Ergebnis rekonstruiert werden kann, hängt von seinen Eigenschaften und denen der Modellparameter ab. Es werden zum Schluss dieser Arbeit Erweiterungsmöglichkeiten zusammengetragen, mit denen der Rekonstruktionsprozess optimiert werden kann.

Thien Schlodinski

Title of Thesis

Image Based Reconstruction of Buildings by Fitting of Volumetric Objects

Keywords

Image based, Reconstruction, Building Reconstruction, Gradient Descent, Simulated Annealing, Texture Mapping

Abstract

In this thesis, an approximation algorithm is developed by which different kinds of buildings are reconstructed from a two dimensional picture to a three dimensional object. An automatic object recognition on the input image is not part of this thesis, as the focus is on the exact reconstruction. Instead, the corner points of the building should be entered by the user. To simplify the process, the buildings are divided into several parts, e.g. a single family house is composed of a cuboid and a prism. These individual object parts exist in a database as model objects. Based on the user input, the appropriate object parts for the reconstruction are taken from this database. The algorithm reconstructs the location and dimension of the object as well as the location and angle of the camera used to capture the object of the input image. For the approximation, the gradient descent method is used. Depending on the properties of the model object, the gradient descent method does not always find the optimal result. To increase the likelihood of finding a very good result, it is combined with a simulated annealing algorithm. In addition, there are cases where individually reconstructed object parts look like a whole building in one perspective, but float in space in a different perspective, an algorithm called „Sticky Objects“ is developed to make sure that the individually reconstructed objects stick together. A prototype for the application is implemented to confirm the suitability of the chosen algorithms for certain examples. Whether an object can be reconstructed with a very good result depends a lot on its properties and those of the model object parameters. At the end of this thesis, several extension possibilities for an optimization of the reconstruction process are compiled.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einführung

Diese Arbeit beschäftigt sich mit der beispielbasierten Rekonstruktion von Gebäuden. Bei der Grundidee geht es darum, aus einem einzigen gegebenen zweidimensionalen Bild die Form des Gebäudes zu ermitteln und daraus die dreidimensionale Form darzustellen. Dabei soll das dreidimensionale Ergebnis sich dem Eingabeobjekt so stark wie möglich ähneln, nicht nur bezüglich der Form, sondern auch des Aussehens. Dafür soll aus dem Eingabebild die Flächen des Gebäudes entnommen und als Textur dem Ausgabeobjekt weitergegeben werden. Um die Form des Objekts zu ermitteln, soll im Hintergrund eine Datenbank verwendet werden, die bereits mit Beispielmodellen gefüllt ist. Falls ein Modellobjekt sich dem Eingabeobjekt ähnelt, wird dieses Modell aus der Datenbank entnommen und durch Änderung der Größe, Drehung und Position sich an das Eingabeobjekt angepasst. Das Ziel ist es, dem im Eingabebild sichtbaren Objekt ein, der Form entsprechend, fast identisches Ausgabeobjekt zu rekonstruieren und dieses als Datei abzuspeichern. Dieses kreierte Objekt soll zur weiteren Verarbeitung in gängigen 3D-Bearbeitungsprogrammen geöffnet werden können.

Zunächst werden bereits erforschte Methoden untersucht, die sich mit Gebäuderekonstruktion beschäftigen. Eine Auswahl dieser Theorien wird im nächsten Kapitel vorgestellt. Es gibt eine Vielzahl dieser Forschungen mit teilweise sehr verschiedenen Ansätzen. Aus diesen werden einige Theorien für diese Arbeit ausgewählt und mit eigenen Ideen kombiniert bzw. für diese angepasst. Nach der Erstellung des Konzepts werden die mathematischen Grundlagen passend für die Umsetzung ausgewählt. Wichtig dabei ist die Entscheidung, welche Informationen über das Eingabeobjekt für die Rekonstruktion bekannt sein sollen. Die Algorithmen sollen die Informationen vom Eingabeobjekt bekommen, die für sie verwertbar sind und mit denen sie das Objekt rekonstruieren können. Anschließend sollen die Algorithmen zu einem Prototypen implementiert werden. Der Fokus dieser Arbeit liegt auf dem Weg vom 2D Bild auf das 3D Objekt. Der Prototyp soll daher einfach gestaltet werden, damit die Algorithmen auf ihre Funktionalitäten und Eignung für das Vorhaben geprüft werden können. Zudem sollen so wenig Hilfsmittel wie

möglich benötigt werden. Bei der Implementierung der Methoden ist zu beachten, dass diese auf einfache Art auch auf andere Plattformen übertragbar sind. Mithilfe des Prototypen werden zum Schluss verschiedene Szenarien anhand von realen und mit einem 3D Programm erstellten Beispielbildern die Anwendbarkeit der Algorithmen getestet. Die ausgewählten Algorithmen werden gegebenenfalls durch Erweiterungen optimiert, um bessere Ergebnisse zu erzielen.

Die Herausforderung bei dieser Arbeit ist der Weg von einem 2D Bild zu einem 3D Objekt. Es gibt zu Beginn aus dem Bild keinerlei Informationen über Größe, Tiefen und Begrenzungen des Objekts. Im Laufe des Projekts soll daher festgelegt werden, welche Informationen über das Objekt bekannt sind und woher diese stammen. Dabei ist es denkbar, dass auch der Anwender involviert wird. Es sollen aber so wenig Hilfsmittel wie möglich für die Erfassung des Objekts eingesetzt werden. Durch die oben beschriebenen Idee, dass eine Datenbank mit Beispielmodellen bei der Rekonstruktion helfen soll, müssen die Informationen über das Eingabeobjekt so gewählt werden, dass aus der Datenbank entweder ein eindeutiges Modellobjekt zum Eingabeobjekt passt oder gar keins. Daher wird im Vorwege geklärt, wie 3D Objekte in dieser Arbeit aufgebaut sind, wie sie gespeichert werden und welche Objekte in der Datenbank für die Tests bereit stehen. Es wird eine begrenzte Anzahl an rekonstruierbaren Objekten geben. Die Datenbank wird aber für zukünftige Projekte erweiterbar sein.
In einem 2D Bild sind selbstverständlich immer mehrere Seiten des Gebäudes nicht sichtbar. Wie mit diesen umgegangen wird und wie es möglich ist, die 3D Modellobjekte mit dem 2D Objekt im Bild zu vergleichen, wird im laufe der Arbeit entschieden. Auch können Texturen auf der nicht sichtbaren Seite nicht für das 3D Resultat entnommen werden. Das Hauptaugenmerk in dieser Arbeit ist aber die Rekonstruktion der Form des Objekts, die Texturen bekommen daher keine besondere Aufmerksamkeit und das Projekt wird dafür nicht durch eine höhere Anzahl an Eingabebildern erweitert.

2 Stand der Technik

Es gibt viele Forschungseinrichtungen, die das Thema der Objekt- und Gebäuderekonstruktion untersuchen. Hier werden einige Ideen präsentiert, die in unterschiedliche Richtungen verlaufen.

Bei Hejrati et al. [15] wird eine 3D Modelldatenbank erstellt. Da man ein 2D Bild nicht einfach mit einem 3D Objekt vergleichen kann, wird eine weitere Datenbank erstellt. Diese speichert zweidimensionale Abbildungen der 3D Objekte aus der anderen Datenbank in verschiedenen Perspektiven. Ein Algorithmus zur Objekterkennung wird zunächst auf das Eingabebild angewendet. Mithilfe einer Brute-force Methode wird dann das erkannte Objekt aus dem Eingabebild mit allen zweidimensionalen Abbildungen aus der Datenbank verglichen und das Objekt mit dem ähnlichsten Ergebnis wird zurückgegeben. Zur Objekterkennung wird der HOG-Algorithmus (*histogram of oriented gradients*) verwendet. Das Konzept zum HOG-Algorithmus wurde von Robert K. McConnell 1986 [10] zum ersten Mal vorgestellt, damals wurde er noch nicht „HOG“ genannt. Erst in 2005 wurde der Algorithmus populär, als Dalal und Triggs [22] ihn auf der Conference on Computer Vision and Pattern Recognition (CVPR) vorstellten. Der Schwerpunkt deren Arbeit lag auf der Erkennung von Fußgängern in Bildern.

Einen sehr ähnlichen Ansatz haben Xiao et al. in ihrem Paper „Localizing 3D cuboids in single-view images“ [21] gewählt. Deren Datenbank beinhaltet nur Quader in verschiedenen Größen und Positionen und der Fokus des Konzepts liegt auf der Erkennung von Ecken der Objekte.

Bei der Arbeit von Xue et al. [20] gibt es ebenso eine Datenbank mit 3D Objekten. Beim Eingabeobjekt wird angenommen, dass die Linien bzw. Kanten des Objekts bereits erkannt wurden. Das Objekt wird mit denen in der Datenbank verglichen und das ähnlichste daraus zurückgegeben.

Neben den genannten Arbeiten gibt es auch ganz andere Forschungsansätze zum Thema 3D Rekonstruktion. Die meisten konzentrieren sich auf Bild- und Objekterkennung.

Beispielsweise werden aktuell für die Objektdetektion/-rekonstruktion oder Erstellung von Tiefenkarten RGB-D Sensoren (z.B. die Kinect von Microsoft) eingesetzt, die nicht nur das zweidimensionale Bild aufnehmen, sondern gleichzeitig auch Informationen zur Tiefe der Objekte in einem Raum aufgenommen. Es gibt beispielsweise ein Konzept zur Mustererkennung [5], das RGB-D Sensoren verwendet, womit ein kleiner 3D Ausschnitt (z.B. Ecke eines Tischs) in einem Raum aus mehreren Perspektiven wiedererkannt werden kann. Mit diesem Konzept kann ein solches System für weitere Forschungsprojekte wie die Rekonstruktion von Objekten verwendet werden.

Es gibt viele Forschungsansätze, bei denen die nicht direkt die Tiefe als Information zur Verfügung steht, sondern mehrere zweidimensionale Bilder zur Hilfe genommen werden. Dieses Vorgehen wurde bereits 1906 von Finsterwalder [24] beschrieben und nennt sich „Photogrammetrie“.

In der IEEE Conference on Computer Vision and Pattern Recognition 2016 wurde von einem Forschungsteam um Blaha et al. [17] ein solches Konzept vorgestellt. Dabei werden aus einer Szene eine Reihe von Bildern aus verschiedenen Perspektiven aufgenommen. Mithilfe der Bilder werden die Objekte (Häuserreihen, Straßen, Bäume) rekonstruiert.

Bei Schindler und Bauer [3] werden in den verschiedenen Bildaufnahmen eines Gebäudes korrespondierende Punkte dieses Gebäudes gefunden und als Punktwolke gespeichert. Eine Linien- und Ecken-Erkennung wird auf diese Punktwolke angewendet und mit den Ergebnissen werden Hauptebenen des Gäudes gewonnen. Nach der Erstellung dieser Hauptebenen werden kleinere Vertiefungen und Vorsprünge mit einem Modellierungsverfahren erfasst. Zuletzt werden die erkannten Eigenschaften mit vordefinierten Formvorlagen verfeinert.

Mithilfe von mehreren Bildaufnahmen haben Schönberger und Frahm [4] ebenfalls erfolgreich Gebäude rekonstruiert. Sie verwenden dabei das Prinzip des „Structure from Motion“ (SfM), bei dem mehrere Bilder eines Objekts aus verschiedenen Perspektiven und Seiten aufgenommen wurden. In diesen Bildern werden dann korrespondierende Punkte auf dem Objekt aus den unterschiedlichen Perspektiven gesucht und gefunden. Mit diesen Informationen wird der Ort der Kamera, die die Bilder aufgenommen hat, für jedes aufgenommene Bild bestimmt. Eine Triangulation der Punkte auf den Bildern und die Zusammenfügung dieser Triangulationen aller Bilder ergibt ein vollständiges 3D Objekt.

Structure from Motion wird auch von Debevec et al. [9] in deren hybridem Ansatz zur Gebäuderekonstruktion, in der SfM und ein modellbasierter Algorithmus kombiniert werden. Dabei wird im ersten Schritt eine photogrammetrische Modellierung mithilfe des SfM durchgeführt. Benötigt werden hier wieder mehrere Bilder des Gebäudes aus verschiedenen Perspektiven. Diese Bilder werden wiederverwendet für das Abbilden der Texturen auf das 3D Objekt. Die Texturen werden aus allen gegebenen Bildern extrahiert und auf das 3D Objekt abgebildet, sodass die Ansicht auf das Gebäude aus verschiedenen Perspektiven realistischer erscheint. Im letzten Schritt werden die Details des Objekts, zum Beispiel Backsteine, mittels eines „Model-Based Stereo“ Algorithmus verfeinert. Dabei wird ein Bilderpaar auf das grobe Modell projiziert. Durch den Vergleich der beiden Bilder aus jeweils einer anderen Perspektive kann die Beschaffenheit der Gebäudefassade extrahiert werden.
Mit der photogrammetrischen Modellierung können historische Gebäude rekonstruiert werden, die in der Vergangenheit zerstört wurden und aktuell nicht mehr existieren. Wie bei Wiedemann et al. [2] werden für die Rekonstruktionen historische Fotos, Grundrisszeichnungen und Fundamente der Gebäude aus Archiven verwendet.

Punktwolken von Objekten können außerdem anders erfasst werden, zum Beispiel mit einem Laserscanner. Brenner und Haala [7] haben zum Beispiel 3D-Stadtmodelle automatisiert rekonstruiert, indem sie Punktwolken-Luftaufnahmen eines Laserscanners mit Grundrissdaten eines 2D-Geoinformationssystems kombiniert wurden.

Neben den genannten Wegen gibt es die Möglichkeit, mithilfe von neuronalen Netzwerken 3D Objekte zu rekonstruieren. Zum Beispiel haben Choy et al. [16] in „3D-R2N2: A Unified Approach for Single and Multi-view 3D Object Reconstruction“ ein neuronales Netzwerk entwickelt, das aus einer großen synthetischen Datensammlung von 3D Objekten zunächst ein Mapping von Bildern auf Objekten lernt. Dann kann das Netzwerk ein oder mehrere Eingabebilder aus beliebigen Perspektiven nehmen und anhand des Gelernten die Rekonstruktion des Objekts als Voxelobjekt zurückgeben. Welche Objekte rekonstruiert werden können, hängt von der Datensammlung ab, mit der der Algorithmus lernt. Beispiele von Choy et al. sind unter anderem ein Flugzeug, ein Stuhl und ein Fahrrad.

3 Konzept

3.1 Randbedingungen

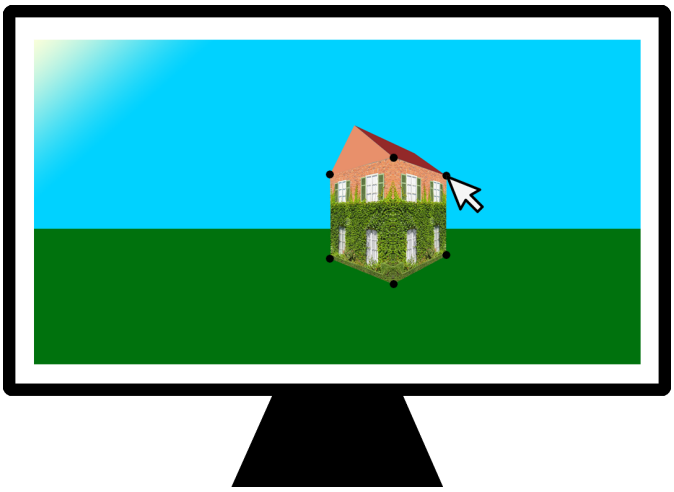

Abbildung 3.1: Der Anwender gibt die Eckpunkte des Gebäudes per Mausklick ein.

Es müssen zu Beginn mehrere Randbedingungen festgelegt werden, um die Ziele aus Kapitel 1 zu erreichen. Als Inspiration wurden die Arbeiten von Hejrati et al. [15] und Xue et al. [20] verwendet. In beiden Konzepten wird mit einer Datenbank gearbeitet, die bereits Modellobjekte enthalten. Wie die Objekte in dieser Arbeit dargestellt sind und in der Datenbank gespeichert werden, wird in Kapitel 3.3 erläutert.

Das Eingabe-Element ist eine zweidimensionale Abbildung eines Gebäudes. Es wird außer der Bilddatei keine dreidimensionalen Eigenschaften wie Tiefeninformationen oder Punktwolken der später entwickelten Software mitgegeben. Für die weiteren Berechnungen werden die Eckpunkte des Gebäudes verwendet. Ähnlich wie bei Xue et al. [20] beinhaltet diese Arbeit keine automatische Objekterkennung. Der Unterschied zu Xue et

al., wo Objektkanten als Orientierung verwendet werden, besteht darin, dass hier Objekt-Eckpunkte für die Rekonstruktion verarbeitet werden. Für die Tests im Prototypen soll der Anwender diese Eckpunkte per Mausklick eingeben (ähnlich wie in Abbildung 3.1). Eine automatische Bild- bzw. Eckpunkterkennung findet hier nicht statt und ist nicht Teil der Arbeit. Stattdessen wird Wert darauf gelegt, das Eingabeobjekt mit seinen Eingabewerten so präzise wie möglich zu rekonstruieren.

Das Gebäude aus der Eingabe wird in mehrere Einzelobjekte aufgeteilt, um die Komplexität des Rekonstruktionsprozesses zu reduzieren. Ein Einfamilienhaus wird zum Beispiel aus einem Quader und einem Prisma zusammengesetzt, die zunächst unabhängig voneinander rekonstruiert und im Anschluss zusammengesetzt werden. In dieser Arbeit werden die Modelle in der Datenbank eingeschränkt auf die Formen Quader, Pyramide und Prisma. Mit diesen Objekten lassen sich viele Häuserarten rekonstruieren.

Damit die Software erkennt, welche Objektart sie verarbeiten soll, sind in der Datenbank die Modelle verschiedener Objektarten vorhanden, die mit der Eingabe verglichen werden. Stimmen die Eigenschaften, zum Beispiel die Anzahl der Eckpunkte, überein, wird dieses passende Objekt aus der Datenbank entnommen und mittels numerischer Annäherung an die Eingabe angepasst. Dabei wird das 3D Modell während der Annäherung bei jeder Bewegung auf die 2D Bildschirmkoordinaten berechnet, um diese mit den 2D Koordinaten der Eingabe zu vergleichen. Diese Berechnung wird in Kapitel 3.6 beschrieben. Der Abstand zwischen den angenäherten 2D Modellpunkten und den 2D Eingabepunkten wird als Orientierung für den Annäherungsalgorithmus verwendet. Es gilt, diesen Abstand bzw. Fehlerwert zu minimieren. Da es sich um ein Minimierungsproblem handelt, wird für die Annäherung des Modellobjekts das Gradientenabstiegsverfahren (siehe Kapitel 3.7) verwendet. Im Laufe des Projekts wird der Rekonstruktionsalgorithmus durch einen Simulated Annealing Algorithmus (Kapitel 3.8) ergänzt, da das Gradientenabstiegsverfahren nicht in jedem Fall das Optimum findet.

Nach einer erfolgreichen Annäherung aller Objektteile werden diese 3D Formen zu einem vollständigen Objekt zusammengesetzt. Ein weiterer Algorithmus, der in dieser Arbeit den Namen „Sticky Objects“ trägt (Kapitel 3.9), sorgt dafür, dass die Objekte nicht verschoben im Raum liegen, sondern fest aneinander haften, wie es bei einem realen Gebäude der Fall ist. Das Ergebnis ist die Ausgabe des rekonstruierten 3D Objekts im OBJ-Format, das sowohl vom Computergrafik Framework der HAW Hamburg [13], als auch von anderen Bildbearbeitungsprogrammen wie Blender oder Adobe Photoshop weiter verarbeitet werden kann.

3.2 Korrespondenzen und Ähnlichkeit von Formen

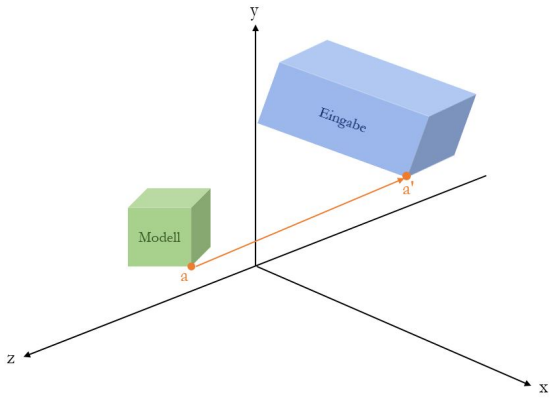

Abbildung 3.2: Beispiel von Korrespondenzen. Modellpunkt a aus der Datenbank korrespondiert mit Punkt a'.

Wie bereits erwähnt, wird in diesem Projekt der Fokus auf die präzise Rekonstruktion der Objekte gelegt. Der Anwender entscheidet durch verschiedene Auswahlfelder, welche Objektformen er rekonstruieren möchte. Er gibt die Eckpunkte der einzelnen Formen nach Anleitung ein und fügt so nach und nach alle Objekte ein, die zusammen ein Gebäude ergeben (z.B. Quader für den Rumpf und Pyramide für das Dach). Um die Eckpunkte aus der Datenbank an die Punkte der Eingabe anzupassen (rekonstruieren), werden sogenannte „korrespondierende" Eckpunkte verwendet. Das sind Punktepaare, bei denen der eine Punkt aus der Eingabe und der andere aus der Datenbank stammt, zum Beispiel die Ecke mit den kleinsten Koordinaten in x-, y- und z-Richtung bei Quadern. Bei der Eingabe der Eckpunkte ist daher die Reihenfolge zu beachten, damit die „richtigen" Punkte miteinander korrespondieren können. Ein Beispiel ist in Abbildung 3.2 zu sehen. Dort ist die Ecke unten rechts vom Modell aus der Datenbank mit der Bezeichnung a versehen. Der korrespondierende Punkt aus dem Eingabeobjekt heißt a'. Um die Punkte voneinander zu unterscheiden, werden im weiteren Verlauf der Arbeit die Eingabepunkte als *Featurepunkt* und die Punkte aus der Datenbank *Modellpunkt* genannt.

3.3 3D Objekte

Es gibt drei Objektformen, die rekonstruiert werden sollen: Quader, Prisma mit dreieckiger Grundfläche und Pyramide. Es gibt bei allen Objekten einen Referenzpunkt c, von dem aus eine Addition oder Subtraktion (von Anteilen) der Breite (b), Höhe (h) oder Tiefe (t) alle Eckpunkte berechnet werden können. Mit diesem Vorgehen werden bestimmte Winkel (z.B. die rechten Winkel beim Quader) während der Rekonstruktion unverändert gehalten. Die Form des δ-Vektors ist in dieser Arbeit bei allen Objekten gleich. Seine Werte werden in der Berechnung optimiert. Der Vektor δ enthält folgende Parameter:

- $c = (c_x, c_y, c_z)^T$, die Koordinaten des Referenzpunktes eines Objekts, in dessen Abhängigkeit die Eckpunkte berechnet werden,
- die Dimensionen des Objekts: b Breite, h Höhe und t Tiefe,
- rotateY, rotateY und rotateZ, die Rotationswinkel um die x-, y- und z-Achse, um das Modellobjekt zu rotieren,
- t_x, t_y und t_z für die Verschiebung des Modellobjekts auf der x-, y- und z-Achse.
- eye_x, eye_y, eye_z sind die Parameter für den Augenpunkt. Es wird nur beim ersten Objekt aus der Objektliste der Augenpunkt berechnet, in Kapitel 3.6.2 wird der Augenpunkt genauer beschrieben.

Damit sieht der δ-Vektor wie folgt aus:

$$\delta = (c_x, c_y, c_z, b, h, t, rotateX, rotateY, rotateZ, eyeX, eyeY, eyeZ) \tag{3.1}$$

3.3.1 Quader

In den folgenden Bildern bestimmen die Großbuchstaben an den Ecken der Objekte die Reihenfolge der Eingabe.

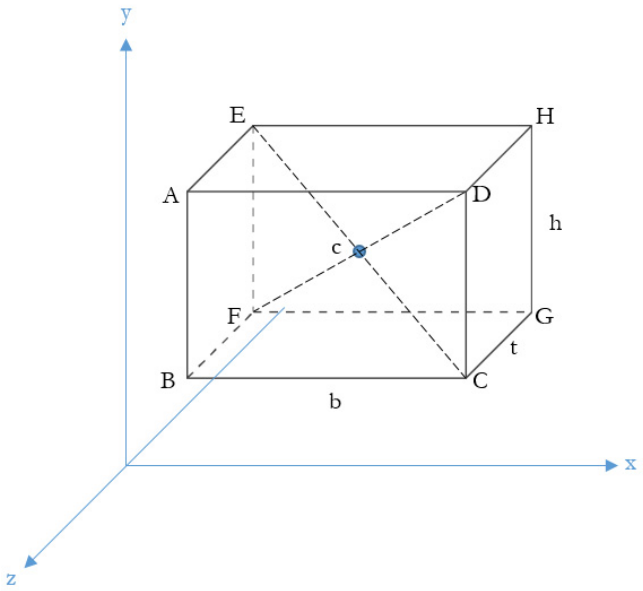

Abbildung 3.3: Modell eines Quaders. Bei diesem Objekt liegt der Referenzpunkt c mittig im Objekt.

Ein Quader braucht mindestens 6 Featurepunkte, um rekonstruiert werden zu können. c ist der Schwerpunkt des Quaders (Abbildung 3.3). Die Eckpunkte A bis G des Quaders werden mit folgender Formel berechnet:

$$\{A, ..., G\} = c + \begin{pmatrix} \pm b/2 \\ \pm h/2 \\ \pm t/2 \end{pmatrix}$$

3.3.2 Prisma

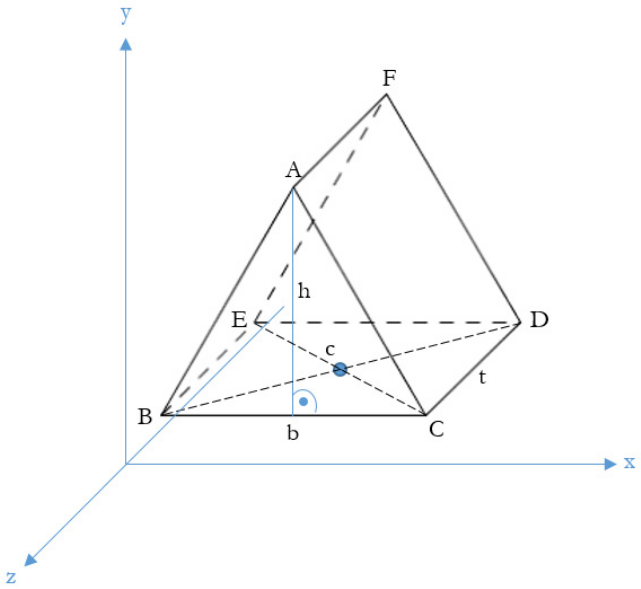

Abbildung 3.4: Modell eines Prismas.

Ein Prisma braucht mindestens 5 Featurepunkte, um rekonstruiert werden zu können. Ausgeschlossen wird der Fall, bei dem weder der Punkt A noch F zu sehen sind. c ist hier der Mittelpunkt der rechteckigen Grundfläche des Prismas (Abb. 3.4).

$$\{A, F\} = c + \begin{pmatrix} 0 \\ h \\ \pm t/2 \end{pmatrix}$$

$$\{B, C, D, E\} = c + \begin{pmatrix} \pm b/2 \\ 0 \\ \pm t/2 \end{pmatrix}$$

3.3.3 Pyramide

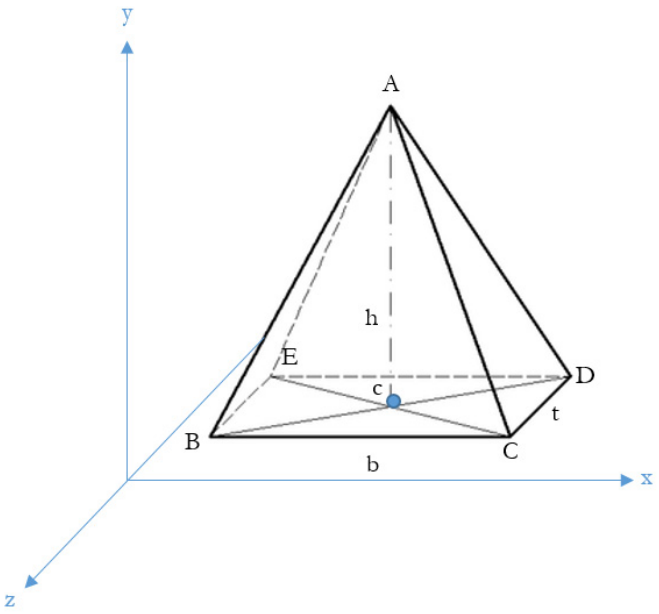

Abbildung 3.5: Modell einer Pyramide.

Eine Pyramide braucht mindestens 4 Featurepunkte, um rekonstruiert werden zu können. c ist hier, wie beim Prisma, der Mittelpunkt der rechteckigen Grundfläche (Abb. 3.5).

$$A = c + \begin{pmatrix} 0 \\ h \\ 0 \end{pmatrix}$$

$$\{B, C, D, E\} = c + \begin{pmatrix} \pm b/2 \\ 0 \\ \pm t/2 \end{pmatrix}$$

3.4 Besondere Fälle

3.4.1 Tiefeninformation

Abbildung 3.6: Links: keine Tiefeninformation des Gebäudes vorhanden, dieses kann nicht rekonsrtuiert werden. Rechts: Tiefeninformation vorhanden.

Für die Rekonstruktion einzelner Objektteile ist es in dieser Arbeit erforderlich, dass die Mindestanzahl an Featurepunkten gegeben sein müssen. Das heißt, ein Haus darf nicht direkt mittig (Bild 3.6, links), sondern muss von der Seite (Bild 3.6, rechts) aufgenommen werden, da ansonsten nicht genügend Punkte für die Eingabe vorhanden sind. Von der Seite gesehen sind die nötigen Tiefeninformationen vorhanden, um das Gebäude zu rekonstruieren.

3.4.2 Verdeckte Punkte

Bei einem Bild eines 3D Objekts gibt es immer Seiten bzw. Eckpunkte, die auf der Rückseite und somit nicht sichtbar sind. Das Ziel in diesem Projekt, ein Gebäude mithilfe eines einzigen Bildes zu rekonstruieren, soll nicht verändert werden. Daher wird dieses Problem gelöst, indem der Anwender bei der Eingabe der einzelnen Objektteile vorher auswählt, um welches Objekt es sich handelt (Quader, Pyramide, Prisma). Aus der Datenbank wird dann bei der Rekonstruktion das passende Modell entnommen und dieses der Eingabe angenähert. Für die verdeckten Seiten werden die Texturen der sichtbaren Seiten übernommen und gespiegelt auf den gegenüberliegenden Seiten dargestellt.

3.5 Mathematisches Konzept im 3-dimensionalen Raum

Nach der Auswahl des Eingabebilds durch den Anwender und der Eingabe der Featurepunkte, entnimmt das Programm das Modellobjekt mit der passenden Anzahl an sichtbaren Eckpunkten aus seiner Datenbank. Mithilfe der numerischen Mathematik sollen die Modellpunkte dann an die Featurepunkte angenähert werden. Dafür wird folgende Formel aufgestellt:

$$x_f = P_p \cdot P_v \cdot x_m. \tag{3.2}$$

x_f ist der Featurepunkt, an den der korrespondierende Modellpunkt x_m angenähert werden soll. P_p ist die Projektionsmatrix für die perspektivische Ansicht des 3D Objekts im 2D Bildschirm. Mit P_v können Rotationen und Translationen am Modell vorgenommen werden. Diese Matrizen werden in Kapitel 3.6.1 genauer erläutert.

Das Ziel ist es, den Abstand zwischen x_f und x_m für jedes Punktepaar zu minimieren:

$$min(||x_f - x_m||) \tag{3.3}$$

3.5.1 Fehler und Fehlerfunktion

Nach der Anwendung der Formel (3.2) bleibt, durch die numerische Annäherung, ein kleiner Fehler zurück, der durch den Betrag der Subtraktion von x_f und x_m beschrieben wird:

$$E = ||x_f - x_m||. \tag{3.4}$$

Um ein gesamtes Objekt zu bewerten, das mehrere Korrespondenzen hat ($i = 1..n$ mit $n =$ Anzahl der Featurepunkte), muss der quadratische Fehler über alle Korrespondenzen betrachtet werden. Die Quadrierung sorgt dafür, dass negative Werte nicht den Fehler verringern.

$$f(\delta) = \sum_{i=1}^{n} (x_f^i - P_p(\delta) \cdot P_v(\delta) \cdot x_m^i(\delta))^2. \tag{3.5}$$

P_p, P_v und x_m^i sind abhängig vom Vektor δ. Dieser enthält, wie in Formel (3.1) beschrieben, die Objekt- und Kameraparameter. Gesucht wird ein δ, mit dem die Fehlerfunktion $f(\delta)$ minimal wird.

3.6 Projektion auf den 2D Monitor

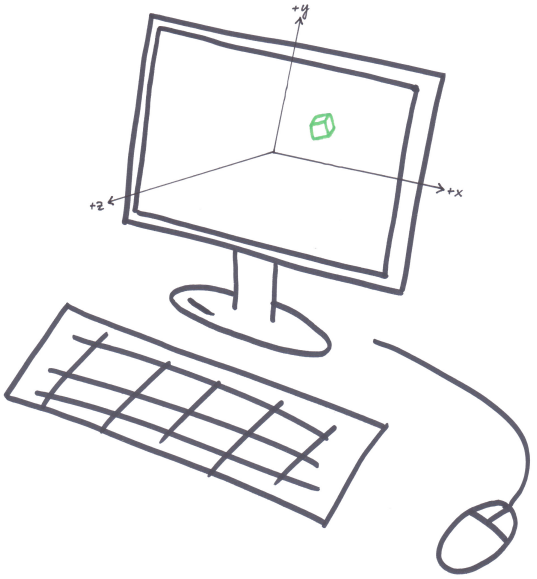

Abbildung 3.7: Die Objekte im 3D Koordinatensystem müssen auf den 2D Bildschirm angezeigt werden.

Während und nach dem Rekonstruktionsvorgang liegen die Eckpunkte des Modellobjekts im dreidimensionalen Raum vor. Um diese Punkte mit den Featurepunkten zu vergleichen und im Anschluss der Rekonstruktion auf den zweidimensionalen Bildschirmmonitor abbilden zu können, muss das 3D Objekt auf die 2D Fläche des Monitors projiziert werden (Abb. 3.7). Dabei werden die 3D Objektkoordinaten (x, y, z) zu 2D Bildkoordinaten (x, y) berechnet. In diesem Projekt müssen die 2D-Bildkoordinaten bei jedem neuen Rechenschritt erstellt werden, um die Abweichung der Modellpunkte zu den Featurepunkten im 2D Bild berechnen zu können. Die Matrizen zur Berechnung der Projektion auf den 2D Monitor sind in den Formeln (3.2) und (3.5) in der Matrix P_p zusammengefasst. Die folgenden Formeln und Grafiken sind angelehnt an die Arbeiten von Jenke [14] und Nischwitz et al. [18].

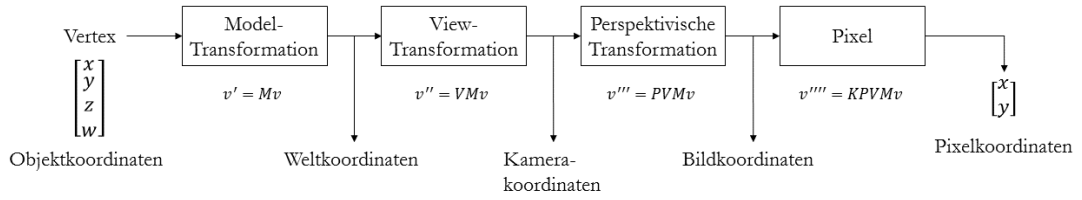

Abbildung 3.8: Pipeline der Transformationen, um von einem Koordinatenpunkt zum Pixelkoordinaten für die Darstellung auf dem Bildschirmmonitor zu kommen.

Für die Projekttion sind mehrere Schritte notwendig (Abb. 3.8), die im folgenden Abschnitten beschrieben werden.

3.6.1 Model-Transformation

Ein dreidimensionales Objekt hat zum Beispiel an einer Ecke die Objektkoordinaten $v = (x, y, z)$. Diese lokale Koordinate gehört nur zu diesem Objekt. Das Objekt liegt (zum Beispiel zusammen mit weiteren Objekten) einer Szene in einem Weltkoordinatensystem. Dort liegt es (je nachdem wie der Modellierer das Objekt definiert) meist um den Ursprung herum. Um dem Objekt die passenden Weltkoordinaten zu geben, muss es transformiert werden. Dazu gehören Rotation, Translation und Skalierung dazu. Für die Transformation von Objekt- in Weltkoordinaten wird der Punkt v mit der sogenannten Model-Matrix multipliziert:

$$v' = P_v \cdot v \tag{3.6}$$

In diesem Projekt sind nur Rotationen und Translationen relevant, da die Skalierung durch die Berechnung der Dimensionen des Objekts bereits flexibel eingebaut wurde. Die Matrix P_v aus der Formel (3.6) wurde bereits in der Formel (3.2) erwähnt und ist eine Multiplikation aus den Rotationen (R^x, R^y, R^z) und der Translation (T). Die Reihenfolge bei der Multiplikation muss dabei beachtet werden, denn das Ergebnis fällt unterschiedlich aus, je nachdem ob zuerst rotiert oder verschoben wird. Die Matrizen für die Transformationen sehen wie folgt aus:

Rotation um die x-Achse mit einem Winkel von α-Grad

$$R^x = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(\alpha) & -sin(\alpha) & 0 \\ 0 & sin(\alpha) & cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotation um die y-Achse mit einem Winkel von β-Grad

$$R^y = \begin{pmatrix} cos(\beta) & 0 & sin(\beta) & 0 \\ 0 & 1 & 0 & 0 \\ -sin(\beta) & 0 & cos(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotation um die z-Achse mit einem Winkel von γ-Grad

$$R^z = \begin{pmatrix} cos(\gamma) & -sin(\gamma) & 0 & 0 \\ sin(\gamma) & cos(\gamma) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Translation in Richtung der x-, y- und z-Achse (T_x, T_y, T_z)

$$T = \begin{pmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Im folgenden Beispiel wird auf den Punkt $v = (2, 1, 0, 1)^T$ eine Verschiebung von $T = (T_x, T_y, T_z, w) = (2.5, 1.5, 0.5, 1)$ und anschließend eine Rotation von 60° um die y-Achse vorgenommen:

Translation:

$$v' = \begin{pmatrix} 2.0 \\ 1.0 \\ 0.0 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 2.5 \\ 0 & 1 & 0 & 1.5 \\ 0 & 0 & 1 & 0.5 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 4.5 \\ 2.5 \\ 0.5 \\ 1 \end{pmatrix}$$

Rotation:

$$v' = \begin{pmatrix} 4.5 \\ 2.5 \\ 0.5 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} cos(60) & 0 & sin(60) & 0 \\ 0 & 1 & 0 & 0 \\ -sin(60) & 0 & cos(60) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 2.683 \\ 2.5 \\ -3.647 \\ 1 \end{pmatrix}$$

3.6.2 View-Transformation

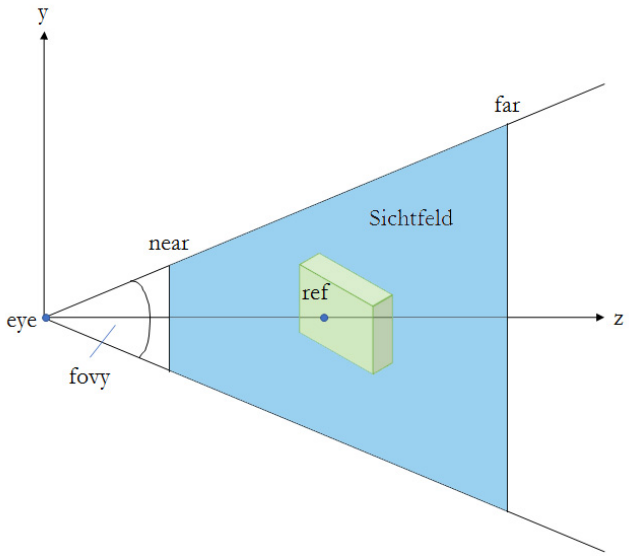

Abbildung 3.9: Der Augenpunkt **eye** schaut auf den Referenzpunkt **ref**. Das Sichtfeld, das in Richtung der y-Achse den Winkel **fovy** hat, wird durch eine **near** und **far** Ebene zugeschnitten. Alle Objekte, die darüber hinaus liegen, werden nicht auf den Bildschirm projiziert.

Um einen Bildausschnitt aus dem Weltkoordinatensystem zu bekommen, schaut eine virtuelle Kamera auf die Szene (Abbildung 3.9). Mehrere Eigenschaften dieser Kamera müssen nun festgelegt werden, damit ein Bildausschnitt aus dem 3D Weltkoordinatensystem auf dem 2D Bildschirmmonitor dargestellt werden kann. Der Ursprung der Kamera wird Augenpunkt (**eye**) genannt. Für die weiteren Berechnungen wird ein Referenzpunkt (**ref**) ausgewählt, der auf der Winkelhalbierenden des vertikalen Kameraöffnungswinkels liegt, sowie der Oben-Vektor (**up**, hier in Richung der y-Achse) festgelegt.

Beispiel:

$$eye = \begin{pmatrix} 0.0 \\ 2.0 \\ 10.0 \end{pmatrix}$$
$$ref = \begin{pmatrix} 0.0 \\ 2.0 \\ 50.0 \end{pmatrix}$$
$$up = \begin{pmatrix} 0.0 \\ 1.0 \\ 0.0 \end{pmatrix}$$

Die Koordinaten der Achsenrichtungen lassen sich wie folgt berechnen:

$$z = \frac{ref - eye}{\|ref - eye\|}$$
$$x = up \times z$$
$$y = z \times x$$

Mit diesen Koordinaten wird die Transformationsmatrix V erstellt, wobei die Vektoren x, y und z in die Spalten geschrieben werden:

$$V = \begin{pmatrix} x & y & z & eye \\ 0 & 0 & 0 & 1 \end{pmatrix}^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 10 \\ 0 & 0 & 0 & 1 \end{pmatrix}^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -2 \\ 0 & 0 & 1 & -10 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Daraus ergibt sich für die View-Transformation:

$$v'' = V \cdot P_v \cdot v$$

Und das Beispiel weiterführend:

$$v'' = \begin{pmatrix} 2.683 \\ 0.5 \\ -12.647 \\ 1.0 \end{pmatrix}$$

3.6.3 Perspektivische Transformation

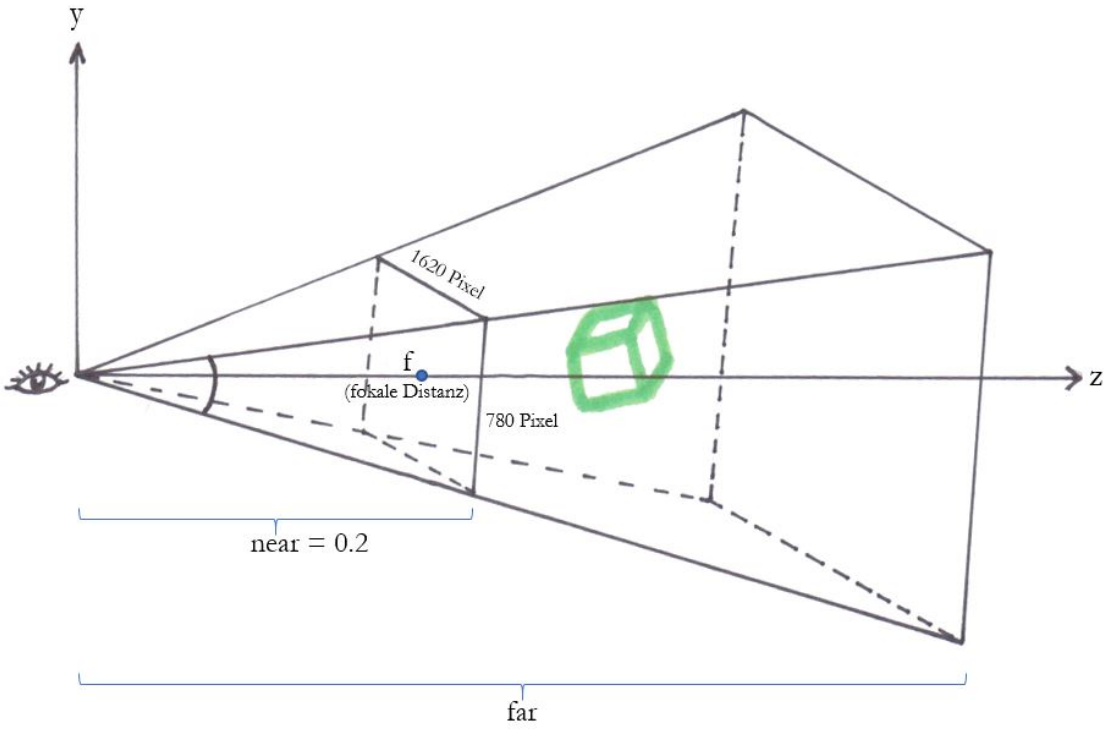

Abbildung 3.10: Sichtfeld einer virtuellen Kamera als Pyramide dargestellt. Der Bereich des Pyramidenstumpfs zwischen **near** und **far** wird später auf dem zweidimensionalen Bildschirm zu sehen sein. Alles darüber hinaus wird nicht abgebildet.

Bei der perspektivischen Transformation werden alle Objekte, die sich innerhalb des Pyramidenstumpfs befinden (siehe Abbildung 3.10), auf die Bildebene projiziert. Die Bildebene ist dabei auf der **near**-Distanz. Objekte, die näher am Augenpunkt sind, erscheinen größer als Objekte, die sich näher an der Ebene in **far**-Distanz befinden. Objekte, die

sich nicht in dem Pyramidenstumpf befinden, werden später auf dem Bildschirm nicht zu sehen sein. Für die Berechnung wird folgende Zentralprojektionsmatrix P verwendet:

$$P = \begin{pmatrix} z_{near} & 0 & 0 & 0 \\ 0 & z_{near} & 0 & 0 \\ 0 & 0 & z_{near} & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Angewendet auf das Beispiel ergibt sich:

$$P = \begin{pmatrix} 0.2 & 0 & 0 & 0 \\ 0 & 0.2 & 0 & 0 \\ 0 & 0 & 0.2 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \cdot v_a''' = \begin{pmatrix} 0.537 \\ 0.1 \\ -2.729 \\ -13.647 \end{pmatrix}$$

Im Anschluss muss das Ergebnis durch den inversen Streckungsfaktor w, also die vierte Koordinate des Vektors, geteilt werden, um von den homogenen 4D-Koordinaten zu den euklidischen 3D-Koordinaten zu kommen:

$$v_b''' = \begin{pmatrix} -0.039 \\ -0.007 \\ 0.2 \\ 1 \end{pmatrix}$$

3.6.4 Pixelkoordinaten

Zuletzt müssen aus den euklidischen 3D-Koordinaten die zweidimensionalen Bildschirmkoordinaten berechnet werden. Es wird zunächst die fokale Distanz $f = f_y = f_x$ ermittelt, also der Abstand zwischen Augenpunkt und dem Mittelpunkt des Bildschirms (siehe Bild 3.10). Dabei wird die Breite b und Höhe h in Pixelwerten und der Öffnungswinkel in y-Richtung $fovy$ benötigt.

Die fokale Distanz lässt sich daraufhin berechnen:

$$f_y = \frac{h}{2 \cdot tan(\frac{fovy}{2})}$$

Falls nur der Öffnungswinkel in x-Richtung $fovx$ bekannt ist, lässt sich die fokale Distanz auch damit berechnen:

$$f_x = \frac{b}{2 \cdot tan(\frac{fovx}{2})}$$

Daraus ergibt sich für die Berechnung der Pixelkoordinaten folgende Matrix K mit der Bildschirmbreite b und -höhe h in Pixeln:

$$K = \begin{pmatrix} f & 0 & 0 & b/2 \\ 0 & f & 0 & h/2 \end{pmatrix}$$

Da in diesem Projekt der Anzeigebereich kleiner ist als der Bildschirm, werden für die Breite $b = 1620$ Pixel und für die Höhe $h = 780$ Pixel verwendet. Die gewählte virtuelle Kamera hat einen Öffnungswinkel in y-Richtung von **fovy** $= 49°$. Der Punkt v hat also auf der Bildebene folgende Koordinaten:

$$p_{Pixel} = K \cdot v_b''' = \begin{pmatrix} 1777.383 & 0 & 0 & 1620/2 \\ 0 & 1777.383 & 0 & 780/2 \end{pmatrix} \cdot \begin{pmatrix} -0.039 \\ -0.007 \\ 0.2 \\ 1 \end{pmatrix} = \begin{pmatrix} 740.114 \\ 376.976 \end{pmatrix}$$

Zusammengefasst ergibt sich daraus für die Matrix P_p der perspektivischen Transformation aus der Formel (3.2) in Abschnitt 3.5 folgende Formel:

$$P_p = V \cdot P \cdot K. \tag{3.7}$$

3.7 Gradientenabstiegsverfahren

Beim Gradientenabstiegsverfahren wird folgendes Problem betrachtet:

$$\min_{x \in \mathbb{R}^n} f(x) \tag{3.8}$$

Es wird ein Parametervektor x gesucht, für den die Funktion f(x) einen minimalen Wert, optimalerweise das globale Minimum innerhalb des Definitionsbereiches von x, annimmt.

Algorithm 1: Algorithmus des Gradientenabstiegsverfahrens in Anlehnung an [25].

Data: Zielfunktion f, Abbruchbedingung $\epsilon > 0$.

Result: Lokales Minimum x_* von f.

```
Wähle x ∈ ℝ^n;
while ||∇f(x)|| > ε do
    Abstiegsrichtung d := −∇f(x);
    Wähle Schrittgröße λ > 0 mit f(x + λd) < f(x);
    x := x + λd;
end
return x;
```

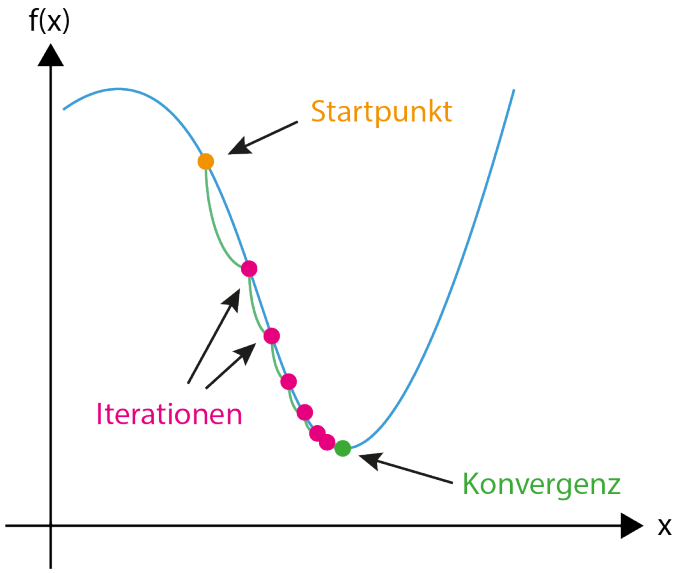

Abbildung 3.11: Veranschaulichung des Gradientenabstiegsverfahrens anhand eines Beispiels im 2D Raum.

Der Algorithmus des Gradientenabstiegsverfahrens ist im Algorithmus 1 dargestellt, Abbildung 3.11 enthält noch ein Beispiel im 2D Koordinatensystem zur Veranschaulichung.

Als Parameter bekommt der Algorithmus die Zielfunktion f, die minimiert werden soll, und als Abbruchbedingung $\epsilon > 0$. Bei der Initialisierung wird ein Startwert $x \in \mathbb{R}^n$ gewählt. Dann wird der Gradient von f an der Stelle x, $\nabla f(x)$, berechnet. Der Gradient $\nabla f(x)$ ist die partielle Ableitung der Funktion f nach der Variable x und zeigt somit an der Stelle x die Steigung an. Die Abstiegsrichtung d, um an das Minimum zu gelangen, ist also der negative Gradient von $f(x)$. Als nächstes wird die Schrittgröße λ gewählt. Für die Wahl eines dynamischen λ gibt es verschiedene Möglichkeiten [23]. Man kann aber auch eine konstante Schrittgröße wählen, wie es im weiteren Verlauf des Projekts gemacht wird. Als letztes wird das aktuelle $x = x + \lambda d$ gesetzt für die nächste Runde der While-Schleife. Sobald die Bedingung $||\nabla f(x)|| \leq \epsilon$ erfüllt ist, bricht die Schleife ab und gibt den zuletzt berechneten x-Wert zurück. Diese Stelle ist das approximierte lokale Minimum von $f(x)$.

3.7.1 Angepasstes Gradientenabstiegsverfahren

Algorithm 2: Algorithmus des angepassten Gradientenabstiegsverfahren in Anlehnung an Algorithmus 1.

Data: Zielfunktion f, Abbruchbedingung $\epsilon > 0$, Schrittgröße $\lambda > 0$.

Result: Lokales Minimum δ_{i+1} von f.

1 Wähle $\delta_i \in \mathbb{R}^n$, woraus sich der Modellpunkt x_m berechnen lässt. **while** $||E(\delta_i) - E(\delta_{i+1})|| > \epsilon$ **do**

2 Abstiegsrichtung $d := -\nabla f(\delta_i)$;

3 $\delta_{i+1} := \delta_i + \lambda d$;

4 Berechne neues x_f aus δ_{i+1};

5 **end**

6 **return** δ_{i+1};

Für die Optimierung der Formel (3.5) wird das oben beschriebene Gradientenabstiegsverfahren verwendet und an das dieser Arbeit zugrunde liegende Problem angepasst (siehe Algorithmus 2). Das Minimum ist für den Algorithmus des Gradientenabstiegsverfahrens in diesem Fall der kleinste Fehlerwert, den die Formel (3.5) erzeugen kann.

Die Berechnung für δ_{i+1} wird so lange ausgeführt, bis δ_{i+1} und δ_i nahezu identisch sind und somit das Minimum hinreichend angenähert wurde. Anders als im originalen Algorithmus 1 wird hier nicht der Gradient $\nabla f(\delta)$ als Abbruchbedingung verwendet, sondern die Änderungsrate des Fehlerwerts $||E(\delta_i) - E(\delta_{i+1})|| \leq \epsilon$, da die Featurepunkte später durch die manuelle Eingabe des Anwenders entstehen und die Abbruchbedingung mit einem gewünschten Fehlerwert bei Ausreißerpunkten aus der Eingabe eventuell nicht erreicht werden kann.

Beim angepassten Gradientenabstiegsverfahren für diese Arbeit kommen die Anfangswerte von δ aus einer Datenbank. Die Wahl der Schrittgröße und welchen Einfluss diese auf das Ergebnis hat, wird später in Abschnitt 6.3 genauer erläutert. Der Gradient wird mittels Differenzenquotienten für ein kleines h (z.B. 10^{-5}) ermittelt:

$$\nabla f(\delta) = \frac{f(\delta + h) - f(\delta)}{h}. \tag{3.9}$$

3.8 Simulated Annealing

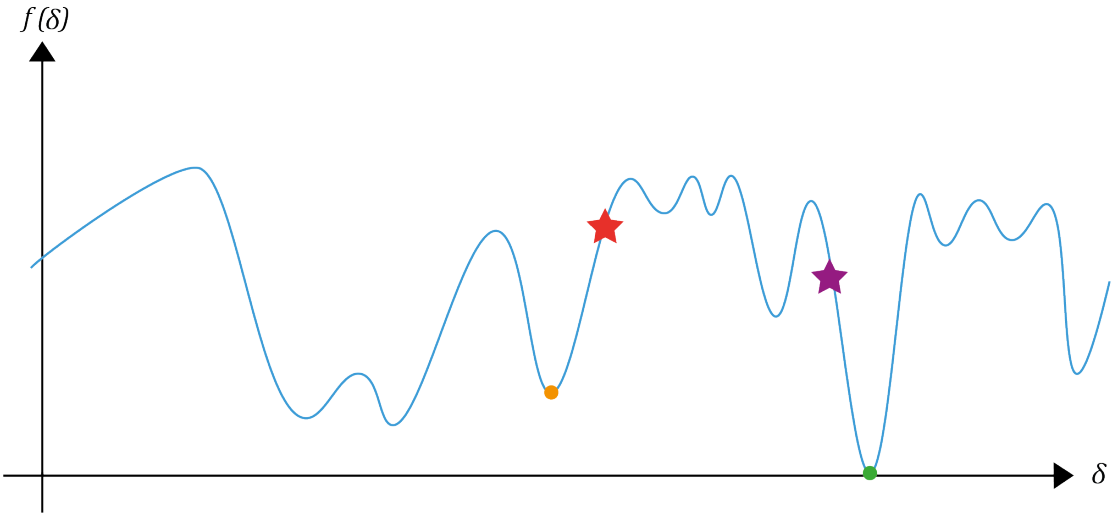

Abbildung 3.12: Beispielhafte Darstellung eines $f(\delta)$ Funktionsgraphen. Beim Start am roten Stern ist das Ergebnis das lokale Minimum beim orangenen Punkt. Der Start am lila Stern lässt das globale Optimum beim grünen Punkt finden.

Das Gradientenabstiegsverfahren findet im besten Fall das globale Optimum, kann aber auch nur ein lokales Minimum finden. In Bild 3.12 ist eine beispielhafte Darstellung einer $f(\delta)$ Kurve zu sehen. Es kann bei einer Lösungsfindung mithilfe des Gradientenabstiegsverfahrens das lokale Minimum an dem orangenen Punkt gefunden werden, wenn die Startwerte von δ an dem roten Stern ausgewählt worden sind. Es ist kein schlechtes Ergebnis, aber optimaler wäre die Lösung bei dem grünen Punkt. Dafür müssen die Startwerte für den nächsten Durchlauf des Gradientenabstiegsverfahren sich aber verändert werden, um beispielsweise im Idealfall an dem lila Stern zu beginnen und dann das Optimum an dem grünen Punkt zu finden.

Um nach einer Rekonstruktion bzw. einer abgeschlossenen Annäherung nach weiteren Minima wie in dem gerade beschriebenen Beispiel zu suchen, wird ein heuristisches Approximationsverfahren mit dem Namen Simulated Annealing (SA) angewendet (siehe Du et al. [19]).

Annealing, auf Deutsch „Abkühlung“, bezieht sich auf eine Analogie aus der Thermodynamik, bei der Stoffe wie Metall oder Glas kurz über ihren Schmelzpunkt erhitzt und anschließend langsam abgekühlt werden. Die Moleküle des Materials begeben sich während dieser Abkühlungsphase in eine perfekte Anordnung und das Material verfestigt sich zu einer optimalen Kristallstruktur. Bei einem Optimierungsproblem kann man das physische Material mit einer Problemlösung vergleichen, die Temperatur mit dem Prozess der Annäherung und das Abkühlen mit mit den Ergebnissen der mehrmaligen Annäherung während des SA Prozesses.

Ausgehend von einer Startlösung wird ein Zufallswert addiert und dann wird die neue Lösung mit der alten verglichen. Dabei kann der neue Zustand sowohl besser als auch schlechter sein als der vorherige. Falls die neue Lösung den Zustand sich verbessert hat, dann soll mit dieser weiter gerechnet werden. Ansonsten wird der Zustand zusammen mit seiner Wahrscheinlichkeit abgespeichert und die alte Lösung wird erneut, mit neuen Zufallswerten addiert, weiter verwendet. Nach jedem Durchlauf verringert sich die Temperatur T, die zu Beginn des Algorithmus mit einem hohen Wert initialisiert wird.

Algorithm 3: Simulated Annealing Algorithmus.

1. 1. Randomisiere $x(0)$.
2. 2. Initialisiere T mit einem großen Wert.
3. 3. **while** $T > 0$ **do**
4. i. Addiere zur Lösung x einen Zufallswert: $x_{new} = x + \Delta x$;
5. ii. Evaluiere $\Delta E(x) = E(x + \Delta x) - E(x)$;
6. **if** $\Delta E(x) < 0$ **then**
7. Behalte den aktuellen Wert x;
8. **else**
9. speichere den neuen Wert x_{new} mit der Wahrscheinlichkeit $P = e^{-\frac{\Delta E}{T}}$ ab.
10. **end**
11. **end**
12. Setze $T = T - \Delta T$;

Das Simulated Annealing garantiert nicht, dass das globale Optimum gefunden wird. Es ist mit diesem Algorithmus aber möglich, ein besseres lokales Minimum zu finden.

3.8.1 Angepasstes Simulated Annealing

Wie beim Gradientenabstiegsverfahren wird auch der SA Algorithmus für das Problem der Objektrekonstruktion angepasst. Vom originalen SA Algorithmus übernommen, werden die Werte von δ aus der ersten Rekonstruktion eines Objektes beim Start des angepassten SA Algorithmus neu gewürfelt. Mit Zufallszahlen $\Delta\delta$, die auf den bei der ersten Optimierung berechneten δ-Werten addiert werden, wird dafür gesorgt, dass das Optimierungsverfahren aus einem lokalen Minimum heraus kommt und die Suche nach dem Optimum weiter erfolgen kann. Diese Zufallszahlen richten sich nach den zuvor im δ-Vektor berechneten Werten: die Intervallgrenzen dieser Zufallszahlen werden festgelegt, indem ein hoher Prozentsatz auf die δ-Werte multipliziert werden. So wird sichergestellt, dass nicht zu kleine Werte auf große Zahlen addiert werden. Ansonsten würde das Gradientenabstiegsverfahren immer wieder in dasselbe lokale Minimum zurückkehren.

Algorithm 4: Simulated Annealing Algorithmus, angepasst an diese Arbeit.

1. 1. Randomisiere $\delta(0)$.
2. 2. Initialisiere T mit einem großen Wert.
3. 3. **while** $T > 0$ *oder* $||E(\delta_i) - E(\delta_{i+1})|| > \epsilon$ **do**
4. Neues $\delta_* = \delta + \Delta\delta$;
5. Rekonstruktionsalgorithmus aus 2 erneut mit δ_* als Startwert ausführen;
6. Evaluiere $\Delta E(\delta) = E(\delta + \Delta\delta) - E(\delta)$;
7. **if** $\Delta E(\delta) < 0$*: Das Ergebnis hat sich verbessert.* **then**
8. Behalte das neue Ergebnis und mache damit weiter;
9. **else**
10. Speichere das Ergebnis ab mit der Wahrscheinlichkeit $P = e^{-\frac{\Delta E}{T}}$;
11. **end**
12. Setze $T = T - \Delta T$.;
13. **end**
14. **return** δ_* mit der größten Wahrscheinlichkeit P.;

Nachdem auf die erste δ-Lösung für jeden Wert des Vektors eine Zufallszahl addiert wurde, wird der Rekonstruktionsalgorithmus erneut ausgeführt. Wie in Algorithmus 3 bereits erwähnt, bekommt der Kontrollparameter T, also die Temperatur, einen hohen Startwert. Der Rekonstruktionsalgorithmus läuft erneut mit den neuen Startwerten durch und danach wird geprüft, ob die neue Lösung besser oder schlechter ist als die vorherige Lösung. Wenn das Ergebnis sich gebessert hat, soll dieses als Startwert für den nächsten

Durchlauf des Rekonstruktionsalgorithmus weiter verwendet werden. Ansonsten wird das Ergebnis mit einer Wahrscheinlichkeit P abgespeichert.

Nach jedem Durchlauf verringert sich die Temperatur T. Es wird der Temperaturabstieg nach Geman und Geman [6] verwendet. Dieser basiert auf eine Markov-Ketten-Analyse des SA-Prozesses und sieht wie folgt aus:

$$T(t) \geq \frac{T_0}{ln(1+t)} \ , t = 1, 2, 3, ...$$

Der Algorithmus läuft, solange die Temperatur T noch nicht bei 0 angekommen ist, und, wie in Algorithmus 2, wenn die Änderungsrate $||E(\delta_i) - E(\delta_{i+1})|| > \epsilon$ noch nicht erreicht wurde. Der Rückgabewert ist, falls der Algorithmus durch $T \leq 0$ abgebrochen wurde, das beste Ergebnis aus allen Ergebnissen während der Berechnung. Falls der Algorithmus durch die Änderungsrate des Gradientenabstiegsverfahrens $||E(\delta_i) - E(\delta_{i+1})|| \leq \epsilon$ abgebrochen wurde, wird das zuletzt berechnete Ergebnis zurückgegeben.

3.9 Optimierung mehrerer Objekte: „Sticky Objects“

Wie in Kapitel 3.5 beschrieben, nimmt sich der Rekonstruktionsprozess das Ergebnis, das vom Gradientenabstiegsverfahren als kleinsten Fehlerwert zurück gegeben wird. Dabei werden die realen Maßstäbe der Objekte nicht berücksichtigt, da sie unbekannt sind. Es kann vorkommen, dass nach der Rekonstruktion von mehreren Objekten Das Endergebnis aus der 2D-Perspektive für „richtig“ wahrgenommen wird. Wenn man das Gesamtobjekt aber in der 3D Perspektive betrachtet und rotiert, fällt auf, dass die Objekte versetzt liegen. Für das weitere Vorgehen, um dieses Problem zu lösen, wird angenommen, dass einzelne Objekte bei einem Gebäude weder übereinander schweben noch mit Abstand nebeneinander liegen, sondern immer symmetrisch aneinander haften müssen.

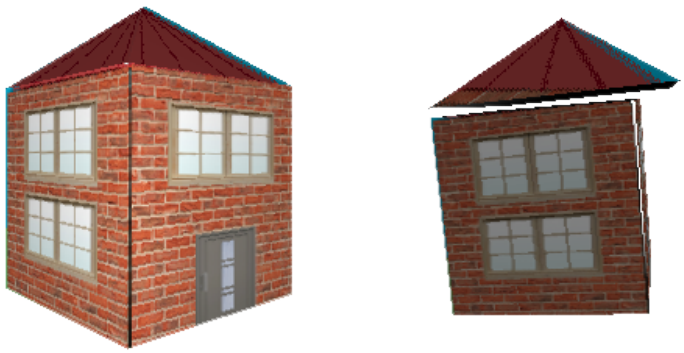

Abbildung 3.13: Beispiel für den Gebrauch von „Sticky“. Links: Objekt nach der Rekonstruktion aus der Ergebnis-Perspektive, Rechts: Linkes Objekt gedreht.

Wie in Bild 3.13 (links) zu sehen, scheint die Rekonstruktion des Hauses gut gelungen zu sein. Erst nach der Drehung des Ergebnisses (3.13 (rechts)) im dreidimesionalen Raum wird sichtbar, dass das Dach nicht direkt auf dem Rumpf des Hauses liegt. Das liegt daran, dass die Dimensionen des rekonstruierten Rumpfs kleiner ist als die des Daches und durch die Verschiebung nach hinten bzw. vorne sieht es so aus als ob die beiden Objekte in der 2D Perspektive aufeinander liegen würden.

Um dieses Problem zu beheben, wird mithilfe vom „Sticky Objects“-Algorithmus das Dach symmetrisch direkt auf den Hausrumpf gelegt und seine Dimensionen an diesen angepasst, sodass beide Teile aneinander haften (Engl. *stick*) und keine Verschiebung mehr vorliegt.

3.9.1 Differenz zwischen zwei Ebenen

Für den „Sticky Objects“-Algorithmus wird angenommen, dass Objekte, die direkt neben- oder aufeinander liegen, zusammen ein symmetrisches Objektkomplex bilden. Es werden nacheinander Objekte verglichen. Wenn zum Beispiel ein Objekt eine Fläche besitzt, zu der ein anderes Objekt eine parallele Fläche aufweist und gleichzeitig kein anderes Objekt und keine andere Fläche zwischen ihnen liegt, gelten sie als direkte Nachbarn und werden mit dem „Sticky Objects“-Algorithmus an den beiden parallelen Flächen fest aneinander gehaftet. Der Algorithmus prüft nach der Rekonstruktion aller Featureobjekte, ob es Ebenen gibt, die parallel zueinander liegen. Mathematisch müssen dabei die Winkel

zwischen den Normalen der beiden Flächen entweder 0° oder 180° sein. Mit der Eingabe der Punkte durch den Anwender ist keine exakte Genauigkeit der Eckpunkte gegeben, weshalb hier für die technische Definition zweier paralleler Ebenen eine Toleranz von +/− 10° akzeptiert wird.

Bei dem Beispiel in Bild 3.13 (rechts) sind die obere Fläche des Hausrumpfes und die untere Fläche des Dachs zueinander parallel. Allerdings ist auch die untere Fläche des Hausrumpfes zur unteren Fläche des Dachs zueinander parallel. Damit das Dach sich bei der erneuten Rekonstruktion sich nicht an den Boden des Hausrumpfes anpasst, wird nach parallelen Flächen der Objekte gesucht, die den geringsten Abstand zueinander haben. Das sind hier die obere Fläche des Hausrumpfes und die Unterseite des Dachs. Wie diese beiden Ebenen gefunden werden, wird im Folgenden erläutert.

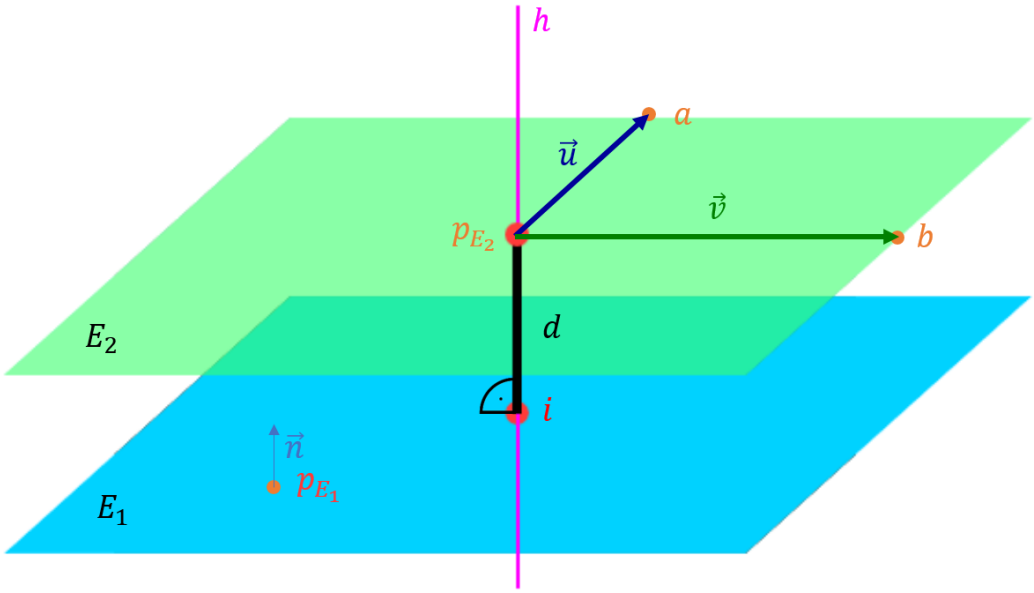

Abbildung 3.14: Berechnung des Abstandes d zweier Ebenen.

Die Abbildung 3.14 veranschaulicht das folgende Vorgehen für die Berechnung des Abstandes zwischen zwei Ebenen.

Eine erste Ebene E_1 wird wie folgt in der Normalenform beschrieben:

$$E_1 : \quad (\vec{x} - \overrightarrow{p_{E_1}}) \cdot \vec{n} = 0$$

Der Punkt $\overrightarrow{p_{E_1}}$ befindet sich an einer beliebigen Stelle auf der Ebene E_1. Bei $\vec{n}$ handelt es sich um eine Normale auf dieser Ebene.

Die zweite Ebene E_2 wird durch drei Punkte beschrieben („Parameterform“):

$$\begin{aligned} E_2: \quad \vec{x} &= \overrightarrow{p_{E_2}} + r \cdot \vec{u} + s \cdot \vec{v} \\ \vec{u} &= \overrightarrow{p_{E_2}a} = \vec{a} - \overrightarrow{p_{E_2}} \\ \vec{v} &= \overrightarrow{p_{E_2}b} = \vec{b} - \overrightarrow{p_{E_2}} \end{aligned}$$

Die Vektoren $\overrightarrow{p_{E_2}}$, $\vec{a}$ und $\vec{b}$ sind beliebige Punkte auf der Ebene. Es werden hier drei Eckpunkte auf der Fläche des Objekts genommen. Die Richtungsvektoren $\vec{u}$ und $\vec{v}$, mit denen die Ebene E_2 aufgespannt wird, werden aus den Abständen zwischen dem Punkt $\overrightarrow{p_{E_2}}$ und den Ortsvektoren $\vec{a}$ und $\vec{b}$.

Eine Hilfsgerade h wird benötigt, um auf den Ebenen E_1 und E_2 jeweils einen Punkt zu finden, die im rechten Winkel zueinander stehen.

$$h: \quad \vec{x} = \overrightarrow{p_{E_2}} + r \cdot \vec{n}$$

Als nächstes wird von einem Punkt in E_2 der Schnittpunkt $\vec{i}$ in der Ebene E_1 mit der Hilfsgeraden berechnet:

$$\begin{aligned} E_1 \cap h: \quad & \vec{n} \cdot (\overrightarrow{p_{E_2}} + r \cdot \vec{n} - \overrightarrow{p_{E_1}}) = 0 \\ \text{d.h.} \quad & r = \frac{-(\vec{n} \cdot \overrightarrow{p_{E_2}}) + \vec{n} \cdot \overrightarrow{p_{E_1}}}{\vec{n} \cdot \vec{n}} \end{aligned}$$

$\vec{i} = \overrightarrow{p_{E_2}} + r \cdot \vec{n}$ ist der Punkt auf E_1, der rechtwinklig zum Punkt $\overrightarrow{p_{E_2}}$ auf E_2 liegt.

Nun kann der Abstand zwischen dem Punkt $\overrightarrow{p_{E_2}}$ und $\vec{i}$ berechnet werden. Das Ergebnis d ist die Differenz zwischen den beiden Ebenen E_1 und E_2.

$$\begin{aligned} \vec{l} &= \vec{i} - \overrightarrow{p_{E_2}} \\ d &= |\vec{l}| \end{aligned} \tag{3.10}$$

Mit dieser Rechnung werden von dem neu zu berechneten Objekt und dem Referenzobjekt alle Ebenen verglichen. Die Ebenen, die parallel zueinander sind und gleichzeitig die kleinste Differenz haben, müssen im nächsten Schritt aneinander gehaftet werden.

3.9.2 Anpassung an das Referenzobjekt

Abbildung 3.15: Zwischenergebnis während „Sticky“ durchgeführt wird: das Dach liegt mittig direkt auf dem Rumpf, hat aber noch die alten Dimensionen.

Das anzupassende Objekt wird an das Referenzobjekt gehaftet, indem die Mittelpunkte m_1 (beim Referenzobjekt) und m_2 (beim anzupassenden Objekt) der zwei gefundenen parallelen Flächen der beiden Objekte aufeinander gebracht werden. Für die Verschiebung wird der Objekt-Referenzpunkt c, der in Kapitel 3.3 beschrieben wurde, um die Differenz l der beiden Flächenmittelpunkte versetzt:

$$\begin{aligned} c' &= c + l \\ l &= (\overrightarrow{m_2} - \overrightarrow{m_1}) \end{aligned} \tag{3.11}$$

Das Zwischenergebnis ist in Abbildung 3.15 zu sehen. Mit den Größen des Zwischenergebnisses wird nun das Gradientenabstiegsverfahren erneut auf dieses Objekt angewendet. Der Freiheitsgrad wird auf den Mittelpunkt m eingeschränkt, damit das Objekt sich nur

rechtwinklig zur parallelen Ebene bewegen darf. Damit werden die Dimensionen (b, h, t) des Objekts neu berechnet, wobei es nun zum Referenzobjekt stets parallel und „sticky“ ist.

Abbildung 3.16: Objekt nach Anwendung von „Sticky“

Das Ergebnis ist im Beispiel von Abbildung 3.16 ein Haus, bei dem das Dach symmetrisch und ohne Abstand auf dem Hausrumpf liegt.

4 Texturen

Wie in der Einleitung in Kapitel 1 erwähnt, soll das rekonstruierte 3D Objekt sich nicht nur mit der Form, sondern sich auch visuell durch Texturen dem Eingabeobjekt ähneln. Dafür werden aus dem Eingabebild Bildteile extrahiert und als neue Bilddateien abgespeichert. Diese Extraktionen werden als Texturen für die 3D Objekte verwendet. Da die Eckpunkte der Objektteile bereits bekannt sind, können sie für die Extraktion der Texturen aus dem Eingabebild verwendet werden. Bei den Objekten, die in dieser Arbeit rekonstruiert werden, gibt es zwei Arten von Objektseiten: Dreiecke (bei der Pyramide und dem Prisma) und Vierecke (beim Prisma und Quader).

4.0.1 Texturen extrahieren bei Dreiecken

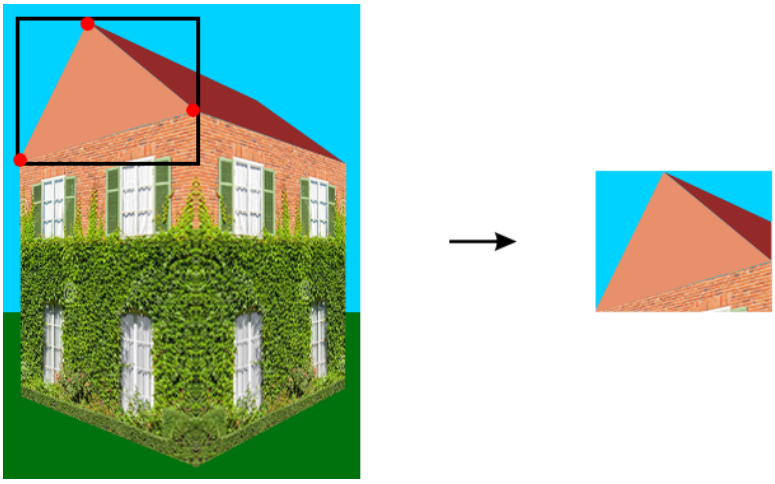

Abbildung 4.1: Extraktion eines Dreiecks aus einem Bild. Links: Originalbild, rechts: extrahierte Fläche. Für die Verbindung zwischen Objekt und Textur müssen hier die Koordinaten der Eckpunkte des Dreiecks in der Textur berechnet werden.

Eine Bilddatei ist grundsätzlich immer rechteckig. Beim Extrahieren wird daher der Bildausschnitt gewählt, der wie ein Hüllkörper das Dreieck umschließt. Abbildung 4.1 zeigt den Prozess der Extraktion. Um die Eckpunkte des Dreiecks in der Textur auf die Eckpunkte der dreieckigen Fläche des 3D Objekts zu legen, müssen die Koordinaten der Eckpunkte in der Textur bekannt sein. Der Maßstab bzw. die Dimension der Koordinaten ist sowohl für die Höhe als auch für die Breite des extrahierten Bildes in Fließkommazahlen zwischen 0.0 und 1.0.

4.0.2 Texturen extrahieren bei Vierecken

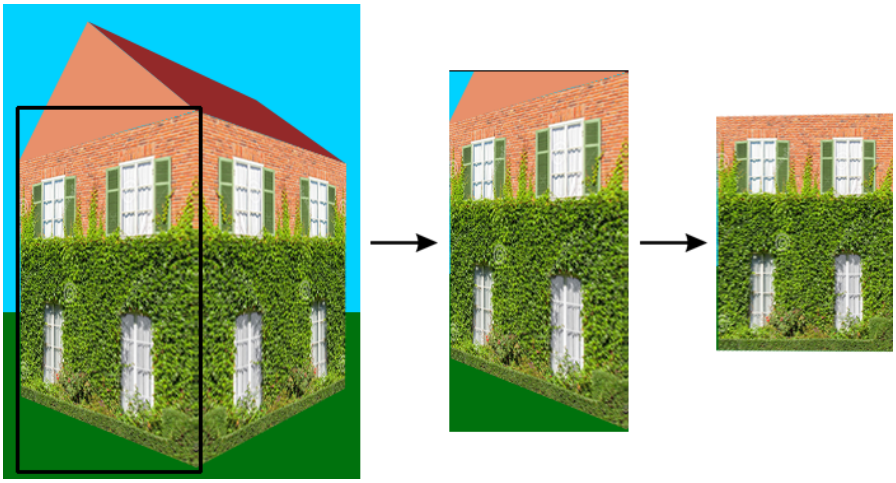

Abbildung 4.2: Extraktion eines Vierecks aus einem Bild in drei Schritten. Links: Originalbild, Mitte: extrahierte Fläche, Rechts: zu einem Rechteck verzerrtes Bild der extrahierten Fläche. Die Ecken aus dieser Texturen können direkt mit den Ecken des Objekts verbunden werden.

Es gibt zwei Möglichkeiten, ein Viereck aus einem Bild zu extrahieren. Die erste Möglichkeit (in Abbildung 4.2) besteht darin, das Viereck an den zweidimensionalen Punkten zu entnehmen und dann zu einem Rechteck zu verzerren. Das Ergebnis wird im Anschluss abgespeichert. Diese Variante hat den Vorteil, dass die Eckpunkte der Textur direkt mit den Ecken des Bildes gegeben sind. Diese Vorgehensweise vereinfacht den Aufbau der Objektdatei.

Die zweite Möglichkeit besteht darin, ein Rechteck an dem kleinsten Eckpunkt und an dem größten Eckpunkt der Eingabe aufzuspannen und dieses zuzuschneiden. Abbildung 4.3 veranschaulicht dieses Vorgehen. Wie im Falle des Dreiecks in Bild 4.1 müssen hier die

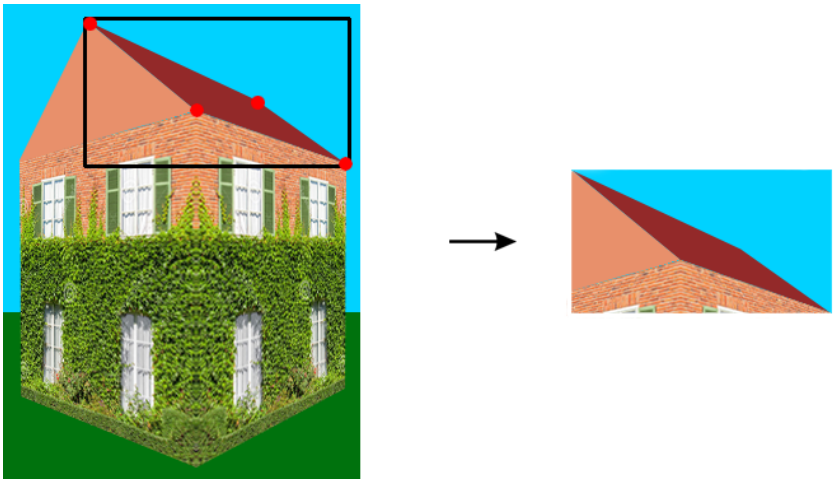

Abbildung 4.3: Zweite Möglichkeit der Extraktion eines Vierecks. Links: Originalbild, rechts: extrahierte Fläche. Um diese Textur zu nutzen, müssen wieder wie in Bild 4.1 die Koordinaten der Eckpunkte in der Textur bekannt sein.

Koordinaten der Eckpunkte des im Bild liegenden Vierecks abgespeichert werden. Der Vorteil hier ist, dass das Bild nicht verzerrt werden muss. Allerdings wird die Objektdatei durch die Speicherung der Eckpunkte größer als bei der Extraktionsmöglichkeit aus dem vorherigen Beispiel 4.2.

Beide Verfahren werden in dieser Arbeit angewendet und funktionieren gleichermaßen gut.

5 Entwicklung in Java

Die Software des Prototypen wird für den Windows-Desktop-PC entwickelt. Die Entwicklung erfolgt in der Programmiersprache Java. Dadurch könne über das Projekt hinaus die Methoden schnell auf weitere Plattformen wie z.B. Android übertragen werden. Über die Benutzeroberfläche, die mit dem GUI-Toolkit Java Swing [12] und AWT [11] erstellt wird, kann der Benutzer per Mausklick die Punkte eingeben. Die korrespondierenden Punkte werden mit jeweils einem Array umgesetzt. Die Punkte liegen in derselben Reihenfolge in den Arrays und somit kann schnell auf die Korrespondenzen zugegriffen werden.

Für die in diesem Projekt notwendigen Matrix-Berechnungen wird die externe Java-Bibliothek „Efficient Java Matrix Library“ (EJML, Version 0.29, [1]) verwendet.

5.1 UML Klassendiagramm des gesamten Projekts

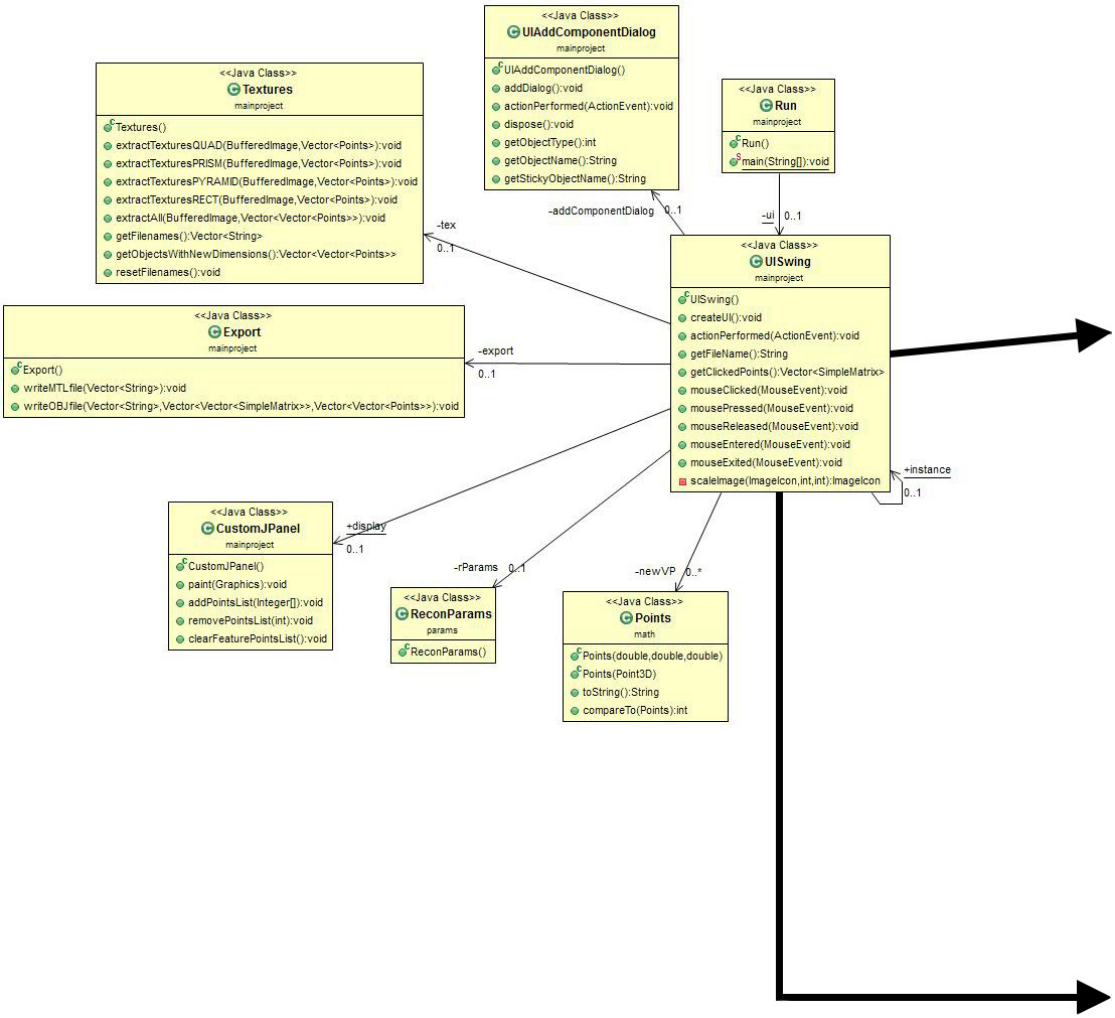

Abbildung 5.1: UML-Klassendiagramm des Projekts (Fortsetzung siehe folgende Seite).

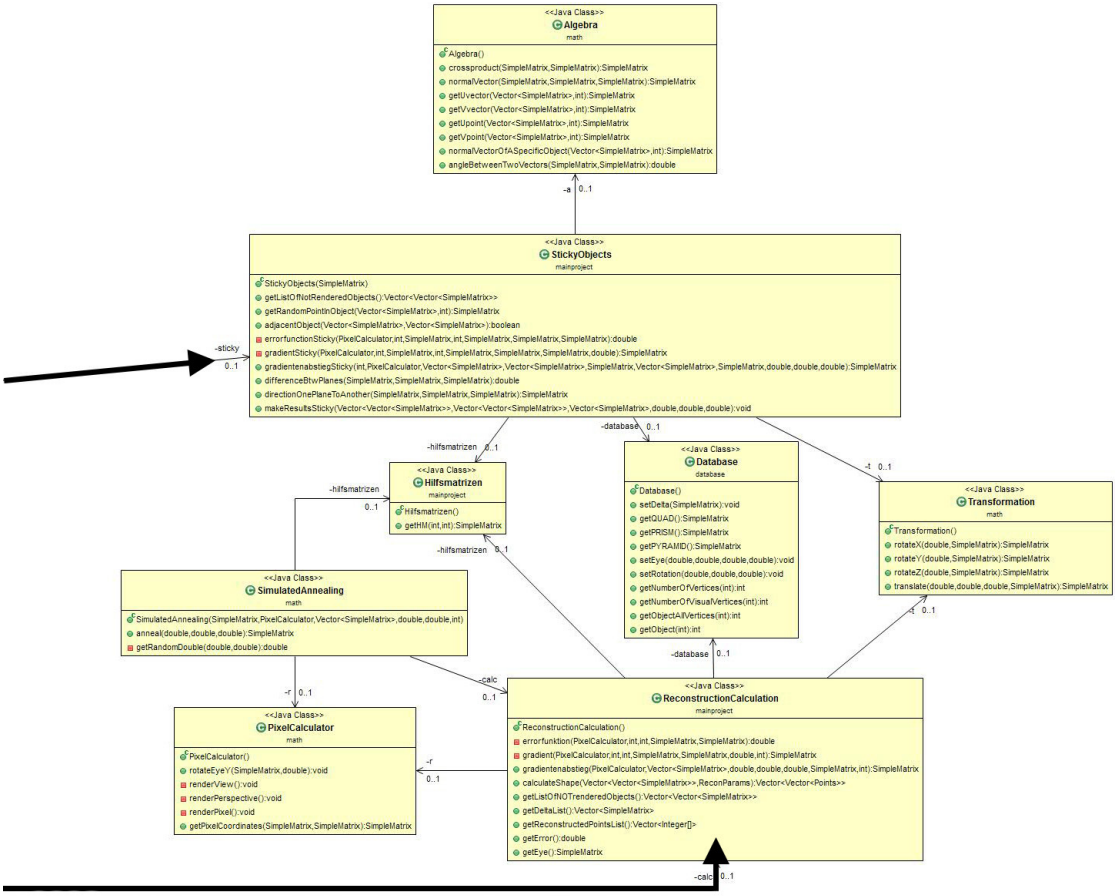

Abbildung 5.2: Fortsetzung von Abbildung 5.1: UML-Klassendiagramm des Projekts.

5.2 Die Benutzeroberfläche

Die Klasse für die Benutzeroberfläche des Prototypen heißt UISwing. Dort kommen alle Funktionalitäten zusammen, die den berechnenden und den schreibenden Klassen Informationen per Klick auf die Oberfläche oder auf die Buttons oder per Eingabe in die Textfelder weitergeben. Abbildung 5.3 zeigt einen Screenshot dieses Prototypen.

Mit Klick auf „Bilder laden...“ kann der Anwender ein Bild auf die Benuzeroberfläche laden. Im zweiten Schritt muss er die Eckpunkte des Hauses eingeben. Dafür muss er die Gebäudeteile wie Dach und Rumpf einzeln hinzufügen. Mit Klick auf „1. Gebäudeteil auswählen“ (auf der rechten Seite des Fensters) öffnet sich ein Popup Fenster und der Benutzer wird durch den Prozess der Eckpunkt-Eingabe aufgeklärt. Er muss sich dabei

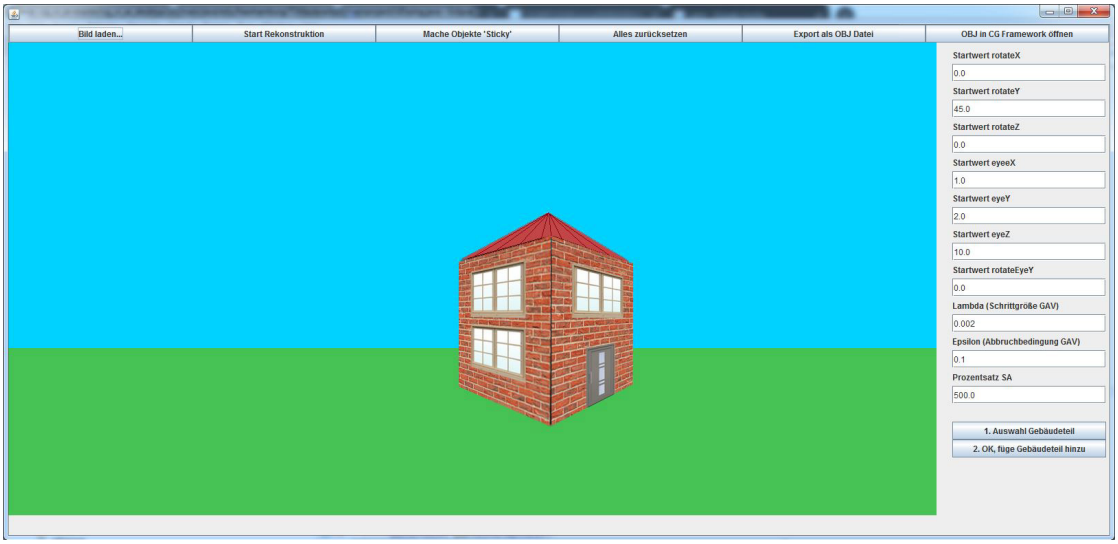

Abbildung 5.3: Screenshot des Prototypen. Hier wurde bereits ein Bild hinein geladen.

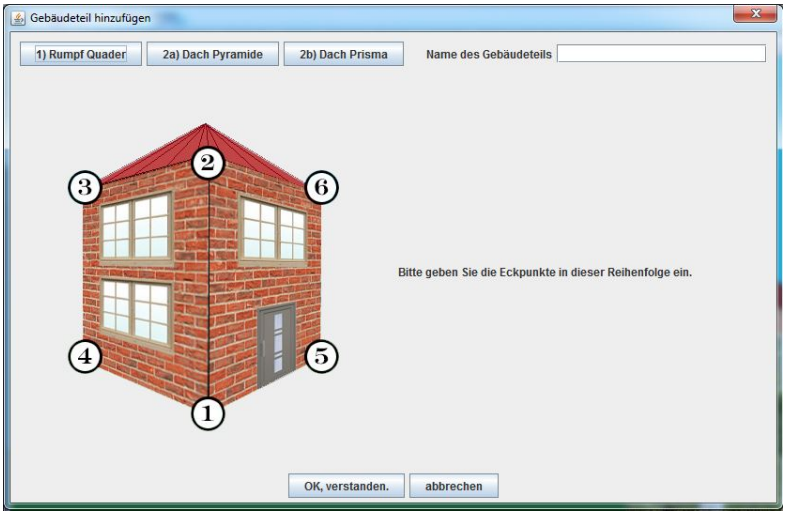

Abbildung 5.4: Fenster für die Auswahl der Objektart, die hinzugefügt werden soll.

festlegen, welche Art eines Gebäudeteils er eingeben möchte. Er kann dem Objektteil einen Namen geben, kann dieses Feld aber auch leer lassen. Im letzten Fall benennt das Programm die Objektteile ohne Namen „Gebäudeteil 1“ und zählt die Anzahl der Objektteile für den Default-Namen hoch. In dem Popup Fenster bekommt der Anwender die Anleitung, in welcher Reihenfolge er die Eckpunkte für die Eingabe zu setzen hat. Mit Klick auf „OK, verstanden“ schließt sich das Fenster und die Bildfläche der Benutzeroberfläche ist für die Eingabe freigegeben.

Bei jedem Klick auf die Bildfläche erscheint an der Stelle ein schwarzer Punkt. Wenn der Anwender die Eckpunkte vollständig eingegeben hat, klickt er auf den Button „2. OK, füge Gebäudeteil hinzu“. Wenn die Anzahl der Punkte zum ausgewählten Objektteil passen (z.B. 6 Punkte für einen Quader), werden die Daten des Gebäudeteils gespeichert und der Name dessen erscheint unter den Buttons auf der rechten Seite. Wenn die Anzahl der Eingabepunkte nicht stimmt, wird die Eingabe gelöscht, der Anwender bekommt eine Popup Meldung und muss den Eingabeprozess von vorne beginnen.

Bevor der Anwender die Rekonstruktion beginnt, hat er noch die Möglichkeit, einige Startparameter für die Berechnungen zu setzen. Das sind unter anderem die Rotations- und Augenpunktwerte aus dem δ-Parameter aus (3.1), der y-Neigungswinkel der virtuellen Kamera, die Schrittgröße und Abbruchbedingung des Gradientenabstiegsverfahren und der Prozent-Anteil der Zufallszahlen für das Simulated Annealing. Diese Startparameter besitzen bereits Default-Werte in der Datenbank, aber da dieser Prototyp nicht für den Endanwender vorgesehen ist, dienen die Felder dazu, den Rekonstruktionsalgorithmus mit verschiedenen Werten zu testen.

Mit Klick auf den Button „Start Rekonstruktion“ wird der Rekonstruktionsalgorithmus mit dem Gradientenabstiegsverfahren ausgeführt. Gegebenenfalls wird nach der Rekonstruktion noch das Simulated Annealing gestartet, wenn das Ergebnis des Gradientenabstiegsverfahren eine große Abweichung zur Eingabe aufweist. Rote Linien zeigen den Rekonstruktionsprozess, und gelbe Linien das aktuell beste Ergebnis des Simulated Annealing an. Abbildung 5.5 zeigt das Ergebnis einer Rekonstruktion. Nach der normalen Rekonstruktion sollte mit „Sticky Objects“ dafür gesorgt werden, dass alle Objektteile fest an- oder aufeinander liegen. Der Button „Mache Objekte 'Sticky'“ startet diesen Algorithmus.

Dieses Objekt lässt sich nun per Klick auf „Export als OBJ Datei“ als .obj Objektdatei exportieren. Der Button „In CG Framework öffnen“ dient dazu, dieses Objekt direkt in

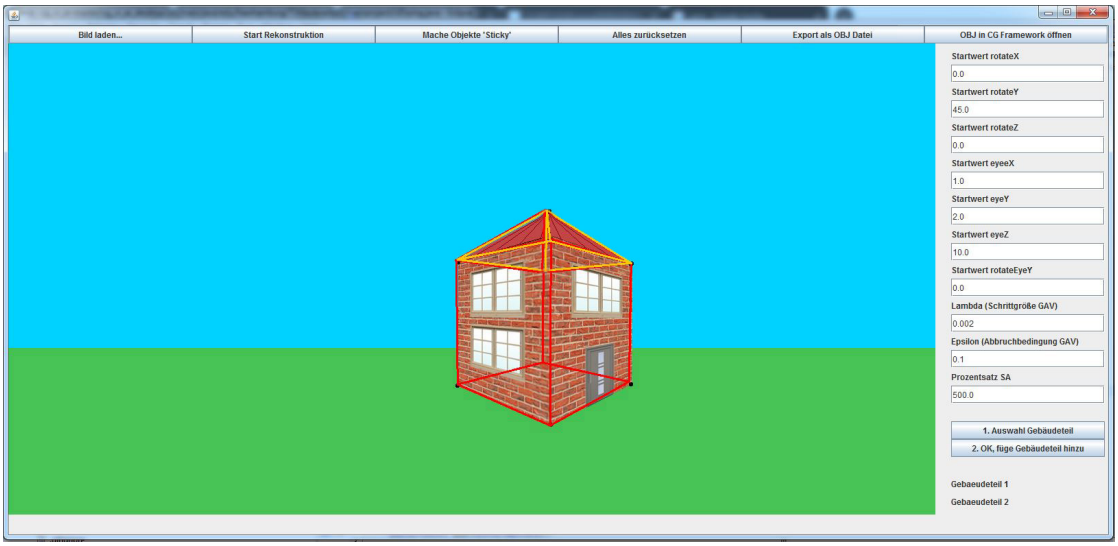

Abbildung 5.5: Beispiel eines gerade rekonstruierten Objekts.

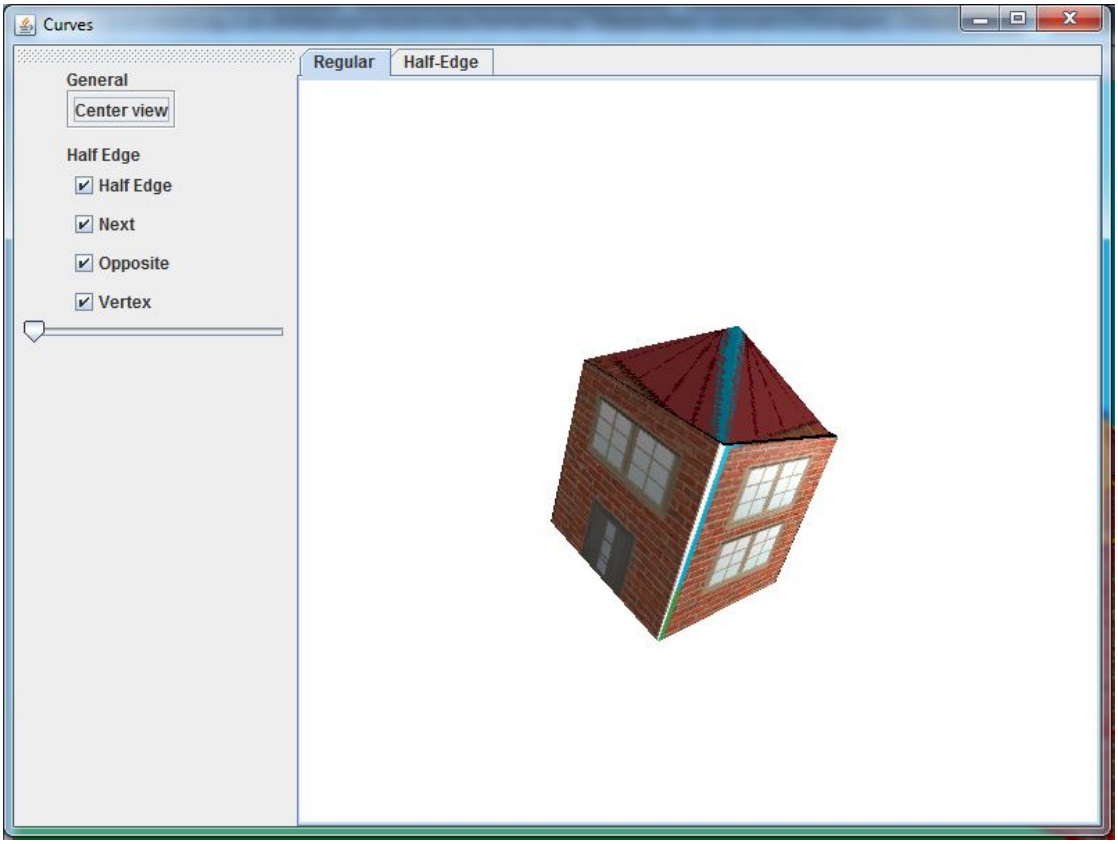

Abbildung 5.6: Anzeige des rekonstruierten Objekts als 3D Form im Computergrafik Framework.

der 3D Form anzeigen zu lassen. In dem neuen Fenster (Abbildung 5.6) lässt sich das Objekt dann herum drehen und zoomen.

5.3 Die Datenbank

Die Datenbank ist eine normale Java-Klasse. Die Modellobjekte werden dort als δ-Vektoren gespeichert (siehe Formel (3.1)). Wie bereits in Kapitel 3.3 erwähnt, haben die ausgewählten Modelle in dieser Arbeit alle dieselbe δ-Struktur, weshalb deren δ-Vektor sich in der Datenbank nicht unterscheiden.

Neben den Modellobjekten bietet die Datenbank auch Methoden an, um bestimmte Objektformen zu identifizieren, zum Beispiel mit `getObject` wird anhand der Anzahl der Eckpunkte die dazugehörige Art des Objekts (Quader, Prisma, Pyramide) zurückgegeben oder mit `getNumberOfVertices` die Anzahl der Eckpunkte eines Objekts, wenn die Objektform bekannt ist.

5.4 Optimierung

5.4.1 Ausführung der Algorithmen zur Rekonstruktion

Die Klasse ReconstructionCalculation ist die wichtigste Klasse dieser Arbeit. Sie setzt die Rechnungen aus Kapitel 3 um. Mit Klick auf den Button „Start Rekonstruktion“ wird die Methode `calculateShape` aus ReconstructionCalculation aufgerufen:

```
public Vector<Vector<Points>> calculateShape(
        Vector<Vector<SimpleMatrix>> inputObjects,
        ReconParams calculateShapeParams)
```

Die Methode `calculateShape` startet den Rekonstruktionsprozess, der nacheinander alle Eingabeobjekte rekonstruiert. `calculateShape` bekommt als Eingabeparameter die Liste aller Eingabeobjekte und ein Objekt aus der Klasse `ReconParams`, das die Werte enthält, die für die Rechnungen relevant sind:

- ϵ, die Abbruchbedingung, und λ, die Schrittgröße, für das Gradientenabstiegsverfahren (siehe Algorithmus 1)
- Einen großen Wert für h, für den Differenzenquotienten (siehe Formel (3.9))
- Einen Prozentwert für das Simulated Annealing 4: anstatt den Zufallszahlen sich selbst zu überlassen, wird mit diesem Prozentwert vorgegeben, in welcher Größenordnung sich die Zufallszahlen bewegen sollen. Der Bereich der Zufallszahlen, die sich auf den bestehenden δ-Wert addieren soll, liegt dann in einer Spanne von $+/-$ einem hohen prozentualen Anteil des Originalwerts. Das sorgt dafür, dass auf sehr große Werte nicht nur sehr kleine Werte addiert werden, sonst würde der Algorithmus sich wieder an dasselbe lokale Optimum annähern.
- Die (x, y, z)-Koordinaten und die Neigung in der y-Achse des Augenpunkts aus der Datenbank. Diese Werte können auch von dem Anwender aus der GUI eingetragen und übernommen werden.
- Die (x, y, z)-Koordinaten der Objektrotation aus der Datenbank. Auch diese Werte können vom Anwender gegeben werden.

Bei der Rekonstruktion des ersten Objekts werden alle Parameter des δ-Vektors aus (3.1) rekonstruiert. Da es sich um eine grafische Szene handelt, in der alle Eingabeobjekte sich befinden, wird nach dem ersten Objekt der Augenpunkt kein weiteres Mal rekonstruiert. Die Eigenschaften des Augenpunkts, die beim ersten Objekt gefunden wurden, werden für alle nachfolgenden Objekte weiterverwendet.

Ineinander verschachtelt werden die Methoden `gradientenabstieg`, `gradient` und `errorfunktion` abgerufen.

5.4.2 Errorfunktion

Nachfolgend gezeigte Methode ist die Implementierung der Errorfunktion $f(\delta)$ (siehe Formel (3.5)). Der Rückgabewert ist die Summe der Elemente aus dem Fehlervektor $(x_f - P_p \cdot P_v(\delta) \cdot x_m(\delta)^2)$.

```
private double errorfunction(PixelCalculator r, int object,
        int vertex, SimpleMatrix xf, SimpleMatrix delta)
```

Mithilfe des PixelCalculator-Objekts aus dem Eingabeparameter wird bei jedem neuen δ_{i+1} geprüft, welche Differenz die aus dem δ_{i+1} resultierenden Punkte im 2D Bild zu den Featurepunkten hat (Formel (3.4)). Der Eingabeparameter „object“ sagt aus, um welche Objektform es sich handelt. 1 ist der Quader, 2 das Prisma und 3 die Pyramide. `delta` ist ein SimpleMatrix-Objekt aus der EJML-Bibliothek. EJML stellt keine separaten Objekte für Vektoren zur Verfügung, Zeilen- und Spaltenvektoren werden daher auch mit der Klasse SimpleMatrix erstellt.

5.4.3 Der Gradient für den Gradientenabstieg

Als nächstes wird mithilfe des Ergebnisses der `errorfunction` der Gradient mittels Differenzenquotienten berechnet:

```
private SimpleMatrix gradient(PixelCalculator r,
        int object, int vertex, SimpleMatrix xf, SimpleMatrix delta,
        double h, int obj)
```

5.4.4 Der Gradientenabstieg

Nun wird der Gradientenabstieg berechnet. Verwendete Methoden sind hier sowohl `errorfunktion` als auch `gradient`. Mit einer while-Schleife wird δ_{i+1} berechnet. ϵ ist die Änderungsrate des quadratischen Fehlers nach jedem Durchlauf in der while-Schleife und dient als deren Abbruchbedingung. Da durch das Quadrieren der Fehlerwert sehr klein wird, muss die Abbruchbedingung ϵ entsprechend klein gewählt werden, z.B. 10^{-10} gibt ein gutes Ergebnis. Zurückgegeben wird δ_{i+1} als SimpleMatrix Vektor.

```
private SimpleMatrix gradientenabstieg(PixelCalculator r,
        Vector<SimpleMatrix> currentObject, double epsilon,
        double lambda, double h, SimpleMatrix deltaI, int obj)
```

Die Methode `gradientenabstieg` wird in der Methode `calculateShape` verwendet, um die neuen Punkte gebündelt zu generieren. Mithilfe von

`gradientenabstieg` wird der Vektor δ berechnet. In einer for-Schleife werden dann alle Punkte des ausgewählten Objekts auf Grundlage von δ berechnet und in einem Vektor gespeichert und zurückgegeben.

5.4.5 Simulated Annealing

Nach der ersten Rekonstruktion eines Objekts wird geprüft, wie groß die Abweichung zwischen der Eingabe und der Rekonstruktion ist. Ist die Abweichung größer als ein festgelegter Wert (in dem Prototypen sind es 15 Pixel pro Eckpunkt), soll mittels Simulated Annealing ein besseres Ergebnis ermittelt werden. Der Algorithmus ist bereits in Kapitel 3.8 beschrieben. Implementiert wurde die für diese Arbeit angepasste Version des Algorithmus aus Kapiel 3.8.1.

In der Klasse SimulatedAnnealing gibt es folgende Methode:

```
public SimpleMatrix anneal(double epsilon, double errorRec,
        double prozentsatz)
```

Der Parameter „epsilon“ ist dabei die Fehlertoleranz von allen Eckpunkten summiert (Anzahl in Pixeln), „errorRec“ ist der Errorwert aus der ersten Rekonstruktion und „prozentsatz“ gibt die Größenordnung der Zufallswerte für den Algorithmus an. Die Zufallszahlen, die auf das Zwischenergebnis addiert werden, müssen immer positiv sein, damit keine negativen Dimensionen entstehen.

Nachdem neue Startwerte fest stehen, wird der Rekonstruktionsalgorithmus mit dem Gradientenabstiegsverfahren erneut aufgerufen. Die Methode gibt das Ergebnis zurück, das den kleinsten Fehlerwert hat.

5.5 Sticky Objects

Wie in Kapitel 3.9 beschrieben, gibt es die Option nach der Rekonstruktion aller Objekte die Funktion „Sticky Objects“ zu verwenden, die dafür sorgt, dass alle rekonstruierten Objekte nicht nur aus der zweidimensionalen Sicht, sondern auch im dreidimensionalen Raum ordentlich aufeinander liegen. In der gleichnamigen Methode „StickyObjects“ wird dieses Konzept umgesetzt.

Um „Sticky Objects“ zu starten, wird folgende Methode aufgerufen:

```
public void makeResultsSticky(Vector<Vector<SimpleMatrix>> ListOfXf,
        Vector<Vector<SimpleMatrix>> listOfNOTrenderedObjects,
        Vector<SimpleMatrix> deltaList, double epsilon, double lambda,
        double h)
```

Diese Methode ist ähnlich aufgebaut wie `calculateShape` aus Kapitel 5.4.1. Es bekommt als Parameter eine Liste mit allen Eingabeobjekte, allen rekonstruierten Objekten (davon die 3D Punkte), und wie bereits in `calculateShape` verwendet (siehe Kapitel 5.4.1), die Werte für ϵ, λ und h.

`makeResultsSticky` geht alle ihm gegebenen rekonstruierten Objekte durch und vergleicht diese miteinander. Es wird ein Objekt genommen und mit allen seinen vorherigen aus der Liste geprüft, ob es parallele Ebenen gibt. Dann wird mit folgender Methode zwei Objekte nach dem Prinzip aus Kapitel 3.9.1 auf parallele Ebenen geprüft:

```
public boolean adjacentObject(Vector<SimpleMatrix> currentObject,
        Vector<SimpleMatrix> stickToThisObject)
```

Falls es mehrere Objekte mit parallelen Ebenen gibt, werden nur die Ebenen-Paare und Objekte abgespeichert, die den geringsten Abstand zueinander haben. Damit wird verhindert, dass zum Beispiel die Unterseite eines Dachs sich an der Unterseite des Rumpfs orientiert oder dass ein Objekt mit einem anderen Objekt fixiert wird, das weiter weg liegt.

Wenn es parallele Ebenen gibt, werden diese dann symmetrisch aufeinander gelegt. Ein Objekt muss sich dabei seinem Vorgänger aus der Objektliste anpassen. Wie in Kapitel 3.9.2 beschrieben, werden von den beiden parallelen Flächen die Mittelpunkte berechnet und das Objekt, das sich anpassen muss, verschiebt sich um die Differenz der Mittelpunkte.

Nach dieser Verschiebung hat das sich anzupassende Objekt zwar auf die richtige Stelle im Raum bewegt, aber seine Dimensionen stimmen immer noch nicht. Auf dieses Objekt wird wieder das Gradientenabstiegsverfahren angewendet. Diesmal werden nur vier Dimensionen gesucht: die Höhe, Breite und Tiefe des Objekts und ein Wert r, der dafür

sorgt, dass sich das Objekt während des Gradientenabstiegsverfahrens fixiert wird. Die folgende Funktion wird nach der Formel (3.11) aus Kapitel 3.9.2 berechnet und stellt sicher, dass das Objekt sich während des Verfahrens nicht wieder von der Fläche des Vergleichsobjekts wegbewegen kann:

$$cNeu = cSticky + r \cdot (m2 - m1)$$

Dabei ist $cNEU$ der während Gradientenabstiegsverfahrens berechnete Wert des Objektmittelpunkts. $cSticky$ ist der „alte“ Objektmittelpunkt des anzupassenden Objekts, der dem Gradientenabstiegsverfahren übergeben wird. $m2-m1$ ist die Differenz zwischen den beiden parallelen Objektflächen. Durch den Faktor r wird die Ebene des anzupassenden Objekts trotz mehrfacher Neuberechnung der Dimensionen in dem Gradientenabstiegsverfahren an die Ebene des Referenzobjekts fixiert.

Nach der Berechnung wird in der Objektliste das Objekt aktualisiert. Alle Objekte müssen mit ihren Vorgängern verglichen werden, dann liefert der „Sticky“ Algorithmus das Ergebnis in Form einer aktualisierten Objektliste ab.

5.6 Berechnung der Pixelkoordinaten aus den 3D Punkten

Die Klasse PixelCalculator setzt die in Abschnitt 3.6 beschriebenen View-Transformation (`renderView`) , die perspektivische Transformation (`renderPerspective`) und die Transformation in Pixelkoordinaten (`renderPixel`) um. Mit der Methode `getPixelCoordinates` werden diese drei Transformationen ausgeführt und man bekommt zu einer 3D Objektkoordinate die passende 2D Pixelkoordinate. Die Einstellungen der virtuellen Kamera (Referenzpunkt, Augenpunkt, Oben-Vektor, near, fovy, Bildschirmhöhe- und Breite) sind dabei fest vorgegeben.

5.7 Extraktion von Texturen

Nachdem alle Objekte am richtigen Platz liegen, werden anschließend die Texturen, wie in Kapitel 4 beschrieben, aus dem Eingabebild extrahiert. Die Methode `extractAll` bekommt als Parameter das Bild als BufferedImage und die Liste mit den rekonstruierten Objekten als zweidimensionale Punkte. Diese Liste wird durchgegangen und für alle

Objekte werden die Texturen extrahiert und als Bilddateien (Portable Network Graphics, .png) abgespeichert. Als Orientierung, wo die Bilder beschnitten werden sollen, werden die bereits bekannten zweidimensionalen Eckpunkte verwendet. Bei den hier in der Arbeit verwendeten Objektformen gibt es zwei verschiedene Arten von Objektflächen: Dreiecke bei Pyramiden und Prismen, und Vierecke bei Quader und Prismen. `extractAll` prüft zunächst, um welche Objektart es sich handelt, extrahiert dann die einzelnen Flächen für alle Objekte und speichert diese als Bilddateien ab.

5.8 Material- und Objektdateien

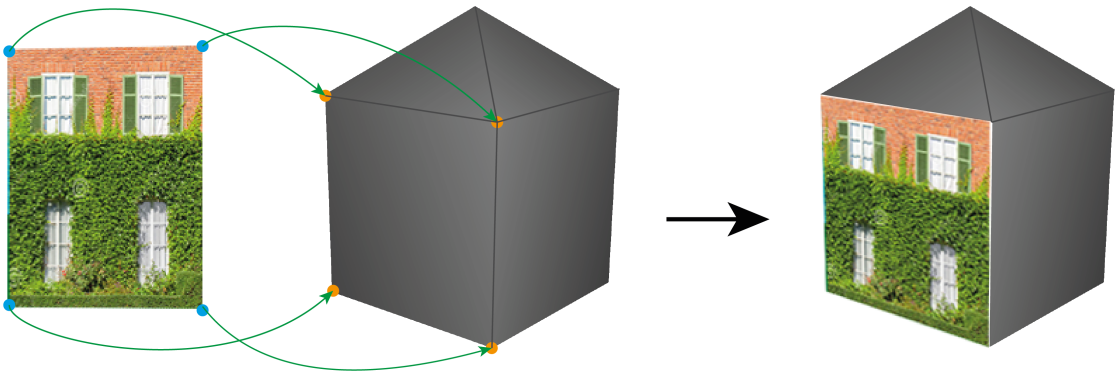

Abbildung 5.7: Anpassung der Textur auf die Objektoberfläche. Links: Ecken werden aufeinander gelegt. Rechts: Ergebnis.

Materialdateien werden als .mtl Dateien abgespeichert und enthalten Informationen über die Texturen wie Speicherort, Textur- und Dateiname.

Die Objektdatei lädt die Materialdatei ein und enthält zudem alle 3D-Eckpunktdaten der Objekte. Damit die Texturen mit den Eckpunkten der Objekte angepasst werden können, gibt es auch Eckpunktdaten der Textur im zweidimensionalen Koordinaten (Dimensionen von 0.0 bis 1.0). Bei einer Textur wie in Abbildung 4.1 oder 4.3 aus Kapitel 4 werden die Koordinaten der Eckpunkte zunächst in die Dimension der Textur zwischen 0.0 und 1.0 umgerechnet und dann in die Objektdatei eingetragen. Wenn die Textur die gesamte Fläche des Bildes einnimmt wie in Abbildung 4.2 aus Kapitel 4.0.2, können direkt die Werte 0.0 und 1.0 für die Textur-Eckpunkte verwendet werden. Für die Zusammenführung des 3D Objekts und der Texturen werden die Eckpunkte der Textur und die des Objekts aufeinander abgebildet (siehe Abbildung 5.7).

Die Seiten des Gebäudes, die nicht im Eingangsbild sichtbar sind, bekommen in der Objektdatei die Texturen von der gegenüberliegenden Seiten. Somit ist im Endergebnis ein gespiegeltes Gebäude zu sehen. Seiten, zu denen es weder gegenüberliegende Textur-Seiten noch andere Textur-Informationen gibt, wie zum Beispiel der Boden des Gebäudes, bekommen die Farbe schwarz.

6 Evaluation

Es wurden verschiedene Testläufe gemacht, um herauszufinden, mit welchen passenden Parametern das Zusammenspiel der Algorithmen (Gradientenabstieg, Simulated Annealing und Sticky Objects) am effizientesten ausgeführt wird. Getestet wurden die Eingabepunkte durch den Anwender, die Änderungsrate bzw. Abbruchbedingung ϵ, die Schrittgröße λ des Gradientenabtiegsverfahrens, die Startparameter der Modelle und die Zufallszahlen des Simulated Annealing Algorithmus. Zum Schluss wird getestet, wie sich die implementierten Algorithmen mit einem realen Eingabebild verhalten.

6.1 Eingabepunkte des Anwenders

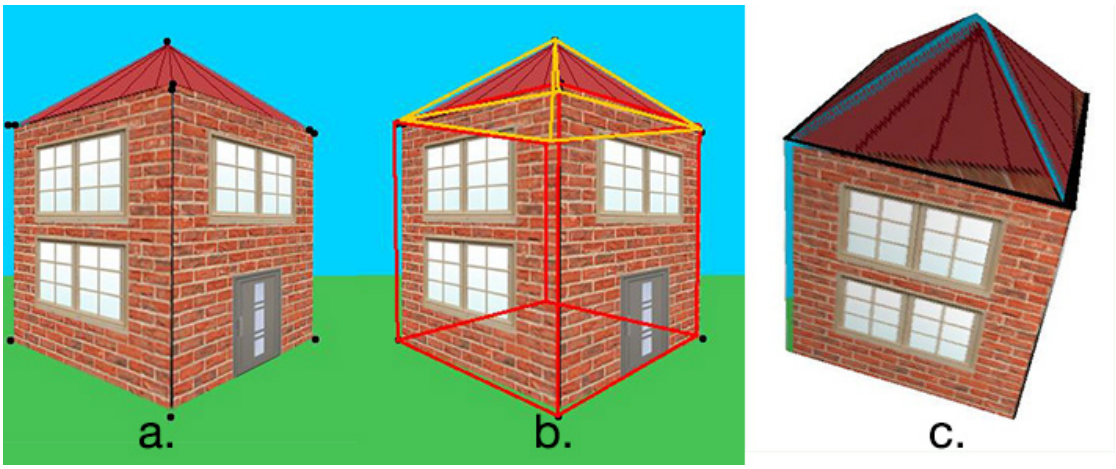

Abbildung 6.1: a. Viele Eckpunkte wurden ungenau ausgewählt. b. Rote und gelbe Linien zeigen das Ergebnis der Rekonstruktion. c. Ergebnis als 3D Modell, leicht gedreht und gekippt.

Die Ergebnisse und der Erfolg des Rekonstruktionsalgorithmus ist sehr abhängig von den Eingabepunkten des Anwenders. Gibt der Anwender die Punkte sehr genau ein, wird das

Ergebnis entsprechend sehr gut ausfallen. Allerdings, wenn der Anwender bei der Eingabe die Eckpunkte der Objekte nicht genau trifft, werden nicht nur das Objekt ungenau rekonstruiert, sondern auch die Texturen nicht an den richtigen Stellen extrahiert. Wie in Bild 6.1a. zu sehen, wurden die Eckpunkte des Hauses durch den Anwender sehr ungenau ausgewählt. Das hat zur Folge, dass die Grenzen des rekonstruierten Hauses, in Bild 6.1b. in der 2D Ansicht zu sehen, nicht exakt an den Grenzen des Modells entlang verläuft. In Bild 6.1c. ist klar zu erkennen, dass die Textur etwas versetzt aus dem Modell beschnitten wurde, sodass ein Teil des Hintergrundes (Blau und Grün) am 3D Objekt-Ergebnis zu sehen ist. Zudem scheint es, als sei das Dach zu klein für das Haus, da der Anwender die Eckpunkte für das Dach versetzt eingegeben hat.

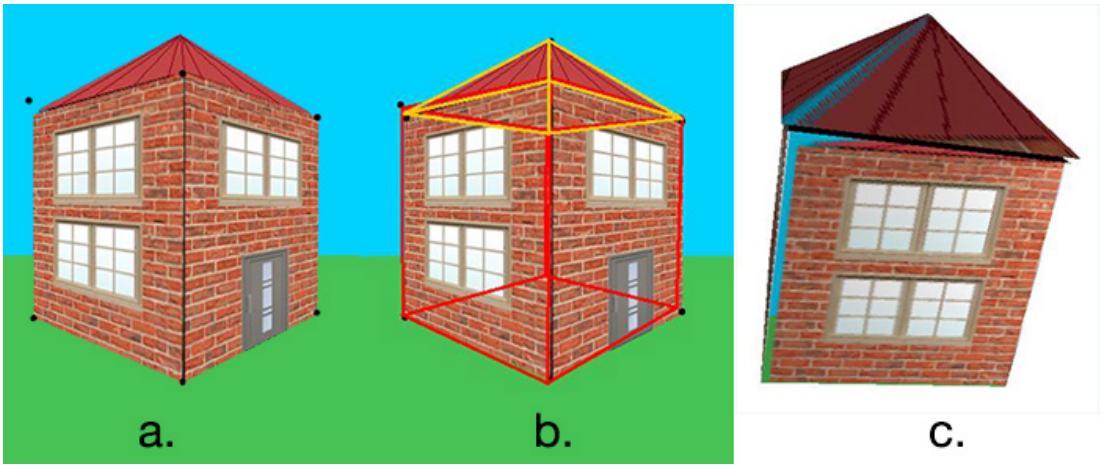

Abbildung 6.2: a. Ein Eckpunkt wurde am Rumpf (oben links) ungenau ausgewählt. b. Rote und gelbe Linien zeigen das Ergebnis der Rekonstruktion. c. Ergebnis als 3D Modell.

Wenn nur ein Punkt ein Ausreißer ist (Abbildung 6.2a.) und alle anderen sehr genau eingegeben wurden, sorgt der Aufbau der Einzelobjekte, bei dem bestimmte Winkel der Objekte während des Rekonstruktionsprozesses nicht verändert werden können, aus Kapitel 3.3 dafür, dass dieser das Ergebnis nicht extrem vom Wunschergebnis abweicht. Trotzdem reichen einige Pixel Ungenauigkeit aus, um das Ergebnis zu beeinflussen. In Abbildung 6.2c. ist zu sehen, dass durch die Abweichung ein Teil des Hintergrundes sich in der Textur befindet, ähnlich wie in dem vorherigen Beispiel. Und hier ist auch wieder das Dach etwas versetzt, weil beide Objektteile nach der Rekonstruktion verschiedene Breiten und Tiefen haben.

6.2 Variation der Änderungsrate

Optimalerweise soll das Gradientenabsteigsverfahren dann das Ergebnis zurück geben, wenn der Algorithmus konvergiert. Das ist in dieser Arbeit der Fall, wenn der alte und der neue Wert der Zwischenergebnisse (2D Koordinatenpunkte des Modellobjekts) sich sehr wenig voneinander unterscheiden. Diese Änderungsrate ϵ muss so gewählt werden, dass der Algorithmus nicht zu früh endet, aber auch nicht zu lange läuft, da die Details des Ergebnisses in Pixel-Nachkommastellen nicht mehr relevant sind.

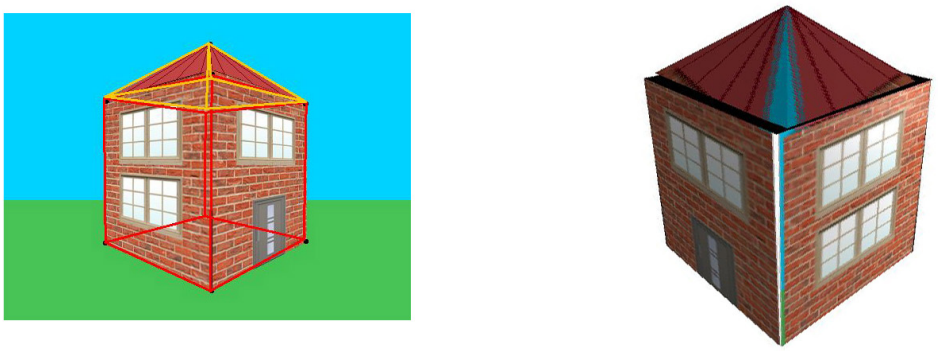

Abbildung 6.3: Ergebnis der Rekonstruktion (inklusive Sticky Objects) mit einem ϵ-Wert von 0,5. Links: Ansicht des Ergebnisses in 2D als rote und gelbe Linien über dem Eingabebild. Rechts: Ansicht des 3D Objekts im Computergrafik Framework.

In Abbildung 6.3 ist ein Test zu sehen, bei dem für die Abbruchbedingung ϵ des Gradientenabstiegsverfahrens der Wert $0,5$ gewählt wurde. Im 2D Ergebnisbild ist sichtbar, wie die Außenlinien der Rekonstruktion die Eingabepunkte nicht genau treffen. In diesem Beispiel liegt der Error-Wert aller Punkte für das gesamte Objekt bei 92 Pixeln (Fließkommazahl aufgerundet). Auch in dem 3D Ergebnisobjekt ist diese Abweichung ersichtlich. Das Dach wurde ungenau rekonstruiert und ist etwas kleiner als der Hausrumpf. Außerdem enthalten die Texturen Ausschnitte aus dem Hintergrund.

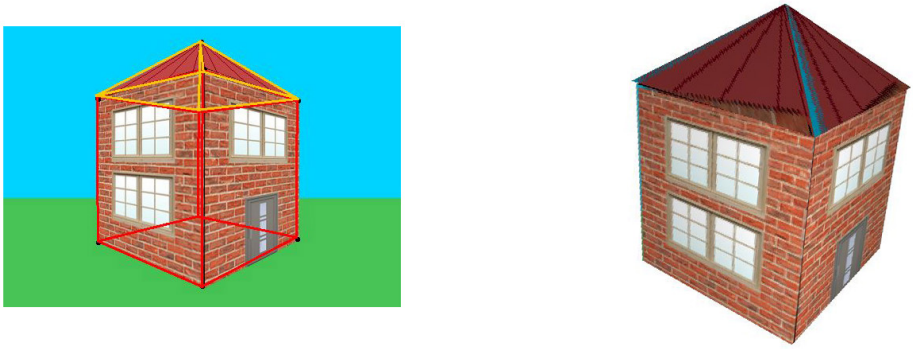

Abbildung 6.4: Ergebnis der Rekonstruktion (inklusive Sticky Objects) mit einem ϵ-Wert von 0,1. Links: Ansicht des Ergebnisses in 2D als rote und gelbe Linien über dem Eingabebild. Rechts: Ansicht des 3D Objekts im Computergrafik Framework.

Das Beispiel in Abbildung 6.4 wurde für ϵ der Wert $0,1$ verwendet. Das Ergebnis sieht schon viel genauer aus, die Ecken der Rekonstruktion liegt ziemlich genau auf den Eingabepunkten. Der Error-Wert ist aber insgesamt immer noch hoch, er ist bei 44 Pixeln. Auch das 3D Objekt des Ergebnisses sieht besser aus als im Beispiel 6.4. Das Dach liegt hier ziemlich genau auf dem Hausrumpf, steht aber trotzdem noch ein wenig über. Lediglich die Texturen enthalten noch Fragmente aus dem Hintergrund des Bildes.

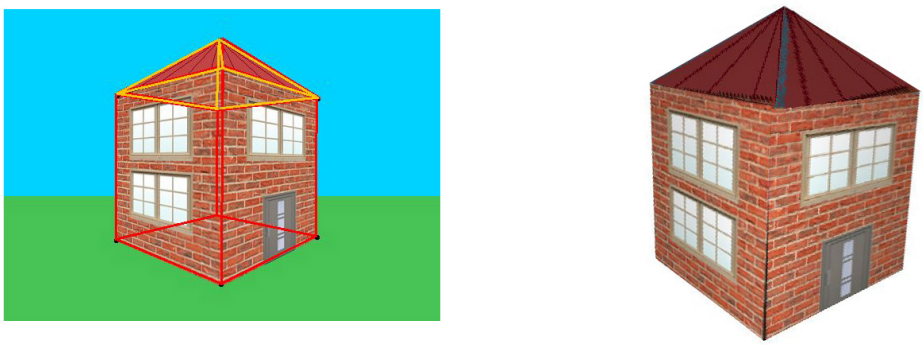

Abbildung 6.5: Ergebnis der Rekonstruktion (inklusive Sticky Objects) mit einem ϵ-Wert von 0,001. Links: Ansicht des Ergebnisses in 2D als rote und gelbe Linien über dem Eingabebild. Rechts: Ansicht des 3D Objekts im Computergrafik Framework.

Optisch sieht man als Anwender in Beispiel von Bild 6.5 hier bei der Objektform kaum einen Unterschied zu dem in Bild 6.4. Rechnerisch gesehen ist das Ergebnis dieses Beispiel aber viel besser, denn der Error-Wert liegt bei nur 14 Pixeln. Im Ergebnisobjekt ist hier am wenigsten Fehler zu erkennen. Das Dach ist von dem Sticky Objects Algorithmus sehr präzise rekonstruiert worden und die Texturen weisen sehr wenig Hintergrundfragmente auf.

6.3 Variation der Schrittgröße im Gradientenabstiegsverfahren

Wenn die Schrittgröße des Gradientenabstiegsverfahrens zu groß gewählt wird, kann es sein, dass das Minimum nicht oder sehr langsam erreicht wird, da es immer wieder „übersprungen“ wird. Verschiedene Tests mit dem Prototypen haben ergeben, dass eine Schrittgröße von 0,1 bereits zu groß ist. Der Algorithmus konvergiert nicht und das Modellobjekt „springt“ im Bild hin und her und findet nicht zu einem Ergebnis.

Auch die Auswahl der Schrittgröße von 0,01 ist immer noch zu groß. Hier reagiert der Algorithmus mit zwei verschiedenen Szenarien: entweder konvergiert er schnell zu einem Ergebnis, das sehr schlecht ist, zum Beispiel mit einer Abweichung von 15824 Pixel. Oder das Modellobjekt bewegt sich sehr schnell im Bild im „Zickzack“, da das lokale Minimum immer wieder übersprungen wird. Bei dem letzten Szenario kann es vorkommen, dass der Algorithmus irgendwann das Minimum noch findet, es kann aber sehr viel Zeit in Anspruch nehmen.

Ist die Schrittgröße wiederum zu klein gewählt, wird mehr Rechenzeit benötigt, um das Minimum zu erreichen. Dieses Ergebnis ist dann sehr genau.

Für die Tests in diesem Projekt erweist sich eine Schrittgröße von 0,002 als ideal. Mit diesem Wert konvergiert das Gradientenabstiegsverfahren zu einem akzeptablen Ergebnis und es wird nicht viel Rechenzeit beansprucht.

6.4 Simulated Annealing Algorithmus

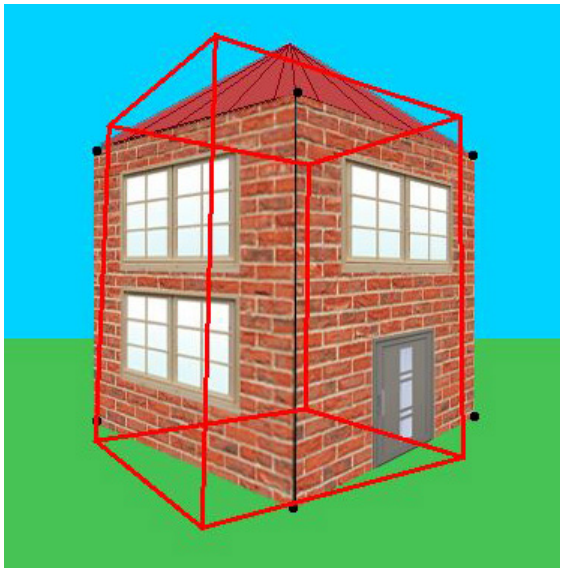

Abbildung 6.6: Ergebnis der Rekonstruktion des Quaders ohne Simulated Annealing.

In Abbildung 6.6 wurden Modellparameter verwendet, die stark von den Parametern des Eingabeobjekts abweichen. Dabei reichte es, nur dem Objektschwerpunkt c des Modells einen anderen Wert zu geben (statt $(5, 5, 5)$ wurde $(5, 55, 515)$ für die (x,y,z)-Koordinaten verwendet). Der Algorithmus des Gradientenabstiegsverfahren schafft es nicht, das Optimum zu finden und stoppt an der Stelle, wo der rote Quader zu sehen ist. Die Abweichung zur Eingabe beträgt 9871 Pixel. Hier wurde ein lokales Minimum gefunden, aus dem das Gradientenabstiegsverfahren ohne dem Simulated Annealing Algorithmus nicht wieder heraus finden würde.

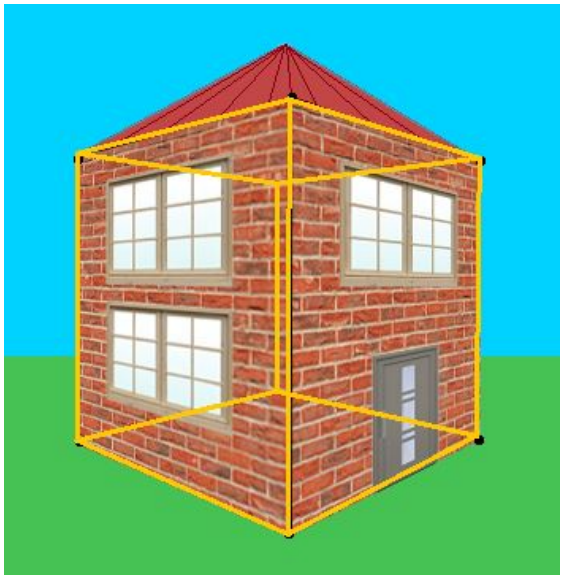

Abbildung 6.7: Ergebnis der Rekonstruktion des Quaders mit Simulated Annealing.

Mit der Anwendung des SA Algorithmus wird im vierten Anlauf das Optimum des Quaders gefunden. In Abbildung 6.7 ist dieses Ergebnis mit rotem Umriss zu sehen. Die Abweichung von der Eingabe liegt bei 18 Pixeln.

6.5 Realistisches Eingabeobjekt

In den vorherigen Beispielen wurde als Eingabebild eine für dieses Projekt eigens erstellte Abbildung verwendet, in der die Umrisse des Gebäudes mithilfe eines 3D Programms gezeichnet wurde. Für die Lage, Dimension, Rotation und auch für die Parameter der virtuellen Kamera wurden Werte ausgewählt. Die Grafiken und Texturen wurden ebenfalls manuell erstellt. Durch die Bekanntheit aller Parameter des Eingabeobjekts kann für diese gesamte Arbeit mit ähnlichen Modellparametern gearbeitet werden, damit die Rekonstruktion des Objekts bei den Tests eine angemessene Laufzeit einnimmt.

Abbildung 6.8: Ergebnis des Simulated Annealing bei einem realistischen Eingabeobjekt. Bildquelle: A. K. Vetter [8]

Da die Werte des Eingabeobjekts bei einer realistischen Verwendung der Software vollkommen unbekannt sind, gibt es hier nur als Hilfestellung für den Annäherungsalgorithmus das Eingreifen durch den Anwender, indem er durch Augenmaß und Probieren die Eingabeparameter so verändert, dass die Modell- und Kameraparameter der Eingabe ähnelt.

Wenn die Größen der Eingabe und des Modells zu sehr voneinander abweichen, kann es vorkommen, dass nach dem Durchlauf des Simulated Annealing Algorithmus immer noch kein gutes Ergebnis zu finden ist. In Abbildung 6.8 ist das Ergebnis der Rekonstruktion eines Quaders als gelbe Linien auf dem Foto nach dem SA Algorithmus zu sehen. Der Durchlauf hat etwa 45 Minuten gedauert und das Ergebnis weicht von der Eingabe um 2590 Pixel ab. In diesem Bild sind weder die Dimensionen des Objekts noch Informationen über den Augenpunkt vorhanden. Gewählt wurden daher nur grob geschätzt folgende Werte für das Modell. Für die Starttemperatur T für den Simulated Annealing Algorithmus wurde der Wert 10000 gewählt.

Es ist beim Simulated Annealing sehr vom Zufall abhängig, ob gute Werte getroffen werden, um das Optimum zu finden. Daher kann es, wie in dem Beispiel 6.8 gezeigt, 45 Minuten dauern, bis der SA Algorithmus ein halbwegs akzeptables Ergebnis erreicht. Sind die Zufallswerte mit Glück ideal getroffen worden, kann der SA-Algorithmus auch ein gutes Ergebnis nach einer Minute finden. Das Ergebnis des Rekonstruktionsalgorithmus ist, bei einem δ-Vektor der Länge 13, von sehr vielen Parametern abhängig. Es müssen nur wenige dieser Parameter ungünstige Zufallszahlen bekommen und der Algorithmus konvergiert in ein schlechtes lokales Minimum.

6.6 Viele lokale Minima

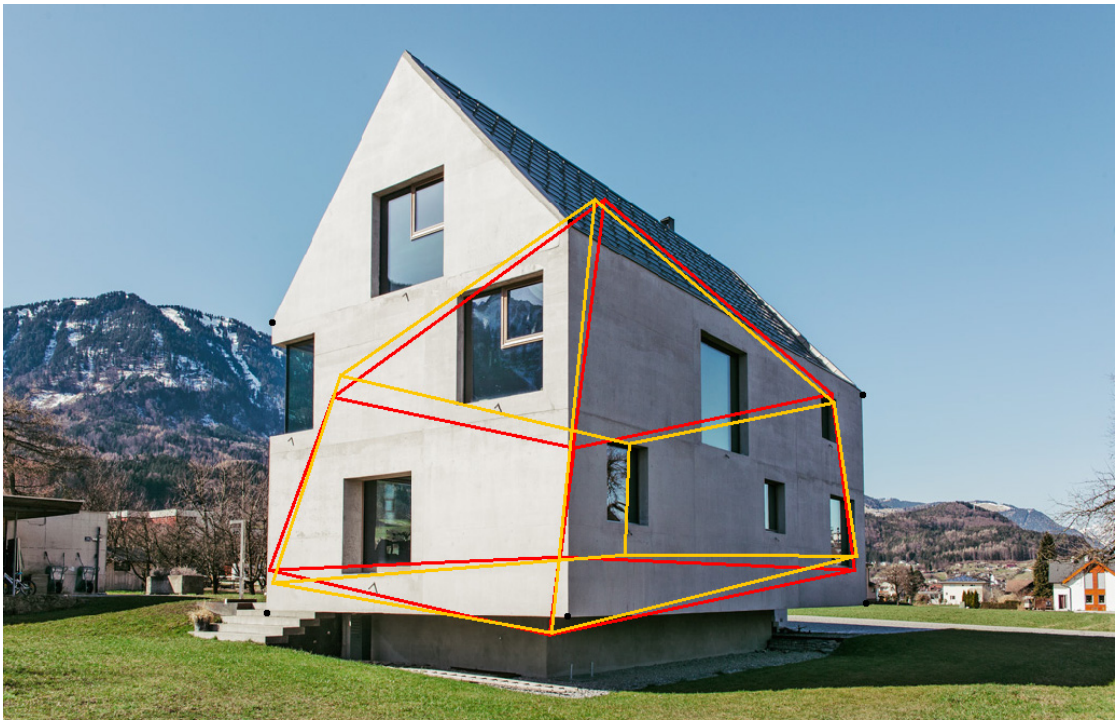

Abbildung 6.9: Das Gradientenabstiegsverfahren findet mehrere ähnliche lokale Minima. Bildquelle: A. K. Vetter [8]

Wenn der Augenpunkt unbekannt ist, ist das Gradientenabstiegsverfahren nicht der optimalste Algorithmus für die Objektrekonstruktion. In Bild 6.9 sind zwei Ergebnisse des Gradientenabstiegsverfahren während des SA Algorithmus zu sehen. Die Objektformen mit den roten und den gelben Linien deuten an, dass die Dimensionen der beiden Objekte

sehr ähnlich sind und dass bei beiden Ergebnissen der Augenpunkt jeweils einen etwas anderen Blickwinkel hat. Leider lässt sich dies nicht verhindern, da bei der Suche nach dem Optimum bzw. einem sehr guten lokalen Minimum bereits kleine Änderungen der Modellobjekt-Parameter und des Augenpunkts als Verbesserung erkannt werden können. Ist der Augenpunkt bereits bekannt, treten diese sich ähnelnden Ergebnisse nicht auf.

6.7 Ausführung mit idealen Eingangsparametern

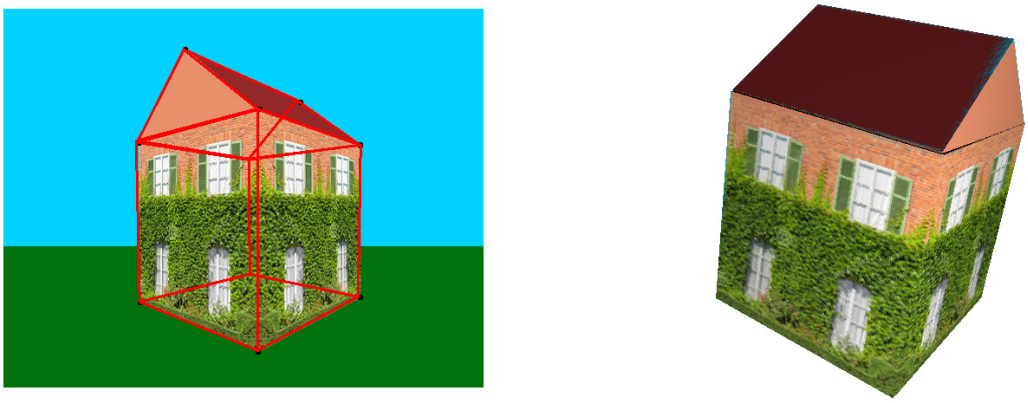

Abbildung 6.10: Ergebnis mit idealen Ausgangswerten.

Mit den in den vergangenen Abschnitten gefundenen optimalen Startwerten wird der Algorithmus für das Beispiel 6.10 gezeigt. Es wird für ϵ der Wert 0,001 gewählt. Die Schrittgröße λ des Gradientenabstiegsverfahrens erhält den Wert 0,002. Als Eingabebild wird wieder das Bild von einem Objekt verwendet, das manuell erstellt worden ist. Für die Startparameter des Modells in der Datenbank werden hier Werte ausgewählt, die sich nicht zu stark von dem Featureobjekt unterscheiden. Bei der manuellen Eingabe der Featurepunkte wird sehr genau darauf geachtet, dass diese präzise die Ecken des Objekts treffen. Das Ergebnis der Rekonstruktion mit diesen idealen Ausgangswerten ist in Abbildung 6.10 zu sehen. Es gibt kaum Abweichungen sowohl bei der Objektform als auch bei den Texturen.

7 Fazit

7.1 Gradientenabstiegsverfahren

Das Gradientenabstiegsverfahren erweist sich nach vielen Tests als praktikablem Annäherungsalgorithmus für das Rekonstruktions-Problem dieser Arbeit, wenn die Startparameter des δ-Vektors dem optimalen Ergebnis sehr ähnlich sind. Hilfreich ist es zum Beispiel, wenn der Augenpunkt bereits bekannt ist und der Ort, wo das Modellobjekt liegt, in der Nähe des Featureobjekts ist. Wie im letzten Kapitel 6 gezeigt, gibt es Beispiele, an denen das Gradientenabstiegsverfahren sich definitiv für die Problemstellungen dieser Arbeit eignet. Die Beispiele zeigen auch, dass es möglich ist, anhand der Parameter auszuwählen, ob man sich für die Genauigkeit des Ergebnisses oder eine längere Laufzeit entscheidet.

In der Konstellation, wie der Algorithmus hier verwendet wird, birgt er auch Verbesserungsmöglichkeiten:

1 - Da die Fehlerfunktion (3.5) viele lokale Minima hat, können Startparameter, die sich vom Optimum zu sehr unterscheiden, dafür sorgen, dass ein ungünstiges lokales Minimum gefunden wird. Gerade wenn der Augenpunkt nicht bekannt ist, werden sehr viele ähnliche lokale Minima gefunden, die alle aus einer anderen Perspektive der virtuellen Kamera gefunden wurden (siehe Kapitel 6.6).
Eine Erweiterungsmöglichkeit für dieses Projekt an dieser Stelle ist zum Beispiel eine höhere Anzahl der Bilder als Eingabe. Wenn es mehrere Fotos von dem Gebäude gibt, kann mithilfe der in Kapitel 2 erwähnten Methode Structure-from-Motion der Augenpunkt eines Bildes berechnet werden. Mit der Information über den Augenpunkt ist die Rekonstruktion eines Objektes schon weniger komplex und es müssen dann nur noch die Parameter des Objekts gefunden werden. Zudem können die vielen Fotos auch dazu

genutzt werden, um Objektteile des Gebäudes zu rekonstruieren, die sich auf einem Einzelfoto an einer verdeckten Stelle befinden und das Endergebnis wäre nicht wie in diesem Projekt gespiegelt, sondern bekommt an jeder Seite eine individuelle Textur.

2 - Bei einer zu großen Schrittgröße wird das Minimum nicht (auf direktem Wege) gefunden und das Objekt dreht sich zum Beispiel mehrmals um die eigenen Achse, weil die Parameter der Objektrotation sich ständig mit zu großen Werten verändern. Eine mögliche Lösung dazu ist es, den δ-Parameter in mehrere Einzelteile zu unterteilen. Zum Beispiel könnten die Objektrotation und die Eigenschaften des Augenpunkts als eigene δ-Einzelvektoren aus dem ursprünglich großen δ-Parametervektor genommen und diesen dann eigene Werte für die Schrittgrößen gegeben werden. Diese δ-Einzelteile müssen zwar weiterhin abhängig voneinander verarbeitet werden, aber durch die verschiedenen Schrittgrößen könnte der Algorithmus insgesamt schneller zum Ergebnis kommen.

3 - Die dritte Möglichkeit ist die dynamische Bestimmung der Schrittgröße. Das Modellobjekt würde dann nicht bzw. nur kurz sich um die eigene Achse drehen, aber durch die Anpassung der Schrittgröße würde es sich in den nächsten Schritten nicht weiter (oder weniger) drehen. Gleichzeitig hat eine dynamische Schrittgröße den Vorteil, dass die Genauigkeit der Rekonstruktion erhöht wird, da die Schrittgröße kleiner werden kann als ein fix vorgegebener Wert.

7.2 Simulated Annealing

Durch die Erweiterung des Rekonstruktionsalgorithmus mit dem Simulated Annealing Algorithmus hat dieser den positiven Aspekt, dass das Optimum trotzdem gefunden werden kann, wenn zuvor das Gradientenabstiegsverfahren nur zu einem lokalen Minimum konvergiert. In dem Beispiel in Kapitel 6.4 hat diese Funktionalität sich als erfolgreich erwiesen. Allerdings ist der SA Algorithmus keine Garantie für das Finden des Optimums. Er benötigt Zufallszahlen, um aus einem lokalen Minimum wieder „herauszukommen“. Wenn es zu viele Abhängigkeiten gibt, kann der SA Algorithmus mit seinen Zufallszahlen, wie in Beispiel 6.6 gezeigt, viele Ähnliche lokale Minima finden, die nacheinander jeweils eine kleine Verbesserung aufweisen. Oder, wie bei dem Beispiel in Kapitel 6.5 kann der SA Algorithmus konvergieren, ohne dass ein optimales Ergebnis gefunden wurde. Abhilfe zum Letzteren könnte eine sehr hohe Anfangstemperatur und damit eine sehr

lange Laufzeit verschaffen. Ob das Optimum dann mit Sicherheit gefunden wird, ist aber trotzdem nicht vorhersehbar.

7.3 Weitere Erweiterungsmöglichkeiten

Diese Arbeit bietet, neben den genannten Veränderungen bei den Algorithmen, weitere Möglichkeiten für Erweiterungen.

Da die Datenbank dieser Arbeit beliebig erweiterbar ist, bietet sie die Möglichkeit, ähnlich wie bei Hejrati et al. [15], von allen Objektformen sehr viele verschiedene Versionen abzuspeichern, die sich in der Größe, Ort und Rotation variieren. Somit werden auch Parameter für Modellobjekte gespeichert, die der SA Algorithmus durch seine Zufallszahlen eventuell nicht treffen würde. Diese Erweiterung würde mehr Rechenzeit bei der Rekonstruktion in Anspruch nehmen, aber gleichzeitig die Wahrscheinlichkeit erhöhen, dass das Optimum gefunden wird.

Die Suche nach dem Augenpunkt der virtuellen Kamera erhöht die Komplexität des Rekonstruktionsalgorithmus. In Kapitel 2 wurde die Arbeit von Schönberger und Frahm [4] erwähnt, bei der das „Structure from Motion“ (SfM) angewendet wird, bei dem mithilfe von mehreren Bildaufnahmen eines Objekts und einer Erkennung von gleichen Punkten im Bild aus verschiedenen Perspektiven es möglich ist, die Position der Aufnahmekamera zu berechnen. Würde man die Eingabevorgaben in dieser Arbeit mit mehr als einem Bild erweitern, könnte man das SfM vor dem Gradientenabstiegsverfahren ausführen, um die Position des Augenpunkts zu berechnen. Der Rekonstruktionsalgorithmus muss dann nur den Ort und die Dimensionen des Objekts bestimmen.

Wie in Abschnitt 3.4.2 beschrieben, wird in dieser Arbeit nur ein Bild des Objekts verarbeitet, was dazu führt, dass es Seiten vom Objekt gibt, das verdeckt ist und von denen es nicht möglich ist, Texturen zu entnehmen. Bei einer Erweiterung des Projekts um mehrere Bildaufnahmen bei der Eingabe können auch diese Texturen extrahiert werden und mögliche verdeckte Objektteile auch rekonstruiert werden.

Die bereits bestehenden Algorithmen könnten auch, statt mit den Eingabepunkten durch den Anwender, mit dreidimensionalen Punktwolken eines Gebäudes durchgeführt werden, die per Laserscanner, ähnlich wie bei Brenner und Haala [7], aufgenommen wurden. Mit den Punktwolken lässt sich eine Bounding-Box um das Objekt erstellen. Ecken und Kanten sind dann somit gegeben. Dieses Objekt kann dann direkt mit den Modellobjekten

im 3D Raum mit dreidimensionalen Punkten verglichen werden. Die Umwandlung der Zwischenergebnisse von 3D-Punkten in 2D-Punkte während der Rekonstruktion, um diese mit den 2D Eingabepunkten zu vergleichen, wäre dann nicht mehr nötig.

Da die Programmiersprache des Prototypen in Java geschrieben wurde, lassen sich die Implementationen einfach und schnell auf andere Plattformen übertragen, wie zum Beispiel auf das Android Betriebssystem. Als App könnte jeder, der ein Android Smartphone besitzt, sein eigenes Haus als 3D Objekt erstellen.

Literaturverzeichnis

[1] Abeles, P.: *Efficient Java Matrix Library (EJML)*. http://ejml.org. – zuletzt aufgerufen am 14.04.2019

[2] Albertz J., Wiedemann A. und Hemmleb M. und: Reconstruction Of Historical Buildings Based On Images From The Meydenbauer Archives, 08 2000

[3] Bauer J., Schindler K. und: A model-based method for building reconstruction. In: *First IEEE International Workshop on Higher-Level Knowledge in 3D Modeling and Motion Analysis, 2003. HLK 2003.*, 10 2003, S. 74–82

[4] Frahm J., Schönberger J. L. und: Structure-from-Motion Revisited. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 06 2016, S. 4104–4113. – ISSN 1063-6919

[5] Funkhouser T., Zeng A. und Song S. und Niessner M. und Fisher M. und Xiao J. und: 3DMatch: Learning Local Geometric Descriptors from RGB-D Reconstructions, 07 2017, S. 199–208

[6] Geman D., Geman S. und: Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-6 (1984), Nov, Nr. 6, S. 721–741. – ISSN 0162-8828

[7] Haala N., Brenner C. und: Erfassung von 3D Stadtmodellen. In: *PFG – Photogrammetrie, Fernerkundung, Geoinformation, Heft 2/2000* (2000), 01, S. 109–118

[8] K., Vetter A.: *Die besten Einfamilienhäuser aus Beton.* Callwey Verlag, 2017

[9] Malik J., Debevec P. E. und Taylor C. J. und: 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, NY, USA : ACM, 1996 (SIGGRAPH '96), S. 11–20. – URL http://doi.acm.org/10.1145/237170.237191. – ISBN 0-89791-746-4

[10] McConnell, R. K.: *Method of and apparatus for pattern recognition.* 07 1986. – U.S. Patent No. 4,567,610

[11] Oracle: *Java AWT Package Summary.* https://docs.oracle.com/javase/7/docs/api/java/awt/package-summary.html. – zuletzt aufgerufen am 14.04.2019

[12] Oracle: *Java Swing Package Summary.* https://docs.oracle.com/javase/7/docs/api/javax/swing/package-summary.html. – zuletzt aufgerufen am 14.04.2019

[13] P., Jenke: *Computergrafik Framework.* 10 2018. – Version vom 10. Oktober 2018

[14] P., Jenke: Einführung in die Computergrafik für Augmented Reality / Hochschule für Angewandte Wissenschaften Hamburg. 2018. – Forschungsbericht

[15] Ramanan D., Hejrati M. und: Analysis by Synthesis: 3D Object Recognition by Object Reconstruction. In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*, June 2014, S. 2449–2456. – ISSN 1063-6919

[16] Savarese S., Choy C. und Xu D. und Gwak J. und Chen K. und: 3D-R2N2: A Unified Approach for Single and Multi-view 3D Object Reconstruction. In: *CoRR* abs/1604.00449 (2016)

[17] Schindler K., Blaha M. und Vogel C. und Richard A. und Wegner J. D. und Pock T. und: Large-Scale Semantic 3D Reconstruction: An Adaptive Multi-resolution Model for Multi-class Volumetric Labeling. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 06 2016, S. 3176–3184. – ISSN 1063-6919

[18] Socher G., Nischwitz A. und Fischer M. und Haberäcker P. und: *Computergrafik und Bildverarbeitung, Band I: Computergrafik.* Vieweg+Teubner Verlag, 2011. – ISBN 978-3-8348-1304-6

[19] Swamy M.N.s, Du K.-L und: *Search and Optimization by Metaheuristics.* Birkhäuser Basel, 08 2016. – ISBN 978-3-319-41191-0

[20] Tang X., Xue T. und Liu J. und: Example-Based 3D Object Reconstruction from Line Drawings, 06 2012, S. 302–309. – ISBN 978-1-4673-1226-4

[21] Torralba A., Xiao J. und Russel B. und: Localizing 3D cuboids in single-view images. In: Pereira, F. (Hrsg.) ; Burges, C. J. C. (Hrsg.) ; Bottou, L. (Hrsg.) ; Weinberger, K. Q. (Hrsg.): *Advances in Neural Information Processing Systems*

25. Curran Associates, Inc., 2012, S. 746–754. – URL http://papers.nips.cc/paper/4842-localizing-3d-cuboids-in-single-view-images.pdf

[22] Triggs B., Dalal N. und: Histograms of oriented gradients for human detection. In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)* Bd. 1, June 2005, S. 886–893 vol. 1. – ISSN 1063-6919

[23] Ulbrich S., Ulbrich M. und: *Nichtlineare Optimierung*. Birkhäuser Basel, 2012. – ISBN 978-3-0346-0142-9

[24] Wiechert E., Furtwängler P. und: *Encyklopädie der Mathematischen Wissenschaften mit Einschluss ihrer Anwendungen*. Kap. Finsterwalder S. (1906) Photogrammetrie, Vieweg+Teubner Verlag, Wiesbaden, 1906

[25] Zimmermann U. T., Burkard R. E. und: *Einführung in die mathematische Optimierung*. Springer-Verlag, 2013

A Anhang

A.1 Bedienungsanleitung des Prototypen

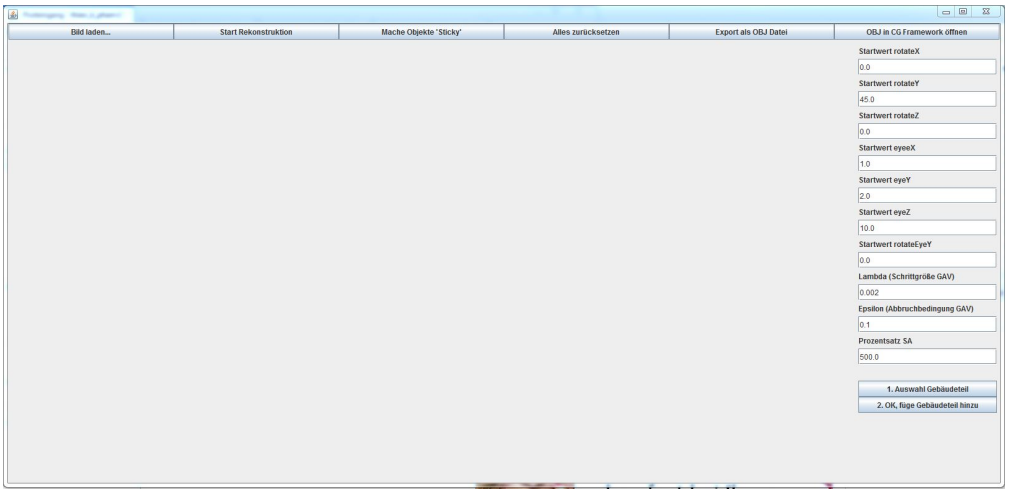

Abbildung A.1: GUI beim Start.

Beim Start des Programms ist die GUI noch leer (Abbildung A.1).

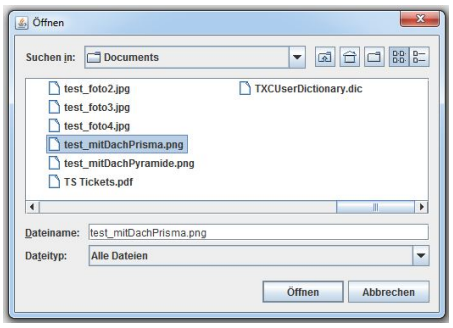

Abbildung A.2: Auswahl einer Bilddatei.

Mit Klick auf den Button „Bild laden...“ öffnet sich das Fenster für die Auswahl einer Bilddatei (Abbildung A.2).

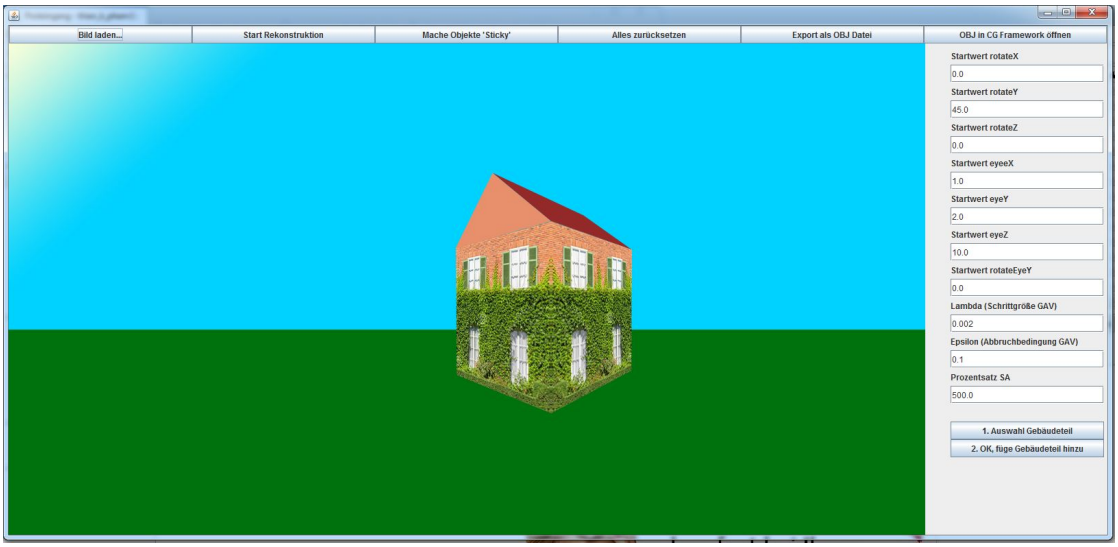

Abbildung A.3: Das Bild wurde in die Benutzeroberfläche geladen.

Danach erscheint das ausgewählte Bild in der Benutzeroberfläche (Abbildung A.3).

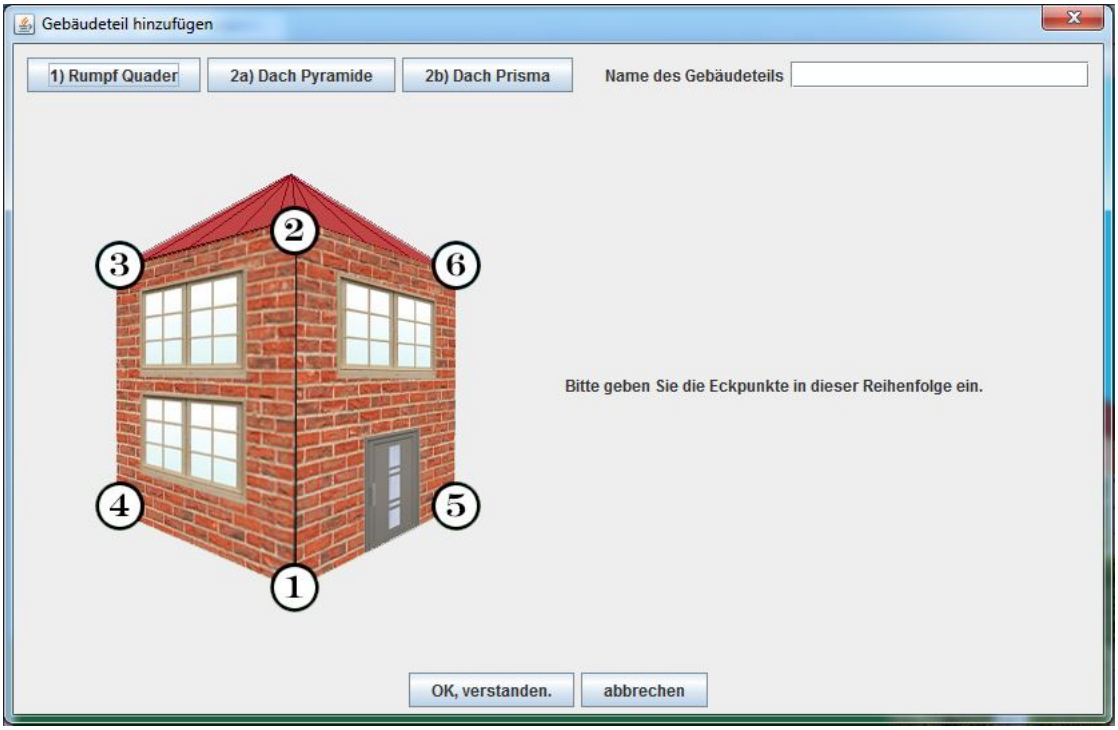

Abbildung A.4: Auswahl des Gebäudeteils, das in die Objektliste aufgenommen werden soll.

Mit Klick auf den Button „1. Auswahl Gebäudeteil“ in der Leiste rechts öffnet sich erneut ein Fenster (Abbildung A.4). Dort wird festgelegt, welches Objektteil in die Objektliste für die Rekonstruktion hinzugefügt werden soll. Die Auswahl eines bestimmten Objektteils erfolgt durch einen Klick auf den jeweiligen Button. Hier ist es wichtig, falls es im Gesamtobjekt einen Quader gibt, dass dieser als erstes hinzugefügt wird. Zu jedem Button der Objektteile gibt es auf der Bildfläche eine kurze Anleitung, in welcher Reihenfolge die Eckpunkte eingegeben werden müssen. Die Reihenfolge der Eingabe spielt eine entscheidende Rolle, da die Modellobjekte sich an diesen orientieren werden. Beim Prisma ist das zudem abhängig von der Seite, von der das Objekt betrachtet wird.

Ein Objektname kann, muss aber nicht in diesem Schritt eingegeben werden. Wenn kein Objektname angegeben wird, heißt das Objekt nur „Gebaeudeteil“, gefolgt von einer inkrementierenden Zahl.

Mit Klick auf „OK, verstanden“ wird die Bildoberfläche im Hauptfenster freigeschaltet und die Punkteeingabe ist nun möglich.

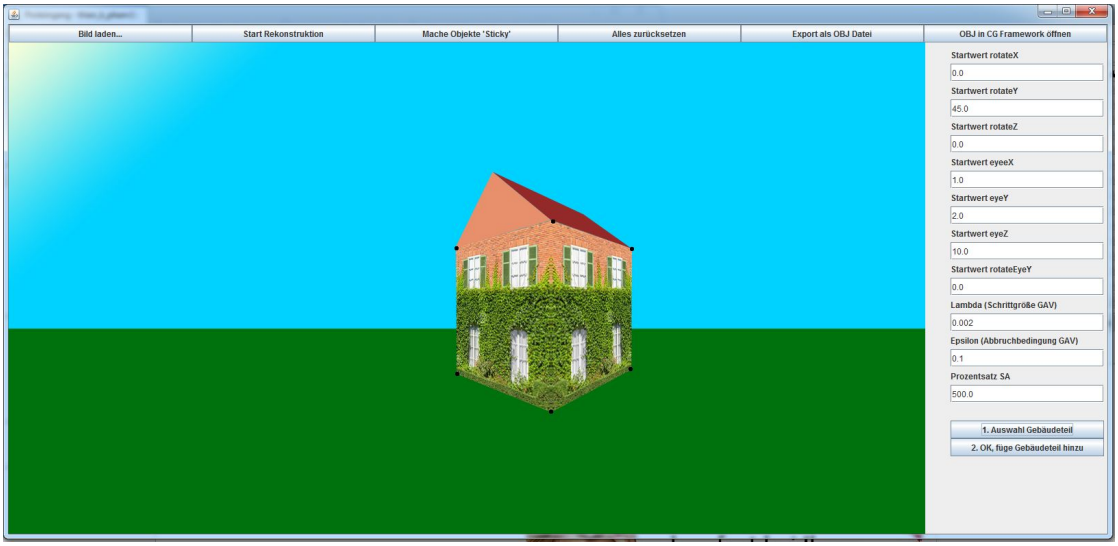

Abbildung A.5: Auswahl Eckpunkte eines Gebäudeteils.

Die Punkte werden nun in der vorher gezeigten Reihenfolge per Mausklick eingegeben. Anschließend wird das Objektteil mit dem Button „2. OK, füge Gebäudeteil hinzu“ (rechts unten) in die Objektliste hinzugefügt. Gleichzeitig erscheint unter diesem Button der Name des Objektteils.

Es können beliebig viele Objekte hinzugefügt werden. Für die Löschung aller Objekte aus der Objektliste gibt es den Button „Alles zurücksetzen“. Dieser löscht alle bisher eingegebenen Punkte und Objekte, das zuvor hinein geladene Bild bleibt jedoch bestehen.

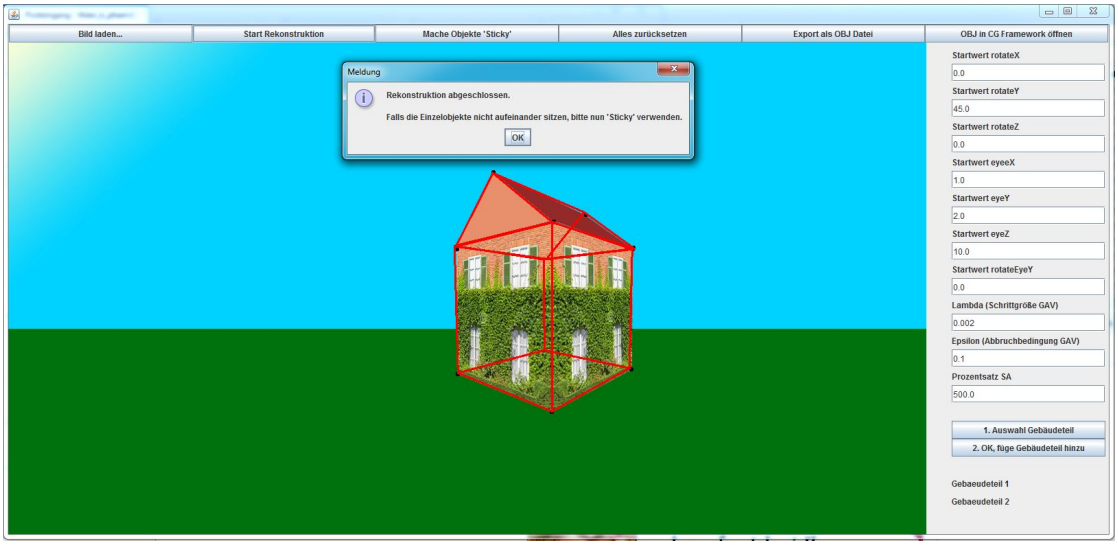

Abbildung A.6: Erfolgreiche Rekonstruktion des Objekts.

Während der Rekonstruktion sind Linienobjekte in rot (Gradientenabstiegsverfahren) und gelb (verbessertes Ergebnis eines Simulated Annealing Durchlaufs) zu sehen (Abbildung A.6). Diese sind die Modellobjekte, die sich an die eingegebenen Objektteile annähern. Man kann hier also die Annäherung live verfolgen.

Nach der Rekonstruktion erscheint eine Info, dass die Einzelobjekte möglicherweise nicht aneinander haften und dass im Anschluss „Sticky Objects“ verwendet werden soll (Button „Mache Objects 'Sticky'“). Dies wird etwas Zeit in Anspruch nehmen. Sticky Objects ist vollständig, wenn ein kleines Fenster erscheint:

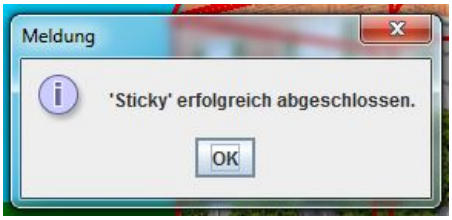

Abbildung A.7: Erfolgreiche Ausführung von Sticky Objects.

Mit Klick auf „Export als OBJ-Datei“ wird das rekonstruierte Objekt (mit allen Objektteilen) als OBJ Datei exportiert. Die dazugehörigen Texturen und die Material-Datei wird in diesem Schritt parallel ausgeführt. Ist der Export erfolgreich, erscheint folgendes Fenster:

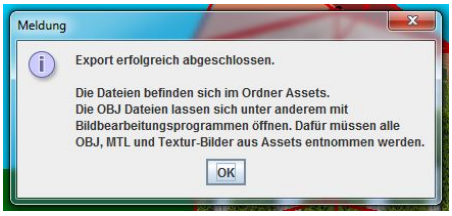

Abbildung A.8: Erfolgreicher Export des Objekts als OBJ-Datei.

Danach kann das Objekt mit dem HAW Computergrafik Framework geöffnet werden. Dazu erfolgt ein Klick auf den Button „OBJ in CG Framework öffnen“ im Hauptfenster der Benutzeroberfläche (oben rechts).

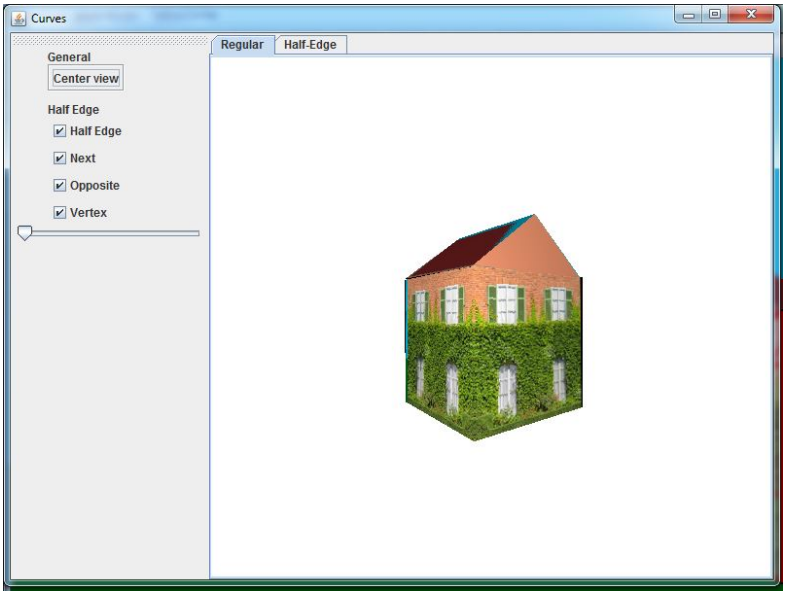

Abbildung A.9: Objektdatei mit Textur im HAW Computergrafik Framework.

Um das Objekt mittig und an das Fenster angepasst zu betrachten, muss der Button „Center view“ betätigt werden. Dann kann das Objekt heran- und herausgezoomt und von allen Seiten betrachtet werden.

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „– bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] – ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen."

Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: _____

Vorname: _____

dass ich die vorliegende Mastertarbeit – bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

Bildbasierte Rekonstruktion von Gebäuden durch Fitting volumetrischer Körper

ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

_____	_____	_____
Ort	Datum	Unterschrift im Original