# Buildings from Images - Beispielbasierte Rekonstruktion von Gebäuden aus Eizelbildern

## Schriftliche Ausarbeitung Hauptseminar WiSe 16/17

Thien Pham

Hochschule für Angewandte Wissenschaften Hamburg

# 1 Einführung

Diese Arbeit beschäftigt sich mit der beispielbasierten Rekonstruktion von Gebäuden. Es wird bis zum Ende der Masterarbeit ein Programm entstehen, das ein zweidimensionales Bild eines Gebäudes als Eingabe bekommt und nach Prozessierung als Ausgabe dieses Gebäude als dreidimensionales Objekt zurückgibt. Für das Programm wird eine Datenbank erstellt, in der verschiedene dreidimensionale Objekte gespeichert sind. Das eingegebene Bild kann mit diesen bereits vorhandenen Objekten verglicken werden, um die Rekonstruktion zu beschleunigen. Es werden keine für das Programm bzw. der Datenbank fremden Objekte während der Prozessierung erstellt.
Im Grundprojekt geht es speziell um die Rekonstruktion von zweidimensionalen Objekten, das heißt, ein zweidimensionales Objekt (z.B. Viereck) wird dem Programm gegeben und ein ebenso zweidimensionales Objekt wird zurückgegeben. Dazwischen ermittelt das Programm das passende gleichförmige Objekt aus der Datenbank und verändert dessen Form so, dass dieses (fast) mit dem Objekt der Eingabe übereinstimmt. Das neu berechnete Objekt ist die Ausgabe.
Das Grundprojekt bildet die Grundlage für das zukünftige Hauptprojekt und die darauffolgende Masterarbeit. Methoden vom Grundprojekt werden später übernommen und für die Berechnungen im dreidimensionalen Raum erweitert. Daher werden hier die Ziele und die Konstruktion des Grundprojekts genau erläutert.

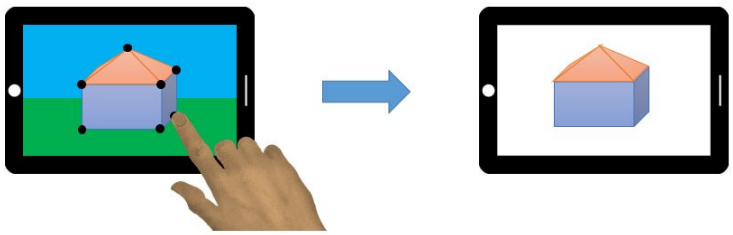

**Abb. 1.** In dem zweidimensionalen Eingabebild gibt der Benutzer die Eckpunkte ein und er erhält als Resultat das dreidimensionale Objekt zurück.

# 2 Grundprojekt

## 2.1 Ziele

Das Ziel des Grundprojekts ist es, zweidimensionale Formen zu rekonstruieren. Formen sind in diesem Fall zum Beispiel Drei- oder Vierecke, ebenjene, die größtenteils an Gebäuden zu finden sind. Die Idee dabei ist, zunächst der Einfachheit halber sich auf zweidimensionale Objekte zu beschränken und das Konstrukt

im Hauptprojekt auf dreidimensionale Objekte auszuweiten. Allerdings können nicht alle Formen bzw. Gebäudearten berücksichtigt werden. Ausgefallene Architekturen, wie z.B. die Elbphilharmonie, haben spezielle Formen, für die eine genaue Rekonstruktion dieses Projekt sprengen würde.

### 2.2 Grundlagen für die Berechnung

Das Programm besteht aus folgenden Komponenten:

- Eingabe: Das Programm nimmt ein zweidimensionales Bild als Eingabe. Um dieses Projekt nicht zu überdimensionieren, wird angenommen, dass die Eckpunkte der Formen bereits erkannt wurden. Diese werden per Mausklick händisch eingetragen. Eine automatische Bild- bzw. Eckpunkterkennung findet hier nicht statt und ist nicht Teil der Arbeit.
- Datenbank: in der Datenbank sind die zu vergleichenden Formen hinterlegt. Bei der Eingabe vergleicht das Programm die Eingabedaten mit den Inhalten aus der Datenbank und gibt das Objekt aus dieser zurück, das den gegebenen Daten am ähnlichsten ist. Mit Inhalten sind Anzahl Eckpunkte, die Eckpunkte selbst und Höhe und Breite des Objekts gemeint.
- Die Ausgabe ist ein neues Bild, in dem die Form aus der Eingabe als Rekonstruktion abgebildet ist.

*Korrespondenzen*
Wie bereits oben erwähnt, wird in diesem Projekt der Fokus auf die präzise Rekonstruktion der Objekte gelegt. Um (Eck-)Punkte 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 „links oben“ bei Vierecken. Bei der manuellen Eingabe der Eckpunkte ist daher die Reihenfolge zu beachten, damit die „richtigen“ Punkte miteinander korrespondieren. Ein Beispiel ist in Abbildung 2 zu sehen. Dort sind die Objekte aus der Datenbank mit den Ecken a, b, c... versehen. Die Objekte aus der Eingabe heißen a', b', c'... und korrespondieren mit den gleichnamigen Ecken aus der Datenbank, die aber kein Apostroph im Namen besitzen (blaue Pfeile zeigen Korrespondenzen).

### 2.3 Berechnung im 2D

Das Programm ermittelt zunächst, wie viele Punkte vom Benutzer eingegeben worden sind und entnimmt dann auf dieser Grundlage aus der Datenbank das Objekt mit derselben Anzahl der gegebenen Punkte. Nachdem ein Objekt aus der Datenbank gefunden wurde, das von der Struktur her dem Objekt aus der Eingabe gleicht, müssen der Ort und die Form noch angepasst werden. Mithilfe

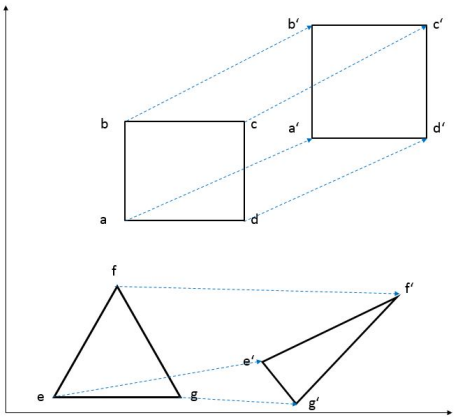


**Abb. 2.** Beispiel von Korrespondenzen. Punkt a aus der Datenbank korrespondiert mit Punkt a', b mit b', usw.

der numerischen Mathematik werden sollen die Eckpunkte aus der Datenbank an die Eingabe-Punkte angenähert werden. Dafür wird folgende Formel verwendet:

$$x_f = B * P_p * P_v * x_m. \tag{1}$$

$x_f$ ist der Featurepunkt, an den der korrespondierende Punkt $x_m$ des Modells aus der Datenbank angenähert werden soll. Um die Bildkoordinaten anzupassen (z.B. Nullpunkt in die Bildmitte verschieben), wird die Bildmatrix B benötigt. $P_p$ ist die Projektionsmatrix und $P_v$ die Verschiebungs- bzw. Translationsmatrix.

*Fehler und Fehlerfunktion*
Nach der Anwendung der Formel 1 bleibt, durch die numerische Annäherung, ein kleiner Fehler zurück, der durch den Betrag der Subtraktion von $x_f$ und $x_m$ beschrieben wird:

$$||x_f - x_m||. \tag{2}$$

Um ein gesamtes Objekt zu bewerten, das mehrere Eckpunkte bzw. Korrespondenzen ($i = 1..n$)hat, muss der quadratische Fehler (damit keine positiven und negativen Werte addiert werden) über alle Korrespondenzen betrachtet werden:

$$f(\delta) = \sum_{i=1}^{n} (x_f^i - B * P_p * P_v(\delta) * x_m^i(\delta))^2. \tag{3}$$

$P_v$ und $x_m^i$ sind abhängig vom Vektor $\delta$. Dieser enthält die Parameter, mit denen sich der Modellpunkt verschieben und skalieren lässt, also die Werte des Richtungsvektors aus der Translationsmatrix $P_v$ und die Höhe und Breite des Objekts in $x_m^i$. Gesucht wird ein $\delta$, womit die Fehlerfunktion $f(\delta)$ minimal wird.

### 2.4 Beispiel im 2D

Wie bereits erwähnt, wird im Grundprojekt zunächst die Rekonstruktion im zweidimensionalen Raum durchgeführt. In diesem Beispiel wird ein Rechteck betrachtet. Die Koordinaten der Eckpunkte für das Rechteck aus Abbildung 4 in der Datenbank sehen wie folgt aus:

$$\begin{aligned} a &= (-0.5s_x, -0.5s_y) \\ b &= (-0.5s_x, 0.5s_y) \\ c &= (0.5s_x, 0.5s_y) \\ d &= (0.5s_x, -0.5s_y) \end{aligned} \tag{4}$$

Das Rechteck sitzt um den Nullpunkt herum und die Höhe und Breite können mit $s_y$ und $s_x$ variiert werden. $s_y$ und $s_x$ sind Teil von $\delta$.

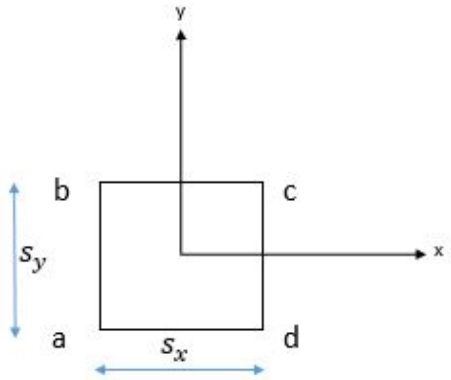


**Abb. 3.** Rechteck aus der Datenbank. Höhe und Breite lassen sich mittels Parameter dynamisch verändern.

In Abbildung 4 werden die Korrespondenzen anhand zweier Rechtecke veranschaulicht. Dabei ist $a = (a_x, a_y, 1)^T$ der Vektor für die homogenen Koordinaten aus der Datenbank. a' ist der Featurepunkt, an den eine Annäherung stattfinden wird. Die Formel 5 ist ein Anwendungsbeispiel der Formel 1. Das Vorgehen der Annäherung wird in Kapitel 2.5 beschrieben, nachdem im folgenden zunächst die wichtigen Matrizen und Vektoren erklärt werden.

$$a' = B * P_p * P_v * a. \tag{5}$$

In dieser Arbeit werden nur homogene Koordinaten verwendet. Während ein zweidimensionaler Punkt im euklidischen Koordinatensystem durch einen Ortsvektor mit zwei Komponenten dargestellt werden kann, bekommt ein solcher

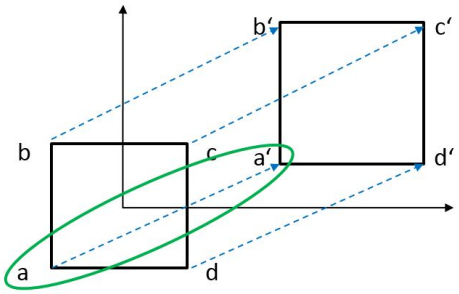


**Abb. 4.** Beispiel von Korrespondenzen. Punkt a aus der Datenbank soll mithilfe der Formel 5 an a' angepasst werden.

Punkt im homogenen Koordinatensystem eine dritte Komponente, dem sogenannten inversen Streckungsfaktor. Dieser sorgt dafür, dass auch Translationen durchgeführt werden können, da die Translationsmatrix im zweidimensionalen Raum eine 3x3 Matrix ist (im dreidimensionalen Raum 4x4). Der inverse Streckungsfaktor ist in einem Ortsvektor, z.B. a aus Beispiel oben, in der untersten Zeile und hat standardmäßig den Wert 1. Damit alle Rechnungen korrekt verlaufen können, werden parallel dazu alle Matrizen um eine weitere Zeile und Spalte erweitert, sodass man im zweidimensionalen Raum 3x3 Matrizen für alle Berechnungen hat. In der Formel 1 ist die Matrix $P_v$ die Translationsmatrix, die den Richtungsvektor $(t_x, t_y)$ enthält:

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

B ist die Bildmatrix. Diese dient dazu, den Nullpunkt bei der Anzeige durch OpenGL von oben links in die Bildmitte zu verschieben. $w$ ist die Breite und $h$ die Höhe des Bildes.

$$B = \begin{pmatrix} 1 & 0 & w/2 \\ 0 & 1 & h/2 \\ 0 & 0 & 1 \end{pmatrix}$$

$P_p$ ist die Projektionsmatrix. Projektionsmatrizen dienen dazu, Abbildungen aus einer dreidimensionalen Szene zur Anzeige auf einem zweidimensionalen Bildschirm zu transformieren. Da wir uns im zweidimensionalen Raum befinden, muss nichts in dieser Form abgebildet werden. Somit sieht die Projektionsmatrix wie folgt aus:

$$P_p = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

In dieser Form beeinflusst die Projektionsmatrix die Rechnung aus der Formel 1 nicht. Trotzdem befindet sie sich in der Formel, damit diese sich bei der Umstellung auf die dreidimensionale Ebene im Hauptprojekt strukturell nicht verändert.

### 2.5 Optimierung

Die Werte für den Parametervektor $\delta$ sollen für die Formel 1 optimiert werden. Der Parametervektor ist hier:

$$\delta = (t_x, t_y, s_x, s_y).$$

Für die Optimierung wird das Gradientenabstiegsverfahren verwendet. Das ist ein iteratives Verfahren aus der numerischen Mathematik, um schrittweise sich an ein Minimum anzunähern. Das Minimum ist hier der kleinste Fehlerwert, den die Formel 3 erzeugen kann. Die Formel für das Gradientenabstiegsverfahren sieht wie folgt aus:

$$\delta_{i+1} = \delta_i - \alpha \Delta f(\delta). \tag{6}$$

Der Anfangswert ist $\delta_0 = (0, 0, 0, 0)$ und $\alpha$ ist die Schrittgröße beim Gradientenabstieg (z.B. 0,1). Der Gradient wird mittels Differenzenquotienten für ein kleines h (z.B. $10^-5$) ermittelt:

$$\Delta f(\delta) = \frac{f(\delta + h) - f(\delta)}{h}.$$

### 2.6 Herausforderungen

Das Gradientenabstiegsverfahren birgt den Nachteil, dass der Algorithmus stoppt, nachdem ein Optimum gefunden hat. Falls es mehrere lokale Optima gibt, kann es sein, dass man als Ergebnis zwar ein Optimum bekommt, aber nicht das richtige bzw. globale. Für das Grundprojekt wird angenommen, dass das Verfahren ein akzeptables Ergebnis liefert. Im laufe der weiteren Projekte wird sich zeigen, ob dieses Verfahren geeignet ist.
Ein anderes Problem könnte die Einbeziehung des Benutzers sein, der die Eckpunkte des Eingabebildes manuell erfassen soll. Es setzt voraus, dass der Benutzer technikaffin ist. Die automatische Bild- bzw. Eckpunkterkennung ist zwar nicht Inhalt dieser Arbeit, aber sich nur darauf zu verlassen, dass der Benutzer die Punkte in der richtigen Reihenfolge eingibt, könnte schnell fehleranfällig werden. Um dies vorzubeugen, muss im Endprodukt dem Benutzer genügend übersichtliche Hilfestellungen gegeben werden, beispielsweise zum Start des Programms eine interaktive Anleitung oder eine feste Leiste an der Seite, in der das Vorgehen visuell beschrieben oder animiert wird. Dennoch werden Punkte, die vom Benutzer eingegeben werden, für die Verarbeitung genommen, auch wenn diese nicht millimetergenau ist.

Die manuelle Eingabe der Eckpunkte wird später, wenn große Gebäude rekonstruiert werden sollen, schnell komplex werden. Möglicherweise wird es dem Benutzer nicht mehr gelingen, die Eckpunkte sauber und vollständig zu erfassen.
Diese genannten Risiken werden im weiteren Verlauf der Projekte weiter beobachtet und gleichzeitig wird für sie Lösungen gesucht.

### 2.7 Entwicklung

*Programm*
Das Programm wird für den Windows-Desktop-PC entwickelt. Entwicklung erfolgt in der Programmiersprache Java. Dadurch können, über das Projekt hinaus, die Methoden schnell auf weitere Plattformen wie z.B. Android zu übertragen werden. Über die Benutzeroberfläche und Visualisierung mittels OpenGL 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.
Neben OpenGL werden zwei weitere Bibliotheken verwendet: JavaFX für das Auslesen aus der XML-Datei, die als Datenbank mit den Objekten dient, und EJML für die Matrix-Berechnungen. EJML hat bei den Benchmarks zu den für dieses Projekt relevanten Berechnungen in der linearen Algebra am besten abgeschnitten[1].

*Datenbank*
Die Datenbank mit den Objekten ist eine XML-Datei. Momentan sind dort ein Kreis (mit Mittel- und Außenpunkt), ein Drei- und ein Viereck gespeichert. Damit das Programm sich das richtige Objekt aus der Datenbank nimmt, sind bei den Objekten jeweils eine Info „Number of Vertices“ gespeichert, eine 0 für den Kreis, 3 für ein Dreieck usw. Außerdem sind zu den Punkten noch Höhe und Breite (normalisiert auf 1) gespeichert. Ein Rechteck sieht zum Beispiel so aus:

```
<Rectangle>
<NoOfVertices>4</NoOfVertices>
<Point0_x>0.5</Point0_x>
<Point0_y>-0.5</Point0_y>
<Point1_x>0.5</Point1_x>
<Point1_y>0.5</Point1_y>
<Point2_x>-0.5</Point2_x>
<Point2_y>0.5</Point2_y>
<Point3_x>-0.5</Point3_x>
<Point3_y>-0.5</Point3_y>
<Height>1</Height>
<Width>1</Width>
</Rectangle>
```

[1] Performance: Java Matrix Benchmark / EJML. Abgerufen am 06.03.2017 von http://ojalgo.org/performance_ejml.html

# 3 Hauptprojekt

Im Hauptprojekt sollen die implementierten Methoden aus dem Grundprojekt übernommen und erweitert werden.

*1 - Erweiterung der Datenbank*
Bisher sind nur zweidimensionale Objekte in der Datenbank gespeichert. Im Hauptprojekt werden zusätzlich dreidimensionale Objekte hinzugefügt. Wie schon im Grundprojekt werden hier auch nur Objekte verwendet, deren Formen meist an Gebäuden verwendet werden (z.B. Quader und Prismen). Dazu wird nicht nur Höhe und Breite, sondern auch Tiefe und gegebenenfalls die Winkel gespeichert. Da bei einem Objekt, das nicht transparent ist, bestimmte Punkte auf der Rückseite nicht einsehbar sind, muss der Ansatz, die Anzahl der Ecken in der Datenbank zu speichern, überarbeitet werden (siehe Abschnitt 3).

*2 - Erweiterung der Formeln und Matrizen*
Auch die Vektoren und Matrizen sehen in einem dreidimensionalen, homogenen Koordinatensystem anders aus[2]. Jeder Vektor und jede Matrix bekommt eine zusätzliche Komponente:

- Die Translationsmatrix sieht im 3D wie folgt aus:

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

- Die Bildmatrix zum Verschieben des Nullpunktes wird für die Erweiterung der Formel 1 nicht mehr gebraucht, da die perspektivische Projektion (nächster Punkt) sich um die gewünschte Anzeige auf dem Bildschirm kümmert.
- Die Projektionsmatrix.

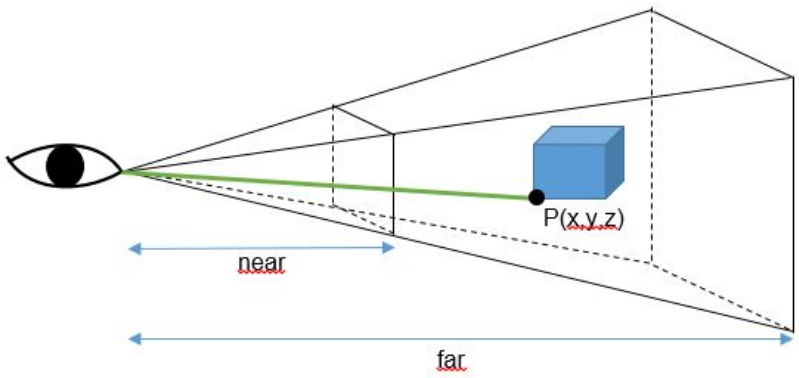


**Abb. 5.** Perspektivische Projektion. Die liegende Pyramide ist die sichtbare Ebene, alle Objekte, die drüber hinaus liegen, werden abgeschnitten.

Um die Objekte aus der dreidimensionalen Koordinatensystem auf den zweidimensionalen Bildschirm abzubilden, wird die Projektionsmatrix benötigt. In OpenGL ähnelt die perspektivische Sicht der natürlichen Wahrnehmung des Auges (siehe Abbildung 5). Es gibt verschiedene Begrenzungspunkte: die sogenannte „far clipping plane“ ist der Boden des Pyramidenstumpfes, die „near clipping plane“ bildet die vordere Begrenzung und die vier Seiten der Pyramide (top, bottom, left, right) sind die weiteren Begrenzungen. Alle Objekte, die über die Begrenzungen hinaus liegen, werden auf dem Bildschirm nicht abgebildet. „Near“ und „Far“ lassen sich verschieden variieren, je nachdem, welchen Blickwinkel man dem Benutzer geben möchte. Ein kleiner Wert für „Near“ gibt dem Benutzer ein weites, sichtbares Volumen. Ein großer Wert würde das sichtbare Volumen für den Benutzer einengen. „Far“ bildet die Sichtweite, z.B. 1000 Meter.
OpenGL bietet für die perspektivische Projektion die Methode mit den genannten Begrenzungen als Parameter:

```
glFrustum(GLdouble left, GLdouble right, GLdouble
    bottom, GLdouble top, GLdouble near, GLdouble far);
```

Diese Methode verwendet folgende Matrix[2] für die perspektivische Projektion:

$$P_p = \begin{pmatrix} \frac{2}{right-left} & 0 & \frac{1}{near}\frac{right+left}{right-left} & 0 \\ 0 & \frac{2}{top-bottom} & \frac{1}{near}\frac{top+bottom}{top-bottom} & 0 \\ 0 & 0 & -\frac{1}{near}\frac{far+near}{far-near} & -\frac{2far}{far-near} \\ 0 & 0 & -\frac{1}{near} & 0 \end{pmatrix}$$

Neben diesen Erweiterungen gibt es für die erweiterung der Formel 1 noch zusätzlich drei verschiedene Rotationsmatrizen:

$$\begin{aligned} P_{rx} &= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\alpha & -sin\alpha & 0 \\ 0 & sin\alpha & cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \\ P_{ry} &= \begin{pmatrix} cos\beta & 0 & sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -sin\beta & 0 & cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \\ P_{rz} &= \begin{pmatrix} cos\gamma & -sin\gamma & 0 & 0 \\ sin\gamma & cos\gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{aligned} \tag{7}$$

[2] Nischwitz, A., Fischer, M., Haberäcker, P., Socher, G.: Computergrafik und Bildverarbeitung, Band I: Computergrafik. 3. Auflage (2011), Seite 145.

$P_{rx}$ ist dabei die Matrix für die Rotation um die x-Achse, $P_{ry}$ für die Rotation um die y-Achse, und $P_{rz}$ um die z-Achse.

Damit sieht die erweiterte Formel so aus:

$$x_f = P_{rx} * P_{ry} * P_{rz} * P_p * P_v * x_m. \tag{8}$$

Auch der Parametervektor $\delta$ hat sich erweitert um die dritte Komponente aus dem Richtungsvektor in der Translationsmatrix $P_v$, der Tiefe des Objekts aus der Datenbank ($s_z$), und den Winkelwerten aus den drei Rotationsmatrizen aus 7.

$$\begin{aligned} \delta = (t, s, \alpha, \beta, \gamma), mit \\ t = (t_x, t_y, t_z), und \\ s = (s_x, s_y, s_z). \end{aligned}$$

*3 - Herausforderungen*

Wie im Abschnitt 3 bereits erwähnt, reicht es nicht aus, allein die Gesamtzahl der Eckpunkte eines dreidimensionalen Objekts zu speichern. Wenn der Nutzer bei nicht-transparenten Objekten nicht alle Eckpunkte sehen und erfassen kann, bekommt das Programm bei der Eingabe möglicherweise weniger Eckpunkte als das Objekt in Wirklichkeit hat. Mögliche Lösung wäre die Annahme, dass das Objekt, was der Nutzer eingibt, symmetrisch ist. Die Rückseite des Objekts wäre dann nur eine Kopie der Vorderseite. Dann muss in der Datenbank Zusätzlich zu der Information über die insgesamte Anzahl der Eckpunkte noch alle möglichen sichtbaren Anzahl Eckpunkte gespeichert werden, da beispielsweise ein Quader aus verschiedenen Perspektiven entweder 6 oder 7 Eckpunkte haben kann (siehe Abbildung 6).

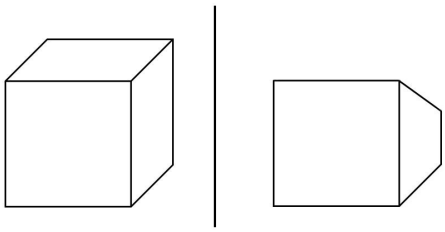

**Abb. 6.** Unterschiedliche Perspektiven auf einen Quader gibt entweder 6 oder 7 sichtbare Eckpunkte.

## 4 Ausblick - Masterarbeit

In der Masterarbeit werden die Objekte, die mithilfe der Formel 8 aus dem Hauptprojekt einzeln rekonstruiert wurden, zu einem vollständigen Objekt zu-

sammengesetzt. Herausforderung dabei ist die Frage, welche Objekte an welchen Ecken aneinander gehören.
Das Ausgabe-Objekt soll zudem eine Textur bekommen, die aus dem Eingabebild extrahiert und in die Rekonstruktion integriert wird. Die Annahme hier wäre, wie schon im Hauptprojekt bzgl. der Gesamtzahl der Eckpunkte, dass das Objekt symmetrisch ist. Eine Kombination aus zwei oder mehr Bildern als Eingabe ist nicht vorgesehen.

## Literatur

1. Pham, T., Jenke, P. „Konzept - Buildings From Images - Projekt 1“. (2016)
2. Nischwitz, A., Fischer, M., Haberäcker, P., Socher, G.: Computergrafik und Bildverarbeitung, Band I: Computergrafik. 3. Auflage (2011)