Fernfachhochschule Schweiz
Zürich | Basel | Bern | Brig
Mitglied der SUPSI

Fernfachhochschule Schweiz
BSc Informatik

Zentralprojektion

Semesterarbeit im Modul LinAlg, des Studiengangs BSc Informatik

von

Tobias Gasche

Dozent:
Prof. Dr. Matthias Dehmer

4554 Etziken, 26. Oktober 2024

Zusammenfassung

Diese Arbeit behandelt die Zentralprojektion von Objekten aus dem dreidimensionalen Raum $\mathbb{R}^3$ in den zweidimensionalen Raum $\mathbb{R}^2$.
Dabei werden die Grundlagen und Definitionen der linearen, sowie der affinen Abbildung besprochen.
Es wird anhand eines Beispiels diskutiert, weshalb die Zentralprojektion eine affine Abbildung ist.
Weiter wird die Abbildungsmatrix für die Zentralprojektion hergeleitet.

Im Anschluss wird die Theorie mit Python vertieft und praktisch angewendet.

Inhaltsverzeichnis

1 Einleitung

1.1 Die Zentralprojektion und ihre Bedeutung

Mit Hilfe der Zentralprojektion werden dreidimensionale Objekte in zweidimensionalen Ebenen dargestellt.
Will zum Beispiel ein Maler ein Objekt auf eine Leinwand zeichnen, so nimmt er eine Zentralprojektion vor.
Man stelle sich vor, der Maler schiesse durch die vor ihm stehende Leinwand Strahlen. Nun betrachtet er jeweils Punkte seines Sujets und hinterlässt aufgrund der Strahlen Abdrücke auf der Leinwand. So wird das dreidimensionale Objekt durch den Augenpunkt auf die zweidimensionale Ebene projiziert.
Eine eindrucksvolle Darstellung dieser Projektion bietet das Gemälde von Albrecht Dürer «Ein Mann zeichnet eine Laute». (Abb. 1)

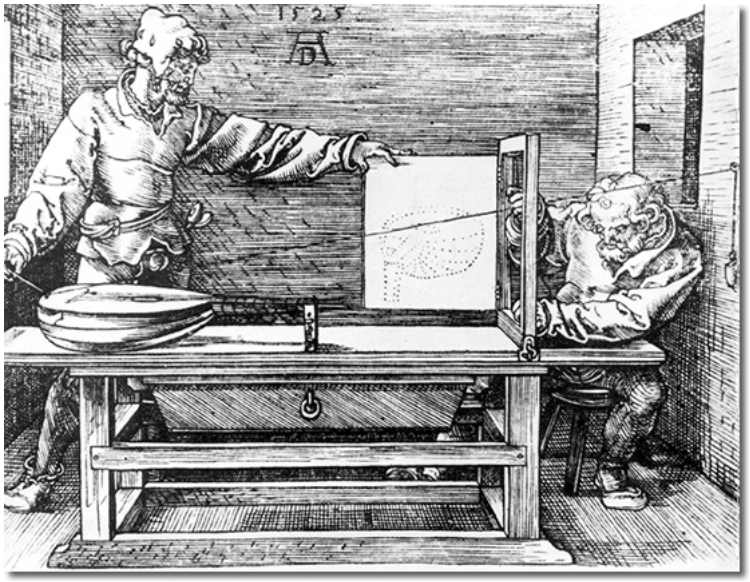

Abb. 1 Albrecht Dürer - Ein Mann zeichnet eine Laute (Quelle: https://de.m.wikipedia.org/wiki/Datei:358durer.jpg, 07.10.2024, 21:50 Uhr)

Der Maler kann seine Leinwand aus dem Rahmen kippen und eine Schnur zum Objekt spannen. Der Augenpunkt ist in diesem Falle der Hacken an der Wand. Der Punkt, an welchem die Schnur die Leinwandebene durchsticht, wird markiert, die Schnur gelockert, die Leinwand wieder in den Rahmen bewegt und anschliessend kann der Punkt aufgemalt werden. Mit diesem Vorgehen wird Punkt für Punkt des zu malenden Objekts auf die Leinwand übertragen und es entsteht eine für den Betrachter realistische und perspektivisch korrekte Abbildung einer Laute.

Durch das Beispiel mit Dürers Werk wird die Bedeutung der Zentralprojektion für die Malerei hinreichend diskutiert. Die Idee, Dürer als Beispiel zu nennen, wurde [1] entnommen. Aber auch in der Architektur spielt die Zentralprojektion eine wichtige Rolle. Ohne sie wäre das Zeichnen von Plänen unvorstellbar.

1.2 Zentralprojektion in der Informatik

In der Informatik betrifft das Thema Zentralprojektion die Darstellung räumlicher Tiefe und dreidimensionaler Objekte sämtliche grafische Themenbereiche.

Jede Computergrafik wird an einem Bildschirm zweidimensional dargestellt. Das heisst, jedes Objekt aus $\mathbb{R}^{>2}$ muss auf $\mathbb{R}^2$ projiziert werden. Es gilt dabei zu beachten, dass der Augenpunkt möglichst dem Augenpunkt des Anwenders entsprechen soll um eine realistische Perspektive zu schaffen [2].
Da die Projektionsebene (der Bildschirm) zwischen dem Betrachter und dem betrachteten Umfeld (dem darstellbaren Bereich) befindet, ergibt sich ein unendlich grosser darstellbarer Bereich. Extrem nahe Objekte würden den kompletten Bildschirm füllen, extrem weit entfernte wären so klein, dass eine Darstellung keinen Sinn mehr ergeben würde.
Aus diesen Gründen wird eine vordere und eine hintere «Clippingebene» definiert, welche den betrachteten Raum begrenzen. Dadurch ergibt als darstellbarer Bereich ein Pyramidenstumpf. [2]

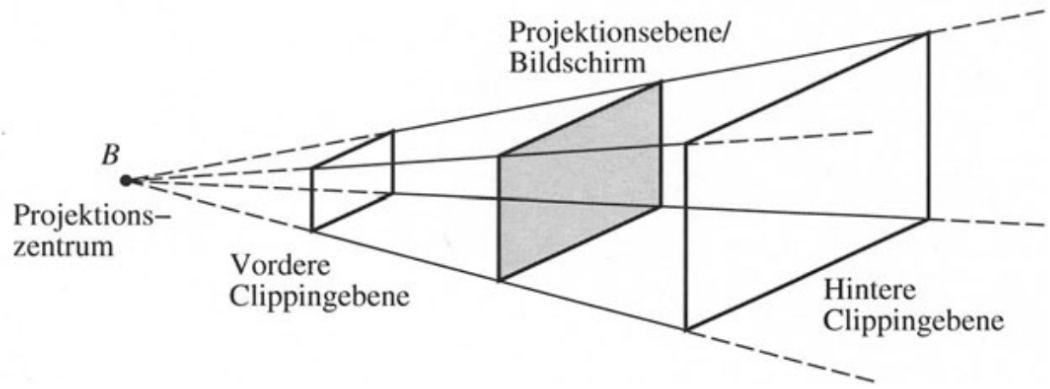

Abb. 2 Clippingebenen und Pyramidenstumpf Quelle: [2]

2 Theorie

In diesem Kapitel werden die Theoretischen Grundlagen für die Zentralprojektion behandelt. Kapitel 2.1 definiert die Eigenschaften linearer Abbildungen, während Kapitel 0 die Eigenschaften affiner Abbildungen erläutert.
In Kapitel 0 erfolgt die Diskussion, weshalb die Zentralprojektion eine affine Abbildung ist.
In Kapitel **Error! Reference source not found.** wird schliesslich die Darstellungsmatrix der Zentralprojektion hergeleitet.

2.1 Definition und Eigenschaften der linearen Abbildung

Nach [3] gilt: Eine Abbildung f heisst linear, wenn für alle Vektoren u und v sowie für alle Skalare λ gilt:

$$f(u+v) = f(u) + f(v)$$
$$f(\lambda v) = \lambda f(v)$$

Das heisst konkret, eine Abbildung auf die Addition zweier Vektoren führt zum selben Resultat, wie wenn die einzelnen Vektoren zuerst abgebildet worden wären und anschliessend die Addition erfolgte.
Für die Skalare Multiplikation ist definiert, dass die Abbildung des multiplizierten Vektors zum selben Resultat führt, wie die Multiplikation des abgebildeten Vektors.
Sei $u = \begin{pmatrix} a \\ c \end{pmatrix}, v = \begin{pmatrix} b \\ d \end{pmatrix}$ und $f(w) = \begin{pmatrix} -x \\ y \end{pmatrix}$ so ergibt sich in der Addition vor der Abbildung

$$u + v = \begin{pmatrix} a+b \\ c+d \end{pmatrix}$$

Das Resultat mit $f(w) = \begin{pmatrix} -x \\ y \end{pmatrix}$ abgebildet ergibt

$$f(u+v) = f\begin{pmatrix} -(a+b) \\ c+d \end{pmatrix} = \begin{pmatrix} -a-b \\ c+d \end{pmatrix}$$

Werden hingegen die Vektoren zuerst abgebildet und anschliessend addiert ergibt sich folgendes:

$$f(u) = \begin{pmatrix} -a \\ c \end{pmatrix}, f(v) = \begin{pmatrix} -b \\ d \end{pmatrix}$$
$$f(u) + f(v) = \begin{pmatrix} -a \\ c \end{pmatrix} + \begin{pmatrix} -b \\ d \end{pmatrix} = \begin{pmatrix} -a+(-b) \\ c+d \end{pmatrix} = \begin{pmatrix} -a-b \\ c+d \end{pmatrix} \blacksquare$$

was $f(u+v)$ entspricht und zu zeigen war.
Entsprechend für die Multiplikation sei $u = \begin{pmatrix} a \\ c \end{pmatrix}, \lambda = k$

$$f(w) = \begin{pmatrix} -x \\ y \end{pmatrix}$$

Entsprechend

$$f(\lambda \cdot u) = f\left(k \cdot \begin{pmatrix} a \\ c \end{pmatrix}\right) = f\begin{pmatrix} k \cdot a \\ k \cdot c \end{pmatrix} = \begin{pmatrix} -(k \cdot a) \\ k \cdot c \end{pmatrix}$$
$$\lambda \cdot f(u) = k \cdot f(u) = k \cdot \begin{pmatrix} -a \\ c \end{pmatrix} = \begin{pmatrix} -(k \cdot a) \\ k \cdot c \end{pmatrix} \blacksquare$$

Abb. 3 veranschaulicht die Addition und Abbildung der Vektoren u und v mit $u = \begin{pmatrix} 2 \\ 0 \end{pmatrix}$, $v = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$.

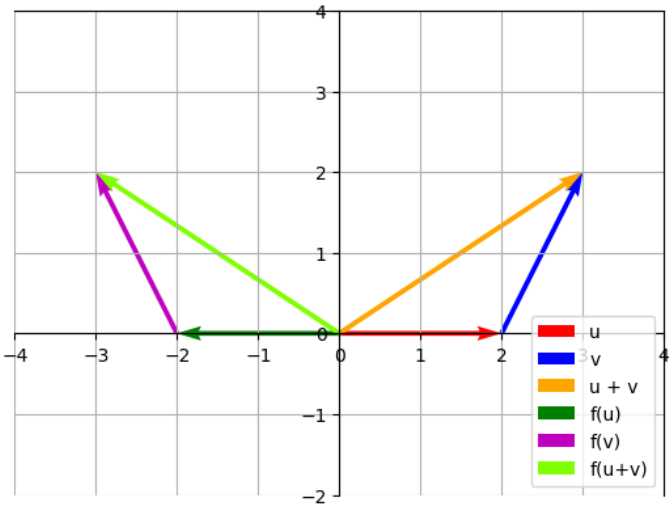

Abb. 3 Lineare Abbildung

Vereinfacht kann eine lineare Abbildung in Form einer Matrix dargestellt werden.

Sei v ein Vektor $\binom{x}{y}$ so ist dieser das Ergebnis der kanonischen Einheitsvektoren

$$e_1 = \binom{1}{0}, e_2 = \binom{0}{1}$$
$$\binom{x}{y} = x \cdot e_1 + y \cdot e_2 = \binom{1x + 0y}{0x + 1y}$$

Die Darstellung $\binom{1x + 0y}{0x + 1y}$ entspricht der Matrix $\begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}$.

Das bisher verwendete Beispiel $f(w) = \binom{-x}{y}$ in eine Matrix A überführt lautet demnach: $A = \begin{bmatrix}-1 & 0\\0 & 1\end{bmatrix}$, da e_1 auf $\binom{-1}{0}$ abgebildet wird, e_2 auf $\binom{0}{1}$ unverändert bleibt.
Es handelt sich bei diesem Beispiel also um eine Spiegelung an der y-Achse, wie in Abb. 3 unschwer zu erkennen ist.

2.2 Definition und Eigenschaften der affinen Abbildung

Bisher wurde die lineare Abbildung erläutert und veranschaulicht. Diese ist eine Spezialform der affinen Abbildungen.
Affine Abbildungen beinhalten gemäss [3] sämtliche linearen Abbildungen und zusätzlich

- Translation (Verschiebung)
- Drehungen um einen anderen Punkt als den Ursprung
- Spiegelungen an einer Achse, die den Ursprung nicht kreuzt
- Skalierungen, die relativ zu einem Punkt sind, der nicht der Ursprung ist

Gegeben sei das Beispiel einer Translation um $v = \binom{2}{2}$. Das heisst, jeder Punkt im Koordinatensystem wird um 2 Einheiten nach rechts (entlang der x-Achse) und 2 Einheiten nach oben (entlang der y-Achse) verschoben. Abb. 4 zeigt diese Verschiebung.

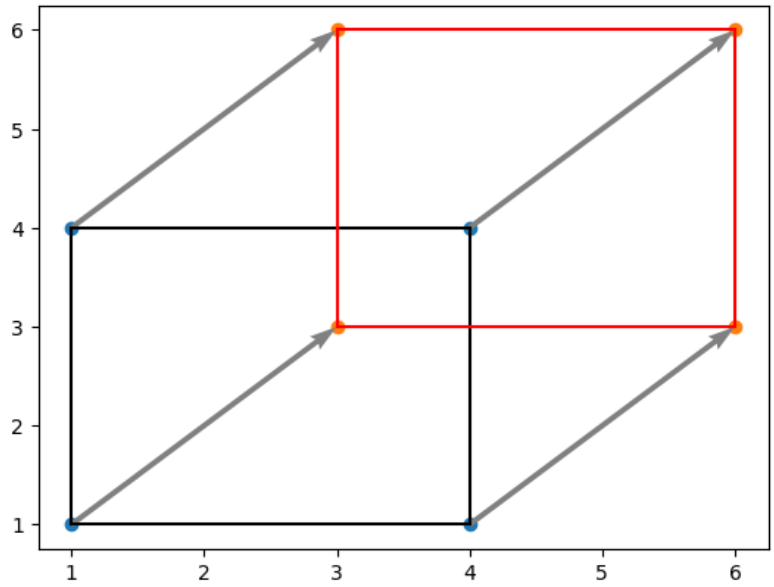

Abb. 4 Translation

Sei $w = \begin{pmatrix} e \\ f \end{pmatrix}$, so ergibt sich für die Translation um w $f(w) = f\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} e \\ f \end{pmatrix}$. In 2.1 erfolgte die Definition, dass $f\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} ax + by \\ cx + dy \end{pmatrix}$ sei.
Bei der affinen Abbildung wird diese Definition um konstante Glieder erweitert:

$$f\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} ax + by + e \\ cx + dy + f \end{pmatrix}$$

Wenn $e = f = 0$ gilt, dann handelt es sich um eine lineare Abbildung, womit die Aussage «lineare Abbildungen sind eine spezialform affiner Abbildungen» am Anfang dieses Kapitels belegt ist.

Wenn die Abbildung als Matrix von der Form $\begin{pmatrix} a & b & e \\ c & d & f \end{pmatrix}$ dargestellt wird, stellt sich die Schwierigkeit, dass eine Multiplikation von Matrix und Vektor nicht zum gewünschten Ergebnis $\begin{pmatrix} ax + bx + e \\ cy + dy + f \end{pmatrix}$ führen würde.
Die Lösung für das Problem ergibt sich durch einen «Sprung» von $\mathbb{R}^2$ nach $\mathbb{R}^3$. Sämtliche Vektoren werden um eine dritte, konstante Komponente 1 erweitert. Aus $\begin{pmatrix} x \\ y \end{pmatrix}$ wird $\begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$.
Man sagt, der Vektor wird in homogenen Koordinaten dargestellt.

Die Multiplikation lautet jetzt $\begin{pmatrix} a & b & e \\ c & d & f \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} ax + by + e \\ cx + dy + f \end{pmatrix}$.
Diesem resultierenden Vektor fehlt, wie unschwer zu erkennen ist, die dritte Komponente. Deshalb muss auch die Matrix um eine Konstante erweitert werden, also in homogenen Koordinaten dargestellt werden. Es gilt:

$$\begin{pmatrix} a & b & e \\ c & d & f \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} ax + by + e \\ cx + dy + f \\ 1 \end{pmatrix}$$

Mit der Darstellung in homogenen Koordinaten wird also aus $v = \begin{pmatrix} x \\ y \end{pmatrix}$ $v = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$ und die affine Abbildung $f\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} ax + by + e \\ cx + dy + f \end{pmatrix}$ wird durch die Matrix mit homogenen Koordinaten $A = \begin{bmatrix} a & b & e \\ c & d & f \\ 0 & 0 & 1 \end{bmatrix}$ dargestellt.

Die Translation um $\begin{pmatrix} e \\ f \end{pmatrix}$ wird folglich mit der Matrix $\begin{bmatrix} 1 & 0 & e \\ 0 & 1 & f \\ 0 & 0 & 1 \end{bmatrix}$ dargestellt.
Zur besseren Verständlichkeit sei dies anhand eines in Abb. 4 verwendeten Punktes gezeigt.
$P(1,1)$ wird um $\begin{pmatrix} 2 \\ 2 \end{pmatrix}$ nach $P'(3,3)$ verschoben.
Gemäss obenstehender Herleitung lautet die Abbildungsmatrix $\begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix}$ und der Vektor mit homogenen Koordinaten zu $P\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$. Die Abbildung entsprechend

$$\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 2 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 \cdot 1 + 0 \cdot 1 + 2 \cdot 1 \\ 0 \cdot 1 + 1 \cdot 1 + 2 \cdot 1 \\ 0 \cdot 1 + 0 \cdot 1 + 1 \cdot 1 \end{pmatrix} = \begin{pmatrix} 3 \\ 3 \\ 1 \end{pmatrix}.$$

Wie bereits besprochen entspricht $\begin{pmatrix} 3 \\ 3 \\ 1 \end{pmatrix}$ den homogenen Koordinaten des Vektors $\begin{pmatrix} 3 \\ 3 \end{pmatrix}$, was dem Ortsvektor von P' entspricht.

2.3 Zentralprojektion als affine Abbildung?

Gegeben sei ein Würfel, der sich komplett im ersten Oktanten von $\mathbb{R}^3$ befindet.
Die Seitenlänge beträgt 3 Einheiten und der Ursprung liegt bei $A(3,3,3)$. Dieser Würfel soll auf die xy-Ebene projiziert werden. Die Kamera befindet sich dabei bei $P_k = (2,8,12)$.
Abb. 5 zeigt die Situation.

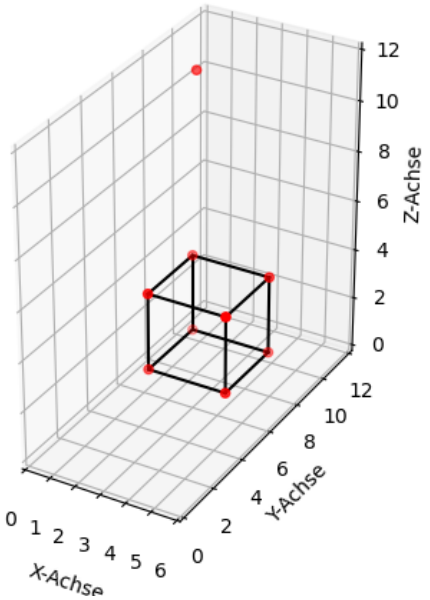

Abb. 5 Würfel im Raum mit Kamerapunkt

Jede Gerade die sowohl P_K, wie auch einen Punkt P des Würfels schneidet, durchsticht die xy-Ebene bei Punkt P'.
Für jeden Punkt auf einer solchen Gerade in $\mathbb{R}^3$ gilt, dass er durch $g: x = a + \lambda \cdot d$ dargestellt werden kann. Wobei x der darzustellende Punkt ist, a entspricht dem Stützvektor zu einem gegebenen Punkt auf der Geraden und d dem Richtungsvektor der Geraden, der durch die zwei gegebenen Punkte berechnet werden kann.
Konkret lautet die Gleichung $\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x_K \\ y_K \\ z_K \end{pmatrix} + \lambda \cdot \begin{pmatrix} x_P - x_K \\ y_P - y_K \\ z_P - z_K \end{pmatrix}$ wobei $_P$ für die Koordinaten des zu projizierenden Punktes und $_K$ für die Koordinaten der Kamera stehen. [1]
Da das Ziel eine Projektion auf die xy-Eben ist, gilt es nun die Gleichung mit $z = 0$ nach λ umzustellen.

$$z_K + \lambda \cdot (z_P - z_K) = 0 \Rightarrow \lambda = -\frac{z_K}{z_p - z_K} = \frac{1}{1 - z_P/z_K}$$

Die Koordinaten des Bildpunktes lassen sich dadurch berechnen und es resultiert:

$$\begin{pmatrix} x' \\ y' \\ 0 \end{pmatrix} = \begin{pmatrix} x_K + \frac{1}{1 - \frac{z_P}{z_K}} \cdot (x_P - x_K) \\ y_K + \frac{1}{1 - \frac{z_P}{z_K}} \cdot (x_P - x_K) \\ 0 \end{pmatrix} = \begin{pmatrix} \frac{x_P - x_K \cdot \frac{z_P}{z_K}}{1 - \frac{z_P}{z_K}} \\ \frac{y_P - y_K \cdot \frac{z_P}{z_K}}{1 - \frac{z_P}{z_K}} \\ 0 \end{pmatrix}$$

Es sei festgestellt, dass die zu projizierenden Koordinaten als Argument in die projizierten Koordinaten einfliesst. Demnach ist dies keinesfalls eine lineare Abbildung.

2.4 Herleitung der Transformationsmatrix der Zentralprojektion

Wie in Kap. 2.2 gesehen, kann eine nicht lineare Abbildung in $\mathbb{R}^2$ als affine Abbildung in $\mathbb{R}^3$ dargestellt werden.
Für die Zentralprojektion eines Objekts in $\mathbb{R}^3$ bedient man sich derselben Idee. Es erfolgt ein Sprung von $\mathbb{R}^3$ in $\mathbb{R}^4$, womit die Translation der Punkte aus $z > 0$ zu $z = 0$ einer Scherung im vierdimensionalen Raum entspricht.
In Kap. 2.2 wurden die homogenen Koordinaten durch das hinzufügen einer weiteren Koordinate mit Wert 1 definiert $\begin{pmatrix} x \\ y \end{pmatrix} \rightarrow \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$. Die x- und y-Koordinaten eines Punktes bleiben also auf der Ebene $z = 1$ unverändert. Es könnte aber ebenso $z = 2$ gewählt werden, womit die homogenen Koordinaten durch $\begin{pmatrix} x \\ y \end{pmatrix} \rightarrow \begin{pmatrix} \frac{x}{2} \\ \frac{y}{2} \\ 2 \end{pmatrix}$ gegeben wären. Jeder Punkt auf der Geraden durch den Nullpunkt und $\begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$ bildet somit homogene Koordinaten des Punktes $\begin{pmatrix} x \\ y \end{pmatrix}$ in Form von $\begin{pmatrix} x \\ y \end{pmatrix} \rightarrow \begin{pmatrix} \frac{x}{z} \\ \frac{y}{z} \\ z \end{pmatrix}$.
Abb. 6 zeigt diesen Sachverhalt.

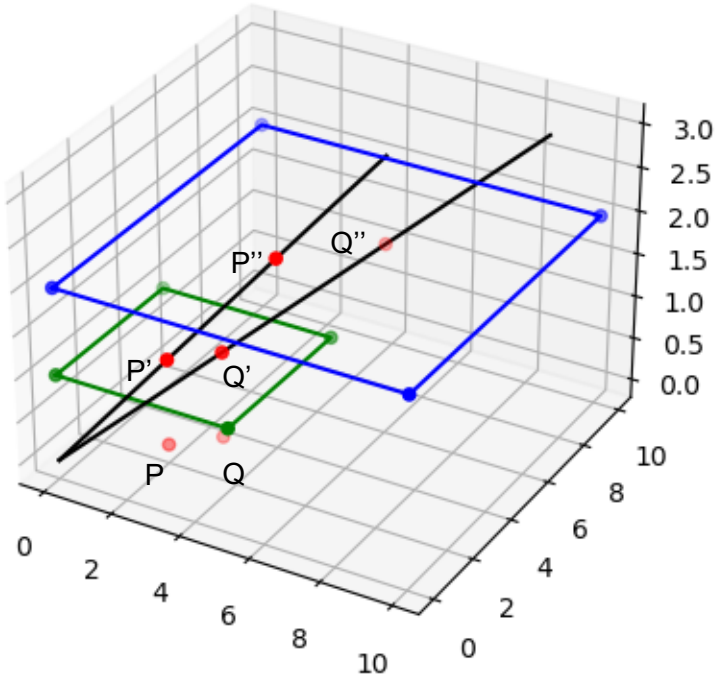

Abb. 6 Abbildung in z=1 und z=2

Diese Idee soll jetzt auch für die Problematik der Nichtlinearität aus Kap. 2.3 angewandt werden. Der Term $1 - \frac{z_P}{z_K}$ wird in die vierte Koordinate w «ausgelagert». Somit ergibt sich für H als Bezeichner der homogenen Koordinaten

$$\begin{pmatrix} x_H \\ y_H \\ z_H \\ w \end{pmatrix} = \begin{pmatrix} x_P - x_K \cdot \frac{z_P}{z_K} \\ y_P - y_K \cdot \frac{z_P}{z_K} \\ 0 \\ 1 - \frac{z_P}{z_K} \end{pmatrix}$$

Folglich wird jeder Punkt auf einer Ebene $w \neq 1$ abgebildet, abhängig von seiner eigenen und der Kamera z-Koordinate.
Es folgt eine Umformung, um in der w-Koordinate eine 1 zu erhalten.

$\begin{pmatrix} \frac{x_K \cdot z_P - x_P \cdot z_K}{z_P - z_K} \\ \frac{y_K \cdot z_P - y_P \cdot z_K}{z_P - z_K} \\ 0 \\ 1 \end{pmatrix} = A \cdot \begin{pmatrix} x \\ y \\ 0 \\ 1 \end{pmatrix}$ wobei A für die gesuchte Abbildungsmatrix steht.

Für die Linearität gilt, dass der Nullpunkt auf den Nullpunkt abgebildet wird. Es bietet sich an, die Matrix mithilfe der kanonischen Einheitsvektoren $e_0 = (0,0,0,1), e_1 = (1,0,0,1), e_2 = (0,1,0,1),\ e3 = (0,0,1,1)$ zu berechnen.

Durch Auflösen des daraus resultierenden Gleichungssystems ergibt sich die allgemeine Matrix:

$$A = \begin{bmatrix} 1 & 0 & \frac{x_K}{z_K} & 0 \\ 0 & 1 & \frac{y_K}{z_K} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{1}{z_K} & 1 \end{bmatrix}$$

Mit dieser allgemeinen Matrix lassen sich jetzt alle Punkte auf die xy-Ebene Projizieren. Im folgenden Kapitel sei dies anhand eines Parallelepipeds mit Python demonstriert.

3 Umsetzung in Python

In der Aufgabenstellung dieser Arbeit wird die Zentralprojektion eines Parallelepipeds (Spat), welches sich komplett im ersten Oktanten (alle x,y,z > 0) des Koordinatensystems befindet, auf die xy-Ebene gefordert.
Ein Spat ist ein dreidimensionaler Körper, dessen Seiten durch Parallelogramme definiert sind. Zudem sind jeweils gegenüberliegende Seiten deckungsgleich, also durch die gleichen Kantenlängen definiert.
Abb. 7 zeigt ein Parallelepiped.

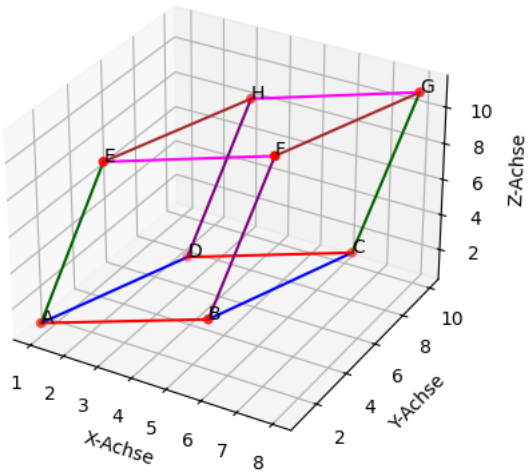

Abb. 7 Parallelepiped

Der gesamte Code befindet sich im Anhang.
Um mit dem Code zu arbeiten, empfiehlt es sich, die Klasse *Spat* in einem eigenen Projekt zu importieren. Dies geschieht mit dem Befehl *from spat import Spat*.

Idealerweise definiert man sich ein Spat-Objekt, mit welchem gearbeitet werden kann. *spatInstance = Spat(x,y,z)*. Der Konstruktor des Objekts erwartet die x-, y-, und z-Koordinate des Ursprungs des Spats. Diese können frei gewählt werden, müssen allerdings im ersten Oktanten des Koordinatensystems liegen. Ist dies nicht gegeben, wirft die Applikation einen Fehler.

Um sich den Spat anzeigen zu lassen, genügt es auf dem Objekt *drawSpat()* anzuwenden. Abb. 7 ist so entstanden.

Weiter kann mit *setCamera(x,y,z)* auf dem Objekt die Kameraposition im Koordinatensystem gesetzt werden, von welcher zukünftig die Zentralprojektion ausgehen soll. Bedingung für die Kamerakoordinaten ist, dass der Spat komplett zwischen der Kamera und der xy-Ebene liegt. Dies ist jederzeit gegeben, wenn die z-Koordinate der Kamer grösser ist als die grösste z-Koordinate eines Spat-Eckpunktes. Sollte die Bedingung nicht erfüllt sein, wirft die Applikation einen entsprechenden Fehler.

Ist die Kamera gesetzt, kann mit der *project()*-Methode die Projektion auf die xy-Ebene durchgeführt werden. Das Resultat wird geplottet.

Als erstes Beispiel sei der Ursprung des Parallelepipeds bei $(3,5,6)$ und die Kamera bei $(5,18,17)$.
Abb. 8 zeigt das Resultat der Projektion.

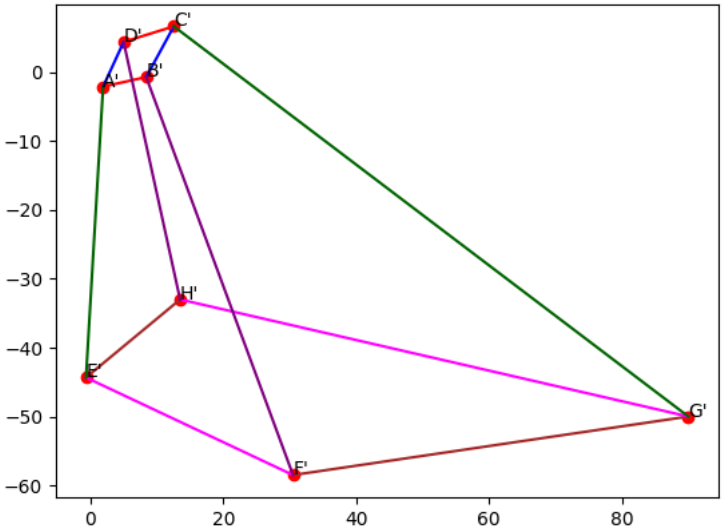

Abb. 8 Projektion auf die xy-Ebene (Kamera bei 10,2.5,15)

Als zweites Beispiel liege der Ursprung des Objekts bei $(1,1,1)$ und die Kamera bei $(15,15,15)$, wie es im folgenden Code definiert ist.

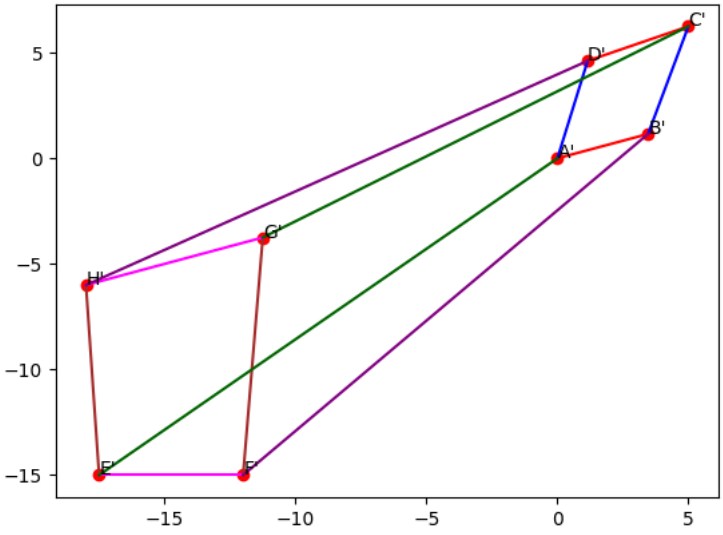

Abb. 9 Projektion auf die xy-Ebene (Kamera bei 15,15,15)

Es folgen einige Auszüge aus dem Code, um das Vorgehen zu demonstrieren.
Oben erwähnte Vorgehensweise:

```
from spat import Spat

spatInstance = Spat(1,1,1)

spatInstance.drawSpat()

spatInstance.setCamera(15,15,15)

spatInstance.project()
```

Im Konstruktor werden die Vektoren gesetzt, die den Spat definieren

```
self.u = np.array([4, 2, 1])
self.v = np.array([2, 5, 1])
self.w = np.array([1, 2, 8])
```

Die Methode *checkCoordinates(x, xMin, y, yMin, z, zMin)*, die die Werte gegen ihre Minimalwerte prüft

```
def checkCoordinates(x, xMin, y, yMin, z, zMin):
    """
    Checks if the coordinates are within the bounds
    :param x:
    :param xMin:
    :param y:
    :param yMin:
    :param z:
    :param zMin:
    :return:
    """
    if x < xMin or y < yMin or z < zMin:
        raise Exception("Coordinates out of bounds. xMin: {}, yMin: {},
zMin: {}".format(xMin, yMin, zMin))
```

Die Berechnung der Eckpunkte aufgrund des gegebenen Ursprungs und den Richtungsvektoren

```
# bottom
A = np.array(self.origin)
B = self.origin + self.u
C = self.origin + self.u + self.v
D = self.origin + self.v

# top
E = self.origin + self.w
F = self.origin + self.w + self.u
G = self.origin + self.w + self.u + self.v
H = self.origin + self.w + self.v
```

Der gesamte Code befindet sich im Anhang. Er ist aber auch auf https://github.com/tobias-hu/ffhs_linalg verfügbar. Ebenfalls Teil des Repositories sind sämtliche Plots und das Jupyter-Notebook, in welchem diese erstellt wurden.

4 Diskussion

Die Zentralprojektion stellt eine allgemein wichtige Disziplin dar. Ausser mit 3D-Druckern, Hologrammen oder «realen» Modellen, werden Abbildungen, Grafiken, Modelle und dergleichen immer zweidimensional dargestellt, obschon es sich in den meisten Fällen um dreidimensionale Objekte handelt.
Mit der Zentralprojektion ist es möglich dies zu bewerkstelligen.
Die Idee eine zweidimensionale Abbildung, die nicht linear ist, in den dreidimensionalen Raum zu transferieren, um sie dort als lineare Abbildung darzustellen, ist greifbar und vorstellbar. Mit der Anwendung auf den dreidimensionalen Raum und dem Transfer in den vierdimensionalen Raum ist man doch in einem hohen Abstraktionslevel angelangt.

Die aus dieser Arbeit resultierenden Projektionen sind mathematisch korrekt.
In einer ersten Version des Codes waren sämtliche Linien des Spats schwarz eingefärbt und die Punkte nicht benannt. Durch eine Erweiterung, die die Linien einfärbt und die Punkte benennt, ist die Verzerrung der Projektion um einiges besser sicht- und realisierbar.
Es dürfte spannend sein, den Code insofern zu erweitern, dass sowohl die Projektion wie auch das ursprüngliche Parallelepiped in demselben Plot zu sehen wären. Dies könnte Aufschluss über die Verzerrung der Abbildung bieten. Es würde dabei aber zu beachten geben, dass ja bereits die Darstellung des dreidimensionalen Objekts eine Projektion auf eine Ebene, nämlich den Bildschirm, ist.

5 Anhang

Kompletter Code der Klasse Spat.py

```
import random

import matplotlib.pyplot as plt
import numpy as np


def checkCoordinates(x, xMin, y, yMin, z, zMin):
    """
    Checks if the coordinates are within the bounds
    :param x:
    :param xMin:
    :param y:
    :param yMin:
    :param z:
    :param zMin:
    :return:
    """
    if x < xMin or y < yMin or z < zMin:
        raise Exception("Coordinates out of bounds. xMin: {}, yMin: {},
zMin: {}".format(xMin, yMin, zMin))


def plotLevels(points, plot, twoDimensional=False):
    """
    Plots the sidelines of the top and bottom level of a spat or his pro-
jection
    :param points:
    :param plot:
    :param twoDimensional:
    :return:
    """

    colors = ['red', 'blue', 'red', 'blue', 'magenta', 'brown',
    'magenta', 'brown']

    for i in range(0, 8):
        j = i + 1
        if j % 4 == 0:
            j = j - 4

        x, y = ([points[i][0], points[j][0]],
                [points[i][1], points[j][1]])

        if not twoDimensional:
            z = [points[i][2], points[j][2]]
            plot.plot(x, y, z, c= colors[i])
        else:
            plot.plot(x, y, c= colors[i])
```

```
def plotSidelines(points, plot, twoDimensional=False):
    """
    Plots the sidelines between top and bottom levels of a spat or his
projection
    :param points:
    :param plot:
    :param twoDimensional:
    :return:
    """
    colors = ['darkgreen', 'purple','darkgreen', 'purple']
    for i in range(0, 4):
        x, y = ([points[i][0], points[i + 4][0]],
                [points[i][1], points[i + 4][1]])

        if not twoDimensional:
            z = [points[i][2], points[i + 4][2]]
            plot.plot(x, y, z, c= colors[i])
        else:
            plot.plot(x, y, c= colors[i])


def getCoordinatesByAxis(points):
    """
    Gets the coordinates of the given points and reorders them into x,y
and z coordinate-lits
    :param points:
    :return:
    """
    xCoordinates = [points[i][0] for i in range(len(points))]
    yCoordinates = [points[i][1] for i in range(len(points))]
    zCoordinates = [points[i][2] for i in range(len(points))]

    return [xCoordinates, yCoordinates, zCoordinates]


def _drawProjection(points):
    """
    Plots the projection on the xy-level
    :param points:
    :return:
    """
    x = [points[i][0] for i in range(len(points))]
    y = [points[i][1] for i in range(len(points))]

    # points
    plt.scatter(x, y, c='r')

   # label points
   pointNames = ["A'", "B'", "C'", "D'", "E'", "F'", "G'", "H'"]
   for x,y,i in zip(x,y,range(len(points))):
     plt.text(x, y, pointNames[i])


    # top and bottom
    plotLevels(points, plt, True)

    # sidelines
    plotSidelines(points, plt, True)

    plt.show()
```

```
class Spat:
    """
    The spat class
    """
    def __init__(self, x, y, z):
        """
        Initializes the spat
        Takes the x,y and z coordinates for the origin, which is in fact
the first corner of the spat
        raises an error if the coordinates are outside the first octant
of the coordinate system
        :param x:
        :param y:
        :param z:
        """
        self.u = np.array([4, 2, 1])
        self.v = np.array([2, 5, 1])
        self.w = np.array([1, 2, 8])

        try:
            checkCoordinates(x, 1, y, 1, z, 1)
        except ValueError as err:
            raise err

        self.origin = np.array([x, y, z])

        self.points = self.__calcPoints()

        self.cameraPoints = []

    def __calcPoints(self):
        """
        Calculates the corners of the spat with the given origin and the
defined vectors (@see __init__)
        :return:
        """

        # bottom
        A = np.array(self.origin)
        B = self.origin + self.u
        C = self.origin + self.u + self.v
        D = self.origin + self.v

        # top
        E = self.origin + self.w
        F = self.origin + self.w + self.u
        G = self.origin + self.w + self.u + self.v
        H = self.origin + self.w + self.v

        return [A.tolist(), B.tolist(), C.tolist(), D.tolist(),
        E.tolist(), F.tolist(), G.tolist(), H.tolist()]

    def getPoints(self):
        """
        Just a normal getter method
        :return:
        """
        return self.points
```

```
    def drawSpat(self):
        """
        Draws the spat
        Which is not actually part of the paper, but it helped me to see
        what's going on...
        :return:
        """
        fig = plt.figure()
        ax = fig.add_subplot(111, projection='3d')

        # get x,y,z coordinates of all the points
        coordinatesByAxis = getCoordinatesByAxis(self.points)
        x = coordinatesByAxis[0]
        y = coordinatesByAxis[1]
        z = coordinatesByAxis[2]

        ax.set_xlabel('X-Achse')
        ax.set_ylabel('Y-Achse')
        ax.set_zlabel('Z-Achse')

        ax.scatter(x, y, z, c='r', marker='o')

        # Label points
        pointNames = ["A","B","C","D","E","F","G","H"]
        for x,y,z,i in zip(x,y,z,range(len(x))):
            ax.text(x,y,z,pointNames[i])


        # TOP AND BOTTOM
        plotLevels(self.points, ax)

        # SIDELINES
        plotSidelines(self.points, ax)

        plt.show()

    def setCamera(self, x, y, z):
        """
        Takes the camera position and checks if it is allowed to be there
        If everything is ok, the camera coordinates are set
        :param x:
        :param y:
        :param z:
        :return:
        """
        spatCoordinates = getCoordinatesByAxis(self.points)
        spatzCoordinates = spatCoordinates[2]

        zMax = max(spatzCoordinates)

        checkCoordinates(x, 1, y, 1, z, zMax + 1)

        self.cameraPoints = [x, y, z]
```

```
    def _projectPoint(self, x, y, z):
        """
        Calculates the projection of the given point to the xy-level
        :param x:
        :param y:
        :param z:
        :return:
        """
        cameraX = self.cameraPoints[0]
        cameraY = self.cameraPoints[1]
        cameraZ = self.cameraPoints[2]

        projectionX = (cameraX * z - x * cameraZ) / (z - cameraZ)
        projectionY = (cameraY * z - y * cameraZ) / (z - cameraZ)
        projectionZ = 0

        return [projectionX, projectionY, projectionZ]

    def project(self):
        """
        Checks if the camera coordinates are set
        Arranges the spat points to x,y,z coordinate lists
        Passes the points over to __projectPoint and then passes the pro
        jected points
        to _drawProjection
        :return:
        """
        if not self.cameraPoints:
            raise Exception("Projection not possible. Please add a camera
            point first.")

        # get all the points of the spat and project them
        spatCoordinates = getCoordinatesByAxis(self.points)
        projectionX = [spatCoordinates[0][i] for i in range(len(spatCoor-
dinates[0]))]
        projectionY = [spatCoordinates[1][i] for i in range(len(spatCoor-
dinates[1]))]
        projectionZ = [spatCoordinates[2][i] for i in range(len(spatCoor-
dinates[2]))]

        projectedPoints = [self._projectPoint(projectionX[i], projec-
tionY[i], projectionZ[i]) for i in
                           range(len(projectionX))]

        _drawProjection(projectedPoints)
```

Die Abbildungen 3-7 sind mit eigenem Python-Code generiert:

Abb. 3:

```
import matplotlib.pyplot as plt
import numpy as np
#%%
u = np.array([2,0])
v = np.array([1,2])
#%%
ax = plt.subplots()
ax.spines['left'].set_position('zero')
ax.spines['bottom'].set_position('zero')
ax.quiver(0,0,u[0],u[1],angles='xy', scale_units='xy', scale=1, co-
lor='r', label='u')
ax.quiver(u[0],u[1],v[0],v[1],angles='xy', scale_units='xy', scale=1, co-
lor='b', label='v')
ax.quiver(0,0,3,2, angles='xy', scale_units='xy', scale=1, color='oran-
ge', label='u + v')
ax.quiver(0,0,-u[0], u[1], angles='xy', scale_units='xy', scale=1, co-
lor='green', label='f(u)')
ax.quiver(-u[0], u[1], -v[0], v[1], angles='xy', scale_units='xy',
scale=1, color='m', label='f(v)')
ax.quiver(0,0,-3,2, angles='xy', scale_units='xy', scale=1, color='chart-
reuse', label='f(u+v)')
ax.grid()
ax.set_xlim([-4, 4])
ax.set_ylim([-2, 4])
ax.legend(loc='lower right')
```

Abb. 4:

```
ax = plt.subplots()
x, y = [1,4,1,4], [1,1,4,4]
ax.scatter(x,y)
plt.plot([1,1],[1,4], 'black')
plt.plot([1,4],[1,1], 'black')
plt.plot([1,4],[4,4], 'black')
plt.plot([4,4],[1,4], 'black')

x2, y2 = [3,6,3,6],[3,3,6,6]
ax.scatter(x2,y2)
plt.plot([3,3],[3,6], 'red')
plt.plot([3,6],[3,3], 'red')
plt.plot([3,6],[6,6], 'red')
plt.plot([6,6],[3,6], 'red')

ax.quiver(1,1,2,2, angles='xy', scale_units='xy', scale=1, color='grey')
ax.quiver(1,4,2,2, angles='xy', scale_units='xy', scale=1, color='grey')
ax.quiver(4,1,2,2, angles='xy', scale_units='xy', scale=1, color='grey')
ax.quiver(4,4,2,2, angles='xy', scale_units='xy', scale=1, color='grey')
```

Abb. 5:

```
A = [3,3,3]
B = [6,3,3]
C = [6,3,6]
D = [3,3,6]
E = [3,6,3]
F = [6,6,3]
G = [6,6,6]
H = [3,6,6]
K = [2,8,12]

points = [A, B, C, D, E, F, G, H,K]


ax = plt.subplot(111, projection='3d')

x = [points[i][0] for i in range(0, len(points))]
y = [points[i][1] for i in range(0, len(points))]
z = [points[i][2] for i in range(0, len(points))]

ax.set_xlabel('X-Achse')
ax.set_ylabel('Y-Achse')
ax.set_zlabel('Z-Achse')

ax.set_xlim([0,6])
ax.set_ylim([0,12])
ax.set_zlim([0,12])
ax.set_box_aspect([1,2,2])

ax.scatter(x, y, z, c='r', marker='o')

# TOP AND BOTTOM
for i in range(0, 8):
    j = i + 1
    if j % 4 == 0:
        j = j-4

    x, y, z = ([points[i][0], points[j][0]],
               [points[i][1], points[j][1]],
               [points[i][2], points[j][2]])

    ax.plot(x, y, z, c='black')

# SIDELINES
for i in range(0, 4):
    x,y,z = ([points[i][0], points[i+4][0]],
             [points[i][1], points[i+4][1]],
             [points[i][2], points[i+4][2]])
    ax.plot(x, y, z, c='black')


plt.show()
```

Abb. 6:

```
P = [2,2,0]
Q = [3,3,0]

P1 = [2,2,1]
Q1 = [3,3,1]

ax = plt.subplot(111,projection='3d')

# Ebene 1
A = [0,0,1]
B = [5,0,1]
C = [5,5,1]
D = [0,5,1]

#ebene 1
edges = [A,B,C,D]

x = [edges[i][0] for i in range(0, len(edges))]
y = [edges[i][1] for i in range(0, len(edges))]
z = [edges[i][2] for i in range(0, len(edges))]
ax.scatter(x,y,z, c='green')

# sidelines
x = [0,5,5,0,0]
y = [0,0,5,5,0]
z = [1,1,1,1,1]

ax.plot(x,y,z, c='green')


# points
points = [P,P1,Q,Q1]

x = [points[i][0] for i in range(0,len(points))]
y = [points[i][1] for i in range(0,len(points))]
z = [points[i][2] for i in range(0,len(points))]

ax.scatter(x,y,z, c='red')


# lines
ax.plot([0,6],[0,6],[0,3],c='black')
ax.plot([0,9],[0,9],[0,3],c='black')

# ebene 2
A = [0,0,2]
B = [10,0,2]
C = [10,10,2]
D = [0,10,2]

edges = [A,B,C,D]

x = [edges[i][0] for i in range(0, len(edges))]
y = [edges[i][1] for i in range(0, len(edges))]
z = [edges[i][2] for i in range(0, len(edges))]
ax.scatter(x,y,z, c='blue')

# sidelines
x = [0,10,10,0,0]
y = [0,0,10,10,0]
z = [2,2,2,2,2]
```

```
ax.plot(x,y,z,c='blue')

P2 = [4,4,2]
Q2 = [6,6,2]

ax.scatter([4,6],[4,6],[2,2],c='red')
```

Abbildung 7 entspricht dem Resultat aus main.py.

Literatur- und Quellenverzeichnis

[1] M. Geuss, «youtube.com,» [Online]. Available: https://www.youtube.com/watch?v=2EsqQJ9irZQ. [Zugriff am 16 10 2024].

[2] H.-J. e. a. Bungartz, Einführung in die Computergrafik, Springer, 2013.

[3] R. Socher, Mathematik für Informatiker: Mit Anwendungen in der Computergrafik und Codierungstheorie, Fachbuchverlag Leipzig im Carl Hanser Verlag, 2011.

Bildverzeichnis