Inhaltsverzeichnis

1 Inhaltliche Grundlagen **1**
1.1 Digitales Bild . . . 1
1.1.1 Definition und Abspeicherung . . . 1
1.1.2 Spektrale Bilddarstellung . . . 2
1.2 Grafikpipeline . . . 5
1.2.1 Übersicht . . . 6
1.2.2 Vertex Operationen – Transformationen . . . 8
1.2.3 Vertex Operationen – Projektion . . . 10
1.2.4 Primitiven Operationen . . . 13
1.2.5 Fragment Operationen . . . 15

2 Anhang **17**
2.1 Digitales Bild – Beispiel-Abbildungen . . . 17
2.2 Vertex-Operationen – Transformationsmatrizen . . . 21
2.3 Vertex-Operationen – Kameramatrix . . . 27
2.4 Vertex-Operationen – Orthographische Projektion . . . 29
2.5 Vertex Operationen – Herleitung Projektionsmatrix **P** für die perspektivische Projektion . . . 30
2.6 Vertex-Transformationen – Clipping . . . 33

Literaturverzeichnis **34**

1 Inhaltliche Grundlagen

1.1 Digitales Bild

1.1.1 Definition und Abspeicherung

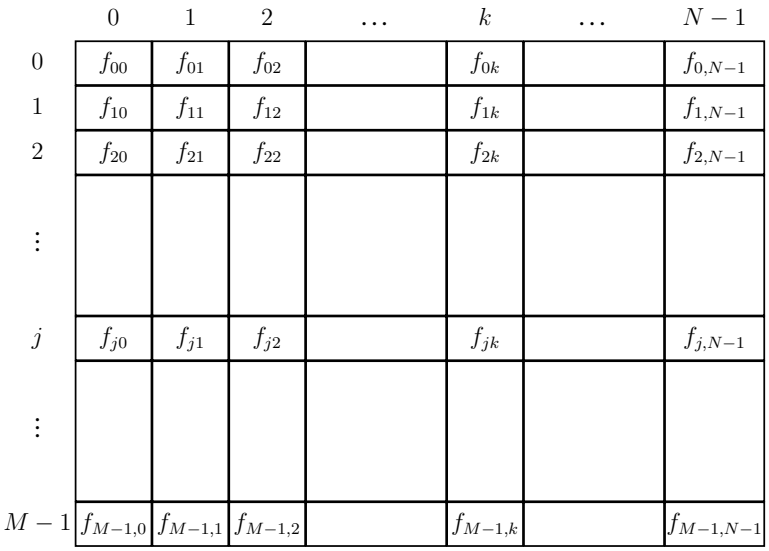
Während des VR-Navigationsexperiments wird dem Teilnehmer eine schnelle Abfolge an *Frames* gezeigt - jeder dieser Frames ist ein einzelnes *Bild*. Dieses Bild zeigt einen Ausschnitt der virtuellen Welt, welcher durch die virtuelle Kamera des Teilnehmers wahrgenommen wird. Wird die virtuelle Kamera bewegt, so ändert sich der angezeigte Frame. Dieser zeigt nun ein leicht anders Bild als der vorherige Frame, da sich das Blickfeld der virtuellen Kamera durch die Bewegung leicht verändert hat. Wenn sich die Frames schnell genug ändern während einer Bewegung, so nimmt der Teilnehmende dies als Bewegung simulierenden, flüssigen Datenstrom wahr, und nicht als einzelne Bilder.

Genau genommen handelt es sich bei jedem Frame nicht nur um ein Bild, sondern um ein *digitales Bild*. Dieses kann mit einer diskreten, endlichen Menge an Zahlen repräsentieren werden, dazu wird es in Pixel aufgeteilt [1, 2, 3]. Den Pixeln selbst können ein oder mehrere Werte zugeordnete werde, welche die Helligkeit und Farbe der korrespondierenden Bildschirmeinheit bestimmen [1, 2, 4, 5].
Spezifischer repräsentieren diese Werte die Intensität einer gemessenen *Strahlung*. Bei dieser Strahlung kann es sich beispielsweise um das sichtbare Licht handeln, welches mittels Strahlenreflexion bei der Fotografie “aufgefangen” und sichtbar gemacht wird. Es könnte sich aber auch um Röntgenstrahlen handeln [2]. Was für eine Strahlung es genau ist und wie diese sichtbar gemacht wird, ist jedoch Nebensache. Der Fokus liegt darauf wie die digitalen Bilder schlussendlich repräsentiert werden und wie mittels dieser Repräsentation Informationen extrahiert und wiedergegeben werden können.

Digitale Bilder lassen sich grob in *Intensitätsbilder* und *Farbbilder* unterscheiden [1, 2].
Ein Intensitätsbild enthält lediglich Informationen über die allgemeine Intensität der gemessenen Strahlung. Handelt es sich zum Beispiel um Strahlung des sichtbaren Lichts, so geben die Werte an wie viel Licht eingefallen ist [1, 2].
Ein Farbbild enthält hingegen Informationen über die Intensität unterschiedlicher Wellenlängen [1, 2].
Bei einem *Intensitätsbild* wird jedem Pixel ein Wert zwischen 0 und 255 zugeordnet; je höher der Wert, desto höher die Intensität und desto heller der Pixel [2, 3]. Abgespeichert werden diese Werte in Form einer Matrix. Die Dimensionen dieser Matrix sind dabei durch die Anzahl der Pixel in die Höhe M und die Anzahl der Pixel in die Breite N gegeben [1, 2, 3, 4]. Um den Intensitätswert f_{jk} eines Pixel an der Stelle mit Höhe j und Breite k zu bekommen, muss folglich der Eintrag (j, k) in der zugehörigen Matrix $\mathbf{A}$ ausgelesen werden: $\mathbf{A}_{jk} = f_{jk}$ [2].
In Abbildung 1 auf der nächsten Seite ist der allgemeine Aufbau solch einer Bild-Matrix $\mathbf{A}$ zu sehen. Ein Beispiel-Intensitätsbild und die zugehörige Matrix sind in Abbildung 9 auf Seite 17 (Anhang) zu finden.

Bei einem *Farbbild* werden jedem Pixel mehrere Werte zugeordnet. Die genau Anzahl dieser Werte ist durch das gewählte *Farbmodell* festgelegt [1]. Wobei ein Farbmodell nichts anders ist als ein strukturiertes System, das mithilfe von einer kleinen Anzahl an Primärfarben alle anderen Farben darstellen kann [6, 7].
Das wohl gängigste Farbmodell ist das *RGB-Farbmodell*. Die bei diesem Farbmodell verwendeten Primärfarben sind **R**ot, **G**rün und **B**lau [1, 2, 3, 7, 8].

Abbildung 1: Allgemeiner Aufbau einer Bild-Matrix **A**; Bildquelle: Darstellung nach [2].



Bei einem Farbbild wird folglich jedem Pixel ein Rot-Wert, ein Grün-Wert und ein Blau-Wert zugewiesen. Diese Werte geben an, mit was für einer Intensität die jeweilige zur Primärfarbe gehörige Wellenlänge in die "finalen" Farbe eingeht; also wie groß der Anteil der Primärfarbe in der "finalen" Farbe ist. Es gilt: Je höher der Wert, desto höher die Intensität, desto mehr ist von der Primärfarbe in der "finalen" Farbe vorhanden [2, 3]. Abbildung 11 auf Seite 18 (Anhang) zeigt welche Farben ein Pixel auf diese Art annehmen kann.

Abgespeichert werden diese Werte ebenfalls in einer Matrix, beziehungsweise in mehreren Matrizen. Die Anzahl der Matrizen hängt von der Anzahl der Primärfarben P des Farbmodells ab; es gibt für jede Primärfarbe genau eine Matrix. Diese Matrix speichert folglich für jeden Pixel den Intensitätswert der zugehörigen Primärfarbe ab; isoliert betrachtet ist sie somit nichts anders als ein Intensitätsbild [1, 2, 3]. Anstelle von Primärfarben wird in diesem Zusammenhang häufig von *Farbkanälen* gesprochen. Ein RGB-Farbbild besitzt somit drei Farbkanäle: einen roten Farbkanal, einen grünen Farbkanal und einen blauen Farbkanal.
Die Dimensionen dieser Farbkanal-Matrizen sind dabei durch die Anzahl der Pixel in die Höhe M und die Anzahl der Pixel in die Breite N gegeben [1, 2, 3]. Allgemein wird ein Farbbild folglich mittels P Matrizen mit den Dimensionen $M \times N$ dargestellt [2, 3].
Um die Werte $f_{i_{jk}}$ eines Pixel an der Stelle mit Höhe j und der Breite k zu bekommen, müssen folglich nur die Einträge (j, k) in den zugehörigen Matrizen $\mathbf{A}_i$ ausgelesen werden $\mathbf{A}_{i_{jk}} = f_{i_{jk}}$ [2].

In Abbildung 1 ist der allgemeine Aufbau solch einer Farbkanal-Matrix $\mathbf{A}_i$ zu sehen. Es ist zu beachten, dass dieser identisch zu dem Aufbau einer zu einem Intensitätsbild gehörenden Matrix ist. Ein Beispiel-RGB-Farbbild und die zugehörigen Matrizen sind in Abbildung 10 auf Seite 17 (Anhang) zu finden.

1.1.2 Spektrale Bilddarstellung

In Unterabschnitt 1.1.1 auf der vorherigen Seite wurde die Darstellung und Abspeicherung eines digitalen Bilds im sogenannten *Ortsraum* eingeführt [2, 3, 9]. Hierbei ist das

Bild über Intensitätswerte an bestimmten Orten definiert; gesammelt werden diese in einer Matrix [2, 10, 9].
Betrachtet man jeden Eintrag der Matrix als einen Punkt im Raum, so kann man den Ortsraum als dreidimensionalen euklidischen Vektorraum veranschaulichen. Der Ursprung dieses Vektorraums liegt in der linken obereren Ecke des digitalen Bildes (beziehungsweise bei Farbbildern in der linken oberen Ecke genau eines Farbkanals, beispielsweise des Rot-Kanals bei RGB-Bildern); seine Basis besteht etwa aus den Einheitsvektoren. Folglich lässt sich jeder Intensitätswert durch genau einen Vektor, welcher eine Linearkombination der Basisvektoren ist, in diesem Raum beschreiben [11, 12].

Diese Interpretation eines Bildes kann nun in eine vollkommen andere, aber mathematisch äquivalente, umgeformt werden. Das geschieht durch einen Basiswechsel; das Bild befindet sich nun im *Frequenzraum* [2, 3, 9, 11, 12].
Realisiert ist dieser Wechsel vom Ortsraum zum Frequenzraum durch die *diskrete Fouriertransformation (DFT)*, der Wechsel vom Frequenzraum zurück zum Ortsraum erfolgt durch die *Inverse der diskrete Fouriertransformation (IDFT)*. Während des Wechsels gehen keinerlei Informationen verloren: Die Repräsentation des Bilds im Ortsraum verändert sich nur, wenn an der Repräsentation des Bilds im Frequenzraum etwas geändert wurde, und umgekehrt [2, 9].

Die allgemeine Idee hinter der DFT ist, dass sich jede (diskrete) periodische Funktion als Überlagerung (Summation) von Sinus-Funktionen und Cosinus-Funktionen unterschiedlicher Frequenzen darstellen lässt. Ist eine Funktion nicht-periodisch, so lässt sie sich durch Aneinanderreihung von sich selbst periodisieren [2, 9, 11]. Ein Beispiel anhand der Box-Funktion ist in Abbildung 12 auf Seite 18 (Anhang) zu finden.
Bei der DFT passiert also nichts weiter als die Zerlegung einer (diskreten) periodischen Funktion in ihre Frequenzen. Diese Frequenzen können visualisiert, analysiert und verändert werden. Dadurch ist es zum Beispiel möglich, Rauschen zu entfernen. Die störende Frequenz wird im Frequenzraum ausfindig gemacht und verworfen, somit ist sie im Ortsraum nicht mehr zu sehen [2, 3, 9, 11]. Ein Beispiel anhand eines digitalen Bildes ist im Anhang (Abbildung 13 auf Seite 19) zu finden.

Diese allgemeine Idee lässt sich ganz leicht auf Bilder übertragen, indem die Zeilen und Spalten des zweidimensionalen Bilds (bei Farbbildern wird jeder Farbkanal als eigens zweidimensionales Bild betrachtet) als nicht-periodische diskrete Funktionen aufgefasst werden [9, 11]. Die Intensitätswerte entsprechen einfach den Funktionswerten. Auch diese Funktionen lassen sich durch Aneinanderreihung von sich selbst, also durch Aneinanderreihung des Bildes, periodisieren.
Wendet man auf diese die *zweidimensionale DFT* an, so bekommt man als Ergebnis das Bild im Frequenzraum. Auch hier können nun die Frequenzen visualisiert, analysiert und verändert werden [2, 3, 9, 11]. Abbildung 13 auf Seite 19 zeigt die Darstellung eines digitalen Bildes im Ortsraum und die zugehörige Darstellung im Frequenzraum nach Anwendung der zweidimensionalen DFT. Man beachte, dass für die bessere Visualisierung im Frequenzraum die Quadranten des Bildes vertauscht werden, vergleiche Abbildung 14 auf Seite 19 (Anhang); der Ursprung des Bilds im Frequenzraum befindet sich folglich in der Mitte [11].

Mathematisch ist die zweidimensionale DFT definiert als

$$F_{mn} = \sum_{j=0}^{M-1} \sum_{k=0}^{N-1} f_{jk} \exp^{-i2\pi\left(\frac{mj}{M}+\frac{nk}{N}\right)}$$

und die zweidimensionale IDFT ist definiert als

$$f_{jk} = \frac{1}{MN} \sum_{j=0}^{M-1} \sum_{k=0}^{N-1} F_{mn} \exp^{i2\pi\left(\frac{mj}{M}+\frac{nk}{N}\right)}$$

Hierbei bezeichnet i die imaginäre Einheit, und es gilt: $0 \leq m \leq M-1, 0 \leq n \leq N-1$. Des weiteren referenzieren f_{jk} die einzelnen Intensitätswerte innerhalb der Bild-Matrix (wie in Unterabschnitt 1.1.1 auf Seite 1 beschrieben), F_{mn} sind die *Fourierkoeffizienten*. Fourierkoeffizienten sind komplexe Zahlen, deren Betrag die Amplitude einer Schwingung ist. Genauer ist diese Schwingung eine Überlagerung einer Schwingung in j-Richtung mit Frequenz m und einer Schwingung in k-Richtung mit Frequenz n. Die Phasen der Fourierkoeffizienten geben bezüglich des Nullpunkts die Verschiebung dieser überlagerten Schwingung an [2].
Da die Fourierkoeffizienten komplexe Zahlen sind, können sie nicht einfach dargestellt werden. Folglich zeigt Abbildung 13 auf Seite 19 für das Bild im Frequenzraum eigentlich "nur" die Amplituden der Fourierkoeffizienten an; je höher die Amplitude, desto höher der Intensitätswert (und desto heller der zugehörige Pixel). Niedrige Frequenzen sind hierbei um den Ursprung des Bildes (in der Mitte) verortet, hohe Frequenzen an den Rändern.

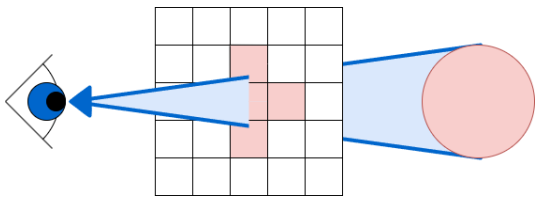
Etwas intuitiver und vereinfachter betrachtet, geben die Fourierkoeffizienten an, wie sehr sich die Intensitätswerte der Pixel in horizontale und vertikale Richtung verändern. Schnelle Veränderungen zeigen eine hohe Frequenz im Bild an, langsame Veränderungen eine niedrige Frequenz. Eine Beispiel ist in Abbildung 15 auf Seite 20 zu sehen.
Allgemein sind in Bildern niedrige Frequenzen in homogenen Bereichen (Beispiel: klarer Himmel, langsame Farbverläufe, konstante Farbe) zu finden und hohe Frequenzen in strukturierten Bereichen (Beispiel: Gras, Kanten, Ecken) [2, 10].

1.2 Grafikpipeline

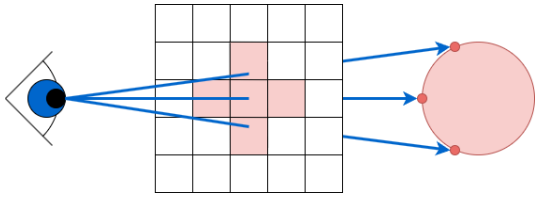
In Unterabschnitt 1.1.1 auf Seite 1 wurde erklärt, dass ein Teilnehmer während des VR-Navigationsexperiments eine schnelle Abfolge von Frames, beziehungsweise digitalen Bildern, sieht. Anschließend wurde darauf eingegangen, wie ein digitales Bild abgespeichert wird, damit es etwa dem Teilnehmer gezeigt werden kann. Es wurde noch nicht veranschaulicht, woher die notwendigen Daten für diese digitalen Bilder überhaupt stammen.

Zugrunde liegt dem VR-Navigationsexperiment eine mit Objekten gefüllte, dreidimensionale virtuelle Welt (Einleitung). Durch diese kann der Teilnehmer sich frei bewegen und er kann mit ihr interagieren. Die Aktionen des Teilnehmers finden folglich in einer *dreidimensionalen Szene* statt, werden dem Teilnehmer aber auf einem *zweidimensionalen* Bildschirm angezeigt. Hierzu müssen die dreidimensionalen Daten auf den Bildschirm *gerendert* werden.
"Rendern" bezeichnet dabei den Prozess, welcher die Daten der dreidimensionalen Szenen-Objekte, der Beleuchtung der Szene und der Position der virtuellen Kamera in der Szene in ein zweidimensionales Bild überführt; dieses kann dann auf dem Bildschirm angezeigt werden [13, 14].
Durchgeführt wird dieser Prozess von einer *Grafikpipeline* beziehungsweise *Rendering-Pipeline* [13, 15, 16]. Die in dieser Arbeit genutzte Grafikpipeline ist die *Built-in Render Pipeline* von Unity [17]. Diese nutzt *Forward Rendering*, um aus den Daten der dreidimensionalen Szene ein zweidimensionales Bild zu erstellen [18].
Beim Forward Rendering werden die dreidimensionalen Objekte in der Szene auf die zweidimensionale Bildebene *projiziert*. Das "Gegenstück" zum Forward Rendering ist das *Backward Rendering*, beziehungsweise *Ray Tracing*. Hierbei werden von der Bildebene Strahlen "geschossen", um herauszufinden, was für einen Farbwert ein Pixel annehmen muss [15, 19]. Abbildung 2 verdeutlicht diesen Unterschied.

Abbildung 2: Visueller Vergleich von Forward Rendering (a) und Backward Rendering (b). Bildquelle: Darstellung nach [15, 20].



(a) Forward Rendering: Die virtuellen Objekte werden auf den Bildschirm projiziert. Die Farbe eines Pixels hängt von der Farbe des auf das Pixel projizierten virtuellen Objektes ab.



(b) Backward Rendering: Vom Betrachter aus werden Strahlen "durch jedes Pixel des Bildschirms hindurchgeschossen". Die Farbe des Pixels hängt von der Farbe des getroffenen virtuellen Objektes ab.

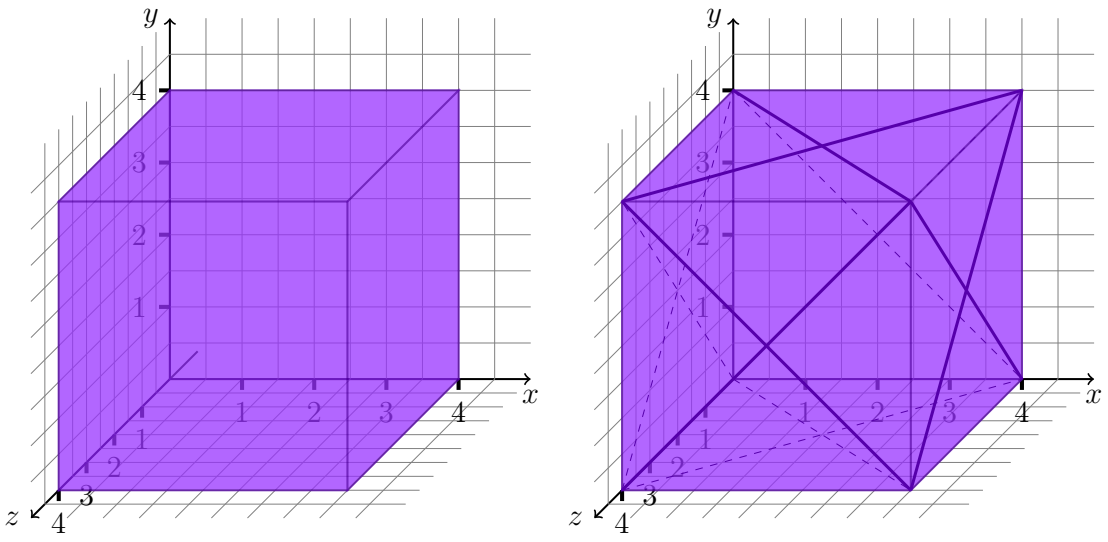
Im Folgenden werden die für das Verständnis von **??** auf Seite ?? notwendigen Teil der Grafikpipeline mit Forward Rendering erläutert. "Grafikpipeline" bezieht sich fortan im-

mer auf eine Pipeline mit Forward Rendering, “Rendering’ bezieht sich auf “Forward Rendering”.

1.2.1 Übersicht

In Abbildung 4 auf der nächsten Seite ist eine schematische Übersicht der Grafikpipeline gegeben. Zu Beginn stehen die in der Szene vorhandenen dreidimensionalen Objekte, beispielsweise eine lila Box (siehe Abbildung 4a). Diese Objekte besten aus vielen *Dreiecken*, welche wiederum durch eine Menge von drei *Vertices* (Singular: *Vertex*, bedeutet so viel wie *Knoten* oder *Punkt*) definiert sind [13, 15, 16]. **??** auf Seite ?? zeigt solchen Aufbau am Beispiel der lila Box.
Diese Objekte durchlaufen nun die Pipeline. Zuerst werden nur die Vertices bearbeitet (Abbildung 4 blaue Box), dann die *Primitiven* (hier: die Dreiecke; Abbildung 4 grüne Box) und schlussendlich die einzelnen Pixel beziehungsweise *Fragmente* (Abbildung 4 rote Box). Das Endergebnis ist ein zweidimensionales Bild, welches dem Teilnehmer angezeigt werden kann [13, 14, 15, 16].

Abbildung 3: Beispiel-Aufbau von dreidimensionale Objekten aus Primitiven (hier: Dreiecke).



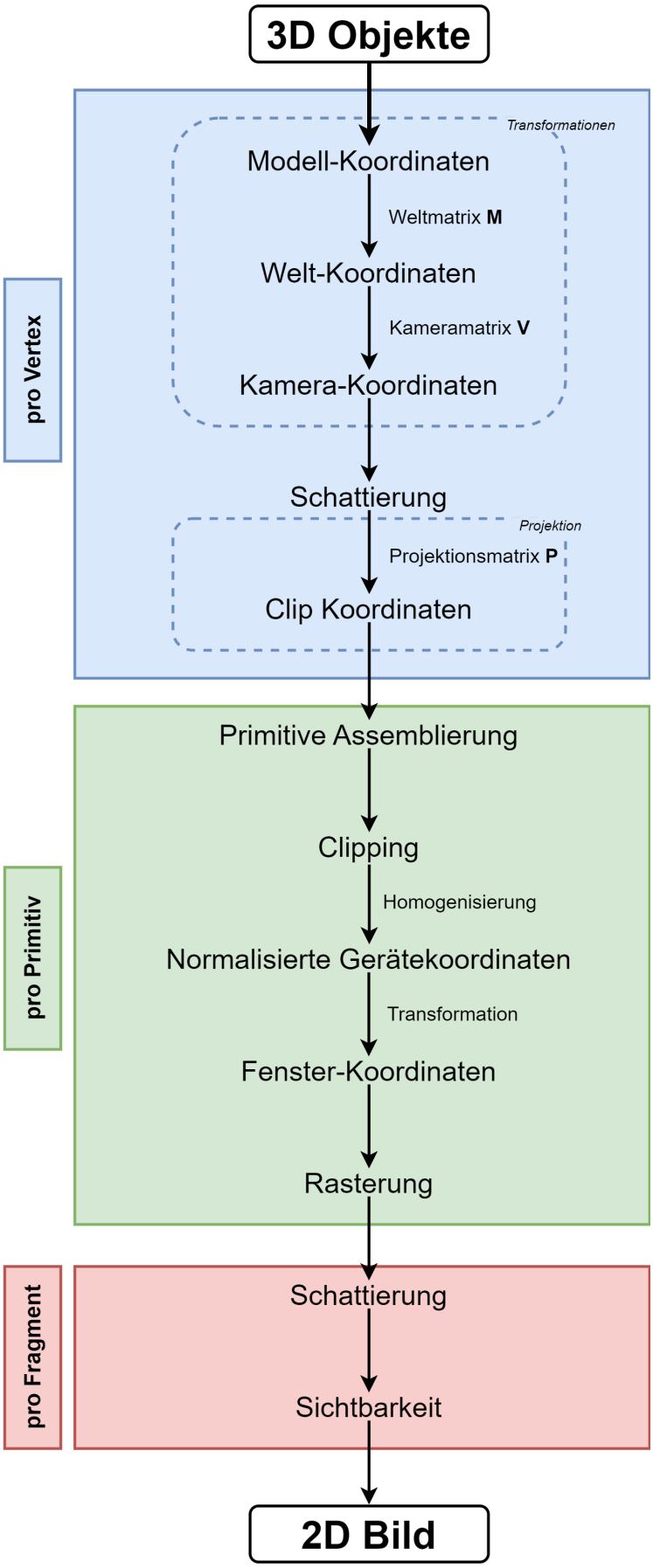
(a) Lila Beispiel-Box.

(b) Zerlegung der lila Beispiel-Box in Dreiecke.

Wie man Abbildung 4 auf der nächsten Seite entnehmen kann, beinhaltet die Grafikpipeline viel Schritte. Jedoch ist sie hoch parallelisierbar; alle Aufgaben innerhalb einer farblich gekennzeichneten Box können pro Datentyp parallel ausgeführt werden. Durch diese Parallelisierbarkeit ist die Grafikpipeline extrem auf die Verarbeitung von riesigen Datenmengen mittels der GPU (englisch: graphics processing unit) zugeschnitten [13, 14, 15]. Dadurch ist es sehr gut möglich Echtzeit-Aufgaben, wie die Auswertung von VR-Navigationsexperimenten, durchzuführen.

Die folgenden Unterabschnitte 1.2.2 bis 1.2.5 werden die einzelnen Vorgänge innerhalb der Grafikpipeline etwas näher beleuchten. Der Fokus liegt dabei auf den für **??** auf Seite ?? notwendigen Grundlagen.

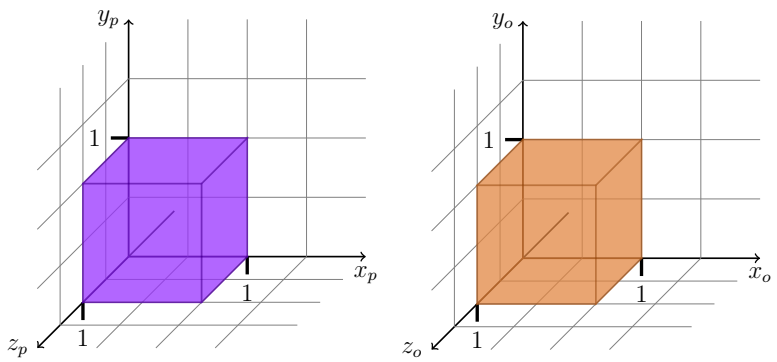
Abbildung 4: Darstellung der Grafikpipeline nach [13, 14, 15, 16]



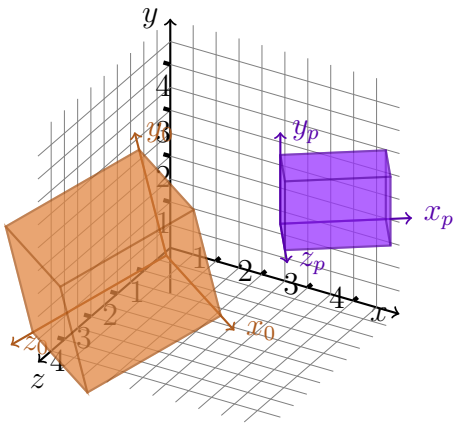
1.2.2 Vertex Operationen – Transformationen

Wie in Unterabschnitt 1.2.1 auf Seite 6 beschrieben, sind einzelnen Objekte innerhalb der Szene mittels einer Menge an Vertices (beziehungsweise Knoten oder Punkten) definiert. Damit diese Vertices miteinander in Beziehung gesetzt werden können, werden sie innerhalb eines (kartesischen) *Modell-Koordinatensystems* (auch *Objekt-Koordinatensystem* oder *lokales Koordinatensystem*) definiert. Dieses Koordinatensystem ist für jedes Objekt individuell [13, 15, 21]. Abbildung 6a zeigt dies für eine lila und eine orange Box.
Um die dreidimensionale Szene auf ein zweidimensionales Bild zu projizieren, müssen die in ihr vorhandenen Objekte jedoch miteinander vergleichbar sein. Hierzu werden sie in ein gemeinsames (kartesisches) *Welt-Koordinatensystem* (auch *globales Koordinatensystem*) *transformiert*. Innerhalb des Welt-Koordinatensystems kann jeder Vertex eindeutig referenziert werden [13, 15, 21]. In Abbildung 6b wird dies für die Beispiel-Boxen dargestellt.

Abbildung 5: Beispiel-Boxen in unterschiedlichen Koordinatensystemen.



(a) Die Beispiel-Boxen innerhalb ihrer lokalen Koordinatensysteme.



(b) Die Beispiel-Boxen innerhalb des Welt-Koordinatensystems. Die lokalen Koordinatensystem sind realtiv zu diesem eingezeichnet.

Die Überführung von Modell-Koordinaten zu Welt-Koordinaten erfolgt mittels *Transformationen*. Hierzu sind drei unterschiedliche Transformationen notwendig: *Skalierung*, *Rotation* und *Translation*. Diese sind im Grunde nichts anderes als Abbildungen, die auf die Vertices angewandt werden [13, 14, 15, 16, 21]. In Unterabschnitt 2.2 auf Seite 21 (Anhang) ist dies anhand der lila Beispiel-Box beispielhaft beschrieben.
Um ein ganzes Objekt zu transformieren, müssen auf jeden Vertex des Objekts die gleichen Transformationen in der gleichen Reihenfolge angewandt werden [15]. Ein Beispiel, bei dem dies schiefgeht, ist in Abbildung 16 auf Seite 21 (Anhang) zu sehen.

Zur effizienten Gestaltung der Transformation werden Matrizen verwendet. Jedes Objekt besitzt seine eigene *Weltmatrix* **M** (auch *world matrix* oder *model matrix*). Diese setzt sich aus einzelnen Transformationsmatrizen zusammen, die zur Überführung von Modell- zu Welt-Koordinaten für das spezifische Objekt notwendig sind [13, 14, 15, 16, 21].
In Unterabschnitt 2.2 auf Seite 21 (Anhang) befindet sich eine detailliertere Beschreibung der unterschiedlichen Transformationsmatrizen anhand von Beispielen.

Ein wichtiger Aspekt der Transformationsmatrizen ist, dass es sich um *affine Transformationen* und nicht um lineare Transformationen handelt. Affine Transformationen vereinen lineare Transformationen mit einer Translation [13, 15].
Im dreidimensionalen Raum der Szene werden hierzu vierdimensionale Matrizen verwendet. In diesen 4×4 Matrizen werden Skalierungen und Rotationen über die linke, obere 3×3 Matrix realisiert und die Translation über die vierte Spalte [13, 15, 16].
Durch die Nutzung von 4×4 Matrizen an dieser und den folgenden Stellen, müssen die dreidimensionalen Vertices um eine Koordinate w erweitert werden: Aus den Koordinaten des Vertex $v = (x, y, z)^T$ werden die Koordinaten $v = (x, y, z, w)^T$ [13, 15, 16].
Bei dieser Erweiterung der kartesischen Koordinaten handelt es sich um *homogene Koordinaten*. Neben der Nutzung von affinen Transformationen wird ebenfalls die notationelle Unterscheidung zwischen Vektoren und Punkten möglich. Ein Vektor ist gegeben als $\vec{q} = (x, y, z, 0)^T$ ($w = 0$), ein Punkt ist gegeben als $q = (x, y, z, 1)^T$ ($w = 1$). Da es sich bei den Vertices um Punkte und nicht um Vektoren handelt, gilt folglich $v_i = (x_i, y_i, z_i, 1)^T$ für alle Vertices v_i [13, 15, 16].

Nachdem alle Vertices der Szene in Welt-Koordinaten vorliegen, müssen diese nochmals in ein anderes Koordinatensystem transformiert werden. Dies trägt zur Effizienz der Durchführung von der in Unterabschnitt 1.2.3 auf der nächsten Seite beschriebenen Projektion und dem in Unterabschnitt 1.2.4 auf Seite 14 beschriebenen Clipping bei [13, 15, 21].

Die Transformation erfolgt mittels der *Kameramatrix* **V** (auch *view matrix* oder *eye matrix*); die Vertex-Koordinaten liegen anschließend als *Kamera-Koordinaten* (auch *eye coordinates*) vor [13, 14, 15].
Wie obiger Benennung zu entnehmen ist, hat das neue Referenz-Koordinatensystem Bezug zur *Kamera*. Die Kamera ist ebenfalls ein Objekt der Szene und wurde folglich mittels ihrer Weltmatrix **M** im Welt-Koordinatensystem platziert. Während des VR-Navigationsexperiments kann der Teilnehmer das Kamera-Objekt bewegen und nimmt durch sie die virtuelle Welt wahr; als wären seine Augen die Kamera (Einleitung). Definiert ist die Kamera über *extrinsische Parameter* und *intrinsische Parameter* [15, 22]; zur Beschreibung der Transformation sind die extrinsischen Parameter notwendig.

Die extrinsischen Parameter beschreiben Position und Ausrichtung der Kamera, diese sind gegeben durch

- die Position der Kamera im Welt-Koordinatensystem $e = (x, y, z, 1)^T$,
- die Blickrichtung der Kamera $\vec{v}$ (Vektor),
- den von der Kamera aus nach oben zeigenden Vektor $\vec{u}$,
- und den von der Kamera aus nach rechts zeigenden Vektor $\vec{r}$.

wobei angenommen wird, dass $\vec{v}, \vec{u}, \vec{r}$ orthonormal sind (siehe Abbildung 20a auf Seite 28) [15, 22, 23].

Für eine *Standard-Kamera* (eine Kamera, die sich am Ursprung befindet und die negative z-Achse entlang schaut; siehe Abbildung 20b auf Seite 28) sind die extrinsischen Parameter folglich

- $e = (0, 0, 0, 1)^T$,
- $\vec{v} = (0, 0, -1, 0)^T$,
- $\vec{u} = (0, 1, 0, 0)^T$,
- und $\vec{r} = (1, 0, 0, 0)^T$.

Das neue Referenz-Koordinatensystem ist nun eines, bei welchem das Kamera-Objekt der Szene einer Standard-Kamera entspricht; die Welt orientiert sich um die "Augen" des Teilnehmers herum.
Folglich transformiert die Kameramatrix **V** die Kamera so, dass sie sich im Ursprung befindet und die negative z-Achse entlang schaut. Alle anderen Objekte in der Szene werden ebenfalls mit der Kameramatrix transformiert, wodurch die Objekte relativ zueinander weiterhin gleich angeordnet sind.
In Unterabschnitt 2.3 auf Seite 27 (Anhang) befindet sich eine ausführlichere Herleitung der Kameramatrix **V**.

Zusammenfassend wurde in diesem Unterabschnitt der erste Teil der Grafikpipeline in Abbildung 4 auf Seite 7 erläutert. Alle Vertices v_i der Objekte innerhalb der Szene werden mittels der korrespondierenden Weltmatrix $\mathbf{M}_j$ ($j \in [0, \text{Anzahl Objekte} - 1]$) und der Kameramatrix **V** in ein Referenz-Koordinatensystem transformiert. Sie liegen jetzt in Kamera-Koordinaten vor: $v_i' = \mathbf{V} \cdot \mathbf{M}_j \cdot v_i$.
Anschließend durchlaufen die Vertices die *Schattierung* (englisch: *shading*). Hierbei wird die Beleuchtung für jeden Vertex berechnet [13, 15]. Da dieser Schritt nicht für **??** auf Seite ?? notwendig ist, wird er hier nicht näher beleuchtet.

1.2.3 Vertex Operationen – Projektion

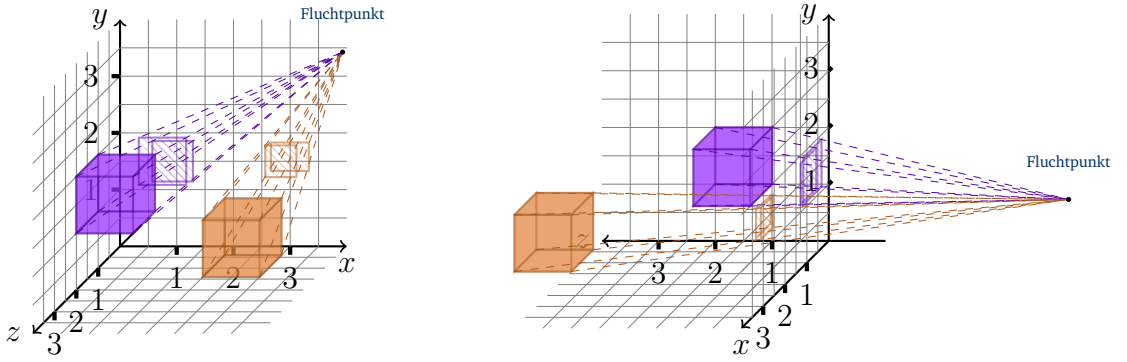
Der Ausgangspunkt für die *Projektion* (siehe Abbildung 4 auf Seite 7 blaue Box unten) sind die Vertices in Kamera-Koordinaten. Das heißt, die Objekte der Szene befinden sich in einem Koordinatensystem, in welchem das Kamera-Objekt im Ursprung liegt und die negative z-Achse entlang schaut (siehe Unterabschnitt 1.2.2).
Eine Projektion ist in diesem Fall eine Abbildung des dreidimensionalen Raums auf die zweidimensionale Bildebene [13, 15]. Sie kann entweder als *orthographische Projektion* oder als *perspektivische Projektion* erfolgen [13, 15, 16].

Bei einer orthographischen Projektion werden die einzelnen Vertices auf eine gegebene Ebene projiziert, beispielsweise auf die xy–Ebene (siehe Abbildung 7a auf der nächsten Seite). Die Projektionsstrahlen sind dabei parallel und stehen orthogonal auf der Bildebene. Hierdurch bleiben Parallelitäten und Längen erhalten, jedoch kommt es zu keiner perspektivischen Verzerrung; das Ergebnis wirkt weniger realitätsgetreu [13, 15, 16].
Aus diesem Grund wird in vielen Anwendungen, wie auch bei den VR-Navigationsexperimenten, die perspektivische Projektion genutzt. Bei dieser bleiben Parallelitäten nicht erhalten, die Kanten laufen auf Fluchtpunkte zu. Dadurch erscheinen weiter entfernte Objekte kleiner als nahe Objekte und es kommt beispielsweise zu perspektivischer Verkürzung (englisch: perspective foreshortening). Dies stimmt eher mit unserer Wahrneh-

mung der Welt überein [13, 15, 16]. Abbildung 7b zeigt die Auswirkung dieser Projektion auf die Beispiel-Boxen.

Abbildung 6: Vergleich einer orthographischen Projektion (a) und einer perspektivischen Projektion (b). Die gestrichelten Linien symbolisieren die Projektionsstrahlen.

(a) Orthographische Projektion: Die Beispiel-Boxen werden mittels orthogonal zur Bildebene stehenden Projektionsstrahlen auf die xy-Ebene projiziert. Es kommt zu keiner perspektivischen Verzerrung.



(b) Perspektivische Projektion: Die Beispiel-Boxen werden mittels auf einen Fluchtpunkt (entspricht der Kamera-Position) zulaufende Projektionsstrahlen auf die xy-Ebene projiziert. Es kommt zu perspektivischer Verzerrung.

Im Folgenden wird auf die perspektivische Projektion näher eingegangen, ein Beschreibung der orthographischen Projektion befindet sich im Anhang (Unterabschnitt 2.4 auf Seite 29).

Bei der perspektivischen Projektion werden die Vertices von Kamera-Koordinaten in *Clip-Koordinaten* transformiert, folglich befindet sich die Szene anschließend im *Clip Raum* (englisch: *Clip Space*) [16, 21]. Dies ist der Raum, in welchem das *Clipping* stattfindet. Das heißt, die (Teil der) Objekte, die nicht innerhalb des Sichtbereichs der Kamera liegen, werden abgeschnitten (*geclippt*; siehe Unterabschnitt 1.2.4 auf Seite 14) [13, 15, 21].

Definiert ist der Sichtbereich der Kamera durch ein *Frustum*, siehe Abbildung 8b auf Seite 13. Dieses Frustum ist gegeben durch die intrinsischen Parameter der Kamera (vergleiche extrinsische Parameter in Unterabschnitt 1.2.2 auf Seite 9) :

- n (für *near plane*): Gibt an, wo der Sichtbereich der Kamera nach vorne hin beginnt. Alle Punkte davor werden nicht von der Kamera gesehen.
- f (für *far plane*): Gibt an, wo der Sichtbereich der Kamera nach hinten hin endet. Alle Punkte dahinter werden nicht von der Kamera gesehen.

- l (für *left plane*): Gibt an, wo der Sichtbereich der Kamera nach links hin beginnt. Alle Punkte links davon werden nicht von der Kamera gesehen.
- r (für *right plane*): Gibt an, wo der Sichtbereich der Kamera nach rechts hin endet. Alle Punkte rechts davon werden nicht von der Kamera gesehen.
- b (für *bottom plane*): Gibt an, wo der Sichtbereich der Kamera nach unten hin beginnt. Alle Punkte darunter werden nicht von der Kamera gesehen.
- t (für *top plane*): Gibt an, wo der Sichtbereich der Kamera nach oben hin endet. Alle Punkte darüber werden nicht von der Kamera gesehen.

[13, 15, 16]
Da die Kamera die negative z-Achse entlang schaut, gilt: $n > f$ [13, 15]. Der horizontale Blickwinkel ist durch den Abstand von l und r gegeben, der vertikale Blickwinkel durch den Abstand von b und t und die zusehende Distanz ist durch den Abstand von n und f gegeben [13].

Durch die perspektivische Projektion wird dieses Frustum auf den $[-1,1]\times[-1,1]\times[-1,1]$ Einheitswürfel abgebildet, siehe Abbildung 8b auf der nächsten Seite [13, 15, 16, 21]. Die Objekte befinden sich nun im Clip Raum und die Koordinaten aller Vertices sind als Clip Koordinaten angegeben [16, 21].
Durchgeführt wird diese Transformation von der *Projektionsmatrix* **P**, diese ist für alle Vertices gleich. Sie ist definiert als

$$\mathbf{P} = \begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{l+r}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{b+t}{t-b} & 0 \\ 0 & 0 & -\frac{n+f}{f-n} & -\frac{2nf}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix} \tag{1}$$

wobei l, r, b, t, n, f sich auf die oben definierten intrinsischen Kamera-Parameter beziehen [13, 15, 16].

Für **??** auf Seite ?? ist ein detailliertes Verständnis über den Aufbau der Projektionsmatrix **P** (Gleichung (1)) nicht notwendig (für Interessierte ist im Anhang (Unterabschnitt 2.4 auf Seite 30) eine Herleitung zu finden). Es sei jedoch die letzte Zeile der Matrix hervorgehoben

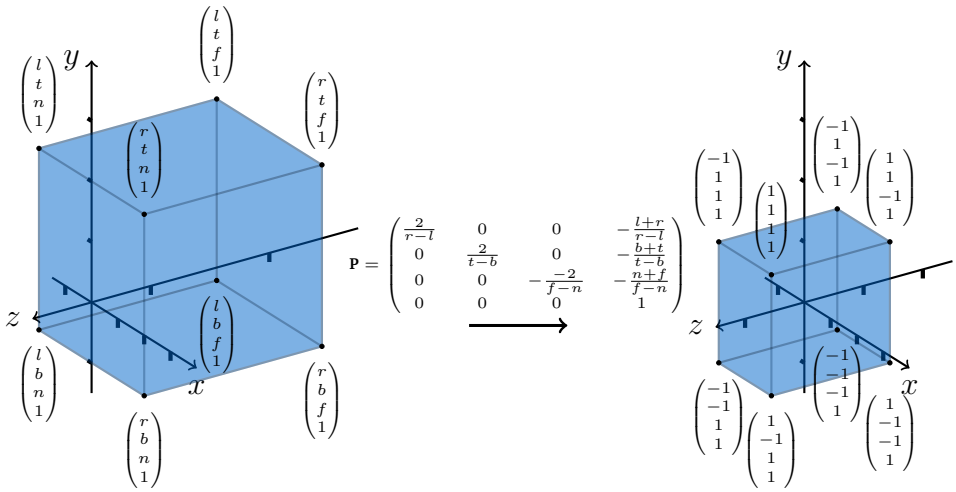
$$\begin{pmatrix} 0 & 0 & -1 & 0 \end{pmatrix}$$

Multipliziert man die Projektionsmatrix **P** von links an einen Vertex v, so ändert sich aufgrund der letzten Zeile dessen w-Koordinate; anstelle von $w = 1$ steht $-z$ in ihr. Dies widerspricht der in Unterabschnitt 1.2.2 auf Seite 9 eingeführten Definition eines Punkts; w ist nicht mehr zwangsweise 1. Dieses "Problem" lässt sich jedoch durch *Homogenisierung* (englisch: *homogenization* oder auch *perspective division*) lösen [13, 15].
Nach der Anwendung von **P** hat ein Vertex $v = (x, y, z, w)^T$ die neuen Koordinaten $v' = (w'x', w'y', w'z', w')$, teilt man diese durch w' erhält man $v' = (x', y', z', 1)$; nach der Homogenisierung ist v' (der Definition in Unterabschnitt 1.2.2 auf Seite 9 nach) ein Punkt. Punkte können dementsprechend von nun an durch $p = (wx, wy, wz, w)^T, w \neq 0$ dargestellt werden, Vektoren werden weiterhin durch $\vec{p} = (x, y, z, 0)$ dargestellt [13, 15].

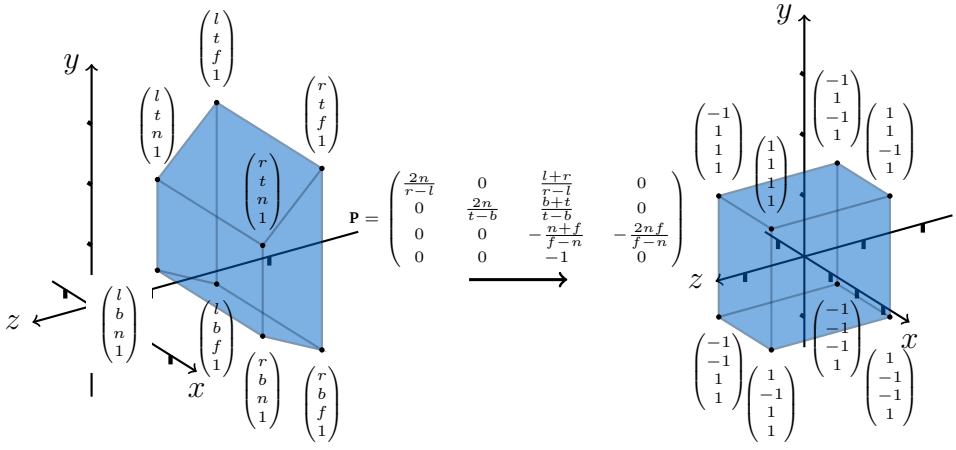
Zu diesem Zeitpunkt in der Grafikpipeline werden die Koordinaten der Vertices jedoch noch nicht homogenisiert (vergleiche Abbildung 4 auf Seite 7), sie werden als Clip-Koordinaten in der Pipeline weitergegeben.

Diese Clip-Koordinaten sind auch weiterhin dreidimensional (beziehungsweise vierdimensional durch w), es ist keine Dimension "verloren gegangen". Dies scheint erstmals erstaunlich, da bei einer Projektion eines Punkts auf die Bildebene die Tiefeninformation wegfallen sollte.
Um jedoch herauszufinden, welches Objekt ein anderes überdeckt, ist es notwendig die Tiefeninformation zu haben; von der Kamera aus verdeckte Objekte werden auf dem Bildschirm nicht dargestellt, von der Kamera aus sichtbare Objekte schon. Aus Effizienzgründen findet dieser Vergleich erst später in der Pipeline statt und die Tiefeninformationen werden so lange mitgespeichert [13, 15, 21].

Abbildung 7: Vergleich einer orthographischen Projektion (a) und einer perspektivischen Projektion (b). Die gestrichelten Linien symbolisieren die Projektionsstrahlen.



(a) Orthographische Projektion: Der Sichtbereich der Kamera ist durch eine Box gegeben. Diese wird mittels der Projektionsmatrix **P** auf den Einheitswürfel abgebildet.



(b) Perspektivische Projektion: Der Sichtbereich der Kamera ist durch ein Frustum gegeben. Dieses wird mittels der Projektionsmatrix **P** auf den Einheitswürfel abgebildet.

1.2.4 Primitiven Operationen

Nachdem in Unterabschnitte 1.2.2 und 1.2.3 der erste Abschnitt der Grafikpipeline (Abbildung 4 auf Seite 7 blaue Box) erklärt wurde, beschäftigt sich dieser Unterabschnitt mit dem zweiten Teil der Grafikpipeline (Abbildung 4 auf Seite 7 grüne Box).

Ausgangspunkt sind die in der Szene vorhandenen Vertices in Clip Koordinaten.

Zu Beginn dieses Grafikpipeline-Abschnitts werden die Vertices miteinander verbunden, sodass sich die ursprünglichen Primitive (hier: Dreiecke) ergeben; waren Vertices v_1, v_3, v_7 vor Beginn der Pipeline miteinander verbunden, so werden sie auch nun miteinander verbunden. Dieser Schritt heißt *Primitive Assemblierung* (englisch: *primitive assembly*) [13, 14, 15]. Diese Primitive werden nun zum *Clipping* genutzt.

Wie Unterabschnitt 1.2.3 auf Seite 11 angesprochen, werden beim Clipping die (Teil der) Objekte beziehungsweise Primitive, die nicht innerhalb des Sichtbereichs der Kamera liegen, abgeschnitten. Wobei zu diesem Zeitpunkt die Objekte innerhalb des Sichtbereichs der Kamera jene sind, die sich innerhalb des Einheitswürfels befinden. Das Clipping dient dazu, die weiterzubearbeitende Datenmenge zu reduzieren; Primitive, die durch die Kamera nicht sichtbar sind, müssen nicht für die Darstellung auf dem Bildschirm vorbereitet werden [13, 14, 15]. In Unterabschnitt 2.6 auf Seite 33 (Anhang) ist hierzu ein Beispiel zu finden.

Die nach dem Clipping verbleibenden Primitive werden, wie in Unterabschnitt 1.2.3 auf Seite 12 beschrieben, homogenisiert. Alle Koordinaten eines Punktes werden folglich durch dessen w-Koordinate geteilt. Die dadurch erhaltenen Koordinaten heißen *normalisierte Gerätekoordinaten* (englisch: *normalized device coordinates*, kurz: *NDC*) [13, 14, 15].
Für normalisierten Gerätekoordinaten eines Vertex v gilt

$$-1 \leq x, y, z \leq 1, w = 1$$

Alle Vertices (und somit Primitive) liegen innerhalb des $[-1, 1] \times [-1, 1] \times [-1, 1] \times 1$ Körper beziehungsweise innerhalb des Einheitswürfels [1] [13, 14, 15].

Die normalisierten Gerätekoordinaten der Primitive innerhalb des Einheitswürfels werden ein letztes Mal transformiert; hierbei wird auf die "Zielgröße" des zweidimensionalen Bilds auf dem Bildschirm transformiert.
Die dabei resultierenden x- und y-Koordinaten der Primitive werden als *Bildschirm-Koordinaten* (englisch: *screen coordinates*) referenziert, zusammen mit der z-Koordinate werden sie auch *Fenster-Koordinaten* (englisch: *window coordinates* oder *pixel coordinates*) genannt [13, 14, 15, 16].

Hierbei sind die Fenster-Koordinaten definiert als $[l, l + w] \times [b, b + h] \times [0, 1]$ Körper, wobei l, w, b, h von den Maßen des Ziel-Bildschirms und der Position des Bilds auf den Bildschirm abhängen.

- w (für *width*) gibt an, wie breit der Bildschirm ist.
- h (für *height*) gibt an, wie hoch der Bildschirm ist.
- l (für *left*) gibt an, wo von links an auf dem Bildschirm das Bild beginnen soll. $l = 0$ hieße, dass das Bild an der linken Grenze des Bildschirms beginnen soll.

[1] Anmerkung: Technisch gesehen waren die Koordinaten eines Vertex während des Clippings folglich noch nicht innerhalb des Einheistwürfels, sondern innerhalb des $[-w, w] \times [-w, w] \times [-w, w] \times w$ Körpers (wobei sich w auf die jeweilige vierte Koordinate eines Vertex bezieht). In der Literatur wurde hier jedoch kein klarer Unterschied gezogen [13, 14, 15]. An dieser Stelle ist allerdings nur wichtig, dass sich nach der Homogenisierung alle Koordinaten der (von der Kamera aus sichtbaren) Vertices innerhalb des Einheitswürfels befinden.

- b (für *bottom*) gibt an, wo von unten an auf dem Bildschirm das Bild beginnen soll. $b = 0$ hieße, dass das Bild an der unteren Grenze des Bildschirms beginnen soll.

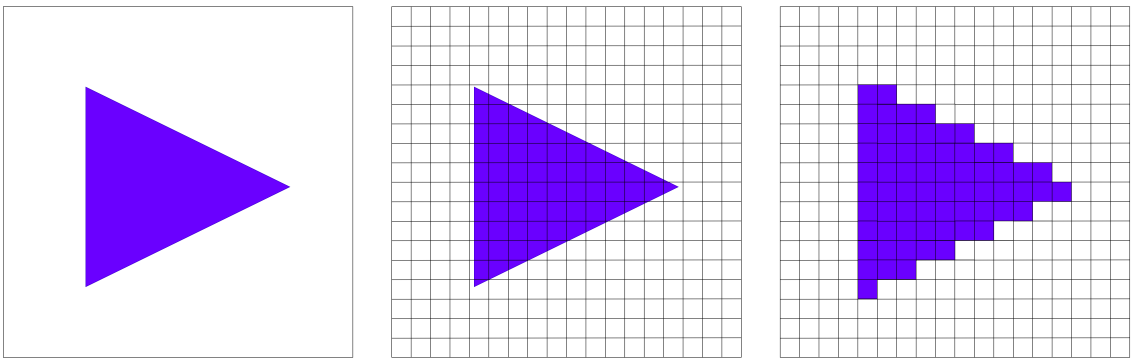
[13, 15]
Diese Transformation ist einfach als Matrix zu notieren

$$\begin{pmatrix} \frac{w}{2} & 0 & 0 & \frac{w}{2} + l \\ 0 & \frac{h}{2} & 0 & \frac{h}{2} + b \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Die x, y-Koordinaten werden auf die richtige Größe für den Bildschirm skaliert und an die korrekte Stelle auf dem Bildschirm verschoben. Die z-Koordinate wird in einen Bereich zwischen 0 und 1 abgebildet, wobei ein z-Wert von 0 bedeutet, dass sich der Vertex auf der near plane befindet und eine ein z-Wert von 1 bedeutet, dass sich der Vertex auf der far plane befindet. Die w-Koordinate wird nicht verändert [15].

Nach dieser letzten Transformation sind nun alle Koordinaten der Primitive (innerhalb des Sichtbereichs der Kamera) auf Koordinaten auf dem Bildschirm abgebildet. Während der *Rasterung* (englisch: *rasterization*) werden diese den korrespondierenden Bildschirm-Pixel zugeordnet (siehe Abbildung 8). Das Ergebnis ist eine Menge von Pixeln beziehungsweise *Fragmenten*, welche in der Grafikpipeline weiter bearbeitet werden [13, 14, 15, 16].
Jedes Fragment enthält dabei mehrere Informationen, beispielsweise seine Position auf dem Bildschirm, seine derzeitige Farbe, seinen z-*Wert* (auch *depth value*) oder auch seine Position im Clip Raum [13, 16].

Abbildung 8: Darstellung der Rasterung am Beispiel eines lila Dreiecks. Bildquelle: Darstellung nach [24, 25].



1.2.5 Fragment Operationen

In den Unterabschnitte 1.2.2 bis 1.2.4 wurde erklärt, wie die dreidimensionalen Objekte der Szene mittels Transformationen auf die einzelnen Pixel des Bildschirms abgebildet werden (vergleiche Abbildung 4 auf Seite 7 blaue und grüne Box). Dieser Unterabschnitt gibt einen kurzen Überblick über die letzten Schritte der Grafikpipeline (Abbildung 4 auf Seite 7 rote Box).

Während der *Schattierung* (englisch: *shading*) wird jedem Fragment seine finale Farbe zugewiesen. Grundlage hierfür können die pro Vertex berechneten Farbwerte sein (vergleiche Unterabschnitt 1.2.2 auf Seite 10), mit welchen interpoliert wird. Es kann aber

auch direkt eine Farbe (beispielsweise weiß mit $(255, 255, 255)$ beziehungsweise $(1, 1, 1)$) jedem Fragment zugewiesen werden. Das Endergebnis ist immer eine Farbe für jedes Fragment [13, 14, 15, 16].

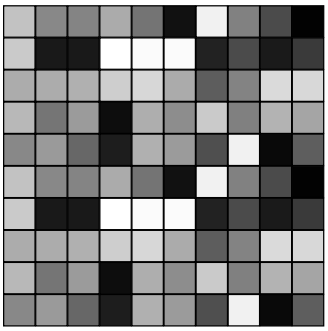
Nachdem für jedes Fragment eine Farbe bestimmt wurde, wird in einem letzten Schritt überprüft welche Fragment auf dem Bildschirm zu sehen sein sollen. Bisher wurde noch nicht überprüft, ob ein Objekt von einem anderen überdeckt wird und dementsprechend gar nicht auf dem Bildschirm zu sehen ist.
Hierzu wird für jedes Fragment f_1 überprüft, ob sich an dessen vorhergesehenen Bildschirm-Position schon ein anderes Fragment f_2 befindet. Wenn dem so ist, werden die z-Werte der Fragmente verglichen. Ist $z_{f_1} < z_{f_2}$, liegt das zu f_1 gehörige Objekt näher an der Kamera; f_1 wird an die Bildschirm-Position geschrieben. Ist $z_{f_2} < z_{f_1}$, liegt das zu f_2 gehörige Objekt näher an der Kamera; f_2 bleibt an der Bildschirm-Position [13, 14, 15, 16].
Dieser Vorgang wird auch als z-*testing* oder z-*buffering* bezeichnet. Die Farbwerte jedes Fragments liegen hierbei im *color buffer* und die z-Werte im z *buffer* (auch *depth buffer*) vor. Wobei *buffer* nichts anders als Zwischenspeicher sind [13, 14, 16].

Das Endergebnis des letzten Abschnitts der Grafikpipeline (Abbildung 4 auf Seite 7 rote Box) ist das finale zweidimensionale Bild, dass auf dem Bildschirm angezeigt werden kann.

2 Anhang

2.1 Digitales Bild – Beispiel-Abbildungen

Abbildung 9: Zeigt an einem Beispiel-Intensitätsbild (a) wie dieses als Matrix (b) notiert und abgespeichert wird.

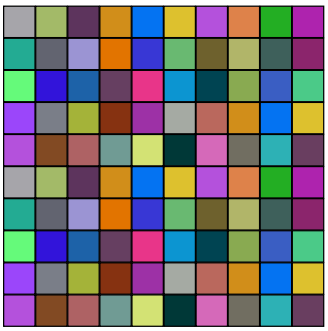


(a) Beispiel-Intensitätsbild.

61	119	124	84	140	239	14	127	181	254
43	53	231	0	4	4	222	181	230	198
83	83	79	49	40	84	163	124	37	39
71	139	100	243	81	114	53	128	76	90
119	101	153	227	79	100	177	14	247	162
61	119	124	84	140	239	14	127	181	254
43	53	231	0	4	4	222	181	230	198
83	83	79	49	40	84	163	124	37	39
71	139	100	243	81	114	53	128	76	90
119	101	153	227	79	100	177	14	247	162

(b) Zugehörige Matrix.

Abbildung 10: Zeigt an einem Beispiel-RGB-Farbbild (a) wie dieses als Matrizen (b) -(d) notiert und abgespeichert wird. Die Intensitätswerte der Farbkanäle sind mit der jeweils korrespondierenden Farbe visualisiert.



(a) Beispiel-RGB-Farbbild.

166	163	93	209	4	221	4	222	35	174
35	98	155	226	56	105	110	177	62	139
101	50	29	102	232	12	0	137	58	75
155	122	164	134	157	165	186	209	4	221
180	130	174	112	211	0	213	113	45	105
166	163	93	209	4	221	4	222	35	174
35	98	155	226	56	105	110	177	62	139
101	50	29	102	232	12	0	137	58	75
155	122	164	134	157	165	186	209	4	221
180	130	174	112	211	0	213	113	45	105

(b) Zum Rot-Kanal zugehörige Matrix.

165	186	52	142	115	193	81	130	174	35
171	100	148	116	55	185	97	181	96	37
250	20	98	63	53	149	68	170	94	202
70	126	179	50	49	170	104	142	115	193
81	74	98	155	226	56	105	110	177	62
165	186	52	142	115	193	81	130	174	35
171	100	148	116	55	185	97	181	96	37
250	20	98	63	53	149	68	170	94	202
70	126	179	50	49	170	104	142	115	193
81	74	98	155	226	56	105	110	177	62

(c) Zum Grün-Kanal zugehörige Matrix.

170	104	93	26	242	46	220	74	35	174
147	112	211	0	213	113	45	105	91	108
122	218	163	97	137	209	82	81	195	136
250	136	57	17	166	163	93	26	242	46
220	35	100	148	116	55	185	97	181	96
170	104	93	26	242	46	220	74	35	174
147	112	211	0	213	113	45	105	91	108
122	218	163	97	137	209	82	81	195	136
250	136	57	17	166	163	93	26	242	46
220	35	100	148	116	55	185	97	181	96

(d) Zum Blau-Kanal zugehörige Matrix.

Abbildung 11: RGB-Farbraum. Bildquelle: [26].

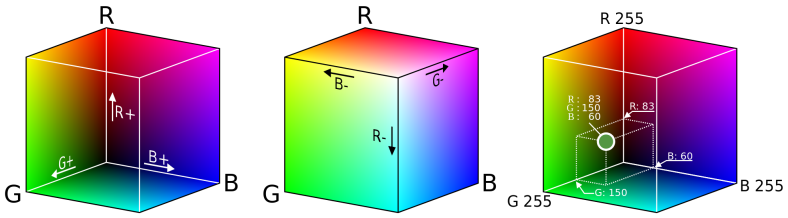
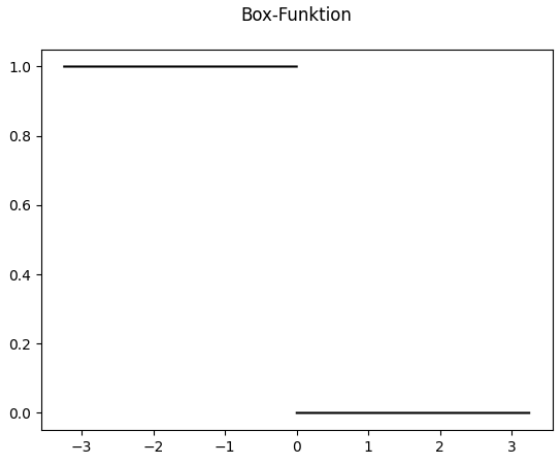
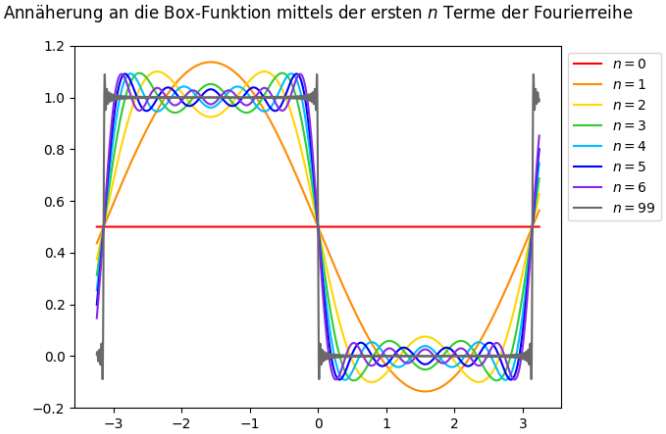


Abbildung 12: Annäherung an die Box-Funktion mittels Überlagerung von Sinus-Funktionen.

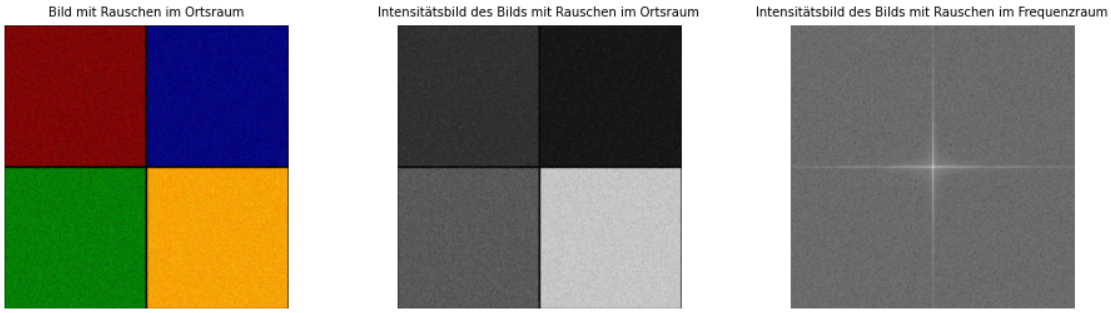


(a) Darstellung der Box-Funktion. Diese ist gegeben durch $f(x) = \begin{cases} 1 & x \in (-\pi, 0) \\ 0 & x \in (0, \pi) \end{cases}$.



(b) Annäherung an die Box-Funktion mittels der ersten n Terme der Fourierreihe. Für die Box-Funktion ist die Fourierreihe gegeben durch $f(x) = \frac{1}{2} - \frac{2}{\pi} \sum_{n=1} \frac{\sin((2n-1)x)}{2n-1}$ [27].

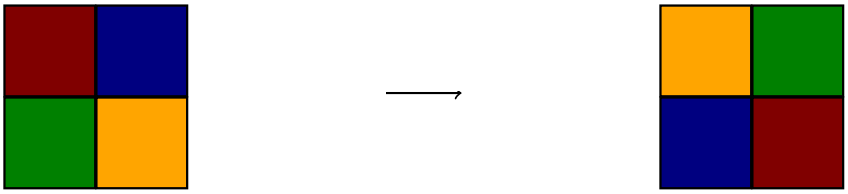
Abbildung 13: Visualisierung des Orts- und Frequenzraums anhand eines Beispiel-Bilds.



(a) Darstellung eines digitalen Bilds mit Rauschen im Orts- und Frequenzraum.

(b) Darstellung obigen digitalen Bilds im Orts- und Frequenzraum nach Glättung des Rauschens im Frequenzraum.

Abbildung 14: Veranschaulichung der Verschiebung der Quadranten zur Darstellung eines digitalen Bildes im Frequenzraum; Bildquelle: Darstellung nach [11].



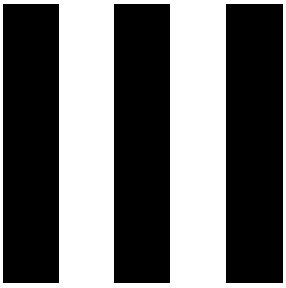
(a) Lage der Quadranten eines digitalen Bildes im Ortsraum.

(b) Lage der Quadranten eines digitalen Bildes im Frequenzraum.

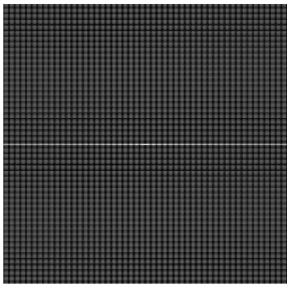
Abbildung 15: Matrix mit Intensitätswerten und deren zugehörigen Darstellungen im Orts- und Frequenzraum. Die horizontale weiße Linie in der Abbildung des Frequenzraums zeigt an, dass das Bild hohe Frequenzen in horizontaler Richtung enthält. Dies ist auch der Fall, auf horizontaler Ebene wechseln die Matrix-Einträge schnell zwischen 0 und 255 hin und her. Die Abwesenheit einer vertikalen weißen Linie in der Abbildung des Frequenzraums zeigt an, dass das Bild keine Frequenzen in vertikaler Richtung enthält. Dies ist auch der Fall, auf vertikaler Ebene sind die Matrix-Einträge konstant 0 oder konstant 255.

0	255	0	255	0
0	255	0	255	0
0	255	0	255	0
0	255	0	255	0
0	255	0	255	0

(a) Digitales Bild als Matrix.



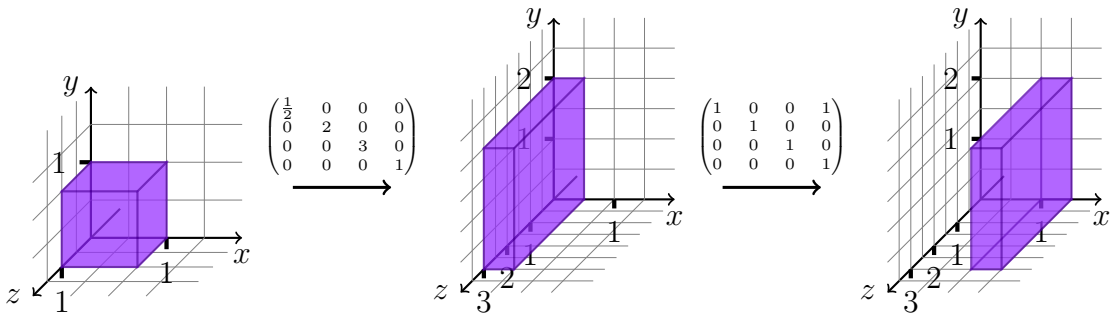
(b) Digitales Bild im Ortsraum.



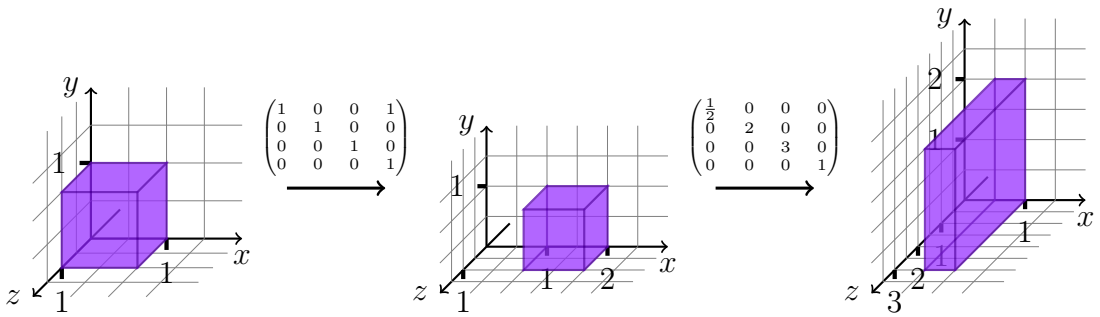
(c) Digitales Bild im Frequenzraum.

2.2 Vertex-Operationen – Transformationsmatrizen

Abbildung 16: Visualisierung der Wichtigkeit, Transformationen in der gleichen Reihenfolge anzuwenden.



(a) Anwendung einer Skalierung gefolgt von einer Translation.



(b) Anwendung einer Translation gefolgt von einer Skalierung.

Im Folgenden werden die Transformationsmatrizen zur Überführung von Vertices in Modell-Koordinaten zu Vertices in Welt-Koordinaten (vergleiche Unterabschnitt 1.2.2 auf Seite 8) anhand von Beispielen definiert und beschrieben. Dies dient zum Erlangen eines tieferen Verständnisses gegenüber der Welt-Matrix **M**, als auch gegenüber der Transformation von (bald vierdimensionalen) Punkten in ein anderes Referenz-Koordinatensystem. Für die in **??** auf Seite ?? erklärten Methoden ist dieses Wissen jedoch nicht notwendig.

Um ein Objekt um einen Faktor $s_i, i \in [x, y, z]$ in i-Richtung zu skalieren (strecken oder stauchen), muss jeder Vertex des Objekts entlang der i-Achse damit multipliziert werden [13, 14, 15, 16]. In Abbildung 18a auf der nächsten Seite wird die Box entlang der x-Achse um den Faktor 5 gestreckt und entlang der y-Achse um den Faktor $\frac{1}{2}$ gestaucht. Entlang der z-Achse gibt es keine Veränderung; der Faktor ist folglich 1.
Die Skalierungsmatrix **S** zur Durchführung dieser Transformation lässt sich dementsprechend notieren als

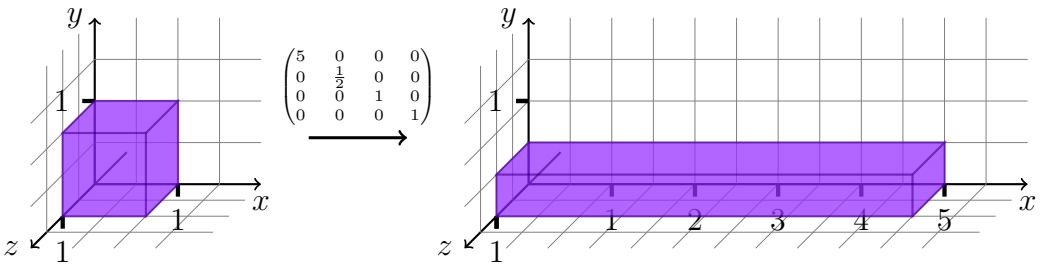
$$\mathbf{S}\left(5, \frac{1}{2}, 1\right) = \begin{pmatrix} 5 & 0 & 0 \\ 0 & \frac{1}{2} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Allgemein ist die Skalierungsmatrix definiert als

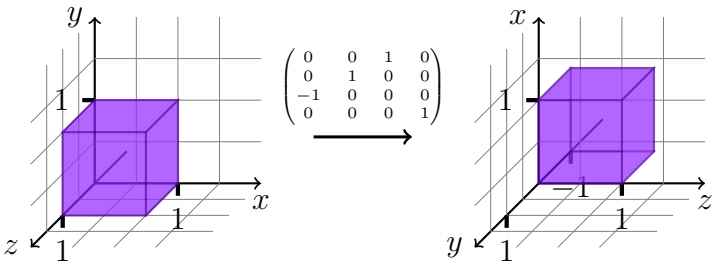
$$\mathbf{S}(s_x, s_y, s_z) = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{pmatrix} \tag{2}$$

Multipliziert man diese Matrix von links an jeden Vertex des Objekts, so wird das gesamte Objekt korrekt skaliert [15, 16].

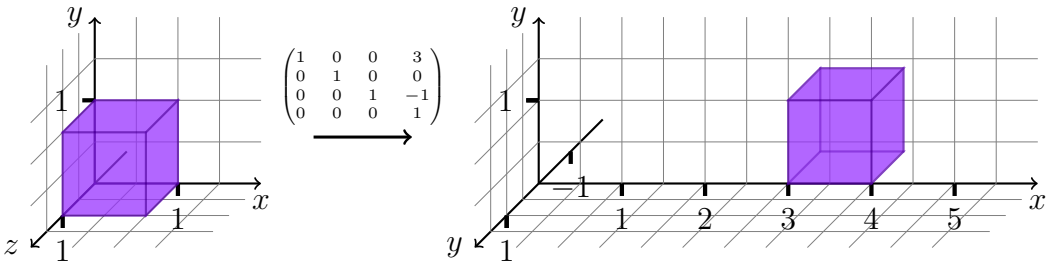
Abbildung 17: Auswirkung von Transformationen auf die lila Beispiel-Box.



(a) Auswirkung einer Skalierung.



(b) Auswirkung einer Rotation.



(c) Auswirkung einer Translation.

Um ein Objekt um einen Winkel $\theta_i, i \in [x, yz]$ um die i-Achse herum zu rotieren, muss jeder Vertex des Objektes entsprechend um die i-Achse herum rotiert werden [13, 15, 16]. In Abbildung 18b wird die Box ausschließlich um die y-Achse rotiert; θ_y ist hierbei $90°$ beziehungsweise $\frac{\pi}{2}$ rad. Für die x- und z-Achse ergeben sich $\theta_x = \theta_z = 0[°/\text{rad}]$. Somit lassen sich die Rotationsmatrizen notieren als

$$\mathbf{R}_x(0) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(0) & \sin(0) \\ 0 & -\sin(0) & \cos(0) \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \mathbb{I}_d$$

$$\mathbf{R}_y\left(\frac{\pi}{2}\right) = \begin{pmatrix} \cos(\frac{\pi}{2}) & 0 & \sin(\frac{\pi}{2}) \\ 0 & 1 & 0 \\ -\sin(\frac{\pi}{2}) & 0 & \cos(\frac{\pi}{2}) \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{pmatrix}$$

$$\mathbf{R}_z(0) = \begin{pmatrix} \cos(0) & -\sin(0) & 0 \\ \sin(0) & \cos(0) & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \mathbb{I}_d$$

wobei $\mathbb{I}_d$ die Identitätsmatrix ist. (Anhand dieser zeigt sich gut, dass in um die x- und z-Achse nicht rotiert wird.)
Allgemein sind die Rotationsmatrizen definiert als

$$\mathbf{R}_x(\theta_x) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta_x) & \sin(\theta_x) \\ 0 & -\sin(\theta_x) & \cos(\theta_x) \end{pmatrix} \tag{3}$$

$$\mathbf{R}_y(\theta_y) = \begin{pmatrix} \cos(\theta_y) & 0 & \sin(\theta_y) \\ 0 & 1 & 0 \\ -\sin(\theta_y) & 0 & \cos(\theta_y) \end{pmatrix} \tag{4}$$

$$\mathbf{R}_z(\theta_z) = \begin{pmatrix} \cos(\theta_z) & -\sin(\theta_z) & 0 \\ \sin(\theta_z) & \cos(\theta_z) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{5}$$

Multipliziert man diese Matrizen von links an jeden Vertex des Objekts, so wird das gesamte Objekt korrekt rotiert [16].

Um ein Objekt um $t_i, i \in [x, y, z]$ Einheiten in i-Richtung zu verschieben, muss die entsprechende Koordinate jedes Vertex des Objekts damit summiert werden [13, 15, 16]. In Abbildung 18c auf der vorherigen Seite wird die Box in entlang der x-Achse um 3 Einheiten verschoben und entlang der z-Achse um -1 Einheiten. Entlang der y-Achse wird sie nicht verschoben; der Summand ist 0.
Die neue Position eines Vertex $v = (x, y, z)^T$ ist folglich

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} 3 \\ 0 \\ -1 \end{pmatrix} = \begin{pmatrix} x+3 \\ y+0 \\ z-1 \end{pmatrix}$$

Für den Vertex $v_1 = (1, 0, 0)^T$ ergibt sich damit

$$v_1' = \begin{pmatrix} 4 \\ 0 \\ -1 \end{pmatrix}$$

Im Gegensatz zur Skalierung und Rotation existiert für die Translation keine (für jeden Vertex einheitliche) dreidimensionale Matrix, die an das Objekt multipliziert werden kann. Der Grund dafür ist, dass es sich bei der Translation nicht um eine *lineare Abbildung* handelt [13, 15].
Eine lineare Abbildung $\mathbf{L} : \mathbb{R}^m \to \mathbb{R}^n$ erhält Vektor-Addition und skalare Multiplikation

$$\mathbf{L}(\vec{a} + \vec{b}) = \mathbf{L}(\vec{a}) + \mathbf{A}(\vec{b})$$
$$\mathbf{L}(\alpha\vec{a}) = \alpha\mathbf{L}(\vec{a})$$

wobei $\vec{a} \in \mathbb{R}^m, \vec{b} \in \mathbb{R}^n$ [13, 15].
Es ist einfach zu sehen, dass für eine Translation $\mathbf{L}(\vec{a} + \vec{b}) = \mathbf{L}(\vec{a}) + \mathbf{A}(\vec{b})$ beispielsweise nicht gilt; auf der linken Seite wird der Vektor $\vec{t}$ nur einmal auf das Ergebnis addiert, auf der rechten Seite zweimal.

Zur Lösung jenes Problems, werden *affine Transformationen* genutzt; diese vereinen lineare Transformationen mit einer Translation [13, 15].
Hierbei werden jedoch keine 3×3 Matrizen, sondern 4×4 Matrizen verwendet. Die

dreidimensionalen Koordinaten der Vertices werden dadurch um eine Koordinate w erweitert: Aus den Koordinaten des Vertex $v = (x, y, z)^T$ werden die Koordinaten $v = (x, y, z, w)^T$ [13, 15, 16].
Bei dieser Erweiterung der kartesischen Koordinaten handelt es sich um *homogene Koordinaten*. Neben der Nutzung von affinen Transformationen wird ebenfalls die notationelle Unterscheidung zwischen Vektoren und Punkten möglich. Ein Vektor ist gegeben als $\vec{q} = (x, y, z, 0)^T$ ($w = 0$), ein Punkt ist gegeben als $q = (x, y, z, 1)^T$ ($w = 1$). Da es sich bei den Vertices um Punkte und nicht um Vektoren handelt, gilt folglich $v_i = (x_i, y_i, z_i, 1)^T$ für alle Vertices v_i [13, 15, 16].

Die Translation der Box in Abbildung 18c auf Seite 22 kann mithilfe der affinen Transformationen nun ebenfalls als Matrix dargestellt werden:

$$\mathbf{T}(3, 0, -1) = \begin{pmatrix} 1 & 0 & 0 & 3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Allgemein ist die Translationsmatrix definiert als

$$\mathbf{T}(t_x, t_y, t_z) = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{6}$$

Multipliziert man diese Matrix von links an jeden Vertex des Objekts, so wird das gesamte Objekt korrekt verschoben [13, 15]. Es ist zu beachten, dass die w-Koordinate der Vertices dabei nicht verändert wird; ein Vertex bleibt weiterhin ein Punkt. Zusätzlich ist durch die Identitätsmatrix $\mathbb{I}_d$ oben links zu erkennen, dass die Vertices in keiner anderen Weise transformiert werden [13, 15, 16].

Durch die Nutzung der homogenen Koordinaten müssen auch die Skalierungsmatrix (Gleichung (2) auf Seite 21) und die Rotationsmatrizen (Gleichungen (3) bis (5) auf Seite 23) entsprechend angepasst werden. Hierzu werden die Matrizen einfach auf vier Dimensionen vergrößert, wobei sich die vierte Dimension durch Anwendung der Matrix nicht verändern soll (ein Vertex bleibt weiterhin ein Punkt).
Die Skalierungsmatrix ist dementsprechend definiert als

$$\mathbf{S}(s_x, s_y, s_z) = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{7}$$

und die Rotationsmatrizen sind definiert als

$$\mathbf{R}_x(\theta_x) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta_x) & \sin(\theta_x) & 0 \\ 0 & -\sin(\theta_x) & \cos(\theta_x) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{8}$$

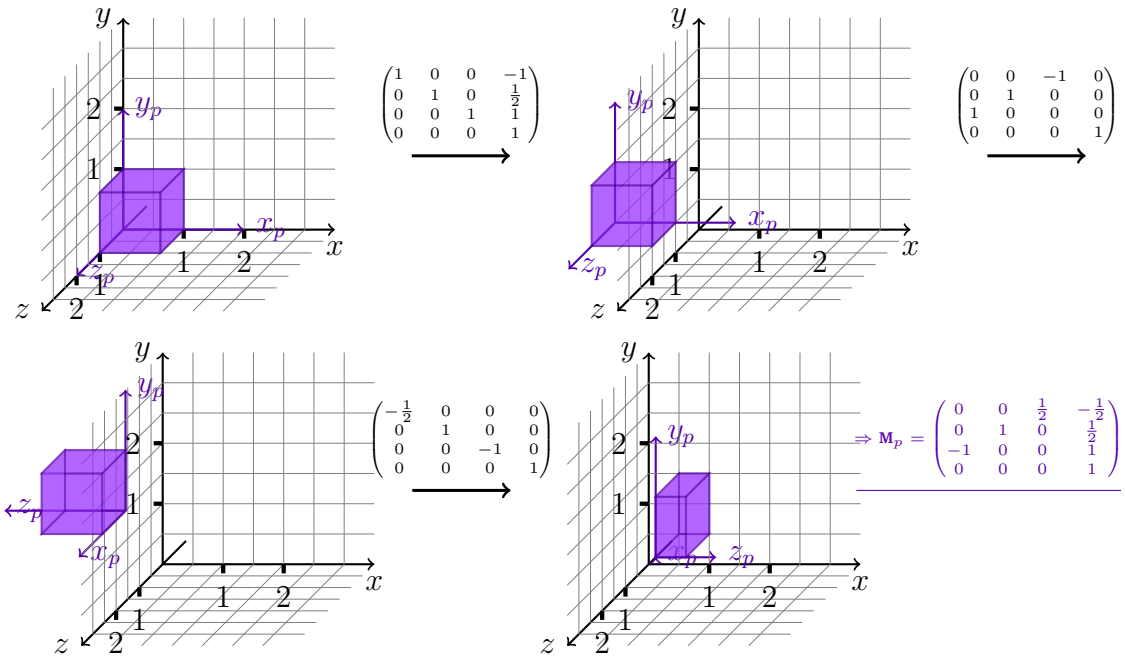
$$\mathbf{R}_y(\theta_y) = \begin{pmatrix} \cos(\theta_y) & 0 & \sin(\theta_y) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta_y) & 0 & \cos(\theta_y) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{9}$$

$$\mathbf{R}_z(\theta_z) = \begin{pmatrix} \cos(\theta_z) & -\sin(\theta_z) & 0 & 0 \\ \sin(\theta_z) & \cos(\theta_z) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{10}$$

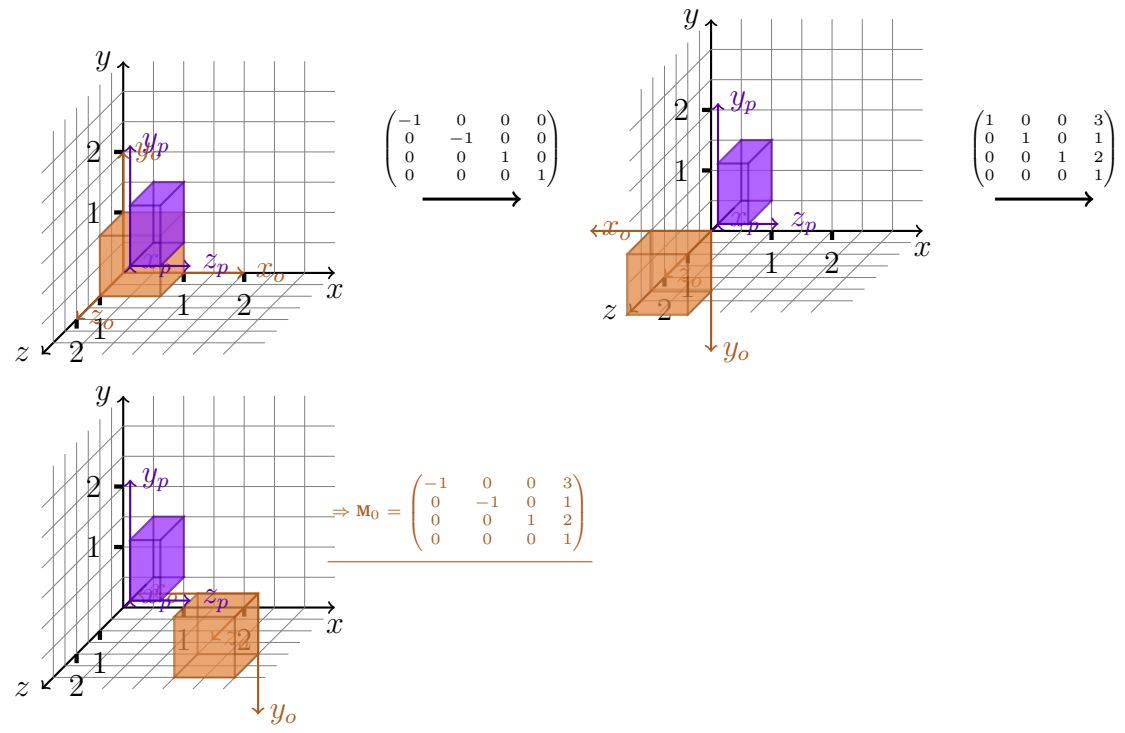
Durch Multiplikation dieser Matrizen von links an jeden (nun vierdimensionalen) Vertex des Objekts, wird das Objekt korrekt transformiert [13, 15, 16].

Abbildung 18 auf der nächsten Seite zeigt für die Beispiel-Boxen, wie sich aus den Transformationsmatrizen in Gleichungen (6) bis (10) die jeweiligen Weltmatrizen **M** zusammensetzen.

Abbildung 18: Zusammensetzung der unterschiedlichen Welt-Matrizen **M** zur Positionierung der Beispiel-Boxen im Welt-Koordinatensystem.



(a) Bildung der Weltmatrix $\mathbf{M}_p$ durch sukzessive Anwendung von Transformationsmatrizen.



(b) Bildung der Weltmatrix $\mathbf{M}_o$ durch sukzessive Anwendung von Transformationsmatrizen.

2.3 Vertex-Operationen – Kameramatrix

In Abbildung 19 auf der nächsten Seite müssen alle Objekte von ihrer Position in Abbildung 20a auf der nächsten Seite zu ihrer Position in Abbildung 20b auf der nächsten Seite transformiert werden.

Die Transformation der Kamera von Abbildung 20a auf der nächsten Seite nach Abbildung 20b auf der nächsten Seite ist tricky; von Abbildung 20b auf der nächsten Seite nach Abbildung 20a auf der nächsten Seite hingegen leicht.
Hierzu muss lediglich

- der Einheitsvektor in x-Richtung auf den $\vec{r}$-Vektor abgebildet werden,
- der Einheitsvektor in y-Richtung auf den $\vec{u}$-Vektor abgebildet werden,
- der Einheitsvektor in z-Richtung auf den negativen $\vec{v}$-Vektor abgebildet werden,
- und der Ursprung $(0, 0, 0, 1)^T$ muss auf die Position e der Kamera verschoben werden.

Dieser Transformation lässt sich direkt als 4×4 Matrix aufschreiben

$$\begin{pmatrix} r_x & u_x & -v_x & e_x \\ r_y & u_y & -v_y & e_y \\ r_z & u_z & -v_z & e_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

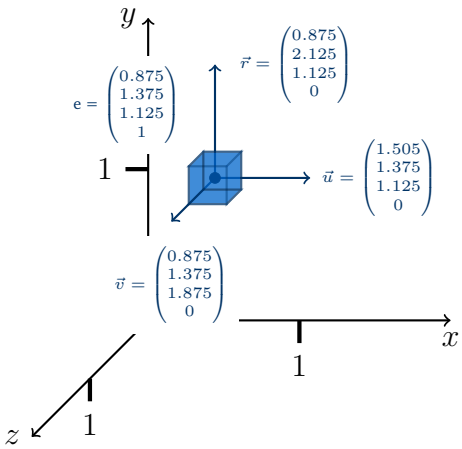
[15, 23].

Um nun die Kamera von Abbildung 20a auf der nächsten Seite nach Abbildung 20b auf der nächsten Seite zu transformieren, muss lediglich die Inverse der obigen Matrix verwendet werden. Diese ist die Kameramatrix **V**

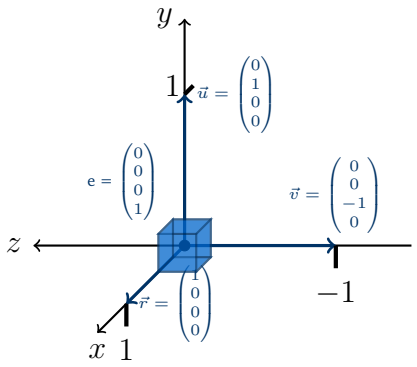
$$\mathbf{V} = \begin{pmatrix} r_x & u_x & -v_x & e_x \\ r_y & u_y & -v_y & e_y \\ r_z & u_z & -v_z & e_z \\ 0 & 0 & 0 & 1 \end{pmatrix}^{-1} = \begin{pmatrix} r_x & r_y & r_z & 0 \\ u_x & u_y & u_z & 0 \\ -v_x & -v_y & -v_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & -e_x \\ 0 & 1 & 0 & -e_y \\ 0 & 0 & 1 & -e_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{11}$$

[15, 23].
Multipliziert man nun alle Vertices der Szene (in Welt-Koordinaten) von links mit der Kameramatrix **V**, so erhält man die Szene in Kamera-Koordinaten.

Abbildung 19: Vergleich der Positionierung der Kamera im Welt-Koordinatensystem vor der Anwendung der Kameramatrix **V** (a) und nach der Anwendung der Kameramatrix **V** (b).



(a) Positionierung der Kamera im Welt-Koordinatensystem nach Anwendung der Weltmatrix **M**.



(b) Positionierung der Kamera im Welt-Koordinatensystem nach Anwendung der Kameramatrix **V**. Die Kamera liegt nun als Standard-Kamera vor, die Objekte befinden sich im Kamera-Koordinatensystem.

2.4 Vertex-Operationen – Orthographische Projektion

Um die in Abbildung 7a auf Seite 11 dargestellt orthographische Projektion durchzuführen, wird folgende Matrix verwendet

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{12}$$

Dies lässt die z-Koordinate fallen (beziehungsweise setzt diese auf 0), wodurch nur noch die zweidimensionale xy-Eben übrig bleibt [13, 15].
Es sei angemerkt, dass technisch gesehen die w-Koordinate der Vertices, welche zur Nutzung von affinen Transformationen in Unterabschnitt 1.2.2 auf Seite 9 einführt wurde, noch vorhanden ist. Des Weiteren ist der Wert von w für alle Vertices gleich (insbesondere verändert sich $w = 1$ bei der orthographischen Projektion nicht, die Vertices belieben weiterhin Punkte). Alle Vertices liegen folglich in derselben w-Dimension, diese kann bei der Visualisierung der Vertices/Objekte einfach "weggedacht" werden.

Obige Matrix (Gleichung (12)) projiziert nun alle Vertices der Szene orthographisch auf die xy-Ebene. Es ist jedoch sinnvoll, nur die Vertices zu projizieren, die von der Kamera gesehen werden. Dadurch muss lediglich ein kleiner Teil der Vertices gerendert werden, dies ist effizienter [13, 15].
Definiert ist der Sichtbereich der Kamera durch die intrinsischen Kamera-Parameter

- n (für *near plane*): Gibt an, wo der Sichtbereich der Kamera nach vorne hin beginnt. Alle Punkte davor werden nicht von der Kamera gesehen.
- f (für *far plane*): Gibt an, wo der Sichtbereich der Kamera nach hinten hin endet. Alle Punkte dahinter werden nicht von der Kamera gesehen.
- l (für *left plane*): Gibt an, wo der Sichtbereich der Kamera nach links hin beginnt. Alle Punkte links davon werden nicht von der Kamera gesehen.
- r (für *right plane*): Gibt an, wo der Sichtbereich der Kamera nach rechts hin endet. Alle Punkte rechts davon werden nicht von der Kamera gesehen.
- b (für *bottom plane*): Gibt an, wo der Sichtbereich der Kamera nach unten hin beginnt. Alle Punkte darunter werden nicht von der Kamera gesehen.
- t (für *top plane*): Gibt an, wo der Sichtbereich der Kamera nach oben hin endet. Alle Punkte darüber werden nicht von der Kamera gesehen.

Bei der orthographischen Projektion definieren diese Parameter eine Box (siehe Abbildung 8a auf Seite 13); bei der perspektivischen Projektion (vergleiche Unterabschnitt 1.2.3 auf Seite 11) definieren sie ein Frustum [13, 15, 16]. Da die Kamera die negative z-Achse entlang schaut, gilt: $n > f$ [13, 15].

Durch die orthographische Projektion wird diese Box auf den $[-1, 1] \times [-1, 1] \times [-1, 1]$ Einheitswürfel abgebildet, siehe Abbildung 8a auf Seite 13 [13, 15, 16, 21].
Durchgeführt wird die Projektion von der Projektionsmatrix $\mathbf{P}$, diese ist für alle Vertices gleich. Bei der orthographischen Projektion ist sie definiert als

$$\mathbf{P} = \begin{pmatrix} \frac{2}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2}{t-b} & 0 & 0 \\ 0 & 0 & -\frac{2}{f-n} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & -\frac{l+r}{2} \\ 0 & 1 & 0 & -\frac{b+t}{2} \\ 0 & 0 & 1 & -\frac{-n-f}{2} \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{l+r}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{b+t}{t-b} \\ 0 & 0 & -\frac{-2}{f-n} & -\frac{n+f}{f-n} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

wobei l, r, b, t, n, f sich auf die oben definierten intrinsischen Kamera-Parameter beziehen [13, 15, 16].
Die hierbei grün markierten Einträge verschiebend das Zentrum des Einheitswürfels auf das Zentrum des Koordinatensystems. Die blau markierten Einträgen skalieren die ursprüngliche Box auf die Größe des Einheitswürfels. Das "$-$" innerhalb von $-\frac{-2}{f-n}$ sorgt dafür, dass die z-Werte "umgedreht werden". Dies geschieht, da die Kamera die negative z-Achse entlang schaut; nach der Projektion sind die z-Werte intuitiver geordnet (je größer der Wert, desto weiter ist der zugehörige Vertex von der Kamera entfernt) [13, 15, 16].

Die w-Koordinaten der Vertices bleiben während der orthographischen Projektion unverändert; $w = 1$ für alle Vertices. Dementsprechend ist keine Homogenisierung der Koordinaten mehr notwendig und die Vertices liegen schon nach der Projektion in normalisierten Gerätekoordinaten vor (vergleiche im Gegensatz die perspektivische Projektion in Unterabschnitt 1.2.3 auf Seite 12). Folglich sind bei einer Anwendung der orthographischen Projektion die Clip-Koordinaten identisch zu den normalisierten Gerätekoordinaten.

2.5 Vertex Operationen – Herleitung Projektionsmatrix P für die perspektivische Projektion

Die zur perspektivischen Projektion verwendet Projektionsmatrix **P** (vergleiche Gleichung (1) auf Seite 12) ist definiert als

$$\mathbf{P} = \begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{l+r}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{b+t}{t-b} & 0 \\ 0 & 0 & -\frac{n+f}{f-n} & -\frac{2nf}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

wobei l, r, b, t, n, f sich auf die intrinsischen Parameter der Kamera (siehe Unterabschnitt 1.2.3 auf Seite 11) beziehen[13, 15, 16].

Zur Herleitung dieser Matrix ist es sinnvoll zuerst die in Abbildung 20 auf der nächsten Seite dargestellt Projektion zu betrachten. Hier wird ein dreidimensionaler Punkt $q = (x, y, z)^T$ auf eine zweidimensionale Bildebene bei $z = -d, d > 0$ abgebildet. Der resultierende Punkt hat die Koordinaten $q' = (x', y', -d)^T$. Das Projektionszentrum liegt bei $(0, 0, 0)^T$ (dort befindet sich auch die Standard-Kamera, vergleiche Unterabschnitt 1.2.2 auf Seite 10) [13, 15, 16].

Aus der Geometrie dieser Darstellung kann entnommen werden, dass das Verhältnis von x zu z gleich dem Verhältnis von x' zu $z' = -d$ ist

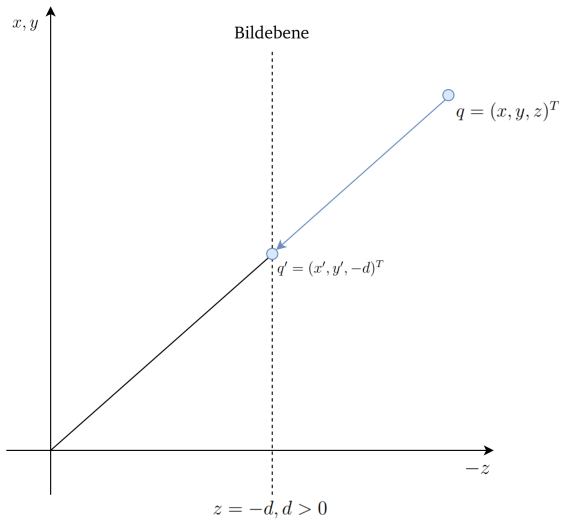
$$\frac{x}{z} = \frac{x'}{-d} \Leftrightarrow x' = \frac{-d}{z} \cdot x$$

Aus selbigem Grund für y folgt

$$\frac{y}{z} = \frac{y'}{-d} \Leftrightarrow y' = \frac{-d}{z} \cdot y$$

Die z-Koordinate von q wird offensichtlich einfach auf $-d$ abgebildet [13, 15, 16].

Abbildung 20: Darstellung der Rasterung am Beispiel eines lila Dreiecks. Bildquelle: Darstellung nach [15, 16].



Somit wird der Punkt q abgebildet, als

$$q = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \mapsto \begin{pmatrix} x \cdot \frac{-d}{z} \\ y \cdot \frac{-d}{z} \\ -d \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = q'$$

Um diese Abbildung als Matrix zu notieren, ist es notwendig die homogenen Koordinaten (vergleiche Unterabschnitt 1.2.2 auf Seite 9) zu verwenden. Ansonsten müsste aufgrund der Multiplikation mit $\frac{-d}{z}$ für jeden Punkt eine eigene Projektionsmatrix existieren; das ist nicht effizient [13, 15].
Mithilfe der homogenen Koordinaten kann der Wert $\frac{-z}{d}$ (Kehrwert von $\frac{-d}{z}$) in die w-Komponente geschrieben werden. Am Ende wird jede Koordinate eines Punktes dann durch seinen w-Wert geteilt (vergleiche Homogenisierung in Unterabschnitt 1.2.3 auf Seite 12) [13, 15].
Für obige Abbildung ergibt sich folglich

$$q = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \mapsto \begin{pmatrix} x \cdot \frac{-d}{z} \\ y \cdot \frac{-d}{z} \\ -d \end{pmatrix} \longleftrightarrow \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{-1}{d} & 0 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \\ \frac{-z}{d} \end{pmatrix} \overset{Homogenisierung}{\cong} \begin{pmatrix} x \cdot \frac{-d}{z} \\ y \cdot \frac{-d}{z} \\ -d \\ 1 \end{pmatrix}$$

[13, 15]

Mittels dieser Einsicht ist es nicht mehr schwer, die Projektionsmatrix **P** herzuleiten.
Die x-Koordinate eines jeden Punkts q, soll mithilfe von **P** auf die near plane (vergleiche Unterabschnitt 1.2.3 auf Seite 11) projiziert werden. Das Frustum (Sichtbereich der Kamera) soll dabei zum Einheitswürfel werden, dessen Zentrum im Ursprung des Koordinatensystems liegt [13, 15, 16].

Die x-,y-Koordinate von q wird somit abgebildet als

$$x \mapsto \left(x\frac{n}{-z} - \frac{l+r}{2} \right) \cdot \frac{2}{r-l}$$
$$y \mapsto \left(y\frac{n}{-z} - \frac{b+t}{2} \right) \cdot \frac{2}{t-b}$$

Dabei beziehen sich die orange markierten Teile auf die Projektion auf die near plane, die grün markierten Teile auf die Verschiebung der Mitte des Würfels auf den Koordinatensystem-Ursprung und die blau markierten Teile auf die Skalierung auf die Größe des Einheitswürfels [13, 15, 16].

Die Abbildung der z-Koordinate erfolgt aus historischen Gründen als

$$z \mapsto \frac{a \cdot z + b}{z}$$

Hierdurch wird eine höhere Genauigkeit der z-Werte von Punkten nahe an der Kamera erhalten (Punkte die weiter von der Kamera entfernt sind, haben eine geringere Genauigkeit der z-werte). Zusätzlich soll $-n \mapsto -1$ und $-f \mapsto 1$ gelten; je näher ein Punkt an der Kamera, desto kleiner der jeweilige z-Wert. Für a, b ergibt sich dadurch

$$a = -\frac{n+f}{f-n}$$
$$b = -\frac{2nf}{f-n}$$

[15, 16]

In Matrix-Notation ergeben diese drei Abbildungen der x, y, z-Koordinaten die Projektionsmatrix **P** [13, 15, 16].

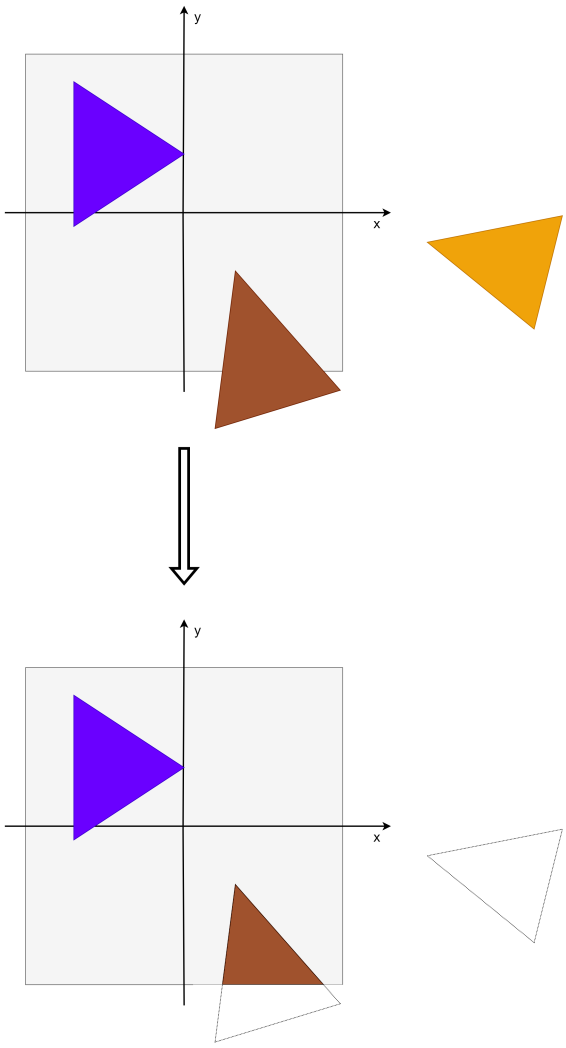
2.6 Vertex-Transformationen – Clipping

Abbildung 21 zeigt die drei Fälle, die beim Clipping auftauchen können:

- Ein Primitiv (Abbildung 21 orangenes Dreieck) befindet sich außerhalb des Einheitswürfels. Es kann vollständig ignoriert werden.
- Ein Primitiv (Abbildung 21 lila Dreieck) befindet sich komplett im Innern des Einheitswürfels. Es kann vollständig in den nächsten Bearbeitungschritt weitergegeben werden.
- Ein Primitiv (Abbildung 21 braunes Dreieck) befindet sich zum Teil außerhalb des Einheitswürfels und zum Teil im Inneren des Einheitswürfels. Das Primitiv wird entgegen dem Einheitswürfel abgeschnitten, der Teil innerhalb des Einheitswürfels wird weiterverarbeitet und der Teil außerhalb wird ignoriert.

[13, 14, 15].

Abbildung 21: Darstellung des Clipping im Zweidimensionalen. Bildquelle: Darstellung nach [28].



Literaturverzeichnis

[1] *Computer Sciences: Digital Images*. URL: https://www.encyclopedia.com/computing/news-wires-white-papers-and-books/digital-images (besucht am 23. 12. 2022).

[2] Franz Kummert. "Bildverarbeitung". Skript zur Vorlesung 'Bildverarbeitung' an der Universität Bielefeld im Wintersemester 2022/2023. Dezember 2022. URL: https://bis.uni-bielefeld.de/sites/50090/Documents/SkriptBildverarbeitung.pdf (besucht am 23. 12. 2022).

[3] Reinhard Klette. "Image Data". In: *Concise Computer Vision: An Introduction into Theory and Algorithms*. London: Springer London, 2014, S. 1–42. URL: https://doi.org/10.1007/978-1-4471-6320-6_1 (besucht am 02. 01. 2023).

[4] Michael E Becker. "Pixelsalat-von Bildschirmauflösungen, Zeichengrößen und Lesbarkeit". In: *Computer-Fachwissen. Bund Verlag* (2005).

[5] Ailsa Allaby und Michael Allaby. *A Dictionary of Earth Sciences: Pixel*. URL: https://www.encyclopedia.com/science/dictionaries-thesauruses-pictures-and-press-releases/pixel (besucht am 23. 12. 2022).

[6] *Merriam-Webster.com dictionary: Color Model*. URL: https://www.merriam-webster.com/dictionary/color%20model (besucht am 24. 12. 2022).

[7] Theresa-Marie Rhyne. "Applying Color Theory to Digital Media and Visualization". In: 1. Aufl. CRC Press, 2016. Kap. 1.

[8] Martin Loesdau, Sébastien Chabrier und Alban Gabillon. "Hue and Saturation in the RGB Color Space". In: *Image and Signal Processing*. Hrsg. von Abderrahim Elmoataz u. a. Cham: Springer International Publishing, 2014, S. 203–212. ISBN: 978-3-319-07998-1.

[9] Michael Rohs und Sven Kratz. *Computergrafik 2: Fourier-Transformation*. 2011. URL: https://www.medien.ifi.lmu.de/lehre/ss11/cg2/slides/CG2-S11-05-DFT.pdf (besucht am 03. 01. 2023).

[10] Anshul Sachdev. *Spatial and Frequency Domain — Image Processing*. 2019. URL: https://medium.com/vithelper/spatial-and-frequency-domain-image-processing-83ffa3fc7cbc (besucht am 03. 01. 2023).

[11] Oruc Ergueven und Torsten Heup. *Fouriertransformation*. 2005/2006. URL: http://www.gm.fh-koeln.de/~hk/lehre/ala/ws0506/Praktikum/Projekt/B_blau/Fouriertransformation.pdf (besucht am 03. 01. 2023).

[12] David A. Forsyth und Jean Ponce. "Computer Vision - A Modern Approach, Second Edition". In: Pearson, 2011. Kap. 4.3, S. 118–121. URL: https://eclass.teicrete.gr/modules/document/file.php/TM152/Books/Computer%20Vision%20-%20A%20Modern%20Approach%20-%20D.%20Forsyth,%20J.%20Ponce.pdf (besucht am 03. 01. 2023).

[13] Tomas Akenine-Möller, Eric Haines und Naty Hoffman. "Real-Time Rendering, Fourth Edition". In: 4th. A. K. Peters, Ltd., 2018. Kap. 1 – 4, S. 1–102. URL: https://research.quanfita.cn/files/Real-Time_Rendering_4th_Edition.pdf (besucht am 04. 01. 2023).

[14] Richard S. Wright u. a. "OpenGL SuperBible: Comprehensive Tutorial and Reference". In: 5th. Addison-Wesley Professional, 2010. Kap. 1, 4, S. 9–31, 125–177. URL: http://www.cosmic-rays.ru/books62/2010OpenGL_SuperBible_5th_Edition.pdf (besucht am 04. 01. 2023).

[15] Mario Botsch. “Graphische Datenverarbeitung (Master)”. Videos und Material zur Vorlesung ’Graphische Datenverarbeitung (Master)’ an der Technischen Universität Dortmund im Wintersemester 2020/2021. URL: https://graphics.cs.tu-dortmund.de/lehre/lehrveranstaltungen/ws-2020/2021/graphische-datenverarbeitung (besucht am 04. 01. 2023).

[16] Eric Lengyel. “Mathematics for 3D Game Programming and Computer Graphics, Third Edition”. In: 3rd. Boston, MA, USA: Course Technology Press, 2011. Kap. 1, 5, S. 1–9, 93–129. URL: https://canvas.projekti.info/ebooks/Mathematics%20for%203D%20Game%20Programming%20and%20Computer%20Graphics,%20Third%20Edition.pdf (besucht am 05. 01. 2023).

[17] Unity Technologies. *Unity Game Engine*. Version 2021.3 (LTS). URL: https://docs.unity3d.com/Manual/index.html (besucht am 04. 01. 2023).

[18] Unity Technologies. *Manual: Rendering paths in the Built-in Render Pipeline*. 2021. URL: https://docs.unity3d.com/Manual/RenderingPaths.html (besucht am 04. 01. 2023).

[19] Brian Caulfield. *What’s the Difference Between Ray Tracing and Rasterization?* 2018. URL: https://blogs.nvidia.com/blog/2018/03/19/whats-difference-between-ray-tracing-rasterization/ (besucht am 04. 01. 2023).

[20] pIXELsHAM. *What’s the Difference Between Ray Casting, Ray Tracing, Path Tracing and Rasterization? Physical light tracing...* 2019. URL: https://www.pixelsham.com/2019/10/24/whats-the-difference-between-ray-tracing-and-rasterization/ (besucht am 12. 01. 2023).

[21] Marco Alamia. *Article - World, View and Projection Transformation Matrices*. URL: http://www.codinglabs.net/article_world_view_projection_matrix.aspx (besucht am 05. 01. 2023).

[22] David A. Forsyth und Jean Ponce. “Computer Vision - A Modern Approach, Second Edition.” In: 2. Aufl. Pitman, 2012. Kap. 1, S. 3–31. URL: https://eclass.teicrete.gr/modules/document/file.php/TM152/Books/Computer%20Vision%20-%20A%20Modern%20Approach%20-%20D.%20Forsyth,%20J.%20Ponce.pdf (besucht am 06. 01. 2023).

[23] Christian Ermann. *Deriving the View Matrix*. URL: https://www.christianermann.dev/posts/view-matrix/ (besucht am 06. 01. 2023).

[24] *Don’t be conservative with Conservative Rasterization*. 2014. URL: https://developer.nvidia.com/content/dont-be-conservative-conservative-rasterization (besucht am 16. 01. 2023).

[25] Nick Evanson. *How 3D Game Rendering Works, A Deeper Dive: Rasterization and Ray Tracing*. 2019. URL: https://www.techspot.com/article/1888-how-to-3d-rendering-rasterization-ray-tracing/ (besucht am 16. 01. 2023).

[26] Maklaan. *Datei:RGB color cube.svg*. 2015. URL: https://de.wikipedia.org/wiki/Datei:RGB_color_cube.svg (besucht am 12. 01. 2023).

[27] David Slavsky. *GRAPHING FOURIER SERIES*. 2017. URL: http://dslavsk.sites.luc.edu/courses/other/classnotes/fouriergraphs.pdf (besucht am 16. 01. 2023).

[28] Vierge Marie. *File:Cube clipping.svg*. 2008. URL: https://commons.wikimedia.org/wiki/File:Cube_clipping.svg (besucht am 16. 01. 2023).