

Summary

Grundlagen Der Grafischen Datenverarbeitung - Meyer

Jonas Weßner

2. August 2021

Inhaltsverzeichnis

1 Graphische Programmierung	3
1.1 Definition von Graphischen Objekten	3
1.2 Koordinatensysteme	3
1.3 Graphische Primitive in OpenGL	3
1.4 Transformationen Graphischer Objekte	4
1.5 Szenegraph	4
1.6 Arbeiten mit OpenGL	5
2 Abbilden Graphischer Objekte	8
2.1 Kameramodell	8
2.2 Perspektivische Projektion	9
2.3 Projektionsverfahren in OpenGL	10
3 Mathematische Grundlagen	12
3.1 Skalierung	12
3.2 Rotation	12
3.3 Translation	13
4 Koordinatensysteme	13
4.1 Polarkoordinatensystem	14
4.1.1 Kreisabschnitt im Polarkoordinatensystem	14
4.2 Zylinderkoordinatensystem	15
4.3 Kreiskoordinatensystem	15
4.3.1 Äquatorebene im Kugelkoordinatensystem	16
4.3.2 Halbkreise im Kugelkoordinatensystem	16
5 Visualisierungstechniken	16
5.1 Lineare Interpolation	16
5.2 Bilineare Interpolation	17
5.3 Visualisierung unter Berücksichtigung von Licht	17
5.3.1 Arten von Lichtquellen	17
5.3.2 Phong'sches Beleuchtungsmodell	18
5.3.3 Beleuchtungsberechnung - Methoden	19
6 Digitale Bilder	20
6.1 Darstellung mittels Pixel	20
6.2 Optische Wahrnehmung	20
6.3 Farbmodelle	21
6.3.1 CIE Farbraum	21
6.3.2 Hardwareorientiertes Modell - RGB	21
6.3.3 Hardwareorientiertes Modell - CMY	22

6.3.4	Wahrnehmungorientiertes Modell - HSV	22
6.3.5	YUV Farbmodell und 4:1:1 Farbraum	23
7	Bildverarbeitung	24
7.1	Maße zur Beurteilung von Bildern	24
7.1.1	Histogramme	24
7.1.2	Entropie	24
7.2	Punktoperationen & lineare Transformationen	24
7.2.1	Addition	26
7.2.2	Binärisierung	26
7.2.3	Äquidensitenbilder	26
7.2.4	Kontrastveränderung	27
7.3	Lokale Bildoperationen - Faltungsoperatoren	27
7.3.1	Faltungsmatrizen /-operatoren und deren Berechnung	27
7.3.2	Identität	28
7.3.3	Glättungsfilter / Box-Filter / Mittelwertoperator	28
7.3.4	Gauß-Filter	29
7.3.5	Differenzoperator & Verständnis für die Ableitung	29
7.3.6	Prewitt-Operator	30
7.3.7	Sobel-Operator	30
7.3.8	Laplace-Operator	30
7.3.9	Kantenhervorhebung bzw. Schärfungsfilter	30
7.3.10	Relief-Filter	31
7.4	Lokale Bildoperatoren - Rangfolgeoperatoren	31
7.4.1	Rangfolgeoperatoren und deren Berechnung	31
7.4.2	Arten von Rangfolgeoperatoren	32
7.4.3	Anwendung und Kombination von Rangfolgeoperatoren	33
7.5	Algorithmen	34
7.5.1	Canny-Edge-Detector	34
7.5.2	Hough-Transformation	36
7.5.3	Harris-Corner-Edge-Detection	38
7.5.4	SIFT - Scale-invariant Feature Transform	38
7.5.5	Template Matching	39
7.5.6	Image Stitching	40

1 Graphische Programmierung

1.1 Definition von Graphischen Objekten

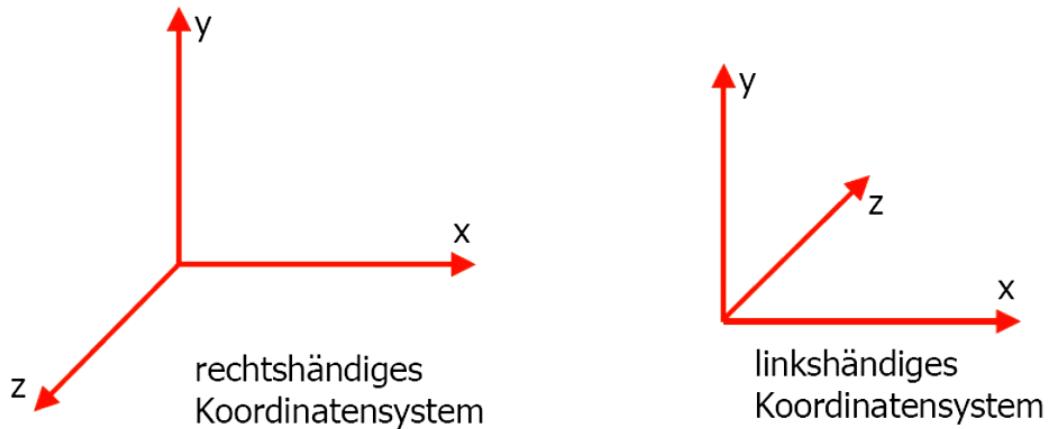
Graphische Objekte bestehen letztendlich aus Punkten, die mit Kanten zu Dreiecken verbunden sind. Man könnte auch denken, wieso denn nicht 4- oder 5-Ecke?

Jedes n-Eck, wobei $3 \leq n$, kann durch hinzufügen zusätzlicher Kanten nur mit Dreiecken dargestellt werden. Dreiecke haben zusätzlich den Vorteil, dass die eine Ebene im 3-dimensionalen kartesischen Koordinatensystem definieren. Dadurch kann das Dreieck als klarer Ebenenteil gezeichnet werden, wohingegen z.B. Vierecke nicht unbedingt in einer Ebene liegen und dadurch beim Zeichnen eventuell ohnehin als 2 getrennte Dreiecke gezeichnet werden müssten.

1.2 Koordinatensysteme

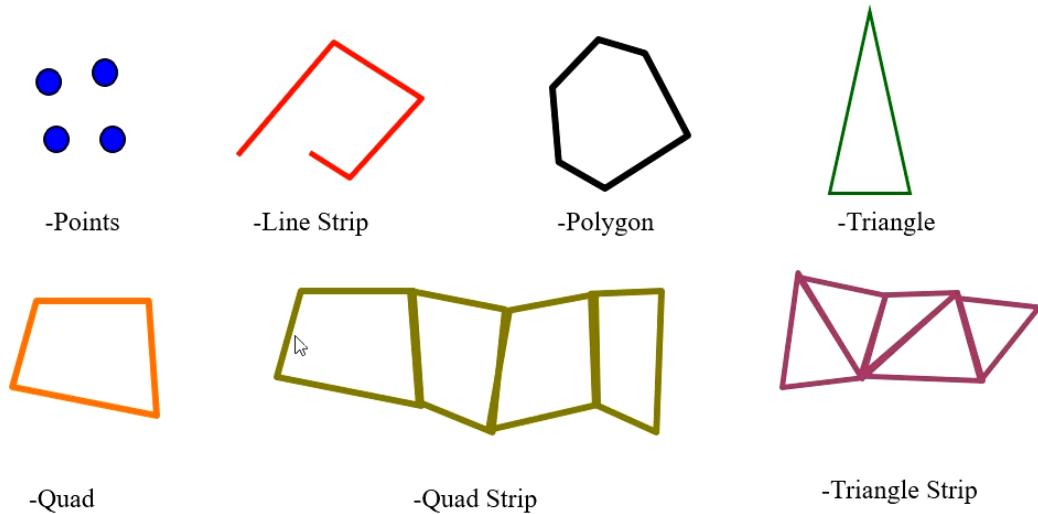
Wir betrachten das 3 dimensionale kartesische Koordinatensystem, welches als 3-dimensionales Orthogonalsystem definiert ist. Das bedeutet, dass es 3 Achsen gibt, die aus Vektoren bestehen, die paarweise Orthogonal zu sich stehen.

Wir unterscheiden weiter zwischen einem rechthändigen und linkshändigen Koordinatensystem, welche sich durch die Richtung der positiven z-Achse unterscheiden. Beim Rechtshändigen Koordinatensystem läuft die positive z-Achse zum betrachter hin, beim linkshändigen Koordinatensystem dementsprechend vom betrachter weg.



1.3 Graphische Primitive in OpenGL

In OpenGL können verschiedene *Primitive* erzeugt werden. Das sind einfache Objekte, die von OpenGL auf Basis einzelner Punkte erzeugt werden. Diese kann man dann zusammenschließen, um komplexere Objekte zu erzeugen. Beispiele für Primitive sind hier gezeigt:



1.4 Transformationen Graphischer Objekte

Jedes Objekt kann zunächst irgendwo in seinem Koordinatensystem definiert werden. Diese Platzierung nennt man nun die Platzierung in seinem *lokalen Koordinatensystem*.

Durch verschiedene Transformationen kann das Objekt verändert werden, um seine Position, Ausrichtung und Größe zu verändern und schließlich im *Weltkoordinatensystem* platziert werden.

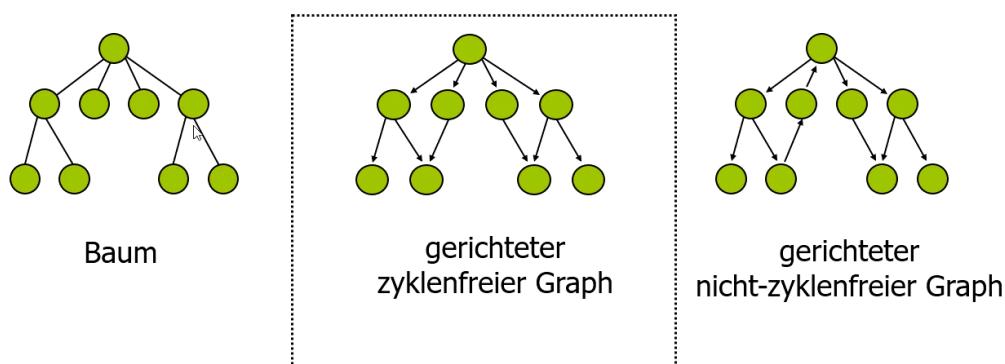
Somit können Objekte einmalig durch geeignete Funktionen definiert werden und dann durch Transformationen für ihre aktuelle Verwendung angepasst werden. Die Transformationen sind konkret:

- **Translation:** Verschiebung der Koordinaten eines Punktes. Dies entspricht der Addition eines Verschiebungsvektors auf den Ortsvektor des Punktes.
- **Rotation:** Drehen des Punktes um eine Koordinatenachse. Hierbei ist der Winkel entgegen des Uhrzeigersinns zu lesen.
- **Skalierung:** Multiplikation des Ortsvektors eines Punktes mit einem Skalar. Wird diese Operation für alle Punkte eines Objektes vorgenommen entsteht eine Stauchung oder Streckung entlang der Koordinatenachsen.

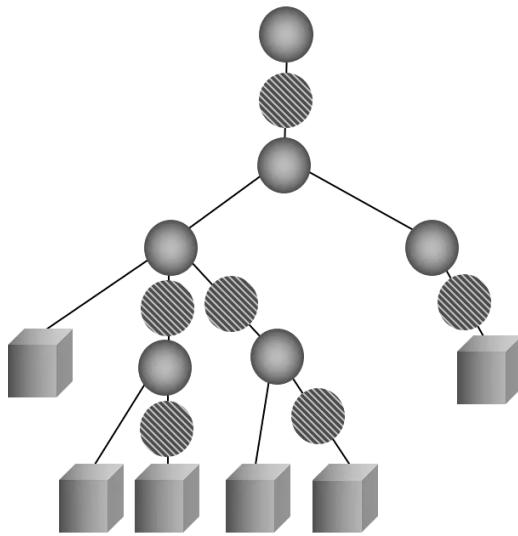
1.5 Szenegraph

Ein Szenegraph ist eine Datenstruktur, die eingesetzt werden kann, um mehrere graphische Objekte zu einer Szene zusammenzuschließen.

Der Szenegraph ist ein gerichteter Zyklenfreier Graph. Das bedeutet, dass im Gegensatz zu einem Baum, ein Kindknoten auch mehrere Elternknoten haben kann, jedoch die Richtung der Abhängigkeiten einheitlich von Wurzel zu Blättern verläuft:



Die Knoten des Szenegraphen können Entweder Gruppen, Geometrien oder Transformationen sein:



Gruppen bezeichnen hierbei zusammenfassungen von Punkten, die Graphische Objekte bilden. Transformationen beziehen sich dann auf alle Objekte, die im Szenegraphen unter ihnen liegen.

Dadurch kann man die Zusammenhänge komplexerer Objekte geeignet darstellen. Zum Beispiel soll eine Komplexe Figur, z.B. ein Mensch, sich generell im Raum bewegen können (Translation). Diese Translation wirkt sich dann auf alle Gruppen, die den Mensch bilden aus (als z.B. seinen Kopf, Rumpf, Arme und Beine). Der Mensch kann aber auch den Arm heben. Diese Translation soll sich dann also nur auf den Arm der Figur auswirken. Läuft der Mensch aber und hebt dabei seine Hand, so muss auf dem Arm offensichtlich sowohl die Translation des Körpers als auch die des Arms angewendet werden. So sieht man, dass der Arm im Szenegraphen also eine Gruppe sein muss, deren Eltengraphen die gesamte Figur Mensch ist.

1.6 Arbeiten mit OpenGL

OpenGL ist im Grunde eine Art Zustandsautomat und sehr explizit. Zunächst sind keine Einstellungen vorgenommen. Wird durch einen Funktionsaufruf eine Einstellung (z.B. der Farbe) vorgenommen, so wird sie so lange beibehalten, bis sie geändert wird.

Ein weiteres Beispiel ist, dass der Bildschirm jedes Frame explizit gecleared werden muss, falls das vorherige Bild verworfen werden soll. Denn das vorherige Bild würde bestehen bleiben und nur die Stellen an denen Objekte gerendert werden aktualisiert werden.

Wie berechnet OpenGL Transformationen?

Transformationen von Vektoren können mathematisch generell durchgeführt werden, indem eine Transformationsmatrix definiert wird, die mit dem Vektor multipliziert wird und so einen neuen Vektor ergibt. Sollen mehrere Transformationen auf einen Punkt angewendet werden, so muss ein Vektor also nacheinander mit mehreren Matrizen multipliziert werden. Dabei ist zu beachten, dass die Multiplikation von Matrizen nicht kommutativ aber assoziativ ist.

$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \cdot \left(\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \cdot \begin{pmatrix} a \\ d \\ g \end{pmatrix} \right) = \left(\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \cdot \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \right) \cdot \begin{pmatrix} a \\ d \\ g \end{pmatrix}$$

OpenGL geht so vor, dass es immer, wenn eine Transformation definiert wird, diese mit der aktuellen globalen Tranformationsmatrix multipliziert (Zu Beginn ist diese Matrix die Dreidimensionale Einheitsmatrix). Immer wenn dann ein Objekt definiert wird, wird diese aktuelle Transformationsmatrix auf das Objekt angewendet.

Weil die Multiplikation assoziativ ist, entspricht das Ergebnis der Multiplikation in jedem Fall dem Ergebnis, das entsteht, wenn man den Vektor von rechts nach links mit den Matrizen multipliziert. Will man also eine bestimmte Transformation vor einer anderen durchführen, so muss sie weiter rechts stehen, um früher angewendet zu werden.

Matrizenstack

Neben der aktuellen Transformationsmatrix, die sich immer auf die aktuell definierten Objekte bezieht, speichert OpenGL auch noch einen sogenannten Matrizenstack.

Ein Aufruf von `glPushMatrix()` legt die aktuelle Transformationsmatrix auf dem Stack ab. Die Matrix wird aber auch als aktuelle Matrix weiter verwendet.

Ein Aufruf von `glPopMatrix()` ersetzt die aktuelle Transformationsmatrix durch die zuletzt gepushte Matrix und entfernt sie vom Matrizenstack.

Somit kann also eine Matrix gesichert und später wiederverwendet werden, wie etwa bei Unterprogrammaufrufen in ASM, wenn man zunächst die aktuellen Register auf dem Stack rettet.

Zum Beispiel könnte man sich vorstellen, dass ein um 45 Grad gedrehter Würfel erzeugt werden soll, der außerdem um 2 Längen entlang der negativen y-Achse verschoben ist und ein Würfel im Ursprung des Koordinatensystems gezeichnet werden soll. Wenn man nicht den Matrizenstack nutzt würde das in etwa so aussehen:

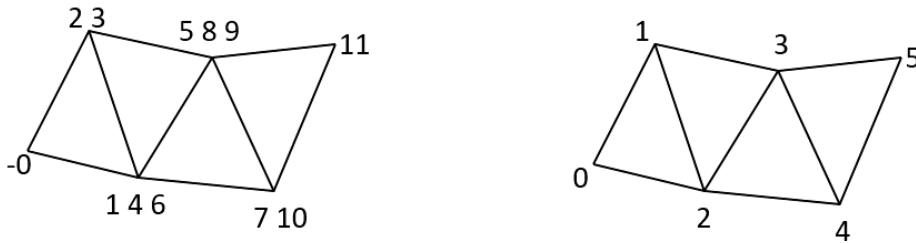
- `glRotatef(45, 1.0, 0.0, 0.0);`
- `glTranslatef(0.0, -2.0, 0.0);`
- `Cube();`
- `glRotatef(-45, 1.0, 0.0, 0.0);`
- `glTranslatef(0.0, 2.0, 0.0);`
- `Cube();`

Mit Benutzung des Matrizenstacks ist das Wiederherstellend er ursprünglichen Transformationsmatrix einfacher:

- `glPushMatrix();`
- `glRotatef(45, 1.0, 0.0, 0.0);`
- `glTranslatef(0.0, -2.0, 0.0);`
- `Cube();`
- `glPopMatrix();`
- `Cube();`

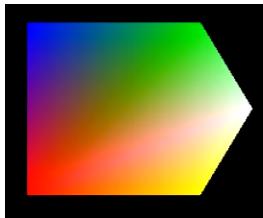
Definition von Dreiecksstreifen

Wenn komplexe Objekte aus Dreiecken gebildet werden sollen, dann sind viele Eckpunkte eines vorherigen Dreiecks wieder ein Eckpunkt eines folgenden Dreiecks und damit die definition und Speicherung dieses redundant. Somit kann durch das Benutzen von `GL_TRIANGLE_STRIPs` effizienter vorgegangen werden.



Definition von Farben und Texturen

Farben und Texturen werden jeweils global umgestellt und gelten dann für alle folgenden Punkte (siehe Zustandsautomat). Flächen oder Linien, die letztendlich durch die Verbindung von Punkten entstehen erhalten dann die lineare Interpolation der Farb und Texturwerte der zugehörigen Punkte:



Rotation von Objekten um sich selbst

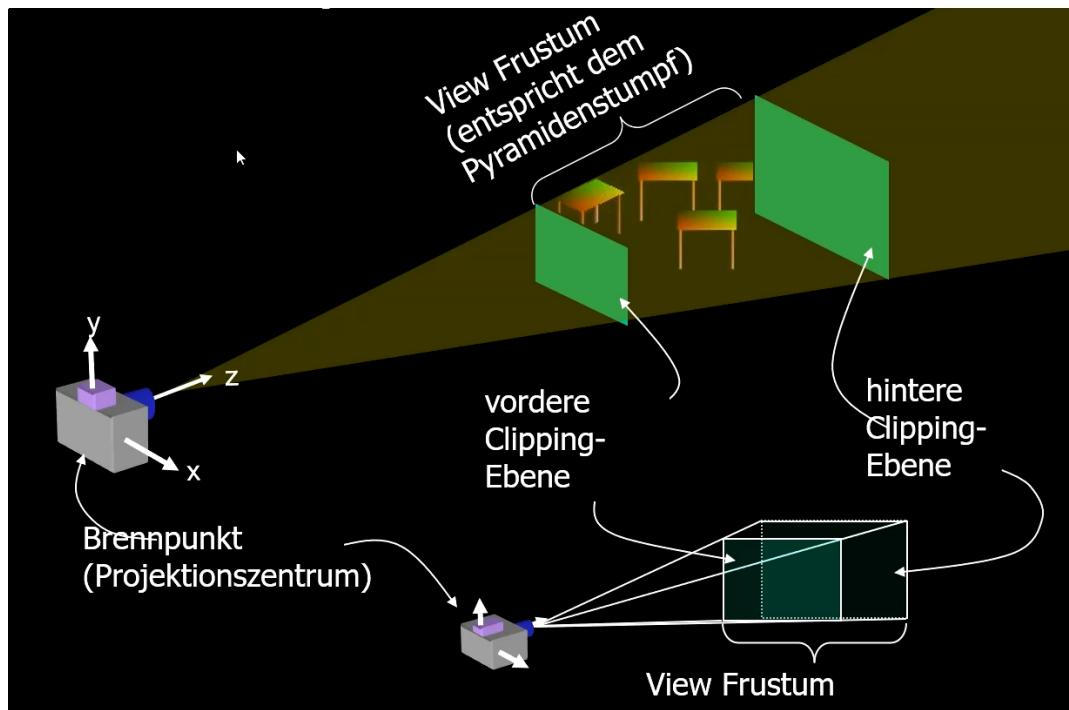
Soll ein Objekt um seinen eigenen Mittelpunkt gedreht werden, so kann dieses Objekt zunächst in den Ursprung des Koordinatensystems bewegt werden, dann die Rotation durchgeführt werden und dann muss es wieder an seinen ursprünglichen Platz zurückbewegt werden.

Auf diese Weise kann der Standort des Objektes vor der Rotation erhalten bleiben.

Hierbei ist auch zu beachten, dass die Rücktranslation zuerst angegeben werden muss, dann die Rotation und dann die Translation, da wie gesagt die zuletzt angegebene Transformation zuerst angewendet wird.

2 Abbilden Graphischer Objekte

2.1 Kameramodell



Im vorherigen Kapitel haben wir gelernt, wie man graphische Objekte im "Weltkoordinatensystem" also im globalen Koordinatensystem erzeugen kann. Die Abbildung dieser 3-dimensionalen Welt auf dem 2-dimensionalen Bildschirm hängt jetzt von der Position des Betrachters, sowie einigen weiteren Parametern ab.

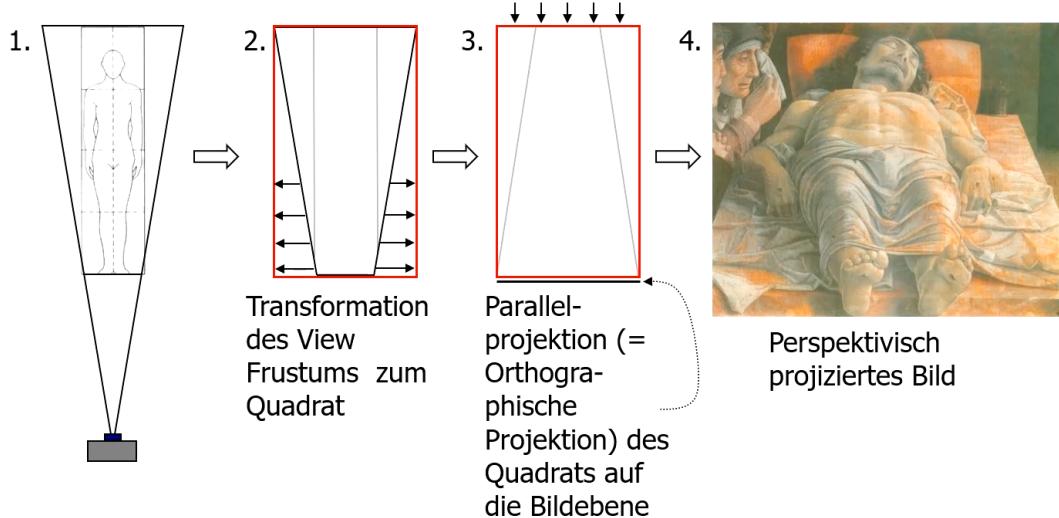
Dazu kann man sich zu visualisierung das folgende Kameramodell vorstellen:

Die Kamera steht an einer beliebigen Position im Weltkoordinatensystem, die durch einen Ortsvektor bestimmt wird. Sie ist in eine bestimmte Richtung gerichtet, die durch einen Punkt, auf den sie gerichtet ist, definiert ist. Dieser Punkt ist wiederum durch seinen eigenen Ortsvektor im Weltkoordinatensystem fest definiert. Wenn sich die Kamera bewegt, dann ändert sich ihre Blickrichtung so, dass sie weiterhin auf den Blickpunkt zeigt.

Die Kamera hat einen Öffnungswinkel, der den Winkel bezeichnet, den das gelbe Dreieck an der Ecke hat, die sich an der Kamera befindet. Wenn der Öffnungswinkel größer ist, ist also das Sichtfeld breiter. Die vordere und hintere Clipping-Ebenen begrenzen das Feld, das gerendert wird nach vorne und hinten (von der Kamera aus gesehen). Die Clipping Ebenen sind relativ zur Kamera positioniert, bewegen sich also mit, wenn sich die Kamera bewegt.

Somit wird letztendlich der gelbe Pyramidenstumpf, den man auch View Frustum, zwischen den beiden grünen Ebenen gerendert. Nun bleibt noch die Frage, wie man es schaffen kann, dass die 3-dimensionale Welt auf einen 2-Dimensionalen Bildschirm abgebildet wird und dabei trotzdem realistisch aussieht. Das wollen wir als Nächstes betrachten.

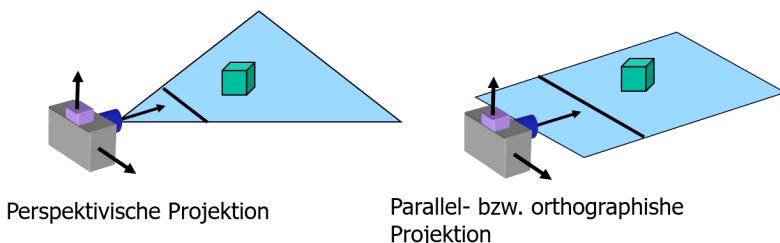
2.2 Perspektivische Projektion



Betrachtet man Bilder von Kameras (oder das eigene Sichtbild, das auf die Netzhaut des Auges projiziert wird), so stellt man schnell fest, dass Dinge, die Nähe am Betrachter sind, größer sind, als Dinge, die weiter entfernt sind. Dieses Verhalten können wir auch bei der Projektion in der Computergraphik herstellen, um realistische Bilder zu erzeugen.

1. Zunächst wird mithilfe des Kameramodells aus dem vorherigen Abschnitt der zu rendernde Bereich bestimmt. In der Realität wäre das der Bereich, dessen Lichtstrahlen durch ihre Position bedingt durch die Pupille auf die Netzhaut gelangen können (nur in der Realität ist der Bereich nicht nach vorne und hinten begrenzt). Das Bett, auf dem der Mensch auf dem Bild liegt bedeckt also den kompletten vorderen Rand des View Frustums und hinten ist neben dem Bett auch noch ein Stück der Umgebung im Bild.
2. Nun wird der Pyramidenstumpf mittels einer Transformation so verzerrt, dass sich ein Quader (nicht Quadrat wie auf dem Bild steht!) bildet. Dadurch werden Objekte, die näher sind breiter. Auf diese Weise kann man sich auch leicht vorstellen, dass, wenn man den Öffnungswinkel der Kamera vergrößert, dieser Effekt stärker wird, d.h. weiter entfernte Objekte wirken dann noch kleiner und nähere noch größer. Auf unserem Bild wird das Bett an seiner vorderen Kante gestreckt, sodass seine eigentlich parallelen Seitenkanten auf dem Bild nach hinten spitz zusammenlaufen.
3. Nun wird eine Parallelprojektion/orthographische Projektion durchgeführt, bei der der Quader auf ein Rechteck (das eigentliche Bild) abgebildet wird. Dabei ist es intuitiv notwendig, dass weiter vorne liegende Objekte weiter hinten liegende Objekte überlagern müssen.

Projektionsmöglichkeiten



OpenGL bietet neben der perspektivischen Projektion auch noch die Möglichkeit Parallelprojektion/orthographische Projektion zu benutzen. Diese Projektionsmethoden haben wir bereits kennengelernt, als wir die perspektivische Projektion betrachtet haben, da sie immer notwendig ist, um den 3-

dimensionalen Raum auf ein 2-dimensionales Bild zu projizieren. Benutzt man nur Parallelprojektion (d.h. nicht bloß im Rahmen der "Toolchain"), dann ist eigentlich die Kamera, die auf dem Bild eingezeichnet ist, überflüssig. Alle Objekte werden dann direkt auf die schwarze Linie, die das Bild darstellen soll, abgebildet. In der Anwendung bedeutet das, dass die Perspektive alleine durch die Position und Größe der Clipping-Ebenen definiert ist und jedes Objekt unabhängig von seiner Position im Bild genau die Größe hat, die es in der Realität hat (aber man kann das Bild danach natürlich noch skalieren). Das Größenverhältnis von Objekten bleibt auf dem Bild so, wie es in der Realität ist. Dieses Verhalten ist unnatürlich. Dabei entstehen Bilder, die ähnlich sind wie das folgende:



2.3 Projektionsverfahren in OpenGL

Die Projektionsmöglichkeiten werden in OpenGL auch über die Multiplikation der aktuellen Matrix mit Transformationsmatrizen geregelt. Dafür gibt es bestimmte Funktionen:

Orthographische Projektion

```

1 void glOrtho(GLdouble left,
2             GLdouble right,
3             GLdouble bottom,
4             GLdouble top,
5             GLdouble near,
6             GLdouble far);

```

Der Funktionsaufruf erstellt eine Matrix für eine orthographische Projektion und multipliziert sie mit der aktuellen Matrix, indem er statt einem View Frustum einen View Quader definiert. Der ganze Inhalt des Quaders wird also auf die near Clipping Plane projiziert, welche dann auf das Ausgabefenster gemappt wird.

- **left, bottom** geben den linken unteren Eckpunkt und **right, top** den rechten oberen Eckpunkt für die Clipping Planes an. Mittels der Information über die 2 Diagonalen Punkte und der Information, dass ein Rechteck entstehen soll, ist die Größe des Rechtecks definiert
- **near** und **far** beschreiben die Entfernung zum Betrachter mit denen die zwei identisch großen Clipping Planes platziert werden sollen

- Die Clipping Planes werden so ausgerichtet, dass der Betrachter orthogonal auf sie schaut.

Perspektivische Projektion

```

1 void gluPerspective(GLdouble fovy,
2                     GLdouble aspect,
3                     GLdouble zNear,
4                     GLdouble zFar);

```

Diese Funktion definiert ein View Frustum im Weltkoordinatensystem, generiert daraus eine Transformationsmatrix und multipliziert sie mit der aktuellen Matrix.

- **fovy** gibt den Öffnungswinkel in y-Richtung an.
- **aspect** gibt das Seitenverhältnis $x \div y$ an.
- **zNear** bezeichnet die Distanz des Betrachters zur near-clipping-plane
- **zFar** bezeichnet die Distanz des Betrachters zur far-clipping-plane

Die konkreten Seitenlängen lassen sich für jeden Teil des Frustums also durch den Öffnungswinkel, die Entfernung, das Seitenverhältnis und den aktuellen Standpunkt des Beobachters (Standardmäßig im Ursprung, ansonsten mit *gluLookAt* veränderbar) bestimmen. Somit ist das Frustum vollständig definiert.

Bewegung des Betrachters

```

1 void gluLookAt(GLdouble eyeX,
2                 GLdouble eyeY,
3                 GLdouble eyeZ,
4                 GLdouble centerX,
5                 GLdouble centerY,
6                 GLdouble centerZ,
7                 GLdouble upX,
8                 GLdouble upY,
9                 GLdouble upZ);

```

Diese Funktion ermöglicht es den Betrachter zu verschieben und seine Ausrichtung zu verändern. Wenn der Betrachter verschoben wird, **bewegen sich die Clipping-Planes mit ihm**.

- **eyeX, eyeY, eyeZ** beschreiben den Ortsvektor der Linse des Betrachters
- **centerX, centerY, centerZ** beschreiben den Ortsvektor des Punktes auf die der Betrachter schaut
- **upX, upY, upZ** beschreibt den Ortsvektor des Punktes, der vom Betrachter aus als oben betrachtet wird

3 Mathematische Grundlagen

Die Transformationen, die wir genauer betrachten wollen sind die Translation (Verschiebung), die Transformation (Größenänderung) und die Rotation (Drehung). Diese Transformationen nennt man auch affine Abbildungen. Das bedeutet, dass:

1. Die Bildpunkte von Punkten, die auf einer Gerade liegen, liegen wieder auf einer Geraden.
2. Die Bilder von parallelen Geraden sind wieder Parallel.
3. Die Größenverhältnisse der Bildpunkte stimmen mit denen der ursprünglichen Punkte überein.

Wir wollen uns nun ansehen, wie diese Transformationen mathematisch umgesetzt werden.

3.1 Skalierung

Die Skalierung eines Vektors \vec{x} wird beschrieben durch die Multiplikation mit einem Skalar y :

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \cdot y$$

Auf diese Weise wird jede Komponente von \vec{x} mit dem gleichen Wert y multipliziert. Um mehr Flexibilität zu erlangen und Objekte auch in verschiedene Achsenrichtungen unterschiedlich skalieren zu können, können wir die Skalierung auch als Matrix Y definieren. Diese Methode wird auch von OpenGL verwendet. Wenn wir eine Transformation definieren wird die aktuelle Matrix A mit einer Transformationsmatrix X multipliziert. Diese werden dann mit den Ortsvektoren \vec{x}_i aller Punkte der Objekte multipliziert.

$$A \cdot \begin{pmatrix} y_1 & 0 & 0 \\ 0 & y_2 & 0 \\ 0 & 0 & 0_3 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

3.2 Rotation

Die Rotation/Drehung wird ebenfalls durch die Multiplikation mit einer Matrix beschrieben. Die Komponente der Vektoren, mit denen die Matrix multipliziert wird, um die sich gedreht wird, darf nicht verändert werden. muss der Zeilenvektor, an dieser Stelle die Komponenten haben, die die Einheitsmatrix an dieser Stelle hätte.

Rotation um α gegen den Uhrzeigersinn um die x Achse:

$$A(\alpha) \cdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos \alpha & -\sin \alpha \\ 0 & \sin \alpha & \cos \alpha \end{pmatrix}$$

Rotation um α gegen den Uhrzeigersinn um die y-Achse:

$$A(\alpha) \cdot \begin{pmatrix} \cos \alpha & 0 & -\sin \alpha \\ 0 & 1 & 0 \\ \sin \alpha & 0 & \cos \alpha \end{pmatrix}$$

Rotation um α gegen den Uhrzeigersinn um die z-Achse:

$$A(\alpha) \cdot \begin{pmatrix} \cos \alpha & -\sin \alpha & 0 \\ \sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

3.3 Translation

Die Translation ist auf den ersten Blick eine Vektoraddition eines Ortsvektors \vec{v} mit einem Translationsvektor \vec{t} :

$$\vec{x} + \vec{t} = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} + \begin{pmatrix} t_1 \\ t_2 \\ t_3 \end{pmatrix} = \begin{pmatrix} v_1 + t_1 \\ v_2 + t_2 \\ v_3 + t_3 \end{pmatrix} \quad (1)$$

Während diese Herangehensweise für das Rechnen mit der Hand gut funktioniert, haben wir bei der maschinellen Berechnung ein Design-Problem. Denn die anderen beiden Transformationen, Skalierung und Rotation, können wir als Multiplikationen mit einer Matrix darstellen. OpenGL kann also alle Transformationsmatrizen multiplizieren und somit eine akkumulierte Matrix berechnen, die dann auf alle Punkte der Objekte angewandt wird. Wenn wir die Translation als Addition mit einem Translationsvektor \vec{y} beschreiben, müsste OpenGL zusätzlich zur akkumulierten Matrix auch noch einen akkumulierten Translationsvektor speichern, auf den alle Transformationen aufaddiert werden. Damit OpenGL trotzdem bloß mit Matrixmultiplikation rechnen kann, werden Homogene Koordinaten verwendet.

Homogene Koordinaten beschreiben das Erweitern der Vektoren und Matrizen eines bestimmten Vektorraums (hier der 3-dimensionale Vektorraum), um die Koordinaten einer weiteren Dimension. Dabei erweitern wir alle Vektoren \vec{v} um eine Koordinate $v_4 = 1$. Alle Matrizen M werden mit den Koordinaten der 4-dimensionalen Einheitsmatrix erweitert. Dabei können aber Koordinaten M_{14}, M_{24}, M_{34} durch die Komponenten des Translationsvektors \vec{t} aus (1) ersetzt werden.

$$\vec{v} = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} \rightarrow \vec{v}' = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ 1 \end{pmatrix} \quad (2)$$

$$M = \begin{pmatrix} M_{11} & M_{12} & M_{13} \\ M_{21} & M_{22} & M_{23} \\ M_{31} & M_{32} & M_{33} \end{pmatrix} \rightarrow M' = \begin{pmatrix} M_{11} & M_{12} & M_{13} & t_1 \\ M_{21} & M_{22} & M_{23} & t_2 \\ M_{31} & M_{32} & M_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

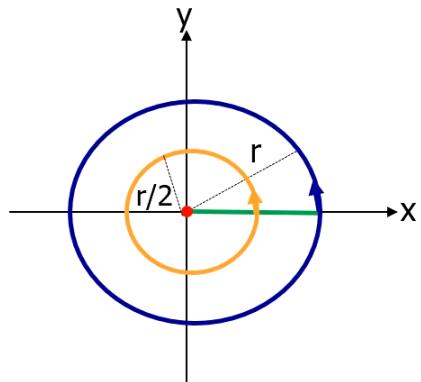
Mithilfe dieser Erweiterung können wir nun auch die Translation als Matrixmultiplikation darstellen. Denn bei der Multiplikation der Matrix M' mit dem Vektor \vec{v}' entsteht ein Vektor $\vec{v}'_{M'}$, der identisch zu dem Vektor \vec{v}_{Mt} ist, der entsteht, wenn man zunächst die Multiplikation mit M und dann die Addition mit \vec{t} durchgeführt hätte und ihn um eine homogene Koordinate erweitert.

$$M \cdot \vec{v} + \vec{t} \rightarrow M' \cdot \vec{v}' = \begin{pmatrix} M_{11} & M_{12} & M_{13} & t_1 \\ M_{21} & M_{22} & M_{23} & t_2 \\ M_{31} & M_{32} & M_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ 1 \end{pmatrix} = \begin{pmatrix} v_1 \cdot M_{11} + v_2 \cdot M_{12} + v_3 \cdot M_{13} + t_1 \\ v_1 \cdot M_{21} + v_2 \cdot M_{22} + v_3 \cdot M_{23} + t_2 \\ v_1 \cdot M_{31} + v_2 \cdot M_{32} + v_3 \cdot M_{33} + t_3 \\ 1 \end{pmatrix} \quad (3)$$

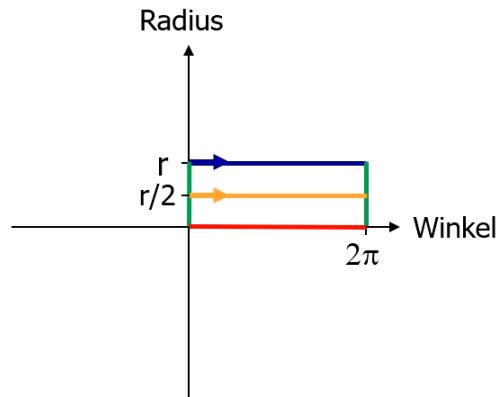
4 Koordinatensysteme

In diesem Kapitel werden wir sehen, wie wir in speziellen Koordinatensystemen bestimmte geometrische Objekte einfacher darstellen können. Der Grund ist, dass wir in der Computergrafik besser mit geraden Strecken rechnen können als mit kontinuierlichen Bögen. Die im Folgenden gezeigten Koordinatensysteme ermöglichen die Darstellung von rundungsintensiven Geometrien als Strecken oder n-Ecke. Dazu wird die Bedeutung der Achsen des kartesischen Koordinatensystems verändert.

4.1 Polarkoordinatensystem



Kartesisches Koordinatensystem



Polarkoordinatensystem

Das Polarkoordinatensystem ist ein 2-dimensionales Koordinatensystem. Bei der Übertragung eines Punktes P aus dem kartesischen Koordinatensystems ins Polarkoordinatensystem wird die Entfernung des Punktes P zum Ursprung auf die eine Achse übertragen (Radius r). Auf die andere Achse wird der Winkel (α) übertragen, den die positive x-Achse mit der Strecke zwischen dem Punkt P und dem Ursprung gegen den Uhrzeigersinn bildet. Dementsprechend sind die Umrechnungsregeln wie folgt definiert:

Von kartesischem Koordinatensystem zu Polarkoordinatensystem:

$$r = \sqrt{x^2 + y^2}$$

$$\alpha = \arctan(y/x)$$

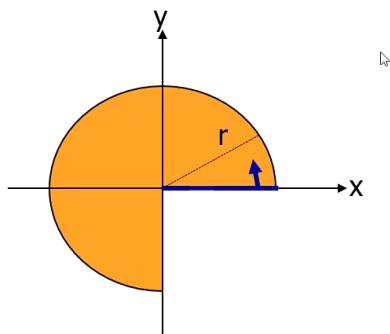
Von Polarkoordinatensystem zu kartesischem Koordinatensystem, mittels der Gesetze für trigon. Funktionen entsprechend umgeformt, sodass die rechte Seite nur die Achsen des Polarkoordinaten-system enthält:

$$x = \cos \alpha \cdot r$$

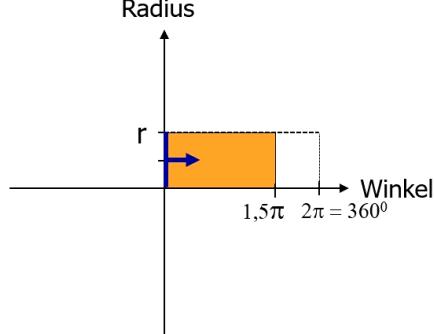
$$y = \sin(\alpha) \cdot r$$

Mittels dieser Definition können wir nun viele geometrische Formen, die normalerweise rundungen haben, als eckige Formen darstellen. Es folgt noch ein Beispiel.

4.1.1 Kreisausschnitt im Polarkoordinatensystem

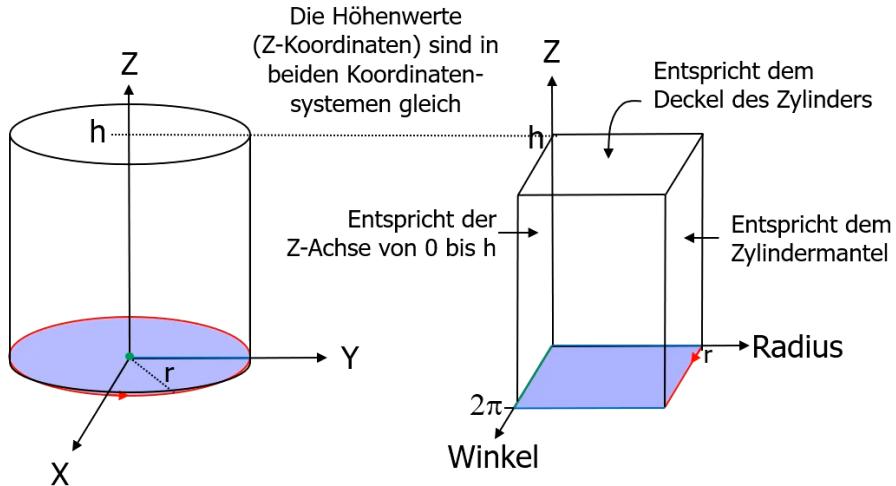


Kartesisches Koordinatensystem



Polarkoordinatensystem

4.2 Zylinderkoordinatensystem



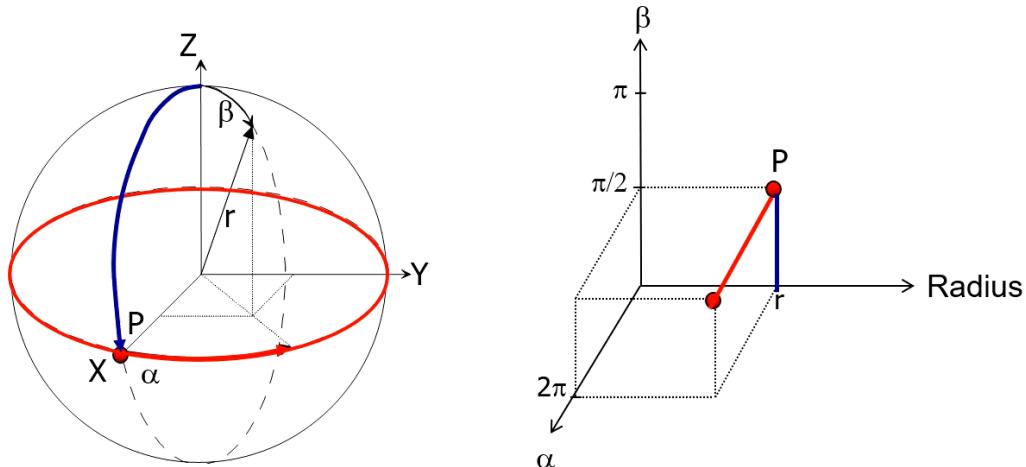
Kartesisches
Koordinatensystem

Zylinderkoordinatensystem

Das Zylinderkoordinatensystem ist ein 3-dimensionales Koordinatensystem, welches das Polarkoordinatensystem durch die z -Achse erweitert. Dabei ist die z -Achse unverändert so, wie sie auch im kartesischen Koordinatensystem ist. Somit ist die Umrechnung zwischen 3-dimensionalem kartesischen Koordinatensystem und Zylinderkoordinatensystem für x - und y -Koordinaten genau wie beim Umrechnen zwischen dem 2-dimensionalen kartesischen Koordinatensystem und dem Polarkoordinatensystem. Die z -Koordinate kann direkt übernommen werden.

Hier

4.3 Kreiskoordinatensystem



Kartesisches Koordinatensystem

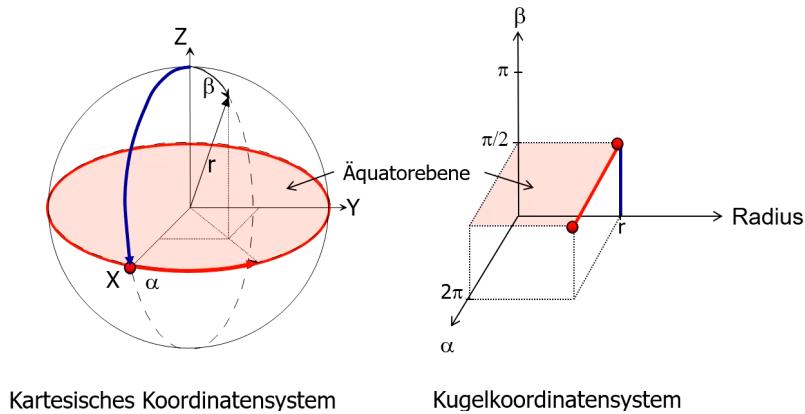
Kugelkoordinatensystem

Im Kreiskoordinatensystem wird die Position eines Punktes auf einer Kugeloberfläche mittels zweier Winkel α und β beschrieben. Durch den Radius, der die Größe der Kugel bestimmt, kann dann jeder Punkt P im Raum beschrieben werden.

Einer der beiden Winkel (hier α) muss $360 = 2\pi$ abdecken. Der andere Winkel (hier β) muss nur $180 = \pi$ abdecken. Denn, wenn wir die Linie, die von β bestimmt wird (hier in blau) um den halben Kreis, also bis $\beta = 180$ ziehen, können wir durch die Rotation um $\alpha = 360$ (hier in rot) bereits die gesamte Oberfläche der Kugel abdecken.

Hier noch 2 Beispiele:

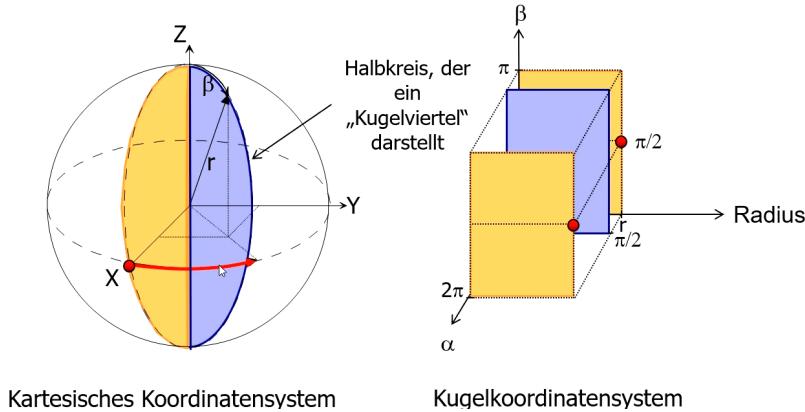
4.3.1 Äquatorebene im Kugelkoordinatensystem



Kartesisches Koordinatensystem

Kugelkoordinatensystem

4.3.2 Halbkreise im Kugelkoordinatensystem



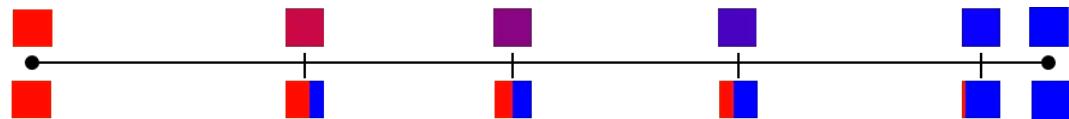
Kartesisches Koordinatensystem

Kugelkoordinatensystem

5 Visualisierungstechniken

Bei den Visualisierungstechniken beschäftigen wir uns vor Allem damit, wie man die Färbung Pixeln berechnen kann.

5.1 Lineare Interpolation



Die lineare Interpolation beschreibt die Berechnung der Farbe eines Punktes P, der auf einer Linie liegt. Die Linie befindet sich zwischen 2 Punkten A und B für welche die Farbe vorgegeben/bekannt ist. Jedem Punkt auf der Linie wird nun ein Anteil der Farben der Punkte A und B entsprechend seiner relativen Nähe zum jeweiligen Punkt zugeteilt.

Die entsprechende Rechnung ist die folgende:

t beschreibt das Verhältnis der Länge des Vektors \vec{AT} zur Länge des Gesamtvektors \vec{AB} :

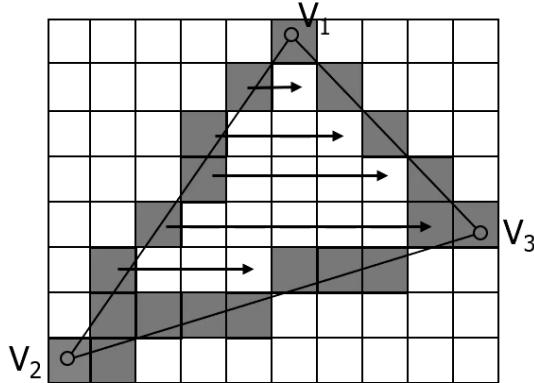
$$t = \frac{|\vec{AT}|}{|\vec{AB}|}$$

$F(X)$ beschreibt die Farbe eines Punktes X.

Somit gilt nun für die Farbe des Punktes P :

$$F(P) = (1 - t) \cdot F(A) + t \cdot F(B)$$

5.2 Bilineare Interpolation



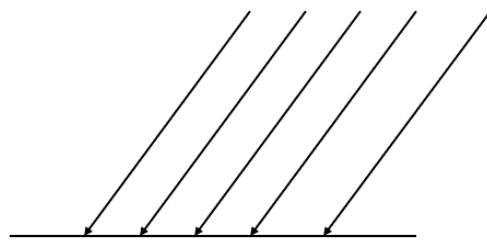
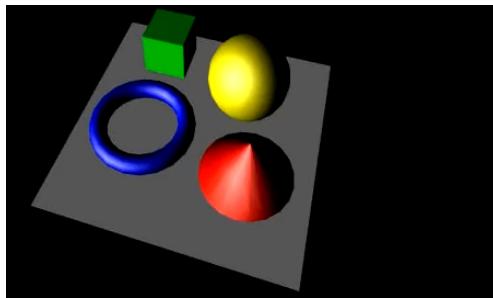
Die bilineare Interpolation ist ein Verfahren zur Berechnung von Farbwerten auf Flächen, die durch Eckpunkte mit vorgegebenen Farbwerten definiert werden. Dabei werden zunächst Farbwerte der Kanten durch lineare Interpolation der Eckpunkte berechnet (graue Quadrate). Danach können dann horizontal (prinzipiell könnte man das natürlich auch vertikal machen) die Farbwerte der Pixel jeder Reihe durch lineare Interpolation aus den Kanten berechnet werden (schwarze Pfeile).

5.3 Visualisierung unter Berücksichtigung von Licht

5.3.1 Arten von Lichtquellen

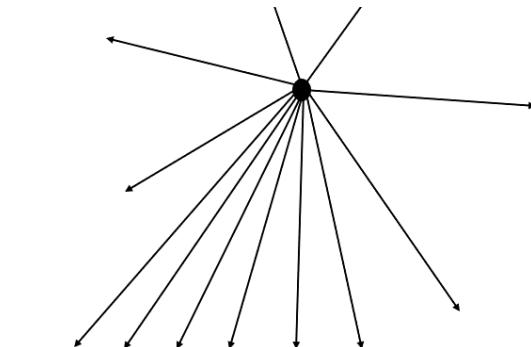
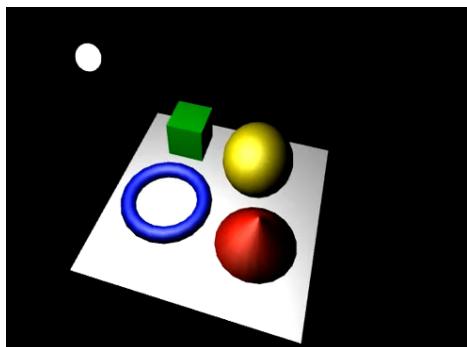
Gerichtete Lichtquelle:

Die Strahlen treffen parallel ein. Entsteht in der Realität annäherungsweise, wenn die Lichtquelle nahezu unendlich weit entfernt ist. Bsp.: Sonne

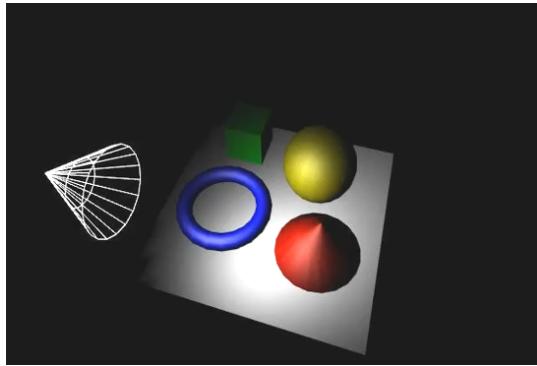


Punktlichtquelle:

Ein Lichtquelle im lokalen Umfeld, die Strahlen in alle Richtungen abgibt. Auf diese Weise treffen Lichtstrahlen an unterschiedlichen Positionen auf einer Ebene mit unterschiedlichen Winkeln auf.



Spotlight-Lichtquelle:

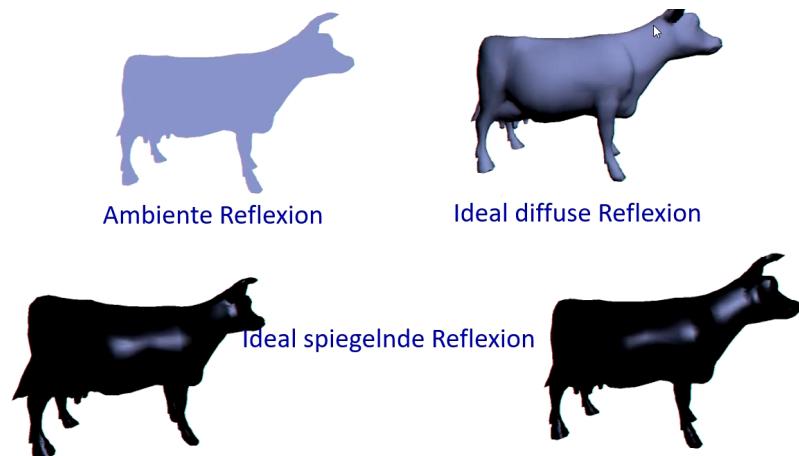


Die Strahlen kommen bezüglich des Winkels wie bei einer Punktlichtquelle an. Allerdings gibt die Lichtquelle nur in einem bestimmten Radius Licht ab. Die Lichtquelle hat also einen bestimmten Öffnungswinkel. Außerdem wird das Licht schwächer, wenn sich die Lichtquelle weiter entfernt.

5.3.2 Phong'sches Beleuchtungsmodell

Das Phong'sche Beleuchtungsmodell ist ein Modell, das zur Berechnung der Beleuchtung von Objekten benutzt werden kann. Dabei wird zunächst angenommen, dass die Lichtquelle punktförmig ist. Dann werden 3 Beleuchtungsarten benutzt, deren Summe dann ein halbwegs realistisches Bild ergeben sollen:

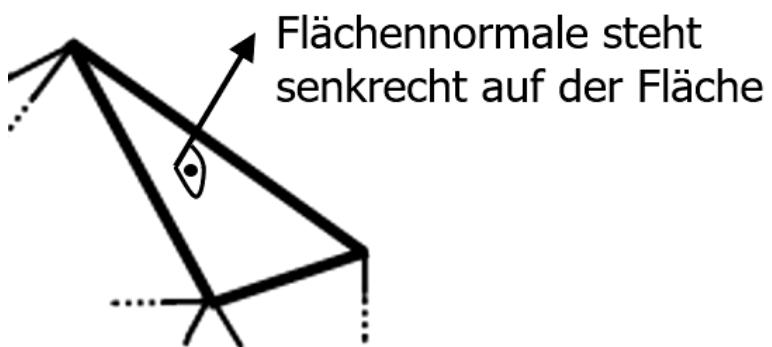
- Die **Spiegelnde Reflexion** bezeichnet die Reflexion, bei welcher der Einfallsinkel auf die Oberfläche, sowie die Position des Betrachters entscheidend sind. Hierbei entstehen bei richtigen glänzenden Flächen, andernfalls bleibt die Oberfläche schwarz, da kein Licht zum Betrachter gelangt. Diese Art von Reflexion kennen wir von Spiegeln oder hochglanzpolierten Oberflächen. Das reflektierte Licht hat die hier die Farbe der der Lichtquelle.
- Die **Diffuse Reflexion** bezeichnet die Art von Reflexion, die man von Matten Oberflächen kennt. Dabei wird das Licht in alle Richtungen gleichermaßen reflektiert. Die diffuse Reflexion ist somit unabhängig vom Standpunkt des Betrachters. Sie ist aber abhängig vom Einfallsinkel des Lichts. Dieser bestimmt nämlich die Helligkeit der Reflexion. Das reflektierte Licht hat die hier die Farbe der Oberfläche.
- Die **Ambiente Reflexion** bezeichnet die Art von Reflexion, die durch eine Grundbeleuchtung der Umgebung entsteht. Die Grundbeleuchtung ist für die gesamte Szene gleich und von der Lichtquelle und dem Betrachter unabhängig. Diese Reflexion soll modellieren, dass durch wiederholte Reflexion von Licht an der Umgebung schließlich die gesamte Umgebung eine gewisse Helligkeit hat. Dies kennt man, wenn es in einem dunklen Raum nie ganz dunkel wird, da immer noch Licht von Lichtquellen in den Raum gelangen kann, die nicht in der Szene bekannt sind. Das reflektierte Licht hat die hier die Farbe der Oberfläche.



5.3.3 Beleuchtungsberechnung - Methoden

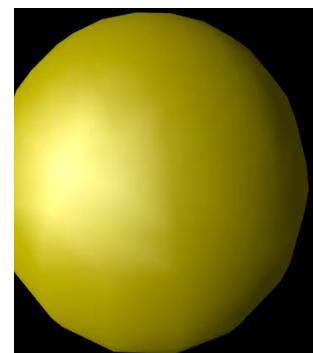
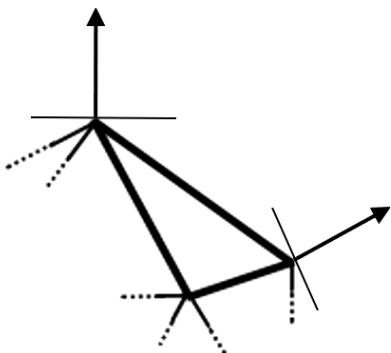
Bei der Berechnung der Beleuchtung nach dem Phong'schen sind die Winkel nötig, in denen das Licht auf die Oberfläche trifft (benötigt für spiegelnde und diffuse Reflexion). Um den Winkel berechnen zu können ist ein Normalenvektor für den Punkt/die Fläche nötig.

Flat Shading - Mittels Flächennormalen:



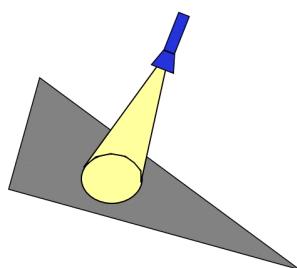
Für jede Fläche wird ein Normalenvektor bestimmt. Somit wird auch für jede Fläche genau ein Einfallsinkel und somit ein Farbwert bestimmt. Um mit dieser Berechnungsmethode ein realistisches Bild zu erschaffen benötigt man sehr viele einzelne Flächen.

Gouraud Shading - Mittels Eckennormalen



Es wird ein Normalenvektor für jeden Eckpunkt der Oberfläche bestimmt. die Farbwerte der Flächen berechnen sich dann durch bilineare Interpolation der Eckpunkte. Mit dieser Methode kann recht einfach und performant ein Bild sehr realistisch Aussehen. Das Objekt selbst sieht glatt aus, Obwohl die Oberfläche aus wenigen Teilflächen besteht (Einsparung von Speicherplatz gegenüber der Benutzung von Flächennormalen mit hoher Anzahl an Flächen).

Die **Grenze des Gouraud Shadings** ist, dass es bei sehr großen Flächen und sehr nahen Lichtquellen oder Spotlight-Lichtquellen dazu führen kann, dass die realen Farbwerte in der Mitte so verschieden zu den Farbwerten der Eckpunkte sind, dass die Fläche eine falsche Farbe erhält. Im unteren Beispiel würde so die Fläche nicht beleuchtet gerendered werden, weil ihre Eckpunkte nicht beleuchtet werden.



Phong Shading - Berechnung jedes Pixels

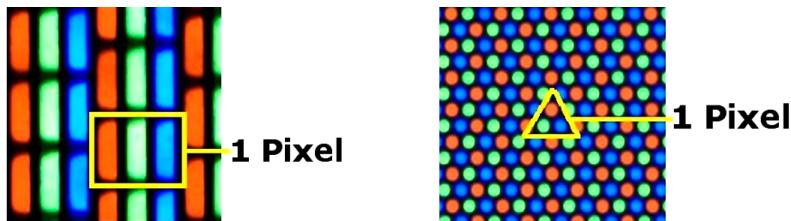
Beim Phong Shading wird für jeden Darzustellenden Pixel ein eigener Normalenvektor berechnet. Damit kann dann der Farbwert für diesen Pixel genau bestimmt werden. Mit diesem Verfahren können uns nun keine Dinge mehr entgehen, so wie es beim Gouraud Shading der Fall war. Allerdings ist die Berechnungszeit dementsprechend auch länger. Die Leistung der heutigen CPUs und GPUs hat aber dazu geführt, dass dieser Nachteil nicht mehr relevant ist.

6 Digitale Bilder

6.1 Darstellung mittels Pixel

Rastergraphiken, die wir nun betrachten wollen, werden als Matrix einzelner Farbpunkte dargestellt, die man Pixel nennt. Dadurch, dass jedes Pixel einen gewissen Farbwert hat, entsteht dann ein komplexes Bild. Pixel können auf 2 verschiedene Arten betrachtet werden:

Geometrische Betrachtung



Ein Pixel ist i.d.R kein atomarer Baustein eines Hardwaregeräts. Stattdessen besteht er aus i.d.R. 3 Lichtern, welche die 3 Grundfarben ausstrahlen. Wie das so ist werden wir noch in einem der folgenden Kapitel erfahren, wenn es um Farbmodelle geht.

Bei der geometrischen Betrachtung interessieren wir uns für die Form und Anordnung der Bestandteile eines einzelnen Pixels, sowie der Pixel zueinander.

Photometrische Betrachtung

Die Photometrische Betrachtung bezieht sich auf die Darstellung von Objekten mit Farben/Grautönen. Es geht also darum welchen Wert die Pixel annehmen sollen. Dies ist die Betrachtung, die uns im Folgenden beschäftigen wird.

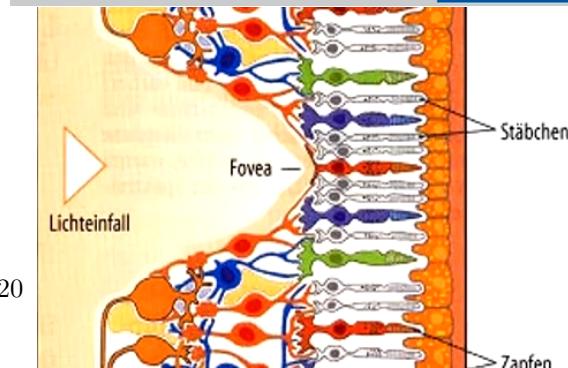
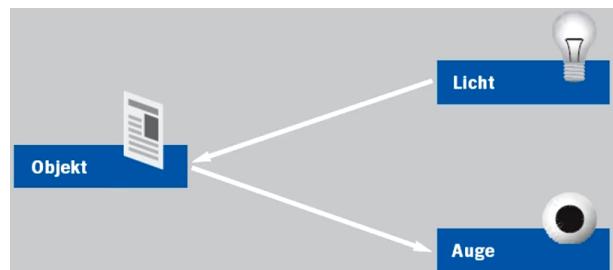
6.2 Optische Wahrnehmung

Physikalische und Biologische Aspekte

Objekte werden sichtbar, indem Licht auf sie fällt, reflektiert wird und in unser Auge gelangt. Je nach Oberfläche und Material des Objektes werden unterschiedliche Wellenlängen und damit Farben des Lichtes mehr oder weniger reflektiert. Die nicht reflektierten Lichtstrahlen werden absorbiert und somit in Wärmeenergie umgewandelt.

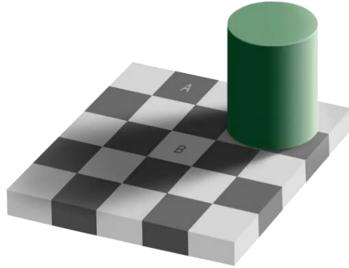
Auf der Retina (syn. Netzhaut) befinden sich Sinneszellen, die Stäbchen und Zapfen. Die Stäbchen sind für das Helligkeitssehen verantwortlich. Die Zapfen gibt es in den 3 Ausprägungen rot, grün und blau. Die Zapfen reagieren hauptsächlich auf die Wellenlänge von licht ihres Typs.

Die Empfindlichkeit für Helligkeit ist wesentlich



Stärker als die für Farben. Somit können bei wenig Licht fast nur noch die Stäbchen aktiviert werden und wir sehen Nachts bloß schwarz-weiß. Das wird nocheinmal interessant, wenn es um das HSV-Farbmodell geht.

Subjektivität der optischen Wahrnehmung



Die optische Wahrnehmung ist stark Subjektiv und keinesfalls absolut. Sie wird vor allem durch Erfahrungswerte und neurologische Effekte beeinflusst.

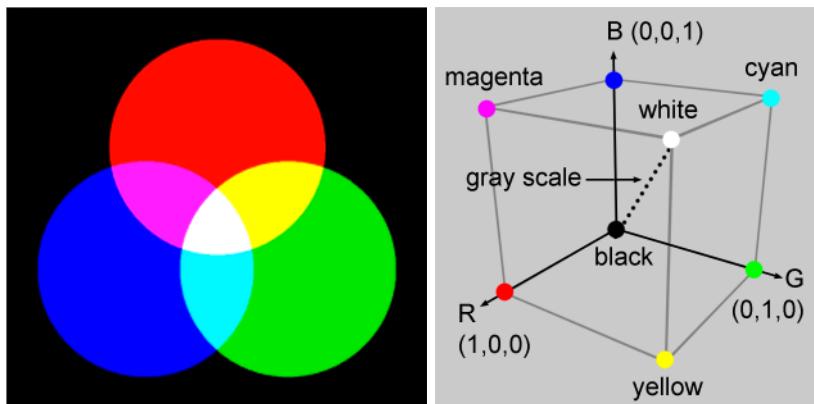
Im oberen Beispiel sieht das Feld A viel dunkler aus als das Feld B. Das Gehirn erkennt, dass das Feld B im Schatten liegt und hellt daher seine Farbe auf. Es erkennt auch das Muster des Schachbretts und versucht die Farben so zu interpretieren, dass das Muster nicht gebrochen wird.

6.3 Farbmodelle

6.3.1 CIE Farbraum

Der CIE Farbraum ist die erste Standardisierung eines Farbraums. Er umfasst alle vom Menschen wahrnehmbaren Farben. Die folgenden Farbmodelle basieren auf diesem Farbraum.

6.3.2 Hardwareorientiertes Modell - RGB



Beim RGB Farbmodell besteht jede Farbe aus einer Mischung der 3 Grundfarben **rot**, **grün** und **blau**. Das bedeutet, dass für jeden Pixel ein Vektor $\vec{v} = \begin{pmatrix} r \\ g \\ b \end{pmatrix}$ mit 3 Werten definiert wird.

Das RGB Modell ist ein additives Farbmodell. Das bedeutet, dass die Menge an Licht sich bei steigendem Wert einer Komponente des Farbvektors erhöht. Das bedeutet, dass $\vec{v}_1 = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$ schwarz entspricht und $\vec{v}_2 = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$ weiß entspricht.

Das additive Farbdenken kennen wir, wenn wir mit Lampen, die verschiedene Farben haben, auf eine weiße Wand leuchten (1. Bild). Das weiße Licht entsteht bei Anwesenheit aller Farben. Entspricht dieses Farbmodell ziemlich direkt der Ansteuerung der einzelnen Pixelkomponenten, die wir schon kennengelernt haben.

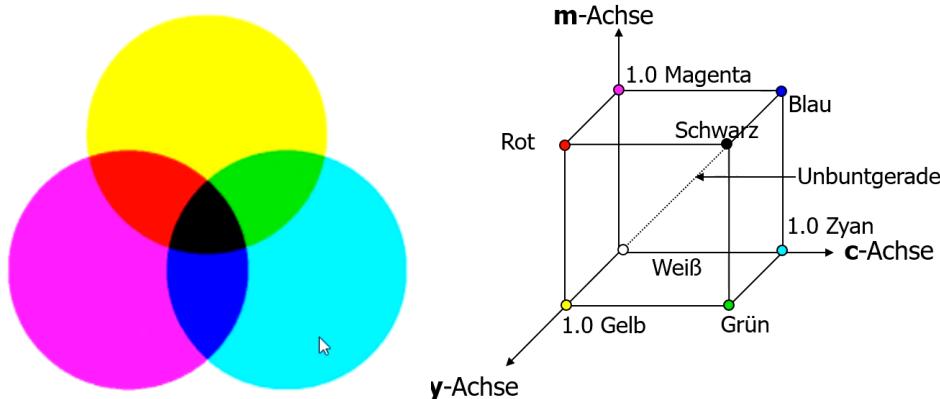
Die Gerade, die von schwarz nach weiß führt nennt man Unbuntgerade und enthält alle Grautöne. Für Grautöne gilt, dass $R=G=B$. Das Sechseck, dass aus \overline{bm} , \overline{mR} , \overline{Ry} , \overline{yG} , \overline{Gc} und \overline{cB} gebildet wird,

enthält alle maximal gesättigten Farben, d.h. Farben mit Sättigung = 100% /255. Je höher der grauanteil ist (i.e. näher an der Unbuntgerade), desto weniger gesättigt ist die Farbe.

In einer Erweiterung des RGB-Modells, dem RGBA-Modell, wird ein weiterer sogenannter Channel angefügt, der sich Alpha-Channel nennt und die Sichtbarkeit bzw. Transparent darstellt. Da wir jeden Channel mit einer Genauigkeit von 8Bit darstellen, benötigt RGBA dann 4Byte pro Pixel.



6.3.3 Hardwareorientiertes Modell - CMY



Das CMY (Cyan Magenta Yellow) Modell ist ein subtraktives Farbmodell mit den entsprechenden Grundfarben. Umso höher die Werte der Komponenten sind, desto dunkler wird die Farbe, es trifft also weniger Licht ins Auge (subtraktiv). Wir kennen dieses Prinzip beim Mischen von Wasserfarben. Der Farbwürfel des CMY-Modells entsteht aus dem Farbwürfel des RGB-Modells, indem man die Achsenbeschriftung ändert und den Würfel so dreht, dass seine Weiße Ecke im Ursprung liegt. Während das RGB der Lichtmischung entspricht und damit für Displays verwendet werden kann, kann das CMY Modell für Pigmentmischung z.B. beim Drucken verwendet werden.

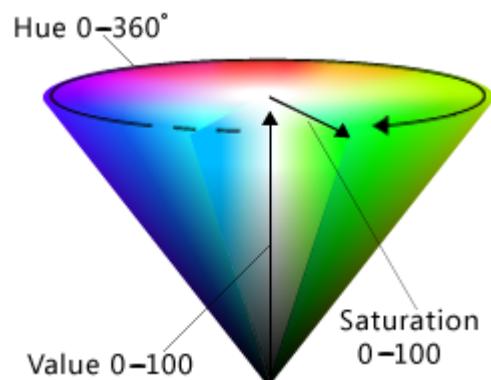
Das CMYK (CMY Black) Modell ergänzt das CMY-Modell um einen weiteren Anteil an Schwarz. Der Vorteil dieses Modells ist die bessere Darstellung von Schwatztönen und Konturen z.B. beim Drucken. Eigentlich ist der Black-Channel redundant, da $v_{CMY} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$ bereits schwarz entspricht. In der Praxis sind die Farben aber nicht perfekt rein, sodass beim Mischen kein perfektes Schwarz entsteht.

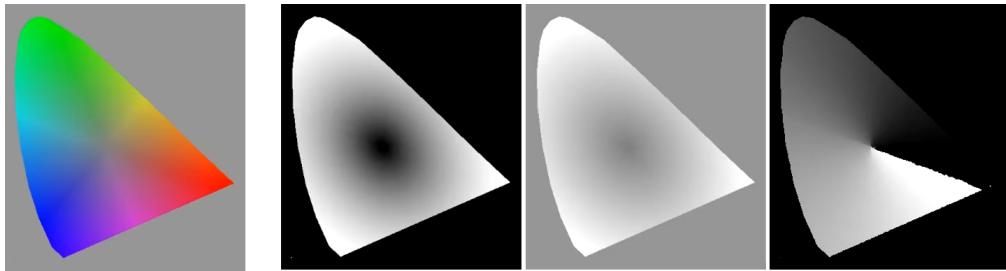
6.3.4 Wahrnehmungsorientiertes Modell - HSV

Beim HSV Modell wird eine Farbe in einem Zylinderkoordinatensystem mit den Werten **Hue**=Farbton (Winkel des Kreisausschnitts 0-360° bzw. 0% - 100%), **Saturation** = Sättigung (Radius des Kreises) und **Value** = Helligkeit/Intensität (vertikale Achse) dargestellt. Der Farbverlauf des oberen Randes des Farbkegels entspricht genau dem Sechseck der maximal gesättigten Farben des RGB-Würfels.

Im folgenden sehen wir ein Beispiel für die Zerlegung eines Farbbilds in die 3 HSV Kanäle, wobei jeder extrahierte Wert $x \in [0, 1]$ wieder als Wert eines schwarz-weiß Bildes interpretiert wird, wobei 0 schwarz entspricht und 1 weiß.

Die Bilder sind in der Reihenfolge All, Saturation, Value, Hue gezeigt:





Erklärung der Zuordnung:

Auf dem **ersten Bild** ist der graue Hintergrund schwarz, da die Sättigung von Grautönen Null ist. Außerdem ist die Außenkante des Objektes weiß, da hier die maximal gesättigten Farben liegen. Somit handelt es sich um die Saturierung.

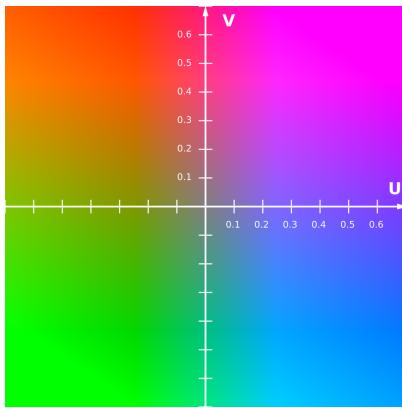
Auf dem **zweiten Bild** ist der Hintergrund identisch mit dem grauen Hintergrund vom Originalbild, denn Ein Grauton entspricht in HSV einem puren Helligkeitswert. Somit handelt es sich um Value.

Auf dem **dritten Bild** ist der Hintergrund schwarz, da Grautöne in HSV keinen Farbwert haben. Außerdem zeigt sich der Farbverlauf als lineare Steigerung der Graustufen, da die Farben genau so angeordnet sind, wie sie im HSV-Farbkegel angeordnet sind. Somit steigt der Wert des ihnen zugeordneten Winkels stetig.

Da das HSV-Modell die Farben getrennt von der Helligkeit speichert, konnte mithilfe dieses Modells früher (in der Zeit des Schwarz-Weiß-Fernsehens) einfach ohne Rechnen aus einem Farbfilm ein Schwarz-Weiß-Film extrahiert werden, der an private Haushalte ausgestrahlt werden konnte.

Außerdem können manche Algorithmen der Bildverarbeitung schneller mit solchen Bildern arbeiten, da viele Techniken hauptsächlich auf Helligkeitswerten basieren.

6.3.5 YUV Farbmodell und 4:1:1 Farbraum



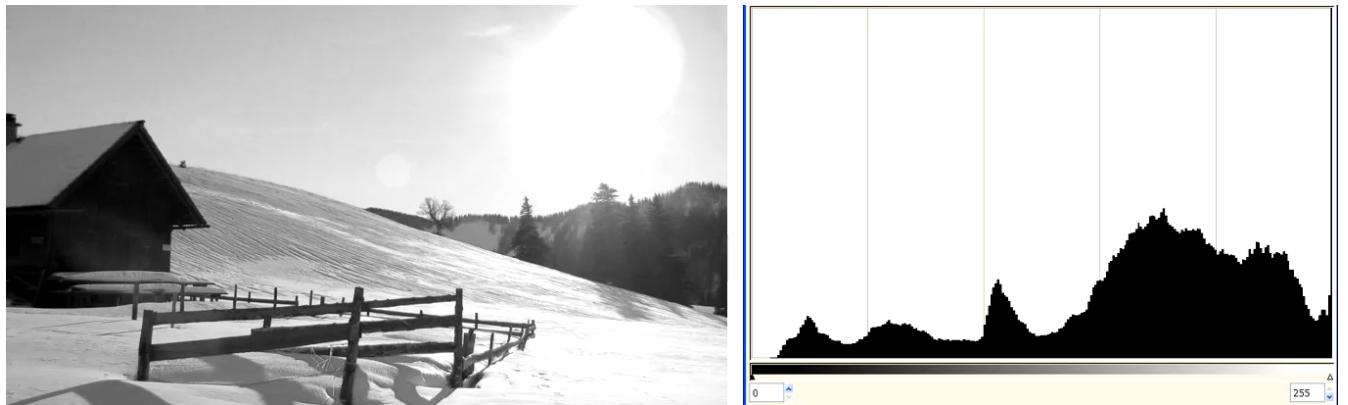
Im YUV-Farbmodell wird eine Farbe mittels eines Chrominanz-Wertes, der aus 2 sättigungslosen Farben U und V (siehe Bild) besteht, und einem Luminanzwert (Helligkeit) bestimmt.

Dieses Modell kann benutzt werden, um Bilder zu komprimieren. Denn das menschliche Auge ist sehr viel empfindlicher für Helligkeit als für Farbe. Somit kann man, wenn die Farbe und Helligkeit getrennt gespeichert sind, weniger Daten für die Farbe als für die Helligkeit benutzen, ohne dass es für den Menschen wahrnehmbar ist. Oft wird der sog. 4:1:1 Farbraum verwendet, in dem für 4 Pixel auch 4 Helligkeitswerte gespeichert werden, aber nur 1 Farbwert U und 1 Farbwert V.

7 Bildverarbeitung

7.1 Maße zur Beurteilung von Bildern

7.1.1 Histogramme



Mit Histogrammen stellen wir die Häufigkeitsverteilung von Werten von Pixeln in einem Bild dar. Da wir mit 8 Bit Wertebereich [0, 255] arbeiten, gibt es 256 Werte denen eine absolute Häufigkeit im Bereich [0, n] zugeordnet wird, wobei n die Anzahl der Pixel des Bildes ist.

Aus diesen Histogrammen kann man Informationen über das Bild extrahieren. Beispielsweise kann man im oberen Beispiel anhand des Histogramms sehen, dass der Großteil der Pixel einen hohen Wert hat, also das Bild eher hell ist. Des Weiteren können wir folgende Dinge erkennen:

- **Belichtungsfehler:**

Belichtungsfehler kann man daran erkennen, dass in einem Bild das eine Ende des Wertebereichs ungenutzt bleibt, am anderen Ende aber Häufungen auftreten (wie im oberen Beispiel)

- **Kontrast:**

Der Kontrast bezeichnet die Differenz zwischen dem minimalen und maximalen genutzten Grauwert. Ein maximaler Kontrast nutzt den vollen Wertebereich. Das bedeutet bei 8 bit Bildern, dass es jeweils mindestens ein Pixel mit den Werten 0 und 255 im Bild geben muss. Ein kleiner Kontrast entsteht, wenn alle genutzten Werte nah beieinander liegen, bis hin zum minimalen Kontrast, bei dem jedes Pixel den gleichen Wert hat.

- **Dynamik:**

Die Dynamik beschreibt wie hoch der Anteil der genutzten Werte im Bereich zwischen dem minimalen und maximalen genutzten Wert liegt. Das bedeutet, dass eine minimale Dynamik entsteht, wenn es nur Pixel mit den Werten 0 und 255 gibt. Dann ist nämlich der genutzte Wertebereich maximal, nämlich [0, 255], und die Anzahl der genutzten Werte dieses Bereichs minimal, nämlich 2. Eine maximale Dynamik entsteht dann, wenn alle Werte zwischen dem höchsten und niedrigsten genutzten Wert genutzt werden. Das obere Bild zeigt z.B. eine maximale Dynamik, da im Wertebereich [20, 255] alle Werte mindestens 1 mal vorkommen und es keine Werte außerhalb dieses Bereichs gibt, die genutzt werden.

7.1.2 Entropie

Die Entropie einer Nachricht (hier eines Bildes) ist ihr mittlerer Informationsgehalt. Es gilt: Je mehr Pixel es mit dem gleichen Wert gibt, desto geringer ist der Informationsgehalt dieser Pixel. Das bedeutet, dass die Entropie zunimmt, wenn sich die Verteilung der Grauwerte eines Bilds einer Gleichverteilung mit vollem Kontrast annähert. Auf die mathematische Berechnung wollen wir an dieser Stelle verzichten.

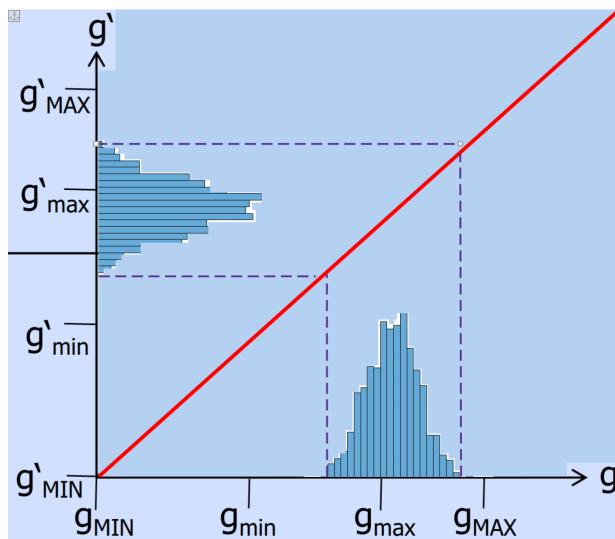
7.2 Punktoperationen & lineare Transformationen

Punktoperationen bezeichnen eine Art von Transformationen bei denen auf jedes Pixel eines Bildes eine Funktion angewendet wird, um das Pixel des Abbilds zu erzeugen, die **unabhängig von den**

anderen Pixeln ist. Das bedeutet, dass die Eingangsvariable der Wert genau eines Pixel ist und die Ausgangsgröße genau der Wert des gleichen Pixels im Abbild. Die Operation wird dann für alle Pixel des Bildes durchgeführt.

Die Punktoperationen, die wir betrachten, sind **lineare Operationen** deshalb nennt man sie auch **lineare Grauwerttransformationen**. Das bedeutet, dass die Zuordnung eines Wertes eines Pixels im Originalbild zum Pixel im Abbild durch eine lineare Funktion beschrieben wird. Diese Funktion ordnet jedem Wert des farblichen Wertebereichs $[0, 255]$ wieder einen Wert genau dieses Bereichs zu. Die Funktionsgraphen in denen wir das Aufzeichnen nennen wir auch gg-Diagramme. Im Grunde ist es eine einfache lineare Funktion. Jedoch können wir auf der x-Achse eine konkrete Verteilung (also ein Histogramm) einzeichnen, und dann die Verteilung dort auf der y-Achse einzeichnen, wo die Säulen der x-Achse auf den Funktionsgraphen treffen würden. So erkennen wir beispielsweise, dass ein Histogramm bei einer Steigung $0 < m < 1$ gestaucht wird (also der Kontrast verringert wird) und für $m > 1$ das Histogramm gestreckt wird (also der Kontrast vergrößert wird). Im unteren Beispiel ist unsere Funktion $g' = g$, die also das Bild auf sich selbst abbildet.

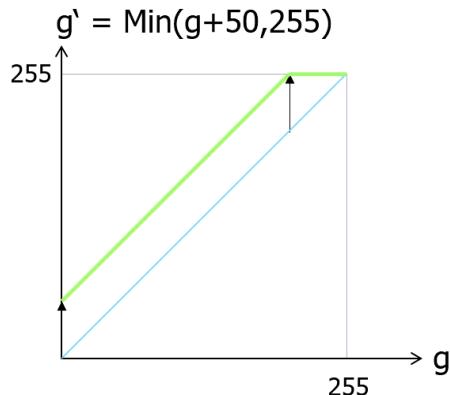
Dabei ist unbedingt zu beachten, dass der Funktionsgraph eine ganz normale lineare Funktion ist, die auch ohne das Histogramm dargestellt werden kann. Die Funktion ordnet jedem Farbwert (x-Wert) einen neuen Ergebniswert (y-Wert) zu. Das Histogramm ist nur eine Veranschaulichung der konkreten Anwendung für mehrere Punkte. Wir setzen also den Grauwert jedes Pixels einzeln in die Funktion ein, um seinen Ergebnispixel zu erhalten. Das Histogramm zeigt uns für ein konkretes Bild wie oft welcher x-Wert eingesetzt werden wird.



Bei der Wahl der Funktion für die lineare Grauwerttransformation müssen wir darauf achten, dass wir den Wertebereich der Funktion einschränken. Der Definitionsbereich und der Wertebereich müssen meist beide innerhalb unseres Bereichs von Farbwerten, also $[0, 255]$, liegen, damit das Ergebnis wieder ein gültiges Bild ist. Um dies umzusetzen können wir Mathematisch beispielsweise eine min bzw. max Funktion benutzen. Programmietechnisch gibt es natürlich noch viele andere Wege dieses recht einfache problem zu lösen.

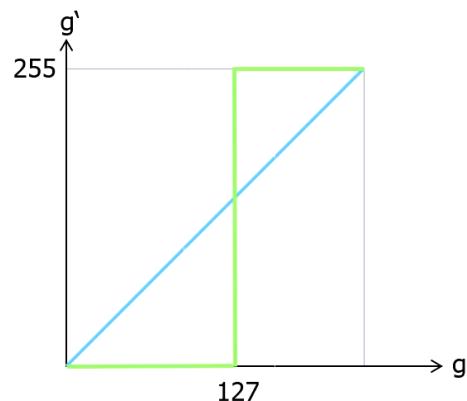
Es kann jedoch auch Anwendungsfälle geben, in denen der Definitionsbereich oder Wertebereich nicht $[0, 255]$ sein müssen. Bei den Differenzoperatoren werden wir beispielsweise sehen, dass wir die lineare Grauwerttransformation anwenden, um Ergebniswerte wieder in den gültigen Bereich zu bewegen. In diesem Fall können die Eingangswerte z.B. auch negativ sein. Im folgenden sollen einige Punktoperationen beispielhaft gezeigt werden:

7.2.1 Addition



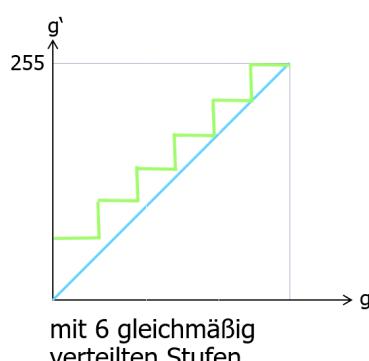
Bei dieser Addition wird auf jeden Pixelwert 50 addiert. Das interessante daran ist, dass der Wertebereich $[0, 49]$ des Ergebnisbildes auf jeden Fall ungenutzt bleibt. Außerdem werden alle Säulen des Histogramms von $[205, 255]$ auf den Wert 255 abgebildet. Dadurch entsteht, falls der Bereich $[205, 255]$ überhaupt im Originalbild genutzt wurde, eine erhöhte Häufigkeit für den Pixelwert 255 im Ergebnisbild. Ansonsten sind im Ergebnishistogramm bloß alle Werte um 50 nach rechts verschoben. Logisch gesehen beschreibt die Addition mit einem positiven Wert eine gleichmäßige Aufhellung des Bildes.

7.2.2 Binarisierung



Die Binarisierung ist eine lineare Grauwerttransformation, bei der die Mächtigkeit des Wertebereichs 2 ist $|W| = 2$. Dabei kann gewählt werden bei welchem Schwellwert der Umbruch vom einen Wert auf den anderen erfolgt (im Beispiel 127) und auf welche 2 Werte abgebildet wird (im Beispiel auf die Werte 0 und 255).

7.2.3 Äquidensitenbilder



Bei einem Äquidensitenbild wird der Wertebereich auf einige (im Beispiel 6) gleich weit voneinander entfernte Werte eingeschränkt. Das entspricht einer Binarisierung mit einem Wertebereich mit mehr als 2 Elementen. Je mehr verschiedene Farbwerte genutzt werden, desto weniger auffallend ist diese lineare Grauwerttransformation. Diese Transformation findet Anwendung beim Drucken. Denn hier stehen eventuell nicht alle Farbwerte zur Verfügung. Deshalb müssen dann einige benachbarte Farbwerte auf die selbe Druckerfarbe abgebildet werden.

7.2.4 Kontrastveränderung



Die Kontrastveränderung können wir zum Beispiel anwenden, um den Kontrast zu erhöhen, wenn ein Bild keinen vollen Kontrast hat, es also links oder rechts im Histogramm gibt, die nicht genutzt werden. Um das Bild auf einen vollen Kontrast zu transformieren muss die in schwarz eingezeichnete Grauwerttransformation durchgeführt werden. Es wird eine Gerade von $P_1 = (g_{min}, g'_{MIN})$ bis $P_2 = (g_{max}, g'_{MAX})$ gezeichnet werden. Dabei sind g_{min} und g_{max} die minimalen und maximalen genutzten Grauwerte im Originalwert und g'_{MIN} und g'_{MAX} die Grenzen des Wertebereichs im Ergebnisbild. Der Bereich von g'_{MIN} bis g'_{MAX} bestimmt also welche Spanne von Werten das Ergebnisbild einnimmt und ist somit bei einer Erhöhung auf vollen Kontrast $g'_{MIN} = 0$ und $g'_{MAX} = 255$.

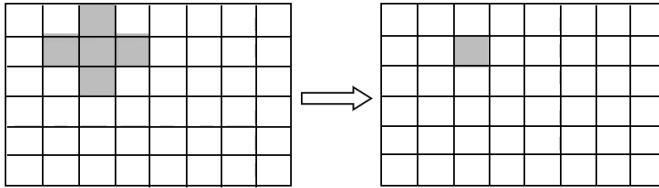
Man sieht, dass auf diese Weise Lücken im Histogramm entstehen, die Dynamik also abnimmt, die Entropie bleibt gleich, da es immernoch genau gleich viele unterschiedliche Pixelwerte mit der gleichen Verteilung gibt, nur haben sich die konkreten Werte geändert.

Beachte unbedingt die Einführung in die Punktoperationen, um den Graphen richtig zu verstehen.

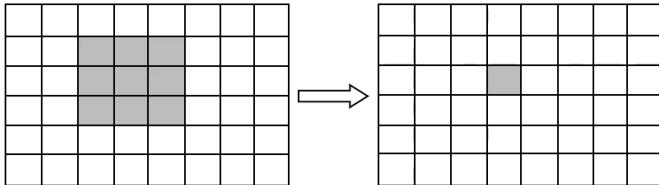
7.3 Lokale Bildoperationen - Faltungsoperatoren

7.3.1 Faltungsmatrizen /-operatoren und deren Berechnung

N4-Nachbarschaft



N8-Nachbarschaft



Lokale Bildoperationen sind Transformationen, bei denen für die Berechnung eines Pixels mehrere, nämlich sinnvoller Weise benachbarte, Pixel betrachtet werden. Bei den Nachbarschaftsbeziehung unterscheiden wir zwischen N4- und N8-Nachbarschaft, in welchen einem Pixel, das in der Mitte liegt, jeweils 4 oder 8 Nachbarn zugeordnet werden.

Haben wir uns für eine Nachbarschaftsform entschieden, dann müssen wir für jedes Pixel der "Schablone", auch Faltungsmatrix genannt, eine Gewichtung definieren. Bei der Berechnung eines Pixelwertes wird jeder Wert der Schablone mit dem Wert des jeweiligen Pixels multipliziert und dann alle so entstandenen Werte addiert. Zur Berechnung aller Pixel des Ergebnisbildes wird nun die Berechnung mit der Faltungsmatrix für jeden Pixel durchgeführt, also so, dass jedes Pixel des Originalbildes einmal in der Mitte der Faltungsmatrix liegt. Eine Ausnahme ist der äußere Rand des Originalbildes, für den es keine äußeren Nachbarn gibt. Deshalb wird der Rand weggelassen und das Ergebnisbild in beide Richtungen 2 Pixel kleiner.

In vielen Fällen ist der Wertebereich nicht mehr identisch mit dem Definitionsbereich $D \neq W$. In diesen Fällen müssen die Werte durch eine Lineare Transformation wieder in den gültigen Bereich versetzt werden. Somit können wir lokale Operationen für Schwarz weiß Bilder anwenden. Wenn wir ein Farbbild haben führen wir die Berechnungen einfach für jeden Farbchannel durch. Im folgenden sehen wir die Berechnungsformel für eine N8-Faltungsmatrix:

$$e(i, j) = \sum_{l=0}^2 \sum_{k=0}^2 [g(i - 1 + k, j - 1 + l) \cdot f(k, l)]$$

Hierbei ist $g(a, b)$ der Wert des Pixels der Faltungsmatrix in der Zeil a und der Spalte b . $f(a, b)$ ist der Wert des Pixels im Originalbild in der Zeile a und der Spalte b . $e(i, j)$ ist der Wert des Pixels in der Zeile i und der Spalte j .

Die Summenzeichen sorgen dafür, dass wie mit einer verschachtelten for-Schleife über alle 9 Pixel der Faltungsmatrix iteriert wird.

Nun werden wir einige Beispiele für Faltungsmatrizen sehen.

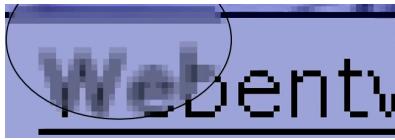
7.3.2 Identität

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

Die Identität ist die Faltungsmatrix bei der nur der Wert des inneren Pixels beachtet wird. Dadurch bildet sie das Originalbild auf sich selbst ab und ist identisch mit der Punktoperation $g'(i, j) = g(i, j)$

7.3.3 Glättungsfilter / Box-Filter / Mittelwertoperator

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$



Bei diesem einfachen Glättungsfilter wird jedes Pixel der Faltungsmatrix gleich stark gewichtet. Dabei entsteht im Bild eine Unschärfe oder Glättung, weil die benachbarten Pixel verschwimmen. Allerdings entstehen um Kanten herum sogenannte **ringing artifacts**. Das sind Pixel, die um die Kanten herum liegen und den gleichen Farbwert wie die Kante haben. Somit wird es schwer die Schrift zu lesen.

7.3.4 Gauß-Filter

$$A = \begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}$$



Der Gauß-Filter ist eine Glättungsfunktion, bei der die Gewichtung der inneren Pixel höher ist. Dadurch wird das Bild zwar geglättet, aber markante Punkte bleiben erkennbar. Im Gegensatz zum normalen Glättungsfilter können wir hier die Schrift des damit erzeugten Bildes noch besser lesbar und die Linien sind besser erkennbar.

7.3.5 Differenzoperator & Verständnis für die Ableitung

$$A = \begin{pmatrix} 0 & 0 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & 0 \end{pmatrix}, B = \begin{pmatrix} 0 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}, C = \begin{pmatrix} 0 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}, D = \begin{pmatrix} 0 & 0 & 0 \\ 0 & -2 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

Der Differenzoperator hat seinen Namen davon, dass er differenziert also die Ableitung bildet. Dazu muss man sich nebeneinanderliegende Pixel zunächst als Funktion vorstellen, wobei jeder Pixelwert ein Funktionswert ist. Die Ableitung kann berechnet werden, indem für jeden Pixel die Steigung berechnet wird. Da wir es mit einem diskreten Definitionsbereich zu tun haben ist die Berechnung der Steigung in einem Punkt sehr einfach. Wir subtrahieren die Differenz der Funktionswerte (Pixelwerte) von benachbarten x-Werten (Pixeln) und teilen durch deren Differenz der x-Werte. Da die Pixel benachbart sind ist ihre Differenz auf der x-Achse 1, wodurch die Division entfällt. Mit dieser Vorstellung sollte klar sein, dass A und B das identische Ergebnis, nämlich die Zeilenweise Ableitung bilden. C bildet die Spaltenweise Ableitung. D bildet die Ableitung in Zeilen- und Spaltenrichtung, was einer hinternanderausführung von A und C entspricht. Da die Anwendung der Transformation auf eine Summe herausläuft können wir die Summe auch in der Matrix D zusammenfassen.

Die Differenzoperatoren haben die Eigenschaft, dass Ergebnispixel stark positive Werte annehmen, wenn im Bild eine Kante von dunkel nach hell sichtbar war, stark negative Werte annehmen, wenn eine Kante von hell nach dunkel war und 0 sind, wenn benachbarte Pixel den gleichen Farbwert hatten. Somit wird klar, dass starke Kontraste hervorgehoben werden und schwache Kontraste weiter gedämpft werden.

Der Wertebereich von Differenzoperatoren ist grundlegend ungültig. Für A ist er z.B. [-255, 255]. Um den Wertebereich wieder in den gültigen Bereich zu transformieren wird eine lineare Transformation durchgeführt, die zunächst den Wertebereich auf die selbe Mächtigkeit bringt, d.h. $|W| = 256$ und dann auf [0, 255] verschiebt. Im Falle von A wäre das dann die lineare Transformationsfunktion $f(x) = 0.5x + 128$

7.3.6 Prewitt-Operator

$$A = \begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix}, B = \begin{pmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}$$

Der Prewitt-Operator ist im Grunde nur eine Abwandlung des Differenzoperators. Hier wird sozusagen der Sichtbereich erweitert.

7.3.7 Sobel-Operator

$$A = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}, B = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

Der Sobel-Operator ist wiederum eine Variation des Prewitt-Operators.

7.3.8 Laplace-Operator

$$A = \begin{pmatrix} 0 & 1 & 1 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

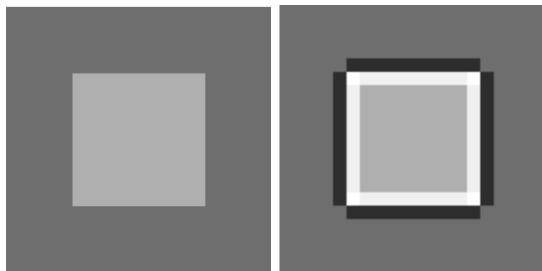
Der Laplace-Operator bildet die 2. Ableitung in Zeilen- und in Spaltenrichtung. Wir sehen also, dass der Laplace-Operator entsteht, wenn wir von den Differenzoperatoren 2 spaltenweise Operatoren und 2 zeilenweise Operatoren addieren. Wir könnten das gleiche Ergebnis auch mit einer nicht-punktsymmetrischen Matrix erzielen, indem wir z.B. die Matrix D 2 mal anwenden, also alle Werte elementweise mit 2 multiplizieren.

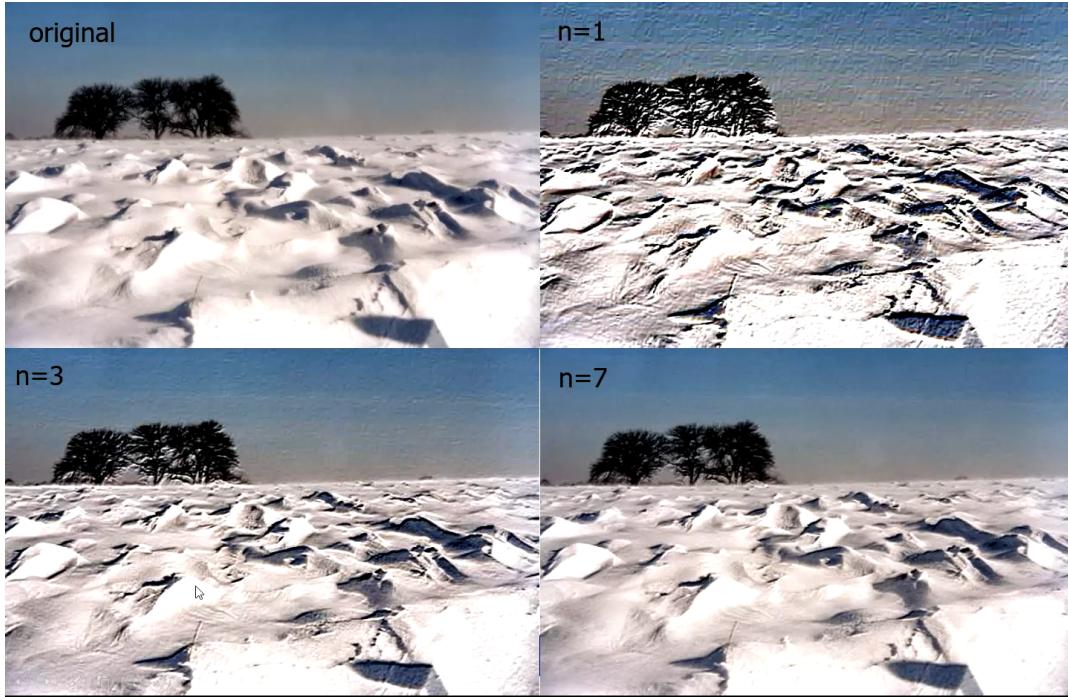
7.3.9 Kantenhervorhebung bzw. Schärfungsfilter

Die Kantenhervorhebung nimmt einen Kantendetektor (meist Laplace) und gibt dem inneren Pixel eine beliebige Wertigkeit n . Somit lässt sich das aktuelle Pixel stärker gewichten. Auf diese Weise kann ein Filterergebnis mit dem Originalbild vermischt werden. Je größer n gewählt wird, desto mehr fällt das Originalbild ins Gewicht.

Beispiel für den kantenhervorhebenden Laplace-Filter:

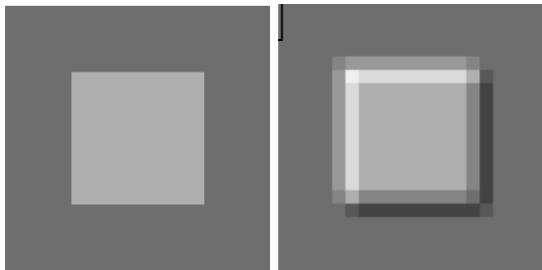
$$A = \begin{pmatrix} 0 & -1 & -1 \\ -1 & n+4 & -1 \\ 0 & -1 & 0 \end{pmatrix}$$





7.3.10 Relief-Filter

$$A = \begin{pmatrix} -2 & -1 & 0 \\ -1 & 1 & 1 \\ 0 & 1 & 2 \end{pmatrix}$$



7.4 Lokale Bildoperatoren - Rangfolgeoperatoren

7.4.1 Rangfolgeoperatoren und deren Berechnung

Rangfolgeoperatoren sind grundlegend ähnlich wie Faltungsoperatoren. Daher empfiehlt es sich das Kapitel [Faltungsoperatoren](#) als Einstieg zu lesen. Bei Rangfolgeoperatoren wird nämlich ebenfalls zunächst eine Nachbarschaftsbeziehung gewählt und die entstandene Matrix pixelweise über das Bild bewegt, um den Wert für das mittlere Pixel zu berechnen. Die Berechnung des Ergebnispixels erfolgt aber nicht über eine kumulierte Multiplikation, sondern über eine Auswahl genau eines Wertes aus den eingehenden Pixelwerten:

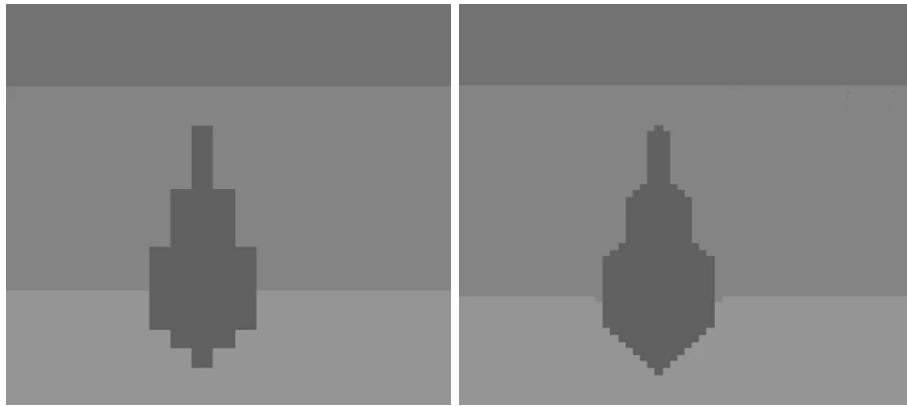
Wir betrachten das Beispiel einer N8-Nachbarschaft, also einer 3x3-Matrix.

1. Bei der Berechnung eines Pixel werden zunächst die Werte der Eingangspixel der Größe nach geordnet.
2. Dann wird einer der Werte nach einer gewissen Vorschrift ausgewählt. Konkrete Vorschriften folgen im nächsten Teilkapitel.

7.4.2 Arten von Rangfolgeoperatoren

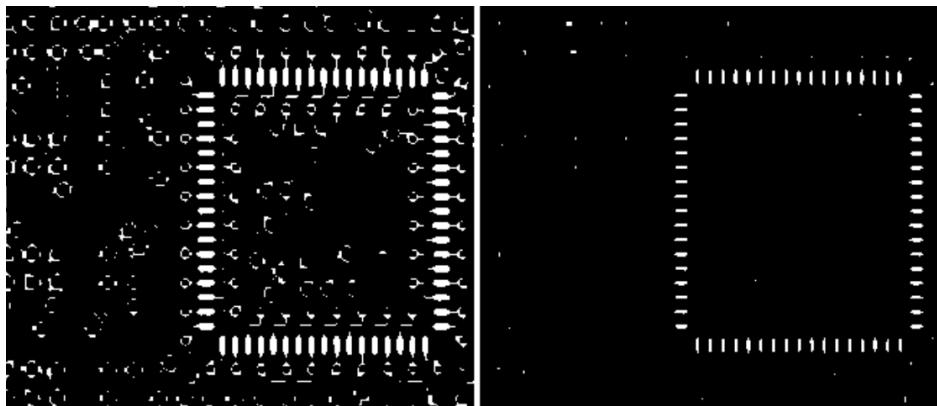
Wir wollen hier 3 verschiedene Rangfolgeoperatoren kennenlernen und beispielhaft ihre Auswirkungen betrachten:

Medianoperator



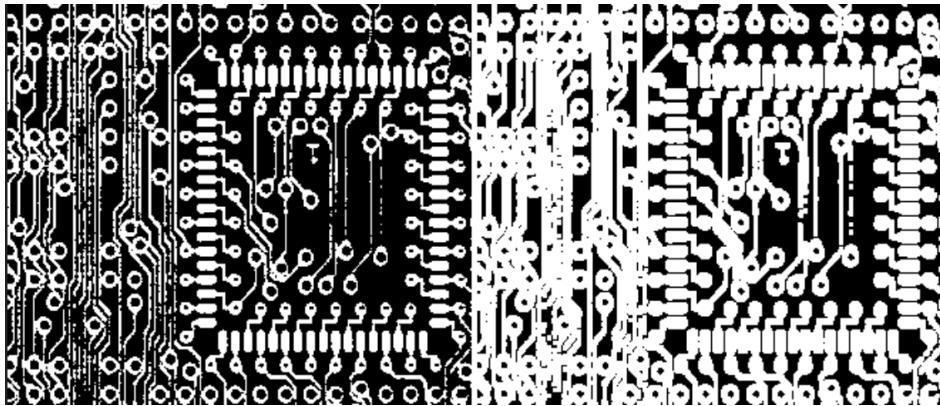
Beim Medianoperator wird der Median unserer Eingangswerte, also der mittlere Wert der Rangfolge, übernommen. Das Ergebnis ist, dass Ecken abgerundet werden, aber im Gegensatz zum [Mittelwertoperator](#) keine verschwommenen Kanten entstehen, da keine Werte gemischt werden.

Erosion



Bei der Erosion wird der dunkelste, also geringste Wert der Rangfolge übernommen. Das bedeutet, dass helle Teile an ihren Kanten, also dort wo sie auf dunklere Teile treffen, kleiner werden und dunkle Teile größer. Falls ein helles Objekt kleiner als die Rangfolgenmatrix ist verschwindet es komplett, da für jeden Punkt mindestens ein umliegender Punkt heller als das Objekt ist.

Dilatation



Hier wird der hellste, also höchste Wert übernommen, dementsprechend das logische Gegenstück zur Erosion.

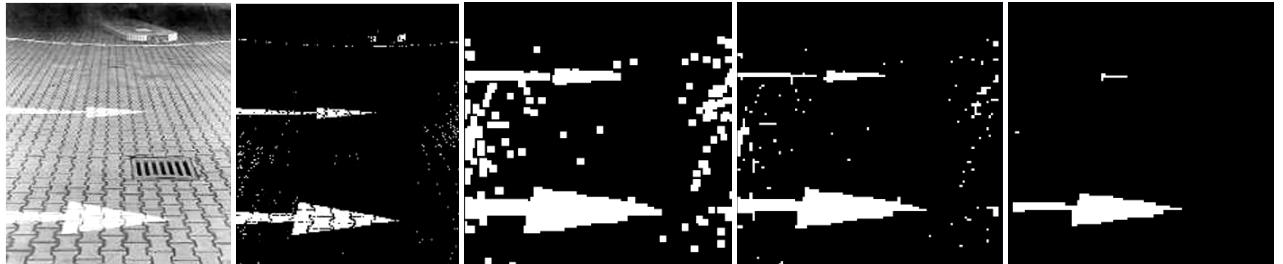
7.4.3 Anwendung und Kombination von Rangfolgeoperatoren

Die Anwendung von Rangfolgeoperatoren kann gut benutzt werden, um die Komplexität eines Bildes so stark zu vermindern, dass nur noch die relevanten Teile vorhanden sind. Solche Bilder können dann als Input für KIs dienen, welche die Bilder dann auswerten können. Wir benennen die folgenden Kombinationen:

- **Opening:** erst Erosion dann Dilatation
- **Closing:** erst Dilatation dann Erosion

Wir wollen an einem Beispiel nun zeigen, wie wir ein Bild verarbeiten können. Dazu wenden wir folgende Schritte an:

- Binarisierung mit Schwellwert 225
- Closing
 - Dilatation
 - Erosion
- Opening
 - Erosion
 - Dilatation



Damit Objekte gleich groß bleiben müssen genau so viele Erosionen wie Dilatationen durchgeführt werden. Dies gilt unter der Voraussetzung, dass die Objekte vor jeder Iteration noch mindestens so groß wie die Rangfolgenmatrix sind, damit sie nicht vollständig verschwinden.

7.5 Algorithmen

7.5.1 Canny-Edge-Detector

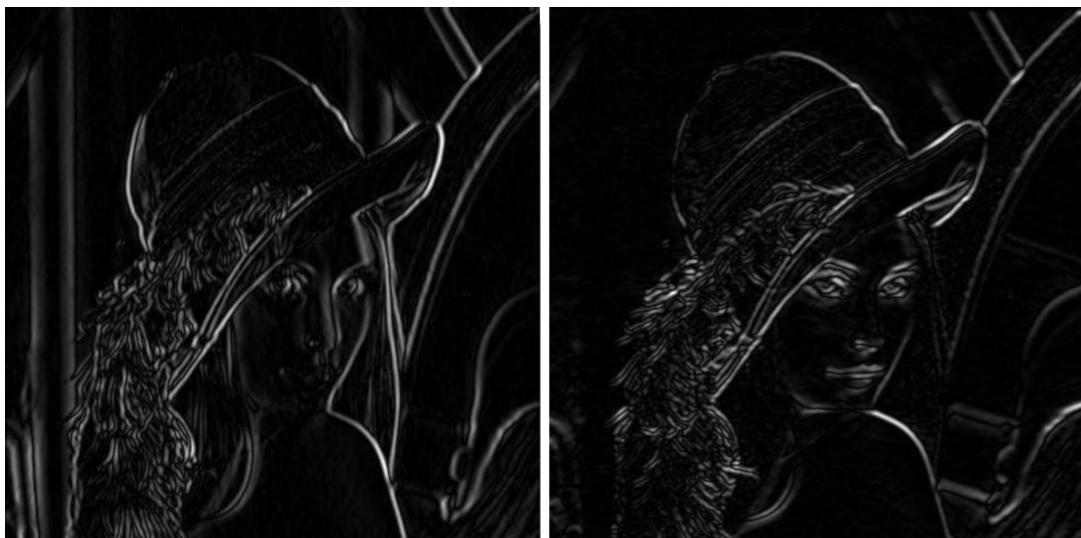
Der Canny-Edge-Detector ist ein Algorithmus zur Kantenerkennung, der aus mehreren Faltungsoperationen besteht und dazu führen soll, dass das Ergebnisbild im Idealfall nur noch die Kanten des Eingangsbild enthält. Den Algorithmus kann man in 5 Schritte unterteilen, die wir hier einmal vorgestellt und erklärt werden sollen:

(i) Konvertierung zum Graubild



Dieser Schritt ist notwendig, da der Algorithmus nur für die Arbeit mit einem Farbkanal definiert ist.

(ii) Berechnung des Gradientenfeldes



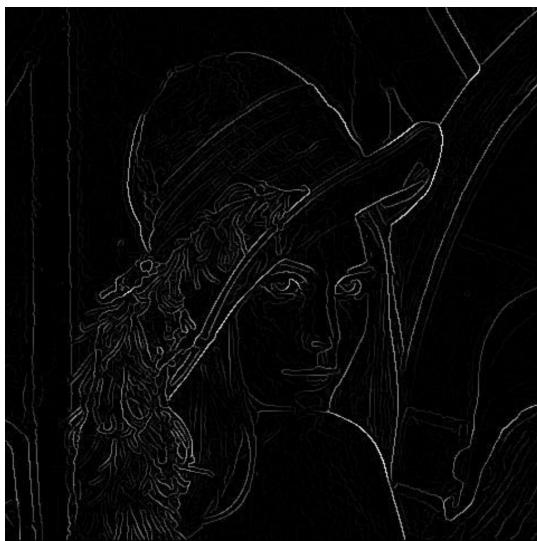
Mit Gradientenfeld ist die Ableitung, also die Steigung (also somit das Gefälle -<Gradient>) gemeint (siehe Differenzierungsoperator). Wir berechnen diese Gradienten sowohl horizontal als auch vertikal und benutzen dafür konkret den Sobel-Operator

(iii) Bestimmung des Betrags der Gradientenvektoren



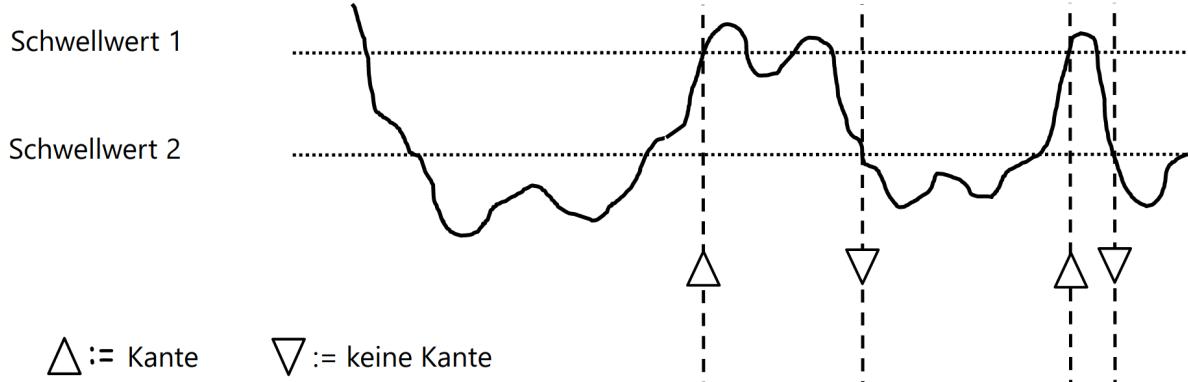
Das Ziel dieses Schrittes ist es die vertikale und die horizontale Information zusammenzuftigen. Dazu wird aus den beiden Werten für jedes Pixel ein 2-dimensionaler Vektor gebildet. Dann wird der Betrag (also die Länge) dieses Vektors berechnet. Das Ergebnis ist der neue Wert des Pixels.

(iv) Ausdünnen der Kanten - Non Maximum Suppression



Hiermit soll sichergestellt werden, dass jede Kante nur 1 pixel breit ist. Dazu wird eine 3×3 -Matrix über das Bild verschoben, und jeweils alle Nicht-Maxima, d.h. alle 8 Werte außer dem größten Wert, auf 0 gesetzt.

(v) Hysterese



Die Hysterese ist eine Art von Binarisierung. Das heißt, dass wir mittels Schwellwerten einen Wert entweder behalten wollen (im Ergebnisbild weiß) oder entfernen wollen (im Ergebnisbild schwarz = Hintergrund). Im Gegensatz zur normalen Binarisierung mit einem einzigen Schwellwert werden bei der Hysterese 2 Schwellwerte $T_1 < T_2$ verwendet.

Im ersten Schritt wird ein Pixel, wenn sein Wert größer als T_2 ist, als Bestandteil einer starken Kante eingestuft. Wenn ein Wert eines Pixels größer als T_1 aber kleiner als T_2 ist, dann wird der Pixel als Bestandteil einer schwachen Kante eingestuft.

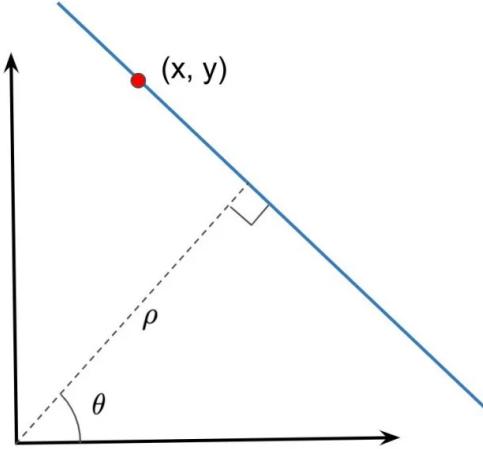
Im zweiten Schritt werden zunächst alle starken Kanten übernommen. Schwache Kanten werden nur dann übernommen, wenn sie starke Kanten berühren.

Das Ziel der Hysterese ist es nicht relevante schwache Kanten zu eliminieren. Im folgenden Ergebnisbild kann man sehen, dass z.B. die schwachen Kanten in der rechten unteren Ecke nicht übernommen wurden:



7.5.2 Hough-Transformation

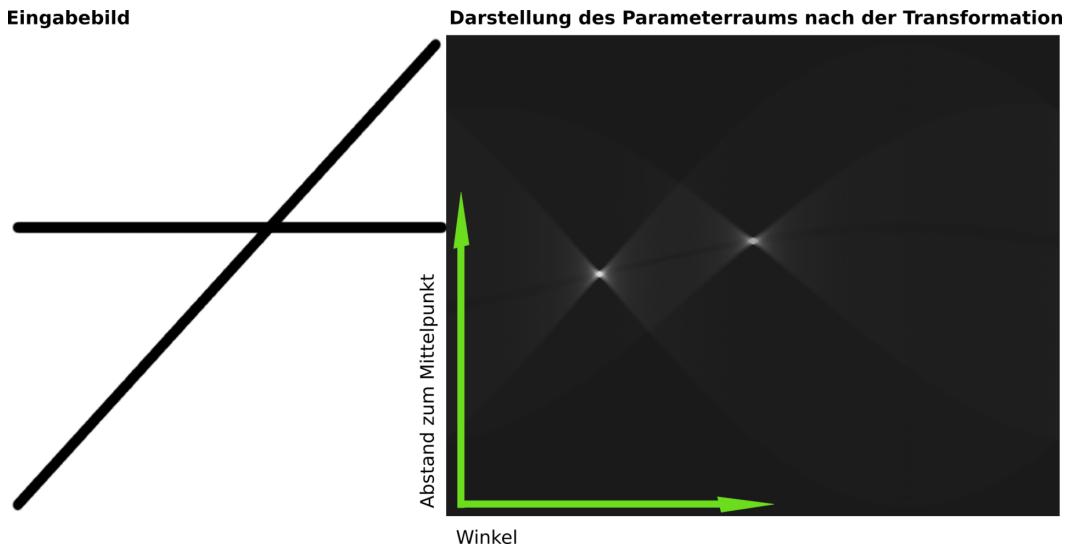
Die Hough-Transformation ist ein Verfahren zur Erkennung von Geraden, Kreisen oder anderen parametrisierbaren geometrischen Figuren in Bildern. Dabei ist schon bekannt welche Figur man erkennen möchte, man will aber die Position und Größe genau lokalisieren können. Wir wollen uns dies anhand von der Erkennung von Geraden ansehen.



Zunächst benötigen wir ein Eingangsbild, das die von uns zu erfassende Figur - die Gerade - möglichst eindeutig enthält. Um solch ein Bild aus einem realen Bild zu extrahieren können andere Algorithmen, wie z.B. der Canny-Edge-Detector verwendet werden.

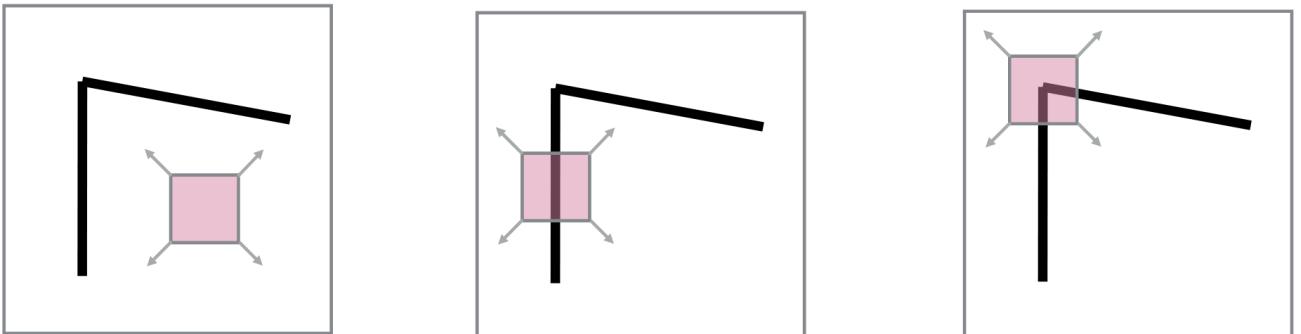
Dann müssen wir festlegen mit welchen Parametern wir die Gerade darstellen wollen. Naheliegend wäre die Steigung und der y-Achsenabschnitt (klassische Darstellung linearer Funktionen). Doch hier entsteht das Problem, dass senkrechte Geraden nicht dargestellt werden können, da sie jedem x-Wert mehrere y-Werte zuordnen würde, also sozusagen eine "unendliche Steigung" hätte. Stattdessen können wir die Gerade auch mittels einem Winkel θ und einer Länge p darstellen. Wir zeichnen eine Orthogonale zur Gerade, die durch den Ursprung verläuft. θ ist dann der Winkel der Orthogonalen zur x-Achse und p die Länge dieser Orthogonalen. Der entstandene 2-dimensionale Vektorraum (im Grunde ein einfaches 2-dimensionales Koordinatensystem) wird auch Hough-Raum genannt. Jeder konkrete Punkt im Hough-Raum stellt also eine Gerade im Eingabebild dar.

Nun iterieren wir über alle relevanten Pixel des Eingangsbildes (die ja alle zu Kanten gehören, da das Bild entsprechend vorverarbeitet wurde) und finden errechnen für jeden Pixel alle möglichen Geraden, die diesen Punkt laufen. Mathematisch gesehen sind das natürlich unendlich viele Geraden, aber da unser Eingabebild nur begrenzt viele Pixel, also eine begrenzte Genauigkeit besitzt ist auch die Anzahl der verschiedenen Geraden im Bild endlich. Für jede dieser Geraden wird ein Wert, der dem entsprechende Punkt im Hough-Raum zugeordnet wird, inkrementiert. Es wird also sozusagen für diesen Punkt gewertet. Wir können die Werte, welche den Punkten im Hough-Raum zugeordnet werden dan mit Helligkeitswerten darstellen. Wenn wir das für alle Punkte durchgeführt haben, haben die Punkte im Hough-Raum, zu denen es am meisten Pixel im Eingangsbild gibt, die zu dieser Geraden gehören könnten, den höchsten Wert. Wenn man nun die Hochpunkte der Werte im Hough-Raum findet, so erhält man also die genauen Gleichungen für die Geraden im Bild und hat somit die Geraden identifiziert.



7.5.3 Harris-Corner-Edge-Detection

Harris-Corner-Edge-Detection ist ein Verfahren zum Erkennen von Interest-Points. Interest-Points, oder auch Feature-Points genannt, sind Punkte, die man in verschiedenen Bildern der gleichen Szene oder des gleichen Objekts gut wiedererkennen kann. Die Erkennung von Interest-Points ermöglicht es z.B. automatisierte Objektverfolgung u.v.m. zu implementieren. Interest-Points sind also besonders signifikante Punkte, die möglichst aus verschiedenen Perspektiven trotzdem ähnlich aussehen.



Der Harris-Corner-Edge-Detector versucht signifikante Punkte zu finden, indem er misst wie verschieden ein Bereich von seinem Umfeld ist. Dabei wird gemessen wie groß der Unterschied der Pixelwerte ist, wenn man eine lokale-Nachbarschaftsmatrix um jeweils ein Pixel in jede Richtung verschiebt. Man erkennt im Beispiel:

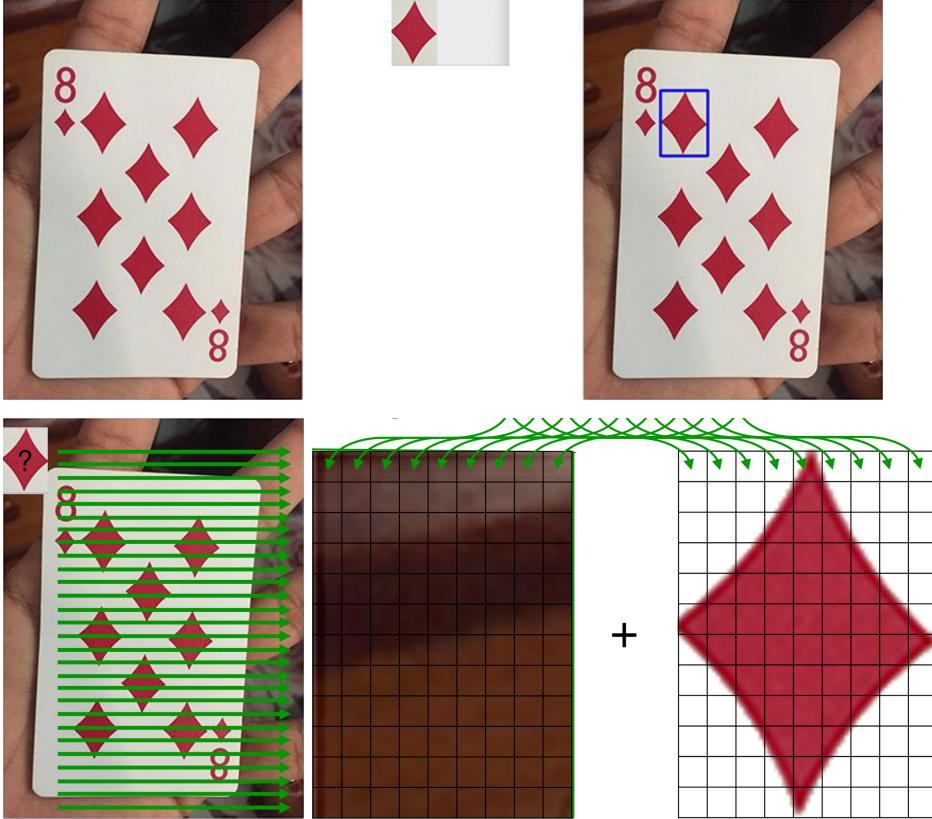
- **Im erstem Bild** verändern sich die Pixelwerte überhaupt nicht
- **Im zweiten Bild** verändern sich die Pixelwerte bei der vertikalen Verschiebung nicht
- **Im dritten Bild** verändern sich die Pixelwerte in jedem Fall → Interest-Point gefunden

7.5.4 SIFT - Scale-invariant Feature Transform

SIFT ist ein Feature-Point-Detektionsalgorithmus, der eine lokale Umgebung auf eine Menge von Vektoren, sog. feature-vectors, abbildet. Das besondere der entstandenen feature-vectors ist, dass sie so berechnet sind, dass sie weitestgehend unabhängig von Translation, Skalierung und Rotation des Objektes sind, das sie beschreiben. Dadurch ergeben sich für das gleiche Objekt aus einer anderen Perspektive die gleichen feature-vectors. Sucht man nur ein bestimmtes bekanntes Objekt in einem Bild, dann kann man es anhand seiner Menge an feature-vectors ermitteln. Auf die Details der Berechnung soll hier nicht näher eingegangen werden.

7.5.5 Template Matching

Wir haben bereits die Hough-Transformation kennengelernt, um Geometrische Formen in einem Bild zu erkennen, wie etwa Linien oder Kreise. Wenn wir aber komplexere Objekte erkennen möchten, die sich nicht einfach durch mathematische Zusammenhänge beschreiben lassen, müssen wir Template Matching betreiben. Der Input, den ein Template-Matching-Algorithmus verarbeitet sind ein Eingabebild, in dem gesuchte werden soll und ein Template, das angibt nach was gesucht werden soll. Der gewünschte Output ist dann die Position des gesuchten Objektes im Bild.



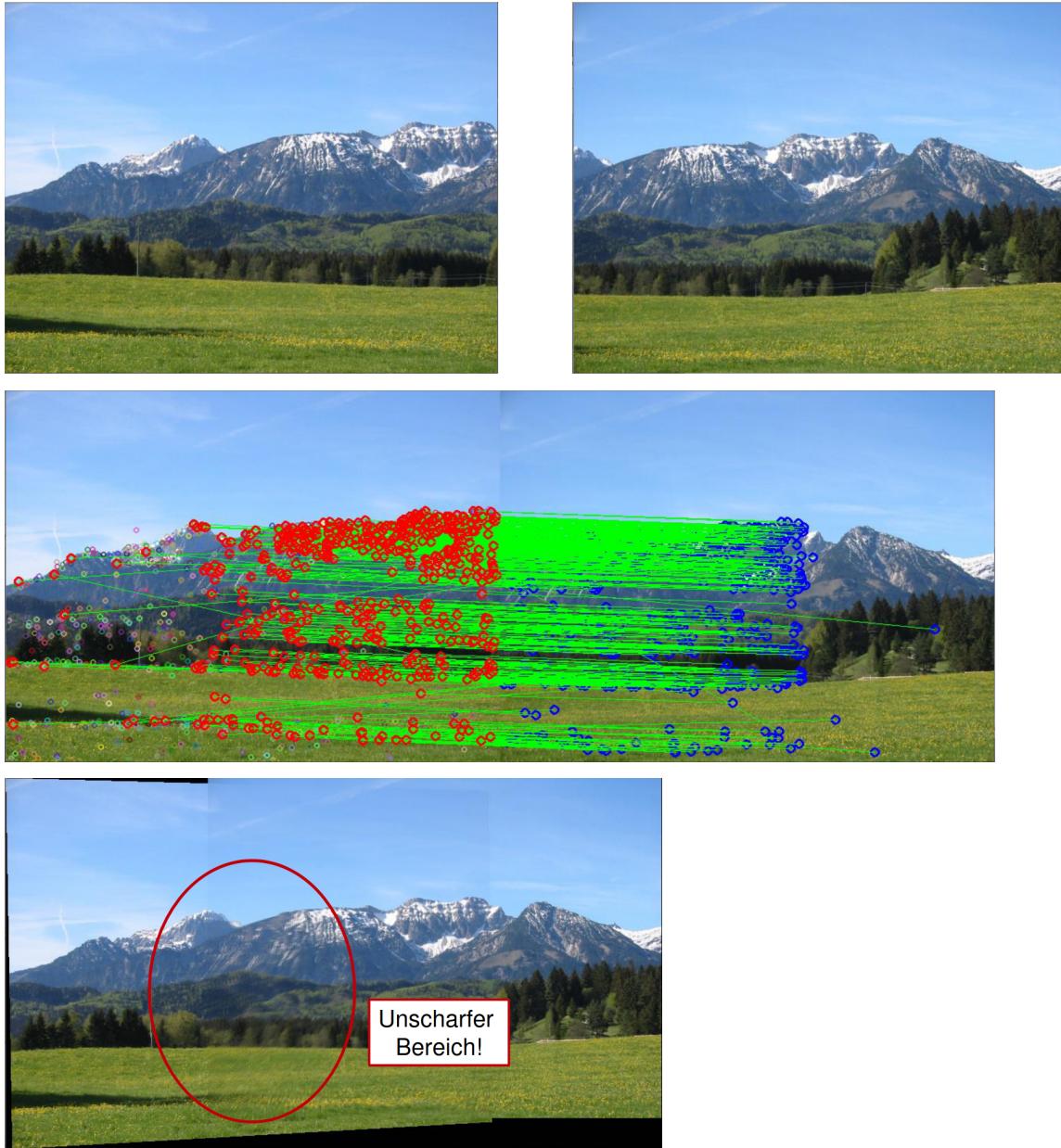
Um das zu bewerkstelligen, kann für jede Möglichkeit das Template auf dem Eingabebild zu platzieren berechnet werden, wie hoch die Abweichung ist. Wenn der Fehler unter einem gewissen Schwellwert liegt, kann man festlegen, dass ein Match erkannt werden soll. Für die Berechnung des Fehlers kann z.B. der Betrag der Summe der pixelweisen Differenzen F_1 herangezogen werden, oder auch die Summe der Quadrate der pixelweisen Differenzen F_2 :

$$F_1 = \sum_{k=0}^n |P_0(x, y) - P_1(x, y)|$$

$$F_2 = \sum_{k=0}^n (P_0(x, y) - P_1(x, y))^2$$

Die Schwierigkeiten beim Template Matching sind Abweichungen der Größe, Rotation, veränderte Beleuchtung und anderer Hintergrund des gesuchten Bereichs im Eingabebild. Es entstehen also insgesamt sehr viele Möglichkeiten, wie das Template im Bild versteckt werden kann. Im Gegensatz zum Brute-Force-Ansatz, bei dem wir alle Möglichkeiten durchprobieren würden, also etwa mit verschiedenen Drehwinkeln und Größen des Templates, gibt es auch noch den Ansatz mit Deep-Learning-Algorithmen. Bei diesem Ansatz berechnet der Algorithmus Feature-Vektoren, die im besten Fall weitestgehend invariant im Bezug auf verschiedene Transformationen, Clutter im Bild und veränderte Lichtverhältnisse sind.

7.5.6 Image Stitching



Beim Image Stitching geht es darum aus mehreren Teilbildern ein gemeinsames Bild zu erzeugen. Ein gutes Beispiel ist die Panorama-Funktion eines Handys, bei der man mehrere Bilder aufnimmt, die dann zusammengefügt werden. Die Grundlegende Herangehensweise ist die folgende:

1. Identifizieren von Feature Points:

Zunächst müssen Feature Points, also markante Punkte im Bild, erkannt werden. Hierfür kann z.B. der SIFT-Algorithmus verwendet werden

2. Finden von Feature-Pairs:

Dann müssen Feature-Pairs, also Paare aus Feature-Points der beiden Bildern, die zu den selben Objekten gehören, gefunden werden

3. Berechnung der Homografie:

Basierend auf den Vektoren, die die Feature-Pairs verbinden kann eine Homografie berechnet werden. Die Homografie ist das Verhältnis der beiden Kamerapositionen der beiden Bilder.

4. Zusammenfügen der Bilder:

Die Bilder werden an geeigneter Stelle übereinandergelegt

5. **Morphing:**

Wenn wir im überlappenden Bereich einfach die Mittelwerte der Pixelwerte bilden führt dies zu einer Unschärfe. Das liegt daran, dass die Bilder auf verschiedenen Winkeln geschossen sind und die Bereiche also nicht komplett identisch aussehen. Wir haben also 2 leicht verschiedene Bilder und wir wollen das Bild errechnen, dass sich im Übergang von einem Bild zum anderen befindet. Den Prozess des findens dieses Übergangsbildes nennt man Morphing.